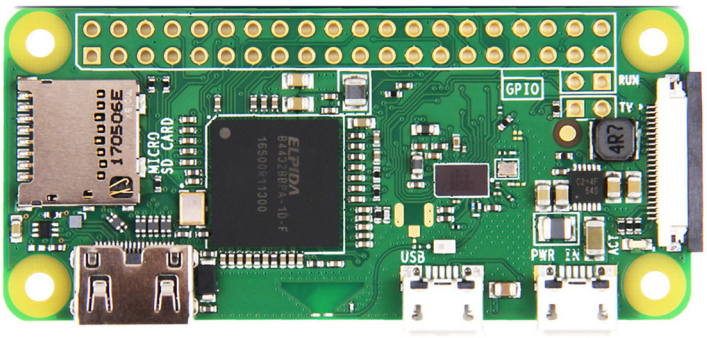
University of Cape Town



EEE3096S/EEE3095S

Embedded Systems II

Work Packages 2021

---

4 October 2021

Abstract

Welcome to the practicals for EEE3096S. These instruction are applicable to all practicals so please take note! It is critical to do the pre-practical/tutorial work. Tutors will not help you with questions with answers that would have been known had you done the pre-practical work. The UCT EE Wiki (wiki.ee.uct.ac.za) is a very useful point to find any additional learning resources you might need. Use the search functionality on the top right of the page.

Practical Instructions

- **Naming**
  All files submitted to Vula need to be in the format. Marks will be deducted if not.

```
pracnum_studnum1_studnum2.fileformat
```

- **Submission**
  - Work packages consist of a tutorial and a practical. Be sure you submit to the correct submission on Vula!
  - All text assignments must be submitted as pdf, and pdf only.
  - Code within PDFs should not be as a screenshot, but as text.
  - Where appropriate, each pdf should contain a link to your GitHub repository.
- **Groups**
  All practicals are to be completed in groups of 2. If a group of 2 can't be formed, permission will be needed to form a group of 3. You are to collaborate online with your partners. See more here.
- **Mark Deductions**
  Occasionally, marks will be deducted from practicals. These will be conveyed to you in the practical, but many will be consistent across practicals, such as incorrectly names submissions or including code as a screenshot instead of formatted text.
- **Late Penalties**
  Late penalties will be applied for all assignments submitted after the due date without a valid reason. They will work as 20% deduction per day, up to a maximum of 60%. After this, you will receive 0% and have the submission marked as incomplete.

Contents

Chapter 1

Work Package 5

1.1 Tutorial - ALU and HDL Basics

In this tutorial, you will learn about:

- Verilog
- Work through some Verilog examples of basic components such as a register or multiplexer

1.1.1 Pre-requisites

This practical makes use of writing Verilog, a hardware descriptor language. You can write Verilog in many ways, from just a plain text editor, to complex, feature-rich IDEs such as Vivado. For this tutorial we will use EDAPlayground (This is just a suggestion, you are free to use Vivado or iVerilog).

In order to complete the practical and get into the mindset for writing HDL, you might find this page useful.

For pointers on writing testbenches, read through http://wiki.ee.uct.ac.za/HDL_Simulation

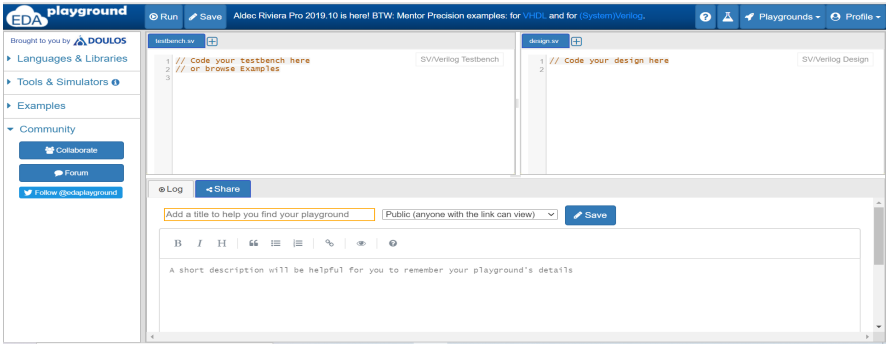
1.1.2 EDA Playground walkthrough

Details on EDA Playground can be found at the official docs. Possibly the best place to start is to go watch EDA's own Introduction to EDA 10min Youtube clip. Although, note: it was made in 2013 so the interface has changed slightly.

The tutorial below will help you navigate through EDA Playground and set up your environment to work on this practical.

1. Go to edaplayground.com

2. Create an account and log in



Your start-up screen after logging in should look like the above.

3. In the left hand side bar: Click on **"Languages & Libraries"**
   - From **"Testbench + Design"**select **"SystemVerilog/Verilog"**
4. In the left hand side bar: Click on **"Tools & Simulators"**
   - Select **"Icarus Verilog 0.10.0 11/23/14"** (All the Icarus Verilog options should work).
5. Now you are set up to start coding:
   - Copy and paste the ALU code below into the **Design.sv block** on the right hand side of the EDA screen. This code describes a extremely simple Arithmetic Logic Unit (ALU) that can perform 1 of 4 operations on 2 operands (values) passed into it. Note: Code is cavailble in the github repo for easy copy too: https://github.com/UCT-EE-OCW/EEE3096S-2021/tree/main/WorkPackage5

```
//Define the module
module ALU(
    input clk,A,B,
    input [1:0] ALU_Sel,
    output reg ALU_out
);

reg ALU_result;

always@(posedge clk)
    begin
        case(ALU_Sel)
            2'b00:  //Manually enumerate Addition = 00
                ALU_result = A + B;

            2'b01:  //Manually enumerate Subtraction1 = 01
                ALU_result = A - B;

            2'b10:  //Manually enumerate Subtraction2 = 10
                ALU_result = B - A;

            2'b11:  //Manually enumerate Multiplication = 11
                ALU_result = A * B;
            default: ALU_result = A;
        endcase

        ALU_out <= ALU_result;
    end
endmodule
```

- To test this ALU, copy and paste the testbench code below into the **"testbench.sv"** block.
  Note: In the testbench below, the clock is manually toggled before each operation. In more complex designs later in the course, it it more useful to use `always` block to define the clock, and then just use regular wait designators between operations. But for now keep it simple as shown.

```
   //Define the ALU testbench module
module ALU_tb();
//inputs
reg clk, A,B;
reg[1:0] ALU_Sel;
// output
wire ALU_Out;

//Instantiate the design under test
ALU ut(.clk(clk),
.A(A),
.B(B),
.ALU_Sel(ALU_Sel),
.ALU_out(ALU_Out));

initial begin  //Initial means this only happens once
    $display("A  B  ALU_Sel  ALU_Out");
    $monitor("%b  %b  %b     %b",A,B,ALU_Sel, ALU_Out);
    clk = 1'b1;
    A = 1'b1;
    B = 1'b1;
    ALU_Sel = 2'b0;
    #5  //Note: This is not synthesizable and only available in simulation
        clk=!clk;
    #5
        clk=!clk;
        ALU_Sel = 2'b01;
    #5
        clk=!clk;
    #5
        clk=!clk;
        ALU_Sel = 2'b10;
    #5
        clk=!clk;
    #5
        clk=!clk;
        ALU_Sel = 2'b11;
    #5
        clk = !clk;
    end
endmodule
```

6. Click on **Save** in the top left corner and then **Run**. Your output (under log) should look like the following;

```
A  B  ALU_Sel  ALU_Out
1  1  00       0
1  1  01       0
1  1  10       0
1  1  11       1
Done
```

7. In order to test different inputs, you can change the values of A and B in the testbench on lines20 21. We'll test it with;
   1. A = 1'b0; B = 1'b1;
   2. A = 1'b1; B = 1'b0;
   3. A = 1'b0; B = 1'b0;

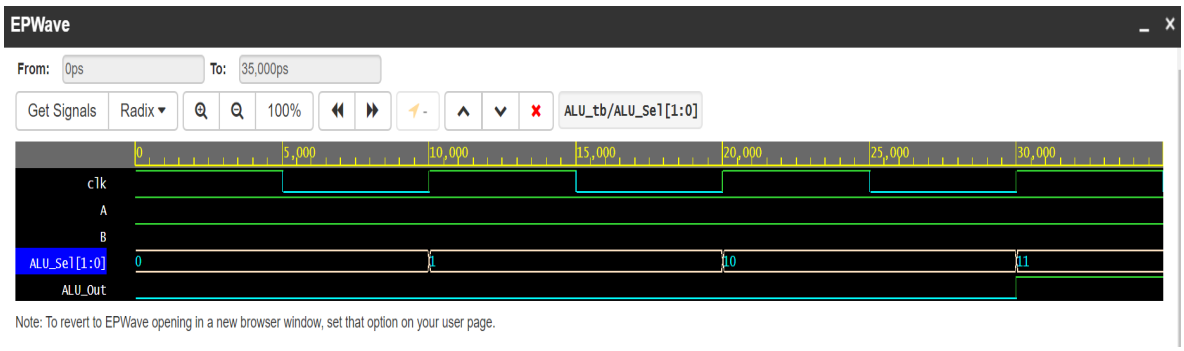
8. To display a waveform, click on **"Tools & Simulators"**, click on the **"Open EPWave"** checkbox. Add the following lines of code in the testbench file in the initial block under monitor;

```
1        $dumpfile("dump.vcd");
2        $dumpvars;
```

   Save and Run

9. The following simulation will pop up.
   - If you don't see any signals click on "Get Signals" to add.
   - To delete any of the fields, click on the signal name and click on the red x.
   - To move a variable and wave up or down, click on the variable name and click on the up or down arrows.

   After deleting and moving fields, a neatened simulation should look like this.



10. Tips:

    - You can configure EDA under your profile to use different editors and usefully to tell it to open the simulation window in a separate tab.
    - You can add your lab partner to collaborate on the same playgound as you. In the sidebar at the bottom, Go to **Community**, click on **Collaborate**. A block containing a link will pop up. Send this link to your partner who will then be able to edit the same playground as you.

1.1.3 Questions

Submit the following answers to Vula in a single PDF under the "tutorials" tab, correctly named.

1. How do FPGAs differ from microcontrollers? Give two advantages of FPGAs, and 2 disadvantages. [2 marks]
2. What is the difference between blocking and non-blocking assignments? Support your answer by comparing the two code snippets below and how their outputs may differ. [4 marks]

```
module blocking(in, clk, out);
  input in, clk;
  output out;
  reg q1, q2, out;
  always @ (posedge clk)
  begin
    q1 = in;
    q2 = q1;
    out = q2;
  end
endmodule
```

```
module nonblocking(in, clk, out);
  input in, clk;
  output out;
  reg q1, q2, out;
  always @ (posedge clk)
  begin
    q1 <= in;
    q2 <= q1;
    out <= q2;
  end
endmodule
```

3. Port Mapping in module instantiation can be done in two different ways;
    1. Port Mapping by order
    2. Port Mapping by name

```
module sum(
    input clk,A,B,
    output reg out
    );
    always@(posedge clk)
    begin
            out <= A+B;
    end
endmodule
```

For the module above; what is the output of the following testbenches done using port mapping by order (on the left) and mapping by name (on the right):

```
module sum_tb();
    //inputs
    reg A,B, clk;
    //output
    wire out;
    //instantiate the module
    sum ut(A,B,clk,out);

    initial begin

      $display("A  B   Out");
      $monitor("%b  %b   %b",A,B, out);
          clk = 1'b1;
      A = 1'b0;
      B = 1'b1;

    end
endmodule
```

```
module sum_tb();
    //inputs
    reg A,B, clk;
    //output
    wire out;
    //instantiate the module
 sum ut(.A(A),.B(B),.clk(clk),.out(out));

    initial begin

      $display("A  B   Out");
      $monitor("%b  %b   %b",A,B, out);
          clk = 1'b1;
      A = 1'b0;
      B = 1'b1;

    end
endmodule
```

Comment on the results from each testbench. Which port mapping method is better to use when instantiating a module with many ports? [2 marks]