RISC-V labs

This labs is broken down into 3 parts lab1, lab2, and lab3 designed to build off of each other for an overall understanding of RISC-V processors.

1 prerequisites

- Access to cadteaching8.
- Basic navigation of the Unix terminal.
- This lab assumes basic competence in writing SystemVerilog and an ability to test your own designs using some of the tools available in cadteaching8.

Each alb

Unix command tutorial:

1. Open the terminal which can be found in the top left of cadteaching8.
2. Make a directory that will house this project
   `<mkdir riscv_labs>, <cd riscv_labs>`
3. Clone the lab repository from GitHub
   `<git clone abs@github.com/askldjf>`
4. `<cd labs>`
5. Go to the lab1 directory
   `<cd ./laboratories/lab1>`
6. Open the lab1 script
   `<evince lab1.pdf>`
7. To edit files, use command
   `<nedit example.sv>`

If you are comfortable using SystemVerilog and some of its features, feel free to skip to lab1.

2 SystemVerilog

SystemVerilog is a hardware description language (HDL), used to describe implementations of digital logic. Based on the C programming language, a lot of syntax carries over. SystemVeriog is a powerful language that is used for Synthesizable[1] and non-synthesizable code for testing.

[1]Synthesis is the process of compiling SystemVerilog into standard Cells (e.g and2, nand3, nor, d-type flipflops, mux2) which is called a netlist. This can then be used to create and tape out a physical microchip.

2.1 Basic structures

Most simple program:

```
// hello_world.sv
module hello_world;
  initial begin
    $display("Hello World!");
  end
endmodule
```

Run <xmverilog hello_world.sv>

Synthesizable designs and non-synthesizable tests are split into separate files asa follows.

```
// counter.sv
module counter (
  input Clock, // is actually input wire Clock, though wire can be left out
  input nReset,
  output logic [7:0] counter, // counter is now an 8 bit register
  output logic is_even
);
timeunit 1ns; timeprecision 100ps;

always_ff @(posedge Clock, negedge nReset) // Sequential, meaning updated at the rising edge
of Clock.
  if (!nReset) counter <= 0; // Asynchronous reset, meaning always runs at the falling edge of
nReset.
  else counter <= counter + 1;

always_comb begin
  is_even = counter[0];
end
endmodule
```

Modules cannot be run without stimulus, where non-synthesizable testbenches are used.

```
// counter_tb.sv
module counter_tb;
timeunit 1ns; timeprecision 100ps;
logic Clock; // Use logic when the signal is driven from this module
logic nReset; // Use wire when the signal is passed into this module
wire [7:0] counter;
wire is_even;

initial begin nReset = 1; #10 nReset = 0; #10 nReset = 1; end // Pulses nReset
always begin Clock = 1; #50 Clock = 0; #50 Clock = 1; end // Starts a Clock that runs forever

test dut(.Clock(Clock), .nReset(nReset), .counter(counter), .is_even(is_even));
// Used to create an instance of our design under test (DUT is test.sv) inside this testbench.

initial begin
  repeat(150) begin
    @(posedge Clock);
    $display("counter=%d, is_even=%b", counter, is_even);
  end
  $finish;
end
endmodule
```

Inside of cadteaching8, copy these two files into a directory of your choosing.

1. run <xmverilog counter_tb.sv counter.sv>
2. To see the individual signals use <xmv_gui counter_tb.sv counter.sv>

Basic design patterns:

```
// case_example.sv
typedef enum logic [2:0] { // enumeration links names to numbers for convenience
  ADAM = 3'd0,
  BILL = 3'd1,
  CONNOR = 3'd2,
  DENVER = 3'd3,
} people_e;

module case_example;
  people_e people = ADAM;
  logic [7:0] favourite_number;

  case (people)
    ADAM: favourite_number = 8'd10;
    BILL: favourite_number = 8'd20;
    CONNOR: favourite_number = 8'd1;
    DENVER: favourite_number = 8'd1;
    default: favourite_number = 8'd0;
  endcase
  $display("person=%s, favourite number=%d", people.name(), favourite_number);
endmodule
```