CSE 260M / ESE 260
Intro. To Digital Logic & Computer Design

Bill Siever
&
Michael Hall

Announcements

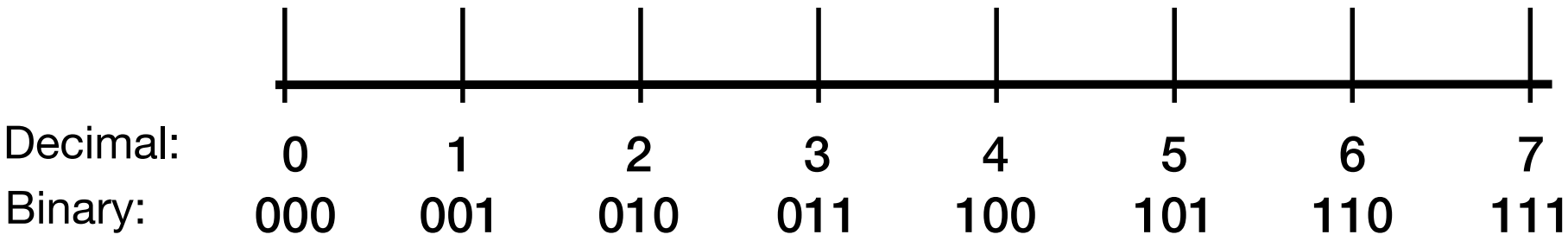
- Office hours: Starting today. See “help” page in Canvas or course site
- Homework 2B Posted / Due Sunday at 11:59pm
  - Dropboxes posted Thursday
- Lab kits: Handed out either this Thurs or next
  - Fee: $65 will be applied to student accounts in next ~2 weeks

Studio 2A Highlights

- Unsigned number line
- Two’s complement

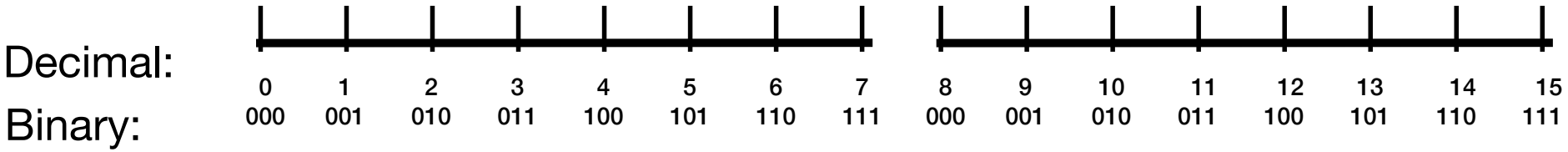
Last Time

- Studio: Binary Number Lines Extended



Last Time

- Studio: Binary Number Lines Extended

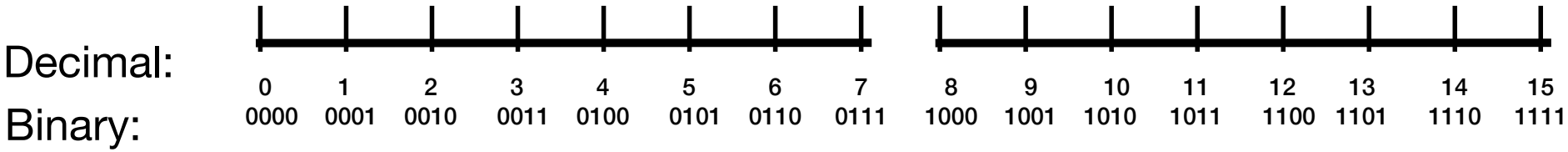


Last Time

- Studio: Binary Number Lines Extended

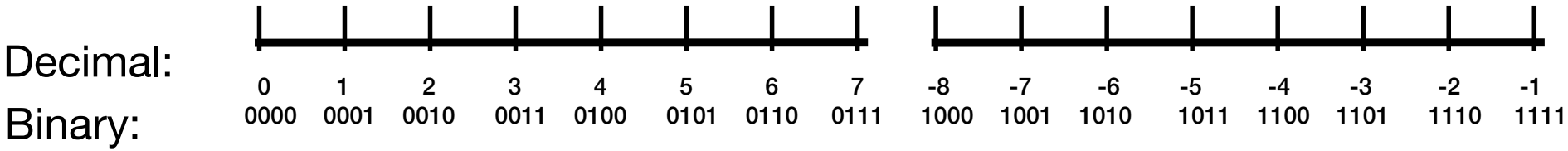
Last Time

- Studio: Binary Number Lines Extended



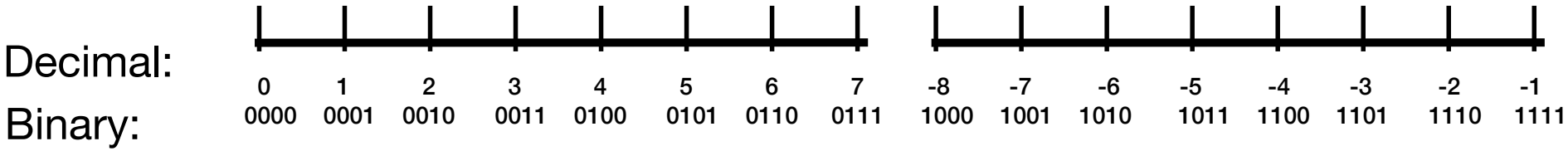
Last Time

- Studio: Two's Complement



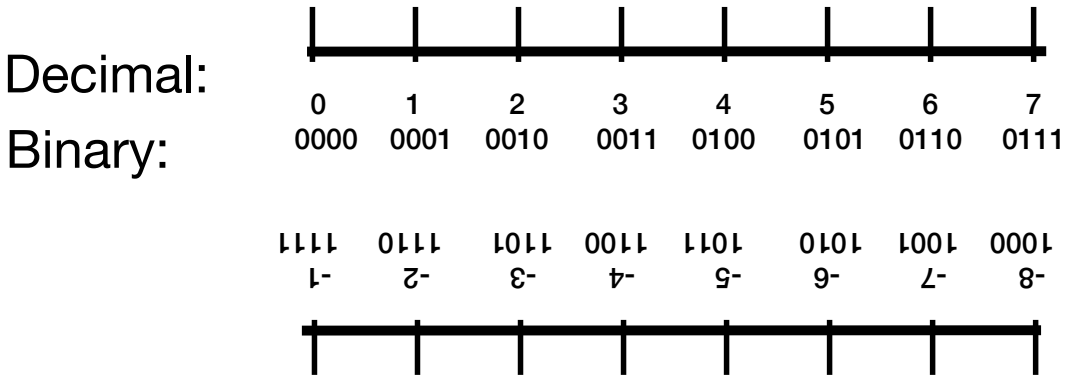
Last Time

- Studio: Two's Complement



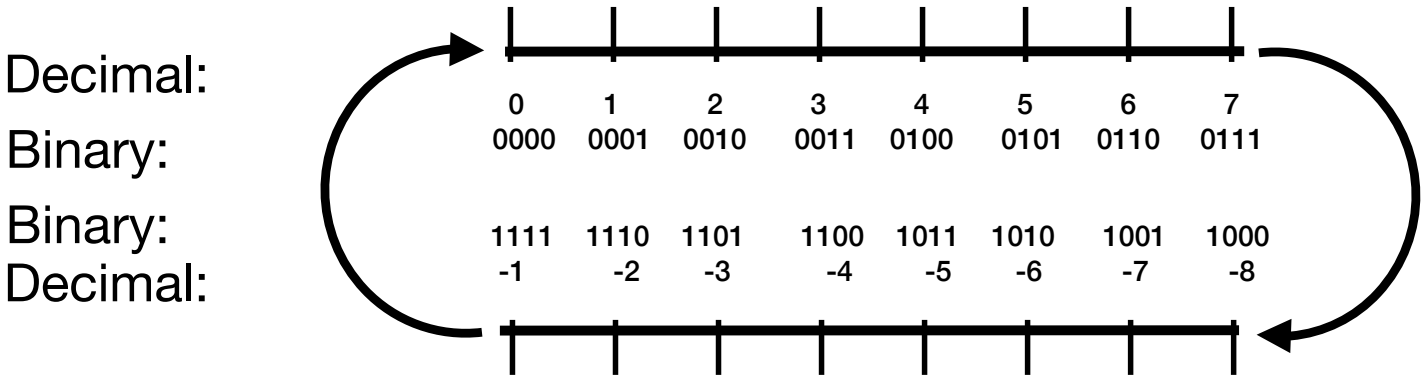
Last Time

- Studio: Two's Complement - Above/Below



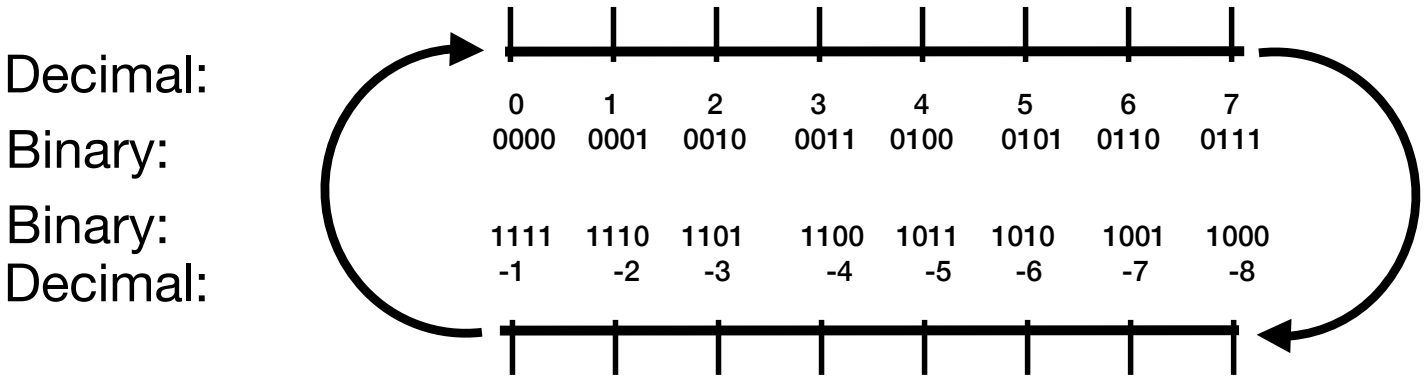
Last Time

- Studio: Two's Complement - Above/Below & Bitwise Inversion



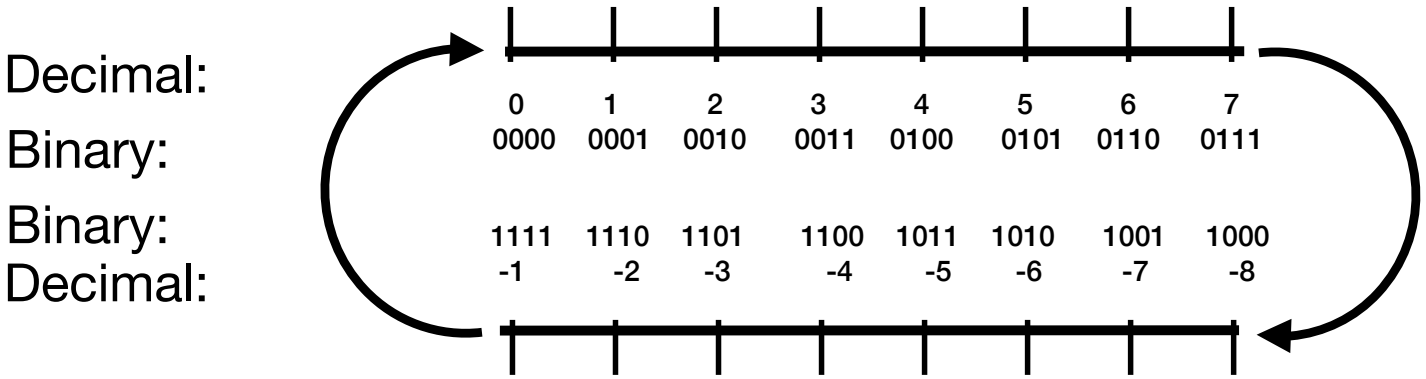
Last Time

- Studio: Two's Complement - Mathematical Negation ($-1\times$)



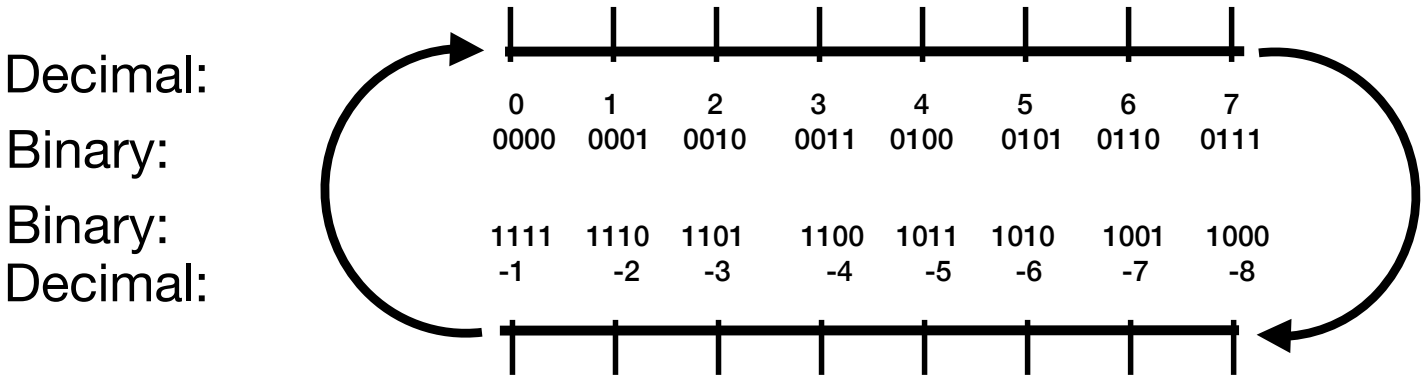
Last Time

- Studio: Two's Complement - Mathematical Negation (-1×6)



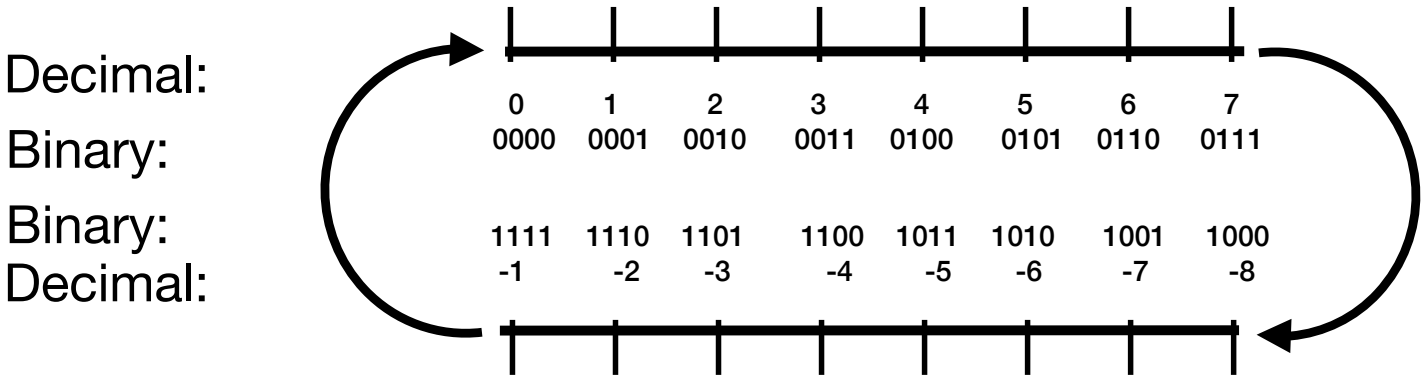
Last Time

- Studio: Two's Complement - Mathematical Negation (-1×6)



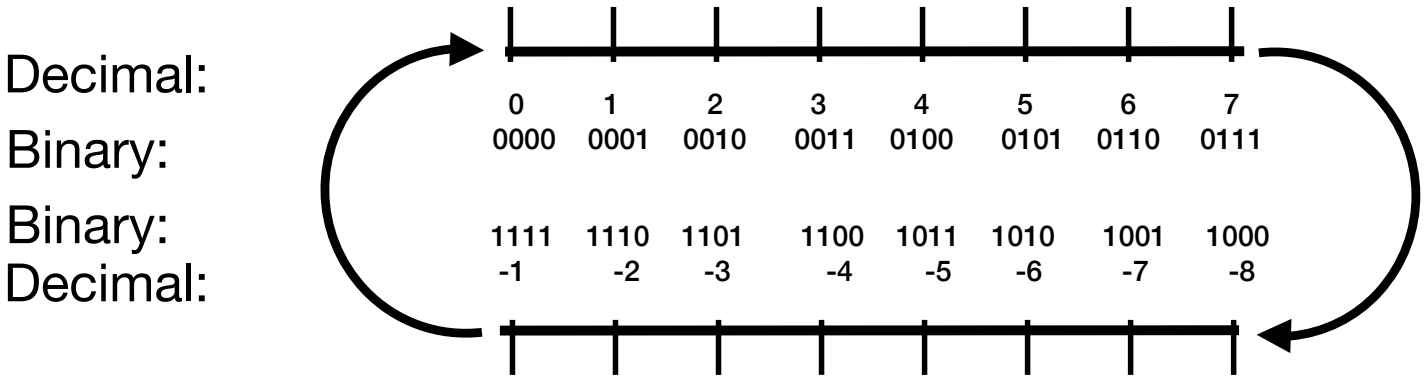
Last Time

- Studio: Two's Complement - Mathematical Negation (-1×6)



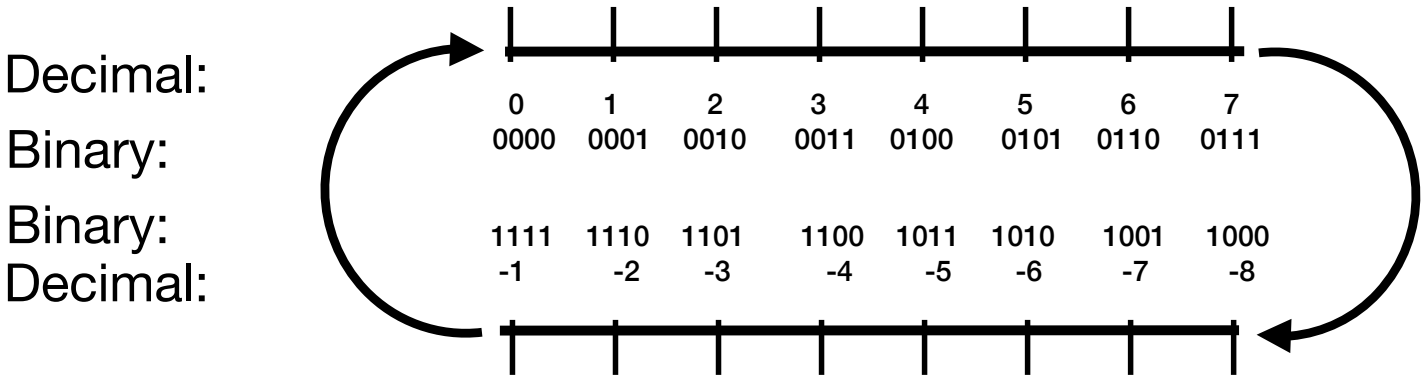
Last Time

- Studio: Two's Complement - Mathematical Negation ($-1 \times$ -6)



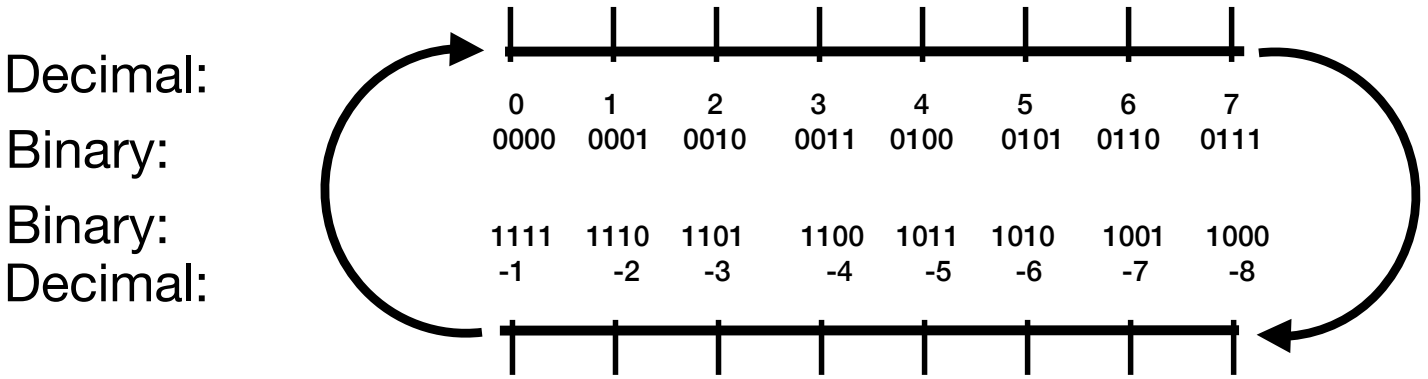
Last Time

- Studio: Two's Complement - Mathematical Negation ($-1 \times$ -6)



Last Time

- Studio: Two's Complement - Mathematical Negation ($-1 \times$ -6)



Last Time

- Studio: Two's Complement - Mathematical Negation ($-1 \times$ A)
- Logic: $\bar{A} + 1$
  - We are assumed to have "machines" for both;
    - Bitwise inversion (n invertors) and n-bit addition
      (more to come on n-bit addition)

Last Time

- Assume an n bit adder, like:

- n bit subtractor can be built:

Last Time: Tee

- $t_b + \overline{t_b}$
  - Theorem / Dual T5': $B + \overline{B} = 1$
  - Therefore "1"
- *Eventually*… JLS

Last Time

- JLS Logic Types: Unsigned & Assumes values are 0 at $t = 0$

Chapter 2

- Tables & Sum-of-products
  - A “can’t go wrong” way to build logic that behaves a specified way
- Karnaugh Maps: A form of optimization

Background: Minterms

- Minterms: Given n variables, a product (AND) containing all n exactly once, in either their original or negated form

  - Consider: $\overline{A} \cdot B \cdot C \cdot \overline{D}$
    Identify all possible combinations of inputs which make it true:

Chapter 2: Minterms

- Consider $n = 3$ and A, B, C; Which are Minterms? Which are *not* and why?
  - ABC
  - $AB\bar{A}$
  - $CB\bar{A}$
  - $\bar{A}C$

Minterms & Truth Tables

Cl	A	B		Carry Out	Sum
0+	0+	0	=	0	0
0+	0+	1	=	0	1
0+	1+	0	=	0	1
0+	1+	1	=	1	0
1+	0+	0	=	0	1
1+	0+	1	=	1	0
1+	1+	0	=	1	0
1+	1+	1	=	1	1

Minterms & Truth Tables

Cl	A	B		Sum
0+	0+	0	=	0
0+	0+	1	=	1
0+	1+	0	=	1
0+	1+	1	=	0
1+	0+	0	=	1
1+	0+	1	=	0
1+	1+	0	=	0
1+	1+	1	=	1

Minterms & Truth Tables

CI	A	B		Sum
0+	0+	0	=	0
0+	0+	1	=	1
0+	1+	0	=	1
0+	1+	1	=	0
1+	0+	0	=	1
1+	0+	1	=	0
1+	1+	0	=	0
1+	1+	1	=	1

- $n = 3$: CI, A, B
- **Where/when is each of these true?**
  - $\bar{C}I \cdot \bar{A} \cdot B$
  - $\bar{C}I \cdot A \cdot \bar{B}$
  - $CI \cdot \bar{A} \cdot \bar{B}$
  - $CI \cdot A \cdot B$

Minterms & Truth Tables

- Minterms are true for a *single* combination of inputs
  - This is essentially *selecting* a row of a truth table

Minterms & Truth Tables

Cl	A	B		Sum
0+	0+	0	=	0
0+	0+	1	=	1
0+	1+	0	=	1
0+	1+	1	=	0
1+	0+	0	=	1
1+	0+	1	=	0
1+	1+	0	=	0
1+	1+	1	=	1

- $n = 3$: Cl, A, B
- Where/when is Sum true (any place)?

Minterms & Truth Tables

CI	A	B		Sum
0+	0+	0	=	0
0+	0+	1	=	1
0+	1+	0	=	1
0+	1+	1	=	0
1+	0+	0	=	1
1+	0+	1	=	0
1+	1+	0	=	0
1+	1+	1	=	1

- $n = 3$: CI, A, B
- Where/when is *any* of these true?

$$Sum = \bar{C}I \cdot \bar{A} \cdot B + \bar{C}I \cdot A \cdot \bar{B} + CI \cdot \bar{A} \cdot \bar{B} + CI \cdot A \cdot B$$

Truth Table -> Sum of Minterms

Canonical Form

Important!

- Any simple function (mapping) can be represented as a truth table
  - n-bit binary numbers can be used to represent all the inputs
    - The table will need 2^n rows to represent all the possible combinations of inputs
  - m-bit binary numbers can represent the output(s)
  - *Each* of the m bits of output can be represent by a sum-of-products (sum of minterms) equation.
    - There's a minterm for each place the bit of m is a 1 (true)
    - Canonical form = The "sum" of these Minterms

Sum-of-Products

- All our combinational logic *could* be represented in a table
- All the outputs can be represented as equations
- Those equations can be realized with just the concept of AND, OR, & NOT
  - I.e., we can build computing machines for anything we can represent in a table if we have AND, OR, or NOT.
- The idea of Tables and “Look Up Tables” (LUTs) is really useful!

Give an SOP equation for…

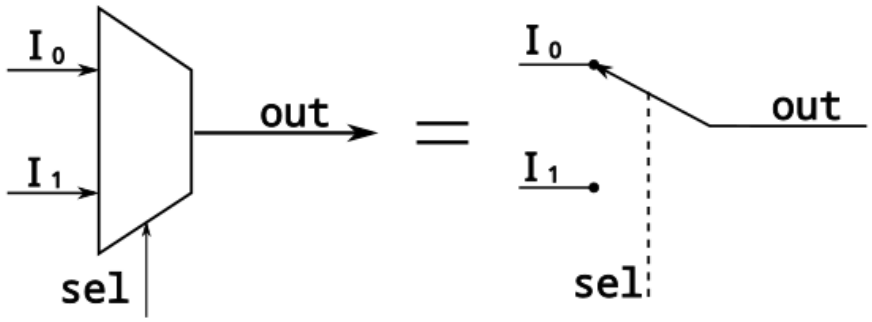
Inputs			Output
A	B	C	O
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Give an SOP equation for…

Inputs			Output
S	D1	D0	O
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Give an SOP equation for…

Inputs			Output
Sel	I1	I0	O
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



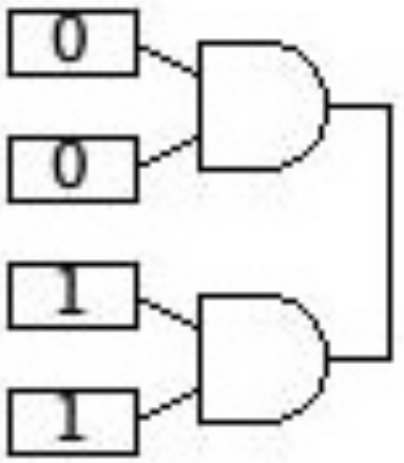
https://commons.wikimedia.org/wiki/File:Multiplexer2.svg

Product-Of-Sums

- Alternative to SOP: Uses maxterms (SUM of all input variables) in large product
  - Form: $Y = (A + B + C) \cdot (A + \overline{B} + C) \cdots$
- Can be constructed from table by focusing on:
  1) rows with zeros
  and
  2) sums that will be zeros in only those rows
- Sum-of-Prod: Smaller when more 1s than 0s in table;
  Otherwise Prod-of-Sums is smaller
- This class: slightly focused on Look-up Table (LUT) concept / usually favor SOP

Real Circuits: Xs and Zs

- 0s and 1s represent real-world, continuous values, like voltages
  - Ex: 0 = 0v (gnd); 1 = 5v
    - What's 2.3v?
    - What happens if a 0v wire is connected to 5v wire? ("Contention")
  - X: That's illegal / don't know



Simulator / Language Types

- Bits & Types: `10101100` can have different interpretations
  - Programming languages use data types
- Verilog (Chapter 4)'s `logic` type:
  - `0`, `1`, `X` (unknown), `Z` (high-impedance)
  - Other simulators often use `X` for initial value
    (Helps catch errors and misunderstandings earlier vs. building on a bad assumption!)

Real Circuits: Xs and Zs

- 0s and 1s represent real-world, continuous values, like voltages
  - Ex: 0 = 0v (gnd); 1 = 5v (relative to that ground)
    - Voltage is a *relative* measure
      (like water pressure: it's the difference between two points)
  - Z: "Floating" value / disconnected
    - Sometimes useful to "disconnect" something to prevent contention
      (to share wires with different things in control at different times)
    - Sometimes an error when *nothing* is connected
      (Behavior depends on technology and conditions; Can be random or influenced by external things — like moving a hand near a circuit!)

Circuit “Optimization”

- Time or performance?
- Number of parts?
- Total cost?
- Combination: E.g., Cheapest way to achieve a specific level of performance

Circuit “Optimization”

- Logic Minimization
  - Canonical Form is seldom the minimum number of parts
  - Can “combine” overlapping terms (implicants / product)
    - Prime Implicant: Can’t be further reduced
    - Ex: $A \cdot B \cdot C + A \cdot \overline{B} \cdot C$
      - True when $A \cdot C$. The B and $\overline{B}$ cancel

Karnaugh (K) Maps

- A *visual* way to do term optimization
- Rely on tables that allow easy identification of ways to combine implicants
  - Uses Gray code ordering, *not counting order!!!*
- Only useful for up to 4 variables. I.e., small problems

Karnaugh (K) Maps

- Goal: Cover all 1s with circles
  - As few circles as possible & as large as possible
  - Span rectangles with sides of 1, 2, 4, or 8
  - Top/bottom and left/right wrap!

Give an opt. equation for…

Inputs			Output
S	D1	D0	O
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

		D1/D0			
		00	??	??	??
S	0				
	1				

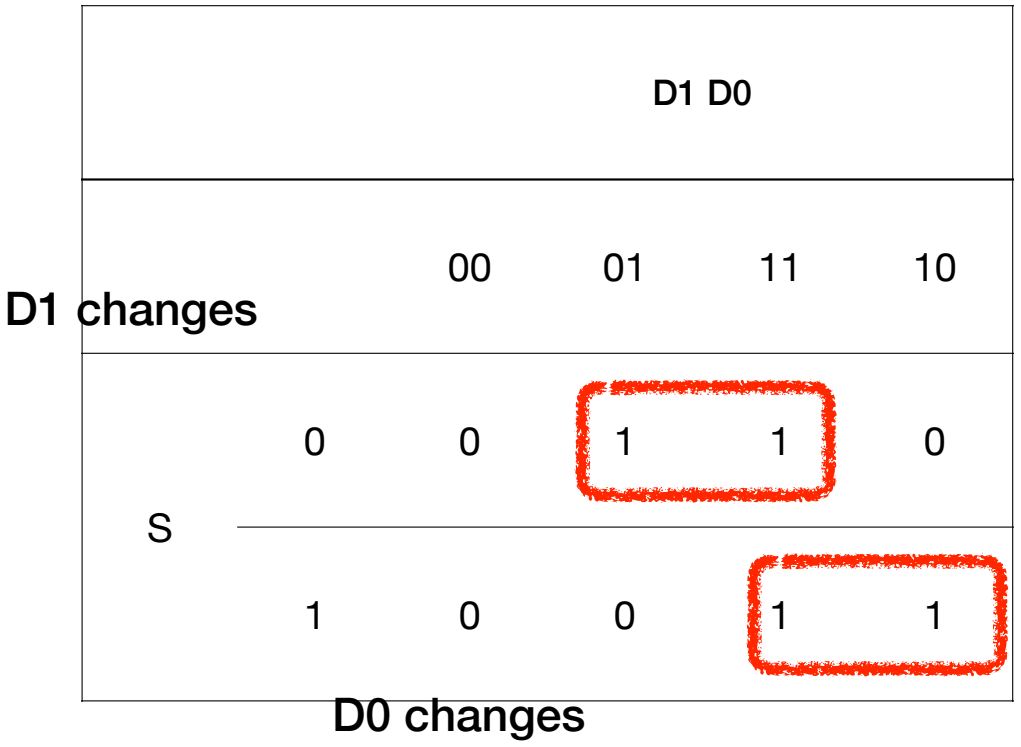
Give an opt. equation for…

Inputs			Output
S	D1	D0	O
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

		D1/D0			
		00	01	11	10
S	0				
	1				

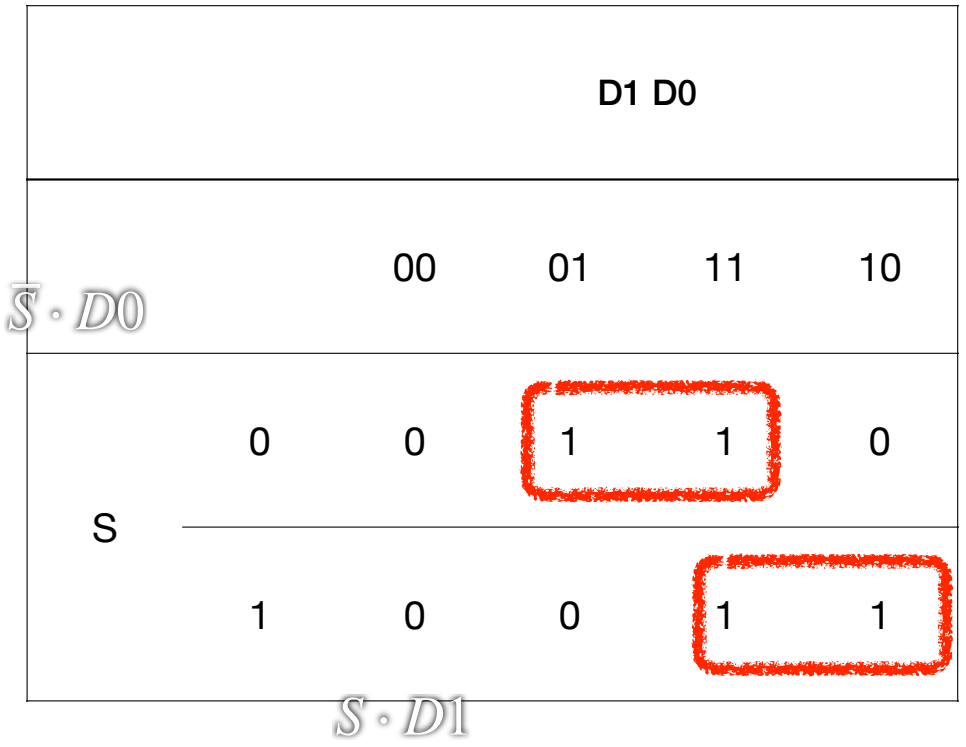
Give an opt. equation for…

Inputs			Output
S	D1	D0	O
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Give an opt. equation for…

Inputs			Output
S	D1	D0	O
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



E2: Give an opt. equation for…

Inputs			Output
A	B	C	O
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

		B/C			
		00	01	11	10
A	0				
	1				

E3: Give an equation for…

Inputs			Output
X	Y	Z	O
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

		X/Z			
		00	01	11	10
X	0				
	1				

2.8: More Parts

- Q: We want a 4-to-1 multiplexor. How big is the full truth table?
- Q: Our CPU may need a 32-to-1 multiplexor. How big is the truth table?
- We need a new approach
  - Hierarchical construction

Questions

- Glitches: Partly a result of different path lengths through a circuit
  - An analogy of one type of glitch via different paths
  - Runner Red & Blue both are going to Brookings.
    - They both leave Hillman 70 at the same time
    - They both run about the same speed
    - The each have have a dye to dump in the the fountain (which is initially clean water)
    - Red runs directly to Brookings. Blue goes by Siegle hall (far end of campus) first
  - Describe what happens to the color of the water?

Questions

- Real-world implications of propagation delay
  - CPU Clock speed is based on the propagation delay
    - Less delay can lead to faster clock speeds and faster programs
- Floating values (covered)
- Multiplexors vs. Encoders vs. Demux vs. Decoder: More on Encoders and Decoders later
  - Encoders and Decoders are about encoding or decoding binary numbers
  - Multiplexors are about “picking” an input of interest. They are a vital part of the data path of CPUs.
- Circuit diagrams get messy as they get bigger. Is there a better way?
  - Hierarchical design (Chapter 3-6)

Questions

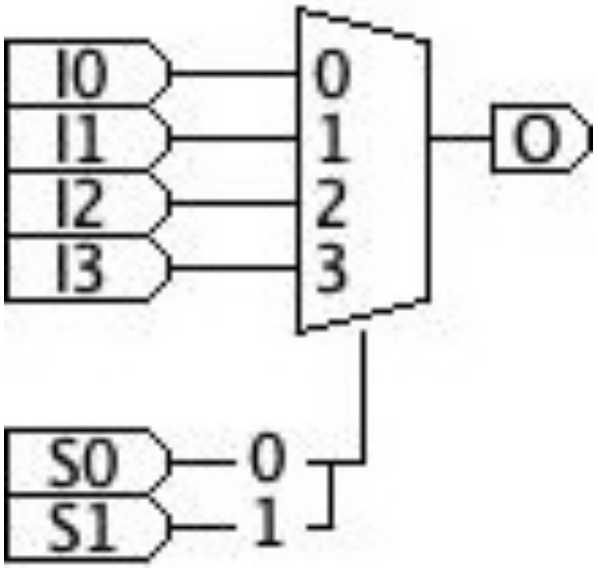
- How do I understand K-maps?
  - Practice & review
- What about more than 4 variables:
  - Quine–McCluskey algorithm

Hierarchical Approach

- We've created a 2-to-1 MUX
  - Construct a 4-to-1 MUX using 2-to-1 MUXes
  - Focus on desired behavior using existing parts

4-to-1 MUX Behavior

- Behavioral Description as a Table



Inputs of Interest		Output (In terms of Inputs)
S1	S0	O
0	0	I0
0	1	I1
1	0	I2
1	1	I3

Hierarchical Construction of 4-input Mix

Review / Catchup

- SOP equations
  - Table to describe behavior
  - Product equation for “matching” exact pattern
  - Sum of Products for full, single-bit behavior
  - Overall circuit structure
  - Pros: Can’t go wrong! Can build any machine….Can be messy though.
  - Product of sums: Apply DeMorgan’s law or focus on 0s and construct