Verilog in the Wild

A true story by Dan Parker

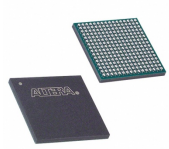
About Me

- Masters student
- Computer Engineer
- Work for Fusion-io

What's this all about?

- Different point of view?
- Be prepared
- Know what you know

Vendors



- Altera
- Xilinx
- A billion more
- Or go ASIC
- eASIC is like the half way point
- Just remember to check DigiKey when you want an IC.

Get to know your FPGA

- Size
- Speed
- Package (Signals)
- Configurable IO
- Hard IP Blocks
- Other awesome stuff (PLLs, BRAMs, partial reconfiguration, IODELAY)

Spartan-3E Libraries Guide for HDL Designs

- Tells you how to use its special pieces

```
// DCM: Digital Clock Manager Circuit
// Virtex-II/II-Pro and Spartan-3
// Xilinx HDL Libraries Guide, version 10.1.2
DCM #(
.SIM_MODE("SAFE"), // Simulation: "SAFE" vs. "FAST", see "Synthesis and Simulation Design Guide" for d
.CLKDV_DIVIDE(2.0), // Divide by: 1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6.0,6.5
// 7.0,7.5,8.0,9.0,10.0,11.0,12.0,13.0,14.0,15.0 or 16.0
.CLKFX_DIVIDE(5), // Can be any integer from 1 to 32
.CLKFX_MULTIPLY(4), // Can be any integer from 2 to 32
.CLKIN_DIVIDE_BY_2("FALSE"), // TRUE/FALSE to enable CLKIN divide by two feature
.CLKIN_PERIOD(20.0), // Specify period of input clock
.CLKOUT_PHASE_SHIFT("NONE"), // Specify phase shift of NONE, FIXED or VARIABLE
.CLK_FEEDBACK("1X"), // Specify clock feedback of NONE, 1X or 2X
.DESKEW_ADJUST("SYSTEM_SYNCHRONOUS"), // SOURCE_SYNCHRONOUS, SYSTEM_SYNCHRONOUS or
// an integer from 0 to 15
.DFS_FREQUENCY_MODE("LOW"), // HIGH or LOW frequency mode for frequency synthesis
.DLL_FREQUENCY_MODE("LOW"), // HIGH or LOW frequency mode for DLL
.DUTY_CYCLE_CORRECTION("TRUE"), // Duty cycle correction, TRUE or FALSE
.FACTORY_JF(16'hC080), // FACTORY JF values
.PHASE_SHIFT(0), // Amount of fixed phase shift from -255 to 255
.STARTUP_WAIT("FALSE") // Delay configuration DONE until DCM LOCK, TRUE/FALSE
)
DCM_inst (
.CLK0(CLK0), // 0 degree DCM CLK output
.CLK180(CLK180), // 180 degree DCM CLK output
.CLK270(CLK270), // 270 degree DCM CLK output
.CLK2X(CLK2X), // 2X DCM CLK output
.CLK2X180(CLK2X180), // 2X, 180 degree DCM CLK out
.CLK90(CLK90), // 90 degree DCM CLK output
.CLKDV(CLKDV), // Divided DCM CLK out (CLKDV_DIVIDE)
.CLKFX(CLKFX), // DCM CLK synthesis out (M/D)
.CLKFX180(CLKFX180), // 180 degree CLK synthesis out
.LOCKED(LOCKED), // DCM LOCK status output
.PSDONE(PSDONE), // Dynamic phase adjust done output
.STATUS(STATUS), // 8-bit DCM status bits output
.CLKFB(CLKFB), // DCM clock feedback
.CLKIN(CLKIN), // Clock input (from IBUFG, BUFG or DCM)
.PSCLK(PSCLK), // Dynamic phase adjust clock input
.PSEN(PSEN), // Dynamic phase adjust enable input
.PSINCDEC(PSINCDEC), // Dynamic phase adjust increment/decrement
.RST(RST) // DCM asynchronous reset input
));
// End of DCM_inst instantiation
```

Tools and Such

- Linux
- Command Line
- Scripts
- Simulators
- Revision control
- IP
- ChipScope

Intellectual Property

- Guard yours (copyright notices and discretion)
- Mostly opposite philosophy from school: don't write things yourself if you can find the answer on the internet.
- All kinds of great products available from synthesis to verification to synthesizable verification blocks.
- The first question you should ask when something becomes non-trivial is "Can I buy this from somebody else?" The next question is "Can my company allocate the funds?"
- Service contracts are important

Simulators

- ModelSim
- QuestaSim
- NCSim – My Favorite
- VCS
- Icarus + GTKWave - Free
- ISim

Revision Control

- SVN
- Mercurial
- Perforce
- Accurev
- Whatever you like really…
- Make sure you can track code changes and revert if things get too messed up.
- Allows you to freely experiment with no permanent damage.

Let's talk about code

- Parameterization
- Generate statements
- Question mark notation
- Always size you literals (13'h4)
- Schematic capture? – Sorry, but no.
- Naming conventions, just some ideas I've seen:
  parameters in all caps
  add _i or _o for ports depending on direction
  add _n for active low signals
  add R for a registered version of the signal
- Style?

Cool example

```
module cool_logic #(parameter WIDTH=8,
                    parameter TYPE=0
              )(
                input [WIDTH-1:0] A,
                input [WIDTH-1:0] B,
                output [WIDTH-1:0] Y);
   localparam ADD=0;
   localparam SUB=1;
   localparam MUL=2;
   localparam MAX=3;

   generate
      case(TYPE)
        ADD: assign Y=A+B;
        SUB: assign Y=A-B;
        MUL: assign Y=A*B;
        MAX: assign Y=A > B ? A : B;
        default: assign Y=A|B;
      endcase
   endgenerate
endmodule // cool_logic

module top();

   wire [15:0] A;
   wire [15:0] B;
   wire [15:0] Y;

   cool_logic #(.WIDTH(16),.TYPE(3)) cl (.A(A),.B(B),.Y(Y));

endmodule // top
```

A BIT MORE ON PARAMETERIZATION

- To create a parameterized module type:
- module <module type> # (<parameter list>) (<port list>);

- To instantiate a parameterized module:
- <module type> #(<parameter list>) <instance name> (<port list>);

GENERATE

- My favorite feature
- Starts with generate ends with endgenerate
- Let's you do if statements, always blocks, for loops.
- Just be careful, errors in a generate block are

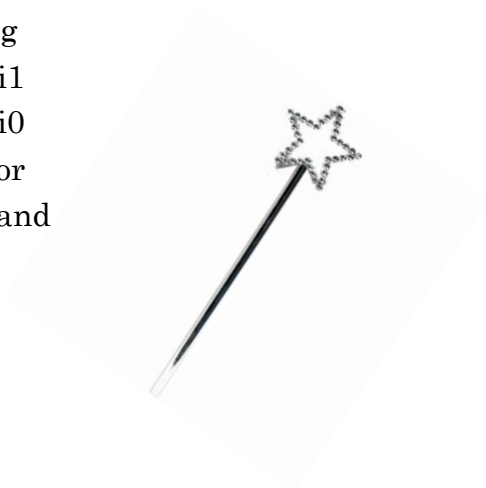
```
localparam WIDTH=8;
wire clk;
reg [3*WIDTH-1:0] data;
genvar i;
generate
  for (i=0;i<3;i=i+1)
    begin : THIS_LOOP_MUST_BE_LABELED_TO_COMPILE
      always @(posedge clk)
        data[i*WIDTH+:WIDTH]=data[i*WIDTH+:WIDTH] + 1;
    end
endgenerate
```

The tick versus the apostrophe

- ` '
- Tick is like a compiler directive
- Gives you macros and conditional blocks
- `timescale 1ns/1ps
- `define Y 5
- `define X(a,b) (a+b);
- To access you use f=`Y; or `X(g,h)
- `ifdef Y
- `else
- `endif

Data Types

- wire
- reg
- tri1
- tri0
- wor
- wand



Verification

- Unit testing
- System level testing
- Self-checking tests
- Constrained random
- Directed tests
- ABSTRACTION!!!
- SystemVerilog
- Assertions/Coverage
- Simulation V.S. Synthesis

Cool Example

```
task add;
  input [15:0] A;
  input [15:0] B;
  begin
    ALUop = 0;//pretend zero is add
    ALUin1 = A;
    ALUin2 = B;
   @(posedge clk);
    if (ALUout != (A+B))
      $display("ERROR: Add failed. A=%h B=%h ALUout=%h Expected: (A+B)=%h",A,B,ALUout,A+B);
  end
endtask

task sub;
  ...
endtask

task press_a_key;
  ...
endtask
```

"It works in simulation"

- The craziest things will work in simulation, but are not remotely synthesizable.
- Use "<=" instead of "=" for synthesizable code.
- Assign statements are great.
- Remember, subsequent lines of code are not subsequent clock cycles. << Biggest mistake

Wait, repeat that last part

- wait(data==7); blocks until data==7
- repeat(5)
  @(posedge clk); waits 5 clock cycles
- #100; waits 100 time units

- Another cool thing, you can traverse the design hierarchy in simulation. E.g. you can read: proc.datapath.alu.ALUout

Constraint Driven Optimization

- The ISE synthesizer is cruel
- The constraint system is nice. (and treats DCMs correctly)

```
NET "raw_clk"  LOC = "B8" | IOSTANDARD = LVCMOS33 | PERIOD = 20ns;
```

• Routing Results:	All Signals Completely Routed
• Timing Constraints:	X 1 Failing Constraint
• Final Timing Score:	4781122 (Setup: 4781122, Hold: 0, Component Switching Limit: 0) (Timing Report)

Clock Domains

- 1 is the easiest
- Crossing clock domains is really nasty
- Asynchronous FIFOs help
- Your Xilinx BRAMs can take 2 different clocks for 2 different ports. This is great for frame buffers.
- Your FPGA may implement your solution in a way that breaks.
- Consider checking OpenCores.
- Or coregen in ISE.

FIFOS

- First In First Out
- Take a dual ported RAM of some kind
- Create a head pointer and a tail pointer
- When you write you store to RAM[head] and increment the head pointer
- When you read you read from RAM[tail] and increment the tail pointer
- Empty is when tail==head
- Full is when (head+1)==tail (Remember this is modular arithmetic.)

JUST SOME TIPS

- Get your assembler done soon, if it isn't done already. You need your assembler to test your processor and you need your processor to test your assembler.
- You can save your waveform configuration ISIM so you don't have to keep finding signals.
- Make sure to divide things appropriately into modules.
- Debouce your keyboard input. It's not actual bounce, it's just super noisy.
- LEDs are your friend.

Quality Matters

- Read your warnings!!! You don't want a latch.
- If you don't test it, it doesn't work.
- Code lingers, if you write it well the first time you will have a long and fruitful relationship.
- In case I forget my stories (Sr. Proj and nand connection test.)