3BA4: VLSI Design

PDF version of the lecture notes by Diarmuid Power and Edsko de Vries

May 12, 2003

Contents

List of Figures

Chapter 1

Chip Economics

The design rules tell us not to design too small. Why should we not design big? It makes the design easier and cheaper. Let's examine the cost versus design size. The cost to produce a wafer is relatively constant.

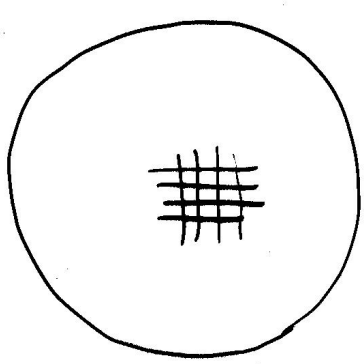


Figure 1.1: Wafer

We want to know the cost c of a working chip. Let A be the area of the wafer (see figure 1.1). Let the area of the chip be a. Then the number of chips $n = \left\lfloor \frac{A}{a} \right\rfloor \approx \frac{A}{a}$. However, $c \neq \frac{C}{n}$: the number w of working chips is less than n: $w < n$ because of manufacturing failure due to random variation or dust; a hair is $\approx 50\mu$m thick!. Hence the need for "clean rooms".

There are complex statistical fault models for analysing the error rate, but we use a very simple model: we assume there are f faults per wafer, and each kills one chip. Then $w = n - f$ (where f is dependent on the manufacturing process). Then the yield $y = \frac{w}{n} \times 100\%$. Then $c = \frac{C}{w} = \frac{100C}{yn}$ ($w = \frac{yn}{100}$). Hence cost $\propto \frac{1}{\text{yield}}$. If we express c as a function of a, we get $c = \frac{C}{w} = \frac{C}{n-f} = \frac{C}{\frac{A}{a} - f} = C \cdot \frac{a}{A - af}$ (compare this to the naive form $c = C\frac{a}{A}$, or $f = 0$). If af is small, the bottom line is approximately A; if $af \to A$, then the bottom line $\to 0$ and $c \to +\infty$.

So there is a pressure to reduce a, but a smaller device size makes the manufacturing harder. Let s denote the minimum feature size (measure of design rule "size"), and let X denote the chip complexity (the number of features needed). Then $a = Xs^2$. But f depends strongly on s, as s fails, f rises.

So we need to balance f vs. s at the optimum point (where $f''(s) = 0$).

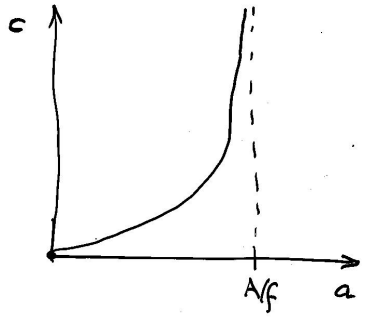


Figure 1.2: Cost vs. chip area

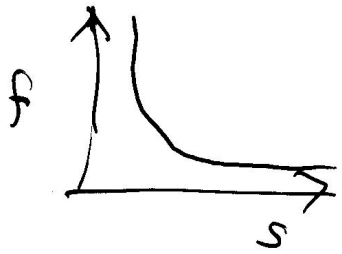


Figure 1.3: Frequency vs. feature size

Chapter 2

Current Density

2.1 Switch / Stick / Layout Diagrams

Help to produce real MOSFET circuits but we need some appreciation of the electricelectronic issues involved.

2.1.1 Current Density

Current flowing per unit cross section area of conductor.

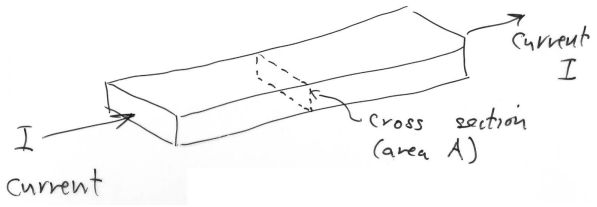


Figure 2.1: Cross Section

Current Density $J = \frac{I}{A}$, but there is a problem, J rises as the technology shrinks. Relevant parameters are current (I) and the current density (J) which specifies how much flow is being squeezed through a given area. Current is motion of charge carriers (electrons) Higher Current Density (J) leads to more, faster electrons moving through the conductor.

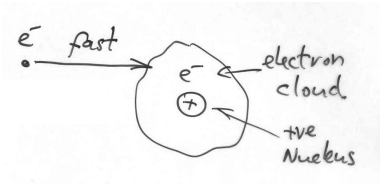


Figure 2.2: Metal Atom

Electrons can knock atoms out of position, if fast enough. So a high Current Density leads to what is termed as Metal Migration.

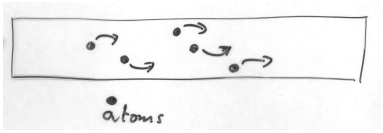


Figure 2.3: High J

In aluminium (most common IC metal), migration occurs when $J = \frac{2mA}{\mu m^2}$ So we establish a safety margin $J_{MAX} = \frac{1mA}{\mu m^2}$. We need to know the current consumption of our circuits. and a guideline as to what metal width can handle what current.

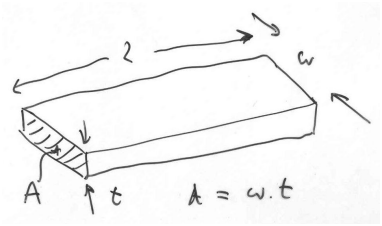


Figure 2.4: Real Wire

We design the length (l) and width (w) specifically and the thick (t) is determined by the manufacturer. So $J = \frac{I}{A} = \frac{I}{W}.\frac{1}{t}$ where $\frac{1}{t}$ is the scale factor. Maximum densities are given as $\frac{mA}{\mu m}$

Conductors "go with the flow"! In other words, a conductor is a material that allows charge carriers to pass through. Real conductors are irregular.

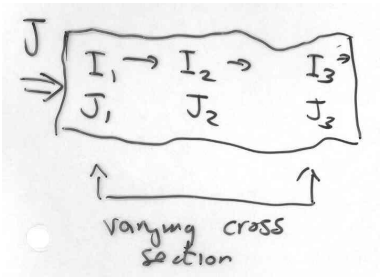


Figure 2.5: Varying Cross Section

The current density (J) is highest for a fixed current (I), where the the area is lowest. In other words, the current density is highest at the thinnest part of conductor. $I_1 = I_2 = I_3, J_1 \neq J_2 \neq J_3$. The highest rate of migration occurs at the thinnest part of the conductor making it thinner still and further increasing the current density. See 2.6, 2.7 and 2.8.

The solution is to ensure that the wires are thick enough so that migration does not occur to begin with. What about contacts like Metal to Diffusion cuts, is current density an issue? The answer is yes. See 2.9.

A Dip can form where metal covers contact cut hole. 2.10

Thinnest part of contact are the walls of metal down the side of the cut. The main restriction of a cut's current carrying capacity is the cut perimeter2.11.

Typical limit: $0.1\frac{mA}{\mu m}$ of perimeter.

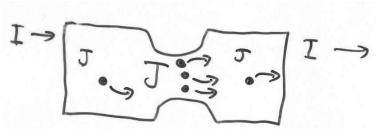


Figure 2.6: Migration 1

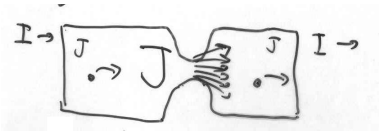


Figure 2.7: Migration 2

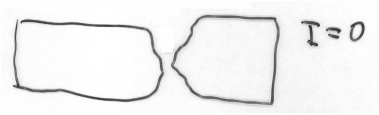


Figure 2.8: Migration 3

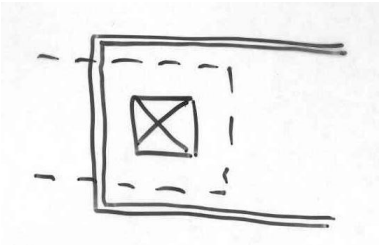


Figure 2.9: Contact Cut

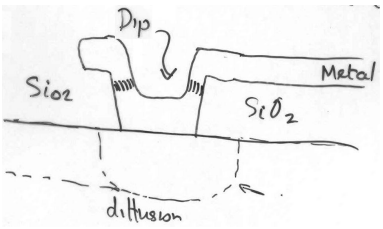


Figure 2.10: Dip in the Cut

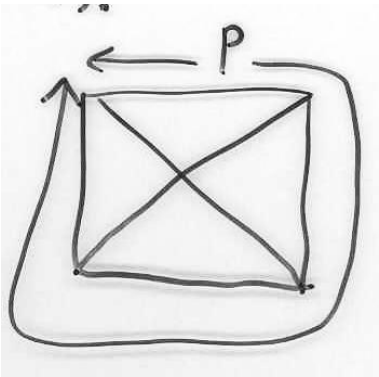


Figure 2.11: Cut Perimeter

Chapter 3

Resistance

3.1 Performance of ICs

IC performance includes the speed of operation and the power consumption by the IC. We must look at the layout as electrical material (figure 3.1 and 3.2). Wires have resistance and capacitance (and inductance). We look at the resistance and capacitance of the different layout shapes and devices.

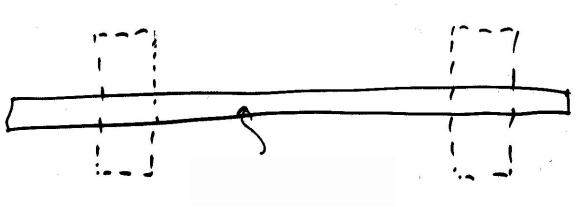


Figure 3.1: Wire with two transistors (layout)

Figure 3.2: Wire with two transistors (circuit)

3.2 Resistance

$R = \left(\frac{\rho}{A}\right) l$ where $A = wt$ (the cross-section area), and ρ is the resistivity of the material. We can rewrite R as $\left(\frac{\rho}{t}\right)\left(\frac{l}{w}\right)$. The first term $\left(\frac{\rho}{t}\right)$ is dependent on the IC process, the second term $\left(\frac{l}{w}\right)$ is determined by the design.

Define the sheet resistance $R_s = \frac{\rho}{t}$. Then $R = R_s \frac{l}{w}$ (with units Ω/unit area), where $\frac{l}{w}$ is the number of "squares" in the wire (figure 3.4).

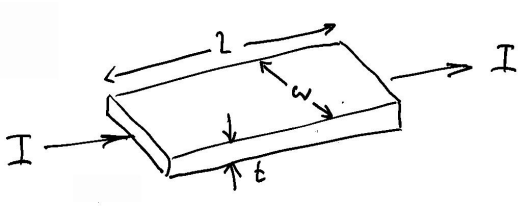


Figure 3.3: Wire dimensions

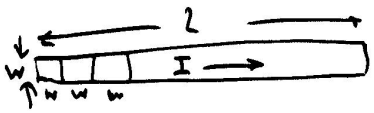


Figure 3.4: Wire as a series of “wire squares”

Typical R_s values are 0.07Ω for metal, 20Ω for polysilicon and 25Ω up to 100Ω for diffusion (the resistance of diffusion varies greatly, even within one manufacturing process). Therefore metal is better than polysilicon for long wires.

3.3 Unusual shapes

Consider figure 3.5.

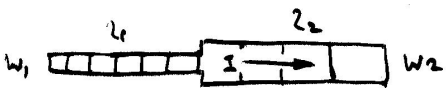


Figure 3.5: Jump in wire width

$R_1 = R_s \frac{l_1}{w_1}$ (number of squares of size w_1) and $R_2 = R_s \frac{l_2}{w_2}$ (number of squares of size w_2). Then $R = R_1 + R_2 +$ (correction for jump in width).

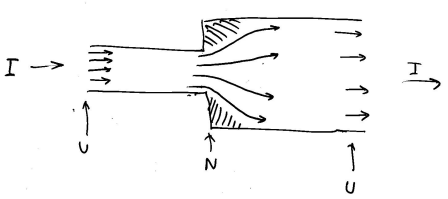


Figure 3.6: Non-uniform spread of current

At U there is a uniform current density across the wire, however at N there is a non-uniform spread of current. We need a correction to allow for this (though in this case this correction is fairly small, and depends on the ratio $\frac{w_1}{w_2}$).

We also need a correction for corners (figure 3.7).

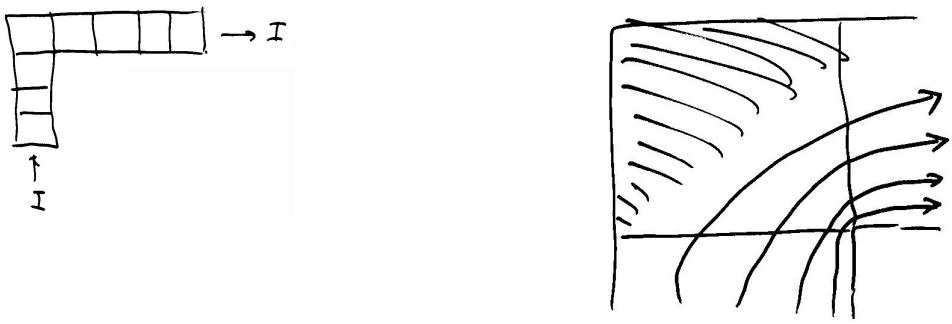


Figure 3.7: Current “around the corner”

However, there are no simple formulae that describe these corrections. We either need complex numerical analysis or exhaustive tests of real samples. However, the rough rule of thumb for corners (figure 3.8) is that a square corner ($w_1 = w_2$) acts as a $\frac{1}{2}$ square resistance. This is useful for hand estimation of wire resistance, though in practise resistances are extracted by software from design data (eg. Mentor Graphics or IC Extract).

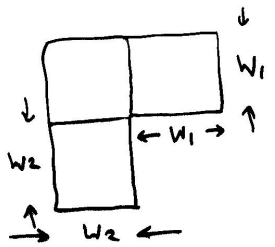


Figure 3.8: Square corner acts as 0.5 square resistance

Chapter 4

Capacitance

In a parallel plate capacitor (figure 4.1), the capacitance $C = \varepsilon_r \varepsilon_0 \frac{A}{t}$, where A is the area of the plates, t is the thickness of the dielectric, ε_0 the permittivity of the vacuum and ε_r the relative permittivity of the dielectric. For SiO_2, $\varepsilon_{SiO_2} = 3.9$, and for Si, $\varepsilon_{Si} = 12$.

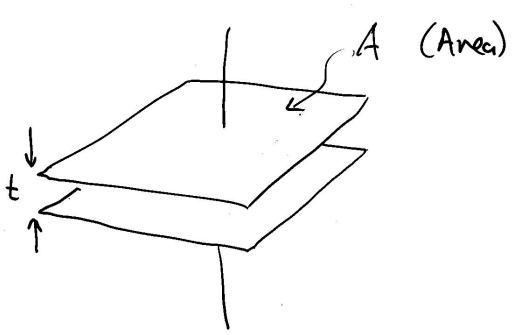


Figure 4.1: Parallel Plate Capacitor

In CMOS, the capacitor looks like (figure 4.2). Unfortunately, capacitance in ICs is a little complicated.

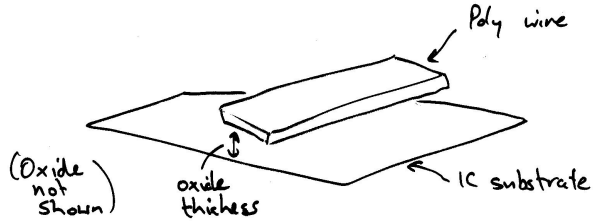


Figure 4.2: Capacitor in CMOS

The problem is that voltages on the conductor (either metal for a Metal / Oxide / Semiconductor or polysilicon for a Polysilicon / Oxide / Semiconductor capacitor) produce electric fields through oxide into the semiconductor. This causes changes in the charge carrier density in the semiconductor. This has a knock-on effect on the capacitor.

Initially there are lots of (positive) holes in the semiconductor (mobile charge carriers). $V_c = 0$ and $C_0 = \frac{\varepsilon_{SiO_2} \varepsilon_0 A}{t_{oxide}}$ (figure

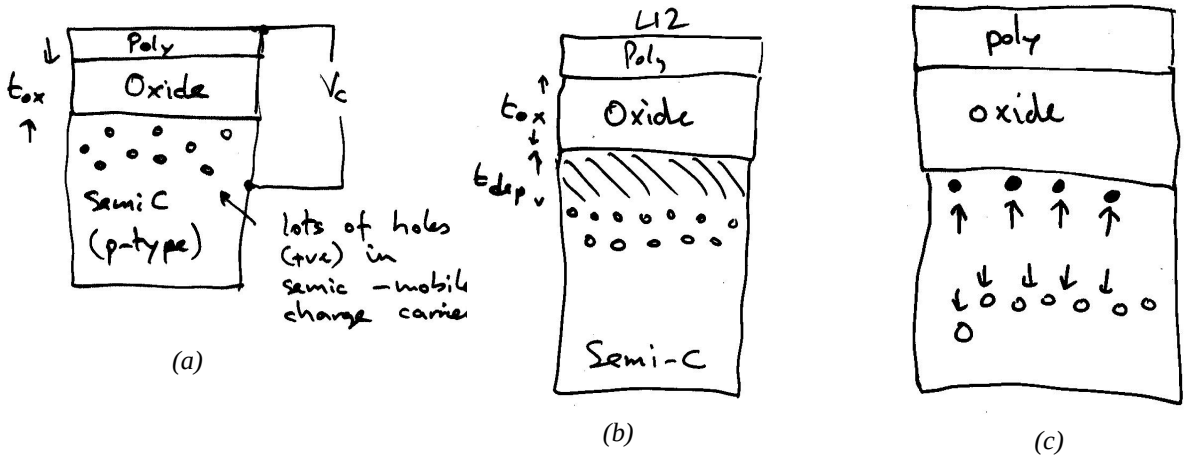


Figure 4.3: Depletion region under polysilicon

4.3*a*). Now consider an increase in V_c. The $+V_e$ voltage repels holes in the substrate and produces a depletion region under the polysilicon (4.3*b*). The capacitance is now $C_0 + C_{\text{dep}}$ in series.

$$C_{\text{dep}} = \varepsilon_{\text{Si}} \frac{\varepsilon_0 A}{t_{\text{dep}}},\ C = C_0 \,||\, C_{\text{dep}} = \frac{C_0 C_{\text{dep}}}{C_0 + C_{\text{dep}}} \tag{4.1}$$

So the capacitance drops as V_c rises (initially). **Capacitance is voltage dependent**.

As V_c gets higher again, it passes a threshold value, where the holes have been pushed away and the electrons are being attracted up. The p-type semiconductor now becomes n-type (inversion, figure 4.3*c*). The substrate is now conducting again, and C is back to C_0.

Capacitance drops as the depletion occurs. It then rises again as inversion takes place, except if the voltage varies very rapidly, in which case the electrons cannot move fast enough and inversion does *not* occur (figure 4.4).

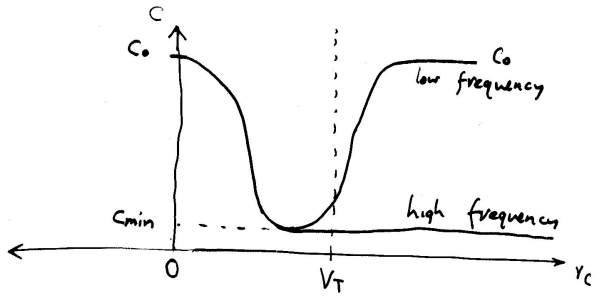


Figure 4.4: Capacitance vs. frequency; V_T is the threshold voltage, $\frac{C_{min}}{C_0} = 0.02 \sim 0.3$

V_T depends on oxide thickness. For the thin oxide layer (the oxide under the polysilicon that *crosses* the diffusion layer), we have $V_T \approx \frac{V_{DD}}{5}$ (by design). For the thick oxide (oxide under metal or polysilicon, but *not* crossing diffusion), we have $V_T \gg V_{DD}$ (again, by design). Over thick oxide, voltages are $\ll V_T$ so C_0 is a good approximation. Over thin oxide, voltages go from $0 - V_{DD}$, mostly $> V_T$, so C_0 is a good approximation (for low frequencies).

The worst case are the highest capacitances, so take C_0 as a conservative (pessimistic) estimate of the capacitance.

$$C = C_{\text{ox}}A = \frac{\varepsilon_{\text{SiO}_2}\varepsilon_0 A}{t_{\text{ox}}} \tag{4.2}$$

The MOS capacitance structure covers polysilicon and metal wiring. The diffusion needs a separate treatment. In the device, there are many reversed biased PN junctions (figure 4.5).

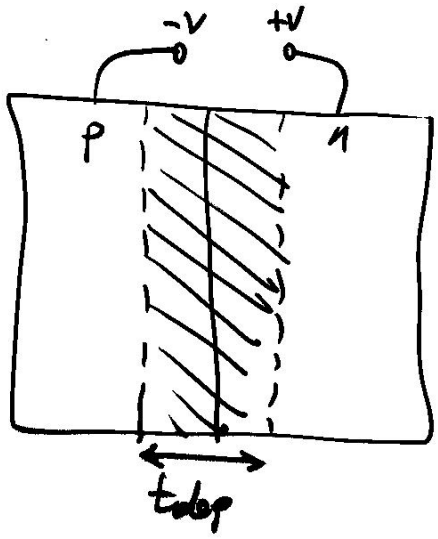


Figure 4.5: A reversed biased PN junction; t_{dep} depends on voltage and doping concentrations

Let C_{j_0} be the capacitance per unit area when the bias voltage is zero, and let C_j be the capacitance per unit area at a given voltage. Then (empirically)

$$C_j = C_{j_0} \cdot \left(1 - \frac{V_j}{V_b}\right)^{-m} \tag{4.3}$$

where $\frac{V_j}{V_b}$ is the junction voltage, m is $0.3 \sim 0.5$ (depending on doping concentrations and profile), and V_b is the junction bias voltage (turn-on) ≈ 0.6V. We shall simply take $C_j = C_{j_0}$ for simplicity. Note however that the diffusion layer is not simply a 2D surface, but a box, with side-walls (figure 4.6) which are part of the PN junction area. The sidewall depth is process dependent, and cannot be changed by the designer. The sidewall capacitance $\propto$ perimeter and the sidewall depth. We thus introduce C_{jp}, the capacitance per unit perimeter, and C_{ja}, the capacitance per unit area.

$$C = C_{ja}wl + C_{jp}(2w + 2l) \tag{4.4}$$

Where w and l and the width and length of the diffusion layer respectively. Some typical values (in pF/μm^2):

	n	p
C_{ja}	$2 \cdot 10^{-4}$	$5 \cdot 10^{-4}$
C_{jp}	$4 \cdot 10^{-4}$	$4 \cdot 10^{04}$

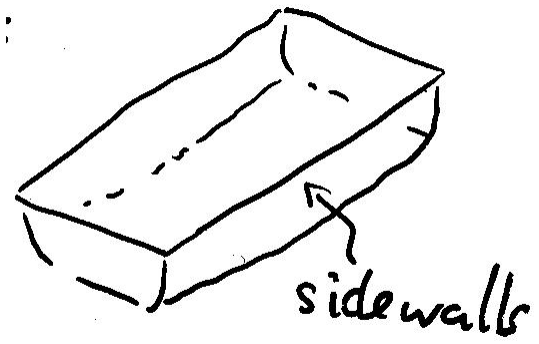


Figure 4.6: Actual diffusion: embedded rectangular box

Chapter 5

Long Thin Wires

Consider a long, thin wire of metal of polysilicon.

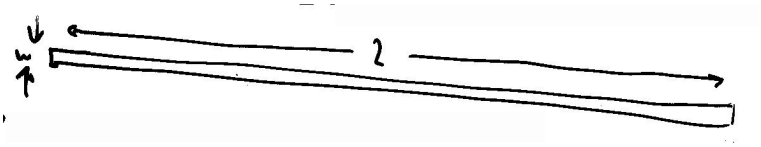


Figure 5.1: Long thin wire of Metal/Poly

Let r be the resistance per unit length. The total resistance $R = R_s \frac{l}{w}$ (with R_s the resistance per unit square), so $r = \frac{R_s}{w}$.
Let c be the capacitance per unit length. $C = C_{\text{ox}}A = C_{\text{ox}}wl$. $c = \frac{C}{l} = C_{\text{ox}}w$.
So, the total resistance $R = rl$, and the total capacitance $C = cl$.

5.1 The Lumped RC-model

A wire can be regarded as an RC circuit (figure 5.2).

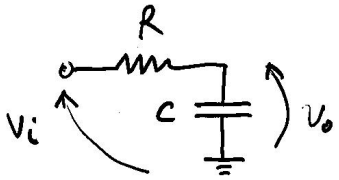


Figure 5.2: $RC = rcl^2$ (time constant)

In starting conditions ($t = 0$), both v_i and v_0 are zero. Now assume the input $v_i \to V$ at $t = 0$ (figure 5.3). Then $v_0 = V(1 - e^{-(t/RC)})$.

When $t = RC$, we see that v_0 has risen to $\left(1 - \frac{1}{c}\right) \approx 69\%$. In practice the rise and fall times measure the time it takes for the voltage to rise from 10% to 90% (or vice versa), see figure 5.4.

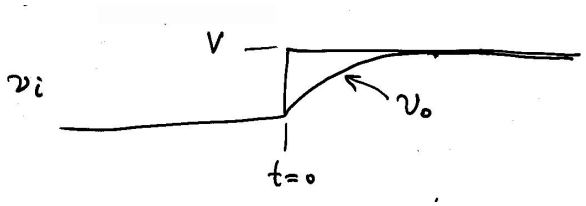


Figure 5.3: Time versus voltage, $v_i \to V$ at $t = 0$

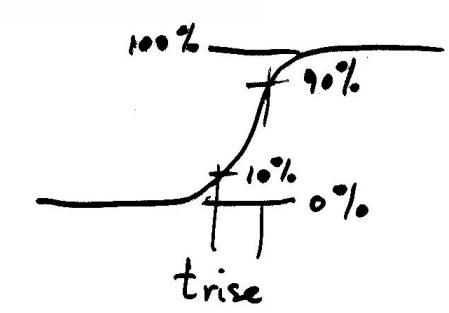


Figure 5.4: The rise time $t_{rise} \approx 2.2RC$

5.2 The Distributed RC Model

A long line can be considered as a series of RC circuits (figure 5.5).

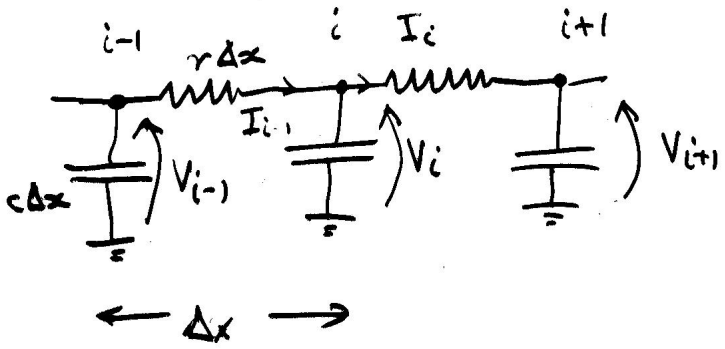


Figure 5.5: A wire as a series of RC circuits

$I_i{}^c = I_{i-1} - I_i$ (see figure 5.6). $I_i = \dfrac{V_i - V_{i+1}}{r\Delta x}$, $I_{i-1} = \dfrac{V_{i-1} - V_i}{r\Delta x}$. $Q = CV$ and $I = C\dfrac{\mathrm{d}V}{\mathrm{d}t}$, so

$$I_i{}^c = C\frac{\mathrm{d}V_i}{\mathrm{d}t} = \frac{V_{i-1} - V_i}{r\Delta x} - \frac{V_i - V_{i+1}}{r\Delta x} \tag{5.1}$$

We have a discrete form (recurrence relation). Now let $\Delta x \to 0$:

$$c\frac{\mathrm{d}v}{\mathrm{d}t} = \frac{1}{v} \cdot \frac{\mathrm{d}^2 v}{\mathrm{d}x^2} \tag{5.2}$$

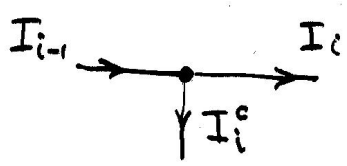


Figure 5.6: Kirchoff's Current Law: $-I_{i-1}+I_i+I_i=0$

If we set up a step input as before, we find

$$v(x,t) \qquad \begin{cases} v(x,0)=0 \\ v(0,t)=V, \quad t\geq 0 \end{cases} \tag{5.3}$$

We find the rise time at $x \propto x^2$. So the time to rise at the end of a wire with length l is $\propto l^2$, and we get $RC = rcl^2$. However, we found $t_{\text{rise}} \approx RC$. The distributed wire has a rise time for the far end of about half of that calculated using the lumped approximation—the lumped model is too conservative by a factor of 2.

So in summary, the lumped models are easy, but pessimistic approximations. Long lines are bad for business, as delay rises with the square of the length: avoid very long signals lines in designs.

Chapter 6

Dense Logic

How do we analyse gate switches times? FETs are like switches, but their "resistance" varies as V_{in}, V_{out} go from 0 to 1 and vice versa. There are several approaches: a crude, RC-like analysis, a detailed dynamic circuit analysis or the use of circuit simulators such as SPICE.

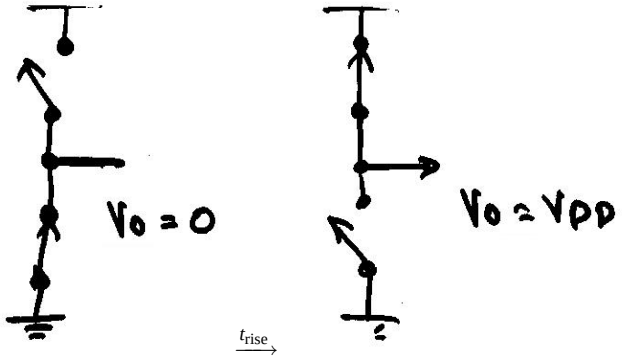


Figure 6.1: Rise time of an inverter

Consider an inverter pair (figure 6.2).

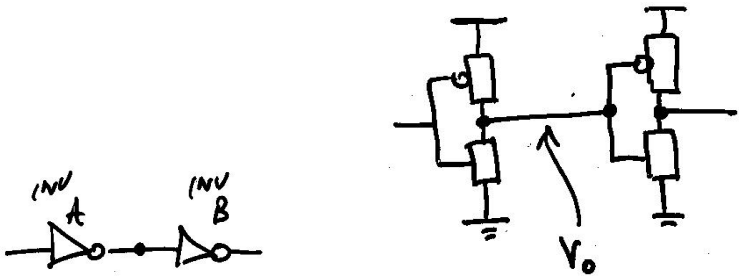
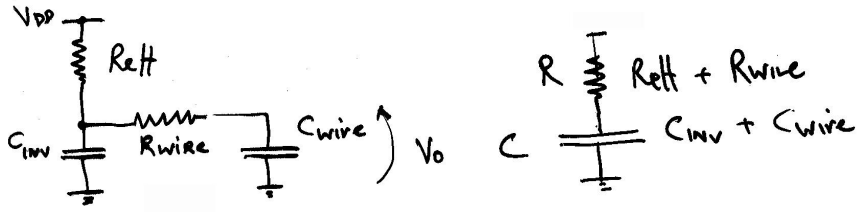


Figure 6.2: An inverter pair

We have the resistance and capacitance of the wiring of the output from the first inverter A to the input of the second

inverter B (easy enough) plus the resistance and capacitance of the switches (complicated).



a *b* $t_{\text{rise}} \approx 2.2RC$

Figure 6.3: Simplified view of a transistor during output rise

We assume that during the output rise the pull-up is closed (ON) with a constant resistance (R_{eff}) and the pull-down is open (OFF), with an infinite resistance. This is shown in figure 6.3*a*, with a simplified picture in figure 6.3*b*. How does R_{eff} compare to and R_{wire}, and C_{inv} to C_{wire}? Let R_{wire} be the R/unit area of the wiring material (25 Ω/unit area for diffusion, 20 Ω/unit area for polysilicon, 0.07 Ω/unit area for metal and 5–25 Ω/unit area for Vias (contact cuts)).

We can estimate R_{eff} as the ratio $\dfrac{V_{\text{ds}}}{i_{\text{ds}}}$ when $V_{\text{out}} = V_{\text{in}} = V_{\text{inv}}$ (which is equal to V_{DD} if the inverter is balanced).

If we analyse the n-type device (figure 6.4), we find that saturation occurs when $V_{\text{GS}} > V_T$ and $V_{\text{DS}} > V_{\text{GS}} - V_{\text{T}}$. $V_{\text{in}} = V_{\text{out}} = \frac{1}{2}V_{\text{DD}}$, $V_{\text{GS}} = \frac{1}{2}V_{\text{DD}} > V_{\text{T}}$, $V_{\text{DS}} = \frac{1}{2}V_{\text{DD}} > \frac{1}{2}V_{\text{DD}} - V_T$, so we indeed have saturation. Then, if we assume $V_{\text{T}} = 1\text{V}$ and $V_{\text{DD}} = 5\text{V}$, we get

$$\begin{aligned} i_{\text{DS}} &= \frac{\beta_n}{2}(V_{\text{GS}} - V_{\text{T}})^2 \\ &= \frac{\beta_n}{2}\left(\frac{V_{\text{DD}}}{2} - V_{\text{T}}\right)^2 \\ &= \frac{\beta_n}{2}(2.5-1)^2 \\ &= \frac{\beta_n}{2}\cdot 2.25 \\ i_{\text{DS}} &\approx \beta_n \end{aligned} \tag{6.1}$$

$R_{\text{eff}} = \dfrac{V_{\text{DS}}}{i_{\text{DS}}} = \dfrac{\frac{1}{2}V_{\text{DD}}}{\beta_n} = \dfrac{2.5}{\beta_n}$. Typically $\beta_n \approx 80\ \mu\text{A/V}^2$, so $R_{\text{eff}} \approx \dfrac{2.5}{8\cdot 10^{-6}} = \dfrac{2.5}{8}\cdot 10^{+6} \approx 2.5\cdot 10^5 \approx 10 - 100\ \text{k}\Omega$.

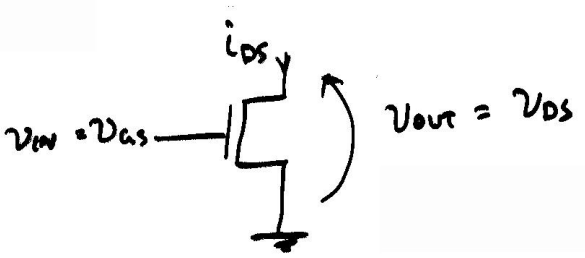


Figure 6.4: n-type device

So we notice that $R_{\text{eff}} \gg R_{\text{wire}}$ (in compact logic). We therefore ignore the wiring resistance in compact logic for a first approximation. $C_{\text{gate}} \approx 25\cdot 10^{-4}\ \text{pF}/\mu\text{m}^2$.

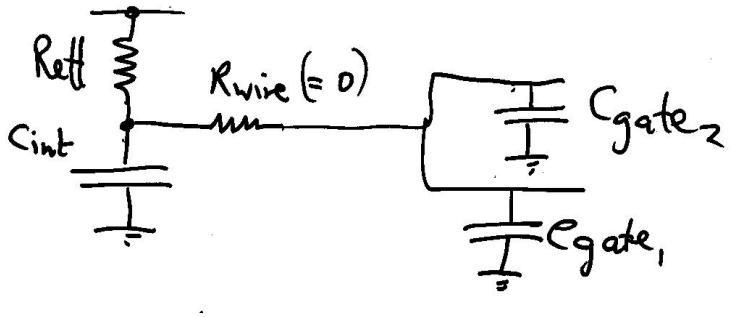


Figure 6.5: Breakdown of the capacitances

$C_{\text{poly}} \approx 0.5 \cdot 10^{-4}$ pF/μm^2, C_{int} consists of the source and drain diffusions, the metal bridge used to connect the pull-up and pull-down networks, and the polysilicon wire to the next gate (approximately $3 + 0.5$ pF). C_{gate} on the other hand is the capacitance of the polysilicon over thin oxide, and is approximately 25 pF. We conclude that we can simplify the picture as done in figure 6.6.

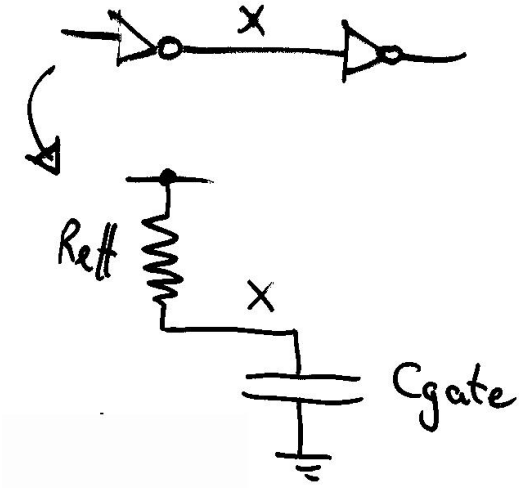


Figure 6.6: Simplification assuming compact logic; $t_{\text{rise}} \approx 2.2 R_{\text{eff}} C_{\text{gate}}$

$R_{\text{eff}} \propto \dfrac{1}{\beta_n}$. $\beta_n = k_n \cdot \dfrac{W_n}{L_n}$ (R_{eff} is approximately 80 μA/V^2 if and only if $W_n = L_n$). So $R_{\text{eff}} \propto L$ (which is usually fixed and minimal) and $R_{\text{eff}} \propto \dfrac{1}{W}$ (wider transistors have less effective resistance).

For a p-device, we have $\beta_p = \frac{1}{3}\beta_n$. We therefore make the p transistors 3 times wider than the n transistors to compensate, and ensure that R_{eff} of the pull-up network equals R_{eff} of the pull-down network.

Let R_0 be the R_{eff} of a minimum width (W_{min}) n transistor (typically $R_0 \approx 10$ kΩ). Then for an n-device of width W, we get $R_{\text{eff,n}} = R_0 \cdot \dfrac{W_{\text{min}}}{W}$, and for a p-device, we get $R_{\text{eff,p}} = 3 \cdot R_0 \cdot \dfrac{W_{\text{min}}}{W}$. So, for an effective resistance of R_0, we have $W_n = 1$ and $W_p = 3$. For an effective resistance of $\frac{1}{4}R_0$, we have $W_n = 4$ and $W_p = 12$.

The total gate capacitance = $(W_p L_p + W_n L_n) \cdot$ (capacitance per unit area C_0). So, noting that $L_n = L_p = L_{\text{min}}$ and $W_p = 3W_n$,

we get $C_{\text{gate}} = C_0(W_pL_p + W_nL_n) = C_0(W_p + Wn)L_{\min} = C_0 \cdot 4 \cdot W_n \cdot L_{\min}$.

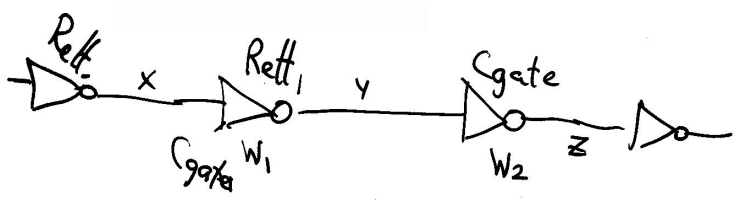


Figure 6.7: String of inverters; $C_{\text{gate}} \propto W_n$

We could speed up transitions on Y by increasing W_i (decreasing R_{eff_i}). However, C_{gate_i} will increase, slowing down transitions on X. We conclude that compact logic should be minimum sized. This generalises from inverters to arbitrary logic gates.

Chapter 7

Large Loads

How do we handle large loads, such as long wires or large external loads? Consider a large "off-chip" load (figure 7.1). We want to drive this efficiently. $t_{\text{rise}} \approx 2.2 R_{\text{eff}} C_L$ (with C_L large, approximately 100 to 1000 times the typical C_{gate}).

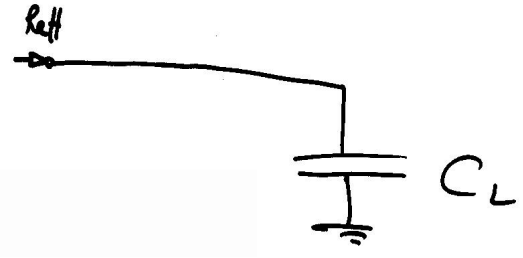


Figure 7.1: Large off-chip load

By increasing the width of the inverter ($W_n \gg W_{\min}$), we reduce $R_{\text{eff}} \ll R_0$ and we get a decent rise time. So, we use large transistors to drive large loads.

However, the big "driver" itself now becomes a big load to the internal circuitry. To solve this problem, we use a cascade of inverters, each one getting bigger by some fixed ratio (figure 7.2).

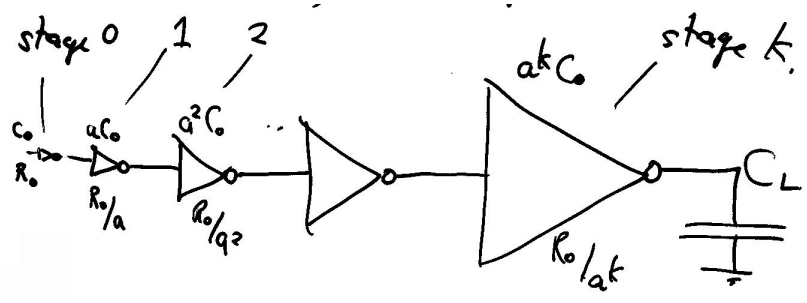


Figure 7.2: Inverter cascade to drive large loads

For gate j, $W = a^j W_0$, $C_{\text{gate}} = a^j C_0$ and $R_{\text{eff}} = \dfrac{R_0}{a^j}$. Then for two cascaded inverters:

$$\begin{aligned} t_{\text{rise}} &= 2.2 \cdot R_{\text{eff}} \cdot C_{\text{gate}} \\ &= 2.2 \cdot \frac{R_0}{a^j} \cdot a^{j+1} C_0 \\ &= 2.2 \cdot a \cdot R_0 \cdot C_0 \\ &= t_{\text{rise}_0} \cdot a \end{aligned} \tag{7.1}$$

Where a is the stage delay and $t_{\text{rise}_0} = 2.2 R_0 C_0$, the rise time minimum sized inverters driving each other. Then

$$\text{total delay} = \text{stage delay} \cdot \text{number of stages } (k+1) \tag{7.2}$$

More precise:

$$\begin{aligned} \ln C_L &= (k+1) \ln a + \ln C_0 \\ (k+1) &= \frac{\ln C_L - \ln C_0}{\ln a} \\ &= \frac{\ln \left(\frac{C_L}{C_0} \right)}{\ln a} \end{aligned} \tag{7.3}$$

So we get

$$\text{total delay} = a \cdot t_{\text{rise}_0} \cdot \frac{\ln \left(\frac{C_L}{C_0} \right)}{\ln a} \tag{7.4}$$

$t_{\text{del}} = \ln \left(\frac{C_L}{C_0} \right) t_{\text{text}_0} \cdot \frac{a}{\ln a}$ is minimised when $a = e \approx 2.71818 \cdots$ (figure 7.3), although it is probably easier (in an exam!) to use $a = 2$ or $a = 3$.

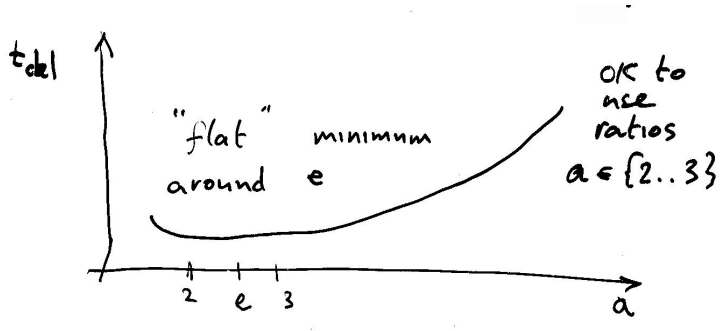


Figure 7.3: t_{del} vs. a, minimum at $a = e$

Chapter 8

Driving Long Wires

In the internal logic, where everything is closely coupled, we keep everything at minimum size. For driving large or external loads, we build a chain of increasing size (ideally with ratio *e*, see also chapter 7, *Large Loads*). However, there are also large internal loads due to long wiring, for example the bus lines in the CPU (fig 8.1).

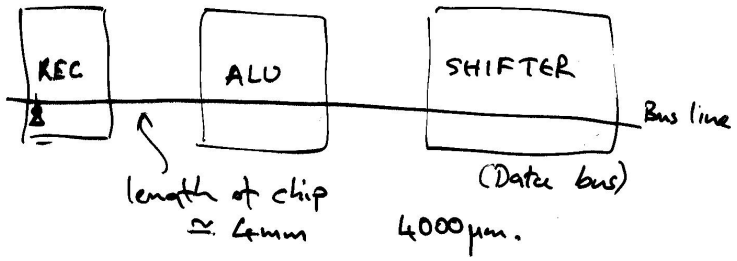


Figure 8.1: Bus line in a CPU

The delay of a long wire is approximately rcl^2 (chapter 5, *Long Thin Wires*). How do we speed things up? Should we treat is as a large load (figure 8.2). The inverter-chain will take up lots of chip space. Besides, the wire might have several drivers (in a multiplexed bus).

Figure 8.2: Long wire as large load.

A better solution is to split the line into segments with buffers (or inverters) in between them (figure 8.3).

Say we divide the line into 3 segments, and let the delay of the buffers t_{buf}. Then the delay of each stage is $rc(\frac{1}{3}l)^2 = rc\frac{l^2}{9}$, so the total delay is $3t_{\text{buf}} + 3rc\frac{l^2}{9} = 3t_{\text{buf}} + \frac{rcl^2}{3}$ (the reduced long line delay). In general, for k stages:

$$\text{delay} = k \cdot t_{\text{buf}} + \frac{rcl^2}{k} \tag{8.1}$$

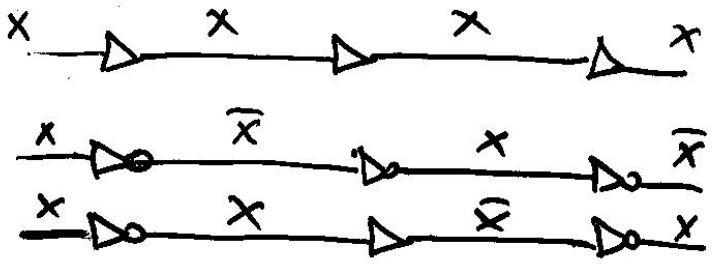


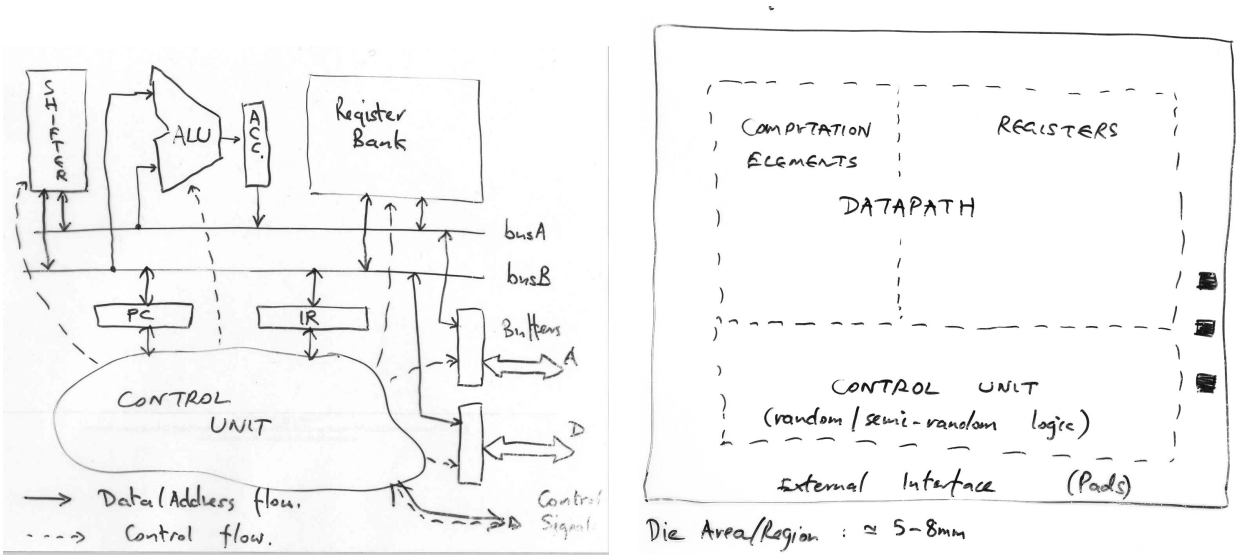
Figure 8.3: Long wire segmented with buffers or inverters

The optimum value for k is when $t_{\text{buf}} = rc \cdot \frac{l^2}{k^2}$, i.e. when the delay of a single line segment equals that of the buffer, we get the best results.

Consider driving a long polysilicon wire. For polysilicon, $r = 20\Omega/\mu$m and $c = 0.05$ fF/μm. For a minimum size inverter, the gate area is 4 μm^2, the gate capacitance C_g is 0.2 fF and the switch resistance R_{eff} is 10 kΩ. Then $t_{\text{inv}} = 2.2 \cdot R_{\text{eff}} \cdot C_g = 2.2 \cdot 10 \cdot 10^3 \cdot 0.2 \cdot 10^{-18} = 4.4 \cdot 10^{-014} = 0.04$ pS. So we choose the length of the segments in the polysilicon wire so that $rcl^2 = 0.04$ pS. I.e., $l^2 = \frac{0.04}{20 \cdot 0.05 \cdot 10^{-18}} = 4 \cdot 10^{16}$ or $l = 2 \cdot 10^8 \mu$m = 200m. (These may not be very realistic values...)

Chapter 9

CPU Floorplanning



a *b*

Figure 9.1: Schematic and IC designers' view of a basic microprocessor

Consider a basic microprocessor CPU (figure 9.1*a*). This CPU has a number of components:

- Computation elements (ALU, shifter)
- Memory elements (registers)
- Communication elements (bus A, bus B)
- Control element (control unit)
- External interface (A, D, control signals)

(Real CPUs would also have caches, an FPU, an MMU, etc.). This, however, is only a schematic view. From an IC designers point of view, figure 9.1*b* is more accurate.

We design that datapath such that every element in the datapath has the same height, with the bus wires running *through* it (figure 9.2). The effect of building the communication wiring into the cell design is illustrated in figure 9.3.

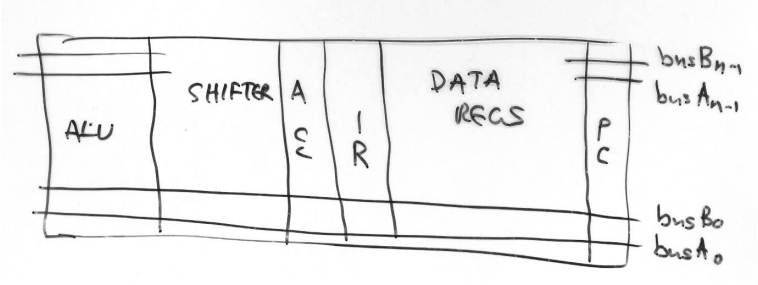
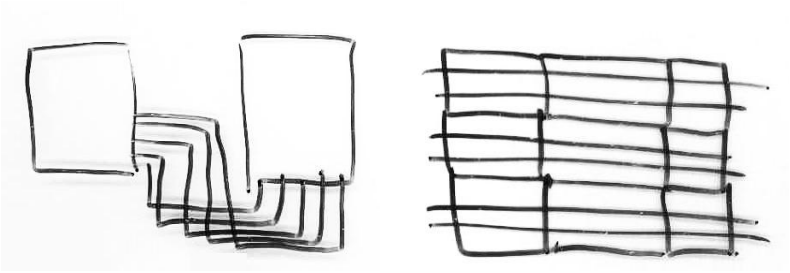


Figure 9.2: *n*-bit data path



Non-uniform wiring Uniform wiring

Figure 9.3: Non-uniform versus uniform wiring

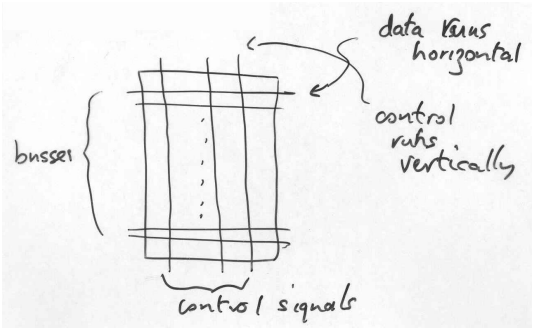


Figure 9.4: Data runs horizontally, control runs vertically

The control signals are fed into the units from above and below (figure 9.4).

Having a good top-level layout strategy (floorplan) pays dividends. This data and control floorplan (with data going horizontally and control going vertically) works very well.

Note that since data busses can be very long, they would typically run on metal. In modern chips with many metal layers, they run in metal (1). Control lines tend to be shorter and run on polysilicon.

Chapter 10

Transmission Gates

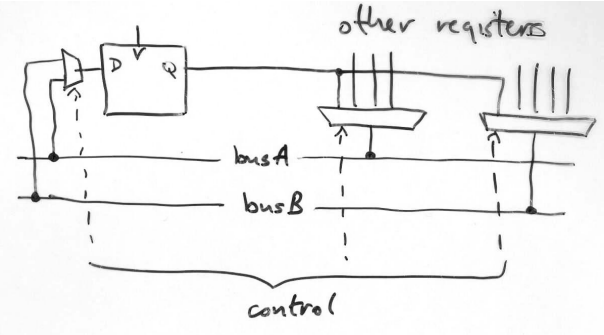


Figure 10.1: Register attached to 2 busses

Consider figure 10.1. Multiplexing is *not* done explicitely as indicated in this figure (it is too expensive). Usually an implicit approach is taken, for example tristate outputs. How do we implement tristate outputs?

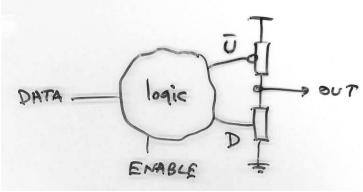


Figure 10.2: Naive implementation of tristate output

A simple solution is shown in figure 10.2. We arrange the logic so that if ENABLE is inactive, both D and $\overline{U}$ (the signals that pull the output Down and Up respectively) are inactive (i.e. both switches are open), and nothing is driving OUT.

However, this requires a minimum of 6 transistors. We can reduce this to two transistors (figure 10.3). Why do we need 2 transistors? Remember

- n-type switches (normally open) are bad at transferring a logic ‘1’
- p-type switches (normally closed) are bad at transferring a logic ‘0’

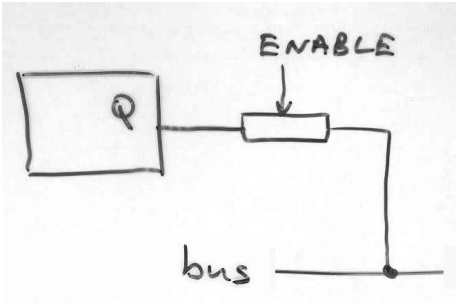


Figure 10.3: Simple solution for tristate output

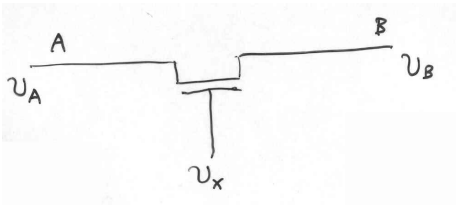


Figure 10.4: Switch; $0 \leq V_a, V_x, V_b \leq V_{\mathrm{DD}}$

Consider figure 10.4. We use V_x to control the path between A and B. If $V_x = 0$, there is no path, and if $V_x = V_{\mathrm{DD}}$, there is a path. This switch is *bidirectional*.

if	$V_a = V_b$	no current
	$V_a > V_b$	A is the drain, current flows from A to B
	$V_a < V_b$	B is the drain, current flows from B to A

Now assume that $V_a = V_{\mathrm{DD}}$ and $V_b = 0$ (A is the drain). Let V_x go from 0 to V_{DD}. $V_{\mathrm{DS}} = V_a - V_b$ and $V_{\mathrm{GS}} = V_x - V_b$.

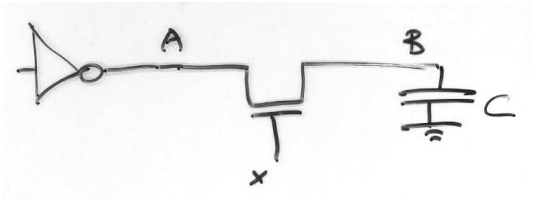


Figure 10.5: B is floating and has a capacitance

B's parasitic capacitance (shown in figure 10.5 as C) is charged up through the MOSFET (figure 10.6). V_x jumps to V_{DD}, the transistor is cut off and V_b reaches $V_{\mathrm{DD}} - V_T$ as its steady state.

No problem occurs if $V_a = 0$ and $V_b \to 0$, as V_{GS} gets larger. In this case B is the drain, A is the source, and V_{GS} is $V_x - V_a$. **So, n-type transistors are bad at transmitting logic '1'.** A similar analysis shows that p-types are bad at transmitting logic '0'.

The solution is to use both (figure 10.7). For this to work we need both the control signal and its inverse—twice as much control wiring. However, this isn't as big a disadvantage as it may seem. Most signals in ICs come from some sort of register, which generally have both outputs Q and $\overline{Q}$.

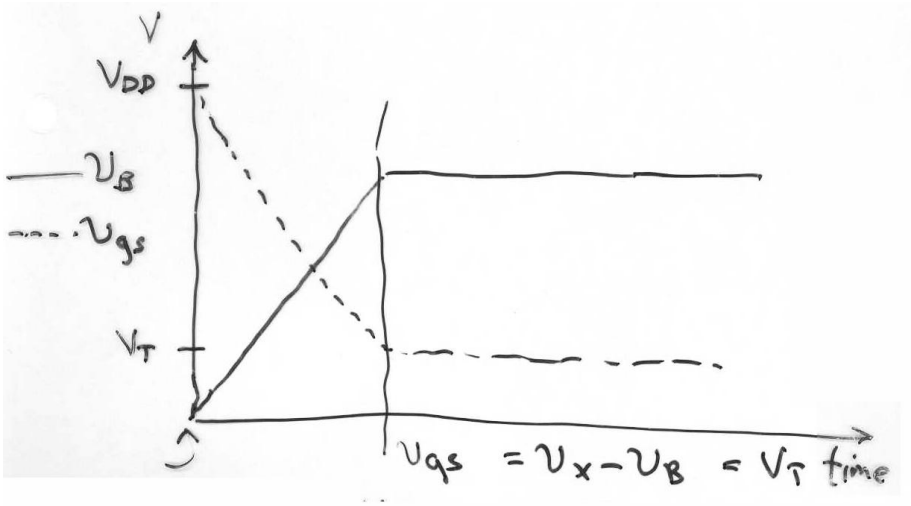


Figure 10.6: Charging parasitic capacitance

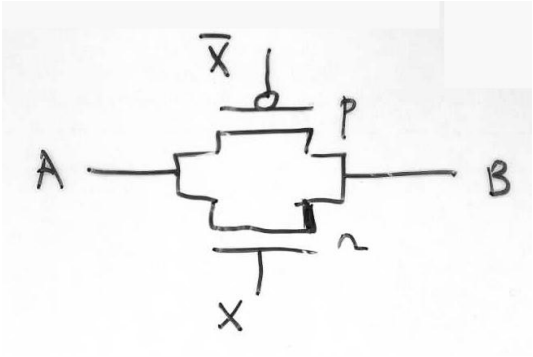
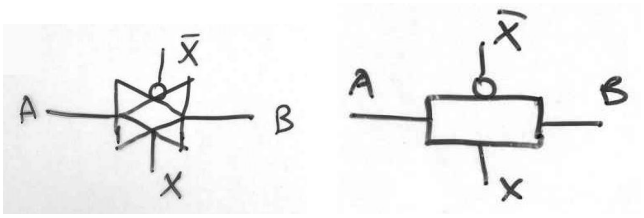


Figure 10.7: Transmission gate using two transistors

The structure above is very common and is called a **transmission gate** (or TX gate). There are two systems for a tranmission gate (figure 10.8*a* and 10.8*b*).



a *b*

Figure 10.8: Transmission gate symbols

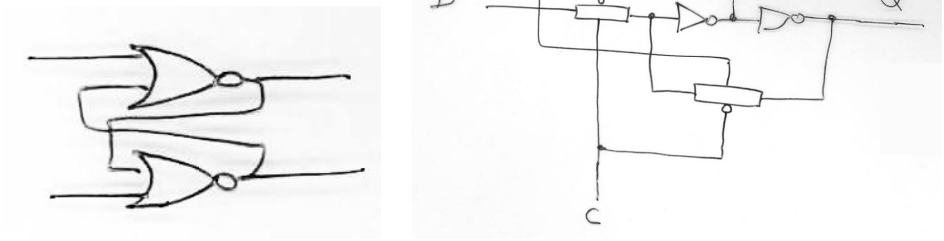
If we are really stuck for space, we can use a single transistor transmission gate (figure 10.4). However, we must not feed either side to the control input of another such gate (figure 10.9).

Figure 10.9: Illegal: output of single transistor TX gate into input of another TX gate

Chapter 11

Transmission Gate Usage

11.1 Unusual uses of transmission gates



(a) *(b)*

Figure 11.1: Memory element as R–S flip-flop

An example of this would be a memory element. The discrete logic approach would be an R-S flip-flop with 8 transistors the layout as shown in figures 11.1*a* and *b*.

So with 2 inverters and 2 TX gates requiring 2 MOSFETs each means a total of 8 MOSFETs. So when C = 0 and $\overline{C}$ = 1 a feedback loop is formed meaning that the previous input is retained and in the other case (C = 1, $\overline{C}$ = 0) the input is passed through. This is known as a level-triggered data latch (transparent latch).

11.2 Other approaches to logic

Random logic of control unit FSMs

During the 70s and 80s the controls path was LARGE, the proportion of silicon real estate was greater than 50% and was often the speed bottleneck. Then in the late 80s and 90s RISC technology was developed and this simplified the control path considerably and brought the proportion down to 10%. So chips became faster.

How can we implement lots of random logic effectively? The answer involves an optimum mix of design effort, compact design and fast results along with other factors.

11.3 Arrays for logic

If we break some rules it is possible to achieve this:

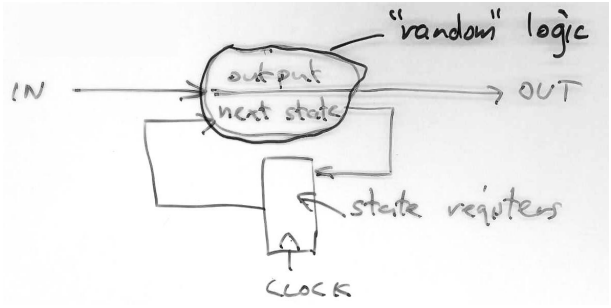


Figure 11.2: Finite State Machine

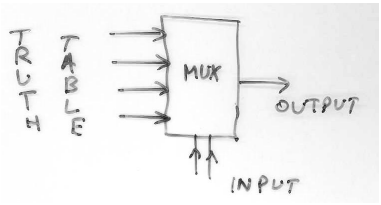


Figure 11.3: Truth tables

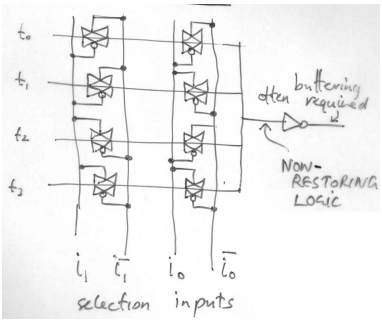


Figure 11.4: Selector Array

This design is a bit inflexible and has a problem that it doesn't scale well, as the number of rows is $2^{\text{number of inputs}}$ and so it is O ($n2^n$) which is not good. So we can construct such an array by simply applying the right configuration the array. An example array is given in figure 11.5.

So when the array contains a 0 the gate is closed when i = 0 (figure 11.6*a*), and when the array contains a 1 the gate is closed when i = 1 (11.6*b*).

From this we can do any two input function easily (exor, and, or ...) and it is much more compact than "standard" CMOS.

11.4 Stick diagram for EXOR gate

11.4.1 TX-gate

The TX-gate converts to this form when put into a stick diagram.

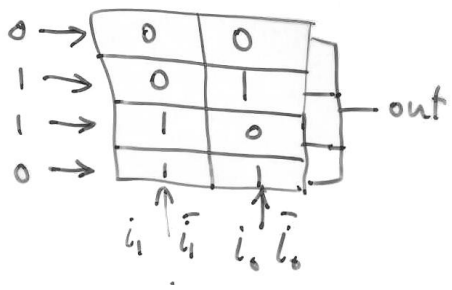
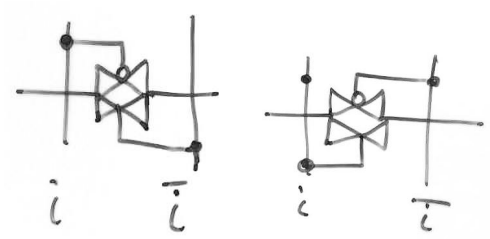


Figure 11.5: Truth table for XOR function



(a) *(b)*

Figure 11.6: Closed on zero / Closed on one

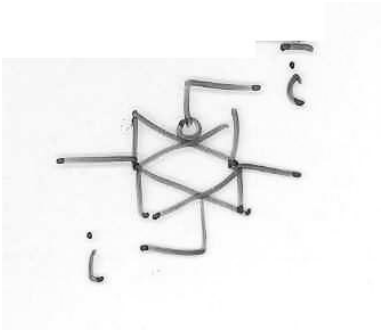


Figure 11.7: Simple form

11.4.2 1_{st} Attempt

This has lots of nWell boundaries.

11.4.3 2_{nd} Attempt

We will merge the nWells.

Figure 11.8: Logic form

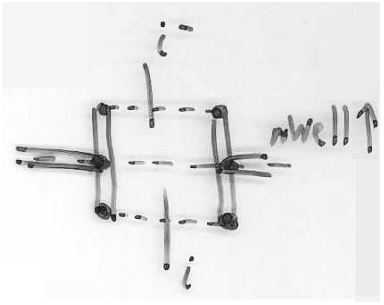


Figure 11.9: Stick diagram

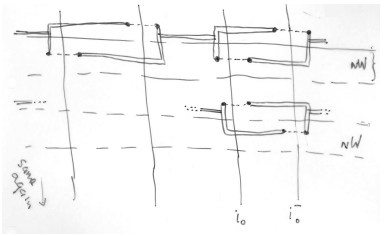


Figure 11.10: Attempt 2

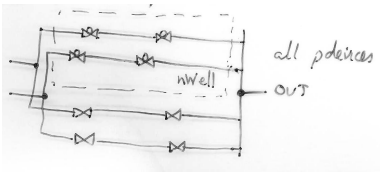


Figure 11.11: Attempt 2

11.4.4 Conclusions

So the selector forms a compact array but it needs buffering afterwards and gets expensive for many inputs O ($n2^n$). There is lots of redundancy for simple functions. Four input NAND still needs $n \times 2^n \times 2 = 2^7 = 128$ which is not great as the same gate only needed 8 transistors old style. On the plus side it is easy for computers to generate as it has a regular structure and straightforward selection of specific connections. What we would like would be a programmable structure with minimal redundancy and much better ”speed performance”

Chapter 12

Programmable Logic Arrays

PLAs are a regular AND array and an OR array with the structure shown in figure 12.1 and 12.2

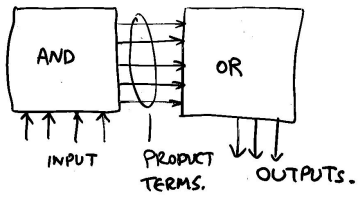


Figure 12.1: PLA Block Diagram

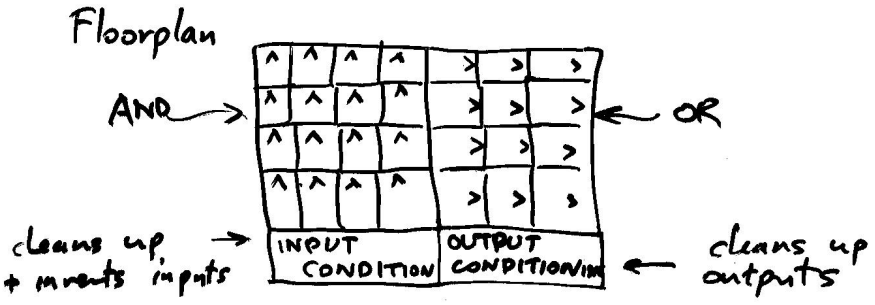


Figure 12.2: PLA floorplan

These arrays of AND and OR are made from identical repeating units/tiles. It is important to point out the De-Morgans laws have an important role to play in PLA design (NOR-NOR). Also it is efficient to enter both X and its its complement on the input tile. Also the input tile may well buffer the input for the rest of the circuit.

How does the tile work to make a NOR-gate? We assume ”pseudo-NMOS”.

This design has the advantages that there are fewer p devices. But also some disadvantages as the static currents is drained when output is low i.e. p device is still on.

The tile can have two constructions, there can be a transistor present as in the figure 12.5 or without one. This is how the PLA is programmed by burning the transistors at certain points to give you the equation that you want.

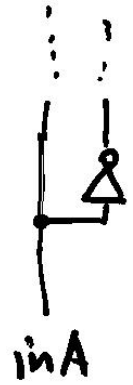


Figure 12.3: Input tile

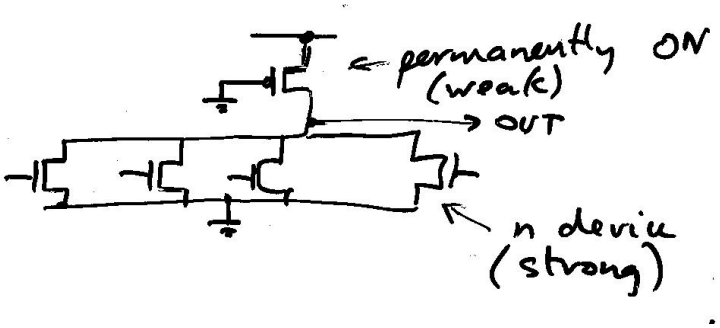


Figure 12.4: NOR gate

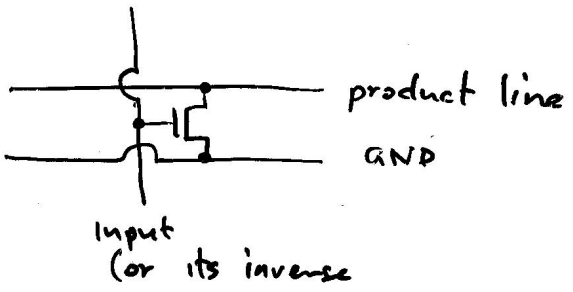


Figure 12.5: PLA with transistor present

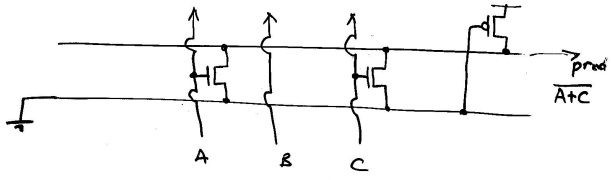


Figure 12.6: A product example

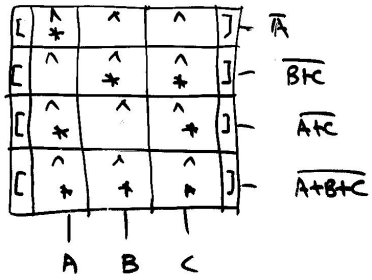


Figure 12.7: Symbolic overview

An example of this in operation would be 12.6 and 12.7

Two of these arrays can give you any logic function of the inputs. They form a regular structure with pre-designed transistors and the sizes of the arrays can be computed by machine readable equations that can make the entire process automated. The algorithm of construction of such arrays is simple, we just turn the desired equation into minterm form (OR of the ANDing of variables and inverses) e.g. $A\bar{B}C + \bar{A}BC + \cdots$ So when implementing this with CMOS technology we choose to use NAND as they are better than NOR as they NAND pull-ups in parallel. But the cost of implementing links and product lines is high and we want a single product line with devices in parallel. In the good old days there was the nMOS which worked in one of two modes, enhancement where $V_T > 0$ and depletion (always on) when $V_T < 0$. Figure 12.8 shows a nMOS in depletion mode.

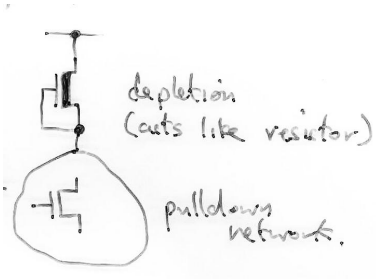


Figure 12.8: nMOS depletion example

Another solution is to use ”pseudo nMOS” in CMOS as shown in figure 12.9.

So when we want a zero output from the gate the path in pull-down network wins over the weak pull-up one. Where weak equals high resistance, low width and possibly higher length. So the net effect of this structure is that it is slow to pull-up and fast to pull-down.

So we construct small cells with the product line going horizontally and the control line vertically. In some of these there

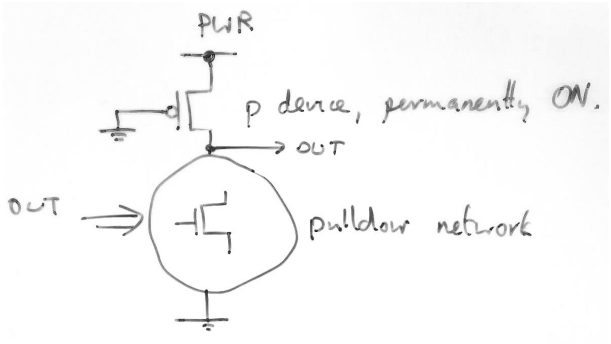


Figure 12.9: Pseudo nMOS

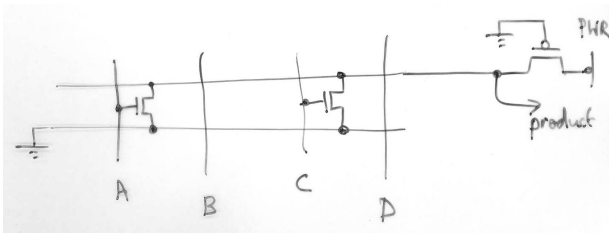


Figure 12.10: Unknown diagram

will be no connection but in others, the control line will pull the product line to ground if high. When constructing this we choose the control line to be constructed from poly. Now if we use metal for the power and ground that we run horizontally, it causes a large spacing gap between the power rails which is not ideal (figure 12.11).

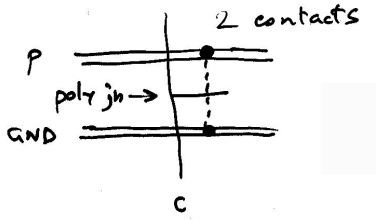


Figure 12.11: Using metal for power rails

Using lateral thinking, we try to running the ground wire vertically and using diffusion. which gives us a much more compact design (figure. 12.12) But this has some problems as the ground now has larger internal resistance and is slower and noisier.

The next step is to merge adjacent columns to share grid lines:

This is possible as in general C_1 is the complement of C_2. So we build an inverting buffer at the input to facilitate this process.

Further, in the interconnect between the AND-OR arrays we must construct a connection tile to take the 2 rows of outputs from the AND array and convert them into 1 double column of the OR array. So a possible layout for only one transistor connected in the pair is shown below:

It is naturally possible to construct a FSM from this by simply passing the outputs of the array back in as inputs.

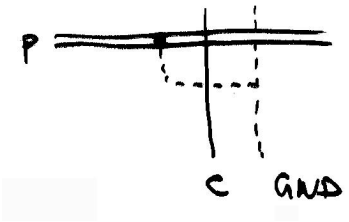


Figure 12.12: Lateral model

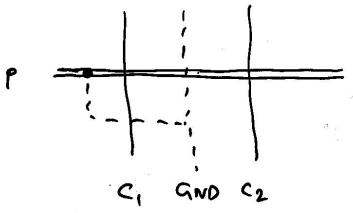


Figure 12.13: Cells that have a common ground

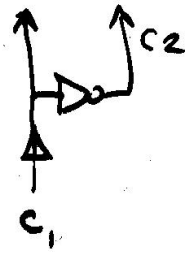


Figure 12.14: An inverting buffer

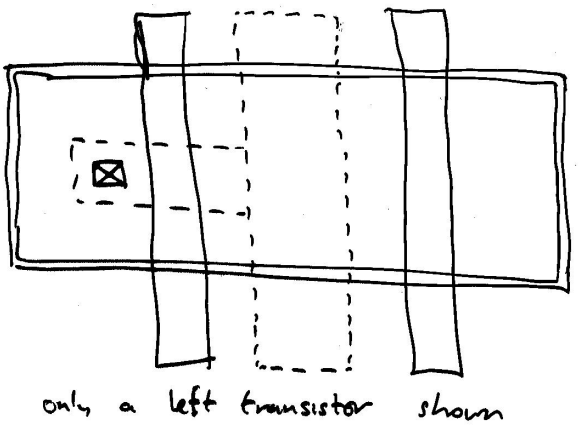


Figure 12.15: Cell layout with only left side connected

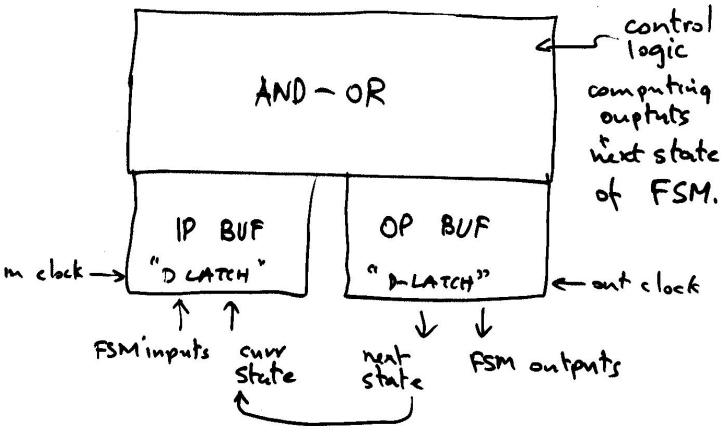


Figure 12.16: A finite state machine

Chapter 13

PLA Design Example

13.1 PLA Tiles (pseudo-nMOS)

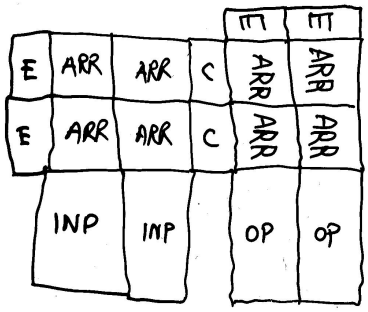


Figure 13.1: PLA tiles array

Tile types:

- ARR - AND/OR array multiple versions
- E - End tiles connect GND and pull-up to product lines.
- C - Connection tiles turn metal product lines onto Polysilicon input for OR-array.

The INP (input) tile is composed of two inverters to strengthen the input and also provide the inverse of the input also that may be necessary for the coming calculation. It is important to buffer the input if the array is large. The OP (output) tile does buffering of the output and optional inversion which will take the weak and slow result from the array and buffers it before making it available on the bus.

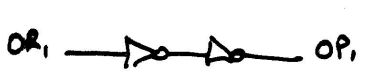


Figure 13.2: Output buffering

Example: X = $A\overline{B} + B\overline{A}, Y = AC + A\overline{B}, Z = \overline{\overline{C} + B}$

Step 1: Determine product terms: Z = $\overline{C}\overline{B}$

A$\overline{B}$	X, Y
$\overline{A}B$	X
AC	Y
$\overline{B}\overline{C}$	Z

Step 2: Sizing: #inputs = 3, #products = 4, #output = 3.

AND-array: #rows = $\left\lceil \frac{\#products}{2} \right\rceil$ = 2, #cols = no. of inputs = 3.

OR-array: #rows = $\left\lceil \frac{\#output}{2} \right\rceil$ = 2, #cols = $\left\lceil \frac{\#products}{2} \right\rceil$ = 2.

Step 3: array PLA floorplan

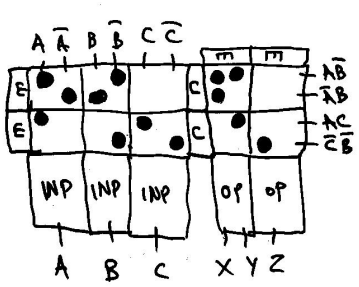


Figure 13.3: An example of a PLA implementation

Step 4: program the array. Each array tile has up to four ”dots” or transistors and the dot denotes that a transistor is present. These structures have strong regular format and it is easy to construct one with 16 possibilities. This is the basis of automated logic generation in ICs. You should minimise using Quine-McClusky and convert the equations into INV-AND-OR forms. This makes for easy logic design but it has some disadvantages like unbalanced CMOS logic - slow pull-up, fast pull-down, long lines forming more slowdown which is not ideal for large amounts of logic.

Chapter 14

Dynamic CMOS

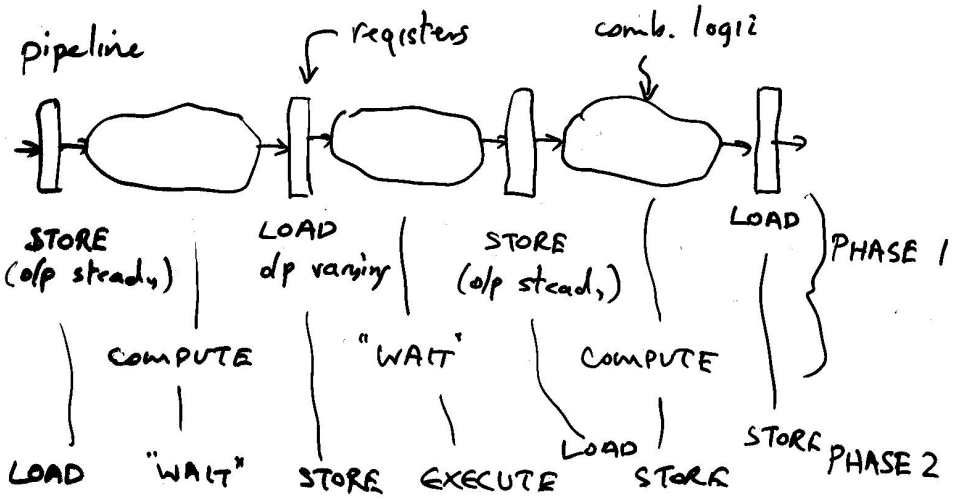


Figure 14.1: Traditional Pipeline

This is a 2 phase pipeline (figure 14.1), where both phases are of equal length. Figure 14.2, shows a 1-bit slice of a register with 8 transistors 4 N-type and 4 P-type.

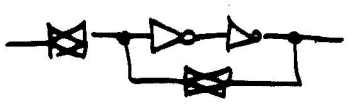


Figure 14.2: Register Implementation

In this implementation, we LOAD when the gate is closed and STORE when the gate is open (See Figure 14.4).

So where is the information stored for the compute block? The answer is in the input (parasitic) capacitance. The method for loading and storing can be seen in figures 14.5, 14.6.

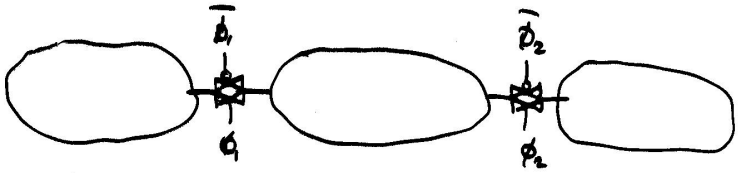


Figure 14.3: Transmission Gate as Register

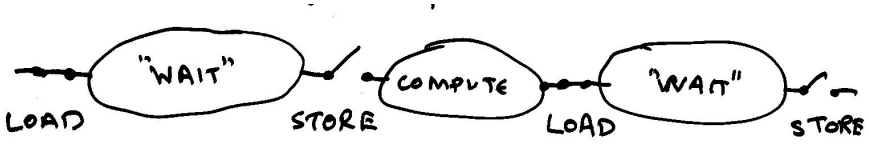


Figure 14.4: The wait/compute cycle

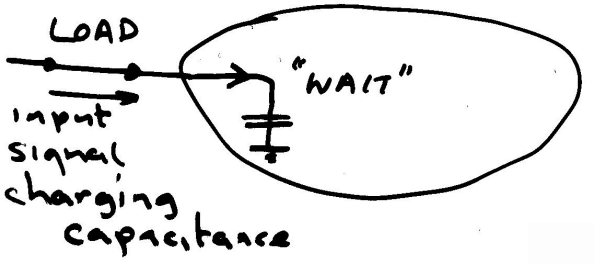


Figure 14.5: Loading value into capacitor

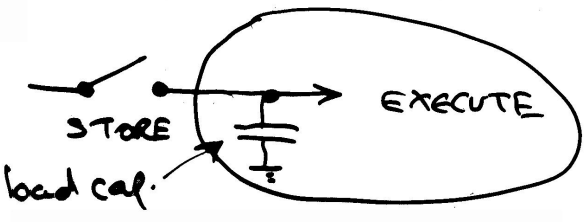


Figure 14.6: Storing value into capacitor

14.1 Dynamic Storage

Dynamic storage relies on charge of internal capacitance. It is non-restoring and no feedback to restore signal strength.

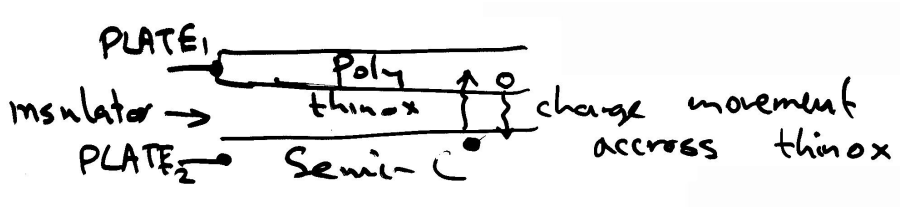


Figure 14.7: Charge movement across thin oxide

The decay time $\approx$ 100ms for big devices and ms for smaller devices. These devices need regular restoring so we are not able to stop the clock without disturbing this process.

14.2 Dynamic RAM

Dynamic RAM memory cell is a 1 transistor cell what uses the load capacitance to store the value specified but this method needs regular refreshing due to the .

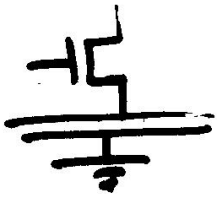


Figure 14.8: DRAM memory cell

14.3 Static RAM

This memory cell uses 4-6 transistors to hold data as long as power is applied.

14.4 Tricks of Dynamic Logic

With the clock symbol $\phi = 0$ C_P closed and C_N is open and so the CMOS is in Precharge (wait) mode (OUT $\leftarrow$ 1). $\phi = 1$ has the circuit in Execute mode (OUT $\leftarrow$ 0 if IN requires it).

n-Transistors are good at transmitting 0 but pull-down nodes are bad at 1s where as pull-ups are precharged to 1 and so compute only requires pull-down then we only need n devices for routing. So we need n instead of n+p transistors in a transmission gate (See figures 14.10, 14.11).

Carry chain adders are an example of this. Carry lookahead adders helps reduce delay. So instead of using the slow and expensive method shown in figure 14.12, 14.13, we can use us the Manchester Carry Chain (figure 14.14). It allows us to propagate a one if necessary otherwise we generate a 0.

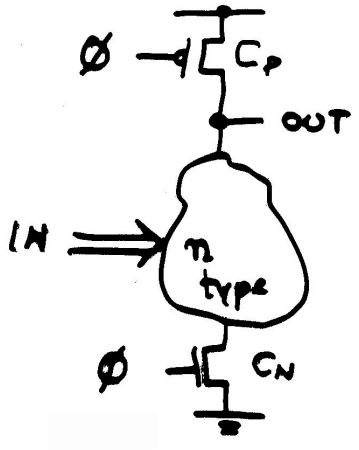


Figure 14.9: Dynamic CMOS

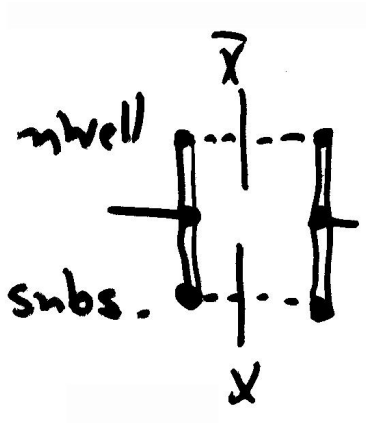


Figure 14.10: Bad design

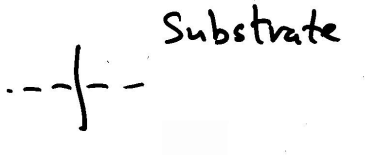


Figure 14.11: Good Design

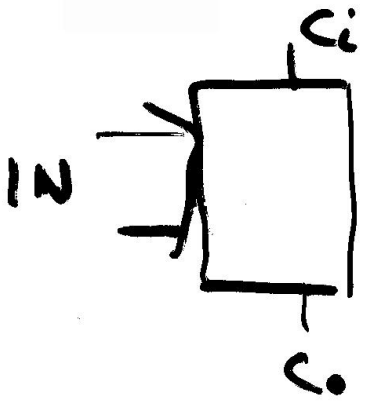


Figure 14.12: Standard propagate adder

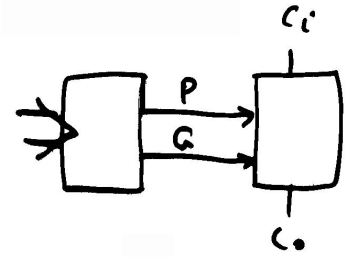


Figure 14.13: Carry Lookahead Adder

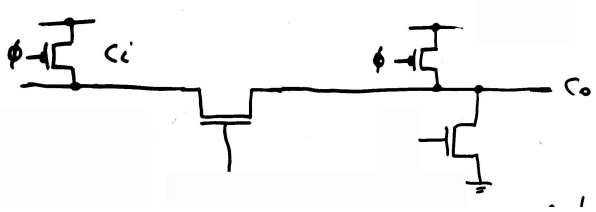


Figure 14.14: Manchester Carry Chain Adder

Dynamic logic seems really good as it is compact, layout area is small and it is fast but does it have any downsides? Timing and charge sharing complexities are involved. Charge sharing problems include the fact that the circuit has a bi-directional nature when we wish the information to only flow in one direction. So in figure 14.15 shows the undesirable nature of a bi-directional transistor, as the right hand side might influence the left hand side rather than vice-versa. We need to ensure that the information source is either properly driven (connected to either 0 or 1) or has more capacitance than the destination.

A timing/race condition occurs in figure 14.16 as we need to delay the star of the computation phase because at the start of the compute cycle all inputs are high, and starting to discharge OUT_2 before the inputs can reach final values.

The solution is to use extra clocks to delay the execution but you need to manage and distribute the multiple clocks and they become difficult to manage (figure 14.17).

Another solution is to alternate n-type and p-type dynamic logic which we refer to as domino CMOS (figure 14.18).

So when the n-block is precharging to 1 the p-block is computing and when the p-block is precharging to 0, the n-block is

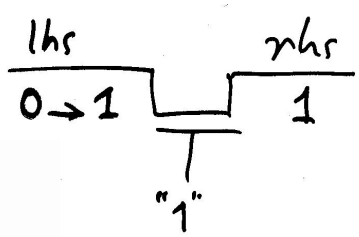


Figure 14.15: Right-hand-side influences left-hand-side

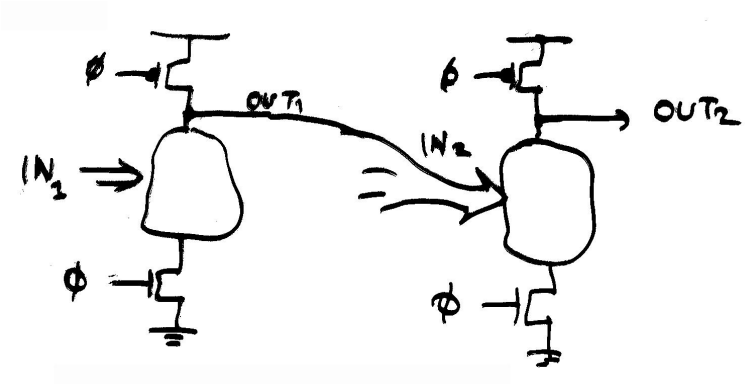


Figure 14.16: Timing/race condition

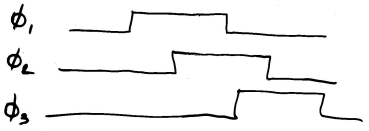


Figure 14.17: Using extra clocks

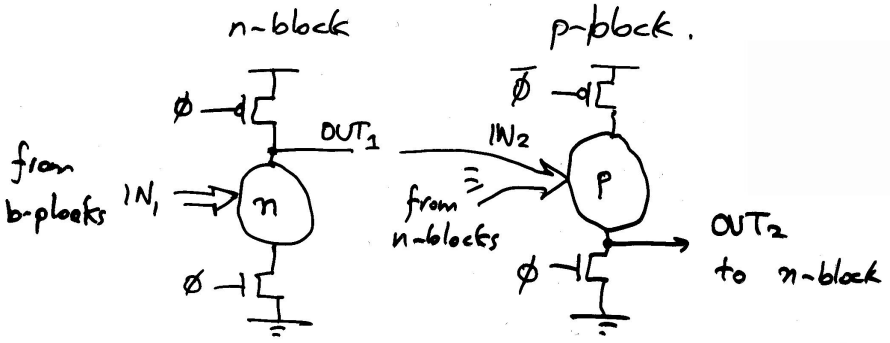


Figure 14.18: Domino CMOS

computing. Figure 14.19 shows an inverse clock structure that will work well.

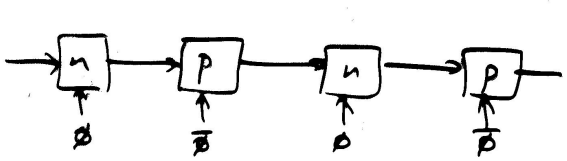


Figure 14.19: Inverse clock structure