COL216 Assignment 2

Stage 5

Rishabh Dhiman
2020CS10837

12 March 2022

1 Objective

Test the processor designed in previous stages using ARM assembly codes.

2 Technical Details

- The VHDL code was analyzed, and simulated using GHDL 1.0.0.
- The waveform viewer used is GTKWave Analyzer v3.3.104.
- The VHDL code was synthesised, and netlist generated using Quartus 21.1.

3 Directory Structure

The directory structure reflects the previous submissions. The directory structure of this submission is as given on the next page.

The following VHDL component files are new or have been updated,

- `processor.vhdl`,
- `shifter.vhdl`,
- `types.vhdl`, and
- `decoder.vhdl`.

`shifter_tb.vhdl` defines a testbench for testing the new `shifter` component.

A new assembly program, `test-dp-rot.s` has been created to test the DP rotation/shift operations. A C++ program to verify it's output has also been created. Along with this, `neg-fib.s` has been updated to use LDR register offset instructions. The corresponding binary representation of the instructions, and the waveform created on testing have also been added.

The netlist of the entire processor along with the new shifter component have also been attached. The states of the control FSM have also been attached.

```
2020CS10837-assgn-2-stage-5/
├── stage-5-report.pdf
├── code/
│   ├── makefile
│   ├── arm/
│   │   ├── assembly/
│   │   │   ├── neg-fib.s
│   │   │   └── test-dp-rot.s
│   │   ├── binary/
│   │   │   ├── neg-fib.txt
│   │   │   └── test-dp-rot.txt
│   │   ├── c++/
│   │   │   └── test-dp-rot.cpp
│   │   └── scripts/
│   │       ├── create-mem.sh
│   │       └── to-bin.sh
│   ├── hdl/
│   │   ├── alu.vhdl
│   │   ├── cond_checker.vhdl
│   │   ├── decoder.vhdl
│   │   ├── flag_circuit.vhdl
│   │   ├── memory.vhdl
│   │   ├── processor.vhdl
│   │   ├── program_counter.vhdl
│   │   ├── reg_file.vhdl
│   │   ├── shifter.vhdl
│   │   └── types.vhdl
│   ├── simulation/
│   │   └── arm-tests
│   └── tests/
│       ├── processor_tb.vhdl
│       └── shifter_tb.vhdl
└── output/
    ├── arm-tests/
    │   ├── neg-fib.ghw
    │   ├── neg-fib.pdf
    │   ├── test-dp-rot.ghw
    │   └── test-dp-rot.pdf
    ├── synthesis/
    │   ├── processor_control_state.pdf
    │   ├── processor_netlist.pdf
    │   └── shifter_netlist.pdf
    └── waveforms/
        ├── shifter_tb.ghw
        └── shifter_tb.pdf
```

4 Testing Procdure

The code was analyzed, and simulated using GHDL. It was synthesized using Quartus 21.1.
You can simulate it yourself as follows.

1. Ensure that you are using a Linux machine with Make and GHDL installed.
2. To simulate the shifter testbench,
   a) Create a directory `simulation/` (if it doesn't already exist).
   b) In the commandline, run `make shifter`.
   c) A `shifter_tb.ghw` waveform file will be created in the `simulation/` directory. Note that the earlier file will be overwritten.
3. To run the ARM programs.
   a) Create a directory `simulation/arm-tests/` (if it doesn't already exist).
   b) In the commandline, run `make testarm INPUT=`X where $X \in \{$`neg-fib`, `test-dp-rot`$\}$.
   c) A X`.ghw` waveform file will be created in the `simulation/arm-tests` directory. Note that the earlier file will be overwritten.
4. Finally run `make clean` to delete any temporary files created.

5 Testbench Descriptions

5.1 `shifter_tb`

This is an automated testbench, it's written in VHDL 2008 to make use of the VHDL `ror`, `srl`, `srl`, `sll` operations.

For 4 input numbers, `0x0000_0001`, `0xFFFF_FFFE`, `0x7FA0_C654`, and `0xA649_5839` it performs the 4 possible rotate/shift operations, along with all 32 possible shift amounts, and uses `assert` statements to check that the expected output matches the final output.

6 ARM Program Descriptions

The description of the three programs follow,

6.1 `neg-fib`

This computes the negative Fibonacci numbers, F_{-i}. If $a_i = F_{-i}$, then

$$F_{i+2} = F_{i+1} + F_i \implies F_i = F_{i+2} - F_{i+1} \implies a_i = F_{-i} = F_{-i+2} - F_{-i+1} = a_{i-2} - a_{i-1}.$$

F_{-i} is stored in memory location, $64 + i$, for $0 \leq i \leq 8$.

```
e3a00c01
e3a01000
e5801000
e3a01001
e5801004
e3a01007
e3a02000
e7903102
e2822001
e7904102
e0433004
e2822001
e7803102
e2422001
e1520001
1afffff6
```

```
@ compute negative fibonacci as a[i] = a[i - 2] - a[i - 1]
@  where a = mem[64], a[0] = 0, a[1] = 1

mov r0, #256
@ location of a

mov r1, #0
str r1, [r0]
mov r1, #1
str r1, [r0, #4]
@ set a[0] = 0, a[1] = 1

mov r1, #7
@ ie,
@ a[0] = 0
@ a[1] = 1
@ for (i = 1; i <= 8; ++i)
@   a[i + 1] = a[i - 1] - a[i]

mov r2, #0 @ index variable
loop:
@ r3 = a[i]
@ ++i
@ r4 = a[i]
@ r3 -= r4
@ ++i
@ a[i] = r4

ldr r3, [r0, r2, LSL #2]
add r2, #1
ldr r4, [r0, r2, LSL #2]
sub r3, r4
add r2, #1
str r3, [r0, r2, LSL #2]

@ update things
@ add r0, #4
@ don't update r0
sub r2, #1
cmp r2, r1
bne loop
```

6.2 test-dp-rot

The program does some arbitrary manipulations involving rotate and shift operations, the result can be compared to the ARMSim output to see that they are the same.

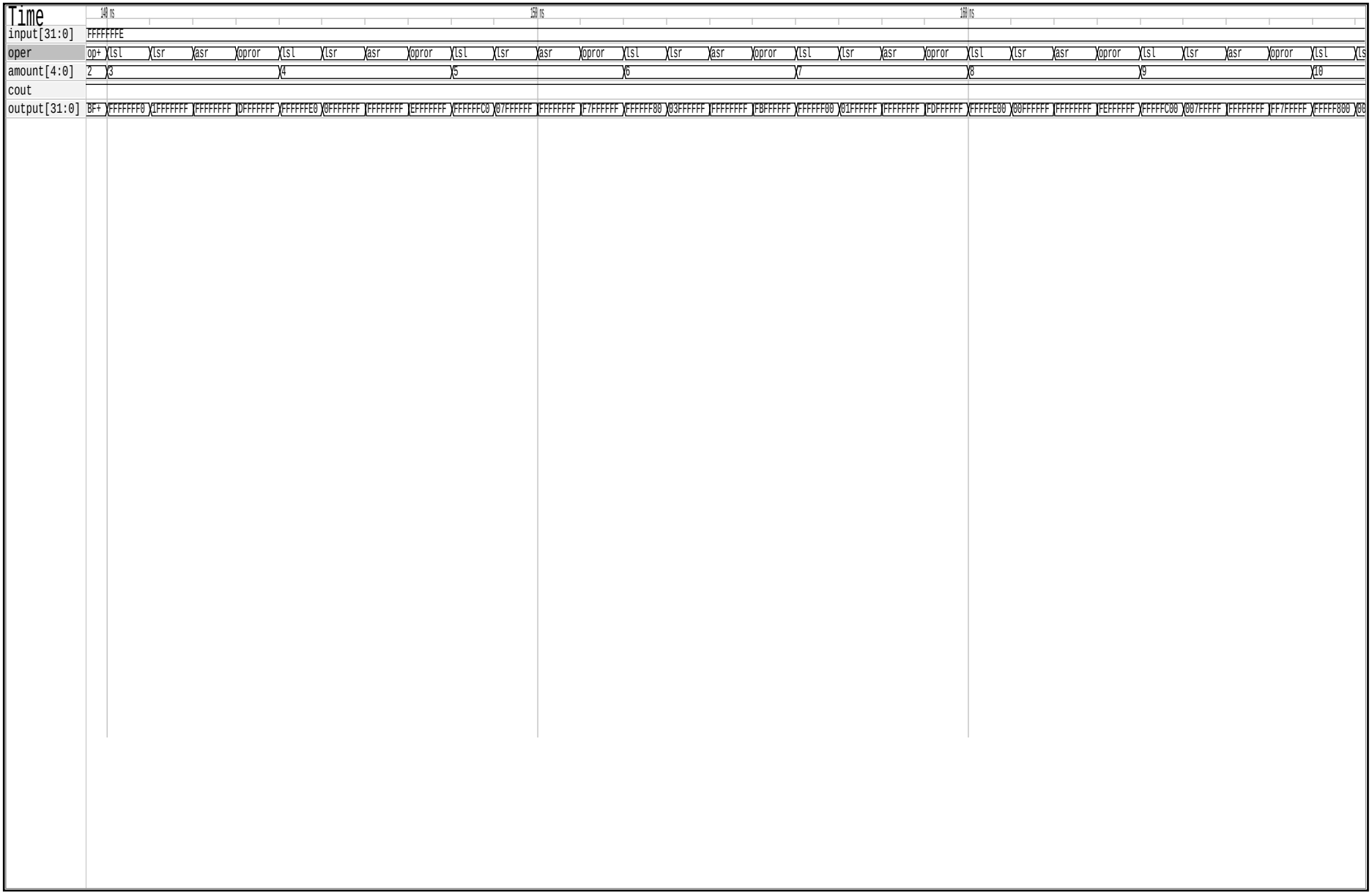
```
                        .text

1   e3a003eb            mov r0, #0xAC000003
2   e1a01360            mov r1, r0, ROR #6
3   e3a02018            mov r2, #24
4   e0813250            add r3, r1, r0, ASR r2
5   e0404383            sub r4, r0, r3, LSL #7
6   e1a020a2            lsr r2, #1
7   e0845230            add r5, r4, r0, LSR r2

                        .end
```

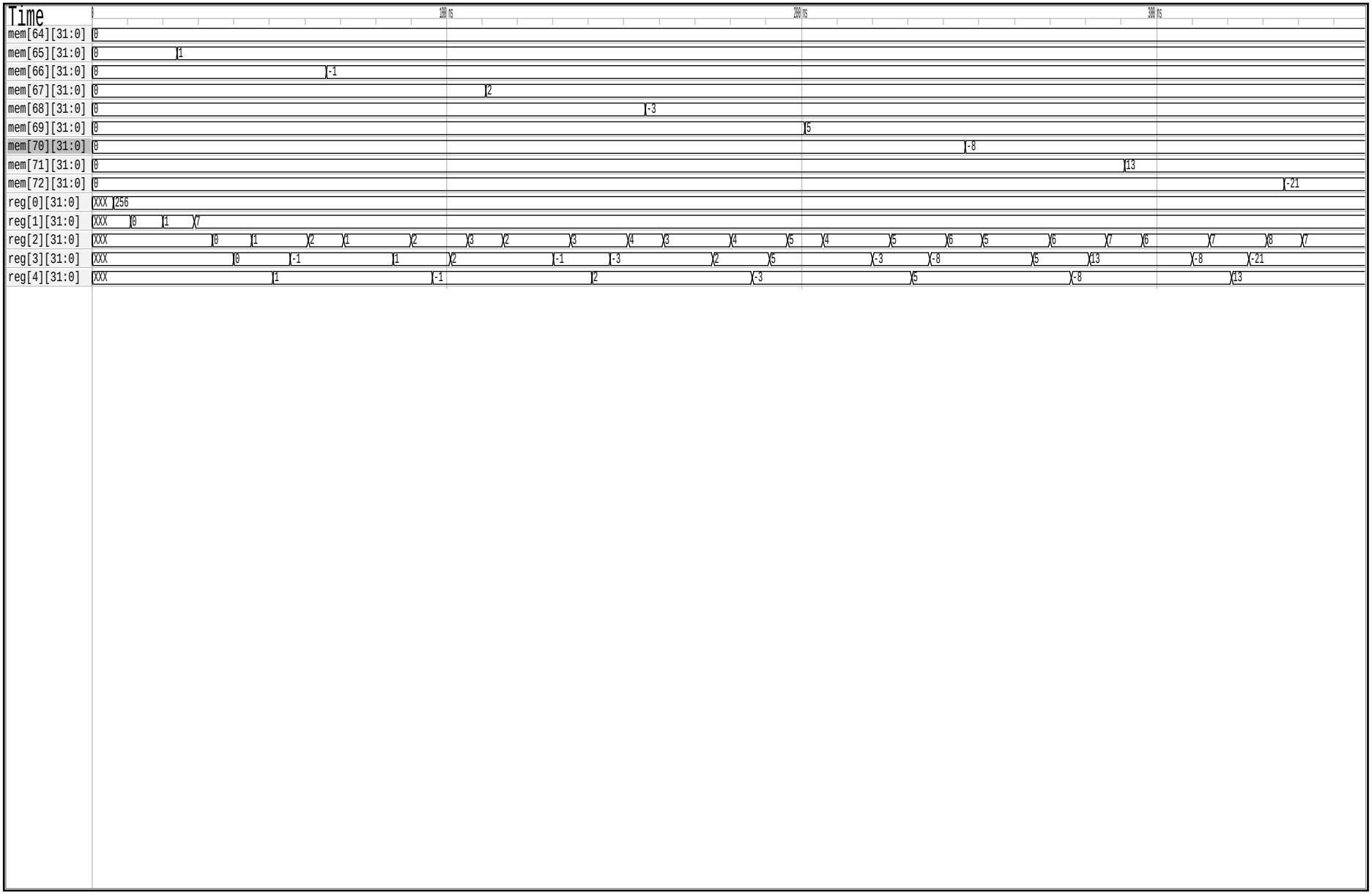
7 Results

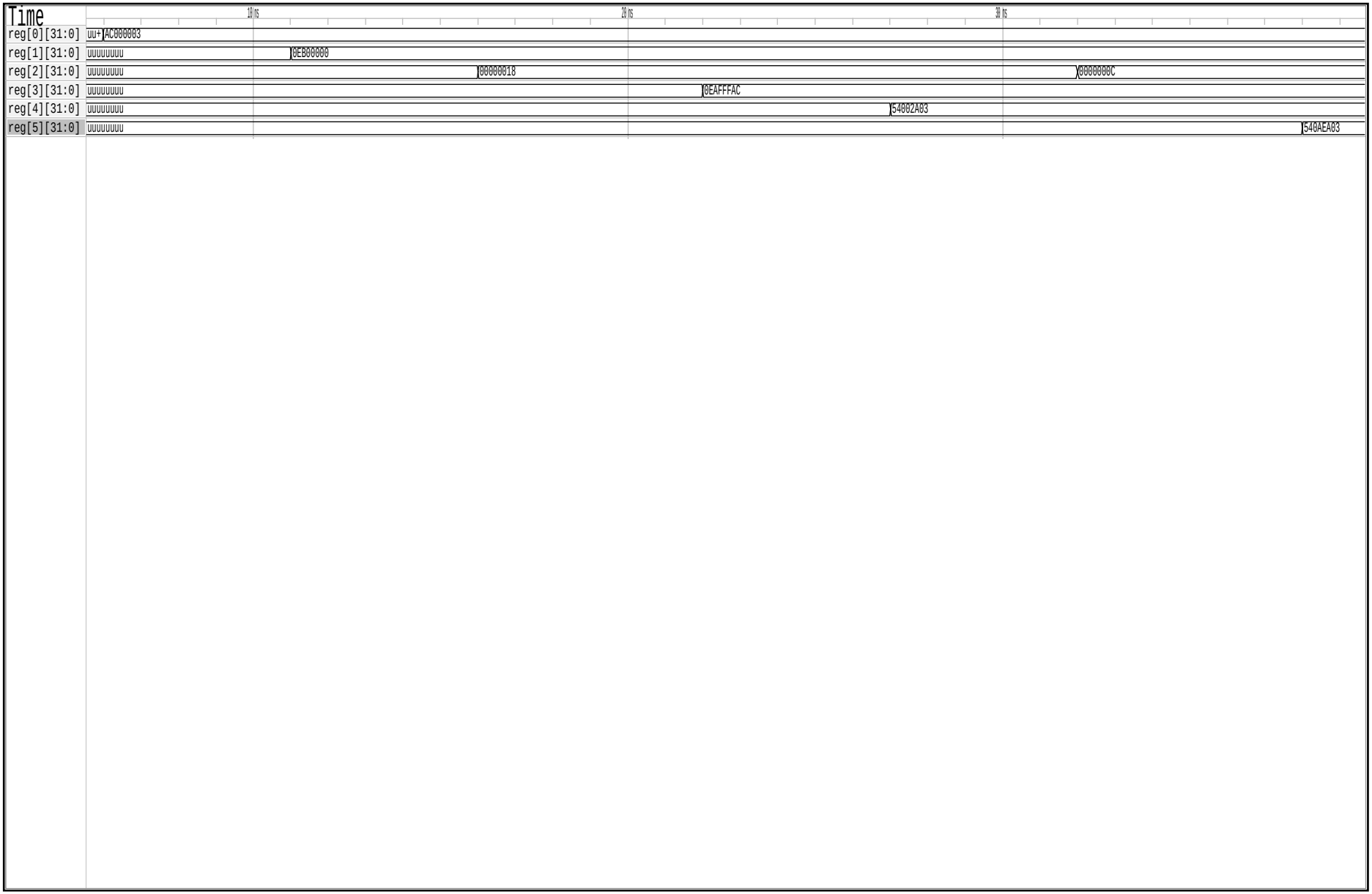
All the testbenches gave the expected results.

The output waveform of the testbench results can be viewed in .ghw and .pdf files in the output folder. The pdf files of the simulation are also attached at the end of this report.

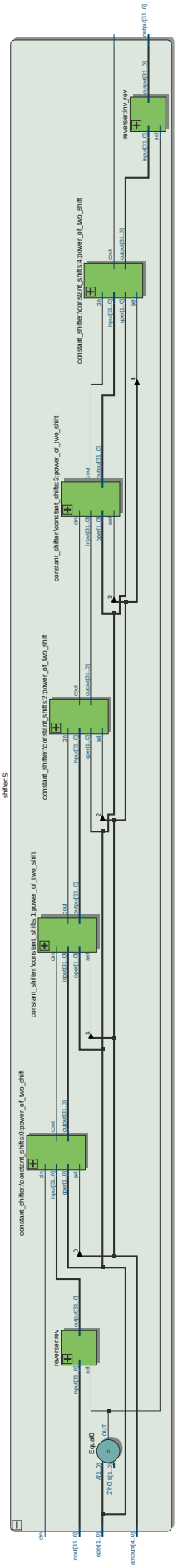
Along with this, the pdf file of the shifter netlist and the states has been added to the end. The processor netlist hasn't been added due to it's large size.


Time
input[31:0]
oper
amount[4:0]
cout
output[31:0]
FFFFFFFE
op+
lsl
lsr
asr
opror
2
3
4
5
6
7
8
9
10
BF+
FFFFFFF0
1FFFFFFF
FFFFFFFF
DFFFFFFF
FFFFFFE0
0FFFFFFF
FFFFFFFF
EFFFFFFF
FFFFFFC0
07FFFFFF
FFFFFFFF
F7FFFFFF
FFFFFF80
03FFFFFF
FFFFFFFF
FBFFFFFF
FFFFFF00
01FFFFFF
FFFFFFFF
FDFFFFFF
FFFFFE00
00FFFFFF
FFFFFFFF
FEFFFFFF
FFFFFC00
007FFFFF
FFFFFFFF
FF7FFFFF
FFFFF800


Time
1000 ns
2000 ns
3000 ns
mem[64][31:0]
mem[65][31:0]
mem[66][31:0]
mem[67][31:0]
mem[68][31:0]
mem[69][31:0]
mem[70][31:0]
mem[71][31:0]
mem[72][31:0]
reg[0][31:0]
reg[1][31:0]
reg[2][31:0]
reg[3][31:0]
reg[4][31:0]


Time
10 ns
20 ns
30 ns
reg[0][31:0]
uu+
AC000003
reg[1][31:0]
uuuuuuuu
0EB00000
reg[2][31:0]
uuuuuuuu
00000018
0000000C
reg[3][31:0]
uuuuuuuu
0EAFFFAC
reg[4][31:0]
uuuuuuuu
54002A03
reg[5][31:0]
uuuuuuuu
540AEA03

synthesis/processor_control_state.pdf



synthesis/shifter_netlist.pdf

: