Digital Electronics

* Operate on digital signals
* made from assemblies of logic gates

Number Systems

Writing system for expressing numbers – using digits or symbols in a consistent manner

Same sequence of symbols may represent different numbers in different systems.

Decimal is used by humans

Binary is used when discussing issues close to the machine

Hex is used when humans interpret what is happening in the machine

The value of the symbol x, which is the ith symbol in the order, is (i-1)\*baseposition, where position is the num of places left of the units.

cab.bc3 = 2\*32 + 0\*31 + 1\*30 + 1\*3-1 + 2\*3-2

Binary

Computer store characters as binary numbers

Each digit is known as a bit or **bi**nary dig**it**

A bit can be 0 or 1

Nibble = 4 bits

Byte = 8 bits

Word = 32? Bits (CPU dependent – can be 64)

Half Word / Double Word – dependent on word size.

11101000.1012

= 232.625

To convert decimal fractional part to binary, repeatedly multiply the fractional part by 2 until it becomes 0

If the ith result is >= 1, put a 1 in the ith position to the right of the radix (decimal point), else place a 0

e.g for 0.4062510

.40625 \* 2 = .8125 < 1 -> 0

.8125 \* 2 = 1.625 > 1 -> 1

.625 \*2 = 1.25 >1 -> 1

.25 \* 2 = .5 < 1 -> 0

.5 \* 2 = 1 -> 1

So answer is 0.011012

Hexadecimal

16 symbols – 0-9 and A-F

Advantages: more compact, easier to write and read than binary, easy to convert to and from binary

Programmers must be aware of what they’re writing… BEEF and BEEF16 mean different things

In Java – 0xBEEF = BEEF16 <- very common notation

C2D.1016 = C\*162 + 2\*16 + D\*1 + 1\*16-1 + 0

= 3117.0625

To translate from binary to hex, separate the binary number into nibbles starting from the radix, adding 0s where needed.

Translate each nibble into hex digit

Reverse is true for hex -> binary

Binary Arithmetic

Do addition one by one to prevent multiple carry bits causing a mess

Overflow error occurs when the computer attempts to handle a number outside of the range of numbers it can represent during the execution of a program. This should trigger a flag in the status register but can cause an error

Negative Numbers

Negative numbers can be represented with *signed magnitude* representation: add a 0 for positive, 1 for negative

e.g. 0000 0110 = 6, 1000 0110 = -6 (not 134)

Similar concept as minus sign, but has 2 different values for 0

*Ones Complement*

The negative of a number is represented by splitting each bit

0100 1001 = 73, 1011 0110 = -73

Higher order bit still indicates sign, but still has 2 representations for 0 (all 0s or all 1s)

*Twos Complement*

The negative of a number obtained by flipping each bit and adding 1

0100 1001 = 73, 1011 0111 = -73

(1000 0000 = -128 – it is its own negative using Twos)

*Biased numbers*

Used in storing floating point numbers

Stores a number N as unsigned value N+B where B = bias (constant)

For k-bit numbers add bias of 2k-1-1 and store in binary

e.g. -65 -> -65 + 127 = 62 -> 0011 1110

We will stick to Twos Complement

Subtracting two binary numbers is done by adding the negative of that number.

Floating Point Representation

Floating point is like standard form – the typical floating point representation contains a sign bit S , the exponent e and the mantissa M.

(+/-) M \* 2e

Single precision 32 bit floating point numbers have 1 bit sign, 8 bit exponent, 23 bit mantissa

Sign Bit:

For sign bit – 0 is +, 1 is –

Exponent:

The exponent e is a value between -126 and 127. A bias of 127 is added giving a number between 1 and 254

Exponent of 0 with mantissa 0 gives number 0

Exponent of 0 with non-zero mantissa – subnormal numbers

Exponent 255 with mantissa 0 = + or – infinity

Exponent 255 with non-zero mantissa = NaN

Mantissa:

Always scaled so the radix follows the leading 1 (since sign bit is separate). Since this leading 1 is always there, it is not stored. Instead only the first 23 bits of the fractional part are stored.

(examples on lecture slides ig)

Floating point operations should return the closest FP number to the answer

Minimum positive number = 2-126, maximum 2127

Transistors and Logic Gates

Transistors are electrically controlled switches that turn on or off when voltage or current is applied to a control terminal. Most common transistor is MOSFET – Metal oxide semiconductor field effect transistor. 57 billion MOSFETs fit onto one silicon chip.

Digital circuits are made up of logic gates, which are designed to perform a function of Boolean logic. Logic gates are made up of at least one transistor. Outputs from logic gates can feed into the inputs of other gates.

Transistors nMOS or pMOS have 2 ports d and s, which are connected depending on the voltage of the third port g. nMOS transistors connect d and s when g = 1, pMOS ones disconnect d and s when g = 1.

Logic gates: (diagrams and truth tables in lecture slides)

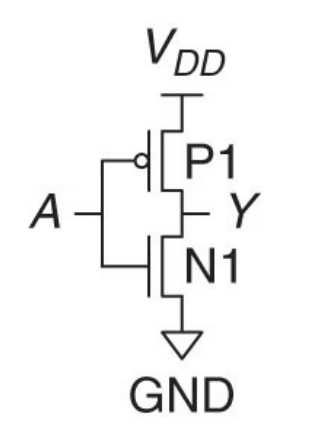
AND gate -> A • B

OR gate -> A + B

XOR gate -> A ⊕ B

NOT gate –>

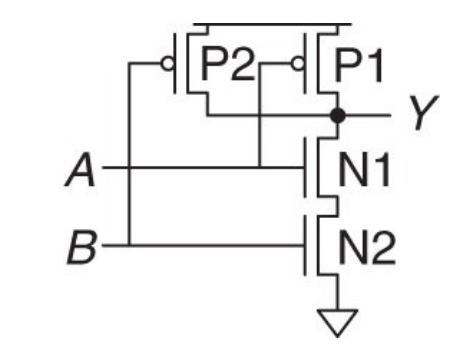
NOR gate



NOT gate in transistors:

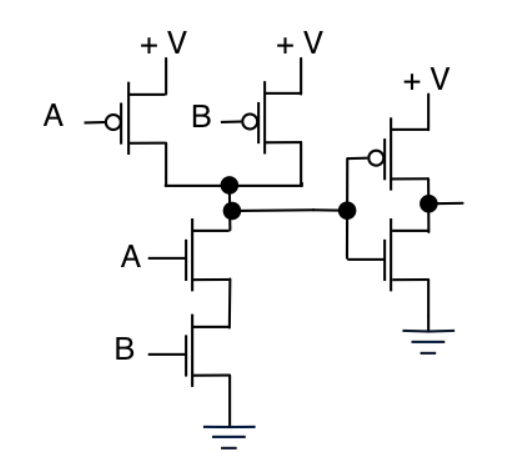
When A is on, pMOS P1 turns off and nMOS N1 turns on, so Y is connected to GND (=0). Otherwise, P1 will connect Y to VDD (=1).

Symbols for VDD, GND, pMOS and nMOS need to be learned.

NAND gate:

Both A and B must be 1 to turn off both p1 and p2 to prevent a connection between VDD and Y, resulting in Y = 0 in this case. N1 and N2 will form a connections between Y and GND instead.

(note 4 way junctions on the diagram only exist where there is a •, otherwise it means the wires pass over each other)

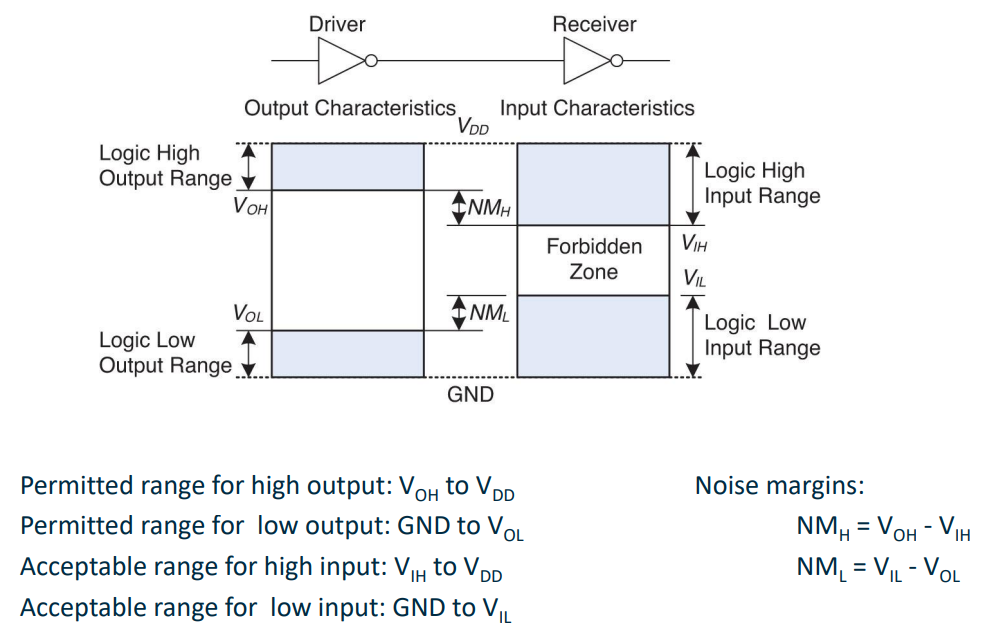
AND gate:

NOT

NAND

Logic Levels

Voltages used to determine 0 or 1 are usually between 0V and 5V.

The low voltage in the system connected to the ground is 0V. Historically, circuits were powered by 5V supply, so high voltage = VDD = 5V.

Receiver has a higher tolerance for noise than driver to account for voltage loss/gain between them.

Static Discipline – guarantee that if inputs meet valid input thresholds then outputs meet output thresholds.

Logic gates are grouped into families, and the static discipline is upheld when gates of the same family are used together. As a result, digital abstraction can be applied without further concerns about voltages or logic levels.

Moore’s Law

Transistor density doubles every two years

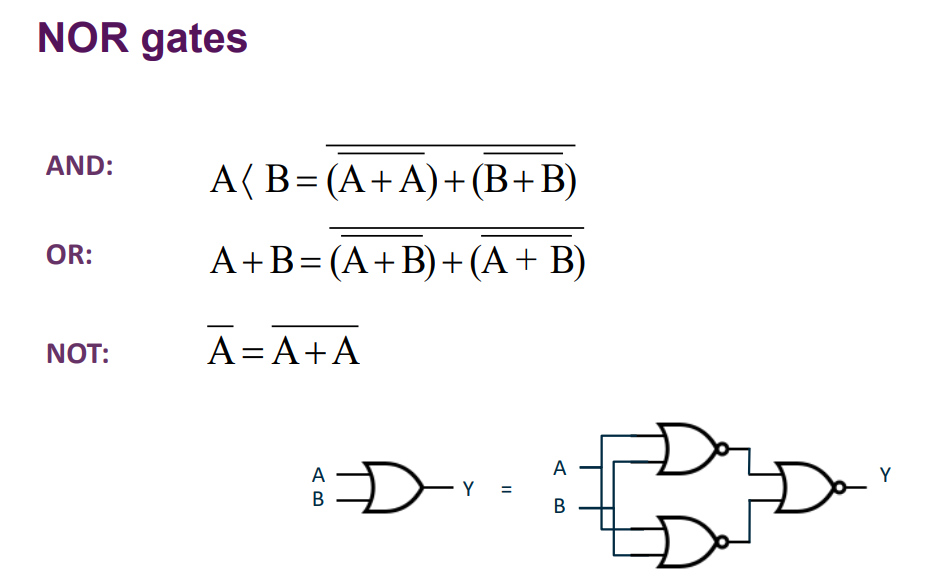
Boolean Algebra

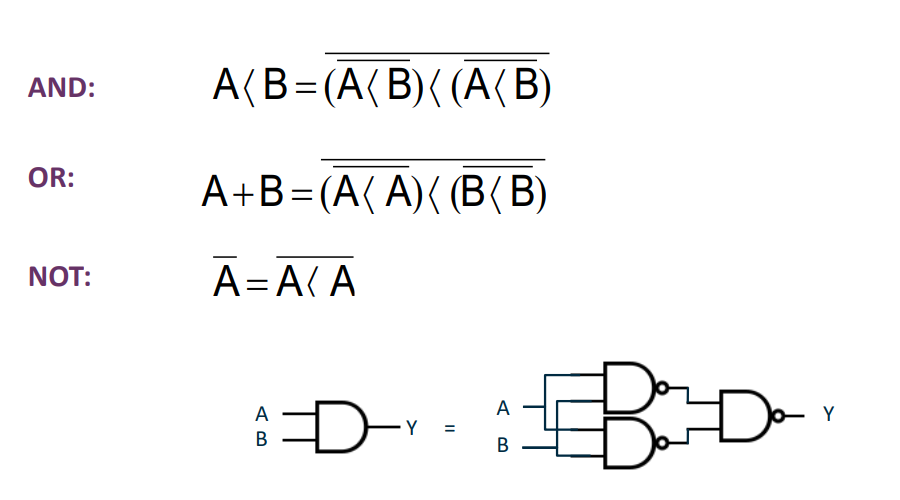
XOR Y = A ⊕ B = (A+B) • ()

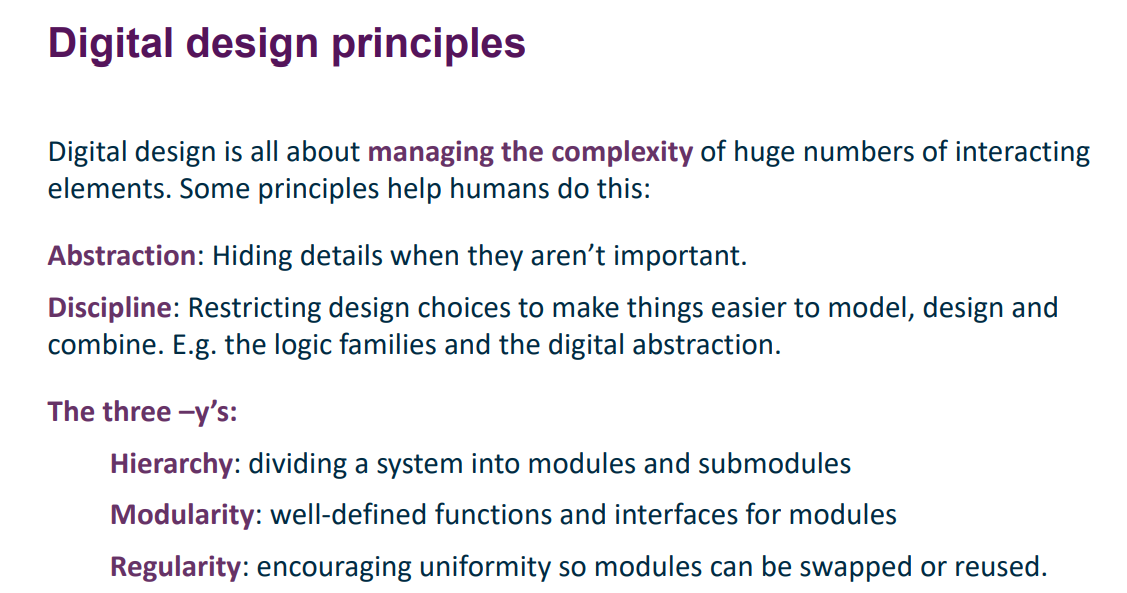
A functionally complete set of boolean operators is one which can be used to express all possible truth tables by combining members of the set into a boolean expression.

Any logic circuit can be constructed with just AND, OR and NOT – they form a functionally complete set.

NOR gates alone can also form a functionally complete set.

(NOR = )

NAND gates alone also form a functionally complete set.



A circuit has one or more discrete valued input terminals, and one or more discretely valued output terminals, as well as specifications of the relationship between inputs and outputs, and of the delay between inputs changing and outputs responding.

Circuits are made up of elements and nodes – elements are circuits with inputs, outputs and specs, and nodes are wires joining elements together, carrying a discretely valued voltage.

Combinational Logic Rules

* individual gates are combinational circuits
* every circuit element must be a combinational circuit
* every node is either an input to the circuit or connecting to exactly one output of a circuit element
* the circuit has no cyclic paths, i.e. every path must visit any node at least once.

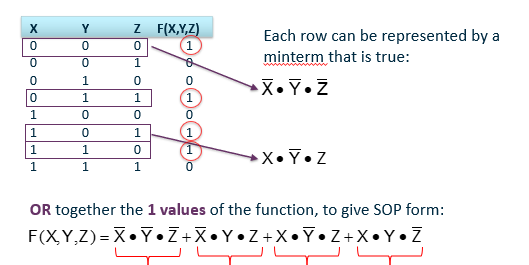
Boolean algebra – the algebra of 0/1 variables. Used for specifying the function of a combinational circuit and used to analyse and simplify circuits to produce truth tables.

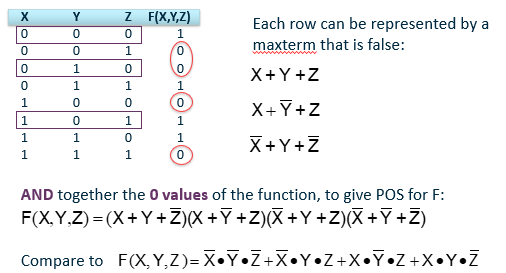
* Variables represented by letters
* The complement or inverse of a variable is written as
* A variable or its complement is called a literal
* The AND of several literals is called a product
* A minterm is a product where all the inputs to a function appear exactly once, either complemented or uncomplemented.
* The OR of several literals is called a sum or implicant, e.g. A+B+C.
* A maxterm is a sum in which all the inputs apprar exactly once, complemented or uncomplemented

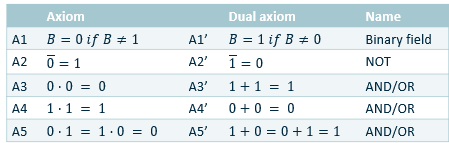
Sum of products (SoP) form – every boolean expression can be written as minterms ORed together.

e.g. (A.B.C)+(A.) + (.B.C)

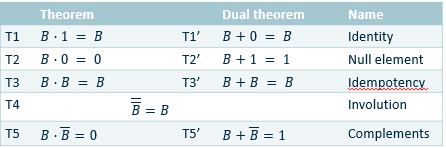
Product of sums (PoS) form – every Boolean expression can also be written as maxterms ANDed together.

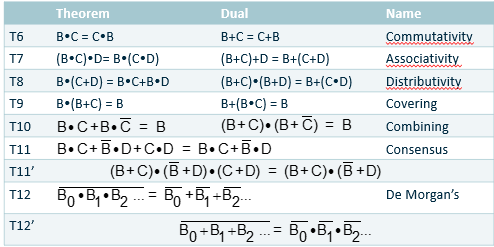
Each row on the truth table can be represented by a minterm which is true. OR together those which have an output of 1 to give the SoP form of the function.

Each row on a truth table can also be represented by a maxterm that is false, where a 1 in the truth table means that element is complemented in the maxterm. AND together those with an output of 0 to give PoS form of the function.

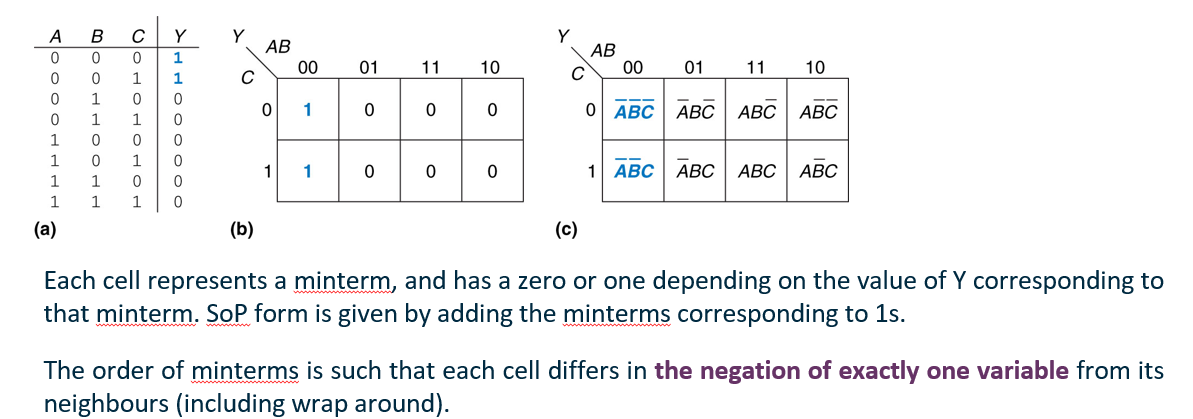
We can use Boolean algebra and Karnaugh maps to produce the simplest expression equivalent to the PoS or SoP, which can then be made into circuitry.

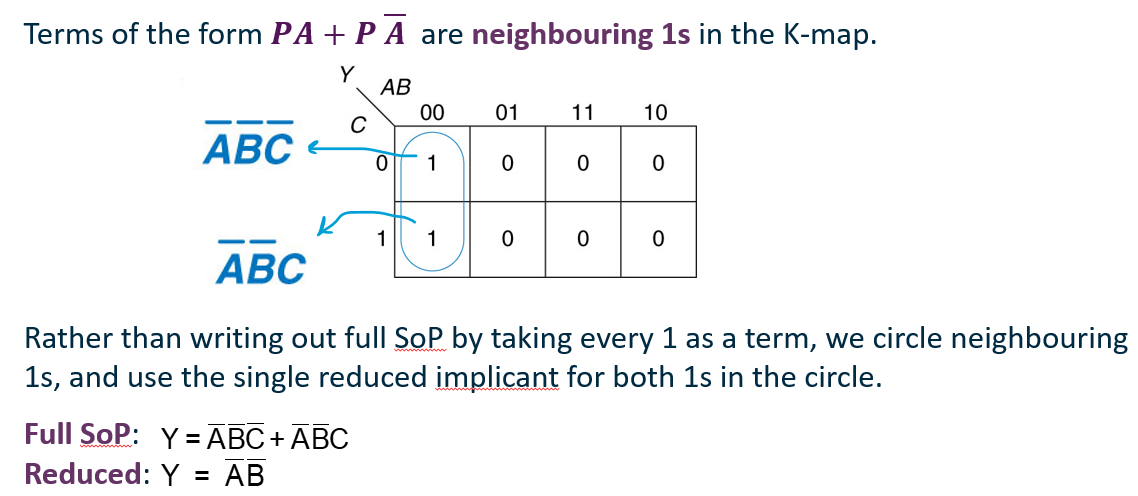
Axioms:

Theorems:

Example of truth table to circuit using SoP and Boolean simplification on lecture slides.

Karnaugh Maps

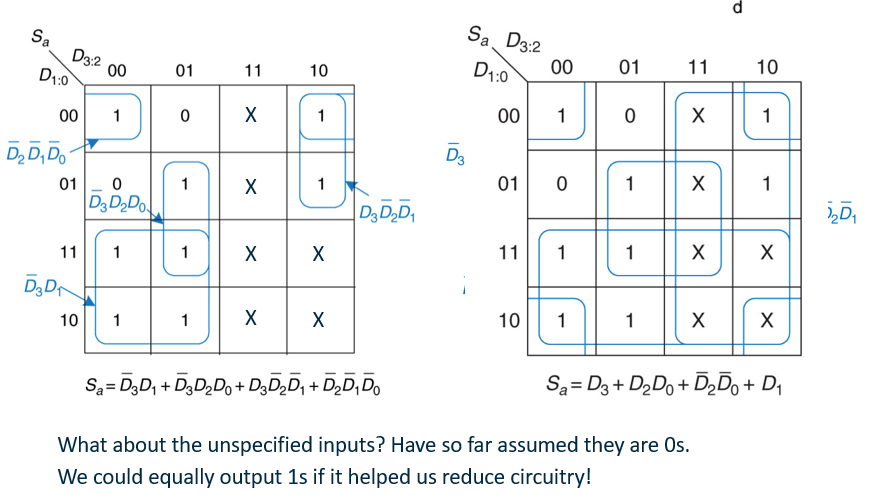
Karnaugh maps are a graphical way of representing Boolean formulae, which makes it easier to identify terms of the form to help with simplification.

(circles can wrap around, and should be as large as possible while being a perfect rectangle which is a power of 2 in each dimension, i.e. not of side length 3)

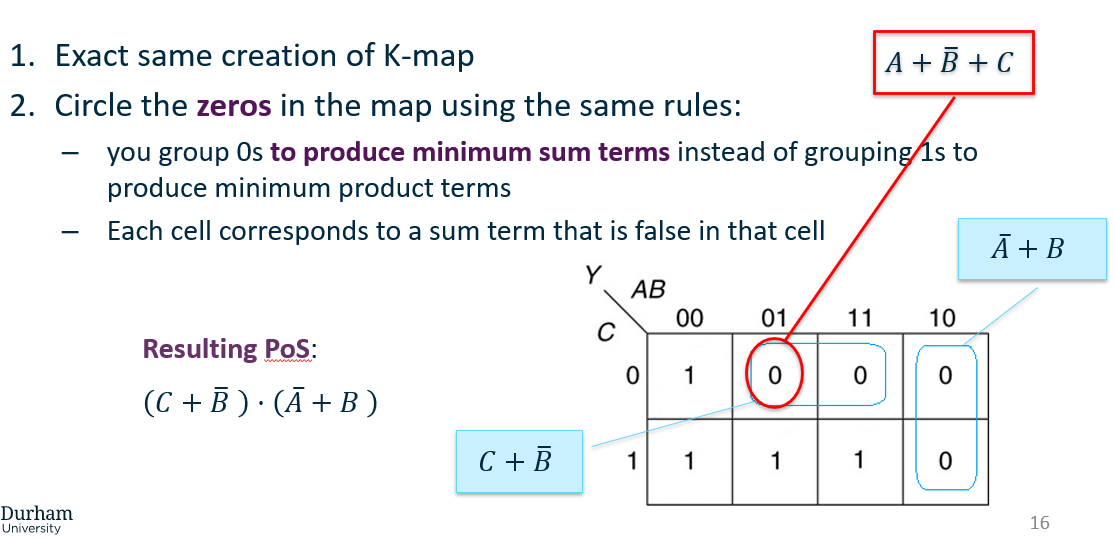
Circle all ones using as few circles as possible and making them as large as possible – a 1 can be in multiple circles.

For a Boolean expression to be minimal it must be the sum of prime implicants, i.e. the implicants cannot be split down any further.

Bigger circles = more simplified Boolean = less circuitry needed



Karnaugh maps can also be made for PoS, where the zeros are circled instead.



Circuits

Combinatorial circuits: (output depends only on current input)

* Adders
* Decoders – decodes a n-digit binary number into 2n data lines
* Multiplexers – uses a binary number to select an input

Sequential circuits: (output depends on state and input, i.e. history of inputs)

* Latches/flipflops

Half-adder:

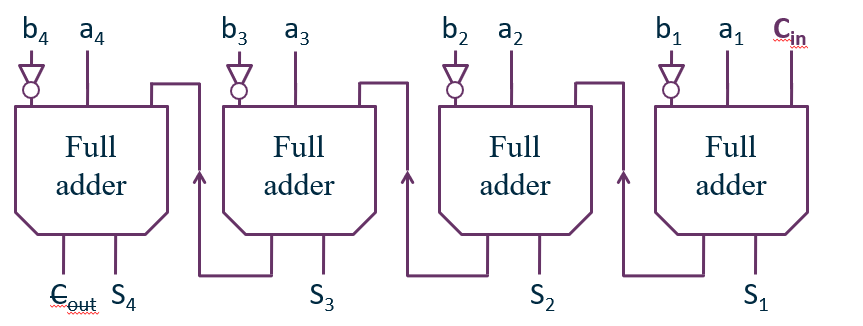
Inputs – A,B

Outputs – Sum, Carry

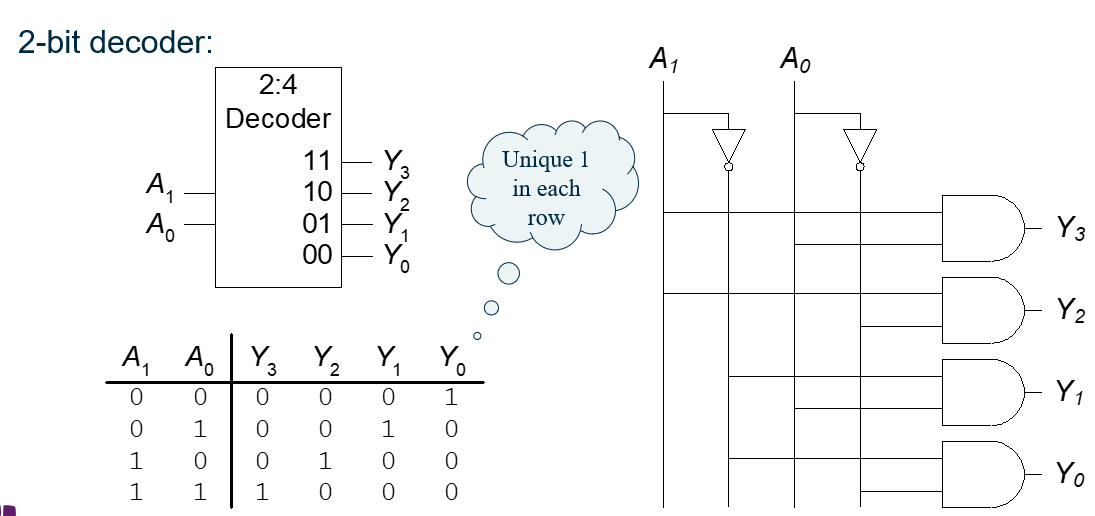
Full adder: same as half but with carry in as a third input

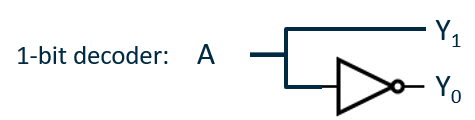
Full adders can be chained together to add multi-bit numbers.

Subtractor:

Subtracts b from a by adding -b using twos complement – i.e. all inputs representing the bits of b are notted, and Cin on the first full adder is set to 1 to represent the adding of 1 in order to convert b into -b.

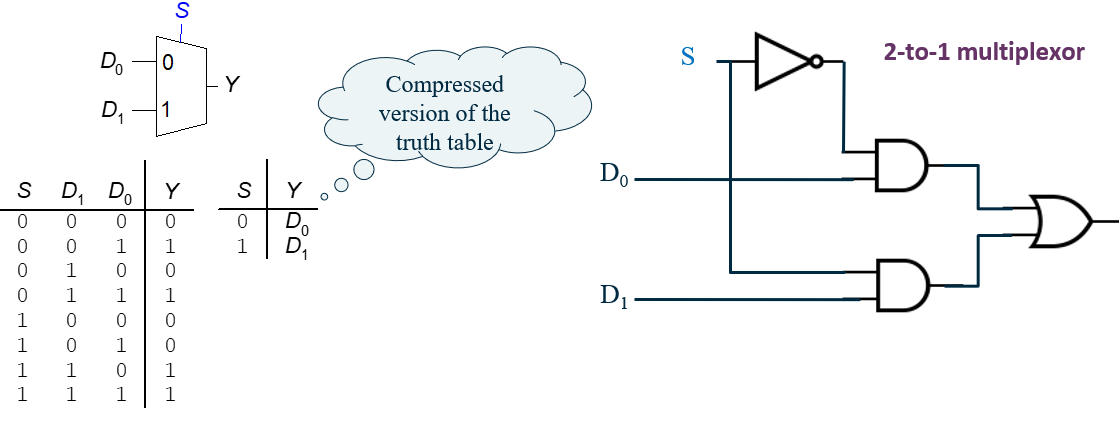
Decoder:

A decoder has N inputs and 2N outputs. Exactly one of its output bits is set to 1 depending on the binary number represented by the N input bits, the exact bit dependent on the input. Outputs are called one-hot since only one output bit is hot/high/1 at a given time.

^ 2 inputs so 22 = 4 outputs. Each combination of inputs gives a unique single-bit output.

Larger decoders require multi-input AND gates to be constructed, which requires a lot of circuitry. Slower but larger circuits can be created with fewer transistors, i.e. lower cost.

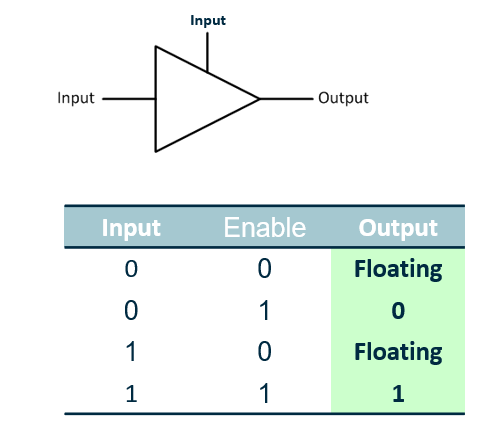
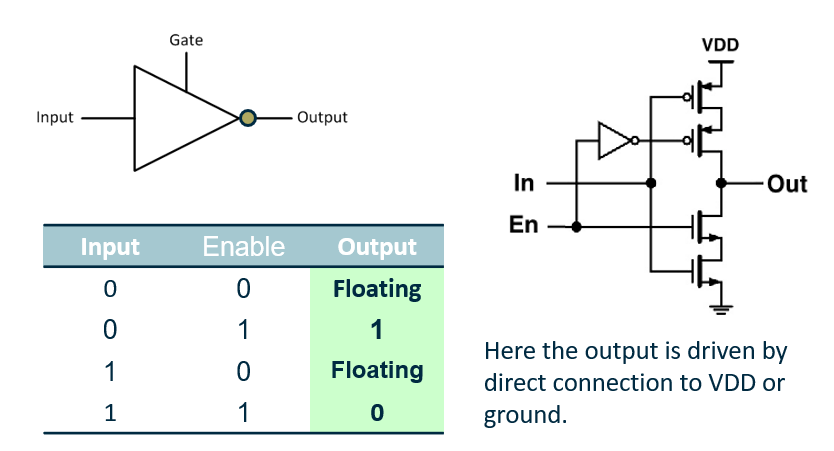
Mux – multiplexor:

Chooses 1 of many inputs to steer to its single output under the direction of control inputs. Has k+2k inputs, one of which will be copied to the 1 output. The first k inputs are the selector S, which represent a binary number. The output takes the value of one of the other 2k inputs, based on the value of the selector S.

^ Y takes the value of the input Dn, where n is the value represented by S. (This probably makes more sense than the waffle above.)

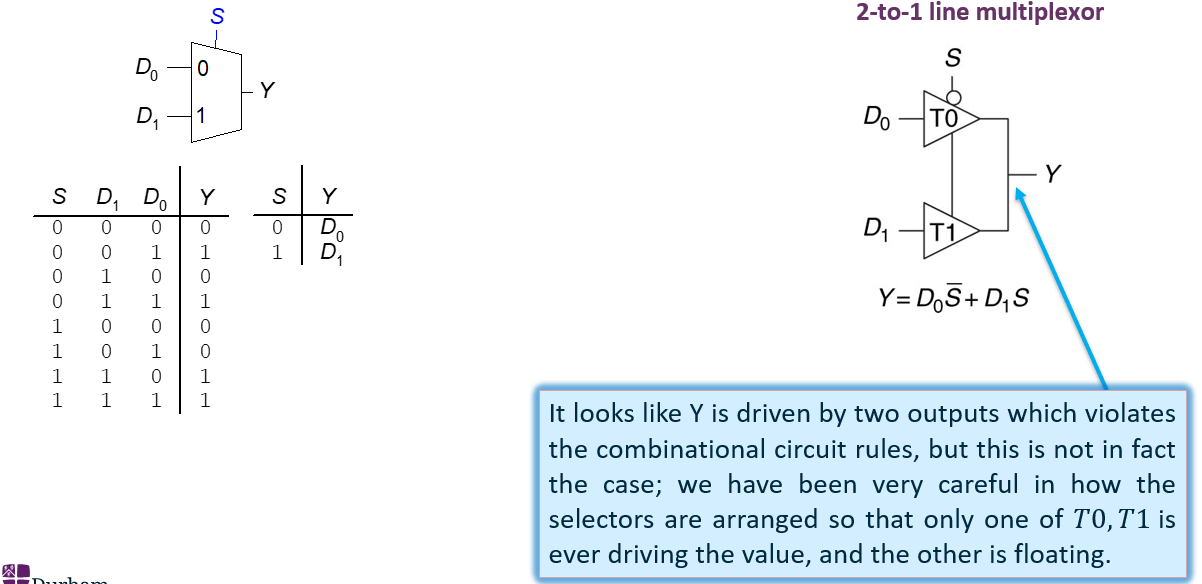
Tristate:

A buffer is a logic gate where its one output = the one input. This is used to regenerate the signal to amplify the current in the signal if it becomes weak.

In contrast, a tristate takes in a second input, called an enabler, which determines whether the input signal makes it through to the output. If the enabler is 1, the tristate functions like a buffer. If the enabler is 0, the output is “floating” – neither 0 or 1 (overridden by any 0 or 1 values in the same circuit. Ignore Boolean abstraction with this). Inverting tristates also exist.

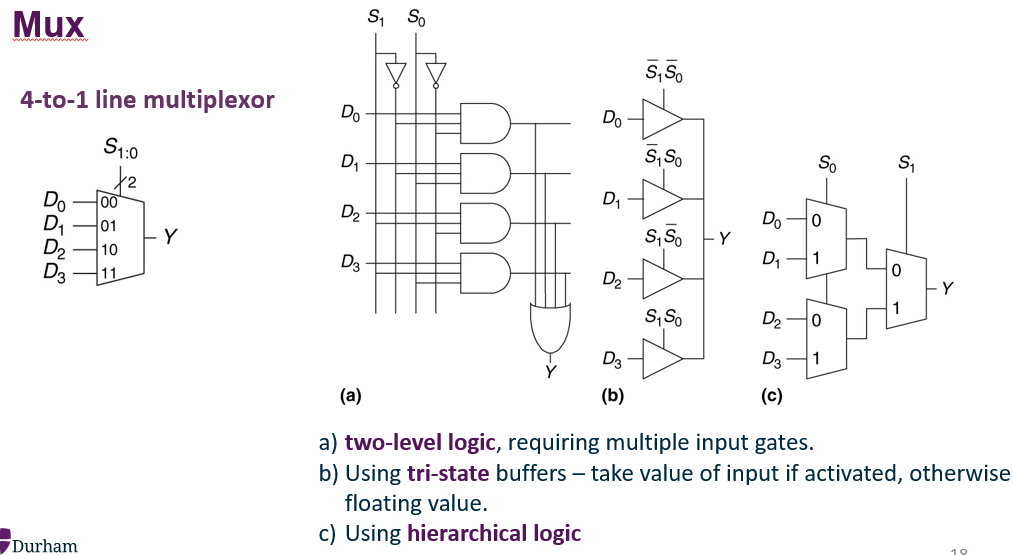
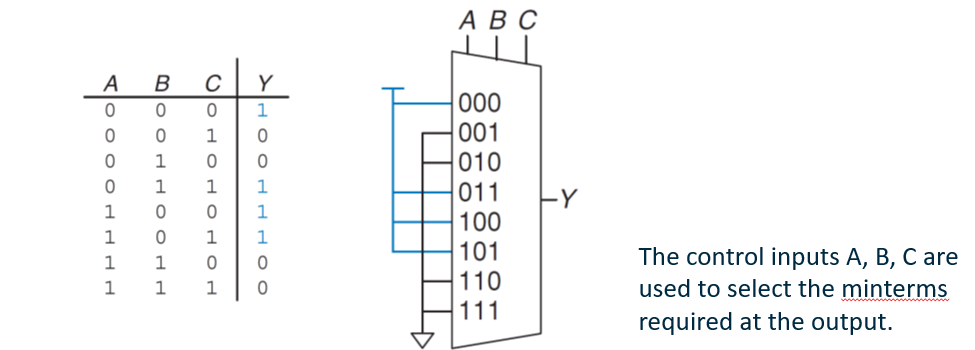


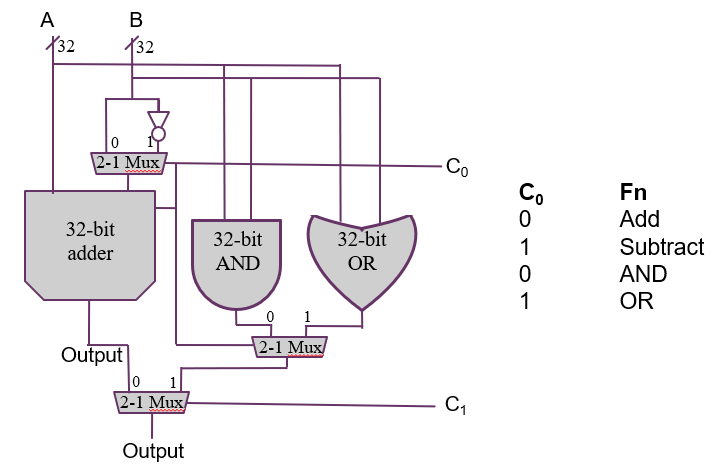
Yellow = NOT gate circuit, blue = enabler circuit (using pMOS and cMOS)

Mux with tristates:

^ S is the enabler for T0 and T1, but is negated for T0 so only the value of either D0 or D1 will be output to Y. This is what we want for a multiplexor 😊

Multiple representations of a Mux:

 An 8 input Mux can be used to implement combinational logic functions on 3 variables.

A simple ALU:

If C0 = 0, the adder will add and 0 is passed to the adder’s carry in. If C0 = 1, the input B will be negated and 1 will be passed through to carry-in – this turns the adder into a subtractor.

Timing

In any physical circuit, there is a delay between the input of a gate changing and the output changing accordingly. This delay can be broken down into Contamination Delay and Propagation Delay.

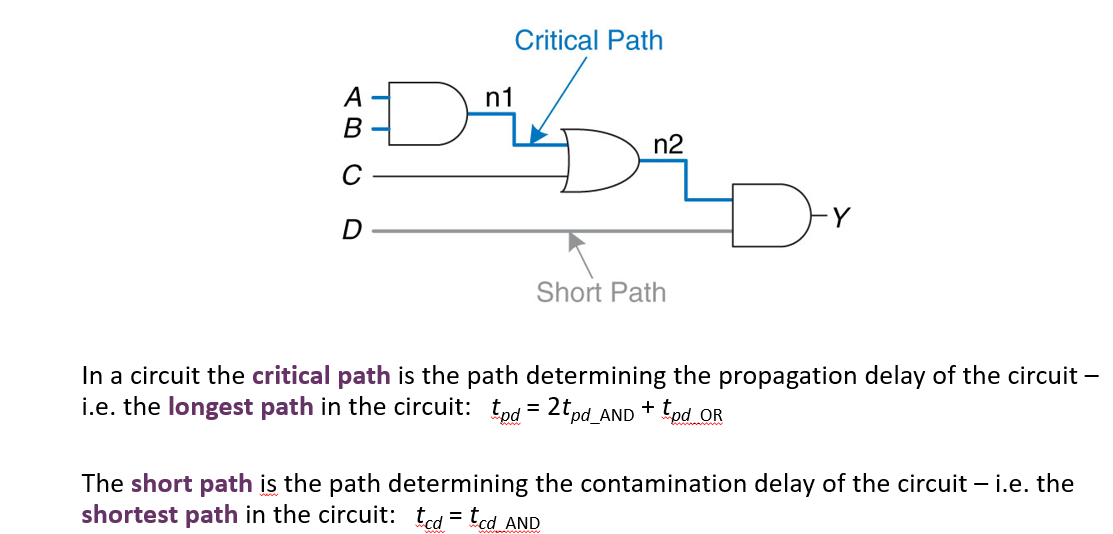
Contamination delay, tcd is the minimum delay before the output changes, i.e. the minimum time from when an input changes until any output starts to change its value. (The output remains at its old value for at least tcd)

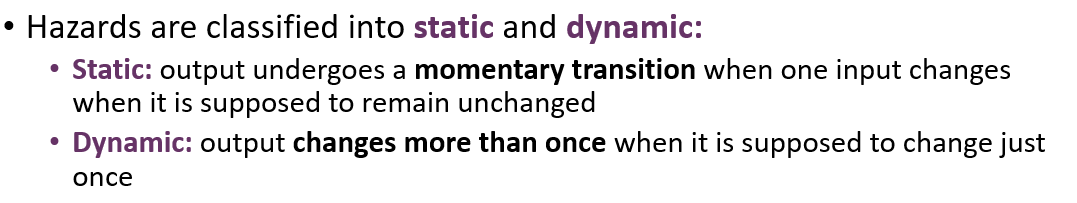
Propagation delay, tpd, is the maximum delay before the output is stable, i.e. the maximum time from when an input changes until all of the outputs reach their final value. (The output may change value but will stabilise after at most tpd)

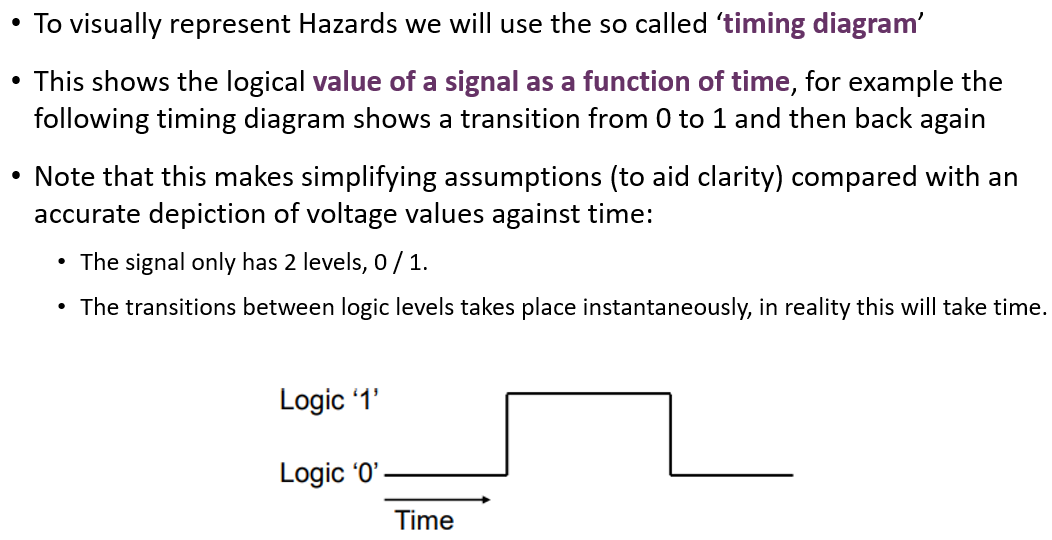
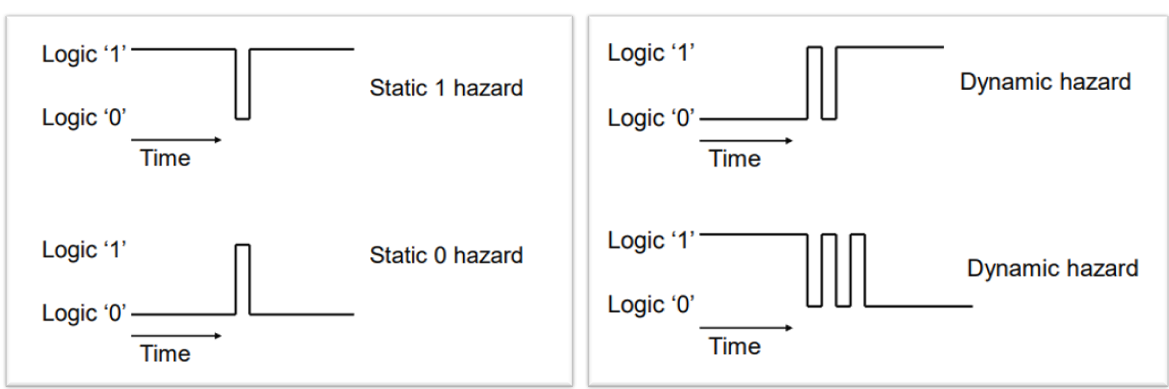
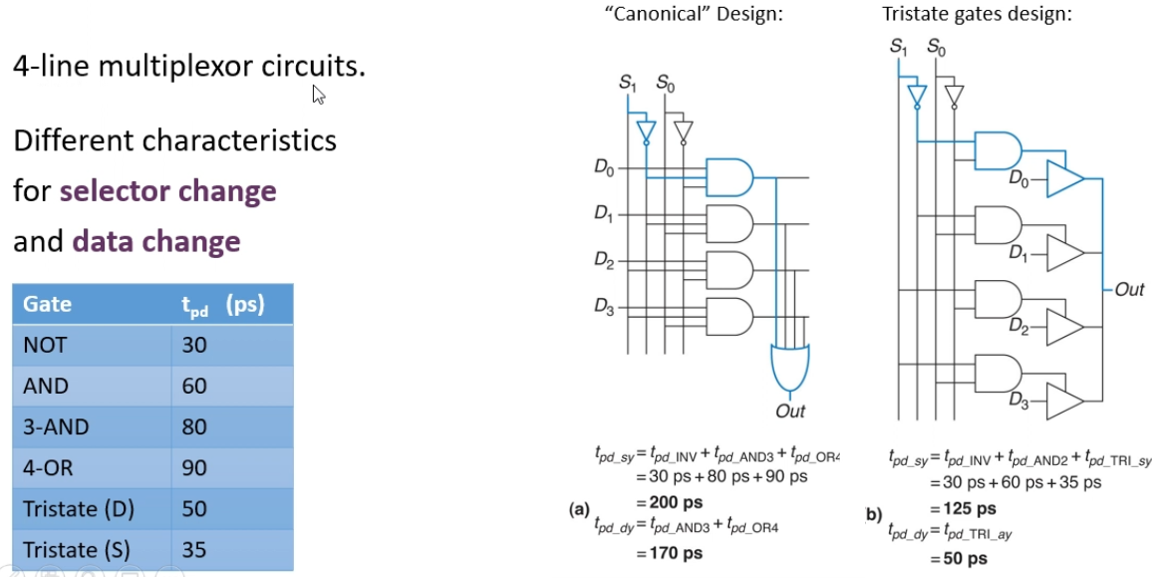
Delay is caused by capacitance and resistance, as well as the speed of light limitation.

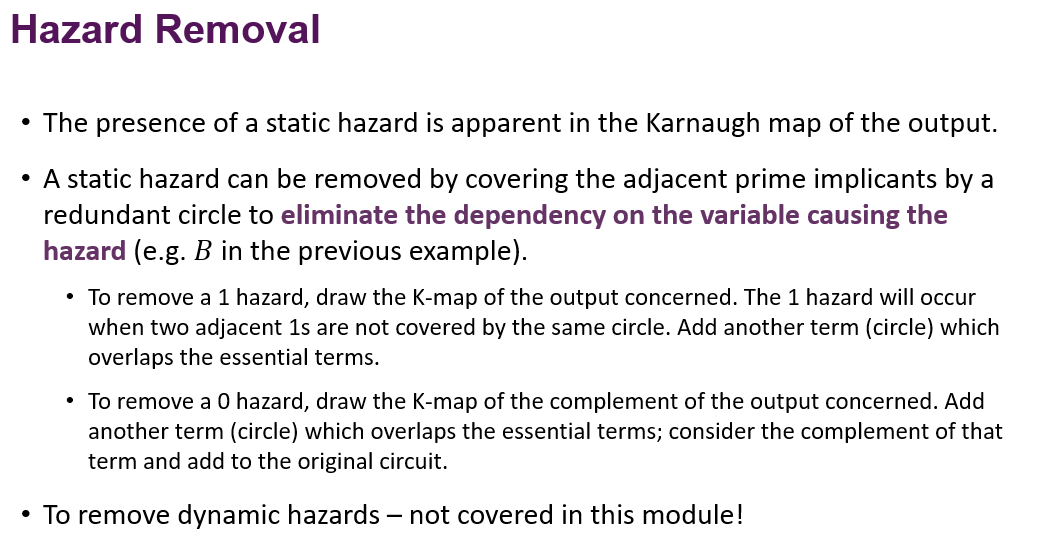
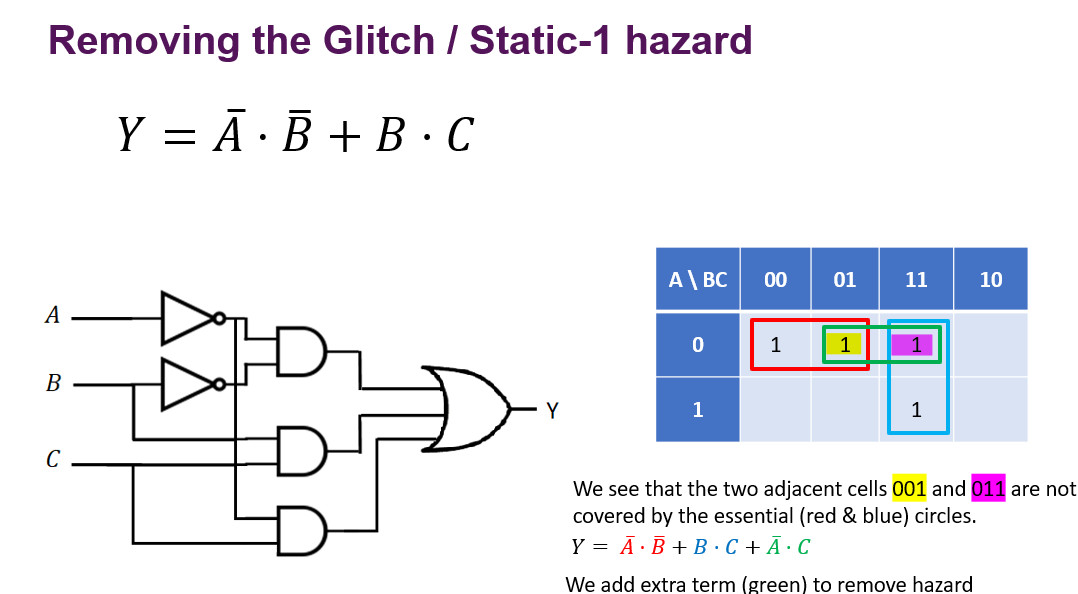
Computation of tpd and tcd is challenging due to differences in speed between different inputs/outputs of a circuit, and because circuit temperature affects speed -> assume values given by manufacturer.

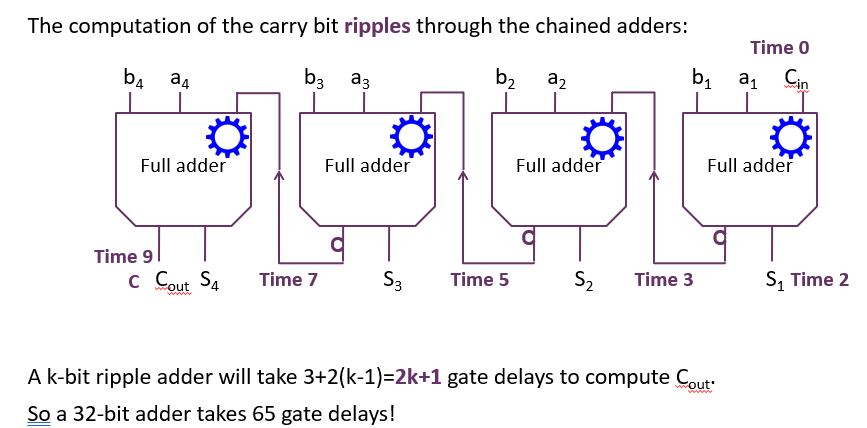
Critical paths:  
The critical path of a circuit is the path determining the propagation delay of a circuit, i.e. the longest path in the circuit. The short path is the path determining the contamination delay of the circuit, i.e. the shortest path.

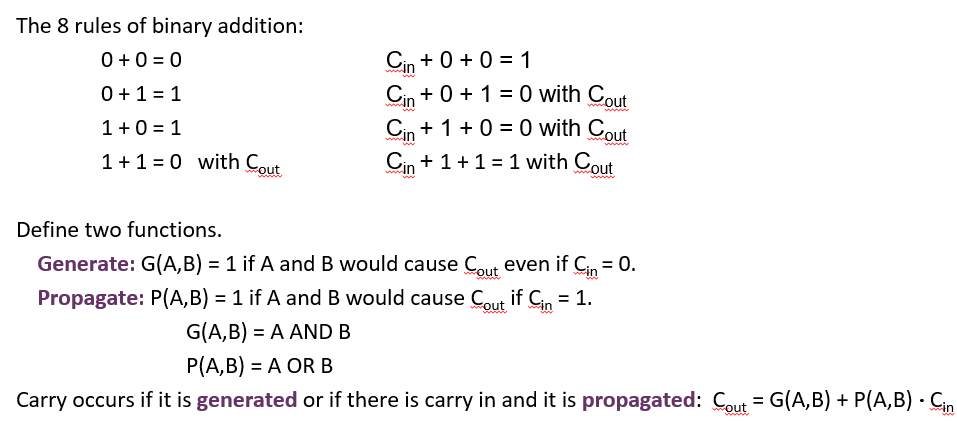
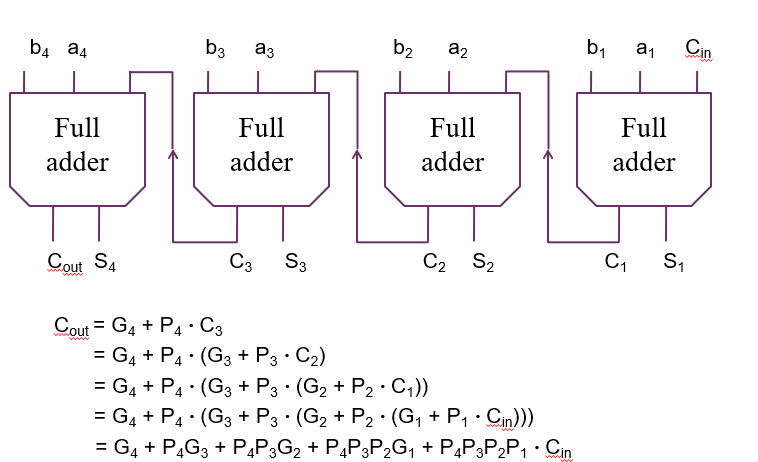
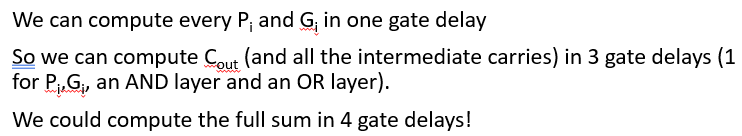
Gate delay:

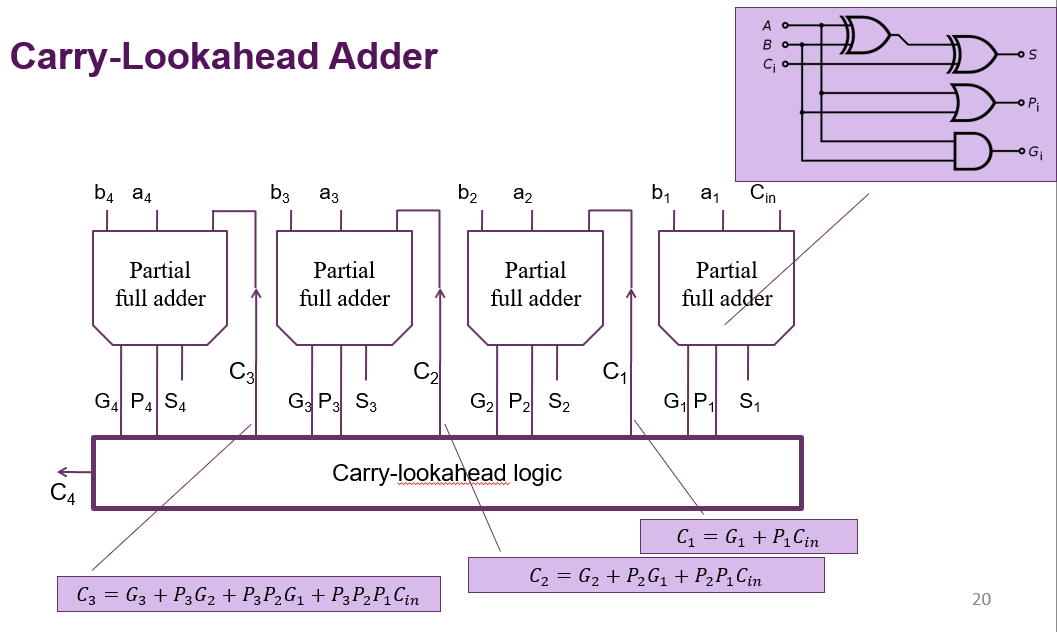
Gate delay can give rise to “hazards” or “glitches” which lead to unwanted brief changes in logic level of the output in response to changing inputs.

Calculating delays:

Advanced Adders

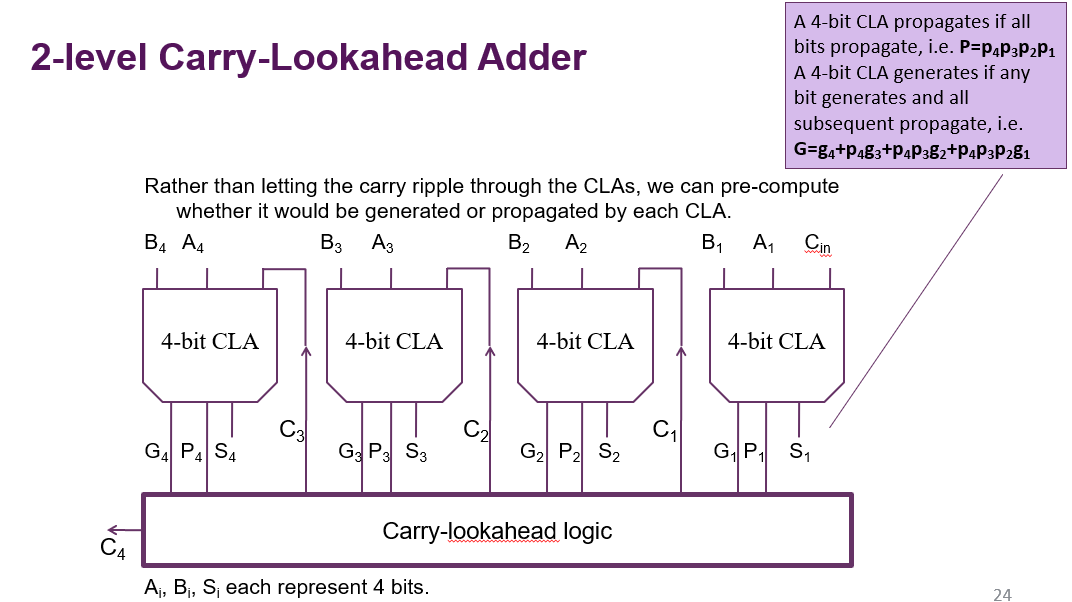
A full adder circuit takes 3 gate delays to calculate Cout, compared to 2 gate delay for calculating the sum S ( assuming all gates take the same time). The computation of the carry bit ripples through chained adders, building up the overall delay in calculation.



Tradeoff: circuit is more complex, but time is saved.

(note adder circuit has changed – now has Pi and Gi circuits instead of just a Cout. Ci is calculated in the mysterious carry-lookahead logic area)

CLAs for many-bit numbers are impractical as would require an order of n2 gates for n input bits. To solve this, CLAs can be chained in a ripple fashion (which will make it 4x faster than a regular ripple chain adder). Instead, 2 level CLAs can be used:



A 16 bit, 2 level CLA has 6 gate delays, compared to 33 for a standard chain of ripple adders.

