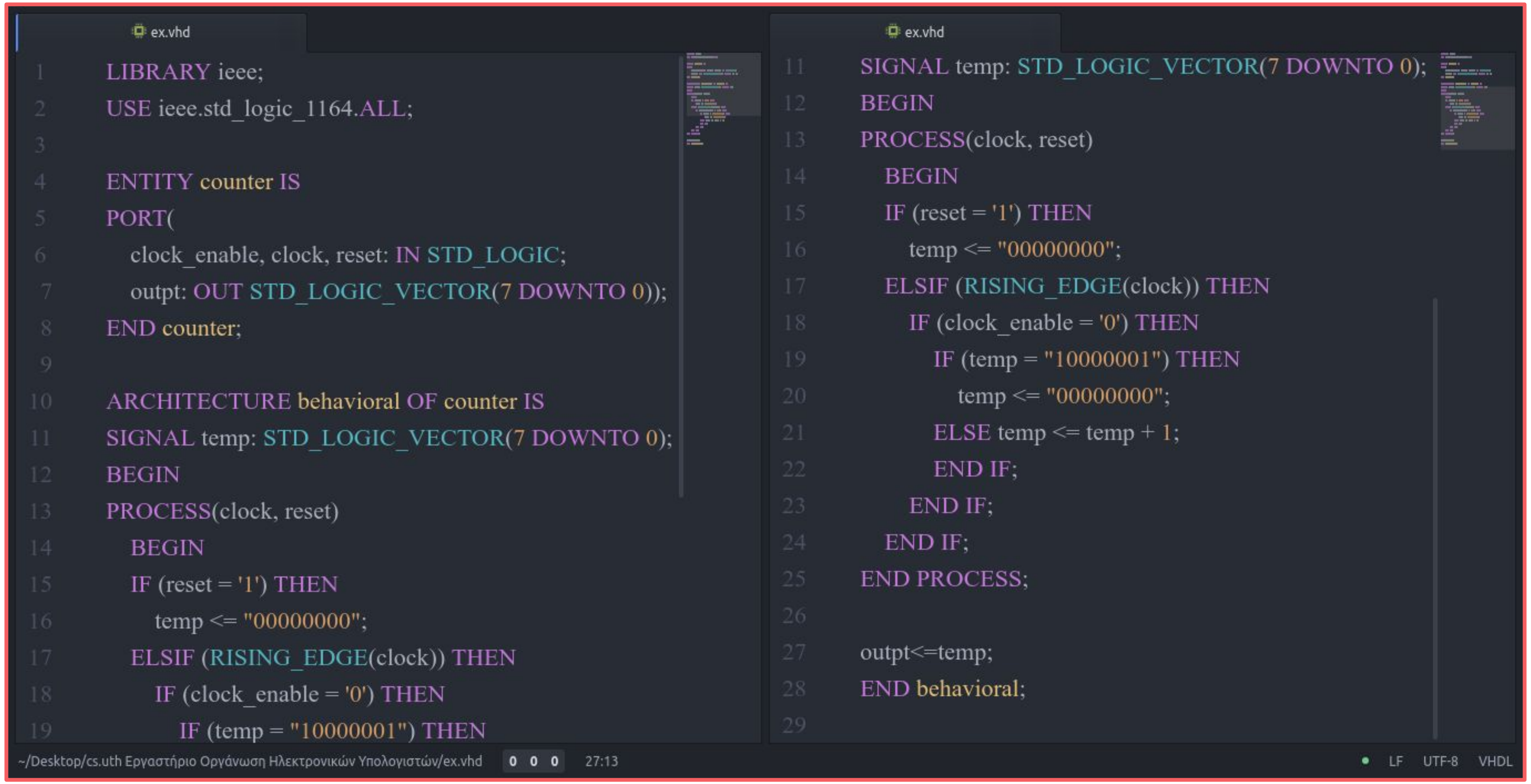
VHDL Basics

Δαδαλιάρης Αντώνιος
dadaliaris@cs.uth.gr

VHDL: 8-bit BCD Counter



ex.vhd

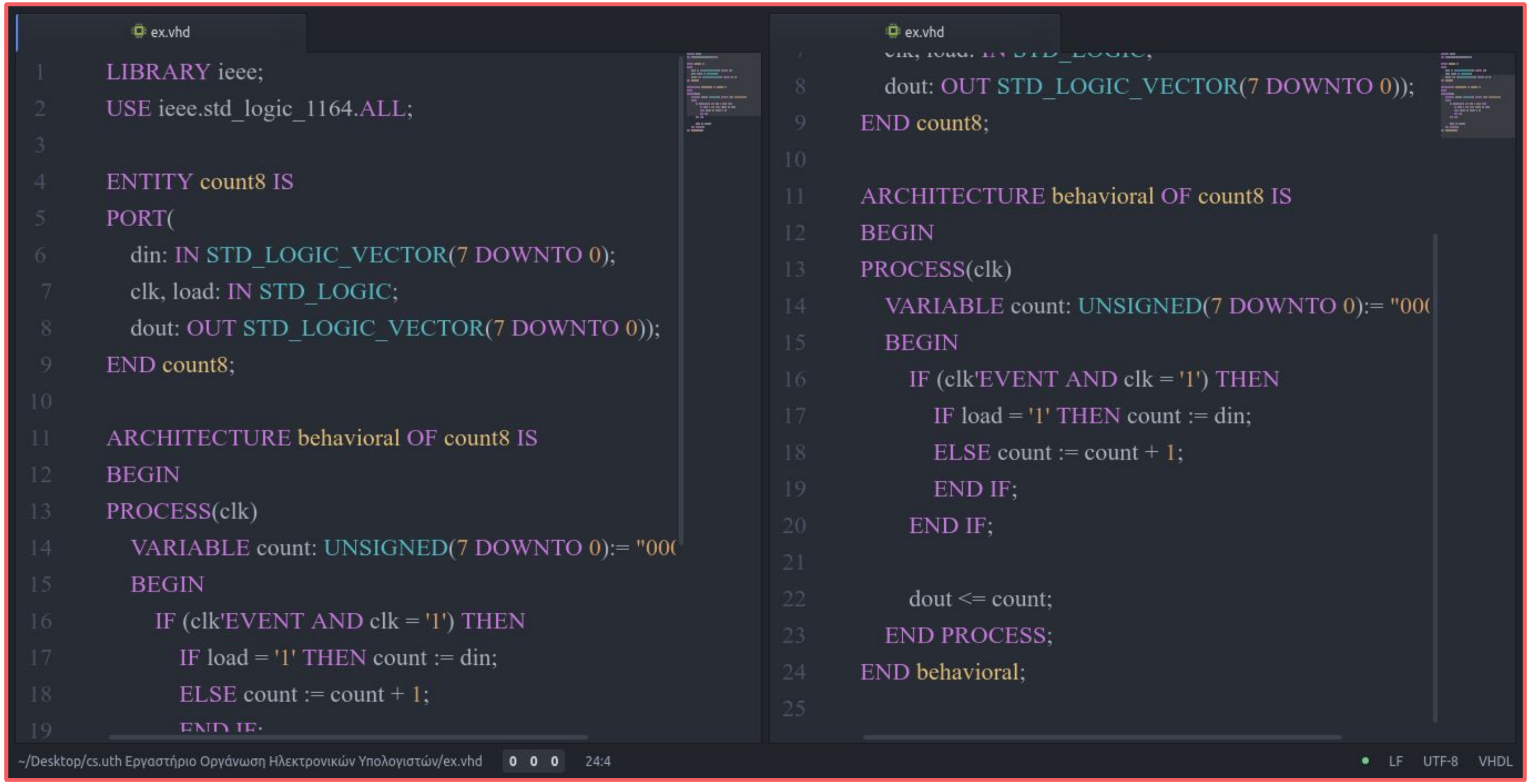
```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY counter IS
PORT(
    clock_enable, clock, reset: IN STD_LOGIC;
    outpt: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END counter;

ARCHITECTURE behavioral OF counter IS
SIGNAL temp: STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
PROCESS(clock, reset)
    BEGIN
    IF (reset = '1') THEN
        temp <= "00000000";
    ELSIF (RISING_EDGE(clock)) THEN
        IF (clock_enable = '0') THEN
            IF (temp = "10000001") THEN
```

ex.vhd

```vhdl
SIGNAL temp: STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
PROCESS(clock, reset)
    BEGIN
    IF (reset = '1') THEN
        temp <= "00000000";
    ELSIF (RISING_EDGE(clock)) THEN
        IF (clock_enable = '0') THEN
            IF (temp = "10000001") THEN
                temp <= "00000000";
            ELSE temp <= temp + 1;
            END IF;
        END IF;
    END IF;
END PROCESS;

outpt<=temp;
END behavioral;
```

~/Desktop/cs.uth Εργαστήριο Οργάνωση Ηλεκτρονικών Υπολογιστών/ex.vhd   0 0 0   27:13   LF   UTF-8   VHDL

VHDL: 8-bit Binary Counter (with load enable)



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

ENTITY count8 IS
PORT(
   din: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
   clk, load: IN STD_LOGIC;
   dout: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END count8;

ARCHITECTURE behavioral OF count8 IS
BEGIN
PROCESS(clk)
   VARIABLE count: UNSIGNED(7 DOWNTO 0):= "00(
   BEGIN
      IF (clk'EVENT AND clk = '1') THEN
         IF load = '1' THEN count := din;
         ELSE count := count + 1;
```

```vhdl
   dout: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END count8;

ARCHITECTURE behavioral OF count8 IS
BEGIN
PROCESS(clk)
   VARIABLE count: UNSIGNED(7 DOWNTO 0):= "00(
   BEGIN
      IF (clk'EVENT AND clk = '1') THEN
         IF load = '1' THEN count := din;
         ELSE count := count + 1;
```

```vhdl
         END IF;
      END IF;

      dout <= count;
   END PROCESS;
END behavioral;
```

~/Desktop/cs.uth Εργαστήριο Οργάνωση Ηλεκτρονικών Υπολογιστών/ex.vhd   0 0 0   24:4   LF   UTF-8   VHDL

VHDL: Finite State Machines (FSMs)

- A Finite State Machine is a mathematical model computation. It is an abstract machine that can be in exactly one of a finite number of states at any given time.
- An FSM can change from one state to another in response to some external inputs (transition).
- An FSM is defined by a list of its states and the conditions for each transition.

VHDL: Finite State Machines (FSMs)

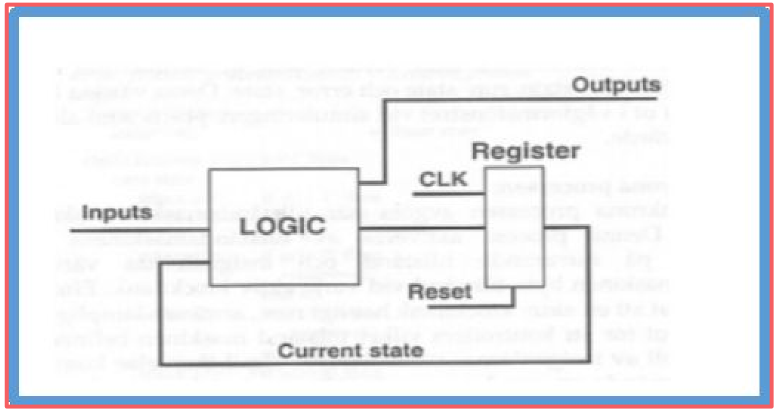
- FSM examples:
  - Vending Machines
  - Elevators
  - Traffic Lights
- Usage (modeling reactive systems):
  - Electrical Engineering
  - Linguistics
  - Computer Science (digital systems, compilers, network protocols)
  - Philosophy
  - Biology

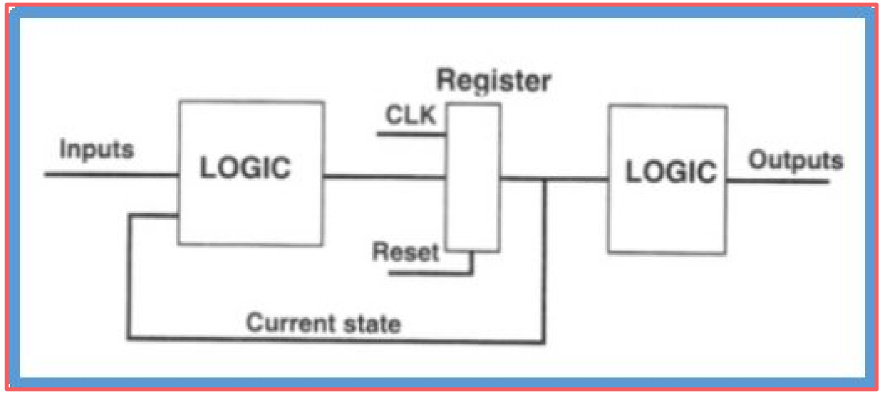
VHDL: Finite State Machines (FSMs)

- In digital design, Finite State Machines are used to model the **behavior** of a design/circuit. (TIP: behavior -> processes in our code).
- The behavior is described by a number of states, a set of input events and a set of transitions between the aforementioned states.
- The hardware implementation of an FSM requires the use of a register for storing the states, a combinational block for the definition of the transitions and a combinational block defining the outputs.

VHDL: FSM Categories

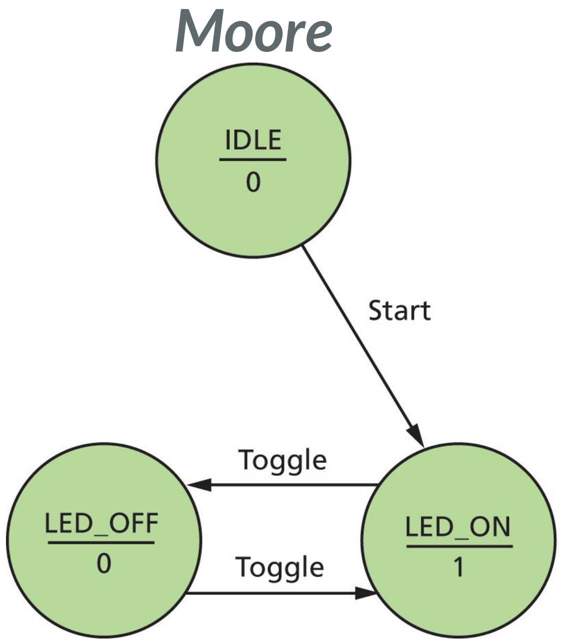
- Mealy:
  - Outputs are a function of both states and inputs (outputs depend on the state of the machine and the input value).
- Moore:
  - Outputs are a function of the state only (outputs depend only on the state of the machine).

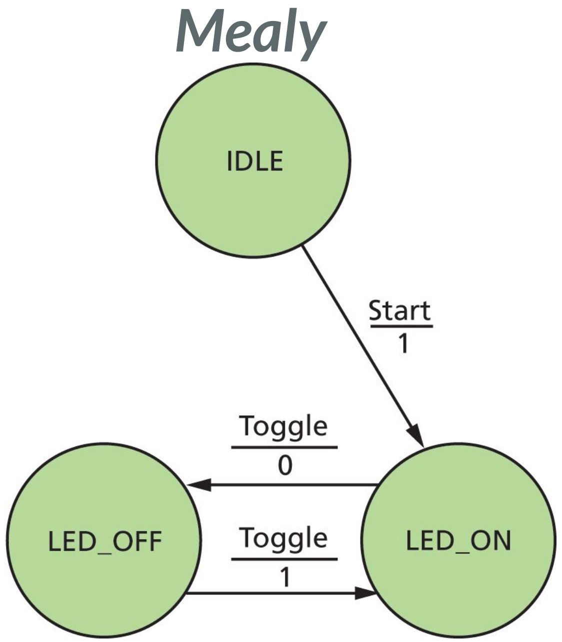




VHDL: FSM Categories

- Example (LED controller)

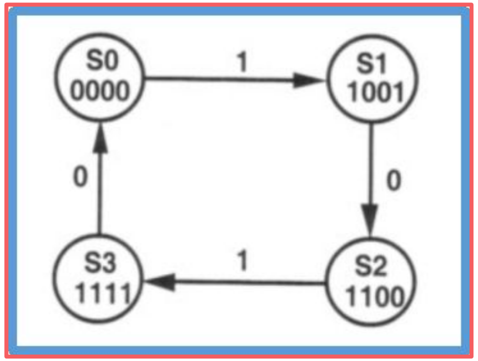


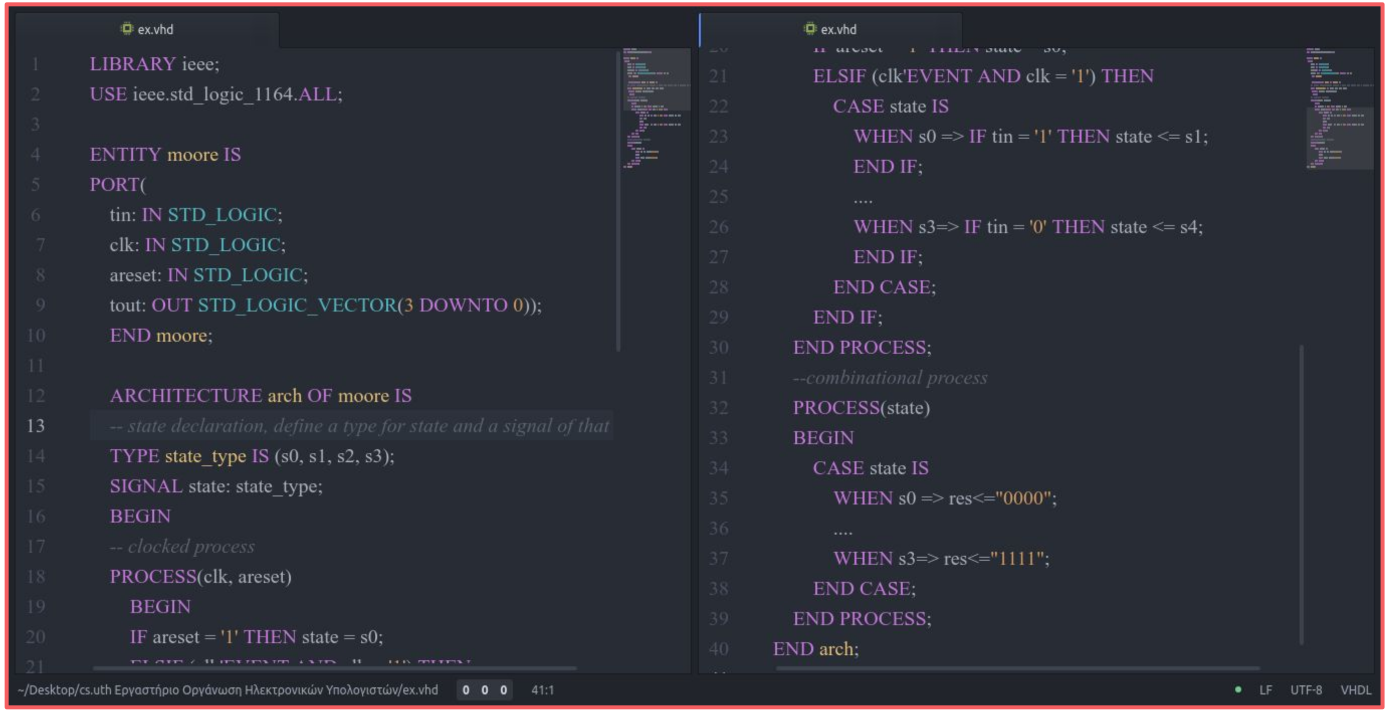


VHDL: Finite State Machines Design

1. Functional specification
2. State Transition Diagram
3. State Transition Table
4. State Encoding
5. Generate Logic Functions
6. Circuit Diagram

VHDL: Moore Machine




ex.vhd

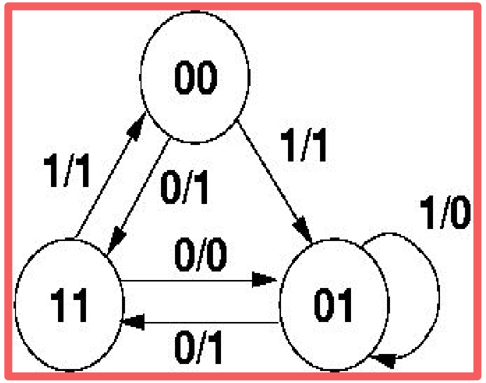
```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY moore IS
PORT(
 tin: IN STD_LOGIC;
 clk: IN STD_LOGIC;
 areset: IN STD_LOGIC;
 tout: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
 END moore;

 ARCHITECTURE arch OF moore IS
 -- state declaration, define a type for state and a signal of that
 TYPE state_type IS (s0, s1, s2, s3);
 SIGNAL state: state_type;
 BEGIN
 -- clocked process
 PROCESS(clk, areset)
  BEGIN
  IF areset = '1' THEN state = s0;
```

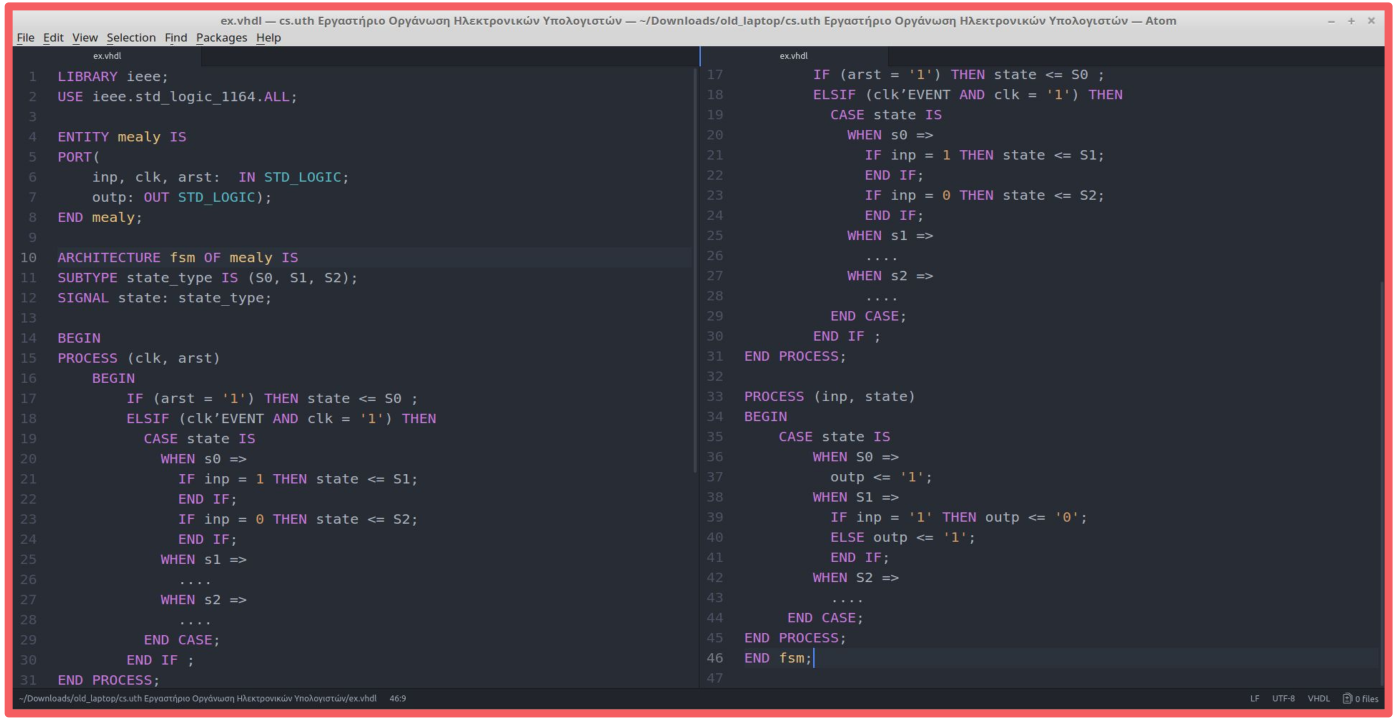
ex.vhd

```vhdl
  ELSIF (clk'EVENT AND clk = '1') THEN
   CASE state IS
    WHEN s0 => IF tin = '1' THEN state <= s1;
    END IF;
    ....
    WHEN s3=> IF tin = '0' THEN state <= s4;
    END IF;
   END CASE;
  END IF;
 END PROCESS;
 --combinational process
 PROCESS(state)
 BEGIN
  CASE state IS
   WHEN s0 => res<="0000";
   ....
   WHEN s3=> res<="1111";
  END CASE;
 END PROCESS;
END arch;
```

~/Desktop/cs.uth Εργαστήριο Οργάνωση Ηλεκτρονικών Υπολογιστών/ex.vhd 0 0 0 41:1 LF UTF-8 VHDL

VHDL: Mealy Machine




ex.vhdl — cs.uth Εργαστήριο Οργάνωση Ηλεκτρονικών Υπολογιστών — ~/Downloads/old_laptop/cs.uth Εργαστήριο Οργάνωση Ηλεκτρονικών Υπολογιστών — Atom

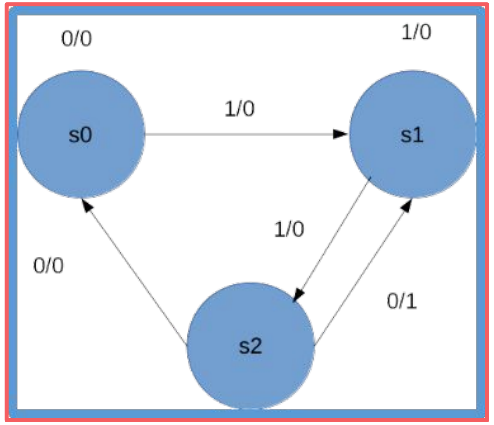
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY mealy IS
PORT(
  inp, clk, arst:  IN STD_LOGIC;
  outp: OUT STD_LOGIC);
END mealy;

ARCHITECTURE fsm OF mealy IS
SUBTYPE state_type IS (S0, S1, S2);
SIGNAL state: state_type;

BEGIN
PROCESS (clk, arst)
  BEGIN
    IF (arst = '1') THEN state <= S0 ;
    ELSIF (clk'EVENT AND clk = '1') THEN
     CASE state IS
      WHEN s0 =>
       IF inp = 1 THEN state <= S1;
       END IF;
       IF inp = 0 THEN state <= S2;
       END IF;
      WHEN s1 =>
       ....
      WHEN s2 =>
       ....
     END CASE;
    END IF ;
END PROCESS;
```

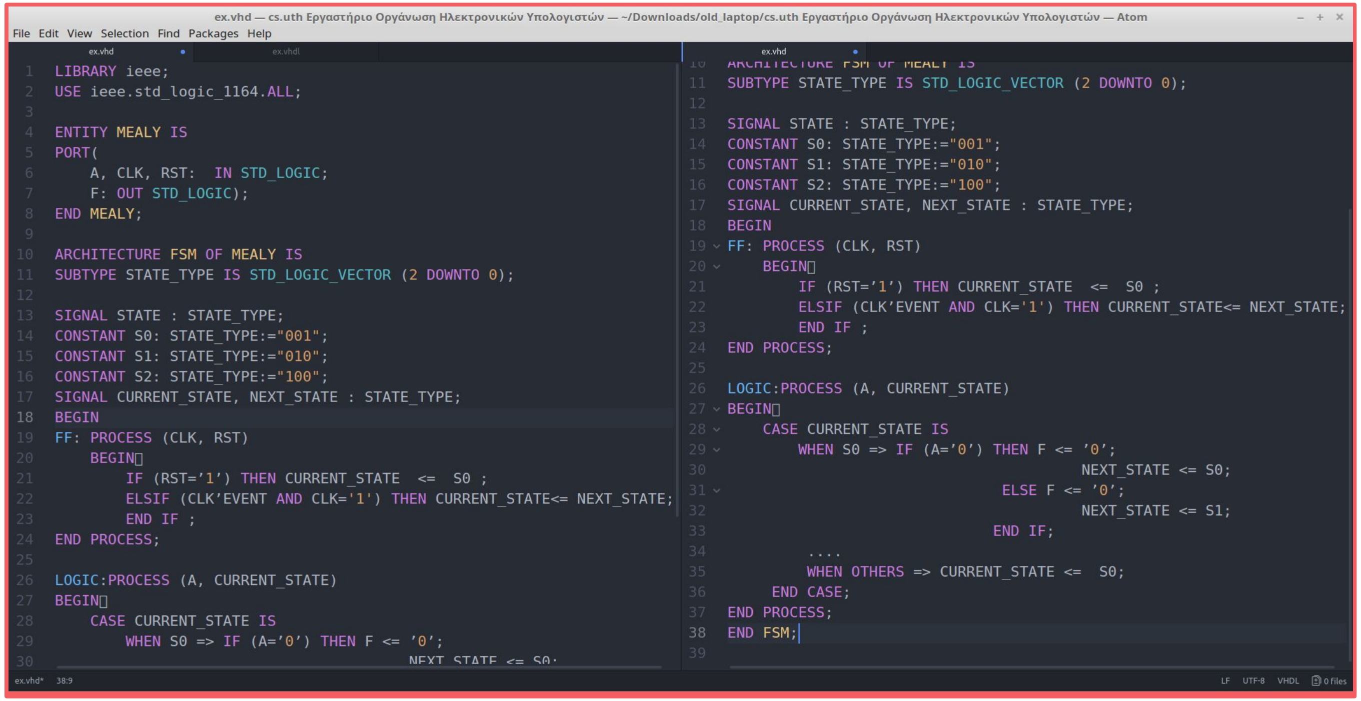
```
    IF (arst = '1') THEN state <= S0 ;
    ELSIF (clk'EVENT AND clk = '1') THEN
     CASE state IS
      WHEN s0 =>
       IF inp = 1 THEN state <= S1;
       END IF;
       IF inp = 0 THEN state <= S2;
       END IF;
      WHEN s1 =>
       ....
      WHEN s2 =>
       ....
     END CASE;
    END IF ;
END PROCESS;
```

```

PROCESS (inp, state)
BEGIN
  CASE state IS
    WHEN S0 =>
     outp <= '1';
    WHEN S1 =>
     IF inp = '1' THEN outp <= '0';
     ELSE outp <= '1';
     END IF;
    WHEN S2 =>
     ....
  END CASE;
END PROCESS;
END fsm;
```

~/Downloads/old_laptop/cs.uth Εργαστήριο Οργάνωση Ηλεκτρονικών Υπολογιστών/ex.vhdl 46:9 LF UTF-8 VHDL 0 files

VHDL: String Detector




ex.vhd — cs.uth Εργαστήριο Οργάνωση Ηλεκτρονικών Υπολογιστών — ~/Downloads/old_laptop/cs.uth Εργαστήριο Οργάνωση Ηλεκτρονικών Υπολογιστών — Atom
File Edit View Selection Find Packages Help
ex.vhd
ex.vhdl
ex.vhd
ex.vhd* 38:9
LF UTF-8 VHDL 0 files

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY MEALY IS
PORT(
    A, CLK, RST:  IN STD_LOGIC;
    F: OUT STD_LOGIC);
END MEALY;

ARCHITECTURE FSM OF MEALY IS
SUBTYPE STATE_TYPE IS STD_LOGIC_VECTOR (2 DOWNTO 0);

SIGNAL STATE : STATE_TYPE;
CONSTANT S0: STATE_TYPE:="001";
CONSTANT S1: STATE_TYPE:="010";
CONSTANT S2: STATE_TYPE:="100";
SIGNAL CURRENT_STATE, NEXT_STATE : STATE_TYPE;
BEGIN
FF: PROCESS (CLK, RST)
    BEGIN□
        IF (RST=’1’) THEN CURRENT_STATE  <=  S0 ;
        ELSIF (CLK’EVENT AND CLK='1') THEN CURRENT_STATE<= NEXT_STATE;
        END IF ;
END PROCESS;

LOGIC:PROCESS (A, CURRENT_STATE)
BEGIN□
    CASE CURRENT_STATE IS
        WHEN S0 => IF (A=’0’) THEN F <= ’0’;
                                        NEXT_STATE <= S0;
```

```
ARCHITECTURE FSM OF MEALY IS
SUBTYPE STATE_TYPE IS STD_LOGIC_VECTOR (2 DOWNTO 0);

SIGNAL STATE : STATE_TYPE;
CONSTANT S0: STATE_TYPE:="001";
CONSTANT S1: STATE_TYPE:="010";
CONSTANT S2: STATE_TYPE:="100";
SIGNAL CURRENT_STATE, NEXT_STATE : STATE_TYPE;
BEGIN
FF: PROCESS (CLK, RST)
    BEGIN□
        IF (RST=’1’) THEN CURRENT_STATE  <=  S0 ;
        ELSIF (CLK’EVENT AND CLK='1') THEN CURRENT_STATE<= NEXT_STATE;
        END IF ;
END PROCESS;

LOGIC:PROCESS (A, CURRENT_STATE)
BEGIN□
    CASE CURRENT_STATE IS
        WHEN S0 => IF (A=’0’) THEN F <= ’0’;
                                        NEXT_STATE <= S0;
                               ELSE F <= ’0’;
                                        NEXT_STATE <= S1;
                              END IF;

         ....
         WHEN OTHERS => CURRENT_STATE <=  S0;
     END CASE;
END PROCESS;
END FSM;

```

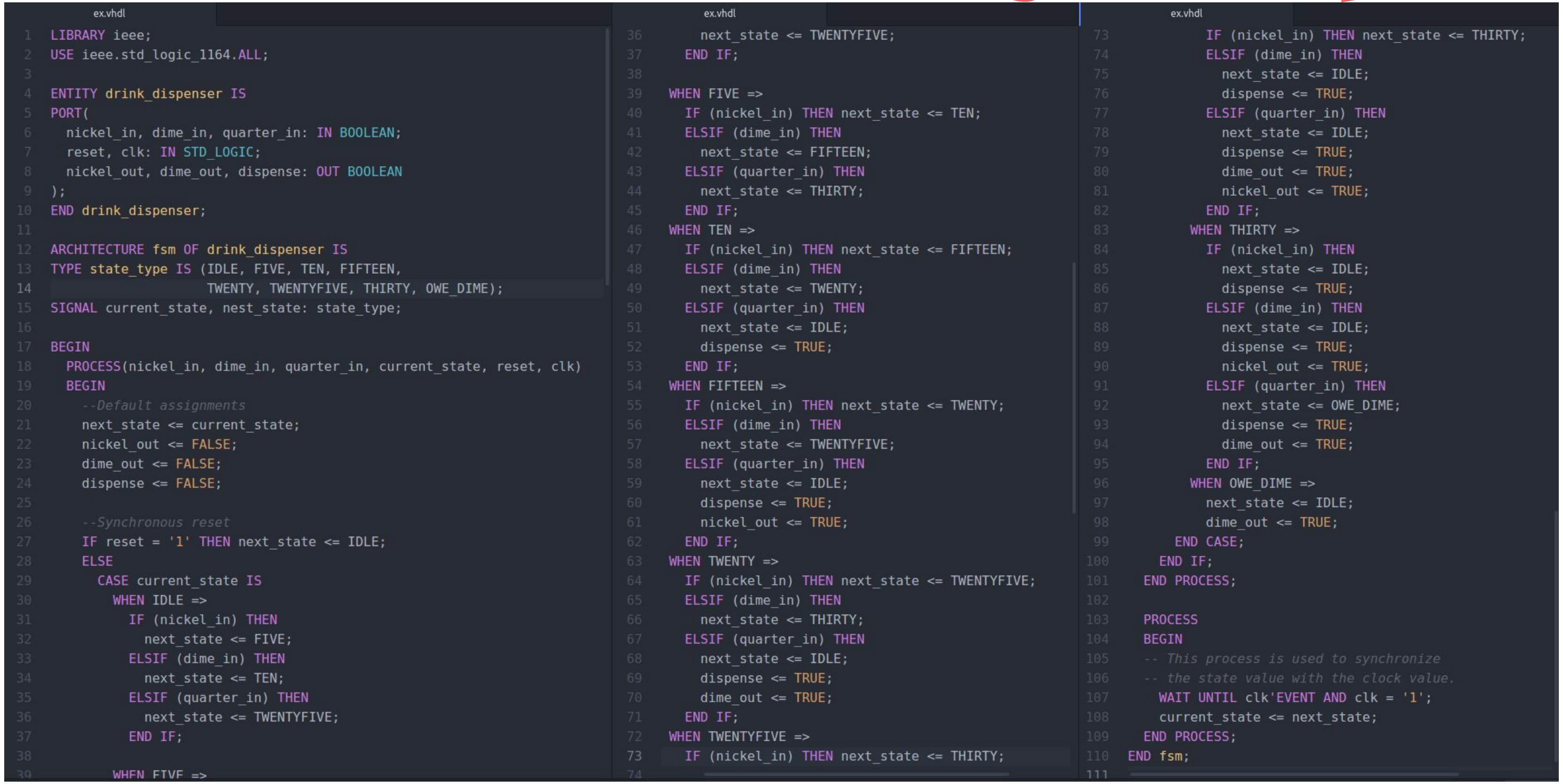
Finite State Machines Design Example

- **Control Unit of a Vending Machine**
- INPUTS:
  a. Clock signal
  b. Reset Signal
  c. Coin
- OUTPUTS:
  a. Change
  b. Dispensed drink

Finite State Machines Design Example

- Additional info:
    a. Only one soft drink is dispensed
    b. A drink costs 35 cents
- Different inputs for different coins (nickel(5), dime(10), quarter(25)), different outputs for different change plus an output for dispense.

Finite State Machines Design Example



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

ENTITY drink_dispenser IS
PORT(
  nickel_in, dime_in, quarter_in: IN BOOLEAN;
  reset, clk: IN STD_LOGIC;
  nickel_out, dime_out, dispense: OUT BOOLEAN
);
END drink_dispenser;

ARCHITECTURE fsm OF drink_dispenser IS
TYPE state_type IS (IDLE, FIVE, TEN, FIFTEEN,
                    TWENTY, TWENTYFIVE, THIRTY, OWE_DIME);
SIGNAL current_state, nest_state: state_type;

BEGIN
  PROCESS(nickel_in, dime_in, quarter_in, current_state, reset, clk)
  BEGIN
    --Default assignments
    next_state <= current_state;
    nickel_out <= FALSE;
    dime_out <= FALSE;
    dispense <= FALSE;

    --Synchronous reset
    IF reset = '1' THEN next_state <= IDLE;
    ELSE
      CASE current_state IS
        WHEN IDLE =>
          IF (nickel_in) THEN
            next_state <= FIVE;
          ELSIF (dime_in) THEN
            next_state <= TEN;
          ELSIF (quarter_in) THEN
            next_state <= TWENTYFIVE;
          END IF;

        WHEN FIVE =>
          IF (nickel_in) THEN next_state <= TEN;
          ELSIF (dime_in) THEN
            next_state <= FIFTEEN;
          ELSIF (quarter_in) THEN
            next_state <= THIRTY;
          END IF;
        WHEN TEN =>
          IF (nickel_in) THEN next_state <= FIFTEEN;
          ELSIF (dime_in) THEN
            next_state <= TWENTY;
          ELSIF (quarter_in) THEN
            next_state <= IDLE;
            dispense <= TRUE;
          END IF;
        WHEN FIFTEEN =>
          IF (nickel_in) THEN next_state <= TWENTY;
          ELSIF (dime_in) THEN
            next_state <= TWENTYFIVE;
          ELSIF (quarter_in) THEN
            next_state <= IDLE;
            dispense <= TRUE;
            nickel_out <= TRUE;
          END IF;
        WHEN TWENTY =>
          IF (nickel_in) THEN next_state <= TWENTYFIVE;
          ELSIF (dime_in) THEN
            next_state <= THIRTY;
          ELSIF (quarter_in) THEN
            next_state <= IDLE;
            dispense <= TRUE;
            dime_out <= TRUE;
          END IF;
        WHEN TWENTYFIVE =>
          IF (nickel_in) THEN next_state <= THIRTY;
          ELSIF (dime_in) THEN
            next_state <= IDLE;
            dispense <= TRUE;
          ELSIF (quarter_in) THEN
            next_state <= IDLE;
            dispense <= TRUE;
            dime_out <= TRUE;
            nickel_out <= TRUE;
          END IF;
        WHEN THIRTY =>
          IF (nickel_in) THEN
            next_state <= IDLE;
            dispense <= TRUE;
          ELSIF (dime_in) THEN
            next_state <= IDLE;
            dispense <= TRUE;
            nickel_out <= TRUE;
          ELSIF (quarter_in) THEN
            next_state <= OWE_DIME;
            dispense <= TRUE;
            dime_out <= TRUE;
          END IF;
        WHEN OWE_DIME =>
          next_state <= IDLE;
          dime_out <= TRUE;
      END CASE;
    END IF;
  END PROCESS;

  PROCESS
  BEGIN
  -- This process is used to synchronize
  -- the state value with the clock value.
    WAIT UNTIL clk'EVENT AND clk = '1';
    current_state <= next_state;
  END PROCESS;
END fsm;
```