Slide Set 10

for ENEL 353 Fall 2019

Steve Norman, PhD, PEng

Electrical & Computer Engineering
Schulich School of Engineering
University of Calgary

Fall Term, 2019

Contents

Important notes about exam topics

Counters

Shift registers

Memory arrays

Bit cells, wordlines, and bitlines

ROM circuits: dot notation, ROM-based logic

Implementation of ROM with NMOS transistors

Programmable ROM circuits

How RAM and EEPROM circuits work

PLAs: Programmable logic arrays

Outline of Slide Set 10

Important notes about exam topics

Counters

Shift registers

Memory arrays

Bit cells, wordlines, and bitlines

ROM circuits: dot notation, ROM-based logic

Implementation of ROM with NMOS transistors

Programmable ROM circuits

How RAM and EEPROM circuits work

PLAs: Programmable logic arrays

Important notes about exam topics

Because there aren't very many hours of lecture left, **not all** of the material in this slide set will be tested on the final exam.

Here are two topics that **were** tested on the Fall 2013 exam, but were **not** tested on the Fall 2014–Fall 2018 exams, and will **not** be tested on the Fall 2019 exam:

- shift registers
- PLAs

The material on implementation of memory cells using transistors is also **not** a Fall 2019 exam topic.

Outline of Slide Set 10

Important notes about exam topics

Counters

Shift registers

Memory arrays

Bit cells, wordlines, and bitlines

ROM circuits: dot notation, ROM-based logic

Implementation of ROM with NMOS transistors

Programmable ROM circuits

How RAM and EEPROM circuits work

PLAs: Programmable logic arrays

Counters (textbook Section 5.4.1)

Counter is a name used for any kind of synchronous sequential circuit designed to move **one step per clock cycle** through some specified **counting sequence**.

There is a huge variety of kinds of counters, such as

- *N-bit binary counters*, which count from 0 up to $2^N - 1$, then return to 0;
- *decade counters*, which count from 0 up to 9, then return to 0;
- *down-counters*, which go to some maximum value on reset, then count down to 0.

We've already seen lots of counters in lectures and Problem Set 5, so it's best to spend our remaining lecture time on other topics.

Outline of Slide Set 10

Important notes about exam topics

Counters

Shift registers

Memory arrays

Bit cells, wordlines, and bitlines

ROM circuits: dot notation, ROM-based logic

Implementation of ROM with NMOS transistors

Programmable ROM circuits

How RAM and EEPROM circuits work

PLAs: Programmable logic arrays

Shift registers (textbook Section 5.4.2)

Shift register is a name used for a synchronous sequential circuit made from a **chain of flip-flops**.

In normal operation of a shift register, a typical D input to DFF is driven by the Q output of a neighbouring DFF.

Although the name **shift register** hasn't been used, shift registers have already appeared a few times in ENEL 353. For example, the divide-by-3 counter with one-hot state encoding stores its state in a shift register.

Two of the many practical applications of shift registers are **parallel-to-serial** conversion and **serial-to-parallel** conversion.

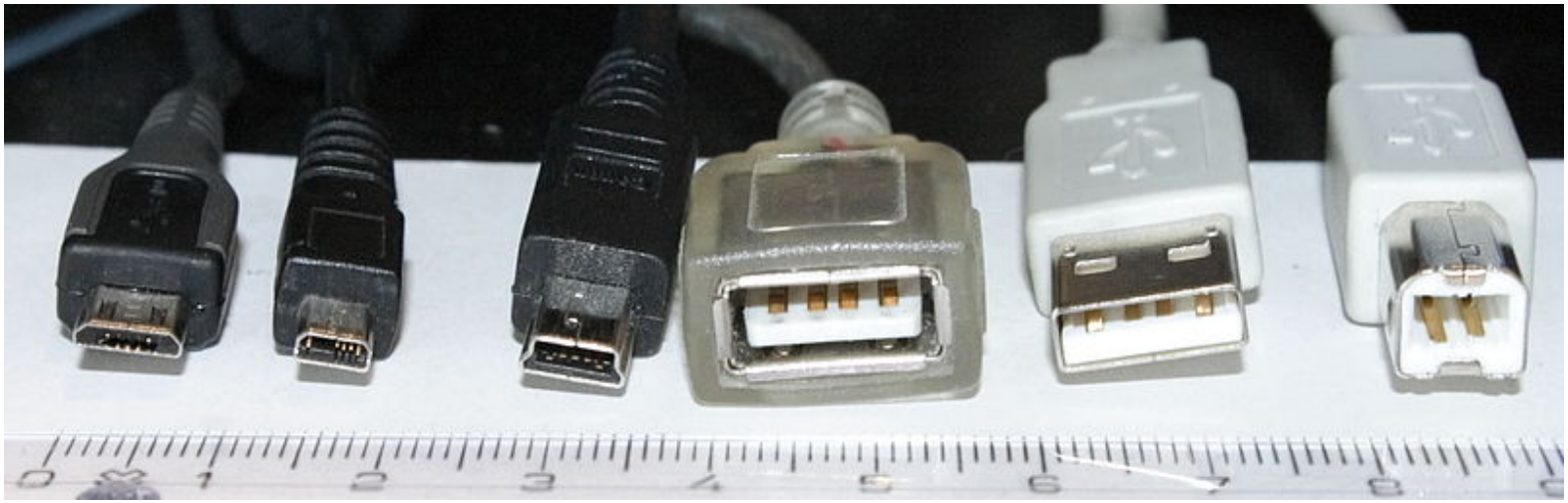
serial and *parallel*

N-bit *parallel* data communication uses *N* wires to transmit *N* bits at the same time. Most data communication within a computer processor circuit is parallel.

Serial data communication transmits one bit per time interval. Often the time interval is one system clock period.

An example of serial communication is the Universal Serial Bus (USB) system, which is very widely used to connect computers to peripherals such as printers, keyboards, and many kinds of external storage devices.

Universal Serial Bus



USB connections—version 2.0 and earlier—have 4 wires:

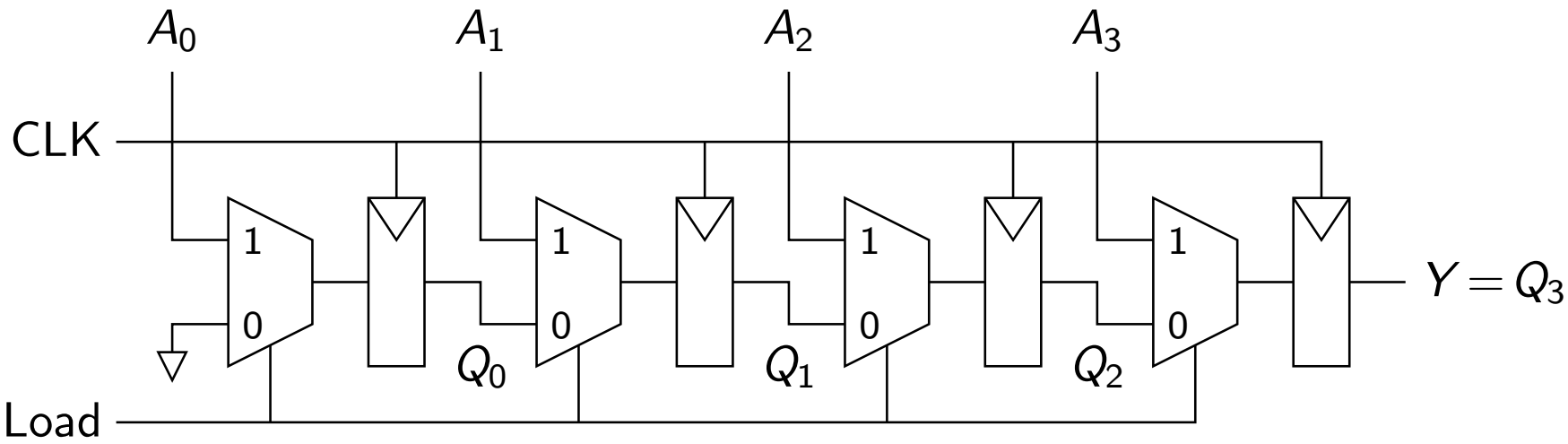
- 5V and ground for powering devices;
- two wires for serial communication—a 1 or 0 is transmitted as a **voltage difference** between these two wires.

(Public domain photo taken from Wikipedia article on USB.)

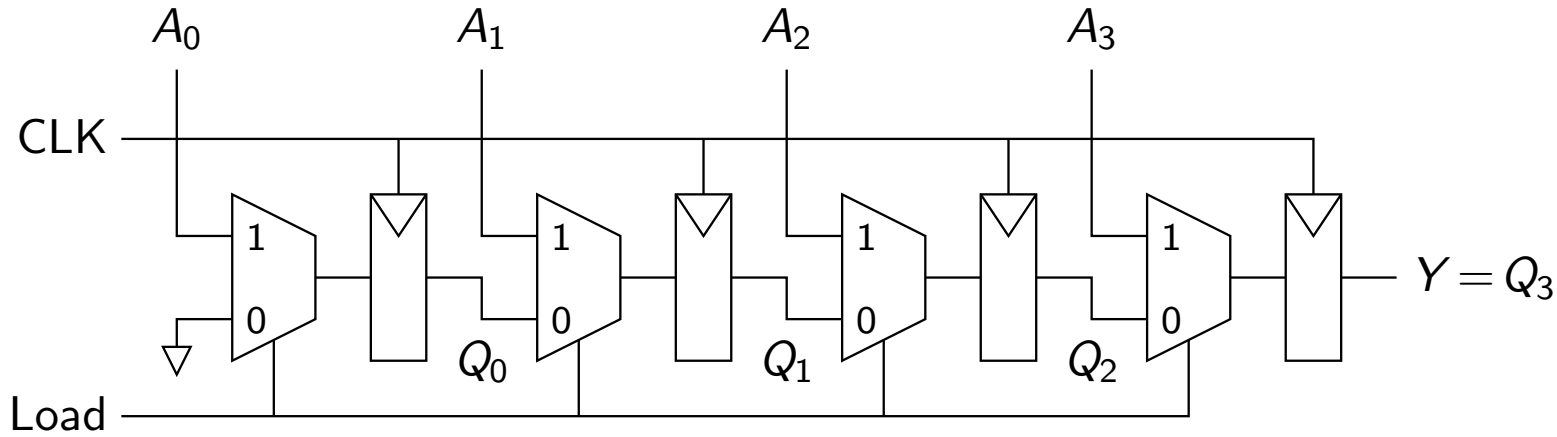
N-bit parallel-in-serial-out shift register

N bits can be copied into the shift register in a single clock cycle. The bits come out one at a time over N clock cycles.

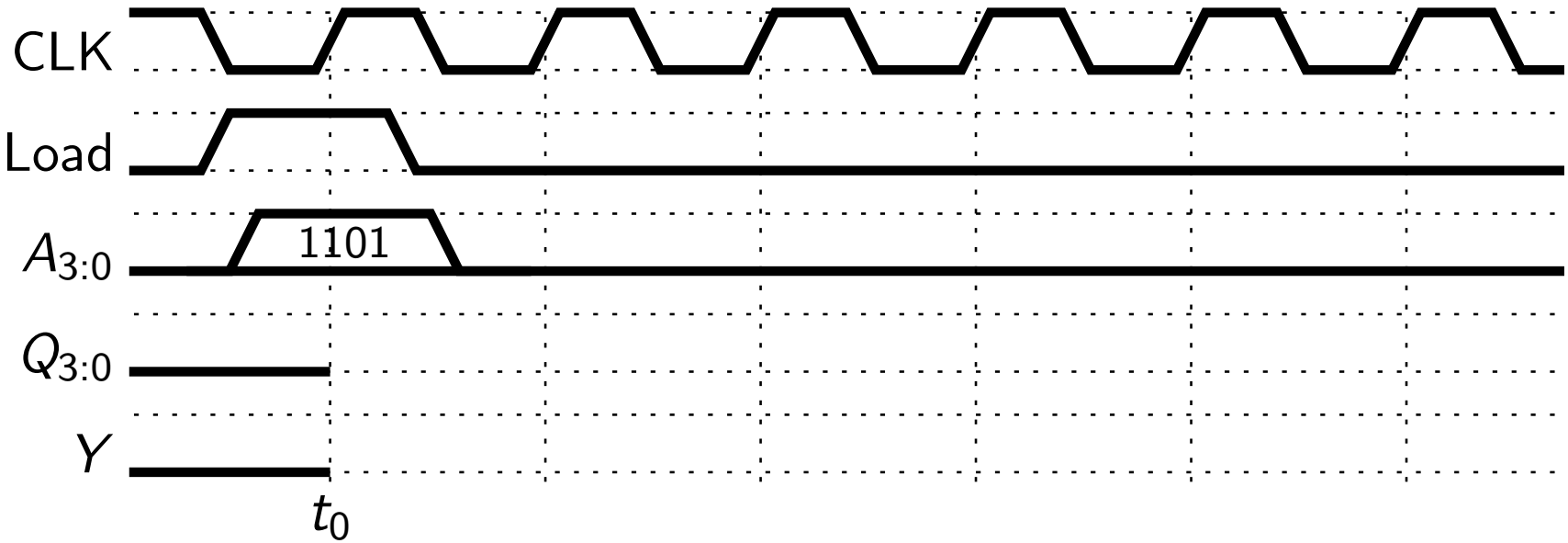
Here's an example with $N = 4$. It should be pretty obvious how to redesign the circuit for some value of N that isn't 4 ...



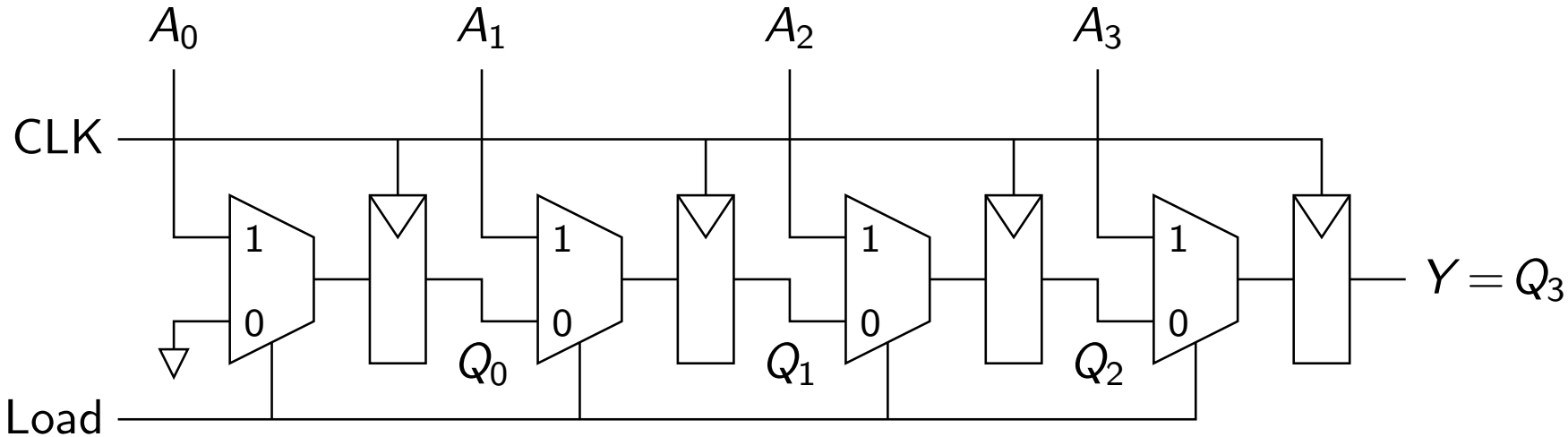
When Load $= 1$, the DFFs do a "parallel load", copying $A_{N-1:0}$ on a rising edge of CLK. When Load $= 0$, the DFFs copy their neighbour's outputs: $Q'_k = Q_{k-1}$ for $0 < k < N$.



Suppose $Q_{3:0} = 0000$ *just before time* t_0*. Let's complete the timing diagram . . .*



When is it time to load more parallel data?

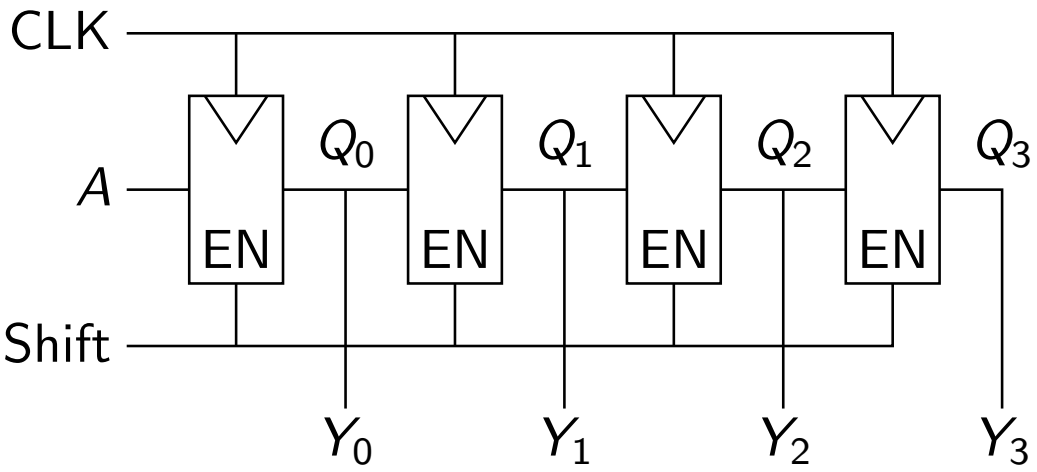


Let's suppose we've built the above circuit, but with $N = 8$ instead of $N = 4$. So there are 8 parallel inputs ($A_{7:0}$), 8 multiplexers, and 8 DFFs.

What would be a useful "companion circuit", that could request a new parallel load after all eight bits from the previous load have been copied out on Y?

N-bit serial-in-parallel-out shift register

Here's an example with $N = 4$. Note the use of DFFs with **enable** inputs.



When Shift $= 1$, data arriving on A, one bit per clock cycle, gets copied into the shift register.

When Shift $= 0$, the shift register is frozen, allowing a parallel transfer of the data on $Y_{N-1:0}$.

For the circuit to work correctly, what must be true about the signals A and CLK?

Outline of Slide Set 10

Important notes about exam topics

Counters

Shift registers

Memory arrays

Bit cells, wordlines, and bitlines

ROM circuits: dot notation, ROM-based logic

Implementation of ROM with NMOS transistors

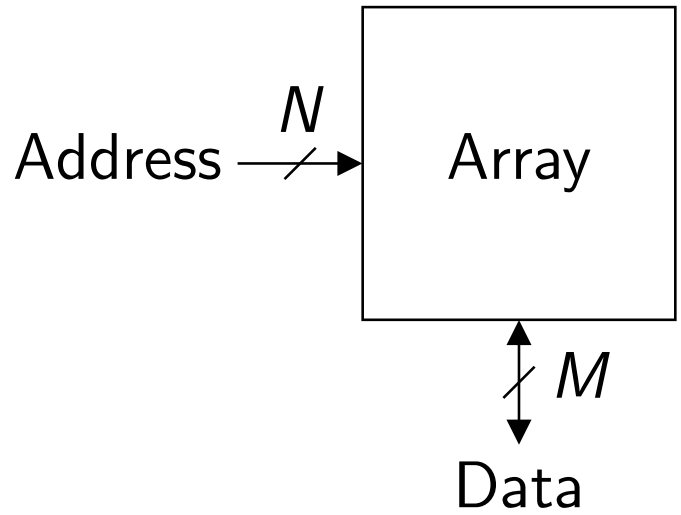
Programmable ROM circuits

How RAM and EEPROM circuits work

PLAs: Programmable logic arrays

Memory arrays (textbook Section 5.5)

A *memory array* is a digital circuit element connected to an *N*-**bit address bus**, an *M*-**bit data bus**, and most likely some other signals.



The address bus is unidirectional—it's **always an input** to the array.

The data bus is bidirectional—an **input** when the array is **written to**, an **output** when the array is **read from**.

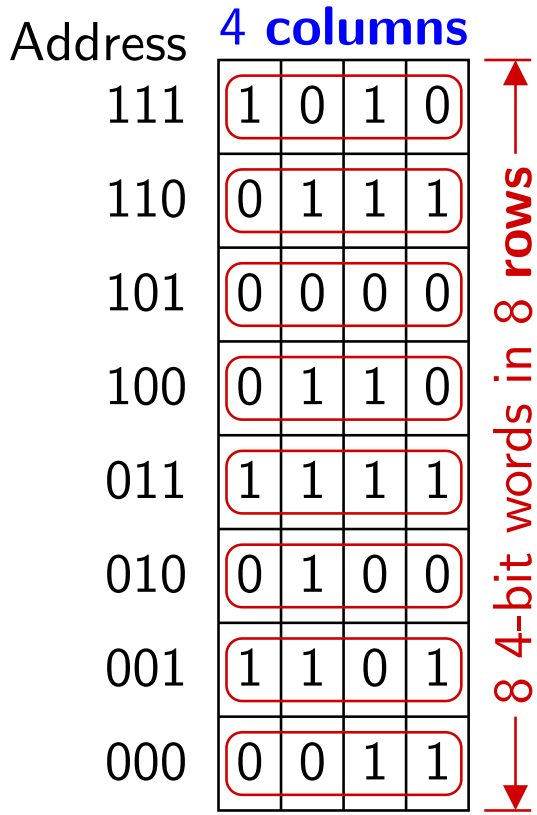
Inside the array there are $2^N \times M$ **stored bits**.

Stored bits in a memory array

Here is an example with $N = 3$ and $M = 4$.

There are $2^N = 2^3 = 8$ different possible addresses. At each address there is a stored 4-bit *word*.

This example would be called an **8 × 4 array**. In this example, 8 is the *depth* and 4 is the *width*. The *width* is sometimes called the *word size*.



Of course the 0's and 1's are not really stored as tiny pictures of the characters 0 and 1! Different kinds of memory arrays use different kinds of **circuit elements** to hold 0's and 1's.

An example of a larger array

Tiny arrays like the one on the previous slide are good for textbook and lecture examples, and there are also some practical uses for such small arrays.

However, memory arrays can be much larger.

Suppose N, the address size, is 16 bits, and M, the data size (word size) is 8 bits.

- *What are the dimensions of the memory array?*
- *How many bits are stored in the array?*

ROM and RAM: Names we're stuck with, part 1

These not-particularly-descriptive names have been around for decades, and are unlikely to change any time soon.

ROM stands for "read-only memory".

It's true that ROM is read-only, when voltages between 0 and V_{DD} are applied. (Some ROM circuits are writeable if voltages outside of the 0-to-V_{DD} range are applied!)

But it's also true and just as important that ROM is **nonvolatile**: In ROM, stored 0's and 1's are preserved when the power is off.

A more descriptive name for "ROM circuit" would be NVMA: nonvolatile memory array.

ROM and RAM: Names we're stuck with, part 2

RAM stands for "random access memory". *Random access* means that access to any one word in an array is no faster and no slower than access to any other word.

Mass storage devices such as magnetic disk drives, magnetic tape drives, and CD/DVD/Blu-ray drives **do not** have this random access property.

But ROM circuits **are** random access, which makes RAM a silly name for something that is different in important ways from ROM!

ROM and RAM: Names we're stuck with, part 3

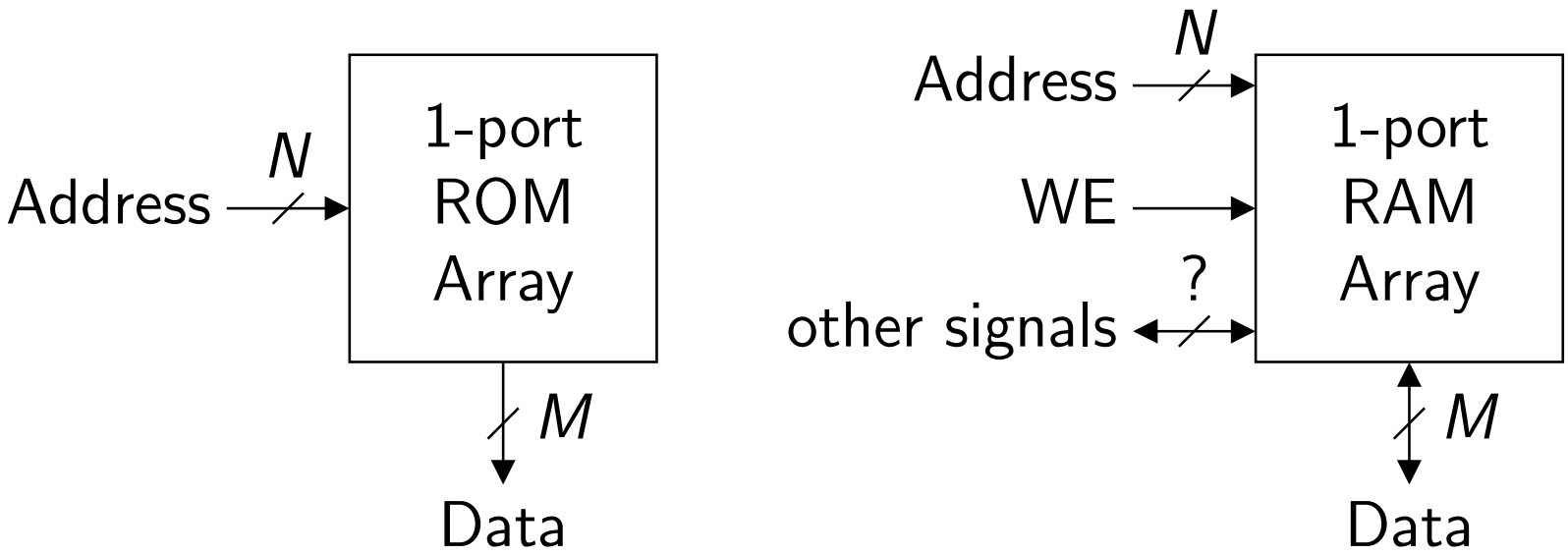
The key properties of a RAM array are:

- it is both readable and writeable, using voltages in the 0-to-V_{DD} range;
- it is **volatile**—all the stored 0's and 1's are lost when power is turned off.

A very precise but somewhat unpronounceable name for "RAM circuit" would be VRWMA (volatile readable/writeable memory array).

Your instructor seriously doubts that "VRWMA" will replace "RAM circuit" as a widely-used name any time in the near future!

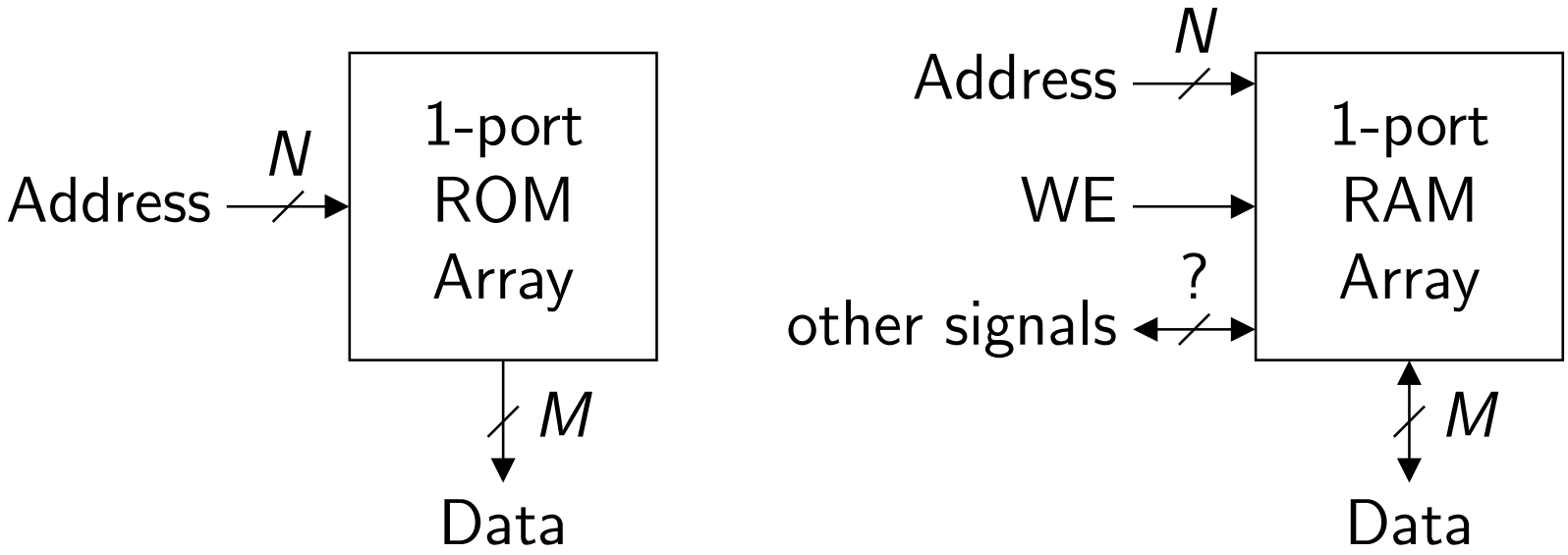
ROM compared to RAM



WE stands for "write enable". WE = 1 tells RAM to **write** to the word specified by Address, and WE = 0 tells RAM to **read** from the word specified by Address.

Which array is combinational logic and which is sequential? Is the sequential circuit synchronous or asynchronous?

How many ports?



The circuits above are called "1-port" because at any given time, the array can receive only one address and can support only one read or write operation.

See Harris & Harris for an example of a 3-port memory array, which can support two read operations and one write operation all at one time, possibly involving three different addresses.

Outline of Slide Set 10

Important notes about exam topics

Counters

Shift registers

Memory arrays

Bit cells, wordlines, and bitlines

ROM circuits: dot notation, ROM-based logic

Implementation of ROM with NMOS transistors

Programmable ROM circuits

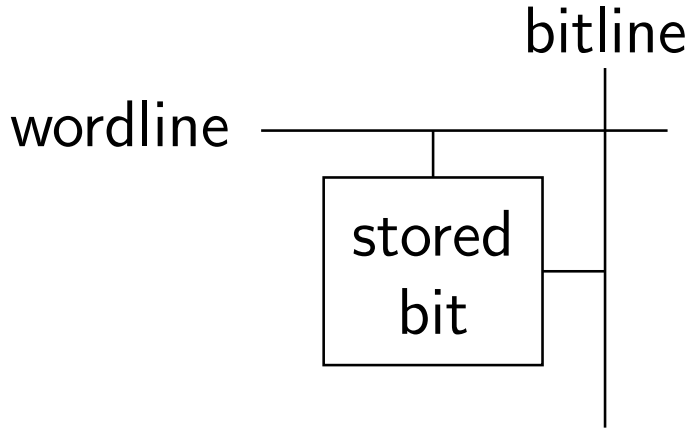
How RAM and EEPROM circuits work

PLAs: Programmable logic arrays

Bit cells, wordlines, and bitlines

Each bit stored within a memory array in stored in a tiny circuit element called a *bit cell*.

Signalling to a bit cell is done through two wires: a *wordline* and a *bitline*.

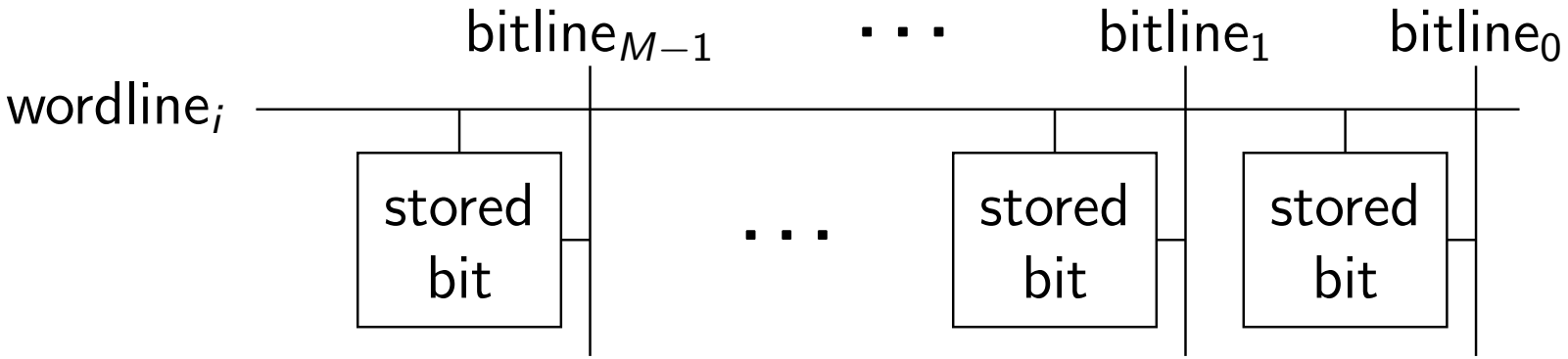


Wordline: Each wordline is connected to all of the bits within a single word.

Bitline: Each bitline is connected to all of the bits within a single column.

If a memory array has an 8-bit address bus and a 9-bit data bus, how many wordlines are there? How many bitlines?

More about wordlines



Each wordline is connected to all of the bits within a single word.

Normally one wordline is ON and all the others are OFF, so that a single word is selected for reading or writing.

What kind of circuit element is perfectly suited for converting an address input into the correct set of wordline signals?

Organization of a 4 × 3 memory array

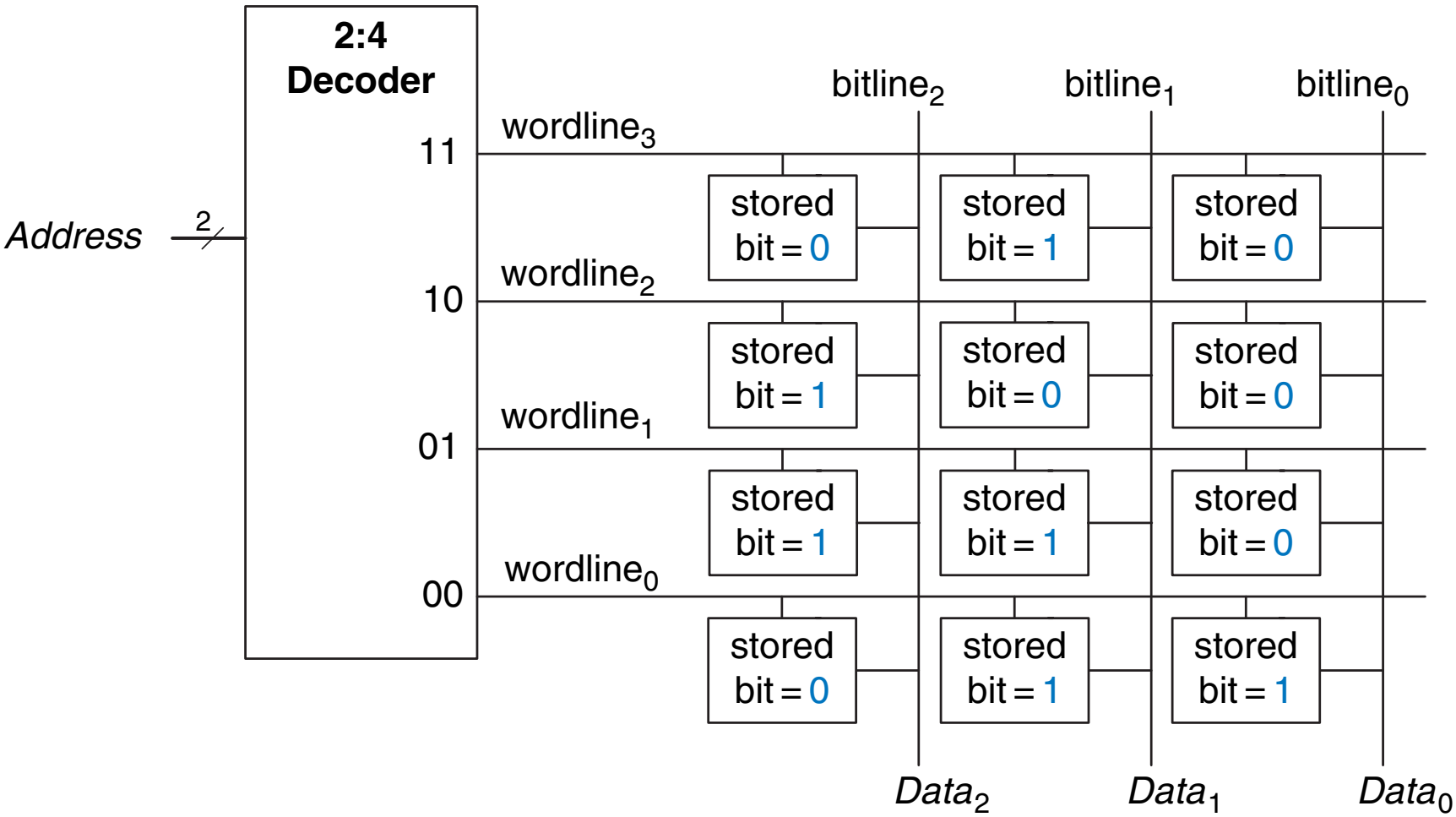


Image is Figure 5.42 from Harris D. M. and Harris S. L., *Digital Design and Computer Architecture, 2nd ed.*, © 2013, Elsevier, Inc.

Outline of Slide Set 10

Important notes about exam topics

Counters

Shift registers

Memory arrays

Bit cells, wordlines, and bitlines

ROM circuits: dot notation, ROM-based logic

Implementation of ROM with NMOS transistors

Programmable ROM circuits

How RAM and EEPROM circuits work

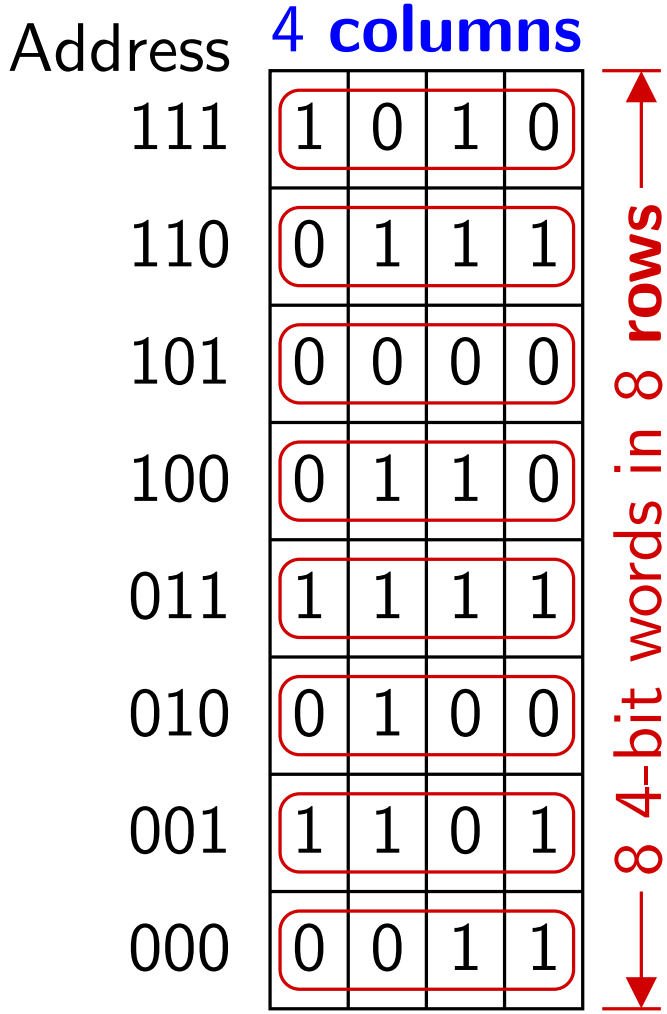
PLAs: Programmable logic arrays

Dot notation for ROM circuits

Often a ROM is drawn showing only its decoder, the wordlines and bitlines, and some dots.

A **dot** at a wordline-bitline crossing point indicates a **stored 1**. **No dot** at a crossing point indicates a **stored 0**.

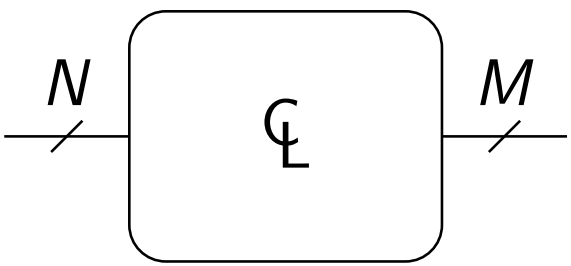
Let's draw a dot-notation diagram for ROM with the contents of the table to the right.



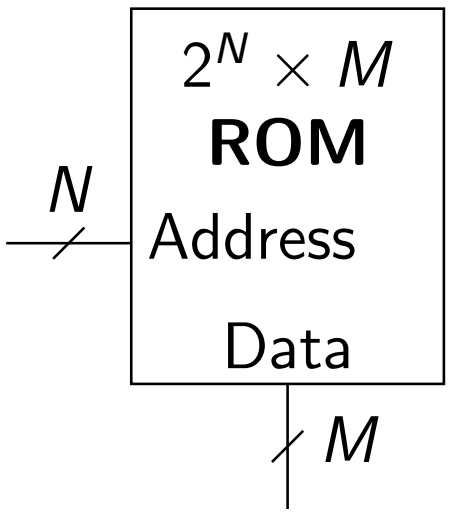
ROM-based implementation of logic functions

A ROM circuit can be thought of as a "truth table baked into silicon."

This way of thinking about ROM leads to the conclusion that **any** combinational logic function can be implemented as a ROM circuit.



can be implemented as



ROM-based logic: Examples

Let's implement the following combinational elements using ROM circuits with appropriate dimensions.

1. $F = A \oplus B$; $G = \overline{(A \oplus B)}$; $H = \overline{A} + \overline{B}$.

   Let's do this one by using algebra to express each of F, G, and H as a sum of minterms.

2. $E = AB\overline{C}$;
   $F = A \oplus B$;
   $G = AB + AC + BC$;
   $H = A + \overline{B} + \overline{C}$.

   Let's do this one by making a truth table.

Why use ROMs for combinational logic? Why not?

Using a ROM structure for N-input, M-output logic is sometimes a good design choice, and sometimes not.

"Custom" solutions with logic gates tend to use less chip area and less power than ROM-based solutions, and can be much faster.

However, the design effort needed for a ROM-based solution is close to zero, so it may be a good choice if area, power, and speed constraints are not pressing.

Outline of Slide Set 10

Important notes about exam topics

Counters

Shift registers

Memory arrays

Bit cells, wordlines, and bitlines

ROM circuits: dot notation, ROM-based logic

Implementation of ROM with NMOS transistors

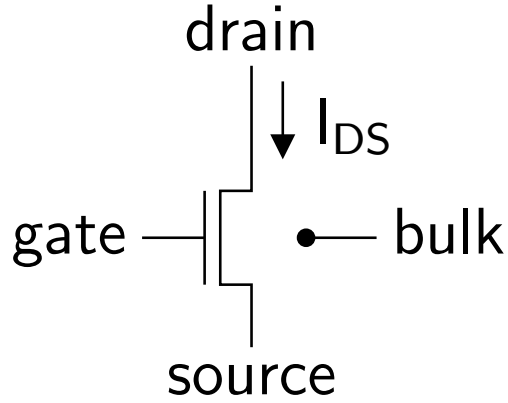
Programmable ROM circuits

How RAM and EEPROM circuits work

PLAs: Programmable logic arrays

NMOS transistors

NMOS transistors are one of the two main kinds of building blocks for CMOS circuits. An NMOS transistor has the four terminals shown, but in CMOS the bulk is assumed to be connected to ground and is usually not shown in circuit diagrams.

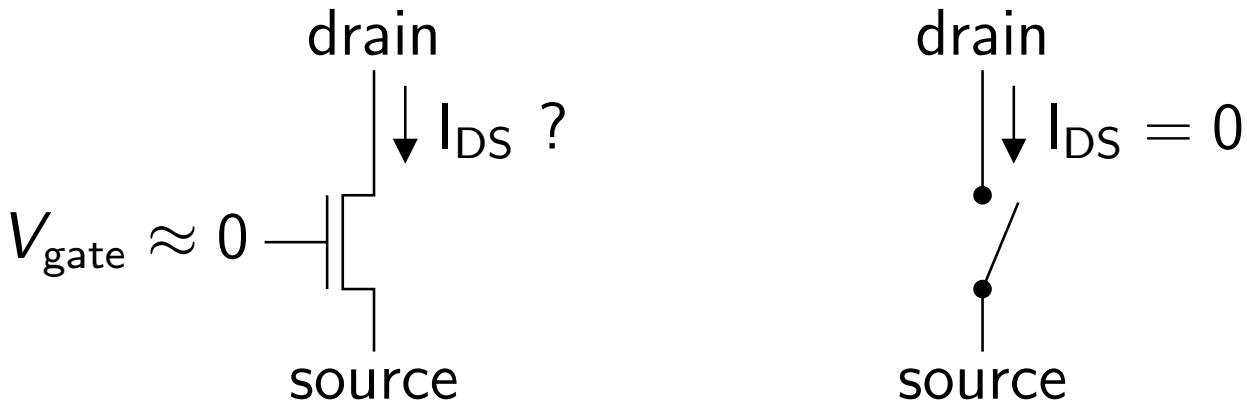


The relationship of the current I_{DS} to the voltages at the gate, drain and source is quite complex. (Electrical Engineers: See ENCM 467 in third year.) But a **simple, crude model** helps explain how bit cells work in memory arrays.

NMOS transistor with V_{gate} close to zero

The first part of our **simple, crude model** is actually quite accurate.

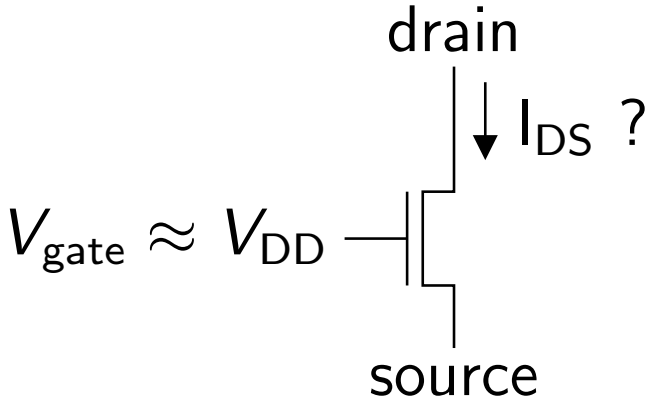
When the gate voltage is close to zero, the drain-to-source connection is like an open switch—no current can flow.

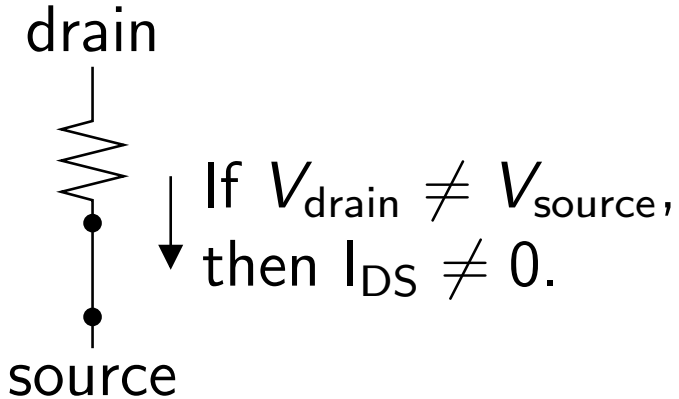


NMOS transistor with V_{gate} close to V_{DD}

The second part of our **simple, crude model** is **not very accurate**, but good enough to get a qualitative feel for how bit cells work.

When the gate voltage is close to the power supply voltage, the drain-to-source connection is somewhat like a **small** resistance in series with a closed switch—current flows if $V_{drain} \neq V_{source}$.



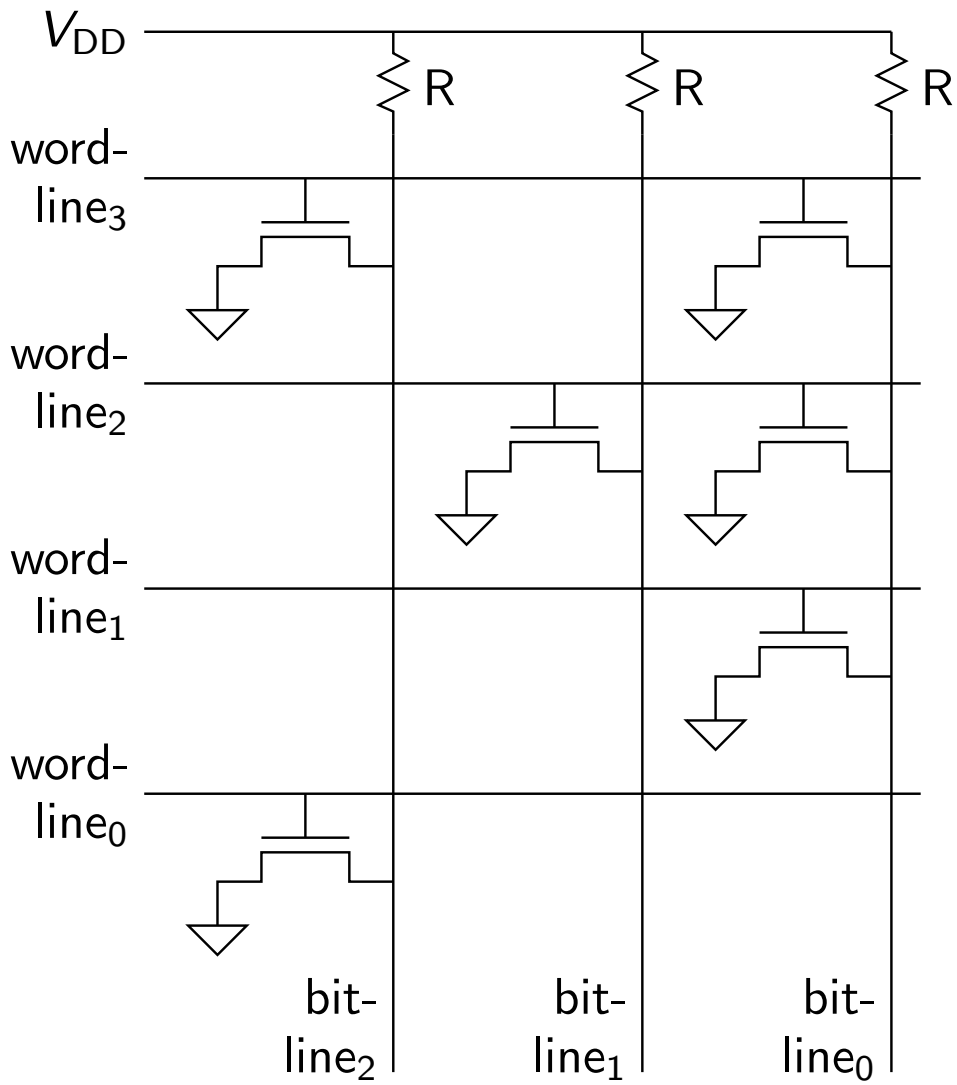


A ROM circuit made with NMOS transistors

A 2:4 decoder drives the wordlines, but is not shown here, to save space on this slide.

What happens to the bitlines if wordline$_2$ is turned ON and the other three wordlines are OFF?

What are the contents of this ROM array?



Outline of Slide Set 10

Important notes about exam topics

Counters

Shift registers

Memory arrays

Bit cells, wordlines, and bitlines

ROM circuits: dot notation, ROM-based logic

Implementation of ROM with NMOS transistors

Programmable ROM circuits

How RAM and EEPROM circuits work

PLAs: Programmable logic arrays

Programmable ROM circuits

The ROM circuit on the previous slide can only be made in a **semiconductor fab**, which is a fancy name for "chip factory".

It's obviously useful to have circuits that have the essential ROM properties—contents not lost when power is turned off, contents won't change when normal digital logic voltages are applied—but are also **programmable**.

A *programmable ROM* is a ROM circuit into which 1's and 0's can be written **after** the circuit is fabricated.

See Figure 5.51 and related discussion in Harris & Harris for explanation of *one-time-programmable* ROM circuits based on **fuses** that are either blown or intact.

Much cooler than a ROM circuit that can be programmed only once is a ROM circuit that can programmed, then erased, and then re-programmed, many, many times.

Production of this kind of erasable, programmable ROM circuit is a **huge industry**, based on one key electronic device: the **floating-gate transistor**.

Some of the many products that depend utterly on floating-gate transistors are . . .

- USB "thumb" drives
- SD cards and related storage techologies
- solid-state drives in laptop and desktop computers
- smartphones and tablet computers

Some slides at the end of this slide set give a brief explanation of floating-gate transistor behaviour.

Outline of Slide Set 10

Important notes about exam topics

Counters

Shift registers

Memory arrays

Bit cells, wordlines, and bitlines

ROM circuits: dot notation, ROM-based logic

Implementation of ROM with NMOS transistors

Programmable ROM circuits

How RAM and EEPROM circuits work

PLAs: Programmable logic arrays

How RAM and EEPROM circuits work

Slides 43–57 try to briefly explain how RAM circuits and EEPROM (electrically-erasable programmable ROM) circuits work, and do **not** contain examinable material.

How writes and read work in RAM circuits

The next few slides attempt to give a rough explanation of how data is written into and read out of bits cells in a RAM array.

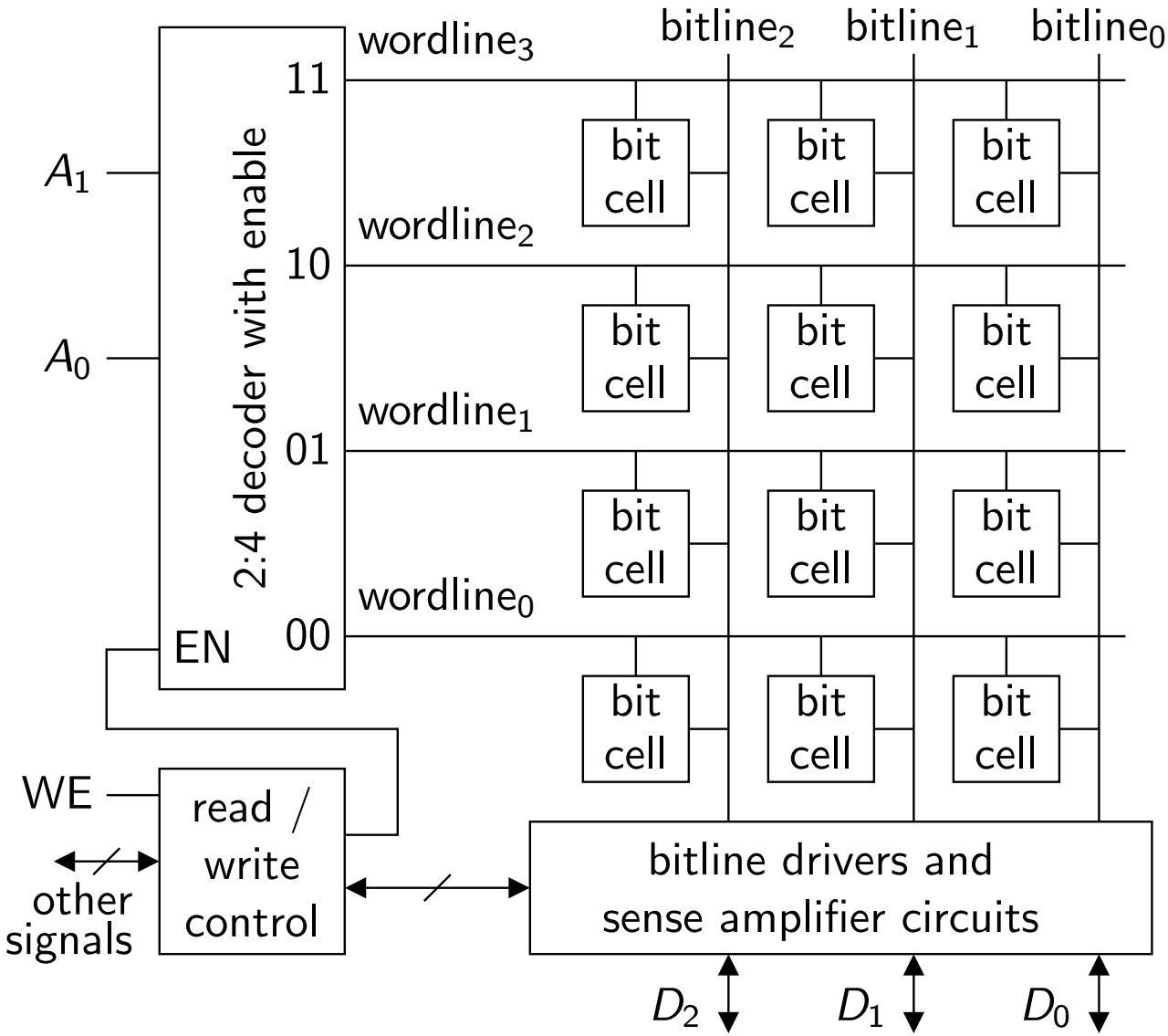
It's hard to give a really detailed explanation without referring to ideas about transistors and circuit theory that come later than Fall Term of Year 2 in the ENEL degree program.

The next slide shows a hypothetical 4×3 RAM array.

Keep in mind that practical RAM circuits are usually much, much larger, with thousands, millions, or even billions of bit cells.

A lot of the circuit design issues have to with the fact that bit cells are tiny, not-very-powerful circuits, while bitlines are relatively lengthy pieces of metal. It's not possible for a single bit cell to quickly drive the voltage on a bitline all the way from LOW to HIGH or HIGH to LOW.

4×3 RAM array



A write in the 4×3 RAM array, part 1

The EN input of the decoder is normally OFF, so that all wordlines are normally LOW.

The read/write control circuit sees that WE $= 1$, and one of its "other signals" tells the circuit that the address $A_{1:0}$ and the input data $D_{2:0}$ are ready.

Bitline drivers quickly change the voltage levels of the bitlines to match $D_{2:0}$.

EN is turned ON, which turns on the wordline corresponding to the bit pattern on $A_{1:0}$.

A write in the 4×3 RAM array, part 2

Bit cells in the selected word copy voltages from the bitlines.

There will be **contention** (also called **fighting**) in cells where a 0 is replacing a 1, or a 1 is replacing a 0.

The bitline drivers **win all the fights**, because the bitline drivers are much stronger than the bit cells.

EN is turned off.

A read in the 4×3 RAM array, part 1

An "other signals" input to the read/write control circuit indicates that a read is requested and that the address $A_{1:0}$ is ready.

The bitline drivers "precharge" the bitlines to a voltage V_{PRE} about halfway between LOW and HIGH, then leave the bitlines floating.

The voltage of V_{PRE} will stay on the bitlines briefly because they act like capacitors.

A read in the 4 $\times$ 3 RAM array, part 2

EN is turned ON, which turns on the wordline corresponding to $A_{1:0}$.

Bit cells containing 0's drive their bitlines slightly lower than V_{PRE}. Bit cells containing 1's drive their bitlines slightly higher than V_{PRE}.

Each small voltage change on a bitline is detected by a "sense amplifier" circuit at the end of the bitline, and converted to a solid LOW or HIGH voltage suitable for a data output wire.

EN is turned off.

Warning about oversimplification

I have tried to keep the story told on the last few slides short and simple, and to convey ideas that are common to the operation of both SRAM and DRAM circuits.

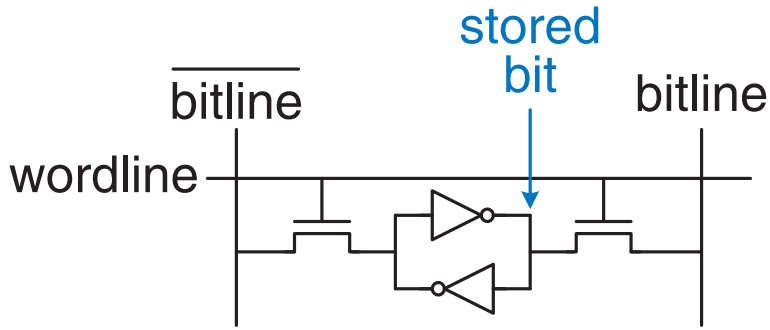
To do that I have had to be **imprecise**, leaving out important details, and somewhat **inaccurate**, suggesting some things that aren't really true.

I'm not going to try to be more precise and accurate about DRAM, because that just takes too much time.

But on the next two slides I'll clear some things up about SRAM.

Writing to an SRAM cell

An SRAM cell is made from a pair of inverters connected in a bistable configuration, and a pair of NMOS transistors.

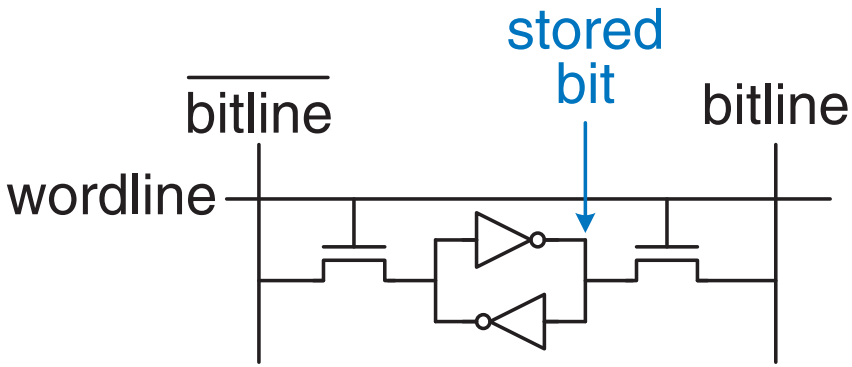


Every column in an SRAM array has **two** wires, called bitline and $\overline{\text{bitline}}$.

On a **write**, bitline and $\overline{\text{bitline}}$ are driven to opposite levels. Whichever line is LOW will do most of the work in changing the state of the cell, because NMOS transistors pass 0's better than they pass 1's.

Image is Figure 5.46 from Harris D. M. and Harris S. L., *Digital Design and Computer Architecture, 2nd ed.*, © 2013, Elsevier, Inc.

Reading from an SRAM cell



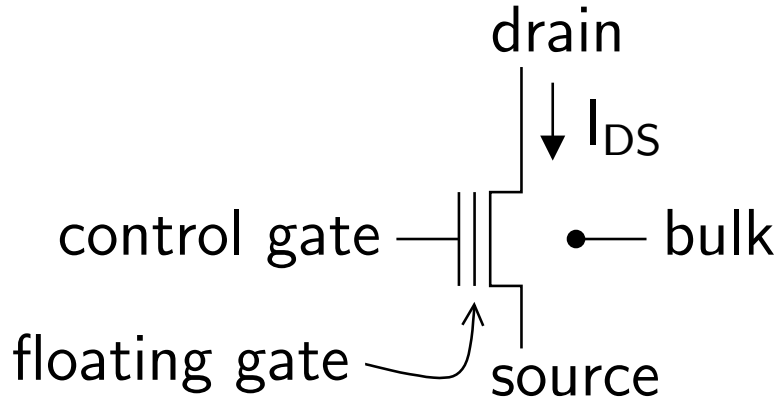
On a **read**, bitline and $\overline{\text{bitline}}$ are precharged to equal V_{PRE} levels before the wordline is turned ON.

The bistable pair of inverters will pull one of the bitline and $\overline{\text{bitline}}$ voltages up slightly and the other voltage down slightly.

A sense amplifier at the end of the bitline-$\overline{\text{bitline}}$ pair will detect a voltage difference and decide whether the cell contains a 1 or a 0.

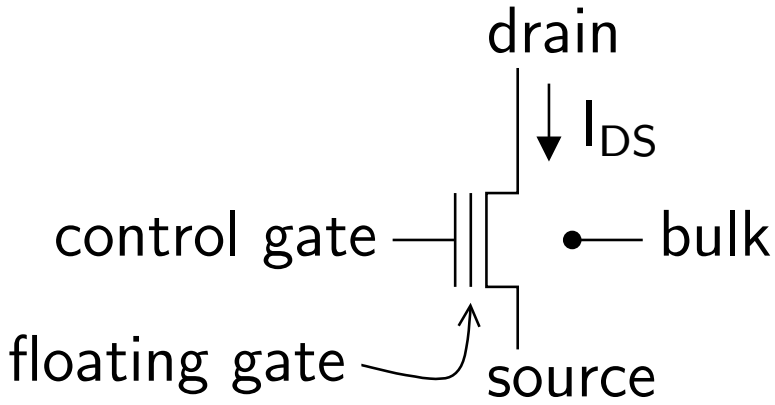
Floating-gate transistors, part 1

The **floating gate** in a *floating-gate transistor* is surrounded by insulating material. Electrons trapped on a floating gate will stay there for **many, many years**.



If the charge on the floating gate is **neutral** and voltages applied are all in the 0 to V_{DD} range, then the circuit acts like an NMOS transistor—the voltage on the control gate will decide whether current can flow from drain to source.

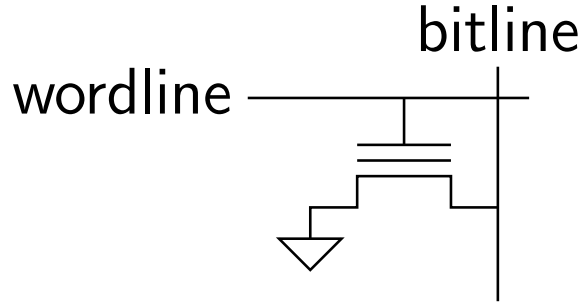
Floating-gate transistors, part 2



If the charge on the floating gate is **significantly negative** and voltages applied are again all in the 0 to V_{DD} range, then $I_{DS} = 0$ regardless of the control gate voltage—there is effectively no electrical connection between drain and source.

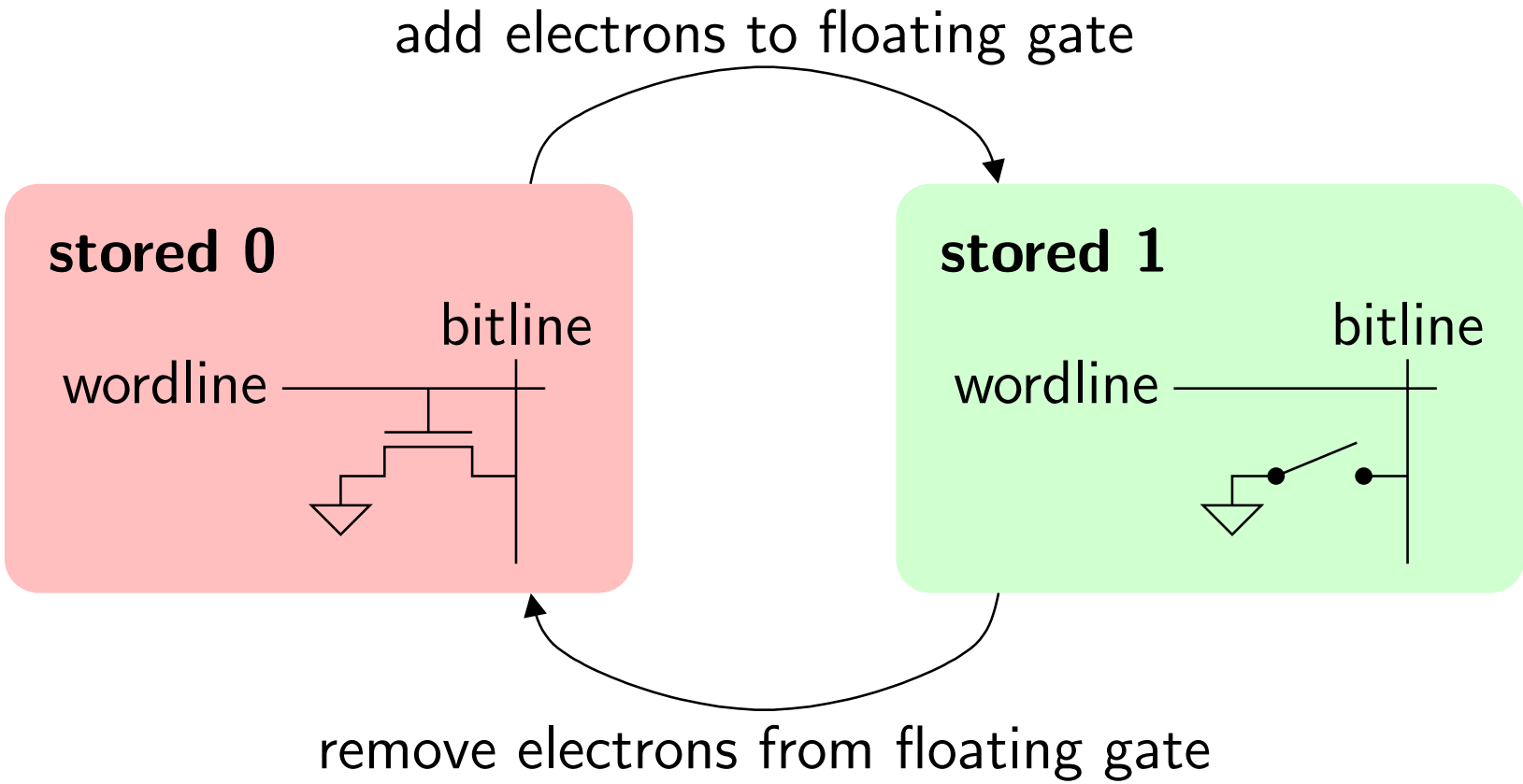
Floating-gate transistors as bit cells in memory arrays, part 1

Memory arrays can be built with floating-gate transistors at every wordline-bitline crossing point.



If you understand the basic operation of the ROM circuit presented many slides back, it should be clear that a floating-gate transistor bit cell holds either a 0 or a 1 depending on the charge on its floating gate.

Floating-gate transistors as bit cells in memory arrays, part 2



Floating-gate transistors and EEPROM arrays

EEPROM: Electrically-erasable programmable ROM.

An EEPROM array has circuitry that can move electrons on to or off of the floating gates in its bit cells.

The key to this circuitry is that by applying voltages significantly outside the 0 to V_{DD} range, it's possible to get electrons to move through the insulation surrounding a floating gate.

Erasing: Restoring neutral charge to all floating gates.

Programming: Selectively injecting electrons on to some floating gates, so that an EEPROM array contains some desired pattern of 0's and 1's.

Outline of Slide Set 10

Important notes about exam topics

Counters

Shift registers

Memory arrays

Bit cells, wordlines, and bitlines

ROM circuits: dot notation, ROM-based logic

Implementation of ROM with NMOS transistors

Programmable ROM circuits

How RAM and EEPROM circuits work

PLAs: Programmable logic arrays

PLAs: Programmable logic arrays (text Section 5.6.1)

As explained previously, **any** combinational logic function can be implemented as a ROM circuit with the appropriate numbers of address inputs and data outputs.

However, ROM implementation of combinational logic has some drawbacks:

- The size of the decoder grows exponentially with the number of input bits.
- Propagation delays may be lengthy due to relatively slow voltage changes on long bitlines.

A **PLA**—*programmable logic array*—is a structure that allows **SOP-based** implementation of combinational logic in a way that may be more efficient than a ROM array.

Review of terminology related to SOP expressions

Literal: Either an input variable or its complement.
If the input variables are A, B, C, then the available literals are A, $\overline{A}$, B, $\overline{B}$, C, $\overline{C}$.

Product: Either a literal, or two or more literals ANDed together.

Minterm: A product that includes literals from **all** of the input variables.

Implicant: A product is an *implicant* of Y if it can be included in a *sum-of-products* (SOP) expression for Y.

What products are available? Let's look at the example of a 3-input system with inputs A, B, C.

All of the literals are products: A, $\overline{A}$, B, $\overline{B}$, C, $\overline{C}$.

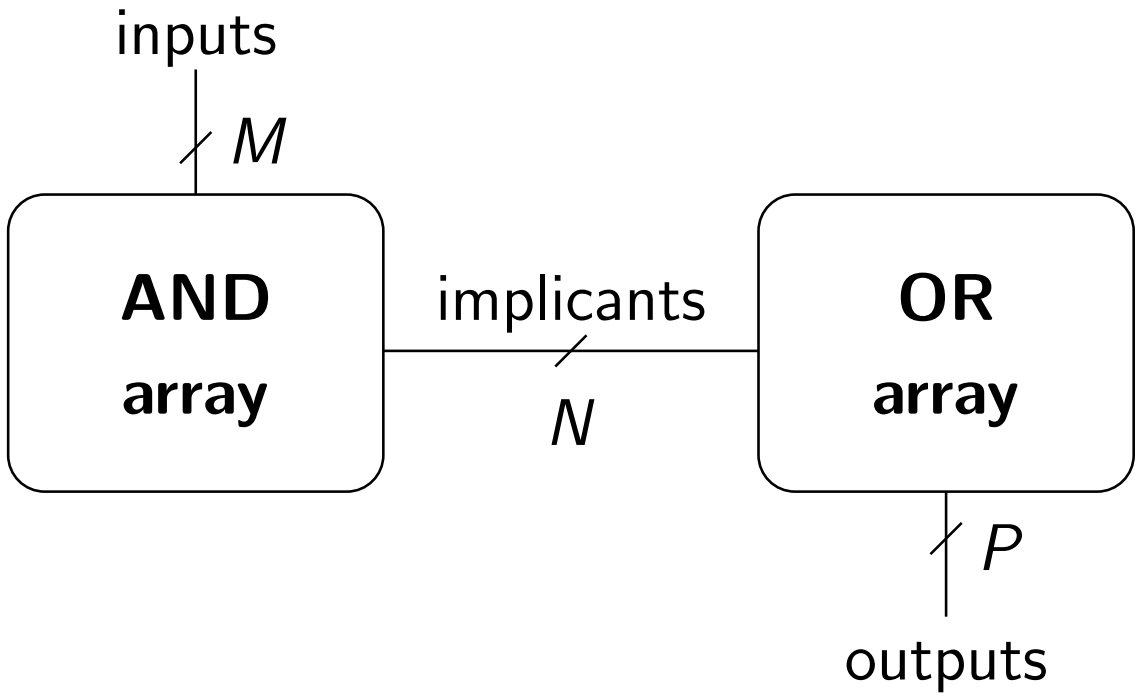
All of the minterms are products:
$\overline{A}\,\overline{B}\,\overline{C}$, $\overline{A}\,\overline{B}\,C$, $\overline{A}\,B\,\overline{C}$, $\overline{A}\,B\,C$, $A\,\overline{B}\,\overline{C}$, $A\,\overline{B}\,C$, $A\,B\,\overline{C}$, $A\,B\,C$.

There are a lot of products that are neither literals nor minterms:
$\overline{A}\,\overline{B}$, $\overline{A}\,B$, $A\,\overline{B}$, $A\,B$, $\overline{A}\,\overline{C}$, $\overline{A}\,C$, $A\,\overline{C}$, $A\,C$,
$\overline{B}\,\overline{C}$, $\overline{B}\,C$, $B\,\overline{C}$, $B\,C$.

As the number of input variables goes from 3 to 4 to 5 and beyond, the number of literals, minterms, and non-literal-non-minterm products gets very big, very fast.

A key aspect of **PLAs** is the ability to select products with **whatever number of literals makes sense**.

A generic $M \times N \times P$ PLA



The **AND array** generates all the products (implicants) that are needed in SOP expressions for the outputs.

The **OR array** does the work of ORing together implicants to generate outputs.

Dot notation for PLAs (and ROM arrays)

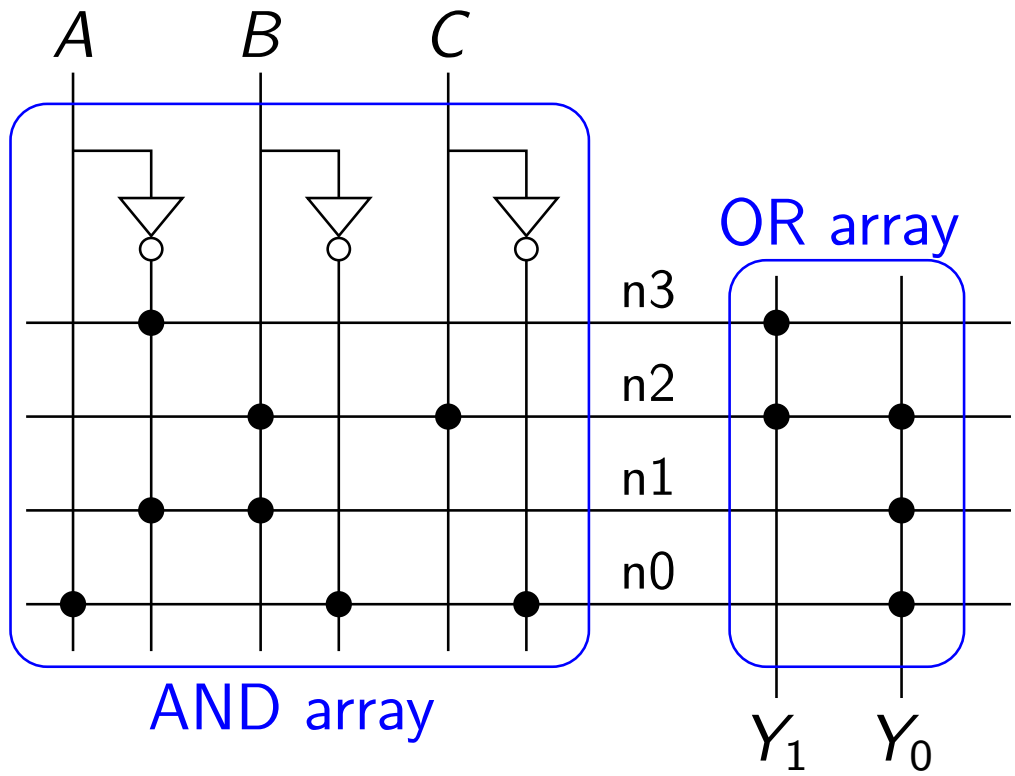
In diagrams for PLAs and ROMs, a dot on a pair of crossing wires **never** indicates simple electrical connection of the two wires!

Review: *What does a dot indicate in a ROM diagram?*

Let's write down what dots mean in PLA diagrams.

An example PLA, shown with dot notation

Note the use of inverters to generate complements of the input variables.



What are the dimensions M, N, and P?
Give algebraic expressions for Y_1 and Y_0.

PLA design example #1

Let's implement the given truth table using a $3 \times 3 \times 2$ *PLA.*

The solution is not unique.
What are two different SOP expressions that work for Y_1*?*

A	B	C	Y_1	Y_0
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	1	0
1	1	0	0	0
1	1	1	0	0

How many implicant wires should a PLA have?

Typically N, the number of implicant wires, is significantly less than 2^M, where M is the number of inputs.

Why would it make no sense at all to have $N > 2^M$?

PLA design example #2

Suppose you have these K-maps and have been asked to implement the logic using a $3 \times 3 \times 2$ PLA.

Y_1

C \ AB	00	01	11	10
0		1		
1		1	1	

Y_0

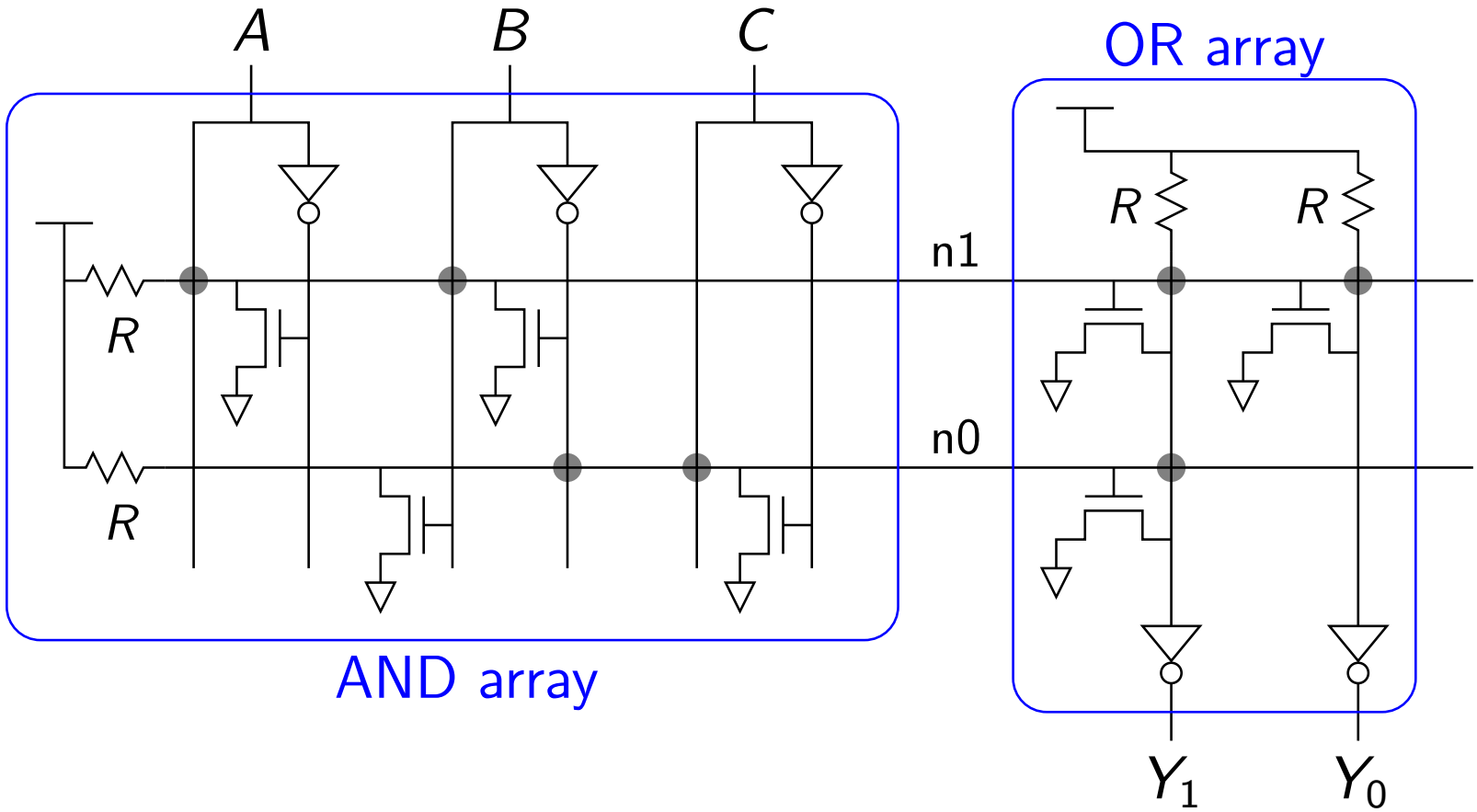
C \ AB	00	01	11	10
0	1	1		
1			1	

What goes wrong if you use the K-maps to find separate minimal SOP expressions for Y_1 and Y_0?

By looking at the K-maps again, let's find SOP expressions that will work for the PLA.

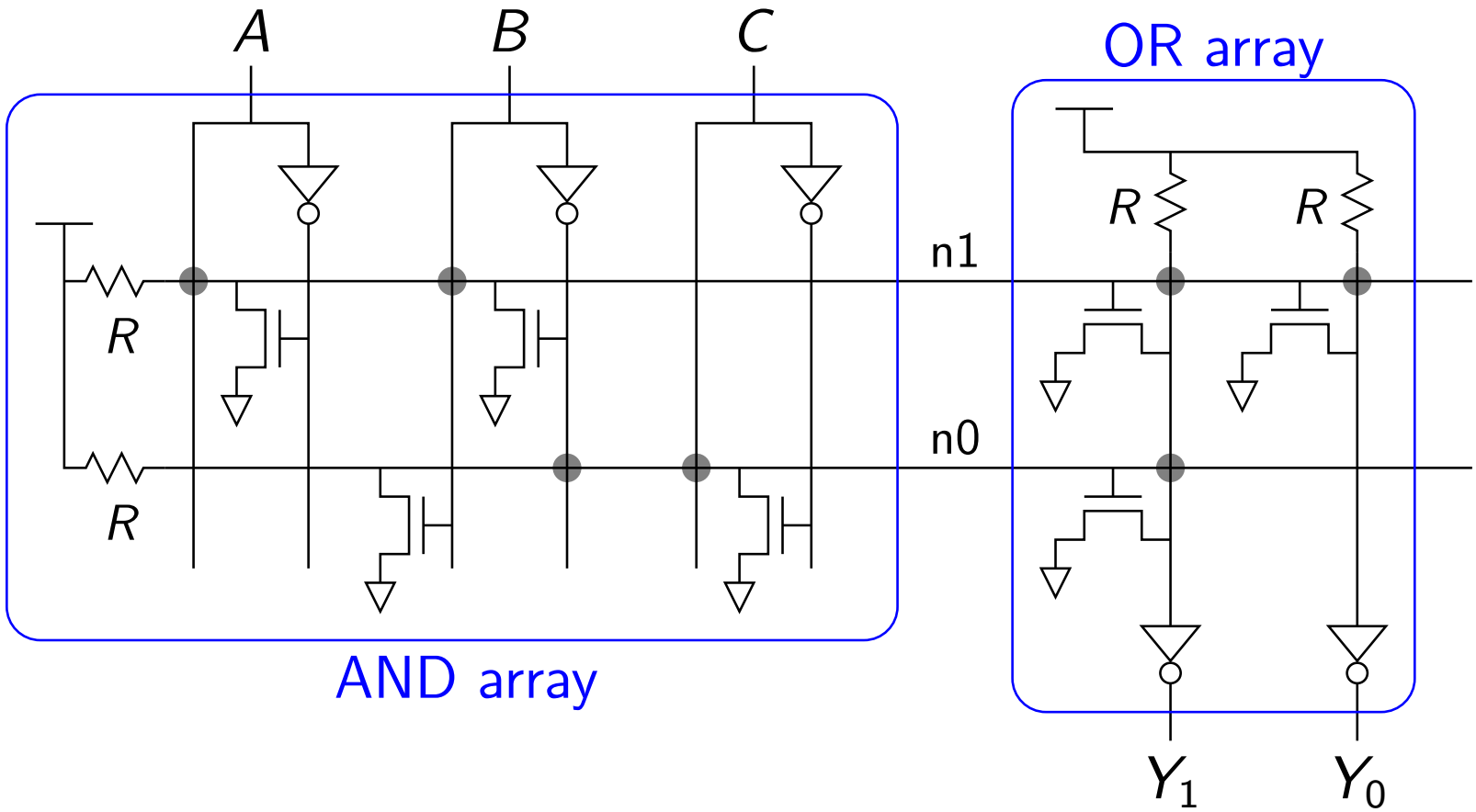
PLA implementation with NMOS transistors

$Y_1 = AB + \bar{B}C$ and $Y_0 = AB$ in this $3 \times 2 \times 2$ PLA. You can tell from the dots! **But why** does this circuit work?



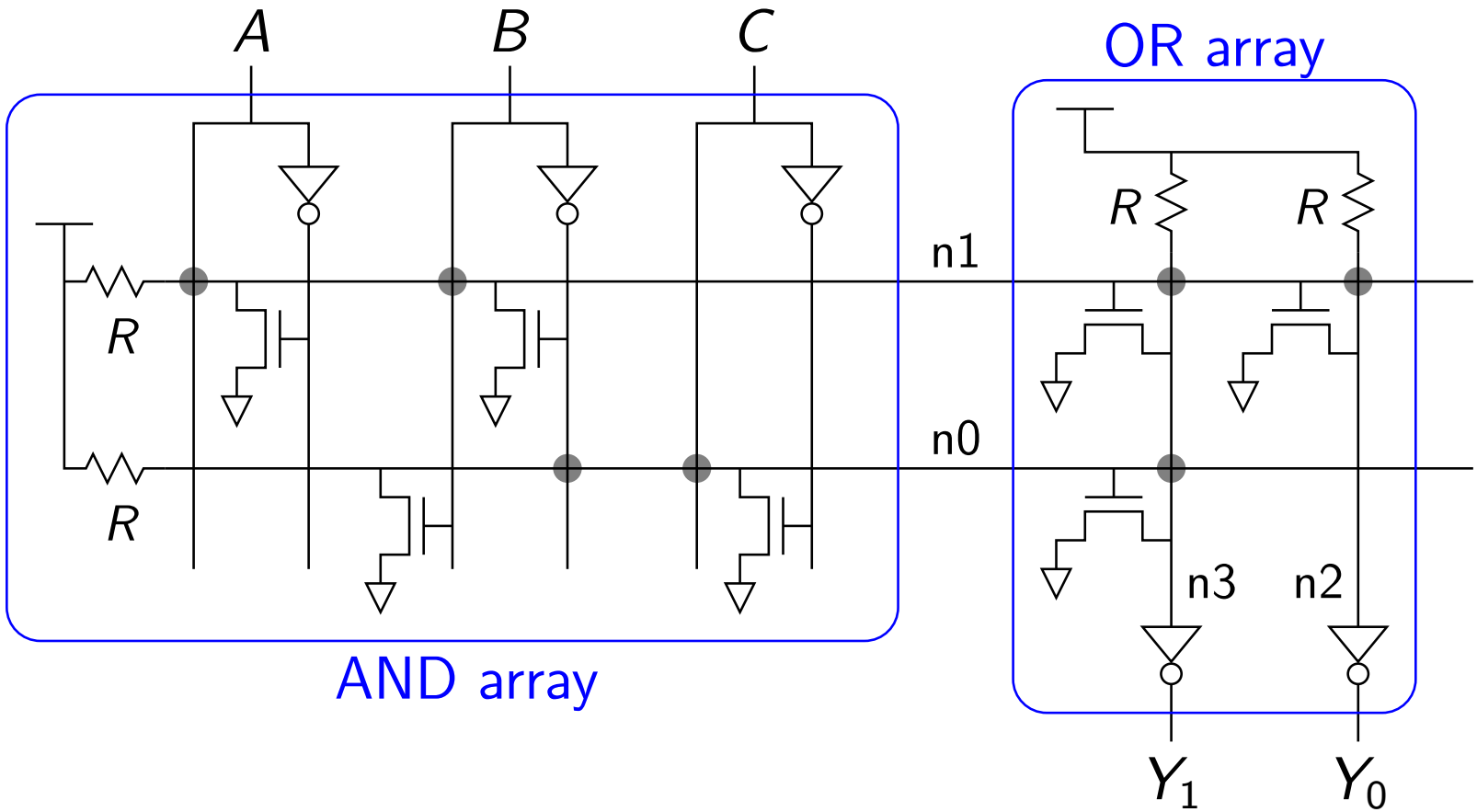
How does the AND array work?

For example, why is it that n1 = *AB?*



How does the OR array work?

For example, why is it that $Y_1 = \mathrm{n1} + \mathrm{n0}$*?*



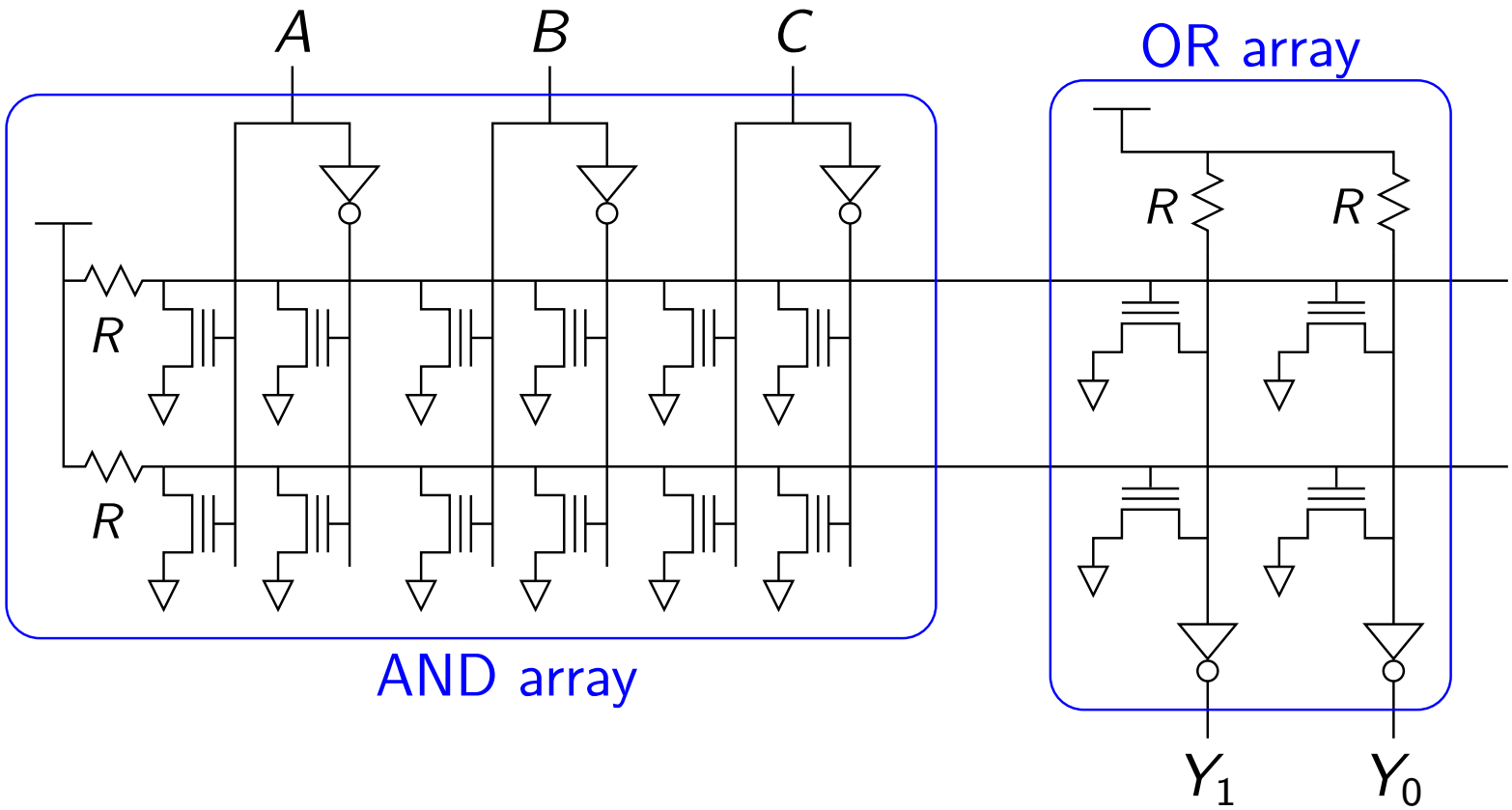
The textbook on ROM and PLA implementations with NMOS transistors

Section 5.6.3 in Harris & Harris covers this topic.

Unfortunately, Figure 5.63 has a couple of confusing typographical errors, which are not hard to spot if you compare it to Figure 5.55 ...

- There is a missing dot in the AND array.
- The implicant labeled as AB should be labeled $A\bar{B}$.

PLA implemented with floating-gate transistors



Put neutral charge on floating gates wherever an NMOS transistor would be needed. Put negative charge on floating gates wherever the absence of an NMOS transistor would be needed.

Why study PLAs in a course like ENEL 353?

The idea that a chip that is just a single PLA could be commercially viable has been obsolete for decades. (Tens of years, not tens of clock cycles!) A typical programmable logic chip in 2019 is much more powerful and much faster than a PLA.

So why study PLAs?

- PLA design provides an excellent review of some fundamental concepts in combinational logic design.
- Use of PLA-like structures within a larger digital integrated circuit design still makes sense.