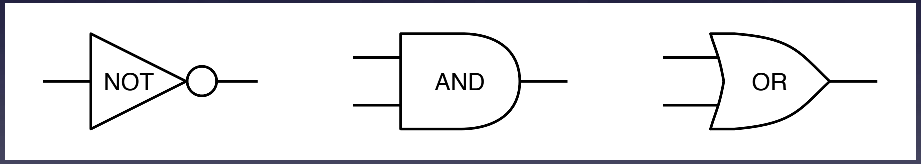
Boolean Logic

Steven R. Bagley

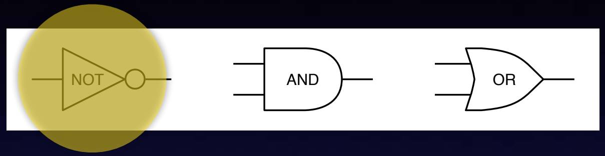
Introduction

- Last week, looked inside a computer…
- Saw how information is represented in binary as digital signals
- Understand how to manipulate those signals
- To do useful things…

Logic Gates

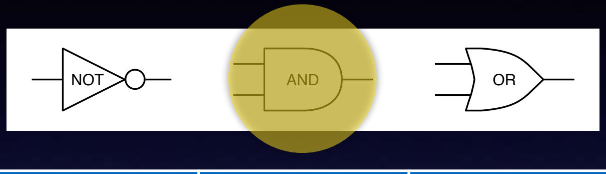


Logic Gates



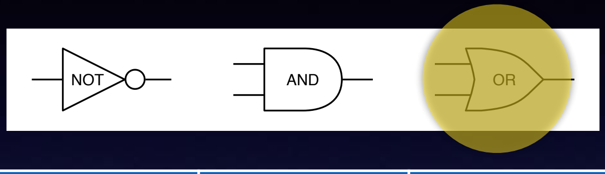
A	Result
0	1
1	0

Logic Gates

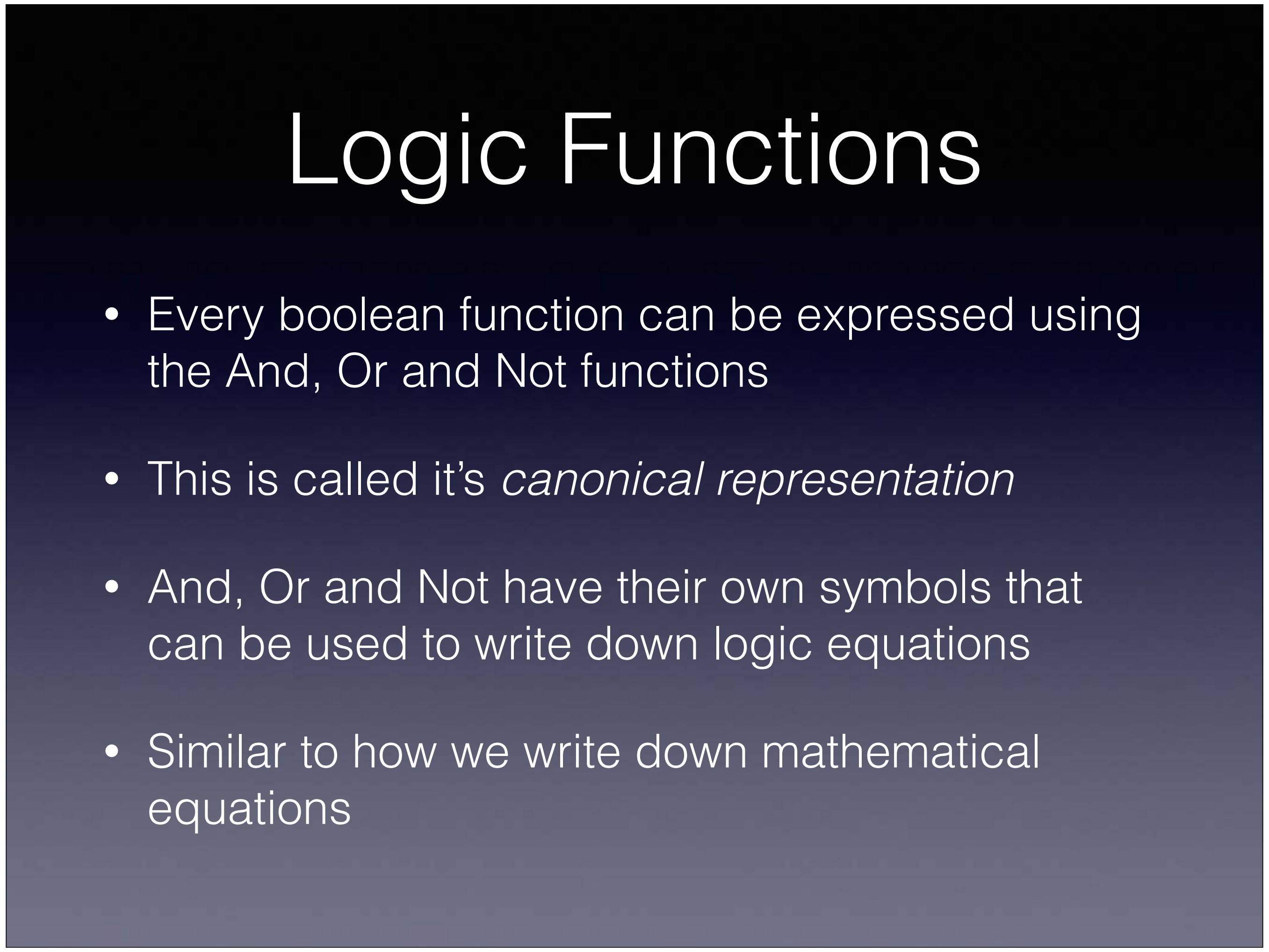


A	B	Result
0	0	0
0	1	0
1	0	0
1	1	1

Logic Gates



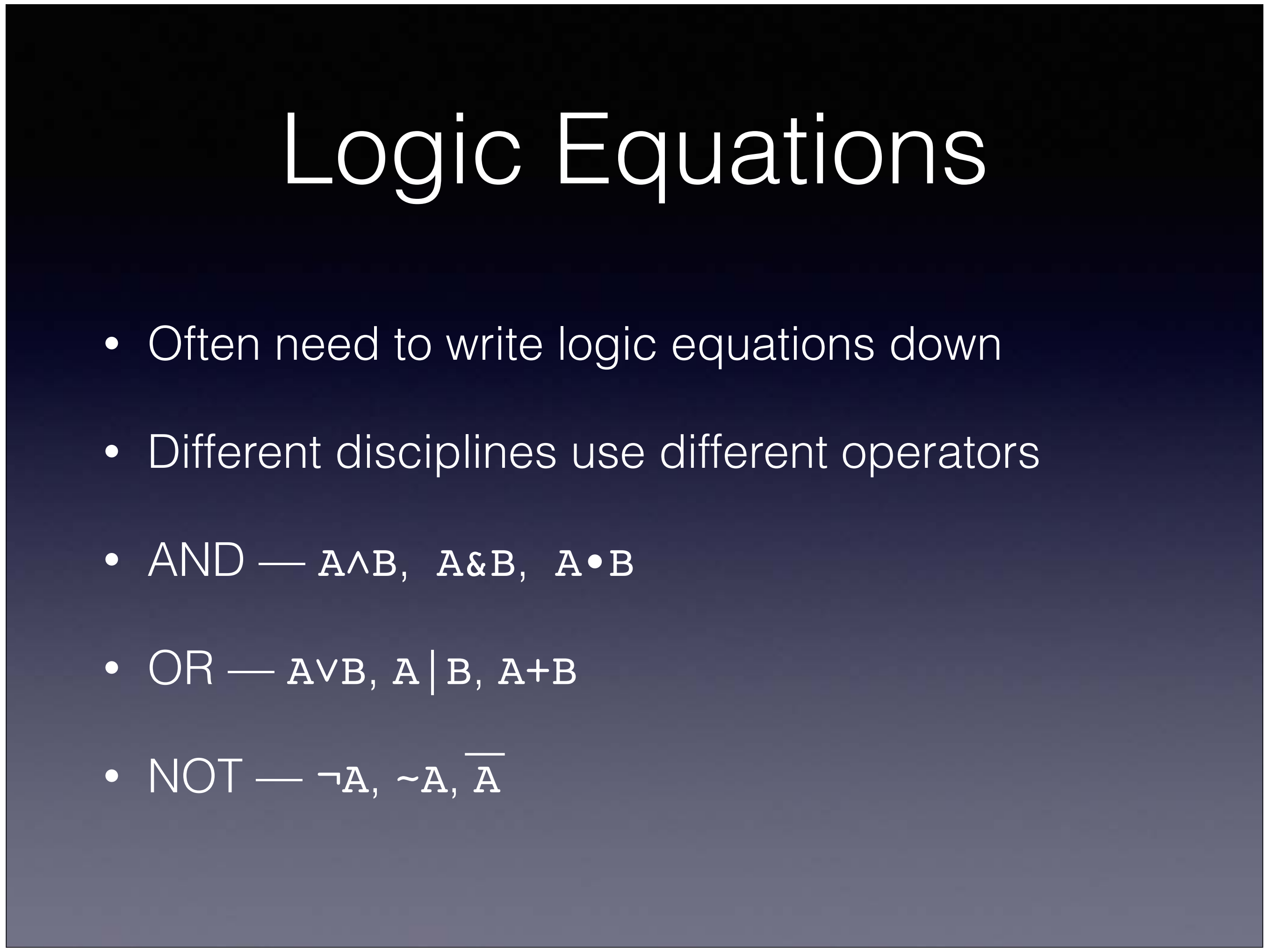
A	B	Result
0	0	0
0	1	1
1	0	1
1	1	1



Logic Functions

- Every boolean function can be expressed using the And, Or and Not functions
- This is called it’s *canonical representation*
- And, Or and Not have their own symbols that can be used to write down logic equations
- Similar to how we write down mathematical equations

Unfortunately there are lots of symbols used…



Logic Equations

- Often need to write logic equations down
- Different disciplines use different operators
- AND — `A∧B`, `A&B`, `A•B`
- OR — `A∨B`, `A|B`, `A+B`
- NOT — `¬A`, `~A`, $\overline{\mathtt{A}}$

Symbols in order, Maths, Programning languages (specifically, C), electronics?
Bar can be extend over other things to show what's inverted
Not C also uses ! in some cases to mean inversion

Precedence

- Operators have precedence
- NOT binds most tightly
- Then AND
- Finally OR
- So $A+B \bullet C$ is equivalent to $A+(B \bullet C)$

Two input functions

Function	A	0	0	1	1
	B	0	1	0	1
Constant Zero	0	0	0	0	0
And	$A \bullet B$	0	0	0	1
A And Not B	$A \bullet \overline{B}$	0	0	1	0
A	A	0	0	1	1
Not A And B	$\overline{A} \bullet B$	0	1	0	0
B	B	0	1	0	1
Xor	$A \bullet \overline{B} + \overline{A} \bullet B$	0	1	1	0
Or	$A+B$	0	1	1	1
Nor	$\overline{A+B}$	1	0	0	0
Equivalence	$A \bullet B + \overline{A} \bullet \overline{B}$	1	0	0	1
Not B	$\overline{B}$	1	0	1	0
If B then A	$A+\overline{B}$	1	0	1	1
Not A	$\overline{A}$	1	1	0	0
If X then Y	$\overline{A}+B$	1	1	0	1
Nand	$\overline{A \bullet B}$	1	1	1	0
Constant One	1	1	1	1	1

Logic Gates

- An electronic circuit representing a logic function is called a *gate*
- So with electronic circuits that represents And, Or and Not
- We can combine them to build any logic function as a circuit
- To process the signals in the computer

Show YouTUBE clip

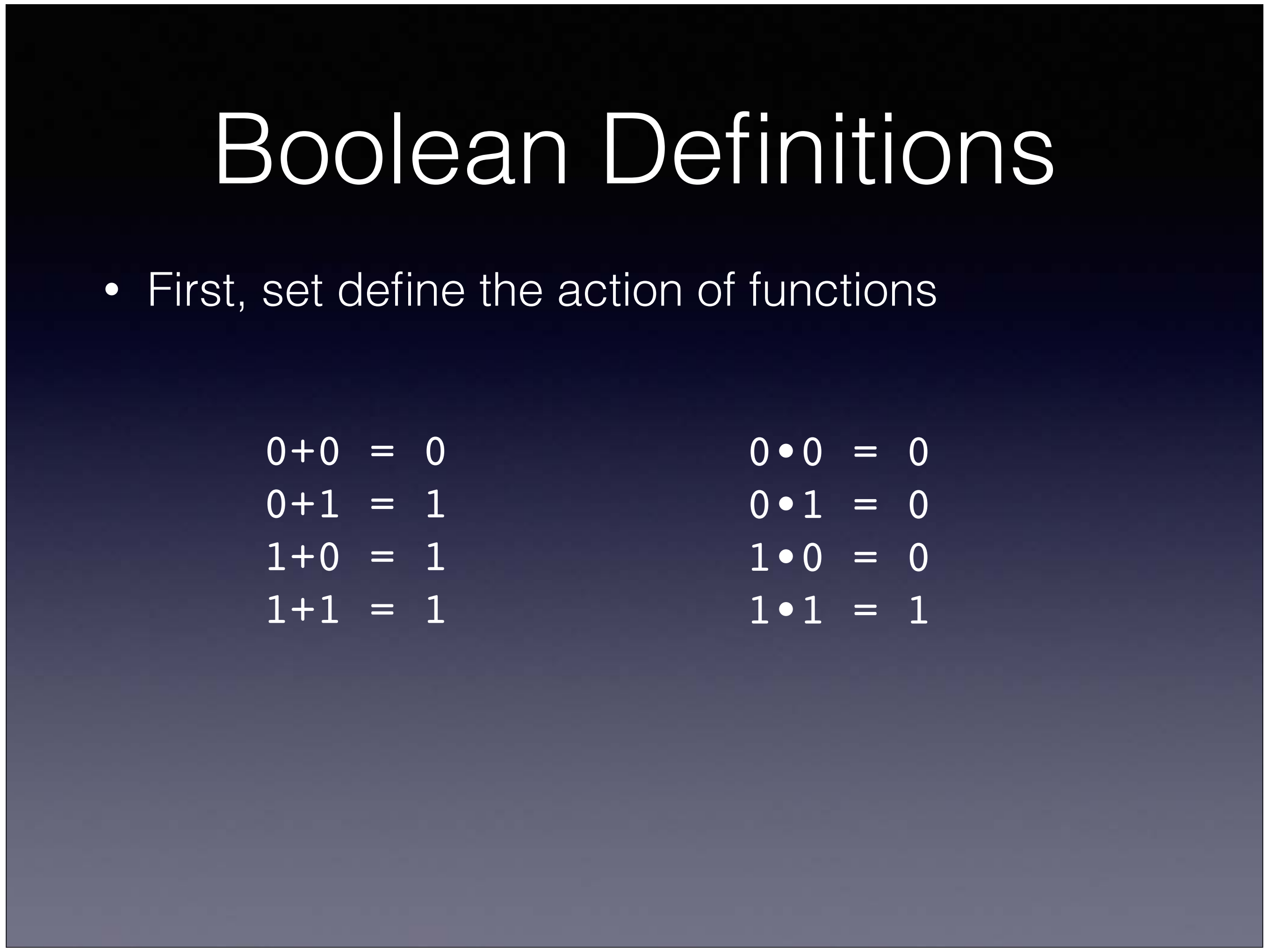
Boolean Algebra

- There are a set of algebraic rules for logic functions
- Set out by George Boole in ‘A Calculus of Logic’
- Allows us to reason about logic functions and simplify them…
- Can also use them to show that we can build every logic function if we are given NAND

Boole, 1815-1864

Boolean Algebra

- Already seen how we can build all two-input functions from And, Or and Not
- So if we can build those three functions using NAND then we can build all the functions

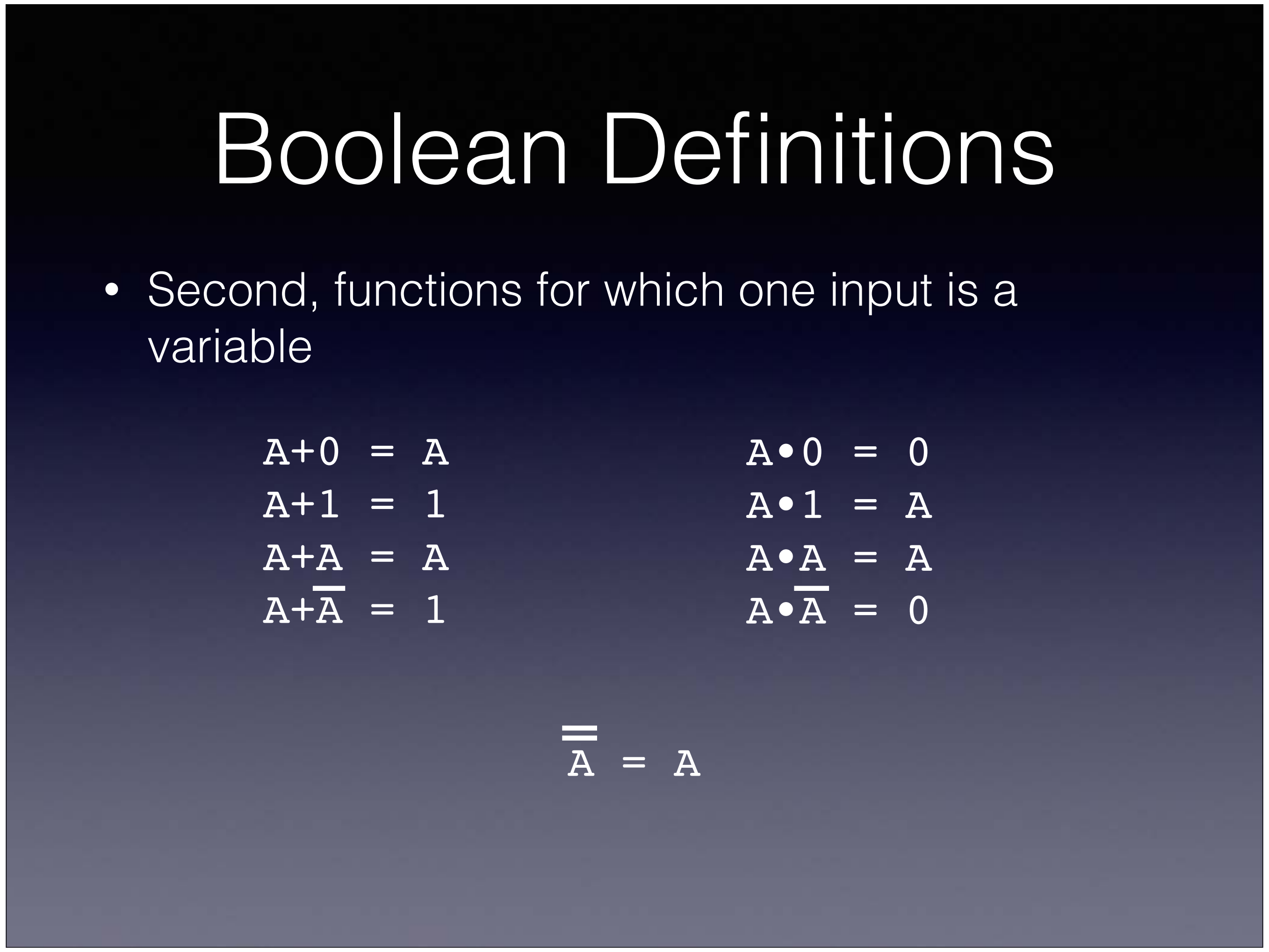


Boolean Definitions

- First, set define the action of functions

$0+0 = 0$	$0 \bullet 0 = 0$
$0+1 = 1$	$0 \bullet 1 = 0$
$1+0 = 1$	$1 \bullet 0 = 0$
$1+1 = 1$	$1 \bullet 1 = 1$

Equivalent to the truth table



Boolean Definitions

- Second, functions for which one input is a variable

$A+0 = A$	$A \bullet 0 = 0$
$A+1 = 1$	$A \bullet 1 = A$
$A+A = A$	$A \bullet A = A$
$A+\overline{A} = 1$	$A \bullet \overline{A} = 0$

$$\overline{\overline{A}} = A$$

Go through them

Last one is NOT of NOT A

Boolean Definitions

- Third, for more than one variable

```
A+B           = B+A      A•B           = B•A
(A+B)+C       = A+(B+C)  (A•B)•C       = A•(B•C)
(A•B)+(A•C)   = A•(B+C)  (A+B)•(A+C)   = A+(B•C)
```

Go through them

Boolean Definitions

- Fourth, De Morgan's Theorem

$$\overline{A \bullet B} = \overline{A} + \overline{B} \qquad \overline{A + B} = \overline{A} \bullet \overline{B}$$

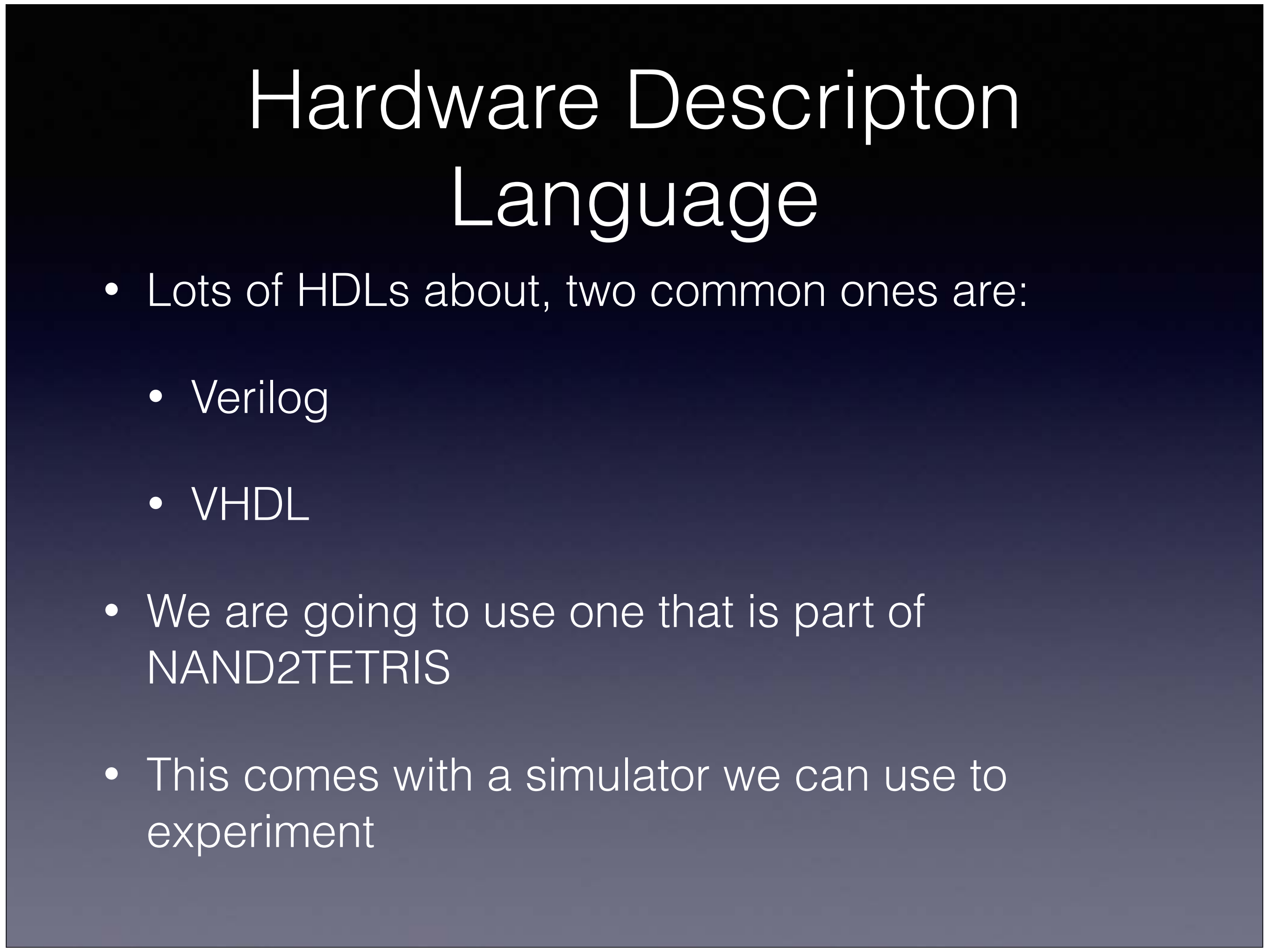
Go through them
These rules are what lets us build everything out of NAND gates
Show truth tables on board for these

Logic Gates

- Given a set of NAND gates, we can design any logic circuit we want
- Can get programmable logic chips that are just arrays of NAND gates
- Easier to design using And, Or and Not gates
- As it's *canonical* representation
- And let the computer convert into NAND gates

Hardware Description Language

- Describe the circuit using a *Hardware Description Language*
- Write the hardware as if it were a computer program using a language
- Express how the various AND, OR and NOT gates connect
- Software then generates necessary NAND gates for a *Programmable Logic Device*



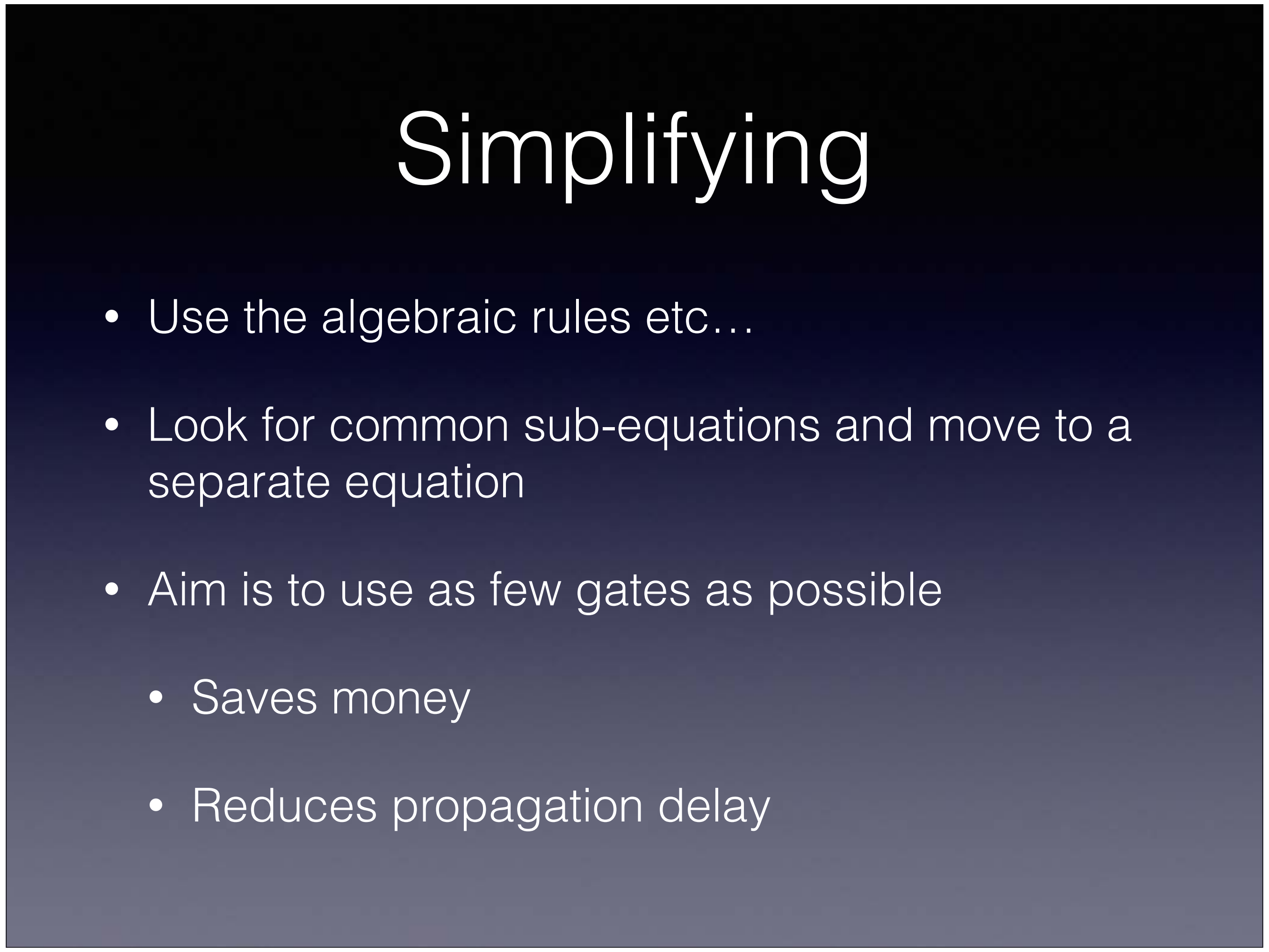
Hardware Descripton Language

- Lots of HDLs about, two common ones are:
    - Verilog
    - VHDL
- We are going to use one that is part of NAND2TETRIS
- This comes with a simulator we can use to experiment

Go demo it…

Designing Logic Circuits

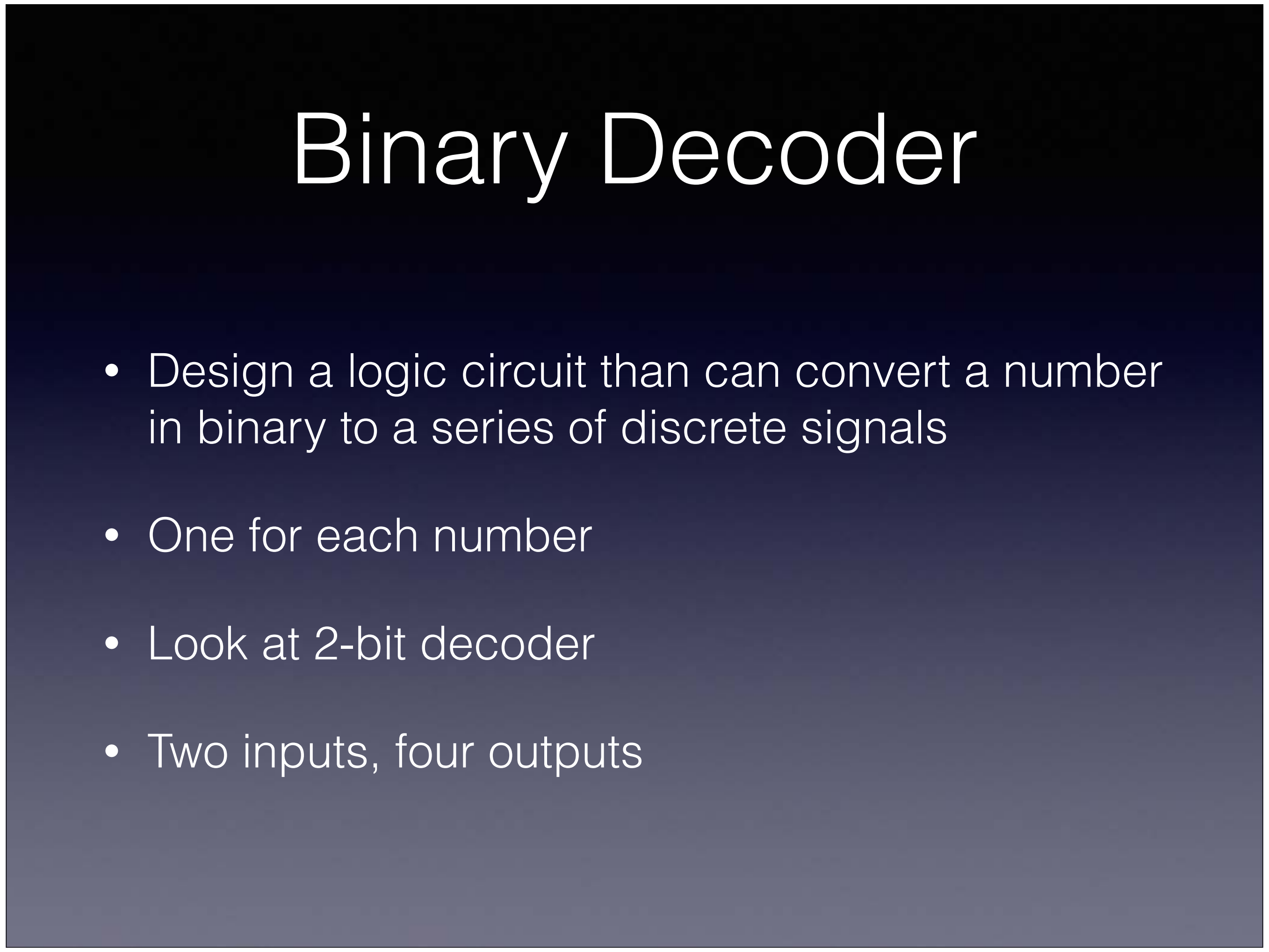
- Often helpful to start by building up the truth table
- Then work out the equations for each output for each line based on the input
- *OR* each of these together for each output
- Then simplify the equations



Simplifying

- Use the algebraic rules etc…
- Look for common sub-equations and move to a separate equation
- Aim is to use as few gates as possible
  - Saves money
  - Reduces propagation delay

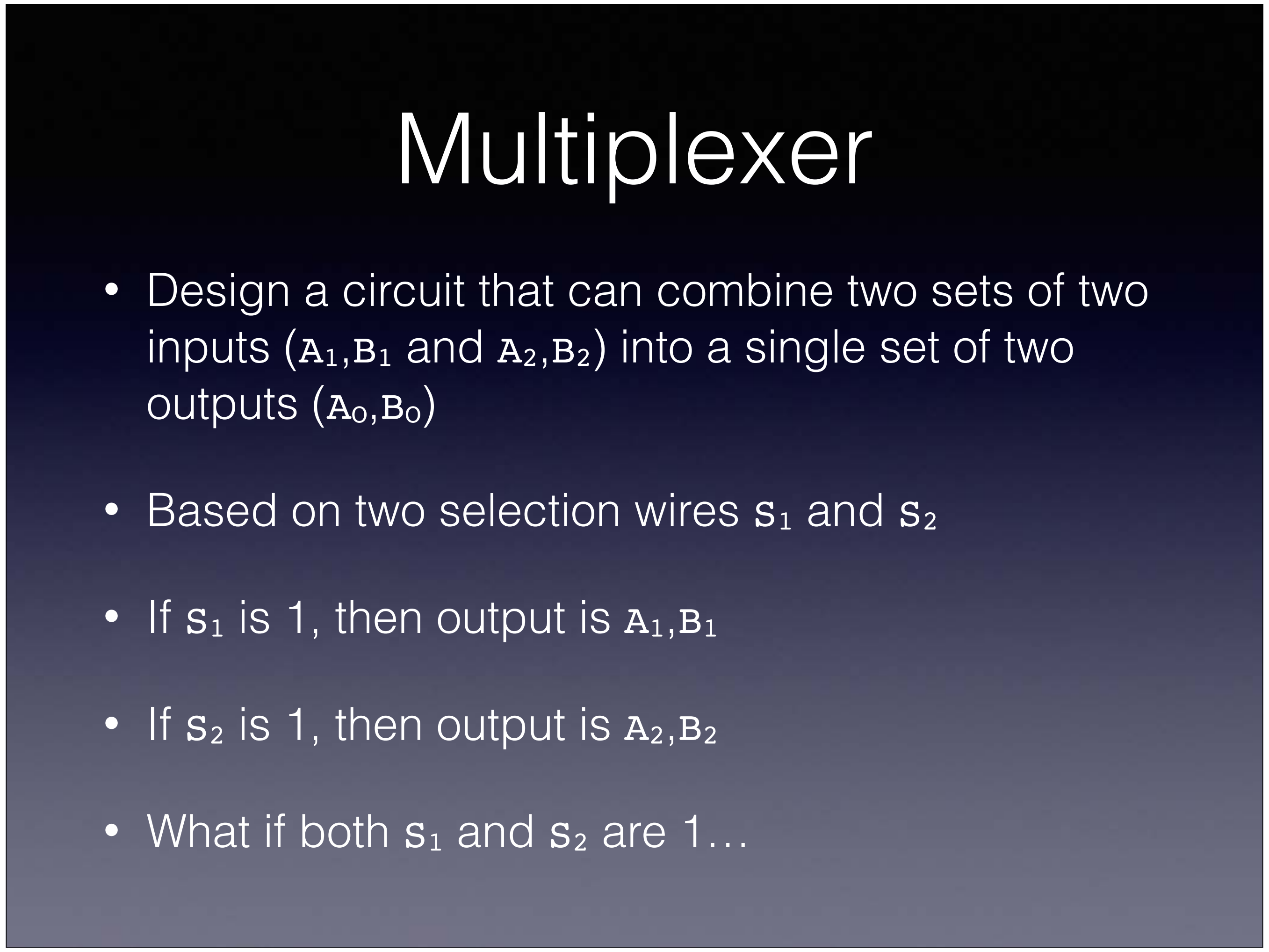
Propagation delay is the time taken for a change in the inputs to be reflected in the output of a gate (74LS00 is around 18ns). As gates are combined together the delays add together



Binary Decoder

- Design a logic circuit than can convert a number in binary to a series of discrete signals
- One for each number
- Look at 2-bit decoder
- Two inputs, four outputs

Go draw output on paper



Multiplexer

- Design a circuit that can combine two sets of two inputs (A_1,B_1 and A_2,B_2) into a single set of two outputs (A_o,B_o)
- Based on two selection wires S_1 and S_2
- If S_1 is 1, then output is A_1,B_1
- If S_2 is 1, then output is A_2,B_2
- What if both S_1 and S_2 are 1…

Not necessarily a problem, depends on how the circuit is driven…