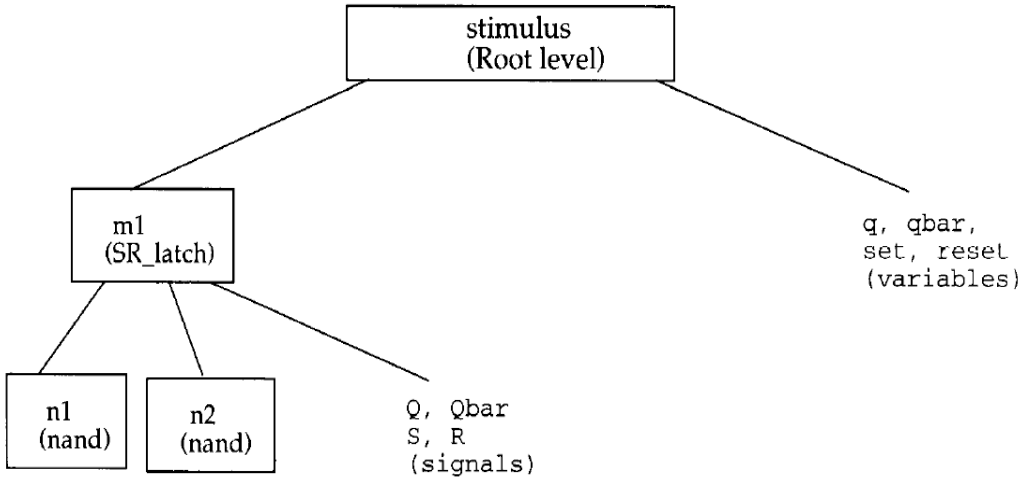
EN2911X: Reconfigurable Computing
Lecture 07: Verilog (4)

Prof. Sherief Reda
Division of Engineering, Brown University
Fall '09
http://scale.engin.brown.edu

BROWN

Hierarchical naming



- As described, every module instance, signal, or variable is identified with an identifier.
- Each identifier has a unique place in the design hierarchy.
- Hierarchical name referencing allows us to denote every identifier in the design hierarchy with a unique name.
- A hierarchical name is a list of identifiers separated by dots "." for each level of hierarchy
- Examples: `stimulus.q, stimulus.m1.Q, stimulus.m1.n2`

Named blocks

- Blocks can be given names
  - Local variables can be declared from the names block
  - Variables in a named block can be accessed by using hierarchical name referencing
  - Named blocks can be disabled

```
…
always
begin : block1
  integer i;
  i=1;
…
end
…
always
begin : block2
 integer j;
 j = block1.i^1;
end
```

Tasks and functions

- Often it is required to implement the same functionality at many times in a behavioral design.
- Verilog provides tasks and functions to break up large behavioral code into smaller pieces.
- Tasks and functions are included in the design hierarchy. Like named blocks, tasks and functions can be addressed by means of hierarchical names.
- Tasks have `input`, `output` and `inout` arguments
- Functions have `input` arguments
- Tasks and functions are included in the design hierarchy. Like named blocks, tasks or functions can be addressed by means of hierarchical names

Functions

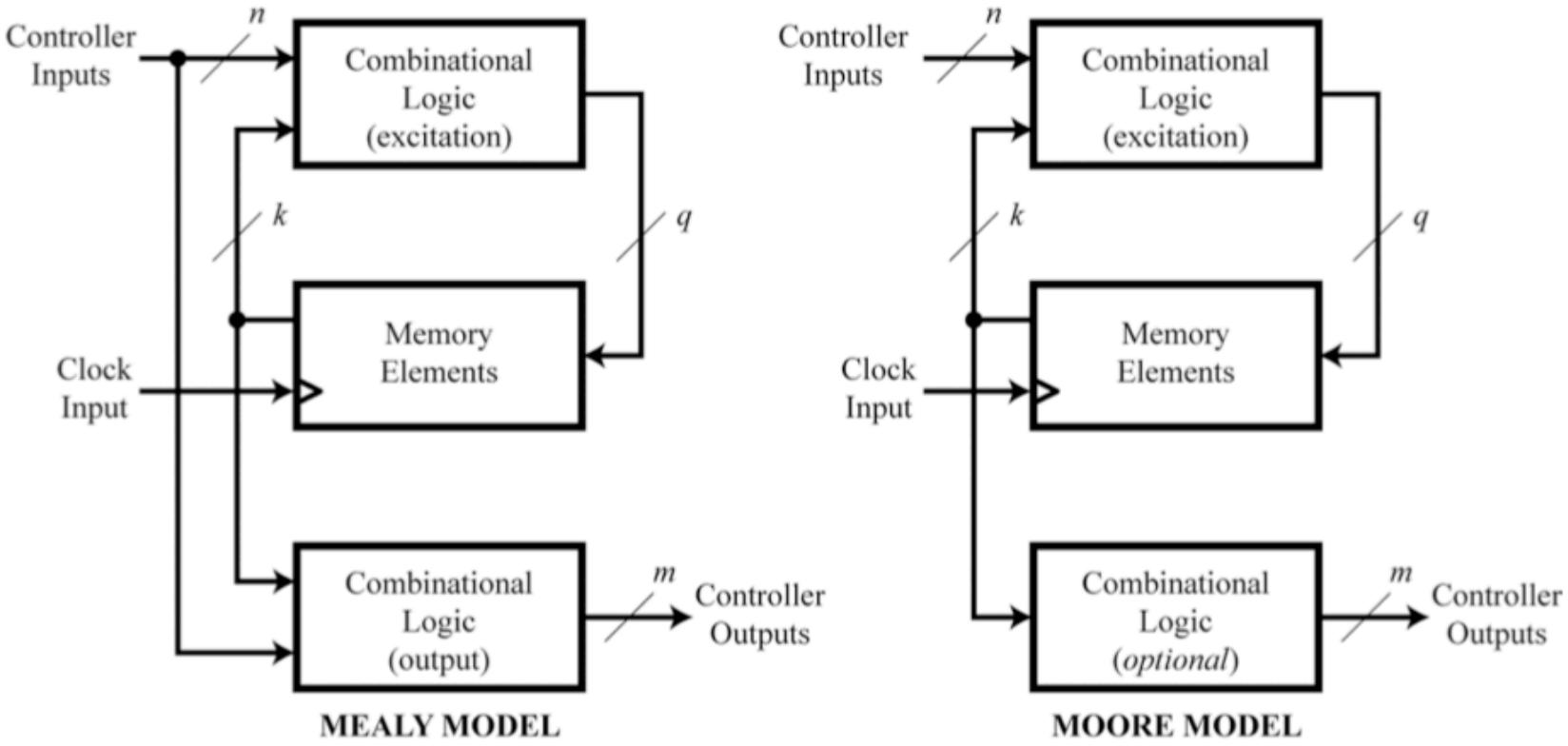
- Functions are typically used for combinational modeling (use for conversions and commonly used calculations.
- Need at least one input argument but cannot have `output` or `inout` arguments.
- The function is invoked by specifying function name and input arguments, and at the end execution, the return value is placed where the function was invoked
- Functions cannot invoke other tasks; they can only invoke other functions. Recursive functions are not synthesizable

```
module …
…
reg [31:0] parity;

always @(addr)
begin
 parity = calc_parity(addr);
end

// you can declare C sytle
function calc_parity;
input [31:0] address;
begin
  calc_parity = ^address;
end
endfunction
…
endmodule
```

Sequential circuit design

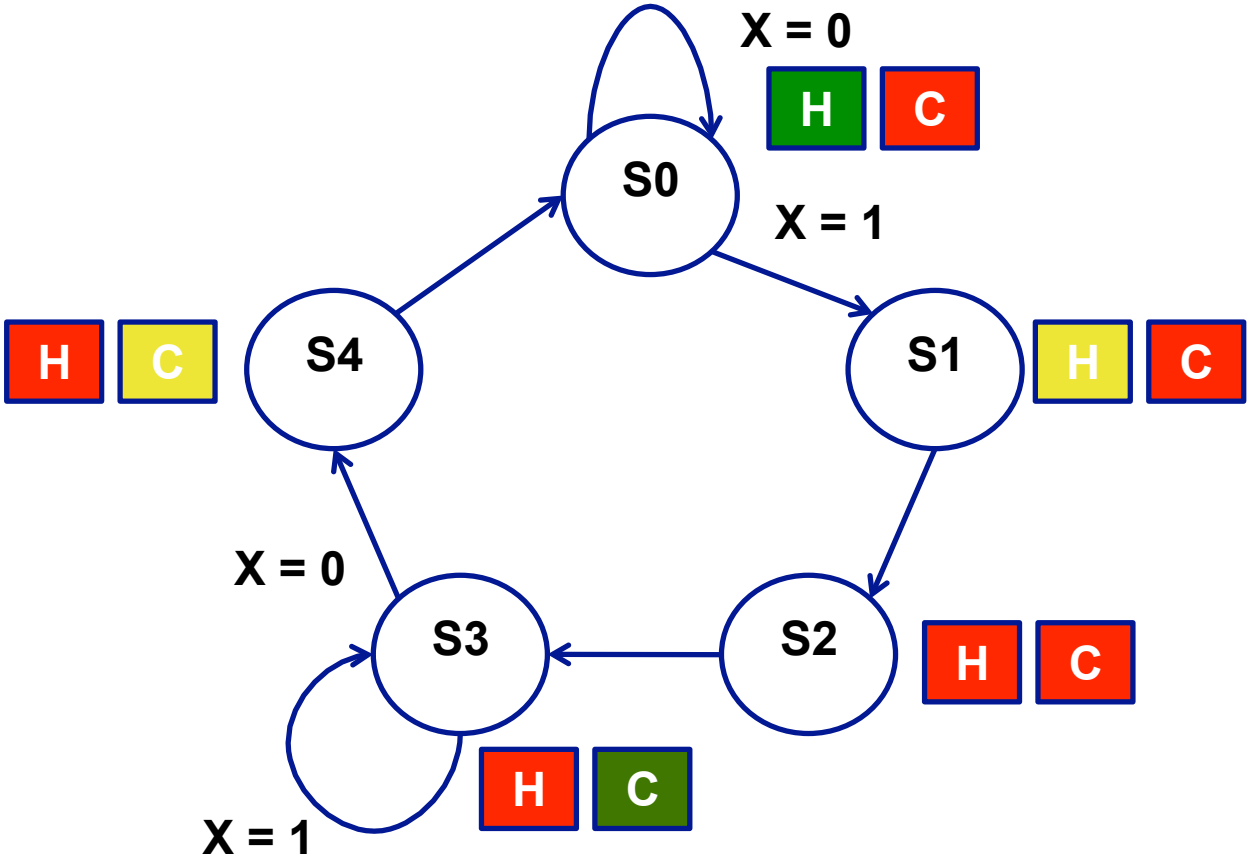


BROWN

Finite state machines (FSM) in Verilog

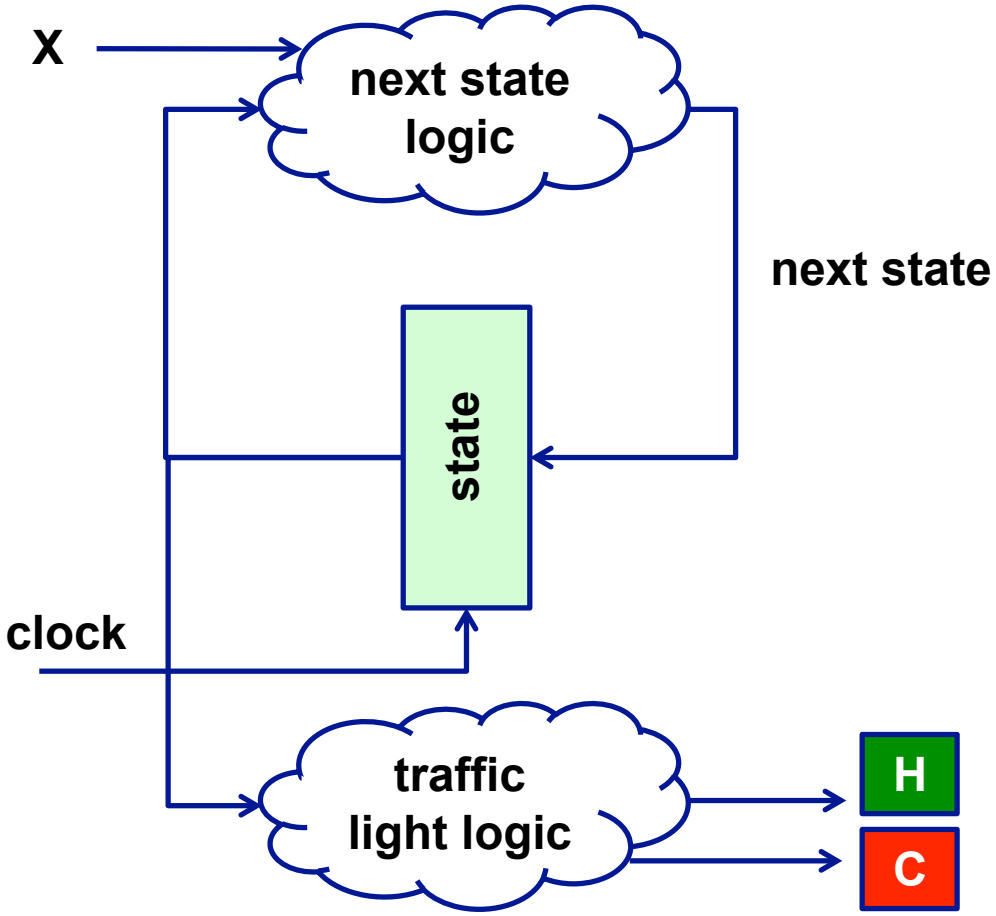
- Traffic signal for main highway gets highest priority, i.e., it is green by default.
- When cars from country road arrive at the intersection, the country traffic load should turn green after main highway turn yellow and then red.
- As soon as there are no cars on country road, country road should turn to yellow and then red.
- A car sensor (X) outputs 1 when there is a car waiting at country road; otherwise X outputs 0
- The two traffic signals can never be both green or yellow.

State diagram



BROWN

Hardware organization



BROWN

Module definition

```
module traffic(hwy, cntry, X, CLOCK_50, clear);

output reg [1:0] hwy, cntry;
input X, CLOCK_50, clear;

reg [2:0] state, next_state;
integer count, ticks, last_tick;

parameter RED = 2'b01, YELLOW = 2'b10, GREEN = 2'b11;
parameter S0 = 3'd0, S1 = 3'd1, S2 = 3'd2, S3 = 3'd3, S4 = 3'd4;

initial
begin
        ticks = 0;
        count = 0;
        state = S0;
        next_state = S0;
end
.
.
.
end
```

`always` statements inside the module

```
always @(posedge CLOCK_50)
        if (!clear) state <= S0;
        else state <= next_state;
```

```
always @(posedge CLOCK_50)
begin
        count <= count + 1;
        if (count == 50_000_000)
        begin
                count <= 0;
                ticks <= ticks +1;
        end
end
```

`always` statements inside the module

```
always @(state)
begin
        hwy = GREEN;
        cntry = RED;
        case(state)
                S0: ;
                S1: hwy = YELLOW;
                S2: hwy = RED;
                S3: begin
                        hwy = RED;
                        cntry = GREEN;
                        end
                S4: begin
                        hwy = RED;
                        cntry = YELLOW;
                        end
        endcase
end
```

`always` statements inside the module

```
always @(state or X)
begin
            case (state)
            S0: if(X == 1'b1)
            begin
                        next_state = S1;
                        last_tick = ticks;
            end
            else next_state = S0;
            S1: begin
                        if (ticks == last_tick+2) next_state = S2;
                        else next_state = S1;
            end
            S2: begin
                        if (ticks == last_tick+2) next_state = S3;
                        else next_state = S2;
            end
            S3: if(X)
            begin
                        next_state = S3;
                        last_tick = ticks;
            end
            else next_state = S4;
            S4: begin
                        if (ticks == last_tick+2) next_state = S0;
                        else next_state = S4;
            end
            default: next_state = S0;
            endcase
end
```