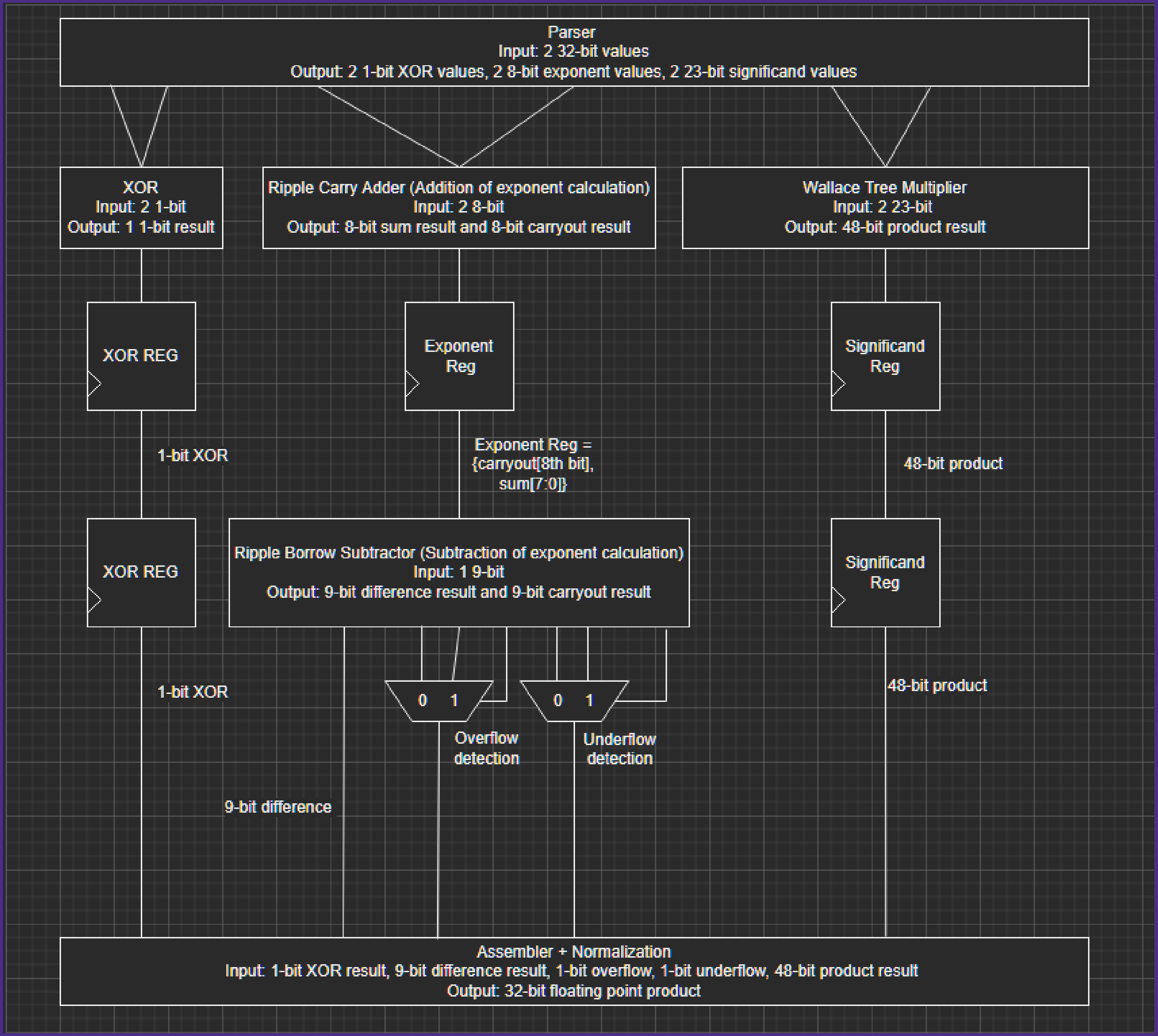
An Efficient Implementation of Floating-Point Multiplier

EE 427 Course Project

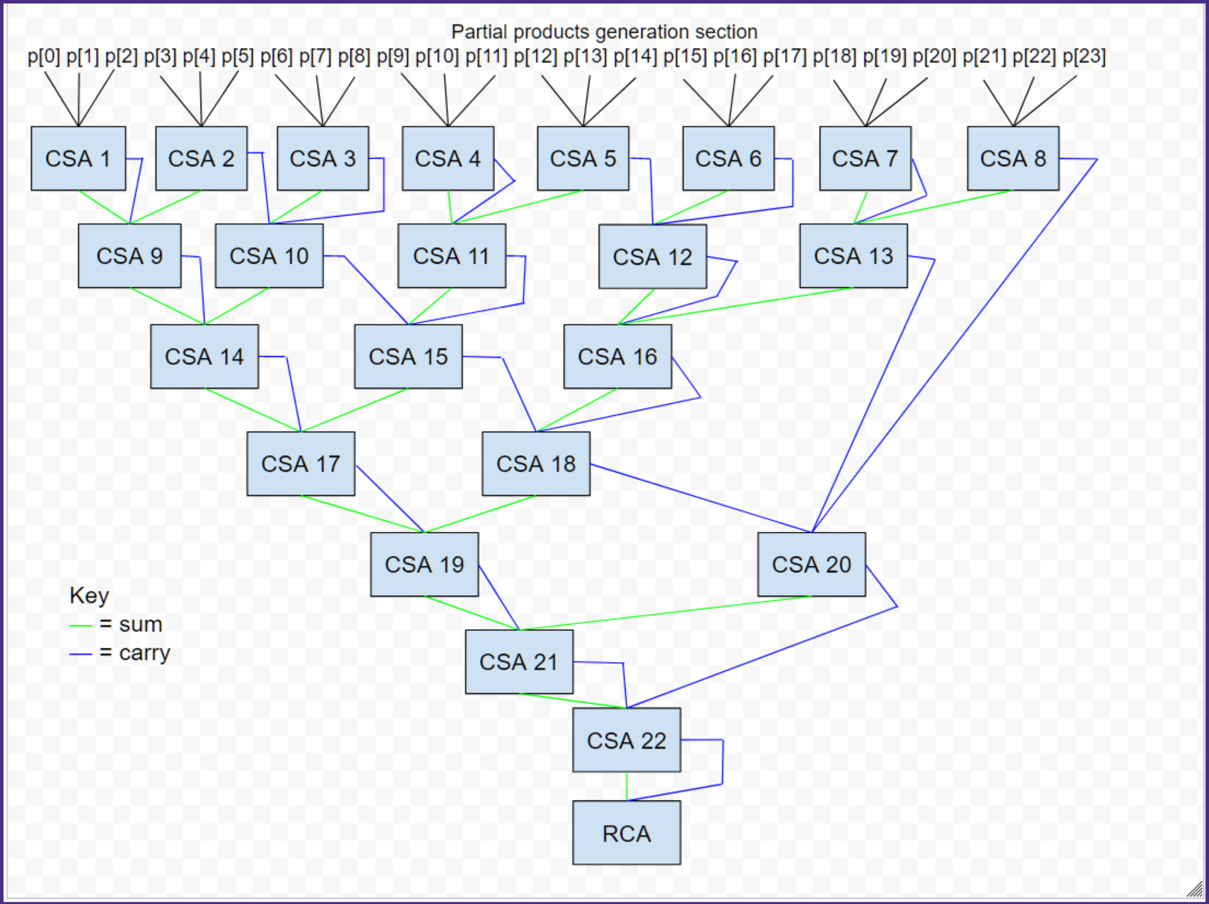
Author: Danny Kha

UNIVERSITY *of* WASHINGTON

Block Diagram of single-precision floating-point Multiplier

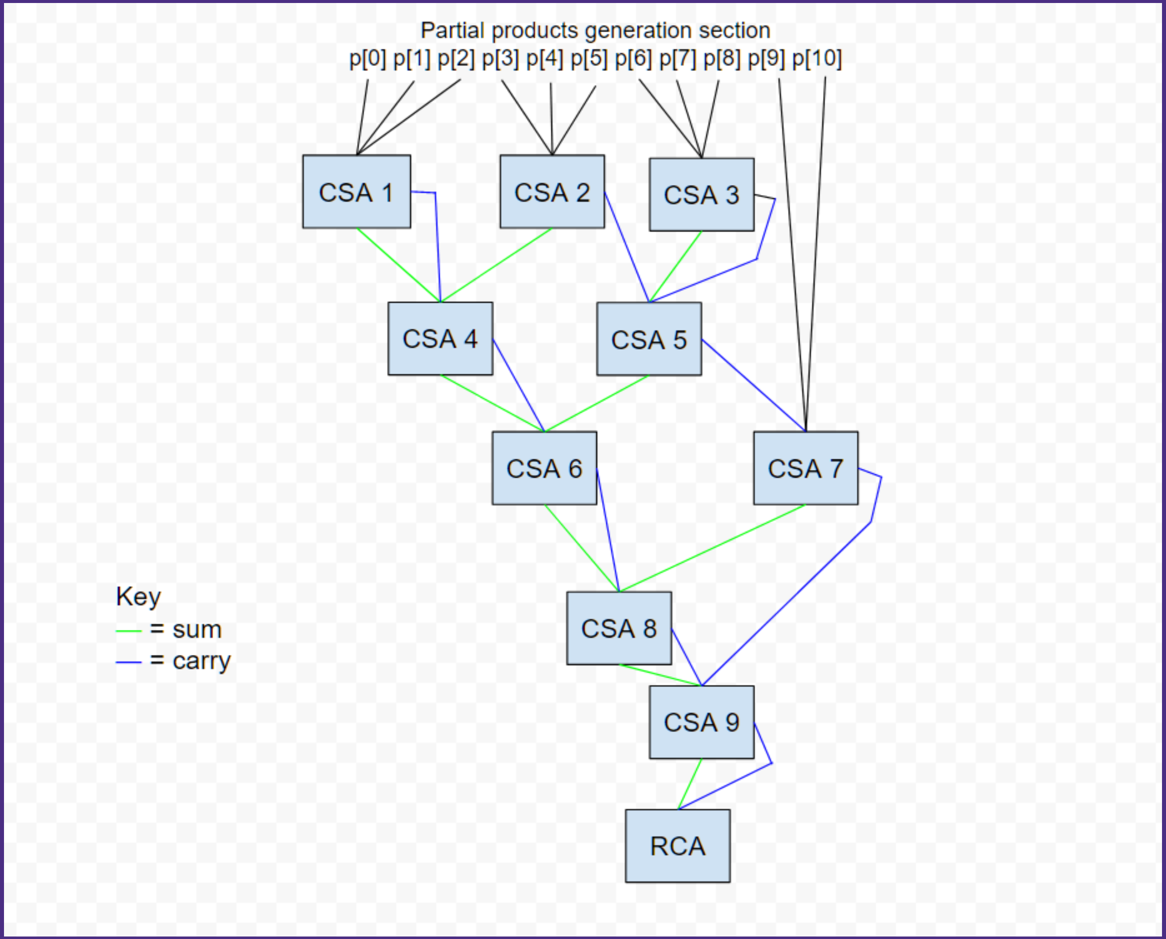

Parser
Input: 2 32-bit values
Output: 2 1-bit XOR values, 2 8-bit exponent values, 2 23-bit significand values
XOR
Input: 2 1-bit
Output: 1 1-bit result
Ripple Carry Adder (Addition of exponent calculation)
Input: 2 8-bit
Output: 8-bit sum result and 8-bit carryout result
Wallace Tree Multiplier
Input: 2 23-bit
Output: 48-bit product result
XOR REG
Exponent Reg
Significand Reg
1-bit XOR
Exponent Reg = {carryout[8th bit], sum[7:0]}
48-bit product
XOR REG
Ripple Borrow Subtractor (Subtraction of exponent calculation)
Input: 1 9-bit
Output: 9-bit difference result and 9-bit carryout result
Significand Reg
1-bit XOR
0 1
0 1
48-bit product
Overflow detection
Underflow detection
9-bit difference
Assembler + Normalization
Input: 1-bit XOR result, 9-bit difference result, 1-bit overflow, 1-bit underflow, 48-bit product result
Output: 32-bit floating point product

Wallace Tree Diagram (24 * 24)

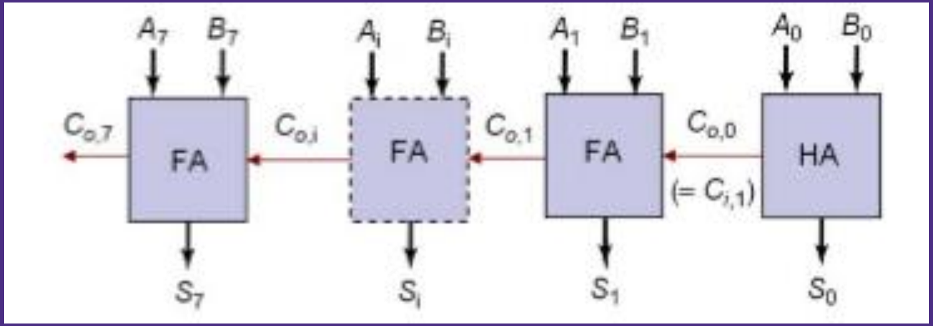


Partial products generation section
p[0] p[1] p[2] p[3] p[4] p[5] p[6] p[7] p[8] p[9] p[10] p[11] p[12] p[13] p[14] p[15] p[16] p[17] p[18] p[19] p[20] p[21] p[22] p[23]
CSA 1
CSA 2
CSA 3
CSA 4
CSA 5
CSA 6
CSA 7
CSA 8
CSA 9
CSA 10
CSA 11
CSA 12
CSA 13
CSA 14
CSA 15
CSA 16
CSA 17
CSA 18
CSA 19
CSA 20
CSA 21
CSA 22
RCA
Key
= sum
= carry

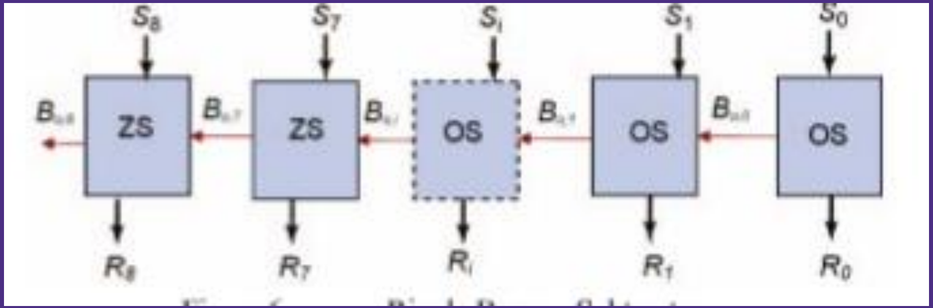
Wallace Tree Diagram (11 * 11)


Partial products generation section
p[0] p[1] p[2] p[3] p[4] p[5] p[6] p[7] p[8] p[9] p[10]
CSA 1
CSA 2
CSA 3
CSA 4
CSA 5
CSA 6
CSA 7
CSA 8
CSA 9
RCA
Key
= sum
= carry

Exponent Calculation Design

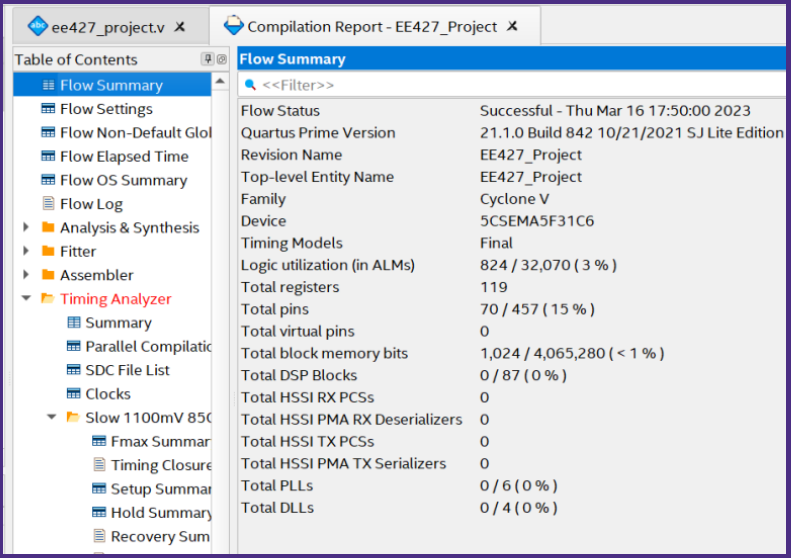


Ripple Carry Adder to preform addition



Ripple Borrow Subtractor to preform subtraction

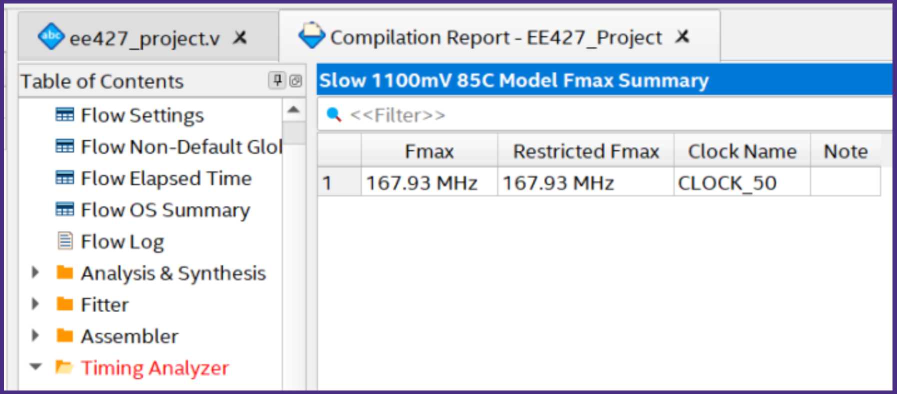
Implementation Results (Single precision floating point SPFP)



Flow Summary

Flow Status	Successful - Thu Mar 16 17:50:00 2023
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	EE427_Project
Top-level Entity Name	EE427_Project
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	824 / 32,070 (3 %)
Total registers	119
Total pins	70 / 457 (15 %)
Total virtual pins	0
Total block memory bits	1,024 / 4,065,280 (< 1 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

# of ALMs, # of Registers, and # of DSPs



Slow 1100mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	167.93 MHz	167.93 MHz	CLOCK_50	

Frequency Results

Exploratory Phase

> To explore how multiplication is preformed in hardware, I decided to implement a Wallace Tree multiplier rather than a carry save multiplier that the paper describes.

> A Wallace Tree multiplier preforms multiplication by first creating partial products with the arguments, then using the partial products in carry save adders until there are two values left, which are then added together using a conventional adder (ripple carry adder in this application).

> The benefits of a Wallace Tree is its faster speed since it has a time complexity of O(log N).

> The sign and exponent calculations were done as described by the paper.

Source:
https://en.wikipedia.org/wiki/Wallace_tree