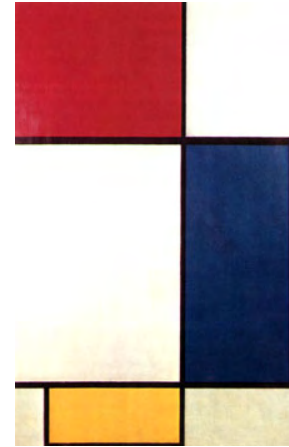


HW Inference rules: sequential logic



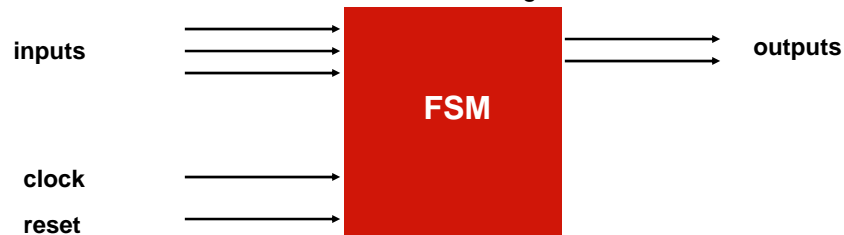
Electronica LB AA 2005-2006

HDL Synthesis & Verilog

Harris & Weste 2004, Thomas

Sequential behavior= Finite State Machines

- In the abstract, an FSM can be defined by:
 - » A set of **states** or actions that the system will perform
 - » The **inputs** to the FSM that will cause a sequence to occur
 - » The **outputs** that the states (and possibly, the inputs) produce
- There are also two special inputs
 - » A **clock event**, sometimes called the sequencing event, that causes the FSM to go from one state to the next
 - » A **reset event**, that causes the FSM to go to a known state



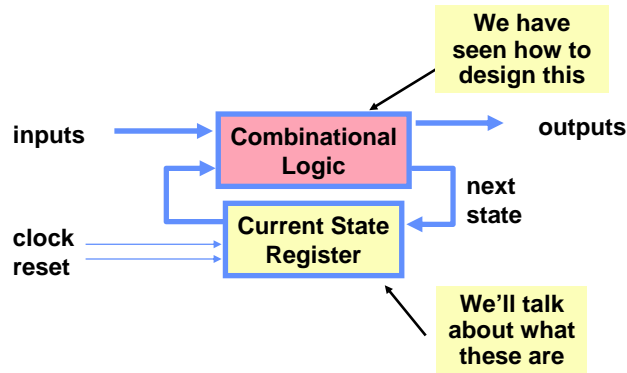
Electronica LB AA 2005-2006

HDL Synthesis & Verilog

Harris & Weste 2004, Thomas

Sequential circuits as FSMs

- The traditional FSM diagram



- Note: feedback is not essential (pipelines)

Electronica LB AA 2005-2006

HDL Synthesis & Verilog

Harris & Weste 2004, Thomas

Modeling state elements: D Flip-Flop

Current state (now)		Next state, after clock event
D	Q	Q+
0	0	0
0	1	0
1	0	1
1	1	1

Note that it doesn't matter what the current state (Q) is. The new state after the clock event will be the value on the D input.

```

module DFF
(output reg q,
input d, clk, reset);

always
@(posedge clk or negedge reset)
if (~reset)
q <= 0;
else q <= d;
endmodule
    
```

The change in q is synchronized to the clk input.

The reset is an asynchronous reset (q doesn't wait for the clk to change).

Electronica LB AA 2005-2006

HDL Synthesis & Verilog

Harris & Weste 2004, Thomas

Synchronous vs. Asynchronous reset

```

module flopr(clk, reset, d, q);
// synchronous reset
input clk;
input reset;
input [3:0] d;
output [3:0] q;
reg [3:0] q;
always @(posedge clk)
    if (reset) q <= 4'b0;
    else q <= d;
endmodule
    
```

Synchronous reset (only upon clock edge)

```

module flopenr(clk, reset, en, d, q);
// asynchronous reset
input clk;
input reset;
input en;
input [3:0] d;
output [3:0] q;
reg [3:0] q;
always @(posedge clk or posedge reset)
    if (reset) q <= 4'b0;
    else if (en) q <= d;
endmodule
    
```

Asynchronous reset (with enable)

Putting it all together: RTL modeling style

```

module FSM (x, z, clk, reset);
input      clk, reset, x;
output    z;
reg [1:2] q, d;
reg       z;

always
@ (posedge clk or negedge reset)
if (~reset)
    q <= 0;
else q <= d;

always @(x or q)
begin
d[1] = q[1] & x | q[2] & x;
d[2] = q[1] & x | ~q[2] & x;
z = q[1] & q[2];
end
endmodule
    
```

• Things to note

- » reg [1:2] — matches numbering in state assignment (Q2 is least significant bit in counting)
- » <= vs. =

The sequential part (the D flip flop)

The combinational logic part

next state

output

Non-blocking assignments and edge-triggered behavior

- Non blocking assign are perfect for edge-triggered primitives

```
module counter(clk, reset, q);
input clk;
input reset;
output [3:0] q;
reg [3:0] q;
// counter using always block
always @(posedge clk)
if (reset) q <= 4'b0;
else q <= q+1;
endmodule
```

Synchronous counter

```
module shiftreg(clk, sin, q);
input clk;
input sin;
output [3:0] q;
reg [3:0] q;
always @(posedge clk)
begin
q[0] <= sin;
q[1] <= q[0];
q[2] <= q[1];
q[3] <= q[2];
// even better: q <= {q[2:0], sin};
end
endmodule
```

Synchronous shift-reg

Electronica LB AA 2005-2006

HDL Synthesis & Verilog

Harris & Weste 2004, Thomas

FSMs with symbolic states

```
module divideby3FSM(clk, reset, out);
input clk;
input reset;
output out;
reg [1:0] state;
reg [1:0] nextstate;
// State Symbols
parameter S0 = 2'b00;
parameter S1 = 2'b01;
parameter S2 = 2'b10;
// State Register
always @(posedge clk or posedge reset)
if (reset) state <= S0;
else state <= nextstate;
// Continues...

// Next State Logic
always @(state)
case (state)
S0: nextstate = S1;
S1: nextstate = S2;
S2: nextstate = S0;
default: nextstate = S0;
endcase

// Output Logic
assign out = (state == S2);
endmodule
```

Next state logic (comb)

Output logic (Moore)

State update (2FFs)

Electronica LB AA 2005-2006

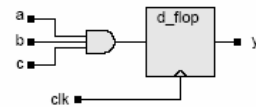
HDL Synthesis & Verilog

Harris & Weste 2004, Thomas

Registered logic

- **reg** with value assigned within a synchronous behavior will be registered

```
module reg_and (a, b, c, clk, y);
input a, b, c, clk;
output y;
reg y;
always @ (posedge clk)
begin
y <= a & b & c;
end
endmodule
```



Memories

- Storage array: register file
Note: this is an asynchronous memory

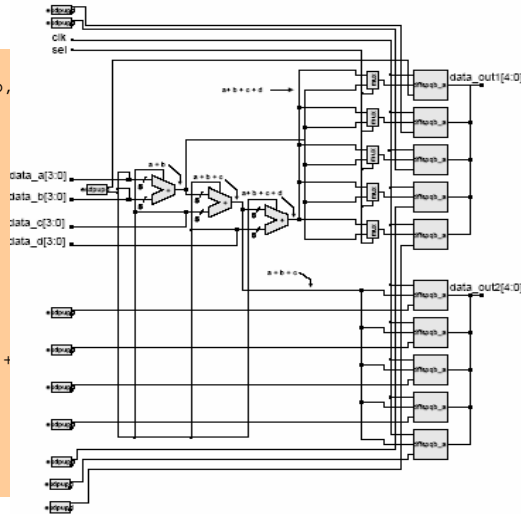
```
module sn54170 (data_in, wr_sel, rd_sel, wr_enb, rd_enb, data_out);
input wr_enb, rd_enb;
input [1:0] wr_sel, rd_sel;
input [3:0] data_in;
output [3:0] data_out;
reg [3:0] latched_data [3:0];
always @ (wr_enb or wr_sel or data_in) begin
if (!wr_enb) latched_data[wr_sel] = data_in;
end
assign data_out = (rd_enb) ? 4'b1111 : latched_data[rd_sel];
endmodule
```

Arithmetic circuits...

```

module expression_sub
(data_out1, data_out2, data_a, data_b,
data_c, data_d, sel, clk);
output [4:0] data_out1, data_out2;
input [3:0] data_a, data_b, data_c,
data_d;
input clk, sel;
reg [4:0] data_out1, data_out2;
always @ (posedge clk)
begin
data_out2 = data_a + data_b + data_c;
if (sel == 1'b0)
data_out1 = data_a + data_b + data_c +
data_d;
else
data_out1 = data_a + data_b;
end
endmodule

```



Electronica LB AA 2005-2006

HDL Synthesis & Verilog

Harris & Weste 2004, Thomas

How? ... A Few Definitions

- There are some restrictions on specification
 - » Input set of an “always” statement — the set of all variables that are used on the RHS of procedural assignments or in conditionals. i.e. anything “sourced”.
 - » Sensitivity list of an “always” statement — the set of all names that appear in the event (“@”) list.

```

module mux
(output reg f,
input sel, b, c);

always @ (sel or b or c)
begin
if (sel == 1)
f = b;
else
f = c;
end
endmodule

```

The elements in these lists are:

input: sel, b, c

sensitivity: sel, b, c

No coincidence here: a combinational circuit is sensitive to all its inputs

Electronica LB AA 2005-2006

HDL Synthesis & Verilog

Harris & Weste 2004, Thomas

More Definitions...

- » A control path of an “always” statement — a sequence of operations performed when executing the always statement
- » Combinational output of an “always” statement — a variable (or variables) assigned to in every control path

```
module mux
(output reg f,
input sel, b, c);

always @(sel or b or c)
begin
if (sel == 1)
f = b;
else
f = c;
end
endmodule
```

What are they here...

Control paths: through “then”
and “else” of if statement

Combinational output: f