Sharif University of Technology
Department of Computer Engineering

Digital System Design
Finite State Machines (FSMs)

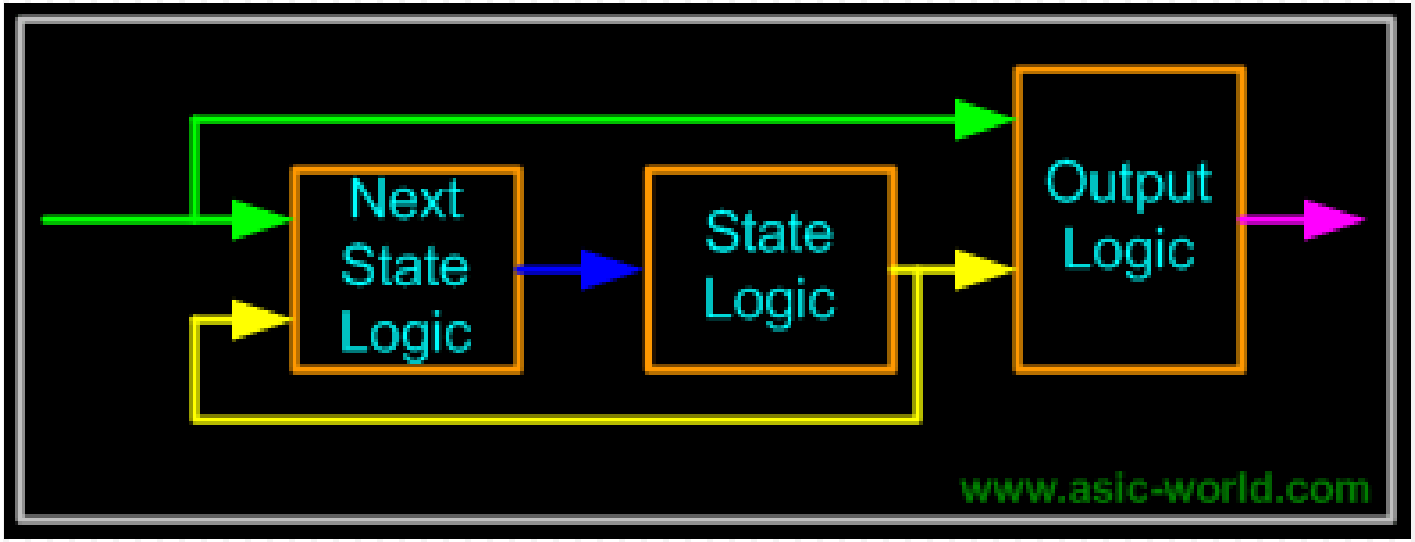
Siavash Bayat-Sarmadi

Introduction

- FSM consists of
  - Combinational logic
    - Deciding the next state of the FSM
  - Sequential logic
    - Store the current state of the FSM
  - Output
    - Mixture of comb. and seq. logic

FSM Types

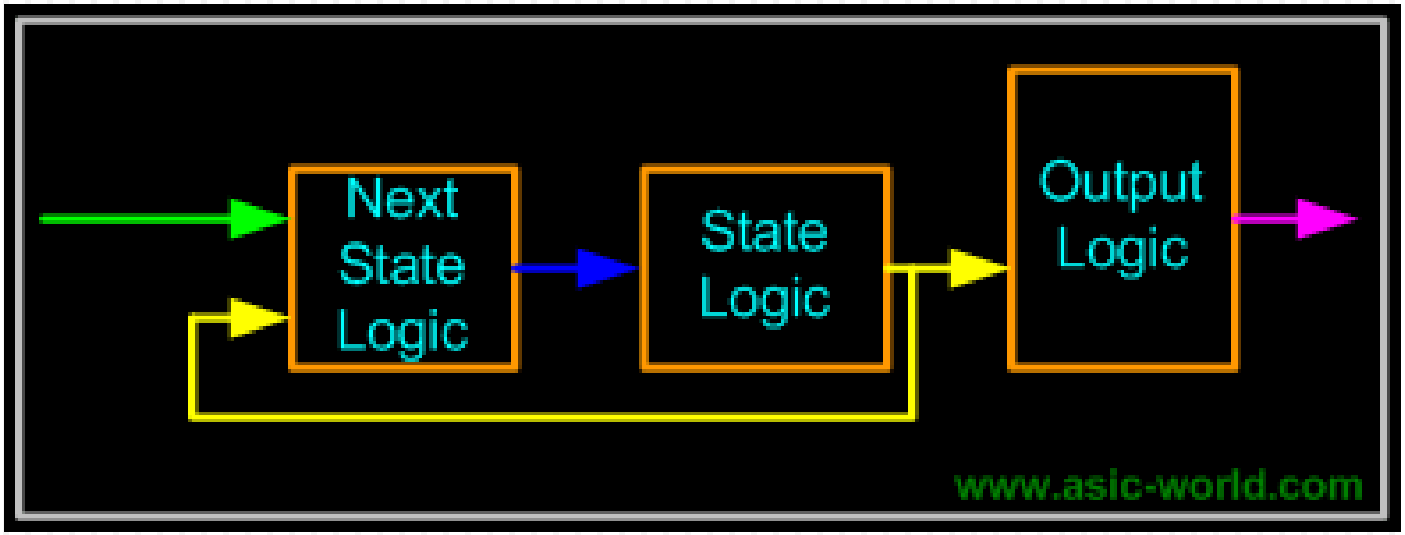
- Mealy



  - output = f (curr_state, curr_input);

FSM Types (Cont'd.)

- Moore



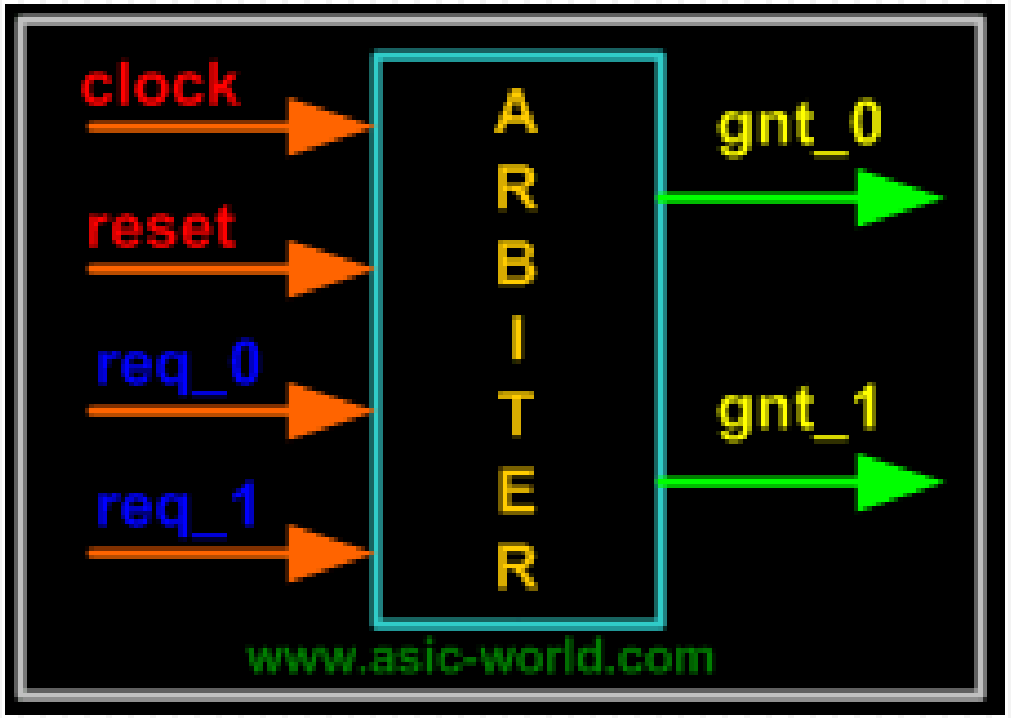
  - output = f (curr_state);

State Encoding Styles

- Binary Encoding
  - 000, 001, 010, …
- Gray Encoding
  - 000, 001, 011, …
- One Hot
  - Just one bit is high and rest are low
  - 00001, 00010, 00100, … (for 5 states)
- One Cold
  - Just one bit is high and rest are low
  - 11110, 11101, 11011, … (for 5 states)

Example

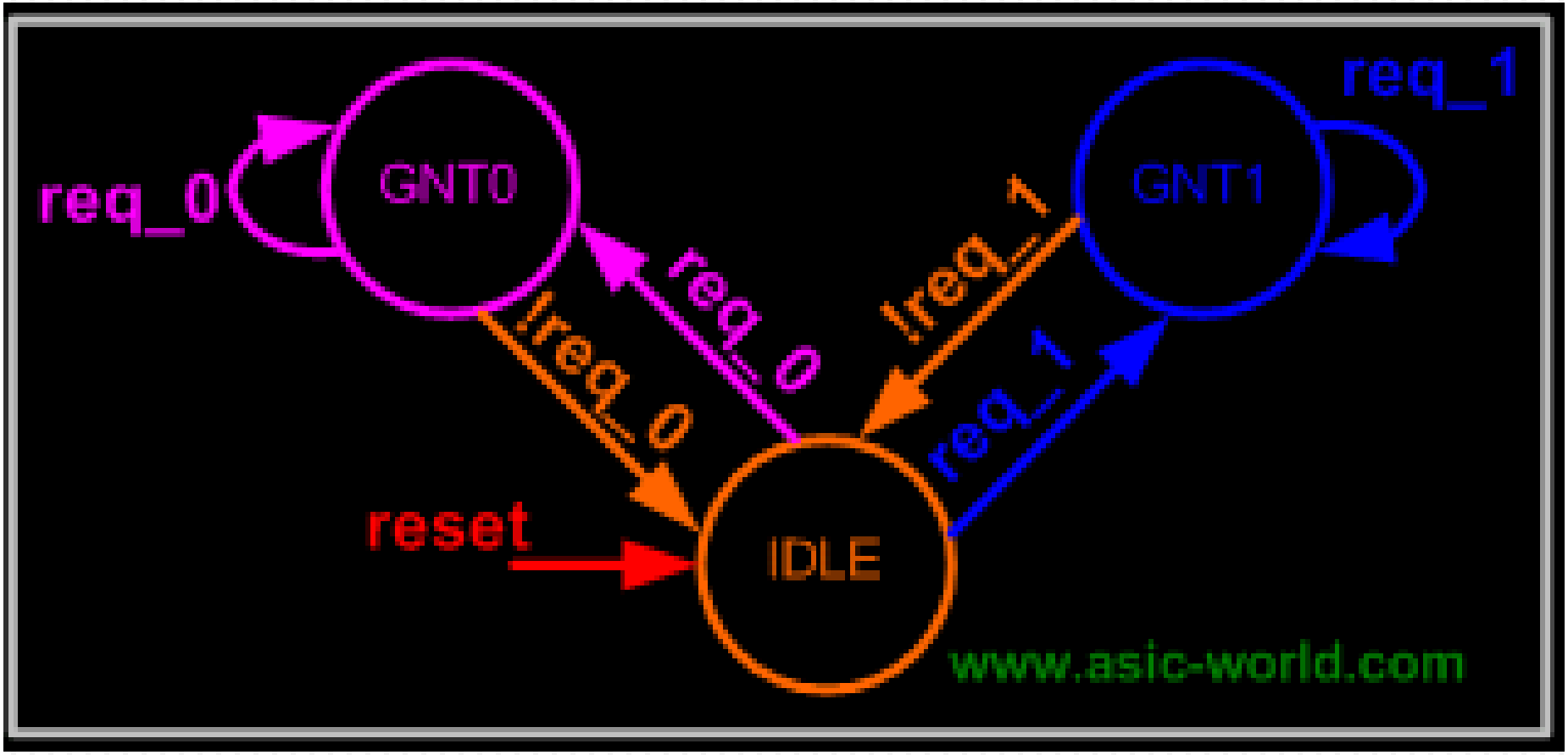
- Simple arbiter
  - 2 request inputs
  - 2 grant outputs



  - Description:
    - req_0=>gnt_0
    - req_1=>gnt_1
    - req_0, req_1 => gnt_0 (higher priority once no req)

Example (Cont'd.)

- Translate description into FSM diagram



Example (Cont'd.)

- Translate description into FSM diagram
- 3 states
  - IDLE
    - Waiting for a request
    - Both outputs low
    - Defaults state after reset/fault recovery
  - GNT0
    - req_0 is asserted (or also re-asserted)
    - When req_0 is de-asserted, FSM returns to IDLE
  - GNT1
    - req_1 is asserted (or also re-asserted)
    - When req_1 is de-asserted, FSM returns to IDLE

Example (Cont'd.)


```
module fsm_using_function (
clock       ,
reset       , // Active high, syn reset
req_0       ,
req_1       ,
gnt_0       ,
gnt_1
);
input    clock,reset,req_0,req_1;
output   gnt_0,gnt_1;
reg      gnt_0,gnt_1;
```

Example (Cont'd.)


```
parameter SIZE = 3;
parameter IDLE  = 3'b001,
          GNT0 = 3'b010,
          GNT1 = 3'b100 ;

// Seq part of the FSM
reg   [SIZE-1:0] state;
// Comb part of FSM
wire  [SIZE-1:0] next_state;
```

Example (Cont'd.)


```
assign next_state =
    fsm_function(state, req_0, req_1);

function [SIZE-1:0] fsm_function;
  input  [SIZE-1:0] state;
  input req_0;
  input req_1;
```

Example (Cont'd.)


```
case(state)
 IDLE:
  if (req_0 == 1'b1)
      fsm_function = GNT0;
  else if (req_1 == 1'b1)
      fsm_function= GNT1;
  else
      fsm_function = IDLE;

 GNT0:
  if (req_0 == 1'b1)
      fsm_function = GNT0;
  else
      fsm_function = IDLE;
```

Example (Cont'd.)


```
    GNT1:
     if (req_1 == 1'b1)
          fsm_function = GNT1;
     else
          fsm_function = IDLE;

    default:
          fsm_function = IDLE;
  endcase
endfunction
```

Example (Cont'd.)


```
//------Seq Logic------
always @ (posedge clock)
begin : FSM_SEQ
   if (reset == 1'b1) begin
      state <= #1 IDLE;
   end else begin
      state <= #1 next_state;
   end
end
```

Example (Cont'd.)


```
//----Output Logic----
always @ (posedge clock)
begin : OUTPUT_LOGIC
    if (reset == 1'b1) begin
        gnt_0 <= #1 1'b0;
        gnt_1 <= #1 1'b0;
    end
    else begin
        case(state)
            IDLE: begin
                gnt_0 <= #1 1'b0;
                gnt_1 <= #1 1'b0;
            end
```

Example (Cont'd.)


```
        GNT0: begin
            gnt_0 <= #1 1'b1;
            gnt_1 <= #1 1'b0;
        end
        GNT1 : begin
            gnt_0 <= #1 1'b0;
            gnt_1 <= #1 1'b1;
        end
        default : begin
            gnt_0 <= #1 1'b0;
            gnt_1 <= #1 1'b0;
        end
      endcase
    end
end //End Of Block OUTPUT_LOGIC
endmodule
```

Example (Cont'd.)

- Next state logic using always

```
always @ (state or req_0 or req_1)
begin : FSM_COMBO
 next_state = IDLE;
 case(state)
    IDLE :
     if (req_0 == 1'b1)
          next_state = GNT0;
     else if (req_1 == 1'b1)
          next_state= GNT1;
     else
          next_state = IDLE;
```

Example (Cont'd.)


```
    GNT0 :
     if (req_0 == 1'b1)
         next_state = GNT0;
     else
         next_state = IDLE;

    GNT1 :
     if (req_1 == 1'b1)
         next_state = GNT1;
     else
         next_state = IDLE;
    default: next_state = IDLE;
   endcase
end
```

Example (Cont'd.)

- A single always for next state and output logic

```
always @ (posedge clock)
begin : FSM
    if (reset == 1'b1)
    begin
        state <= #1 IDLE;
        gnt_0 <= 0;
        gnt_1 <= 0;
    end
```

Example (Cont'd.)


```
else
 case(state)
    IDLE:
     if (req_0 == 1'b1)
     begin
          state <= #1 GNT0;
          gnt_0 <= 1;
     end else if (req_1 == 1'b1)
     begin
          gnt_1 <= 1;
          state <= #1 GNT1;
     end else
     begin
          state <= #1 IDLE;
     end
```

Example (Cont'd.)


```
        GNT0: if (req_0 == 1'b1)
                state <= #1 GNT0;
              else begin
                gnt_0 <= 0;
                state <= #1 IDLE;
              end
        GNT1: if (req_1 == 1'b1)
                state <= #1 GNT1;
              end else begin
                gnt_1 <= 0;
                state <= #1 IDLE;
              end
        default : state <= #1 IDLE;
    endcase
end
```

Misc.

- gnt_0 and gnt_1 can be defined as wires.
- Better coding style
  - Separating control path from data path
- How can we have a latch synthesized in combo logic?