Verilog 2 - Design Examples

6.375 Complex Digital Systems
Arvind
February 9, 2009

Courtesy of Arvind http://csg.csail.mit.edu/6.375/
Verilog Design Examples

Greatest Common Divisor

Courtesy of Arvind http://csg.csail.mit.edu/6.375/
GCD in C

```c
int GCD(int inA, int inB)
{
    int done = 0;
    int A = inA;
    int B = inB;
    while (!done)
    {
        if (A < B)
        {
            swap = A;
            A = B;
            B = swap;
        }
        else if (B != 0)
            A = A - B;
        else
            done = 1;
    }
    return A;
}
```

What does the RTL implementation need?

- **inputs**
- **State**
- **iteration**
- **Less-Than Comparator**
- **Swap**
- **Equal Comparator**
- **Subtractor**
- **Termination control**
- **output**

Courtesy of Arvind http://csg.csail.mit.edu/6.375/
Step 1: Design an appropriate port interface

- **input_available**
- **idle**
- **operand_A**
- **operand_B**
- **clk**
- **reset**
- **result_rdy**
- **result_taken**
- **result_data**
Step 2: Design a datapath which has the functional units

A = inA; B = inB;
while (!done) begin
  if (A < B)
    swap = A;
    A = B;
    B = swap;
  else if (B != 0)
    A = A - B;
  else
    done = 1;
End
Y = A;
Step 3: Add the control unit to sequence the datapath

Control unit should be designed to be either busy or waiting for input or waiting for output to be picked up.

\[
A = \text{inA}; \quad B = \text{inB};
\]

\[
\text{while} \ ( \neg \text{done} ) \begin{align*}
\text{if} \ ( A < B ) & \quad \text{swap} = A; \\
& \quad A = B; \\
& \quad B = \text{swap}; \\
\text{else if} \ ( B \neq 0 ) & \quad A = A - B; \\
\text{else} & \quad \text{done} = 1;
\end{align*}
\]

End

\[
Y = A;
\]
module GCDdatapath#( parameter W = 16 )
  ( input clk,

    // Data signals
    input  [W-1:0] operand_A,
    input  [W-1:0] operand_B,
    output [W-1:0] result_data,

    // Control signals (ctrl->dpath)
    input   A_en,
    input   B_en,
    input   [1:0] A_sel,
    input   B_sel,

    // Control signals (dpath->ctrl)
    output  B_zero,
    output  A_lt_B
  );

Courtesy of Arvind http://csg.csail.mit.edu/6.375/
Connect the modules

```verilog
wire [W-1:0] B;
wire [W-1:0] sub_out;
wire [W-1:0] A_out;

vcMux3#(W) A_mux
   ( .in0 (operand_A),
     .in1 (B),
     .in2 (sub_out),
     .sel (A_sel),
     .out (A_out)   );

wire [W-1:0] A;

vcEDFF_pf#(W) A_pf
   ( .clk (clk),
     .en_p (A_en),
     .d_p  (A_out),
     .q_np (A)      );
```

---

February 9, 2009

Courtesy of Arvind http://csg.csail.mit.edu/6.375/
Connect the modules ...

wire [W-1:0] B;
wire [W-1:0] sub_out;
wire [W-1:0] A_out;

vcMux3#(W) A_mux
   ( .in0 (operand_A),
     .in1 (B),
     .in2 (sub_out),
     .sel (A_sel),
     .out (A_out)   );

wire [W-1:0] A;
vcEDFF_pf#(W) A_pf
   ( .clk  (clk),
     .en_p (A_en),
     .d_p  (A_out),
     .q_np (A)     );

wire [W-1:0] B_out;
vcMux2#(W) B_mux
   ( .in0 (operand_B),
     .in1 (A),
     .sel (B_sel),
     .out (B_out)   );

vcEDFF_pf#(W) B_pf
   ( .clk  (clk),
     .en_p (B_en),
     .d_p  (B_out),
     .q_np (B)     );

assign B_zero   = (B==0);
assign A_lt_B   = (A < B);
assign sub_out  = A - B;
assign result_data = A;

Using explicit state helps eliminate issues with non-blocking assignments

Continuous assignment combinational logic is fine
Control unit requires a state machine for valid/ready signals

- **WAIT**
  - Waiting for new input operands
  - input_available

- **CALC**
  - Swapping and subtracting
  - \( B = 0 \)

- **DONE**
  - Waiting for consumer to take the result
  - result_taken

Diagram:
- **reset**
- **WAIT**
- **CALC**
- **DONE**

Implementing the control logic FSM in Verilog

```verilog
localparam WAIT = 2'd0;
localparam CALC = 2'd1;
localparam DONE = 2'd2;
reg [1:0] state_next;
wire [1:0] state;

always @(posedge clk)
begin
    state <= state_next;
end
```

Localparams are not really parameters at all. They are scoped constants.
Control signals for the FSM

```vhdl
reg [6:0] cs;
always @(*)
begin
    //Default control signals
    A_sel    = A_SEL_X;
    A_en     = 1'b0;
    B_sel    = B_SEL_X;
    B_en     = 1'b0;
    input_available = 1'b0;
    result_rdy   = 1'b0;
    case ( state )
        WAIT :     
            ...
        CALC :     
            ...
        DONE :     
            ...
    endcase
end

WAIT: begin
    A_sel    = A_SEL_IN;
    A_en     = 1'b1;
    B_sel    = B_SEL_IN;
    B_en     = 1'b1;
    input_available = 1'b1;
end

CALC: if ( A_lt_B )
    A_sel = A_SEL_B;
    A_en    = 1'b1;
    B_sel = B_SEL_A;
    B_en    = 1'b1;
    else if ( !B_zero )
    A_sel = A_SEL_SUB;
    A_en    = 1'b1;
end

DONE: result_rdy = 1'b1;
```

February 9, 2009

Courtesy of Arvind http://csg.csail.mit.edu/6.375/

L03-12
always @(*)
begin
    // Default is to stay in the same state
    state_next = state;

case ( state )
    WAIT :
        if ( input_available )
            state_next = CALC;
    CALC :
        if ( B_zero )
            state_next = DONE;
    DONE :
        if ( result_taken )
            state_next = WAIT;
endcase
end

FSM state transitions

Courtesy of Arvind http://csg.csail.mit.edu/6.375/
RTL test harness requires proper handling of the ready/valid signals