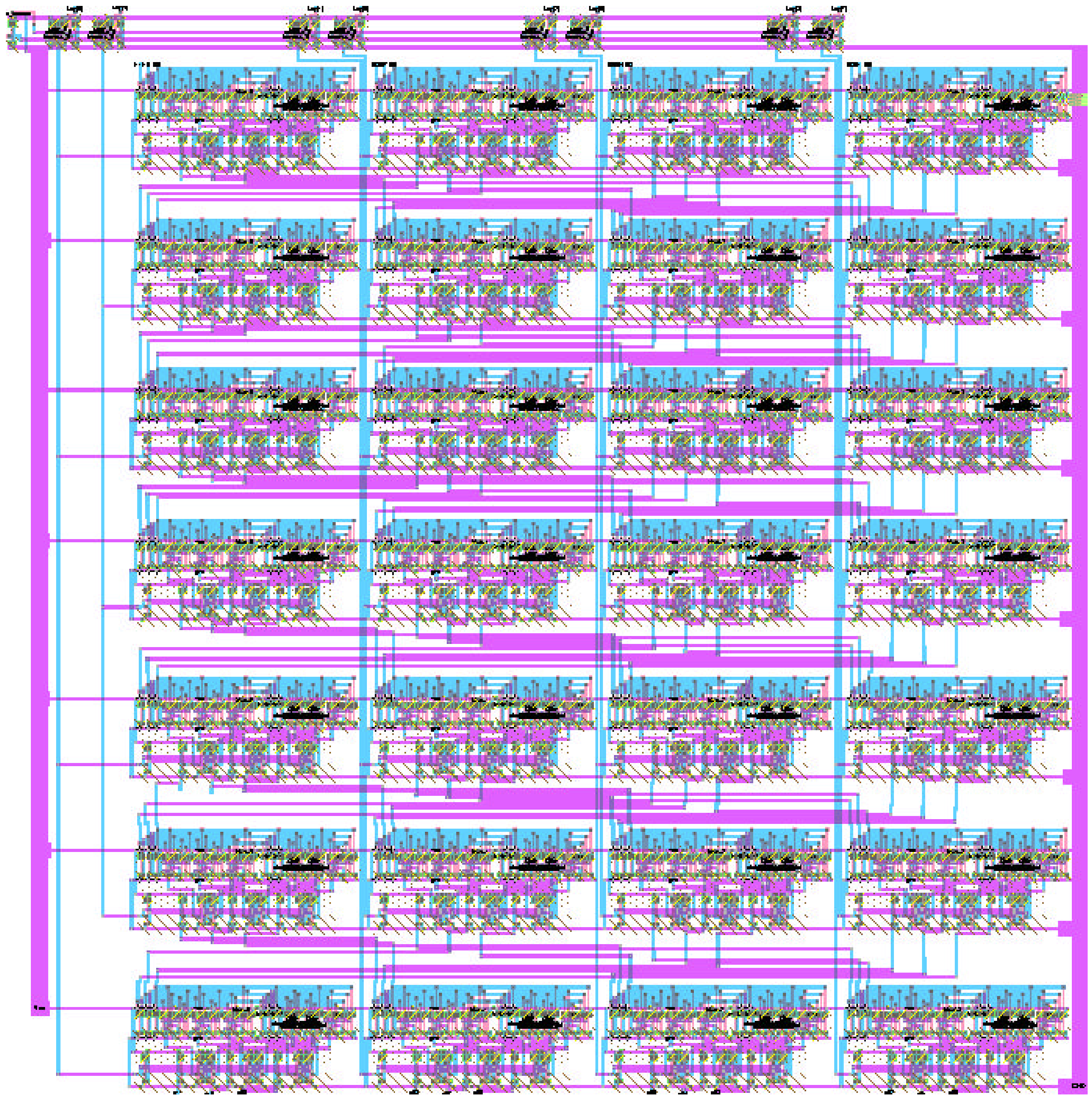
Jerod Meacham
Brooke Bassage-Glock
HMC Eng 158: CMOS VLSI
Final Project: Psuedo-DES Encryption/Decryption TinyChip



Introduction:

In the technology driven world of today many personal and private pieces of data are being transmitted over the airwaves and cables and are able to be picked up by anyone. An increased need for privacy has made the use of encryption very necessary. By implementing a hardware encryption and decryption system, the speed with which data can be encrypted/decrypted is drastically increased thus allowing for faster data transmission. We have designed a single chip that does is capable of doing both encryption and decryption.

Functional Overview:

The encryption/decryption system will be based on a basic encryption system that was used in IBM's LUCIFER encryption systems and is loosely related to the national Data Encryption Standard used by the government. The system utilizes a series of alternating substitution and permutation cells through which the twelve bit input travels. Every cell modifies the data in a specific way – by either permutating all the bits, or by substituting each of the 4 three bit sets with another 3 bit set, in a specified manner. The substitution method is based on an eight-bit user supplied key.

Technical Approach:

We designed an integrated encryptor/decryptor chip. As input data is fed through the cells, the modification will be determined by the Encode_decode input. If Encode_decode is HIGH, then the input will be encrypted; if it is low, the input will decrypted, as long as it was encrypted with this some chip.

A number of leaf cells were created in order to accomplish the different functions needed for the encryption of the data. These cells are used to construct the substitution and permutation cells, which make up the data path.

Substitution Cells:

The substitution cells operate on a 3-bit input and produce a 3-bit output. The input to the cells is substituted according to one of two possible combinational logic circuits. Using this technique, the numbers of ones and zeros may not stay the same, but will be recoverable, since the substitution is know and can be inverted.

Two different substitution cells were created for each set of 3-bit inputs. Each was created from custom-designed combinational logic circuit (see #17 and #19 in the appendix). The same three inputs goes to both cells (which are wired together, see #16). The out put used is determined by the selector box. The selector takes the six data inputs (three from each sub cell) and one select signal from the key logic (see #12 in the appendix). It uses three MUXes (#13) to pass the appropriate set of data through. The selectors and both sub cells are together in one sub box (#11). Since each sub box takes three inputs, four sub boxes are needed in a row to handle the twelve input bits. These four together are called a sub row (#10 in appendix).

Key Logic:

The 'select' signal is generated by doing a logical XNOR on the Encode_decode input and each bit of the key input key (see #2 in the appendix for the key logic).

Changing Encode_decode effectively inverts the select signal. The select signal is distributed to substitution boxes in order. Since it is an 8 bit key, and there are 4 sub boxes per row, the same four key bits go to every other sub row. Because the data path is symmetric (starts and ends with a sub row that takes the same select bits as input), inverting the select signal by inverting Encode_decode changes the functionality of the whole chip from encryption to decryption, or visa versa.

Permutation Cells:
Permutation cells simply change the order of the bits in the input. In our case, there is 12 bits going into each permutation cell and 12 bits out of each permutation cell. In addition, since only the order of the bits is changing, there will be the same number of ones and zeros in the input and output data. By making the second three permutation blocks the inversion of the first three, we can use the same logic to encrypt and decrypt. The three unique permutations can be seen in #4-6 in the appendix, with their inverses, #7-9.

Encryptor/Decryptor:
The Substitution and Permutation cells combine with the key logic to make the encryption/decryption block. It is constructed from a series of alternating substitution and permutation cells. The data flows through all these cells and the output is an encrypted or decrypted version of the input.

Simulation results:

All cells simulated correctly with the exception of the toplevel layout. We were unable to even write an IRSIM deck for the layout of the toplevel for quite some time. Consequently we were unable to get a verified simulation for the toplevel layout. However, all other cells simulate correctly. The toplevel schematic simulates correctly as seen in the following page. The easiest way to simulate the design is to chose a random key and input and set encode_decode to high. This will encrypt the data. Then, take the encrypted output, set that as the input, set encode_decode to low and run it through again with the same key. It should give the original input. Also, note that the output from the simulation is unique and encrypted (that is to say, there is no regularity and not necessarily the same number of 1's and 0's in the input and output).

Postfabrication Test Plan:
We have written a random binary generator to generate keylogic and input data. The data would have to be run through as given in the simulation results.

Pins:
The following pins will be needed:

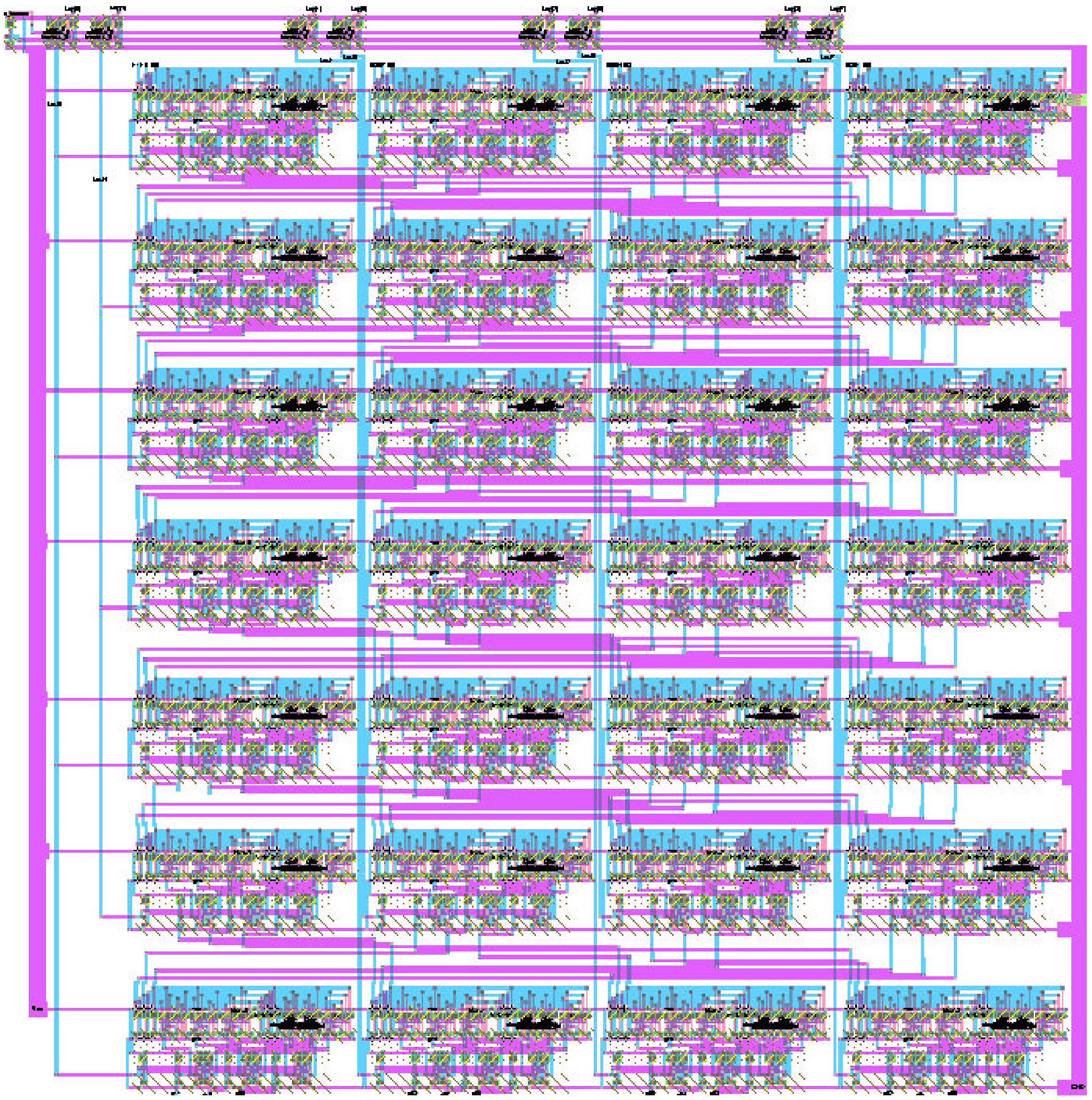
Pin Range	**Function**	**Description**
0-6	Vdd/GND	Required power and ground pins
7-18	Inputs	Data inputs for operation
19-30	Outputs	Data outputs after operation

31-39	Key inputs	Encryption Key
40	Select	Operation select (Encryption/decryption)

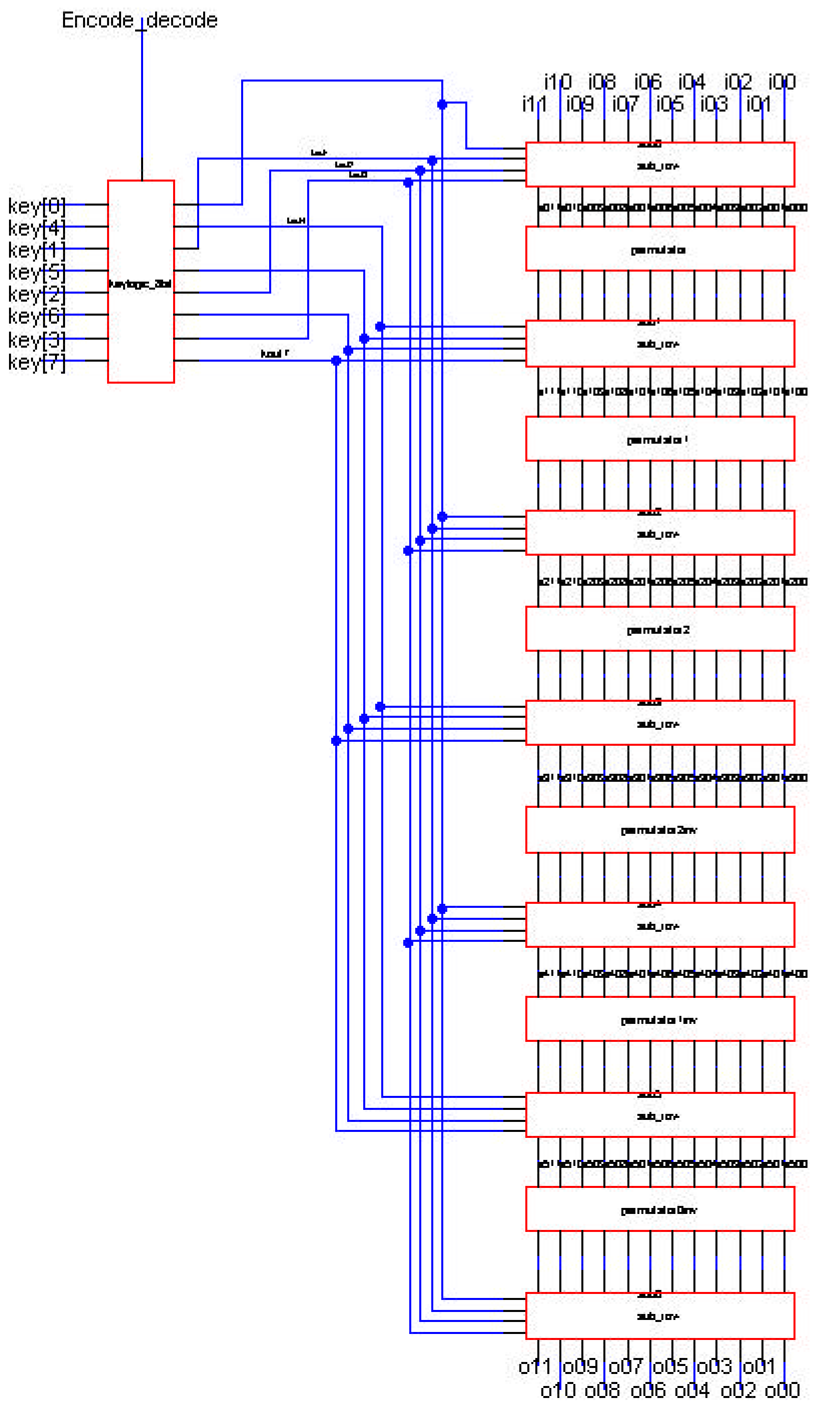
Verification Results and Area and Design Time Data:

Cell Name	Complexity	schematic	Layout	DRC	ERC	NCC	area	time
Toplevel	2	X	X	X	X		2193.5 x 2200	5
Keylogic_8bit	3	X	X	X	X	X	1704 x 87	4
keyxor	4	X	X	X	X	X	74.5 x 78	3
permutator0	1	X	X	X	X	X	1661 x 87.5	0.5
permutator1	1	X	X	X	X	X	1660.5 x 87.5	0.5
permutator2	1	X	X	X	X	X	1661 x 87.5	0.5
permutator0inv	1	X	X	X	X	X	1660 x 87.5	0.5
permutator1inv	1	X	X	X	X	X	1660.5 x 87.5	0.5
permutator2inv	1	X	X	X	X	X	1661 x 87.5	0.5
inv	1	X	X	X	X	X	25 x 68.5	0.1
s_block	2	X	X	X	X	X	455 x 135	0.5
sub_row	2	X	X	X	X	X	1908.25 x 229	2
s+selector_block	3	X	X	X	X	X	465 x 229	2
sub0	3	X	X	X	X	X	213 x 113.5	4
sub1	3	X	X	X	X	X	251.5 x 113.5	4
c(a+b)+ab.bb.cb	5	X	X	X	X	X	73.5 x 77	6

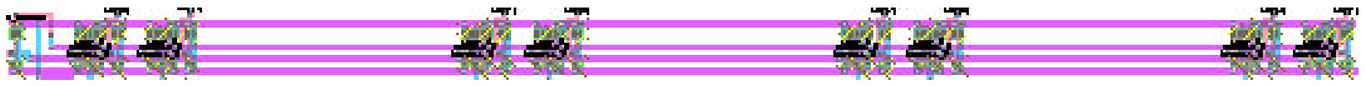
ab.b+a.bb	4	X	X	X	X	X	51.5 x 73	6
selector	3	X	X	X	X	X	402 x 113	5
mux2	3	X	X	X	X	X	66 x 87	0
tri	3	X	X	X	X	X	38 x 87	0

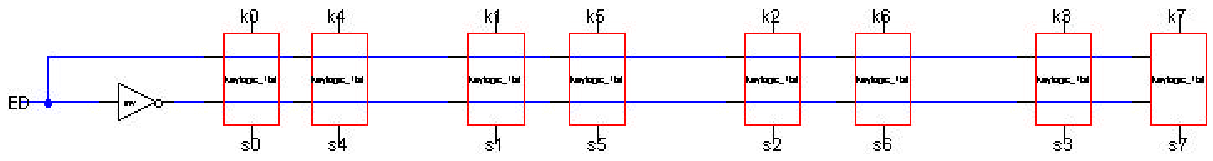


Top Level Layout

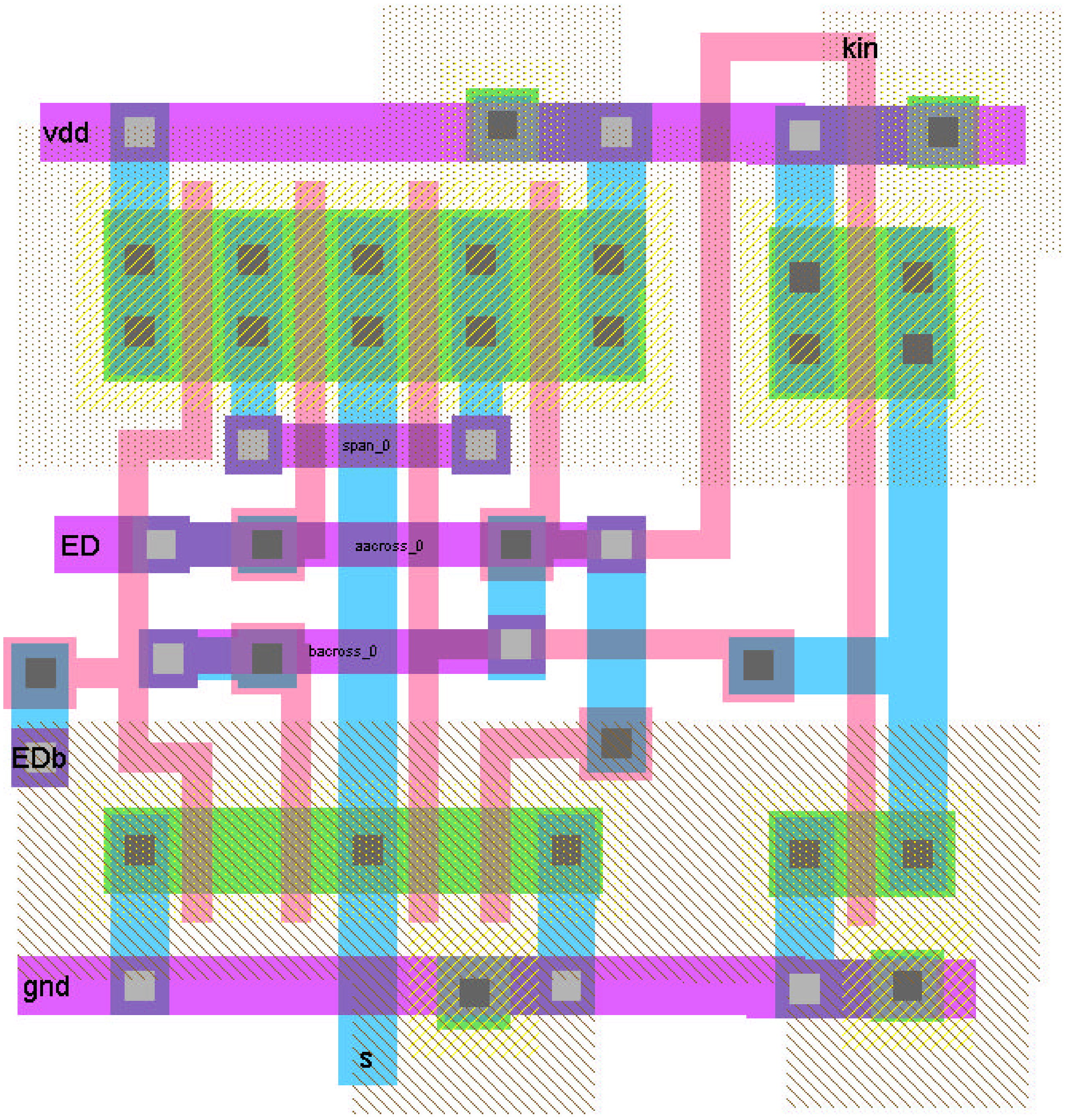


Top Level Schematic

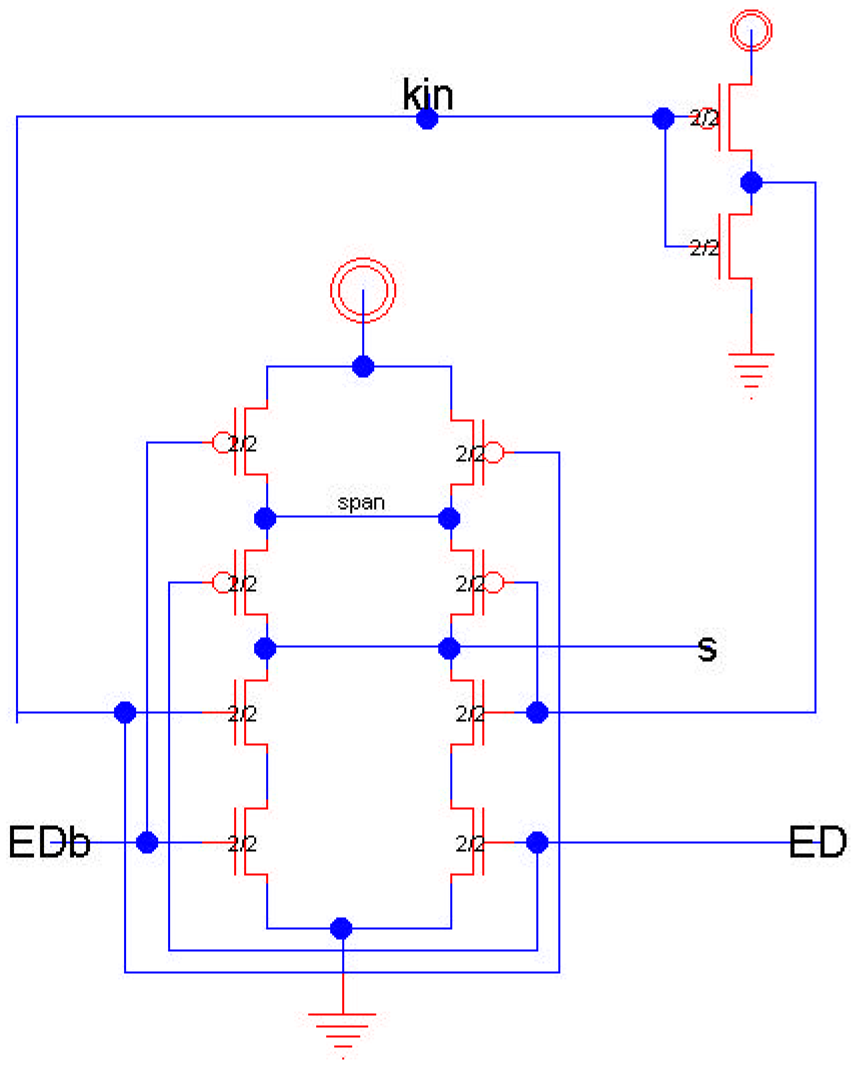

Key Logic Layout



Key Logic Schematic

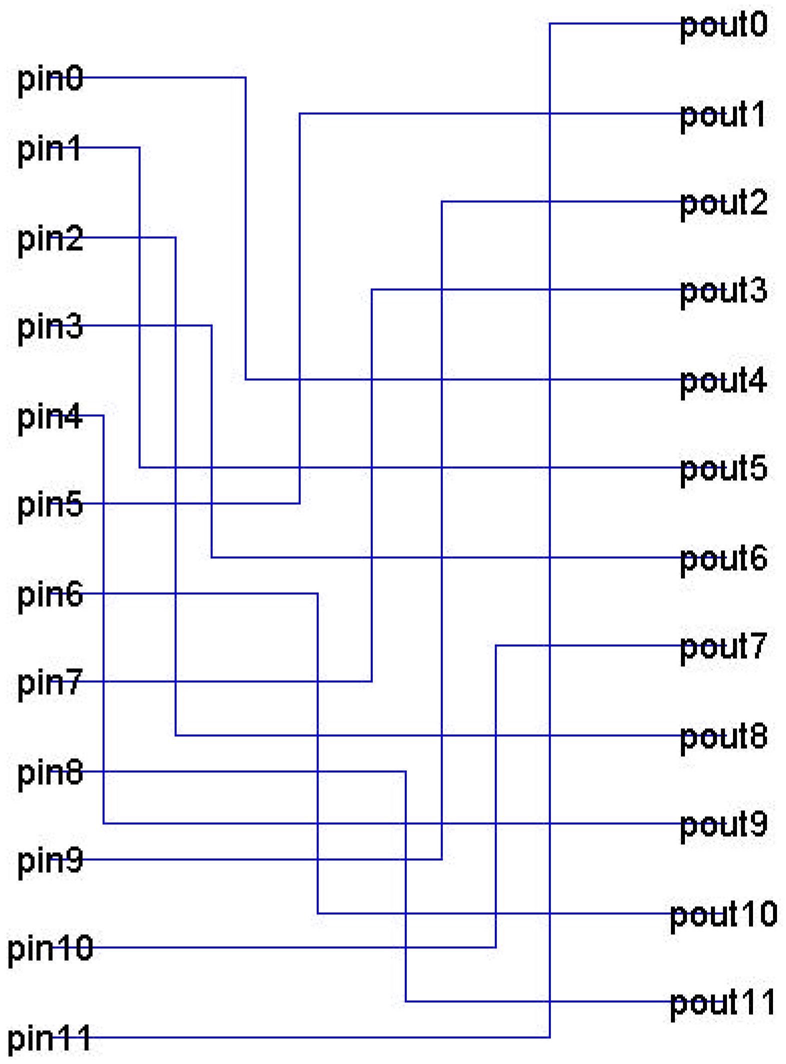


One bit Key XNOR Layout

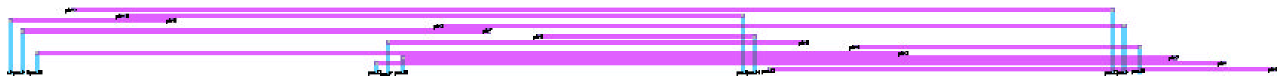


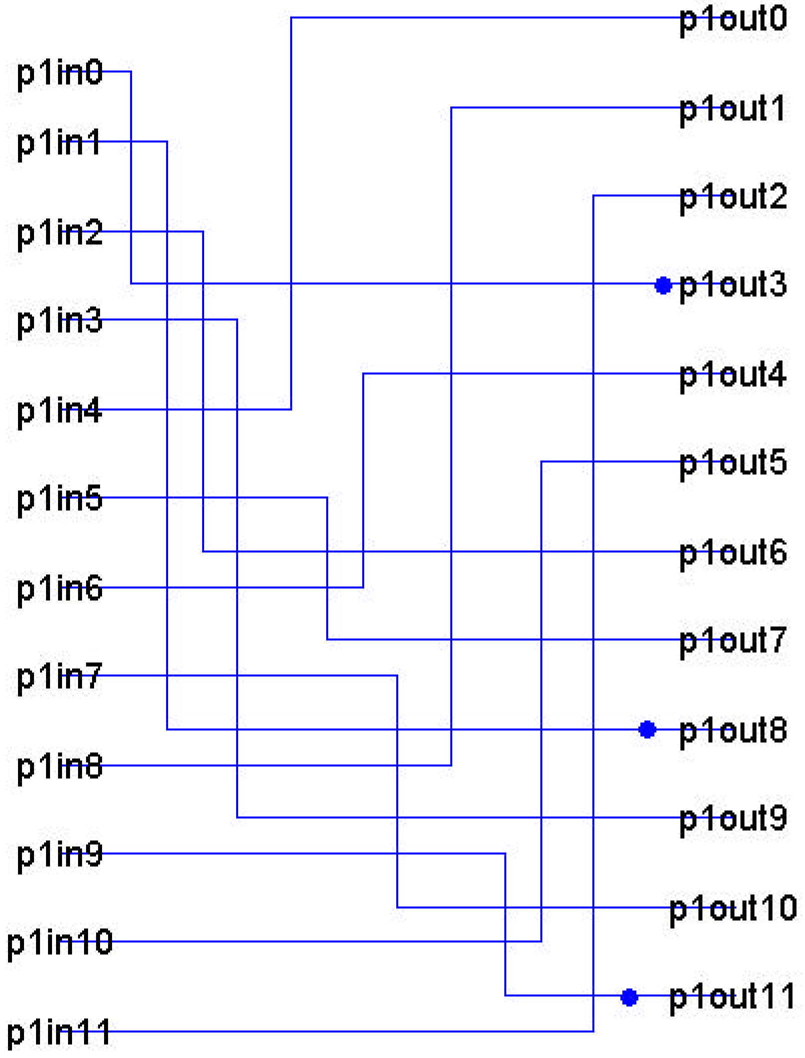
One bit Key XNOR Schematic

Permutation Block 0 Layout



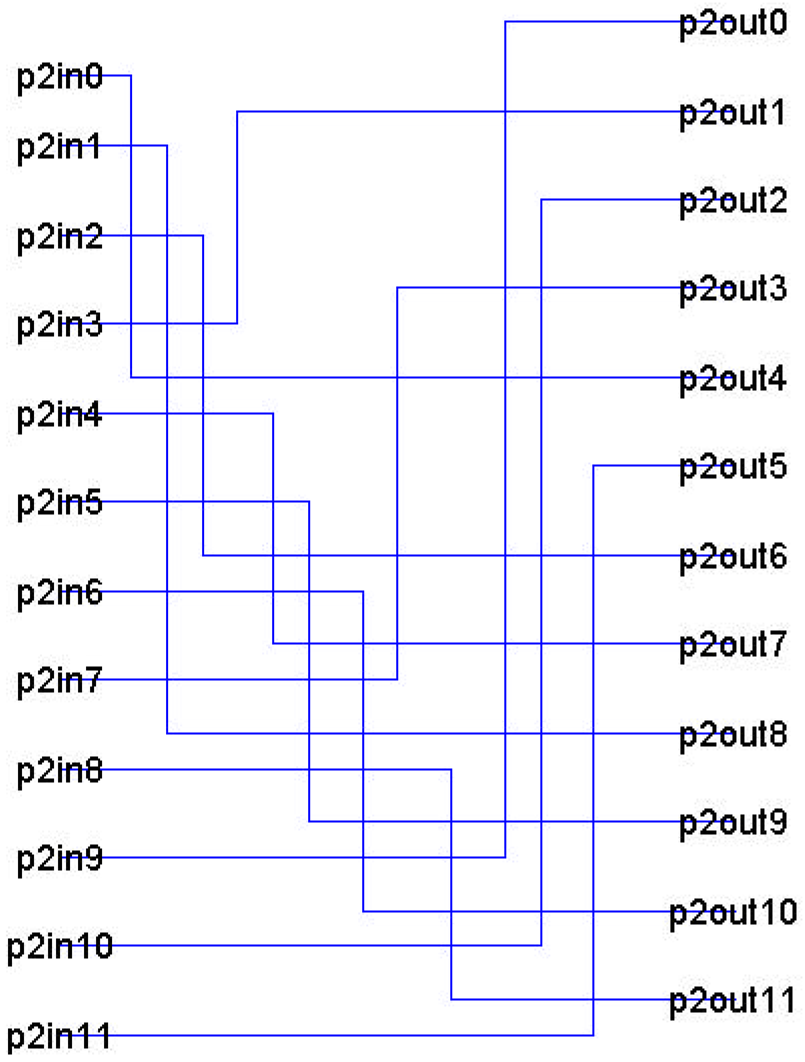
Permutation Block 0 Schematic


Permutation Block 1 Layout

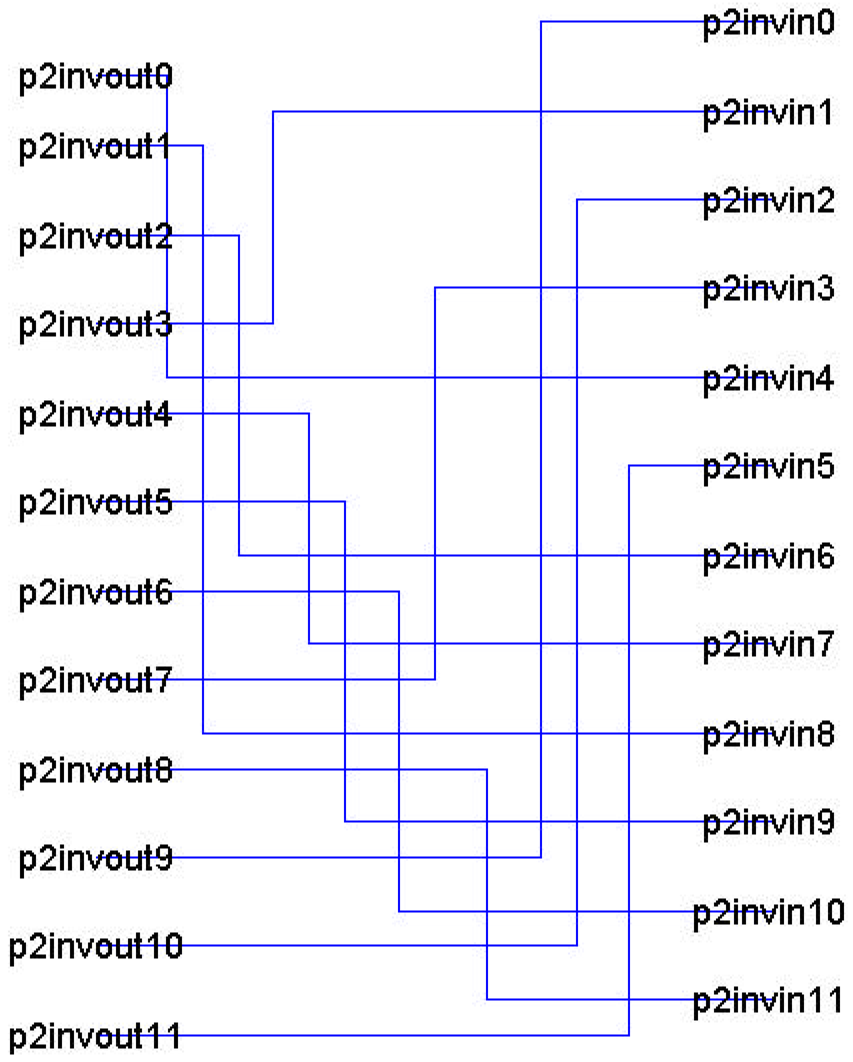


Permutation Block 1 Schematic

Permutation Block 2 Layout

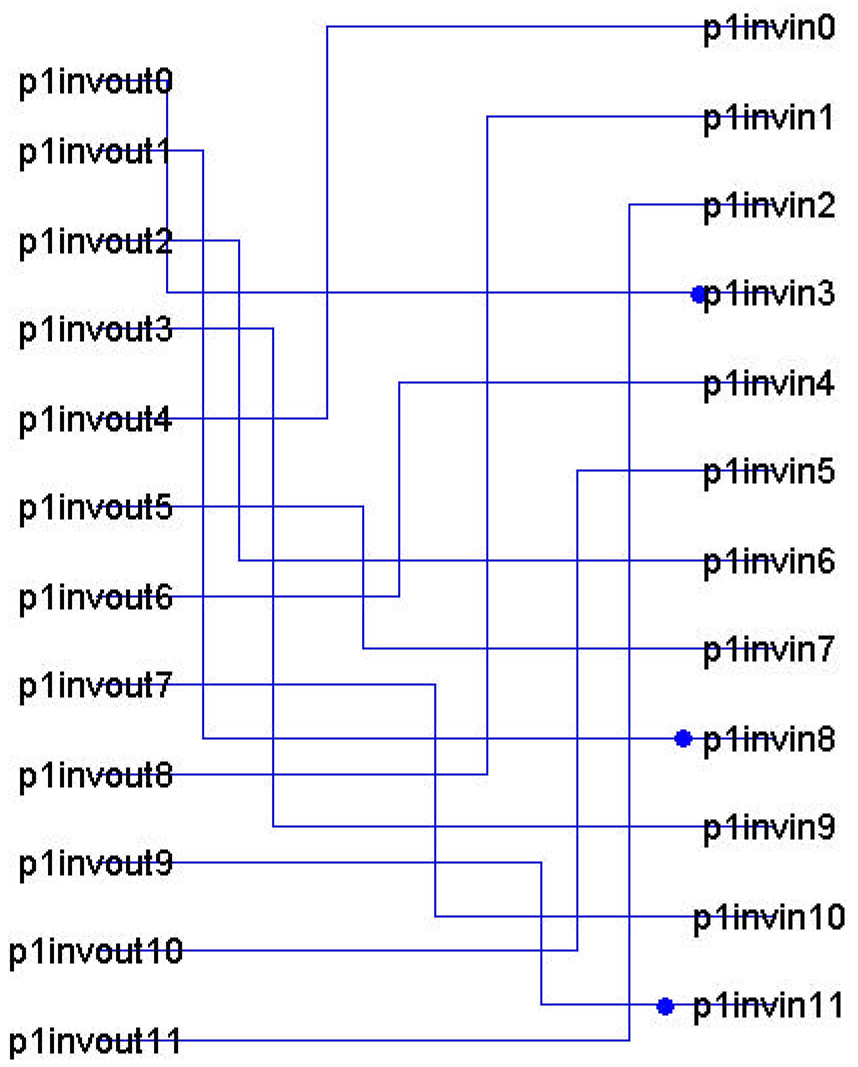

p2in0
p2in1
p2in2
p2in3
p2in4
p2in5
p2in6
p2in7
p2in8
p2in9
p2in10
p2in11
p2out0
p2out1
p2out2
p2out3
p2out4
p2out5
p2out6
p2out7
p2out8
p2out9
p2out10
p2out11

Permutation Block 2 inverse Layout



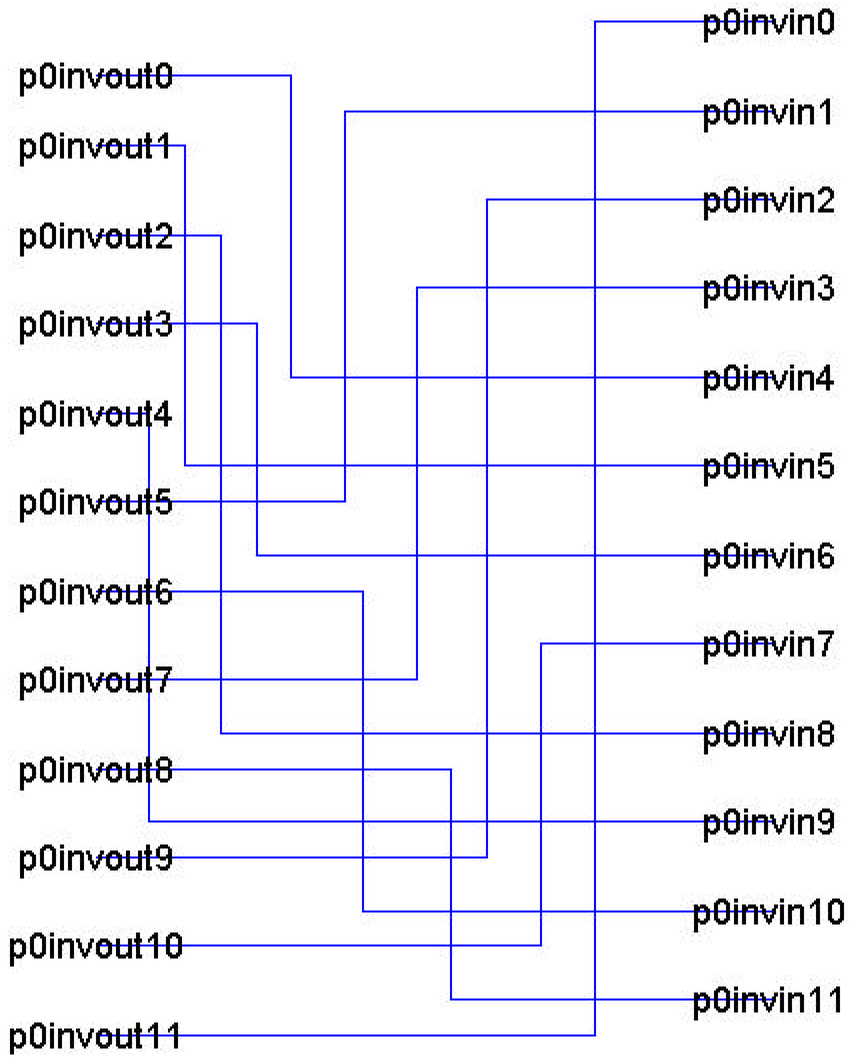
Permutation Block 2 inverse Schematic

Permutation Block 1 inverse Layout

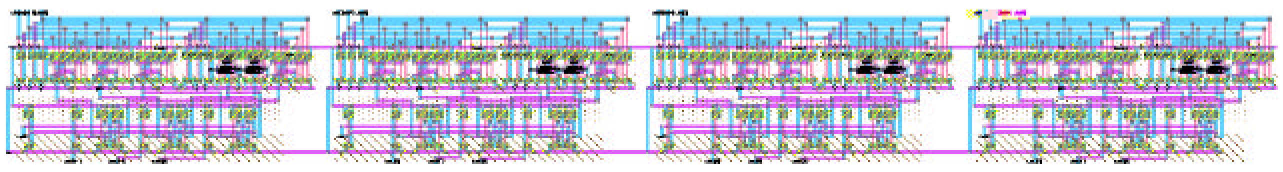


Permutation Block 1 inverse Schematic

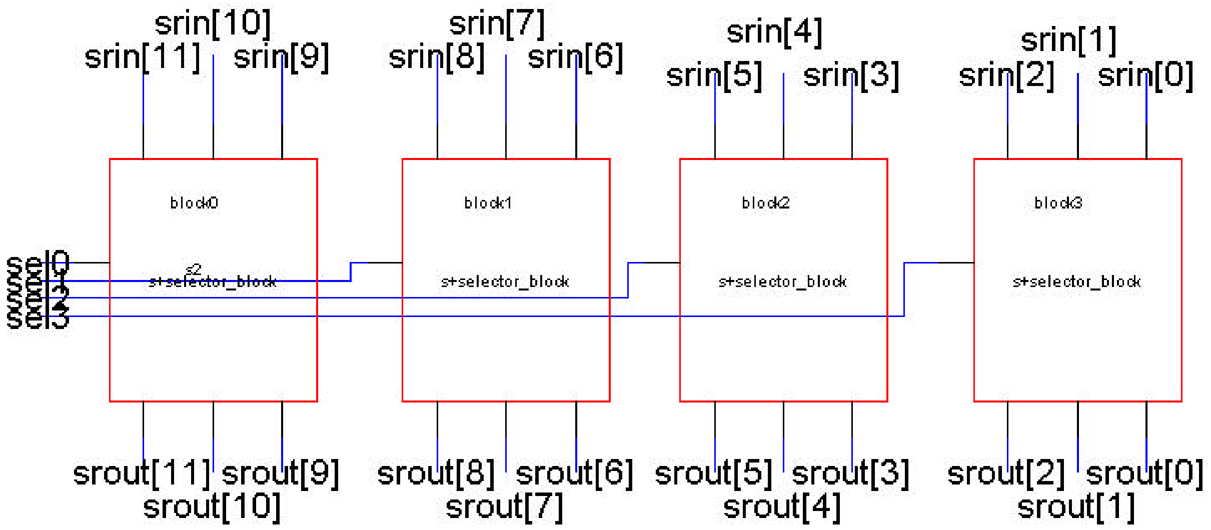
Permutation Block 0 inverse Layout



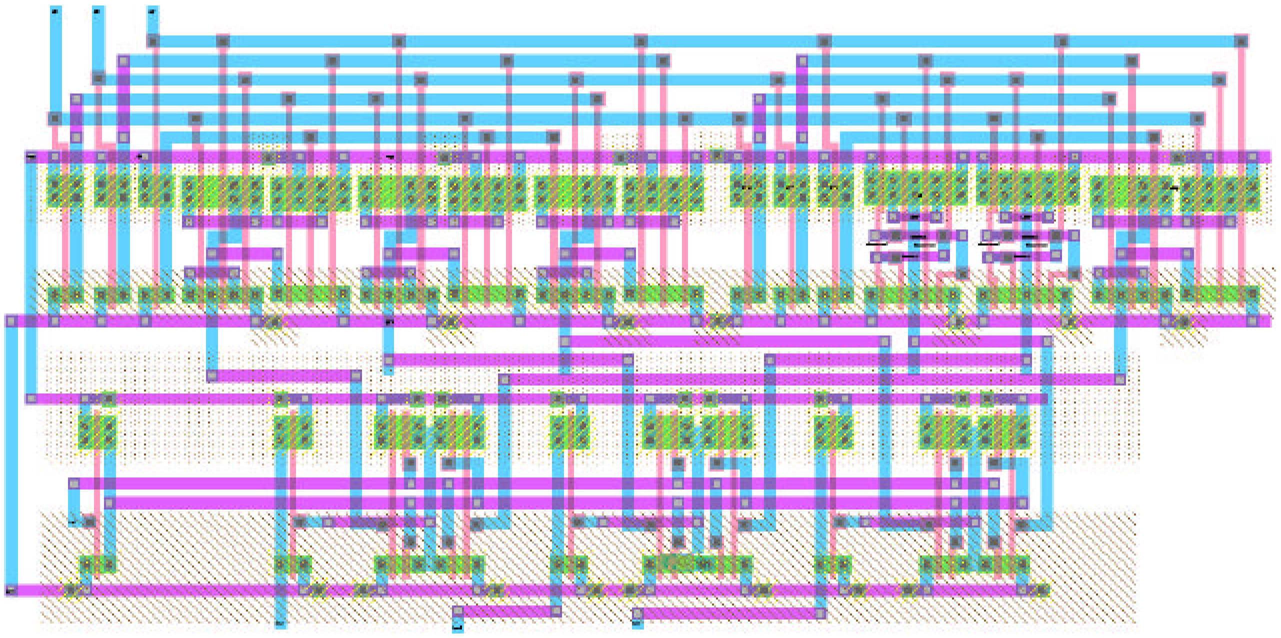
Permutation Block 0 inverse Schematic

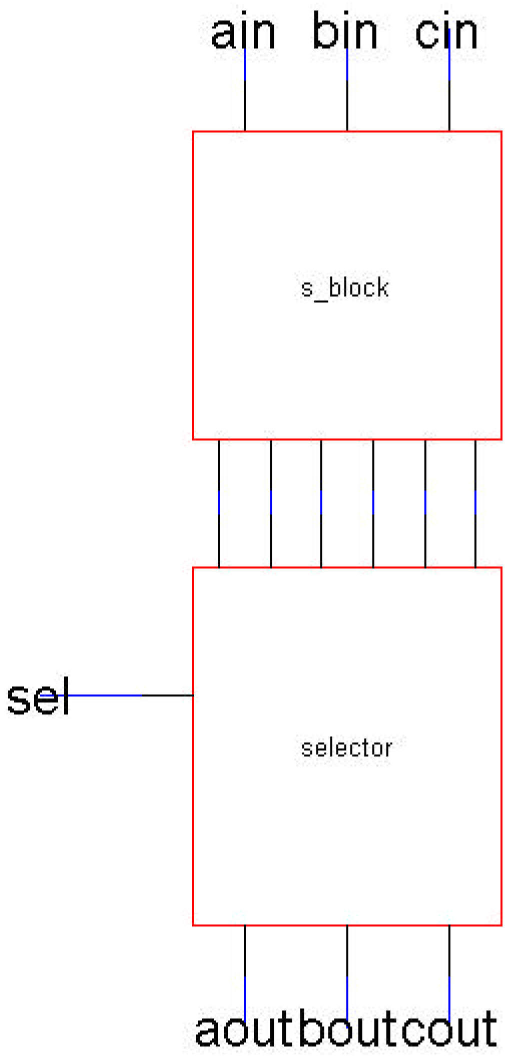


Sub Row Layout

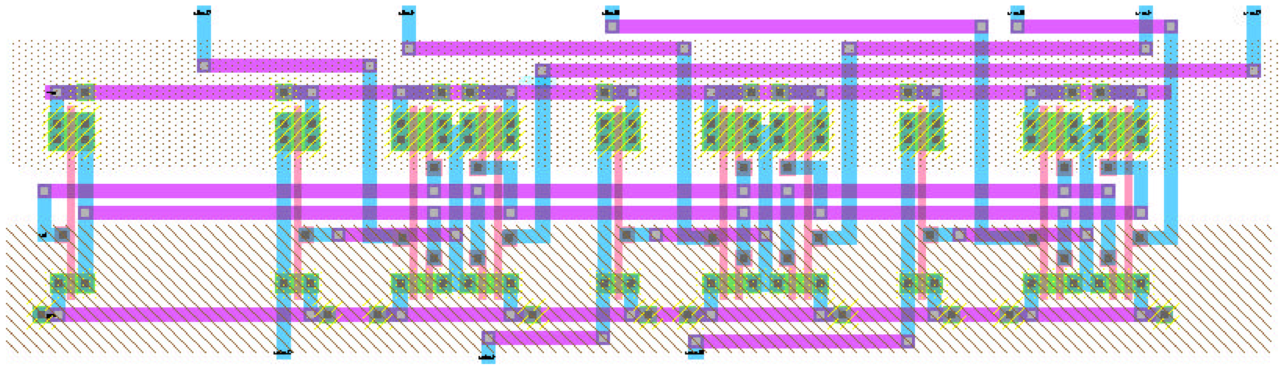


Sub Row Schematic

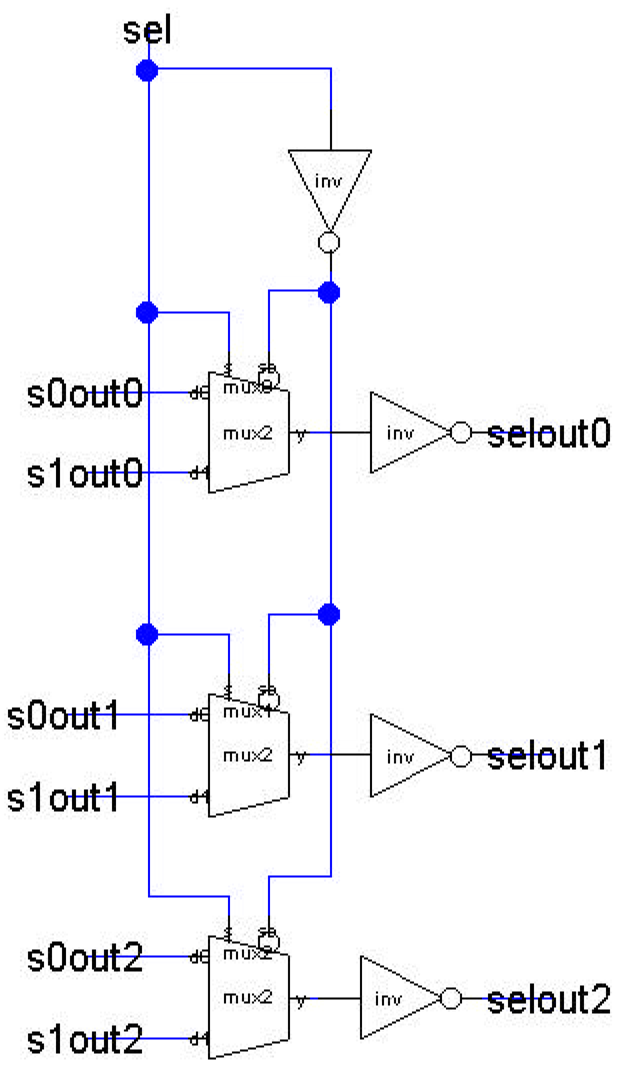

Sub Block + Selector Layout



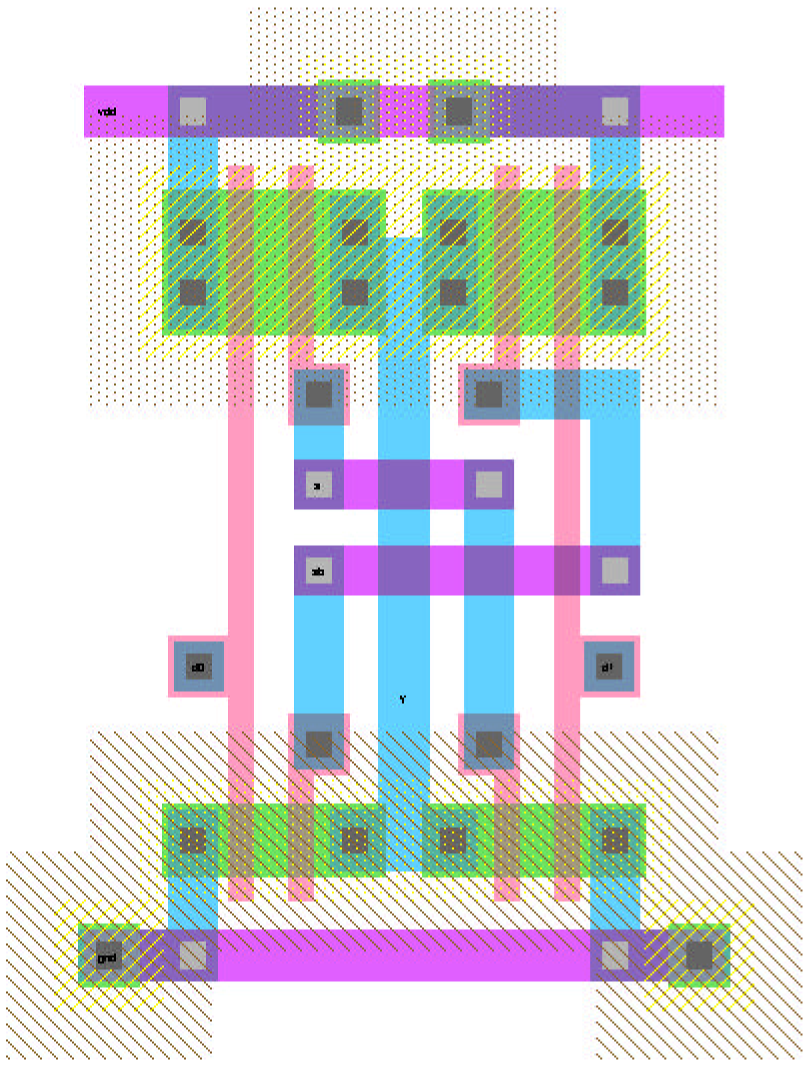
Sub Block + Selector Schematic



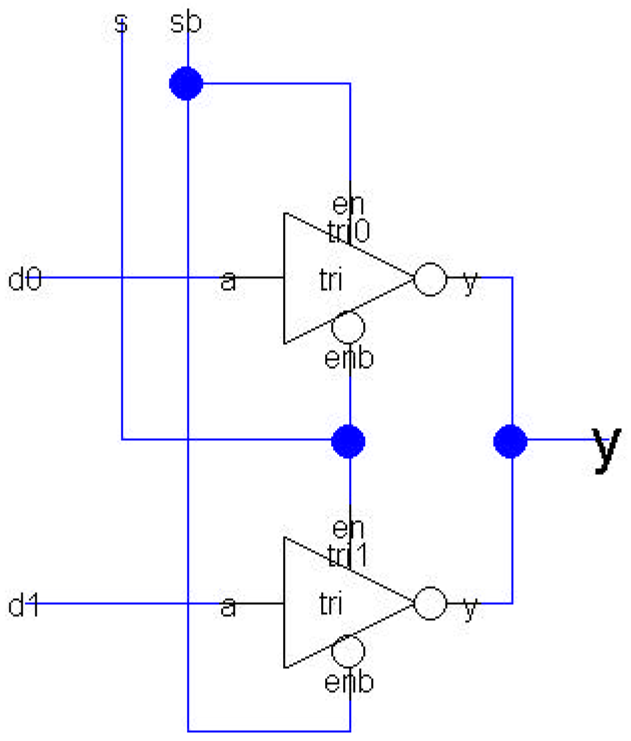
Selector Layout



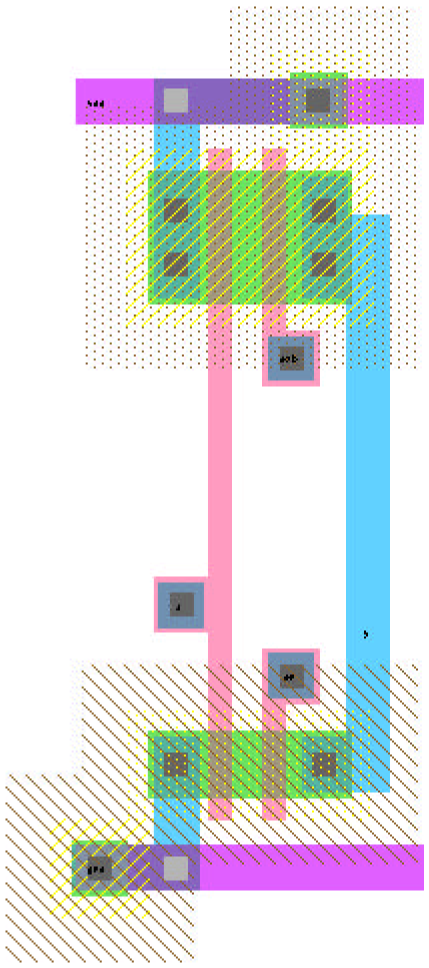
Selector Schematic



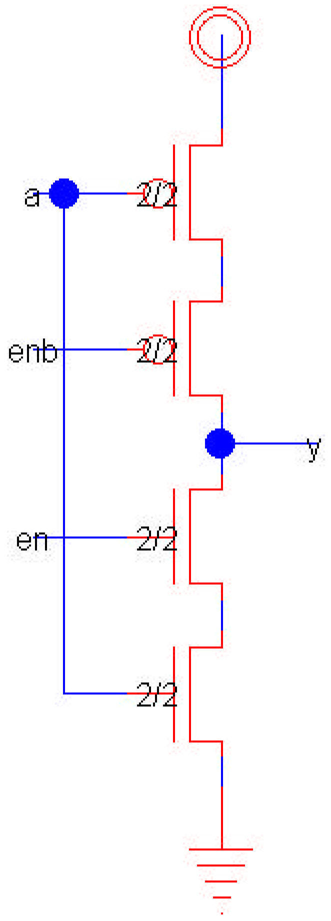
MUX Layout



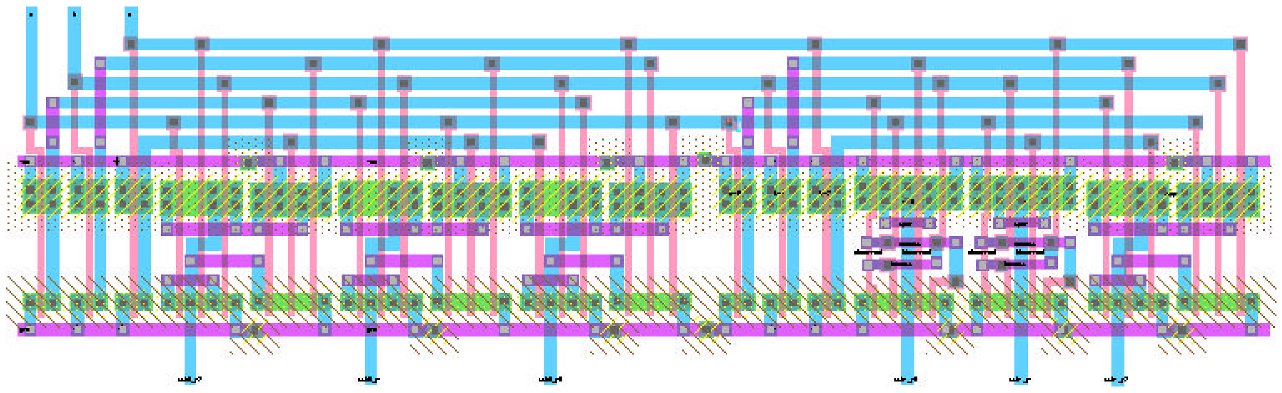
MUX Schematic

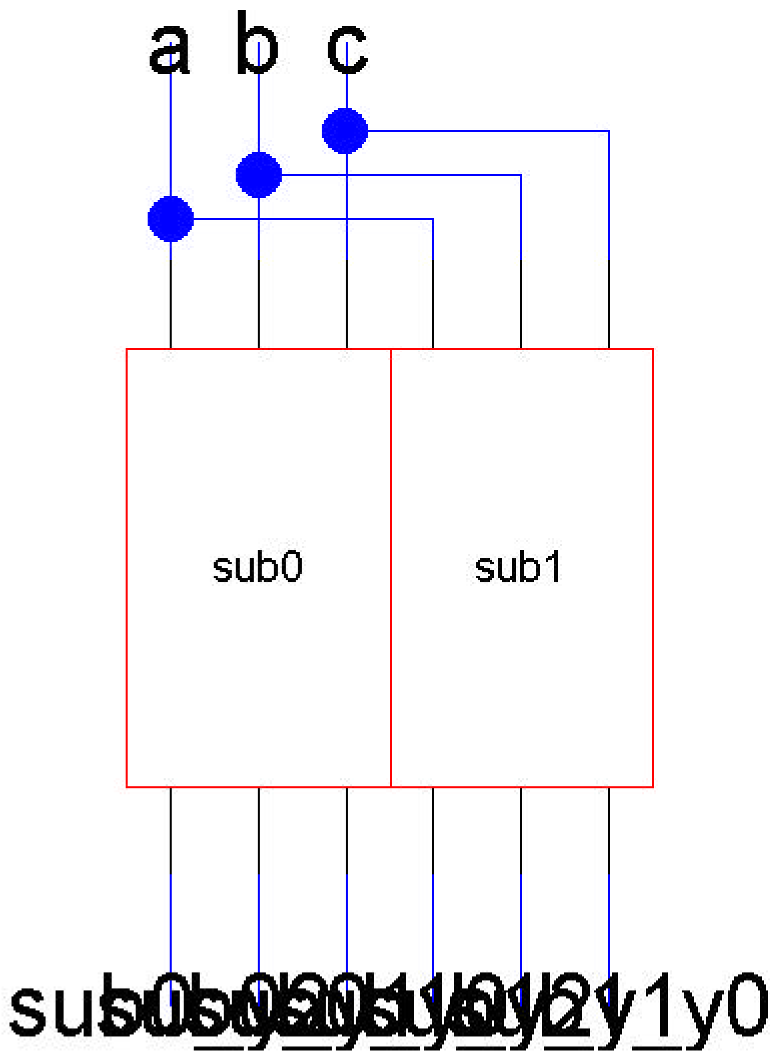


Tri-state Layout

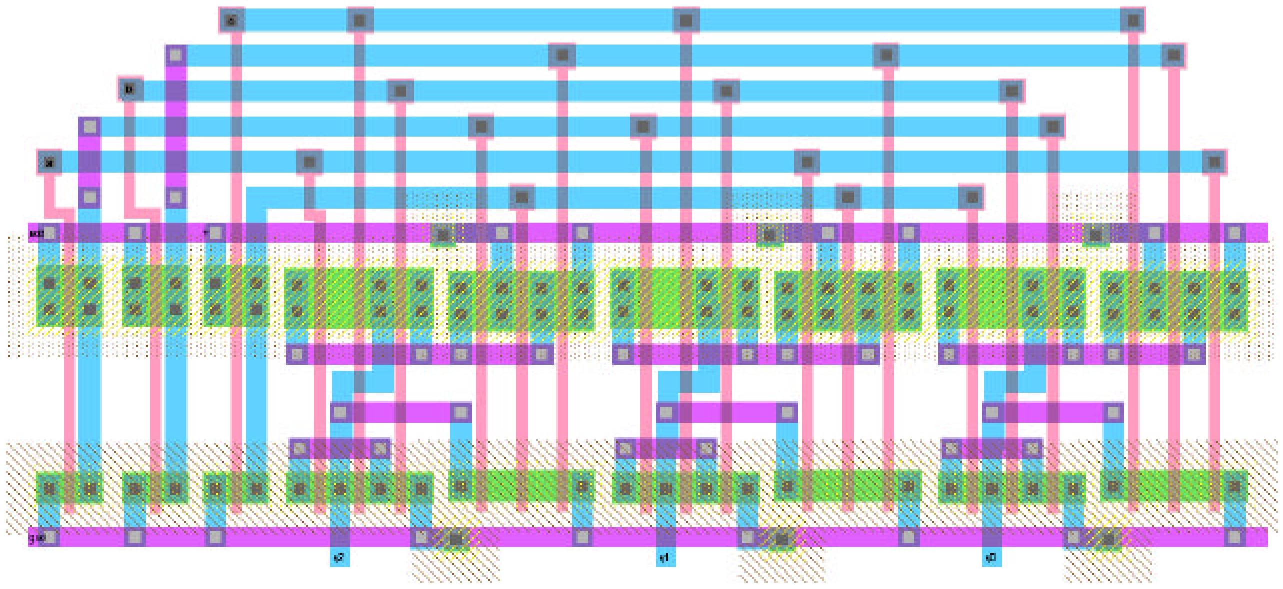


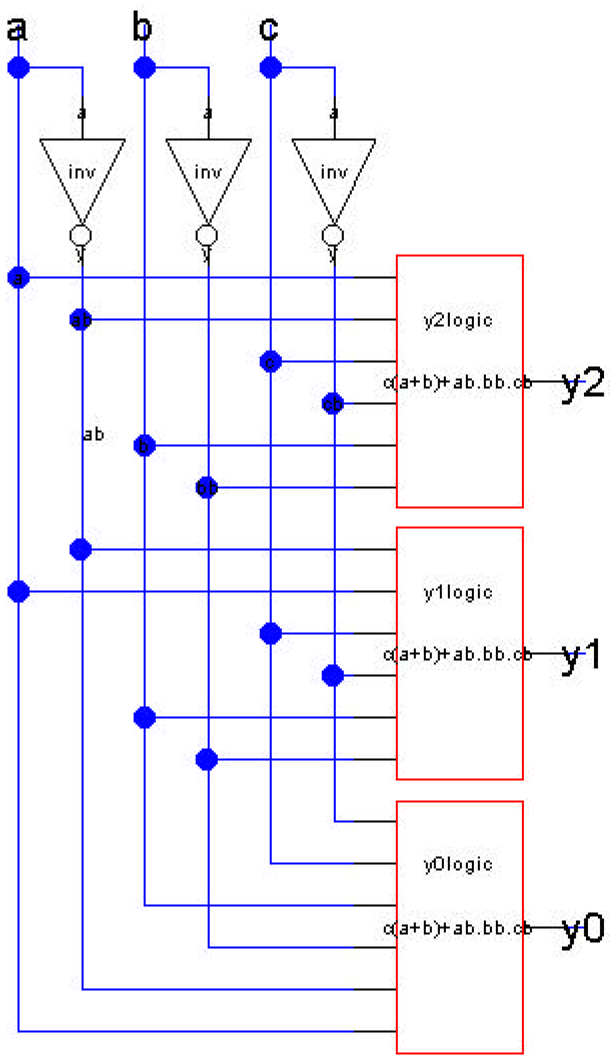
Tri-state Schematic


Substitution 0 and Substitution 1 Layout

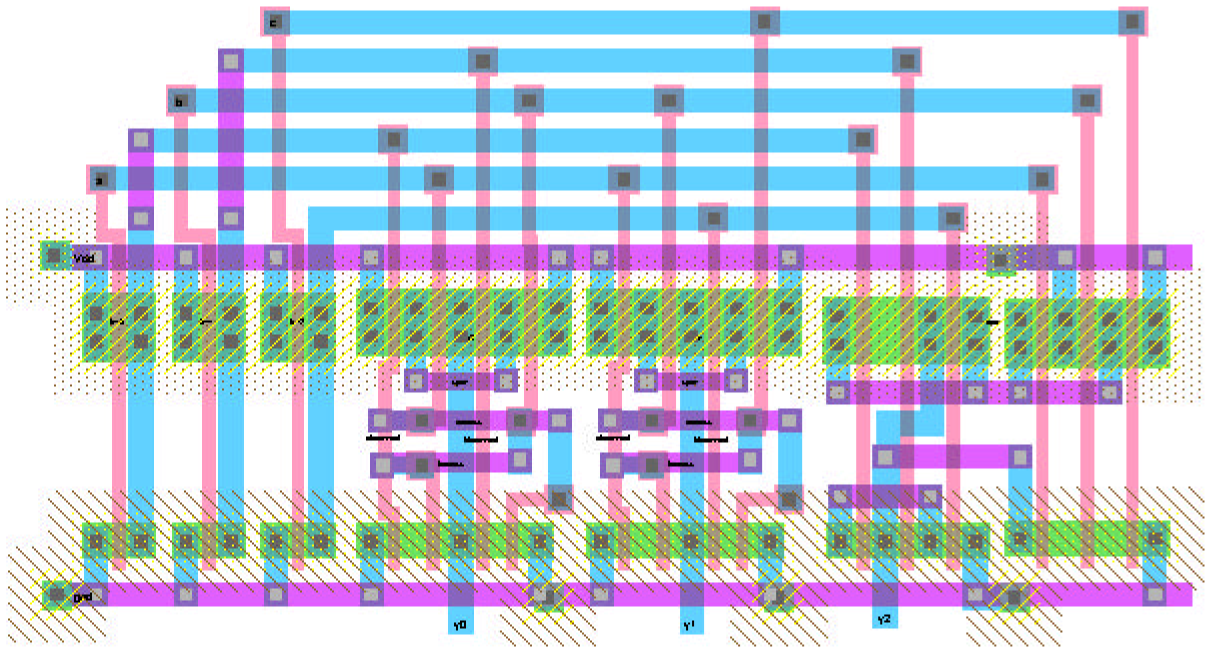


Substitution 0 and Substitution 1 Schematic

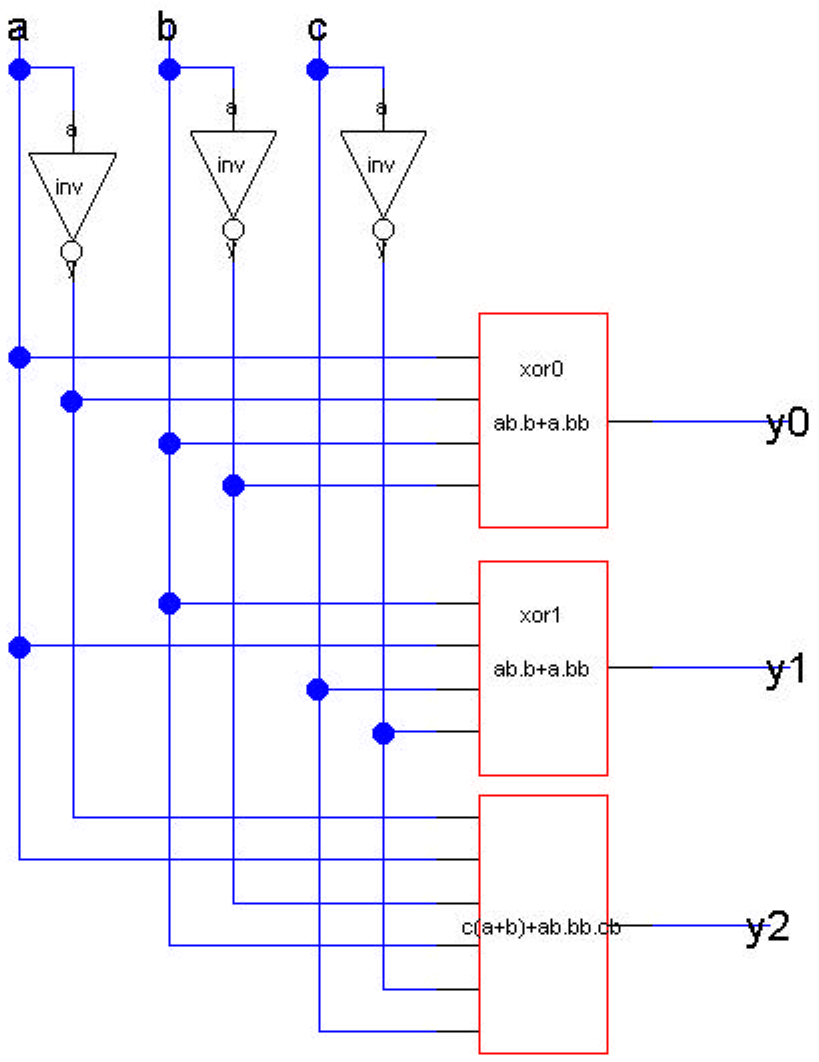

Sub 0 Layout



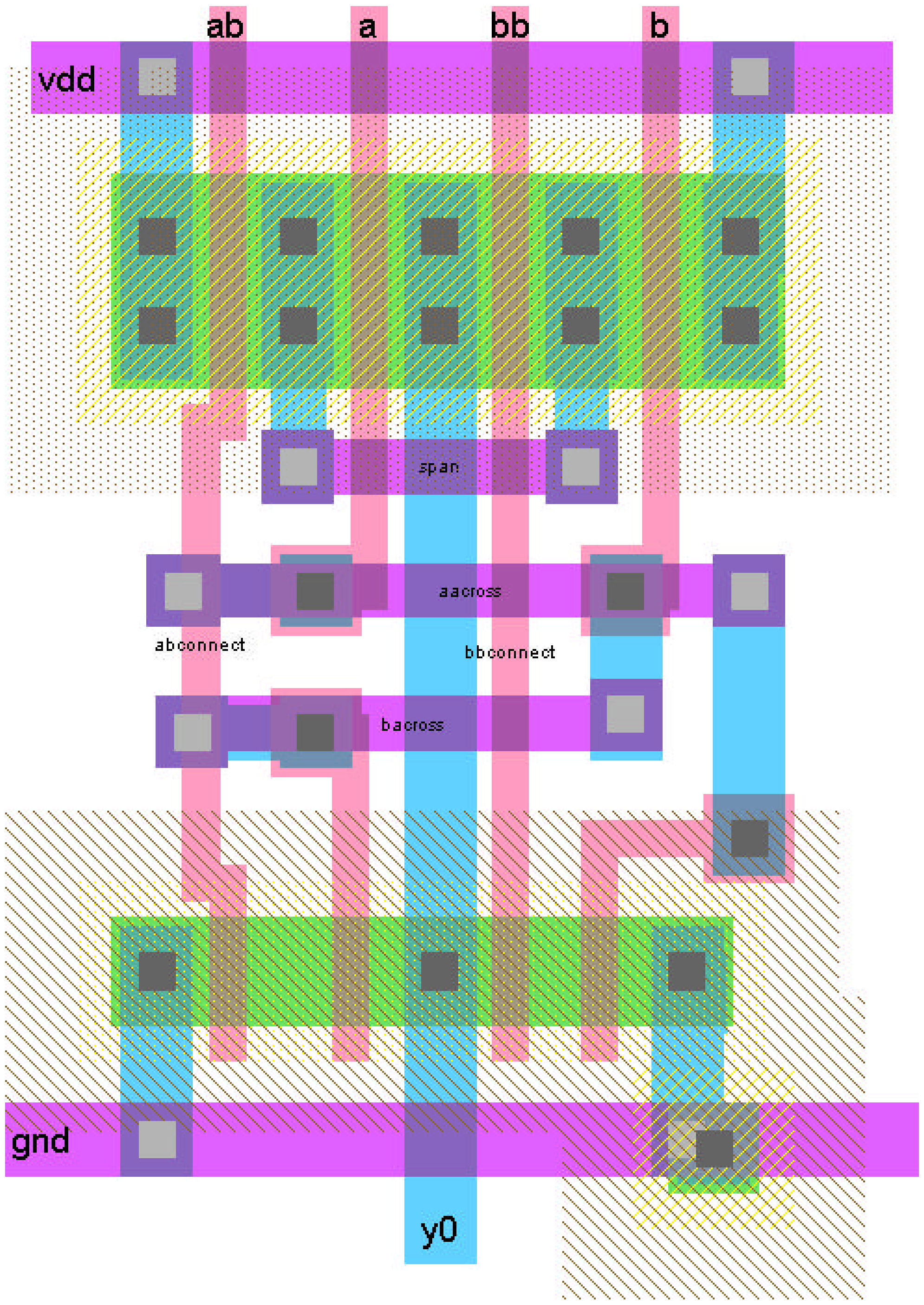
Sub Zero Schematic



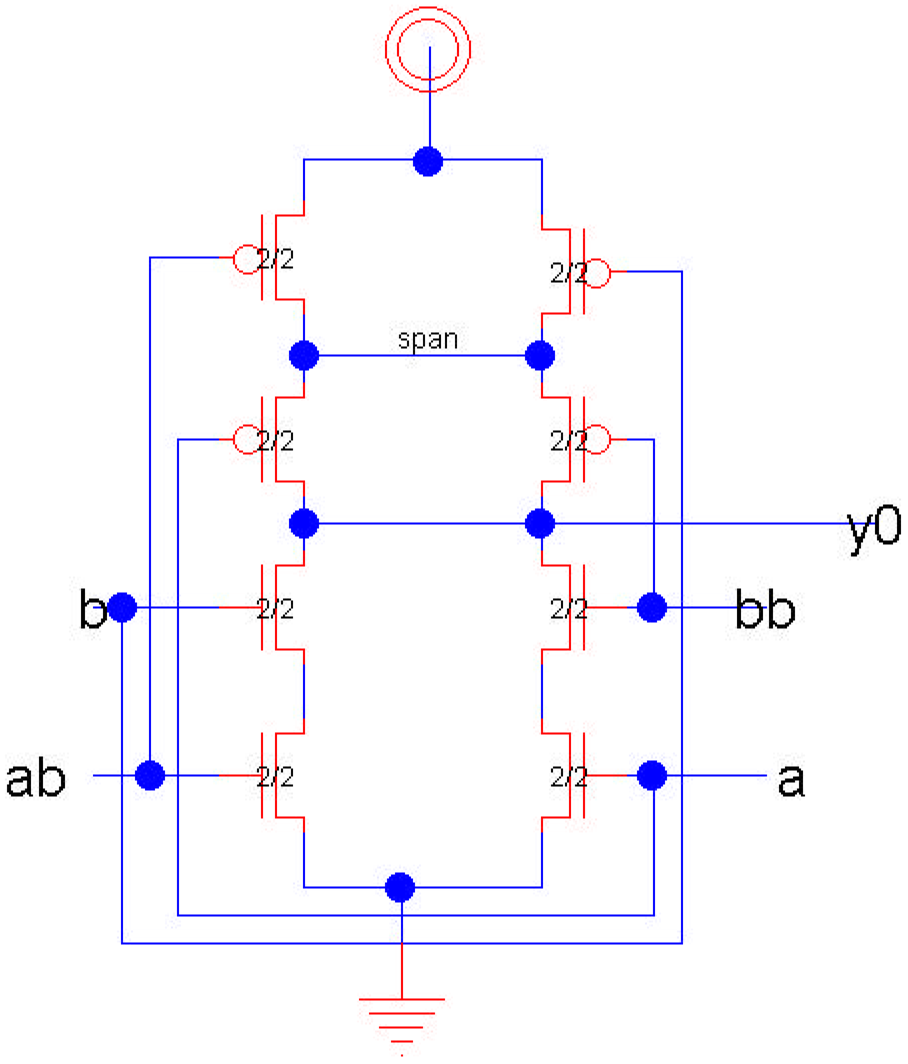
Sub 1 Layout



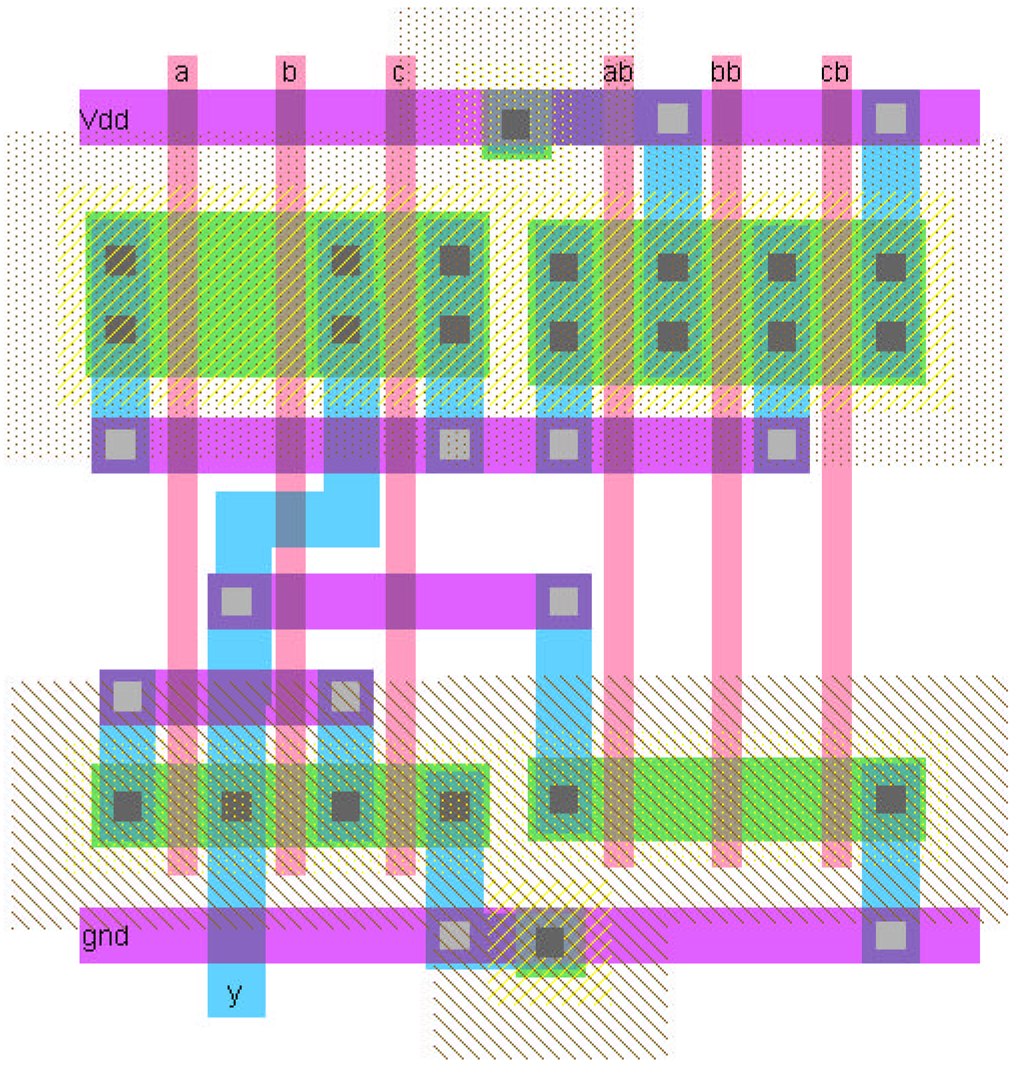
Sub 1 Schematic


ab
a
bb
b
vdd
span
aacross
abconnect
bbconnect
bacross
gnd
y0

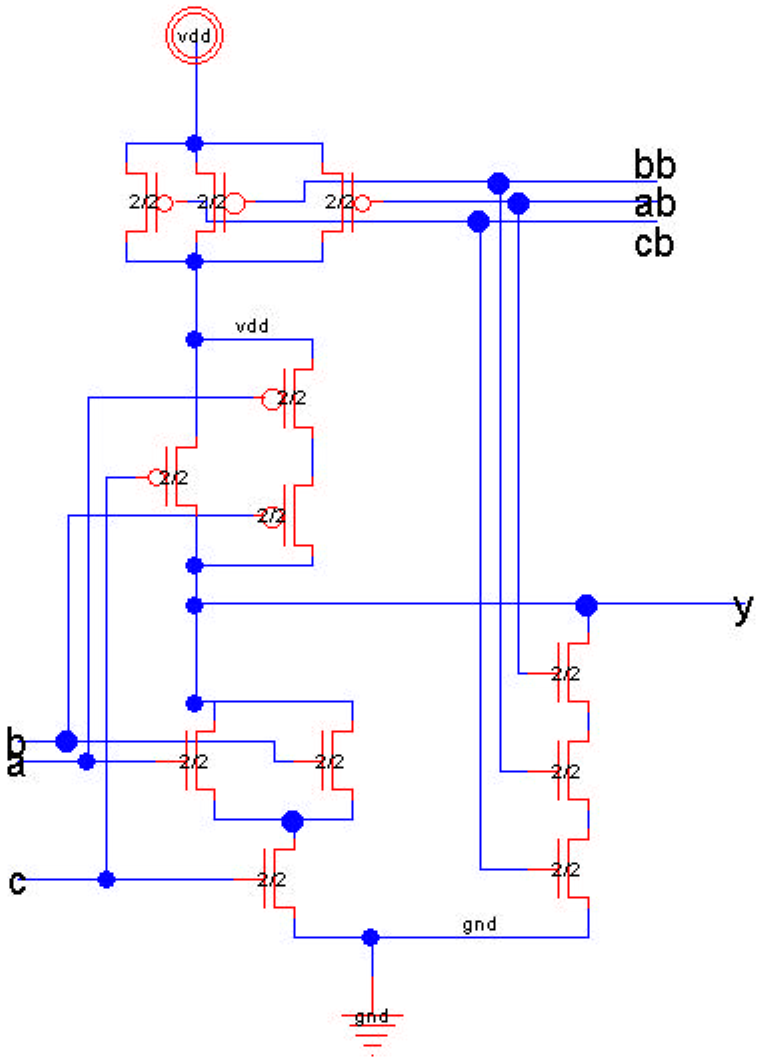
XOR Layout



XOR Schematic



Combinational Logic: a*(b+c) + ab*bb*cb Layout



Combinational Logic: a*(b+c) + ab*bb*cb Schematic