UNIT VI - ARRAY SUBSYSTEMS

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 1. *Programmable Logic Arrays* (PLAs) perform logic rather than storage functions

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.

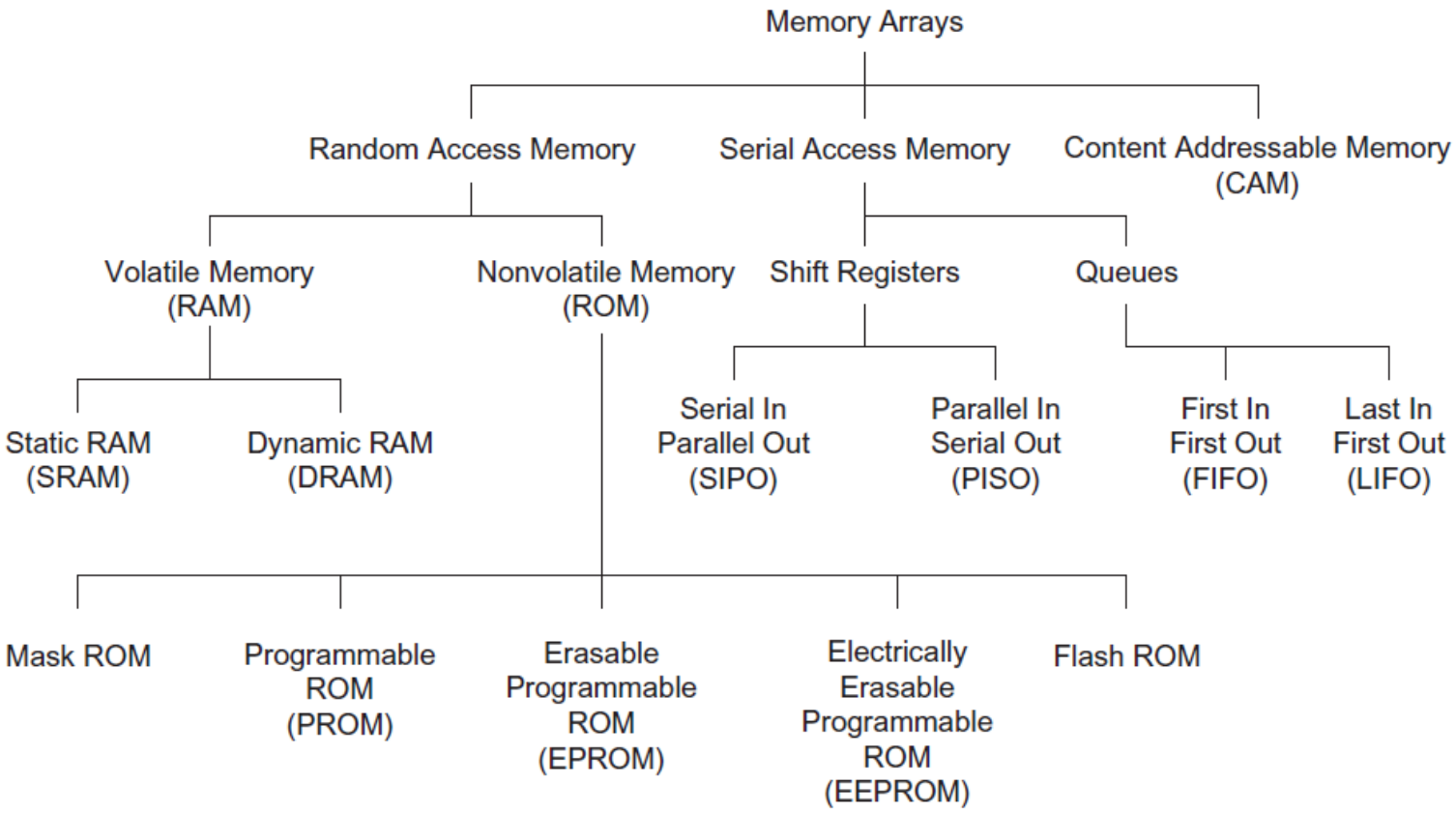

FIGURE 1 Categories of memory arrays

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 memory array contains 2^n *words* of 2^m bits each. Each bit is stored in a memory cell. Figure 2 shows the organization of a small memory array containing 16 4-bit words ($n = 4$, $m = 2$). Figure 2(a) shows the simplest design with one row per word and one column per bit. The row decoder uses the address to activate one of the rows by asserting the wordline. 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 circuitry may contain amplifiers or buffers to sense the data. A typical memory array may have thousands or millions of words of only 8–64 bits each, which would lead to a tall, skinny layout 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. Figure 12.2(b) shows a two-way fold ($k = 1$) with eight rows and eight columns. The column decoder controls a multiplexer in the column circuitry to select 2^m bits from the row as the data to access. Larger memories are generally built from multiple smaller subarrays so that the wordlines and bitlines remain reasonably short, fast, and low in power dissipation.

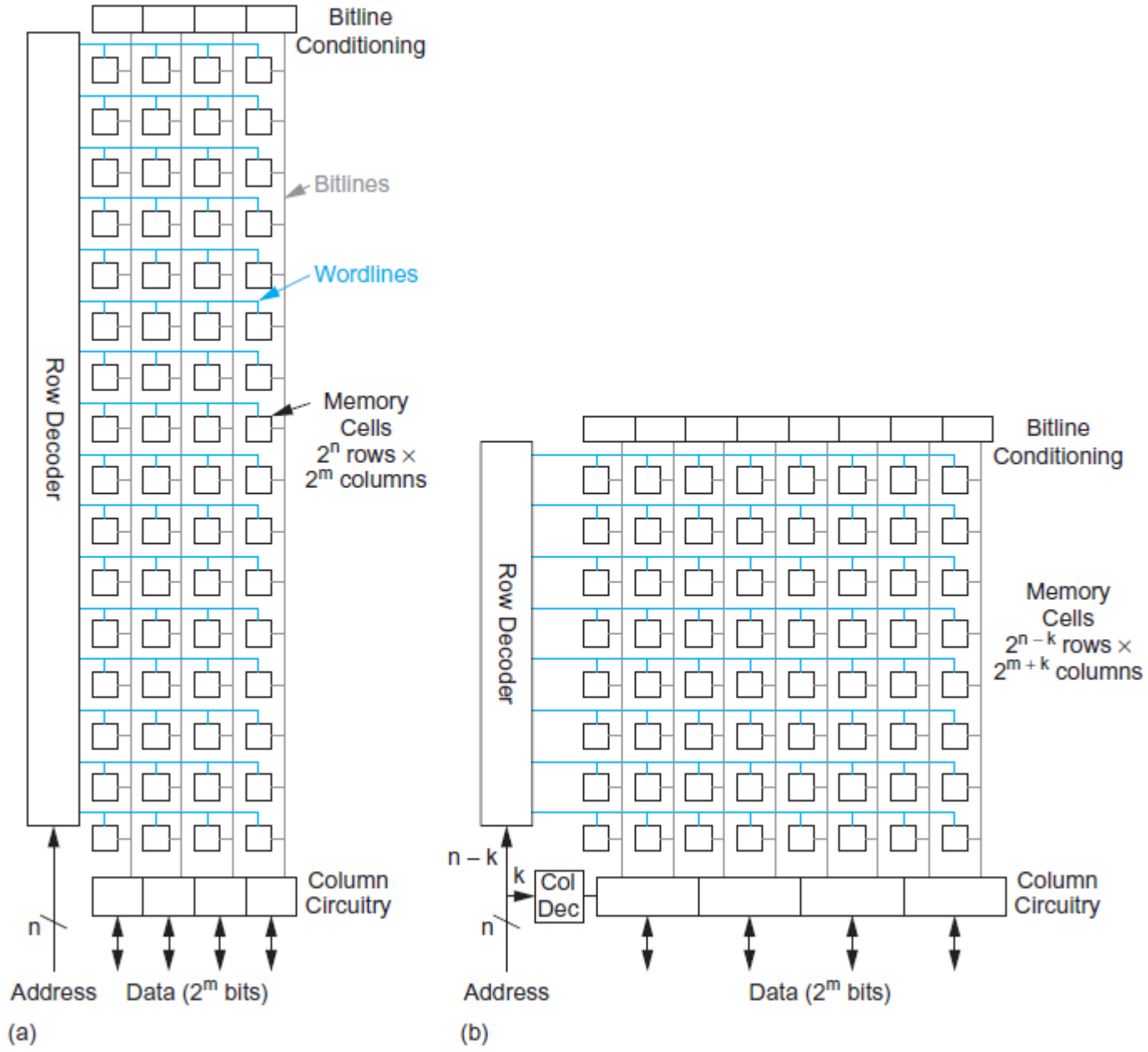

FIGURE 2 Memory array architecture

SRAM

Static RAMs use a memory cell with internal feedback that retains its value as long as power is applied. It has the following attractive properties:

- Denser than flip-flops
- Compatible with standard CMOS processes
- Faster than DRAM
- Easier to use than DRAM

For these reasons, SRAMs are widely used in applications from caches to register files to tables to scratchpad buffers. The SRAM consists of an array of memory cells along with the row and column circuitry.

SRAM Cells

A SRAM cell needs to be able to read and write data and to hold the data as long as the power is applied. An ordinary flip-flop could accomplish this requirement, but the size is quite large. Figure3 shows a standard 6-transistor (6T) SRAM cell that can be an order of magnitude smaller than a flip-flop. The 6T cell achieves its compactness at the expense of more complex peripheral circuitry for reading and writing the cells.

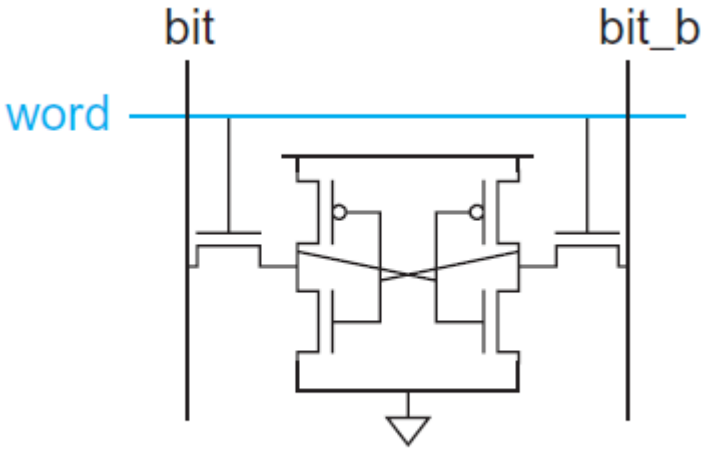

FIGURE 3 6T SRAM cell

The 6T SRAM cell contains a pair of weak cross-coupled inverters holding the state and a pair of access transistors to read or write the state. The positive feedback corrects disturbances caused by leakage or noise. The cell is written by driving the desired value and its complement onto the bitlines, *bit* and *bit_b*, then raising the wordline, *word*. The new data overpowers the cross-coupled inverters. It is read by precharging the two bitlines high, then allowing them to float. When *word* is raised, *bit* or *bit_b* pulls down, indicating the data value. The central challenges in SRAM design are minimizing its size and ensuring that the circuitry holding the state is weak enough to be overpowered during a write, yet strong enough not to be disturbed during a read.

SRAM operation is divided into two phases. In phase 2, the SRAM is precharged. In phase 1, the SRAM is read or written.

Read Operation

Figure.4 shows a SRAM cell being read. The bitlines are both initially floating high. Without loss of generality, assume Q is initially 0 and thus Q_b is initially 1. Q_b and *bit_b* both should remain 1. When the wordline is raised, *bit* should be pulled down through *driver* and *access* transistors $D1$ and $A1$.

At the same time *bit* is being pulled down, node *Q* tends to rise. *Q* is held low by *D*1, but raised by current flowing in from *A*1. Hence, the driver *D*1 must be stronger than the access transistor *A*1. Specifically, the transistors must be ratioed such that node *Q* remains below the switching threshold of the *P2*/*D2* inverter. This constraint is called *read stability*. Waveforms for the read operation are shown in Figure 12.4(b) as a 0 is read onto *bit*. Observe that *Q* momentarily rises, but does not glitch badly enough to flip the cell.

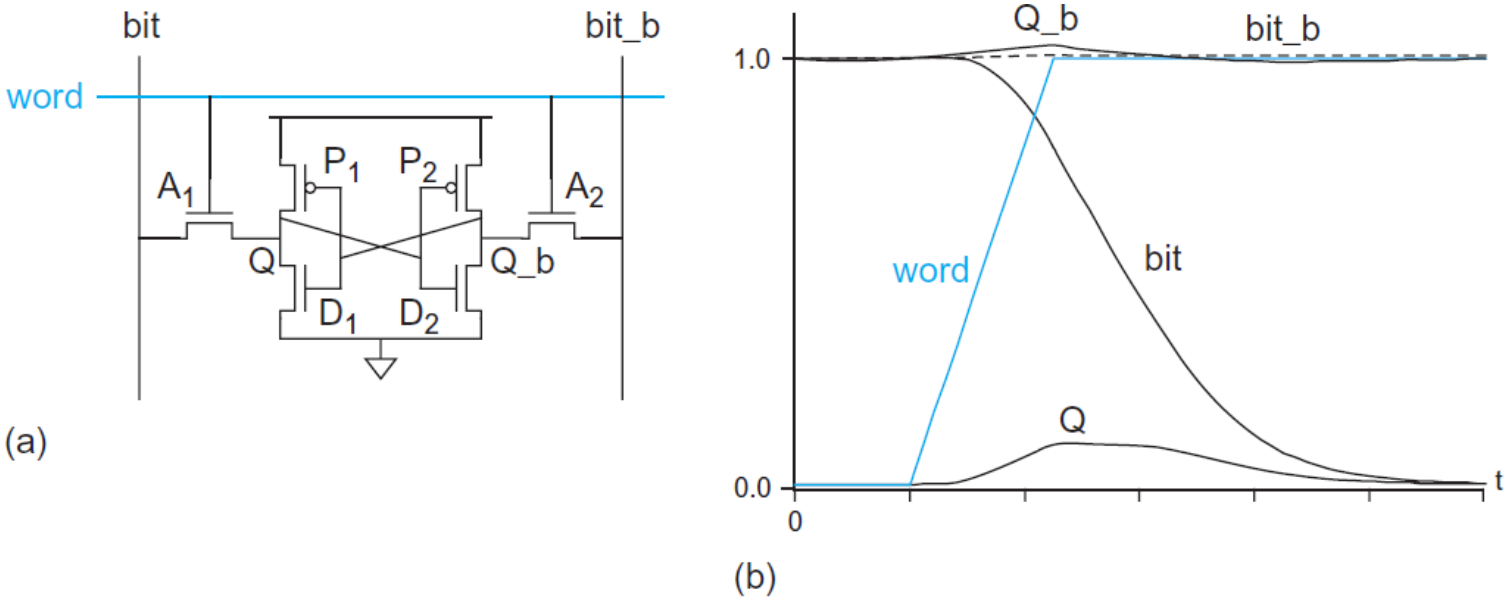

FIGURE .4 Read operation for 6T SRAM cell

Figure .5 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.

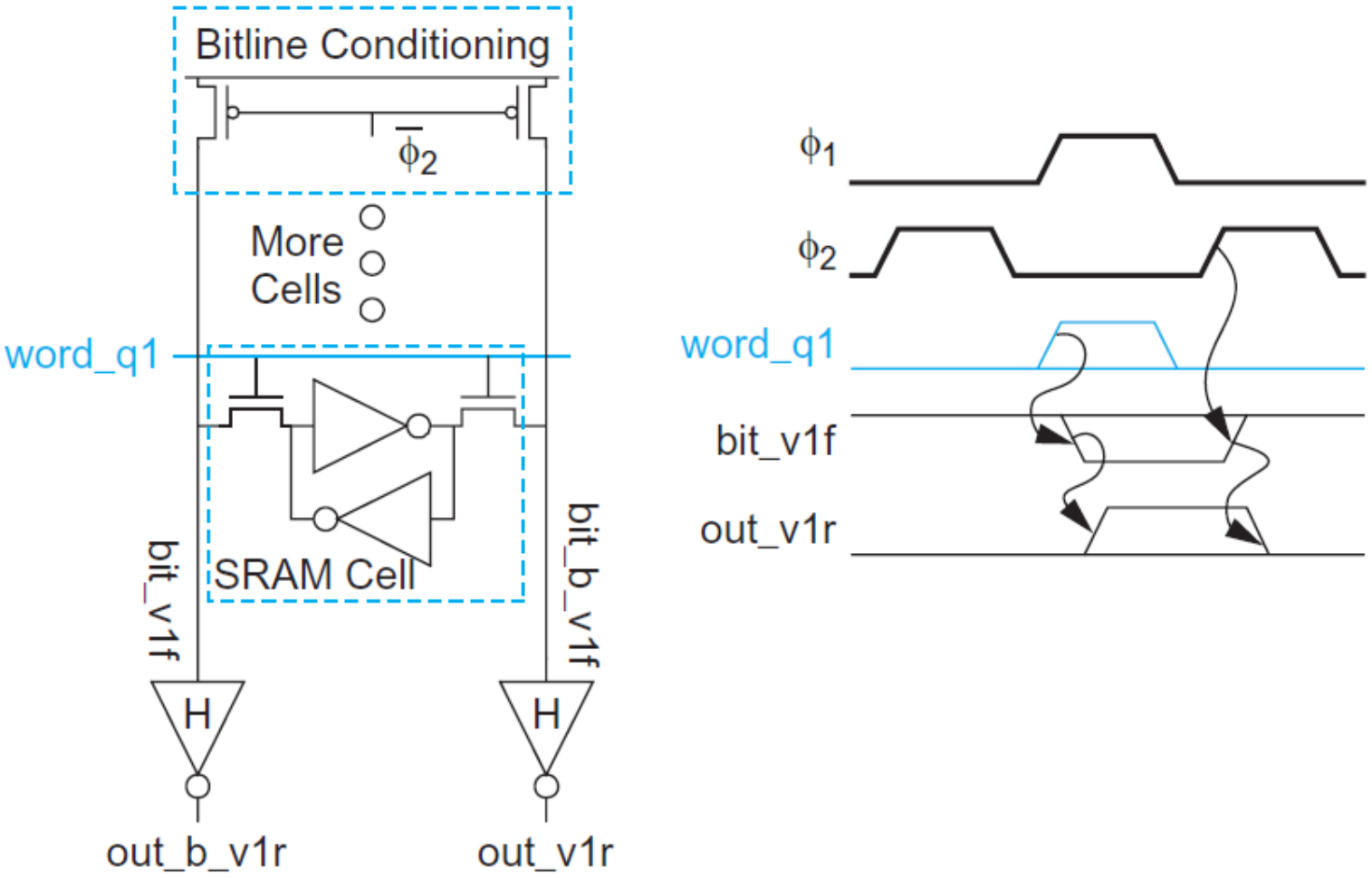

FIGURE.5 SRAM column read

Write Operation

Figure .6 shows the SRAM cell being written. Again, assume Q 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 Q high through $A1$. Hence, the cell must be written by forcing Q_b low through $A2$. $P2$ opposes this operation; thus, $P2$ must be weaker than $A2$ so that Q_b can be pulled low enough. This constraint is called *writability*. Once Q_b falls low, $D1$ turns OFF and $P1$ turns ON, pulling Q high as desired.

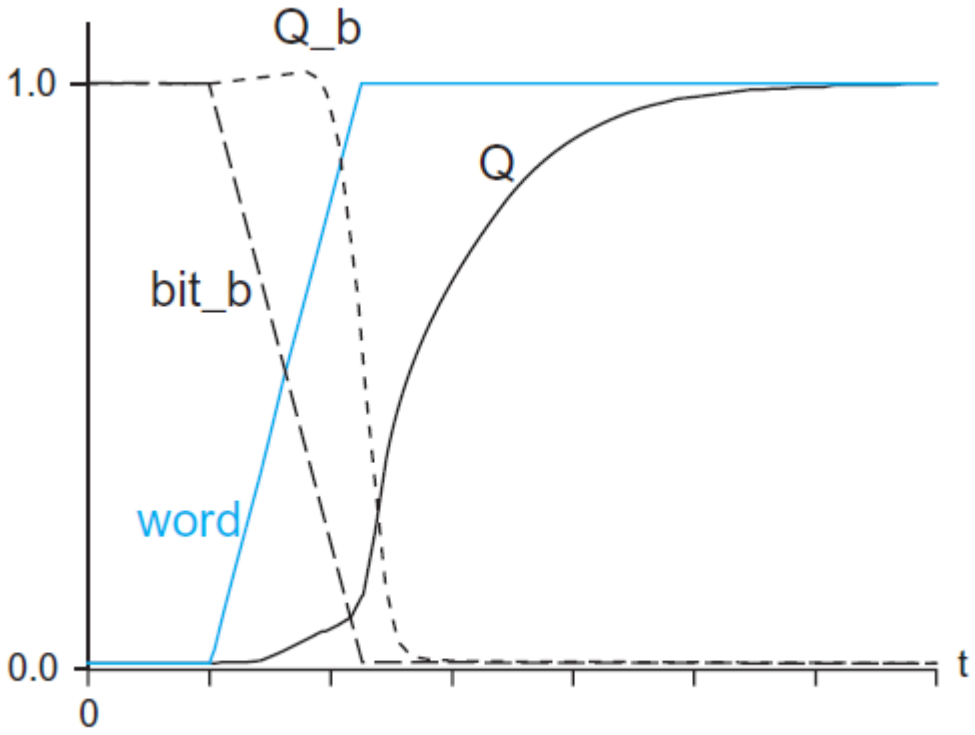

FIGURE .6 Write operation for 6T SRAM cell

Figure.7(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.7(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. Some arrays use tristate write drivers to improve writability by actively driving one bitline high while the other is pulled low.

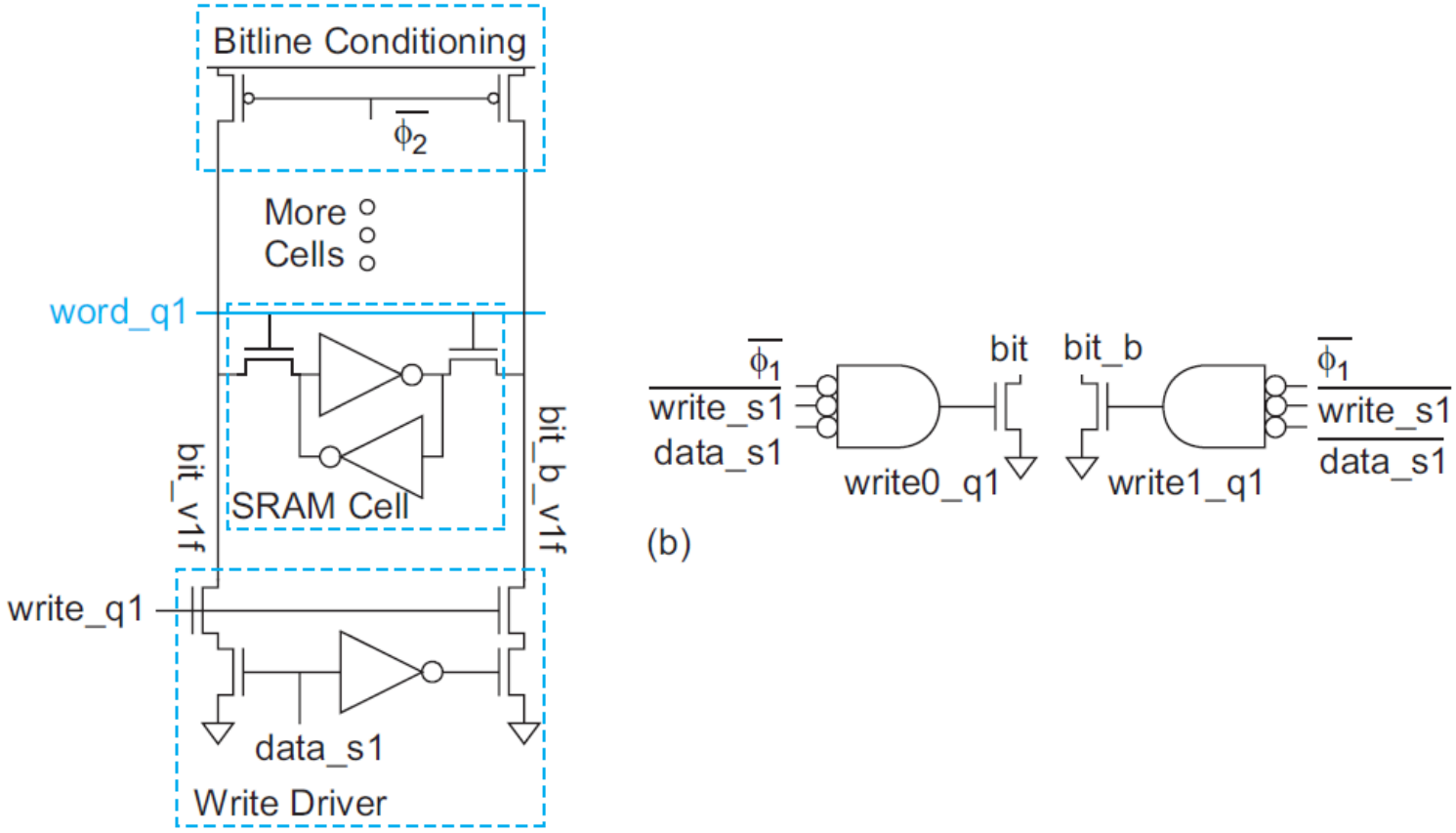

FIGURE.7 SRAM column write

Cell Stability

To ensure both read stability and writability, the transistors must satisfy ratio constraints. The nMOS pulldown transistor in the cross-coupled inverters must be strongest. The access transistors are of intermediate strength, and the pMOS pullup transistors must be weak. To achieve good layout density, all of the transistors must be relatively small. For example, the pulldowns could be 8/2 λ, the access transistors 4/2, and the pullups 3/3. The SRAM cells must operate correctly at all voltages and temperatures despite process variation.

The stability and writability of the cell are quantified by the hold margin, the read margin, and the write margin, which are determined by the static noise margin of the cell in its various modes of operation. A cell should have two stable states during hold and read operation, and only one stable state during write. The *static noise margin* (SNM) measures how much noise can be applied to the inputs of the two cross-coupled inverters before a stable state is lost (during hold or read) or a second stable state is created (during write).

Alternative Cells

Figure.18 shows a *dual-port* SRAM cell using eight transistors to provide independent read and write ports. For a write, the data and its complement are applied to the *wbl* and *wbl_b* bitlines and the *wwl* wordline is asserted. For a read, the *rbl* bitline is precharged, then the *rwl* wordline is asserted. Notice that read operation does not backdrive the state nodes through the access transistor, so read margin is as good as hold margin.

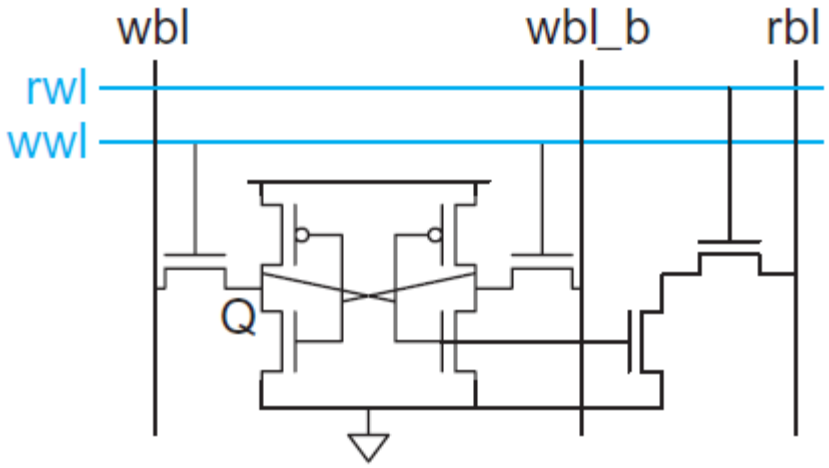

FIGURE.18 8T dual-port SRAM cell

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 a factor of 10–20 greater density (bits/cm^2) than high-performance SRAM built in a standard logic process, but they also have much higher latency.

A 1-transistor (1T) dynamic RAM cell consists of a transistor and a capacitor, as shown in Figure.41(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.41(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.'

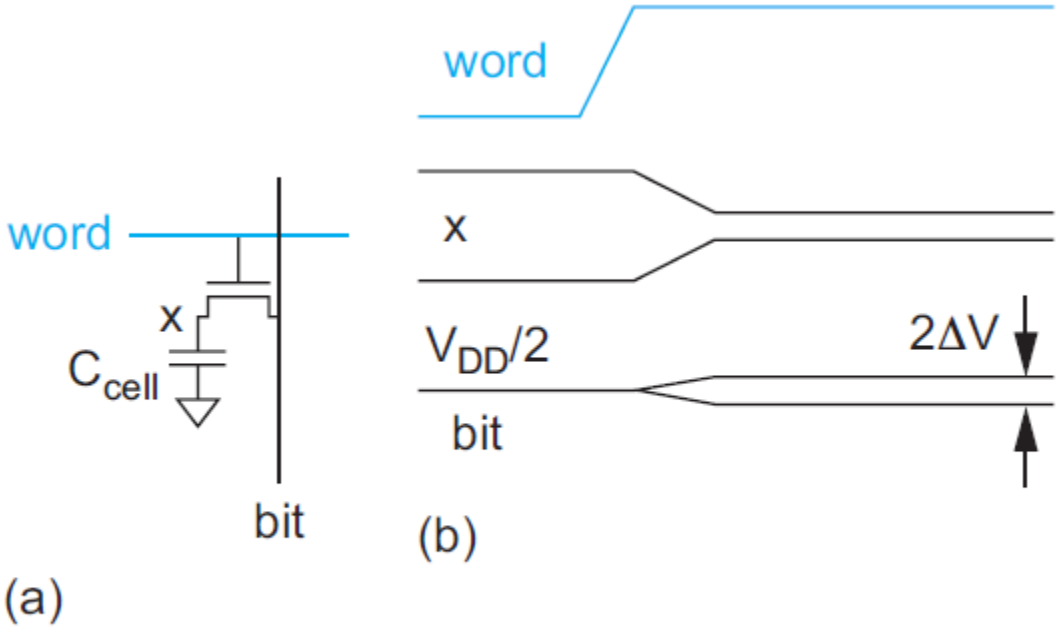

FIGURE.41 1T DRAM cell read operation

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

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

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. 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 .42 shows a cross-section and 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.

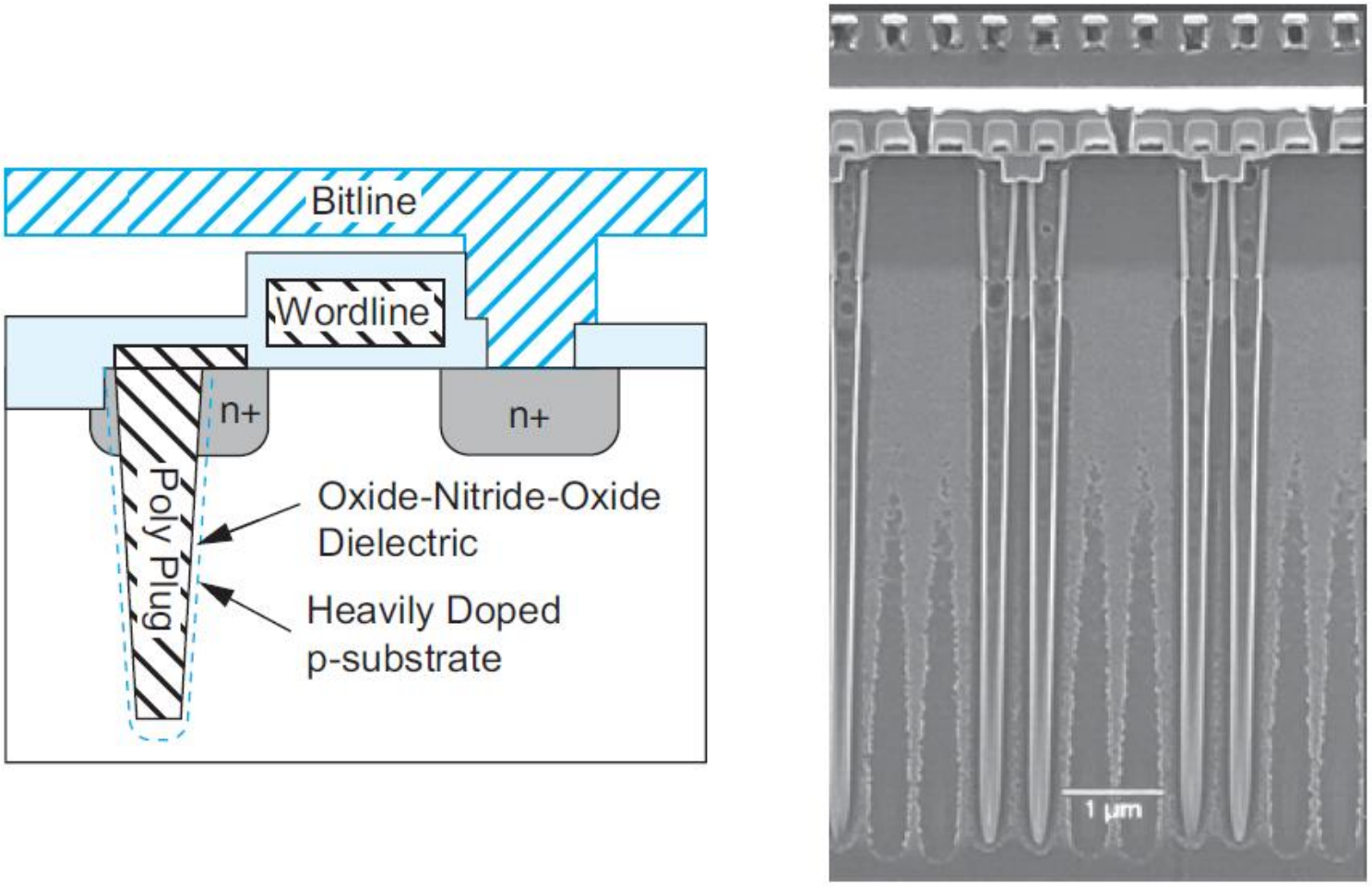

FIGURE.42 Trench capacitor

Subarray Architectures

Large DRAMs are divided into multiple subarrays. The subarray size represents a trade-off between density and performance. Larger subarrays amortize the decoders and sense amplifiers across more cells and thus achieve better array efficiency. 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 43.

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 tiny. 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 compact to fit the tight pitch of the array. The low-swing bitlines are sensitive to noise.

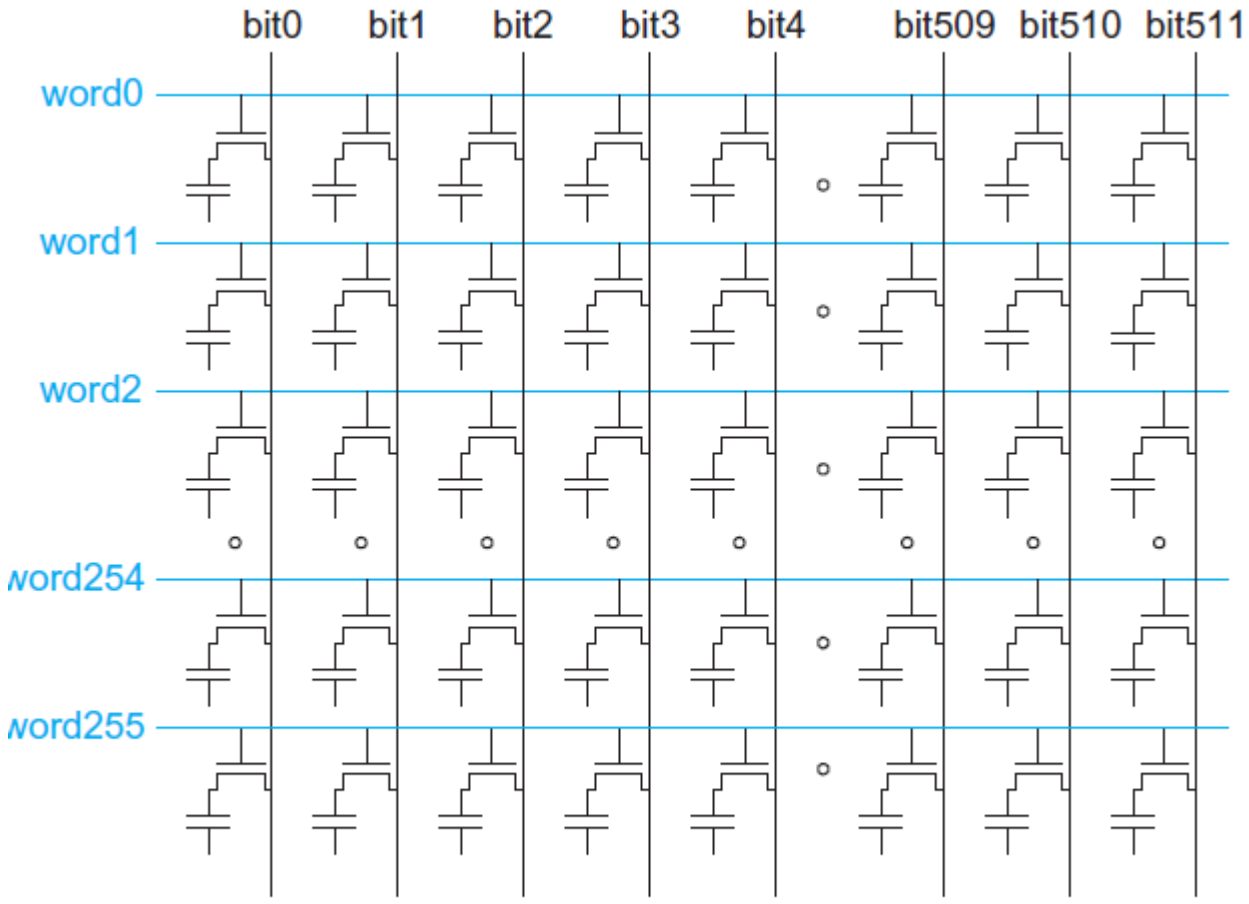

FIGURE 43 DRAM subarray

Three bitline architectures, *open*, *folded*, and *twisted*, offer different compromises between noise and area.

Early DRAMs (until the 64-kbit generation) used the open bitline architecture shown in Figure 44. 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-capacity DRAM.

In Figure .45 folded bitline 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.

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 solves this problem by swapping the positions of the folded bit-lines part way along the array in much the same way as SRAM bitlines were twisted.
The twists cost a small amount of extra area within the array.

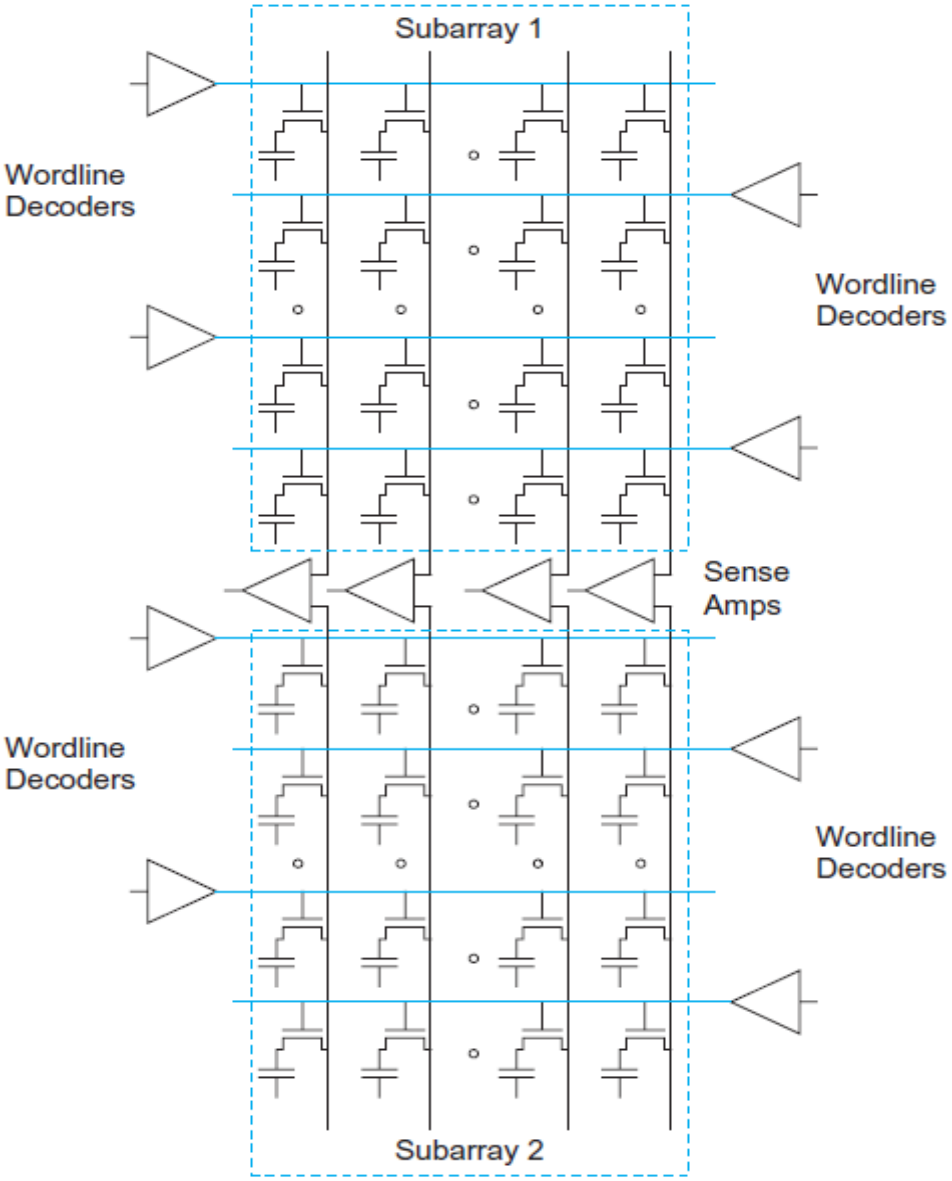

FIGURE .44 Open bitlines

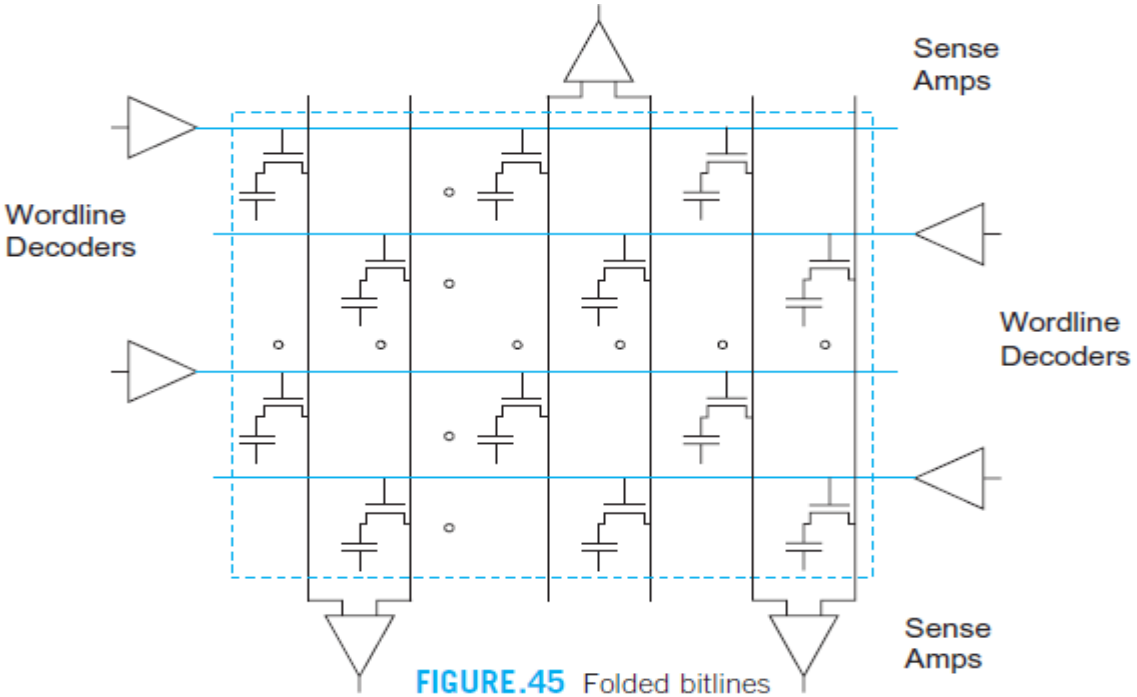

FIGURE .45 Folded bitlines

Embedded DRAM

Memories now account for half or more of the area of many chips. Replacing the SRAM with a denser DRAM could save a good fraction of this area and reduce manufacturing costs. Unfortunately, DRAM processes are designed for low leakage using high thresholds and thick oxides. Attempts to incorporate logic onto DRAM processes have been uninspiring. Standard logic processes lack the specialized capacitor and stacked contact structures to build extremely high density DRAM. However, some foundries offer an embedded DRAM (eDRAM) option with a dense capacitor structure. For example, the IBM 65 nm process supports a 0.127 μm^2 eDRAM cell using a trench capacitor. The cell is four times denser than SRAM in the same process but is not as fast.

Alternatively, DRAM can be constructed in a standard logic process using additional transistors in place of the capacitor. Figure .51 shows some 3T and 4T DRAM *gain cells*. These cells store a value on the gate capacitance of a transistor. The read operation involves an active transistor rather than simple charge sharing, so they can produce a stronger signal. Early DRAMs used these cells, but they were superseded by Dennard's invention of the 1T cell at IBM in 1968. They might become relevant again as technology and power supplies continue to scale.

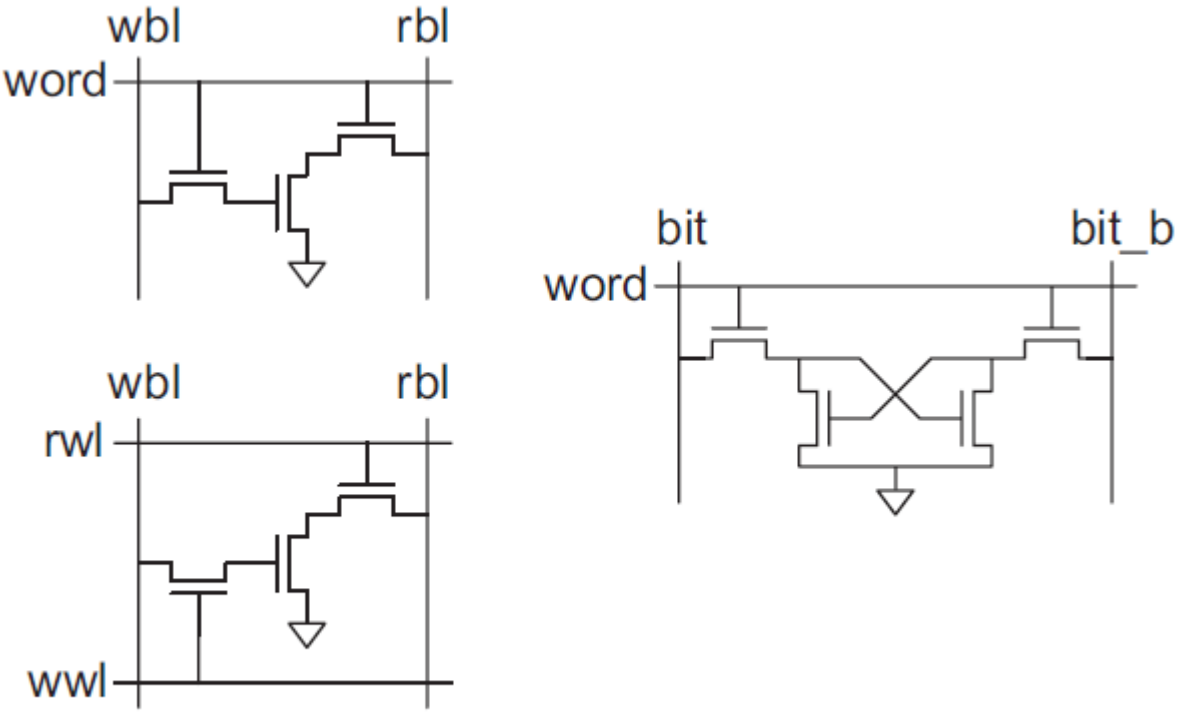

FIGURE .51 3T and 4T DRAM cells

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. Commercial ROMs are normally dynamic, although pseudo-nMOS is simple and suffices for small structures. 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 pullup transistors after the column multiplexer.

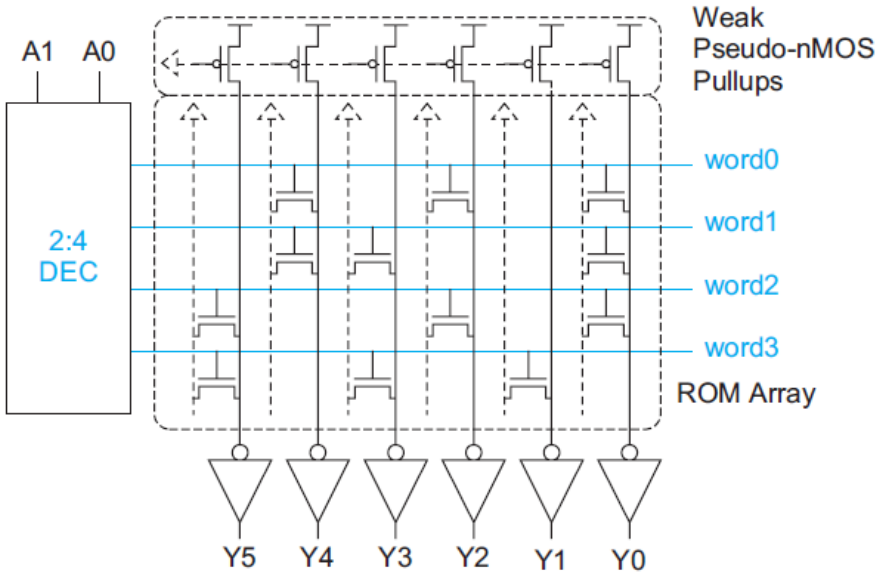

FIGURE .52 Pseudo-nMOS ROM

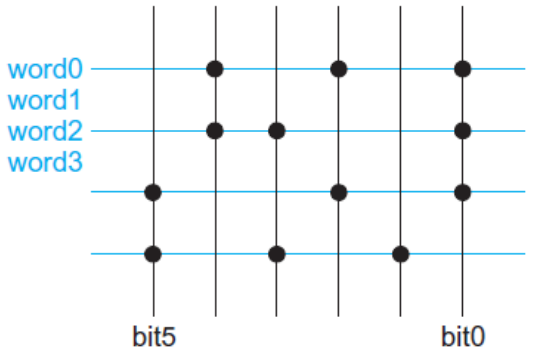

FIGURE .53 Dot diagram representation of ROM

Figure.52 shows a 4-word by 6-bit ROM using pseudo-nMOS pullups with the following contents:

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

The contents of the ROM can be symbolically represented with a dot diagram in which dots indicate the presence of 1s, as shown in Figure .53. The dots correspond to nMOS pulldown 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.

Progammable ROMs

It is often desirable for the user to be able to program or reprogram a ROM after it is manufactured. 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 burned 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 control gate and the channel, as shown in Figure.57. The floating gate is a good conductor, but it is not attached to anything. Applying a high voltage to the control gate causes electrons to jump through the thin oxide onto the floating gate through the processes called *Fowler-Nordheim (FN) 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.

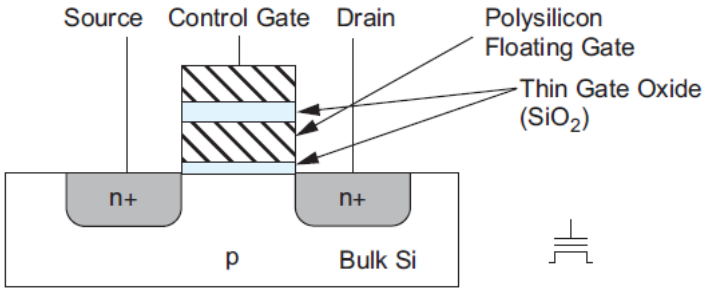

FIGURE .57 Cross-section of floating gate nMOS transistor

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 achieve this versatility, so Flash has become the most economical form of convenient nonvolatile storage.

NAND ROMs

Figure 12.58 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.

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.

Flash

Flash memory has become tremendously popular because of its nonvolatile storage and exceptionally low cost per bit. For example, Flash memory cards are widely used in digital cameras to store hundreds of high-resolution images. 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. Most of the Flash market has become a commodity business driven almost entirely by cost, with performance and even reliability being secondary considerations. This section summarizes the principles of Flash operation.

Most stand-alone Flash memory uses the NAND architecture to minimize bit cell size and cost. NAND Flash memories are divided into *blocks*, which in turn are made of *pages*. The memory is written one page at a time and erased one block at a time. For example, a conventional NAND flash memory might be made of 8 KB (64 Kb) blocks, each of which contain sixteen 512 B (4 Kb) pages.

Flash uses floating gate transistors as memory cells. The charge on the floating gate determines the threshold of the transistor and indicates the state of the cell. A negative threshold represents a logic '1' and a positive threshold represents a logic 0.'

In NAND Flash, the floating gate transistors are connected in series to form *strings*. Figure .60 shows the organization of a string, page, and block in a simple Flash memory. Each string consists of 16 cells, a *string select* transistor, and a *ground select* transistor all connected in series and attached to the bitline.

The control gate of each cell is connected to a wordline. The array contains one column for each bit in a page. Each column contains one string per block. The number of cells in the string determines the number of pages per block.

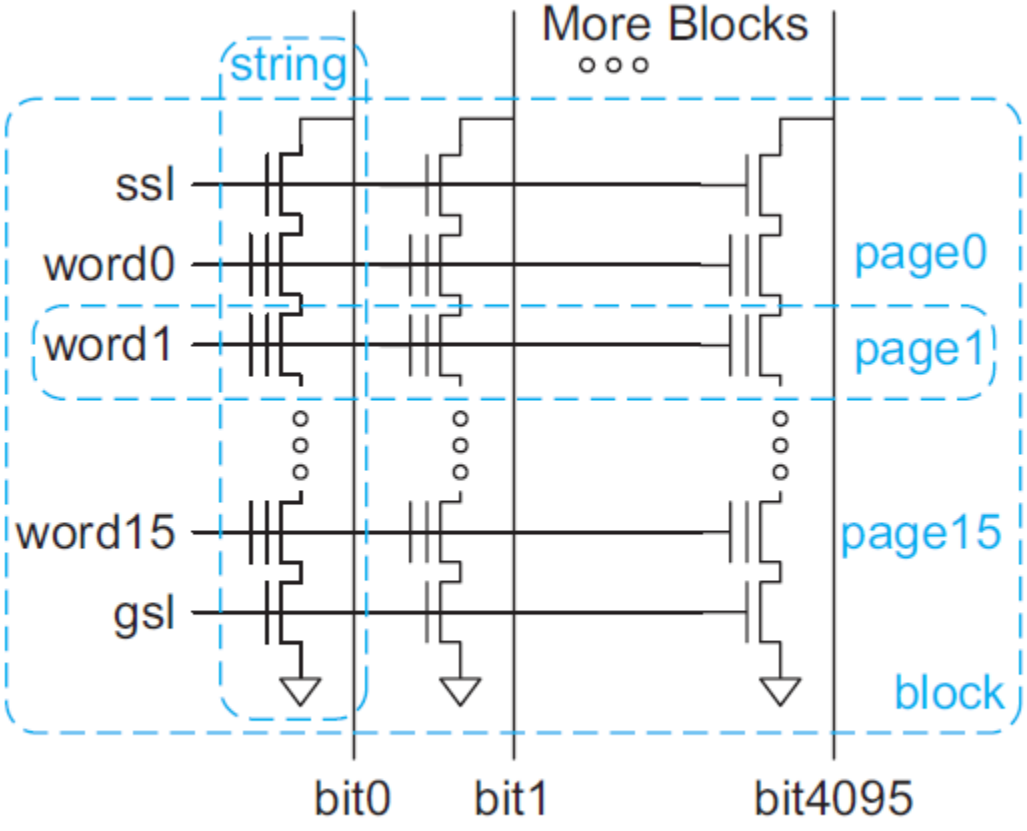

FIGURE .60 NAND Flash string

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.

A *shift register* is commonly used in signal-processing applications to store and delay data. Figure.63(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.63(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 .63(c).

The *tapped delay line* is a shift register variant that offers a variable number of stages of delay. Figure .64 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 .65(a) shows a 4-stage Serial In Parallel Out (SIPO) memory and Figure.65(b) shows a 4-stage Parallel In Serial Out (PISO) memory. These are also often useful in signal processing and communications systems.

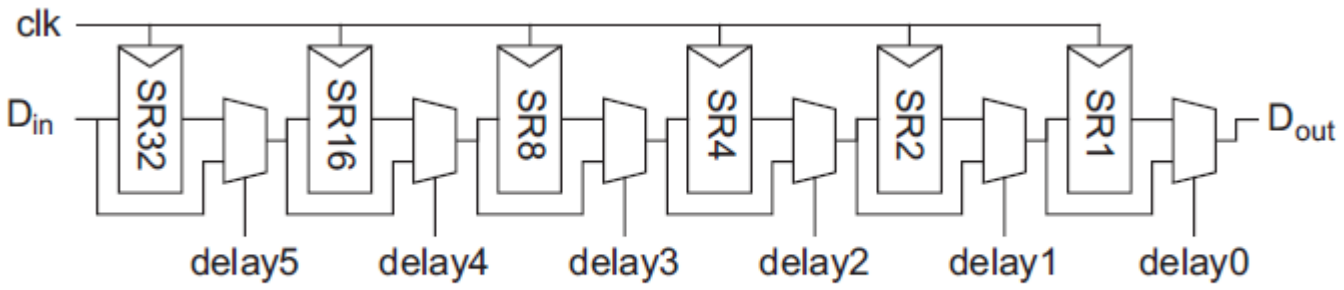

FIGURE.64 Tapped delay line

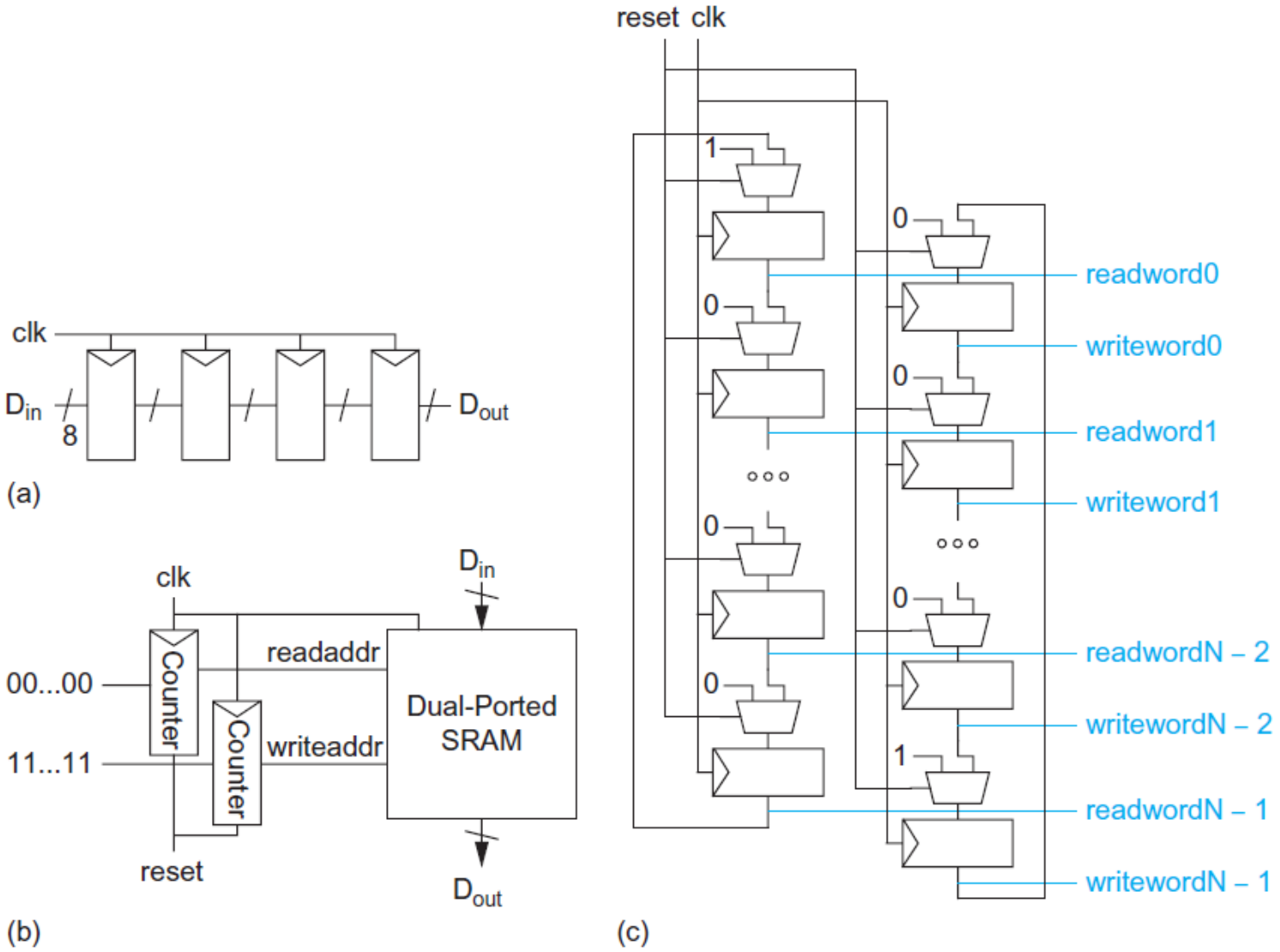

FIGURE .63 Shift registers

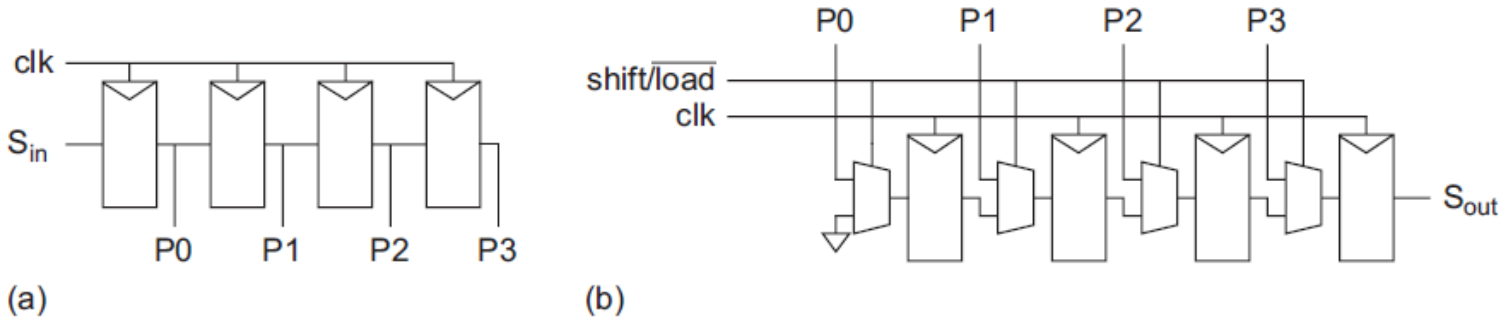

FIGURE.65 Serial/parallel memories

Queues (FIFO, LIFO)

Queues allow data to be read and written at different rates. Figure.66 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 *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.

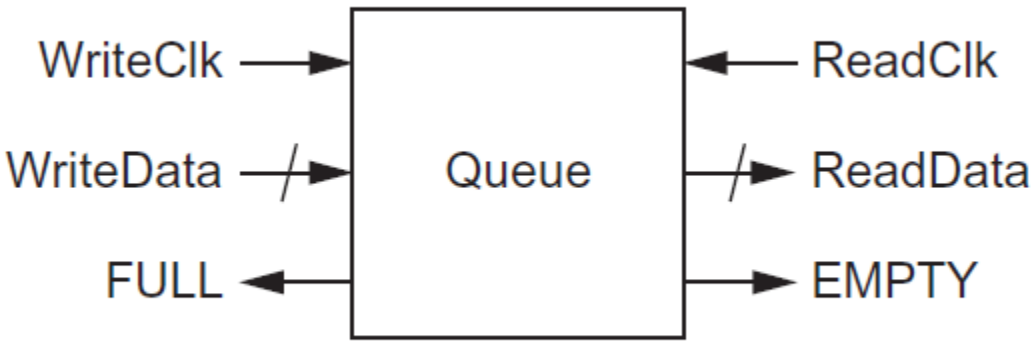

FIGURE.66 Queue

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.

Content-Addressable Memory

Figure.67 shows the symbol for a content-addressable memory (CAM). 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 matchline can serve as the wordline to access a RAM containing the associated physical address, as shown in Figure.68. 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.

Figure.69 shows a 10T CAM cell consisting 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. 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. The inside front cover shows a layout of this cell in a $56 \times 43\ \lambda$ area; CAMs generally have about twice the area of SRAM cells. Sometimes the key is provided on separate *searchlines* rather than on the bitlines to reduce the capacitance and power consumption of a search.

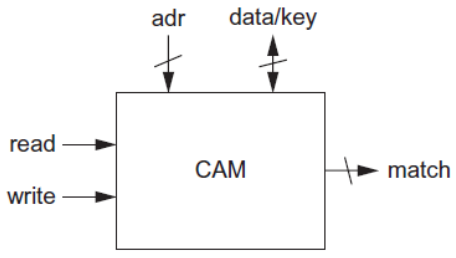

FIGURE .67 Content-addressable memory

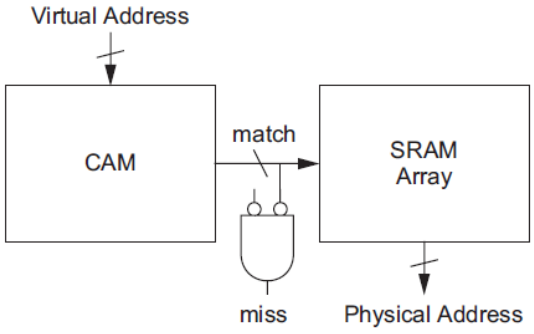

FIGURE.68 Translation Lookaside Buffer (TLB) using CAM

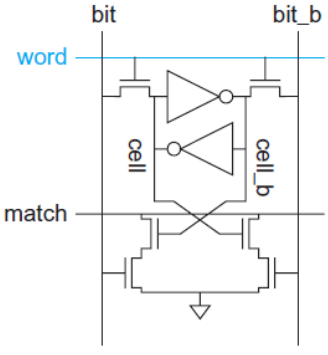

FIGURE.69 CAM cell implementation

Figure 12.70 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.

When the matchlines are used to access aRAM, 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 .71 refines Figure.68 with strobed AND gates driving the wordlines as early as possible after the matchlines have settled.

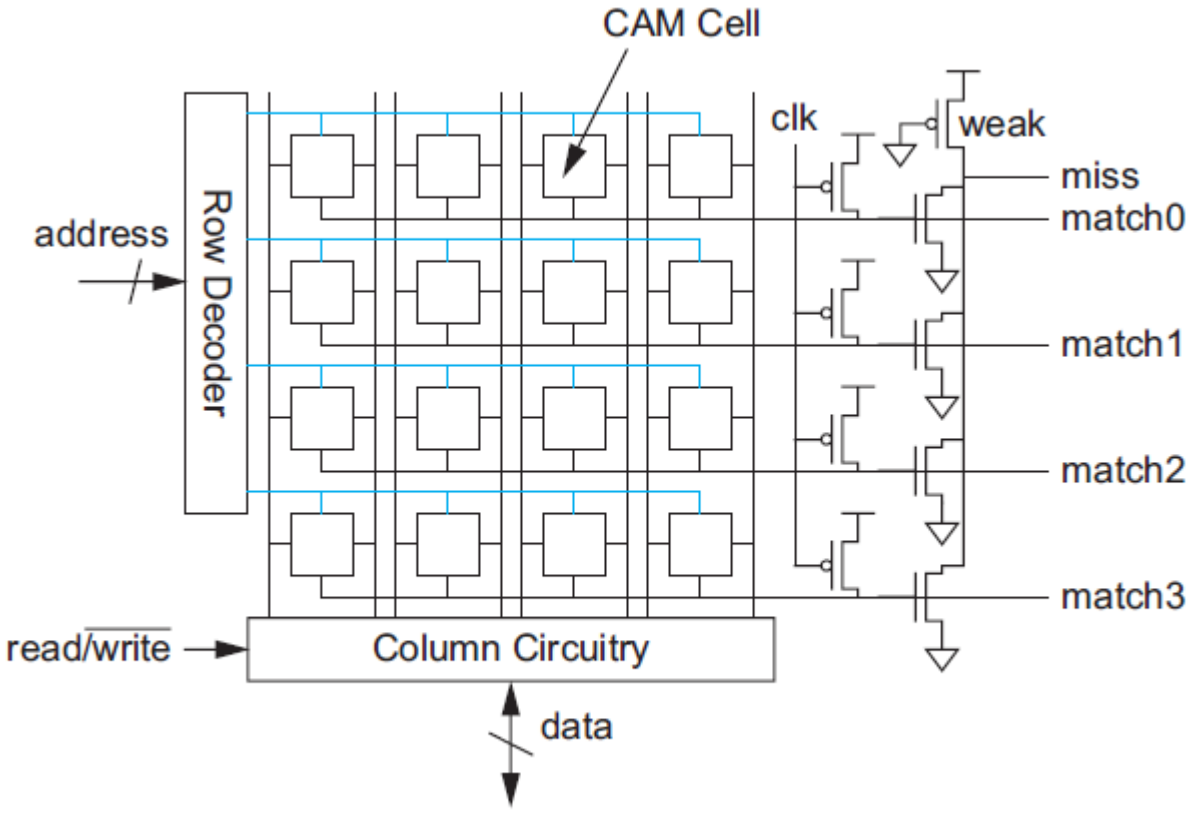

FIGURE.70 4 × 4 CAM array

The strobe can be timed with an inverter chain or replica delay line in much the same way that the sense amplifier clock for an SRAM was generated. As usual, self-timing margin must be provided so the circuit operates correctly across all design corners. The strobe must be deasserted before the match lines precharge.

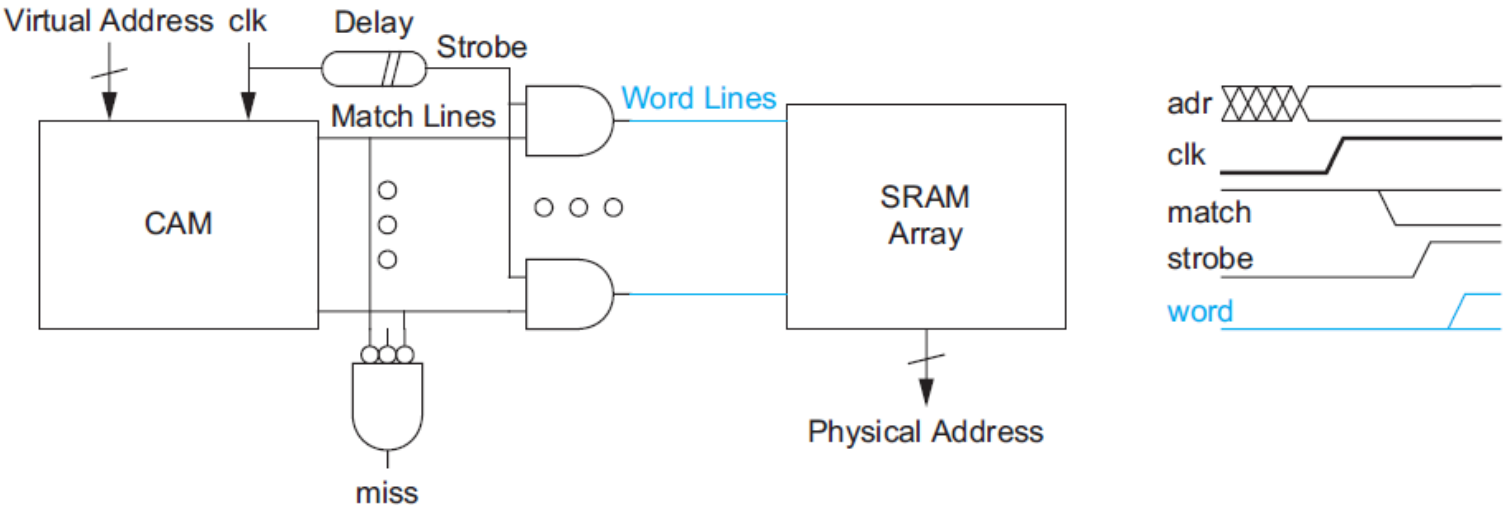

FIGURE .71 Refined TLB path with monotonic wordlines

In some applications, a CAM doesn't care about the value of certain bits. For example, a CAM used in a network router may not care about the subnet address when it is seeking to route data to the correct continent. A *ternary CAM* (TCAM) can store *X* (don't care) values as well as 0 and 1 bits. Figure.72 shows a TCAM cell using two bits of state to store the three values. This cell also illustrates separating the search lines from the bitlines. When $A = 1$ and $B = 0$, the cell matches a 0. When $A = 0$ and $B = 1$, the cell matches a 1. When $A = 0$ and $B = 0$, the cell matches both 0 and 1.

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 matchlines are heavily loaded and have an activity factor close to 1.

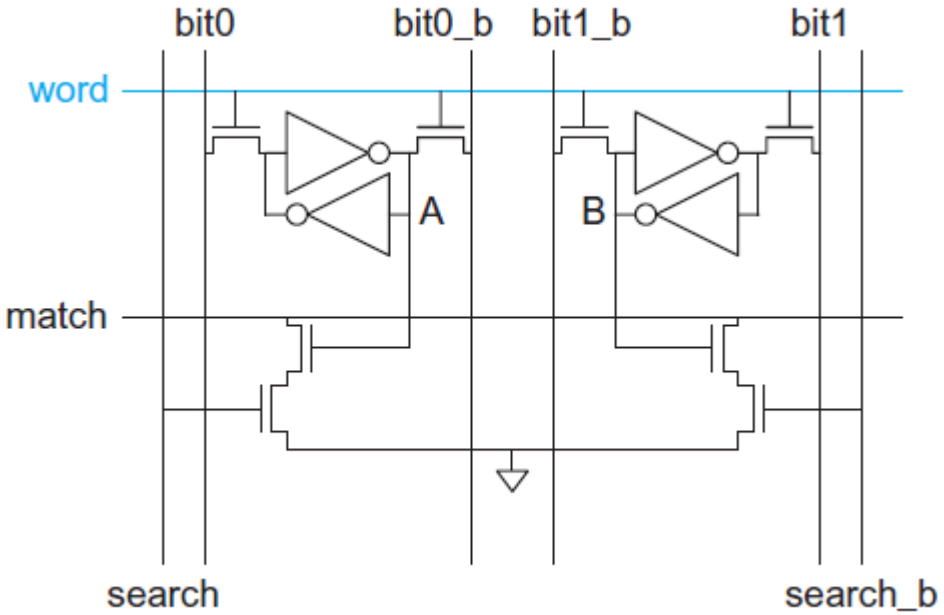

FIGURE.72 TCAM