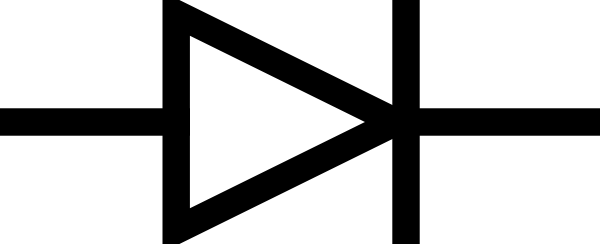
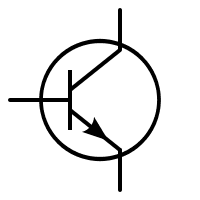
Numbers

* Numbers systems
  + Place values work like this: rightmost is base0, second rightmost is base1, third rightmost is base2 and so on
  + Converting to decimal (base 10)
    - Align the digits to correct place, and multiply the digit by its place value
    - Add the products
  + Converting from decimal
    - Keep dividing the number by destination base until the quotient is less than 1
    - Record the integer part of the quotient, storing the first quotient on the right, second quotient left of the first, and so on
    - Take the destination base modulo of these quotients, and write directly underneath the corresponding quotient
  + Number of possible combinations: basedigits
    - Tip: an increase by 10bits is about 1000x increase in possible combinations
  + When adding or subtracting, use the base as the number used for carrying over when necessary
  + Binary number system (base 2)
    - 0 and 1 are the digits used
    - Bits: **b**inary dig**it**
    - A group of 3 bits can represent an octal digit
    - A group of 4 bits can represent a hexadecimal digit
    - Signed bits
      * Most significant bit becomes a signed bit
      * Meaning, when it’s 0, it has positive value
      * And when it’s 1, it has a negative value
      * Ex: 1001
      * -8 4 2 1
      * 1(-8)+1(1)
    - To add signed binaries, fix the bit width, then add and carry over when 2 is reached instead of 10. Any bit that flows past the most significant fixed bit is dropped (ex: if width is set to 4, ex: 1111 and 1 is added, instead of 10000, drop the 1 so it becomes 0000)
    - To subtract, add the 2s complement (aka the negative)
    - 2s complement
      * First, make bit width fixed
      * Then, turn all 1s into 0s, vice versa
      * Then, add 1, and drop overflow bit if needed
  + Octal number system (base 8)
    - 0, 1, 2, 3, 4, 5, 6, 7 are the digits used
    - 3 octal digits represent 1 bit
  + Decimal number system: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 are the digits used
  + Hexadecimal number system (base 16)
    - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A (means 10), B (means 11), C (means 12), D (means 13), E (means 14), F (means 15) are the digits used
    - 4 hexadecimal digits represent 1 bit
* Notations
  + Scientific notation
    - A \* 10n, where A is a number [1, 10) and n is number of places to move the decimal (positive: to right, negative: to left)
    - add/subtract: Make the power of 10 terms have the same exponent, then add/subtract the non-power of 10 terms. Finally, readjust if necessary
    - Multiply/divide: Multiply/divide the non-power of 10 terms, then multiply/divide the power of 10 terms. Finally, readjust if necessary
  + Engineering notation
    - Like scientific notation, but A is [0, 999]
    - The exponent of the power of 10 is a multiple of 3
    - Makes converting with SI notations easier
  + SI notation
    - T replaces 1012
    - G replaces 109
    - M replaces 106
    - k replaces 103
    - m replaces 10-3
    - μ replaces 10-6
    - n replaces 10-9
    - p replaces 10-12
  + Symbols
    - l - current - ampere
    - V - voltage - volt
    - R - Resistance - Ω
    - f - Frequency - hertz
    - C - capacitance - farad
    - L - inductance - henry
    - P - power - Watt

Components

* Analog: smooth, infinite, support in-between
  + Resistors: resists the flow of electrical current, analog
    - 4 or 5 band carbon film resistors
      * The color bands represent its resistance nominal value in Ω
        + 3rd band is the multiplier for the number represented by the first 2 bands, 4th band is tolerance
    - Surface-mount resistors
    - Variable resistors (potentiometer)
  + Capacitors: stores electrical charge, analog
    - Ceramic disc capacitors
      * The third digit represents the number of 0’s to add after the first 2 digits. The last digit represents tolerance of the capacitance. Units: pF
    - mylar/tantalum monolithic ceramic
    - Surface mount ceramic capacitors
    - Electrolytic capacitors 
      * polarized
      * Capacitance labeled, and tolerance is +/- 20% unless specified otherwise. Units: μF
    - Surface mount tantalum capacitors
    - Charging rate:
    - Discharging rate:
    - Where V is voltage, R is resistance of the resistor dis/charging through and C is the capacitance of the capacitor
  + Fuses: a strip of wire that melts and break if the current exceeds a safe level
  + Diodes: semiconductor with 2 terminals that only allows current to flow one direction, analog
    - To light one up, need a power source connected to a resistor connected to the anode end of the diode, and cathode end connected to a ground
    - Anode has to be at least 1.5V greater than cathode
    - LED
    - 7-segment display
      * Made up of 7 LED
      * A-G, going clockwise from top-most one. The middle line is G.
      * Common cathode: the LEDs share the same pin as cathode connection to ground. Turns on when logic 1
      * Common anode: the LEDs share the same pin as anode connection to power. Turns on when logic 0
    - Alphanumeric display
    - By packaging a red, green and blue LED together, their brightness can be altered to display different colors
  + Voltage regulator: maintains voltage within acceptable limits
* Digital: discrete, finite values, binaries
  + Transistor: controls the flow of electricity using at least 3 electrodes - digital 
    - Amplifier
    - Switch
  + Logic gates (series 74)
    - Arranged of transistors and resistors
    - NOT: outputs the opposite of the input 
    - AND: outputs 1 only if all inputs are 1
    - OR: outputs 1 as long as at least one input is 1
    - NOR: opposite of OR
    - NAND: opposite of AND
    - XOR: outputs 1 when an odd number of 1’s are inputted
    - XNOR: opposite of XOR
  + Integrated circuits: contains transistors, diodes, resistors, capacitors, gate
    - Technology
      * Transistor-transistor logic
        + Made from bipolar junction transistors
        + Faster than CMOS
        + Not sensitive to electrostatic damage
        + Uses more power than CMOS
        + Subfamilies

None: original, obsolete, slowest, takes the most power

L: low power: consumes less power than above (obsolete)

S: Schottky: above optimized for speed, but takes more power (obsolete)

LS: Low-power Schottky: faster and lower power consumption than above

AS: advanced Shottky: very fast and power-consuming

ALS: advanced low-power Schottky: great speed:power ratio

* + - * Complementary metal oxide semiconductor
        + Made from metal oxide semiconductor field-effect transistors
        + Uses less power than TTL
        + Slower than TTL
        + Sensitive to electrostatic damage
    - Scale of integration
      * Small scale integration
        + <10 gates
        + For logic gates
      * Medium-scale
        + 10-100 gates
        + For flip flops, counters, multiplexers
      * Large scale integration
        + 100-10000 gates
        + For small memory chips, programmable logic devices
      * Very large scale integration
        + 10000-100000 gates
        + For large memory chips, complex programmable logic devices
      * Ultra-large scale integration
        + 100000-1000000 gates
        + For 8, 16-bit processors
      * Giga-scale integration
        + >1000000 gates
        + For computer processors
      * The larger the scale the higher the level of abstraction, and less control over what happens in the “magic box”
    - Packaging style
      * Through-hole technology
        + Uses pins inserted into holes of a circuit board and soldered into place
        + Easy to hand assemble
        + Takes up more space
        + Dual inline package (N)

Numbering: start by orienting dot to lower left. The lower left is pin 1. Count increases counterclockwise

Upper left most pin connects to the power source

Lower right most pin connects to ground

Obsolete, only used in educational settings

* + - * Surface mount technology
        + Takes up less space
        + Has more pins
        + Can be mounted on either side of a board
        + More expensive
        + More complex
        + Small outline IC (B)
        + Quad flat pack (F)
        + Plastic lead chip carrier (C)
* Breadboards can be used to test a design
  + Busses are connected together
  + A column with the same number is connected together
* Troubleshoot all components before starting

Electricity

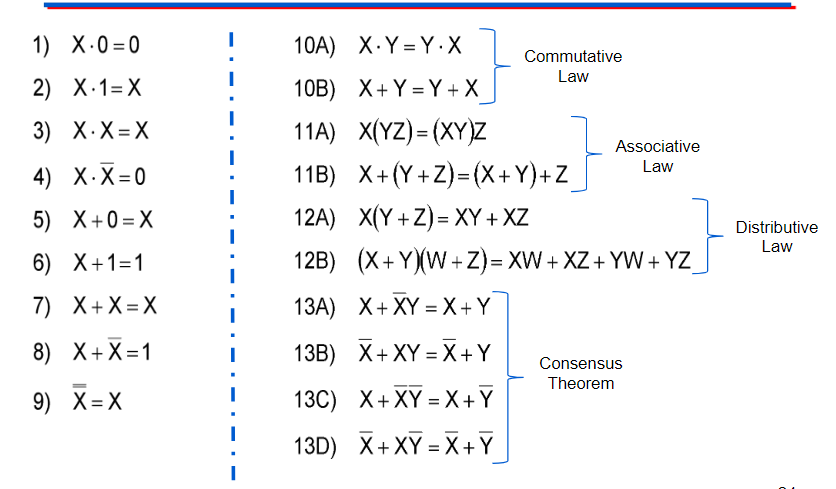
* Current (A, amps): the flow of electrical charge
  + Electron flow: flow from anode to cathode
  + Conventional current: direction opposite of electron flow
  + Can be measured using a multimeter by putting 2 leads before the component, remove the wire in between the leads, to make sure current flows through multimeter like it’s a component
* Voltage (V, volts): the force that causes current to flow
  + Can be measured using a multimeter by putting one lead before the component and another lead after the component
* Resistance (Ω, ohms): opposition to current
  + Current flows where there is less resistance
  + Can be measured by replacing power source with a multimeter
* Ohm’s Law: I=V/R, where I is current, V is voltage and R is resistance
* Series circuits
  + Components are connected end to end, with only one path for current
  + Total resistance is the sum of all resistor’s resistance
  + Voltage drops after passing through each component
  + Current stay the same
  + Kirchhoff's Voltage Law: sum of voltage drops in a loop equals voltage gain from battery
* Parallel circuits
  + Multiple pathways for current flow
  + Total resistance is the reciprocal of the sums of the reciprocal of all resistor’s resistances
  + Each branch receives the same voltage, but not current
  + Kirchhoff’s Current Law: sum of current going into a junction equals sum of current leaving the junction
* Combined circuits: it may help to temporarily combine multiple resistors based on series/parallel characteristics

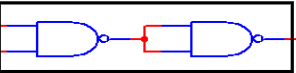
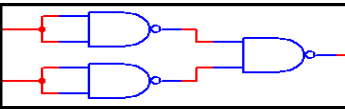
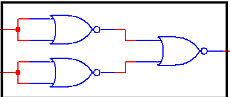
Data Sheets

* TTL Part number
  + First 2 letters is manufacture
  + Next 2 is series
  + Next is TTL logic subfamily infix
  + Next 2 numbers is logic function
  + Next letter is packaging style
* Manufacturers
  + DM: Fairchild
  + SN: Texas Instrument
* Contains: general description, diagram, function table, operating conditions, electrical and switching characteristics, physical dimensions

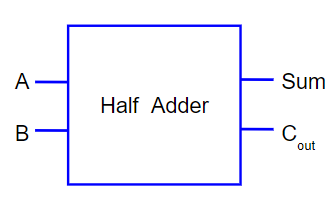
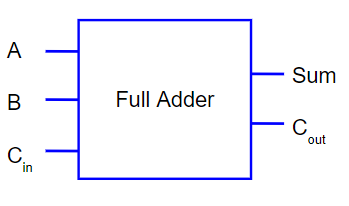
Digital Logic

* Logic expressions
  + output = input [logic symbol] input …
  + AND uses multiplication as the logic symbol
  + OR uses addition as the logic symbol
  + NOR uses x in a circle as the logic symbol
  + NOT uses a not or an apostrophe as the logic symbol
  + Sum of products
    - Indicates which combinations result in an output of 1
    - Minterms: inputs binded by AND that results in an output of 1
    - In a SOP expression, bind all the minterms by OR
  + Product of sums
    - Indicates which combinations result in an output of 0
    - Maxterms
      * Zeros are considered true and ones are considered false
      * Inputs binded by OR that results in an output of 0
    - In a POS expression, bind all maxterms by AND
  + Simplified
    - If a letter is omitted in a term, then that means a wildcard (any truth value) in its place
* Truth table
  + A design has 2n possible input combinations, where n is the number of inputs
  + On input 1’s column, top half is 0s, the bottom half is 1s. In the next column, for the height of the previous column’s 0s, the top half is 0s, and bottom half is 1s. For the height of previous column’s 1s, the top half is 0s, and bottom half is 1s. Repeat recursively for as many inputs as present. The last column is the output.
* Excitation table: like a truth table, but also shows rising and falling transitions
* Circuit design process: review specs > determining inputs/outputs > truth table > SOP/POS expressions > simplified expression > determining gates required > build and test
* Going from circuit to expression
  + Test points method
    - Number each gate
    - Generate a truth table with all possible input combos
    - Create a column on the truth table for each labeled gate
    - For each combo set, write the output coming out of each gate in the corresponding cell in the table (work one column at a time)
    - Write SOP or POS for the last gate
  + Expressions method
    - In terms of the inputs, for each gate, write an expression to represent its output.
    - Work your way to the last gate for final expression
* Simplifying SOP/POS equations-Boolean algebra
  + Check the theorems on the reference sheet of how to combine things



* Simplifying equations-Karnaugh Mapping
  + Divide the inputs amongst row and column label. If there’s an odd number of inputs, row label gets the extra. Ex: XYZ: XY is row, Z is column
  + Start left most of column labels with all inputs NOT’ed, and top most of row labels with all inputs NOT’ed
  + Left to right for column labels, top to bottom for row labels, change combos one bit at a time to progress towards all possible combos listed. Change the least significant bit possible without duplicating any of the combos already listed. Ex: X’Y’ > X’Y> XY > XY’
  + Each cell represents the input combo resulted from AND’ing the cell’s row and column name. Ex: cell of row XY’ and column Z represents combo XY’Z, or 101. Populate the table with output 0 or 1 based on that
  + “Don’t care” values: denoted with “X”, and can be treated as a 1 or 0
  + Circle groups of 1’s. Rule:
    - number of 1’s in a group has to be a power of 2 (1, 2, 4, 8, 16…)
    - Shapes can only be linear or box. No T or L
    - Groups can wrap around the table
    - Overlaps are allowed
    - Goal: least amount of the largest possible sized groups
  + The terms in common within a group represents a minterm. Ex: XY and YZ’ have Y in common, so Y is a minterm
  + OR together the minterms for simplified expression
* NAND
  + Universal gate: can replace AND, OR, NOT in a SOP circuit so that only one type of chip would be needed
  + DeMorgan’s theorem: if multiple AND’ed or OR’ed terms are all under the same not bar, break the bar at the ANDs and ORs. Then, change ANDs to ORs, and vice versa
  + Replace AND with 
  + Replace OR with 
  + Replace NOT with 
  + 2 consecutive NOTs cancel out
* NOR
  + Universal gate: can be used to replace AND, OR, NOT in a POS circuit
  + Use NOR, NAND, AOI combo so that the longest path of the circuit has the least possible number of gates to reduce delay
  + Replace AND with 
  + Replace OR with 
  + Replace NOT with 
  + 2 consecutive NOTs cancel out
* XOR: X’Y+XY’ becomes X (xor) Y
* XNOR: X’Y’ + XY becomes NOT(X (xor) Y)

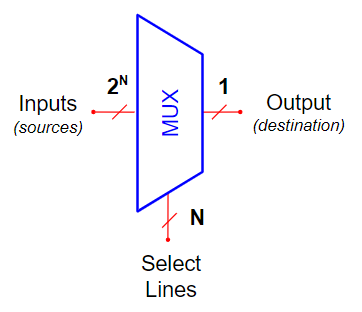
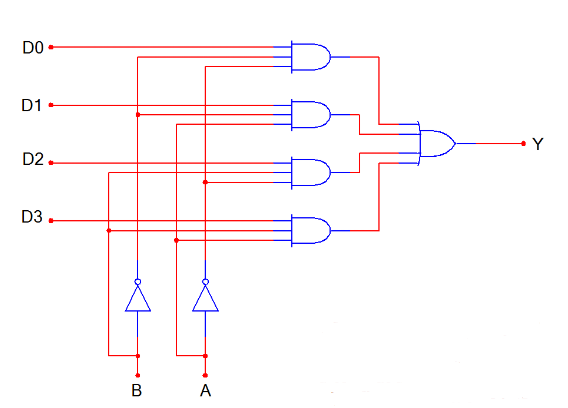
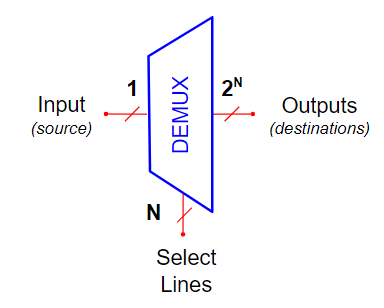
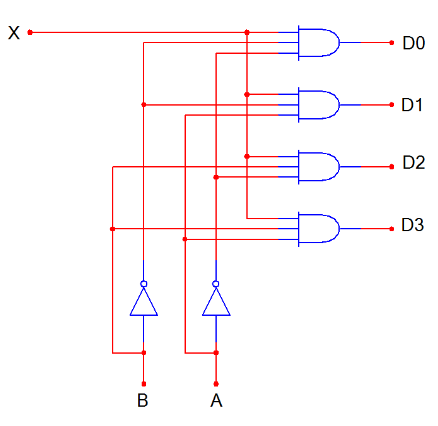
Adders

* Addition is basically adding the numbers instructed to add, then add the number carried over from the previous place value. Results in a sum, and a value that’s carried over to the next place value
* Half adder
  + Used only for LSB
  + Sums 2 inputted bits, and outputs a sum and a value to carry over the next place value
  + MSI: 
  + SSI: K-Map the truth table
    - Sum is
      * XOR outputs 1 only when the number of inputs that are 1 is odd, which is like adding binary: the place value is only 1 when the number of 1s being added in the place value is odd
    - Carry out is AB
      * AND only outputs 1 when both inputs are 1, which is like adding binary: adding 2 1’s would result in a carryover
* Full adder
  + Sums 3 inputted bits: the 2 being added + the carryover from the previous place value. Outputs a sum and carry out just like half adders
  + MSI: 
  + SSI: K-Map the truth table
    - Sum is
      * Same reason as half adder
    - Carry out is AB+ACIN+BCIN
      * Same reason as half adder
* Ripple adder
  + made of multiple full adders. One for each place value
  + To link full adders, connect A and B to the input for that place value, Sum to the output of that place value, Cout to the Cin of the next adder

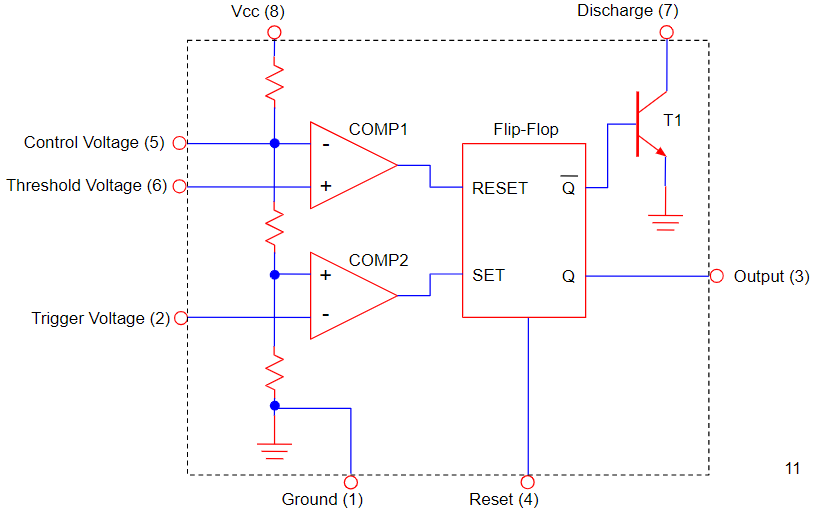
Waveforms

* A graph, where x axis is time and the y axis is voltage level
* Digital waveforms
  + Square waves
  + Flat horizontal lines to represent the voltage staying the same
  + The lines at the higher voltage level is logic 1, and the lines at the lower voltage level is logic 0
  + A slightly slanted line that basically looks straight vertical indicate a change between logic 1 and logic 0
  + No negative voltage levels
* Oscilloscope
  + Generate waveforms
  + Use a 4 channel one for digital
  + Use a 2 channel one for analog
  + Adjust time interval spacing
  + Shift channels up to seperate them
* Amplitude: difference between max and min voltage
* Period: time in sec for a repetitive signal to repeat
* Frequency: number of occurrences of a repeated signal in a second, in Hz
  + reciprocal of period
* Time high: amount of time at logic 1 within a period
* Time low: amount of time at logic 0 within a period
* Duty cycle: ratio of time high to period
* Rising edge: a transition from logic 0 to logic 1
* Falling edge: a transition from logic 1 to logic 0
* Clock voltage: cycles between 0 and 1 output based on its configured frequency, voltage and duty cycle
* Pulse Width Modulation
  + Encodes an analog “message” using a digital signal by varying duty cycle
  + Noncontinuous rotation servos: interprets the signal as angular displacement
  + Continuous rotation servos: interprets the signal as angular velocity
    - 90: stop
    - <90: forward
    - >90: reverse

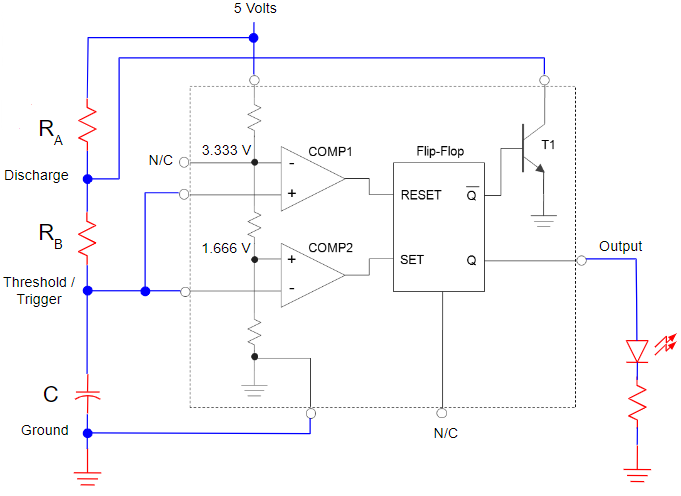
De/Multiplexers

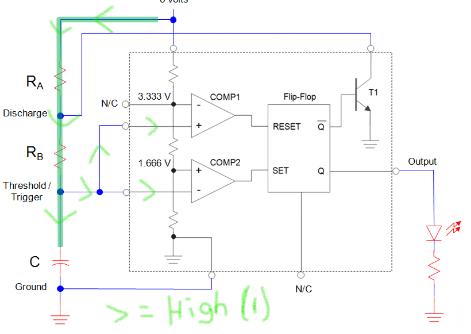
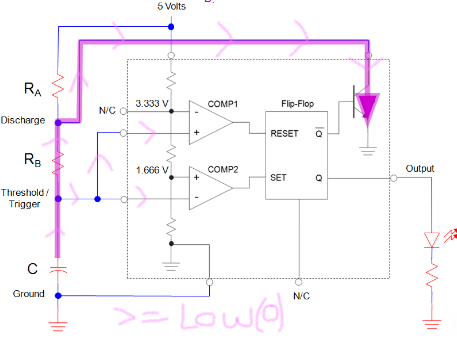
* Multiplexer (MUX)
  + has multiple inputs but only 1 output
  + Support 2ninputs, where n is number of select lines
  + MSI
  + Select lines determine which input to pass as output
  + SSI
  + The correct select conditions need to be met for the input signal to pass the AND gate then OR to output
* Demultiplexer (DEMUX)
  + Takes an input and shares to one of the multiple outputs
  + Support 2n outputs, where n is number of select lines
  + MSI
  + SSI 
  + The correct select conditions need to be met for the input signal to get out an AND gate

555 Timers



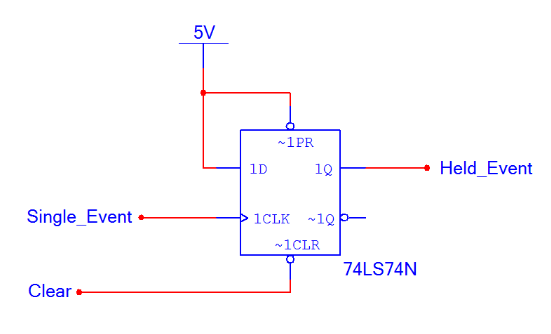
* Monostable
  + Creates a delay
  + Requires a resistor and a capacitor
* Astable



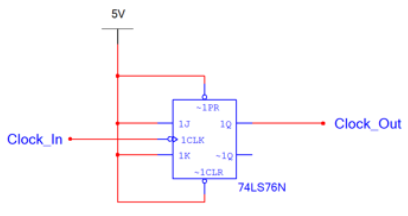
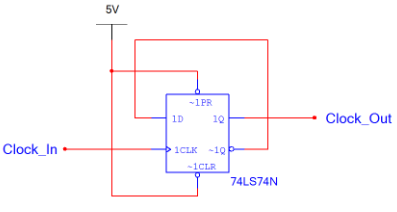
* + Creates an oscillation
  + Requires a capacitor and 2 resistors
  + Connect one side of resistor RA to Vcc and the other side to resistor RB and Discharge
  + Connect one side of resistor RB to resistor RA and Discharge. The other side connects to capacitor C, Threshold and Trigger Voltage
  + Connect one side of capacitor C to resistor RB, Trigger and Threshold Voltage. The other side to Ground
  + Output outputs a clock voltage
  + Time high
    - Is when the capacitor is charging
    - 
    - 0.693(RA+RB)C
  + Time low
    - Is when capacitor is discharging
    - 
    - 0.693RBC
  + Using the expressions for time high and low, the duty cycle can be calculated as , and period can be calculated as 0.693(RA+2RB)C
  + Decreasing RAand/or RB increases frequency because dis/charging through less resistance means dis/charge happens more quickly (refer to dis/charging rate equations), so more periods can be done
  + Decreasing RA decreases duty cycle, as charging occurs through RA+RB, so charging rate increased (take less time), but discharging occurs through RB and discharging rate didn’t increase. The 555 Timer would spend more time receiving a 0 from discharging, and less time receiving a 1
  + Decreasing RB increases duty cycle, as discharging occurs through RB, so less resistance would make discharging faster (less time receiving a 0). Charging (through RA+RB) also increases, but not as much, as RA doesn’t change, so overall, more time spent on receiving a 1
  + Decreasing C increases frequency, as less capacity to charge means dis/charge rate increases and can perform more periods

Flip-Flops

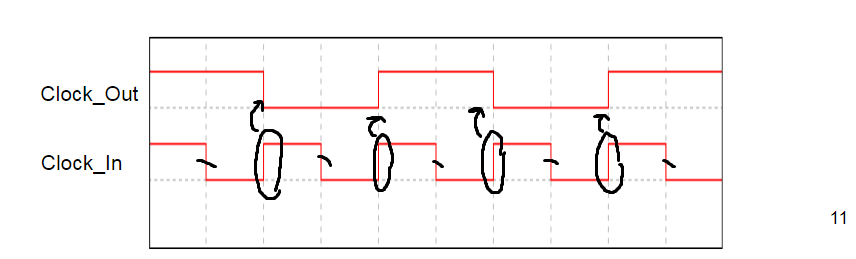
* Note that a circle and a ~ means inverted/active low input
* D Latch
  + Q mirrors D when EN is 1
  + Q “locks” its state when EN is 0
  + ~Q is the opposite of Q
  + EN may have inversion
* D Flip Flop
  + Q is output
  + ~Q is inverted Q
  + Clock for triggering (may have inversion)
  + CLK is indicated by a triangle, defaulted to triggers on the rising edge. When there is a circle with the triangle, it triggers on the falling edge.
  + Asynchronous
    - Doesn’t update Q based on Clock
    - ~Preset overrides Q to 1 when activated, until next Clock trigger
    - ~Reset overrides Q to 0 when activated, until next Clock trigger
    - “~” In Preset/Set or Clear/Reset indicates “active low” - it is active and overrides/interrupts Q when set to 0. Without this symbol, it is at an “active high,” overriding Q when set to 1.
  + Synchronous
    - Updates Q based on Clock
    - D is input
    - Q updates to match D during Clock’s rising edge, and locks until next rising edge
* JK Flip Flop
  + Same thing as a D Flip Flop, but instead of having D, there’s J and K
  + On Clock’s rising edge, if
  + J has a different value than K, Q updates to J
  + J and K are both 1, Q updates to ~Q
  + J and K are both 0, Q doesn’t update its value
* Applications
  + Event detector
    - “Holds on” to a signal until circuit is reset



* + - ~Preset and D are connected to 1. When the Clock is triggered on an edge, the 1 from D would be sent out to Q, until ~Clear receives a 0 to cause the ~1 (0) from ~Preset to be sent out Q
  + Divide by 2
    - Divides the frequency of a circuit by 2

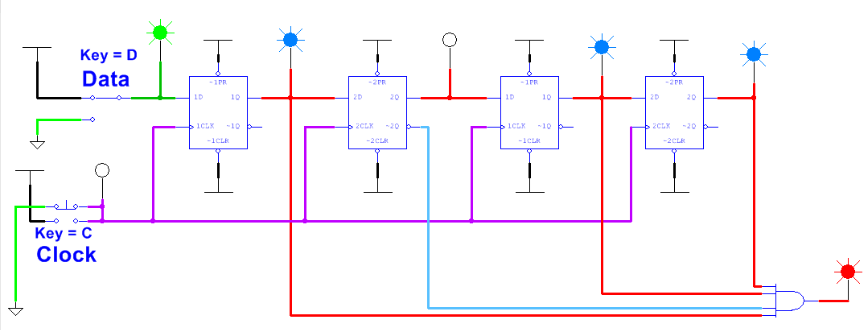


* + - ~Preset and ~Clear are connected to 1. Every time the Clock is triggered by an edge, Q refreshes to what D or J is. Then D or J inverts, and this cycle keeps on going. Q’s frequency is half of Clock’s, as D or J only inverts either at Clock rising edges or falling edges, but not both. Rising edges happen half as often as both rising and falling edges. Same deal with using falling edges.

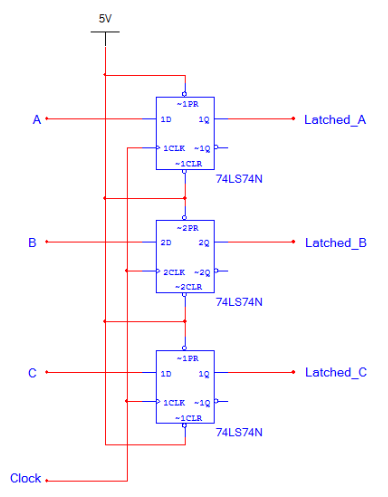


* + Shift Register
    - Shifts data from one flip flop to the next on clock trigger
    - Serial in/Serial out: data shifting from left to right or right to left in a line
    - Parallel in/Serial out: data come in in parallel (at once) and existing data shift out serially (in a line)
    - Parallel out/Serial in: Data comes in serially and shifts parallely

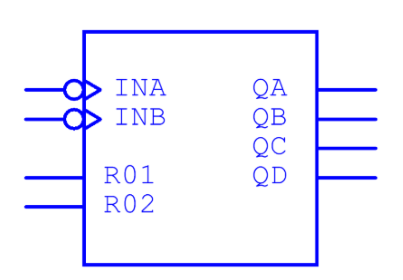
Serial in/Serial out:



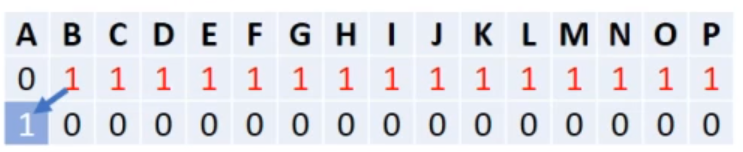
* + - ~Preset and ~Clear are connected to 1. Each Q is connected to the D of the previous flip flop. Each Clock is connected to the same clock
    - On Clock trigger, each flip flop makes their Q output what its D is connected to (the Q of the previous flip flop) to pass a signal along
  + Data Synchronizer

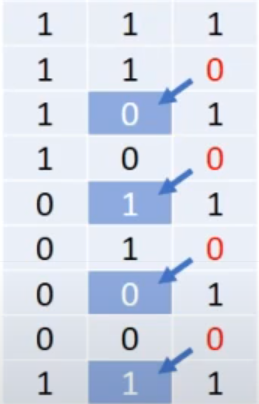
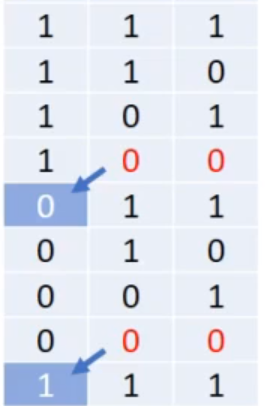
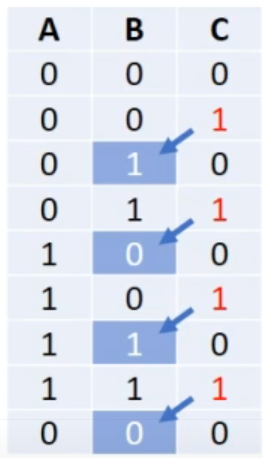
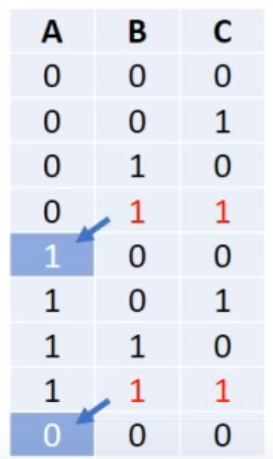


* + - All ~Clear and ~Preset are connected to 1. All Clock are connected to the same clock. The D inputs don’t make it across out Q until the Clock is triggered, to account for different propagation delay, to ensure signals are sent at the same time
  + Asynchronous counter
    - Look at a truth table. The LSB is just toggling back and forth between 1 and 0. Every successive bit is just toggling like the previous bit, but at half the frequency of the previous.
    - So, you can take the divide by 2 circuit design as the LSB
    - To make the next bit toggle at half that frequency, set up another divide by 2 circuit, but instead of connecting the clock to a clock voltage, connect it to the Q or ~Q of the previous flip flop
    - To count up with positive edge trigger clocks, connect next clock to ~Q, since on the truth table, when counting up, a bit toggles on falling edges
    - To count down with a positive edge trigger clocks, connect next clock to Q, since on the truth table, when counting down, a bit toggles on rising edges
    - For negative edge trigger clocks, connect opposite of what you would for positive edge trigger clocks
    - Due to propagation delay, not all flip flops toggle at the same time. As a result, the counter would momentarily display a random number, since some bits haven’t been updated yet.
    - States are the amount of numbers a counter can count
      * States = 2# of flip flops
    - Modulus is the number of states that are used
    - Asynchronous Truncation
      * Limits count range
      * Set up SSI logic to detect a number
      * When counting up, the number to detect should be one more than the max count
      * When counting down, the number to detect should be one less than the min count
      * When that number is detected, use the preset (1) and clear (0) on flip flops to set the number to resume count from
    - 74LS93
      * 3 or 4 bit up counter

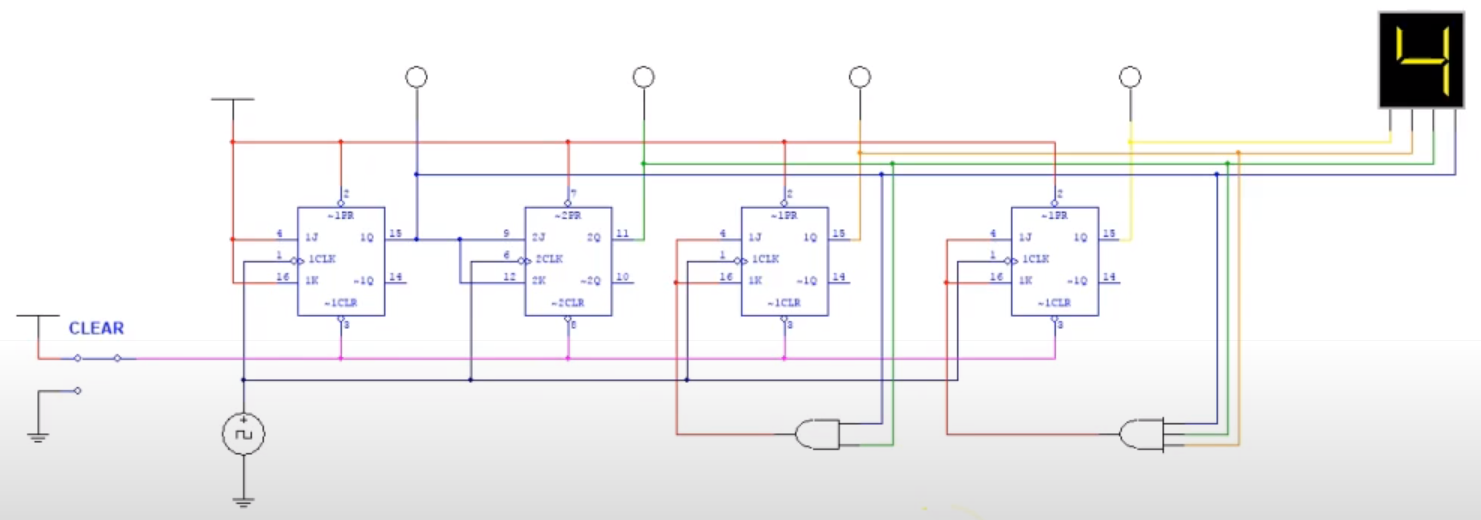


* + - * QA is LSB, QD is MSB
      * R01 and R02 both have to be true to reset count to 0000
      * INB is the clock for flip flop B
      * INA is the clock for flip flop A
  + Synchronous counters
    - The external clock is connected to the clock in of each flip flop, so all bits update at the same time
    - Look at a truth table

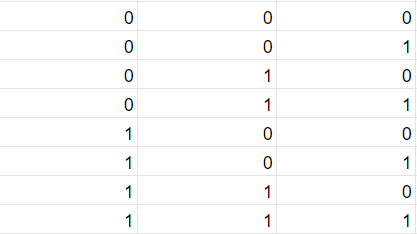




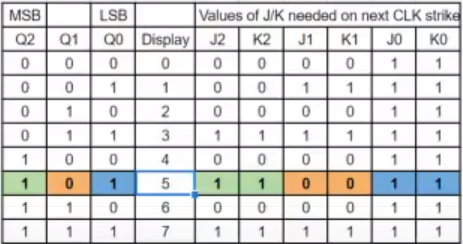
* + - Up counter
      * A bit toggles when all the bits of lesser significance (between the bit to toggle, exclusive, and the LSB, inclusive) were 1 during the previous count. In the next count, the bit toggles, and the lesser significance bits become 0 (because carry over: 111 + 1 = 1000)
      * To wire a circuit, use JK flip flops that enters toggle mode (J and K are activated on clock) whenever all (use AND gates) the flip flops representing lesser significant bits have 1 as output (Q)



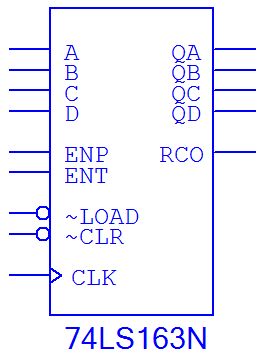
* + - Down counter
      * Like the up counter, but a bit toggles when all the lesser significant bits are 0. (because borrowing: 1000 - 1 = 0111)
      * To wire a circuit, the AND gate should detect the inverted output instead (~Q)
    - Asynchronous truncation works the same way as asynchronous counters
    - For D flip flops, a flip flop outputs a 1 as long as in the counter’s previous state:
      * That bit was a 1, and the lesser significant bits were not all 1
      * That bit was a 0 and all the lesser significant bits were all 1
      * Use XOR gates



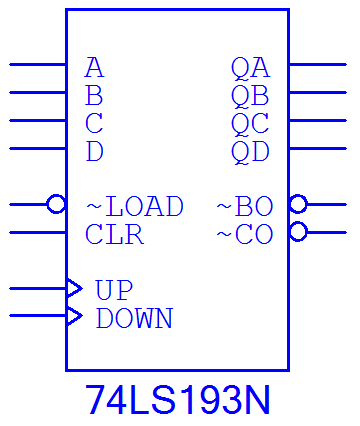
* + Synchronous counters with synchronous truncation
    - Look at a truth table

 ← 0-5 up

* + - Should show the count range. Note when a flip flop has to toggle. Then, fill in what J/K of each flip flop have to be, so that on next clock strike, the flip flops that need to toggle would toggle
    - Mark where you want to end count at. Note which flip flops have to toggle in order to get the counter’s start value on the next clock pulse.
    - Then, K-map the J value of each flip flop to determine what logic the J and K of a flip flop should be connected to
    - 74LS163
      * 4 bit synchronous up counter with synchronous truncation



* + - * QA-QD are outputs, with QA being LSB and QD being MSB
      * RCO outputs a 1 when count is at 15. Useful for cascading counters
      * Counting occurs when ENP and ENT are both enabled. Disabling either of them will pause the count
      * CLK is for a clock signal
      * When ~CLR is activated, the counter resets to 0 on the next clock pulse
      * When ~LOAD is activated, counter resets to whatever ACBD is on the next clock pulse, where A is the LSB and D is the MSB. Useful for truncation
  + 74LS193
    - 4 bit up/down synchronous counter with asynchronous truncation



* + - QA-QD are outputs, where QA is the LSB and QD is the MSB
    - UP takes a clock signal for counting up
    - DOWN takes a clock signal for counting down
    - ~BO outputs a 0 when count is 0. Useful for cascading down counters
    - ~CO outputs a 0 when count is F. Useful for cascading up counters
    - When CLR is activated, count immediately resets to 0. Useful for truncation
    - When ~LOAD is activated, count immediately resets to whatever ABCD is, where A is LSB and D is MSB. Useful for truncation

Computing devices

* Microcontroller: A small, cheap computer all on a single IC chip
  + Ex: Arduino Uno boards have a ATmega 328 microcontroller
* Arduino Uno Reference sheet:

<https://www.ele.uri.edu/courses/ele205/ELE205Lab/ELE205_Lab_files/Arduino%20-%20Reference.pdf>

* + - * Software are called sketches
      * Supports variables, comments, functions
      * Language is similar to C/C++, and that’s what it gets transpiled into
      * Core libraries are mainly written in C and C++, and compiled using avr-gcc and AVR Libc
      * setup() initializes pins and libraries
      * loop() loops code
      * // opens single line comments (not executed)
      * void indicates that the function doesn’t return a value
      * Procedures go inside {}
      * Variables store a value behind a name
        + Defining a variable requires defining its type too
      * Pins with a ~ support PWM
      * void setup()
        + Assigns whether pins are inputting or outputting
        + pinMode([pin], [INPUT/OUTPUT])

1st argument is the pin number

2nd argument is whether that pin should be an input or output

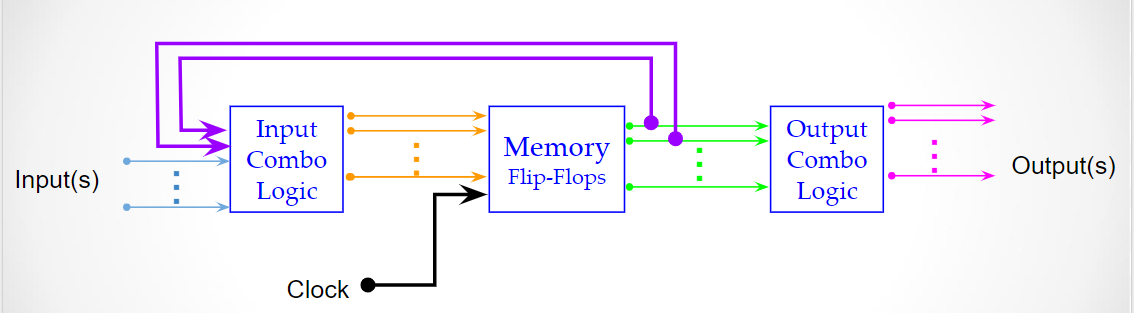
* + - * void loop()
        + Code that loops
      * digitalWrite([pin], [HIGH/LOW])
        + The first argument is which pin to send the signal out of
        + The 2nd argument is whether to send out a 1 or a 0
      * delay([time])
        + Time in ms to pause code execution
      * digitalRead([pin])
        + Reads whether a pin is outputting a 1 or 0
      * Variables
        + To initialize them, type the type (ex: bool, char, int, etc.), then its name, then = initial value
        + Add [] after the name to make into an array

Array contents enclosed in {}

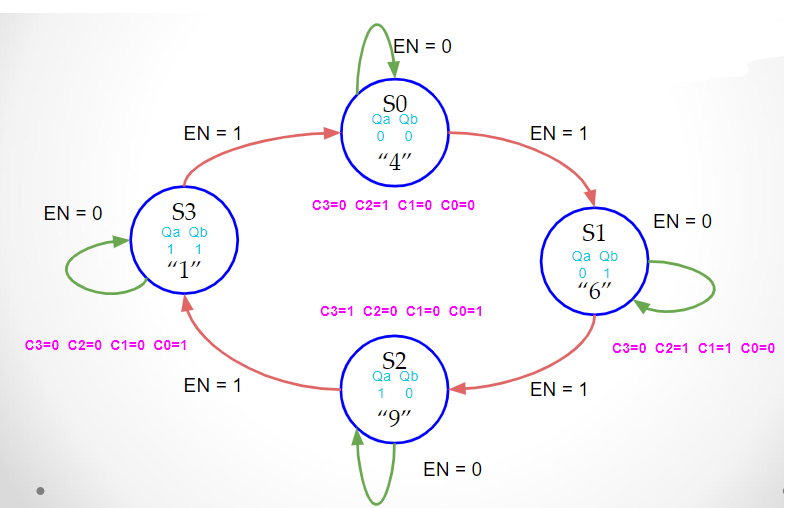
* + - * + Variables defined in a void are localized to that void
        + Global variables are defined outside of voids
        + == checks for equality
        + = assigns values
      * Serial monitor is like console.log in js
    - Has 14 digital I/O, 6 of which are PWM (encode analog content in a digital signal by altering duty cycle) enabled
    - Has 6 analog I/O
    - Draws power from USB (type B) or 2.1mm center-positive plug
  + Has a CPU, memory and I/O
  + Programmed using a Programming language, with specific syntax/grammar
    - Text code written in an editor
    - Editor may be part of an IDE, an environment with added features to aid in software development
* Microcomputer: small, cheap computer containing a microprocessor/CPU
* Microprocessor: a single IC chip that only does processing. Requires RAM, ROM connected externally for it to be functional
* Prefix micro means it’s not one of those gigantic computers that fill up a whole room a few decades ago

State Machines

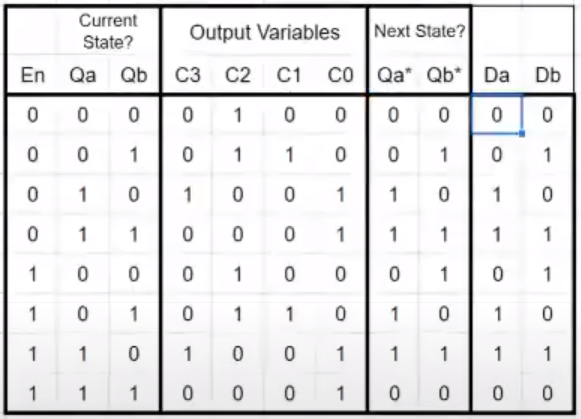
* Synchronous, sequential circuit with sequential and combinational logic. Progresses through a predictable sequence in response to something.



* Basically puts combinational logic (logic gates and switches/inputs) together with synchronous counters/flip flops as memory, as well as possibly additional combinational logic to reach output
* State graph
  + State bubble
    - Encapsulates a state
    - Inside a circle
    - State number
    - State variables
      * Represents what the flip flops should output to notify the machine of what state it’s in
    - Output variables
      * Represents what the user of the machine should see
  + Transition arc
    - Arrow going from one state bubble to another (or can loop back to itself)
    - Labeled with what conditions (coming from inputs - combinational logic section) need to be met to make that transition



* Transition table
  + Truth table that determines how the flip flops should change to reach its next state, as well as what the output should show in each state



* + A section that shows every possible state variable combination for every possible input combination
  + A section that shows, based on the input and state variables are, what would the state variables become on clock pulse, according to the state graph
  + A section that shows what would need to be fed to the inputs of flip flops in order to reach the combinations listed from the previous section
  + K-map the first section I mentioned (input) with the third section I mentioned (output) to determine what logic (coming from inputs + memory) would need to feed into the inputs of each flip flop
  + A section that shows the machine’s output in its corresponding state
  + K-map the above section (output) with the first section I mentioned (input) to get the logic needed to feed the output device based on current state