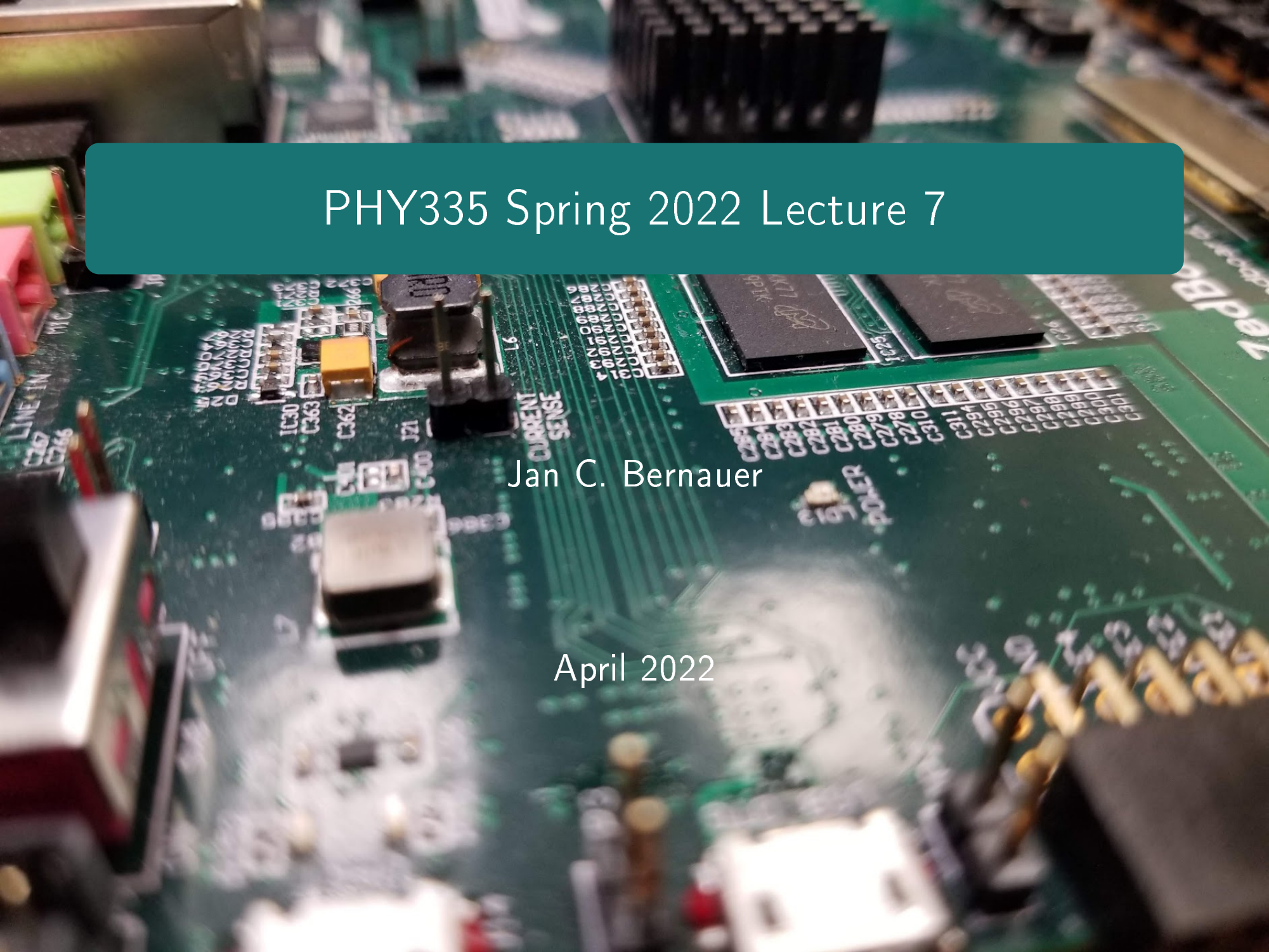
PHY335 Spring 2022 Lecture 7



Jan C. Bernauer

April 2022

Digital logic

- Analog signals degenerate through noise.
- Replace continuous analog signal levels with only two, discrete levels (/windows/ranges)

Digital logic

- Analog signals degenerate through noise.
- Replace continuous analog signal levels with only two, discrete levels (/windows/ranges)
- These are called logic states
  - HIGH vs. LOW (voltage level)
  - TRUE vs. FALSE (Boolean logic)

Digital logic

- Analog signals degenerate through noise.
- Replace continuous analog signal levels with only two, discrete levels (/windows/ranges)
- These are called logic states
  - HIGH vs. LOW (voltage level)
  - TRUE vs. FALSE (Boolean logic)
- A system where HIGH represents TRUE is called active-high, a system where LOW represents TRUE is called active-low

Voltage ranges for HIGH and LOW

- Separate for input and output: Allows a bigger range for input than guarantees for the output.
- This gives noise immunity
- Classic: TTL (Transistor-Transistor-logic):
  - Input: <+0.8V is low, >+2.0V is high (meaning that a TTL compatible device is allowed to transition from low to high anywhere between 0.8 and 2V
  - Output worst case: low +0.4V, high +2.4V
  - So only 0.4V worst-case voltage margin
- CMOS has better voltage-noise immunity (for same V_{DD}) and wider V_{DD} range. Ranges for 5V:
  - Input: 0-1.5V low, 3.5 to 5 high
  - Output: 0-0.05V low, 4.95 to 5 V high.

Logic families: the 74xx series

- Originally 74YYXX, where XX represents the code number for different functions.
- Series: YY
  - Nothing: original TTL, 10ns propagation delay, 25 MHz operation, 5V
  - L(S): Low power TTL: 33ns , 3MHz, 1/10 of power
  - H: high speed TTL, 6ns, 43MHz, 2.2x power
  - F: Fast TTL, 3.5ns, 100MHz, but 1/2x power!
  - HC(T): High speed CMOS, 9 ns, 50MHz, 1/20 of power. T has TTL compatible voltage levels
  - AC(T): Advanced CMOS, 3 ns, 125MHz, 1/20 of power, typically 3.3V or 5V

There are also other logic series like the 4000 CMOS series

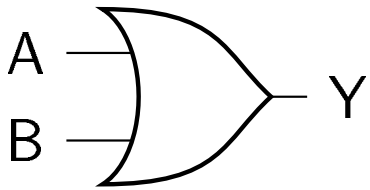
Other relevant logic codes

- ECL (Emitter-coupled-logic): ECL (negative power supply), PECL (positive supply) Less noise immunity, a lot of power required, but very very fast
- NIM logic (Nuclear Instrumentation Module): active-low!
- Many many CMOS variants with higher speed, lower power, lower V_{CC} etc.

Combinatoric logic, simple logic gates & logic simplification

Combinatorical logic is logic which only acts on the current state of signals. There is no history/memory.
(The delay between change of input and change of output is called the propagation delay)
We can build any function out of some standard building blocks!
Represent the function in a table. Short hand: we can represent the logic state TRUE/FALSE with a binary digit, 1=TRUE, 0=FALSE

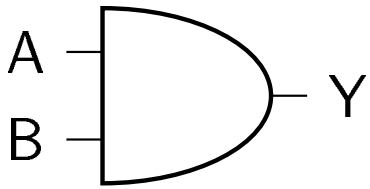
OR gate



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

- $Y = A + B$

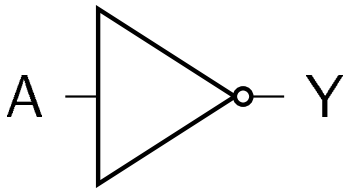
AND gate



A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

- $Y = A \cdot B = AB$

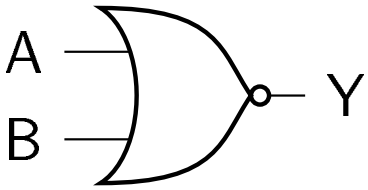
Inverter gate "NOT"



A	Y
0	1
1	0

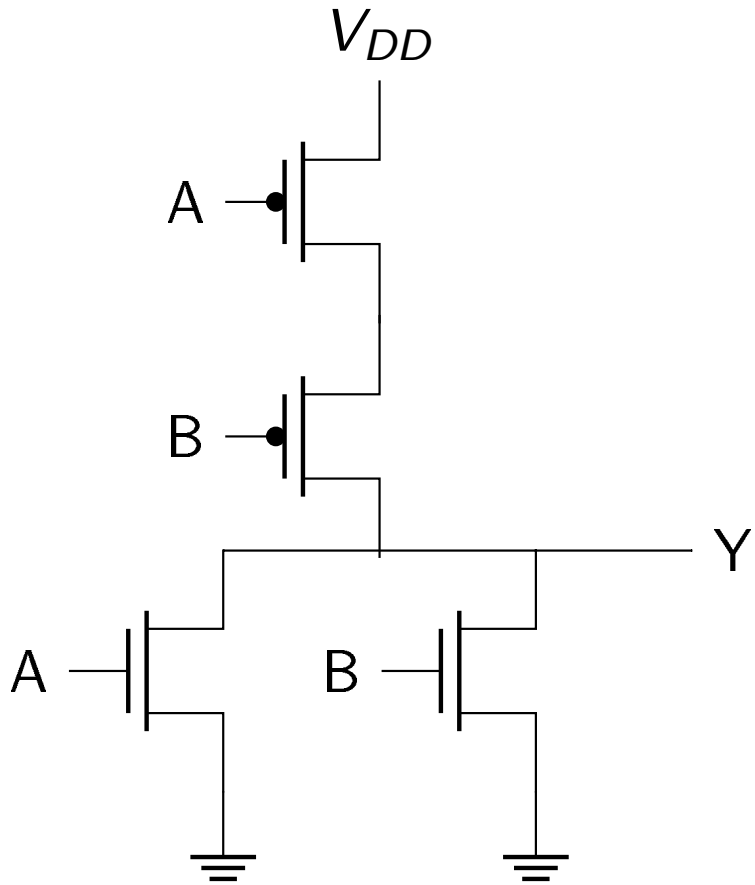
- $Y = \bar{A} = /A = *A = A' = -A = \sim A = !A$

NOR gate

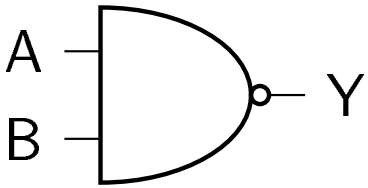


A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

- $Y = \overline{A + B}$

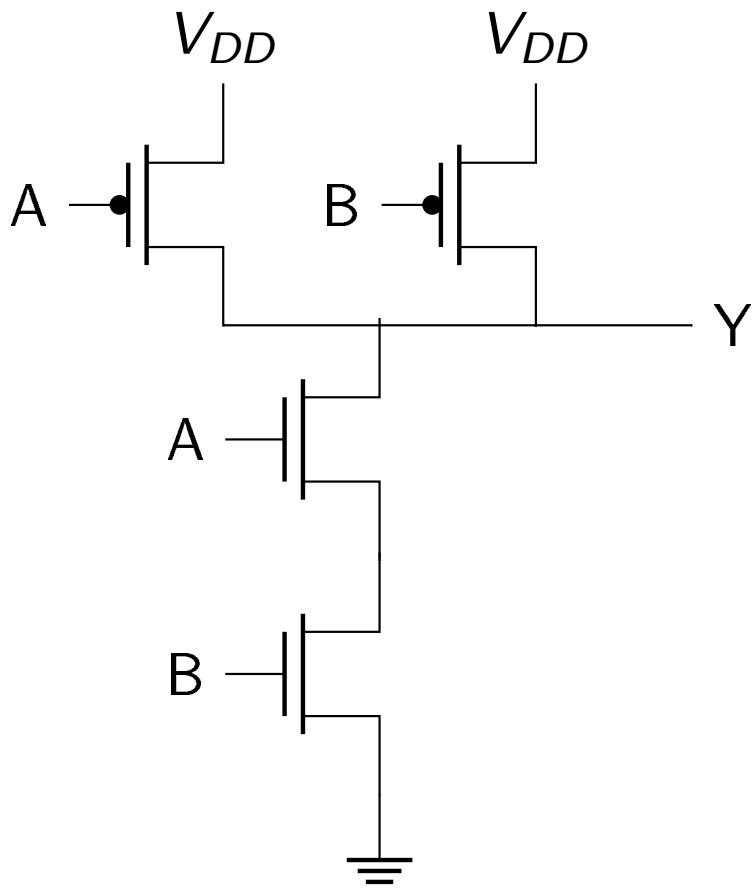


NAND gate

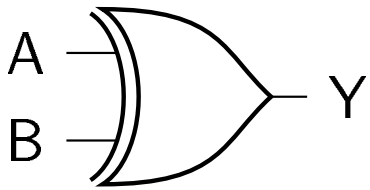


A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

- $Y = \overline{AB}$



XOR gate (exclusive or)



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

- $Y = A \otimes B = A \hat{} B$
- This is often used in cryptography:

$$A \otimes B \otimes B = A$$

De Morgan's Theorem

De Morgan's theorem:

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

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

De Morgan's Theorem

De Morgan's theorem:

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

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

This means that having inverters and one of OR / AND is enough to produce all logic functions!

De Morgan's Theorem

De Morgan's theorem:

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

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

This means that having inverters and one of OR / AND is enough to produce all logic functions!
Since a NAND or NOR with both inputs tied together is an inverter, one can build all logic functions just with NAND or NOR gates!

Logic Identities

- $ABC = (AB)C = A(BC)$
- $AB = BA$
- $AA = A$
- $A1 = A$
- $A0 = 0$
- $A(B + C) = AB + AC$
- $A + AB = A$
- $A + BC = (A + B)(A + C)$
- $A + B + C = (A + B) + C = A + (B + C)$
- $A + B = B + A$
- $A + A = A$
- $A + 1 = 1$
- $A + 0 = A$
- $\bar{1} = 0$
- $\bar{0} = 1$
- $A + \bar{A} = 1$
- $A\bar{A} = 0$
- $\bar{\bar{A}} = A$

From a truth table to a circuit

- For each line where the output is 1, write an AND of all input variables, negate them if the truth table is 0 for them.

From a truth table to a circuit

- For each line where the output is 1, write an AND of all input variables, negate them if the truth table is 0 for them.
- OR all terms.

From a truth table to a circuit

- For each line where the output is 1, write an AND of all input variables, negate them if the truth table is 0 for them.
- OR all terms.
- Simplify, with an eye on what gates you have, and what signals you already have in your circuit

Example:

A	B	C	Y
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	0
1	1	1	1

Example:

A	B	C	Y
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	0
1	1	1	1

$$Y = (\bar{A}\bar{B}C) + (\bar{A}B\bar{C}) + (\bar{A}BC) + (ABC)$$

Example:

A	B	C	Y
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	0
1	1	1	1

$$Y = (\bar{A}\bar{B}C) + (\bar{A}B\bar{C}) + (\bar{A}BC) + (ABC)$$
$$Y = \bar{A}(\bar{B}C + B\bar{C} + BC) + ABC$$

Example:

A	B	C	Y
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	0
1	1	1	1

$$Y = (\bar{A}\bar{B}C) + (\bar{A}B\bar{C}) + (\bar{A}BC) + (ABC)$$
$$Y = \bar{A}(\bar{B}C + B\bar{C} + BC) + ABC$$
$$Y = \bar{A}(\bar{B}C + B) + ABC$$

Example:

A	B	C	Y
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	0
1	1	1	1

$$Y = (\bar{A}\bar{B}C) + (\bar{A}B\bar{C}) + (\bar{A}BC) + (ABC)$$
$$Y = \bar{A}(\bar{B}C + B\bar{C} + BC) + ABC$$
$$Y = \bar{A}(\bar{B}C + B) + ABC$$
$$Y = \bar{A}(B + C) + ABC$$

Binary Numbers

- Represent integers in base 2:
  $6_{10} = 110_2 = b110 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

Binary Numbers

- Represent integers in base 2:
  $6_{10} = 110_2 = b110 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
- Each binary digit can be a signal
- An array of signal can represent an integer in a certain range

Binary Numbers

- Represent integers in base 2:
  $6_{10} = 110_2 = b110 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
- Each binary digit can be a signal
- An array of signal can represent an integer in a certain range
- Common names:
  - 1 bit: 1 signal
  - Nibble: 4 bit
  - Byte: 8 bit
  - Word (architecture dependent, but often): 16 bit
  - DWORD (double word) 32 bit
- Most significant bit (MSB): Highest valued bit
- Least significant bit (LSB): bit with value 1
- Often use hexadecimal: 1 hexadecimal digit (0-9,A-F) maps directly to a nibble. 0xFF=255=0b1111 1111

Examples

- 0xA=?

Examples

- 0xA=? =10

Examples

- 0xA=? =10=0b1010
- 0b1110=?

Examples

- 0xA=? =10=0b1010
- 0b1110=?=14

Examples

- 0xA=? =10=0b1010
- 0b1110=?=14=0xE
- 0b1111 1111 1111 1111= 0xffff=2^{16}-1=65535

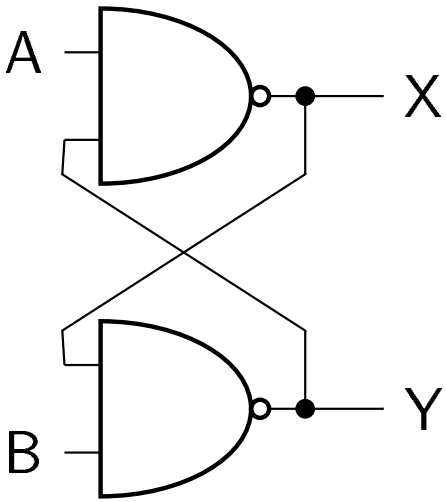
Negative numbers in binary

- Sign-magnitude: one bit for sign, remaining bits for magnitude
- Offset binary: 0b1000=0, 0b1001=1, 0b0111=-1
- 2's complement: 0b0111=7, 0b0000=0, 0b1111=-1,0b1000=-8

Often 2's complement is used often because it simplifies arithmetic and has no doubled zero.

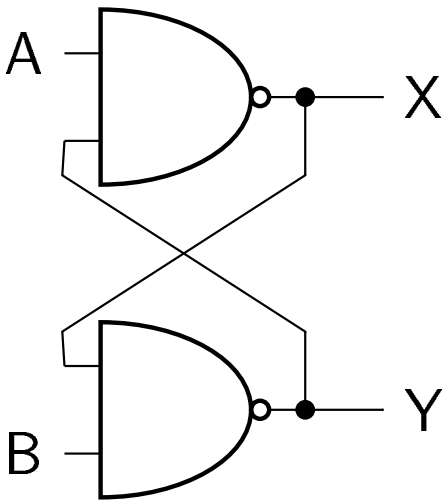
Sequential logic & one bit memory: flip-flops

The Reset-Set Flip-Flop: RS-FF



Sequential logic & one bit memory: flip-flops

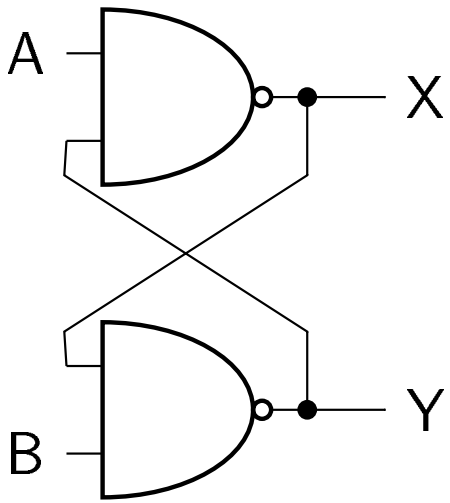
The Reset-Set Flip-Flop: RS-FF



- $X = \overline{AY} = \overline{A} + \overline{Y}$,
  $Y = \overline{BX} = \overline{B} + \overline{X}$

Sequential logic & one bit memory: flip-flops

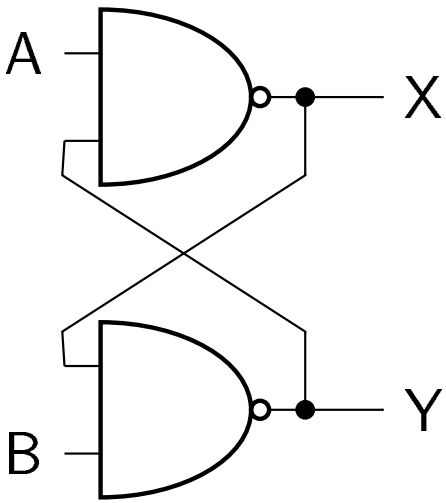
The Reset-Set Flip-Flop: RS-FF



- $X = \overline{AY} = \overline{A} + \overline{Y}$,
  $Y = \overline{BX} = \overline{B} + \overline{X}$
- Assume we put A low:
  - X must be high, Y depends on what B is.

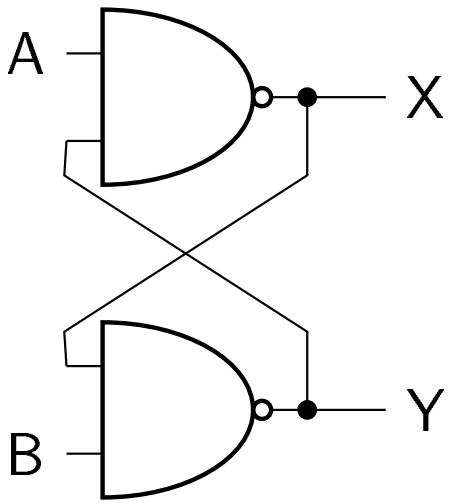
Sequential logic & one bit memory: flip-flops

The Reset-Set Flip-Flop: RS-FF



- $X = \overline{AY} = \overline{A} + \overline{Y}$,
  $Y = \overline{BX} = \overline{B} + \overline{X}$
- Assume we put A low:
  - X must be high, Y depends on what B is.
- Assume we put B low:
  - Y must be high, X depends on what A is.

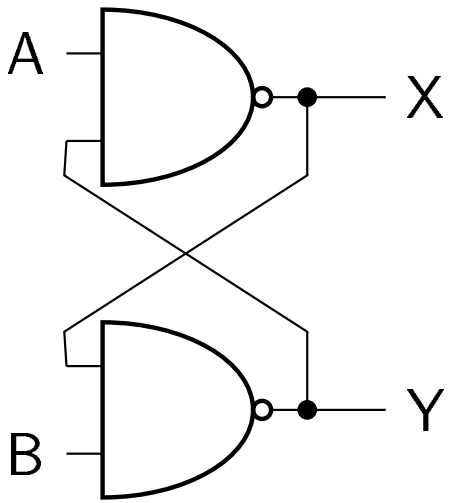
The Reset-Set Flip-Flop: RS-FF



- $X = \overline{AY} = \overline{A} + \overline{Y}$,
  $Y = \overline{BX} = \overline{B} + \overline{X}$
- Assume we put A and B HIGH. Two possibilities:
  - X is high. This means Y has to be LOW, which is fine, because both B and X are high
  - X is low. This means that Y has to be HIGH.
  - X and Y can not both be LOW (or HIGH) at the same time!

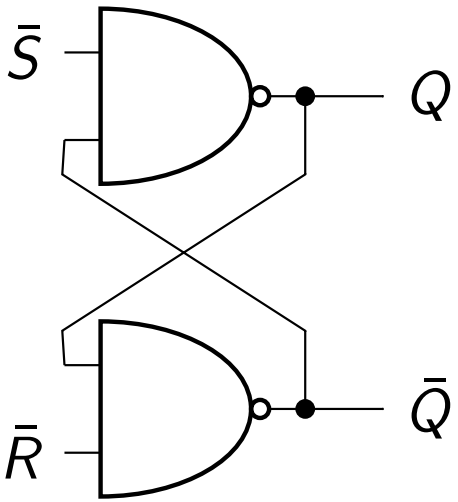
Sequential logic & one bit memory: flip-flops

The Reset-Set Flip-Flop: RS-FF



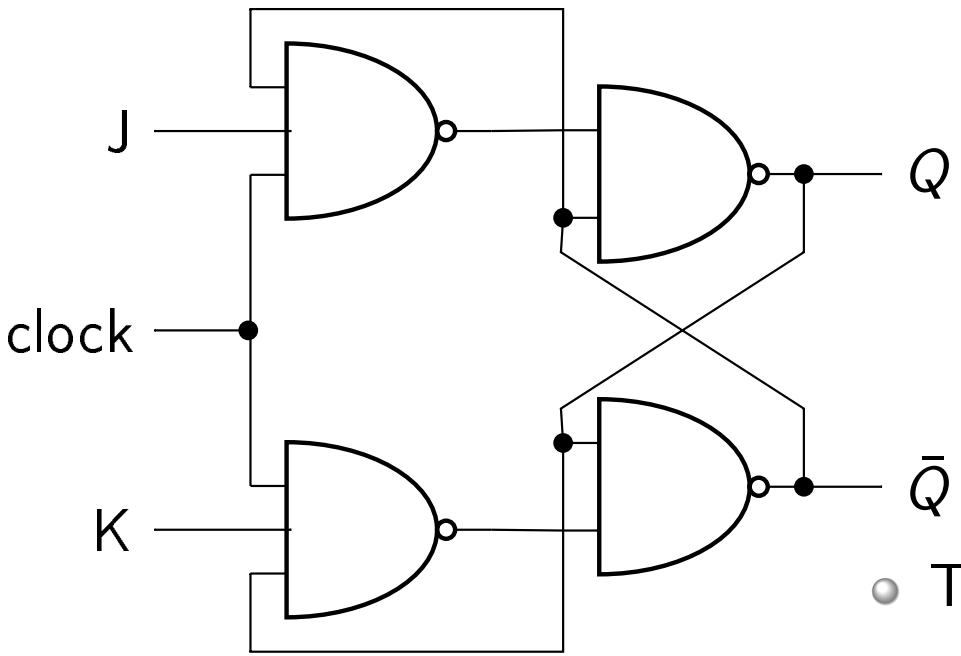
- $X = \overline{AY} = \overline{A} + \overline{Y}$,
  $Y = \overline{BX} = \overline{B} + \overline{X}$
- Assume we put A and B HIGH. Two possibilities:
  - X is high. This means Y has to be LOW, which is fine, because both B and X are high
  - X is low. This means that Y has to be HIGH.
  - X and Y can not both be LOW (or HIGH) at the same time!
  - So the circuit is bi-stable. It depends on the history whether X is HIGH or LOW

RS flip flop function summarized



- Normal state: $\bar{S}$ and $\bar{R}$ is high. Q and $\bar{Q}$ stay constant.
- Setting $\bar{S}$ low sets the FF, so Q goes high
- Setting $\bar{R}$ low reset the FF, so Q goes low
- Setting $\bar{S}$ and $\bar{R}$ low at the same time is a forbidden state (Because then $Q = \bar{Q}$)

Clocked flip flops: JK



- If clock is low, nothing happens
- if clock is high,
  - J and K are low: nothing happens
  - J high: If Q is low, S goes low, Q goes high, S goes high, then nothing happens
  - K high: If Q is high, Q goes low, then nothing happens
  - J and K are high: Q toggles/oscillate
- Two fixes:
  - Short clock high periods (e.g. with a capacitor)
  - Master-slave config (two JK in series, with inverted clock)

Most common flip flop: D FF

- Take a MS JK flip flop. Rename J to D, and connect K to an inverted D
- D is the data line. On every clock edge, D is transferred to Q. This is called latching.
- Exist with "latching" on rising and/or falling edge.

Oscillators and clocks

- We already talked about the NE555.
- Many variants. Parameters to look out for:
  - Frequency stability
  - High-Low-ratio
  - Jitter: short term frequency oscillations
  - Can it drive all my chips?
- Often combined with a quartz, which swings on its resonance frequency

Adders: Half adder

We want to add two 1 bit numbers to get one 2 bit output

A	B	C(arry)	L(SB)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Adders: Half adder

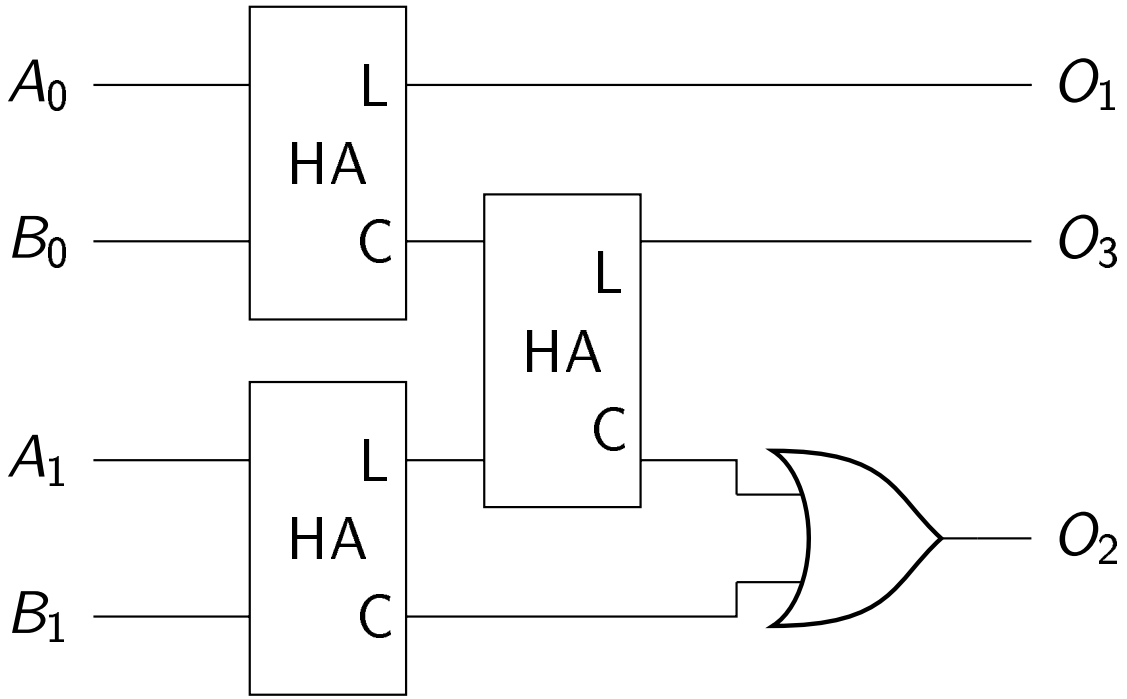
We want to add two 1 bit numbers to get one 2 bit output

A	B	C(arry)	L(SB)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- $C = AB$
- $L = A \otimes B$ (xor)

Adding more bits with Half Adders

Two two-bits to 3 bits:



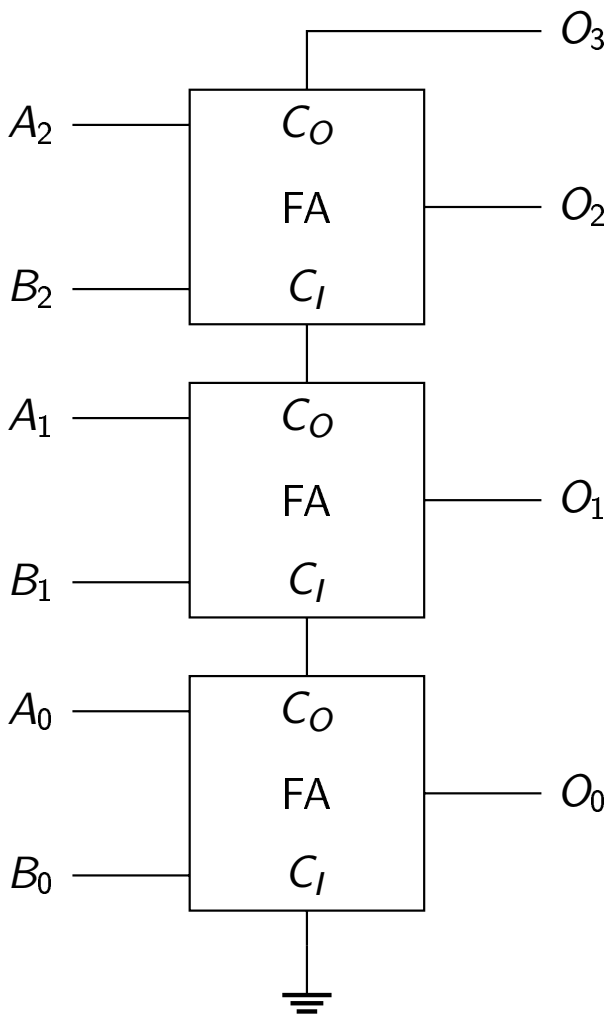
Exercise: Build the truth table for this circuit!

Full adder

Truth table:

A	B	C_i	C_o	L
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

Multiple bits with the Full adder



- Slowest path is carry, which has to ripple through all FAs.
  Other name: Ripple carry adder RCA

Programmable logic: FPGAs

In addition to fixed function ICs, there exists programmable logic. The most capable version of this are called FPGA: Field Programmable Gate Arrays

- Consists of many LUTs, look-up-tables. These are the hardware realization of truth tables.
- One can program these truth tables!
- Additionally, one can program how these elements are connected to each other
- This is not the same as a CPU

From analog to digital: Analog to Digital Converters

We want to convert an analog signal into a binary number proportional to the size of the signal.

From analog to digital: Analog to Digital Converters

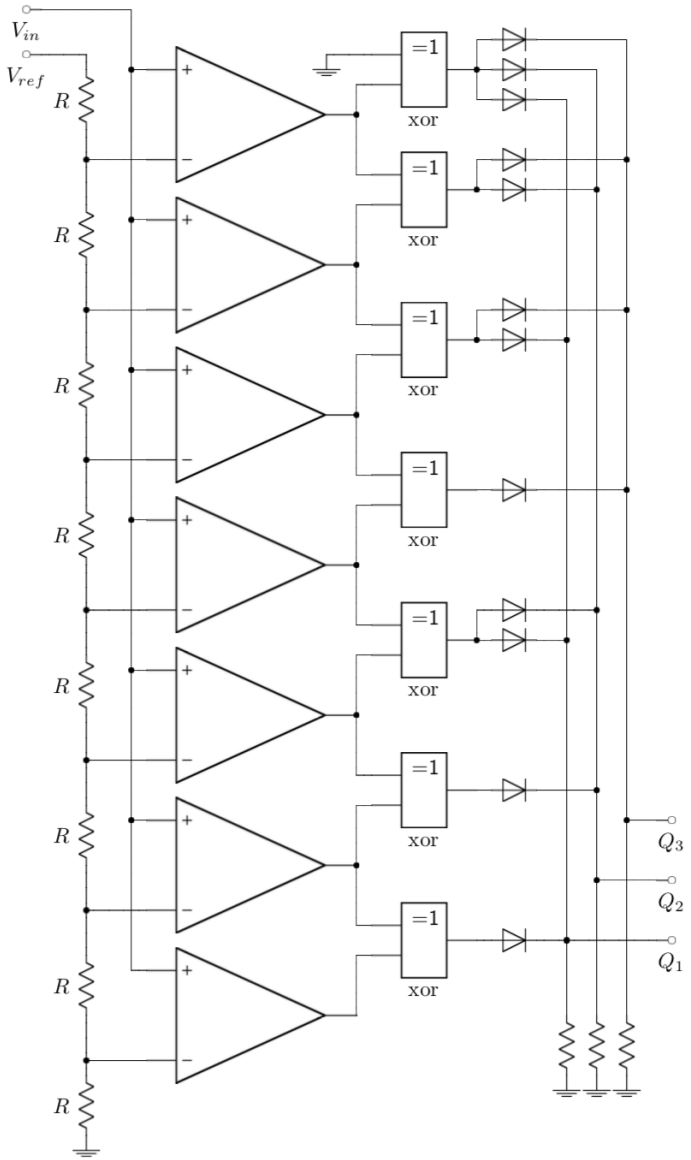
We want to convert an analog signal into a binary number proportional to the size of the signal.
We already now a simple ADC with 1 bit output! The discriminator!

From analog to digital: Analog to Digital Converters

We want to convert an analog signal into a binary number proportional to the size of the signal.
We already now a simple ADC with 1 bit output! The discriminator!
There are many different ways to built multi-bit ADCs:

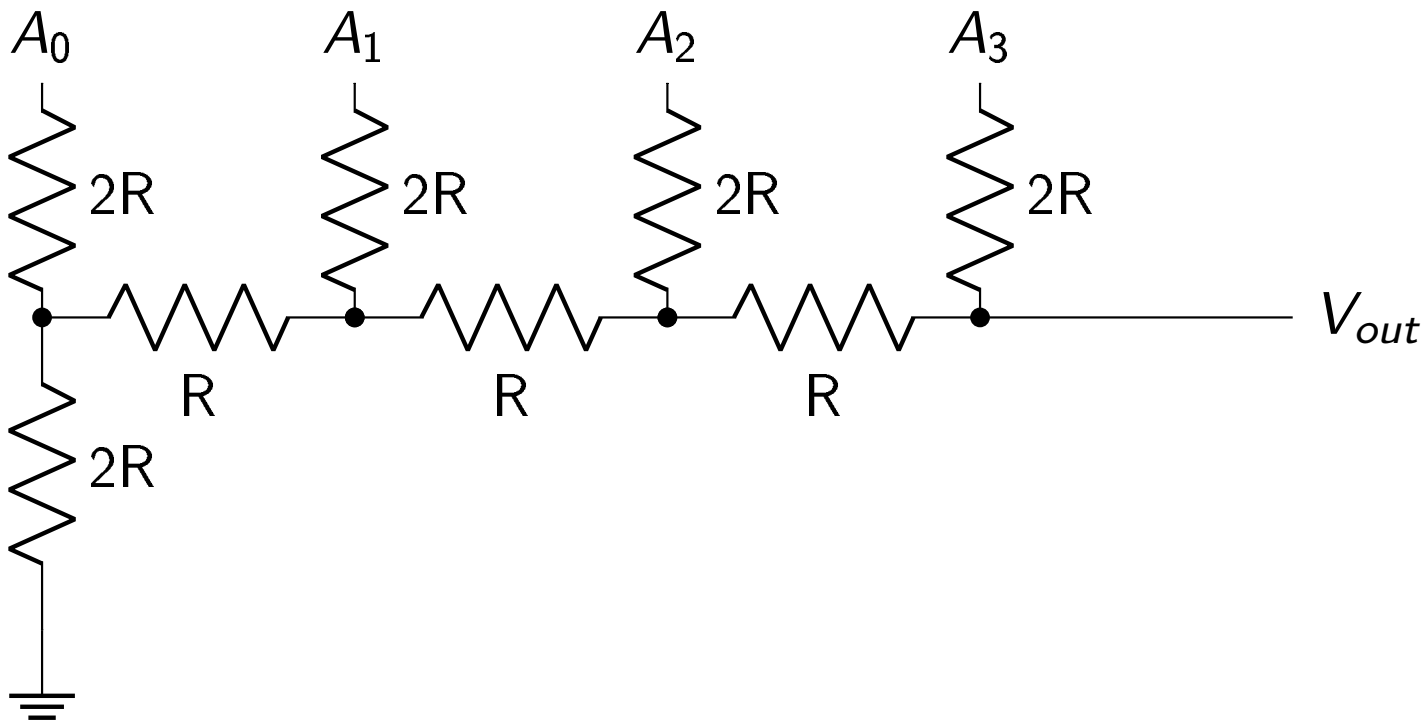
- Flash or direct conversion ADCs: Result in one clock
- Successive approximation: Result in N clocks for n bits
- $\Sigma\Delta$ ADCs: 1 bit with high oversampling. Slowest, but very linear

Flash ADC



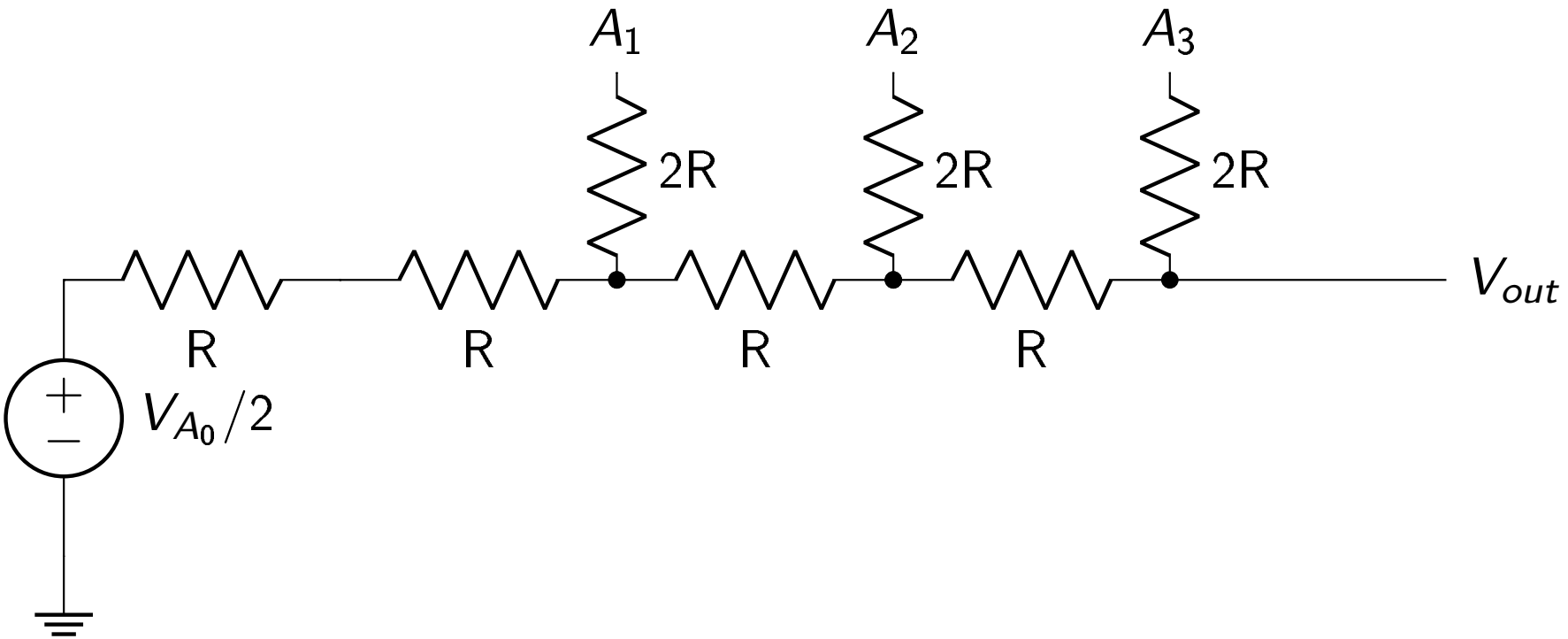
Digital Analog Converts: DACs

Simplest form: R-2R network:



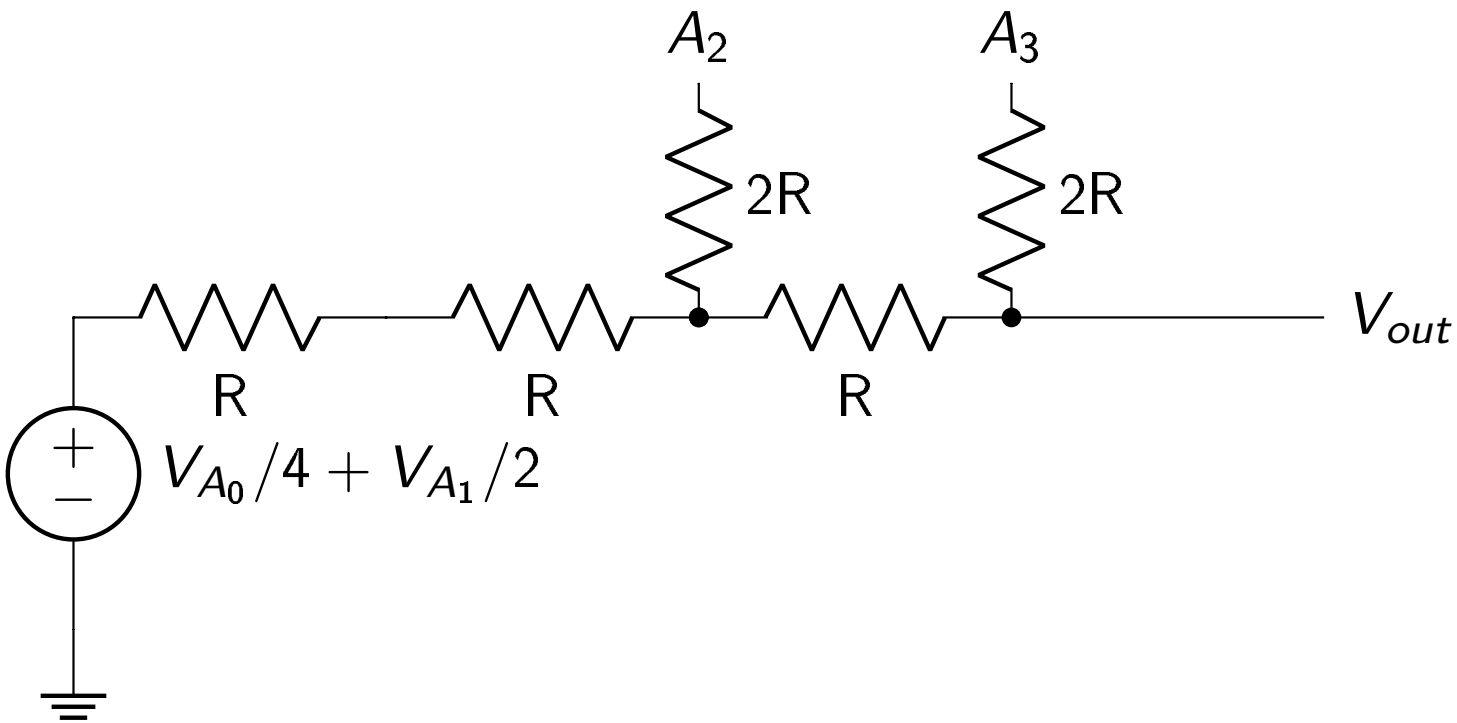
Digital Analog Converts: DACs

Simplest form: R-2R network:



Digital Analog Converts: DACs

Simplest form: R-2R network:



Digital Analog Converts: DACs

Simplest form: R-2R network: