

UNIVERSITY OF DUBLIN,  
TRINITY COLLEGE



---

COMPUTER ARCHITECTURE I (CS2022)  
PROJECT 2 - MICROCODED INSTRUCTION SET PROCESSOR

---

*Author:*

Edmond O'FLYNN 12304742

*Lecturer:*

Dr. Michael MANZKE

March 25, 2016

# Contents

<b>1</b>	<b>VHDL Component Source Code</b>	<b>1</b>
1.1	ALU Unit . . . . .	1
1.2	Control Address Register . . . . .	2
1.3	Control Memory . . . . .	3
1.4	Datapath . . . . .	9
1.5	Decoder 4-9 Bit . . . . .	11
1.6	Extended Programme Counter . . . . .	12
1.7	Full Adder . . . . .	12
1.8	Function Unit . . . . .	12
1.9	Instruction Register . . . . .	14
1.10	Logic Circuit A-B . . . . .	14
1.11	Logic Circuit B . . . . .	15
1.12	Memory Module . . . . .	17
1.13	Microprogramme Controller . . . . .	21
1.14	Mux 2-1 Bit . . . . .	24
1.15	Mux 2-8 Bit . . . . .	24
1.16	Mux 2-16 Bit . . . . .	25
1.17	Mux 3-1 Bit . . . . .	25
1.18	Mux 8-1 Bit . . . . .	26
1.19	Mux 9-16 Bit . . . . .	26
1.20	Programme Counter . . . . .	27
1.21	Project 2 Top Level . . . . .	27
1.22	Register . . . . .	29
1.23	Register File . . . . .	29
1.24	Ripple Adder . . . . .	33
1.25	Shifter . . . . .	36
1.26	Zero Fill . . . . .	39
<b>2</b>	<b>Component Test Benches</b>	<b>40</b>
2.1	Control Address Register . . . . .	40
2.2	Control Memory . . . . .	41
2.3	Decoder 4-9 bit . . . . .	42
2.4	Extended Programme Counter . . . . .	45
2.5	Instruction Register . . . . .	46
2.6	Memory Module . . . . .	47
2.7	Mux 2-8 Bit . . . . .	48
2.8	Mux 8-1 Bit . . . . .	49
2.9	Mux 9-16 Bit . . . . .	51
2.10	Programme Counter . . . . .	53
2.11	Project 2 Top Level . . . . .	55
2.12	Zero Fill . . . . .	56
<b>3</b>	<b>Results of Test Benches</b>	<b>57</b>
3.1	Control Address Register . . . . .	58
3.2	Control Memory . . . . .	58
3.3	Decoder 4-9 bit . . . . .	58
3.4	Extended Programme Counter . . . . .	58
3.5	Instruction Register . . . . .	59
3.6	Memory Module . . . . .	59
3.7	Mux 2-1 Bit . . . . .	59
3.8	Mux 8-1 . . . . .	59

3.9	Mux 9-16 Bit . . . . .	59
3.10	Programme Counter . . . . .	60
3.11	Project 2 Top Level . . . . .	60
3.12	Register File . . . . .	60
3.13	Zero Fill . . . . .	60

# 1 VHDL Component Source Code

## 1.1 ALU Unit

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

entity alu_unit is
    Port(
        a_in, b_in : in STD_LOGIC_VECTOR(15 downto 0);
        G_select : in STD_LOGIC_VECTOR(3 downto 0);
        V, C : out STD_LOGIC; -- flags
        G : out STD_LOGIC_VECTOR(15 downto 0)
    );

end alu_unit;

architecture Behavioral of alu_unit is

    --components in ALU
    --ripple adder
    Component RippleAdder
        Port(
            A, B : in STD_LOGIC_VECTOR(15 downto 0);
            Cin : in STD_LOGIC;
            Cout, V_out : out STD_LOGIC;
            G_out : out STD_LOGIC_VECTOR(15 downto 0)
        );
    End Component;
    --a b logic for and or xor not
    Component logic_circuit_a_b
        Port(
            a_logic_in, b_logic_in : in STD_LOGIC_VECTOR(15 downto 0);
            select_in : in STD_LOGIC_VECTOR(1 downto 0);
            logic_output_a_b : out STD_LOGIC_VECTOR(15 downto 0)
        );
    End Component;
    --b logic circuit
    Component logic_circuit_b
        Port(
            B : in STD_LOGIC_VECTOR(15 downto 0);
            S_in : in STD_LOGIC_VECTOR(1 downto 0);
            Y_out : out STD_LOGIC_VECTOR(15 downto 0)
        );
    End Component;
    --2-1 mux
    Component Mux2to16
        Port(
            In0, In1 : in STD_LOGIC_VECTOR(15 downto 0);
            s : in STD_LOGIC;
            Z : out STD_LOGIC_VECTOR(15 downto 0)
        );
    End Component;

    signal logic_out, logic_output_a_b, ripple_out : STD_LOGIC_VECTOR(15 downto 0);
```

```

begin
    --instantiation of components
    r_adder: RippleAdder PORT MAP(
        A => a_in,
        B => b_in,
        Cin => G_select(0),
        Cout => C,
        V_out => V,
        G_out => ripple_out
    );

    logic_circuit_a_b00: logic_circuit_a_b PORT MAP(
        a_logic_in => a_in,
        b_logic_in => b_in,
        select_in => G_select(2 downto 1),
        logic_output_a_b => logic_output_a_b
    );

    logic_circuit_b00 : logic_circuit_b PORT MAP(
        B => b_in,
        S_in => G_select(2 downto 1),
        Y_out => logic_out
    );

    mux_2_1600: Mux2to16 PORT MAP(
        In0 => ripple_out,
        In1 => logic_output_a_b,
        s => G_select(3),
        Z => G
    );

end Behavioral;

```

## 1.2 Control Address Register

```

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

entity ControlAddressRegister is
    Port( car_in : in STD_LOGIC_VECTOR(7 downto 0);
          s_car, reset : in STD_LOGIC;
          car_out : out STD_LOGIC_VECTOR(7 downto 0)
        );
end ControlAddressRegister;

architecture Behavioral of ControlAddressRegister is

begin
    process(reset, car_in)
        variable curr_car : STD_LOGIC_VECTOR(7 downto 0);
        variable temp_curr_car : integer;
        variable temp_inc_car : STD_LOGIC_VECTOR(7 downto 0);
    end process

```

```

begin
    if(reset = '1') then curr_car := x"C0";
    elsif(s_car = '1') then curr_car := car_in;
    elsif(s_car = '0') then
        temp_curr_car := conv_integer(curr_car);
        temp_curr_car := temp_curr_car + conv_integer(1);
        temp_inc_car := conv_std_logic_vector(temp_curr_car, 8);
        curr_car := temp_inc_car;
    end if;
    car_out <= curr_car after 20ns;
end process;
end Behavioral;

```

### 1.3 Control Memory

```

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

entity ControlMemory is
    Port( in_car : in STD_LOGIC_VECTOR(7 downto 0);
          MW, MM, RW, MD, MB, TB, TA, TD, PL, PI, IL, MC : out STD_LOGIC;
          FS_cm : out STD_LOGIC_VECTOR(4 downto 0);
          MS_cm : out STD_LOGIC_VECTOR(2 downto 0);
          NA : out STD_LOGIC_VECTOR(7 downto 0)
        );
end ControlMemory;

architecture Behavioral of ControlMemory is
    --instantiate an array for each given memory allocation
    type mem_array is array(0 to 255) of STD_LOGIC_VECTOR(27 downto 0);

begin
    memory_m : process(in_car)
        variable ControlMemory : mem_array := (
            --Module 0
            x"C020304", --0 start of intermediate value in register
            x"C020304", --1 immediate value in register
            x"C020304", --2 immediate value in register
            x"C020304", --3 immediate value in register
            x"C020304", --4 immediate value in register
            x"C020304", --5 immediate value in register
            x"C020304", --6 immediate value in register
            x"C020304", --7 end of immediate value in register
            x"C020224", --8 ADI -> add the immediate operand
            x"C02000C", --9 LDR -> load to register
            x"C020001", --A STR -> store in register
            x"C020014", --B INC -> increment the register's value by 1
            x"C0200E4", --C NOT -> compliment
            x"C020024", --D ADD -> add values from source and destination
            x"1228002", --E B -> branch unconditionally
            x"0000000", --F

```

```

--Module 1
x"0000000", --0
x"0000000", --1
x"C020000", --2
x"C020024", --3 ADD -> add values from source and destination
x"169A002", --4 BXX -> branch conditionally to area
x"C020024", --5 ADD -> add values from source and destination
x"C020024", --6 ADD -> add values from source and destination
x"C020024", --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

```

```

--Module 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

```

```

--Module 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

```

```

--Module 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
```

```
--Module 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
```

```
--Module 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
```

```
--Module 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
```

```
--Module 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
```

```
--Module 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
```

```
--Module 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

--Module 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

--Module C
x"C10C002", --0 IF fetching
x"0030000", --1 Exit signal
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

--Module 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

--Module 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

--Module 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 := ControlMemory(addr);
MW <= control_out(0);
MM <= control_out(1);
RW <= control_out(2);
MD <= control_out(3);
FS_cm <= 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_cm <= control_out(19 downto 17);
NA <= control_out(27 downto 20);
end process;

end Behavioral;

```

## 1.4 Datapath

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Datapath is
    Port( data_in, pc_in : in STD_LOGIC_VECTOR(15 downto 0);
          control_word : in STD_LOGIC_VECTOR(17 downto 0);
          clk_sig, TD, TA, TB : in STD_LOGIC;
          data_out, addr_out : out STD_LOGIC_VECTOR(15 downto 0);
          status_out : out STD_LOGIC_VECTOR(3 downto 0)
        );
end Datapath;

architecture Behavioral of Datapath is

    component RegFile
        Port( des_d, add_a, add_b : in STD_LOGIC_VECTOR(3 downto 0);
              Clk, load_in : in STD_LOGIC;
              data : in STD_LOGIC_VECTOR(15 downto 0);
              out_data_a, out_data_b : out STD_LOGIC_VECTOR(15 downto 0)
            );
    end component;

    component Mux2to16
        Port( In0, In1 : in STD_LOGIC_VECTOR(15 downto 0);
              s : in STD_LOGIC;
              Z : out STD_LOGIC_VECTOR(15 downto 0)
            );
    end component;

    component ZeroFill
        Port( SB_in : in STD_LOGIC_VECTOR(2 downto 0);

```

```

        zero_fill_out : out STD_LOGIC_VECTOR(15 downto 0)
    );
end component;

component FunctionUnit
    Port( FunctionSelect : in STD_LOGIC_VECTOR(4 downto 0);
          a_in, b_in : in STD_LOGIC_VECTOR(15 downto 0);
          N_fu, Z_fu, V_fu, C_fu : out STD_LOGIC;
          F : out STD_LOGIC_VECTOR(15 downto 0)
    );
end component;

signal mux_b_out, mux_d_out, mux_m_out, reg_file_out_a, reg_file_out_b, func_unit_out,
        zero_fill_out, pc_sig : STD_LOGIC_VECTOR(15 downto 0);
signal dest_d, addr_a, addr_b, status_bits : STD_LOGIC_VECTOR(3 downto 0);

begin

mux_b : Mux2to16 PORT MAP(
    In0 => reg_file_out_b,
    In1 => zero_fill_out,
    s => control_word(8),
    Z => mux_b_out
);

mux_d : Mux2to16 PORT MAP(
    In0 => func_unit_out,
    In1 => data_in,
    s => control_word(2),
    Z => mux_d_out
);

pc_sig <= pc_in;

mux_m : Mux2to16 PORT MAP(
    In0 => reg_file_out_a,
    In1 => pc_sig,
    s => control_word(0),
    Z => mux_m_out
);

dest_d <= TD & control_word(17 downto 15);
addr_a <= TA & control_word(14 downto 12);
addr_b <= TB & control_word(11 downto 9);

zero_fill : ZeroFill PORT MAP(
    SB_in => control_word(11 downto 9),
    zero_fill_out => zero_fill_out
);

reg_file : RegFile PORT MAP(
    des_d => dest_d,
    add_a => addr_a,
    add_b => addr_b,
    Clk => clk_sig,

```

```

    load_in => control_word(1),
    data => mux_d_out,
    out_data_a => reg_file_out_a,
    out_data_b => reg_file_out_b
);

data_out <= mux_b_out;
addr_out <= mux_m_out;

func_unit : FunctionUnit PORT MAP(
    FunctionSelect => control_word(7 downto 3),
    a_in => reg_file_out_a,
    b_in => mux_b_out,
    N_fu => status_bits(1),
    Z_fu => status_bits(0),
    V_fu => status_bits(3),
    C_fu => status_bits(2),
    F => func_unit_out
);

status_out <= status_bits;

end Behavioral;

```

## 1.5 Decoder 4-9 Bit

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Decoder4to9 is
    Port( A0, A1, A2, A3 : in STD_LOGIC;
          Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8 : out STD_LOGIC
    );
end Decoder4to9;

architecture Behavioral of Decoder4to9 is

begin
    Q0 <= ((not A0) and (not A1) and (not A2) and (not A3)) after 1ns; --0000
    Q1 <= ((A0) and (not A1) and (not A2) and (not A3)) after 1ns; --1000
    Q2 <= ((not A0) and (A1) and (not A2) and (not A3)) after 1ns; --0100
    Q3 <= ((A0) and (A1) and (not A2) and (not A3)) after 1ns; --1100
    Q4 <= ((not A0) and (not A1) and (A2) and (not A3)) after 1ns; --0010
    Q5 <= ((A0) and (not A1) and (A2) and (not A3)) after 1ns; --1010
    Q6 <= ((not A0) and (A1) and (A2) and (not A3)) after 1ns; --0110
    Q7 <= ((A0) and (A1) and (A2) and (not A3)) after 1ns; --1110
    Q8 <= ((not A0) and (not A1) and (not A2) and (A3)) after 1ns; --0001

end Behavioral;

```

## 1.6 Extended Programme Counter

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

entity ExtendedProgrammeCounter is
    Port( SR_SB : in STD_LOGIC_VECTOR(5 downto 0);
          ExtendedProgrammeCounter : out STD_LOGIC_VECTOR(15 downto 0)
        );
end ExtendedProgrammeCounter;

architecture Behavioral of ExtendedProgrammeCounter is
    signal extended_signal : STD_LOGIC_VECTOR(15 downto 0);
begin
    extended_signal(5 downto 0) <= SR_SB;
    extended_signal(15 downto 6) <= "0000000000" when SR_SB(5) = '0' else "1111111111";
    ExtendedProgrammeCounter <= extended_signal;

end Behavioral;
```

## 1.7 Full Adder

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

entity FullAdder is
    Port(
        X, Y, Cin : in STD_LOGIC;
        Cout, S : out STD_LOGIC
    );
end FullAdder;

architecture Behavioral of FullAdder is
    signal S0, S1, S2 : STD_LOGIC;
begin
    S0 <= (X xor Y) after 1ns;
    S1 <= (Cin and S0) after 1ns;
    S2 <= (X and Y) after 1ns;
    S <= (S0 xor Cin) after 1ns;
    Cout <= (S1 or S2) after 1ns;

end Behavioral;
```

## 1.8 Function Unit

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

entity FunctionUnit is
    Port(
        FunctionSelect : in STD_LOGIC_VECTOR(4 downto 0); -- 5 input
        a_in, b_in : in STD_LOGIC_VECTOR(15 downto 0);
        N_fu, Z_fu, V_fu, C_fu : out STD_LOGIC;
        F : out STD_LOGIC_VECTOR(15 downto 0)
    );
```

```

end FunctionUnit;

architecture Behavioral of FunctionUnit is

    --2 to 1 mux
    Component Mux2to16
    Port(
        In0, In1 : in STD_LOGIC_VECTOR(15 downto 0);
        s : in STD_LOGIC;
        Z : out STD_LOGIC_VECTOR(15 downto 0)
    );
End Component;

    --shifter
    Component shifter
    Port(
        B : in STD_LOGIC_VECTOR(15 downto 0);
        S : in STD_LOGIC_VECTOR(1 downto 0);
        IL, IR : in STD_LOGIC;
        H : out STD_LOGIC_VECTOR(15 downto 0)
    );
End Component;

    --alu
    Component alu_unit
    Port(
        a_in, b_in : in STD_LOGIC_VECTOR(15 downto 0);
        G_select : in STD_LOGIC_VECTOR(3 downto 0);
        V, C : out STD_LOGIC; -- flags
        G : out STD_LOGIC_VECTOR(15 downto 0)
    );
End Component;

    signal H_out, ALU_out, mux_out : STD_LOGIC_VECTOR(15 downto 0);

begin

    shifter00: shifter PORT MAP(
        B => b_in,
        S => FunctionSelect(3 downto 2),
        IL => '0',
        IR => '0',
        H => H_out
    );

    mux_2_1600: Mux2to16 PORT MAP(
        In0 => ALU_out,
        In1 => H_out,
        s => FunctionSelect(4),
        z => mux_out
    );

    alu: alu_unit PORT MAP(
        a_in => a_in,
        b_in => b_in,
        G_select => FunctionSelect(3 downto 0),
        V => V_fu,
        C => C_fu,

```



```

    G => ALU_out
);

F <= mux_out;
N_fu <= mux_out(15);
Z_fu <= (mux_out(15) or mux_out(14) or mux_out(13) or mux_out(12) or mux_out(11)
        or mux_out(10) or mux_out(9) or mux_out(8) or mux_out(7) or mux_out(6)
        or mux_out(5) or mux_out(4) or mux_out(3) or mux_out(2) or mux_out(1) or
        mux_out(0));

end Behavioral;

```

## 1.9 Instruction Register

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Instructions is
    Port( IR_in : in STD_LOGIC_VECTOR(15 downto 0);
          IL_in : in STD_LOGIC;
          Opcode : out STD_LOGIC_VECTOR(6 downto 0);
          DR_out, SA_out, SB_out : out STD_LOGIC_VECTOR(2 downto 0)
        );
end Instructions;

architecture Behavioral of Instructions is

begin
    Opcode <= IR_in(15 downto 9) after 1ns when IL_in = '1';
    DR_out <= IR_in(8 downto 6) after 1ns when IL_in = '1';
    SA_out <= IR_in(5 downto 3) after 1ns when IL_in = '1';
    SB_out <= IR_in(2 downto 0) after 1ns when IL_in = '1';

end Behavioral;

```

## 1.10 Logic Circuit A-B

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity logic_circuit_a_b is
    Port(
        a_logic_in, b_logic_in : in STD_LOGIC_VECTOR(15 downto 0);
        select_in : in STD_LOGIC_VECTOR(1 downto 0);
        logic_output_a_b : out STD_LOGIC_VECTOR(15 downto 0)
    );
end logic_circuit_a_b;

architecture Behavioral of logic_circuit_a_b is

begin
    logic_output_a_b <= (a_logic_in and b_logic_in) after 1ns when select_in = "00" else
        (a_logic_in or b_logic_in) after 1ns when select_in = "01" else
        (a_logic_in xor b_logic_in) after 1ns when select_in = "10" else
        (not (a_logic_in)) after 1ns;

```

```
end Behavioral;
```

## 1.11 Logic Circuit B

```
library IEEE;
```

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

```
entity logic_circuit_b is
```

```
  Port(
```

```
    B : in STD_LOGIC_VECTOR(15 downto 0);
```

```
    S_in : in STD_LOGIC_VECTOR(1 downto 0);
```

```
    Y_out : out STD_LOGIC_VECTOR(15 downto 0)
```

```
  );
```

```
end logic_circuit_b;
```

```
architecture Behavioral of logic_circuit_b is
```

```
  --mux 2-1 component
```

```
  Component Mux2to1
```

```
  Port(
```

```
    B_i, S0, S1 : in STD_LOGIC;
```

```
    Y_i : out STD_LOGIC
```

```
  );
```

```
End Component;
```

```
begin
```

```
  mux00: Mux2to1 PORT MAP(
```

```
    B_i => B(0),
```

```
    S0 => S_in(0),
```

```
    S1 => S_in(1),
```

```
    Y_i => Y_out(0)
```

```
  );
```

```
  mux01: Mux2to1 PORT MAP(
```

```
    B_i => B(1),
```

```
    S0 => S_in(0),
```

```
    S1 => S_in(1),
```

```
    Y_i => Y_out(1)
```

```
  );
```

```
  mux02: Mux2to1 PORT MAP(
```

```
    B_i => B(2),
```

```
    S0 => S_in(0),
```

```
    S1 => S_in(1),
```

```
    Y_i => Y_out(2)
```

```
  );
```

```
  mux03: Mux2to1 PORT MAP(
```

```
    B_i => B(3),
```

```
    S0 => S_in(0),
```

```
    S1 => S_in(1),
```

```
    Y_i => Y_out(3)
```

```
  );
```

```

mux04: Mux2to1 PORT MAP(
    B_i => B(4),
    S0 => S_in(0),
    S1 => S_in(1),
    Y_i => Y_out(4)
);

mux05: Mux2to1 PORT MAP(
    B_i => B(5),
    S0 => S_in(0),
    S1 => S_in(1),
    Y_i => Y_out(5)
);

mux06: Mux2to1 PORT MAP(
    B_i => B(6),
    S0 => S_in(0),
    S1 => S_in(1),
    Y_i => Y_out(6)
);

mux07: Mux2to1 PORT MAP(
    B_i => B(7),
    S0 => S_in(0),
    S1 => S_in(1),
    Y_i => Y_out(7)
);

mux08: Mux2to1 PORT MAP(
    B_i => B(8),
    S0 => S_in(0),
    S1 => S_in(1),
    Y_i => Y_out(8)
);

mux09: Mux2to1 PORT MAP(
    B_i => B(9),
    S0 => S_in(0),
    S1 => S_in(1),
    Y_i => Y_out(9)
);

mux10: Mux2to1 PORT MAP(
    B_i => B(10),
    S0 => S_in(0),
    S1 => S_in(1),
    Y_i => Y_out(10)
);

end Behavioral;

```

## 1.12 Memory Module

[illegible]

[illegible]

[illegible]

[illegible]

```

x"0000", x"0000", x"0000", x"0000",
x"0000", x"0000", x"0000", x"0000",
x"0000", x"0000", x"0000", x"0000",
x"0000", x"0000", x"0000", x"0000",

--module 1E
x"0000", x"0000", x"0000", x"0000",
x"0000", x"0000", x"0000", x"0000",
x"0000", x"0000", x"0000", x"0000",
x"0000", x"0000", x"0000", x"0000",

--module 1F
x"0000", x"0000", x"0000", x"0000",
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 range 0 to 511;
variable addr_out : STD_LOGIC_VECTOR(15 downto 0);

begin
  addr := conv_integer(address_mem(8 downto 0));
  addr_out := data_mem(addr);
  if mem_write = '1' then
    data_mem(addr) := write_data;
  else read_data <= addr_out;
  end if;
end process;

end Behavioral;

```

## 1.13 Microprogramme Controller

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MicroprogrammeController is
  Port( IR : in STD_LOGIC_VECTOR(15 downto 0);
        status_bits : in STD_LOGIC_VECTOR(3 downto 0);
        reset_mpc : in STD_LOGIC;
        control_word_mpc : out STD_LOGIC_VECTOR(17 downto 0);
        PC_out : out STD_LOGIC_VECTOR(15 downto 0);
        TD_mpc, TA_mpc, TB_mpc, MW_mpc : out STD_LOGIC
        );
end MicroprogrammeController;

architecture Behavioral of MicroprogrammeController is
  component ControlMemory
    Port( in_car : in STD_LOGIC_VECTOR(7 downto 0);
          MW, MM, RW, MD, MB, TB, TA, TD, PL, PI, IL, MC : out STD_LOGIC;
          FS_cm : out STD_LOGIC_VECTOR(4 downto 0);
          MS_cm : out STD_LOGIC_VECTOR(2 downto 0);
          NA : out STD_LOGIC_VECTOR(7 downto 0)
          );
  end component;

```



```

end component;

component Mux2to8
  Port( In0_NA, In1_opcode : in STD_LOGIC_VECTOR(7 downto 0);
        S_mc : in STD_LOGIC;
        out_car : out STD_LOGIC_VECTOR(7 downto 0)
        );
end component;

component Mux8to1
  Port( In_zero, In_one, In_n, In_z, In_c, In_v, In_not_c, In_not_z : in STD_LOGIC;
        S_ms : in STD_LOGIC_VECTOR(2 downto 0);
        out_s_car : out STD_LOGIC
        );
end component;

component ControlAddressRegister
  Port( car_in : in STD_LOGIC_VECTOR(7 downto 0);
        s_car, reset : in STD_LOGIC;
        car_out : out STD_LOGIC_VECTOR(7 downto 0)
        );
end component;

component Instructions
  Port( IR_in : in STD_LOGIC_VECTOR(15 downto 0);
        IL_in : in STD_LOGIC;
        Opcode : out STD_LOGIC_VECTOR(6 downto 0);
        DR_out, SA_out, SB_out : out STD_LOGIC_VECTOR(2 downto 0)
        );
end component;

component ProgrammeCounter
  Port( PC_module_in : in STD_LOGIC_VECTOR(15 downto 0);
        PL_module_in, PI_module_in, reset : in STD_LOGIC;
        PC_module_out : out STD_LOGIC_VECTOR(15 downto 0)
        );
end component;

component ExtendedProgrammeCounter
  Port( SR_SB : in STD_LOGIC_VECTOR(5 downto 0);
        ExtendedProgrammeCounter : out STD_LOGIC_VECTOR(15 downto 0)
        );
end component;

--signalling
signal control_word_sig : STD_LOGIC_VECTOR(17 downto 0);
signal opcode_sig, car_out_sig, out_car_sig, na_sig : STD_LOGIC_VECTOR(7 downto 0);
signal out_s_car_sig, mc_sig, il_sig, pl_sig, pi_sig : STD_LOGIC;
signal ms_cm_sig, sa_sig, sb_sig, dr_sig : STD_LOGIC_VECTOR(2 downto 0);
signal extend_in : STD_LOGIC_VECTOR(5 downto 0);
signal extend_out : STD_LOGIC_VECTOR(15 downto 0);

begin
  control_mem_mpc : ControlMemory PORT MAP(
    in_car => car_out_sig,

```

```

MW => mw_mpc,
MM => control_word_sig(0),
RW => control_word_sig(1),
MD => control_word_sig(2),
MB => control_word_sig(8),
TB => tb_mpc,
TA => ta_mpc,
TD => td_mpc,
PL => pl_sig,
PI => pi_sig,
IL => il_sig,
MC => mc_sig,
FS_cm => control_word_sig(7 downto 3),
MS_cm => ms_cm_sig,
NA => na_sig
);

mux2to8_mpc : mux2to8 PORT MAP(
    In0_NA => na_sig,
    In1_opcode => opcode_sig,
    S_mc => mc_sig,
    out_car => out_car_sig
);

mux8to1_mpc : Mux8to1 PORT MAP(
    In_zero => '0',
    In_one => '1',
    In_z => status_bits(0),
    In_n => status_bits(1),
    In_c => status_bits(2),
    In_v => status_bits(3),
    In_not_z => not status_bits(0),
    In_not_c => not status_bits(2),
    S_ms => ms_cm_sig,
    out_s_car => out_s_car_sig
);

car_mpc : ControlAddressRegister PORT MAP(
    car_in => out_car_sig,
    s_car => out_s_car_sig,
    reset => reset_mpc,
    car_out => car_out_sig
);

instructions_mpc : Instructions PORT MAP(
    IR_in => IR,
    IL_in => il_sig,
    Opcode => opcode_sig(6 downto 0),
    DR_out => dr_sig,
    SA_out => sa_sig,
    SB_out => sb_sig
);

extended_mpc : ExtendedProgrammeCounter PORT MAP(
    SR_SB => extend_in,

```

```

    ExtendedProgrammeCounter => extend_out
);

pc_mpc : ProgrammeCounter PORT MAP(
    PC_module_in => extend_out,
    PL_module_in => pl_sig,
    PI_module_in => pi_sig,
    reset => reset_mpc,
    PC_module_out => pc_out
);

extend_in(5 downto 3) <= dr_sig;
extend_in(2 downto 0) <= sb_sig;
opcode_sig(7) <= '0';

control_word_sig(17 downto 15) <= dr_sig;
control_word_sig(14 downto 12) <= sa_sig;
control_word_sig(11 downto 9) <= sb_sig;
control_word_mpc <= control_word_sig;

end Behavioral;

```

## 1.14 Mux 2-1 Bit

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Mux2to1 is
    Port(
        B_i, S0, S1 : in STD_LOGIC;
        Y_i : out STD_LOGIC
    );
end Mux2to1;

architecture Behavioral of Mux2to1 is
begin
    Y_i <= S0 after 1ns when B_i = '1' else
           S1 after 1ns when B_i = '0' else
           '0' after 1ns;

end Behavioral;

```

## 1.15 Mux 2-8 Bit

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Mux2to8 is
    Port( In0_NA, In1_opcode : in STD_LOGIC_VECTOR(7 downto 0);
          S_mc : in STD_LOGIC;
          out_car : out STD_LOGIC_VECTOR(7 downto 0)
    );
end Mux2to8;

```

```

architecture Behavioral of Mux2to8 is

begin
    out_car <= In0_NA after 1ns when S_mc='0' else
                In1_opcode after 1ns when S_mc='1' else
                x"00" after 20ns;

end Behavioral;

```

## 1.16 Mux 2-16 Bit

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Mux2to16 is
    Port( In0, In1 : in STD_LOGIC_VECTOR(15 downto 0);
          s : in STD_LOGIC;
          Z : out STD_LOGIC_VECTOR(15 downto 0)
    );
end Mux2to16;

architecture Behavioral of Mux2to16 is

begin
    Z <= In0 after 1ns when s='0' else
          In1 after 1ns when s='1' else
          x"0000" after 1ns;

end Behavioral;

```

## 1.17 Mux 3-1 Bit

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux_3_1 is
    Port(
        In0, In1, In2 : in STD_LOGIC;
        S0, S1 : in STD_LOGIC;
        Z : out STD_LOGIC
    );
end mux_3_1;

architecture Behavioral of mux_3_1 is

begin
    Z <= In0 after 1ns when S0 = '0' and S1 = '0' else
          In1 after 1ns when S0 = '0' and S1 = '1' else
          In2 after 1ns when S0 = '1' and S1 = '0' else
          '0' after 1ns;

end Behavioral;

```

## 1.18 Mux 8-1 Bit

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

entity Mux8to1 is
    Port( In_zero, In_one, In_n, In_z, In_c, In_v, In_not_c, In_not_z : in STD_LOGIC;
          S_ms : in STD_LOGIC_VECTOR(2 downto 0);
          out_s_car : out STD_LOGIC
        );
end Mux8to1;

architecture Behavioral of Mux8to1 is

begin
    out_s_car <= In_zero after 1ns when S_ms = "000" else
                 In_one after 1ns when S_ms = "001" else
                 In_c after 1ns when S_ms = "010" else
                 In_v after 1ns when S_ms = "011" else
                 In_z after 1ns when S_ms = "100" else
                 In_n after 1ns when S_ms = "101" else
                 In_not_c after 1ns when S_ms = "110" else
                 In_not_z after 1ns when S_ms = "111";

end Behavioral;
```

## 1.19 Mux 9-16 Bit

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

entity Mux9to16 is
    Port( In0, In1, In2, In3, In4, In5, In6, In7, In8 : in STD_LOGIC_VECTOR(15 downto 0);
          S0, S1, S2, S3 : in STD_LOGIC;
          Z : out STD_LOGIC_VECTOR(15 downto 0)
        );
end Mux9to16;

architecture Behavioral of Mux9to16 is

begin
    Z <= In0 after 5ns when S0='0' and S1='0' and S2='0' and S3='0' else
         In1 after 5ns when S0='1' and S1='0' and S2='0' and S3='0' else
         In2 after 5ns when S0='0' and S1='1' and S2='0' and S3='0' else
         In3 after 5ns when S0='1' and S1='1' and S2='0' and S3='0' else
         In4 after 5ns when S0='0' and S1='0' and S2='1' and S3='0' else
         In5 after 5ns when S0='1' and S1='0' and S2='1' and S3='0' else
         In6 after 5ns when S0='0' and S1='1' and S2='1' and S3='0' else
         In7 after 5ns when S0='1' and S1='1' and S2='1' and S3='0' else
         In8 after 5ns when S0='0' and S1='0' and S2='0' and S3='1' else
         x"0000" after 5ns;

end Behavioral;
```

## 1.20 Programme Counter

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

entity ProgrammeCounter is
    Port( PC_module_in : in STD_LOGIC_VECTOR(15 downto 0);
          PL_module_in, PI_module_in, reset : in STD_LOGIC;
          PC_module_out : out STD_LOGIC_VECTOR(15 downto 0)
        );
end ProgrammeCounter;

architecture Behavioral of ProgrammeCounter is
begin
    process(reset, PL_module_in, PI_module_in)
        variable current_PC : STD_LOGIC_VECTOR(15 downto 0);
        variable temp_curr_PC : integer;
        variable temp_inc_PC : STD_LOGIC_VECTOR(15 downto 0);

    begin
        if(reset = '1') then current_PC := x"0000";
        elsif(PL_module_in = '1') then
            current_PC := current_PC + PC_module_in;
        elsif(PI_module_in = '1') then
            temp_curr_PC := conv_integer(current_PC); -- get current allocation
            temp_curr_PC := temp_curr_PC + conv_integer(1); -- increment
            temp_inc_PC := conv_std_logic_vector(temp_curr_PC, 16); -- cast from int to vector
            current_PC := temp_inc_PC; -- store as current PC
        end if;
        PC_module_out <= current_PC after 2ns;
    end process;
end Behavioral;
```

## 1.21 Project 2 Top Level

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

entity Proj2 is
    Port( Clk, reset : in STD_LOGIC
        );
end Proj2;

architecture Behavioral of Proj2 is

    component Datapath
        Port( data_in, pc_in : in STD_LOGIC_VECTOR(15 downto 0);
              control_word : in STD_LOGIC_VECTOR(17 downto 0);
              clk_sig, TD, TA, TB : in STD_LOGIC;
              data_out, addr_out : out STD_LOGIC_VECTOR(15 downto 0);
              status_out : out STD_LOGIC_VECTOR(3 downto 0)
            );
    end component;
```

```

end component;

component MicroprogrammeController
  Port( IR : in STD_LOGIC_VECTOR(15 downto 0);
        status_bits : in STD_LOGIC_VECTOR(3 downto 0);
        reset_mpc : in STD_LOGIC;
        control_word_mpc : out STD_LOGIC_VECTOR(17 downto 0);
        PC_out : out STD_LOGIC_VECTOR(15 downto 0);
        TD_mpc, TA_mpc, TB_mpc, MW_mpc : out STD_LOGIC
        );
end component;

component Memory
  Port( address_mem : in STD_LOGIC_VECTOR(15 downto 0);
        write_data : in STD_LOGIC_VECTOR(15 downto 0);
        mem_write : in STD_LOGIC;
        read_data : out STD_LOGIC_VECTOR(15 downto 0)
        );
end component;

signal mm_read_data, mpc_pc_out, dp_data_out, dp_address_out : STD_LOGIC_VECTOR(15 downto 0);
signal mpc_control_word : STD_LOGIC_VECTOR(17 downto 0);
signal dp_status_out : STD_LOGIC_VECTOR(3 downto 0);
signal mpc_TD, mpc_TA, mpc_TB, mpc_MW : STD_LOGIC;

begin
  data_path : Datapath PORT MAP(
    data_in => mm_read_data,
    pc_in => mpc_pc_out,
    control_word => mpc_control_word,
    clk_sig => Clk,
    TD => mpc_TD,
    TA => mpc_TA,
    TB => mpc_TB,
    data_out => dp_data_out,
    addr_out => dp_address_out,
    status_out => dp_status_out
  );

  micro_pc : MicroprogrammeController PORT MAP(
    IR => mm_read_data,
    status_bits => dp_status_out,
    reset_mpc => reset,
    control_word_mpc => mpc_control_word,
    PC_out => mpc_pc_out,
    TD_mpc => mpc_TD,
    TA_mpc => mpc_TA,
    TB_mpc => mpc_TB,
    MW_mpc => mpc_MW
  );

  memory_module : Memory PORT MAP(
    address_mem => dp_address_out,
    write_data => dp_data_out,

```

```

        mem_write => mpc_MW,
        read_data => mm_read_data
    );

end Behavioral;

```

## 1.22 Register

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Reg16 is
    Port( D : in STD_LOGIC_VECTOR(15 downto 0);
          load0, load1, Clk : in STD_LOGIC;
          Q : out STD_LOGIC_VECTOR(15 downto 0)
    );
end Reg16;

architecture Behavioral of Reg16 is

begin
    process (Clk)
    begin
        if(rising_edge(Clk)) then
            if((load0 = '1') and (load1 = '1')) then
                Q <= D after 5ns;
            end if;
        end if;
    end process;

end Behavioral;

```

## 1.23 Register File

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity RegFile is
    Port( des_d, add_a, add_b : in STD_LOGIC_VECTOR(3 downto 0);
          Clk, load_in : in STD_LOGIC;
          data : in STD_LOGIC_VECTOR(15 downto 0);
          out_data_a, out_data_b : out STD_LOGIC_VECTOR(15 downto 0)
    );
end RegFile;

architecture Behavioral of RegFile is
    component Reg16
        Port( D : in STD_LOGIC_VECTOR(15 downto 0);
              load0, load1, Clk : in STD_LOGIC;
              Q : out STD_LOGIC_VECTOR(15 downto 0)
        );
    end component;

    component Decoder4to9
        Port( A0, A1, A2, A3 : in STD_LOGIC;

```



```

        Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8 : out STD_LOGIC
    );
end component;

component Mux2to16
    Port( In0, In1 : in STD_LOGIC_VECTOR(15 downto 0);
          s : in STD_LOGIC;
          Z : out STD_LOGIC_VECTOR(15 downto 0)
    );
end component;

component Mux9to16
    Port( In0, In1, In2, In3, In4, In5, In6, In7, In8 : in STD_LOGIC_VECTOR(15 downto 0);
          S0, S1, S2 : in STD_LOGIC;
          Z : out STD_LOGIC_VECTOR(15 downto 0)
    );
end component;

signal load_reg0, load_reg1, load_reg2, load_reg3, load_reg4, load_reg5, load_reg6,
        load_reg7, load_reg8 : STD_LOGIC;
signal reg0_q, reg1_q, reg2_q, reg3_q, reg4_q, reg5_q, reg6_q, reg7_q, reg8_q, out_sig_a,
        out_sig_b : STD_LOGIC_VECTOR(15 downto 0);

begin
    --reg0
    reg0: Reg16 PORT MAP(
        D => data,
        load0 => load_reg0,
        load1 => load_in,
        Clk => Clk,
        Q => reg0_q
    );

    --reg1
    reg1: Reg16 PORT MAP(
        D => data,
        load0 => load_reg1,
        load1 => load_in,
        Clk => Clk,
        Q => reg1_q
    );

    --reg2
    reg2: Reg16 PORT MAP(
        D => data,
        load0 => load_reg2,
        load1 => load_in,
        Clk => Clk,
        Q => reg2_q
    );

    --reg3
    reg3: Reg16 PORT MAP(
        D => data,
        load0 => load_reg3,

```

```

    load1 => load_in,
    Clk => Clk,
    Q => reg3_q
);

--reg4
reg4: Reg16 PORT MAP(
    D => data,
    load0 => load_reg4,
    load1 => load_in,
    Clk => Clk,
    Q => reg4_q
);

--reg5
reg5: Reg16 PORT MAP(
    D => data,
    load0 => load_reg5,
    load1 => load_in,
    Clk => Clk,
    Q => reg5_q
);

--reg6
reg6: Reg16 PORT MAP(
    D => data,
    load0 => load_reg6,
    load1 => load_in,
    Clk => Clk,
    Q => reg6_q
);

--reg7
reg7: Reg16 PORT MAP(
    D => data,
    load0 => load_reg7,
    load1 => load_in,
    Clk => Clk,
    Q => reg7_q
);

--reg8
reg8: Reg16 PORT MAP(
    D => data,
    load0 => load_reg8,
    load1 => load_in,
    Clk => Clk,
    Q => reg8_q
);

DesDecoder4to9 : Decoder4to9 PORT MAP(
    A0 => des_d(0),
    A1 => des_d(1),
    A2 => des_d(2),
    A3 => des_d(3),

```

```

    Q0 => load_reg0,
    Q1 => load_reg1,
    Q2 => load_reg2,
    Q3 => load_reg3,
    Q4 => load_reg4,
    Q5 => load_reg5,
    Q6 => load_reg6,
    Q7 => load_reg7,
    Q8 => load_reg8
);

Mux9to16A : Mux9to16 PORT MAP(
    In0 => reg0_q,
    In1 => reg1_q,
    In2 => reg2_q,
    In3 => reg3_q,
    In4 => reg4_q,
    In5 => reg5_q,
    In6 => reg6_q,
    In7 => reg7_q,
    In8 => reg8_q,
    S0 => add_b(0),
    S1 => add_b(1),
    S2 => add_b(2),
    Z => out_sig_a
);

Mux9to16B : Mux9to16 PORT MAP(
    In0 => reg0_q,
    In1 => reg1_q,
    In2 => reg2_q,
    In3 => reg3_q,
    In4 => reg4_q,
    In5 => reg5_q,
    In6 => reg6_q,
    In7 => reg7_q,
    In8 => reg8_q,
    S0 => add_b(0),
    S1 => add_b(1),
    S2 => add_b(2),
    Z => out_sig_b
);

out_data_a <= out_sig_a;
out_data_b <= out_sig_b;

end Behavioral;

```

## 1.24 Ripple Adder

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

entity RippleAdder is
    Port(
        A, B : in STD_LOGIC_VECTOR(15 downto 0);
        Cin : STD_LOGIC;
        Cout, V_out : out STD_LOGIC;
        G_out : out STD_LOGIC_VECTOR(15 downto 0)
    );

end RippleAdder;

architecture Behavioral of RippleAdder is

    Component FullAdder
        Port(
            X, Y, Cin : in STD_LOGIC;
            Cout, S : out STD_LOGIC
        );
    End Component;

    --signals - 16 carry bits and 1 output
    signal C0, C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12, C13, C14, C15, C_out :
        STD_LOGIC;

begin
    full_adder_00: FullAdder PORT MAP(
        X => A(0),
        Y => B(0),
        Cin => Cin,
        Cout => C0,
        S => G_out(0)
    );

    full_adder_01: FullAdder PORT MAP(
        X => A(1),
        Y => B(1),
        Cin => C0,
        Cout => C1,
        S => G_out(1)
    );

    full_adder_02: FullAdder PORT MAP(
        X => A(2),
        Y => B(2),
        Cin => C1,
        Cout => C2,
        S => G_out(2)
    );

    full_adder_03: FullAdder PORT MAP(
        X => A(3),
```

```

    Y => B(3),
    Cin => C2,
    Cout => C3,
    S => G_out(3)
);

full_adder_04: FullAdder PORT MAP(
    X => A(4),
    Y => B(4),
    Cin => C3,
    Cout => C4,
    S => G_out(4)
);

full_adder_05: FullAdder PORT MAP(
    X => A(5),
    Y => B(5),
    Cin => C4,
    Cout => C5,
    S => G_out(5)
);

full_adder_06: FullAdder PORT MAP(
    X => A(6),
    Y => B(6),
    Cin => C5,
    Cout => C6,
    S => G_out(6)
);

full_adder_07: FullAdder PORT MAP(
    X => A(7),
    Y => B(7),
    Cin => C6,
    Cout => C7,
    S => G_out(7)
);

full_adder_08: FullAdder PORT MAP(
    X => A(8),
    Y => B(8),
    Cin => C7,
    Cout => C8,
    S => G_out(8)
);

full_adder_09: FullAdder PORT MAP(
    X => A(9),
    Y => B(9),
    Cin => C8,
    Cout => C9,
    S => G_out(9)
);

full_adder_10: FullAdder PORT MAP(

```

```

    X => A(10),
    Y => B(10),
    Cin => C9,
    Cout => C10,
    S => G_out(10)
);

full_adder_11: FullAdder PORT MAP(
    X => A(11),
    Y => B(11),
    Cin => C10,
    Cout => C11,
    S => G_out(11)
);

full_adder_12: FullAdder PORT MAP(
    X => A(12),
    Y => B(12),
    Cin => C11,
    Cout => C12,
    S => G_out(12)
);

full_adder_13: FullAdder PORT MAP(
    X => A(13),
    Y => B(13),
    Cin => C12,
    Cout => C13,
    S => G_out(13)
);

full_adder_14: FullAdder PORT MAP(
    X => A(14),
    Y => B(14),
    Cin => C13,
    Cout => C14,
    S => G_out(14)
);

full_adder_15: FullAdder PORT MAP(
    X => A(15),
    Y => B(15),
    Cin => C14,
    Cout => C15,
    S => G_out(15)
);

--carry
Cout <= C_out;
--overflow
V_out <= (C_out xor C15);

end Behavioral;

```

## 1.25 Shifter

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

entity shifter is
  Port(
    B : in STD_LOGIC_VECTOR(15 downto 0);
    S : in STD_LOGIC_VECTOR(1 downto 0);
    IL, IR : in STD_LOGIC;
    H : out STD_LOGIC_VECTOR(15 downto 0)
  );
end shifter;

architecture Behavioral of shifter is

  --2 to 1 mux
  Component mux_3_1
    Port(
      In0, In1, In2, S0, S1 : in STD_LOGIC;
      Z : out STD_LOGIC
    );
  End Component;

begin
  mux00: mux_3_1 PORT MAP(
    In0 => B(0),
    In1 => B(1),
    In2 => IL,
    S0 => S(0),
    S1 => S(1),
    Z => H(0)
  );

  mux01: mux_3_1 PORT MAP(
    In0 => B(1),
    In1 => B(2),
    In2 => B(0),
    S0 => S(0),
    S1 => S(1),
    Z => H(1)
  );

  mux02: mux_3_1 PORT MAP(
    In0 => B(2),
    In1 => B(3),
    In2 => B(1),
    S0 => S(0),
    S1 => S(1),
    Z => H(2)
  );

  mux03: mux_3_1 PORT MAP(
    In0 => B(3),
    In1 => B(4),
```

```

    In2 => B(2),
    S0 => S(0),
    S1 => S(1),
    Z => H(3)
);

mux04: mux_3_1 PORT MAP(
    In0 => B(4),
    In1 => B(5),
    In2 => B(3),
    S0 => S(0),
    S1 => S(1),
    Z => H(4)
);

mux05: mux_3_1 PORT MAP(
    In0 => B(5),
    In1 => B(6),
    In2 => B(4),
    S0 => S(0),
    S1 => S(1),
    Z => H(5)
);

mux06: mux_3_1 PORT MAP(
    In0 => B(6),
    In1 => B(7),
    In2 => B(5),
    S0 => S(0),
    S1 => S(1),
    Z => H(6)
);

mux07: mux_3_1 PORT MAP(
    In0 => B(7),
    In1 => B(8),
    In2 => B(6),
    S0 => S(0),
    S1 => S(1),
    Z => H(7)
);

mux08: mux_3_1 PORT MAP(
    In0 => B(8),
    In1 => B(9),
    In2 => B(7),
    S0 => S(0),
    S1 => S(1),
    Z => H(8)
);

mux09: mux_3_1 PORT MAP(
    In0 => B(9),
    In1 => B(10),
    In2 => B(8),

```



```

    S0 => S(0),
    S1 => S(1),
    Z => H(9)
);

mux10: mux_3_1 PORT MAP(
    In0 => B(10),
    In1 => B(11),
    In2 => B(9),
    S0 => S(0),
    S1 => S(1),
    Z => H(10)
);

mux11: mux_3_1 PORT MAP(
    In0 => B(11),
    In1 => B(12),
    In2 => B(10),
    S0 => S(0),
    S1 => S(1),
    Z => H(11)
);

mux12: mux_3_1 PORT MAP(
    In0 => B(12),
    In1 => B(13),
    In2 => B(11),
    S0 => S(0),
    S1 => S(1),
    Z => H(12)
);

mux13: mux_3_1 PORT MAP(
    In0 => B(13),
    In1 => B(14),
    In2 => B(12),
    S0 => S(0),
    S1 => S(1),
    Z => H(13)
);

mux14: mux_3_1 PORT MAP(
    In0 => B(14),
    In1 => B(15),
    In2 => B(13),
    S0 => S(0),
    S1 => S(1),
    Z => H(14)
);

mux15: mux_3_1 PORT MAP(
    In0 => B(15),
    In1 => IR,
    In2 => B(14),
    S0 => S(0),

```

```

        S1 => S(1),
        Z => H(15)
    );

```

```

end Behavioral;

```

## 1.26 Zero Fill

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ZeroFill is
    Port( SB_in : in STD_LOGIC_VECTOR(2 downto 0);
          zero_fill_out : out STD_LOGIC_VECTOR(15 downto 0)
        );
end ZeroFill;

architecture Behavioral of ZeroFill is
    signal ZeroFill : STD_LOGIC_VECTOR(15 downto 0);
begin
    ZeroFill(2 downto 0) <= SB_in;
    ZeroFill(15 downto 3) <= "00000000000000";
end Behavioral;

```

## 2 Component Test Benches

### 2.1 Control Address Register

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

ENTITY ControlAddressRegister_TB IS
END ControlAddressRegister_TB;

ARCHITECTURE behavior OF ControlAddressRegister_TB IS

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

    COMPONENT ControlAddressRegister
    PORT(
        car_in : IN std_logic_vector(7 downto 0);
        s_car : IN std_logic;
        reset : IN std_logic;
        car_out : OUT std_logic_vector(7 downto 0)
    );
    END COMPONENT;

    --Inputs
    signal car_in : std_logic_vector(7 downto 0) := (others => '0');
    signal s_car : std_logic := '0';
    signal reset : std_logic := '0';

    --Outputs
    signal car_out : std_logic_vector(7 downto 0);

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: ControlAddressRegister PORT MAP (
        car_in => car_in,
        s_car => s_car,
        reset => reset,
        car_out => car_out
    );

    -- Stimulus process
    stim_proc: process
    begin
        wait for 5ns;
        reset <= '1';

        wait for 30ns;
        reset <= '0';

        wait for 30ns;
        car_in <= x"01";

        wait for 30ns;
```

```

    car_in <= x"A1";
    s_car <= '1';

    wait;
end process;

END;

```

## 2.2 Control Memory

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ControlMemory_TB IS
END ControlMemory_TB;

ARCHITECTURE behavior OF ControlMemory_TB IS

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

    COMPONENT ControlMemory
    PORT(
        in_car : IN std_logic_vector(7 downto 0);
        MW : OUT std_logic;
        MM : OUT std_logic;
        RW : OUT std_logic;
        MD : OUT std_logic;
        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;
        FS_cm : OUT std_logic_vector(4 downto 0);
        MS_cm : OUT std_logic_vector(2 downto 0);
        NA : OUT 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 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 FS_cm : std_logic_vector(4 downto 0);
signal MS_cm : std_logic_vector(2 downto 0);
signal NA : std_logic_vector(7 downto 0);

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: ControlMemory PORT MAP (
        in_car => in_car,
        MW => MW,
        MM => MM,
        RW => RW,
        MD => MD,
        MB => MB,
        TB => TB,
        TA => TA,
        TD => TD,
        PL => PL,
        PI => PI,
        IL => IL,
        MC => MC,
        FS_cm => FS_cm,
        MS_cm => MS_cm,
        NA => NA
    );

    -- Stimulus process
    stim_proc: process
    begin
        wait for 10ns;
        in_car <= x"00";

        wait for 10ns;
        in_car <= x"08";

        wait for 10ns;
        in_car <= x"0D";

        wait;
    end process;

END;

```

## 2.3 Decoder 4-9 bit

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Decoder4to9_TB IS
END Decoder4to9_TB;

ARCHITECTURE behavior OF Decoder4to9_TB IS

```

```

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

COMPONENT Decoder4to9
PORT(
    A0 : IN std_logic;
    A1 : IN std_logic;
    A2 : IN std_logic;
    A3 : IN std_logic;
    Q0 : OUT std_logic;
    Q1 : OUT std_logic;
    Q2 : OUT std_logic;
    Q3 : OUT std_logic;
    Q4 : OUT std_logic;
    Q5 : OUT std_logic;
    Q6 : OUT std_logic;
    Q7 : OUT std_logic;
    Q8 : OUT std_logic
);
END COMPONENT;

--Inputs
signal A0 : std_logic := '0';
signal A1 : std_logic := '0';
signal A2 : std_logic := '0';
signal A3 : std_logic := '0';

--Outputs
signal Q0 : std_logic;
signal Q1 : std_logic;
signal Q2 : std_logic;
signal Q3 : std_logic;
signal Q4 : std_logic;
signal Q5 : std_logic;
signal Q6 : std_logic;
signal Q7 : std_logic;
signal Q8 : std_logic;

BEGIN

-- Instantiate the Unit Under Test (UUT)
 uut: Decoder4to9 PORT MAP (
    A0 => A0,
    A1 => A1,
    A2 => A2,
    A3 => A3,
    Q0 => Q0,
    Q1 => Q1,
    Q2 => Q2,
    Q3 => Q3,
    Q4 => Q4,
    Q5 => Q5,
    Q6 => Q6,
    Q7 => Q7,

```

```

        Q8 => Q8
    );

-- Stimulus process
stim_proc: process
begin
    wait for 5ns;
    A0 <= '0';
    A1 <= '0';
    A2 <= '0';
    A3 <= '0';

    wait for 5ns;
    A0 <= '1';
    A1 <= '0';
    A2 <= '0';
    A3 <= '0';

    wait for 5ns;
    A0 <= '0';
    A1 <= '1';
    A2 <= '0';
    A3 <= '0';

    wait for 5ns;
    A0 <= '1';
    A1 <= '1';
    A2 <= '0';
    A3 <= '0';

    wait for 5ns;
    A0 <= '0';
    A1 <= '0';
    A2 <= '1';
    A3 <= '0';

    wait for 5ns;
    A0 <= '1';
    A1 <= '0';
    A2 <= '1';
    A3 <= '0';

    wait for 5ns;
    A0 <= '0';
    A1 <= '1';
    A2 <= '1';
    A3 <= '0';

    wait for 5ns;
    A0 <= '1';
    A1 <= '1';
    A2 <= '1';
    A3 <= '0';

    wait for 5ns;

```

```

A3 <= '1';

wait for 5ns;
A0 <= '0';
A1 <= '0';
A2 <= '0';

wait for 5ns;
A0 <= '1';
A1 <= '0';
A2 <= '0';

wait for 5ns;
A0 <= '0';
A1 <= '1';
A2 <= '0';
end process;

END;

```

## 2.4 Extended Programme Counter

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ExtendedProgrammeCounter_TB IS
END ExtendedProgrammeCounter_TB;

ARCHITECTURE behavior OF ExtendedProgrammeCounter_TB IS

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

    COMPONENT ExtendedProgrammeCounter
    PORT(
        SR_SB : IN std_logic_vector(5 downto 0);
        ExtendedProgrammeCounter : OUT std_logic_vector(15 downto 0)
    );
    END COMPONENT;

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

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

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: ExtendedProgrammeCounter PORT MAP (
        SR_SB => SR_SB,
        ExtendedProgrammeCounter => ExtendedProgrammeCounter
    );

    -- Stimulus process

```



```

stim_proc: process
begin
    wait for 10ns;
    SR_SB <= "010110";

    wait for 10ns;
    SR_SB <= "110110";

    wait;
end process;

END;

```

## 2.5 Instruction Register

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Instructions_TB IS
END Instructions_TB;

ARCHITECTURE behavior OF Instructions_TB IS

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

    COMPONENT Instructions
    PORT(
        IR_in : IN std_logic_vector(15 downto 0);
        IL_in : IN std_logic;
        Opcode : OUT std_logic_vector(6 downto 0);
        DR_out : OUT std_logic_vector(2 downto 0);
        SA_out : OUT std_logic_vector(2 downto 0);
        SB_out : OUT std_logic_vector(2 downto 0)
    );
    END COMPONENT;

    --Inputs
    signal IR_in : std_logic_vector(15 downto 0) := (others => '0');
    signal IL_in : std_logic := '0';

    --Outputs
    signal Opcode : std_logic_vector(6 downto 0);
    signal DR_out : std_logic_vector(2 downto 0);
    signal SA_out : std_logic_vector(2 downto 0);
    signal SB_out : std_logic_vector(2 downto 0);

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: Instructions PORT MAP (
        IR_in => IR_in,
        IL_in => IL_in,
        Opcode => Opcode,
        DR_out => DR_out,

```

```

        SA_out => SA_out,
        SB_out => SB_out
    );

-- Stimulus process
stim_proc: process
begin
    wait for 10ns;
    IR_in <= "1111111000001010";

    wait for 5ns;
    IL_in <= '1';

    wait for 10ns;
    IR_in <= "0000000110110000";
    IL_in <= '0';

    wait for 5ns;
    IL_in <= '1';

    wait;
end process;

END;

```

## 2.6 Memory Module

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Memory_TB IS
END Memory_TB;

ARCHITECTURE behavior OF Memory_TB IS

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

    COMPONENT Memory
    PORT(
        address_mem : IN std_logic_vector(15 downto 0);
        write_data : IN std_logic_vector(15 downto 0);
        mem_write : IN std_logic;
        read_data : OUT std_logic_vector(15 downto 0)
    );
    END COMPONENT;

    --Inputs
    signal address_mem : std_logic_vector(15 downto 0) := (others => '0');
    signal write_data : std_logic_vector(15 downto 0) := (others => '0');
    signal mem_write : std_logic := '0';

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

```

```
BEGIN
```

```
-- Instantiate the Unit Under Test (UUT)
```

```
uut: Memory PORT MAP (  
    address_mem => address_mem,  
    write_data => write_data,  
    mem_write => mem_write,  
    read_data => read_data  
);
```

```
-- Stimulus process
```

```
stim_proc: process  
begin  
    wait for 10ns;  
    address_mem <= x"0000";  
  
    wait for 10ns;  
    address_mem <= x"0001";  
  
    wait for 10ns;  
    address_mem <= x"0006";  
  
    wait for 10ns;  
    address_mem <= x"0007";  
  
    wait;  
end process;
```

```
END;
```

## 2.7 Mux 2-8 Bit

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

```
ENTITY Mux2to8_TB IS  
END Mux2to8_TB;
```

```
ARCHITECTURE behavior OF Mux2to8_TB IS
```

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

```
COMPONENT Mux2to8  
PORT(  
    In0_NA : IN std_logic_vector(7 downto 0);  
    In1_opcode : IN std_logic_vector(7 downto 0);  
    S_mc : IN std_logic;  
    out_car : OUT std_logic_vector(7 downto 0)  
);  
END COMPONENT;
```

```
--Inputs
```

```
signal In0_NA : std_logic_vector(7 downto 0) := (others => '0');  
signal In1_opcode : std_logic_vector(7 downto 0) := (others => '0');
```

```

signal S_mc : std_logic := '0';

--Outputs
signal out_car : std_logic_vector(7 downto 0);

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: Mux2to8 PORT MAP (
        In0_NA => In0_NA,
        In1_opcode => In1_opcode,
        S_mc => S_mc,
        out_car => out_car
    );

    -- Stimulus process
    stim_proc: process
    begin
        In0_NA <= x"FF";
        In1_opcode <= x"AA";

        wait for 20ns;
        S_mc <= '1';

        wait;
    end process;

END;

```

## 2.8 Mux 8-1 Bit

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Mux8to1_TB IS
END Mux8to1_TB;

ARCHITECTURE behavior OF Mux8to1_TB IS

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

    COMPONENT Mux8to1
    PORT(
        In_zero : IN std_logic;
        In_one : IN std_logic;
        In_n : IN std_logic;
        In_z : IN std_logic;
        In_c : IN std_logic;
        In_v : IN std_logic;
        In_not_c : IN std_logic;
        In_not_z : IN std_logic;
        S_ms : IN std_logic_vector(2 downto 0);
        out_s_car : OUT std_logic
    );
END COMPONENT;

```

```

--Inputs
signal In_zero : std_logic := '0';
signal In_one : std_logic := '0';
signal In_n : std_logic := '0';
signal In_z : std_logic := '0';
signal In_c : std_logic := '0';
signal In_v : std_logic := '0';
signal In_not_c : std_logic := '0';
signal In_not_z : std_logic := '0';
signal S_ms : std_logic_vector(2 downto 0) := (others => '0');

--Outputs
signal out_s_car : std_logic;

BEGIN

-- Instantiate the Unit Under Test (UUT)
 uut: Mux8to1 PORT MAP (
    In_zero => In_zero,
    In_one => In_one,
    In_n => In_n,
    In_z => In_z,
    In_c => In_c,
    In_v => In_v,
    In_not_c => In_not_c,
    In_not_z => In_not_z,
    S_ms => S_ms,
    out_s_car => out_s_car
 );

-- Stimulus process
stim_proc: process
begin
    wait for 5ns;
    In_one <= '1';
    In_c <= '1';
    In_v <= '1';
    In_not_z <= '1';

    wait for 5ns;
    S_ms <= "001";

    wait for 5ns;
    S_ms <= "010";

    wait for 5ns;
    S_ms <= "011";

    wait for 5ns;
    S_ms <= "100";

    wait for 5ns;
    S_ms <= "101";

```

```

    wait for 5ns;
    S_ms <= "110";

    wait for 5ns;
    S_ms <= "111";

    wait for 5ns;

    wait;
end process;

END;

```

## 2.9 Mux 9-16 Bit

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

```

```

ENTITY Mux9to16_TB IS
END Mux9to16_TB;

```

```

ARCHITECTURE behavior OF Mux9to16_TB IS

```

```

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

```

```

COMPONENT Mux9to16
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);
    S0 : IN std_logic;
    S1 : IN std_logic;
    S2 : IN std_logic;
    S3 : IN std_logic;
    Z : 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 In2 : std_logic_vector(15 downto 0) := (others => '0');
signal In3 : std_logic_vector(15 downto 0) := (others => '0');
signal In4 : std_logic_vector(15 downto 0) := (others => '0');
signal In5 : std_logic_vector(15 downto 0) := (others => '0');
signal In6 : std_logic_vector(15 downto 0) := (others => '0');
signal In7 : std_logic_vector(15 downto 0) := (others => '0');

```

```

signal In8 : std_logic_vector(15 downto 0) := (others => '0');
signal S0 : std_logic := '0';
signal S1 : std_logic := '0';
signal S2 : std_logic := '0';
signal S3 : std_logic := '0';

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

BEGIN

-- Instantiate the Unit Under Test (UUT)
 uut: Mux9to16 PORT MAP (
    In0 => In0,
    In1 => In1,
    In2 => In2,
    In3 => In3,
    In4 => In4,
    In5 => In5,
    In6 => In6,
    In7 => In7,
    In8 => In8,
    S0 => S0,
    S1 => S1,
    S2 => S2,
    S3 => S3,
    Z => Z
  );

-- Stimulus process
stim_proc: process
begin
  In0 <= x"FFFF";
  In1 <= x"EEEE";
  In2 <= x"DDDD";
  In3 <= x"CCCC";
  In4 <= x"BBBB";
  In5 <= x"AAAA";
  In6 <= x"9999";
  In7 <= x"8888";
  In8 <= x"7777";

  wait for 10ns;
  S0 <= '1';
  S1 <= '0';
  S2 <= '0';

  wait for 10ns;
  S0 <= '0';
  S1 <= '1';
  S2 <= '0';

  wait for 10ns;
  S0 <= '1';
  S1 <= '1';

```

```

S2 <= '0';

wait for 10ns;
S0 <= '0';
S1 <= '0';
S2 <= '1';

wait for 10ns;
S0 <= '1';
S1 <= '0';
S2 <= '1';

wait for 10ns;
S0 <= '0';
S1 <= '1';
S2 <= '1';

wait for 10ns;
S0 <= '1';
S1 <= '1';
S2 <= '1';

wait for 10ns;
S3 <= '1';

wait for 10ns;
S0 <= '1';
S1 <= '0';
S2 <= '0';

wait for 10ns;
S0 <= '0';
S1 <= '1';
S2 <= '0';

wait for 10ns;
S0 <= '1';
S1 <= '1';
S2 <= '0';

end process;

END;

```

## 2.10 Programme Counter

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ProgrammeCounter_TB IS
END ProgrammeCounter_TB;

ARCHITECTURE behavior OF ProgrammeCounter_TB IS

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

```



```

COMPONENT ProgrammeCounter
PORT(
    PC_module_in : IN std_logic_vector(15 downto 0);
    PL_module_in : IN std_logic;
    PI_module_in : IN std_logic;
    reset : IN std_logic;
    PC_module_out : OUT std_logic_vector(15 downto 0)
);
END COMPONENT;

--Inputs
signal PC_module_in : std_logic_vector(15 downto 0) := (others => '0');
signal PL_module_in : std_logic := '0';
signal PI_module_in : std_logic := '0';
signal reset : std_logic := '0';

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

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: ProgrammeCounter PORT MAP (
        PC_module_in => PC_module_in,
        PL_module_in => PL_module_in,
        PI_module_in => PI_module_in,
        reset => reset,
        PC_module_out => PC_module_out
    );

    -- Stimulus process
    stim_proc: process
    begin
        wait for 5ns;
        reset <= '1';
        PC_module_in <= x"0000";

        wait for 5ns;
        reset <= '0';

        wait for 5ns;
        PI_module_in <= '1';
        PC_module_in <= x"0002";

        wait for 20ns;
        PI_module_in <= '0';
        PL_module_in <= '1';
        PC_module_in <= x"000F";

        wait;
    end process;

END;

```

## 2.11 Project 2 Top Level

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

ENTITY Proj2_TB IS
END Proj2_TB;

ARCHITECTURE behavior OF Proj2_TB IS

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

    COMPONENT Proj2
    PORT(
        Clk : IN std_logic;
        reset : IN std_logic
    );
    END COMPONENT;

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

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

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: Proj2 PORT MAP (
        Clk => Clk,
        reset => reset
    );

    -- 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
        reset <= '1';
        wait for 40ns;

        reset <= '0';
        wait;
    end process;
```

```
END;
```

## 2.12 Zero Fill

```
LIBRARY ieee;
```

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

```
ENTITY ZeroFill_TB IS
```

```
END ZeroFill_TB;
```

```
ARCHITECTURE behavior OF ZeroFill_TB IS
```

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

```
COMPONENT ZeroFill
```

```
PORT(
```

```
    SB_in : IN std_logic_vector(2 downto 0);
```

```
    zero_fill_out : OUT std_logic_vector(15 downto 0)
```

```
);
```

```
END COMPONENT;
```

```
--Inputs
```

```
signal SB_in : std_logic_vector(2 downto 0) := (others => '0');
```

```
--Outputs
```

```
signal zero_fill_out : std_logic_vector(15 downto 0);
```

```
BEGIN
```

```
-- Instantiate the Unit Under Test (UUT)
```

```
uut: ZeroFill PORT MAP (
```

```
    SB_in => SB_in,
```

```
    zero_fill_out => zero_fill_out
```

```
);
```

```
-- Stimulus process
```

```
stim_proc: process
```

```
begin
```

```
    wait for 10ns;
```

```
    SB_in <= "110";
```

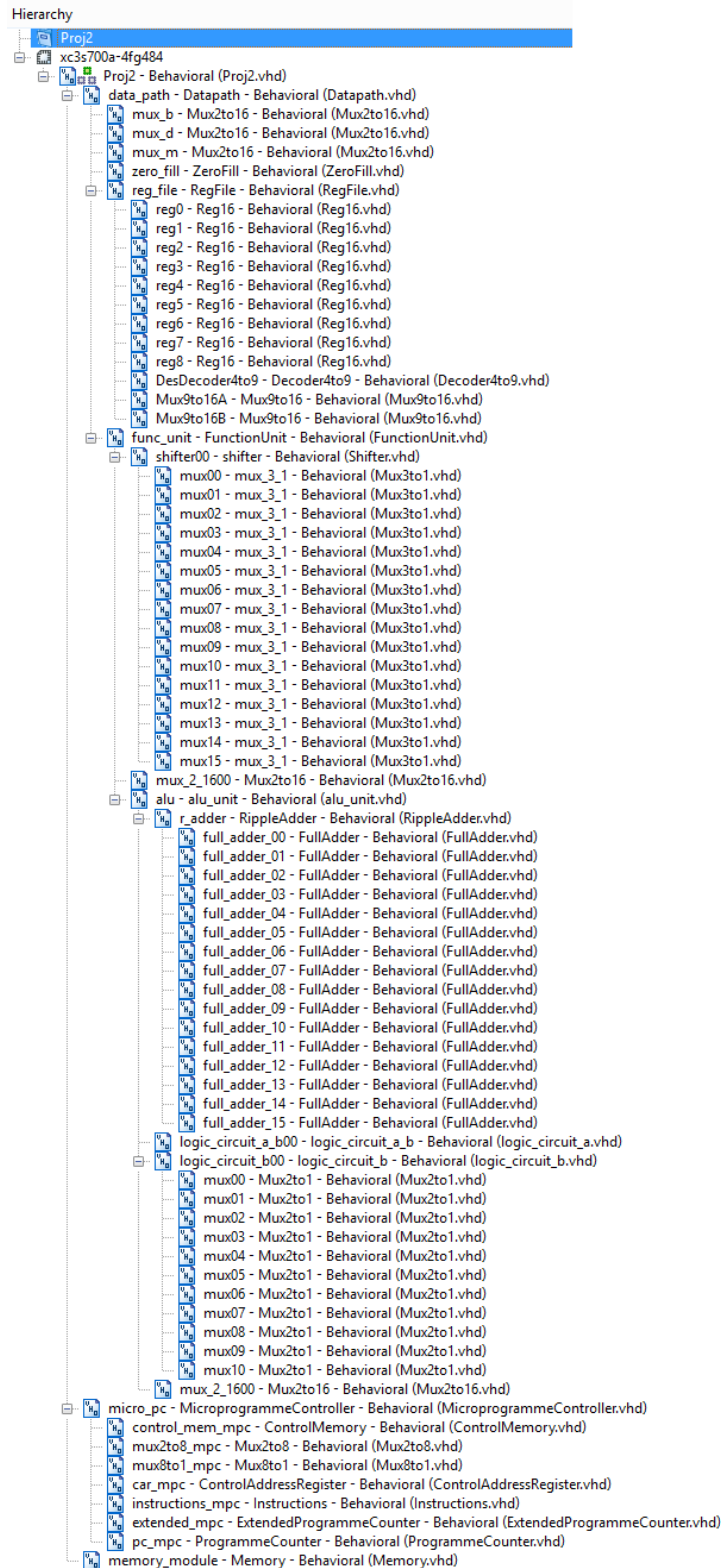
```
    wait;
```

```
end process;
```

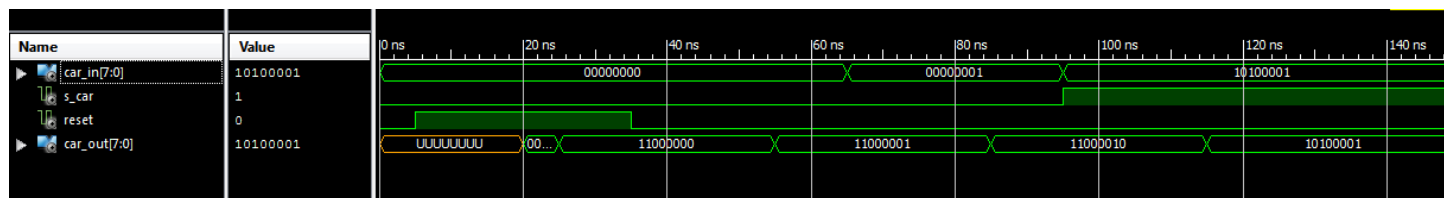
```
END;
```

### 3 Results of Test Benches

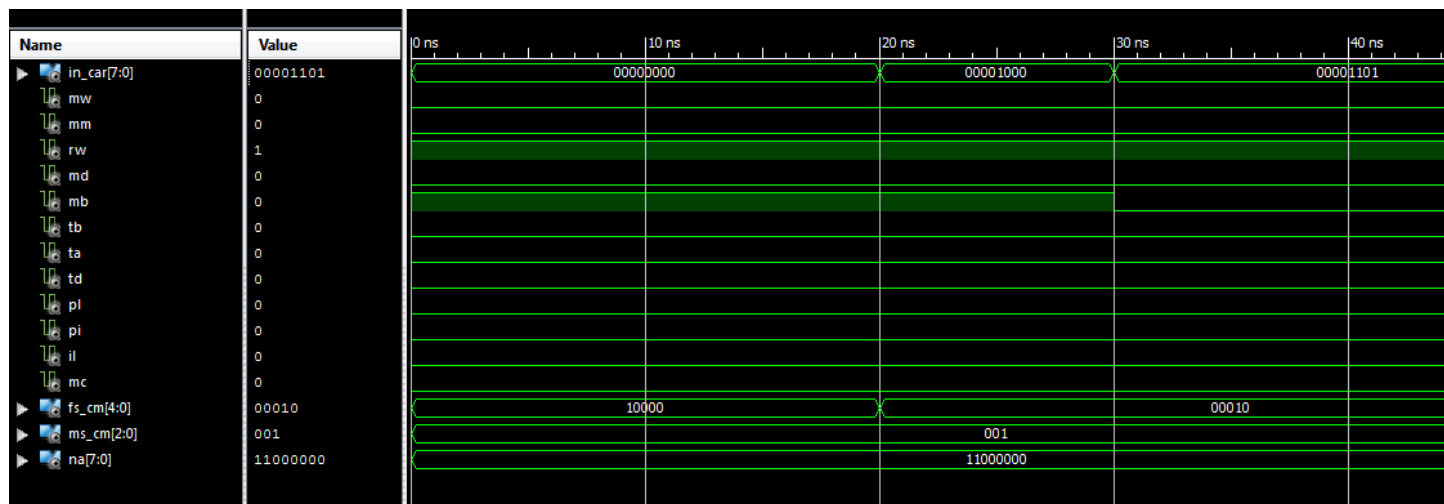
The testbenches show the results of the operands performed by each component within the arithmetic state machine, where the top-level test bench clearly shows the contents and the resulting changes in the registers over time. Each change and resulting operation in time is reflected by loading, add increment (ADI), load to register (LDR), store to register (STR), increment (INC), inverse (NOT), add (ADD), unconditional branch (BCH), and conditional branch (BXX). These instructions pertain to modifying the contents of the registers in memory at given time intervals, and are more easily seen in the control memory and memory module components.



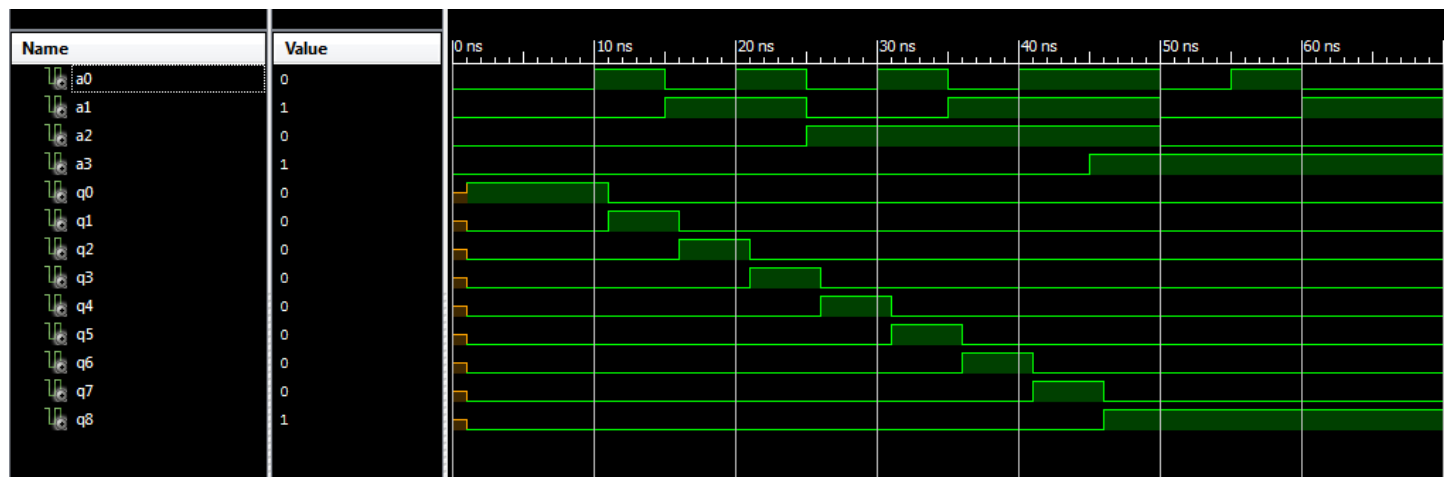
### 3.1 Control Address Register



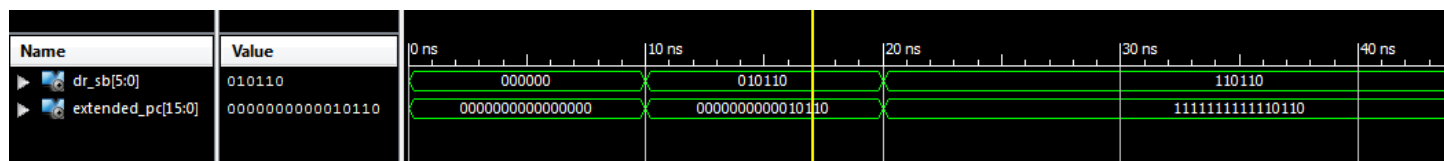
### 3.2 Control Memory



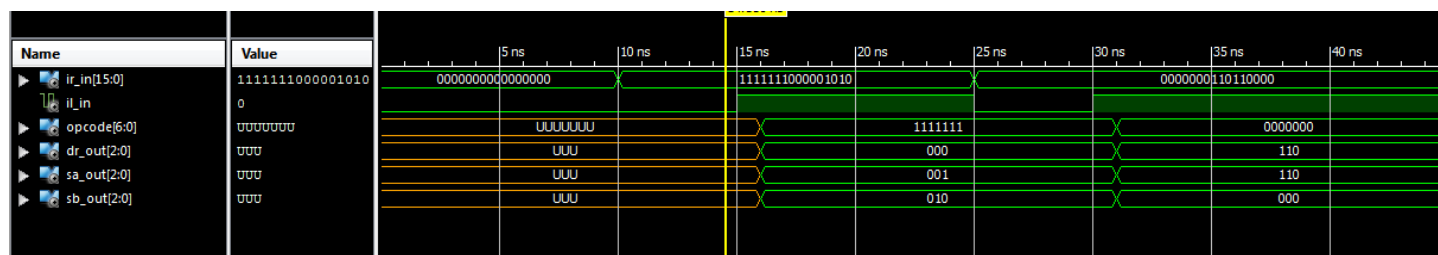
### 3.3 Decoder 4-9 bit



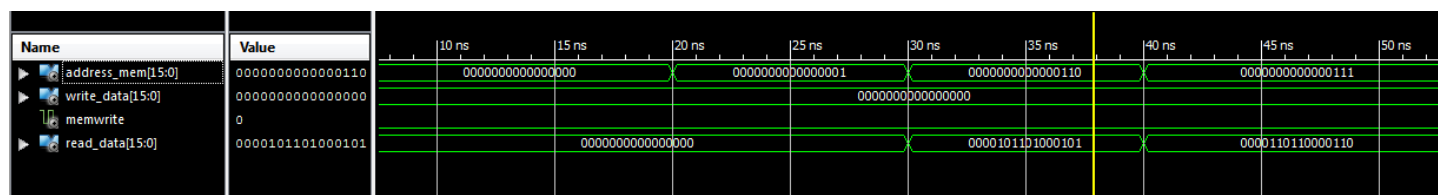
### 3.4 Extended Programme Counter



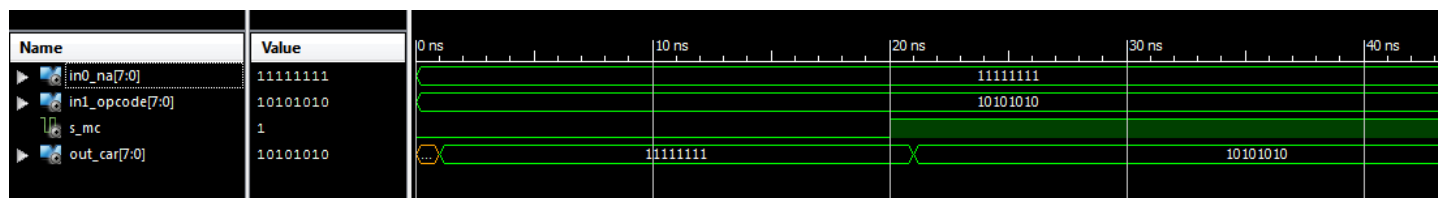
## 3.5 Instruction Register



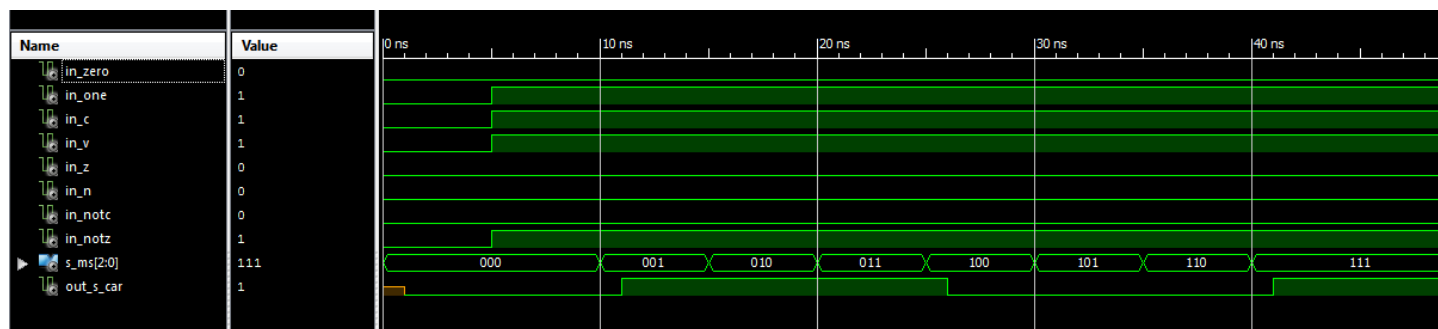
## 3.6 Memory Module



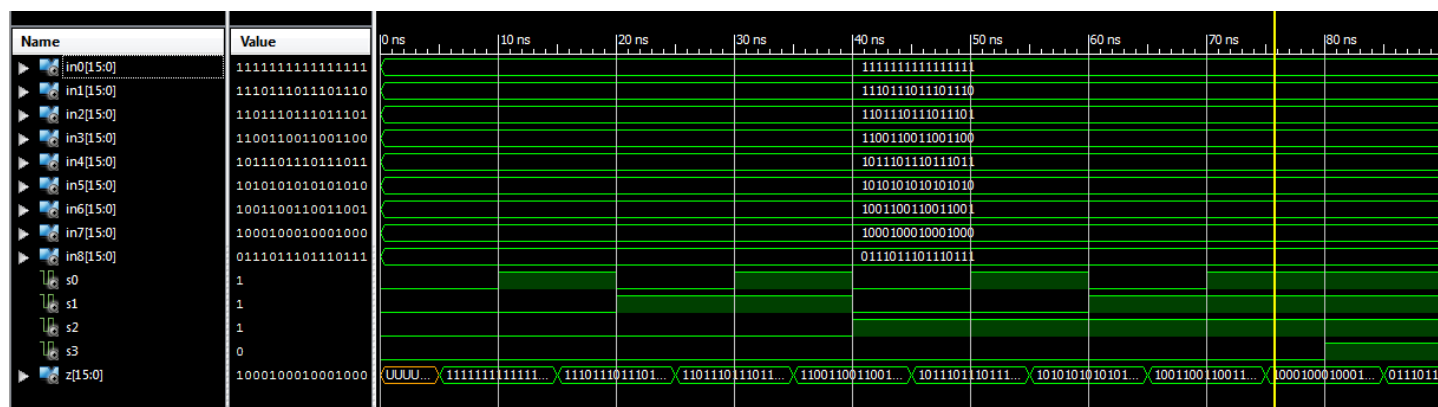
## 3.7 Mux 2-1 Bit



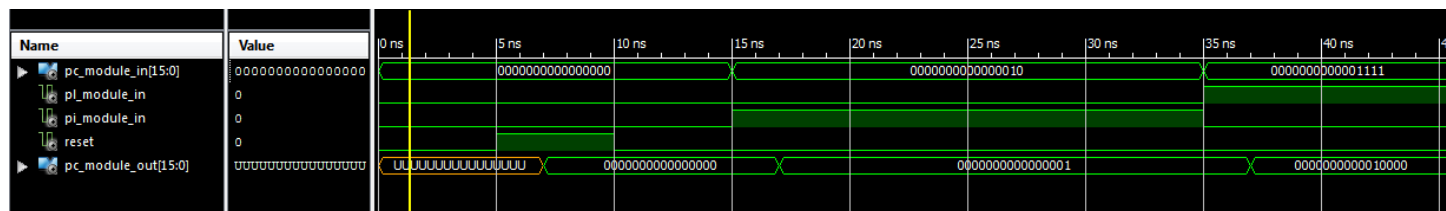
## 3.8 Mux 8-1



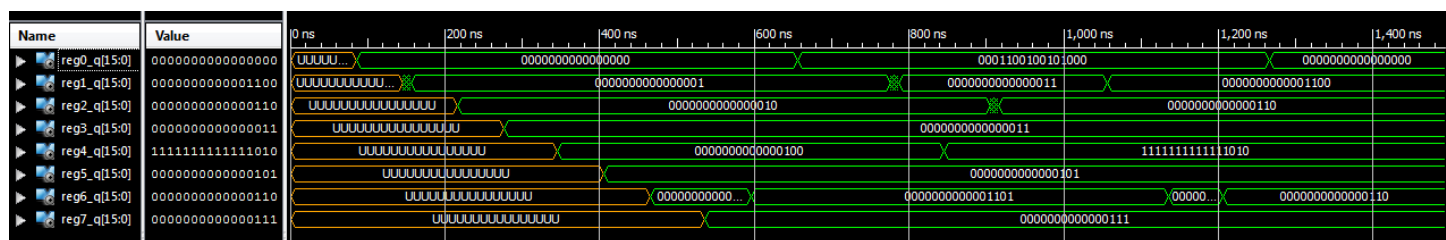
## 3.9 Mux 9-16 Bit



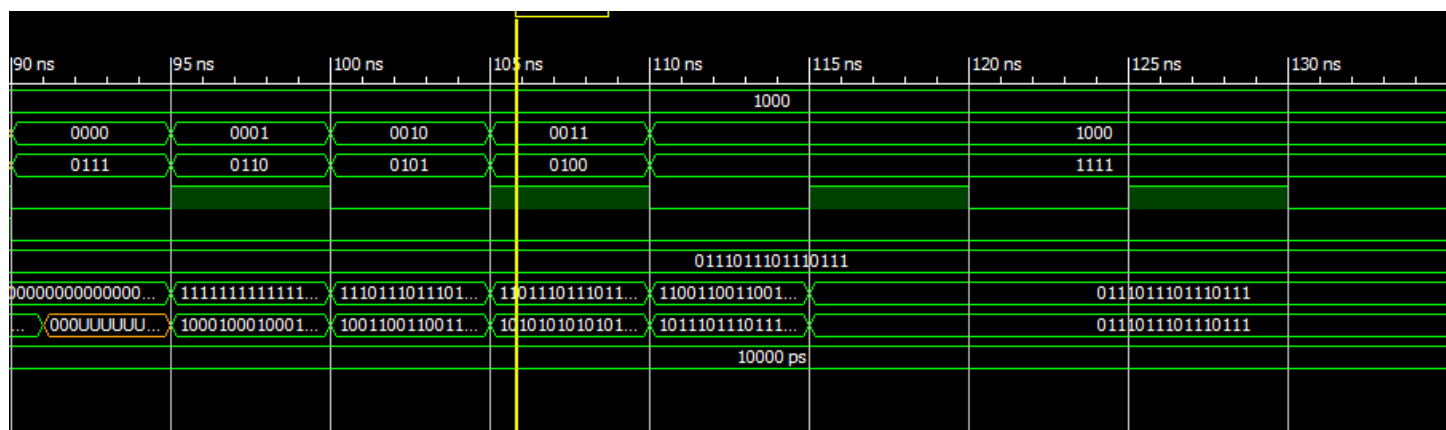
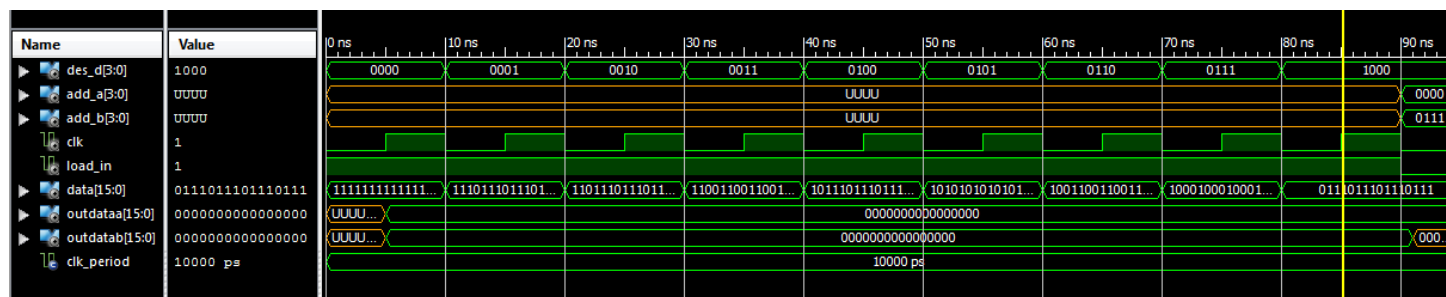
### 3.10 Programme Counter



### 3.11 Project 2 Top Level



### 3.12 Register File



### 3.13 Zero Fill

