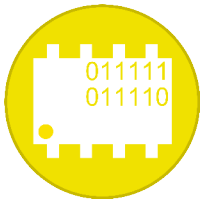
Deutsche JuniorAkademien

JuniorAkademie Adelsheim

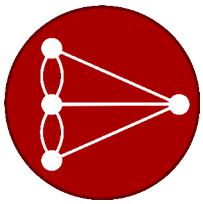
12. Science Academy Baden-Württemberg 2014



Digitaltechnik

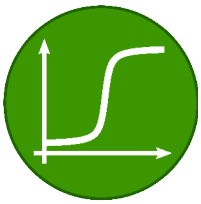
Geophysik

Geschichte/Germanistik



Mathematik

Physik



TheoPrax

H.W. & J. Hector Stiftung

Regierungspräsidium Karlsruhe
Abteilung 7 – Schule und Bildung

Dokumentation der JuniorAkademie Adelsheim 2014

12. Science Academy Baden-Württemberg

Träger und Veranstalter der JuniorAkademie Adelsheim 2014:
Regierungspräsidium Karlsruhe
Abteilung 7 –Schule und Bildung–
Hebelstr. 2
76133 Karlsruhe
Tel.: (0721) 926 4454
Fax.: (0721) 933 40270
E-Mail: georg.wilke@scienceacademy.de
petra.zachmann@scienceacademy.de
www.scienceacademy.de

Die in dieser Dokumentation enthaltenen Texte wurden von den Kurs- und Akademieleitern sowie den Teilnehmern der 12. JuniorAkademie Adelsheim 2014 erstellt. Anschließend wurde das Dokument mit Hilfe von LaTeX gesetzt.

Gesamtredaktion und Layout: Jörg Richter
Druck und Bindung: RTB Reprotechnkik Bensheim

Vorwort

Die Science-Academy Baden-Württemberg fand in diesem Jahr bereits zum 12. Mal am Landesschulzentrum für Umwelterziehung auf dem Eckenberg in Adelsheim statt. Gemeinsam mit einem fast 30-köpfigen Leiterteam verbrachten hier rund 70 Schülerinnen und Schüler aus ganz Baden-Württemberg die zweiwöchige Sommerakademie, das Eröffnungswochenende und das Dokumentationswochenende.

Am Eröffnungswochenende stehen sich die Teilnehmerinnen und Teilnehmer gegenüber, ohne sich jemals zuvor begegnet zu sein. Am Dokumentationswochenende hat sich jeder von ihnen nicht nur in die wissenschaftlichen Inhalte seines Kurses vertieft, sondern sich auch persönlich weiterentwickelt.

Während der gemeinsamen Zeit wurde aus diesen einzelnen Personen eine große Gemeinschaft. Es entstand eine Atmosphäre, die die Zeit zu einer sehr besonderen machte. Das Gefühl, im Kurs eine „bahnbrechende“ Erkenntnis gewonnen zu haben und etwas Neues ausprobiert zu haben, trägt ebenso dazu bei wie das Gefühl, seine Grenzen kennengelernt zu haben, vielleicht überwunden zu haben, zumindest aber daran gewachsen zu sein. Auch sind es der respektvolle Umgang miteinander und der Raum für Kreativität und Individualität, die diese besondere Atmosphäre entstehen ließen und sie prägten.

Um der Gemeinschaft zusätzlich einen gemeinsamen Rahmen zu geben, steht jede Akademie unter einem bestimmten Motto. In diesem Jahr war es das „Glück“, das uns über die Zeit hinweg begleitete. Glück ist ein weiter Begriff und hat unwahrscheinlich viele Facetten. Für uns standen die kleinen Glücksmomente, die Freundschaften, die entstehen und die Erfahrungen, die hier gemacht werden, im Vordergrund. Für einige Glücksmomente sorgte unser *geheimer Freund*, der

uns immer wieder mit kleinen Aufmerksamkeiten überrascht hat. Gemeinsam haben wir viele schöne Momente erlebt. Diese Momente werden uns immer begleiten und in Erinnerung bleiben.

Am Ende haben wir die Akademie wieder durch die Akademietür verlassen, und unsere gemeinsame Zeit ist zu Ende gegangen. Doch eines möchten wir euch mit auf dem Weg geben: Eure Wege werden sich wieder kreuzen! Die Freundschaften, die hier entstehen, halten oft noch über Jahre hinweg, und die Erfahrungen und die Erkenntnisse, die ihr hier gewonnen habt, gehen euch nie mehr verloren. Geht also mit offenen Augen durchs Leben und haltet Ausschau nach neuen „Türen“. Habt den Mut, sie zu öffnen und durch sie hindurch zu gehen.

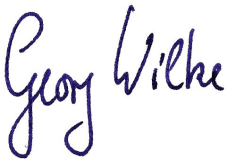
Wir wünschen euch alles Liebe und Gute für euren weiteren Weg und für das, was als nächstes auf euch zukommt. Wir freuen uns sehr darauf, euch bald – in egal welchem Zusammenhang – wieder zu sehen. Vielleicht ja sogar wieder hier in Adelsheim!

Viel Spaß beim Lesen und Schmökern!

Eure/Ihre Akademieleitung

Patricia Keppler (Assistenz)

Nico Röck (Assistenz)



Georg Wilke

Dr. Petra Zachmann

Inhaltsverzeichnis

Kurs 1 – Digitaltechnik: Wir bauen einen Computer

Vorwort – *grit*

Laura Merker, Kevin Sommer,
Michael Mattes

Das englische Wort *grit* trifft gut, was wir bei den Teilnehmerinnen und Teilnehmern beobachten können. Es könnte mit Charakterstärke beziehungsweise Durchhaltevermögen übersetzt – oder mit *ich will es ganz genau wissen und bin bereit, dafür zu arbeiten* umschrieben werden. Nicht die reine Intelligenz ist wichtig, sondern dieser vorurteilsfreie Hunger nach neuem Wissen und die Bereitschaft, sich darauf einzulassen. Wenn diese Voraussetzungen erfüllt sind, folgt der Spaß ganz von alleine.

Wir Kursleiter sind jedenfalls stolz, eine so motivierte Gruppe junger Menschen betreut haben zu dürfen. Im Folgenden stellen unsere „Digis“ Ihnen vor, womit sie im Sommer 2014 voller *grit* ihre Zeit verbracht haben.

Unser Kurs

Michael ist einer unserer Kursleiter und begeisterte uns sehr für die Inhalte des Digitaltechnikkurses. Die hat er uns immer gutgelaunt nahegebracht und uns jedes Mal geholfen, wenn es Probleme gab. Trotzdem gab er uns immer neue Denksportaufgaben zum Knobeln, beispielsweise während der Zugfahrt an unserem Exkursionstag, als er uns mit schweren Rätseln den ganzen Tag beschäftigte.

Kevin hat uns, genauso wie Michael, als Kursleiter mit seinem großen Wissen und Können unterstützt. Kein Fehler in unseren Programmen war vor seinen Adleraugen sicher. Über den Kurs hinaus hat er auch seine Begeisterung für das Tanzen an alle Interessierte in einer KüA weitergegeben und mit seiner Kamera viele besondere Momente der Science-Academy festgehalten.

Laura, unsere Schülermentorin, stand uns mit Rat und Tat zur Seite, egal um was es ging, sei es eine Verständnisfrage oder das Wählen des „Schlachtrufs“ am Sporttag. Mit ihrer liebenswürdigen Art hat sie nie die Geduld mit uns verloren und war immer bereit, uns etwas zu erklären – auch wenn es nur die Welt war.

Alina ist diejenige, die noch eine Idee hatte, wenn keiner mehr weiter wusste. Sie fand immer alle Programmierfehler und war uns eine große Hilfe. Zudem ist sie eine super Teamarbeiterin und immer schnell fertig, wenn es um das Programmieren geht. Außerdem war sie in der Sport-KüA sehr aktiv und zeigte am Abschlussabend ihr turnerisches Können.

Florian ist ein sehr netter Kursteilnehmer mit seinem ganz eigenen Humor. Während des Kurses hat er mit seiner schnellen Kombinationsfähigkeit und seinem Wissen auch die härtesten Nüsse geknackt. Außerdem hat er den Inhalt des Kurses sehr schnell verinnerlicht und war somit immer einer der Ersten, der die Aufgaben gelöst hatte.

Helen kam grundsätzlich pünktlich eine Minute zu spät. Doch wenn sie dann da war, hat sie immer freudig mitgearbeitet und mit ihrer lockeren Art alle zum Lachen gebracht. Außerdem hat sie bei den Vorbereitungen für unsere Abschlusspräsentation tatkräftig mitgeholfen.

Holger ist ein eher ruhigerer Teilnehmer unseres Kurses. Er zeigt sich immer freundlich und aufgeschlossen und war daran interessiert aus der Akademiezeit viele neue Eindrücke mitzunehmen. Er konnte darüber hinaus schnell HDL-Dateien verfassen und machte hierbei fast keine Fehler. Außerdem konnte man ihn immer gut um Hilfe fragen, falls man mal etwas nicht verstanden hatte.

Kevin ist ein gut gelaunter Teilnehmer, der schon viele Vorkenntnisse zum Thema Programmieren und Computeraufbau hatte. Mit seiner lässigen Art meisterte er die schwierigen und stressigen Phasen unseres Kurses. Auch in seiner Freizeit engagierte er sich sowohl als DJ an den Partys, als auch in der Theater-KüA, in der er die Hauptrolle spielte.

Lukas ist ein sehr aufmerksamer Teilnehmer mit vielen guten Ideen für den Kurs. Bei der Gruppenarbeit hat er immer sein Wissen zeigen können und hat dadurch die Gruppe voran gebracht. Sobald es mal Fehler in den Programmen gab, hat er sie schnell gefunden.

Melina ist eine sympathische und offene Teilnehmerin und man hat gemerkt, wie sehr sie sich für das Thema Digitaltechnik interessiert. Sie ist immer motiviert in den Kurs gegangen, und beim Programmieren war sie eine große Hilfe. Nie hatte sie Schwierigkeiten, etwas zu verstehen oder umzusetzen. Ferner ist sie eine begnadete Tänzerin und zeigte dies am Bergfest und am Abschlussabend in der Zirkusaufführung.

Nevena ist ein wissbegieriges und super nettes Mädchen, das neuen Dingen stets offen ist. Im Kurs arbeitete sie sehr aufmerksam und aktiv mit und trug entscheidend zu den Kurs-Präsentationen bei. Außerdem brachte sie sich mit der Querflöte in verschiedenen Ensembles ein. Sie war also nicht nur eine tolle Teilnehmerin im Kurs, sondern man hatte mit ihr auch viel Spaß und Freude in der restlichen Zeit.

Noah ist ein energiegeladener und lebendiger Teilnehmer unseres Kurses. Er wollte die Kursinhalte immer sehr genau verstehen und trieb durch seine vielen Fragen den Kurs voran. Sein besonderes Interesse lag beim Programmieren, und durch seine Vorkenntnisse konnte er uns vor allem beim Programmieren unseres Spiels helfen.

Peter hat mit seiner guten Laune und seinem Elan Schwung in die ganze Gruppe gebracht. Im Kurs hat ihn besonders das Programmieren und die HDL interessiert, und sein Lieblingsbaustein ist der „Multiplexer“. Auch wenn er zu allem ein Kommentar hatte, brachte er uns doch immer zum Lachen.

Teresa ist ein aufgeschlossenes, total freundliches und fröhliches Mädchen, und sie war während der ganzen Kurszeit immer sehr interessiert und motiviert bei der Sache. Sie ist eine begeisterte Turnerin, stellte uns sogar beim Abschlussabend etwas von ihrem

Können vor und liebt es, sich Dinge wie Taschen und Geldbeutel selbst zu nähen. Teresa hat alles, was wir im Kurs neu gelernt haben, immer mit erstaunlicher Schnelligkeit verinnerlicht und verstanden. Es hat riesigen Spaß gemacht, mit ihr zusammenzuarbeiten.

Tobias ist ein aufmerksamer und gewissenhafter Mensch, der alle Aufgaben gründlich erledigte. Er kennt sich gut mit den HDL-Dateien aus und war schnell mit dem Schreiben der Dateien fertig. Wenn man nicht weiter kam, konnte man Tobias immer um Hilfe bitten.

Einleitung

Helen Zwölfer

Der Kurs Digitaltechnik: Drei engagierte Betreuer, zwölf interessierte Jugendliche – mit mehr oder weniger Vorwissen – und ein gemeinsames Ziel: Wir wollen einen Computer bauen – und das in nur zwei Wochen!

Vielleicht dachte der ein oder andere am Anfang noch, dass wir in das nächste Elektronikfachgeschäft gehen, dort die groben Einzelteile kaufen und diese dann zusammenbauen. Doch weit gefehlt! Unsere Betreuer hatten etwas ganz anderes mit uns vor: Wir bauen einen Computer, aber von Anfang an. Das heißt, wir beschäftigen uns mit jedem Einzelteil und versuchen, es zu verstehen.

Dazu „zoomen“ wir sozusagen in den Computer, um den Aufbau Schritt für Schritt zu analysieren: Zuerst trifft man auf die CPU (Central Processing Unit → Prozessor). Darin befindet sich die ALU (Rechenzentrum). In dieser findet man die logischen Operatoren, die alle aus einem Grundbaustein, dem NAND, aufgebaut sind. Würden wir noch eine Stufe weiter „hineinzoomen“, kämen wir zu den Transistoren, aber mit diesen haben wir uns in unserem Kurs nicht beschäftigt.

Stattdessen haben wir mit der Hardware Description Language (kurz HDL) gearbeitet, damit konnten wir die logischen Operatoren, die ALU, die CPU und sogar unseren ganzen Computer simulieren. Allerdings war unser Computer nicht so modern wie die, die heutzutage jeder zu Hause hat. Unser Computer entsprach eher einem älteren Modell, ähnlich dem Apple II von 1977: Er hat einen Schwarz-Weiß-Monitor, keine Maus und ist zudem etwas langsam – aber dafür wurde unser Computer in nur zwei Wochen von uns programmiert bzw. simuliert.

Nachdem die „Hardware“ stand, sind wir noch eine Stufe weiter gegangen und haben ein Betriebssystem programmiert. Mit diesem konnten wir das kleine – aber bekannte – Spiel „Galgenmännchen“ programmieren, welches dann auf unserem simulierten Computer lief.

Apple II von 1977[1]

In den folgenden Abschnitten wollen wir die einzelnen Teile und Schritte genauer erklären. Dazu schauen wir wieder ganz tief in den Computer hinein und treffen zunächst auf die logischen Operatoren – und unser erstes Problem! Denn diese Operatoren können mit unserer Sprache nichts anfangen – sie verstehen nur 0en und 1en. Anders gesagt: Das Binärsystem.

Binärsystem und ASCII

Florian Wiese

Wie schafft es der Computer, mit diesen zwei Möglichkeiten alles Nötige zu berechnen? Dazu sehen wir uns als erstes die zwei Zustände genauer an. Es gibt einerseits den Zustand „1“, der für „Spannung liegt an“ oder „wahr“ steht. Andererseits den Zustand „0“, der für „Spannung liegt nicht an“ oder „falsch“ steht. Aber es gibt nicht nur einzelne 1en oder 0en,

[1] Photo: Wikimedia (Marcin Wichary, CC-BY-2.0)

sondern man kann diese zu Ketten aus 0en und 1en zusammenschließen. Diese stehen zum Beispiel für dezimale Zahlen, Zeichen oder Befehle. Diese Schreibweise aus 0en und 1en wird Binärsystem genannt.

Wie aber werden die Zahlen dargestellt? Im Groben ist es sehr ähnlich zum Dezimalsystem. Allerdings gibt es ja nur zwei mögliche Zustände, nämlich 0 und 1. Wenn wir eine Zahl um 1 erhöhen wollen, dann müssen wir eine 1 an der letzten Stelle hinzuzählen. Ist die letzte Ziffer jedoch 1 und zählen wir dann die 1 hinzu, wird diese auf 0 gesetzt und es gibt einen Übertrag, der zur nächsten Stelle hinzugefügt wird. Hier wird dann wieder genauso verfahren, wie bei der ersten Stelle und immer so weiter. Dies ähnelt sehr der schriftlichen Addition. So ist es möglich, auf die Binärzahlen zu schließen:

- $0000 \rightarrow 0$
- $0001 \rightarrow 1$
- $0010 \rightarrow 2$
- $0011 \rightarrow 3$
- ...

Die letzte Stelle hat den Wert 2^0, die zweite Stelle 2^1, die dritte 2^2 und so weiter. So können wir durch eine Binärzahl auf die entsprechende dezimale Zahl schließen, als Beispiel hier die Zahl 5: 0101. Die letzte Stelle steht für 2^0, also 1. Da hier eine 1 steht $1 \cdot 2^0$. Jetzt zur zweiten Stelle: Sie steht für 2^1, also 2. Dies multipliziert mit Null, da hier eine 0 steht. Also $0 \cdot 2^1$. Jetzt zur dritten Stelle, diese steht für 2^2, also 4. An dieser Stelle steht eine 1, also $1 \cdot 2^2$. Zählen wir jetzt die Zahlen zusammen $(1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3)$, so ist das Ergebnis die entsprechende Dezimalzahl, hier also 5.

Wenn wir dies jetzt mit dem Dezimalsystem vergleichen, stellen wir fest, dass einige Gemeinsamkeiten vorhanden sind. Allerdings gibt es auch einige Unterschiede, denn hier steht die erste Ziffer für 10^0, also 1, die zweite Ziffer für 10^1, also 10, die dritte Stelle für 10^2, also 100 und so weiter. Hier kann man aber die Zahl nicht nur mit 0 oder 1 multiplizieren, wie in dem Binärsystem, sondern mit Zahlen von 0 bis 9.

Zum Darstellen von Buchstaben und Zeichen im Binärsystem wird das ASCII-System verwendet. Hierzu wird jedem Buchstabe und Symbol eine Zahl zugeordnet, welchen die Tastatur dann an den Computer weitergibt. Beispiel:

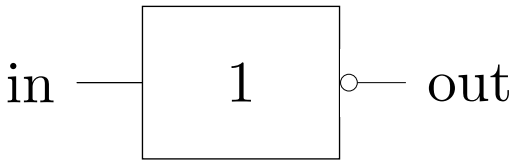
- A wird mit 75 codiert,
- B mit 76, ...
- a mit 90,
- b mit 91, ...

Diese Zahlen werden natürlich auch im Binärsystem an den Computer übermittelt, A ist für den Computer also 01001011. Auf diese Weise kann der Computer die Eingaben der Tastatur verarbeiten und speichern.

Logische Operatoren

Melina Soysal

Im Abschnitt zuvor haben wir bereits gelesen, dass der Computer mit nur zwei Zuständen arbeiten kann. Wie schafft es aber so ein Computer, nur durch zwei Zustände so viele verschiedene Befehle, wie zum Beispiel 7+9, auszuführen? Hier kommen die kleinsten logischen Bauteile des Computers ins Spiel: Die logischen Operatoren. Diese sind aus noch kleineren Bauteilen aufgebaut, den Transistoren. Transistoren sind elektronische Schalter, aus denen elektronische Schaltungen gebaut werden, aber auf diese sind wir im Kurs nicht genauer eingegangen.



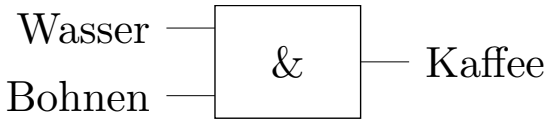
Hier sieht man das NOT-Schaltbild: Es besteht aus einem Rechteck mit einer 1 in der Mitte und einem Kreis auf der rechten Seite, also am Ausgang. Dieser Kreis steht für das Invertieren des Signals.

Die einfachste logische Operation ist das NOT. Es hat einen Eingang und einen Ausgang. Liegt ein Signal am Eingang an, wird es invertiert. Das heißt, das Signal wird umgekehrt: Aus einer 1 wird eine 0 und aus einer 0 eine 1. Das ist auch an der Wahrheitstabelle erkennbar.

IN	OUT
0	1
1	0

Die linke Spalte zeigt das Signal am Eingang und die rechte das am Ausgang.

Schon etwas schwieriger wird es beim AND, das zwei Eingänge und einen Ausgang hat. Man kann diesen Operator mit einer Kaffeemaschine vergleichen. Dabei ist der Eingang A Wasser und der Eingang B stellt die Bohnen dar. Der Ausgang ist Kaffee. Diesen bekommt man nur, wenn Bohnen und Wasser zeitgleich vorhanden sind, nicht aber, wenn man nur Bohnen, nur Wasser oder keins von beidem hat. So ergibt sich, dass ein AND nur eine 1 ausgibt, wenn an beiden Eingängen eine 1 anliegt.

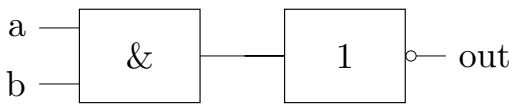


AND-Baustein am Beispiel der Kaffeemaschine

B	A	OUT
0	0	0
0	1	0
1	0	0
1	1	1

An der Wahrheitstabelle kann man ebenfalls erkennen, dass nur dann eine 1 ausgegeben wird, wenn an beiden Eingängen eine 1 anliegt.

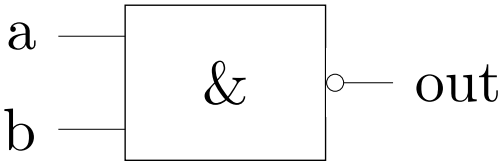
Die beiden ersten Bausteine kann man miteinander kombinieren. Dabei entsteht ein NAND, also ein NOT-AND. Wie der Name schon sagt, ist dieses Logikgatter genau gleich aufgebaut wie ein AND, jedoch wurde der Ausgang invertiert. Deshalb wird auch hier wieder hinter das Schaltsymbol, das genauso aussieht wie ein AND-Gatter, ein Kreis hinzugefügt. Dieser steht wie schon beim NOT für das Invertieren des Ausgangssignals.



Aufbau eines NAND

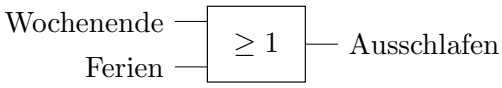
B	A	OUT
0	0	1
0	1	1
1	0	1
1	1	0

Wahrheitstabelle eines NAND – die Spalte Ausgang ist im Vergleich zum AND invertiert.



Schaltbild eines NAND

Ein weiteres Logikgatter ist das OR. Das OR kann man auch mit etwas Alltäglichem vergleichen: Nämlich dem Ausschlafen. Dabei sind die beiden Eingänge Wochenende (A) und Ferien (B) und der Ausgang ist Ausschlafen. Wenn nun der Eingang A 1 ist, ist es Wochenende und man kann ausschlafen. Am Ausgang liegt also eine 1 an. Das Gleiche gilt für die Ferien oder falls beides zutrifft. Ist aber keins von beidem wahr, ist der Ausgang auf 0, das heißt man kann nicht ausschlafen, da weder Wochenende noch Ferien sind.



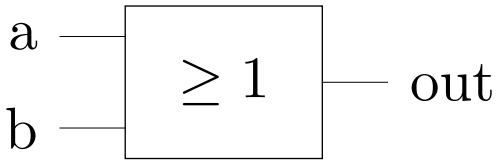
OR-Baustein am Beispiel Ausschlafen

B	A	OUT
0	0	0
0	1	1
1	0	1
1	1	1

Wahrheitstabelle eines OR

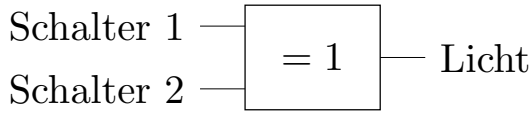
Auch hier kann man das OR mit dem NOT verbinden. Man erhält ein NOR, also ein invertiertes OR. Die Wahrheitstabelle ist wieder genau gleich wie beim OR, sobald man die Spalte des Ausgangs invertiert.

Außerdem gibt es noch ein XOR (engl. exclusive OR), das man sich praktisch wie zwei Lichtschalter im Treppenhaus als Eingänge und das Licht selbst als Ausgang vorstellen kann. Egal welchen Schalter man zuerst drückt, das Licht



Schaltbild eines OR

geht an, und egal welchen Schalter man danach drückt, das Licht wird ausgehen. Das kann man auch in der Wahrheitstabelle sehen.

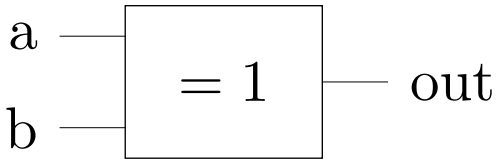


XOR-Baustein am Beispiel Lichtschalter

B	A	OUT
0	0	0
0	1	1
1	0	1
1	1	0

Wahrheitstabelle eines XOR

Der ganze Computer lässt sich aus NANDs aufbauen. Deshalb benutzen wir es als Grundbaustein. So bauen wir auch unser NOT aus einem NAND. Das einzige Eingangsignal des NOTs schließt man an beide Eingänge des NANDs. Dadurch entsteht ein NOT, denn wenn das Eingangssignal eine 0 ist, sind beide Eingänge des NANDs auch auf 0. Laut der Wahrheitstabelle bedeutet das, dass eine 1 am Ausgang anliegt, so wie es bei einem NOT der Fall ist. Das Gleiche gilt, wenn das Eingangssignal eine 1 ist. Beide Eingänge des NOTs sind auf 1, das heißt der Ausgang ist eine 0, wie bei einem NOT.



Schaltbild eines XOR

Wenn man nun das NAND mit dem NOT verbindet, erhält man ein AND. Da der Ausgang des NANDs einem invertierten AND entspricht, heißt das, man erhält durch erneute Invertierung des NANDs wieder den Ausgang eines ANDs. So hat man schon ein NOT und ein AND aus NANDs aufgebaut. Führt man dieses Schema nun immer weiter und baut größere Bauteile aus Teilen, die man bereits aus NANDs gebaut hat, erhält man eine fast fertige Computerhardware.

Minimierung und Boolesche Algebra

Teresa Augustin

Wie man logische Operatoren baut, haben wir im Abschnitt zuvor schon erläutert. Wie man sie aber einfach aufschreibt, ohne Schaltsymbole, und wie man den Endwert einer komplizierten Schaltung schnell herausfindet, werden wir nun mit Hilfe der Booleschen Algebra kennenlernen. Die Boolesche Algebra dient der Verallgemeinerung logischer Operationen mit Hilfe von Symbolen. Dafür werden folgende Symbole verwendet:

AND-Operator: $\wedge$

OR-Operator: $\vee$

NOT-Operator: $\neg$ oder ein Strich über einer Variablen $\rightarrow \neg x = \overline{x}$

Diese Symbole werden wie mathematische Symbole in Gleichungen verwendet. Dabei ist jedoch folgende Regel zu beachten:

$\neg$ kommt vor $\wedge$, und $\wedge$ kommt vor $\vee$

Durch Klammern lassen sich andere Prioritäten setzen. Außerdem gelten die folgenden Gesetze:

Für alle $x, y, z \in \{0, 1\}$ gilt:

Kommutativgesetze:
$x \wedge y = y \wedge x$
$x \vee y = y \vee x$

Assoziativgesetze:
$x \wedge (y \wedge z) = (x \wedge y) \wedge z$
$x \vee (y \vee z) = (x \vee y) \vee z$

Distributivgesetze:
$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$

Idempotenzgesetz:
$x \wedge x = x$
$x \vee x = x$

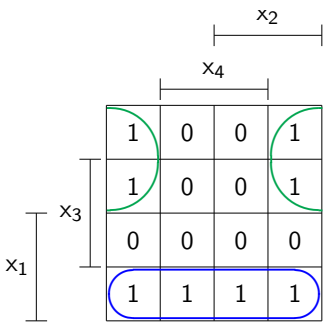
Absorbtionsgesetz:
$x \wedge (x \vee y) = x$
$x \vee (x \wedge y) = x$

De Morgansche Gesetze:
$\neg(x \wedge y) = \neg x \vee \neg y$
$\neg(x \vee y) = \neg x \wedge \neg y$

Um von der Wahrheitstabelle einer Schaltung auf die passende Boolesche Gleichung zu kommen, gibt es die Möglichkeit, ein Karnaugh-Veitch-Diagramm (KV-Diagramm) zu verwenden. Dieses kann man jedoch nur bis maximal 4 Variablen anwenden, ohne den Überblick zu verlieren. Dennoch ist es sehr sinnvoll, denn wenn man bei 4 Variablen den Schaltplan direkt aus der Wahrheitstabelle ablesen würde, müsste man einen sehr großen und komplizierten Schaltplan entwerfen. Das Prinzip funktioniert für 4 Variablen folgendermaßen:

Wir zeichnen ein Quadrat, wobei dieses in 16 Kästchen unterteilt wird (siehe Bild).

Die mit x_1, x_2, x_3 bzw. x_4 gekennzeichneten Bereiche, wie auf dem Bild, beziehen sich immer auf 8 Kästchen und bedeuten, dass bei diesen 8 Kästchen die Variable 1 ist. In den restlichen 8 Kästchen ist die Variable 0.



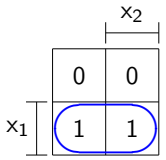
KV-Diagramm mit vier Variablen

Nun tragen wir von der Wahrheitstabelle die jeweiligen Werte in die Kästchen. Das heißt, die Werte, die sich ergeben, wenn bestimmte Bedingungen gelten (Variable 1 oder 0).

Dabei gibt es zwei verschiedene Vorgehensweisen, die Minterm- und die Maxterm-Methode.

Im KV-Diagramm von oben wird die Minterm-Methode angewandt. Das heißt, dass wir alle 1en zu 2er-, 4er-, 8er- oder 16er-Blöcken zusammenfassen. Dabei darf man auch, wie im Beispiel eben, über die Kanten des Quadrats hinaus gehen. Wenn sich 1en nicht mit anderen verbinden lassen, kann man auch eine einzelne 1 als Block einzeichnen. Generell gilt, dass man eine vollständige Überdeckung der 1en mit möglichst großen rechteckigen Blöcken sucht.

Als nächstes werden Bedingungen für diese 1er-Blöcke formuliert. Das bedeutet, welche Variable bei dem gesamten Block immer 1 und welche 0 ist.



KV-Diagramm mit zwei Variablen

Der nächste Schritt ist, dass wir unsere Bedingungen eines Blocks in Konjunktionsterme umwandeln. Das heißt, wir schreiben zwischen die Variablen ein AND.

Zum Schluss verknüpfen wir diese Konjunktionsterme mit OR. Damit haben wir unsere fertige Boolesche Gleichung, aus der wir problemlos den dazugehörigen Schaltplan ablesen können. Die fertige Gleichung lautet also: $(x_1 \wedge \neg x_3) \vee (\neg x_1 \wedge \neg x_4)$

Bei der Maxterm-Methode ist die Vorgehensweise grundlegend gleich, nur dass hier die 0en zusammengefasst werden. Außerdem werden die Variablen eines Blocks zu Disjunktionstermen umgewandelt, also mit einem OR verknüpft, und die Blöcke werden mit AND verknüpft.

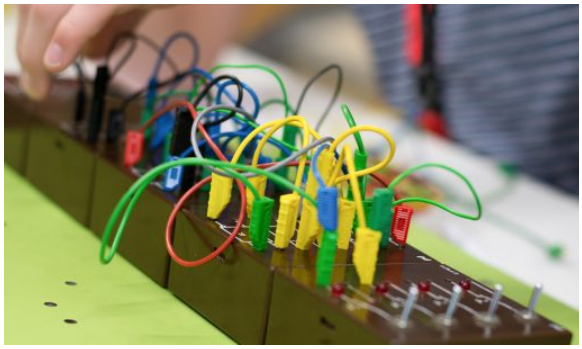
x_1	x_2	x_3	x_4	out
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

HDL

Peter Martin

Um diese komplexen Gleichungen erfolgreich umzusetzen, bedienen wir uns der HDL. HDL ist die Abkürzung für Hardware Description Language. Wie dieser Name uns schon verrät, ist die HDL eine Sprache zur Beschreibung von Hardware. In dieser Sprache beschreiben wir unseren ganzen Computer. Mithilfe eines Programms simulieren wir dann an einem Computer unseren nun erstellten Computer.

Wir haben unser Ziel erreicht: Wir haben einem richtigen Computer gebaut, den wir danach mit Software ausstatten (siehe „Das Spiel“). Den Bildschirm des fertigen Computers mit der Software können wir dann an einer Simulation betrachten und unseren simulierten Computer mit der Tastatur bedienen.



Auf diesem Bild sieht man verkabelte NAND-Bausteine.

Aber warum die HDL? Wir könnten doch einfach auch alles mit unseren NAND-Bausteinen bauen. Als wir die ersten größeren Chips bauten, wie etwa das XOR, das vergleichsweise noch ziemlich klein ist, fiel uns auf, dass wir eine Menge Platz benötigen würden, wenn wir den ganzen Computer mit den NAND-Bausteinen bauen würden. Wie wir am Ende des Projektes ausrechneten, bräuchten wir etwa 2500 Quadratmeter, um alle Bausteine aufzustellen. Die circa 3 500 000 NANDs würden uns 162 Tage 24 Stunden am Tag beanspruchen, wenn wir nur zwei Sekunden für jedes NAND bräuchten. Außerdem sind 100.000 € allein für die Bausteine einfach kein allzu angenehmer Betrag, wenn man beachtet, dass wir diese auch noch mit Strom versorgen und miteinander verkabeln müssten. Deshalb verwenden wir für unseren Computer die HDL.

Nun müssen wir in der HDL strikte Regeln beachten, die wir an einigen Chips darstellen möchten. Zunächst müssen wir jedoch noch erwähnen, dass der einzige, durch die Simulationssoftware vorgegebene Chip, das NAND ist.

```
CHIP Kaffee{
 IN Bohnen, Wasser;
 OUT Kaffee;

 PARTS:
 AND(a = Bohnen, b = Wasser, out = Kaffee);
}
```

Mit den Wörtern „CHIP Kaffee“ zeigen wir der Simulation, dass wir einen Chip erstellen, den wir Kaffee nennen. Nun schreiben wir „IN“, wodurch wir angeben können, wie viele Eingänge der Chip haben soll und wie diese heißen. In diesem Fall haben wir zwei Eingänge, mit den Namen „Wasser“ und „Bohnen“. Nun geben wir die Menge und Namen der Ausgänge mit Hilfe des „OUT“ an.

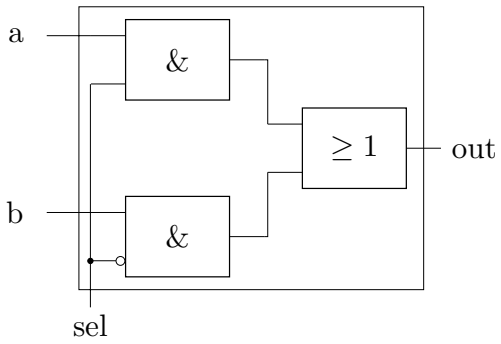
Nun folgt die genaue Beschreibung des Inneren des Chips. Diese Beschreibung wird mit „PARTS:“ eingeleitet. Hier greifen wir nun den bereits vorhandenen Chip „AND“ auf, welchen wir zuvor bereits in HDL beschrieben haben. Wir teilen die Eingänge und die Ausgänge des „Kaffee-Chips“ den Eingängen und Ausgänge des „AND-Chips“ zu.

```
CHIP And {
 IN a, b;
 OUT out;

 PARTS:
 Nand(a=a, b=b, out=nand);
 Not(in=nand, out=out);
}
```

Des Weiteren werden wir noch kurz auf den sogenannten „Multiplexer“ oder kurz den „MUX“ eingehen. Dieser hat drei Eingänge: A, B, SEL. Der Eingang „SEL“ entscheidet, welcher dieser beiden Eingänge dem OUT zugewiesen wird. Wenn SEL 0 ist, nimmt OUT den gleichen Wert

an, der auch dem Eingang A zugewiesen wurde. Wenn SEL 1 ist übernimmt OUT den Wert von B.



Auf diesen Bild sieht man einen MUX in seinen Einzelteilen.

In der Abbildung sieht man, dass man für diesen MUX ein OR, zwei ANDs und ein NOT benötigt. In der HDL beschrieben muss man, anders als in dem Schaubild, das NOT extra aufschreiben. Theoretisch ist es egal in welcher Reihenfolge man die einzelnen Chips aufschreibt, dennoch ist es für eine übersichtliche Schreibweise ratsam. Außerdem kann man an diesem Beispiel sehen, dass man einen Eingang an mehrere Chips anschließen kann. Hier ist es der SEL-Eingang. Er geht sowohl in eines der ANDs als auch in das NOT. Zusätzlich ist es möglich, einen Ausgang eines Chips innerhalb eines größeren Chips nicht weiter zu verwenden, solange das letztendliche OUT definiert wird.

```
CHIP Mux {
 IN a, b, sel;
 OUT out;

 PARTS:
 And(a = sel, b = b, out = selB);
 Not(in = sel, out = notSel);
 And(a = notSel, b = a, out = notSelA);
 Or(a = selB, b = notSelA, out = out);
}
```

Die vorhandenen, bereits beschriebenen Chips kann man weiterverwenden. So bauen wir uns immer größere Chips, wie zum Beispiel das Rechenwerk, das Steuerwerk und letztendlich den Computer.

CPU und Program Counter

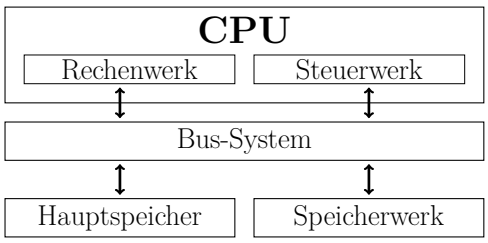
Lukas Ruf

Wie im vorherigen Abschnitt beschrieben konnten wir nun unseren ganzen Computer simulieren. Dazu zählt alles, was physisch zum System gehört, also alle Computerteile und Peripheriegeräte – auch als Hardware bezeichnet, die man in die Hand nehmen kann.

Computer mit Peripheriegeräten

Das wichtigste Element eines Computers ist heutzutage die Hauptplatine, auch Mainboard oder Motherboard genannt, mit dem Hauptprozessor, auch CPU (Central Processing Unit).

In unserem Kurs haben wir uns gezielt mit der CPU, dem Arbeitsspeicher, der Tastatur und dem Bildschirm beschäftigt.



Schema der Bauteile im Computer

Die CPU besteht aus integrierten Schaltkreisen, die mittlerweile mehrere Millionen Transistoren enthalten. Die Integrationsdichte und damit die Leistungsstärke nimmt dabei ständig zu. Die CPU setzt sich im Wesentlichen aus drei Teilen zusammen: Dem Steuerwerk, dem Rechenwerk und den Registern. Das Steuerwerk steuert, wie der Name schon sagt, wohin die Befehle gehen, holt die benötigten Daten aus dem Hauptspeicher und sagt dem Rechen-

werk, was es rechnen muss. Das Rechenwerk führt nun den eigentlichen Befehl aus.

Sowohl das Steuerwerk als auch das Rechenwerk brauchen sehr schnelle Zwischenspeicher. Das sind die Register, die alle in einem Registersatz zusammengefasst sind. Verbunden sind die drei Teile durch ein internes Bussystem, das über den angeschlossenen Systembus auch Zugriff auf den Hauptspeicher oder Peripheriegeräte wie den Bildschirm hat.

Ein spezielles Element des Steuerwerks ist der Program Counter. Er zeigt an, an welcher Stelle man sich im Programmablauf befindet und ist ein spezielles Register, das die Speicheradresse des Befehls enthält. Es gibt drei verschiedene Befehle, die der Program Counter ausführen kann: Sprungbefehle, bei denen an eine bestimmte Stelle des Ablaufes gesprungen wird; Resetbefehle, bei denen der Zähler auf 0 gesetzt wird und Zählbefehle, bei denen einfach auf die nächste Adresse gezählt wird.

Das Rechenwerk führt den sogenannten aktuellen Befehl aus und rechnet in der ALU (Arithmetic Logic Unit), was das Steuerwerk vorgibt, und leitet das Ergebnis wieder zurück an das Steuerwerk.

Die Register dienen zur Datenspeicherung. Hier werden die auszuführenden Befehle gespeichert. Es gibt verschiedene Register, wie zum Beispiel das Befehlsregister, das Statusregister und die Datenregister.

In den Datenregistern sind die Operanden der ALU und ihre Zwischenergebnisse gespeichert. In dem Adressregister sind die Speicheradressen eines Operanden oder Befehls gespeichert.

Der Bus (Binary Unit System) ist ein System zur Datenübertragung zwischen mehreren Teilnehmern über einen gemeinsamen Datenweg. Dabei sind die Teilnehmer nur an ihrem Datenaustausch beteiligt und vom Datenaustausch zwischen zwei anderen Teilnehmern nicht betroffen. Man unterscheidet zwischen drei verschiedenen Bussen: Dem Adressbus, dem Datenbus und dem Steuerbus. Ein Adressbus überträgt Speicheradressen. Ein Datenbus überträgt Daten zwischen zwei Computerkomponenten. Der Steuerbus bewerkstelligt die Steuerung des Bussystems. Alle Komponenten der CPU und des Computers können auf das Bussystem zugreifen und es benutzen.

Addierer und 2er-Komplement

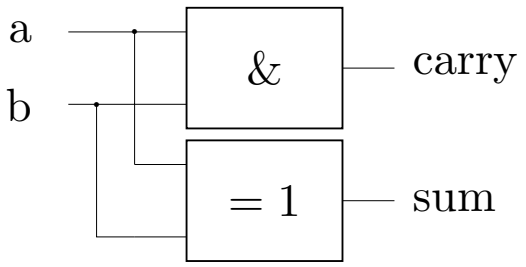
Tobias Gross

Um die ALU bauen zu können, benötigt man Addierer. Addierer sind Bauteile, die in Computern verwendet werden. Es gibt sowohl Halb- als auch Volladdierer. Sie dienen dazu, zwei beziehungsweise drei Ziffern miteinander zu verrechnen. Allerdings sind sie nur dazu in der Lage zu addieren und nicht zu subtrahieren, multiplizieren oder dividieren. Sie gehen dabei wie bei der schriftlichen Addition vor:

- $0 + 0 = 0$
- $1 + 0 = 1$
- $0 + 1 = 1$
- $1 + 1 = 0$, Übertrag 1

Der Übertrag 1 wird zur nächsten Stelle weitergereicht.

Ein Halbaddierer ist ein Bauteil, das dazu in der Lage ist, zwei einstellige Binärzahlen, also $0 + 0$, $0 + 1$, $1 + 0$ oder $1 + 1$ miteinander zu addieren. Er hat zwei Eingänge, a und b, und zwei Ausgänge, carry out und sum. Carry out gibt den Übertrag beider Zahlen an (Bei $1 + 1 = 0$, Übertrag 1, sonst ist der Übertrag 0). Hierfür verwendet man ein AND. Sum gibt die Summe zweier Zahlen an (Ergibt bei $1 + 0$ und $0 + 1$ die 1, sonst 0). Hierfür verwendet man ein XOR.



Halbaddierer

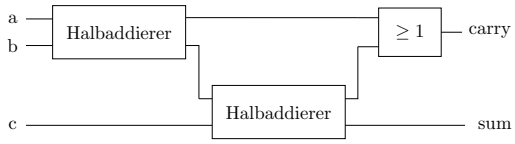
Ein Volladdierer dagegen hat drei Eingänge: a, b und c. Der letzte Eingang c wird auch Carry in genannt, in den der Übertrag des vorherigen Addierers eingeht. Dieser Volladdierer kann je

a	b	carry	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Halbaddierer Wahrheitstabelle

drei einstellige Binärzahlen miteinander addieren.

Um einen solchen Addierer zu bauen, schließt man zwei Halbaddierer und eine OR-Operation hintereinander an.



Volladdierer

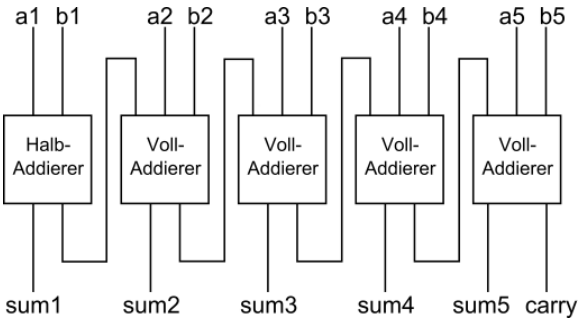
a	b	carryin	sum	carryout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Volladdierer Wahrheitstabelle

Um aber größere Binärzahlen miteinander addieren zu können, reicht ein Volladdierer allein nicht mehr aus. Deswegen schaltet man mehrere Addierer so zusammen, dass der Übertrag des vorherigen Addierers in den nächsten Addierer miteingeht. So kann man beliebig lange Binärzahlen addieren. Diesen Zusammenschluss aus mehreren Addierern nennt man dann Paralleladdierwerk.

In diesem Zustand kann unser Addierer aber bis jetzt nur addieren. Für eine Subtraktion stellt man folgende Überlegung an: Wenn eine negative Zahl mit einer Positiven addiert wird, z. B. $5+(-3)=2$, hat das denselben Effekt, als wenn die zuvor negative Zahl (-3) invertiert wird und von der Positiven (5) subtrahiert wird. Hier: $5-3=2$.

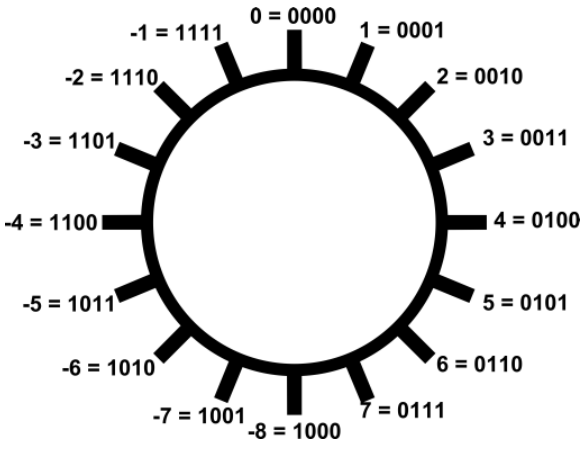
Deswegen können wir für die Subtraktion eben-



Paralleladdierer

falls einen Addierer nutzen. Will man zwei Zahlen voneinander subtrahieren, wandelt man eine Zahl in eine Negative um und addiert sie dann nach denselben Regeln, wie man auch davor zwei positive Zahlen addiert hat.

Nun steht man aber erneut vor einem Problem: Diese negativen Zahlen müssen auch im binären System dargestellt werden können. Dazu werden alle Stellen, beispielsweise 0101 für die 5, mithilfe einer NOT-Operation negiert (aus jeder 0 wird eine 1 und aus jeder 1 wird eine 0). Man erhält also den Binärcode 1010. Zuletzt addiert man noch eine 1 hinzu: Wichtig ist hierbei jedoch die 0en vor der ersten 1 zu berücksichtigen. Beispiel: $1010+1=1011$ Das ist nun der binäre Code für -5.



Zahlenkreis

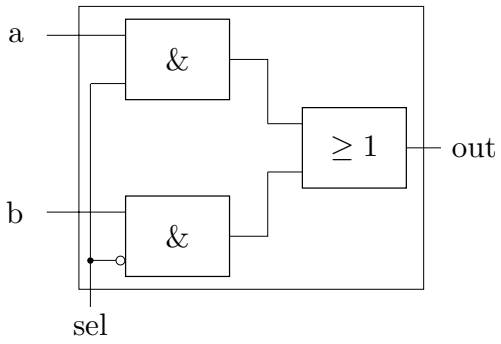
Schaut man genau auf den Zahlenkreis, fällt auf, dass alle positiven Zahlen als erstes Bit eine 0 besitzen und alle negativen Zahlen eine 1. Deswegen kann man das erste Bit als Vorzeichenbit bezeichnen. Es zeigt an, ob es sich bei der Zahl um eine negative oder positive Zahl handelt. Diese Darstellung der negativen und positiven Zahlen nennt man 2er-Komplementdarstellung.

Oder kurz: 2er-Komplement. So kann ein n-Bit-Rechner (ein Rechner, der n Bits gleichzeitig verarbeiten kann) Zahlen bis 2^{n-1} sowohl in positive als auch in negative Richtung verarbeiten. Allerdings gibt es da noch eine Unstimmigkeit, die auffällt, wenn man sich den Zahlenkreis näher anschaut. Man erkennt, dass die Zahl 7 direkt an die Zahl -8 grenzt. So ist $7+1=-8$. Oder allgemein: Der höchsten positiven Zahl $2^{n-1}-1$, folgt die kleinste negative Zahl -2^{n-1}. Bei einem 4-Bit-System sieht das dann folgendermaßen aus: $0111+0001=1000$ $(7+1=-8)$

ALU und Multiplexer

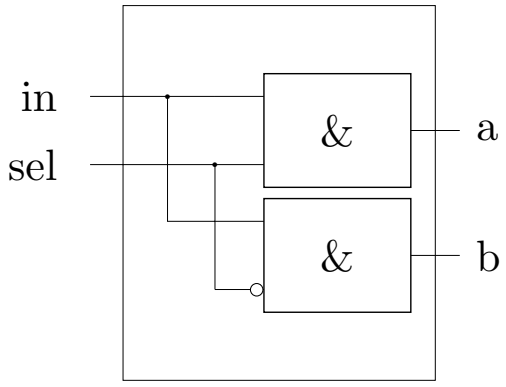
Kevin Yuan

Nun können wir die fehlenden Teile für die CPU fertig stellen, da wir wissen wie unser Computer rechnen kann. Im Prozessor befinden sich auch noch weitere Bauteile, wie der Multiplexer, Demultiplexer und das Herzstück – die ALU.



Schaltbild eines Multiplexers

Ein Multiplexer ist ein elektronisches Bauteil, bei dem man das am Ausgang anliegende Signal mit Hilfe des Selektorpins aus einem der mehreren Eingänge auswählen kann. Er ist vergleichbar mit einem Drehschalter, welcher rein mechanisch funktioniert. Der Multiplexer besitzt zwei oder mehr Dateneingänge (A, B, ...), ausreichend Selektor-Eingänge, sowie einen Ausgang (OUT). Er wird oft mit MUX [+Zahl] Way abgekürzt, wobei MUX für einen Multiplexer mit zwei Engängen steht, folglich bezeichnet man mit MUX4Way einen zweifachen Multiplexer mit vier Eingängen. Man kann einen Multiplexer aus einem NOT, zwei ANDs und einem OR bauen. Man verwendet ihn beispielsweise in einem Speicherchip oder in der ALU, um die gewünschte Rechenoperation auszuführen.



Schaltbild eines Demultiplexers

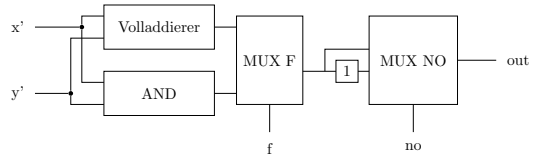
Der Demultiplexer macht genau das Gegenteil des Multiplexers. Mit dem Demultiplexer kann man ein am Eingang liegendes Signal auf einen der mehreren Ausgänge legen. Auf welchen Ausgang man das Signal nun schaltet, legt man auch hier wieder mit Hilfe des Selektorpins fest. Der Demultiplexer besitzt einen Eingang (IN), mindestens zwei Ausgänge (OUT1, OUT2), bei zwei Ausgängen ein Selektorpin (SEL1), bei vier Ausgängen zwei Selektorpins (SEL1, SEL2) und so weiter. Abgekürzt wird der Demultiplexer oft mit DMUX oder DMUX [+Zahl] Way. Ein DMUX4Way ist ein Demultiplexer mit einem Eingang und vier Ausgängen, ein DMUX8Way ein Demultiplexer mit acht Ausgängen. Den DMUX4Way kann man aus zwei NOTs und vier ANDs bauen, indem man sie so verschaltet wie auf dem Bild. Der Demultiplexer wird beispielsweise im Speicherchip verwendet.

Die ALU oder auch Arithmetic Logic Unit (deutsch: Arithmetisch-logische Einheit) ist ein Bauteil, welches sowohl arithmetische Operationen, wie beispielsweise Addieren, als auch logische Operationen, wie AND-Verknüpfungen, NOT-Verknüpfungen oder OR-Verknüpfungen durchführen kann.

Man findet die ALU innerhalb des Prozessors. Dort sitzt sie neben ein paar Registern und führt die Befehle, die sie bekommt, aus. Die Bezeichnung für die ALU lautet n-Bit-ALU,

wobei n für die Anzahl der Bits am Eingang steht. Derzeit anzutreffen sind 8-; 16-; 32- und 64-Bit-ALUs. Die ALU hat zwei Eingänge, X und Y, welche jeweils die maximale Bitbreite der ALU haben, bei einer 16-Bit-ALU wäre die Bitbreite der Eingänge 16 Bit. Des Weiteren besitzt sie einen 16-Bit-Ausgang und Selektorpins, deren Anzahl und Funktion nach Prozessortyp und Architektur variieren. Die ALU, die wir in unserem Kurs gebaut haben, besitzt sechs Selektorpins, welche alle unterschiedliche Funktionen besitzen.

Die meisten ALUs sind unserer ähnlich aufgebaut. Die am Eingang liegenden Signale gehen zuerst durch aktivierbare Inverter (NOTs) und gelangen anschließend zu den intern verbauten Logik-Gattern und arithmetischen Bauteilen. Um am Ausgang nur das Ergebnis einer Rechenoperation zu erhalten, wird mit einem Multiplexer eines der Ergebnisse ausgewählt.



Ausschnitt aus dem Schaltbild der ALU. Über die Multiplexer-Selektoren f und no werden die Rechenoperation sowie die Ausgabeart gewählt.

Bei unserer selbst gebauten ALU wird das Verhalten über verschiedene Selektoren – die Flags – eingestellt. Unsere ALU ist in der Lage, einen oder beide Eingänge zu ignorieren sowie die Ein- und Ausgabewerte zu invertieren. Über das f-Flag wird die Rechenoperation – entweder AND oder Addition – gewählt.

Mit Hilfe der verschiedenen Flag-Kombinationen kann die ALU verschiedene Rechenoperationen wie Subtrahieren, welche hardwareseitig nicht direkt implementiert sind, ausführen.

Memory und Flipflop

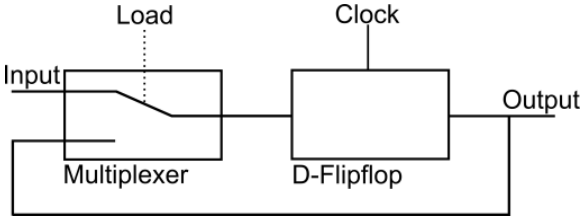
Holger Hujjon

Ein Flipflop ist ein logischer Baustein, der einen Datenwert, also eine 1 oder eine 0 speichern kann. Der Flipflop hat zwei Eingänge, INPUT und CLOCK und einen Ausgang OUT. Die CLOCK ist ein Baustein des Computers, der in regelmäßigen Abständen ein Signal an alle angeschlossenen Bauteile sendet und so das Arbeitstempo im Computer vorgibt. Immer dann, wenn am Eingang CLOCK 1 anliegt, wird ein Schritt ausgeführt, das heißt ist der Eingang CLOCK auf 1 gestellt, speichert das Flipflop den INPUT. Der Ausgang OUT gibt den gespeicherten Zustand, also 1 oder 0, aus.

Schaltbild eines D-Flipflop

Wenn der Wert gespeichert bleiben soll, muss man immer, wenn sich die CLOCK auf 1 stellt, den gespeicherten Wert nochmals einspeichern. Um dies zu umgehen, kann man mit dem Flipflop einen neuen Chip, den 1-Bit-Speicher, bauen.

Der 1-Bit-Speicher wird mit einem Flipflop und einem Multiplexer gebaut. Er besitzt 3 Eingänge und 1 Ausgang. Über den INPUT-Eingang lässt sich ein Wert eingeben, der dann gespeichert wird, aber nur, wenn der Eingang LOAD 1 ist. Auch dieser Chip hat einen CLOCK-Eingang. Der Ausgang OUT gibt auch hier den gespeicherten Zustand aus. Der Multiplexer entscheidet, ob der Speicher den INPUT oder OUT, also den momentan gespeicherten Wert speichert. Diese Entscheidung trifft der Multiplexer aufgrund des LOAD-Eingangs. Ist LOAD auf 0, so bleibt der eingespeicherte Wert gleich, daher leitet der Multiplexer OUT weiter, ist LOAD auf 1, so wird der INPUT vom Multiplexer weitergegeben und gespeichert.

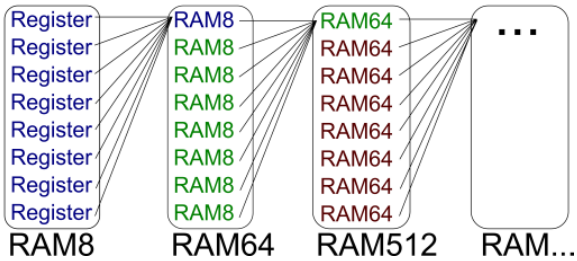


1-Bit Speicher

Mehrere 1-Bit-Speicher lassen sich zu einem Register zusammensetzen, welches bei unserem

Computer 16 Bit speichern kann. Heutzutage haben moderne Computer normalerweise Registergrößen von 32 und 64 oder sogar 128 Bit. Die Registergröße entscheidet, wie groß die größte Zahl ist, mit der der Computer arbeiten kann. Aus acht einzelnen 16-Bit-Registern haben wir anschließend einen Speicher für acht 16-Bit-Zahlen, genannt einen RAM8, gebaut. Ein RAM8 hat vier Eingänge. Die ersten drei, INPUT, LOAD und CLOCK funktionieren wie beim Register. Über ADDRESS, dem vierten Eingang, wird mit Hilfe mehrerer Demultiplexer ausgewählt, welches Register überschrieben werden soll. Der Demultiplexer entscheidet aufgrund der ADDRESS-Eingänge, die an die Selektorpins angeschlossen sind, bei welchem Register LOAD auf 1 gesetzt wird. Alle Register bekommen also den INPUT, aber nur ein Register speichert die Daten, da LOAD nur bei diesem Register auf 1 gesetzt wird. Der RAM8 hat natürlich auch einen Ausgang OUT. Dabei wird über einen Multiplexer einer der Ausgänge der Register ausgewählt, wobei als Selektorpins wieder ADDRESS verwendet wird. Der Eingang ADDRESS wählt also aus, welches Register überschrieben wird, gleichzeitig aber auch, welches Register ausgelesen wird.

Da wir aber noch größere Speicher benötigten, um einen Computer zu bauen, setzten wir immer größere Speicher zusammen: Acht RAM8 ergeben zusammengesetzt einen RAM64. Acht RAM64 ergeben einen RAM512 und so weiter...



Wie unser RAM aufgebaut ist

Je größer der Speicher, desto mehr Eingänge zur Angabe der Adresse hat er. Der größte Speicher, denn man in einem Computer einbaut, ist dann der fertige RAM. Der RAM ist ein Teil des Memory-Chips, welcher allerdings noch ein paar andere Dinge kann. Der Memory-Chip beinhaltet den Zugang zum Bildschirm und die Ausgabe der Tastatur. Er hat als Eingänge INPUT, LOAD und ADDRESS, die wir ja schon vom RAM8 kennen. Gibt man eine Zahl in ein Zugangsregister des Bildschirms ein, so werden die entsprechenden Pixel auf unserem 512*256 Pixel großen Bildschirm schwarz gefärbt. Beim Memory-Chip kann OUT auch die momentan gedrückte Taste der Tastatur sein. Der RAM ist, neben dem ROM, dem Programmspeicher des Computers, und der CPU, dem Prozessor, einer der drei wichtigsten Bausteine des Computers. Nun haben wir alle Teile, die wir für die Hardware benötigen und können an die Software gehen.

Maschinensprache, Assembly und Hochsprache

Nevena Dimitrova

Unsere nun fertige Hardware versteht nur 0 und 1, also müssen die Befehle, die sie erhält und ausführen soll, genauso formuliert werden. Die Sprache, die das erfüllt, nennt sich Maschinensprache und besteht aus 16 Bit pro Befehl, die die auszuführenden Befehle enthalten. Die Maschinensprache ist die niedrigste Instanz der Programmiersprachen und für den Computer lesbar. Ein Befehl kann zum Beispiel so aussehen: 0000000000001101

Alle Befehle sind im ROM, dem Programmspeicher des Computers, gespeichert. Von dort werden sie ausgelesen und ausgeführt. Je nach Befehl wird anschließend, gesteuert durch den Wert im Program Counter, der nächste Befehl oder ein anderer Befehl ausgeführt.

Dabei werden zwei Befehle der Maschinensprache unterschieden, die nur an dem ersten Bit erkannt werden. Der A-Befehl fängt im Binärcode mit einer 0 an: $0xxx$ und dient dazu, eine neue Zahl in dem A-Register der CPU zu speichern.

Die zweite Befehlsart wird C-Befehl genannt. Ihr Befehl fängt mit einer 1 an: 1xxx. Er bestimmt, wohin der Code geleitet wird und was mit diesem gemacht werden soll, beispielsweise welche Berechnungen die ALU, das Rechenzentrum der CPU, vornehmen soll. Der C-Befehl kann alle drei Speicher der CPU verändern:

Das A-Register, das D-Register und das M-Register (Memory).

Durch das erste Bit wird nun entschieden, ob ein neuer Befehl, der im A-Register gespeichert wurde, oder der alte Befehl, der im M-Register (dem Memory) ist, übernommen werden soll. Anschließend wird dieser ausgewählte Code mit dem vorherigen Befehl in die ALU geschickt und dort bestimmen die anderen Bits des 16stelligen Befehls, was nun mit dem ausgewählten Code passiert, zum Beispiel welche Rechenoperationen ausgeführt werden sollen.

Man programmiert natürlich nicht in dieser Maschinensprache, denn für uns ist es viel zu umständlich, komplexe Computerprogramme nur mithilfe von 0 und 1 zu schreiben. Deshalb programmiert man heutzutage nur noch mit Hochsprachen wie Java oder C++. Die Hochsprache ist die höchste und abstrakteste Instanz der Programmiersprachen und liegt auf einer viel höheren Ebene als die Maschinensprache. Diese Hochsprachen haben eine sehr menschennahe und an unser Denken angepasste Schreibweise, damit das Programmieren möglichst schnell und einfach funktioniert.

Wir haben zum Beispiel unser Spiel, das wir für unseren Computer programmiert haben, mithilfe von JACK geschrieben.

Zwischen der Maschinensprache und der Hochsprache gibt es allerdings noch eine Zwischenstufe: Die Assembly.

Die Assembly ist etwas abstrakter als die Maschinensprache, ist aber trotzdem noch sehr hardwarenah. Sie verwandelt die binären Befehle der Maschinensprache in „vereinfachte“ Assembler-Sprache. Durch diese Vereinfachung können wir schon besser nachvollziehen, was der Computer macht. Eine beispielhafte Assembly-Befehlsfolge ist die Vorgehensweise für das Überschreiben der Speicherstelle 108 im Memory mit dem Wert 17.

@17	Hier wird der Wert 17 im A-Register gespeichert.
D = A	Der Wert, der im A-Register gespeichert wurde, wird nun in das D-Register übernommen.
@108	Diese Zeile legt dann die Speicherstelle fest,
M = D	in die, als letzter Schritt, der Wert aus dem D-Register, also 17, in das Memory an der Adresse 108 gespeichert wird.

Damit wir allerdings in unserer Hochsprache schreiben können und der Computer uns trotzdem versteht, gibt es eine Art „Übersetzer“: den Compiler. Er wandelt die Befehle, die wir in der Hochsprache eingeben, in Assembly und anschließend in Maschinensprache um. Mit diesem haben wir uns aber nicht näher im Kurs beschäftigt und deshalb wird hier nicht weiter darauf eingegangen.

Betriebssystem

Alina Valta

Da unser Computer nun Hochsprachen – in unserem Fall JACK – versteht, können wir nun beginnen, mit dieser Programmiersprache unser Betriebssystem zu programmieren. Denn wie jeder Computer braucht auch unserer ein Betriebssystem.

Das Betriebssystem ist ein Programm, das die Verbindung zwischen Hardware und Anwenderprogrammen, wie Word oder Spielen, herstellt. Dies macht es, indem es den Programmen eine Reihe von Methoden (eine Methode ist ein kurzes Programm) zur Verfügung stellt, zum Beispiel eine zum Zeichnen eines Kreises. Man kann die Aufgaben eines Betriebssystems in zwei Bereiche einteilen.

Zum Einen hat das Betriebssystem systemorientierte Aufgaben. Es sorgt dafür, dass der Computer reibungslos läuft. Das beinhaltet Dinge wie die Benutzeroberfläche (Programmfenster, Menü, ...), Starten vom Computer, Speicherverwaltung (lesen, speichern, löschen von Daten), Sicherheit (zum Beispiel Zugriffsbeschränkungen) oder Gerätetreiber.

Auf der anderen Seite stellt es Grundmethoden zur Verfügung, die in den Anwenderprogrammen benötigt werden. Dazu zählen Grundlegende mathematische Operatoren, wie Multiplikation oder Division (unsere ALU kann ja nur addieren und subtrahieren), Ausgeben von

Buchstaben und Wörtern, Auslesen der Tastatur oder Maus und Grafikmethoden zum Zeichnen von geometrischen Figuren, wie Linien, Kreisen oder Rechtecken.

Unser Betriebssystem hat grundsätzlich die gleichen Aufgaben. Allerdings stellt es nur einige wichtige Grundfunktionen zur Verfügung. Auch die systemorientierten Aufgaben sind sehr beschränkt. Dinge wie Benutzeroberfläche, Betreiben von mehreren Programmen gleichzeitig, Zugriffsbeschränkungen oder Dateimanager besitzt es nicht. Unser Betriebssystem JackOS hat folgende Klassen (Klassen bestehen aus mehreren Methoden):

- Math: Mathematische Operatoren (Multiplikation, Division, Wurzel, . . .)
- String: Bei Strings handelt es sich um Zeichenketten, sodass man nicht nur einzelne Zeichen sondern ganze Wörter oder Sätze speichern kann.
- Array: Ein Array kann mehrere Elemente haben, so kann man mehre Werte zusammen speichern.
- Output: Die Klasse Output kann etwas an eine bestimmte Stelle am Bildschirm schreiben.
- Screen: Zeichnet Pixel, Linien und Rechtecke in schwarz oder weiß auf den Bildschirm.
- Keyboard: Liest die Tastatur aus.
- Sys: Kann Fehlermeldungen ausgeben und das Programm anhalten.

Da wir keine Zeit hatten, das ganze Betriebssystem selbst zu schreiben, beschränkten wir uns auf einen Teil der Klasse Screen und der Klasse Output.

In der Klasse Screen schrieben wir zwei Methoden selbst: Zum einen *setColor()*, die die Farbe, mit der wir zeichnen, auf schwarz *setColor(1)* oder weiß *setColor(0)* setzt. Außerdem schreiben wir die Methode *drawPixel (x-Koordinate, y-Koordinate)* zum Zeichnen von einzelnen Pixeln in der momentan gesetzten Farbe an der angegebenen Position. Dazu mussten wir einen Programmcode für die folgenden Schritte schreiben.

Zuerst mussten wir die Koordinaten in unsere entsprechende Speicherstelle umrechnen, sodass wir wussten, mit welcher Adresse wir den Inhalt der entsprechenden Speicherstelle ändern können und somit unser Bildschirm an dieser Stelle etwas anderes anzeigt. Unsere Speicherstellen speichern immer eine 16-stellige Binärzahl, also 16 Pixel unseres Bildschirms in einer Speicherstelle. Wir wollen aber nur einen Pixel verändern, also müssen wir noch berechnen, an welcher Position in der Speicherstelle wir unseren Pixel speichern.

Schüler in Betrieb

Danach muss der Wert der Speicherstelle ausgelesen werden, denn wir können immer nur die ganze Speicherstelle auf einmal ändern. Wir wollen aber nur eine Stelle – ein einzelnes Pixel – ändern. Damit sich auch wirklich nur das eine Pixel schwarz beziehungsweise weiß wird, benutzen wir verschiedenen Bitoperationen. Das Ergebnis wird an der entsprechenden Stelle gespeichert und somit ändert sich das Pixel auf dem Bildschirm.

```
function void drawPixel(int x, int y)
 {
 var int speicherstelle;
 var int pixel;
 var int index;
 var int spalte;

 let spalte = x / 16;
 let index = x - (spalte * 16);
 let speicherstelle
= 24576 + ((y * 32) + spalte);
 let pixel = Memory.peek(speicherstelle);

if(black)
 {
 let pixel = Math.Pow(2, index) | pixel;
 }
```

```
 else
 {
 let pixel = ~(Math.Pow(2, index)) & pixel;
 }

do Memory.poke(speicherstelle, pixel);
 return;
 }
```

Weil wir aber nicht nur einzelne Pixel zeichnen wollen, sondern Buchstaben ausgeben wollen, entwarfen wir exemplarisch die Großbuchstaben A, B und C und schrieben diese Entwürfe in unsere Klasse Output. Hierzu malten wir auf Papier in ein Rechteck von sieben auf neun Pixeln unsere Entwürfe der Buchstaben. Dabei sind die zwei unteren Zeilen und die zwei rechten Spalten der Abstand zur nächsten Zeile und zum nächsten Buchstaben. Dann wurde jede Zeile in eine Zahl umgewandelt. Waren zum Beispiel von den sieben Pixeln in der Reihe nur der dritte und vierte angemalt, ergab sich die Zahl 0011000, also 48. Dies wurde für jede Zeile gemacht und dann von bereits fertigen Methoden, die wir nicht selbst geschrieben hatten, dem entsprechenden ASCII-Wert zugewiesen und gezeichnet.

Das Spiel

Noah-Yannick Schmid und Kevin Yuan

Da wir jetzt ja ein Betriebssystem zur Verfügung hatten, wollten wir nun ein kleines Spiel programmieren.

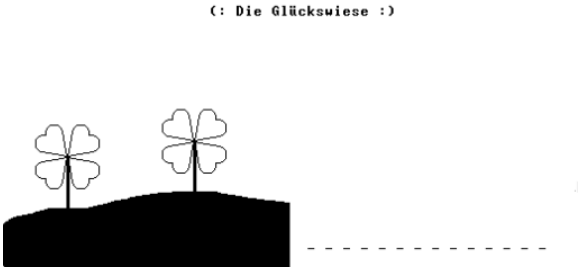
Planung und Aufbau

Zuerst haben wir uns Gedanken darüber gemacht, was wir für ein Spiel programmieren wollen. Lange hat das nicht gedauert, da wir nur sehr wenige Möglichkeiten zur Verfügung hatten. Beispielsweise konnten wir kein Spiel programmieren, bei dem große Animationen auf dem Bildschirm stattfinden, also haben wir uns für eine Computer-Version des Spiels Galgenmännchen entschieden und es passend zum Akademiemotto „Die Glückswiese“ genannt. In dem Spiel befindet sich der Spieler auf einer Wiese, auf der zwei vierblättrige Kleeblätter wachsen. Glückswiese soll ähnlich wie Galgenmänchen funktionieren: Zu Beginn gibt der Spielleiter ein Wort ein, welches später im Spiel von den Spielern erraten werden soll. Nennt der Spieler einen falschen Buchstaben, so verliert er eines seiner acht Leben/Kleeblätter. Da wir nicht alle am selben Script arbeiten konnten, mussten wir uns in Gruppen aufteilen, die alle jeweils verschiedene Aufgaben übernahmen.

Durchführung

Während der Durchführungsphase haben die verschiedenen Gruppen Teile des Spiels in der Programiersprache JACK programiert. Von den insgesammt sechs Gruppen haben sich fünf Gruppen mit der Grafik des Spiels, also dem Startbildschirm, Endbildschirm, Alphabet, Kleeblätter und Wiese beschäftigt. Die sechste Gruppe beschäftigte sich mit dem Spielablauf und dem Zusammenfügen der einzelnen Teile.

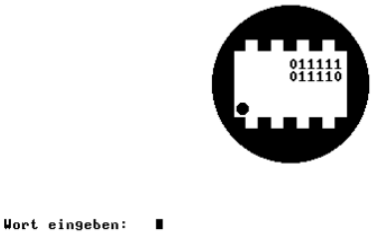
Die Kleeblatt-Gruppe zeichnete ein Musterkleeblatt zunächst in ein Papierraster. Anschließend beschrifteten sie die Seiten, sodass aus dem Raster ein Koordinatensystem wurde, dessen Punkte leicht abgelesen werden konnten. Diese vielen Punkte auf dem Papier wurden nun Pixel für Pixel nach JACK übertragen. Daraus entstand dann ein Kleeblatt.



Spielbildschirm zu Beginn

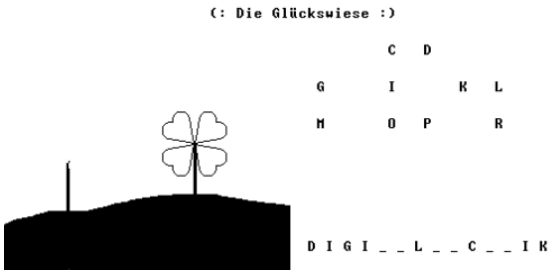
Da die Wiese-Gruppe ein relativ großes Objekt erstellen musste, konnte die „Pixel-Methode“, die die Kleeblatt-Gruppe angewendet hat, nicht mehr benutzt werden. Stattdessen haben sie die Wiese mit vielen unterschiedlich langen, horizontalen, übereinander geschichteten Linien gezeichnet. Für diese Linien gab es im JackOS schon einen vordefinierten Befehl, mit dem man diese zeichnen lassen konnte.

Außerdem gab es einen Gruppe, die den Startbildschirm programmierte. Auf diesem war unser Kurslogo abgebildet, und man hatte die Möglichkeit, das Wort für das Spiel einzugeben. Für das Logo arbeiteten wir hier mit den Befehlen einen Kreis und ein Rechteck zu zeichnen, denn unser Logo besteht eigentlich nur aus Kreisen, Rechtecken und Zahlen.



Startbildschirm

Um das Alphabet und die bereits genutzten Buchstaben anzuzeigen, gab es eine Alphabet-Gruppe, die dies übernahm. Hierfür zeigten wir das Alphabet in richtiger Reihenfolge auf den Bildschirm an, die jeweils nach der Benutzung eines Buchstabens gezeichnet werden.



Alphabet

Selbstverständlich musste unser Spiel auch ein Ende haben, und dafür programmierte eine Gruppe den Endbildschirm. Je nachdem, ob man gewonnen hat, ließ unser geschriebener Programmcode einen aus Kreisen und Strichen bestehenden traurigen beziehungsweise glücklichen Smiley zeichnen.

Die letzte Gruppe setzte die einzelnen Programmteile zusammen, damit es ein fließendes Programm ergab. Es bestand daraus, dass zuerst der Startbildschrirm angezeigt wird, bei dem man das zu erratende Wort eingeben sollte. Dann erschien das eigentliche Spiel mit der Glückswiese, den Kleeblättern, die anzeigten,

Endbildschirm

wie viele Leben man noch hat und das Alphabet. Wenn man das Wort herausfindet, bevor man alle Kleeblätter verliert, soll das Programm den fröhlichen Smiley anzeigen. Wenn man das Wort binnen der acht Leben nicht herausfindet, wird ein trauriger Smiley angezeigt.

Als das Spiel nun fertig war, haben wir es alle zusammen ausprobiert und waren sehr stolz darauf, was wir geschafft hatten. Deshalb haben wir dann auch bei der Abschlusspräsentation das Spiel vorgeführt. Dort gab es große Begeisterung bei den Zuhörern, und so spielten wir oft ein paar Runden mehr.

Exkursion

Helen Zwölfer

Wir saßen aber nicht nur vor unseren Rechnern, sondern verließen auch einmal das kleine Adelsheim. Denn während der Akademie gab es einen Tag, an dem alle Kurse eine Exkursion unternahmen.

Also machten wir uns mit dem Zug auf nach Stuttgart. Dort hatten wir zuerst zwei Stunden Freizeit, die wir natürlich gerne nutzten. Hier gab es für manche von uns den ersten langersehnten Kaffee seit mehr als einer Woche.

Nach dieser leckeren Kräftigung ging es weiter in das Institut für Mikroelektronik (IMS CHIPS). Dort hat man extra für uns eine Führung mit einem Vortrag organisiert, dem wir aufmerksam zuhörten. Obwohl wir schon früh morgens um neun losfuhren, kamen wir erst um halb neun Uhr abends wieder an, weil es so interessant war, dass wir alles noch genauer wissen wollten.

Was bekamen wir nun alles zu hören und zu sehen? Zuerst durften wir einen Reinraum von

außen sehen. Davor musste aber jeder von uns kleine Plastiktüten über seine Schuhe anziehen, die sehr witzig aussahen.

Ein Reinraum ist, wie der Name schon sagt, ein Raum, der völlig rein und somit möglichst staubfrei gehalten wird. Damit innen auch alles schön sauber bleibt, müssen die Wissenschaftler, die dort arbeiten, erst durch eine Schleuse gehen. In dieser werden sie mithilfe von Luftströmen grob gesäubert. Drei von uns durften diese Schleuse auch ausprobieren: Von allen Seiten kam Luft, und die Haare wurden nach oben gezogen, man kam sich vor wie in einem großen Föhn.

Danach muss man einen weißen Ganzkörperanzug aus Plastik anziehen und alle Wertsachen draußen lassen: Schlüssel, Geldbeutel und sogar die Handys. Aber diese wären sowieso nutzlos, weil es in dem Raum keinen Empfang gibt. Die Wände sind dazu viel zu dick. Bevor man dann endgültig im Reinraum ist, muss man noch eine letzte Schleuse durchqueren.

Die Wissenschaftler arbeiten dort mit lang ausgestreckten Armen, um so viel Abstand wie möglich zu den empfindlichen Chips zu haben. Damit kommen so wenige Fremdpartikel wie möglich an die Chips. Es muss auch dauernd die gleiche Temperatur herrschen, denn dies ist notwendig für das, was in diesem Raum entsteht: Die Mikrochips. So ein Chip ist nicht größer als ein kleiner Fingernagel, trotzdem besteht er aus über 100 Millionen Transistoren. Dies ist eine Zahl, die man sich kaum noch vorstellen kann. Noch weniger kann man sich vorstellen, wie die Transistoren auf den Chip gelangen. Aber wie werden diese Chips nun hergestellt?

Am Anfang gibt es nur Silizium. Aus diesem wird das Fundament der Chips erstellt, indem man es in hauchdünne Platten zerschneidet. Die Transistoren werden anschließend mit dem sogenannten Fotolackverfahren erzeugt:

Es wird zuerst die Siliziumplatte mit Aluminium beschichtet. Diese wird durch Rotation mit einer Schicht Fotolack besprüht. Jetzt wird eine Vorlage aus Glas, die mit Chrom beschichtet ist, auf die Platte gelegt. Die Platte wird nun belichtet, und durch die Vorlage dringt das Licht nur durch die richtigen Stellen. Der beschädigte Fotolack kann somit anschließend durch ein Säurebad entfernt werden.

Der Wafer, also die Siliziumplatte mit der Aluminiumschicht und der Fotolackschicht, wird nun geätzt, und nur die Stellen mit dem Fotolack bleiben unversehrt. Zum Schluss wird der Wafer noch gereinigt.

Dieses Fotolackverfahren wird nicht nur einmal gemacht, sondern ungefähr 10 Mal wiederholt. Jetzt haben wir viele Millionen Transistoren, aber es kommen nochmals circa sechs Schichten dazu. Diese sind Verbindungen, sodass aus den einzelnen Transistoren Logikgatter entstehen, denn die brauchen wir unbedingt für den Computer.

Da diese Chips nun sehr empfindlich sind, müssen sie in ein Gehäuse. Dazu werden die Chipanschlüsse durch sogenannte Bonddrähte mit dem Gehäuse verbunden. Zur Wahl stehen Keramikgehäuse und Plastikgehäuse. Beide haben ihre Vor- und Nachteile. Keramik ist teuer, aber schnell herstellbar. Deshalb nimmt man sie in Deutschland zur Testversion, denn da muss die Lieferzeit kurz sein. Die Plastikgehäuse sind billiger, aber es braucht seine Zeit, bis sie fertig sind. Diese werden hauptsächlich in Asien hergestellt.

Nach diesem spannenden Vortrag waren unsere Beine schon ganz schwer, weil wir fast die gesamte Zeit über stehen mussten. Zum Glück gab es eine kleine Trinkpause, in der wir auch die ganzen neuen Informationen verarbeiten konnten. Als krönenden Abschluss gab es noch einen kleinen Vortrag über Kamerachips. Das sind Chips, die so gebaut sind, das sie auch fotografieren können. In der Industrie gibt es dadurch auch bei sehr schlechten Lichtverhältnissen gute Bilder. Und es gibt keine Verzerrungen mehr, da alle Pixel vom selben Zeitpunkt ausgegeben werden. Nicht nur in der Industrie, sondern auch in der Medizin kann man diese Kamerachips benutzen. Diese ganz winzigen Chips können sich zum Beispiel in Endoskopen finden. Das sind Geräte, mit dem man das Innere von Menschen oder Tieren untersuchen kann.

Als wir am Stuttgarter Bahnhof ankamen, hatten wir aber unseren Zug zurück ins beschauliche Adelsheim verpasst. Da wir erwarteten,

dass es, wenn wir zurückkamen, schon viel zu spät für das Abendessen sein würde und wir außerdem noch eine Stunde Zeit hatten, schlugen wir uns die Bäuche mit Fast Food voll, und genossen noch ein letztes Mal die richtige Zivilisation. Als wir dann um halb neun Adelsheim erreichten, begrüßten uns schon freudig alle anderen Kurse und die Wiedersehensfreude nach der langen Trennung war groß.

Vielen Dank an das Institut für Mikroelektronik Stuttgart für die Organisation dieses tollen Tages. Insbesondere wollen wir uns bei Herrn Dr. Zimmermann, Herrn Strobel, Herrn Berndt sowie Herrn Futterer bedanken!

Danksagung

Die JuniorAkademie Adelsheim / Science-Academy Baden-Württemberg fand in diesem Jahr bereits zum 12. Mal statt. Daher möchten wir uns an dieser Stelle bei denjenigen bedanken, die ihr Stattfinden überhaupt möglich gemacht haben.

Die JuniorAkademie Adelsheim ist ein Projekt des Regierungspräsidiums Karlsruhe, das im Auftrag des Ministeriums für Kultus, Jugend und Sport, Baden-Württemberg und mit Unterstützung der Bildung & Begabung gGmbH Bonn für Jugendliche aus dem ganzen Bundesland realisiert wird. Wir danken daher dem Schulpräsidenten im Regierungspräsidium Karlsruhe, Herrn Prof. Dr. Werner Schnatterbeck, der Referatsleiterin Frau Leitende Regierungsschuldirektorin Dagmar Ruder-Aichelin, Herrn Jurke und Herrn Rechentin vom Ministerium für Kultus, Jugend und Sport sowie dem Koordinator der Deutschen Schüler- und JuniorAkademien in Bonn, Herrn Volker Brandt.

Die Akademie wurde finanziell in erster Linie durch die H. W. & J. Hector Stiftung, durch die Stiftung Bildung und Jugend sowie den Förderverein der Science-Academy unterstützt. Dafür möchten wir an dieser Stelle allen Unterstützern ganz herzlich danken.

Wie in jedem Jahr fanden die etwas über einhundert Gäste sowohl während des Eröffnungswochenendes und des Dokumentationswochenendes als auch während der zwei Wochen im Sommer eine liebevolle Rundumversorung am Eckenberg-Gymnasium mit dem Landesschulzentrum für Umwelterziehung (LSZU) in Adelsheim. Stellvertretend für alle Mitarbeiter möchten wir uns für die Mühen, den freundlichen Empfang und den offenen Umgang mit allen bei Herrn Oberstudiendirektor Meinolf Stendebach, dem Schulleiter des Eckenberg-Gymnasiums, besonders bedanken.

Zuletzt sind aber auch die Kurs- und KüA-Leiter gemeinsam mit den Schülermentoren und der Assistenz des Leitungsteams diejenigen, die mit ihrer hingebungsvollen Arbeit das Fundament der Akademie bilden. Ein besonderer Dank gilt an dieser Stelle Jörg Richter, der auch in diesem Jahr für die Gesamterstellung der Dokumentation verantwortlich war.

Diejenigen aber, die die Akademie in jedem Jahr einzigartig werden lassen und die sie zum Leben erwecken, sind die Teilnehmerinnen und Teilnehmer. Deshalb möchten wir uns bei ihnen und ihren Eltern für ihr Vertrauen ganz herzlich bedanken.