Digitaltechnik

Contents

MSB - Most significant bit
LSB - Least significant bit

$x\%1$ - drop decimal value from x.
2^n - number of possible states with n bits.

Resistance of a wire
ρ - resistivity of the metal (Ωm)
l - length of the wire
A - cross sectional area of the wire
$R = \frac{\rho l}{A}$

Modern electronics uses 0.8V as high.

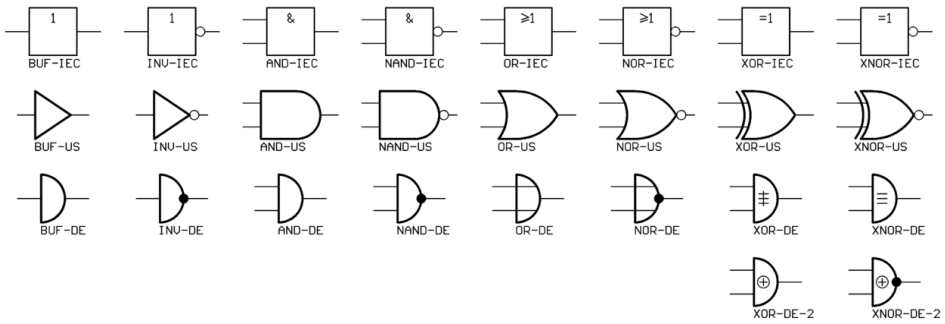
Floating Voltage - when a pin / contact is not connected by a “normal” (lower than that of air) resistance to V_DD / circuit ground. Essentially the same as any conductive surface in the room, on which a very weak 50Hz signal is usually seen due to induction from all the EM sources in the room.

Why digital instead of analog? Error correction.

Schaltfunktionen

Schaltfunktion - $Y = f(X_0, X_1, X_2, ..., X_{N-1})$ - Nimmt mehrere Bits als Input und produziert eine einzige Bit als Ausgang.

Alle Schaltfunktionen lassen sich als einer Wahrheitstabelle darstellen mit mindestens $N + 1$ Spalten und 2^N Zeilen, wo N ist der Nummer von Inputs.



NOT'ing a gate usually means the resistor just needs to be moved before the transistors (essentially appending a NOT gate).

OR - Disjunction

AND - Conjunction - The resistor after the output point is needed to prevent a short circuit when both inputs are high.

Antivalenz (XOR) - High if only one of the inputs is high.

XNOR - High if both inputs are the same, gate symbol is a =.

CMOS (Complementary Metal Oxide Semiconductor) Technology

Transistor - Trans-Resistor (changable resistor)
MOS Transistor - Electronic component with contacts **S** ource, **D** rain und **G** ate. Charge carriers flow from S to D. They are always controlled through a voltage between Gate and Source (unlike a current with BJT) and is therefore more efficient for very low / high power applications. They are also easier to etch in ICs and are therefore predominantly used in logic circuits.

Although very high pull up resistors vastly reduce power loss when using a single MOS transistor, such large resistances are difficult to fabricate in ICs. CMOS uses a PMOS instead which has practically ∞ resistance when "open".

$|V_{GS}| < |V_{th}|, R_{SD} \to \infty$ - The transistor is off
$|V_{GS}| > |V_{th}|, R_{SD} \to 0$ - The transistor is on

N-Type (NMOS) - Threshold voltage is positive. Negative electrons flow from S to D (Hence D is connected to the positive terminal in a circuit)
P-Type (PMOS) - Threshold voltage is negative. Positive Holes flow from S to D. Circle at the gate in symbol.

- CMOS Gatter müssen aus genau so vielen NMOS und PMOS Transistoren bestehen
- Bei m Eingängen gibt es m NMOS und m PMOS transistoren

The V_D of an "off" MOS transistor is floating (undefined) unless it is pulled up / down.

A CMOS gate can be split into two networks / Pfads:

	Pull-up	Pull-down
MOS Type	PMOS	NMOS
NAND	Parallel	Series
NOR	Series	Parallel

These can be converted between one another by breaking the circuit into parallel / series blocks until each block contains one transistor, then switching the type of transistor and connecting them again in the opposite manner (parallel ⇔ series). V_DD becomes the output and the output becomes ground.

LTD: Add some diagrams of CMOS circuits

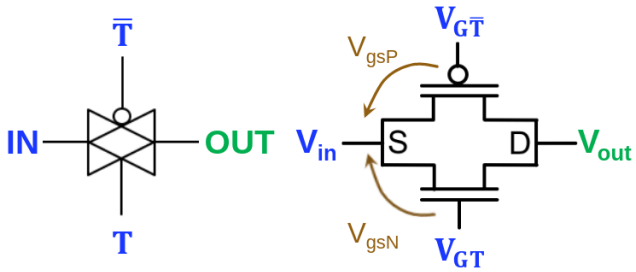
Switching Delays

- t_{pHL}, t_{pLH} - Propogation delay - Time taken between a 50% change in the input voltage leading to a 50% change in the output
- $t_{tHL}(t_{\text{fall}}), t_{tLH}(t_{\text{rise}})$ - Time between the output rising / falling between 10% and 90% voltage

$t_d = \frac{t_{pHL}+t_{pLH}}{2}$ - Average switching time, easier to work with in practice

Transmission Gate

These extremely simple gates are the CMOS equivalent of physical switches / relays, transmitting a signal if the T pin is high.



T	IN	Widerstand	OUT
0	0	hochohmig	-
0	1	hochohmig	-
1	0	niederohmig	0
1	1	niederohmig	1

Schaltsymbol **CMOS Schaltung** **Wahrheitstabelle**

LTD: Understand why it works like that, the GS voltage doesn't always make sense...

Boolean Algebra

The following properties apply to an expresion only containing AND / OR gates:

- Commutative, order does not matter
- Associative, grouping / order of (the same) operations is irrelevant
- Distributive, $A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$

Some not so obvious axioms of boolena algebra:

- $A \vee (A \wedge B) = A = A \wedge (A \vee B)$ - consider the effect on the whole circuit when the outer variable is high / low
- $(A \wedge B) \vee (\neg A \wedge B) = (A \vee B) \wedge (\neg A \vee B) = B$ - Neighbourhood law

Order of Operations

1. Brackets
2. Negation
3. AND, NAND, OR, NOR
4. XOR, XNOR

An expression with missing brackets is ambiguous and invalid.

De Morgan's Laws

$$\neg(A \wedge B \wedge C \wedge ...) \equiv \neg A \vee \neg B \vee \neg C \vee ...$$
$$\neg(A \vee B \vee C \vee ...) \equiv \neg A \wedge \neg B \wedge \neg C \wedge ...$$

The conversion between the pull up and pull down expression in a CMOS circuit uses De Morgan's laws:

$$\mathbf{Y_{pd}} = \overline{((\mathrm{A} \cdot (\mathrm{B} + \mathrm{C}) \cdot \mathrm{D}) + \mathrm{F} + (\mathrm{G} \cdot \mathrm{H})) \cdot \mathrm{E}}$$
$$\mathbf{Y_{pu}} = ((\overline{\mathrm{A}} + (\overline{\mathrm{B}} \cdot \overline{\mathrm{C}}) + \overline{\mathrm{D}}) \cdot \overline{\mathrm{F}} \cdot (\overline{\mathrm{G}} + \overline{\mathrm{H}})) + \overline{\mathrm{E}}$$

Universal Gates

Any logic circuit can be expressed using only NAND / NOR gates. This is very advantageous as all gates in the circuit would have the same timing properties, reducing costs and errors.

To convert a logical expression into NAND / NOR, double negation + De Morgan's laws can be used, for example:

$$A \wedge B \equiv ?$$
$$\neg\neg A \wedge B \equiv \neg\neg A \vee \neg B$$
$$\neg A \text{ NOR } \neg B \equiv (A \text{ NOR } A) \text{ NOR } (B \text{ NOR } B)$$

Min / Maxterms

For n variables there are 2^n possible min / maxterms:

- Minterm: Expression that outputs 1 for **one** specific combination of inputs, if we want 1 only when A low, B High, minterm: $\neg A \wedge B$
- Maxterm: Expression that outputs 0 for **one** specific combination of inputs, if we want 0 only when

Let us consider we want expressions that output high / low only when $A = 0, B = 1$:

- Minterm: $\neg A \wedge B$, high only in this case
- Maxterm: $A \vee \neg B$, low only in this case

Normal Forms

A way of expressing a boolean expression that can easily be determined from the desired truth table (and then simplified using boolean algebra / Karnaugh diagrams). After constructing either a list of minterms for each 1 in the output, or list of maxterms for each 0:

- Disjunctive Normal Form - Minterms (ANDed variables) joined using disjunctions (OR)
- Conjunctive Normal Form - Maxterms (ORed negations) joined using conjunctions (AND)

Both result in the same output.

Karnaugh Diagrams

Used to visually and systematically simplify boolean expressions instead of through often complicated boolean algebra manipulation. Furthermore, race conditions (hazards) can be easier identified using this method.

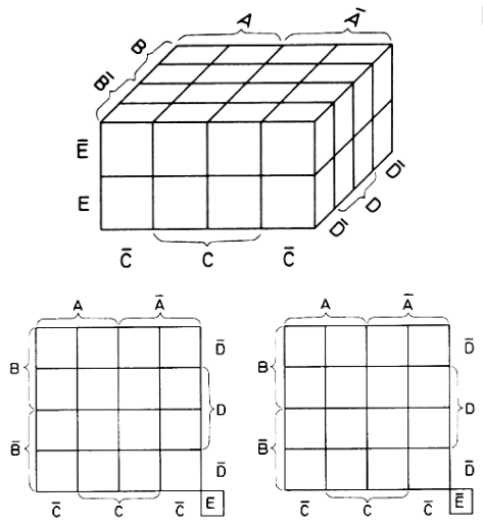
They are analogous to a truth table but represented as a matrix with 2^n elements, where n is the number of inputs.

Simply a methodical way of using the neighbour simplification rule: $(\neg A \wedge \neg B) \vee (A \wedge \neg B) = \neg B$

Don't care - Combinations of inputs for which the output doesn't matter, for example additional unneeded numbers in a boolean counting system. Marked with an X in a Karnaugh Diagram. The X's can be treated as 1s or 0s when creating packets if it reduces the amount of packets (and therefore joining gates) in the simplified expression.

Method

- Construct a matrix with 2^n elements, where each side of the matrix represents the two states of a variable. One of each pair of variables facing opposite one another must be "split" so the neighbour rule can still be applied (See examples).
- Each element contains a 1 for each minterm (DNF) and the rest 0, or 0 for each maxterm (in this case the element headers are negated when allocating) and the rest as 1. *Don't Care* conditions can be written as X when the output for a specific combination doesn't matter.
- Create packets (also known as blocks) using the largest possible rectangle with 1s / 0s (may include Xs as they fit best). The packets must contain 2^n cells!
- Packets may overlap, "pacman" over the border and pass through layers, but not take non rectangle shapes (for example an L shape) or be diagonal. It is better to err on the side of caution and choose less packets, which may be possible to simplify later through linear algebra.
- They must capture the entire state of the diagram - Either all the 1s or all the 0s. Xs of course don't all need to be included.
- Each packet represents a minterm (if it contains 1s) or maxterm (contains 0s), which can then be simplified to the variable(s) which remain constant in the packet ADDed / ORed together (depending on if min or maxterm).
- The result of each packet can then be combined as the DNF / KNF.
- **IMPORTANT**: When combining results of the packets, they must all represent the same type (min or maxterm). If it's easier to formulate different types of packets, min and maxterms can of course be converted between another.



Static hazards - When the same variable is used in a parent logic gate, changes in the variable can lead to delayed "notches" in the parent's output due to time delays. These can be recognized in Karnaugh diagrams: where two packets are orthogonally next to each other but do not overlap. They can be directly fixed by introducing an extra packet to join the place of the hazard - this results in more gates overall but avoids the hazard. IMPORTANT: This packet should also be as large as possible.

TODO: Finish exercise series

Number Systems

Base (Radix) - b-adischen Reiehen like in analysis, negative indices of the base for defining decimals

Decimal - Base 10, sometimes written with a $ at the start.

Hexadecimal - Uses digits $0 - 9$ and $A - F$ for 16 possible digits in total. Often written with 0x at start. Used to represent binary numbers in a more compact format by splitting a binary number into groups of 4 digits

Octal - Radix 8, can be converted from binary using groups of 3 binary digits.

Converting Decimal to Radix R

1. Perform whole number division of the decimal D by the Radix R: $\frac{D}{R} = Q_0 + r_0$, the remainder is the first digit in the target radix
2. Divide the result of the previous whole-number division Q_0 by the radix R again, this remainder is now the second digit in the target radix and so on
3. Continue until Q_i reaches 0

$\boldsymbol{D = 243_{(10)}}$ als Dualzahl, also Basis $R = 2$

Operation	Quotient	Rest	Bemerkung
$^{D}/_{R} = 243/2$	$Q_0 = 121$	$r_0 = 1$	Least Significant Bit (LSB)
$^{Q_0}/_{R} = 121/2$	$Q_1 = 60$	$r_1 = 1$	
$^{Q_1}/_{R} = 60/2$	$Q_2 = 30$	$r_2 = 0$	
$^{Q_2}/_{R} = 30/2$	$Q_3 = 15$	$r_3 = 0$	
$^{Q_3}/_{R} = 15/2$	$Q_4 = 7$	$r_4 = 1$	
$^{Q_4}/_{R} = 7/2$	$Q_5 = 3$	$r_5 = 1$	
$^{Q_5}/_{R} = 3/2$	$Q_6 = 1$	$r_6 = 1$	
$^{Q_6}/_{R} = 1/2$	$Q_7 = 0$	$r_7 = 1$	Most Significant Bit (MSB)

Also $243_{(10)} = 1111\ 0011_{(2)} = r_7 r_6 r_5 r_4 r_3 r_2 r_1 r_0$

Converting $0 \leq D_{10} < 1$ to Radix R

This is a similar process but the decimal is multiplied by the radix R and the floor of the resulting product is the current digit's value. The remainder after subtracting the floor from the current result is the next calculation's input.

$\boldsymbol{D = 0.171875_{(10)}}$ als Dualzahl, also Basis $R = 2$

Operation	Produkt	Koeffizient	Bemerkung
$D \cdot R = 0.171875 \cdot 2$	$P_0 = 0.34375$	$K_{-1} = \text{floor}(P_0) = 0$ $(a_{-1} = P_0 - K_{-1})$	Most Significant Bit (MSB)
$a_{-1} \cdot R = 0.34375 \cdot 2$	$P_{-1} = 0.6875$	$K_{-2} = \text{floor}(P_{-1}) = 0$ $(a_{-2} = P_{-1} - K_{-2})$	
$a_{-2} \cdot R = 0.6875 \cdot 2$	$P_{-2} = 1.375$	$K_{-3} = \text{floor}(P_{-2}) = 1$ $(a_{-3} = P_{-2} - K_{-3})$	
$a_{-3} \cdot R = 0.375 \cdot 2$	$P_{-3} = 0.75$	$K_{-4} = \text{floor}(P_{-3}) = 0$ $(a_{-4} = P_{-3} - K_{-4})$	
$a_{-4} \cdot R = 0.75 \cdot 2$	$P_{-4} = 1.5$	$K_{-5} = \text{floor}(P_{-4}) = 1$ $(a_{-5} = P_{-4} - K_{-5})$	
$a_{-5} \cdot R = 0.5 \cdot 2$	$P_{-5} = 1.0$	$K_{-6} = \text{floor}(P_{-5}) = 1$ $(a_{-6} = P_{-5} - K_{-6} = \mathbf{0})$	Least Significant Bit (LSB)

Also $0.171875_{(10)} = 0.001011_{(2)} = 0.K_{-1}K_{-2}K_{-3}K_{-4}K_{-5}K_{-6}$

⚠ Nicht jede rationale Dezimalzahl ist exakt mit **endlicher** Stellenanzahl in einem anderen Zahlensystem darstellbar

Signed Binary Numbers

There are several possible ways to represent signed binary numbers:

- Signing bit
- 1s complement
- 2s complement: Advantageous because addition works using it, making it viable for subtraction too

2s Complement

A 2s complement binary number can be converted back and forth using the following method:

1. Set apart the leading signing bit
2. Perform bitwise inversion on the other bits and add 1
3. Append the inverted leading signing bit (Do not cut off leading 0s that arose from inversion!)

Converting 2s complement to decimal is also straightforward. Either the corresponding positive binary number can be converted or:

1. Treat the leading bit as −1 * its place value
2. Add the following binary number as positive to the negative number (ex. −256 + 16 + 2 + 1)

Fractional numbers still have $\pm 2^0$ as the signing bit, they can be converted in the same way. In the case of a whole + fractional number only the first bit is the signing bit.

IMPORTANT: Do not forget signing bit for positive numbers

The mQn notation simply tells us how many bits before (m) and after the decimal place (n) represent the amount.

Binary Arithmetic

Addition of two binary numbers, with maximum n digits has at most $n + 1$ bits in the result.

Addition of 4x1s = Carry over 1 two places. This applies in general to all carry bits, because the entire addition result is simply being “overlayed” on the existing sum

Binary subtraction can be written as the addition of 2s complement numbers. It will simply work and the MSB will be an accurate signing bit. IMPORTANT: Do not forget +‘ve number signing bit!

Encoding

Tetrad / Nibble - groups of 4 bits

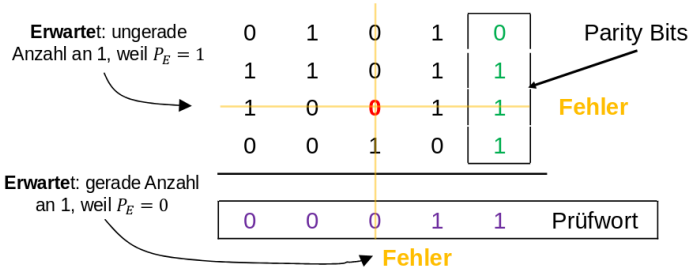
Many different ways to encode 10 numbers, each has their advantages / disadvantages. For example, Gray / O’Brien encoding is useful for counting, because they are assigned in such a way that only 1 bit changes per increment.

Parity Bit

Additional bit representing if the number of 1s in a word / block is odd / even

An extra word containg parity bits of the columns when the previous words are arranged as a matrix can be sent with the purpose of error checking, and when used in combination with word parity bits it can be used to pinpoint and correct errors.

Fehlererkennung und –korrektur:

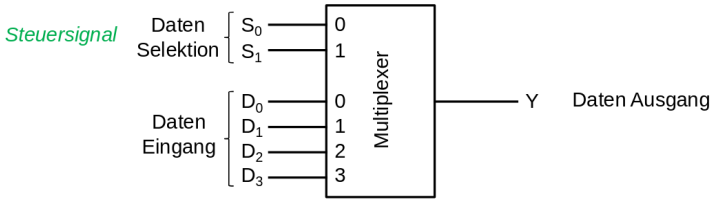


Combinatronic Circuits

Combinatronic (aka Datenpfad) circuits create a pure output that solely depends on the current inputs.

Multiplexer

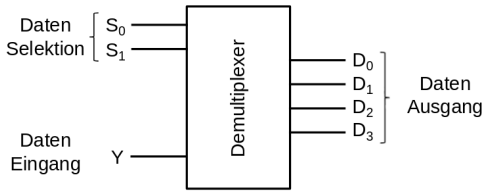
Outputs one selected bit from several inputs using a (in this case 2 digit) binary selection signal. Circuit is simply a set of minterms $(\neg S_0 \wedge \neg S_1 \wedge D_0) \vee (\neg S_0 \wedge S_1 \wedge D_1) \vee (S_0 \wedge \neg S_1 \wedge D_2) \vee (S_0 \wedge S_1 \wedge D_3)$



Demultiplexer

Inverse of a multiplexer, selects at which output a signal is outputed:

Beispiel: 1-zu-4 Demultiplexer



Die Ausgänge können zum Beispiel so programmiert werden:

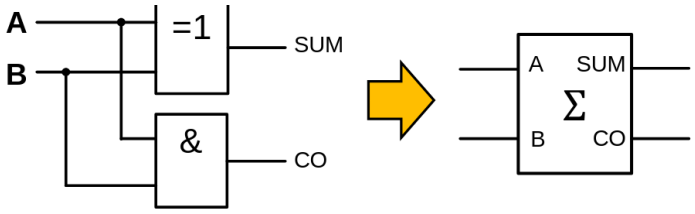
$$D_0 = \overline{S_0} \wedge \overline{S_1} \wedge Y, \qquad D_1 = \overline{S_0} \wedge S_1 \wedge Y,$$
$$D_2 = S_0 \wedge \overline{S_1} \wedge Y, \qquad D_3 = S_0 \wedge S_1 \wedge Y$$

Code Translator (Umsetzer)

Converts between number encoding. Create KNF -> Karnaugh diagram for each output and the set of inputs, regular circuit synthesis procedure.

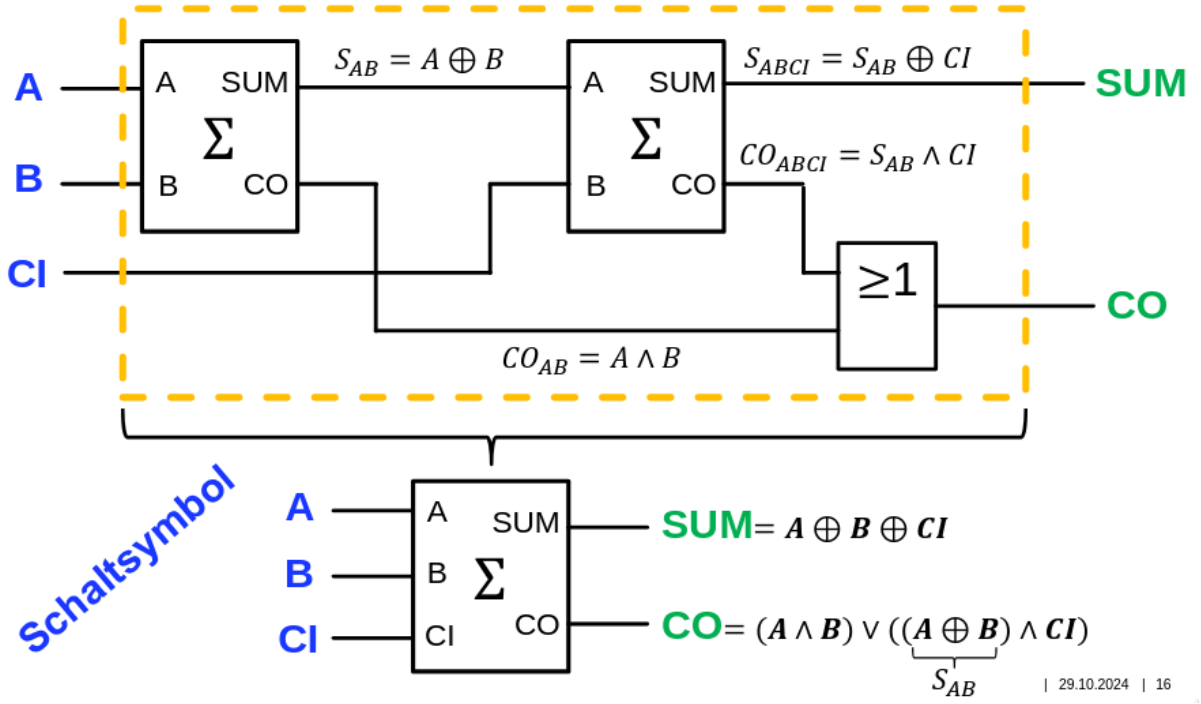
Half Adder

Outputs the sum of two binary digits and the remainder (Carry Out CO) to be passed to an adjacent full-adder one place value higher.



Full Adder

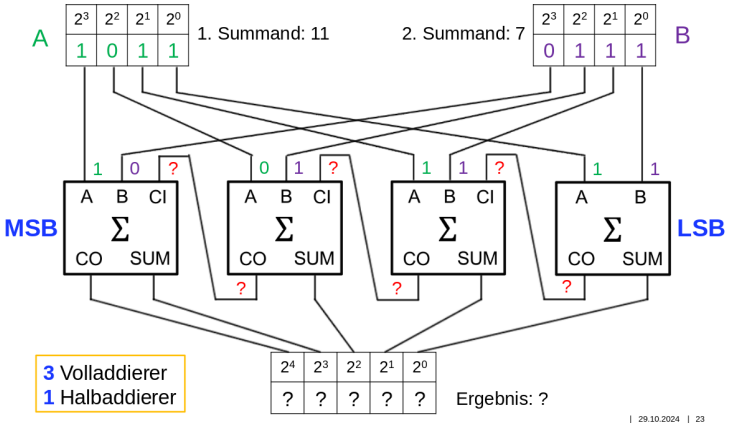
3 inputs: A, B and Carry In (CI) for a lower CO bit. Simply a combination of two half adders plus an OR gate taking in CI and CO of the internal half adder.



Parallel Adder

Addition of 2 multiple digit binary numbers can be synthesised as usual using Karnaugh diagrams. This is the most efficient multi digit adder but is very application specific and needs a lot of work to extend.

Ripple Carry



Advantage: Easy to extend and combine

Disadvantage: Carry bits take time to ripple up the place values, the time needed to get a correct output increases linearly

Carry-Look-Ahead Adder

Due to the fact that adders are so central to more complex electronic circuits, it is very useful to combine the delay and extendability advantages of both the parallel and ripple carry adder into one.

This is based on propagating carry bits as soon as possible, as these do not depend on an accurate sum. A full adder will output high on CO if:

- Both of the inputs are high
- One of the inputs is high and CI is high

Although this leads to slightly more complex hardware, the sums can therefore also be calculated in parallel and the circuit remains extendable.

The carry logic can be calculated using this recursive formula:

$$C_i = (A_i \wedge B_i) \vee \big((A_i \oplus B_i) \wedge C_{i-1}\big) = G_i \vee (P_i \wedge C_{i-1})$$

Subtraction

As mentioned before, this is straightforward using addition of 2s complement signed numbers.

Conversion to 2s complement can alternatively be done using XOR Gates instead of NOT for the inversion step, allowing us to control if the current amount should be subtracted (inverted) or added:

$B_{alt,i}$	S	$B_{neu,i} = B_{alt,i} \oplus S$	
0	0	0	Passiert nichts
1	0	1	
0	1	1	Bit Inversion
1	1	0	

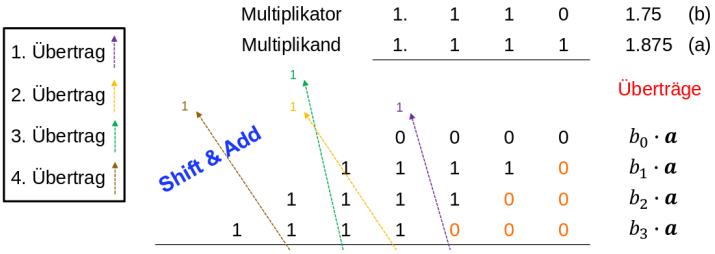
Mit **S=0** wird eine **normale Addition** durchgeführt, der 2. Summand B bleibt unverändert, da $B_{neu,i} = B_{alt,i} \oplus 0 = B_{alt,i}$

Mit **S=1** wird eine **Subtraktion** durchgeführt. Das **2er-Komplement** vom 2. Summanden B wird gerechnet, indem $B_{alt,i} \oplus 1 = \overline{B_{alt,i}}$ verwendet wird und der CI Eingang des ersten Volladdierers auf S=1 gesetzt wird. Damit wird eine 1 an der LSB Stelle vom invertierten 2. Summanden addiert

Multiplication

Multiplication occurs in the same way for any radix, through shifting and addition:

Problem zu lösen: $1.110_{(2)} \times 1.111_{(2)} = b_3b_2b_1b_0 \times a_3a_2a_1a_0$



The shifting logic can be built by simply wiring several adders (Basiszellen) together or using a single adder and multiplexer + counter (Booth's Algorithm).

Sequential Circuits

Sequential circuits depend not only on the current inputs but also the previous state ie a function with two sets of inputs. They incorporate memory through back connections (Rückkopplungen).

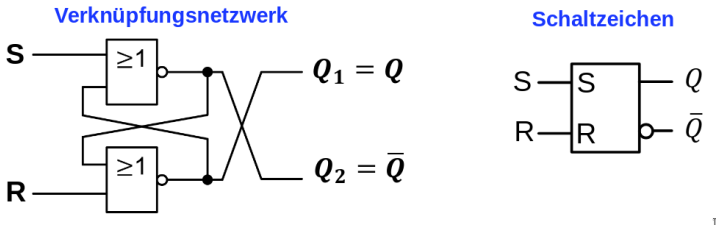
Truth tables / Karnaugh diagrams for such circuits can be found by including previous states Q_n as an input variable and calculating resulting output states Q_{n+1}. The back connection has effectively been cut and replaced with an additional input.

Latches

These are components that store a single bit based on signal levels (Zustandsgesteuert) instead of signal transitions.

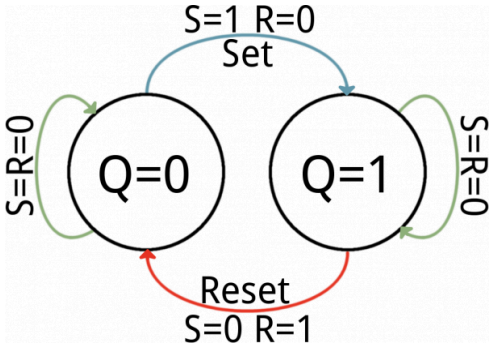
SR-Latch

This is the most basic latch, storing a single bit which can be controlled using the Set and Reset inputs.

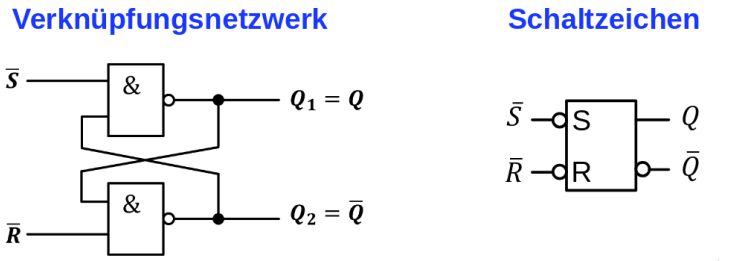


TODO: Diagrams of both variants

When set is high, the output Q instantly becomes high. When reset goes high, Q instantly becomes low (the memory is reset). If they are both low, the current saved state is output. S and R high is an invalid input (can be written as a Don't Care state) and results in unstable behaviour.



Alternatively, it can be built using NAND gates, using inverted inputs instead. This is useful when the inputs are off by default, for example a switch.



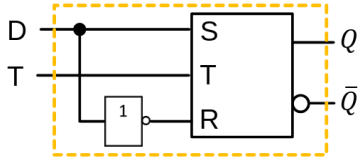
Clock-controlled Latch

This has the same behaviour as a regular SR-Latch, but the S and R pins only take effect when T is high (also known as a gate controlled latch), simply by introducing two AND gates between S+T and

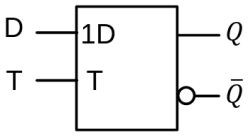
R+T. This is useful to only allow changes during a high clock pulse and it is the basic building block of flip flops later (however unlike flip-flops, its memory can be changed anytime whilst T is high).

D-Latch

Co-ordinating two separate set / reset signals can be tricky, so this latch allows us to control the new desired state using a single Data pin. Of course, if the clock signal were always 1, this simply mirrors the state of D, therefore it is designed for storing during the off clock period.



Basisschaltung D-Latch

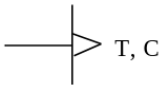


Schaltzeichen D-Latch

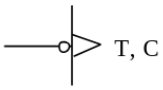
Flip-Flops

These are similar to latches but they only store the value during a rising / falling edge of a clock signal (Taktflankengesteuert), protecting against undesired signal errors during the storing state of the cycle.

Schaltzeichen für die Taktflankensteuerung



Eingangsvariable werden beim
0 - 1 Übergang von C wirksam



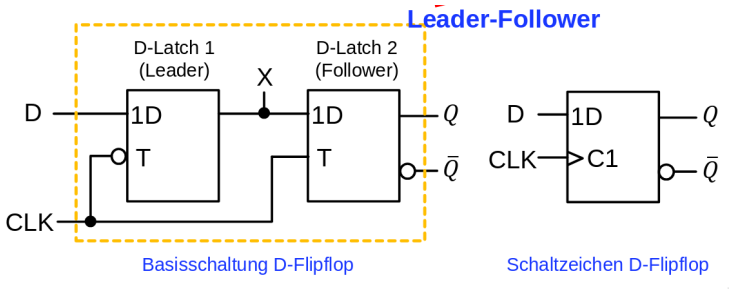
Eingangsvariable werden beim
1 - 0 Übergang von C wirksam

Sometimes a reset pin is also built in, allowing us to interrupt the signal and instantly reset regardless of the current clock state.

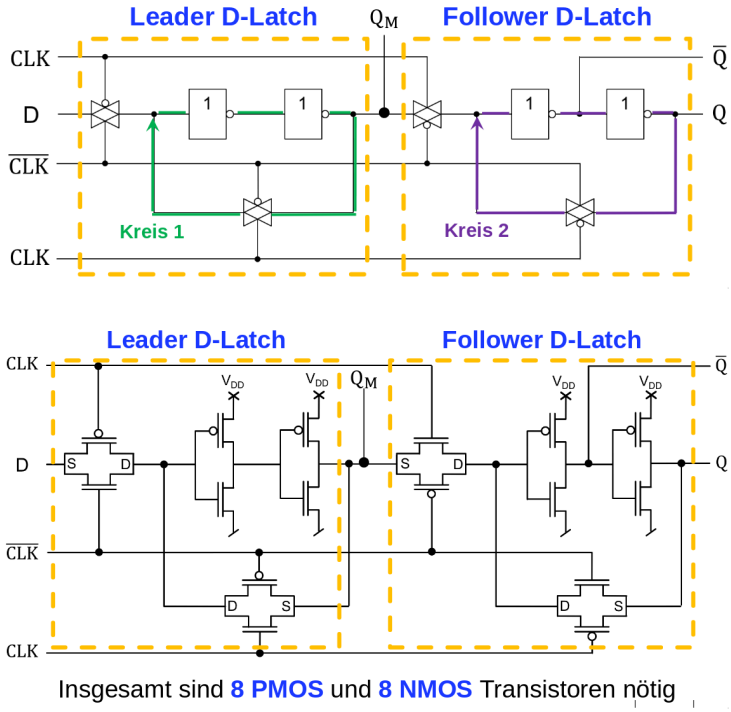
D-Flipflop

This can be achieved using a leader-follower structure of two internal latches operating on an inverted clock, whether the leader or follower recieves the inversion chooses if its a rising or falling edge operated flipflop.

In the example below, the first latch is set whilst the clock is low, then the second latch accepts this saved value as the clock rises. This is a safer design as the connection between the 1st and 2nd latch is safe from external hazards.



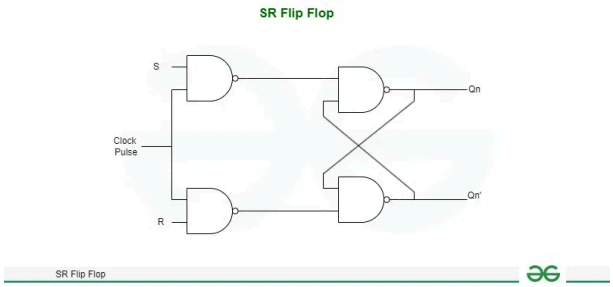
They are most commonly used in practice as they can be constructed using very few transistors in CMOS technology, where the transmission gates control if the data can progress through each step:



LTD: The double NOT gates are needed for some kind of stability rather than a short circuit, understand exactly why

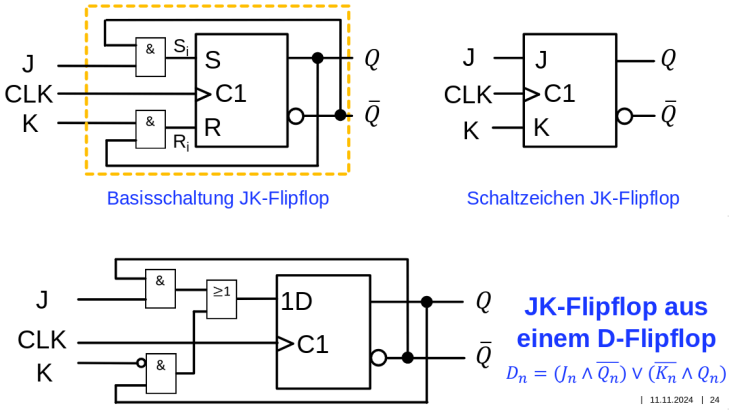
SR-Flipflop

These are rarely used in practice but are the same concept - the current state of S and R only takes effect during a rising / falling clock.



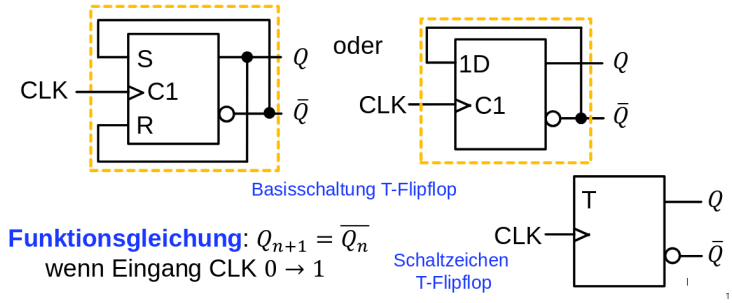
JK-Flipflop

The Jump and Kill pins function as the S and R pins, however both are allowed to be high, which simply toggles the output.

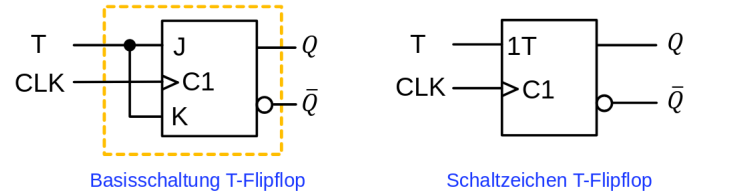


T-Flipflop

This simply toggles the output at every rising edge.



Alternatively the toggling functionality can be turned on or off using a JK-flipflop.

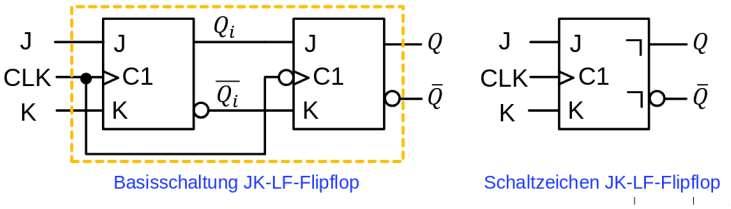


Funktionsgleichung: $Q_{n+1} = \overline{Q_n}$ wenn CLK 0 → 1 und T=1

Sonst wird das aktuelle Ausgangssignal gespeichert

Delayed-Flipflop

This consists of 2 combined flipflops, a change taken in during a rising signal is only outputed after the corresponding falling signal and vice versa.

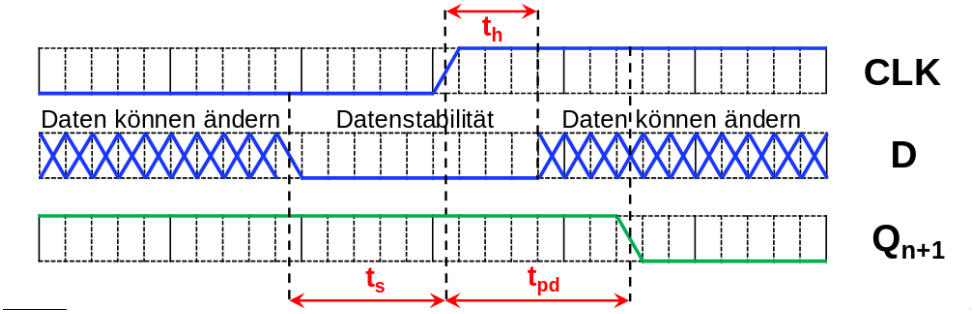


Timing

The delays for a given flipflop must be respected to ensure correct operation:

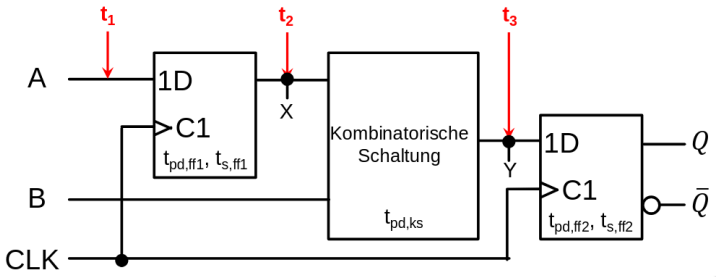
- t_{pd} - Propagation delay - time until a succesful save appears at the output, starts at 50% of the clock edge, ends when 50% of the new output is reached (if a change occurs).

- t_s - Setup time - How long the input signal must stay constant before the edge, so that it's guaranteed to have been taken in by the leader latch.
- t_h - Hold time - How long the input must stay constant after the edge to ensure the follower latch accepts it.



These delays can be combined to analyse the minimum clock period (and therefore maximum clock frequency; the fastest rate at which we are allowed to operate a computer) of a complex circuit involving several flip flops and combinatronic circuits.

For instance in the circuit below, the minimal clock period must be $t_{pdff1} + t_{pdks} + t_{pdff2}$ to ensure a signal can make it through the circuit before the clock signal changes:



Of course if a circuit contains several branches, the branch with the longest delay represents hte minimum clock period.

Applications

Apart from stable 1-bit storage, flipflops can be useful building blocks for a variety of functional components.

Frequency Divider

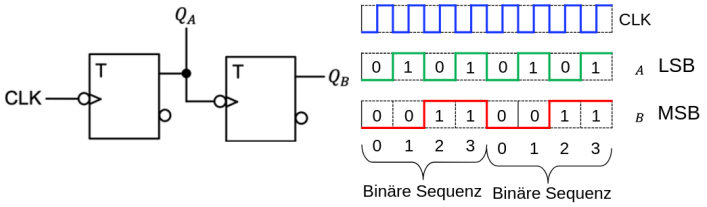
Chained T-flipflops are useful to reduce the clock frequency, each flipflop reduces it by a factor of two, hence:

$$f_{\text{Target}} = \frac{f_{\text{Clock}}}{2^n}$$

$$n = \log_2\left(\frac{f_{\text{Clock}}}{f_{\text{Target}}}\right)$$

Counter

The intermittent and final output state can also be used as a 2-digit binary counter:



] LTD: Can this extended to more than 2 digits?

Automata

Mealy vs moore mealy can be used to reduce number of states needed - previous n + latest input considered for example binary pattern recognizer binary encoding to reduce number of flip flops needed

Moore automata more stable, outputs are accurate regarding inputs, synchronisation between clock and inputs needed, use flipflops.

sequence detecter can be made with only 3 flip flops and binary encoding, or simply a series of 7 flipflops and a series of (notted) inputs to an and gate

automata can be connected with each other to abstract separate functions

cpus are infinite state machines, opcodes have arbitray operands and modern memory holds a huge amount of states.