SAN JOSÉ STATE UNIVERSITY
DEPARTMENT OF ELECTRICAL ENGINEERING

EE 275 – Advanced Computer Architecture

Mini-Project Report

Topic: Pipelined half precision Floating-Point Adder Design

Deekshith Krishnegowda
MSEE
SJSU ID: 012417080
Date: 03/07/2019

1. Introduction

Floating Point Arithmetic is the most used way of approximating real number arithmetic for numerical calculations on modern computers. IEEE 754 is the technical standard for floating point computations.

IEEE floating point numbers have three basic components: the sign, the exponent, and the mantissa. The mantissa is composed of the fraction an implicit leading digit.

IEEE Floating Point Format:

Precision	**Sign**	**Exponent**	**Mantissa**
Single	1 [31]	8 [30-23]	23 [22-00]
Double	1[63]	11 [62-52]	52[51-00]
Half	1[15]	5[14:10]	10[9:0]

The aim of this project is implementing half precision 4 stage pipelined floating point adder according with the IEEE 754 standard. Verilog HDL is used for programming. The tool used for simulation is Synopsys VCS. The waveform is observed in EPwave.

2. Description of the design

Initially, unpipelined floating-point adder has been designed. The working of the designed has been checked. Then the design was changed to 4 stage pipelined adder. The algorithm for floating-point addition is as shown in figure-1. The 4 stages of pipeline are as described below.

Stage 1: Compare the exponents and determine the number of shifts required to align the mantissa to make the exponents equal. Then, right-shift the mantissa of the smaller exponent by the required amount (alignment).

Stage 2: Compare the two aligned mantissas and determine which is the smaller of the two. Take 2's complement of the smaller mantissa if the signs of the two numbers are different. Then, add the two mantissas. (addition).

Stage 3: Determine the number of shifts required and the corresponding direction to normalize the result (normalization-1).

Stage 4: Shift the mantissa to the required direction by the required amount. Adjust the exponent accordingly and check for any exceptional condition (normalization-2).

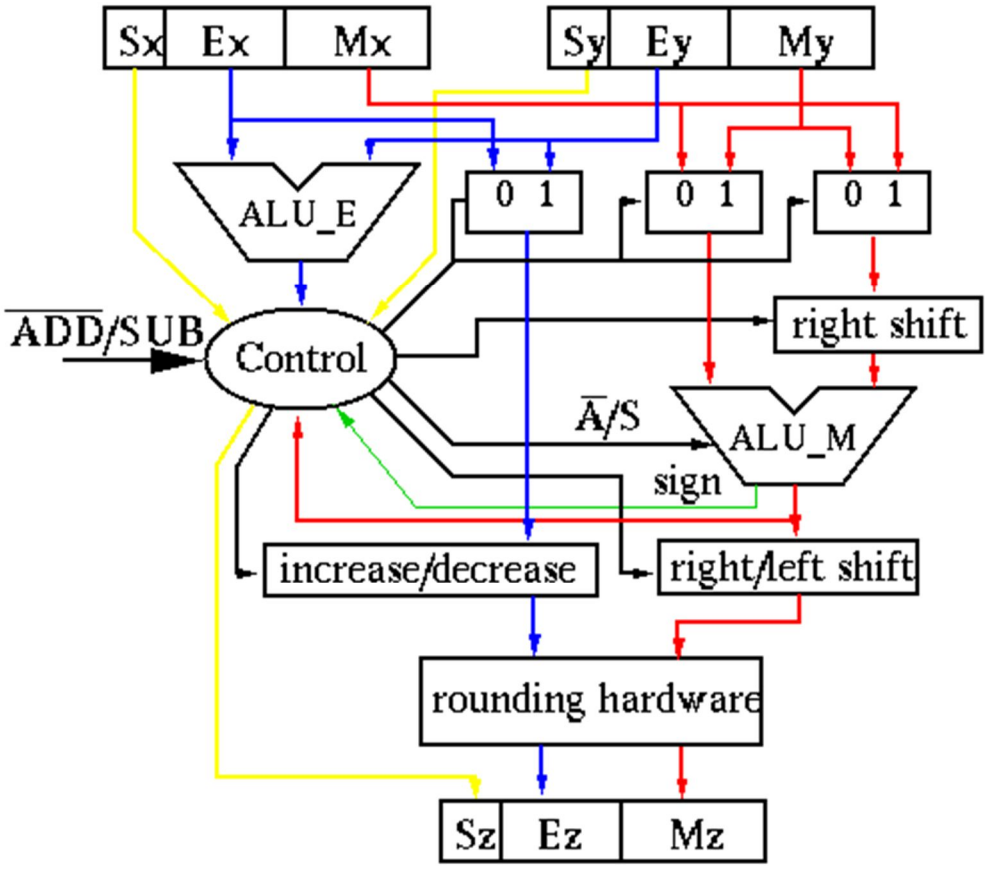
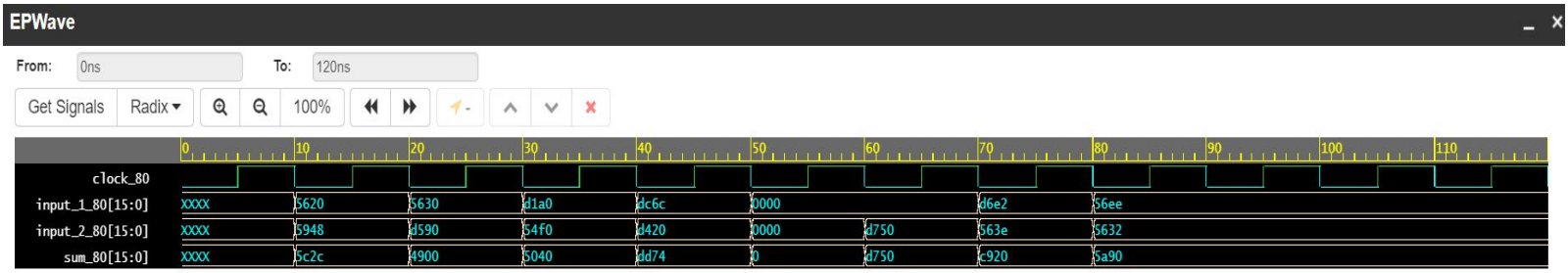


Figure1. Floating point arithmetic algorithm

3. Simulation and verification of results

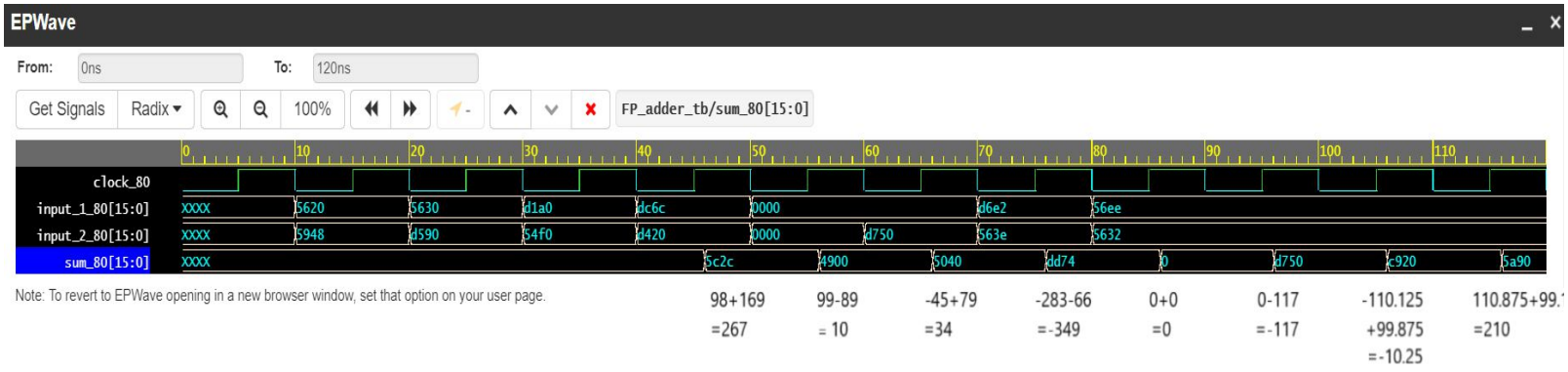
1. Simulations of adder without pipeline



98+169	99-89	-45+79	-283-66	0+0	0-117	-110.125	110.875+99.125
=267	= 10	=34	=-349	=0	=-117	+99.875	=210
						=-10.25	

As seen in the waveforms, the inputs are passed at negative edge of clock. Since the adder is pure combinational circuit, the results are obtained at the same negative edge of clock. We cannot see any delay because the waveform is just simulation of behavioral model. If we would do the Gate level simulation, then we will see the delay of the combinational circuit. This will require proper synthesis.

2. Simulations of adder with pipeline (4 stage pipeline)



The inputs are passed at negative edge of clock which. Since this is 4 stage pipelined, we get the output after 4 clock cycles. Same as the previous simulation, this does not account the gate delays.

Testcases and obtained results:

Input 1 (Decimal)	Input 2 (Decimal)	Result (Decimal)	Single-Precision-FP result in (Hexa-decimal)
98	169	267	5c2c
99	-89	10	4900
-45	79	34	5040
-283	-66	-349	dd74
0	0	0	0
0	-117	-117	d750
-110.125	99.875	-10.25	c920
110.875	99.125	210	5a90

4. Conclusion

Un-pipelined floating-point adder was designed and tested for the given test cases. Then, 4 stage pipelined floating point adder was designed and tested for the given test cases. The pipelined floating-point adder takes a total (8+4) cycles for 8 numbers to be added. The objective of the project was successfully achieved.

Additional work that can be done:

The project focused on just behavioral simulation of the design and not on synthesis and gate level simulations. In order to observe the gate level delay, one needs to perform synthesis and then do the gate level simulation on the netlist.

5. Bio

The author of this report is Deekshith, a 4th semester graduate student in Electrical Engineering student pursuing masters in San Jose State University. Deekshith has completed courses such as ASIC design, SOC design, Digital System Verification (UVM), Hardware design for AI/DSP. In his 4th semester, Deekshith is focusing on computer architecture and MS project.

Deekshith has successfully completed 6 months of internship in Toshiba Memory America Inc. as design engineer and has good industry knowledge of semiconductor industry. He will be graduating this spring and will join Marvell Semiconductor as design engineer upon graduation.

6. Appendix

1. Floating Point adder without pipeline.

```
// Code your design here
/********************** Arithmetic Floating Point Adder (without
Pipeline)**********************/

module FP_adder (final_sum_80, input_1_80, input_2_80, clock_80);

input [15:0] input_1_80, input_2_80;        //16 bit inputs
input clock_80;                             //clock input
output reg [15:0] final_sum_80;             //16 bit final result

reg sign_1_80, sign_2_80;                   //registers to store sign bits
reg [4:0] exponent_1_80, exponent_2_80;     //registers to store exponent bits
reg [9:0] mantissa_1_80, mantissa_2_80; //registers to store mantissa bits
reg [15:0] result_80;                       //registers to store end result

reg [4:0]exponent_max_80;                   //registers to store final exponent
reg [10:0] mantissa_1_11_80, mantissa_2_11_80;  //registers to store mantissa appended with 1 or
0
reg sign_80;                                //registers to store final sign
reg [11:0] sum_80;                          //registers to store final sum (with carry)

integer i_80;                               //for loop index integer

always @ (*) begin

    //to extract 1 bit sign from 32 bit input
    sign_1_80 = input_1_80[15];
    sign_2_80 = input_2_80[15];

    //to extract 8 bit exponent from 32 bit input
    exponent_1_80 = input_1_80[14:10];
    exponent_2_80 = input_2_80[14:10];

    //to extract 23 bit mantissa from 32 bit input
    mantissa_1_80 = input_1_80[9:0];
    mantissa_2_80 = input_2_80[9:0];

    //to assign result to final_sum
    final_sum_80 [15] = sign_80;
    final_sum_80[14:10] = exponent_max_80;
    final_sum_80 [9:0] = sum_80;
```

```
end


always @(*) begin

        // to compare exponents
        // if exponent1 is greater than or equal to exponent2
        if ( exponent_1_80 >= exponent_2_80) begin

            //choose exponent1
            exponent_max_80 = exponent_1_80;

            if (mantissa_2_80 == 10'b0000_0000_00) begin
                //append mantissa1 with 0
                mantissa_2_11_80 = {1'b0,mantissa_2_80};
            end else begin
                //append mantissa1 with 1 and shift left by the amount of difference in exponents
                mantissa_2_11_80 = {1'b1,mantissa_2_80} >> (exponent_1_80 - exponent_2_80);
            end
            if (mantissa_2_80 == 10'b0000_0000_00) begin
                //append mantissa2 with 0
                mantissa_1_11_80 = {1'b0,mantissa_1_80};
            end else begin
                //append mantissa2 with 1
                mantissa_1_11_80 = {1'b1,mantissa_1_80};
            end

        end else begin
            // if exponent2 is greater than exponent1
            //choose exponent2
            exponent_max_80 = exponent_2_80;

            if (mantissa_1_80 == 10'b0000_0000_0000) begin
                //append mantissa1 with 0
                mantissa_1_11_80 = {1'b0,mantissa_1_80};
            end else begin
                //append mantissa1 with 1 and shift left by the amount of difference in exponents
                mantissa_1_11_80 = {1'b1,mantissa_1_80} >> (exponent_2_80 - exponent_1_80);
            end
            if (mantissa_2_80 == 10'b0000_0000_00) begin
                //append mantissa2 with 0
                mantissa_2_11_80 = {1'b0,mantissa_2_80};
            end else begin
                //append mantissa2 with 1
```

```
                mantissa_2_11_80 = {1'b1,mantissa_2_80};
            end
        end

        //if both the numbers have same sign,
        if (sign_1_80 == sign_2_80) begin

            //retain the sign
            sign_80 = sign_1_80;
            //add the two mantissa
            sum_80 = mantissa_1_11_80 + mantissa_2_11_80;

        end else begin

            //if both the numbers have different sign,
            if (mantissa_1_11_80 > mantissa_2_11_80) begin

                //if mantissa1 > mantissa2, sign of the bigger number (mantissa 1)
                sign_80 = sign_1_80;
                // add mantissa1 and 2's compliment of mantissa2
                sum_80 = mantissa_1_11_80 + ((~mantissa_2_11_80) + 1'b1);

            end else begin

                //if mantissa2 > mantissa1, sign of the bigger number (mantissa 2)
                sign_80 = sign_2_80;
                // add mantissa2 and 2's compliment of mantissa1
                sum_80 = mantissa_2_11_80 + ((~mantissa_1_11_80) + 1'b1);

            end
        end

        //overflow occurs if carry is 1
        if(sign_1_80 == sign_2_80 && sum_80[11] == 1'b1) begin

            //shift mantissa sum to right by 1
            sum_80 = sum_80 >>1'b1;
            //and increment the exponent by 1
            exponent_max_80 = exponent_max_80 + 1'b1;

        end else begin

            //to normalize the mantissa of sum by shifting it left.
            for (i_80=0; ((i_80<=11) && (sum_80[10]) == 1'b0 && (sum_80 != 12'b0)); i_80=i_80+1)
begin
```

```
                sum_80 = sum_80 << 1'b1;
                exponent_max_80 = exponent_max_80 - 1'b1;
            end

        end
end

endmodule
```

2. Testbench for floating point adder.

```
// Code your testbench here
// or browse Examples
// Code your testbench here
// or browse Examples
module FP_adder_tb();

reg [15:0] input_1_80, input_2_80;
reg clock_80;
wire [15:0] sum_80;

//module call
FP_adder DUT (sum_80, input_1_80, input_2_80, clock_80);

//to generate dump files
initial begin
    $dumpfile ("FP_adder.vcd");
    $dumpvars (0);
end

//generate clock
initial begin
    clock_80 = 0;
    forever begin
        #5 clock_80 = ~clock_80;
    end
end

//to provide inputs
initial begin
  $monitor ($time, "sum=%b",
        sum_80);
```

```
    @(negedge clock_80);
        input_1_80 = 16'b0_10101_1000100000;
        input_2_80 = 16'b0_10110_0101001000;
    @(negedge clock_80);
        input_1_80 = 16'b0_10101_1000110000;
        input_2_80 = 16'b1_10101_0110010000;
    @(negedge clock_80);
        input_1_80 = 16'b1_10100_0110100000;
        input_2_80 = 16'b0_10101_0011110000;
    @(negedge clock_80);
        input_1_80 = 16'b1_10111_0001101100;
        input_2_80 = 16'b1_10101_0000100000;
    @(negedge clock_80);
        input_1_80 = 16'b0_00000_0000000000;
        input_2_80 = 16'b0_00000_0000000000;
    @(negedge clock_80);
        input_1_80 = 16'b0_00000_0000000000;
        input_2_80 = 16'b1_10101_1101010000;
    @(negedge clock_80);
        input_1_80 = 16'b1_10101_1011100010;
        input_2_80 = 16'b0_10101_1000111110;
    @(negedge clock_80);
        input_1_80 = 16'b0_10101_1011101110;
        input_2_80 = 16'b0_10101_1000110010;
  repeat (4) @(negedge clock_80);
     $finish;

end

endmodule
```

3. Pipelined floating-point adder with 4 stage pipeline.

```
/********************** Arithmetic Floating Point Adder (Pipeline)**********************/

module FP_adder (final_sum_80, input_1_80, input_2_80, clock_80);

input [15:0] input_1_80, input_2_80;        //16 bit inputs
input clock_80;                             //clock input
output reg [15:0] final_sum_80;             //16 bit final result

reg sign_1_80, sign_2_80;                   //registers to store sign bits
```

```
reg [4:0] exponent_1_80, exponent_2_80;     //registers to store exponent bits
reg [9:0] mantissa_1_80, mantissa_2_80;     //registers to store mantissa bits
reg [15:0] result_80;                       //registers to store end result

reg [4:0]exponent_max_80;                   //registers to store final exponent
reg [10:0] mantissa_1_11_80, mantissa_2_11_80;  //registers to store mantissa appended with 1 or
0
reg sign_80;                                //registers to store final sign
reg [11:0] sum_80;                          //registers to store final sum (with carry)

integer i_80;                               //for loop index integer

reg [10:0] mantissa1_1,mantissa2_1;         // pipeline registers
reg [4:0] exponent1,exponent2,exponent3,exponent4;
reg sign1_2,sign1_1,sign2_1,sign2_2;
reg [11:0] sum_80_2,sum_80_3,sum_80_4;
reg sign_80_2,sign_80_3,sign_80_4;

always @ (*) begin

    //to extract 1 bit sign from 32 bit input
    sign_1_80 = input_1_80[15];
    sign_2_80 = input_2_80[15];

    //to extract 8 bit exponent from 32 bit input
    exponent_1_80 = input_1_80[14:10];
    exponent_2_80 = input_2_80[14:10];

    //to extract 23 bit mantissa from 32 bit input
    mantissa_1_80 = input_1_80[9:0];
    mantissa_2_80 = input_2_80[9:0];

    //to assign result to final_sum
    final_sum_80 [15] = sign_80_4;
    final_sum_80[14:10] = exponent4;
    final_sum_80 [9:0] = sum_80_4;

end


always @(*) begin

        // to compare exponents
        // if exponent1 is greater than or equal to exponent2
        if ( exponent_1_80 >= exponent_2_80) begin
```

```
            //choose exponent1
            exponent_max_80 = exponent_1_80;

            if (mantissa_2_80 == 10'b0000_0000_00) begin
                //append mantissa1 with 0
                mantissa_2_11_80 = {1'b0,mantissa_2_80};
            end else begin
                //append mantissa1 with 1 and shift left by the amount of difference in exponents
                mantissa_2_11_80 = {1'b1,mantissa_2_80} >> (exponent_1_80 - exponent_2_80);
            end
            if (mantissa_2_80 == 10'b0000_0000_00) begin
                //append mantissa2 with 0
                mantissa_1_11_80 = {1'b0,mantissa_1_80};
            end else begin
                //append mantissa2 with 1
                mantissa_1_11_80 = {1'b1,mantissa_1_80};
            end

        end else begin
            // if exponent2 is greater than exponent1
            //choose exponent2
            exponent_max_80 = exponent_2_80;

            if (mantissa_1_80 == 10'b0000_0000_0000) begin
                //append mantissa1 with 0
                mantissa_1_11_80 = {1'b0,mantissa_1_80};
            end else begin
                //append mantissa1 with 1 and shift left by the amount of difference in exponents
                mantissa_1_11_80 = {1'b1,mantissa_1_80} >> (exponent_2_80 - exponent_1_80);
            end
            if (mantissa_2_80 == 10'b0000_0000_00) begin
                //append mantissa2 with 0
                mantissa_2_11_80 = {1'b0,mantissa_2_80};
            end else begin
                //append mantissa2 with 1
                mantissa_2_11_80 = {1'b1,mantissa_2_80};
            end
        end
end


// first stage pipeline
always @ (posedge clock_80) begin
 mantissa1_1 <= #1 mantissa_1_11_80;
```

```
 mantissa2_1 <= #1 mantissa_2_11_80;
 exponent1 <= #1 exponent_max_80;
 sign1_1 <= #1 sign_1_80;
 sign2_1 <= #1 sign_2_80;
end

always @(*) begin
        //if both the numbers have same sign,
        if (sign1_1 == sign2_1) begin

            //retain the sign
            sign_80 = sign1_1;
            //add the two mantissa
            sum_80 = mantissa1_1 + mantissa2_1;

        end else begin

            //if both the numbers have different sign,
            if (mantissa1_1 > mantissa2_1) begin

                //if mantissa1 > mantissa2, sign of the bigger number (mantissa 1)
                sign_80 = sign1_1;
                // add mantissa1 and 2's compliment of mantissa2
                sum_80 = mantissa1_1 + ((~mantissa2_1) + 1'b1);

            end else begin

                //if mantissa2 > mantissa1, sign of the bigger number (mantissa 2)
                sign_80 = sign2_1;
                // add mantissa2 and 2's compliment of mantissa1
                sum_80 = mantissa2_1 + ((~mantissa1_1) + 1'b1);

            end
        end
end


// 2nd stage pipeline
always@ (posedge clock_80) begin
sum_80_2 <= #1 sum_80;
sign1_2 <= #1 sign1_1;
sign2_2 <= #1 sign2_1;
exponent2 <= #1 exponent1;
sign_80_2 <= #1 sign_80;
end
```

```
always @(*) begin
        //overflow occurs if carry is 1
        if(sign1_2 == sign2_2 && sum_80_2[11] == 1'b1) begin

            //shift mantissa sum to right by 1
            sum_80_2 = sum_80_2 >>1'b1;
            //and increment the exponent by 1
            exponent2 = exponent2 + 1'b1;

        end else begin

            //to normalize the mantissa of sum by shifting it left.
            for (i_80=0; ((i_80<=11) && (sum_80_2[10]) == 1'b0 && (sum_80_2 != 12'b0));
i_80=i_80+1) begin
                sum_80_2 = sum_80_2 << 1'b1;
                exponent2 = exponent2 - 1'b1;
            end

        end
end
// 3rd stage pipeline
always @ (posedge clock_80) begin
sum_80_3 <= #1 sum_80_2;
exponent3 <= #1 exponent2;
sign_80_3 <= #1 sign_80_2;
end
//4th stage pipeline
always @ (posedge clock_80) begin
sum_80_4 <= #1 sum_80_3;
exponent4 <= #1 exponent3;
sign_80_4 <= #1 sign_80_3;
end

endmodule
```

4. Compilation report for floating point adder without pipeline.

```
[2019-03-07 14:58:49 EST] vcs -licqueue '-timescale=1ns/1ns' '+vcs+flush+all' '+warn=all' '-
sverilog' design.sv testbench.sv  && ./simv +vcs+lic+wait

Warning-[LNX_OS_VERUN] Unsupported Linux version
  Linux version 'CentOS Linux release 7.1.1503 (Core) ' is not supported on
  'x86_64' officially, assuming linux compatibility by default. Set
  VCS_ARCH_OVERRIDE to linux or suse32 to override.
  Please refer to release notes for information on supported platforms.


Warning-[LINX_KRNL] Unsupported Linux kernel
  Linux kernel '3.13.0-71-generic' is not supported.
  Supported versions are 2.4* or 2.6*.

                        Chronologic VCS (TM)
         Version J-2014.12-SP1-1 -- Thu Mar  7 19:58:50 2019
               Copyright (c) 1991-2014 by Synopsys Inc.
                        ALL RIGHTS RESERVED

This program is proprietary and confidential information of Synopsys Inc.
and may be used and disclosed only as authorized in a license agreement
controlling such use and disclosure.

Parsing design file 'design.sv'

Warning-[TMBIN] Too many bits in Based Number
design.sv, 73
  The specified width is '10' bits, actually got '12' bits.
  The offending number is : '0000_0000_0000'.

Parsing design file 'testbench.sv'
Top Level Modules:
       FP_adder_tb
TimeScale is 1 ns / 1 ns
Starting vcs inline pass...
2 modules and 0 UDP read.
recompiling module FP_adder_tb
Both modules done.
rm -f _csrc*.so linux_scvhdl_*.so pre_vcsobj_*.so share_vcsobj_*.so
ld -m elf_i386 -shared  -o .//../simv.daidir//_csrc0.so amcQwB.o
rm -f _csrc0.so
if [ -x ../simv ]; then chmod -x ../simv; fi
g++  -o ../simv  -m32 -m32   -Wl,-rpath-link=./ -Wl,-rpath='$ORIGIN'/simv.daidir/ -Wl,-
rpath='$ORIGIN'/simv.daidir//scsim.db.dir    _308_archive_1.so _csrc0.so  SIM_l.o  _csrc0.so
rmapats_mop.o rmapats.o rmar.o  rmar_llvm_0_1.o rmar_llvm_0_0.o
/apps/vcsmx/linux/lib/libzerosoft_rt_stubs.so /apps/vcsmx/linux/lib/libvirsim.so
/apps/vcsmx/linux/lib/liberrorinf.so /apps/vcsmx/linux/lib/libsnpsmalloc.so
/apps/vcsmx/linux/lib/libvcsnew.so /apps/vcsmx/linux/lib/libuclinative.so   -Wl,-whole-
archive /apps/vcsmx/linux/lib/libvcsucli.so -Wl,-no-whole-archive
/apps/vcsmx/linux/lib/vcs_save_restore_new.o /apps/vcsmx/linux/lib/ctype-stubs_32.a -ldl  -lc
-lm -lpthread -ldl
../simv up to date
CPU time: .116 seconds to compile + .215 seconds to elab + .192 seconds to link
Chronologic VCS simulator copyright 1991-2014
Contains Synopsys proprietary information.
Compiler version J-2014.12-SP1-1; Runtime version J-2014.12-SP1-1;  Mar  7 19:58 2019
                   0sum=xxxxxxxxxxxxxxx
                  10sum=0101110000101100
                  20sum=0100100100000000
                  30sum=0101000001000000
                  40sum=1101110101110100
```

```
                50sum=0000000000000000
                60sum=1101011101010000
                70sum=1100100100100000
                80sum=0101101010010000
$finish called from file "testbench.sv", line 58.
$finish at simulation time                  120
           V C S   S i m u l a t i o n   R e p o r t
Time: 120 ns
CPU Time:      0.240 seconds;       Data structure size:   0.0Mb
Thu Mar  7 19:58:51 2019
Finding VCD file...
./FP_adder.vcd
[2019-03-07 14:58:51 EST] Opening EPWave...
Done
```

5. Compilation report for floating point adder with pipeline.

```
[2019-03-07 14:53:48 EST] vcs -licqueue '-timescale=1ns/1ns' '+vcs+flush+all' '+warn=all' '-
sverilog' design.sv testbench.sv  && ./simv +vcs+lic+wait

Warning-[LNX_OS_VERUN] Unsupported Linux version
  Linux version 'CentOS Linux release 7.1.1503 (Core) ' is not supported on
  'x86_64' officially, assuming linux compatibility by default. Set
  VCS_ARCH_OVERRIDE to linux or suse32 to override.
  Please refer to release notes for information on supported platforms.


Warning-[LINX_KRNL] Unsupported Linux kernel
  Linux kernel '3.13.0-71-generic' is not supported.
  Supported versions are 2.4* or 2.6*.

                        Chronologic VCS (TM)
         Version J-2014.12-SP1-1 -- Thu Mar  7 19:53:49 2019
               Copyright (c) 1991-2014 by Synopsys Inc.
                        ALL RIGHTS RESERVED

This program is proprietary and confidential information of Synopsys Inc.
and may be used and disclosed only as authorized in a license agreement
controlling such use and disclosure.

Parsing design file 'design.sv'

Warning-[TMBIN] Too many bits in Based Number
design.sv, 78
  The specified width is '10' bits, actually got '12' bits.
  The offending number is : '0000_0000_0000'.

Parsing design file 'testbench.sv'
Top Level Modules:
       FP_adder_tb
TimeScale is 1 ns / 1 ns
Starting vcs inline pass...
2 modules and 0 UDP read.
recompiling module FP_adder_tb
Both modules done.
rm -f _csrc*.so linux_scvhdl_*.so pre_vcsobj_*.so share_vcsobj_*.so
ld -m elf_i386 -shared  -o .//../simv.daidir//_csrc0.so amcQwB.o
rm -f _csrc0.so
if [ -x ../simv ]; then chmod -x ../simv; fi
g++  -o ../simv  -m32 -m32   -Wl,-rpath-link=./ -Wl,-rpath='$ORIGIN'/simv.daidir/ -Wl,-
rpath='$ORIGIN'/simv.daidir//scsim.db.dir    _308_archive_1.so _csrc0.so  SIM_l.o  _csrc0.so
```

```
rmapats_mop.o rmapats.o rmar.o  rmar_llvm_0_1.o rmar_llvm_0_0.o
/apps/vcsmx/linux/lib/libzerosoft_rt_stubs.so /apps/vcsmx/linux/lib/libvirsim.so
/apps/vcsmx/linux/lib/liberrorinf.so /apps/vcsmx/linux/lib/libsnpsmalloc.so
/apps/vcsmx/linux/lib/libvcsnew.so /apps/vcsmx/linux/lib/libuclinative.so   -Wl,-whole-
archive /apps/vcsmx/linux/lib/libvcsucli.so -Wl,-no-whole-archive
/apps/vcsmx/linux/lib/vcs_save_restore_new.o /apps/vcsmx/linux/lib/ctype-stubs_32.a -ldl  -lc
-lm -lpthread -ldl
../simv up to date
CPU time: .123 seconds to compile + .208 seconds to elab + .187 seconds to link
Chronologic VCS simulator copyright 1991-2014
Contains Synopsys proprietary information.
Compiler version J-2014.12-SP1-1; Runtime version J-2014.12-SP1-1;  Mar  7 19:53 2019
                  0sum=xxxxxxxxxxxxxxxx
                 46sum=0101110000101100
                 56sum=0100100100000000
                 66sum=0101000001000000
                 76sum=1101110101110100
                 86sum=0000000000000000
                 96sum=1101011101010000
                106sum=1100100100100000
                116sum=0101101010010000
$finish called from file "testbench.sv", line 56.
$finish at simulation time                  120
           V C S   S i m u l a t i o n   R e p o r t
Time: 120 ns
CPU Time:      0.240 seconds;       Data structure size:   0.0Mb
Thu Mar  7 19:53:50 2019
Finding VCD file...
./FP_adder.vcd
[2019-03-07 14:53:50 EST] Opening EPWave...
Done
```