Cristian’s Rules for Good VHDL

Prof. Stephen A. Edwards
sedwards@cs.columbia.edu

Columbia University
Spring 2006
List all process inputs in the sensitivity list.

```vhdl
process (current_state, long)
begin
    if (reset = '1') then
        next_state <= HG;
        start_timer <= '1';
    else
        case current_state is
            when HG =>
                farm_yellow <= '0';
                if (cars = '1' and long = '1') then
                    next_state <= HY;
                else
                    next_state <= HG;
                end if;
            when HY =>
                farm_yellow <= '0';
                if (short = '1') then
                    next_state <= FG;
                else
                    next_state <= HY;
                end if;
        end case;
    end if;
end process;
```

```vhdl
process (current_state, reset, cars, short, long)
begin
    if (reset = '1') then
        next_state <= HG;
        start_timer <= '1';
    else
        case current_state is
            when HG =>
                farm_yellow <= '0';
                if (cars = '1' and long = '1') then
                    next_state <= HY;
                else
                    next_state <= HG;
                end if;
            when HY =>
                farm_yellow <= '0';
                if (short = '1') then
                    next_state <= FG;
                else
                    next_state <= HY;
                end if;
        end case;
    end if;
end process;
```
Always assign all outputs

Synthesis infers level-sensitive latches otherwise.

```vhdl
process (current_state, input)
begin
    case current_state is
        when S1 =>
            if (input = '1') then
                output <= '0';
            end if;
        when S2 =>
            output <= '1';
    end case;
end process;
```

```vhdl
process (current_state, input)
begin
    case current_state is
        when S1 =>
            if (input = '1') then
                output <= '0';
            else
                output <= '1';
            end if;
        when S2 =>
            output <= '1';
    end case;
end process;
```
Accidental Level-Sensitive Latches

Section from .mrp when you have latches

Design Summary
--------------
Number of errors: 0
Number of warnings: 0
Logic Utilization:
  Total Number Slice Registers: 18 out of 6,144
  Number used as Flip Flops: 16
  Number used as Latches: 2
  Number of 4 input LUTs: 23 out of 6,144

Section from .mrp with no latches

Design Summary
--------------
Number of errors: 0
Number of warnings: 0
Logic Utilization:
  Number of Slice Flip Flops: 31 out of 6,144
  Number of 4 input LUTs: 16 out of 6,144
“Default” values are convenient

-- OK

process (current_state, input) begin
  case current_state is
    when S1 =>
      if (input = '1') then
        output <= '0';
      else
        output <= '1';
      end if;
    when S2 =>
      output <= '1';
  end case;
end process;

-- Better

process (current_state, input) begin
  output <= '1';
  case current_state is
    when S1 =>
      if (input = '1') then
        output <= '0';
      end if;
    when S2 =>
      output <= '1';
  end case;
end process;
Better to use an enumeration to encode states:

```vhdl
type mystate is (START, RUN, IDLE, ZAPHOD);
signal cst : mystate;
signal nxst : mystate;

process(cst)
begin
    case cst is
    when START => ...
    when RUN => ...
    when IDLE => ...
    end case;
end process;
```

Running this produces a helpful error:

```
Compiling vhdl file "/home/cristi/cs4840/lab4/main.vhd" in Library work.
Entity <system> compiled.
Enumerated value zaphod is missing in case.
-->
Always include the clock. Include reset if asynchronous, and nothing else.

```
process (Clk, D)
begin
  if (Clk'event and Clk = '1') then
    Q <= D;
  end if;
end process;

process (Clk, D)
begin
  if (reset = '1') then
    Q <= '0';
  else
    if (Clk'event and Clk = '1') then
      Q <= D;
    end if;
  end if;
end process;

process (Clk)
begin
  if (Clk'event and Clk = '1') then
    Q <= D;
  end if;
end process;

process (Clk, reset)
begin
  if (reset = '1') then
    Q <= '0';
  else
    if (Clk'event and Clk = '1') then
      Q <= D;
    end if;
  end if;
end process;
```
Only use asynchronous reset when there is one global signal from outside.

-- OK if Reset is from outside
process (Clk, Reset)
begin
  if (Reset = '1') then
    Q <= '0';
  else
    if (Clk'event and Clk = '1') then
      Q <= D;
    end if;
  end if;
end process;

-- Better
process (Clk)
begin
  if (Clk'event and Clk = '1') then
    if (Reset = '1') then
      Q <= '0';
    else
      if (Clk'event and Clk = '1') then
        Q <= D;
      end if;
    end if;
  end if;
end process;
Simulation: One version only

- Never assume signals from the test bench that are not there on the board.
- It is hard enough to make simulation match the design; do not make it any harder.
- If you must slow down hardware, carefully generate a slower clock and only use that clock globally.
Don’t Add Fictitious I/O

Ports on the topmost entity must correspond to FPGA I/O pins and must be defined in the .ucf file.

entity system is
  port (    entity system is
  clk : in std_logic;    clk : in std_logic;
  PB_D0, PB_D1, PB_D2, PB_D3,    PB_D0, PB_D1, PB_D2, PB_D3,
  PB_D4, PB_D5, PB_D6, PB_D7,    PB_D4, PB_D5, PB_D6, PB_D7,
  PB_D8, PB_D9, PB_D10, PB_D11,    PB_D4, PB_D5, PB_D6, PB_D7
  PB_D12, PB_D13, PB_D14, PB_D15 end system;
    : out std_logic
);    : out std_logic
end system;

UCF file:

net CLK loc="p77";
net PB_D0 loc="p153";
net PB_D1 loc="p145";
net PB_D2 loc="p141";
net PB_D3 loc="p135";
net PB_D4 loc="p126";
net PB_D5 loc="p120";
net PB_D6 loc="p116";
net PB_D7 loc="p108";
Stick to the Synchronous Model

- Exactly one value per signal per clock cycle
- Do not generate asynchronous reset signals; only use them if they are external
- Edge-triggered flip-flops only. No level-sensitive logic.
- Do not generate clock signals. Use multiplexers to create “load enable” signals on flip-flops.