Finite state machines

Modelling FSM in VHDL

Types of automata (FSM)

- A sequential automaton has:
  - Inputs
  - States (a finite number of states)
  - Outputs
  - Transitions from one state to another
- Types of FSM:
  - Moore:
    - outputs depend only on the current state
  - Mealy:
    - outputs depend on both state and inputs

VHDL modelling

- Can be done with
  - one process
  - two processes
- We define a type states (of type ennumeration)
  - TYPE states IS (s0, s1, s2, s3);
  - TYPE states IS (init, add, shift);
- We have:
  - Either present state (`present_state`) and next state (`next_state`) of type `states`
  - Or only state (`state`) of type `states`
- The states (`present_state, next_state, state`) may be signals or variable, depending on the implementation (they should be signals if we implement FSM with two processes)
- One process determines next state and the outputs
  - Contains a CASE after the present state, and on each branch (choice) there are IF statements with the inputs in conditions

Modelling

- With two processes:
  - one process is sensitive to clock and does only `present_state<=next_state;`
  - The other process is sensitive to `present_state` and inputs; it computes `next_state`
- With one process:
  - The process is sensitive to clock
  - The state (`state`) may be a variable
- Moore / Mealy:
  - For Moore: outputs depend only on the current state
  - For Mealy: outputs depend on both state and inputs
- One possibility would be to use the negative (falling) edge of the clock for the FSM, if the other sequential circuits use the positive (rising) edge of the clock
- The outputs of the FSM are commands for the other circuits (e.g. shift, load, etc)

Moore FSM example

	x=‘0’	x=‘1’
stare = s0 z=“00”	s0	s3
stare = s1 z=“01”	s1	s0
stare = s2 z=“10”	s2	s1
stare = s3 z=“11”	s2	s1

```
ENTITY FSM_Moore IS
        GENERIC(tp:TIME:=5ns);
        PORT(x: IN BIT; clock: IN BIT;
                z: OUT BIT_VECTOR(1 DOWNTO 0));
END ENTITY;

ARCHITECTURE one_process OF FSM_Moore IS
TYPE states IS (s0, s1, s2, s3);
SIGNAL display_state: states;
BEGIN
PROCESS (clock)
        VARIABLE state: states:=s0;
        --VARIABLE count: integer:=0;
BEGIN
        IF clock='0' AND clock'EVENT THEN
        --display_state<=state; -- visualize present state
                CASE state IS
                        WHEN s0 =>
                                z<="00" after tp;
                                --count:=0;--init count
                                IF x='0' THEN
                                        state:=s0;
                                ELSE
                                        state:=s3;
                                END IF;
                        WHEN s1 =>
```

```
                                        z<="01" after tp;
                                        --count:= count+1;-- increment count
                                        IF x='0' THEN
                                                state:=s1;
                                        ELSE
                                                state:=s0;
                                        END IF;
                                WHEN s2 =>
                                        z<="10" after tp;
                                        --IF count>=5 THEN -- test count
                                        IF x='0' THEN
                                                state:=s2;
                                        ELSE
                                                state:=s1;
                                        END IF;
                                WHEN s3 =>
                                        z<="11" after tp;
                                        IF x='0' THEN
                                                state:=s2;
                                        ELSE    state:=s1;
                                        END IF;
                        END CASE;
                        display_state<=state;--visualize next state
                END IF;
END PROCESS;
END ARCHITECTURE;
```

Example of Mealy FSM

	x=‘0’	x=‘1’
present_state = s0	s0 / “00”	s3 / “11”
present_state = s1	s1 / “01”	s0 / “00”
present_state = s2	s2 / “10”	s1 / “01”
present_state = s3	s2 / “10”	s1 / “01”

```
ENTITY FSM_Mealy IS
        GENERIC(tp:TIME:=5ns);
        PORT(x: IN BIT; clock: IN BIT; z: OUT BIT_VECTOR(1 DOWNTO 0));
END ENTITY;
ARCHITECTURE two_processes OF FSM_Mealy IS
        TYPE states IS (s0, s1, s2, s3);
        SIGNAL present_state, next_state: states:=s0;
BEGIN
proc_clock: PROCESS(clock)
BEGIN
        IF clock='0' AND clock'EVENT THEN
                present_state <= next_state;
        END IF;
END PROCESS proc_clock;
compute_next_state: PROCESS (present_state, x)
        --VARIABLE count: integer:=0;
BEGIN
        CASE present_state IS
                WHEN s0 =>              --count:=0;--initialize count
                        IF x='0' THEN
                                next_state<=s0;
                                z<="00" after tp;
                        ELSE
                                next_state<=s3;
                                z<="11" after tp;
                        END IF;
                WHEN s1 =>              --count:= count+1;-- increment count
```

```
                    IF x='0' THEN
                            next_state<=s1;
                            z<="01" after tp;
                    ELSE
                            next_state<=s0;
                            z<="00" after tp;
                    END IF;
            WHEN s2 =>              --IF count>=5 THEN -- test count
                    IF x='0' THEN
                            next_state<=s2;
                            z<="10" after tp;
                    ELSE
                            next_state<=s1;
                            z<="01" after tp;
                    END IF;
            WHEN s3 =>
                    IF x='0' THEN
                            next_state<=s2;
                            z<="10" after tp;
                    ELSE
                            next_state<=s1;
                            z<="01" after tp;
                    END IF;
            END CASE;
END PROCESS compute_next_state;
END ARCHITECTURE;
```