Array Subsystems

9

9.1 Introduction

Memory arrays often account for the majority of transistors in a CMOS system-on-chip. Arrays may be divided into categories as shown in Figure 9.1. *Programmable Logic Arrays* (PLAs) perform logic rather than storage functions, but are also discussed in this chapter.

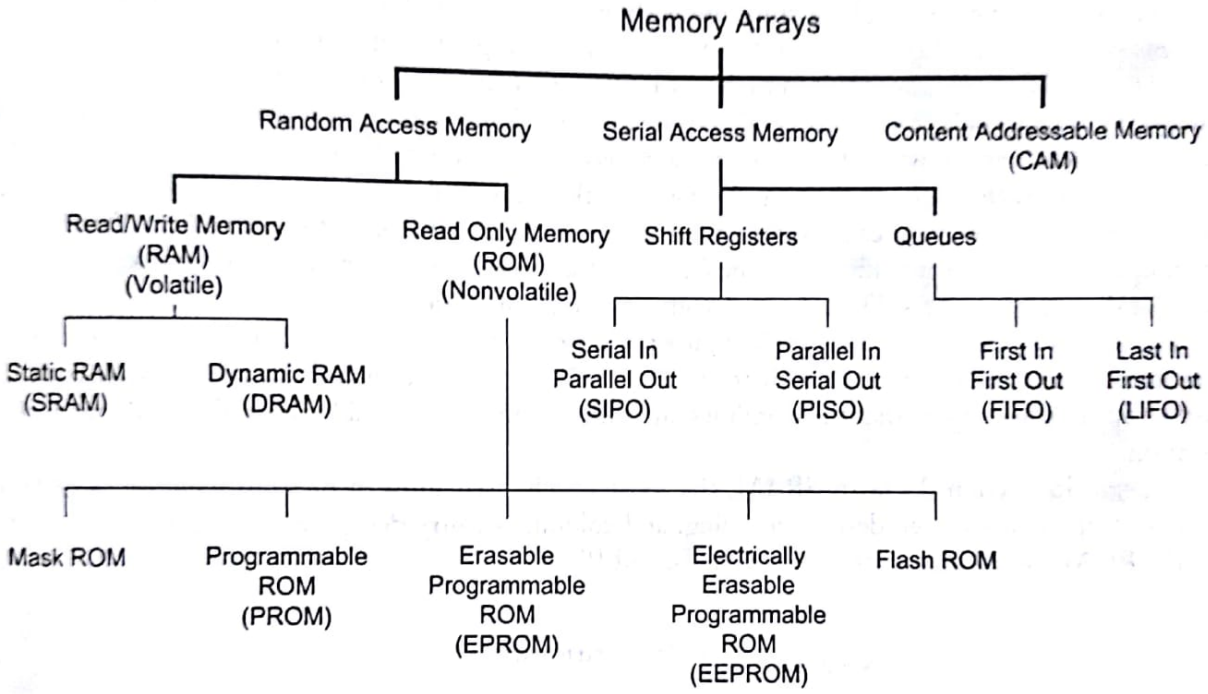


FIG 9.1 Categories of memory arrays

Random access memory is accessed with an *address* and has a latency independent of the address. In contrast, *serial access memories* are accessed sequentially so no address is necessary. *Content addressable memories* determine which address(es) contain data that matches a specified *key*.

Random access memory is commonly classified as *read-only memory* (ROM) or *read/write memory* (confusingly called RAM). Even the term ROM is misleading because many ROMs can be written as well. A more useful classification is *volatile* vs. *nonvolatile* memory. Volatile memory retains its data as long as power is applied, while nonvolatile memory will hold data indefinitely. RAM is synonymous with volatile memory, while ROM is synonymous with nonvolatile memory.

Like sequencing elements, the memory cells used in volatile memories can further be divided into *static* structures and *dynamic* structures. Static cells use some form of feedback to maintain their state, while dynamic cells use charge stored on a floating capacitor through an access transistor. Charge will leak away

through the access transistor even while the transistor is OFF, so dynamic cells must be periodically read and rewritten to refresh their state. Static RAMs (SRAMs) are faster and less troublesome, but require more area per bit than their dynamic counterparts (DRAMs).

Some nonvolatile memories are indeed read-only. The contents of a mask ROM are hardwired during fabrication and cannot be changed. But many nonvolatile memories can be written, albeit more slowly than their volatile counterparts. A *programmable* ROM (PROM) can be programmed once after fabrication by blowing on-chip fuses with a special high programming voltage. An *erasable programmable* ROM (EPROM) is programmed by storing charge on a floating gate. It can be erased by exposure to ultraviolet (UV) light for several minutes to knock the charge off the gate. Then the EPROM can be reprogrammed. *Electrically erasable programmable* ROMs (EEPROMs) are similar, but can be erased in microseconds with on-chip circuitry. *Flash* memories are a variant of EEPROM that erases entire blocks rather than individual bits. Sharing the erase circuitry across larger blocks reduces the area per bit. Because of their good density and easy in-system reprogrammability, Flash memories have replaced other nonvolatile memories in most modern CMOS systems.

Memory cells can have one or more *ports* for access. On a read/write memory, each port can be read-only, write-only, or capable of both read and write.

A typical small memory array architecture is shown in Figure 9.2. Central to the design is a memory array consisting of 2^n *words* of storage of 2^m bits each. In the simplest design, the array is organized with one row per word and one column per bit in each word. Often there are far more words in the memory than bits in each word, which would lead to a very tall, skinny memory that is hard to fit in the chip floorplan and slow because of the long vertical wires. Therefore, the array is often folded into fewer rows of more columns. After folding, each row of the memory contains 2^k words, so the array is physically organized as 2^{n-k} *rows* of 2^{m+k} *columns* or bits. The row decoder activates one of the rows by asserting one of the *wordlines*. During a read operation, the cells on this wordline drive the *bitlines*, which may have been conditioned to a known value in advance of the memory access. The column decoder controls a multiplexer in the column circuitry to select 2^m bits from the row as the data to access. The figure illustrates an 8-word by 4-bit 2-way multiplexed memory folded into a 4-row by 8-column array with $n = 3$, $m = 2$, $k = 1$. Larger memories are generally built from multiple smaller subarrays so that the wordlines and bitlines remain reasonably short, fast, and low in power dissipation.

We begin in Section 9.2 with SRAM, the most widely used form of on-chip memory. SRAM also illustrates all the issues of cell design, decoding, and column circuitry design. Subsequent sections address DRAMs, ROMs, serial access memories, CAMs, and PLAs.

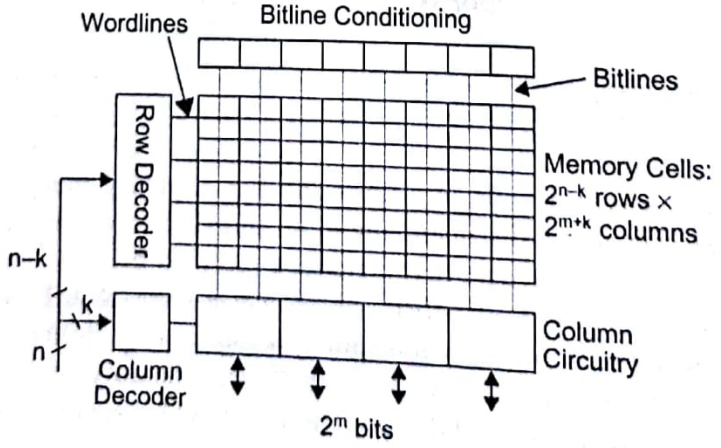


FIG 9.2 General memory array architecture

9.2 SRAM

The fundamental building block of a static RAM is the SRAM memory cell. The cell is activated by raising the wordline and is read or written through the bitline. Figure 9.3(a) shows a 12-transistor SRAM cell built from a simple static latch and tristate inverter. The cell has a single bitline. True and complementary read and write signals are used in place of a single wordline. A representative layout in Figure 9.3(b) has an area of 46 × 75 λ. The power and ground lines can be shared between mirrored adjacent cells, but the area is still limited by the wires and is undesirably large. However, the cell is easy to design because all nodes swing rail-to-rail and it is fast when used in small RAMs and register files.

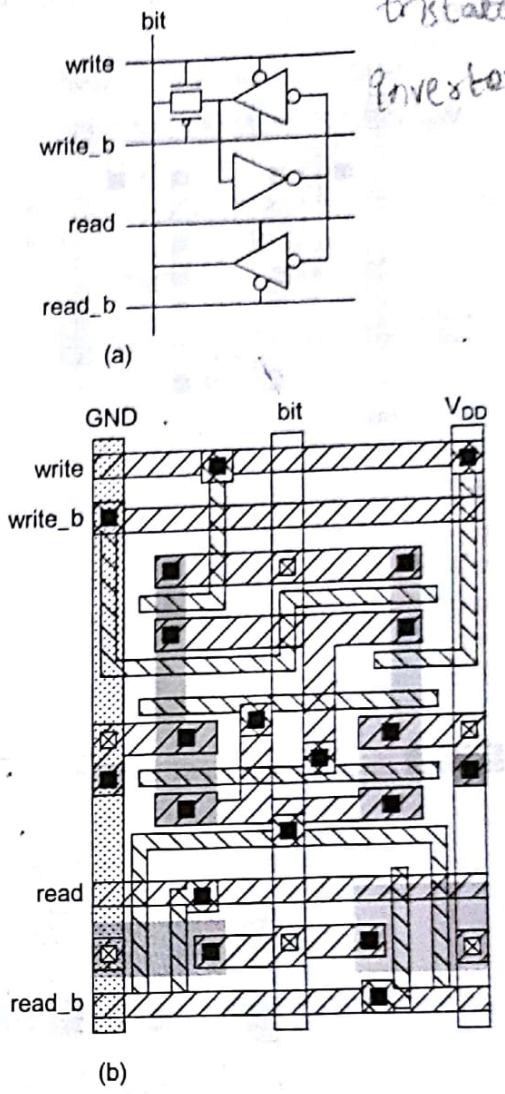


FIG 9.3 12-transistor SRAM cell

Figure 9.4 shows a 6-transistor (6T) SRAM commonly used in practice. Such a cell uses a single wordline and both true and complementary bitlines. The complementary bitline is often called bit_b or bit. The cell contains a pair of cross-coupled inverters and an *access transistor* for each bitline. True and complementary versions of the data are stored on the cross-coupled inverters. If the data is disturbed slightly, positive feedback around the loop will restore it to V_{DD} or GND. The wordline is asserted to read or write the cell. The nMOS access transistors are best at passing '0's. For reads, the bitlines are initially precharged high and one is pulled down by the SRAM cell through the access transistor. For writes, the bitline or its complement is actively driven low and this low value overpowers the cell to write the new value. Careful choice of transistor sizes is necessary for correct operation, as will be examined in Section 9.2.1. The 6T cell achieves its compactness at the expense of more complex peripheral circuitry for reading and writing the cells. This is a good tradeoff in large RAM arrays where the cell size dominates the area. The small cell size also offers shorter wires and hence lower power consumption.

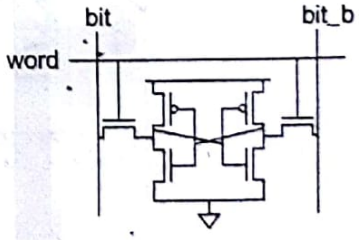


FIG 9.4 6-transistor SRAM cell

SRAM cells require clever layout to achieve good density. Figure 9.5(a) shows a stick diagram of a typical design. The cell is designed to be mirrored and overlapped to share V_{DD} and GND lines between adjacent cells along the cell boundary, as shown in Figure 9.5(b). Note how a single diffusion contact to the bitline is shared between a pair of cells. This halves the diffusion capacitance, and hence reduces the delay discharging the bitline during a read access. The wordline is run in both metal1 and polysilicon; the two layers must occasionally be strapped (e.g., every four or eight cells). Sample layouts derived from the stick diagram are shown in Figure 9.6. Figure 9.6(a) shows a conservative cell of 26 × 45 λ, obeying the MOSIS submicron design rules (see also inside front cover). In this layout, the metal1 and polysilicon wordlines are contacted in each cell. The substrate and well are also contacted in each cell. Figure 9.6(b) shows a scanning electron micrograph of the polysilicon and diffusion layers in a more aggressive (20 × 22 λ) cell in the LSI Logic 130 nm process [Kong01w]. SRAM is so important that design rules are very carefully studied and bent where possible to

minimize cell area in commercial processes. Notice that diagonal lines are also used. Moreover, each substrate contact can be shared among multiple cells to save area at the expense of regularity.

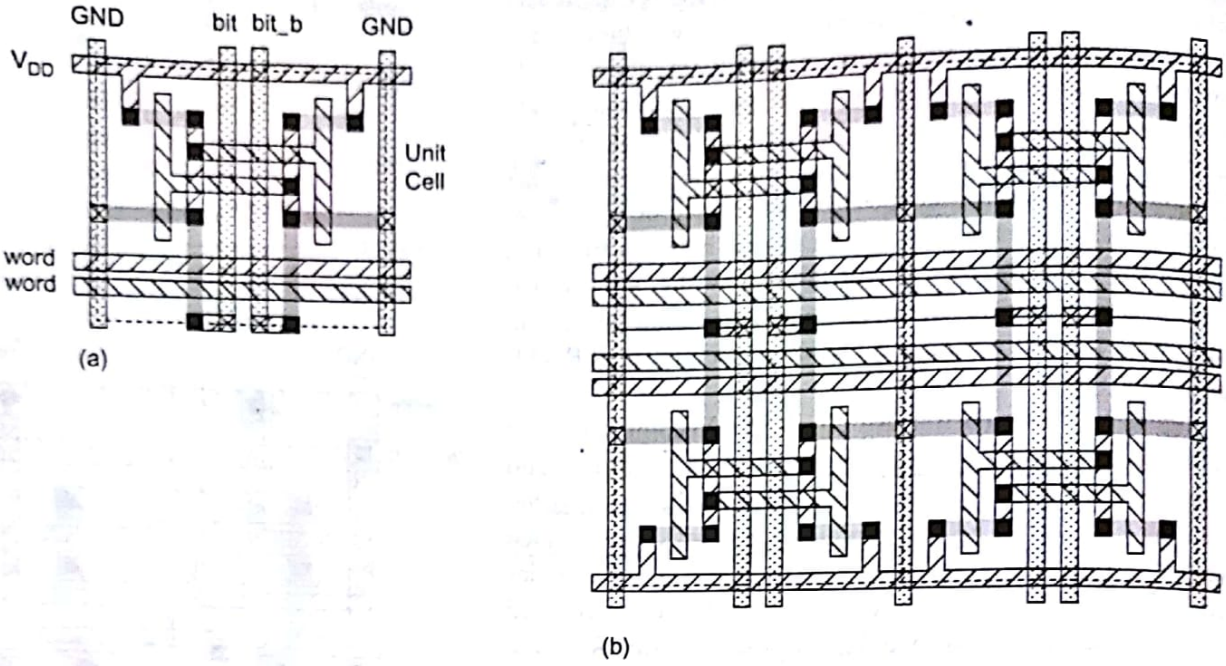


FIG 9.5 Stick diagram of 6T SRAM cell

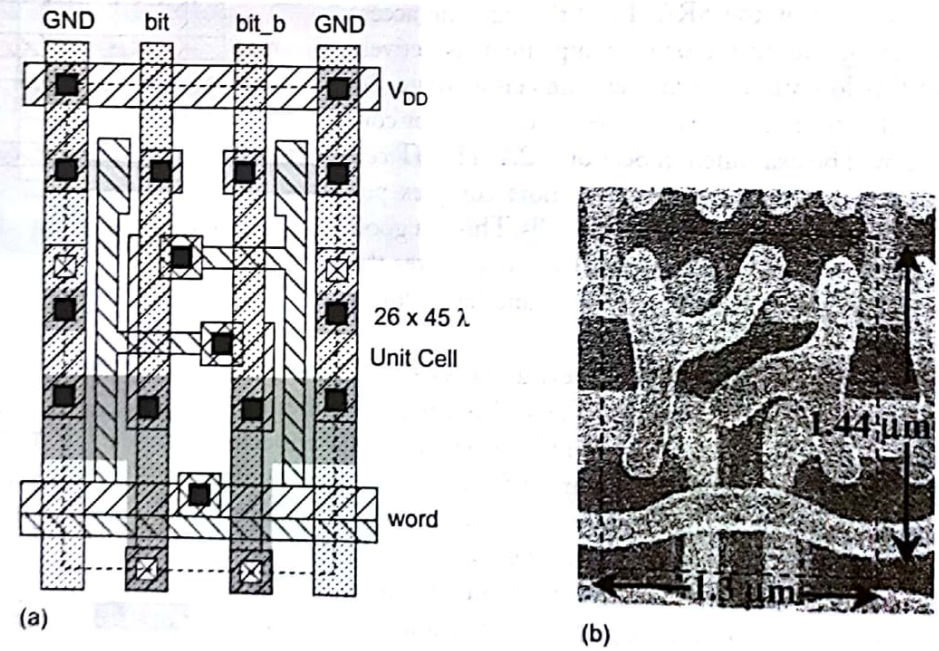


FIG 9.6 Layout of 6T SRAM cell. © IEEE 2001. (See color version given at the beginning of the book.)

SRAM operation is divided into two phases. As described in Section 7.4.3, the phases will be called ϕ_1 and ϕ_2, but may actually be generated from *clk* and its complement *clkb*. Assume that in phase 2, the SRAM is precharged. In phase 1, the SRAM is written or read by raising the appropriate wordline and either driving

the bitlines to the value that should be written or leaving the bitlines floating and observing which one is pulled down. Reading a large SRAM can be slow because the capacitance of all the cells sharing the bitline is large. Sense amplifiers accelerate reads by detecting small differences between the bitline and its complement. The following sections discuss the role of each block from Figure 9.2 in the SRAM operation.

9.2.1 Memory Cell Read/Write Operation

Figure 9.7 shows a SRAM cell being read. The bitlines are both initially floating high. Without loss of generality, assume *A* is initially '0' and thus *A_b* is initially '1.' *A_b* and *bit_b* both should remain '1'. When the wordline is raised, *bit* should be pulled down through transistors *N1* and *N2*. At the same time *bit* is being pulled down, node *A* tends to rise. *A* is held low by *N1*, but raised by current flowing in from *N2*. Hence, *N1* must be stronger than *N2*. Specifically, the transistors must be ratioed such that node *A* remains below the switching threshold of the *P2/N3* inverter. This constraint is called *read stability*. Waveforms for the read operation are shown in Figure 9.7(b) as a 0 is read onto *bit*. Observe that *A* momentarily rises, but does not glitch badly enough to flip the cell.

Figure 9.8 shows the same cell in the context of a full column from the SRAM. During phase 2, the bitlines are precharged high. The wordline only rises during phase 1; hence, it can be viewed as a _q1 qualified clock. Many SRAM cells share the same bitline pair, which acts as a distributed dual-rail footless dynamic multiplexer. The capacitance of the entire bitline must be discharged through the access transistor. The output can be sensed by a pair of HI-skew inverters. By raising the switching threshold of the sense inverters, delay can be reduced at the expense of noise margin. The outputs are dual-rail monotonically rising signals, just as in a domino gate.

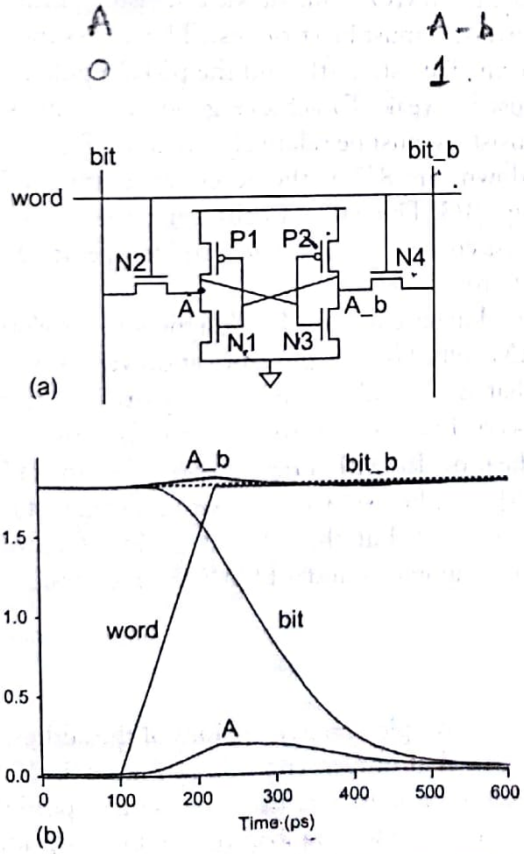


FIG 9.7 Read operation for 6T SRAM cell

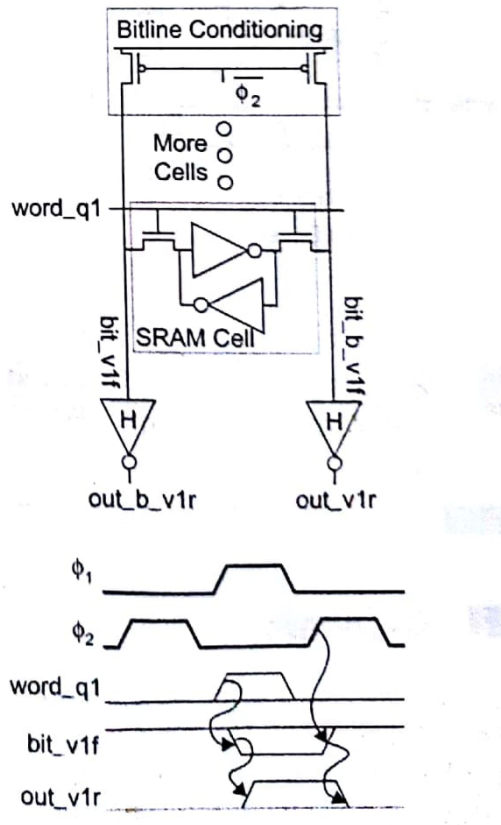


FIG 9.8 SRAM column read

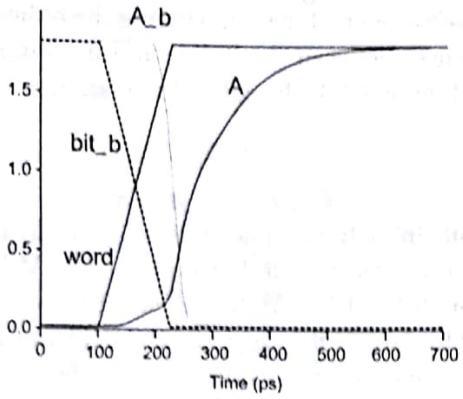


FIG 9.9 Write operation for 6T SRAM cell

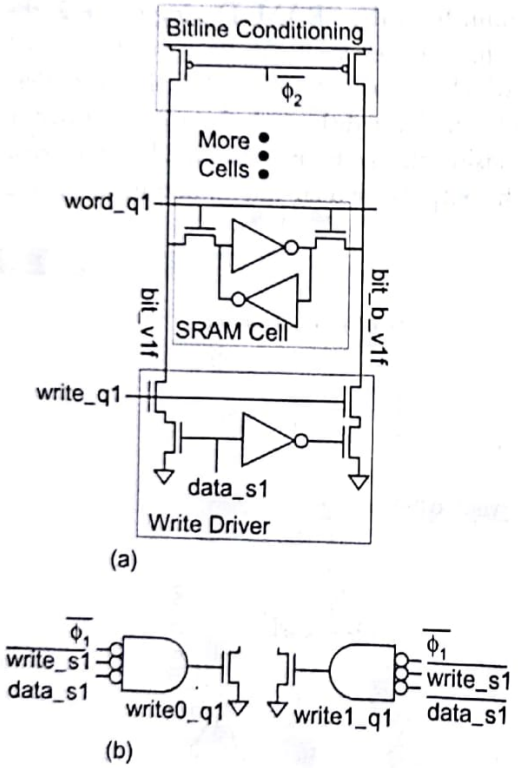


FIG 9.10 SRAM column write

The waveforms of Figure 9.9 show the SRAM cell being written. Again, assume *A* is initially '0' and that we wish to write a '1' into the cell. *bit* is precharged high and left floating. *bit_b* is pulled low by a write driver. We know on account of the read stability constraint that *bit* will be unable to force *A* high through *N2*. Hence, the cell must be written by forcing *A_b* low through *N4*. *P2* opposes this operation; thus, *P2* must be weaker than *N4* so that *A_b* can be pulled low enough. This constraint is called *writeability*. Once *A_b* falls low, *N1* turns OFF and *P1* turns ON, pulling *A* high as desired.

Figure 9.10(a) again shows the cell in the context of a full column from the SRAM. During phase 2, the bitlines are precharged high. Write drivers pull the bitline or its complement low during phase 1 to write the cell. The write drivers can consist of a pair of transistors on each bitline for the data and the write enable, or a single transistor driven by the appropriate combination of signals (Figure 9.10(b)). In either case, the series resistance of the write driver, bitline wire, and access transistor must be low enough to overpower the pMOS transistor in the SRAM cell.

In summary, to ensure both read stability and writeability, the nMOS pull-down transistor in the cross-coupled inverters must be strongest. The access transistors are of intermediate strength, and the pMOS pull-up transistors must be weak. To achieve good layout density, all of the transistors must be relatively small. In Figure 9.6(a), the pull-downs are 8/2 λ, the access transistors 4/2, and the pull-ups 3/3. The SRAM cells must operate correctly in all process corners at all voltages and temperatures. This requires thorough simulation.

It is no longer common for designers to develop their own SRAM cells. Usually, the fabrication vendor will supply cells that are carefully tuned to the particular manufacturing process. In high-performance processes, two or more cells may be provided with different speed/density tradeoffs. [Glasser85] describes SRAM cells with four transistors and two large resistors, but the 6T cell is almost universally used in contemporary standard CMOS processes.

9.2.2 Decoders

The simplest decoder is a collection of AND gates using true and complementary versions of the address bits. Figure 9.11 shows several straightforward implementations. The first implementation in Figure 9.11(a) is a static NAND gate followed by an inverter. This structure is useful for up to 5–6 inputs or more if speed is not critical. The NAND transistors are usually made minimum size to reduce the load on the buffered address lines because there are 2^{n-k} transistors on each true and complementary address line in the row decoder. The second implementation shown in Figure 9.11(b) uses a pseudo-nMOS NOR gate buffered with two inverters. The NOR gate transistors can be made minimum size and the inverters can be scaled appropriately to drive the wordline.

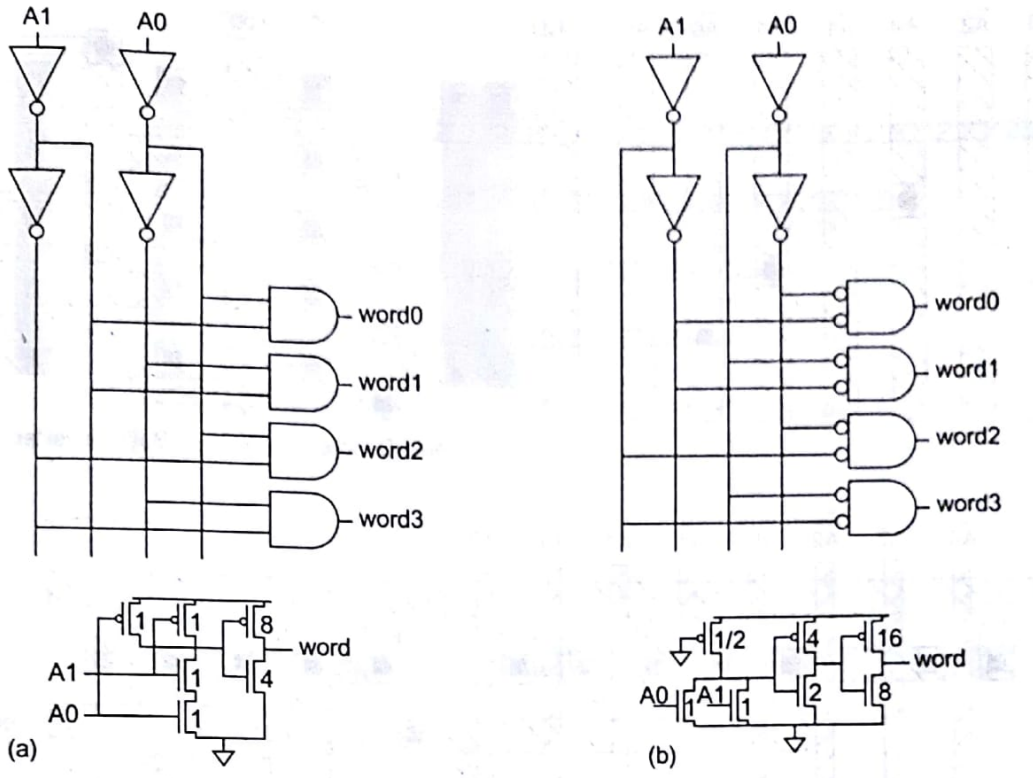


FIG 9.11 Decoders

The layout of the decoder must be pitch-matched to the memory array, i.e., the height of each decoder gate must match the height of the row it drives. This can be tricky for SRAM and even harder for ROMs and other arrays with small memory cells. Figure 9.12(a) shows a layout of a conventional standard-cell style approach. The minimum-sized transistors in the NAND gate drive a larger buffer inverter. The decoder height grows with the number of inputs. The AND gates are easily programmed by connecting the polysilicon inputs to the appropriate address inputs. Figure 9.12(b) shows a layout on a pitch that is tighter and independent of the number of inputs. The decoder is programmed by placement of transistors and metal straps; this is best done with scripting software that generates layout, as discussed in Section 10.2.6. The polysilicon address lines should be strapped with metal2 to reduce their resistance, but the metal2 is left out of the figure for readability. The decoder pitch is 5 tracks or 40 λ. If every other row is mirrored to share V_{DD} and GND, the pitch can be reduced to 4 tracks or 32 λ.

9.2.2.1 Predecoding Decoders with many inputs can be formed from a cascade of smaller gates. For example, Figure 9.13(a) shows a 16-word decoder in which the 4-input AND function is built from a pair of 2-input NANDs followed by a 2-input NOR. Many NAND gates share exactly the same inputs and are thus redundant. The decoder area can be improved by factoring these common NANDs out, as shown in Figure 9.13(b). This technique is called *predecoding*. It does not change the path effort of the decoder, but does improve area. In general, blocks of p address bits can be predecoded into 1-of-2^p-hot predecoded lines that serve as inputs to the final stage decoder. For example, Figure 9.13(b) shows a p = 2-bit design that decodes each pair of address bits into a 1-of-4-hot code.

The wordline generally must be qualified with the clock for proper bitline timing. This is often performed with another AND gate after the decoder or with an extra clk input to the final stage of decoding.

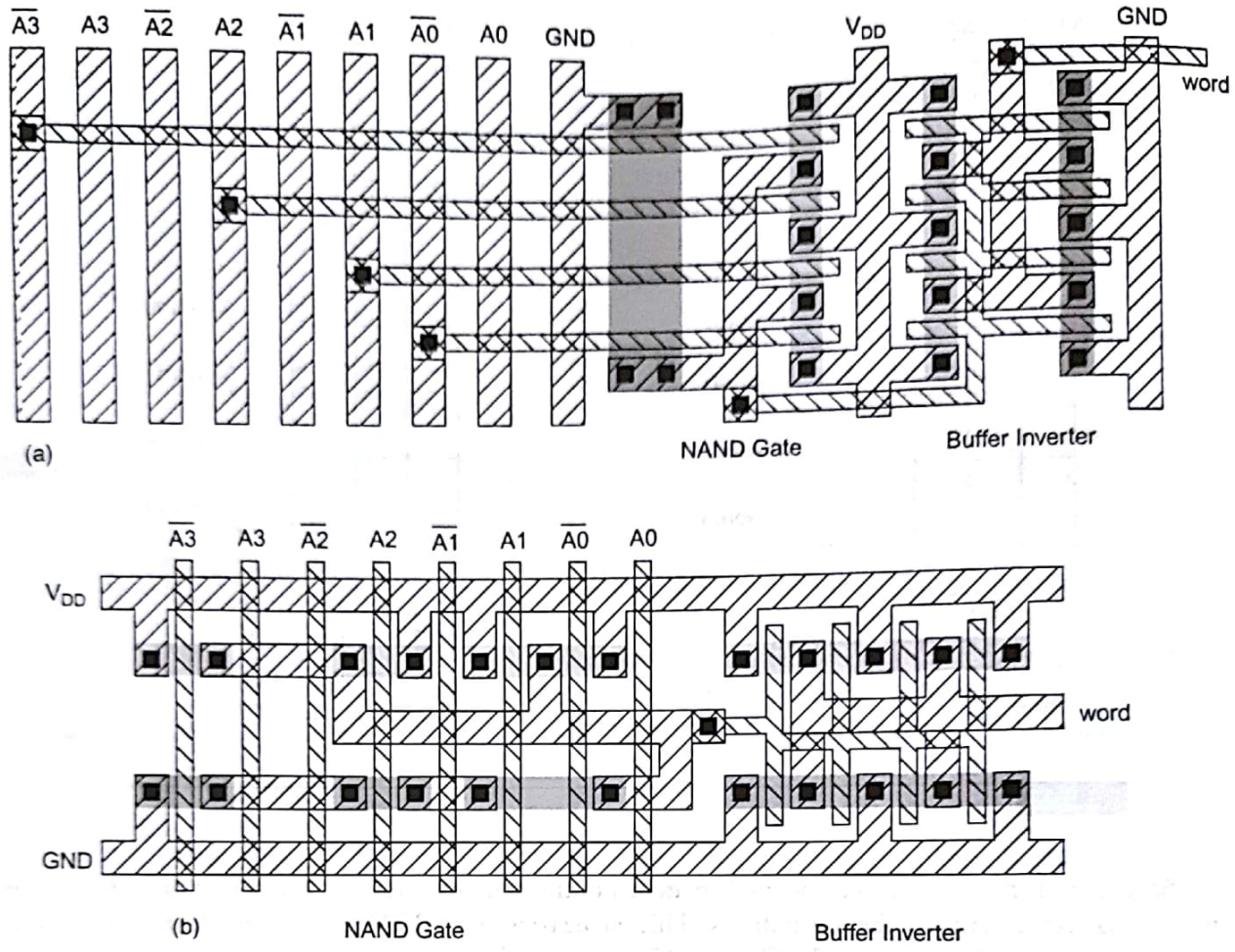
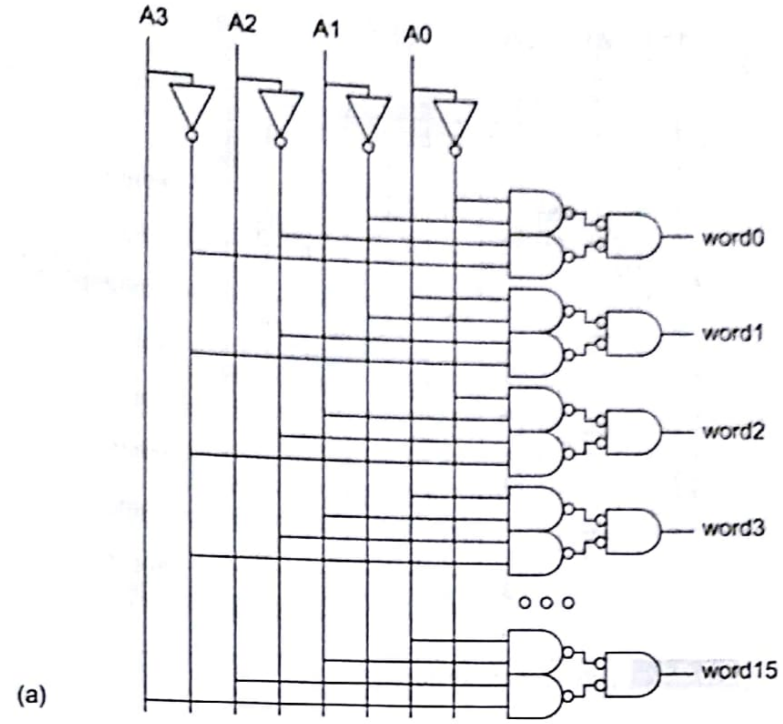


FIG 9.12 Stick diagrams of two decoder layouts

9.2.2.2 Faster Decoders The logical effort of a decoder can be reduced by observing that only one of the outputs will be high so the pMOS transistors can be shared among many outputs [Lyon87]. A NOR gate pulls low efficiently through parallel transistors, but has a poor logical effort because the output is pulled high through wide series transistors. The Lyon-Schediwy decoder can be viewed as 2^n n-input NOR gates sharing pMOS pull-ups, as shown in Figure 9.14 for a 3:8 decoder. The cost of the wide transistors is amortized across many outputs. Relative transistor widths are chosen to present the same capacitance to each input while providing current drive equal to a unit inverter. The logical effort of each input is only $(1 + 3.5)/ 3 = 1.5$ per wordline output, as compared to 7/3 for an ordinary NOR3 (see Exercise 9.4).

Decoders typically have high electrical and branching effort. Therefore, they need many stages, so the fastest design is the one that minimizes the logical effort. A tree of 2- and 3-input NAND gates and inverters offers the lowest logical effort to build high fan-in gates in static CMOS [Sutherland99]. Dynamic gates are attractive for fast decoders because they have lower logical effort.

A major problem with traditional domino decoders is the high power consumption. For example, even though only one of the 256 wordlines in the previous example will rise on each cycle, all 256 AND gates must precharge so the clock load is extremely large. A much lower-power approach is to use self-resetting domino gates that only precharge the wordline that evaluated. Self-resetting domino has essentially the same performance as traditional domino because it uses the same basic gates.



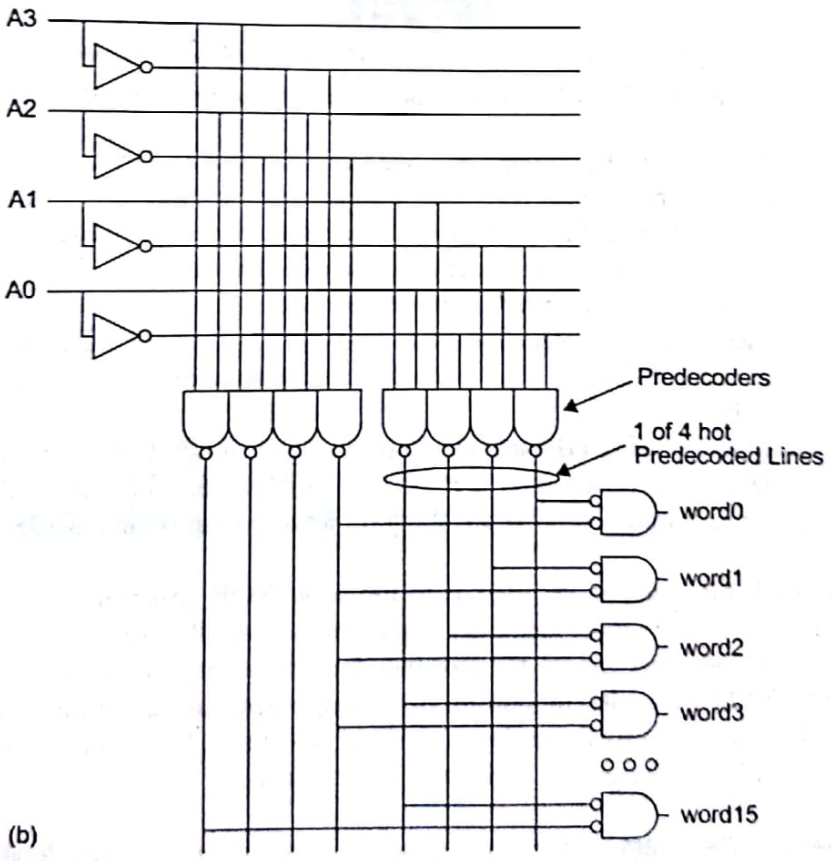


FIG 9.13 Ordinary and predecoding circuits

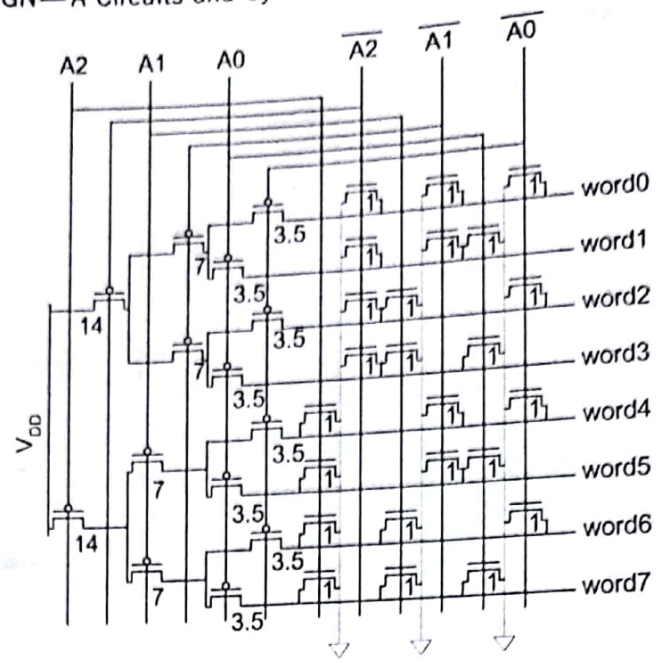


FIG 9.14 Lyon-Schediwy decoder

Example

Estimate the delays of 8:256 decoders using static CMOS and footed domino gates. Assume the decoder has an electrical effort of $H = 10$ and that both true and complementary inputs are available.

Solution: The decoder consists of 256 8-input AND gates. It has a branching effort of $B = 256/2 = 128$ because each of the true inputs and each of the complementary inputs are used by half the gates. Assuming the logical effort of the path G is close to 1, the path effort is $F = GBH = 1280$ and the best number of stages is $\log_4 F = 5.16$. Let us consider a six-stage design using three levels of 2-input AND gates, each constructed from a 2-input NAND and an inverter.

The static CMOS design has a logical effort of $G = [(4/3) \cdot (1)]^3 = 64/27$. Therefore, the stage effort is $F = 3034$. The parasitic delay is $P = 3 \cdot (2 + 1) = 9$. The total delay is $D = NF^{1/N} + P = 31.8\ \tau$ or 6.4 FO4 inverter delays.

The footed domino design using HI-skew inverters has a logical effort of $[(1) \cdot (5/6)]^3 = 125/256$ and a stage effort of 625. The parasitic delay is $P = 3 \cdot (4/3 + 5/6) = 6.5$. The total delay is 4.8 FO4 inverter delays. In general, domino decoders are about 33% faster than static CMOS.

Yet another approach for dynamic decoders is to use wide NOR structures in which N–1 of the N outputs discharge on each cycle. As most memories require monotonically rising outputs but the NORs are monotonically falling, such decoders require the race-based nonmonotonic techniques. For example, Figure 9.15 shows a 4-input AND gate with monotonically rising output using a race-based NOR structure [Nambu98]. This technique is slightly faster than a domino AND tree, but dissipates more power because the dynamic node X must be precharged on each cycle [Amrutur01].

9.2.2.3 Sum-addressed Decoders Many microprocessor instruction sets include addressing modes in which the effective address is the sum of two values, such as a base address and an offset. In conventional SRAMs used as caches, the two values must first be added, and then the result decoded to determine the cache wordline. If access latency needs to be minimized, these two steps can be combined into one in a *sum-addressed memory* [Heald98].

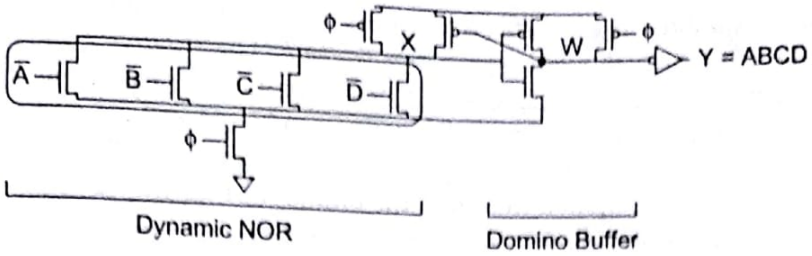


FIG 9.15 4-input AND using race-based NOR

Recall from Section 10.4.3 that checking if $A + B = K$ is faster than actually computing $A + B$ because no carry propagation need occur. A *sum-addressed decoder* for an N-word memory accepts two inputs, A and B. In a simple form, it contains N comparators driving the N wordlines. The first checks if $A + B = 0$. The second checks if $A + B = 1$, and so forth. The comparators contain redundant logic repeated across wordlines. [Heald98] shows how to reduce the area by factoring out common terms in a predecoder.

9.2.3 Bitline Conditioning and Column Circuitry

The bitline conditioning circuitry is used to precharge the bitlines high before operation. A simple conditioner consists of a pair of pMOS transistors, as shown in Figure 9.16(a). It is also possible to construct pseudo-nMOS SRAMs with weak pull-up transistors in place of the precharge transistors (Figure 9.16(b)) where no clock is available. Another technique is to precharge through nMOS transistors to $V_{DD} - V_t$ (Figure 9.16(c)). This results in faster single-ended bitline sensing because the bitlines do not swing as much, but reduces noise margins and may require more precharge time.

Each column must also contain write drivers and read sensing circuits. Figure 9.10 showed two examples of write drivers and Figure 9.8 showed the use of HI-skew inverters to sense reads. Many *sense amplifiers* have been invented to provide faster sensing by responding to a small voltage swing. The differential sense amplifier in Figure 9.17(a) is based on an analog differential pair and requires no clock. The differential gain of the amplifier is $g_{mN1}(r_{oN1} \parallel r_{oP1})$, as will be discussed in Section 11.6.5. However, the circuit consumes a significant amount of DC power. The clocked sense amplifier in Figure 9.17(b) only consumes power while activated, but requires a timing chain to activate at the proper time. When the sense clock is low, the amplifier is

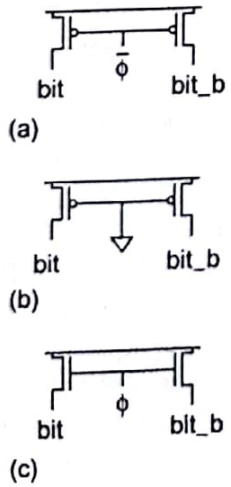


FIG 9.16 Bitline conditioning circuits

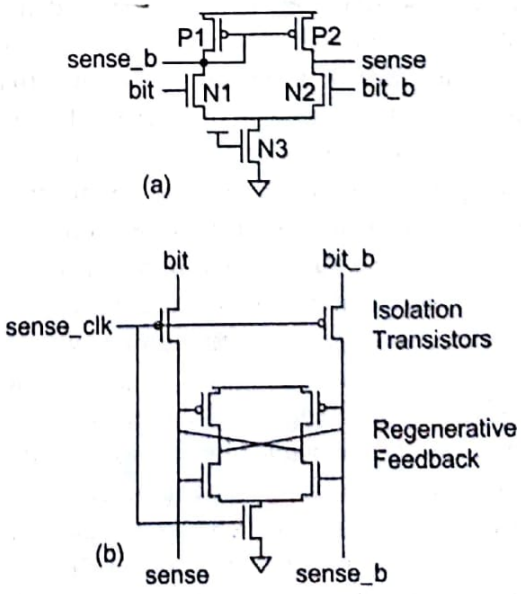


FIG 9.17 Sense amplifiers

inactive. When the sense amplifier rises, it effectively turns on the cross-coupled inverter pair, which pulls one output low and the other high through *regenerative feedback*. The *isolation transistors* speed up the response by disconnecting the outputs from the highly capacitive bitlines during sensing. See Section 6.3.2 for more discussion of sense amplifier circuits. Power dissipation can be reduced for read operations by turning off the wordlines once sufficient differential voltage has been achieved on the bitlines. This reduces the bitline swing and hence the charge required to restore the bitlines to V_{DD} after sensing. *Current sense amplifiers* that detect a differential current rather than a small voltage swing are another promising technique [Wicht01].

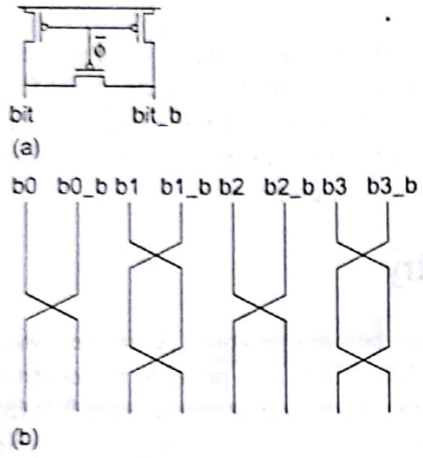


FIG 9.18 Bitline noise reduction through equalizers and twisting

Sense amplifiers are very susceptible to differential noise on the bitlines because they detect small voltage differences. If bitlines are not precharged long enough, residual voltages on the lines from the previous read may cause pattern-dependent failure. An equalizer transistor (Figure 9.18(a)) can be added to the bitline conditioning circuits to reduce the required precharge time by ensuring that *bit* and *bit_b* are at nearly equal voltage levels even if they have not precharged quite all the way to V_{DD}. Coupling from transitioning bitlines in neighboring cells may also introduce noise. The bitlines can be *twisted* or *transposed* to cause equal coupling onto both the bitline and its complement, as shown in Figure 9.18(b). For example, careful inspection shows that *b1* couples to *b0_b* for the first quarter of its length, *b2* for the next quarter, *b2_b* for the third quarter, and *b0* for the final quarter. *b1_b* also couples to each of these four aggressors for a quarter of its length, so the coupling will be the same onto both lines.

The sense amplifier offset voltage is the differential input voltage (*bit* – *bit_b*) necessary to produce zero differential output voltage (*sense* – *sense_b*). If *N1* is identical to *N2* and *P1* to *P2*, the sense amplifier will ideally have zero offset voltage. In practice, the offset voltage is nonzero because of statistical dopant fluctuations that affect V_t. The differential input must substantially exceed the offset voltage to be sensed reliably. A typical budget for offset voltage is 50 mV [Amrutur00]. Unfortunately, the threshold variations and offset voltage are not changing very much with technology scaling, so the offset voltage is becoming a larger fraction of the supply voltage, making sense amplifiers less effective [Mizuno94].

Clocked sense amplifiers must be activated at just the right time. If they fire too early, the bitlines may not have developed enough voltage difference to operate reliably. If they fire too late, the SRAM is unnecessarily slow. The clock is generated by circuitry that must match the delay of the decoder, wordlines, and bitlines. Many arrays use a chain of inverters, but inverters do not track the delay of the access path very well across process and environmental corners: A margin of more than 30% is often necessary in the typical corner for reliable operation in all corners. Alternatively, the array may use dummy or *replica* cells and bitlines to more closely track the access path. For example, [Amrutur98] describes a 16-kbit SRAM array that eliminates 3 FO4 inverter delays of margin by replacing the inverter chain with an extra replica row and column configured to operate no faster than the slowest real access.

In general, 2^k:1 column multiplexers may be required to extract 2^m bits from the 2^{m+k} bits of each row. The column multiplexers can either act as their own tree decoder or require a separate column decoder to generate select signals. Figure 9.19 shows an 8:1 column multiplexer using a tree decoder. The data is routed through pass transistors enabled by the column address lines. The address decoding is in essence distributed. Decoders for both polarities of the bitline are shown, although one of these can be omitted for single-ended read operations. The read (and usually of lesser importance, write) operations are somewhat delayed by the series pass transistors. Figure 9.20 shows a single-ended 4:1 column mux using a separate column decoder. The multiplexer is faster because data from the bitline must propagate through only one series transistor. The

column decoding takes place in parallel with row decoding so it does not impact delay. In both cases, the outputs may need to be precharged because only nMOS pass transistors are used. Column multiplexers can also use full transmission gates. pMOS pass transistors can be used if a sense amplifier responds to voltages near V_{DD}.

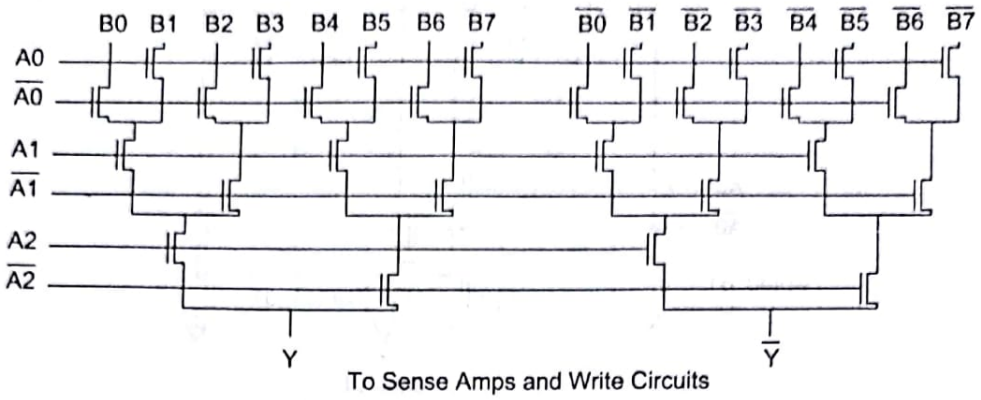


FIG 9.19 Tree decoder column multiplexer

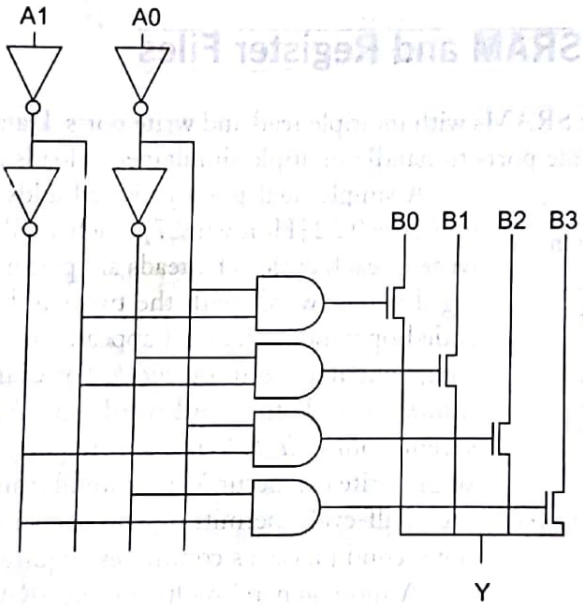


FIG 9.20 Column multiplexer with separate decoder

Figure 9.21 shows a complete pair of bits and associated column circuitry for a 2-way multiplexed SRAM. The output of the nMOS-only multiplexer is precharged high. Both the write drivers and the read sensing inverter are connected to the multiplexer outputs.

Column multiplexing is also helpful because the bit pitch of each column is so narrow that it can be difficult to lay out a sense amplifier for each column. After multiplexing, multiple columns are available for the remainder of the column circuitry. Moreover, placing sense amplifiers after the column multiplexers reduces the number of power-hungry amplifiers required in the array.

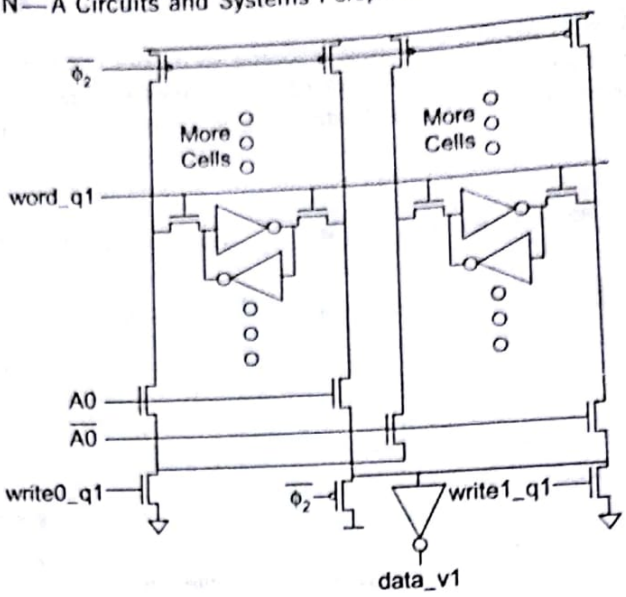


FIG 9.21 Complete pair of columns for 2-way multiplexed SRAM

9.2.4 Multi-ported SRAM and Register Files

Register files are generally fast SRAMs with multiple read and write ports. Data caches in superscalar microprocessors often require multiple ports to handle multiple simultaneous loads and stores.

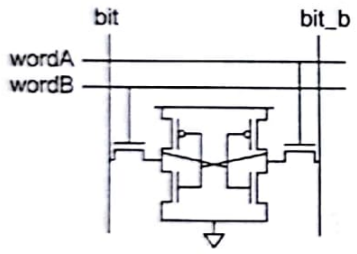


FIG 9.22 Simple dual-ported SRAM

A simple dual-ported SRAM adds a second wordline, as shown in Figure 9.22 [Horowitz87]. Such a cell can perform two reads or one write in each cycle. The reads are performed by independently selecting different words with the two wordlines. Read becomes a single-ended operation; one read appears on *bit*, while the other appears in complementary form on *bit_b*. For example, asserting *wordA*[7] and *wordB*[3] reads the third word onto *bit* and the complement of the seventh onto *bit_b*. Write still requires both *bit* and *bit_b*, so only a single write can occur. With careful timing, accesses can be performed each half-cycle, permitting two reads in the first phase and a write in the second phase, as commonly required for pipelined processors.

A more general multi-ported SRAM with three write ports and four read ports is shown in Figure 9.23. Merely adding more access transistors causes problems for read stability when multiple ports attempt to read the same cell because all the access transistors will be driving the cell high. Instead, the bitlines are isolated from the cell during reads. The multi-port SRAM cell still is built around the cross-coupled inverters in the center. Ports *A*, *B*, and *C* perform writes by driving a value onto one side of the inverter pair and its complement onto the other side. Ports *D*, *E*, *F*, and *G* perform reads by discharging the bitline if the cell stores a 0.

Register files for superscalar processors often require an enormous number of ports. For example, the Itanium 2 processor issues up to six integer instructions in a cycle, each of which requires two source registers and a destination. The register file requires two more write ports for late cache data returns, leading to a total of 12 read ports and 8 write ports [Fetzer02]. The area of the large register file is dominated by the mesh of wordlines and bitlines. A rough rule for estimating multiport SRAM cell area is to count the number of tracks for the wordlines and bitlines and then add three in each dimension for internal wiring. The number of

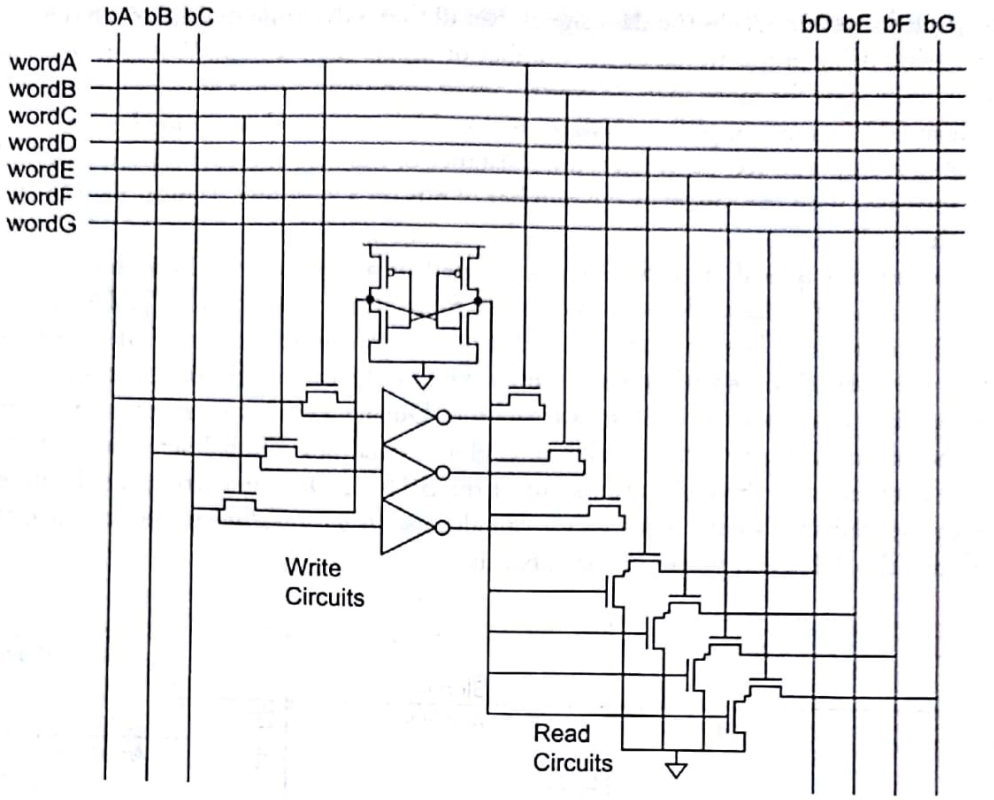


FIG 9.23 General multi-ported SRAM

bitlines can be reduced by performing single-ended read operations. Similarly, writes can use a local inverter placed in empty space under the wires so only one, rather than two bitlines are required for write ports. The Itanium 2 further reduces the area to only 12 wordlines by time-multiplexing, reading in one half-cycle, and then writing on the other using pulses on the wordlines. [Golden99] and [Heald00] show other designs for the large register files of the AMD Athlon and Sun UltraSparc III, respectively.

A single-ended register file with 16 read ports and 4 write ports has a cell size of 23×23 tracks, or about $184 \times 184\ \lambda = 33856\ \lambda^2$. The area can be improved by partitioning the register file into two parts, each with 8 read ports and 4 write ports. Write operations update both register files so the data remains consistent, but half the reads use one register file and half use the other. The cell size is now 15×15 tracks with an area of $14400\ \lambda^2$ per file, or $28800\ \lambda^2$ all together. The partitioned register file is not only smaller but also faster because of the shorter bitlines and wordlines.

9.2.5 Large SRAMs

The critical path in a static RAM read cycle includes the clock to address delay time, the row address driver time, row decode time, bitline sense time, and the setup time to any data register. The write operation is usually faster than the read cycle because the bitlines are actively driven by large transistors. However, the bitlines may have to be allowed to recover to their quiescent values before any more access cycles take place.

If the memory array becomes large, the wordlines and bitlines become rather long. The long lines have high capacitance, leading to long delay and high power consumption. Thus, large memories are partitioned into multiple smaller memory arrays, sometimes called *banks* or *subarrays*. Each subarray presents some area overhead for its periphery circuitry, so the size of the subarrays represents a tradeoff between area and speed.

Reading data onto a bitline is much like activating a wide footless dynamic multiplexer because the bitline is pulled down through two series transistors. The wordline transistor acts as the select and the nMOS

in the cross-coupled inverter acts as the data signal. Recall that a dynamic multiplexer has a constant logical effort but a parasitic delay proportional to the number of inputs (i.e., words). Without sense amplifiers, the bitlines become quite slow for more than 32 words. Sense amplifiers permit smaller bitline swings, reducing the parasitic delay. With sense amplifiers, bitlines reasonably accommodate up to about 128 words.

The wordline presents an RC delay from the resistance of the wire and gate capacitance of the transistors it drives. This increases with the square of the number of bits on a wordline. Typical SRAMs use up to about 256 bits on each wordline.

Figure 9.24 shows a typical large SRAM partitioned into S 4-kbyte (128-word × 256-bit) subarrays [Amrutur00]. The *divided wordline* decoder operates in three stages: predecoding, global wordline decoding, and local wordline decoding [Yoshimoto83]. The 128 global wordlines are long, but lightly loaded and can use wide, upper-level metal wires with low resistance. The local wordline decoder gates the global wordline with a bank select line to activate the appropriate subarray. During a read, the subarray sense amplifiers detect the swing of the bitline and drive the results onto global datalines. Global sense amplifiers detect small swings on these datalines and drive the results out of the SRAM. During a write, the datalines and bitlines are driven in reverse. Subarrays with many words can also use *divided bitlines* to reduce the diffusion capacitance and improve the delay and power of the subarray.

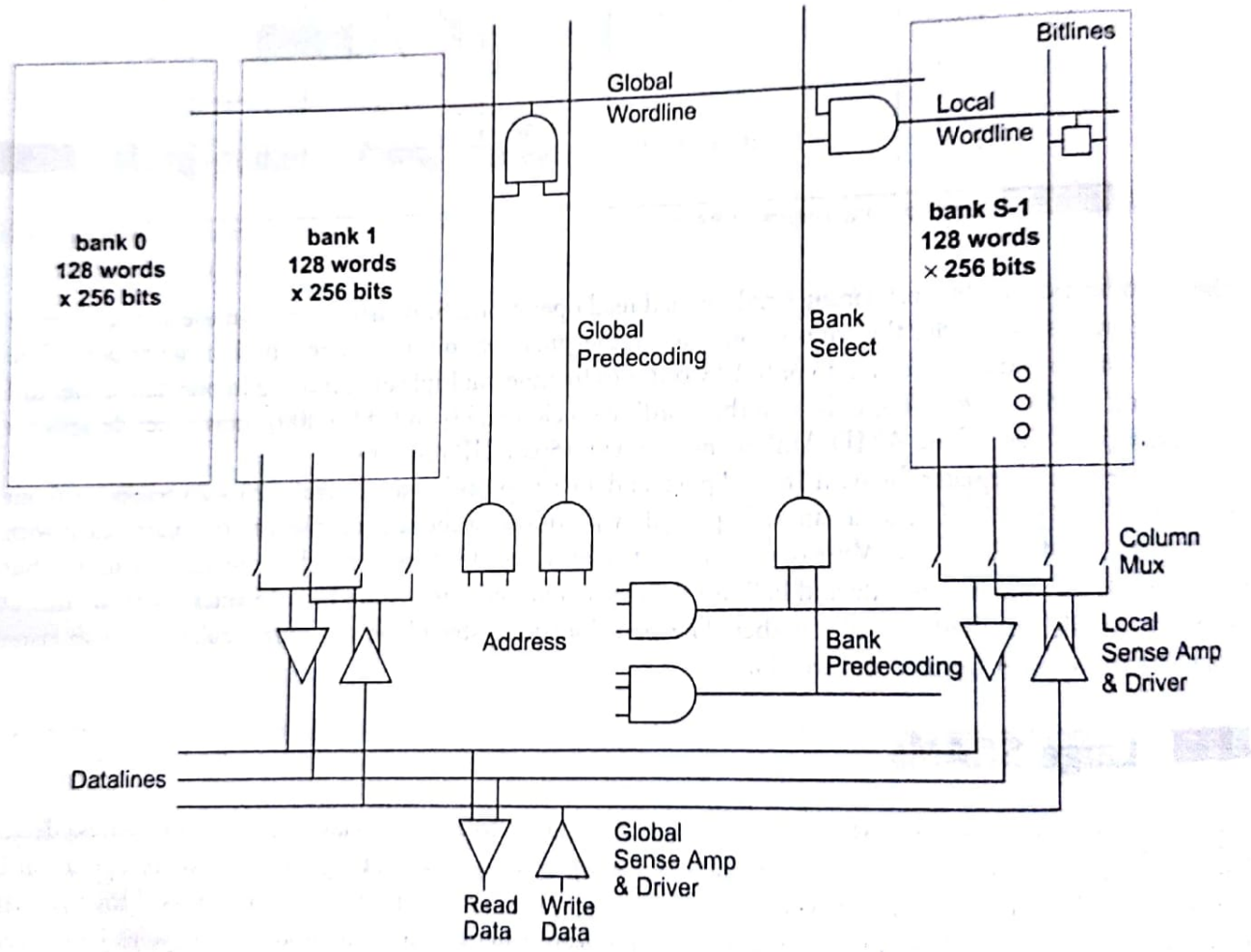


FIG 9.24 Large memory array architecture with subarrays

Large memories with multiple subarrays can simulate more than one access port even if each subarray is single-ported. For example, in a system with two subarrays, even-numbered words could be stored in one subarray while odd-numbered words are stored in the other. Two accesses could occur simultaneously if one addresses an even word and another an odd word. If both address an even word, we encounter a *bank conflict*

and one access must wait. Increasing the number of banks offers more parallelism and lower probability of bank conflicts.

9.3 DRAM

Dynamic RAMs (DRAMs) store their contents as charge on a capacitor rather than in a feedback loop. Thus, the basic cell is substantially smaller than SRAM, but the cell must be periodically read and refreshed so that its contents do not leak away. Commercial DRAMs are built in specialized processes optimized for dense capacitor structures. They offer an order of magnitude greater density (bits/cm^2) than high-performance SRAM built in a standard logic process, but they also have much higher latency. DRAM circuit design is a very specialized art and is the topic of excellent books such as [Keeth01]. This section provides an overview of the general technique.

A 1-transistor (1T) dynamic RAM cell consists of a transistor and a capacitor, as shown in Figure 9.25(a). Like SRAM, the cell is accessed by asserting the wordline to connect the capacitor to the bitline. On a read, the bitline is first precharged to $V_{DD}/2$. When the wordline rises, the capacitor shares its charge with the bitline, causing a voltage change ΔV that can be sensed, as shown in Figure 9.25(b). The read disturbs the cell contents at x, so the cell must be rewritten after each read. On a write, the bitline is driven high or low and the voltage is forced onto the capacitor. Some DRAMs drive the wordline to $V_{DDP} = V_{DD} + V_t$ to avoid a degraded level when writing a '1.'

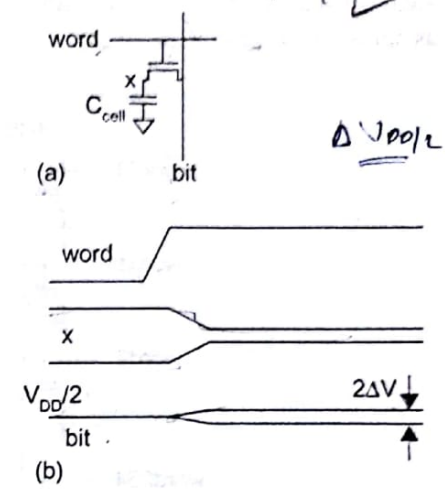
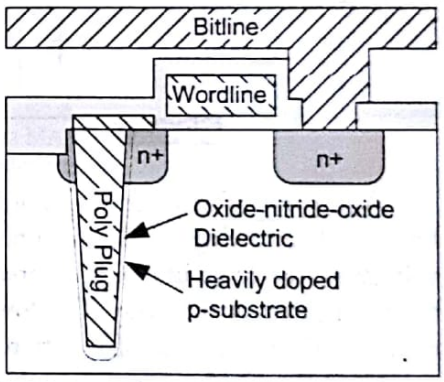


FIG 9.25 DRAM cell read operation

The DRAM capacitor C_{cell} must be as physically small as possible to achieve good density. However, the bitline is contacted to many DRAM cells and has a relatively large capacitance C_{bit}. Therefore, the cell capacitance is typically much smaller than the bitline capacitance. According to the charge-sharing equation, the voltage swing on the bitline during readout is

$$\Delta V = \frac{V_{DD}}{2} \frac{C_{cell}}{C_{cell} + C_{bit}} \tag{9.1}$$

We see that a large cell capacitance is important to provide a reasonable voltage swing. It also is necessary to retain the contents of the cell for an acceptably long time and to minimize soft errors, as was mentioned in Section 4.7.7. For example, 30 fF is a typical target. The most compact way to build such a high capacitance is to extend into the third dimension. For example, Figure 9.26 shows a cross-section and scanning electron microscope (SEM) image of *trench capacitors* etched under the source of the transistor. The walls of the trench are lined with an oxide-nitride-oxide dielectric. The trench is then filled with a polysilicon conductor that serves as one terminal of the capacitor attached to the transistor drain, while the heavily doped substrate serves as the other terminal. A variety of three-dimensional capacitor structures have been used in specialized DRAM processes that are not available in conventional CMOS processes.



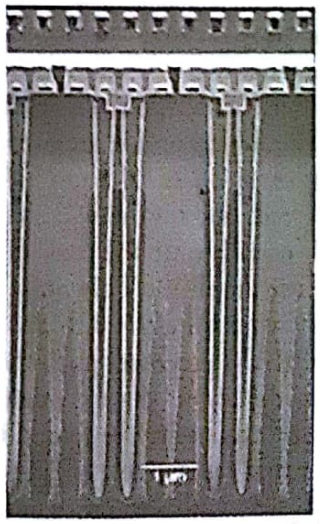


FIG 9.26 Trench capacitor

9.3.1 Subarray Architectures

Like SRAMs described in Section 9.2.5, large DRAMs are divided into multiple subarrays. The subarray size represents a tradeoff between density and performance. Larger subarrays amortize the decoders and sense amplifiers across more cells and thus achieve better density. But they also are slow and have small bitline swings because of the high wordline and bitline capacitance. A typical subarray size is 256 words by 512 bits, as shown in Figure 9.27.

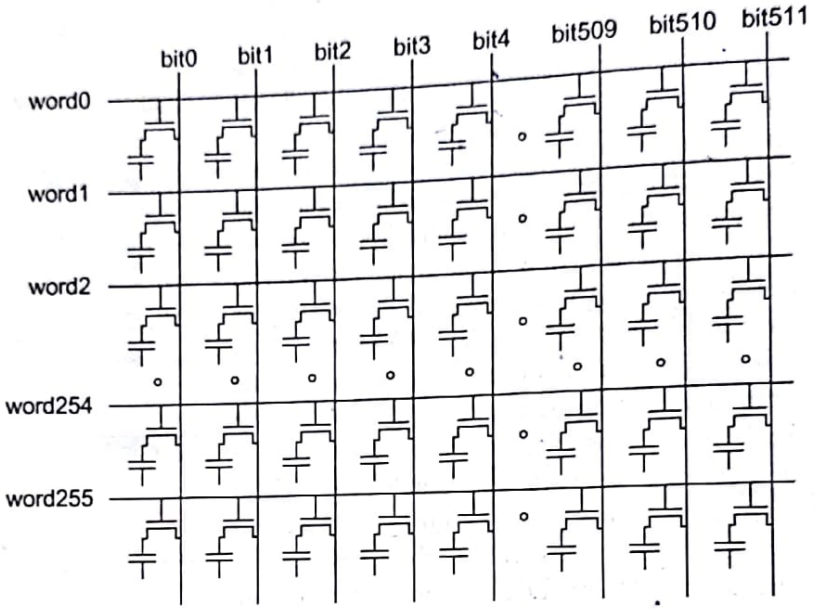


FIG 9.27 DRAM subarray

A subarray of this size has an order of magnitude higher capacitance on the bitline than in the cell, so the bitline voltage swing ΔV during a read is very small. The array uses a sense amplifier to compare the bitline voltage to that of an idle bitline (precharged to $V_{DD}/2$). The sense amplifier must also be very compact to fit the tight pitch of the array. The low-swing bitlines are very sensitive to noise. Three bitline architectures, *open*, *folded*, and *twisted*, offer different compromises between noise and area.

Until the 64 kbit generation, DRAMs used the open bitline architecture shown in Figure 9.28. In this architecture, the sense amplifier receives one bitline from each of two subarrays. The wordline is only asserted in one array, leaving the bitlines in the other array floating at the reference voltage. The arrays are very dense. However, any noise that affects one array more than the other will appear as differential noise at the sense amplifier. Thus, open bitlines have unacceptably low signal-to-noise ratios for high-density DRAM.

The folded bitline architecture is shown in Figure 9.29. In this architecture, each bitline connects to only half as many cells. Adjacent bitlines are organized in pairs as inputs to the sense amplifiers. When a wordline is asserted, one bitline will switch while its neighbor serves as the quiet reference. Many noise sources will couple equally onto the two adjacent bitlines so they tend to appear as common mode noise that is rejected by the sense amplifier. This noise advantage comes at the expense of greater layout area. Figure 9.30 shows a clever layout for a 6 × 8 folded bitline subarray that is only 33% larger than an open bitline layout. Observe how DRAM processes push the design rules and use diagonal polysilicon to reduce area. Notice how pairs of cells in the layout share a single bitline contact to minimize the bitline capacitance.

Unfortunately, the folded bitline architecture is still susceptible to noise from a neighboring switching bitline that capacitively couples more strongly onto one of the bitlines in the pair. Capacitive coupling is very significant in modern processes. The twisted bitline architecture [Hidaka89] solves this problem by swapping the positions of the folded bitlines part way along the array in much the same way as SRAM bitlines were twisted in Figure 9.18(b). The twists cost a small amount of extra area within the array.

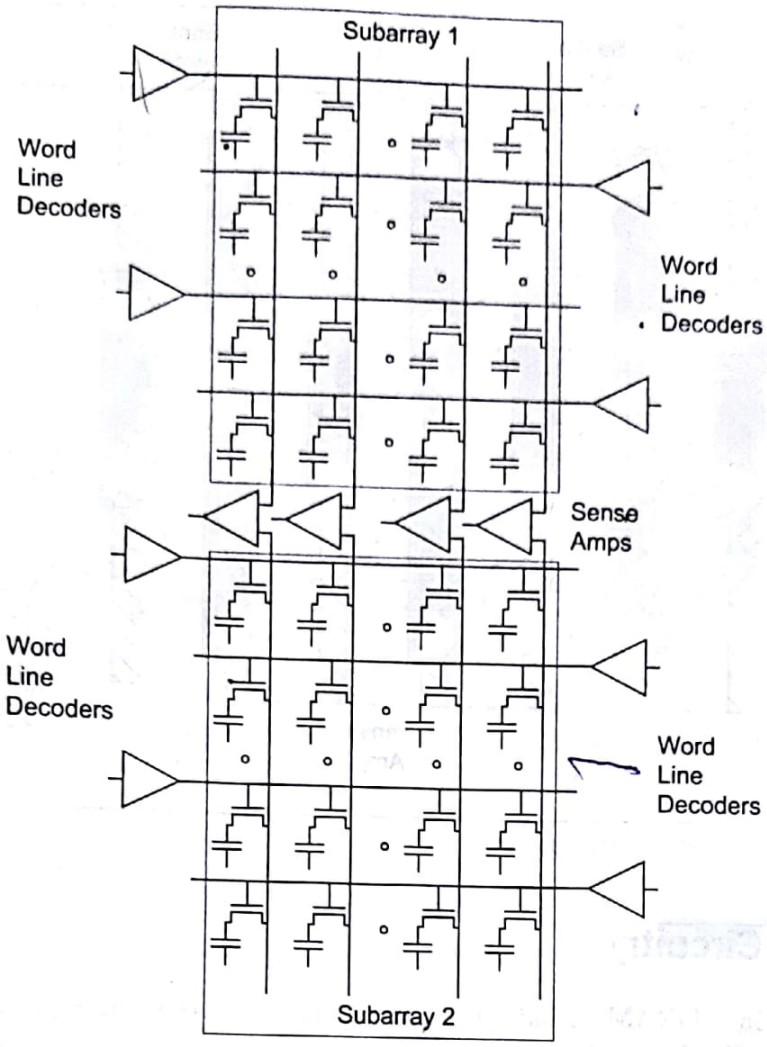


FIG 9.28 Open bitlines

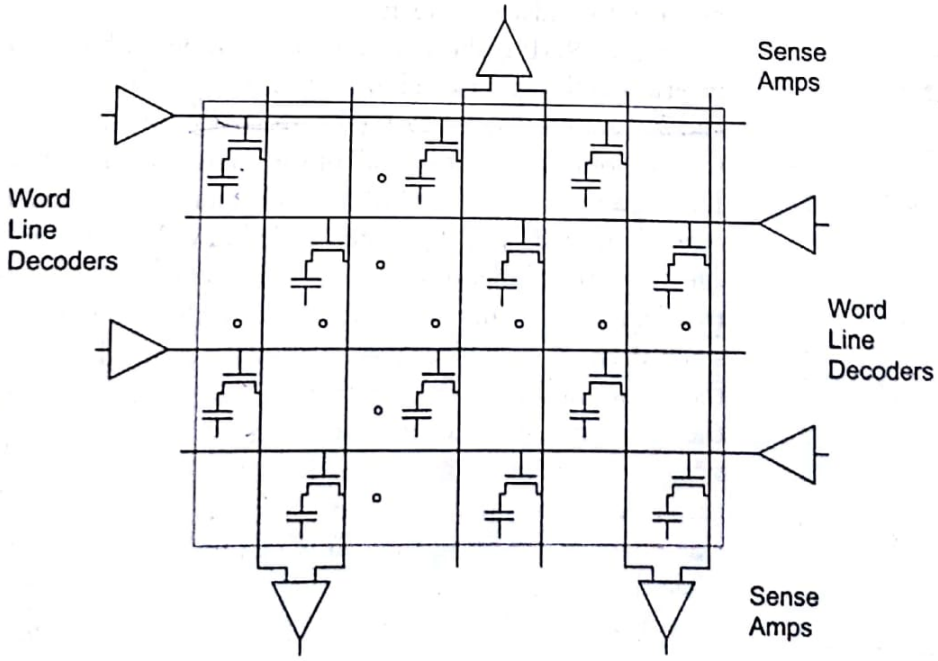


FIG 9.29 Folded bitlines

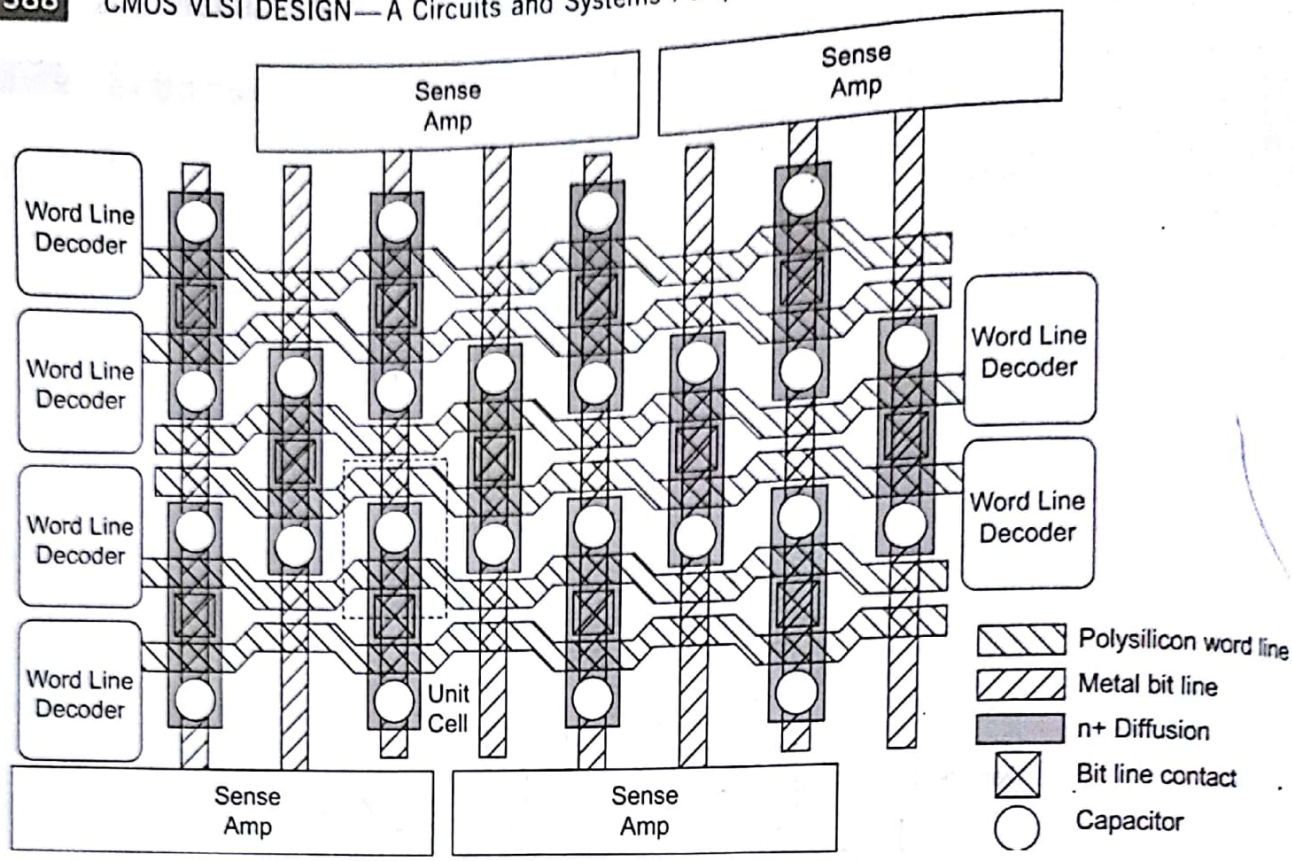


FIG 9.30 Layout of folded bitline subarray

9.3.2 Column Circuitry

The column circuitry in a DRAM includes the sense amplifiers, write drivers, column multiplexing, and bitline conditioning circuits. In a folded or twisted bitline architecture, the column circuitry is placed on both sides of the array so that it can be laid out on four times the pitch of a single column, as was shown in Figure 9.30. Part of the circuitry can be shared between two adjacent subarrays.

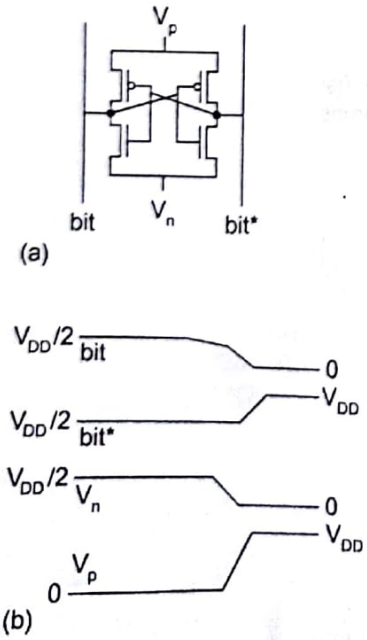


FIG 9.31 Sense amplifier

Figure 9.31(a) shows a basic sense amplifier built from cross-coupled inverters with supplies tied to control voltages. Initially, the two bitlines *bit* and *bit** are precharged to $V_{DD}/2$, the bottom voltage V_n is at $V_{DD}/2$, and the top voltage V_p is at 0 so all of the transistors in the amplifier are OFF. During a read, one of the bitlines will change by a small amount while the other floats at $V_{DD}/2$. V_n is then pulled low. As it falls to a threshold voltage below the higher of the two bitline voltages, the cross-coupled nMOS transistors will begin to pull the lower bitline voltage down to 0. After a small delay, V_p is pulled high. The cross-coupled pMOS transistors pull the higher bitline voltage up to V_{DD}. For example, Figure 9.31(b) shows the waveforms while reading a '0' on *bit* while using *bit** as a reference. Driving the active bitline to one of the rails has the side effect of rewriting the cell with the value that was just read.

Figure 9.32 shows a bitline conditioning circuit that precharges and equalizes a pair of bitlines to $V_{DD}/2$ when EQ is as-

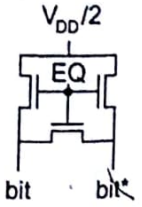


FIG 9.32 Bitline conditioning

serted. This consumes very little power because the voltage is reached by sharing charge between one bitline at V_{DD} and the other at GND.

Figure 9.33 puts together the complete column circuitry serving two folded subarrays. Each subarray column produces a pair of signals, *bit* and *bit**. The *CSEL* signal, produced by the column decoder, determines if this column will be connected to the I/O line for the array. Each subarray has its own equalization transistors and pMOS portion of the sense amplifier. However, the nMOS sense amplifier and I/O lines are shared between the subarrays. Either *ISO1* or *ISO2* is asserted to connect one subarray to the I/O lines while leaving the other isolated. During a read operation, the data is read onto the I/O lines. During a write, one I/O line is driven high and the other low to force a value onto the bitlines. The cross-coupled pMOS transistors pull the bitlines to a full logic level during a write to compensate for the threshold drop through the isolation transistor.

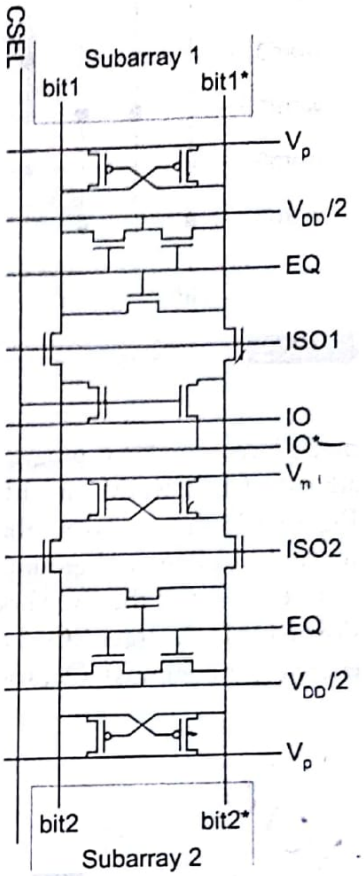


FIG 9.33 Column circuitry

9.4 Read-only Memory

Read-only Memory (ROM) cells can be built with only one transistor per bit of storage. A ROM is a nonvolatile memory structure in that the state is retained indefinitely—even without power. A ROM array is commonly implemented as a single-ended NOR array using any of the NOR gate structures studied so far, including the pseudo-nMOS and the footless-dynamic NOR gate. As in SRAM cells and other footless dynamic gates, the wordline input must be low during precharge on dynamic NOR gates. In situations where DC power dissipation is acceptable and the speed is sufficient, the pseudo-nMOS ROM is the easiest to design, requiring no timing. The DC power dissipation can be significantly reduced in multiplexed ROMs by placing the pull-up transistors after the column multiplexer.

Figure 9.34 shows a 4-word by 6-bit ROM using pseudo-nMOS pull-ups with the following contents:

```
word0: 010101
word1: 011001
word2: 100101
word3: 101010
```

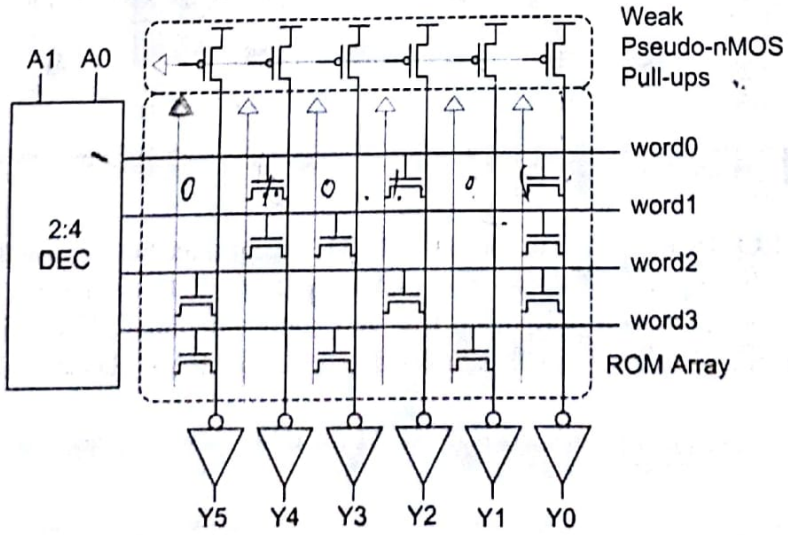


FIG 9.34 Pseudo-nMOS ROM

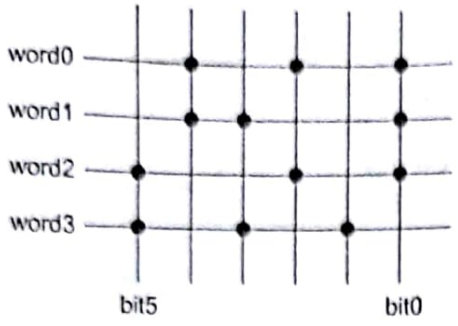


FIG 9.35 Dot diagram representation of ROM

The contents of the ROM can be symbolically represented with a dot diagram in which dots indicate the presence of 1's, as shown in Figure 9.35. The dots actually correspond to nMOS pull-down transistors connected to the bitlines, but the outputs are inverted.

Mask-programmed ROMs can be configured by the presence or absence of a transistor or contact, or by a threshold implant that turns a transistor permanently OFF where it is not needed. Omitting transistors has the advantage of reducing capacitance on the wordlines and power consumption. Programming with metal contacts was once popular because such ROMs could be completely manufactured except for the metal layer, and then programmed according to customer requirements through a metallization step. The advent of EEPROM and Flash memory chips has reduced demand for such mask-programmed ROMs. Figure 9.36 shows a layout for the 4-word by 6-bit ROM array. The wordlines run horizontally in polysilicon, while the bitlines and grounds run vertically in metal1. Notice how each ground is shared between a pair of cells. Each bit of the ROM occupies a $12 \times 8\ \lambda$ cell[1]. Polysilicon wordlines are only appropriate for small or slow ROMs. A larger ROM can run metal2 straps over the polysilicon and contact the two periodically (e.g., every eight columns). Occasional substrate contacts are also required.

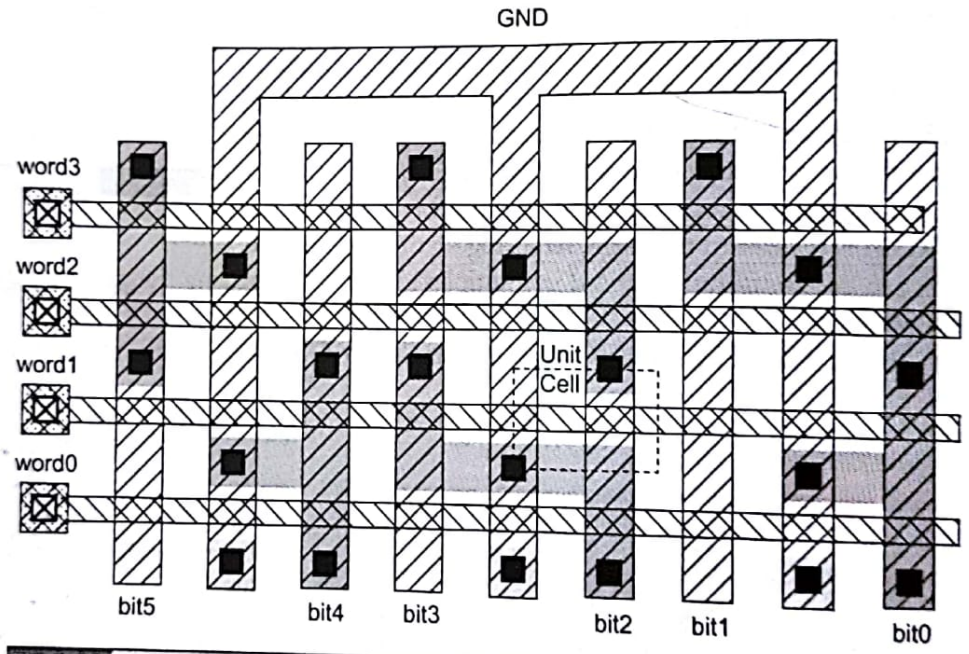


FIG 9.36 ROM array layout

Row decoders for ROMS are similar to those for RAMs except that they are usually very constrained by the ROM wordline pitch. Figure 9.37 shows how each output of a 2:4 decoder can be shoehorned into a single horizontal track using vertical polysilicon true and complementary address lines and metal supply lines. Column decoders for ROMs are usually simpler than those for RAMs because single-ended sensing is commonly employed.

Figure 9.38 shows a complete pseudo-nMOS ROM including row decoder, cell array, pMOS pull-ups, and output inverters.

[1]The cell can be reduced to $11 \times 7\ \lambda$ by running the ground line in diffusion and by reducing the width and spacing to $3\ \lambda$.

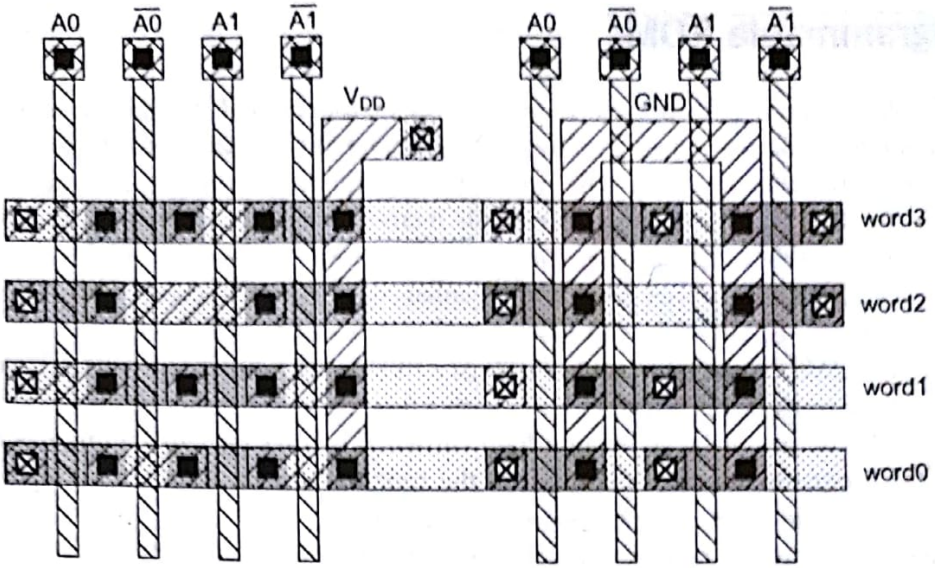


FIG 9.37 Row decoder layout on tight pitch

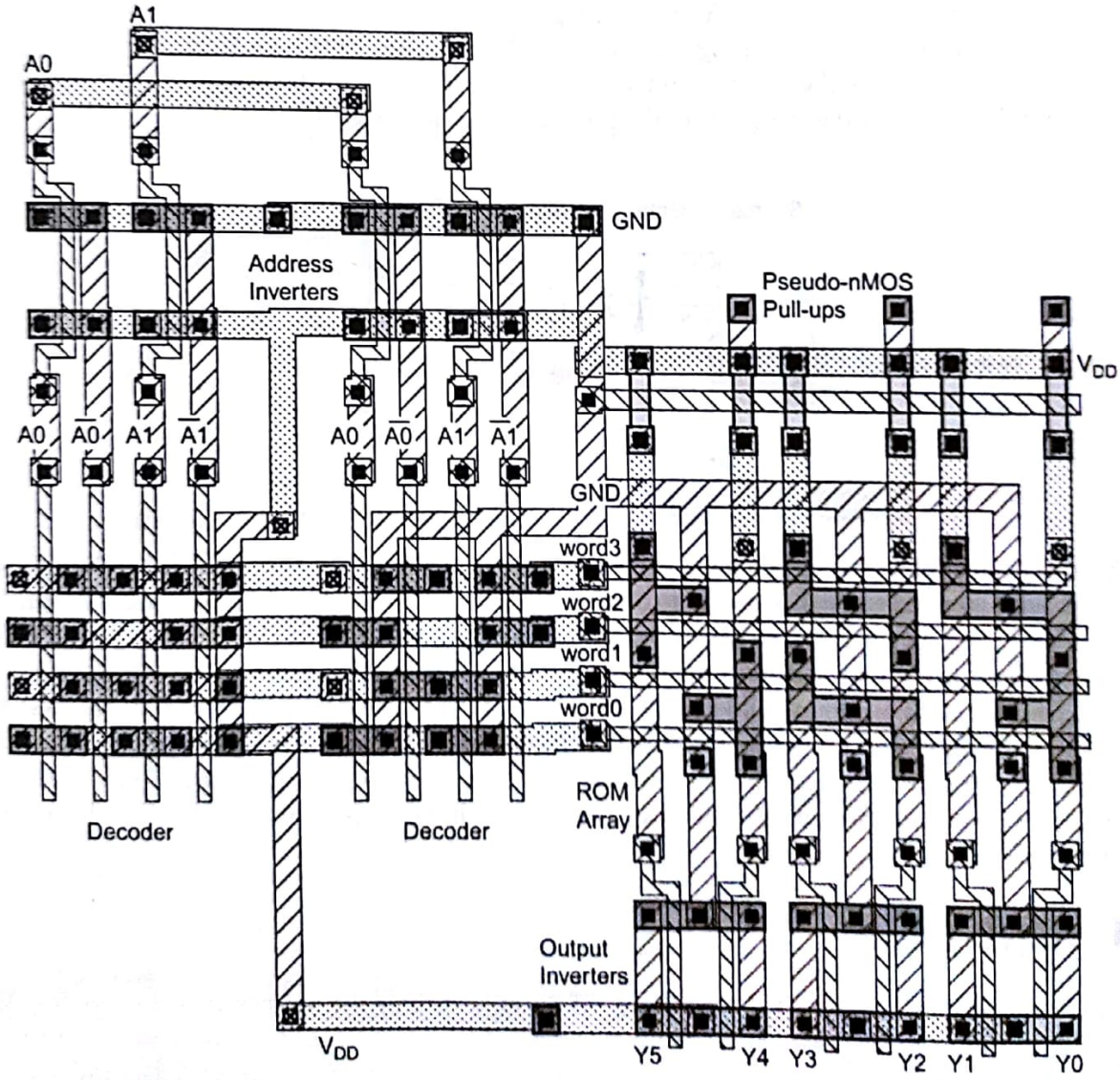


FIG 9.38 Complete ROM layout

9.4.1 Programmable ROMs

It is often desirable for the user to be able to program or reprogram a ROM after it is manufactured. As discussed in Section 9.1, this contradicts the "Read Only" nature of the device; ROM has in practice become synonymous with *nonvolatile*, not read-only memory. Programming/writing speeds are generally slower than read speeds for ROMs. Four types of nonvolatile memories include *Programmable* ROMs (PROMs), *Erasable Programmable* ROMs (EPROMs), *Electrically Erasable Programmable* ROMs (EEPROMs), and *Flash* memories. All of these memories require some enhancements to a standard CMOS process: PROMs use fuses while EPROMs, EEPROMs, and Flash use charge stored on a floating gate.

Programmable ROMs can be fabricated as ordinary ROMs fully populated with pull-down transistors in every position. Each transistor is placed in series with a fuse made of polysilicon, nichrome, or some other conductor that can be burnt out by applying a high current. The user typically configures the ROM in a specialized PROM programmer before putting it in the system. As there is no way to repair a blown fuse, PROMs are also referred to as *one-time programmable* memories.

As technology has improved, reprogrammable nonvolatile memory has largely displaced PROMs. These memories, including EPROM, EEPROM, and Flash, use a second layer of polysilicon to form a floating gate between the primary gate and the channel, as shown in Figure 9.39. The floating gate is a good conductor, but it is not attached to anything. Applying a high voltage to the upper gate causes electrons to jump through the thin oxide onto the floating gate through the processes called *avalanche injection* or *Fowler-Nordheim tunneling*. Injecting the electrons induces a negative voltage on the floating gate, effectively increasing the threshold voltage of the transistor to the point that it is always OFF.

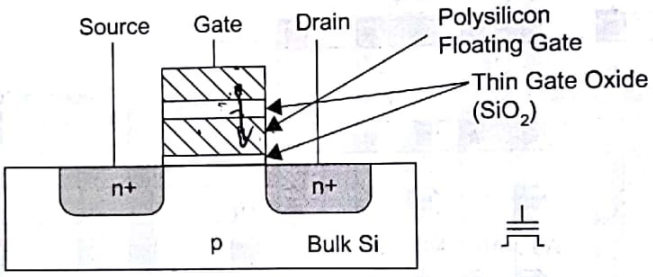


FIG 9.39 Cross-section of floating gate nMOS transistor

[Itoh01k] and [Rabaey03] offer a good overview of these memories. In brief, EPROM is programmed electrically, but it is erased through exposure to ultraviolet light that knocks the electrons off the floating gate. It offers a dense cell, but it is inconvenient to erase and reprogram. EEPROM and Flash can be erased electrically without being removed from the system. EEPROM offers fine-grained control over which bits are erased, while Flash is erased in bulk. EEPROM cells are larger to enable them to provide this versatility, so Flash has become the most economical form of convenient nonvolatile storage. For example, Flash memory cards are widely used in digital cameras to store pictures even after the camera is turned off. Flash is also useful for firmware or configuration data because it can be rewritten to upgrade a system in the field without opening the case or removing parts.

9.4.2 NAND ROMs

The ROM from Figure 9.34 is called a NOR ROM because each of the bitlines is just a pseudo-nMOS NOR gate. The bitline pulls down when a wordline attached to any of the transistors is asserted high. The size of the cell is limited by the ground line. Figure 9.40 shows a NAND ROM that uses active-low wordlines. Transistors are placed in series and the transistors on the nonselected rows are ON. If no transistor is associated with the selected word, the bitline will pull down. If a transistor is present, the bitline will remain high.

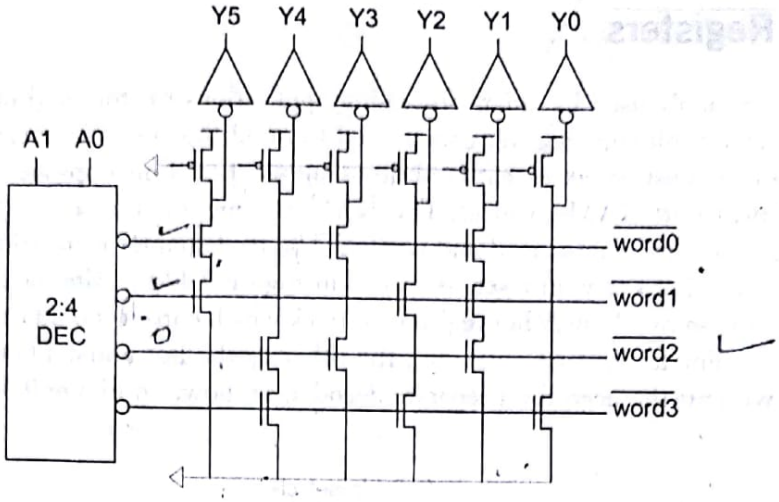


FIG 9.40 Pseudo-nMOS NAND ROM

Figure 9.41(a) shows a layout of the NAND ROM. The cell size is only 7 × 8 λ. The contents are specified by using either a transistor or a metal jumper in each bit position. The contacts limit the cell size. Figure 9.41(b) shows an even smaller layout in which transistors are located at every position. In this design, an extra implantation step can be used to create a negative threshold voltage, turning certain transistors permanently ON where they are not needed. In such a process, the cell size reduces to only 6 × 5 λ, assuming that the decoder and bitline circuitry can be built on such a tight pitch.

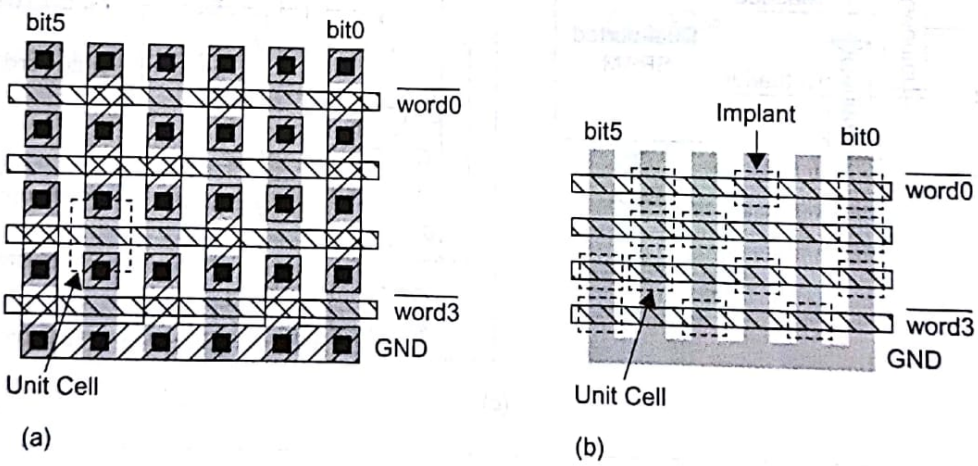


FIG 9.41 NAND ROM array layouts

A disadvantage of the NAND ROM is that the delay grows quadratically with the number of series transistors discharging the bitline. NAND structures with more than 8–16 series transistors become extremely slow, so NAND ROMs are often broken into multiple small banks with a limited number of series transistors. Nevertheless, these NAND structures are attractive for Flash memories in which density and cost are more important than access time.

9.5 Serial Access Memories

Using the basic SRAM cell and/or registers, we can construct a variety of serial access memories including shift registers and queues. These memories avoid the need for external logic to track addresses for reading or writing.

9.5.1 Shift Registers

A *shift register* is commonly used in signal-processing applications to store and delay data. Figure 9.42(a) shows a simple 4-stage 8-bit shift register constructed from 32 flip-flops. As there is no logic between the registers, particular care must be taken that hold times are satisfied. Flip-flops are rather big, so large, dense shift registers use dual-port RAMs instead. The RAM is configured as a circular buffer with a pair of counters specifying where the data is read and written. The read counter is initialized to the first entry and the write counter to the last entry on reset, as shown in Figure 9.42(b). Alternately, the counters in an N-stage shift register can use two 1-of-N hot registers to track which entries should be read and written. Again, one is initialized to point to the first entry and the other to the last entry. These registers can drive the wordlines directly without the need for a separate decoder, as shown in Figure 9.42(c).

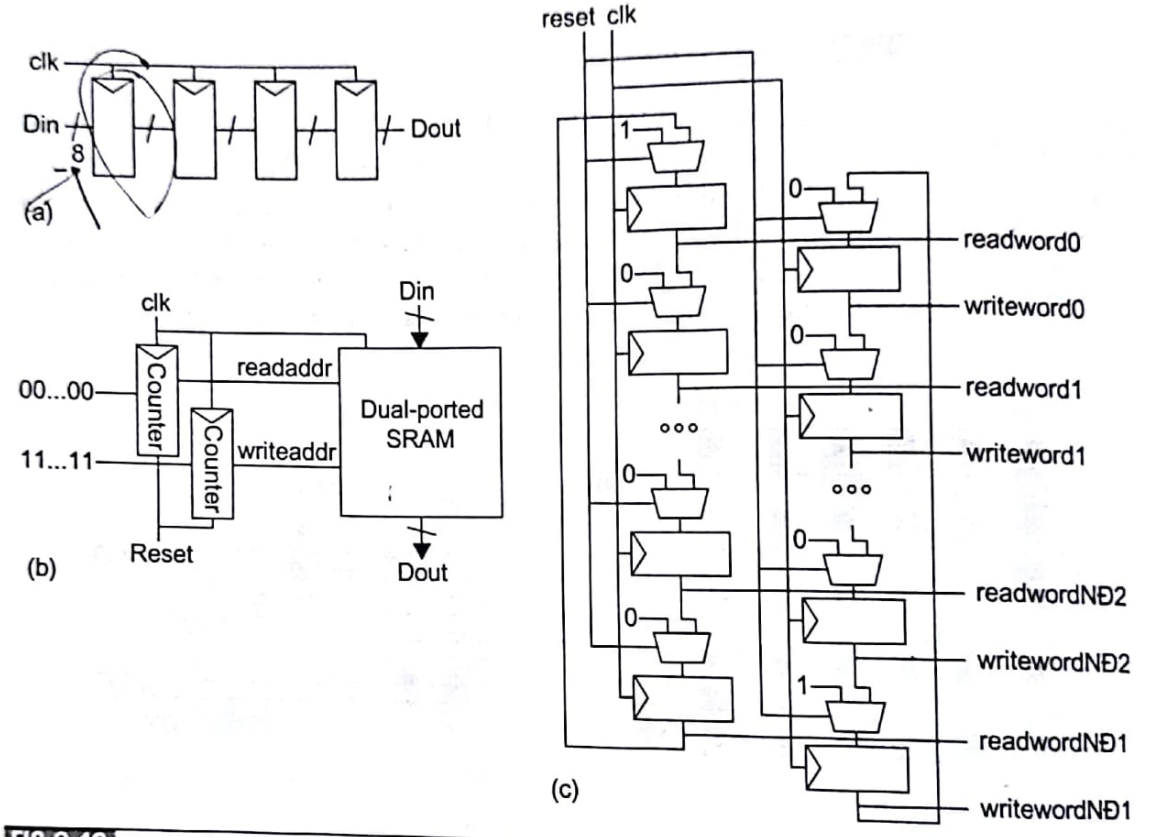


FIG 9.42 Shift registers

One variant of a shift register is a *tapped delay line* that offers a variable number of stages of delay. Figure 9.43 shows a 64-stage tapped delay line that could be used in a video processing system. Delay blocks are built from 32-, 16-, 8-, 4-, 2-, and 1-stage shift registers. Multiplexers control pass-around of the delay blocks to provide the appropriate total delay.

Another variant is a serial/parallel memory. Figure 9.44(a) shows a 4-stage Serial In Parallel Out (SIPO) memory and Figure 9.44(b) shows a 4-stage Parallel In Serial Out (PISO) memory. These are also often useful in signal processing and communications systems.

9.5.2 Queues (FIFO, LIFO)

Queues allow data to be read and written at different rates. Figure 9.45 shows an interface to a queue. The read and write operations each are controlled by their own clocks that may be asynchronous. The queue asserts the

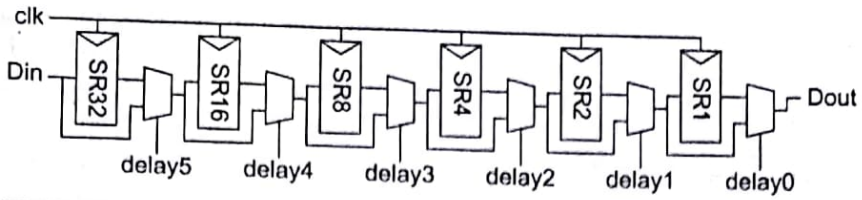


FIG 9.43 Tapped delay line

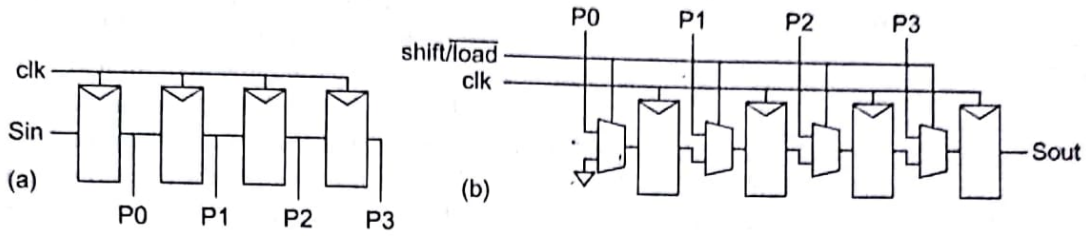


FIG 9.44 Serial/parallel memories

FULL flag when there is no room remaining to write data and the *EMPTY* flag when there is no data to read. Because of other system delays, some queues also provide ALMOST-FULL and ALMOST-EMPTY flags to communicate the impending state and halt write or read requests. The queue internally maintains read and write pointers indicating which data should be accessed next. As with a shift register, the pointers can be counters or 1-of-*N* hot registers.

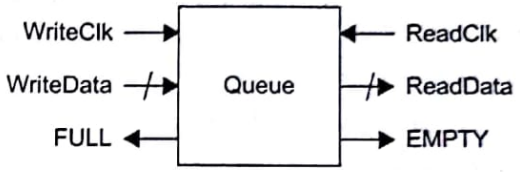


FIG 9.45 Queue

First In First Out (*FIFO*) queues are commonly used to buffer data between two asynchronous streams. Like a shift register, the FIFO is organized as a circular buffer. On reset, the read and write pointers are both initialized to the first element and the FIFO is EMPTY. On a write, the write pointer advances to the next element. If it is about to catch the read pointer, the FIFO is FULL. On a read, the read pointer advances to the next element. If it catches the write pointer, the FIFO is EMPTY again.

Last In First Out (*LIFO*) queues, also known as *stacks*, are used in applications such as subroutine or interrupt stacks in microcontrollers. The LIFO uses a single pointer for both read and write. On reset, the pointer is initialized to the first element and the LIFO is EMPTY. On a write, the pointer is incremented. If it reaches the last element, the LIFO is FULL. On a read, the pointer is decremented. If it reaches the first element, the LIFO is EMPTY again.

9.6 Content-addressable Memory

Figure 9.46 shows the symbol for a content-addressable memory (CAM) [Grosspietsch92, Schultz95, Miyatake01]. The CAM acts as an ordinary SRAM that can be read or written given adr and data, but also performs *matching* operations. Matching asserts a *matchline* output for each word of the CAM that contains a specified *key*.

A common application of CAMs is translation lookaside buffers (TLBs) in microprocessors supporting virtual memory. The virtual address is given as the key to the TLB CAM. If this address is in the CAM, the corresponding matchline is asserted. This

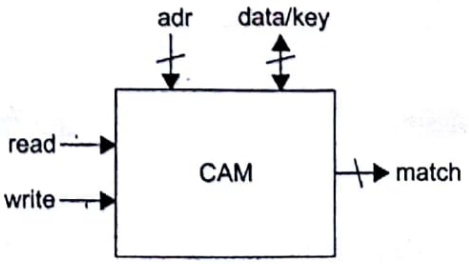


FIG 9.46 Content-addressable memory

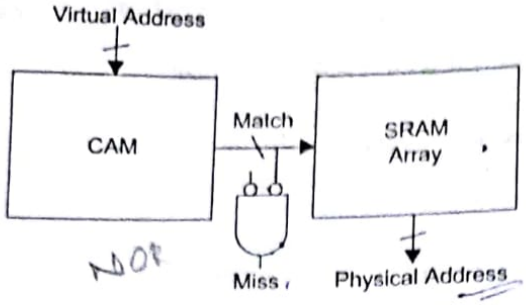


FIG 9.47 Translation Lookaside Buffer (TLB) using CAM

matchline can serve as the wordline to access a RAM containing the associated physical address as shown in Figure 9.47. A NOR gate processing all of the matchlines generates a *Miss* signal for the CAM. Note that the *read*, *write*, and *adr* lines for updating the TLB entries are not drawn.

10T and 9T implementations of the CAM cell are shown in Figure 9.48. The cells consist of a normal SRAM cell with additional transistors to perform the match. Multiple CAM cells in the same word are tied to the same matchline. The matchline is either precharged or pulled high as a distributed pseudo-nMOS gate. The key is placed on the bitlines. In Figure 9.48(a), if the key and the value stored in the cell differ, the matchline will be pulled down. Only if all of the key bits match all of the bits stored in the word of memory will the matchline for that word remain high. The key can contain a "don't care" by setting both *bit* and *bit_b* low. Figure 9.49 shows a layout of this cell in a $56 \times 43\ \lambda$ area; CAMs generally have about twice the area of SRAM cells. Figure 9.48(b) shows another CAM cell design with one fewer transistor. *N1* and *N2* perform an XOR of the key and cell data. If the values disagree, *N3* is turned on to pull down the wordline. However, the gate of *N3* sees a degraded high logic level.

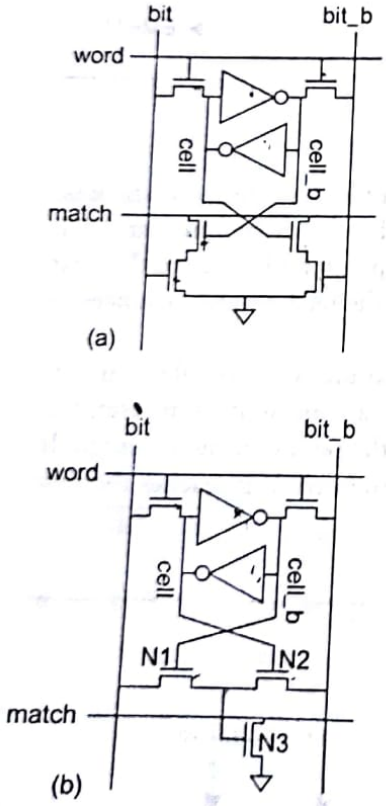


FIG 9.48 CAM cell implementations

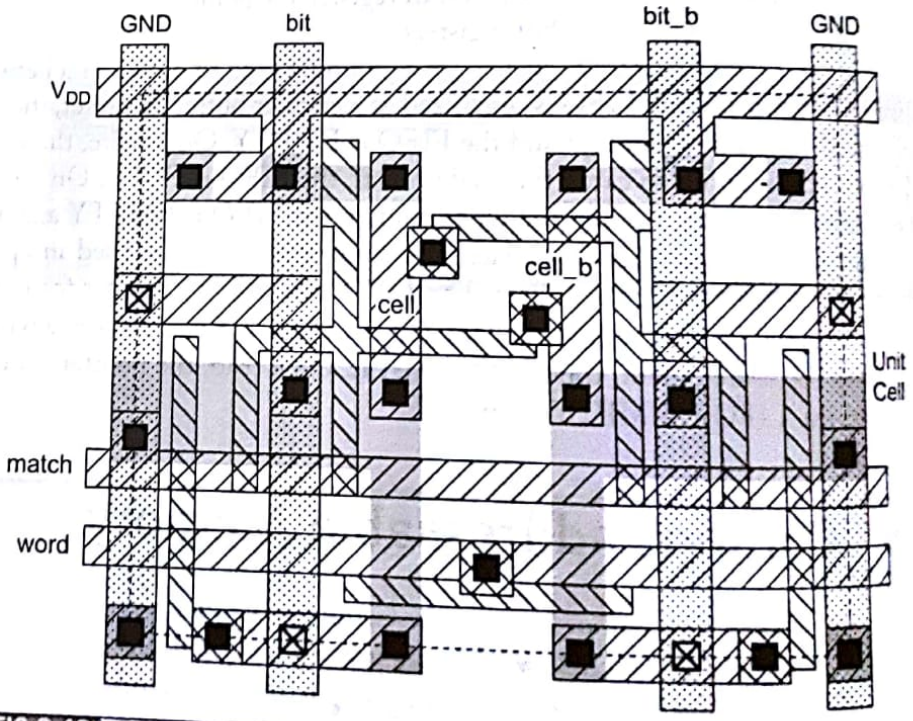


FIG 9.49 CAM cell layout. Color version given at the beginning of the book.

Figure 9.50 shows a complete 4×4 CAM array. Like an SRAM, it consists of an array of cells, a decoder, and column circuitry. However, each row also produces a dynamic matchline. The matchlines are precharged with the clocked pMOS transistors. The *miss* signal is produced with a distributed pseudo-nMOS NOR.

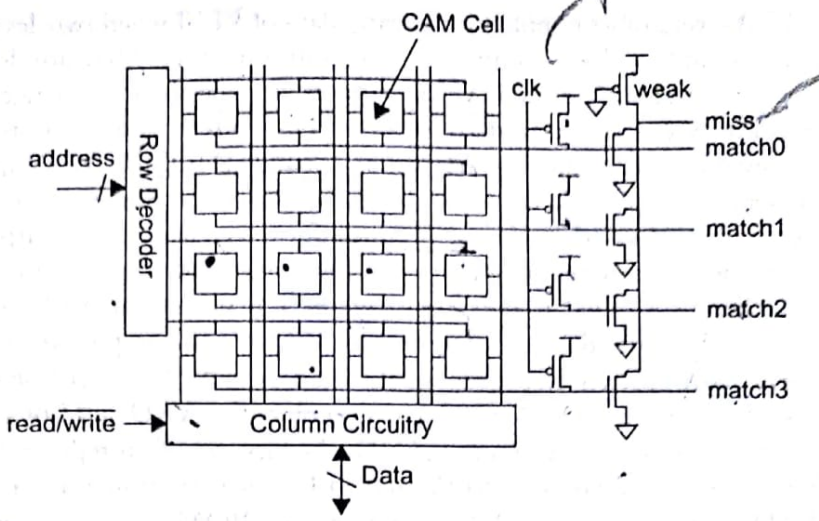


FIG 9.50 4 × 4 CAM array

When the matchlines are used to access a RAM, the monotonicity problem must be considered. Initially, all the matchlines are high. During CAM operation, the lines pull down, leaving at most one line asserted to indicate which row contains the key. However, the RAM requires a monotonically rising wordline. Figure 9.51 refines Figure 9.47 with strobed AND gates driving the wordlines as early as possible after the matchlines have settled. The strobe can be timed with an inverter chain or replica delay line in much the same way that the sense amplifier clock for a SRAM was generated in Section 9.2.3. As usual, self-timing margin must be provided so the circuit operates correctly across all design corners.

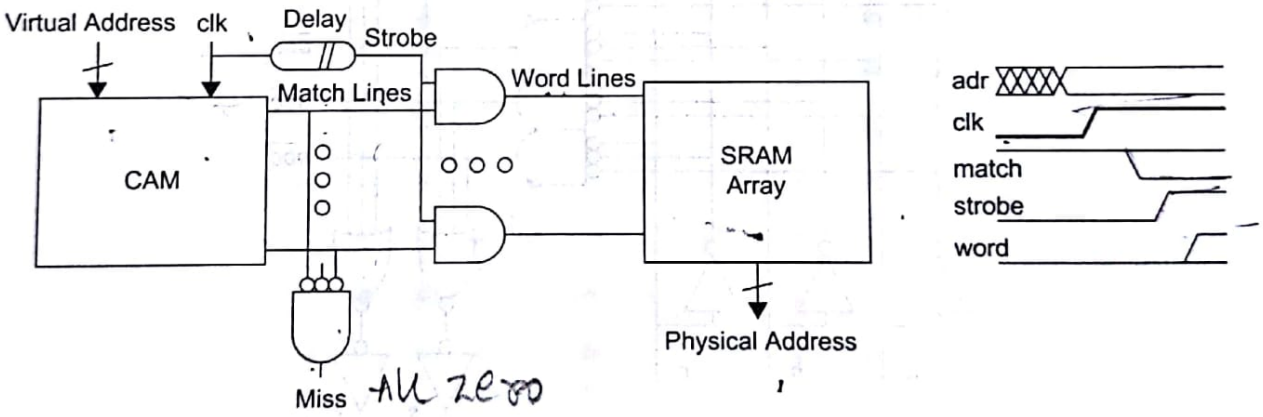


FIG 9.51 Refined TLB path with monotonic wordlines

Large CAMs can use many of the same techniques as large RAMs, including sense amplifiers and multiple subarrays. They tend to consume relatively large amounts of power because the bitlines and matchlines may all transition during a parallel search. [Miyatake01] describes a design of a mid-sized CAM using a pMOS match-line driver to reduce the swing of the matchlines and save power.

9.7 Programmable Logic Arrays

A *programmable logic array* (PLA) provides a regular structure for implementing combinational logic specified in *sum-of-products canonical form*. If outputs are fed back to inputs through registers, PLAs also can form