FLOATING POINT

ADDERS AND MULTIPLIERS

Lecture #4

In this lecture we will go over the following concepts:

1) Floating Point Number representation
2) Accuracy and Dynamic range; IEEE standard
3) Floating Point Addition
4) Rounding Techniques
5) Floating point Multiplication
6) Architectures for FP Addition
7) Architectures for FP Multiplication
8) Comparison of two FP Architectures
9) Barrel Shifters

- Single and double precision data formats of IEEE 754 standard

Sign *S*	8 bit - biased Exponent *E*	23 bits - unsigned fraction P

(a) IEEE single precision data format

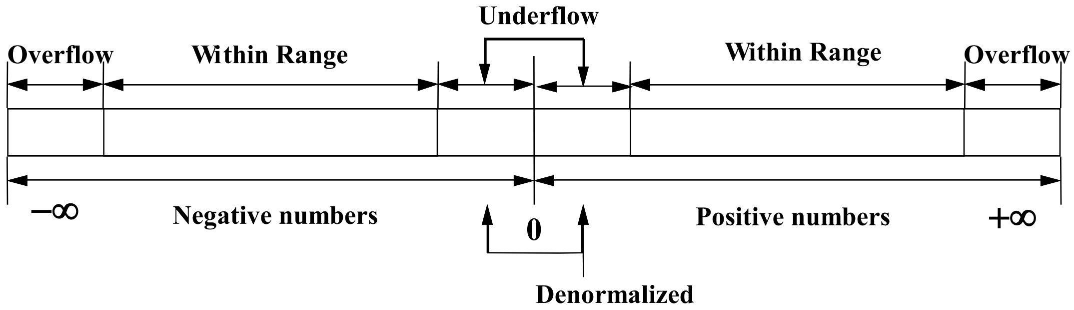
Sign *S*	11 bit - biased Exponent *E*	52 bits - unsigned fraction p

(b) IEEE double precision data format

Format parameters of IEEE 754 Floating Point Standard

Parameter	Format	
	Single Precision	Double Precision
Format width in bits	32	64
Precision (p) = fraction + hidden bit	23 + 1	52 + 1
Exponent width in bits	8	11
Maximum value of exponent	+ 127	+ 1023
Minimum value of exponent	-126	-1022

- Range of floating point numbers



Exceptions in IEEE 754

Exception	Remarks
Overflow	Result can be $\pm\infty$ or default maximum value
Underflow	Result can be 0 or denormal
Divide by Zero	Result can be $\pm\infty$
Invalid	Result is NaN
Inexact	System specified rounding may be required

- Operations that can generate Invalid Results

Operation	Remarks
Addition/ Subtraction	An operation of the type $\infty \pm \infty$
Multiplication	An operation of the type 0 x ∞
Division	Operations of the type 0/0 and ∞/∞
Remainder	Operations of the type x REM 0 and ∞ REM y
Square Root	Square Root of a negative number

IEEE compatible floating point multipliers

- ***Algorithm***

Step 1

Calculate the tentative exponent of the product by adding the biased exponents of the two numbers, subtracting the bias, (). bias is 127 and 1023 for single precision and double precision IEEE data format respectively

Step 2

If the sign of two floating point numbers are the same, set the sign of product to '+', else set it to '-'.

Step 3

Multiply the two significands. For p bit significand the product is 2p bits wide (p, the width of significand data field, is including the leading hidden bit (1)). Product of significands falls within range .

Step 4

Normalize the product if MSB of the product is 1 (i.e. product of), by shifting the product right by 1 bit position and incrementing the tentative exponent.

Evaluate exception conditions, if any.

Step 5

Round the product if R(M0 + S) is true, where M0 and R represent the pth and (p+1)st bits from the left end of normalized product and Sticky bit (S) is the logical OR of all the bits towards the right of R bit. If the rounding condition is true, a 1 is added at the pth bit (from the left side) of the normalized product. If all p MSBs of the normalized product are 1's, rounding can generate a carry-out. In that case normalization (step 4) has to be done again.

Operands Multiplication and Rounding

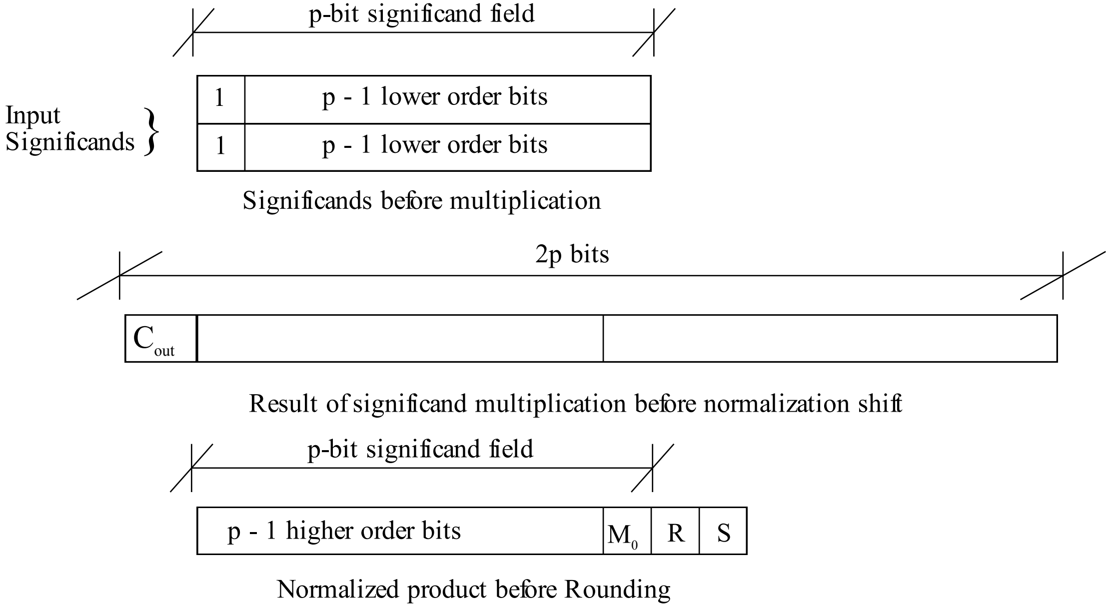
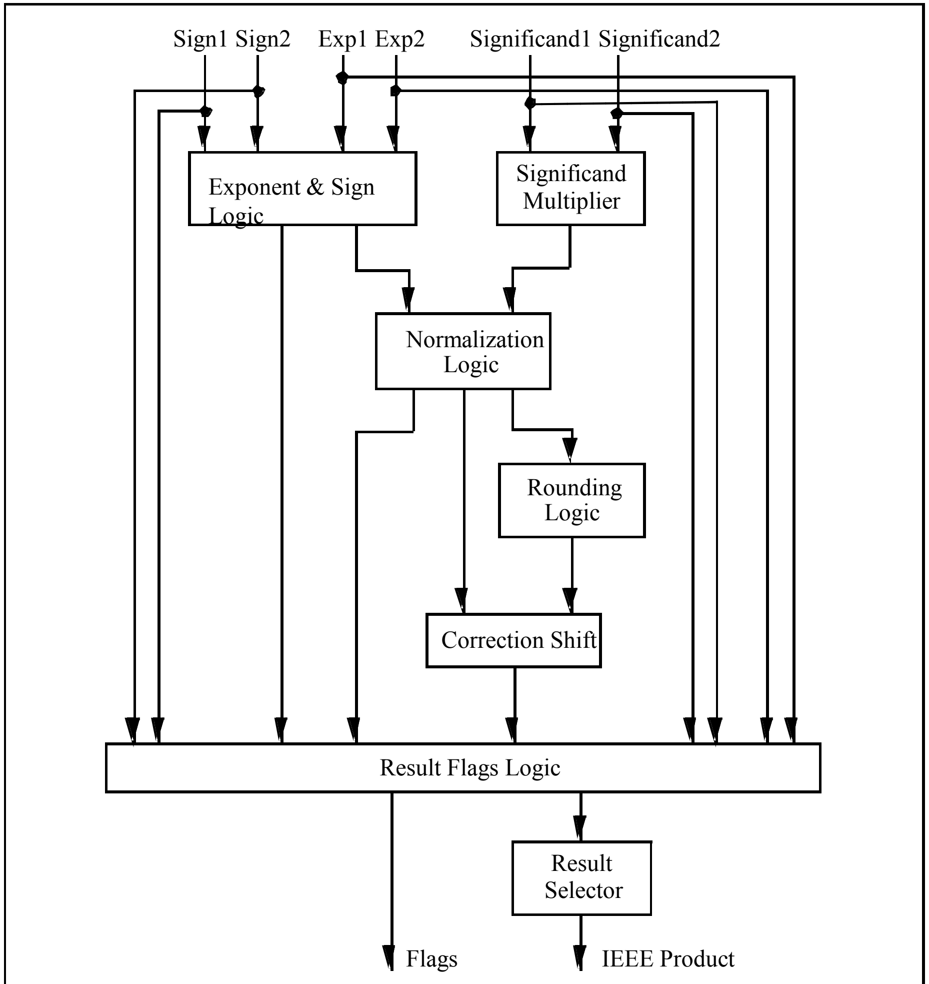


Figure 2.4 - Significand multiplication, normalization and rounding

What's the
best
architecture?

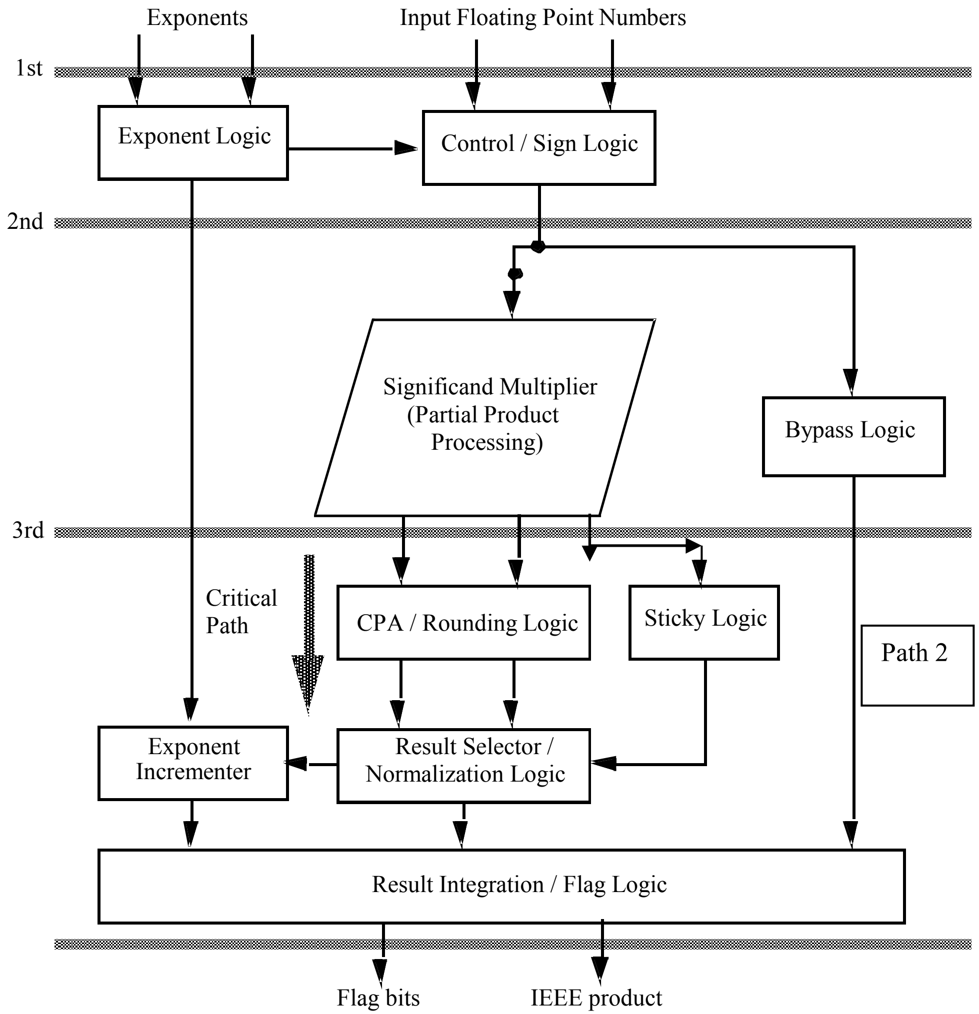
Architecture Consideration

A Simple FP Multiplier

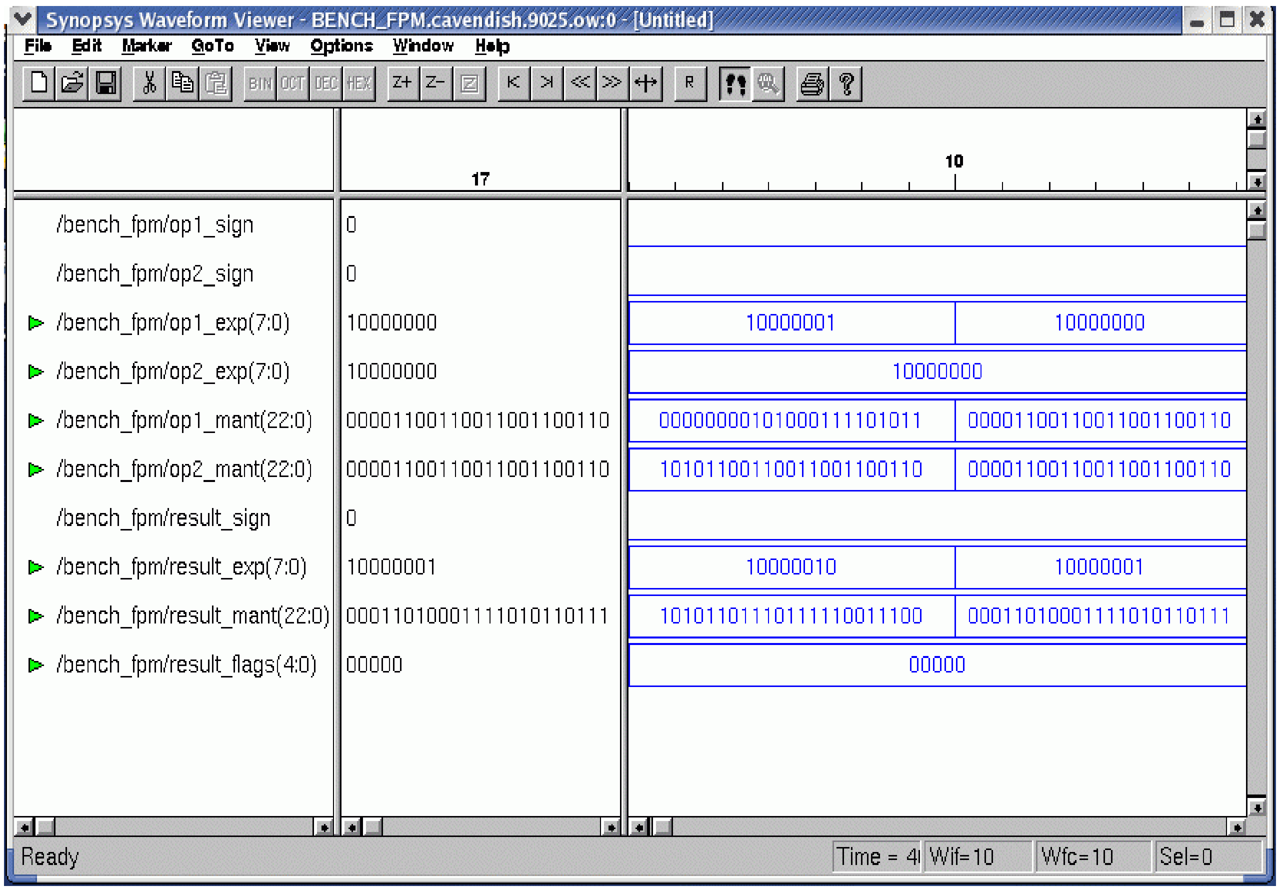


A Dual Path FP Multiplier

Concordia University



Case-1 Normal Number				
	Operand1	0	10000001	00000000101000111101011
	Operand2	0	10000000	10101100110011001100110
	Result	0	10000010	10101101110111110011100
Case-2 Normal Number				
	Operand1	0	10000000	00001100110011001100110
	Operand2	0	10000000	00001100110011001100110
	Result	0	10000001	00011010001111010110111



Comparison 0f 3 types of FP Multipliers using 0.22 micron CMOS technology

	AREA (cell)	POWER (mW)	Delay (ns)
Single Data Path FPM	2288.5	204.5	69.2
Double Data Path FPM	2997	94.5	68.81
Pipelined Double Data Path FPM	3173	105	42.26

IEEE compatible floating point adders

- ***Algorithm***

Step 1

Compare the exponents of two numbers for (or) and calculate the absolute value of difference between the two exponents (). Take the larger exponent as the tentative exponent of the result.

Step 2

Shift the significand of the number with the smaller exponent, right through a number of bit positions that is equal to the exponent difference. Two of the shifted out bits of the aligned significand are retained as guard (G) and Round (R) bits. So for p bit significands, the effective width of aligned significand must be p + 2 bits. Append a third bit, namely the sticky bit (S), at the right end of the aligned significand. The sticky bit is the logical OR of all shifted out bits.

Step 3

Add/subtract the two signed-magnitude significands using a p + 3 bit adder. Let the result of this is SUM.

Step 4

Check SUM for carry out (C_{out}) from the MSB position during addition. Shift SUM right by one bit position if a carry out is detected and increment the tentative exponent by 1. During subtraction, check SUM for leading zeros. Shift SUM left until the MSB of the shifted result is a 1. Subtract the leading zero count from tentative exponent.

Evaluate exception conditions, if any.

Step 5

Round the result if the logical condition R''(M_0 + S'') is true, where M_0 and R'' represent the pth and (p + 1)st bits from the left end of the normalized significand. New sticky bit (S'') is the logical OR of all bits towards the right of the R'' bit. If the rounding condition is true, a 1 is added at the pth bit (from the left side) of the normalized significand. If p MSBs of the normalized significand are 1's, rounding can generate a carry-out. in that case normalization (step 4) has to be done again.

Floating Point Addition of Operands with Rounding

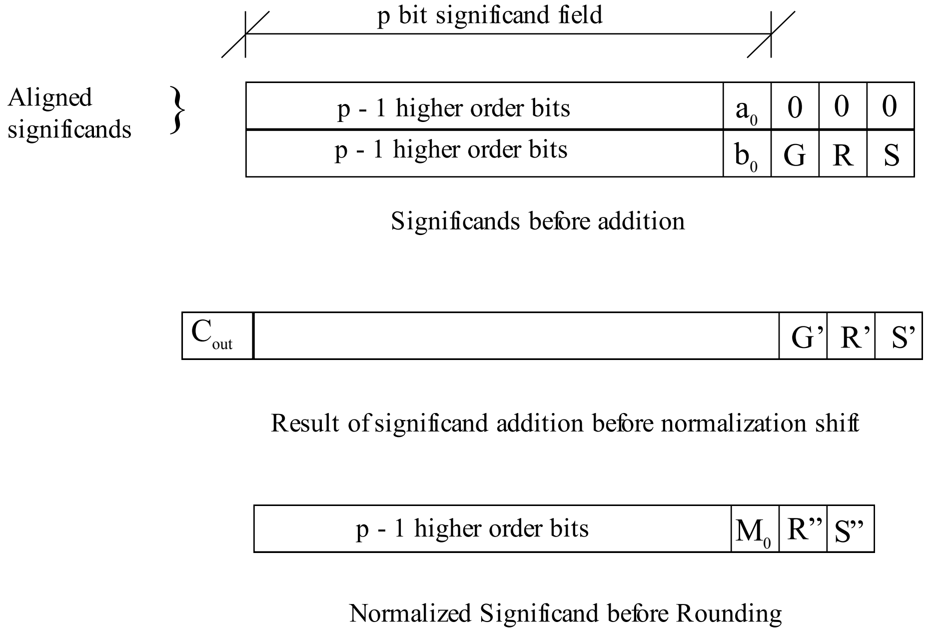


Fig 2.6 - Significand addition, normalization and rounding

IEEE Rounding

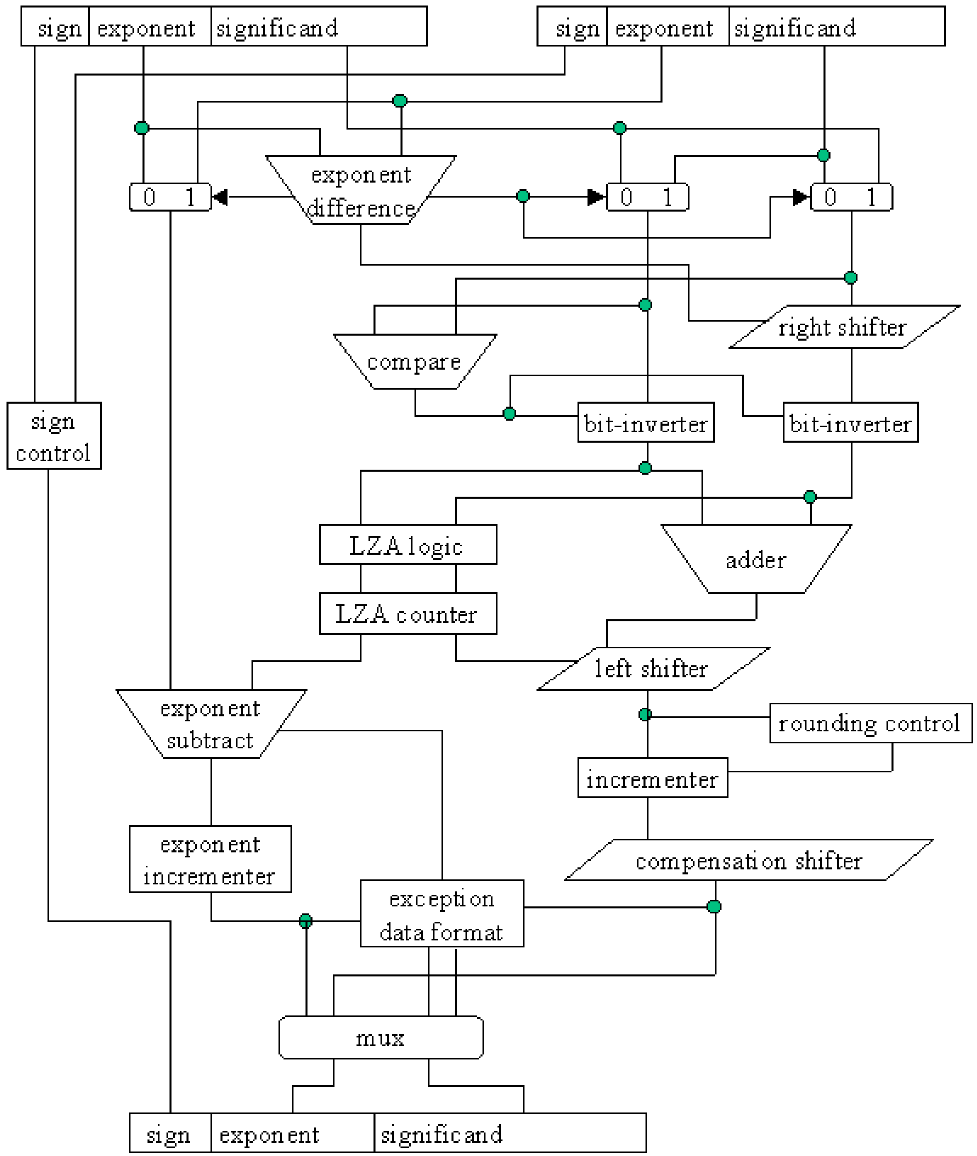
- IEEE default rounding mode -- Round to nearest - even

Significand	Rounded Result	Error	Significand	Rounded Result	Error
X0.00	X0.	0	X1.00	X1.	0
X0.01	X0.	- 1/4	X1.01	X1.	- 1/4
X0.10	X0.	- 1/2	X1.10	X1. + 1	+ 1/2
X0.11	X1.	+ 1/4	X1.11	X1. + 1	+ 1/4

What’s the best architecture?

Architecture Consideration

Floating Point Adder Architecture



Triple Path Floating Point Adder

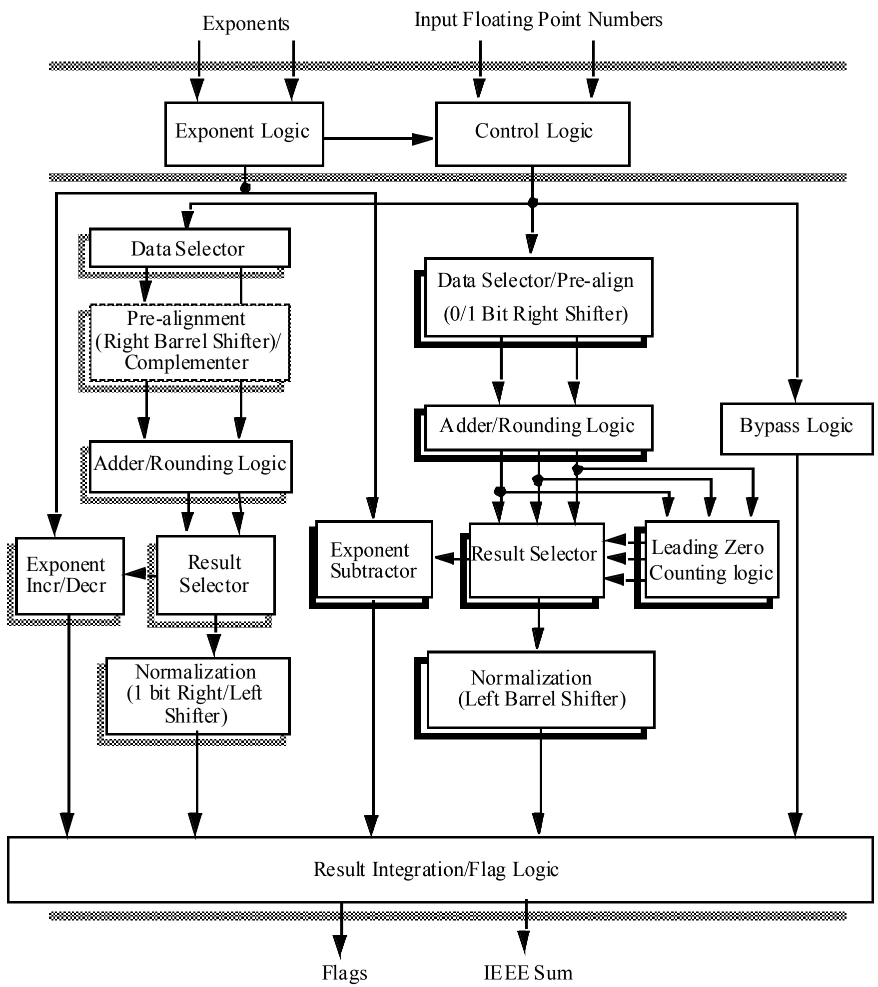
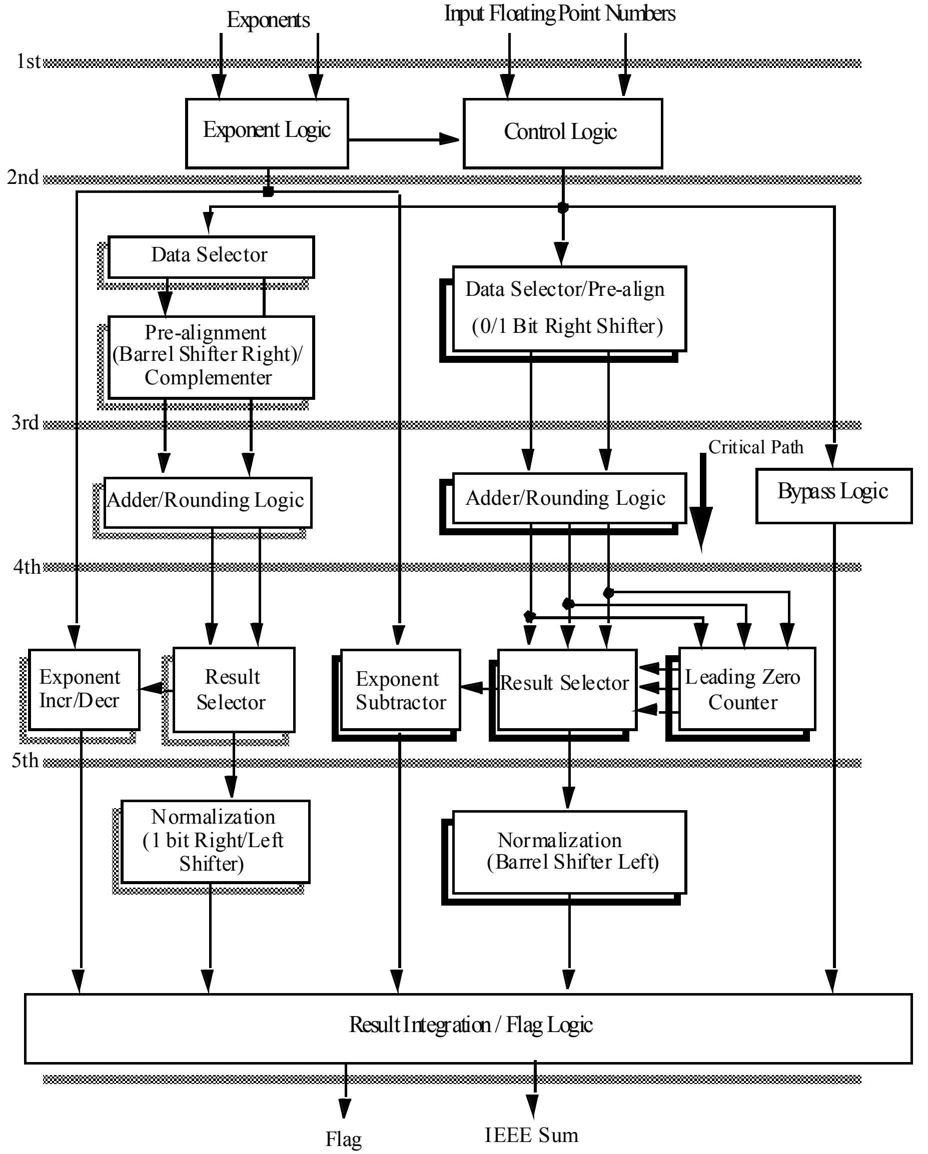
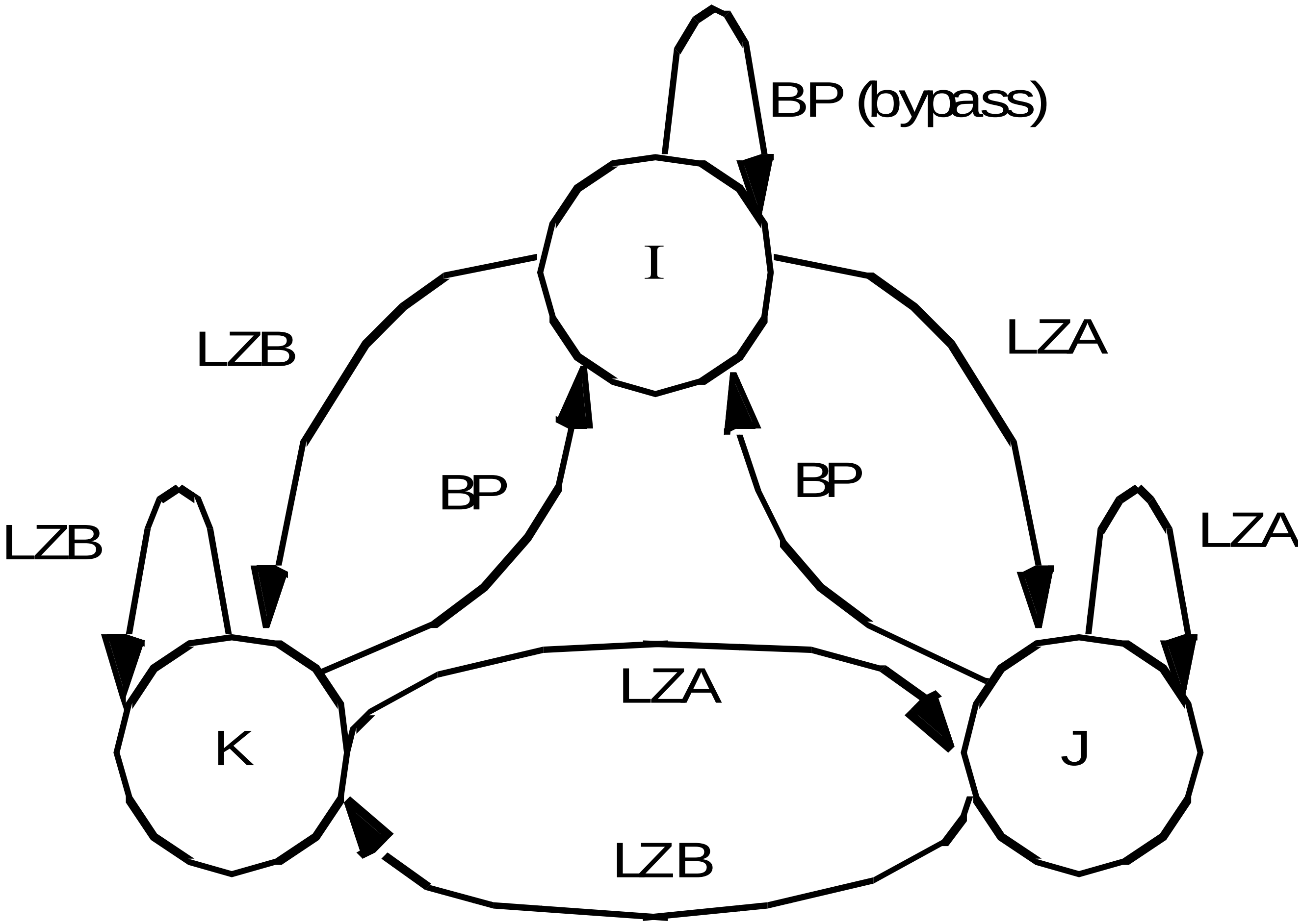


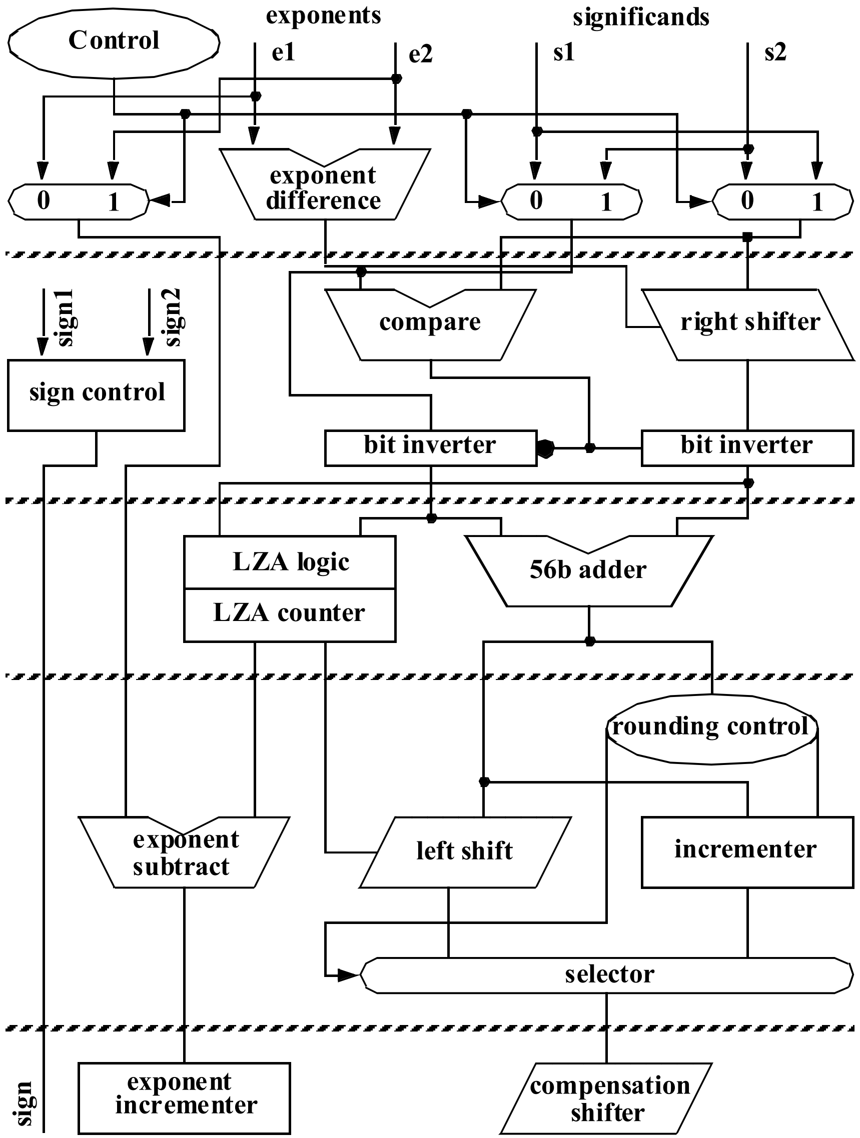
Fig 4.2 - Block diagram of the TDPFADD

Pipelined Triple Paths Floating Point Adder TPFADD

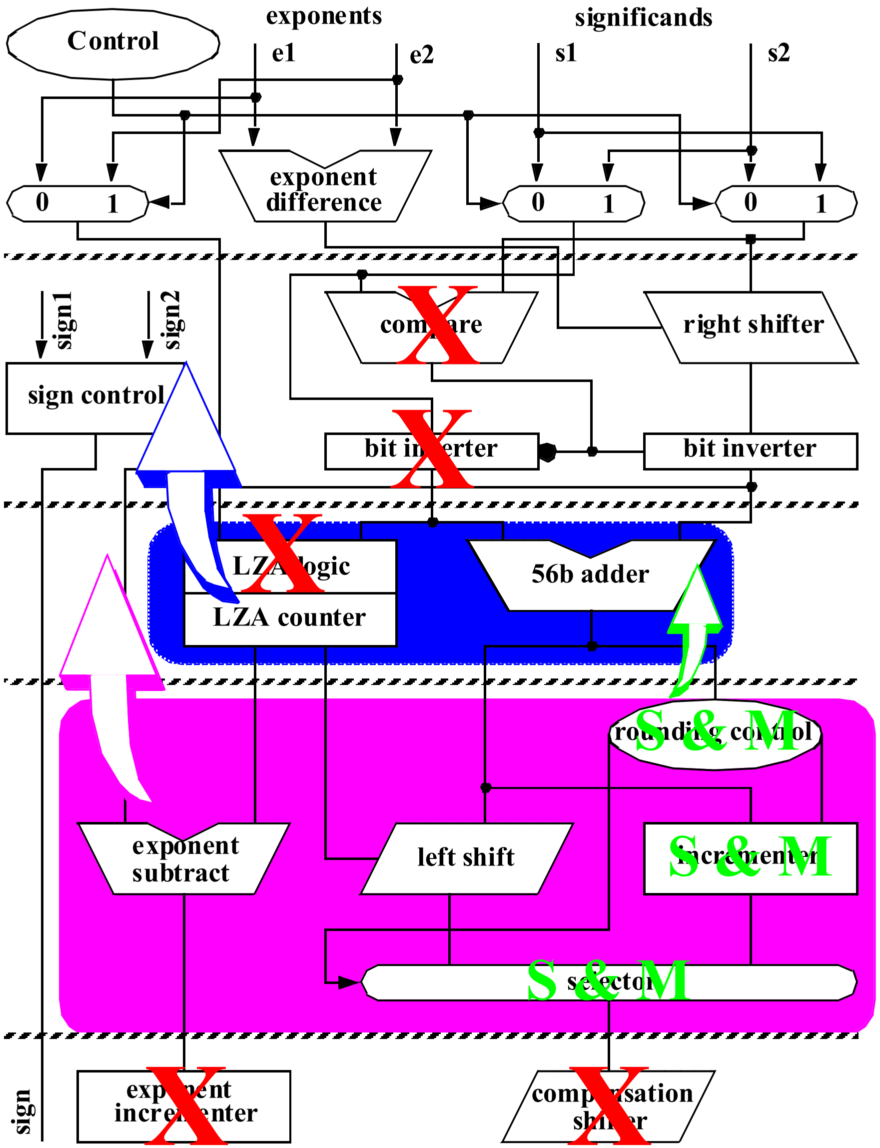



BP (bypass)
I
LZB
LZA
BP
BP
LZB
LZA
LZA
K
J
LZB

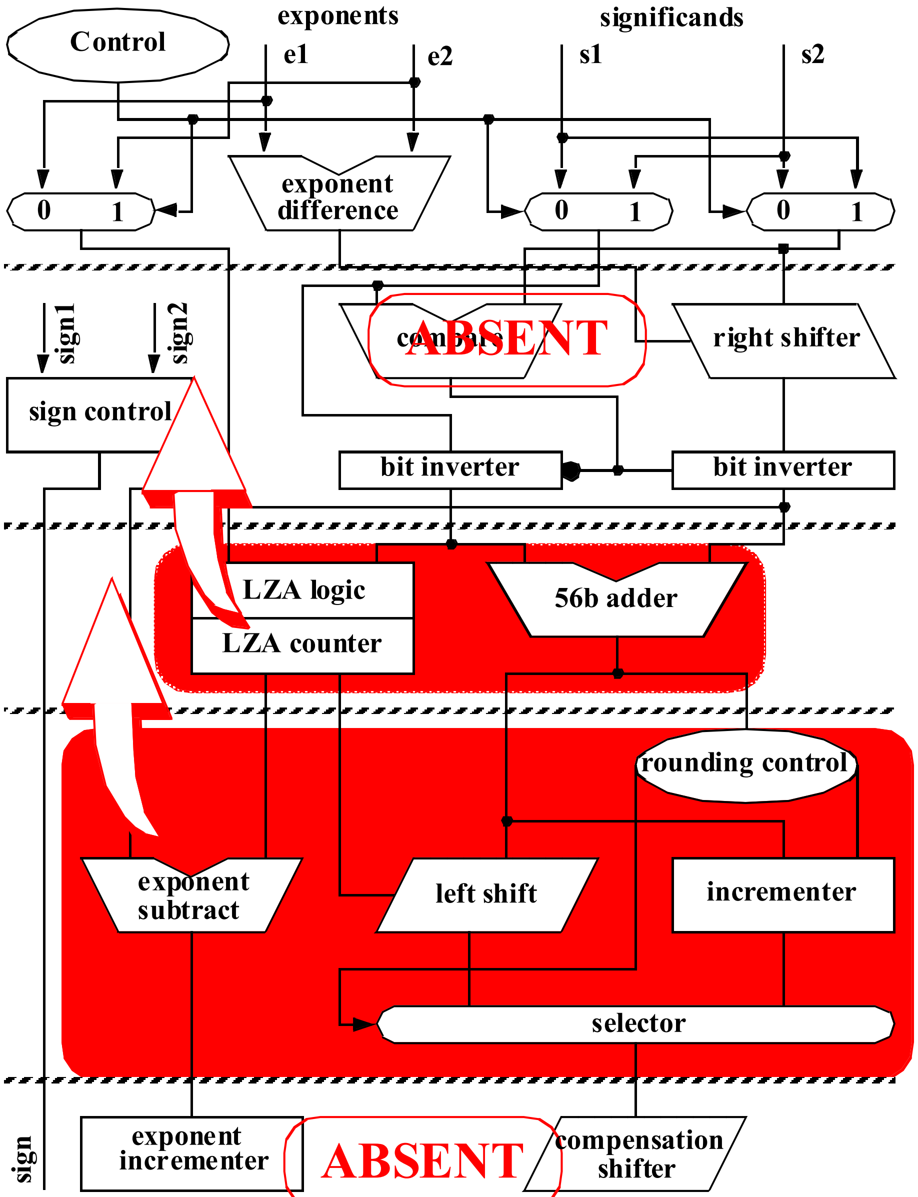
FPADDer with Leading Zero Anticipation Logic



Improvements to previous Designs

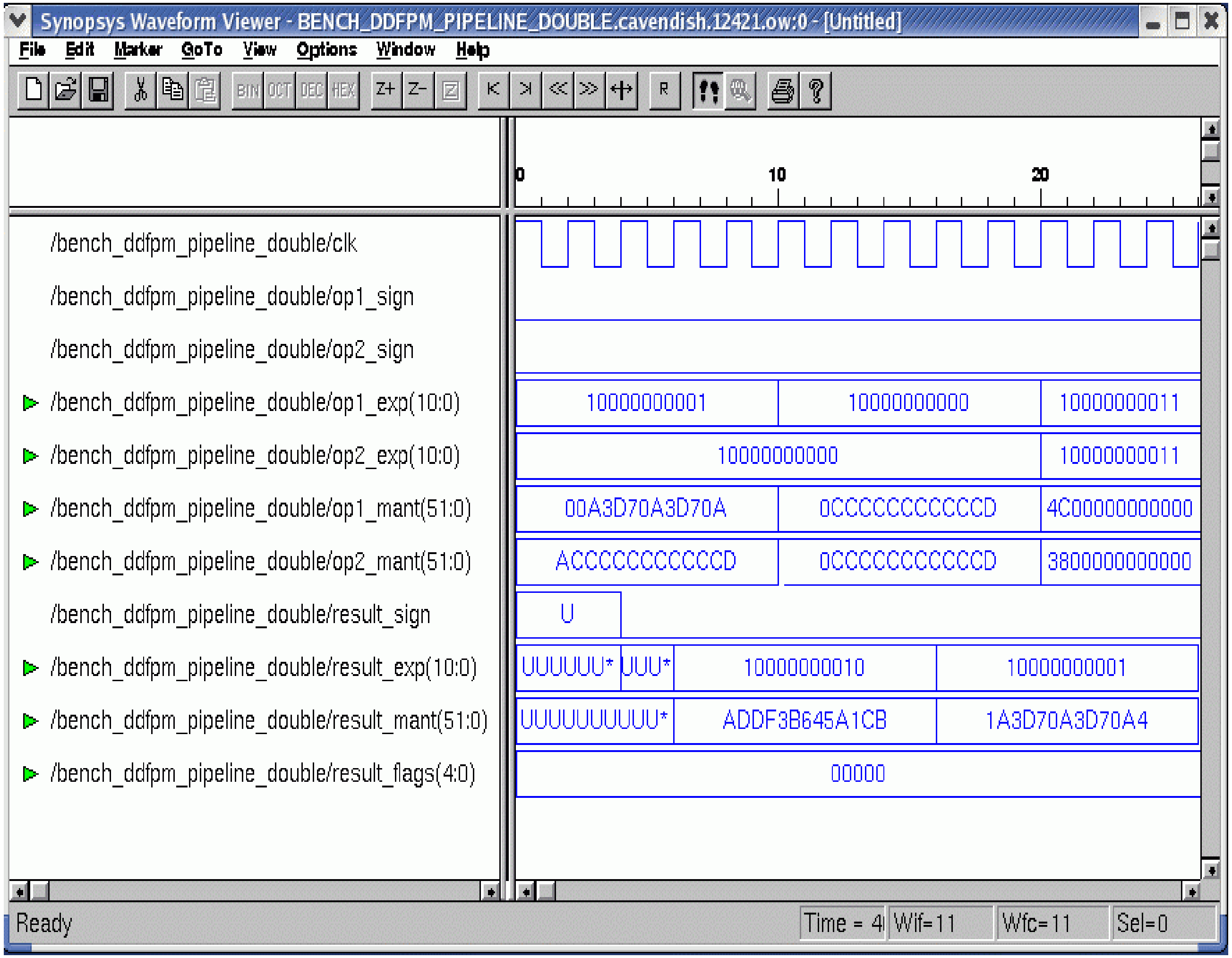


Improvements in FADD from Previous Designs



Comparison of Synthesis results for IEEE 754 Single Precision FP addition Using Xilinx 4052XL-1 FPGA

Parameters	SIMPLE	TDPFADD	PIPE/ TDPFADD
Maximum delay, D (ns)	327.6	213.8	101.11
Average Power, P (mW)@ 2.38 MHz	1836	1024	382.4
Area A, Total number of CLBs (#)	664	1035	1324
Power Delay Product (ns. 10mW)	$7.7. *10^4$	$4.31 *10^4$.	$3.82 *10^4$
Area Delay Product (10 # .ns)	2.18\`$*10^4$	$2.21 * 10^4$	$1.34 *10^4$
Area-Delay^2 Product (10# . ns^2)	$7.13.*10^6$	$4.73 * 10^6$	$1.35 *10^6$



Synopsys Waveform Viewer - BENCH_DDFPM_PIPELINE_DOUBLE.cavendish.12421.ow:0 - [Untitled]
File Edit Marker GoTo View Options Window Help
0
10
20
/bench_ddfpm_pipeline_double/clk
/bench_ddfpm_pipeline_double/op1_sign
/bench_ddfpm_pipeline_double/op2_sign
/bench_ddfpm_pipeline_double/op1_exp(10:0)
10000000001
10000000000
10000000011
/bench_ddfpm_pipeline_double/op2_exp(10:0)
10000000000
10000000011
/bench_ddfpm_pipeline_double/op1_mant(51:0)
00A3D70A3D70A
0CCCCCCCCCCCD
4C00000000000
/bench_ddfpm_pipeline_double/op2_mant(51:0)
ACCCCCCCCCCCD
0CCCCCCCCCCCD
3800000000000
/bench_ddfpm_pipeline_double/result_sign
U
/bench_ddfpm_pipeline_double/result_exp(10:0)
UUUUUU*
UUU*
10000000010
10000000001
/bench_ddfpm_pipeline_double/result_mant(51:0)
UUUUUUUUUU*
ADDF3B645A1CB
1A3D70A3D70A4
/bench_ddfpm_pipeline_double/result_flags(4:0)
00000
Ready
Time = 4
Wif=11
Wfc=11
Sel=0

How can a compound adder compute fastest?

Compound Adder

Compound Adder Cont.

- **Round to nearest** Sum, Sum+1
  - **if g=1**
    - **if (LSB=1) OR (r+s=1)**
      - **Add 1 to the result**
  - **else Truncate at LSB**
- **Round Toward zero** Sum
  - **Truncate**
- **Round Toward +Infinity** Sum, Sum+1 and Sum+2
  - **if sign=positive**
    - **if any bits to the right of the result LSB=1**
      - **Add 1 to the result**
    - **else**
      - **Truncate at LSB**
  - **if sign=negative**
    - **Truncate at LSB**
- **Round Toward -Infinity** Sum, Sum+1 and Sum+2
  - **if sign=negative**
    - **if any bits to the right of the result LSB=1**
      - **Add 1 to the result**
    - **else**
      - **Truncate at LSB**
  - **if sign=positive**
    - **Truncate at LSB**

CLOSE PATH

$$Sel_{+1}^{nearest} = C_{out}(\overline{g} + MSB \cdot L)$$

$$Sel_{+1}^{\infty} = C_{out}(\overline{g} + up \cdot MSB)$$

FAR PATH

$$Sel_{+1}^{nearest} = \begin{cases} C_{out} \cdot g \cdot (L + r + s) + C_{out} \cdot L \cdot [(L-1) + g + r + s] & if\ add = 1 \\ C_{out} \cdot [\overline{g} \cdot \overline{r} \cdot \overline{s} + g \cdot r + MSB \cdot g \cdot (L + s)] & if\ sub = 1 \end{cases}$$

$$Sel_{+1}^{\infty} = \begin{cases} up \cdot \overline{C_{out}} \cdot (g + r + s) & if\ add = 1 \\ C_{out} \cdot [\overline{g} \cdot \overline{r} \cdot \overline{s} + up \cdot (g \cdot (r + s) + MSB)] & if\ sub = 1 \end{cases}$$

$$Sel_{+2}^{\infty} = add \cdot up \cdot Cout \cdot (L + g + r + s)$$

Rounding Block

Compound Adder Cont.

- **Round to nearest** Sum, Sum+1
  - **if g=1**
    - **if (LSB=1) OR (r+s=1)**
      - **Add 1 to the result**
  - **else Truncate at LSB**
- **Round Toward zero** Sum
  - **Truncate**
- **Round Toward +Infinity** Sum, Sum+1
  - **if sign=positive**
    - **if any bits to the right of the result LS**
      - **Add 1 to the result**
    - **else**
      - **Truncate at LSB**
  - **if sign=negative**
    - **Truncate at LSB**
- **Round Toward -Infinity** Sum, Sum+1 and Sum+2
  - **if sign=negative**
    - **if any bits to the right of the result LSB=1**
      - **Add 1 to the result**
    - **else**
      - **Truncate at LSB**
  - **if sign=positive**
    - **Truncate at LSB**

CLOSE PATH

$$Sel_{+1}^{nearest} = C_{out}(\overline{g} + MSB \cdot L)$$

$$Sel_{+1}^{p} = C_{out}(\overline{g} + up \cdot MSB)$$

FAR PATH

$$Sel_{+1}^{nearest} = \begin{cases} C_{out} \cdot g \cdot (L + r + s) + C_{out} \cdot L \cdot [(L-1) + g + r + s)] & if\ add = 1 \\ C_{out} \cdot [\overline{g} \cdot \overline{r} \cdot \overline{s} + g \cdot r + MSB \cdot g \cdot (L + s)] & if\ sub = 1 \end{cases}$$

$$Sel_{+1}^{p} = \begin{cases} up \cdot \overline{C_{out}} \cdot (g + r + s) & if\ add = 1 \\ C_{out} \cdot [\overline{g} \cdot \overline{r} \cdot \overline{s} + up \cdot (g \cdot (r + s) + MSB)] & if\ sub = 1 \end{cases}$$

$$Sel_{+2}^{p} = add \cdot up \cdot Cout \cdot (L + g + r + s)$$

Rounding Block

Reference List

[1] ***Computer Arithmetic Systems, Algorithms, Architecture and Implementations.*** A. Omondi. Prentice Hall, 1994.

[2] ***Computer Architecture A Quantitative Approach,*** chapter Appendix A. D. Goldberg. Morgan Kaufmann, 1990.

[3] ***Reduced latency IEEE floating-point standard adder architectures.*** Beaumont-Smith, A.; Burgess, N.; Lefrere, S.; Lim, C.C.; Computer Arithmetic, 1999. Proceedings. 14th IEEE Symposium on , 14-16 April 1999

[4] ***Rounding in Floating-Point Addition using a Compound Adder.*** J.D. Bruguera and T. Lang. Technical Report. University of Santiago de Compostela. (2000)

[5] ***Floating point adder/subtractor performing ieee rounding and addition/subtraction in parallel.*** W.-C. Park, S.-W. Lee, O.-Y. Kown, T.-D. Han, and S.-D. Kim. IEICE Transactions on Information and Systems, E79-D(4):297–305, Apr. 1996.

[6] ***Efficient simultaneous rounding method removing sticky-bit from critical path for floating point addition.*** Woo-Chan Park; Tack-Don Han; Shin-Dug Kim; ASICs, 2000. AP-ASIC 2000. Proceedings of the Second IEEE Asia Pacific Conference on , 28-30 Aug. 2000 Pages:223 – 226

[7] ***Efficient implementation of rounding units.*** Burgess. N.; Knowles, S.; Signals, Systems, and Computers, 1999. Conference Record of the Thirty-Third Asilomar Conference on, Volume: 2, 24-27 Oct. 1999 Pages: 1489 - 1493 vol.2

[8] ***The Flagged Prefix Adder and its Applications in Integer Arithmetic.*** Neil Burgess. Journal of VLSI Signal Processing 31, 263–271, 2002

[9] ***A family of adders.*** Knowles, S.; Computer Arithmetic, 2001. Proceedings. 15th IEEE Symposium on , 11-13 June 2001 Pages:277 – 281

[10] ***PAPA - packed arithmetic on a prefix adder for multimedia applications.*** Burgess, N.; Application-Specific Systems, Architectures and Processors, 2002. Proceedings. The IEEE International Conference on, 17-19 July 2002 Pages:197 – 207

[11] ***Nonheuristic optimization and synthesis of parallel prefix adders.*** R. Zimmermann, in Proc. Int.Workshop on Logic and Architecture Synthesis, Grenoble, France, Dec. 1996, pp. 123–132.

[12] ***Leading-One Prediction with Concurrent Position Correction.*** J.D. Bruguera and T. Lang. IEEE Transactions on Computers. Vol. 48. No. 10. pp. 1083-1097. (1999)

[13] ***Leading-zero anticipatory logic for high-speed floating point addition.*** Suzuki, H.; Morinaka, H.; Makino, H.; Nakase, Y.; Mashiko, K.; Sumi, T.; Solid-State Circuits, IEEE Journal of , Volume: 31 , Issue: 8 , Aug. 1996 Pages:1157 – 1164

[14] ***On low power floating point data path architectures.*** R. V. K. Pillai. Ph. D thesis, Concordia University, Oct. 1999.

[15] ***A low power approach to floating point adder design.*** Pillai, R.V.K.; Al-Khalili, D.; Al-Khalili, A.J.; Computer Design: VLSI in Computers and Processors, 1997. ICCD '97. Proceedings. 1997 IEEE International Conference on, 12-15 Oct. 1997 Pages:178 – 185

[16] ***Design of Floating-Point Arithmetic Units.*** S.F.Oberman, H. Al-Twaijry and M.J.Flynn. Proc. Of the 13th IEEE Symp on Computer Arithmetic. pp. 156-165 1997

[17] ***Digital Arithmetic.*** M.D. Ercegovac and T. Lang. San Francisco: Morgan Daufmann, 2004. ISBN 1-55860-798-6

[18] ***Computer Arithmetic Algorithms.*** Israel Koren. Pub A K Peters, 2002. ISBN 1-56881-160-8

[19] ***Parallel Prefix Adder Designs.*** Beaumont-Smith, A.; Lim, C.-C.; Computer Arithmetic, 2001. Proceedings. 15th IEEE Symposium on, 11-13 June 2001 Pages:218 – 225

[20] ***Low-Power Logic Styles: CMOS Versus Pass-Transistor Logic.*** Reto Zimmmemann and Wolfgang Fichtner, IEEE Journal of Solid-State Circuits, VOL.,32, No.7, July 1997

[21] ***Comparative Delay, Noise and Energy of High-performance Domino Adders with SNP.*** Yibin Ye, etc., 2000 Symposium on VLSI Circuits Digest of Technical Papers

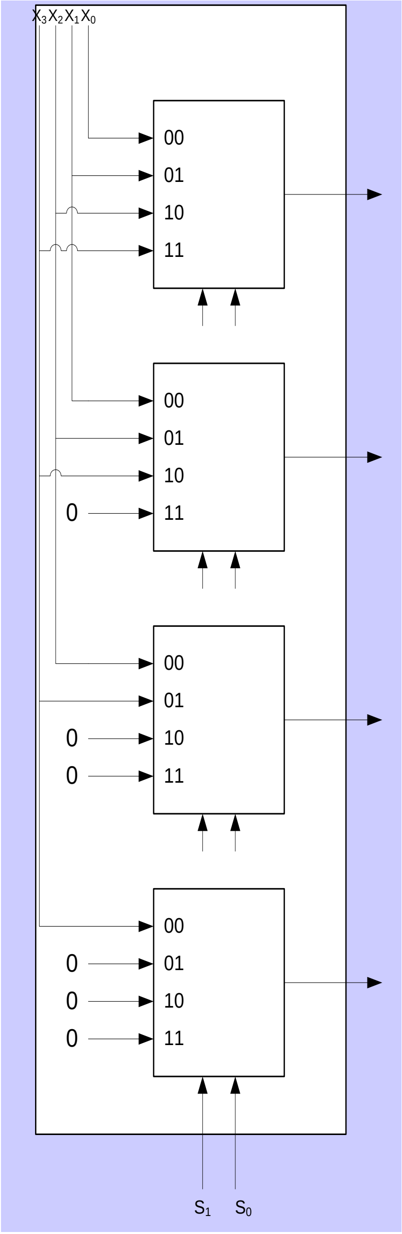
[22] ***5 GHz 32b Integer-Execution Core in 130nm Dual-Vt CMOS.*** Sriram Vangal, etc., IEEE Journal of Solid-State Circuits, VOL.37, NO.11, November 2002

[23] ***Performance analysis of low-power 1-bit CMOS full adder cells.*** A.Shams, T.Darwish and M.Byoumi, IEEE Trans. on VLSI Syst., vol. 10, no.1, pp. 20-29, Feb 2002.

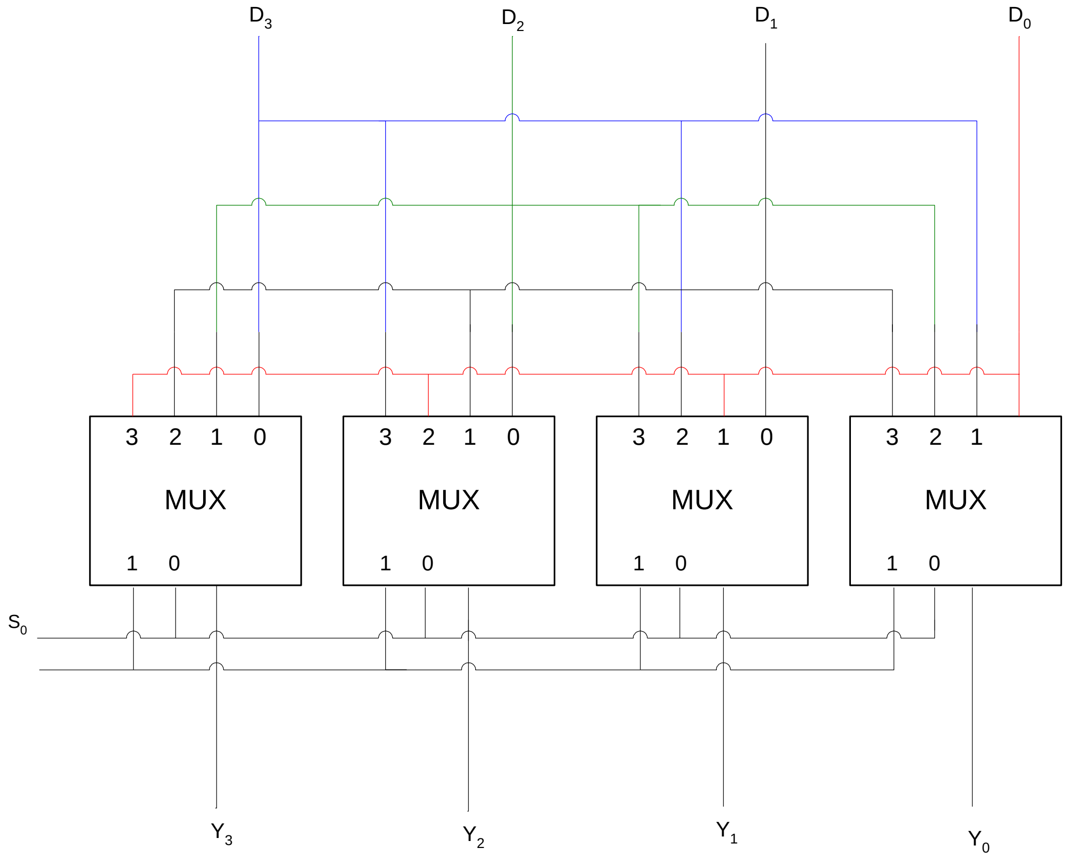
What about shifting?
How to shift several
bits at once ?

Barrel Shifters

Right Shift
Barrel Shifter

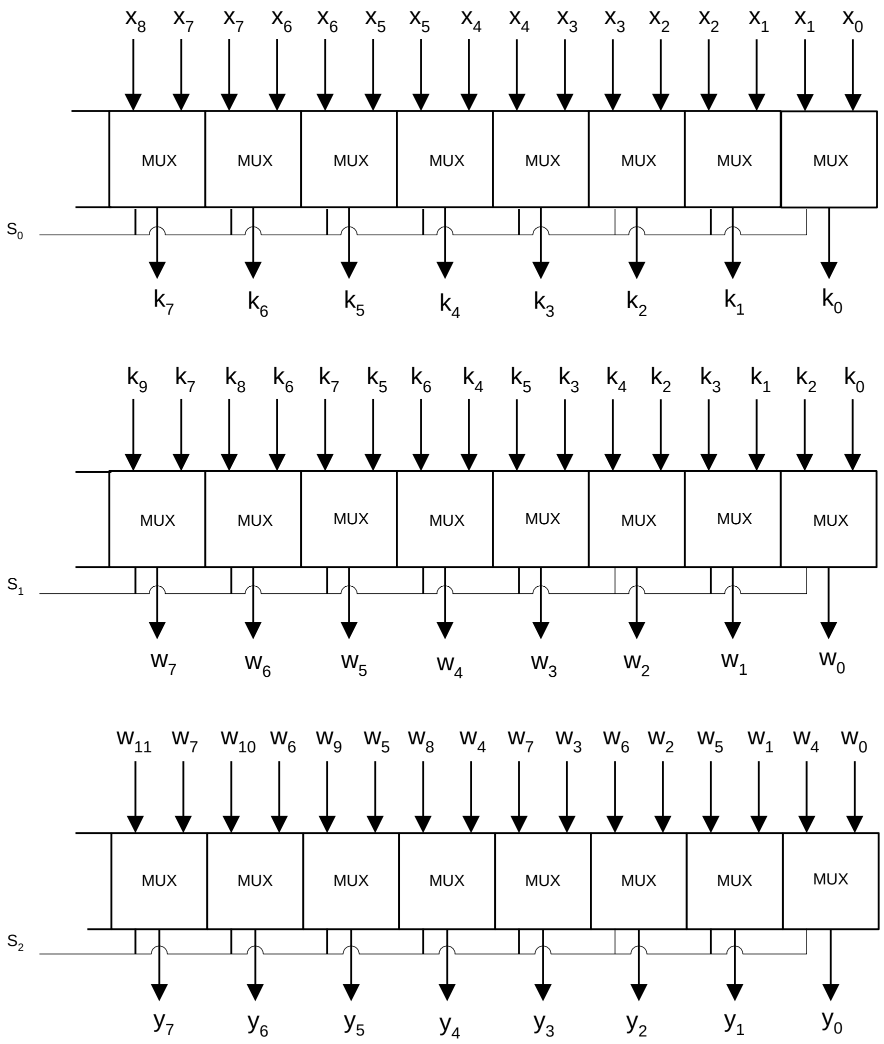


Shift and Rotate Barrel Shifter



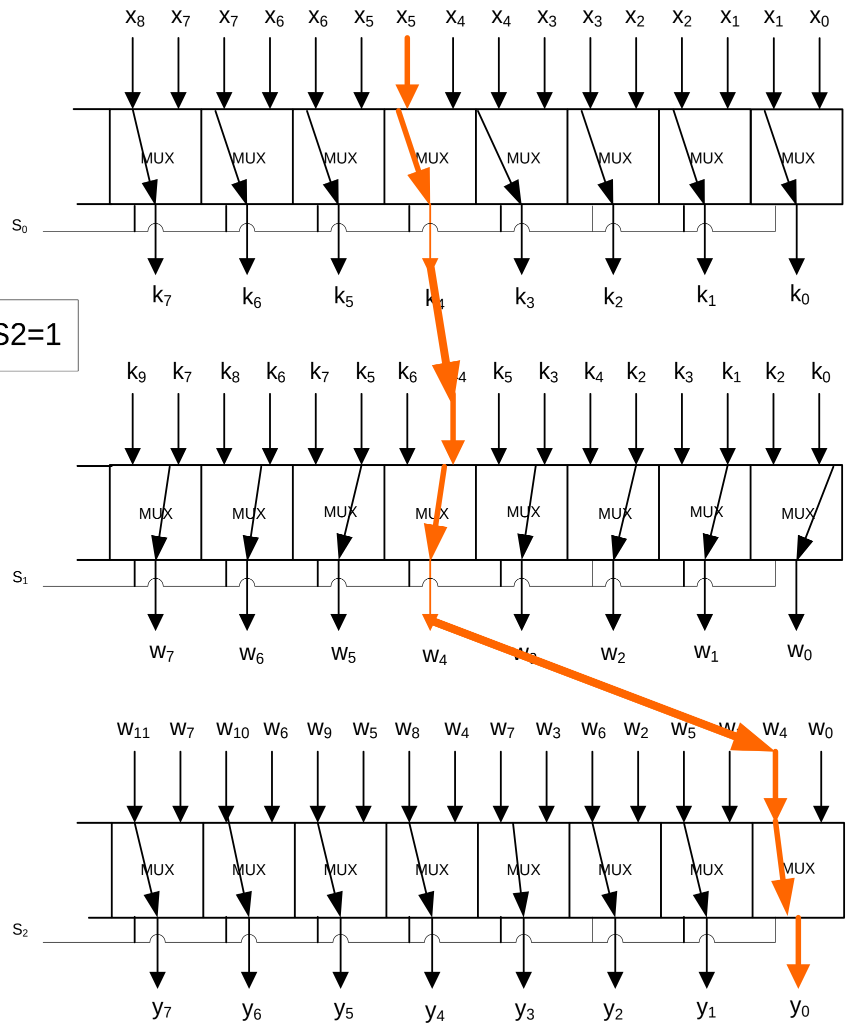
Select		Out Put				Operation
S_i	S_o	Y_3	Y_2	Y_1	Y_0	
0	0	D_3	D_2	D_1	D_0	No Shift
0	1	D_2	D_1	D_0	D_3	Rotate Once
1	0	D_1	D_0	D_3	D_2	Rotate Twice
1	1	D_0	D_3	D_2	D_1	Rotate 3 times

Distributed Barrel Shifter



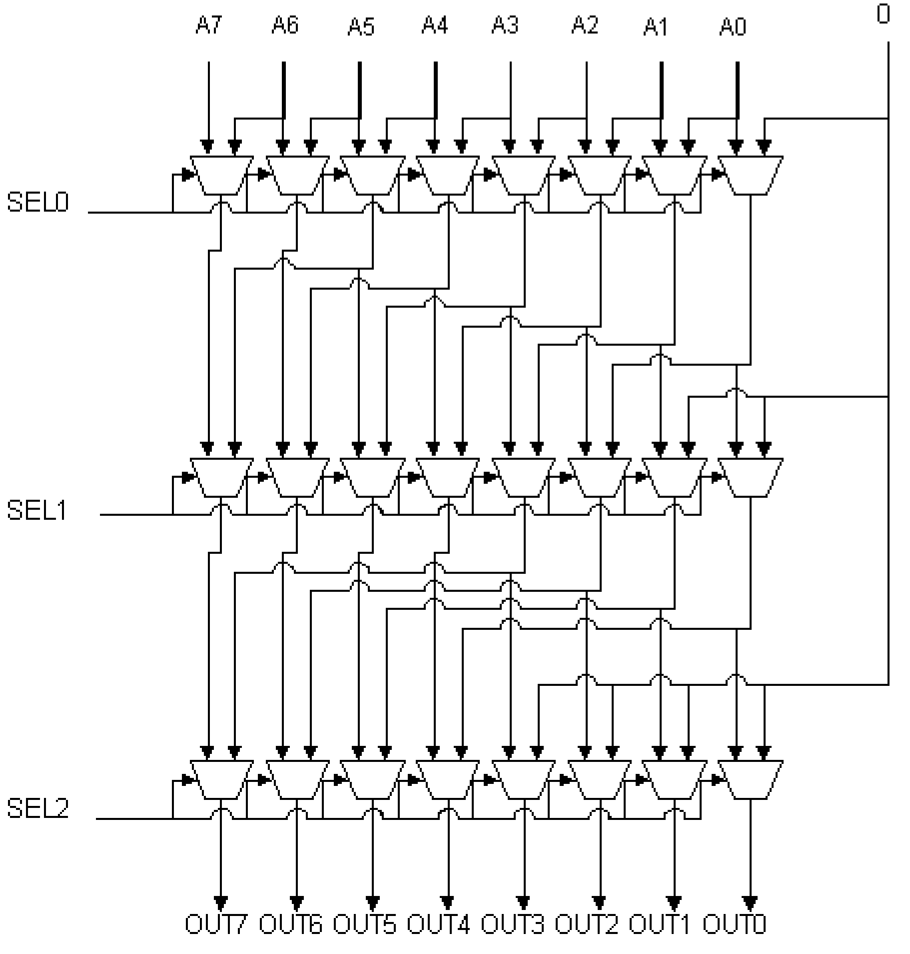
Paths of the distributed Barrel Shifter

S0=1 S1=0 S2=1

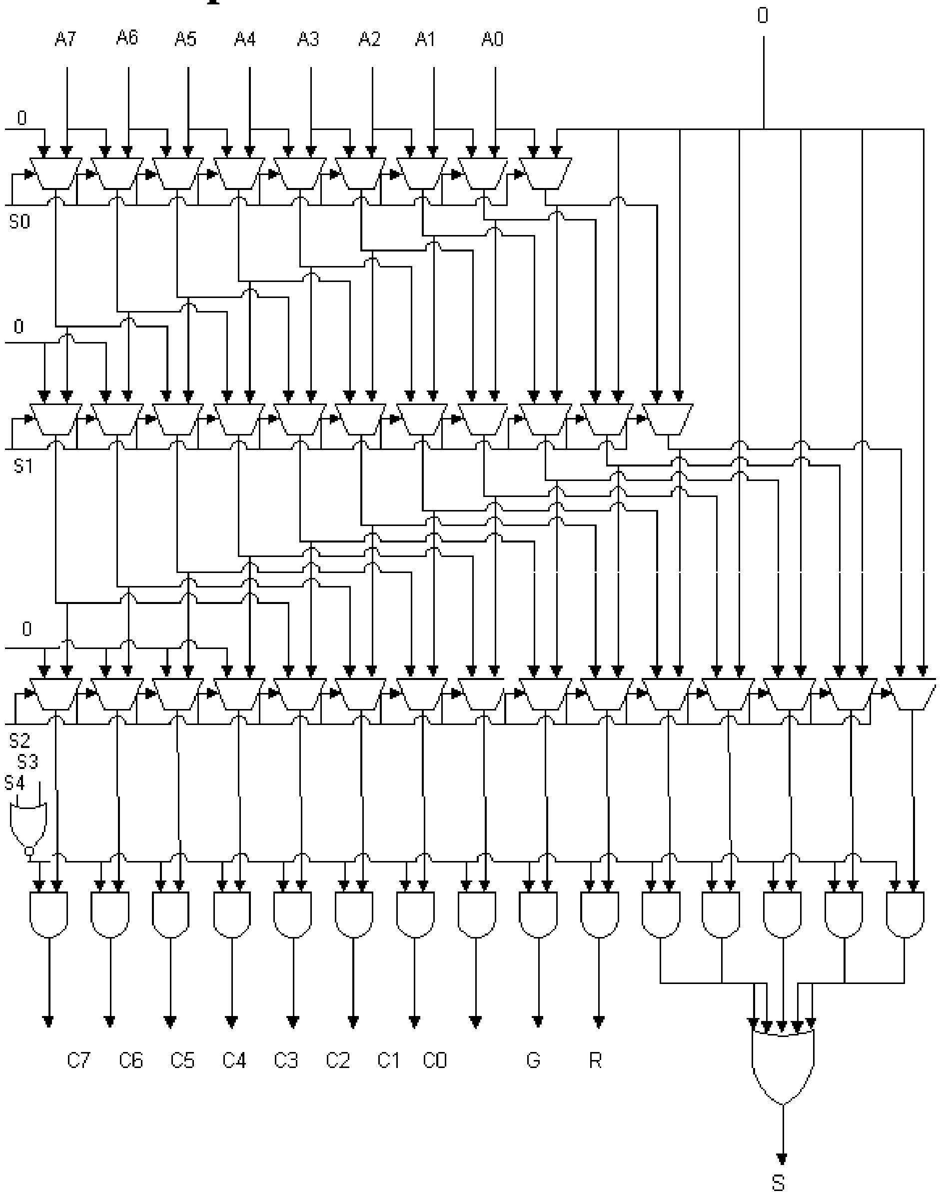


Please note that in this case if we have 8 bits of data then inputs to MUXes greater than 7 should be be set to a desired value

A Normalization Shifter for FP Arithmetic



. **Block Diagram of the Right Shifter & GRS-bit Generation Component**



The end

Thank you for your attendance