UNIT-3

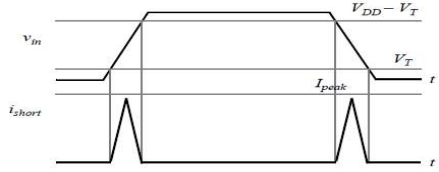
Device and Technology Impact on Low Power Electronics

Device & Technology Impact on Low Power:

Dynamic Dissipation in CMOS:

Consider a CMOS logic gate. The power dissipation consists of a static and dynamic component.

Dynamic power is usually the dominant component and occurred when the node voltage is switched.



$$P = P_{dynamic} + P_{static} \qquad P_{dynamic} = CV_{dd}^2 \alpha f + P_{sc} \qquad P_{dp} = t_{sc} V_{DD} I_{peak} f$$

The short-circuit power (Psc) occurred when the inverter input is around Vdd/2 during the turn-on and turn-off switching transients, both the PFET and the NFET are on and a short circuit current Isc flows from Vdd to ground.

The width of this short-circuit current pulse is about 1/4 of the input rise and fall time.

$$P_{sc} \approx V_{dd} I_{sc} \frac{\tau_{in}}{4} 2f \approx V_{dd}^2 f \frac{C}{10} \qquad \text{----------2}$$

Where τin is assumed to be $\tau_{in} \approx \tau_{out} \approx V_{dd} C / I_{dsat} \approx V_{dd} C / 5 I_{sc}$

On Combining Eqn 1 and 2. $$P_{dynamic} = CV_{dd}^2 \alpha f + V_{dd}^2 f \frac{C}{10}$$

➔ $CVdd^2 \alpha f(1+1/10\alpha)$

$$P_{dynamic} = kCV_{dd}^2 \alpha f$$

$$\text{Switching Energy, } E = kCV_{dd}^2$$

Where C is

C = Oxide capacitance + Junction capacitance + InterConnect capacitance

$$C = C_{ox} + C_j + C_{int} = \frac{b}{T_{ox}} + C_j + C_{int}$$

Cox=εA/D

f is the clock frequency. *αf is the average rate of cycling this node experiences. For example, an* **idle block of the circuit may not experience switching because the clock signals to the function blocks are gated.**

Effects of Vdd and Vt on Speed

WKT Pd = $kCVdd^2\alpha f$

Reducing Vdd in in the above eq. is an obvious and very effective way of reducing CMOS power. However, lower Vdd for a given device technology, leads to lower gate speed.

Parallelism can be applied to compensate for speed loss to realize a large net reduction in power. On the other hand, optimizing the technology for a lower Vdd could minimize or eliminate the speed loss due to Vdd reduction.

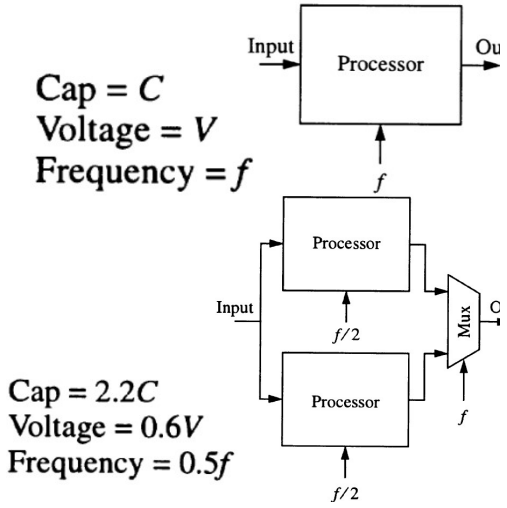
SPICE simulations confirm that the gate delay may be expressed as

$$\tau = \frac{CV_{dd}}{4}\left(\frac{1}{I_{dsat\,n}} + \frac{1}{I_{dsat\,p}}\right)$$

τ =RC=CR=C(V/I)

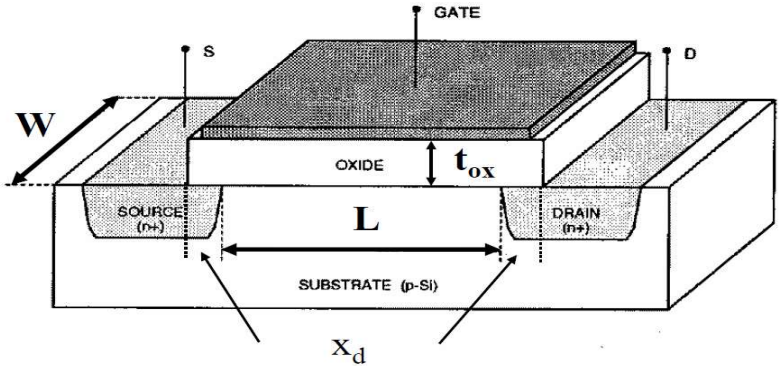
$$\approx \frac{CV_{dd}}{4I_{dsat\,n}}\left(1 + 2.2\frac{W_n}{W_p}\right)$$

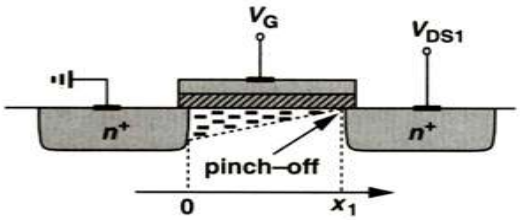
If Wn = Wp then Idsatn = 2.2Idsatp



τ may be interpreted as the average of the time for the NFET saturation current, Idsatn to discharge C from Vdd to Vdd/2, and the time for Idsatp to charge C from zero to Vdd/2.

The behavior of transistors with very short channel lengths deviates considerably from long *channel devices*





$$E_{ds} = \frac{V_{ds}}{L}$$

Velocity(v) = μnEds

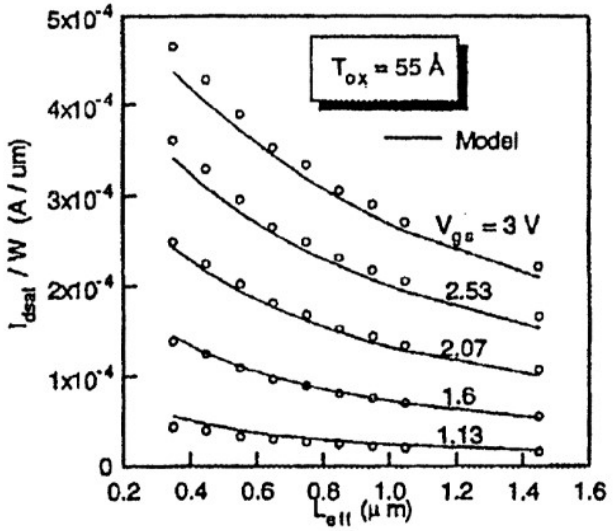
$$\upsilon_n = -\mu_n \xi(x) = \mu_n \frac{dV}{dx}$$

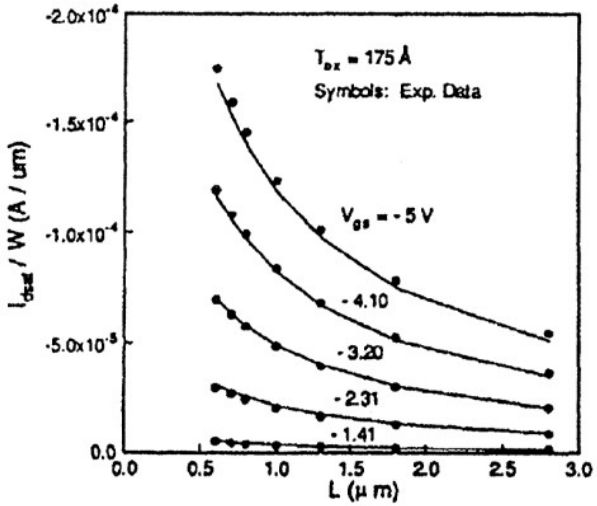
When the electrical field along the channel reaches a critical value ξ_C, *the velocity of the carriers tends to saturate* due to scattering effects (collisions suffered by the carriers)

For Short channel transistors, Idsat=WVsatCox(Vg-Vt)

where W is the channel width, and Vsat is the velocity saturation

Figure shows variation of Idsat/w v/s Channel length





(a)Plot of Idsat/w v/s Effective Channel length(Leff), (b) Plot of Idsat/w v/s Channel length(L

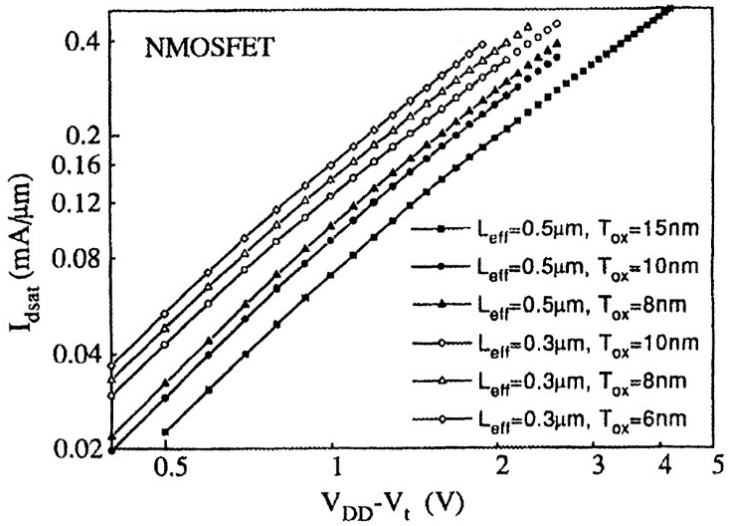


Fig: Plot of Idsat v/s (Vdd-Vt)

Constraints on Vt Reduction

WKT dynamic power can be reduced by minimizing Vdd, but Vdd cannot be minimized beyond certain limits. If Vdd is kept larger than 4Vt the speed is not suffer excessively(i.e effect of speed reduction is minimal).

The lower bound of Vt is set by the subthreshold static current that the transistors conduct at Vgs = 0V. At 50 C.

Every 0.1V reduction in Vt raises I_{static} by 10 times.

In order to accommodate low Vdd, keep Vdd/Vt > 4, lower Vt 's may be tolerated by accepting higher static current or use circuit techniques to raise Vs or Vt (by raising the body potential) in idle circuit blocks.

We may eventually see Vt's as low as 0.2V for 1 V circuits. It is also likely that more than one (fixed) Vt may be used on the same chip.

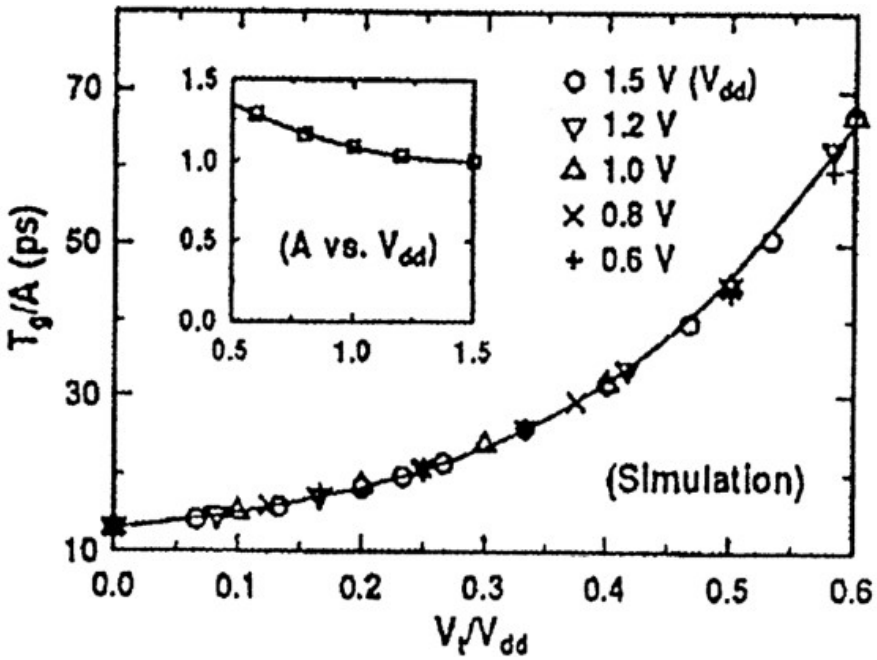


Fig: Gate delay of a CMOS inverter as a function of Vt/Vdd

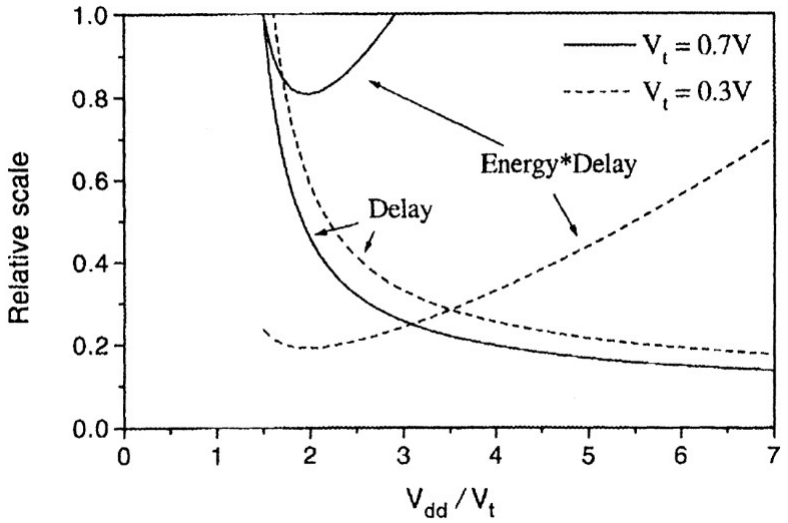


Fig: Speed is independent of Vdd as long as Vt is linearly scaled with Vdd

Transistor sizing & gate oxide thickness

For a given Wn + Wp there is a certain Wp/Wn ratio that minimizes τ(delay) and τE(delay energy). This optimal ratio is independent of the load capacitance.

$$\frac{W_p}{W_n} = \sqrt{2.2} = 1.5$$

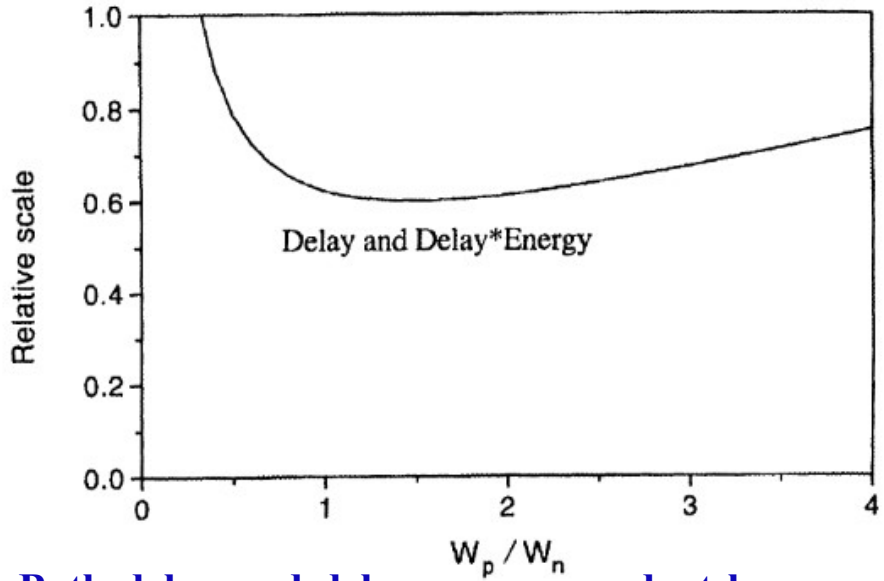


Fig: Both delay and delay-energy product have a very broad minimum around Wp/Wn= 1 to 3 independent of load capacitance.

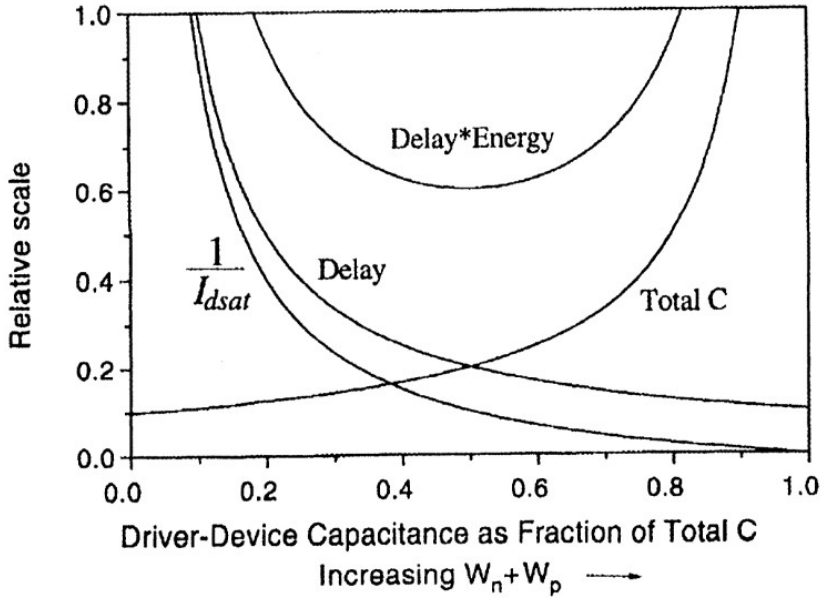


Figure: Delay decreases monotonically with increasing Wn+Wp but the delay-energy product is minimized when the drive devices contribute half of the total load capacitance.

Figure indicate that τE is minimized when the total oxide capacitance is 1/4 (i.e.0.25) of the load capacitance.

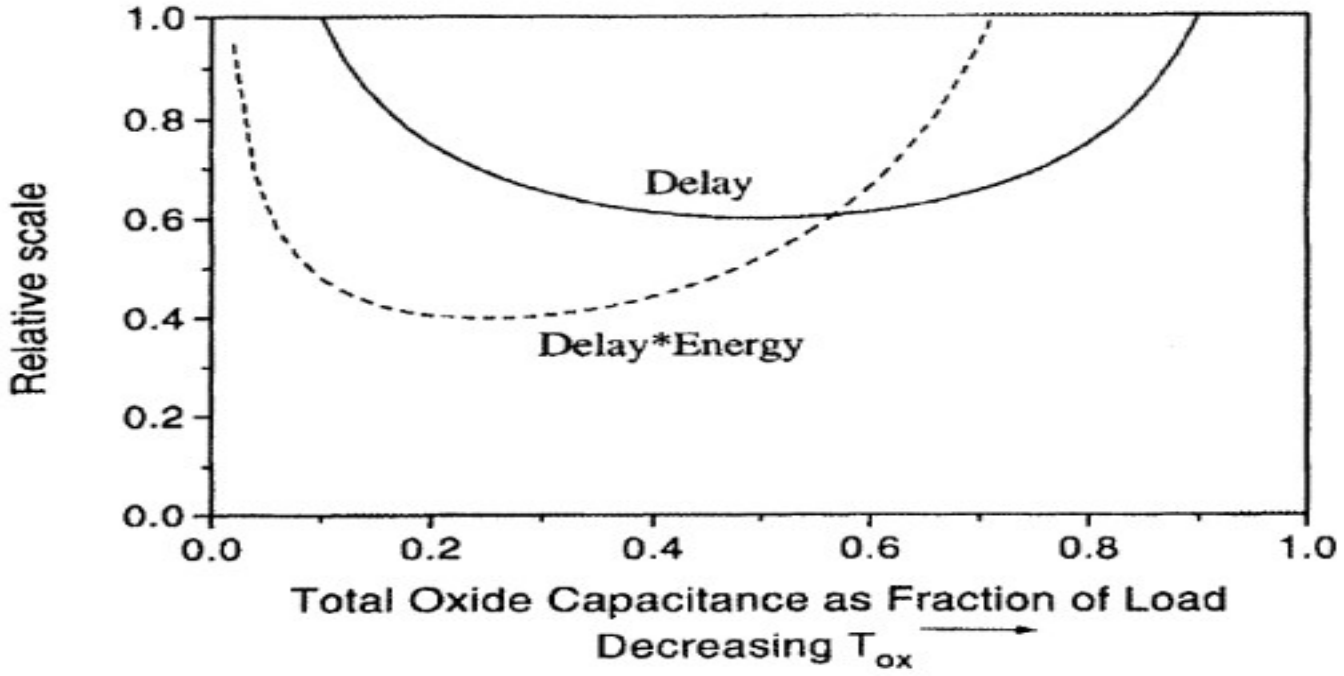


Figure 2.7 Minimum delay is obtained when T_{ox} is chosen such that oxide capacitance accounts for half the total load. The minimum delay-energy product requires thicker T_{ox} such that $^1/_4$ of the load is attributable to oxide capacitance.

The values of various parameters for Delay and Delay-Energy Product Optimization is summarized in Table

Delay: $\tau = \frac{CV_{dd}}{4}\left(\frac{1}{I_{dsatn}} + \frac{1}{I_{dsatp}}\right)$, Energy: $E = CV_{dd}^2$

	L	V_{dd}	T_{ox}	W_p/W_n	W_p+W_n
to minimize τ	min	max, $>4V_t$	$C_{tox}=\frac{C}{2}$	1-3	max
to minimize τE	min	$2V_t$	$C_{tox}=\frac{C}{4}$	1-3	$C_d=\frac{C}{2}$

C: total load capacitance, C_{tox}: all load capacitances attributable to gate oxide, C_d: load capacitance attributable to driver devices.

Table: Optimization for Delay and Delay-Energy Product

Impact of Technology Scaling

The Impact of scaling Vdd, L, and Tox on MOSFET current and inverter speed is shown in Table. Note that the PFET current remains to be about half the NFET current.

Gate Length (μm)	3	2	1.5	1	0.7	0.5	0.35	0.25	0.18	0.12
V_{dd} (V)	5	5	5	5	5	5/3.3	3.3/2.5	2.5/2.0	1.5	1.5
V_T (V)	0.7	0.7	0.7	0.7	0.7	0.7/0.7	0.7/0.6	0.6/0.5	0.4	0.4
T_{ox} (nm)	70	40	25	25	20	15/10	9/7	6.5/5.5	4.5	4
I_{dsatn} (mA/μm)	0.1	0.14	0.23	0.27	0.36	0.56/0.35	0.49/0.40	0.48/0.41	0.38	0.48
I_{dsatp} (mA/μm)		0.06	0.11	0.14	0.19	0.27/0.16	0.24/0.18	0.23/0.19	0.18	0.24
Inverter Delay (ps)	800	350	250	200	160	90/100	70/65	50/47	40	32

Table: Impact of Vdd, L, and Tox scaling on MOSFET current and inverter speed

Figure shows the projected trend of *Idsat with technology scaling.*

Idsat increases with scaling upto 0.5 um and it ceases to rise with scaling beyond the 0.5um because of velocity saturation. *In* the high-speed scenario the oxide field strength(Eox) rises due to higher voltage when compared to low power voltages in the graph.

$$E_{gc} = \frac{(V_{gs} - V_t) - \frac{Vds}{2}}{D}$$

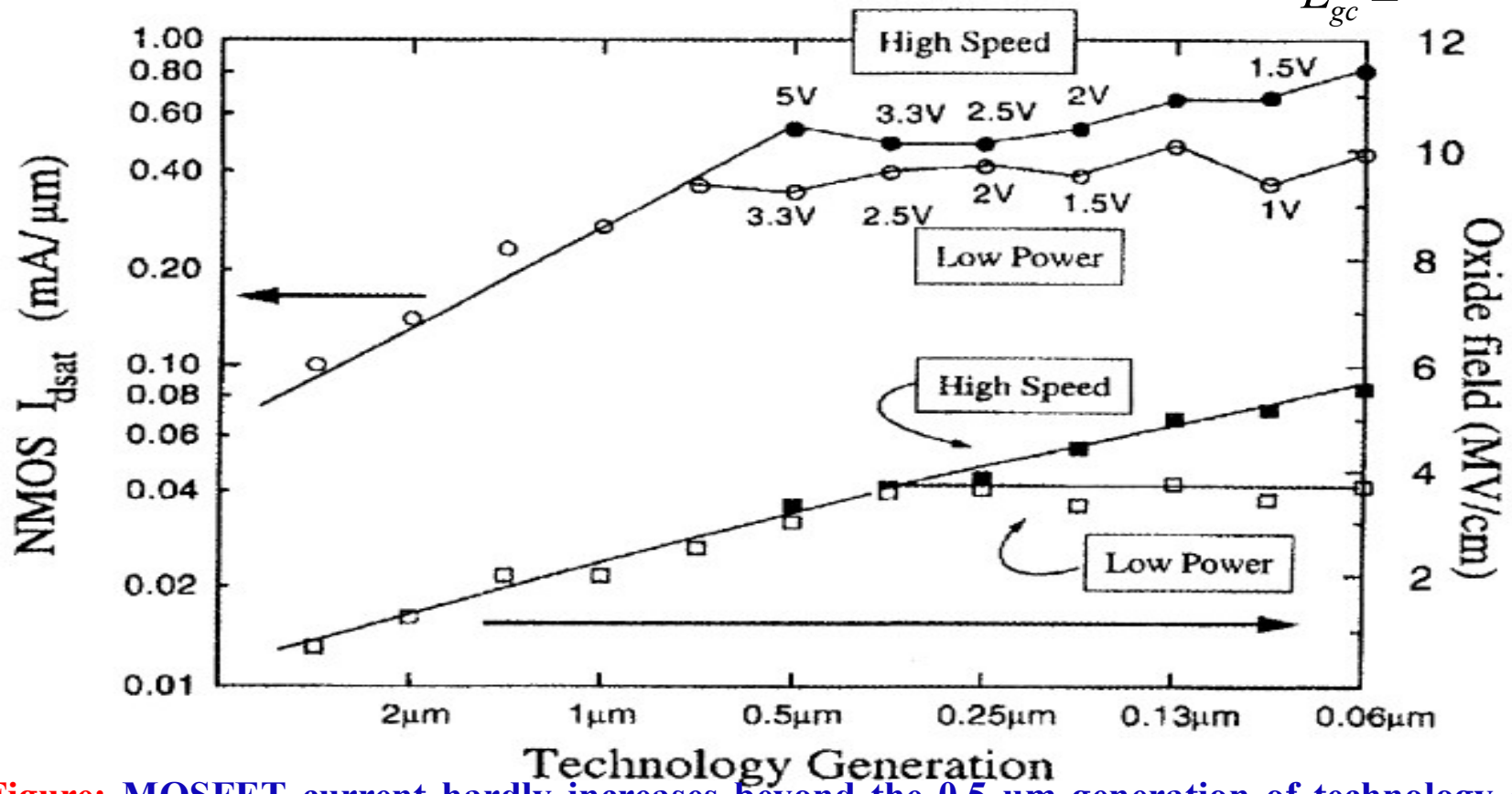


Figure: MOSFET current hardly increases beyond the 0.5 um generation of technology due to velocity saturation even in the high-speed scenario, where oxide field rises aggressively.

Figure shows the projected inverter delay. The trend of speed doubling every two generations will slow down slightly velocity saturation. Speed in the low-power scenario is slightly lower than in the high-power scenario.

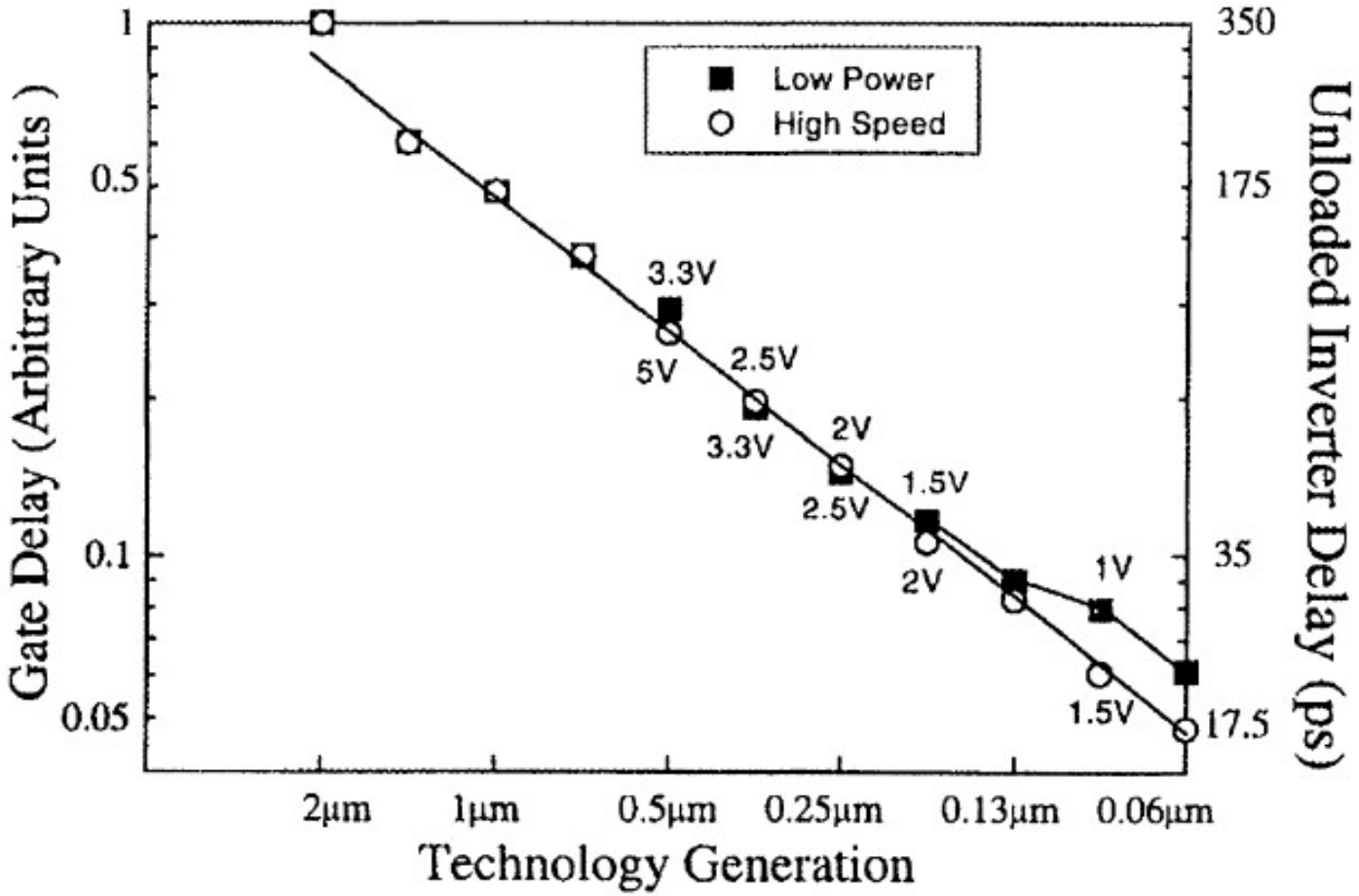


Figure: Speed in the low-power and high-speed scenario

Technology and Device Innovations

Many devices are proposed for high-speed and low-power based on:

- **Quantum tunneling,**
- **Single-electron effect**

These devices have excellent intrinsic switching speed and energy, but they are not capable of :

- **Driving the capacitance of long interconnects.**
- **Difficulty of manufacturing,**
- **No suitable circuit architectures that are compatible with the characteristics of these devices.**

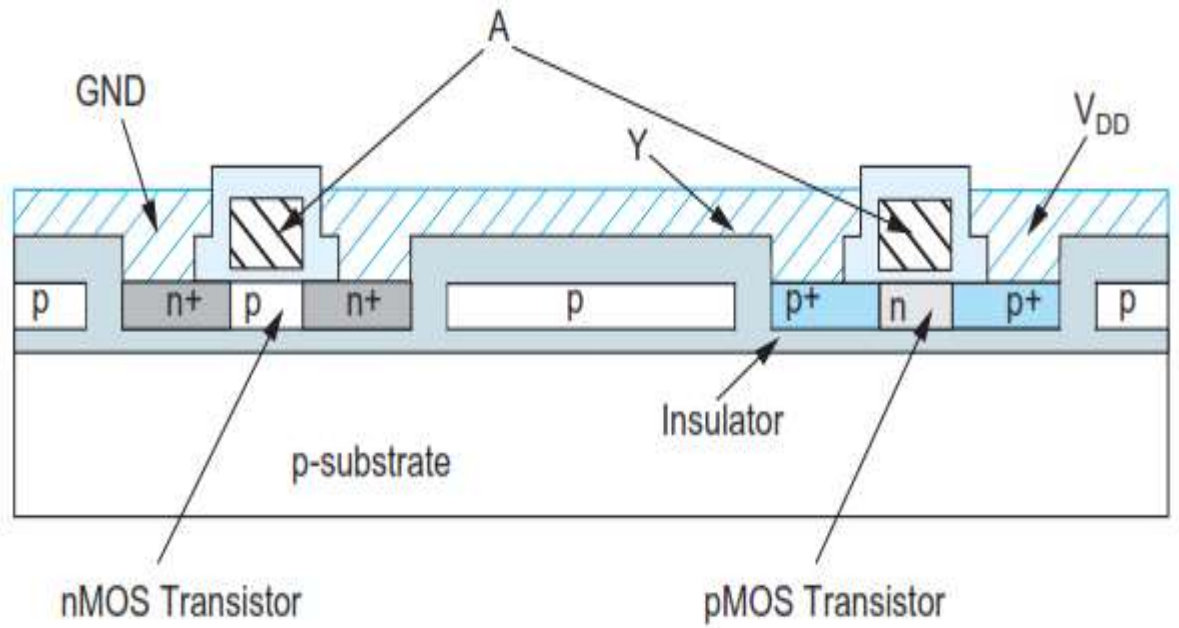
The innovations in silicon CMOS technology together with voltage and device scaling discussed earlier, will have more advantages in the low power electronics industry.

Silicon-On-Insulator (SOl) technology can improve delay and power with around 25% reduction in total capacitance .

The fundamental difference between SOI and conventional bulk CMOS technology is that the transistor source, drain, and body are surrounded by insulating oxide rather than the conductive substrate or well (called the *bulk). Using an insulator eliminates most of the* parasitic capacitance of the diffusion regions. However, it means that the body is no longer tied to GND or *Vdd* through the substrate or well.

Figure below shows a cross-section of an inverter in a SOI process. The process is similar to standard CMOS, but starts with a wafer containing a thin layer of SiO2 buried beneath a thin single-crystal silicon layer.

Fig: SOI inverter cross-section



SOI substrates are produced by either wafer bonding or SIMOX (Separation by IMplantation of OXygen).

In wafer bonding, two bulk silicon wafers are oxidized and bonded at a moderately high temperature.

In SIMOX, oxygen is ion implanted into a Si substrate at 150keV

Optimized SOl MOSFETs can have a

- **Lower capacitance**
- **Slightly higher Idsat than bulk devices (because of the reduced body charge and a small reduction in the minimum acceptable Vt).**
- **Some improvement in layout density,**

This results in up to 40% improvement in speed as Figure. That is two generations' worth of speed advantage. This may be used for lower Vdd and obtain up to a factor 3 in power savings.

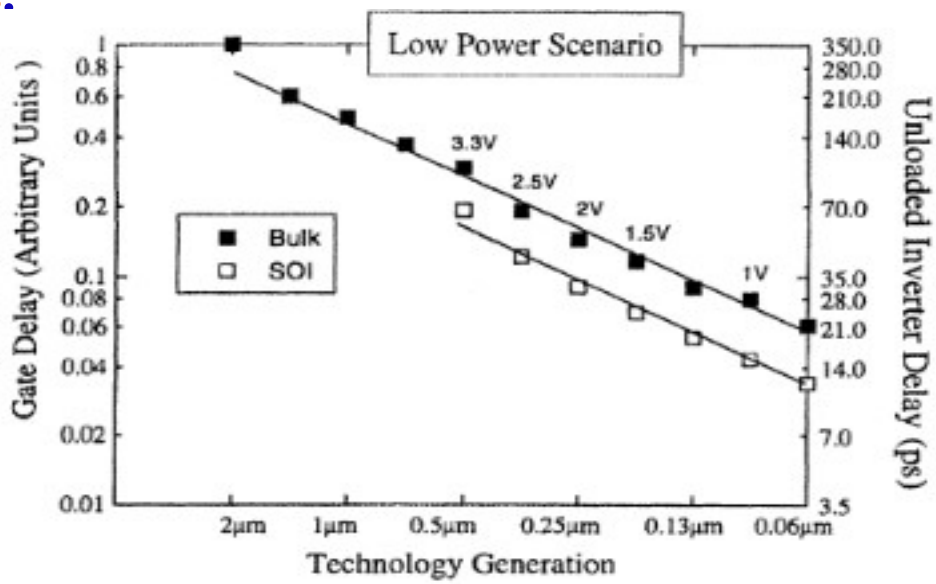


Fig: Potential speed advantage of SOl technology. Speed may be traded for lower Vdd to obtain a 3-fo1d power reduction.

Another way to reduce capacitance is:

Reduce the width of metal interconnects that carry ac signals such as the clock and data busses.

This is due to ac interconnects can carry a many times higher current density than dc interconnect, this is due to a self-healing effect (electromigration lifetime is higher under ac stress than dc stress).

Similar behavior has been reported for vias and other metal systems.

Another possible way to reduce metal capacitance would be to use low-permittivity insulators for inter-metal dielectrics.

Relative to the CVD Sio2 ε=4.2, SiOF(Fluorine doped Silicon Oxide) has an ε= 3.3 and organic polymers may achieve ε= 2.5.

For very low power/voltage electronics design, Dynamic Threshold MOS(DTMOS) transistor can be used which works at Vdd lower than 0.5 V, Vt lower than 0.15 V, also operates at higher speed.

DTMOS demonstrates that a MOSFET can have a higher Vt at input gate voltage Vg = 0 for low leakage current and a lower Vt at Vg = Vdd for high speed. In other words, Vt may be a function of Vg.

This(variable Vt) is accomplished by connecting the floating body of an SOI MOSFET to the gate as shown in Figure. At Vg = 0, Vt is the usual threshold voltage, e.g. 0.35V. As Vg rises, Vt falls to 0.1 V due to the (reverse) body effect.

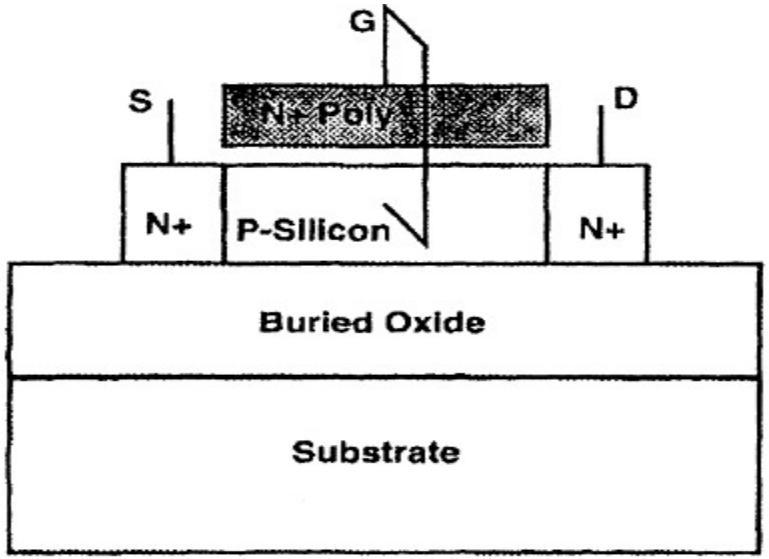


Fig: Schematic of DTMOS, an SOI MOSFET with body connected to the gate.

In DTMOS, the gate and body are tied together, so the body voltage follows the gate voltage

- **When gate voltage is high, body voltage (Vsb)is also high → lowers Vt**
- **When gate voltage is low, body voltage (Vsb)is also low → increases Vt**

Figure shows a speed comparison between CMOS and DTMOS as obtained from simulations. A threefold speed improvement may be achieved.

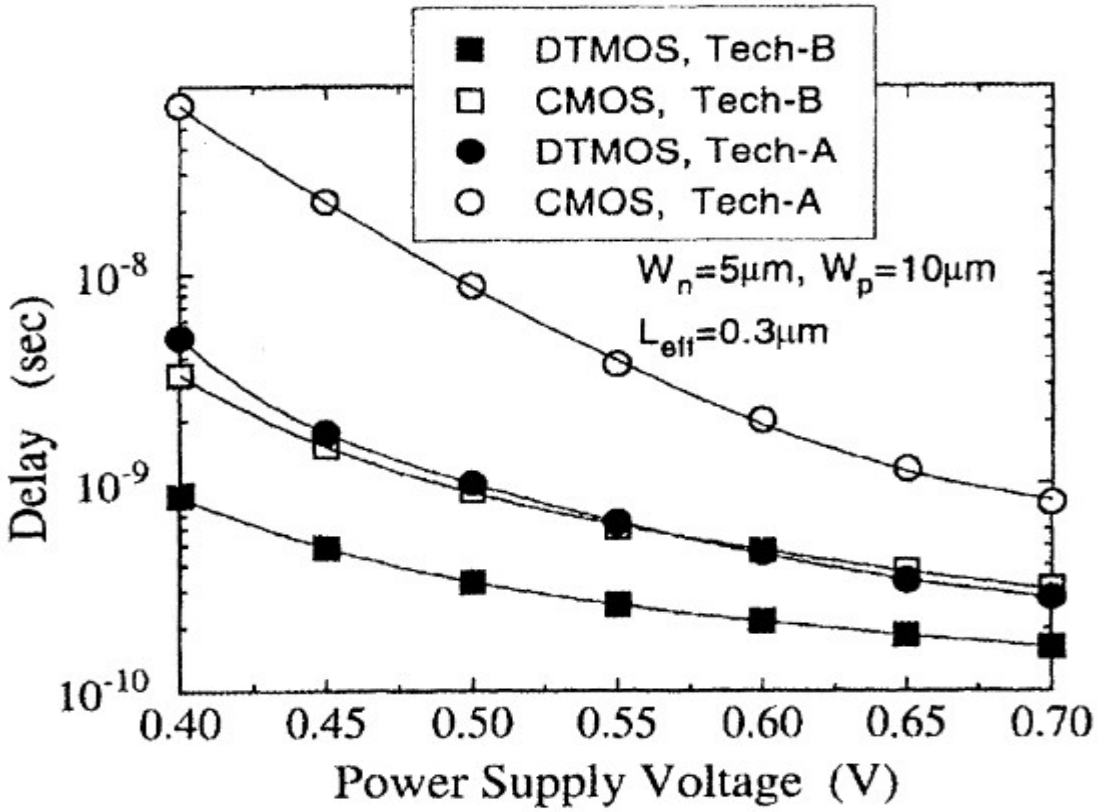


Fig: Delay of CMOS and DTMOS inverter chains.

Equivalent Pin Ordering

Consider a simple two-input CMOS NAND gate shown in Figure. The input pins that are logically equivalent. Logically equivalent pins may not have identical circuit characteristics, which means that the pins have different delay or power consumption. Such property can be exploited for low power design.

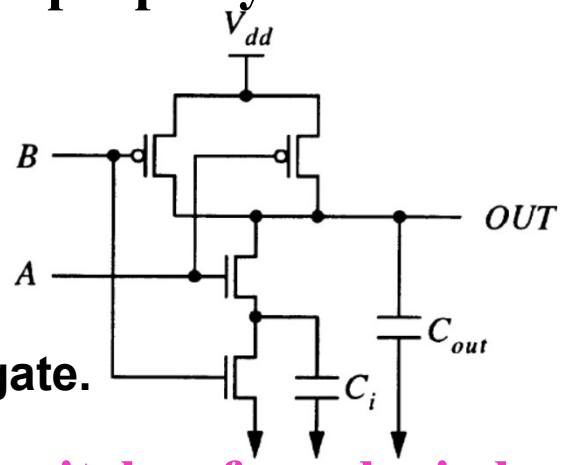


Fig: A two-input CMOS NAND gate.

When the input A is at logic high and the input *B* switches from logic low to high. Initially, capacitance *Cout* and Ci are charged to *Vdd*. After the input *B* switches, *Cout* and Ci are fully discharged to the ground.

The charge stored in *Cout* and Ci has to pass through the bottom N-transistor to discharge.

Consider another symmetrical situation in which the input *B* is at logic high and the input A switches from low to high. Before A switches, C*out* is charged to *Vdd,* but Ci is at the ground level because input *B* is high. Thus, less charge transfer is required to complete the logic switching and the delay and power dissipation are less.

This simple qualitative analysis illustrates the differences in circuit characteristics of logically equivalent pins. This low power technique is known as *pin ordering.*

To apply pin reordering, the designer needs to know the signal statistics at the gate inputs such as the switching frequencies and static probabilities. This can be observed from the logic simulation of a gate-level circuit. When performing pin reordering, the timing constraints of the design should not be violated.

In a circuit, other factors such as parasitic capacitance, resistance, transistor sizes and input signals may contribute to the differences in pin characteristics.

Network Restructuring and Reorganization

To achieve better power efficiency within the allowable timing constraints. Various versions of transistors restructure and reordering techniques will be discussed in this section.

Transistor Network Restructuring

Consider the Boolean function $Y = A\,(B + C)$, *the CMOS* circuit is shown in Figure a, another implementation of the function Y =$A\,(B + C)$ is shown in Figure b

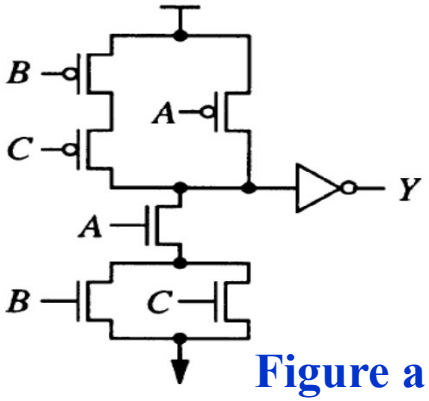


Figure a

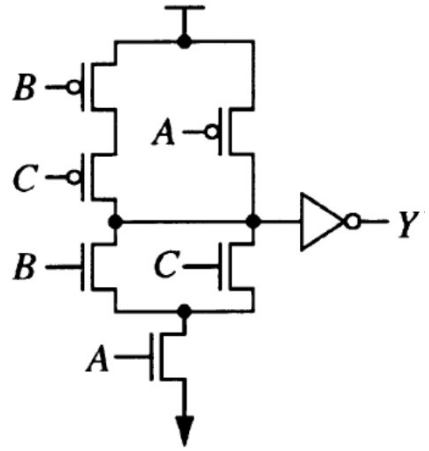


Figure b

The ordering of the two sub-networks is not relevant to the logical behavior of the gate. However, the circuit characteristics depend on the ordering of the serial connection.

The four different circuit implementations that are logically equivalent, as shown in Figure, but different in circuit characteristics such as propagation delay and power dissipation.

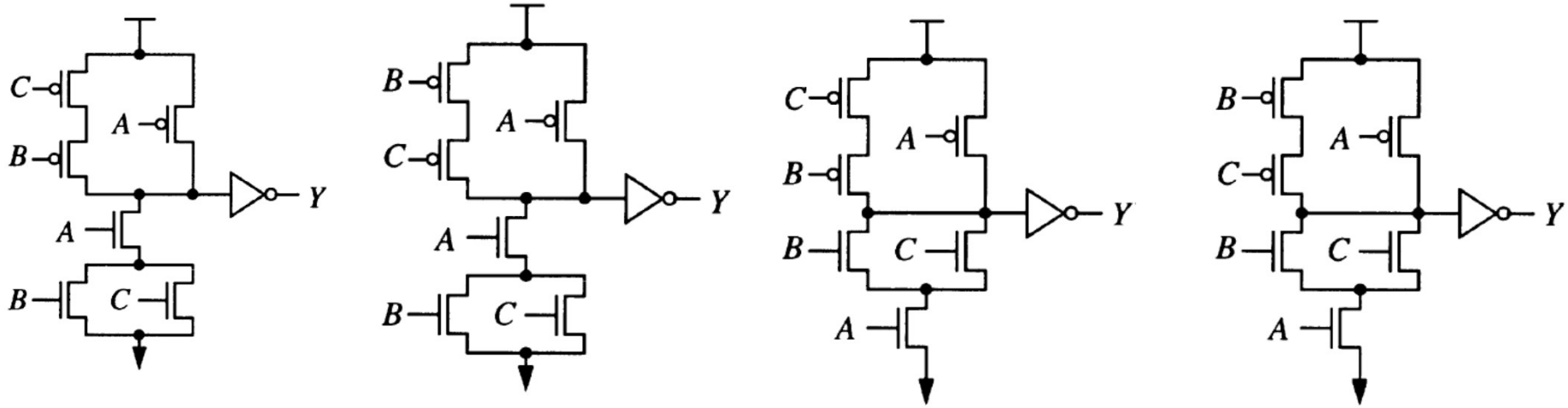


Figure: Four different circuit implementations of $Y = A\,(B + C)$.

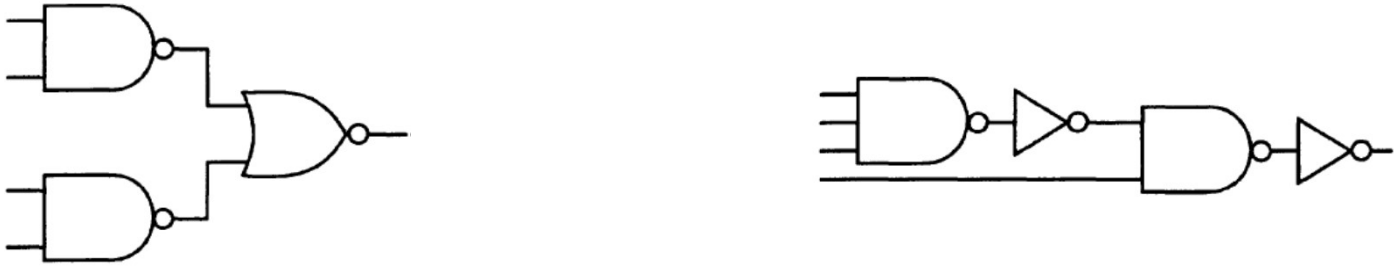
The timing and power characteristics depend on the layout of the cell. As a general rule, transitions involving transistors closer to the output node have less delay and consume less energy. Therefore, the transition probabilities of each input need to be known to evaluate the circuits. Power reduction up to 20% was reported using the transistor network restructuring technique.

Transistor Network Partitioning and Reorganization

Here partitioning and reorganizing the network to explore the different power-area-delay trade-off will be discussed. Network *reorganization* by definition is the task of composing different transistor networks that can implement the same functionality.

When a Boolean function is large, it cannot be implemented in a single complex gate. When transistors are connected in series, the effective resistance of the serial transistor chain increases. To compensate for the increased resistance, the sizes of the transistors have to be increased to maintain an acceptable delay. Also, the *body effect* which increases the effective threshold voltage of the serial transistors, further limiting their conductance. The same problem occurs with too many parallel transistors because each transistor contributes some drain diffusion capacitance to the output node, causing slow transition time. Therefore, a single CMOS gate cannot have too many serial or parallel transistors due to the efficiency concerns. The exact limit of the serial or parallel transistor length is dependent on the process technology, operating voltage, system speed and other factors. As a rule of thumb, the serial chain 4 to 6 and the parallel transistors are restricted from 8 to 10 in today's CMOS digital circuits.

Figure (a) and (b) show two ways to implement a **4-input AND** operation. The partition in (a) requires 12 transistors while the partition in (b) requires 14 transistors. Although both circuit organizations implement an identical Boolean function, their circuit characteristics are different. The circuit in Figure (a) probably has better area and longest pin-to-pin delay but the other circuit may have better shortest delay and power dissipation in some situations.



(a) 12 transistors. (b) 14 transistors.

Figure: Partitioning of a 4-input AND gate

The choice of network structures increases exponentially with the circuit size.

A human designer, guided by intuition, can only handle small circuits with a limited set of constraints.

For large networks, an automated CAD tool has to be used.

Low Power Design at Logic level

Logic analysis is still the basis of VLSI chip design. A strong foundation in logic design is the key to produce high quality chips.

Even with the use of text-based *hardware description language design style, there are* many techniques for the VLSI designer to reduce power at the logic level. The most important one in logic level power optimization techniques is the reduction of switching activities. Switching activities directly contribute to the charging and discharging capacitance, and the short-circuit power. The choice of logic encoding, data representation and Boolean function implementation has significant effects on the power dissipation of a digital circuit.

However suppressing unnecessary activities usually requires additional hardware logic that increases the area and consumes power. The challenge is to justify the low power techniques via intelligent analysis and trade-off while managing the increased design complexity.

Gate reorganization

Gate reorganization is an operation to transform one logic circuit to another that is functionally equivalent.

The reorganized network hopefully has better power efficiency than the original network.

Most gate reorganization tasks today are performed by automated software in the logic synthesis system. Gate-level reorganization is a central operation of logic synthesis. In the past, logic synthesis systems perform optimization and trade-off primarily for area and delay only. Power optimization requirements add another dimension to the complexity of the problem. Commercial and academic logic synthesis tools that automatically perform power-area-delay optimization.

Local Restructuring

WKT Gate reorganization is an operation to transform one logic circuit to another that is functionally equivalent. Most logic restructuring techniques use *local restructuring rules* ***to transform one network to another. Such*** ***local transformations*** **only affect very few of gates and wires in the circuit. Typically these affected gates are directly connected in the logic network. The incremental changes in power, delay and area of such local transformation can be evaluated easily.**

From the hardware description language, a logic synthesis system typically constructs an initial logic network that is believed to be quite good. Then, the gate reorganization procedure applies a series of local transformations to the network. Many networks are generated and evaluated and the best circuit is retained. Some basic transformation operators are:

1. Combine several gates into a single gate.

2. Decompose a single gate into several gates.

3. Duplicate a gate and redistribute its output connections.

4. Delete a wire.

5. Add a wire.

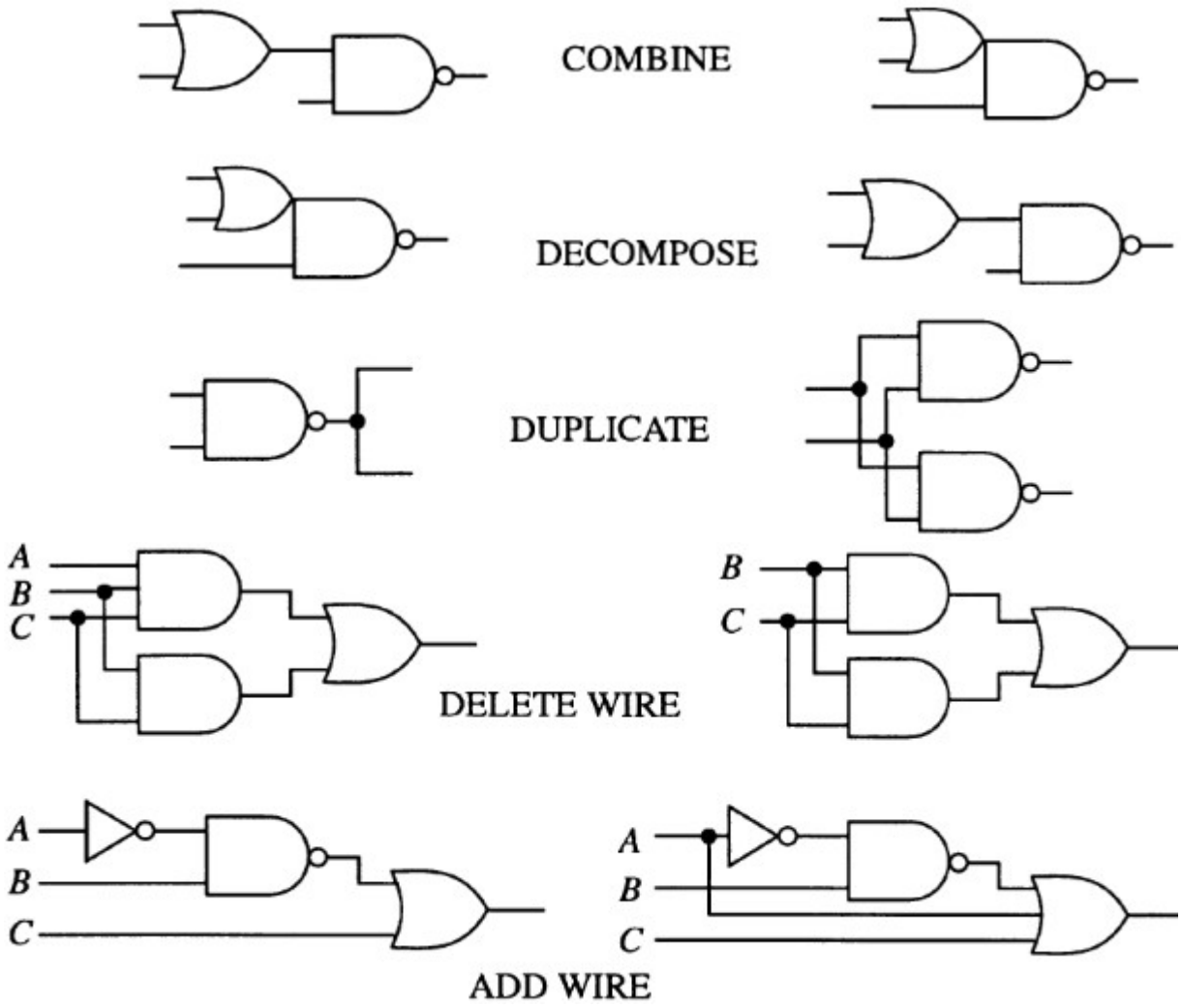


FIGURE: Local transformation operators for gate reorganization.

Figure illustrates some local transformation operators. It can be easily verified that the circuits on the left and those on the right are logically identical. Some of the transformation operators have a direct contribution to power dissipation or propagation delay. The COMBINE operator can be used to "hide" high-frequency nodes inside the cell so that the node capacitance is not being switched.

The DECOMPOSE and DUPLICATE operators help to separate the critical path from the non-critical ones so that the latter can be sized down.

The DELETE WIRE operator reduces the circuit size and the ADD WIRE operator helps to provide an intermediate circuit that may eventually lead to a better one.

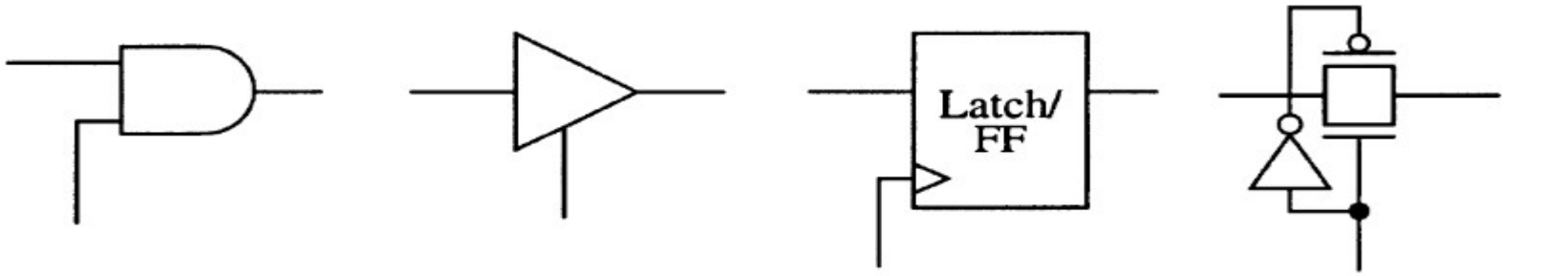
Signal Gating

Signal gating refers to masking unwanted switching activities from propagating forward, causing unnecessary power dissipation.

Most signal gating techniques are applied at the logic level because switching activities of the signals can be easily analyzed.

The simplest method is to put an AND/OR gate at the signal path to stop the propagation of the signal when it needs to be masked. Another method is to use a latch or flip-flop to block the propagation of the signal. Sometimes, a transmission gate or a tristate buffer can be used in place of a latch if charge leakage is not a concern

Figure shows the various logic implementations of signal gating. The signals at the bottom of the circuits are control signals used to suppress the source signal on the left from propagating to the gated signal on the right.



(a) Simple gate. (b) Tristate buffer. (c) Latch/FF. (d) Transmission gate.

FIGURE: Various logic implementations of signal gating.

All signal gating methods require control signals to stop the propagation of switching activities, as indicated by the lower terminals in Figure. The generation of control signals requires additional logic circuitry at the controllers. Thus, the additional power required to generate the control signals needs to be considered to see if the gating leads to overall power saving.

If a group of signals can share a common control, such as on a bus or a clock tree network, the power saving will be remarkable.

The identification of signals to be gated is highly application dependent. Major considerations are the power saving and the complexity of the control signal generation circuit.

Good candidates for signal gating are clock signals, address or data busses and signals with high frequency or glitches.

Logic Encoding

The logic designer of a digital circuit can choose a different encoding scheme as long as the functional specification of the circuit is met. For example, an 8-bit counter can be implemented using the Binary counting sequence or the Gray code sequence. Different encoding implementations often lead to different power, area and delay trade-off. This section discusses some techniques for using different logic encoding to achieve low power consumption.

Binary versus Gray Code Counting

Consider two n-bit counters implemented with Binary and Gray code counting sequences. The counting sequences of the two counters are listed in Table1. In a full counting cycle, the number of transitions of a Binary counter is

$$B_n = 2(2^n - 1)$$

TABLE1 : Binary and Gary code counting sequences.

Binary code		Gray code	
Sequence	No. toggles	Sequence	No. toggles
000	3	*000*	1
001	1	*001*	1
010	2	*011*	1
011	1	*010*	1
100	3	*110*	1
101	1	*111*	1
110	2	*101*	1
111	1	*100*	1

TABLE2: Toggle activities of Binary versus Gray code counter.

No. bits	No. of toggles		B_n / G_n
	Binary $B_n = 2(2^n - 1)$	Gray $G_n = 2^n$	
1	2	2	1
2	6	4	1.5
3	14	8	1.75
4	30	16	1.88
5	62	32	1.94
6	126	64	1.99
∞	.	.	2.00

For example, the 2-bit Binary counting cycle 00, 01, 10, 11, back to 00, has 1 + 2 + 1 + 2 = 6 bit-flips. However, since a Gray code sequence only has one bit flip in each count increment, the number of transitions in a full counting sequence is $G_n = 2^n$

Table2 shows the ratio of *Bn to Gn* for different values of *n. When n is large, the* Binary counter has twice as many transitions as the Gray counter. Since power dissipation is related to toggling activities, a Gray counter is more power efficient than a Binary counter.

Bus Invert Encoding

Bus invert encoding is a low power encoding technique that is suitable for a set of parallel synchronous signals, e.g., off-chip busses. The architecture of bus invert encoding is illustrated in Figure. At each clock cycle, the data sender examines the current and next values of the bus and decides whether sending the *true or the compliment signal leads to fewer toggles.* ***Since the data signals on the bus may be*** complemented, an additional polarity signal is sent to the bus receiver to decode the bus data properly.

For example, if the current value of the bus is 0000 and the next value is 1110, it will send the complement of the next value, 0001, and assert the polarity signal. The assertion of the polarity signal tells the receiver to invert the received bus signals.

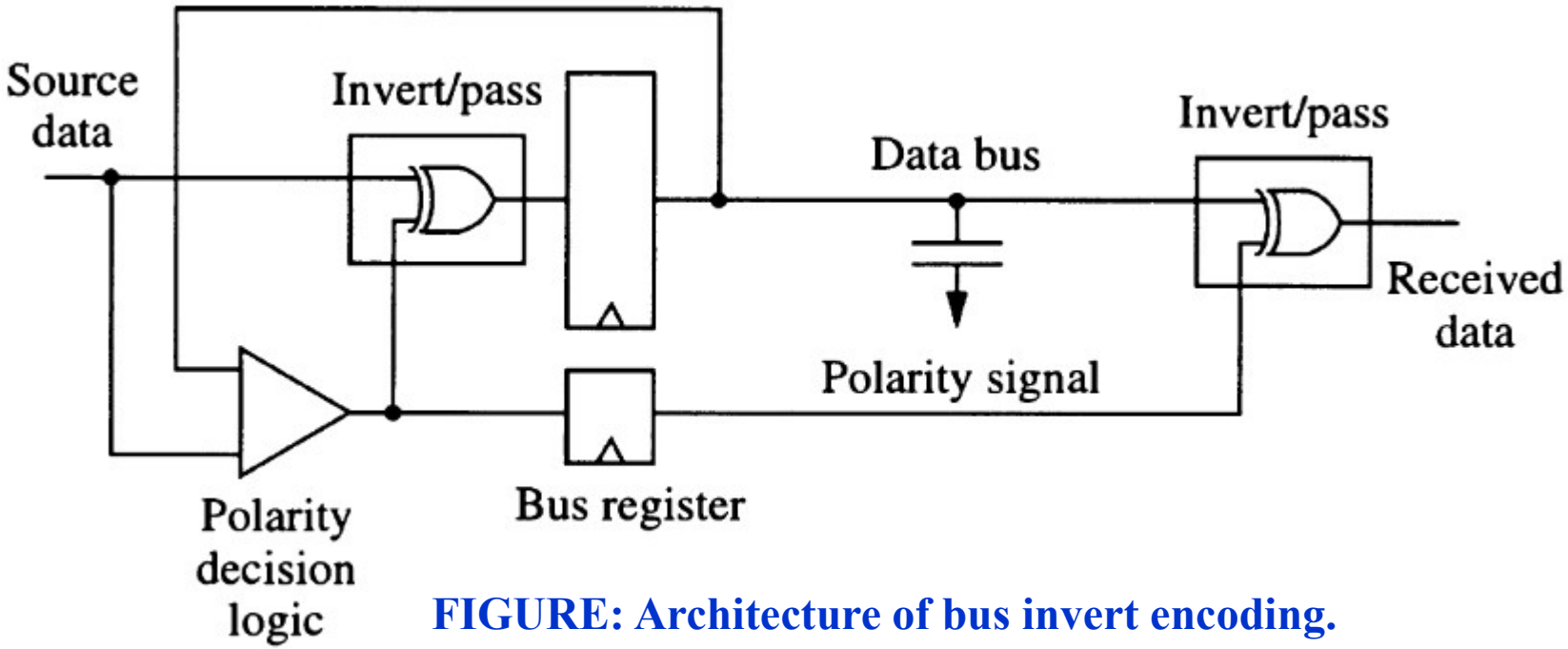


FIGURE: Architecture of bus invert encoding.

The implementation of the bus invert encoding technique is quite simple. We need a register to store the current value of the bus. A polarity decision logic module, which contains combinational logic only, compares the current and next value of the bus and decides whether to invert the bus signals. A bank of XOR gates at the sending and receiving ends inverts the bus signals.

The overhead associated with the bus invert technique is obviously the additional area, power and sometimes propagation delay to implement the polarity decision logic, the invert/pass gates and the polarity signal. If the capacitance of the bus is large enough, the power saving gained from reduced bus signal switching will offset the additional power dissipation required by the polarity and invert logic.

Analysis of Efficiency of the bus invert encoding scheme:

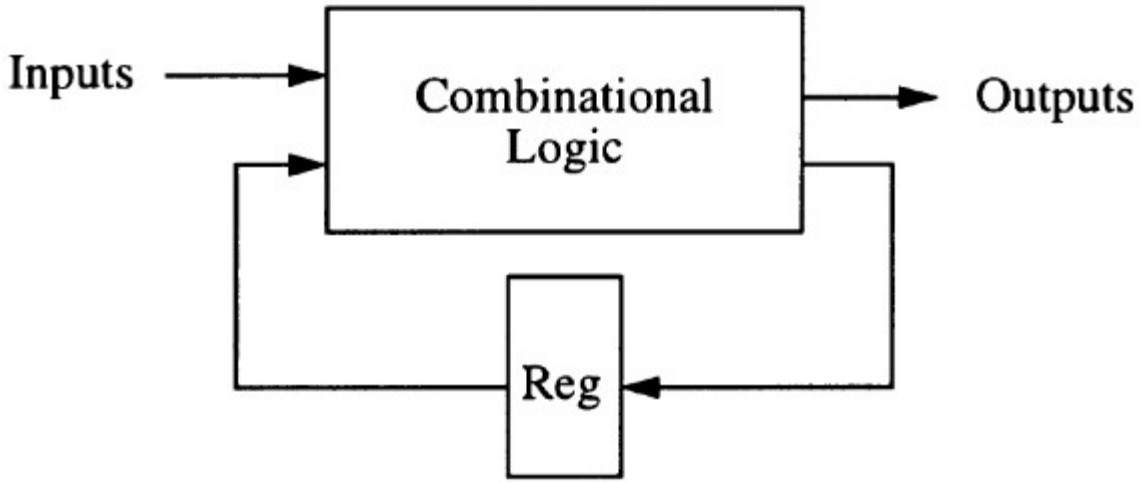
Compared to a regular bus, The maximum number of toggling bits of the inverted bus is reduced from *n to n/2.* *Because* if there were more than *n/2 toggling bits in the inverted bus, we* would have inverted the data signals, thus reducing the toggling bits below *n/2. The* power efficiency of the inverted bus depends on the signal statistics of bus data.

The design decision to apply bus invert technique is dependent on signal statistics and the overhead associated with the polarity decision logic, the polarity signal and the invert/pass gates.

In general, bus invert encoding adds a substantial amount of circuitry to the final implementation. Unless the bus capacitance is large enough, the overhead may not be acceptable. This technique is particularly suited for off-chip busses, because of large loading capacitance and small overhead in additional on-chip logic. Also, since the maximum number of transitions on the bus is reduced from *n to n/2, this technique is suited when the peak current needs to be* reduced.

State Machine Encoding

A state machine can be implemented using Boolean logic and flip-flops, as shown in Figure. The *state transition graph is specified by the designer and the synthesis* system will produce a gate-level circuit based on the machine's specification. The behavior of a state machine can also described in a high-level description language suitable for synthesis, in which case the description is either generated manually or obtained from a state machine compiler.



The state transition graph is a functional description of a machine specifying the inputs and outputs for particular state and its transition to the next state. The very first step of a state machine synthesis process is to allocate the state register and assign binary codes to represent the symbolic states. This process is called the *encoding of a state machine*. *The encoding of a state machine is one of the* most important factors that determine the quality (area, power, speed, etc.) of the gate-level circuit.

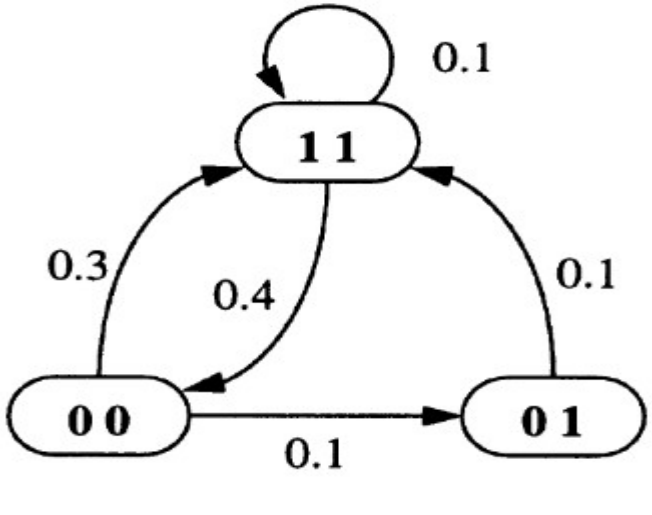
During state machine implementation, the only optimization decision regarding the state register is the number of bits to be assigned. Often, the choice is to use the minimum number of bits that are enough to represent all states.

The real optimization issue that determines the quality of a state machine is the synthesis of the combinational logic network. When a state transition graph is properly encoded, the state machine synthesis problem becomes a combinational logic synthesis problem because the Boolean functions of the combinational network have been completely specified.

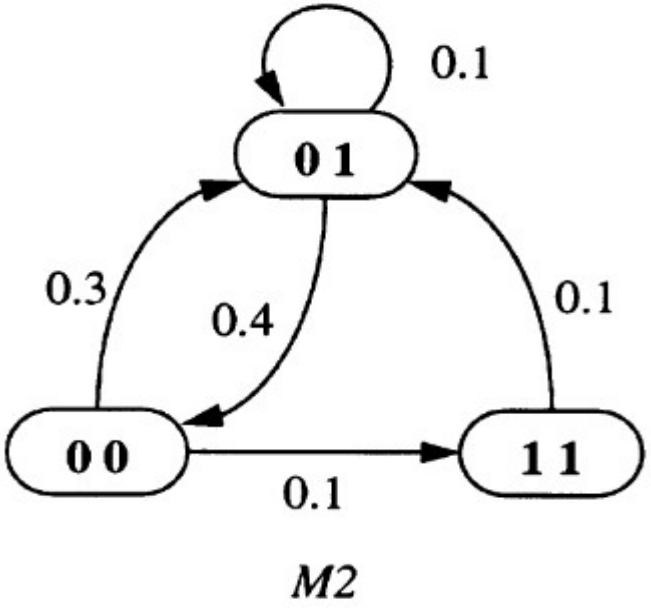
Transition Analysis of State Encoding

Several key parameters have been observed to be very important to the power efficiency of state encoding. One such parameter is the expected number of bit transitions in the state register. Another parameter is the expected number of transitions of output signals.

Consider two functionally identical state machines Ml and M2 with different encoding shown in Figure. The binary codes in the state bubbles indicate the state encoding. The labels at the state transition edges represent the probabilities that transitions will occur at any given clock cycle. The sum of all edge probabilities equals to unity. The expected number of state-bit transitions *E [M] is given by the sum of prod*ucts of edge probabilities and their associated number of bit-flips as dictated by the encoding.



M1



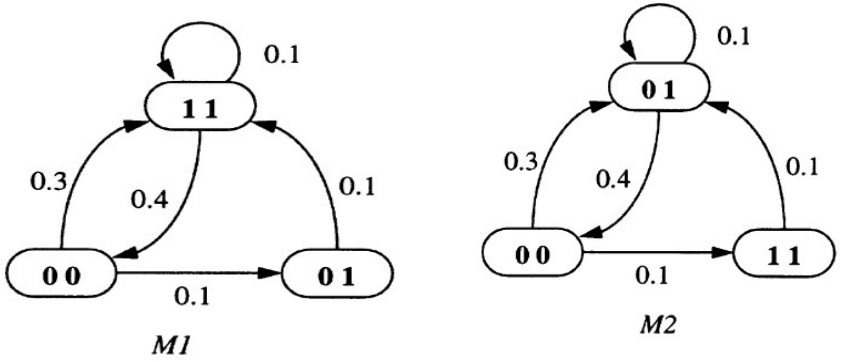
M2

Fig: Functionally identical state machines with different encoding

For example, the machine Ml has two edges with two bit transitions (00 and 11) at probabilities 0.3 and 0.4. The contribution of the edges to the expected transition is 2 (0.3 + 0.4). The expected transitions per clock cycle for the two machines are given in the figure.

E[M1]=(2x.3+2x.4)+1x.1+1x.1=1.6

E[M2]=(1x.3+1x.4)+1x.1+2x.1=1.0



In general. machines with lower *E[M]* *are more power efficient* *because:*

1. Fewer transitions of the state register lead to lower power dissipation.

2. Fewer transitions are propagated into the combinational logic of the machine.

However, a state encoding with the lowest *E [M] may not be the one that results in* the lowest overall power dissipation. The reason is that the particular encoding may require more gates in the combinational logic. resulting in more signal transitions and power.

In some state machine specifications, an output signal is often undefined or *don't-care* at some states and thus can have a logic 0 or 1. A proper assignment of the don't-care signal can reduce the expected transitions of the output signal and saves power.

Design Trade-offs in State Machine Encoding

The area-power trade-off of state machine encoding is not well understood and is heavily influenced by the logic synthesis system. State encoding affects the power Dissipation as well as the area of the machine.

Most commercial logic synthesis system can perform automatic state encoding for the Goal of area minimization and it is not be desirable for power dissipation because the expected transition is high.

If we encode the states to minimize the expected transition, the area is often not good because the logic synthesis system has lost the freedom of assigning state codes that favor area optimization.

One practical solution is to balance the area-power trade-off by encoding only a subset of states that spans the high probability edges. For example, if several edges constitute more than 90% of all state transitions, we can encode the states incident to the high probability edges to reduce *E [M]*. *The remaining state codes can be left to the* logic synthesis system so that it has the freedom of choosing the proper encoding for better area efficiency.

Precomputation Logic

The principle of precomputation logic is to identify logical conditions at some inputs to a combination logic that is invariant (remains unchanged) **to the output. Since those input values do not affect the output, the input transitions can be disabled to reduce switching activities.**

Basics of Precomputation Logic

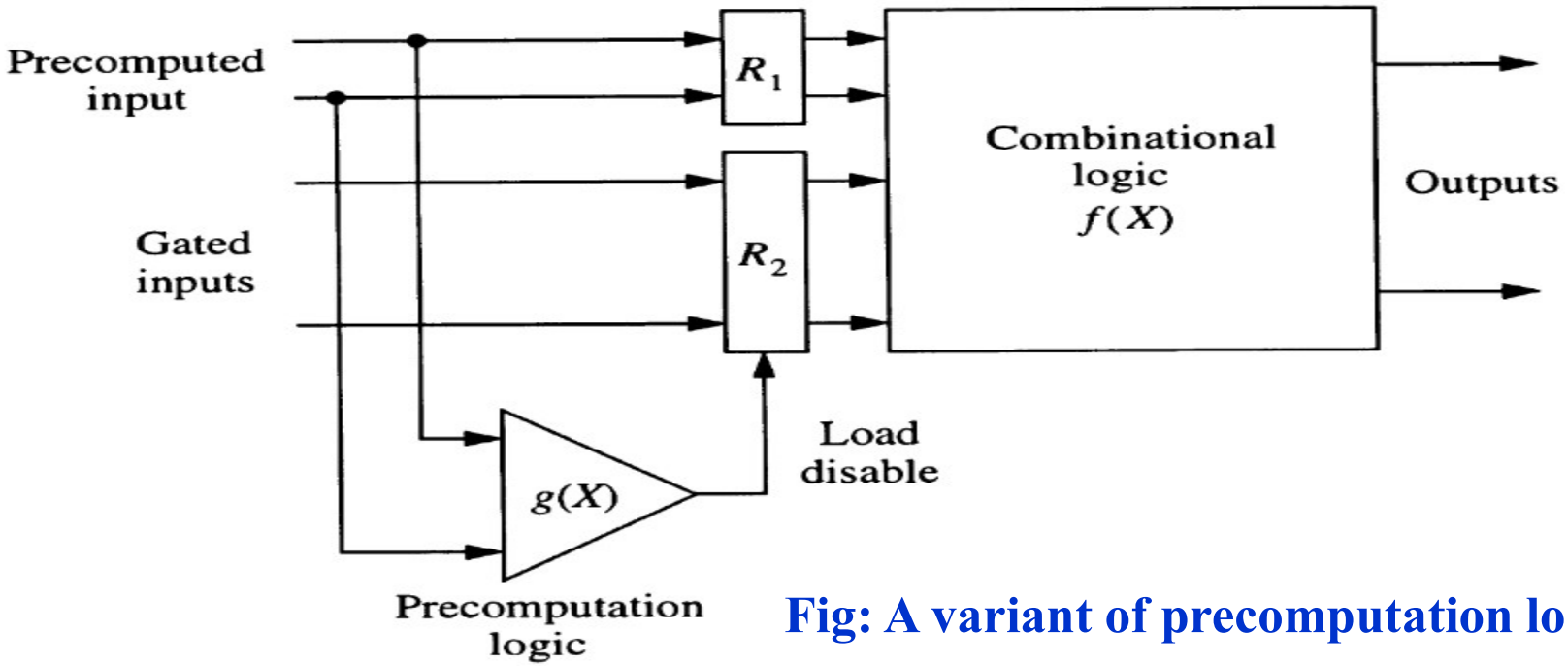


Fig: A variant of precomputation logic

One variant of the precomputation architecture is shown in Figure. *R1 and R2 are* registers with a common clock feeding a combinational logic circuit with a known Boolean function *f(X). Due to the nature of the function f(X), there may be some* conditions under which the output of *f(X) is independent of the logic value of R2.*

Under such condition, we can disable the register loading of *R2 to avoid causing* unnecessary switching activities, thus reducing power. The Boolean function *f(X)* is correctly computed because it receives all required value from *R1. To generate the* load-disable signal to *R2,* a precomputation Boolean function *g(X) is required to* detect the condition at which *f(X) is independent of R2.*

The g(X) depends on the input signals of R1 only because the load disable condition is independent of *R2.* Otherwise, *f(X) will depend on the inputs of R2 when the load*disable signal is active.

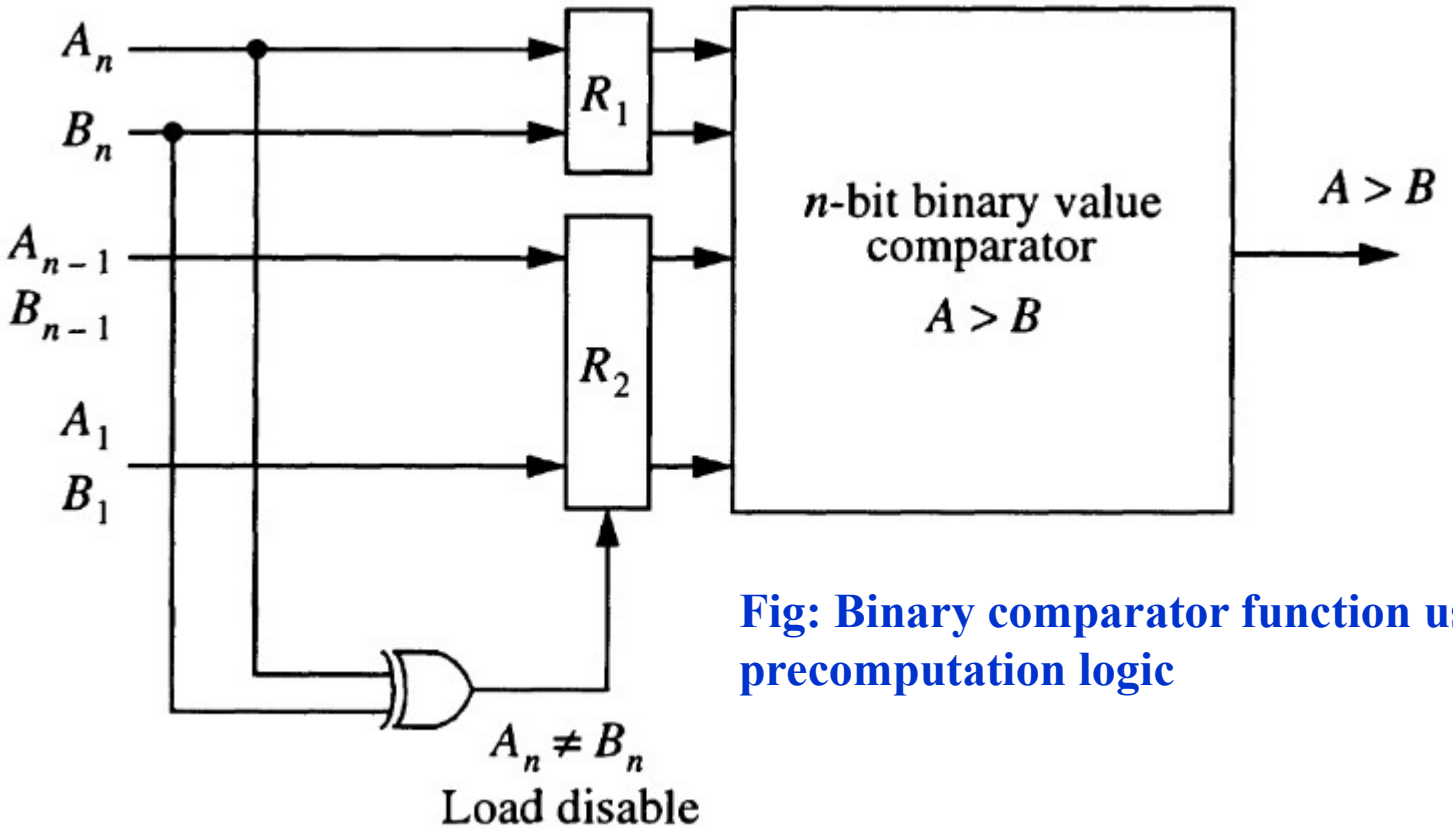


Fig: Binary comparator function using precomputation logic

Consider an example of precomputation logic is the Binary Comparator function f(*A, B) that computes A > B, as shown in Figure . Let inputs A1 ... , An and* B1 *... , Bn be n-bit signals that represent binary values. The output f(A, B) is logic 1* if the binary value *A is greater than B. We choose R1* to be *(An, Bn) and R2 to be the* other signals. The signals, *An and Bn are the most significant bits of the input values.*

An obvious precomputation function is *g(X) = An XOR Bn. When An XOR Bn = 1, mean*ing the two bits differ, the output of *f(A, B) can be determined without the input bits* of *R2.*

If *An* = ***1, Bn = 0, the output is 1 independent of the least significant bits of A or B. Again, if An = 0, Bn = 1, the output is 0 regardless of the inputs of R2.*** **When *An XOR Bn = 0, R2 cannot be disabled because its signals are required to com*pute the output** *f(A, B).*

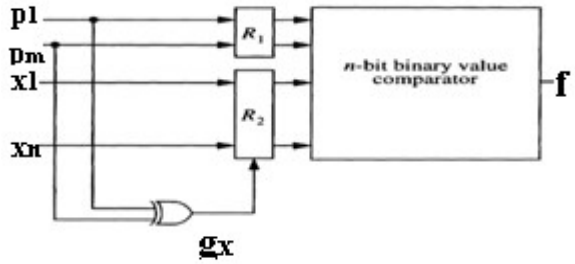
A simple statistical analysis shows that the precomputation logic implementation for uncorrelated input bits with every bit has an equal probability of zero or one, There is a 50% probability that *An XOR Bn = 1 and the register R2 is disabled in 50% of the clock cycles.*

Also, when the load-disable signal is asserted, the combinational logic of the comparator has fewer switching activities because the out puts of *R2 are not switched. The extra power required to compute An XOR Bn is negli*gible compared to the power saving even for moderate size of *n.*

From the above discussion, it is obvious that a designer needs to have some knowledge of the input signal statistics to apply the precomputation logic technique. In Figure, if the probability of *An XOR Bn is zero, the precomputation logic* circuit may be inferior, in power and area, compared to direct implementation. Experimental results shows up to 75% power reduction with an average of 3% area overhead and 1 to 5 additional gate-delay in the worst-case delay path.

Precomputation Condition

Given *f(X), R1 and R2,* there is a systematic method to derive a precomputation function *g(X).*



Let f(p1, ... , pm, x1,.... , xn) be the Boolean function where P1 ... , Pm are the precomputed inputs corresponding to R1 and *x1 … , xn are the gated inputs corre*sponding to *R2.* Let *fxi (fxi') be the Boolean function obtained by substituting xi = 1 (xi = 0) in the function f, then*

$$U_{x_i} f = f_{x_i} \cdot f_{\overline{x}_i}$$

Note that the functions *fxi and fx' are new Boolean functions constructed from f and* do not contain the variable *xi, The function Uxif is the Boolean AND operation of* the two functions *fxi and fx',* ***and by definition also does not contain the variable xi.***

The Boolean function *Uxif* is called the *universal quantification of f with respect to* the input variable *xi, An important observation with the universal quantification func*tion is that Uxif=1 *implies f = 1 regardless of the value of xi. This can be* observed with the help of the Shannon's decomposition of *f with respect to xi*

$$f = x_i f_{x_i} + \overline{x_i} f_{\overline{x_i}}$$

The condition *Uxif =1 means that fxi = fxi' = 1 and therefore f = 1 regardless of* the value of *xi.*

Let

$$g_1 = U_{x_1} U_{x_2} \ldots U_{x_n} f$$

The Boolean function g1 is constructed by first taking the universal quantification of *f with xn to obtain Uxnf, then applying the universal quantification of Uxnf, with xn-1, to obtain the second function, etc. The procedure is applied iteratively until all* variables have been exhausted and we obtain the function *g1*. *We also note that the* function gl does not contain variables x1, ... , *xn*.

Similarly we can define

$$g_0 = U_{x_1} U_{x_2} \ldots U_{x_n} \bar{f}$$

Here, *go is obtained by applying universal quantification to the compliment function* of *f. The condition go = 1 implies f = 0 regardless of the values of x1, ... , xn.*

Thus, the function *g = g1 + go is a precomputation function in which g = 1 means that* we can disable the loading of gated inputs x1, ... , *xn into R2 because the output f is* independent of the gated inputs.

The precomputation function *g only contains the precomputed variables p1, ... , Pm.* In general, the function *g may not exist because the universal quantification of f may* evaluate to a logic 0. For example, the universal quantification of any variable of *f = x1 • x2* is always 0 and no precomputation function exists.

The function *g contains the most number of* one's in its truth table among all precomputation functions qualified. Maximizing the load-disable probability is desirable because it leads to fewer activities in the combinational logic of *f and the gated register R2.* However, it is difficult to predict if a function that maximizes the load-disable probability is also one that minimizes the total power consumption. This is because the power dissipation of the precomputation logic *g itself cannot be ignored. An unoptimized precomputation function g' may* require fewer logic gates so that it could lead to better total power consumption. In practice, it makes sense to explore other precomputation functions for better total power dissipation.

Alternate Precomputation Architectures

Many precomputation schemes are used. One scheme based on the Shannon's decomposition, is shown in Figure. The Shannon's decomposition states that a Boolean function f*(x1, ... ,xn)* can be decomposed with respect to the variable *xi as follows*

$$f(x_1, \ldots, x_n) = x_i f_{x_i} + \overline{x_i} f_{\overline{x_i}}$$

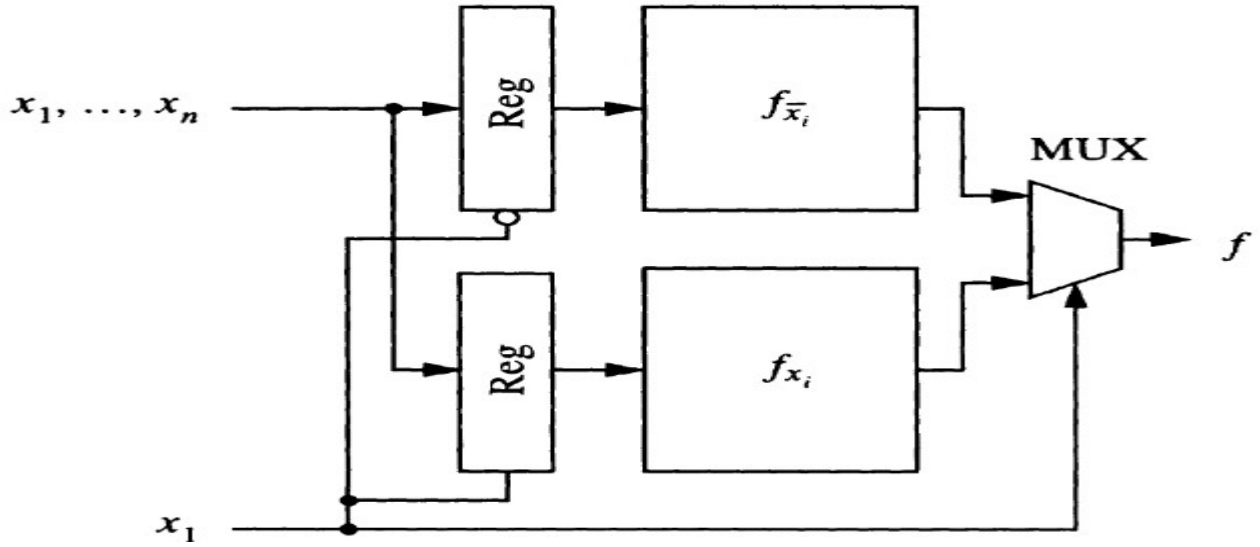


Fig: A precomputation architecture based on Shannon's decomposition

The equation allows us to use x*i* *as the load-disable signal as shown in Figure.*

When *xi = 1 (xi = 0), the inputs to the logic block fxi' (fxi) can be disabled. The* **multiplexor selects the output of the combinational logic block that is active. This means that only one combinational logic block is activated at any clock cycle. Power saving is achieved if each of the two decomposed logic blocks consumes less power than a direct implementation of** f*(x1, … ,xn).* ***However,*** ***the precomputation architecture consumes more area and delay in general.***

Another precomputation architecture for combinational logic is shown in Figure.

The original logic implementation is shown in Figure (a) and the low power implementation is shown in (b). This architecture is also called *guarded evaluation* because some inputs to the logic block C are isolated when the signals are not required, to avoid unnecessary transition. Transmission gates may be used in place of the latches if the charge storage and noise immunity conditions permit.

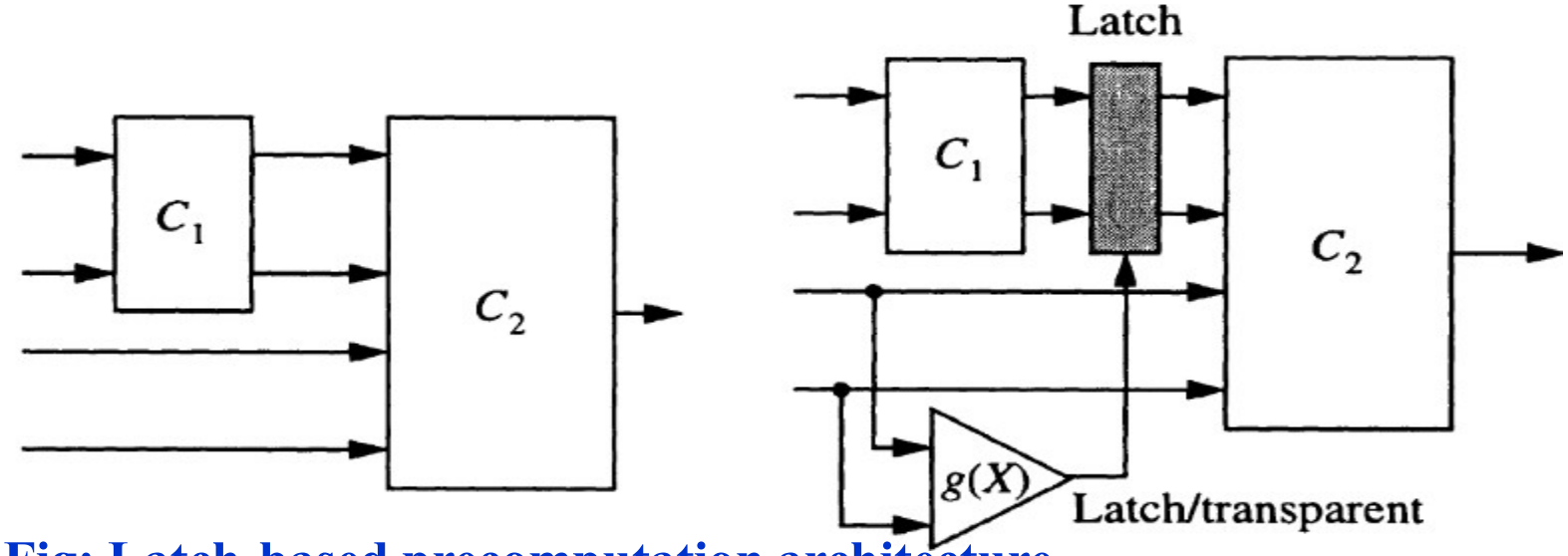


Fig: Latch-based precomputation architecture

Design Steps and Design Issues in Precomputation Logic Technique

The basic design steps with precomputation logic are as follows:

1. Select precomputation architecture.

2. Determine the precomputed inputs *R1 and gated inputs R2 given the function* $f(x)$.

3.With *R1 and R2 selected, find a precomputation logic function g(X). Note that g(X) is not unique and the choice greatly affects the power efficiency. The function g(X) may also fail to exist for poor choices of R1 and R2.*

4. Evaluate the probability of precomputation condition for the potential power savings. Make sure that the final circuit is not overwhelmed by the additional logic circuitry and power consumption required to compute *g(X).*

Issues in Precomputation Logic

i) In practice, the selection of *R1, R2 and precomputation logic g(X) depends heavily on* the designer's knowledge of the input signal statistics. If the input signal statistics is unknown, the precomputation method cannot be applied because it is impossible to evaluate the power savings resulted from gating some inputs.

ii) Another prerequisite in using precomputation logic is the availability of CAD tools for area and power estimation. The power saved from input gating and the additional power dissipated in precomputation logic have to be assessed carefully so that the area overhead can be justified.

iii) Precomputation is by definition creating redundant logic. The logic circuits that perform precomputation generally pose difficulties in testing.