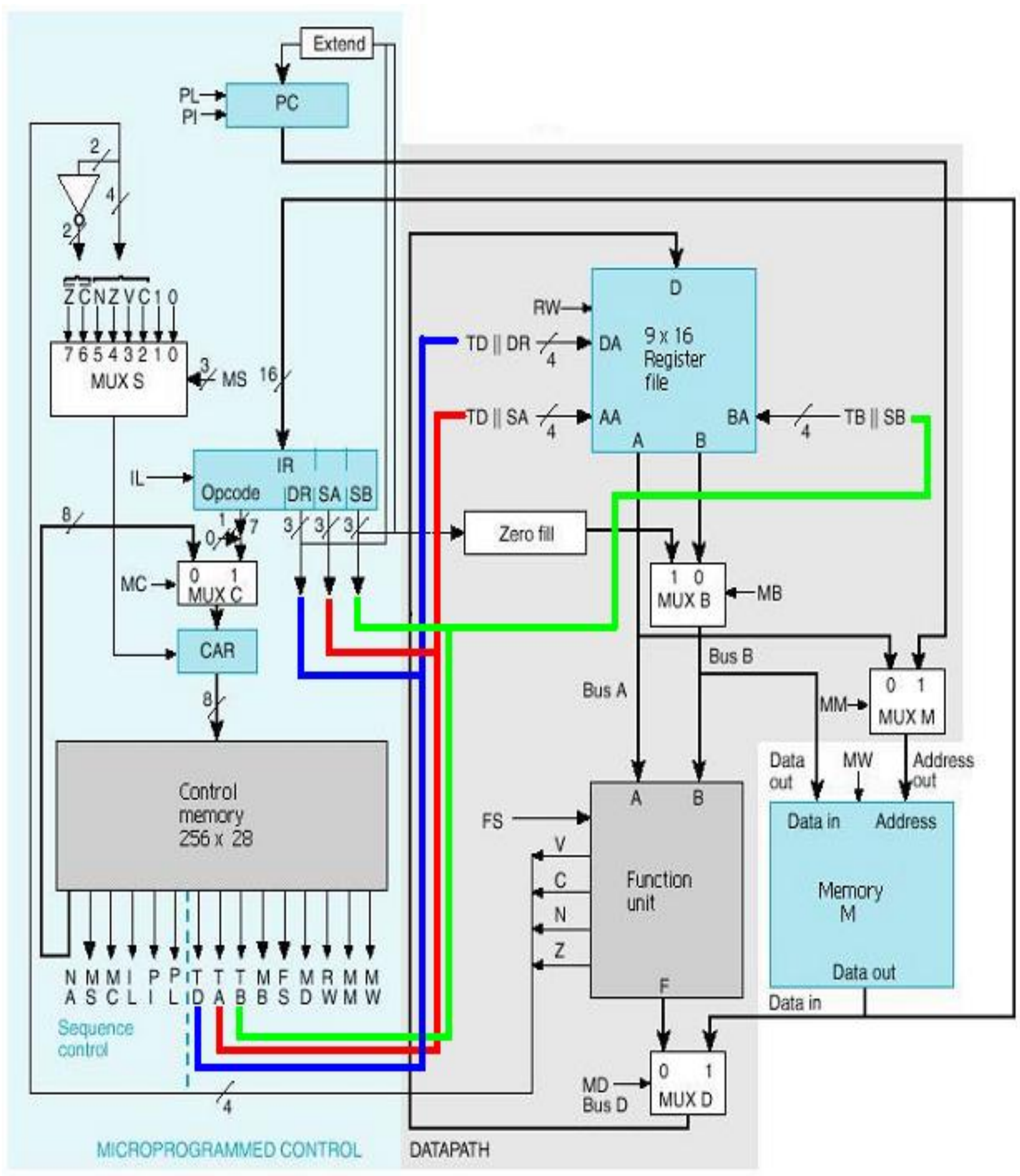
TADHG RIORDAN
12309240

ASSIGNMENT 3: COMPUTER ARCHITECHTURE
MICROCODED INSTRUCTION SET PROCESSOR

I will show the components for the microcoded instruction set processor component by component, along with a testbench for each and a short comment on how it works. While each component individually works as specified and and I have a ‚processor' file which connects them together, I could not in the end get the microcoded instructions work. I believe that I would eventually work this out if I had more time, but underestimated the degree of work in debugging the connected components. The overall Datapath (assignment 2) and register file(assignment 1) work perfectly however and I will specify these also.



PROGRAM COUNTER:

The program counter works by the control of the PI and PL signals. PI will increment it and PL will add it to the enable output and load it.

CODE:

```
entity Program_counter is
    Port ( in0 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
                    reset : IN std_logic;
           PI : in  STD_LOGIC;
           PL : in  STD_LOGIC;
           Clk : in  STD_LOGIC;
           out0 : inout  STD_LOGIC_VECTOR(15 DOWNTO 0)
                    );
end Program_counter;

architecture Behavioral of Program_counter is

component Arith_fullAdder
Port (
        X : in  STD_LOGIC;
        Y : in  STD_LOGIC;
        Cin : in  STD_LOGIC;
              Cout : out STD_LOGIC;
        Gout : out  STD_LOGIC);
end component;

signal p, PICout_sig, PIGout_sig, PLCout_sig, PLGout_sig
:std_logic_vector(15 downto 0);
signal temp : std_logic_vector(1 downto 0);
begin
    --PI PORT MAPS
    PI00:Arith_fullAdder PORT MAP (
       X => out0(0),
       Y => '1',
    Cin => '0',
    Cout => PICout_sig(0),
    Gout => PIGout_sig(0)
    );

       PI01:Arith_fullAdder PORT MAP(
    X => out0(1),
    Y => '0',
    Cin =>  PICout_sig(0),
    Cout => PICout_sig(1),
    Gout => PIGout_sig(1)
    );

       PI02:Arith_fullAdder PORT MAP(
    X => out0(2),
    Y => '0',
    Cin =>  PICout_sig(1),
    Cout => PICout_sig(2),
    Gout => PIGout_sig(2)
    );

       PI03:Arith_fullAdder PORT MAP(
    X => out0(3),
    Y => '0',
    Cin =>  PICout_sig(2),
    Cout => PICout_sig(3),
    Gout => PIGout_sig(3)
    );
```

```
   PI04:Arith_fullAdder PORT MAP(
X => out0(4),
Y => '0',
Cin =>  PICout_sig(3),
Cout => PICout_sig(4),
Gout => PIGout_sig(4)
);

   PI05:Arith_fullAdder PORT MAP(
X => out0(5),
Y => '0',
Cin =>  PICout_sig(4),
Cout => PICout_sig(5),
Gout => PIGout_sig(5)
);

   PI06:Arith_fullAdder PORT MAP(
X => out0(6),
Y => '0',
Cin =>  PICout_sig(5),
Cout => PICout_sig(6),
Gout => PIGout_sig(6)
);

   PI07:Arith_fullAdder PORT MAP(
X => out0(7),
Y => '0',
Cin =>  PICout_sig(6),
Cout => PICout_sig(7),
Gout => PIGout_sig(7)
);

   PI08:Arith_fullAdder PORT MAP(
X => out0(8),
Y => '0',
Cin =>  PICout_sig(7),
Cout => PICout_sig(8),
Gout => PIGout_sig(8)
);

   PI09:Arith_fullAdder PORT MAP(
X => out0(9),
Y => '0',
Cin =>  PICout_sig(8),
Cout => PICout_sig(9),
Gout => PIGout_sig(9)
);

   PI10:Arith_fullAdder PORT MAP(
X => out0(10),
Y => '0',
Cin =>  PICout_sig(9),
Cout => PICout_sig(10),
Gout => PIGout_sig(10)
);

   PI11:Arith_fullAdder PORT MAP(
X => out0(11),
Y => '0',
Cin =>  PICout_sig(10),
Cout => PICout_sig(11),
Gout => PIGout_sig(11)
```

```
);

   PI12:Arith_fullAdder PORT MAP(
X => out0(12),
Y => '0',
Cin =>  PICout_sig(11),
Cout => PICout_sig(12),
Gout => PIGout_sig(12)
);

   PI13:Arith_fullAdder PORT MAP(
X => out0(13),
Y => '0',
Cin =>  PICout_sig(12),
Cout => PICout_sig(13),
Gout => PIGout_sig(13)
);

   PI14:Arith_fullAdder PORT MAP(
X => out0(14),
Y => '0',
Cin =>  PICout_sig(13),
Cout => PICout_sig(14),
Gout => PIGout_sig(14)
);

   PI15:Arith_fullAdder PORT MAP(
X => out0(15),
Y => '0',
Cin =>  PICout_sig(14),
Cout => PICout_sig(15),
Gout => PIGout_sig(15)
);

   --PL port maps
   PL00:Arith_fullAdder PORT MAP (
   X => out0(0),
   Y => in0(0),
Cin => '0',
Cout => PLCout_sig(0),
Gout => PLGout_sig(0)
);

   PL01:Arith_fullAdder PORT MAP(
X => out0(1),
Y => in0(1),
Cin => PLCout_sig(0),
Cout => PLCout_sig(1),
Gout => PLGout_sig(1)
);

   PL02:Arith_fullAdder PORT MAP(
X => out0(2),
Y => in0(2),
Cin => PLCout_sig(1),
Cout => PLCout_sig(2),
Gout => PLGout_sig(2)
);

   PL03:Arith_fullAdder PORT MAP(
X => out0(3),
Y => in0(3),
Cin => PLCout_sig(2),
Cout => PLCout_sig(3),
Gout => PLGout_sig(3)
);

   PL04:Arith_fullAdder PORT MAP(
X => out0(4),
Y => in0(4),
Cin => PLCout_sig(3),
Cout => PLCout_sig(4),
Gout => PLGout_sig(4)
);

   PL05:Arith_fullAdder PORT MAP(
X => out0(5),
Y => in0(5),
Cin => PLCout_sig(4),
Cout => PLCout_sig(5),
Gout => PLGout_sig(5)
);

   PL06:Arith_fullAdder PORT MAP(
X => out0(6),
Y => in0(6),
Cin => PLCout_sig(5),
Cout => PLCout_sig(6),
Gout => PLGout_sig(6)
);

   PL07:Arith_fullAdder PORT MAP(
X => out0(7),
Y => in0(7),
Cin => PLCout_sig(6),
Cout => PLCout_sig(7),
Gout => PLGout_sig(7)
);

   PL08:Arith_fullAdder PORT MAP(
X => out0(8),
Y => in0(8),
Cin => PLCout_sig(7),
Cout => PLCout_sig(8),
Gout => PLGout_sig(8)
);

   PL09:Arith_fullAdder PORT MAP(
X => out0(9),
Y => in0(9),
Cin => PLCout_sig(8),
Cout => PLCout_sig(9),
Gout => PLGout_sig(9)
);

   PL10:Arith_fullAdder PORT MAP(
X => out0(10),
Y => in0(10),
Cin => PLCout_sig(9),
Cout => PLCout_sig(10),
Gout => PLGout_sig(10)
);

   PL11:Arith_fullAdder PORT MAP(
X => out0(11),
Y => in0(11),
    Cin => PLCout_sig(10),
    Cout => PLCout_sig(11),
    Gout => PLGout_sig(11)
    );

       PL12:Arith_fullAdder PORT MAP(
    X => out0(12),
    Y => in0(12),
    Cin => PLCout_sig(11),
    Cout => PLCout_sig(12),
    Gout => PLGout_sig(12)
    );

       PL13:Arith_fullAdder PORT MAP(
    X => out0(13),
    Y => in0(13),
    Cin => PLCout_sig(12),
    Cout => PLCout_sig(13),
    Gout => PLGout_sig(13)
    );

       PL14:Arith_fullAdder PORT MAP(
    X => out0(14),
    Y => in0(14),
    Cin => PLCout_sig(13),
    Cout => PLCout_sig(14),
    Gout => PLGout_sig(14)
    );

       PL15:Arith_fullAdder PORT MAP(
    X => out0(15),
    Y => in0(15),
    Cin => PLCout_sig(14),
    Cout => PLCout_sig(15),
    Gout => PLGout_sig(15)
    );
 process(Clk)
 begin

 if(rising_edge(Clk)) then
    if reset = '1' then out0 <= "0000000000000000";
    elsif PI = '1' then out0 <= PIGout_sig;
       elsif PL = '1' then out0 <= PLGout_sig;
       end if;
 end if;
 end process;
end Behavioral;
```

TESTBENCH:

```
ENTITY program_counterTest IS
END program_counterTest;

ARCHITECTURE behavior OF program_counterTest IS

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

```
     COMPONENT Program_counter
     PORT(
          in0 : IN  std_logic_vector(15 downto 0);
               reset : IN std_logic;
          PI : IN  std_logic;
          PL : IN  std_logic;
          Clk : IN  std_logic;
          out0 : inout  std_logic_vector(15 downto 0)
         );
     END COMPONENT;


    --Inputs
    signal in0 : std_logic_vector(15 downto 0) :=
(others => '0');
      signal reset : std_logic :=  '0';
    signal PI : std_logic := '0';
    signal PL : std_logic := '0';
    signal Clk : std_logic := '0';

      --Outputs
    signal out0 : std_logic_vector(15 downto 0);

    -- Clock period definitions
    constant Clk_period : time := 10 ns;

BEGIN

      -- Instantiate the Unit Under Test (UUT)
    uut: Program_counter PORT MAP (
           in0 => in0,
                reset => reset,
           PI => PI,
           PL => PL,
           Clk => Clk,
           out0 => out0
         );

    -- Clock process definitions
    Clk_process :process
    begin
          Clk <= '0';
          wait for Clk_period/2;
          Clk <= '1';
          wait for Clk_period/2;
    end process;
```

```
   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;
      --wait for Clk_period*10;
      reset <= '1';
      wait for 5ns;
         reset <= '0';
         --out0 <= "0000000000000000";
         --in0 <= "0000000000000010";

         --PI <= '1';
         --wait for 10ns;
         --assert out0 = "0000000000000001";
         --PI <= '0';
         --wait for 10ns;

         reset <= '1';
         wait for 10ns;
         reset <= '0';

         --PI <= '0';
         --wait for 10ns;

         --PL <= '1';
         --wait for 10ns;
         --assert out0 = "0000000000000010";

      wait;
   end process;

END;
```

ENTITY

CODE

```
entity extend is
    Port ( in0 : in  STD_LOGIC_VECTOR(5 DOWNTO 0);
           out0 : out  STD_LOGIC_VECTOR(15 DOWNTO 0)
                );
end extend;

architecture Behavioral of extend is

begin
process(in0(5))
```

```
begin
 if in0(5) = '1' then
  out0(15 downto 6) <= "1111111111";
  out0(5 downto 0) <= in0;
 elsif in0(5) = '0' then
  out0(15 downto 6) <= "0000000000";
  out0(5 downto 0) <= in0;
 end if;
end process;
end Behavioral;
```

TESTBENCH

```
ENTITY extendTest IS
END extendTest;

ARCHITECTURE behavior OF extendTest IS

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

    COMPONENT extend
    PORT(
         in0 : IN  std_logic_vector(5 downto 0);
         out0 : OUT  std_logic_vector(15 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal in0 : std_logic_vector(5 downto 0) :=
(others => '0');

     --Outputs
   signal out0 : std_logic_vector(15 downto 0);
   -- No clocks detected in port list. Replace
<clock> below with
   -- appropriate port name

   --constant <clock>_period : time := 10 ns;

BEGIN

     -- Instantiate the Unit Under Test (UUT)
   uut: extend PORT MAP (
          in0 => in0,
          out0 => out0
```

```vhdl
        );

    -- Clock process definitions
    --<clock>_process :process
    --begin
          --<clock> <= '0';
          --wait for <clock>_period/2;
          --<clock> <= '1';
          --wait for <clock>_period/2;
    --end process;


    -- Stimulus process
    stim_proc: process
    begin
       -- hold reset state for 100 ns.
       wait for 100 ns;

          in0 <= "101010";
          wait for 10ns;
          assert out0 = "1111111111101010";

          in0 <= "001010";
          wait for 10ns;
          assert out0 = "0000000000001010";

       --wait for <clock>_period*10;
       -- insert stimulus here

       wait;
    end process;

END;
```

MEMORY

CODE

```vhdl
entity Memory is -- use unsigned for memory address
Port ( address : in unsigned(15 downto 0);
          write_data : in std_logic_vector(15 downto
0);
          MW : in std_logic;
        read_data : out std_logic_vector(15 downto
0));
       end Memory;
```

```
architecture Behavioral of Memory is

type mem_array is array(0 to 511) of
std_logic_vector(15 downto 0); -- define type, for
memory arrays

begin
mem_process: process (address, write_data, MW)
-- initialize data memory, X denotes hexadecimal
number
variable data_mem : mem_array := (
X"0000", X"0000", X"0003",X"0003",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000",
X"0000", X"0000", X"0000",X"0000"
);

variable addr:integer;
begin
addr:= conv_integer(address(8 downto 0));
```

```vhdl
if MW ='1' then
data_mem(addr):= write_data;
elsif MW='0' then
read_data <= data_mem(addr) after 10 ns;
end if;
end process;
end Behavioral;
```

TESTBENCH

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;


-- Uncomment the following library declaration if
using
-- arithmetic functions with Signed or Unsigned
values
--USE ieee.numeric_std.ALL;

ENTITY MemoryTest IS
END MemoryTest;

ARCHITECTURE behavior OF MemoryTest IS

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

    COMPONENT Memory
    PORT(
         address : IN  unsigned(15 downto 0);
         write_data : IN  std_logic_vector(15 downto
0);
         MW : IN  std_logic;
         read_data : OUT  std_logic_vector(15 downto
0)
        );
    END COMPONENT;


   --Inputs
   signal address : unsigned(15 downto 0) := (others
=> '0');
   signal write_data : std_logic_vector(15 downto 0)
:= (others => '0');
   signal MW : std_logic := '0';
```

```
     --Outputs
   signal read_data : std_logic_vector(15 downto 0)
:= (others => '0');
   -- No clocks detected in port list. Replace
<clock> below with
   -- appropriate port name

   --constant <clock>_period : time := 10 ns;

BEGIN

     -- Instantiate the Unit Under Test (UUT)
   uut: Memory PORT MAP (
          address => address,
          write_data => write_data,
          MW => MW,
          read_data => read_data
        );

   -- Clock process definitions
   --<clock>_process :process
   --begin
         --<clock> <= '0';
         --wait for <clock>_period/2;
         --<clock> <= '1';
         --wait for <clock>_period/2;
   --end process;


   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;
         address <= "0000000000000000";
      MW <= '0';
         wait for 100ns;

         write_data <= "0000000000001111";
         MW <= '1';
         wait for 100ns;

         MW <= '0';
         wait for 10ns;
         assert read_data = "0000000000001111";

      --wait for <clock>_period*10;
```

```
        -- insert stimulus here

        wait;
    end process;

END;
```

INVERTER

CODE

```
entity NOT_cond is
    Port ( in0 : in  STD_LOGIC;
           in1 : in  STD_LOGIC;
           out0 : out  STD_LOGIC;
           out1 : out  STD_LOGIC);
end NOT_cond;

architecture Behavioral of NOT_cond is

begin
out0 <= not(in0);
out1 <= not(in1);

end Behavioral;

TESTBENCH

ENTITY not_condTest IS
END not_condTest;

ARCHITECTURE behavior OF not_condTest IS

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

    COMPONENT NOT_cond
    PORT(
         in0 : IN  std_logic;
         in1 : IN  std_logic;
         out0 : OUT  std_logic;
         out1 : OUT  std_logic
        );
```

```
     END COMPONENT;


   --Inputs
   signal in0 : std_logic := '0';
   signal in1 : std_logic := '0';

     --Outputs
   signal out0 : std_logic;
   signal out1 : std_logic;
   -- No clocks detected in port list. Replace
<clock> below with
   -- appropriate port name

   --constant <clock>_period : time := 10 ns;

BEGIN

     -- Instantiate the Unit Under Test (UUT)
   uut: NOT_cond PORT MAP (
          in0 => in0,
          in1 => in1,
          out0 => out0,
          out1 => out1
        );

   -- Clock process definitions
   --<clock>_process :process
   --begin
          --<clock> <= '0';
          --wait for <clock>_period/2;
          --<clock> <= '1';
          --wait for <clock>_period/2;
   --end process;


   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;

          in0 <= '0';
          in1 <= '1';
          wait for 5ns;
          assert out0 <= '1';
          assert out1 <= '0';
      wait;
   end process;

END;
```

MUX S

CODE

```
entity MUX_S is
    Port ( in0 : in  STD_LOGIC;
           in1 : in  STD_LOGIC;
           in2 : in  STD_LOGIC;
           in3 : in  STD_LOGIC;
           in4 : in  STD_LOGIC;
           in5 : in  STD_LOGIC;
           in6 : in  STD_LOGIC;
           in7 : in  STD_LOGIC;
           sel : in  STD_LOGIC_VECTOR(2 DOWNTO 0);
           out0 : out STD_LOGIC
              );
end MUX_S;

architecture Behavioral of MUX_S is

begin
out0 <= in0 when sel = "000" else
        in1 when sel = "001" else
           in2 when sel = "010" else
           in3 when sel = "011" else
           in4 when sel = "100" else
           in5 when sel = "101" else
           in6 when sel = "110" else
           in7 when sel = "111" else
           'U' after 5ns;

end Behavioral;
```

TESTBENCH

```
ENTITY MUX_STest IS
END MUX_STest;

ARCHITECTURE behavior OF MUX_STest IS

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

    COMPONENT MUX_S
    PORT(
         in0 : IN  std_logic;
         in1 : IN  std_logic;
         in2 : IN  std_logic;
         in3 : IN  std_logic;
         in4 : IN  std_logic;
         in5 : IN  std_logic;
         in6 : IN  std_logic;
         in7 : IN  std_logic;
         sel : IN  std_logic_vector(2 downto 0);
         out0 : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
   signal in0 : std_logic := '0';
   signal in1 : std_logic := '0';
   signal in2 : std_logic := '0';
   signal in3 : std_logic := '0';
   signal in4 : std_logic := '0';
   signal in5 : std_logic := '0';
   signal in6 : std_logic := '0';
   signal in7 : std_logic := '0';
   signal sel : std_logic_vector(2 downto 0) :=
(others => '0');

    --Outputs
   signal out0 : std_logic;
   -- No clocks detected in port list. Replace
<clock> below with
   -- appropriate port name

   --constant <clock>_period : time := 10 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: MUX_S PORT MAP (
          in0 => in0,
          in1 => in1,
          in2 => in2,
          in3 => in3,
          in4 => in4,
          in5 => in5,
          in6 => in6,
          in7 => in7,
          sel => sel,
          out0 => out0
```

```
      );

-- Clock process definitions
--<clock>_process :process
--begin
        --<clock> <= '0';
        --wait for <clock>_period/2;
        --<clock> <= '1';
        --wait for <clock>_period/2;
--end process;


-- Stimulus process
stim_proc: process
begin
   -- hold reset state for 100 ns.
   wait for 100 ns;
       in0 <= '0';
       in1 <= '1';
       in2 <= '0';
       in3 <= '1';
       in4 <= '0';
       in5 <= '1';
       in6 <= '0';
       in7 <= '1';

       sel <= "000";
       wait for 10ns;
       assert out0 = '0';

       sel <= "001";
       wait for 10ns;
       assert out0 = '1';

       sel <= "010";
       wait for 10ns;
       assert out0 = '0';

       sel <= "011";
       wait for 10ns;
       assert out0 = '1';

       sel <= "100";
       wait for 10ns;
       assert out0 = '0';

       sel <= "101";
       wait for 10ns;
```

```
            assert out0 = '1';

            sel <= "110";
            wait for 10ns;
            assert out0 = '0';

            sel <= "111";
            wait for 10ns;
            assert out0 = '1';

        --wait for <clock>_period*10;

        -- insert stimulus here

        wait;
     end process;

END;
```

INSTRUCTION REGISTER

CODE

```
entity Instruction_reg is
    Port ( in0 : in   STD_LOGIC_VECTOR(15 DOWNTO 0);
           sel : in   STD_LOGIC;
           opcode : out   STD_LOGIC_VECTOR(6 DOWNTO
0);
           DR : out   STD_LOGIC_VECTOR(2 DOWNTO 0);
           SA : out   STD_LOGIC_VECTOR(2 DOWNTO 0);
                Clk : in STD_LOGIC;
           SB : out   STD_LOGIC_VECTOR(2 DOWNTO 0)
                );
end Instruction_reg;

architecture Behavioral of Instruction_reg is
begin
process(Clk)
 begin
  if(rising_edge(Clk)) then
   if sel = '1' then
    opcode <= in0(15 downto 9);
    DR <= in0(8 downto 6);
    SA <= in0(5 downto 3);
    SB <= in0(2 downto 0);
     end if;
  end if;
 end process;
end Behavioral;
```

TESTBENCH

```
ENTITY InstructionRegTest IS
END InstructionRegTest;

ARCHITECTURE behavior OF InstructionRegTest IS

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

    COMPONENT Instruction_reg
    PORT(
         in0 : IN  std_logic_vector(15 downto 0);
         sel : IN  std_logic;
         opcode : OUT  std_logic_vector(6 downto 0);
         DR : OUT  std_logic_vector(2 downto 0);
         SA : OUT  std_logic_vector(2 downto 0);
              Clk : IN  std_logic;
         SB : OUT  std_logic_vector(2 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal in0 : std_logic_vector(15 downto 0)  :=
(others => '0');
   signal Clk : std_logic := '0';
   signal sel : std_logic := '0';

    --Outputs
   signal opcode : std_logic_vector(6 downto 0);
   signal DR : std_logic_vector(2 downto 0);
   signal SA : std_logic_vector(2 downto 0);
   signal SB : std_logic_vector(2 downto 0);

   -- Clock period definitions
   constant Clk_period : time := 10 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: Instruction_reg PORT MAP (
          in0 => in0,
          Clk => Clk,
          sel => sel,
          opcode => opcode,
          DR => DR,
          SA => SA,
         SB => SB
       );

   -- Clock process definitions
   Clk_process :process
   begin
        Clk <= '0';
        wait for Clk_period/2;
        Clk <= '1';
        wait for Clk_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;

        in0 <= "0101010101010101";
        sel <= '0';
        wait for 10ns;
        assert opcode = "UUUUUUU";
        assert DR = "UUU";
        assert SA = "UUU";
        assert SB = "UUU";

        sel <= '1';
        wait for 10ns;
        assert opcode = "0101010";
        assert DR = "101";
        assert SA = "010";
        assert SB = "101";

        sel <= '0';
        wait for 10ns;
        assert opcode = "0101010";
        assert DR = "101";
        assert SA = "010";
        assert SB = "101";

        in0 <= "1111111111111111";
        wait for 10ns;
        assert opcode = "0101010";
        assert DR = "101";
        assert SA = "010";
        assert SB = "101";
```

```
            sel <= '1';
            wait for 10ns;
            assert opcode = "1111111";
            assert DR = "111";
            assert SA = "111";
            assert SB = "111";

         wait for Clk_period*10;
         -- insert stimulus here

         wait;
      end process;

END;
```

MUX C

CODE

```
entity MUX_C is
    Port ( in0 : in  STD_LOGIC_VECTOR(7 DOWNTO 0);
           in1 : in  STD_LOGIC_VECTOR(7 DOWNTO 0);
           sel : in  STD_LOGIC;
           out0 : out  STD_LOGIC_VECTOR(7 DOWNTO 0)
              );
end MUX_C;

architecture Behavioral of MUX_C is

begin
out0 <= in0 when sel = '0' else
        in1 when sel = '1' else
           "UUUUUUUU" after 5ns;

end Behavioral;
```

TESTBENCH

```
ENTITY MUX_CTest IS
END MUX_CTest;

ARCHITECTURE behavior OF MUX_CTest IS

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

    COMPONENT MUX_C
    PORT(
         in0 : IN  std_logic_vector(7 downto 0);
         in1 : IN  std_logic_vector(7 downto 0);
         sel : IN  std_logic;
         out0 : OUT  std_logic_vector(7 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal in0 : std_logic_vector(7 downto 0) :=
(others => '0');
   signal in1 : std_logic_vector(7 downto 0) :=
(others => '0');
   signal sel : std_logic := '0';

    --Outputs
   signal out0 : std_logic_vector(7 downto 0);
   -- No clocks detected in port list. Replace
<clock> below with
   -- appropriate port name

   --constant <clock>_period : time := 10 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: MUX_C PORT MAP (
          in0 => in0,
          in1 => in1,
          sel => sel,
          out0 => out0
        );

   -- Clock process definitions
   --<clock>_process :process
   --begin
         --<clock> <= '0';
         --wait for <clock>_period/2;
         --<clock> <= '1';
         --wait for <clock>_period/2;
   --end process;


   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;
         in0 <= "00000000";
```

```
            in1 <= "11111111";

            sel <= '0';
            wait for 10ns;
            assert out0 <= "00000000";

            sel <= '1';
            wait for 10ns;
            assert out0 <= "11111111";

         --wait for <clock>_period*10;
         -- insert stimulus here

         wait;
      end process;

   END;
```

CONTROL ACCESS REGISTER (CAR)

CODE

```
entity control_access_reg is
    Port ( in0 : in  STD_LOGIC_VECTOR(7 DOWNTO 0);
                Clk : in STD_LOGIC;
                reset : in STD_LOGIC;
           sel : in  STD_LOGIC;
           out0 : inout  STD_LOGIC_VECTOR(7 DOWNTO 0)
                );
end control_access_reg;

architecture Behavioral of control_access_reg is

component Arith_fullAdder
Port (
        X : in  STD_LOGIC;
        Y : in  STD_LOGIC;
        Cin : in  STD_LOGIC;
           Cout : out STD_LOGIC;
        Gout : out  STD_LOGIC);
end component;
--signals
signal Cout_sig, Gout_sig : STD_LOGIC_VECTOR(7 DOWNTO
0);

 begin
  CAR00:Arith_fullAdder PORT MAP(
  X => out0(0),
  Y => '1',
  Cin => '0',
  Cout => Cout_sig(0),
  Gout => Gout_sig(0)
  );

  CAR01:Arith_fullAdder PORT MAP(
  X => out0(1),
  Y => '0',
  Cin => Cout_sig(0),
  Cout => Cout_sig(1),
  Gout => Gout_sig(1)
  );

  CAR02:Arith_fullAdder PORT MAP(
  X => out0(2),
  Y => '0',
  Cin => Cout_sig(1),
  Cout => Cout_sig(2),
  Gout => Gout_sig(2)
  );

  CAR03:Arith_fullAdder PORT MAP(
  X => out0(3),
  Y => '0',
  Cin => Cout_sig(2),
  Cout => Cout_sig(3),
  Gout => Gout_sig(3)
  );

  CAR04:Arith_fullAdder PORT MAP(
  X => out0(4),
  Y => '0',
  Cin => Cout_sig(3),
  Cout => Cout_sig(4),
  Gout => Gout_sig(4)
  );

  CAR05:Arith_fullAdder PORT MAP(
  X => out0(5),
  Y => '0',
  Cin => Cout_sig(4),
  Cout => Cout_sig(5),
  Gout => Gout_sig(5)
  );

  CAR06:Arith_fullAdder PORT MAP(
  X => out0(6),
  Y => '0',
  Cin => Cout_sig(5),
  Cout => Cout_sig(6),
  Gout => Gout_sig(6)
  );

  CAR07:Arith_fullAdder PORT MAP(
  X => out0(7),
  Y => '0',
  Cin => Cout_sig(6),
  Cout => Cout_sig(7),
  Gout => Gout_sig(7)
  );


  process(Clk)
  begin
  if reset = '1' then out0 <= "00000000";
  end if;
  if(falling_edge(Clk)) then
    --if sel = 'U' then out0 <= "00000000";
    if sel = '0' then out0 <= Gout_sig;
    elsif sel = '1' then out0 <= in0;
    end if;
  end if;
  end process;
end Behavioral;
```

TESTBENCH

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

-- Uncomment the following library declaration if
using
-- arithmetic functions with Signed or Unsigned
values
--USE ieee.numeric_std.ALL;

ENTITY CARTest IS
END CARTest;

ARCHITECTURE behavior OF CARTest IS

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

    COMPONENT control_access_reg
    PORT(
        in0 : IN  std_logic_vector(7 downto 0);
        sel : IN  std_logic;
             reset : in std_logic;
             Clk : in std_logic;
        out0 : inout  std_logic_vector(7 downto 0)
       );
  END COMPONENT;


   --Inputs
   signal in0 : std_logic_vector(7 downto 0) :=
(others => '0');
   signal sel : std_logic := '0';
   signal Clk : std_logic := '0';
   signal reset : std_logic := '0';

     --Outputs
   signal out0 : std_logic_vector(7 downto 0);
   -- No clocks detected in port list. Replace
<clock> below with
   -- appropriate port name

   constant Clk_period : time := 10 ns;

BEGIN

     -- Instantiate the Unit Under Test (UUT)
   uut: control_access_reg PORT MAP (
          in0 => in0,
          sel => sel,
          out0 => out0,
               reset => reset,
               Clk => Clk
        );

   --Clock process definitions
   Clk_process :process
   begin
         Clk <= '0';
         wait for Clk_period/2;
         Clk <= '1';
         wait for Clk_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
```

```
      -- hold reset state for 100 ns.
      wait for 100 ns;
      sel <= 'U';
         reset <= '1';
         assert out0 <= "00000000";
         wait for 10ns;
         reset <= '0';

         sel <= '1';
         wait for 10ns;
         assert out0 = "00000110";

      wait for Clk_period*10;

      wait;
   end process;

END;
```

CONTROL MEMORY

CODE

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

entity control_memory is
Port ( MW : out std_logic;
          MM : out std_logic;
          RW : out std_logic;
        MD : out std_logic;
          FS : out std_logic_vector(4 downto 0);
        MB : out std_logic;
          TB : out std_logic;
          TA : out std_logic;
          TD : out std_logic;
          PL : out std_logic;
        PI : out std_logic;
        IL : out std_logic;
        MC : out std_logic;
        MS : out std_logic_vector(2 downto 0);
        NA : out std_logic_vector(7 downto 0);
        IN_CAR : in std_logic_vector(7 downto 0));
end control_memory;

architecture Behavioral of control_memory is
type mem_array is array(0 to 255) of
```

```
std_logic_vector(27 downto 0);
begin
memory_m: process(IN_CAR)
variable control_mem : mem_array:=(
--0
X"000001F", -- 0 - transfer 16 0's to R0, a/b data
outputs all 0's
X"000001F", -- 1
X"000001F", -- 2
X"0000000", -- 3
X"BBBBBBB", -- 4
X"0000000", -- 5
X"CCCCCCC", -- 6
X"0000000", -- 7
X"DDDDDDD", -- 8
X"0000000", -- 9
X"1111111", -- A
X"0000000", -- B
X"2222222", -- C
X"0000000", -- D
X"3333333", -- E
X"0000000", -- F

-- 1
X"0000000", -- 0
X"0000000", -- 1
X"0000000", -- 2
X"0000000", -- 3
X"0000000", -- 4
X"0000000", -- 5
X"0000000", -- 6
X"0000000", -- 7
X"0000000", -- 8
X"0000000", -- 9
X"0000000", -- A
X"0000000", -- B
X"0000000", -- C
X"0000000", -- D
X"0000000", -- E
X"0000000", -- F

-- 2
X"0000000", -- 0
X"0000000", -- 1
X"0000000", -- 2
X"0000000", -- 3
X"0000000", -- 4
X"0000000", -- 5
```

```
X"0000000", -- 6
X"0000000", -- 7
X"0000000", -- 8
X"0000000", -- 9
X"0000000", -- A
X"0000000", -- B
X"0000000", -- C
X"0000000", -- D
X"0000000", -- E
X"0000000", -- F

-- 3
X"0000000", -- 0
X"0000000", -- 1
X"0000000", -- 2
X"0000000", -- 3
X"0000000", -- 4
X"0000000", -- 5
X"0000000", -- 6
X"0000000", -- 7
X"0000000", -- 8
X"0000000", -- 9
X"0000000", -- A
X"0000000", -- B
X"0000000", -- C
X"0000000", -- D
X"0000000", -- E
X"0000000", -- F

-- 4
X"0000000", -- 0
X"0000000", -- 1
X"0000000", -- 2
X"0000000", -- 3
X"0000000", -- 4
X"0000000", -- 5
X"0000000", -- 6
X"0000000", -- 7
X"0000000", -- 8
X"0000000", -- 9
X"0000000", -- A
X"0000000", -- B
X"0000000", -- C
X"0000000", -- D
X"0000000", -- E
X"0000000", -- F

-- 5
```

```
X"0000000", -- 0
X"0000000", -- 1
X"0000000", -- 2
X"0000000", -- 3
X"0000000", -- 4
X"0000000", -- 5
X"0000000", -- 6
X"0000000", -- 7
X"0000000", -- 8
X"0000000", -- 9
X"0000000", -- A
X"0000000", -- B
X"0000000", -- C
X"0000000", -- D
X"0000000", -- E
X"0000000", -- F

-- 6
X"0000000", -- 0
X"0000000", -- 1
X"0000000", -- 2
X"0000000", -- 3
X"0000000", -- 4
X"0000000", -- 5
X"0000000", -- 6
X"0000000", -- 7
X"0000000", -- 8
X"0000000", -- 9
X"0000000", -- A
X"0000000", -- B
X"0000000", -- C
X"0000000", -- D
X"0000000", -- E
X"0000000", -- F

-- 7
X"0000000", -- 0
X"0000000", -- 1
X"0000000", -- 2
X"0000000", -- 3
X"0000000", -- 4
X"0000000", -- 5
X"0000000", -- 6
X"0000000", -- 7
X"0000000", -- 8
X"0000000", -- 9
X"0000000", -- A
X"0000000", -- B
```

```
X"0000000", -- C
X"0000000", -- D
X"0000000", -- E
X"0000000", -- F

-- 8
X"0000000", -- 0
X"0000000", -- 1
X"0000000", -- 2
X"0000000", -- 3
X"0000000", -- 4
X"0000000", -- 5
X"0000000", -- 6
X"0000000", -- 7
X"0000000", -- 8
X"0000000", -- 9
X"0000000", -- A
X"0000000", -- B
X"0000000", -- C
X"0000000", -- D
X"0000000", -- E
X"0000000", -- F

-- 9
X"0000000", -- 0
X"0000000", -- 1
X"0000000", -- 2
X"0000000", -- 3
X"0000000", -- 4
X"0000000", -- 5
X"0000000", -- 6
X"0000000", -- 7
X"0000000", -- 8
X"0000000", -- 9
X"0000000", -- A
X"0000000", -- B
X"0000000", -- C
X"0000000", -- D
X"0000000", -- E
X"0000000", -- F

-- A
X"0000000", -- 0
X"0000000", -- 1
X"0000000", -- 2
X"0000000", -- 3
X"0000000", -- 4
X"0000000", -- 5
```

```
X"0000000", -- 6
X"0000000", -- 7
X"0000000", -- 8
X"0000000", -- 9
X"0000000", -- A
X"0000000", -- B
X"0000000", -- C
X"0000000", -- D
X"0000000", -- E
X"0000000", -- F

-- B
X"0000000", -- 0
X"0000000", -- 1
X"0000000", -- 2
X"0000000", -- 3
X"0000000", -- 4
X"0000000", -- 5
X"0000000", -- 6
X"0000000", -- 7
X"0000000", -- 8
X"0000000", -- 9
X"0000000", -- A
X"0000000", -- B
X"0000000", -- C
X"0000000", -- D
X"0000000", -- E
X"0000000", -- F

-- C
X"0000000", -- 0
X"0000000", -- 1
X"0000000", -- 2
X"0000000", -- 3
X"0000000", -- 4
X"0000000", -- 5
X"0000000", -- 6
X"0000000", -- 7
X"0000000", -- 8
X"0000000", -- 9
X"0000000", -- A
X"0000000", -- B
X"0000000", -- C
X"0000000", -- D
X"0000000", -- E
X"0000000", -- F

-- D
```

```
X"0000000", -- 0
X"0000000", -- 1
X"0000000", -- 2
X"0000000", -- 3
X"0000000", -- 4
X"0000000", -- 5
X"0000000", -- 6
X"0000000", -- 7
X"0000000", -- 8
X"0000000", -- 9
X"0000000", -- A
X"0000000", -- B
X"0000000", -- C
X"0000000", -- D
X"0000000", -- E
X"0000000", -- F

-- E
X"0000000", -- 0
X"0000000", -- 1
X"0000000", -- 2
X"0000000", -- 3
X"0000000", -- 4
X"0000000", -- 5
X"0000000", -- 6
X"0000000", -- 7
X"0000000", -- 8
X"0000000", -- 9
X"0000000", -- A
X"0000000", -- B
X"0000000", -- C
X"0000000", -- D
X"0000000", -- E
X"0000000", -- F

-- F
X"0000000", -- 0
X"0000000", -- 1
X"0000000", -- 2
X"0000000", -- 3
X"0000000", -- 4
X"0000000", -- 5
X"0000000", -- 6
X"0000000", -- 7
X"0000000", -- 8
X"0000000", -- 9
X"0000000", -- A
X"0000000", -- B
```

```
X"0000000", -- C
X"0000000", -- D
X"0000000", -- E
X"0000000"  -- F
);
variable addr : integer;
variable control_out : std_logic_vector(27 downto 0);

begin
addr := conv_integer(IN_CAR);
control_out := control_mem(addr);
     MW <= control_out(0);
     MM <= control_out(1);
     RW <= control_out(2);
     MD <= control_out(3);
     FS <= control_out(8 downto 4);
     MB <= control_out(9);
     TB <= control_out(10);
     TA <= control_out(11);
     TD <= control_out(12);
     PL <= control_out(13);
     PI <= control_out(14);
     IL <= control_out(15);
     MC <= control_out(16);
     MS <= control_out(19 downto 17);
     NA <= control_out(27 downto 20);
end process;

end Behavioral;
```

TESTBENCH

```
ENTITY ControlMemoryTest IS
END ControlMemoryTest;

ARCHITECTURE behavior OF ControlMemoryTest IS

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

    COMPONENT control_memory
    PORT(
         MW : OUT  std_logic;
         MM : OUT  std_logic;
         RW : OUT  std_logic;
         MD : OUT  std_logic;
         FS : OUT  std_logic_vector(4 downto 0);
         MB : OUT  std_logic;
```

```vhdl
          TB : OUT  std_logic;
          TA : OUT  std_logic;
          TD : OUT  std_logic;
          PL : OUT  std_logic;
          PI : OUT  std_logic;
          IL : OUT  std_logic;
          MC : OUT  std_logic;
          MS : OUT  std_logic_vector(2 downto 0);
          NA : OUT  std_logic_vector(7 downto 0);
          IN_CAR : IN  std_logic_vector(7 downto 0)
         );
    END COMPONENT;


   --Inputs
   signal IN_CAR : std_logic_vector(7 downto 0) :=
(others => '0');

    --Outputs
   signal MW : std_logic;
   signal MM : std_logic;
   signal RW : std_logic;
   signal MD : std_logic;
   signal FS : std_logic_vector(4 downto 0);
   signal MB : std_logic;
   signal TB : std_logic;
   signal TA : std_logic;
   signal TD : std_logic;
   signal PL : std_logic;
   signal PI : std_logic;
   signal IL : std_logic;
   signal MC : std_logic;
   signal MS : std_logic_vector(2 downto 0);
   signal NA : std_logic_vector(7 downto 0);
   -- No clocks detected in port list. Replace
<clock> below with
   -- appropriate port name

   --constant <clock>_period : time := 10 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: control_memory PORT MAP (
          MW => MW,
          MM => MM,
          RW => RW,
          MD => MD,
          FS => FS,
          MB => MB,
          TB => TB,
          TA => TA,
          TD => TD,
          PL => PL,
          PI => PI,
          IL => IL,
          MC => MC,
          MS => MS,
          NA => NA,
          IN_CAR => IN_CAR
        );

   -- Clock process definitions
   --<clock>_process :process
   --begin
         --<clock> <= '0';
         --wait for <clock>_period/2;
         --<clock> <= '1';
         --wait for <clock>_period/2;
   --end process;


   -- Stimulus process
   stim_proc: process
   begin

      wait for 100 ns;
      --wait for <clock>_period*10;
         IN_CAR <= "00000010";
        wait for 50ns;

      wait;
   end process;

END;
```

REGISTER FILE 9X16

CODE

```
entity Register_file is
    Port ( s0_Amux9to16 : in  STD_LOGIC_VECTOR(2
DOWNTO 0);
                s0_Bmux9to16 : in  STD_LOGIC_VECTOR(2
DOWNTO 0);
           s0_dec3to9 : in  STD_LOGIC_VECTOR(2 DOWNTO
0);
```

```
                TD : in STD_LOGIC;
                TA : in STD_LOGIC;
                TB : in STD_LOGIC;
                loadEnable : in  STD_LOGIC;
           DData      : in  STD_LOGIC_VECTOR(15
DOWNTO 0);
           Clk :  in  STD_LOGIC;
           Adata : out  STD_LOGIC_VECTOR(15 DOWNTO
0);
           Bdata : out  STD_LOGIC_VECTOR(15 DOWNTO 0)
                );
end Register_file;

architecture Behavioral of Register_file is
--components

     --16 bit register for register file
     component reg
    Port ( load : in  STD_LOGIC;
           Clk : in  STD_LOGIC;
           in0 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
           out0 : out  STD_LOGIC_VECTOR(15 DOWNTO 0)
              );
     end component;

     --AND gate to control decoder to register load
bit
     component load_reg
     Port (  in0 : in  STD_LOGIC;
           in1 : in  STD_LOGIC;
           out0 : out  STD_LOGIC
            );
     end component;

     --A Data 16 bit 9 to 1 multiplexor
     component Amux_9to16bit
     Port (
                in0 : in  STD_LOGIC_VECTOR(15 DOWNTO
0);
           in1 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
           in2 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
           in3 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
           in4 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
           in5 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
           in6 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
           in7 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
                in8 : in  STD_LOGIC_VECTOR(15 DOWNTO
0);
```

```
                 TA : in STD_LOGIC;
            s0 :  in  STD_LOGIC_VECTOR(2 DOWNTO 0);
            out0 : out  STD_LOGIC_VECTOR(15 DOWNTO 0)
                 );
     end component;

     --B Data 16 bit 9 to 1 multiplexor
     component Bmux_9to16bit
     Port (
                 in0 : in  STD_LOGIC_VECTOR(15 DOWNTO
0);
            in1 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
            in2 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
            in3 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
            in4 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
            in5 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
            in6 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
            in7 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
                 in8 : in  STD_LOGIC_VECTOR(15 DOWNTO
0);
                 TB : in STD_LOGIC;
            s0 :  in  STD_LOGIC_VECTOR(2 DOWNTO 0);
            out0 : out  STD_LOGIC_VECTOR(15 DOWNTO 0)
                 );
     end component;

     --1 bit 3 to 9 decoder
     component Decoder_3to9
     Port (
                 in0 : in  STD_LOGIC_VECTOR(2 downto
0);
                 TD : in STD_LOGIC;
            out0 : out  STD_LOGIC_VECTOR(8 downto 0)
                 );
   end component;

-- signals. and to reg, dec to and, reg to mux
signal load_reg0, load_reg1, load_reg2, load_reg3,
       load_reg4, load_reg5, load_reg6, load_reg7,
load_reg8 : std_logic;
signal selDec_reg0, selDec_reg1, selDec_reg2,
selDec_reg3,
       selDec_reg4, selDec_reg5, selDec_reg6,
selDec_reg7,selDec_reg8 : std_logic;
signal reg0_sig, reg1_sig, reg2_sig, reg3_sig,
reg4_sig,
       reg5_sig, reg6_sig, reg7_sig, reg8_sig,
mux_2to16bit_sig,
          Amux_9to16bit_sig, Bmux_9to16bit_sig :
std_logic_vector(15 downto 0);

begin
      gate00:load_reg PORT MAP(
         in0 => loadEnable,
         in1 => selDec_reg0,
         out0 => load_reg0
         );

         gate01:load_reg PORT MAP(
         in0 => loadEnable,
         in1 => selDec_reg1,
         out0 => load_reg1
         );

         gate02:load_reg PORT MAP(
         in0 => loadEnable,
         in1 => selDec_reg2,
         out0 => load_reg2
         );

         gate03:load_reg PORT MAP(
         in0 => loadEnable,
         in1 => selDec_reg3,
         out0 => load_reg3
         );

         gate04:load_reg PORT MAP(
         in0 => loadEnable,
         in1 => selDec_reg4,
         out0 => load_reg4
         );

         gate05:load_reg PORT MAP(
         in0 => loadEnable,
         in1 => selDec_reg5,
         out0 => load_reg5
         );

         gate06:load_reg PORT MAP(
         in0 => loadEnable,
         in1 => selDec_reg6,
         out0 => load_reg6
         );

         gate07:load_reg PORT MAP(
         in0 => loadEnable,
         in1 => selDec_reg7,
         out0 => load_reg7
         );

         gate08:load_reg PORT MAP(
         in0 => loadEnable,
         in1 => selDec_reg8,
         out0 => load_reg8
         );

         reg00:reg PORT MAP(
         in0 => DData,
         load => load_reg0,
         Clk => Clk,
         out0 => reg0_sig
         );

         reg01:reg PORT MAP(
       in0 => DData,
         load => load_reg1,
         Clk => Clk,
         out0 => reg1_sig
         );

         reg02:reg PORT MAP(
         in0 => DData,
         load => load_reg2,
         Clk => Clk,
         out0 => reg2_sig
         );

         reg03:reg PORT MAP(
         in0 => DData,
         load => load_reg3,
         Clk => Clk,
         out0 => reg3_sig
         );

         reg04:reg PORT MAP(
         in0 => DData,
         load => load_reg4,
         Clk => Clk,
         out0 => reg4_sig
         );

         reg05:reg PORT MAP(
         in0 => DData,
         load => load_reg5,
```

```vhdl
Clk => Clk,
out0 => reg5_sig
);

reg06:reg PORT MAP(
in0 => DData,
load => load_reg6,
Clk => Clk,
out0 => reg6_sig
);

reg07:reg PORT MAP(
in0 => DData,
load => load_reg7,
Clk => Clk,
out0 => reg7_sig
);

reg08:reg PORT MAP(
in0 => DData,
load => load_reg8,
Clk => Clk,
out0 => reg8_sig
);

--decoder port map
decoder:Decoder_3to9 PORT MAP(
     in0 => s0_dec3to9,
     out0(0) => selDec_reg0,
     out0(1) => selDec_reg1,
     out0(2) => selDec_reg2,
     out0(3) => selDec_reg3,
     out0(4) => selDec_reg4,
     out0(5) => selDec_reg5,
     out0(6) => selDec_reg6,
     out0(7) => selDec_reg7,
     out0(8) => selDec_reg8,
     TD => TD
);

--A data mux 8 to 16 bit port map
Amux9to16:Amux_9to16bit PORT MAP(
   s0  => s0_Amux9to16,
     in0 => reg0_sig,
     in1 => reg1_sig,
     in2 => reg2_sig,
     in3 => reg3_sig,
     in4 => reg4_sig,
              in5 => reg5_sig,
              in6 => reg6_sig,
              in7 => reg7_sig,
              in8 => reg7_sig,
              out0 => Amux_9to16bit_sig,
              TA => TA
         );

         --B data mux 8 to 16 bit port map
         Bmux9to16:Bmux_9to16bit PORT MAP(
             s0  => s0_Bmux9to16,
              in0 => reg0_sig,
              in1 => reg1_sig,
              in2 => reg2_sig,
              in3 => reg3_sig,
              in4 => reg4_sig,
              in5 => reg5_sig,
              in6 => reg6_sig,
              in7 => reg7_sig,
             in8 => reg8_sig,
              out0 => Bmux_9to16bit_sig,
              TB => TB
         );

         Adata <= Amux_9to16bit_sig;
         Bdata <= Bmux_9to16bit_sig;

end Behavioral;

TESTBENCH
ENTITY registerFileTest IS
END registerFileTest;

ARCHITECTURE behavior OF registerFileTest IS

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

    COMPONENT Register_file
    PORT(
         s0_Amux9to16 : IN  std_logic_vector(2 downto
0);
         s0_Bmux9to16 : IN  std_logic_vector(2 downto
0);
         s0_dec3to9 : IN  std_logic_vector(2 downto
0);
         loadEnable : IN  std_logic;
              TD : in STD_LOGIC;
              TA : in STD_LOGIC;
              TB : in STD_LOGIC;
         DData : IN  std_logic_vector(15 downto 0);
         Clk : IN  std_logic;
         Adata : OUT  std_logic_vector(15 downto 0);
         Bdata : OUT  std_logic_vector(15 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal s0_Amux9to16 : std_logic_vector(2 downto 0)
:= (others => '0');
   signal s0_Bmux9to16 : std_logic_vector(2 downto 0)
:= (others => '0');
   signal s0_dec3to9 : std_logic_vector(2 downto 0)
:= (others => '0');
   signal loadEnable : std_logic := '0';
   signal DData : std_logic_vector(15 downto 0) :=
(others => '0');
   signal Clk : std_logic := '0';
     SIGNAL TD : std_logic := '0';
     SIGNAL TA : std_logic := '0';
     SIGNAL TB : std_logic := '0';

     --Outputs
   signal Adata : std_logic_vector(15 downto 0);
   signal Bdata : std_logic_vector(15 downto 0);

   -- Clock period definitions
   constant Clk_period : time := 10 ns;

BEGIN

     -- Instantiate the Unit Under Test (UUT)
   uut: Register_file PORT MAP (
          s0_Amux9to16 => s0_Amux9to16,
          s0_Bmux9to16 => s0_Bmux9to16,
          s0_dec3to9 => s0_dec3to9,
          loadEnable => loadEnable,
          DData => DData,
               TD => TD,
               TA => TA,
               TB => TB,
          Clk => Clk,
          Adata => Adata,
          Bdata => Bdata
        );
   -- Clock process definitions
   Clk_process :process
   begin
         Clk <= '0';
         wait for Clk_period/2;
         Clk <= '1';
         wait for Clk_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;
         DData <= "1111111111111111";
         TD <= '0';
         TA <= '0';
         TB <= '0';
         s0_dec3to9 <= "000";
         loadEnable <= '1';
         s0_Amux9to16 <= "000";
         s0_Bmux9to16 <= "000";
         wait for 20ns;

         assert AData = "1111111111111111";
         assert BData = "1111111111111111";

      wait for Clk_period*10;

      -- insert stimulus here

      wait;
   end process;

END;
```

ZERO FILL

CODE

```
     entity zeroFill is
    Port ( in0 : in  STD_LOGIC_VECTOR(2 DOWNTO 0);
           out0 : out  STD_LOGIC_VECTOR(15 DOWNTO 0)
                );
end zeroFill;

architecture Behavioral of zeroFill is
```

```
begin
 out0(15 downto 3) <= "0000000000000";
 out0(2 downto 0) <= in0;

end Behavioral;

TESTBENCH
ENTITY zeroFillTest IS
END zeroFillTest;

ARCHITECTURE behavior OF zeroFillTest IS

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

    COMPONENT zeroFill
    PORT(
         in0 : IN  std_logic_vector(2 downto 0);
         out0 : OUT  std_logic_vector(15 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal in0 : std_logic_vector(2 downto 0) :=
(others => '0');

     --Outputs
   signal out0 : std_logic_vector(15 downto 0);
   -- No clocks detected in port list. Replace
<clock> below with
   -- appropriate port name

   --constant <clock>_period : time := 10 ns;

BEGIN

     -- Instantiate the Unit Under Test (UUT)
   uut: zeroFill PORT MAP (
          in0 => in0,
          out0 => out0
        );

   -- Clock process definitions
   --<clock>_process :process
   --begin
         --<clock> <= '0';
```

```
            --wait for <clock>_period/2;
            --<clock> <= '1';
            --wait for <clock>_period/2;
      --end process;


      -- Stimulus process
      stim_proc: process
      begin
         -- hold reset state for 100 ns.
         wait for 100 ns;
            in0 <= "101";
            wait for 10ns;
            assert out0 <= "0000000000000101";

         --wait for <clock>_period*10;
         -- insert stimulus here

         wait;
      end process;

   END;
```

B MUX

CODE

```
entity Datapath_Bmux2to1 is
    Port ( in0 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
           in1 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
           S : in  STD_LOGIC;
           out0 : out  STD_LOGIC_VECTOR(15 DOWNTO 0)
                );
end Datapath_Bmux2to1;

architecture Behavioral of Datapath_Bmux2to1 is

begin
out0 <= in0 when S <= '0' else
        in1 when S <= '1';

end Behavioral;
```

TESTBENCH

```
ENTITY Datapath_Bmux2to1Test IS
END Datapath_Bmux2to1Test;
```

```
ARCHITECTURE behavior OF Datapath_Bmux2to1Test IS

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

    COMPONENT Datapath_Bmux2to1
    PORT(
         in0 : IN  std_logic_vector(15 downto 0);
         in1 : IN  std_logic_vector(15 downto 0);
         S : IN  std_logic;
         out0 : OUT  std_logic_vector(15 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal in0 : std_logic_vector(15 downto 0) :=
(others => '0');
   signal in1 : std_logic_vector(15 downto 0) :=
(others => '0');
   signal S : std_logic := '0';

    --Outputs
   signal out0 : std_logic_vector(15 downto 0);
   -- No clocks detected in port list. Replace
<clock> below with
   -- appropriate port name

   --constant <clock>_period : time := 10 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: Datapath_Bmux2to1 PORT MAP (
          in0 => in0,
          in1 => in1,
          S => S,
          out0 => out0
        );

   -- Clock process definitions
   --<clock>_process :process
   --begin
         --<clock> <= '0';
         --wait for <clock>_period/2;
         --<clock> <= '1';
         --wait for <clock>_period/2;
   --end process;
   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;

         in0 <= "0000000000000000";
         in1 <= "1111111111111111";

         S <= '0';
         wait for 10ns;
         assert out0 <= "0000000000000000";

         S <= '1';
         wait for 10ns;
         assert out0 <= "1111111111111111";

      --wait for <clock>_period*10;
      -- insert stimulus here
      wait;
   end process;

END;
```

MUX M

CODE

```
entity MUX_M is
    Port ( in0 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
           in1 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
           sel : in  STD_LOGIC;
           out0 : out STD_LOGIC_VECTOR(15 DOWNTO 0)
                );
    end MUX_M;

architecture Behavioral of MUX_M is

begin
out0 <= in0 when sel = '0' else
        in1 when sel = '1';

end Behavioral;
```

TESTBENCH

```
ENTITY MUX_MTest IS
END MUX_MTest;

ARCHITECTURE behavior OF MUX_MTest IS

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

    COMPONENT MUX_M
    PORT(
         in0 : IN  std_logic_VECTOR(15 DOWNTO 0);
         in1 : IN  std_logic_VECTOR(15 DOWNTO 0);
         sel : IN  std_logic;
         out0 : OUT  std_logic_VECTOR(15 DOWNTO 0)
        );
    END COMPONENT;


   --Inputs
         signal in0 : std_logic_vector(15 downto 0) :=
(others => '0');
   signal in1 : std_logic_vector(15 downto 0) :=
(others => '0');
   signal sel : std_logic := '0';

    --Outputs
   signal out0 : std_logic_vector(15 downto 0) :=
(others => '0');

   -- No clocks detected in port list. Replace
<clock> below with
   -- appropriate port name

   --constant <clock>_period : time := 10 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: MUX_M PORT MAP (
          in0 => in0,
          in1 => in1,
          sel => sel,
          out0 => out0
        );

   -- Clock process definitions
   --<clock>_process :process
   --begin
```

```
        --<clock> <= '0';
        --wait for <clock>_period/2;
        --<clock> <= '1';
        --wait for <clock>_period/2;
   --end process;


   -- Stimulus process
   stim_proc: process
   begin
      wait for 100 ns;
      in0 <= "0000000000000000";
        in1 <= "1111111111111111";

        sel <= '0';
        wait for 10ns;
        assert out0 = "0000000000000000";

        sel <= '1';
        wait for 10ns;
        assert out0 = "1111111111111111";


      wait;
   end process;

END;
```

FUNCTION UNIT

CODE

```
entity FunctionUnit_16bit is
    Port ( A : in   STD_LOGIC_VECTOR(15 DOWNTO 0);
           B : in   STD_LOGIC_VECTOR(15 DOWNTO 0);
           FSselect : in  STD_LOGIC_VECTOR(4 DOWNTO
0);
                C : out STD_LOGIC;
                V : out STD_LOGIC;
                N : out STD_LOGIC;
                Z : out STD_LOGIC;
           F : out   STD_LOGIC_VECTOR(15 DOWNTO 0)
                );
end FunctionUnit_16bit;

architecture Behavioral of FunctionUnit_16bit is

 component ALU_16bit
 Port ( A : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
       B : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
       Cin : in  STD_LOGIC;
       S : in  STD_LOGIC_VECTOR(2 DOWNTO 0);
       C : out  STD_LOGIC;
          V : out  STD_LOGIC;
       N : out  STD_LOGIC;
       Z : out  STD_LOGIC;
       G : out  STD_LOGIC_VECTOR(15 downto 0)
        );
 end component;

 component Shifter_16bit
 Port ( B : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
        S : in  STD_LOGIC_VECTOR(1 DOWNTO 0);
        H : out  STD_LOGIC_VECTOR(15 DOWNTO 0)
          );
 end component;

 component FU_mux2to1
 Port ( in0   : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
        in1   : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
        S     : in  STD_LOGIC;
        out0  : out STD_LOGIC_VECTOR(15 DOWNTO 0)
              );
 end component;

--signals
SIGNAL Gout_sig, Hout_sig, Fout_sig:
STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL Cout_sig, Vout_sig, Nout_sig, Zout_sig:
STD_LOGIC;

begin

  ALU:ALU_16bit PORT MAP(
  A => A,
  B => B,
  S(2) => FSselect(1),
  S(1) => FSselect(2),
  S(0) => FSselect(3),
  Cin =>  FSselect(4),
  C => Cout_sig,
  V => Vout_sig,
  N => Nout_sig,
  Z => Zout_sig,
  G => Gout_sig
  );
  Shifter:Shifter_16bit PORT MAP(
  B => B,
  S(0) => FSselect(1),
  S(1) => FSselect(2),
  H => Hout_sig
  );

  Mux:FU_mux2to1 PORT MAP(
  in0 => Gout_sig,
  in1 => Hout_sig,
  S => FSselect(0),
  out0 => Fout_sig
  );

  C <= Cout_sig;
  V <= Vout_sig;
  N <= Nout_sig;
  Z <= Zout_sig;
  F <= Fout_sig;

end Behavioral;
```

TESTBENCH

```
ENTITY FunctionUnit_16bitTest IS
END FunctionUnit_16bitTest;

ARCHITECTURE behavior OF FunctionUnit_16bitTest IS

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

    COMPONENT FunctionUnit_16bit
    PORT(
         A : IN  std_logic_vector(15 downto 0);
         B : IN  std_logic_vector(15 downto 0);
         FSselect : IN  std_logic_vector(4 downto 0);
         C : OUT  std_logic;
            V : OUT  std_logic;
         N : OUT  std_logic;
         Z : OUT  std_logic;
         F : OUT  std_logic_vector(15 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal A : std_logic_vector(15 downto 0) :=
(others => '0');
   signal B : std_logic_vector(15 downto 0) :=
(others => '0');
   signal FSselect : std_logic_vector(4 downto 0) :=
(others => '0');

     --Outputs
   signal C : std_logic;
     signal V : std_logic;
   signal N : std_logic;
   signal Z : std_logic;
   signal F : std_logic_vector(15 downto 0);
   -- No clocks detected in port list. Replace
<clock> below with
   -- appropriate port name

   --constant <clock>_period : time := 10 ns;

BEGIN

     -- Instantiate the Unit Under Test (UUT)
   uut: FunctionUnit_16bit PORT MAP (
          A => A,
          B => B,
          FSselect => FSselect,
          C => C,
          V => V,
          N => N,
          Z => Z,
          F => F
        );

   -- Clock process definitions
   --<clock>_process :process
   --begin
         --<clock> <= '0';
         --wait for <clock>_period/2;
         --<clock> <= '1';
         --wait for <clock>_period/2;
   --end process;


   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;
```

```
        A <= "0000000000000000";
        B <= "1000000000000001";

        --THE FS SELECTS ARE INPUTTED BACKWARDS.
        --EG. 00101 IS SUBTRACT, BUT VECTOR INPUT
WILL BE "10100".
        --COULD CHANGE, BUT EASIER IN THE LONG RUN.
        --AS OF 2/4/2014 6:32PM ALL OF THIS SHIT
WORKS.

        A <= "1000000000000001";
        --INCREMENT A
        FSselect <= "10000";
        wait for 10ns;
        assert F = "1000000000000010";
        assert Z = '0';
        assert V = '0';
        assert N = '1';
        assert C = '0';

        --ADD A + B (00010)
        FSselect <= "01000";
        wait for 10ns;
        assert F = "0000000000000010";
        assert Z = '0';
        assert V = '1';
        assert N = '0';
        assert C = '1';

        --A + B + 1
        FSselect <= "00011";
        wait for 10ns;
        assert F = "0000000000000100";

        -- A + not B
        B <= "1111111111111110";
        FSselect <= "00100";
        wait for 10ns;
        assert F = "0000000000000010";

       -- A - B
        A <= "0000000000000010";
        B <= "0000000000000001";
        FSselect <= "00101";
        wait for 10ns;
        assert F = "0000000000000001";
```

```
-- A - 1
FSselect <= "00110";
wait for 10ns;
assert F = "0000000000000001";

-- TRANSFER A
FSselect <= "00111";
wait for 10ns;
assert F = "0000000000000010";

-- AND
FSselect <= "01000";
wait for 10ns;
assert F = "0000000000000000";

--OR
A <= "0000000000000100";
FSselect <= "01010";
wait for 10ns;
assert F = "0000000000000101";

--XOR
A <= "0000000000001000";
FSselect <= "01100";
wait for 10ns;
assert F = "0000000000001001";

--NOT A
FSselect <= "01110";
wait for 10ns;
assert F = "1111111111110111";


B <= "0000000000001111";

--B TRANSFER
FSselect <= "10000";

wait for 10ns;
assert F = "0000000000001111";

--SHIFT B RIGHT
FSselect <= "10100";
wait for 10ns;
assert F = "0000000000000111";

--SHIFT B LEFT
FSselect <= "11000";
```

```
        wait for 10ns;
        assert F = "0000000000011110";


      --wait for <clock>_period*10;

      -- insert stimulus here

      wait;
   end process;

END;
```

D MUX

CODE

```
entity Datapath_Dmux2to1 is
    Port ( in0 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
           in1 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
           S : in  STD_LOGIC;
           out0 : out  STD_LOGIC_VECTOR(15 DOWNTO 0)
                );
end Datapath_Dmux2to1;

architecture Behavioral of Datapath_Dmux2to1 is

begin
out0 <= in0 when S <= '0' else
        in1 when S <= '1';

end Behavioral;
```

TESTBENCH

```
ENTITY Datapath_Dmux2to1Test IS
END Datapath_Dmux2to1Test;

ARCHITECTURE behavior OF Datapath_Dmux2to1Test IS

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

    COMPONENT Datapath_Dmux2to1
    PORT(
         in0 : IN  std_logic_vector(15 downto 0);
         in1 : IN  std_logic_vector(15 downto 0);
         S : IN  std_logic;
         out0 : OUT  std_logic_vector(15 downto 0)
```

```
        );
    END COMPONENT;


   --Inputs
   signal in0 : std_logic_vector(15 downto 0) :=
(others => '0');
   signal in1 : std_logic_vector(15 downto 0) :=
(others => '0');
   signal S : std_logic := '0';

     --Outputs
   signal out0 : std_logic_vector(15 downto 0);
   -- No clocks detected in port list. Replace
<clock> below with
   -- appropriate port name

   --constant <clock>_period : time := 10 ns;

BEGIN

     -- Instantiate the Unit Under Test (UUT)
   uut: Datapath_Dmux2to1 PORT MAP (
          in0 => in0,
          in1 => in1,
          S => S,
          out0 => out0
        );

   -- Clock process definitions
   --<clock>_process :process
   --begin
         --<clock> <= '0';
         --wait for <clock>_period/2;
         --<clock> <= '1';
         --wait for <clock>_period/2;
   --end process;


   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;
         in0 <= "0000000000000000";
         in1 <= "1111111111111111";

         S <= '0';
```

```
            wait for 10ns;
            assert out0 <= "0000000000000000";

            S <= '1';
            wait for 10ns;
            assert out0 <= "1111111111111111";



        --wait for <clock>_period*10;
        -- insert stimulus here

        wait;
     end process;

END;
```

PROCESSER

CODE

```
entity Processer is
    Port ( reset : in STD_LOGIC;
             Clk : in STD_LOGIC;
           Datapath_out: out STD_LOGIC_VECTOR(15
DOWNTO 0);
                PCouttest: out STD_LOGIC_VECTOR(15
DOWNTO 0);
                CARouttest : out STD_LOGIC_VECTOR(7
DOWNTO 0);
                TBouttest : out STD_LOGIC_VECTOR(15
downto 0)

           );
end Processer;

architecture Behavioral of Processer is

component Program_counter
Port ( in0 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
          reset : IN std_logic;
       PI : in  STD_LOGIC;
       PL : in  STD_LOGIC;
       Clk : in  STD_LOGIC;
       out0 : inout  STD_LOGIC_VECTOR(15 DOWNTO 0)
         );
```

```
end component;

component extend
Port ( in0 : in  STD_LOGIC_VECTOR(5 DOWNTO 0);
       out0 : out  STD_LOGIC_VECTOR(15 DOWNTO 0)
                );
end component;

component NOT_cond
Port ( in0 : in  STD_LOGIC;
       in1 : in  STD_LOGIC;
       out0 : out  STD_LOGIC;
       out1 : out  STD_LOGIC);
end component;

component MUX_S
Port ( in0 : in  STD_LOGIC;
           in1 : in  STD_LOGIC;
           in2 : in  STD_LOGIC;
           in3 : in  STD_LOGIC;
           in4 : in  STD_LOGIC;
           in5 : in  STD_LOGIC;
           in6 : in  STD_LOGIC;
           in7 : in  STD_LOGIC;
           sel : in  STD_LOGIC_VECTOR(2 DOWNTO 0);
           out0 : out STD_LOGIC
         );
end component;

component instruction_reg
Port ( in0 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
       sel : in  STD_LOGIC;
       opcode : out  STD_LOGIC_VECTOR(6 DOWNTO 0);
       DR : out  STD_LOGIC_VECTOR(2 DOWNTO 0);
       SA : out  STD_LOGIC_VECTOR(2 DOWNTO 0);
         Clk : in STD_LOGIC;
       SB : out  STD_LOGIC_VECTOR(2 DOWNTO 0)
                );
end component;

component MUX_C
Port ( in0 : in  STD_LOGIC_VECTOR(7 DOWNTO 0);
           in1 : in  STD_LOGIC_VECTOR(7 DOWNTO 0);
           sel : in  STD_LOGIC;
           out0 : out  STD_LOGIC_VECTOR(7 DOWNTO 0)
              );
end component;
```

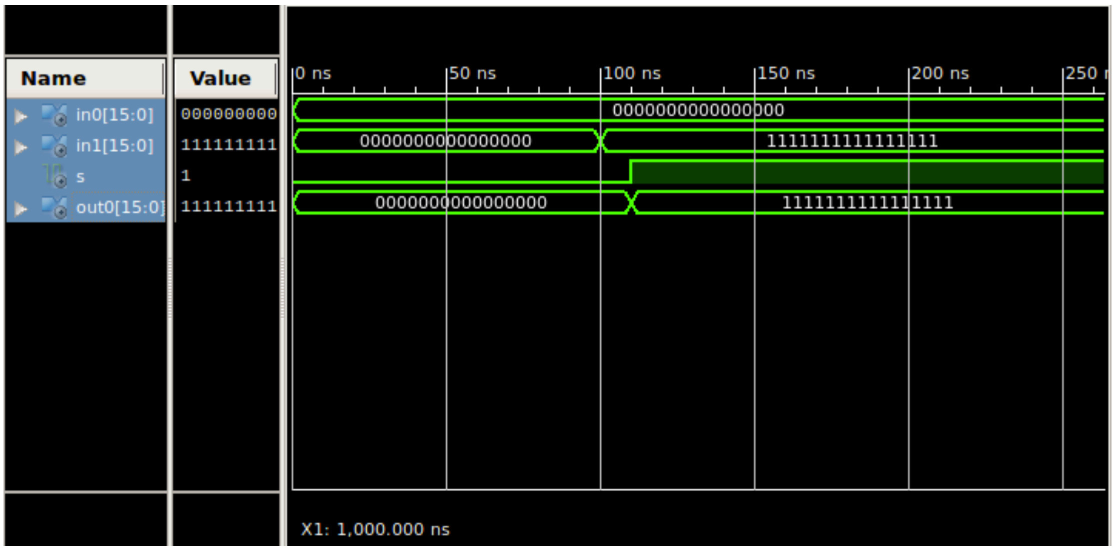
```
component control_access_reg
Port (      in0 : in   STD_LOGIC_VECTOR(7 DOWNTO 0);
                 Clk : in STD_LOGIC;
                 reset : in STD_LOGIC;
                 out0 : inout  STD_LOGIC_VECTOR(7
DOWNTO 0);
            sel : in   STD_LOGIC
                 );
end component;

component control_memory
Port ( MW : out std_logic;
           MM : out std_logic;
           RW : out std_logic;
         MD : out std_logic;
           FS : out std_logic_vector(4 downto 0);
         MB : out std_logic;
           TB : out std_logic;
           TA : out std_logic;
           TD : out std_logic;
           PL : out std_logic;
         PI : out std_logic;
         IL : out std_logic;
         MC : out std_logic;
         MS : out std_logic_vector(2 downto 0);
         NA : out std_logic_vector(7 downto 0);
         IN_CAR : in std_logic_vector(7 downto 0)
           );
end component;

component zeroFill
    Port ( in0 : in   STD_LOGIC_VECTOR(2 DOWNTO 0);
           out0 : out  STD_LOGIC_VECTOR(15 DOWNTO 0)
                 );
end component;

component MUX_M
Port ( in0 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
            in1 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
            sel : in  STD_LOGIC;
            out0 : out  STD_LOGIC_VECTOR(15 DOWNTO 0)
                 );
end component;

component Memory
Port (
   address : in unsigned(15 downto 0);
   write_data : in std_logic_vector(15 downto 0);
   MW: in std_logic;
      read_data : out std_logic_vector(15 downto 0));
end component;

component register_file
Port ( s0_Amux9to16 : in  STD_LOGIC_VECTOR(2 DOWNTO
0);
                s0_Bmux9to16 : in  STD_LOGIC_VECTOR(2
DOWNTO 0);
          s0_dec3to9 : in  STD_LOGIC_VECTOR(2 DOWNTO 0);
                TD : in STD_LOGIC;
                TA : in STD_LOGIC;
                TB : in STD_LOGIC;
                loadEnable : in  STD_LOGIC;
          DData       : in  STD_LOGIC_VECTOR(15 DOWNTO
0);
          Clk :  in  STD_LOGIC;
          Adata : out  STD_LOGIC_VECTOR(15 DOWNTO 0);
          Bdata : out  STD_LOGIC_VECTOR(15 DOWNTO 0)
              );
end component;

component FunctionUnit_16bit
        Port ( A : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
                B : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
                FSselect : in  STD_LOGIC_VECTOR(4 DOWNTO
0);
                        C : out STD_LOGIC;
                        N : out STD_LOGIC;
                        V : out STD_LOGIC;
                        Z : out STD_LOGIC;
                F : out  STD_LOGIC_VECTOR(15 DOWNTO 0)
                 );
end component;

component Datapath_Bmux2to1
        Port ( in0 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
                in1 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
                S : in  STD_LOGIC;
                out0 : out  STD_LOGIC_VECTOR(15 DOWNTO 0)
                 );
end component;

component Datapath_Dmux2to1
        Port ( in0 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
                in1 : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
                S : in  STD_LOGIC;
                out0 : out  STD_LOGIC_VECTOR(15 DOWNTO 0)
```

```
           );
end component;

--signals
signal MUXS_sig,not_c_sig,not_z_sig,Cout_sig,
          Nout_sig, Vout_sig, Zout_sig : STD_LOGIC;

signal DR_sig, SA_sig, SB_sig : STD_LOGIC_VECTOR(2
DOWNTO 0);

signal opcode_sig : STD_LOGIC_VECTOR(6 DOWNTO 0);

signal MUXC_sig, CAR_sig : STD_LOGIC_VECTOR(7 DOWNTO
0);

signal PCout_sig, MemMout_sig, extend_sig,
       zeroFill_sig, BusA_sig, BusB_sig,
          muxB_sig, muxM_sig, FU_sig,
          BusD_sig: STD_LOGIC_VECTOR(15 DOWNTO 0);

signal CM_sig : STD_LOGIC_VECTOR(27 DOWNTO 0);

begin
--port maps

  -- PROGRAM COUNTER
  PC:program_counter PORT MAP(
  in0 => extend_sig,
  reset => reset,
  PI => CM_sig(14),
  PL => CM_sig(13),
  Clk => Clk,
  out0 => PCout_sig
  );

  --EXTEND
  PCExtend:extend PORT MAP(
  in0(5 downto 3) => DR_sig,
  in0(2 downto 0) => SB_sig,
  out0 => extend_sig
  );

  --INVERTER
  inverter:NOT_cond PORT MAP(
  in0 => Cout_sig,
  in1 => Zout_sig,
  out0 => not_c_sig,
  out1 => not_z_sig
```

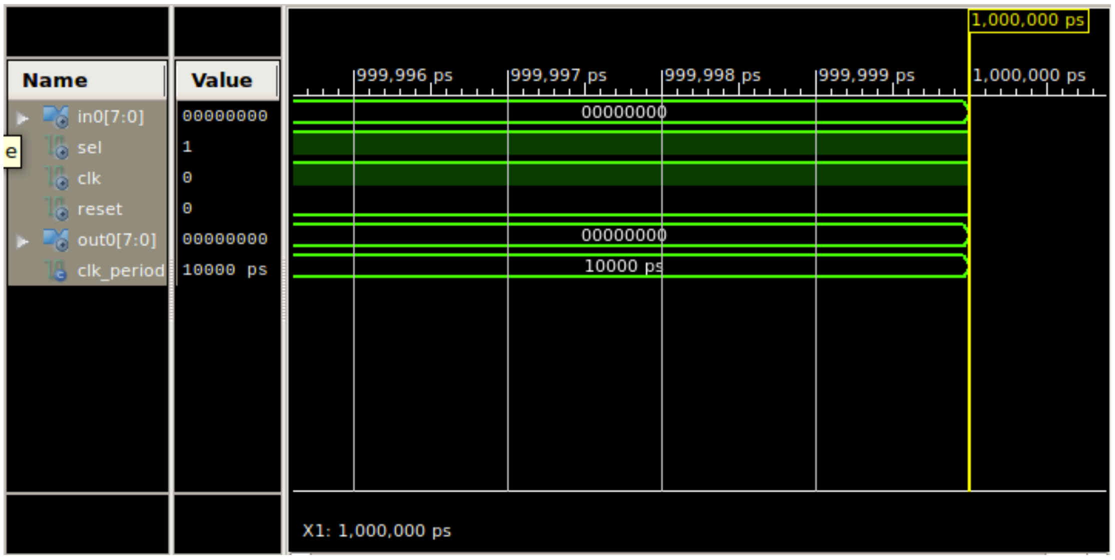
```vhdl
);

--MUX S
MUXS:MUX_S PORT MAP(
in0 => '0',
in1 => '1',
in2 => Cout_sig,
in3 => Vout_sig,
in4 => Zout_sig,
in5 => Nout_sig,
in6 => not_c_sig,
in7 => not_z_sig,
sel(0) => CM_sig(17),
sel(1) => CM_sig(18),
sel(2) => CM_sig(19),
out0 => MUXS_sig
);

--INSTRUCTION REGISTER
IR:instruction_reg PORT MAP(
  in0 => MemMout_sig,
  sel => CM_sig(15),
  opcode => opcode_sig,
  DR => DR_sig,
  SA => SA_sig,
    SB => SB_sig,
    Clk => Clk
    );

   --MUX C
   MUXC:MUX_C PORT MAP(
   in0(0) => CM_sig(20),
   in0(1) => CM_sig(21),
   in0(2) => CM_sig(22),
   in0(3) => CM_sig(23),
   in0(4) => CM_sig(24),
   in0(5) => CM_sig(25),
   in0(6) => CM_sig(26),
   in0(7) => CM_sig(27),
 in1(6 downto 0) => opcode_sig,
   in1(7) => '0',
 sel => CM_sig(16),
 out0 => MUXC_sig
   );

   --CONTROL ACCESS REGISTER
   CAR:control_access_reg PORT MAP(
     in0 => MUXC_sig,
      Clk => Clk,
    reset => reset,
  sel => MUXS_sig,
  out0 => CAR_sig
  );

--CONTROL MEMORY
CM:control_memory PORT MAP(
  MW => CM_sig(0),
      MM => CM_sig(1),
      RW => CM_sig(2),
    MD => CM_sig(3),
      FS(0) => CM_sig(4),
      FS(1) => CM_sig(5),
      FS(2) => CM_sig(6),
      FS(3) => CM_sig(7),
      FS(4) => CM_sig(8),
    MB => CM_sig(9),
      TB => CM_sig(10),
      TA => CM_sig(11),
      TD => CM_sig(12),
      PL => CM_sig(13),
    PI => CM_sig(14),
    IL => CM_sig(15),
    MC => CM_sig(16),
    MS(0) => CM_sig(17),
    MS(1) => CM_sig(18),
    MS(2) => CM_sig(19),
    NA(0) => CM_sig(20),
    NA(1) => CM_sig(21),
    NA(2) => CM_sig(22),
    NA(3) => CM_sig(23),
    NA(4) => CM_sig(24),
    NA(5) => CM_sig(25),
    NA(6) => CM_sig(26),
    NA(7) => CM_sig(27),
    IN_CAR => CAR_sig
      );

--REGISTER FILE
registerfile:register_file PORT MAP(
s0_dec3to9 => DR_sig,
S0_Amux9to16 => SA_sig,
S0_Bmux9to16 => SB_sig,
TD => CM_sig(12),
TA => CM_sig(11),
TB => CM_sig(10),
loadEnable => CM_sig(2),
  Clk => Clk,
  DData => BusD_sig,
  AData => BusA_sig,
  BData => BusB_sig
  );

  --ZERO FILL
  fillWithZero:zeroFill PORT MAP(
  in0 => SB_sig,
out0 => zeroFill_sig
  );

  --B MUX
  MUXB:Datapath_Bmux2to1 PORT MAP(
  in0 => BusB_sig,
  in1 => zeroFill_sig,
S => CM_sig(9),
  out0 => muxB_sig
  );

  --M MUX
  MUXM:MUX_M PORT MAP(
  in0 => busA_sig,
  in1 => PCout_sig,
  sel => CM_sig(1),
  out0 => muxM_sig
  );

  --FUNCTIONAL UNIT
  FunctionUnit:FunctionUnit_16bit PORT MAP(
A => BusA_sig,
  B => muxB_sig,
  FSselect(4) => CM_sig(8),
  FSselect(3) => CM_sig(7),
  FSselect(2) => CM_sig(6),
  FSselect(1) => CM_sig(5),
  FSselect(0) => CM_sig(4),
  C => Cout_sig,
  N => Nout_sig,
  V => Vout_sig,
  Z => Zout_sig,
  F => FU_sig
  );

  --MEMORY M
  MemoryM:Memory PORT MAP (
  address => unsigned(muxM_sig),
  write_data => muxB_sig,
   MW => CM_sig(0),
     read_data => MemMout_sig
   );

     --MUX D
     MUXD:Datapath_Dmux2to1 PORT MAP(
     in0 => FU_sig,
     in1 => MemMout_sig,
     S => CM_sig(3),
     out0 => BusD_sig
     );

   CARouttest <= CAR_sig;
   Datapath_out <= BusD_sig;
     PCOuttest <= PCout_sig;
     TBouttest <= memMout_sig;
end Behavioral;

TESTBENCH
ENTITY ProcesserTest IS
END ProcesserTest;

ARCHITECTURE behavior OF ProcesserTest IS

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

    COMPONENT Processer
    PORT(
         reset : IN  std_logic;
         Clk : IN  std_logic;
         Datapath_out : OUT  std_logic_vector(15
downto 0);
              PCouttest : out std_logic_vector(15
downto 0);
              CARouttest : out std_logic_vector(7
downto 0);
              TBouttest : out STD_LOGIC_VECTOR(15
downto 0)
        );
    END COMPONENT;


   --Inputs
   signal reset : std_logic := '0';
   signal Clk : std_logic := '0';
```

```
     --Outputs
     signal TBouttest : STD_LOGIC_VECTOR(15 downto
0);
   signal Datapath_out : std_logic_vector(15 downto
0);
     signal PCouttest : std_logic_vector(15 downto
0);
   signal CARouttest : std_logic_vector(7 downto 0);

   -- Clock period definitions
   constant Clk_period : time := 10 ns;

BEGIN

     -- Instantiate the Unit Under Test (UUT)
   uut: Processer PORT MAP (
          reset => reset,
          Clk => Clk,
          Datapath_out => Datapath_out,
               PCouttest => PCouttest,
               CARouttest => CARouttest,
               TBouttest => TBouttest

        );

   -- Clock process definitions
   Clk_process :process
   begin
         Clk <= '0';
         wait for Clk_period/2;
         Clk <= '1';
         wait for Clk_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;

         reset <= '1';
      wait for 10ns;
         reset <= '0';

      -- insert stimulus here

      wait;
```

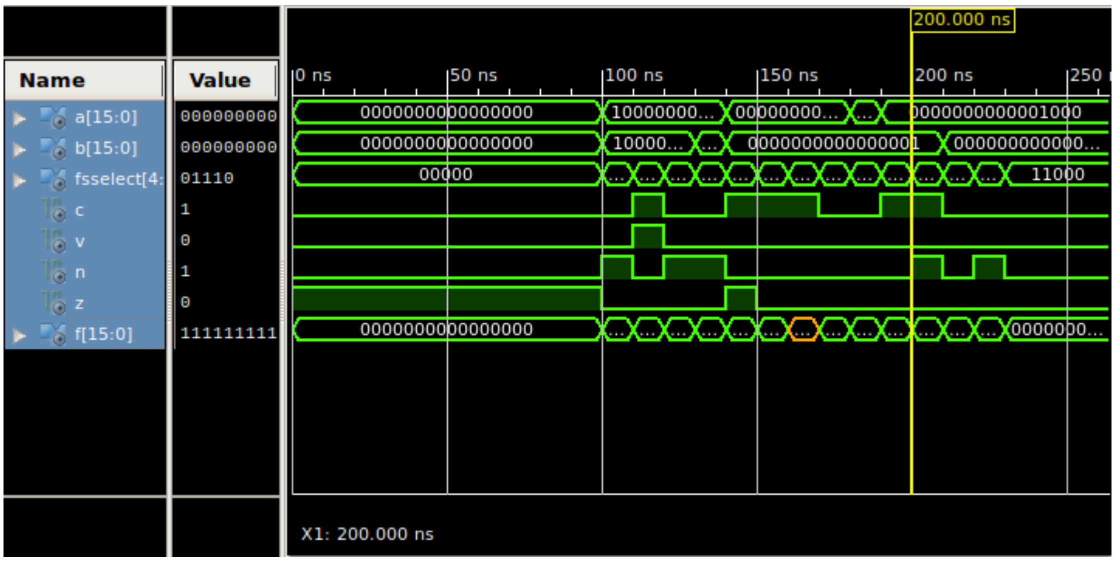
```
    end process;

END;
```



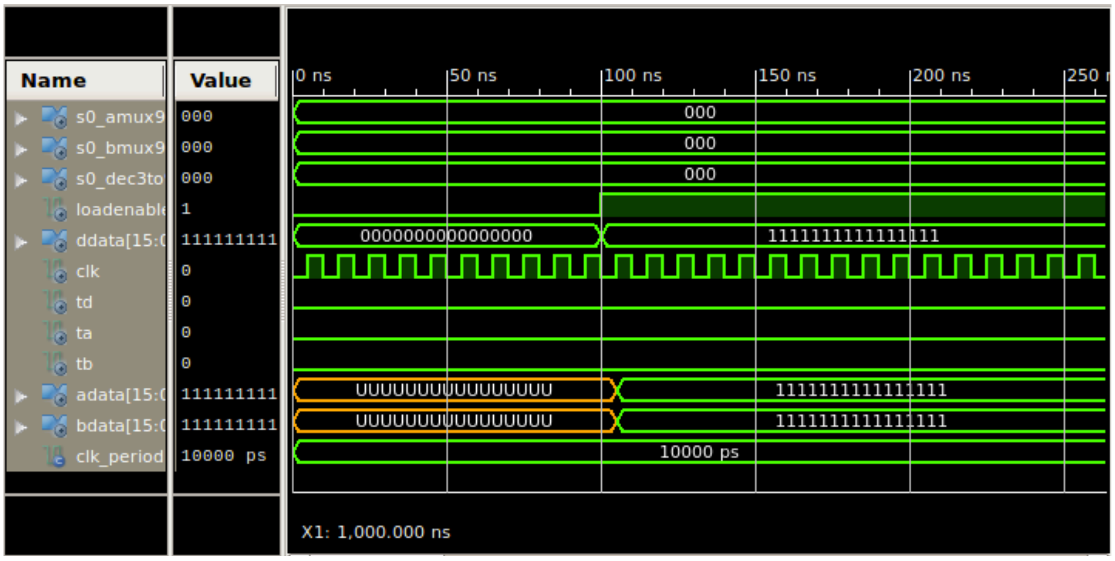
D MUX 2 TO 1 WAVEFORM



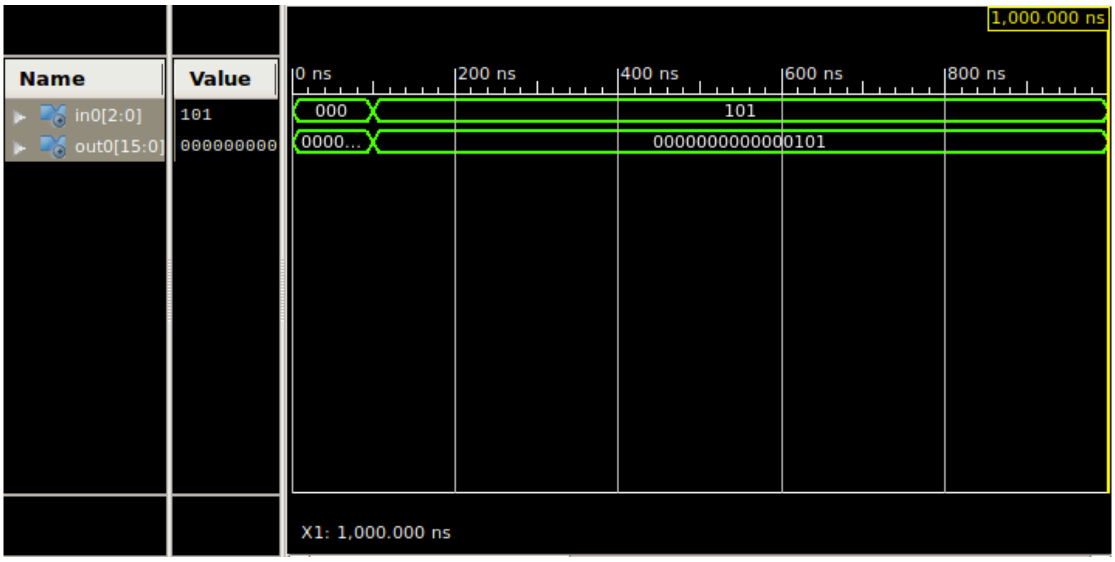
CAR WAVEFORM



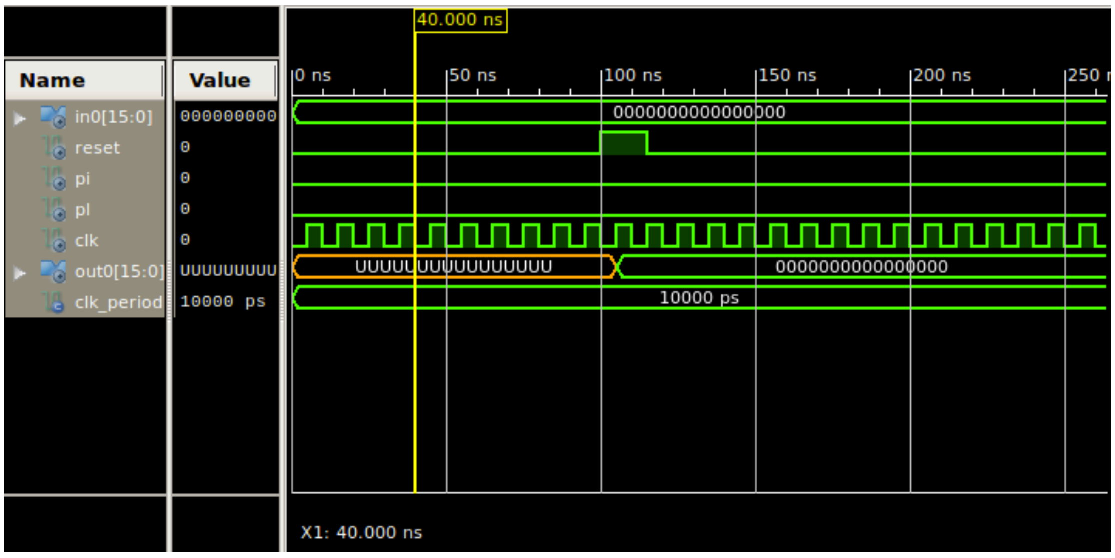
FUNCTIONAL UNIT WAVEFORM



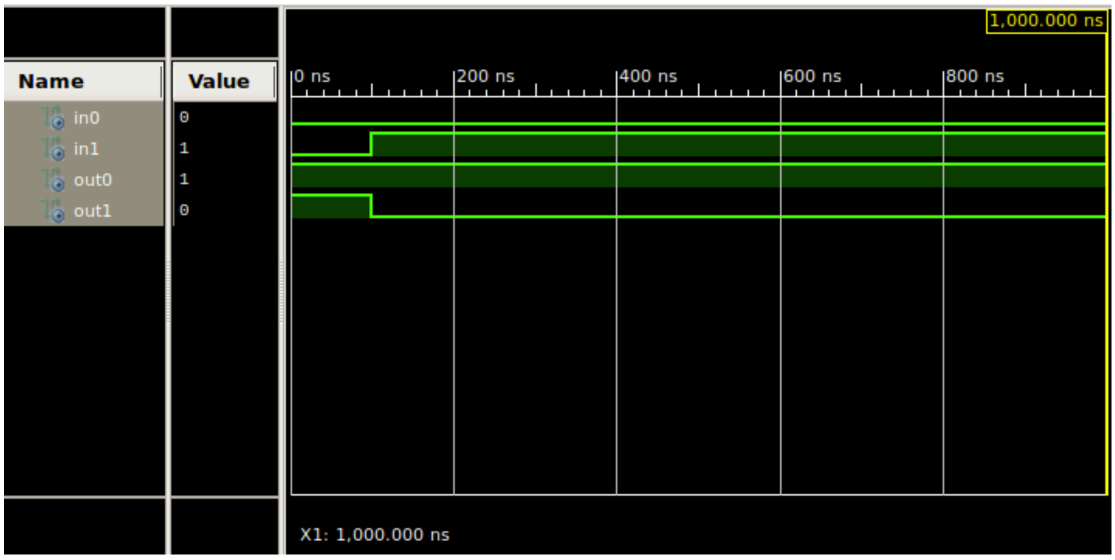
REGISTER FILE WAVEFORM



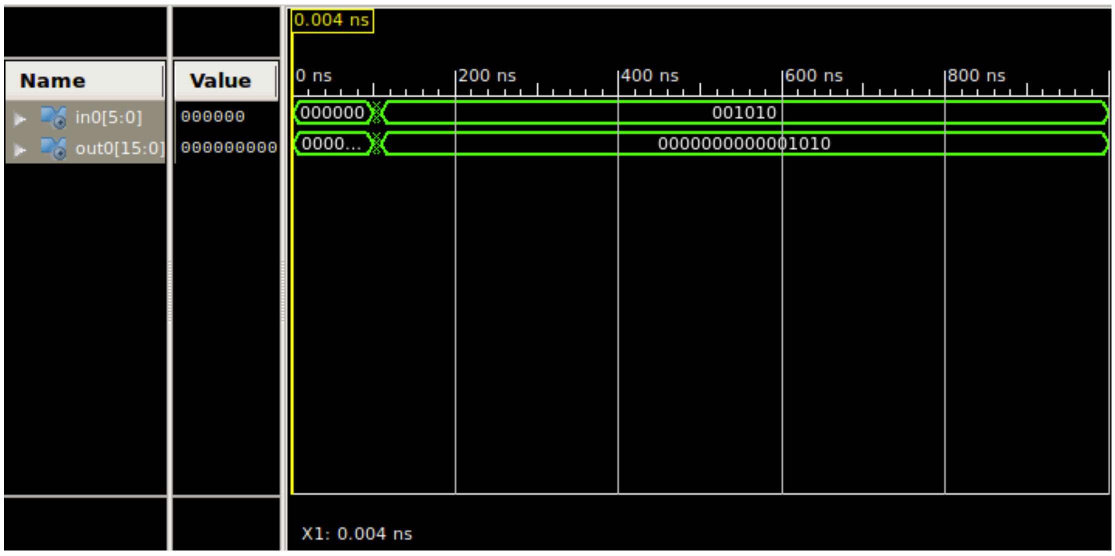
ZERO FILL WAVEFORM



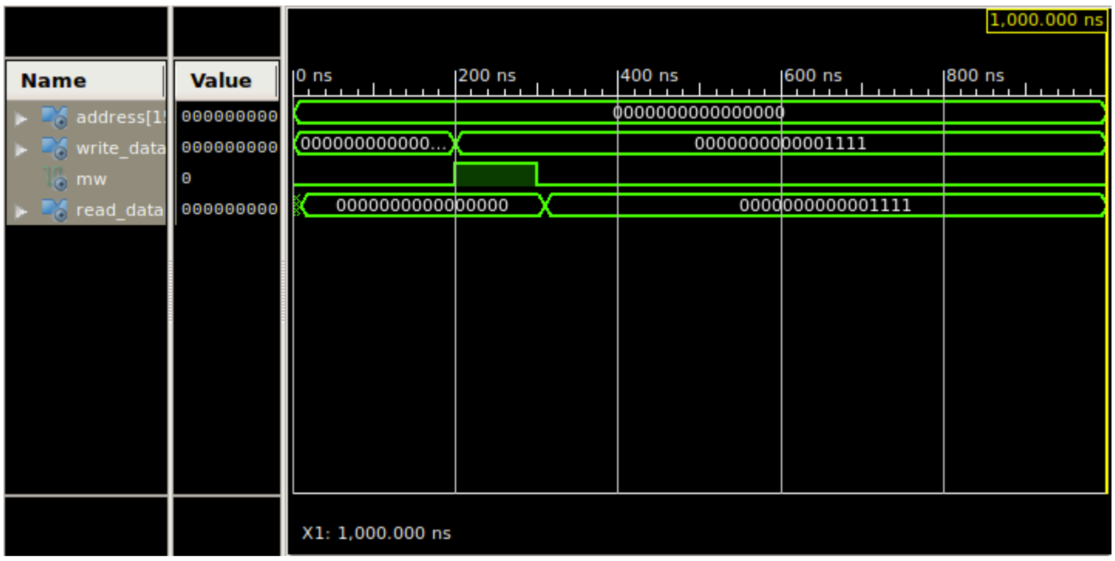
PROGRAM COUNTER WAVEFORM



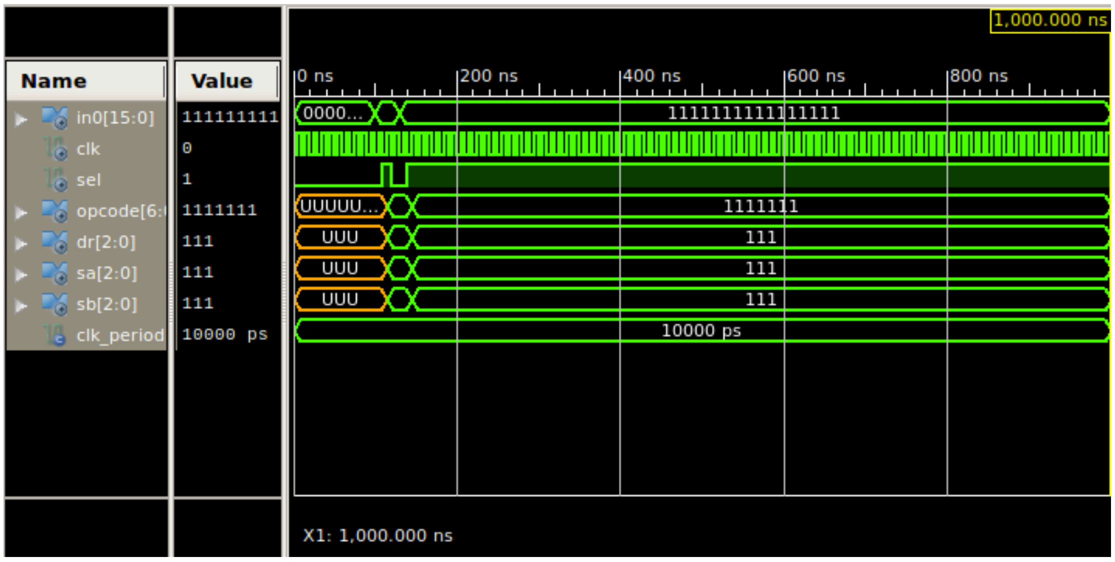
INVERTER WAVEFORM



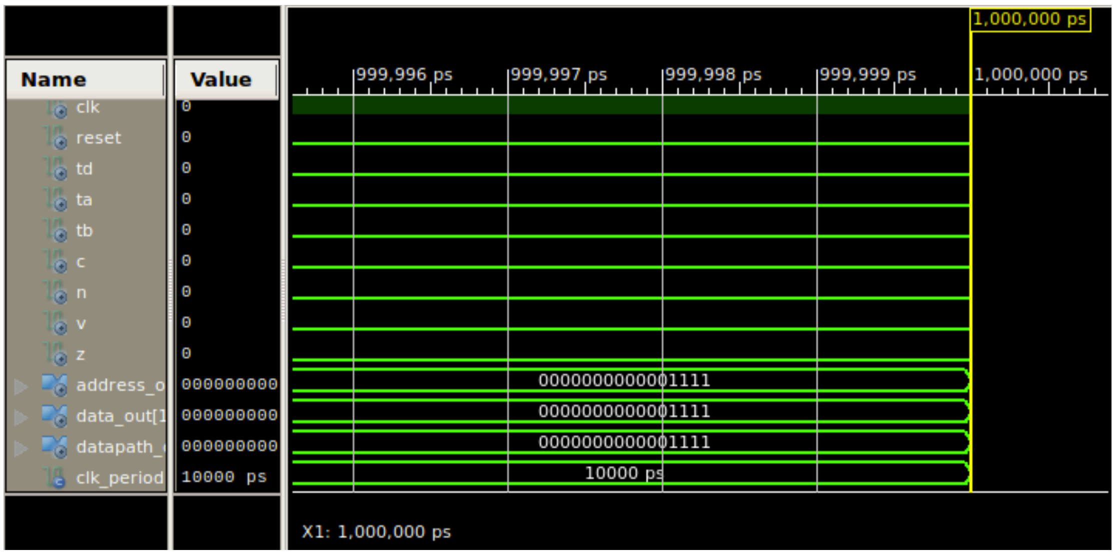
EXTEND WAVEFORM



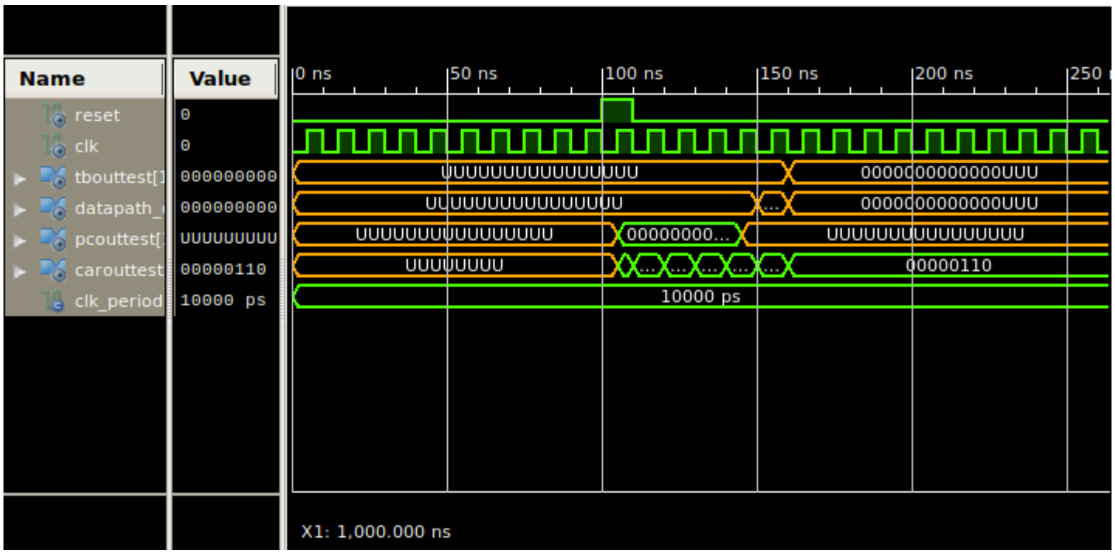
MEMORY WAVEFORM



INSTRUCTION REGISTER WAVEFORM



DATAPATH WAVEFORM



PROCESSOR WAVEFORM