Implementation Issues

This chapter presents the physical implementation (gate/transistor level) and architectural organization of modern-day caches, including a discussion of how this fits in a typical pipeline. One of the most important implementation issues today is that of dealing with the increasingly significant effects of technology scaling. In modern cache implementation, technology scaling exposes and/or exacerbates the following problems:

- The subthreshold leakage current caused by decreasing transistor threshold voltages becomes larger, increasing static power dissipation and requiring both evolutionary and revolutionary solutions to offset its effects.
- The decreasing amount of charge stored per bit makes it easier to corrupt a bit either by an external source (e.g., alpha emission, EMI) or by an internal source (e.g., crosstalk, ground bounce).
- Process variations increasingly cause mismatches of transistor characteristics in the implementation.
- The increasing contribution of wire propagation delay to the total delay must be taken into account in the pipeline's timing budget.

Some contemporary solutions are presented to counter some of the technology scaling problems present in deep and very deep submicron design.

5.1 Overview

To start things off, we discuss a sample cache operation, a cache read hit in moderate detail, exposing some of the implementation issues involved in its design.

Figure 5.1 shows an example cache organization: a two-way, set-associative cache with virtual addressing, along with a timing diagram showing the various events happening in the cache (to be discussed in much more detail in later sections).

Basically, the following steps involve:

1. Providing an address to the cache, along with an address strobe signal (ADS) confirming the validity of the address. A read/write signal (R/W#) is also sent to specify the operation.
2. The index part of the address chooses a word within the tag and data arrays of the two-way, set-associative cache (signified by the wordline signal, WL). This, in turn, causes the internal bitlines to develop a differential, which is amplified by sense amplifiers to produce a full-swing differential output voltage.
3. The translated address from the TLB is compared with the output of the tag array to decide if the cache access hit or miss. In case of a hit, the proper data is chosen among the two ways by controlling the output multiplexer and is forwarded. A cache miss requires the cache controller to perform

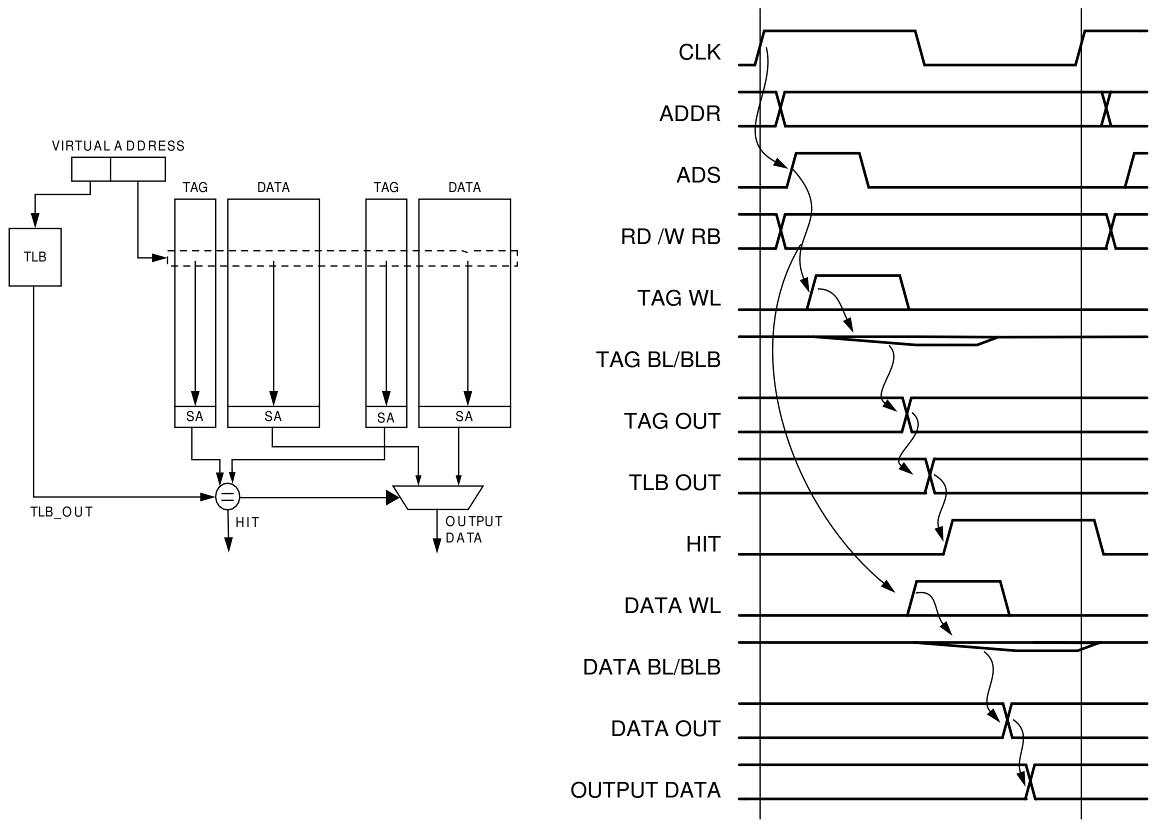


FIGURE 5.1: Block diagram of a two-way, set-associative cache organization, along with the timing diagram for a read operation.

a separate operation to retrieve data from external memory (or another level of cache) and to perform a write access to the cache.

We have now demonstrated the basic cache read operation and shown some of the blocks used in implementing a cache. We will proceed to more in-depth details, starting with the implementation of the basic storage structures comprising the tag and data array and moving on to how a cache is implemented and how these data arrays are used. Along the way, we also discuss advanced topics related to contemporary cache issues such as low-leakage operation.

5.2 SRAM Implementation

In this section, we discuss the implementation of a static random-access memory (SRAM). This is the type of memory used as the building block of most caches because of its superior performance over other memory structures, specifically DRAM. SRAM uses the same fabrication process as the microprocessor core, simplifying the integration of cache onto the processor die. By contrast, DRAM typically uses a different process that is sub-optimal for logic circuits, making the integration of logic and DRAM less attractive than the integration of logic and SRAM.

The discussion included here is complete enough to use as a basis for designing an SRAM from scratch.

5.2.1 Basic 1-Bit Memory Cell

This subsection describes how the most basic unit of an SRAM—a single data bit—is implemented. For SRAMs, the memory cell (MC) is implemented as two cross-coupled inverters accessed using two pass transistors, as shown in Figure 5.2.

This cross-coupled connection creates regenerative feedback that allows it to indefinitely store a single bit of data. This configuration shows a 1-bit memory cell with one R/W port that can be used for either a read or a write, but not both simultaneously. As shown in Figure 5.2, the bit is read by asserting a WL and detecting the voltage differential between the bitline pair, which is initially precharged to a logic high voltage.

The second phase, shown in Figure 5.1, is a write operation. The bitline is driven with a differential voltage from an external source to force the data onto the memory cell.

Physical Implementation of the 1-Bit Memory Cell

One of the assumptions of caches and memory, in general, is the ability to store digital data, and we have shown how a single bit can be stored by an SRAM using cross-coupled inverters.

This cross-coupled inverter has been implemented in different ways during its evolution, where the main difference has been how the inverter's pullup network is implemented. Figure 5.3 shows different circuit configurations that have been used to implement the memory cell.

Early SRAMs typically used either the full-CMOS or the polysilicon load memory cell configuration. The main advantage then of the poly-load configuration was its smaller area since only four transistors were required for one bit; the pullup was implemented using an additional highly resistive polysilicon layer such that the load was fabricated on top of the existing transistors, requiring less area.

With decrease in feature sizes, the amount of area occupied by the poly-load needed to produce a large enough resistance to overcome leakage currents that became too big. Along with its low soft error rate (SER) [List 1986], the poly-load implementation has now become impractical.

With the increase in space occupied by the poly-load, it started to be replaced by the poly-PMOS or thin-film-transistor (TFT) PMOS [Minato et al. 1987] load configurations, which could still be fabricated on top of the active NMOS, but with characteristics superior to the poly-load.

With the increasing on-chip integration of caches with microprocessor logic circuits, the additional process complexity required to implement the poly-PMOS/TFT-PMOS circuits on the same die as the digital circuits became too cumbersome.

An alternative to these circuits, the loadless four-transistor (LL4T) cell completely removes the pullup

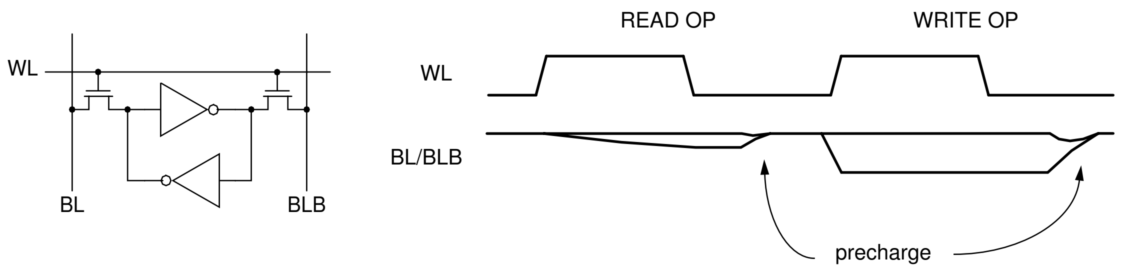


FIGURE 5.2: The basic SRAM cell showing cross-coupled inverters accessed through pass transistors. The state of important signals is also shown during a read and write access.

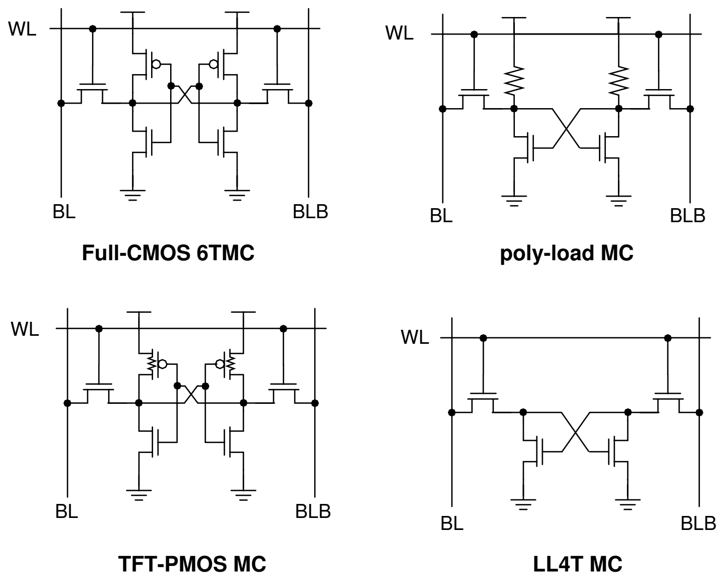


FIGURE 5.3: Different implementations of the SRAM cell.

load [Noda et al. 1998]. Although this scheme results in a smaller area compared to the full-CMOS implementation, because only four NMOS transistors are involved, the cell has to be very carefully designed to make sure that leakage currents do not interfere with the latching operation of the cell and its ability to retain data.

With continuous technology scaling, the charge stored within the cell decreases, while leakage current increases, making it more difficult to design reliable LL4T cells, especially if external radiation from alpha particles is taken into account.

Currently, most conventional designs use the full-CMOS six-transistor memory cell (6T MC), with different variations existing based on issues of sizing, physical layout, and transistor threshold voltages for low power (to be discussed later).

The rest of the discussion will be limited only to the 6T MC variant, and an example layout using MOSIS SCMOS rules for a 0.25-μm process is shown in Figure 5.4, along with its transistor-level circuit defining the access, driver, and pullup transistors.

Transistor Sizing

The main initial considerations when sizing the transistors of the memory cell are the cell's area and its stability as measured by its static-noise margin (SNM). The SNM is defined as the amount of DC noise necessary to disturb the internal storage node of the cell and flip its stored data [Lohstroh et al. 1983]. The cell's stability affects the cell's SER and its sensitivity to PVT variations.

The cell's SNM can be calculated analytically by solving the cross-coupled inverter voltage transfer equations. Alternatively, it can be estimated using the graphical "maximum squares" method using the inverter's I/O voltage transfer function [Seevinck 1987], as shown in Figure 5.5.

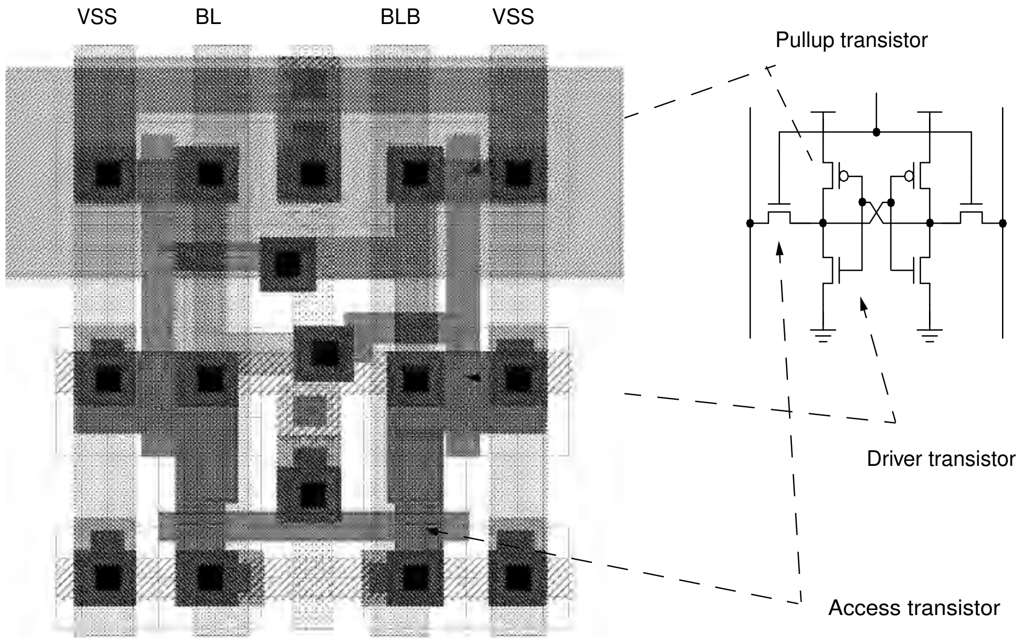


FIGURE 5.4: Layout of a six-transistor memory cell (6T MC) using MOSIS SCMOS rules (SCMOS_SUBM using TSMC 0.25 µm). Also shown is the definition of the access, driver, and pullup transistors.

When sizing, the main variables that can be varied are the widths and lengths of the driver, pullup, and access transistors.

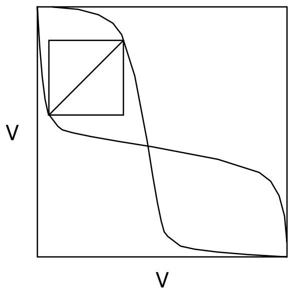


FIGURE 5.5: The inverter's voltage transfer function.

Using the terminology from Seevinck, sizing involves the parameters (r, β, q), where the beta figure β is the ratio of the widths and lengths of the individual FETs, r is the ratio between the beta figures of the driver and the access transistor, and q is the ratio of the beta figures of the pullup and the access transistors.

The main conclusions from Seevinck's analysis is as follows:

1. The SNM depends only on Vdd, transistor voltages, and beta ratios, not on the absolute values of the transistor betas.
2. Designing the cells for maximum SNM requires maximizing r and q/r by the appropriate choice of W/L ratios, constrained by area limitations and a proper cell-write operation (i.e., some choices of W/L ratios will be difficult to write to).
3. For fixed r and q, the SNM of 6T MC will be independent of Vdd variations.
4. Finally, the SNM increases with increasing threshold voltages.

All of these factors must be considered and balanced when designing the memory cell to achieve the desired cell area, stability, and performance.

Soft Error Rate (SER)

The SER is a measure of the rate at which the bits stored inside a memory's cells are corrupted. It is often discussed in the context of effects such as alpha particle radiation, where charged particles may induce enough voltage in a cell storage node that exceeds the cell's SNM, resulting in corruption of data.

The value Q_{crit} of a cell refers to the critical amount of charge that will result in exceeding the cell's SNM. It is directly related to the amount of charge inside the cell's storage nodes (the outputs of the cross-coupled inverters).

Various ways exist to increase Q_{crit} (hence improving the SER), among them increasing the diffusion area of the transistors, resulting in increased capacitance and, accordingly, the amount of stored charge within the storage nodes. Other methods use more complex processing techniques to increase the node capacitance [Ishibashi 1990, Sato et al. 1999].

The SER and some accepted solutions will be discussed in more detail in the advanced sections.

5.2.2 Address Decoding

Address decoding is a conceptually simple process of receiving address information and providing signals to initiate and perform the desired operation on the addressed location.

At its simplest, it involves feeding an address value into a binary decoder (n to 2^n) and using the asserted output to activate the wordline of a subset of memory cells associated with this address. This involves a single AND operation on the input address, with the output connected to memory cell wordlines, as shown in Figure 5.6.

The main concern in address decoding is the large fan-in and fan-out requirements of typical memories

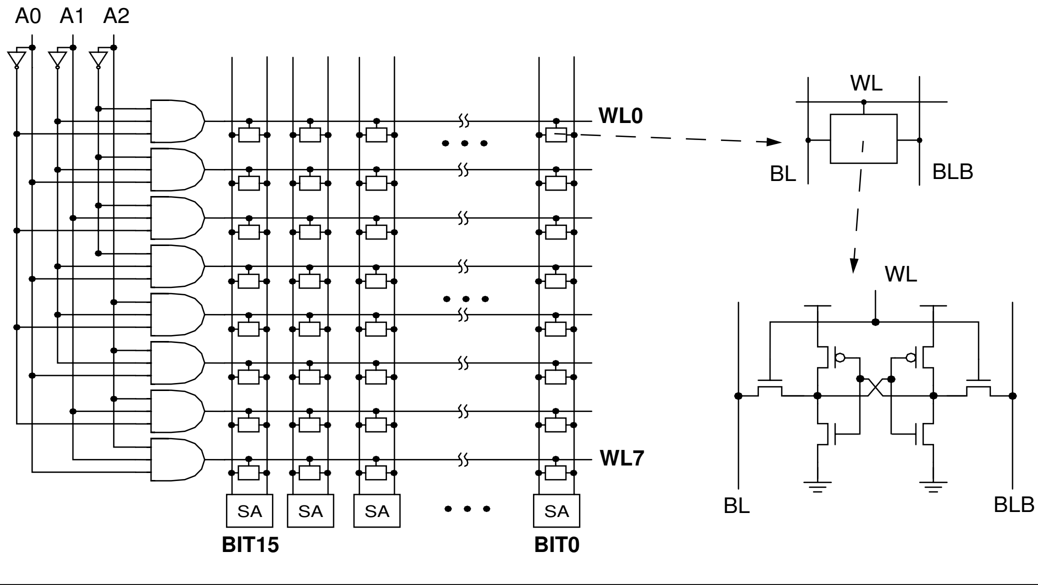


FIGURE 5.6: Address decoding shown for a simplistic 8 x 16 bit memory showing 8 3-input AND gates that enable the wordlines of a specific number of cells (16 for one wordline). Also shown are the sense amplifiers and the internals of a single cell.

because of the number of address bits being decoded and the large amount of cells that have to be driven. This makes the simple one-level AND structure inefficient, and virtually all SRAM designs implement a multi-level decode hierarchy to implement the logic AND function.

In typical designs, the address decoder contributes significantly to the critical path delay and total power dissipation, emphasizing the need to optimize the memory array's decode hierarchy implementation.

Predecoding

One of the main functions of a decode hierarchy is to minimize the fan-in of parts of the decode circuit, because higher fan-in gates have a large logical effort [Sutherland and Sproull 1991, Sutherland et al. 1999] making them less efficient. Simply put, logical effort expresses how hard it is to drive an arbitrary gate compared to an inverter of the same drive strength. High fan-in static gates typically have high logical efforts and are less efficient to use.

One of the techniques used in minimizing fan-in is a method called predecoding. Predecoding involves using one level of logic to AND subsets of the address. The outputs of these predecoders are then combined by low fan-in gates to produce the final decoder outputs, which are the wordline signals connected to the memory cells. In this way, predecoding simply involves performing the AND operation using multiple levels of logic.

Consider an example where an 8-bit address is to be decoded. A simplistic approach using 256 (2^8) 8-input AND gates is used (see Figure 5.7(a)). Although logically correct, these ANDs will have significantly higher gate capacitances compared to 2-input ANDs, resulting in larger delays, higher power dissipation, and larger area.

An alternative decoder implementation using predecoding is shown in Figure 5.7(b). The 8-bit address is divided into two subsets, each with 4 bits. For both subsets, 16 4-input ANDs are used to generate all possible combinations of the 4-bit address. The final wordlines are generated using 2-input AND gates to

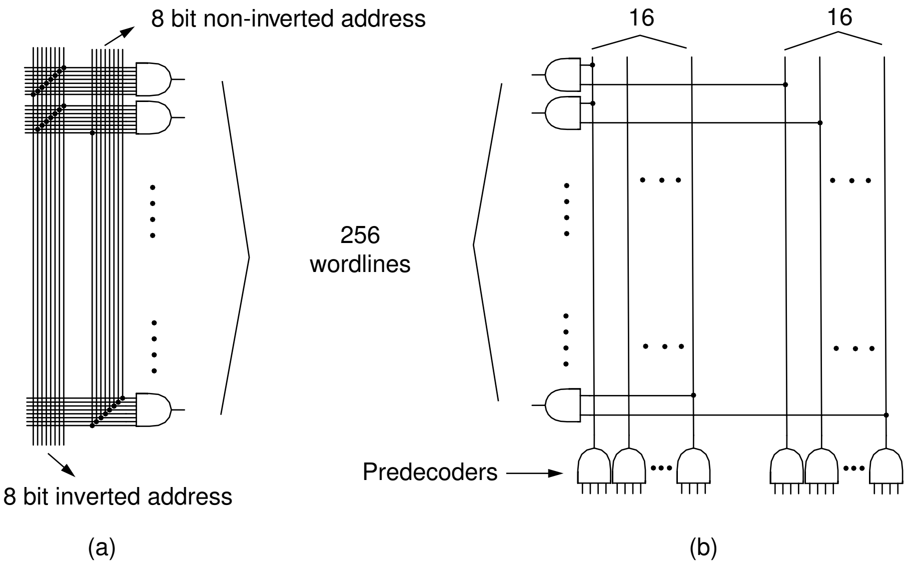


FIGURE 5.7: Sample wordline decoders for an 8-bit address: (a) a simplistic high fan-in AND approach and (b) a decoder with predecoding.

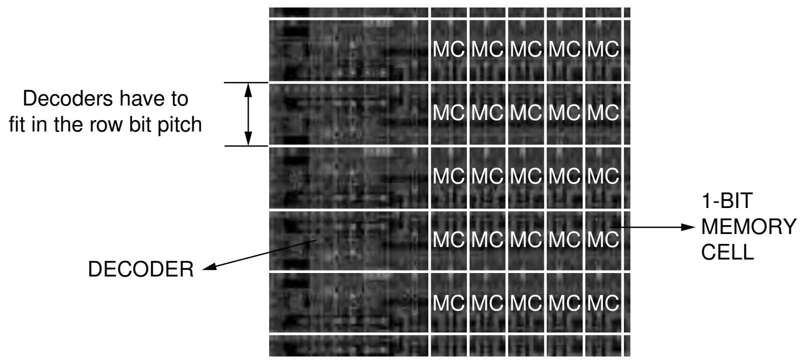


FIGURE 5.8: Sample floor plan of the decoders beside the memory array showing the bit pitch limitation.

combine one output each from the two subsets. In this case, 256 2-input AND gates are used to generate all the 256 wordlines.

Although conceptually simple, predecoding has numerous advantages. Since the final AND gates only have two inputs, they will have significantly less gate capacitance compared to the first implementation. This results in a smaller, faster, lower power circuit that is also more scalable than the original. Although some of the area advantage is offset by requiring the initial high fan-in ANDs, this approach is overall still much better. One main reason that is not immediately obvious is that the possible implementation of the predecoders is more flexible since it can be separated from the memory arrays and will have less restrictions imposed by the memory array dimensions.

With this flexibility the predecoders can be implemented using sophisticated circuit designs that enable circuits with faster speed, lower power, and smaller area. Some of these techniques will be covered in more detail in a later subsection.

The application of these advanced circuit techniques to the first approach is not feasible because of the higher cost involved in applying it to a much larger number of gates (in the example, there are 256 gates for the simplistic approach and only 32 for the predecoder approach). At the same time, the implementation possibilities for gates embedded within the regular structure of the memory array are much more limited since they are affected by other factors including the cell-to-cell spacing, or pitch, of the memory arrays, as shown in Figure 5.8.

The main disadvantage of predecoding is the need to distribute more wires to propagate all the intermediate predecoder outputs. This is a minor issue, and the advantages of predecoding make it almost necessary in most SRAM designs.

Row and Column Decoding

For added flexibility in operation, memory arrays like SRAMs (and in later chapters, DRAMs) typically employ two-dimensional decoding, where a subset of the address accesses a single row of the array (the row address), and a separate subset is used to select a fraction of all of the columns accessed within the row, as shown in Figure 5.9.

In the figure, predecoding for both the row and the column addresses is not shown for simplicity, and it is assumed that the row and/or column decoder utilize these predecoder outputs. The multiplexer allows multiple bitlines to share a single sense amplifier, saving both power and area. These

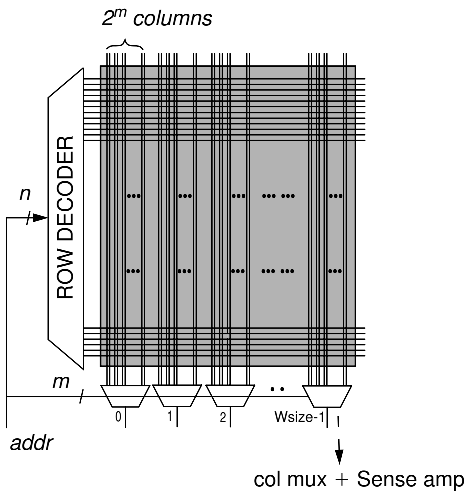


FIGURE 5.9: Two-dimensional decoding.

multiplexers are almost always of the pass transistor variety, allowing them to multiplex small-swing voltages in the bitlines. To simplify the figure, only the data read path is shown since address decoding for read and write operations is essentially the same.

Non-Partitioned

A non-partitioned decode hierarchy refers to a memory organization where all cells in a given row are activated by a single wordline output from the row decoder; the organization of the sample memory system in the previous subsection and Figure 5.9 are examples of this.

For very small SRAMs (i.e., 1 kB and smaller), this simple non-partitioned approach is sufficient, but as the size of the SRAM increases, several problems start to become significant:

1. As the number of memory cells in an SRAM increases with increasing memory size, more and more transistors from each memory cell are connected to the row's wordlines and the column's bitlines, increasing the total capacitance and resulting in an increase in delay and power dissipation.
2. Increasing the number of memory cells results in a physical lengthening of the SRAM array, increasing the wordline wire length and its parasitic wiring capacitance.
3. More power in the bitlines is wasted during each access because more and more columns are activated by a single wordline even though only a subset of these columns are actually accessed. In a non-partitioned scheme, every column will have an active memory cell, uselessly discharging the bitlines of the unaccessed columns and resulting in wasted power needed to precharge these bitlines back to their original value.

To a certain extent, this problem can be mitigated by minimizing the number of columns in an array. For a fixed memory size, this can be done by increasing the number of rows accordingly. This solution is obviously very shortsighted as it will eventually produce its own problems and doesn't solve the original one. Instead, the number of columns (and rows) is used as an additional parameter that can be changed to come up with an optimal memory partitioning and hierarchy.

Divided Wordline (DWL)

To solve the problems of the non-partitioned approach, the divided wordline (DWL) technique was introduced by Yoshimoto [Yoshimoto et al. 1983] and is now used in virtually all SRAMs because of its usefulness and minimal disadvantages.

DWL involves dividing the memory array along the top-level wordline (called the global wordline or GWL) into a fixed number of blocks. Instead of enabling all the cells within a row, the GWL is ANDed with a block-select signal (derived from another subset of the input address) and asserts a local wordline (LWL). Only cells connected to this asserted LWL are enabled.

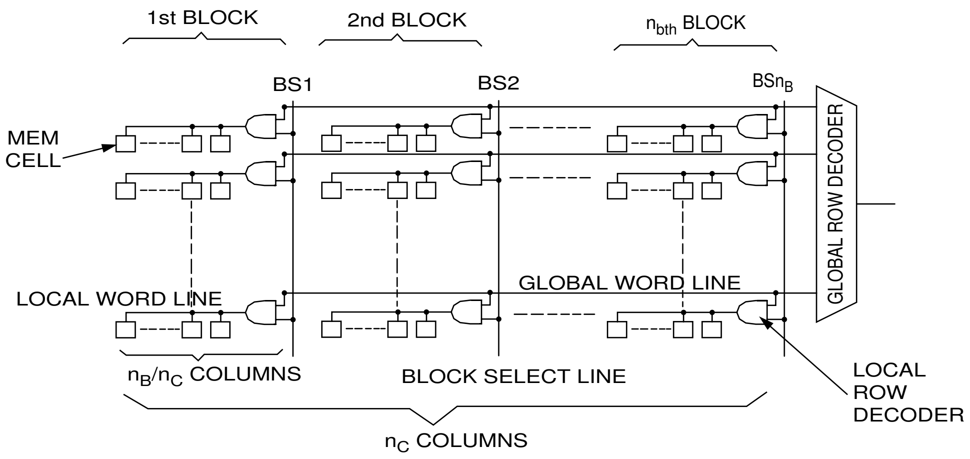


FIGURE 5.10: An SRAM using the DWL address decoding technique. (Figure used with permission from Yoshimoto et al. 1983.)

This DWL structure is shown in Figure 5.10, where the memory array is divided into n_B blocks. Assuming a total of n_C columns, each block contains n_C/n_B columns.

Since only a fraction of the total cells ($1/n_B$ to be exact) are connected to an asserted LWL and consequently activated during an access, the total power consumed in the bitlines is reduced significantly to roughly $1/n_B$ of the original. This value is exactly $1/n_B$ for reads since the nature of wasted power is similar to the power dissipated by the bitline read access, while the savings is less for a write access because of the higher power dissipated by accessed bitlines during a write.

Additionally, since fewer cells are connected to the LWLs, the power and delay will be significantly less. The GWL also only needs to drive the wire capacitance and the much smaller number of LWL decoders. The total wordline delay (the sum of the GWL and LWL delays), for the DWL compared to the non-partitioned implementation, will be much less for moderately sized SRAMs and above.

Figure 5.11 shows the effect in column power and wordline delay of changing the number of blocks for an 8k × 8 SRAM. Although this may seem like a small SRAM by today's standards, it is still useful as a basic building block for wide caches. For example, a cache with a 128-bit block can employ 16 of these small SRAMs, resulting in a 128-kB cache, which is getting close to the typical size of an L2 cache. Figure 5.11 shows that even though column power can be continuously reduced by increasing the number of blocks, the benefit to the wordline delay lessens, and it reaches a point where it starts to adversely affect the delay.

The implementation of the DWL technique is simple, requiring only additional block-select circuits and LWL decoders that can be made simple since they drive less cells.

For the 8k × 8 SRAM here, dividing the array into 8 blocks (nB = 8) results in only a 4–6% increase in area, while resulting in very significant power and speed improvements, compared to a non-partitioned design.

Hierarchical Word Decoding (HWD)

A logical extension of the DWL scheme is a technique called hierarchical word decoding (HWD) [Hirose et al. 1990], shown in Figure 5.12.

As the memory size increases, maintaining a DWL structure requires an increase in the number of blocks, which results in an increase in the GWL capacitance since more wordline decoders tap into the GWL. HWD simply introduces additional levels of hierarchy into the decoding scheme to more efficiently distribute capacitances, with the number of levels determined by the total load capacitance of the word decoding path. At 256 KB, the difference between DWL and HWD is insignificant, but a 4-MB SRAM using the HWD architecture can reduce delay time by 20% and total load capacitance by 30%.

Pulsed Wordline

Early SRAM implementations asserted the wordlines for a significant fraction of the cycle time. The wordlines are typically asserted early (after the decode delay) and deasserted late in the access. This method, while functionally correct, is inefficient [Amrutur and Horowitz 1994]. For a read access, wordline assertion causes one of the bitline pair to be pulled down,

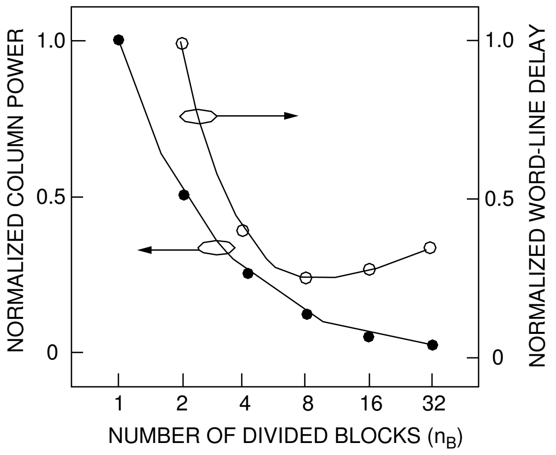


FIGURE 5.11: Graph showing the effect of the number of blocks, n_B, in the memory's column power and wordline delay. (Graph used with permission from Yoshimoto et al. 1993).

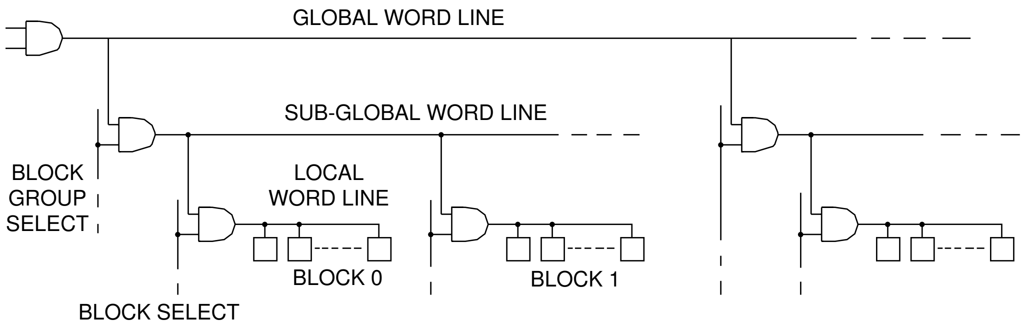


FIGURE 5.12: HWD architecture showing three levels of decoding. (Figure used with permission from Hirose et al. 1990).

creating a voltage differential across the bitlines. When a sufficient differential exists (the exact value will depend on the process technology and the offset voltage of the sense amplifier), a sense amp can amplify this differential and speed up sensing. At this point, any additional differential developed across the bitlines, as a result of the continued assertion of the wordline, will not significantly speed up sensing and will require more power and precharge time.

Most SRAMs now use some kind of pulsed wordline, where the wordline is allowed to assert only for a small amount of time necessary to create a sufficient bitline differential voltage, after which it is turned off. This technique prevents the development of bitline voltage differential more than is necessary, reducing the power dissipated during the precharge process.

The width of the pulsed wordline is controlled either using static delays (where the wordline is turned off after a certain number of delays fixed at design time or fixed by special delay circuitry during a built-in self-test) or using some feedback taken from information extracted from the circuits (to be discussed in detail later).

Figure 5.13 shows timing diagrams for a system with and without the pulsed-wordline (PWL) scheme. As can be seen from the diagram, output data from both systems is produced almost at the same time, but continued assertion of the wordline for non-PWL schemes results in larger differentials developed across the bitline, serving only to dissipate additional power during precharge.

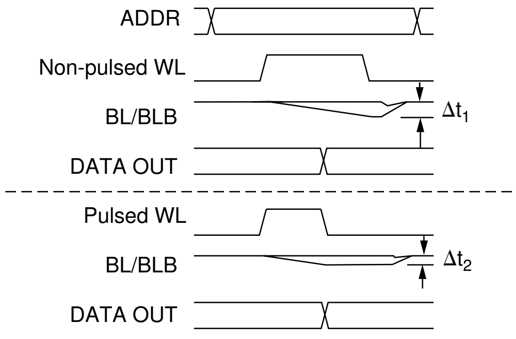


FIGURE 5.13: Timing diagrams showing pulsed and non-pulsed wordline schemes.

5.2.3 Physical Decoder Implementation

With the division of the decoder into multiple levels of hierarchy (i.e., predecoders, GWL decoders, LWL decoders, column decoders, block decoders, etc.), different circuit styles and implementations can be used for each decode level depending on different factors including power, speed, area, complexity, and layout requirements.

The relative importance of these specifications varies within the decoder hierarchy because of the presence of different constraints. As a quick example, predecoders can use relatively power-hungry circuit styles, which is not true of the LWL decoders simply because of the much larger number of LWL decoders in the circuit compared to predecoders.

Much research has been done to optimize the decoders in the entire hierarchy from the initial predecoder to the final LWL decoder [Yoshimoto et al. 1983, Yamamoto 1985, Sasaki et al. 1988, Aizaki 1990, Nakamura 1997, Mai et al. 1998, Nambu et al. 1998, Osada et al. 2001]. Because of space limitations, only a small (but very relevant and proven valuable) subset of these circuits will be discussed here in detail.

Digital Logic Styles

Before proceeding, it is beneficial to discuss different logic circuit styles and where in the decode hierarchy each style is suited. CMOS logic circuits can be generalized by the circuit shown in Figure 5.14. The figure shows an output node connected to Vdd by a pullup network and connected to ground by a pulldown network, where both networks are controlled by inputs (e.g., data and/or a clock signal).

The pullup network consists of a combination of PMOS transistors that conditionally connect the output to Vdd, while the pulldown network consists of a combination of NMOS transistors that conditionally connect the output node to ground. Although the inputs and the clock signal are shown to be provided

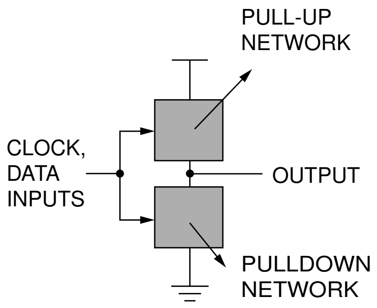


FIGURE 5.14: Generalized CMOS logic circuit.

to both the pullup and pulldown networks, they may or may not be utilized depending on the specific implementation.

Static CMOS The first logic style we review is *static* CMOS one in which there always exists a relatively low impedance path from output to either V_{dd} or ground depending on the logic inputs and desired output. This distinction will be clearer when *dynamic* CMOS is discussed, where we will see that some logic values depend on the charge stored within parasitic capacitances in the gates themselves.

Static CMOS is the easiest and most robust implementation, since there are fewer variations and problems associated with the design. The two variations of the static CMOS style are the PMOS- or active-load and the full-CMOS styles. Figure 5.15 illustrates 3-input NAND and NOR gates for both styles. The main difference of the two styles is the implementation of the pullup net (notice that the pulldown networks are exactly the same). The full-CMOS implementation utilizes a pullup net that is the complement of the pulldown (i.e., parallel connections become series, and vice-versa), while the active-load uses an always-on PMOS transistor pulling up the output constantly to V_{dd}.

The main results of this difference are the following:

1. There is a typically smaller area for the active-load design because of fewer transistors.
2. The total active-load gate capacitance for each input is typically less than half of the full-CMOS implementation. This results in a smaller logical effort, making this gate easier to drive.
3. The always-on PMOS in the active-load design will result in significant static power dissipation whenever the pulldown net is enabled. In addition, the logic low voltage will not reach the full Vdd value because of the pulling up effect of the PMOS and how it fights the pulldown network. The exact output value will depend on the relative strengths of the transistors.

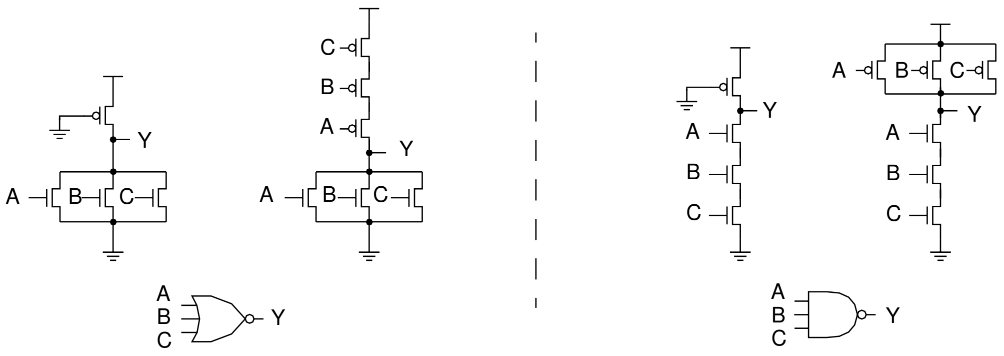


FIGURE 5.15: Active-load and full-CMOS circuit styles for 3-input NOR and NAND gates.

Speed and power comparisons of these circuits are interesting [Sasaki et al. 1988]. Given equal capacitance characteristics, the PMOS-load circuit will produce about 8% more delay than the full-CMOS decoder (because the PMOS-load will tend to fight the pulldown net, producing delay); however, the capacitance characteristics of these circuits, in practice, will usually be different and will tend to make the PMOS-load faster (about 15% faster). In addition, average currents in their decoder (which directly relates to power consumption) show that even though the PMOS-load consumes DC current, there exists a crossover point where cycle times become smaller and AC current becomes more dominant. It must be stressed, though, that these current numbers are greatly influenced by the sequencing and control of the gates.

Because of these factors, the speed comparison will be relevant regardless of how the memory is controlled, but the power comparison needs to be studied on a case-to-case basis to take into account the specific characteristics of a system. For example, a PMOS-load used with the PWL technique will be active for a smaller amount of time than if a non-PWL technique is used. This makes active-load circuits feasible for use in LWL decoders using PWL. Compared to a full-CMOS system, the speed is increased, and yet not too much power is wasted, since only a very small fraction of LWL decoders are active at the same time.

Dynamic CMOS One of the main objectives of dynamic CMOS is to minimize the capacitance of the logic gate inputs. At its simplest, this is done by implementing only either the pulldown or pullup network. Further discussions will involve only the dynamic CMOS circuits with the pulldown network, as, because of the better characteristics of NMOS transistors compared to PMOS transistors, pull up networks tend to be useful only in special applications.

Dynamic CMOS is similar to the circuit of the PMOS active-load gate. The main difference is that instead of having always-on loads, dynamic CMOS uses clocked elements that precharge (in this case, charge up to Vdd) the output node during the precharge phase. This output node is later conditionally discharged during the evaluate phase depending on the state of the inputs.

Figure 5.16 shows simple dynamic implementations of 3-input NAND and NOR gates, along with a timing diagram showing the precharge and evaluate phases of the dynamic gate operation.

Note the presence of the additional clocked NMOS in the pulldown net, called "foot" transistors. During

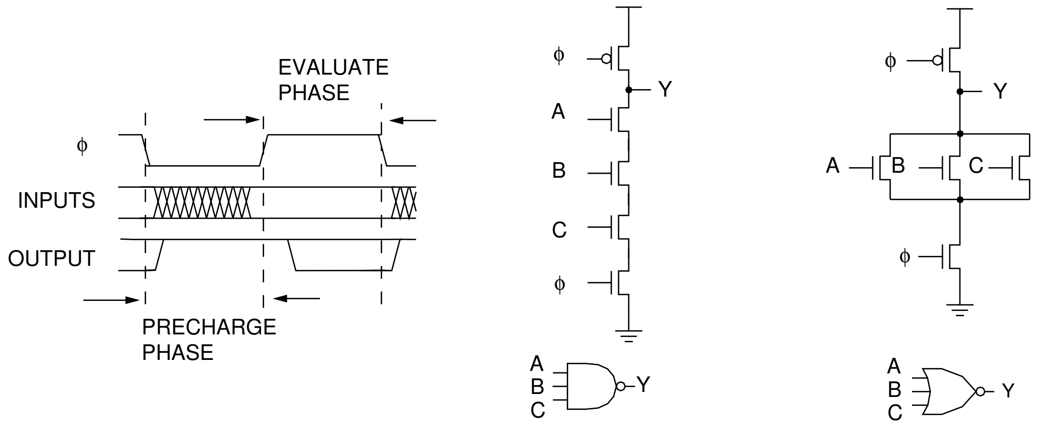


FIGURE 5.16: Simple dynamic NOR and NAND gates. Also shown is a generic timing diagram showing the precharge and evaluate phase.

the precharge phase, the PMOS precharge transistor precharges the output node Y to a logic high voltage. The precharge NMOS is not enabled, to prevent the possible inadvertent early discharge of the output and to prevent forming a low-impedance path from Vdd to ground. The evaluate phase is started when ϕ asserts, disabling the PMOS and enabling the NMOS input transistors. At this point, the output node dynamically stores charge that is conditionally discharged by the pulldown network if the right combination of input signals exists.

It is important to emphasize that inputs to dynamic gates must be carefully designed not to inadvertently discharge the output node. In general, we require the inputs to a dynamic gate to be valid before the evaluate phase or to change value monotonically to prevent an inadvertent discharge of the dynamic output node.

Domino Dynamic Logic

In general, individual dynamic logic gates are implemented with buffers at the output to provide more robust driving capability and to protect the dynamic node from being corrupted by noise. These buffers are usually implemented using full-CMOS inverters. This gives rise to a style called domino logic, and Figure 5.17 shows a circuit where domino gates are cascaded together.

During the precharge phase, all internal dynamic nodes are precharged high, causing Y1, Y2, and Y3 to go low (the inverted version of the dynamic nodes). At the start of the evaluate phase (ϕ = 1, and for simplicity, assuming all other inputs required to form a pulldown path are already high), the dynamic node of gate 1 is discharged to ground, and consequently Y1 goes high. Y1 then discharges the dynamic node of gate 2, eventually causing Y2 to go high. This in turn enables the NMOS in gate 3 and causes Y3 to go high. This sequence of events gives rise to the name "Domino Logic," where the outputs of a gate cause downstream gates to be activated, even though all of the gates are put in the evaluate phase simultaneously. In domino logic, all outputs are initialized to a precharge value, and once the evaluate phase starts, the primary input causes a domino effect that eventually reaches the primary output.

An additional advantage in using cascaded domino gates is the possibility of removing all the NMOS foot transistors in all of the pulldown stages after the first one. When doing this, it must be ensured that no low-impedance path from Vdd to ground is created during the operation. If some inputs are supplied by non-domino logic, further analysis has to be done to ensure that the discharge path does not form. If all inputs are coming from other domino logic gates, it can be ensured that precharging the domino gates is enough to make sure that no pulldown path will be

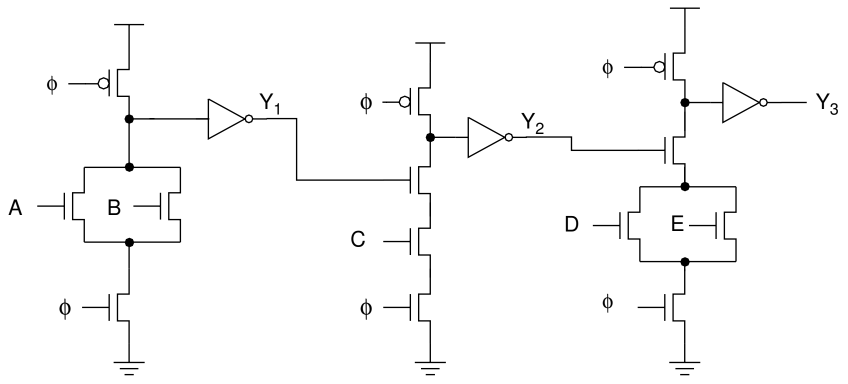


FIGURE 5.17: Cascaded domino gates. For simplicity, inputs not shown are assumed to also be driven by domino gates.

formed, making the clocked-NMOS unnecessary and making the gates both smaller (fewer transistors) and faster (less effective resistance and, hence, larger current drive).

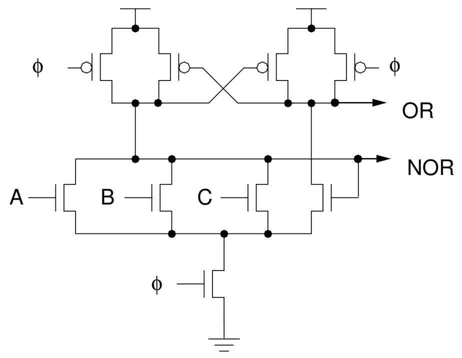
Although domino logic has numerous advantages, it also introduces additional concerns, among them lower noise tolerance and the need of a clocking scheme to generate the precharge signal. As more and more dynamic gates are used in the system, the power dissipated by the precharge signal also increases. Because of this, more sophisticated dynamic circuits have been proposed and used for memory circuits in the form of self-resetting logic (discussed in a moment).

Source-Coupled Logic (SCL) One dynamic technique that has been used in memory decoders is the source-coupled logic (SCL) [Nambu et al. 1998 a,b]. (Note: It must be mentioned that the name source-coupled logic has been used for other completely unrelated techniques, and we chose to retain the original author's naming convention.)

A 3-input SCL NOR/OR gate is shown in Figure 5.18, along with a timing diagram showing its operation. This SCL circuit is similar to the basic dynamic 3-input NOR gate with an additional cascaded inverting branch and additional cross-coupled PMOS pullups to maintain stability.

This SCL circuit has three main advantages. First, increasing the gate fan-in causes only an insignificant increase in delay; unlike a full CMOS NOR, transistors added for fan-in are connected in parallel, and their associated diffusion capacities at the NOR output do not add as they would were they series-connected transistors in a PMOS NOR pull-up network. Note this is true for dynamic NOR gates in general. Second, the output delay of the NOR and OR signals are the same, which is not true of circuits deriving the complement signal using an additional inverter. This reduces the worst-case delay of the gate. Finally, a subtle, but very important advantage of this circuit is shown in Figure 5.19. Here, the gate output is considered active (i.e., selected) when the output is low. The timing diagram shows the activity of a domino OR and an SCL OR for two cycles, the first where both are unselected, and the second where both are selected. The important observation here is that the dynamic OR output undergoes a transition when it is unselected, it stays the same. Otherwise, When this dynamic OR is used as a predecoder gate that drives the GWL drivers of every row, all the unselected predecode lines will burn power because of the transistions of the unselected outputs. On the other hand, SCL eliminates this unnecessary power dissipation by ensuring that only the single selected predecode wire will burn power.

Here, the presence of the BUF-OR is only to emphasize that power is dissipated only for transitions of the OR output. The SCL still burns power in the internal



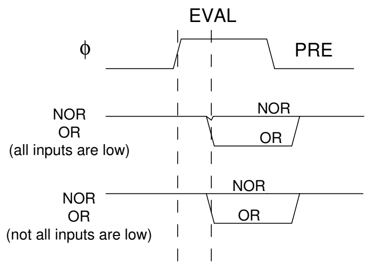


FIGURE 5.18: A 3-input SCL NOR/OR gate.

node transitions when it is unselected, but this is almost negligible compared to the power burned by the final gate output when driving its large load.

As mentioned earlier, the advantages of SCL become much clearer in the context of a decode hierarchy, where many NOR gates will exist (equivalent to many multiple-input ANDs) with only a very small minority of them being selected. It is therefore very desirable to have only this small minority burn significant power, as opposed to having most of the gates needlessly burn power.

Buffering and Skewing Before discussing the next two logic styles, we take some time to discuss the concept of buffering and device skewing in the context of memory decoders.

Figure 5.20 shows a typical SRAM floor plan showing where predecoders, GWL decoders, and LWL decoders are located. The figure shows an example amount wire length being driven by the various components of the decoder. The GWL decoder, for example, is driving 2 × 800 μm of wire. Given sample parameters of the TSMC 0.25-μm process taken from

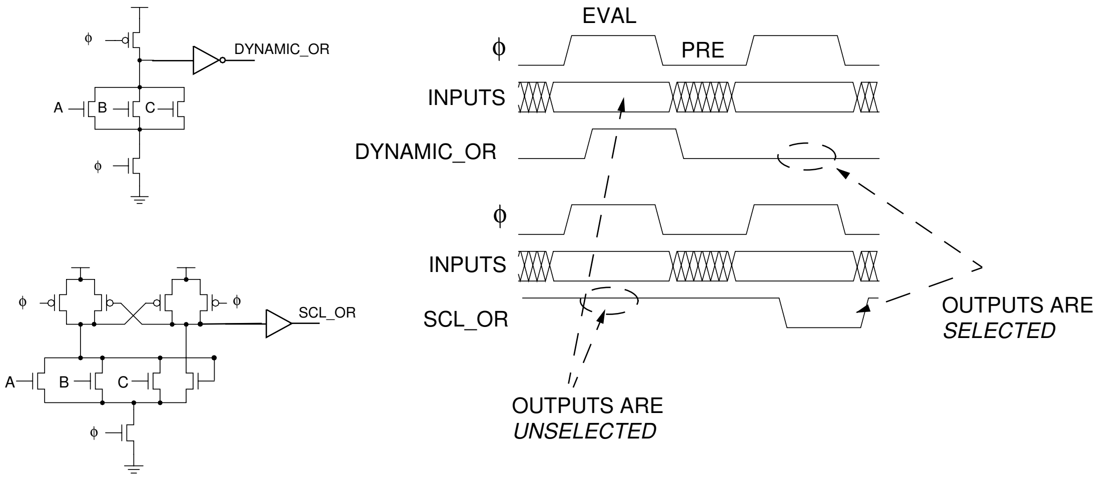


FIGURE 5.19: Comparison of domino and SCL 3-input OR gates.

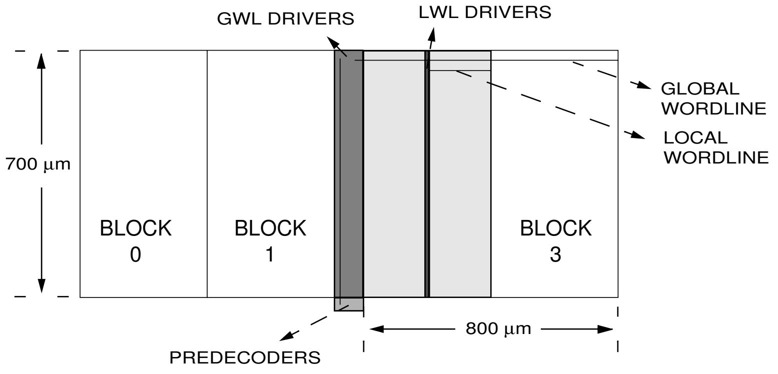


FIGURE 5.20: Example SRAM floor plan showing the memory arrays and the decoder hierarchy.

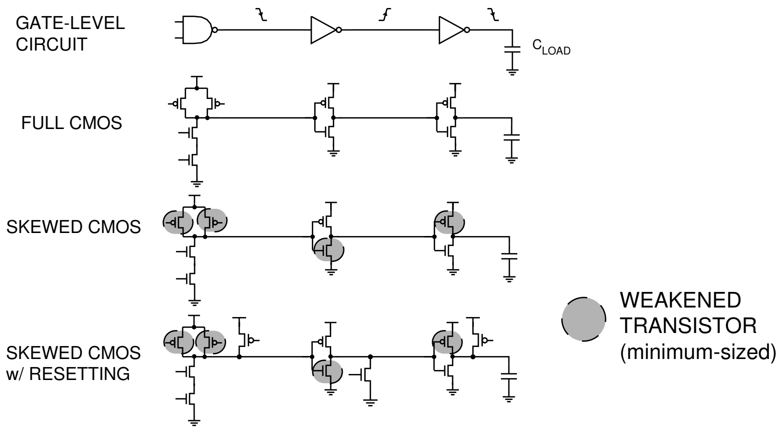


FIGURE 5.21: Different circuits demonstrating how to speed up an output edge.

MOSIS, a 1-µm thick metal 3 line of this length has roughly 200 fF effective capacitance.

This kind of load, in addition to the transistor capacitances connected to these lines, cannot be driven efficiently by any single-stage gate, and we will have to insert buffer chains to the outputs of our gates to drive the load properly while optimizing the delay, as shown in Figure 5.22. This is done by sizing each stage properly, which has been a very widely studied problem [Jaeger 1975, Cherkauer and Friedman 1995]. A rule of thumb [Jaeger 1975] is that delay optimization requires the delay of each stage to be the same and that the practical fan-out of each gate is set to be about 4. In addition, when it is important to speed up only one specific edge of the output, we can skew the devices inside the gates and buffers to speed up the important transition.

This device skewing is very useful in decoder design because it is important to speed up the assertion of the wordline to activate the memory cell access transistors, while the time to deassert the gates producing the wordline is less critical since it can be done in parallel with other operations within the SRAM. Figure 5.21 shows a gate-level circuit and three transistor-level circuits demonstrating these concepts.

The first circuit shows the gate-level representation that is equivalent to a 2-input NAND gate with the inverters assumed to be sized to drive the load optimally. In this example, we want to optimize the falling edge at the last output. Given this information, we can determine which edge has to be favored at every gate in the chain.

The second circuit simply shows the basic full-CMOS implementation of the 2-input NAND gate and the inverters where the drive strength of the pullup and pulldown network are the same to yield roughly the same speed for both the rising and the falling edge.

The third circuit is derived directly from the second circuit, where in all the transistors not participating in the favored edge are weakened and made minimum size. For example, the last stage has the PMOS weakened

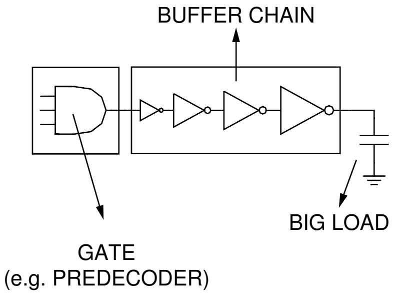


FIGURE 5.22: Gate sizing.

since it does not help in pulling down or discharging the output to get the desired fast falling edge.

The main advantage of this skewing technique is the reduction of the gate's logical effort because of the reduction of its input capacitances resulting from weakening some of the transistors significantly, making them easier to drive and resulting in significant speedup. The problem in weakening some of the transistors is that the reverse transition will proceed much more slowly than before, probably negating any speedup in the forward path by requiring a much longer reset period.

To offset this problem, additional reset transistors are inserted, as shown in the third circuit. These reset transistors are as strong as their full-CMOS counterpart, providing enough strength to perform the reset properly. A simple way to activate these reset transistors is to treat them as precharge logic for dynamic circuits and control them through an external precharge clock (being careful that no path from Vdd to ground is enabled during precharge). This is a simple method to use since no special circuits have to be designed other than distributing the existing precharge clock (assuming the system already has one). But in the context of decoder design, this method will be very power inefficient because the precharge logic has to be fed to all gates using this technique even though most of these gates that are within the decoder tend to be inactive and will not require resetting. This results in an unnecessary increase in the power dissipation of the precharge clock.

One way to solve this power inefficiency is to generate localized reset signals such that only the few gates that are activated generate reset signals and burn power driving the reset transistors. Two different methods that accomplish this are discussed next.

SRCMOS Self-resetting CMOS, or SRCMOS [Chappel et al. 1991, Park et al. 1998], uses its own output pulse to generate a local reset signal to precharge its transistors. As long as no output pulse is produced by the gate, the reset transistors are inactive and do not burn dynamic power.

Figure 5.23 shows two SRCMOS circuits, including the timing waveforms as a result of the pulse-mode inputs A and B. The first circuit is derived from the

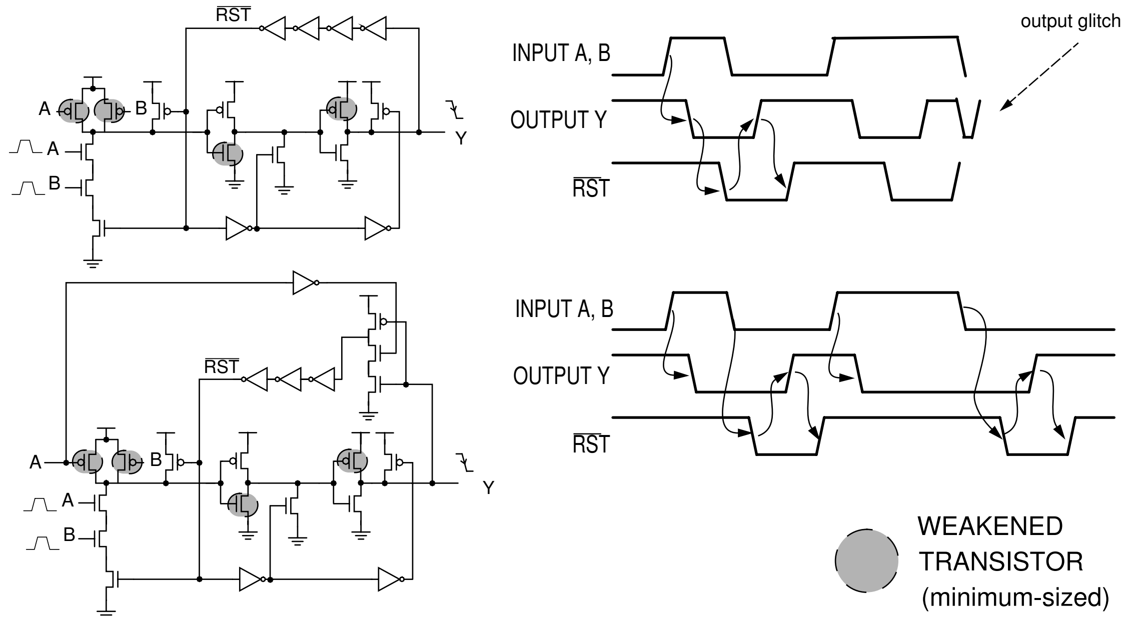


FIGURE 5.23: SRCMOS 2-input NAND circuits.

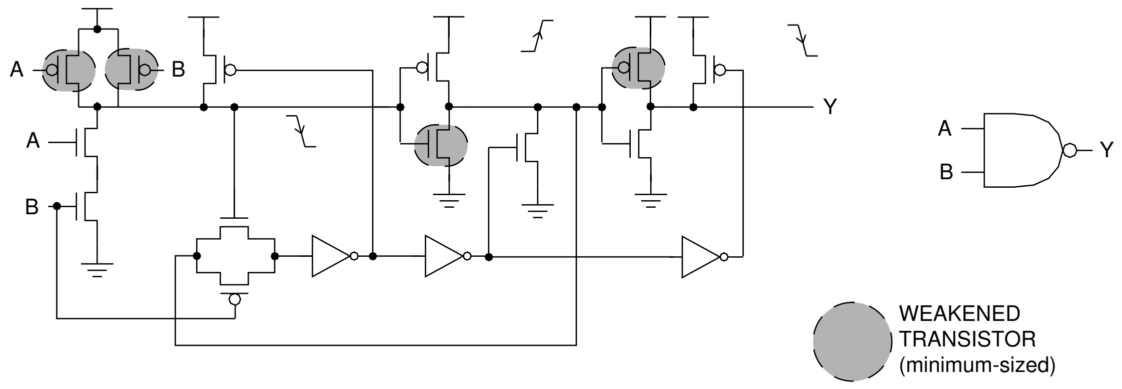


FIGURE 5.24: DRCMOS 2-input NAND circuit.

skewed CMOS with reset transistors in Figure 5.21. A delay chain has been inserted to derive the reset signals of all reset transistors from the output pulse. Without this pulse, all these circuits are quiet and burn no AC power. Once the reset signal has initialized the state of the gate, the new state travels back to deassert this reset. In addition, an extra series NMOS has been inserted in the initial NAND gate to disable the pulldown network in case the pulse-mode inputs are not guaranteed to turn off in time before the reset signal reenables the NAND PMOS for the next state of inputs, thereby creating a current path from Vdd to ground.

One problem with the first circuit exists if the pulse widths of the inputs are large enough that they are still active at the time when the reset signal reaches its original deasserted state after the entire reset process. When this occurs, an additional, undesired pulse will be produced at the output. In this case, one solution is to predicate the initial reset upon the falling edge of one of the inputs, as shown in the second circuit, where the first inverter in the delay chain requires one of the inputs to be low to be activated.

DRCMOS The extra series NMOS in SRCMOS will tend to increase the logical effort of the gate, which reduces the initial benefits of SRCMOS. Dynamically Resettable CMOS, or DRCMOS [Nambu et al. 1998a,b, Amrutur and Horowitz 1998, Heald and Hoist 1993] solves this problem by predicating the reset signal activation with the falling input even for the initial propagation around the loop, as shown in Figure 5.24.

At its initial state, the transmission gate NMOS is enabled to initially provide no reset signals. Once the input and NAND output changes, the transmission gate is disabled and then enabled only when the input goes low again, allowing the reset signal to propagate through the gates and eventually return itself automatically back to its initial state.

With the removal of the extra series NMOS, the DRCMOS technique will have a lower logical effort and be faster than even SRCMOS logic. The main caveat is that the output pulse width is always longer than the input pulse width since the output is reset only after the inputs finish. This limits the number of levels in the decode path that DRCMOS can use without exceeding the cycle time.

The SRCMOS and DRCMOS techniques can be used for different kinds of logic, including full-CMOS logic used in the previous examples. Figure 5.25 shows how the SCL technique, discussed previously, can utilize DRCMOS logic in implementing a NOR-style decoder that is useful as a 4-to-16 predecoder [Amrutur and Horowitz 1998].

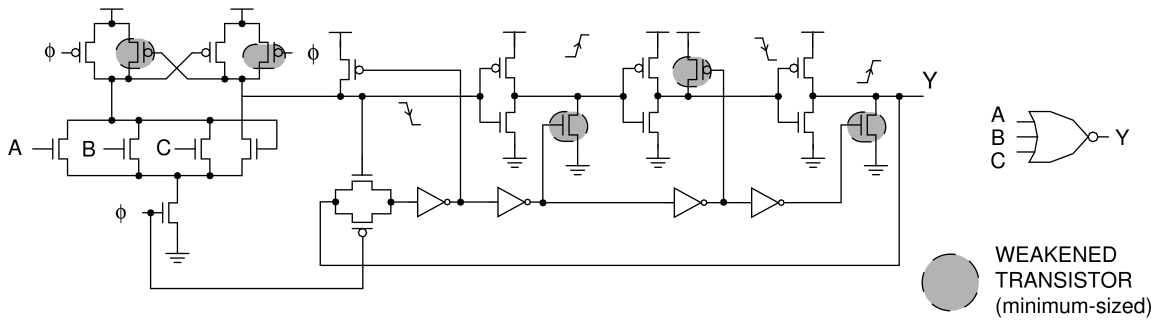


FIGURE 5.25: A 4-input SCL NOR circuit using the DRCMOS technique.

In this case, the delayed reset is predicated upon the precharge clock. It must be emphasized that although all gates of these types use the precharge clock, only the gates whose outputs actually undergo transitions will burn power to turn on the reset transistors.

SCL and DRCMOS complement each other in avoiding unnecessary power dissipation, since SCL logic only undergoes output transitions whenever they are activated and selected by the right combination of inputs. By not undergoing unnecessary transitions, DRCMOS further saves power in the reset circuitry and, at the same time, speeds up the favored transition due to its low logical effort and device-skewed implementation while keeping the reset period manageable.

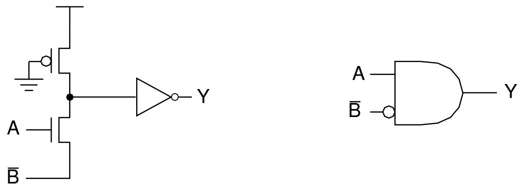


FIGURE 5.26: Transfer-word driver (TWD) circuit.

Transfer-Word Driver (TWD) As an alternative to full-CMOS NAND gates, the transfer-word driver (TWD) shown in Figure 5.26 can be used [Aizaki 1990]. By using a single, always-on PMOS as a pullup, and by using a single NMOS transistor to perform the ANDing operation, the circuit has a reduced input capacitance compared to a full-CMOS gate (less than half depending on PMOS sizing of the full-CMOS gate). This allows the TWD gate to operate faster than its full-CMOS counterpart. In addition, its simplicity results in a smaller decoder area (although this is not overly significant—20% according to Aizaki—because the area of the decoder circuit will typically be dominated by the drivers, not the initial stage).

It should be noted from Figure 5.26 that the TWD configuration needs one of its inputs to be low asserted to perform the same functionality as a NAND gate. This is not typically a problem when used as part of a decode hierarchy, as the inversion can be done using an extra driver stage. This additional stage should not affect the critical path because it can be used on the non-critical part of the decode. For example, when used in the DWL/LDL decoder, the TWD ANDs together the GNL and block-select signal to drive the LDL; the block select usually arrives earlier than the GWL signal and can even absorb another stage if inversion if necessary.

The main potential disadvantage of the TWD circuit is the static current drawn because of the path from Vdd to ground whenever the gate is active (i.e., the NMOS is activated). Although this makes the TWD gate impractical for use in general circuits, it is only a minor concern when used in LWL decoders. The decreased gate capacitance lowers dynamic power and helps compensate for the static power. In addition, only a very, very small fraction of these gates within LWL decoders would actually be activated (and hence, dissipate static power) during any given access. Moreover, the use of PWL techniques serves to minimize the amount of time these gates are active, further decreasing the amount of static power dissipation.

5.2.4 Peripheral Bitline Circuits

To support the operation of the SRAM memory cells, the cells are accompanied by additional peripheral circuitry, as shown in Figure 5.27.

Referring again to part of Figure 5.1 (which is replicated in Figure 5.28), all bitlines are assumed to be precharged to a given voltage (currently, this is most often Vdd). When WL0 asserts, all the memory cells connected to this wordline have their access transistors enabled. For a read operation, the accessed memory cells are allowed to pull down the voltage of the bitline (the other bitline will be pulled up because of the complementary signals stored in the cell). Since the initial bitline voltage is already high, it is usually the pulldown operation that is important. The voltage of the bitline being pulled down will steadily drop, with the rate of change dependent on mainly the bitline capacitance (made up of the diffusion capacitances of each access transistor connected to the bitline and various wire parasitics) and

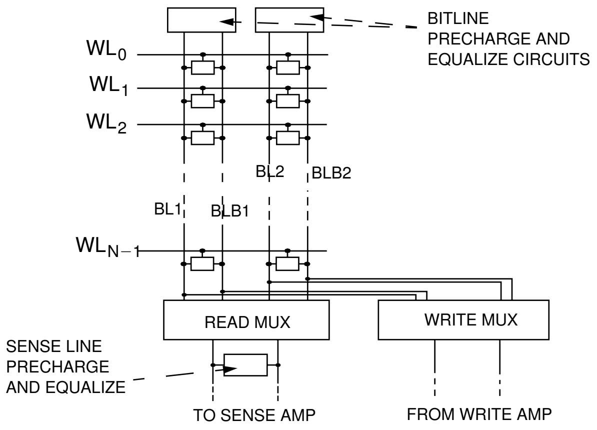


FIGURE 5.27: Memory cells showing peripheral bitline circuits, including precharge devices and read/write muxes.

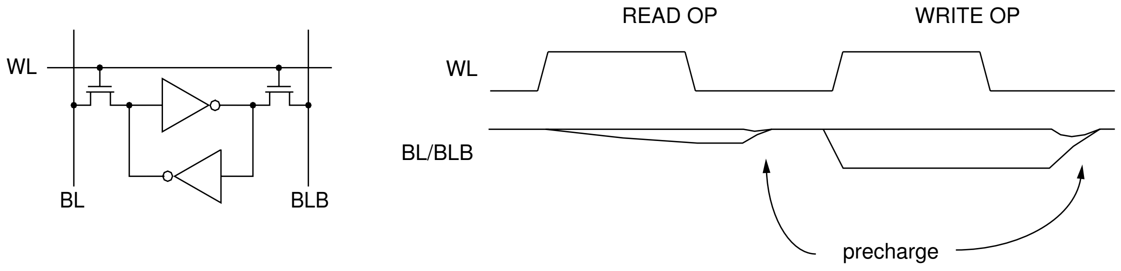


FIGURE 5.28: Read and write timing diagram for a memory cell, including the precharge operations.

the strength of the driver and access transistors. This process continues until WL0 is deactivated. Depending on the type of precharge circuit, the bitline voltage will either start to be precharged or stay constant (more on this later). In the timing diagram shown, it is assumed to stay constant.

During this process, the read mux selects the desired column based on the address input and relays its voltage difference to a sense amplifier that serves to speed up the sensing process. Before the next operation can occur, the bitlines (and the sense lines) again must be precharged to a fixed initial value. This is done by the precharge and equalization circuit in the bitlines and sense lines.

As WL0 is asserted for the second time for a write operation, the access transistors connected to WL0 are again enabled. But this time, external data from the write amplifier has imposed a full-swing differential voltage on the bitlines. Once the access transistors are enabled, the higher capacitance of the bitline and the stronger drive strength of the write amp forces the accessed memory cells to have the same logic value as the bitlines. As with the read operation, the bitlines are again precharged to an initial value after the operation.

We now discuss individual parts of the bitline peripheral circuits in more detail.

Precharge and Equalize Circuits

Various methods have been used for precharge and equalization, and some representative circuits are shown in Figure 5.29. The first circuit is one of the early implementations. It uses a diode-connected NMOS pair without equalization circuits. This configuration precharges the bitlines to Vdd–Vt, where Vt is the threshold voltage of the NMOS.

This configuration continuously tries to pull up the bitlines high, and this serves both as a strength and a weakness. Since there is no need for additional control to enable precharge, this circuit helps to control complexity. But at the same time, the constant current path provided by the transistor toward Vdd tends to slow down development of a bitline voltage differential during read operations (since it fights the memory cell pulling the bitline down) and burns more power during write operations where one of the bitlines is pulled low by the write amp, again fighting this constant pullup. In some cases, these pullups are implemented using always-on PMOS transistors.

The second circuit shows diode-connected NMOS transistors with a PMOS transistor bridging the two bitlines. This PMOS transistor serves to equalize the voltage between the two bitlines whenever it is enabled. This becomes especially important for active NMOS-load because of possible variations between the threshold voltage of the NMOS, causing a difference in the precharge level. As mentioned earlier, the active NMOS pullups can also be replaced by PMOS transistors if the inherent Vt level-shift due to the diode connection is unnecessary. (A typical use of this voltage shift is to adjust the bitline common-mode voltage to fit the sense amplifier's high-gain region.)

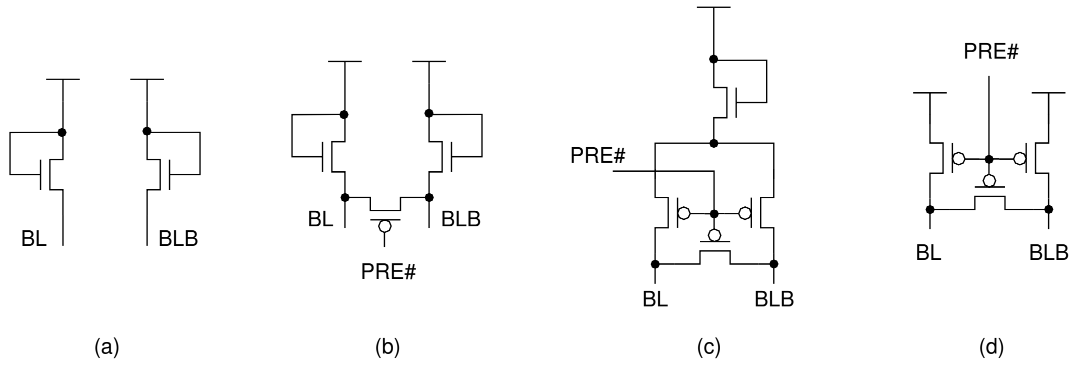


FIGURE 5.29: Different precharge and equalize circuitry.

The third circuit uses a combination of PMOS and NMOS transistors for the precharge and equalization circuits. It uses a single NMOS to establish a precharge level of Vdd–Vt, but it uses PMOS pass transistors to selectively connect the load to the bitlines only during precharge operations (more on this in the explanation of the next circuit). This kind of configuration was typical for moderate supply voltages (e.g., 3.3 V*)* or when the sense amplifiers used for amplification performed more optimally only at common-mode voltage levels below the supply voltage. But as supply voltages go down, and Vt differences due to process variations may significantly affect the performance, this configuration becomes impractical.

The last circuit shown is currently the most popular implementation. During precharge and equalize operations, all three PMOS transistors are enabled, providing low-impedance paths from both bitlines to Vdd and to each other. When the bitlines have been precharged, these transistors are turned off, resulting in very high impedances isolating Vdd from both bitlines. With this circuit, development of the bitline voltage differential is sped up because the pulldown only has to discharge the existing bitline capacitance. During write operations, no unnecessary power is wasted in the precharge circuit since it does not affect the operation of the write amp. The main disadvantage of this technique is the additional complexity and power needed to control the clocked precharge and equalization transistors. Although power dissipation during clocking of these precharge elements within the whole SRAM may be lessened by special circuits with conditional locally generated control signals, power will always be dissipated when switching the gates of these transistors on and off.

The sizing of these precharge transistors is dictated by how much time is allocated to the precharge operation. Larger transistors are able to precharge the bitlines faster, but they will dissipate more power. For example, using larger transistors for the hi-Z precharge implementation will dissipate more power in the precharge clock network (either global or local) because of the larger gate capacitances. The write operation often dictates how big these precharge transistors have to be because bitlines during writes are discharged completely, unlike the partial discharge due to a typical read.

To take advantage of the difference between the read and write operation, the precharge circuits are often separated into read precharge and write precharge. The read precharge is usually enabled every time and is sized small enough to charge up the bitlines from the expected bitline voltage drop during a read. The write precharge is enabled only during writes and serves to help the read precharge to charge up the bitline. In clocked schemes, this avoids the power dissipated by in unnecessarily switching the gate inputs of big write precharge transistors during read operations.

Read and Write Multiplexers

Read and write multiplexers allow multiple bitlines to share common sense amps and write amps, as shown by the example in Figure 5.30, where a single sense amp and write amp are shared by multiple bitline pairs. The mux can easily be expanded to accomodate more bitline pairs, and these considerations are discussed later in the partitioning subsection.

The most efficient way of implementing these muxes is the use of simple pass transistors. For the read mux, PMOS transistors are used since the common-mode voltages that need to be passed through are near the logic high value, resulting in better device transconductance. For the write mux, the write amp will try to discharge one of the bitlines, requiring an NMOS pass transistor to pass a strong logic low value.

In both cases, only a single pass transistor and not a complementary transmission gate is needed since the complement transistor will not be used effectively. During reads, an NMOS will often go unused since it will not be able to pass voltages above Vdd–Vt, and modern SRAMs do not develop enough differential to dip below this value. During writes, only a logic low has to be transmitted to one side of the bitline pair, with the perfectly reasonable assumption that the bitlines being precharged high are enough to form a full-swing voltage differential to force a write to the memory cell.

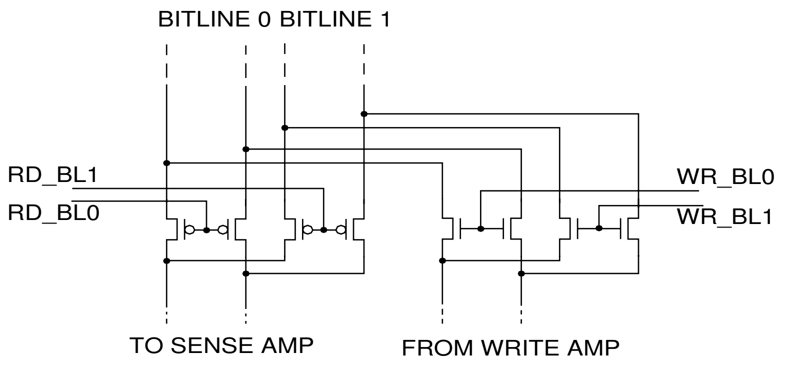


FIGURE 5.30: A circuit showing two bitline pairs sharing a single sense amp and write amp using read and write multiplexers.

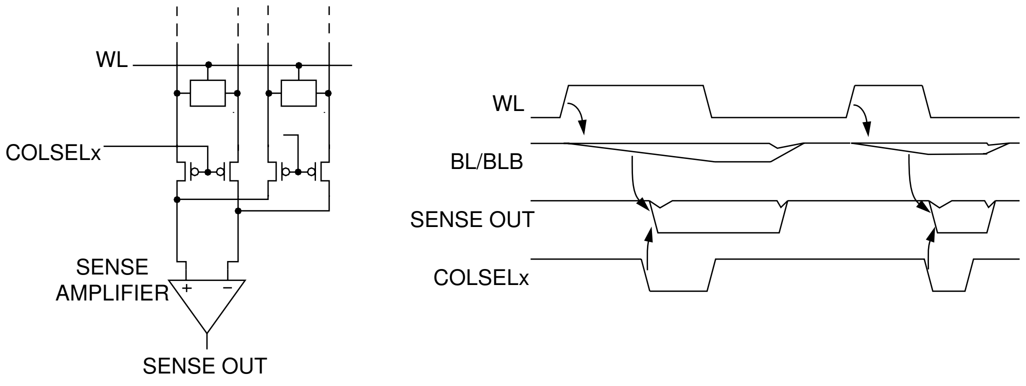


FIGURE 5.31: Use of a sense amplifier.

5.2.5 Sense Amplifiers

The development of a voltage differential in the bitline pairs during a read access is a slow process because the pulldown device of the memory cell is hard pressed to discharge the large capacitive load of the bitline.

A sense amplifier can be used to speed up the development of this bitline voltage differential. Figure 5.31 shows the operation of a sense amplifier along with a timing diagram showing some signal waveforms. When the wordline asserts, a differential slowly develops across the bitline pair. After some time, the PMOS pass gate is enabled (usually only after a minimum differential has been established as required by the particular sense amp). The sense amp then amplifies the bitline differential, quickly producing a full-swing differential output.

Note again that further development of the bitline differential is unnecessary and will actually be undesirable because more power will be dissipated to precharge the bitline. The second read access demonstrates how using a PWL serves to conserve power (by decreasing the final bitline differential) without any sacrifice in sense speed.

Although often advantageous, the use of sense amps is not absolutely necessary for sensing since it simply speeds up the development of the differential across the bitlines, which is continuously being discharged by the memory cell (as long as its wordline is enabled). This is unlike DRAM, where the differential voltage being sensed is due to a one-time contribution from the memory cell capacitor, which necessitates the use of sensing. In fact, most very small memories like register files often do not use sense amplifiers, as the bitline capacitances of these structures are often small enough to be discharged quickly by a memory cell; by doing so, they avoid the significant power consumed by typical sense amps.

Some contemporary caches that rely on single-ended sensing (like the Intel Itanium) also do away with the sense amplifiers in their cache hierarchy.

Physical Implementation

A large collection of different sense amp circuits can be found in the literature. These circuits range from simple cross-coupled inverters to very sophisticated amplifier circuits. Some of the more common sense amp circuits are shown in Figure 5.32.

The first circuit is the simple latch-type sense amplifier [Uchiyama et al. 1991] that forms cross-coupled inverters whenever the sense amp is enabled. Because of its simplicity, this sense amplifier occupies a very small area, and often satisfies the speed, area, and power trade-offs involved in designing wide memories that require a significant number of sense amps. One problem with the latch-type sense amp is the requirement to enable the amplifier only after a minimum voltage differential is present in the bitlines; otherwise, the latch could flip in the wrong direction and overpower the bitline differential.

The second and third circuits employ current mirror amplifiers. The first circuit uses a single current mirror amplifier, while the second circuit uses dual current mirror amplifiers and is named a paired current mirror amplifier (PCMA). These amplifiers offer fast sense speed, large voltage gain, and good output voltage stability. One problem with these circuits is their high power dissipation because of the existence of a static current path from Vdd to ground whenever the amplifier is enabled. In addition, this current becomes larger as the amplifier is designed to become faster. Moreover, its factor-of-two complexity wakes the PCMA circuit an impractical choice for wide-word applications that require large numbers of sense amplifiers.

The fourth circuit is the PMOS cross-coupled amplifier (PCCA) [Sasaki et al. 1989]. It offers fast sense speed at a much lower current than the PCMA because no static current path exists when the sense amp is enabled. The problem with the PCCA is its need for some preamplification to achieve its fast operation, and it is often better to pair it with a preamp like a PCMA or a latch-type sense amp to avoid spurious data output because of its high voltage gain. This is especially true when paired transistors in the amplifiers are mismatched [Sasaki et al. 1990].

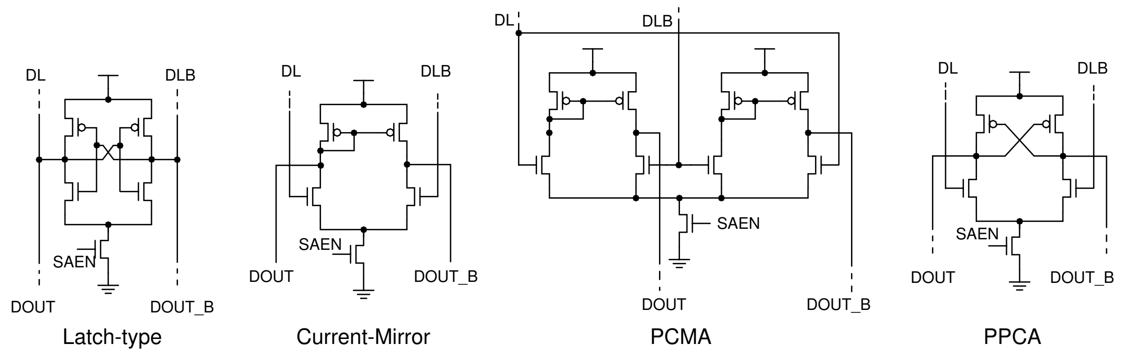


FIGURE 5.32: Sense amplifier circuits.

Sensing Hierarchy

Large megabit SRAMs typically use multi-level sense amp hierarchies such as the one shown in Figure 5.33. The figure shows a scheme with three levels of hierarchy and a single bit of output data. Oftentimes, amplifiers in different levels have differing topologies to optimize the entire hierarchy.

Although these types of sensing hierarchies are almost a given in discrete SRAMs, they are not widely used in typical on-core cache implementations, where a single level of sensing is most often adequate, and, in fact, some contemporary caches even dispense with the amplifier altogether (like the Intel Itanium). The main reason for this is the wide-word nature of caches, which is typically much greater than 32 bits in length, as opposed to most discrete SRAMs, which are typically from 1 to 9 or 18 bits wide.

The wide output of typical caches often necessitates the use of smaller, identical blocks of SRAM whose outputs are then used to form the required wide data bus. Consequently, the use of these similar SRAM blocks does not require the implementation of complex, multi-level sensing hierarchies.

5.2.6 Write Amplifier

Write amplifiers are used to generate the required voltages to flip the state of a memory cell when needed.

For the 1 R/W 6T MC shown in Figure 5.2 and repeated in Figure 5.34, the easiest way the cell can be written is by applying a full-swing differential voltage to the bitlines (BL and BLB) and enabling the cell's access transistors.

Typically, the performance of the write operation in any SRAM is not critical, so write amplifiers are much simpler than sense amps. In addition, if bitlines are assumed to be precharged to a high level, the write amp only needs to discharge one of the bitlines to ground, with the full differential voltage being applied with the help of the precharged high value of the remaining bitline. This concept is demonstrated by the fourth circuit shown in Figure 5.34, where the right NMOS transistor discharges BLB to ground, while the left NMOS is disabled, maintaining the precharged high state of BL.

A complete write amp circuit in shown in Figure 5.35, along with some of the other bitline peripheral circuits. In the figure, the NMOS write-mux transistors connect a single write amplifier to their corresponding

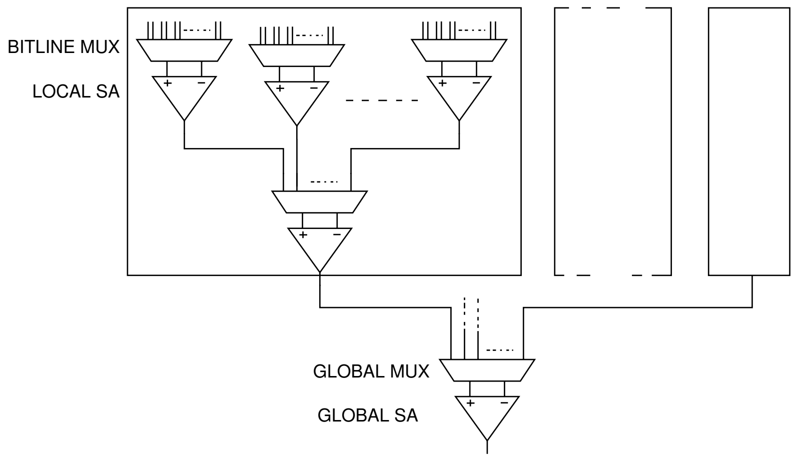


FIGURE 5.33: An example multi-level sensing hierarchy showing three levels of amplification.

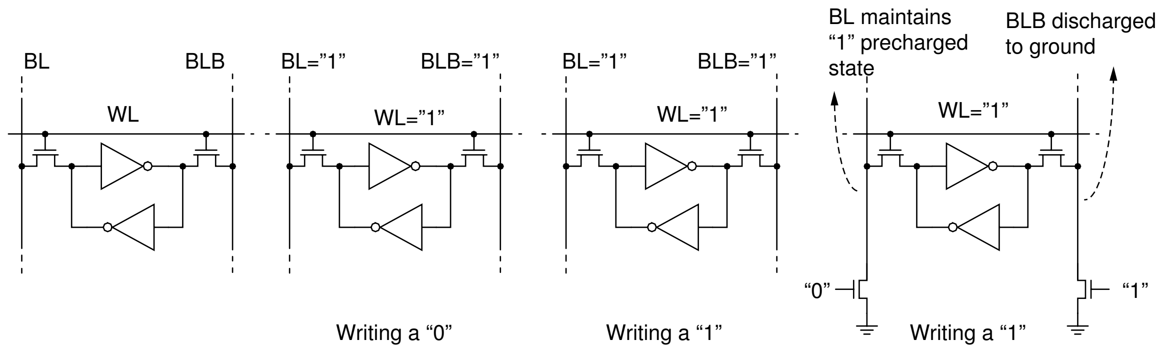


FIGURE 5.34: Writing to a memory cell.

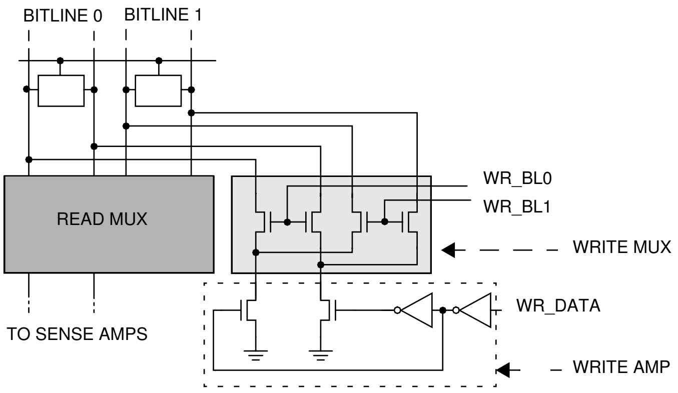


FIGURE 5.35: Bitline peripheral circuits showing a detailed write amplifier circuit.

bitline pairs. Each of these write muxes is enabled only if the column it is connected to is chosen during a write operation.

The write amplifier consists of two NMOS transistors and proper buffering and inversion logic to enable the proper discharging transistor. These transistors, along with the write mux, are sized to ensure that they can discharge the bitlines within the allotted time-window. Typically, this window is dictated by the bitline read time during read operations, so the write transistors need only moderate pulldown discharging capabilities.

Also, since write muxes employing differential writes only need to discharge one of the bitlines when it is assumed that all bitlines are precharged high, only NMOS transistors are needed for the write amp.

Methods for current-mode writes have also been developed, aiming to avoid the significant power required to return fully discharged bitlines back to their precharged states. The main drawback of these methods is that they typically require the modification of the base memory cell to facilitate the current-mode write. For example, an equalizing transistor is often used to put the cell's cross-coupled inverters into a quasi-stable state and is eventually written to its final state by the relatively small current produced by the current-mode amplifier.

5.2.7 SRAM Partitioning

Having discussed many parameters involved in the organization of the SRAM in terms of its decode hierarchy and its peripheral circuits, we now discuss partitioning SRAMs to optimize performance in terms of speed, power, and area.

Among the SRAM parameters that have already been discussed (e.g., row height, number of blocks, number of columns per block, etc.), it is relatively easy to judge the isolated effect of varying a single parameter. Reducing the number of columns of an SRAM, for example, will serve to reduce the word-line capacitance, making the decode process faster. Another example is dividing the SRAM into as many blocks as possible to reduce the amount of unnecessary bitline power dissipation. When properly used, these techniques are effective, but improper application of these techniques without taking their consequences into account will result in a less-than-optimal implementation. In the first of the two previous examples, indiscriminately increasing the number of rows does reduce the wordline load (given constant memory size), but it also increases the bitline capacitance. In the second example, the increase in the number of blocks does lower bitline power dissipation, but it also increases both the decoder power dissipation and the delay.

The key to balancing these requirements is to partition the SRAM in such a way that the positive effects of changing a certain SRAM parameter are not counterbalanced by the negative effects caused by the same parameter.

Amrutur and Horowitz provide a very good discussion of speed and power paritioning for SRAMs [Amrutur and Horowitz 2000]. Amrutur and Horowitz model the speed, power, and area characteristics of an SRAM and solve for the optimal partitioning based on given priorities for speed, power, and area. Figure 5.36 shows the organizational parameters that are used for the optimization, a plot of optimal area versus delay (given no power constraints), and a plot of energy versus delay for a 4-MB, 0.25-μm SRAM. The figure shows a 1024×1024 array partitioned using three organizational parameters: the number of macros (*nm*), the block height (*bh*), and the block width (*bw*). Each macro supplies a subset of the output word (in this case, 16 bits of the 64-bit word), and each macro is divided into sub-blocks, with each sub-block being of size $bh \times bw$. The output of a macro is supplied by a single sub-block selected by the given address. As shown in the figure, the division of a macro into sub-blocks can be done both vertically or horizontally. When divided vertically, a multi-level sensing hierarchy can be used where sub-blocks in the same vertical axis can share a common global bitline and global sense amplfier.

The first graph in Figure 5.36 shows the optimal partitioning (given no power constraints), resulting in minimum area for a given delay, with the sweet spots labeled as points A and B. Amrutur and Horowitz find that the RAM delay is most sensitive to the block height, and small block heights result in the fastest access times.

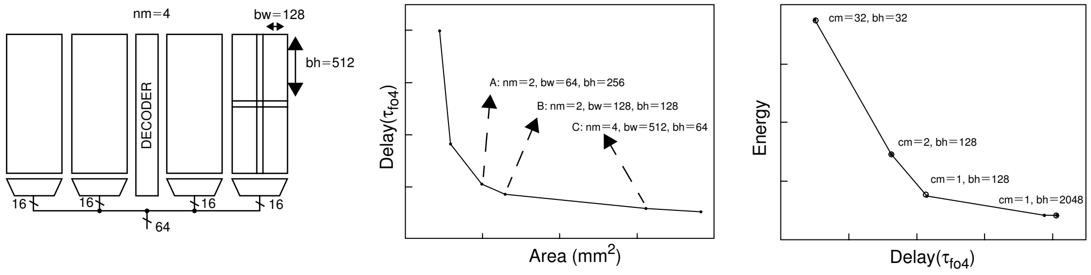


FIGURE 5.36: SRAM partitioning.

The second graph in Figure 5.36 shows the energy and delay relationship with no area constraints. An additional parameter, the amount of column multiplexing (*cm*, or the number of columns sharing a single sense amp), is shown. It shows that minimum delay is achieved first by having a *cm* of 1, where each bit has its own sense amp such that no columns are unnecessarily activated, and second, by having a large block height to allow the muxing to be performed in the bitlines.

5.2.8 SRAM Control and Timing

This subsection describes how the SRAM operation is controlled, both internally and externally. Figure 5.37 shows all of the components of an SRAM necessary to perform a complete read or write access to a specific memory cell. To simplify the figure, only one memory cell in a single column is shown.

It is important for an SRAM operation to have a distinct window of time within which it can perform the desired operation and afterward reinitialize itself to perform the next access. Within this window of time, the SRAM must be allowed to finish the complete sequence of operations before the next access is started. Otherwise, both the present access and the next access will result in corruption of data. This characteristic is unlike combinational logic, whose inputs can be changed at any time while expecting the output to stabilize after a given propagation delay.

With this in mind, in general, there are two ways of starting the SRAM operation. The first is to detect transitions in the address or control inputs. An access is started when these inputs are sensed to have changed. This method is called the address transition detection (ATD) method, and a typical circuit implementation is shown in Figure 5.38.

The pulse generated by the ATD circuit propagates to control the SRAM sequencing of events, including precharging the SRAM to prepare for the next operation.

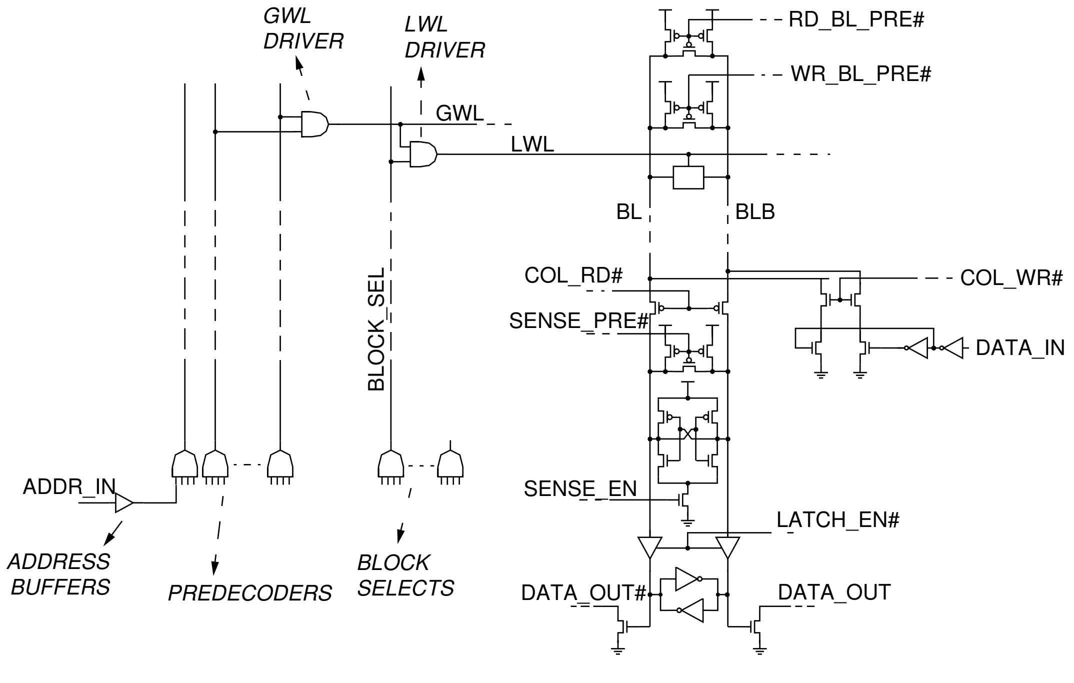


FIGURE 5.37: Important components of an SRAM. Also shown are data and control signals within the circuits.

ATD-controlled SRAMs are classified as asynchronous SRAMs because SRAM operations are started after its input change regardless of the presence of any synchronizing clock in the system.

A second way to start the SRAM operation is to use a synchronizing clock signal (or a signal directly related to the clock). It is important to note that, in this case, only the interface of the SRAM proceeds synchronously with the clock. Significant parts of the internal SRAM operation proceed asynchronously with the clock, triggered using various methods that we will discuss next. In this case, the clock is important only to define the start and/or end points of the SRAM operation, with the designer being careful that the clock period is long enough to allow the SRAM to perform all the required internal operations.

The start pulse that is either generated by the ATD circuit or derived from the system clock or any other external synchronizing signal then triggers parts of the SRAM in turn. This start pulse will serve a different purpose depending on whether the circuits are implemented using dynamic or static logic. For dynamic logic, the initial edge of the start pulse can be used to start the evaluate phase of the dynamic logic. It is important to emphasize that the address inputs of the decoder, especially the dynamic circuits, have to be stable to ensure the correctness of the SRAM operation. Otherwise, output nodes can be unwantedly discharged, with the circuit recovering only after a precharge operation, which will be too late for the present operation. In addition, we may want the start of a dynamic circuit's precharge to be dependent on a control signal different from the signal that triggered the evaluate phase.

Figure 5.39 shows five waveforms demonstrating this concept. The clock and address signals are

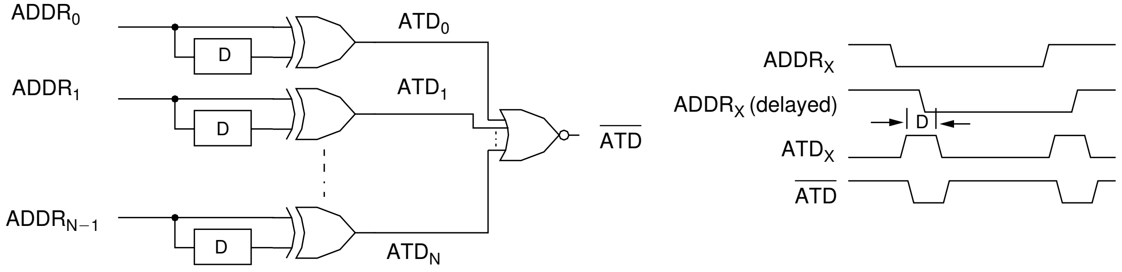


FIGURE 5.38: Address transition detection (ATD).

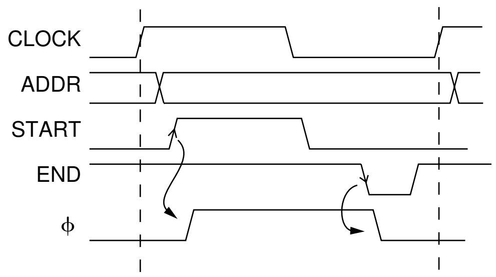


FIGURE 5.39: Generation of the dynamic circuit clock signal (ϕ). Although the START signal that is generated when the address signals are valid can be used to start the evaluate phase, its falling edge is not suitable to start the precharge phase. The END signal is generated internally when it is safe to start precharging, so it is used to trigger the falling edge of ϕ to start the precharge phase.

external inputs to the SRAM, where the address usually changes right after the clock edge (with the delay representing the clock to output delay of their respective storage elements). The third waveform is an internally generated signal that is a delayed version of the clock with enough delay to ensure the stability of the address. Alternatively, this could also be generated using an ATD circuit, with the falling edge occurring much earlier. In any case, only the first edge is of interest. The fourth waveform is an internally generated signal that is used to signify the start of the precharge phase to prepare the SRAM for the next access (later we will discuss how this is generated). To serve as a valid precharge/evaluate signal to our dynamic logic, we need something similar to the fifth waveform, whose rising edge follows the START signal's rise and its falling edge follows the END signal's fall. It is also important to note that even though the START signal by itself may be sufficient for the dynamic logic in the circuits, using the signal as is implies that equal time is allotted to the evaluate and precharge phases, which will be inefficient since the precharge phase is given more time than is necessary.

Figure 5.40 shows a circuit that can generate the clock signal of decoder dynamic circuits. Its main functionality is to produce a pulse whose rising edge is initiated by START's rising edge, while its falling edge is initiated by END's falling edge. It consists of level-to-pulse converters that convert the START and END signals to pulses that drive the inputs of a set-reset latch. A START rising edge generates a pulse that sets the latch, while an END falling edge generates a pulse that resets it. As with all RS latches, we need to insure that its inputs are not asserted simultaneously (causing indeterminate operation). We will see later that this is trivial for SRAM decoders as the START and END signals involved have very large separations relative to the width of the pulses being generated.

Amrutur and Horowitz [2001] state that an optimal decoder hierarchy will have high fan-in inputs only at the initial stage (the predecoder stage) and that all other succeeding blocks of the decoders (e.g., GWL and LWL decoders) should have minimal gate capacitances. The high fan-in requirement for predecoders necessitates the use of dynamic logic for their implementation. Although dynamic logic could also be used for the GWL and LWL decoders, the large number of these circuits will magnify the difficulty associated with dynamic logic.

For static decoders, the use of the START signal to trigger the decoder operation is not as critical as in the dynamic implementation. The reason is that the static logic will always be able to produce the correct output and assert the correct wordline after some delay once its inputs have stabilized. The same is not true with dynamic logic. The main consequence

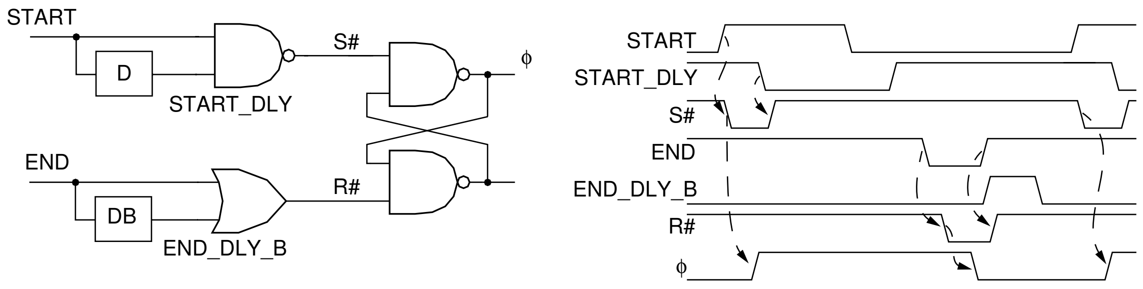


FIGURE 5.40: Generation of a signal triggered by two separate events. In this case, the rising edge of ϕ is triggered by the rising edge of START, while the ϕ falling edge is triggered by the falling edge of END.

of allowing the decoder to proceed without the start signal is the creation of glitches in some of the wordlines that are non-critical, but nevertheless will increase the power dissipation during bitline precharge.

As mentioned earlier, it is beneficial to use a PWL scheme, where the wordline is pulsed only for the minimum amount of time needed to develop a sufficient bitline voltage differential that can be used by the sense amp. Limiting the differential reduces the amount of power required to precharge the bitlines to their original state, and allows some of the precharge operations to proceed sooner and in parallel with the sensing operation, possibly resulting in a decrease to the total cycle time of the SRAM.

PWL schemes can be implemented using either open-loop or closed-loop schemes. In an open-loop scheme, the wordlines are turned off after a certain time delay determined at design time. This is often done with the use of gate delays that produce delayed outputs on parts of the decoder hierarchy which, when they catch up with the original signals, then serve to turn the signals off, producing a pulse whose width is determined by the gate delay, as shown in Figure 5.41. The total gate delay (and the corresponding pulse width) is designed to be long enough such that the memory cells have enough time to discharge the bitlines and develop sufficient differential.

Although this traditional way of implementing a PWL scheme performs sufficiently well, the effects of technology scaling has made it difficult to maintain the required correlation between the gate delays and the time required to discharge capacitive bitlines because of the varying extent to which scaling affects the memory cells and the decoder drivers. This is exacerbated by PVT variations, which affect the circuits' characteristics differently, worsening the correspondence. This problem is due to the bitline delay being dependent on the large bitline capacitance discharged by a memory cell's (often) minimum-sized pulldown transistor, which is affected more significantly by PVT variations compared to delay of non-minimum sized gates.

These effects discourage the use of open-loop techniques, necessitating the use of feedback from structures that more closely follow the run-time behavior of the circuit. Different structures have been proposed that use this kind of feedback [Amrutur and Horowitz 1998, Nambu et al. 1998, Osada et al. 2001]. A good example of this scheme

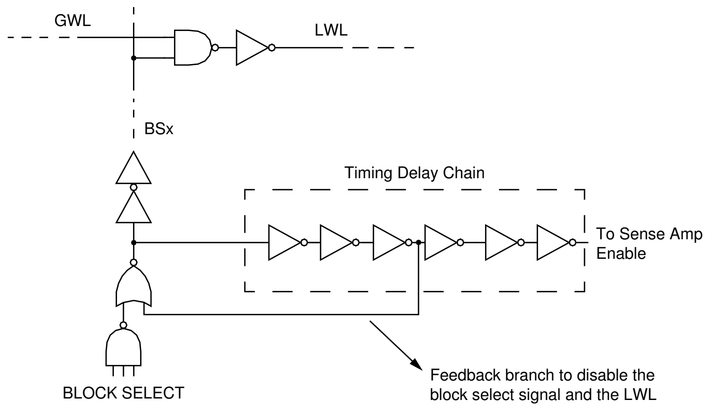

GWL
LWL
BSx
Timing Delay Chain
To Sense Amp Enable
Feedback branch to disable the block select signal and the LWL
BLOCK SELECT

FIGURE 5.41: PWL using an open-loop gate delay.

is the replica technique by Amrutur and Horowitz [1998], as shown in Figure 5.42.

Figures 5.42(a–c) show a feedback scheme based on capacitance ratioing. Figure 5.42(a) shows the addition of a replica column to the standard memory array. The replica memory cell is hardwired to store a zero such that it will discharge the replica bitline once it is accessed. Because of its similarity with the actual memory cells (in terms of design and fabrication), the delay of the replica bitline tracks the delays of the real bitlines very well and can be made roughly equal by varying (at design time) the number of cells connected to the replica bitline. With this method of generating a delay that is equal to the bitline delay even with significant PVT variations, the problem is shifted to equalizing the delays between two chains of gates, as shown in 5.42(c), which is a much easier problem.

Alternatively, feedback based on a current ratioing method can be used instead. The main advantage of this technique over the first method is the ability to generate local resets for each row, which can be used to directly turn off the LWL drivers. This makes the delay balancing easier to do and enables the use of skewed gates even for the LWL decoders, speeding up the decoder delay.

To accomplish this, a replica row and column are added to the regular memory block. In this method,

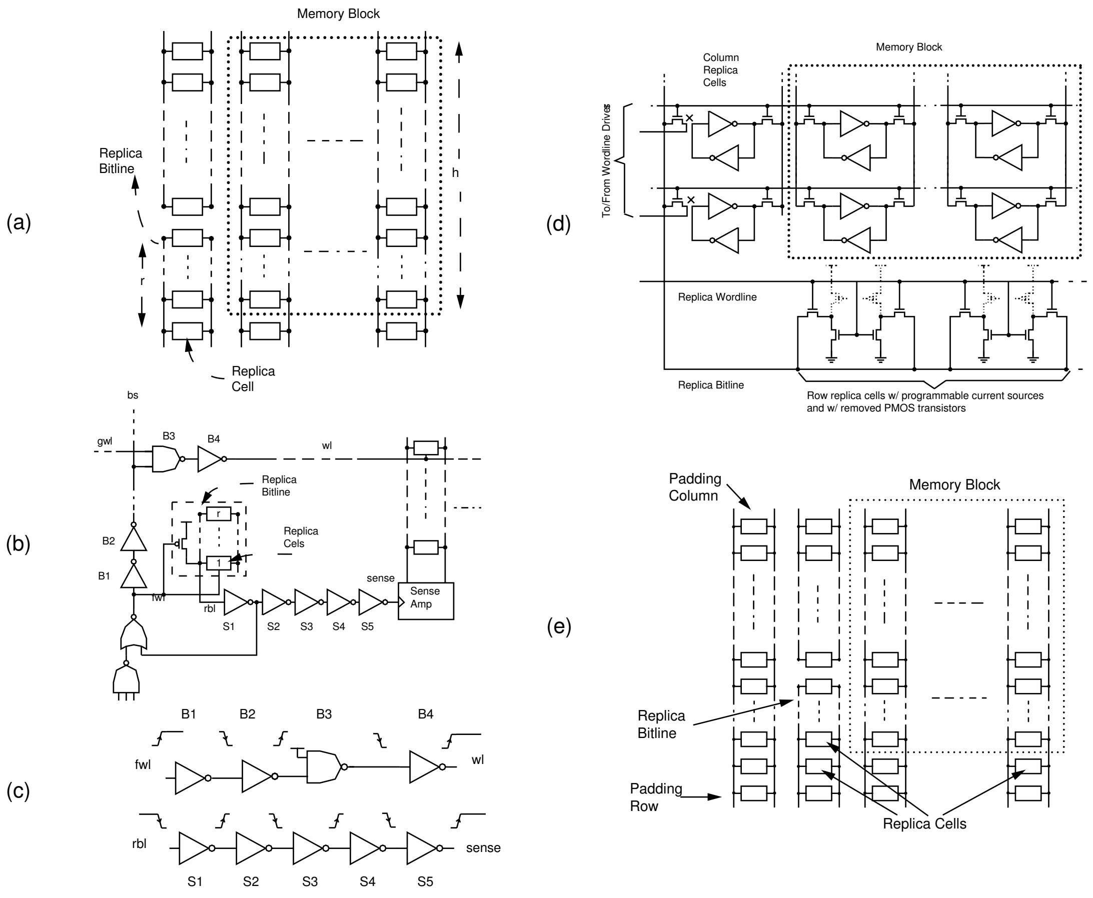


FIGURE 5.42: Replica technique.

it is the replica row that performs the bitline discharge instead of the replica column. The memory cells within the replica rows become programmable current sources and have their unnecessary PMOS transistors disabled or totally removed. They are configured such that they discharge the replica bitline whenever the replica LWL driver asserts (which is every cycle). Like the previous method, different numbers of current sources can be configured to activate to equalize the delays as desired. The signal at the replica bitline then propagates up the replica column (whose memory cells are used only as pass transistors) and propagates to the single LWL driver that is active. This signal can then be used to quickly deactivate the LWL driver and stop the memory cells from further discharging the bitlines.

Figure 5.42(d) shows how this scheme can be integrated within the entire scheme, while Figure 5.42(e) shows the addition of padding rows to minimize the PVT variations between the regular memory cells and the replica cells caused by the replica being in the outer edge of the array.

Whatever scheme is used (and, for that matter, whatever control circuitry), it is important to emphasize that the basic concept of these schemes is the generation of a control signal that can reliably track the development of the bitline voltage differential and, hence, can be used to control the proper activation of the sense amps for correct operation and the deactivation of the decoder to generate a PWL for lower power consumption.

The correct generation of its activation signal allows the sense amplifier to function correctly and speeds up the sensing of the voltage differential across the bitline. For a single-level sensing hierarchy, the output of the sense amp is typically a full-voltage swing output. It is typically desired to latch this output so that its value is retained until the end of the cycle (and some time after). Without a latch, the value is lost when the sense amp is precharged.

The data is gated on to the latch only after the sense output has stabilized to prevent a glitching in the output data (which may or may not be critical depending on the downstream circuits). Sophisticated feedback information like the replica technique used for bitline control is unnecessary to control this process since sense amp characterization is easier and more tolerant of PVT variations compared to the minimum-sized pulldowns of the memory cells. Consequently, simple gate delays can be used to generate a delayed version of the sense-enable signals to gate the sense amp data to the output latches.

To prepare the SRAM for the next access, the dynamic nodes within the SRAM circuits have to be precharged back to their original values. Referring back to Figures 5.40 and 5.39, the END signal can be derived from the signal present at the replica bitline. This END signal then deasserts ϕ, starting the precharge phase. This ensures that the address decoder precharge is started only after it has performed its funtion and the selected bitlines have developed sufficient differential.

The END signal can also be used to directly derive the precharge signals of the bitline (including the replica) and the sense amplifier. Precharging of the bitlines (and the assumed early reset of the LWL drivers) also acts to precharge the replica bitline back to its high initial state. The complete process is summarized in a timing diagram shown in Figure 5.43.

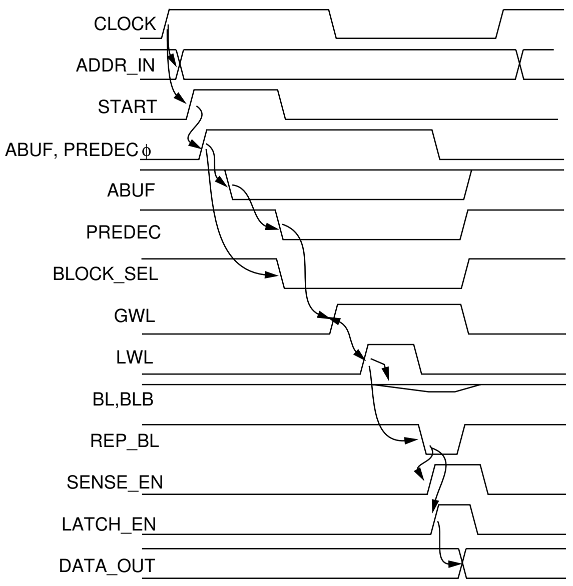


FIGURE 5.43: Timing diagram showing a complete sequence of an SRAM read operation.

The write operation shares much of the control sequence used for the read operation, especially the decoder and the replica technique. The main difference, aside from the obvious enabling of different sets of muxes and not enabling the sense amp and data latches, is that the data to be written has to be applied to the bitlines much earlier in the access, even before the LWL has asserted. This ensures that the moment the access transistors of selected cells are enabled, the full-swing differential is available to flip the cross-coupled inverter of the memory cell (if necessary). This requires the SRAM to receive write data early in the access. For most pipelines that buffer the write data, this requirement is not a problem because the data is easily available from the write buffer at the early part of the clock cycle. In situations where the data is not available early (like in cases where it is computed in the same cycle), care must be taken to delay the start of the SRAM write access until it is ensured that the data will be available at the right time.

It must also be ensured that memory cells are properly written during the short time that the wordlines are on. The width of the wordline pulse will be the same as in the read access, since the behavior of the replica bitline will be the same for both types of accesses.

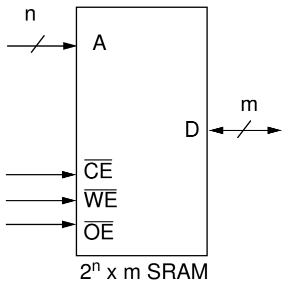
Finally, the discharge of the replica bitline eventually triggers the precharge of the bitlines. In the case of systems with separate write precharge devices (as shown in Figure 5.43), both precharge devices are enabled to help the fully discharged bitline recover faster.

5.2.9 SRAM Interface

SRAMs can be used either synchronously or asynchronously. A typical discrete SRAM asynchronous interface is shown in Figure 5.44.

Aside from the address and bidirectional data bus, there are typically three other types of control signal: the chip enable (CE#), the write enable (WE#), and the output enable (OE#). The OE# signal serves to enable the 3-state output buffers of the data bus during read operations, while the WE# signal is used to signify write operations. The CE# signal is used for both operations to enable access to the SRAM. Typically, the entire address bus and the control signals are used by the internal ATD circuits to start an access. Figure 5.45 shows read and write timing diagrams for an SRAM with an asynchronous interface.

In a lot of instances, very fast cycle times for SRAMs are paramount (e.g., caches). In many cases, a further decrease in SRAM cycle times is difficult (i.e., no faster part exists for discrete applications, or techniques to reduce cycle time are currently in use and are too expensive to extend, in the case of embedded or integrated SRAM). In these cases, a straight forward technique to reduce the system cycle time is to use a technique called pipelined-burst access, and a typical interface is shown in Figure 5.46. The main concept behind this technique is the use of wrapper circuits that communicate with the external circuits at a fast frequency (small cycle time), but at the same time communicate with the SRAM at the memory's slower speed. This technique is shown in Figure 5.47.



$\overline{CE}$	$\overline{WE}$	$\overline{OE}$	TYPE OF ACCESS
0	0	0	SRAM WRITE
0	1	0	SRAM READ
1	X	X	SRAM DISABLED

FIGURE 5.44: Discrete SRAM asynchronous interface.

Internally, the SRAM word width is made wider by an amount PB_{num} equal to the desired burst count (and corresponding decrease in cycle time). Figure 5.47 shows an SRAM with a burst count of four, so an SRAM communicating in a 32-bit-wide system has to be 128-bits wide internally. In effect, the serial-to-parallel interface (SPI) used for the writes collects PB_{num} words (at the faster system speed) before writing it out once to the SRAM (at the native SRAM speed). Likewise, the parallel-to-serial interface (PSI) reads a wide word at the native SRAM speed and then breaks it up into PB_{num} chunks before sending it out at the faster system speed.

Although this technique enables the use of an SRAM in a much faster environment, latency is sacrificed to obtain the speed gain. In the case of the previous example, the read data initiated on cycle N–1 only becomes avaiable at cycle $N+PB_{num}$.

5.3 Advanced SRAM Topics

5.3.1 Low-Leakage Operation

When CMOS circuits were first used, one of their main advantages was the negligible leakage current flowing when the gate was at DC or steady state. Practically all of the power dissipated by CMOS

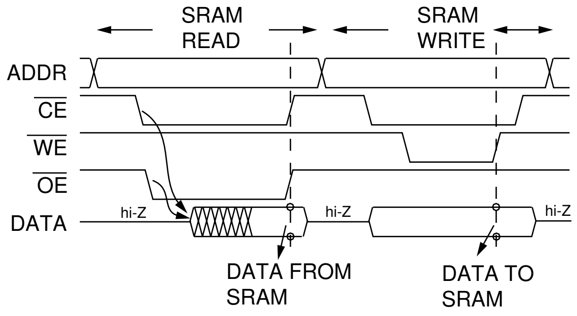


FIGURE 5.45: SRAM asynchronous read and write waveforms.

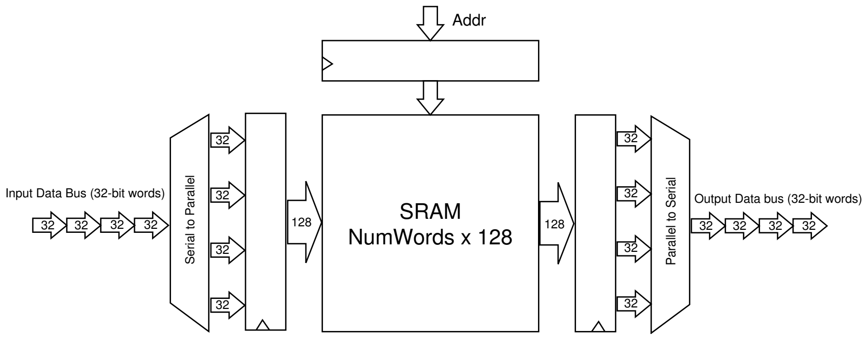


FIGURE 5.46: An example synchronous pipelined-burst SRAM interface showing a burst write and burst a read operation. In this example, the data I/O bus operates at 4x the speed of the internal SRAM core.

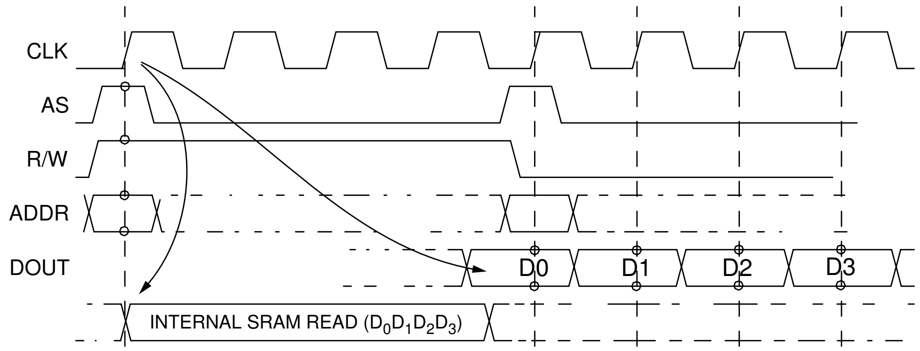


FIGURE 5.47: SRAM pipelined-burst read access.

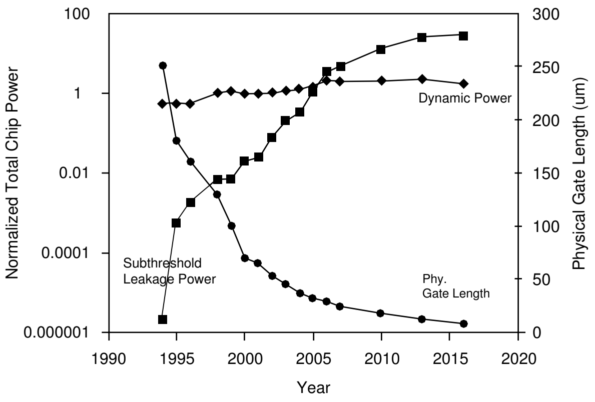


FIGURE 5.48: Projections for dynamic and leakage, along with gate length. (Used with permission from Kim [2004a]).

gates was due to dynamic power consumed during the transition of the gate. But as transistors become increasingly small, the CMOS leakage current starts to become significant and is actually projected to be larger than the dynamic power, as shown in Figure 5.48.

Aside from using techniques to reduce dynamic power (e.g., PWL, sub-blocking), SRAM designs targeted for low power also have to start accounting for the increasingly large amount of power dissipated by leakage currents. In this section, we present a collection of various solutions that have been proposed to lessen the adverse effects of leakage current.

Multi-Vt Memory Cells

The main cause of leakage power is due to subthreshold leakage current (although leakage due to DIBL short-channel effects, as well as gate leakage, is also starting to become significant). This is due to the fact that when one scales transistors to increase performances one must also lower the transistor threshold voltage, and as Vt decreases, subthreshold

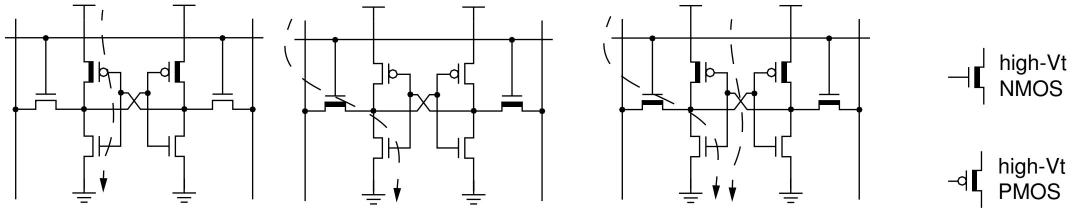


FIGURE 5.49: Different multi-Vt configurations for the 6T memory cell showing which leakage currents are reduced for each configuration.

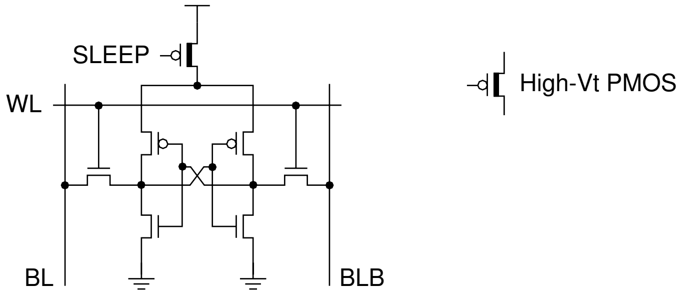


FIGURE 5.50: Gated-Vdd technique using a high-Vt transistor to gate Vdd.

leakage (which has an exponential dependence on the threshold) increases.

One of the conceptually simple solutions to the problem is the use of multi-threshold CMOS circuits. This solution involves using process-level techniques to increase the threshold voltage of transistors to reduce the leakage current. Increasing this threshold serves to reduce the gate overdrive and reduces the gate's drive strengths resulting in increased delay. Because of this, the technique is mostly used on the non-critical paths of the logic, and fast, low-Vt transistors are used for the critical paths. In this way, the delay penalty involved in using higher Vt transistors can be hidden in the non-critical paths while reducing the leakage currents drastically.

In this scheme, multi-Vt transistors are selectively used for the memory cells since they represent a majority of the circuit, reaping the most benefit in leakage power consumption with a minor penalty in the access time. Different multi-Vt configurations are shown in Figure 5.49, along with the leakage current path that each configuration is designed to minimize.

Gated-Vdd Technique

Another conceptually simple way to reduce the leakage current is to gate the power supply of the SRAM with a series transistor as shown in Figure 5.50. This is called the Gated-Vdd technique [Powell et al. 2000]. With the stacking effect introduced by this transistor, the leakage current is reduced drastically. This technique benefits from having both low-leakage current and a simpler fabrication process requirement since only a single threshold voltage is conceptually required (although the gating transistor can also have a high threshold to decrease the leakage even further at the expense of process complexity).

Without any special tricks, the parts of the memory that have their power supply gated will lose their state whenever the gating transistor is turned off. When applied to caches, this technique can be advantageous if the current working set is smaller than the cache size such that parts of the cache can be turned off without significantly adverse effects to performance. When the disabled part of the cache is accessed, the access misses, the addressed part of the cache is turned on, and the desired data can be retrieved from the lower level of the memory hierarchy.

Gated-Ground Technique

As an alternative to gating the power supply and corrupting the stored state in the memory, a technique called Data Retention Gated-ground (DRG) [Agarwal et al. 2003] can be used, where memory cells use a virtual ground that is connected to the actual ground through a gating transistor. This has the same effect on leakage current as power-supply gating. The main advantage of this technique is that, with proper sizing, the cells are able to retain their state even without being directly connected to ground. This technique is shown in Figure 5.51.

Aside from the reduced leakage current, this technique has the advantage that no other circuitry besides the gating transistor is required in the implementation. No extra control circuitry is necessary, as the gating transistor can be controlled with the wordline supplied to the row.

In cache applications, this technique has no impact on the cache hit rate since the data stored within the cache is retained even when the gating transistors are turned off. In addition, having the control use existing circuitry means that the cache controller isn't burdened with additional complexity to keep track of which parts of the cache are disabled.

The main disadvantage of the DRG technique is its reduced tolerance to noise, as evidenced by its smaller SNM margin compared to a conventional implementation, as shown in Figure 5.52. The technique also has a small negative effect on both the delay and the area of the circuit because of the introduction of the gating transistor. An additional minor disadvantage is the extra complexity in design required to carefully size the transistor within the memory cell to ensure that the data is actually retained even when the gating transistor is disabled.

Drowsy SRAMs

One last technique we present that reduces leakage power in SRAMs is the Drowsy technique [Kim et al. 2004]. This technique has similarities to both the gated-Vdd and the DRG techniques discussed previously in that it uses a transistor to conditionally enable the power supply to a given part of the

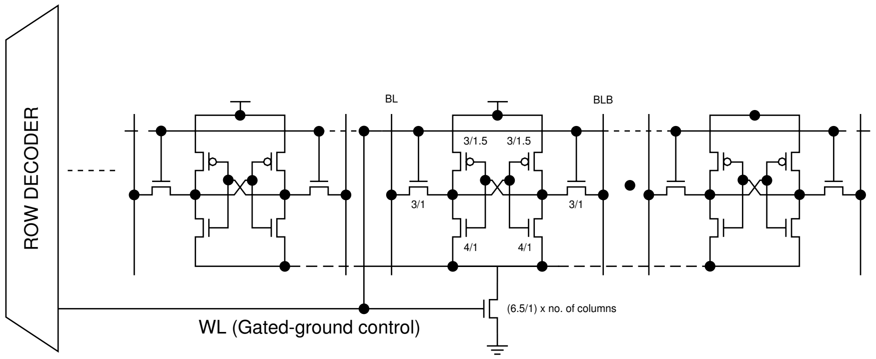


FIGURE 5.51: An SRAM row using the DRG technique.

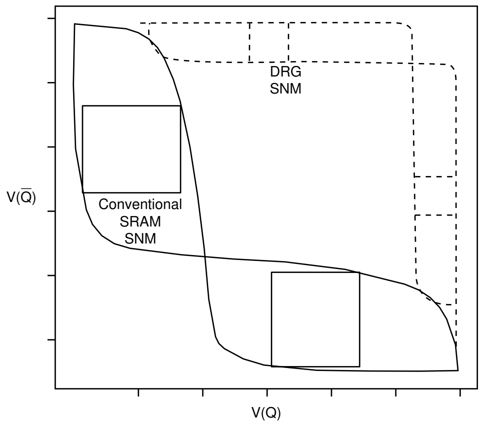


FIGURE 5.52: SNM of the DRG SRAM compared to a conventional SRAM.

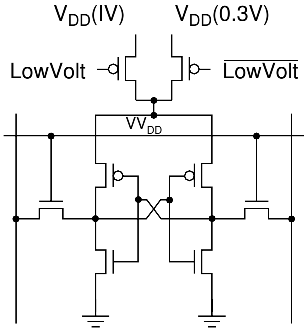


FIGURE 5.53: A drowsy SRAM cell containing the transistors that gate the desired power supply.

SRAM. This method reduces leakage power by putting infrequently accessed parts of the SRAM into a state-preserving, low-power drowsy mode.

This technique uses another power supply with a lower voltage than the regular supply to provide power to memory cells in the drowsy mode. Leakage power is effectively reduced because of its dependence on the value of the power supply. This technique is shown in Figure 5.53. The configuration is more tolerant to noise compared to the DRG technique because the cross-coupled inverters within the memory cells are still active and driving their respective storage nodes, which is not exactly true for the DRG technique.

The control heuristic is straightforward: periodically, all blocks in the cache are made drowsy, and a block is only brought out of the drowsy mode if it is actually referenced

5.4 Cache Implementation

5.4.1 Simple Caches

After discussing the implementation of SRAM blocks, we could now use these memory primitives as building blocks for constructing caches. As shown in Figure 5.54, the tag and data part of the cache can be made up of appropriately sized memory arrays. Adding in the complete support circuitry to simple SRAM storage elements results in a complete, functional cache subsystem.

For an N-way associative cache, we use N tag data pairs (note that these are logical pairs and that they are not necessarily implemented in the same memory array), an N-way comparator, and an N-way multiplexer to determine the proper data and to select it appropriately. For systems that do not employ some form of virtual addressing, the TLB is optional, but otherwise it is needed, especially if physical addresses are stored in the tag area. Also, although the TLB access here is shown to be performed in parallel with the cache tag and data access, it can be performed at any time as long as the translation is available in time for use by the comparator.

In Figure 5.54, the cache controller has the responsibility of keeping track of cache operations and accesses to implement additional cache specifications like write-back/write-through behavior. It is also responsible for facilitating other functions like multiporting through banking to control each access and keep track of bank collisions. Finally, the cache controller is responsible for interfacing to the lower level of memory when the access misses in order to fill the cache. Although caches can be implemented

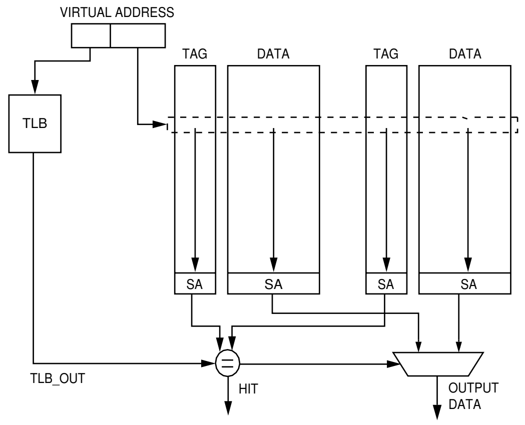


FIGURE 5.54: Cache block diagram.

in a myriad of different ways, this simple cache implementation serves as a usable, functional cache design.

5.4.2 Processor Interfacing

Figure 5.55 shows two typical ways of interfacing a microprocessor to a data cache. Figure 5.55(a) shows where the cache could connect to an in-order processor pipeline. It shows the address, control, and data being supplied to the cache stage by pipeline registers, and cache access is assumed to fit in one processor cycle, providing result signals (like HIT) and the data, in case of a load. The result signals are then used by the pipeline control circuitry to decide whether to stall the pipe depending on whether the access is a hit or a miss.

Alternatively, Figure 5.55(b) shows where and how the cache could fit in an out-of-order execution pipeline. The figure shows the internals of the load/store unit, and with the assumption that the proper structures exist outside this unit that allow for non-dependent instructions to execute out of order, this unit operates independently of the other functional units, and cache misses do not need to stall the processor core (unless during extreme cases where internal data structures within the load/store unit have filled up and cannot accept new instructions).

The load/store unit typically includes a load/store queue used as a holding tank for memory instructions. When a load or store is cleared to access the cache (i.e., it has all its needed operands and performing the access will not cause any memory hazards or inconsistencies), information is retrieved from the load/store instruction and used to access the cache. Processor structures are updated with the results of the access. In the case of a miss, the load/store instruction is made to wait and retry the access, and in some implementations, the instruction is transferred to another structure inside the load/store unit that holds accesses that missed.

5.4.3 Multiporting

Caches can be multiported using different methods. True multiported caches employ multiported SRAMs that are specially designed to allow concurrent accesses to any location. Two examples of true multiported memory cells are shown in Figure 5.56.

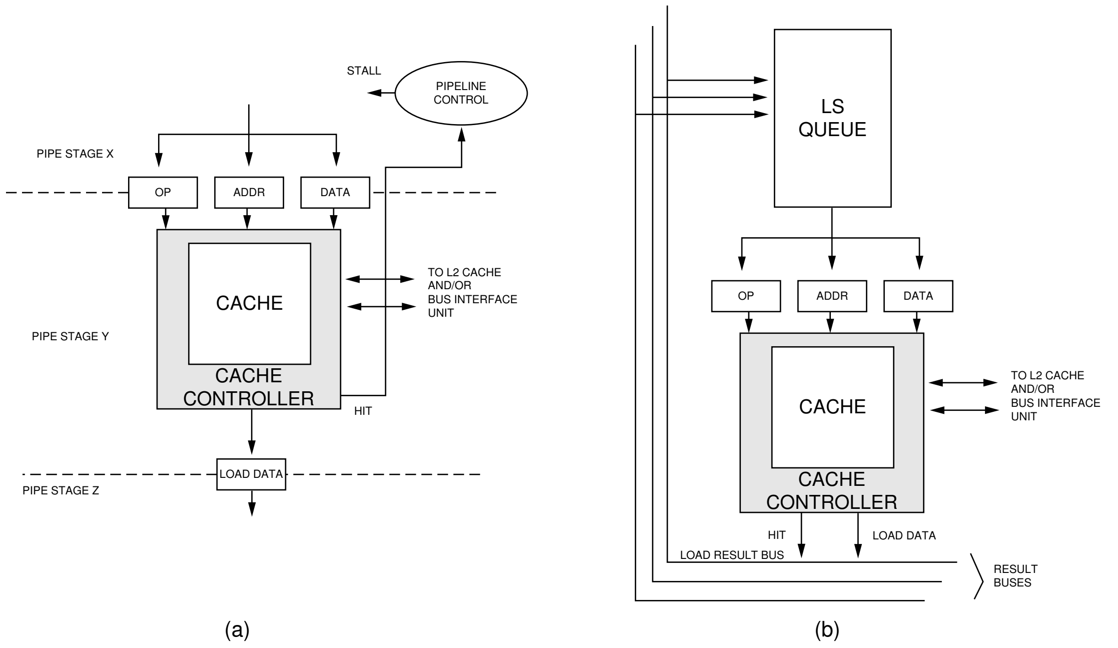


FIGURE 5.55: Cache-processor interface for an (a) in-order and (b) an out-of-order processor.

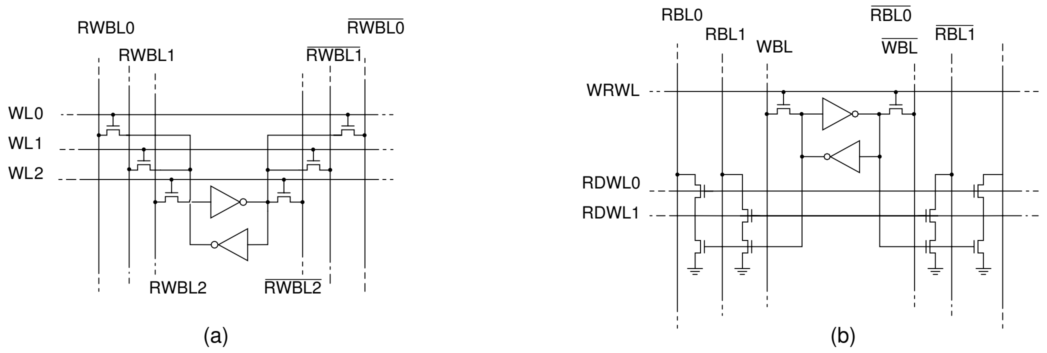


FIGURE 5.56: Multiported memory cells: (a) 3 R/W ports, (b) 2R 1 R/W port.

Although using true multiporting allows relatively simple control of the SRAM, the addition of multiple access transistors for each port results in a very significant increase in memory area. Aside from the area increase, this also adversely affects the delay because of wire length increase within the memory.

An alternative to true multiporting is the use of multiple independent banks for the cache, where each bank is implemented as a simple single-ported

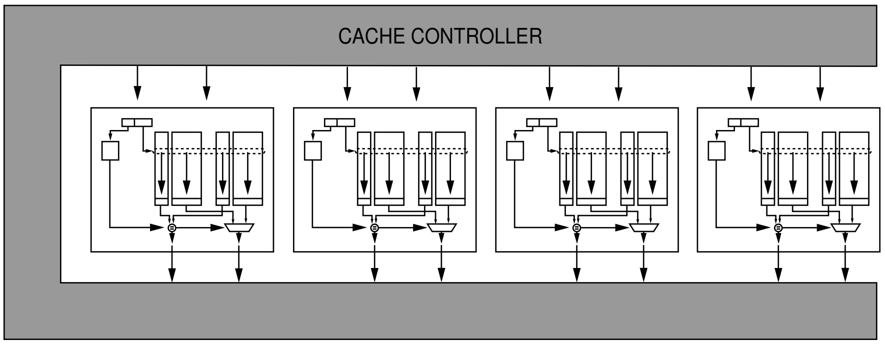


FIGURE 5.57: Multiported cache using banking. In this case, the cache is implemented as four banks of identical two-way associative, 1 R/W ported cache.

cache. This configuration can satisfy multiple cache accesses as long as the accesses are to different banks. This configuration is shown in Figure 5.57 for a two-way associative cache with four banks.

This configuration can perform a maximum of four R/W accesses concurrently. The main consequence of this configuration is the additional complexity and intelligence required in the cache controller to manage the control and I/O of each individual bank, along with keeping track of bank conflicts that may arise. Even with this complication, the benefits of true multiporting often do not justify the drastic increase in cache size, making the banking approach more popular.