COL216 Assignment 2

Stage 1

Rishabh Dhiman
2020CS10837

10 February 2022

1 Objective

Construct an ALU, a register file, a program memory unit and a data memory unit for a rudimentary ARM processor in VHDL.

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.

3 Documentation

The submission contains four VHDL files defining the various units,

- `types.vhdl`,
- `alu.vhdl`,
- `reg_file.vhdl`, and
- `memory.vhdl`.

Along with four testbenches for testing these units

- `alu_tb.vhdl`,
- `reg_file_tb.vhdl`,
- `program_memory_tb.vhdl`, and
- `data_memory_tb.vhdl`.

Along with the code, an output folder, containing the waveforms on simulating the testbenches, in the form of `.ghw` and `.pdf` files are stored in the output folder.

For units with a clock, actions were performed only at the rising edge of the clock.

types.vhdl

This files defines some custom types for words, half-words and bytes along with an enumerated type for the 16 opcodes. It's taken from Piazza post 163, with minor modifications.

alu.vhdl

It defines the ALU satisfying the specifications given in the problem statement. The entity declaration of the ALU is given by,

```
entity alu is
        port(
                op1, op2: in word;
                result: out word;
                instr: in optype;
                carry_in: in std_logic;
                carry_out: out std_logic);
end alu;
```

where `word` and `optype` data types are defined in `types.vhdl`.

In the case of logical operations, the `carry_out` bit is set to 0.

memory.vhdl

It defines the program memory and data memory satisfying the specifications given in the problem statement.

Since the program memory is hardcoded, I decided to initialize it such that the i-th position stores the value i for ease of testing.

The entity declaration of the program memory is given by

```
entity program_memory is
        port(
                    addr: in word;
                    data_out: out word);
end program_memory;
```

The entity declaration of the data memory is given by,

```
entity data_memory is
        port(
                    addr: in word;
                    write_enable: in std_logic_vector(3 downto 0);
                    clock: in std_logic;
                    data_in: in word;
                    data_out: out word);
end data_memory;
```

Note that though the memory has only 64 locations, I chose to make `addr` have word length, which is 32 here, to ensure compatibility with the rest of the processor.

reg_file.vhdl

It defines the register file satisfying the specifications given in the problem statement. The entity declaration of the register file is given by,

```
entity reg_file is
        port(
        read_addr1, read_addr2, write_addr: in std_logic_vector(3 downto 0);
        data_in: in word;
        write_enable: in std_logic;
        clock: in std_logic;
        data_out1, data_out2: out word);
end reg_file;
```

Note that unlike with the memories, the addresses are 4-bits.

4 Testing Procdure

The code was analyzed and simulated using GHDL.

You can simulate it yourself using the `makefile` provided on.

1. Ensure that Make and GHDL are installed.
2. Navigate to the `src` directory.
3. Create a folder called `output` inside the `src` directory (if it doesn't exist already).
4. In the commandline, run `make` or `make all` to analyze, and then simulate the programs.
5. `.ghw` waveform files will be created in the output directory. Note that the earlier `.ghw` in the output folder will be overwritten.
6. You can finally run `make clean` to delete any temporary files created in the process.

There are 4 testbenches,

- `alu_tb`,
- `data_memory_tb`,
- `reg_file_tb`,
- `program_memory_tb`.

These testbenches follow the rule that ports of the unit being tested are connected to the signals of the same name in the testbench. For example, `signal op1` of the testbench `alu_tb` is connected to the port `op2` of the `alu`.

alu_tb

This tests the ALU by iterating over 3 values of the two operands, `0x7FFF_FFFF`, `0x0000_0000`, and `0x0000_0001`, and all possible opcodes and carry-in bits. Giving a total of $16 \times 2 \times 3 \times 3 = 288$ combination.

`program_memory_tb`

This testbench simply reads the data from the 64 memory addresses sequentially.

`data_memory_tb`

The clock period of the testbench is 1 ns.

This testbench first sequentially goes through all 16 values of the `write_enable` vector from `0000` to `1111`. For each such value of `write_enable`, it iterates through all 64 memory locations and passes the value $256x + i$ to `data_in` of the memory, where x is the current value of `write_enable` when viewed as a binary number.

At the end, it sequentially reads all 64 values stored in the memory.

`reg_file_tb`

The clock period of the testbench is 1 ns.

The testbench first writes to the memory. It sequentially writes in the value i to memory location i. Simultaneously, it reads out the value from memory locations i and $15 - i$. Note that the memory locations read may be uninitialized.

Then, it reads out values from the memory. For every 16×16 pairs of the two read addresses, the value from these locations is read out of the memory.

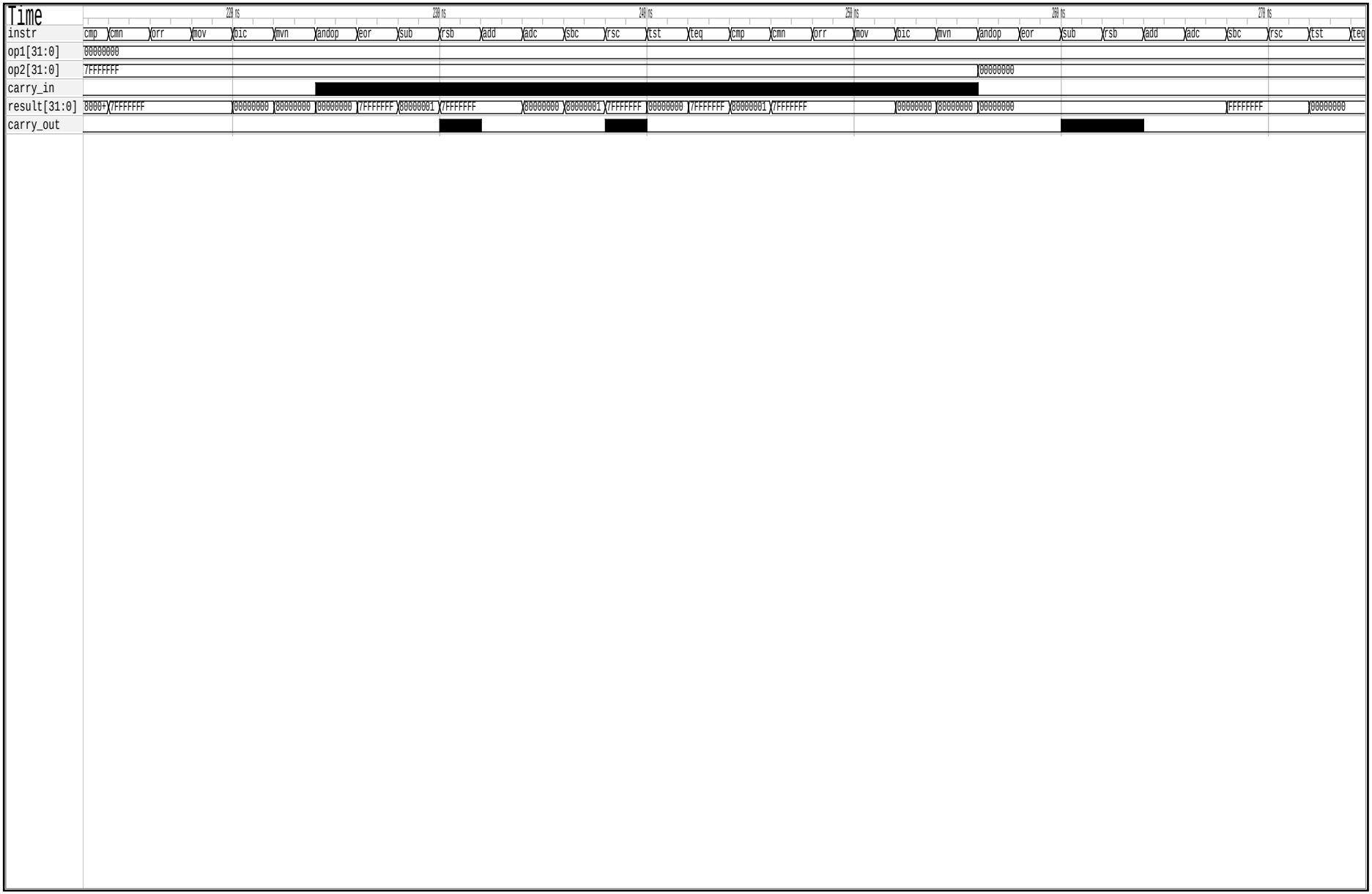
5 Results

The output waveform of the testbench results can be viewed in .ghw and .pdf files in the output folder.

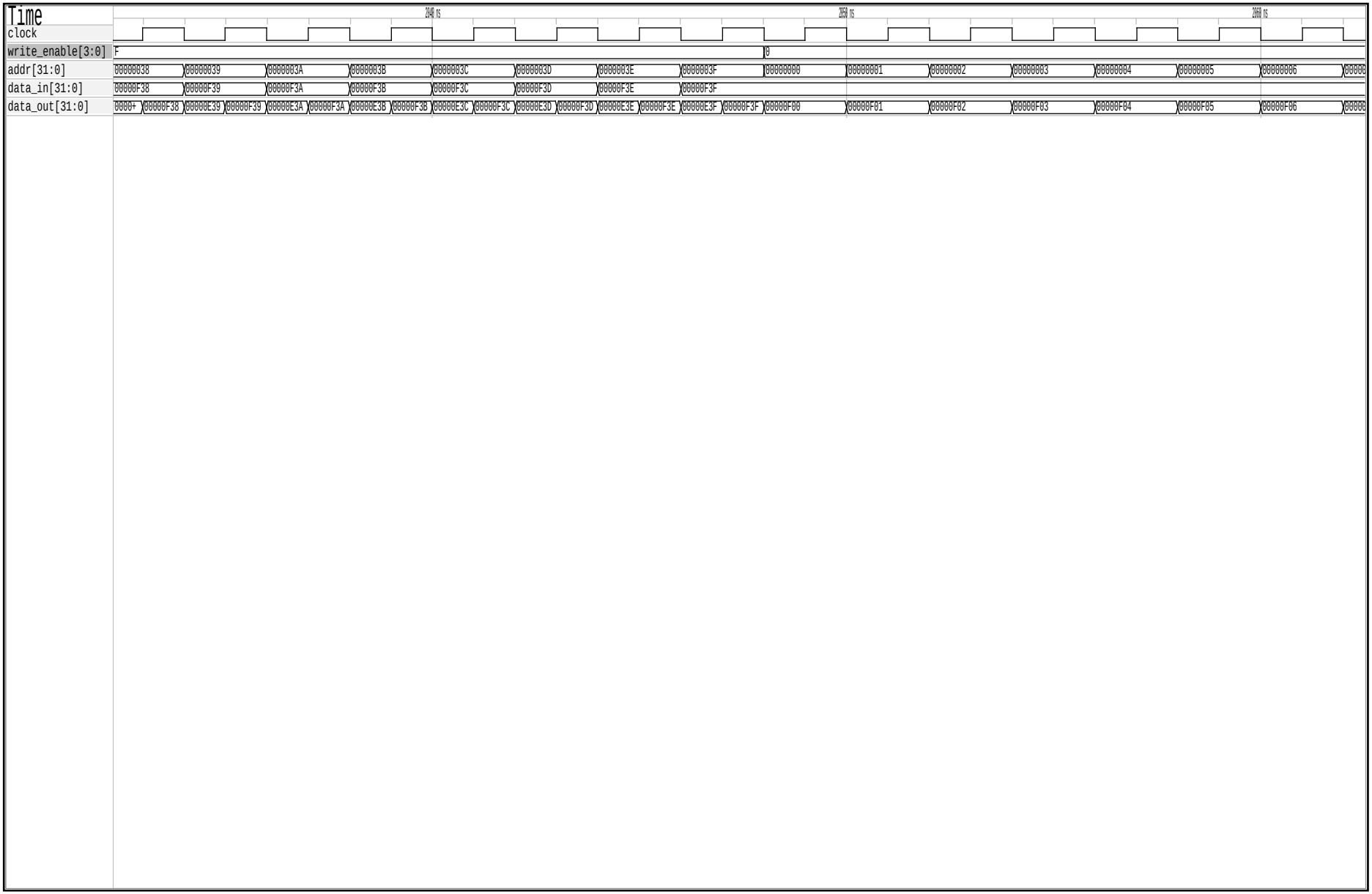
The .pdf files don't contain the entire waveform as the total number of pages required would be prohibitively large. However, the .ghw files can be opened in any waveform analyzer like GTKWave to view the output waveform.

The pdf files are also attached at the end of this report.

alu_tb.pdf


Time
228 ns
238 ns
248 ns
258 ns
268 ns
278 ns
instr
op1[31:0]
op2[31:0]
carry_in
result[31:0]
carry_out
cmp cmn orr mov bic mvn andop eor sub rsb add adc sbc rsc tst teq cmp cmn orr mov bic mvn andop eor sub rsb add adc sbc rsc tst teq
00000000
7FFFFFFF
00000000
8000+ 7FFFFFFF 00000000 80000000 80000000 7FFFFFFF 80000001 7FFFFFFF 80000000 80000001 7FFFFFFF 00000000 7FFFFFFF 80000001 7FFFFFFF 00000000 80000000 00000000 FFFFFFFF 00000000

data_memory_tb.pdf



program_memory_tb.pdf

reg_file_tb.pdf