Last Time

- Introduction to embedded systems
  - Properties
  - Examples
- Class structure

Administrivia

Lab hours proposal:

- Monday (Alois): 1-5
- Tuesday (Me): 10:30-3:30
- Wednesday (Alois): 9-1
- Thursday (Me): 12-5

Lab location: EL 124

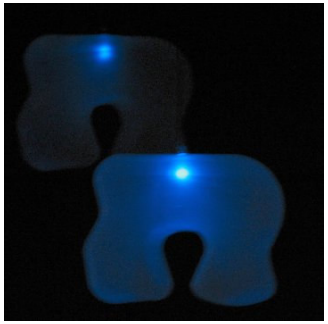
- If you do not find the lab open, find me in my office (EL 152)

Administrivia

Office hours (until lab opens):

- Andrew Fagg (EL 152):
  - Tuesday: 10:30-11:30
  - Thursday: 1-2
  - Or by appointment
- Alois:
  - Monday: 1-2
  - Wednesday: 9-10
  - Or by appointment

“Bion:” An Experiment in Sensor Networks and Art



- Test installation in Stephenson Research and Technology Center (South Campus)
- Best viewing will be Friday

Today: Introduction to Digital Logic

- Binary encoding
- Boolean algebra
- Transistors to logic gates

Encoding Information

In your ‘circuits and sensors’ class: how did you encode information?

- e.g., the acceleration measured by your accelerometer?
- or the rate of bend of a piezoelectric device?

Encoding Information

Following some form of (implementation dependent) analog conditioning:

- Acceleration (or bend rate) is encoded in the voltage that is output from the circuit
- As acceleration increases, the voltage also increases

Encoding Information

Following some form of (implementation dependent) analog conditioning:

- Acceleration (or bend rate) is encoded in the voltage that is output from the circuit
- As acceleration increases, the voltage also increases

- We say that this is an **analog** or **continuous** encoding of the information

Analog Encoding

What is the problem with analog encoding?

Analog Encoding

What is the problem with analog encoding?

- Small injections of noise – either in the sensor itself or from external sources – will affect this analog signal
- This leads to errors in how we interpret the sensory data

How do we fix this?

Digital Encoding

How do we fix this?

- At any instant, a single signal encodes one of two values:
  - A voltage around 0 (zero) Volts is interpreted as one value
  - A voltage around +5 V is interpreted as another value

Binary Encoding

- Binary digits can have one of two values: 0 or 1
- We call 0V a binary “0” (or FALSE)
- And +5V a binary “1” (or TRUE)

Binary Encoding

- Exactly what these levels are depends on the technology that is used (it is common now to see +1.8V as a binary 1 in low-power processors)

- This encoding is much less sensitive to noise: small changes in voltage do not affect how we interpret the signal

Transistors

What do transistors do for us?

Transistors

What do transistors do for us?

- They act as current amplifiers

- But: we can use them as electronic switches to process digital signals

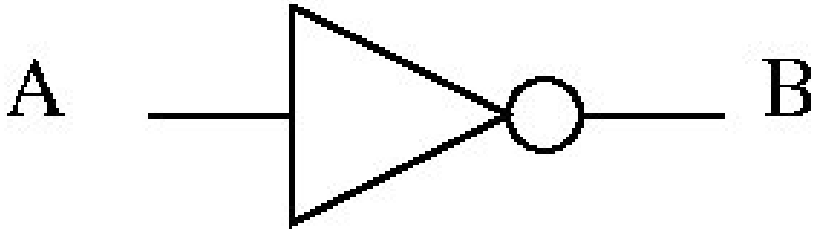
Transistors to Digital Processing

- Input: 0V -> Output +5V
- Input: +5V -> Output 0V

- We call this a “NOT” gate

The NOT Gate

- Logical Symbol:



- Algebraic Notation: $B = \overline{A}$

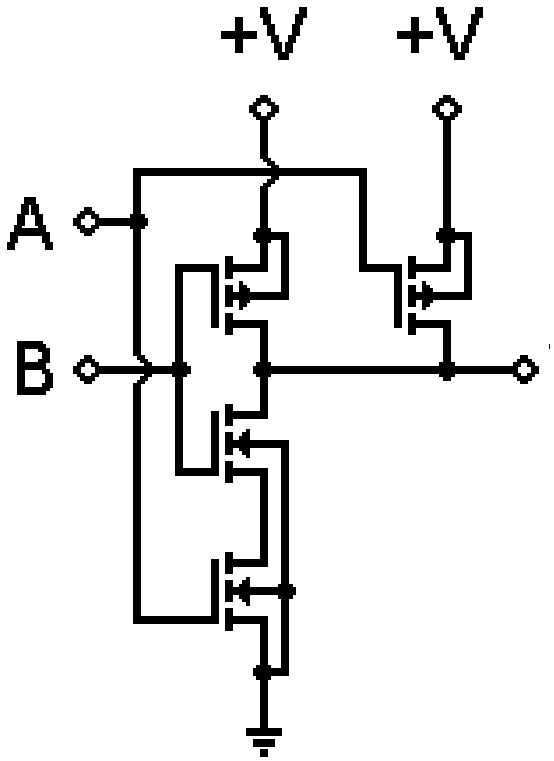
- Truth Table:

A	B
0	1
1	0

A Two-Input Gate

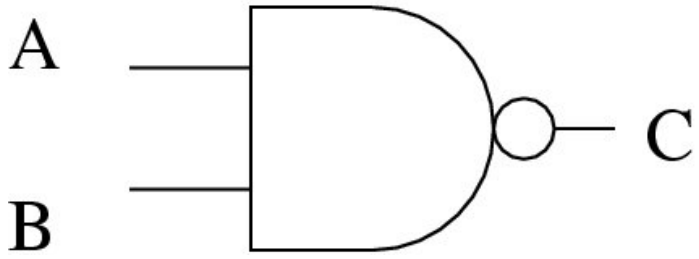
What does this circuit compute?

- A and B are inputs
- C is the output



The “NAND” Gate

- Logical Symbol:



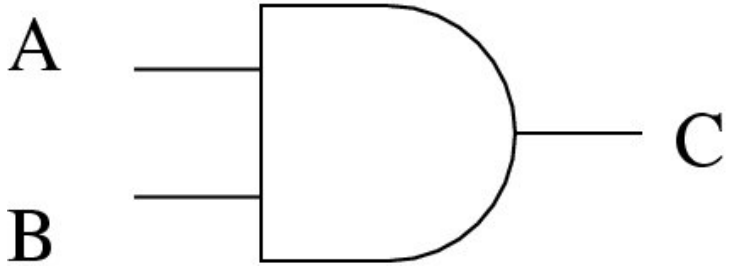
- Algebraic Notation: $C = \overline{A*B} = \overline{AB}$

- Truth Table:

A	B		C
0	0		1
0	1		1
1	0		1
1	1		0

The “AND” Gate

- Logical Symbol:



- Algebraic Notation: C = A*B = AB

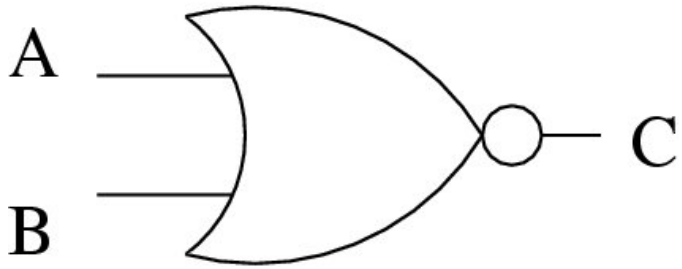
- Truth Table:

A	B		C
0	0		0
0	1		0
1	0		0
1	1		1

Yet Another Gate

The “NOR” Gate

- Logical Symbol:



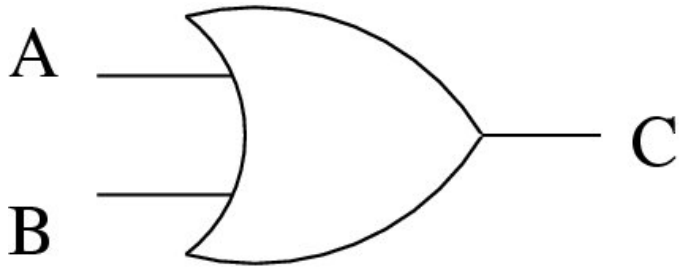
- Algebraic Notation: $C = \overline{A+B}$

- Truth Table:

A	B		C
0	0		1
0	1		0
1	0		0
1	1		0

The “OR” Gate

- Logical Symbol:



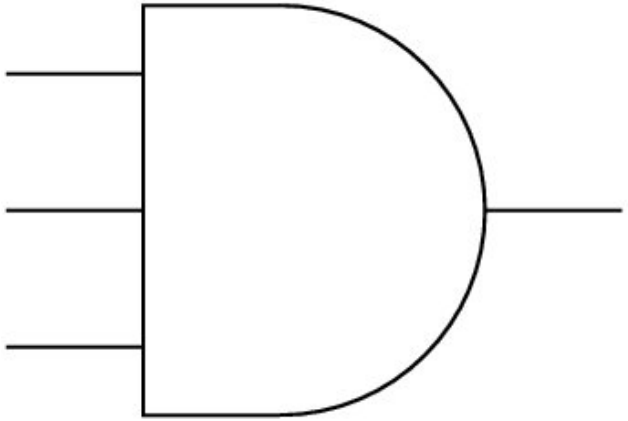
- Algebraic Notation: C = A+B

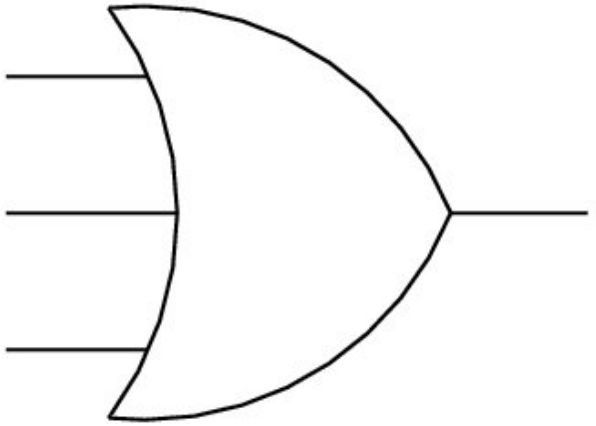
- Truth Table:

A	B		C
0	0		0
0	1		1
1	0		1
1	1		1

N-Input Gates

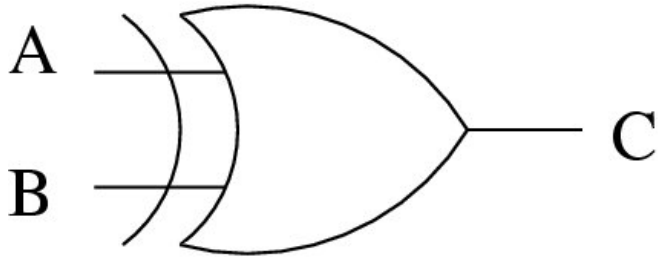
Gates can have an arbitrary number of inputs (2,3,4,8,16 are common)





Exclusive OR ("XOR") Gates

- Logical Symbol:

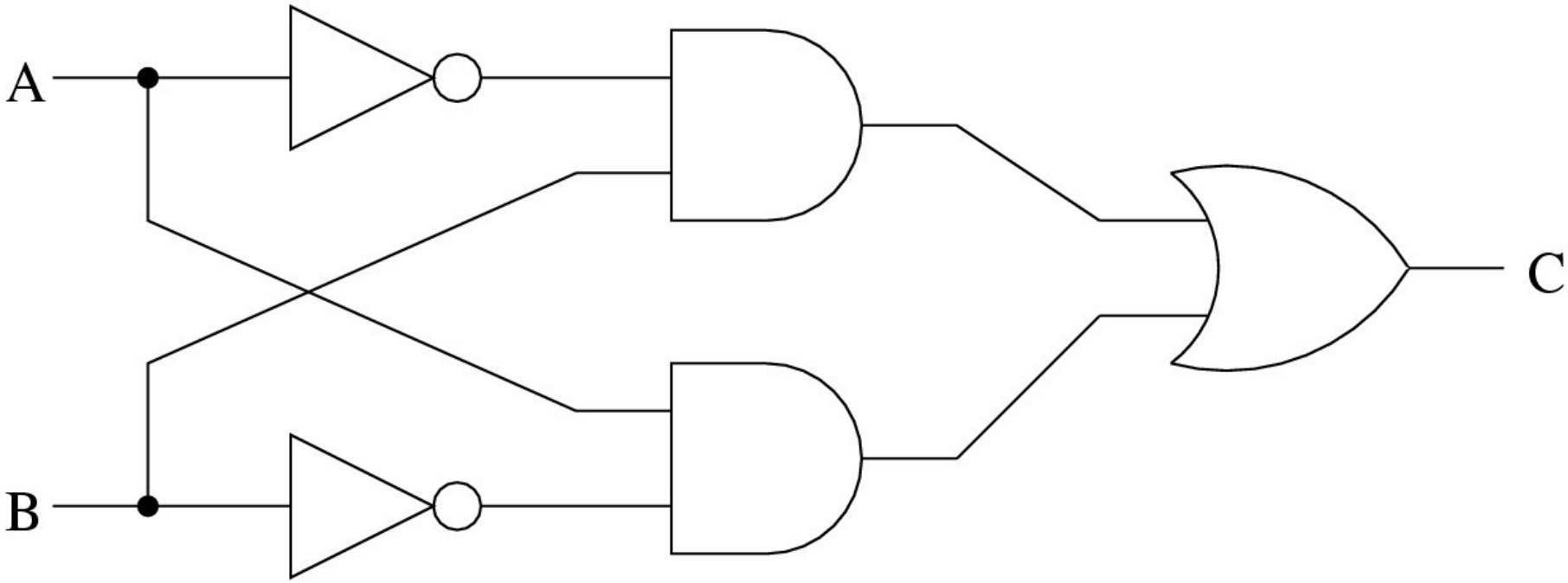


- Algebraic Notation: $C = A \oplus B$
- Truth Table:

A	B		C
0	0		0
0	1		1
1	0		1
1	1		0

How would we implement this with AND/OR/NOT gates?

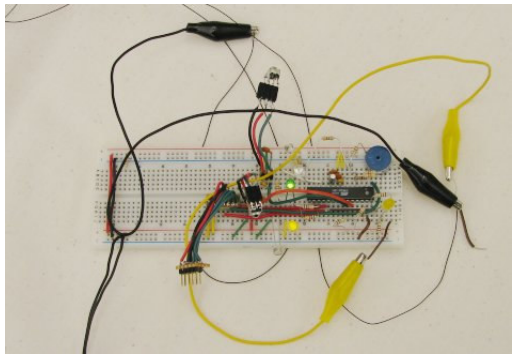
An XOR Implementation



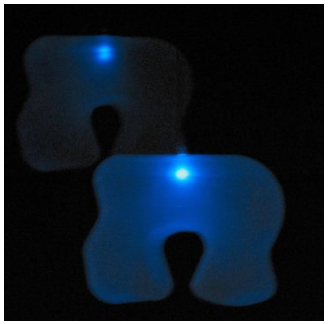
Next Time

- Some notes on Boolean Algebra
- DeMorgan's Laws
- Multiplexers
- Demultiplexers
- Circuit reduction with Karnaugh Maps
- Readings from play-hookey.com (see the schedule)

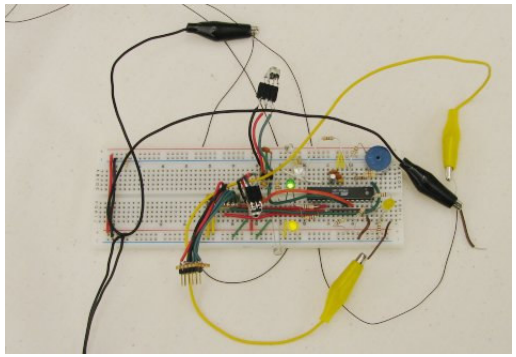
Bions …

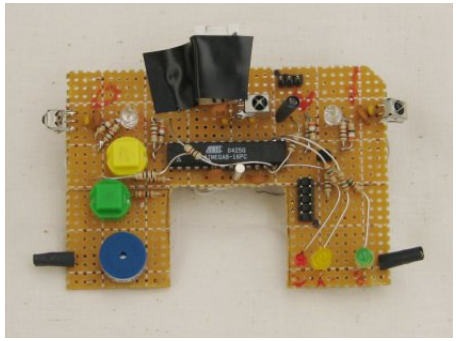


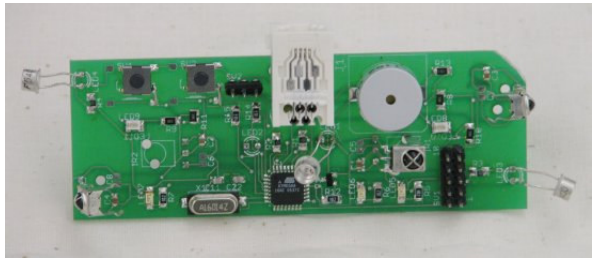
?

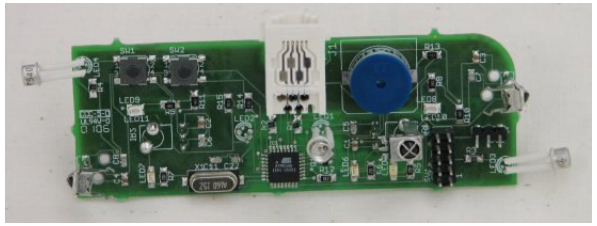


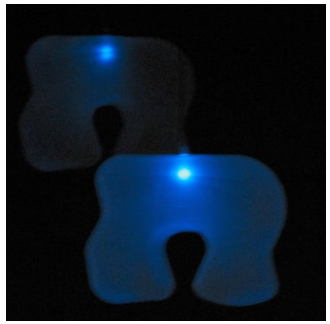
Bions



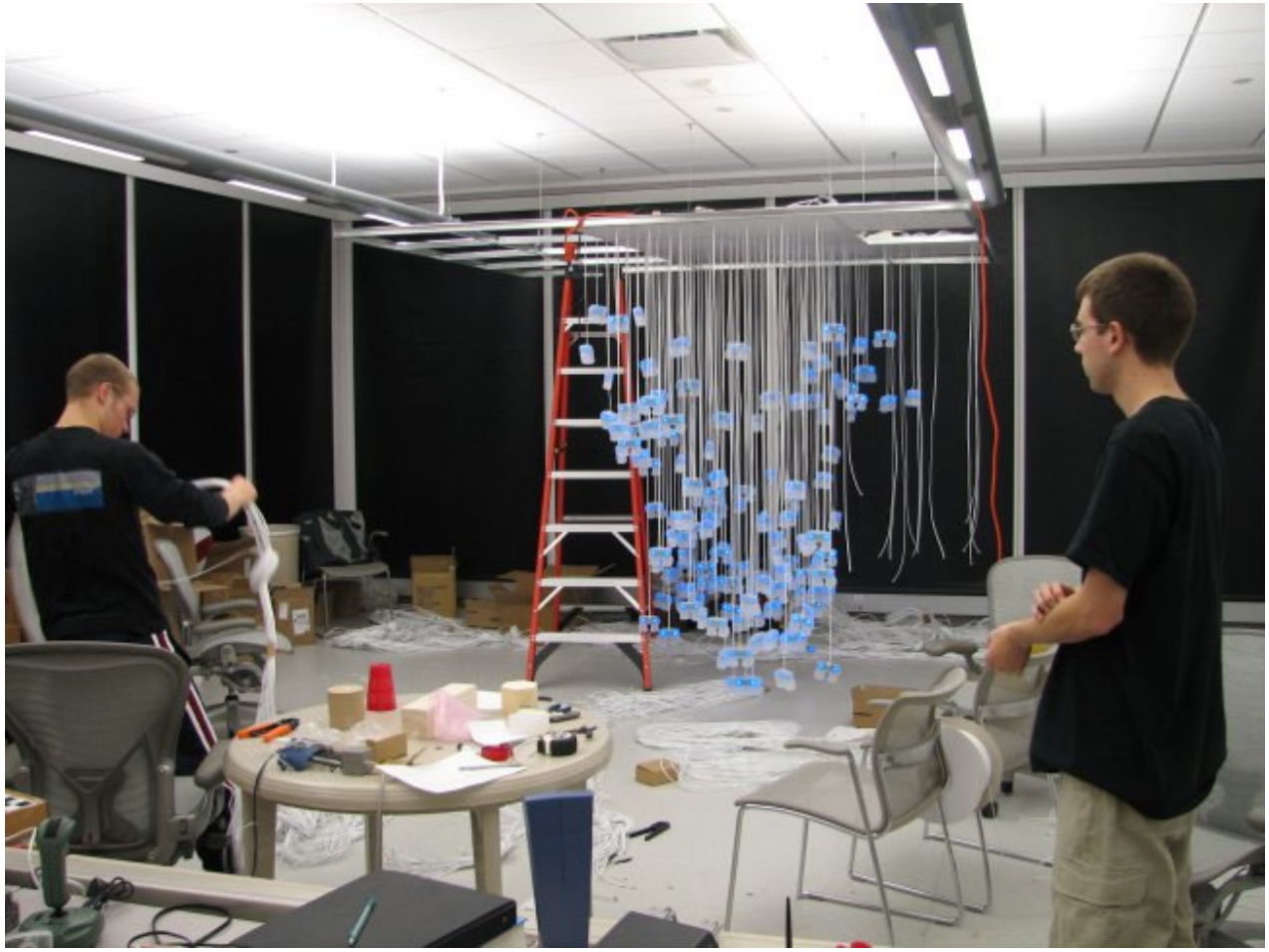




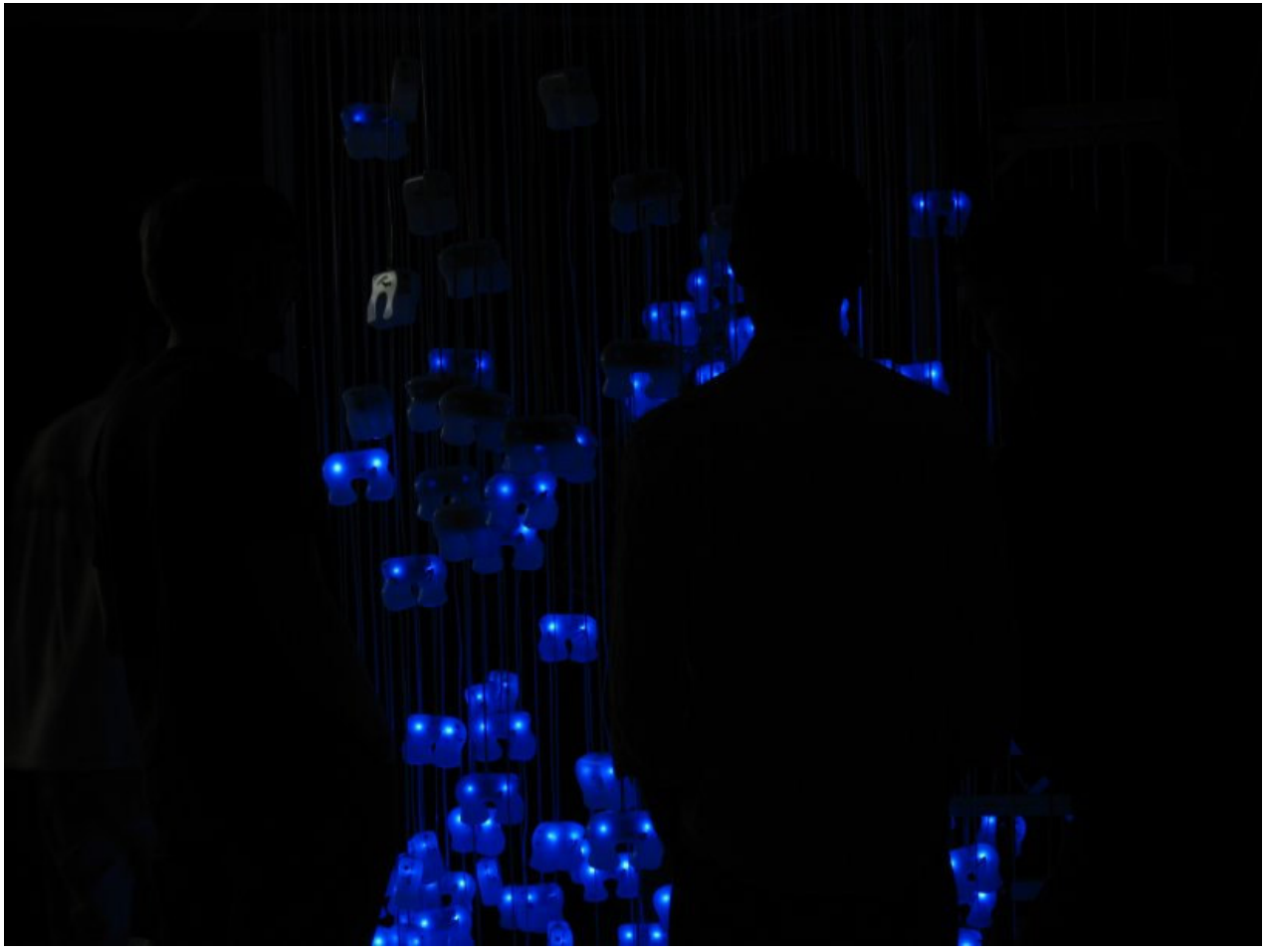




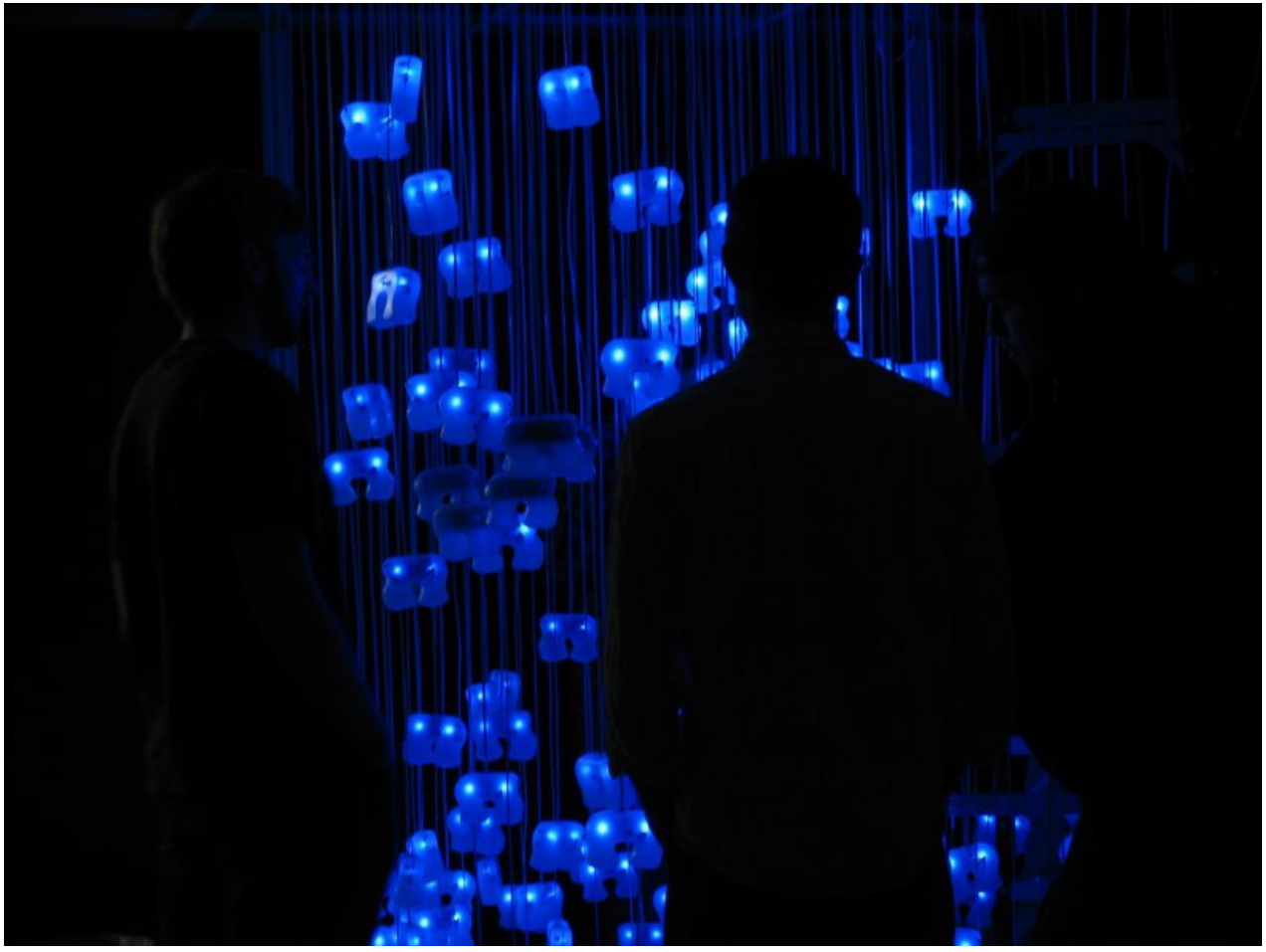
Bion Installation



Bion

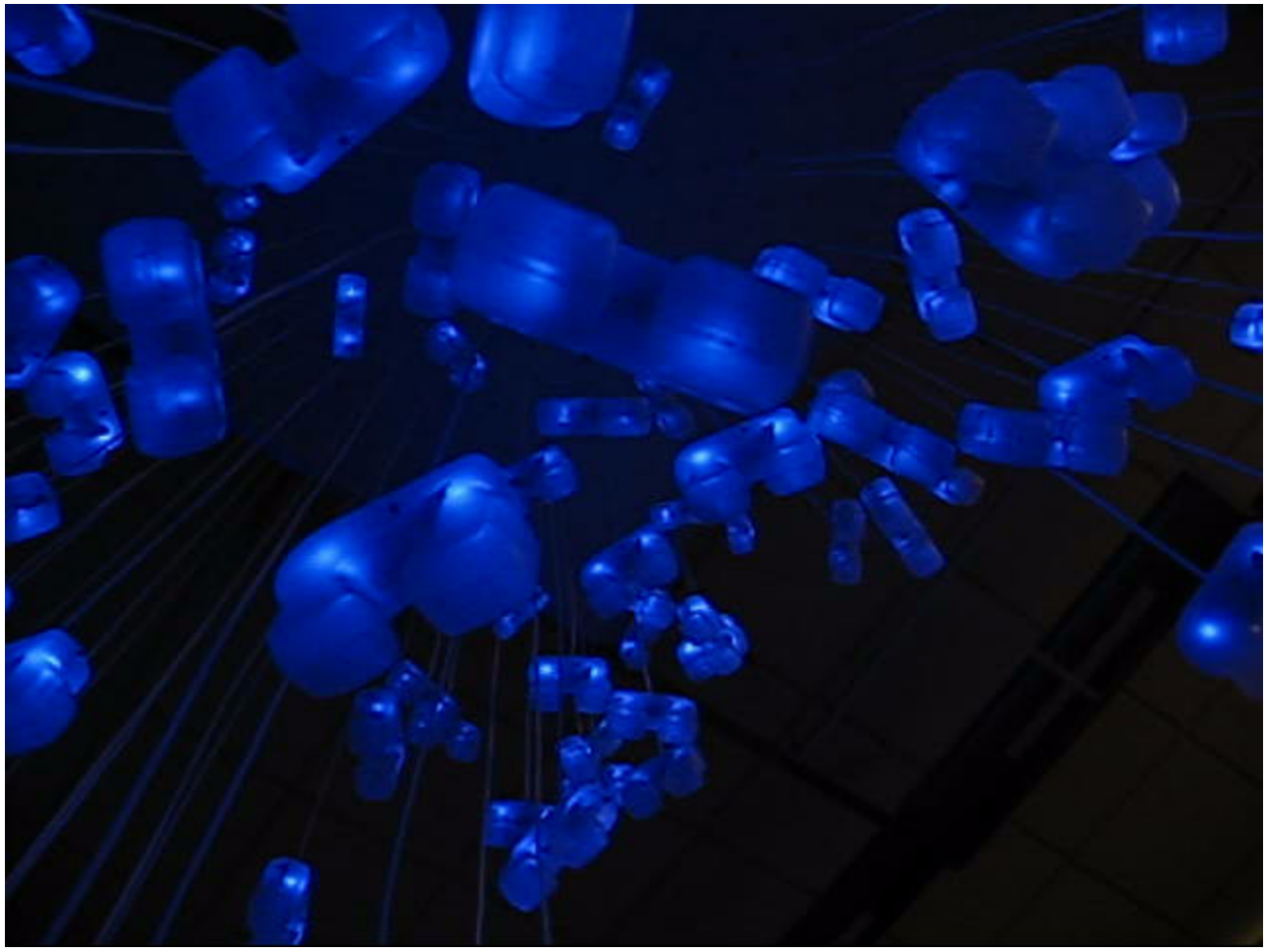


Bion



Bion

Bion



Last Time

- Transistors (analog) to logic (digital)
- Basic gates: AND, OR, NOT
- Other gates: XOR, NOR, NAND

These tools form the basis of computation in digital computers

Today

More complicated circuits

Circuit reduction tools:

- Boolean Algebra
- DeMorgan's Laws
- Karnaugh Maps

Administrivia

- Homework 1 goes out tonight; it is due in one week
- A second test email message will go out announcing the homework. If you don't get the message – then we need to fix it ASAP

- General announcements to email list?

An Example

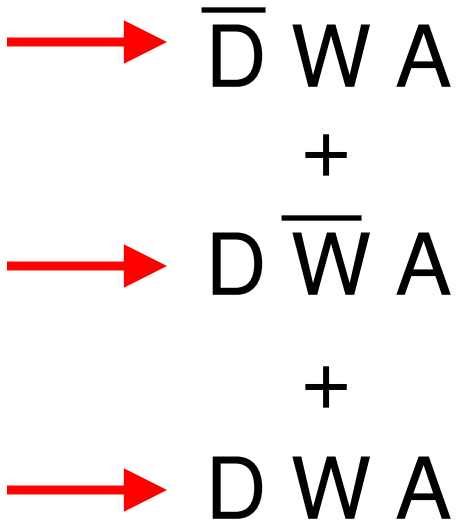
Problem: implement an alarm system

- There are 3 inputs:
  - Door open (“1” represents open)
  - Window open
  - Alarm active (“1” represents active)
- And one output:
  - Siren is on (“1” represents on) when either the door or window are open – but only if the alarm is active

What is the truth table?

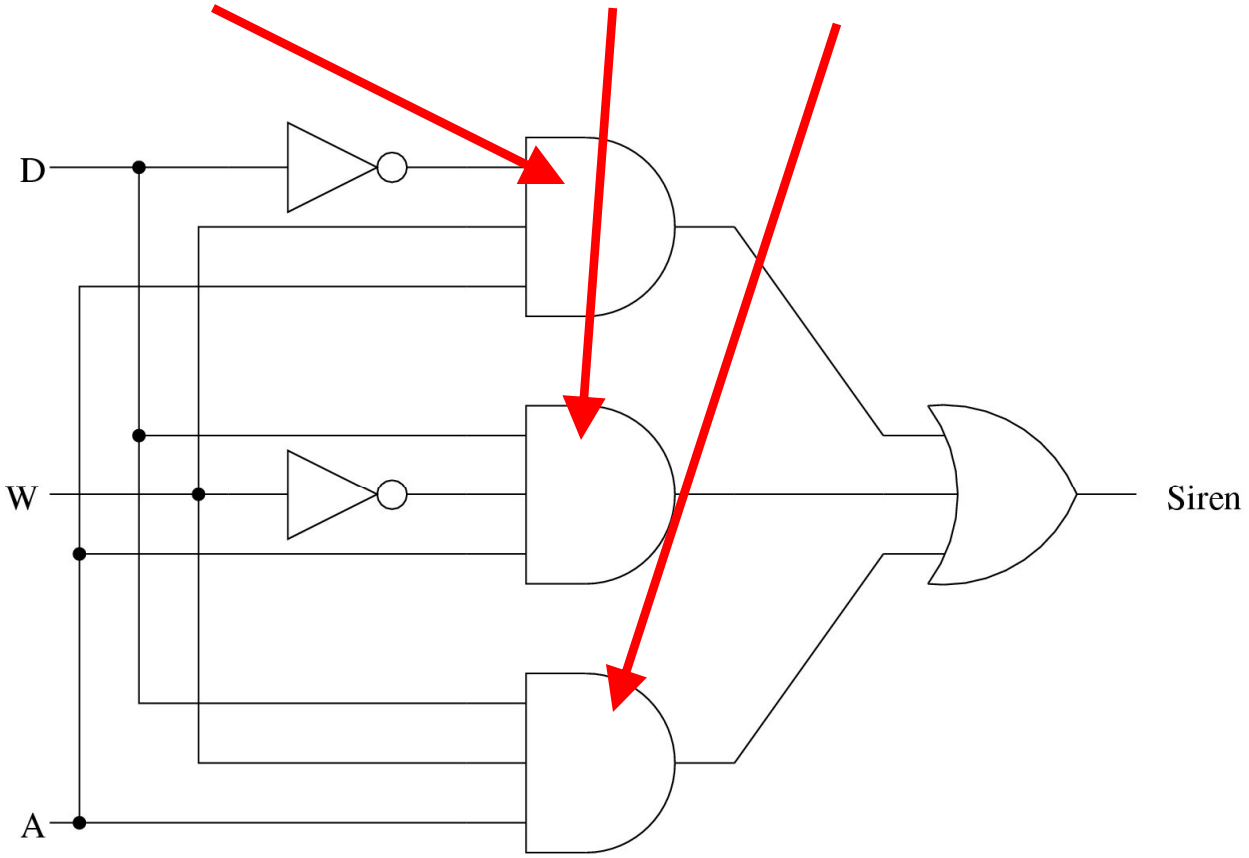
Alarm Example: Truth Table

D	W	A		Siren
0	0	0		0
0	0	1		0
0	1	0		0
0	1	1		1
1	0	0		0
1	0	1		1
1	1	0		0
1	1	1		1



Alarm Example: Circuit

$$\text{Siren} = \overline{D}\ W\ A + D\ \overline{W}\ A + D\ W\ A$$



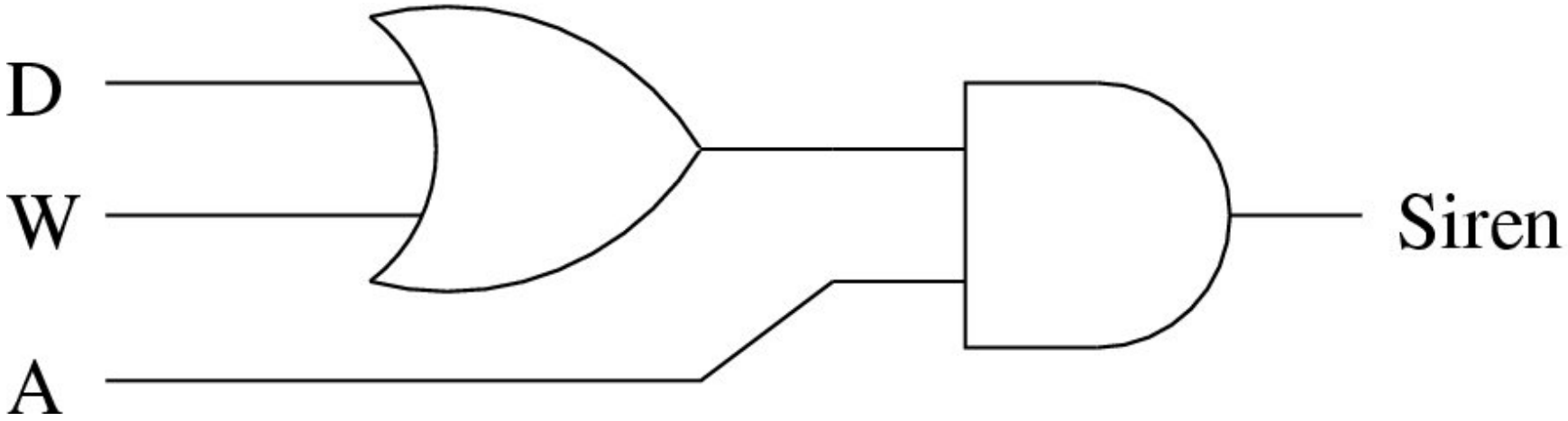
Alarm Example: Circuit

Is a simpler circuit possible?

Alarm Example: Truth Table

D	W	A		Siren
0	0	0		0
0	0	1		0
0	1	0		0
0	1	1		1
1	0	0		0
1	0	1		1
1	1	0		0
1	1	1		1

Alarm: An Alternative Circuit



Boolean Algebra

- There are exactly two numbers in Boolean System: “0” and “1”
- You are already familiar with the “integers”: {… -2, -1, 0, 1, 2, …}
  - (and Integer Algebra)

Boolean Algebra

- Like the integers, Boolean Algebra has the following operators:

		Integers	Boolean
+		addition	OR
*		product	AND
inverse		negation	NOT

NOT Operator

Definition:

- $\overline{0} = 0' = 1$
- $\overline{1} = 1' = 0$

NOTE: this is identical to our truth table (just a slightly different notation)

Suppose that "X" is a Boolean variable, then:

- $\overline{\overline{X}} = X'' = X$

OR (+) Operator

Definition:

- 0+0 = 0
- 0+1 = 1
- 1+0 = 1
- 1+1 = 1

OR (+) Operator

Suppose “X” is a Boolean variable, then:

- 0 + X = X
- 1 + X = 1
- X + X = X
- X + X’ = 1

AND (*) Operator

Definition:

- 0*0 = 0
- 0*1 = 0
- 1*0 = 0
- 1*1 = 1

AND (*) Operator

Suppose “X” is a Boolean variable, then:

- 0 * X = 0
- 1 * X = X
- X * X = X
- X * X’ = 0

Boolean Algebra Rules: Precedence

The AND operator applies before the OR operator:

$A * B + C = (A * B) + C$

$A + B * C = A + (B * C)$

Boolean Algebra Rules: Association Law

If there are several AND operations, it does not matter which order they are applied in:

$$A * B * C = (A * B) * C = A * (B * C)$$

Boolean Algebra Rules: Association Law

Likewise for the OR operator:

$$A + B + C = (A + B) + C = A + (B + C)$$

Boolean Algebra Rules: Distributive Law

AND distributes across OR:

$A * (B + C) = (A * B) + (A * C)$

$A + (B * C) = (A + B) * (A + C)$

Boolean Algebra Rules: Commutative Law

Both AND and OR are symmetric operators (the order of the variables does not matter):

$A + B = B + A$

$A * B = B * A$

DeMorgan's Laws

(A * B)' = A' + B'

How do we convince ourselves that this is true?

Proof by Truth Table

A	B		(A * B)'	A' + B'
0	0		1	1
0	1		1	1
1	0		1	1
1	1		0	0

NOTE: change in the NOT notation

DeMorgan's Laws (cont)

$(A + B)' = A' * B'$

Proof by Truth Table

A	B		(A + B)'	A' * B'
0	0		1	1
0	1		0	0
1	0		0	0
1	1		0	0

Alarm Example: Truth Table

D	W	A		Siren
0	0	0		0
0	0	1		0
0	1	0		0
0	1	1		1
1	0	0		0
1	0	1		1
1	1	0		0
1	1	1		1

D' W A

+

D W' A

+

D W A

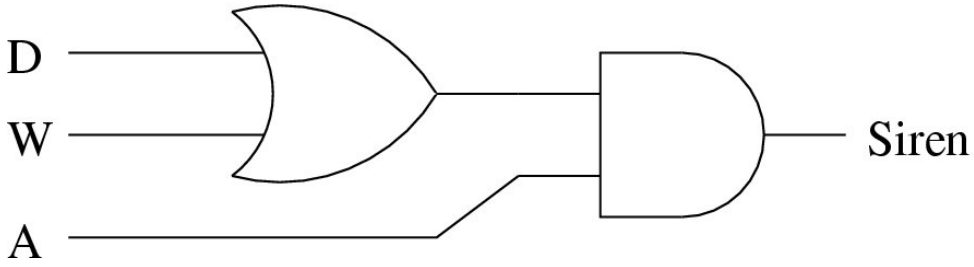
Reduction with Algebra

D’ W A + D W’ A + DWA	
= D’ W A + D W’ A + DWA + DWA	X + X = X
= D’ WA + DWA + DW’A + DWA	Commutative Law
= D’ WA + DWA + W’DA + WDA	“
= (D’ + D) WA + (W’ + W) DA	Assoc +Dist Laws
= 1* W A + 1* DA	X + X’ = 1

Reduction with Algebra (cont)

1* W A + 1* DA	
= WA + DA	X * 1 = X
= (W + D) * A	Distributive Law

We have the same circuit as before!



Karnaugh Maps

Geometric technique for reducing circuits

- Step 1: Construct the map
- Step 2: Fill in the output
- Step 3: Identify “clusters” in the map
- Step 4: Read out the terms

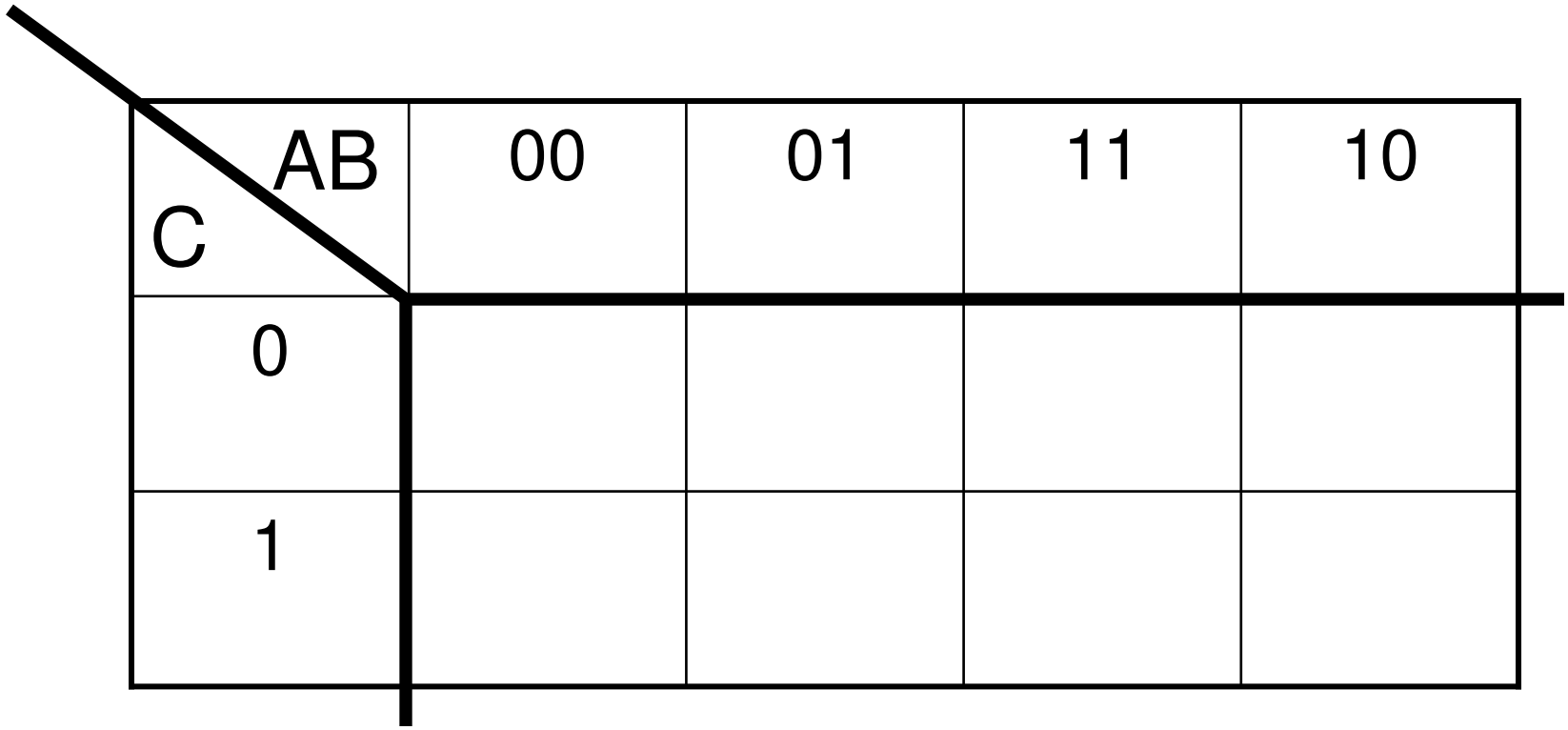
Suppose We Have the Following Truth Table:

A	B	C		D
0	0	0		0
0	0	1		1
0	1	0		1
0	1	1		1
1	0	0		0
1	0	1		0
1	1	0		1
1	1	1		1

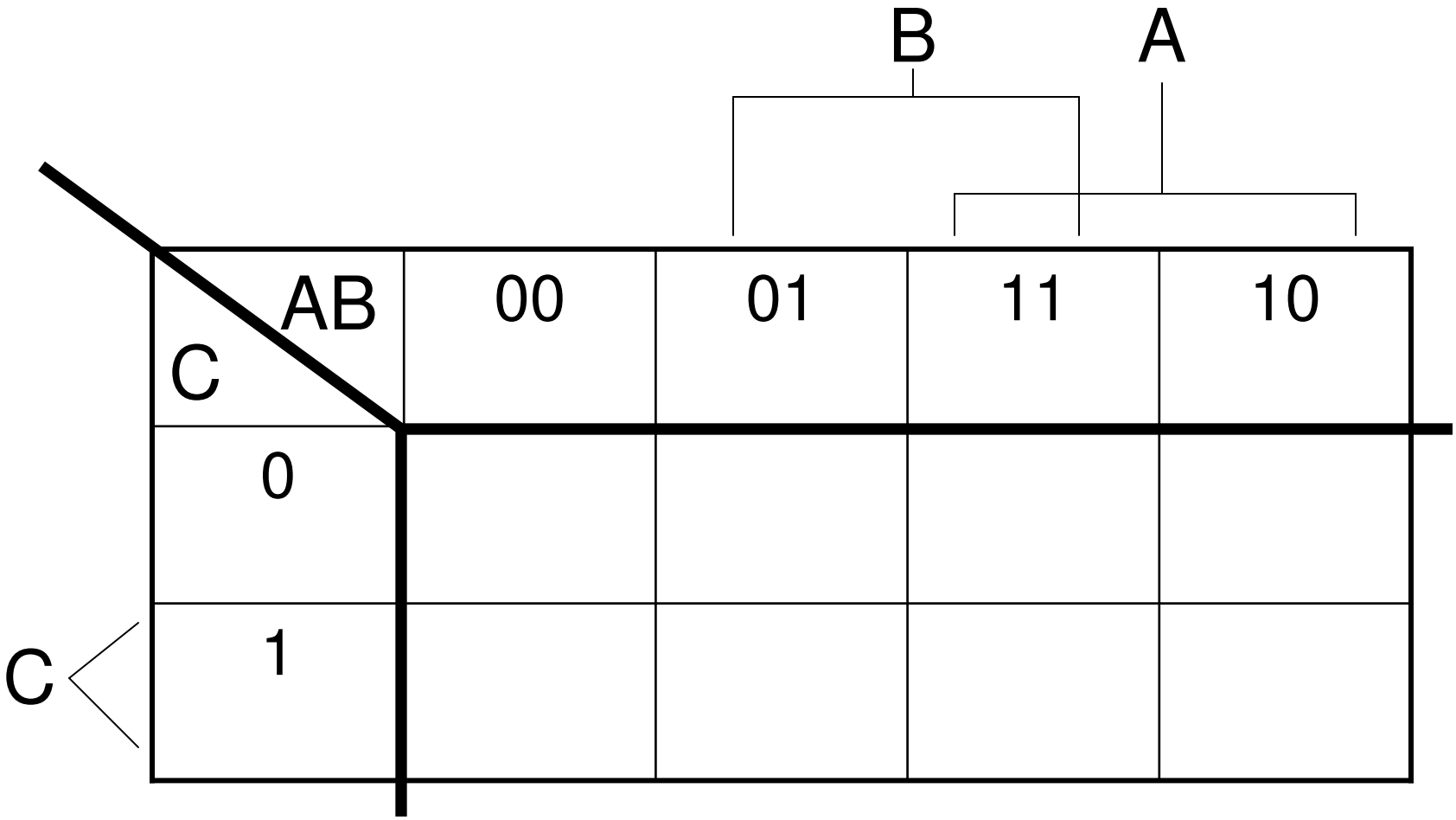
3 Inputs

1 Output

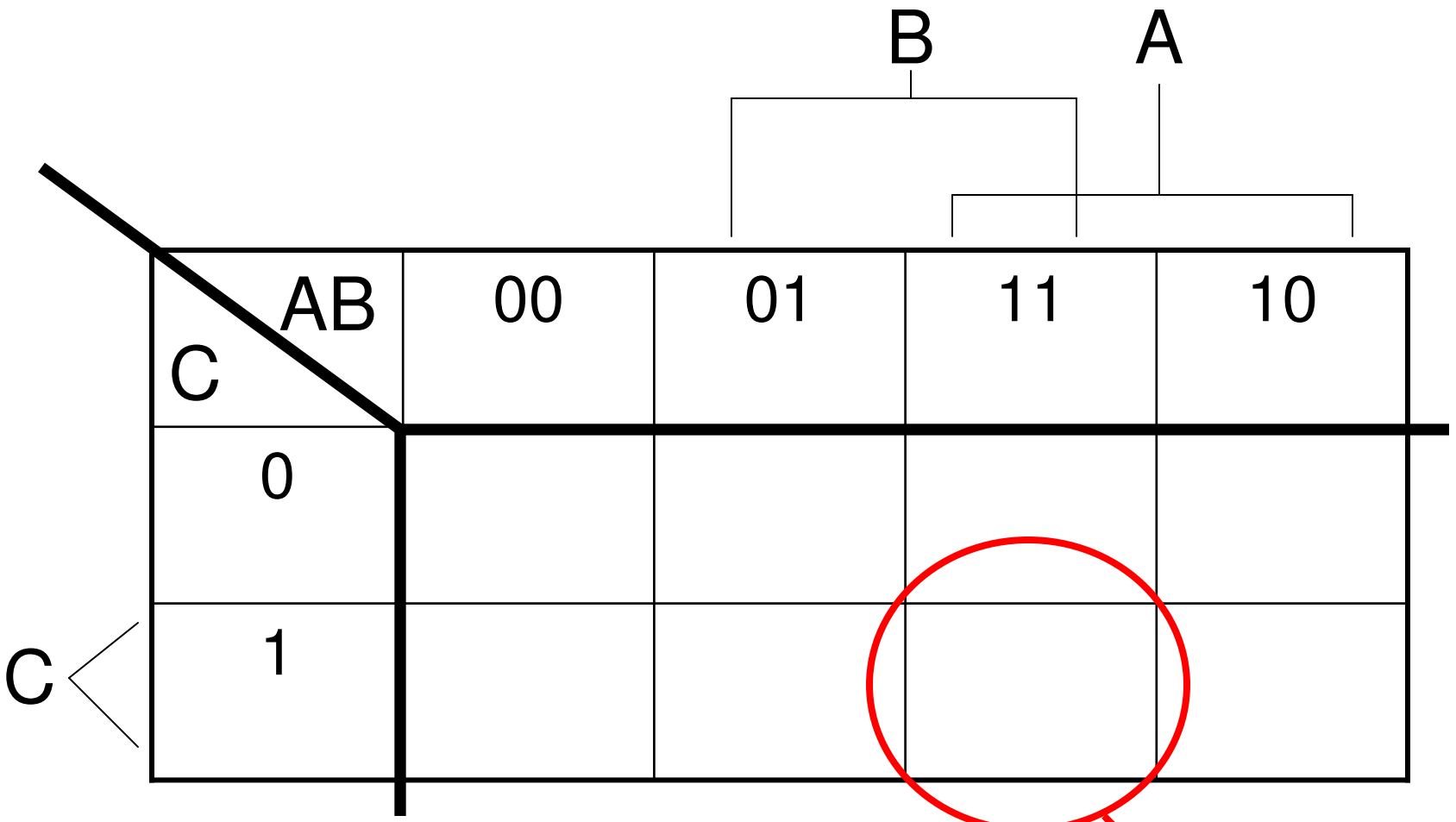
Karnaugh Maps
Step 1: Construct Map



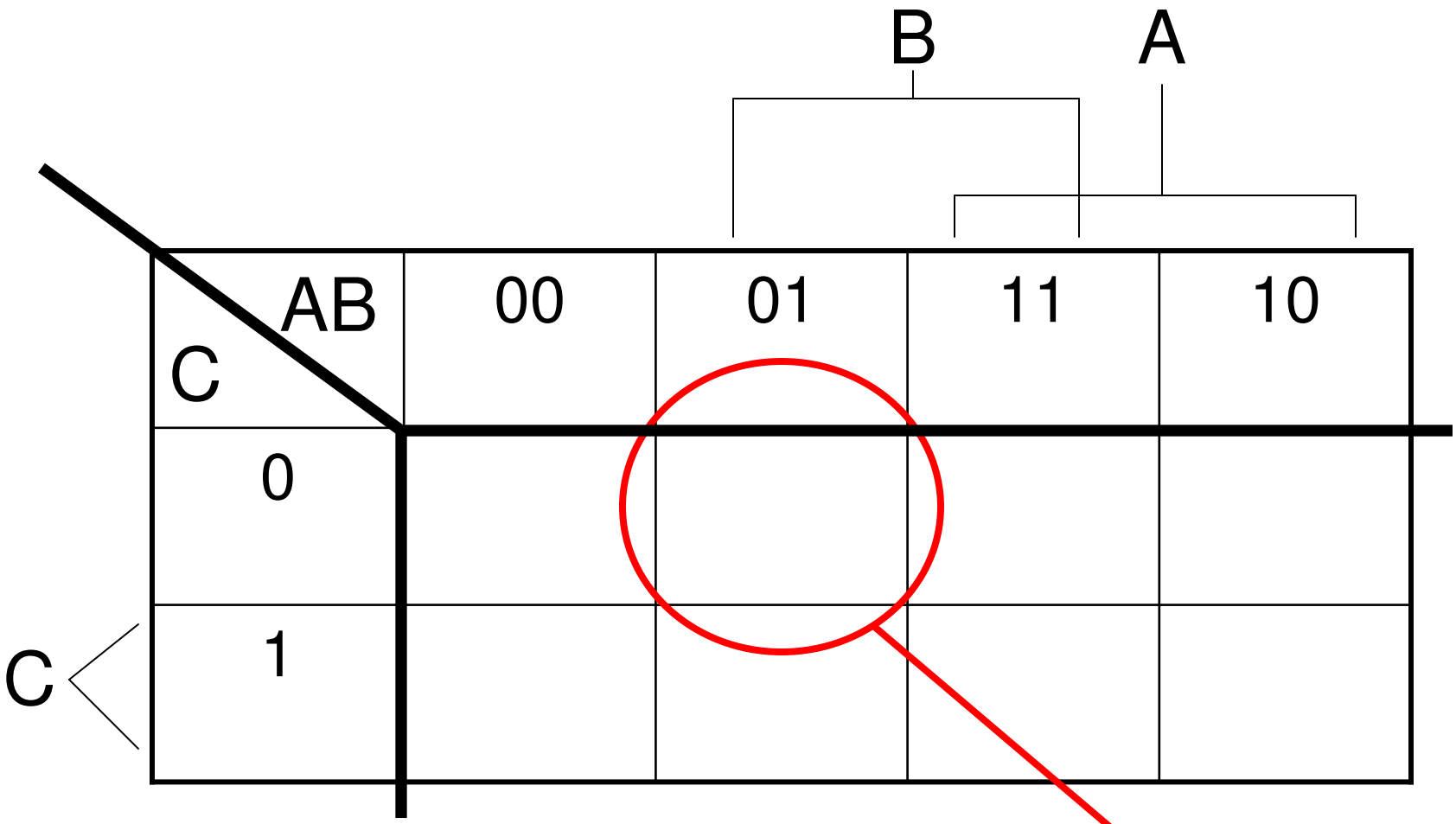
Karnaugh Maps
Step 1: Construct Map



Karnaugh Maps
Step 1: Construct Map



Karnaugh Maps
Step 1: Construct Map



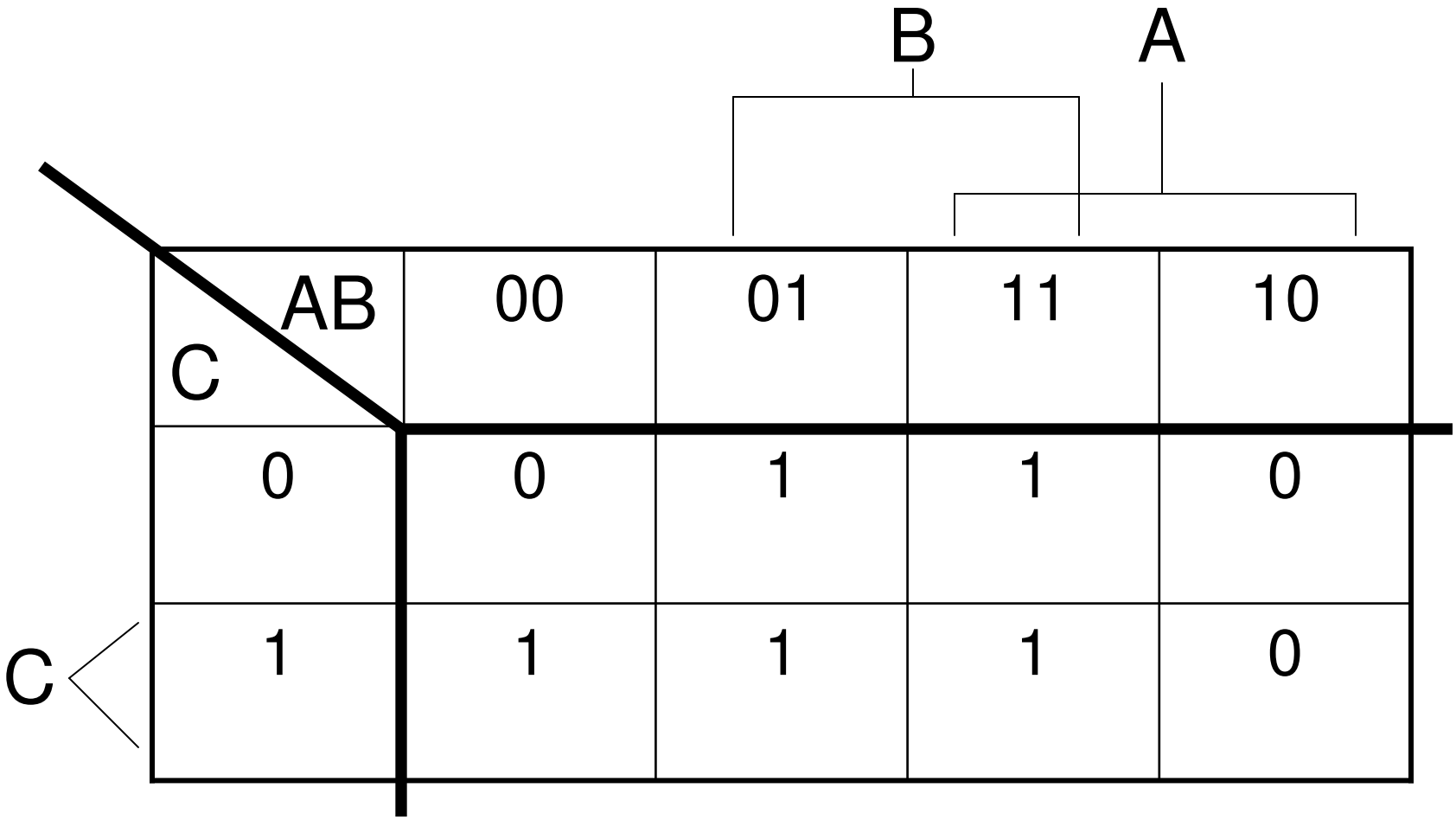
A’ B C’

Karnaugh Maps
Step 2: Insert Output Values

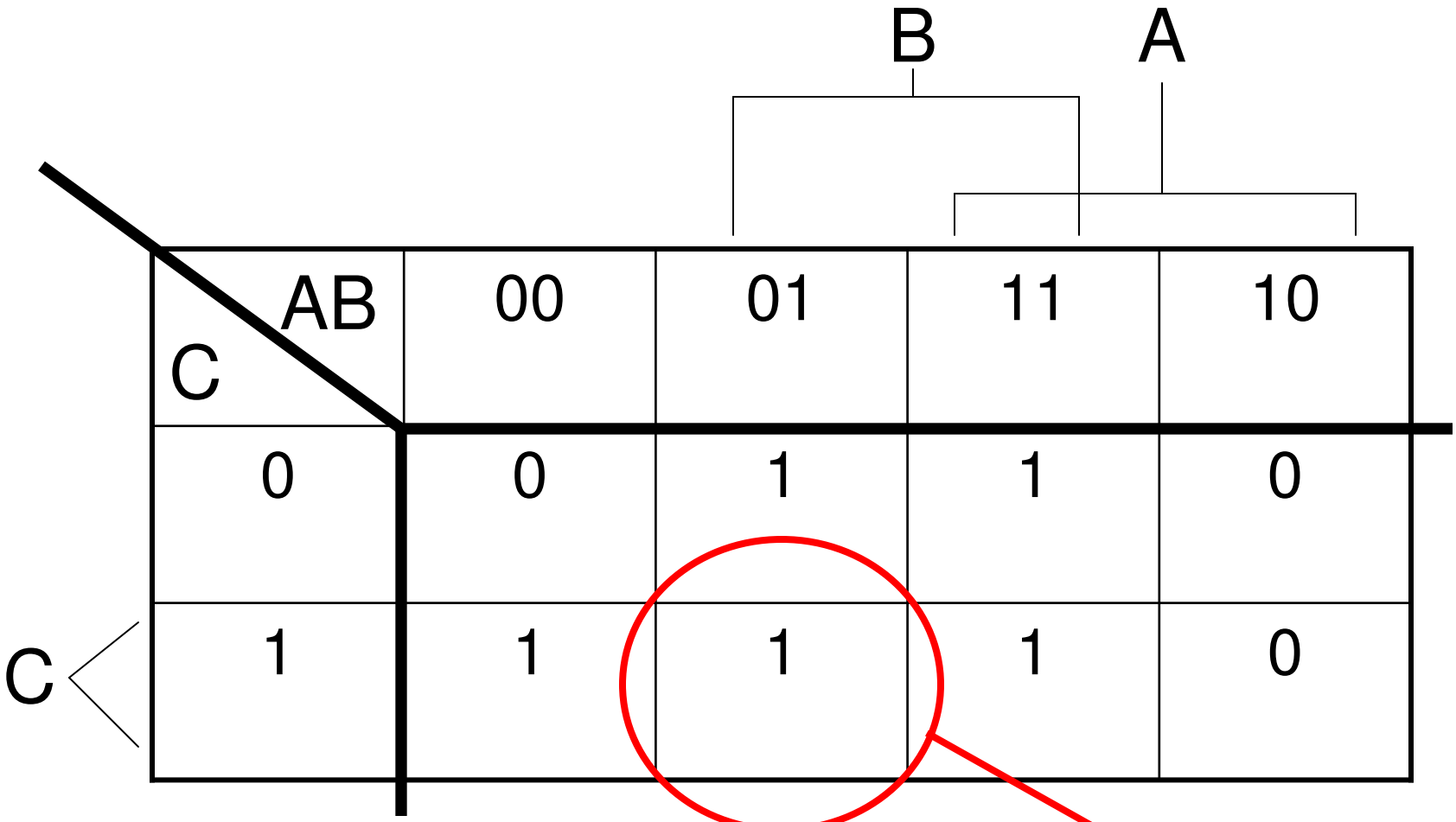
A	B	C		D
0	0	0		0
0	0	1		1
0	1	0		1
0	1	1		1
1	0	0		0
1	0	1		0
1	1	0		1
1	1	1		1

Enter output values into map

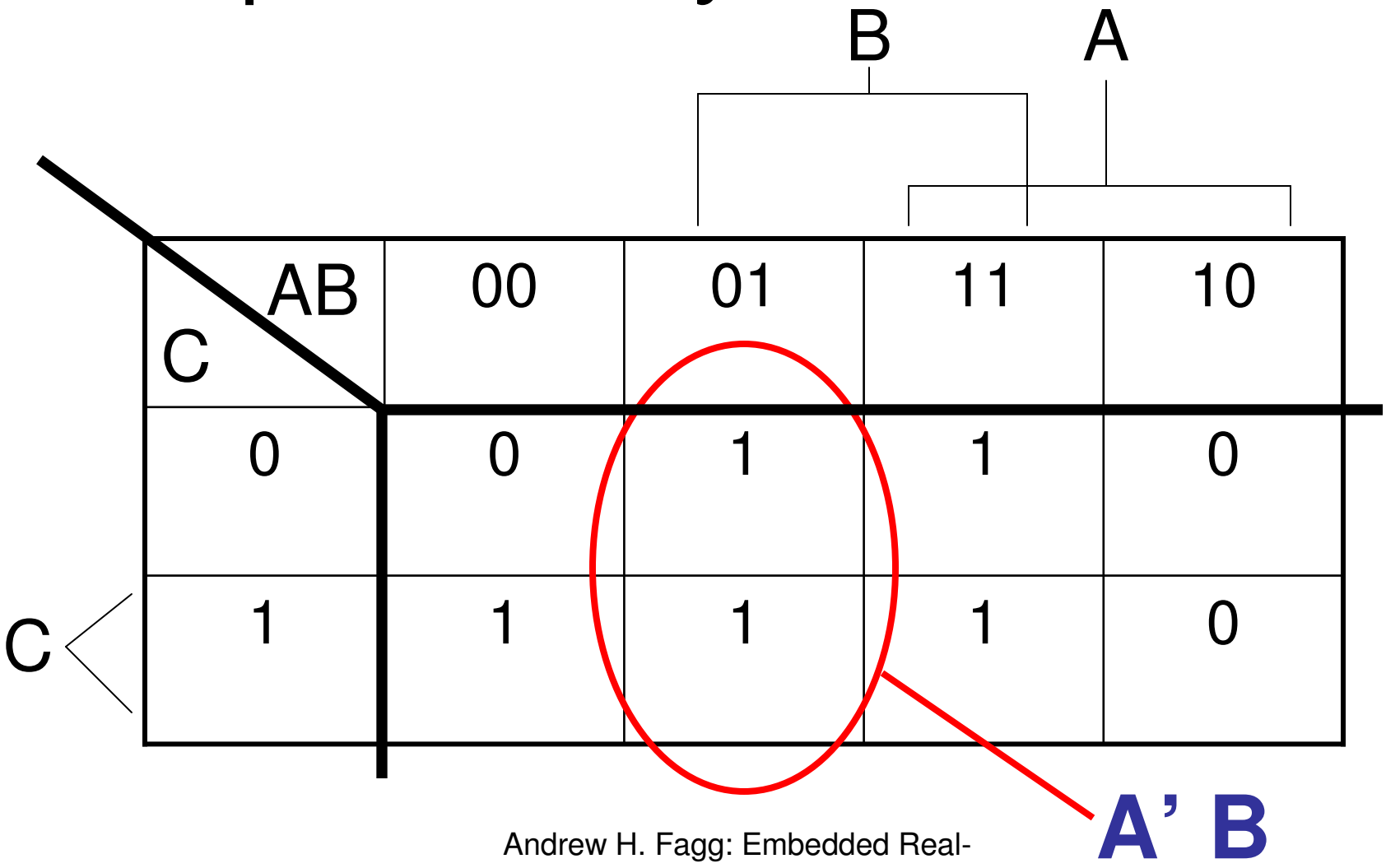
Karnaugh Maps
Step 2: Insert Output Values



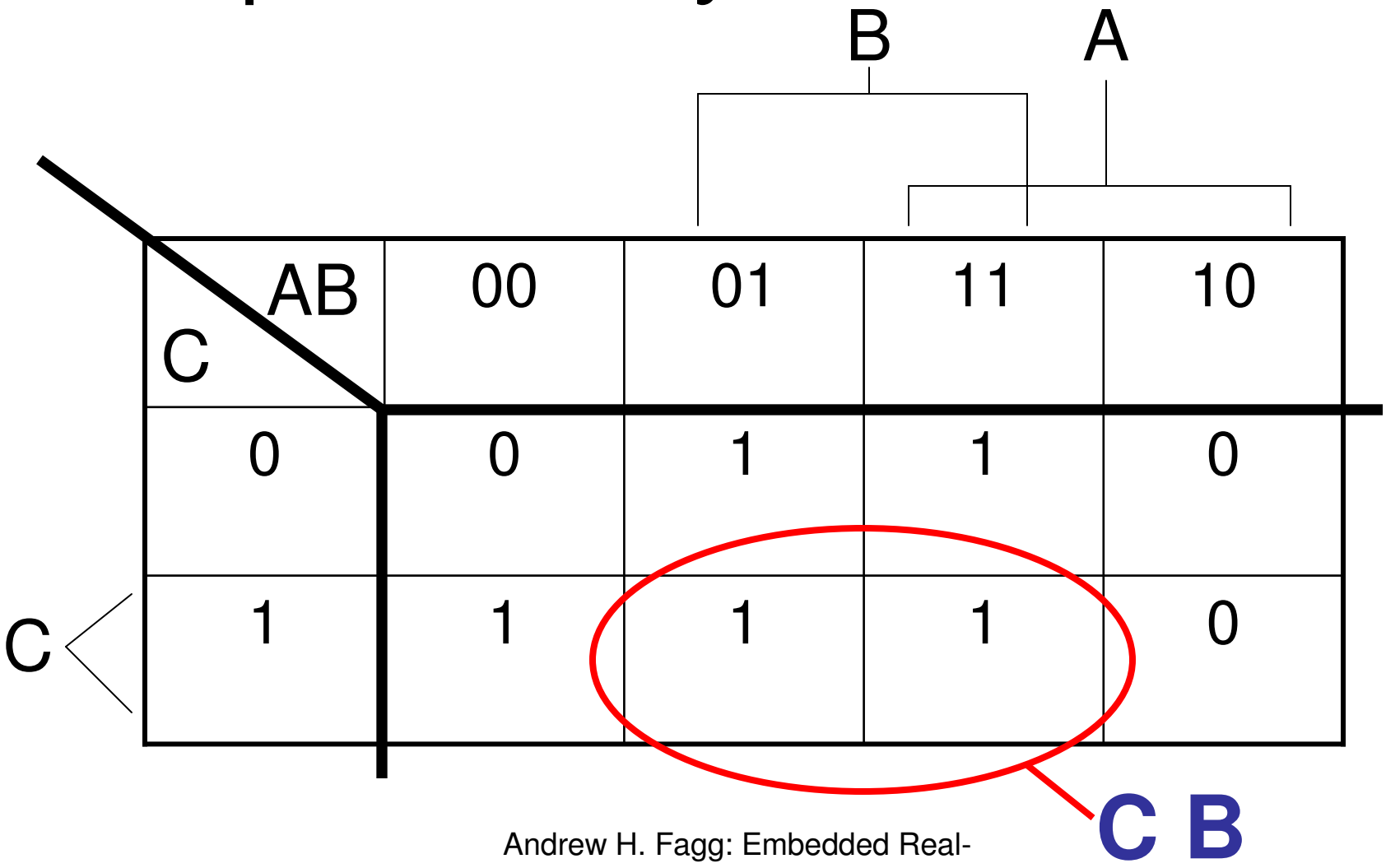
Karnaugh Maps
Step 3: Identify Clusters of 1's



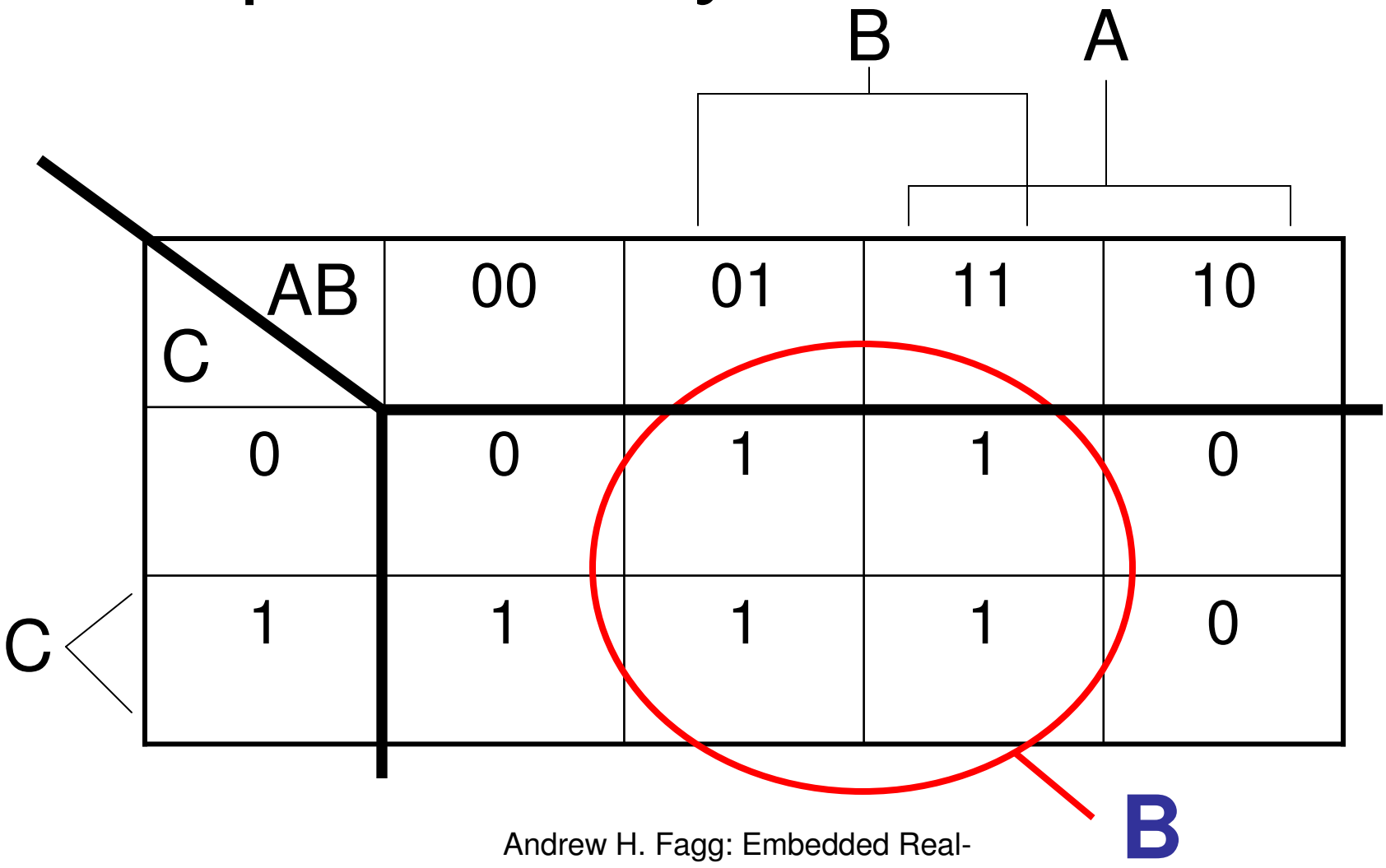
Karnaugh Maps
Step 3: Identify Clusters of 1's



Karnaugh Maps
Step 3: Identify Clusters of 1's



Karnaugh Maps
Step 3: Identify Clusters of 1's



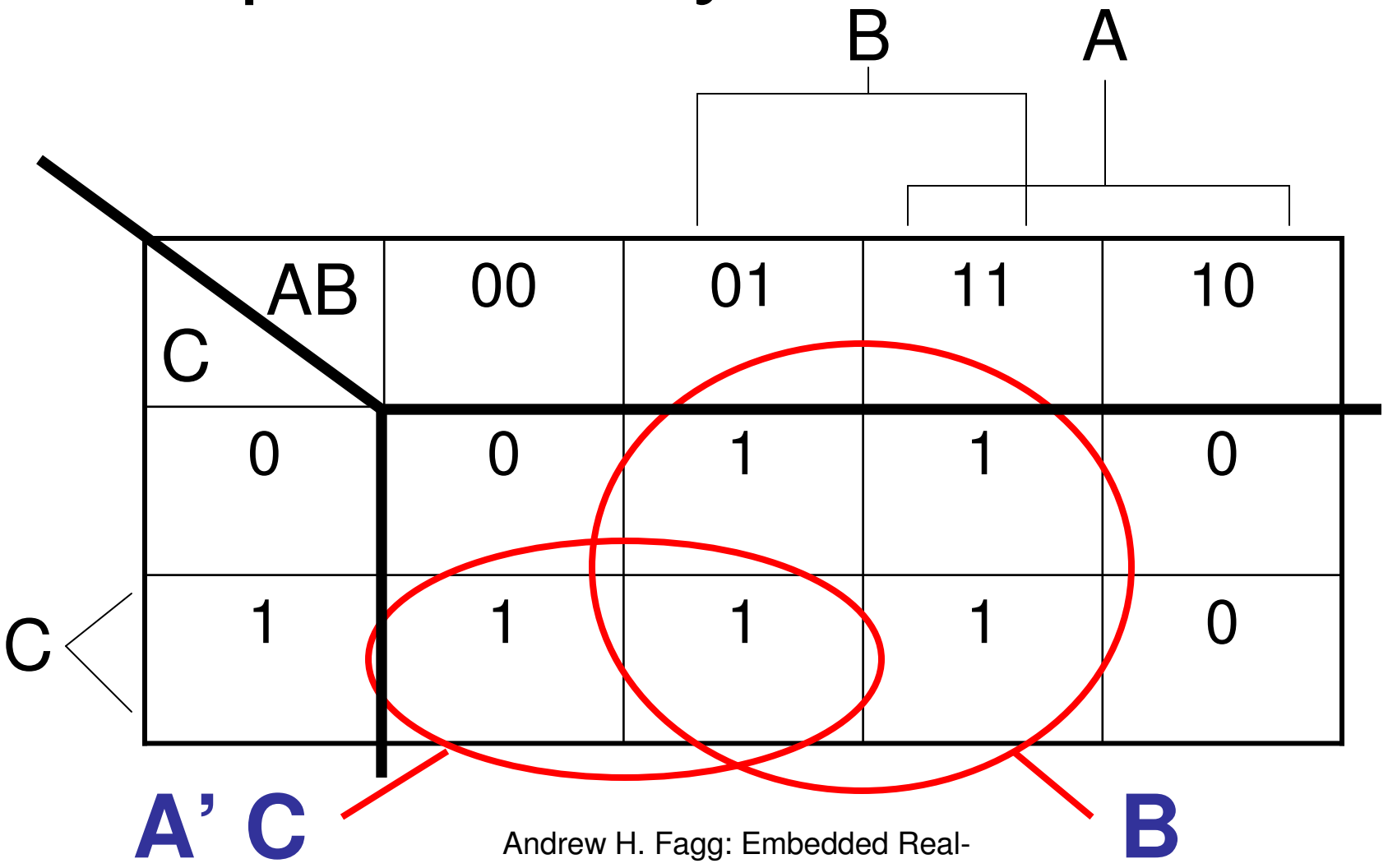
Karnaugh Maps
Step 3: Identify Clusters of 1's

Clustering Rules:

- A cluster may not contain a '0'
- All '1's must be covered

- But: it is OK for a '1' to be covered by more than one cluster

Karnaugh Maps
Step 3: Identify Clusters of 1's



Karnaugh Maps
Step 4: Read Out the Terms

"OR" the clusters together

Resulting logical rule:

B + A'C

Next Time

- More on Karnaugh Maps
- Multiplexers
- Demultiplexers

Thinking About Graduate School?

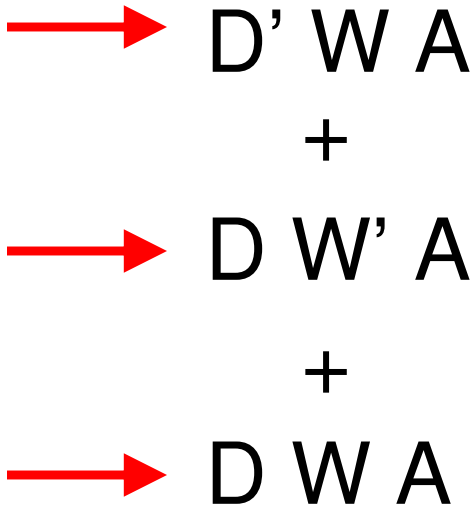
- Research Experiences for Undergraduates: Embedded Machine Learning Systems
  - www-symbiotic.cs.ou.edu/reu
- McNair Scholars Program
  - Supporting underrepresented groups in PhD programs
  - www.ou.edu/special/mcnair
- Computer Science PhD Fellowships
  - fellowships.cs.ou.edu

Administrivia

- Homework 1 is due this coming Tuesday
- My office hours today: 10:15-11:00

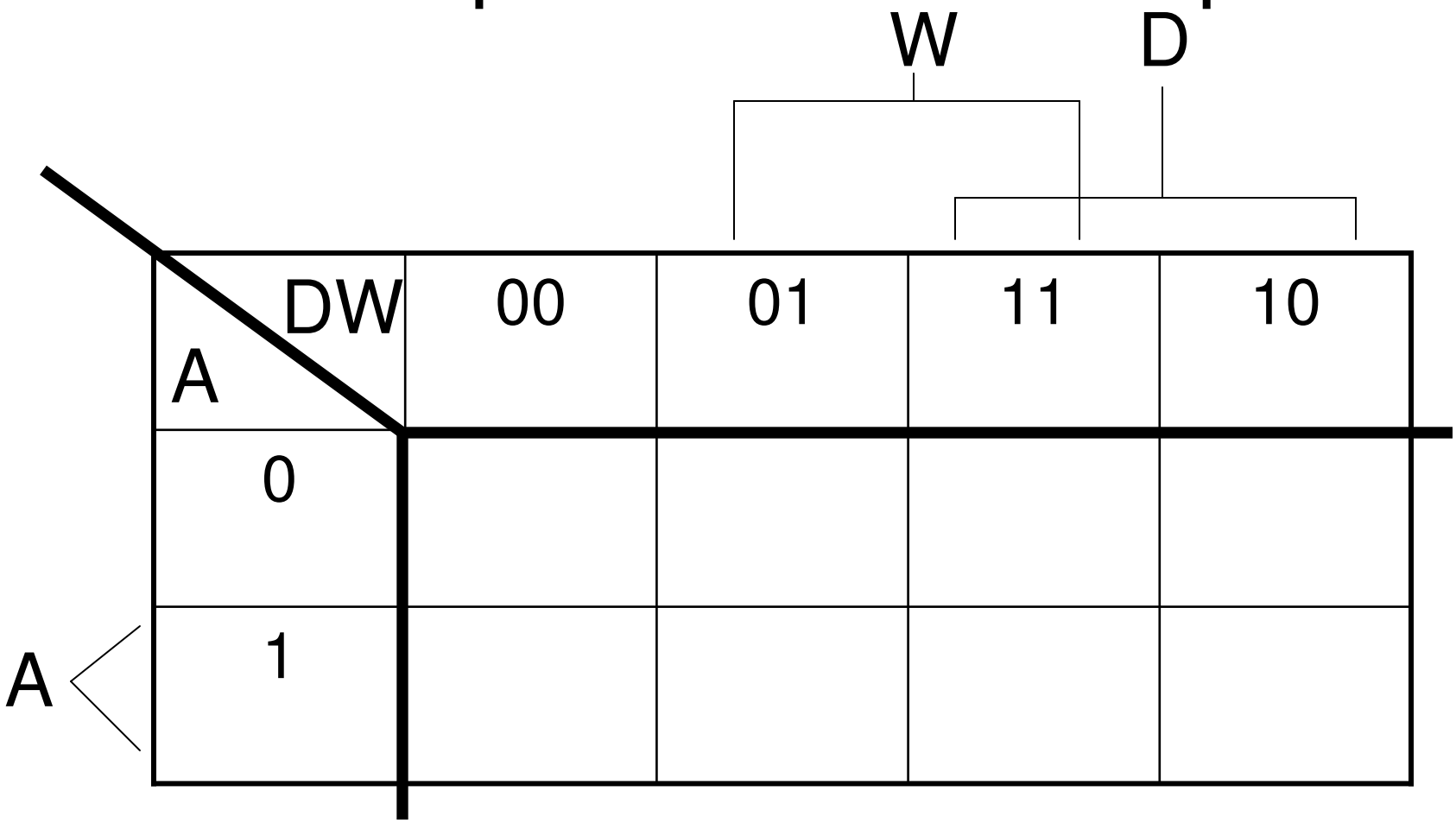
The Alarm Example (again)

D	W	A		Siren
0	0	0		0
0	0	1		0
0	1	0		0
0	1	1		1
1	0	0		0
1	0	1		1
1	1	0		0
1	1	1		1



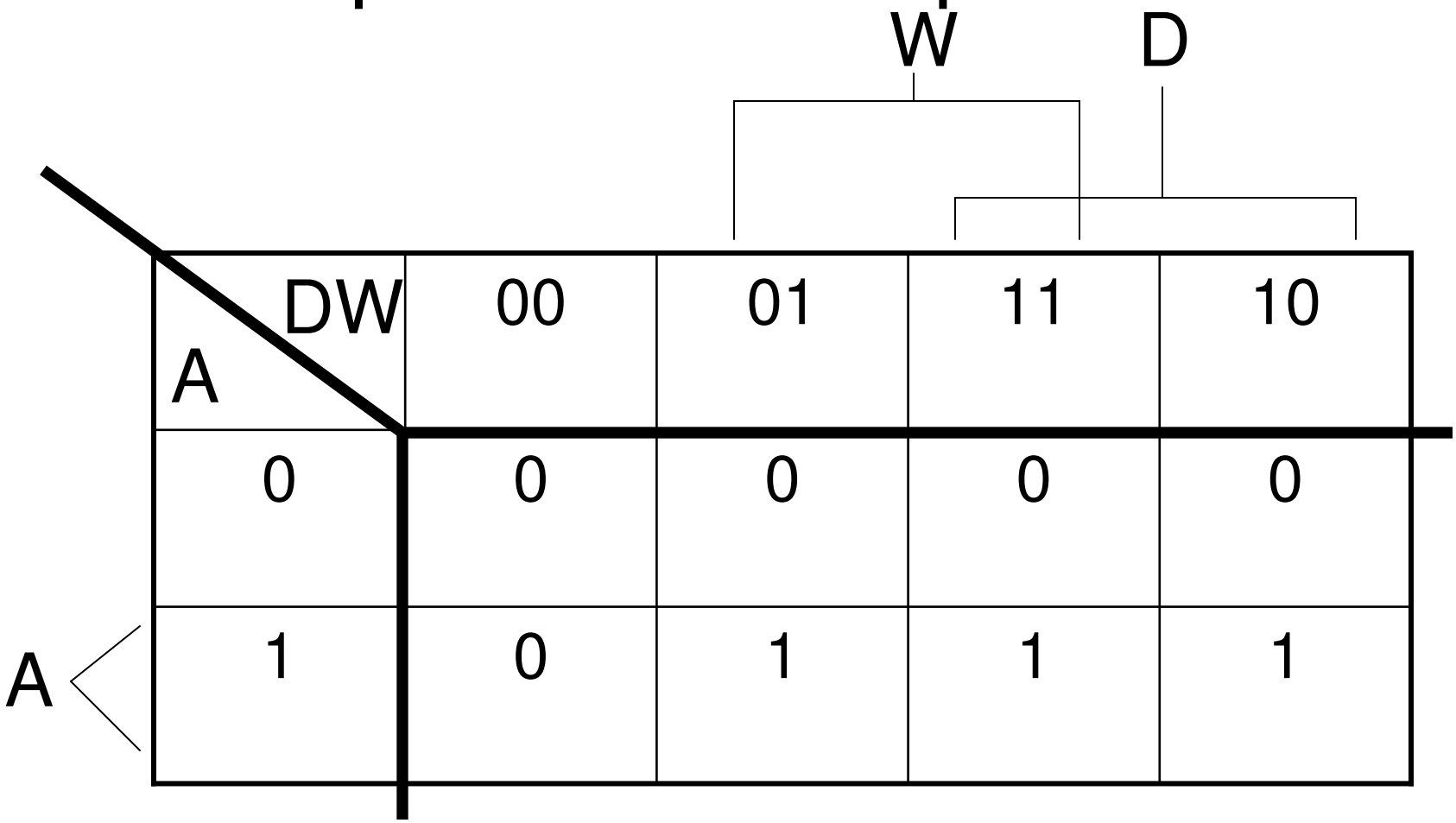
Karnaugh Maps

Step 1: Construct Map



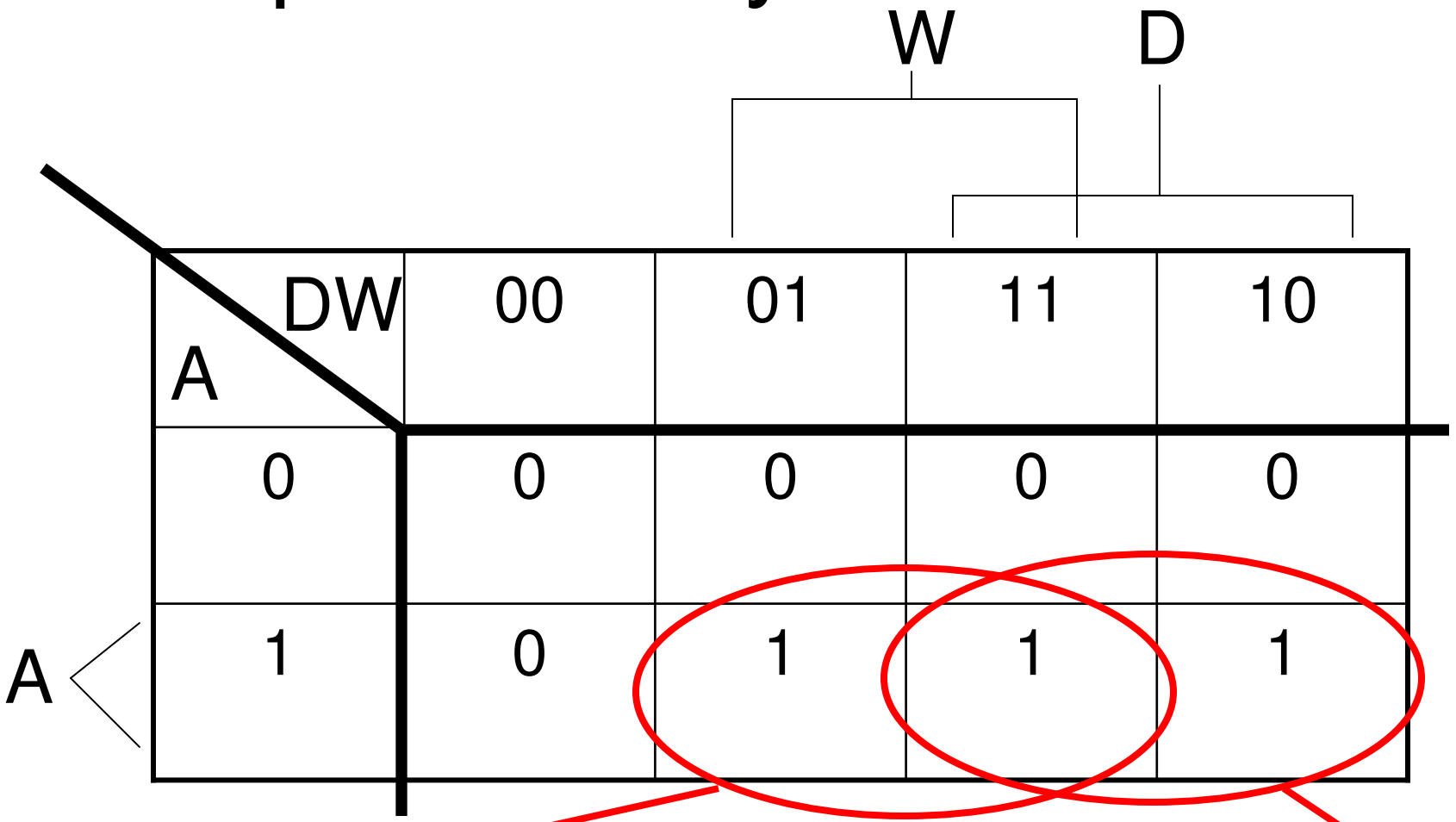
Karnaugh Maps

Step 2: Insert Output Values



Karnaugh Maps

Step 3: Identify Clusters of 1's



Karnaugh Maps
Step 4: Read Out the Terms

Resulting logical rule:

WA + DA

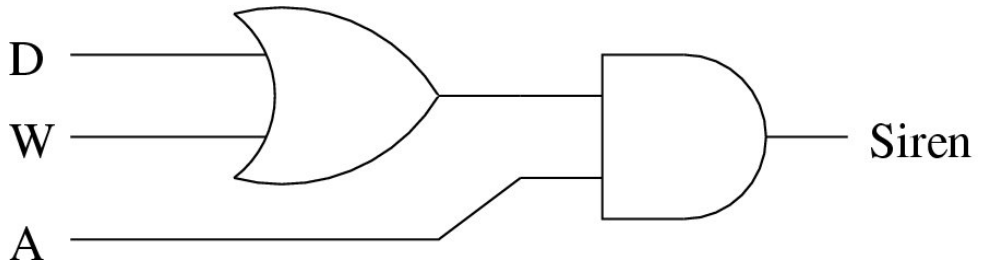
Karnaugh Maps
Step 4: Read Out the Terms

Resulting logical rule:

WA + DA

Which can be simplified to:

A * (W + D)



A New K-Map Example

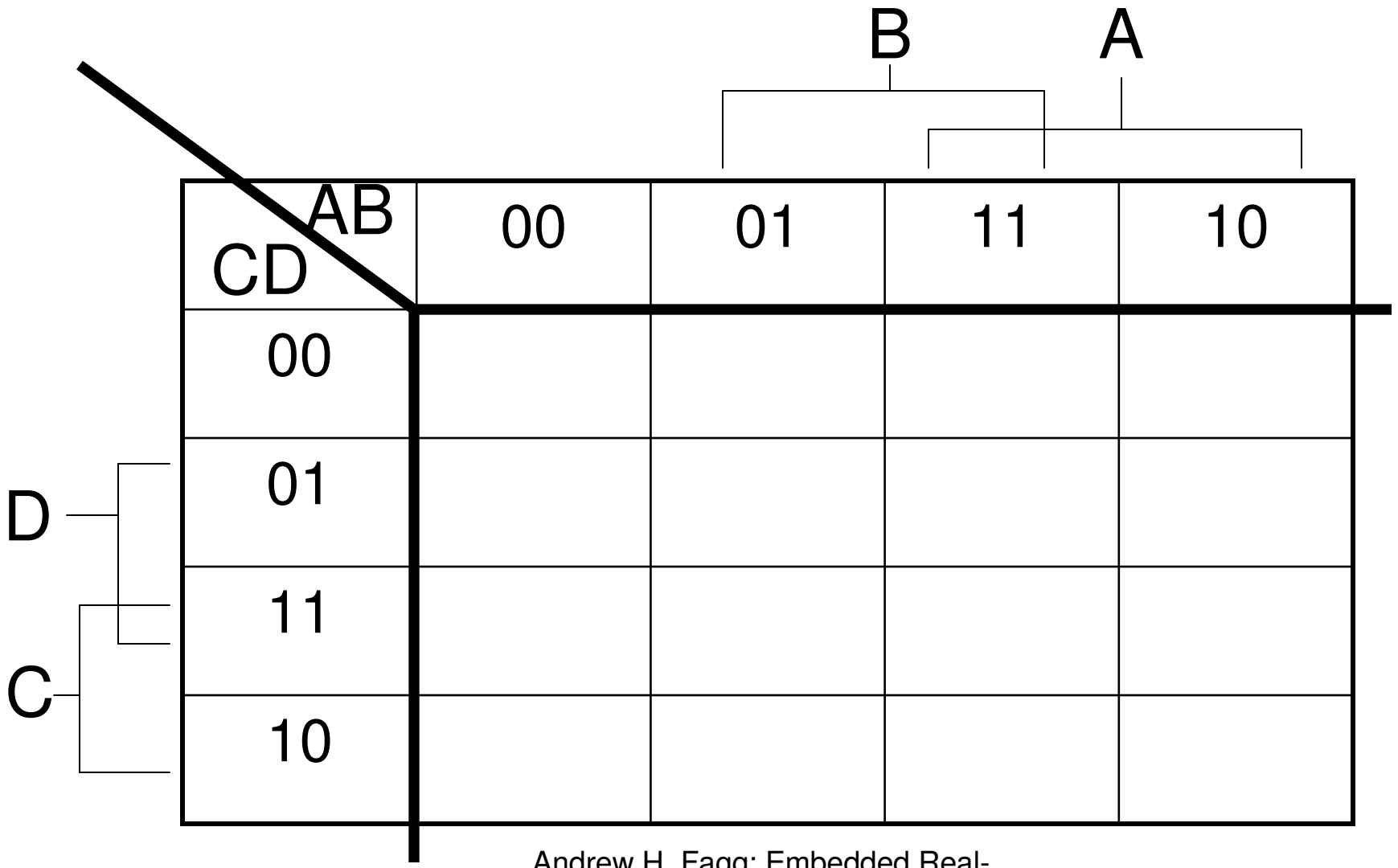
- Four input variables: A, B, C, D
- One output variable: E

Truth Table

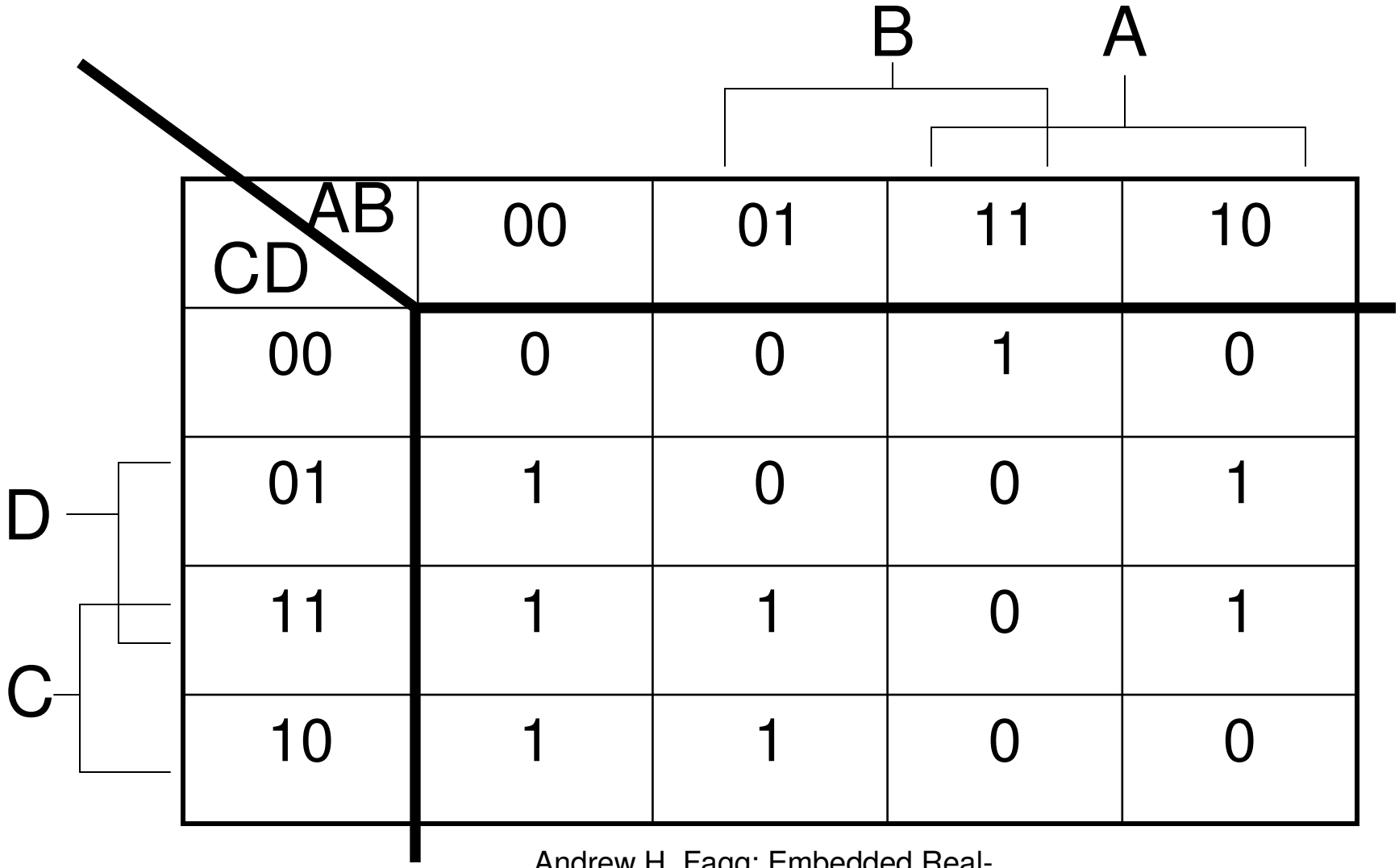
A	B	C	D		E
0	0	0	0		0
0	0	0	1		1
0	0	1	0		1
0	0	1	1		1
0	1	0	0		0
0	1	0	1		0
0	1	1	0		1
0	1	1	1		1

A	B	C	D		E
1	0	0	0		0
1	0	0	1		1
1	0	1	0		0
1	0	1	1		1
1	1	0	0		1
1	1	0	1		0
1	1	1	0		0
1	1	1	1		0

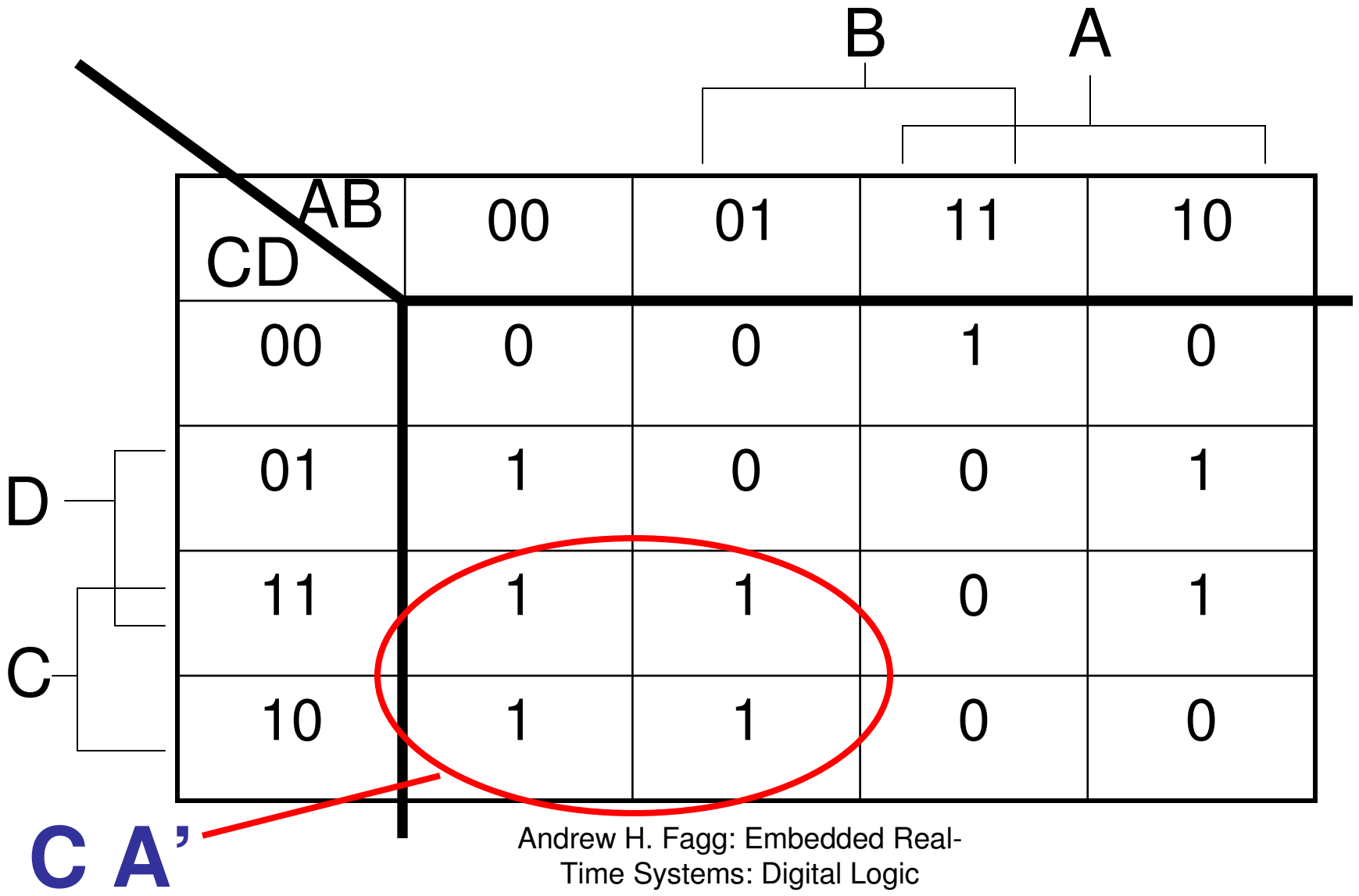
Step 1: Construct Map



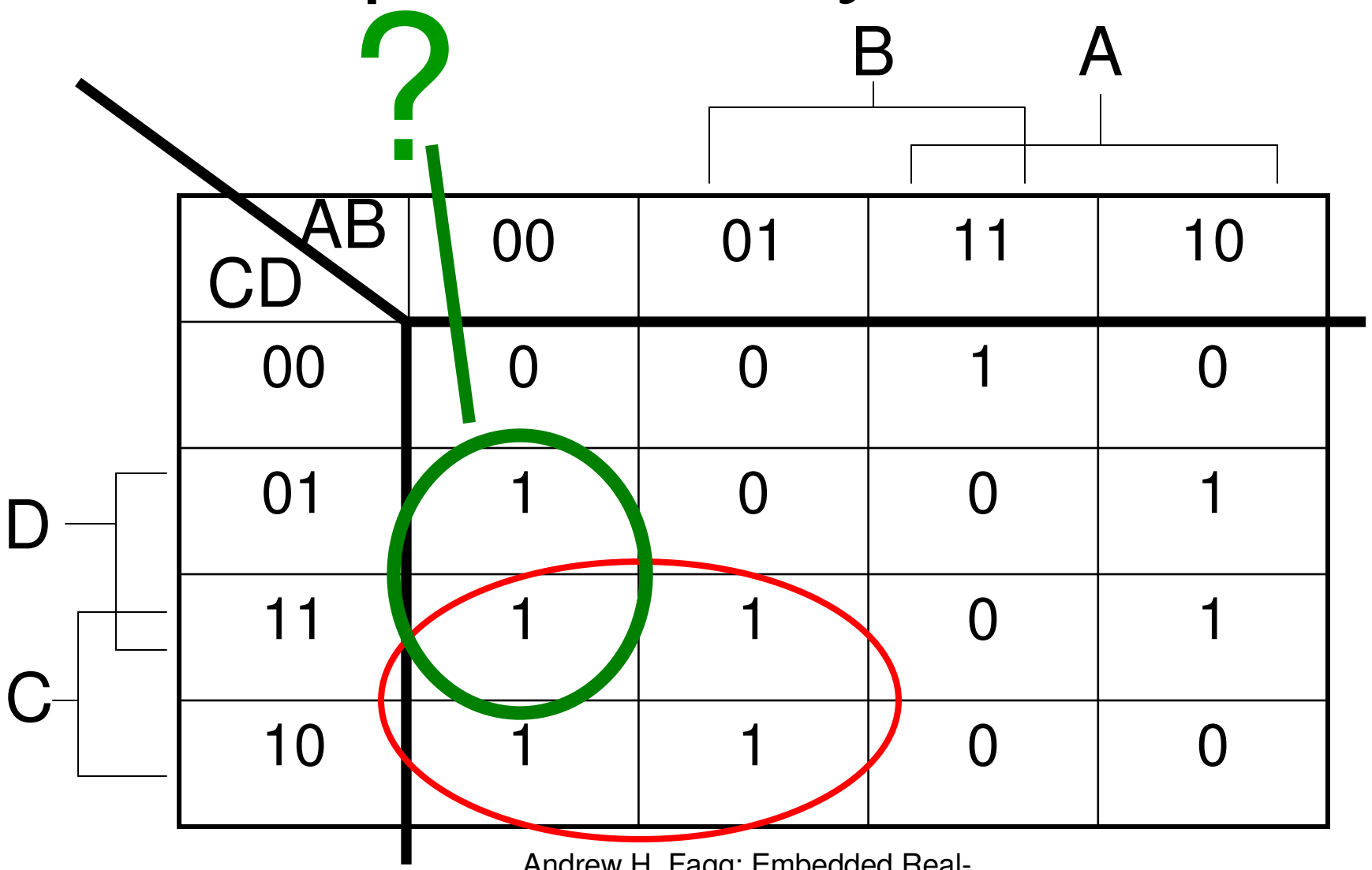
Step 2: Insert Output Values



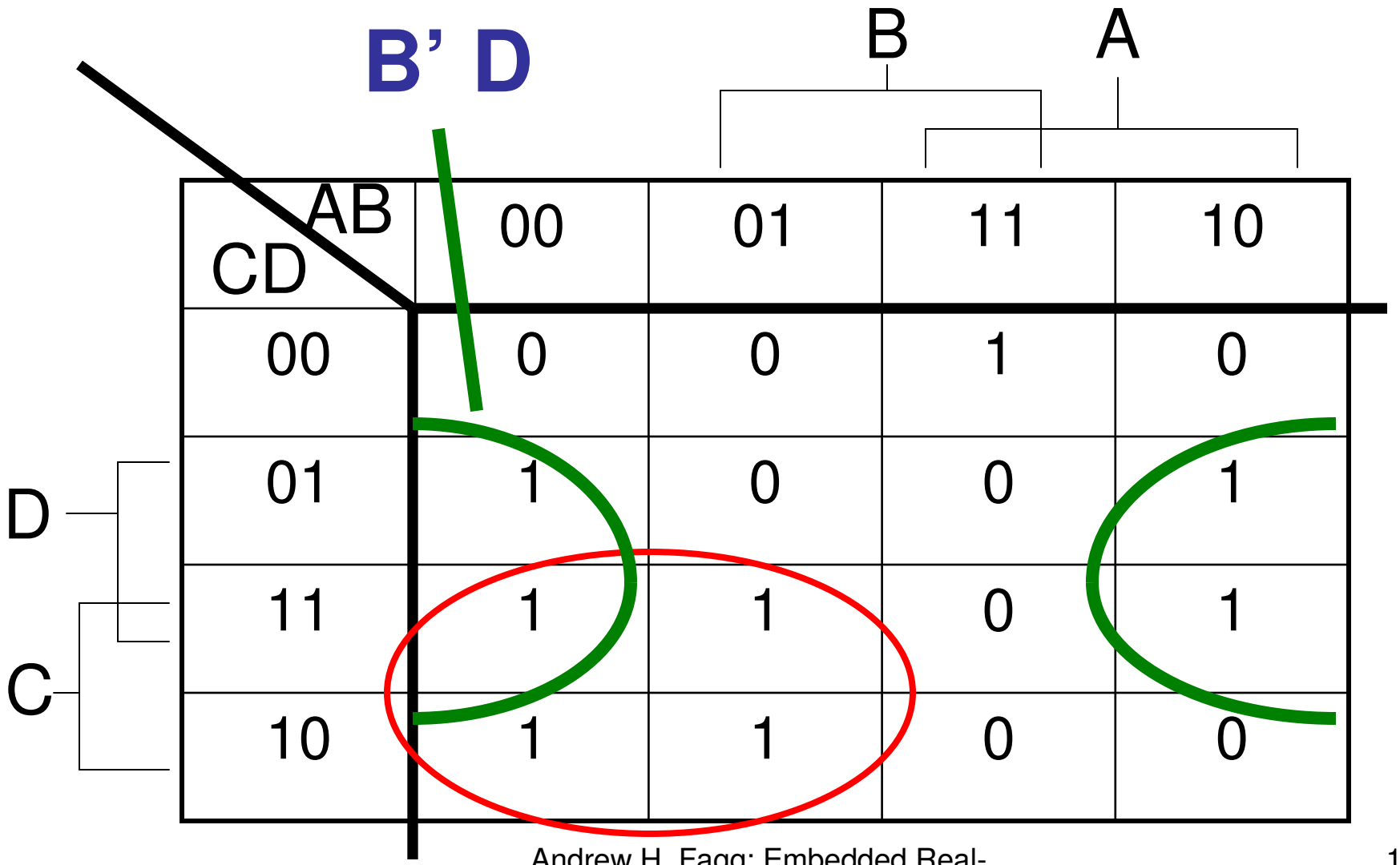
Step 3: Identify Clusters



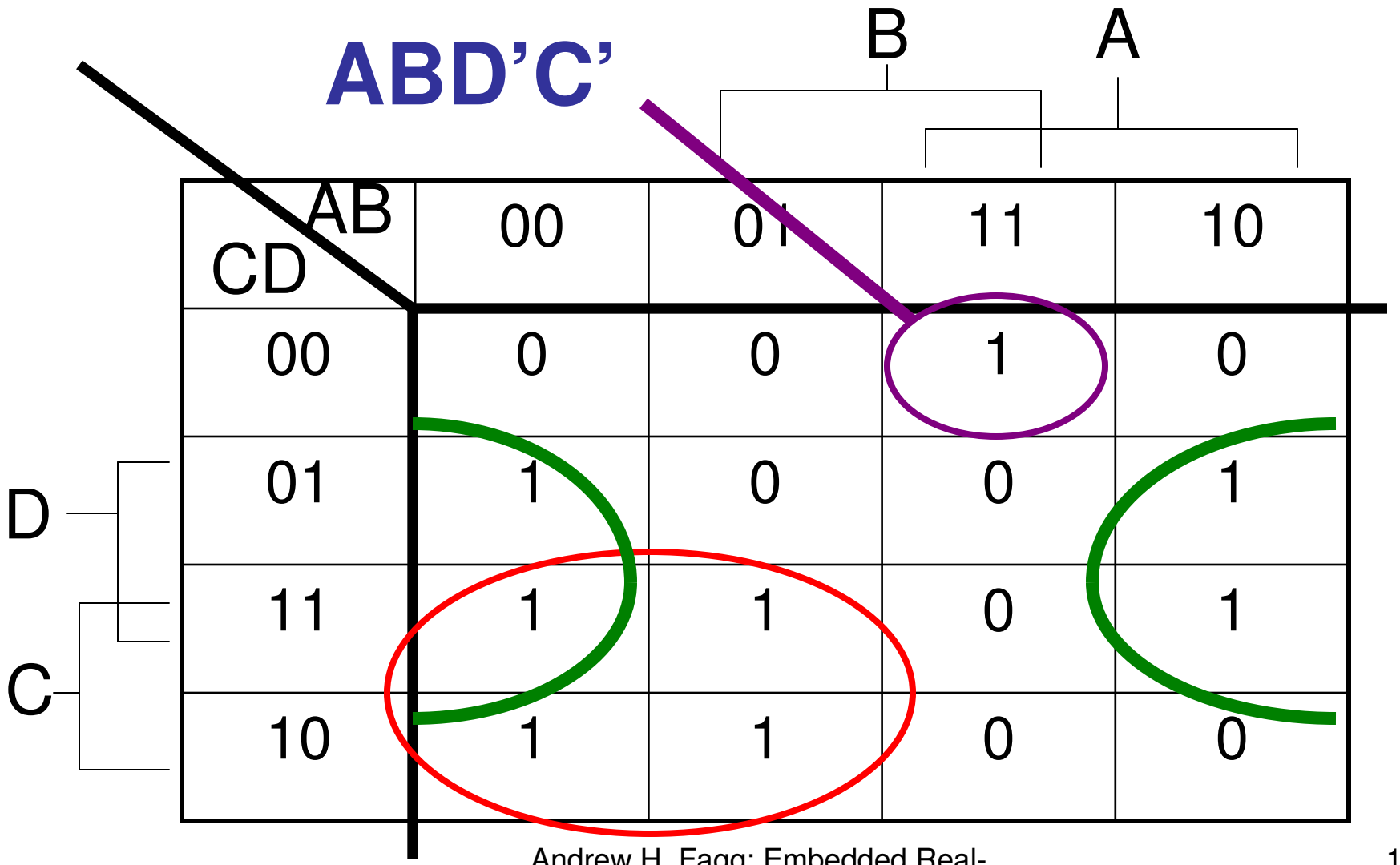
Step 3: Identify Clusters



Step 3: Identify Clusters



Step 3: Identify Clusters

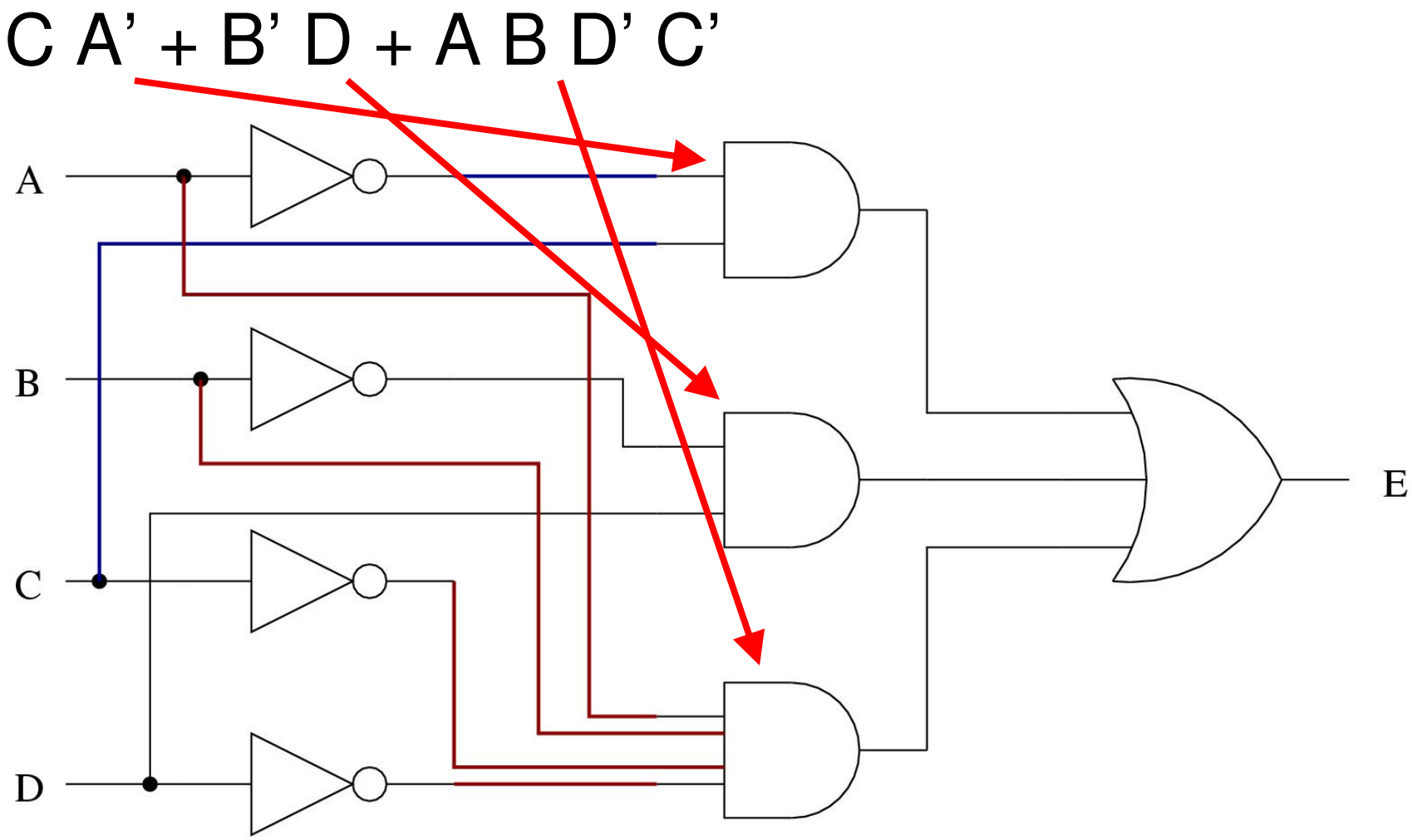


Step 4: Read Out the Terms

C A’ + B’ D + A B D’ C’

What does the circuit look like?

Resulting Circuit



“Don’t Care” Cases

- For some functions that we will implement, some of the input cases will never occur

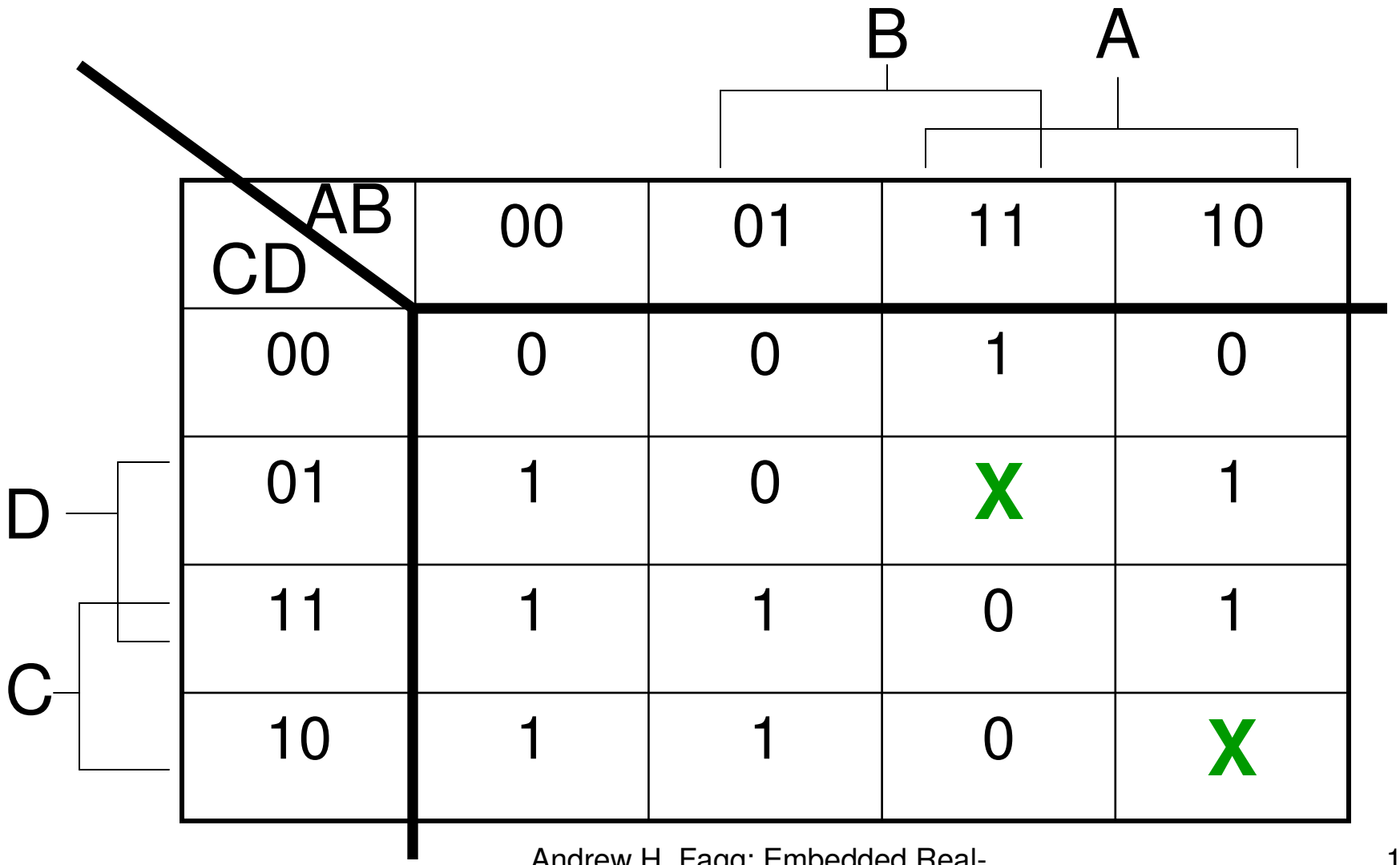
- How does this affect our Karnaugh Map?

Modified Truth Table

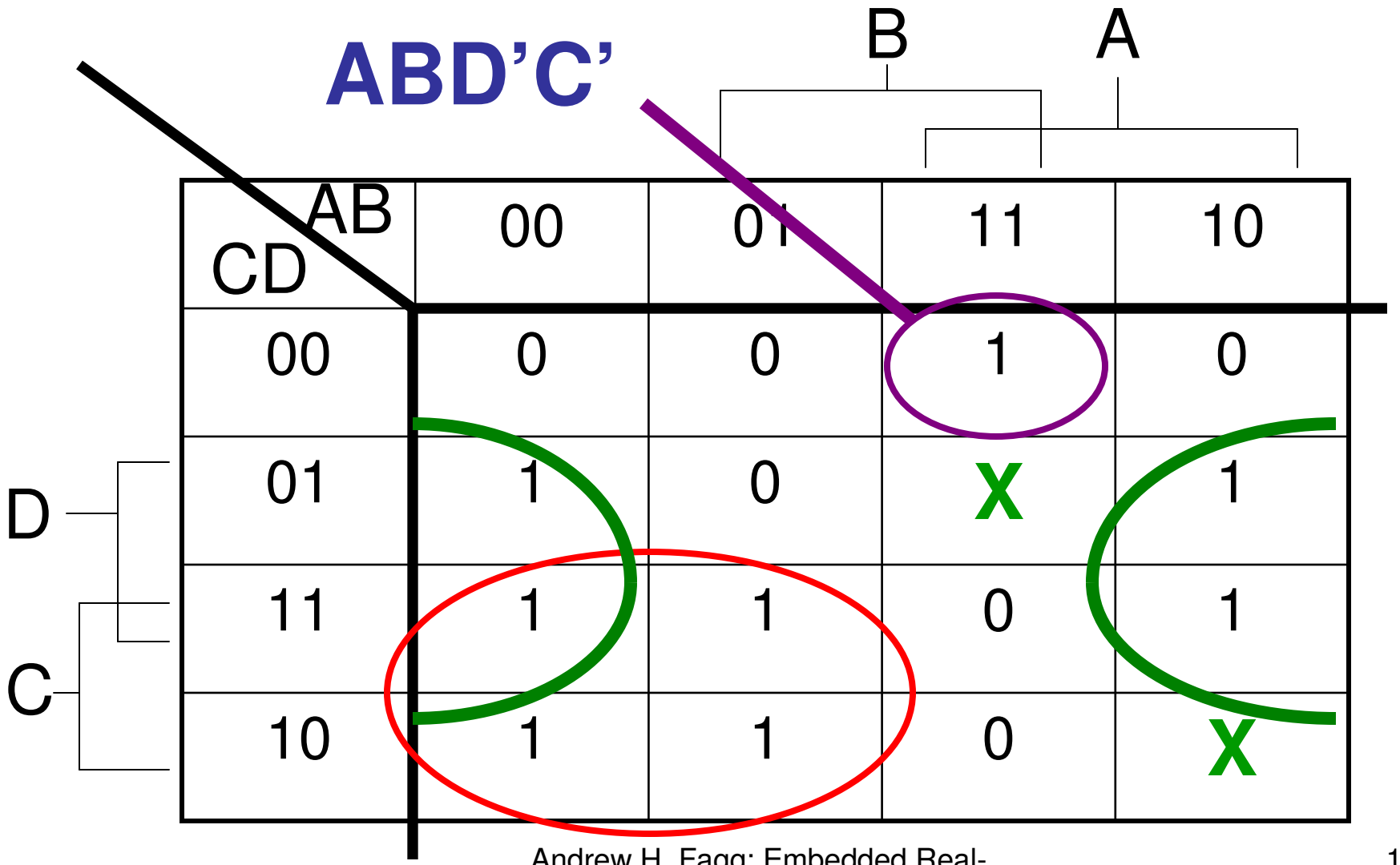
A	B	C	D		E
0	0	0	0		0
0	0	0	1		1
0	0	1	0		1
0	0	1	1		1
0	1	0	0		0
0	1	0	1		0
0	1	1	0		1
0	1	1	1		1

A	B	C	D		E
1	0	0	0		0
1	0	0	1		1
1	0	1	0		**X**
1	0	1	1		1
1	1	0	0		1
1	1	0	1		**X**
1	1	1	0		0
1	1	1	1		0

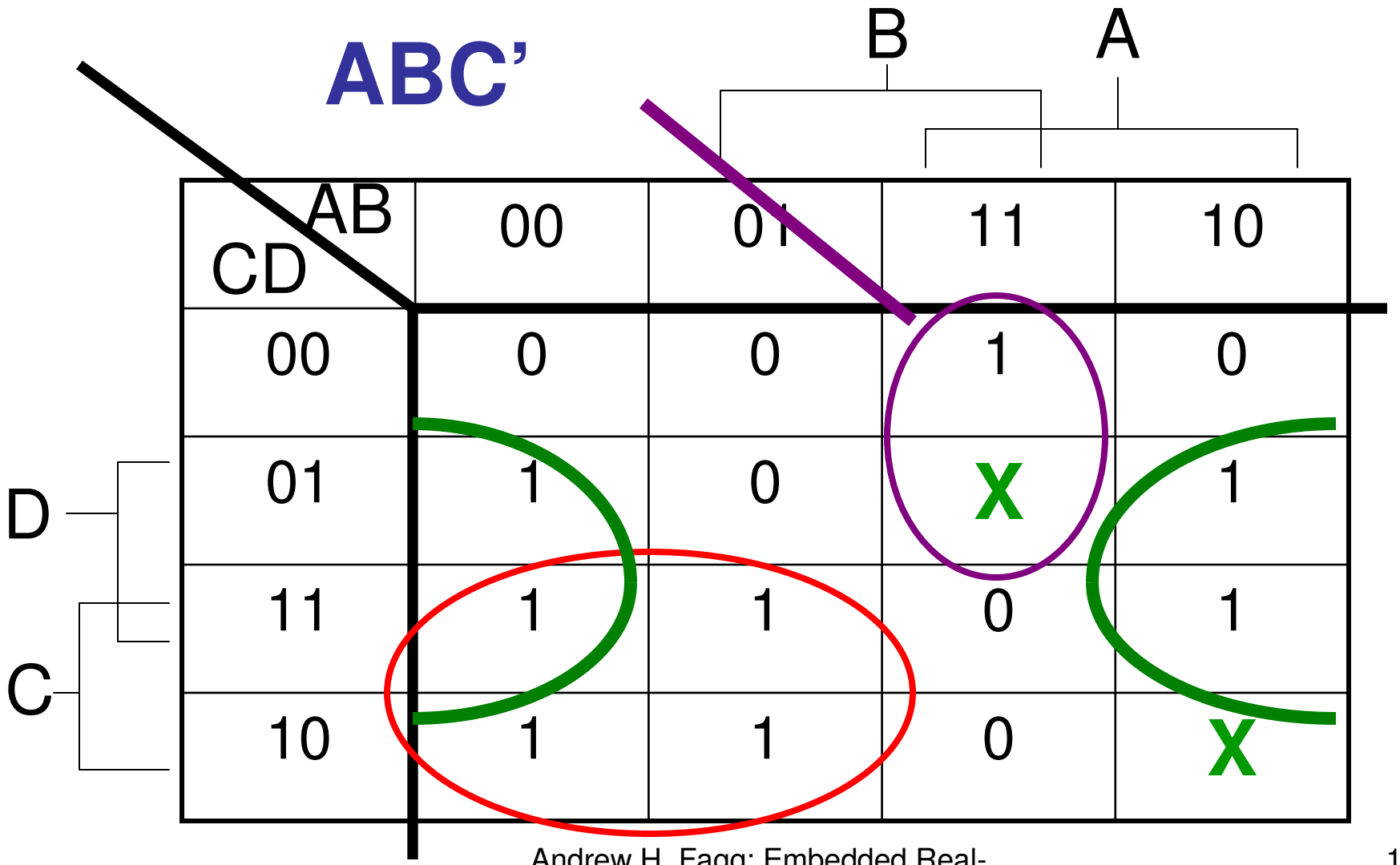
Step 2: Insert Output Values



Step 3: Identify Clusters



Step 3: Identify Clusters



Step 4: Read Out the Terms

C A’ + B’ D + A B C’

The resulting circuit is simpler (but just a little in this case)

Multiple Output Variables

Suppose we have a function with multiple output variables?

- How do we handle this with Karnaugh Maps?

Multiple Output Variables

How do we handle this with Karnaugh Maps?

- We produce one Karnaugh Map for each output variable
- But: in the final implementation, some sub-circuits may be shared

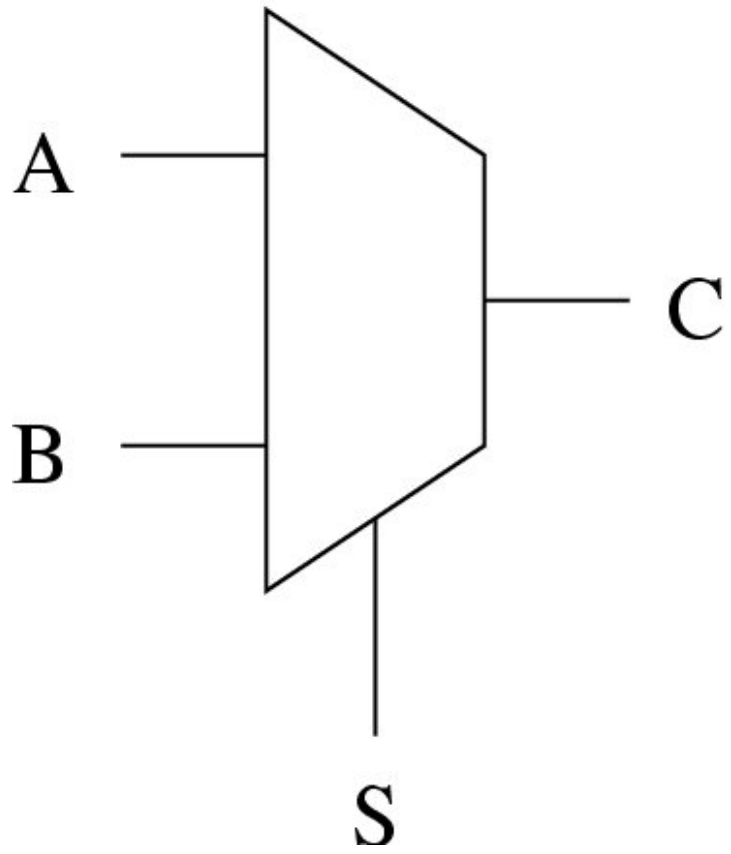
Karnaugh Maps: Final Notes

- Always use clusters that are as large as possible
- Possible to extend technique to 5 and 6 inputs (and more), but it starts to get hairy
  - Tend to make use of circuit design "assistants"

Multiplexer

A multiplexer is a device that selects between two input lines

- A & B are the inputs
- S is the selection signal (also an input)
- C is a copy of A if S=0
- C is a copy of B if S=1

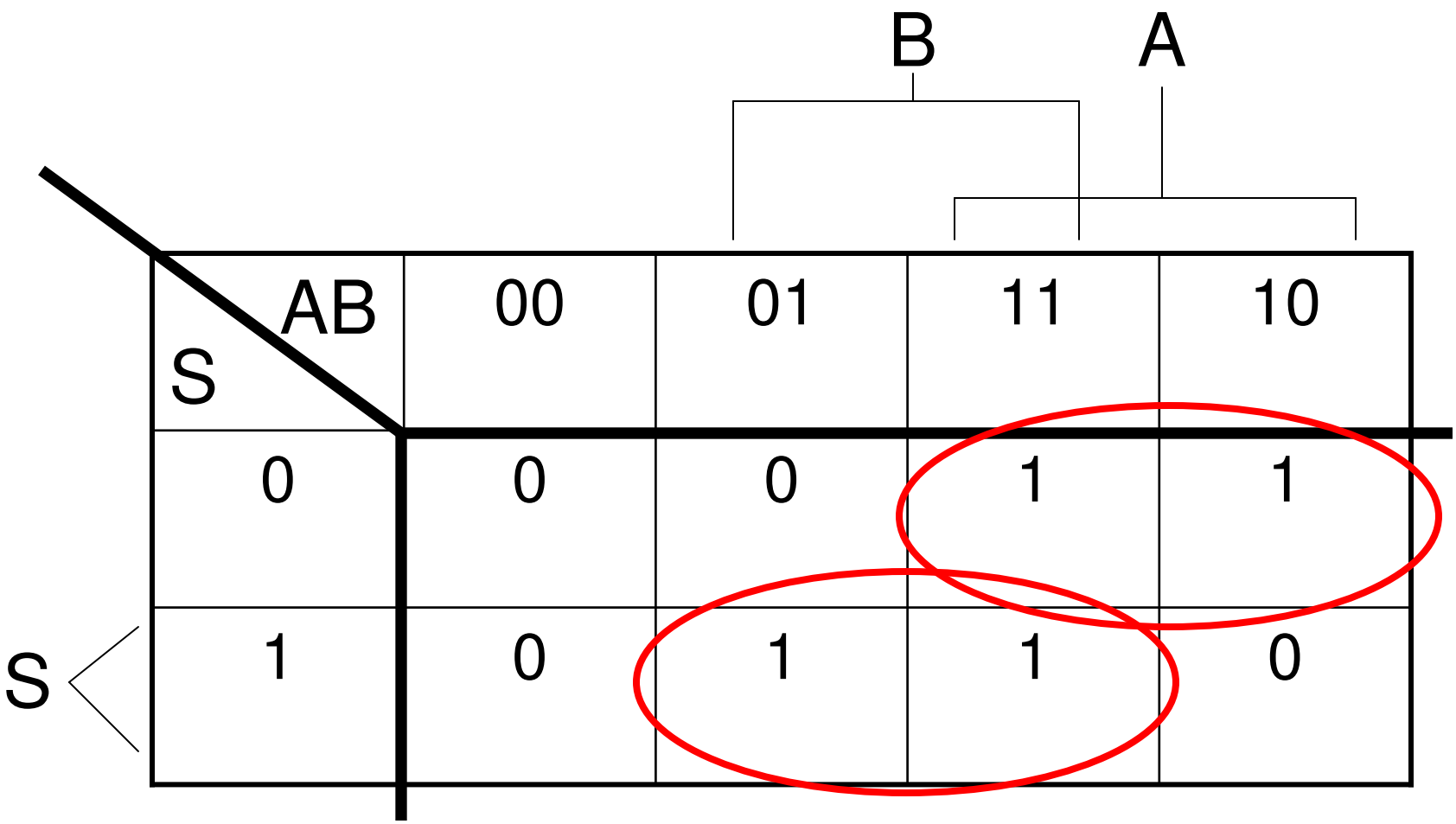


Multiplexer Truth Table

What would the Karnaugh map look like?

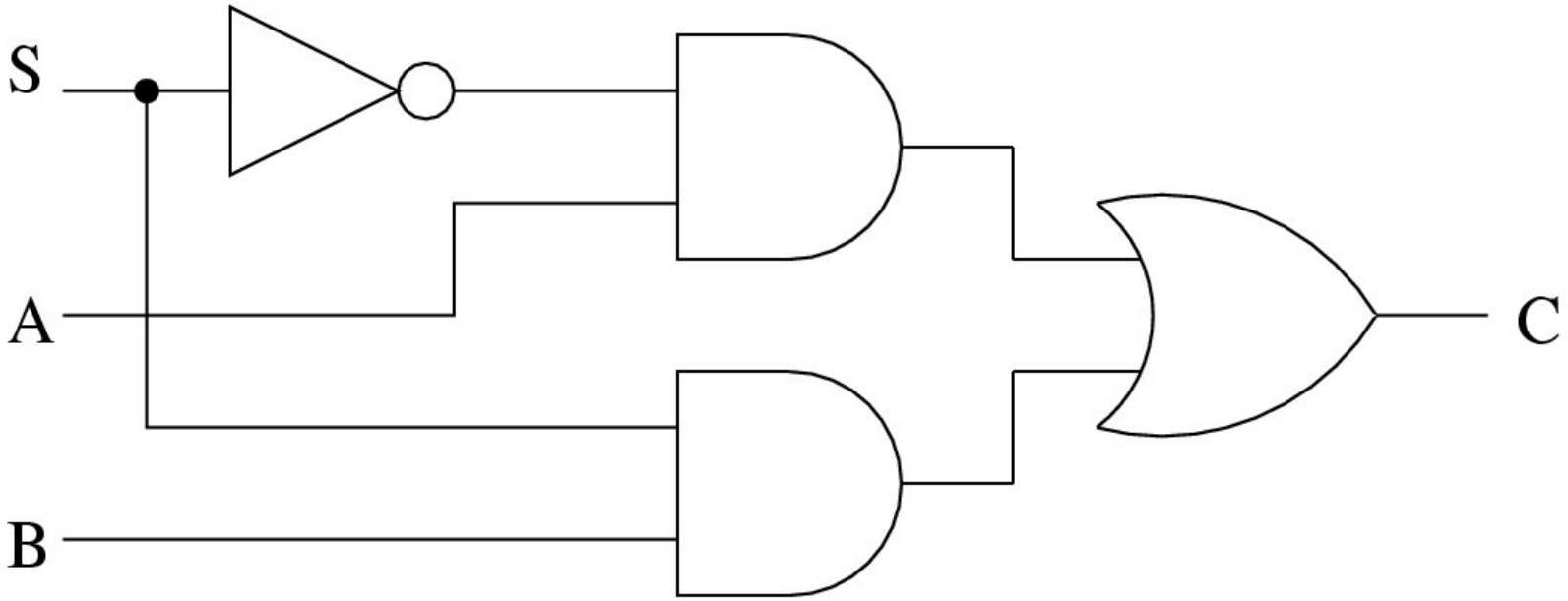
S	A	B	C
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Multiplexer K-Map



Multiplexer Implementation

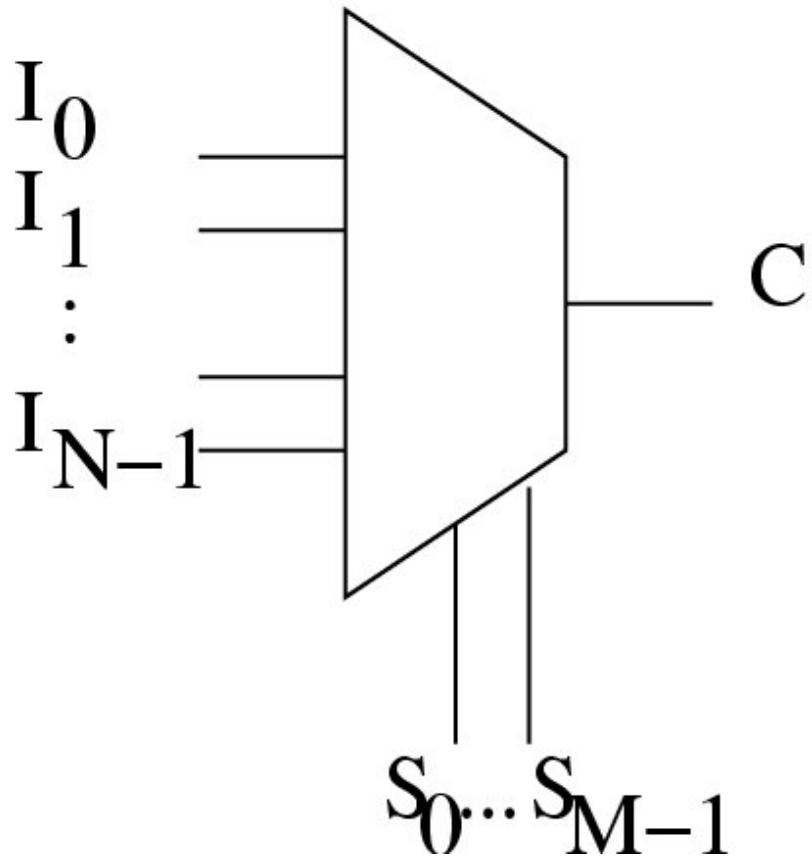
C = S'A + SB



N-Input Multiplexer

Suppose we want to select from between N different inputs.

- This requires more than one select line. How many?



N-Input Multiplexer

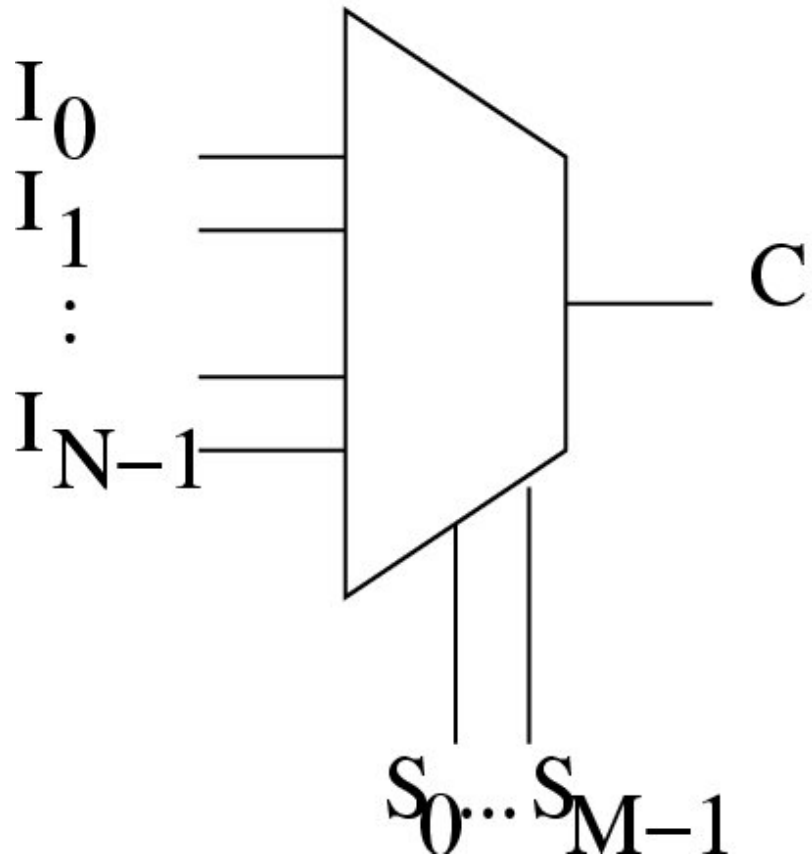
How many select lines?

- $M = \log_2 N$

  or

- $N = 2^M$

What would the N=8 implementation look like?



8-Input Multiplexer Implementation

$$C = I_0S'_2S'_1S'_0 + I_1S'_2S'_1S_0 + I_2S'_2S_1S'_0 + I_3S'_2S_1S_0 + I_4S_2S'_1S'_0 + I_5S_2S'_1S_0 + I_6S_2S_1S'_0 + I_7S_2S_1S_0$$

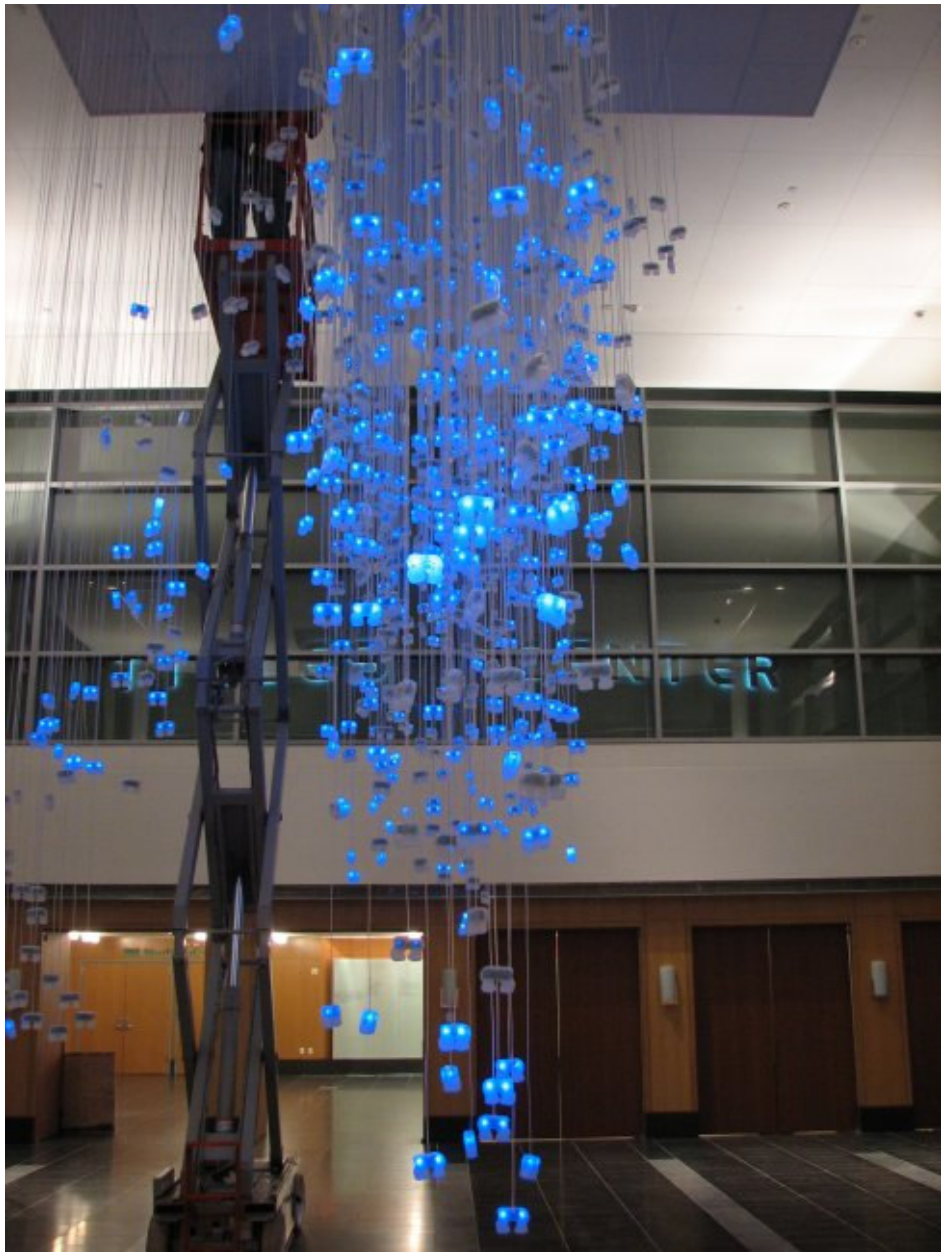
Note that we have one of each possible select line combination (or addressing terms)

Next Time

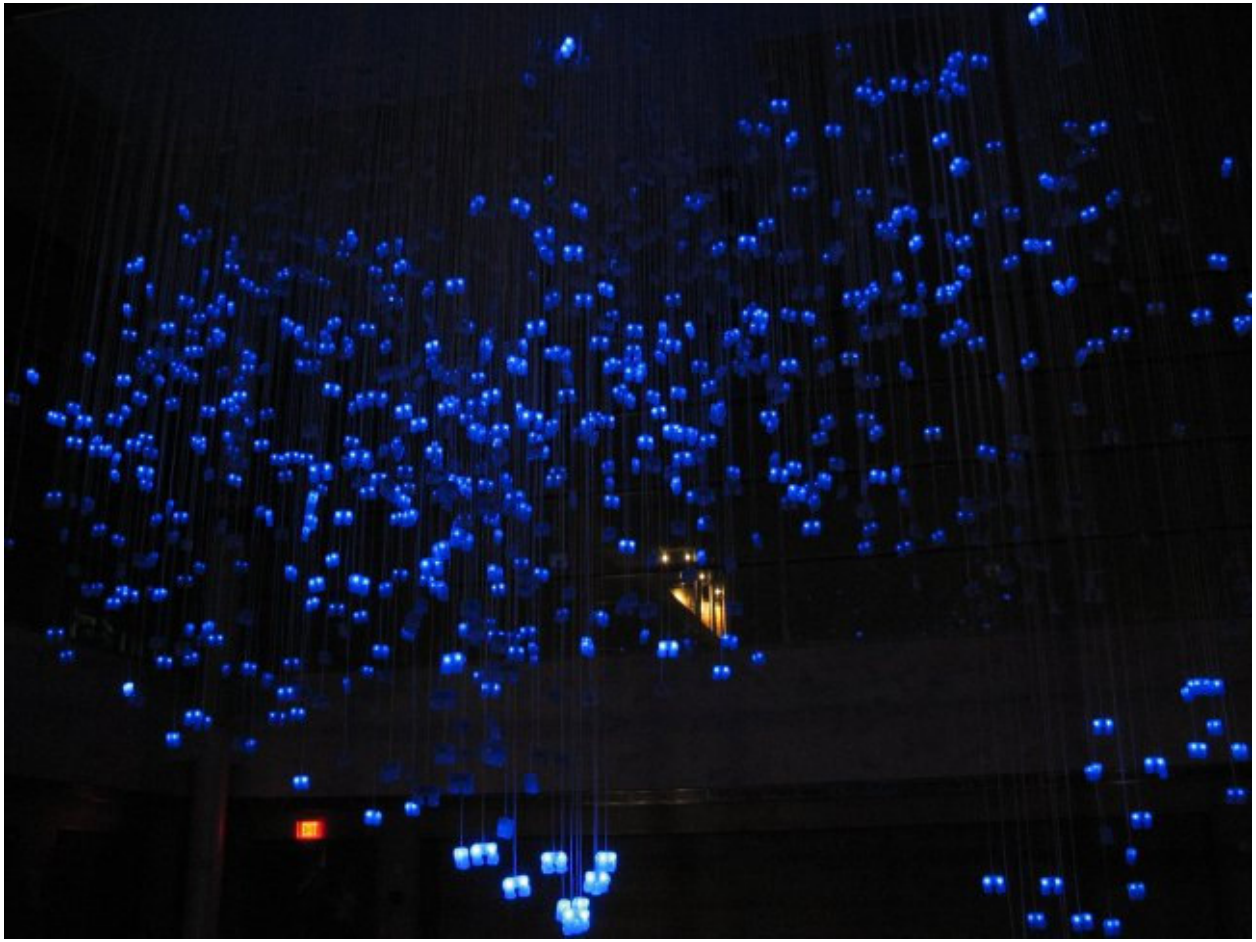
- Digital logic in practice
- Reading (listed on the schedule):
  - ESA 3.3, 3.5.1, 3.5.2, 3.6.1, 3.7, Appendix B

Bion Installation at Long Island University

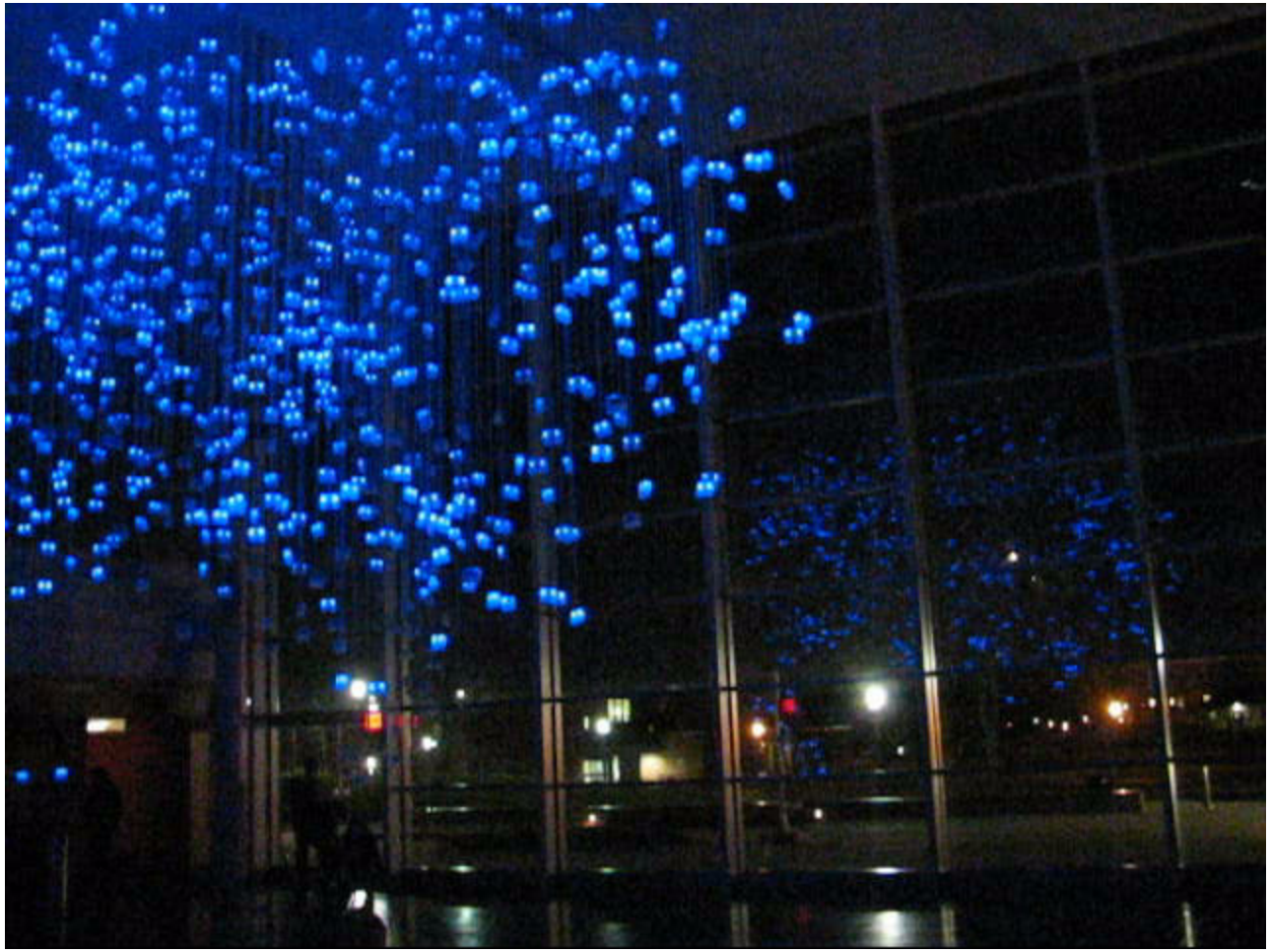
Bion



Bion



Bion



Last Time

- Karnaugh Maps and Circuit Reduction
- Multiplexers

Today

- Demultiplexers
- Tristate buffers
- Practical issues in digital logic implementation

Administrivia

- Homework 1 is due today at 5:00
- Homework 2 available later today (and is due in 1 week)

Administrivia II

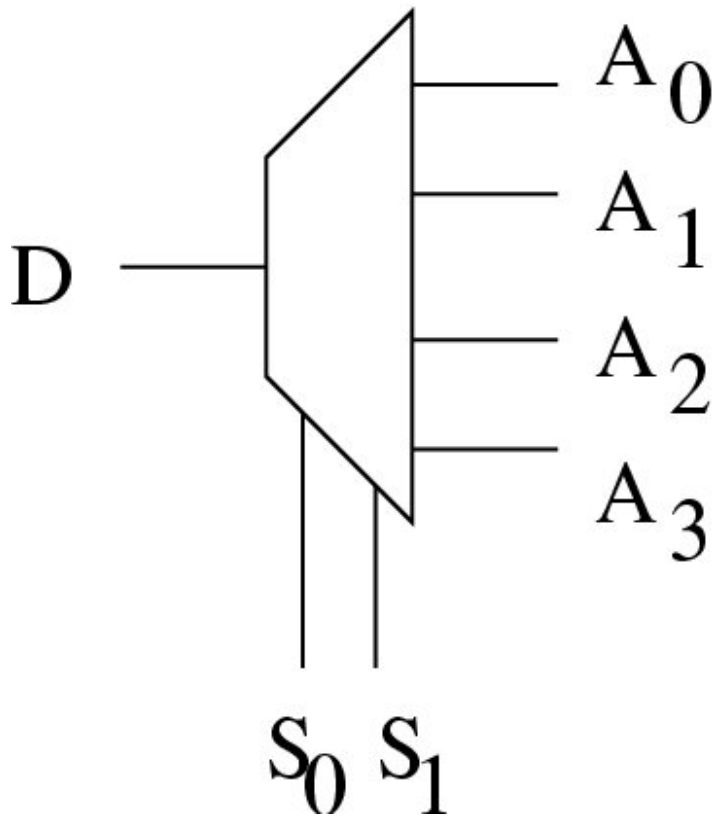
Make sure you fill out and hand-in group placement forms today

“Experimental Group:”

- First 2 labs would be with the mobile robots; last 2 would be with a custom robot
- Between now and the 3rd lab: we would be responsible for designing and constructing the robot

Demultiplexer

- The multiplexer reduces N signals down to 1 (with M select lines)
- A demultiplexer routes a data input (D) to one of N output lines (As)
  - Which A depends on the select lines



2-Input Demultiplexer Truth Table

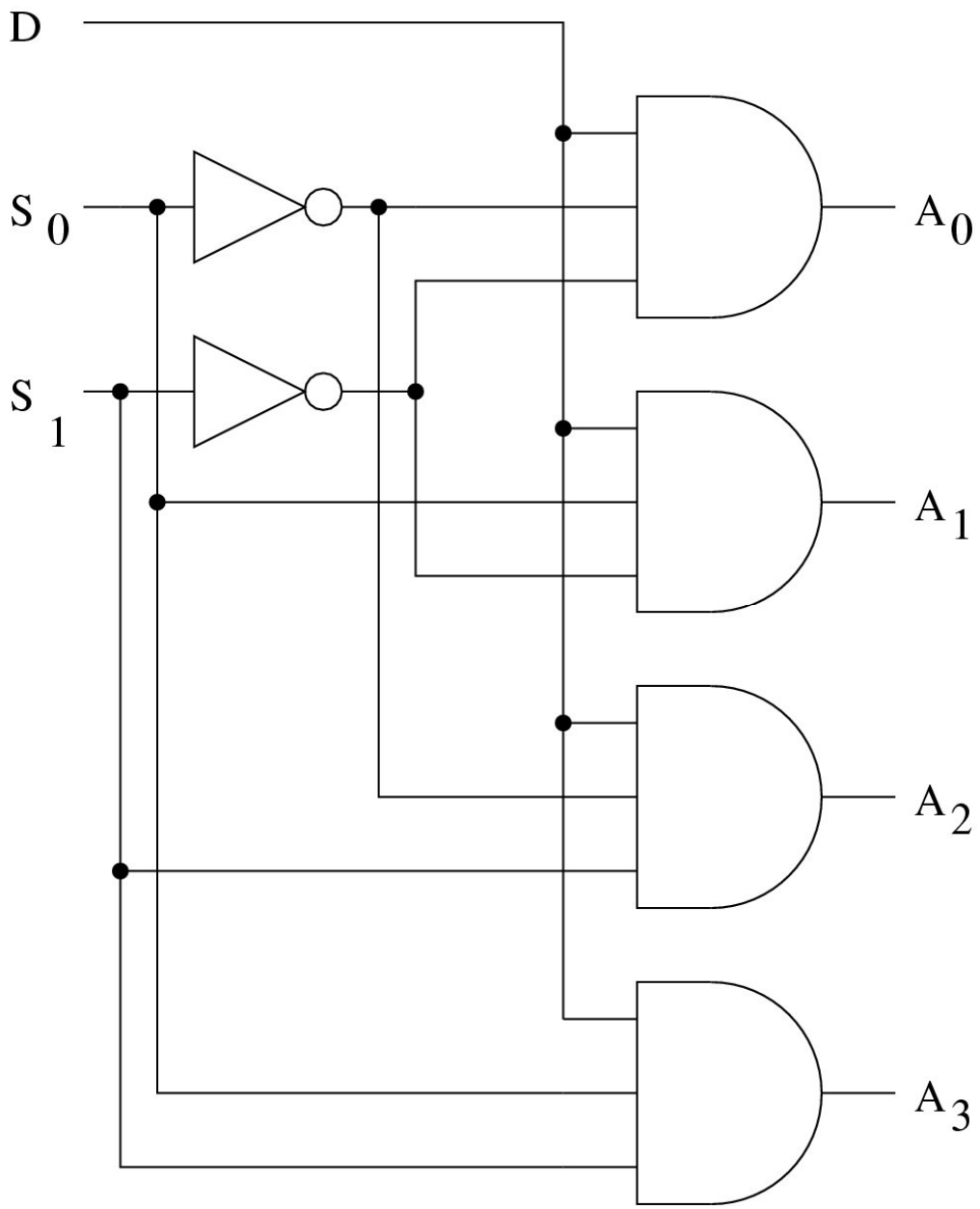
Inputs Outputs

S_1	S_0	D		A_3	A_2	A_1	A_0
0	0	0		0	0	0	0
0	0	1		0	0	0	1
0	1	0		0	0	0	0
0	1	1		0	0	1	0
1	0	0		0	0	0	0
1	0	1		0	1	0	0
1	1	0		0	0	0	0
1	1	1		1	0	0	0

Demultiplexer Implementation

- What does it look like?

Demultiplexer Implementation



M-Input (Select) Demultiplexer

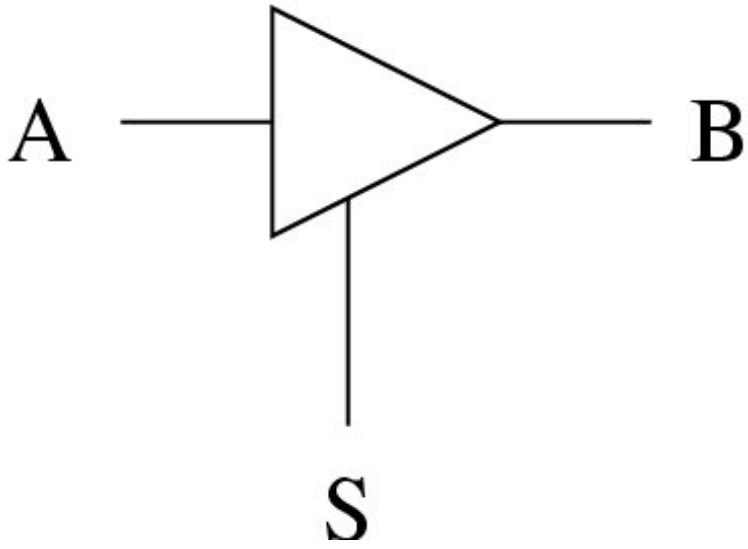
- Will have 2^M output lines
- Useful for converting a binary address into select lines for devices and memory elements
  - (more on binary soon)

Tristate Buffers

- Until now: the output line(s) of each device are driven either high or low
  - So the line is either a source or a sink of current
- Tristate buffers can do this or leave the line floating (as if it were not connected to anything)

Tristate Buffers

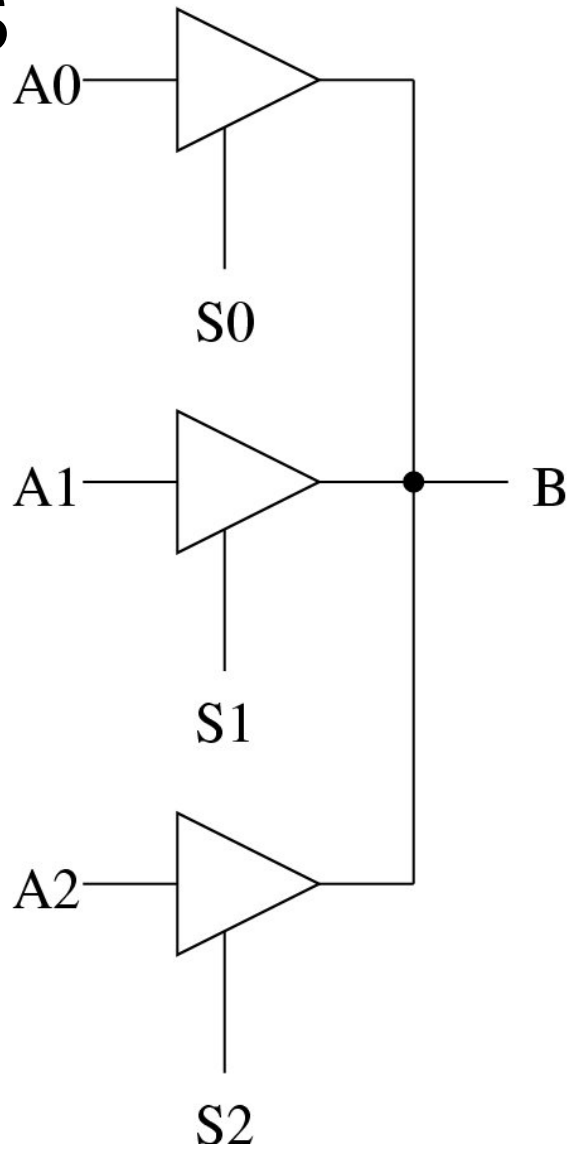
S	A		B
0	0		floating
0	1		floating
1	0		0
1	1		1



How are tristate buffers useful?

Tristate Buffers

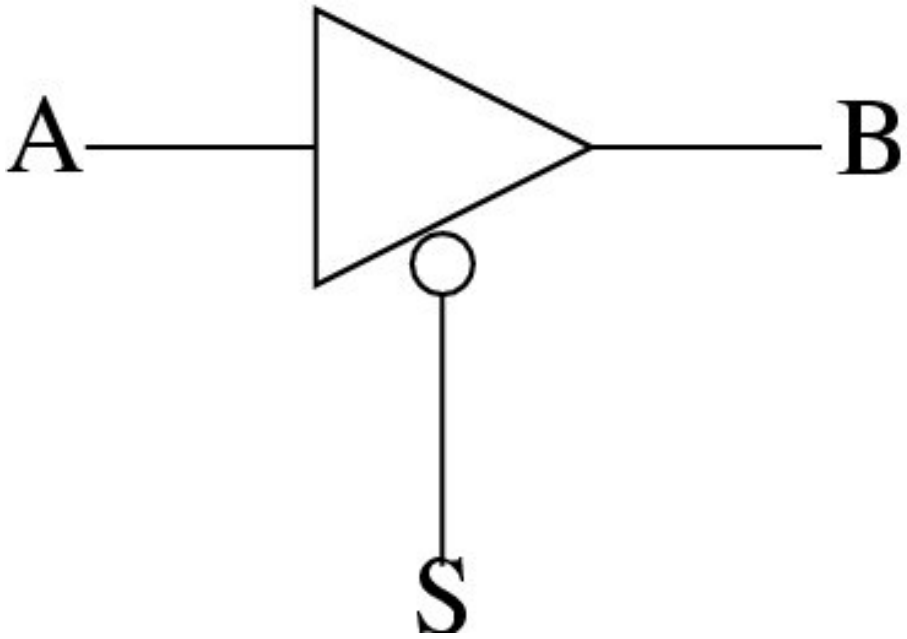
We can wire the outputs of multiple tristate buffers together without any other logic



Tristate Buffers

- We must guarantee that only one select line is active at any one time
- Tristate buffers will turn out to be useful when we start building data and address buses

Another Tristate Buffer



What does the truth table look like?

Another Tristate Buffer

S	A		B
0	0		0
0	1		1
1	0		floating
1	1		floating