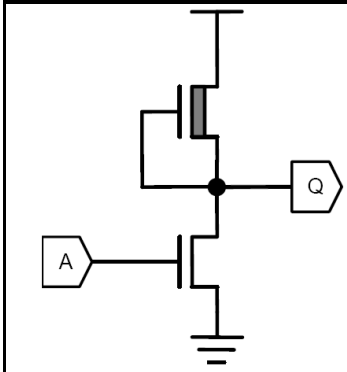
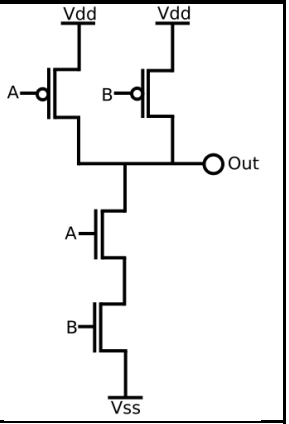
Transistors

A small current flowing from the base to the emitter of a npn transistor induces a large current from the collector to the emitter. This is used in TTL logic.

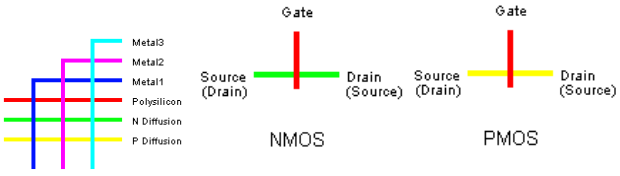
When the gate of a nMOS FET is positive with respect to the source, current is conducted from drain to source. The switched-on resistance of a pMOS FET is 3 times higher than that of the nMOS.

The threshold voltage of an nMOS FET can be made negative to give a depletion mode transistor which always conducts.

Logics

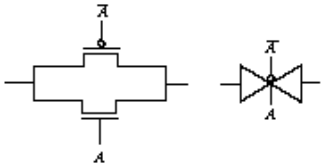
NMOS	CMOS
	
Don't get a full fall	Fast switching
Current always flows	Low power usage
NOR better than NAND	NAND better than NOR

Layout

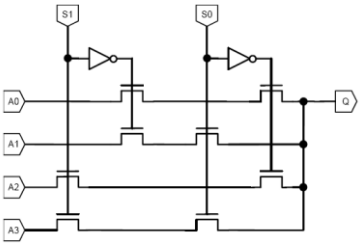


In general, CMOS gate layouts are done by using p-type pull-ups for the situations when the output is 1 and n-type pull downs from DeMorgan's laws.

Transmission gates conduct logic signals in either direction when they are switched on:

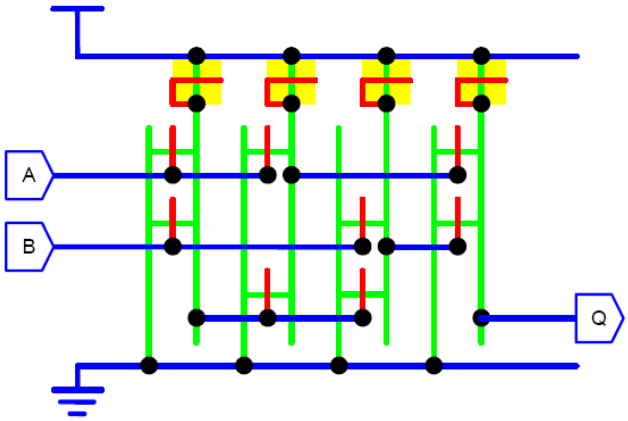


Pass transistors can be used as well, e.g. in a multiplexer however they may not be used to drive further pass transistors since the maximum source voltage of a transistor is that of the gate – threshold voltage.

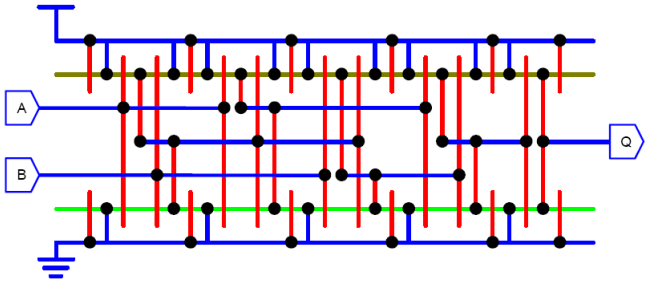


Design

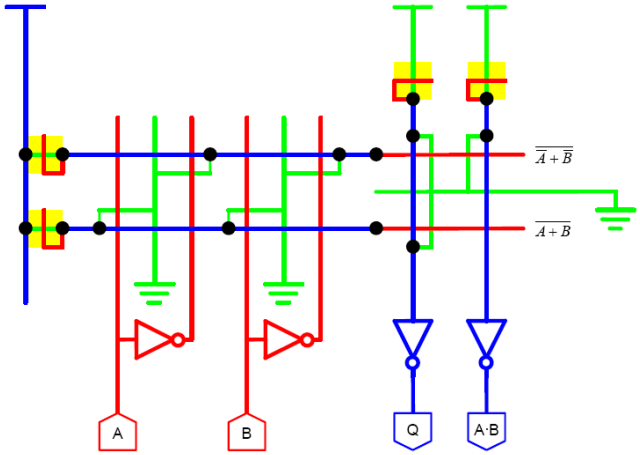
Weinberger Gate Array (NMOS):



Gate Matrix (CMOS):

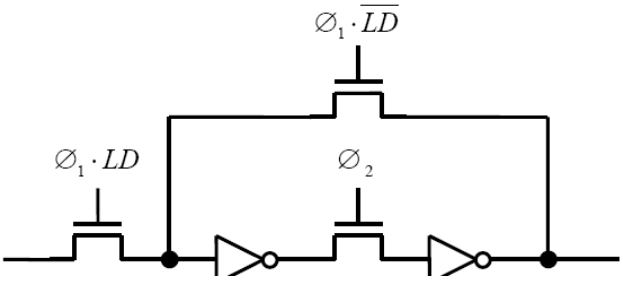


PLA (AND, OR and INV planes):

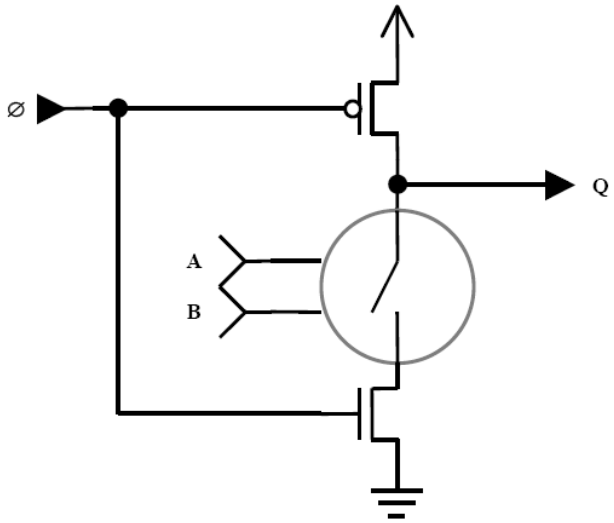


Dynamic Logic

Clocked latch:

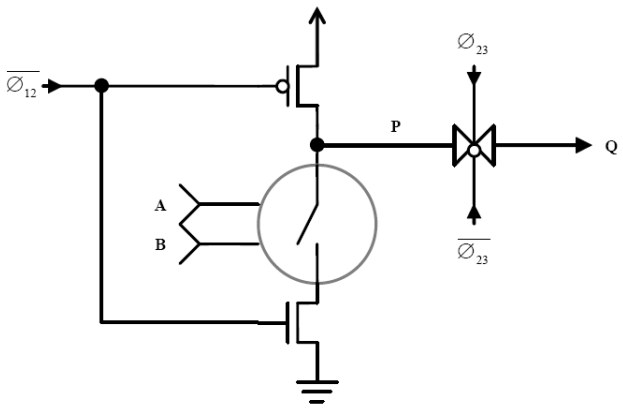


Precharging:



Such blocks cannot be concatenated because a transient high output may discharge the output of the next stage (an internal race).

In a four-phase clock system, have 4 kinds of gates. A gate is a type *n* gate if it samples its inputs during phase *n*. Its outputs are held stable during the next two phases, and hence may drive gates of type *n + 1* and *n + 2*. Clock distribution is a problem in this scheme. An example of a type 3 gate is:



In Domino logic, an inverter is added to the output of the precharged gate. Now, when the pulldown network is evaluated only a rising edge is possible and hence spurious discharges are impossible. Unfortunately only non-inverting logic is possible here.

NORA (no-race) logic works by having both n-type and p-type dynamic blocks, and you are only allowed to use them alternately. You may connect two blocks of the same kind together using an inverter as in domino, however. This is slow due to the use of slow p-type pull-up networks.

Pseudo-NMOS

This uses a passive pull-up consisting of a p-type transistor with its gate tied low.

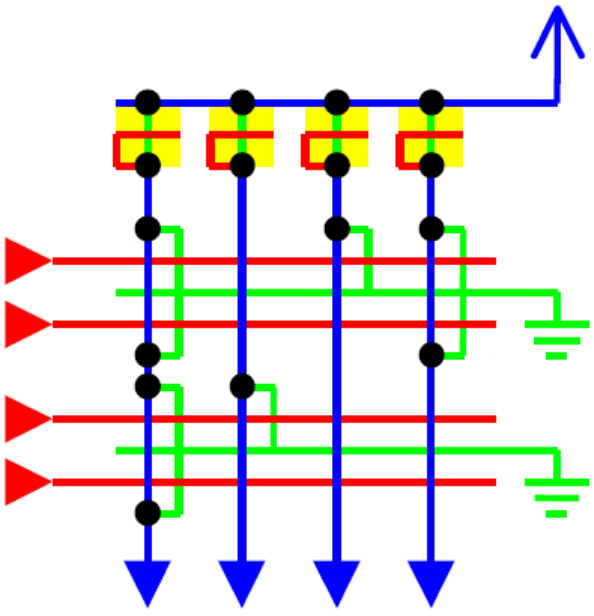
Deriving the widths of the pull up and pull down transistor networks can be tricky. First, use potential dividers to ensure the output is pulled low to 1V, i.e. $\frac{u}{d} = \frac{\vartheta}{4}$. Then model the pull-down resistance by a resistance of 1 (the inverter FET) and the pull-up resistance in parallel. Use that constraint on *d* to obtain a value for it and hence find *u*.

There is no way to make rise and fall times identical.

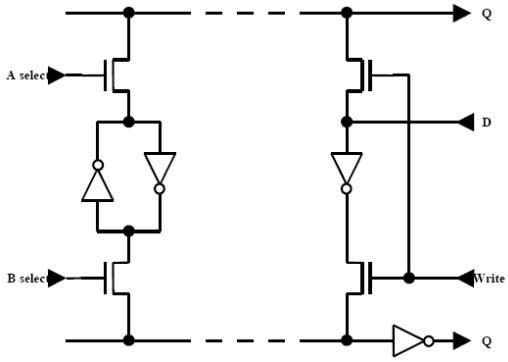
Velocity subtraction means that resistance of transistors in series is less than additive, but there are a maximum number of transistors that can be chained in this way.

Memory

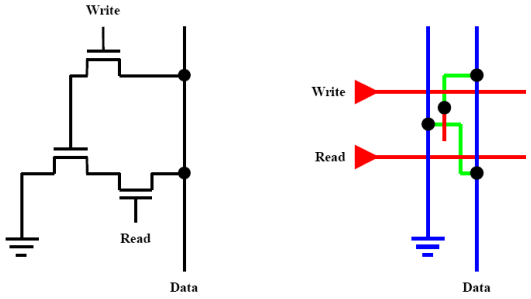
ROM:



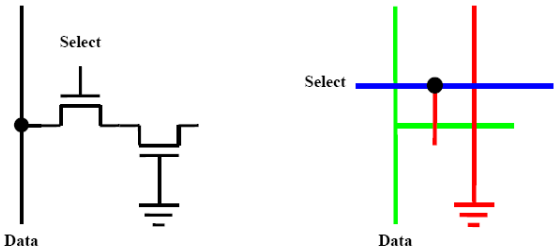
SRAM:



DRAM (3-transistor):

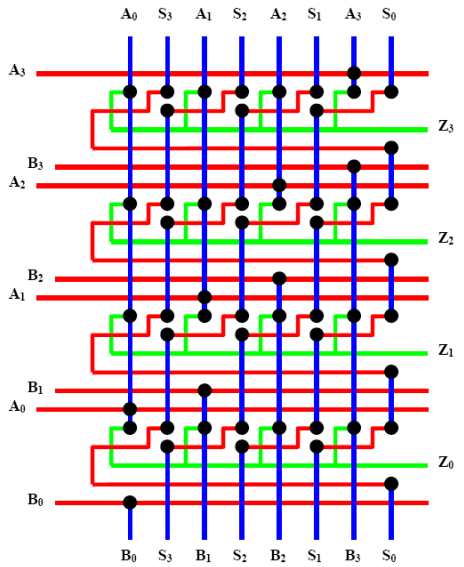


DRAM (2-transistor):



Barrel Shifter

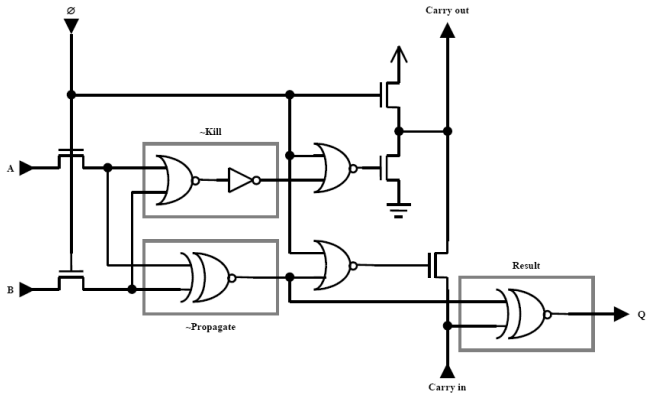
An *n*-bit barrel shifter concatenates two *n*-bit words, shifts them a given number of places and emits the bottom *n* bits.



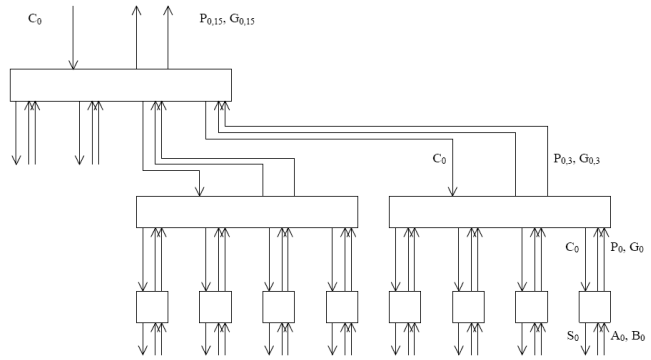
Carry In Arithmetic

Ripple carry is the basic scheme where carry propagates linearly through the output.

Manchester carry is a fast dynamic ripple carry scheme which relies on pre-computing propagate and kill to allow fast computation as carry in arrives:



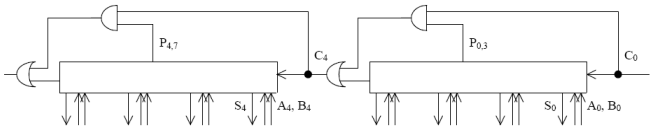
Carry look-ahead observes that since $C_{i+1} = G_i + P_i.C_i$, $G_i = A_i.B_i$, $P_i = A_i + B_i$ then multiple stages of carry and generate can be grouped together by recursive expansion into blocks of, say, 4. Now a tree structure of identical blocks can be built up to do the carry calculation. This is $O(logn)$ in time and $O(n)$ in space:



Carry select divides an *n*-bit word into blocks again, and then does the calculation of each block with both possible carry in. When the actual carry in arrives it is used to select the results from one of the blocks using a multiplexer. Since total delay for *k*-bit blocks is $T = \frac{n}{k} + k$ it can be shown that optimally $k = \sqrt{n}$ and hence time is $O(\sqrt{n})$ with linear space.

Carry skip is based on the fact that it is easier to compute propagate signals than generate signals. The delay for ripple carry out of the lowest block is *k*, and it carries through the middle blocks in $\frac{n}{k} - 2$. Finally the most significant bit arrives after a further *k* and hence total delay is $T = 2k + \frac{n}{k} - 2$, which can be

minimized if $k = \sqrt{\frac{n}{2}}$, again giving time in $O(\sqrt{n})$ with linear space. However, by varying the length of blocks to match the time before the carry in is available and the time it takes *G* and *P* to be calculated.



Fabrication

1. Coat the wafer with resist
2. Expose resist to UV light via the mask
3. Develop the resist, remove exposed material
4. Process unprotected areas with chemicals
5. Strip the remaining resist

The problem of CMOS fabrication is the formation of parasitic bipolar transistors between the p-wells and n-substrate.

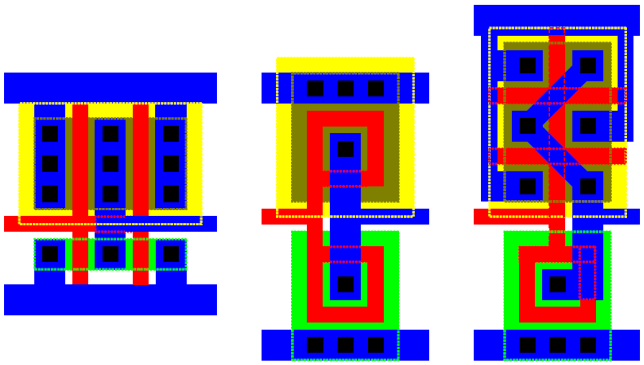
Design rules specify the minimum size of features, separations and overlaps. The constant λ is the maximum amount by which any mask may be displaced.

Design Feature	Minimum Width (λs)
Metal width	3
Metal separation	3
Polysilicon width	2
Polysilicon separation	3
Diffusion width	3
Diffusion separation	3
Well extension around diffusion	6
Spacing to active of opposite type	4
Polysilicon separation from diffusion	1
Transistor length	2
Polysilicon continuing past transistor	2
Diffusion continuing around transistor	3
Contact dimensions	2 (exactly)
Contact material surrounding	1
Contact separation	3
Contact separation from transistor	2

Capacitive Loads

These can be driven by using a chain of inverters. The *n*th inverter in the chain will offer a load of $f^{n-1}C_g$ where f is the scaling factor. If the delay to charge a gate output is t then each inverter has a delay of ft and hence *f* should be *e* to minimize total delay.

Suitable inverters for this job include these:



Logical Effort

All delays are scaled relative to that of a minimal inverter driving another with no parasitic capacitance, which is called τ. Total delay is $d_{abs} = d\tau$, $d = f + p$, $f = gh$, *g* is the ratio of logics RC product to that of a minimal inverter and *h* is the load being driven as a ratio of the capacitance of that logic to that of an input. Finally, p is proportional to the sum of the widths of transistors connected to the logic output, where the width of the minimal inverter is $\gamma + 1$ and its parasitic delay, $p_{inv} \approx 1$.

In a path, there is branching effort at each stage, so that $b = \frac{C_{total}}{C_{useful}}$. Logical, branching and electrical effort along a path is the product of the individual efforts along the path. The path effort *F* is just the product of these three efforts again, and finally total delay $D = \sum_i f_i + p_i$. This is minimized when path effort is spread equally across the stages.

Self-Timed Circuits

These get around some problems of clocked circuits such as skew, power consumption, limits to speed of the worst case delay, non-compositionality and electromagnetic interference.

Dual Rail Code		Meaning
Q_0	Q_1	
0	**0**	Clear
0	**1**	Logical 0
1	**0**	Logical 1

Inversion can be achieved by swapping wires over, and the fact that signals start at 0 means that outputs are hazard-free.

1-Of-4 Code				Meaning
Q_0	Q_1	Q_2	Q_3	
0	**0**	**0**	**0**	Clear
1	**0**	**0**	**0**	Logical 0
0	**1**	**0**	**0**	Logical 1
0	**0**	**1**	**0**	Logical 2
0	**0**	**0**	**1**	Logical 3

This consumes less power than the dual rail code since it only uses one transition per bit on average, compared to two transitions a bit for dual rail.

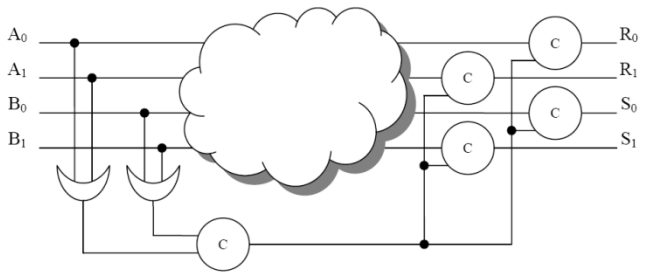
Completion detection is done by ORing together wire pairs and the using a tree of C-elements, which have the following truth table are typically implemented as AND gates with weak feedback.

A	B	C
0	**0**	0
0	**1**	C
1	**0**	C
1	**1**	1

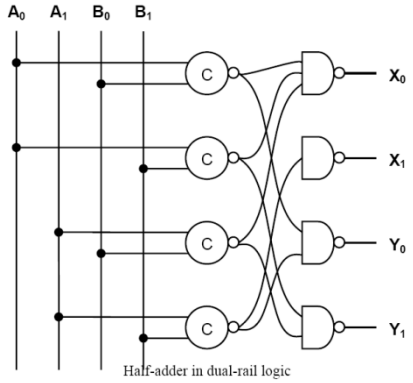
Seitz's weak conditions:

1. Some input defines before any output defines
2. All inputs define before all outputs define
3. All outputs define before any input undefines
4. Some input undefines before any output does
5. All inputs undefine before all outputs do
6. All outputs undefine before any input defines

Combinatorial envelope:

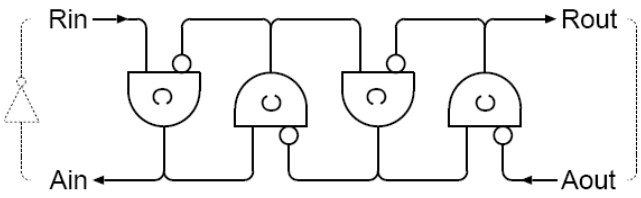


Minterm expansion:



Half-adder in dual-rail logic

In an event (transition) FIFO, each C-element takes on a value from its left only when it disagrees with the one on its right, and only ends up using every 2nd element:



Micropipelines interpose event-controlled latches throughout combinatorial logic: