VHDL => Finite State Machines



1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machine

1

How To Model FSMs?

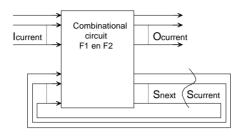
- **⇒** Moore / Mealy
 - asynchronous
 - synchronous register outputs
- **⇒**Reset
 - synchronous
 - asynchronous



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines



Asynchronous Mealy

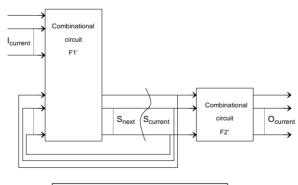




© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines

3

Asynchronous Moore

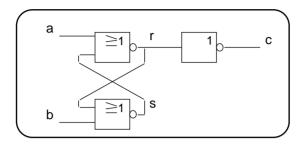


$$S_{next} = F1' (S_{current}, I_{current})$$
 $O_{current} = F2' (S_{current})$



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines

Summary Asynchronous FSM's





Moore

State variable: r r = /a (r + b) c = /r

Mealy

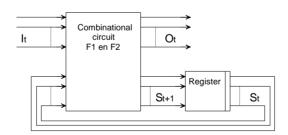
State variable: s s = /b (s + a) c = a + s



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines

_

Synchronous Mealy

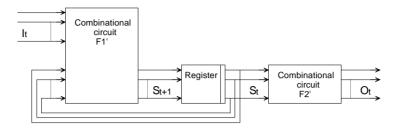


$$S_{t+1} = F1 (S_t, I_t)$$
 $O_t = F2 (S_t, I_t)$



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines

Synchronous Moore



$$S_{t+1} = F1' (S_t, I_t)$$

 $O_t = F2' (S_t)$



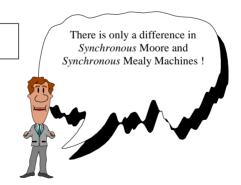
1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machine.

7

Summary Synchronous FSM's

Synchronous Mealy

(some) inputs may change (some) outputs directly.



Synchronous Moore

- · Inputs never changes outputs directly.
- · Outputs can only change after an active edge of the clock.



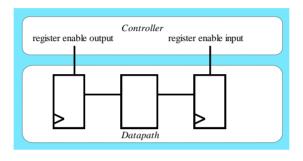
© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines



Synchronous FSM With Register Outputs

In applications often some of the outputs of the FSM should be available soon after the active edge of the clock.

Example: a Controller (a FSM) and a Data path. (Some) *register enable output* signals are part of the critical path.

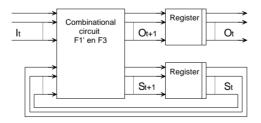




© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machine

9

Register Outputs /2





© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines



Register Outputs /3

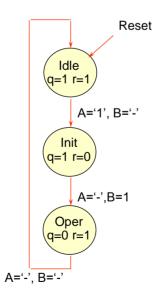
- Advantage
 - output signals are early available in the clock period
 - faster
- Disadvantage
 - (often) more registers needed
 - more hardware needed? (maybe technology dependent, especially for programmable logic)



1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines

11

FSM Alternatives; Moore



Write a VHDL description for this FSM

• asynchronous reset
• clock signal is 'clk'

4

© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines



FSM VHDL Coding Styles

- One process only
 - Handles both state transitions and outputs
- Two processes
 - A synchronous process for updating the state register
 - A combinational process for conditionally deriving the next machine state and updating the outputs
- Three processes
 - A synchronous process for updating the state register
 - A combinational process for conditionally deriving the next machine state
 - A combinational process for conditionally deriving the outputs



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines

13

Synchronous Moore - Single Process

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY moore1 IS
 PORT (clk : IN std_logic; reset : IN std_logic;
     a, b : IN std_logic;
     q, r : OUT std_logic);
END moore1;
ARCHITECTURE asyn_reset_sol1 OF moore
BEGIN
 PROCESS(reset,clk)
  TYPE states IS (idle, init, oper);
  VARIABLE state: states;
 BEGIN
  IF reset='0' THEN
   state := idle; q<='1'; r<='1';
  ELSIF clk'event and clk='1' THEN
```

```
-- next state
   CASE state IS
     WHEN idle => IF a='1' THEN state:=init; END IF;
     WHEN init => IF b='1' THEN state:=oper; END IF;
     WHEN oper => state:=idle;
    WHEN OTHERS => state := idle;
   END CASE;
   -- output
   CASE state IS
     WHEN idle => q<='1'; r<='1';
    WHEN init => q<='1'; r<='0'; WHEN oper => q<='0'; r<='1';
    WHEN OTHERS => q<='-'; r<='-';
   END CASE;
  END IF:
 END PROCESS:
END asyn_reset_sol1;
  Questions:
```

- How are the states coded?
- Is the "WHEN OTHERS" needed?
- · How many registers are expected?



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines



Synchronous Moore - Single Process (Cont.)

- Coding of the states by synthesis tools
 - binary coded (default for most synthesis tools)
 - gray
 - one-hot

• ..

It is the experience of many Xilinx users that 'one hot' coding seems to work fine.



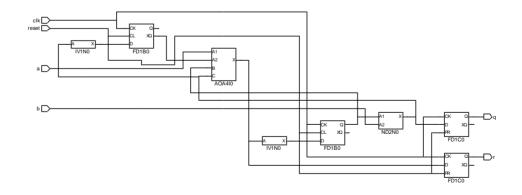
- ⇒ Is the enumeration order of the type states important?
- ⇒ A way to prevent 'deadlock' is to extend the number of literals of the enumeration type to a power of two (binary coding is assumed). Most synthesis tool will prevent deadlock automatically.
- This way of coding results in 'register outputs'.



1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machine.

1

Synchronous Moore - Single Process (Cont.)



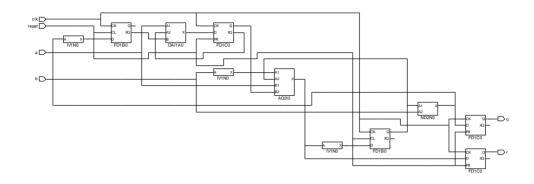
Outputs have registers



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines



Synchronous Moore - Single Process (Cont.)



One-hot encoding



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines

17

18

Synchronous Moore - Two Processes

TYPE states IS (idle, init, oper); SIGNAL state: states;

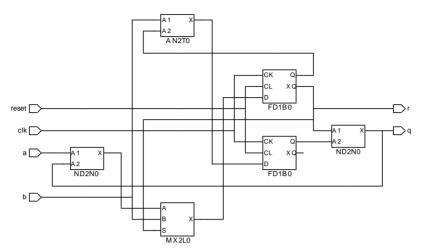
```
PROCESS(reset,clk)
                                                        output:PROCESS(state)
                                                       BEGIN
BEGIN
  IF reset='0' THEN
                                                        CASE state IS
   state <= idle:
                                                          WHEN idle => q<='1'; r<='1';
  ELSIF clk'event and clk='1' THEN
                                                          WHEN init => q<='1'; r<='0';
   -- next state
                                                          WHEN oper => q<='0'; r<='1';
   CASE state IS
                                                         WHEN OTHERS => q<='-'; r<='-';
    WHEN idle => IF a='1' THEN state<=init; END IF;
                                                        END CASE:
    WHEN init => IF b='1' THEN state<=oper; END IF;
                                                       END PROCESS:
    WHEN oper => state <= idle;
    WHEN OTHERS => state <= idle;
   END CASE;
  END IF:
 END PROCESS;
```

Mostly not registered outputs!



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines

Synchronous Moore - Two Processes (Cont.)



Mostly not registered outputs!



1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machine

19

Synchronous Moore - Two Processes (Cont.)

- Register outputs can be achieved using user defined coding of the states:
 - The length of the vector is equal to the number of outputs.

Each output has its own 'bit' in the state coding vector.

Additional internal registers are needed if more states have equal output patterns.

Example:

output 'q' and 'r' and all states have different outputs. => type of states is std_logic_vector(1 DOWNTO 0);

	q	r
idle	ĺ	1
init	1	0
oner	0	1



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines



Synchronous Moore - Two Processes (Cont.)

```
CONSTANT idle: std_logic_vector (1 DOWNTO 0) := "11";
CONSTANT init: std_logic_vector (1 DOWNTO 0) := "10";
CONSTANT oper: std_logic_vector (1 DOWNTO 0) := "01";
SIGNAL state: std_logic_vector (1 DOWNTO 0);
```

```
PROCESS(clk,reset)
                                                      Concurrent statements
BEGIN
 IF reset='1' THEN
                                                       q \le state(1):
  state <= idle:
                                                        r \le state(0);
 ELSIF clk'event AND clk='1' THEN
  CASE state IS
    WHEN idle => IF a='1' THEN state<=init; END IF;
    WHEN init => IF b='1' THEN state<=oper; END IF;
    WHEN oper => state<=idle;
    WHEN OTHERS => state <= idle;
  END CASE:
 END IF;
END PROCESS:
```

register outputs due to user defined coding of the states

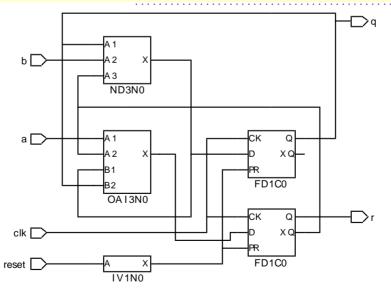
9

© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines

21

22

Synchronous Moore - Two Processes (Cont.)





© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines



Synchronous Moore; IF THEN

State transaction using IF .. THEN .. ELSIF

```
PROCESS
BEGIN

WAIT UNTIL clk='1';
IF reset='1'

THEN state <= idle;
ELSIF state= idle

THEN IF a='1' THEN state<=init; END IF;
ELSIF state= init

THEN IF b='1' THEN state<=oper; END IF;
ELSIF state=oper

THEN state<=idle;
ELSE state <= idle;
END IF;
END PROCESS;
```

- Notice that the IF THEN ELSIF structure is less readable!
- Most synthesis tools advice to use the CASE in stead of the IF THEN ELSE.

q <= state(1);
r <= state(0);</pre>



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines

2

State Encoding: Tool Independent Solution

```
attribute ENUM_ENCODING : string;
type STATES is (REQ_WAIT, ACK_SET, NREQ_WAIT, ACK_CLR);
attribute ENUM_ENCODING of STATES: TYPE IS "00 11 01 10";

signal STATE : STATES := REQ_WAIT;
begin
SET_OUTPUT: process
begin
...
case STATE is
when REQ_WAIT =>
```

- The IEEE Std. 1076.6-1999 standardized the coding of the states.
- In this example two registers are used.
- State "ACK SET" is coded with "11".
- Enum_encoding can be used for (sub)types.



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines



State Encoding: Tool Independent Solution

```
attribute ENUM_ENCODING : string;
type STATES is (REQ_WAIT, ACK_SET, NREQ_WAIT, ACK_CLR);
attribute ENUM_ENCODING of STATES: TYPE IS "0001 0100 1000 0010";

signal STATE : STATES := REQ_WAIT;
begin
SET_OUTPUT: process
begin
...
case STATE is
when REQ_WAIT =>

Note:
- synplify use(d) the attribute
syn_enum_encoding
- Quartus use(d) the attribute
syn_encoding
```

- In this example four registers are used.
- State "ACK SET" is coded with "0100".



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines

25

State Encoding: Tool Independent Solution

```
attribute FSM_STATE: string;
attribute FSM_COMPLETE: string;
type STATES is (REQ_WAIT, ACK_SET, NREQ_WAIT, ACK_CLR);
signal STATE1, STATE2: STATES:= REQ_WAIT;
attribute FSM_STATE of STATE1: SIGNAL IS "BINARY";
attribute FSM_STATE of STATE2: SIGNAL IS "00 11 10 01";
attribute FSM_COMPLETE of STATE1: signal is TRUE;
```

- **⇒** The IEEE Std. 1076.6-2004 added FSM STATE
 - Can be used for (sub)types, signals and variables
- Can be used similar as enum_encoding. But also "binary", "gray", one_hot", "one_cold" and "auto"
- Attribute FSM_COMPLETE then no deadlock in FSM



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines



Synchronous Reset / Synchronous Moore FSM

BEGIN

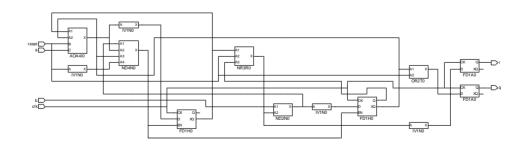
WAIT UNTIL clk='1';
IF reset='1' THEN state := idle; END IF;
CASE state IS



1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machine.

27

Asynchronous Reset / Synchronous Moore FSM (Cont.)

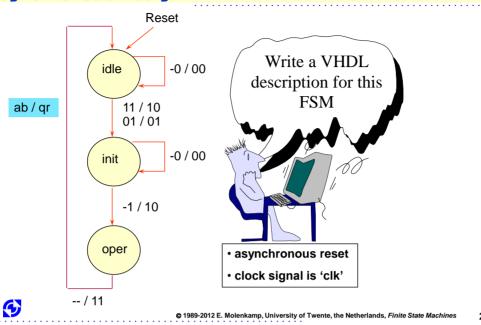




© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines



Synchronous Mealy



Synchronous Mealy - Two Processes

```
TYPE states IS (idle, init, oper);

SIGNAL state: states;

PROCESS(reset,clk)

BEGIN

IF reset='1' THEN

state <= idle;

ELSIF rising_edge(clk) THEN

CASE state IS

WHEN idle => IF b='1' THEN state<=init; END IF;

WHEN init => IF b='1' THEN state<=oper; END IF;

WHEN oper => state<=idle;

WHEN OTHERS => state <= idle;

END CASE;

END IF;

END PROCESS;
```



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines



Synchronous Mealy - Two Processes (Cont.)

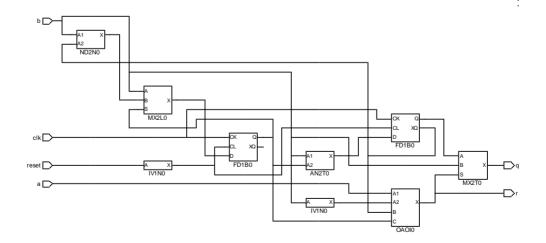
```
PROCESS(state,a,b)
BEGIN
CASE state IS
  WHEN idle => CASE std_logic_vector'(a & b) IS
                              => q<='1'; r<='0';
                 WHEN "11"
                 WHEN "01"
                               => q<='0'; r<='1';
                 WHEN OTHERS => q<='0'; r<='0';
               END CASE;
   WHEN init => IF b='1'
                 THEN q<='1'; r<='0';
                 ELSE q<='0'; r<='0';
                END IF;
   WHEN oper => q<='1'; r<='1';
   WHEN OTHERS => q<='0'; r<='0';
  END CASE:
 END PROCESS:
```



1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines

3

Synchronous Mealy - Two Processes (Cont.)





© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines



Synchronous Mealy - Three Processes

```
TYPE states IS (idle, init, oper);
SIGNAL inp_state, state : states;
PROCESS(clk,reset)
BEGIN
IF reset='1' THEN
state <= idle;
ELSIF clk='1' AND clk'EVENT THEN
state <= inp_state;
END IF;
END PROCESS;
```

```
next_state:PROCESS(state, a, b)
BEGIN
 CASE state IS
  WHEN idle => IF b='1' THEN
          inp state<=init;
         ELSE
          inp_state<=idle;
         END IF:
  WHEN init => IF b='1' THEN
          inp state<=oper;
         ELSE
          inp state <= init;
         END IF;
  WHEN oper => inp_state<=idle;
  WHEN OTHERS => inp_state <= idle;
 END CASE;
END PROCESS:
```



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines

33

34

Synchronous Mealy - Three Processes (Cont.)

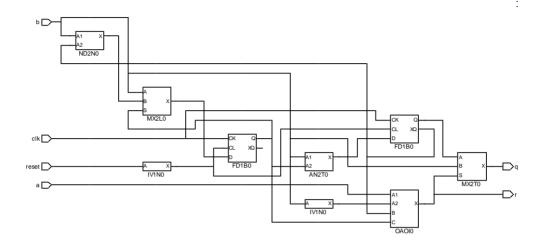
```
output_function:PROCESS(state,a,b)
BEGIN
 CASE state IS
  WHEN idle => CASE std logic vector'(a & b) IS
           WHEN "11" => q<='1'; r<='0';
           WHEN "01" => q<='0'; r<='1';
           WHEN OTHERS => q<='0'; r<='0';
          END CASE;
  WHEN init => IF b='1'
          THEN q<='1'; r<='0';
          ELSE q<='0'; r<='0';
          END IF:
  WHEN oper => q<='1'; r<='1';
  WHEN OTHERS => q<='0'; r<='0';
 END CASE:
END PROCESS;
```



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines



Synchronous Mealy - Three Processes (Cont.)





1989-2012 E. Molenkamp, University of Twente, the Netherlands, Finite State Machines