

Lecture 5: Boolean Logic

Spring 2024
Jason Tang

Topics

- Boolean logic (review)
- Combinational logic (mostly review)
- Gate delays

Binary Algebra (Review)

- Values are either 0 or 1
- Three basic operations:
 - OR (also known as **sum**): $A + B$, **A | B**
 - AND (also known as **product**): $A \cdot B$, **A & B**
 - NOT (also known as **negation** or **inversion**): \bar{A} , **~A**

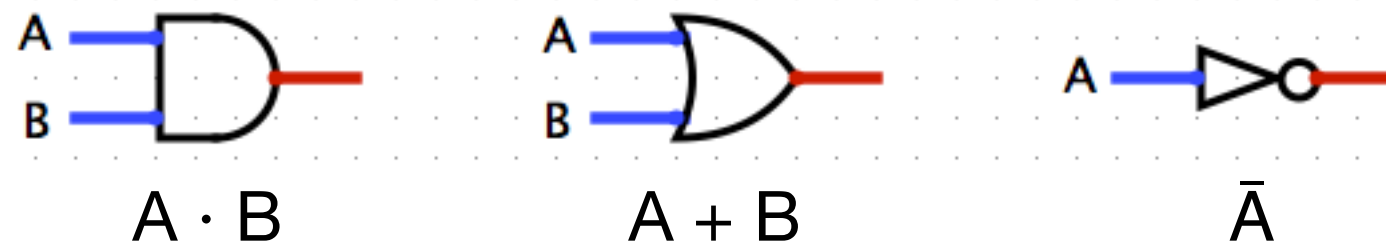


Boolean Laws

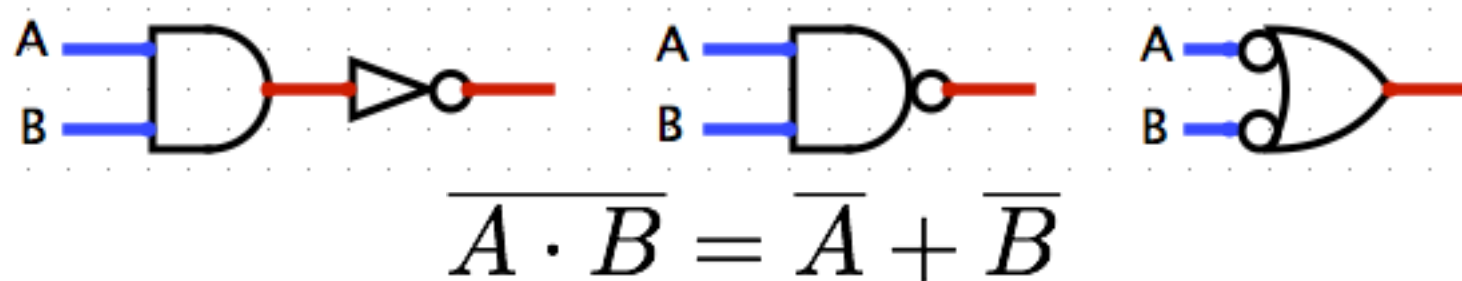
- Identity Law: $A + 0 = A$, and $A \cdot 1 = A$
- Inverse Laws: $A + \bar{A} = 1$, and $A \cdot \bar{A} = 0$
- Commutative Laws: $A + B = B + A$, and $A \cdot B = B \cdot A$
- Associative Laws: $A + (B + C) = (A + B) + C$, and $A \cdot (B \cdot C) = (A \cdot B) \cdot C$
- Distributive Laws: $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$, and $A + (B \cdot C) = (A + B) \cdot (A + C)$
- De Morgan's Laws: $\overline{A \cdot B} = \bar{A} + \bar{B}$, and $\overline{A + B} = \bar{A} \cdot \bar{B}$

Gates

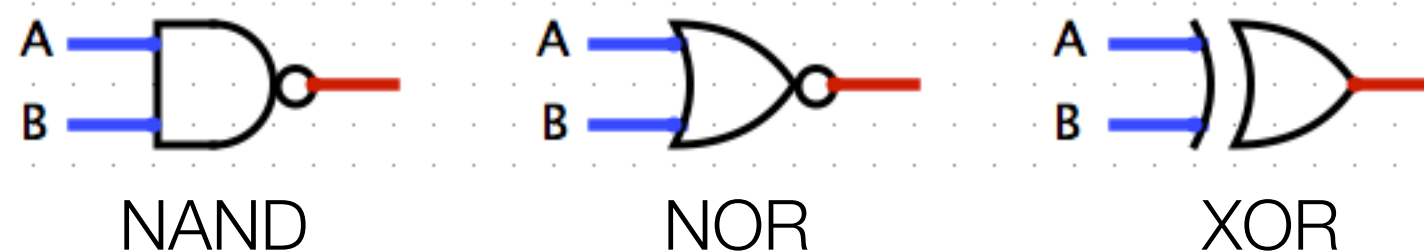
- Basic logic functions implemented using the three basic gates:



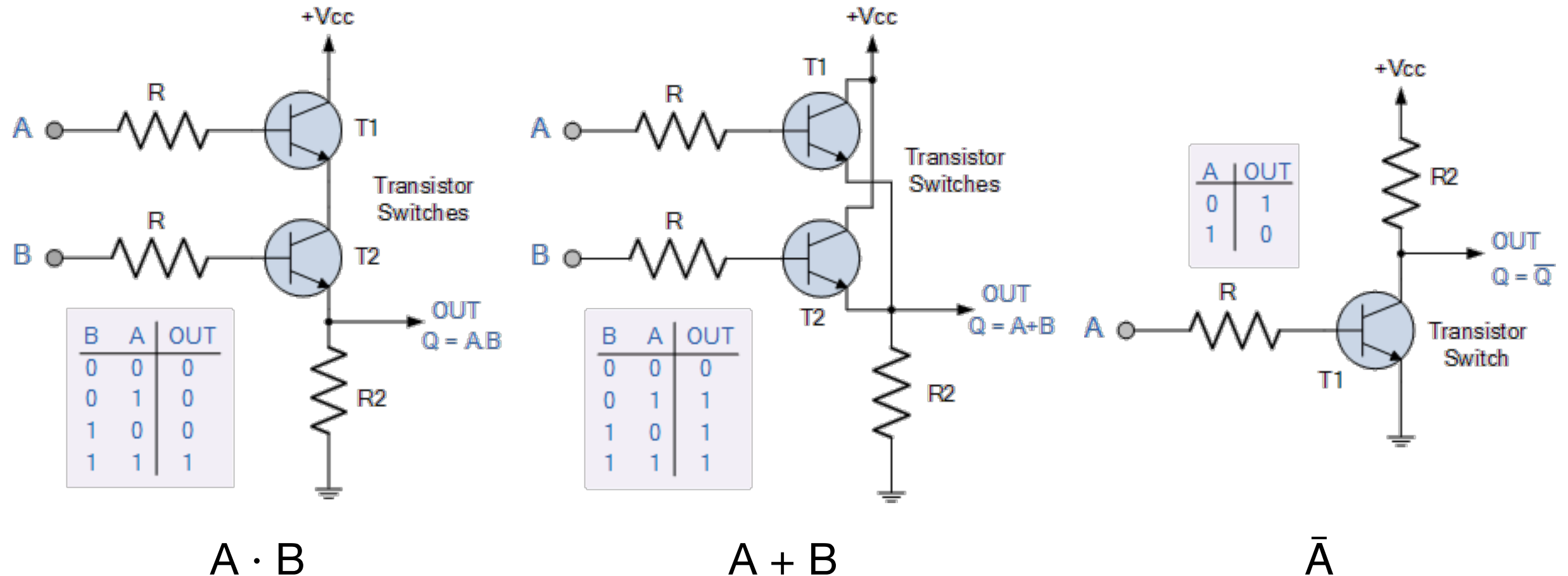
- Inverting an input or output from a gate is often abbreviated as a bubble:



- Can combine these gates into NAND, NOR, and XOR gates:



Physically Building Gates

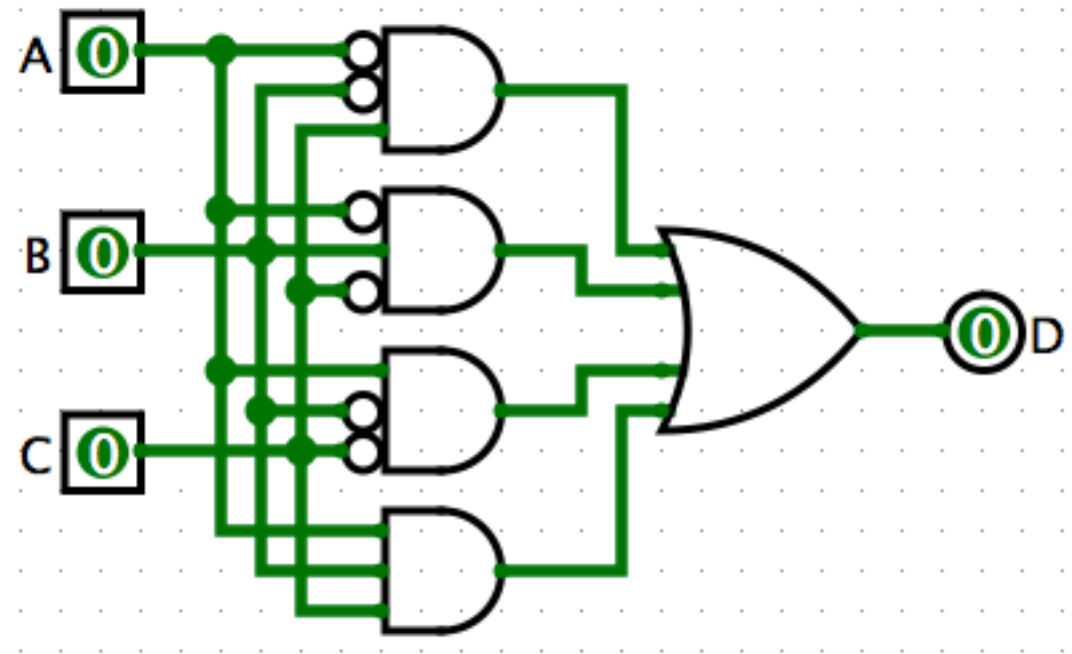


Canonical Form

- Any logic function can be expressed in **canonical form**, using two levels of gates:
 - **Sum of Products**: logical sum of products (sum of **minterms**)
 - **Product of Sums**: logical product of sums (product of **maxterms**)
- Both are equivalent, as per De Morgan's laws

Sum of Products Example

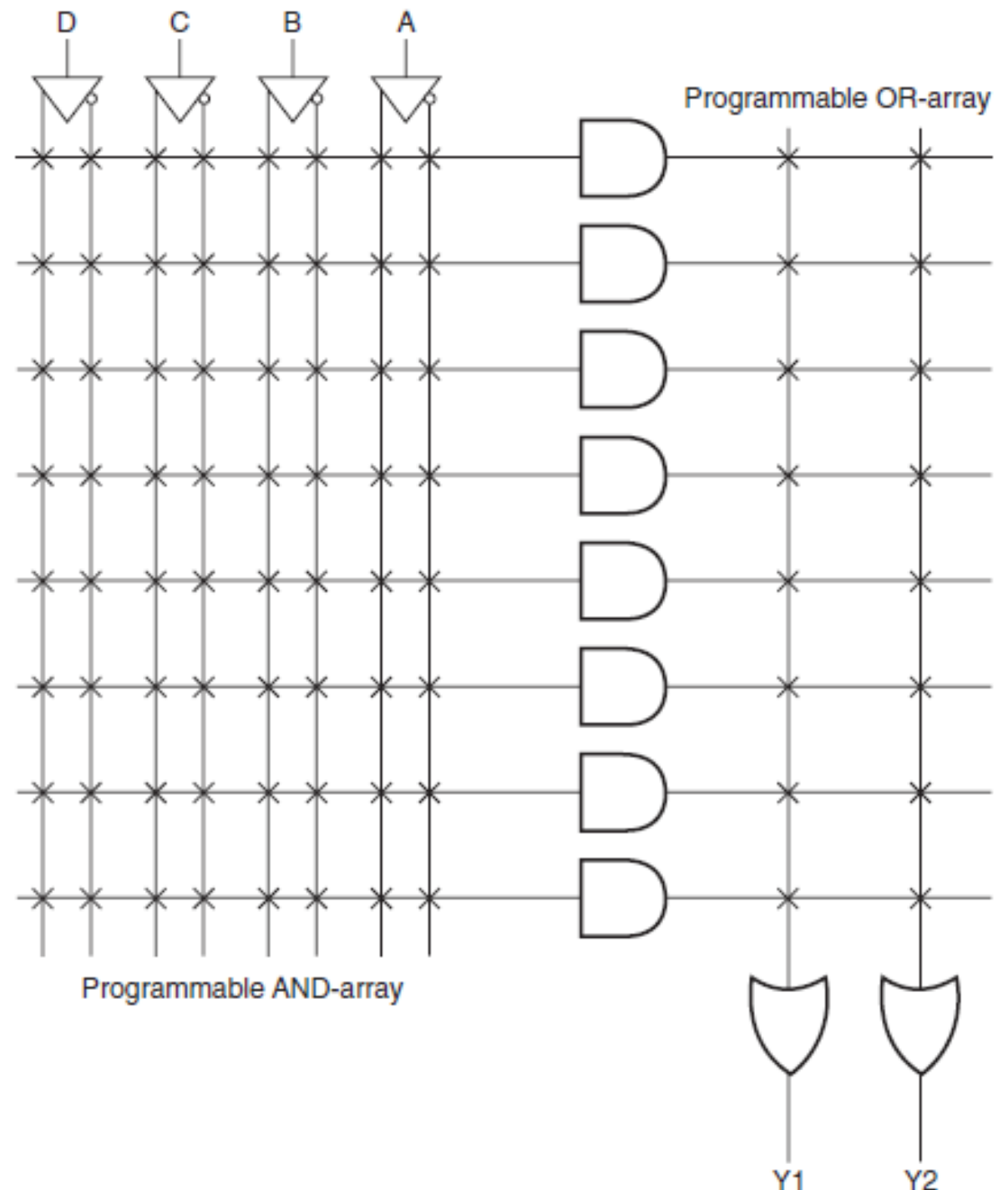
Inputs			Output
A	B	C	D
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



$$D = (\overline{A} \cdot \overline{B} \cdot C) + (\overline{A} \cdot B \cdot \overline{C}) + (A \cdot \overline{B} \cdot \overline{C}) + (A \cdot B \cdot C)$$

Programmable Logic Array

- Device that implements combinatorial logic, via a plane of AND and OR gates
- Designer generates the sum of products
- PLA is then programmed, by removing unneeded fuses
- Easiest to use when equation is **optimized**



Don't Cares

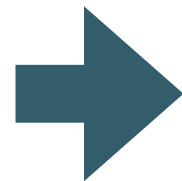
- When implementing logic, sometimes a particular value does not matter
 - **Input don't care** occurs when an output depends only on some input lines; other input lines do not affect output
 - **Output don't care** occurs when an output value is unused later in the circuit
- Represent both types with **x**s in truth table

Karnaugh Map

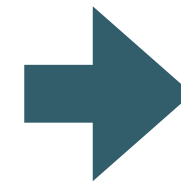
- Method of simplifying a logic function to a canonical form
- Given a truth table, create a grid of its output values
- Construct groups, potentially overlapping, of all 1s on the map
 - Each group must be rectangular and its area must be a power of two
 - Resulting groups are the minterms

Karnaugh Map Example 1

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



		B	
		0	1
A	0	1	1
	1	0	1



		B	
		0	1
A	0	1	1
	1	0	1

- One canonical form is to group top row together, then add bottom-right:

$$C = \overline{A} + A \cdot B$$

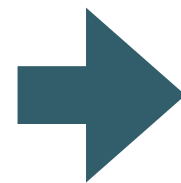
- Alternatively, have two groups that partially overlap top-right cell:

$$C = \overline{A} + B$$

K-Map Example 2

- When don't cares exist, add them to a group if it results in a larger group
- Groups can “wrap around” the edges

Inputs			Output
A	B	C	D
0	0	0	0
0	0	1	1
0	1	X	X
1	0	0	1
1	0	1	X
1	1	0	1
1	1	1	0



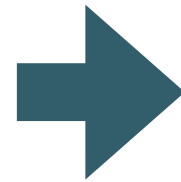
		B/C			
		00	01	11	10
A	0	0	1	X	X
	1	1	X	0	1



$$D = A \cdot \overline{C} + \overline{B} \cdot C$$

K-Map Example 3

Inputs				Output
A	B	C	D	E
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0



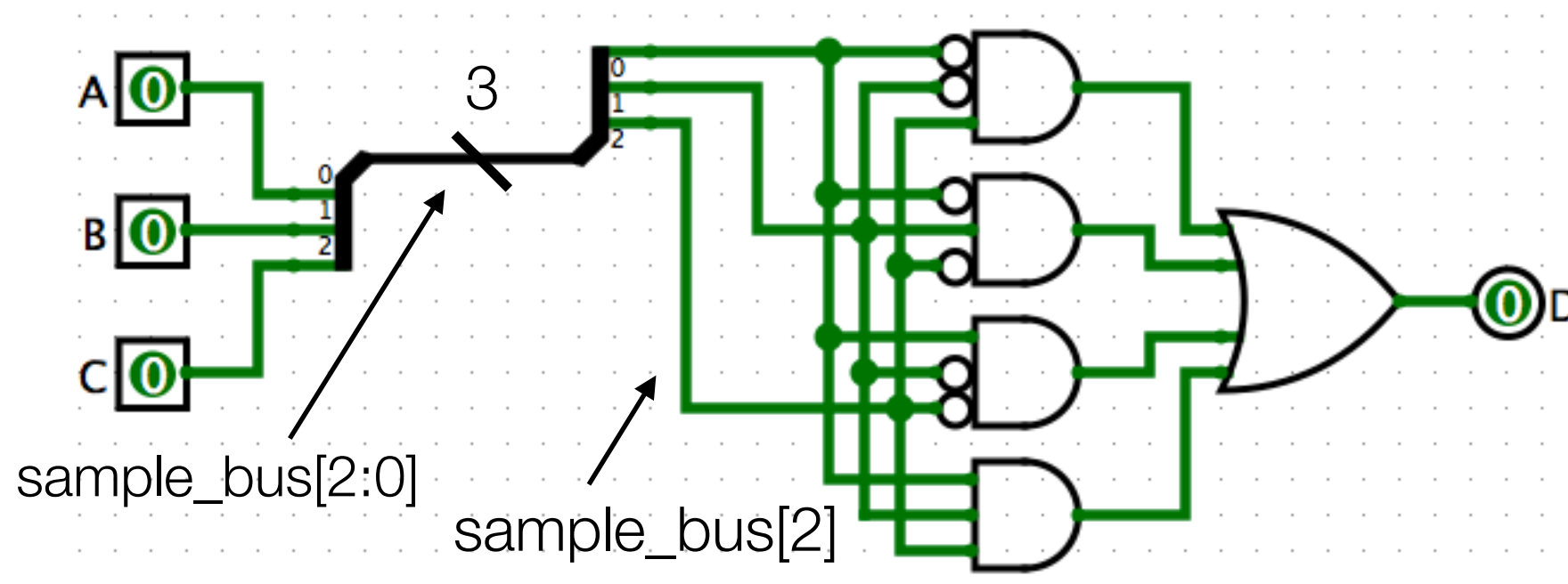
		C/D			
		00	01	11	10
A/ B	00	0	0	0	0
	01	0	0	0	1
	11	1	1	0	1
	10	1	1	1	1



$$E = A \cdot \overline{C} + A \cdot \overline{B} + B \cdot C \cdot \overline{D}$$

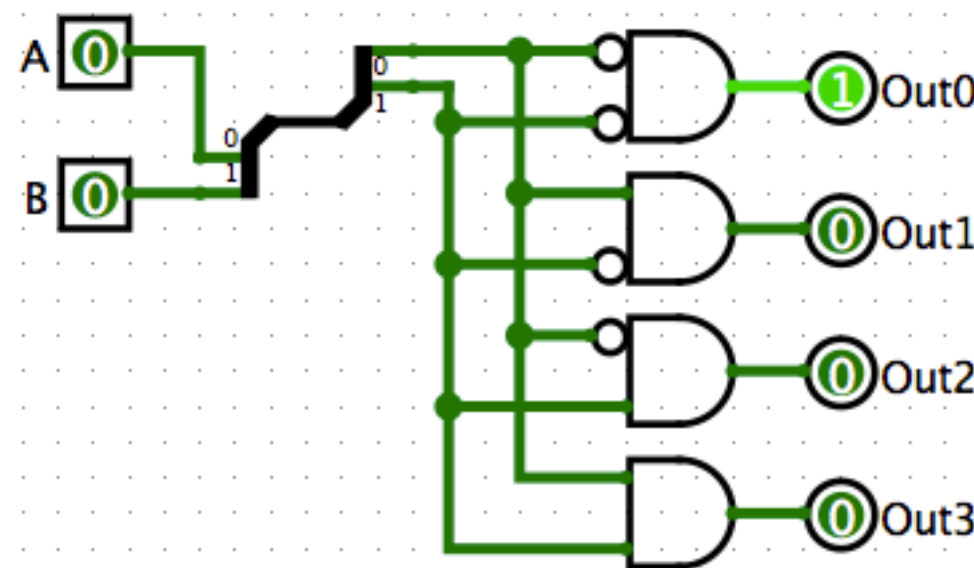
Wire Bundles

- Drawing individual wires will quickly clutter diagrams
- Multiple wires can be grouped together into a **wire bundle** (or **bus**)
 - Number of wires tied together is known as its **width**
- In diagrams, **splitters** break up and join wire bundles into individual wires



Decoding / Encoding

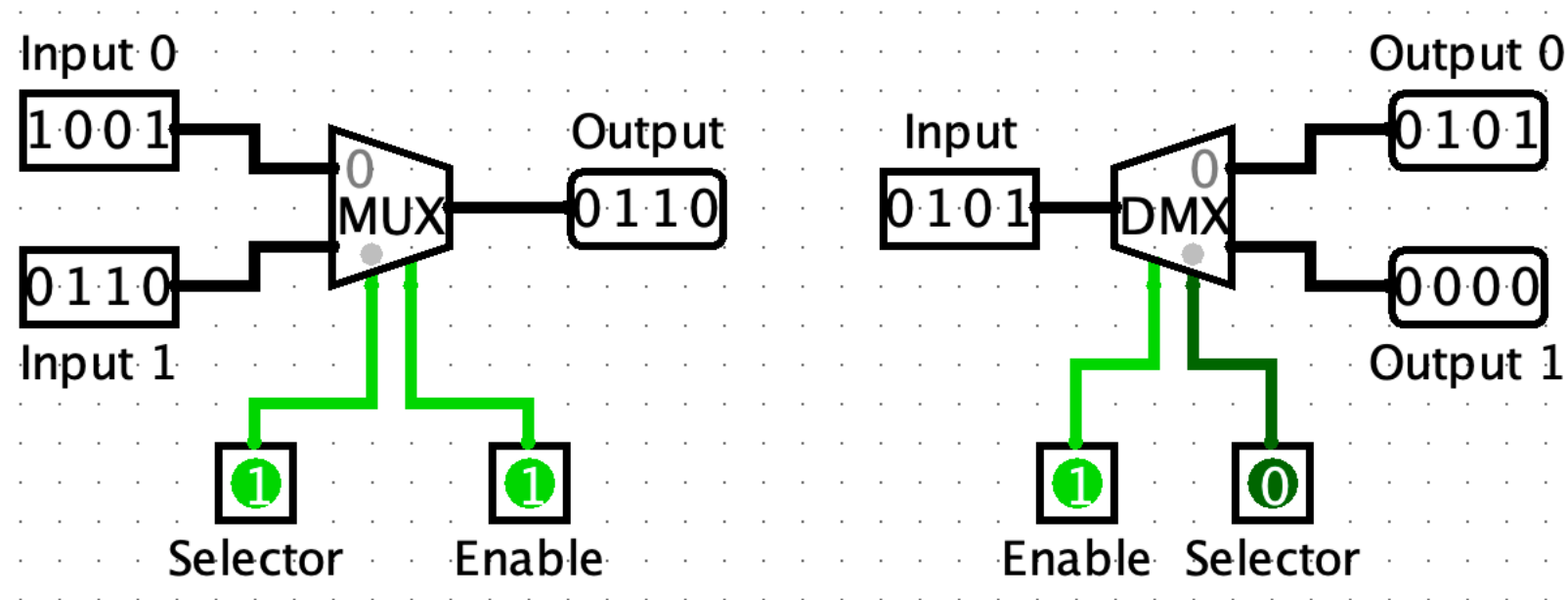
- **Decoder** is a **logic block** with n input wires and 2^n output wires
 - Given binary input **X**, output a 1 to the wire numbered **X**, and 0 to the rest



- **Priority Encoder** is the opposite: given a wire bundle with n width, output the binary value for the highest numbered input whose value is 1

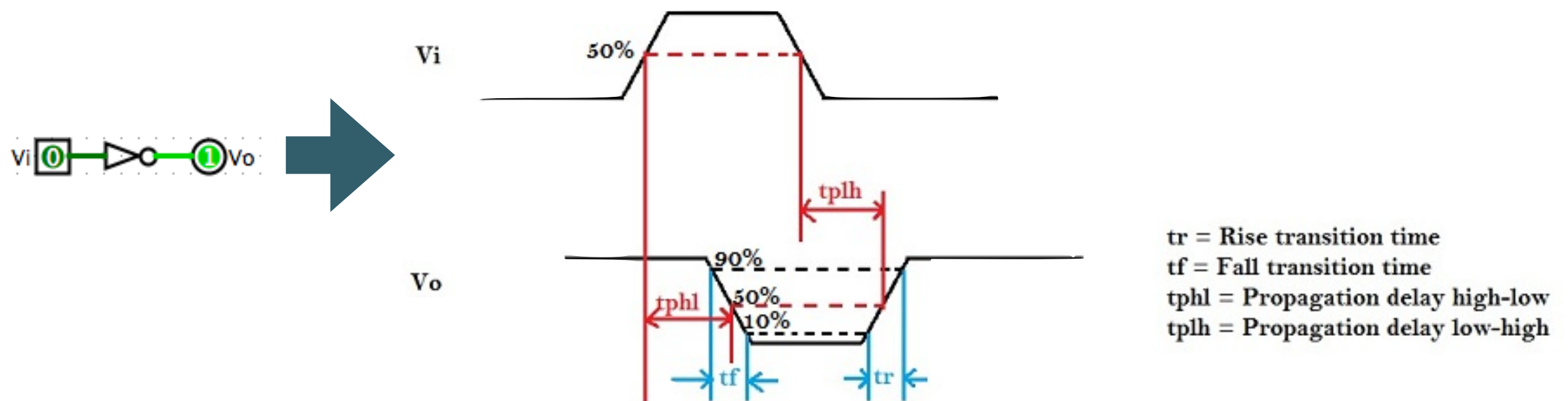
Multiplexers / Demultiplexers

- **Multiplexer** (also known as multiplexor or **selector** or simply **mux**) chooses from several possible input values given a selector value
- **Demultiplexer** (**demux**) performs reverse operation



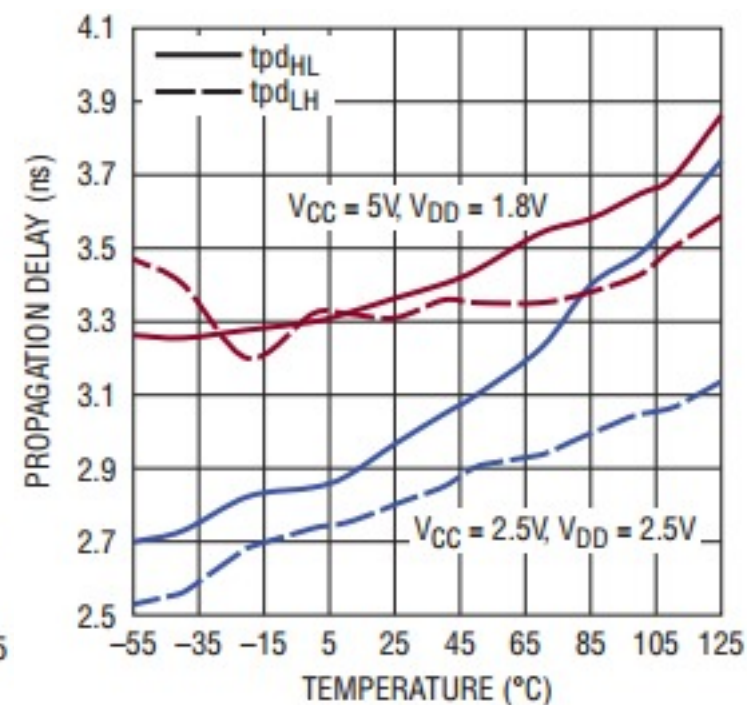
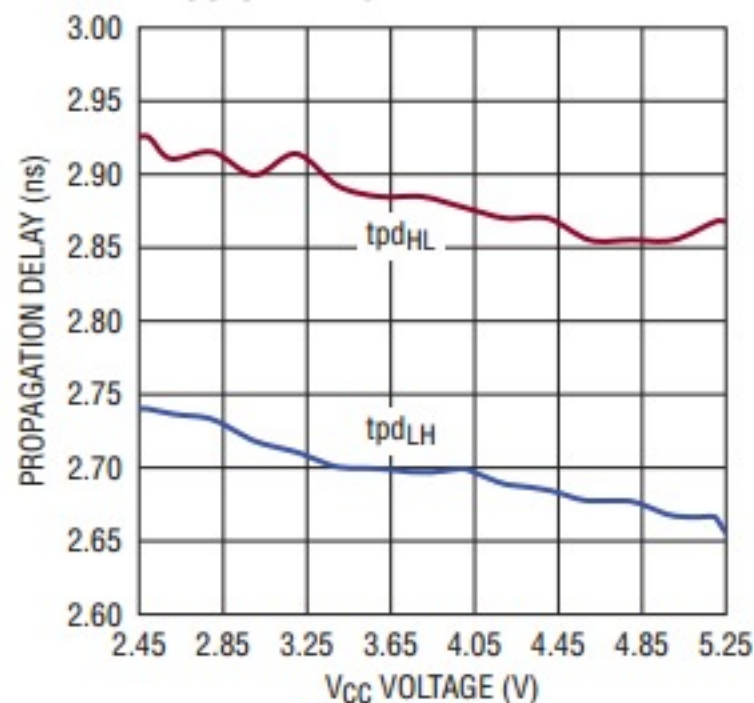
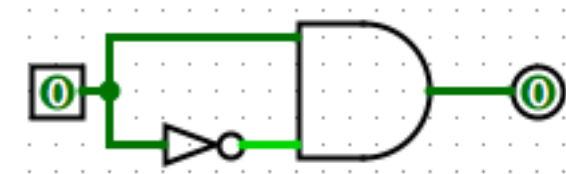
Gate Delays

- Also known as **propagation delay**, is time it takes for a **stable output** to be calculated given changes to input values
 - Changing a binary value is not instantaneous - limited by physics
- Depends upon wire length, thickness, material, and number of gates



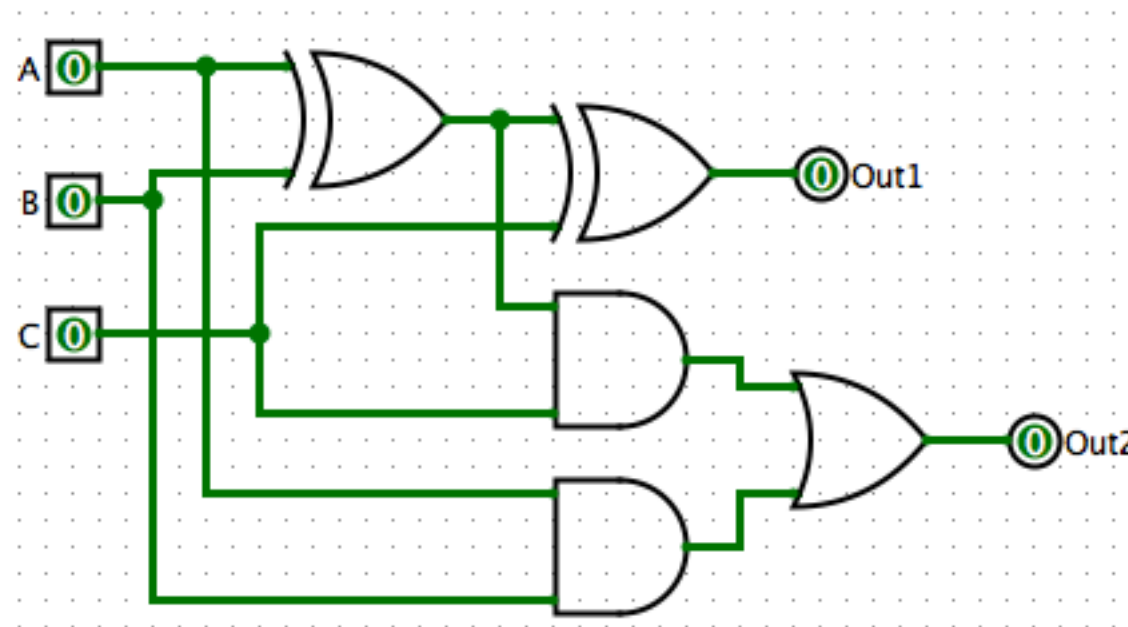
Gate Delays

- Because signals take time to propagate, a circuit may temporarily calculate the wrong value
 - Normally not a problem, except for **edge triggered** circuits and without a **debouncer**
- Manufacturers' **datasheets** should list delays based upon voltage and temperature



Calculating Gate Delay

- Supposing the delay for wires is 0 ns and each gate is 1 ns, what is the gate delay from inputs to each of these outputs?



Clocking Diagram

- Logic components compute new values whenever the clock changes, either on a **leading edge** or a **falling edge**
 - Most components have a **clock input** line along with its **data lines**
- Clock rate is influenced by maximum gate propagation delay

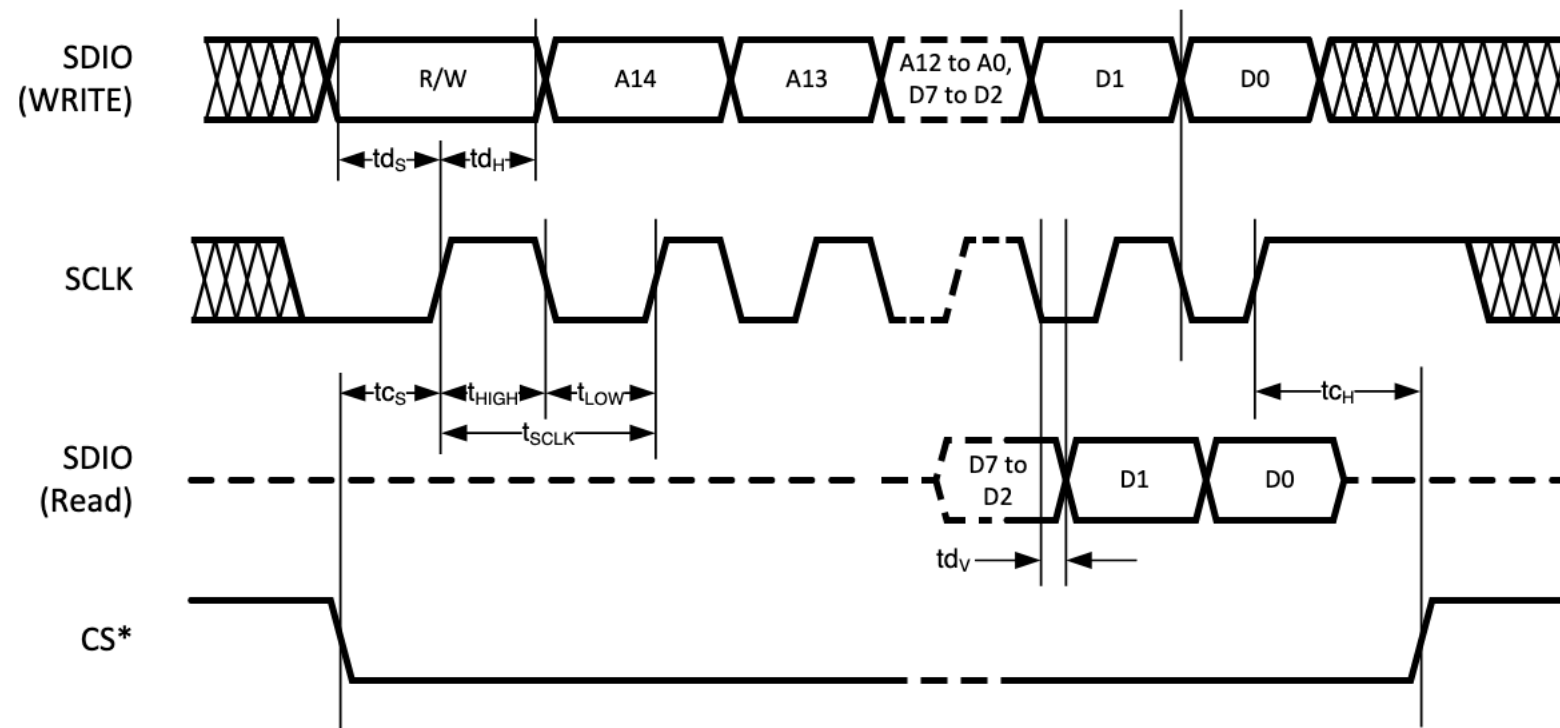


Figure 1. SPI Timing Diagram