Active-HDL PDF Export
finalProject workspace

Contents

1 Table of Contents
2 finalproject 1
2.1 simplecpu.vhd 1
2.2 multiplying.awc 11
2.3 waves 13
2.3.1 add.awc 14
2.4 TestBench 16
2.4.1 simplecpu_TB.vhd 16
2.4.2 simplecpu_TB_runtest.do 16

2 finalproject

2.1 simplecpu.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.unsigned;
use ieee.numeric_std.all;
USE ieee.std_logic_1164.ALL;
use IEEE.Numeric_Std.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity simplecpu is
    port(
    reset: in std_logic;
    clk:in std_logic;
    output:OUT std_logic_vector(6 downto 0));


end simplecpu;

--}} End of automatically maintained section

architecture simplecpu of simplecpu is
type states is (state0,state1,state2,state3,state4,state5,state6,state7,state8
,state9);
signal nx_State:states;
signal pr_state:states:=state1;
type memory is array ( 12 downto 0) of std_logic_vector( 6  downto 0);
constant my_Rom: memory :=(
--load r0
0 =>"0010000",
--value 0
1 =>"0000000",
--load r1
2 =>"0010100",
--value 7
3 =>"0000111",
-- load r2
4 =>"0011000",
-- value 8
5 =>"0001000",
-- load r3
6 =>"0011100",
--  value 1
7 =>"0000001",
--L0: add r0,r2
8 =>"0100010",
--sub r1,r3
9 =>"0110111",
-- jnz r1 ,L0
10=>"1000100",
--L0=8
11=>"0001000",
--HLT
12=>"1111111");
```

```
signal rin_r0,rout_r0:std_logic_vector(6 downto 0):="0000000";
signal rin_r1,rout_r1:std_logic_vector(6 downto 0):="0000000";
signal rin_r2,rout_r2:std_logic_vector(6 downto 0):="0000000" ;
signal rin_r3,rout_r3:std_logic_vector(6 downto 0):="0000000"  ;
signal pc:std_logic_vector(6 downto 0):="0000000";
signal rin_ir,rout_ir:std_logic_vector(6 downto 0):="0000000";
signal LD0,LD1,LD2,LD3:std_logic:='0';
signal zr0,zr1,zr2,zr3:std_logic:='0';
signal LD_PC,LD_IR:std_logic:='0';
signal inc_pc,clr_pc:std_logic:='0';

signal s0,s1,reg_select_mux0,reg_select_mux1:std_logic_vector(1 downto 0);
signal ir_select:std_logic_vector(2 downto 0);
signal mux0,mux1 :std_logic_vector(6 downto 0):="0000000";
signal aluresult: std_logic_vector(6 downto 0):="0000000";
signal busline :std_logic_vector(6 downto 0):="0000000";
signal MData: std_logic_vector(6 downto 0):="0000000";
signal bus_sel:std_logic:='0';
signal cmd:std_logic_vector(1 downto 0):="00";
signal data :std_logic_vector(6 downto 0):="0000000";


begin
----------------------R0--------------------
process(clk,LD0,rin_r0,busline,zr0)
begin
        rin_r0<=busline;
    if (rising_edge(clk)) then
        if (LD0='1') then
            rout_r0<=rin_r0;
        else
            rout_r0<=rout_r0;
        end if;
    end if;
        if LD0='1' then
            if rin_r0="000000" then
                zr0<='1';
            end if;
        --else
--              zr0<='0';
        end if;


end process;
--------------------R1----------------------
process(clk,LD1,busline,rin_r1,zr1)
begin
    rin_r1<=busline;
    if (rising_edge(clk)) then
        if (LD1='1') then
            rout_r1<=rin_r1;
        else
            rout_r1<=rout_r1;
        end if;

        if LD1='1' then
            if rin_r1="000000" then
                zr1<='1';
            end if;

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

```
-----------------R2--------------------------
process(clk,LD2,busline,rin_r2,zr2)
begin
    rin_r2<=busline;

    if (rising_edge(clk)) then
        if (LD2='1') then
            rout_r2<=rin_r2;
        else
            rout_r2<=rout_r2;
        end if;

        if LD2='1' then
            if rin_r2="000000" then
                zr2<='1';
            end if;

        end if;
    end if;
end process;
-----------------R3--------------------------------
process(clk,LD3,busline,rin_r3,zr3)
begin

    rin_r3<=busline;


    if (rising_edge(clk)) then
        if (LD3='1') then
            rout_r3<=rin_r3;
        else
            rout_r3<=rout_r3;
        end if;

        if LD3='1' then
            if rin_r3="000000" then
                zr3<='1';
            end if;

        end if;

    end if;
end process;
---------------------IR REG-------------------------
process(clk,LD_IR,rin_ir,busline)
begin
    rin_ir<=busline;
    if (rising_edge(clk)) then
          if  LD_IR ='1' then
            rout_ir <=rin_ir ;
          else
              rout_ir<=rout_ir;


        end if;
    end if;


end process;
-------------------- PC REG--------------------------
```

```
process(clk,LD_PC,busline,clr_pc,pc,inc_pc,pr_state)
begin

    if rising_edge(clk) then

        if  inc_pc='1'and (pr_state=state1 or pr_state=state3) then
            pc<=pc + "0000001";


        elsif clr_pc='1' then
            pc<= "0000000";

        elsif (LD_PC='1') then

            pc <=busline ;
            end if;
    end if;
end process;

-------------------------------------------------

------mux1 ---------------------------------------
process (reg_select_mux1,rout_r0,rout_r1,rout_r2,rout_r3)

begin

    case reg_select_mux1 is
        when "00" => mux1<=rout_r0;
        when "01" => mux1<=rout_r1;
        when "10" => mux1<=rout_r2;
        when "11" => mux1<=rout_r3;
        when others =>mux1<="1111111";
    end case;
end process;
--------mux0 ---------------------------------------
process (reg_select_mux0,rout_r0,rout_r1,rout_r2,rout_r3)

begin

    case reg_select_mux0 is
        when "00" => mux0<=rout_r0;
        when "01" => mux0<=rout_r1;
        when "10" => mux0<=rout_r2;
        when "11" => mux0<=rout_r3;
        when others =>mux0<="1111111";
    end case;
end process;

----------------ALU-----------------------------------
process(cmd,mux0,mux1)
begin

    case cmd is
        when "01" => aluresult<= mux0 + mux1;
        when "10" => aluresult<= mux0 - mux1;
        ---multiply
        --when "11" => aluresult<= mux0 mux1;
        when others => aluresult <= "0000000";
    end case;
end process;
-------------------------output--------------------------
```

```
    output<=rout_r0;

-------------------------------------------------------

-------------------- bus mux ---------------------------------
process(bus_sel,MData,aluresult)
begin
    case bus_sel is
        when '0' => busline <= MData ;
        when '1'=> busline<=aluresult;
        when others =>
    end case;
end process;
------------------------ ROM memory ---------------------------------
process(pc)
begin
    case pc is
        when "0000000" => MData <=my_Rom(0);
        when "0000001" => MData <=my_Rom(1);
        when "0000010" => MData <=my_Rom(2);
        when "0000011" => MData <=my_Rom(3);
        when "0000100" => MData <=my_Rom(4);
        when "0000101" => MData<=my_Rom(5);
        when "0000110" => MData<=my_Rom(6);
        when "0000111" => MData<=my_Rom(7);
        when "0001000" => MData<=my_Rom(8);
        when "0001001" => MData<=my_Rom(9);
        when "0001010" => MData<=my_Rom(10);
        when "0001011" => MData<=my_Rom(11);
        when "0001100" => MData<=my_Rom(12);
        when others =>    MData <="0000000";
    end case;
end process;
------------------------ control unit ---------------------------------
process(reset,clk)
begin
    if reset='1' then
        pr_state<=state0;
    elsif (rising_edge(clk)) then
        pr_state<= nx_state;
    end if;
end process;




process(nx_state,pr_state,rout_ir,pc,busline, cmd)
begin
    reg_select_mux0<=rout_ir(3)&rout_ir(2);
    reg_select_mux1<=rout_ir(1)&rout_ir(0);
    ir_select<=rout_ir(6)&rout_ir(5)&rout_ir(4);
    case pr_state is
        when state0 =>
            clr_pc <='1';
            nx_state<=state1;
        when state1 =>
            LD_IR   <='1';
            LD0<='0';
            LD1<='0';
            LD2<='0';
            LD3<='0';
            LD_pc<='0';
```

```
            inc_pc<='1';
            bus_sel<='0';
    --------------end of rom memory---------------------------
            if pc >"0001100" then
                nx_state<=state0;
            else
                nx_state<=state8;
            end if;

----  wait for one clk --------
        when state8 =>
        inc_pc<='0';
        LD_IR<='0';

        case ir_select is
            when "111" =>
            nx_state<=state2;  --hlt

            when "001" =>
            nx_state<=state3;   --load
            when "010" =>
            nx_state<=state4; --add
            when "011" =>
            nx_state<=state5; --sub
            when "100"=>
            --checking zrx------------
                    case reg_select_mux0 is
                    --enabling reg0 first input
                    when "00" =>
                    if zr0='1' then
                        nx_state <=state9;  ---do nothing
                    else
                        nx_state<=state6;   ---jnz

                    end if;
                    --checking register2
                    when "01" =>

                    if zr1='1' then
                        nx_state<=state9;---do nothing
                    else
                        nx_state<=state6;---jnz
                    end if;

                    --enabling reg2 first input
                    when "10"=>
                    if zr2='1' then
                        nx_state<=state9;--- do nothing
                    else
                        nx_state<=state6;---jnz
                    end if;

                    --checking register3
                    when "11" =>
                    if zr3='1' then
                        nx_state<=state9;-- do nothing
                    else
                        nx_state<=state6;--jnz
                    end if;


                    when others =>
```

```
                end case;

            when "101"=>
            nx_state<=state7; --mul
            when others=>
        end case;
-------------------------------------------
        when state2=>
            nx_state<=state2;
-----------------------------load----------------
        when state3=>
        nx_state<=state1;

            inc_pc<='1' ;
            bus_sel<='0';

        case reg_select_mux0 is
            --enabling reg0 first input
            when "00" =>
                LD0<='1';
            --enabling reg1 first input
            when "01" =>
                LD1<='1';
            --enabling reg2 first input
            when "10"=>
                LD2<='1';
            --enabling reg3 first input
            when "11" =>
            LD3<='1';
            when others =>

        end case;
------------------------------add ---------------------------------
        when state4 =>
                nx_State<=state1;
                cmd<="01";
                bus_sel<='1';

        case reg_select_mux0 is
            --enabling reg0 first input
            when "00" =>
            LD0<='1';

            --enabling reg1 first input
            when "01" =>
            LD1<='1';

            --enabling reg2 first input
            when "10"=>
            LD2<='1';

            --enabling reg3 first input
            when "11" =>
            LD3<='1';

            when others =>

        end case;

        --case reg_select_mux1 is
--          --enabling reg0 second input
--          when "00" =>
--          LD0<='1';
```

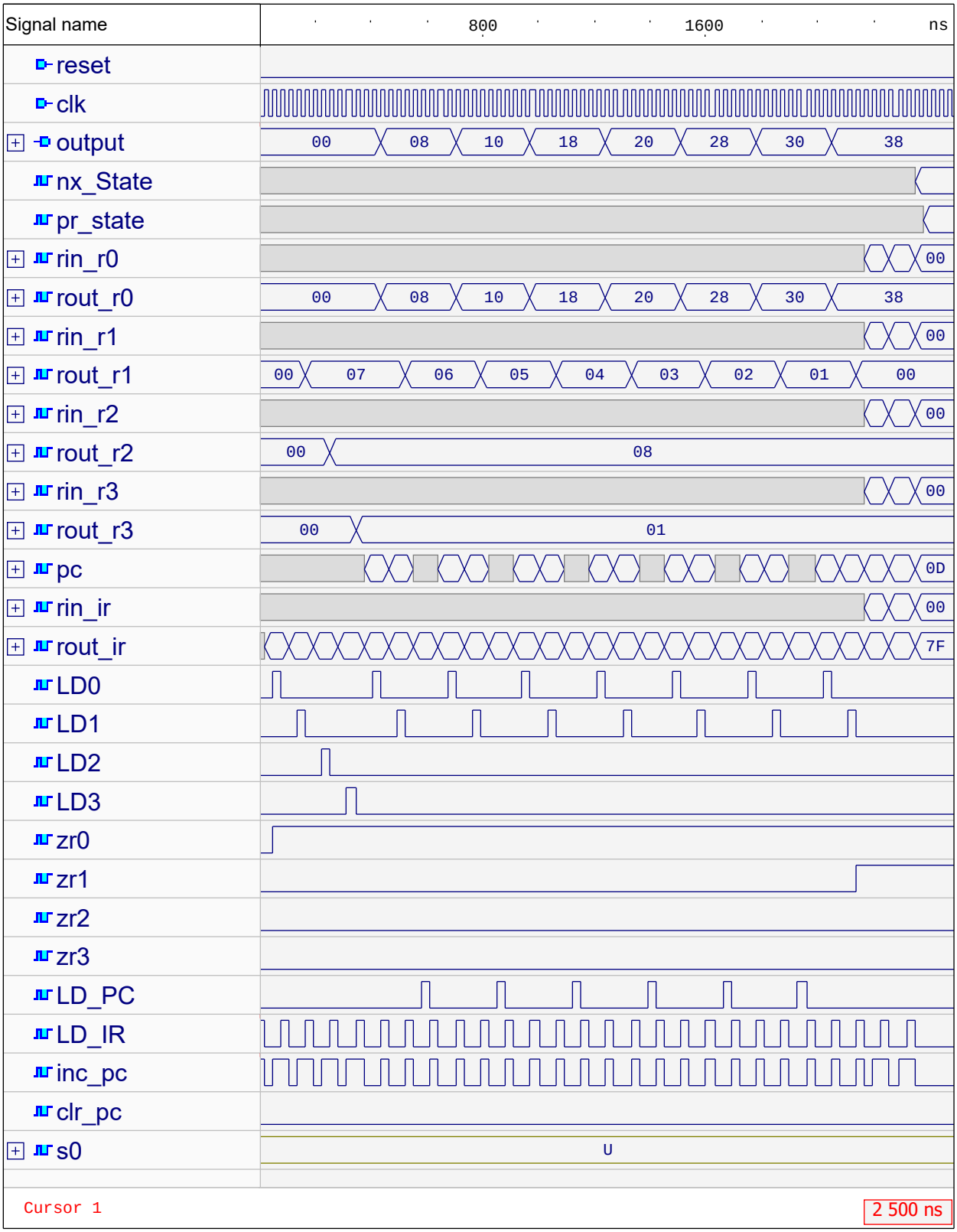
```
--
--          --enabling reg1 second input
--          when "01" =>
--          LD1<='1';
--
--          --enabling reg2 second input
--          when "10"=>
--          LD2<='1';
--
--          --enabling reg3 second input
--          when "11" =>
--          LD3<='1';
--
--          when others =>
--
--      end case;
```

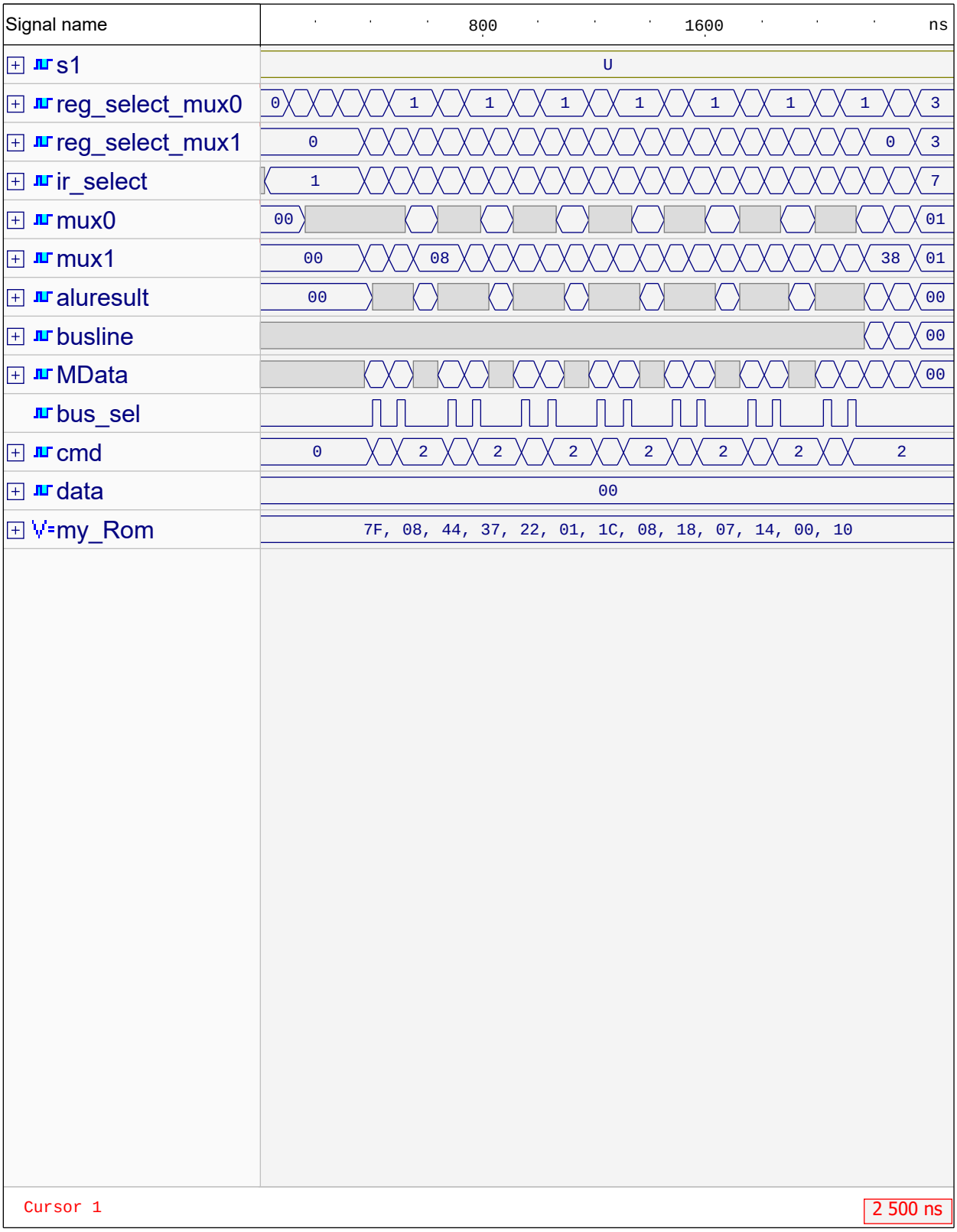
```
-------------------------sub---------------------------------
        when state5=>
                nx_state<=state1;
                cmd<="10";
                bus_sel<='1';
            case reg_select_mux0 is
            --enabling reg0 first input
            when "00" =>
            LD0<='1';


            --enabling reg1 first input
            when "01" =>
            LD1<='1';

            --enabling reg2 first input
            when "10"=>
            LD2<='1';

            --enabling reg3 first input
            when "11" =>
            LD3<='1';

            when others =>

        end case;

        --case reg_select_mux1 is
--          --enabling reg0 second input
--          when "00" =>
--          LD0<='1';
--
--          --enabling reg1 second input
--          when "01" =>
--          LD1<='1';
--
--          --enabling reg2 second input
--          when "10"=>
--          LD2<='1';
--
```

```
--          --enabling reg3 second input
--          when "11" =>
--          LD3<='1';
--
--          when others =>
--
--      end case;
-----------------------------jnz-------------------------------
        when state6=>
            LD_PC<='1' ;
            bus_sel<='0';
            nx_state<=state1;
---------------------------mul ---------------------------------------
        when state7=>
                nx_state<=state1;
                cmd<="11";
                bus_sel<='1';
                --enabling reg0 first input
        case reg_select_mux0 is
            when "00" =>
            LD0<='1';

            --enabling reg1 first input
            when "01" =>
            LD1<='1';

            --enabling reg2 first input
            when "10"=>
            LD2<='1';

            --enabling reg3 first input
            when "11" =>
            LD3<='1';

            when others =>

        end case;

        --case reg_select_mux1 is
--          --enabling reg0 second input
--          when "00" =>
--          LD0<='1';
--
--          --enabling reg1 second input
--          when "01" =>
--          LD1<='1';
--
--          --enabling reg2 second input
--          when "10"=>
--          LD2<='1';
--
--          --enabling reg3 second input
--          when "11" =>
--          LD3<='1';
--
--          when others =>
--
--      end case;

-----------------------------------------------nothing? ------------------
    when state9=>
        inc_pc<='1';
        nx_state<=state1;
```

```
    end case;
end process;

-----------------------------------------------------------------


end simplecpu;
```

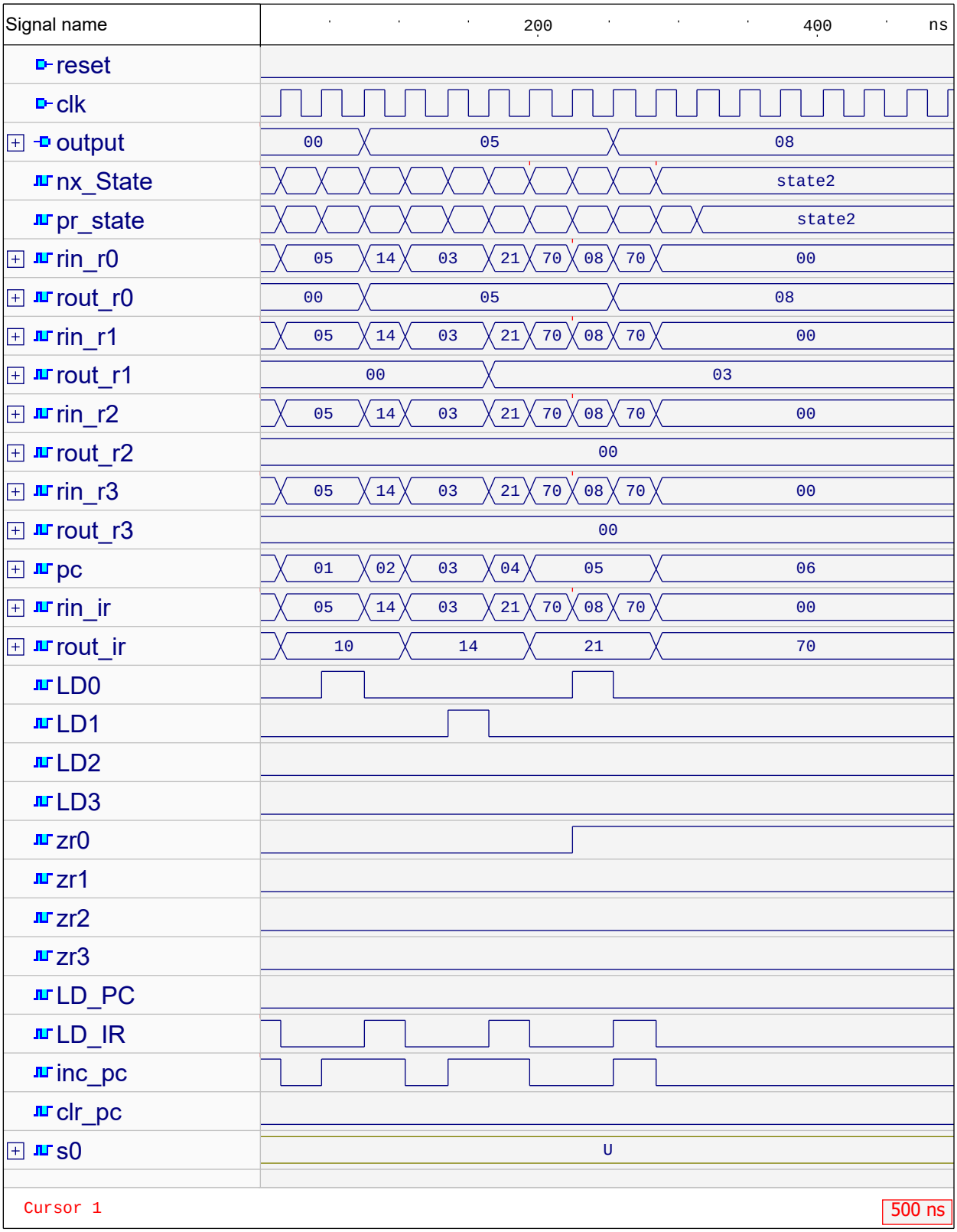
2.2 multiplying.awc

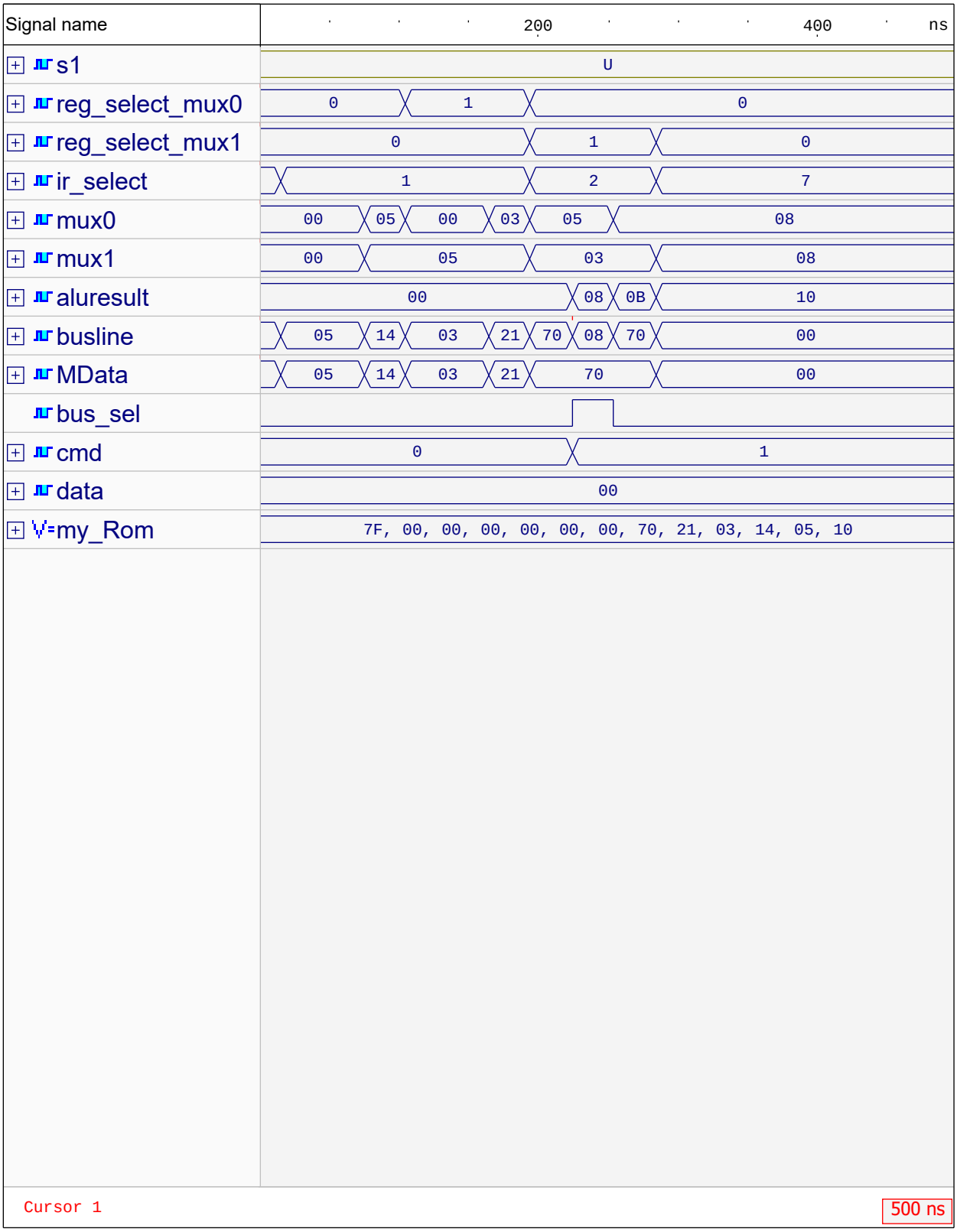



Signal name
800
1600
ns
s1
U
reg_select_mux0
0
1
1
1
1
1
1
1
3
reg_select_mux1
0
0
3
ir_select
1
7
mux0
00
01
mux1
00
08
38
01
aluresult
00
00
busline
00
MData
00
bus_sel
cmd
0
2
2
2
2
2
2
2
data
00
my_Rom
7F, 08, 44, 37, 22, 01, 1C, 08, 18, 07, 14, 00, 10
Cursor 1
2 500 ns

2.3 waves

2.3.1 add.awc


Signal name
200
400
ns
reset
clk
output
00
05
08
nx_State
state2
pr_state
state2
rin_r0
05 14 03 21 70 08 70 00
rout_r0
00 05 08
rin_r1
05 14 03 21 70 08 70 00
rout_r1
00 03
rin_r2
05 14 03 21 70 08 70 00
rout_r2
00
rin_r3
05 14 03 21 70 08 70 00
rout_r3
00
pc
01 02 03 04 05 06
rin_ir
05 14 03 21 70 08 70 00
rout_ir
10 14 21 70
LD0
LD1
LD2
LD3
zr0
zr1
zr2
zr3
LD_PC
LD_IR
inc_pc
clr_pc
s0
U
Cursor 1
500 ns


Signal name
200
400
ns
s1
U
reg_select_mux0
0
1
0
reg_select_mux1
0
1
0
ir_select
1
2
7
mux0
00
05
00
03
05
08
mux1
00
05
03
08
aluresult
00
08
0B
10
busline
05
14
03
21
70
08
70
00
MData
05
14
03
21
70
00
bus_sel
cmd
0
1
data
00
my_Rom
7F, 00, 00, 00, 00, 00, 00, 70, 21, 03, 14, 05, 10
Cursor 1
500 ns

2.4 TestBench

2.4.1 simplecpu_TB.vhd

```
library ieee;
use ieee.NUMERIC_STD.all;
use ieee.STD_LOGIC_UNSIGNED.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

    -- Add your library and packages declaration here ...

entity simplecpu_tb is
end simplecpu_tb;

architecture TB_ARCHITECTURE of simplecpu_tb is
    -- Component declaration of the tested unit
    component simplecpu
    port(
        reset : in STD_LOGIC;
        clk : in STD_LOGIC;
        output : out STD_LOGIC_VECTOR(6 downto 0) );
    end component;

    -- Stimulus signals - signals mapped to the input and inout ports of teste
d entity
    signal reset : STD_LOGIC;
    signal clk : STD_LOGIC;
    -- Observed signals - signals mapped to the output ports of tested entity
    signal output : STD_LOGIC_VECTOR(6 downto 0);

    -- Add your code here ...

begin

    -- Unit Under Test port map
    UUT : simplecpu
        port map (
            reset => reset,
            clk => clk,
            output => output
        );

    -- Add your stimulus here ...

end TB_ARCHITECTURE;

configuration TESTBENCH_FOR_simplecpu of simplecpu_tb is
    for TB_ARCHITECTURE
        for UUT : simplecpu
            use entity work.simplecpu(simplecpu);
        end for;
    end for;
end TESTBENCH_FOR_simplecpu;
```

2.4.2 simplecpu_TB_runtest.do

```
SetActiveLib -work
comp -include "$dsn\src\simplecpu.vhd"
```

```
comp -include "$dsn\src\TestBench\simplecpu_TB.vhd"
asim +access +r TESTBENCH_FOR_simplecpu
wave
wave -noreg reset
wave -noreg clk
wave -noreg output
# The following lines can be used for timing simulation
# acom <backannotated_vhdl_file_name>
# comp -include "$dsn\src\TestBench\simplecpu_TB_tim_cfg.vhd"
# asim +access +r TIMING_FOR_simplecpu
```