

DSD

PROJECT Report

Lecturer: Dr. Ronny Veljanovski

PROJECT:

FLOATING POINT ADDITION & SUBTRACTION

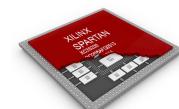
Student:

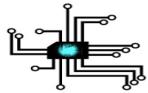
IBRAHIM HAZMI 3772584

REENA ASHANTI 3731182

Tutor:

David Fitrio

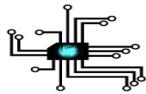




CONTENTS

<i>Introduction-----</i>	<i>3</i>
<i>Materials and Methods -----</i>	<i>3</i>
<i>IEEE754 format and Arithmetic Operations-----</i>	<i>3-14</i>
<i>Simulation Results-----</i>	<i>15-17</i>
<i>Synthesis and Fitting Results-----</i>	<i>18-20</i>
<i>Hardware demonstration Results-----</i>	<i>21-22</i>
<i>Conclusion & References -----</i>	<i>23</i>
<i>Appendix -----</i>	<i>24-42</i>





I. INTRODUCTION

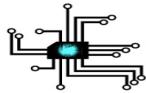
Floating-point numbers are widely adopted in many applications due to their dynamic representation and capabilities. Floating-point representation is able to retain its resolution and accuracy compared to fixed-point representations. IEEE specified the IEEE754 standard for floating-point representation in 1985. Based on this standard, floating-point representation for digital systems should be platform-independent and data are interchanged among different digital systems. ALU is a block in a microprocessor that handles arithmetic operations. It always performs computation of floating-point operations. Some CPUs such as AMD Athlon have more than one floating point unit that handles floating point operations. The AMD Athlon processor boasts an FPU count of three, which in an indirect way reiterates the importance of the inclusion of floating-point operations in a microprocessor's functionality.

II. MATERIALS AND METHODS

This report describes the implementation of floating-point ALU in design using VHDL. In order to design the floating-point ALU, a number of sub-objectives that support and supplement the main objective are defined. The sub-objectives are to design a 32-bit (single precision) floating-point ALU operating on the IEEE 754 standard floating-point representations, supporting the four basic arithmetic operations: addition, subtraction, multiplication and division. Second sub-objective is to model the behaviour of the ALU design using VHDL.

This paper discusses the main two operations: addition and subtraction.





III. IEEE754 FORMAT AND ARITHMETIC OPERATIONS:

- The IEEE754 format:

The Bit 31 is the sign bit, the next eight bits are the exponent bits (EXP) and the rest are the mantissas (FN, 23-bit).

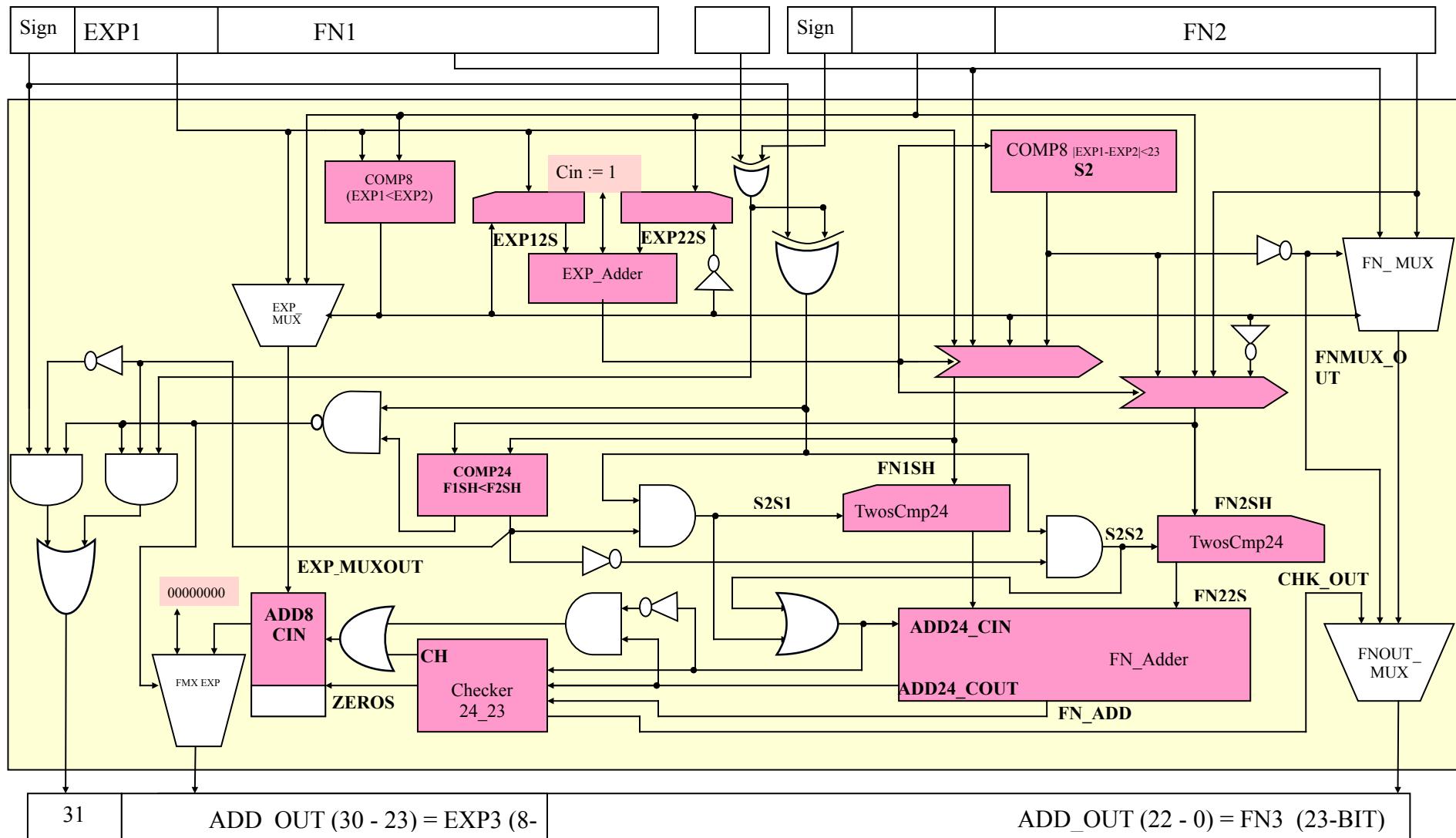
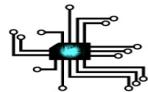
For all ordinary numbers the exponent field represents the exponent of a number plus a bias of 127. A sign bit of 1 indicates a negative number. The highest bit of the significand is a hidden bit and always assumed to be 1, therefore, the representation only encodes the lower 23 bits. This 24-bit number is considered as a normalized number, since the digit to the left of the binary point is 1.

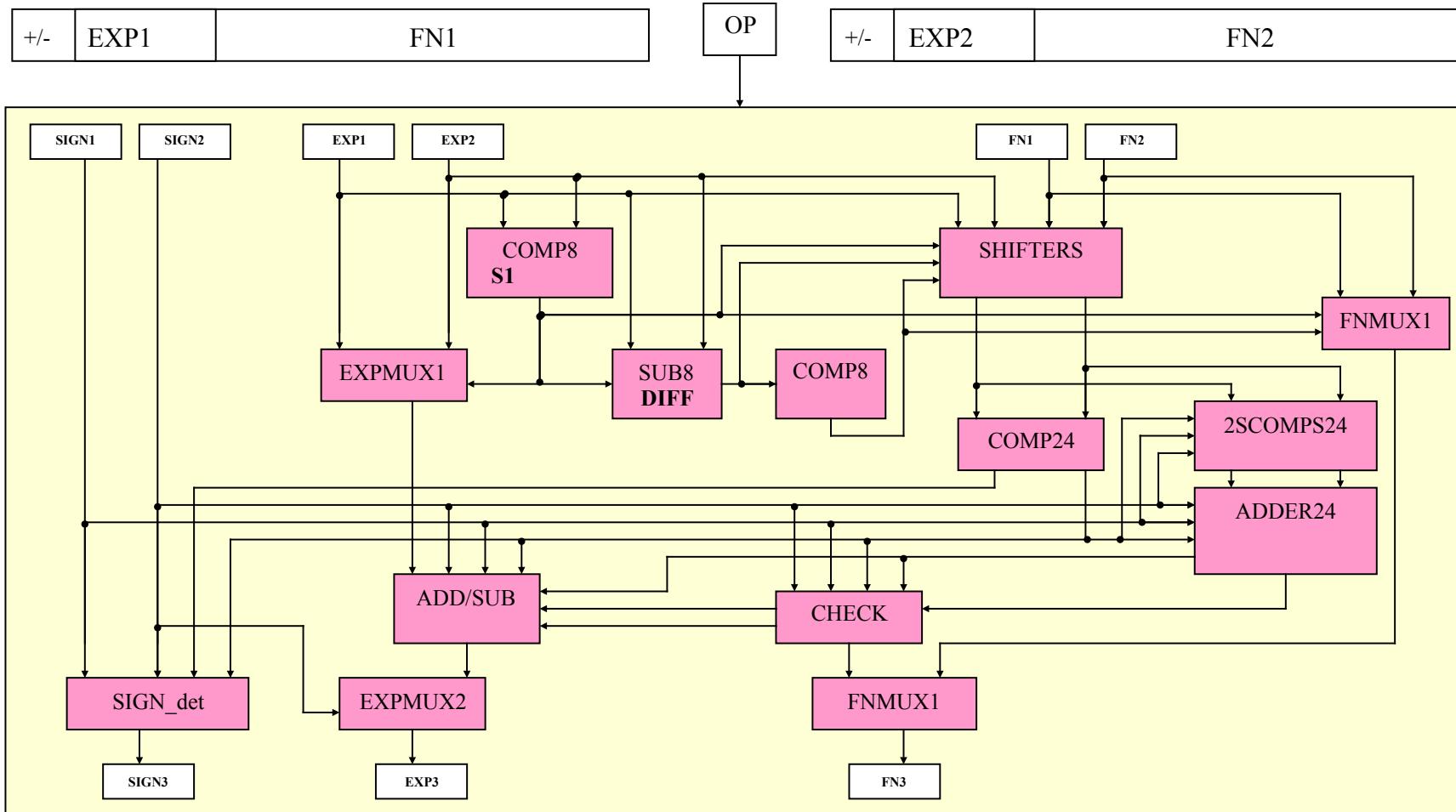
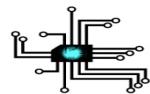
- The IEEE754 Addition/Subtraction:

- To add two numbers in IEEE754 format, the exponent parts should be equal. So, we need to check them first in order to align them if they are different. To do so, we need to compare the two exponents to determine the bigger one which is the possible result for the final exponent and to control shift and 2's complement for the mantissas.
- Then, we check the absolute difference between the two exponents ($|EXP1 - EXP2|$), if the difference greater than or equal 23, the bigger number (32-bit) will be the result. Otherwise, shift the small number (24-bit including "1.") by the amount of the difference in order to align the two numbers.
- Subsequently, we compare the two (24-bit) numbers after the shift and generate the 2's complement for the smaller number. Then, we add/ subtract the two numbers using the 24 bit adder/subtractor.
- Finally, we normalize the number by checking the 24-bit adder/subtracter output, if it is less than 1 ($\neq 1.---$) then shift it to be (1.--- - final result of the mantissas). Then, subtract the shifting amount from the 1st possible exponent.

31	30	23 22	0
Sign	EXP	FN	

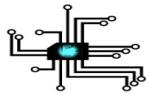






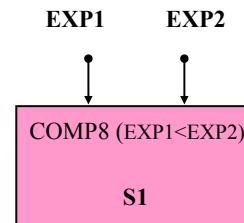
31	ADD_OUT (30 - 23) = EXP3 (8-BIT)	ADD_OUT (22 - 0) = FN3 (23-BIT)
----	----------------------------------	---------------------------------





IV. A complete functional description of the system:

1. The 8-bit Comparator:

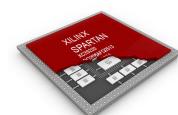


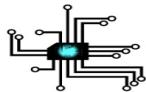
The comparison between two numbers is an operation that determines if one number is greater or less than, or equal the other number. In our circuit, we have designed the 8-bit comparator to determine if the exponent of the first number is less than the exponent of the second number or not. To do this operation we use the following equations:

$$X(i) = EXP1(i) \text{ XNOR } EXP2(i); \text{ Where } i \text{ from 0 to 7}$$

$$\begin{aligned} S = & ((\text{NOT } EXP1(7)) \text{ AND } EXP2(7)) \text{ OR } ((X(7) \text{ AND } (\text{NOT } EXP1(6)) \text{ AND } EXP2(6)) \\ & \text{ OR } ((X(7) \text{ AND } X(6) \text{ AND } (\text{NOT } EXP1(5)) \text{ AND } EXP2(5)) \\ & \text{ OR } ((X(7) \text{ AND } X(6) \text{ AND } X(5) \text{ AND } (\text{NOT } EXP1(4)) \text{ AND } EXP2(4)) \\ & \text{ OR } ((X(7) \text{ AND } X(6) \text{ AND } X(5) \text{ AND } X(4) \text{ AND } (\text{NOT } EXP1(3)) \text{ AND } EXP2(3)) \\ & \text{ OR } ((X(7) \text{ AND } X(6) \text{ AND } X(5) \text{ AND } X(4) \text{ AND } X(3) \text{ AND } (\text{NOT } EXP1(2)) \text{ AND } EXP2(2)) \\ & \text{ OR } ((X(7) \text{ AND } X(6) \text{ AND } X(5) \text{ AND } X(4) \text{ AND } X(3) \text{ AND } X(2) \text{ AND } (\text{NOT } EXP1(1)) \text{ AND } EXP2(1)) \\ & \text{ OR } ((X(7) \text{ AND } X(6) \text{ AND } X(5) \text{ AND } X(4) \text{ AND } X(3) \text{ AND } X(2) \text{ AND } X(1) \text{ AND } (\text{NOT } EXP1(0)) \text{ AND } EXP2(0)); \end{aligned}$$

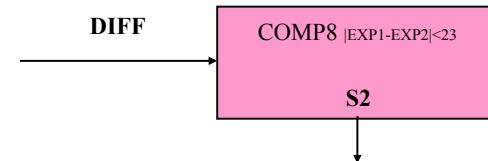
If $S = 1 \rightarrow EXP1 < EXP2$





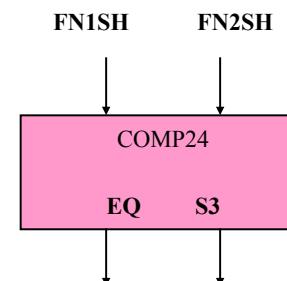
We use this comparator also to check if the difference between the exponents is less than 23 or not.

If $S = 1 \rightarrow |EXP1 - EXP2| < 23$

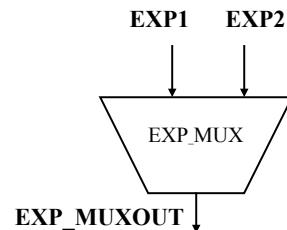


We use another comparator (24-bit) to compare the two mantissas after adding the hidden One (1---) and performing the shift. In this comparator, we have two outputs, one output is the same as the 8-bit comparators (S) and the other one is the EQ signal in which:

$$EQ = (X(23) \text{ AND } X(22) \text{ AND } \dots \text{ X}(0))$$

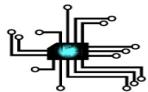


2. The Exponent Multiplexer:

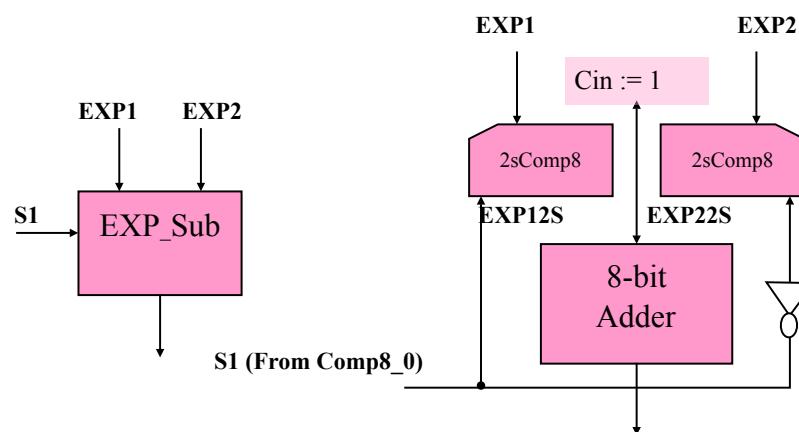


A digital multiplexer is a combinational circuit that selects binary information from one of many input lines. This MUX has two inputs (EXP1 and EXP2) and is controlled by S1. If $S1 = 1$ ($EXP1 < EXP2$) then EXP2 is selected as the output, otherwise EXP1 is the output of this MUX.





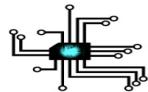
3. The 8-bit Subtractor:



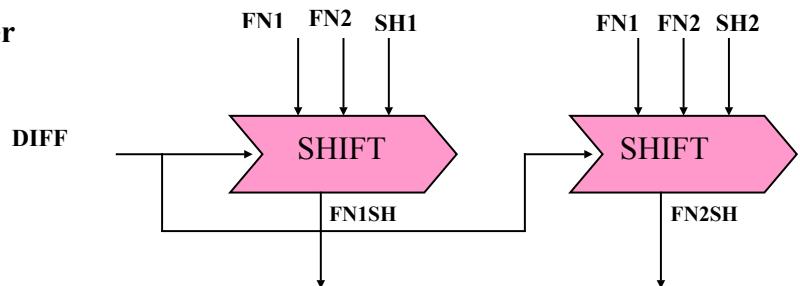
To subtract two numbers we need to get the 2's complement for the smaller number first, then add it to the greater one with a carry-in = 1, then ignore the carry out.

The 2sComp8 is the part which does the 2's complement operation for the smaller number, which is determined by signal the output signal of the comparator Comp8 (S1). The result is the difference between the two exponents, i.e., the number of shifts we need to perform for the smaller number in order to align the two exponents allowing us to add their mantissas. If S1 = 1, then the first 2sComp will work by xorring each bit of EXP1 with 1 which will give us the 1's complement, since xorring any number with 1 results in the inverse of the number. On the second 2sComp, S1' = 0 (because of the inverter), so it will xor each bit of EXP2 with 0, which results in the same number (EXP2). In the 8-bit adder, Cin = 1 always, and this results in the addition of EXP2 with the 2's complement of EXP1. By ignoring the carry out of the adder (the last full adder has no carry out), we can get the difference between the two exponents (EXP2 – EXP1 in this case).



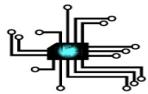


4. 24-bit Shifter

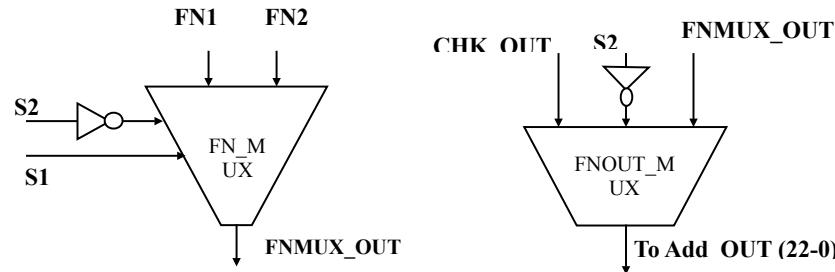


DIFF	Shifter Contents																								
	1
0	1
1	0	1
2	0	0	1
3	0	0	0	1
...
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
>23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Output is selected by DIFF (the out put of the subtracter).																									

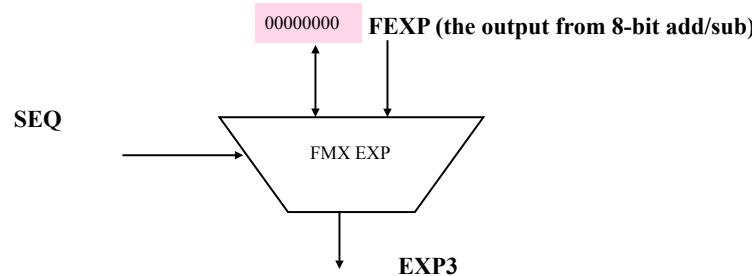
If $\text{DIFF} \geq 23$, i.e., FN1 might be greater than FN2 before shifting, we don't shift to avoid the occurrence of a fault in the 24-bit comparator, which affects the sign bit result. If $S2 = 1$, then the shifter will work and shift right one of the two 24-bit numbers depending on S1. If $S1 = 1$ ($\text{EXP1} < \text{EXP2}$) and $\text{DIFF} < 23$, then the shift will occur in FN1 (the smaller) rather than FN2 (the bigger). If $S1 = 0$ then the shift will occur in FN2 since S1 is inverted as an input to the second shifter. The number of shifts is determined by DIFF.



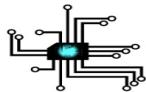
5. Mantissas First and Final Multiplexers:



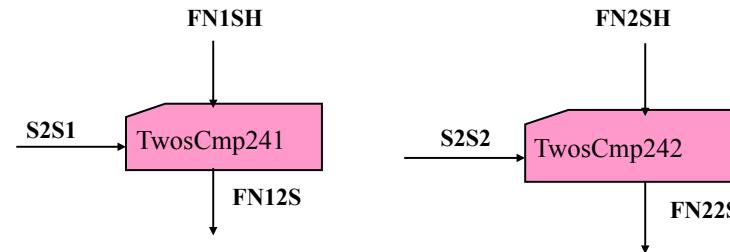
This multiplexer has two inputs, FN1 and FN2, and is controlled by S1 and S2', where S2' shows that DIFF is greater than 23, which means that the result is the bigger number without doing any further operations. S1 determines which number is the greater by checking their exponent parts as mentioned before. The result will go to the final FN_MUX which selects between this result and addition operation result depending on S2' again. If S2' = 1, then the final 23-bit (add_out(22 down to 0)) is FN_MUX output, otherwise the output from the checker circuit (CHK_OUT) will be the final result.



Another Multiplexer is designed in case that the two numbers have the same value but different sign, which will result in zero by xoring the two sign bit and NANAD the result of them with EQ (the comparator output which means the two numbers are equal).



6. The 24-bit Tows Complement Circuit:



This circuit is similar to the 8-bit 2's complement circuit, used in the 8-bit subtracter, but it is controlled by two signals rather than one. These signals are the output of sign bits xor and S3 (an output from the 24-bit comparator). If the output of sign bits xor is 1, then the signs are different and the operation is subtraction rather than addition. Therefore, we need to get the 2's complement for the smaller number (determined by Comp24 output – S3). Otherwise (xor = 0), no 2's complement circuit will work. The equations of the control circuit of the two's complement circuits are as follows:

$$S1S2 = (N1(31) \text{ XOR } N2(31)) \text{ AND } S3$$

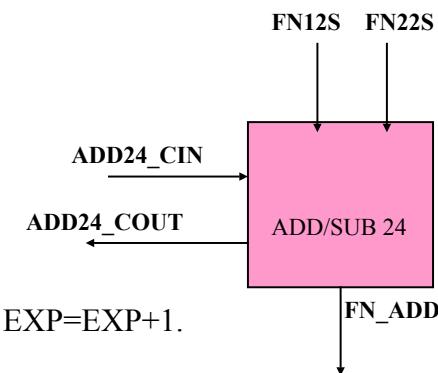
$$S2S2 = (N1(31) \text{ XOR } N2(31)) \text{ AND } S3'$$

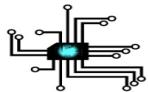
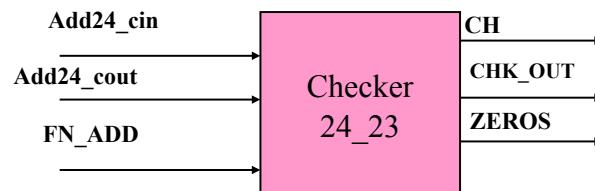
7. The 24-bit Adder/Subtractor:

If $Cin_{24} = 0$, then normal addition occurs and the value of $Cout_{24}$ determine whether: $EXP=EXP$ or $EXP=EXP+1$.

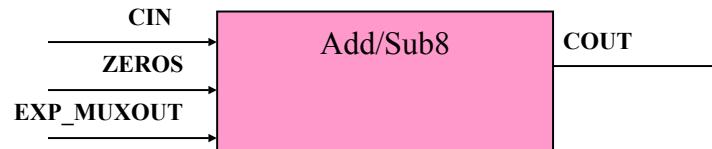
Otherwise, the subtraction occurs and we ignore the value of $Cout_{24}$.

The output of this circuit is going to be one of the inputs to the next circuit (The Checker).

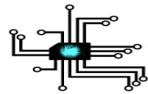


**8. The 24-bit to 23-bit Checker:**

The idea behind this checker is to check the output from the 24-bit adder/subtractor circuit, if there is any special case of normalization that needs to be considered. So, we might need to do some shifts and add to or subtract from the output of the EXP_MUX. The circuit is controlled by two signals Cin_24 and Cout_24 and has three outputs; 23-bit number CHK_OUT, CH and ZEROS (8-bit). When the checker normalizes a number, it calculates the number of performed shifts, which results in the output "ZEROS". The complement of the number of shifts which has been done in the checker ("ZEROS") is subtracted in the 8-bit adder/subtractor from the output of the EXP_MUX. The signal CH determines if "ZEROS" greater than '0' in order to perform subtraction, since CH is connected to the carry-in of the 8-bit adder/subtractor. Actually the normalization happens when Cin_24 = 1, i.e. when the operation at the 24-bit adder/subtractor is subtraction.

9. The 8-bit Adder/Subtractor:

If Cin_24 = 0 and Cout_24 = 1, then the addition of 1 to the output of the EXP_MUX will occur ($\text{EXP} = \text{EXP} + 1$), but if Cout_24 = 0, nothing will happen to the output of the EXP_MUX ($\text{EXP} = \text{EXP}$). If "ZEROS" is not zero and, of course, CH=1, then the subtraction will occur, since Add8_Cin = CH = 1 (to compute the 2's complement for "ZEROS").



How can we determine the sign of the addition?

In each number, the most significant bit is responsible to specify the sign of that number, so, these bits are XORed and the output is NANDed with EQ signal (the output from FN comparator, which means the mantissas are equal). Then, each number is ANDed with SEQ and a form of S3 as follows:

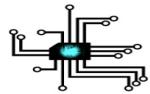
$$\text{SEQ} = ((\text{N1}(31) \text{ XOR } \text{N2}(31)) \text{ NANAD } \text{EQ}) \quad \text{equation 1}$$

$$\text{M} = \text{SEQ AND S3 AND N2}(31) \quad \text{equation 2}$$

$$\text{R} = \text{SEQ AND S3' AND N1}(31) \quad \text{equation 3}$$

From equation 1,2 and 3

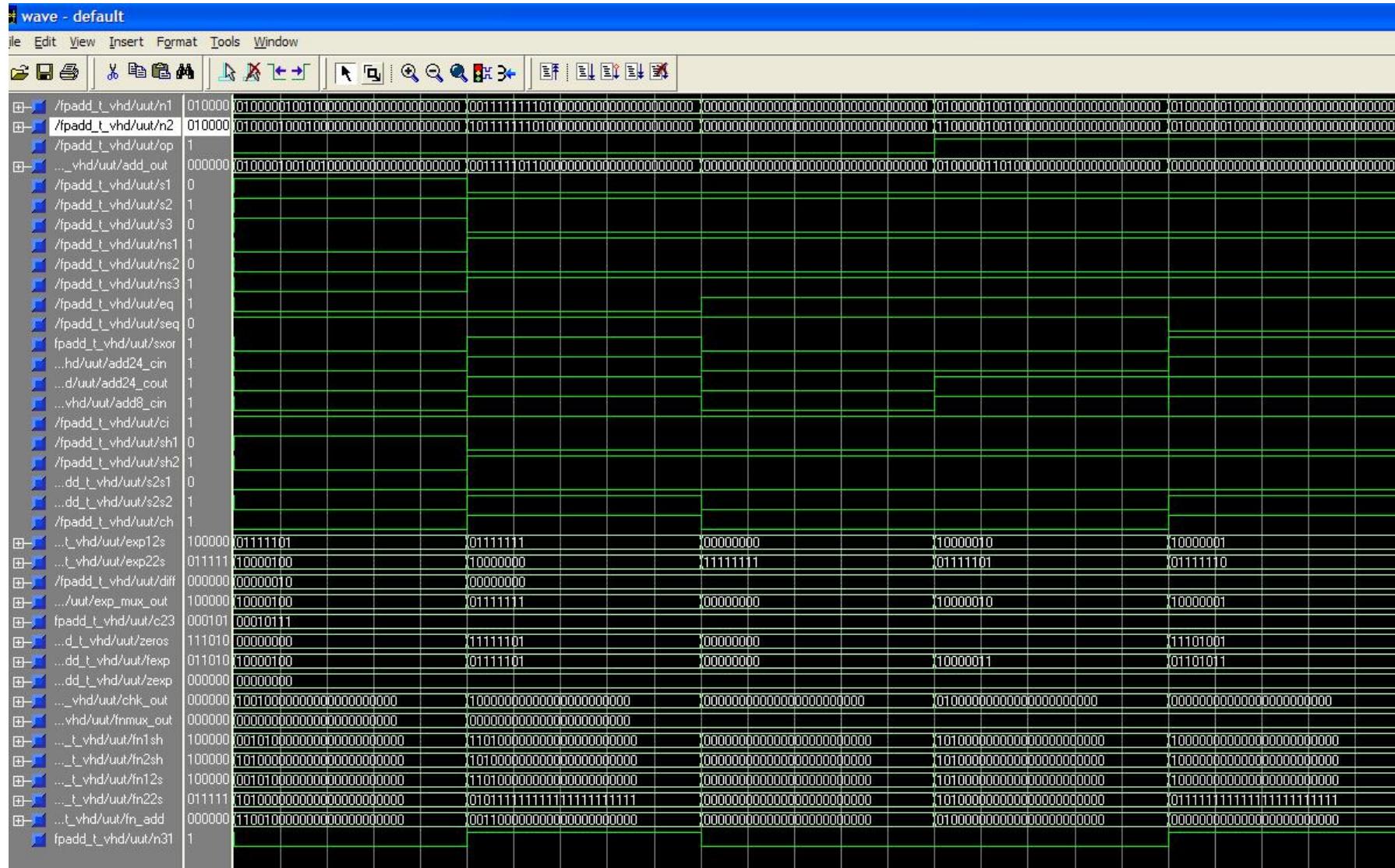
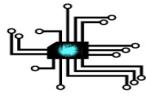
The sign bit: $\text{ADD_OUT}(31) = \text{M OR R}$

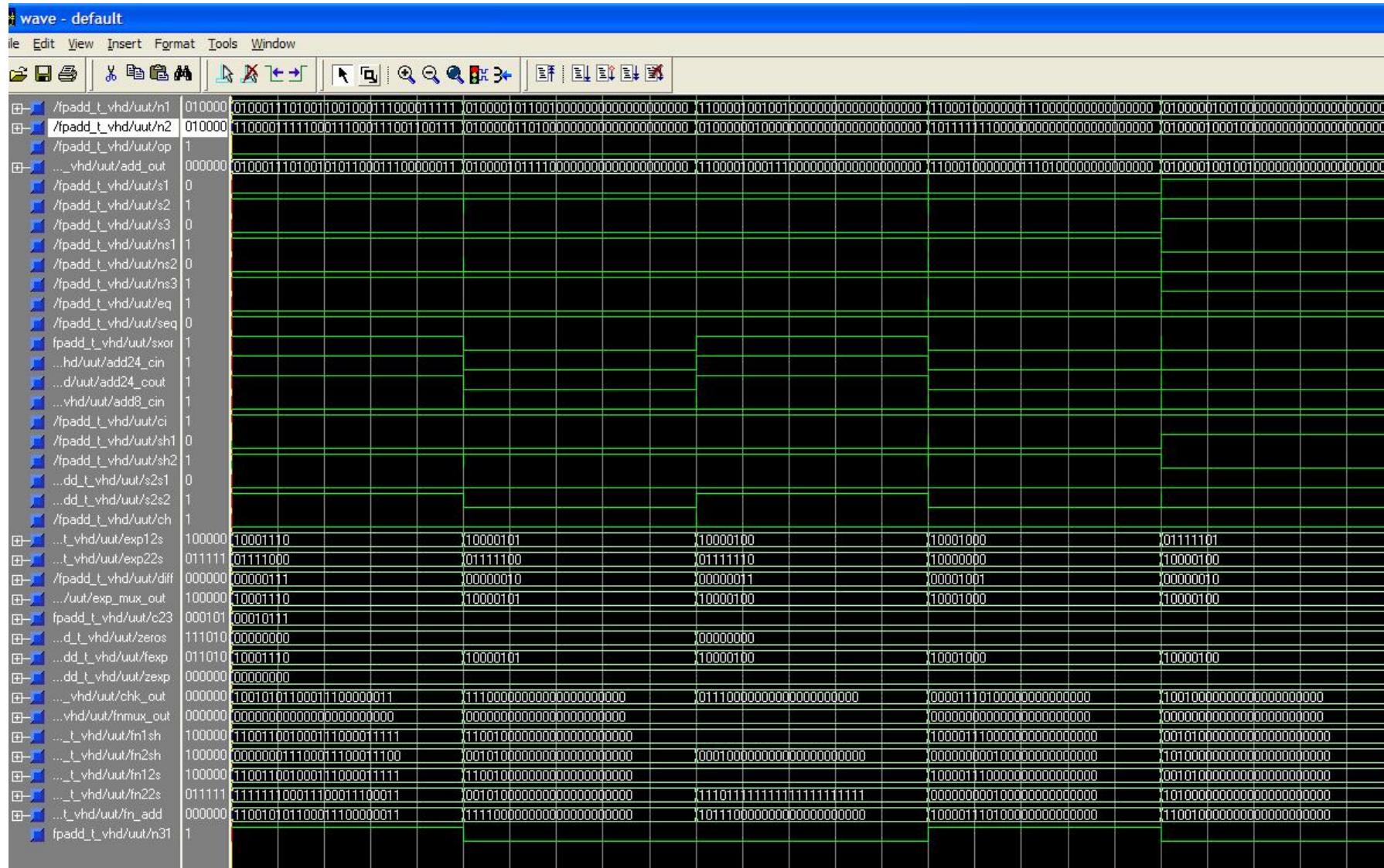
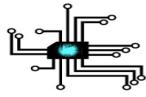


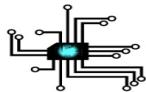
V. SIMULATION Results

The design was verified through simulation, which is done in a bottom-up fashion. Small modules are simulated in separate testbenches before they are integrated and tested as a whole. The arithmetic operations available in the design are tested with the same inputs. The sequence of operations done in the simulation is addition and subtraction. The results of operation on the test vectors are manually computed and are referred to as expected result, later compared with the simulation results. The generated outputs are in hexadecimal form. All hexadecimal values are converted back to the decimal format.

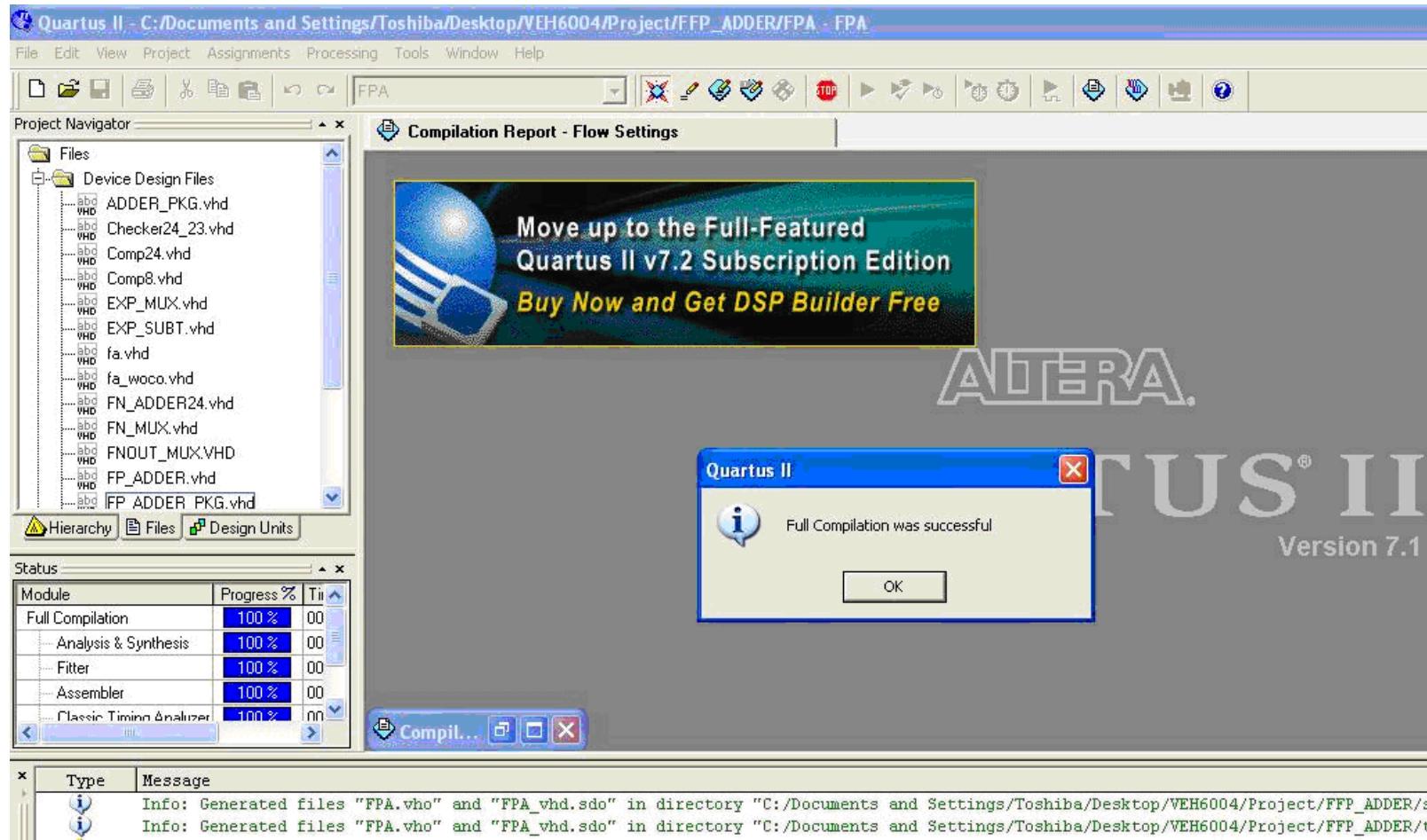
The next two pages are the results of our simulation using ModelSim Software.

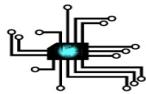






VI. Synthesis and Fitting Results





II - C:/Documents and Settings/Toshiba/Desktop/VEH6004/Project/FP_ADDER/FPA - FPA - [Compilation Report - Assembler Summary]

File Edit View Project Assignments Processing Tools Window Help

ct Navigator

Files

- Device Design Files
 - ADDER_PKG.vhd
 - Checker24_23.vhd
 - Comp24.vhd
 - Comp8.vhd
 - EXP_MUX.vhd
 - EXP_SUBT.vhd
 - fa.vhd
 - fa_woco.vhd
 - FN_ADDER24.vhd
 - FN_MUX.vhd
 - FNOUT_MUX.VHD
 - FP_ADDER.vhd
 - FP_ADDER_PKG.vhd

Hierarchy Files Design Units

stage	Progress %	Time
Compilation	100 %	00:00
Analysis & Synthesis	100 %	00:00
Filter	100 %	00:00
Assembler	100 %	00:00
Classic Timing Analyzer	100 %	00:00

Type Message

Info: Generated files "FPA.vho" and "FPA.vhd.sdo" in directory "C:/Documents and Settings/Toshiba/Desktop/VEH6004/Project/FP_ADDER/FPA - FPA"

Analysis & Synthesis Summary

Compilation Report - Analysis & Synthesis Summary

Analysis & Synthesis Status

Succes

Quartus II Version 7.1 Bi

Revision Name FPA

Top-level Entity Name FP_ADDER

Family FLEX10K

Total logic elements 783

Total pins 96

Total memory bits 0

Device Design Files

- ADDER_PKG.vhd
- Checker24_23.vhd
- Comp24.vhd
- Comp8.vhd
- EXP_MUX.vhd
- EXP_SUBT.vhd
- fa.vhd
- fa_woco.vhd
- FN_ADDER24.vhd
- FN_MUX.vhd
- FNOUT_MUX.VHD
- FP_ADDER.vhd
- FP_ADDER_PKG.vhd

Hierarchy Files Design Units

stage	Progress %	Time
Compilation	100 %	00:02
Analysis & Synthesis	100 %	00:01
Filter	100 %	00:00
Assembler	100 %	00:00
Timing Analyzer	100 %	00:00

Timing Analyzer

Assembler Summary

Compilation Report - Assembler Summary

Assembler Status Successful - Thu Oct 25 10:04:53 2007

Revision Name FPA

Top-level Entity Name FP_ADDER

Family FLEX10K

Device EPF10K20TC144-3

III - C:/Documents and Settings/Toshiba/Desktop/VEH6004/Project/FP_ADDER/FPA - FPA - [Compilation Report - EDA Netlist Writer Summary]

File Edit View Project Assignments Processing Tools Window Help

ct Navigator

Files

- Device Design Files
 - ADDER_PKG.vhd
 - Checker24_23.vhd
 - Comp24.vhd
 - Comp8.vhd
 - EXP_MUX.vhd
 - EXP_SUBT.vhd
 - fa.vhd
 - fa_woco.vhd
 - FN_ADDER24.vhd
 - FN_MUX.vhd
 - FNOUT_MUX.VHD
 - FP_ADDER.vhd
 - FP_ADDER_PKG.vhd

Hierarchy Files Design Units

stage	Progress %	Time
Compilation	100 %	00:00
Analysis & Synthesis	100 %	00:00
Filter	100 %	00:00
Assembler	100 %	00:00
Classic Timing Analyzer	100 %	00:00

Type Message

Info: Generated files "FPA.vho" and "FPA.vhd.sdo" in directory "C:/Documents and Settings/Toshiba/Desktop/VEH6004/Project/FP_ADDER/FPA - FPA"

Info: Generated files "FPA.vho" and "FPA.vhd.sdo" in directory "C:/Documents and Settings/Toshiba/Desktop/VEH6004/Project/FP_ADDER/FPA - FPA"

Filter Summary

Compilation Report - Filter Summary

Filter Status Succes

Quartus II Version 7.1 Bi

Revision Name FPA

Top-level Entity Name FP_ADDER

Family FLEX10K

Device EPF10K20TC144-3

Timing Models Final

Total logic elements 783 / 1

Total pins 96 / 1C

Total memory bits 0 / 128

Navigator

EDA Netlist Writer Summary

Compilation Report - EDA Netlist Writer Summary

EDA Netlist Writer Status Successful - Thu Oct 25 10:05:00 2007

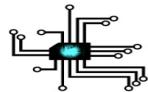
Revision Name FPA

Top-level Entity Name FP_ADDER

Family FLEX10K

Simulation Files Creation Successful

Timing Analysis Files Creation Successful



Project Navigator

Entity	Logic Cells	LC Reg
FLEX10K: AUTO		
FP_ADDER	783 (6)	0

Hierarchy Files Design Units

Status

Module	Progress %	Time
Full Compilation	100 %	00:02
Analysis & Synthesis	100 %	00:01
Fitter	100 %	00:00
Assembler	100 %	00:00
Classic Timing Analyzer	100 %	00:00
EDA Netlist Writer	100 %	00:00

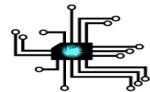
Compilation Report - Timing Analysis Generated Files

Timing Analysis Generated Files

Generated Files
1 C:/Documents and Settings/Toshiba/Desktop/VEH6004/Project/FFP_ADDER/timing/primetime/FPA.vho
2 C:/Documents and Settings/Toshiba/Desktop/VEH6004/Project/FFP_ADDER/timing/primetime/FPA_vhd.sdo
3 C:/Documents and Settings/Toshiba/Desktop/VEH6004/Project/FFP_ADDER/timing/primetime/FPA_pt_vhd.tcl

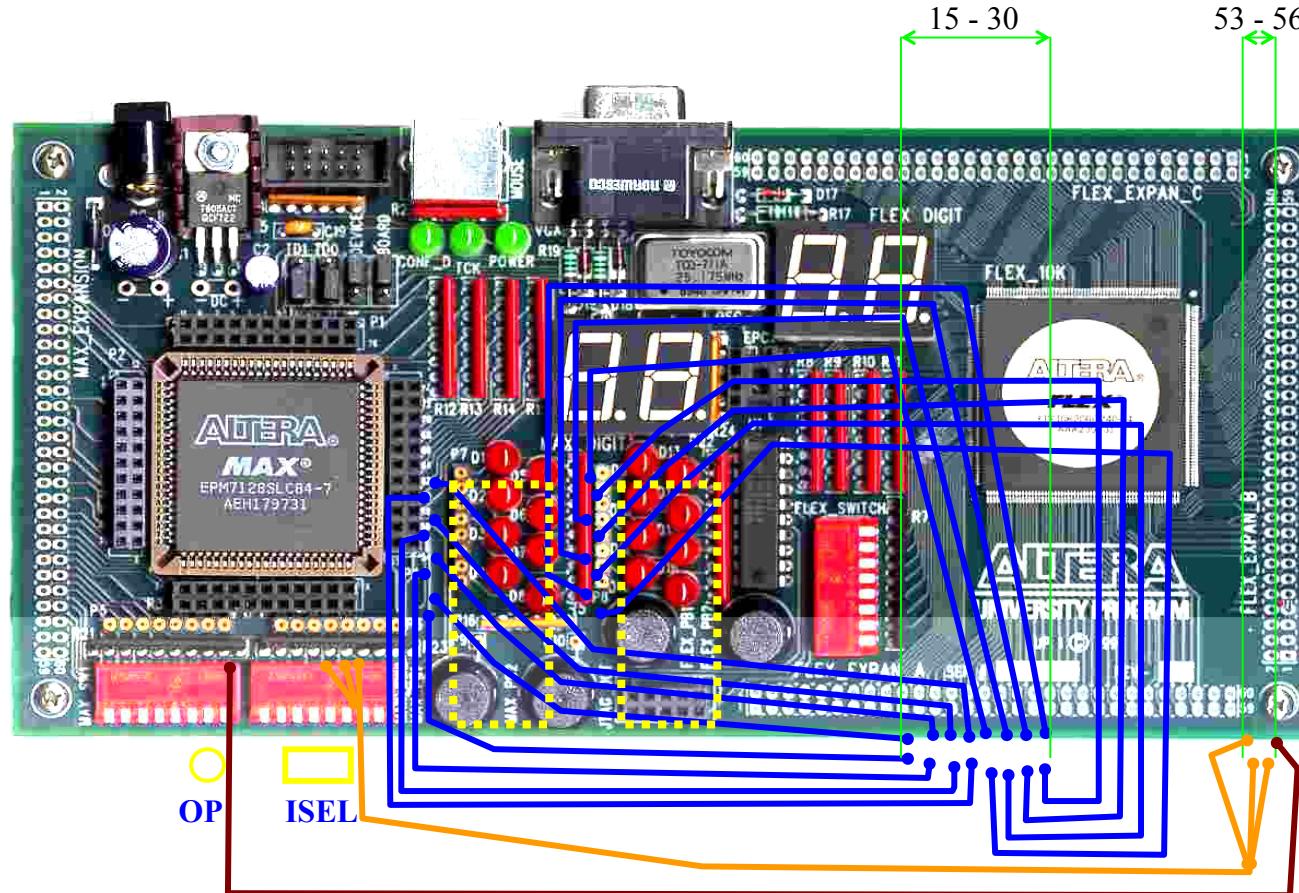
Compilation Report

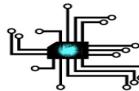
- Legal Notice
- Flow Summary
- Flow Settings
- Flow Non-Default Global Settings
- Flow Elapsed Time
- Flow Log
- Analysis & Synthesis
 - Summary
 - Settings
 - Source Files Read
 - Resource Usage Summary
 - Resource Utilization by Entity
 - Optimization Results
 - Messages
- Fitter
 - Summary
 - Settings
 - Device Options
 - Resource Section
 - Pin-Out File
 - Messages
- Assembler
 - Summary
 - Settings
 - Messages
- Timing Analyzer
 - Summary
 - Settings
 - tpd
 - Messages
- EDA Netlist Writer
 - Summary
 - Simulation
 - Timing Analysis
 - Messages



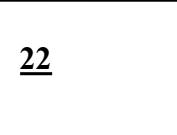
6. Hardware demonstration Results:

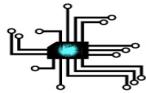
Finally, we downloaded the code into Altera FLEXIOK, then we put some wires to demonstrate the most 16-significant bits on the 16 LEDs as shown bellow:





These are some real results regarding to some inputs:

	INPUTS	N3 (ADDER, OP = 0)	HW - RSULT	N3 (SUBTRACTER, OP = 1)	HW - RSULT
A1	01000010110010000000000000000000	01000010111000000000000000000000 100 + 20 = 120		01000010101000000000000000000000 100 - 20 = 80	--
A2	01000001101000000000000000000000				
B1	11000010010010000000000000000000	11000010001100000000000000000000 -50 + 4 = -46		11000010010111000000000000000000 -50 - 4 = -54	
B2	01000000100000000000000000000000				
C1	01000001001000000000000000000000	01000010010010000000000000000000 10 + 40 = 50		11000001111100000000000000000000 10 - 40 = -30	
C2	01000010001000000000000000000000				
D1	00111111101000000000000000000000	00111110110000000000000000000000 1.625 + (-1.25) = 0.375		01000000000111000000000000000000 1.625 - (-1.25) = 2.875	--
D2	10111111101000000000000000000000				
E1	00000000000000000000000000000000	00000000000000000000000000000000 0 + 0 = 0		00000000000000000000000000000000 0 - 0 = 0	--
E2	00000000000000000000000000000000				
F1	01000001001000000000000000000000	01000001101000000000000000000000 10 + 10 = 20		00000000000000000000000000000000 10 - 10 = 0	--
F2	01000001001000000000000000000000				
G1	01000000100000000000000000000000	00000000000000000000000000000000 4 + (-4) = 0		01000001000000000000000000000000 4 - (-4) = 8	--
G2	11000000100000000000000000000000				
H1	01000000100000000000000000000000	01000001010000000000000000000000 4 + 8 = 12		11000000100000000000000000000000 4 - 8 = -4	--
H2	01000001000000000000000000000000				

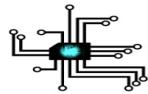


7. CONCLUSION

Coming up with an Idea that gives us the chance to apply and reflect our understanding of what we have learnt in this course was the most challenging part of our project, as much as the design itself. Designing an Adder/Subtracter for Floating point numbers in the IEEE754 format was a bit complicated project in which we have to be very careful about every single bit in the 32-bit frame and also differentiate between the exponent and mantissa parts of the whole number. Actually, our design makes the design of subtracter very easy as it needs just to invert bit#31 for the second number xorring that bit with OP input signal. By simulating the code with various test vectors the proposed approach of floating point, ADD/SUB design using VHDL is successfully designed, implemented, and tested .The work has considered the further reductions in the hardware complexity in terms of synthesis and finally the code was download into Altera FLEX10K: FPGA chip on the package for hardware realization. Everything has been done correctly which also has fitted the capability and capacity of the device (FLEX10k). The simulation, synthesis and demonstration results have been shown accordingly. In general, it was a very nice opportunity for us to work in a team and apply what we have learnt in this course in a practical work.

References:

1. Digital design and modeling with VHDL and synthesis by K.C Chang
 2. ANSIWEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic, IEEE, New York, 1985.
 3. <http://ieeexplore.ieee.org>
 4. Prof. W. Kahan, “IEEE Standard 754 for Binary Floating-Point Arithmetic”, University of California.
 5. , Jiun-Ren Lin, “Introduction to IEEE Standard 754 for Binary Floating-Point Arithmetic”, COAL, NTU CSIE, 2004.
-



Appendix:

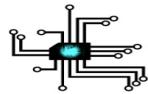
The code :

The Top Level design (The Adder – FP_Adder.vhd):

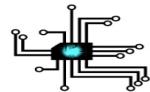
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity FP_ADDER is
    Port ( --N1 : in STD_LOGIC_VECTOR (31 downto 0);
           --N2 : in STD_LOGIC_VECTOR (31 downto 0);
           OP : in STD_LOGIC;
           ISEL : in STD_LOGIC_VECTOR (2 downto 0);
           ADD_OUT : out STD_LOGIC_VECTOR (31 downto 0));
end FP_ADDER;

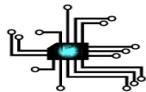
architecture Behavioral of FP_ADDER is
-- Components Declarations
    component Comp8 port (
        EXP1 : in STD_LOGIC_VECTOR (7 downto 0);
        EXP2 : in STD_LOGIC_VECTOR (7 downto 0);
        S : out STD_LOGIC);
    end component;
    component Comp24 port (
        FN1 : in STD_LOGIC_VECTOR (23 downto 0);
        FN2 : in STD_LOGIC_VECTOR (23 downto 0);
        S, EQ : out STD_LOGIC);
    end component;
    component TowsComp8 port (
        EXP : in STD_LOGIC_VECTOR (7 downto 0);
        S : in STD_LOGIC;
        EXP_2s : out STD_LOGIC_VECTOR (7 downto 0));
    end component;
```



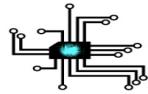
```
component TwosComp24 port (
    FN : in STD_LOGIC_VECTOR (23 downto 0);
    S : in STD_LOGIC;
    FN_2s : out STD_LOGIC_VECTOR (23 downto 0));
end component;
component SHIFTER24 port (
    FN : in STD_LOGIC_VECTOR (22 downto 0);
    DIFF, EXP : in STD_LOGIC_VECTOR (7 downto 0);
    S1, S2 : in STD_LOGIC;
    FN_OUT : out STD_LOGIC_VECTOR (23 downto 0));
end component;
component EXP_ADDER port (
    z_n1_e, z_n2_e: in std_LOGIC_vector (7 downto 0);
    C_in: in std_LOGIC;
    exp_add_out: out std_LOGIC_vector (7 downto 0));
end component;
component FN_ADDER port (
    z_n1_e, z_n2_e: in std_LOGIC_vector (23 downto 0);
    C_in: in std_LOGIC;
    FN_add_out: out std_LOGIC_vector (23 downto 0);
    C_out: out std_LOGIC);
end component;
component EXP_MUX port (
    EXP1 : in STD_LOGIC_VECTOR (7 downto 0);
    EXP2 : in STD_LOGIC_VECTOR (7 downto 0);
    S : in STD_LOGIC;
    EXP : OUT STD_LOGIC_VECTOR (7 downto 0));
end component;
component FN_MUX port (
    FN1 : in STD_LOGIC_VECTOR (22 downto 0);
    FN2 : in STD_LOGIC_VECTOR (22 downto 0);
    SS1 : in STD_LOGIC;
    SS2 : in STD_LOGIC;
    FN : OUT STD_LOGIC_VECTOR (22 downto 0));
end component;
```



```
component FNOUT_MUX Port (
    FN_MUX : in STD_LOGIC_VECTOR (22 downto 0);
    FN_CHK : in STD_LOGIC_VECTOR (22 downto 0);
    S : in STD_LOGIC;
    FN_OUT : OUT STD_LOGIC_VECTOR (22 downto 0)); end component;
component Checker24_23 port (
    FNA_OUT : in STD_LOGIC_VECTOR (23 downto 0);
    C_IN : in STD_LOGIC;
    C_OUT : in STD_LOGIC;
    FN_ADD_OUT : out STD_LOGIC_VECTOR (22 downto 0);
    ZEROS : out STD_LOGIC_VECTOR (7 downto 0);
    CH : out STD_LOGIC);
end component;
COMPONENT OUT_MUX Port (
    S : in STD_LOGIC_vector (2 DOWNTO 0);
    A1,A2,B1,B2,C1,C2,D1,D2,E1,E2,F1,F2,G1,G2,H1,H2:in STD_LOGIC_VECTOR (31 downto 0);
    N1, N2 : OUT STD_LOGIC_VECTOR (31 downto 0));
end COMPONENT;
-- SIGNALS DECLARATION
SIGNAL S1, S2, S3, NS1, NS2, NS3, EQ, SEQ, SXOR:STD_LOGIC;
SIGNAL ADD24_CIN,ADD24_COUT, ADD8_CIN:STD_LOGIC;
SIGNAL CI, SH1, SH2, S2S1, S2S2, CH:STD_LOGIC;
SIGNAL EXP12S, EXP22S, DIFF, EXP_MUX_OUT, C23, ZEROS, FEXP, ZEXP:STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL CHK_OUT, FNMUX_OUT:STD_LOGIC_VECTOR(22 DOWNTO 0);
SIGNAL FN1SH, FN2SH, FN12S, FN22S, FN_ADD:STD_LOGIC_VECTOR(23 DOWNTO 0);
SIGNAL n31: STD_LOGIC;
SIGNAL A1,A2,B1,B2,C1,C2,D1,D2,E1,E2,F1,F2,G1,G2,H1,H2, N1,N2:STD_LOGIC_VECTOR (31 downto 0);
begin
-- SIGNALS DEFINITIONS
NS1 <= NOT S1;          NS2 <= NOT S2;          NS3 <= NOT S3;
SH1 <= S1 AND S2 ;      SH2 <= NS1 AND S2;
n31 <= OP xor N2(31);
CI <= '1';              C23 <= "00010111";
ZEXP <= "00000000";
SXOR <= N1(31)XOR n31;    SEQ <= SXOR NAND EQ;      S2S1 <= S3 AND SXOR;      S2S2 <= NS3 AND SXOR;
ADD24_CIN <= S2S1 OR S2S2; ADD8_CIN <= ((NOT ADD24_CIN)AND ADD24_COUT)OR CH;
ADD_OUT(31) <= ((N1(31)AND NS3)OR(n31 AND S3))AND SEQ;
```



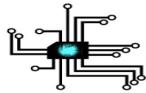
```
-- INPUTS ASSIGNMENT
A1 <= "01000010110010000000000000000000";
A2 <= "01000001101000000000000000000000";
-- N3="01000010111000000000000000000000", 100 + 20 = 120
-- N3="01000001010000000000000000000000", 100 - 20 = 80
B1 <= "11000010010010000000000000000000";
B2 <= "01000000100000000000000000000000";
-- N3="11000010001110000000000000000000", -50 + 4 = -46
-- N3="11000010010110000000000000000000", -50 - 4 = -54
C1 <= "01000001001000000000000000000000";
C2 <= "01000010001000000000000000000000";
-- N3="01000010010000000000000000000000", 10 + 40= 50
-- N3="11000010011100000000000000000000", 10 - 40= -30
D1 <= "00111111101000000000000000000000";
D2 <= "10111111101000000000000000000000";
-- N3="00111110110000000000000000000000", 1.625 + (-1.25) = 0.375
-- N3="00111110110000000000000000000000", 1.625 - (-1.25) = 2.875
E1 <= "00000000000000000000000000000000";
E2 <= "00000000000000000000000000000000";
-- N3="00000000000000000000000000000000", 0 + 0 = 0
-- N3="00000000000000000000000000000000", 0 - 0 = 0
F1 <= "01000001001000000000000000000000";
F2 <= "01000001001000000000000000000000";
-- N3="01000001101000000000000000000000", 10 + 10 = 20
-- N3="00000000000000000000000000000000", 10 - 10 = 20
G1 <= "01000000100000000000000000000000";
G2 <= "11000000100000000000000000000000";
-- N3="00000000000000000000000000000000", 4 + (-4) = 0
-- N3="01000001000000000000000000000000", 4 - (-4) = 8
H1 <= "01000000100000000000000000000000";
H2 <= "01000000100000000000000000000000";
-- N3="01000001010000000000000000000000", 4 + 8 = 12
-- N3="11000000100000000000000000000000", 4 - 8 = -4
```



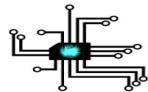
-- COMPONENTS PORT MAPPING

OUT_MUX_01: OUT_MUX port map (ISEL,A1,A2,B1,B2,C1,C2,D1,D2,E1,E2,F1,F2,G1,G2,H1,H2, N1,N2);
COMP8_0: Comp8 port map (N1(30 DOWNT0 23),N2(30 DOWNT0 23),S1);
COMP24_0: Comp24 port map (FN1SH,FN2SH,S3,EQ);
COMP8_1: Comp8 port map (DIFF,C23,S2);
TowsComp8_0: TowsComp8 port map (N1(30 DOWNT0 23),S1,EXP12S);
TowsComp8_1: TowsComp8 port map (N2(30 DOWNT0 23),NS1,EXP22S);
EXP_ADDER_0: EXP_ADDER port map (EXP12S,EXP22S,CI,DIFF);
EXP_MUX_0: EXP_MUX port map (N1(30 DOWNT0 23),N2(30 DOWNT0 23),S1,EXP_MUX_OUT);
FN_MUX_0: FN_MUX port map (N1(22 DOWNT0 0),N2(22 DOWNT0 0),S1,NS2,FNMUX_OUT);
SHIFTER24_0: SHIFTER24 port map (N1(22 DOWNT0 0),DIFF,N1(30 DOWNT0 23),S1,S2,FN1SH);
SHIFTER24_1: SHIFTER24 port map (N2(22 DOWNT0 0),DIFF,N2(30 DOWNT0 23),NS1,S2,FN2SH);
TwosComp24_0: TwosComp24 port map (FN1SH,S2S1,FN12S);
TwosComp24_1: TwosComp24 port map (FN2SH,S2S2,FN22S);
FN_ADDER_0: FN_ADDER port map (FN12S,FN22S,ADD24_CIN,FN_ADD,ADD24_COUT);
Checker24_23_1: Checker24_23 port map (FN_ADD,ADD24_CIN,ADD24_COUT,CHK_OUT,ZEROS,CH);
EXP_ADDER_1: EXP_ADDER port map (EXP_MUX_OUT,ZEROS,ADD8_CIN,FEXP);
EXP_MUX_1: EXP_MUX port map (ZEXP,FEXP,SEQ,ADD_OUT(30 DOWNT0 23));
FNOOUT_MUX_0: FNOOUT_MUX port map (FNMUX_OUT,CHK_OUT,NS2,ADD_OUT(22 DOWNT0 0));

end Behavioral;

**TwosComp24.vhd:**

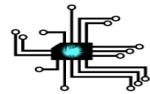
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity TwosComp24 is
    Port ( FN : in STD_ULOGIC_VECTOR (23 downto 0);
           S : in STD_ULOGIC;
           FN_2s : out STD_ULOGIC_VECTOR (23 downto 0));
end TwosComp24;
architecture Behavioral of TwosComp24 is
begin
    FN_2s(23) <= FN(23) XOR S;
    FN_2s(22) <= FN(22) XOR S;
    FN_2s(21) <= FN(21) XOR S;
    FN_2s(20) <= FN(20) XOR S;
    FN_2s(19) <= FN(19) XOR S;
    FN_2s(18) <= FN(18) XOR S;
    FN_2s(17) <= FN(17) XOR S;
    FN_2s(16) <= FN(16) XOR S;
    FN_2s(15) <= FN(15) XOR S;
    FN_2s(14) <= FN(14) XOR S;
    FN_2s(13) <= FN(13) XOR S;
    FN_2s(12) <= FN(12) XOR S;
    FN_2s(11) <= FN(11) XOR S;
    FN_2s(10) <= FN(10) XOR S;
    FN_2s(9) <= FN(9) XOR S;
    FN_2s(8) <= FN(8) XOR S;
    FN_2s(7) <= FN(7) XOR S;
    FN_2s(6) <= FN(6) XOR S;
    FN_2s(5) <= FN(5) XOR S;
    FN_2s(4) <= FN(4) XOR S;
    FN_2s(3) <= FN(3) XOR S;
    FN_2s(2) <= FN(2) XOR S;
    FN_2s(1) <= FN(1) XOR S;
    FN_2s(0) <= FN(0) XOR S;
end Behavioral;
```

**TwosComp8.vhd:**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity TowsComp8 is
    Port ( EXP : in STD_ULOGIC_VECTOR (7 downto 0);
           S : in STD_ULOGIC;
           EXP_2s : out STD_ULOGIC_VECTOR (7 downto 0));
end TowsComp8;
architecture Behavioral of TowsComp8 is
begin
    EXP_2s(7) <= EXP(7) XOR S;
    EXP_2s(6) <= EXP(6) XOR S;
    EXP_2s(5) <= EXP(5) XOR S;
    EXP_2s(4) <= EXP(4) XOR S;
    EXP_2s(3) <= EXP(3) XOR S;
    EXP_2s(2) <= EXP(2) XOR S;
    EXP_2s(1) <= EXP(1) XOR S;
    EXP_2s(0) <= EXP(0) XOR S;
end Behavioral;
```

fa.vhd:

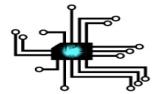
```
library ieee;          -- Load the ieee 1164 library
use ieee.std_logic_1164.all; -- Make the ieee package 'visible'
entity fa is port(
    a, b, c_in: in std_ulogic;
    c_out, s: out std_ulogic);
end entity fa;
architecture behavioral of fa is
begin
    process (a, b, c_in)
        begin
            c_out <= (a and b) or (a and c_in) or (b and c_in);
            s <= a xor b xor c_in;
        end process;
end architecture behavioral;
```

**fa_wcoB.vhd:**

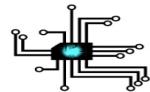
```
library ieee;          -- Load the ieee 1164 library
use ieee.std_logic_1164.all; -- Make the ieee package 'visible'
entity fa_woco is port(
    a, b, c_in: in std_ulogic;
    s: out std_ulogic);
end entity fa_woco;
architecture behavioral of fa_woco is
begin
    process (a, b, c_in)
        begin
            s <= a xor b xor c_in;
        end process;
end architecture behavioral;
```

EXP_SUB.vhd:

```
library ieee;          -- Load the ieee 1164 library
use ieee.std_logic_1164.all; -- Make the ieee package 'visible'
entity EXP_ADDER is port(
    z_n1_e, z_n2_e: in std_ulogic_vector (7 downto 0);
    C_in: in std_ulogic;
    exp_add_out: out std_ulogic_vector (7 downto 0));
end entity EXP_ADDER;
use work.ADDER_pkg.all; -- note need to compile all sub-entities before this entity can be compiled!
architecture struct of EXP_ADDER is
    signal carry_bit: std_ulogic_vector(8 downto 0);
    begin
        carry_bit(0)<= C_in;
        fa_0: fa port map (z_n1_e(0), z_n2_e(0), carry_bit(0), carry_bit(1), exp_add_out(0));
        fa_1: fa port map (z_n1_e(1), z_n2_e(1), carry_bit(1), carry_bit(2), exp_add_out(1));
        fa_2: fa port map (z_n1_e(2), z_n2_e(2), carry_bit(2), carry_bit(3), exp_add_out(2));
        fa_3: fa port map (z_n1_e(3), z_n2_e(3), carry_bit(3), carry_bit(4), exp_add_out(3));
        fa_4: fa port map (z_n1_e(4), z_n2_e(4), carry_bit(4), carry_bit(5), exp_add_out(4));
        fa_5: fa port map (z_n1_e(5), z_n2_e(5), carry_bit(5), carry_bit(6), exp_add_out(5));
        fa_6: fa port map (z_n1_e(6), z_n2_e(6), carry_bit(6), carry_bit(7), exp_add_out(6));
        fa_woco_7: fa_woco port map (z_n1_e(7), z_n2_e(7), carry_bit(7), exp_add_out(7));
    end architecture struct;
```

**EXP_MUX.vhd:**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity EXP_MUX is
    Port ( EXP1 : in STD_ULOGIC_VECTOR (7 downto 0);
           EXP2 : in STD_ULOGIC_VECTOR (7 downto 0);
           S : in STD_ULOGIC;
           EXP : OUT STD_ULOGIC_VECTOR (7 downto 0));
end EXP_MUX;
architecture Behavioral of EXP_MUX is
begin
PROCESS(S, EXP1, EXP2)
begin
    if S = '1' then
        EXP <= EXP2;
    else
        EXP <= EXP1;
    END IF;
END PROCESS;
end Behavioral;
```

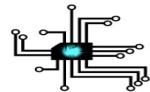
**Comp8.vhd:**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Comp8 is
    Port ( EXP1 : in STD_ULOGIC_VECTOR (7 downto 0);
           EXP2 : in STD_ULOGIC_VECTOR (7 downto 0);
           S : out STD_ULOGIC);
end Comp8;
architecture Behavioral of Comp8 is
signal X:std_Ulogic_vector(7 downto 1);
begin

X(7)<= EXP1(7) XNOR EXP2(7);
X(6)<= EXP1(6) XNOR EXP2(6);
X(5)<= EXP1(5) XNOR EXP2(5);
X(4)<= EXP1(4) XNOR EXP2(4);
X(3)<= EXP1(3) XNOR EXP2(3);
X(2)<= EXP1(2) XNOR EXP2(2);
X(1)<= EXP1(1) XNOR EXP2(1);
--X(0)<= EXP1(0) XNOR EXP2(0);

S <= ((NOT EXP1(7)) AND EXP2(7))
      OR (X(7)AND(NOT EXP1(6)) AND EXP2(6))
      OR (X(7)AND X(6)AND(NOT EXP1(5)) AND EXP2(5))
      OR (X(7)AND X(6)AND X(5)AND(NOT EXP1(4)) AND EXP2(4))
      OR (X(7)AND X(6)AND X(5)AND X(4)AND(NOT EXP1(3)) AND EXP2(3))
      OR (X(7)AND X(6)AND X(5)AND X(4)AND X(3)AND(NOT EXP1(2)) AND EXP2(2))
      OR (X(7)AND X(6)AND X(5)AND X(4)AND X(3)AND X(2)AND(NOT EXP1(1)) AND EXP2(1))
      OR (X(7)AND X(6)AND X(5)AND X(4)AND X(3)AND X(2)AND X(1)AND(NOT EXP1(0)) AND EXP2(0));

end Behavioral;
```

**TwosComp24.vhd:**

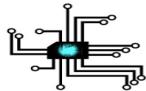
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Comp24 is
    Port ( FN1 : in STD_ULOGIC_VECTOR (23 downto 0);
           FN2 : in STD_ULOGIC_VECTOR (23 downto 0);
           S, EQ : out STD_ULOGIC);
end Comp24;
```

```
architecture Behavioral of Comp24 is
```

```
signal X:std_Ulogic_vector(23 downto 0);
```

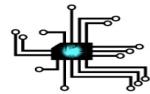
```
begin
```

```
X(23)<= FN1(23) XNOR FN2(23);
X(22)<= FN1(22) XNOR FN2(22);
X(21)<= FN1(21) XNOR FN2(21);
X(20)<= FN1(20) XNOR FN2(20);
X(19)<= FN1(19) XNOR FN2(19);
X(18)<= FN1(18) XNOR FN2(18);
X(17)<= FN1(17) XNOR FN2(17);
X(16)<= FN1(16) XNOR FN2(16);
X(15)<= FN1(15) XNOR FN2(15);
X(14)<= FN1(14) XNOR FN2(14);
X(13)<= FN1(13) XNOR FN2(13);
X(12)<= FN1(12) XNOR FN2(12);
X(11)<= FN1(11) XNOR FN2(11);
X(10)<= FN1(10) XNOR FN2(10);
X(9)<= FN1(9) XNOR FN2(9);
X(8)<= FN1(8) XNOR FN2(8);
X(7)<= FN1(7) XNOR FN2(7);
X(6)<= FN1(6) XNOR FN2(6);
X(5)<= FN1(5) XNOR FN2(5);
X(4)<= FN1(4) XNOR FN2(4);
X(3)<= FN1(3) XNOR FN2(3);
X(2)<= FN1(2) XNOR FN2(2);
X(1)<= FN1(1) XNOR FN2(1);
X(0)<= FN1(0) XNOR FN2(0);
```



S <= ((NOT FN1(23)) AND FN2(23))
OR (X(23)AND(NOT FN1(22)) AND FN2(22)) OR (X(23)AND X(22)AND(NOT FN1(21)) AND FN2(21))
OR (X(23)AND X(22)AND X(21)AND(NOT FN1(20)) AND FN2(20)) OR (X(23)AND X(22)AND X(21)AND X(20)AND(NOT FN1(19)) AND FN2(19))
OR (X(23)AND X(22)AND X(21)AND X(20)AND X(19)AND(NOT FN1(18)) AND FN2(18))
OR (X(23)AND X(22)AND X(21)AND X(20)AND X(19)AND X(18)AND(NOT FN1(17)) AND FN2(17))
OR (X(23)AND X(22)AND X(21)AND X(20)AND X(19)AND X(18)AND X(17)AND(NOT FN1(16)) AND FN2(16))
OR (X(23)AND X(22)AND X(21)AND X(20)AND X(19)AND X(18)AND X(17)AND X(16)AND(NOT FN1(15)) AND FN2(15))
OR (X(23)AND X(22)AND X(21)AND X(20)AND X(19)AND X(18)AND X(17)AND X(16)AND X(15)AND(NOT FN1(14)) AND FN2(14))
OR (X(23)AND X(22)AND X(21)AND X(20)AND X(19)AND X(18)AND X(17)AND X(16)AND X(15)AND X(14)AND(NOT FN1(13)) AND FN2(13))
OR (X(23)AND X(22)AND X(21)AND X(20)AND X(19)AND X(18)AND X(17)AND X(16)AND X(15)AND X(14)AND X(13)AND(NOT FN1(12)) AND FN2(12))
OR (X(23)AND X(22)AND X(21)AND X(20)AND X(19)AND X(18)AND X(17)AND X(16)AND X(15)AND X(14)AND X(13)AND X(12)AND(NOT FN1(11))
AND FN2(11))
OR (X(23)AND X(22)AND X(21)AND X(20)AND X(19)AND X(18)AND X(17)AND X(16)AND X(15)AND X(14)AND X(13)AND X(12)AND X(11)AND(NOT
FN1(10)) AND FN2(10))
OR (X(23)AND X(22)AND X(21)AND X(20)AND X(19)AND X(18)AND X(17)AND X(16)AND X(15)AND X(14)AND X(13)AND X(12)AND X(11)AND
X(10)AND(NOT FN1(9)) AND FN2(9))
OR (X(23)AND X(22)AND X(21)AND X(20)AND X(19)AND X(18)AND X(17)AND X(16)AND X(15)AND X(14)AND X(13)AND X(12)AND X(11)AND
X(10)AND X(9)AND(NOT FN1(8)) AND FN2(8))
OR (X(23)AND X(22)AND X(21)AND X(20)AND X(19)AND X(18)AND X(17)AND X(16)AND X(15)AND X(14)AND X(13)AND X(12)AND X(11)AND
X(10)AND X(9)AND X(8)AND(NOT FN1(7)) AND FN2(7))
OR (X(23)AND X(22)AND X(21)AND X(20)AND X(19)AND X(18)AND X(17)AND X(16)AND X(15)AND X(14)AND X(13)AND X(12)AND X(11)AND
X(10)AND X(9)AND X(8)AND X(7)AND(NOT FN1(6)) AND FN2(6))
OR (X(23)AND X(22)AND X(21)AND X(20)AND X(19)AND X(18)AND X(17)AND X(16)AND X(15)AND X(14)AND X(13)AND X(12)AND X(11)AND
X(10)AND X(9)AND X(8)AND X(7)AND X(6)AND(NOT FN1(5)) AND FN2(5))
OR (X(23)AND X(22)AND X(21)AND X(20)AND X(19)AND X(18)AND X(17)AND X(16)AND X(15)AND X(14)AND X(13)AND X(12)AND X(11)AND
X(10)AND X(9)AND X(8)AND X(7)AND X(6)AND X(5)AND(NOT FN1(4)) AND FN2(4))
OR (X(23)AND X(22)AND X(21)AND X(20)AND X(19)AND X(18)AND X(17)AND X(16)AND X(15)AND X(14)AND X(13)AND X(12)AND X(11)AND
X(10)AND X(9)AND X(8)AND X(7)AND X(6)AND X(5)AND X(4)AND(NOT FN1(3)) AND FN2(3))
OR (X(23)AND X(22)AND X(21)AND X(20)AND X(19)AND X(18)AND X(17)AND X(16)AND X(15)AND X(14)AND X(13)AND X(12)AND X(11)AND
X(10)AND X(9)AND X(8)AND X(7)AND X(6)AND X(5)AND X(4)AND X(3)AND(NOT FN1(2)) AND FN2(2))
OR (X(23)AND X(22)AND X(21)AND X(20)AND X(19)AND X(18)AND X(17)AND X(16)AND X(15)AND X(14)AND X(13)AND X(12)AND X(11)AND
X(10)AND X(9)AND X(8)AND X(7)AND X(6)AND X(5)AND X(4)AND X(3)AND X(2)AND(NOT FN1(1)) AND FN2(1))
OR (X(23)AND X(22)AND X(21)AND X(20)AND X(19)AND X(18)AND X(17)AND X(16)AND X(15)AND X(14)AND X(13)AND X(12)AND X(11)AND
X(10)AND X(9)AND X(8)AND X(7)AND X(6)AND X(5)AND X(4)AND X(3)AND X(2)AND X(1)AND(NOT FN1(0)) AND FN2(0));

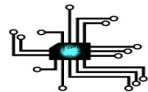
EQ <= X(23)AND X(22)AND X(21)AND X(20)AND X(19)AND X(18)AND X(17)AND X(16)AND X(15)AND X(14)AND X(13)AND X(12)AND X(11)AND X(10)AND
X(9)AND X(8)AND X(7)AND X(6)AND X(5)AND X(4)AND X(3)AND X(2)AND X(1)AND X(0);
end Behavioral;

**FN_MUX.vhd:**

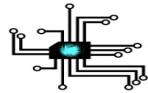
```
entity FN_MUX is
  Port ( FN1, FN2 : in STD_ULOGIC_VECTOR (22 downto 0);
         SS1, SS2 : in STD_ULOGIC;
         FN : OUT STD_ULOGIC_VECTOR (22 downto 0));
end FN_MUX;
architecture Behavioral of FN_MUX is
begin
PROCESS(SS1,SS2, FN1, FN2)
begin
  IF SS2 = '1' then
    IF SS1 = '1' then
      FN <= FN2;
    ELSE
      FN <= FN1;
    END IF;
  ELSE
    FN <= "000000000000000000000000";
  END IF;
END PROCESS;
end Behavioral;
```

FNOUT_MUX.vhd:

```
entity FNOUT_MUX is
  Port ( FN_MUX, FN_CHK : in STD_ULOGIC_VECTOR (22 downto 0);
         S : in STD_ULOGIC;
         FN_OUT : OUT STD_ULOGIC_VECTOR (22 downto 0));
end FNOUT_MUX;
architecture Behavioral of FNOUT_MUX is
begin
PROCESS(S, FN_MUX, FN_CHK)
begin
  IF S = '1' then
    FN_OUT <= FN_MUX;
  ELSE
    FN_OUT <= FN_CHK;
  END IF;
END PROCESS;
end Behavioral;
```

**SHIFTER24.vhd:**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity SHIFTER24 is
    Port ( FN : in STD_ULOGIC_VECTOR (22 downto 0);
           DIFF, EXP : in STD_ULOGIC_VECTOR (7 downto 0);
           S1, S2 : in STD_ULOGIC;
           FN_OUT : out STD_ULOGIC_VECTOR (23 downto 0));
end SHIFTER24;
architecture Behavioral of SHIFTER24 is
Signal SH: std_Ulogic;
Signal ADDR: std_Ulogic_vector(7 downto 0);
Signal FN_IN: std_Ulogic_vector(23 downto 0);
begin
SH <= S1 AND S2;
process (FN, EXP)
BEGIN
FN_IN(22 downto 0)<= FN;
    IF (FN <= "00000000000000000000000000000000" AND EXP <= "00000000") THEN
        FN_IN(23)<='0';
    ELSE
        FN_IN(23)<='1';
    END IF;
END PROCESS;
process (S1, S2, SH, DIFF)
begin
    if S1 = '1' AND S2 = '0' THEN
        ADDR <= "11111111";
    ELSE IF SH = '1' then
        ADDR <= DIFF;
    else --if CLK'event and CLK = '1' then
        ADDR <= "00000000";
    END if;
    END IF;
end process;
```

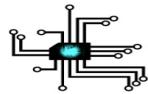


Process (ADDR,FN_IN)

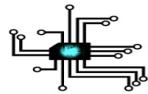
BEGIN

case ADDR is

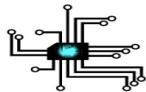
```
when "00011000" => FN_OUT(23 DOWNTO 0)<= "0000000000000000000000000000";
when "00010111" => FN_OUT(0)<= FN_IN(23); FN_OUT(23 DOWNTO 1)<= "0000000000000000000000000000";
when "00010110" => FN_OUT(1 DOWNTO 0)<= FN_IN(23 DOWNTO 22); FN_OUT(23 DOWNTO 2)<= "0000000000000000000000000000";
when "00010101" => FN_OUT(2 DOWNTO 0)<= FN_IN(23 DOWNTO 21); FN_OUT(23 DOWNTO 3)<= "0000000000000000000000000000";
when "00010100" => FN_OUT(3 DOWNTO 0)<= FN_IN(23 DOWNTO 20); FN_OUT(23 DOWNTO 4)<= "0000000000000000000000000000";
when "00010011" => FN_OUT(4 DOWNTO 0)<= FN_IN(23 DOWNTO 19); FN_OUT(23 DOWNTO 5)<= "0000000000000000000000000000";
when "00010010" => FN_OUT(5 DOWNTO 0)<= FN_IN(23 DOWNTO 18); FN_OUT(23 DOWNTO 6)<= "0000000000000000000000000000";
when "00010001" => FN_OUT(6 DOWNTO 0)<= FN_IN(23 DOWNTO 17); FN_OUT(23 DOWNTO 7)<= "0000000000000000000000000000";
when "00010000" => FN_OUT(7 DOWNTO 0)<= FN_IN(23 DOWNTO 16); FN_OUT(23 DOWNTO 8)<= "0000000000000000000000000000";
when "00001111" => FN_OUT(8 DOWNTO 0)<= FN_IN(23 DOWNTO 15); FN_OUT(23 DOWNTO 9)<= "0000000000000000000000000000";
when "00001110" => FN_OUT(9 DOWNTO 0)<= FN_IN(23 DOWNTO 14); FN_OUT(23 DOWNTO 10)<= "0000000000000000000000000000";
when "00001101" => FN_OUT(10 DOWNTO 0)<= FN_IN(23 DOWNTO 13); FN_OUT(23 DOWNTO 11)<= "0000000000000000000000000000";
when "00001100" => FN_OUT(11 DOWNTO 0)<= FN_IN(23 DOWNTO 12); FN_OUT(23 DOWNTO 12)<= "0000000000000000000000000000";
when "00001011" => FN_OUT(12 DOWNTO 0)<= FN_IN(23 DOWNTO 11); FN_OUT(23 DOWNTO 13)<= "0000000000000000000000000000";
when "00001010" => FN_OUT(13 DOWNTO 0)<= FN_IN(23 DOWNTO 10); FN_OUT(23 DOWNTO 14)<= "0000000000000000000000000000";
when "00001001" => FN_OUT(14 DOWNTO 0)<= FN_IN(23 DOWNTO 9); FN_OUT(23 DOWNTO 15)<= "0000000000000000000000000000";
when "00001000" => FN_OUT(15 DOWNTO 0)<= FN_IN(23 DOWNTO 8); FN_OUT(23 DOWNTO 16)<= "0000000000000000000000000000";
when "00000111" => FN_OUT(16 DOWNTO 0)<= FN_IN(23 DOWNTO 7); FN_OUT(23 DOWNTO 17)<= "0000000000000000000000000000";
when "00000110" => FN_OUT(17 DOWNTO 0)<= FN_IN(23 DOWNTO 6); FN_OUT(23 DOWNTO 18)<= "0000000000000000000000000000";
when "00000101" => FN_OUT(18 DOWNTO 0)<= FN_IN(23 DOWNTO 5); FN_OUT(23 DOWNTO 19)<= "0000000000000000000000000000";
when "00000100" => FN_OUT(19 DOWNTO 0)<= FN_IN(23 DOWNTO 4); FN_OUT(23 DOWNTO 20)<= "0000000000000000000000000000";
when "00000011" => FN_OUT(20 DOWNTO 0)<= FN_IN(23 DOWNTO 3); FN_OUT(23 DOWNTO 21)<= "0000000000000000000000000000";
when "00000010" => FN_OUT(21 DOWNTO 0)<= FN_IN(23 DOWNTO 2); FN_OUT(23 DOWNTO 22)<= "0000000000000000000000000000";
when "00000001" => FN_OUT(22 DOWNTO 0)<= FN_IN(23 DOWNTO 1); FN_OUT(23)<= '0';
when "00000000" => FN_OUT <= FN_IN;
when OTHERS => FN_OUT <= "00000000000000000000000000000000"; end case; end process; end Behavioral;
```

**FN_Adder24.vhd:**

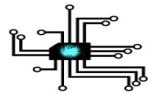
```
library ieee;      use ieee.std_logic_1164.all;
entity FN_ADDER is port(
    z_n1_e, z_n2_e: in std_ulogic_vector (23 downto 0); -- note extra bit for (1.) PART OF HTE FLOAT NUBER
    C_in: in std_ulogic; FN_add_out: out std_ulogic_vector (23 downto 0); C_out: out std_ulogic);
end entity FN_ADDER;
use work.ADDER_PKG.all; -- note need to compile all sub-entities before this entity can be compiled!
architecture struct of FN_ADDER is
    signal carry_bit: std_ulogic_vector(24 downto 0);
begin
    carry_bit(0)<= c_in; -- out put of comps_OR
    fa_0: fa port map (z_n1_e(0), z_n2_e(0), carry_bit(0), carry_bit(1), FN_add_out(0));
    fa_1: fa port map (z_n1_e(1), z_n2_e(1), carry_bit(1), carry_bit(2), FN_add_out(1));
    fa_2: fa port map (z_n1_e(2), z_n2_e(2), carry_bit(2), carry_bit(3), FN_add_out(2));
    fa_3: fa port map (z_n1_e(3), z_n2_e(3), carry_bit(3), carry_bit(4), FN_add_out(3));
    fa_4: fa port map (z_n1_e(4), z_n2_e(4), carry_bit(4), carry_bit(5), FN_add_out(4));
    fa_5: fa port map (z_n1_e(5), z_n2_e(5), carry_bit(5), carry_bit(6), FN_add_out(5));
    fa_6: fa port map (z_n1_e(6), z_n2_e(6), carry_bit(6), carry_bit(7), FN_add_out(6));
    fa_7: fa port map (z_n1_e(7), z_n2_e(7), carry_bit(7), carry_bit(8), FN_add_out(7));
    fa_8: fa port map (z_n1_e(8), z_n2_e(8), carry_bit(8), carry_bit(9), FN_add_out(8));
    fa_9: fa port map (z_n1_e(9), z_n2_e(9), carry_bit(9), carry_bit(10), FN_add_out(9));
    fa_10: fa port map (z_n1_e(10), z_n2_e(10), carry_bit(10), carry_bit(11), FN_add_out(10));
    fa_11: fa port map (z_n1_e(11), z_n2_e(11), carry_bit(11), carry_bit(12), FN_add_out(11));
    fa_12: fa port map (z_n1_e(12), z_n2_e(12), carry_bit(12), carry_bit(13), FN_add_out(12));
    fa_13: fa port map (z_n1_e(13), z_n2_e(13), carry_bit(13), carry_bit(14), FN_add_out(13));
    fa_14: fa port map (z_n1_e(14), z_n2_e(14), carry_bit(14), carry_bit(15), FN_add_out(14));
    fa_15: fa port map (z_n1_e(15), z_n2_e(15), carry_bit(15), carry_bit(16), FN_add_out(15));
    fa_16: fa port map (z_n1_e(16), z_n2_e(16), carry_bit(16), carry_bit(17), FN_add_out(16));
    fa_17: fa port map (z_n1_e(17), z_n2_e(17), carry_bit(17), carry_bit(18), FN_add_out(17));
    fa_18: fa port map (z_n1_e(18), z_n2_e(18), carry_bit(18), carry_bit(19), FN_add_out(18));
    fa_19: fa port map (z_n1_e(19), z_n2_e(19), carry_bit(19), carry_bit(20), FN_add_out(19));
    fa_20: fa port map (z_n1_e(20), z_n2_e(20), carry_bit(20), carry_bit(21), FN_add_out(20));
    fa_21: fa port map (z_n1_e(21), z_n2_e(21), carry_bit(21), carry_bit(22), FN_add_out(21));
    fa_22: fa port map (z_n1_e(22), z_n2_e(22), carry_bit(22), carry_bit(23), FN_add_out(22));
    fa_23: fa port map (z_n1_e(23), z_n2_e(23), carry_bit(23), carry_bit(24), FN_add_out(23));
    c_out <= carry_bit(24);
end architecture struct;
```

**Checker24_23.vhd:**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Checker24_23 is
    Port ( FNA_OUT : in STD_ULOGIC_VECTOR (23 downto 0);
           C_IN : in STD_ULOGIC;
           C_OUT : in STD_ULOGIC;
           FN_ADD_OUT : out STD_ULOGIC_VECTOR (22 downto 0);
           ZEROS : out STD_ULOGIC_VECTOR (7 downto 0);
           CH : out STD_ULOGIC);
end Checker24_23;
architecture Behavioral of Checker24_23 is
begin
    process (C_IN, FNA_OUT, C_OUT)
        begin
            IF C_IN = '1' THEN
                IF FNA_OUT(23 downto 1) <= "000000000000000000000000000001" THEN FN_ADD_OUT(22)<= FNA_OUT(0); FN_ADD_OUT(21 DOWNT0 0)<=
                    "000000000000000000000000"; ZEROS <= "11101001"; CH <= '1';
                ELSE IF FNA_OUT(23 downto 2) <= "000000000000000000000000000001" THEN FN_ADD_OUT(22 DOWNT0 21)<= FNA_OUT(1 DOWNT0 0); FN_ADD_OUT(20
                    DOWNT0 0)<= "000000000000000000000000"; ZEROS <= "11101010"; CH <= '1';
                ELSE IF FNA_OUT(23 downto 3) <= "000000000000000000000000000001" THEN FN_ADD_OUT(22 DOWNT0 20)<= FNA_OUT(2 DOWNT0 0); FN_ADD_OUT(19
                    DOWNT0 0)<= "000000000000000000000000"; ZEROS <= "11101011"; CH <= '1';
                ELSE IF FNA_OUT(23 downto 4) <= "000000000000000000000000000001" THEN FN_ADD_OUT(22 DOWNT0 19)<= FNA_OUT(3 DOWNT0 0); FN_ADD_OUT(18
                    DOWNT0 0)<= "000000000000000000000000"; ZEROS <= "11101100"; CH <= '1';
                ELSE IF FNA_OUT(23 downto 5) <= "000000000000000000000000000001" THEN FN_ADD_OUT(22 DOWNT0 18)<= FNA_OUT(4 DOWNT0 0); FN_ADD_OUT(17
                    DOWNT0 0)<= "000000000000000000000000"; ZEROS <= "11101101"; CH <= '1';
                ELSE IF FNA_OUT(23 downto 6) <= "000000000000000000000000000001" THEN FN_ADD_OUT(22 DOWNT0 17)<= FNA_OUT(5 DOWNT0 0); FN_ADD_OUT(16 DOWNT0
                    0)<= "000000000000000000000000"; ZEROS <= "11101110"; CH <= '1';
                ELSE IF FNA_OUT(23 downto 7) <= "000000000000000000000000000001" THEN FN_ADD_OUT(22 DOWNT0 16)<= FNA_OUT(6 DOWNT0 0); FN_ADD_OUT(15 DOWNT0
                    0)<= "000000000000000000000000"; ZEROS <= "11101111"; CH <= '1';
                ELSE IF FNA_OUT(23 downto 8) <= "000000000000000000000000000001" THEN FN_ADD_OUT(22 DOWNT0 15)<= FNA_OUT(7 DOWNT0 0); FN_ADD_OUT(14 DOWNT0
                    0)<= "000000000000000000000000"; ZEROS <= "11110000"; CH <= '1';
                ELSE IF FNA_OUT(23 downto 9) <= "000000000000000000000000000001" THEN FN_ADD_OUT(22 DOWNT0 14)<= FNA_OUT(8 DOWNT0 0); FN_ADD_OUT(13 DOWNT0
                    0)<= "000000000000000000000000"; ZEROS <= "11110001"; CH <= '1';
                ELSE IF FNA_OUT(23 downto 10)<= "000000000000000000000000000001" THEN FN_ADD_OUT(22 DOWNT0 13)<= FNA_OUT(9 DOWNT0 0); FN_ADD_OUT(12 DOWNT0 0)<=
                    "000000000000000000000000"; ZEROS <= "11110010"; CH <= '1';
            end if;
        end process;
```



```
ELSE IF FNA_OUT(23 downto 11) <= "0000000000001" THEN FN_ADD_OUT(22 DOWNT0 12)<= FNA_OUT(10 DOWNT0 0); FN_ADD_OUT(11 DOWNT0 0)<= "00000000000"; ZEROS <= "11110011"; CH <= '1';
ELSE IF FNA_OUT(23 downto 12) <= "000000000001" THEN FN_ADD_OUT(22 DOWNT0 11)<= FNA_OUT(11 DOWNT0 0); FN_ADD_OUT(10 DOWNT0 0)<= "00000000000"; ZEROS <= "11110100"; CH <= '1';
ELSE IF FNA_OUT(23 downto 13) <= "00000000001" THEN FN_ADD_OUT(22 DOWNT0 10)<= FNA_OUT(12 DOWNT0 0); FN_ADD_OUT(9 DOWNT0 0)<= "0000000000"; ZEROS <= "11110101"; CH <= '1';
ELSE IF FNA_OUT(23 downto 14) <= "0000000001" THEN FN_ADD_OUT(22 DOWNT0 9)<= FNA_OUT(13 DOWNT0 0); FN_ADD_OUT(8 DOWNT0 0)<= "000000000"; ZEROS <= "11110110"; CH <= '1';
ELSE IF FNA_OUT(23 downto 15) <= "0000000001" THEN FN_ADD_OUT(22 DOWNT0 8)<= FNA_OUT(14 DOWNT0 0); FN_ADD_OUT(7 DOWNT0 0)<= "00000000"; ZEROS <= "11110111"; CH <= '1';
ELSE IF FNA_OUT(23 downto 16) <= "000000001" THEN FN_ADD_OUT(22 DOWNT0 7)<= FNA_OUT(15 DOWNT0 0); FN_ADD_OUT(6 DOWNT0 0)<= "0000000"; ZEROS <= "11111000"; CH <= '1';
ELSE IF FNA_OUT(23 downto 17) <= "00000001" THEN FN_ADD_OUT(22 DOWNT0 6)<= FNA_OUT(16 DOWNT0 0); FN_ADD_OUT(5 DOWNT0 0)<= "000000"; ZEROS <= "11111001"; CH <= '1';
ELSE IF FNA_OUT(23 downto 18) <= "0000001" THEN FN_ADD_OUT(22 DOWNT0 5)<= FNA_OUT(17 DOWNT0 0); FN_ADD_OUT(4 DOWNT0 0)<= "00000"; ZEROS <= "11111010"; CH <= '1';
ELSE IF FNA_OUT(23 downto 19) <= "00001" THEN FN_ADD_OUT(22 DOWNT0 4)<= FNA_OUT(18 DOWNT0 0); FN_ADD_OUT(3 DOWNT0 0)<= "0000"; ZEROS <= "11111011"; CH <= '1';
ELSE IF FNA_OUT(23 downto 20) <= "0001" THEN FN_ADD_OUT(22 DOWNT0 3)<= FNA_OUT(19 DOWNT0 0); FN_ADD_OUT(2 DOWNT0 0)<= "000"; ZEROS <= "11111100"; CH <= '1';
ELSE IF FNA_OUT(23 downto 21) <= "001" THEN FN_ADD_OUT(22 DOWNT0 2)<= FNA_OUT(20 DOWNT0 0); FN_ADD_OUT(1 DOWNT0 0)<= "00"; ZEROS <= "11111101"; CH <= '1';
ELSE IF FNA_OUT(23 downto 22) <= "01" THEN FN_ADD_OUT(22 DOWNT0 1)<= FNA_OUT(21 DOWNT0 0); FN_ADD_OUT(0)<= '0'; ZEROS <= "11111110"; CH <= '1'; -- # OF ZEROS IS COMPLEMENTED!
ELSE IF FNA_OUT(23) <= '1' THEN FN_ADD_OUT <= FNA_OUT(22 DOWNT0 0); ZEROS <= "00000000"; CH <= '0';
ELSE FN_ADD_OUT <= "00000000000000000000000000000000"; ZEROS <= "00000000"; CH <= '0'; -- SPECIAL CASE (E-P=0)
END IF; END IF;
END IF; END IF; END IF; END IF; END IF; END IF; END IF; END IF; END IF; END IF;
ELSE
    ZEROS <= "00000000"; CH <= '0';
    IF C_OUT = '0' THEN FN_ADD_OUT <= FNA_OUT(22 DOWNT0 0);
    ELSE FN_ADD_OUT <= FNA_OUT(23 DOWNT0 1);
    END IF;
    END IF;
end process;
end Behavioral;
```

The Multiplexer of inputs to the Adder/Subtractor (OUT_MUX.vhd):

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

entity OUT_MUX is Port (
  S : in STD_LOGIC_vector (2 DOWNTO 0);
  A1,A2,B1,B2,C1,C2,D1,D2,E1,E2,F1,F2,G1,G2,H1,H2:in STD_LOGIC_VECTOR (31 downto 0);
  N1, N2 : OUT STD_LOGIC_VECTOR (31 downto 0));
end OUT_MUX;

architecture Behavioral of OUT_MUX is

begin
PROCESS(S, A1,A2, B1,B2, C1,C2, D1,D2, E1,E2, F1,F2, G1,G2, H1,H2)
begin
  case S is
    when "000" => N1 <= A1; N2 <= A2;
    when "001" => N1 <= B1; N2 <= B2;
    when "010" => N1 <= C1; N2 <= C2;
    when "011" => N1 <= D1; N2 <= D2;
    when "100" => N1 <= E1; N2 <= E2;
    when "101" => N1 <= F1; N2 <= F2;
    when "110" => N1 <= G1; N2 <= G2;
    when "111" => N1 <= H1; N2 <= H2;
    when others => N1 <= "11111111111111111111111111111111"; N2 <= "11111111111111111111111111111111";
  END CASE;
END PROCESS;
end Behavioral;
```