ECE 445 Spring 2022 Course Review

Saif K.

July 31, 2022

Chapter 0

MOSFETs

0.1 Intrinsic Semiconductors

0.1.1 Silicon

Silicon is a group IV element, so it forms a covalent bond with 4 neighbours. At 0 Kelvin, the bonds are intact and no electrons are available for conduction, so it is an insulator. At room temperature, some bonds break, freeing electrons. This also creates **holes** in equal proportion. Equation 1 gives the number of free electrons per unit volume as a function of temperature.

$$n_i = BT^{\frac{3}{2}} e^{-\frac{E_g}{2kT}} \tag{1}$$

- B is a material-dependent parameter
- E_g is the bandgap energy (for Silicon, 1.12eV) - the minimum energy required to break a covalent bond
- k is Boltzmann's constant

0.1.2 Doped Semiconductors

Adding Phosphorous to a Silicon network adds free electrons. This creates **n-type** silicon. Adding Boron adds free holes. This creates **p-type** silicon. The dopant carriers (electrons and holes) cause two types of current to flow:

Drift Current

Drift current begins to flow when an electrical field is applied (or, equivalently, a potential difference appears) to a piece of silicon. The velocity of the carriers is given by

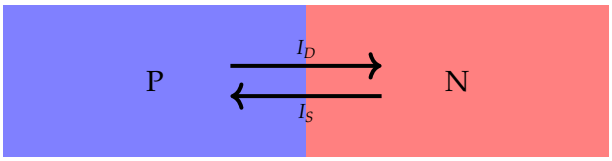
$$v_{\text{p-drift}} = \mu_p E$$
$$v_{\text{n-drift}} = \mu_n E$$

Where μ is the **mobility** of the type of carrier. In silicon, μ_p is typically **lower** than μ_n

Diffusion Current

Diffusion current is a result of a non-uniform density of carriers (e.g. at the boundary of two different dopings of Silicon). This is a typical differential equation problem and the main result is that this causes some carriers to travel into the rest of the silicon and even out the distribution, though the rate decreases as the process continues.

0.2 PN Junction



In a PN junction, we have P-type silicon touching N-type silicon. The P-type silicon has free holes from doping, and some electrons from the intrinsic Silicon (depending on temperature). The N-type silicon has free electrons, and some holes from the intrinsic Silicon (depending again on the temperature). These are respectively the **majority** and **minority** carriers of each node. Recall that positive charges move towards a *lower* potential (you can think of this as positive voltages *repelling* positive charges.

0.2.1 Diffusion Current

As holes migrate from P to N due to diffusion, they neutralize some electrons. This *uncovers* the minority carriers in N forming a slightly positive charged region in the N-type silicon near the boundary. The opposite process happens for electrons going from N to P which forms a **depletion region**. This forms a potential difference across the depletion region equal to the *barrier voltage* V_0 that prevents more current from flowing from P to N.

0.2.2 Drift Current

Minority holes in the N-type silicon that drift towards the edge of the depletion region will be *accelerated* by the potential drop towards the P-type silicon. The value of this current strongly depends on the temperature since that determines the density of minority carriers in each node.

0.2.3 Equilibrium

In equilibrium and open-circuit conditions, no current flows and thus $I_D = I_S$ which is maintained by the barrier voltage V_0. Thus, at room temperature we can experimentally determine V_0 for silicon as being in the range 0.6V to 0.9V

0.2.4 Applied Voltage

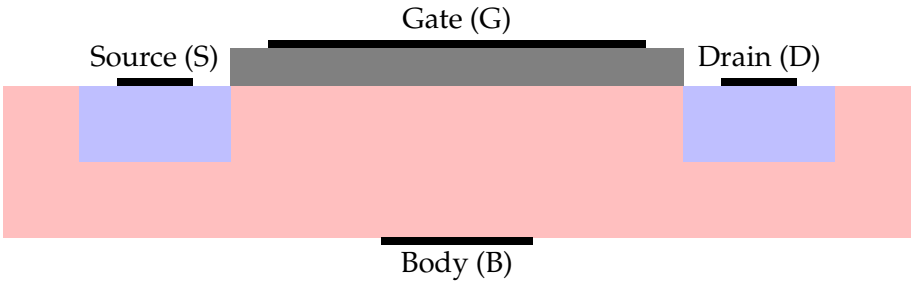
A PN junction is **reverse biased** when the net barrier voltage is positive. In this case, little to no current flows, except for a drift current that increases with the barrier voltage (as minority carriers are moved through more quickly in this case). A PN junction is **forward biased** when the applied voltage is negative. To maintain the applied voltage, there must be a source of electrons that tends to 'feed' the N-type silicon and thus drastically increases the **diffusion current**.

0.2.5 Reverse Breakdown

At some point, a very negative positive barrier voltage will suddenly cause a huge current to flow from N to P. This is cause either by the **zener effect** or **avalanche effect**, the details of which are outside the scope of this course.

0.3 NMOS Transistors

The **N**-channel **M**etal-**O**xide-**S**emiconductor transistor has the following structure



0.3.1 Structure

Where blue is P-type silicon, red is N-type (often you will see n^+ to denote that it is 'heavily' doped, and n^- would be lightly doped), black is metal, and gray is SiO_2 (the oxide part of NMOS). The thickness of this oxide layer is important, and denoted t_{ox}. The body is typically connected to the **source**. The Source and Drain are interchangable - there is no intrinsic difference between them in the structure of the NMOS, so the drain is whichever terminal is more positive in a given circuit - current **always** flows from drain to source.

0.3.2 Operation

Applying a positive voltage at the gate (relative to the source) causes an electric field that pushes holes in the substrate down towards the body terminal. When the applied voltage V_{GS} exceeds V_t, a **channel** is established - the p-type substrate essentially flips to n-type and allows electrons to flow. It is useful to define the **overdrive voltage**

$$V_{OV} = V_{GS} - V_t \tag{2}$$

When V_{DS} is small, we can compute the charge in the channel as

$$|Q| = C_{ox} \cdot WL \cdot V_{OV} \tag{3}$$

$$C_{ox} = \frac{\epsilon_{ox}}{t_{ox}} \tag{4}$$

There are three regions of operation, which determine what the current I_{DS} is. Note first the following definitions:

$$k_n' = \mu_n C_{ox} \tag{5}$$

$$k_n = \left(\frac{W}{L}\right) k_n' \tag{6}$$

k_n' is called the **process transconductance** parameter.

$$\text{Triode} \quad i_{DS} = k_n \left(V_{OV} - \frac{1}{2} v_{DS}\right) v_{DS} \tag{7}$$

$$\text{Saturation} \quad i_{DS} = \frac{1}{2} k_n V_{OV}^2 \tag{8}$$

$$\text{Cut-Off} \quad i_{DS} = 0 \tag{9}$$

When $V_{OV} < 0$, the NMOS is in cutoff. When $V_{OV} \geq 0$, there are two cases:

- $v_{DS} < V_{OV}$: Triode

- $v_{DS} \geq V_{OV}$: Saturation

Note that in triode, with small v_{DS}, the v_{DS}^2 term can be omitted and thus the transistor acts as a linear resistance between the source and drain controlled by V_{OV}:

$$r_{DS} = \frac{1}{k_n V_{OV}} \tag{10}$$

0.4 PMOS

PMOS is the opposite of NMOS, and all the equations work out the same if you use the parameters V_{SG}, V_{SD}, and $|V_{tp}|$. In PMOS, applying a **negative** voltage at the gate establishes a channel. The drain is always the more **negative** of the two terminals, and current always flows from source to drain (these two statements are the opposite as those for the NMOS).

0.5 MOSFET Circuits

To solve DC MOSFET circuits, you will sometimes need to *assume* regions of operation for each transistor, solve the currents/voltages, and then confirm that the assumptions were correct.

0.6 Secondary Effects

0.6.1 Channel-Length Modulation

In saturation, using the **long-channel** model we have assumed, the transistor has infinite output resistance. This is not true as in reality, increasing v_{DS} causes channel length modulation. That is, it shortens the channel and increases the potential drop between the channel and the other terminal, slightly increasing the current. This is modelled by adding the factor $\lambda = \frac{1}{V_A}$ to the saturation current where V_A is the 'Early voltage', as seen in 11. This results in a **finite** output resistance r_o given by 12

$$i_{DS} = \frac{1}{2} k_n V_{OV}^2 (1 + \lambda V_{DS}) \tag{11}$$

$$r_o = \frac{1}{\lambda I_D} \tag{12}$$

Chapter 1

CMOS Logic

1.1 CMOS Gates

1.1.1 Switches

It is pure **convention** that a logical 1 is considered to be a high voltage, e.g. V_{DD} and a logical 0 is GND. Regardless, with this convention in mind, we can treat a NMOS transistor as a switch that is ON when provided with a logical 1 and OFF otherwise. Similarly, a PMOS is a switch that is ON when provided with a logical 0 and OFF when provided with a logical 1. By 'provided' we mean the logical value is presented at the gate, and the switch is ON when it connects the drain terminal to the source.

1.1.2 Static CMOS Inverter

Figure 1.1 shows the circuit. Some notable features:

- Exactly one path (through Q1 or Q2) is active for any input, so there is no middle state output
- Input is connected to the gates of MOSFETs, so there is low input leakage
- The input is electrically isolated from the output

1.1.3 Static CMOS Logic

In general, a static CMOS gate has the structure shown in Figure 1.2. The pull-up network (PUN) is formed with PMOS, while the pull-down network (PDN) is formed with NMOS. To find the PDN for a function $y = f(\vec{x})$, apply DeMorgan's rules (1.1 and 1.2) to find $\bar{y}$ as a function of the inputs, and use transistors in series to implement AND and transistors in parallel to implement OR. Then complement the PDN to find the PUN (using parallel PMOS in the PUN to correspond to series NMOS in the PDN and so forth).

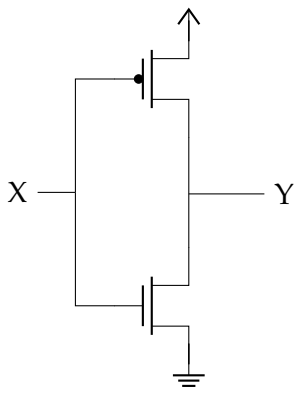


Figure 1.1: A Static CMOS Inverter

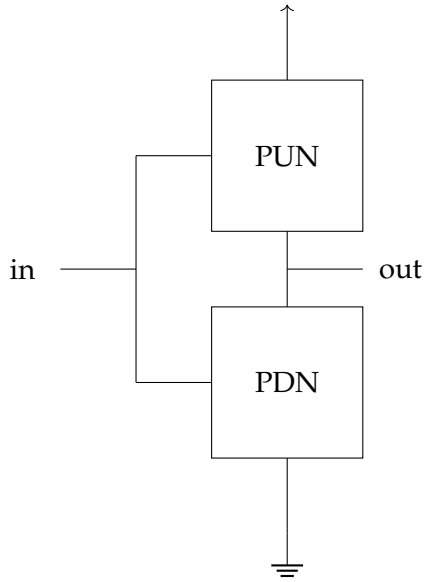


Figure 1.2: Structure of a static CMOS gate

$$\overline{(A+B)} = \overline{A}\,\overline{B} \tag{1.1}$$

$$\overline{AB} = \overline{A} + \overline{B} \tag{1.2}$$

1.2 Inverter Voltage Transfer Curve (VTC)

1.2.1 Overview

For the inverter circuit in Figure 1.1, we can find V_{out} as a function of V_{in} under the **quasi-static** assumption (rather than **dynamic**). In this analysis, we assume V_{in} is stable for a long time and that $I_{DN} = I_{DP}$ (the current flowing into the NMOS drain and out of the PMOS drain are equal). There are 5 regions of operation. We can derive the transistor voltages using V_{in} and V_{out} as follows:

$$\begin{aligned} v_{DSN} &= V_o \\ v_{GSN} &= V_i \\ v_{SDP} &= V_{DD} - V_o \\ v_{SGP} &= V_{DD} - V_i \end{aligned}$$

1.2.2 Regions of Operation

A common case for analysis is when the transistors are **balanced**, that is $k_n = k_p$ and $V_{tn} = |V_{tp}| = V_t$. The regions of operation are as follows, in order of increasing $V_i = V_{in}$:

1. Region 1 - $V_{in} < V_{tn}$
   - By definition, the NMOS is off. v_{SGP} is very high and v_{SDP} is low, so PMOS is in triode
   - $I = 0$ and $V_o = V_{DD}$
2. Region 2 - $V_{in} \geq V_{tn}$
   - NMOS turns ON, and PMOS is still in triode
   - NMOS will be in saturation as v_{DSN} is large
   - Output begins to discharge through the NMOS, but PMOS remains triode until $v_{SDP} \geq v_{SGP} - |V_{tp}|$, which is equivalent to $V_o \leq V_i + |V_{tp}|$
   - So region 2 ends when V_o decreases to $V_i + V_{tp}$

- By the quasi-static condition, we have $\frac{1}{2}k_n(v_{GSN} - V_{tn})^2 = k_p(v_{SGP} - |V_{tp}| - \frac{1}{2}v_{SDP})v_{SDP}$, from which we can find V_o as a function of V_i in this region.

3. Region 3 - $V_i = V_M$
   - Both PMOS and NMOS in saturation
   - $V_M - V_{tn} \leq V_o \leq V_i + |V_{tp}|$
   - When **balanced**, $V_M = \frac{1}{2}V_{DD}$ and $V_{tn} = |V_{tp}| = V_t$ so $|V_0 - \frac{1}{2}V_{DD}| \leq V_t$
   - If we let $r = \sqrt{\frac{k_p}{k_n}}$, we can analyze how V_M shifts: $r > 1$ means **PMOS** is stronger, so V_M shifts to the right. Intuitively, the minimum V_{SGP} required to turn the PMOS on is now lower, so that region shifts right.
4. Region 4 - $V_{in} \leq V_{DD} - |V_{tp}|$
   - NMOS in triode, PMOS turns ON in saturation
   - Mirrors region 2
5. Region 5 - $V_{in} \leq V_{DD}$
   - NMOS in triode, PMOS is cut off
   - Mirrors region 1

Note that these assume that there is no channel length modulation. Adding the CLM factor will give a finite width for region 3 that has a slight slope.

If we were to do something similar for the drain current as a function of input voltage, we would find a curve that is 0 when the input is near a logic 0 or 1, and a spike at around V_M ($\frac{V_{DD}}{2}$ for balanced circuit). This is related to the dynamic or 'switching' power - it takes energy to switch this circuit.

1.2.3 Noise Margins

Noise margins determine how much noise the circuit can tolerate. There are two noise margins, the margin for low input $\text{NM}_L = V_{IL} - V_{OL}$ and the margin for high input $\text{NM}_H = V_{OH} - V_{IH}$. The overall noise margin NM is the **minimum** of the two.

- V_{IL} is the maximum input that will be considered a 0
- V_{IH} is the minimum input that will be considered a 1
- V_{OL} is the maximum output value that will be produced when it should be a 0
- V_{OH} is the minimum output value that will be produced when it should be a 1

The definition we adopt in this course is that these values are defined by the two points on the inverter VTC where $\frac{\partial V_o}{\partial V_i} = -1$. The lower one is (V_{IL}, V_{OH}) and the upper is (V_{IH}, V_{OL}).

Chapter 2

Dynamic CMOS Behaviour

In this section we look at the response of CMOS circuits in the **dynamic** situation. We no longer assume the quasi-static case. The main focus here is the **delay**, although in modern technologies power equally if not more important. We begin with the following definitions (with a static CMOS inverter in mind):

- Switching delay t_{pLH} and t_{pHL}, which is the time from the switching point of the input to the time when the output becomes high or low respectively
- Propagation delay $t_p = \frac{t_{pHL}+t_{pLH}}{2}$
- Fall time t_f (90% V_{DD} to 10%V_{DD})
- Rise time t_r (10% V_{DD} to 90%V_{DD})

2.1 RC Delay Model

The full analysis is complex, so we assume a simple linear model where the transistor is a resistor in series with a switch and some capacitance.

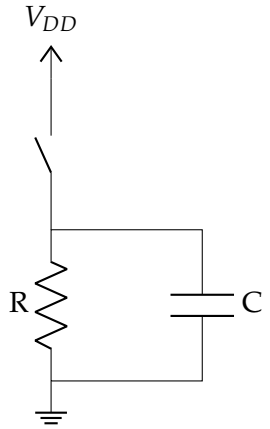


Figure 2.1: RC model of a transistor

In this model, the output response to a step input will be an exponential decay with time constant τ. The capacitance and resistance are functions of the transistor aspect ratios. The NMOS and PMOS of a given technology will have some effective resistance R. If we scale the width of a transistor by a factor k, we get the following effects:

$$R' \leftarrow \frac{R}{k} \tag{2.1}$$

$$C' \leftarrow kC \tag{2.2}$$

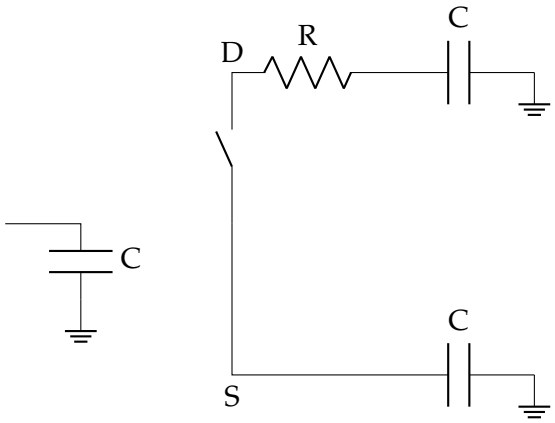


Figure 2.2: NMOS RC Model

That is, the capacitance goes up and the resistance goes down (in turn increasing the current the transistor can supply).

2.2 Sizing a Single Stage

The goal in sizing one stage of a circuit is to get $t_{pHL} = t_{pLH}$. This is accomplished by computing the 'effective widths' of the PUN and PDN in Figure 1.2. The effective width of the PUN should equal the width of the PMOS in the technology's basic inverter (called the **unit inverter**). The effective width of the PDN should equal the width of the NMOS in the basic inverter.

Given μ_p and μ_n, the first step is to find the relative sizes in the basic inverter. Since $k = \mu C_{ox} \frac{W}{L}$ and L and C_{ox} are typically fixed values, $k \propto \mu W$ and thus we have $\mu_N W_N = \mu_p W_P$ to balance the inverter. Thus we have the ratio $\frac{W_P}{W_N} = \frac{\mu_N}{\mu_P}$. Given a value of W_N we can solve for W_P.

The goal is to get the PUN to sum up to W_P and the PDN to sum up to W_N by assigning the width of each transistor to be some multiple of W_P and W_N respectively. In series, the effective width will be given by Equation 2.3 and in parallel they simply add up. So for example, if we have N transistors in series they should all be N times the unit width to get the correct sum. **Important** in the parallel case, we need *each branch* to be equal to the basic width since a single branch supplying current is the 'worst case' for supplying current, and we want the worst case delays to be equivalent to the unit inverter.

$$\frac{1}{W_e} = \frac{1}{W_1} + \frac{1}{W_2} \tag{2.3}$$

2.3 Elmore Delay

A more useful RC model for the delay is given by Figure 2.2. Note that the resistances and capacitances are determined by Equation 2.1 and Equation 2.2.

Using this, and noting that the capacitances connected to a constant supply may as well be connected to ground, we can draw out the full RC model for the inverter, shown in Figure 2.4. Note that technically there are 3 process-dependent capacitances C_g, C_s, C_d (and technically even more), but this is a *first-order* assumption that they are all equal to kC where C is the unit width capacitance.

Also note that Figure 2.4 assumes the PMOS was scaled by 2, which decreased its effective resistance to be the same as that of the NMOS path.

Delay calculations are based on the **Elmore delay**. Simply put, we estimate the delay of a path in an RC network by taking the sum of the delay at each R along the path. The delay at a node with a resistance R is RC_p where C_p

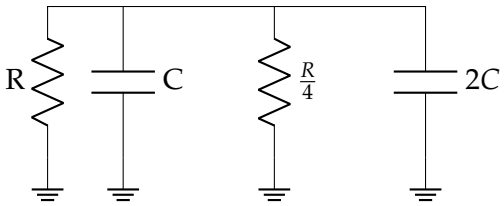


Figure 2.3: Simple Elmore delay example

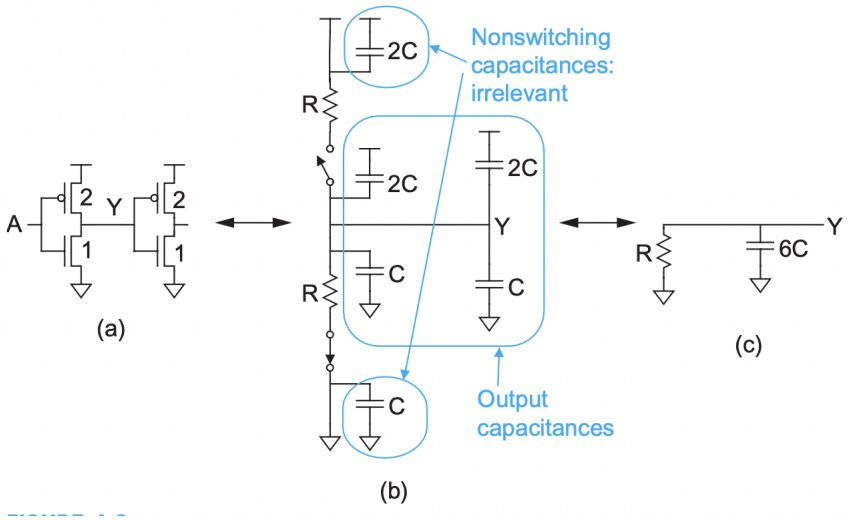


Figure 2.4: The full RC model of the inverter

is the sum of the capacitances from the node R to the desired output node. For example, the delay in the network of Figure 2.3 is $R(C+2C)+\frac{R}{4}(2C)$.

2.4 Power

2.4.1 Dynamic Power

Charging a capacitor to some voltage takes energy. The power is given by $P(t) = I(t)V(t)$ and the energy is $E = \int_0^\infty P(t)dt$. When a source at voltage V_{DD} charges the capacitor to V_{DD}, it delivers a current $i_c(t) = C\frac{dv_c(t)}{dt}$ and thus uses energy

$$E_{supply} = \int_0^\infty C\frac{dv_c(t)}{dt}V_{DD}dt = CV_{DD}\int_0^{V_{DD}} dv_c = CV_{DD}^2 \tag{2.4}$$

Meanwhile, the energy stored in the capacitor is given by

$$E_C = \int_0^\infty C\frac{dv_c(t)}{dt}v_c(t)dt = C\int_0^{V_{DD}} v_c(t)dv_c = \frac{1}{2}CV_{DD}^2 \tag{2.5}$$

That is, half of the energy given by the supply is stored on the capacitor and the other half is dissipated. In a CMOS inverter, we thus have that every time the *output* has to go from 0 to 1, it dissipates $\frac{1}{2}CV_{DD}^2$ Joules of energy. When the output then has to go from 1 to 0, the capacitor discharges through the NMOS (**the supply does not provide any energy**) and dissipates $\frac{1}{2}CV_{DD}^2$.

Note that the average power in a period T is given by $P_{av} = \frac{E}{T}$ where E is the energy used in that period. Consider the clock. In one period, it must make **both** transitions (1 to 0 and 0 to 1), thus the average power of a clock with

period T is $\frac{1}{T}(\frac{1}{2}CV_{DD}^2 + \frac{1}{2}CV_{DD}^2) = \frac{1}{T}CV_{DD}^2 = fCV_{DD}^2$ where f is the clock frequency. Similarly, for a signal that switches on average α fraction of the time, the power is

$$P_{dyn} = \alpha f C V_{DD}^2 \tag{2.6}$$

Where α is the activity factor. Note that the activity factor for the clock is **always** 1. For a signal that switches value, the activity factor is 0.5 (**very important** - it is *not* 1 because such a signal actually only changes value half as often as the clock).

If you are given a signal over N clock periods, you can find the activity factor by counting k, the number of edges depicted, and the activity factor is $\frac{k}{2N}$

An alternative definition of α is the probability that a signal switches from a 0 to a 1.

2.4.2 Subthreshold Leakage

In modern technologies, we actually have a nonzero drain current even when $v_{gs} < V_t$ (similar for PMOS). This is proportional to a few things, and the current is usually given in a datasheet. In general,

$$I_D = I_0 e^{\frac{v_{gs}}{\frac{nkT}{q}}} \left(1 - e^{-\frac{v_{ds}}{\frac{kT}{q}}}\right) \tag{2.7}$$

2.4.3 Static Power Dissipation

Due to subthreshold leakage (subsection 2.4.2), the NMOS and PMOS of a gate will leak power even when they are not switching. Given base values for the currents, I_{N0} and I_{P0}, we can compute an estimate for the static power by multiplying them by the widths of the sized transistors.

To estimate the power, we need to consider two cases. When the output is a 1 (V_{DD}), only the leakage current of the **NMOS** consumes power. This is because $v_{sdp} = 0$ and thus the current I_D is 0 in Equation 2.7. When the output is a 0, only the leakage current of the **PMOS** consumes power, for similar reasons.

Thus, for a given set of inputs, you can estimate the static power as follows. Find the output, and determine whether the NMOS or PMOS case is to be considered. Then find the effective width of the transistors that are OFF, and the power is $w_e I_0 V_{DD}$ where w_e is the effective width and I_0 is the per-width leakage current for that type of transistor (NMOS or PMOS).

Chapter 3

Combinational Circuits

3.1 The Logic Effort Method

3.1.1 Single Stage

We need some way of intuiting which design for a given boolean function is good. The delay is what we focus on here, though power is also important. So, for a given topology, how can we quickly estimate the delay? Using the Elmore delay of section 2.3. We would like to see how a single logic gate performs in a given circuit. The delay of a gate that is connected to some other gates is given by the product $R(C_{parasitic} + C_{load}) = R(C_{parasitic} + C_{in} \cdot \frac{C_{load}}{C_{in}})$.

- R is the path resistance which has been designed to be equal to the path resistance of the basic inverter (see section 2.2)
- $C_{parasitic}$ is the capacitance at the output of the logic gate, which depends on what transistors are connected to the output node of the logic gate and what their sizes are
  - What makes it parasitic? Well because ideally we would want it to be 0 so we can just drive the load capacitances (which by their very existence are non-zero). It is non-zero because we need to use transistors to do this, which come with some output capacitance.
- C_{load} is the input capacitance of the total load that the logic gate drives (i.e. the sum of input capacitances of all the gates it drives)

We can write that delay also in the following manner:

$$d = p + f = p + gh \tag{3.1}$$

Let the delay of the unit inverter be τ_0 (e.g. $\tau_0 = 3RC$ if $\mu_n = 2\mu_p$) and the 'unit capacitance' $C_0 = \frac{\tau_0}{R}$. Note the following:

$$\tau_0 = \left(\frac{\mu_n}{\mu_p} + 1\right) RC \tag{3.2}$$

$$C_0 = \left(\frac{\mu_n}{\mu_p} + 1\right) C \tag{3.3}$$

- d is the **normalized** delay $\frac{\tau}{\tau_0}$ - that is, to get the actual time constant we need to multiply it by τ_0.
  - Why normalize? Since reducing R or C always decreases delay, and the μ_p:μ_n ratio changes the unit inverter delay, normalizing it makes sure that all the technology-dependent factors are absorbed into d and we can just worry about relative values
- p is the normalized parasitic delay $\frac{C_{parasitic}}{C_0}$. It is **independent** of the transistor sizes in the gate.
- $f = gh$ is the 'stage effort', i.e. the effort that this logic gate has to put in.
- $g = \frac{C_{in}}{C_0}$ is called the 'logical effort'. It is a measure of the complexity of a gate, and can change with the widths of the transistors (sum of gate capacitances)

  - Since this is related to the input capacitance which is proportional to the transistor widths, this is sort of like a measure of how much current this gate can deliver relative to that of the basic inverter
- $h = \frac{C_{load}}{C_{in}}$ is called the 'fan-out' or 'electrical effort' factor. It's the ratio of the load capacitance of the logic gate to its input capacitance (or, if a gate is driving n gates identical to itself, $h = n$ which is where 'fan-out' comes from)
  - Why the *ratio* of load to input capacitance and not just the load capacitance? This is just so that the full path-effort method has a nice form (H can be computed as fixed for a given logic block) that can be optimized

3.1.2 Path Logic Effort

We need one more piece, which is the branching effort for a path $b = \frac{C_{loadonpath}+C_{loadoffpath}}{C_{loadonpath}}$. In any circuit there is some critical path whose delay limits the frequency it can be operated on. We introduce three terms that can be computed for the path which has N stages:

$$\text{Path Effort} \quad F = \prod f_i = \prod g_i h_i \tag{3.4}$$
$$\text{Path Logic Effort} \quad G = \prod g_i \tag{3.5}$$
$$\text{Path Branch Effort} \quad B = \prod b_i \tag{3.6}$$
$$\text{Path Electric Effort} \quad H = \frac{C_{out,path}}{C_{in,path}} \tag{3.7}$$
$$\text{Path Parasitic Delay} \quad P = \sum p_i \tag{3.8}$$
$$\text{Path Effort Delay} \quad D_F = \sum g_i h_i \tag{3.9}$$
$$\text{Path Delay} \quad D = P + D_F \tag{3.10}$$

Note that $F = GBH$. The goal is to minimize D by selecting h_i. That is, we have fixed a circuit topology and want to size each gate (i.e. select the W_N since the ratio of $W_P : W_N$ has already been decided). How do we do it? P is fixed so we want to minimize D_F which can be done by selecting all the $g_i h_i$ terms to be **equal**. That is, we want to find the optimal **stage effort** $\hat{f}$:

$$\hat{f} = g_i h_i = F^{\frac{1}{N}} \quad \forall i \tag{3.11}$$

If we do this, the minimum delay is given by

$$\hat{D} = P + N\hat{f} = P + NF^{\frac{1}{N}} \tag{3.12}$$

The process for a given circuit and path is as follows

1. Size each gate (pick widths relative to W_N)
2. Compute g_i and an expression for each h_i
3. Compute G, B, H, and thus F. Compute P
4. Compute $\hat{f} = F^{\frac{1}{N}}$. Compute $\hat{D} = P + N\hat{f}$.
5. Using $\hat{f} = g_i h_i$, compute a value for the input capacitance (this will be for a *single input* of the gate)
6. Using the input capacitance from the previous stage, write an expression for W_N

For example of step 6, if the input capacitance is x for an input A, and the total input capacitance for the gate is given by $kW_n + jW_p$ and $W_p = \frac{\mu_n}{\mu_p} W_n$, we can write

$$x = kW_n + j\frac{\mu_p}{\mu_n} W_n \implies W_n = \frac{x}{k + j\frac{\mu_p}{\mu_n}}, \quad W_p = \frac{\mu_p}{\mu_n}\frac{x}{k + j\frac{\mu_p}{\mu_n}}$$

From which we can find the absolute widths kW_n and jW_p.

3.1.3 Adding Inverters

It is possible that adding a chain of inverters to the end of a logic block can, by scaling each element of said change, *decrease* the overall delay. If we let $\rho = \hat{f} = F^{\frac{1}{N}}$ and differentiate Equation 3.10 w.r.t. N, this is given by

$$p_{inv} + \rho(1 - \ln \rho) = 0 \tag{3.13}$$

Since this is not exactly solvable, we use the estimate $\rho \approx 4$ to get

$$N_{best} = \log_4(F) \tag{3.14}$$

Note that $N_{best} = N_{orig} + n_{inv}$.

3.2 Other Delay Reduction Methods

3.2.1 Gate Topology: Inner vs. Outer Inputs

Think of the Elmore delay of an input closer to the rail in a series of transistors connected to some output node. The outer nodes need to charge both the output capacitance and the series capacitance of all the drains/sources of the other transistors.

An **inner** node is one closer to the output, an **outer** node is closer to the rail. If we know which inputs arrive later than others, we should always put the latest input at the *innermost* node, so it has to charge the least capacitance.

3.2.2 Layout: Shared Capacitance

For transistors in series, using the n-well of one transistor's drain as the n-well for another transistor's source, can reduce the output node capacitance from $2C + 2C$ to just $2C$

3.2.3 Layout: Multi-Finger Devices

For very wide transistors, we can use N fingers to lay it out instead. N is the number of places the gate touches the device. This reduces the source/drain capacitances while maintaining the channel width.

3.2.4 Transistor Sizing: Asymmetric Gates

When we know one input is more critical than another, for example a reset, we can reduce its delay by reducing the widths of the transistors it is connected to. We must proportionally *increase* the widths of the other transistors to maintain the single-stage sizing.

3.2.5 Transistor Sizing: Skewed Gates

If a rising output of an inverter is more critical, we can downsize the nMOS so that the charging resistance is now proportionally larger than the discharging resistance.

3.3 Alternative Logic Families

Static CMOS is just one way to implement boolean functions.

3.3.1 Pseudo-NMOS

PMOS is slow, so what if we could omit it entirely? In pseudo-NMOS we use a very weak PMOS with its gate connected to ground (always-on) that serves as a pull-up resistor (why not just use a resistor? It would use more space in a CMOS technology). Then we implement our PDN as usual using NMOS.

When the output should be a 1, the PDN is disconnected and the output charges through the PMOS. When it should be a 0, the PDN pulls it down (and fights with the PMOS pullup!). This means that there is a lot of static power usage when the output should be a 0 (not when it should be a 1 as no current flows across the resistor).

Pseudo-NMOS has fast $1 \rightarrow 0$ transitions but slow $0 \rightarrow 1$ transitions. It has lower noise margins than static CMOS, and uses more power.

3.3.2 Pass-Transistor Logic

This logic family only uses NMOS gates. The idea is to use both the gates and source/drains of the transistors as inputs.

Pass-Gate

A 2-input PTL gate is implemented as follows. For inputs A,B there are two NMOS transistors Q1 and Q2. Q1 has its gate connected to B and Q2 has its gate connected to $\bar{B}$. The source of Q1 and the source of Q2 are each connected to one of $\{A, \bar{A}, 0, 1\}$. The drains of Q1 and Q2 are connected.

Unlike static CMOS, we must **ensure** that exactly one path is conducting (Q1 or Q2). We can implement 2^4 functions in this family. This family is good for reducing the number of transistors in specialized applications like FPGA matrices and memories.

V_{th} Drop

When G=1 and D rises to 1 in an NMOS transistor, S will rise (since it is a resistor with G fixed) until $V_S = V_{DD} - V_t$ at which point $V_{GS} = V_{DD} - (V_{DD} - V_t) = V_t$ and the transistor becomes cut-off. Therefore the 'output' in this logic family for a V_{DD} is $V_{DD} - V_t$. This is called the V_{th} **drop** and means that **the output of a PTL gate cannot be used as the gate input to another PTL gate**. It can be used as the drain input, provided $V_{DD} - V_t$ is large enough, but using it as the input to a gate means the next output will be $V_{DD} - 2V_t$ and so on.

Transmission Gates

A transmission gate is a NMOS-PMOS pair with the source of one connected to the drain of the other and vice versa. The input is a value X where X is passed to the gate of one transistor and $\bar{X}$ is connected to the gate of the other.

This essentially implements a **switch** that doesn't have a V_{th} drop. If X is connected to the NMOS, the switch turns on when $X = 1$. If $\bar{X}$ is connected to the NMOS instead, the switch turns on when $X = 0$.

Complementary Pass-Transistor Logic (CPL)

If we use pass-gate logic to implement both $F(A, B)$ and $\bar{F}(A, B)$, we can assume that all inputs to a logic gate have both polarities provided, and using a CMOS inverter to restore the logic levels we can reduce the number of inverters required.

3.3.3 Dynamic Logic

In dynamic logic, we rely on some node's capacitance to store a voltage rather than directly connecting it to some pull-up or pull-down.

General characteristics of 'dynamic' things:

- They use the capacitance of some node to store charge
- Higher density
- Less robust
- Higher power
- Fast

Dynamic logic has the drawback that the clock needs to be slow enough for precharging.

Clocked-Evaluation Dynamic Logic

Figure 3.1 shows the schema. In the **precharge** phase, the clock is low and the output node is charged to a 1. In the **evaluation** phase the clock is high and the output is equal to the function implemented by the PDN. There is one important exception: once Y discharges through the PDN, it cannot rise as the PMOS is disconnected. This is called **monotonicity**, and means that the output of such a logic gate cannot cascade to other similar logic gates unless monotonicity is guaranteed.

Note that the NMOS at the bottom of Figure 3.1 is called a **foot** - if we can guarantee the inputs are never 1 during evaluation, we can omit the foot and reduce the load capacitance of the clock.

We can decrease the logical effort for inputs in the PDN by *increasing* the width of the foot. The drawback is that this increases the clock load.

The pull-up transistor is usually given $\frac{1}{2}R$ so the precharge occurs quickly.

If we had some way of guaranteeing that no input goes from $1 \rightarrow 0$, then everything would be okay. This is what leads us to **domino** logic.

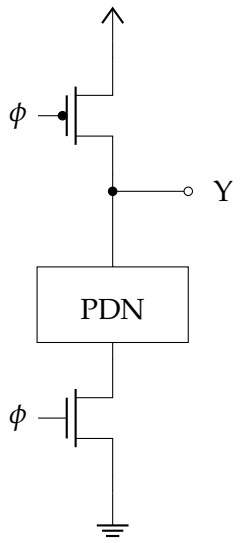


Figure 3.1: Clocked Evaluation Dynamic Logic Gate

Domino Logic

The term 'monotonically rising' means **only** that a signal *cannot* go from $1 \rightarrow 0$. Similarly, a 'monotonically falling' signal may not go from $0 \rightarrow 1$.

If a signal is monotonically falling, then feeding that signal to a static CMOS inverter causes an output that is monotonically rising. This is the idea behind domino logic. We stick some inverting static CMOS gate at the output of a dynamic logic gate and can thus cascade such **compound gates**. The drawback is that only *noninverting* logic can be implemented (e.g. no NANDs or XORs).

Zipper Logic

Consider an altered version of Figure 3.1 where we implement a PUN and feed $\bar{\phi}$ to the transistors instead. Such a gate would have a 'predischarge' phase when $\phi = 0$ and in eval allows a monotonically rising *output*. That is, it can **tolerate monotonically falling inputs**. Thus if we alternate stages of 'positive dynamic logic' (Figure 3.1) with the described 'negative dynamic logic', we have no issues with monotonicity.

Problems with Zipper logic:

- Requires ϕ and $\bar{\phi}$ everywhere, which is not an easy task
- Greatly increases the activity factor α (usually in the 30%-40% range compared to the $\sim$ 1% of static CMOS)
- Lower noise margin

3.3.4 Dynamic Logic Issues

Charge Leakage

Dynamic logic relies on a node's capacitance to store a logic level. In modern processes, due to high leakage, such nodes may only hold their charge on the order of nanoseconds. To solve this we can add a 'weak keeper' which is essentially a level restorer at the output.

Figure 3.2 shows a level restorer. It maintains the node Y at whatever its value is even as it drifts. Suppose Y is a 1 and it begins to discharge to some voltage below V_{DD}. As long as this is below the noise margin of the inverter, the inverter will continue to output $\bar{Y} = 0$. The V_{SG} of Q1 is held at V_{DD} so it is ON. decreasing Y *increases* V_{SD}, increasing the current drawn through Q1 which in turn decreases V_{SD} (thus brining Y back to a 1). A similar analysis holds for keeping a 0. **Note** that Q1 must be **weak** enough to be 'overridden' by the logic gate driving Y.

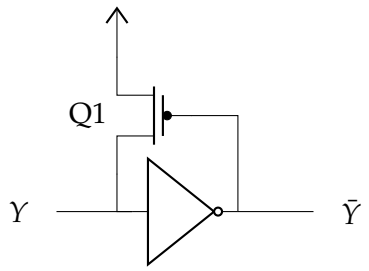


Figure 3.2: A CMOS level restorer

Charge Sharing

The output node of a dynamic circuit has to share charge with the internal capacitances of the logic gate when the PDN is in the ON state. To avoid this, we must also precharge all the internal capacitances, by adding more ϕ-connected PMOSes to pull them up.

This can also be solved by adding a large load capacitance that can store voltages for long enough.

3.4 Process, Voltage, and Temperature (PVT) Variation

A lot of things can go wrong and we must tolerate some variation in device parameters to improve yield. The three main ones to consider are:

- **Process**: from the actual construction of the device
- **Voltage**: error in the supply voltage
- **Temperature**: due to operating conditions or heat disippation

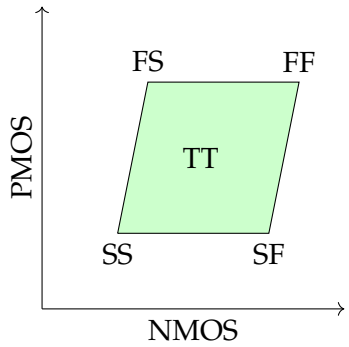


Figure 3.3: Process Corner Diagram, Axes in Increasing Speed

3.4.1 Process

The actual construction of the transistors is an enormously complex process and there are a lot of factors to consider. To the first order, these essentially change the channel length and threshold voltages of PMOS and NMOS. We usually address this by simulating **process corners** about the relative strengths of the two polarities as seen in Figure 3.3. The lower V_t or L is of a transistor, the faster it is relative to the other.

The middle point is the Typical-Typical case, whereas the other 4 are the corners (Slow-Slow, Fast-Slow, etc.). The information about these corners comes from the foundry who actually makes the chips, and they will say something like '99% of all chips will be greater than [some specification of the SS corner]'.

Chips that end up being in the FF corner can be 'binned' as more expensive and rated for higher frequency, and the SS ones can be sold for cheaper (though they should still be functional)

3.4.2 Voltage

Due to various interconnect or supply issues, V_{DD} can typically vary by about $\pm 10\%$. When it is *above* nominal voltage, the chip will be faster but higher power. The opposite is true when it is *below* nominal voltage

3.4.3 Temperature

Note the following relationships

$$\begin{aligned} Speed &\propto Current && (3.15)\\ &\propto Mobility && (3.16)\\ &\propto \frac{1}{Bandgap} && (3.17)\\ &\propto \frac{1}{T} && (3.18) \end{aligned}$$

Thus as the temperature decreases, we expect the circuit to operate *faster*.

Chapter 4

Sequential Logic

In combinational logic we designed circuits that implement some boolean function over a set of inputs. For a given set of inputs, the output is always the same. In **sequential logic** we product outputs dependent on the current inputs and the *previous* outputs or the 'state'. There are two styles of sequential logic: **Finite State-Machine** and **Pipeline**.

A sequencing element is some circuit that forwards tokens from its input to the output at a regular rate, usually controlled by a clock.

In FSM, there is some combinational logic that feeds into a sequencing element, which feeds back its state every clock cycle. In pipeline, we have N stages of combinational logic separated by sequencing elements. There are two fundamental sequencing elements to consider: latches, which are 'level-triggered' devices, and flip-flops which are 'edge-triggered' devices.

4.1 Latches and Flip-Flops

A **Latch** is a circuit with two phases: in the *transparent* (or *follow*) state, the output is equal to the input. In the *hold* state, the circuit holds its previous output independent of the input. A **positive latch** holds when the clock is low and is transparent when the clock is high. A **negative latch** holds when the clock is *high* and is transparent when it is low.

A **Flip-Flop** is a circuit which holds its value except for a small period around a clock **edge**. In a **positive edge-triggered flip-flop**, the output takes on the value of the input every positive (i.e. rising) edge of the clock signal. In a **negative edge-triggered flip-flop**, the output takes on the value of the input every negative (i.e. falling) edge of the clock signal.

The following sections show that if we know how to construct a positive latch, we can construct all polarities of latches and flip-flops. We can then look at how to implement the positive latch using CMOS technology and rely on these constructions to do the rest.

4.1.1 Construction of a Negative Latch

Given a positive latch, we can easily construct a negative latch by simply negating the clock signal.

4.1.2 Construction of a Positive Flip-Flop

If we have a way to implement both positive and negative latches, we can implement either polarity of flip flop. To implement a positive FF, we connect the input to a negative latch which feeds into a positive latch. Call these L_1 and L_2 respectively. Call the input of latch i D_i and its output Q_i. Consider a clock edge that arrives as in Figure 4.1. Before $t_{\phi 1}$, the clock is low and:

1. The negative latch is transparent so $Q_1 = D_1$

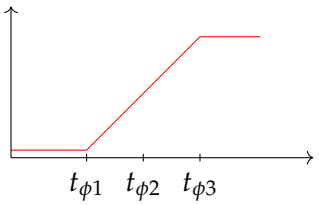


Figure 4.1: Clock Edge for Flip-Flop Construction

2. The positive latch is holding its previous value
3. The effect of 1 and 2 is that the overall circuit *holds* when clock=0

When the clock edge arrives ($t_{\phi 1}$ to $t_{\phi 3}$):

1. At some point *after* $t_{\phi 1}$, L_1 stops following and holds Q_1
2. At some point *before* $t_{\phi 3}$, L_2 begins following and $Q_2 = D_2 = Q_1$
3. The effect of 1 and 2 is that by $t_{\phi 3}$ the output will take on the value that L_1 ended up holding.

Thus we have implemented a positive flip flop that, provided inputs are stable prior to the clock edge, will register and hold the value provided.

4.1.3 Construction of a Negative Flip-Flop

The construction is similar to subsection 4.1.2, but with the clock signals negated (i.e. a positive latch followed by a negative one).

4.2 Latch Design

As seen in section 4.1, we need only focus on the implementation of positive latches. **In this section, the term 'latch' is equivalent to 'positive latch' unless otherwise specified**.

4.2.1 Dynamic Latches

Pass-Gate Latch

The simplest possible latch is a pass gate transistor, seen in Figure 4.2. It is fast, has low clock load, and only needs ϕ (not $\bar{\phi}$). One problem is that it has a V_{th} drop (see subsection 3.3.2), and it allows the output to **backdrive** the input. It also is dynamic and thus has the same pros/cons as most dynamic circuits.

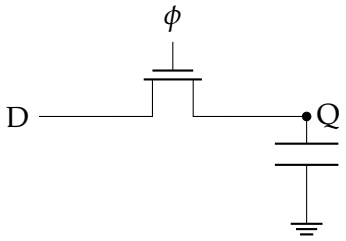


Figure 4.2: Pass-Gate Dynamic Latch

Transmission Gate

The transmission gate eliminates the V_{th} drop and requires $\bar{\phi}$. It still allows for backdriving: we can add a static inverter at the output to prevent this. This slows it down but does add isolation and level-restoration.

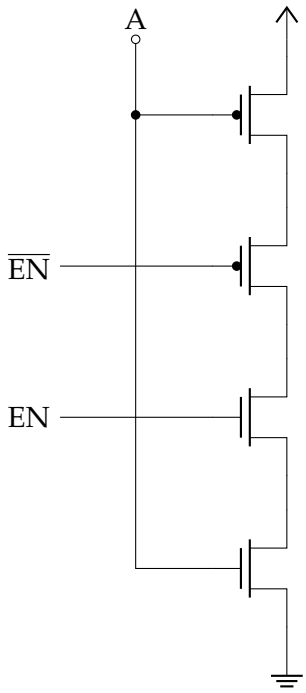


Figure 4.3: Tri-State Gate

4.2.2 Static Latches

A chain of two static CMOS inverters forms a static latch with some interesting properties. It is known as a **bi-stable** element since it has two stable states. The two states are logic 0 ($v_{i1} = v_{o2} = 0$) and logic 1 ($v_{i1} = v_{o2} = 1$). To find the properties of the circuit, we can overlay the VTC of each inverter on the same v_{o1} vs v_{o2} plot. What we find is a **butterfly curve** that has 3 solutions (only the 0 and 1 state are stable). In fact, the two states are self-reinforcing since a small perturbation will cause the state to return to the same value.

We still need some way to write values though, and this is acheived by overpowering the feedback mechanism that keeps the inverter pair in a stable state. This is determined by the point on the butterfly curve where $\frac{\partial v_{o1}}{\partial v_{i1}} = \frac{\partial v_{o2}}{\partial v_{i2}} = 1$

The length of the diagonals in the 'wings' in the butterfly curve determines the static noise margins for the bistable points.

MUX-Based Latches

We can implement a latch by a MUX whose select is ϕ. When ϕ is low, the MUX selects the first input, and when ϕ is high, the MUX selects the second input. By choosing D to be the first input and Q to be the second input, we get a negative latch. Similarly, if we choose D to be the second input and Q to be the first input, we get a positive latch.

Tri-State Gate

A tri-state gate is an efficient way to implement a MUX, seen in Figure 4.3.

4.2.3 Robust Static Latch

We omit the many variations of static latches and instead present the fully developed one in Figure 4.4.

It has 2 buffers: one for input and one for output. This provides isolation. During the hold stage, $\phi = 0$ the tristate inverter is enabled and thus we have the feedback chain storing the value of X. The transmission gate is off which isolates the node X from the D inverter. The output keeps the value of $\overline{X} = \overline{\overline{D}} = D$ and thus we can

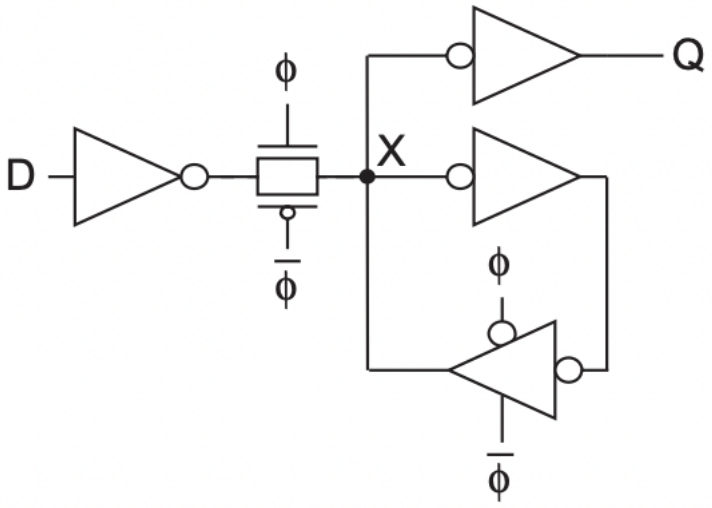


Figure 4.4: Static Latch

successfully hold. During the transparent stage, the tristate inverter floats and the transmission gate turns on, allowing $\overline{D}$ to be loaded into X while Q follows as $\overline{X}$ as before.

Note that we can reduce the clock load at the cost of robustness by using a weak inverter instead of a tristate one.

4.2.4 Enable and Reset Signals

An enable signal causes the output to hold when it is off, otherwise allows normal operation. A reset signal forces the output to a 0 when it is raised, otherwise the output is unaffected.

Enable Signal

In the MUX design, we may surround a latch with another MUX that feeds the current latch output when EN=0 and otherwise feeds in D. This increases the D to Q delay of the overall latch.

In the clock-gating approach we replace the clock signal with an AND2 over the EN and CLK signal. This adds some skew and increases the setup time of the circuit (the edge takes longer to get to the actual latch).

Synthesis tools generally prefer the MUX design as it doesnt cause clock skew.

Reset Signal

A reset may be synchronous or asynchronous. Synchronous resets can by done by replacing the input with an AND2 of D and $\overline{\text{RST}}$. When the reset is high, the and will be forced to 0 and the output will store a 0 on the *next* clock edge.

An asynchronous reset can be acheived by also putting such an AND2 on the inner node that stores the output (or comeplement of the output as in Figure 4.4).

If D is already being inverted (e.g. in Figure 4.4) we can use a NAND2 to do the AND and inversion at the same time.

4.2.5 Pulsed and TSPC Latches

Instead of sequencing logic with flip-flops, we could also use latches and simply give a short pulse to all of them. This is called a **pulsed** latch and its behaviour is similar to a flip-flop, but generating the pulse while maintining timing is a little trickier.

To generate a pulse we can feed a clock signal into a NAND2 twice, with one path going through $2k+1$ inverters. This causes some overlap between the edges of $\overline{\phi}$ and ϕ which will create a regular short pulse from a clock.

A **True Single-Phase Clock** latch is a clever way of implementing a latch with low clock load. It involves a precharged node and thus the setup/hold time analysis depends on the edge.

4.3 Timing

4.3.1 Setup and Hold

For a latch or flip flop, the **setup** time is the latest point *before* the clock edge that an input must be stable to ensure that the output gets the correct value. The **hold** time is the minimum time *after* the clock edge that a signal must be stable for the output to get the correct value. That is, in Figure 4.1, a signal must be stable for $t \in [t_{\phi 2} - t_{setup},\ \ t_{\phi 2} + t_{hold}]$ to correctly propagate to the output.

For example, recall the analysis of subsection 4.1.2. In that circuit, x, the point at which L_1 stops following D_1 would be related to the setup time as $t_{setup} = t_{\phi 2} - x$. The circuit would have **no hold time** since after x the inner node Q_1 is stable and at some point will be output by L_2.

4.3.2 Clock-to-Q and D-to-Q

The Clock or D signals have some delay to correspond to a change to the output Q. The D-to-Q value is particularly important in latches since Q may change right after a D change during the transparent state, whereas flip-flops cannot.

The **minimum** of each of these delays is called the corresponding **contamination** delay. The **maximum** of these delays is called the **propagation** delay.

The contamination and propagation delays can be thought of respectively as the earliest that Q starts 'glitching' and the latest point at which it will stabilize.

4.3.3 Logic Delay

Recall that sequencing elements are there to order the flow of information; that is, they are placed between or at the end of some combinational logic (either FSM or Pipeline as discussed in the section intro). Thus there is also some logic delay with a contamination and propagation value denoted t_{cd} and t_{pd} respectively.

4.3.4 Timing Constraints

The existence of the timing conditions described in subsection 4.3.1 combined with the existence of the delays in subsection 4.3.2 and subsection 4.3.3 leads to two conditions that must be obeyed for a circuit.

In this section we assume we are working with pipelined flip-flops, thus the **D-to-Q delay is irrelevant**. Furthermore, we focus on the case of a pair of flip-flops separated by some combinational logic. Call these flip-flops F_1 and F_2 respectively.

Maximum Delay Constraints

This is closely related to the setup time of a latch or flip-flop. Let the period of the clock be T.

When the clock edge arrives at F_1, it may or may not cause a change in Q_1. If it does not cause a change, then there is nothing to worry about as the combinational logic receives the same input, produces the same output, and thus D_2 is stable long before the next clock edge. If it *does* cause a change, it may do so **as late as** the delay t_{pcq} (propagation clock to Q). The corresponding change in the combinational logic may take as long as t_{pd}. Thus the total time from clock edge 1 to D_2 is $t_{pcq} + t_{pd}$. This time by definition must be less than the setup time for the next clock edge. This is how we get the **max-delay constraint** of Equation 4.1.

$$t_{pcq} + t_{pd} \leq T - t_{setup} \tag{4.1}$$

Note that since $f = \frac{1}{T}$, Equation 4.1 gives us the maximum frequency we can run the circuit at, and we can solve this delay simply by slowing down the clock.

Minimum Delay Constraints

This is closely related to the hold time of the flip-flop. It is much harder to remedy as T is not a term in the constraint.

A clock transition may cause a Q_1 change as soon as t_{ccq}. The combinational logic, receiving this change, may cause the value of D_2 to change as soon as t_{cd} after. This total time must be **greater** than the hold time otherwise the data has changed too soon after the clock edge. The constraint is given by Equation 4.2.

$$t_{ccq} + t_{cd} \geq t_{hold} \tag{4.2}$$

The only way to fix this is to redesign the circuit.

4.4 Clock Uncertainties

Ideally, all clock inputs receive the exact same clock signal with period T. In reality, there are many complications that may mess with the clock signal. We consider the two most important: skew - where clock paths have a fixed offset delay relative to eachother, and jitter - where each clock edge may arrive slightly sooner or later than expected in a way that varies over time.

There are many ways to mitigate these issues such as clock H-trees and time borrowing but we do not discuss any of these in detail.

4.4.1 Skew

Skew comes from the layout of each path to the system clock source. Each path may have different RC characteristics and thus each path has a fixed delay relative to each other path. This is **constant** for a given path and does not affect the period.

The skew δ for the circuit of subsection 4.3.4 is t_{ϕ_2} - t_{ϕ_1} where t_{ϕ_i} is the time at which any clock edge at flip-flop i arrives. Thus it may be positive or negative - each has different implications on the timing constraints of subsection 4.3.4.

Positive Skew

If the skew is positive, the clock edge at F_2 later than it did at F_1. This means that Equation 4.1 is relaxed since the signal must only arrive at D_2 by $T + \delta - t_{setup}$ rather than $T - t_{setup}$. The full constraint is given by Equation 4.3.

$$t_{pd} + t_{pcq} + t_{setup} - \delta \leq T \tag{4.3}$$

On the other hand, Equation 4.2 is worse since we must now hold for d time units longer. The full constraint is given by Equation 4.4.

$$t_{ccq} + t_{cd} \geq t_{hold} + \delta \tag{4.4}$$

Negative Skew

This is essentially the inverse of positive skew. It negatively affects the maximum delay constraint because the second clock edge arrives *sooner*. It positively affects the minimum delay because we do not need to hold as long.

The nice thing is that if we simply keep our definition of δ, Equation 4.3 and Equation 4.4 still hold for negative skew.

4.4.2 Jitter

Jitter randomly causes edges to arrive sooner or later than usual. The maximum value by which this occurs is t_{jitter}. We need to consider the worst case. For maximum delay, the worst case is that ϕ_1 arrives late and ϕ_2 arrives early. Adding this to Equation 4.3 results in Equation 4.5. For minimum delay, the worst case is that ϕ_1 arrives early and ϕ_2 arrives late. Adding this to Equation 4.4 gives Equation 4.6.

$$t_{pd} + t_{pcq} + t_{setup} \leq T + \delta - 2t_{jitter} \tag{4.5}$$

$$t_{ccq} + t_{cd} \geq t_{hold} + \delta + 2t_{jitter} \tag{4.6}$$

Chapter 5

Adders

Adders are often the speed bottleneck in something like an ALU so it is worth looking at optimizing them heavily. This can occur at the circuit or logic level.

5.1 Full Adder

A full adder takes in bits A_i, B_i, C_i and outputs C_o, S (carry and sum). We can write the functions for the outputs as Equation 5.1 and Equation 5.2

$$C_o = AB + C_iA + C_iB = AB + C_i(A + B) \tag{5.1}$$
$$S = A \oplus B \oplus C_i = \overline{C_o}(A + B + C_i) + ABC_i \tag{5.2}$$

Note that C_i arrives later than A and B so it is useful to factor it out and optimize for it (e.g. setting it as the inner input). The value for the sum reuses $\overline{C_o}$ - why? Because the critical path in an adder is the chain of C_o so we might as well reuse it to compute S to reduce the overall area.

The properties Equation 5.4 and Equation 5.3 are very important, as they allow us to perform more optimizations.

$$\overline{C_o}(A, B, C_i) = C_o(\overline{A}, \overline{B}, \overline{C_i}) \tag{5.3}$$
$$\overline{S}(A, B, C_i) = S(\overline{A}, \overline{B}, \overline{C_i}) \tag{5.4}$$

We also define the mutually exclusive **G**enerate, **P**ropagate and **K**ill signals. When the adder generates, its carry out depends only on the bits A_i and B_i. When it propagates, its carry out is equal to its carry in independent of A_i and B_i. When it kills the carry out is 0.

$$G = AB \tag{5.5}$$
$$P = A \oplus B \tag{5.6}$$
$$K = \overline{A + B} \tag{5.7}$$

5.2 Ripple Carry-Adder

To add N bits, we chaing N full adders together. The critical path is the A or B input of bit 0 to the S of bit $N - 1$. Thus the overall delay is

$$t_{add} = (N-1)t_{carry} + t_{sum} \tag{5.8}$$

The circuit involves 3 sections. First, using A, B, C_i we generate $\overline{C_o}$. This value feeds into 2 things: an inverter to generate C_o, and the gate that generates S.

5.3 Inversion

Given the inversion properties (Equation 5.3 and Equation 5.4) we can improve the worst case delay by removing the inverter that converts the intermediate $\overline{C_o}$ into C_o. Bit 0 generates $\overline{C_o}$ (i.e. it doesnt add the inverter) which is fed into bit 1's C_i. Bit 1 also inverts the inputs A_i, B_i and thus generates $\overline{C_o(\overline{A_i}, \overline{B_i}, \overline{C_i})} = C_o$. Thus by the inversion properties the output of bit 1 will be $\overline{S_1}$ and C_{o1}.

Essentially, every other full-adder takes input $\overline{A_i}, \overline{B_i}$ and generates $\overline{S_i}$ which must then be inverted. This clears the carry-out path of inverters, improving the worst case delay.

5.4 Mirror Adder

A mirror adder is a better structure to implement the inverting adder at the transistor level. We use the form $C_o = (A+B)(C_i + AB)$ to implement the **PDN** and give the **PUN** a symmetrical structure. This reduces the transistor stack and thus lets us use narrower transistors, which reduces load capacitance. The C_i bit should also be closest to the output (see subsection 3.2.1).

5.5 Carry-Bypass Adder

Carry-bypass adders are used when we have large words to add. We group each k bits together and compute group generate/propagate signals and add a MUX on the output of the block. The MUX takes signal $\prod_i P_i$ to output the block carry in. That is, if all the elements of the block are propagating, we can immediately forward the overall carry in to the output of the block.

If $\prod_i P_i$ is not true, then at least one of the block bits must be the one that generates the carry-out. Thus the worst case path is the time for a carry in in the 0th bit of block 0 to carry to its block output, and then bypass every other block until the end. Once it reaches the last block it must travel through each bit to the end and also factor in t_{sum}.

For example, the overall delay for a block size of 4 given N bits is given by Equation 5.9.

$$t_{add} = t_{setup} + 4t_{carry} + (N-1)t_{bypass} + (3t_{carry} + t_{sum}) \tag{5.9}$$

Chapter 6

Memory

6.1 RAM Structure

RAM is an array of N M-bit words. To index them we use a **row decoder** that takes in $\lg(N)$ 'address bits' and outputs N wordlines. The worldlines are 'one-hot' in that exactly one of them is a 1. The M-bit word is grouped into blocks of 2^K bits. Each block outputs all 2^K bits accessed by a wordline, which is then MUXed by a 2^K-to-1 **column MUX**.

The notation used is that there are L total address bits. K bits go to the column decoder to select each bit of the word. L-K bits go to the row decoder inputs. The dimension of each block is thus 2^{L-K} rows by 2^K columns, for a total cell count of $2^{L-K} \cdot 2^K = 2^L$. For the overall RAM then, the number of cells/bits is $M \cdot 2^L$.

To lay out N M-bit words we do the following:

1. $L = \lg N$
2. $\text{Rows} = \text{Columns} = \sqrt{MN}$
3. $K = L - \lg(\text{Rows})$

This is all the information we need to show the widths of everything in the memory structure.

6.1.1 Row Decoders

The wordlines generated by the row decoders are attached to all $M \cdot 2^K$ cells. Thus they may have very high load capacitances so we need to do some design to get a fast access.

Basic Operation

Consider the expression for wordline i. It consists of a conjunction of the input address bits corresponding to the binary representation of i. For example, when $L - K = 2$ (4 wordlines):

$$WL_0 = \overline{A_3}\ \overline{A_2}\ \overline{A_1}\ \overline{A_0} \tag{6.1}$$

$$WL_{10} = A_3\ \overline{A_2}\ A_1\ \overline{A_0} \tag{6.2}$$

$$\tag{6.3}$$

Pre-Decoding

In the previous example, we can observe that one quarter of the wordlines will need the signal $\overline{A_3}\ \overline{A_2}$ while a quarter will use $\overline{A_3}\ A_2$ and so on. Thus to pre-decode we can compute pairwise signals to form a stage which gets fed into NOR. For example:

$$
\begin{aligned}
WL_3 &= \left(\overline{A_3}\ \overline{A_2}\right)\left(A_1\ A_2\right) \\
&= \overline{\overline{\overline{A_3}\ \overline{A_2}} + \overline{A_1\ A_2}} \\
&= \text{NOR}(\text{NAND}(\overline{A_3}, \overline{A_2}) + \text{NAND}(A_1, A_2))
\end{aligned}
$$

This is just a small example, we can keep adding more pre-decoding levels for higher N.

6.1.2 Column MUX

Essentially we implement a decoder for 2^K and feed each of these lines to the gate of an NMOS. These NMOS are connected to the **bitlines** which all join at the output of the current block. Thus exactly one column is connected to the output at a time.

6.1.3 Sense Amplifiers

At the output of the column blocks we need to amplify any voltage swings using an analog circuit known as a sense amplifier. Essentially a small δv gets amplified to a full rail-to-rail swing under sufficient conditions.

6.2 Memory Cells

6.2.1 SRAM

Static RAM has a low activity factor, has low density but is very robust. The standard configuration is called 6T SRAM since it uses 6 transistors. The inner cell is a pair of feedback inverters (subsection 4.2.2) connected to bitlines by **access transistors**. The access transistors' drains to the bitlines BL and $\overline{\text{BL}}$. The access transistors' gates are connected to the wordline.

Each cell provides 2 access gate capacitances ($2C_gW_{access}$) and 1 drain capacitance on each bitline (C_dW_{access}).

Read Operation

1. BL and $\overline{\text{BL}}$ precharge to V_{DD} (all other bitlines charge to 0)
2. Turn on the wordline
3. Current will discharge through one of the inverters' NMOS (the other line will already be high so no current flows)

To ensure that the storage node doesnt float too high during this process, we must ensure that the access transistors are **weaker** than the inverters' NMOS.

Write Operation

1. Precharge bitlines
   - To write a 1, BL high and $\overline{\text{BL}}$ low
   - To write a 0, BL low and $\overline{\text{BL}}$ high
2. Turn on the wordline
3. One of the nodes will need to discharge through the access transistor while avoiding being pulled up through the PMOS

To ensure that the storage node is overridden, the PMOS must be weaker than the access transistor. **Note that this includes the mobility factor** and not just the width. If $\mu_n = 2\mu_p$ then equal widths of PMOS and NMOS will satisfy this requirement.

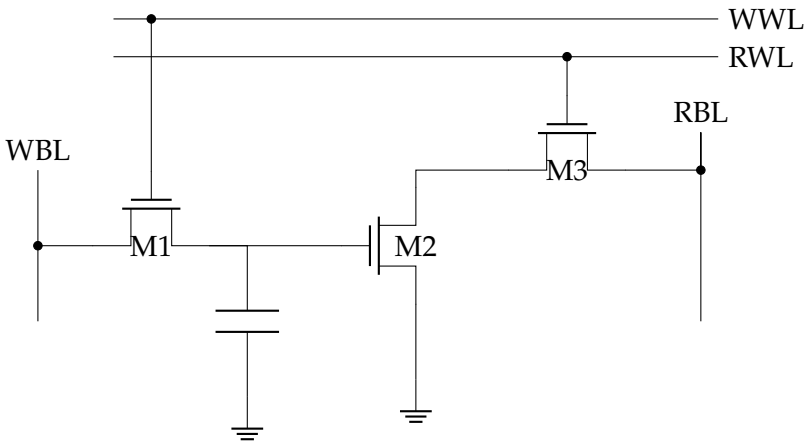


Figure 6.1: A 3T DRAM Configuration

6.2.2 DRAM

Dynamic RAM is higher density, cheaper, less robust, and uses more power. It uses an enhanced source capacitance to store information. This is acheived through a special process which is why memory is manufactured in separate fabs (basically dig a well at the transistor source to increase the capacitances).

1T DRAM

Connect the NMOS gate to the wordline and its drain to the bitline. The enhanced capacitance C_s is connected to ground. To read, precharge $\frac{V_{DD}}{2}$ and then connect the bitline. If it is a 0, the bitline will be driven low as charge redistributed between the bitline capacitance and C_s. This difference will be amplified by the sense amplifiers.

To write a value, we force the bitline to the desired value and turn on the wordline.

Reads are destructive - this process also moves charge from C_s which may flip the state. Most DRAM is write-after-read to refresh the values (though the latency of the write can be hidden).

3T DRAM

The ciruit is shown in Figure 6.1. This is less higher density than 1T DRAM but prevents the destructive read. To write, we precharge the WBL (write bitline) and then turn on the WWL (write wordline). To read, we precharge the RBL to V_{DD} and then turn on the RWL. If the capacitor is charged, M2 will be ON and the output will be driven low - the circuit stores $\overline{X}$. If the capacitor is not charged, the bitline will not change value. The only leakage is through the transistor gate which is fairly low.

Chapter 7

Integrated Circuit Testing

The goal is to improve the yield of circuits. It is not unusual to only get a ∼80% yield in modern processes, whereas typical consumer requirements are just 5-50ppm defective. The field of IC testing is vast and is becoming bigger, here we only consider a few basic techniques for validating them.

The error model we assume is that of *stuck-at* faults. There are two types: stuck at 0 or stuck at 1. A stuck-at fault is usually a layout issue and causes a node to be a constant 0 or 1 regardless of the inputs (i.e. it is tied to V_{DD} or GND).

7.1 Combinational Logic

We can design **test vectors** of input-output pairs that are designed to detect stuck-at faults at particular nodes. Consider the example in Figure 7.1. A common assumption is that there may be at most one stuck-at fault - consider the case of Y being stuck at 1. If we use $(A, B, C, D) = (1, 1, 0, 0)$, then $f = 0$. If Y were stuck at one however, f would be 1. This test vector thus depends on Y *not* being stuck at one - if we run it and it is successful, we know Y is *not* stuck at one. If it fails however, we may not conclude that it is because Y is stuck at one - it could be C or D too.

Using this method we can design $2N$ vectors for an N-node circuit to cover all possibilities. This is better than the 2^N exhaustive vectors and it comes from the assumption that only one stuck at fault can occur at a time.

7.2 Sequential Logic

The method of section 7.1 cannot be used in sequential circuits. The focus here is pipelined circuits (chapter 4). What we do is design a **scan chain** by surrounding every flip-flop with a 2-to-1 MUX at its input (known as a **scan structure**). The MUX is enabled by a 'scan-enable' signal and if it is a 1 the input to the flip-flop comes from the scan-in signal. The scan in signal for a register in a pipeline comes from the output of the previous stage.

Using this, we can manually set the state of N registers in N clock cycles by feeding each bit to the first register.

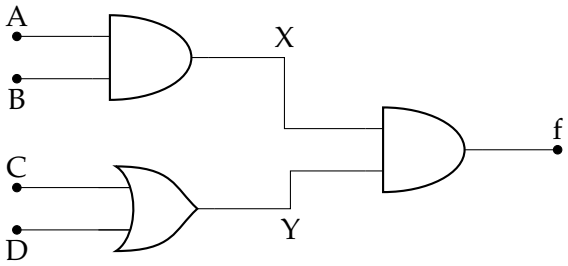


Figure 7.1: Example for Combinational Logic