Assignment-2 Stage-5 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 5 of assignment. In this stage, I have integrated a shifter with the overall design. This shifter supports 4 types of shift instructions, LSL, LSR, ASR and ROR.

2 Stage 4

2.1 Objective

The objective of this stage was to build a shifter, which shifts the input by an amount ranging from 0 bits to 31 bits. This was done by building 5 independent shifters, each of which shifts the input by a constant amount given by 2^i where i = 0, 1, 2, 3, 4. This shifter was then integrated with the overall design, so that the shift operation could be used with the DP instructions and DT instructions.

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.

2.3 Logistics

The project directory has the following category of files:

1. **VHDL files:** In Stage 5, following new files have been added:

a) Shifter1.vhd: This file contains the logic of shifting the given input in right position by 1 bit. The new most significant bit has been taken as input. This has been done to ensure a common logic for ASR, LSR and ROR.

b) Shifter2.vhd: This file contains the logic of shifting the given input in right position by 2 bits. The 2 new most significant bits have been taken as input. This has been done to ensure a common logic for ASR, LSR and ROR.

c) Shifter4.vhd: This file contains the logic of shifting the given input in right position by 4 bits. The 4 new most significant bits have been taken as input. This has been done to ensure a common logic for ASR, LSR and ROR.

d) Shifter8.vhd: This file contains the logic of shifting the given input in right position by 8 bits. The 8 new most significant bits have been taken as input. This has been done to ensure a common logic for ASR, LSR and ROR.

e) Shifter16.vhd: This file contains the logic of shifting the given input in right position by 16 bits. The 16 new most significant bits have been taken as input. This has been done to ensure a common logic for ASR, LSR and ROR.

f) Shifter.vhd: This file finally integrates all the shifters implemented above by instantiating each of the components. The most significant bits have been passed considering the type of the shifting required. For LSL shifts, the input and output have been reversed, before and after passing through the shifter respectively.

Other than these files, there are some VHDL files which were already submitted in stage 3 and 4. Some minor changes were made to accomodate the shifter. For example, 2 new states were added in the FSM, one for loading the data from 3rd register and the other for using the shifter. These states have been numbered 10 and 11 respectively. My new FSM looks like this:

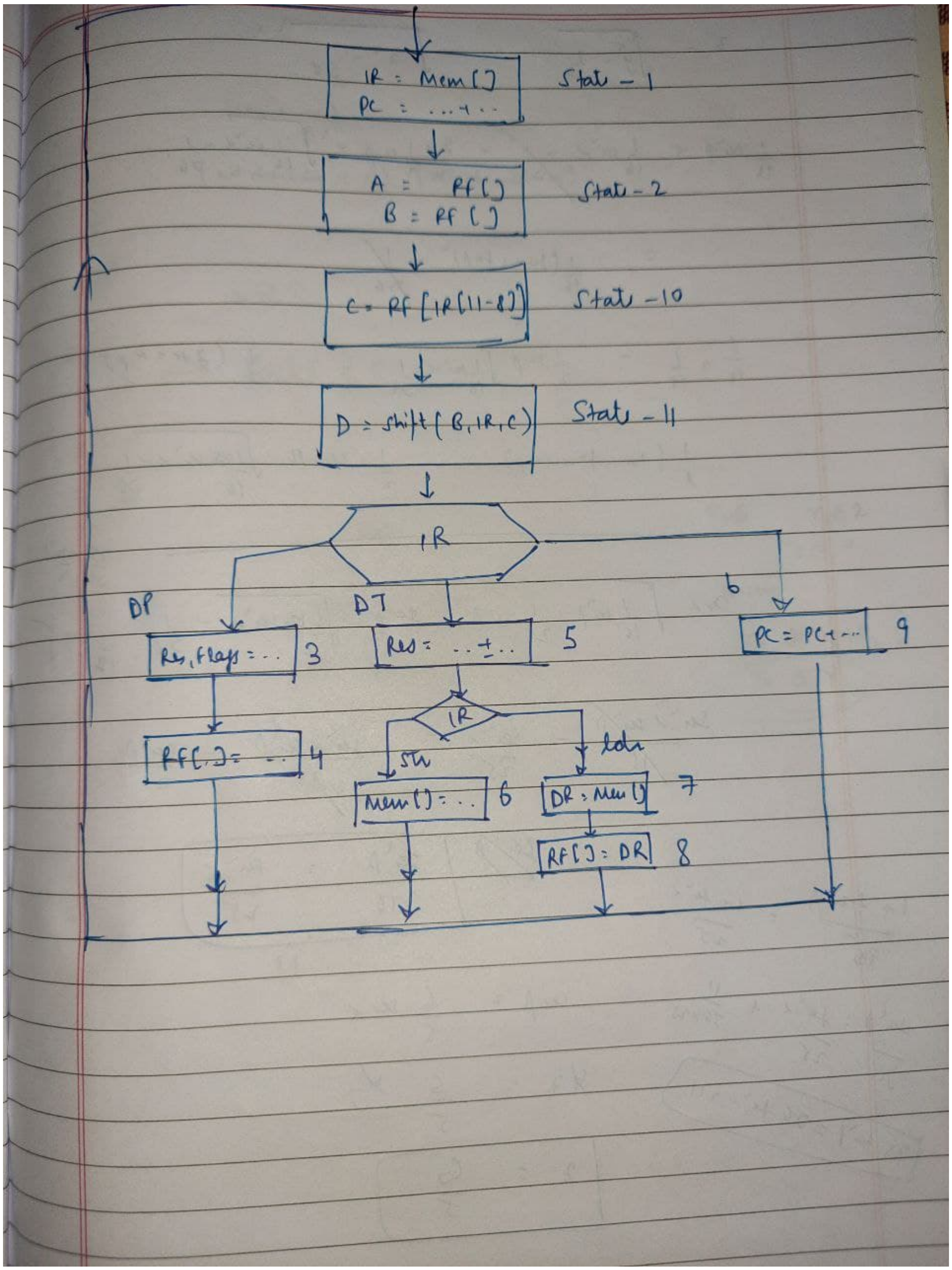


Figure 1: New FSM after shifter integration

Some new control signals were also added, to enable the writing of new registers. The controller and datapath were also changed to accomodate the new states in the Finite State Machine.

The shifter has been added between the register file and ALU. This shifter takes an input, either from the register file, or directly from the instruction, and places the output in a new register, called D. When Asrc2 = "00", the operand 2 for ALU is then read from this register D. This is done when either the instruction is a DP instruction, or the instruction is a DT instruction but with a shifted offset. For normal offsets without shifts, Asrc2 is set to "10".

The files already submitted in stage 3 and stage 4 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.

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 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.
As explained above in the ARM files, these were the EPWaves corresponding to each instruction set.

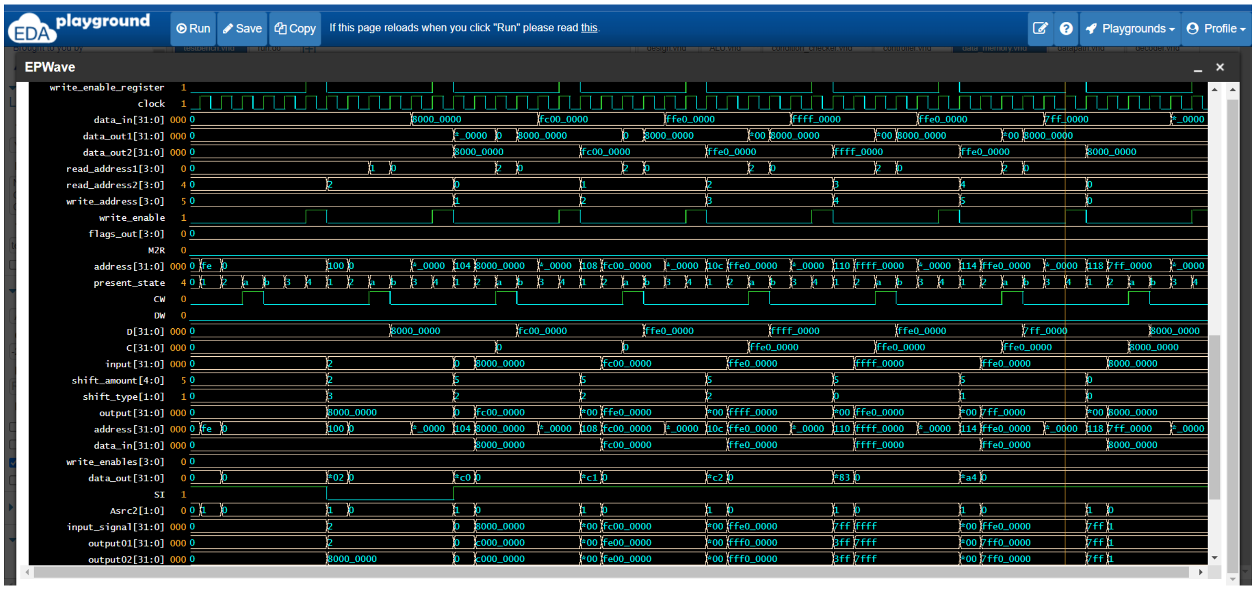


Figure 2: Output for file ASR_test.s

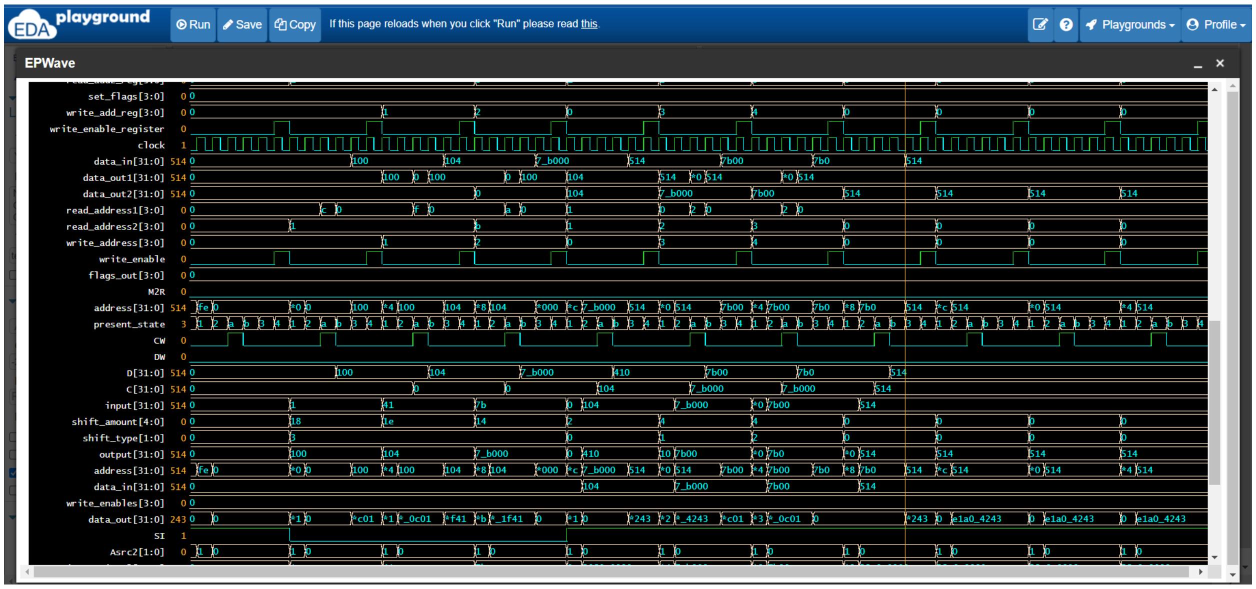


Figure 3: Output for file ROR_test.s

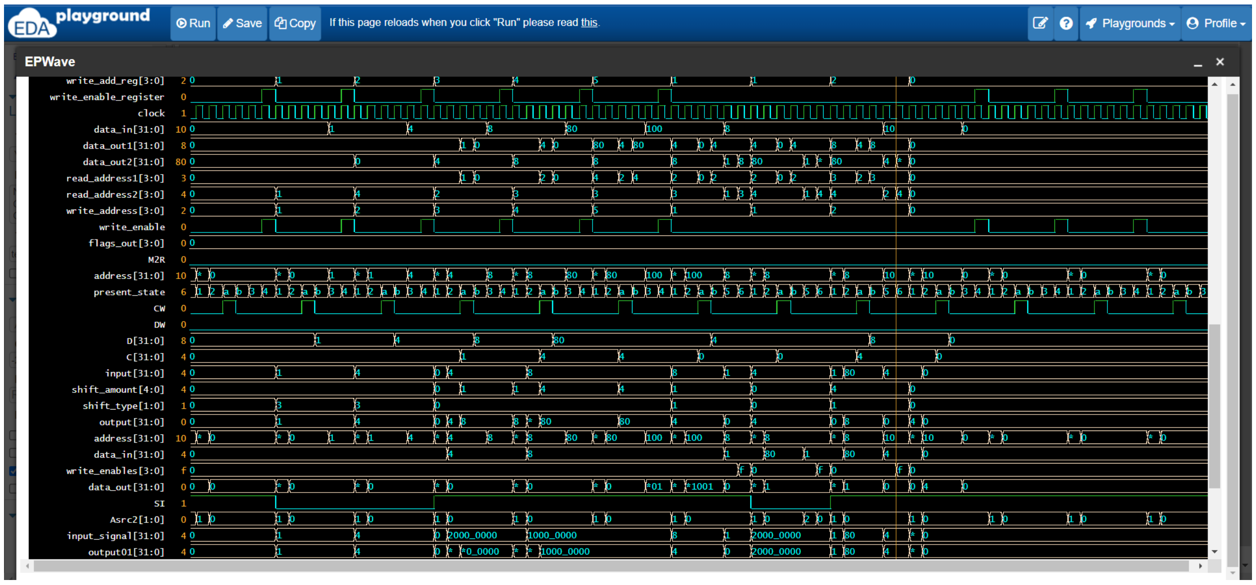


Figure 4: Output for file DT_test.s

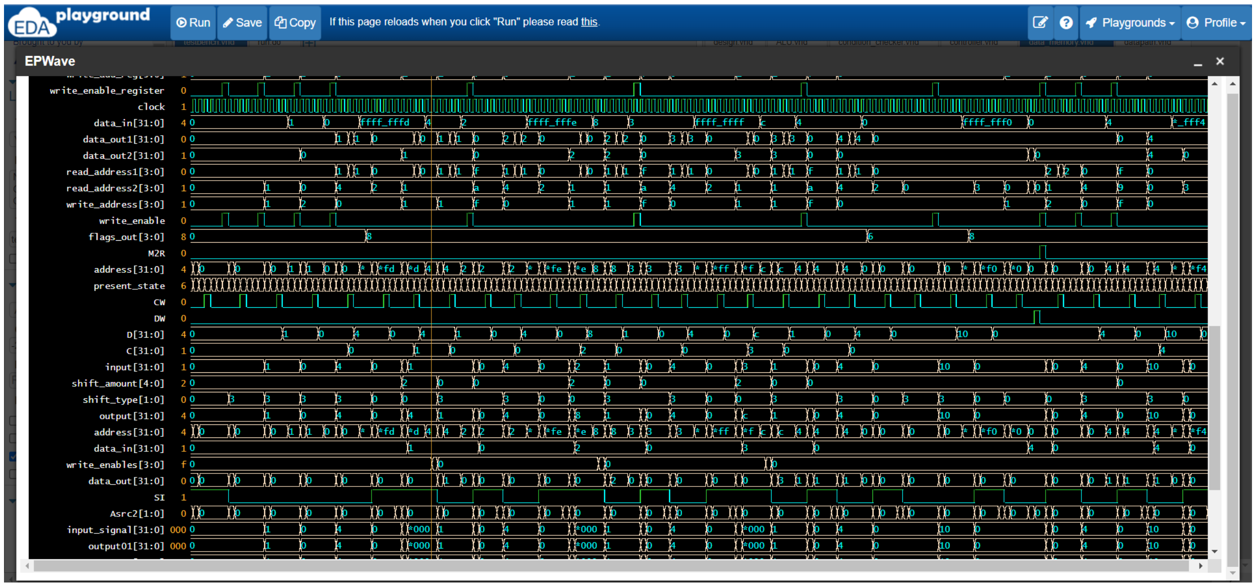


Figure 5: Output for file load_store_test.s

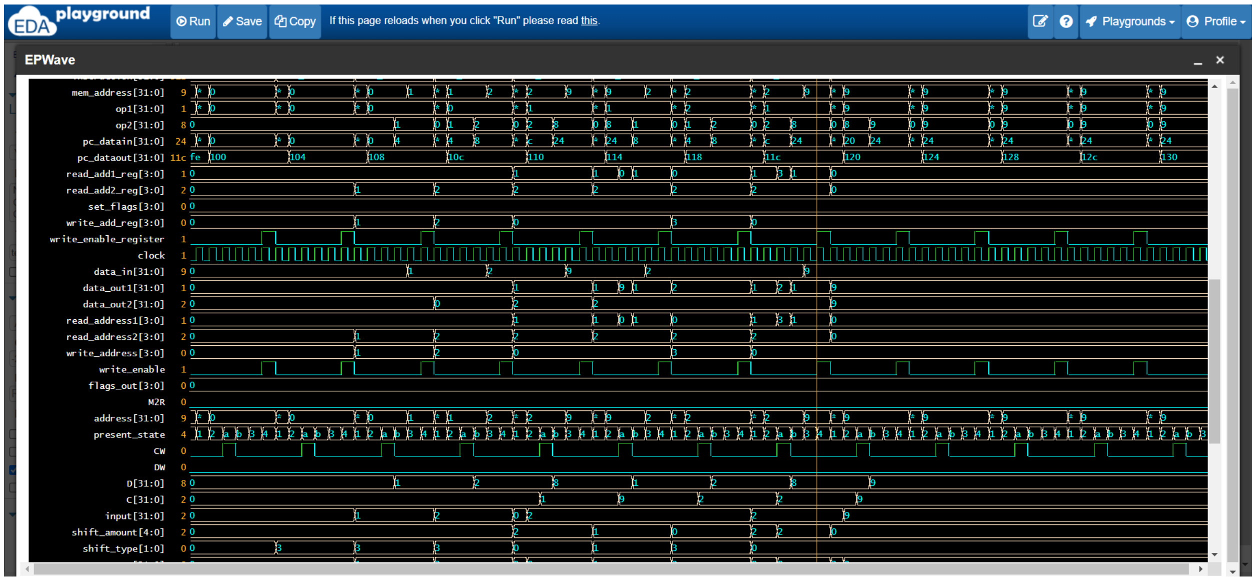
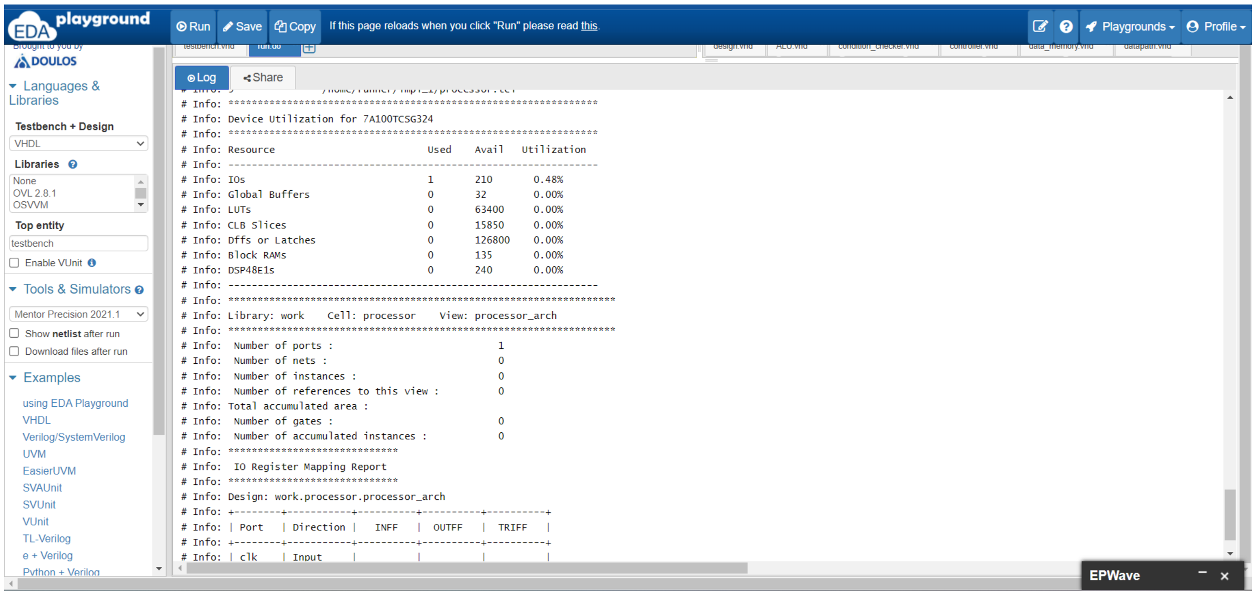


Figure 6: Output of file shift_test.s



Testbench + Design: VHDL

Top entity: testbench

Tools & Simulators: Mentor Precision 2021.1

```
# 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: *****************************
# Info: Design: work.processor.processor_arch
# Info: +--------+-----------+----------+----------+----------+
# Info: | Port   | Direction |   INFF   |  OUTFF   |  TRIFF   |
# Info: +--------+-----------+----------+----------+----------+
# Info: | clk    | Input     |          |          |          |
```

Figure 7: 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.