Assignment-2 Stage-7 Submission

Indian Institute of Technology, Delhi

COL216: Computer Architecture

1 Introduction

The overall assignment is designing hardware for implementing a processor that can execute a subset of ARM instructions. This is the stage 7 of the assignment. In this stage, I have implemented multiply group instructions.

2 Stage 7

2.1 Objective

The objective of this stage was to support multiply group instructions. 6 kind of multiply instructions have been implemeneted: mul, mla, smull, smlal, umull, umlal.

2.2 Assumptions

I have used Edaplayground to write and test my code. EdaPlayGround has been used to Simulate the code using Aldec Riviera Pro tool. Quartus was downloaded and used to synthesize the design. Results of both of these have been attached.

For this design, I have assumed that the Program Counter is a separate register and has nothing to do with R15 of register file.

I have assumed Data Memory is the only memory that is used both for data and program instructions. This memory has a total of 128 registers each of 32 bits. The first 64 registers (0 to 63) take care of the data memory and the next 64 registers take care of the program instructions.

For the purposes of this stage, I have assumed that the instructions are hardcoded in the data memory, and as soon as a X"00000000" instruction is encountered, the program halts. The program can be easily modified though to take the instructions through the testbench.

All the executions are done on the rising edge of the clock, except the flags, that are set on the falling edge of the clock. This helps them to be available for the next instruction.

I have not set flags for Multiplier output in this stage, as this was not mentioned. Also, I have made new control states for the multiply group instructions.

2.3 Logistics

The project directory has the following category of files:

1. **<u>VHDL files:</u>** In Stage 7, Multiplier.vhd has been added whose structure is similar to that given in the assignment statement. This component works independent of ALU and doesn't involve ALU for accumulation.

Other than these files, there are other VHDL files which were already submitted in stage 6. Some minor changes were made in FSM, and control signals to accomodate the changes.

Some new control signals were also added, to enable the read and write of registers from new part of the instruction. The controller and datapath were also changed.

The files already submitted in stage 6 are:

a) TestBench.vhd : This file has the top level entity testbench. The logic of this file is just a clock with a period of 1 ns, which is input to the processor,

b) Processor.vhd : This file is the crux of this assignment. It has the overall logic of a processor, It has 3 component instantiations i.e. a datapath, a controlpath and FSM. The data path has control signals as input and status signals as output, while the controlpath has control signals as output and status signals and the state as input. The FSM has decoded instruction has the input, and outputs the next state, using the local signal of present state. The intial value of this present state is set to 0. The processor only takes a clock as an input and gives out no output.

c) Datapath.vhd : This file has the overall logic of datapath. It has many component instantiations, all of which receive control signals by the controller.

d) flags.vhd : This file contains the logic of resetting flags on each clock edge depending on the s-bit of the instruction.

e) program_counter.vhd : This file contains the PC register. It takes as input the ”next” value of the program counter register, which is written in it, but only available in next clock edge. The initial value of data_in is set as 0 and that of data_out is -4, so that it is synced.

f) condition_checker.vhd : Contains the logic of updating the predicate value, p. This value along with the present state of flags is used to decide whether to branch or not.

g) Decoder.vhd : Given an instruction, it decodes the instruction based on its different fields.

h) MyTypes.vhd : This file contains the package of types that have been used in this project to easily identify different types and make the code more readable.

i) Controller.vhd : The overall control logic is in this file. On input of instruction and flag status, it returns control signals as an output to the datapath.

j) FSM.vhd : This file contains the state logic. It has a local signal of present state and based on that, and the input decoded instruction, it decides the next state. k) Shifter.vhd (And other shifter files that shift by 1, 2, 3, 4 and 5 positions). l) PMConnect.vhd

The other 3 files are the same as submmited in stage 1. No changes were made in ALU, data_memory and register_file.

2. **ARM files:** The directory also contains some .s files, that were used to test the processor (apart from the examples already tested in stage 3 and 4).

3. **Result files:** There are multiple PNG and PDF files that contain the results of simulation and synthesis.

Apart from all these files, there are some standard files that were generated by Quartus to test during synthesis.

2.4 Contributions

The MyTypes and Decoder file was taken from Moodle. No other help of any form was taken from anybody/anywhere.

2.5 FSM

The FSM was modified to accomodate the multiply group instructions. The new FSM now looks like:

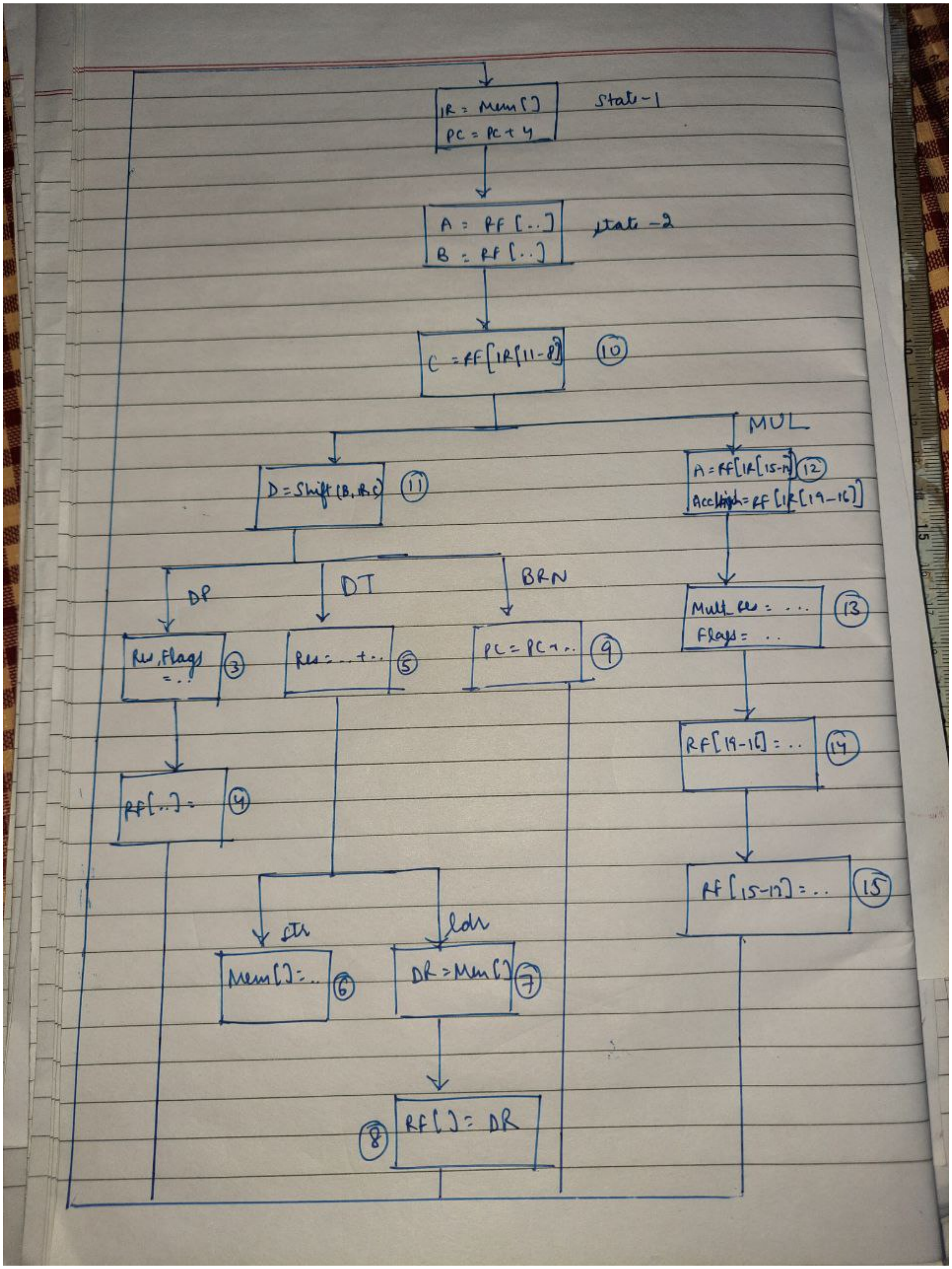
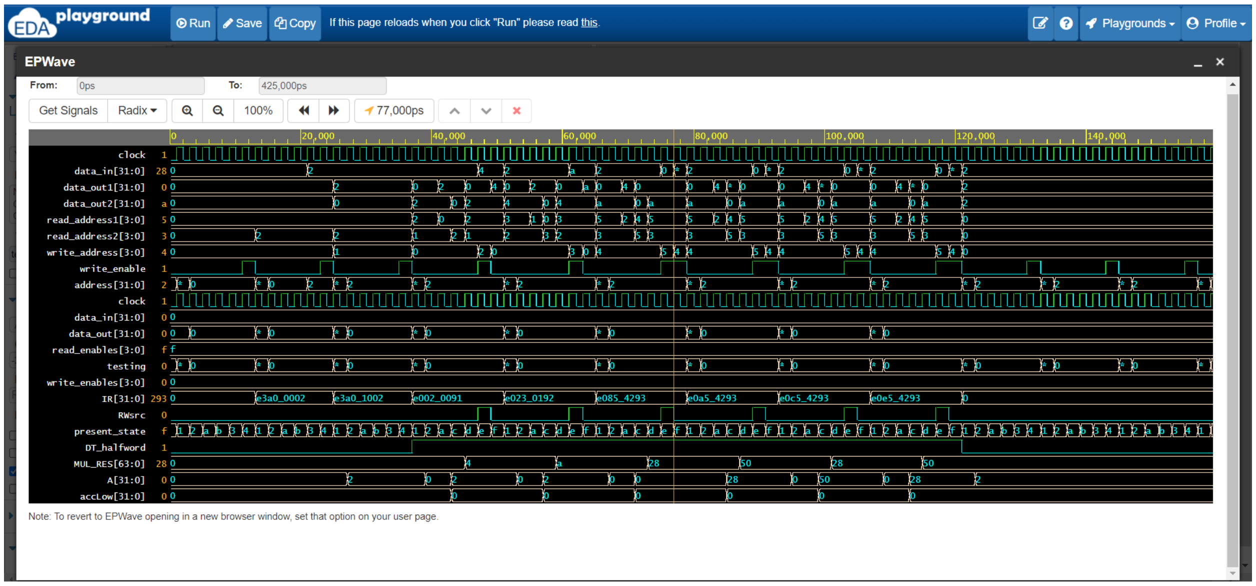


Figure 1: New Finite State Machine (FSM)

2.6 Test Cases

The program was tested on many different test cases.

The test cases were designed to exhaustively cover all the cases.
I also tested the code on previously submitted test cases, and got correct outputs.
The main directory of the assignment contain 3 new test cases: multiply_testing.s, multiply_hard.s and random_test.s. The other test cases are the same used in stage 6, and were tested again. They were found to give correct outputs.
As explained above in the ARM files, these were the EPWaves corresponding to each instruction set.



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

Figure 2: Output for file multiply_testing.s

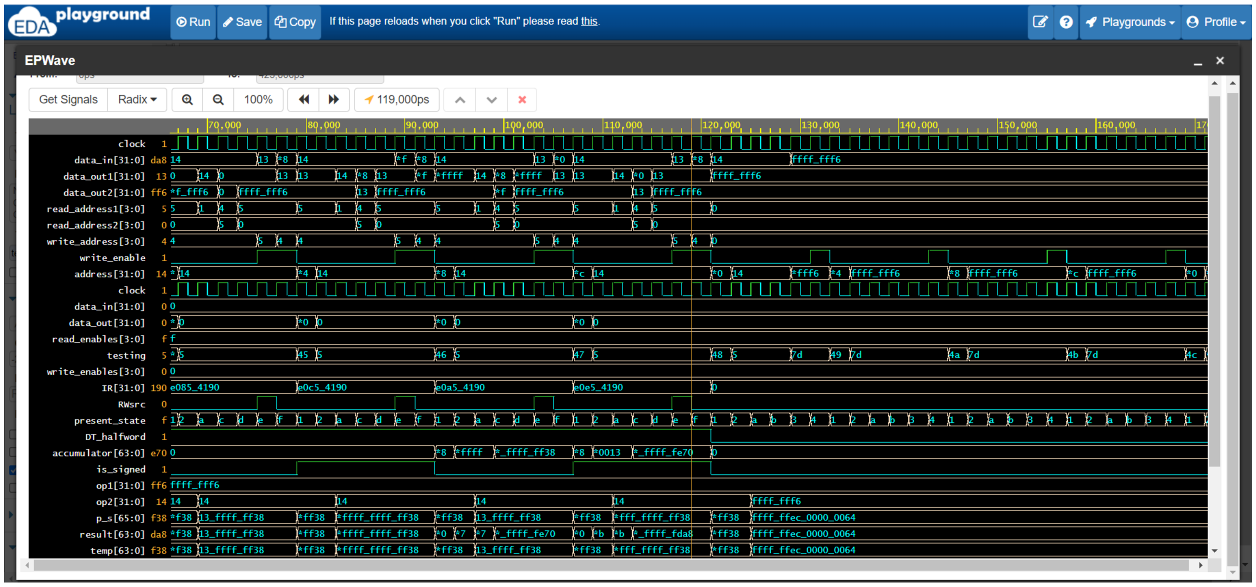


Figure 3: Output for file multiply_hard.s

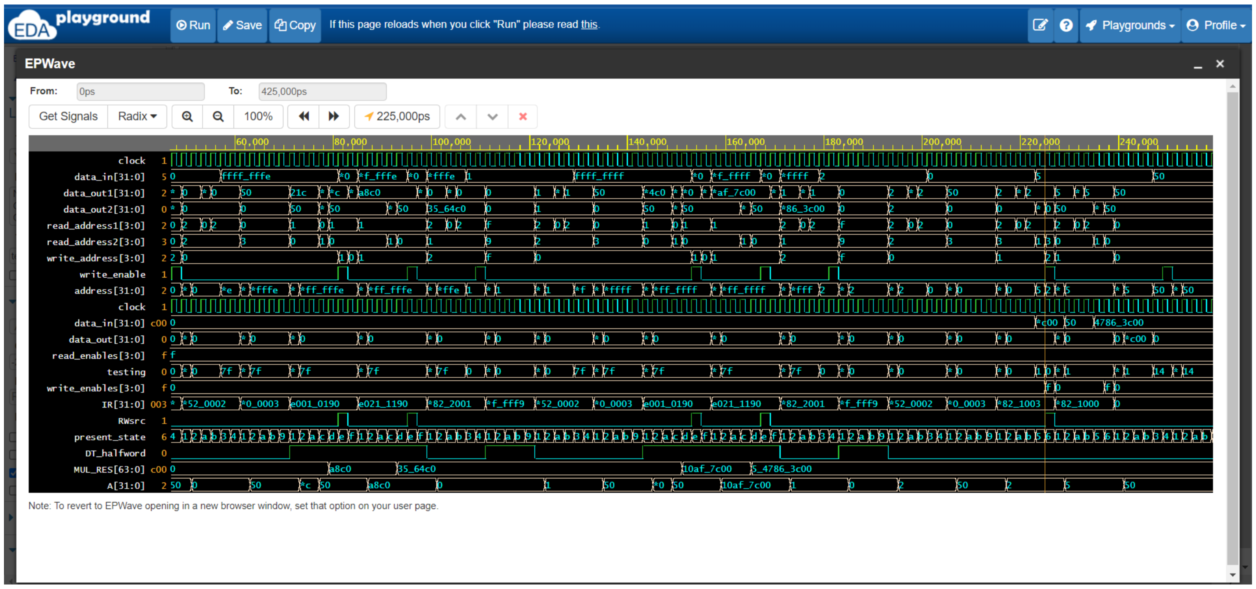
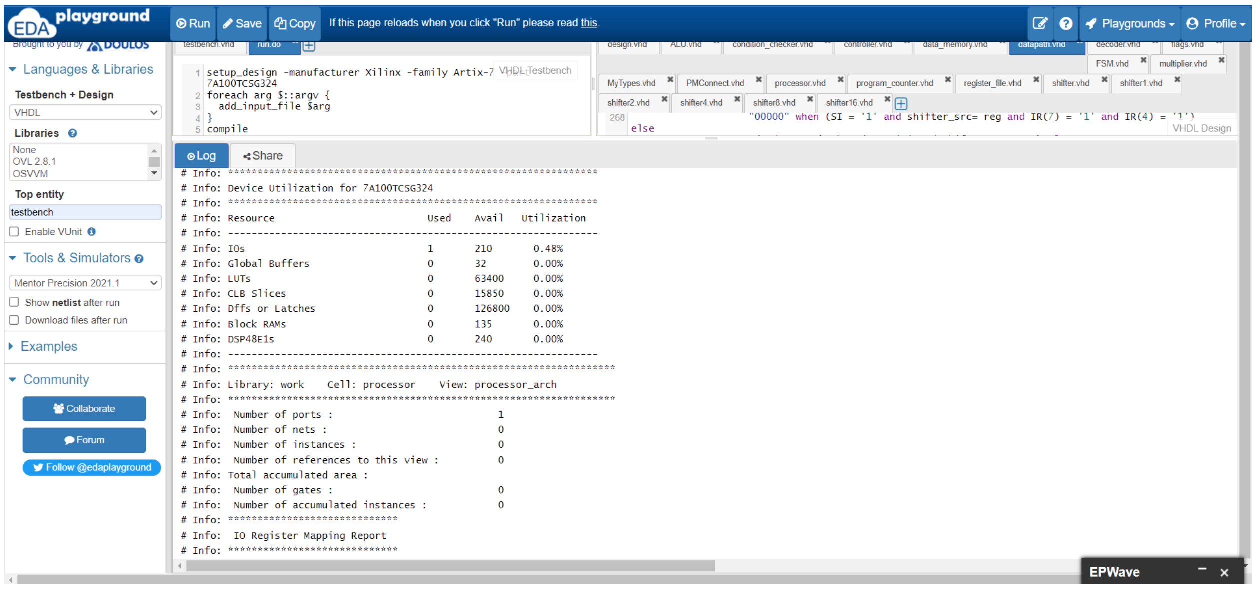


Figure 4: Output for file random_test.s



Testbench + Design: VHDL

Libraries: None, OVL 2.8.1, OSVVM

Top entity: testbench

Enable VUnit: ☐

Tools & Simulators: Mentor Precision 2021.1

Show **netlist** after run: ☐

Download files after run: ☐

```
setup_design -manufacturer Xilinx -family Artix-7
7A100TCSG324
foreach arg $::argv {
  add_input_file $arg
}
compile
```

```
268                 "00000" when (SI = '1' and shifter_src= reg and IR(7) = '1' and IR(4) = '1')
      else
```

```
# Info: ***************************************************************
# Info: Device Utilization for 7A100TCSG324
# Info: ***************************************************************
# Info: Resource                          Used    Avail   Utilization
# Info: ---------------------------------------------------------------
# Info: IOs                               1       210       0.48%
# Info: Global Buffers                    0       32        0.00%
# Info: LUTs                              0       63400     0.00%
# Info: CLB Slices                        0       15850     0.00%
# Info: Dffs or Latches                   0       126800    0.00%
# Info: Block RAMs                        0       135       0.00%
# Info: DSP48E1s                          0       240       0.00%
# Info: ---------------------------------------------------------------
# Info: ******************************************************************
# Info: Library: work    Cell: processor    View: processor_arch
# Info: ******************************************************************
# Info:  Number of ports :                            1
# Info:  Number of nets :                             0
# Info:  Number of instances :                        0
# Info:  Number of references to this view :          0
# Info: Total accumulated area :
# Info:  Number of gates :                            0
# Info:  Number of accumulated instances :            0
# Info: *****************************
# Info:  IO Register Mapping Report
# Info: *****************************
```

Figure 5: Synthesis Report of EDAPlayground

Though these EPWaves were really difficult to analyse, I went through each instructions, and ensured that the value at each port was correct. As a final check, the value at register_file ports can be checked, which has been highlighted in some of the screenshots.

The program works fine on all the test cases(both new and old). The required implementations were successfully included.