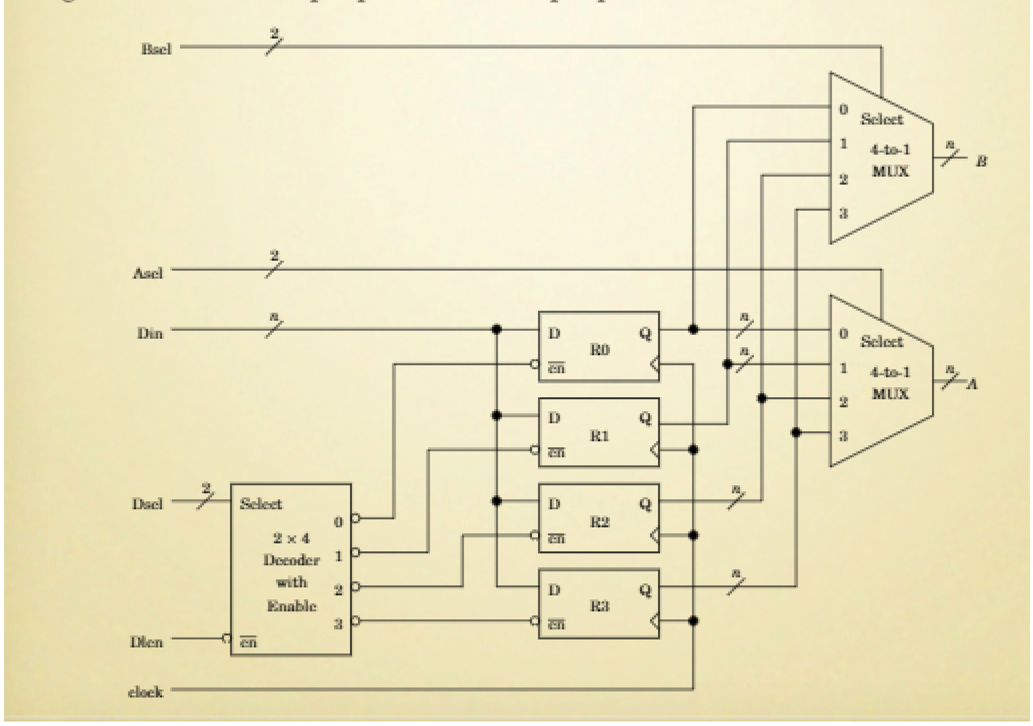
Lab 7 Report

Register File

CENG 342 Digital Systems

A register file with one input port and two output ports.



Samuel Donovan

Samantha Pfeiffer

2021-03-15

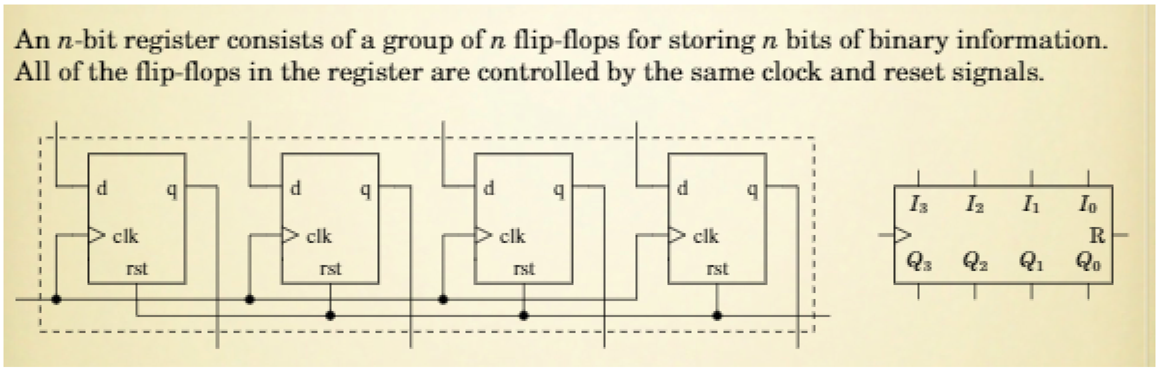
OVERVIEW

The purpose of this lab was to build a register file that will serve as the primary SRAM for the CPU.

--- REGISTER ---

A register is a number of flip flops that are accessed in unison. A register can be written to, or read from. Registers commonly also have reset capability. See figure below.

Figure 1 Register Diagram


An n-bit register consists of a group of n flip-flops for storing n bits of binary information. All of the flip-flops in the register are controlled by the same clock and reset signals.

--- REGISTER FILE ---

The register file consists of a number of registers that can be selectively written to and read from. The file consists of 6 inputs and 2outputs.

Table 1 Register File I/O

Name	In/Out	Size	Description
Reset	In	1	Clears all registers in the file
CLK	In	1	Write is triggered on rising edge
WriteEnable	In	1	Enables writing to the registers
InputSel	In	$\log_2$(# of Reg)	Write register select
Input	In	Register Size	Write input
OutputASel	In	$\log_2$(# of Reg)	Read Output A register select
OutputBSel	In	$\log_2$(# of Reg)	Read Output B register select
OutputA	Out	Register Size	Read Output A
OutputB	Out	Register Size	Read Output B

As can be seen in **Table 1**, two registers can be read at once from the write file.

DESIGN

All designs were made for a Nexys A7-100T (xc7a100tcsg324-1) FPGA board in VHDL using Vivado 2020.2.

--- REGISTER ---

A generic *n*-bit register was designed using a simple `process`, where the state of the output (`q`) is conditionally set to the input (`d`) on `clk`'s rising edge, or to 0 when `reset`.

See **Appendix A**.

--- REGISTER FILE ---

For the register file, two generics are taken as input parameters: `REG_SIZE`, and `N_REG`. Save for that, the port scheme is identical to that seen in **Table 1**.

Some math is done to compute $\log_2$(`REG_SIZE`) which is then used as the `Sel` `STD_LOGIC_VECTOR`'s sizes.

During operation, the `Sel` `STD_LOGIC_VECTOR`s are converted into unsigned integers to be used as indexes in the register array, which are selectively tied to the input and outputs.

See **Appendix D**.

--- CONSTRAINTS ---

For synthesizing a 4-bit register, four switches were set to be it's input, and a push button was tied to its reset. As is required by Vivado/VHDL, the internal clock was tied to `CLK`. Four LEDs were tied to `q`. See **Appendix C** for the .xdc file.

For synthesizing a 4-bit register file with 8 registers, four switches were again set to be it's input, and a push button was tied to its reset, and the internal clock was again used as `CLK`. Additionally, for each of the three selects (`InputSel` , `OutputB` , `OutputASel`), three switches were used. Eight LEDs were divided between `OutputA` and `OutputB` . See **Appendix F** for the .xdc file.

SIMULATION RESULTS

Figure 2 Register Sim Results (REG_SIZE = 4) [Appendix B]

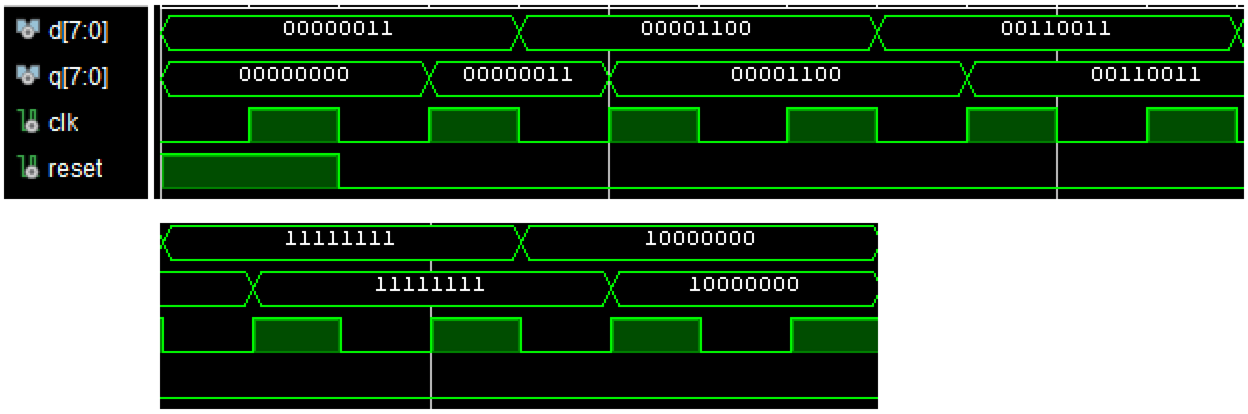
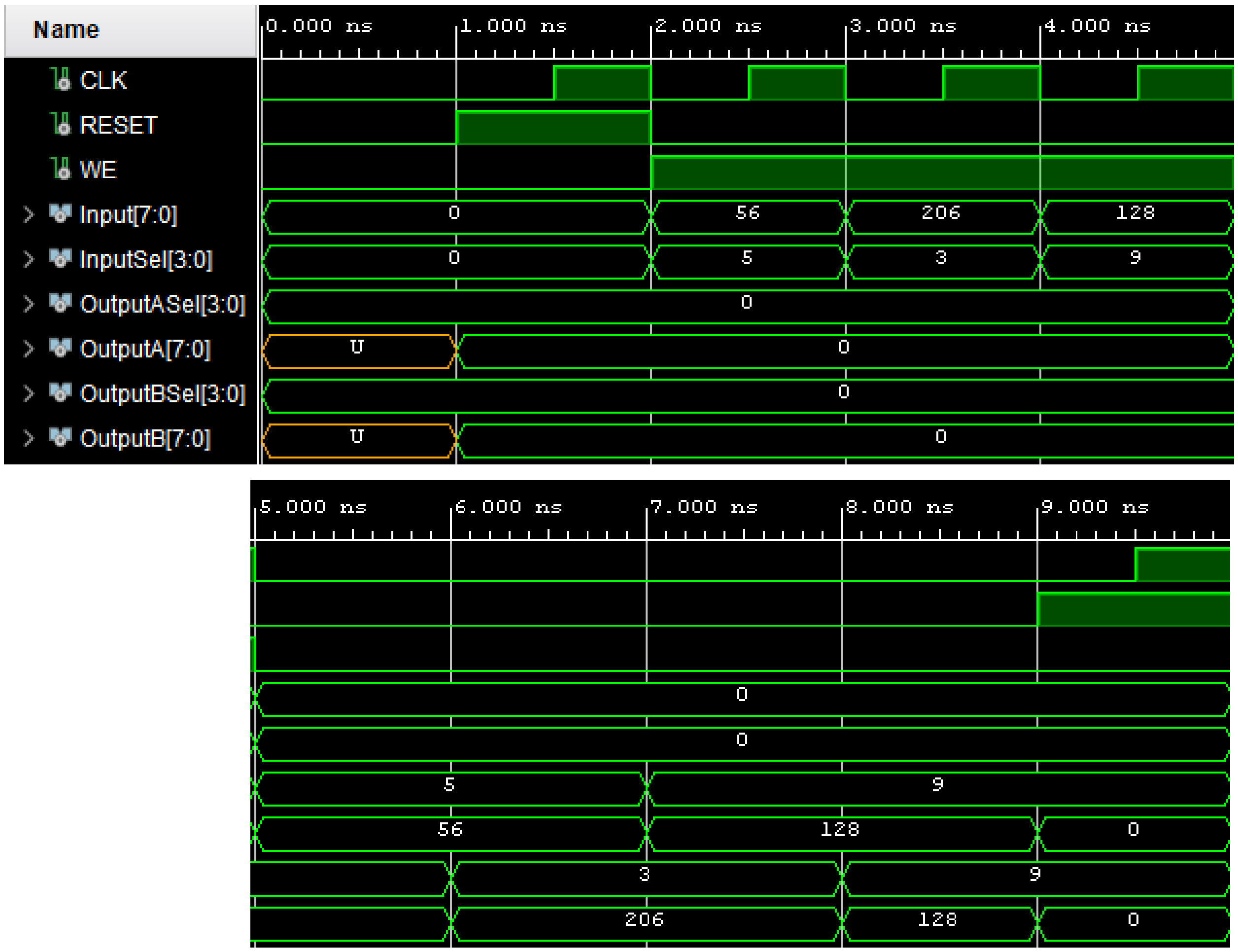


Figure 3 Register File Sim Results (REG_SIZE = 8, N_REGS = 10) [Appendix F]



SYNTHESIS RESULTS

Figure 4 Register Synth - Output Displaying ‘1001'

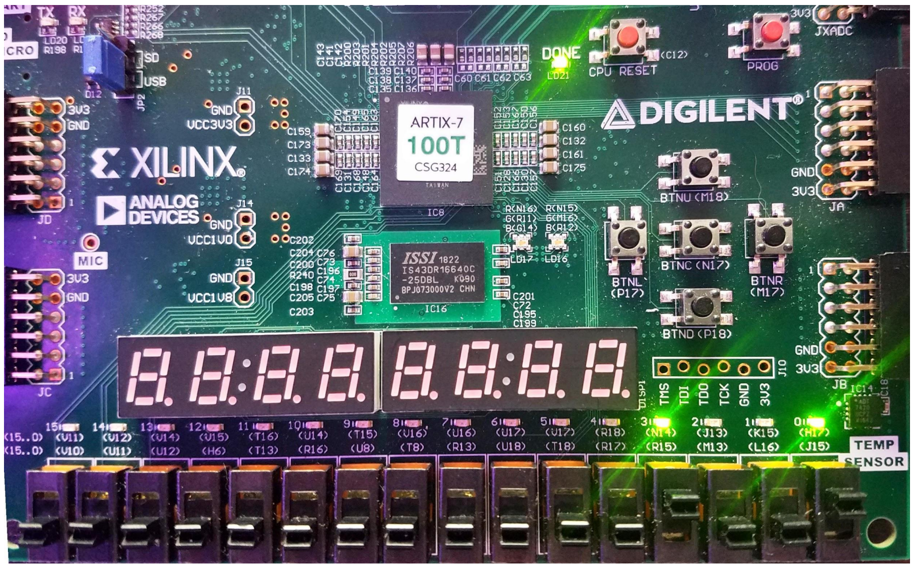


Figure 5 Register Synth - Output Displaying ‘1111’

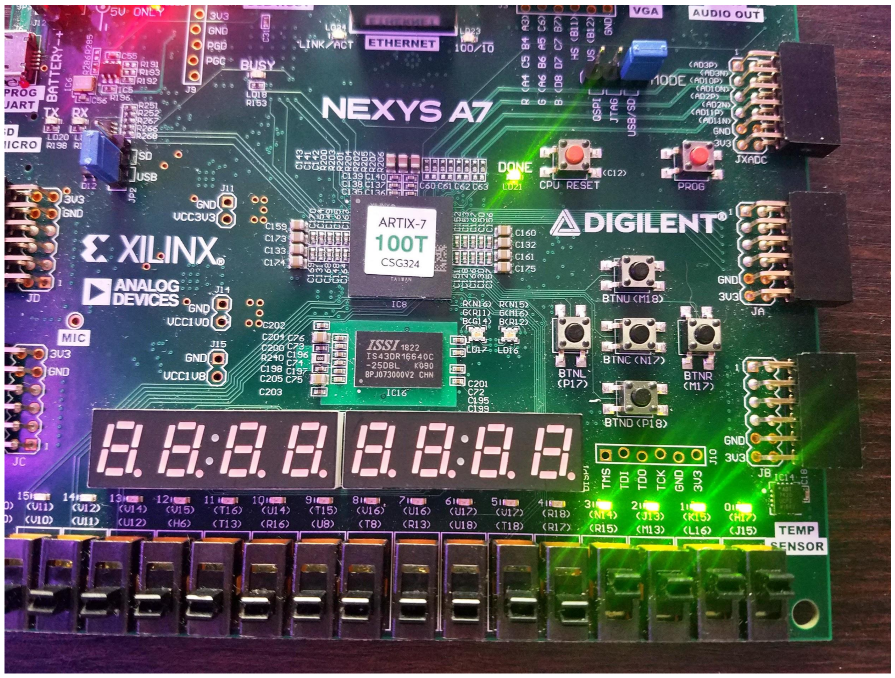


Figure 6 Register File Synth - A and B Displaying Different Registers

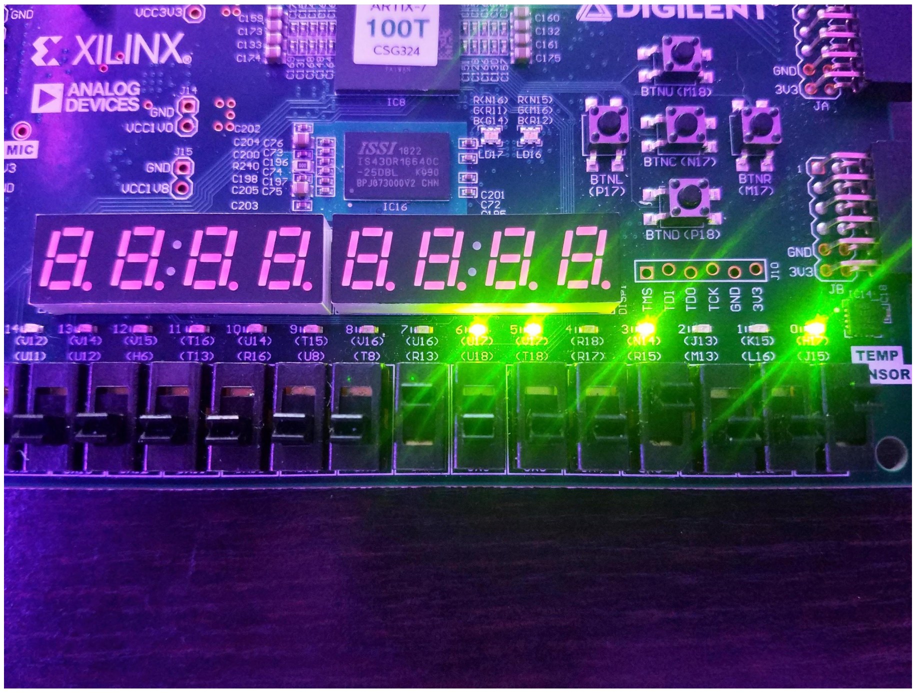


Figure 7 Register File Synth - A and B Displaying the Same Register

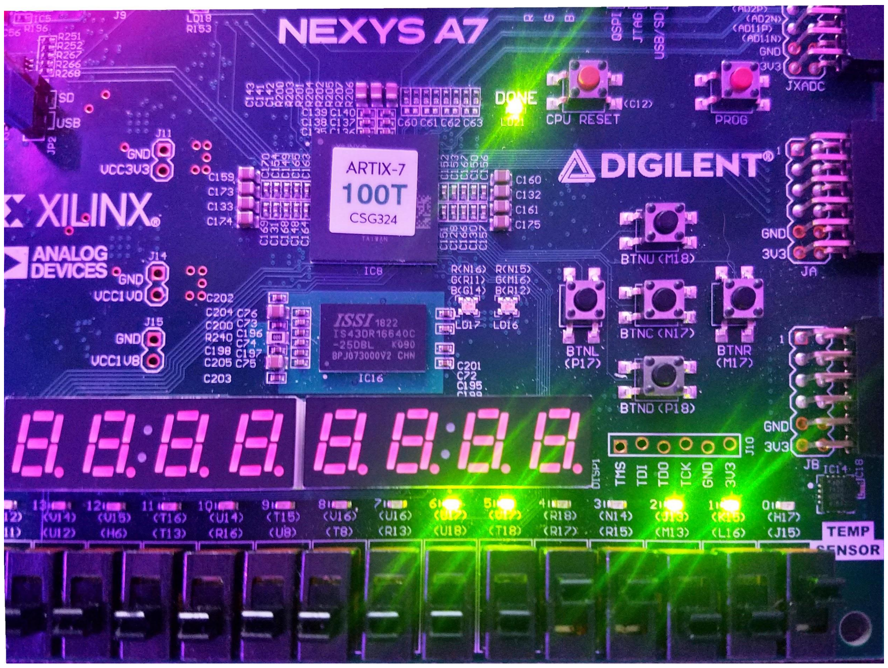
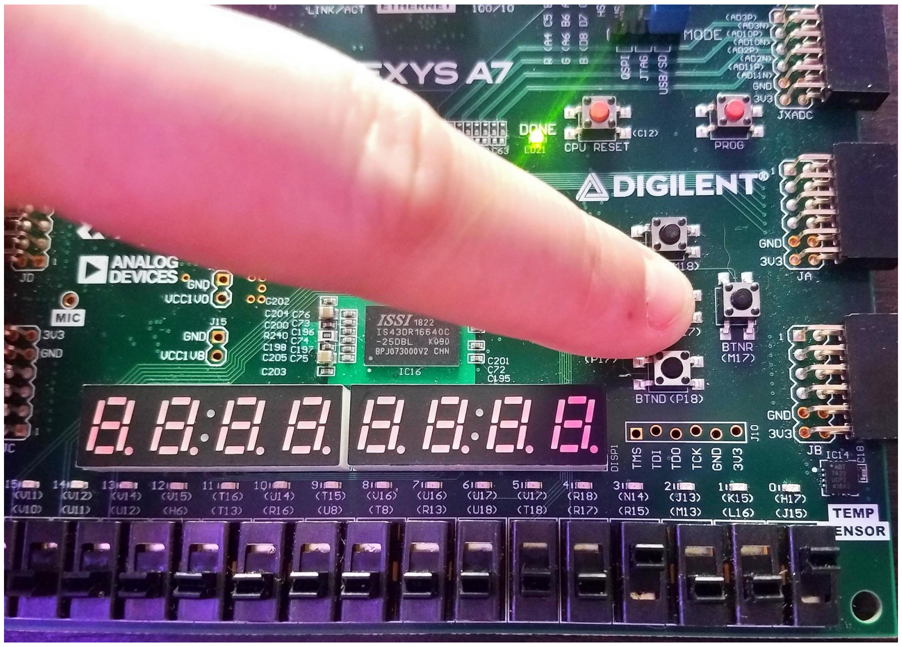


Figure 8 Register File Synth - Reset Button Clears Output



ISSUES ENCOUNTERED

During synthesis, it was discovered that the `CLK` variable could not be tied to switch, and must be tied to the inner clock of the board due to the way that VHDL's `CLK` variable works.

CONCLUSIONS

Given our familiarity with registers and flip-flops , the simplicity of this lab meant writing and simulating our components took very little time.

While the single register was easy to physically test, due to the number bits inputted and outputted from the register file it proved a bit confusing to test physically. However, once testing was complete, both components were found to work as expected.

APPENDIX A: Generic Register VHDL Code (gen_register.vhdl)

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

entity gen_register is
  generic(N: integer := 8);
  Port (
    clk, reset: in std_logic;
    d: in std_logic_vector((N-1) downto 0);
    q: out std_logic_vector((N-1) downto 0)
   );
end gen_register;

architecture Behavioral of gen_register is
begin
    process(clk, reset)
    begin
        if (reset='1') then
            q <= (others=>'0');
        elsif(clk'event and clk='1') then
            q <= d;
        end if;
    end process;
end Behavioral;
```

APPENDIX B: Generic Register Testbench VHDL Code (gen_register_testbench.vhdl)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity gen_reg_testbench is
end gen_reg_testbench;

architecture Behavioral of gen_reg_testbench is
    component gen_register
    PORT(
    clk, reset: in std_logic;
    d: in std_logic_vector(7 downto 0);
    q: out std_logic_vector(7 downto 0)
    );
    end component;

signal d,q: std_logic_vector(7 downto 0) := (others => '0');
signal clk: std_logic := '0';
signal reset: std_logic := '1';
constant clk_period : time := 20ns;

begin

EightBitReg: entity work.gen_register (Behavioral)
    generic map(N=>8)
    port map(clk=>clk, d=>d, q=>q, reset=>reset);

clk_process: process
begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
 end process;

 sim_proc: process
 begin
```

```
    reset <= '0' after 20ns;
    d <= "00000011";
    wait until falling_edge(clk);
    wait until falling_edge(clk);
    d <= "00001100";
    wait until falling_edge(clk);
    wait until falling_edge(clk);
    d <= "00110011";
    wait until falling_edge(clk);
    wait until falling_edge(clk);
    d <= "11111111";
    wait until falling_edge(clk);
    wait until falling_edge(clk);
    d <= "10000000";
    wait until falling_edge(clk);
    wait until falling_edge(clk);
    wait;
  end process;

end Behavioral;
```

APPENDIX C: Generic Register Constraints (gen_reg_constraints.xdc)

```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 }
[get_ports { clk }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz

##Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 }
[get_ports { d[0] }];
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 }
[get_ports { d[1] }];
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 }
[get_ports { d[2] }];
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 }
[get_ports { d[3] }];

## LEDs
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 }
[get_ports { q[0] }];
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 }
[get_ports { q[1] }];
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 }
[get_ports { q[2] }];
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 }
[get_ports { q[3] }];

##Buttons
set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 }
[get_ports { reset }];
```

APPENDIX D: Generic Register File VHDL Code (gen_register_file.vhdl)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.MATH_REAL.ALL;
use IEEE.NUMERIC_STD.ALL;

entity gen_reg_file is
    generic(
        REG_SIZE: integer := 8;
        N_REGS: integer := 2
        );
    Port (
        clk, reset, writeEnable: in std_logic;
        InputSel: in
std_logic_vector(natural(ceil(log2(real(N_REGS))))-1 downto 0);
        Input: in std_logic_vector(REG_SIZE-1 downto 0);
        OutputASel, OutputBSel: in
std_logic_vector(natural(ceil(log2(real(N_REGS))))-1 downto 0);
        OutputA, OutputB: out std_logic_vector (REG_SIZE-1
downto 0)
    );
end gen_reg_file;

architecture Behavioral of gen_reg_file is
    type REG_ARR is array(N_REGS-1 downto 0) of
        std_logic_vector(REG_SIZE-1 downto 0);

    signal regs: REG_ARR;

begin
    process(clk, reset)
    begin
        if(reset = '1') then
            regs <= (others=>(others=>'0')); --Clear all
registers
        elsif(clk'event and clk='1') then
            if writeEnable='1' then
                regs(to_integer(unsigned(InputSel)))<= Input;
            end if;
        end if;
```

```
     end process;
  OutputA <= regs(to_integer(unsigned(OutputASel)));
  OutputB <= regs(to_integer(unsigned(OutputBSel)));

end Behavioral;
```

APPENDIX E: Generic Register File Testbench VHDL Code (gen_register_file_testbench.vhdl)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.MATH_REAL.ALL;
use IEEE.NUMERIC_STD.ALL;
use STD.ENV.STOP;

entity gen_reg_file_testbench is
end gen_reg_file_testbench;

architecture Behavioral of gen_reg_file_testbench is
    constant REGSIZE: integer := 8;
    constant NREGS:   integer := 16;
    constant SELSIZE: natural :=
natural(ceil(log2(real(NREGS))));
    signal CLK, RESET, WE: STD_LOGIC := '0';
    signal InputSel, OutputASel, OutputBSel:
STD_LOGIC_VECTOR(SELSIZE-1 downto 0);
    signal Input, OutputA, OutputB : STD_LOGIC_VECTOR(REGSIZE-1
downto 0);

begin

reg_file: entity work.gen_reg_file (behavioral)
            generic map(REG_SIZE=>REGSIZE, N_REGS=>NREGS)
            port map(
                clk=>CLK,
                reset=>RESET,
                writeEnable=>WE,
                InputSel=>InputSel,
                Input=>Input,
                OutputASel=>OutputASel,
                OutputBSel=>OutputBSel,
                OutputA=>OutputA,
                OutputB=>OutputB
            );

testProc: process
```

```
begin
    --Set signals 0
    CLK <= '0';
    RESET <= '0';
    WE <= '0';
    InputSel <= (others => '0');
    Input <= (others => '0');
    OutputASel <= (others => '0');
    OutputBSel <= (others => '0');
    wait for 1ns;

    --Reset
    RESET <= '1';
    wait for 0.5ns;
    CLK <= '1';
    wait for 0.5ns;
    CLK <= '0';
    RESET <= '0';

    --Write 56 to reg 5
    WE <= '1';
    Input <= std_logic_vector(to_unsigned(56, REGSIZE));
    InputSel <= std_logic_vector(to_unsigned(5, SELSIZE));
    wait for 0.5ns;
    CLK <= '1';
    wait for 0.5ns;
    CLK <= '0';
    Input <= std_logic_vector(to_unsigned(0, REGSIZE));
    InputSel <= std_logic_vector(to_unsigned(0, SELSIZE));

    --Write 206 to reg 3
    WE <= '1';
    Input <= std_logic_vector(to_unsigned(206, REGSIZE));
    InputSel <= std_logic_vector(to_unsigned(3, SELSIZE));
    wait for 0.5ns;
    CLK <= '1';
    wait for 0.5ns;
    CLK <= '0';
    WE <= '0';
    Input <= std_logic_vector(to_unsigned(0, REGSIZE));
    InputSel <= std_logic_vector(to_unsigned(0, SELSIZE));
    --Write 128 to reg 9
    WE <= '1';
    Input <= std_logic_vector(to_unsigned(128, REGSIZE));
    InputSel <= std_logic_vector(to_unsigned(9, SELSIZE));
    wait for 0.5ns;
    CLK <= '1';
    wait for 0.5ns;
    CLK <= '0';
    WE <= '0';
    Input <= std_logic_vector(to_unsigned(0, REGSIZE));
    InputSel <= std_logic_vector(to_unsigned(0, SELSIZE));

    --Read reg 5 to output A
    OutputASel <= std_logic_vector(to_unsigned(5, SELSIZE));
    wait for 1ns;

    --Read reg 3 to output
    OutputBSel <= std_logic_vector(to_unsigned(3, SELSIZE));
    wait for 1ns;

    --Read reg 9 to output A
    OutputASel <= std_logic_vector(to_unsigned(9, SELSIZE));
    wait for 1ns;

    --Read reg 9 to output B
    OutputBSel <= std_logic_vector(to_unsigned(9, SELSIZE));
    wait for 1ns;

    --Reset
    RESET <= '1';
    wait for 0.5ns;
    CLK <= '1';
    wait for 0.5ns;
    CLK <= '0';
    RESET <= '0';

    stop;
end process;
end Behavioral;
```

APPENDIX F: Generic Register File Constraints File (gen_reg_file_constraints.xdc)

```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 }
[get_ports { clk }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz

##Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 }
[get_ports { Input[0] }];
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 }
[get_ports { Input[1] }];
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 }
[get_ports { Input[2] }];
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 }
[get_ports { Input[3] }];
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 }
[get_ports { InputSel[0] }];
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 }
[get_ports { InputSel[1] }];
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 }
[get_ports { InputSel[2] }];
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 }
[get_ports { OutputASel[0] }];
set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 }
[get_ports { OutputASel[1] }];
set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 }
[get_ports { OutputASel[2] }];
set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 }
[get_ports { OutputBSel[0] }];
set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 }
[get_ports { OutputBSel[1] }];
set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 }
[get_ports { OutputBSel[2] }];
#set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 }
[get_ports { SW[13] }]; #IO_L
set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 }
[get_ports { clk }];
set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 }
[get_ports { writeEnable }];
```

```
## LEDs
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 }
[get_ports { OutputA[0] }];
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 }
[get_ports { OutputA[1] }];
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 }
[get_ports { OutputA[2] }];
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 }
[get_ports { OutputA[3] }];
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 }
[get_ports { OutputB[0] }];
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 }
[get_ports { OutputB[1] }];
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 }
[get_ports { OutputB[2] }];
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 }
[get_ports { OutputB[3] }];

##Buttons
set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 }
[get_ports { reset }];
```