

Kurs 2 – Digitaltechnik



Vorwort

MICHAEL MATTES, KEVIN SOMMER

Was darf man erwarten, wenn zwölf ambitionierte junge Menschen für zwei Wochen gemeinsam an einem Projekt arbeiten? Die Ziele haben wir hoch angesetzt, und wir waren uns nicht sicher, ob wir sie erreichen würden. Dennoch haben es die zwölf Teilnehmerinnen und Teilnehmer geschafft, mit ihrer Begeisterung, ihrer Neugierde und ihrem Fleiß unsere Erwartungen zu sprengen. Bereits nach wenigen Tagen stand fest, dass wir die ursprüngliche Planung um weitere Inhalte erweitern mussten. Hier sieht man, dass Interesse und Fleiß die wichtigsten Voraussetzungen für erfolgreiche Arbeit sind. Spaß und Motivation entsteht als angenehmes Nebenprodukt. Es ist Außenstehenden nur schwer vermittelbar, wie motivierend die Science Academy auch für uns Kursleiter ist.

Mit großem Stolz übergeben wir nun das Wort an „unsere“ Kurs Digitaltechnik.

Teilnehmer

Alissa Es gibt niemanden, dem Alissa nicht auf Anhieb sympathisch ist. Mit ihrer offenen und freundlichen Art schaffte sie es immer wieder, uns ein Lächeln aufs Gesicht zu zaubern. Egal wie viel gerade schief lief, Alissa war immer zuversichtlich und schaffte es auch, diese Zuversicht auf uns zu übertragen. Gab es ein Problem und alle verfielen in Panik, war es letzten Endes immer Alissa, die Ruhe bewahrte und sich mit kühlem Kopf eine Lösung überlegte.

Annika half mit großer Begeisterung beim Aufbau der Teststrecke und kümmerte sich besonders um das Löten und Reparieren

der Lichtschranken. Beim Rotationsvortrag und der Endpräsentation konnte sie den Inhalt unseres Kurses anschaulich und genau erklären. Davon profitierten nicht nur die Zuhörer, sondern oftmals auch wir.

Daniela war im Kurs immer sehr engagiert. So konnte sie sich schnell für das Programmieren begeistern und brachte mit ihrer ruhigen und freundlichen Art Spaß und Freude in den Kurs. Bei den Präsentationen strahlte sie Ruhe und Gelassenheit aus, sodass man ihr immer gerne zuhörte.

Trotz Krankheit kam Daniela zum Dokumentationswochenende und arbeitete – wie auch schon während der Akademie im Sommer – beneidenswert konzentriert mit.

Dominik war jemand, der uns im Kurs immer wieder mit seinen Kenntnissen zur Elektronik zum Staunen brachte. So gut wie kein Schaltplan konnte ihm Verständnisprobleme bereiten, und auch beim schlimmsten Kabelsalat konnte er gut den Überblick bewahren.

Doch das war nicht sein einziges Talent. Bei unserem Bergfest und der Abschlussparty machte Dominik einen guten Job als DJ, weshalb er später von vielen nur noch „DJ Eisbär“ genannt wurde.

Jelle konnte im Kurs besonders gut unsere geliebten „Rainbow Cables“ fertigen und war beim Aufbau der Teststrecke engagiert dabei. Er interessierte sich aber auch für die Programmierung unseres Empfangsprogramms am Computer. Ansonsten war er ein ruhiger Vertreter unseres Kurses, weshalb es uns auffiel, wie gut und übersichtlich er die Themen bei der Präsentation vortrug.

Johanna Mit Johanna in unserem Kurs waren wir alle immer gut gelaunt. Probleme waren mit ihrer Hilfe immer schnell und mit großer Motivation gelöst. Am meisten konnte sie sich für das Programmieren des Mikrocontrollers begeistern, doch auch wenn man sie bei der Teststrecke brauchte, war sie sofort mit vollem Elan dabei. Arbeitete man mit Johanna in einer Gruppe zusammen, wurden Aufgaben immer mit guter Laune und viel Spaß gelöst.

Johannes verstand alles, was mit Programmieren zu tun hatte, sofort und war uns immer einen Schritt voraus – sogar in den Pausen klebten seine Finger an der Laptoptastatur. Er konnte aber auch gut erklären, sodass wir auch von seinem Geschick und Wissen über das Programmieren profitieren konnten. Ansonsten strahlte Johannes Ruhe und Gelassenheit aus.

Lisa war jederzeit mit viel Begeisterung beim Programmieren, Löten und Präsentieren dabei, und mit ihrer humorvollen Art entstand immer eine ausgelassene Stimmung. Gemeinsam mit ihr an Aufgaben oder Problemen zu knobeln, machte allen großen Spaß und man kam auch immer zu einem guten Ergebnis. Außerdem fand sie Freude am Verbessern der Dokumentation.

Nathanael konnten wir dank seiner wuscheligen Locken immer schnell im Akademiegewusel ausfindig machen. Trotz seiner Größe ist er einer der jüngsten Teilnehmer in unserem Kurs gewesen. Dies fiel allerdings nicht auf, da er ständig hilfreiche Ideen einbrachte.

Auch beim praktischen Arbeiten an der Teststrecke war er immer dabei, ob es nun darum ging, Platinen für die Sensoren zu löten oder die Teststrecke höhenverstellbar zu machen. Am liebsten testete er natürlich – wie wir anderen auch – immer wieder unsere Legoeisenbahn.

Simon interessierte sich im Kurs sowohl für das Programmieren des Mikrocontrollers, als auch für den Aufbau der Teststrecke. Daher half er immer und überall engagiert mit. Außerdem war er jederzeit gut gelaunt und fasziniert, was alles mit Hilfe von Platinen und anderen elektronischen Bauteilen möglich ist. Ihm gelang es auch gut, anderen sein Wissen in Präsentationen mitzuteilen.

Tönnis Wir alle können sehr dankbar sein, jemanden, der so viel Spaß versteht, in unserem Kurs dabeigehabt zu haben. Wie schwierig und kompliziert ein Vortrag auch war, mit Tönnis zusammen machte es immer Spaß diesen zu halten. Dafür sorgte unter anderem sein Lieblingswort „Jo“, ohne das er keinen Vortrag beginnen konnte,

sowie auch die Tatsache, dass er manchmal die Namen seiner Mitreferenten vergaß oder vertauschte.

Vincent war immer gut gelaunt, hat sich für alles interessiert was wir gerade machten und brachte immer gute Ideen ein. Besonders interessiert hat er sich für die Programmierung. Er half vor allem beim Erstellen der Präsentationen und machte außerdem selbstständig die komplette Excel-Tabelle zur Veranschaulichung der Ergebnisse.

Wenke Unsere Schülermentorin Wenke hat unsere Gruppe erst zu einem richtigen Team gemacht. Ob beim Teststreckenaufbau, dem Sportfest oder während der kleinen Pausen im Kurs, Wenke war immer am Motivieren, Witze machen und Helfen. Außerdem war sie die kreative Seele unseres Kurses und die coolste Schülermentorin, die man sich vorstellen kann. Leider konnte sie aufgrund ihres Auslandsaufenthaltes in Neuseeland beim Dokumentationswochenende nur über Skype in Adelsheim sein, aber in Gedanken ist sie sowieso immer bei uns.

Kevin Von Kevins umfassendem Wissen konnte während der Juniorakademie jeder profitieren und keine Frage blieb unbeantwortet. Ihm gelang es zusammen mit Michael, uns die kleinen Feinheiten der Digitaltechnik näher zu bringen. Außerdem leitete Kevin die von unseren Kursteilnehmern immer reichlich besuchte Tanz-KüA, bei der er auch viele Anfänger für den Paartanz begeistern konnte. In unserem Kurs begeisterte er sowohl Neulinge als auch Erfahrene für Mikrocontroller und Sensorik.

Michael schaffte es mit seiner guten Laune immer wieder, uns für den Lernstoff zu begeistern. Mit seinem unerschöpflichem Ideenreichtum gelang es ihm auch, unser Interesse für Projekte über die Akademie hinaus zu wecken. Mithilfe seines großen Wissens konnte er uns jede noch so erdenkliche Frage beantworten und nicht selten schaffte er es, uns die kompliziertesten Sachverhalte anschaulich und auf unterschiedliche Weisen zu erklären. Er ließ es sich auch nicht nehmen, uns durch Tanzeinlagen à la Michael immer wieder aufs Neue zu motivieren.

Einleitung

Im Leben braucht man immer Ziele. Doch welche Ziele setzt man sich als Gruppe, wenn man vor hat, sich die kommenden zwei Wochen mit Mikrocontrollern zu beschäftigen, aber absolut keine Ahnung hat, was genau auf einen zukommen wird?

Zum Glück standen unsere Kursleiter uns bei dieser Frage stets zur Seite, und schließlich hatten wir auch unsere Kursziele festgelegt. Als großes Ziel hatten wir ein Bewegungsexperiment in Form einer Legoeisenbahn auf einer schiefen Ebene vor Augen. Die Zeit und die Geschwindigkeit des herunterfahrenden Wagens sollten mithilfe eines Mikrocontrollers und Sensoren automatisch erfasst und auf einer Anzeige ausgegeben werden. Doch um auf dieses Ziel hinarbeiten, lernten wir zunächst den Mikrocontroller in einigen Theorieeinheiten kennen, um die Vorgänge in diesem besser verstehen zu können. Für das Verständnis des Experiments aus physikalischer Sicht, setzten wir uns auch mit der Bewegungslehre auseinander. Sehr intensiv lernten wir außerdem zu programmieren, um den Vorgang elektronisch steuern zu können.

Abschließend können wir völlig zufrieden auf unser Projekt zurückblicken. Die Themengebiete wurden fleißig erarbeitet und alle Ziele erreicht. Im Folgenden stellen wir die einzelnen Themengebiete vor.

Mikrocontroller

TÖNNIS TRITTLER, LISA MIRLINA

Mikrocontroller werden in der heutigen Welt zwar häufig benutzt, aber kaum einer weiß wirklich etwas mit diesem Wort anzufangen. Im Prinzip ist er wie ein kleiner Computer, nur ist seine Leistung wesentlich geringer. Trotzdem wird der Mikrocontroller sehr häufig in technischen Gegenständen, wie beispielsweise einer Computermouse oder einem Auto, verwendet. Er besteht aus einem Steuerwerk, einem Rechenwerk, einem Speicher und Anschlüssen bzw. Schnittstellen. Im Steuerwerk werden die einzelnen Befehlszeilen eines Programms abgearbeitet. Im Rechenwerk werden alle Rechen-

operationen ausgeführt. Der Speicher besteht aus folgenden Elementen:

ROM Aus dem ROM-Speicher können nur Daten gelesen, aber nicht gelöscht oder bearbeitet werden. Daten bleiben hier erhalten, auch wenn der Mikrocontroller gerade nicht an eine Stromversorgung angeschlossen ist. Hier wird das auszuführende Programm gespeichert.

RAM Der RAM-Speicher hingegen kann vom Programm aus beliebig oft überschrieben werden, er funktioniert aber nur mit Stromversorgung. Er wird auch Arbeitsspeicher genannt. Hier werden die Variablen abgelegt.

Die Anschlüsse bestehen aus Ports und deren Pins sowie einer seriellen Schnittstelle. Dies sind alles Bauteile, mit deren Hilfe der Mikrocontroller mit der Außenwelt kommunizieren – also Daten empfangen und versenden – kann.

In unserer Akademiezeit benutzten wir einen 8-Bit-Mikrocontroller. Ein Prozessor versendet beziehungsweise empfängt immer nur einzelne Ziffern hintereinander. Diese einzelnen Ziffern nennt man in der Digitaltechnik *Bits*. Ein Bit kann den Wert 0 (unwahr) oder 1 (wahr) annehmen. Acht Bits werden zu einem *Byte* zusammengefasst. Unser Mikrocontroller wird 8-Bit-Controller genannt, da er immer acht Bit – also ein Byte – auf einmal verarbeitet.

Bau der Teststrecke

DOMINIK SCHWAHN

Schon am Eröffnungswochenende versuchten wir uns erstmals am Löten. Wir übten uns im Umgang mit dem LötKolben, indem wir auf kleinen Platinen Bahnen aus Lötzinn löten. Nachdem wir die Übungsphase „bestanden“ hatten, durften wir uns in Zweierteams an das Löten der Microcontrollerplatine heranwagen. Dazu erklärten uns Kevin und Michael, wie diese Platine aufgebaut sein sollte und auf was wir achten mussten, sodass wir am Ende des Eröffnungswochenendes sechs funktionsfähige Mikrocontrollerplatinen fertiggestellt hatten. Außerdem machten wir uns am Eröffnungswochenende auch schon Gedanken, welche Art

von Sensoren wir für die Messung an unserer zukünftigen Teststrecke verwenden wollen und wie diese aufgebaut sein sollte.



Das Grundgerüst unserer Teststrecke

In den ersten Tagen der Sommerakademie gingen wir auf unser Projektziel ein und erstellten einen Plan, was genau zu tun war. Danach bildeten wir zwei Teams, die jeweils unterschiedliche Aufgaben übernahmen. Eine Gruppe konzentrierte sich auf das Programmieren, die andere auf den Aufbau der Teststrecke und der Sensorplatinen. Als unsere Controllerplatinen bereits fertig waren, fehlten uns noch die Sensoren für die Messungen. Für die ausgewählten Sensoren bauten wir auf einem Steckbrett jeweils eine kleine Testschaltung. Ein Steckbrett ist eine Art Platine, auf der man die Bauteile miteinander verbinden kann, ohne sie verlöten zu müssen, was den Vorteil hat, dass man die Bauteile danach nochmals verwenden kann.

Nach verschiedenen Tests entschieden wir uns letzten Endes dafür, dass Lichtschranken die passenden Sensoren für unsere Messwerterfassung seien. Sie sind zwar schwierig zu verbauen, aber für uns waren sie geeignet, da sie sehr genau und zuverlässig arbeiten. Wir erstellten acht passende Sensorplatinen. Auf jeder Platine verlöteten wir eine Lichtschranke, mehrere Widerstände und eine Buchse (Halterung) für unsere sogenannten „Rainbow Cables“. Diese

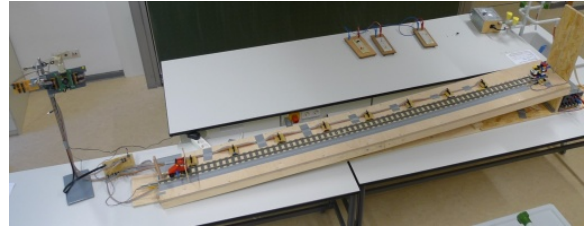
Kabel haben zehn Adern, also zehn getrennte Leitungen. Acht davon sind für die Datenübertragung (je Ader ein Bit) und die beiden übrigen Adern sind für die Stromversorgung zuständig. Nebenher beschäftigten wir uns mit dem Aufbau unserer Teststrecke.

Wir schraubten Legoschienen auf eine Holzplatte, die Kevin mit seinem Großvater in den Tagen vor der Akademie zu einer schiefen Ebene verbaut hatte. Des Weiteren mussten wir uns überlegen, wie wir den Winkel der schiefen Ebene messen können, damit wir später die Steigung der Teststrecke verändern und die Daten dann auch mit in die errechnete Beschleunigung des Autos einfließen lassen konnten. Auch brauchten wir den Winkel, um die theoretisch erreichbare Geschwindigkeit berechnen zu können. Dann brachten wir noch einen Elektromagneten am oberen Ende der Teststrecke an, um das Starten des Wagens zu automatisieren. Dies erreichten wir durch das Deaktivieren des Elektromagneten, woraufhin der Wagen nicht mehr festgehalten wird und losfährt.

Da der Mikrocontroller jedoch nicht genügend Energie für den Elektromagneten bereitstellen kann, benötigten wir eine zusätzliche Ansteuerung. Diese Schaltung besteht aus Transistoren, Widerständen und einer weiteren Buchse. Außerdem befindet sich dort auch der Input (Eingang) für die externe Stromversorgung des Magneten, sowie der Output (Ausgang), an dem der Magnet angeschlossen war. Somit waren unsere Hauptplatinen fertig. Nun benötigten wir noch eine Art Verteilerplatine. Mit dieser Platine verbanden wir alle Kabel der Sensoren, den Anschluss des Potentiometers und den Anschluss für unseren Elektromagneten. Außerdem erstellten wir auf der Verteilerplatine auch eine Schaltung für die Interrupts. Dazu wurde jeder Ausgang eines Sensors mit einem Eingang unseres Interruptbausteins verbunden. So liefen letztendlich acht Kabel von den einzelnen Sensoren, eine Interruptleitung, eine Elektromagnetsteuerleitung und eine Schiebepotentiometerleitung zu unserem Mikrocontroller.

Die Teststrecke war vor unserer Software fertig, sodass der Aufbaugruppe genügend Zeit blieb, ein bisschen mit der Teststrecke zu experimentieren – beziehungsweise zu spielen. Dabei

wurde uns schnell klar, dass wir noch einen Weg finden mussten, den Wagen am Ende der Teststrecke abzubremsen. Zuerst versuchten wir das mithilfe eines Luftballons. Dies funktionierte aber nicht, sondern führte zu einem Entgleisen des Wagens. Dann hatten wir die Idee, die Räder des Wagens mithilfe einer weiteren Schiene zu bremsen. Der Wagen fuhr so in eine Spalte, die sich immer mehr verengte und ihn somit sanft abbremste.



Die fertige Teststrecke mit Sensoren, Elektromagnet, Mikrocontroller und 7-Segment-Anzeige

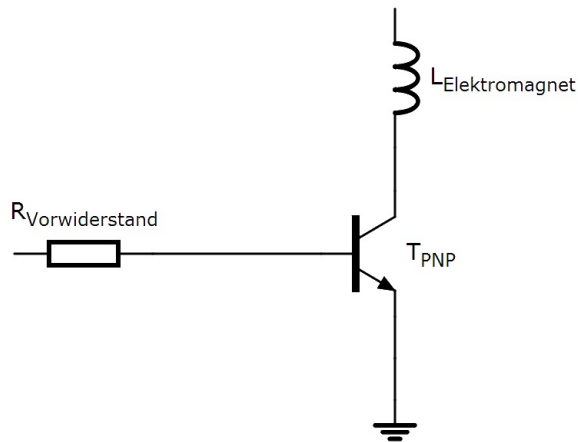
Zuletzt brauchten wir noch einen ganzen Kurs-tag, um den Versuchsaufbau zu testen, teilweise zu ändern und zu verbessern, bis die Teststrecke schließlich voll funktionsfähig war. Wir brachten nun unsere Teststrecke in unseren Präsentationsraum und bauten sie dort wieder auf, womit die Arbeit des Teststreckenaufbau-Teams beendet war. Nun konnte unser Experiment bei den Präsentationen anschaulich vorgeführt werden.

Transistoren

ALISSA KERN

Transistoren sind sehr wichtige elektrische Bauelemente. Sie sind die Grundbausteine von Prozessoren, also Recheneinheiten im Computer. Statt einzelner Transistoren in je einem eigenen Gehäuse werden zunehmend Chips verwendet, auf denen mehrere 100 Millionen Transistoren gemeinsam angeordnet sind. Der Name Transistor ist ein Kurzwort vom englischen *Transfer varistor*, was ihn als „durch Strom/Spannung steuerbaren Widerstand“ beschreibt. Bei unserer Teststrecke haben wir sie für den Halte-Elektromagneten benötigt. Dieser braucht nämlich so viel Strom, dass wir ihn nicht direkt an den Mikrocontroller anschließen konnten. Deshalb

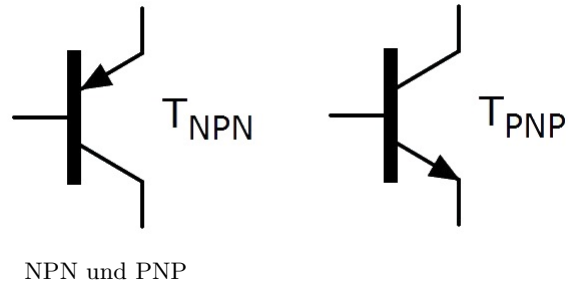
mussten wir einen Transistor als elektrischen Schalter einbauen.



Ansteuerung des Elektromagneten durch den Mikrocontroller

Allgemein werden Transistoren in zwei grundsätzliche Funktionsprinzipien unterteilt: Die Bipolartransistoren und die Feldeffekttransistoren (FETs). Wir haben uns hauptsächlich mit den Bipolartransistoren beschäftigt, da sie in der Digitaltechnik als elektrische Schalter verwendet werden. Sie sind aus drei übereinandergelegten speziell gefertigten Metallen hergestellt, sogenannten Halbleiterschichten. Diese sind mit drei metallischen Anschlüssen versehen, welche aus dem Gehäuse herausführen. Die drei Anschlüsse heißen Kollektor (C), Basis (B) und Emitter (E). Kollektor und Emitter sind an den beiden äußeren Halbleiterschichten, die Basis an der mittleren befestigt. Die Basis ist somit auch der Steuereingang des Transistors. Es wird je nach Polung zwischen NPN- und PNP-Transistoren unterschieden. Ein kleiner Stromfluß auf der Basis kann einen großen Stromfluß auf der Kollektor-Emitter-Strecke ermöglichen und somit als elektrischer Schalter dienen. Generell kann Strom vom Kollektor zum Emitter nur dann fließen, wenn die Basis positiver (bei NPN-Transistoren) bzw. negativer (PNP-Transistoren) als der Emitter ist.

Damit der entstehende Basisstrom nicht zu groß ist, darf die Basis nicht direkt mit dem Mikrocontroller verbunden werden, da man sonst die Basis-Emitter-Strecke kurzschließt. Deshalb muss man einen geeigneten Basiswiderstand benutzen.



NPN und PNP

Abgesehen von den „normalen“ Transistoren gibt es aber auch andere Transistortypen wie den Fototransistor. Dieser wird an der Basis nicht mit Strom, sondern mit Licht gesteuert. Er schaltet also, wenn er beleuchtet wird. Verwendet wird dieser Transistortyp beispielsweise in Lichtschranken.

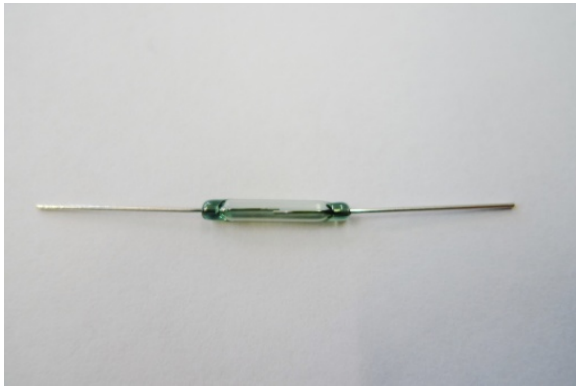
Sensoren

SIMON FEHLING

Um Messwerte mit einem Mikrocontroller zu erfassen, werden Sensoren benötigt. Die folgenden drei sind digitale Sensoren, d. h. sie können entweder eine logische 0 – also eine Spannung von 0 V – oder eine logische 1 – also eine Spannung von 5 V – ausgeben. Die folgenden drei Sensoren hatten wir in der Vorauswahl, um damit den vorbeifahrenden Wagen zu erkennen.

1. Reedkontakt

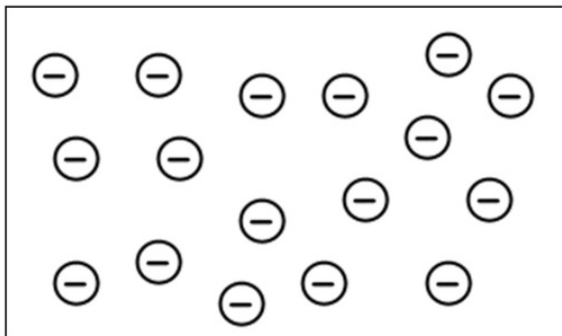
Ein Reedkontakt besteht aus einem Glasröhrchen, in dem sich zwei Kontaktdrähte befinden. Die Enden der Kontaktdrähte führen aus dem Glasröhrchen hinaus und bilden somit die beiden Anschlüsse des Reedkontakts. Die anderen beiden Enden der Kontaktdrähte liegen im Inneren des Glasröhrchens nah beieinander. Im Normalzustand berühren sie sich nicht. Kommt allerdings ein Magnet in die Nähe des Reedkontakts, werden die beiden Enden im Glasröhrchen zusammengezogen und stellen einen Kontakt her, sodass Strom durch den Reedkontakt fließen kann. Weil der Reedkontakt nur zwei Anschlüsse hat, ist er sehr einfach zu verbauen, hat aber den Nachteil, dass der Magnet die Kontaktdrähte nur von bestimmten Positionen aus aneinander ziehen kann.



Reed-Kontakt

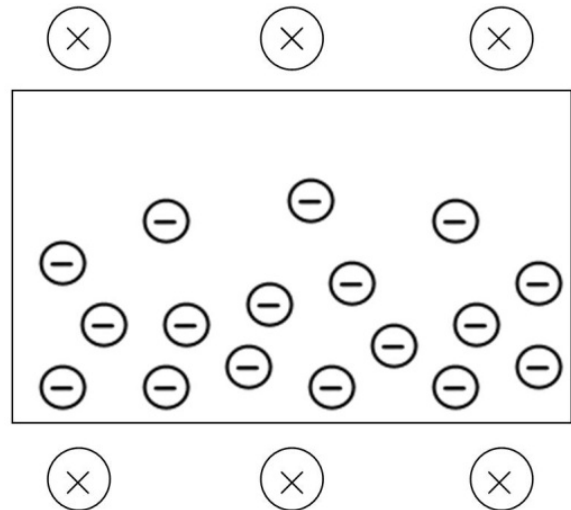
2. Hallsensor

Ein Hallsensor ist ein sehr kleiner Sensor, der sich den Hall-Effekt zu Nutze macht. Dieser basiert auf der Beobachtung, dass ein Magnet in der Nähe eines elektronendurchflossenen Leiters die Elektronen im Leiter leicht ablenkt. So entsteht zwischen dem oberen und dem unteren Rand des Leiters eine messbare Spannung (Hallspannung).



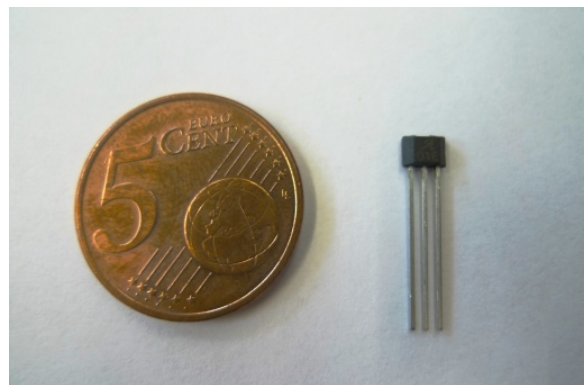
Elektronendurchflossener Leiter ohne Magnetfeld. Die Elektronen bewegen sich nach rechts und sind gleichmäßig verteilt.

Im Hallsensor spielt sich dieser Effekt im Hallgenerator ab. Da die dort entstehende Spannung sehr gering ist, wird sie verstärkt, bevor sie an den Schmitt-Trigger weitergegeben wird. Dieser entscheidet, ob die Spannung groß genug ist. Wenn ja, gibt er eine logische 1 aus, sonst eine 0. Dieser Wert wird an einen Transistor weitergegeben, der einen externen Stromkreis schließt, wenn eine logische 1 anliegt. Der Hallsensor hat den Vorteil, dass er deutlich genauer als der Reedkontakt ist. Es ist allerdings zu be-



Elektronendurchflossener Leiter in einem in die Zeichenebene gerichteten Magnetfeld. Die Elektronen werden nach unten abgelenkt.

achten, dass er nur schaltet, wenn der Magnet richtig gepolt ist.

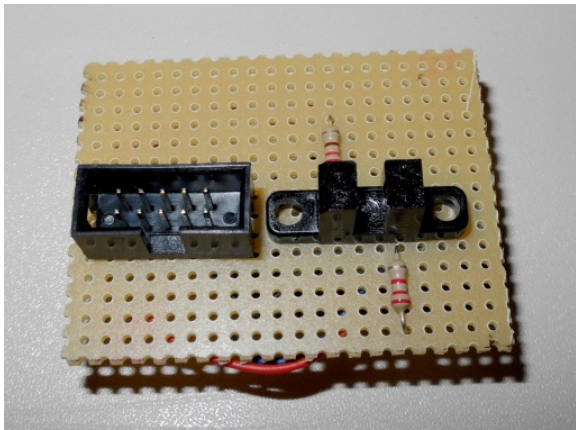


Hallsensor im Größenvergleich

3. Lichtschranke

Eine Lichtschranke besteht aus zwei Bauteilen, die über zwei getrennte Stromkreise versorgt werden. Das eine Bauteil ist eine Leuchtdiode, die Lichtstrahlen im Infrarotbereich aussendet. Das andere Bauteil ist ein Fototransistor, der einen Stromkreis schließt, wenn infrarotes Licht darauf fällt. Die beiden Bauteile sind durch eine Lücke voneinander getrennt. Im Normalzustand kann das Licht der Leuchtdiode durch die Lücke auf den Fototransistor fallen. Wird die Lichtschranke allerdings unterbrochen, in-

dem beispielsweise ein Plättchen in die Lücke zwischen den beiden Bauteilen gehalten wird, so ist auch der Stromkreis am Transistor unterbrochen, weil kein Licht mehr auf den Fototransistor fällt. Die Lichtschranke hat den Vorteil, dass man die Zeit ermitteln kann, die das Plättchen des Autos in der Schranke verbracht hat. Mit dieser „Verdunklungszeit“ lässt sich später die Beschleunigung berechnen. Die Lücke der Lichtschranke ist sehr schmal, weshalb man die Lichtschranke präzise einbauen muss.



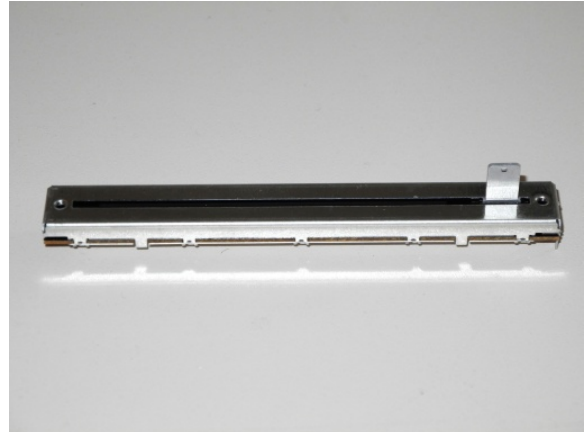
Lichtschranke auf Adapterplatine

Schiebepotentiometer

Zur Bestimmung des Steigungswinkels unserer Teststrecke, welchen wir für die Berechnung der theoretischen Geschwindigkeit brauchten, war ein analoger Sensor geeigneter, da er nicht nur eine logische 1 und eine logische 0 (also entweder 0 V oder 5 V), sondern beliebige Spannungen zwischen 0 V und 5 V ausgeben kann. Ein solcher analoger Sensor ist beispielsweise das Schiebepotentiometer. Bei diesem Sensor verändert sich je nach Position des Schieberegler die ausgegebene Spannung. Das Schiebepotentiometer ist ein regelbarer Widerstand. Er teilt die Spannung und liefert somit einen stufenlosen Wert zwischen 0 V und 5 V. Diese Spannung muss allerdings erst wieder in einen digitalen Wert umgewandelt werden, damit der Mikrocontroller sie verarbeiten kann. Das geschieht in einem Analog-Digital-Wandler.

Um nun durch das Schiebepotentiometer auf den Steigungswinkel unserer Teststrecke zu kommen, bestimmten wir im Voraus für die

einzelnen Steigungswinkel den Wert, den das Schiebepotentiometer beim jeweiligen Winkel ausgibt. In unserem Programm ordneten wir dann einfach den vorher bestimmten Werten die Steigungswinkel zu, sodass der Winkel direkt in Excel angegeben werden konnte.



Schiebepotentiometer

Logikgatter

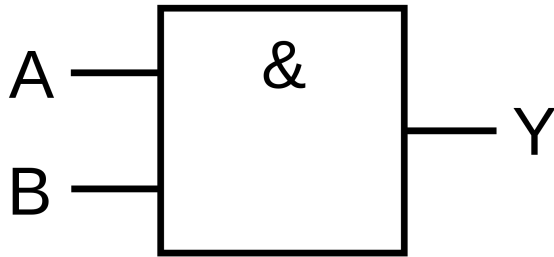
JELLE KÜBLER

Im Alltag benutzen wir häufig logische Sätze wie zum Beispiel: „Um die Wäsche zu waschen, brauche ich Wasser UND Seife!“, „Um etwas zu bezahlen, brauche ich eine Kreditkarte ODER Bargeld!“, „Wenn es dunkel ist, ist es NICHT hell!“.

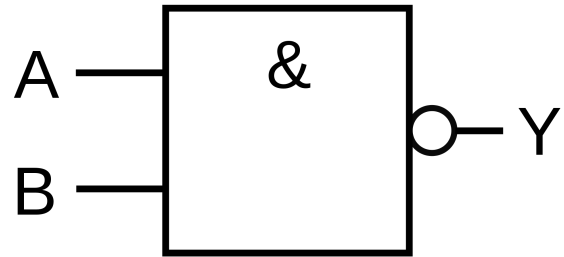
Diese logischen Aussagen beschreiben einen Sachverhalt. Somit kann man ihnen – im Gegensatz zu Fragesätzen, Ausrufen, Wünschen, etc. – einen Wahrheitswert zuordnen, um mit ihnen die Frage nach der Wahrheit sinnvoll zu stellen. Während der ersten Tage der Akademie haben wir uns mit Logikgattern beschäftigt. Das sind elektronische Bausteine, die ein oder mehrere Eingangssignale – logisch – zu einem Ausgangssignal verarbeiten. Wir haben uns mit verschiedenen Gattern beschäftigt.

Das AND-Gatter (UND)

Am Ausgang dieses Gatters erhält man nur eine logische 1, wenn an *beiden* Eingängen logische 1en anliegen.



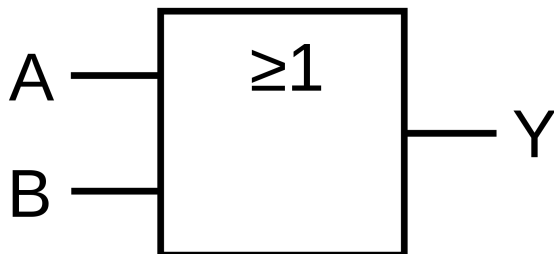
Schaltbild eines AND-Gatters



Schaltbild eines NAND-Gatters

Das OR-Gatter (ODER)

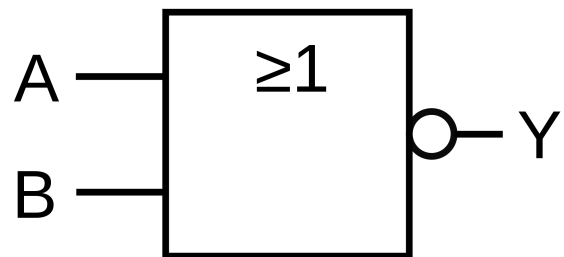
Wenn an *mindestens* einem der beiden Eingänge eine logische 1 anliegt, dann liegt am Ausgang auch eine logische 1 an.



Schaltbild eines OR-Gatters

Das NOR-Gatter (ODER, NICHT)

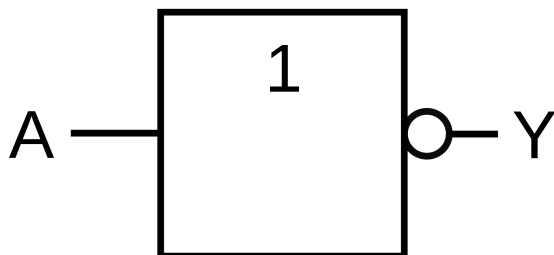
Wenn an einem der beiden Eingänge eine logische 1 anliegt, dann liegt eine logische 1 vor, die durch das NOT-Gatter in eine logische 0 invertiert wird.



Schaltbild eines NOR-Gatters

Das NOT-Gatter (NICHT)

Dieses Gatter invertiert den am Eingang anliegenden Wert. Das heißt, dass aus einer logischen 0 eine 1 wird oder aus einer logischen 1 eine 0 wird.



Schaltbild eines NOT-Gatters

Das NAND-Gatter (UND, NICHT)

Dieses Gatter erzeugt nur eine logische 1, wenn an beiden Eingängen logische 1en anliegen. Diese wird jedoch durch das NOT-Gatter in eine logische 0 invertiert.

Es gibt noch viele weitere Gatter, wir beließen es allerdings bei den genannten.

An den Eingängen der Logikgatter liegen elektrische Spannungen an. Diese repräsentieren den logischen Zustand als 0 und 1. Alle möglichen Ausgangswerte der Logikgatter und die dazugehörigen Eingangswerte, werden in einer sogenannten Wahrheitstabelle aufgelistet. In der Wahrheitstabelle des AND-Gatters steht beispielsweise, dass eine 0 und eine 1 als Eingangswerte zu einer logischen 0 als Ausgangswert führen. In einem NAND-Gatter wird (bei gleichen Eingangswerten) die 0 noch invertiert, und in der Wahrheitstabelle steht als Ausgangswert eine logische 1. In der Wahrheitstabelle des OR-Gatters steht für die Eingangswerte 0 und 1 eine logische 1 als Ausgangswert. Für dieselben Werte im NOR-Gatter steht als Ausgangswert eine logische 0, da das Ergebnis des OR-Gatters noch invertiert wird.

Auch in unserer Teststrecke haben wir einige Logikgatter eingebaut. Da wir acht Lichtschranken auf der Teststrecke angebracht hatten und

E1	E2	AND	OR	NAND	NOR
1	1	1	1	0	0
1	0	0	1	1	0
0	1	0	1	1	0
0	0	0	0	1	1

Wahrheitstabelle für die Gatter AND, OR, NAND sowie NOR anhand der Eingänge E1 und E2

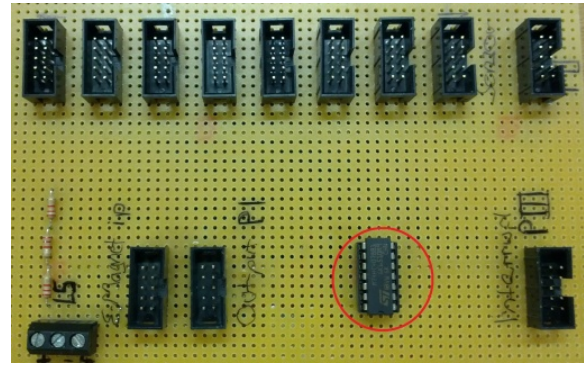
der Mikrocontroller erkennen musste, wenn eine von ihnen durch den Wagen unterbrochen wird, brauchten wir ein OR-Gatter (ODER), damit sich das Signal – egal welche Lichtschranke unterbrochen war – am Logikgatter änderte. Aus zwei Gründen haben wir jedoch kein einfaches OR-Gatter verwendet, sondern ein NOR-Gatter (ODER, NICHT). Zum einen kann man heute keine Logikgatter mehr kaufen, in die kein NOT-Gatter integriert ist. Das heißt jedes Gatter hat zwei Ausgänge, bei einem NOR-Gatter hätte man also einen normalen Ausgang (OR) und einen invertierten (NOR). Der zweite Grund, warum wir ein NOR-Gatter brauchten, ist, dass das Unterbrechen der Lichtschranke einen Interrupt am Mikrocontroller auslösen sollte. Mit einem einfachen OR-Gatter würde das Ausgangssignal beim Unterbrechen der Lichtschranke von einer logischen 0 auf eine logische 1 wechseln und hätte eine „steigende Flanke“ ($0 \rightarrow 1$ = steigende Flanke), da die am Ausgang anliegende Spannung von zwischen 0 V und 0,8 V auf über 2 V steigt. Da ein Interrupt an unserem Mikrocontroller nur durch eine „fallende Flanke“ ausgelöst werden kann, brauchten wir ein NOR-Gatter, damit das Ausgangssignal von einer logischen 1 auf eine logische 0 wechselt ($1 \rightarrow 0$ = fallende Flanke).

Zahlensysteme

ANNIKA BUTLER

Es gibt viele verschiedene Arten, Zahlen darzustellen. Wir Menschen verstehen die Zahl 138 im Dezimalsystem, das heißt, wir rechnen $138 = 1 \cdot 10^2 + 3 \cdot 10^1 + 8 \cdot 10^0$.

Ein Computer versteht Zahlen im Dualsystem, er „sieht“ die Zahl 1011 (= 11 im Dezimalsystem) als $1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$.



Das NOR-Gatter (Mitte unten) auf der Verteilerplatine.

Umwandlung von dezimal nach dual

Es ist für die Programmierung wichtig, Dezimalzahlen in Dualzahlen – auch binäre Zahlen genannt – umwandeln zu können. Es gibt zwei Varianten, wie man die Zahlen umrechnen kann.

Bei der ersten Methode wird geschaut, welche größtmögliche Zweierpotenz von der Dezimalzahl subtrahiert werden kann. Ist es möglich die Zahl zu subtrahieren, ohne einen negativen Wert zu erhalten, setzt man eine 1, wenn nicht eine 0.

816	−512 (2^9)	1
304	−256 (2^8)	1
48	−128 (2^7)	0
48	−64 (2^6)	0
48	−32 (2^5)	1
16	−16 (2^4)	1
0	−8 (2^3)	0
0	−4 (2^2)	0
0	−2 (2^1)	0
0	−1 (2^0)	0

Umrechnung von dezimal 816 in binär mit erster Methode

Die binäre Zahl, die man erhält, ist 1100110000, was der Dezimalzahl 816 entspricht.

Eine zweite Methode zur Umwandlung einer Dezimalzahl in eine Dualzahl ist das Teilen der Dezimalzahl durch zwei. Wenn die Zahl nach der Teilung einen Rest hat, wird eine 1 gesetzt. Gibt es keinen Rest, setzt man eine 0. Bei der Division durch zwei sind logischerweise

nur 1 oder 0 als Rest möglich. Das Ergebnis muss noch von unten nach oben gelesen werden, damit die Dualzahl stimmt.

$$\begin{array}{r|l}
 816 : 2 = 408 & 0 \\
 408 : 2 = 204 & 0 \\
 204 : 2 = 102 & 0 \\
 102 : 2 = 51 & 0 \\
 51 : 2 = 25,5 & 1 \\
 25 : 2 = 12,5 & 1 \\
 12 : 2 = 6 & 0 \\
 6 : 2 = 3 & 0 \\
 3 : 2 = 1,5 & 1 \\
 1 : 2 = 0,5 & 1
 \end{array}$$

Umrechnung von dezimal 816 mit zweiter Methode

Umwandlung von dual nach dezimal

Im Gegenzug können Binärzahlen auch in Dezimalzahlen umgewandelt werden. Das funktioniert, indem man jede Ziffer mit der zur jeweiligen Stelle gehörenden Zweierpotenz multipliziert.

Beispiel: Umwandlung der Dualzahl 1001 in eine Dezimalzahl.

$$1001 = 1 \cdot (2^3) + 0 \cdot (2^2) + 0 \cdot (2^1) + 1 \cdot (2^0) = 1 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 = 9$$

Rechnen im Dualsystem

Folgende Regeln gelten bei der Addition:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \text{ mit Übertrag } 1$$

$$\begin{array}{r|l|l}
 & 0 & 1 & 1 & 0 & & 6 \\
 + & 0 & 0 & 1 & 1 & & 3 \\
 & 1 & 1 & & & & \\
 \hline
 = & 1 & 0 & 0 & 1 & & 9
 \end{array}$$

Beispiel für binäre Addition

Folgende Regeln gelten bei der Subtraktion:

$$1 - 0 = 1$$

$$1 - 1 = 0$$

$$0 - 0 = 0$$

$$0 - 1 = 1 \text{ mit Übertrag } 1$$

$$\begin{array}{r|l|l}
 & 1 & 0 & 1 & 1 & 1 & & 23 \\
 - & & 1 & 0 & 1 & 1 & & 11 \\
 & 1 & & & & & & \\
 \hline
 = & 0 & 1 & 1 & 0 & 0 & & 12
 \end{array}$$

Beispiel für binäre Subtraktion

Darstellung negativer Zahlen im Dualsystem

Negative Dualzahlen werden im sogenannten 2er-Komplement dargestellt. Um das 2er-Komplement zu bilden, werden zuerst die Ziffern 1 und 0 vertauscht, sie werden also komplementiert. Dabei ist es wichtig zu erwähnen, dass jede Zahl im 2er-Komplement mit der vollen Anzahl an Ziffern geschrieben wird – bei unserem 8-Bit-Mikrocontroller also 8 Ziffern – und die vorderste Ziffer als Vorzeichen gilt. Positive Dualzahlen haben immer eine 0 als erste Ziffer, negative Zahlen eine 1.

Die Umwandlung von +13 in -13 kann beispielsweise wie folgt erreicht werden.

Komplementierung: 0000 1101 (13) wird zu 1111 0010

Zu dieser Zahl muss nun 1 addiert werden, sodass -13 binär als 11110011 dargestellt wird.

$$\begin{array}{r|l|l}
 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\
 + & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
 \hline
 = & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1
 \end{array}$$

Addition von 1 zur Bildung des 2er-Komplements von +13

Man sieht, dass -13 bei gleicher Vorgehensweise wieder zu +13 wird.

Zunächst invertiert man 1111 0011 und erhält 0000 1100. Anschließend addiert man 1 dazu, sodass das Ergebnis 0000 1101 tatsächlich +13 ergibt.

$$\begin{array}{r|l|l}
 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
 + & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
 \hline
 = & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1
 \end{array}$$

Addition von 1 zur Bildung des 2er-Komplements von -13

Darstellung von Zahlen im Hexadezimalsystem

Beim Programmieren werden allerdings nicht nur Dualzahlen, sondern häufig auch Hexadezimalzahlen verwendet. Im Hexadezimalsystem (16er-System) wird jede Stelle von 0 bis F (entspricht der 15) gezählt.

hex	dez	hex	dez
0	0	10	16
1	1	11	17
2	2	12	18
3	3	13	19
4	4	14	20
5	5	15	21
6	6	16	22
7	7	17	23
8	8	18	24
9	9	19	25
A	10	1A	26
B	11	1B	27
C	12	...	...
D	13	1F	31
E	14	20	32
F	15	21	33

Zählen im Hexadezimalsystem

Datentypen und Operatoren

JOHANNA DINKEL

Datentypen

Um Variablen in einem Programm zu definieren, stehen in der Programmiersprache C unterschiedliche Datentypen zur Verfügung. Jeder Datentyp hat einen unterschiedlichen Bedarf an Speicherplatz. Bei den Datentypen unterscheidet man in Datentypen für natürliche Zahlen (beispielsweise unsigned short) und Datentypen für ganze Zahlen (beispielsweise short).

Operatoren

Ein Programm besteht aus einer Reihe von Anweisungen, die der Mikrocontroller Schritt

Name	Speicherbedarf	Wertebereich
char	1 Byte	−128 bis +127
unsigned char	1 Byte	0 bis 255
short	2 Byte	−32768 bis +32767
unsigned short	2 Byte	0 bis 65535
long	4 Byte	−2 Mrd. bis +2 Mrd.
unsigned long	4 Byte	0 bis 4 Mrd.

Aufstellung verschiedener Datentypen

für Schritt abarbeitet. Für mathematische Berechnungen werden sogenannte Operatoren verwendet. In der Programmiersprache C gibt es verschiedene Arten von Operatoren. Die Operatoren, die wir in den zwei Wochen gelernt und verwendet haben, waren die Rechenoperatoren samt Modulo, die Bit-Operatoren und die Schiebeoperatoren.

Rechenoperatoren

Unter den Rechenoperatoren, auch arithmetische Operatoren genannt, versteht man die Addition, Subtraktion, Multiplikation und Division von in ganzen Zahlen. Natürlich verwendet man beim Programmieren keine Wörter, sondern einzelne Zeichen, wie sie auch in der Mathematik verwendet werden. Für die Addition steht ein +, für die Subtraktion ein −, für die Multiplikation ein * und für die Division ein /.

Modulo (mod)

Der Modulo-Operator beschreibt den Rest der Division ganzer Zahlen. Bei einer solchen Division entsteht als Ergebnis ein Divisionsergebnis und ein Divisionsrest. Das Modulo liefert genau diesen Rest. Dieser Operator kann nur ganze Zahlen verarbeiten und im Programm wird hierfür das Prozentzeichen % geschrieben. Beispiel: $9 \bmod 4 = 1$

in C: `9 % 4`

dies entspricht $9 : 4 = 2 \text{ Rest } 1$

Bit-Operatoren

Mit den Bit-Operatoren kann man einzelne Bits manipulieren. Dabei werden zwei Operanden bitweise verknüpft. Die gängigsten Bit-Operatoren sind das AND (&) und das OR (|). Beim Bit-Operator AND ist das Ergebnis-Bit 1, wenn die Operanden-Bits beide 1 sind. Ansonsten ist das Ergebnis 0. Beim Bit-Operator OR ist das Ergebnis-Bit 1, wenn mindestens eines der Operanden-Bits 1 ist.

Beispiel: $0000\ 0111 \& 1000\ 0100 = 0000\ 0100$
 $0000\ 0111 | 1000\ 0100 = 1000\ 0111$

Darüber hinaus haben wir noch das exklusiv ODER bzw. XOR (^) verwendet. Damit das Ergebnis-Bit 1 wird, müssen die beiden Operanden-Bits unterschiedlich sein, d. h. entweder 1 und 0 oder 0 und 1.

Beispiel: $0000\ 0111 \wedge 1000\ 0100 = 1000\ 0011$

Als letzten Bit-Operator verwendeten wir die Bit-Umkehrung (~). Das heißt, dass alle Nullen zu Einsen werden und umgekehrt.

Beispiel: $\sim 5_{10} (= \sim 00000101_2 = 11111010_2)$

Schiebeoperatoren

Die Schiebeoperatoren, die wir uns angeschaut haben, waren der Bit-Shift nach links (<<) und der Bit-Shift nach rechts (>>). Das heißt, dass eine Zahl um n Stellen nach links oder rechts verschoben wird. Wenn eine ganze positive Zahl um n Bits nach links verschoben wird, entspricht das einer Multiplikation mit 2^n .

Beispiel: $5 << 3 \rightarrow 40$

Binär: $0000\ 0101 << 3 \rightarrow 0010\ 1000$

Wenn eine ganze positive Zahl um n Bits nach rechts verschoben wird, entspricht das einer Division durch 2^n .

Beispiel: $112 >> 2 \rightarrow 28$

Binär: $0111\ 0000 >> 2 \rightarrow 0001\ 1100$.

C-Code vs. Assembler

JOHANNES GALINSKY

C und Assembler sind Programmiersprachen. Jeder Mikrocontroller oder Computer, welcher letztendlich auch nur aus mehreren Mikrocontrollern besteht, versteht nur Assembler. Da Assembler aber sehr aufwändig ist, hat man die

Hochsprachen erfunden. C ist eine davon. Ein Compiler (= „Übersetzer“) wandelt das Programm vor der Ausführung in Assembler um.

Natürlich haben die beiden Sprachen ihre Vor- und Nachteile. Der größte und wichtigste Vorteil von C-Code ist, dass er der menschlichen Sprache näher ist, wodurch er übersichtlicher ist und die Arbeit erleichtert. Eine kurze Rechenoperation, die in C beispielsweise gerade mal eine halbe Zeile lang ist, kann in Assembler mehrere Seiten in Anspruch nehmen. Es gibt viele vorgefertigte Programmblöcke (Bibliotheken), die man benutzen kann, um sich Arbeit zu ersparen. Allerdings ist C dafür weniger maschinennah und durch die Übersetzungen des Compilers werden manchmal unnütze Programmzeilen eingefügt, welche Speicherplatz rauben und das Programm verlangsamen.

Bei Assembler ist der große Vorteil, dass es am maschinennächsten ist. Dadurch sind die geschriebenen Programme deutlich schneller, und ein effizientes Nutzen der Hardware ist möglich. Die Arbeit des Systems lässt sich wesentlich besser nachvollziehen, wodurch man dann Details besser und leichter optimieren kann. Die in Assembler geschriebenen Programme verbrauchen in der Regel auch weniger Speicherplatz.

Allerdings entstehen bei der Nutzung von Assembler auch eine Reihe von Nachteilen. Der Code nimmt schnell gigantische Ausmaße an und wird extrem unübersichtlich, was wiederum die Entwicklung von reinen Assembler-Programmen sehr aufwändig macht. Das Programm ist an die Architektur der Hardware gebunden, das heißt, beim Ändern der Hardware muss das Programm neu bzw. umgeschrieben werden. Bei einem anderen Mikrocontroller muss man eventuell erst einen neuen Befehlsatz erlernen.

Eine Zeile wie

```
a=b/c;
```

sieht in Assembler wesentlich komplizierter aus.

Aufgrund der verschiedenen Vor- und Nachteile haben wir uns im Kurs zuerst Assembler angeschaut und erlernt, aber letztendlich in C programmiert.

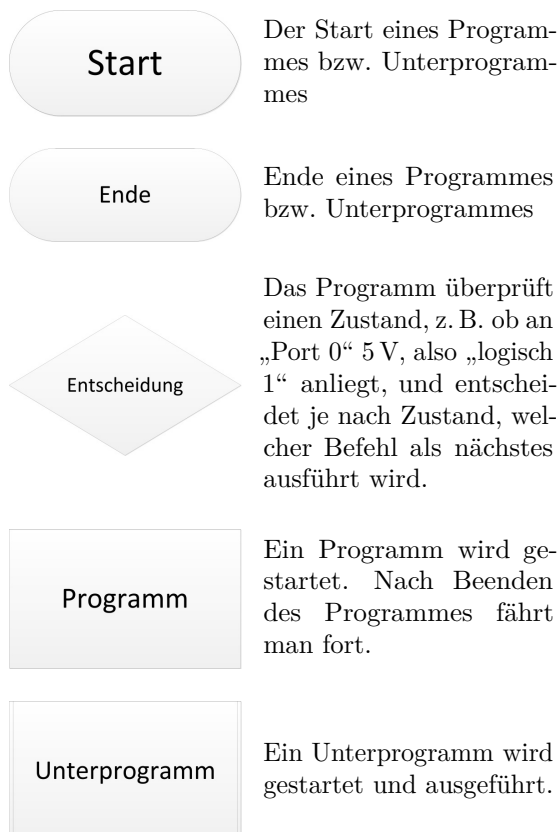
C:0x0933	AB49	MOV	R3,0x49
C:0x0935	AA48	MOV	R2,0x48
C:0x0937	A947	MOV	R1,0x47
C:0x0939	A846	MOV	R0,0x46
C:0x093B	AF0B	MOV	R7,0x0B
C:0x093D	AE0A	MOV	R6,0x0A
C:0x093F	AD09	MOV	R5,0x09
C:0x0941	AC08	MOV	R4,0x08
C:0x0943	1209BB	LCALL	C:09BB

... 50 weitere Zeilen vervollständigen die einfache C-Anweisung $a=b/c$.

Programmablaufplan

TÖNNIS TRITTLER

Nachdem wir das ganze Programm geschrieben hatten, mussten wir für die Abschlusspräsentation und die Dokumentation noch einen Weg finden, das Programm den Zuschauern und Lesern auf eine anschauliche und informative Weise vermitteln zu können. Wir wählten hierfür den Programmablaufplan, kurz PAP. Ein PAP besteht im Wesentlichen aus folgenden Elementen:



——— Pfeil ——— Verbindung zum nächsten Element.

Am Beispiel des Zustands „Messen“ kann man die verschiedenen Elemente erklären. Mit dem Icon „Start“ beginnt der Zustand „Messen“. Das Programm fragt danach wieder und wieder bei Sensor 0 ab, ob er aktiviert ist. Dieses andauernde Abfragen nennt man Polling. Sobald Sensor 0 aktiviert ist, speichert der Mikrocontroller die vergangene Zeit seit dem Start. Sobald der Sensor 0 nicht mehr aktiviert ist, speichert der Mikrocontroller die Zeit des Austritts. Dieses Vorgehen wird noch mit sieben weiteren Sensoren wiederholt. Danach endet der Zustand „Messen“ mit dem „Ende“-Icon.

Interrupts

JOHANNES GALINSKY

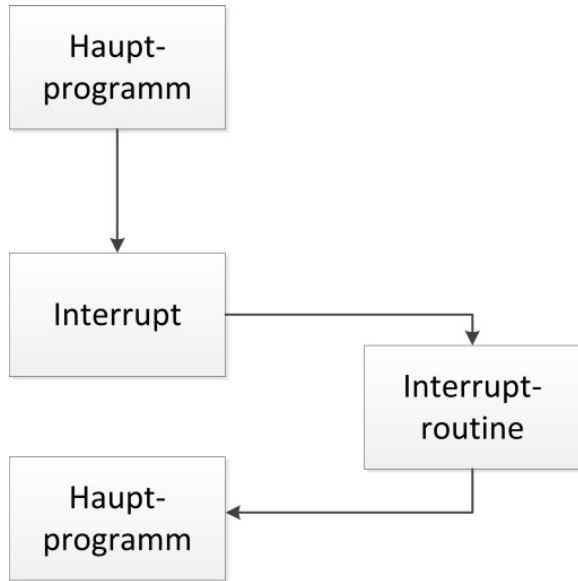
Interrupts braucht man, wenn man wissen will, ob bestimmte Pins ihren Wert ändern, dies aber nicht ständig im Programm kontrollieren will, da es Zeit raubt oder zu aufwändig in der Programmierung ist. Interrupts sind praktisch, da sie den normalen Programmablauf kurz unterbrechen. So können in der Interrupt-Routine dringende Ereignisse bearbeitet werden.

Immer wenn ein bestimmtes Ereignis eintritt, kann ein Interrupt ausgelöst werden. In diesem Fall unterbricht der Mikrocontroller das, was er gerade macht, und führt bestimmte Programmzeilen aus. Wenn diese dann fertig ausgeführt wurden, kehrt der Mikrocontroller zu seiner ursprünglichen Programmzeile zurück und führt dort das ursprüngliche Programm weiter aus.

Interrupts werden durch bestimmte Bits freigegeben, damit kein ungewollter Interrupt ausgelöst wird. Es gibt verschiedene Arten von Interrupts. Je nach Interrupt werden bestimmte Programmzeilen ausgeführt. Je nach Bauart des Mikrocontrollers kann während eines Interrupts kein neuer Interrupt ausgelöst werden, oder nur ein Interrupt mit höherer Priorität, aber keiner mit niedrigerer.

Nachfolgend eine Aufstellung der wichtigsten Interrupts und wie sie ausgelöst werden.

Externer Interrupt wird durch Ändern eines externen Signals an einem bestimmten Pin



Unterbrechung des Programmablaufs durch eine Interrupt-Routine

– bei uns die Lichtschranken – ausgelöst.

Serieller Interrupt Auslösung durch Empfangen oder Senden von Daten an der seriellen Schnittstelle.

Timer-Interrupt wird ausgelöst, sobald eine bestimmte Zeit abgelaufen ist oder im Counter-Modus eine bestimmte Anzahl gezählt wurde. Wir haben ihn für die Zeitmessung verwendet.

AD-Wandler-Interrupt Auslösung, sobald der Analog-Digital-Wandler mit einer Umwandlung fertig ist. Bei uns wird er für die Winkelmessung verwendet.

Timer

LISA MIRLINA

Der Timer ist ein Teil der Hardware eines Mikrocontrollers. Er nimmt ein oder auch mehrere Zählregister darin ein. In diesen Registern speichert er einen Wert, also eine Zahl, und erhöht diesen Wert fortlaufend um eins. Das Fachwort beim Programmieren für das Erhöhen ist „inkrementieren“.

Der Timer kann unterschiedliche Funktionen übernehmen. Als Timer, beziehungsweise Zeitzähler oder Zeitgeber, erhöht er den Wert nach jedem Takt um eins, arbeitet also wie eine Uhr.

In diesem Fall kann man die Genauigkeit des Quarzes (= Taktgeber im Mikrocontroller) nutzen und so den Timer unabhängig vom restlichen Programmablauf hochzählen lassen. Als sogenannter Counter, oder auch Zähler, erhöht er den Wert nach jedem externen Signal um eins und kann so bestimmte Ereignisse zählen, wie zum Beispiel das Betätigen eines Schalters. Bei unseren ersten Übungen mit dem Timer beim Programmieren verwendeten wir die Funktion des Zählers. Wir schrieben ein Programm, durch das sich bei jeder Betätigung eines Schalters die Zahl auf einer Anzeige immer um eins erhöhte. Später verwendeten wir ihn aber in unserem physikalischen Experiment als Zeitgeber. Er maß die Zeit, die zwischen dem Losfahren des Wagens und dessen Ankunft bei den einzelnen Zwischenstationen vergangen war. Diese konnte schließlich auf einer Anzeige ausgegeben werden.

Der Mikrocontroller, den wir verwendeten, besitzt drei verschiedene Timer: Timer 0, Timer 1 und Timer 2, welche sich durch ihre verschiedenen Modi und Einstellungen voneinander unterscheiden. Jeder Timer besteht aus zwei 8-bit-Registern (THx und TLx). In einem bestimmten Modus kann man diese zusammen als einen 16-bit Timer verwenden. Zunächst wird in diesem Fall das TLx-Register inkrementiert, bis es bei seinem Endwert, der bei einem 8-bit-Register bei 255 liegt, angekommen ist.

Daraufhin findet ein so genannter Überlauf statt. Der Maximalwert des Registers ist also erreicht. Bei weiterem Inkrementieren würde der Wert wieder auf null zurückfallen. Deshalb wird nun das THx-Register um eins erhöht, das andere dabei wieder auf null gesetzt und wieder wie davor fortlaufend inkrementiert, sodass wieder und wieder Overflows des TLx-Registers und Erhöhungen des THx-Registers stattfinden. Das passiert so lange, bis sich auch im THx-Register ein Überlauf ereignet. Sofern der Mikrocontroller passend eingestellt wurde, kann jetzt ein Interrupt stattfinden.

Bei unserem Experiment verwendeten wir einen anderen Modus, den „Auto-Reload“. Bei diesem wurde nur das TLx-Register erhöht und im THx-Register gleichzeitig der Reload-Wert

gespeichert, bei dem das TLx-Register nach einem Überlauf wieder anfangen sollte. Mithilfe dieses Modus kann man den Timer also nicht nur von 0–255 zählen lassen, sondern auch von 6–255, was bei Zeitmessungen sehr von Vorteil sein kann. Zählt er nämlich von 6 bis 255, dann zählt er vor einem Überlauf 250 mal hoch. Da der Timer mit einer Frequenz von 1 MHz zählt, vergehen pro Überlauf 250 μ s.

Der Timer besitzt noch zwei Steuerregister, nämlich das TMOD und das TCON. TMOD steht für Timer Modus und TCON für Timer Control. Im Register TMOD kann man auswählen, ob man die Anwendung als Timer oder als Counter verwenden und welchen Modus man nutzen möchte. Im Register TCON dagegen wird der Startbefehl für den Timer gegeben.

Der Teil des Programms, der sich mit dem Timer befasste, beginnt mit

```
void timer () interrupt 1 {zaehler++; }
```

Diese Programmzeile definiert den Befehlstyp des Timers und legt fest, dass bei jedem Überlauf die Variable „zaehler“ um 1 inkrementiert werden soll. Der Timer zählt also jede $\frac{1}{4}$ Millisekunde um eins hoch. Zwei weitere Programmzeilen stellten die einzelnen Modi ein, legten den Reloadwert fest und starteten den Timer.

Serielle Schnittstelle

VINCENT SCHMIDT

Die serielle Übertragung braucht man, um Informationen auszutauschen. Bei der seriellen Übertragung werden die Bits, anders als bei der parallelen Übertragung, nacheinander übertragen.

Eine Übertragung beginnt mit einem Startbit, einer logischen 0. Es folgen acht (manchmal auch neun) Datenbits, welche das eigentliche Zeichen bilden. Danach wird ein Bit zur Kontrolle der gesendeten Daten übertragen, gefolgt von einem Stoppbit. Schließlich beginnt das Ganze wieder von vorne, bis es keine Daten mehr zu übertragen gibt. Eine logische 0 wird mit einer Spannung von +3 V bis +15 V übertragen, eine logische 1 mit einer Spannung von –3 V bis –15 V.

Das Stoppbit ist immer eine logische 1. Es können ein normal langes Stoppbit, ein verlängertes Stoppbit oder zwei Stoppbits übertragen werden. Zusammen mit dem Startbit ermöglicht das Stoppbit, dass das Senden und das Empfangen synchron läuft.

Um zu überprüfen, ob es einen Übertragungsfehler gegeben hat, gibt es ein Kontroll-Bit, die sog. Parität. Es gibt folgende Paritäten: gerade Parität (even parity), ungerade Parität (odd parity), immer-1-Parität (mark parity), immer-0-Parität (space parity) und keine Parität (none parity). Es werden die gesendeten Einsen gezählt, die Summe soll gerade oder ungerade sein. Wenn also die Zahl 76 übertragen werden soll, also binär 01001100, ist die Summe der Einsen 3, also ungerade. Demnach wird als Paritätsbit eine 0 übertragen, da, wie oben schon genannt, eine ungerade Parität verwendet wird. Wenn die Zahl jedoch 77, 01001101, wäre, gäbe es vier übertragene Einsen, folglich wäre das Paritätsbit 1. Der Nachteil ist jedoch, dass die Parität nur einen Fehler erkennt, aber keinen verbessern kann. Zum Beispiel ist bei ungerader Parität die Summe der Einsen 6. Der Empfänger erkennt also, dass es einen Fehler gegeben hat. Allerdings kann der Empfänger nicht wissen, wenn er 1011110 empfangen hat, ob die Zahl 10110110 oder 10101110 heißen sollte.

Außerdem ist für die serielle Übertragung die Baudrate wichtig. Ein Baud (Bd) entspricht einem Bit pro Sekunde und hat die Dauer eines Taktes. Je länger das Übertragungskabel ist, desto kleiner ist die maximale Baudrate. Sehr oft wird eine Baudrate von 9600 (9600 Bits pro Sekunde) verwendet, wobei das Kabel bis zu ca. 150 m lang sein kann.

Ein serielles Übertragungskabel besteht aus einer GND-Leitung und aus TxD- und RxD-Leitungen. GND bedeutet „ground“, legt das Masse-Potential fest und ist oft mit dem Minuspol der Spannungsquelle verbunden. Beide Spannungsquellen bei einer seriellen Übertragung sind mit dem Minuspol an GND angeschlossen, sodass beide dasselbe Bezugspotential haben. TxD bedeutet „Transmit Data“, die Leitung geht also vom Endgerät zum Übertragungsgerät, während die Leitung RxD („Recei-

ve Data“) vom Übertragungsgerät zum Endgerät führt.

Um Daten mit einer seriellen Schnittstelle übertragen zu können, braucht man zusätzlich zur Schnittstelle ein Hardwareelement, den sogenannten MAX232, denn der Mikrocontroller verträgt solche hohen Spannungen nicht. Der MAX232 ist ein Pegelkonverter, der die Spannung auf +10 V zu verdoppelt und auf –10 V invertiert. Er wandelt ein TTL-Signal (bei unserem Mikrocontroller: 0–5 V; 0 V entspricht einer logischen 0, 5 V einer logischen 1) in ein RS232-Signal um, das der PC versteht, indem er das Signal auf höhere Pegel umsetzt und invertiert. Genauso funktioniert das auch rückwärts.

ASCII

VINCENT SCHMIDT

ASCII bedeutet „American Standard Code for Information Interchange“ und hat die Grundidee, dass jede Zahl einem Zeichen entspricht. Da es ein 7-bit-Code ist, gibt es 128 Ziffern, wobei manche Ziffern sogenannten „nicht druckbare Zeichen“ entsprechen (die ersten 32). Andere Ziffern stehen für Zeichen wie #, !, -, \$, &, %, (,), usw. Buchstaben gibt es auch, sowohl klein als auch groß, aber ohne ä, ö, ü und ß, da es ja der „American Code“ ist.

Dezimal	Hexadezimal	Zeichen
33	0x21	!
34	0x22	"
...	...	...
48	0x30	0
49	0x31	1
50	0x32	2
...	...	...
65	0x41	A
66	0x42	B
67	0x43	C
...	...	...
97	0x61	a
98	0x62	b
99	0x63	c
...	...	...

Beispiel für druckbare Zeichen

Natürlich findet man auch nichtdruckbare Zeichen wie Leerzeichen und Tab in der Tabelle.

Dezimal	Hexadezimal	Zeichen
2	0x02	Start vom Text
9	0x09	horizontaler Tab
27	0x1B	Escape
32	0x20	Leerzeichen

Beispiele für nicht druckbare Zeichen

Man verwendet ASCII beispielsweise, um Zeichen auf einem LC-Display (liquid crystal display) anzuzeigen. In C kann man die Kodierung von ASCII direkt nutzen. Zum Beispiel ist „char x=72“ das gleiche wie „char x='H'“ oder „char x=0x48“. Mit entsprechenden Befehlen kann man z. B. einem LC-Display das senden, was angezeigt werden soll. Es gibt Tabellen (wie oben schon genannt), mit denen man das ASCII-Zeichen für die entsprechenden Zahlen (sowohl dezimal, binär als auch hexadezimal) findet.

Man benötigt ASCII, weil man Texte schreiben möchte, Zeichen aber nicht binär speichern kann, sondern nur Zahlen.

Ergebnisse der Teststrecke

NATHANAEL PFANNER

Nach einer Woche harter Arbeit war unsere Teststrecke und die Programmierung so weit, dass die Zeit für das Hinunterfahren des Wagens angezeigt werden konnte. Zu unserer großen Freude funktionierte dies auf Anhieb fast reibungslos. Dennoch wollten wir noch weitergehen, sodass die gemessene Zeit auf dem Computer (in Excel) angezeigt wird. Dafür mussten wir ein weiteres Programm schreiben. Damit beschäftigte sich ein kleiner Teil unseres Kurses, der – mit Hilfe unseres Kursleiters Michael – das Programm in C# erstellte. Als nach einigen Stunden und etlichen Verbesserungen das Programm endlich fertig gestellt war, warteten alle gespannt auf die große Premiere.

Als das Programm geöffnet wurde, erschien ein Fenster mit einem Button „Starte Messung“. Sobald dieser angeklickt wurde, begann nach einem Countdown die Testfahrt. Nach der

Testfahrt klickte man auf „Messdaten Empfangen“, wodurch der Computer die Daten vom Mikrocontroller empfing. Jetzt fehlte nur noch ein Button, der gedrückt werden musste und zwar „Starte Excel“, wodurch sich ein Fenster mit Excel öffnete, das die Messdaten der Lichtschranken auf die tausendstel Sekunde genau anzeigte.

Damit man aber nicht nur Zahlen sieht, stellen wir die Messergebnisse auch in einem Diagramm dar. In diesem Diagramm sind die Durchfahrtsgeschwindigkeiten des Wagens bei den einzelnen Lichtschranken in Meter pro Sekunde angegeben. Um zu überprüfen, ob unsere Messwerte realistisch waren, berechneten wir mithilfe des Neigungswinkels die theoretische Geschwindigkeit. Das war jedoch nicht ganz so einfach wie gedacht, denn zuerst mussten wir die verschiedenen Starthöhen und die dazugehörigen Winkel bestimmen, wofür wir den Satz des Pythagoras verwendeten. Damit konnten wir jedoch noch nicht die theoretische Geschwindigkeit berechnen. Deshalb gaben uns unsere Kursleiter noch eine physikalische Einführung in die Kinematik (Bewegungslehre). Dies war unser zweiter großer Theorietag, nachdem wir den ersten ganz gut überstanden hatten.

Dieser Theorietag begann mit der Frage nach den sieben SI-Basiseinheiten. Für die Geschwindigkeitsberechnung benötigten wir davon die Zeit, die Länge und die Masse. Mit diesen Einheiten wiederholten wir einige Formeln, unter anderem für die Geschwindigkeit und die Bewegungsgesetze. Diese Formeln inklusive der Berechnung von Winkeln wurden dann so zusammengesetzt, dass eine große Formel entstand, in die man nur noch die Werte einsetzen musste. Die Formel enthielt auch, wie sich ein Körper verhält, wenn er eine schiefe Ebene hinunterfährt bzw. rutscht. Das ist eine Kombination aus der Normalkraft, die im rechtem Winkel von der Ebene wirkt, der Schwerkraft und der Hangabtriebskraft, die parallel zur schiefen Ebene besteht.

Als alles fertig war, wir die Formel vereinfacht hatten und schließlich das Ergebnis mit den passenden Zahlen ausgerechnet hatten, verglichen wir die berechnete Geschwindigkeit und

die tatsächliche Geschwindigkeit im Diagramm. Dabei fiel jedoch auf, dass die Werte nicht übereinstimmten und wir dachten zuerst, dass wir einen Fehler gemacht hatten, denn die beiden Kurven wichen voneinander ab. Nach kurzem Überlegen wurde uns klar, warum die zwei Geschwindigkeiten nicht gleich waren – wir hatten weder die Reibung noch den Luftwiderstand in unserer Rechnung berücksichtigt. Außerdem führten wir den Versuch mit Legospielzeug durch, wodurch klar war, dass nicht alles hundertprozentig stimmen konnte. Im Grunde waren sich die beiden Kurven im Diagramm jedoch sehr ähnlich, nur dass die tatsächliche Geschwindigkeit geringer als die berechnete Geschwindigkeit war. Beide Kurven hatten den gleichen Anstiegswinkel, weshalb wir sehr zufrieden mit dem Ergebnis waren.

Gegen Ende hatte unser Countdown – dank Kevin – sogar noch das Sprechen gelernt. Das heißt, er zählte von Drei bis Null herunter und ließ dann das Auto mithilfe des Elektromagneten los.

Exkursion

DANIELA KROMER

Eines Tages verkündeten Kevin und Michael uns die frohe Botschaft, dass wir nicht am Exkursionsmontag in Adelsheim sitzen, sondern nach Stuttgart zum Institut für Mikroelektronik (IMS Chips) gehen würden. So liefen wir also am Montag früh zum Bahnhof. Mental mussten wir uns noch darauf vorbereiten, wieder in die richtige Zivilisation zu gehen, doch dafür würden wir bei den knapp zwei Stunden Zugfahrt noch genügend Zeit finden. Einige hatten sich da jedoch zu früh gefreut: Michael bot Zugunterricht an – für Freiwillige – also ließen sich drei von uns wirklich bis zum Stuttgarter Bahnhof mit Wissen über Mikrocontroller und Computer berieseln – ob sie sich das so vorgestellt hatten? Das bleibt wohl offen, es wurde aber gemunkelt, dass es ganz interessant gewesen sei. In Stuttgart angekommen, hieß es nur noch: „Ihr habt jetzt zwei Stunden Freizeit – wir treffen uns um halb eins wieder hier.“

So rasten dann alle Dreiergruppen los – machten Powershopping oder Lebensmittelvorrats-

kauf, aber trafen letztenendes doch alle mehr oder weniger zufällig bei McDonalds zum Mittagessen wieder aufeinander. Mit der S-Bahn (wobei hier anzumerken ist, dass die Überlegungen zum Unterschied zwischen S- und U-Bahn in Stuttgart zu keinem wirklichen Ergebnis kamen) ging es dann zum Unigelände und von dort – sportlich wie wir sind – zu Fuß über einige Umwege zum IMS. Dort wurden wir von Herrn Futterer, der uns später auch die Einführung gab, freundlich empfangen. Hier muss ich nun doch noch die wahnsinnig bequemen Stühle im Seminarraum erwähnen – das waren wirklich die tollsten Stühle, auf denen ich je gesessen bin! Wir bekamen eine kleine Einführung in die Institutsgeschichte und aktuelle Fakten zum Betrieb.

Danach gab uns Herr Balazs eine Wiederholung der verschiedenen Logikgatter (AND, OR und NOT), und zur Übung durften wir noch auf Steckbrettern mit den Gattern verschiedene Schaltungen ausprobieren, die in einem Heftchen erklärt waren. Schließlich kam das eigentlich Interessante unseres Besuches: Die Führung am Reinraum entlang und zum „Verpacken“ der winzigen Chips in ihre Gehäuse.

Dem IMS steht ein 750 m^2 großer Reinraum zur Verfügung, also ein Raum, in dem nur eine wahnsinnig geringe Anzahl an Fremdpartikeln enthalten ist. Hier wird alles rund um die Chipfertigung an sich gemacht. Zuerst braucht man sozusagen einen Boden für den Chip. Dieser besteht aus Silizium. Dieser Träger wird im IMS auch selbst hergestellt indem man Silizium so schmilzt, dass man Siliziumrohmassestangen mit dem Durchmesser von 6 cm hat (es gibt durchaus auch hier verschiedene Größen – bis zu 24 cm). Diese Stangen werden dann zugesägt, sodass hauchdünne Siliziumplatten entstehen. Darauf werden so viele Chips wie möglich aufgebracht (man muss beachten, dass ein Chip nur ungefähr $4 \times 4\text{ mm}$ groß ist).

Ein Chip besteht aus über 100 Millionen Transistoren. Diese werden in den unteren zehn Schichten des Chips angebracht. Das funktioniert so, dass man auf die Siliziumplatte eine Metallschicht sprüht. Darauf wird dann ein Lack gegeben, der nicht löslich ist. Der Lack wird nun an den Stellen entfernt, an denen



Reinraum

später kein Metall der unteren Schicht sein soll. Dieses Lackentfernen macht man mithilfe von selbst hergestellten Masken. Diese Masken sind wie ein Stück Glas, auf dem die Stellen schwarz gemacht wurden, auf denen der Lack später erhalten bleiben soll. In einer der vielen riesigen Maschinen im Reinraum wird nun eine solche Maske über einen Wafer gelegt und dann wird dieser durch die Maske hindurch belichtet. Wafer werden die Siliziumplatten mit den angefangenen Chips darauf, in diesem Fall eine Siliziumplatte mit einer Metallschicht und einer Lackschicht darüber, genannt. So wird der Lack an den Stellen entfernt, an denen die Maske durchlässig war, an denen also später kein Metall sein soll. Nun wird der Wafer in eine Säure gelegt, die das Metall an den Stellen wegätzt, an denen es nicht durch den Lack geschützt ist.

Dieser Vorgang wird mehrmals übereinander wiederholt, sodass in ca. 10 Schichten, die insgesamt weniger als einen Mikrometer dick sind

einige 10 Millionen Transistoren entstehen. In den nächsten ca. 6 Schichten werden diese nun mithilfe von Halbleiterbahnen so verbunden, dass wir in unserem Chip Logikgatter haben. Nach dieser reichlich komplizierten Chipfertigung folgt dann der nicht weniger komplizierte Schritt, den Chip in ein Gehäuse zu setzen und dann die Anschlüsse vom Chip nach außen an das Gehäuse zu führen, sodass wir dort möglichst viele winzig kleine Drahtfüßchen haben. Das passiert allerdings nicht mehr im Reinraum. Auf dem Weg zurück vom Reinraumgebäude zum Seminarraum waren einigen die Fragezeichen förmlich ins Gesicht geschrieben. Die meisten Fragen sollten sich bei interessanten Diskussionen und Nachbesprechungen in der Akademie jedoch schnell klären. Nach einem Gruppenbild vor dem IMS ging es dann wieder zurück zum Bahnhof, von wo aus wir größtenteils vor uns hin dösend wieder zurück nach Adelsheim fuhren. Nach etwas Zugverspätung mussten wir beim Hochlaufen zum Eckenberg-Gymnasium auch noch bangen, dass noch etwas zu Essen für uns übrig ist, doch es musste abends glücklicherweise keiner hungrig ins Bett gehen. Stattdessen fielen wir in dieser Nacht mit viel neuem Wissen und der tollen Erfahrung einer Reinraumbesichtigung zufrieden und todmüde in unsere Betten.

Danksagung

TEILNEHMER DES KURSES

Zum Schluss wollen wir, die Kursteilnehmer, uns ganz herzlich bedanken. Zunächst einmal bei den Menschen, die es für uns möglich gemacht haben, an der Akademie teilnehmen zu können, sowie bei der Akademieleitung.

Doch vor allem möchten wir unseren Kursleitern Michael und Kevin und unserer Mentorin Wenke danken! Kevin, Michael, Wenke, auch wenn mancher von uns zu Beginn des Kurses nicht glauben konnte, sich irgendwann für Digitaltechnik begeistern zu können, ihr habt uns alle vom Gegenteil überzeugt! Dank euch hatten wir zwei informative Wochen, deren Inhalte wir so schnell nicht wieder vergessen werden. Ihr wart immer verständnisvoll, hilfsbereit, motiviert und freundlich, die perfekten Kursleiter und die perfekte Mentorin!

DANKE!



Unser Kurs vor dem Institut für Mikroelektronik Stuttgart (IMS Chips)