Finite state machines (FSM) in VHDL
Finite State Machine

- Model of behavior composed of a **finite number of states**, **input events**, **transitions between those states** (rules or conditions), and **actions**.

- Effective method for implementing control functionality.

- FSM design - just a step beyond sequential design:
  - HW implementation requires a register to store state variables, a block of combinational logic which determines the state transition, and a block of combinational logic that determines the output of the FSM.

- Current state is stored in registers and updated synchronous to the clock

- Two phases:
  - calculate new state
  - new state is sampled into a register
Two main types

- **Moore machine**
  - Edward F. Moore, 1956

- **Mealy machine**
  - George H. Mealy, 1955
Moore machine

- Outputs of Moore machine are a function of the present state only
- Output transitions are “synchronous” to system clock
- Propagation delay through output logic nevertheless leads to asynchronous outputs which can lead to slower operating frequencies
• Outputs of Mealy machine are a function of the present state and all the inputs.

• Outputs transitions are asynchronous to the clock
  – They change immediately when the inputs change

• ⇒ A Mealy machine works one clock cycle in advance of a Moore machine
Design of FSM in VHDL

Declaration of states

Choose explanatory names using enumerated types

```vhdl
-- Declaration part

type state_type is (IDLE, START, TASK1, DELAY, TASK2, STOP);
signal current_state : state_type;
```

Clocked process

Moore

Mealy

Combinational output process

Moore

Mealy
library ieee;
use ieee.std_logic_1164.all;

entity statemachine is
port(--inputs,outputs);
end statemachine;

architecture skeleton of statemachine is

--Declaration part----------------------------------
type state_type is (IDLE,START,TASK1,DELAY,TASK2,STOP);
signal current_state : state_type;

begin

--synchronous process--------------------------------
state_proc: process(clock,reset) --clocked process
begin
    --update active state
end process;

--Combinatoric output process------------------------
output_proc: process(current_state,[inputs])
begn
    --output assignment
end process;
end skeleton;
Enumerated data type

• It is possible to define your own enumerated data type in VHDL

• This data type allows a user to specify the list of legal values that a variable or signal of the defined type may be assigned

• Commonly used for state machines:

```vhdl
    type state_type is (start, idle, waiting, run);
    signal state : state_type
```

• Most synthesis tools can build logic from enumerated types
Examples of predefined types

```vhdl
type std_ulogic is ('U', -- Uninitialized
    'X', -- Forcing unknown
    '0', -- Forcing 0
    '1', -- Forcing 1
    'Z', -- High impedance
    'W', -- Weak unknown
    'L', -- Weak 0
    'H', -- Weak 1
    '-', -- Don't care
);

type std_logic is resolved std_ulogic;
```

```vhdl
type boolean is (false, true);
```

```vhdl
type bit is ('0', '1');
```

Converted to 1 and 0 during synthesis
Design of FSM in VHDL

• 1 process

• 2 processes
  – Clocked state process
  – Combinational output process

• 3 processes
  – Combinational coding of next state
  – Clocked update of present state (present state <= next state)
  – Combinational output process
Example – Moore (2 processes)
library ieee;
use ieee.std_logic_1164.all;

entity statemachine is
  port(clk, test, reset : in std_logic;
       out1: out std_logic_vector(3 downto 0));
end statemachine;

architecture moore of statemachine is

type state_type is (s0, s1, s2, s3);  -- state declaration
signal state: state_type;

Moore - 2 prosesser
Clocked process

Moore - 2 prosesser
Clocked process

```vhdl
16  state_proc: process(clk,areset) --clocked process
17    begin
18      if areset = '1' then
19          state <= s0;
20      elsif rising_edge(clk) then
21        --state <= state; --keep old value if not changed
22        case state is
23          when s0 =>
24            if test = '1' then
25                state <= s1;
26            end if;
27          when s1 =>
28            if test = '0' then
29                state <= s2;
30            end if;
31          when s2 =>
32            if test = '1' then
33                state <= s3;
34            end if;
35          when s3 =>
36            if test = '0' then
37                state <= s0;
38            end if;
39          end case;
40      end if;
41    end process;
```
Combinational output process

Outputs of Moore machine are a function of the present state only.
Combinational output process

Outputs of Moore machine are a function of the present state only.
Example - Moore (3 processes)
Declaraction

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity statemachine is
  port(clk, test, areset : in std_logic;
       out1: out std_logic_vector(3 downto 0));
end statemachine;

architecture moore of statemachine is

  type state_type is (s0, s1, s2, s3);  -- state declaration
  signal next_state: state_type;
  signal current_state : state_type;
```

Moore - 3 prosesser
Decoding of next state

Moore - 3 prosesser
Decoding of next state

```plaintext
state_decode_proc: process(current_state, test) --combinatoric process
begin
  case current_state is
    when s0 =>
      if test = '1' then
        next_state <= s1;
      else
        next_state <= s0;
      end if;
    when s1 =>
      if test = '0' then
        next_state <= s2;
      else
        next_state <= s1;
      end if;
    when s2 =>
      if test = '1' then
        next_state <= s3;
      else
        next_state <= s2;
      end if;
    when s3 =>
      if test = '0' then
        next_state <= s0;
      else
        next_state <= s3;
      end if;
  end case;
end process;
```

Moore - 3 prosesser
Clocked process

- Update of present state (current state)
Clocked process

- Update of present state (current state)

```vhdl
48 state_proc: process(clk, areset) -- clocked process
49 begin
50 if areset = '1' then
51 current_state <= s0;
52 elsif rising_edge(clk) then
53 current_state <= next_state;
54 end if;
55 end process;
```
Combinational output process

Outputs of Moore machine are a function of the present state only.
Combinational output process

```
58  output_proc: process(current_state) --combinatorial process
59 begin
60   case current_state is
61     when s0 =>
62       out1 <= "0000";
63     when s1 =>
64       out1 <= "1001";
65     when s2 =>
66       out1 <= "1100";
67     when s3 =>
68       out1 <= "1111";
69   end case;
70 end process;
71 end moore;
```

Outputs of Moore machine are a function of the present state only.
Mealy

- Outputs of Mealy machine are a function of the present state and all the inputs.
• Outputs of Mealy machine are a function of the present state and all the inputs.
library ieee;
use ieee.std_logic_1164.all;

entity statemachine_mealy is
port(clk, test, arest : in std_logic;
     out1: out std_logic_vector(3 downto 0));
end statemachine_mealy;

architecture mealy of statemachine_mealy is

type state_type is (s0,s1,s2,s3); --state declaration
signal state: state_type;

Identical to Moore machine
Clocked process

state_proc: process(clk,areset) --clocked process
begin
  if areset = '1' then
    state <= s0;
  elsif rising_edge(clk) then
    case state is
      when s0 =>
        if test = '1' then
          state <= s1;
        end if;
      when s1 =>
        if test = '0' then
          state <= s2;
        end if;
      when s2 =>
        if test = '1' then
          state <= s3;
        end if;
      when s3 =>
        if test = '0' then
          state <= s0;
        end if;
    end case;
  end if;
end process;

Identical to Moore machine
Combinational output process

```
output_proc: process(state,test) --combinatoric process
begin
  case state is
    when s0 =>
      if test = '1' then
        out1 <= "1001";
      else
        out1 <= "0000";
      end if;
    when s1 =>
      if test = '0' then
        out1 <= "1100";
      else
        out1 <= "1001";
      end if;
    when s2 =>
      if test = '1' then
        out1 <= "1111";
      else
        out1 <= "1100";
      end if;
    when s3 =>
      if test = '1' then
        out1 <= "0000";
      else
        out1 <= "1111";
      end if;
  end case;
end process;
```

Outputs directly dependent on value of inputs
Mealy

Clk = ’1’

Utgang endres

Inngang endres

Utgang følger både inngang og nå-tilstand, mens nå-tilstand skifter på klokken
Mealy vs Moore
Mealy vs Moore