Besondere Lernleistung:

Entwicklung und Realisierung einer Universalinfrarotfernbedienung mit Timerfunktionen

von

Sven Schwermer

Hamburg, 13.01.2009

Inhaltsverzeichnis

1. Einleitung und Motivation

Ich habe in meinem Zimmer ein kleines Arsenal an Unterhaltungselektronik stehen, das ich mein Eigen nennen darf:

1. Denon DRA-700AE (Stereoreceiver)
2. Mustek DVB-T102 (DVB-T-Receiver)
3. Sony KV-C27 TD (Fernseher)
4. Pioneer DV-600AV (DVD-Player)

Nun hat man ja dank des Fortschritts in der Technik die Möglichkeit all seine Unterhaltungselektronik bequem vom Bett oder Sofa aus fernzubedienen. Doch dieser technische Meilenstein hat natürlich auch Schattenseiten. Eine davon möchte ich hier einmal ausleuchten: Die Fernbedienungen häufen sich auf dem Sofatisch oder Nachtschrank. Jetzt wird der ein oder andere sagen: „Wozu gibt's denn Universalfernbedienungen?". Derjenige hat natürlich Recht, allerdings habe ich Ansprüche an eine Universalfernbedienung, die anscheinend noch nicht viele Leute vor mir hatten, denn meine Wunschfernbedienung habe ich noch nicht im Laden finden können. So muss die Fernbedienung beispielsweise eine Timerfunktion haben, die zu einer einstellbaren Uhrzeit bestimmte Signale aussendet und so den Fernseher, den Stereoreceiver und den DVB-T-Receiver einschaltet. Als Pendent sollte ein Sleeptimer existieren, der Signale zum Ausschalten aller Geräte sendet. Die von mir ausgedachte Fernbedienung würde also gleich mehrere nützliche Funktionen miteinander in einem Gerät vereinen. Das ist ja alles leicht gesagt, nur wie soll man diese „eierlegende Wollmilchsau" bauen? Mit dieser und noch wesentlich grundlegenderen Fragen werde ich mich in den folgenden Seiten näher befassen.

Da oft recht spezifische Begriffe verwendet werden, habe ich ein Glossar angehängt, das viele Unklarheiten aus der Welt schaffen sollte. Ein kurzes Wort zur Notation von Zahlenwerten: Ziffernfolgen, die mit dem Präfix *0x* beginnen, beschreiben Hexadezimalzahlen, *0b* deutet auf binäre Werte hin und Zahlen ohne Präfix sollen als dezimal gedeutet werden. Hexadezimalzahlen werden in der Programmierung recht häufig verwendet, weil zwei Hexadezimalziffern genau 2^8 Zustände, was einem Byte entspricht, darstellen können.

2. Infrarotfernbedienungen

Der weit überwiegende Teil der Fernbedienungen, die uns in unserem täglichen Leben begegnen, sind so genannte Infrarotfernbedienungen. Das bedeutet, dass sie die Informationen über eine gedrückte Taste über Lichtwellen übertragen werden. Allerdings wird hierbei nicht Licht aus dem für den Menschen sichtbaren Spektralbereich verwendet, sondern aus dem des nahen infraroten Licht (near infrared, NIR). Bei Infrarotfernbedienungen sind Wellenlängen um 950nm üblich. Wie bei jedem Licht muss auch in diesem Fall eine Sichtverbindung zwischen Lichtquelle und Lichtempfänger bestehen. In der Praxis muss man also die Fernbedienung in Richtung des fernzubedienenden Gerätes halten. Es ist jedoch auch möglich, dass das Gerät auf ein Fernbedienungssignal reagiert, obwohl man die

Fernbedienung in eine ganz andere Richtung hält. Das kommt daher, dass infrarotes Licht – genau wie jedes andere Licht auch – von Körpern reflektiert wird, also indirekt zum Gerät gelangt.

Bei der Datenübertragung mittels Licht ist wichtig, dass es nur zwei Zustände gibt; an und aus. Die Daten werden also nicht analog, sondern digital übertragen. Jede Taste auf der Fernbedienung bewirkt nun einen anderen Code, der binär übertragen wird. Nehmen wir also an, eine imaginäre Taste soll bei Tastendruck folgenden Code an das Gerät übermitteln: `010101`, wobei `0` Licht aus und `1` Licht an bedeutet. Das Signal dazu würde so aussehen:

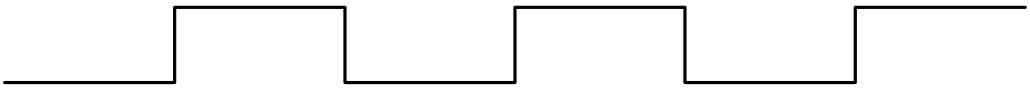

Abb. 1: Signaldarstellung von 010101

Aus Gründen der Störsicherheit wird das Signal allerdings nicht wie in Abbildung 1 gezeigt, sondern moduliert übertragen. Das bedeutet, dass wenn eine 1 übertragen wird, die Fernbedienung also infrarotes Licht emittiert, in Wirklichkeit nur kurze Lichtimpulse übertragen werden. Die Dauer dieser Lichtimpulse wird von der Modulationsfrequenz bestimmt. Das Signal aus Abbildung 1 sieht folglich schematisch dargestellt so aus:

Abb. 2: Modulierte Signaldarstellung von 010101

Das modulierte Lichtsignal wird dann auf Empfängerseite wieder demoduliert, sodass das Signal wieder wie in Abbildung 1 aussieht. So wird erreicht, dass nur moduliertes Licht „durchgelassen“ wird. Störendes Sonnenlicht und das Licht aus Glühlampen, die auch infrarote Anteile beinhalten, wird so erst gar nicht vom Empfänger wahrgenommen.

In der Realität sind die Signale nicht so einfach, wie oben dargestellt (`1`=Licht an; `0`=Licht aus), sondern durch bestimmte Definitionen festgelegt. `0`en und `1`en sind meist durch einen HIGH- (Licht an) und LOW-Puls (Licht aus) dargestellt. Die Länge der HIGH- und LOW-Phasen und/oder deren Reihenfolge geben an, ob es sich um eine `1` oder eine `0` handelt. Als Beispiel sieht man in Abbildung 3 die Bitdefinitionen des JAPAN-Protokolls.

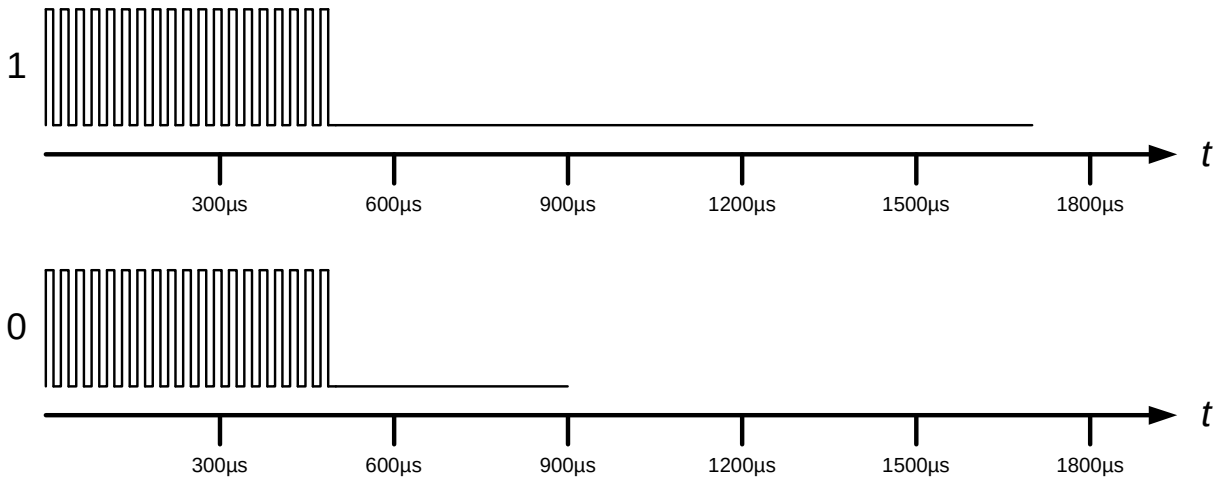

Abb. 3: Bitdefinitionen des JAPAN-Protokolls

Da nahezu jeder Hersteller ein anderes Protokoll, eine andere Modulationsfrequenz und andere Definitionen von `0` und `1` verwendet, musste ich diese drei Variablen bei jeder Fernbedienung neu herausfinden.

3. Analyse der gegebenen Fernbedienungen

Wie bereits erwähnt, arbeiten die Fernbedienungen verschiedener Hersteller meist anders. Die verschiedenen Protokolle der Hersteller sind teilweise sehr gut im Internet dokumentiert, andere sind wiederum lückenhaft, fehlerhaft oder gar nicht dokumentiert. Deshalb musste ich mir einen Weg überlegen die Tastencodes herauszufinden. Eine IR-Fotodiode an einem Oszilloskop machte die Signale schon einmal sichtbar und man konnte sogar mit viel Mühe die einzelnen Tastencodes herausfinden. Das hätte jedoch für die vielen Tasten zu lange gedauert. Also habe ich eine kleine Schaltung entworfen, die alle notwendigen Informationen an den PC schickt, wo die Daten ausgewertet werden konnten.

a. Empfängerschaltung

Als Infrarotempfängerbaustein wählte ich einen TSOP1736. Das ist ein Baustein, wie er auch in den meisten Geräten, die fernbedient werden, zu finden ist. Er vereint Infrarotfilter, Fotodiode, Vorverstärker und Demodulator. Man kann also am Ausgang ein „fertiges“ 5V-Signal abgreifen. Das folgende Schaubild veranschaulicht den Aufbau des TSOP1736:

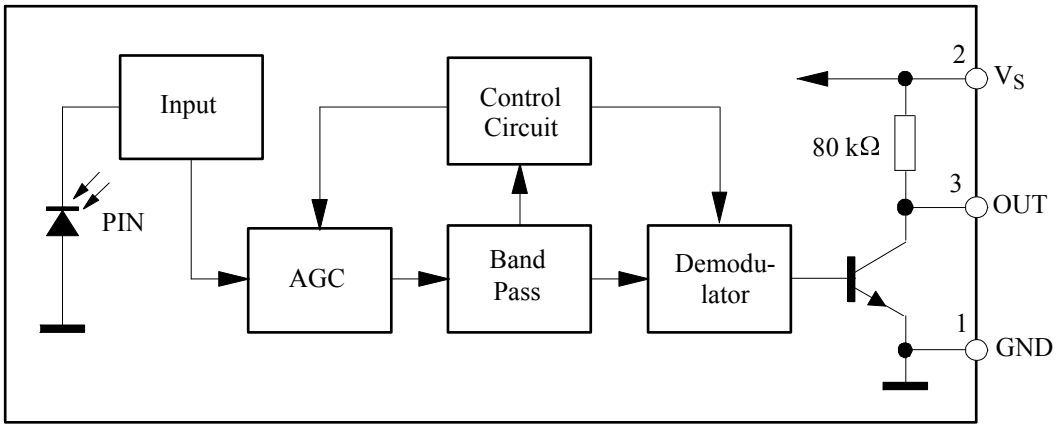

Abb. 4: Blockschaltbild des TSOP1736

Der TSOP1736 ist – wie die Bezeichnung schon sagt – auf eine Modulationsfrequenz von 36kHz abgestimmt. In der Praxis ist er allerdings diesbezüglich recht großzügig, was das folgende Diagramm (Abbildung 5) aus dem Datenblatt des TSOP1736 zeigt:

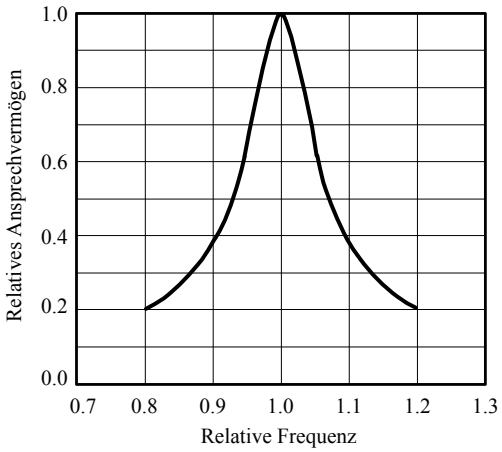

Abb. 5: Frequenzabhängigkeit vom Ansprechvermögen

Man sieht, dass der TSOP1736 bei 28,8kHz bzw. 43,2kHz immer noch 20% seines Ansprechvermögens hat, was bei extrem kurzen Übertragungsdistanzen (bei der Analyse etwa 5cm) locker ausreicht um fehlerfreie Ergebnisse zu erzielen. Diese Spanne ist bei der Analyse allerdings auch zwingend notwendig, denn man weiß ja vorher nicht unbedingt, welche Modulationsfrequenz verwendet wird.

Ich baute also eine kleine Schaltung auf, in der der TSOP1736 an einem externen Interruptpin (INT0) eines Atmel ATmega16 Mikrocontroller angeschlossen ist, welcher wiederum über einen RS232-Treiber (MAX232) an die serielle RS232-Schnittstelle des Computers angeschlossen ist.

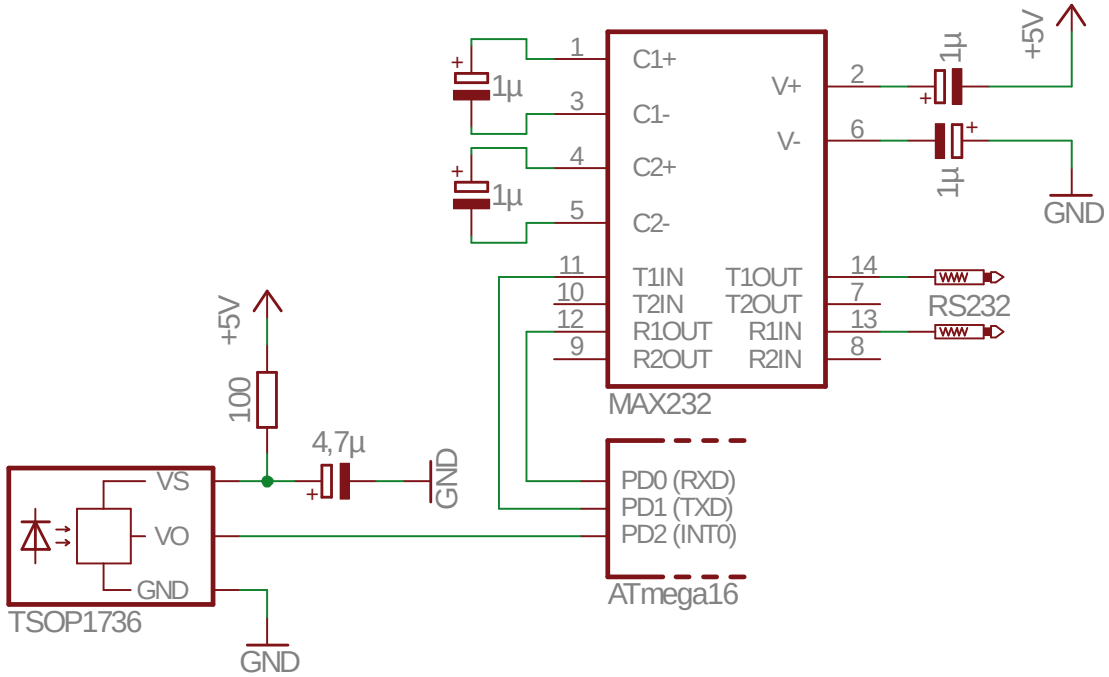

Abb. 6: Schaltplan der Empfängerschaltung (ATmega16 stark vereinfacht dargestellt)

b. Empfängerprogramm

Das Programm der Analyseschaltung mit dem TSOP1736 als Empfängerbaustein misst die Zeit zwischen zwei Flanken an PD2 (INT0) und schickt diese an die RS232-Schnittstelle. Im Folgenden ist die leicht vereinfachte C-Firmware des ATmega16 zu sehen:

```
1   #include <avr/io.h>
2   #include <avr/interrupt.h>
3   #include "uart.h"
4
5   #define UART_BAUD_RATE 19200
6   #define SPEICHER 120
7
8   volatile uint8_t counter = 0;
9   volatile uint16_t daten[SPEICHER];
10
11  ISR(INT0_vect){
12     uint16_t timer = TCNT1;
13     TCNT1 = 0;
14     MCUCR ^= (1<<ISC00);
15     daten[counter] = timer;
16     counter++;
17  }
18
19  int main(void){
20     uart_init(UART_BAUD_SELECT(UART_BAUD_RATE, F_CPU));
21
22     sei();
23
24     DDRD &= ~(1<<PD2);
25     MCUCR |= (1<<ISC01);
26     GICR |= (1<<INT0);
27     TCCR1B |= (1<<CS11);
28
29     while(1){
30        if(counter == SPEICHER+1){
31           GICR &= ~(1<<INT0);
32
33           uint8_t start = 0;
34           for(uint8_t i=0; i<SPEICHER+1; i++){
35              uart_putc((char) (daten[i]/10000)+48);
36              uart_putc((char) ((daten[i]/1000)%10)+48);
37              uart_putc((char) ((daten[i]/100)%10)+48);
38              uart_putc((char) ((daten[i]/10)%10)+48);
39              uart_putc((char) (daten[i]%10)+48);
40              uart_putc('\r');
41           }
42
43           counter++;
44        }
45     }
46
47     return 0;
48  }
```

Listing 1: Analyse-Empfängerprogramm

c. Ergebnisse

Wenn man nun die Fernbedienung auf den TSOP1736 richtet und eine zu analysierende Taste etwas länger drückt (bis das *daten*-Array voll ist), sendet der Controller alle Werte über die serielle Schnittstelle an den Computer. An diesem kann man sich die Daten über ein Terminalprogramm (z.B. Hyperterminal) anzeigen lassen. Aus dem Terminalprogramm lassen sich die Daten problemlos in ein Excel-Dokument kopieren. Dort kann man die Datensätze der unterschiedlichen Tasten einer Fernbedienung miteinander vergleichen und das Übertragungsprotokoll herausfiltern. Nachfolgend werde ich auf die Übertragungsprotokolle aller vier Geräte näher eingehen:

Denon DRA-700AE

Bei Recherchen im Internet fand ich heraus, dass es einen so genannten Denon-Code gibt, der aus 5 Bits Geräteadresse, 10 Bits Befehl und einem Stopbit besteht. Als ich das Signal der Fernbedienung jedoch mit meiner Analyseschaltung auswertete stieß ich auf einen 50 Bits langen Code. Nach weiteren Recherchen stieß ich auf die IR-Code-Datenblätter anderer Denon-Receiver, in denen nicht nur vom Denon-Code, sondern auch von Kaseikyo-Code die Rede war. Dieser Bestand aus 48 Bits plus Start- und Stopbit. Der Kaseikyo-Code, der üblicherweise Japan-Code genannt wird, wurde von einem japanischen Ausschuss, der aus Haushaltsgeräteherstellern besteht (Japan's Association for Electric Home Application), festgelegt und wird von mehreren Herstellern aus Fernost verwendet, darunter auch Denon.

Der Japan-Code ist mit 38kHz moduliert. Gestartet wird eine Sequenz mit einem Startbit, das aus ca. 3,4ms HIGH- und ca. 1,7ms LOW-Phase besteht. Ein `1`-Bit wird mit ca. 0,5ms HIGH- und ca. 1,2ms LOW-Phase definiert, während ein `0`-Bit auch mit ca. 0,5ms HIGH- allerdings mit nur ca. 0,4ms LOW-Phase bestimmt wird. Als Stopbit wird einfach ein `0`-Bit an die 48 Datenbits gehängt. Diese komplette Folge von Startbit, Datenbits und Stopbit wird mit einem Abstand von ca. 74,4ms wiederholt, solange die Taste auf der Fernbedienung gedrückt wird. Alle Zeiten sind relativ unkritisch. Was mich überraschte, war die Tatsache, dass die Zeiten derselben Fernbedienung manchmal erheblich abwichen.

Nun zum eigentlichen Datenteil, der insgesamt aus 48 Bits besteht: Da der Japan-Code von verschiedenen Herstellern verwendet wird, wird zuerst eine 2 Byte (16 Bit) lange Herstellerkennung übermittelt. Für Denon ist das `0010101001001100` (erstgesandtes Bit zuerst). Als nächstes folgt die Herstellerparität, die gebildet wird, indem die 4 Herstellernibbles mit XOR (Addition mit Modulo 2) verknüpft werden. Im Falle Denon also `0000`. Anschließend werden 2 Genre-Nibbles übertragen, die die Art des Gerätes festlegen. Meinem Receiver wurde `1110` bzw. `0110` zugewiesen. Es folgen 10 Datenbits, die die eigentliche Funktion festlegen. Die 2 folgenden ID-Bits sind dafür bestimmt verschiedene gleiche Fernbedienungen zu unterscheiden. Standardmäßig wird ID 1 verwendet, wobei die ID-Bits folgendermaßen aussehen: `00`. Als letztes wird noch eine 8 Bit lange Parität des dritten, vierten und fünften Bytes (mit XOR verknüpft) gesendet. Das Ganze ist in Tabelle 1 noch einmal anhand der ON-Taste dargestellt.

Herstellercode																**Herstellerparität**				**Genre 1**			
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0	0	1	0	1	0	1	0	0	1	0	0	1	1	0	0	0	0	0	0	1	1	1	0
Genre 2				**Daten**										**ID**		**Parität**							
25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0

Tabelle 1: Bitfolge der ON-Taste auf der Denon-Fernbedienung (Japan-Code)

Bei der weiteren Analyse der Tasten fiel auf, dass nicht alle Tasten im Japan-Code codiert waren, sondern auch einige im Denon-Code. Da diese Tasten allerdings allesamt für die

Bedienung eines CD-Players der Marke Denon vorgesehen sind und ich keinen solchen besitze, habe ich die nähere Analyse und Nachbildung dieser Tasten nicht in Angriff genommen.

Natürlich sind mit den auf der mitgelieferten Fernbedienung angebrachten Tasten noch nicht alle Möglichkeiten für die 10 Datenbits ausgeschöpft. Denn immerhin sind mit 10 Bits 2^{10}=1024 Möglichkeiten gegeben, von denen lediglich 27 benutzt werden. Also schrieb ich ein kleines Programm, das alle Möglichkeiten durchgeht. So entdeckte ich Funktionen, die auf der Fernbedienung gar nicht abrufbar sind, wie beispielsweise Toneinstellungen, Displaydimmer oder das direkte Abrufen von Radiosendern. Das Suchen hatte sich also gelohnt.

Mustek DVB-T102

Über Mustek-Fernbedienungen war im Internet nichts zu finden, auch der Support der Firma konnte mir nicht weiterhelfen. Also musste ich ganz von Vorn anfangen und habe mit meiner Analyseschaltung den Code untersucht. Beim Vergleichen des übersandten Codes mit Beschreibungen einiger IR-Protokolle stellte sich heraus, dass es sich hierbei um das Erweiterte NEC-Protokoll handelte. Ob dies der offizielle Name ist, kann ich nicht genau sagen, da auf der entsprechenden Website auch nur Vermutungen angestellt wurden.

Das Erweiterte NEC-Protokoll ist genau wie das NEC-Protokoll mit 38kHz moduliert. Eine Übertragungssequenz beginnt mit einem Startbit, das aus ca. 9,1ms HIGH- und ca. 4,4ms LOW-Phase besteht. Ein `1`-Bit wird mit ca. 0,6ms HIGH- und ca. 1,6ms LOW-Phase definiert, während ein `0`-Bit auch mit ca. 0,6ms HIGH- allerdings mit nur ca. 0,5ms LOW-Phase bestimmt wird. Als Stopbit wird einfach eine 0,6ms lange HIGH-Phase an die 32 Datenbits gehängt. Das besondere an diesem Protokoll ist nun, dass bei gedrückter Taste nicht die komplette Datensequenz noch mal übertragen wird, sondern nur eine Wiederholungssequenz, die aus ca. 9,1ms HIGH-, 2,2ms LOW und 0,6ms HIGH-Phase besteht. Die Datensequenz und die Wiederholungssequenzen werden in einem Intervall von ca. 108,4ms gesendet. Der Vorteil dieser Methode ist, dass die Infrarotdiode weniger leuchtet, also weniger Strom verbraucht wird, was sich in der Batterielebensdauer bemerkbar macht. Der Nachteil ist, dass man beim Drücken der Taste direkt den Empfänger des zu bedienenden Gerätes treffen muss, weil nach der ersten Sequenz keine Informationen übermittelt werden. Das kann man jedoch sehr einfach umgehen, indem man einfach die Datensequenz erneut sendet und die Wiederholungssequenz ganz weglässt. Das wird vom Gerät genauso gut verstanden.

Der Datenteil gliedert sich in 16 Bits Adresse, die das zu bedienende Gerät bestimmen, und 16 Bits Befehl. Beim NEC-Protokoll werden zuerst 8 Bits Adresse gesendet und anschließend 8 Bits der invertierten Adresse (1 wird durch 0 ausgetauscht und umgekehrt). Genauso läuft es bei dem Befehl. Das dient der Störsicherheit, weil die gleiche Information zweimal übertragen wird. Der Unterschied zum Erweiterten NEC-Protokoll ist, dass bei jenem das zweite Adressbyte nicht die Invertierung des ersten ist. Somit stehen nun 16 Adressbits zur Verfügung, was die Anzahl unterschiedlicher Geräte, die mit dem (Erweiterten) NEC-

Protokoll angesprochen werden können, deutlich erhöht. Ansonsten sind die beiden Protokolle identisch. Nachfolgend ist in Tabelle 2 einmal die Bitfolge der POWER-Taste dargestellt.

Adresse															
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	1	0	0	0	1	0	1	1	0	1	1	1
Befehl								**Invertierter Befehl**							
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1

Tabelle 2: Bitfolge der POWER-Taste auf der Mustek-Fernbedienung (Erweitertes NEC-Protokoll)

Sony KV-C27 TD

Der von Sony verwendete SIRC-Code ist sehr gut im Internet dokumentiert, weshalb ich nicht einmal die Fernbedienung mit meiner Schaltung analysieren musste.

Der SIRC-Code ist mit 40kHz moduliert. Zuerst wird ein Startbit bestehend aus 2,4ms HIGH- und 0,6ms LOW-Phase übertragen. Ein `1`-Bit wird mit ca. 1,2ms HIGH- und ca. 0,6ms LOW-Phase definiert, während ein `0`-Bit nur mit ca. 0,6ms HIGH- allerdings auch mit ca. 0,6ms LOW-Phase bestimmt wird. Die Datensequenzen werden in einem Intervall von 45ms übertragen.

Die Datensequenz gliedert sich in 7 Befehlsbits und 5 Adressbits. Die Datensequenz für die POWER-Taste sieht wie in Tabelle 3 beschrieben aus.

Befehl							**Adresse**				
1	2	3	4	5	6	7	8	9	10	11	12
1	0	1	0	1	0	0	1	0	0	0	0

Tabelle 3: Bitfolge der POWER-Taste auf der Sony-Fernbedienung (SIRC-Code)

Pioneer DV-600AV

Die Pioneer-Fernbedienung verwendet wie die von Denon zwei Protokolle. Für die grundsätzlichen Funktionen wie Play, Pause, Stop, Next und Previous wird das NEC-Protokoll verwendet, während bei allen anderen Tasten eine Abwandlung dessen benutzt wird. Die gemessenen Timings sind etwas anders als die der Mustek-Fernbedienung. Die Abweichungen sind jedoch relativ gering, weshalb ich hier nicht näher darauf eingehen möchte. Lediglich das Intervall, in dem die Sequenzen übertragen werden, ist mit ca. 91ms signifikant unterschiedlich.

Wie bereits erwähnt, werden die Grundfunktionen im normalen NEC-Protokoll gesendet. So wird erst ein Adressbyte, anschließend das invertierte Adressbyte, dann das Befehlsbyte und schließlich das invertierte Befehlsbyte übermittelt. In Tabelle 4a ist noch einmal die Datensequenz der PLAY-Taste zu sehen.

Adresse								**Invertierte Adresse**							
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	0	0	0	1	0	1	0	0	1	1	1	0	1	0
Befehl								**Invertierter Befehl**							
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
0	1	1	1	1	0	0	1	1	0	0	0	0	1	1	0

Tabelle 4a: Bitfolge der PLAY-Taste auf der Pioneer-Fernbedienung (NEC-Protokoll)

Bei den nicht grundsätzlichen (oben genannten) Tasten werden immer zwei 32-Bit-Sequenzen übermittelt, wobei die erste Sequenz bei allen Tasten gleich ist. Diese beinhaltet die Adresse, die auch bei den grundsätzlichen Tasten verwendet wurde und immer den gleichen Befehl, der dem Gerät vermutlich mitteilt, dass die nachfolgende Sequenz auch noch beachtet werden muss bis die entsprechende Funktion ausgeführt wird. Die zweite Sequenz beinhaltet auch immer die gleiche Adresse, die allerdings von der aus der ersten Sequenz abweicht, und einen Befehl, der die Funktion der Taste bestimmt. In Tabelle 4b ist die (immer gleiche) erste Sequenz zu sehen.

Adresse 1								**Invertierte Adresse 1**							
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	0	0	0	1	0	1	0	0	1	1	1	0	1	0
Immergleicher Befehl								**Invertierter Befehl**							
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	0	0	1	1	0	0	1	0	1	1	0	0	1	1	0

Tabelle 4b: Bitfolge der ersten Sequenz der nicht grundsätzlichen Tasten auf der Pioneer-Fernbedienung

Nachfolgend in Tabelle 4c ist die zweite Sequenz der STANDBY/ON-Taste zu sehen.

Adresse 2								**Invertierte Adresse 2**							
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	1	1	0	1	0	1	0	0	0	0	1	0	1	0
Wechselnder Befehl								**Invertierter Befehl**							
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	1	1	1	1	0	1	1	1	0	0	0	0	1	0

Tabelle 4c: Bitfolge der zweiten Sequenz der nicht grundsätzlichen Tasten auf der Pioneer-Fernbedienung

Was für einen Sinn das Ganze hat, kann ich mir nicht erklären, allerdings kann ich mir vorstellen, dass das Ganze aus der Historie entstanden ist und keinen technischen Hintergrund hat.

4. Entwicklung einer eigenen Universalfernbedienung

Bei der Entwicklung meiner eigenen Universalfernbedienung stehen wie im Projekttitel bereits erwähnt die Timerfunktionalitäten im Vordergrund. Aber auch andere Funktionen wie Direktkommandos und so weiter sollten möglich sein. Um das zu ermöglichen bräuchte man entweder extrem viele Tasten und Platz oder eine Art Menüsystem. Ich habe mich für letzteres entschieden. Das Menüsystem wird mit einem Display und einem Drehimpulsgeber mit Taster realisiert.

a. Hardware

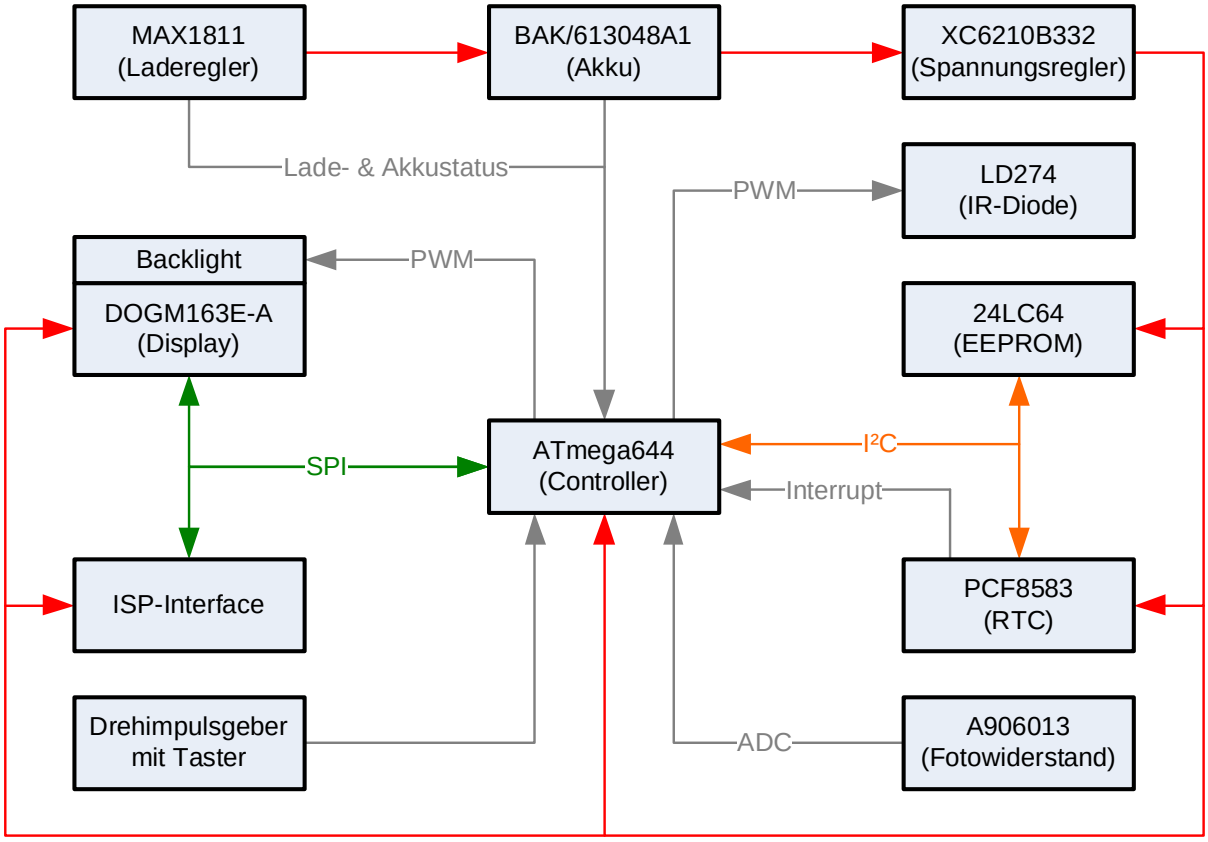

Abb. 7: Blockschaltbild der Fernbedienung

Das Kernstück der Fernbedienung ist ein Atmel **ATmega644** Controller. Dieser bietet unter Anderem die folgenden Kernfunktionen:

- Bis zu 20 MIPS bei 20MHz (MIPS = Millionen Instruktionen Pro Sekunde)
- 64kB Flash-Programmspeicher
- 2kB EEPROM (EEPROM = Electrically Erasable Programmable Read-Only Memory)

- 4kB Static Random Access Memory (SRAM)
- 2* 8-Bit und 1* 16-Bit Timer/Counter
- 6 PWM-Kanäle (PWM = Pulsweitenmodulation)
- 8 10-Bit ADC-Kanäle (ADC = Analog/Digital Converter)
- I²C-Bus (I²C = Inter-Integrated Circuit)
- SPI-Bus (SPI = Serial Peripheral Interface)
- Verschiedene Schlafmodi
- In der Hochsprache C programmierbar

Auf diesem Controller läuft das Hauptprogramm und die gesamte Peripherie wird von hier gesteuert. Die wichtigste Komponente einer Infrarotfernbedienung ist die Infrarotleuchtdiode. In diesem Fall ist das die **LD274**. Sie hat einen engen Abstrahlwinkel und eine hohe Impulsbelastbarkeit (in meinem Fall 500mA). Somit ist sie gut für Infrarotfernbedienungen geeignet. Als Energieträger dient ein Lithium-Ionen-Akku mit der Bezeichnung **BAK/613048A1**. Er ist eigentlich ein Ersatzakku für Camcorder und somit sehr günstig zu beziehen. Er hat eine Kapazität von 900mAh und eine Nennspannung von 3,7V. Da die Akkus dieser Technologie recht kompliziert zu laden sind, wird ein Laderegler vom Typ Maxim **MAX1811** verwendet. Nun galt es nur die Ladeschlussspannung des Akkus herauszufinden, da diese um 0,05V genau eingehalten werden muss. Ansonsten könnte der Akku und der Anwender Schaden nehmen (Explosion). Bei Recherchen auf der chinesischen Herstellerwebsite kam 4,2V als Ladeschlussspannung heraus. Geladen wird der Akku mit 100mA Ladestrom. Der aktuelle Ladestatus wird über den CHG-Pin des MAX1811 an den ATmega644 mitgeteilt. Die Akkuspannung wird über einen Spannungsteiler an einem ADC-Pin des ATmega644 gemessen. Der Spannungsteiler ist mit einem Gesamtwiderstand von 5MΩ sehr hochohmig, was die mögliche Samplingrate aber auch den kontinuierlichen Stromverbrauch herabsetzt. Da die Spannung jedoch mit steigender Entladung sinkt und somit variabel ist, muss die Spannung auf 3,3V geregelt werden. Dies geschieht mit dem **XC6210B332**. Das ist ein Festspannungsregler der Marke Torex mit einem sehr geringen Spannungsabfall, d.h. die Differenz zwischen Eingangs- und Ausgangsspannung (50mV bei 100mA Ausgangsstrom). Außerdem hat er einen äußerst geringen Selbstverbrauch von 35µA. Das typische Limit des Ausgangsstroms ist 800mA und damit sehr großzügig ausgelegt für mein Projekt. Der Regler versorgt alle peripheren Geräte. Die Schnittstelle zum Benutzer besteht aus einem dreizeiligen 16-Zeichen-Display vom Typ **DOGM163E-A** der Marke Electronic Assembly, das grün hintergrundbeleuchtet ist. Die Kommunikation erfolg seriell über den SPI-Bus. Die Hintergrundbeleuchtung besteht aus LEDs (Light Emitting Diode) und sollte ursprünglich abhängig von der Umgebungshelligkeit per PWM gedimmt werden. Die Umgebungshelligkeit sollte über einen Spannungsteiler mit einem LDR (Light Dependent Resistor) vom Typ **A906013** an einem ADC-Pin des ATmega644 erfasst werden. Als Eingabegerät steht dem Benutzer ein Drehimpulsgeber mit Taster vom Typ **STEC11B** der Firma ALPS zur Verfügung. Er gibt einen Gray-Code aus, wodurch sich die Drehrichtung bestimmen lässt. Zur Speicherung der Fernbediencodes sollte ein EEPROM vom Typ **24LC64** verwendet werden. Es bietet 64kBits (= 8kB) Platz und wird seriell über den I²C-Bus beschrieben und ausgelesen. Um den Controller zu entlasten verwende ich eine externe Real-Time-Clock vom Typ **PCF8583**, sie läuft selbstständig weiter und kann einen Alarm speichern. Wenn der Alarm auslöst wird ein Interrupt im ATmega644 ausgelöst, womit dieser

auch aus dem Schlafmodus geweckt werden kann. Das ist für die Timerfunktionalitäten von entscheidender Bedeutung. Um den ATmega644 zu programmieren wird das **ISP-Interface** (In System Programming) verwendet. Es basiert auf dem SPI-Bus. Um den Programmiermodus zu aktivieren, wird der RESET-Pin des Controllers auf LOW gezogen. Zusätzlich wird ein Programmiergerät, das die Schaltung mit dem PC verbindet, benötigt. Dafür gibt es diverse einfache bis komplizierte Schaltungen, für LPT-, COM- oder USB-Schnittstelle. Ich baute mir extra für dieses Projekt ein neues Programmiergerät mit dem Namen USBasp (www.fischl.de) für den USB-Port, da es in einer Variation mit 3,3V-Schaltungen klarkommt. Über vier Messingkontakte werden folgende Leitungen aus dem Gehäuse der Fernbedienung gebracht: +5V und GND (zum Anschluss der Ladespannung) sowie RX- und TX-Pin der UART-Schnittstelle (Universal Asynchronous Receiver Transmitter). Dadurch kann der Controller später auch ohne Öffnen des Gehäuses mit der Außenwelt kommunizieren.

b. Ladeschale

Die Zuvor angesprochenen Messingkontakte müssen natürlich auch entsprechende Gegenkontakte haben. Diese habe ich mit einer Ladeschale realisiert, in der vier Messingstifte mittels Federn gegen die Messingkontakte der Fernbedienung drücken und so für einen sicheren Kontakt sorgen. Messing habe ich deshalb gewählt, weil es relativ korrosionsbeständig, leicht zu bearbeiten und gut lötbar ist sowie eine sehr gute Leitfähigkeit besitzt.

In der Ladeschale ist aber nicht nur die 5V-Spannungsversorgung für den MAX1811, sondern auch ein USB-Schnittstellenbaustein (FTDI FT232RL) verbaut, der dem PC einen virtuellen COM-Port vorgaukelt und mit dem ATmega644 über UART kommunizieren kann. Man hätte genauso gut einen MAX232 verwenden können, aber USB ist ganz einfach zukunftssicherer als RS232. Über die so hergestellte Schnittstelle mit dem PC, kann man den ATmega644 zum Beispiel über einen Bootloader neu programmieren (mehr dazu später). Man kann über einen Schalter wählen, ob man die Fernbedienung über die 5V des USB-Ports oder über ein Steckernetzteil, dessen Spannung von einem 7805 auf 5V gebracht wird, laden will. Die Platine, die in der Ladeschale verbaut ist, wurde nach folgendem Schaltbild entworfen.

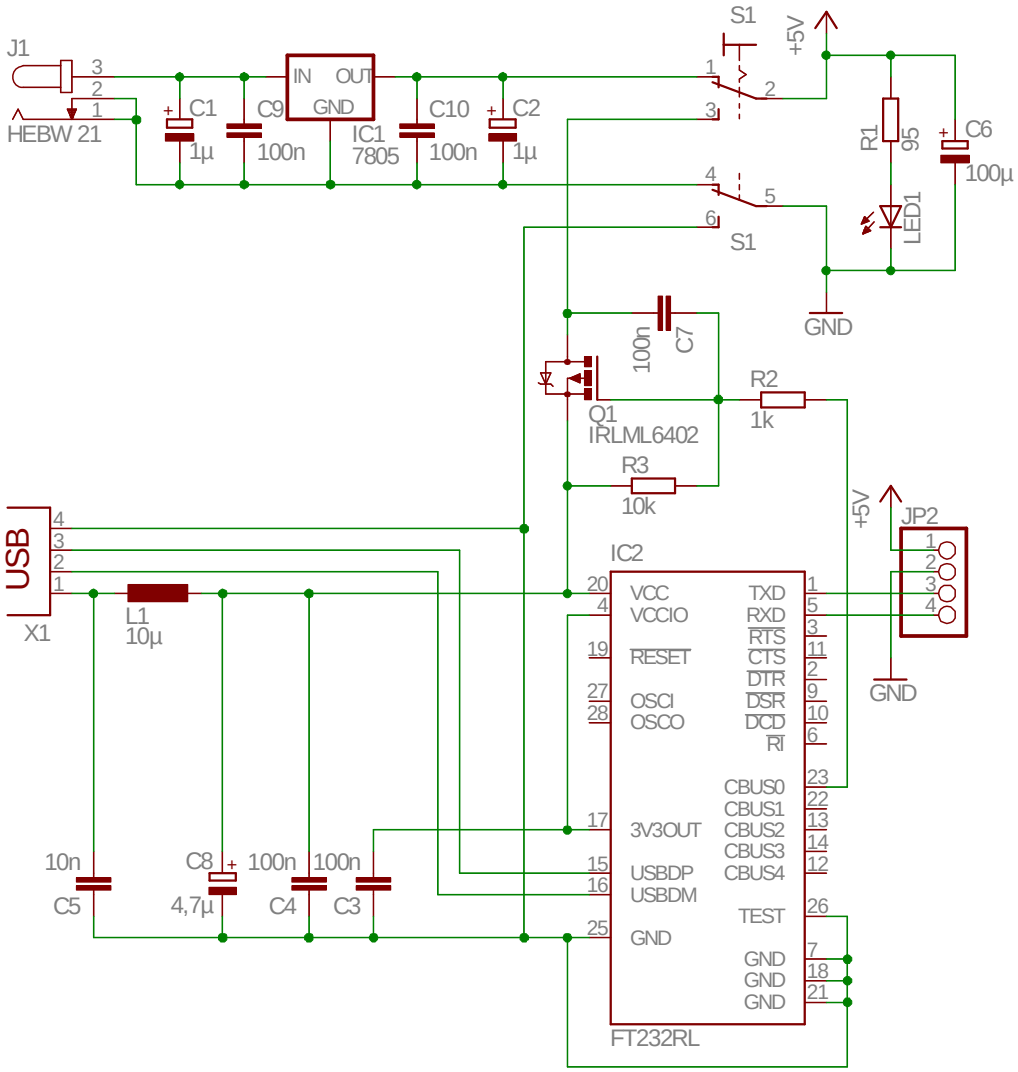

Abb. 8: Schaltbild der Ladeschale

Die USB-Schnittstelle stellt seinen Verbrauchern 5V mit einer Strombegrenzung von 100mA zur Verfügung (bis zu 500mA auf Anforderung). Dies gilt jedoch nur, wenn sich der USB-Port nicht im Suspend-Mode befindet. Ist dies der Fall, muss der Strom unter 2,5mA liegen. Dies kann nur erreicht werden, indem der FT232RL den Strom für die Fernbedienung dann abschaltet, was mit einem MOSFET realisiert wird, der vom Pin CBUS0 gesteuert wird. Die CBUS-Pins können über eine Konfigurationssoftware programmiert werden. In meinem Fall ist CBUS0 mit der Funktion belegt, nur dann auf HIGH zu gehen, wenn der USB-Port nicht im Suspend-Mode ist. Andere Funktionen sind zum Beispiel Ausgänge für LEDs, die Empfang oder Senden von Daten anzeigen oder verschiedene Taktausgänge. Wie man sieht werden sämtliche Handshake-Leitungen von mir nicht benutzt, sondern lediglich TX und RX. Eine weitere Problemstellung waren die Ausgangspegel des FT232RL, die normalerweise auf 5V-Niveau sind. Diese würden jedoch die Eingänge des ATmega644 zerstören, weil die Versorgungsspannung nur 3,3V beträgt. Für dieses Problem hat der FT232RL den VCCIO-Pin, mit dem man den Signalpegel festlegen kann. Praktischerweise liefert der Baustein gleich eine 3,3V-Referenz (3V3OUT) mit, sodass diese beiden Pins lediglich noch miteinander verbunden werden mussten.

An die Stiftleiste JP2 werden die Messingfederstifte angeschlossen, die die Verbindung zur Fernbedienung herstellen. C6 dient der Stabilisierung der Ausgangsspannung und an LED1 ist ersichtlich, ob eine Spannung am Ausgang anliegt. Der Reihenwiderstand R1 ist allerdings so dimensioniert, dass die LED den maximalen Strom zieht und somit sehr hell ist, was in der Praxis wohl noch einmal angepasst werden muss.

c. PC-Software

Zunächst war es mein Plan die einzelnen Befehle, Fernbedienungen und Makros (Zusammenschluss mehrerer Befehle) bequem am PC verwalten zu können. Die dafür benötigten Daten sollten im EEPROM (24LC64) abgelegt werden. Deshalb schrieb ich eine PC-Software, die mit der Fernbedienung über einen COM-Port kommuniziert. Der ATmega644 diente in diesem Modus, in dem die Fernbedienung mit der PC-Software verbunden war, lediglich der Vermittlung zwischen PC und EEPROM. War die Fernbedienung dann wieder im normalen Betrieb sollte sie die Daten direkt aus dem EEPROM abrufen und verarbeiten. Diese Technik erwies sich jedoch als nicht sehr praktikabel, denn der Speicher des ATmega644 hätte auf einmal das komplette EEPROM aufnehmen müssen um Menüs darzustellen, in denen die Befehle aufgelistet sind. Dafür ist einerseits nicht genügend SRAM da und andererseits stellte sich heraus, dass das ständige Lesen aus dem EEPROM die Fernbedienung ausbremste. Der Mehrwert jener Lösung war auch relativ gering, denn, wenn man etwas an den Befehlen oder Makros ändern wollte, so kann man dies auch machen, indem man schnell den Source-Code verändert, neu kompiliert und dank des Bootloaders bequem über einen COM-Port überträgt. Den Plan habe ich also wieder verworfen, dass nun das EEPROM in der Fernbedienung keine Funktion mehr hat. Ich habe das EEPROM jedoch nicht ausgelötet, weil der Stromverbrauch mit unter 1µA absolut zu vernachlässigen ist, zumal das EEPROM in der Zukunft vielleicht andere Aufgaben übernehmen wird.

Dennoch möchte ich einige Worte zur PC-Software sagen. Sie wurde in der objektorientierten Hochsprache C++, mit Hilfe der kostenlosen IDE Microsoft Visual C++ Express Edition geschrieben. Als Framework wurde das ebenfalls kostenlose wxWidgets verwendet, das plattformunabhängig ist und umfassende Funktionen bietet. Nachfolgend ist ein Screenshot des Programms zu sehen.

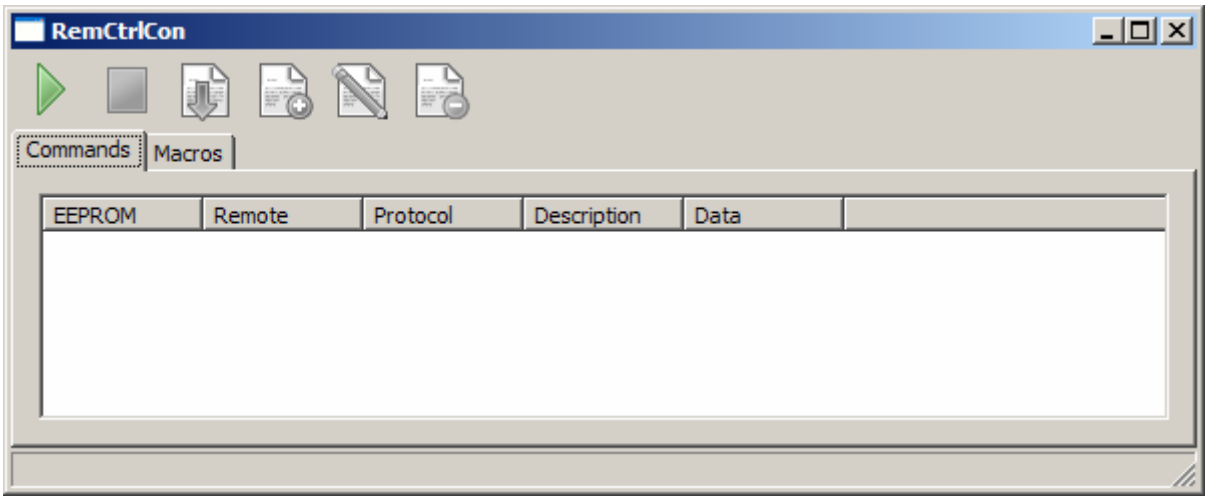

Abb. 9: Die später verworfene PC-Software, die eine Verbindung zu der Fernbedienung aufbaut

Die Funktionsweise des Programms ist eigentlich recht simpel; zunächst wird die Verbindung hergestellt, dazu wird als erstes der entsprechende COM-Port geöffnet und ein Kommando (`0x01`) geschickt. Der ATmega644 schickt ein Acknowledge (`0x06`) und wird in den PC-Modus versetzt. Damit „hört" er nur noch auf Kommandos des PCs. Als nächster Schritt schickt das PC-Programm den Befehl (`0x11`) das komplette EEPROM auszulesen. Nach 2-3 Sekunden hat das PC-Programm die Daten kann diese je nach Wunsch des Anwenders bearbeiten. Nach jedem Bearbeiten schickt das PC-Programm den geänderten Datensatz zurück an den ATmega644, der den alten Datensatz überschreibt. Das ist möglich, weil jedes Byte im EEPROM eine spezifische Adresse hat. So umgeht man nach jeder Änderung das komplette Neubeschreiben des EEPROMs, was Zeit spart und das EEPROM nicht unnötig belastet, denn die Schreibzyklen einer jeden Speicherzelle sind zwar groß (> 1 Mio.) aber endlich.

d. Software

Die komplette Software für den ATmega644 ist in C mit Hilfe der kostenlosen IDE WinAVR in der Version 20080610 geschrieben und kompiliert.

Diesen Abschnitt werde ich Aufgrund des großen Umfangs in Unterabschnitte gliedern. Diese Abschnitte richten sich mehr oder weniger nach den Untermodulen in der Fernbedienung. Ich werde bei keinem der Module den kompletten Source-Code in diesem Dokument zeigen, sondern immer nur exemplarische Ausschnitte. Der komplette Source-Code befindet sich im Anhang.

Bootloader

Wie schon erwähnt wird der ATmega644 über ISP mit einem speziellen ISP-Adapter programmiert. Nun möchte man aber beispielsweise einem Kunden ermöglichen das erworbene Gerät auch ohne spezielles Programmiergerät zu aktualisieren. Oder man möchte nicht immer wie in meinem Fall das Gerät aufschrauben um an die ISP-Buchse zu kommen,

sondern das Gerät bequem über RS232 oder USB mittels UART (bei der Fernbedienung herausgeführt) neu programmieren. Dafür bietet sich ein Bootloader an. Der Bootloader ist ein kleines Programm (< 512 Bytes), dass nach jedem Reset – egal ob durch Hard- oder Software ausgelöst – ausgeführt wird. Dieses Programm schaut, ob es über den UART Daten empfängt, die von einem speziellen PC-Programm gesendet werden. Wenn dem so ist, versetzt sich der Controller in einen Programmiermodus und bespielt sich selbst mit den über UART empfangenen Daten. Wenn nach einem Reset keine Daten empfangen werden können, wird ganz normal das Hauptprogramm aufgerufen. Dieses Prozedere verlängert das Hochfahren um eine Drittelsekunde. Die Geschwindigkeit des Programmierens ist stark von der verwendeten Baudrate ab. Bei 115200 Baud dauert es 8,2 Sekunden einen kompletten ATmega644 zu bespielen. Natürlich belegt der Bootloader Programmspeicher, dieser ist jedoch in Anbetracht des Gesamtspeichers von 65.536 Bytes zu vernachlässigen.

Der Bootloader selbst wurde in Assembler verfasst, der mit dem AVR Studio, das von Atmel zur Verfügung gestellt wird, kompiliert werden kann. Das Kompilat wird einmalig über den ISP-Adapter auf den ATmega644 programmiert. Der Code wurde von Peter Danneger auf mikrocontroller.net veröffentlicht (http://www.mikrocontroller.net/topic/73196) und wurde von mir nur leicht angepasst.

Hauptfunktion

Die Hauptfunktion *main()*, die sich in der Datei *main.c* wird nach einem Reset direkt (nachdem der Bootloader durchgelaufen ist) aufgerufen. Sie ist bewusst sehr übersichtlich gehalten.

```
1  #include <avr/interrupt.h>
2  #include <avr/wdt.h>
3  #include "dogm.h"
4  #include "i2cmaster.h"
5  #include "input.h"
6  #include "ir.h"
7  #include "makro_timer.h"
8  #include "pcf8583.h"
9  #include "tinykon.h"
10
11 int main(void){
12    // Watchdogtimer auf ca. 8 Sekunden setzen
13    wdt_enable(WDTO_8S);
14
15    // Interrupts global aktivieren
16    sei();
17
18    // Initialisierungsfunktionen
19    input_init();
20    dogm_init(40);
21    IRInit();
22    i2c_init();
23    pcf8583_init();
24    timer_init();
25
26    // Rekursive Hauptschleife nach menu() ausgelagert
27    start_menu();
28
29    return 0;
30 }
```

Listing 2: Quelle von main.c

In den Zeilen bis einschließlich 9 werden zunächst die sämtlichen erforderlichen Headerdateien eingebunden. Dann wird in Zeile 11 die Hauptfunktion definiert und deklariert. Sie muss als Rückgabewert `0` haben, der vom Typ *int* sein muss. Sie nimmt keine Parameter an. Als erstes wird dann der Watchdogtimer auf 8 Sekunden gesetzt. Er muss nicht erst aktiviert werden, weil dies bereits der Bootloader tut. In Zeile 16 werden dann noch die Interrupts aktiviert und sämtliche Initialisierungsfunktionen aufgerufen. Schlussendlich wird dann das Menüsystem gestarten, indem sich auch die Endlosschleife befindet, die das Programm nie Zeile 29 und damit das *return* erreichen lässt.

Menüsystem

Das Menüsystem stellte mich vor eine der größten Aufgaben, weil hier sehr große Datenmengen auf begrenzten Arbeitsspeicher treffen. Besonders die Texte schlucken viel Speicherplatz. Deshalb durften wenigstens die Texte nicht im SRAM sondern mussten im ROM, also dem Programmspeicher abgelegt werden. Der Zugriff hierauf erfolgt über Zeiger und spezielle Zugriffsfunktionen. Als Grundmodell verwendete ich Tinykon (http://www.mikrocontroller.net/articles/Tinykon), welches noch stark angepasst und erweitert werden musste. Nachfolgend ist die Menüstrukturarray des Hauptmenüs zu sehen:

```
1  menu_t main_menu[] = {
2    {&main_menu[3], &main_menu[1], sende_makro_menu, menu_texts[1], 0,{submenu,curup,curdwn}},
3    {&main_menu[0], &main_menu[2], sende_befehl_menu, menu_texts[2], 0, {submenu,curup,curdwn}},
4    {&main_menu[1], &main_menu[3], timer_menu, menu_texts[3], 0, {submenu,curup,curdwn}},
5    {&main_menu[2], &main_menu[0], einstellungen_menu, menu_texts[4], 0, {submenu,curup,curdwn}}
6  };
```

Listing 3: Menüstrukturarray des Hauptmenüs

Die Zeilen 2 bis 5 beschreiben je ein Menüelement. Der erste Eintrag eines jeden Menüelements beschreibt das Menüelement vor und der zweite Eintrag das Menüelement nach dem Menüelement. Der dritte Eintrag ist der Name des Submenüs, das beim Wählen des Menüelements aufgerufen wird. Der Text wird mit dem vierten Eintrag beschrieben, der fünfte kann für zusätzliche Daten verwendet werden. Der letzte Eintrag beinhaltet drei Untereinträge, die je die entsprechende Funktion für jede Taste repräsentieren.

Eingabe

Als Eingabe dient der Drehimpulsgeber mit Taster. Diese werden über einen Timerinterrupt, der jede Millisekunde auslöst, also eine Frequenz von 1kHz besitzt, ausgewertet. In der der Interruptroutine wird geschaut, wie sich die Eingangspins von Drehimpulsgeber und Taster verändert haben. Über die Funktionen *input_get_key()* und *input_read_encoder()* wird dann die Differenz zurückgegeben. Der Timerinterrupt übernimmt außerdem noch den Watchdogreset und das automatische Abschalten der Hintergrundbeleuchtung, das zusammen mit der manuell regulierbaren Helligkeit das Messen der Umgebungshelligkeit mit dem Fotowiderstand überflüssig macht, was ohnehin nicht sehr praktikabel bei einer Fernbedienung wäre. Als weitere Aufgabe jener Routine wäre das automatische Überführen in einen Schlafmodus, in dem der Controller weniger Strom verbraucht, denkbar.

Ausgabe

Die Ausgabe erfolgt über das dreizeilige 16-Zeichen-Display, dass über SPI angesteuert wird. Da ein SPI-Interface schon hardwaremäßig im ATmega644 integriert ist, ist der Programmieraufwand relativ gering. Die Funktion zum Übertragen eines Kommandos ist in Listing 4 zu sehen. Zum Initialisieren müssen dann nur noch verschiedene Kommandos übertragen werden. Das genaue Prozedere ist im Datenblatt des LCD-Controllers ST7036 zu entnehmen und leicht umzusetzen.

```
1  void dogm_cmd(char cmd){
2     DOGM_PORT &= ~(1<<DOGM_RS);
3     DOGM_PORT &= ~(1<<DOGM_CS);
4
5     SPDR = cmd;
6
7     while(!(SPSR & (1<<SPIF)));
8
9     DOGM_PORT |= (1<<DOGM_CS);
10
11    _delay_us(27);
12 }
```

Listing 4: Funktion zum Übertragen eines Kommandos an das Display über SPI

Infrarot-Signalübertragung

Das Kernstück der Fernbedienung beinhaltet drei Übertragungsfunktionen: *SIRC()*, *NEC()* und *JAPAN()*. Die Modulationsfrequenz wird mit dem Hardware-PWM erzeugt, was den Programmieraufwand massiv vereinfacht. Diese PWM muss lediglich an- und ausgeschaltet werden. Das geschieht mit den Makros *IR_LED_ON()* und *IR_LED_OFF()*. Nachfolgend ist exemplarisch die SIRC-Funktion aufgeführt.

```
1  void SIRC(uint8_t data){
2     //Modulationsfrequenz: 40kHz
3     OCR2A = 99;
4
5     uint16_t sequenz = data | ((uint16_t) 1 << 7);
6
7     for(uint8_t i=0; i<2; i++){
8        uint16_t temp_sequenz = sequenz;
9
10       //Start
11       IR_LED_ON(); _delay_us(2400); IR_LED_OFF(); _delay_us(600);
12
13       //Datensequenz
14       for(uint8_t j=0; j<12; j++){
15          IR_LED_ON();
16          _delay_us(600);
17
18          if(temp_sequenz & 1)
19             _delay_us(600);
20
21          IR_LED_OFF();
22          _delay_us(600);
23
24          temp_sequenz >>= 1;
25       }
26
27       //Stop
28       _delay_ms(28);
29    }
30 }
```

Listing 5: Funktion zum Übertragen eines IR-Codes im SIRC-Protokoll

Die Schleife in Zeile 7 veranlasst, dass jeder Code zweimal übertragen wird. Wird dies nicht gemacht, reagiert der Fernseher nicht auf den Befehl. Dann folgt zuerst die Startsequenz. In Zeile 14 ist die Schleife zu sehen, die die Datensequenz Stück für Stück durchläuft. Je nachdem, ob eine `1` oder eine `0` übertragen werden soll, wird die Pause länger oder kürzer.

PCF8583

Die PCF8583-Real-Time-Clock wird über I²C mit der Adresse `0xA1` zum Lesen bzw. `0xA0` zum Schreiben. Es wurden die I²C-Routinen von Peter Fleury (http://www.jump.to/fleury) verwendet, die wiederum auf dem Hardware I²C basieren. Die Zeit wird mit der Funktion *pcf8583_set_time()* gesetzt. Da der Baustein dann automatisch weiterzählt, kann die Zeit dann jederzeit wieder abgerufen werden. Zusätzlich kann ein Alarm einprogrammiert werden, der dann, wenn Zeit und einprogrammierter Alarm übereinstimmen, einen Interrupt im Atmega644 auslöst. Dieser veranlasst dann das Senden des gewählten Makro (mehr dazu später).

```
1  void pcf8583_set_time(DATETIME_T *DT){
2     i2c_start(PCF8583_WR_ADR);
3     i2c_write(PCF8583_CTRL_STATUS_REG);
4     i2c_write(PCF8583_STOP_COUNTING | PCF8583_MASK);
5
6     //Time
7     i2c_write(0x00);  //1/100 Seconds
8     i2c_write(BIN2BCD(DT->sec));
9     i2c_write(BIN2BCD(DT->min));
10    i2c_write(BIN2BCD(DT->hour));
11
12    //Date
13    i2c_write(BIN2BCD(DT->day) | (DT->year << 6));
14    i2c_write(BIN2BCD(DT->mon) | (DT->wday << 5));
15    i2c_stop();
16
17    pcf8583_write_byte(PCF8583_YEAR_REG, DT->year);
18
19    pcf8583_init();        //Enable counting
20 }
```

Listing 6: Funktion zum Setzen der Zeit und des Datums in der PCF8583 über I²C

Makros & Timer

Makros sind ein Zusammenschluss von mehreren Befehlen. Sie sind mit Strukturarrays realisiert, wobei jedes Arrayelement einen Befehl repräsentiert. Für jeden Befehl kann man Optionen angeben: Gerät, Befehl, Wiederholungen, Pause nach jeder Wiederholung und Pause nach den Wiederholungen. Jedem Makro wird dann noch ein Name gegeben. Sämtliche Informationen werden im ROM gespeichert. Die Typdefinitionen von Makros und der Makrozuordnungstabelle sind in Listing 7 dargestellt.

```
1  typedef struct{
2     uint8_t device;
3     uint16_t data;
4     uint8_t rpt;
5     uint16_t rpt_delay;
6     uint16_t delay;
7  } makro_t;
8
9  typedef struct{
```

```
   makro_t *makro;
   uint8_t cmds;
   char label[16];
} makrotable_t;
```

Listing 7: Typdefinitionen von Makros und Makrozuordnungstabelle

Die Makros und die Makrozuordnungstabelle sehen dann so aus wie in Listing 8 gezeigt.

```
makro_t PROGMEM tv_makro[] = {
   { 4,  2, 1, 0, 3000 },  // Receiver an
   { 4, 73, 1, 0, 0 },     // Quelle: V.AUX
   { 1,  1, 1, 0, 0 },     // TV an
   { 2, 31, 1, 0, 0 }      // DVB-T an
};

makro_t PROGMEM radio_makro[] = {
   { 4,  2, 1, 0, 3000 },  // Receiver an
   { 4, 49, 1, 0,  0 }     // Quelle: Tuner
};

makro_t PROGMEM dvd_makro[] = {
   { 4,   2, 1, 0, 3000 }, // Receiver an
   { 4,  64, 1, 0,  0 },   // Quelle: DVD
   { 1,   1, 1, 0,  0 },   // TV an
   { 3, 188, 1, 0,  0 }    // DVD an
};

makro_t PROGMEM cd_makro[] = {
   { 4,   2, 1, 0, 3000 }, // Receiver an
   { 4,  64, 1, 0,  0 },   // Quelle: DVD
   { 3, 188, 1, 0,  0 }    // DVD an
};

makro_t PROGMEM alles_aus_makro[] = {
   { 4,  3, 1, 0, 0 },  // Receiver aus
   { 1, 21, 1, 0, 0 },  // TV aus
   { 2, 31, 1, 0, 0 }   // DVB-T aus
};

makrotable_t PROGMEM makro_zuordnung[MAKROANZAHL] = {
   { tv_makro, 4, "TV" },              // 0
   { radio_makro, 2, "Radio" },        // 1
   { dvd_makro, 4, "DVD" },            // 2
   { cd_makro, 3, "CD" },              // 3
   { alles_aus_makro, 3, "Alles aus" }// 4
};
```

Listing 8: Makros und Makrozuordnungstabelle

Die Timerfunktion baut auf den Makros auf. Es gibt zwei Timer; den Weck- und den Schlaf-Timer und jedem ist ein Makro zugeordnet, dass der Benutzer auswählen kann. Auch kann er jeden Timer einzeln ein- und ausschalten und eine Alarmzeit einstellen. Wenn ein Timer auslöst, wird das eingestellte Makro ausgeführt und die Funktion *sort_timer()* ausgeführt. Diese sortiert die Timer neu und programmiert den nächsten Timer in die PCF8583. Diese Funktion ist notwendig, weil die PCF8583 nur ein Alarmzeitpunkt aufnehmen kann.

Andere Funktionen

Es gibt auch noch weitere Funktionen. So ist das Senden von einzelnen Befehlen möglich. Die Befehle sind nach Gerät sortiert. Praktisch ist dies besonders in Hinblick auf die Befehle, die weder am Gerät noch über die mitgelieferte Fernbedienung ausführbar sind. Auch die Makros können nicht nur über einen Timer ausgeführt werden, sondern können ebenfalls manuell angewählt werden.

Im Untermenü *Einstellungen* können die Stärke der Hintergrundbeleuchtung sowie das Datum und die Uhrzeit eingestellt werden. Zusätzlich informieren drei Seiten über den Systemstatus. Dort sind Informationen für Akkuspannung, Ladestatus, aktuelle Zeit und Datum sowie der nächste Alarm verfügbar. Die Batteriespannung wird wie bereits erwähnt mit einem Spannungsteiler an einem ADC-Pin des ATmega644 gemessen. Es handelt sich um einen 10-Bit-ADC und als Referenzspannung dienen einfach die von dem Festspannungsregler erzeugten 3,3V. Diese Spannung kann bis zu 60mV abweichen. Die Messung ist also nicht absolut genau, aber ich wollte ja auch lediglich eine ungefähre Information darüber, wie „voll“ der Akku noch ist.

5. Fazit und Ausblick

Nach diesem Projekt kann ich das Fazit ziehen sehr viel über Elektronik, Mikrocontroller, Bussysteme und besonders C und C++ gelernt zu haben. Ich habe mein Ziel erreicht, eine funktionierende Fernbedienung mit den vorgenommenen Kernfunktionen zu bauen. Ein Ersatz für die bestehenden Fernbedienungen wird sie jedoch aller Wahrscheinlichkeit nach nicht werden, weil der Bedienkomfort aufgrund der begrenzten Tastenanzahl im täglichen Einsatz auf der Strecke bleibt. Dennoch ist der Einsatz als Wecker oder Schlaftimer absolut vorstellbar, auch wenn ich noch längere Zeit einen zweiten Wecker stellen werde bis ich meinem eigenen Produkt traue.

Die Fernbedienung entspricht zwar jetzt meinen Vorstellungen, jedoch wird es in Zukunft sicher noch Modifikationen, die die Software betreffen, geben, so werde ich sicher noch das eine oder andere Makro erstellen oder gar ganz neue Funktionen hinzufügen. Dies ist dank dem Bootloader in Verbindung mit der USB-Schnittstelle schnell und einfach machbar - sogar ohne das Gerät zu öffnen.

6. Herstellen von Leiterplatten

Im Laufe der Entwicklung der Fernbedienung mussten natürlich auch ein paar Leiterplatten hergestellt werden. Da ich auch SMD-Bauteile mit sehr kleinen Pinabständen verwendet habe, waren die Ansprüche an die Qualität der Leiterplatten recht groß. Man kann die Leiterplatten zwar auch professionell fertigen lassen, was einige Vorteile bietet (Lötstopmaske, chem. Durchkontaktierungen) aber auch recht teuer ist. Deswegen stellte ich

die Leiterplatten selbst her. Diesen Vorgang möchte ich hier kurz anhand eines Beispiels beschreiben.

Zunächst muss man die Leiterplatte entwerfen. Dafür gibt es viel Software im Internet, die von kostenlos bis sehr teuer reicht. Ich benutze ein recht günstiges Programm namens EAGLE (www.cadsoft.de), das zugleich das im Hobbybereich am weitesten verbreitete ist. Erster Schritt beim Entwerfen der Leiterplatte ist das Zeichnen des Schalplans. Es existieren umfassende Bauteilbibliotheken, die die üblichsten Bauteile beinhalten. Das Zeichen des Schaltplans ist in Abbildung 9 zu sehen.

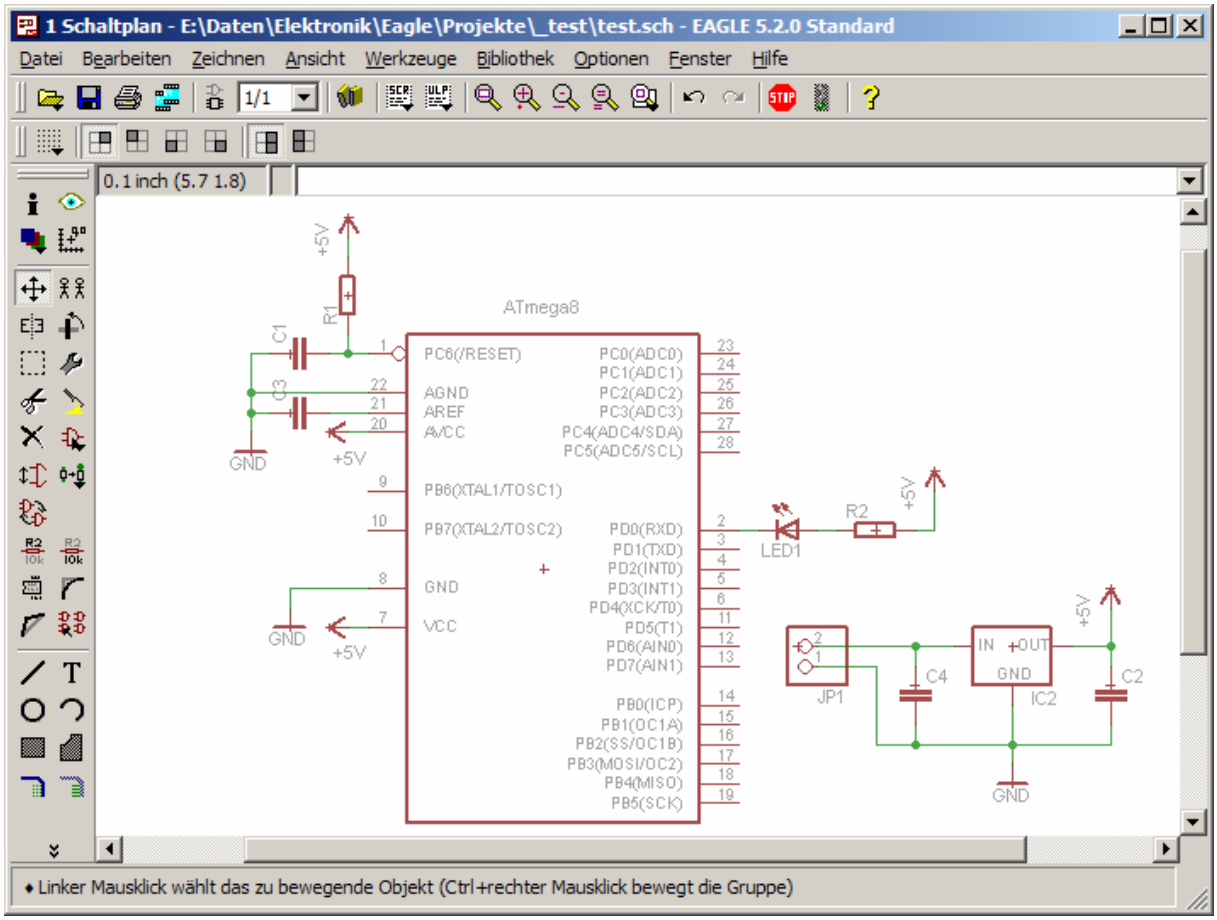

Abb. 10: Zeichnen eines Schaltplans mit EAGLE

Anschließend überführt man den Schaltplan auf die Leiterplatte. Die Bauteile können angeordnet werden und gelbe Linien zeigen dem Anwender, wo später Leiterbahnen gezogen werden müssen. Dann werden die Leiterbahnen von Hand gezeichnet und das Ganze exportiert.

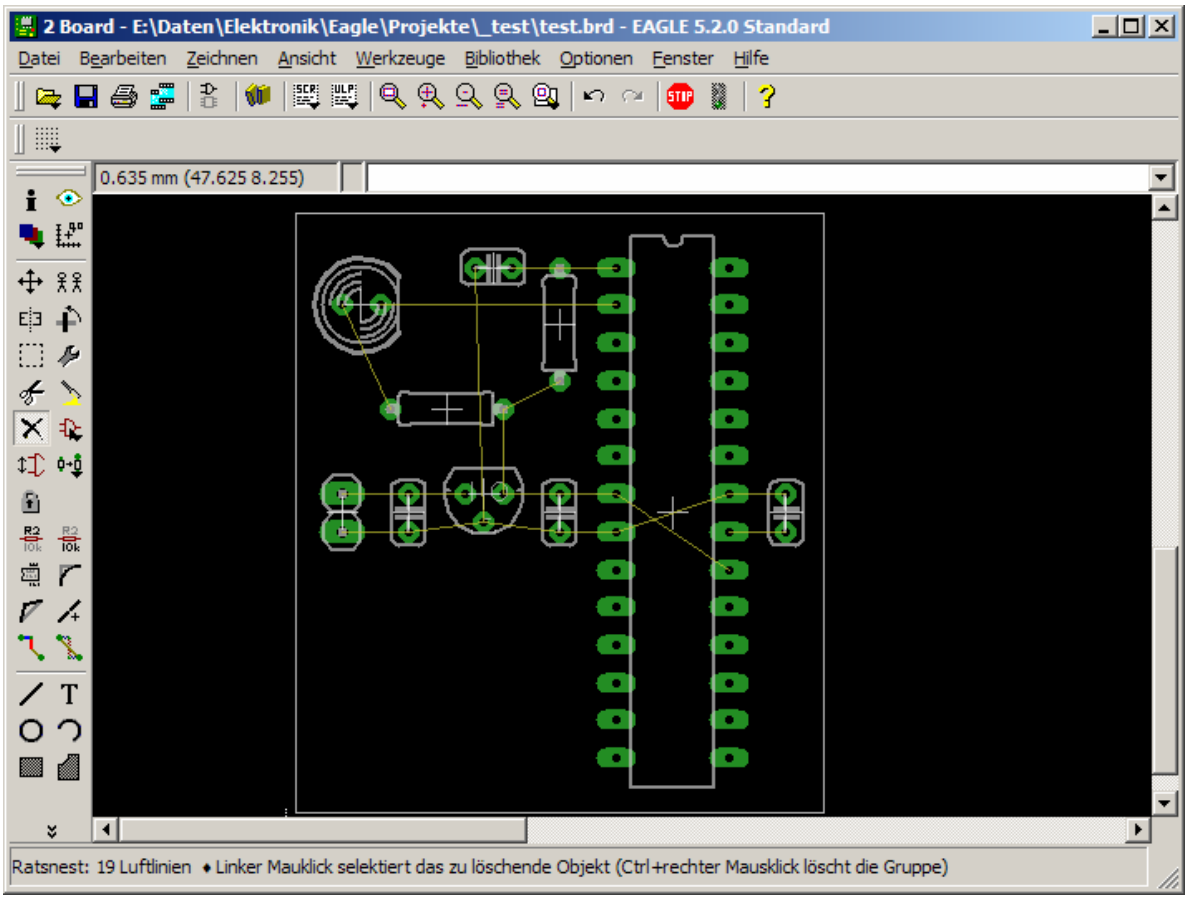

Abb. 11: Leiterplattenlayout mit gelben Signallinien

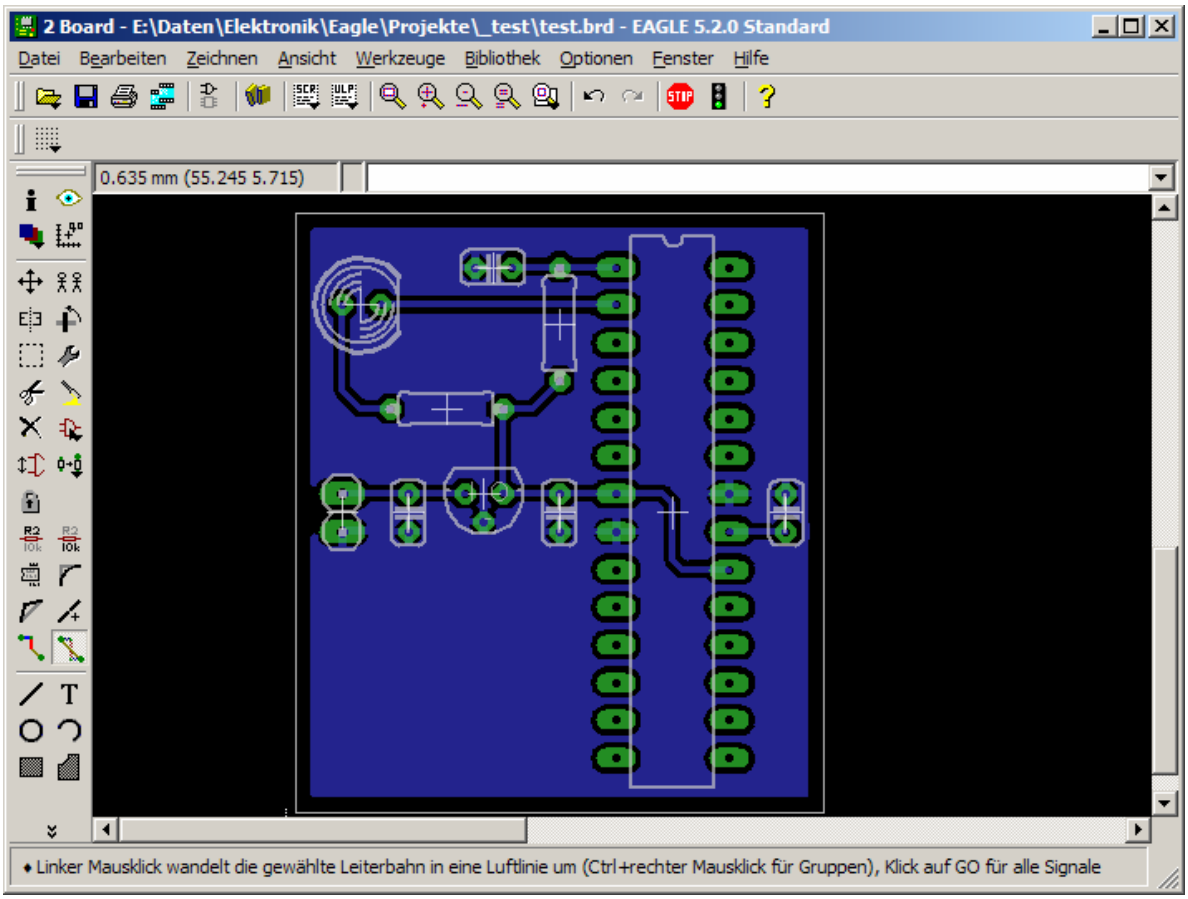

Abb. 12: Fertiges Leiterplattenlayout

Nun gilt es einen Belichtungsfilm herzustellen. Dieser muss möglichst lichtdicht sein, weshalb selbstbedruckte Overheadfolien für qualitativ hochwertige Leiterplatten ausscheiden. Ich lasse meine Filme seit einiger Zeit für günstiges Geld professionell fertigen (www.cadgrafik-bauriedl.de/leiterplattenfilme.htm). Resultat ist eine absolute Kantenschärfe und Lichtdichtigkeit.

Nun wird die Folie auf eine mit Fotolack beschichtete Kupferplatine gelegt und mit einer Glasscheibe angedrückt. Belichtet wird mit UV-Licht. Die Dauer ist bei solchen Qualitätsbelichtungsfilmen relativ unkritisch und liegt bei mir immer im Bereich 5 Minuten. Nach dem Belichten wird in Natronlauge entwickelt, wobei die Stellen, wo der Lack belichtet wurde, von der Platine abgelöst werden und das Kupfer frei liegt. Beim anschließenden Ätzvorgang mit Ammoniumperoxidisulfat werden dann nur diese Stellen weggeätzt. Die Dauer des Ätzvorgangs hängt maßgeblich von der Temperatur ab und ist bei circa 50°C minimal.

Nach dem Ätzen kann die Platine noch verzinnt werden, das hat die Vorteile der besseren Löteigenschaften und der Korrosionsbeständigkeit. Dies wird im professionellen Bereich stets chemisch gemacht, was sich in meinem Fall jedoch nicht lohnt, da die Verzinnungsbäder meist nur eine begrenzte Haltbarkeit haben und sich bei meinen geringen Stückzahlen nicht auszahlen würde. Deshalb nehme ich Fittingslot, das eigentlich zum Verlöten von Rohren gedacht ist. Dieses pinsel ich deckend auf die Platine und erhitze diese anschließend mit einem Heißluftfön. Das Zinn schmilzt und verbindet sich mit dem Kupfer der Platine.

Nach dem Bohren kann die Platine dann bestückt werden und ist einsatzbereit.

7. Glossar

7805	Ein äußerst verbreiteter 5V-Festspannungsregler. Er liefert 1A, ist jedoch in zahlreichen Varianten verfügbar.
Acknowledge	Ein Acknowledge-Signal (für engl. Acknowledgement = Bestätigung, Zustimmung, Quittierung) ist ein Signal, das bei einer Datenübertragung verwendet wird, um den Erhalt oder die Verarbeitung von Daten oder Befehlen zu bestätigen.
ADC	Ein Analog-Digital-Umsetzer, englisch ADC für Analog-to-Digital-Converter setzt analoge Eingangssignale in digitale Daten bzw. einen Datenstrom um, die dann weiterverarbeitet oder gespeichert werden können.
Array	Ein Array (deutsch: Feld) bezeichnet in der Informatik einen Datentyp. Mit Hilfe eines Arrays können Daten eines einheitlichen Typs geordnet so im Speicher eines Computers abgelegt werden, dass ein Zugriff auf die Daten über einen Index möglich ist.
ATmega	ATmega bildet eine Gruppe der Atmel AVR 8-Bit-Mikrocontroller-Familie des US-amerikanischen Herstellers Atmel. Die Controller dieser Familie sind wegen ihres einfachen Aufbaus und ihrer leichten Programmierbarkeit bei Hobby-Anwendern weit verbreitet.
Atmel	Die Atmel Corporation ist ein US-amerikanischer Hersteller von Integrierten Schaltungen mit Sitz in San Jose, Kalifornien.
Bit	Der Begriff Bit (binary digit) wird in der Informatik als Bezeichnung für eine Binärziffer (üblicherweise „0“ und „1“) verwendet.
Bootloader	Ein Bootloader ist eine spezielle Software, die gewöhnlich im ROM bzw. Flash sitzt und nach einem Reset zuerst ausgeführt wird. Der Bootloader startet dann das Hauptprogramm.
Bus	Ein Bus ist in der Datenverarbeitung ein Leitungssystem mit zugehörigen Steuerungskomponenten, das zum Austausch von Daten und/oder Energie zwischen Hardware-Komponenten dient.
Byte	Das Byte ist ein Mengen-Begriff aus der Digitaltechnik und Informatik, der für eine Zusammenstellung von 8 Bits steht.
C	C ist eine imperative Programmiersprache, die der Informatiker Dennis Ritchie in den frühen 1970er Jahren an den Bell Laboratories für das Betriebssystem Unix entwickelte. Seitdem ist sie auf vielen Computer-Systemen verbreitet.
COM	Mit Communication Equipment („COM-Schnittstelle“) wird die serielle Schnittstelle EIA-232 (RS232) in der PC-Technik genannt.

Compiler	Der Compiler setzt den Quelltext in eine für das Zielsystem verständliche Maschinensprache um.
C++	C++ ist eine höhere Programmiersprache. Sie wurde in den 1980er Jahren von Bjarne Stroustrup als Erweiterung von C entwickelt. C++ wurde als Mehrzwecksprache konzipiert und unterstützt mehrere Programmierparadigmen, wie die objektorientierte, generische und prozedurale Programmierung.
Drehimpulsgeber	Drehimpulsgeber erzeugen bei Drehung der Achse an den zwei Datenleitungen am Ausgang ein sogenanntes Gray-Code-Signal. Der Vorteil dieser Codierung ist, dass eine weitere Fehlerkorrektur in der Regel überflüssig ist.
EEPROM	Ein EEPROM (Electrically Erasable Programmable Read-Only Memory) ist ein nichtflüchtiger, elektronischer Speicherbaustein.
Flanke	Eine Flanke ist der Übergang vom HIGH- zum LOW-Pegel (fallende Flanke) oder umgekehrt (steigende Flanke).
Flashspeicher	Flashspeicher sind digitale Speicherchips. Im Gegensatz zu „gewöhnlichem“ EEPROM-Speicher lassen sich bei Flash-EEPROMs Bytes nicht einzeln löschen. Bei Mikrocontrollern befindet sich der Programmcode in einem Flashspeicher.
Framework	Ein Framework ist ein Programmiergerüst, welches in der Softwaretechnik, insbesondere im Rahmen der objektorientierten Softwareentwicklung sowie bei komponentenbasierten Entwicklungsansätzen, verwendet wird.
Handshake-Leitung	Eine Handshake-Leitung übernimmt neben den Datenleitungen in einer Schnittstelle Kontrollfunktionen.
Headerdatei	Eine Headerdatei ist in der Programmierung, insbesondere bei den Programmiersprachen C und C++, eine Textdatei, die Deklarationen und andere Bestandteile des Quelltextes enthält.
I²C	I²C (Inter-Integrated Circuit) ist ein von Philips entwickelter serieller Datenbus, der hauptsächlich geräteintern für die Kommunikation zwischen verschiedenen Schaltungsteilen genutzt wird.
IDE	IDE (Integrated Development Environment) bezeichnet eine integrierte Entwicklungsumgebung in der Programmierung.
Impulsbelastbarkeit	Der Strom mit dem ein Bauteil für eine (sehr) kurze Zeit belastet werden kann, wird als Impulsbelastbarkeit bezeichnet.
Interrupt	In der Informatik versteht man unter Interrupt die kurzfristige Unterbrechung eines Programms, um eine andere, meist kurze, aber zeitkritische Verarbeitung durchzuführen.

ISP	Die In-System Programmierung (ISP) ermöglicht das Programmieren einer Logischen Schaltung direkt im Einsatzsystem.
LDR	Ein Fotowiderstand (engl.: LDR - Light Dependent Resistor) ist ein lichtabhängiger Widerstand aus einer amorphen Halbleiter-Schicht.
LED	Eine Leuchtdiode (engl.: LED - Light Emitting Diode) ist ein Halbleiter-Bauelement. Fließt durch die Diode Strom in Durchlassrichtung, so strahlt sie Licht, Infrarotstrahlung oder auch Ultraviolettstrahlung ab.
Lötstopmaske	Die Lötstopmaske verhindert, dass Leiterbahnen Lötzinn annehmen, auf denen nicht gelötet werden soll.
LPT	LPT (line printing terminal) bezeichnet den Druckerport an einem PC. Es handelt sich dabei um eine parallele Schnittstelle.
Makro	Unter einem Makro versteht man ein Programm, das eine fest vorgegebene Folge von Befehlen enthält. Alle Anweisungen des Makros werden automatisch ausgeführt, wenn das Makro aufgerufen wird.
Modulation	Die Modulation beschreibt in der Nachrichtentechnik einen Vorgang, bei dem ein zu übertragendes Nutzsignal ein sogenanntes Trägersignal verändert (moduliert) und damit die Übertragung des Nutzsignals über das Trägersignal möglich wird.
MOSFET	Der Metall-Oxid-Halbleiter-Feldeffekttransistor (engl.: metal oxide semiconductor field-effect transistor, MOSFET) ist eine Variante der Feldeffekttransistoren mit isoliertem Gate.
Nibble	Ein Nibble ist eine Datenmenge, die 4 Bits umfasst.
Parität	Die Paritätskontrolle dient der Erkennung fehlerhaft übertragener Informationswörter.
Pegel	In der Digitaltechnik bezeichnen Logikpegel definierte Bereiche für elektrische Spannungen, die sich auf Masse beziehen. Bei den üblichen binären Signalen sind zwei Spannungsbereiche erlaubt, die HIGH-Pegel bzw. LOW-Pegel genannt werden.
Pin	Ein Pin ist der äußere Anschluss eines Integrierten Schaltkreises.
PWM	Die Pulsweitenmodulation (PWM) ist eine Modulationsart, bei der eine technische Größe (z. B. elektrischer Strom) zwischen zwei Werten wechselt.
(S)RAM	SRAM steht für „statisches RAM“ und bezeichnet einen elektronischen Speicherbaustein, der hauptsächlich als Cache eingesetzt wird. Sein Inhalt ist flüchtig.
Reset	Ein Reset ist ein Vorgang, durch den ein elektronisches System in einen definierten Anfangszustand gebracht wird.

ROM	*siehe Flashspeicher*
RS232	RS232 bezeichnet einen Standard für eine serielle Schnittstelle, die in den frühen 1960ern von einem US-amerikanischen Standardisierungskomitee eingeführt wurde.
RTC	Eine Echtzeituhr (engl.: RTC – real time clock) ist eine Uhr, die die physikalische Zeit misst. Es handelt sich meist um einen Chip, der selbstständig die Zeit weiterzählt.
Samplingrate	Mit Samplingrate (Abtastrate), wird in der Signalverarbeitung die Häufigkeit benannt, mit der ein Signal pro Zeitintervall abgetastet wird.
Screenshot	Bildschirmfoto
Source-Code	Der Source-Code ist der Quelltext eines Computerprogramms.
SMD	Der Begriff oberflächenmontierbares Bauteil (engl.: SMD - surface-mounted device) bezeichnet ein Bauteil, dessen Pins auf der Bestückungsseite verlötet werden.
Suspend-Mode	Standby-Modus des USB
SPI	Das Serial Peripheral Interface (SPI) ist ein von Motorola entwickeltes Bus-System für einen synchronen seriellen Datenbus, mit dem digitale Schaltungen nach dem Master-Slave-Prinzip miteinander verbunden werden können.
Struktur	In der Informatik ist eine Struktur ein mathematisches Objekt zur Speicherung von Daten.
Terminalprogramm	Der Begriff Terminalprogramm wird für Programme verwendet, die den PC zu einem Terminal reduzieren und z.B. über die serielle Schnittstelle Zeichen mit anderen Geräten austauschen.
Timerinterrupt	Wenn ein Interrupt durch das Überlaufen eines Zählers oder das Treffen des Zählers eines bestimmten Wertes ausgelöst wird, so nennt man ihn Timerinterrupt.
UART	Die UART-Schnittstelle dient zum Senden und Empfangen von Daten über eine Datenleitung und bildet den Standard der seriellen Schnittstellen an PCs und Mikrocontrollern.
USB	Der Universal Serial Bus (USB) ist ein serielles Bussystem zur Verbindung eines Computers mit externen Geräten.
Watchdog	Der Watchdog ist ein Hardwaretimer, der nach einer definierten Zeit einen automatischen Reset auslöst, deshalb muss er regelmäßig zurückgesetzt werden. Er verhindert Ausfälle nach Fehlfunktionen.
XOR	Logische Verknüpfung „entweder oder“; auch Modulo-2-Addition

8. Anhänge

Sämtliche Anhänge befinden sich auf der beiliegenden CD. Der besseren Übersicht halber möchte ich hier den genauen Inhalt anhand eines Verzeichnisbaumes mit Beschreibungen der einzelnen Dateien angeben.

CD

Code

Mikrocontroller

Analyse – *Quelltexte der Schaltung zum Analysieren der Fernbedienungen*

Bootloader – *Quelltexte (Assembler) des Bootloaders*

Fernbedienung – *Quelltexte der Fernbedienung*

Fernbedienung (EEPROM) – *Unfertige Quelltexte der alten Fernbedienung mit EEPROM*

PC

FBOOT – *Quelltexte des PC-Tools für den Bootloader*

RemCtrlCon – *Quelltexte des PC-Programms zur alten Fernbedienung mit EEPROM*

Debug – *Arbeitsordner von Visual C++ mit ausführbarer Datei „RemCtrlCon.exe“*

Bilder – *Bilder der Fernbedienung im JPEG-Format*

Dokumente

Datenblätter – *Sämtliche Datenblätter aller verbauten nicht Standardbauelemente*

DENON AVR4308 RC CODE.pdf – *Dokumentation über Denon IR-Befehle & -Protokolle*
Entwicklung und Realisierung eines PDA-Testers.pdf – *Beschreibung diverser IR-Protokolle*
Timings.xls – *Ergebnisse meiner Analysen der gegebenen Fernbedienungen*

Programme

aStudio4b623.exe – *Atmel AVR Studio Setup*
eagle-win-5.3.0.exe – *Setup des Schaltplan- und Leiterplattendesignprogramm EAGLE*
HTerm.exe – *Terminalprogramm*
WinAVR-20080610-install.exe – *WinAVR IDE*
wxMSW-2.8.9-Setup.exe – *WxWidgets Framework für C++*

Aufgrund der Größe befindet sich das Setup von Visual C++ nicht auf der CD. Es kann jedoch hier heruntergeladen werden: http://go.microsoft.com/?linkid=9340867

Schaltungen

Eagle – *Sämtliche Projektdateien für EAGLE*

Schaltungen – *Sämtliche Schaltpläne im PNG-Format*

Fernbedienung.pdf – *Dieses Dokument als (farbiges) PDF-Dokument*

9. Quellen

http://bjorn.rhoads.nu/hp48/remote/

http://de.wikipedia.org

http://en.wikipedia.org

http://www.boehmel.de/sircs.htm

http://www.mikrocontroller.com/de/IR-Protokolle.php

http://www.mikrocontroller.net/articles/Spezial:Alle_Seiten

http://www.mikrocontroller.net/forum/

http://www.sbprojects.com/knowledge/ir/ir.htm

sowie alle auf der CD befindlichen Dokumente und Quelltexte