EF Informatik / *Elektronik und Digitaltechnik*

Vom Transistor zur CPU

Teil 1 – Eine einfache Central Processing Unit

Die **C**entral **P**rocessing **U**nit, auch „Prozessor“ genannt, ist das Herzstück jedes Computers. Moderne Computer verfügen bereits über mehrere CPUs oder enthalten Prozessoren mit mehreren Kernen (*Multicore-Prozessoren*). Ein Kern ist sozusagen eine CPU innerhalb einer CPU. Bild 1 zeigt das Blockschaltbild einer sehr einfachen 8-Bit-CPU. Es besteht aus mehreren Modulen, die über

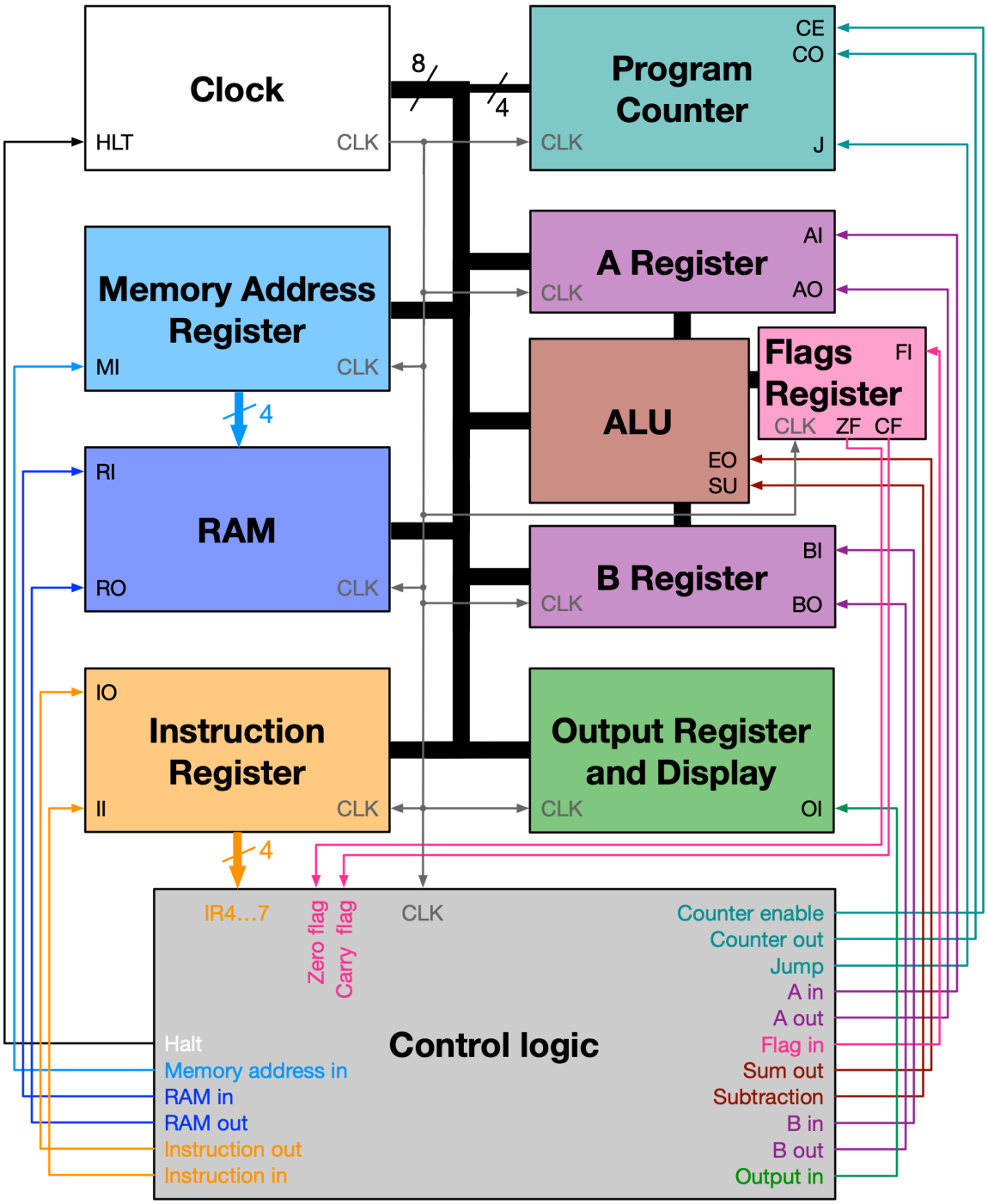


Bild 1 – Blockschaltbild der 8-Bit-CPU nach dem Design von Ben Eater.

Leitungen miteinander verbunden sind. Dieses CPU-Design stammt von Ben Eater. Auf seiner Webseite eater.net/8bit findest du Videos und Beschreibungen, die den Aufbau und die Funktionsweise jedes Moduls ausführlich erklären. Für eine Übersicht betrachten wir kurz die wichtigsten Module und ihre Aufgaben:

Der Bus besteht aus 8 parallelen Leitung, die in Bild 1 als eine dicke Linie gezeichnet sind. Der Bus ist quasi die Daten-Autobahn: über ihn tauschen die Module Daten aus. Es können gleichzeitig **8 Bits** übertragen werden. „Bit" steht für „**Bi**nary Digi**t**" (etwa: „binäre Ziffer"). Ein Bit kann zwei Werte haben; 0 oder 1.

Wert	Bezeichnungen	Spannung in 5-V-Logik
0	LOW, false	0 V
1	HIGH, true	5 V

Angenommen, die acht Bus-Leitungen haben die Werte 0, 1, 0, 1, 0, 1, 0, 1, so würden wir abwechslungsweise die Spannungen 0 V und 5 V messen. Zusammen bilden die Leitungen die binäre Zahl 0101010101. Das entspricht der Zahl 42, doch dazu später.

Das **Clock**-Modul generiert ein Rechteck-Signal, das dazu dient, die Module zeitlich zu steuern:

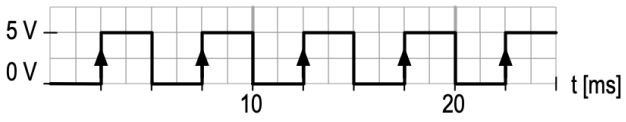


Bild 2 – Clock-Signal

Das Clock-Signal in Bild 2 wechselt alle 2.5 Millisekunden zwischen 0 V und 5 V. Alle 5 Millisekunden erfolgt eine positive Flanke, also ein Wechsel von 0 auf 5 V – hier durch die Pfeile bezeichnet. Wenn ein Modul eine Anweisung ausführt (wenn es zum Beispiel Daten auf den Bus schreibt), dann tut es das exakt zum Zeitpunkt der positiven Flanke des Clock-Signals. So wird erreicht, dass alle Anweisungen zeitlich gesteuert und damit in der richtigen Reihenfolge ausgeführt werden. Die Clock gibt den Takt an. Die Taktfrequenz in Bild 2 beträgt 1/0.005 s = 200 Hz. Eine CPU mit einer Taktfrequenz von 200 Hz kann 200 Anweisungen pro Sekunde ausführen – alle 5 ms eine Anweisung.

Der **Program Counter** ist ein einfacher Zähler. Er sendet über den Bus eine vierstellige Binärzahl, die er stets um 1 erhöht. Was passiert mit dieser Zahl?

Wenn du am Computer ein Programm schreibst – zum Beispiel in *python* oder in *C* – dann wird dieser Code irgendwann übersetzt in sogenannten *Maschinencode*. Dieser enthält eine Reihe von Befehlen und besteht nur aus *Einsen* und *Nullen*. Die Zahl vom Program Counter gibt die Position des nächsten Befehls an. Wenn der Counter hoch zählt, wird ein Befehl nach dem anderen ausgeführt. Es ist auch möglich, den Counter auf eine bestimmte Zahl zu setzen, um auf eine bestimmte Stelle im Programm zurück- oder vorzuspringen. Dadurch werden Schleifen (z. B. *for)* oder Verzweigungen (z. B. *if-then-else*) möglich.

Das **RAM** (Random Access Memory) speichert Daten – unter anderem den Maschinencode, der ausgeführt werden soll. Bei unserer CPU beträgt die Kapazität des Speichers 16 Byte. Ein Byte enthält 8 Bits. Wir können uns ein RAM als Tabelle vorstellen – hier mit 16 Zeilen und 8 Spalten:

	Byte							
Adresse	**Bit0**	**Bit1**	**Bit2**	**Bit3**	**Bit4**	**Bit5**	**Bit6**	**Bit7**
'0000'	0	0	0	0	0	0	0	0
'0001'	0	0	0	0	0	0	0	0
'0010'	0	0	0	0	0	0	0	0
…	…	…	…	…	…	…	…	…
'1111'	0	0	0	0	0	0	0	0

Oben sind die ersten drei und die letzte Zeile dargestellt. Jede Zeile hat eine eigene Adresse, von

0 (binär: 0000) bis 15 (binär 1111); unter jeder Adresse können 8 Bits oder eben 1 Byte gespeichert werden. Im Moment ist unter jeder Adresse der Wert 0000'0000 abgespeichert.

Das **Memory address register** speichert die vierstellige RAM-Adresse und gibt diese an das RAM weiter.

Die **Register A und B** dienen als kleine Zwischenspeicher. Sie können einen Wert vom Bus speichern und später an die ALU weitergeben oder wieder auf den Bus zurückschreiben. In grösseren Prozessoren werden solche Register auch „general purpose registers", also etwa „Mehrzweck-Register" genannt, da sie unterschiedlichen Zwecken dienen können.

Die **ALU** (Arithmetic logic unit) ist das Rechenwerk der CPU. Unsere CPU kann aber nur dies: Die Zahlen aus den Register A und B entweder addieren oder subtrahieren und das Resultat auf den Bus schreiben – das ist alles.

Das **Flags Register** speichert Flags. Ein Flag ist meistens ein 1-Bit-Speicher und kann entweder den Wert 1 oder 0 haben. Unsere CPU verfügt über zwei Flags, das *Zero-Flag* und das *Carry-Flag*. Das Zero-Flag zeigt an, ob das Ergebnis der Rechenoperation der ALU = 0 ist. Wenn das Ergebnis 0 ist, dann ist das Zero-Flag 1; in allen anderen Fällen ist es 0. Das Carry-Flag zeigt an, ob die Rechenoperation einen Übertrag ergab.

Das **Instruction register** speichert den Befehlscode für den Befehl, der als nächstes ausgeführt werden soll. In unserer CPU besteht ein Befehlscode aus einer vierstelligen Binärzahl. Wir haben bereits gesehen, dass mit vier Binärstellen maximal 16 Zustände möglich sind. Das heisst, unsere CPU kann maximal 16 verschiedene Befehle ausführen. Der vierstellige Befehlscode wird zur Control logic weitergeleitet.

Die **Control logic** führt die Befehle aus. Ein Befehl (engl.: *instruction*) steht für eine Reihe von Anweisungen (engl.: *micro instructions*). Das funktioniert etwa so:

Angenommen, der Code, der vom Instruction register kommt, lautet: 0011. Dieser Code steht zum Beispiel für den Befehl „Schreibe das Resultat aus der ALU ins RAM". Nun muss die Control Logic aufgrund dieses Codes einige Anweisungen ausführen: Sie muss zuerst die eigene Instruktion laden, dann muss sie dem Memory address register mitteilen, dass es die Adresse vom Bus lesen muss, welche bestimmt, wo im RAM das Resultat abgespeichert wird. Danach muss sie der ALU mitteilen, dass sie das Resultat auf den Bus schreiben soll und schliesslich muss sie dem RAM sagen, dass es dieses Resultat vom Bus lesen soll. Die Control Logic kann also allen Modulen sagen, was sie tun und lassen sollen – und das tut sie über die Steuerleitungen, die du im Bild 1 siehst.

Das **Output register** schliesslich speichert den Wert, der am Display angezeigt werden soll. Aus Sicht des Menschen ist das sehr wichtig, denn all das Herumschieben und Rechnen in der CPU wäre für uns sinnlos, wenn nicht irgendwann etwas angezeigt würde – wir könnten damit so viel tun wie mit einem Computer ohne Bildschirm.

Was unserer CPU aber fehlt, was sonst alle Computer haben, ist ein Input-Modul, das es erlaubt, auch Daten einzugeben – etwa mit einer Tastatur. Es wäre bei der bestehenden Architektur aber einfach, noch ein Input-Modul an den Bus anzuschliessen und die Control-Logic um Steuerleitungen für dieses Modul zu erweitern.

Im **Vergleich mit modernen Prozessoren** fehlt unserer CPU aber einiges mehr, zum Beispiel:

- Der Aufbau moderner Prozessoren ist sehr viel komplexer als die in Bild 1 gezeigte. Sie verfügen über ein Vielfaches an Modulen und Registern.
- Die Busbreite in heutigen Prozessoren beträgt bis zu 64 Bit. Viele Prozessoren verfügen über getrennte Bussysteme für Daten und Adressen. Also beispielsweise ein 64-Bit-Datenbus und ein 40-Bit-Adressbus.

Bild 3 – Fotografie eines Intel-Pentium-4-Chips aus dem Jahr 2000. Der Chip ist etwa so gross wie ein 10-Rappen-Stück. Hier die Beschriftung der Bereiche.

- Die Taktfrequenz moderner Prozessoren liegt bei 2 bis 5 GHz. Deren Clock tickt also etwa 10 Millionen mal schneller als unsere Clock.
- Der Arbeitsspeicher (RAM), der modernen Prozessoren zur Verfügung steht, liegt im zwei-bis dreistelligen Gigabyte-Bereich und ist damit milliardenmal höher als bei unserer CPU. Prozessoren in heutigen Computern können ausserdem auf Festspeicher (FLASH etc.) zugreifen, auf denen das Betriebssystem und andere Programme permanent gespeichert sind. Das RAM unserer CPU ist kein Festspeicher, das heisst, dass sowohl das Programm als auch alle Daten gelöscht sind, sobald die Stromversorgung wegfällt. Bei nur 16 Bytes ist das nicht schlimm, die sind schnell wieder reingehackt.
- Die ALUs in modernen Prozessoren beherrschen nebst der Addition und Subtraktion auch Multiplikation, Division und einige logische Operationen – und das mit sehr viel grösseren bzw. komplexeren Zahlen als bei der 8-Bit-CPU, die nur mit Zahlen von 0 bis 255 rechnen kann.

Die grundlegende Funktionsweise und Architektur moderner Prozessoren sind aber nicht anders als bei unserer 8-Bit-CPU. Das heisst: Wenn du verstehst, wie diese 8-Bit-CPU funktioniert, dann hast du gute Voraussetzungen, heutige Prozessoren zu verstehen.

Ein Vorteil der 8-Bit-CPU liegt darin, dass wir sie selbst zusammenbauen können: Dazu brauchen wir Breadboards (die weissen, löchrigen Platten), einige integrierten Schaltungen (so genannte „ICs“, die schwarzen Bausteine), Drähte etc.

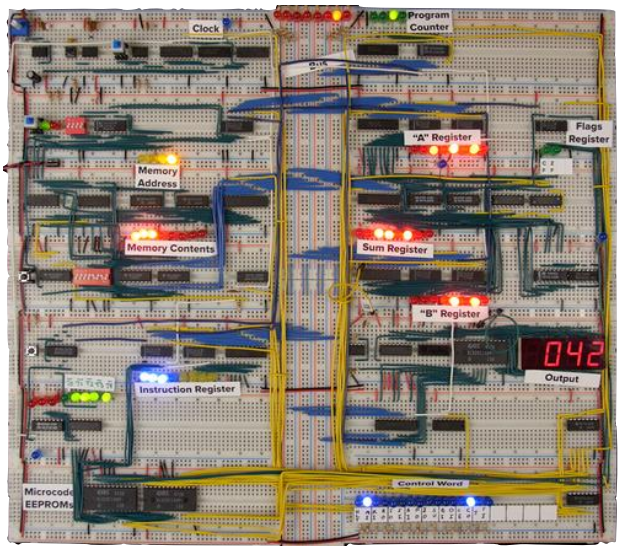


Bild 4 – Fotografie der aufgebauten 8-Bit-CPU. Der Aufbau 40 x 40 cm gross.

In diesem Kurs betrachten wir Aufbau und Funktionsweise der wichtigsten Module der 8-Bit-CPU. Zunächst aber schauen wir den einen, elementaren Baustein an, der in einer heutigen CPU milliardenfach verbaut ist und ohne den die heutige Digitaltechnik unmöglich wäre. Es handelt sich um den **Transistor**.

Teil 2 – Der Transistor

Ein Transistor ist ein Halbleiter-Bauteil mit drei Anschlüssen.Transistoren gibt es in vielen Grössen und Bauformen, hier eine kleine Auswahl:

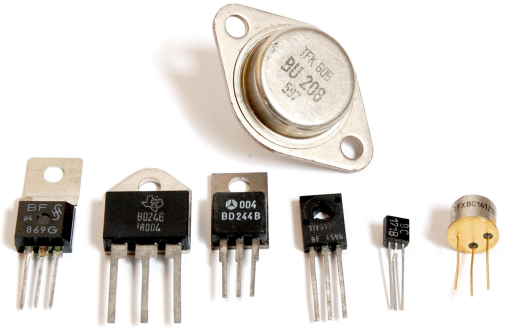


Bild 5 – Transistoren in unterschiedlichen Bauformen.

Dieses Video gibt dir eine Übersicht über den Transistor. Transistoren werden hauptsächlich als **Verstärker** und als **Schalter** verwendet. Sie haben drei Anschlüsse und erlauben es, mit einem kleinem Strom (oder mit einer kleinen Spannung) einen viel grösseren Strom (eine viel grössere Spannung) zu steuern. Das funktioniert wie folgt:

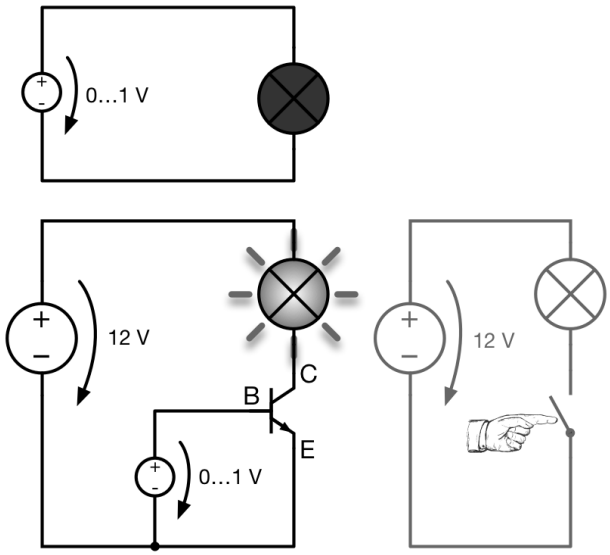


Bild 6 – Ein Transistor schaltet eine Glühlampe

In der Schaltung oben ist eine Glühlampe an eine sehr schwache Spannungsquelle angeschlossen: Die Spannung bzw. die Stromstärke reicht nicht aus, um die Glühlampe zum Leuchten zu bringen.

In der Schaltung unten links ist die Glühlampe an eine ausreichend starke 12V-Spannungsquelle angeschlossen – jedoch ist noch ein Transistor nachgeschaltet. Die schwache Spannungsquelle aus obiger Schaltung ist nun mit dem Transistor verbunden. Liegt zwischen dem mittleren (B) und dem unteren Anschluss (E) des Transistors eine Spannung an, so wird die Verbindung zwischen dem oberen (C) und dem unteren Anschluss (E) leitend.

Die Schaltung unten rechts zeigt dieselbe Schaltung, wenn wir uns den Transistor als normalen Schalter vorstellen.

Der Transistor ist also ein **elektronischer Schalter**, weil er mit einer elektrischen Spannung gesteuert werden kann. Er ist aber auch ein **dynamischer Schalter**: Ein normaler Schalter (z.B. ein Lichtschalter) kennt nur zwei Zustände: *offen* oder *geschlossen* – entweder fliesst Strom oder es fliesst kein Strom. Beim Transistor hingegen fliesst *mehr* oder *weniger* Strom: Je grösser die Spannung am Eingang, desto besser leitet er, desto mehr Strom fliesst durch. Im Vergleich mit einem Lichtschalter hiesse das: Je stärker du drückst, desto heller leuchtet das Licht.

Diese Eigenschaft des dynamischen Schaltens macht den Transistor als Verstärker verwendbar.

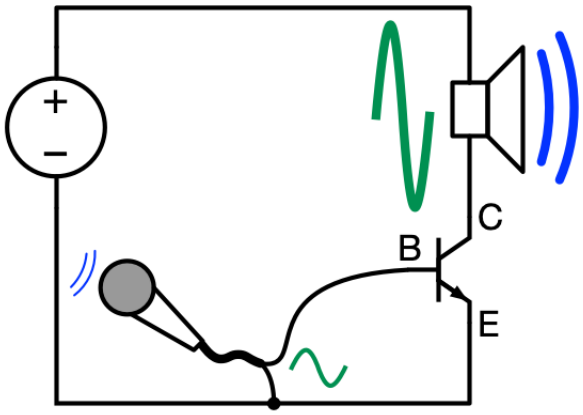


Bild 7 – Ein Transistor verstärkt ein Audiosignal.

Wenn du laute Musik hörst, sind heutzutage fast immer Transistoren im Einsatz. Bild 7 zeigt eine

vereinfachte Transistor-Verstärker-Schaltung: Das Mikrofon erzeugt eine kleine, wechselnde Spannung (das Audiosignal). Diese schaltet den Transistor dynamisch, so dass er analog zum Audiosignal mehr oder weniger leitet. Wenn der Transistor stark leitet, fliesst viel Strom durch den Lautsprecher, wenn er schwach leitet, fliesst wenig Strom. So erklingt dann der Lautsprecher, der an eine starke Spannungsquelle angeschlossen ist, analog zum Audiosignal am Mikrofon.

Im Folgenden interessiert uns diese dynamische Eigenschaft des Transistors jedoch nicht. Wie du wohl weisst, gibt es in der Digitaltechnik nur die Zustände 0 und 1. In digitalen Schaltungen sollen Transistoren also nicht mehr oder weniger schalten, wie sie das in Verstärkern tun, sondern sie sollen ganz oder gar nicht schalten.

Es gibt zahlreiche verschiedene Transistor-Arten mit unterschiedlichen Eigenschaften. Sie können in zwei Hauptgruppen unterteilt werden: In **bipolare** und **unipolare** Transistoren: In den Bildern 6 und 7 siehst du das Schemasymbol eines bipolaren Transistors. Seine Anschlüsse heissen *Base (B)*, *Emitter (E)* und *Collector (C)*. Die Schemasymbole von unipolaren Transistoren sehen etwas anders aus und die Anschlüsse heissen *Gate (G)*, *Source (S)* und *Drain (D)*. Hier eine kleine Übersicht:

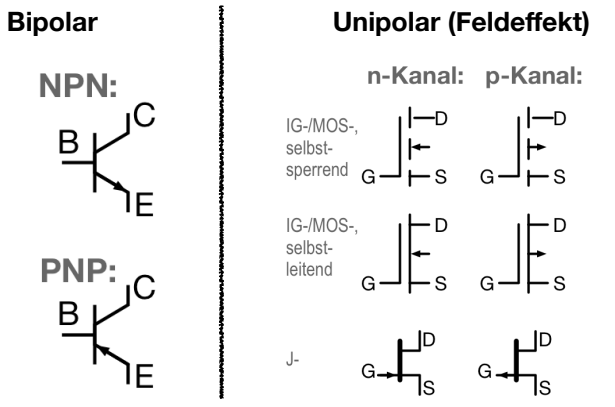


Bild 8 – Symbole der wichtigsten Transistor-Arten.

Unipolare Transistoren werden meist **Feldeffekt-Transistoren**, kurz **FET**, gennant. Warum sie so heissen, kannst du auf der nächsten Seite im Exkurs zur Funktionsweise von Transistoren nachlesen.

Was ist nun der grosse Unterschied zwischen bipolaren Transistoren und Feldeffekt-Transistoren? Oft wird gesagt: *„Bipolare Transistoren werden mit Strom gesteuert, Feldeffekt-Transistoren werden mit Spannung gesteuert.“* Weil Spannung und Strom zusammenhängen und sich gegenseitig beeinflussen, ist diese Aussage nicht ganz richtig. Bipolare Transistoren können mit einem kleinen Strom einen etwa hundertmal grösseren Strom schalten. Feldeffekt-Transistoren benötigen zum Schalten *nahezu keinen Strom*. Durch die geringe Verlustleistung entsteht auch weniger Wärme. In einer CPU, in der Milliarden von Transistoren auf einem Raum von der Grösse eines Daumennagels untergebracht sind, ist das wichtig. Wäre eine solche CPU mit bipolaren Transistoren aufgebaut, würde sie viel mehr Strom und Platz benötigen.

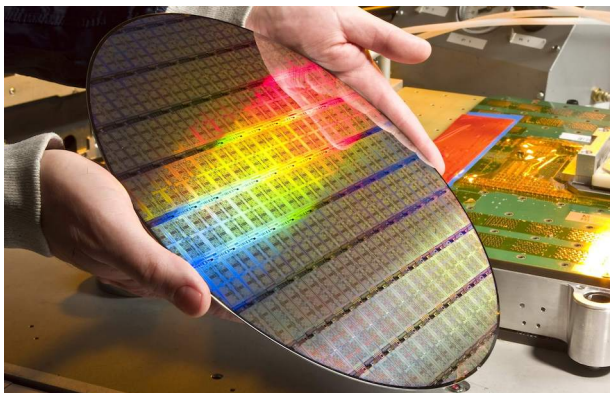


Bild 9 – Ein „Wafer“: Eine Silizium-Platte, die bereits mehrere hundert CPU-Chips enthält.

Fast alle Transistoren – und damit ein Grossteil aller Bausteine in digitalen Geräten – sind aus dem Halbleiter **Silizium** hergestellt. Die englische Bezeichnung **„Silicon“** gab dem Technologie-Tal in Kalifornien den Namen und wird heute auch synonym mit „Halbleiter“ und „Halbleiter-Technologie“ gebraucht. Hier ist knapp erklärt, wie eine CPU aus Silizium hergestellt wird (inkl. Video). Auf der folgenden Seite erfährst du, wie Transistoren aufgebaut sind und wie sie funktionieren.

Exkurs A – Die Vorgänge im Transistor

Folgende Erklärungen sind leichter zu verstehen, wenn du dich mit diesem Video über Halbleiter informierst.

Funktionsweise des bipolaren Transistors

Bild 10 zeigt den inneren Aufbau eines bipolaren Transistors in drei Zuständen. Als Ergänzung zu folgenden Erklärungen ist dieses Video hilfreich.

Löcher-Strom I_B. Der Widerstand zwischen C und E ist nun klein. Für ein Elektron, das zur Base geht, gehen etwa 99 zum Collector. Je grösser I_B, desto grösser I_C.

Funktionsweise des unipolaren Transistors:

Aufgrund der Wirkung durch ein elektrisches Feld wird der unipolare Transistor auch Feldeffekt-Transistor (FET) genannt.

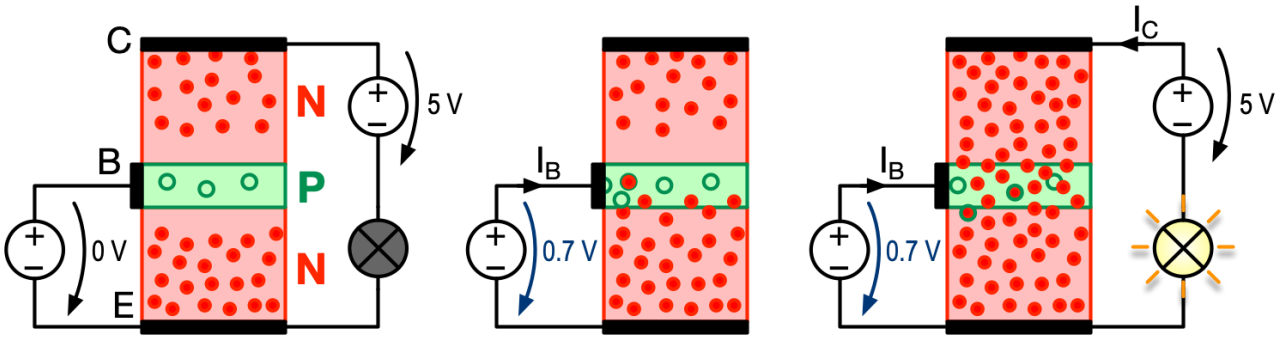


Bild 10 – Funktionsweise des bipolaren Transistors: Elektronen passieren zwei unterschiedlich gepolte Schichten.

Bild 10 links: Im nicht-beschalteten Zustand ist der Transistor nicht leitend. In den Grenzbereichen zwischen den N-Schichten und der P-Schicht bestehen sogenannte *Sperrschichten*, das sind Bereiche ohne freie Ladungsträger. Der Widerstand zwischen C und E ist hier sehr gross. Es fliesst kein Strom.

Bild 10 Mitte: Wird über B und E eine Spannung angelegt, so wird die Sperrschicht dieses PN-Übergangs abgebaut, Elektronen (rot) wandern vom *Emitter* zur *Base*, Löcher (grün) in die Gegenrichtung: Es fliesst ein Strom I_B.

Bild 10 rechts: Da die P-Schicht sehr dünn ist (viel dünner als hier dargestellt) und verhältnis-mässig wenige Löcher aufweist, springt ein Grossteil der Elektronen aus der unteren gleich in die obere N-Schicht. Nur ein geringer Teil rekombiniert mit den Löchern und führt so zum

Bild 11 links: Die beiden N-Schichten (rot) an den Anschlüssen D und S sind von einer P-Schicht (grün) und damit von Sperrschichten umgeben. Der FET leitet nicht. In der P-Schicht befinden sich freie Elektronen.

Bild 11 rechts: Wird eine positive Spannung zwischen G und S angelegt, so entsteht zwischen G und Bulk ein **elektrisches Feld** (grüne Pfeile). Dieses Feld lässt die Elektronen nach links wandern. So wird ein n-leitender Kanal erzeugt. Der FET leitet.

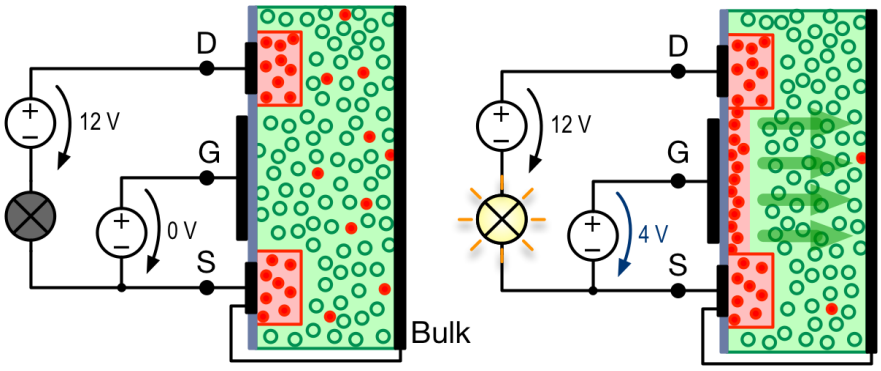


Bild 11 – Funktionsweise eine FET: Elektronen passieren einen unipolaren Kanal.

Teil 3 – Logische Verknüpfungen

Wie können aus Transistoren komplexe, programmierbare Strukturen entstehen? Der Prozessor in einem Smartphone schafft es, „gleichzeitig“ die Eingaben am Touchscreen auszuwerten, Bilder auf dem Display zu zeichnen, Musik abzuspielen und vieles mehr. Wie geht das mit einem Haufen elektronischer Schalter? Die Antwort erfolgt Schritt für Schritt und beginnt bei den logischen Verknüpfungen.

Die UND-Verknüpfung

Eine sehr einfache Aufgabe einer elektrischen Steuerung wäre die: Die Leuchtdiode (LED), die an einer Kaffeemaschine anzeigt, dass der Wassertank leer ist, soll dann leuchten, wenn A) der Wassertank eingesetzt ist ***und*** B) der Wasserstand gering ist:

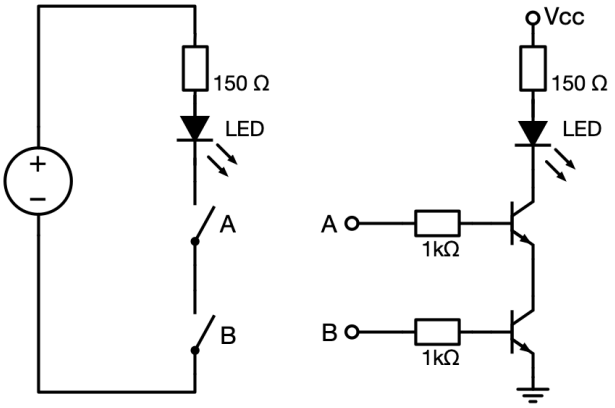


Bild 12 – Und-Verknüpfung mit Schaltern/Transistoren
Vcc-Symbol = Pluspol der Spannungsquelle. Symbol mit drei Strichen = Minuspol der Spannungsquelle.

Bild 12 zeigt links die beiden Sensoren A und B als Schalter. Die meisten Sensoren sind aber nicht so gebaut, dass sie wie Schalter einen Kontakt öffnen und schliessen. Viele Sensoren ändern passend zur Messgrösse ihren Widerstand oder geben eine bestimmte Spannung ab. Solche könnten in der Schaltung rechts verwendet werden: Wenn die Spannung vom Sensor A oder B gross genug ist, schaltet der entsprechende Transistor.

In beiden Schaltungen kann nur dann Strom durch die LED fliessen, wenn *beide* Schalter geschlossen bzw. *beide* Transistoren leitend sind.

Die ODER-Verknüpfung

Bei der ODER-Verknüpfung leuchtet die LED, wenn Sensor A *oder* B (oder beide) aktiv sind:

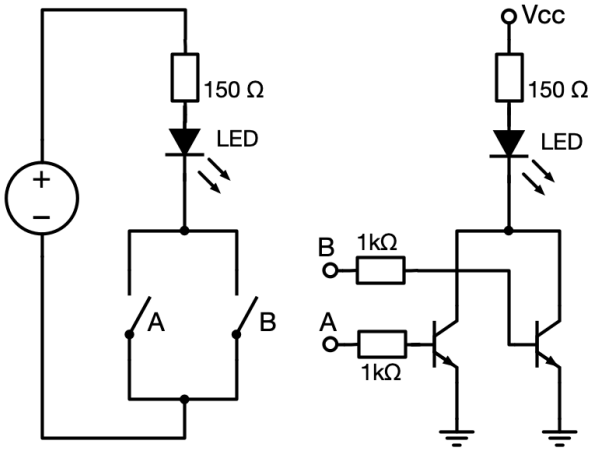


Bild 13 – Oder-Schaltung mit Schaltern/Transistoren

Damit Strom durch die LED fliessen kann, reicht es, wenn nur Schalter A oder nur Schalter B geschlossen ist. Für die Schaltung mit den Transistoren gilt das gleiche.

Die NICHT-Schaltung (Inverter-Schaltung)

Bei der NICHT-Schaltung leuchtet die LED, wenn der Sensor *nicht* aktiv ist:

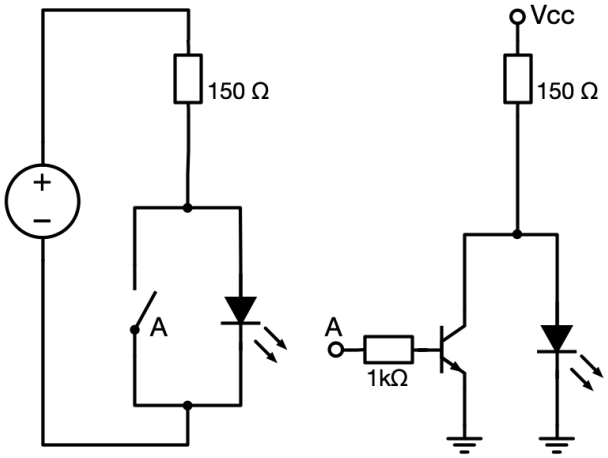


Bild 14 – NICHT-Schaltung mit Schaltern/Transistoren

Wenn Schalter A offen ist, fliesst Strom durch die LED. Wenn Schalter A geschlossen ist, fliest der Strom lieber durch Schalter A, da er auf diesem Weg viel weniger Widerstand hat.

HIGH oder LOW, 1 oder 0, true or false

In den Erklärungen oben hast du von Spannungen und Strömen gelesen. In der Digitaltechnik spricht man meistens bloss von 0 und 1 oder von LOW und HIGH:

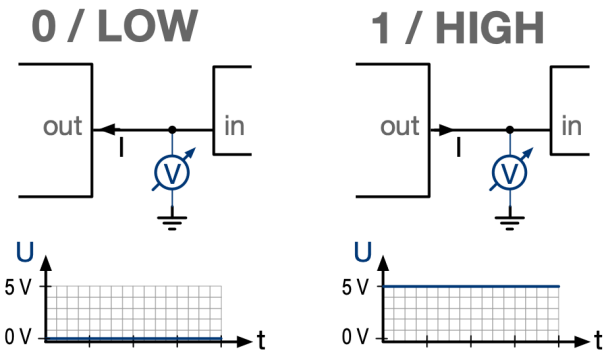


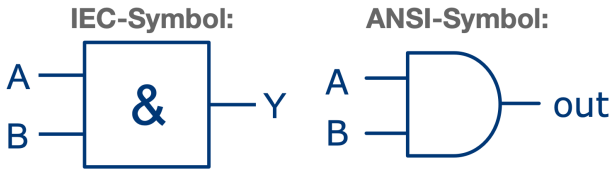
Bild 15 – Spannung und Strom am digitalen Ausgang.

Bild 15 zeigt zwei digitale Bausteine: der Ausgang des linken Bausteins ist mit dem Eingang des rechten Bausteins verbunden. Dazwischen ist ein Voltmeter angeschlossen. Im Diagramm darunter siehst du die Spannung, die das Voltmeter misst. Wenn ein Ausgang an einem digitalen Baustein 0 ist, dann beträgt die Spannung dort 0 Volt, die Spannung ist also tief, „LOW" und Strom fliesst in den Ausgang hinein. Wenn der Ausgang 1 ist, dann beträgt die Spannung je nach Logik-Standard 3.3 V oder 5 V (seltener auch andere Spannungen), die Spannung ist also hoch, „HIGH" und es fliesst Strom aus dem Ausgang heraus. In der Software-Entwicklung werden die beiden Werte oft *true* und *false* genannt.

Logik-Gatter / Logic gates

Mit Transistoren kannst du logische Verknüpfungen bauen. Diese gibt es auch als fertige Bausteine. Sie heissen Logik-Gatter (*logic gates*) und haben bestimmte Symbole und Funktionen, die du fortan kennen solltest. Folgend sind jeweils die Symbole der IEC (*International Electrotechnical Commission*) und des ANSI (*American National Standards Institute*) aufgeführt. Beide Standards sind verbreitet:

AND-Gate:



Wahrheitstabelle:

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

Boolsche Algebra:

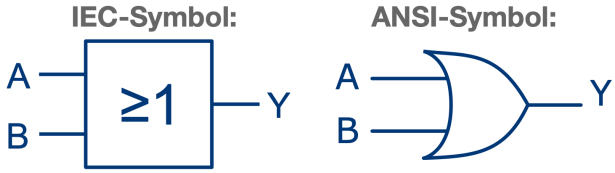
$Y = A \wedge B$

logisch UND

Bild 15 – UND-Gatter

Beim AND-Gate ist der Ausgang nur dann 1, wenn alle Eingänge 1 sind. Das Et-Zeichen '&' deutet die Funktion an. Das ANSI-Symbol kommt ohne Zeichen aus: Die Eingangs-Seite links ist eine gerade Linie, die Ausgangs-Seite ein runder Bogen.

OR-Gate:



Wahrheitstabelle:

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

Boolsche Algebra:

$Y = A \vee B$

logisch ODER

Bild 16 – ODER-Gatter

Das Grösser-Gleich-1-Zeichen '≥1' im IEC-Symbol bedeutet, dass einer oder mehrere Eingänge 1 sein können, damit der Ausgang 1 ist. Das ANSI-Symbol unterscheidet sich vom AND-Gate auf beiden Seiten: Die Eingangsseite ist hier ein Bogen, die Ausgangsseite ein Spitzbogen (wie in der gotischen Architektur –> Merkhilfe: G**o**tik–**O**der).

XOR-Gate:

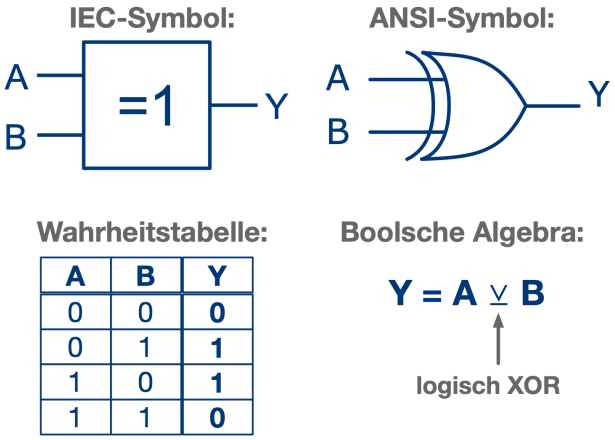


Bild 17 – Exklusiv-ODER-Gatter

Das Gleich-1-Zeichen im IEC-Symbol bedeutet, dass genau einer der Eingänge 1 sein muss, damit der Ausgang 1 ist. Das ANSI-Symbol entspricht dem OR-Gate – mit zusätzlichem Bogen an der Eingangsseite. XOR steht für „e**x**klusive-**OR**“, also „ausschliessendes ODER“. Wir können es auch „Entweder-ODER“ nennen: Der Ausgang ist 1, wenn *entweder* A *oder* B – aber nicht beide – 1 sind.

NOT-Gate/Inverter

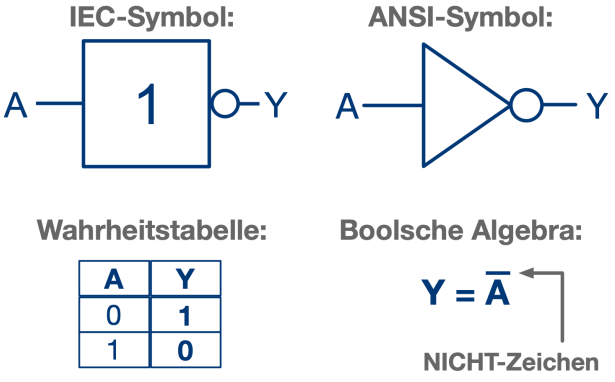


Bild 18 – NICHT-Gatter

Das NICHT-Gatter hat im Gegensatz zu allen anderen Gattern nur einen Eingang und eine sehr einfache Funktion: Der Ausgang ist 1, wenn der Eingang 0 ist – und umgekehrt.

Eine kleine Schaltung mit Gattern:

Die automatischen Sonnenstoren auf Karls Sitzplatz fahren aus (Motor M=1), wenn die Sonne scheint (Lichtsensor S=1) oder wenn es regnet (Regensensor R=1). Sie fahren auf jeden Fall ein (M=0), wenn es windet (Windsensor W=1). Diese Funktion wird durch folgende Schaltung erfüllt:

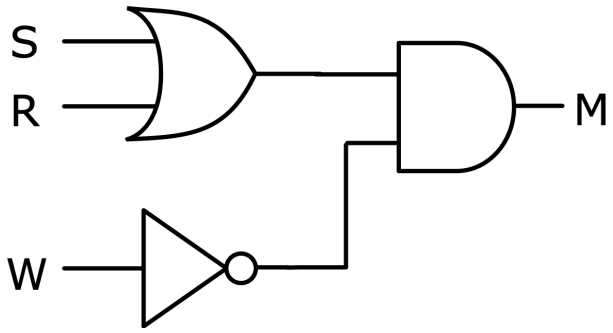


Bild 19 – Sonnenstorensteuerung

Der Motor M ist 1, wenn S oder R 1 ist (oder beide) *und* W 0 ist.