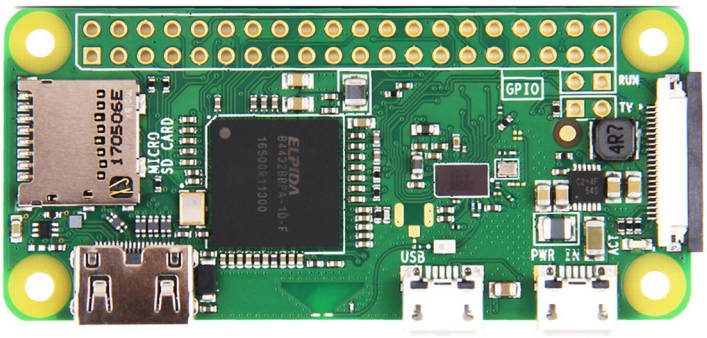
University of Cape Town



EEE3096S/EEE3095S

Embedded Systems II

Work Packages 2021

---

18 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 6

An important note about this work package and EEE3095S students. Computer Science students are required to carry out additional work in this course due to a higher credit rating. This tutorial and practical include these additional tasks and are therefore required for all students registered for EEE3095S. EEE3096S students are strongly encouraged to at least look through and possibly also execute these additional tasks for their educational value, but are not required to do so, demmonstrate such, or submit handins for these additional tasks. All EEE3095S additional tasks are clearly indicated as such. All students are to complete all tasks unless otherwise indicated.

1.1 Tutorial - Verilog Test-benches and Intro to Containerisation for IoT

In this tutorial, you will learn about:

- Verilog Test-benches in preparation for a more complex mini CPU in the practical
- Computer Science Students will work through an external tutorial preparing them to deploy an application on the Pi Zero using docker and an open source project Balena OS in the prac.

1.1.1 Testbenches

In the practical of this workpackage you will create a functional simple CPU. As an initial step towards this, the rubric version of the ALU from WP5 and a Small Register memory are available in the course prac git repo here. Both the raw source code are available for use in EDA Playground and a Xilinx Vivado project version if your would like to use that instead.

The following simple register memory module is supplied. You should note particularly:

- The use of parameters to define memory width and depth
- The use of wen (write enable) which will serve as a control signal when used with the ALU in the practical later

```
`timescale 1ns / 1ps

module reg_mem (addr, data_in, wen, clk, data_out);

    parameter DATA_WIDTH = 4; //4 bit wide data
    parameter ADDR_BITS = 3; //8 Addresses

    input [ADDR_BITS-1:0] addr;
    input [DATA_WIDTH-1:0] data_in;
    input wen;
    input clk;
    output [DATA_WIDTH-1:0] data_out;

    reg [DATA_WIDTH-1:0] data_out;

    //8 memory locations each storing a 4bits wide value
    reg [DATA_WIDTH-1:0] mem_array [(2**ADDR_BITS)-1:0];

    always @(posedge clk) begin

        if (wen) begin //Write
            mem_array [addr] <= data_in;
            data_out <= #(DATA_WIDTH)'b0;
        end

        else begin //Read
            data_out <= mem_array[addr];
        end
    end

endmodule
```

The testbench for this memory is also supplied. You should note particularly:

- How parameters are passed to the module when it's instantiated
- How 2 for loops are used to test all of the module's addresses with different values written to each, and then how the same values are read back.
- How parameterised $display commands are used to report each test

- While Verilog uses integer as a keyword, SystemVerilog uses int. If you are using EDAPlayground, take note that it assumes you've written your testbench in SystemVerilog unless you manually update the testbench file.

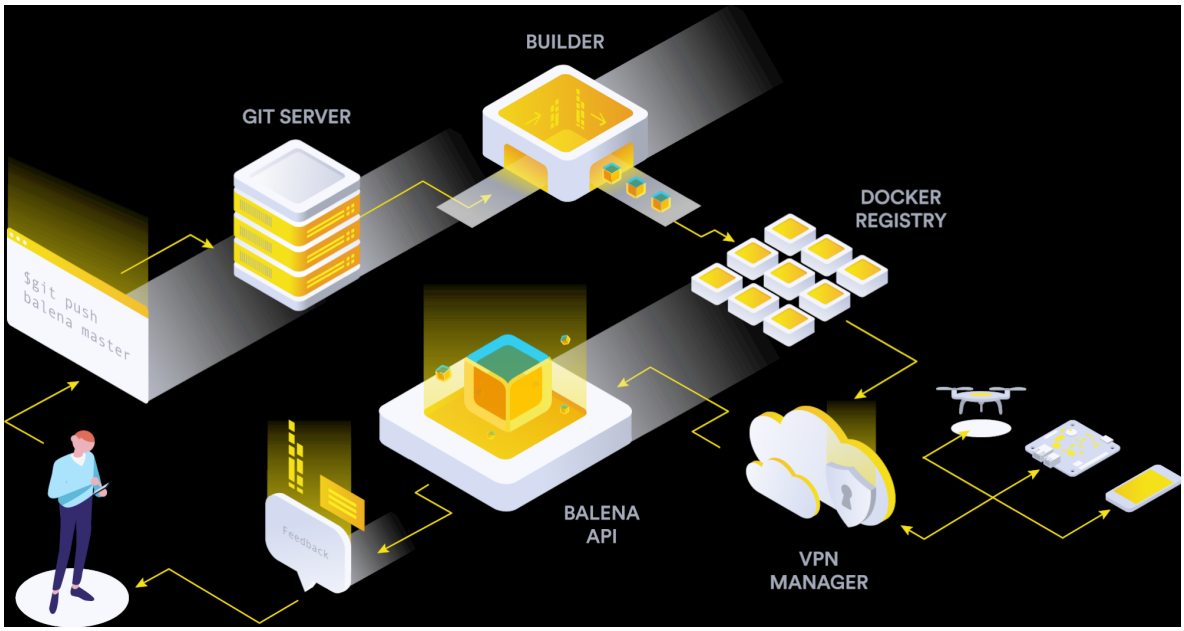
```
`timescale 1ns / 1ps

module tb_reg_mem;

    parameter DATA_WIDTH = 4; //4 bit wide data
    parameter ADDR_BITS = 3; //8 Addresses

    reg [ADDR_BITS-1:0] addr;
    reg [DATA_WIDTH-1:0] data_in;
    wire [DATA_WIDTH-1:0] data_out;
    reg wen, clk;

    //Note passing of parameters syntax
    reg_mem #(DATA_WIDTH,ADDR_BITS) RM (addr, data_in, wen, clk, data_out);

    initial begin
        /* For use in EDAPlayground
        $dumpfile("dump.vcd");
        $dumpvars(1, tb_reg_mem);
        */
        clk = 0;
        wen = 1;

        //Write 10-17 to addresses 0-7
        for(integer i=10;i<18;i=i+1)
        begin
            data_in = i;
            addr = (i+2);
            $display("Write %d to address %d",data_in,addr);
            repeat (2) #1 clk = ~clk;
        end
        wen =0;
        #1;
        //Read 10-17 from addresses 0-7
        for(integer i=10;i<18;i=i+1)
        begin
            data_in = i;
            addr = (i+2);
            $display("Read %d from address %d",data_in,addr);
            repeat (2) #1 clk = ~clk;
        end
    end
endmodule
```

Tutorial Tasks:

1. Modify the Testbench and the memory module to be 8bits (1byte) wide and 32 entries deep.
2. Get the testbench running in either EDA playground or Vivado with your modified code. Your testbench must test writing and reading for each address in your new memory.
3. Submit a pdf showing:
   (a) A screenshot the resulting simulation waveform with values stored and written to memory shown in decimal.
   (b) A **public** git repository with your code. This will be used again for the practical section next week. Note, if you use Vivado, include the whole project in your gitrepo. If you use EDAPlayground, save your project as public with a suitable name and include a link to it in your report. Your tutor will follow this link/clone your Vivado project to verify operation.

1.1.2 IoT deployment via containerisation

Required of EEE095S students only: The theory underlying Containerisation and Virtualisation will be covered in the last week of lectures. However, the following tutorial can easily be followed prior to this content. Balena offers a tool ecosystem that facilitates the deployment of applications to IoT devices. Given the remote nature and scale/number of devices commonly deployed by a company offering IoT infrastructure, it's impossible to manually services individual devices. However, as digital devices connected to the internet devices need to be monitored for failures and updated as security vulnerabilities are detected and patched. The Balena toolchain provides a scalable mechanism for easily managing thousands of remote devices.



The figure above shows the core components of the Balena ecosystem:

- The Balena operating system (Linux with kernel integrated docker) is deployed on devices running both a supervisor container and your application container(s).
- Once deployed a user is able to both monitor (via a web interface) and update one or any or all deployed devices by rebuilding the application container and initiating an update
- Update of deployed containers can be acomplished via: (1) The Command Line Application which you will use today, (2) Balena Deploy, and (3) git. For more information on each of these see Balena Update Tools. Depending on which is used, different components of the shown ecosystem are used.

For now, the primary information you need to know is that applications can be containerised using a Dockerfile (a file that describes where the source code for your application is and how to execute it). And that the Balena tools allow you to deploy, monitor, and update these applications on 1 or more IoT devices such as your Raspberry Pi over the internet.

Tutorial Task: Work through this getting started using a PiZeroW and Python tutorial. Once complete you will have a simple Python Flask based webserver running on your Pi. You must include in your tutorial submission pdf:

1. A link to a **public** git repository containing your python flask application and dockerfile. This repository should contain all the same files as this repo balena-python-hello-world which the tutorial will lead you to use. In your repo however, 2 changes are needed:

- Update the README appropriately to reflect what the repo is (no longer a tutorial repo)
- Change the hello world message to something unique. Eg your team name, or your favourite language/anything unique to show you've modified the python code.

2. Include in your tutorial pdf report screenshots of both your webserver running and accessed showing your unique message, and your Pi connected to and with a running status in the Balena monitoring web interface (up to 10 devices are free after).