Ein RC5-Code Übersetzer

Seit dem Ende des normalen terrestrischen TV's ist die Benutzerfreundlichkeit meines DVD Recorders stark reduziert. Der interne Tuner ist nicht mehr nutzbar. Nun, der Recorder ist zwar in der Lage Satreceiver über Infrarot zu steuern, genau genommen nur die Kanalwahl, was ich bisher auch verwendete um SAT zu empfangen. Um auch die verschlüsselten lokalen Sender wieder sehen zu können habe ich mir einen Satreceiver anschaffen müssen, der das kann.

Damit begann eine umständliche Prozedur, da ich alles, was ich aufnehmen wollte zwei mal, einmal auf dem Sat und dann auf dem DVD programmieren mußte.
Dieser Satempfänger kam nämlich nicht mehr im Repertoire der zu steuernden Satreceiver im DVD Recorder vor.

Zu Ostern dieses Jahres bekam ich das „Lernpaket Mikrocontroller" aus dem Franzisverlag. Als Entwicklungsingenieur in Ruhe braucht man ja was um das Gedächtnis in Trab zu halten. Noch dazu, wenn der Beruf auch das Hobby war.

In dem Paket ist ein Attiny13 Mikrokontroller von Atmel , ein Entwicklungssystem und im Begleitbuch sind viele Programmbeispiele enthalten.
Ich lernte also wieder einmal was neu dazu und nach kurzer Zeit war mir klar, dass dieses Paket viel, viel mehr kann als nur die Programme nachzuvollziehen, die da beispielhaft gelistet sind. Nötig war dazu, neben dem Buch zu dem Paket, auch die Atmel Dokumentation für den uC aus dem Internet zu holen und darin wirklich viel zu lesen, um die Möglichkeiten des uC kennen zu lernen. Wenn man also ein Blinkprogramm nachvollzieht, so ist diese uC Dokumentation unverzichtbar um die Befehle, ihren Inhalt und ihre Wirkung nachzuvollziehen.

Nebenbei: Ich schreibe diesen Artikel deshalb so ausführlich, weil ich gerne viele Leute ermuntern möchte sich mit diesem Zweig der Technologie zu befassen.
Normal sind ja viele Programme für Insider geschrieben, die den „Anfang" hinter sich haben und schneller folgen was da beschrieben ist.
Den Neuen schreckt das ab und baut Schranken auf. Ich möchte sie gar nicht erst entstehen lassen.
Z.B. „Interruptprogrammierung" was soll den das sein?
Stellen sie sich vor sie machen gerade etwas und das Telefon läutet. So was ist ein Interrupt in uC Terminologie. Sie können jetzt entscheiden ob sie den Hörer abheben, den Interrupt bedienen oder nicht. Damit ist, in uC Terminologie auch entschieden, ob sie Interrupts zulassen oder nicht (Befehle sei und cli).
So können sie Assoziationen ihrem bisherigen Erfahrungsschatz bilden.

Manche Befehle verstehen sich von selbst:

```
add  r16,r17        ; addiert zum Inhalt von r16 = Kontostand  die r17 = Rente
                    ; r16 enthält nun die Summe von beiden Summanden
```

man kann auch schreiben
add Kontostand, Rente wenn man in der .def Anweisung r16 und r17 so definiert, doch dazu später.

Zusätzlich haben sie jetzt schon gelernt, dass man einen Befehl mit einem Kommentar ergänzt. Wenn sie nur

```
add  a,b              ; Summe bilden
```

schreiben, so haben sie nach spätestens einem Monat vergessen, welche Bedeutung diese Addition hatte. Es ist dann mühsam sich wieder in das Programm einzulesen, wenn sie ggfs. etwas ändern wollen.

Ehe sie selbst etwas programmieren wollen, mit den Werkzeugen, die in dem „Paket Mikrocontroller“ enthalten sind gehen sie bitte etliche Beispiele durch nur um genügend Routine oder Neudeutsch die nötigen skills zu erwerben.

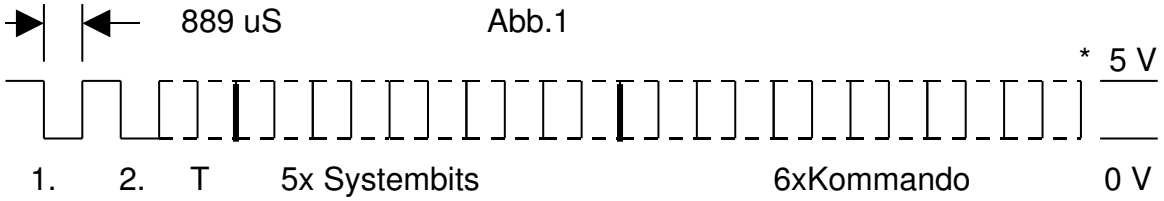
Nun zu meinem Projekt. Ich will also mit meinem DVD Recorder den neuen Satreceiver steuern, sodass der Satrececorder auf den aufzunehmenden Sender umschaltet.
Ich wollte das mit dem „RC5 Code“ machen.

Was ist denn das schon wieder? Nun eine Fernbedienung für ihr TV Gerät haben sie sicher. Diese sendet Infrarotlicht = IR aus, das den TV Empfänger entsprechend reagieren lässt. RC5 ist ein Codierungssystem von Philips wo ich gearbeitet habe und außerdem recht einfach codiert ist. Viele Hersteller verwenden diesen Code. Auch der neue Satreceiver. Und der DVD Recorder kann Philips Satreceiver steuern. Also lag es nahe RC5 zu wählen. Nebenbei gesagt gibt es neben RC5 noch viele ander Systeme wie Panasonic, JVC, Pioneer, Daewoo, Syncmaster, Kathrein, etc. nur um einige zu nennen. Ein Freund von mir hat einen Homecast Satreceiver, der einen abgewandelten Pioneercode verwendet, damit Pioneergeräte so einen IR-Befehl nicht verstehen können. Es muss ja dafür gesorgt werden, dass sich die vielen Fernbedienungen nicht in die Quere kommen. Das wäre was, wenn die Lautstärkeregelung des Verstärkers die Sender am TV umschaltet oder umgekehrt oder sonst ein anderer Blödsinn passiert..
Da wurde eine große Chance vergeben ein einheitliches System zu schaffen.

Jetzt wird es aber Zeit auf mein Projekt im Detail einzugehen.

1. Aufbau der RC5 Signals

   14 bit Code: 2 Startbits 1Togglebit 5 Systembits und 6 Commandbits



   1 bit besteht aus zwei Halbbits. 1 Halbbit dauert 889 Mikrosekunden [uS].
   Die bits können nun 0 oder 1 sein können.

   Eine „1“ = 1 Halbbit „kein Licht“ und 1 Halbbit „Licht“ mit 36 kHz ein/aus geschaltet
   Eine „0“ = 1 Halbbit „Licht“ und 1 Halbbit „kein Licht“
   Wenn sie Lust haben könnten sie schon selbst ausrechnen wie lange ein Tastenkommando dauert, ca 24,89 mS. Das Kommando wird nach 114 mS wiederholt, siehe Lautstärke und Togglebit.

Das mit 36 kHz modulierte Licht hat also eine Periodendauer von 1/36 = 27,777 uS.
Wovon wirklich nur 6,944 uS Licht scheint und dann 20,833 uS Pause ist.
Ein Halbbit „Licht“ besteht demnach aus 32 Lichtimpulsen.
Es wird eigentlich nur sehr wenig Licht ausgesendet und das erklärt, warum die Batterien von RC5 Fernbedienungen so lange halten.

1.1 Die Startbits dienen dazu um dem IR Empfänger anzukündigen, das was folgt
Die beiden Startbits sind immer 1 (wenn die 6 Komandobits ausreichen, sonst ist das 2. Startbit eine 0)

1.2 Das Togglebit dient dazu mitzuteilen, dass nun ein neues Kommando folgt.
Es wechselt mit jedem Tastendruck seinen Wert, der 0 oder 1 sein kann.
Wenn sie den TV lauter stellen wollen, lassen sie den Finger einfach so lange auf der Taste bis ihnen die Lautsrärke angenehm ist. Es folgt also ein dauernder Strom von Kommandos mit lauter lauter lauter lauter ..... mit demselben Togglebit.
Auch wenn sie den Sender umschalten ist ihr Finger wahrscheinlich so lange auf der Taste, dass statt, was sie wollen, z.B. „5“ 555555 gesendet wird.
Alle 5er werden mit dem selben Togglebit gesendet, so weiss der Empfäger was gemeint ist, nämlich 5 und nicht mehr..
Wenn sie aber auf 11 umschalten wollen, so wird die erste 1 als wahrscheinlich 111 und die zweite 1 ebenfalls 1111, aber mit dem umgekehrten Togglebit gesendet. D.h. bei jedem neuen Tastendruck schaltet das Togglebit um.

1.3 Die 5 (2^5=32) Systembits erlauben bis zu 32 verschiedene Geräte

00=TV1, 01=TV2, 05=VCR1, 15= frei, 21=Plattenspieler usw.
Alle Informationen sind im Internet vorhanden und am Ende des Berichts werde ich eine Linkliste anhängen.

1.4 Die 6 Kommandobits erlauben bis zu 64 verschiedene Tasten auf der Fernbedienung. Das war manchmal zu wenig, so hat man das 2. Startbit zu einem zusätzlichen Kommandobit verwendet, aber in meinem Projekt habe ich dies vernachlässigt, da die Zifferntasten und mehr brauche ich nicht, das nicht verwenden.

2. Hardware
Zum empfangen des IR Lichtstromes mit je 14bit / Kommando wird ein IR Empfänger z.B. TSOP1736 oder SFH506-36 verwendet. Ich habe etliche aus Elektronikschrott ausgebaut, so wie die IR Sendediode auch.

Das Ausgangssignal eines TSOP1736 ist 5V bei „kein Licht“ und 0V bei „Licht“ während der ganzen 889 uS; das modulierte Licht wird in diesem Baustein gleich demoduliert.
Siehe auch * in Abb.1
Auf dem beschriebenen Entwicklungssystem ist der TSOP1736 eingesteckt an Masse, Vcc und Port PB3 als Eingang. Die Sendediode über 390 Ohm an Port PB4 als Augang und Masse.

Ursprünglich dachte ich, dass das RC5 Übersetzungsprogramm in den kleinen Speicher des Attiny13 nicht hineinpassen würde. Aber es geht sich locker aus und auch der Bootloader der vom Entwicklungssytem geladen wird ist auch noch aktiv. Das iR-Kommando nicht nur

übersetzt, sondern auch noch zur Kontrolle an den PC gesendet und wird dort ausgegeben.
Weiters können verbesserte, d.h. debuggte Versionen leicht neu geladen werden.
Ich habe versucht sehr Code sparend zu programmieren, aber es war gar nicht nötig.

3. uC Programm

Ich habe übrigens „Assembler“ zum programmieren verwendet. Viele wenden sich mit Grauen ab und verwenden lieber eine Hochsprache wie C oder Basic.
Meine Erfahrung ist jedoch, dass sich Assembler schneller lernen lässt als speziell C.
Meine bisherigen Projekte, die zum Teil mehr als 25 Jahre zurückliegen waren für die uP'S 6502, Z80 und die PC-CPU „286er“ geschrieben.

Um ein wenig schneller mit dem Programm fertig zu werden, habe ich die Ressourcen im Internet durchstöbert, was es alles zu diesem Thema gibt. Es ist nicht wenig kann ich nur sagen: Die Application Note AVR410 von Atmel listet ein Programm zum RC5 Empfang. Hier musste ich doch eine Anzahl Änderungen vornehmen. Hauptsächlich weil die Taktfrquenz des uC bei mir nur 1,2 MHz und nicht 4 MHz wie in dem Beispiel ist. Weiters war es für einen anderen Typ uC geschrieben.

Das IR Sendeprogramm habe ich auch im Internet gefunden, das wiederum für einen PIC-uC geschrieben ist, auch hier war die Taktfrqeuenz anders.
Weiters wollte ich wissen, ob das gesendete Kommando richtig war und habe es an den, am Entwicklungssystem angeschlossenen PC gesendet.
Es könnte ja ein Fehler vorliegen und so kann ich die Ursache eingrenzen.

Das ist auch ein wichtiger Tip: normal funktioniert nichts sofort, sie müssen Fehler suchen.
Es sind überwiegend Denkfehler. Entweder man setzt was voraus, was nicht so ist, oder man hat einen uC Befehl falsch angewendet, oder ist unaufmerksam.
Oder man hat etwas am falschen Port angeschlossen. Der Möglichkeiten sind viele.
Da muss man sich durchbeissen! Das benötigt viel Zeit, nicht entmutigen lassen!

Das Programm teilt sich in folgene Blöcke:

1. Definitionen
   Das kennen sie schon ein wenig siehe weiter oben

```
.include "tn13def.inc"
;Port B
.equ   TXD    = 1    ;senden
.def S                =          R0          ; Storage for the Status Register
.def inttemp          =          R1          ; Temporary variable for ISR
.def ref1  =          R2
.def ref2  =          R3          ; Reference for timing
.def temp             =          R16         ; Temporary variable auch für WrCom
.def timerL           =          R17         ; Timing variable updated every 64 us
.def A                =          R18         ; allg. Verwendung z.B. serielle Ausgabe zum PC
.def system           =          R19         ; System data received
.def comman           =          R20         ; command data received
.def bitcnt           =          R21         ; Counter auch für WrCom
.def B                =          R22         ; Variable für lian , noli,
.def C                =          R23         ;   -"-
```

2. Das Interruptprogramm
   Es dient als Taktgeber um die Zeit eines Halbbits in Vielfachen von 64 uS zu messen. Dort kommt u.a. der Terminus „sreg“ vor. Das ist das Statusregister des tiny13 und sie finden Information in der Attiny13 Dokumentation. Die anderen Befehle könne sie auch dort nachlesen.

```
.cseg
.org 0
        rjmp Anfang
.org 0x0006     ;OCIE0A Interrupt
;*************************************************************************
;* "OCIE0A: Timer/Counter0 Output Compare Match A Interrupt Enable" handler Seite 74 ATtiny13 doc
;* The timo_compa interrupt increments the "timerL" and "initemp"
;* every 64us, bei mir 64,11666 us  (and 16,384ms, bei mir 16,426667 us.. brauch ich nicht)
;* Crystal Frequency is original 4 MHz, BEI MIR 1,2 MHz!!
;* 4 MHZ = 250 ns * 256 = 64 mikrosec. Ich habe 833ns Takt also nicht bis 256 zählen
;* sondern nur bis 833/250=76.8, geht nicht also nehme ich 77 als CTC mode Wert in OCR0A
;* Number of words:7 (5)
;*************************************************************************
;TIM0_COMPA:    ;Timer Interrupt
        in S,sreg               ; Store SREG
        inc timerL              ; Updated every 64,1us
        inc inttemp             ; wird auch alle 64,1 us incrementiert
TIM0_ende:
        out sreg,S              ; Restore SREG
        reti
```

3. Die Initialisierung von verschiedenen Registern.
   Datenrichtungen von Ports, ob Eingang oder Ausgang setzen und den internen Zähler, der den Interrupt auslöst so setzen, dass wirklich alle 64 uS ein Interrupt ausgelöst wird.

```
;*********************************************
;* Example program
;* Initializes timer, ports and interrupts.
;*********************************************

Anfang: ldi     A,18            ;Portsetzen geht nicht immediate
        out             ddrb,A          ;Datenrichtung Port 1 + 4 auf "out" Komm. mit PC +IR-Led
        ldi             A,RAMEND        ;init stackpointer
        out             SPL,A
         ldi            A,1             ;Start ohne Vorteiler, Seite 72 ATtiny13.pdf
        out             TCCR0B,A        ;setze die Bits CS02, 01 und 00 auf: 001 = kein Vorteiler
        ldi             A,77            ;OCROA TOP Wert setzen, Seite 58
        out             OCR0A,A
        ldi             A,2             ;CTC mode of TCCR0A WGM01 bit set, Seite 71
        out             TCCR0A,A        ;"clear timer on compare match" Modus
        ldi             A,4
         out            TIMSK0,A        ;Timer Interrupt OCIEOAfreigeben
            ;TIFR0,Bit 2 is set by  Counter/Timer Seite 74
        clr system
        clr comman
        sei                             ;Enable global interrupt
```

4. Die Hauptschleife
   Sie besteht aus dem Aufruf der IR Empfangsroutine incl. dem Kommandoübersetzer
   Der Ausgabe des zu sendenden Kommandos an den PC
   Die IR Senderoutine

```
main:   rcall detect            ;hole das 1. Startbit, messe Dauer und dann den Rest
        ;Totzeit vielleicht einfügen
        cli                     ;no Interupt während Ausgabe
        mov A,system            ;Systemadresse ausgeben address
        andi A,0x1F             ;die 5 Systembits maskieren
        rcall WrCOM
        clr A                   ;togglebit ausgeben via A
        bst system,5            ;Status des Togglebits nach T-Flag
        bld A,0                 ;und von dort ins Bit0 von A
        rcall WrCOM             ;ausgeben
        mov A,comman            ;Tastencode ausgeben
        andi comman,0x3F        ;6 bits maskieren
        rcall WrCOM
        rcall TX                ;RC5 IR Senderoutine
        sei                     ;Interrupt wieder erlauben und auf nächtsten Code warten
        rjmp main
```

5. Unterprogramme

5.1 „detect“ das Unterprogramm decodiert die 14 bits.
Die Dauer des 0 Pegels von Startbit 1 wird gemessen und dient dann als Referenz nach welcher Zeit der Pegel am Ausgang des TSOP1736 nachgeschaut werden muss um zu identifizieren, ob eine 1 oder eine 0 als nächstes bit vorliegt.
Danach wird auf den Wechsel des Pegels gewartet, der ja in jedem bit vorkommt, da ja ein bit, wie bekannt, aus 2 Halbbits besteht, Damit hat man einen Referenzpunkt zur Identifikation der nächsten bits. Siehe „synchronize timing“ in der detect Subroutine.

Wie sie sehen/erleben werden, nützt es nichts Programme nur abzuschreiben. Man muss sie bis ins Detail verstehen. Das geht nur bei ausführlichen Kommentaren schnell. Sonst kann es dauern bis man auf den Sinn draufkommt.
Die Unsitte wenig Kommentar zu schreiben ist weit verbreitet. Ich habe im Internet gefunden warum der Arianeprototyp abgestürzt ist. Meines Erachtens war zu wenig Kommentar dran Schuld. Auch warum am Anfang Patriot Raketen daneben geschossen haben. Es war zu wenig beschrieben unter welchen Umständen ein Programmteil verwendet werden darf.
Im Anschluss daran ist das Übersetzungsprogramm, das bei meinen Verhältnissen nur aus dem Austausch der Systemadresse von 8 (Philips Satreceiver) auf 15 des GrandPrix Satreceivers besteht.

```
;************************************************************
;* "detect"  RC5 decode routine, wartet auf Licht von der Remotecontrol
;* This subroutine decodes the RC5 bit stream applied on PORTB pin3
;*
;* wartet auf das Startbit, d.h. auf den Lowpegel = Licht des Startbits
;* Dies gilt für die meisten IR-Decoder IC's
;* die Decodierung beginnt also mit der 2. Hälfte des Startbits 1
;* If success: The comman and system address are
;* returned in "comman" and "system".
;* If failed: $FF in both "system" and "comman"
;************************************************************
detect:
        sbic pinb,3         ;If line is low, dann hat der 2. Teil des Startbits begonnen
        rjmp detect                 ;false, da noch high - warte weiter
        ;jetzt gehts los
        clr timerL                  ;Measure length of start bit
        clr inttemp                 ;Init Counters
start2:                     ;das Halbbit dauert 889us und wird mit 64us 14x abgetastet
        cpi timerL,17               ;If startbit longer than 1.1ms,
        brge fault                  ;exit
        sbis pinb,3                 ;ist Licht schon aus?
        rjmp start2                 ;count still for pos. edge of lst start bit
        mov temp,timerL     ;Licht ist aus, timerL = temp is 1/2 bit time

        clr timerL                  ;ab jetzt läuft timerL von 0 weg fürs 2. Startbit
        mov ref1,temp               ;ref1 ist                       1/2 bittime
        lsr ref1                    ;jetzt / 2 =                    1/4 bittime
        mov ref2,ref1               ;ref2 ist                       1/4 bittime
        add ref1,temp               ;letztendlich ref1 =        3/4 bit time
        lsl temp                    ;temp *2 =                              1/1 bittime
        add ref2,temp               ;letztendlich ref2 =        5/4 bittime
start3:
        cp timerL,ref1              ;If high period St2 > 3/4 bit time
        brge fault                  ;war zu lange high, Fehler also exit
        sbic pinb,3                 ;Wait for falling edge start bit 2
        rjmp start3                 ;noch high

        clr timerL                  ;gerade hat die 2.Startbithälfte begonnen, also timerL reset
                                    ;für Empfang des Togglebits
        ldi bitcnt,12               ;Receive 12 bits, beginnt mit dem Togglebit
        clr comman                  ;1.  | 2.bit| togglebit
        clr system                  ;____----____========
sample:                             ;       |-ref1-|   nach ref1 nachschauen, wie ist der Pegel
        cp timerL,ref1              ;die Abtastung läuft
        brlo sample                 ;bis timerL die richtige Zeit erreicht hat
        sbic pinb,3                 ;jetzt wird nachgeschaut welcher Pegel vorliegt.
        rjmp bit_is_a_1             ;Jump if line high

bit_is_a_0:
        clc                         ;Store a "0" im Carry Flag und schiebe es
        rol comman                  ;hier hinein und weiter
        rol system                  ;ins system
;Synchronize timing
bit_is_a_0a:
        cp timerL,ref2              ;If no edge within 5/4 bit time
        brge fault                  ;exit
        sbis pinb,3                 ;Wait for rising edge
        rjmp bit_is_a_0a            ;in the middle of the bit
        clr timerL                  ;der timerL beginnt jetzt neu zu laufen
        rjmp nextbit
```

```
bit_is_a_1:
        sec                     ;Store a "1" im Carry Flag und schiebe es
        rol comman              ;hier hinein und weiter
        rol system              ;ins system
;Synchronize timing
bit_is_a_1a:
        cp timerL,ref2          ;If no edge within 5/4 bit time
        brge fault              ;exit
        sbic pinb,3             ;Wait for falling edge
        rjmp bit_is_a_1a        ;in the middle of the bit
        clr timerL              ;der timerL beginnt jetzt neu zu laufen
nextbit:                        ;für das nächste bit
        dec bitcnt              ;If bitcnt > 0
        brne sample             ;get next bit

;All bits sucessfully received!
;Die 12 bits sind in system (4bits, toggle + 3 Sysetmbits) und
;comman (8bits, 2 Systembits und die 6 Kommandobits)
                ;Place system bits in "system"
        mov temp,comman
        rol temp                ;also 2 bits von comman via temp und C-flag ins System rollen
        rol system              ;temp deshalb, damit comman unverändertbleibt
        rol temp
        rol system
                                ;Clear remaining bits
        andi comman,0b00111111;6* Tastenkommandobits maskieren
        andi system,0x3F        ;1* Toggle + 5* Systemcodebits maskieren
;*********************************************************************************
;*jetzt folgt die Übersetzung, die im einfachsten Fall nur die 5 Systembits verändert
;*beim Globo Satreceiver müssten auch die Ziffern der Tasten übersetzt werden
;*Globo wollte mir nix über dieses Thema sagen, sind halt sehr stolz! Taste 1 sendet 3 und nicht 1 usw.
;*generell kann z.B. NEC in Skymaster übersetzt werden. Es gibt dann andere IR-Codes
;*********************************************************************************
        andi system, 0b00100000 ;Togglebit bleibt wie es ist
        ori  system, 0b00001111 ;GrandPrix Satreceiver hat 15, Globo 21=Plattenspieler!!
        ori  system, 0b11000000 ;Startbits setzen,
                                ;könnte auch in einem Befehl gemacht werden:ist so übersichtlicher
        ret                     ;comman bleibt unverändert = gleiche Tastencodes
;       KEINE Tastencodes ÜBER 63 erlaubt, Zifferntasten sind unter 63
;       dies führt in einen Fehler, falls solche übersetzt werden

fault:
        ser comman              ;Both "comman" and "system"
        ser system              ;0xFF indicates failure
        ret
```

5.2 Die IR Senderoutine mit ihren Unterprogrammen. Die Programmierung ist recht heikel, da die Zeiten der Halbbits und der Lichtan- und Lichtaus-Perioden möglichst genau einzuhalten sind.

```
;*****************************************************************************************
;*Die Senderoutine TX, auf mehrere Unterprogramme aufgeteilt
;*
;*****************************************************************************************
TX:     ldi     A,8             ;startbits, Toggle und systembits senden
        mov     S,system        ;in eine Zwischenvariable
        rcall   sende           ;ein kleiner Trick, spart etwas Code, weil sende eine Marke s.u. ist
        mov     S,comman        ;Tastencodes holen
        rol     S               ;nur 6 bits Kommando, also nach links schieben
        rol     S
        ldi     A,6
sende:  rol     S               ;1.bit ins carry schieben
        brcc    tx0             ;war eine 0, also 0 senden
tx1:    rcall   TXeins  ;
        rjmp    goon
tx0:    rcall   TXnull
goon:   dec     A               ;alle bits gesendet?
        brne    sende           ;wenn nicht dann weitersenden
        ret                     ;Alle 14 bits gesendet

TXnull: rcall   lian    ;Bicode, eine null beginnt mit Licht für 889us
        rcall   noli    ;dann das Halbbit ohne Licht
        ret

TXeins: rcall   noli    ;eine 1 beginnt mit dem Halbbit 889us ohne Licht
        rcall   lian    ;dann das Halbbit mit Licht
        ret

;*************************************************************************
;*die noli = keinLicht-Halbbit und lian = Licht-Halbbit Routinen
;*sind zeitkritisch und erfordern ziemliche Pizzelei bis man die
;*richtigen Zeiten zusammen hat. Basis 36 kHz = 27,8 us mit 6,944 + 20,833 us
;*Tastverhältnis und die 889us je Halbbit entsprechend 32 Perioden je Halbbit
;*mit 1,2 Mhz vom ATtiny13 = 0.833 us je Periode ergibt sich angenähert
;*8 x 0.833 + 25 x 0.833 = 6,664 + 20.825 us = 27.49 = 36,37 kHz = 1% Fehler
;*
;*1 Bit dauert also 890 + 880 (nops wer will) = 1,77 ms statt ideal 1,778 ms
;*oder anders gesagt der Fehler ist 0,45% . Es fehlt etwas "Licht ein"
;*wobei die optionalen nops nichts daran ändern, also lasse ich's
;*außerdem gibts ja noch die Aufrufe für die IR-Senderoutine mit 2*7 cycles
;* der Fehler bleibt in jedem Fall unter 1%
;*************************************************************************

;noli soll 889us = 1067 cycles incl. Aufruf (3 cyc) und return (4 cyc) dauern
noli:   rcall   noli1   ; 3 + 531 cycles
noli1:  ldi     B,175   ; 1 + 3 x 175 +1 +4 = 531 cycles
noli2:  nop             ; 1             ;eine einfache Delayschleife
        dec     B       ; 1
        brne    noli2   ; 1 innerhalb der Schleife, 2 Ende der Schleife
        ret             ; 4     in Summe 1065 cycles +3 für Aufruf = 1068 = 890 us

;lian soll 889 us: 3 Aufruf +1 +32x(20 + 8 + 5) +1 +4 = 1056 +9 = 1065 =880 us
;das Tastverhältnis ist etwas "ruckelig"
lian:   ldi             B,32    ; 1     ;32 Perioden eben
lian1:  ldi             C,6     ; 1 ;6x innere Schleife für 20 cyc kein Licht
lian2:  nop                     ; 1 ;
        dec             C       ; 1 ;
        brne    lian2   ; 1 + 1 fürs Ende ; = 20 cyc kein Licht
        sbi             pinb,4  ; 2 ;Licht einschalten für LED an PortB4
```

```
                nop
                nop
                nop
                nop
                nop
                nop                                     ; 6 ; nops = 8 cycles Licht an
                cbi             pinb,4    ; 2         ; Licht wieder ausschalten
                nop                                     ; 1 ; Zeitanpassung
                dec             B                       ; 1
                brne    lian1   ; 1 + 1 fürs Ende       ;schleife 32x durchlaufen
                ret                                     ; 4 ;
```

5.3 Die serielle Ausgaberoutine der Kommandos an den PC.

Diese Routine ist im „Lernpaket Mikrokontroller“ von Herrn Burkhard Kainka beschrieben.

```
;*********************************************************************************
;*Die serielle Ausgaberoutine an den PC siehe Buch Lernpaket Mikrokontrooler
;*
;*********************************************************************************


WrCOM:  sbi   portb,TXD         ;Sende Startbit, das zu sendende Byte ist in A
        ldi   temp,38
D4:     dec   temp
        brne  D4
        ldi   bitcnt,8          ;8 bit sollen raus
L2:     sbrc  A,0               ;ist bit 0 clear?
        rjmp  OFF               ;false, bit=1, wird als 0 gesendet!!!!
         rjmp  ONt              ;true, skip hierher, bit=0 BIT wird invertiert!!
ONt:    sbi   portb,TXD         ;weiter mit bit=0, setze Port auf 1
        rjmp  BitD
OFF:    cbi   portb,TXD         ;weiter mit bit=1, setze Port auf 0
        rjmp  BitD
BitD:   ldi   temp,38           ;und lass das Bit die Delayzeit stehen, es wird von der
D5:     dec   temp                      ;Readroutine auf der anderen Seite = PC abgeholt
        brne  D5
         lsr   A                ;schiebe nächstes bit vom Byte ins carry
        dec   bitcnt
        brne  L2                ;alle bits gesendet?
        cbi   PORTB,TXD         ;dann Port clear=0 und drauf warten, dass Stopbit
        ldi   temp,38
D6:     dec   temp
        brne  D6
```

Wie sie ja nun wissen, wird ein Kommano, das 25 mS dauert, detektiert, übersetzt, an den PC gesendet und dann als IR Kommano an den Satreceiver gesendet.
Das geschieht so schnell, dass alles erledigt ist, ehe ein Wiederholkommando nach 114 mS eintrifft, sprich Finger ist lange auf der Taste. So konnte ich feststellen, dass mein DVD Recorder jedes Tastenkommando 4x sendet und eine VCR Fernbedienung z.B. jedes Kommando 2x. Übrigens ein netter/übler Nebeneffekt: Jede RC5 Fernbedienung deren Licht beim Übersetzer ankommt wird auf Systemkommando 15 übersetzt.
Jetzt kann ich Ihnen nur alles Gute wünschen und hoffe, dass Ihnen ein sehr detailliert geschriebener Report einen leichten Anfang bietet in die uC Programmierung einzusteigen.

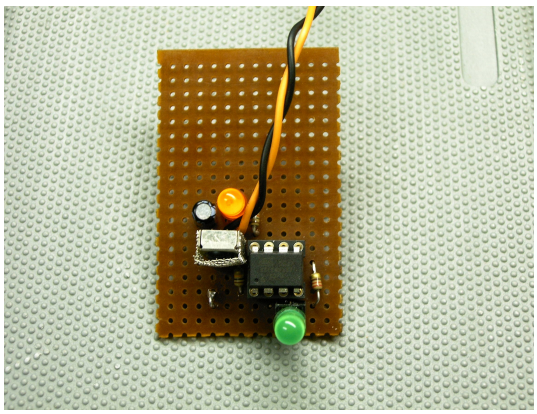
Die Linkliste:

http://www.avr-asm-tutorial.net/
http://www.atmel.ru/Disks/AVR%20Technical%20Library/appnotes/appnotes.html
http://atmel.com/dyn/products/datasheets.asp?family_id=607#791
http://www.roboternetz.de/wissen/index.php/RC5-Decoder_f%C3%BCr_ATMega
http://www.sprut.de/electronic/ir/rc5.htm
http://www.sprut.de/electronic/pic/programm/irrc/irrc.html
http://www.mikrocontroller.net/topic/12216
http://www.holger-klabunde.de/
http://www.mikrocontroller.net/en/links
http://www.myplace.nu/avr/yaap/
http://s-huehn.de/elektronik/avr-prog/avr-prog.htm
http://www.lancos.com/ppwin95.html
http://www.lancos.com/siprogsch.html
http://www.esnips.com/web/AtmelAVR

Eine sehr lehrreiche Seite, man kann den Pfad hinaufgehen
http://ltc.cit.cornell.edu/courses/ee476/FinalProjects/s2000/chan/MATH.ASM
diese werde ich für mein nächstes Projekt brauchen.

http://avr-asm.tripod.com/avr200.html
http://www.mikrocontroller.net/articles/Linksammlung
http://www.roboternetz.de/wissen/index.php/Kategorie:Projekte
http://www.educypedia.be/electronics/televisionrc5.htm
http://www.epanorama.net/links/irremote.html
http://www.remotecentral.com/cgi-bin/codes/daewoo/
http://www.sbprojects.com/knowledge/index.htm
http://www.sbprojects.com/knowledge/ir/recs80.htm

Letzte Änderungen am Programm:
.org 16 eingefügt vor der Marke Anfang: . Hängt mit dem Bootloader zusammen, der ja im Lernpaket benützt wird. Kann mit anderen Loadern wie Ponyprog entfallen.
Nach dem HW Aufbau des Prototyps stellte sich heraus, dass das Programm nicht immer startet. Es hängt auch mit dem Bootloader zusammen, der auf ein Break Signal für den Aufruf des Bootloaders wartet. Das kann bei unbeschalteten Eingängen auftreten. Habe also an Reset eine 10 kOhm nach Vcc eingelötet und an den Ports 0 und 2 je einen Widerstand ca. 4k gegen Masse eingelötet.
Dann startet das Programm sicher. Wie schon beschrieben ist an Port 3 der TSOP und an Port 4 über 100 Ohm gegen Masse die IR Diode. GND und Vcc ist klar. Damit sind alle Anschlüsse definiert.



Ein Bild sagt mehr als tausend Worte: Das gelbe Led ist der Indikator für die Betriebsspannung.
Das grüne Led statt der IR-Diode. Wenn man eine Photodiode verwendet und beide IR-Signale empfängt, das Original und die Übersetzung, so sieht man nur einen Zwischenraum von etwa 5 mS. Kaum ist das Original empfangen, so ist die Übersetzung schon gesendet. Totzeit vielleicht doch vorsehen!