Das große All-in-All CPLD/FPGA Tutorial

Mit diesem Tutorial sollen die ersten Schritte in die Welt der programmierbaren Logik vereinfacht werden. Es werden sowohl die Grundlagen der Logik, die benötigte Hardware als auch Software beschrieben.

Für Fragen und Anmerkungen stehe ich gerne zu Verfügung.

Thomas Unmuth, den 18.September.2005

thomas@unmuth.de

Inhaltsverzeichnis

1.Die Geschichte der programmierbaren Logikbausteine

Die Ursprünge der digitalen Elektronik liegen bereits in der Mitte des 19 Jahrhunderts. Zu dieser Zeit wurde das Morsen erstmals eingesetzt. Das Morsen basiert auf digitaler Elektronik, da der Morser nur 1 oder 0 senden kann.

Digitale Elektronik im heutigen Sinne, bei der auch logische Verknüpfungen möglich sind, wird seit Mitte des 20. Jahrhunderts verwendet. Zu dieser Zeit wurden große komplexe digitale Schaltungen komplett mit Logik ICs aufgebaut. Die dabei entstandenen Schaltungen waren viele Leiterplatten groß, bei nur bescheidenem Funktionsumfang. So hat zum Beispiel Motorola eine seiner ersten Mikrocontroller in der Entwicklung komplett mit einzelnen Logik-ICs aufgebaut.

Um dieser Komplexität Herr zu werden, wurde 1978 der erste frei konfigurierbare Logikbaustein auf den Markt gebracht. Mit diesem war es erstmals möglich digitale Elektronik auf einem Baustein zu erstellen.

Diese Systeme wurden weiterentwickelt, und heute stellen CPLD(Complex Programmable Logic Device) und FPGA(Field Programmable Gate Array) den Stand der Technik dar. Der Unterschied von CPLDs und FPGAs ist im Wesentlichen der, dass FPGAs bei jedem Neustart konfiguriert werden müssen, da der interne Aufbau auf SRAM basiert. Die Logikinformation von CPLDs wird auf EEPROM gespeichert, deshalb bleibt die Information auch ohne Versorgungsspannung erhalten.

Um Logikbausteine programmieren zu können, wurde VHDL(Very High Speed Integrated Circuit Hardware Description Language), eine speziell dafür zugeschnittene Software entwickelt. Mit dieser Software ist es möglich Logik auf einer höheren Abstrakten Ebene zu beschreiben.

Der große Unterschied zum Programmieren von Mikrocontrollern ist, dass mit VHDL wirklich Hardware designt wird. Alle Prozesse laufen parallel ab. Dieser Unterschied macht vor allem eingefleischten Programmierern große Probleme.

Im Folgenden wird anhand eines einfachen Beispieles die Welt der digitalen Logikbausteine erklärt. Dabei wird sowohl die Hardware beschrieben, die benötigt wird um einen CPLD zu programmieren als auch die Software vom programmieren, über das Simulieren bis zum Programmieren auf den Baustein.

Dabei ist wichtig zu beachten, dass dies ein Einsteiger-Tutorial ist. Die verwendeten Tools sind extrem komplex. Deshalb werden nur die wichtigsten Funktionen erklärt um einfache Projekte zu realisieren.

2.Digitale Schaltungen und Logik

Digitale Schaltungen sind Schaltungen, bei denen jede Art von Information nur über „0“ und „1“ übertragen werden. Bei der positiven Logik, welche in der Regel verwendet wird, entspricht eine „0“ dem Massezustand, eine „1“ entspricht der Versorgungsspannung. Versorgungsspannungen sind bei herkömmlichen Systemen 5V oder 3,3V, bei moderneren Schaltungen sind auch geringere Logik-

spannungen möglich.
Um mit Nullen und Einsen ganze Systeme darstellen zu können, sind einige theoretische Grundlagen notwendig.

2.1.Zahlendarstellungen

Will man Zahlen mit Nullen und Einsen darstellen, wird die binäre Zahlendarstellung benötigt.

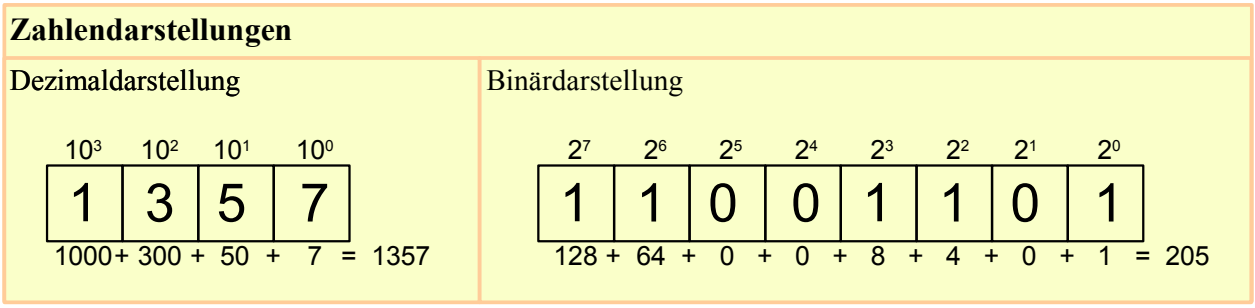


Abb. 1 - Dezimale und Binäre Zahlendarstellung

In unserem Alltag verwenden wir das dezimale Zahlensystem. In diesem Zahlensystem entspricht die letzte Stelle dem vielfachen von 1, die nächste Stelle entspricht dem vielfachen von 10 usw. Dieser Zusammenhang ist in Abb.1 zu sehen. Die Basis dieses Zahlensystems ist dabei 10.
In der binären Zahlendarstellung können die Zahlen nur durch „0“ und „1“ dargestellt werden. Mathematisch betrachtet wird eine binäre Zahl

$$Z=\sum_{i=0}^{i=n} B_i \cdot 2^i, \quad (n \in \mathbb{N}; B \in \{0,1\})$$

dargestellt. B ist dabei die Basis, und i steht für die i-te Stelle von rechts. Bildlich gesprochen steht die rechte Stelle für 1, die 2-te Stelle von rechts für 2, dann 4, 8, 16, 32, usw. Wie in Abb.1 zu sehen lässt sich aus dieser Summe jede positive ganze Zahl darstellen.
Neben den ganzen positiven Zahlen lassen sich auch sowohl negative als auch gebrochen rationale Zahlen in mehreren Formen darstellen. Da diese Darstellungen für dieses Tutorial nicht notwendig sind, werden sie hier nicht näher ausgeführt.
Eine weitere wichtige Darstellung ist die hexadezimale Zahlendarstellung. Bei dieser wird im Vergleich zu der dezimalen Darstellung nicht mit der Basis 10, sondern mit der Basis 16 gerechnet. Das heißt, dass mit der letzten Ziffer ein vielfaches von 1, mit der nächsten ein vielfaches von 16, dann ein vielfaches von 256 dargestellt wird, u.s.w. . Um Zahlen von 0 bis 15 darstellen zu können, reichen die 10 Ziffern unseres Zahlensystems nicht aus. Deshalb wird dieses auf folgende Werte ergänzt: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. Diese Zahlendarstellung wird häufig dazu genutzt, binäre Zahlen in Blöcken zu je 4 Bit zusammenzufassen.

Weiterführende Links: www.wikipedia.de/dualsystem

2.2.Logikbausteine

Eine Haupteigenschaft von programmierbaren Logikbausteinen sind Logikelemente. Diese verknüpfen einen oder mehrere Eingänge zu einem Ausgang. Dabei sind mehrere Grundverknüpfungen möglich.

Gebräuchliche logische Operationen

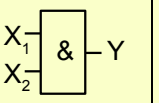
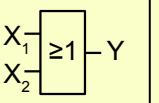
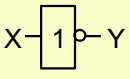
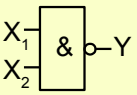
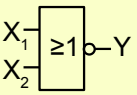
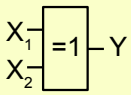
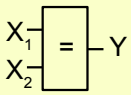
	AND	OR	NOT	NAND	NOR	EXOR	Äqui-valenz
Schaltzeichen							
Wahrheitstabelle	X_2 X_1 Y 0 0 0 0 1 0 1 0 0 1 1 1	X_2 X_1 Y 0 0 0 0 1 1 1 0 1 1 1 1	X Y 0 1 1 0	X_2 X_1 Y 0 0 1 0 1 1 1 0 1 1 1 0	X_2 X_1 Y 0 0 1 0 1 0 1 0 0 1 1 0	X_2 X_1 Y 0 0 0 0 1 1 1 0 1 1 1 0	X_2 X_1 Y 0 0 1 0 1 0 1 0 0 1 1 1
Mathem. Notation	$X_1 \cdot X_2$	$X_1 + X_2$	$\overline{X}$	$\overline{X_1 \cdot X_2}$	$\overline{X_1 + X_2}$	$X_1 \not\equiv X_2$	$X_1 \equiv X_2$

Abb. 2 - Gebräuchliche logische Operationen

In Abb.2 sind alle logischen Grundoperationen, deren Schaltzeichen, Wahrheitstabellen und mathematische Notationen dargestellt.

Diese logischen Grundoperationen lassen sich anhand eines Beispieles besser verstehen. Bei einer Tastatur werden unter anderem die logische UND und die logische ODER Funktionen verwendet. Das große A lässt sich mit einer Tastatur schreiben, wenn man Die Shift-Taste und die A-Taste gleichzeitig drückt. Wenn man sich die Wahrheitstabelle der UND Funktion anschaut, wird klar, dass nur für den Fall dass beide Eingänge aktiv sind, der Ausgang auch aktiv wird.

Bei der ODER Funktion ist der Ausgang immer dann aktiv, wenn ein oder beide Eingänge aktiv sind. Bei der Tastatur sind die beiden Shift-Tasten mit dieser Funktion verknüpft. Sobald eine der beiden, oder beide Shift-Tasten gedrückt werden, ist die Funktion aktiv. Mit diesen beiden, und den fünf anderen Logikfunktionen lassen sich alle logischen Kombinationen darstellen.

2.3.Kombinatorische Netzwerke

Mehrere der in Kapitel 2.2 beschriebenen Logikelemente bilden zusammen kombinatorische Netzwerke. Diese besitzen eine gewisse Anzahl an Ein- und Ausgängen, die auf jede beliebige Art miteinander verknüpft sein können.

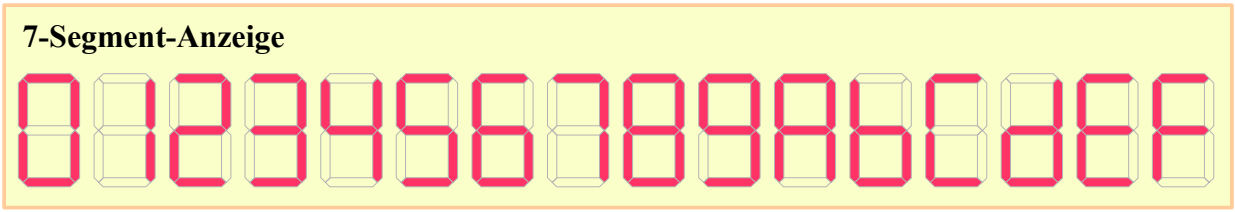


Abb. 3 - 7-Segment-Anzeige

Der Aufbau eines solchen kombinatorischen Netzwerkes soll an einem Beispiel verdeutlicht werden. Ziel ist es eine 7-Segment (vgl. Abb.3) Anzeige anhand einer 4-Bit binär-kodierten Zahl darzustellen. Diese Umsetzung kann rein kombinatorisch erfolgen. Da die komplette Umsetzung zu Aufwändig wäre, beschränken wir uns hier auf den obersten Anzeigebalken. Dieser Balken ist aktiv für die Fälle:

$$LED_{oben} = 0 + 2 + 3 + 5 + 6 + 7 + 8 + 9 + A + C + E + F$$

Einfacher kann man die Fälle betrachten, für die der oberste Balken nicht leuchtet, und das Ergebnis dann verneinen. Auch auf diese Weise kann der Balken dargestellt werden.

$$LED_{oben} = \overline{1 + 4 + B + D}$$

Werden die Zahlen Binär Codiert, folgt folgende Gleichung:

$$LED_{oben} = \overline{0001 + 0100 + 1011 + 1101}$$

Wie solche Gleichungen weiter vereinfacht werden, wird in diesem Tutorial nicht behandelt, da es sehr viele verschiedene Verfahren gibt, die zu komplex sind um sie kurz zu beschreiben. Außerdem übernimmt diese Vereinfachung später die Software mit der die Logikbausteine programmiert werden.
Die zuletzt beschriebene Gleichung kann dann mit Logikbausteinen realisiert werden. Dieses kombinatorische Netzwerk ist in Abb. 4 dargestellt

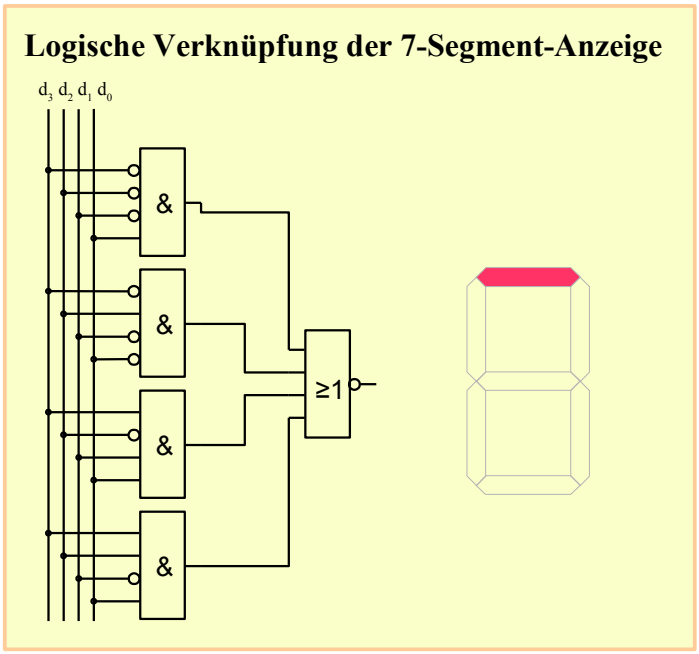


Abb. 4 - Diese logische Verknüpfung erzeugt das oberste Element der 7-Segment-Anzeige

2.4.Flip-Flops

Um kompliziertere Abläufe in einem Logikbaustein zu realisieren, genügt es nicht Eingänge mit Ausgängen logisch zu verknüpfen. Es müssen auch Zustände und Zustandsübergänge realisiert werden. Dazu ist ein Element nötig, das einzelne Zustände speichern kann. Diese Elemente heißen Flip-Flops.

Die verschiedenen Ausführungen der Flip-Flops sind in Abb. 5 dargestellt. Die einfachste Ausführung ist die des D-Flip-Flops. Um Zustandsübergänge und damit Programmabläufe zu realisieren, ist ein zentraler Takt nötig. Das D-Flip-Flop übernimmt bei jeder steigenden Taktflanke des zentralen Taktes den Wert, der am Eingang D anliegt. Ändert sich der Wert am Eingang während eines Taktes, hat dies keine Auswirkung auf den Ausgang. Der Wert, der an der letzen steigenden Takflanke des Taktes am Eingang anlag ist somit gespeichert.

Ein einfaches, praktisches Beispiel um zu verstehen wie Flip-Flops eingesetzt werden, ist das Entprellen eines Tasters. Wird ein normaler Taster gedrückt, ändert sich der Wert nicht sofort von Aus auf Ein, bzw. von 0 auf 1, sondern er prellt. Das bedeutet, das für den Zeitraum von wenigen Millisekunden der Wert zwischen 0 und 1 hin- und herspringt. Um dieses Problem bei Mikrocontrollern zu umgehen, wird meistens nach dem ersten Sprung von 0 auf 1 der Eingang erst nach 10ms wieder abgefragt. Nach dieser Zeit kann sichergestellt werden, dass sich der Wert des Tasters stabilisiert hat, er somit entprellt ist.

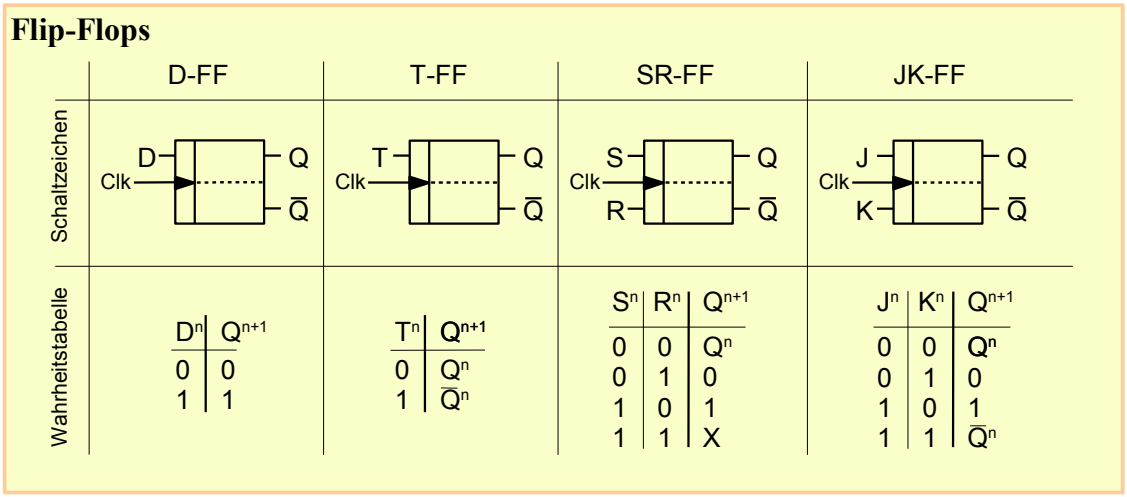


Abb. 5 - Ausführungen von Flip-Flops

Bei digitaler Logik wird eine andere Methode verwendet. Dazu wird ein Set-Reset-FF (SR-FF) verwendet. Dieses FF speichert seinen aktuellen Wert für den Fall, dass beide Eingänge 0 sind. Wird der Eingang S auf 1 gesetzt, wird der Ausgang auch auf 1 gesetzt. Wird der Eingang R aktiv, wird das FF wieder auf 0 zurückgesetzt. Diese Eigenschaft wird zum entprellen eines Tasters verwendet. Dazu wird ein Taster benötigt, der nicht nur öffnet und schließt, sondern einer der zwischen zwei Ausgängen hin- und herschaltet vgl. Abb. 6.

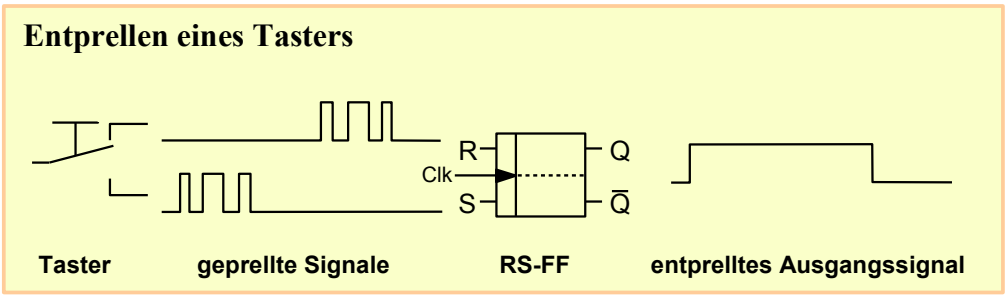


Abb. 6 - Entprellen eines Tasters mit einem RS-FF

Wird dieser Taster betätigt, kommt ein geprelltes Signal auf den Set-Eingang des Flip-Flops. Da dieser Eingang den Ausgang aber nur setzen kann, macht das geprellte Signal kein Problem. Erst wenn der Taster wieder losgelassen wird, und ein aktives Signal auf den Reset-Eingang des Flip-Flops setzt, wird der Ausgang wieder zu 0. Somit kann ein sauber entprelltes Signal am Ausgang des RS-FF erzeugt werden.

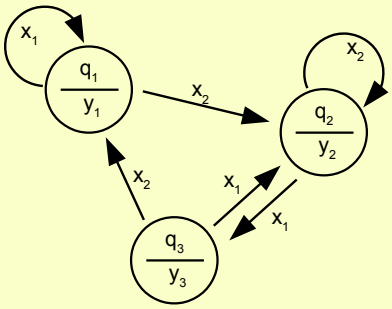
2.5.Automaten

Mit den eben beschriebenen Flip-Flops können Werte gespeichert werden, und damit kleine Programme auf Logikebene realisiert werden. Diese „Programme“ werden als Automaten dargestellt. Da Automatentheorie ein eigenes Buch füllen würde, beschränkt sich dieses Tutorial auf die Grundlagen des Moore-Automaten.

Zum Einen weil dieser einfach verständlich ist und für die allermeisten Fälle ausreicht, zum Anderen weil der andere Automatentyp (Mealy-Automat) Tücken mit sich bringt, auf die geachtet werden muss.

Moore-Automat

Zustandsübergangsdiagramm



Zustandsübergangstabelle

Ist-zustand	Folge-zustand		Ausgangs-zeichen
	x_1	x_2	
q_1	q_1	q_2	y_1
q_2	q_3	q_2	y_2
q_3	q_2	q_1	y_3

Abb. 7 - Moore-Automat

Der Moore-Automat wird in aller Regel in einem Zustandsübergangsdiagramm dargestellt. In diesem Diagramm sind die Zustände, in dem sich der Automat befindet als Kreise dargestellt, die Zustandsübergänge als Pfeile.

Die Bezeichnung der Zustände steht in dem Kreis über der Linie, in Abb. 7 z.B. q_1. Der Wert, der in diesem Zustand am Ausgang anliegt, wird in dem Kreis unter der Linie dargestellt. Um von einem Zustand in den nächsten zu kommen, ist eine Zustandsübergangsbedingung erforderlich. Dieser am Eingang anliegende Wert wird über dem Pfeil dargestellt.

In dem Moore-Automaten in Abb. 7 bedeutet das konkret: befindet sich der Automat im Zustand q_1, liegt am Ausgang der Wert y_1 an. Liegt am Eingang der Wert x_1 an, verbleibt der Automat im Zustand q_1. Liegt am Eingang der Wert x_2 an, so wechselt der Automat in den Zustand q_2. In diesem Zustand liegt dann der Wert y_2 am Ausgang an.

Moore Automaten werden immer synchron betrieben, d.h. der Automat funktioniert mit dem zentralen Takt des Automaten. Ein Zustandsübergang kann so nur mit jeder steigenden Taktflanke erfolgen.

Die Funktionsweise der Moore Automaten kann wieder am besten mit einem konkreten Beispiel veranschaulicht werden. Ziel soll es sein, ein einfaches Zahlenschloss mit Hilfe eines Moore-Automaten zu realisieren. Dieses besonders einfache Zahlenschloss besteht aus zwei Tasten, T_1 und T_2 und einem Schloss S. Dieses Schloss kann mit der Kombination T_1, T_1, T_2 geöffnet werden. Der Moore-Automat dieses Schlosses ist in Abb. 8 dargestellt.

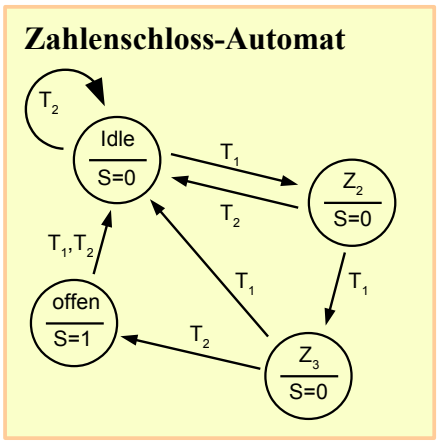


Abb. 8 - Zahlenschloss-Automat

Zu Beginn befindet sich der Automat im Zustand Idle. Wird die Taste T_2 gedrückt, verbleibt der Automat in diesem Zustand. Erst wenn die Taste T_1 gedrückt wird, wechselt der Automat in den Zustand Z_2. Auch in diesem Zustand muss die richtige Taste T_1 gedrückt werden damit der Automat in den Zustand Z_3 wechselt, sonst wechselt er wieder in den Zustand Idle. Sobald nun die Taste T_2 gedrückt wird, geht der Automat in den Zustand offen über und das Schloss öffnet sich mit S=1. Bei einer beliebigen Taste geht der Automat wieder in den Startzustand Idle über. Auf diese Weise kann sehr einfach ein Automat für ein Zahlenschloss realisiert. werden. Dieser ist natürlich absolut minimalistisch, bei einem richtigen Zahlenschloss müsste es mehrere Tasten und eine längere Kombination geben. Aber zur Anschauung, wie solch ein Automat funktioniert, ist dieser ideal. Mit den in diesem Kapitel beschriebenen Grundlagen der Logik können bereits sehr komplexe Schaltungen realisiert werden. Welche Hard- und Software dazu benötigt wird, wird in den folgenden Kapiteln beschrieben.