Finite State Machines

Dr. Paul D. Franzon
Outline

- Types of Finite State Machines
- Coding Template
- Controllers
- Reset

References
1. Smith & Franzon, 8.1, 8.2 (I suggest avoiding the implicit style)
2. Ciletti, Chapters 7.1
Finite State Machine Types

Finite State Machines can be classified by the following attributes:

- **Moore or Mealy type outputs**

  **Moore Outputs**
  Outputs depend solely on state vector (generally, a Moore FSM is the simplest to design)

  **Mealy Outputs**
  Outputs depend on inputs and state vector (only use if it is significantly smaller or faster)
… FSM Types

- **State Vector Encoding**
  - Minimal encoding
    - Minimum number of bits
  - Minimum, sequential encoding
    - Minimum number of bits and states in sequence
  - Gray encoding
    - state bit changes by only one bit between sequential states
  - One-hot encoding
    - one bit per state
    - usually gives fastest ‘next state’ logic
... FSM Types

- What is the encoding for the first 3 states in a 7 state FSM for each style?
… FSM Types

- **Resets:**
  - Reset usually occurs only on power-up and when someone hits the ‘reset’ button
  - **Asynchronous** reset:
    - FSM goes to reset state whenever reset occurs
  - **Synchronous** reset:
    - FSM goes to reset state on the next clock edge after reset occurs
  - Asynchronous reset leads to smaller flip-flops while synchronous reset is ‘safer’ (noise on the reset line is less likely to accidently cause a reset).
• **Fail-Safe Behavior:**
  - If the FSM enters an ‘illegal’ state due to noise is it guaranteed to then enter a legal state?
    - ‘Yes’ is generally desirable

• **Registered Outputs**
  - Sometimes useful to register FSM outputs – adds a stage of pipelining to the “control”
  - Or to interface with an asynchronous sub-system
Outline

- Types of Finite State Machines
- Coding Template
- Controllers
- Reset
Example - Drag Racing Lights

• At the start of a new race (‘car’), go through the Red-Yellow-Green sequence:

  Moore Machine:
  
  Nomenclature: inputs
  
  car?
  
  On states: red yellow green

  Mealy Machine:
  
  Nomenclature: inputs / outputs
  
  car? / red yellow green

©2011, Dr. Paul D. Franzon, www.ece.ncsu.edu/erl/faculty/paulf.html
module traffic_light_controller (clock, reset, car, red, yellow, green);

input clock;
input reset;
input car;
output red, yellow, green;

parameter [1:0] // synopsys enum states
S0 = 2'b00;
S1 = 2'b01;
S2 = 2'b10;
S3 = 2'b11;

reg [1:0] /* synopsys enum states */ current_state, next_state;// synopsys state

reg red, yellow, green;

/*------- Sequential Logic ----*/
always@(posedge clock or negedge reset)
if (!reset)   current_state <= S0;else  current_state <= next_state;

/* next state logic and output logic */
always@(current_state or car)
begin
  red = 0; yellow = 0; green = 0; /* defaults to prevent latches */
end

Next State Logic

Output Logic

©2011, Dr. Paul D. Franzon, www.ece.ncsu.edu/erl/faculty/paulf.html
1. Code each FSM by itself in one module.
2. Separate Sequential and Combinational Logic
3. Is this reset Synchronous or Asynchronous?  
   Asynchronous
4. Note use of Synthesis directives:
   
   ```verilog
   //synopsys enum states and //synopsys state_vector current_state tell Synopsys what the state vector is.
   You can optionally use Synopsys FSM optimization procedures
   ```
Mealy Alternative

Exercise: Code Mealy

```verilog
ealways@(current_state or car) begin
  red = 0; yellow = 0; green = 0;
  case (current_state) // synopsys f1
    S0:
      S1:
      S2 :
    default : next_state = S0;
  endcase
end
```

Mealy Machine:

Nomenclature: inputs / outputs
  car? / red yellow green
Outline

- Types of Finite State Machines
- Coding Template
- Controllers
- Reset
Controllers

- A useful design approach is to clearly separate the datapath from the controller

- Controller styles
  - FSMs
  - Hierarchy of FSMs
  - Counters
Hierarchical FSMs

- Most chips are too large to be controlled via one FSM
- Large FSMs (>50-100 states +/-) tend to be too slow
- Might need a hierarchical controller:
Reset

• Reset is a global signal that **the designer can not modify**
• It is generally asserted on power up or a "hard" reset
• It is used to get the machine into a “known” state
• Thus it must be distributed to
  • All FSMs
  • Selected counters
  • Selected status registers
Review Questions

What is the role of reset?

What coding style is used for FSMs?