Stage-2.3-COL216

Vatsal Jingar 2020CS50449

February 2022

1 Introduction

In this Stage, we have to make a multi-cycle processor with the specifications given in the Assignment Problem Paper. I have used the data memory for both storing the instructions and data. The file named as Machine.vhd contains the logic and datapath for the simulation and synthesis of the multi-cycle processor. It will also change the state whenever required. So my FSM, Controller and Datapath are merged together in one file which is "Machine.vhd".
The multicycle processor uses one cycle to execute one part of a instruction where the execution of a single instruction is broken in several states.

1.0.1 Disclaimer

The memory for data is starting from 64th index. And the memory for storing instructions of memory are from 0 to 63.

2 Some points for the simulation of my files

- Machine.vhd : Main file contains all the logic and data signals
- The other files signifies the files of different components used in the Machine.

3 Logic for implementation

In the Main file "Machine.vhd", There is an entity machine with its architecture whose process trigger when clock have a rising edge. There is a signal State which will keep the track of the states. The initial state for the process is "0000".

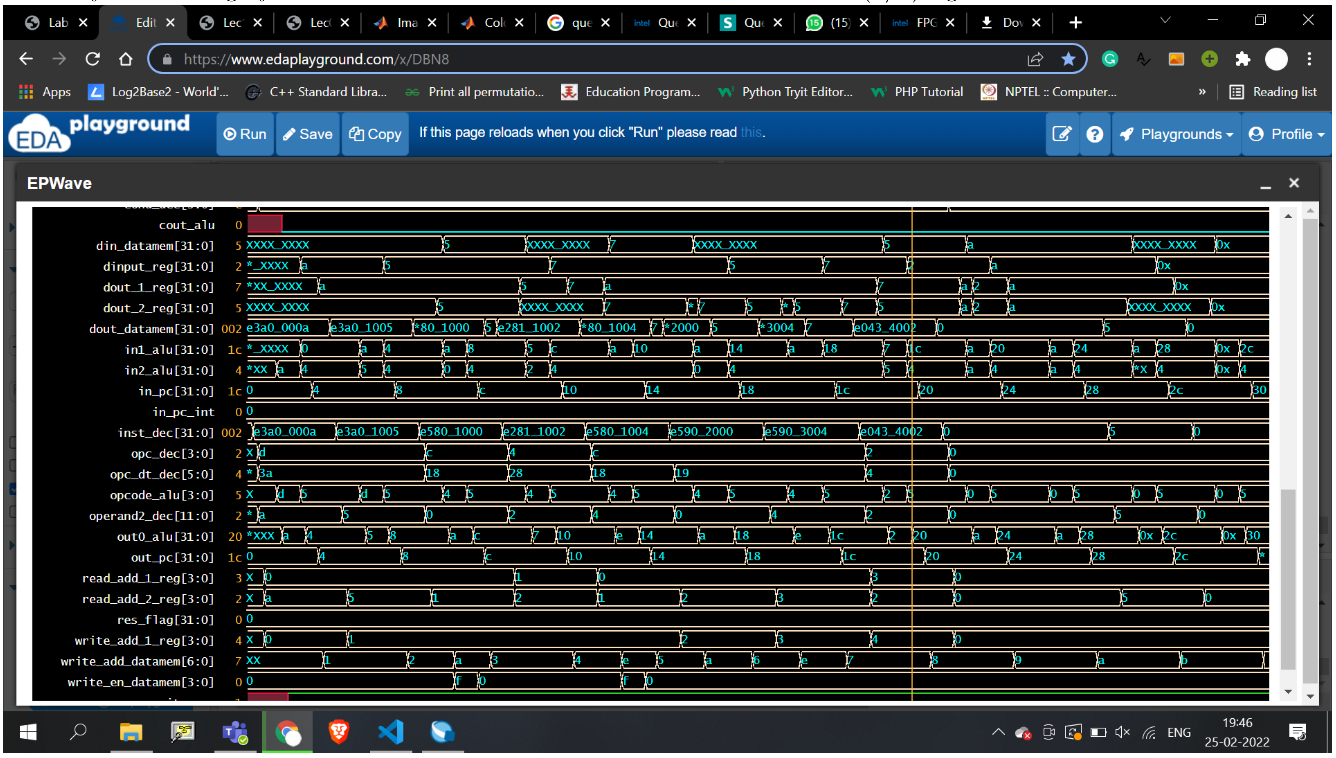
- "0000" :- It is the initial state where IR store the instruction to be executed and the inputs for next state. At every falling edge the individual inside components are triggered and for rising edge machine states are triggered.
- "0001" :- At this stage, the values of outputs of register are stored in the local machine signals(used as register), A and B. Also at this stage we will choose on which stage we have to move on the basis of the value of F if 00 then at DP state, 01 then at DT state, and 10 then at B state.
- As my controller and data path are in one single file where states are changing so with the assignment of new state, I have also included the changes to be made for the inputs of individual components.
- "0010" :- This stage will store the output of ALU in Res and will select on the basis of instruction whether we have to store or load.
- After this state, when the clock will trigger it will perform DP, DT or B instruction. At the end of every instruction i.e is when we reach the last states(for DP,for DT,for B), The input to ALU is given to update the PC by 4. Also I have included three dummy states which helps to execute the process.

4 Test-Cases-1

The first Test Case used by me is signal memarray : memorytype := (0 = X"E3A0000A", 1 = X"E3A01005", 2 = X"E5801000", 3 = X"E2811002", 4 = X"E5801004", 5 = X"E5902000", 6 = X"E5903004", 7 = X"E0434002", others = X"00000000");

4.1 EP-WAVE IMAGE FOR THE ABOVE CASE

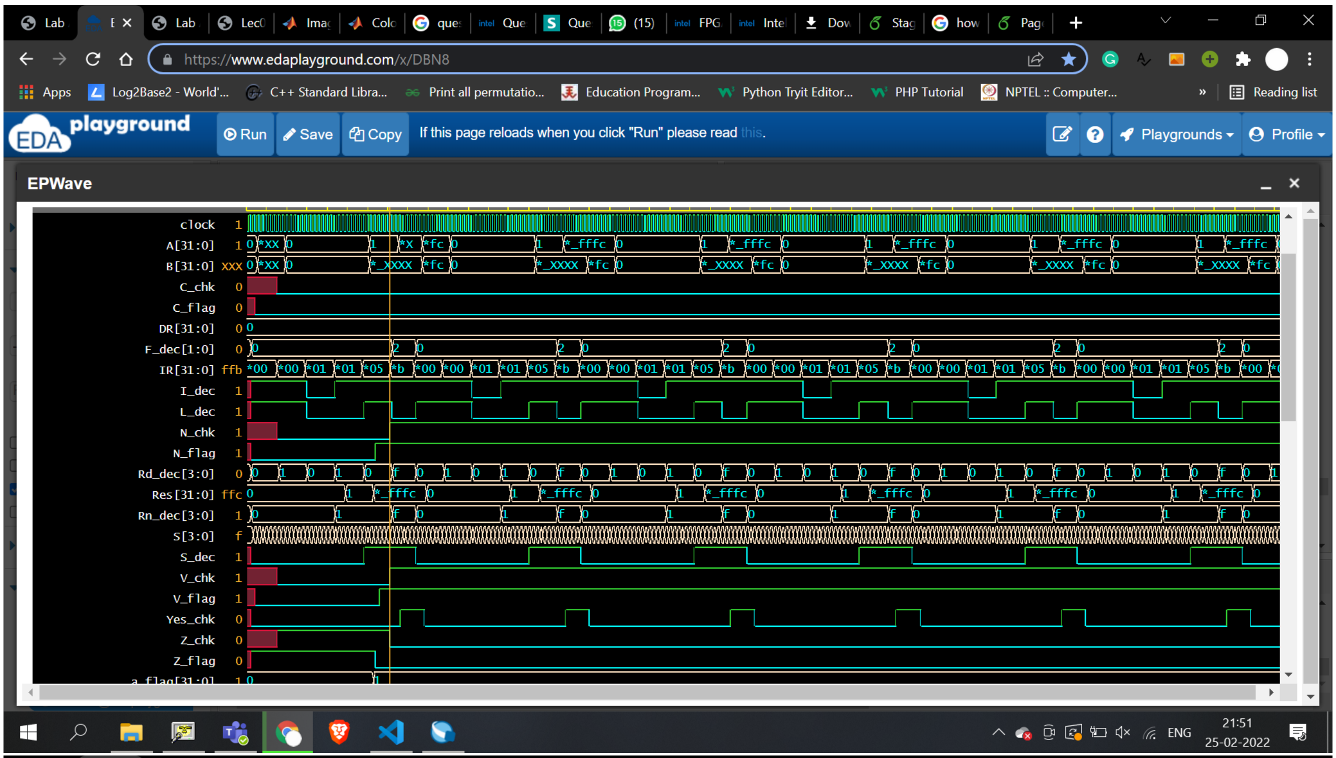
To analyse the image you can check the values of doutdatamem and dout(1/2)reg value.

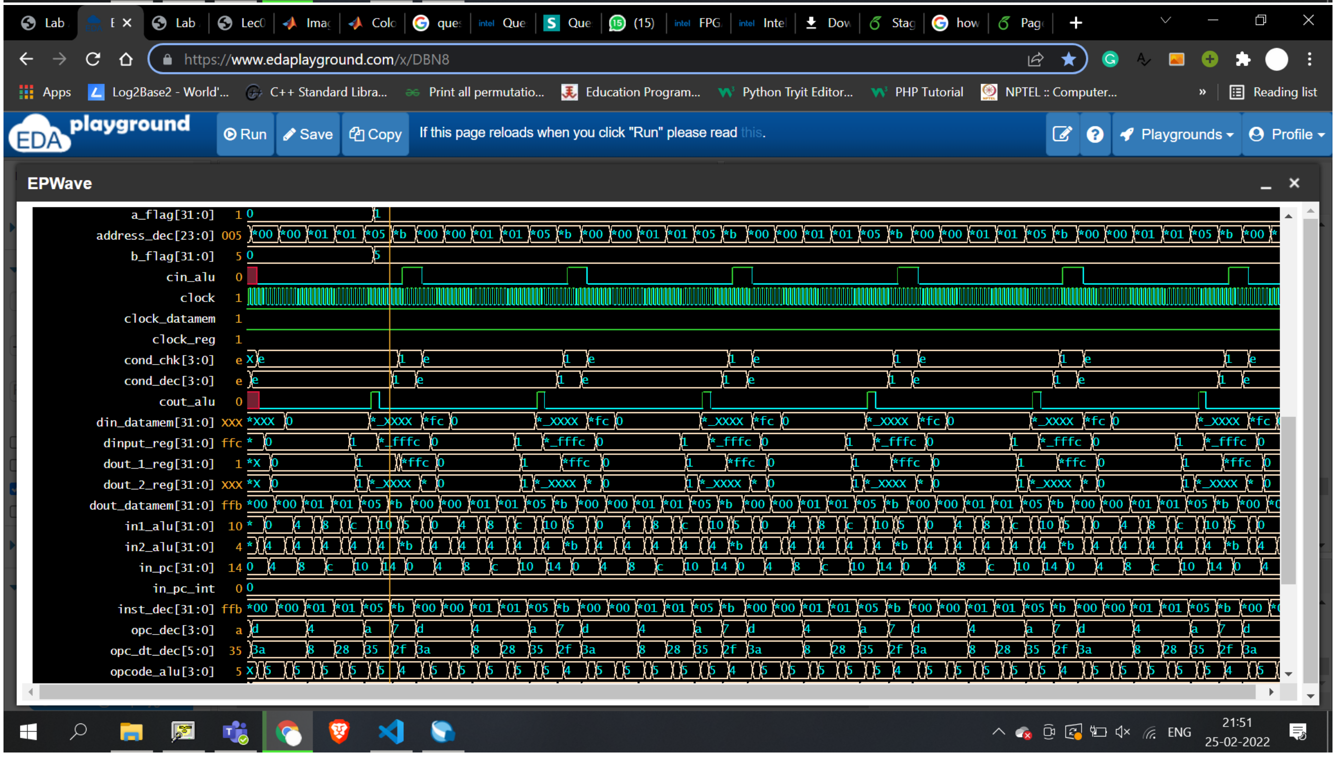


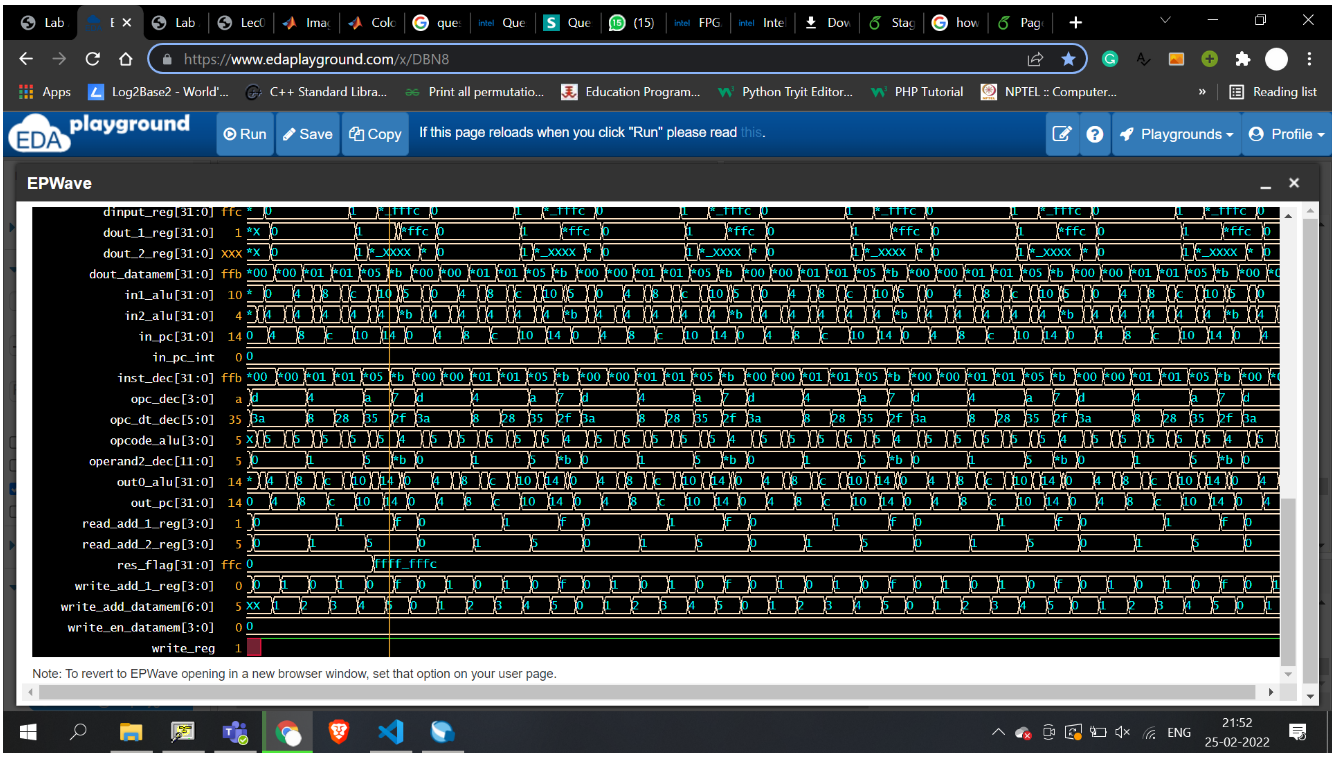
5 Test-Case-2

signal memarray : memorytype := (0 = X"E3A00000", 1 = X"E3A01000", 2 = X"E0800001", 3 = X"E2811001", 4 = X"E3510005", 5 = X"1AFFFFFB", others = X"00000000");

5.1 EP-WAVE IMAGE FOR THE ABOVE CASE






https://www.edaplayground.com/x/DBN8
EDA playground
Run
Save
Copy
If this page reloads when you click "Run" please read this.
Playgrounds
Profile
EPWave

```
      dinput_reg[31:0] ffc
       dout_1_reg[31:0]   1
       dout_2_reg[31:0] xxx
    dout_datamem[31:0] ffb
          in1_alu[31:0]  10
          in2_alu[31:0]   4
            in_pc[31:0]  14
              in_pc_int   0
         inst_dec[31:0] ffb
           opc_dec[3:0]   a
        opc_dt_dec[5:0]  35
        opcode_alu[3:0]   5
     operand2_dec[11:0]   5
         out0_alu[31:0]  14
           out_pc[31:0]  14
     read_add_1_reg[3:0]   1
     read_add_2_reg[3:0]   5
        res_flag[31:0] ffc
    write_add_1_reg[3:0]   0
  write_add_datamem[6:0]   5
   write_en_datamem[3:0]   0
              write_reg   1
```

Note: To revert to EPWave opening in a new browser window, set that option on your user page.