Von der Aussagenlogik zum Computer

Markus Koch
Gymnasium in der Glemsaue Ditzingen
Januar 2012

Inhaltsverzeichnis

Einleitung....3
Der Computer....3
Grundlagen....4
Wahrheitstabellen....4
Aussagenlogik....4
Aussagen....4
Beispiele für Aussagen....5
Und (And)....5
Oder (Or)....5
Nicht (Not)....6
X-Oder (Exklusives Oder, XOr)....6
Kombinationen....6
Technische Darstellungsweise....7
Verbindungen....7
Elemente....7
Von der Aussagenlogik zum Computer....8
Was muss ein Computer können?....8
Wie geht die CPU dabei vor?....8
Die verschiedenen Operationen....8
Addierer....8
Erweitern auf mehrere Bits (Zahlen größer 1)....10
Schlussfolgerung....12
Abschließende Worte....12
Weiterführendes Material....13
Eigenproduktion....13
Extern....13
Quellen....13
Selbstständigkeitserklärung....14

Einleitung

Der Computer

Der Computer besteht aus vielen verschiedenen Bauteilen. So benötigt ein Computer Arbeitsspeicher, um Daten während dem Betrieb abzulegen, Festplatten um diese danach auch zu speichern und die Grafikkarte, um diese anzuzeigen. Das Herzstück des Computers ist jedoch der Prozessor (CPU[1]). Die CPU führt die Programme aus, verarbeitet die Eingaben und steuert somit alle Abläufe im PC. Dieser Teil des Computers ist also dafür verantwortlich, dass wir unsere Dokumente schreiben, Musik hören oder komplexe Spiele spielen können.
Die CPU bearbeitet diese Aufgaben jedoch ausschließlich mit sehr einfachen Operationen. Genau genommen ordnet sie nur Einsen und Nullen nach bestimmten Mustern neu an und schiebt sie an andere Stellen im Speicher.

Im Folgenden werde ich die Grundlagen erläutern und danach an einem Beispiel zeigen, wie man eine Funktion (die Addition zweier Zahlen) herleitet.

1 CPU: Central Processing Unit

Grundlagen

Wahrheitstabellen

Mit Hilfe von Wahrheitstabellen wird ein Ergebnis in Abhängigkeit von einer oder mehreren Aussagen[2] dargestellt. Folgendes Beispiel stellt eine Und-Verknüpfung dar. Was dies genau bedeutet wird später noch erklärt.

A (Aussage)	B (Aussage)	Q (Ergebnis)
Falsch	Falsch	Falsch
Falsch	Wahr	Falsch
Wahr	Falsch	Falsch
Wahr	Wahr	Wahr

An einer solchen Wahrheitstabelle kann man sehr schnell ein bestimmtes Ergebnis ablesen beziehungsweise erkennen. Hier sieht man sofort, dass das Ergebnis nur wahr ist, wenn beide Aussagen wahr sind. Ansonsten ist das Ergebnis immer Falsch.

Aussagenlogik

Die Aussagenlogik beschreibt die Verknüpfung von mehreren Aussagen zu einem Ergebnis.

Aussagen

Die Aussagen können in der klassischen Aussagenlogik jedoch nur die Wahrheitswerte „Wahr“ oder „Falsch“ annehmen. Anstatt Wahr und Falsch werden oft auch die englischen Namen true und false oder 1 und 0 verwendet.

Für die Aussagen werden der Übersichtlichkeit halber im Folgenden die Buchstaben A,B,C,... verwendet. Das Ergebnis wird mit dem Buchstaben Q dargestellt.

In der Aussagenlogik gibt es 4 grundlegende Operationen (logische Funktionen) um diese Aussagen zu verknüpfen:

Und, Oder, Nicht und X-Oder.

Es ist auch möglich, diese Operationen miteinander zu verknüpfen.

2 Aussage: Ein bestimmter Wert. Hier: Wahr oder Falsch.

Beispiele für Aussagen

Gültige Aussagen:

A: Der Eiffelturm ist 324,82 Meter hoch.

B: Paris liegt in Australien.

Ungültige Aussagen:

A: Was gibt es zu essen?

B: Hilfe!

Und (And)

A (Aussage)	B (Aussage)	Q (Ergebnis)
Falsch	Falsch	Falsch
Falsch	Wahr	Falsch
Wahr	Falsch	Falsch
Wahr	Wahr	Wahr

Die Und-Verknüpfung verbindet 2 oder mehrere Aussagen zu einem Ergebnis.
Hierbei ist das Ergebnis wahr, wenn *A* **UND** *B* wahr sind. Sobald eine der Aussagen den Wert falsch annimmt, ist auch das Ergebnis falsch. Ausgeweitet auf 3 Aussagen wird das Ergebnis nur wahr, wenn A, B **UND** C wahr sind.

Übertragen auf das Beispiel für gültige Aussagen bedeutet das, dass ich nur nach Paris gehe (Q), wenn der Eiffelturm 324,82 Meter hoch ist (A) **und** Paris in Australien liegt (B). In der Tabelle wäre dieses Beispiel in Zeile 3 und das Ergebnis somit falsch. Ich würde also nicht nach Paris gehen.

Oder (Or)

A (Aussage)	B (Aussage)	Q (Ergebnis)
Falsch	Falsch	Falsch
Falsch	Wahr	Wahr
Wahr	Falsch	Wahr
Wahr	Wahr	Wahr

Die Oder-Verknüpfung verbindet ebenfalls 2 oder mehrere Aussagen zu einem Ergebnis.

Im Vergleich zur Und-Verknüpfung wird hier das Ergebnis bereits wahr, sobald **eine der Aussagen zutrifft**. Es können auch beide Aussagen wahr sein.

Im Beispiel würde es bedeuten, dass ich nach Paris gehe, wenn **entweder** der Eiffelturm 324,82 Meter hoch ist **oder** Paris in Australien liegt. Ich würde auch gehen, wenn beides zutrifft. In der Tabelle ist es erneut die dritte Zeile, jedoch ist diesmal der Wert „Wahr“. Das bedeutet ich würde nach Paris gehen.

Nicht (Not)

A (Aussage)	Q (Ergebnis)
Falsch	Wahr
Wahr	Falsch

Die Nicht-Verknüpfung nimmt als einzige Operation nur eine Aussage.
Diese Aussage wird bei der Nicht-Verknüpfung immer invertiert. Aus Wahr wird also Falsch und aus Falsch immer Wahr.

Auf das Beispiel übertragen bedeutet dies, dass wir nun eine Aussage verwerfen müssen, da wir der Nicht-Verknüpfung nur eine Aussage übergeben können. Der Eiffelturm ist 324,82 Meter hoch. A ist also wahr. Nach der Nicht-Verknüpfung müssen wir diesen Wert nun invertieren. Also gehe ich nicht nach Paris.

X-Oder (Exklusives Oder, XOr)

A (Aussage)	B (Aussage)	Q (Ergebnis)
Falsch	Falsch	Falsch
Falsch	Wahr	Wahr
Wahr	Falsch	Wahr
Wahr	Wahr	Falsch

Die X-Oder-Verknüpfung verbindet ebenfalls 2 oder mehrere Aussagen zu einem Ergebnis.
Im Vergleich zur normalen Oder-Verknüpfung muss bei X-Oder eine ungerade Zahl der Eingänge wahr sein. Bei 2 Eingängen bedeutet dies, dass entweder A oder B wahr sein muss, jedoch nicht beide.

Kombinationen

Alle oben genannten Funktionen können mit einander Verknüpft werden. Die wichtigsten Verknüpfungen für grundlegende Funktionen sind die Verknüpfungen von *Und* beziehungsweise *Oder* mit *Nicht*.
Hierfür muss man die Werte der Spalte *Q* in der jeweiligen Wahrheitstabelle invertieren (→Nicht-Funktion auf Q anwenden):

Nicht-Oder (Nor)		
A (Aussage)	**B (Aussage)**	**Q (Ergebnis)**
Falsch	Falsch	Wahr
Falsch	Wahr	Falsch
Wahr	Falsch	Falsch
Wahr	Wahr	Falsch

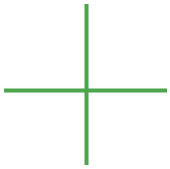
Nicht-Und (Nand)		
A (Aussage)	**B (Aussage)**	**Q (Ergebnis)**
Falsch	Falsch	Wahr
Falsch	Wahr	Wahr
Wahr	Falsch	Wahr
Wahr	Wahr	Falsch

Technische Darstellungsweise

Da ein Computer aus elektronischen Schaltkreisen besteht, werden im Folgenden die Schaltungen mit Hilfe von Schaltplänen dargestellt.
Die Verknüpfungen heißen als Bauteile „Logikgatter“.

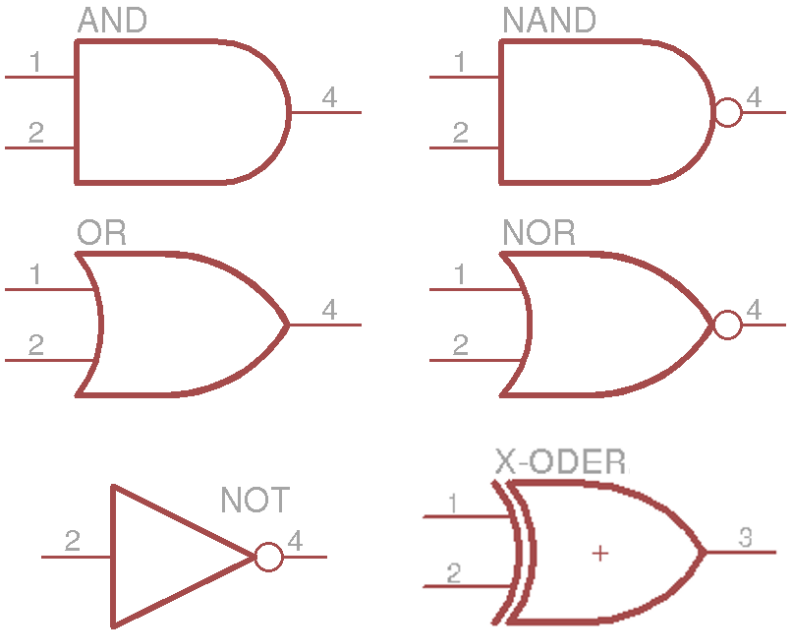
Verbindungen

Die elektrischen Verbindungen zwischen den Bauteilen sind als Linien dargestellt. Kommt diese Verbindung vom Anschluss eines Bauteils (in diesem Fall sind dies ausschließlich die Gatter) ist es mit diesem Verbunden. Überkreuzen sich zwei Leitungen, so sind diese nur verbunden, wenn ein Punkt am Schnittpunkt ist.

Die Leitungen sind nicht verbunden	Die Leitungen sind verbunden
	

Elemente

Für die Darstellung der Logikgatter werden die ANSI- bzw. MIL-Symbole verwendet.
Die Eingänge (Aussagen) befinden sich an der linken Seite, der Ausgang (Ergebnis) befindet sich an der rechten, abgerundeten beziehungsweise spitz zulaufenden Seite:



Von der Aussagenlogik zum Computer

Was muss ein Computer können?

Wie der Name „Computer“ (engl. to compute = rechnen) bereits vermuten lässt, muss ein Computer eigentlich nur rechnen können. Dazu zählen einerseits die einfachen Rechenoperationen wie Addieren, Subtrahieren, Dividieren und Multiplizieren aber auch Funktionen zum Dekodieren diverser Kompressions- oder Verschlüsselungsmechanismen. Des Weiteren muss der Computer Zahlen von verschiedenen Adressen[3] kopieren, modifizieren und auswerten können. Der Computer verarbeitet die Zahlen als Dualzahlen, also als eine Folge von Einsen und Nullen.
All diese teils sehr komplizierten Vorgänge können mit den oben genannten logischen Funktionen realisiert werden.

Wie geht die CPU dabei vor?

Im Prozessor gibt es ein Register[4], in dem die Speicheradresse des aktuellen Befehls steht. Der Inhalt dieser Zelle wird von der CPU eingelesen und entsprechend werden bestimmte Bereiche der CPU aktiviert. Nach der Ausführung wird dieser Zähler um 1 erhöht.

Die verschiedenen Operationen

Wie oben bereits erwähnt, gibt es sehr viele und umfangreiche Funktionen innerhalb der CPU. Im Folgenden werde ich anhand des Addierers beschreiben, wie man beim Entwerfen eines solchen Teils der CPU vorgehen kann.

Addierer

Um ein Bauteil wie einen Addierer zu entwerfen ist es am besten mit einem Bit[5] pro Zahl anzufangen und das Konzept dann auf mehrere Bits zu erweitern.

Da wir jedoch durch Addition der Maximalwerte bei 1 Bit pro Eingang (*1+1*) „*2*“ als Ergebnis erhalten, brauchen wir bereits zwei Ergebnisse: 1 Bit für das Ergebnis und ein weiteres Bit für den Übertrag. Auch bei diesem Schritt ist es hilfreich, eine Wahrheitstabelle zu verwenden.

A und *B* sind hierbei die beiden zu addierenden Zahlen. *S* (Sum) ist die Summe und *C* (Carry) ist

3 Adresse: In diesem Fall bestimmte Positionen im Speicher.
4 Register: Speicher innerhalb der CPU.
5 Bit: **Bi**nary digi**t** (Kann 1 oder 0 sein) Durch Kombinationen mehrerer Bits können Zahlen >1 dargestellt werden.

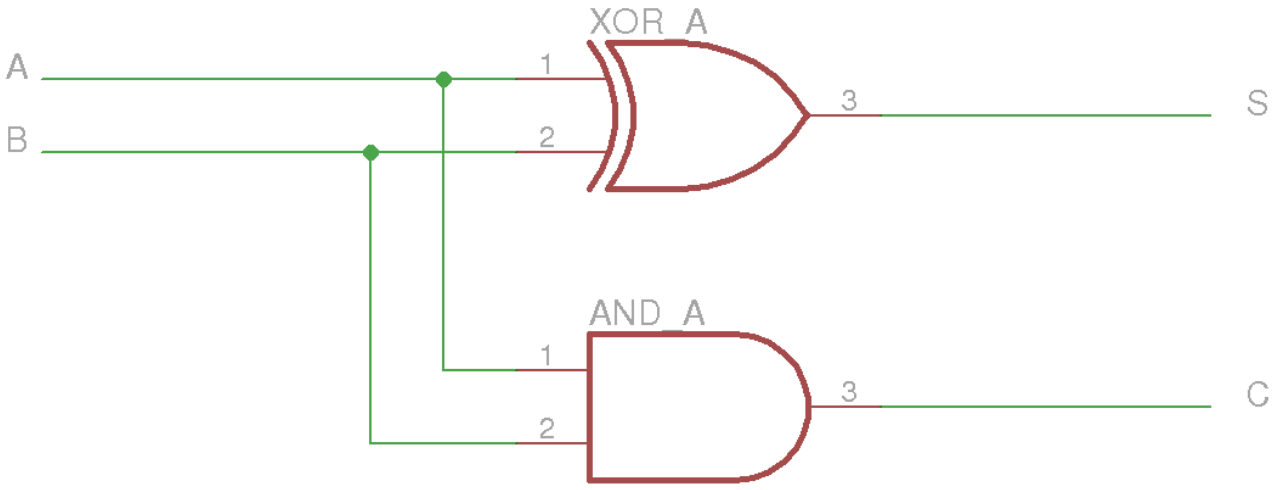
der Übertrag.

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Rechnet man eine solche Tabelle aus, so erkennt man, dass...

- *S* 1 ist, wenn *A* oder *B* 1 ist. Wenn jedoch beide gemeinsam 1 sind, so ist das Ergebnis wieder 0.
  → X-Oder-Verknüpfung
- *C* nur 1 ist, wenn A und B 1 sind.
  → Und-Verknüpfung

Das Schaltbild wäre demnach:



Diese Schaltung bezeichnet man als Halbaddierer. Jedoch kann man diesen nicht einfach auf weitere Bits erweitern, da sie kein Carry-Bit (Der Übertrag) als Eingabe nimmt. Um dieses Problem zu lösen kann man erneut eine Wahrheitstabelle erstellen. Für einen Addierer mit Carry-Bit-Eingang, einem so genannten Volladdierer, sieht diese wie folgt aus:

A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Beim Rechnen der Tabelle rechnet man diesmal $A+B+C_{in}$ und schreibt das Ergebnis in Binärform in S und C_{out}.

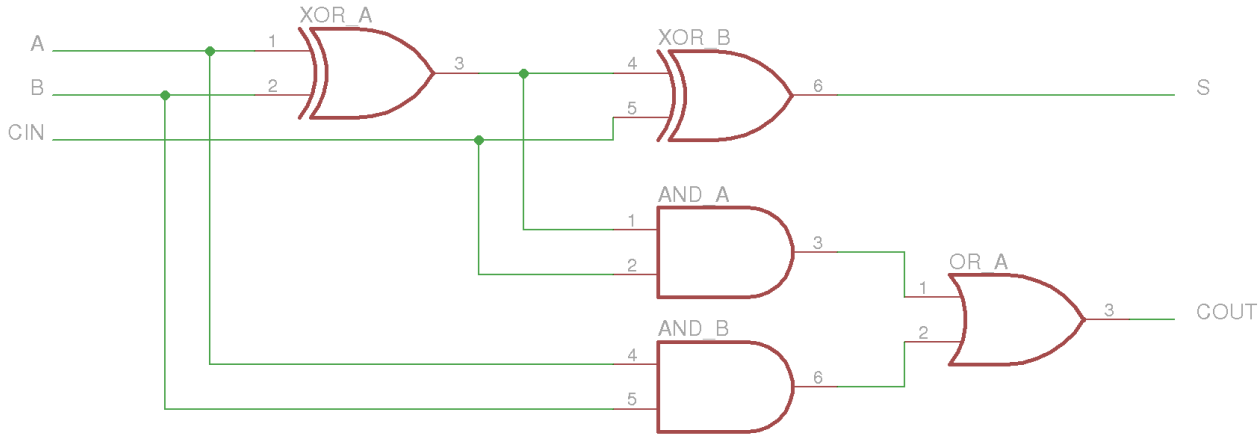
In dieser Tabelle erkennt man, dass C_{out} 1 ist, wenn entweder...

- *A* und *B* 1 sind
- oder C_{in} gleichzeitig mit *A* oder *B* 1 ist
- oder Alle Eingänge gleichzeitig 1 sind.

S ist 1, wenn die Anzahl der aktiven Eingänge (1) ungerade ist.

Der Summe-Ausgang ist relativ einfach aufgebaut. Man benötigt hierfür ausschließlich 2 X-Oder-Gatter hintereinander geschaltet. Das zweite Gatter (*XOR_B*) erhält hierbei als Eingänge den

Ausgang des ersten Gatters und den C_{in}-Eingang. Für das C_{out}-Bit können wir ebenfalls unsere Erkenntnisse aus der Tabelle einfach übertragen. Als erstes werden A und B mittels einer Und-Verknüpfung (*AND_B*) verknüpft. Für den zweiten Punkt nimmt man einfach den C_{in}-Eingang und den Ausgang des ersten X-Oder-Gatters (*XOR_A*) aus der Bestimmung des Summe-Ausgangs. Diese Ausgänge verknüpft man nun noch mit einem Oder-Gatter (*OR_A*), da es reicht, wenn eine der beiden Aussagen zutrifft. Der dritte Punkt aus unserer Tabelle ist hiermit direkt erledigt, da Eingänge A und B gemeinsam 1 sein müssen, wenn alle Eingänge 1 sind. Aus diesen Informationen heraus ergibt sich folgendes Schaltbild:



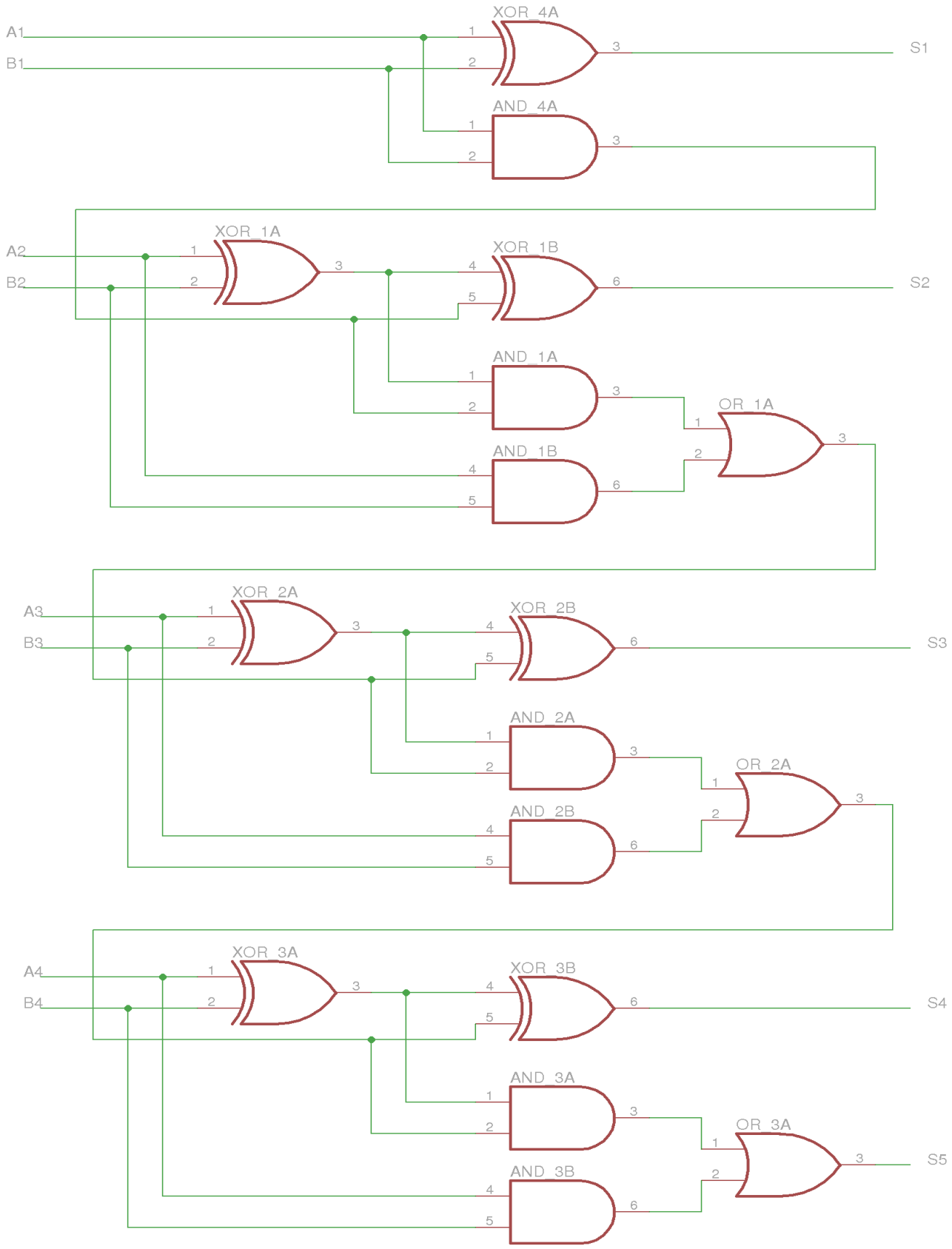
Erweitern auf mehrere Bits (Zahlen größer 1)

Um diesen Addierer nun auf mehrere Bits zu erweitern, muss man ihn nur mehrmals hintereinander schalten. Das bedeutet man verbindet den C_{in} des Zweiten mit dem C_{out} des Ersten. A und B des ersten Addierers sind das jeweilige erste Bit der Zahlen. A und B des zweiten Addierers sind somit das zweite Bit und so weiter. Der erste Addierer ist hierbei ein Halbaddierer, da kein C_{in}-Bit benötigt wird. Beim letzten Addierer wird das Carry-Bit zum fünften Bit der Summe.

Auf der folgenden Seite ist der Schaltplan eines 4-Bit-Addierers zu sehen. Möchte man zwei Zahlen hiermit addieren, so muss man diese in das Binärsystem umrechnen. Das niedrigste Bit (ganz rechts) ist dabei A1 beziehungsweise B1. Nun kann man entweder die Ergebnisse der einzelnen Gatter manuell berechnen oder man erstellt eine solche Schaltung in einer Simulationssoftware. (*Siehe „Weiterführendes Material → Extern“*)

Da dieser Addierer 4-Bit pro Zahl verarbeiten kann, können wir Zahlen bis $(2^4-1)=15$ miteinander addieren. Das höchste Ergebnis ist somit 30. Diese Zahl ist mit 5 Bit darstellbar. **Beispiel:**

Dezimal	9	+	11	=	20
Binär	1001		1011		10100



[4-Bit-Addierer]

Schlussfolgerung

Wie man bereits an diesem vermeintlich einfachen Beispiel sehen kann, ist ein Computer sehr komplex aufgebaut. Heutige Prozessoren enthalten oft Milliarden von solchen Logikgattern und entsprechend groß kann man sich den Aufwand vorstellen, diese zu Entwerfen.

Abschließende Worte

Hat man alle wichtigen Dinge im Prozessor implementiert und auch die nötige Hardware aufgebaut, so ist der Computer trotzdem noch nicht so benutzbar, wie wir es kennen. An diesem Punkt kann man den Computer zwar einschalten, jedoch würde der Bildschirm schwarz bleiben. Hier kommt die Software beziehungsweise das Betriebssystem ins Spiel.
Der Computer kann von sich aus nichts machen. Er führt ausschließlich „blind“ die Programmbefehle in Form von Maschinencode[6] aus. Um diesen zu erstellen gibt es viele Möglichkeiten. Eine Möglichkeit ist es, den Maschinencode von Hand, Zahl für Zahl, einzugeben. Dies ist jedoch aus mehreren Gründen unpraktisch und sehr aufwendig. Deswegen wurden die Programmiersprachen erfunden. Die Sprache „C“ ist mit ihrem Nachfolger „C++“ eine der am weit Verbreitetsten. Eine ebenso wichtige Sprache ist „Assembler“, da hiermit eine sehr hardwarenahe Programmierung ermöglicht wird.

6 Maschinencode: Folge von Zahlen, die der Computer interpretiert und ausführt.

Weiterführendes Material

Eigenproduktion

Nähere Informationen zu den Logikgattern (Videos, deutsch):

http://www.youtube.com/watch?v=NTdEcM_hFeg

http://www.youtube.com/watch?v=kA-ENhSPa-o

http://www.youtube.com/watch?v=1INg4gJ-jd4

Extern

Simulationsprogramm für el. Schaltungen:

Logisim (Carl Burch, open source): http://ozark.hendrix.edu/~burch/logisim/

Yenka (Crocodile Clips Ltd., kostenlose Heimnutzung): http://www.yenka.com/

Zahlen mit Google umrechnen:

Dezimal nach Binär: „[Zahl] in Binary“

Binär nach Dezimal: „0b[Zahl] in Decimal“

Selbstgebaute CPU mit Erklärungen:

http://www.homebrewcpu.com/

Funktionsprinzip einer CPU (AMD):

http://buzbee.net/bitslice/

Quellen

http://de.wikipedia.org/wiki/Aussagenlogik, 02.01.2012

http://en.wikipedia.org/wiki/Adder_(electronics), 03.01.2012

Schaltbilder: Erstellt mit „eagle“

Selbstständigkeitserklärung

Ich habe diese GFS ohne fremde Hilfe und nur mit den im Verzeichnis angegebenen Hilfsmitteln angefertigt. Ich habe alle Stellen, die im Wortlaut oder dem Sinn nach anderen Veröffentlichungen entnommen sind, durch Angabe der Quellen als Entlehnungen kenntlich gemacht.

Markus Koch