Verilog Coding: What To Do With All These Charts?

What to do with all these charts?

After getting all the diagrams ready, you may wonder: why did I even draft all these? Is it going to be useful in any way?

The answer is yes. Ultimately, our goal is to write Verilog code to make the FPGA device function the same as our designed system. If you know the drill, it is almost like a copy-and-paste with our drafted diagrams. Now let us see how:

1. Modulize each component. Simply do this once and the same component can be reused in the current or later projects.

For example, we can create the 2-to-1 Mux module like this:

```
module mux2(sel, d1, d0, y);
  input sel;
  input [31:0] d1;
  input [31:0] d0;
  output wire [31:0] y;

  assign y = (sel == 0)? d0: d1;
endmodule
```

Or we can create the multiplier module like this:

```
module multiplier(
    input  wire [31:0] A,
    input  wire [31:0] B,
    output wire [31:0] Y
  );

  assign Y = A * B;
endmodule
```

2. Connecting the components in a datapath module according to your datapath design.

For example, we can connect the multiplier with the Mux in the datapath:

```
module DataPath(
    input wire [5:0] control_signals,
... );
    ...
    wire [31:0] connection;
    ...
    mux2 mymux(
    .sel (control_signals[1]), //control signals to be handled by the control unit
    .d1  (myd1),
    .d0  (myd0),
    .y   (connection) //connection of the two components
    );
    multiplier mymultiplier (
    .A (connection), //connection of the two components
    .B (myB),
    .Y (myY)
    );
    ...
endmodule
```

3. According to the current state, next state, and their related control signals in the output table, build the FSM in the control unit module. Also, you can double-check with the ASM chart and the bubble diagram.

For example:

```
module ControlUnit(
    input wire clk,
    input wire rst,
    output reg [2:0] CS, //current state
    output reg [5:0] control_signals,
```

```
    ...
  );

   reg [2:0] NS; //next state

    ...

   always @(CS) //can have other status signals from the datapath as well
   begin
   case(CS)
     3'b000: begin
           control_signals = 6'b000000; //can have control outputs as well
           NS = 3'b001;
         end

     3'b001: begin
           control_signals = 6'b000001;
           NS = 3'b010;
         end

     3'b010: begin
           control_signals = 6'b000010;
           if ( ... ) //can have different NS according to some control signals as well
           ...
         end

       ...
   endcase
  end

   ...

   always @(posedge clk, posedge rst)
     begin
       if(rst) CS <= 3'b000;
       else   CS <= NS;
   end

   ...

endmodule
```

4. Connect your datapath module and control unit module according to your CU-DP block diagram.

For example:

```
module my_design(
   input wire clk,
   input wire rst,
   output wire [1:0] CS,
   ... // other external input/outputs
 );

   ...
   wire [5:0] ctrl_signals;

   ControlUnit CU(
   .clk          (clk),
   .rst          (rst),
   .CS           (CS),
   .control_signals (ctrl_signals), //connection between the CU and DP. Can have other connections as well.

   ....
   );

   DataPath DP(
   .control_signals (ctrl_signals), //connection between the CU and DP. Can have other connections as well.
   ....
   );

   ....
endmodule
```

5. Now basically the system's code is done by following the diagrams. Next, you can build a testbench to debug & test your system with simulations. You then can see your virtually designed system running in the simulation. All signals and outputs can be shown in the waveform. Test out different options and have fun!

But how to use Verilog?

Well, if you do not understand what I have been saying on this page or do not know how to code with Verilog, then this course might be a little bit too much for you. But do not worry. It is not that hard to catch up, as there are plenty of good resources to help you learn Verilog quickly. I will list a few here.

1. HDLBits. The #1 website I would recommend to anyone who wants to learn Verilog. Very fun to learn. Simply go to your interested topics.

https://hdlbits.01xz.net/wiki/Problem_sets#Verilog_Language (https://hdlbits.01xz.net/wiki/Problem_sets#Verilog_Language)

2. Verilog In One Day. This website will help you go through the basics of Verilog quickly.

http://www.asic-world.com/verilog/verilog_one_day.html (http://www.asic-world.com/verilog/verilog_one_day.html)

3. Simply search online. There are plenty of good articles, Q&As, videos, etc. out there.