COL 216 Assignment 2 Stage 1

Manshi Sagar - 2020CS50429

9 February 2022

1 Introduction

In this assignment we aim to design hardware for implementing a processor that can execute a subset of ARM instructions .The designs are to be expressed in VHDL and then simulated and synthesized.
In Stage 1, we design the skeleton of four modules: ALU , Register File, Program Memory and Data memory.

2 Program Files

I have tested my program files on edaplayground.com using
Compile Options : ”2019 -o”
Tools and Simulators : Aldec Rivera Pro 2020.04
There are:

- 4 files containing design code for ALU, Register File, Program Memory and Data Memory.
- 4 files containing testbench code for each design module.
- 1 common run.do file for synthesis.

3 ALU

This module does 16 arithmetic and logiacl operations decided by the opcode given as input.

Inputs are:

- cin: 1 bit carry in
- op: 4 bit operation code
- op1: 32 bit operand 1
- op2: 32 bit operand 2

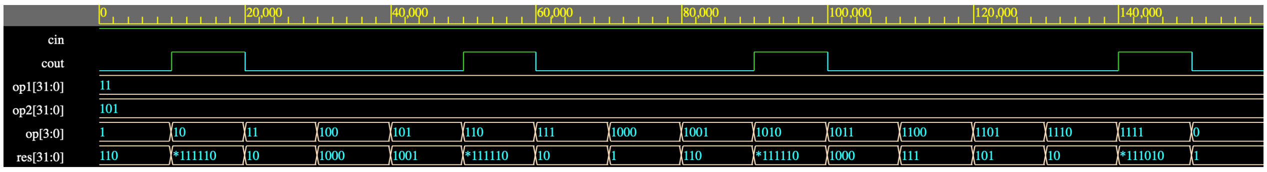
Outputs are:

- cout: 1 bit carry out
- res: 32 bit result of operation on op1 and op2

TestBench

- We take op1 = ”00000000000000000000000000000011”
  op2 = ”00000000000000000000000000000101”
  cin =’1’
  and initialize op = ”0000”.
- Then we run a for loop to increment op by 1 each time. In the for loop, op takes all values from ”0000” to ”1111”.

- Since the process in alu-arch is sensitive to opcode,appropriate operation on operand 1 and operand 2 is performed after every change in opcode.



The logs obtained on synthesis of this module are:

```
# Info: ****************************************
# Info: Device Utilization for 7A100TCSG324
# Info: ****************************************
# Info: Resource                          Used    Avail   Utilization
# Info: ----------------------------------------------------------------
# Info: IOs                               102     210      48.57%
# Info: Global Buffers                    0       32        0.00%
# Info: LUTs                              131     63400     0.21%
# Info: CLB Slices                        32      15850     0.20%
# 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: alu_io    View: alu_arch
# Info: **************************************
# Info:  Number of ports :                          102
# Info:  Number of nets :                           367
# Info:  Number of instances :                      298
# Info:  Number of references to this view :          0
# Info: Total accumulated area :
# Info:  Number of LUTs :                           131
# Info:  Number of LUTs with LUTNM/HLUTNM :           2
# Info:  Number of MUX CARRYs :                      32
# Info:  Number of accumulated instances :          298
# Info: **********************
```

4 Register File

This module stores 16 registers in the form of an array of std-logic-vector(s), each of 32 bits.

Inputs are:

- radd1: address of first register to be read
- radd2: address of second register to be read
- wadd: address of register where we wish to write data
- wdata: data to be written in wadd
- w-enable: enable for writing data (wdata) in register
- clock: clock of period 20 ns

Outputs are:

- rdata1: data read from radd1
- rdata2: data read from radd2

- There are two read inputs which contain address(index) of two registers in the register file. The values stored in these registers are given in output.
- There is one write port which allows data in wdata to be written in register present at wadd in register file. Write can be performed only if w-enable = '1' and clock is present in rising edge.

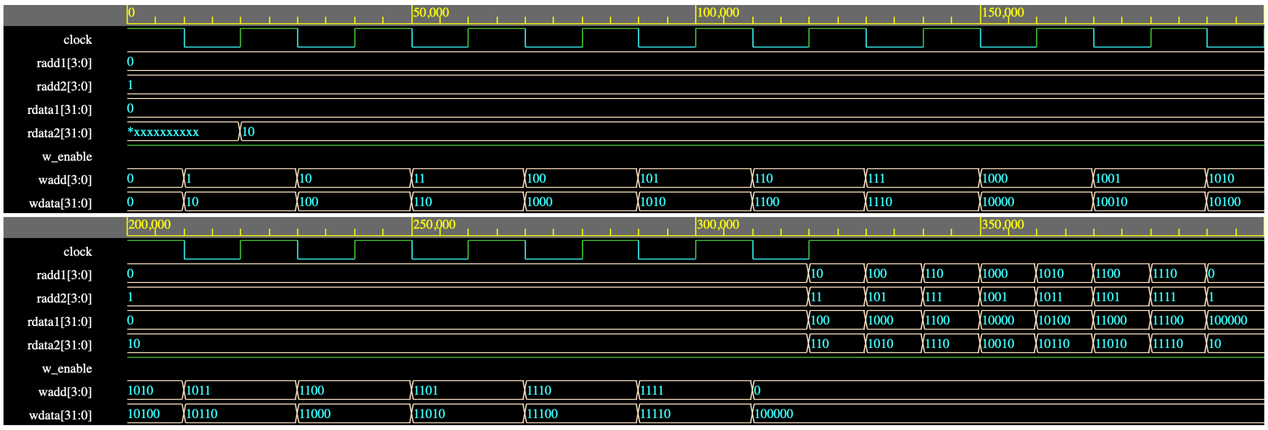
TestBench

- We initialise
  clock ='1'
  w-enable ='1'
  wadd ="0000"
  and run a loop to write data in 15 registers as follows:

```
for I in 0 to 15 loop --loop to write on first 15 registers
    wait for 10 ns; --clock=0
    clock<=not(clock);
    wdata<=wdata+2; --increment value of write data
    wadd<=wadd+1; -- increment write address by 1 to write in next address
    wait for 10 ns;
    clock<=not(clock); --clock =1
end loop;
```

- Clock is also updated in the loop twice so that it completes one cycle in 1 loop.
- Then we run a similar loop to read data written in these 15 registers as follows:

```
for I in 0 to 7 loop --loop to read first 15 registers
    radd1<=radd1+2; --covers all even addresses
    radd2<=radd2+2; --covers all odd addresses
    wait for 10 ns;
end loop;
```



The logs obtained on synthesis are:

```
# Info: ****************************************
# Info: Device Utilization for 7A100TCSG324
# Info: ****************************************
# Info: Resource                          Used    Avail   Utilization
# Info: -------------------------------------------------------------
# Info: IOs                               110     210      52.38%
# Info: Global Buffers                    1       32        3.12%
# Info: LUTs                              48      63400     0.08%
# Info: CLB Slices                        12      15850     0.08%
# Info: Dffs or Latches                   0       126800    0.00%
# Info: Block RAMs                          0       135       0.00%
# Info: Distributed RAMs
# Info:    RAM32M                           10
# Info:    RAM64M                           2
# Info: DSP48E1s                            0       240       0.00%
# Info: -------------------------------------------------------------
# Info: **************************************
# Info: Library: work    Cell: rf_io    View: rf_arch
# Info: **************************************
# Info:  Number of ports :                            110
# Info:  Number of nets :                             220
# Info:  Number of instances :                        111
# Info:  Number of references to this view :            0
# Info: Total accumulated area :
# Info:  Number of LUTs :                              48
# Info:  Number of accumulated instances :            123
# Info: **********************
```

5 Program Memory

This module stores all instructions written in a program in the form of an array of size 64 and each element is a 32 bit std-logic-vector. This receives address of next instruction to be executed and returns the instruction as output.

Input:

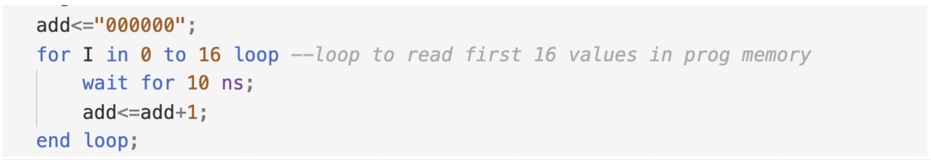
- pm-add : address from where instruction is to be read

Output:

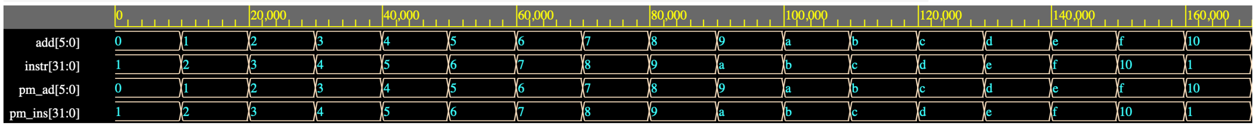
- pm-ins : Instruction present at pm-add in Program memory

TestBench

- We intialise the reading address to ”000000” and run a loop on the address. Since the process is sensitive to read address, it is executed each time the read address is changed and values are read from the program memory.



```
add<="000000";
for I in 0 to 16 loop --loop to read first 16 values in prog memory
    wait for 10 ns;
    add<=add+1;
end loop;
```



The logs obtained on synthesis of this component are:

```
# Info: ****************************************
# Info: Device Utilization for 7A100TCSG324
# Info: ****************************************
# Info: Resource                            Used    Avail   Utilization
# Info: -------------------------------------------------------------
# Info: IOs                                 38      210      18.10%
# Info: Global Buffers                      0       32        0.00%
# Info: LUTs                              5       63400     0.01%
# 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: pm_io    View: pm_arch
# Info: ***************************************
# Info:  Number of ports :                          38
# Info:  Number of nets :                           46
# Info:  Number of instances :                      42
# Info:  Number of references to this view :         0
# Info: Total accumulated area :
# Info:  Number of LUTs :                            5
# Info:  Number of LUTs with LUTNM/HLUTNM :          4
# Info:  Number of accumulated instances :          42
# Info: **********************
```

6 Data Memory

This module stores all data in the form of an array of 64 std-logic-vectors, each of size 32bits.
This memory allows reading data present at input address and writing data given as input to input address. Reading is unclocked while writing in memory is clocked.

- Write can be of three types:
  Writing 1 byte : w-enable ="0001" or "0010" or "0100" or "1000"

  Bits (7-0) , (15-8) , (23-16) , (31-24) (respectively) of word present at wadd in data memory are replaced by (7-0) bits of wdata.
  Writing half word(2 bytes) : w-enable = "0011" or "1100"

  Bits (15-0) and (31-16) (respectively) of word present at wadd in data memory are replaced by (15-0) bits of wdata.
  Writing full word(4 bytes) : w-enable = "1111"
- All bits of word present at wadd in data memory are replaced by all bits of wdata.
- For all other values of w-enable , word remains unchanged.

Inputs are:

- w-enable : 4 bit enable to choose the bytes where data is to be written
- wadd : 6 bit address where data is to be written in memory
- wdata : 32 bit data to be written in wadd in memory
- clk : clock of period 20 ns

Outputs are:

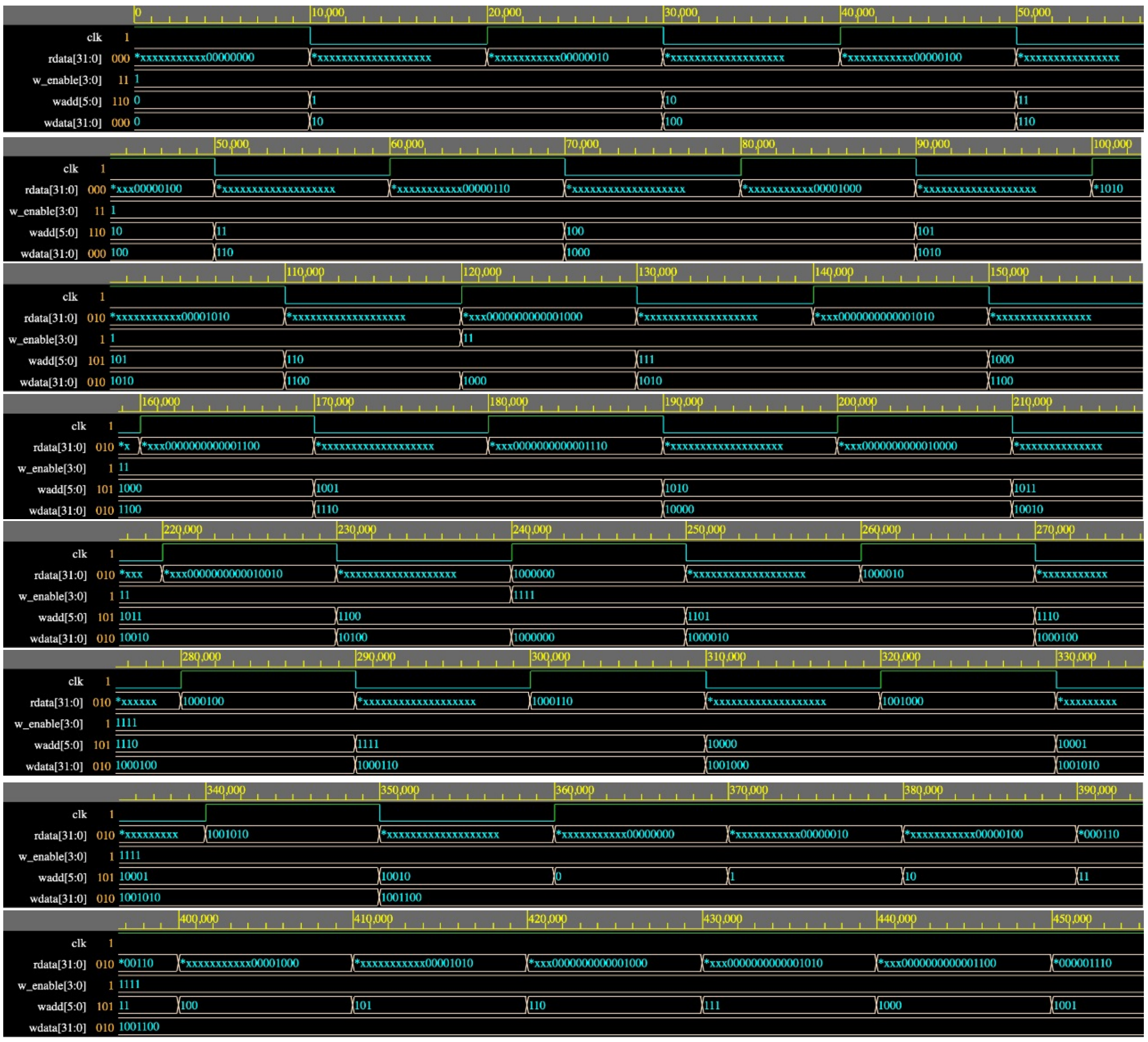
- rdata : 32 bit data present at wadd (read from memory)

TestBench

- Clock, write address and write data are initialised as follows:
  clk = '1'
  wadd = "000000"
  wdata = "00000000000000000000000000000000"
- For sample values of w-enable ie "0001" , "0011" and "1111", a loop is run to write byte/ half-word/ full-word 5 times in each type of write. Write address is incremented by 1 in the loop to write in next consecutive address in next run of the loop. Write data is incremented by 2 so that different values are written in each address.

- Clock is also updated in the loop twice so that it completes one cycle in 1 loop.
- As data gets written in the memory at appropriate address at rising edge of clock, rdata is also updated simultaneously.

```
clk<='1';--clk initialised to 1
wadd<="000000";
wdata<="00000000000000000000000000000000";
w_enable<="0001";
for I in 0 to 5 loop --loop to write in leftmost byte for first 5 words
    wait for 10 ns;
    clk<= not(clk);--clk=0
    wdata<=wdata+2;--increment write data value
    wadd<=wadd+1;--increment write address
    wait for 10 ns;
    clk<=not(clk);--clk=1
end loop;
```



The logs obtained on synthesis of this component are:

```
# Info: ****************************************
# Info: Device Utilization for 7A100TCSG324
# Info: **************************************
# Info: Resource                          Used    Avail   Utilization
# Info: -------------------------------------------------------------
# Info: IOs                               75      210      35.71%
# Info: Global Buffers                    1       32        3.12%
# Info: LUTs                              62      63400     0.10%
# Info: CLB Slices                        13      15850     0.08%
# Info: Dffs or Latches                   0       126800    0.00%
# Info: Block RAMs                        0       135       0.00%
# Info: Distributed RAMs
# Info:    RAM64X1S                       32
# Info: DSP48E1s                          0       240       0.00%
# Info: -------------------------------------------------------------
# Info: **************************************
# Info: Library: work    Cell: dm_io    View: dm_arch
# Info: **************************************
# Info:  Number of ports :                          75
# Info:  Number of nets :                          180
# Info:  Number of instances :                     106
# Info:  Number of references to this view :         0
# Info: Total accumulated area :
# Info:  Number of LUTs :                           62
# Info:  Number of LUTs with LUTNM/HLUTNM :          6
# Info:  Number of accumulated instances :         137
# Info: *********************
```