

ATtiny2313 – FB-Empfänger

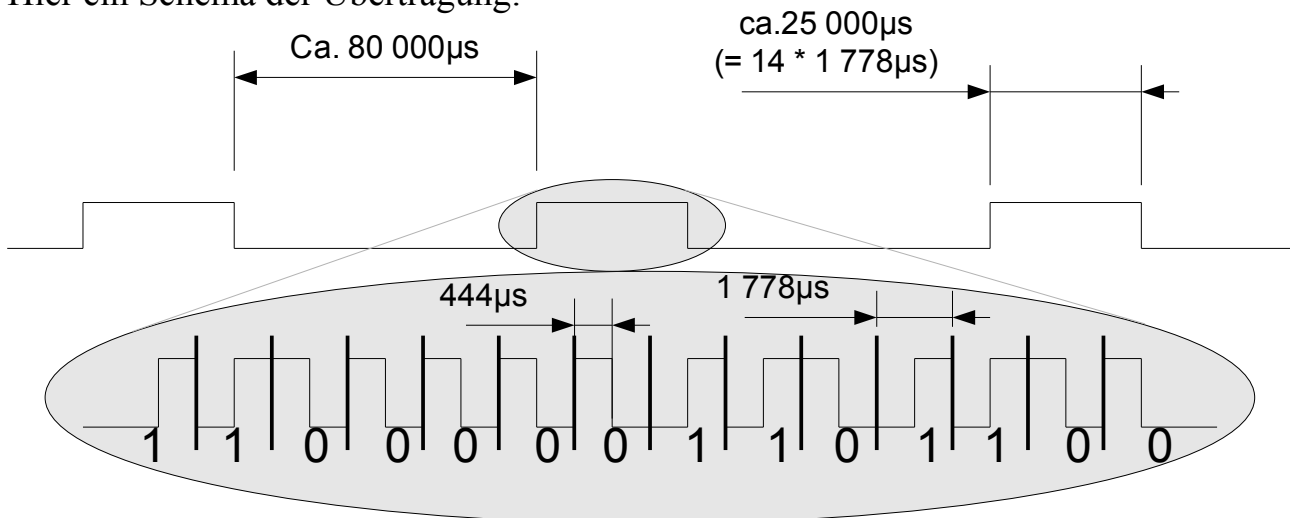
Überlegung wie man RC-5 dekodiert

Wir haben uns mit der Frage auseinandergesetzt, wie wir am einfachsten und effizientesten RC-5 dekodieren. Zu Beginn des Projekts habe ich schon erklärt, wie RC-5 arbeitet, jedoch will ich das an dieser Stelle nochmal auffrischen:

RC-5(x) – Ein großes Protokoll, das man in kleine Stücke zerlegt:

RC-5 hat eine kleine Besonderheit, es ist Manchester-codiert. D.h. Jedes Bit wird als Übergang übertragen. Dadurch ist mehr Sicherheit bei der Übertragung gewährleistet, da man sofort mitbekommt, wenn die Übertragung abbricht. Jedoch darf man nicht in der Mitte eines Bits überprüfen, welchen Status es hat, da dort die Flanke ist, wo sich der Zustand ändert, sondern muss in der Mitte der Halbbits testen. Sind die beiden Halbbits gleich, stimmt die Übertragung nicht mehr und man kann abbrechen!

Hier ein Schema der Übertragung:



Eine Übertragung besteht aus 14 Bits:

- 1 Startbit ist immer 1
- 1 2. Startbit (oder bei RC-5x ein 7. invertiertes Kommandobit (=>128 Kommandos))
- 1 Toggelbit, das sich bei jedem ERNEUTEM Tastendruck ändert
- 5 Adressbits, die die Adresse des Zielgerätes angeben
- 6 Kommandobits

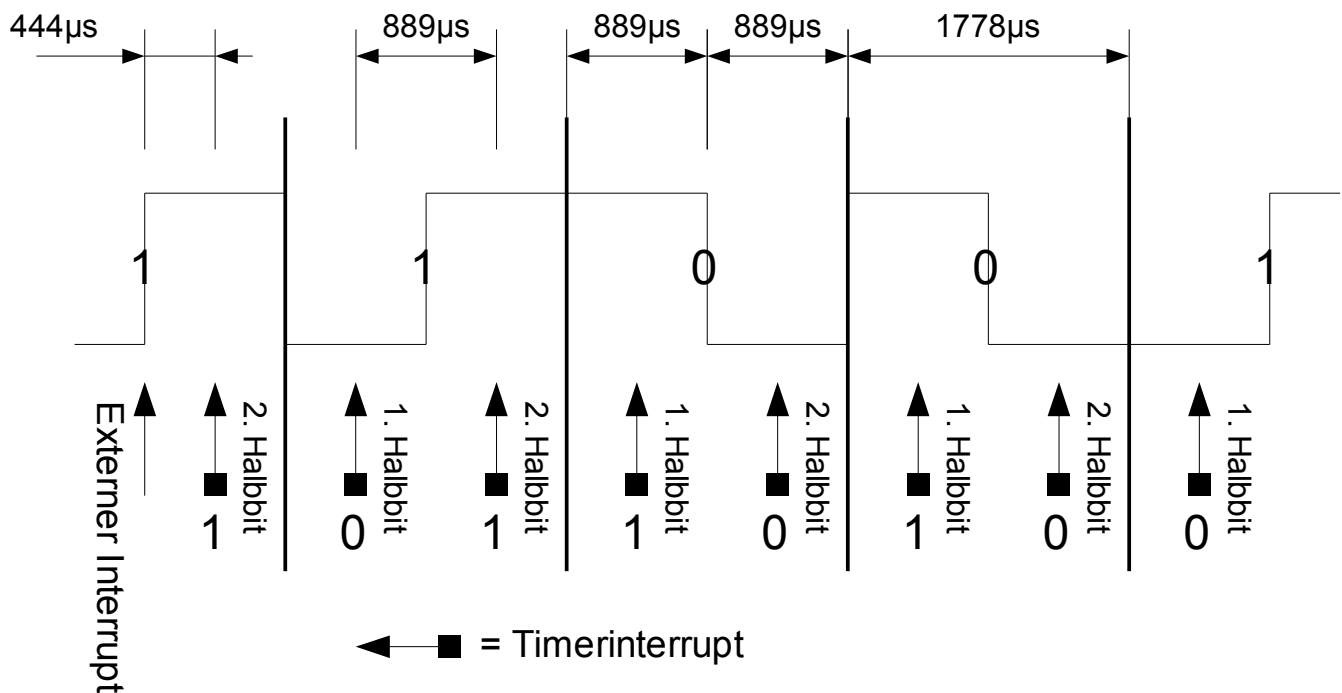
Daraus ergibt sich, dass jede Übertragung 14 Bits lang ist.

Das Dekodieren - Synchronisieren und nacheinander abspeichern:

Das Protokoll kann man auf viele Arten „lesen“. Am Freitag haben wir etwas diskutiert, wie wir es am einfachsten dekodieren. Es gab ein paar Möglichkeiten, wobei wir uns schließlich auf folgende geeinigt haben:

Der µC geht von einer vollständigen Übertragung aus, wenn er das Startbit durch einen Externen Interrupt merkt.

Hierzu eine Grafik:



Von nun an werden Timerinterrupts nach 889µs ausgelöst, in denen speichert man die Zustände ab und schiebt das Register um eins nach links. Dabei unterscheidet man in 1. und 2. Halbbit; jedes bekommt ein eigenes 16Bit Schieberegister. Also hat man effektiv zwei Register in die abwechselnd der 1.Halbbitzustand und der 2.Halbbitzustand geschrieben werden.

Dadurch erhält man 2*2Byte Daten(2 * 14 Datenbits->16Bit->2Byte), die zum Schluss einfach übereinander gelegt werden und mit dem XOR Gatter verglichen werden. Findet sich nach dem XOR irgendwo eine 0, waren die beiden Halbbits gleich und wie wir oben schon erklärt haben, die Übertragung somit nicht korrekt. Hier soll das mal an einem Bsp. mit 8-Bit gezeigt werden:

XOR-Gatter

A	0	0	1	1
B	0	1	0	1
Ergebnis	0	1	1	0

Bsp.: 1 Byte mit korrekten Daten; 1 Byte mit Fehlerhaften:

1	1	0	0	1	1	0	1	Byte der 1. Halbbits
0	0	1	1	0	0	1	0	Byte der 2. Halbbits
=	=	=	=	=	=	=	=	XOR-Verknüpfung
1	1	1	1	1	1	1	1	Alles OK

1	1	1	1	0	0	1	0	Byte der 1. Halbbits
0	0	1	0	1	1	0	0	Byte der 2. Halbbits
=	=	=	=	=	=	=	=	XOR-Verknüpfung
1	1	0	1	1	1	1	0	Bei zwei Bits ist ein Fehler aufgetreten!

Der Code – Anwenden was man schon kennt:

Jetzt kennt man das RC-5 Protokoll und hat sich überlegt, wie man es dekodiert. Nun muss man sich, bevor man Code schreibt überlegen, was wo gemacht werden muss. Diesen Schritt haben wir am Freitag übersprungen.

- Zuerst muss man vieles initialisieren. Darunter fällt sowohl, den externen Interrupt zu initialisieren als auch den Prescaler des Timers zu setzen. Welchen Prescaler müssen wir eigentlich verwenden? 8MHz ist die Prozessorfrequenz und nach $889\mu\text{s}$ muss der Timer auslösen: $8\text{MHz} * 889\mu\text{s} = 7112$; Jetzt muss man ausprobieren, welcher **Prescaler** die Zahl soweit teilt, dass das Ergebnis unter 256 ist: $7112 / 64 = \text{ca.}111$
- Wenn der externe Interrupt ausgelöst wird muss man diesen Interrupt deaktivieren und stattdessen den Timerinterrupt aktivieren. Der Timer soll so eingestellt werden, dass er in $444\mu\text{s}$ oder mehr auslöst. Außerdem müssen die Register resettet werden, die für die Dekodierung benötigt werden.
- Wenn der Timerinterrupt ausgelöst wurde, dann muss man unterscheiden, das wievielte Mal dieser Interrupt seit den externen Interrupt ausgerufen worden ist.
 - Beim 1.Mal muss man den Zustand für das 1. Halbbit auf null setzen und für das 2. auf eins. Außerdem muss man den Timer von nun an auf $889\mu\text{s}$ auslösen lassen.
 - Beim 27. Mal ($13 \text{ Datenbits} * 2 \text{ Halbbits} + 1 \text{ halbes Startbit}$) muss man das Empfangene auf Fehler überprüfen und dann evtl. weiter verwenden. Jedoch dazu in der nächsten Stunden mehr...
 - Wenn die Anzahl der Aufrufe GERADE ist, muss man den Zustand in das 1.Halbbitregister schreiben.
 - Wenn die Anzahl der Aufrufe UNGERADE ist, muss man den Zustand in das 2.Halbbitregister schreiben.

Soweit sollte alles klar sein. Diesen Text kann man nun in Assembler übersetzen und dann sollte alles wie erwartet klappen! Dann kommen wir jedoch zur nächsten Frage, wie wir erkennen, dass eine Taste länger oder kürzer gedrückt worden ist, und wie man darauf reagieren kann. Man muss sich dann also nach der Lowlevel-Software mit der Higherlevel-SW beschäftigen. D.h. aber nicht, dass es schwieriger wird, sondern dass der Bezug zur Hardware entfernter wird.