Presentation Slides

Chapter 9

Multi-value Numeric Design

Logically Determined Design:
Clockless System Design With NULL Convention Logic

by Karl Fant

John Wiley & Sons, Inc.

Introduce quaternary logic

Does Multi-value Representation Provide Advantages For Expressing Numeric Processes?

With the familiar electronic representation a digit is a path and value is represented as discriminable voltage levels on that path; values are clearly much more expensive than digits.

With multi-path representation enabling multi-value variables, values and digits are both represented with paths. The cost to represent a digit and the cost to represent its values are identical and there is no bias.

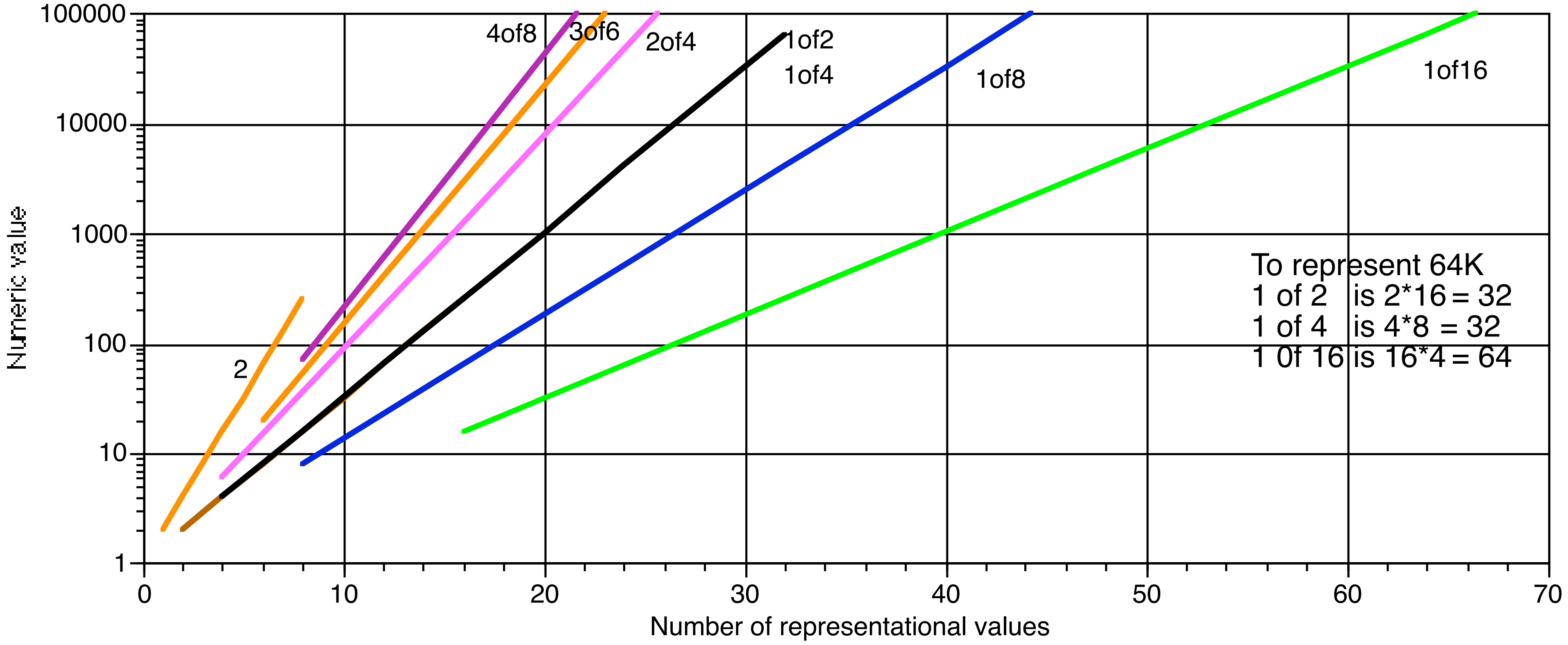
Multi-path representation allows the consideration of radices and encodings other than binary for numeric representation.

Various encodings are considered in terms of cost of transmission and cost of functional combination.

Cost of Representation

	Binary	Multi-rail
Cost of place	voltage threshold	wire
Cost of value	wire	wire

The cost of representing a given number with M of N encoding is the number of values per digit times the number of digits to represent the number.



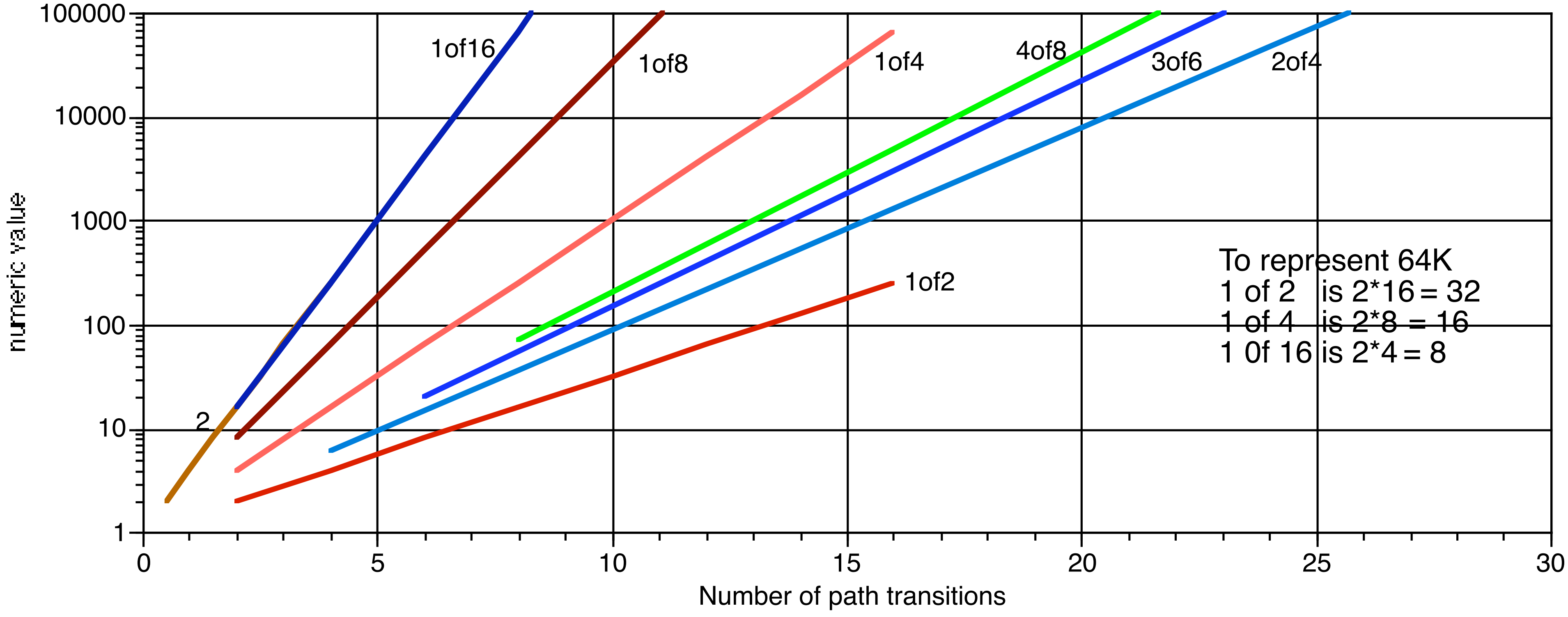
The N/2 of N encodings show significant advantage.

The 1 of 2 and 1 of 4 encodings are the optimal 1 of N encodings.

Cost of Flow

	binary	N/2ofN	1ofN
Cost of flow per digit	0.5	N	2

The cost of flow is how many path switches per digit times the number of digits to move a number.



The multi-path encoded digits switch twice per digit because of the NULL transition between each data wavefront. Ignoring the cost of the clock signal and explicit registers, the binary representation switches 0.5 times per digit because of the average switching behavior of binary representation.

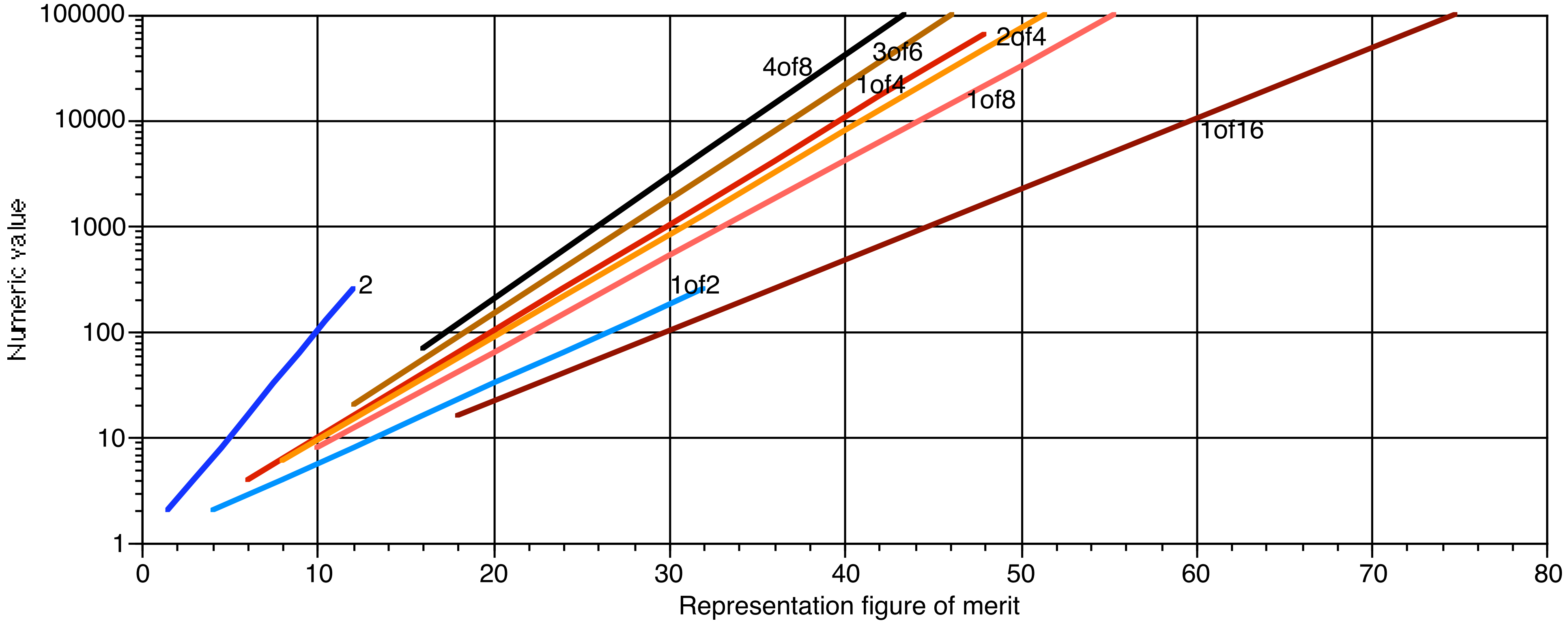
.

The important observation is that 1of4 is better than any of the N/2ofN encodings

Combined Cost of Representation

Adding the two costs to get a figure of merit. One can see that 1of4 is the optimal 1 of N encoding. It is superior to 1of2. Only 4of8 and 3of6 appear to be better.

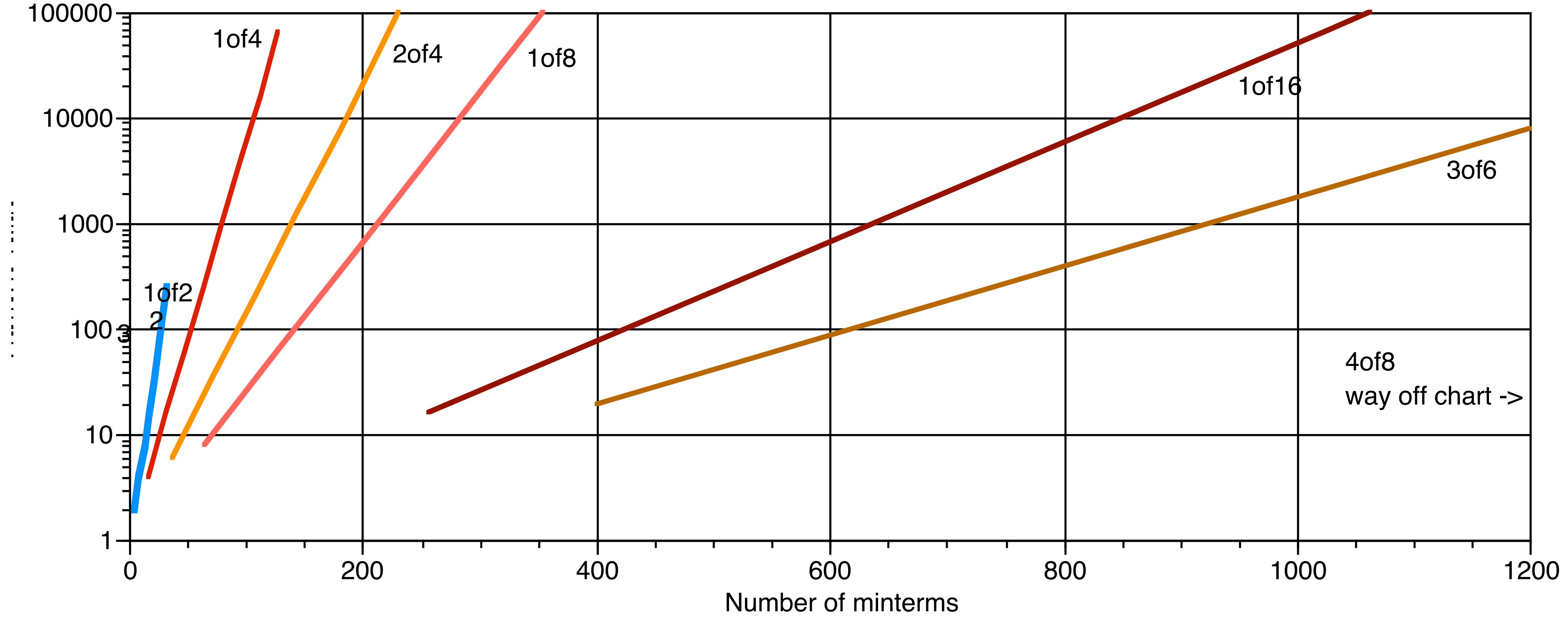
1of4 and 1of2 have the identical representation cost in number of paths but 1of4 has half the switching cost.



Cost of Expressing Combination

	binary	1of2	1of4	2of4	1of8
The cost of combining digits	4	4	16	36	64

The cost of combining numbers represented in the various encodings with arithmetic operations such as addition is measured by taking the number of minterms defining a function combining two digits.



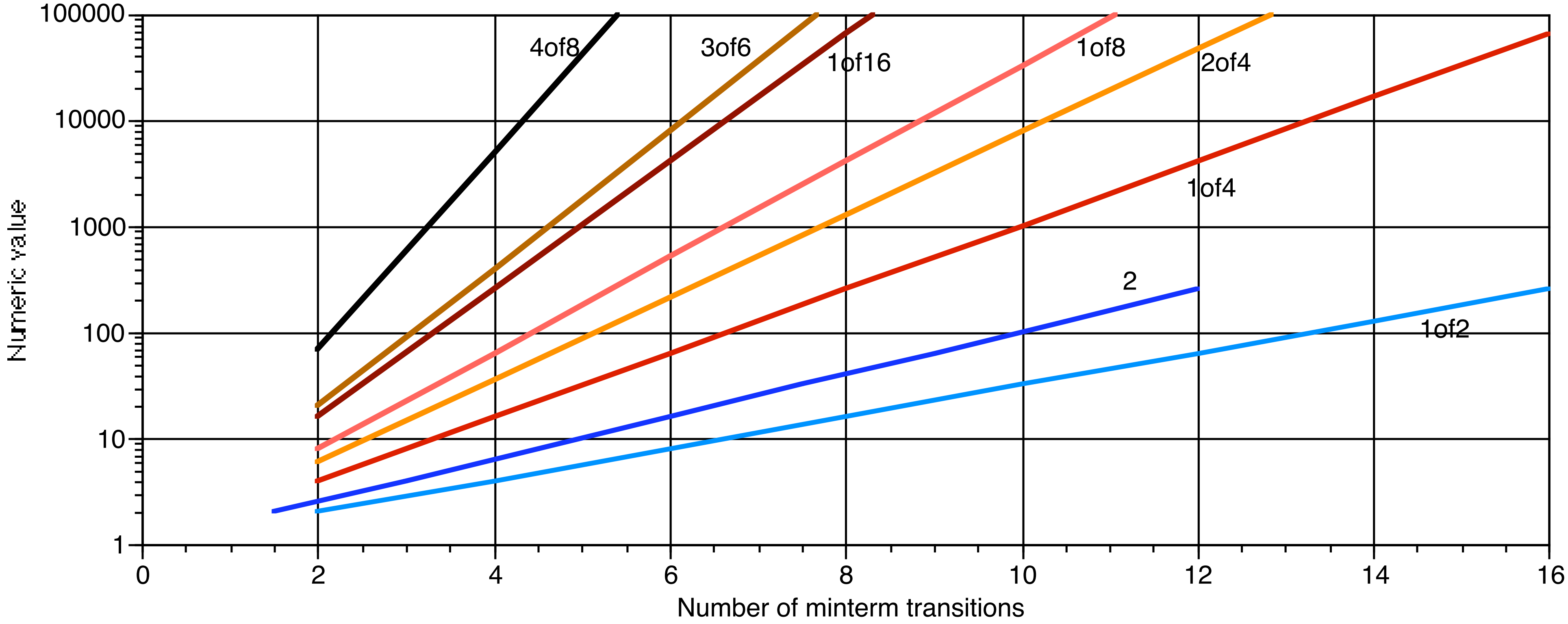
Again, 1of4 stands out as an optimal representation only bested by 1of2.

To represent 64K
1 of 2 is 2*16 = 32
1 of 4 is 2*8 = 16
1 0f 16 is 2*4 = 8

Cost of Resolving Combination

	Binary	N/2ofN	1ofN
The cost of resolving digit combination	1.5	2	2

The cost of resolving digit combination is the number of minterm transitions to resolve a combination of two digits. For all the multi-path encodings exactly one minterm per digit combination will transition to DATA then to NULL for two transitions.

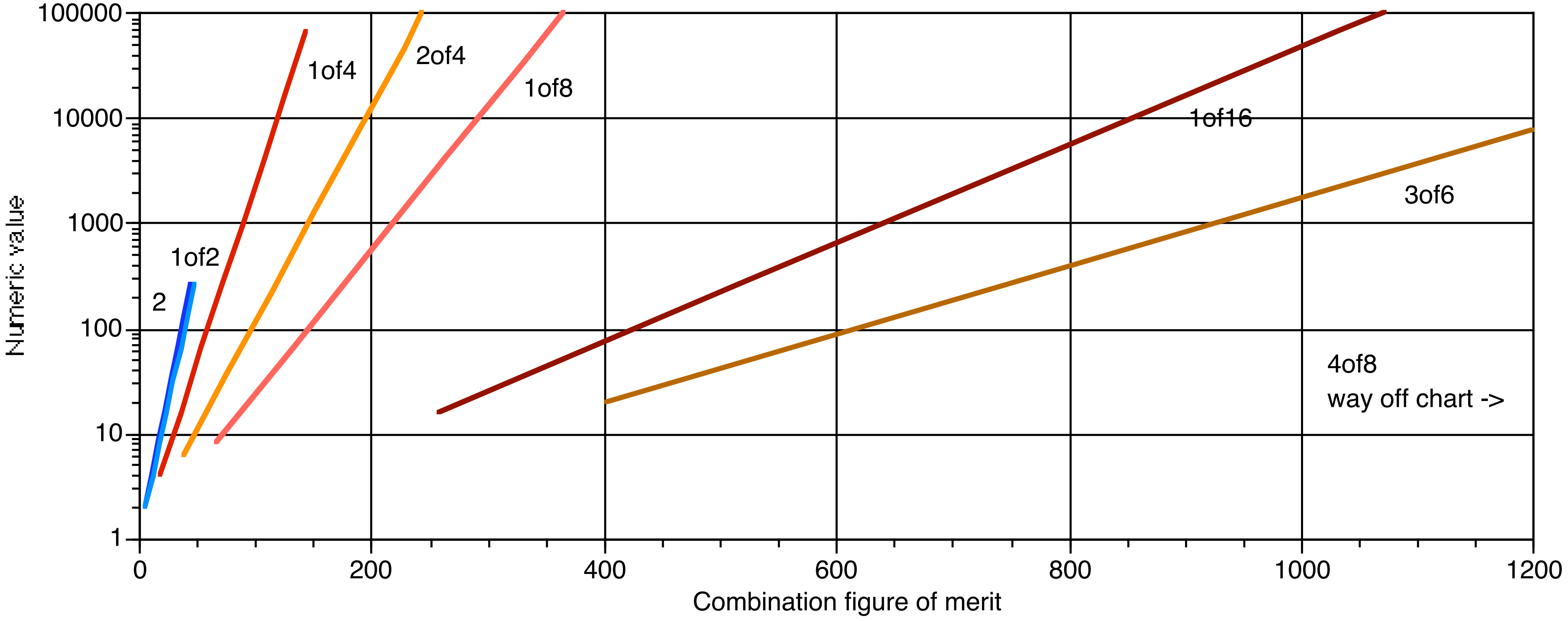


For Boolean representation there are 4 minterms per digit. The probability that the same minterm will be used consecutively and not cause 2 switches is 0.25 leaving an average of 1.5 transitions per digit combination.

The chart does not take into account the fan-in and fan-out complexity of the minterm which is much greater for N/2 of N encodings than for the 1 of N encodings.

Combined Cost of Combination

When the resource and energy costs of digit combination are combined
it shows that 1 of 2 and 1 of 4 are the most efficient encodings.



Summary of Multi-path Numeric Representation

The 1of4 encoding is clearly superior to 1of2 encoding for transmission. The path resources for 1of2 and 1of4 are identical and 1of4 requires half the switching energy.

The N/2ofN encodings provide slightly better performance for transmission but their combinational performance is abysmal.

The 1ofN>4 codes fall below 1of2 and 1of4 in both categories so that leaves only 1of2 and 1of4 to consider.

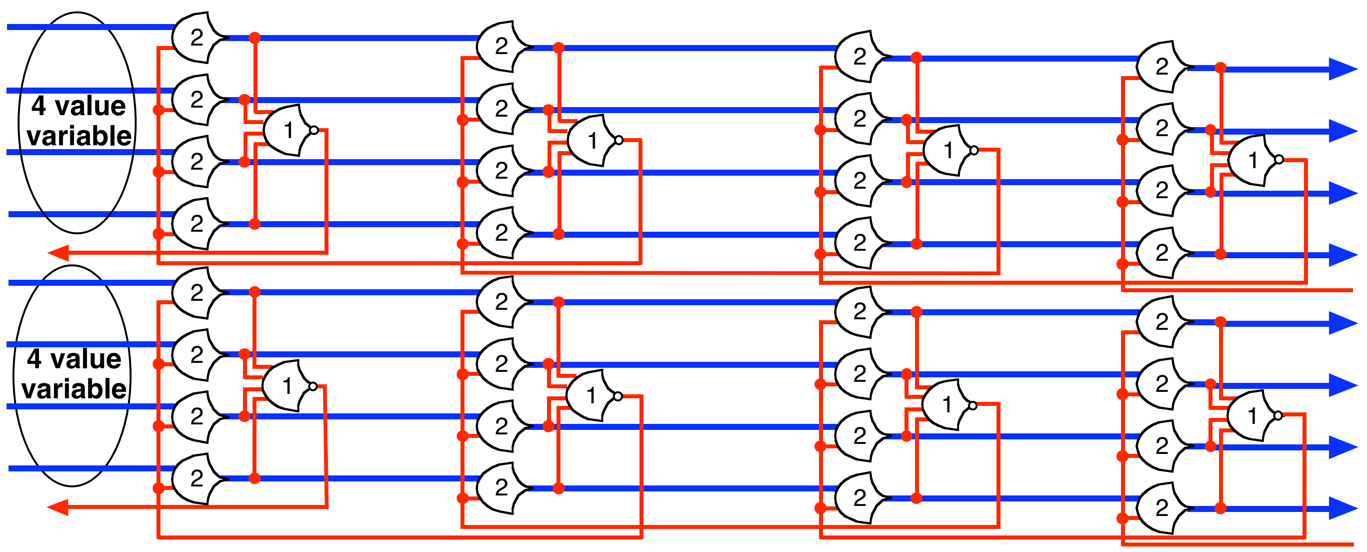
While combining 1of4 variables is more costly in resources, it has a considerable advantage in terms of energy and speed (fewer stages of logic, fewer digits asserting a value, shorter addition carry chain).

The 1of4 (quaternary) encoding seems to be a viable option for numeric representation with multi-path representation.

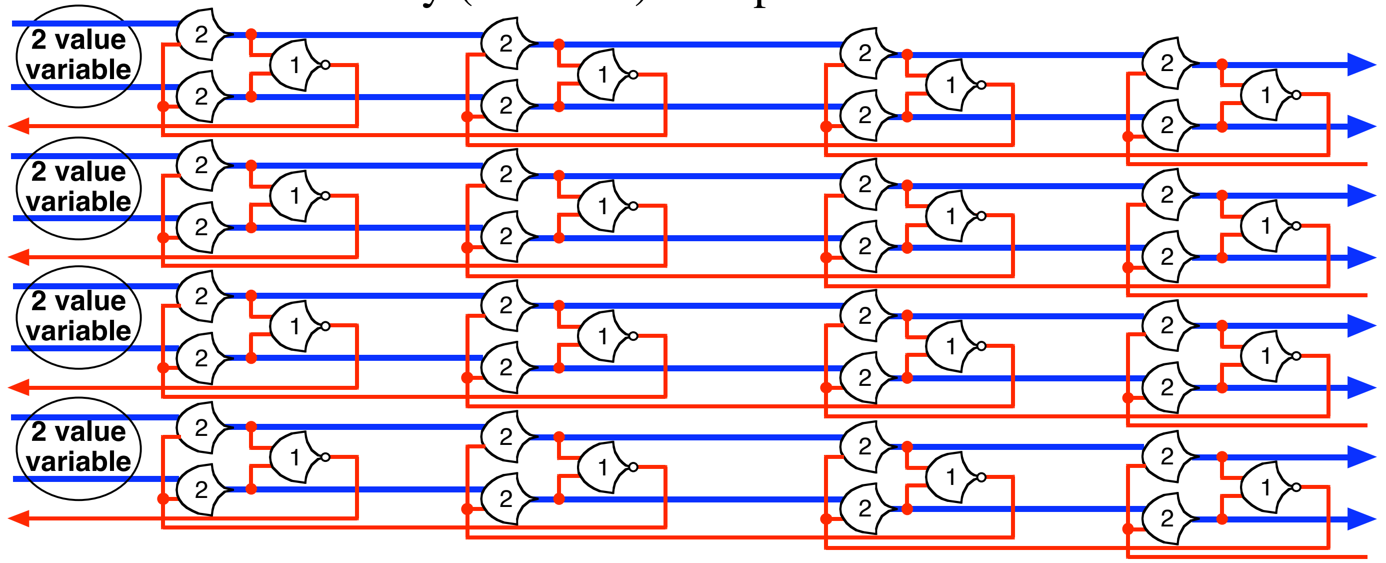
The Quaternary Data Path

The quaternary data path requires half the switching than does a binary data path with slightly less hardware.

Quaternary (four-rail) data path



Binary (dual-rail) data path



A quaternary ALU and a binary ALU are compared.

The binary ALU is considered both

with 1 of 4 data path representation converted to binary and

with 1 of 2 data path representation.

The Quaternary Logic Operations are Derived in Terms of Tandem Binary Logic Operations.

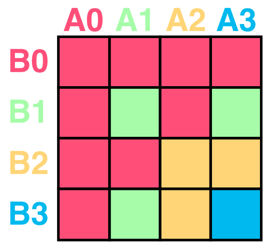
AND

2 bit binary encoding

		A			
		00	01	10	11
B	00	00	00	00	00
	01	00	01	00	01
	10	00	00	10	10
	11	00	01	10	11

Four rail encoding

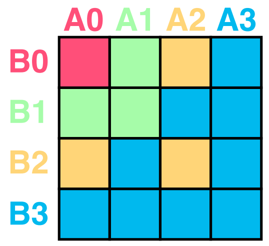
		A			
		0	1	2	3
B	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3



OR

		A			
		00	01	10	11
B	00	00	01	10	11
	01	01	01	11	11
	10	10	11	10	11
	11	11	11	11	11

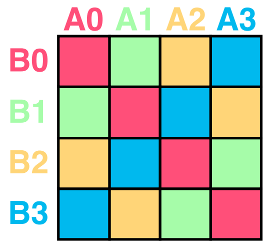
		A			
		0	1	2	3
B	0	0	1	2	3
	1	1	1	3	3
	2	2	3	2	3
	3	3	3	3	3



XOR

		A			
		00	01	10	11
B	00	00	01	10	11
	01	01	00	11	10
	10	10	11	00	01
	11	11	10	01	00

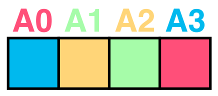
		A			
		0	1	2	3
B	0	0	1	2	3
	1	1	0	3	2
	2	2	3	0	1
	3	3	2	1	0



NOT

A			
00	01	10	11
11	10	01	00

A			
0	1	2	3
3	2	1	0



The Quaternary Arithmetic Operations are Simply Radix Four Arithmetic Operations.

CI = 0

quaternary SUM

B \ A	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

SUM

	A0	A1	A2	A3
B0	🟥	🟩	🟨	🟦
B1	🟩	🟨	🟦	🟥
B2	🟨	🟦	🟥	🟩
B3	🟦	🟥	🟩	🟨

quaternary CARRY

B \ A	0	1	2	3
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	1	1	1

CARRY

	A0	A1	A2	A3
B0	🟥	🟥	🟥	🟥
B1	🟥	🟥	🟥	🟩
B2	🟥	🟥	🟩	🟩
B3	🟥	🟩	🟩	🟩

CI = 1

quaternary SUM

B \ A	0	1	2	3
0	1	2	3	0
1	2	3	0	1
2	3	0	1	2
3	0	1	2	3

SUM

	A0	A1	A2	A3
B0	🟩	🟥	🟦	🟥
B1	🟥	🟦	🟥	🟩
B2	🟦	🟥	🟩	🟨
B3	🟥	🟩	🟨	🟦

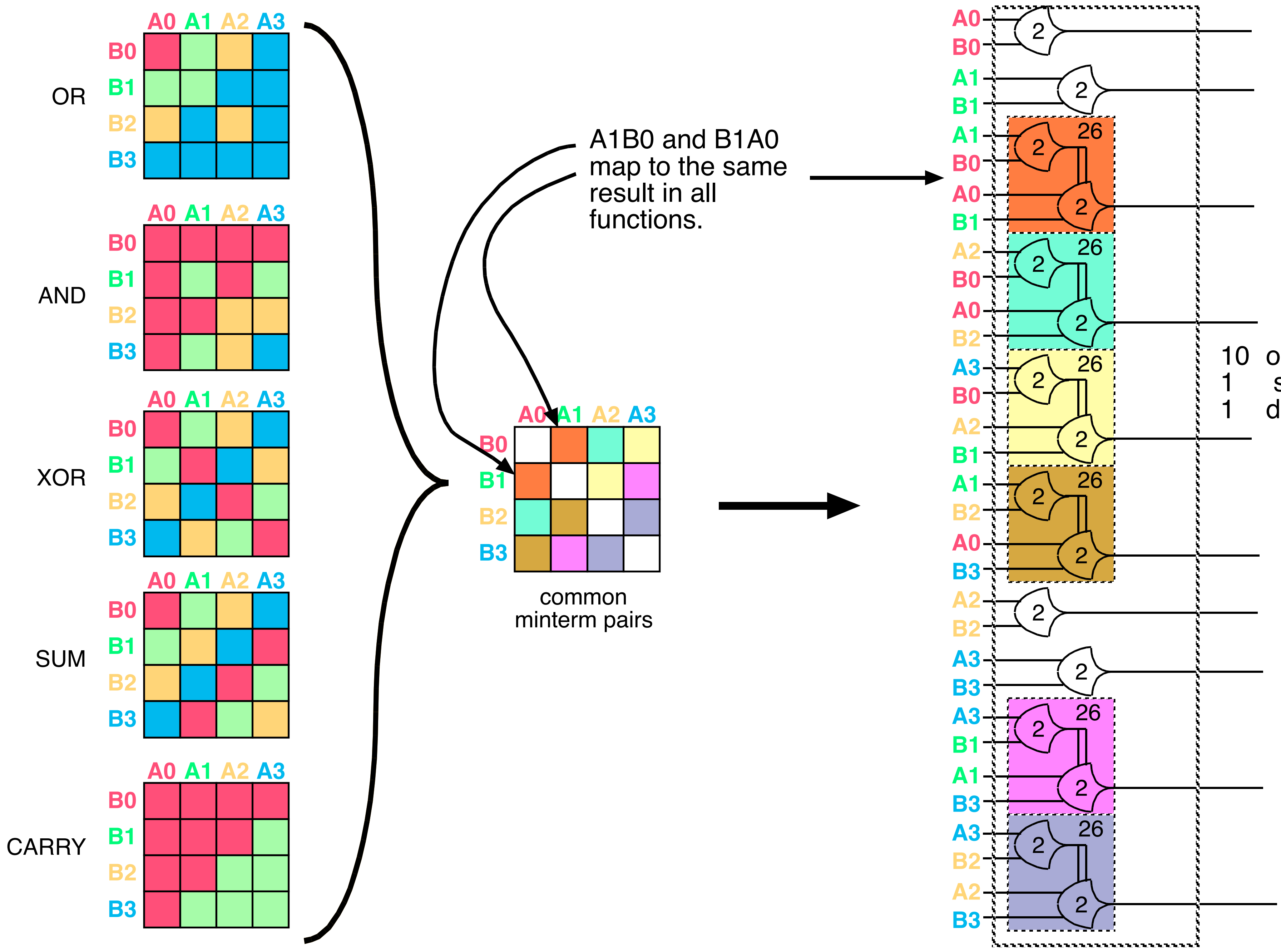
quaternary CARRY

B \ A	0	1	2	3
0	0	0	0	1
1	0	0	1	1
2	0	1	1	1
3	1	1	1	1

CARRY

	A0	A1	A2	A3
B0	🟥	🟥	🟥	🟩
B1	🟥	🟥	🟩	🟩
B2	🟥	🟩	🟩	🟩
B3	🟩	🟩	🟩	🟩

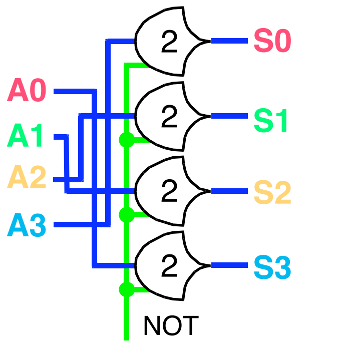
Common Minterm



A1B0 and B1A0 map to the same result in all functions.

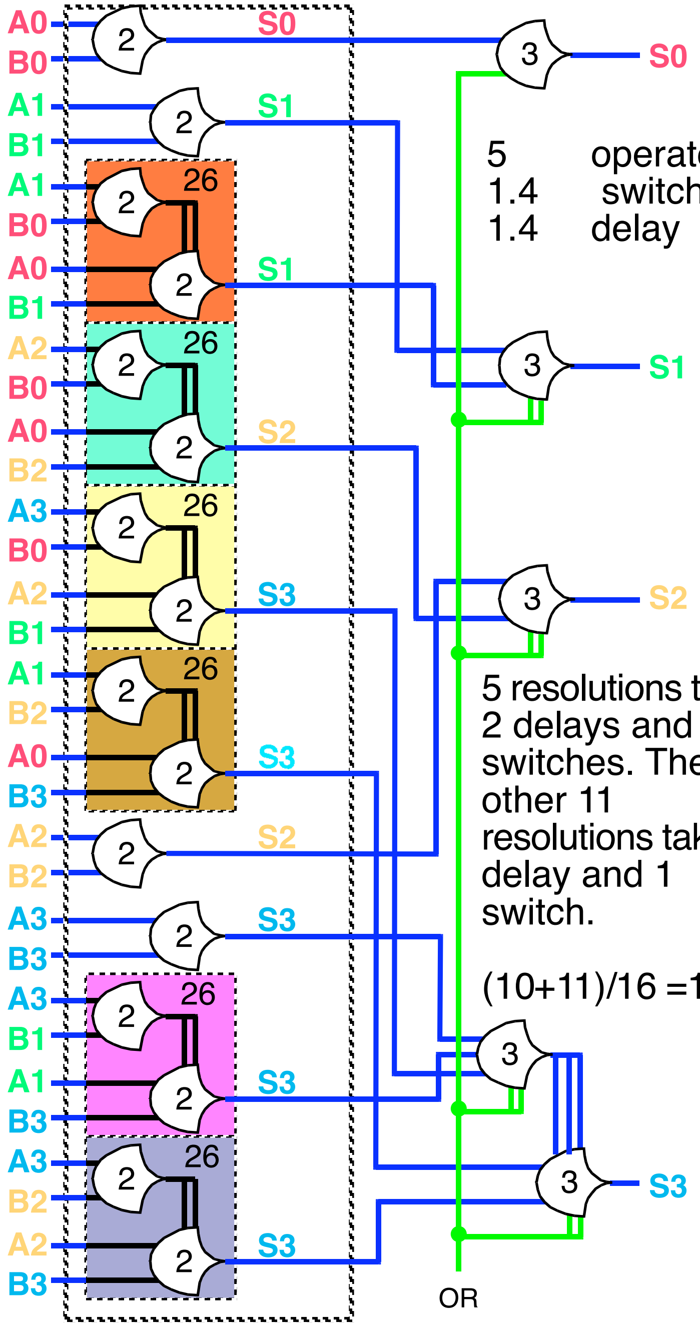
10 operators
1 switch
1 delay

Quaternary NOT



4 operators
1 switch
1 delay

Quaternary OR

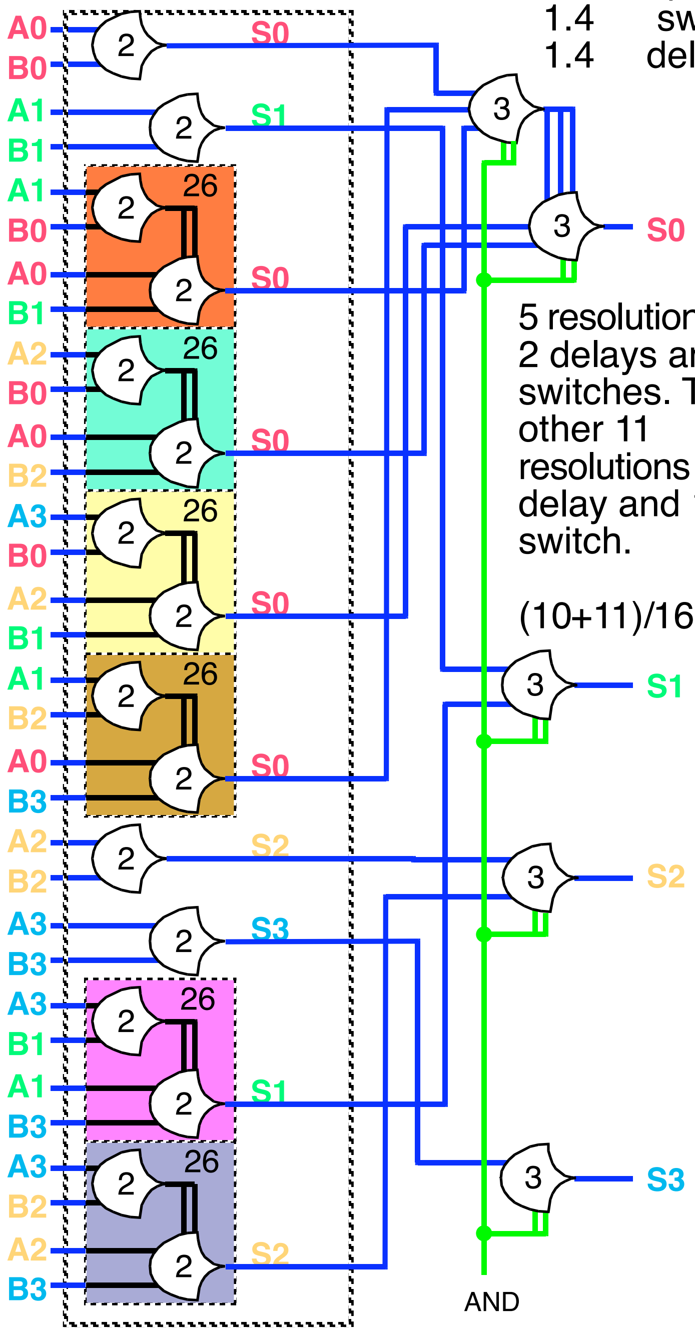


5 operators
1.4 switch
1.4 delay

5 resolutions take 2 delays and 2 switches. The other 11 resolutions take 1 delay and 1 switch.

(10+11)/16 =1.37

Quaternary AND

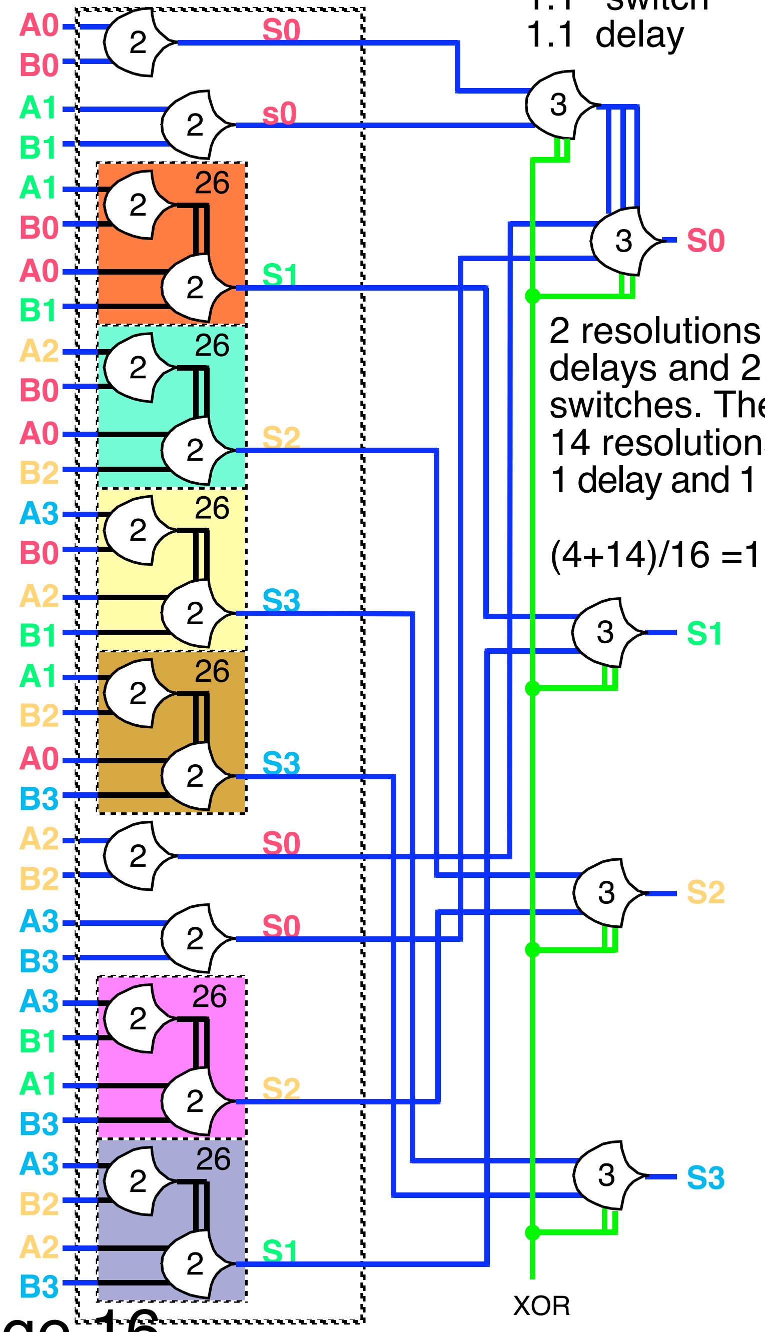


5 operators
1.4 switch
1.4 delay

5 resolutions take 2 delays and 2 switches. The other 11 resolutions take 1 delay and 1 switch.

(10+11)/16 =1.37

Quaternary XOR

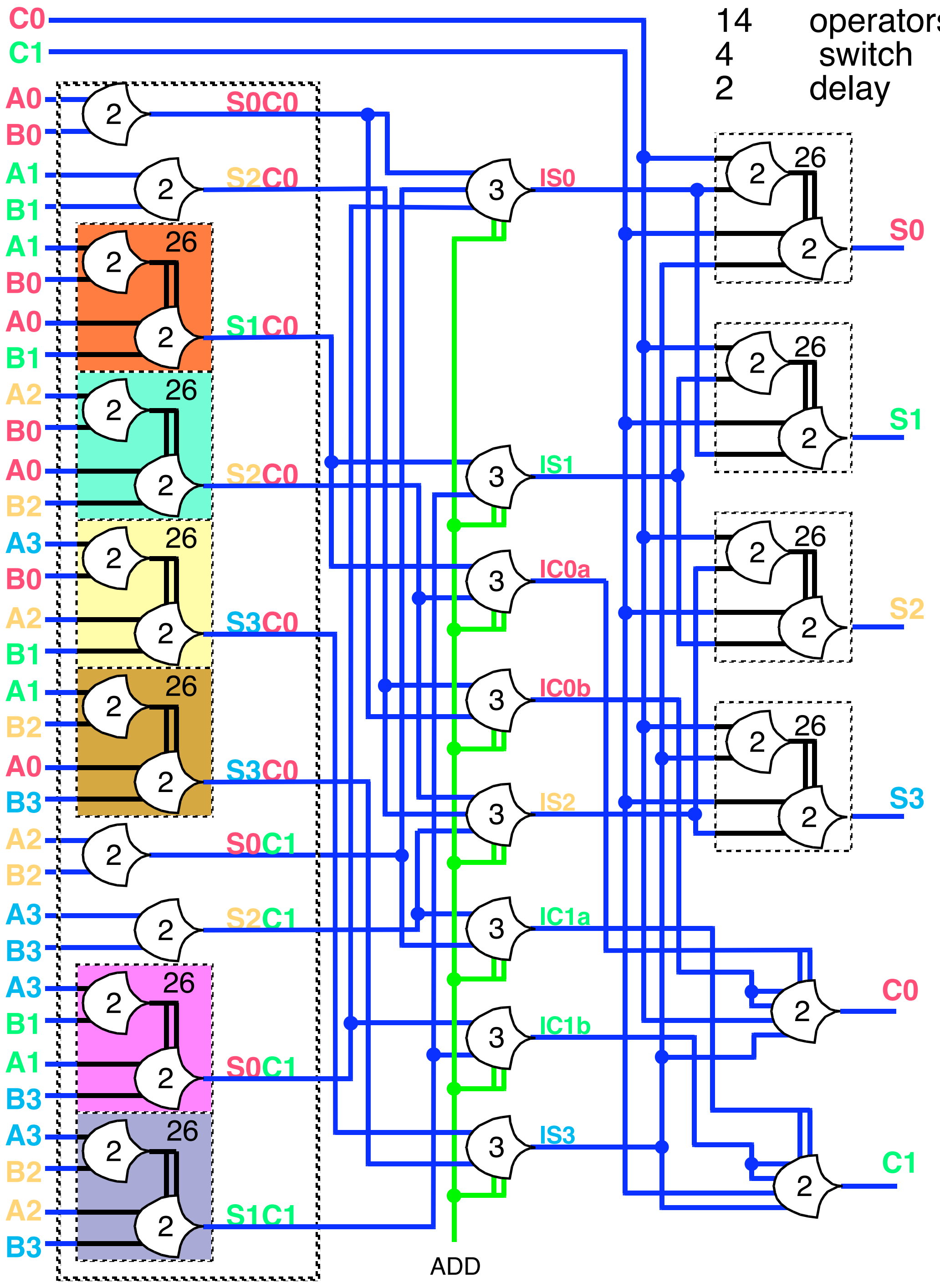


5 operators
1.1 switch
1.1 delay

2 resolutions take 2 delays and 2 switches. The other 14 resolutions take 1 delay and 1 switch.

(4+14)/16 =1.125

Quaternary ADD



14 operators
4 switch
2 delay

The Quaternary Shift Operation is Derived in Terms of Tandem Binary Shift Operations.

Shift Right one bit with carry in zero (SRC0)

in			out		
carry	quat	binary	binary	quat	carry
0	0	00	00	0	0
0	1	01	00	0	1
0	2	10	01	1	0
0	3	11	01	1	1

Shift Right one bit with carry in one (SRC1)

in			out		
carry	quat	binary	binary	quat	carry
1	0	00	10	2	0
1	1	01	10	2	1
1	2	10	11	3	0
1	3	11	11	3	1

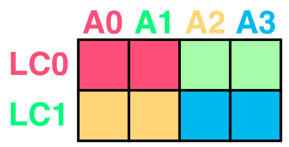
Shift Left one bit with carry in zero (SLC0)

in			out		
carry	quat	binary	binary	quat	carry
0	0	00	00	0	0
0	1	01	10	2	0
0	2	10	00	0	1
0	3	11	10	2	1

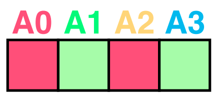
Shift Left one bit with carry in one (SLC1)

in			out		
carry	quat	binary	binary	quat	carry
1	0	00	01	1	0
1	1	01	11	3	0
1	2	10	01	1	1
1	3	11	11	3	1

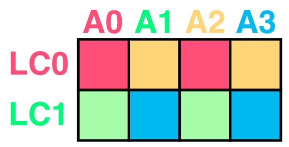
Right shift



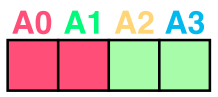
Right carry out



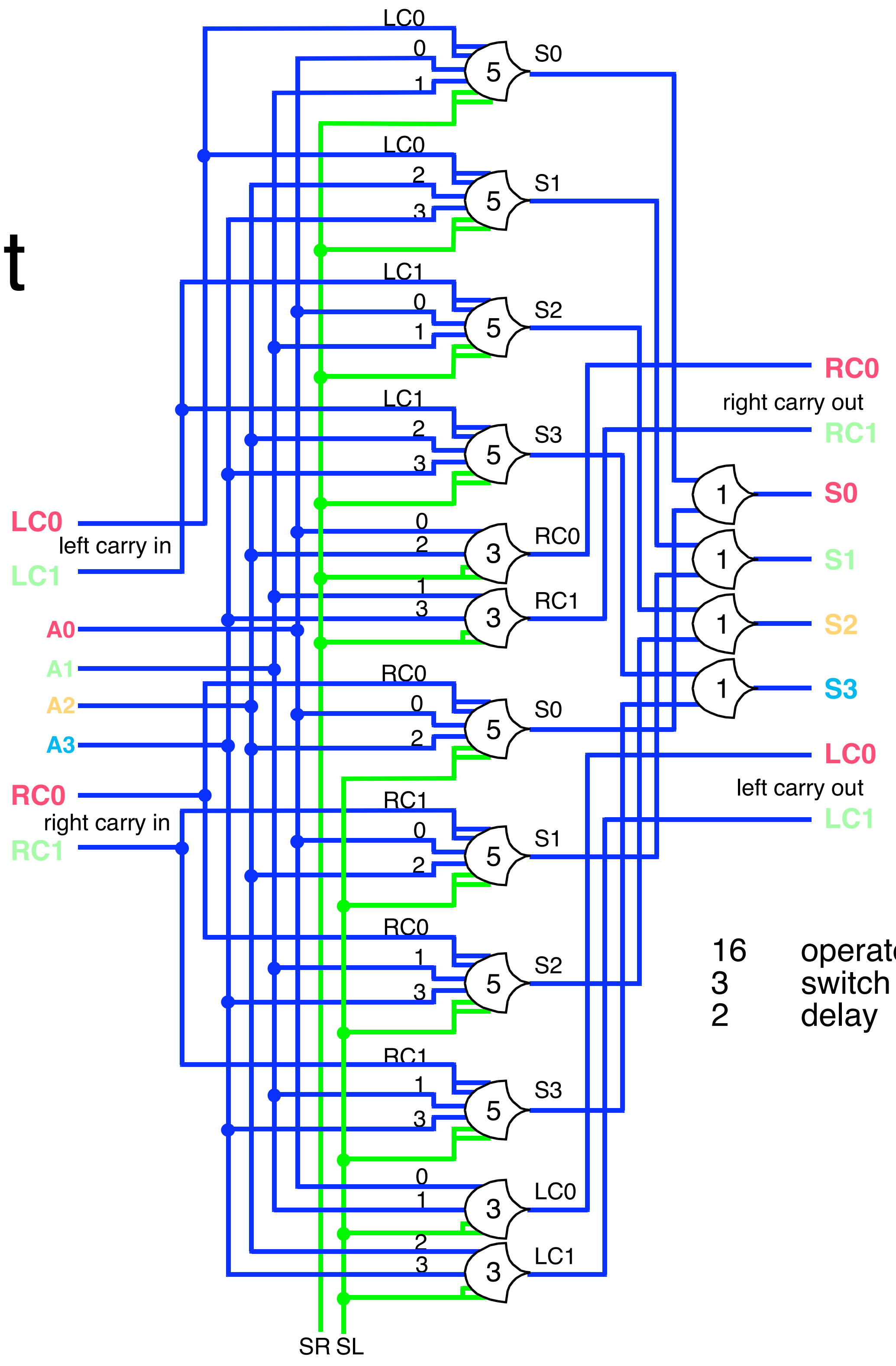
Left shift



Left carry out

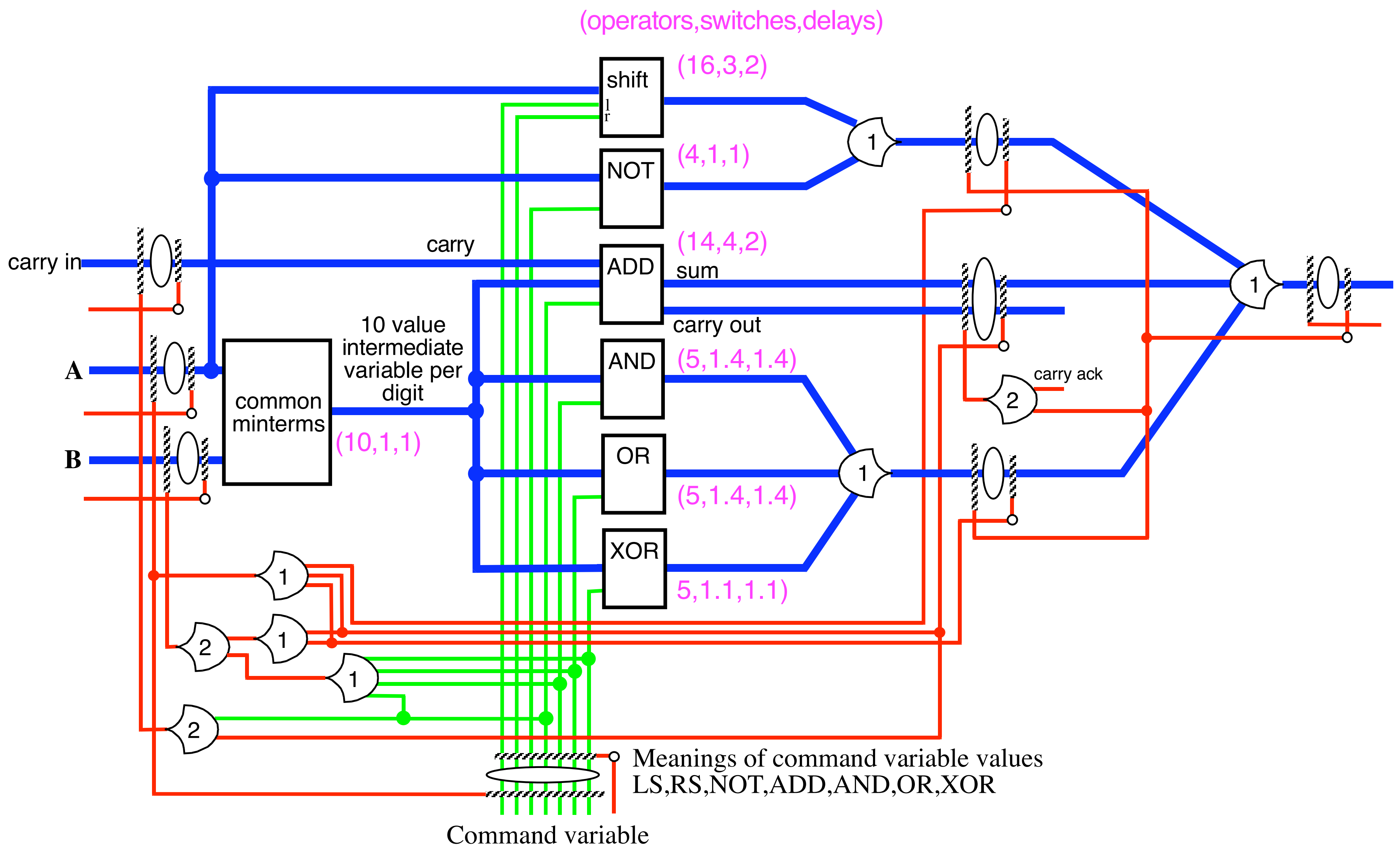


Quaternary One Bit Shift Left or Right



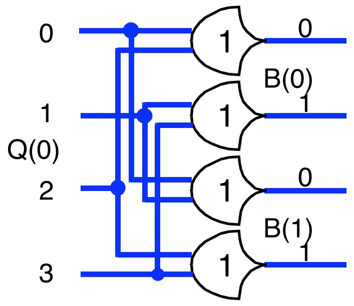
16 operators
3 switch
2 delay

Quaternary ALU



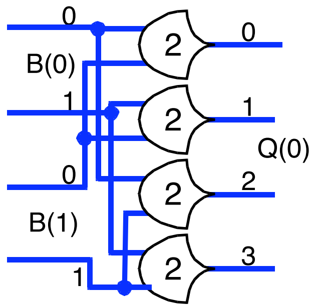
Binary Operators

Quaternary to Binary conversion



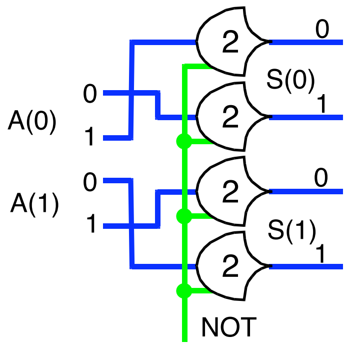
4 operators
2 switch
1 delay

Binary to Quaternary conversion



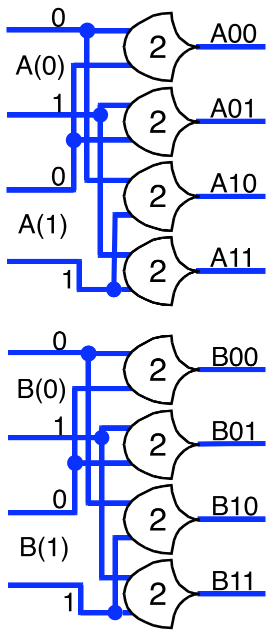
4 operators
1 switch
1 delay

Binary NOT



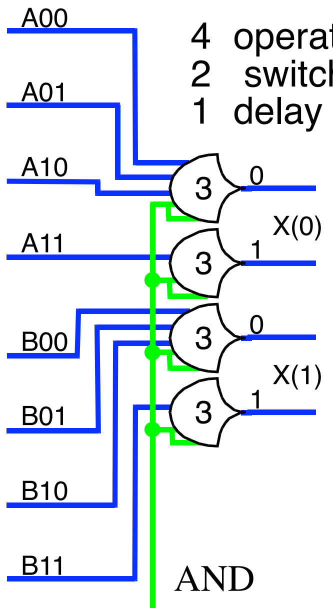
4 operators
2 switch
1 delay

Common minterm



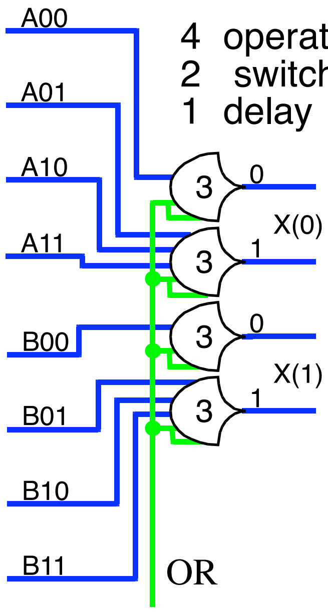
8 operators
2 switch
1 delay

Binary AND



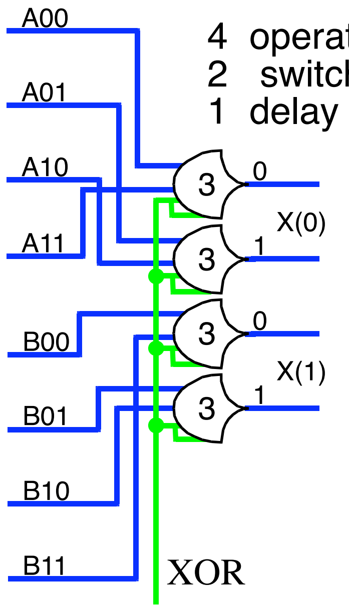
4 operators
2 switch
1 delay

Binary OR



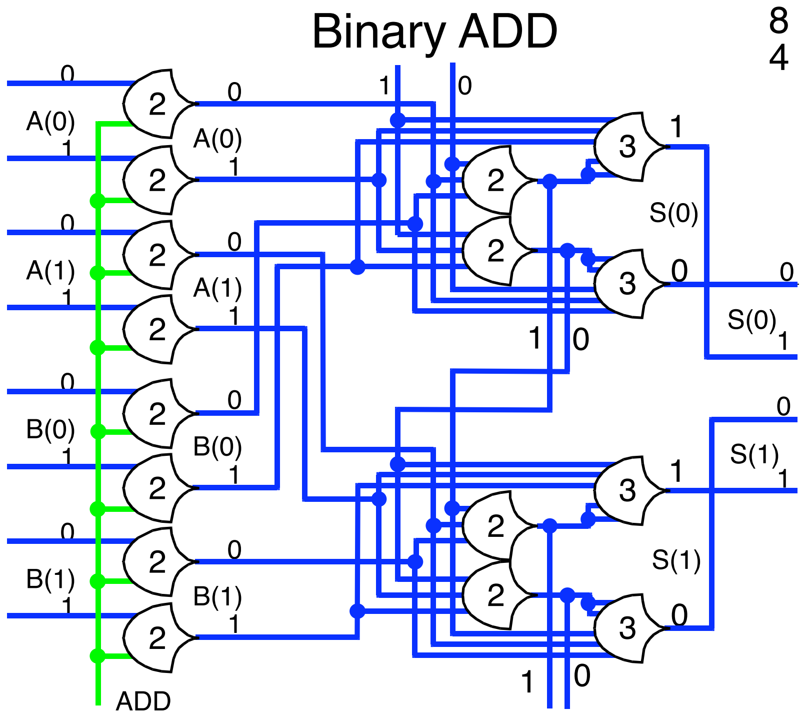
4 operators
2 switch
1 delay

Binary XOR

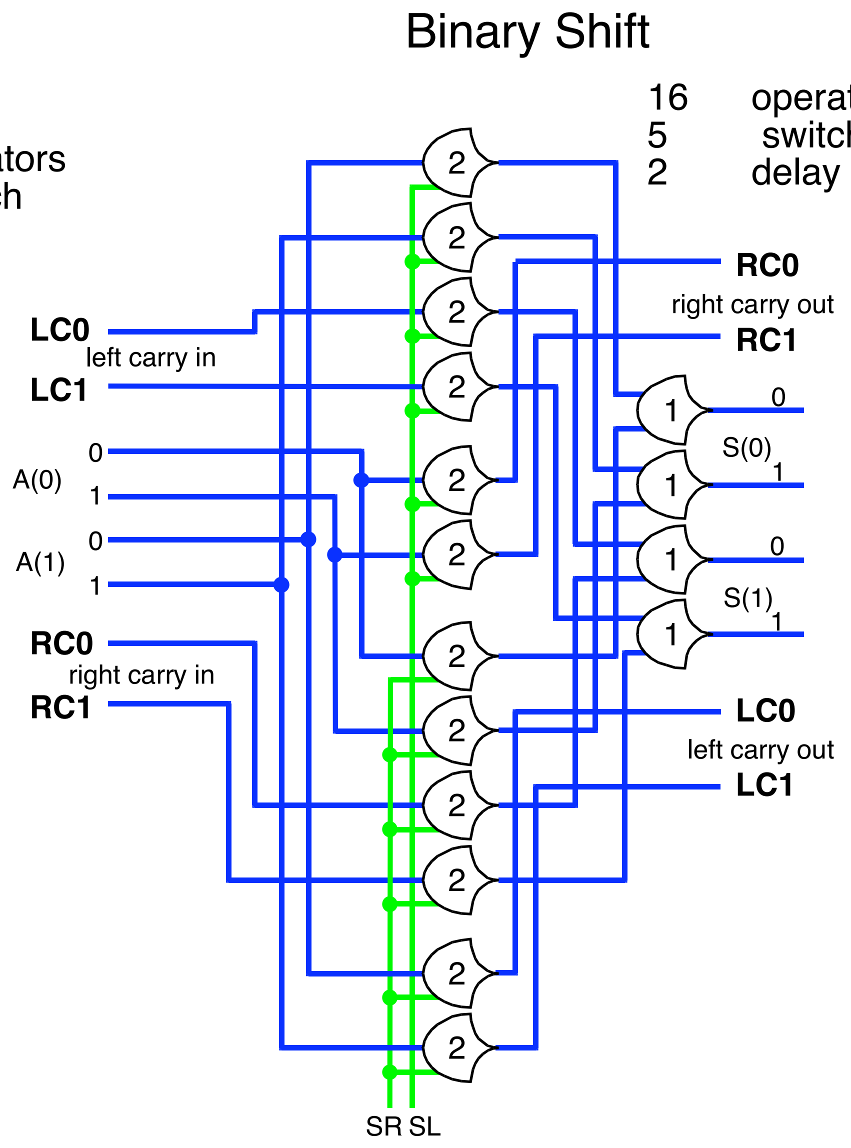


4 operators
2 switch
1 delay

Binary Operators

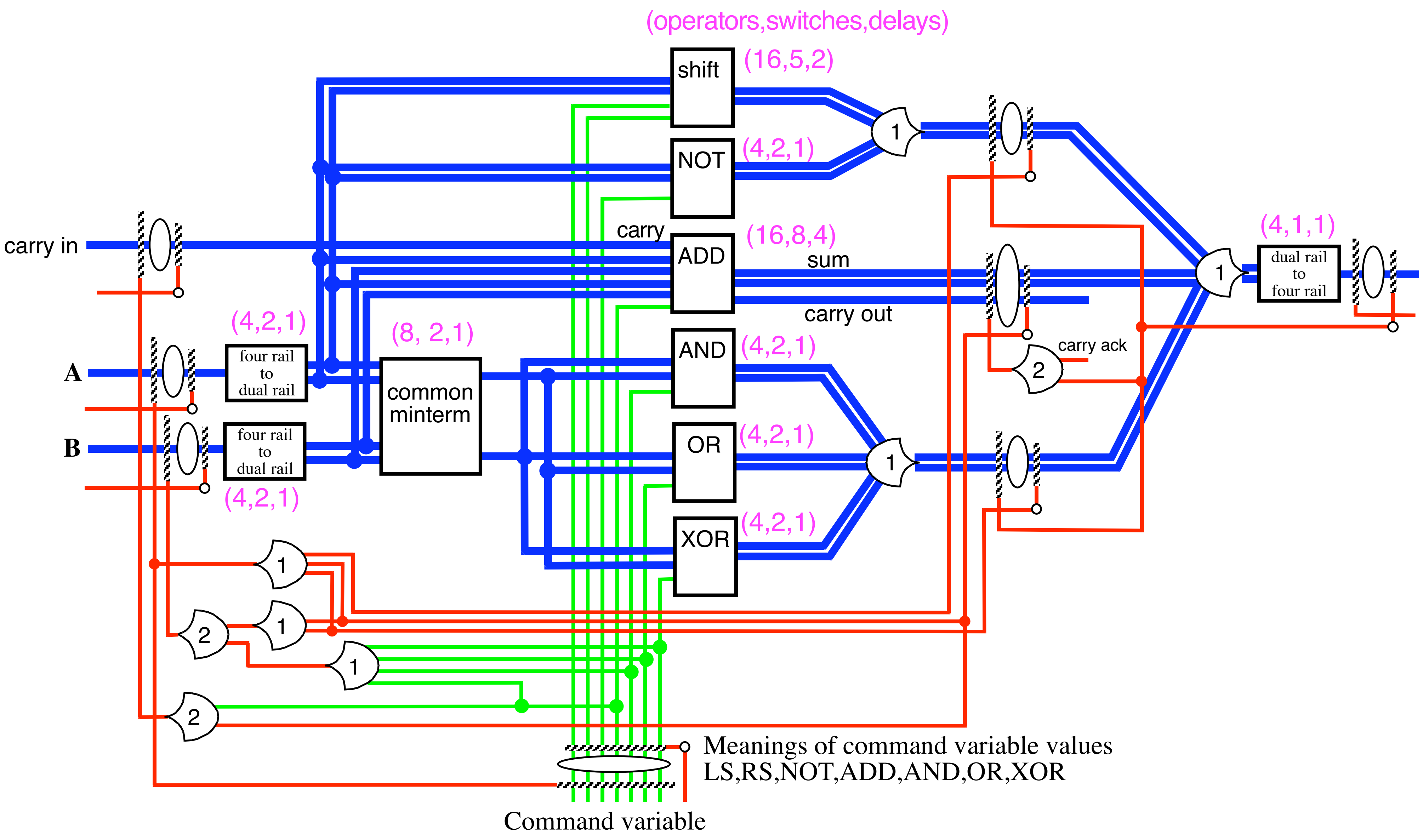


16	operators
8	switch
4	delay



16	operators
5	switch
2	delay

Binary ALU With Converison



ALU Comparisons

The Quaternary ALU is clearly competitive.

Converting from 1 of 4 to 1 of 2 and back is clearly expensive. It is much more economic to remain in 1 of 4 mode and perform quaternary arithmetic.

Is it more economical to remain entirely in 1 of 2 mode?

Removing the data path conversion from the binary ALU removes 2 delays from each operation, 5 switchings from AND, OR XOR and ADD and 3 switchings from NOT and SHIFT. The resulting ALU has slightly fewer operators but is still more expensive in terms of switching and delay and the switching cost of data path transmission doubles.

	Binary ALU with conv			*Quaternary ALU*			*Binary ALU without conv*		
	operators	*switches*	*delay*	*operators*	*switches*	*delay*	*operators*	*switches*	*delay*
4 to 2	**8**	**4**	**1**						
2 to 4	**4**	**1**	**1**						
com mi	**8**	**1**	**1**	**10**	**1**	**1**	**8**	**1**	**1**
ADD	**16**	**13**	**6**	**14**	**5**	**3**	**16**	**8**	**4**
OR	**4**	**9**	**4**	**5**	**2.4**	**2.4**	**4**	**4**	**2**
AND	**4**	**9**	**4**	**5**	**2.4**	**2.4**	**4**	**4**	**2**
XOR	**4**	**9**	**4**	**5**	**2.1**	**2.1**	**4**	**4**	**2**
NOT	**4**	**5**	**3**	**4**	**1**	**1**	**4**	**2**	**1**
SHIFT	**16**	**8**	**4**	**16**	**3**	**2**	**16**	**5**	**2**
totals	**68**	**53**	**25**	**59**	**15.9**	**12.9**	**56**	**27**	**13**

The red entries indicate differences with the table in the book. Apparently I did not vet the table as carefully as I thought I had.

The differences are minor and the conclusion remains the same.

For comparison purposes the raw totals are shown instead of dividing all the switch and delay totals by 6 to get an average per operator.

The switch and delay numbers for the operations include the conversions and minterm as a total cost of each operator.

This page intentionally blank