ENGG*3380: Computer Organization and Design
February 14th, 2024

Lab 3: Implementation of a Register File and Memory on FPGA

Lab Instructor: Haleh Vahedi

Group #37:
Members: Eric Yates, Lukas Krampitz, Alec McBurney

Problem Statement

1. Create and simulate single port memory using write-first, read-first, and no-change implementations using block and distributed RAM for a total of 6 memory simulations. Utilization data was collected for each memory configuration.

2. Create and simulate a three-port register file with two read ports and one write port. The register file used 4 bits of memory addresses each storing 16 bits of data.

3. Implement in hardware a single port memory with 3 bits of addresses containing 4 bits of data. Access the memory using switches for addresses, and display the 4 bits of data in hex on the seven segment display.

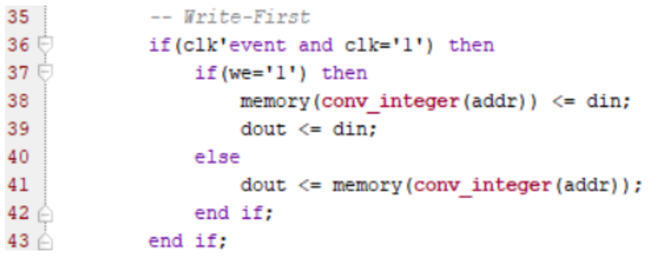
Assumptions and Constraints

Files were written in VHDL 2008 with the ability to access memory with write-first, read-first and no-change memory capabilities while making use of the Block memory onboard the FPGA or using Distributed LUT memory. The lab was to be completed over two weeks; the single port simulations were to be demonstrated in week 1, and the register file and hardware in week 2.

System Overview

The provided code for single port memory, and the schematic of the 3 port register are depicted below:

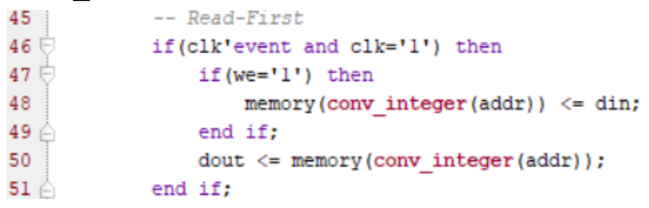
1. **Write_First**



```
-- Write-First
if(clk'event and clk='1') then
    if(we='1') then
        memory(conv_integer(addr)) <= din;
        dout <= din;
    else
        dout <= memory(conv_integer(addr));
    end if;
end if;
```

Write_First segment of the program.

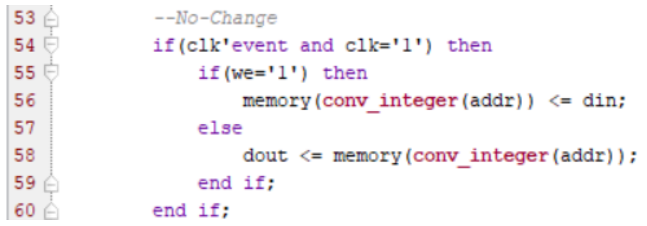
2. **Read_First**



```
        -- Read-First
        if(clk'event and clk='1') then
            if(we='1') then
                memory(conv_integer(addr)) <= din;
            end if;
            dout <= memory(conv_integer(addr));
        end if;
```

Read_First segment of the program.

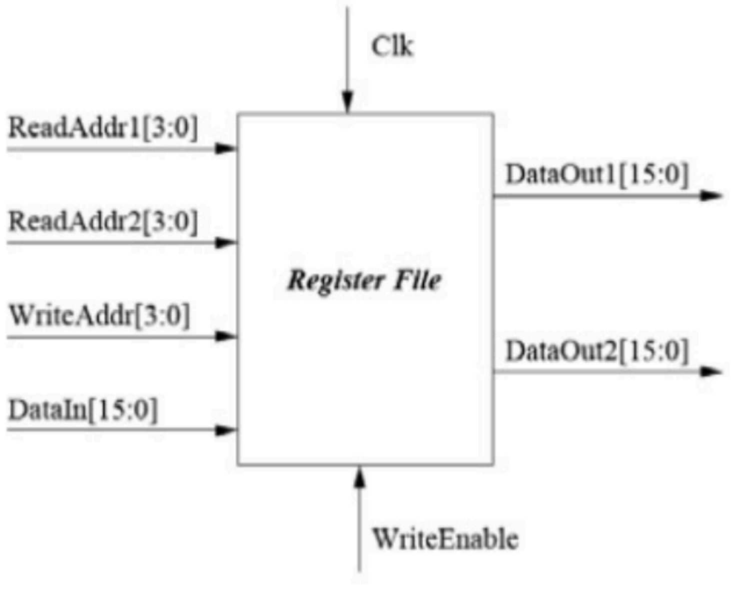
3. **No_Change**



```
        --No-Change
        if(clk'event and clk='1') then
            if(we='1') then
                memory(conv_integer(addr)) <= din;
            else
                dout <= memory(conv_integer(addr));
            end if;
        end if;
```

No_Change segment of the program.

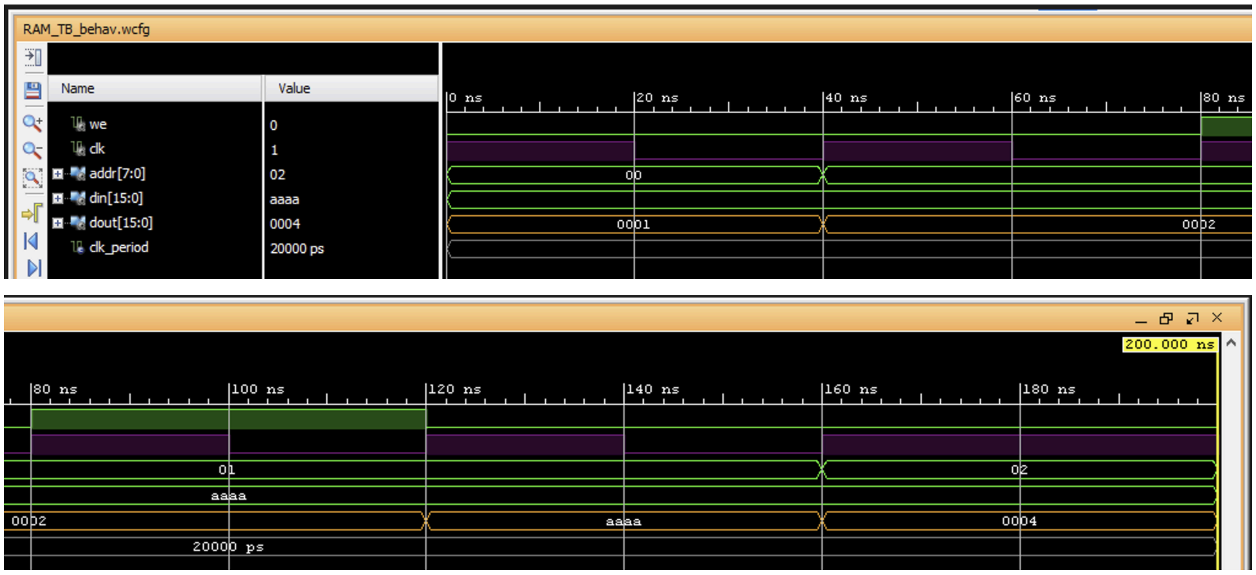
4. **Register file - Single Port Memory Schematic**



Verification

Read First

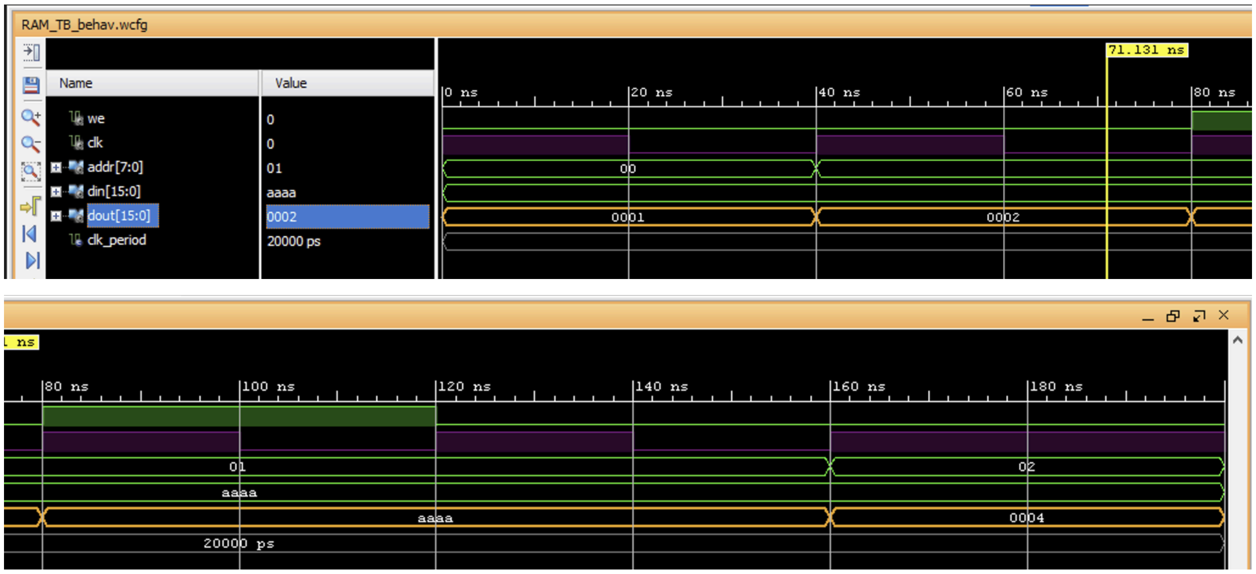
The simulation waveform of the Read-First memory.


RAM_TB_behav.wcfg

Name	Value
we	0
clk	1
addr[7:0]	02
din[15:0]	aaaa
dout[15:0]	0004
clk_period	20000 ps

Write First

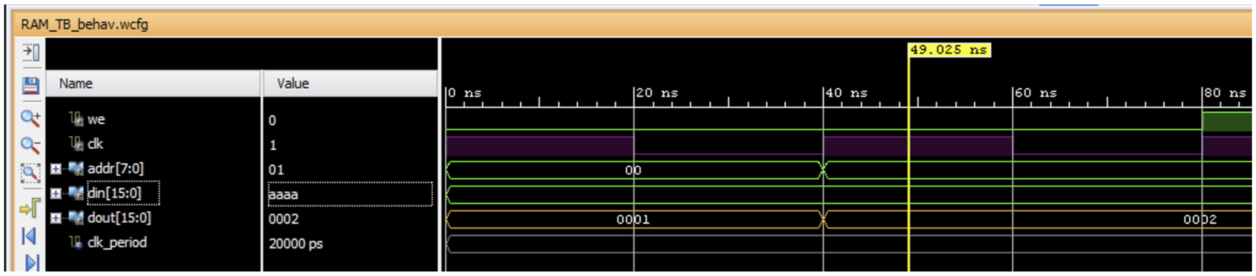
The simulation waveform of the Write-First memory.


RAM_TB_behav.wcfg

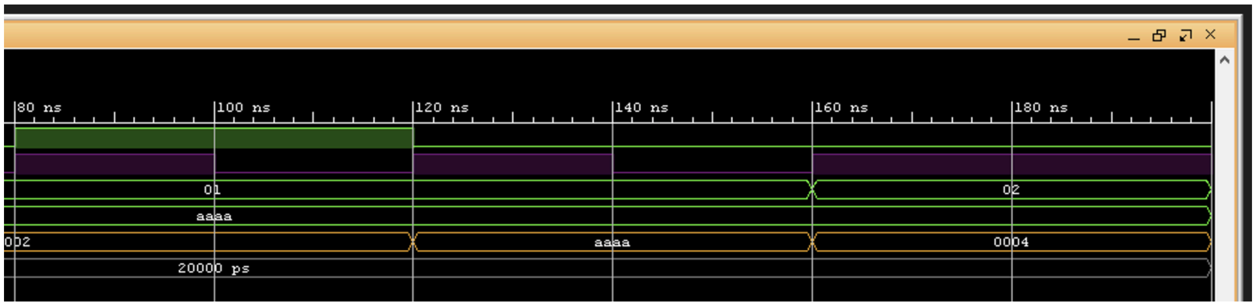
Name	Value
we	0
clk	0
addr[7:0]	01
din[15:0]	aaaa
dout[15:0]	0002
clk_period	20000 ps

No Change

The simulation waveform of the No-Change memory.

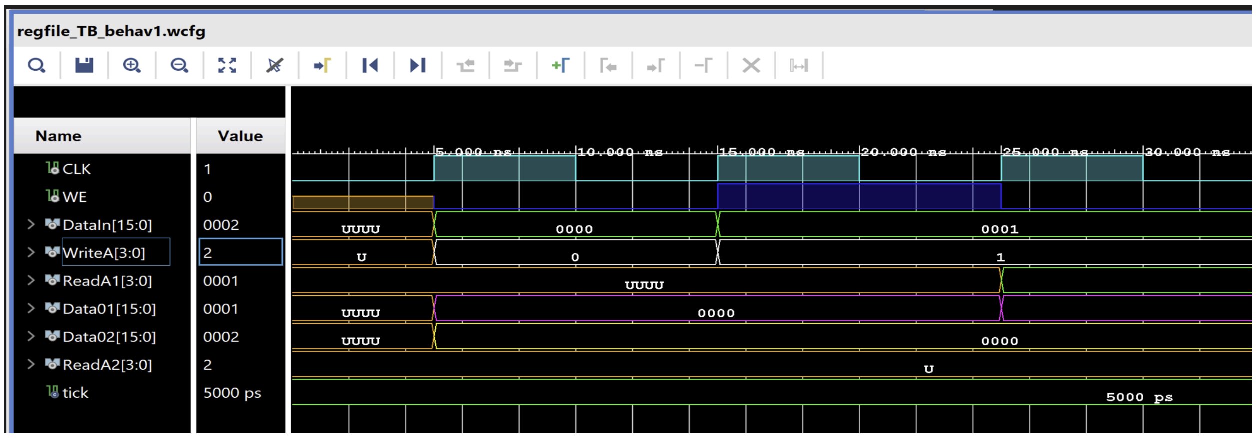

RAM_TB_behav.wcfg

Name	Value
we	0
clk	1
addr[7:0]	01
din[15:0]	aaaa
dout[15:0]	0002
clk_period	20000 ps

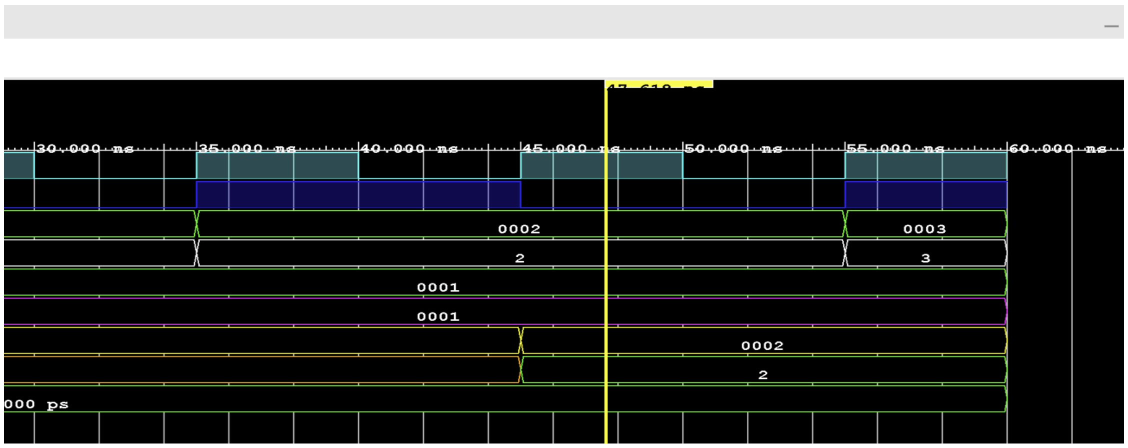


3 Port Register File

The simulation waveform of the 3 port register file.


regfile_TB_behav1.wcfg

Name	Value
CLK	1
WE	0
DataIn[15:0]	0002
WriteA[3:0]	2
ReadA1[3:0]	0001
Data01[15:0]	0001
Data02[15:0]	0002
ReadA2[3:0]	2
tick	5000 ps



Resource Utilization

The charts detailing the resources utilized across all six simulations can be found in **Appendix A** below. There are the tables for No-Change Block, No-Change Distributed, Read-First Block, Read-First Distributed, Write-First Block, Write-First Distributed RAM simulation and implementation. All of the Block memory types have the same resource utilization. The distributed RAM only has slight variation in the slice LUT usage,

Errors and Issues

A) While setting up the 7-segment display for the demonstration of memory on the FPGA there was an error with the code. When the 7-segment decoder received a value of '5' it would instead display what appeared to be an 'H' on the display. This was quickly remedied and the proper output that resembled a '5' was then applied to the 7-segment decoder.

Summary

The results of this lab were a tremendous success and have placed the group in an advantageous position for future labs. This lab made use of the Block memory on the FPGA in preparation for future labs that will be used to design an ISA. This was a crucial step in the process to ensure a thorough understanding of memory hardware for these next steps.

Appendix A - Resources Utilized

Included below are the six tables detailing the resources utilized across the Block and Distributed RAM.

No-Change Block Resources Utilized

```
Memory
---------

+-------------------+------+-------+-----------+-------+
|     Site Type     | Used | Fixed | Available | Util% |
+-------------------+------+-------+-----------+-------+
| Block RAM Tile    |  0.5 |     0 |       135 |  0.37 |
|   RAMB36/FIFO*    |    0 |     0 |       135 |  0.00 |
|   RAMB18          |    1 |     0 |       270 |  0.37 |
|   RAMB18E1 only   |    1 |       |           |       |
+-------------------+------+-------+-----------+-------+
```

No-Change Distributed Resources Utilized

```
Slice Logic
--------------

+----------------------------+------+-------+-----------+-------+
|          Site Type         | Used | Fixed | Available | Util% |
+----------------------------+------+-------+-----------+-------+
| Slice LUTs*                |   65 |     0 |     63400 |  0.10 |
|   LUT as Logic             |    1 |     0 |     63400 | <0.01 |
|   LUT as Memory            |   64 |     0 |     19000 |  0.34 |
|     LUT as Distributed RAM |   64 |     0 |           |       |
|     LUT as Shift Register  |    0 |     0 |           |       |
| Slice Registers            |   16 |     0 |    126800 |  0.01 |
|   Register as Flip Flop    |   16 |     0 |    126800 |  0.01 |
|   Register as Latch        |    0 |     0 |    126800 |  0.00 |
| F7 Muxes                   |   32 |     0 |     31700 |  0.10 |
| F8 Muxes                   |   16 |     0 |     15850 |  0.10 |
+----------------------------+------+-------+-----------+-------+
```

Read-First Block Resources Utilized

```
Memory
---------

+------------------+------+-------+-----------+-------+
|     Site Type    | Used | Fixed | Available | Util% |
+------------------+------+-------+-----------+-------+
| Block RAM Tile   |  0.5  |   0 |   135 |  0.37 |
|   RAMB36/FIFO*   |     0 |   0 |   135 |  0.00 |
|   RAMB18         |     1 |   0 |   270 |  0.37 |
|   RAMB18E1 only |     1 |     |             |       |
+------------------+------+-------+-----------+-------+
```

Read-First Distributed Resources Utilized

```
Slice Logic
--------------

+----------------------------+------+-------+-----------+-------+
|          Site Type         | Used | Fixed | Available | Util% |
+----------------------------+------+-------+-----------+-------+
| Slice LUTs*                |   64 |     0 |   63400    |  0.10 |
|   LUT as Logic             |    0 |     0 |   63400    |  0.00 |
|   LUT as Memory            |   64 |     0 |   19000    |  0.34 |
|     LUT as Distributed RAM |   64 |     0 |            |       |
|     LUT as Shift Register  |    0 |     0 |            |       |
| Slice Registers            |   16 |     0 |   126800   |  0.01 |
|   Register as Flip Flop    |   16 |     0 |   126800   |  0.01 |
|   Register as Latch        |    0 |     0 |   126800   |  0.00 |
| F7 Muxes                   |   32 |     0 |   31700    |  0.10 |
| F8 Muxes                   |   16 |     0 |   15850    |  0.10 |
+----------------------------+------+-------+-----------+-------+
```

Write-First Block Resources Utilized

```
Memory
---------

+------------------+------+-------+-----------+-------+
|     Site Type    | Used | Fixed | Available | Util% |
+------------------+------+-------+-----------+-------+
| Block RAM Tile   |  0.5  |   0 |   135 |  0.37 |
|   RAMB36/FIFO*   |     0 |   0 |   135 |  0.00 |
|   RAMB18         |     1 |   0 |   270 |  0.37 |
|   RAMB18E1 only |     1 |     |             |       |
+------------------+------+-------+-----------+-------+
```

Write-First Distributed Resources Utilized

```
Slice Logic
--------------

+-----------------------------+------+-------+-----------+-------+
|          Site Type          | Used | Fixed | Available | Util% |
+-----------------------------+------+-------+-----------+-------+
| Slice LUTs*                 |   72 |     0 |   63400   |  0.11 |
|   LUT as Logic              |    8 |     0 |   63400   |  0.01 |
|   LUT as Memory             |   64 |     0 |   19000   |  0.34 |
|     LUT as Distributed RAM  |   64 |     0 |           |       |
|     LUT as Shift Register   |    0 |     0 |           |       |
| Slice Registers             |   16 |     0 |   126800  |  0.01 |
|   Register as Flip Flop     |   16 |     0 |   126800  |  0.01 |
|   Register as Latch         |    0 |     0 |   126800  |  0.00 |
| F7 Muxes                    |   32 |     0 |   31700   |  0.10 |
| F8 Muxes                    |   16 |     0 |   15850   |  0.10 |
+-----------------------------+------+-------+-----------+-------+
```

Appendix B - VHDL Test Bench - Block and Distributed RAM

Included below is the VHDL code used in the Test Bench and simulation of the circuit.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY RAM_TB IS
END RAM_TB;

ARCHITECTURE behavior OF RAM_TB IS

      -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT lab2mem is
      generic ( Dwidth : integer := 16; -- Each location is 16 bits
                  Awidth : integer := 8); -- 8 Address lines (i.e., 64
locations)
      port ( we,clk: in std_logic;
            addr: in std_logic_vector(Awidth-1 downto 0);
            din: in std_logic_vector(Dwidth-1 downto 0);
            dout: out std_logic_vector(Dwidth-1 downto 0) );
      end COMPONENT;


   --Inputs
   signal we : std_logic := '0';
   signal clk : std_logic := '0';
   signal addr : std_logic_vector(7 downto 0) := "00000000";
   signal din : std_logic_vector(15 downto 0) := "1010101010101010";

       --Outputs
   signal dout : std_logic_vector(15 downto 0) := "0101010101010101";

   -- Clock period definitions
   constant clk_period : time := 20 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: lab2mem
       generic map(16, 8)
       PORT MAP (
      we => we,
      clk => clk,
      addr => addr,
      din => din,
      dout => dout
      );


   -- Stimulus process
   stim_proc: process
   begin

       -- insert stimulus here
       clk <= '0';
      --wait for clk_period; --don't need this is should be initialized to the
correct values already
       addr <= "00000000";
       clk <= '1';
       wait for clk_period;

       -- Dear Erich: I hope you are doing well and enjoying marking all the
       -- Lab reports <3

       clk <= '0';
       wait for clk_period;
       addr <= "00000001";
       clk <= '1';
       wait for clk_period;

       clk <= '0';
      wait for clk_period;
      addr <= "00000001";
      we <= '1';
      clk <= '1';
      wait for clk_period;

      clk <= '0';
      wait for clk_period;
      addr <= "00000001";
      we <= '0';
      clk <= '1';
      wait for clk_period;

       clk <= '0';
       wait for clk_period;
       addr <= "00000010";
       clk <= '1';
       wait for clk_period;

      wait;
   end process;

END;
```

Appendix C - VHDL Source Code - Block and Distributed RAM

Included below is the VHDL code that has been written. The code has comments and documentation.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity lab2mem is
      generic ( Dwidth : integer := 16; -- Each location is 16 bits
            Awidth : integer := 8); -- 8 Address lines (i.e., 64 locations)
      port ( we,clk: in std_logic;
            addr: in std_logic_vector(Awidth-1 downto 0);
            din: in std_logic_vector(Dwidth-1 downto 0);
            dout: out std_logic_vector(Dwidth-1 downto 0) );
end lab2mem;

architecture Behavioural of lab2mem is
type memType is array(0 to 2**Awidth-1) of std_logic_vector(Dwidth-1 downto
0);

-- The first 8 locations are initialized, the rest set to 0.
signal memory: memType:= ( "0000000000000001",
                        "0000000000000010",
                        "0000000000000100",
                        "0000000000001000",
                        "0000000000010000",
                        "0000000000100000",
                        "0000000001000000",
                        "0000000010000000",
                        others=> "0000000000000000" );

attribute ram_style: string;
attribute ram_style of memory : signal is "block"; --Valid attributes include
block, distributed and auto

begin
      process(clk)
      begin
--     -- Write-First
--    if(clk'event and clk='1') then
--          if(we='1') then
--                memory(conv_integer(addr)) <= din;
--                  dout <= din;
--            else
--                  dout <= memory(conv_integer(addr));
--            end if;
--      end if;

       -- Read-First
--     if(clk'event and clk='1') then
--          if(we='1') then
--                memory(conv_integer(addr)) <= din;
--          end if;
--           dout <= memory(conv_integer(addr));
--    end if;

       --No-Change
       if(clk'event and clk='1') then
            if(we='1') then
            memory(conv_integer(addr)) <= din;
            else
            dout <= memory(conv_integer(addr));
            end if;
      end if;


      end process;
end Behavioural;
```

Appendix D - VHDL Test Bench - Register File

Included below is the VHDL code that has been written. The code has comments and documentation.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity regfile_TB is
end regfile_TB;

architecture behavior of regfile_TB is

component regfile is
      generic (
            Dwidth : integer := 16; -- Each register is 16 bits
            Awidth : integer := 4 -- 16 registers
      );
      port (
            ReadA1, ReadA2, WriteA: in std_logic_vector(Awidth-1 downto 0);
            Data01, Data02 : out std_logic_vector(Dwidth-1 downto 0);
            DataIn: in std_logic_vector(Dwidth-1 downto 0);
            WE, CLK: in std_logic);
end component;

constant tick : time := 5 ns;

--Declare signals

signal ReadA1, ReadA2, WriteA: std_logic_vector(3 downto 0);
signal Data01, Data02, DataIn: std_logic_vector(15 downto 0);
signal WE, CLK: std_logic;

BEGIN

uut:regfile
generic map(16,4)
port map(
    ReadA1 => ReadA1,
    ReadA2 => ReadA2,
    WriteA => WriteA,
    Data01 => Data01,
    Data02 => Data02,
    DataIn => DataIn,
    WE => WE,
    CLK => CLK
);


test : process
begin

    CLK <= '0';
    wait for tick;
    WE <= '0';
    DataIn <= "0000000000000000";
    --ReadA1 <= "0000";
    --ReadA2 <= "0000";
    WriteA <= "0000";
    CLK <= '1';
    wait for tick;

    --write 1 in address 1
    CLK <= '0';
    wait for tick;
    WE <= '1';
    DataIn <= "0000000000000001";
    WriteA <= "0001";
    CLK <= '1';
    wait for tick;

    --read from address 1 at Data01 port
    CLK <= '0';
    wait for tick;
    ReadA1 <= "0001";
    WE <= '0';
    CLK <= '1';
    wait for tick;

    --write 2 in address 2
    CLK <= '0';
    wait for tick;
    WE <= '1';
    DataIn <= "0000000000000010";
    WriteA <= "0010";
    CLK <= '1';
    wait for tick;
    --read from address 2 at Data02 port
    CLK <= '0';
    wait for tick;
    ReadA2 <= "0010";
    WE <= '0';
    CLK <= '1';
    wait for tick;

    --write 3 in address 3
    CLK <= '0';
    wait for tick;
    WE <= '1';
    DataIn <= "0000000000000011";
    WriteA <= "0011";
    CLK <= '1';
    wait for tick;

    --read from address3 at Data01 port
    CLK <= '0';
    wait for tick;
    ReadA1 <= "0011";
    WE <= '0';
    CLK <= '1';
    wait for tick;


    end process;

END;
```

Appendix E - VHDL Source Code - Register File

Included below is the VHDL code that has been written. The code has comments and documentation.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity regfile is
	generic (
		Dwidth : integer := 16; -- Each register is 16 bits
		Awidth : integer := 4 -- 16 registers
	);
	port (
		ReadA1, ReadA2, WriteA: in std_logic_vector(Awidth-1 downto 0);
		Data01, Data02 : out std_logic_vector(Dwidth-1 downto 0);
		DataIn: in std_logic_vector(Dwidth-1 downto 0);
		WE, CLK: in std_logic);
end regfile;

architecture Behavioral of regfile is
	type regType is array(0 to 2**Awidth-1) of std_logic_vector(Dwidth-1
downto 0);
	signal registers: regType := (others => (others => '0'));

begin
	process(CLK)
	begin
		if (clk'event and clk='1') then
			if WE = '1' then
				registers(conv_integer(WriteA)) <= DataIn;
			end if;

			Data01 <= registers(conv_integer(ReadA1));
			Data02 <= registers(conv_integer(ReadA2));

		end if;
	end process;
end Behavioral;
```