VLSI Design

UNIVERSITY OF
CAMBRIDGE

Computer Laboratory

Computer Science Tripos Part 2

Peter Robinson

Michaelmas 2001

William Gates Building
JJ Thomson Avenue
Cambridge
CB3 0FD

http://www.cl.cam.ac.uk/

Introduction

This course will introduce the design of very large scale integrated circuits. The material develops an understanding of the whole spectrum from semiconductor physics through transistor-level design and system design to architecture, and promotes the associated tools for computer aided design.

Syllabus

The course consists of 12 lectures divided into three main headings:

Transistor design

- Simple logic. MOS layers, stick diagrams. Layout of an invertor. Transmission gates and pass transistor logic.
- Combinatorial logic. NOR and NAND in nMOS and CMOS. Compound gates. Delays.
- Logic design. Stereotyped design and PLAs.

System design

- Clocking and registers. Storage elements and sequential machines. Dynamic logic.
- Memory design.
- Building blocks. Shifters, adders, ALUs.

Computer-aided design

- Fabrication. Design rules and lambda rules. Performance and large loads. Scaling.
- Semi-custom techniques. Gate arrays, standard cell, full custom.
- Self-timed circuits.

Objectives

On completing the course, students should be able to:

- Describe the structure and operation of an MOS transistor.
- Design simple logic in CMOS.
- Compare different designs as circuits, stick diagrams and layout.
- Explain gate matrix and PLA design in CMOS.
- Apply clocked design for dynamic logic and storage.
- Discuss different approaches to the design of memory.
- Describe the modules making up a processor.
- Explain the fabrication process and analyse its implications.
- Compare different approaches to the implementation of systems.
- Discuss the relevance and design of self-timed circuits.

It should be pointed out that these notes do not constitute a complete transcript of all the lectures and they are not a substitute for text books. They are intended to give a reasonable synopsis of

the subjects discussed, but they give neither complete descriptions nor all the background material.

Appropriate books

The following books are relevant for the course:

- S Augarten: *State of the art*, Ticknor & Fields 1983.
  A pictorial history of semiconductors, just right for your coffee table.
- GM Blair: *MOS circuit design*, Chartwell-Bratt 1992.
- SB Furber: *VLSI RISC architecture and organisation*, Marcel Dekker 1989.
  Details of the ARM processor.
- SH Gerez: *Algorithms for VLSI design automation*, Wiley 1999.
- J Mavor, MA Jack & P Denyer: *Introduction to MOS LSI design*, Addison-Wesley 1983.
  A gentle introduction, beginning to show its age.
- C Mead & L Conway: *Introduction to VLSI systems*, Addison-Wesley 1980.
  The old classic, emphasis on nMOS.
- NHE Weste & K Eshragian: *Principles of CMOS VLSI design* (2nd edition), Addison-Wesley 1993.
  The new classic?
- W Wolf: *Modern VLSI design - a system approach*, Prentice-Hall 1994.

Colouring conventions

You will need a set of colouring pencils (including yellow, green, brown, red and blue) to gloss the diagrams in these notes during the lectures.

The following conventions are used for the colours here:

Yellow Green Brown Red Blue Black

A copy of these notes (and other relevant teaching material) can be read on-line by following the links from http://www.cl.cam.ac.uk/Teaching/. This may be particularly helpful when checking colours. Access is limited to computers within the Computer Laboratory.

Semiconductor technology

Semiconductors can be made from crystalline silicon into which impurities have been introduced:

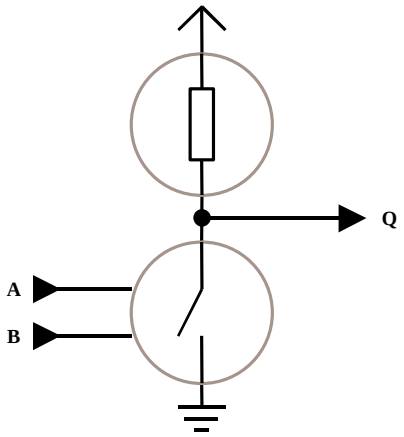
- A high valency implant such as phosphorous gives free electrons, creating an *n-type* region.
- A low valency implant such as boron gives free holes, creating a *p-type* region.

The junction of an n-type and a p-type region in a single crystalline lattice creates a diode which only conducts if it is *forward biased* with the p-type region (the *anode*) more positive than the n-type region (the *cathode*).

A light emitting diode has the additional property that it glows when current is flowing through it. It is prudent to limit this current to a few milli-Amps by means of a kilohm series resistor, or it glows very brightly, but only for a short time.

Digital switching

Most digital logic is based on the idea of switching signals between a high voltage (which we will usually treat as being 5V, although modern systems more commonly use 3.3V or less) and a low voltage (0V, or ground). The sense may be determined by current flowing or not (as in bipolar circuits) or by the presence or absence of charge (as in MOS circuits). A logic function takes some input signals and computes an output function using pull-up and pull-down circuits which may be passive (always switched on) or active (selectively switched).



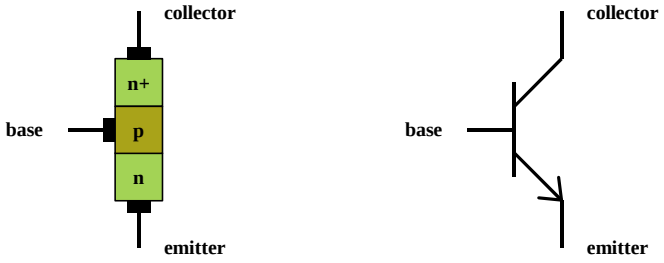
Passive pull-up and active pull-down

The diagram shows a circuit with an active pull-down and a passive pull-up. The pull-down can be thought of as a remote-control switch, usually made from transistors but possibly relays or valves.

A further complexity with MOS circuits is that the charge on wires persists after they have ceased to be driven; this means that the wires have a memory (typically lasting a thousandth of a second or so) of the last value driven on them.

Bipolar circuits

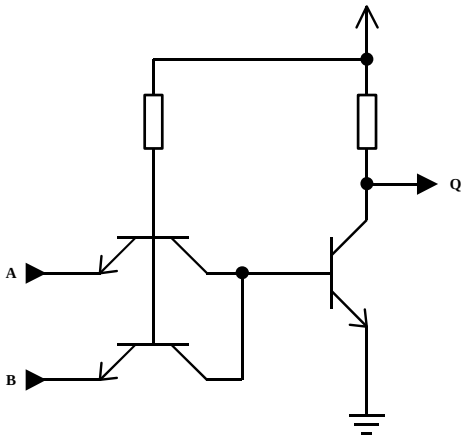
A *bipolar* transistor is formed by a sandwich of n-type, p-type and n-type regions in a single crystalline lattice. It can be thought of two diodes connected anode-to-anode such that a current through the forward biased diode overwhelms the reverse biased diode.



npn bipolar transistor

A small current flowing from the base to the emitter of an npn transistor induces a large current from the collector to the emitter. A pnp transistor has the opposite polarity.

These can be used to construct a NAND gate using transistor-transistor logic (TTL).

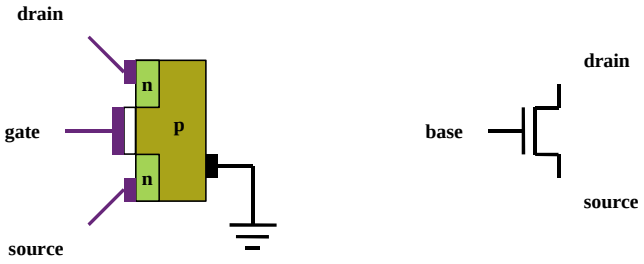


TTL NAND gate

The two transistors on the left calculate the logical function and that on the right is simply an invertor. Sometimes this is followed by an additional buffer for a totem pole output. With a 1k7 Ohm pull-up, about 1 mA flows through the gate whenever an input is high and the gate then dissipates 5 mW. There are also difficulties in finding the right sizes for the transistors and resistor.

MOS circuits

An *enhancement mode, n-channel, metal-oxide-silicon field-effect transistor* (nMOS FET) is formed on a crystal of p-type silicon. Two n-type regions (known as *diffusion*) lie on either side of a region of the p-type substrate which is covered by a thick layer of insulating silicon dioxide (or *oxide*) and a metal plate.

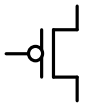


n-channel enhancement mode metal-oxide-semiconductor field-effect transistor

When the gate is positive with respect to the source, an n-type *channel* is formed under the gate and current is conducted from drain to source. Even when turned on, a MOS transistor has a resistance of about 10 kΩ.

The construction of the transistor is symmetric with respect to the source and drain - the labels merely indicate the relative voltages. This contrasts with the different processing used to make the collector and emitter of a bipolar transistor.

A *p-channel* MOS FET has the opposite polarity and conducts when its gate is low. However, the resistance of a p-type channel is about 2½ times that of an n-type channel of the same size.



In integrated circuits, the metal gate is replaced by one made from polycrystalline silicon (or *polysilicon*) for ease of fabrication.

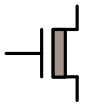
The nMOS transistor operate in three modes:

- *off* when $V_{gs} < V_t$
- *saturated* when $V_{gs} > V_t$ and $V_{ds} > V_{gs} - V_t$
- *linear* when $V_{gs} > V_t$ and $V_{ds} < V_{gs} - V_t$

where Vt is the threshold voltage (= 0.2 V_{dd} = 1V for a 5V system)

Note that, even when the transistor is turned on, the source voltage can not rise above the gate voltage less the threshold voltage.

The threshold voltage can be adjusted by implanting further impurities into the channel regions. It can even be made negative (V_t = -0.8 V_{dd} = -4V), giving a *depletion mode* nMOS FET which always conducts. This can be used as a compact way of making a resistor.

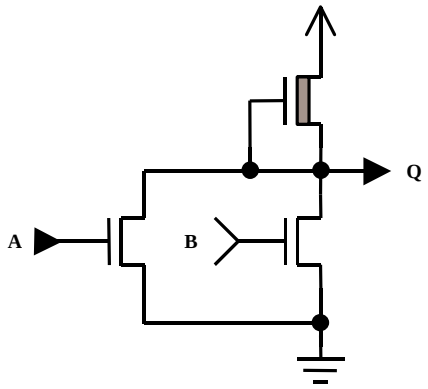


nMOS

An nMOS NOR gate can be made with two n-type pull-down transistors in parallel and a passive pull-up. There are three ways that the pull-up could be made:

- A resistor – using polysilicon (which is the most resistive material available in a MOS process) this would have to be several hundred times the size of the pull-down transistor.
- An enhancement mode transistor with its gate wired high – this could never pull the output above $V_{dd} - V_t$.

- A depletion mode transistor with its gate wired to its source is used in practice.



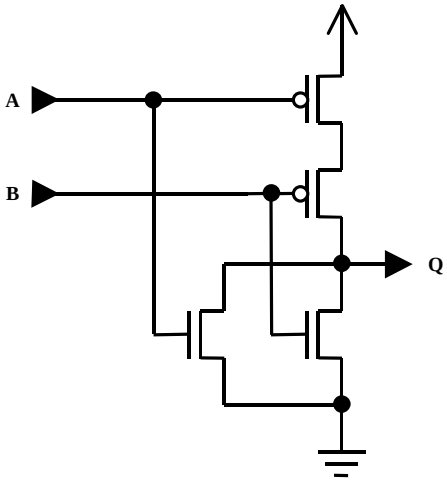
NOR gate in nMOS

Current flows mainly when the gate is switched and the output is charged or discharged; only a small leakage current flows otherwise. With a 40 kΩ pull-up and a 10 kΩ pull-down in series, a current of 0.1 mA flows when an input is high and 0.5 mW is dissipated.

When the pull-down network is switched on, the depletion mode pull-up and the enhancement mode pull-down form a potential divider, and the output voltage approaches the appropriate ratio of the supply voltage – usually the ratio is 1:4, so the output falls to 1 V.

CMOS

A CMOS NOR gate can be made with two n-type pull-down transistors in parallel and two p-type transistors in series as an active pull-up. The complementary Boolean circuits in the pull-up and pull-down networks give the technology its name.



NOR gate in CMOS

Current only flows when the gate is switched and the output signal (which may be regarded as a capacitor) is charged or discharged, making the power consumption very low.

A further advantage of CMOS over nMOS and bipolar circuitry is that is does not rely on the ratio of the resistances in the pull-up and pull-down networks to determine the output voltage. The output switches between 0 V and 5 V rather than between about 1 V and 5 V. The disadvantage is the additional complexity of the complementary circuit.

For reasons that will be discussed later, modern integrated circuits operate with supply voltages much lower than 5V and all the voltages scale accordingly.

Simple logic in MOS

There are several layers in an nMOS chip:

- a p-type substrate
- paths of n-type diffusion
- a thin layer of silicon dioxide
- paths of polycrystalline silicon
- a thick layer of silicon dioxide
- paths of metal (usually aluminium)
- a further thick layer of silicon dioxide

with contact cuts through the silicon dioxide where connections are required.

The three layers carrying paths can be considered as independent conductors that only interact where polysilicon crosses diffusion to form a transistor. These tracks can be drawn as stick diagrams with

- diffusion in green
- polysilicon in red
- metal in blue

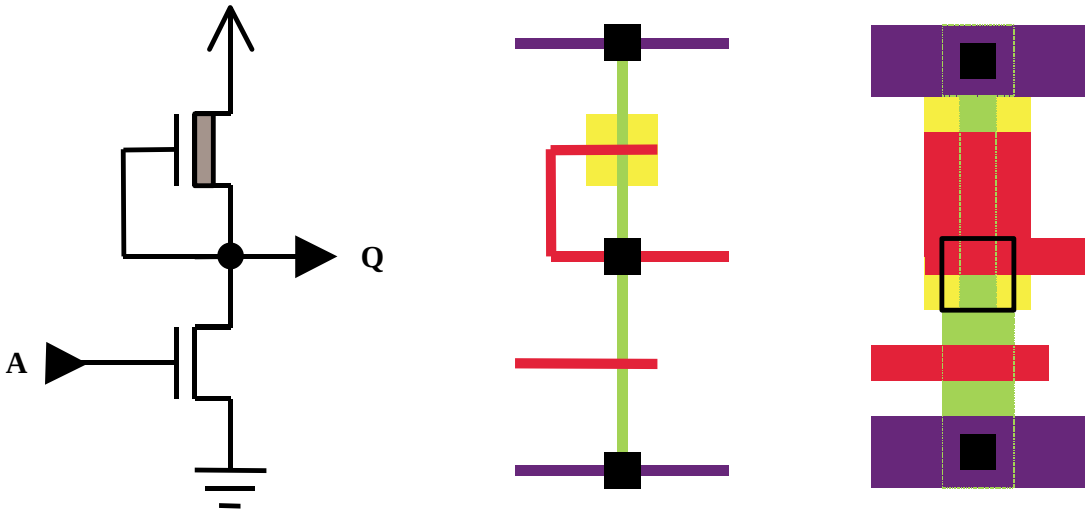
using black to indicate contacts between layers and yellow to mark regions of implant in the channels of depletion mode transistors.

With CMOS there are two types of diffusion: n-type is drawn in green and p-type in brown. These are on the same layers in the chip and must not meet. In fact, the method of fabrication required that they be kept relatively far apart.

Modern CMOS processes usually support more than one layer of metal. Two are common and three or more are often available.

Actually, these conventions for colours are not universal; in particular, industrial (rather than academic) systems tend to use red for diffusion and green for polysilicon. Moreover, a shortage of coloured pens normally means that both types of diffusion in CMOS are coloured green and the polarity indicated by drawing a circle round p-type transistors or simply inferred from the context. Colouring for multiple layers of metal are even less standard.

There are three ways that an nMOS invertor might be drawn:

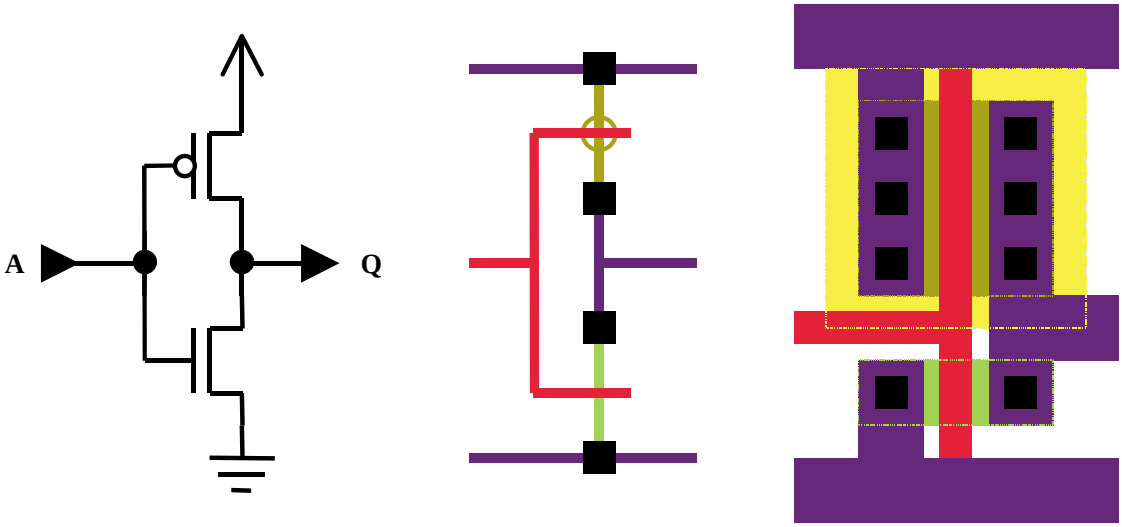


Invertor designs in nMOS

The three different representations are useful in different contexts:

- a circuit diagram – used to plan the logic of the system;
- a *stick diagram* – used to plan the topology of a layout, committing signals to particular layers; and
- layout – final decisions of sizes

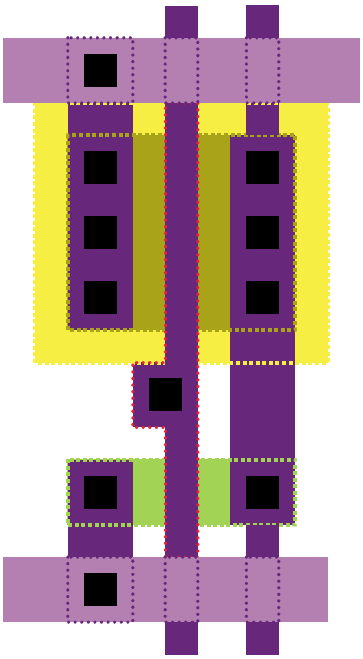
The equivalent pictures in CMOS are:



Invertor designs in CMOS

This layout shows the input arriving through polysilicon on the left and the output leaving through metal on the right. A second layer of metal might be used to allow connections above and below the invertor with a third layer left free to run other, quite separate, signals (such as a global clock) across the top of the invertor.

The following design runs power and ground in the second metal layer and signals in the first, with the polysilicon hidden underneath it.

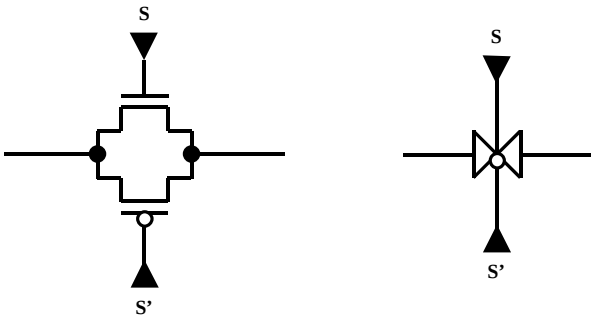


Transmission gates

It is possible to compute logic functions without making logic gates – networks of MOS transistors can be connected together directly. These are known as *transmission gates*. (Of course, the transistors still have gates, but that is a different use of the word, as also would be logic gates!).

With CMOS, the nMOS transistors are good at conducting low signals and the pMOS transistors are good at conducting high signals, so transmission gates are often made from a pair of complementary transistors.

When the control signal S is high, the transmission gate conducts logic signals of either sense in either direction. A special symbol is used for the CMOS transmission gate:



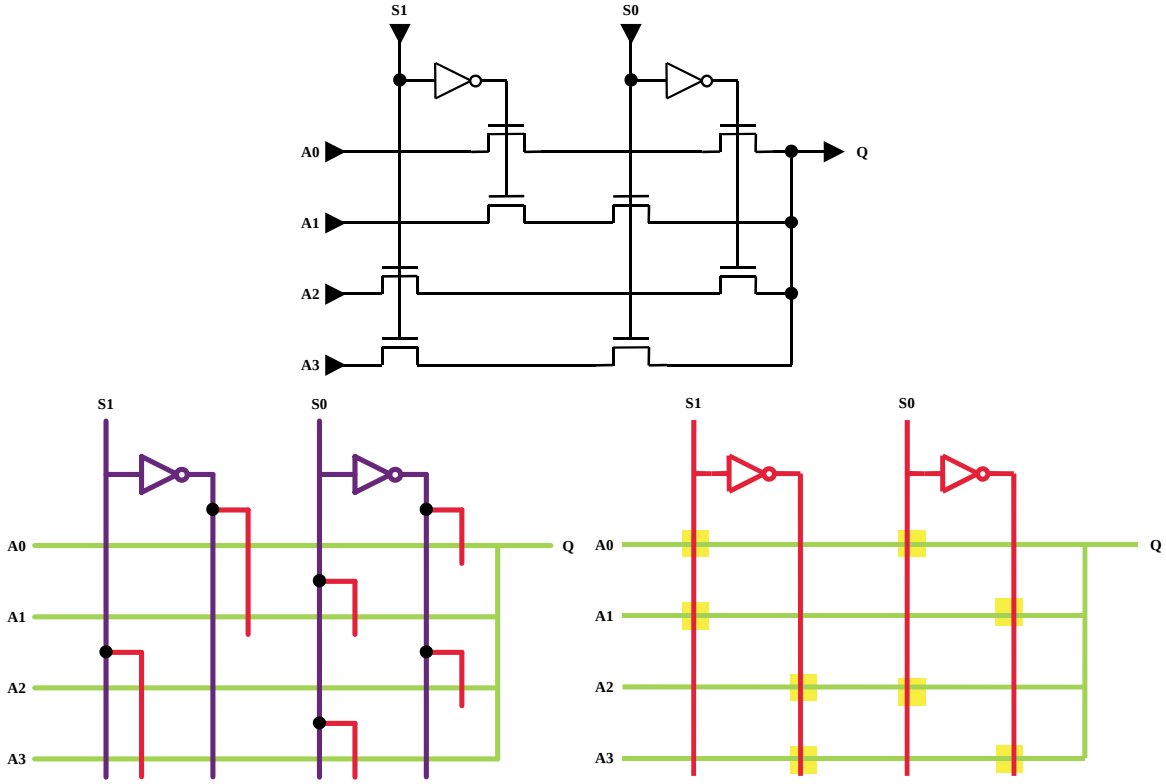
Pass transistor logic

MOS transistors can be used simply as switches to steer current using *pass transistor logic*.

Consider a simple multiplexor with 2 control inputs, S_0 & S_1, 4 data inputs, I_n, and a single output, Z. The function can be specified by a simple table:

S_1	S_0	Z
0	0	A_0
0	1	A_1
1	0	A_2
1	1	A_3

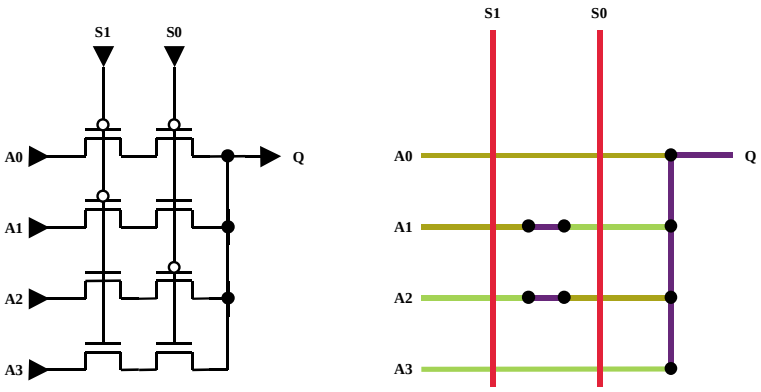
This can be implemented using nMOS pass transistors as follows:



Multiplexor using pass transistors

Note how a mixed notation (inverters and transistors or even invertors in a stick diagram) can be used. The second stick diagram uses depletion mode transistors as conductors to avoid going into metal, giving a more compact layout.

This could be implemented in CMOS even more simply, although care needs to be taken about mixing the two types of diffusion (or else a diode would be formed):

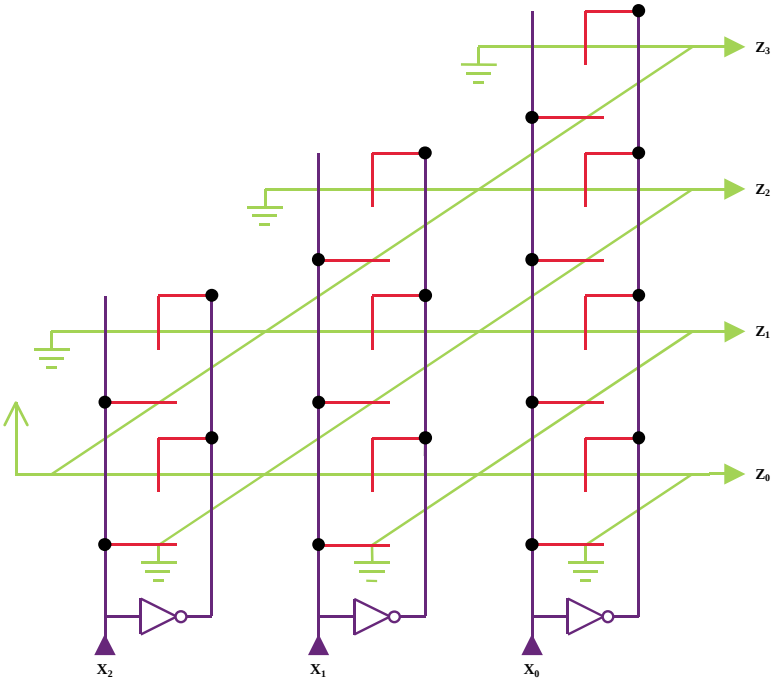


The design would also work better if transmission gates were used.

As a further example, consider a tally circuit to count the number of 1s in an input word, giving the answer in unary. The specification is:

X_2	X_1	X_0	Z_0	Z_1	Z_2	Z_3
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	1	0	0
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	0	1	0	0	1	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

This can be implemented using nMOS pass transistors as follows:



Note how each output signal in pass transistor logic is driven precisely once for any pattern of input signals – no signal should be left undriven and there should be no contention for any output value.

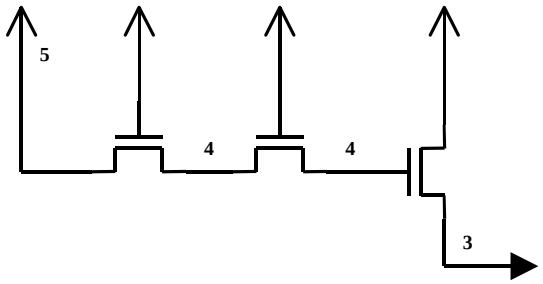
The idea of repeating a simple structure to make a complicated circuit is important in VLSI design.

A caution

The source potential of a MOS transistor can not rise above the gate potential less the threshold voltage. When using a chain of pass transistors, this results in a significant voltage drop across the first transistor, and rather less across subsequent ones.

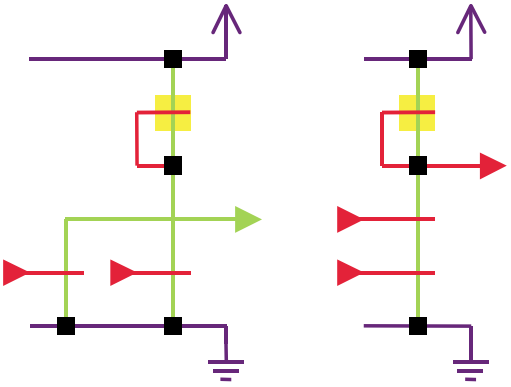
The attenuated signal must be restored by giving the gate that it drives a more sensitive (larger) pull-down transistor.

A signal that has been switched by a pass transistor must not itself control a further pass transistor, or a second voltage drop would occur:

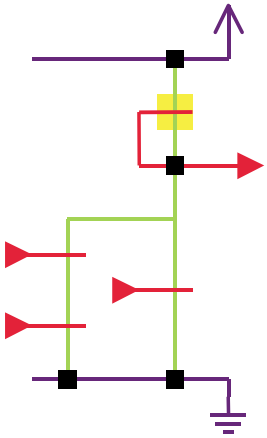


Combinatorial logic

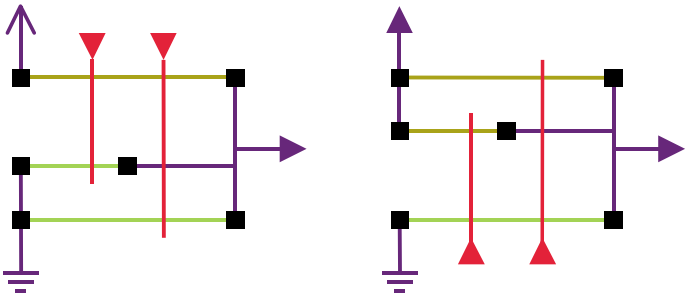
In nMOS, the NOR gate has better speed and area characteristics than the NAND gate:



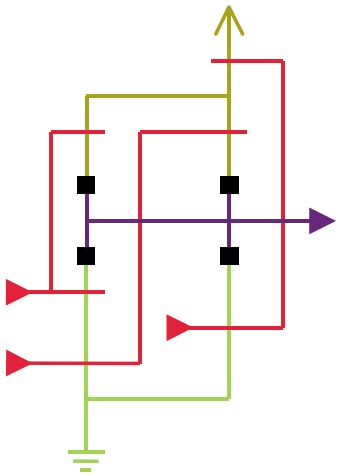
It is also possible to compute complex functions such as $\overline{X \cdot Y + Z}$ in one step:



In CMOS, the NAND gate has better speed and area characteristics than the NOR gate:



Again, complex logic is possible:



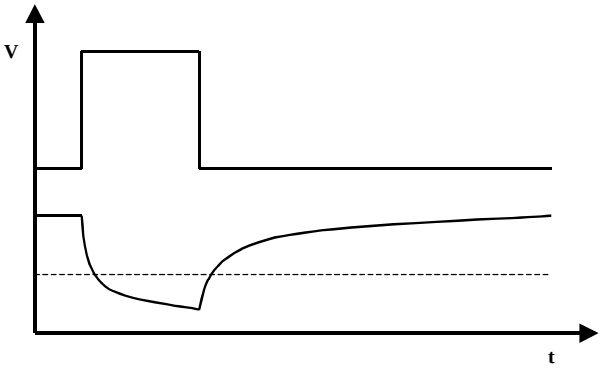
The complementary pull-up network becomes tedious and may be replaced by a p-type transistor tied low, rather like an nMOS depletion load (but this wastes power). Alternatively, dynamic (or clocked) logic may be used – see below.

Delays

The delay through a MOS gate is simply the time that it takes to charge (or discharge) its output signal above (or below) the threshold voltage of any transistors in further circuitry that it drives. The voltage on the output will move asymptotically towards its final voltage in an exponential decay whose time constant, RC, is dominated by the product of the resistance of the channel in the transistor driving the output and the capacitance of the output signal.

The series resistance can be reduced (speeding up the gate) by increasing the size of the driving transistor, allowing a trade to be made between speed and size. The capacitance is determined by the length of the output track and, significantly, by the area of the gates that it drives; this in turn depends on the fan-out and the power of the gates in the fan-out.

For an nMOS invertor, the ratio of the resistances of the pull-up and pull-down transistors determines the sensitivity of the invertor and also the shape of the curve describing the output voltage after a change of the input.



Note that an input change from 0V to 5V results in an output swing from 5V to 1V and also that there are differential delays for falling and rising outputs.

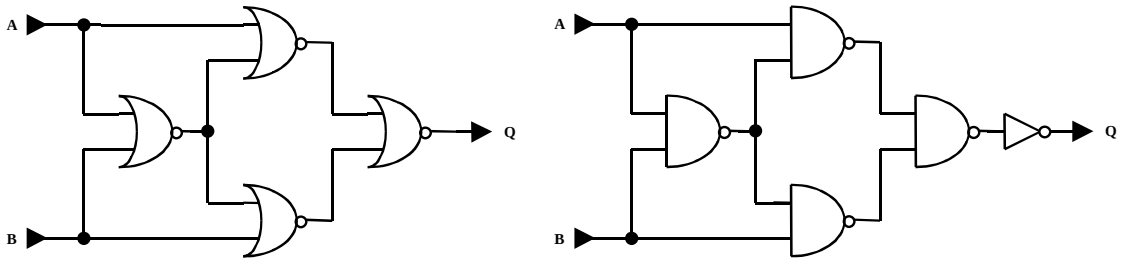
A simple CMOS invertor will also have different resistances of p-type and n-type channels. However, this can be resolved by changing the sizes of the transistors, giving a symmetric response as well as full logic swing.

Logic design

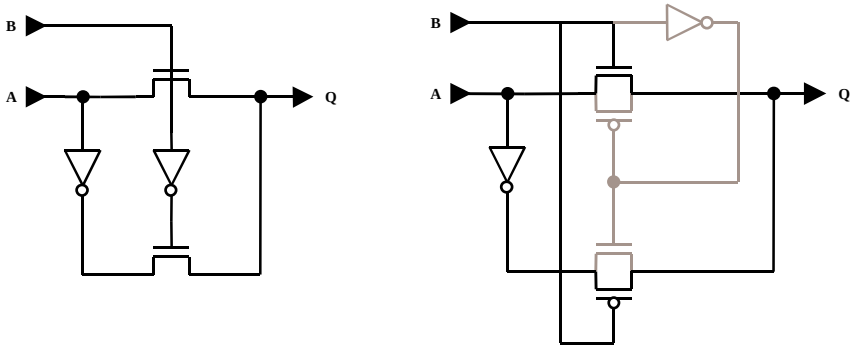
Consider the design of a circuit to compare two signals and test them for equality – an exclusive NOR gate. This has the following definition:

A	B	A=B
0	0	1
0	1	0
1	0	0
1	1	1

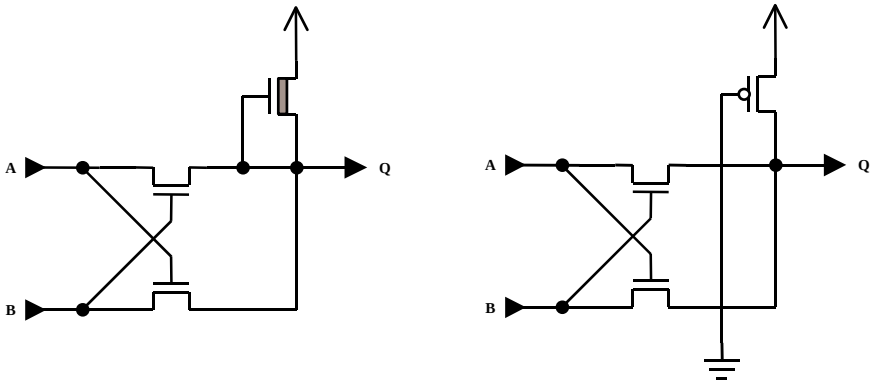
This can be implemented in a number of ways. Firstly using NOR or NAND gates:



This leads to an nMOS implementation using 12 transistors or CMOS using 16 (or 18 with the extra invertor). However, a more compact solution can be achieved using pass transistors:



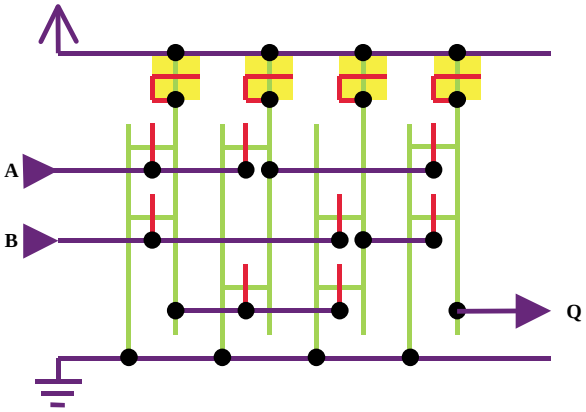
The nMOS version uses 6 transistors and the CMOS version uses 8 (or only 4 if the complementary parts of the transmission gates shown in grey are omitted). A little further thought reduces this to only 3 transistors:



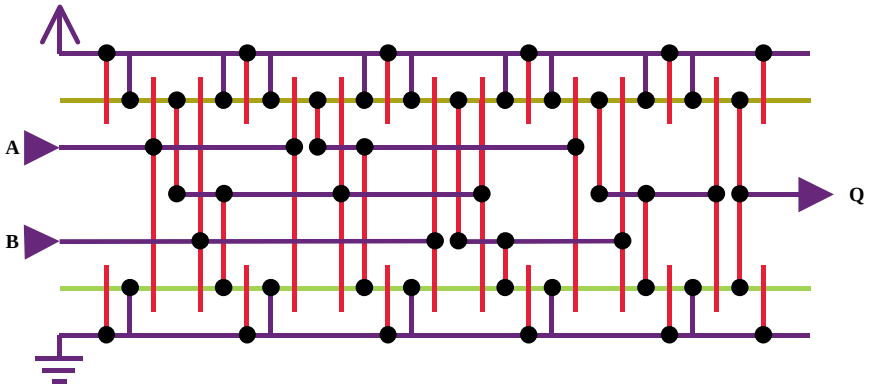
However, these simpler circuits must be used with care. The inputs to the pass transistor logic must be driven signals, not just potentials, or else there might be *charge sharing* (reverse flow of information). The 3-transistor circuits have passive pull-ups and so will dissipate power.

Stereotyped design

Random logic consisting of nMOS NOR gates can be laid out in a regular form as a *Weinberger gate array*. For example, the 12 transistor exclusive NOR gate could be laid out as follows:



Random CMOS can be laid out similarly as a gate matrix:



This example uses only NAND gates and an invertor, but clearly arbitrary gates including transmission gates could be laid out in this way. Both of these schemes have the advantage that they are amenable to automatic layout in a reasonable space.

PLAs

When several different functions of a set of input signals are required, a *programmable logic array* (PLA) may be useful.

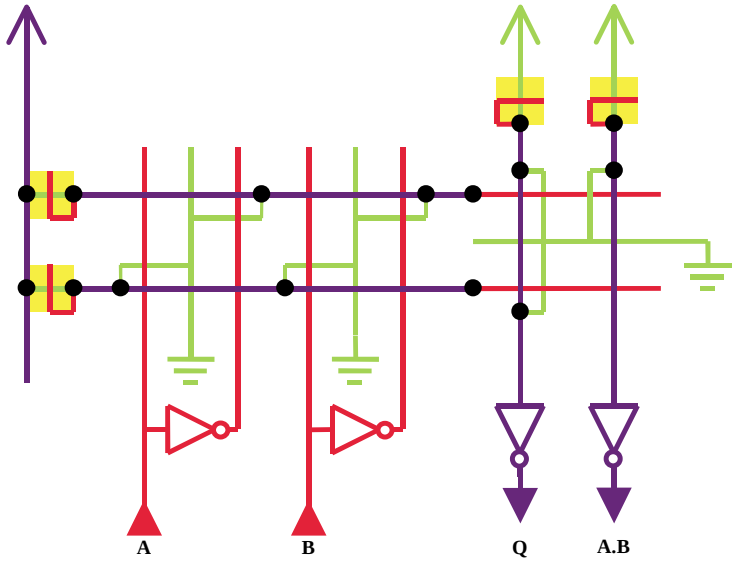
Any boolean function can be reduced to disjunctive normal form - a sum (OR) of products (AND) of the inputs and their inverses. The product terms are referred to as *minterms*.

Using only n-type transistors, it is convenient to remember de Morgan's law when computing the minterms: $X \cdot Y = \overline{\overline{X} + \overline{Y}}$

The exclusive NOR function can be thus written as:

$$\begin{aligned} Q = & \quad A \cdot B + \overline{A} \cdot \overline{B} \\ = & \quad \overline{\overline{A} + \overline{B}} + \overline{A + B} \end{aligned}$$

This leads to the following nMOS layout:

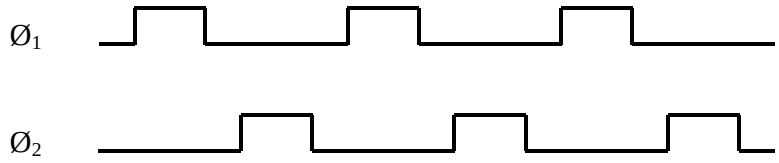


Note how *distributed gates* are used in the pull-down circuitry of the minterms. The regular structure of the PLA again makes it amenable to automatic layout. This is particularly useful when several different functions are being computed that share minterms.

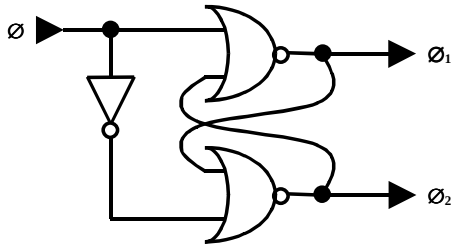
CMOS PLAs are similar and use p-FETs with their gates tied low as passive pull-ups. Alternatively, dynamic circuitry using a clock can be used.

Clocked logic

Data can be moved through sections of combinational logic under the control of a clock signal. It is convenient to use a *two-phase non-overlapping* clock:



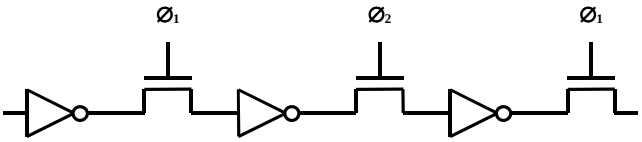
$Ø_1$ and $Ø_2$ can be generated on the chip from a single external clock:



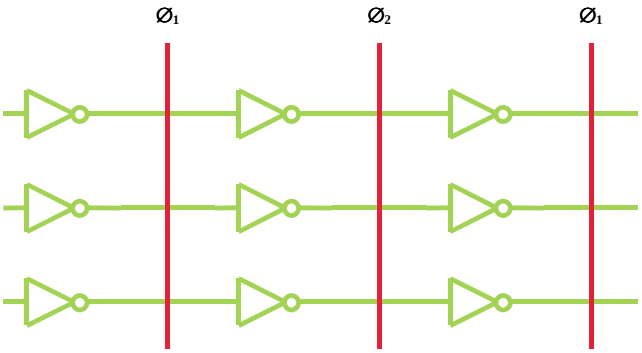
The durations and separations of the high periods can be controlled by introducing delay into the feedback loops.

Shift register

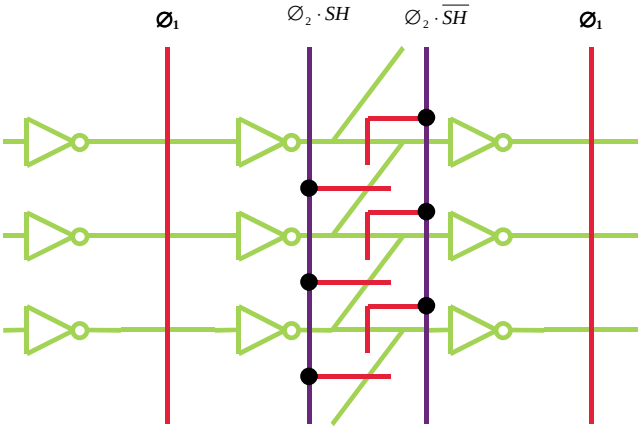
A shift register can be made by connecting a sequence of inverters together using pass transistors switched on alternate clock phases:



This can be extended in parallel to shift words rather than bits:



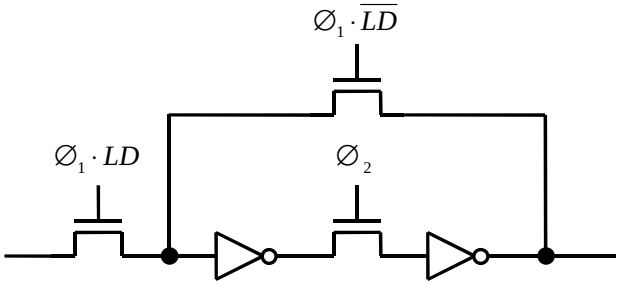
The clock signal can be combined with a control signal to give different modes of operation. For example, the shift register could rotate the bits in a word:



Notice that the SH signal has to be held throughout the $Ø_2$ phase and only changed during $Ø_1$.

Pseudo-static register

A clocked latch can be made by storing a bit as charge on the input of an invertor, and refreshing it every so often:



If LD is high during $Ø_1$, new data is loaded in to the latch. If LD is low then the old data is retained. The feedback loop is broken by both phases of the clock.

This is an example of *dynamic logic* and imposes a lower limit on the clock frequency. Charge can be expected to persist on the signal tracks for a thousandth of a second or so, depending on their capacitance. The clock rate must, therefore, exceed 1 kHz and should be considerably higher for safety.

Stack

Consider the design of a hardware stack for, say, *w* words, each containing *b* bits. When designing the layout of a circuit like this, it is important to consider the composition of repeated elements. A stack has three operations: *Push*, *Pop* and *Hold*. Four control signals **SHR, TRR, SHL** and **TRL** can be generated from these as follows:

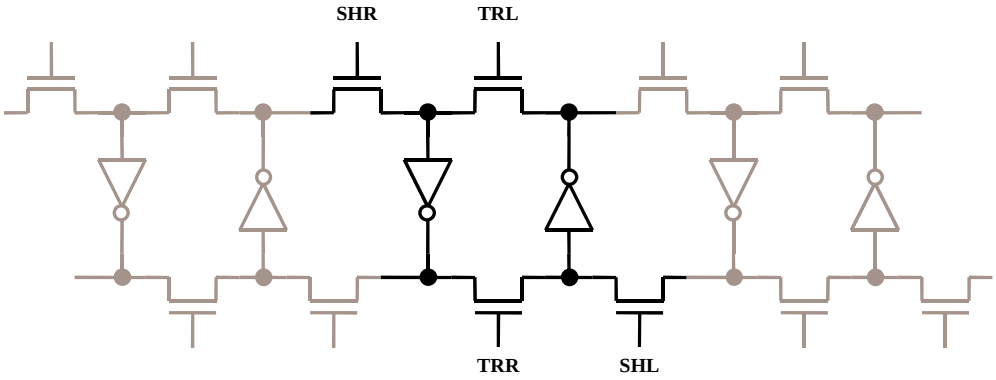
$$\begin{aligned} \text{TRL} &\leftarrow Ø_1.Hold \\ \text{TRR} &\leftarrow Ø_2.Hold \\ \text{SHR} &\leftarrow Ø_1.Push \\ \text{SHL} &\leftarrow Ø_2.Pop \end{aligned}$$

Or, put differently:

Operation	Phase	
	$Ø_1$	$Ø_2$
Hold	TRL	TRR
Push	SHR	TRR
Pop	TRL	SHL

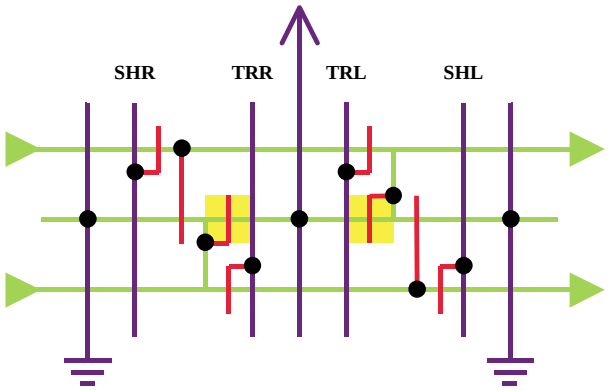
and these used to control the flow of data. Note that data can only be pushed into the stack during $Ø_1$ and only popped out during $Ø_2$. The second table emphasises the clock phases appropriate for different operation. The control signals should only change when both clocks are low.

A single bit store can now be constructed thus:



where the grey parts represent adjacent bit stores in a regular array.

The individual stack element can be laid out as follows:

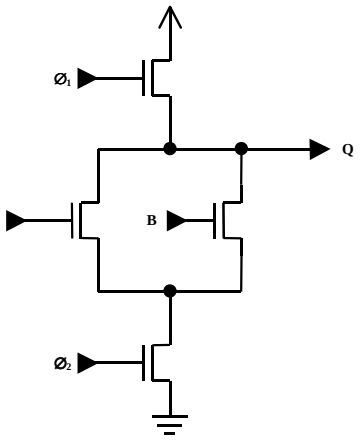


which allows adjacent elements to be connected simply by abutting them, sharing the ground lines.

Pre-charging

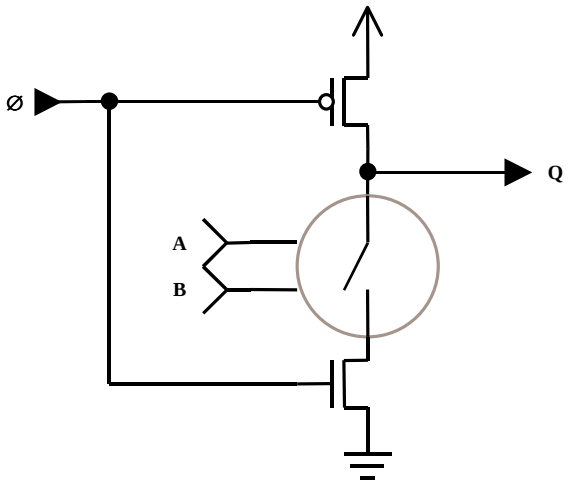
With nMOS, the depletion pull-ups have to be 4 or 8 times weaker than the pull-down transistors to give the correct ratio in the potential divider. With CMOS, the p-type pull-ups are 2½ times less conductive than the n-type pull-downs. In both cases, this makes switching to one slower than switching to zero.

One solution is to *pre-charge* the output of a gate during one clock phase and then discharge it selectively through a pull-down network during a second phase.



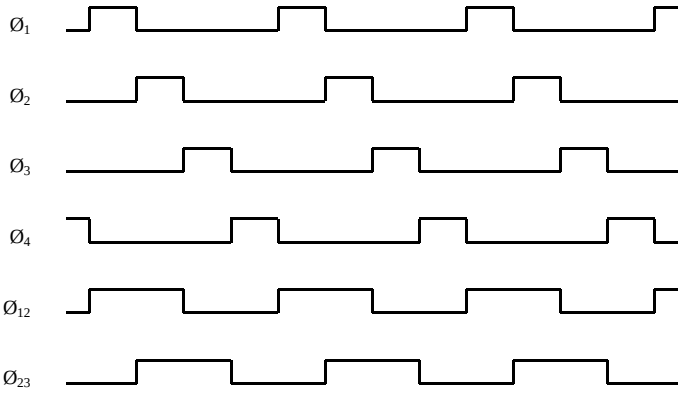
This is useful when the output lines have large capacitance, but care must be taken only to read the output after it has been discharged. In particular, this means that two such logic blocks can not be concatenated – when $Ø_2$ goes high the output may change as the pull-down is activated; however the transient high output may have discharged the output of the next stage in the mean time. This is known as an *internal race*.

Pre-charging is particularly useful with CMOS where it dispenses with the need for a fully complementary pull-up circuit. A similar design can be used:



where the output is pre-charged when Ø = 0 and evaluated when Ø = 1. The same restriction on concatenation applies.

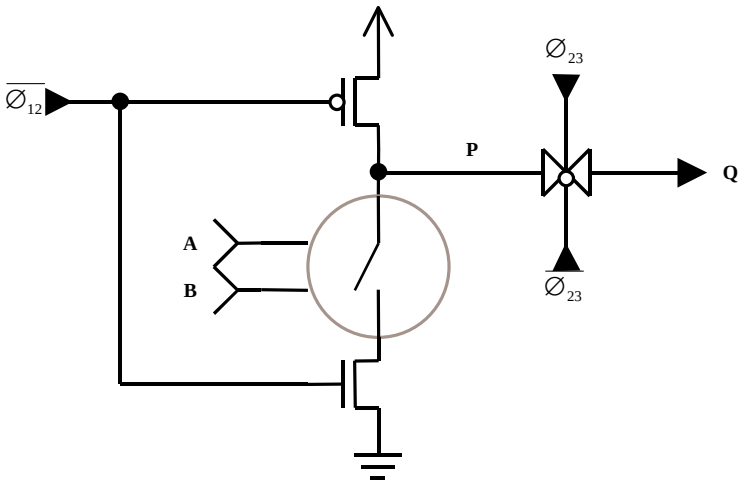
One solution to this involved the use of a four phase clock. Given four abutting or slightly overlapping clocks:



form derived signals such as

$$Ø_{12} \leftarrow Ø_1 + Ø_2$$

Then a circuit such as:



works as follows:

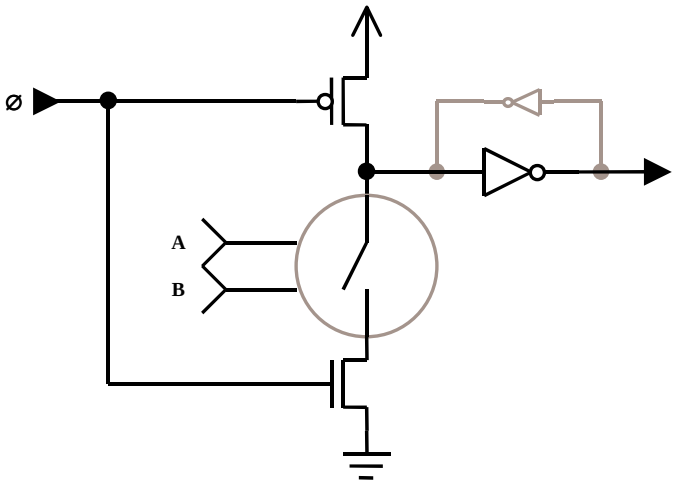
Phase 1 – Pre-charge **P**, hold existing **Q**.
Phase 2 – Continue to pre-charge **P**, pre-charge **Q**.
Phase 3 – Evaluate **P**, pass output to **Q**.
Phase 4 – Continue to evaluate **P**, hold new **Q**.

We can call this a *type 3* gate (because it samples its inputs during phase 3), and note that its outputs are held during the whole of phases 4 and 1. It is therefore possible to drive type 4 and type 1 gates safely from a type 3 gate. In fact, there is a general set of rules for the composition of the different types of block:

Type 1 can drive types 2 and 3;
Type 2 can drive types 3 and 4;
Type 3 can drive types 4 and 1; and
Type 4 can drive types 1 and 2.

This makes the scheme suitable for pipelined designs.

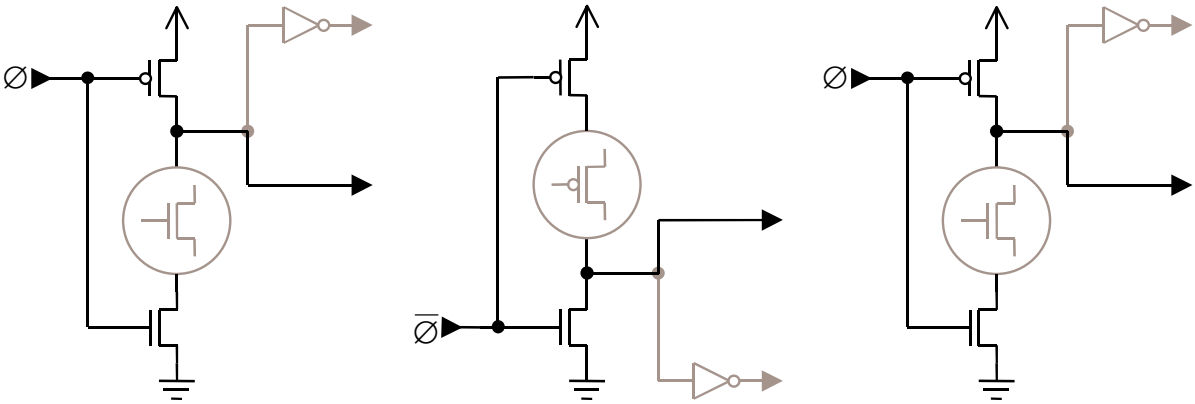
It is, however, rather complicated to implement and there are alternative schemes. One is *Domino* logic:



This works by pre-charging when Ø= 0, at which the output goes low. On Ø = 1, the pull-down network is evaluated and the output may rise from 0 to 1. Since only a rising edge is possible, there can not be a spurious discharge of the next stage.

The gate can even be made to hold its output by adding the weak feed-back invertor shown in grey.

This limits the logic to non-inverting structures, and extra buffers are required. There can also be charge-sharing and race problems. A solution to this is *NORA* (no race) logic:

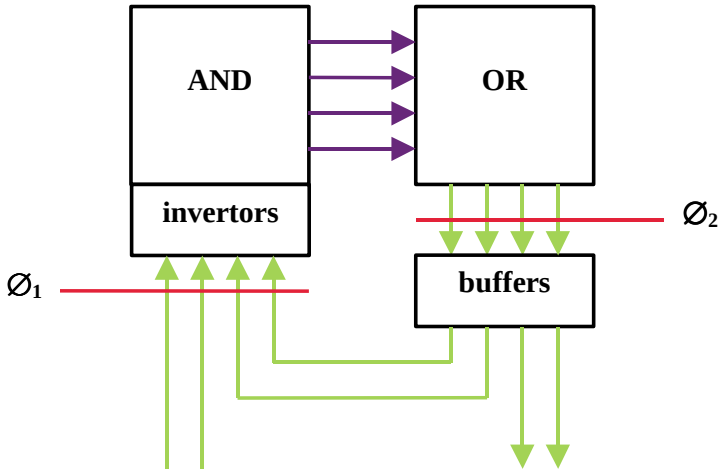


There are two types of logic block using n-type and p-type evaluation circuits. On Ø = 0, the n-type block pre-charges its output high and the p-type block pre-charges its output low. On Ø = 1, both blocks evaluate.

There is a simple rule of composition: p-type blocks drive n-type blocks and n-type blocks drive p-type blocks. However, either type of block may be connected to one of the same class via an invertor, effectively using a Domino circuit. There is a performance penalty in using relatively slow p-type pull-up networks.

Clocking PLAs

The operation of a circuit can be controlled by a finite state machine made from a state register and a PLA. The PLA processes the current state and any input signals (often condition codes arising from the previous operation) to give the new state and any ouput signals (often controls to other parts of the circuit).

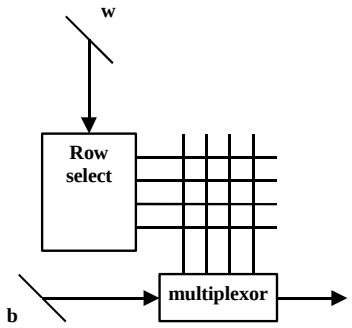


It is convenient to latch the input signals on one clock phase and the output signals on a second, with the clock period being sufficiently long to compute the outputs in the PLA.

CMOS PLAs can be made either by using static pull-ups in the form of p-type transistors with their gates wired low or by pre-charging the outputs and discharging them on a second clock phase.

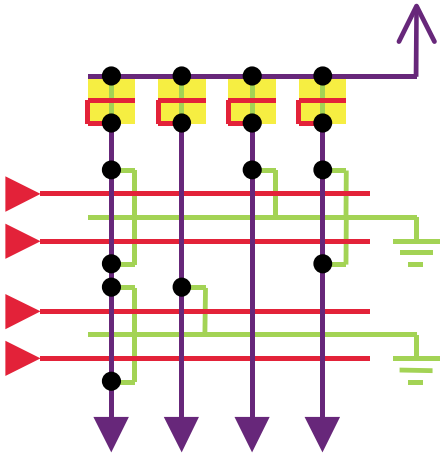
Memory design

Memories are usually constructed as two dimensional arrays of bits. Thus a memory containing 2^w words each of 2^b bits will be configured as 2^w rows by 2^b columns. w address bits will be decoded to give the row and either the whole word will be output or multiplexor used to select a single bit using a further b address bits.



Read-only memory

A *read-only* memory (ROM) is like a PLA with all the possible minterms being calculated. The individual memory cells can be very compact; here is a 4 x 4 fragment of the memory array:

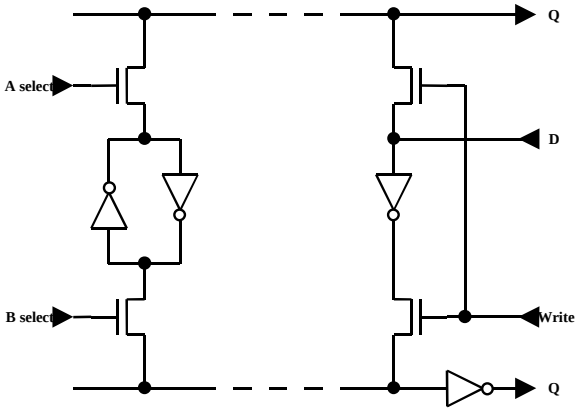


Diffusion tabs are run under the polysilicon word lines wherever a 0 is to be stored, other bit positions read as 1. The 4 words stored here will read as 4, 6, 3 and 7.

Progammable read-only memories (PROMs) allow the diffusion tabs to be switched in electrically. *Erasable* PROMs allow this switching to be reversed, either by exposure to ulta-violet light (EPROMs) or under digital control (*electrically erasable* PROMs or EEPROMs).

Static read/write memory

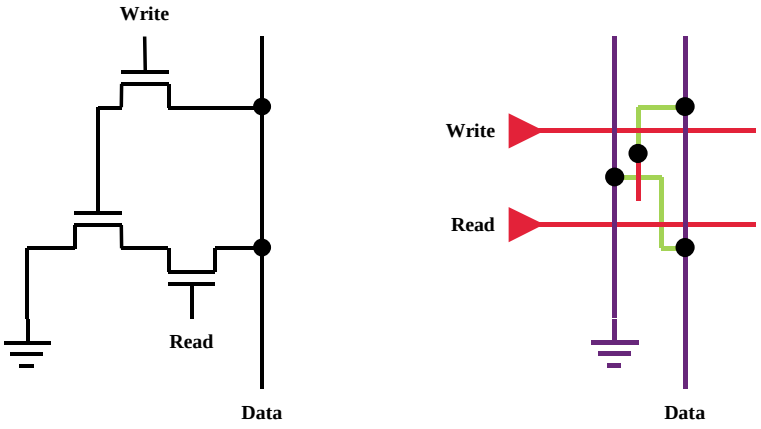
The simplest form of writeable memory (RAM) is static memory. A bit is stored in a pair of cross-coupled invertors, with separate circuits to control the reading and writing of the data.



The memory has two independent ports for reading; both selection lines are opened for writing. Six transistors are required to store each bit, plus some overheard for the control circuitry.

Dynamic RAM

Fewer transistors are needed if the bit is stored as charge on the gate of a FET.

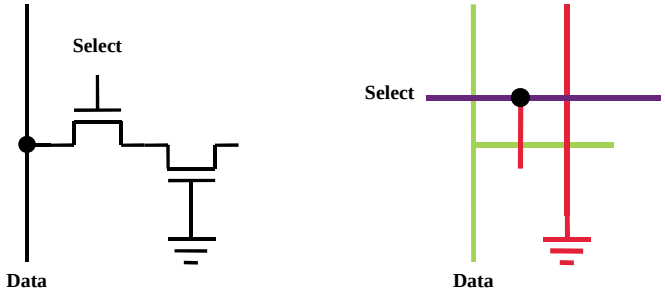


The three-transistor memory cell operates as follows:

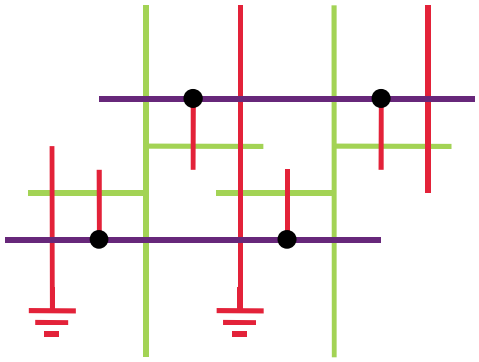
- Write by putting data on **Data** and strobing **Write**
- Read by pre-charging **Data** and strobing **Read**; the value obtained has to be inverted.
- Refresh by reading and re-writing at least every millisecond or so.

Less circuitry is required for each individual bit at the expense of more sophisticated control circuits.

This is taken to an extreme with a one-transistor memory cell:



The bit is stored as charge under the grounded gate of a second transistor. Again, refreshing is required and reading requires the use of subtle analogue sense amplifiers. The tessellated layout is, however, very compact:



Really dense memory circuits use specialised processes not available for normal digital logic.

System design

A simple processor includes three principal elements:

- a register file for fast memory
- an arithmetic and logical unit for calculations, and
- control logic in PLA

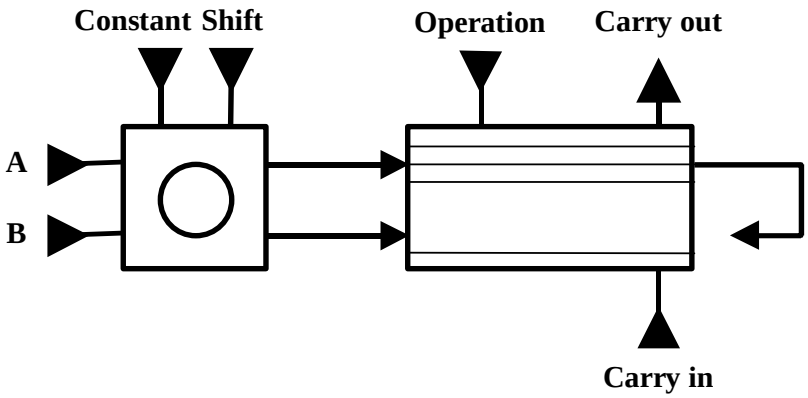
VLSI structures for the first and last of these have already been presented. This section discusses VLSI structures for ALUs.

Arithmetic and logical unit

The ALU may contain units such as

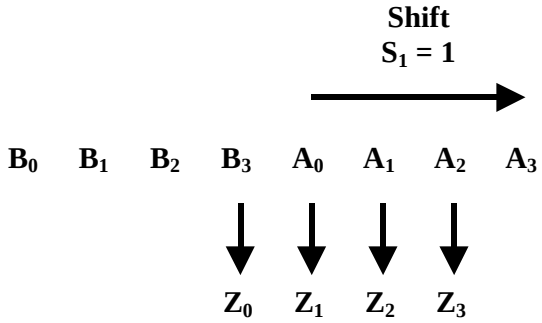
- a barrel shifter
- a number of function calculators, and
- a carry chain.

These are often arranged as bit slices which are repeated to give the desired word size.

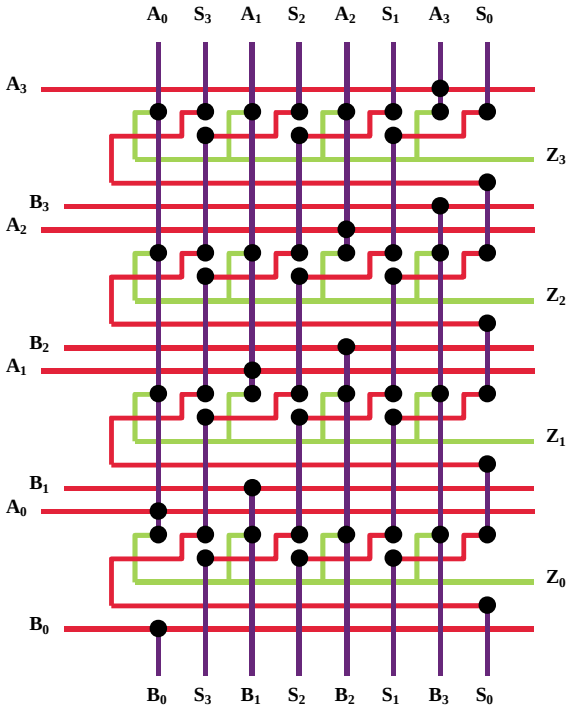


Barrel shifter

For example, a 4 bit barrel shifter concatenates two input words $B_0...B_3$ and $A_0...A_3$, shifts them a given number of places and emits the bottom 4 bits $Z_0..Z_3$.



If the size of the shift is encoded in unary on a set of control signals $S_0..S_3$, it can be laid out quite compactly using pass transistor logic:



A and B would probably be drawn from a dual-ported register file; if they read the same register then this unit could be used for arbitrary circular shifts. The shift unit is also a convenient place to introduce a constant into the ALU using the alternative A from above or B from below.

Carry chain

Within the ALU, there are several approaches to handling carry:

- ripple carry,
- *Manchester* carry chain,
- carry skip,
- carry select, or
- full carry look-ahead.

The Manchester carry chain fits well with a bit-slice approach using a two phase clock. Consider a full adder with inputs A, B and C_{in} and outputs Q (for the sum) and C_{out}. It is convenient to calculate control signals K (for *kill* when C_{out} is 0) and P (for *propagate* when C_{out} is the same as C_{in}); otherwise C_{out} will be pre-charged to 1.

The functions are as follows:

A	B	C_{in}	Q	C_{out}	K	P
0	0	0	0	0	1	0
0	0	1	1	0	1	0
0	1	0	1	0	0	1
0	1	1	0	1	0	1
1	0	0	1	0	0	1
1	0	1	0	1	0	1
1	1	0	0	1	0	0
1	1	1	1	1	0	0

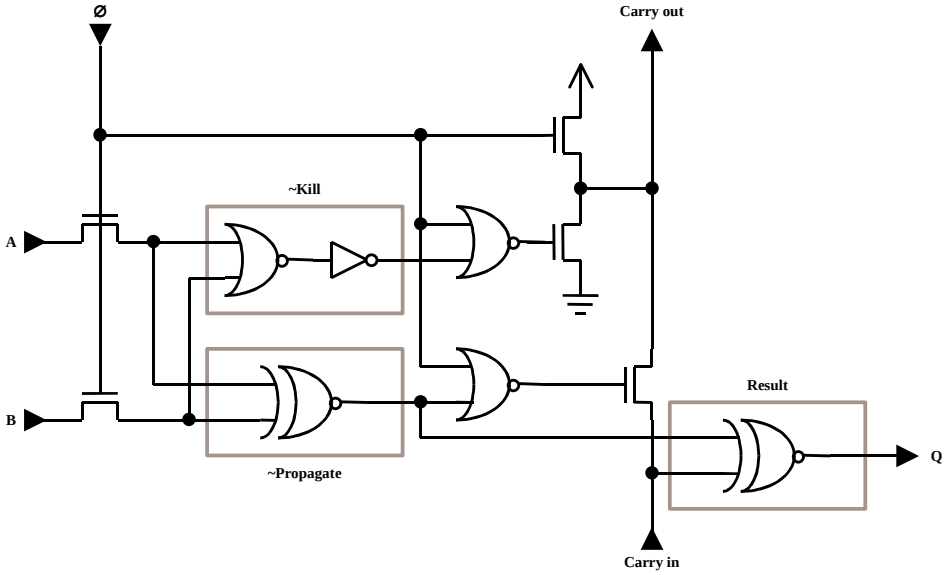
Observe that K and P can be calculated as soon as A and B are known, without waiting for C_{in}:

$$K = \sim(A + B)$$
$$P = A \oplus B$$

then

$$Q = C_{in} \oplus P = \sim(C_{in} \oplus \sim P)$$

The resulting circuit is as follows:



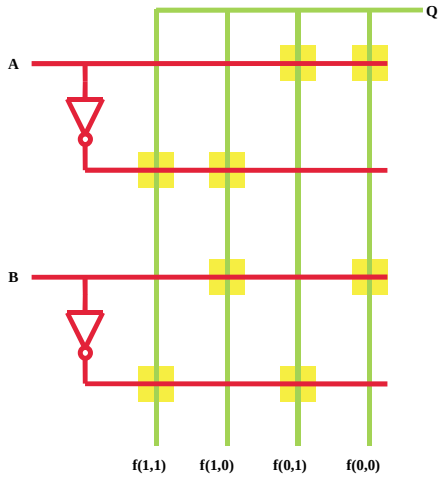
The circuit is precharged, also latching the A and B inputs, when Ø=1 and evaluated when Ø=0.

This adder could be made to perform a number of different operations by introducing general functions for the carry kill, carry propagate and result calculations. For example,

$$A - B = A + \sim B + 1$$

Function generators

A four into one multiplexor built in pass transistors can be used as a general *red-green function generator*:

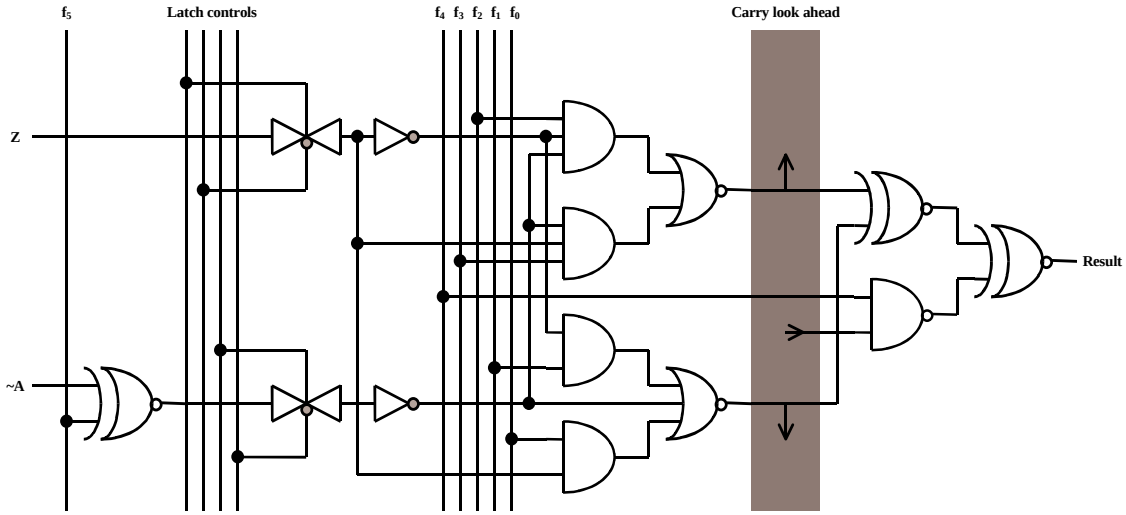


The four values f (1, 1), f (1, 0), f (0, 1), f (0, 0) are fed in at the bottom and f (A,B) is output. Thus 13 bits (4 each for 3 function units and carry in to the bottom stage) determine the ALU operation.

Here are some possible control values:

Operation	**~K**	**~P**	**R**	**C_0**	
A	15	12	12	0	Just A
A & B	15	8	12	0	Bit-wise AND
A | B	15	14	12	0	Bit-wise OR
A ⊕ B	15	6	12	0	Bit-wise exclusive OR
A + B	14	9	9	?	Add with carry-in
A – B	13	6	9	1	Subtract
B – A	11	6	9	1	Reverse subtract

The ARM processor uses direct combinational logic:



The six control wires now determine the function:

Operation	f_5	f_4	f_3	f_2	f_1	f_0	
A	0	1	0	0	0	0	Just A
A & B	0	1	0	1	0	0	Bit-wise AND
A | B	0	1	0	0	0	1	Bit-wise OR
A ⊕ B	0	1	1	0	0	1	Bit-wise exclusive OR
A + B	0	0	0	1	1	0	Add with carry-in
A – B	0	0	1	0	0	1	Subtract
B – A	1	0	0	1	1	0	Reverse subtract

Fast carry

For larger word sizes, other techniques such as carry look-ahead carry select and carry skip give better performance. Recall the equations for a full adder:

$$\text{Sum} = A \oplus B \oplus C_{in}$$

and

$$\begin{aligned} C_{out} &= A.B + A.C_{in} + B.C_{in} \\ &= A.B + (A + B).C_{in} \\ &= G + P.C_{in} \end{aligned}$$

where G and P are generate and propagate signals defined as:

$$G = A.B$$

and

$$P = A + B$$

In general, for stage *i* of an *n*-bit adder:

$$C_{i+1} = G_i + P_i.C_i$$

where

$$G_i = A_i.B_i$$

and

$$P_i = A_i + B_i$$

Several stages can then be grouped together:

$$\begin{aligned} C_{i+1} &= G_i + P_i.G_{i-1} + P_i.P_{i-1}.G_{i-2} + P_i.P_{i-1}.\ P_{i-2}.G_{i-3} + P_i.P_{i-1}.P_{i-2}.P_{i-3}.C_{i-3} \\ &= G_{i-3,i} + P_{i-3,i}.C_{i-3} \end{aligned}$$

where

$$G_{i-3,i} = G_i + P_i.G_{i-1} + P_i.P_{i-1}.G_{i-2} + P_i.P_{i-1}.\ P_{i-2}.G_{i-3}$$

and

$$P_{i-3,i} = P_i.P_{i-1}.P_{i-2}.P_{i-3}$$

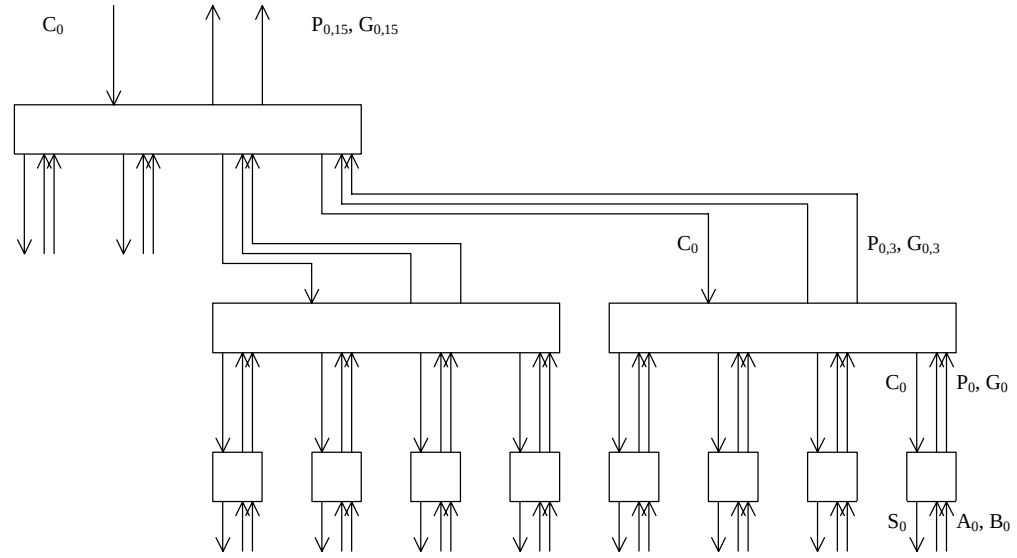
Moreover:

$$G_{i-15,i} = G_{i-3,i} + P_{i-3,i}.G_{i-7,i-4} + P_{i-3,i}.P_{i-7,i-4}.G_{i-11,i-8} + P_{i-3,i}.P_{i-7,i-4}.P_{i-11,i-8}.G_{i-15,i-12}$$

and

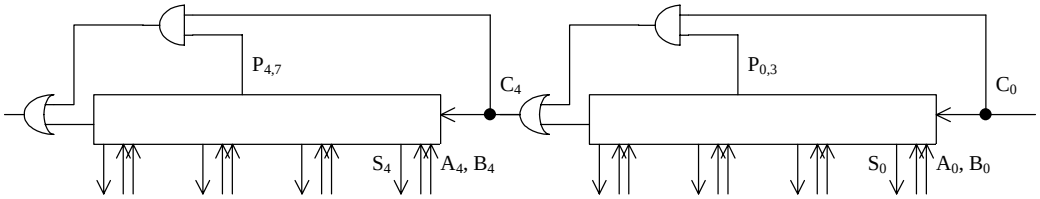
$$P_{i-15,i} = P_{i-3,i}.P_{i-7,i-4}.P_{i-11,i-8}.P_{i-15,i-12}$$

Carry look-ahead implements these equations as a tree structure of identical circuits:



Carry select divides the *n*-bit word up into blocks of, say, 4 bits. Two adders are used for each block, one assuming a carry-in of 0 and the other a carry-in of 1. The actual carry-in is then used to select between the outputs of the two adders, including selecting the carry-out which is rippled on to the selector for the next block.

Carry skip exploits the observation that the propagate signals are much easier to compute than generate, allowing fast carry propagation alongside blocks of bits producing the generated carry signals more slowly. The system is particularly well suited to dynamic logic.



The system can be further optimised by having blocks of different sizes – short at the beginning and the end of the word, and longer in the middle. This reduces the worst case carry propagation delays.

Performance can be summarised as follows:

	Time	**Space**
Ripple	O(*n*)	O(*n*)
Carry look-ahead	O(log *n*)	O(*n* log *n*)
Carry select	O($\sqrt{n}$)	O(*n*)
Carry skip	O($\sqrt{n}$)	O(*n*)

Carry save is used in multipliers and retains the carries arising from partial sums to be included in the next addition.

Detailed layout and fabrication

Having designed the stick diagram for a circuit, the next problem is to translate it into actual layout. This involves:

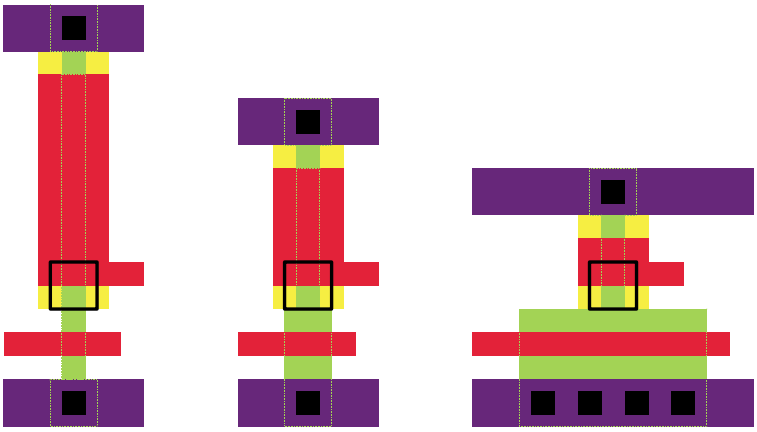
- shuffling tracks around to overlap each other where possible (running metal over the top of gates and so on)
- fixing widths for the tracks in the various materials, taking account of factors such as capacitance, resistance and current carrying ability
- deciding on sizes for transistor
  - the ratio of the channel length to width determines the resistance of the transistor when turned on
  - the ratio of the resistance of the pull-up and pull-down circuits in an nMOS gate determines its sensitivity

resulting in a set of areas in which the different materials are to be made (or *fabricated*).

Sizing nMOS gates

In nMOS, the ratio of the pull-up and pull-down resistances should be 4:1 for ordinary gates and 8:1 for gates driven by pass transistor logic.

An 8:1 gate could be made in several ways:



- an 8:1 pull-up and a 1:1 pull-down (low power)
- a 4:1 pull-up and a 1:2 pull-down (small)
- a 1:1 pull-up and a 1:8 pull-down (fast)

Fabrication

Fabrication involves the transfer of the layout to wafers of crystalline silicon. This involves several stages of chemical processing.

Photographic plates called *masks* are made from each layer of material by photographic means or by writing with a steered electron beam (the latter being more common for finely detailed masks).

These are used to control each processing steps as follows:

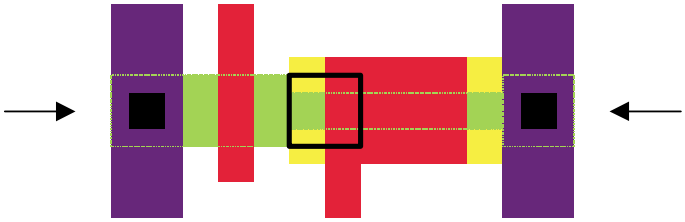
- Coat the wafer with a photographic emulsion known as a *resist*.

- Expose this selectively to ultra-violet light using the mask.
- Develop the resist and remove the exposed material (or the unexposed material, depending on its polarity)
- Process the wafer with some chemical implantation – only the areas not protected by the resist will be affected.
- Strip the remaining resist.

In practice, many of the processes are applied to the whole wafer and then selectively removed using a mask and resist. Wafers can be exposed directly using a electron beams, without using masks as an intermediate stage.

NMOS processing

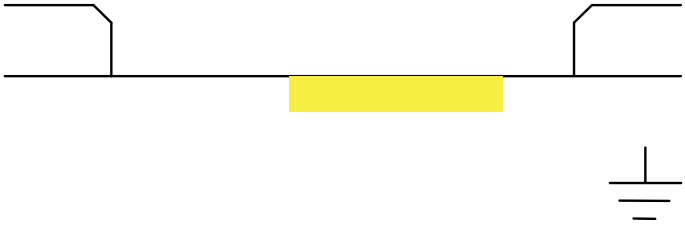
Consider the manufacture of an nMOS invertor. Given the layout (in plan view):



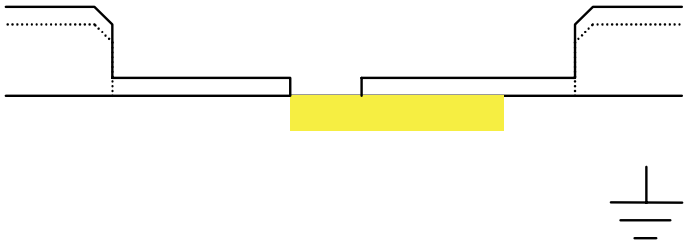
Processing involves the following steps (with cross sections through the chip being shown):

- Dope the crystalline silicon wafer to make a p-type substrate.
- Grow a (relatively) thick layer of silicon dioxide over the whole wafer)
- Remove this in the source, drain and channel regions (using the green mask).

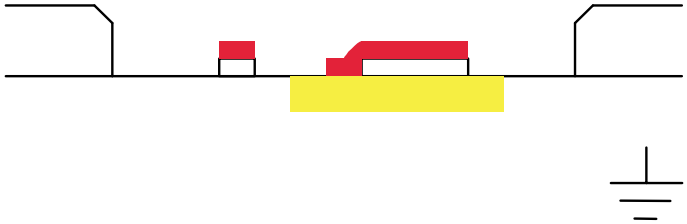
- Implant ions for depletion mode transistors (using the yellow mask).



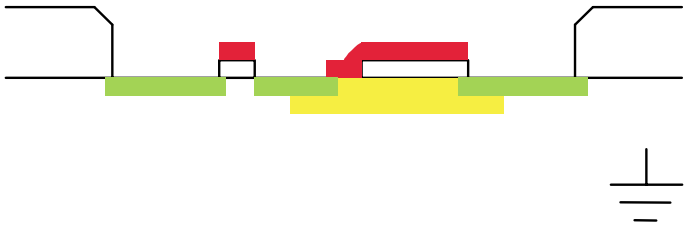
- Regrow a thin layer of silicon dioxide over the whole wafer.
- Remove this for buried contacts between polysilicon and diffusion (using a mask derived from the black outlines).



- Make the polysilicon (using the red mask).
- Remove a thin layer of silicon dioxide from the whole wafer – the polysilicon will preserve it in the gate regions.

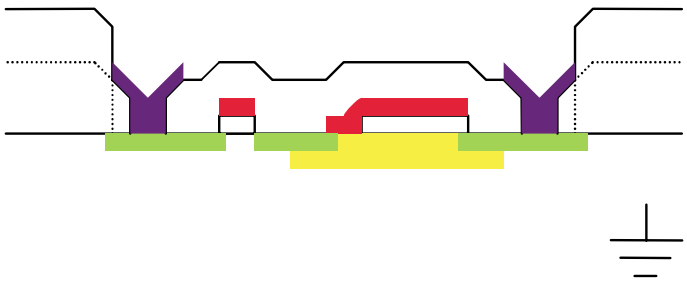


- Diffuse n-type material over the whole wafer – this only affects the source and drain regions, and aligns automatically with the polysilicon gate regions.



The *self-alignment* achieved by using the polysilicon as a template to control the removal of thin oxide and the implanting of the diffusion is an important feature of the process.

- Cover with a thick layer of silicon dioxide insulator.
- Cut contacts through the insulator (using a mask derived from solid black squares).
- Cover with aluminium.
- Etch away unwanted metal (using the inverse of the blue mask).



- Cover with a protective layer of silicon dioxide.
- Cut holes for bonding leads.
- As the diagrams suggest, the surface of the wafer becomes very uneven during processing and additional steps are taken to level, or *planarise*, it.

CMOS processing

CMOS processing is similar to nMOS processing, but required two different types of diffusion in two separate regions.

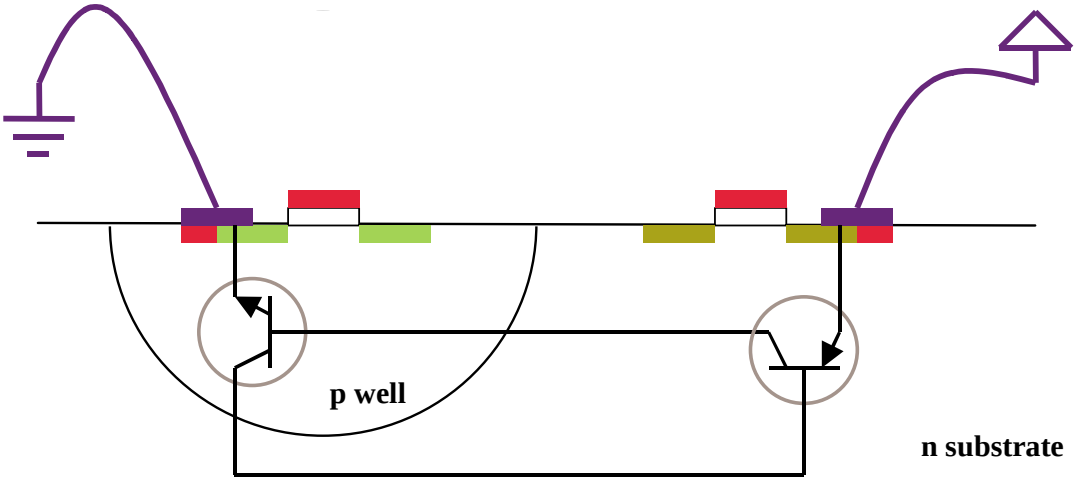
This can be achieved in a number of ways:

- Bulk p-well - p-type transistors are built directly on an n-type substrate and n-type transistors are made in deep p-type wells.
- Bulk n-well type transistors are built directly on a p-type substrate and p-type transistors are made in deep n-type wells. (This is widely used, being compatible with nMOS.)
- Twin tub-separate wells are made for each type of transistor
- Silicon on insulator (usually sapphire for SOS)
  - grow crystalline silicon on a sapphire substrate,
  - remove it except in gate regions,
  - make gates as usual.

There are also exotic experimental techniques such as laser annealing of raw silicon.

Modern processes also provide several (3-5) layers of metal connection.

The main problem with CMOS fabrication, apart from the increased complexity caused by using more masks, is the formation of parasitic bipolar transistors between the wells.



If there is a spike on a voltage rail, these can form a feed-back amplifier and *latch-up*, destroying the chip. The solution is to ensure that there are low resistance, or *ohmic*, contacts between p-type regions and ground and n-type regions and Vcc.

Keeping it clean

Wafer processing has to performed in extremely clean surrounding.

- A human hair is about 50µ (microns = 10^{-6}m 10^{4}Å) in diameter.
- A dust particle 10µ in diameter is about as small as is visible by the human eye and floats in the air.
- A transistor channel may be as little as 0.2µ long.
- DNA is about 2nm = 0.002µ wide.

- Atoms are about 0.1 to 0.4 nm in diameter.

One measure of cleanliness is the number of 10μ particles in a cubic meter of air.

- Ordinary rooms have about 10^7
- Hospital operating theatres have about 10^5
- Fabrication lines have less than 10^3
- Mask making areas have less than 10.

Blinking scatters several thousand 10μ particles from the eyelashes into the air.

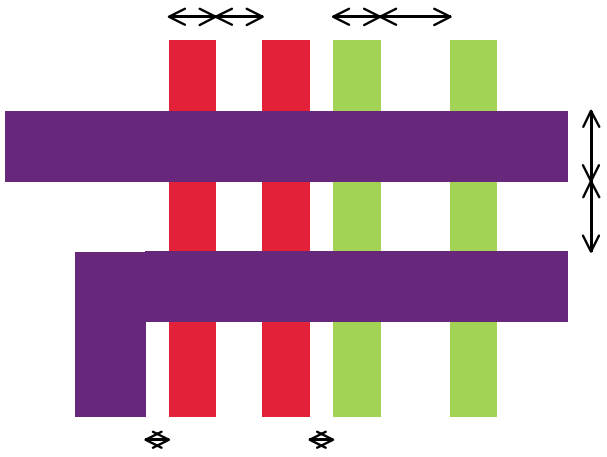
Design rules

There are limitations to the precision with which individual processing steps can be performed and with which separate processes can be aligned. This gives rise to a number of *design rules* specifying the minimum sizes of features and the separations and overlaps that must be established. Each process has an individual set of design rules, usually specified as dimensions in microns.

Mead and Conway reduced these complex rules to a relatively simple set of rules expressed in terms of a normalised scaling factor λ (lambda rather than μ for micron). λ represents the maximum amount by which any singe mask may be displaced; if two masks are displaced by λ in opposite directions, the chip will just work, but performance will be marginal.

The Mead and Conway rules for tracks in the three nMOS conducting layers are:

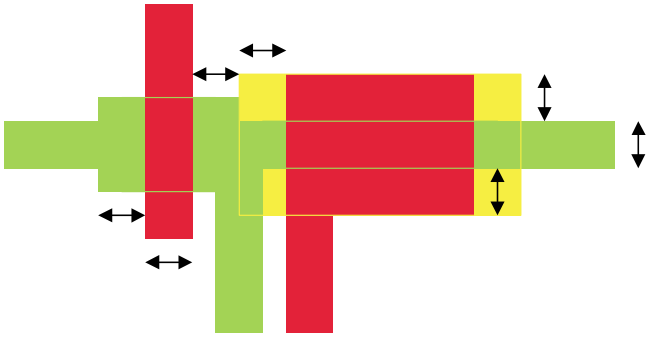
- Minimum polysilicon width 2λ
- Minimum polysilicon separation 2λ.
- Minimum diffusion width 2λ.
- Minimum diffusion separation 3λ.
- Minimum polysilicon separation from diffusion 1λ.
- Minimum metal width 3λ
- Minimum metal separation 3λ.
- Minimum polysilicon separation from metal 1λ (where possible).



The rules for nMOS transistors are:

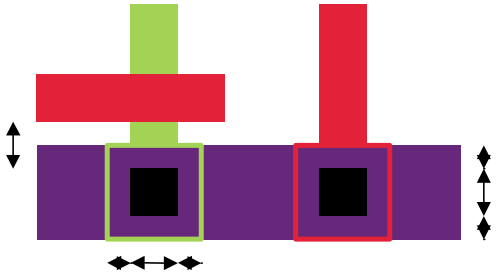
- Minimum transistor size 2λ square (essentially because of minimum track widths for diffusion and polysilicon).

- Polysilicon must continue past transistor for at least 2λ.
- Diffusion must continue around transistor for at least 2λ.
- Implant must extend for 2λ around a depletion transistor.
- Minimum separation between implant and enhancement transistor 2λ



The rules for contacts between the nMOS layers are:

- Minimum contact size 2λ square.
- Both materials must extend for 1λ around the contact.
- Minimum separation between contacts 2λ.
- Minimum separation between contact and transistor 2λ.
- Thin oxide removal must extend for 1λ around a buried contact and for 2λ along conduction diffusion.



Rules for CMOS are analogous, if somewhat more complicated.

Processes are usually described by the size of the smallest transistor, which will be 2λ square. 0.25μ CMOS is now (2000) common and 0.18μ processes are appearing.

Performance considerations

The resistance and capacities of the different materials will affect speed. The following are rough figures:

- Metal – about 0.1 Ω/□ and 0.3 x 10^{-4} pF/μ^2 (pF = 10^{-12} Farad)
- Polysilicon – about 50 Ohm/□ and 0.4 x 10^{-4} pF/μ^2
- Diffusion – about 10 Ohm/□ and 1 x 10^{-4} pF/μ^2. There is also *edge-wall capacitance* related to the perimeter of the diffusion area.
- Gate polysilicon – about 4 x 10^{-4} pF/μ^2.
- Conducting channel – about 10^4 Ω/□

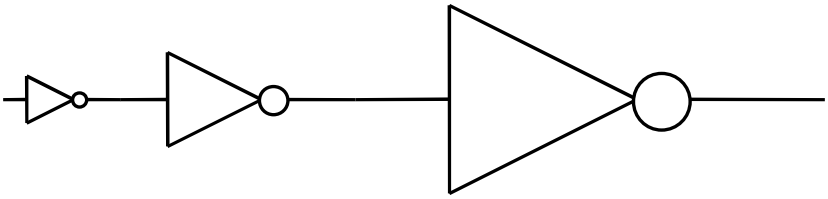
Note how the sheet resistance of material is measured in Ω/□ (read as Ohms per square). The current through power and ground lines must also be considered – the metal migrates under the influence of excessive currents.

- 1μ thick metal can carry about 1mA for every micron of width.
- A 4:1/1:1 invertor draws about 0.1 mA and a 3:1/1:3 invertor about 0.15 mA.
- As a general rule, about half the gates will be conducting at any time.

Driving capacitive loads

Long tracks with a large fan-out (for example, clock signals) have to be driven carefully if reasonably sharp transitions are to be achieved. This can be implemented by using chains of invertors graduated in size or by using analogue buffer circuitry.

Consider a chain of invertors, each f times the size of its predecessor; that is, the first has a 4:1 pull-up and 1:1 pull-down, the second a 4:f pull-up and a 1:f pull-down, the third a 4:f^2 pull-up and a 1:f^2 pull-down and so on:

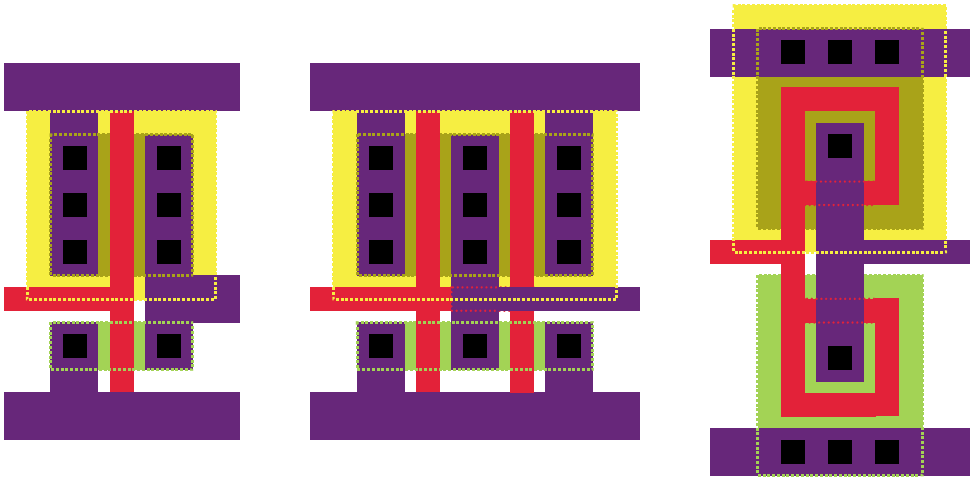


If the gate of the first invertor offers a capacitive load C_g, then the second one offers a load $f.C_g$, the third $f^2.C_g$ and so on. A chain of n such invertors can drive a final load $C_L = f^n.C_g$. So $n = \log_f (C_L / C_g) = \ln (C_L / C_g) / \ln f$.

If the time taken to charge a gate through a 4□ pull-up is t, then the delay through each of these invertors will be $f.t$ and the total delay will be $T = n.f.t = [f / \ln f] \, . \, t \, . \, \ln (C_L / C_g)$.

For any given load, the delay will be minimised when $f = e$ (the base of natural logarithms).

Large invertors require special designs for large transistors:



Fundamental limitations

It is worth asking what happens as technology advances and processing can be carried out more accurately.

The *constant field* model of MOS scaling applies a dimensionless factor α to manufacturing dimensions (length, width and thickness), voltages and processing concentrations, so that the electric field and the channel thickness remains unchanged. For example, with $\alpha = 2$, all dimensions would be halved.

Constant voltage is an alternative model in which only the manufacturing dimensions are scaled, leaving voltages unchanged, so the channel thickness increases by a factor α.

The effects can be summarised as follows:

		Constant field	Constant voltage
Length, width, thickness	L, W, D	$1/\alpha$	$1/\alpha$
Voltage	V	$1/\alpha$	1
Channel thickness	d	1	α
Gate area	$A = L \times W$	$1/\alpha^2$	$1/\alpha^2$
Channel resistance	$R \propto L \div (d \times W)$	1	$1/\alpha$
Current	$I = V \div R$	$1/\alpha$	α
Load capacitance	$C \propto A \div D$	$1/\alpha$	$1/\alpha$
Gate delay	$T \propto R \times C$	$1/\alpha$	$1/\alpha^2$
Static power consumption	$W = V \times I$	$1/\alpha^2$	α
Power density	$W \div A$	1	α^3
Current density	$I \div (W \times D)$	α	α^3

Under the constant field model, speed is increased by α, density by α^2 while keeping power density constant. With constant voltage, speed and density are increased by α^2 but power density goes up by α^3.

The approximations are plausible down to 0.1 micron geometries, when further effects (tunnelling, leakage, edge wall capacitance, metal migration, wire delay…) come into play.

Visible light has a wavelength of about ½ micron, so direct electron beam lithography has to replace masks for small devices and dry etching and ion implantation are used instead of chemical processing.

Power dissipation is a very important consideration with chips:

- 1 W/cm^2 can be dissipated from a plastic package.
- 2-4 W/cm^2 required a heat sink.
- More than 8 W/cm^2 required forced cooling.

Most circuitry speeds up if it is cooled down, so immersing the entire system in a cooling bath has additional advantages.

Yield

The *yield* of a fabrication process is the proportion of manufactured chips that work. It is affected by the size of each individual die, the quality of the materials and processing employed and the complexity of the process. As a rough approximation, yield varies with the die size, A, and the defect density, D, as follows:

$$Yield = k.e^{-A.D}$$

The following table gives and idea of the figures involved:

Year	**Defect density** defects/cm^2	**Die size** mm x mm	**Yield**	**# dice per 6" wafer**	**# good**
1984	1.83				
1987	1.16	10x15 15x15	18% 7%	85 49	15 3
1989	0.72	10x15 15x15	18% 7%	85 49	28 9
1992	0.38				

Unfortunately, semiconductor manufacturers are very cautious about releasing current yield figures, but one admits to having produced an 8" wafer that was free from defects.

Other technologies

As the physical limits of silicon semiconductor technology are reached, other materials and processes become interesting.

BiCMOS chips mix bipolar and CMOS circuits on a single die. The bipolar circuits are used where power or speed is required and the CMOS circuits for regular VLSI structures and non-critical digital parts.

Gallium arsenide (GaAs) is a promising contender (already widely used for analogue radio frequency circuits), having an electron mobility better by a factor of 6 than silicon. This gives particularly good speed/power performance. However, there are problems with fabrication, poor thermal conduction, release of arsenic and so on, which makes them expensive and low on yield.

The most promising design style for VLSI is *directly coupled FET logic* (DCFL) which looks rather like nMOS. The main drawbacks are the stringent processing requirements for the enhancement mode devices and a low noise threshold. Nevertheless, gate arrays are now available with 2000 gate equivalents and delays of the order of 100ps, dissipating 0.4 mW per gate.

Superconductors are potentially 10-20 times faster than conventional circuits, but the mechanical difficulties of refrigerating them mean that they are only used in exceptional circumstances.

Rather more speculative are technologies such as bipolar resonant tunnelling transistors using quantum effects, where electrons are treated as waves rather than particles. The active regions are of monatomic dimensions (0.01 – 0.02 microns) and the devices are 3 orders of magnitude faster than current semiconductors.

Testing

In the good old days it used to be possible to probe individual signal tracks on a chip using micrometer probes. As track widths have shrunk this becomes less tractable and inspection using scanning electron microscopes has become useful. An image can be synthesised using the voltage levels on the chip to provide contrast; thus digital and even analogue signal levels can be displayed on a picture of the chip. If the electron source is pulsed synchronously with the chip's clock, the action of the chip can be slowed down stroboscopically.

Semi-custom circuit design

Checking all the electrical characteristics of a large integrated circuit is a difficult process and a number of techniques have been developed to simplify the design. These involve the division of design into two parts:

- a preliminary, detailed design of elementary units, and
- a subsequent composition of these units to implement the circuit.

This is the familiar divide and conquer strategy of programming (and most other computer science). The latter step can rely on the performance achieved by the first step. This is known as *semi-custom design*, as full control of the silicon design is not available in the second stage of the design.

This can be applied to chip design in two ways:

- *Standard cell* design – where small custom units (such as gates, multiplexors, registers and so on) are designed and then arranged and connected to implement a circuit.
- *Gate arrays* (sometimes known as *uncommitted logic arrays*) - where an array of basic components (such as transistors and possibly resistors) is designed and pre-fabricated on a wafer. These are then connected together by one or two further connecting layers (usually metal) to make gates and other logic.

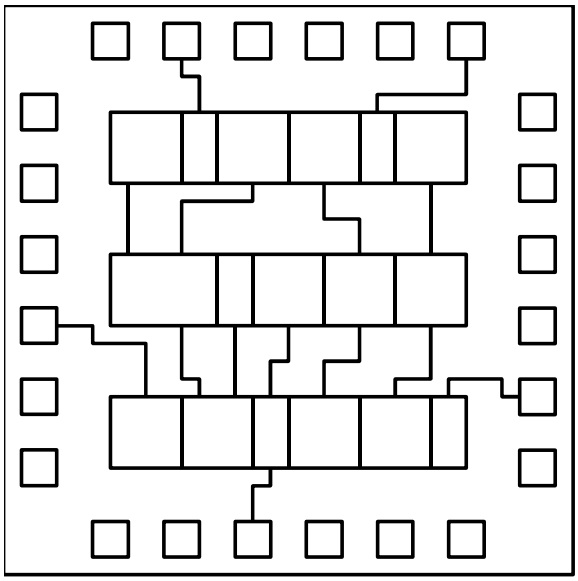
Both techniques provide approaches to design which are amenable to automation.

By contrast to these *application specific integrated circuits* (ASICs), there are also standard function chips such as PLAs and ROMs, which may be programmable after fabrication.

The choice of technology will involve assessment of the overall design time, the capital cost of design and tooling, the recurrent cost per product and the performance achieved.

Standard cell design

In a standard cell system, leaf designs are taken from a previously designed library and assembled (usually in rows) to make a chip.



- Power and ground run horizontally through the cells.

- Inputs, outputs and feed-through connections are presented into the channels.
- A channel router completes the connections using two or more layers of metal and polysilicon, possibly running over the top of the cells.

This can be considered as a structured approach to full custom design – the design and characterisation of the elementary cells can be undertaken before starting to assemble the main circuit.

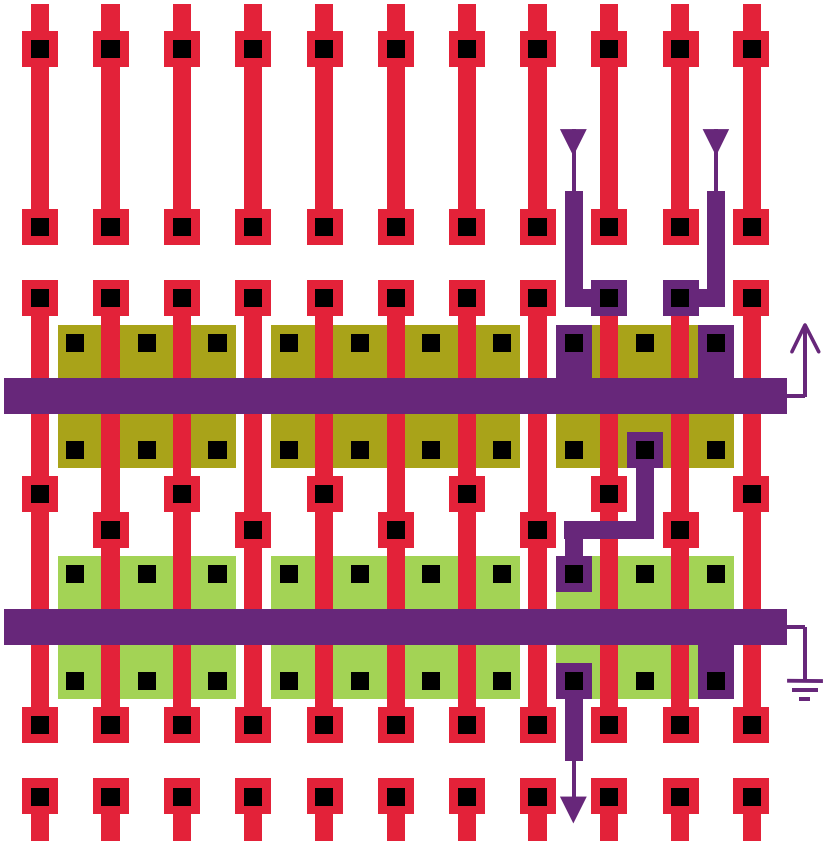
A full set of masks still has to be made for each design.

Gate arays

Gate arrays present a pre-fabricated array of components that are linked by connections in one or two layers of metal, perhaps with selective contacts through to underlying array.

Large arrays with 10^5 gates and 38 kbit static RAM are available, so these can be used for substantial circuits.

A smaller example, the TI TAHC CMOS gate array, presents rows of n-type and p-type transistors with common gates for connection in a single layer of metal. The diagram shows an approximation to the layout with a 2 input NAND gate wired on the right hand side.



A popular alternative is to cover the die with a sea of gates, and use an extra processing stage to open contacts to these where needed, allowing wires to run over the top of unused transistors.

Only metal (and, perhaps, polisilicon) processing is required for gate arrays, making them simple to use for prototypes.

Field programmable gate arrays

Field programmable gate arrays have an array of transistors and a switching network to establish connections between them are made on a chip. The switching network can be programmed through a couple of extra pins when the chip is turned on, to implement an arbitrary circuit on the array.

The Xilinx XC3000 series of *logic cell arrays* (LCAs) present an array on *configurable logic blocks* (CLBs). Each CLB is programmed by 42 control bits to provide a combinational section, essentially a look-up table computing any 2 functions of 4 input signals, and 2 flip-flops. The complete LCA consists of an array of CLBs surrounded by configurable IO buffers, the whole linked by a programmable interconnect including occasional buffers for long signals.

The XC3000, with 64 CLBs and 64 IOs requires a total of 14779 program bits and is deemed to be equivalent to 2000 2-input gates.

The program is loaded serially when the chip is turned on, either from a separate ROM or PROM chip or down-line from a host computer. Clearly, modification of the circuit is extremely, indeed frighteningly, straightforward.

Comparisons

The different approaches can be compared as follows:

	Design time	**Capital cost**	**Recurrent cost**	**Performance**
Full custom	Slow	High	Low	High
Standard cell	Medium	High	Low	Medium/high
Mask programmable gate array	Fast	Medium	Medium	Medium
Electrically programmable gate array	Fast	Low	High	Medium/low

Computer aided design

Electronic computer-aided design can be viewed as manipulating four different descriptions of a circuit:

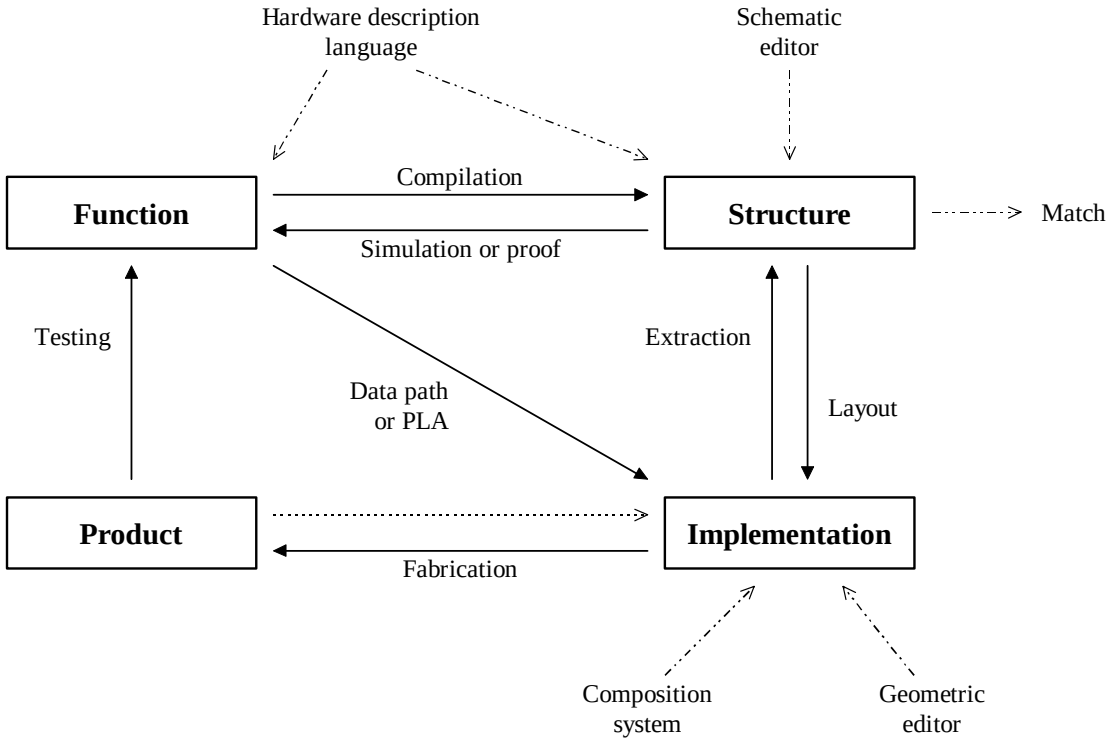
- **behavioural description** - what it does, expressed as an algorithm in something like a programming language,
- **functional description** - how it does it, expressed as an algorithm in something like a register-transfer language,
- **structural description** - how to build it, expressed as a circuit diagram or textual description,
- **implementation** - a physical description of how to make it, expressed as something like a set of masks,

and, of course, the actual product. Each of these will use hierarchy to provide a framework for the description in a way analogous to top-down programming in software. A large design will be stored in many files and may well separate interfaces and implementations (even alternative implementations).

Different tools are used to manipulate these descriptions, affording different qualities of assistance:

- **Synthesis** - automatic translation between descriptions,
- **Constrained construction** - the CAD system restricts manual design, inhibiting faults,
- **Verification** - two descriptions are compared and are shown to be equivalent,
- **Simulation** - one description is investigated under particular circumstances and observed to work in that case.

Ideally, an automatic system would translate automatically between the behavioural description and a product; in practice we accept rather less. The overall framework can be viewed as follows:



Aspects of electronic CAD

The design is entered as one or more of:

- a hardware description language,
- a circuit schematic (essentially a pictorial structural description),
- layout geometry, or
- a composition of primitive geometric elements.

These are then manipulated by programs that either proceed with the design automatically (clockwise round the diagram) or confirm the equivalence of two different descriptions (anti-clockwise).

Silicon compilation

Silicon compilation is in the process of transforming an algorithm expressed in a behavioural HDL into a structural design.

For certain, highly-structured circuit models, such as ROMs, RAMs and PLAs, this is relatively straightforward, as is the subsequent generation of implementation details for a particular process.

This has also been extended to stereotyped processing units such as arithmetic and logical units (ALUs) and in signal processing.

More recent work has allowed the synthesis of control structures as well as data paths for circuits described as algorithms in a programming language. These generate structural descriptions combining standard cells from a library: variables are translated into registers, operations into function blocks such as adders or even complete ALUs, and so on. This usually imposes a clocking strategy on the circuit.

An important issue is allowing the designer to control the trade-off between speed and size in the design. The simplest synthesis might use separate, parallel hardware for each operation; a

more subtle approach would be to share the function block between several operations using multiplexors. This can be visualised by using a *Gantt diagram* to represent the flow of data through the clocked registers in time; this also provides a convenient interface for the designer to rearrange the degrees of parallelism and sharing in the circuit being synthesised.

Simulation

In the absence of more formal analysis techniques, simulation is often used to observe the behaviour of a circuit given its structure. These involve the preparation of a set of stimuli, or *test patterns*, which are applied to the circuit and the resulting state of signals within the circuit displayed, usually graphically as if they were being observed with an oscilloscope.

A number of different simulation tools are useful:

- *Behavioural simulation* – executing the program that is the functional description.
- *Logic simulation* – given functional models of elementary components, and their composition in a structure, exercise the circuit with a set of input stimuli and deduce the aggregate effect of the circuit.
- *Switch-level simulation* – check the logical structure of a MOS circuit, using nominal delays and capacitive effects.
- *Timing verification* – find critical paths through combinational logic and deduce the maximum clock frequency at which it might work.
- *Circuit simulation* – model transistors as analogue devices by considering the solution of simultaneous differential equations.

Special hardware *accelerators* are often used to execute the simulation algorithms at high speed. These may even incorporate real chips as fast simulators of themselves.

Hardware verification involved the formal proof that the aggregate effect of a set of components assembled in a particular structure is the same as a given functional description.

Silicon assembly

The translation of a structural description into an implementation is known as *layout* or *silicon assembly*. It divides conveniently into two related operations:

- *Placement* – arranging the elementary components (gates and transistors or higher level modules) on the chip.
- *Routing* – finding paths for the signal wires between them.

Each of these is applied at different levels in the structural hierarchy, in particular:

- *Globally* – for high level modules in the circuit
- *Locally* – within an elementary module.

Good automatic tools are available for all these operations, especially for stereotyped design such as gate arrays and standard cell layout.

If manual intervention with the design is allowed, it is important that *design rule checking* is performed by the system to ensure that it will actually be possible to fabricate the chip from its physical description. Many layout editors do not allow direct manipulation of masks, but only of artefacts which are synthesised to comply with the design rules.

Even where automatic tools are available, they often need manual assistance to complete the design. In this case it is helpful if the layout editor checks the implementation both for consistency with the structural description and for compliance with design rule. This is known as *correctness by construction*.

When a physical description has been edited directly, it is useful to extract a structural description from it an to match this against the original description as a verification stage. This can also be used to check the correctness of the CAD tools in an automatic system.

A common, if curious, design technique is to enter layout using a geometric editor, extract a structural description and then simulate to determine if the correct function has been achieved.

Testing

After chips have been made, they must be tested (usually before cutting up the wafers and putting the chips in packages). This involves *probing* them with leads that inject test signals and monitor the chips' responses.

These test patterns can usefully be derived from the test data used when the circuit was simulated previously. However, it becomes even more important that the test patterns should exercise the circuit fully.

The probing equipment has to operate at very high speeds if the chips' marginal performance is to be tested. This has resulted in testers that compare a newly made chip against a known *gold chip* which has been characterised previously.

There is also a role for *reverse engineering* – the extraction of a physical description from a product,

Design for test

With the increasing complexity of circuits, it is becoming more important to include additional circuitry to ensure that a system (be it a chip, a board or rack full of equipment) will actually be testable after construction.

This involves assessing the design for *testability* which divides into two factors:

- *Observability* – the ease with which the internal state of the circuit can be checked.
- *Controllability* – the ease with which the internal state can be changed.

These can be measured both statically (in some sense by the distance of latches from the external pins of a chip) and dynamically during simulation. The latter uses a special sort of logic simulator known as a *fault simulator*.

It is useful to check that a set of input stimuli exercises the circuit fully. In particular, any fault in the circuit should give rise to an observable difference on the outputs. This means that the test patterns will be useful for testing the chip in production and also that the logic has been fully exercised during simulation.

The number of possible faults in a circuit is vast. There can be gross faults like breaks in tracks, short circuits, faults in the crystalline structure of the silicon, and also subtle faults like capacitive coupling and increased resistance. A useful approximation is simply to consider the possibility of any signal being stuck at either zero or one. For a circuit with s signals, there will be 3^s - 1 possible sets of faults, which is far too many to consider, so only the $2s$ single faults are considered.

In principle, the correct circuit is simulated once and the results recorded. Each of the $2s$ faulty circuits is then simulated until a discrepancy is observed at the outputs. However, even this process is slow.

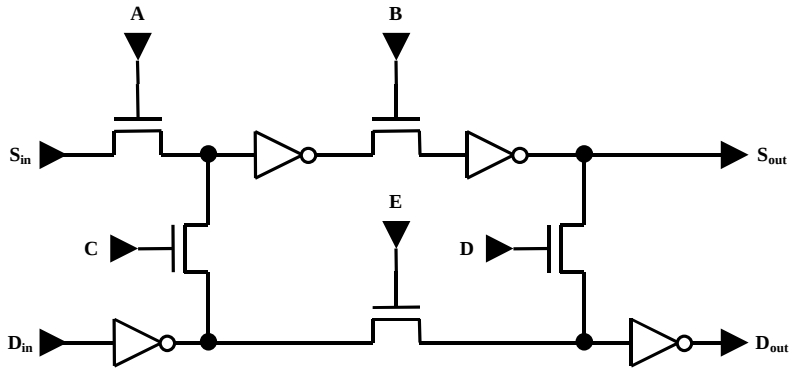
After completing the application of the test patterns, the vast majority (over 95%) of faults should have been exposed in this way. If they have not been observed, then either the test pattern is insufficiently challenging or the design is inherently hard to test. Both of these are bad things.

Automatic test pattern generation (ATPG) attempts to analyse the logic of a circuit and, in particular, the access paths to observe and control internal state, and formulate test stimuli that should give a high fault coverage.

The main difficulty in testability lies in internal state registers, and one solution is to make these easily accessible by connecting them into a long shift register. The chip can then run in a number of modes:

- *Normal* – the registers are disconnected from the shift register.
- *Shift* – the shift register steps along.
- *Read* – the state registers are copied into the shift register.
- *Write* – the shift register is copied into the state registers.

A simple dynamic latch would then be replaced with the following circuit:



which is clocked as follows:

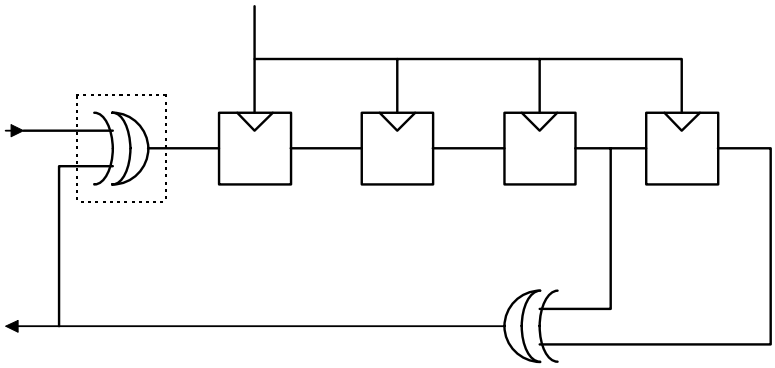
Operation	$\varnothing_1$	$\varnothing_2$
Normal	E	
Shift	A	B
Read	C	
Write	D	

This system, known as *level sensitive scan design* (LSSD) adds as much as 30% to the size of the chip, as well as four extra pins (shift in, shift out and two control). Such scan paths can also be extended to cover whole boards and, indeed, racks of equipment.

Self test

Controlling and observing state using a scan path makes a chip testable, but the testing can be quite slow because the serial access path presents a bottleneck. An alternative scheme is to build additional testing circuitry into the design.

Signature analysis uses a *linear feedback shift register* to generate a pseudo-random test pattern and to compute a characteristic function of an output signal. A register with *n* latches can generate a sequence of length $2^n - 1$.



Linear feedback shift register

The additional XOR gate is only used in the analyzer, where the value left in the register after running an appropriate number of clock cycles is the signature.

An extension of this is *built-in logic block observation* where a pair of control signals allow the block to be configured as:

- a scan path shift register,
- a normal set of system latches copying inputs to outputs,
- a LFSR with multiple inputs, or
- reset to a known state.

These are then placed between blocks of conventional circuitry. During testing, alternate BILBOs are configured as pseudo-random generators and analyzers, and the signatures are then shifted out along the scan path.

Standards

Exchange of data between different CAD systems and for delivering designs to silicon foundaries are facilitated by standard data formats. Some of the important ones are:

- *GDS-2* – a binary mask description widely used in industry
- *Caltech intermediate form* (CIF) – a textual mask description widely used by academics.
- *Electronic design interchage format* (EDIF) – a LISP like language incorporating different *views* of a design as behavioural, structural and physical descriptions.
- *VHSIC HDL* (VHDL(- an Ada-like language for behavioural descriptions. (VHSIC is the US DoD very high speed integrated circuit programme).

Designing a very large scale integrated circuit is an extremely complicated business and all the usual techniques used by computer scientists to tackle problems of scale are applied.

Within a chip, power distribution requires care if sufficient current carrying capacity is to be provided. Interdigitated power and ground lines are the standard solution. Care must also be taken over clock distribution if phases are to be synchronised over a wide area.

On a practical front, it is convenient if prototype chips can be of standard sizes and have similar pin configurations, perhaps even a standard external interface to some micro-processor bus. This leads to the idea of *design frames*, standard arrangements of I/O pads and driver circuitry for chips. This can be extended to standard test boards into which the prototype chips are inserted.

Another prototyping technique is the multi-project chip in which several small designs are fabricated within a single die, sharing some of the external connection. This makes for economies on the costs of overheads such as mask making and bonding.

Self-timed circuits

Early digital circuit designers explored various synchronization mechanisms. Global synchronization (clocked) proved to be faster and use fewer devices (valves/vacuum tubes) than local synchronization (self-timed) counterparts. Design is much simpler when time is quantised and internal state only changes at discrete intervals. Correct operation is ensured by making the clock period slower than the time that combinatorial logic takes to settle.

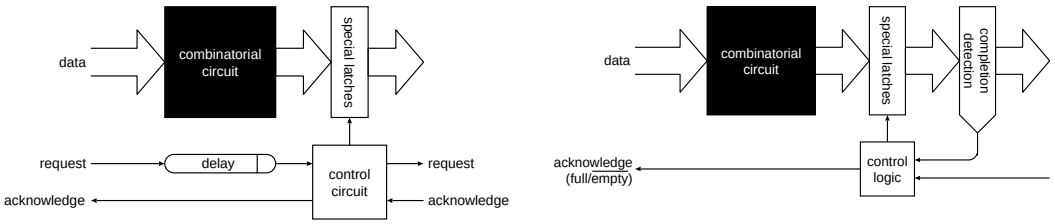
However, advances in technology are making this approach less satisfactory for a number of reasons:

- With deep submicron CMOS, wire delays are becoming more significant than logic delays. Consequently, global synchronization is becoming impractical due to clock skew problems.
- Distributing a fast clock consumes a lot of power and dynamic circuits consume power even when they are not performing useful calculations.
- The maximum clock speed is dictated by the worst case delay in combinational logic, which may only arise with pathological data values.
- Delays on long paths mean that circuits can not be composed and still work at the same clock speed.
- A safety margin on clock speed must be allowed to cater for variations in performance with fabrication, age, temperature and operating voltage.
- A strong clock signal will radiate harmonics that may give rise to electro-magnetic compatibility problems or pose a threat to security.

This is leading to a revival of research in self-timed design techniques that were largely laid to rest in the mid 1960s.

Computing without clocks

There are two principle approaches to self-timing: matched delays (local clocking via delay element) and completion detection (embedding control signals in with the data).



Matched Delays Completion Detection

Matched delays are usually generated via a combination of watching critical paths (e.g. carry propagation for an adder) and additional delay elements (e.g. extra inverters). This approach necessitates careful design and layout if the result is to be fast. Currently this means a good deal of hand placement.

Completion detection is a more “pure” form of self-timed circuit where a completion signal is encoded with the data. Local completion determination (local “timing”) is achieved by detecting (decoding) this completion signal.

Encoding completion signals

Validity can be indicated by using *dual-rail encoding.* Two wires are used to represent every logical bit:

code Q_1Q_0	**meaning**
00	clear
01	logical 0
10	logical 1

This complicates logic slightly. For example, a half adder in conventional, single-rail logic has the following truth table:

A	B	H	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

So

$$H = \sim A.B + A.\sim B$$
$$C = A.B$$

using 6 gates with a maximum delay of 3 gates.

In dual-rail this becomes:

A_1	A_0	B_1	B_0	H_1	H_0	C_1	C_0
0	0	X	X	0	0	0	0
X	X	0	0	0	0	0	0
0	1	0	1	0	1	0	1
0	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1
1	0	1	0	0	1	1	0

So:

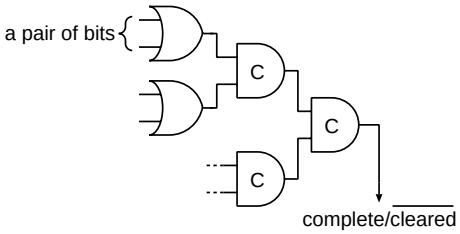
$$H_1 = A_0.B_1 + A_1.B_0$$
$$H_0 = A_0.B_0 + A_1.B_1$$
$$C_1 = A_1.B_1$$
$$C_0 = A_0 + B_0$$

using 8 gates but with a maximum delay of 2 gates.

Note how inversion can be achieved in dual-rail logic simply by swapping wires over.

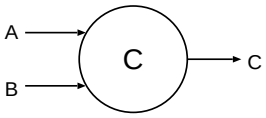
Completion Detection

Completion detection may be achieved by ORing each pair of wires and then using a tree of Muller C-elements spanning all the signal wires (see next section).



Completion Detection Circuit

Muller C-elements



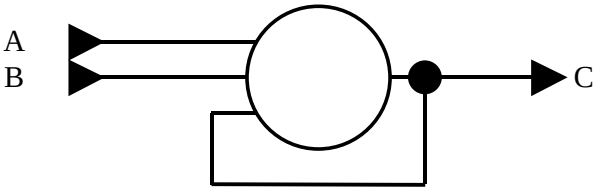
(a) symbol

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

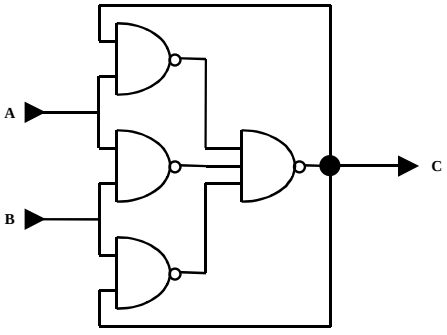
(b) truth table

Classical Muller C-element

The Muller C element is rather like and AND gate with hysteresis, or an AND gate for events. It can be considered as a majority gate with feed-back:

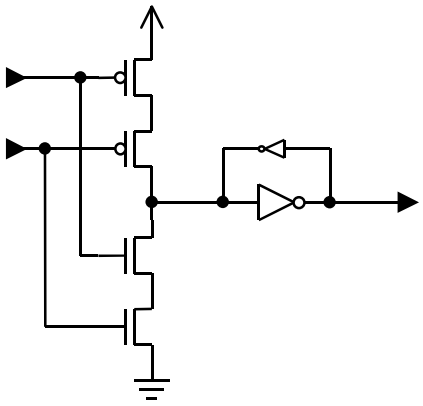


Alternatively it could be implemented in gates:



However, this gives rise worries about correct operation if the delay along the feed-back path is too great.

The following circuit is possible in direct CMOS:

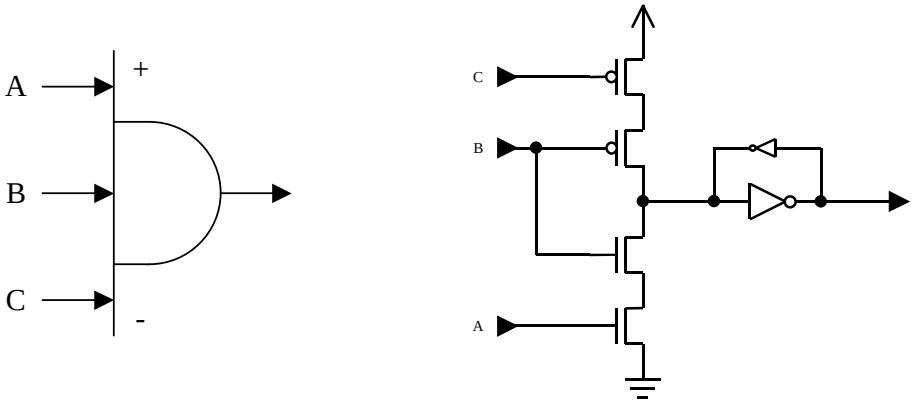


The small invertor provides a weak feed-back to retain state.

A generalised C-element only requires a subset of the signals to be at appropriate levels to switch its output:

A	B	C	Q
X	0	0	0
X	0	1	Q
0	1	X	Q
1	1	X	1

The previous CMOS circuit can be adapted as follows:



Delay models

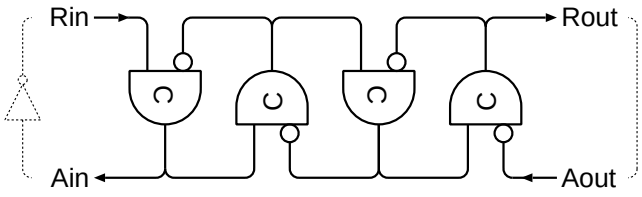
When considering the correctness of a circuit that does not use a global clock, it may be necessary to make assumptions about the implementation. Several different models for the delays in gates and on wires may be used:

- **Fundamental mode** circuits assume upper and lower bounds on gate and wire delays so that outputs settle between changes of the inputs. There will be a minimum delay within which a set of changes to input signals must have occurred and a maximum delay within which the outputs will have settled (before the next input change can occur). These assumptions underpin the correct operation of clocked storage devices such as D-type flip-flops made out of NOR gates.
- **Speed independent** circuits assume that all gate delays are finite (but unbounded) but with no wire delays. Clearly matched delays can not be used and completion detection is necessary.

- **Delay insensitive** circuits assume that both gate and wire delays are finite (but unbounded). This is the most general model but designs are very complicated, essentially only using invertors and C elements.
- **Quasi delay insensitive** circuits broaden the DI model to allow *isochronic forks*, separate paths carrying the same signal where the difference in delays on two paths is less than a gate delay.
- **Field forks** are a special arrangement in MOS where a signal can control a sequence of transistors by running polysilicon across their gates. It can then be assumed that the transistors will switch in the same sequence.

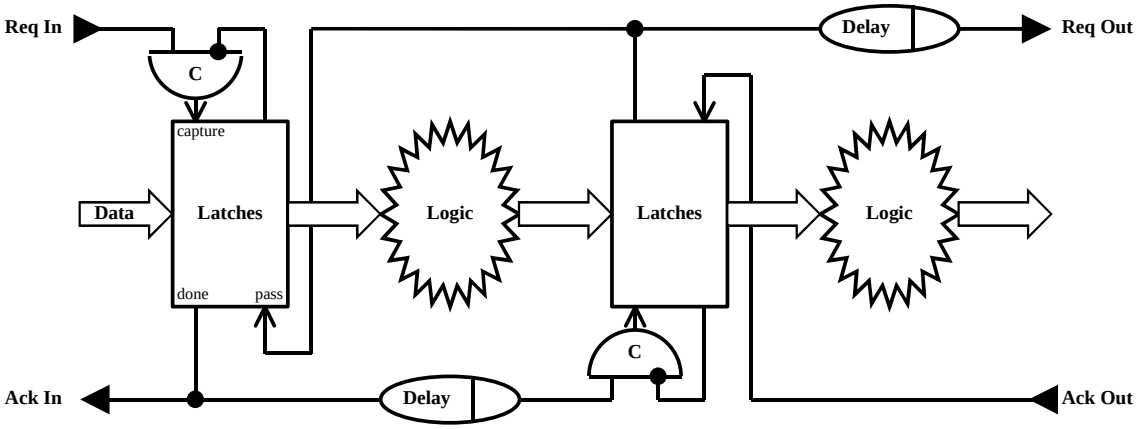
Event FIFO

The circuit is depicted below. Events are edges (both positive and negative) with the environment response indicated by the dotted inverter/wire.



Self-timed Event FIFO

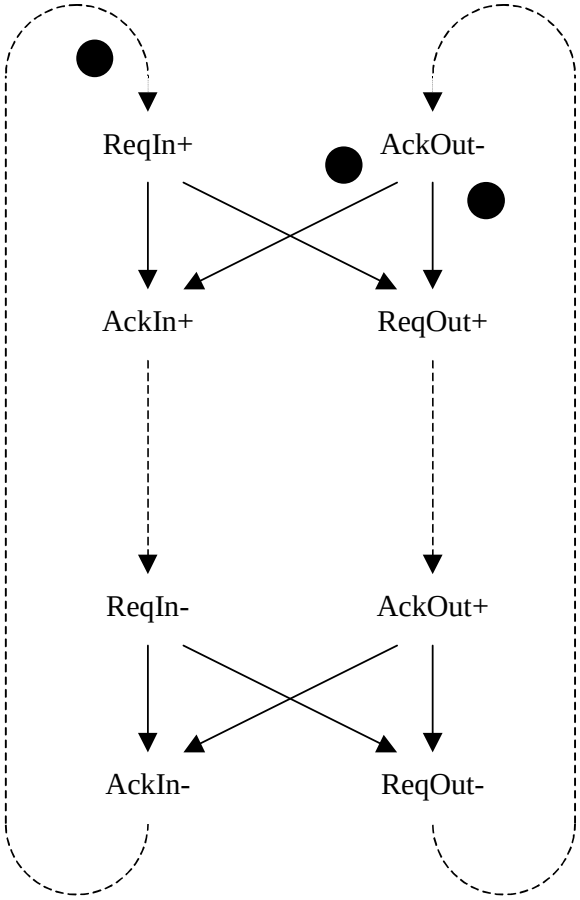
This can be used to create self-timed *micro-pipelines* by interposing latches controlled by the events that store values on a data bus feeding through blocks of combinational logic:



This is used in the Amulet, the asynchronous version of the ARM processor developed at the University of Manchester.

Signal Transition Graphs

Signal Transition Graphs (STGs) are used to specify the ordering of positive and negative edges. This is actually a form of Petri Net but without the places. The STG for one stage of an event FIFO is depicted in the next figure. The dotted lines represent the response from the environment (effect of output to input changes) and solid lines represent internal response (effect of input to output changes).



STG for one stage of an event FIFO

Challenges for ECAD

- Synthesizing self-timed handshake circuits (good tools are becoming available).
- Constraining wiring delays and routing to ensure that control signals arrive in the desired order (current floor planning tools can help constrain placement and new deep submicron CMOS tools allow bounds on wire delays to be specified).

Lecture review form

If high lecturing standards are to be maintained and lower standards to be raised, it is important for lecturers to receive feedback about their lectures. Consequently, we would be grateful if you would complete this questionnaire and return it to the Student Administration Office. A digest of the information will be passed to the Staff-Student Liaison and Teaching Committees.

VLSI Design
Michaelmas 2001

Please tick the boxes below:

Interest

Tedious	Uninteresting	Interesting	Exciting

Level of material

Much too basic	Too basic	Slightly basic	About right	Slightly complicated	Too complicated	Much too complicated

Breadth of coverage

Much too general	Too general	Slightly general	About right	Slightly specific	Too specific	Much too specific

Organisation of lectures

Chaotic	Confused	Adequate	Brilliant

Assumptions

Assumed too little	About right	Assumed too much

Ease of understanding

Incomprehensible	Confused	Adequate	Clear

Speed

Much too slow	Too slow	Slightly slow	About right	Slightly fast	Too fast	Much too fast

Delivery

Incoherent	Halting	Adequate	Fluent

Notes

Poor	Adequate	Excellent

Classes

Poor	Adequate	Excellent

Supervision

Poor	Adequate	Excellent

PTO

Best thing about the course

Worst thing about the course

Further comments