Appendix - VHDL Sourcecode Listings (In the order of Compile)

Table of Contents

Appendix - VHDL Sourcecode Listings (In the order of Compile)..... 1

myFA.vhd 2

myMux.vhd 2

Adder.vhd..... 3

ArithUnit.vhd 5

LogicUnit.vhd 6

SLL64.vhd 7

SRA64.vhd 8

SRL64.vhd..... 9

ShiftUnit.vhd 10

ExecUnit.vhd 11

TBExecUnit.vhd 12

ConfigExecUnit.vhd 16

Note: The LIBRARY/USE clauses at the top of the files are all omitted as requested except for the ones that are special cases.

myFA.vhd

```
Entity myFA is
    port(aFA, bFA, cin : in std_logic;
             sFA, coFA : out std_logic);
End myFA;

Architecture FABehave of myFA Is
Begin
    sFA <= (aFA XOR bFA) XOR cin;
    coFA <= ((aFA XOR bFA) AND cin) OR (aFA AND bFA);
End FABehave;

-- this entity would act as a sub-component to the carry-select adder network
```

myMux.vhd

```
Entity myMux is
    Port(A,B : in STD_LOGIC;
    Sel: in STD_LOGIC;
    Z: out STD_LOGIC);
end myMux;

Architecture MuxBehave of myMux is
Begin
    process(A,B,Sel)
    Begin
    if Sel = '0' then
    Z <= A;
    else
    Z <= B;
    end if;
    end process;
end MuxBehave;

-- This entity would act as a sub-component for the carry select adder
network
```

Adder.vhd

```
Entity Adder is
    Generic ( N : natural := 64 );
    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 Entity Adder;

Architecture Ripple of Adder is -- UNUSED in ArithUnit architecture beceause
of prop delay too high, kept here for record

Signal gen, prop: std_logic_vector(N-1 downto 0 ) := (others => '0');
Signal carry: std_logic_vector( N downto 0) := (others => '0');

Begin
    gen <= A AND B;
    prop <= A XOR B;
    carry(0) <= Cin;

    CarryNetwork: for i in 0 to N-1 generate -- using generate to create a
ripple carry adder chain
        carry(i+1) <= gen(i) OR (prop(i) AND carry(i));
    end generate CarryNetwork;

    Y <= carry(N-1 downto 0) XOR prop;
    Cout <= carry(N);
    Ovfl <= carry(N) XOR carry(N-1);

End Ripple;

Architecture CarrySelect of Adder is -- Optimized adder with way less prop
delay, USED in ArithUnit architecture

-- declaring the Full Adder and 2-input MUX as components for easy port
mapping
Component myFA is
port(aFA, bFA, cin : in std_logic;
            sFA, coFA : out std_logic);
End component;

Component myMux is
    Port(A,B : in STD_LOGIC;
    Sel: in STD_LOGIC;
    Z: out STD_LOGIC);
end component;

signal AMux,BMux,C0,C1: std_logic_vector(N-1 downto 0);
signal CarryS : std_logic_vector(N/4 downto 0);
signal CarryNMinus1 : std_logic;

Begin
```

```
CarryS(0) <= Cin;

CarrySelectNetwork: for j in 0 to (N/4)-1 generate
--generate 4-bit carry select adders 16 times in the case of N=64
FA1: myFA PORT MAP(A(0+4*j),B(0+4*j),'0' ,AMux(0+4*j),C0(0+4*j));
FA2: myFA PORT MAP(A(1+4*j),B(1+4*j),C0(0+4*j),AMux(1+4*j),C0(1+4*j));
FA3: myFA PORT MAP(A(2+4*j),B(2+4*j),C0(1+4*j),AMux(2+4*j),C0(2+4*j));
FA4: myFA PORT MAP(A(3+4*j),B(3+4*j),C0(2+4*j),AMux(3+4*j),C0(3+4*j));

FA5: myFA PORT MAP(A(0+4*j),B(0+4*j),'1' ,BMux(0+4*j),C1(0+4*j));
FA6: myFA PORT MAP(A(1+4*j),B(1+4*j),C1(0+4*j),BMux(1+4*j),C1(1+4*j));
FA7: myFA PORT MAP(A(2+4*j),B(2+4*j),C1(1+4*j),BMux(2+4*j),C1(2+4*j));
FA8: myFA PORT MAP(A(3+4*j),B(3+4*j),C1(2+4*j),BMux(3+4*j),C1(3+4*j));

MUX1: myMux PORT MAP(AMux(0+4*j),BMux(0+4*j),CarryS(j),Y(0+4*j));
MUX2: myMux PORT MAP(AMux(1+4*j),BMux(1+4*j),CarryS(j),Y(1+4*j));
MUX3: myMux PORT MAP(AMux(2+4*j),BMux(2+4*j),CarryS(j),Y(2+4*j));
MUX4: myMux PORT MAP(AMux(3+4*j),BMux(3+4*j),CarryS(j),Y(3+4*j));

MUX5: myMux PORT MAP(C0(3+4*j),C1(3+4*j),CarryS(j),CarryS(j+1));
End generate CarrySelectNetwork;

OvMux: myMux Port Map(c0(N-2),c1(N-2),CarryS((N/4)-1), CarryNMinus1);
--overflow Mux to fetch the second last carry bit for calculating Ovfl
Cout <= CarryS(N/4);
Ovfl <= CarryS(N/4) XOR CarryNMinus1;


end CarrySelect;
```

ArithUnit.vhd

```
use ieee.std_logic_misc.all; -- This is for or_reduce (or gate for bits of
the same vector)


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 arithBehaviour of ArithUnit is

    Signal Bsig, AddYsig: std_logic_vector( N-1 downto 0 ):= (others => '0');
    Signal C0, Ovflsig, Coutsig: std_logic := '0';

Begin


    Bsig <= NOT B when AddnSub = '1' else B; -- Invert B if subtraction is
desired
    C0 <= '1' when AddnSub = '1' else '0'; -- Set Cin as '1' to act as the
added 1 for 2's comp

    AddInst: ENTITY WORK.Adder(CarrySelect) generic map(N => N) port map(A =>
A, B => Bsig, Cin => C0, Y => AddYSig, Cout => Coutsig, Ovfl => Ovflsig);
    --Zero <= '1' when unsigned(AddYsig) = 0 else '0';
    Zero <= NOT or_reduce(AddYsig); -- NOR gate for bits of the same vector
    AltB <= Ovflsig XOR AddYsig(N-1);
    AltBu <= NOT Coutsig;

    Ovfl <= Ovflsig;
    AddY <= AddYsig;
    Cout <= Coutsig;
    Y <= AddYsig when ExtWord = '0' else ((N-1 downto N/2 => AddYsig((N/2)-
1)) & AddYsig((N/2)-1 downto 0));

End arithBehaviour;
```

LogicUnit.vhd

```
Entity LogicUnit is
    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 logicBehaviour of LogicUnit is

Begin

    -- LogicUnit MUX
    with LogicFN select
        Y <= B when "00",
            A XOR B when "01",
            A OR B when "10",
            A AND B when "11",
            (others => '0') when others;

End logicBehaviour;
```

SLL64.vhd

```
Entity SLL64 is
    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 shiftLL of SLL64 is

    Signal Mux1Out, Mux2Out : std_logic_vector(N-1 downto 0);

Begin

    --00:SLL(0) 01:SLL(16) 10:SLL(32) 11:SLL(48)
    with ShiftCount(5 downto 4) select
        Mux1Out <= X when "00",
            (X((N-1)-16 downto 0) & (15 downto 0 => '0')) when "01",
            (X((N-1)-32 downto 0) & (31 downto 0 => '0')) when "10",
            (X((N-1)-48 downto 0) & (47 downto 0 => '0')) when "11",
            (others => '0') when others;

   --00:SLL(0) 01:SLL(4) 10:SLL(8) 11:SLL(12)
    with ShiftCount(3 downto 2) select
        Mux2Out <= Mux1Out when "00",
            (Mux1Out((N-1)-4 downto 0) & (3 downto 0 => '0')) when "01",
            (Mux1Out((N-1)-8 downto 0) & (7 downto 0 => '0')) when "10",
            (Mux1Out((N-1)-12 downto 0) & (11 downto 0 => '0')) when "11",
            (others => '0') when others;

    --00:SLL(0) 01:SLL(1) 10:SLL(2) 11:SLL(3)
    with ShiftCount(1 downto 0) select
        Y <= Mux2Out when "00",
            (Mux2Out((N-1)-1 downto 0) & ('0')) when "01",
            (Mux2Out((N-1)-2 downto 0) & (1 downto 0 => '0')) when "10",
            (Mux2Out((N-1)-3 downto 0) & (2 downto 0 => '0')) when "11",
            (others => '0') when others;

end shiftLL;
```

SRA64.vhd

```
Entity SRA64 is
    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 shiftRA of SRA64 is

    Signal Mux1Out, Mux2Out : std_logic_vector(N-1 downto 0);

Begin

    --00:SRA(0) 01:SRA(16) 10:SRA(32) 11:SRA(48)
    with ShiftCount(5 downto 4) select
        Mux1Out <= X when "00",
            ((N-1 downto (N-1)-15 => X(N-1)) & X(N-1 downto 16)) when "01",
            ((N-1 downto (N-1)-31 => X(N-1)) & X(N-1 downto 32)) when "10",
            ((N-1 downto (N-1)-47 => X(N-1)) & X(N-1 downto 48)) when "11",
            (others => '0') when others;

   --00:SRA(0) 01:SRA(4) 10:SRA(8) 11:SRA(12)
    with ShiftCount(3 downto 2) select
        Mux2Out <= Mux1Out when "00",
            ((N-1 downto (N-1)-3 => Mux1Out(N-1)) & Mux1Out(N-1 downto 4))
when "01",
            ((N-1 downto (N-1)-7 => Mux1Out(N-1)) & Mux1Out(N-1 downto 8))
when "10",
            ((N-1 downto (N-1)-11 => Mux1Out(N-1)) & Mux1Out(N-1 downto 12))
when "11",
            (others => '0') when others;

    --00:SRA(0) 01:SRA(1) 10:SRA(2) 11:SRA(3)
    with ShiftCount(1 downto 0) select
        Y <= Mux2Out when "00",
            (Mux2Out(N-1) & Mux2Out(N-1 downto 1)) when "01",
            ((N-1 downto (N-1)-1 => Mux2Out(N-1)) & Mux2Out(N-1 downto 2))
when "10",
            ((N-1 downto (N-1)-2 => Mux2Out(N-1)) & Mux2Out(N-1 downto 3))
when "11",
            (others => '0') when others;

End shiftRA;
```

SRL64.vhd

```
Entity SRL64 is
    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 shiftRL of SRL64 is

    Signal Mux1Out, Mux2Out : std_logic_vector(N-1 downto 0);

Begin

    --00:SRL(0) 01:SRL(16) 10:SRL(32) 11:SRL(48)
    with ShiftCount(5 downto 4) select
        Mux1Out <= X when "00",
            ((N-1 downto (N-1)-15 => '0') & X(N-1 downto 16)) when "01",
            ((N-1 downto (N-1)-31 => '0') & X(N-1 downto 32)) when "10",
            ((N-1 downto (N-1)-47 => '0') & X(N-1 downto 48)) when "11",
            (others => '0') when others;

   --00:SRL(0) 01:SRL(4) 10:SRL(8) 11:SRL(12)
    with ShiftCount(3 downto 2) select
        Mux2Out <= Mux1Out when "00",
            ((N-1 downto (N-1)-3 => '0') & Mux1Out(N-1 downto 4)) when "01",
            ((N-1 downto (N-1)-7 => '0') & Mux1Out(N-1 downto 8)) when "10",
            ((N-1 downto (N-1)-11 => '0') & Mux1Out(N-1 downto 12)) when
"11",
            (others => '0') when others;

    --00:SRL(0) 01:SRL(1) 10:SRL(2) 11:SRL(3)
    with ShiftCount(1 downto 0) select
        Y <= Mux2Out when "00",
            ('0' & Mux2Out(N-1 downto 1)) when "01",
            ((N-1 downto (N-1)-1 => '0') & Mux2Out(N-1 downto 2)) when "10",
            ((N-1 downto (N-1)-2 => '0') & Mux2Out(N-1 downto 3)) when "11",
            (others => '0') when others;

end shiftRL;
```

ShiftUnit.vhd

```
Entity ShiftUnit is
    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 barrelShifter of ShiftUnit is

    Signal SwapWord: std_logic;
    Signal ShiftCount: unsigned( 5 downto 0 );
    Signal Asig, SLLsig, SRLsig, SRAsig, SLLorCmux, SLLExtmux, SRLorSRAmux,
SRLExtmux: std_logic_vector( N-1 downto 0 );

Begin

    -- Swap upper and lower 32 bits of a 64 bit operand when ExtWord and
ShiftFN(1) (right shift) are true
    SwapWord <= ExtWord and ShiftFN(1);
    Asig <= A when SwapWord = '0' else (A((N/2)-1 downto 0) & A(N-1 downto
N/2)); -- SwapWord MUX

    -- "Mask" we only need 5 LSB for 32-bit shift, or 6 LSB for 64-bit shift
    ShiftCount <= unsigned('0' & B(4 downto 0)) when ExtWord = '1' else
unsigned(B(5 downto 0));

    -- instantiates the three types of barrel shifters for different shift
operations
    ShiftLL: ENTITY WORK.SLL64(shiftLL) generic map(N => N) port map(X =>
Asig, Y => SLLsig, ShiftCount => ShiftCount);
    ShiftRL: ENTITY WORK.SRL64(shiftRL) generic map(N => N) port map(X =>
Asig, Y => SRLsig, ShiftCount => ShiftCount);
    ShiftRA: ENTITY WORK.SRA64(shiftRA) generic map(N => N) port map(X =>
Asig, Y => SRAsig, ShiftCount => ShiftCount);

    SLLorCmux <= C when ShiftFN(0) = '0' else SLLsig; -- SLL or C mux
    SLLExtmux <= SLLorCmux when ExtWord = '0' else ((N-1 downto N/2 =>
SLLorCmux((N/2)-1)) & SLLorCmux((N/2)-1 downto 0)); -- Sign extend lower 32
bits
    SRLorSRAmux <= SRLsig when ShiftFN(0) = '0' else SRAsig; -- SRL or SRA
mux
    SRLExtmux <= SRLorSRAmux when ExtWord = '0' else ((N-1 downto N/2 =>
SRLorSRAmux(N-1)) & SRLorSRAmux(N-1 downto N/2)); -- Sign extend upper 32
bits

    Y <= SLLExtmux when ShiftFN(1) = '0' else SRLExtmux;

End barrelShifter;
```

ExecUnit.vhd

```
Entity ExecUnit is
    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 execBehaviour of ExecUnit is

Signal AddResult, ShiftArithResult, LogicResult, AltBOut, AltBuOut :
std_logic_vector( N-1 downto 0 ); -- 64-bit signals
Signal AltBResult, AltBuResult : std_logic; -- 1-bit signals

Begin
    -- instantiate the three internal units and port mapping accordingly
    ArithUnit: ENTITY WORK.ArithUnit(arithBehaviour) generic map(N => N)
        port map(A => A, B => B, AddY => AddResult, Y => open,
                AddnSub => AddnSub, ExtWord => ExtWord,
                Cout => open, Ovfl => open, Zero => Zero,
                AltB => AltBResult, AltBu => AltBuResult);

    ShiftUnit: ENTITY WORK.ShiftUnit(barrelShifter) generic map(N => N)
        port map(A => A, B => B, C => AddResult, Y => ShiftArithResult,
ShiftFN => ShiftFN, ExtWord => ExtWord);

    LogicUnit: ENTITY WORK.LogicUnit(logicBehaviour) generic map(N => N)
        port map(A => A, B => B, Y => LogicResult, LogicFN => LogicFN);

    -- inserts 63 bits of '0' in front of the 1-bit AltBResult and
AltBuResult
    AltBOut <= (N-1 downto 1 => '0') & AltBResult;
    AltBuOut <= (N-1 downto 1 => '0') & AltBuResult;

    AltB <= AltBResult;
    AltBu <= AltBuResult;

    -- output MUX
    with FuncClass select
        Y <= ShiftArithResult when "00",
            LogicResult when "01",
            AltBOut when "10",
            AltBuOut when "11",
            (others => '0') when others;

End execBehaviour;
```

TBExecUnit.vhd

```
entity TBExecUnit is
    generic (N : natural := 64);
end entity TBExecUnit;


architecture behaveTB of TBExecUnit is
    constant TestVectorFile : string := "ExecUnit00.tvs"; -- filename
    file VectorFile : text; -- define file as type text
    signal MeasurementIndex : Integer := 0; -- measurement index for line
(and instruction) number of file

    constant PreStimTime : time := 1 ns; -- pre-stimulous time similar to
setup
    constant PostStimTime : time := 50 ns; -- post-stimulous time similar to
hold

    constant ClockPeriod : time := 2 ns;
    constant nResetPeriod : time := 5 ns;

    signal clock, nReset : std_logic := '0'; -- a clock for the ExUnit to
follow, and an active low reset line to initiate stimulous
    -- signals for the TB to connect to the ins and outs of the ExUnit
    signal A, B, Y, tbCompareY : std_logic_vector(N-1 downto 0);
    signal FuncClass, LogicFN, ShiftFN : std_logic_vector(1 downto 0);
    signal AddnSub, ExtWord, Zero, AltB, AltBu : std_logic;
    signal YandStatOut : std_logic_vector((N-1)+3 downto 0);

    signal stableYSig, quietYSig : boolean := false; -- stable and quiet
indicators

    constant nResetTime : time := 2 ns; -- a reset line to initiate the start
of the stimulous process
    constant clockTime : time := 2 ns; -- a the period of the clock

    -- bring in ExecUnit component to testbench
    component ExecUnit is
        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 component ExecUnit;

begin
    nReset <= '0', '1' after nResetPeriod; -- give nReset the time to change
back to non-reset
    clock <= NOT clock after ClockPeriod/2; -- construct the clock based on
clock period timing

    stableYSig <= Y'stable(PostStimTime); -- set stable Y signal indicator
    quietYSig <= Y'quiet(PostStimTime); -- set quiet Y signal indicator

    YandStatOut <= Y & Zero & AltB & AltBu; -- combine Y and Stat to see when
all outputs are quiet
```

```
DUT: component ExecUnit generic map( N => N ) -- Instantiate ExecUnit as our
DUT and do the port mapping
        port map ( A=>A, B=>B,
            FuncClass=>FuncClass, LogicFN=>LogicFN, ShiftFN=>ShiftFN,
            AddnSub=>AddnSub, ExtWord=>ExtWord, Y=>Y,
            Zero=>Zero, AltB=>AltB, AltBu=>AltBu );


STIMULOUS: process is -- Stimulous Process (main process for the testbench)
    variable startTime, endTime, propDelay : time := 0 ns; -- Timing
variables
    variable compareRes : std_logic := 'X'; -- variable used to compare Y
with tbCompareY result

-- Create variables to store parsed data from file into
    variable LineBuffer : line;
    variable AVar, BVar, YVar : std_logic_vector(N-1 downto 0);
    variable AddnSubVar, ExtWordVar, ZeroVar, AltBVar, AltBuVar : std_logic;
    variable FuncClassVar, LogicFNVar, ShiftFNVar : std_logic_vector(1 downto
0);

    begin
        wait until nReset = '1'; -- wait until the active low reset line
pulls high to start the process
        --wait for 10 ns; -- give some delay for line to
        file_open( VectorFile, TestVectorFile, read_mode );
        report "Opening test vector list from: " & TestVectorFile;
        while NOT endfile( VectorFile ) loop -- loop through file until the
end
            -- Initialize all tb signals as unknown (X)
            A <= (others => 'X');
            B <= (others => 'X');
            FuncClass <= "XX";
            LogicFN <= "XX";
            ShiftFN <= "XX";
            AddnSub <= 'X';
            ExtWord <= 'X';
            ZeroVar := 'X';
            AltBVar := 'X';
            AltBuVar := 'X';
            compareRes := 'X';

            MeasurementIndex <= MeasurementIndex + 1; -- incremement the
measurement index so we can keep track of our line number

            wait for PreStimTime; -- wait for pre-stimulous, then begin file
reading and ExUnit actions

            readline(VectorFile, LineBuffer); -- read current line in the
vector file

            -- Parse the data from LineBuffer into appropriate variables
            hread(LineBuffer, Avar);
            hread(LineBuffer, Bvar);
            read(LineBuffer, FuncClassvar(1));
            read(LineBuffer, FuncClassvar(0));
```

```
            read(LineBuffer, LogicFNvar(1));
            read(LineBuffer, LogicFNvar(0));
            read(LineBuffer, ShiftFNvar(1));
            read(LineBuffer, ShiftFNvar(0));
            read(LineBuffer, AddnSubvar);
            read(LineBuffer, ExtWordvar);
            hread(LineBuffer, Yvar);
            read(LineBuffer, Zerovar);
            read(LineBuffer, AltBvar);
            read(LineBuffer, AltBuvar);

            StartTime := NOW; -- Start the timer for propogation delay
measurement
            compareRes := '1'; -- Set comparison flag to true, it is easier
to interpret what doesn't match this way

            -- Pass the line-read variables to the TB signals that interact
with the ExUnit
            A <= AVar;
            B <= BVar;
            FuncClass <= FuncClassVar;
            LogicFN <= LogicFNVar;
            ShiftFN <= ShiftFNVar;
            AddnSub <= AddnSubVar;
            ExtWord <= ExtWordVar;
            tbCompareY <= YVar;

            wait until YandStatOut'Stable(PostStimTime) = true; -- Wait until
all output signals are stable...
            EndTime := NOW; -- ...then capture the end time to complete the
propogation delay measurement

            -- Propogation delay measurement that includes when the outputs
were last active
            PropDelay := EndTime - StartTime - YandStatOut'Last_Active;
            -- Report the propogation delay of the current measurement
instruction
            report "   Measurement Index: " & to_string(MeasurementIndex) & "
--  Propagation Delay = " & to_string(PropDelay);

            if Y /= tbCompareY then -- if the output Y and known Y do not
match, compareRes is false and assert the error
                compareRes := '0';
                assert Y = tbCompareY report "Y /= Expected Y for Measurement
Index: " & to_string(MeasurementIndex) & CR &
                    "  Y = " & to_hstring(Y) & CR & "Y = tbCompareY" &
to_hstring(tbCompareY) severity error;
            end If;

            if Zero /= ZeroVar then -- if the output zero flag and known flag
do not match, compareRes is false and assert the error
                compareRes := '0';
                assert Zero = ZeroVar report "Zero Flag Incorrect for
Measurement Index: " & to_string(MeasurementIndex) & CR &
                    "  Zero = " & to_string(Zero) & CR & "ZeroVar = " &
to_string(ZeroVar) severity error;
            end if;
```

```
            if AddnSubVar = '1' and ExtWordVar = '0' then -- check if we are
doing a 64-bit subtraction (used for branch conditions)
                if AltB /= AltBVar then -- if the output AltB flag and known
flag do not match, compareRes is false and assert the error
                    compareRes := '0';
                    assert AltB = AltBVar report "A less than B Flag
Incorrect for Measurement Index: " & to_string(MeasurementIndex) & CR &
                        "  AltB = " & to_string(AltB) & CR & "AltBVar = " &
to_string(AltBVar) severity error;
                end if;
                if AltBu /= AltBuVar then -- if the output AltBu flag and
known flag do not match, compareRes is false and assert the error
                    compareRes := '0';
                    assert AltBu = AltBuVar report "A less than B Unsigned
Flag Incorrect for Measurement Index: " & to_string(MeasurementIndex) & CR &
                        "  AltBu = " & to_string(AltBu) & CR & "AltBuVar = "
& to_string(AltBuVar) severity error;
                end if;
            end if;


        wait until clock = '1'; -- wait until next rising edge to begin next
instruction
        end loop;

        report "TestVector Finished, Closing File and Ending Simulation";
        file_close( VectorFile );
        wait;
    end process STIMULOUS;

end architecture behaveTB;
```

ConfigExecUnit.vhd

```
-- Functional Simulation
Configuration FuncExecUnitSim of TBExecUnit is
    for behaveTB
        for DUT : ExecUnit use entity
                work.ExecUnit(execBehaviour);
        end for;
    end for;
End Configuration FuncExecUnitSim;

-- Timing Simulation
Configuration TimeExecUnitSim of TBExecUnit is
    for behaveTB
        for DUT : ExecUnit use entity
                work.ExecUnit(structure);
        end for;
    end for;
End Configuration TimeExecUnitSim;
```