433-MHz-Funksteckdosen - nicht nur!

Einleitung

Schon seit Jahren gibt es die unterschiedlichsten Geräte, die auf der Basis von Funkwellen im 433-MHz-Band arbeiten: Autoschlüssel, Fernbedienungen für Steckdosen, Funkrauchmelder, um nur einige zu nennen. Bei manchen sind die Protokolle nicht offen gelegt oder durch Codes geschützt, andere wiederum benutzen Protokolle, die bekannt sind; dazu gehören insbesondere verschiedene Funksteckdosen, wie sie z. B. bei Discountern oder Baumärkten preisgünstig zu erwerben sind.

Nun gibt es - ebenfalls sehr preiswert (genauergesagt ab etwa 1 Euro) - einfache Empfänger und Sender für genau diesen Frequenzbereich zu kaufen. Auf der Webseite http://www.forum.g-heinrichs.de/viewtopic.php?f=12&t=76 habe ich ein solches Paar recht ausführlich vorgestellt. Dort habe ich auch gezeigt, wie man damit Textnachrichten über ein einfachen Protokoll senden und empfangen kann.

Abbildung 1

Im Internet findet man nun auch Anleitungen, wie man mit genau diesen Moduln eine Kommunikation zwischen den oben genannten Geräten und einem Mikrokontroller herstellen kann. Für das Arduino-System gibt es eine Library (RCSwitch), die hier gute Dienste leistet. Leider funktioniert diese nicht immer; das liegt daran, dass diese Geräte unterschiedliche Protokolle benutzen. Einige dieser Protokolle sind bekannt, andere sind nicht veröffentlicht oder sogar verschlüsselt.

Nun arbeitet nicht jeder mit einem Arduino-System; vielleicht aber will man sich auch nicht mit einer Black-Box-Sichtweise begnügen. Möglicherweise benutzt das Gerät, mit dem man arbeiten möchte, gerade eine Protokollvariante, die von der Arduino-Library nicht erfasst wird. Grund genug, sich einmal systematisch mit solchen Protokollen zu beschäftigen.

Abbildung 2

Damit dies aber nicht zu theoretisch wird, möchte ich dies an Hand eines konkreten Beispiels durchführen: Ziel soll es sein, das Protokoll einer ELRO-Funksteckdose zu ermitteln und mit dem Attiny 2313 einen entsprechenden Sender sowie einen Empfänger bauen. Als Programmiersprache verwende ich BASCOM. Dabei kommt es mir insbesondere darauf an, die Vorgehensweise so darzulegen, dass man sie auch auf andere Geräte mit ähnlichen Protokolle anwenden kann.

Gängige Protokolle

Eine Recherche liefert verschiedene Protokolle für Funksteckdosen. Sie lassen sich übersichtlich mit ihren **Telegrammen** beschreiben; das sind Diagramme, in denen die High- und Low-Zustände in ihrem zeitlichen Verlauf dargestellt sind:

Abbildung 3

Auf den ersten Blick könnte man vielleicht meinen, dass die High-Zustände für ein Bit 1 und die Low-Zustände für ein Bit 0 stehen. Eine solche Zuordnung wird zum Beispiel bei der Datenübertragung über die serielle Schnittstelle (UART) benutzt. So einfach ist es bei unseren Funkprotokollen nicht. Bei vielen Funkprotokollen sieht die Zuordnung so aus:

Abbildung 4

Das **Bit 0** wird also durch einen kurzen Puls gekennzeichnet. Die Dauer dessen High-Zustands (hier 1 Kästchen) bezeichnen wir als **Basislänge**. Auf den High-Zustand folgt ein Low-Zustand von 3 Basislängen. Die Gesamtheit der beiden Zustände repräsentiert das Bit 0.

Das **Bit 1** wird hingegen durch einen langen Puls gekennzeichnet. Die Dauer dessen High-Zustands beträgt 3 Basislängen. Auf den High-Zustand folgt ein Low-Zustand von 1 Basislänge. Die Gesamtheit dieser beiden Zustände repräsentiert das Bit 1.

Daneben gibt es noch weitere Signaltypen, auf die wir gleich noch zu sprechen kommen werden. Vorher wollen wir uns klar machen, warum hier diese Art der Kodierung sinnvoll ist. Kurz gesagt ist diese Übertragung weniger störanfällig; in unserem Fall könnte das UART-Protokoll sogar überhaupt nicht funktionieren. Die High- und Low-Signale werden hier nämlich einfach dadurch übertragen, dass eine elektromagnetische Welle gesendet wird oder eben nicht. Kommt beim Empfänger eine elektromagnetische Welle an, setzt er dies in High-Signal um und gibt sie an seinem Ausgang aus, andernfalls gibt er ein Low-Signal aus. Nun besitzt der Empfänger einen Verstärker für das empfangene elektrische Signal; ist das elektrische Signal schwach, so wird die Verstärkung automatisch erhöht. Geht nun vom Sender keine elektromagnetische Welle aus, so bedeutet das nicht, dass der Empfänger auch keine elektromagnetische Welle empfängt. Denn in der Regel sind wir einer Störstrahlung ausgesetzt, die von den unterschiedlichsten Geräten ausgeht. Wenn nun unser Empfänger diese Störstrahlung als schwaches Signal registriert, erhöht er die Verstärkung so lange, bis er schließlich auch High-Signale ausgibt.

Abbildung 5

Nun erfolgt diese Erhöhung der Verstärkung nicht schlagartig: In Abb. 5 wurde der Sender bei ca. 0,35 s ausgeschaltet; erst knapp 0,2 s später ist die Verstärkung so stark angewachsen, dass die Störstrahlung als unregelmäßiges Signal registriert wird. Demnach werden die vom Sender erzeugten Zustände (Welle ein- bzw. ausgeschaltet), wie sie in Abbildung 4 gezeigt sind, vom Empfänger immer korrekt umgesetzt, wenn die Basislänge so klein gewählt ist, dass alle Low-Phasen deutlich kleiner als 0,2 s sind. Das wurde bei der Festlegung der Protokolle natürlich berücksichtigt: Low-Phasen sind hier meist kürzer als 1 ms (vgl. Abb. 3). Würde man zur Übertragung ein einfaches USART-Protokoll benutzen, müsste man mit Störungen rechnen, wenn mehrere 0-Bits hintereinander übertragen würden; in diesem Zeitraum würde nämlich die Verstärkung erhöht und der Empfänger würde “falsche 1-Bits” ausgeben.

Kommen wir nun zum so genannten **Synchronisationssignal**. Das sieht so aus:

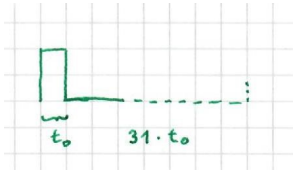

Abbildung 6

Es besteht aus einer High-Phase von 1 Basislänge, hier mit t_0 gekennzeichnet; es schließt sich eine Low-Phase an, die 31 Basislängen dauert. Funksteckdosen übertragen ihre Daten in Form von **Datenpaketen** mit 24 Bit. Wir werden diese im Folgenden als 3 Bytes deuten. Die Übertragung jedes Datenpakets wird durch ein Synchronisationssignal eingeleitet. In Abb. 3 können wir ein solches Synchronisationssignal ausmachen; hier schließt es sich direkt an ein zuvor gesendetes Datenpaket (mit 24 Bit) an, um das nächste Datenpaket einzuleiten, dessen erstes High-Signal so gerade eben noch am rechten Rand des Bildes zu erkennen ist.

Viele Funksteckdosen benutzen Protokolle, so wie sie gerade dargestellt worden sind. Allerdings unterscheiden sie sich z. B. in der Basislänge und der Deutung der Datenbits.

Typische Basislängen liegen zwischen 300 und 500 Mikrosekunden. Ein Empfänger könnte die Basislänge beim Synchronisationssignal messen und den so ermittelten Wert zur Dekodierung der folgenden Signale nutzen. Leider arbeiten die Geräte nicht immer präzise: Bei meiner Fernbedienung dauerte z. B. die High-Phase des Synchronisationssignals etwas länger als die High-Phase des 1-Signals. Zudem war die Low-Phase des Synchronisationssignals auch länger als das 31-fache der Basislänge (vgl. Abb. 7 und 8). Solche Toleranzen wird man bei der Programmierung eines Dekoders berücksichtigen müssen.

Bei manchen Protokollen werden jeweils 2 Datenbits zur Beschreibung eines so genannten Tri-State (-Zustands) zusammen gefasst.

Doppelbit	*Tri-State*
0 0	0
1 1	1
0 1	F

Diese Tri-State-Deutung ist für uns hier nicht so wichtig, weil es uns nur um die Identifizierung der Signale gehen wird. Deswegen soll hier auch nicht weiter darauf eingegangen werden.

Daneben gibt es auch ein Protokoll, bei dem die einzelnen Bits jeweils durch ein Signal mit zwei Pulsen dargestellt werden; dabei sind beide Signale insgesamt gleich lang, unterscheiden sich aber in der Dauer der einzelnen Low-Phasen: 0: HLHLL und 1: HLLHL. Auch hierauf möchte ich hier nicht weiter eingehen.

Analyse der Fernbedienungssignale

Zunächst schließe ich das Empfangsmodul an eine elektrische Quelle von 5 V an; dazu benutze ich meine Attiny-Platine (http://www.g-heinrichs.de/wordpress/index.php/attiny/). Die beiden Data-Ausgänge unterscheiden sich nicht in ihrer Funktion; einen davon schließe ich an Kanal 0 meines Logik-Analysators an; außerdem verbinde ich die Masse der Attiny-Platine mit dem Masse-Anschluss des Logik-Analysators.

Nun starte ich das zugehörige Programm, stelle die Messdauer auf einige 100 ms ein. Getriggert wird über eine positive Flanke. Nun halte ich den Knopf "A on" bei der Fernbedienung gedrückt und starte anschließend die Messung. Man sieht eine Reihe von Datenpaketen; Abb. 3 zeigt eine Ausschnittvergrößerung davon.

In der Regel wird die Signalaufzeichnung nicht mit einem Synchronisationssignal beginnen. Den Anfang eines Datenpakets erkennt man aber daran, dass er nach einer langen Low-Phase liegt.

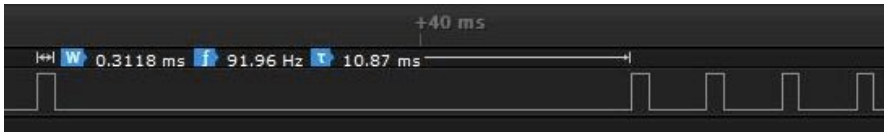

Abbildung 7

Die High-Phase des Synchronisationssignals dauert hier ca. 0,32 ms und die Lowphase dauert etwa 10,87 ms - 0,32 ms = 10,55 ms. Das Verhältnis der beiden Zeiten ist ungefähr 33, weicht also etwas von dem oben genannten Wert 31 ab.

Schauen wir uns nun den Anfang des Datensignals an:

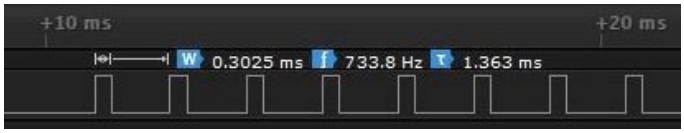

Abbildung 8

Offensichtlich handelt es sich hier um lauter 0-Bits. Die Basislänge des ersten 0-Bits beträgt ungefähr 0,30 ms, weicht demnach etwas von der High-Phase des Synchronisationssignals ab. Die Low-Phase des 0-Bits dauert etwa 1,36 ms - 0,30 ms = 1,06 ms. Das ist etwas mehr als das Dreifache der Basislänge. Die Protokollangaben werden also nur näherungsweise erfüllt.

Betrachten wir nun das ganze Datenpaket:

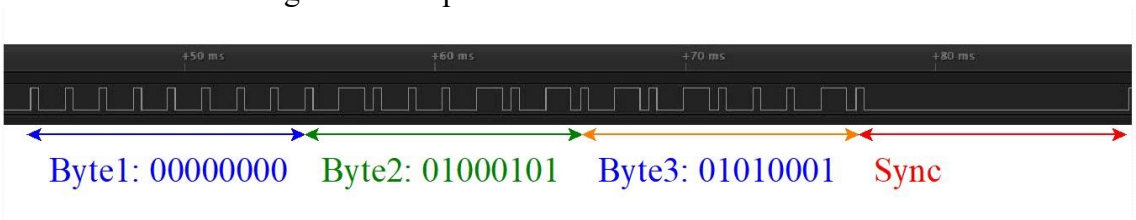

Abbildung 9

Im Zehnersystem lautet das Datenpaket also 0:69:81. Wenn man das Datenpaket für die Taste "A off" in gleicher Weise ermittelt, erhält man 0:69:84.

Sender für die Funksteckdose

Nun soll ein Mikrocontroller im Verbund mit dem Sendemodul die Funksteckdose über die Taster Ta0 und Ta1 ein- und ausschalten. Dazu schließe ich das Sendemodul an die Buchsenleiste der Attiny-Platine an; Vcc und GND werden dabei mit +5V bzw. Masse verbunden, der Eingang "ATAD" (Beschriftung des Moduls wohl versehentlich in Spiegelschrift!) mit Port B.0. Die beiden Taster liegen zwischen D.2 bzw. D.3 und Masse. Das folgende BASCOM-Programm sollte aufgrund der Modularisierung selbsterklärend sein:

```
' Programm-Datei für Attiny-Platine von E. Eube, G. Heinrichs und U. Ihlefeldt
' Über die zu zugehörige Konfigurationsdatei werden automatisch Voreinstellungen
' für die Kompilierung übernommen; diese betreffen z. B. die Taktfrequenz, die
' Baudrate, die Anschlüsse von LCD, I2C- und SPI-Modulen.
'
' ELRO-Funksteckdose; Sender (Codes über Dualzahlen angegeben)
' Taster Ta0: A einschalten
' Taster Ta1: A ausschalten
' 433MHz-Sender: Data ("ATAD") an B.0
'
'_____________________________________________________________________________
```

```
$regfile = "attiny2313.dat"

'*********************************************************
'******************* Deklarationen ***********************

Dim B1 As Byte
Dim B2 As Byte
Dim B3 As Byte
Dim Puls1 As Word
Dim Puls2 As Word
Dim Puls3 As Word
Dim I As Byte
Declare Sub A_einschalten
Declare Sub A_ausschalten
Declare Sub Sendebyte(b As Byte)
Declare Sub Sende_bit1
Declare Sub Sende_bit0
Declare Sub Sync

'*********************************************************
'****************** Initialisierung **********************

Ddrb = &B11111111                           'Port B als Ausgangsport
Ddrd = &B01110000                           'D4, D5, D6 als Ausgang; Rest als Eingang
Portd = &B10001111                          'Eingänge auf high legen
Waitms 50                                   'warte bis Kondensator bei Ta0 geladen

Puls1 = 300                                 'Zeit in Mikrosekunden
Puls2 = 3 * Puls1
Puls3 = 33 * Puls1

'*********************************************************
'******************* Hauptprogramm ***********************

' Der Taster muss solange betätigt werden, dass das Unterprogramm mehrere Male
' (mindestens 3 mal) hinereinander ausgeführt wird, also mindestens ca. 150 ms.
Do
  If Pind.2 = 0 Then Call A_einschalten
  If Pind.3 = 0 Then Call A_ausschalten
Loop

'*********************************************************
'****************** Unterprogramme ***********************

Sub A_einschalten
  B1 = &B00000000
  B2 = &B01000101
  B3 = &B01010001
  Call Sync
  Call Sendebyte(b1)
  Call Sendebyte(b2)
  Call Sendebyte(b3)
End Sub

Sub A_ausschalten
```

```
  B1 = &B00000000
  B2 = &B01000101
  B3 = &B01010100
  Call Sync
  Call Sendebyte(b1)
  Call Sendebyte(b2)
  Call Sendebyte(b3)
End Sub

Sub Sync
  Portb.0 = 1
  Waitus Puls1
  Portb.0 = 0
  Waitus Puls3
End Sub

Sub Sendebyte(b As Byte)
  For I = 7 To 0 Step -1
    If B.i = 1 Then Call Sende_bit1 Else Call Sende_bit0        'B.i ist i-tes Bit von B
  Next I
End Sub

Sub Sende_bit1
  Portb.0 = 1
  Waitus Puls2
  Portb.0 = 0
  Waitus Puls1
End Sub

Sub Sende_bit0
  Portb.0 = 1
  Waitus Puls1
  Portb.0 = 0
  Waitus Puls2
End Sub
```

Unter http://www.forum.g-heinrichs.de/download/file.php?id=194 findet man ein Video, in welchem zu sehen ist, dass sich die Funksteckdose über unser selbst konstruierte “Fernbedienung” einwandfrei steuern lässt. Natürlich kann man diese Steuerung modifizieren: so kann man z. B. die Taster durch andere Sensoren ersetzen; denkbar ist auch eine Zeitsteuerung.

Fernbedienungssignale empfangen und dekodieren

Haben wir bislang die Dekodierung der Signale von der Fernbedienung manuell vorgenommen, kann dies auch vom Mikrocontroller vorgenommen werden. Verschiedene Anwendungen ergeben sich hierfür: Mit der Fernsteuerung der Funksteckdose können andere Geräte, die nicht über eine Steckdose betrieben werden, ein und ausgeschaltet werden; gegebenenfalls kann auch der Programmablauf eines Regelungsprozesses per Fernsteuerung beeinflusst werden. Kommen die Signale z. B. von einem Rauchmelder oder einer Einbruchssicherungsanlage, können so vom

Mikrocontroller automatisch Warnungen (z. B. per SMS, vgl. den Forumsbeitrag "Attiny sendet Alarm-SMS" unter http://www.forum.g-heinrichs.de/viewtopic.php?f=12&t=95) gesendet werden.

Wie geht man nun vor? Zunächst schließe ich das Empfangsmodul an die Buchsenleiste der Attiny-Platine an; Vcc und GND werden dabei mit +5V bzw. Masse verbunden, einer der Ausgänge DATA wird an den Port B.2 angeschlossen; beim Attiny2313 dient dieser Anschluss auch als Interrupteingang INT0. Von dieser Funktion wird unser Programm Gebrauch machen.

Der Interrupt INT0 wird nun so konfiguriert, dass bei jeder Änderung - sei es ein Wechsel von Low nach High oder ein Wechsel von High nach Low - ein Interrupt ausgelöst wird. Die Interruptroutine sieht so aus:

```
Pulskontrolle:
  If Pind.2 = 1 Then                'Steigende Flanke
    Low_time = Timer1               'Länge der vorangegangenen Low-Phase
    Puls_end = 0                    'Ende der Low-Phase erreicht
  Else
    High_time = Timer1              'Länge der vorangegangenen High-Phase
    Puls_end = 1                    'Ende der High-Phase erreicht
  End If
  Timer1 = 0
Return
```

Diese Routine hält in der (globalen) Variable Puls_end fest, ob die vorangegangene Phase eine Low-Phase oder eine High-Phase war; in den Variablen Low-time und High_time wird jeweils die Länge dieser Phase gespeichert. Die Zeiten werden mit Hilfe der Timer1-Komponente des Attinys in der Einheit 2 us gemessen; ein Zählerstand von 150 bedeutet demnach eine Zeit von 300 us.

Durch diese Informationen können die registrierten Pulse analysiert werden. Am aufwendigsten fällt das Aufspüren des Synchronisationssignals aus; vor diesem Signal müssen wir nämlich mit Störsignalen rechnen (s. o.).

```
Sub Warte_auf_sync
  Do
    If Low_time > Sync_min And Low_time < Sync_max Then
                          Sync_low_phase_vorhanden = 1 Else Sync_low_phase_vorhanden = 0
  Loop Until Sync_low_phase_vorhanden = 1 And High_time > High_time_min
End Sub
```

Dabei geben die Variablen Sync_min und Sync_max den Toleranzbereich für die Basislänge, d. h. für die Dauer des High-Zustand beim Synchronisationssignal an.

Bei den nun folgenden Datensignalen müssen wir nicht mit solchen Störsignalen rechnen. Somit kann beim Detektieren der Datenbits einfach mit einer Do-Loop-Until-Schleife gearbeitet werden. Es folgt das gesamte Empfangsprogramm:

```
' Programm für Attiny-Platine von E. Eube, G. Heinrichs und U. Ihlefeldt
' Über die zu zugehörige Konfigurationsdatei werden automatisch Voreinstellungen
' für die Kompilierung übernommen; diese betreffen z. B. die Taktfrequenz (hier: 4 MHz),
' die Baudrate, die Anschlüsse von LCD, I2C- und SPI-Modulen.
'
' ELRO-Funksteckdose
' Signale empfangen und dekodieren; Anzeige auf LCD
' 433MHz-Empfänger: Data an D.2 (INT0)
'
'-------------------------------------------------------------------------------

$regfile = "attiny2313.dat"

'*********************************************************
'****************** Deklarationen ************************

Dim B1 As Byte
Dim B2 As Byte
Dim B3 As Byte
Dim Basislaenge As Word
Dim Puls1 As Word
Dim Puls2 As Word
Dim Puls4 As Word
Dim I As Byte
Dim Sync_min As Word
Dim Sync_max As Word
Dim Puls_end As Byte
Dim High_time As Word
Dim High_time_min As Word
Dim Low_time As Word
Dim Wert As Byte
Dim Wert1 As Byte
Dim Wert2 As Byte
Dim Wert3 As Byte
Dim Bitwert As Byte
Dim Sync_low_phase_vorhanden As Byte
Declare Function Empfange_byte() As Byte
Declare Function Empfange_bit() As Byte
Declare Sub Warte_auf_sync

'*********************************************************
'****************** Initialisierung **********************

Ddrb = &B11111111                           'Port B als Ausgangsport
Ddrd = &B01110000                           'D4, D5, D6 als Ausgang; Rest als Eingang
Portd = &B10001111                          'Eingänge auf high legen
Waitms 50                                   'warte bis Kondensator bei Ta0 (D.2) geladen
Enable Int0                                 'vgl. BASCOM-Hilfe...
Config Int0 = Change
On Int0 Pulskontrolle                       'misst Dauer der High- bzw. Low-Phasen
Basislaenge = 300                           'Zeit in us
Puls1 = 300 / 2                             'Zeit in Timer1-Einheiten (2 us)
Puls2 = 3 * Puls1
Puls4 = 2 * Puls1
High_time_min = Basislaenge / 3
```

```
Sync_min = 30 * Puls1                          'untere Grenze für sync_low
Sync_max = 40 * Puls1                          'obere Grenze für sync_low
Low_time = 0
High_time = 0

'*********************************************************
'******************** Hauptprogramm **********************

Cls
Lcd "RC-Codes"
Wait 1
Cls
Enable Interrupts
Tccr1b = &B000000010                           'Timer1Clock = Clock/8, d.h. 2us; einschalten

Do
  Call Warte_auf_sync
  Wert1 = Empfange_byte()
  Wert2 = Empfange_byte()
  Wert3 = Empfange_byte()
  Lcd Wert1
  Lcd ":"
  Lcd Wert2
  Lcd ":"
  Lcd Wert3
  Wait 3                                        'Anzeige-Pause
  Cls
Loop

'*********************************************************
'****************** Unterprogramme ***********************

Sub Warte_auf_sync
  Do
    If Low_time > Sync_min And Low_time < Sync_max Then Sync_low_phase_vorhanden = 1 Else
                                                        Sync_low_phase_vorhanden = 0
  Loop Until Sync_low_phase_vorhanden = 1 And High_time > High_time_min
End Sub

Function Empfange_byte()
  Wert = 0
  For I = 0 To 7
    Wert = 2 * Wert
    Bitwert = Empfange_bit()
    Wert = Wert + Bitwert
  Next I
  Empfange_byte = Wert
End Function

Function Empfange_bit()
  Puls_end = 0
  Do                                            'Ende der High-Phase abwarten
  Loop Until Puls_end = 1
  If High_time > Puls4 Then Empfange_bit = 1 Else Empfange_bit = 0
End Function
```

```
'*********************************************************
'*****************Interruptroutinen***********************

Pulskontrolle:
  If Pind.2 = 1 Then                    'Steigende Flanke
    Low_time = Timer1                   'Länge der vorangegangenen Low-Phase
    Puls_end = 0                        'Ende der Low-Phase erreicht
  Else
    High_time = Timer1                  'Länge der vorangegangenen High-Phase
    Puls_end = 1                        'Ende der High-Phase erreicht
  End If
  Timer1 = 0
Return
```

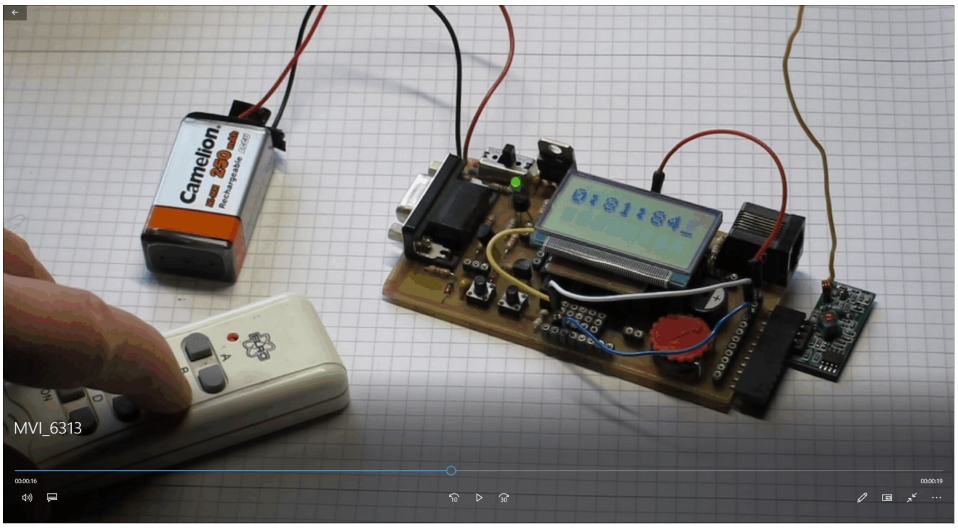

Abbildung 10: http://www.forum.g-heinrichs.de/download/file.php?id=193

Abb. 10 zeigt den Screenshot eines Videos von meinem Attiny-Forum. Hier wurde gerade der B-Off-Knopf gedrückt; die empfangene Byte-Folge ist 0:81:84.

Zuletzt noch eine Zusammenstellung der Byte-Folgen für sämtliche Tasten meiner ELRO-Fernbedienung:

Steckdose	On	Off
A	0:69:81	0:69:84
B	0:81:81	0:81:84
C	0:84:81	0:84:84
D	0:85:17	0:85:20