Decimal Floating-Point Adder and Multifunction Unit with Injection-Based Rounding

Liang-Kai Wang and Michael J. Schulte
University of Wisconsin-Madison

ARITH-18, Montpellier, France

This research is supported by the UW-Madision Graduate School and IBM

Outline

- Motivation
- Related Research
- Algorithm for Decimal Floating-Point (DFP) Adder and Multifunction Unit
- Hardware Design
- Experimental Results and Analysis
- Conclusions

Motivation

- Important in business applications

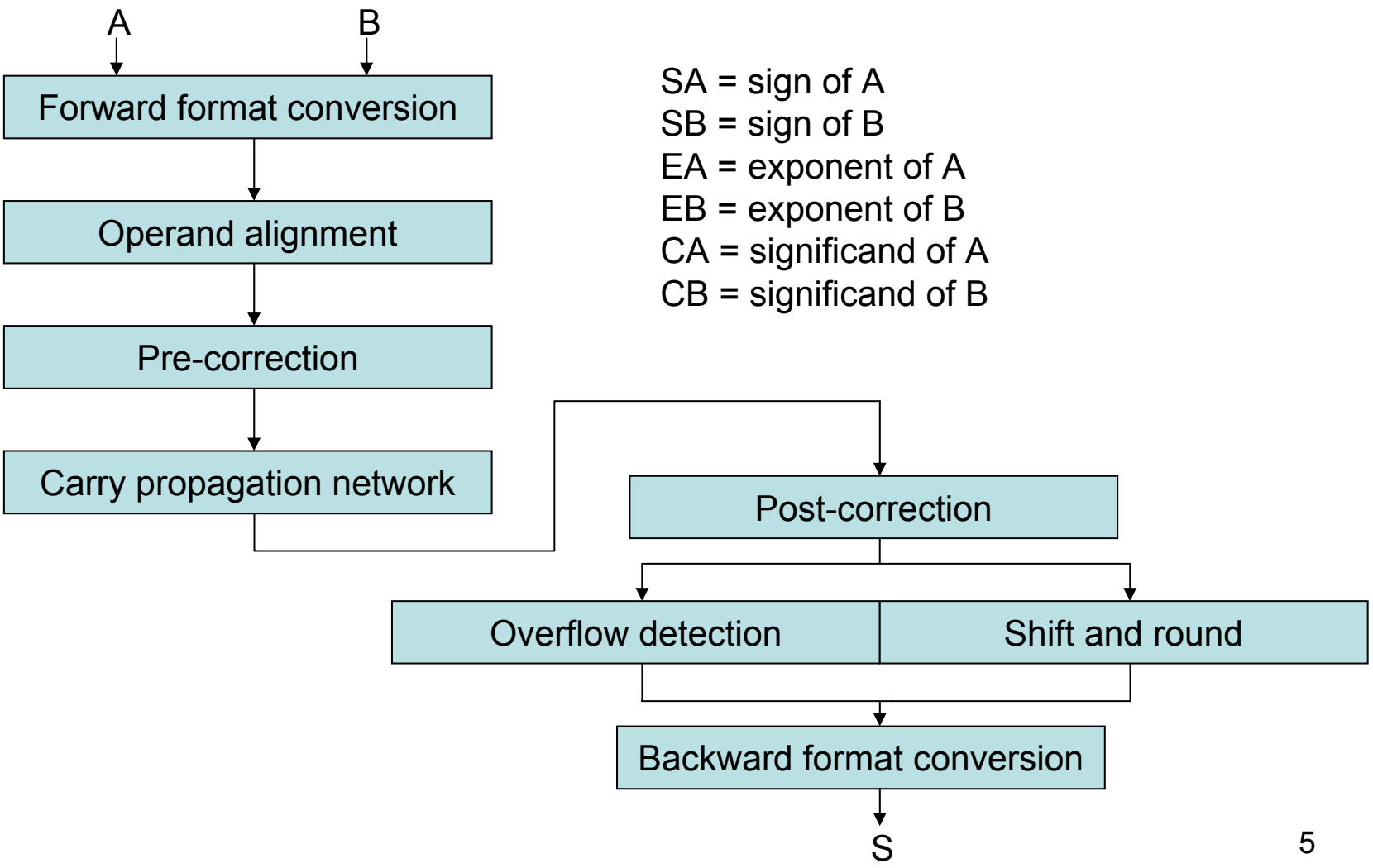
$=0.2_{10}= 0.00110011\ldots_2$

- The IEEE P754 floating-point standard
  - Three DFP formats: 34-digit decimal128 format, **16-digit decimal64 format** (this paper), and 7-digit decimal32 format
- Decimal floating-point software is slow
- Decreasing transistor costs

Previous Research and Proposed Design

- **Previous designs**
  - Focus on fixed-point addition and subtraction
    - For example, [Adiletta89], [Schmookler71]
  - [Thompson04] presents the first IEEE P754 compliant DFP adder
- **We propose an DFP multifunction unit that**
  - Supports eight DFP operations
    - add, sub, quantize, sameQuantum, roundToIntegral, minNum, maxNum, and compare
  - Optimizes significand alignment
  - Applies decimal injection-based rounding
  - Uses a decimal flag-tracing mechanism

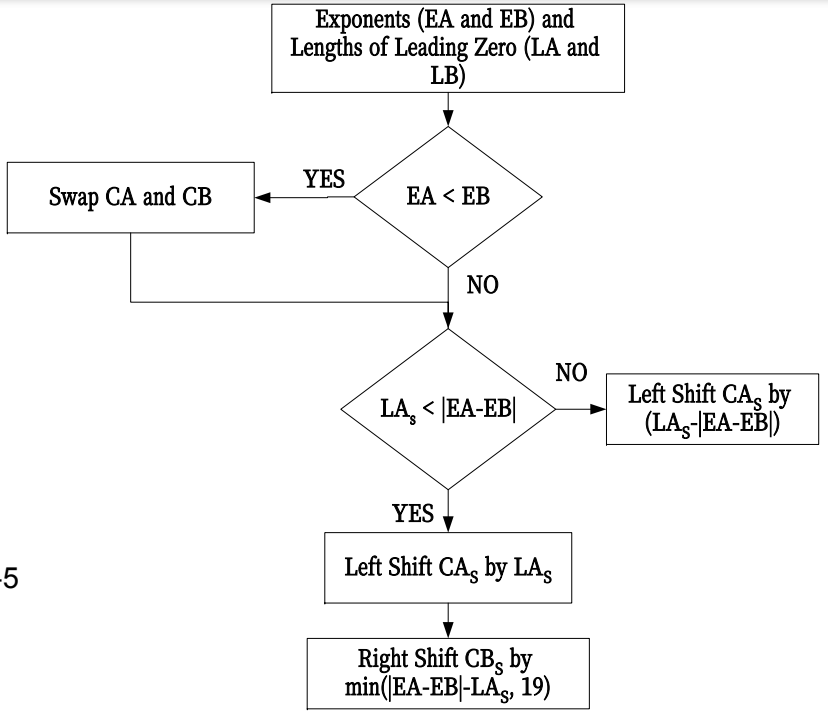
DFP Adder and Multifunction Unit

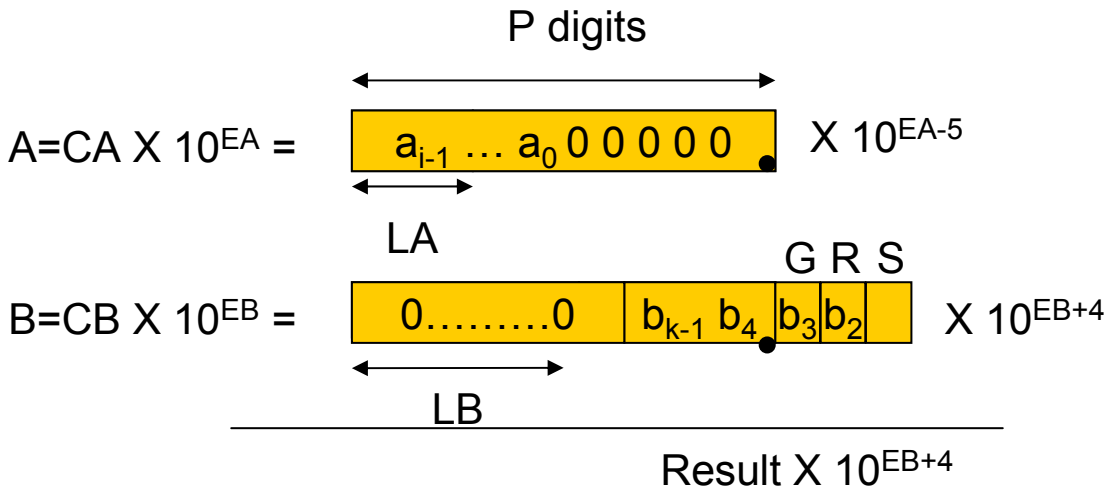


SA = sign of A
SB = sign of B
EA = exponent of A
EB = exponent of B
CA = significand of A
CB = significand of B

Operand Alignment

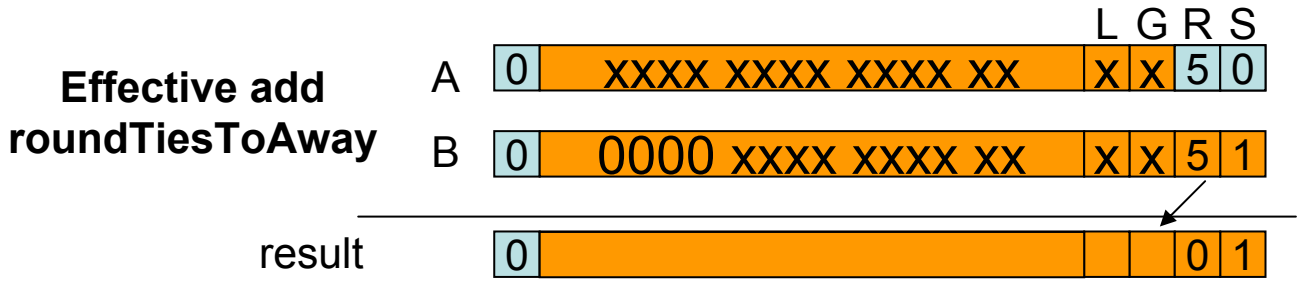
- Decimal operands are not normalized
- Operand alignment calculation
- E.g. LA = 5 , EA – EB = 9





Pre-correction

- Effective operation = SA⊕SB⊕OP
- Place operands based on effective operations simplifies result shifting
- Inject value into the digit positions, R and S, based on rounding modes replaces rounding by truncation.



Pre-correction

- Injection value

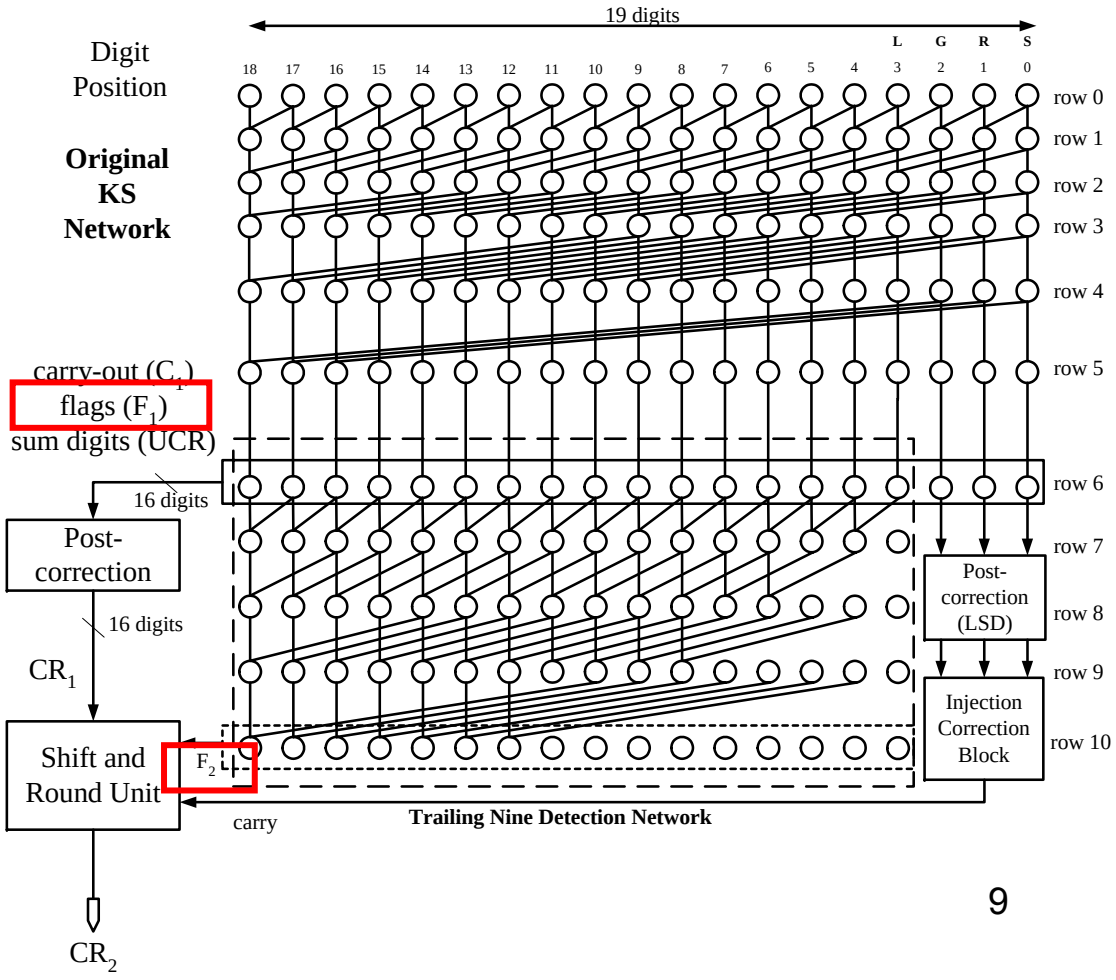
$Sign_{inj}$	Rounding Mode	Injection Value (R, S)
X	TowardZero	(0, 0)
X	TieToAway	(5, 0)
X	TieToZero	(4, 9)
X	TieToEven	(5, 0)
-	+∞	(0, 0)
+	-∞	(9, 9)
-	+∞	(9, 9)
+	-∞	(0, 0)
X	AwayZero	(9, 9)

- Operands are corrected to generate correct carry-out

$$(CA_3)_i = \begin{cases} (CA'_2)_i + 6 & \text{If EOP = add} \\ (CA'_2)_i & \text{Otherwise} \end{cases} \qquad (CB_3)_i = \begin{cases} (CB'_2)_i & \text{If EOP = add} \\ \overline{(CB'_2)_i} & \text{Otherwise} \end{cases}$$

Carry Propagation Network

- Kogge-Stone parallel prefix network
- Two sets of flags
  - Flag F_1 handles the digit increment in the post-correction stage.
  - Flag F_2 handles the carry propagation from the injection correction value.

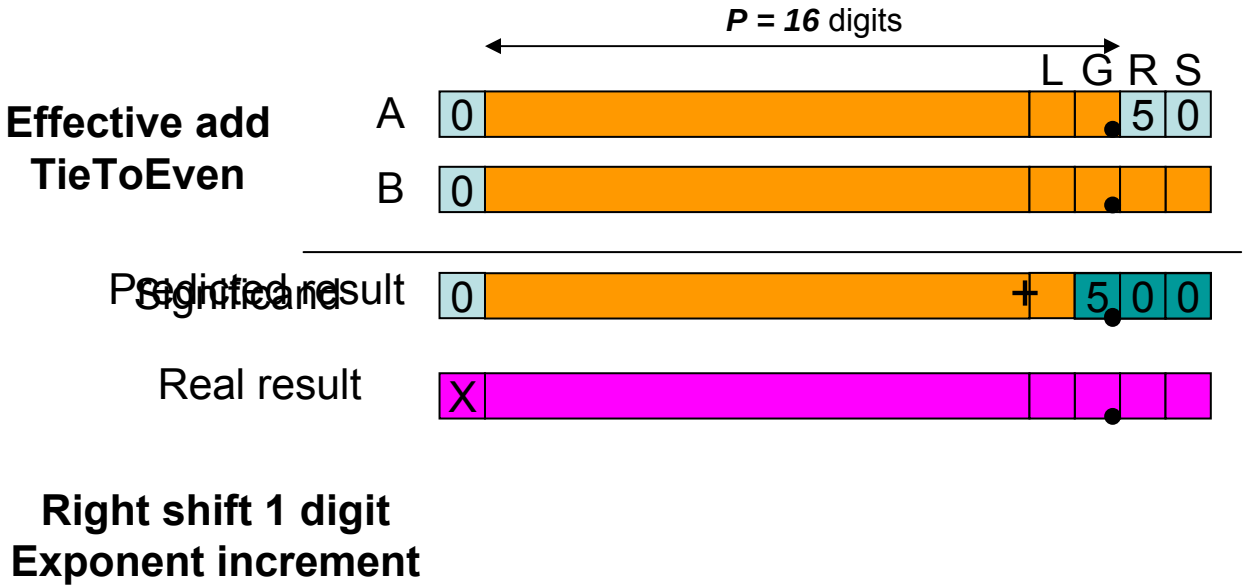


Post-correction

- Compensate the result from the K-S network
- Rule 1: effective operation is ADD
  - Subtract 6 from digit ***i*** for which $(C_1)_{i+1}$ is 0
- Rule 2: effective operation is SUB
  - If the result is positive
    - Increment the result using F_1
    - Subtract 6 from digit ***i*** for which $(C_1)_{i+1} \oplus (F_1)_i \equiv 0$
  - If the result is negative
    - Invert all bits of the result
    - Subtract 6 from digit ***i*** for which $(C_1)_{i+1} \equiv 1$

Shift and Round

- Most significant digit is zero
  - No action is needed
- Most significant digit is non-zero
  - Requires an injection correction step



Shift and Round

- Injection correction value for different rounding modes

$Sign_{inj}$	Rounding Mode	Injection Correction Value (G, R, S)
X	TowardZero	(0, 0, 0)
X	TieToAway	(4, 5, 0)
X	TieToZero	(4, 5, 0)
X	TieToEven	(4, 5, 0)
-	+∞	(0, 0, 0)
+	-∞	(9, 0, 0)
-	+∞	(9, 0, 0)
+	-∞	(0, 0, 0)
X	AwayZero	(9, 0, 0)

- Injection correction value may trigger carry propagation
- Flag F_2 eliminates carry propagation

Comparison

	Thompson's Design	This Design
Supported DFP Operations	2: add, subtract	8: add, subtract, minNum, maxNum, compare, quantize, sameQuantum, roundToIntegral
Internal format	Excess-3 encoding	BCD encoding
Operand Alignment	Exponent computation and LZD in series	Exponent computation and LZD in parallel
Carry-propagate network	Kogge-Stone with flag tracing for post-correction	Two extra flags for rounding
Rounding	Random logic and decimal incrementer.	Injection-based rounding with correction.
Overflow Detection	After result is rounded	Before the result is rounded

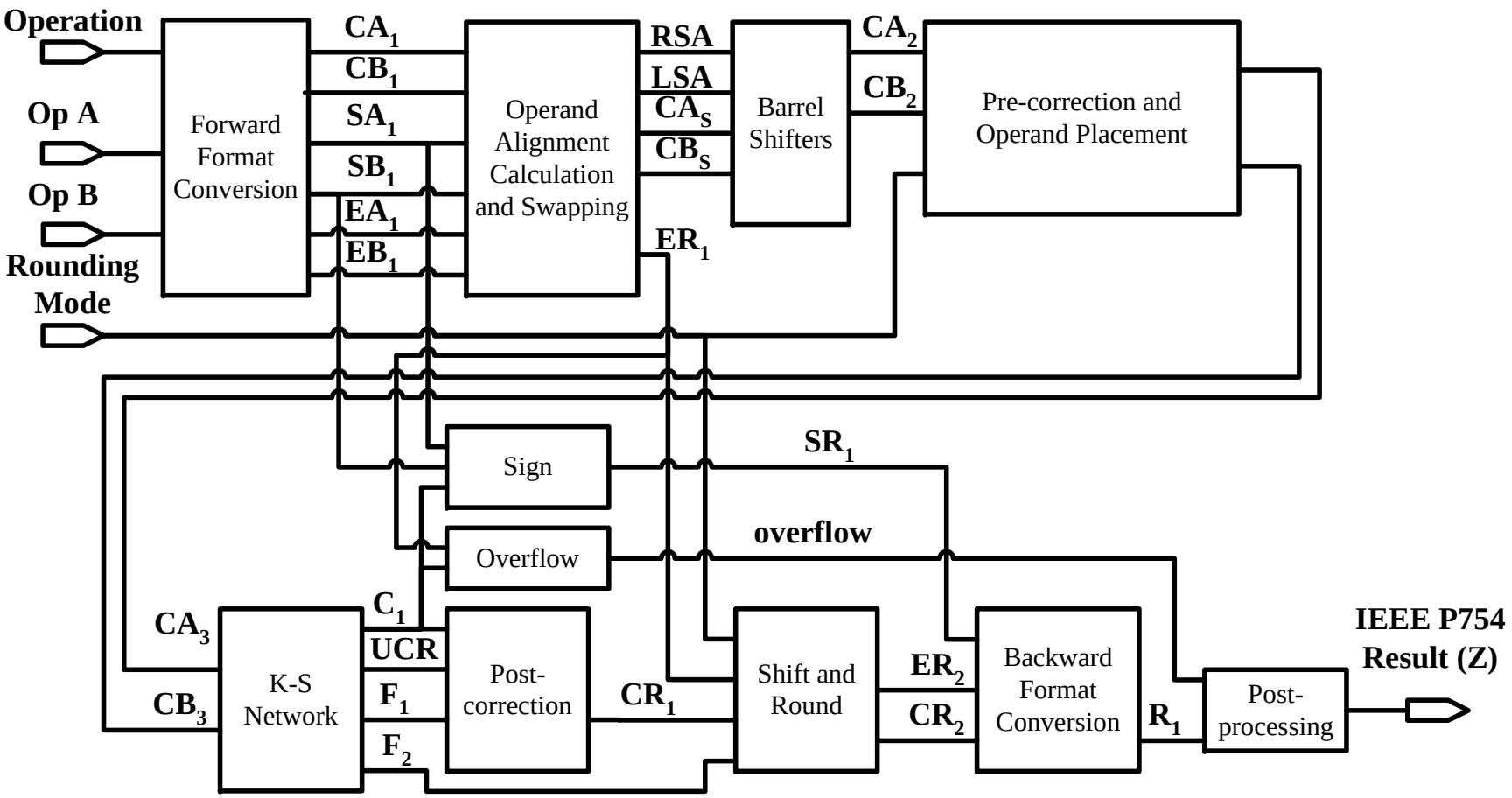
Extension to Support More DFP Operations

- ToIntegralValue(A)
  - Round A to an integer value
    - ToIntegralValue(13545 x 10^{-3}) = 14 with round-ties-to-even
  - Design strategy
    - Set CB_1 and EB_1 to zero
    - Enable right shift even if CB_1=0
    - Set effective operation to ADD
- Quantize (A, B)
  - Change EA to EB
    - Quantize(12345 x 10^{-4}, 1 x 10^{-2}) = 123 x 10^{-2} with round-down
  - Design strategy
    - Set CB_1 to zero
    - Enable right shift even if CB_1=0
    - Set effective operation to ADD

Extension to Support More DFP Operations

- SameQuantum(A, B)
  - Check if EA ≡ EB
  - Generate an extra flag in the operand alignment stage
- minNum, maxNum, and compare use the original datapath
- Many changes are made to exception flag logic
- A post-processing unit is added to handle special operands such as infinity and Not-a-Number

Block Diagram of the DFP Adder and Multifunction Unit



Hardware Implementation

- Modeled using RTL Verilog and simulated using Modelsim
- Synthesized using LSI Logic's 0.11um Standard Cell Library and Synopsys Design Compiler
- Tested using a comprehensive testbench generator and the decNumber library 3.32

Delay and Area Comparison

- Combinational circuit designs

Metric	Thompson's adder	Injection-based adder	Improvement
Delay (comb.)	3.50 ns, 63.6 FO4	2.76 ns, 50.2 FO4	21.0%
Area	22443 NAND eq. gates	22086 NAND eq. gates	1.6%

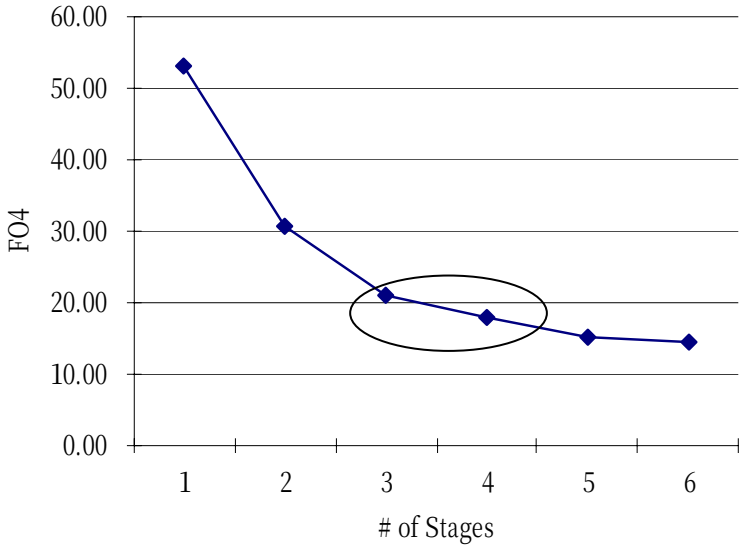
Table 1. Improvement over Thompson's Design

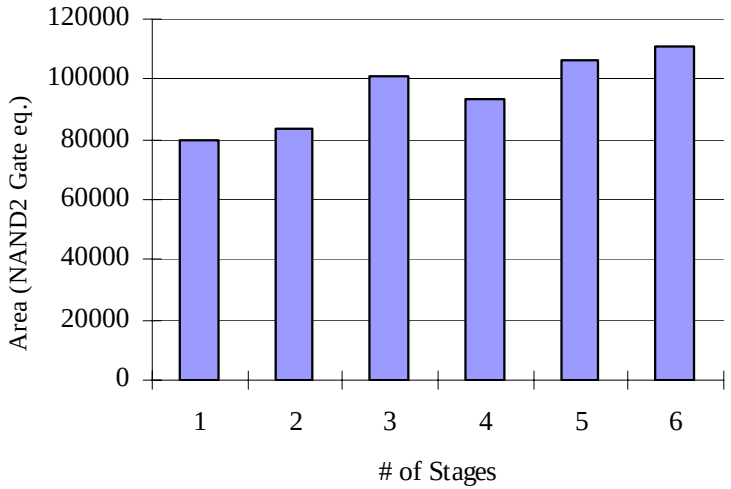
Metric	Injection-based adder	Multifunction Unit	Overhead
Delay	2.76 ns, 50.2 FO4	2.84ns, 51.6 FO4	2.8%
Area	22086 NAND eq. gates	24233 NAND eq. gates	9.7%

Table 2. Overhead of the Multifunction Unit Compared to the Injection-based Adder

Cycle Times vs. Pipeline Depth

- Synthesized using the ***pipeline_design*** command from the Synopsys Design Compiler





Conclusion

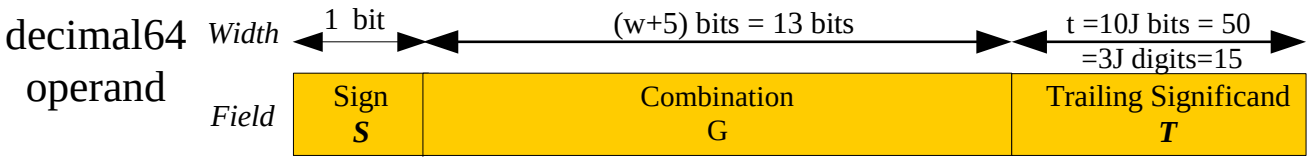
- A 16-digit DFP adder and multifunction unit compliant with the IEEE P754 standard
- Novel features:
  - Delay optimization in the operand alignment, rounding, and overflow detection units
  - A modified injection-based rounding method
  - Extensions to support multiple DFP operations
- Design analysis
  - 21% delay improvement over Thompson's design
  - 2.8% delay overhead for DFP multifunction unit

Questions?

Backup Slide

- More on Forward Conversion
- More on Operand Alignment
- More on Post-correction
- More on Carry Propagation Network
- More on Overflow Detection
- More on Sign and Backward Conversion
- More on Extension to Support More DFP Operations
- More on Area Comparison

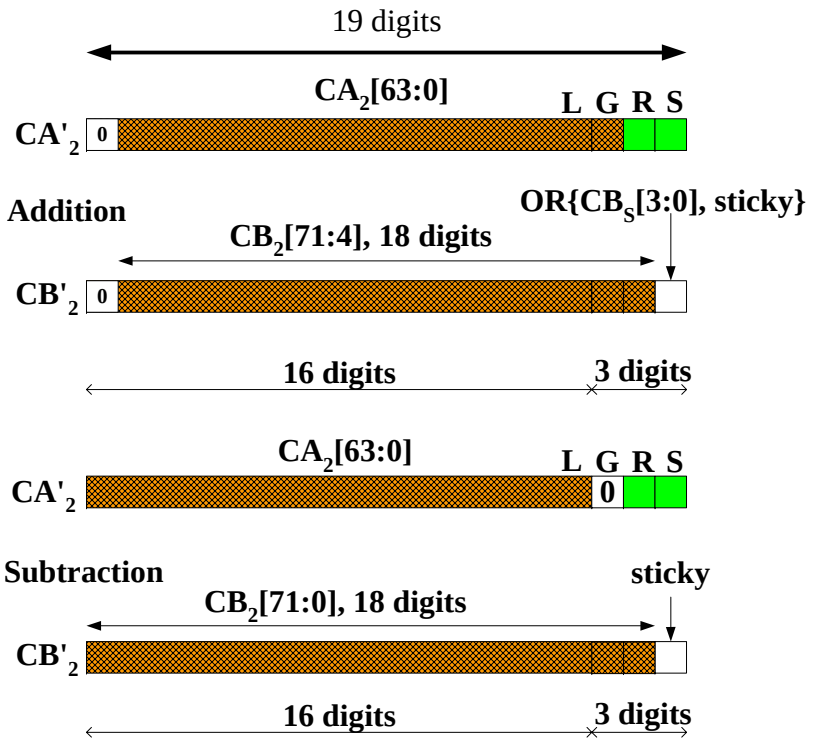
Forward Format Conversion



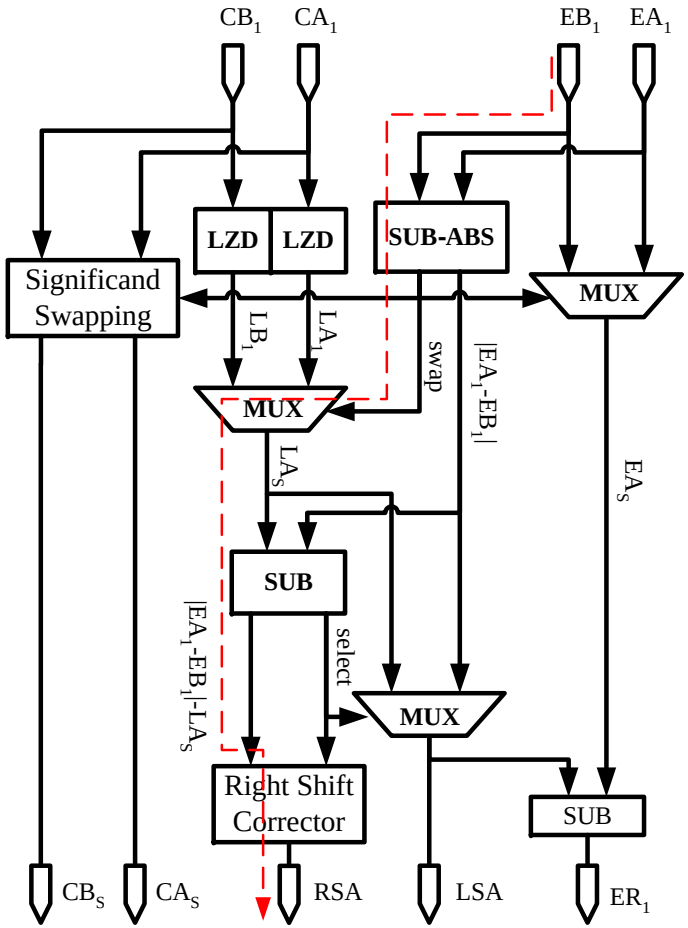
- Extract sign bits, biased exponents, and significands from operands in the IEEE format
  - Combination field ***G*** contains the classification of a number, the encoding information, the most significant digit of significand and a biased exponent.
  - Trailing significand field ***T*** encodes a significand using Densely Packed Decimal (DPD) encoding. DPD encoding represents three digits using ten bits.
- Convert significands in DPD encoding to the BCD encoding
- Generate flag signals for special operands (signaling NaN, quiet NaN, zero, and infinity)

Operand Alignment and Pre-correction

- Operands are shifted using one 16-digit left-shift and one 18-digit right-shift decimal barrel shifters.
- Guard and round digits, and sticky bit are generated. CB becomes a 18-digit operand with a sticky bit.
- Operands are placed based on the effective operation flag to simplify the rounding.



Operand Alignment and Pre-correction



Validity of Post-correction

- Add:
  - At Pre-correction: $A_i + B_i + 6$
  - If digit carry is 0, $A_i + B_i + C_{i-1} < 10$, subtract 6 from Sum_i
- Sub:
  - Expect: A + (10…0 - B)
  - At Pre-correction: A + (9…9- B) + 6…6
  - If carry out of MSD is 1,
    - Result is positive. Add the late carry-in from the LSD.
    - If the digit sum after incrementing the late carry-in is less than 10 (A + (9-B) + 6 + C < 10), subtract 6 from Sum

Validity of Post-correction

- Else
  - Result is negative. Invert Sum. $Sum_i = 15 – (A_i + 15 – B_i + C_{i-1}) = B_i – A_i – C_{i-1}$
  - If $B_i – (A_i + C_{i-1}) < 0$
    - Need to borrow from the next digit
    - $25 >= 15-[B_i – (A_i + C_{i-1})] >= 16$ ➔ $9 >= Sum >= 0$. This generates a carry to the next digit.
    - After inverting, $F >= Sum_i' >= 6$. Need to subtract 6
  - Else,
    - No borrow from the next digit
    - $15 >= 15-[B_i – (A_i + C_{i-1})] > 6$, No carry is generated
    - After inverting, $9 >= Sum_i' >= 0$. No subtraction is needed.
- E.g 135 – 424 = 135 + bdb = d10 with 011 as borrow signals. After inversion, d10➔2ef. Subtract by six on two LSDs, 2ef➔289

Carry Propagation Network

- Use Kogge-Stone parallel prefix network
- Three sets of flags in addition to the carry bits are generated.
- Flag F_1 handles the digit increment in the post-correction stage to increment results and is generated from the propagate bits.

$$(P_x)_i = (P_{x-1})_i \wedge (P_{x-1})_{i-2^x} \text{ where } x = 4...0$$

$$F_1 = (P_4)_i$$

- Flags F_2 traces the trailing nine of the result before the post-correction stage.

$$i = 19...4$$

$$(flagADD_0)_i = ((UCR)_i \equiv 15) \vee ((UCR)_i \equiv 9) \wedge ((C_1)_{i+1})$$

$$(flagSUB_0)_i = \begin{cases} ((UCR)_4 \equiv 15) \wedge (P_3 \equiv 0) \vee ((UCR)_4 \equiv 14) \wedge (P_3 \equiv 1) \\ ((UCR)_i \equiv 15), \text{i} = 19...5 \end{cases}$$

$$(flagADD_x)_i = (flagADD_{x-1})_i \wedge (flagADD_{x-1})_{i-2^{x-1}} \text{ , where X = 1~4}$$

$$(flagSUB_x)_i = (flagSUB_{x-1})_i \wedge (flagSUB_{x-1})_{i-2^{x-1}}$$

$$F_2 = \begin{cases} flagADD_4 & \text{EOP = ADD} \\ flagSUB_4 & \text{EOP = SUB} \end{cases}$$

Overflow Detection

- Injection-based rounding simplifies the overflow detection
  - The result is overflow before rounding (carry-out is generated from the most significant digit of the result)
    - Not influenced by the injection correction value
  - The result is overflow after rounding
    - Handle by the injected value
  - Overflow detection can examine the result before the rounding unit

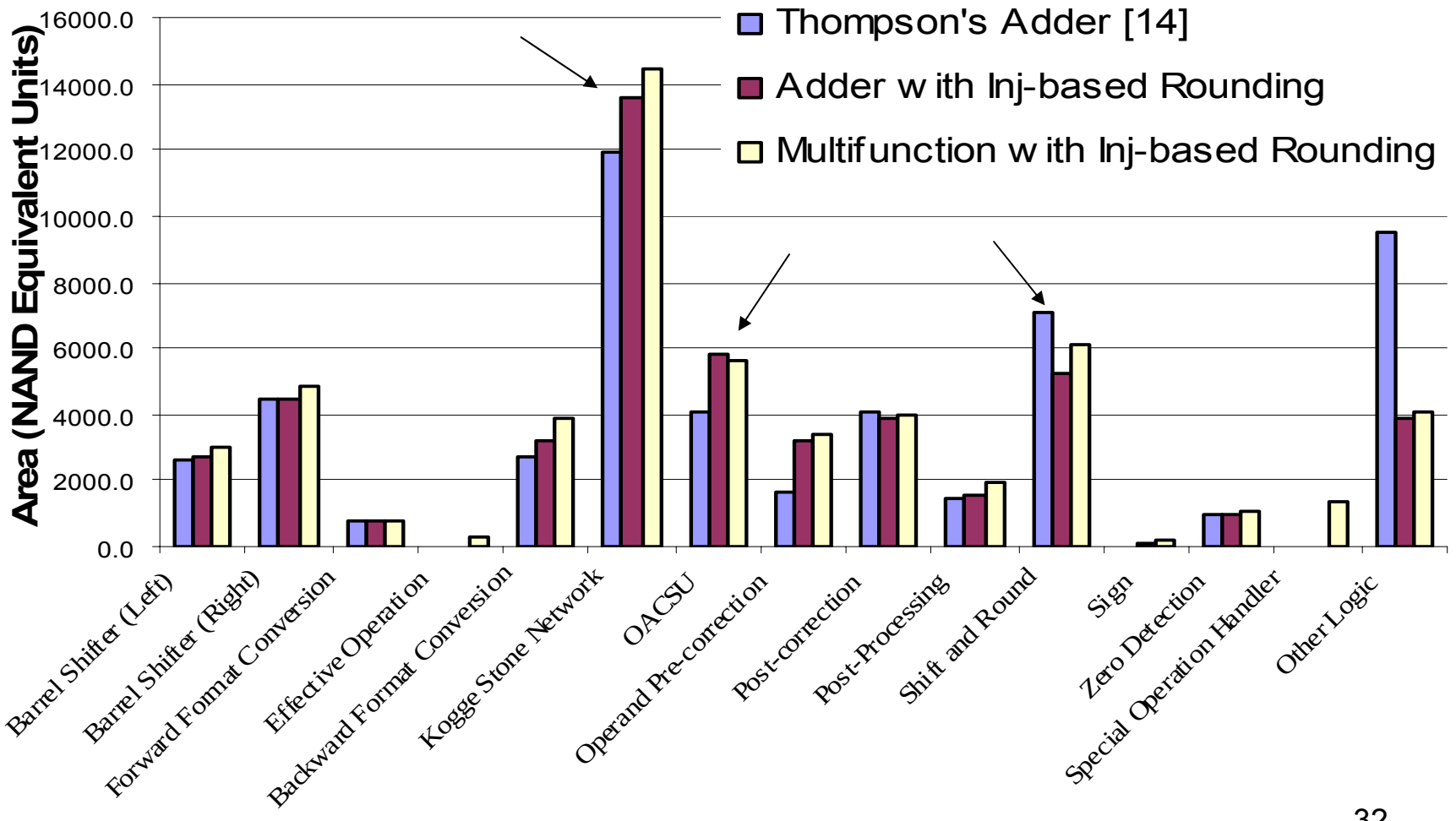
Sign and Backward Conversion

- Sign bit is determined by the signs of operands, the rounding mode, and if either of the operands is normal numbers.
  - Sign = (!EOP $\cap$ SignA) $\cup$ (EOP $\cap$ ((EA$\geq$EB) $\oplus$ SignA $\oplus$ carryout)
- Backward conversion combines the sign bit, the exponent, and the significand to form the P754 compliant result.

Extension to Support More DFP Operations

- Quantize (A, B)
  - Change the unit of A to EB
  - Set CB to zero
  - Enable right shift even if CB=0
  - Set effective operation to ADD to avoid wrong rounding operations
- SameQuantum(A, B)
  - Check if EA ≡ EB
  - Generate an extra flag in the operand alignment stage.
- MinNum, MaxNum, and Compare
  - Set the operator to SUB and observe the sign
- ToIntegralValue(A)
  - Round A to an integer value
  - Set CB and EB to zero
  - Enable right shift even if CB=0
  - Set effective operation to ADD to avoid wrong rounding operations
- Many changes to the conditions of exception flags are added. The post-processing unit is added to handle special operands such as infinity and Not-a-Number.

Area Comparison



Comparison with Software (IBM's decNumber library)

- decNumber library using the SimpleScalar simulator with PISA architecture

DFP Operations	Software	Hardware		Improvement	
		Fast	Slow	Fast	Slow
ADD	499.4	4	6	124.9	83.2
SUB	496.2			124.1	82.7
Quantize	265.4			66.4	44.2
SameQuantum	89.4	1	1	89.4	89.4
ToIntegralValue	364.6	4	6	91.2	60.8
MaxNum	405.4			101.4	67.6
MinNum	643.1			160.8	107.2
Compare	282.8			70.7	47.1

Comparison with Software (Intel's BID library)

- Intel's BID library and EM64t Xeon 5100 3.0GHz
- Results taken from their paper

DFP Operations	Software		Hardware		Improvement	
	Fast	Slow	Fast	Slow	Max	Min
ADD	71	133	4	6	33.3	11.8
SUB	71	133			33.3	11.8
Quantize	27	45			11.3	4.5
ToIntegralValue	27	45	4	6	11.3	4.5
MaxNum*	75	113			28.3	12.5
MinNum*	69	108			27	11.5

Other DFP Operations

- Other DFP operations that can reuse our DFP adders include:
  - nextUp
  - nextDown
- Other DFP operations that can use our DFP with little extra gate
  - ABS
  - Negate
  - copySign

Delay Comparison