VHDL Source Code Master Index

Table of Contents

1. Addition Unit – Adder.vhd

```
2.  LIBRARY ieee;
3.  USE ieee.std_logic_1164.all;
4.  USE ieee.numeric_std.all;
5.  --decided to keep all modules in the same file for submission purposes
6.
7.  Entity full_adder is --initialize entity
8.    port (
9.        input1  : in  std_logic;
10.       input2  : in  std_logic;
11.       i_carry : in  std_logic;
12.       result   : out std_logic;
13.       o_carry : out std_logic);
14. end full_adder;
15.
16. architecture logic of full_adder is
17.
18. begin
19.  -- logic for a single full adder
20.  result <= input1 XOR input2 XOR i_carry ;
21.  o_carry <= (input1 AND input2) OR (i_carry AND input1) OR (i_carry AND
    input2) ;
22.
23. end logic;
24.
25.
26. LIBRARY ieee;
27. USE ieee.std_logic_1164.all;
28. USE ieee.numeric_std.all;
29.
30.
31. Entity Adder is -- initialize entity
32. Generic ( N : natural := 64 );
33. Port ( A, B : in std_logic_vector( N-1 downto 0 );
34. Y : out std_logic_vector( N-1 downto 0 );
35. -- Control signals
36. Cin : in std_logic;
37. -- Status signals
38. Cout, Ovfl : out std_logic );
39. End Entity Adder;
40.
41.
42. architecture rtl of Adder is
43.
44.   component full_adder is --load component into adder
45.     port (
46.       input1  : in  std_logic;
47.       input2  : in  std_logic;
48.       i_carry : in  std_logic;
49.       result   : out std_logic;
50.       o_carry : out std_logic);
51.   end component full_adder;
52.
```

```
53.   signal temp_carry : std_logic_vector(N downto 0); --create temporary signals
    to hold carry and result of calculation
54.   signal temp_result   : std_logic_vector(N-1 downto 0);
55.
56.
57. begin
58.
59.   temp_carry(0) <= Cin;
60.
61.   adder_loop : for i in 0 to N-1 generate -- generate sequence of full adders
62.     full_adder_inst : full_adder
63.       port map (
64.         input1  => A(i),
65.         input2  => B(i),
66.         i_carry => temp_carry(i),
67.         result   => temp_result(i),
68.         o_carry => temp_carry(i+1)
69.         );
70.   end generate adder_loop;
71.  --create and pass output signals
72.   Y <= temp_result;
73.   Cout <= temp_carry(N);
74.   Ovfl <= temp_carry(N) XOR temp_carry(N-1);
75.
76.
77. end rtl;
78.
```

2. Arithmatic Unit – ArithUnit.vhd

```
library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity ArithUnit is
Generic ( N : natural := 64 );
Port ( A, B : in std_logic_vector( N-1 downto 0 );
 AddY, Y : out std_logic_vector( N-1 downto 0 );
-- Control signals
AddnSub, ExtWord : in std_logic := '0';
-- Status signals
Cout, Ovfl, Zero, AltB, AltBu : out std_logic );
End Entity ArithUnit;

architecture rtl of ArithUnit is -- declare architecture
signal sgn_ext_vec : unsigned( (N/2)-1 downto 0 );
signal nor_temp : std_logic_vector( N-1 downto 0 );
signal Bsig : std_logic_vector(N-1 downto 0);
signal ZeroVector : std_logic_vector(N-1 downto 0) := (others => '0');

component Adder is -- load component
Port ( A, B : in std_logic_vector( N-1 downto 0 );
```

```
Y : out std_logic_vector( N-1 downto 0 );
-- Control signals
Cin : in std_logic;
-- Status signals
Cout, Ovfl : out std_logic );
end component;
begin

Adder1 : entity work.Adder(rtl) -- mapping adder component
        generic map (N => N)
        port map (
            A => A,
            B => Bsig,
            Y => AddY,
            Cin => AddnSub,
            Cout => Cout,
            Ovfl => Ovfl);


ones_complement: process(AddnSub) -- change number into its ones compliment form in
order to perform subtraction
begin
     if AddnSub = '1' then
        Bsig <= NOT B;
    else
        Bsig <= B;
     end if;
 end process ones_complement;

p_mux : process(AddY, ExtWord) -- sign extention for 32 bit vectors
begin
    case ExtWord is
        when '0' => sgn_ext_vec <= unsigned(AddY(N-1 downto (N/2)));
        when '1' => sgn_ext_vec <= (others => AddY((N/2)-1));
        when others => sgn_ext_vec <= (others => '0');
    end case;
end process p_mux;

Y <= std_logic_vector(sgn_ext_vec) & AddY((N/2)-1 downto 0); -- passing addition
output

nor1: process(AddY) -- nor gate to determine if result of arithmetic is zero
variable result: STD_LOGIC;
    begin
    result := '0';
    for i in 0 to N-1 loop
        result := result or AddY(i);
    end loop;
        Zero <= not result; -- zero flag
end process nor1;

AltBu <= NOT Cout; -- flag signals for branch and SLT instructions
AltB <= Ovfl XOR AddY(N-1);

end rtl;
```

Execution Unit – ExecUnit.vhd

```
library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity ExecUnit is -- initialize entity
Generic ( N : natural := 64 );
Port ( A, B : in std_logic_vector( N-1 downto 0 );
FuncClass, LogicFN, ShiftFN : in std_logic_vector( 1 downto 0 );
AddnSub, ExtWord : in std_logic := '0';
Y : out std_logic_vector( N-1 downto 0 );
Zero, AltB, AltBu : out std_logic );
End Entity ExecUnit;


architecture rtl of ExecUnit is -- declare architecture
signal Cout, Ovfl : std_logic;
signal ArithOut, LogicOut, ShiftOut : std_logic_vector( N-1 downto 0 );
signal AltOut : unsigned( N-1 downto 0 ) := (others => '0');

component ArithUnit is -- add components
Port ( A, B : in std_logic_vector( N-1 downto 0 );
AddY, Y : out std_logic_vector( N-1 downto 0 );
-- Control signals
AddnSub, ExtWord : in std_logic := '0';
-- Status signals
Cout, Ovfl, Zero, AltB, AltBu : out std_logic );
end component;

component ShiftUnit is -- add component
Port ( A, B, C : in std_logic_vector( N-1 downto 0 );
Y : out std_logic_vector( N-1 downto 0 );
ShiftFN : in std_logic_vector( 1 downto 0 );
ExtWord : in std_logic );
end component;

component LogicUnit is -- add component
Port ( A, B : in std_logic_vector( N-1 downto 0 );
            Y : out std_logic_vector( N-1 downto 0 );
     LogicFN : in std_logic_vector( 1 downto 0 ) );
end component;

begin


    ArithUnit1 : entity work.ArithUnit(rtl) -- port map to component
            generic map (N => N)
            port map (
                A => A,
                B => B,
```

```
                AddnSub => AddnSub,
                ExtWord => ExtWord,
                Zero => Zero,
                AltB => AltB,
                AltBu => AltBu,
                Y => ArithOut
            );

    ShiftUnit1 : entity work.ShiftUnit(rtl) -- port map to component
            generic map (N => N)
            port map (
                A => A,
                B => B,
                C => ArithOut,
                ShiftFN => ShiftFN,
                ExtWord => ExtWord,
                Y => ShiftOut
            );


    LogicUnit1 : entity work.LogicUnit(rtl) -- port map to component
            port map (
                A => A,
                B => B,
                LogicFN => LogicFN,
                Y => LogicOut
            );


    out_mux : process(AltBu, AltB, LogicOut, shiftOut, FuncClass)
    begin
        case FuncClass is -- select signal for multiplexor at execution unit output
            when "11" => Y <= std_logic_vector(AltOut(N-1 downto 1)) & AltBu;
            when "10" => Y <= std_logic_vector(AltOut(N-1 downto 1)) & AltB;
            when "01" => Y <= LogicOut;
            when "00" => Y <= ShiftOut;
            when others => Y <= ShiftOut;
        end case;
    end process out_mux;


end rtl;
```

Logic Unit – LogicUnit.vhd

```
library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity LogicUnit is -- initialize entity
Generic ( N : natural := 64 );
Port ( A, B : in std_logic_vector( N-1 downto 0 );
```

```
Y : out std_logic_vector( N-1 downto 0 );
LogicFN : in std_logic_vector( 1 downto 0 ) );
End Entity LogicUnit;


architecture rtl of LogicUnit is -- declare architecture
begin

p_mux : process(A, B, LogicFN) -- multiplexor to determine logic operation performed
if any and pass it to output signal.
begin
    case LogicFN is
        when "00" => Y <= B;
        when "01" => Y <= A XOR B;
        when "10" => Y <= A OR B;
        when "11" => Y <= A AND B;
        when others => Y <= B;
    end case;
end process p_mux;

end rtl;
```

Shifting Unit – ShiftUnit.vhd

```
library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.math_real.all; --required for ceil()

Entity ShiftUnit is -- declare entity
Generic ( N : natural := 64 );
Port ( A, B, C : in std_logic_vector( N-1 downto 0 );
Y : out std_logic_vector( N-1 downto 0 );
ShiftFN : in std_logic_vector( 1 downto 0 );
ExtWord : in std_logic );
End Entity ShiftUnit;

architecture rtl of ShiftUnit is -- initialize signals
signal wordAnd : std_logic;
signal shiftSig : unsigned( integer(ceil(log2(real(N))))-1 downto 0 ); -- determine
size of shiftcount vector based on N
signal SLLout, SRLout, SRAout, shiftIn, shiftOut : std_logic_vector( N-1 downto 0 );
signal SwapWord : std_logic_vector( N-1 downto 0 );
signal mux1OutUp, mux1OutLow : std_logic_vector( N-1 downto 0 );
signal mux2OutUp, mux2OutLow : std_logic_vector( N-1 downto 0 );
signal SgnExtUp, SgnExtLow : std_logic_vector( (N/2)-1 downto 0 );

component SLL64 is -- import component for logical left
Port ( X : in std_logic_vector( N-1 downto 0 );
Y : out std_logic_vector( N-1 downto 0 );
ShiftCount : in unsigned( integer(ceil(log2(real(N))))-1 downto 0 ) );
end component;
```

```
component SRL64 is -- import component for logical right
Port ( X : in std_logic_vector( N-1 downto 0 );
Y : out std_logic_vector( N-1 downto 0 );
ShiftCount : in unsigned( integer(ceil(log2(real(N))))-1 downto 0 ) );
end component;

component SRA64 is -- import component for arithmetic right shift
Port ( X : in std_logic_vector( N-1 downto 0 );
Y : out std_logic_vector( N-1 downto 0 );
ShiftCount : in unsigned( integer(ceil(log2(real(N))))-1 downto 0 ) );
end component;

begin

    wordAnd <= ShiftFN(1) AND ExtWord; -- signal to decide if swapping top 32 and
bottom 32 bits for 32 bit SRA
    SwapWord <= A((N/2)-1 downto 0) & A((N-1) downto (N/2)) ; -- swapping of the bits

    sixthbit_mask : process(shiftSig, B)
    begin
        case ExtWord is
            when '0' => shiftSig <= unsigned(B(5 downto 0)); -- take last 6 bits for
64 bit number
            when '1' => shiftSig <= '0' & unsigned( B( integer(ceil(log2(real(N))))-2
downto 0 )); -- take last 5 bits for 32 bit number
            when others => shiftSig <= unsigned(B(5 downto 0)); -- others cmd to
avoid breaking code
        end case;
    end process sixthbit_mask;

    word_mux : process(A, wordAnd)
    begin
        case wordAnd is -- passing of the swapped or non swapped bits to the shifting
units
            when '0' => shiftIn <= A;
            when '1' => shiftIn <= SwapWord;
            when others => shiftIn <= A;
        end case;
    end process word_mux;


    SLL641 : entity work.SLL64(rtl) -- mapping component
            generic map (N => N)
            port map (
                X => shiftIn,
                Y => SLLout,
                ShiftCount => shiftSig
            );

    SRL641 : entity work.SRL64(rtl) -- mapping component
            generic map (N => N)
            port map (
                X => shiftIn,
                Y => SRLout,
```

```
                ShiftCount => shiftSig
            );

    SRA641 : entity work.SRA64(rtl) -- mapping component
            generic map (N => N)
            port map (
                X => shiftIn,
                Y => SRAout,
                ShiftCount => shiftSig
            );


    shift_mux_low1 : process(SLLOut, C, ShiftFN) -- deciding which output to pass
from shifting units
    begin
        case ShiftFN(0) is
            when '0' => mux1OutLow <= C;
            when '1' => mux1OutLow <= SLLout;
            when others => mux1OutLow <= C;
        end case;
    end process shift_mux_low1;

    shift_mux_up1 : process(SRLOut, SRAout, ShiftFN) -- deciding which output to pass
from shifting units
    begin
        case ShiftFN(0) is
            when '0' => mux1OutUp <= SRLout;
            when '1' => mux1OutUp <= SRAout;
            when others => mux1OutUp <= SRLout;
        end case;
    end process shift_mux_up1;

    sgnExtLow <= (others => mux1OutLow((N/2)-1)); -- sign extention creation
    sgnExtUp <= (others => mux1OutUp((N-1)));

    shift_mux_low2 : process(mux1OutLow, SgnExtLow, ExtWord)
    begin
        case ExtWord is -- sign extention passing
            when '0' => mux2OutLow <= mux1OutLow;
            when '1' => mux2OutLow <= std_logic_vector(SgnExtLow) & mux1OutLow((N/2)-
1 downto 0);
            when others => mux2OutLow <= mux1OutLow;
        end case;
    end process shift_mux_low2;

    shift_mux_up2 : process(mux1OutUp, SgnExtUp, ExtWord)
    begin
        case ExtWord is -- sign extention passing
            when '0' => mux2OutUp <= mux1OutUp;
            when '1' => mux2OutUp <= std_logic_vector(SgnExtUp) & mux1OutUp((N-1)
downto (N/2));
            when others => mux2OutUp <= mux1OutUp;
        end case;
    end process shift_mux_up2;
```

```
    shift_final_mux : process(mux2OutUp, mux2OutLow, ShiftFN) -- final mux for
filtering output from top or bottom half of circuit
    begin
        case ShiftFN(1) is
            when '0' => shiftOut <= mux2OutLow;
            when '1' => shiftOut <= mux2OutUp;
            when others => shiftOut <= mux2OutLow;
        end case;
    end process shift_final_mux;

    Y <= shiftOut;

end rtl;
```

Shift left logical – SLL64.vhd

```
library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.math_real.all; --required for ceil()

Entity SLL64 is -- declare entity
Generic ( N : natural := 64 );
Port ( X : in std_logic_vector( N-1 downto 0 );
Y : out std_logic_vector( N-1 downto 0 );
ShiftCount : in unsigned( integer(ceil(log2(real(N))))-1 downto 0 ) );
End Entity SLL64;

architecture rtl of SLL64 is -- declare architecture
signal mout1, mout2, mout3 : std_logic_vector( N-1 downto 0 );
begin

shift1_mux : process(X, ShiftCount)
begin
    case ShiftCount(5 downto 4)  is -- first mux deciding 0 16 32 48 bit shift using
most significant bits of shiftcount
        when "00" => mout1 <= X;
        when "01" => mout1 <= std_logic_vector(shift_left(unsigned(X),16));
        when "10" => mout1 <= std_logic_vector(shift_left(unsigned(X),32));
        when "11" => mout1 <= std_logic_vector(shift_left(unsigned(X),48));
        when others => mout1 <= X;
    end case;
end process shift1_mux;

shift2_mux : process(mout1, ShiftCount)
begin
    case ShiftCount(3 downto 2) is -- second mux deciding 0 4 8 12 bit shift using
middle bits of shiftcount
        when "00" => mout2 <= mout1;
        when "01" => mout2 <= std_logic_vector(shift_left(unsigned(mout1),4));
        when "10" => mout2 <= std_logic_vector(shift_left(unsigned(mout1),8));
        when "11" => mout2 <= std_logic_vector(shift_left(unsigned(mout1),12));
        when others => mout2 <= mout1;
```

```
    end case;
end process shift2_mux;

shift3_mux : process(mout2, ShiftCount)
begin
    case ShiftCount(1 downto 0) is -- third mux deciding 0 1 2 3 bit shift using
least significant bits of shiftcount
        when "00" => mout3 <= mout2;
        when "01" => mout3 <= std_logic_vector(shift_left(unsigned(mout2),1));
        when "10" => mout3 <= std_logic_vector(shift_left(unsigned(mout2),2));
        when "11" => mout3 <= std_logic_vector(shift_left(unsigned(mout2),3));
        when others => mout3 <= mout2;
    end case;
end process shift3_mux;

Y <= mout3; -- passing output


end rtl;
```

Shift Right Logical – SRL64.vhd

```
library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.math_real.all; --required for ceil()

Entity SRL64 is -- declare entity
Generic ( N : natural := 64 );
Port ( X : in std_logic_vector( N-1 downto 0 );
Y : out std_logic_vector( N-1 downto 0 );
ShiftCount : in unsigned( integer(ceil(log2(real(N))))-1 downto 0 ) );
End Entity SRL64;

architecture rtl of SRL64 is -- declare architecture
signal mout1, mout2, mout3 : std_logic_vector( N-1 downto 0 );
begin

shift1_mux : process(X, ShiftCount)
begin
    case ShiftCount(5 downto 4)  is -- first mux deciding 0 16 32 48 bit shift using
most significant bits of shiftcount
        when "00" => mout1 <= X;
        when "01" => mout1 <= std_logic_vector(shift_right(unsigned(X),16));
        when "10" => mout1 <= std_logic_vector(shift_right(unsigned(X),32));
        when "11" => mout1 <= std_logic_vector(shift_right(unsigned(X),48));
        when others => mout1 <= X;
    end case;
end process shift1_mux;

shift2_mux : process(mout1, ShiftCount)
begin
```

```
    case ShiftCount(3 downto 2) is -- second mux deciding 0 4 8 12 bit shift using
middle bits of shiftcount
        when "00" => mout2 <= mout1;
        when "01" => mout2 <= std_logic_vector(shift_right(unsigned(mout1),4));
        when "10" => mout2 <= std_logic_vector(shift_right(unsigned(mout1),8));
        when "11" => mout2 <= std_logic_vector(shift_right(unsigned(mout1),12));
        when others => mout2 <= mout1;
    end case;
end process shift2_mux;

shift3_mux : process(mout2, ShiftCount)
begin
    case ShiftCount(1 downto 0) is -- third mux deciding 0 1 2 3 bit shift using
least significant bits of shiftcount
        when "00" => mout3 <= mout2;
        when "01" => mout3 <= std_logic_vector(shift_right(unsigned(mout2),1));
        when "10" => mout3 <= std_logic_vector(shift_right(unsigned(mout2),2));
        when "11" => mout3 <= std_logic_vector(shift_right(unsigned(mout2),3));
        when others => mout3 <= mout2;
    end case;
end process shift3_mux;

Y <= mout3; -- passing output


end rtl;
```

Shift Right Arithmetic – SRA64

```
library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.math_real.all; --required for ceil()


--SHIFT RIGHT ARITHMATIC
Entity SRA64 is -- declare entity
Generic ( N : natural := 64 );
Port ( X : in std_logic_vector( N-1 downto 0 );
Y : out std_logic_vector( N-1 downto 0 );
ShiftCount : in unsigned( integer(ceil(log2(real(N))))-1 downto 0 ) );
End Entity SRA64;

architecture rtl of SRA64 is -- declare architecture
signal mout1, mout2, mout3 : std_logic_vector( N-1 downto 0 );
begin

shift1_mux : process(X, ShiftCount)
begin
    case ShiftCount(5 downto 4)  is -- first mux deciding 0 16 32 48 bit shift using
most significant bits of shiftcount
        when "00" => mout1 <= X;
```

```
        when "01" => mout1 <= std_logic_vector(shift_right(signed(X),16));
        when "10" => mout1 <= std_logic_vector(shift_right(signed(X),32));
        when "11" => mout1 <= std_logic_vector(shift_right(signed(X),48));
        when others => mout1 <= X;
    end case;
end process shift1_mux;

shift2_mux : process(mout1, ShiftCount)
begin
    case ShiftCount(3 downto 2) is -- second mux deciding 0 4 8 12 bit shift using
middle bits of shiftcount
        when "00" => mout2 <= mout1;
        when "01" => mout2 <= std_logic_vector(shift_right(signed(mout1),4));
        when "10" => mout2 <= std_logic_vector(shift_right(signed(mout1),8));
        when "11" => mout2 <= std_logic_vector(shift_right(signed(mout1),12));
        when others => mout2 <= mout1;
    end case;
end process shift2_mux;

shift3_mux : process(mout2, ShiftCount)
begin
    case ShiftCount(1 downto 0) is -- third mux deciding 0 1 2 3 bit shift using
least significant bits of shiftcount
        when "00" => mout3 <= mout2;
        when "01" => mout3 <= std_logic_vector(shift_right(signed(mout2),1));
        when "10" => mout3 <= std_logic_vector(shift_right(signed(mout2),2));
        when "11" => mout3 <= std_logic_vector(shift_right(signed(mout2),3));
        when others => mout3 <= mout2;
    end case;
end process shift3_mux;

Y <= mout3; -- passing output


end rtl;
```