CONTROL CIRCUITS

Control in Digital Systems

- Three primary components of digital systems
  - Datapath (does the work)
  - Control (manager)
  - Memory (storage)
Control in Digital Systems

- Control blocks in chips
  - Typically small amount of HW
  - Typically substantial verilog code
  - Can be complex
    - Many opportunities for bugs
    - May be impossible to test all states and transitions
    - Often the focus of testing by Verification Engineers

Control with Finite State Machines

- Moore type FSM
  - outputs depend only on the state
- Mealy type FSM
  - outputs depend on state and inputs
Sequential Logic

- Combinational circuits’ outputs are a function of the circuit’s inputs and a time delay
- Sequential circuits’ outputs are a function of the circuit’s inputs, previous circuit state, and a time delay

Writing Verilog for State Machines

- Design process
  - Think
  - Draw state diagrams if helpful
  - Draw block diagrams if helpful
  - Pick signal names
  - Think
  - Then...
    - Write verilog
    - Test it
Verilog Signal Names

- Good to have conventions for signal names
- \( c_* \) – input to a register (e.g., \( c_{sum} \))
- \( r_* \) – output of a register (e.g., \( r_{sum} \))
- Ex:
  - \( c_{state} \) → \( state \)
  - \( indata \) → \( r_{indata} \)

State in FSMs

- Clear code = bugs less likely
- Reduce the amount of state if possible (clarity)
  - Ex: Maybe better to have one global state machine instead of two separate ones
- Increase the amount of state if helpful (clarity)
  - Ex: Instantiate a separate counter to count events
Control Block Example

- Four states
  - IDLE
  - PREP
    - Do something for 10 cycles
  - JOB1
    - Do something for 5 cycles
  - JOB2
    - Do something for 20 cycles
- Reset at any time returns control to IDLE

Control Block Example

- State registers
  - Choose two bits
- Counter(s)
  - Choose one five bit counter since states are independent and counter can be shared
  - Count down may be best (simpler comparator)
- Safest to keep registers (flip-flops) separate from state machine logic
- Sometimes cleaner to define states with
  parameter IDLE = 2'h0;
`define vs. parameter

- Two methods
  - **parameter**
    - Convenient method to use text to represent a number
    - Local to a module
    - Usage
      ```
      parameter HALT = 4'b0101;
      ...
      if (inst == HALT) begin   // no “back tick”
      ```
  - **define** macro
    - Global text macro substitution using a compiler directive
    - Best when helpful to put all definitions in a global file
    - Usage
      ```
      `define HALT 4'b0101
      ...
      if (inst == `HALT) begin   // requires “back tick”
      ```

Example State Machine

- State diagram

```
reset

IDLE

PREP

JOB1

JOB2

x <= 5

x > 5

go
```
Example State Machine
(untested, could have bugs—especially syntax or off-by-one-cycle bugs)

parameter IDLE = 2'h0;  // constants in hex notation
parameter PREP = 2'h1;
parameter JOB1 = 2'h2;
parameter JOB2 = 2'h3;

reg [1:0] state, c_state;  // declare both FF regs
reg [4:0] count, c_count;  // and comb. logic regs

// Combinational logic for state machine
always @(state or count or go or x or reset) begin
  // defaults (place first)
  c_state = state;             // default same state
  c_count = count – 5'b00001;  // default count down

  // main state machine logic
  case (state) begin
    IDLE: begin
      if (go) begin
        c_state = PREP;            // default same state
        c_count = 5'd10;           // constant in decimal
      end
      else begin
        c_count = 0;              // default count down
      end
    end
    PREP: begin
      if (count == 5'b00000) begin
        if (x <= 5) begin         // goto JOB1
          c_state = JOB1;
          c_count = 5'd05;
        end else begin           // goto JOB2
          c_state = JOB2;
          c_count = 5'd20;
        end
      end
    end
    JOB1: begin
      if (count == 5'b00000) begin
        c_state = IDLE;          // count will underflow to -1 but it is ok
      end
    end
    JOB2: begin
      if (count == 5'b00000) begin
        c_state = IDLE;          // count will underflow to -1 but it is ok
      end
    end
    default: begin  // good practice, but not used here
      c_state = 2'bxx;         // better for testing
      c_state = IDLE;          // another option
    endcase
  end

  // reset logic (place last to override other logic)
  if (reset) begin
    c_state = IDLE;          // another option
    c_count = 5'b00000;
  end
end // end of always block

// Instantiates registers (flip-flops)
always @(posedge clk) begin
  state <= #1 c_state;
  count <= #1 c_count;
end

B. Baas, © 2011  EEC 281  106