Atmel 328-Prozessor und RaspberryPi

AD-Wandler, Frequenzmessung, Ereigniszähler, IR-Fernsteuerung

Gerhard Hepp, März 2014

Inhaltsverzeichnis

Überblick....3
Setup....3
Einkaufsliste....3
Aufbau....4
Software auf dem RPi installieren....5
Überprüfen der Hardware und der Programme....5
Fuses auslesen....5
Auslesen des aktuellen Programmes....6
Blink-Programm laden....6
Programmieren der Fuses....6
Programmieren der Firmware....6
Firmware Beschreibung....6
Firmware: Version auslesen....7
Firmware: LED-Kommandos....7
Firmware: Lesen der ADC Werte....7
ADC, Anbindung an Scratch....8
Firmware: Ereigniszähler, Frequenzmessung....8
Firmware: Ereigniszählung COUNTER....9
Firmware: Frequenzmessung TIMER....9
Überprüfen der Genauigkeit....9
Firmware: Frequenzmessung mit einer Zeitbasis 10 ms TIMEDCOUNTER....11
Firmware: Infrarot Fernsteuerung....11
Anhang: GPIO#4 des RPi als Taktgenerator für den 328....14

Überblick

Zum Anschluss analoger Signale wie ein Potentiometer, von Termperatursensoren oder Abstandsmessern werden AD-Wandler benötigt.
Leider gibt es keine große Auswahl an Zusatzkarten für den RaspberryPi mit AD-Wandlern. Deshalb hier eine Anleitung, wie der Microcontroller Atmega328 auf dem Steckbrett in Betrieb genommen werden kann.

Beispielprogramme für den Raspberry Pi sind in python verfügbar. Integration mit Scratch ist mit der scratchClient-Software möglich von heppg.de.

Der 328-Prozessor hat viele eingebaute Funktionen, wie z.B. AD-Wandler, Timer, SPI und vieles anderes mehr.
Die Firmware ermöglicht weitere Funktionen wie Ereigniszählung, Frequenzmessungen oder den Anschluss eines IR-Fernbedienungs-Empfängers.

Der Prozessor ist im Hobbybereich sehr bekannt, da die Arduino-Boards mit diesem Prozessor arbeiten.
Die AD-Wandler haben eine ausreichende Auflösung mit 10 Bit. Der 328 kann einfach an den Raspberry Pi angeschlossen werden und das Programm im 328, die Firmware, kann vom RPi aus einprogrammiert werden.
Die Kosten für den Prozessor sind mit einigen Euro etwa so hoch wie für einen AD-Wandler. Je nach Firmware kann man aber noch andere Funktionen benutzen.

Nachteil ist die etwas komplizierte Inbetriebnahme. Die ist deshalb so lang geworden, da viele einzelne Schritte und Überprüfungen aufgenommen wurden.

Die Firmware, das interne Programm des 328-Prozessors ist in den Beispielen enthalten. Das Einspielen der Firmware erfolgt mit den RPi, es ist also kein extra Programmiergerät erforderlich.
Auf die Programmierung des Atmel controllers wird hier nicht eingegangen.

Voraussetzung ist, dass der Prozessor den internen 8MHz RC-Oszillator aktiviert hat. Das ist bei fabrikneuen Prozessoren der Fall.
Die Steuerung der Oszillatoren wird über 'Fuses' eingestellt. Damit kann man den interen Oszillator aktivieren, oder es werden Quartz-Oszillatoren oder eine externe Taktquelle aktiviert. Das Problem ist, dass ohne aktiven Oszillator das Programmieren mit dem RPI nicht funktioniert.

Falls man einen bereits programmierten Prozessor erhält, dann kann dieser den internen Oszillator abgeschaltet haben. In diesem Fall funktioniert bereits das Auslesen der 'Fuses' nicht.
Dann braucht man entweder ein spezielles Programmiergerät oder muss eine Taktquelle aktivieren. Im Anhang ist eine Prozedur hierfür beschrieben.

Setup

Einkaufsliste

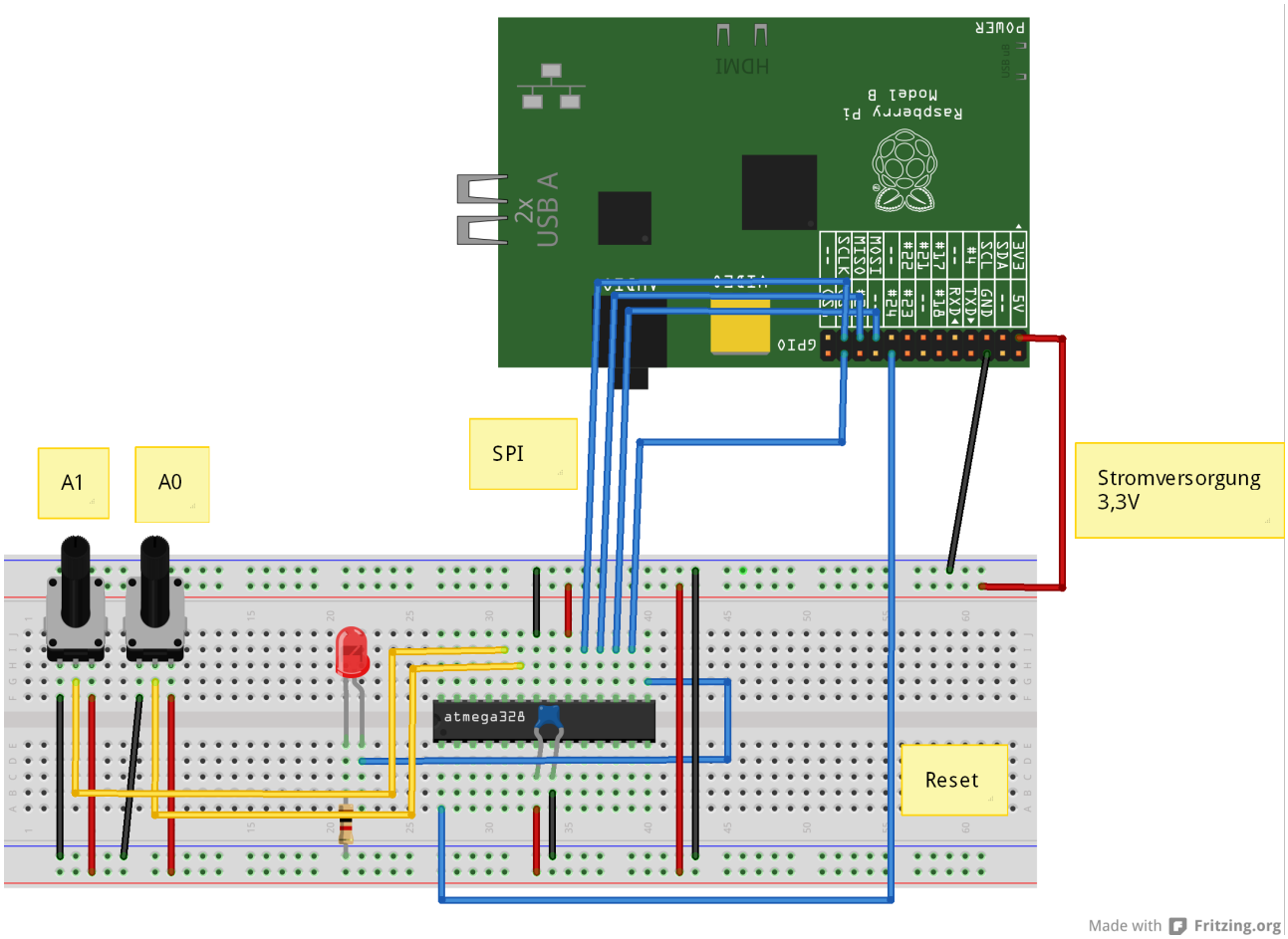
- ATmega 328 P-PU (DIL-Gehäuse)
- Steckbrett

- (optional) Fassung 28 pol, DIL 28 pol, 0.3 breit. Die Fassung ist für den Prozessor, damit die Beinchen nicht so leicht verbiegen. Fassung auf verbogene Kontakte kontrollieren (vorsichtig gerade biegen) und ins Steckbrett. Den 328 in die Fassung einsetzen. Die Füsse des Prozessors sind ab Fabrik nicht genau parallel ausgerichtet. Vorsichtig auf einer glatten Platte ausrichten.
- Leuchtdiode, standard 20mA
- Widerstand 1kOhm
- Kondensator 100nF, 10V min, keramisch
- Trimmpotentiometer zum Test
- Steckbrücken Buchse-Stecker, 5 Stück mindestens
- Kabelbrücken fürs Steckbrett

Ein RPi wird ebenfalls benötigt.

Aufbau

Strom des RPi abschalten.
Prozessor auf dem Steckbrett aufbauen.



VCC des Prozessors wird aus 3.3V des Raspberry versorgt (schwarz, rot).
SPI-Verbindung MISO, MOSI, SCK und SS an CS0 des RPI (balu).
RESET des Prozessors PIN 1 an GPIO24 (blau)
LED mit Serienwiderstand 1kOhm von Port PB1, Pin 15.
ADC-Eingänge sind ADC0 und ADC1, Pin 23, 24.

Eingangsspannungsbereich der AD-Wandler ist 0..3,3V.

Double check wiring.
Power on.

Software auf dem RPi installieren

```
sudo apt-get update
sudo apt-get upgrade -y
sudo apt-get install python-dev pip
sudo pip install spidev intelhex
```

Den SPI-Treiber aktivieren. Der ist über raspi-config zu starten, (enable SPI).
Oder über

```
sudo modprobe  spi_bcm2708
```

Programmiersoftware in /home/pi kopieren
Das Archiv auf dem RPi in /home/pi speichern und auspacken.

```
tar xzvf program_328.tar.gz
```

Überprüfen der Hardware und der Programme

Fuses auslesen

```
cd ~/program_328
sudo python src/program.py -rf
```

Die Ausgabe sollte folgendermassen aussehen

```
PROGRAMMING_READ_CALIBRATION_BYTE   b0 10110000
PROGRAMMING_READ_EXTENDED_FUSE_BITS ff 11111111
   BODLEVEL0          1
   BODLEVEL1          1
   BODLEVEL2          1
PROGRAMMING_READ_FUSE_BITS         e2 11100010
   CKSEL0             0 ENABLED
   CKSEL1             1
   CKSEL2             0 ENABLED
   CKSEL3             0 ENABLED
   SUT0              0 ENABLED
   SUT1              1
   CKOUT              1
   CKDIV8             0
PROGRAMMING_READ_FUSE_HIGH_BITS     d9 11011001
   BOOTRST            1
   BOOTSZ0            0 ENABLED
   BOOTSZ1            0 ENABLED
   EESAVE             1
   WDTON              1
   SPIEN             0 ENABLED
   DWEN               1
   RSTDISBL           1
PROGRAMMING_READ_LOCK_BITS         ff 11111111
```

Wenn es Fehlermeldungen gibt (Device not in sync), dann liegt ggf ein Schaltfehler vor, oder der Prozessor ist bereits mit unpassenden Fuses programmiert.
Dann erst mal nicht weitermachen. Verkabelung überprüfen.

Auslesen des aktuellen Programmes

Wenn die Ausgabe stimmt, dann kann man versuchen das aktuelle Programm auszulesen. Das ist bei einem neuen Prozessor noch leer. Hier dient das zu einer weiteren Überprüfung der Kommunikation.

```
cd ~/program_328
sudo python src/program.py -r
```

Die Ausgabe sollte wie folgt aussehen

```
root@raspberrypi:/home/pi/program_328# python src/program.py -r
('read', 'out.hex')
programming_readCode
programming_enable
('PROGRAMMING_ENABLE', [172, 83, 0, 0])
(0, [255, 255, 83, 0])
programming_enable end
programming_disable
programming_readCode ende
ok
```

Blink-Programm laden

Ist dieser Schritt erfolgreich, dann kann man das erste Testprogramm in den Prozessor laden. Es sollte die LED blinken lassen.

```
cd ~/program_328
sudo python src/program.py -p 328/steckbrett_328_blink.hex
```

Jetzt sollte die LED langsam blinken.

Programmieren der Fuses

Nach diesen Vorbereitungen kann man es wagen, die Fuses zu programmieren.

```
cd ~/program_328
sudo python src/program.py -wf
```

Das Blinken hört kurz auf, und sollte dann deutlich schneller wieder anfangen, 5 mal die Sekunde.
Jetzt ist im Prozessor der Teilerfaktor für den Takt abgeschaltet und der Prozessor läuft mit 8MHz.

Programmieren der Firmware

Jetzt kann das eigentliche Anwendungsprogramm geladen werden. Es sorgt für regelmässiges Auslesen der AD-Werte ADC0, ADC1.

```
cd ~/program_328
sudo python src/program.py -p 328/steckbrett_328.hex
```

Firmware Beschreibung

Die Firmware des Controllers bietet verschiedenen Funktionen, die jeweils aktiviert werden müssen.
Die Einstellungen sind nicht persistent, müssen also nach jeden Reboot oder Reset neu eingestellt werden.

Für jedes Feature gibt es die entsprechenden SET_CONFIG Befehle. Das Auslesen der Konfiguration erfolgt mit GET_CONFIG-Befehlen.

Die LED-Befehle sind ohne vorherige Aktivierung immer möglich.
Nach einem Reset blinkt die LED 8 mal.

Anmerkung zu den Kommandos an den Controller. Wenn eine Antwort erwartet wird, muss der host eine passende Anzahl von dummy-Bytes an den Controller schicken. Das wird durch die '0'-Bytes bei den Kommandos angegeben.

SPI-Taktrate ist 240000kHz für die 8MHz-Version der Firmware. Bei dieser Geschwindigkeit lässt die spidev-Bibliothek gerade noch genügend Zeit zwischen den einzelnen Zeichen und das Interrupt-Programm kann Rückgabewerte rechtzeitig bereitstellen. Bei grösserer Geschwindigkeit wird die Kommunikation nicht mehr zuverlässig funktionieren. Eine geringere Geschwindigkeit ist problemlos.

Firmware: Version auslesen

GET_VERSION,0,0	0x80	Der Controller antwortet mit der Versionsnummer.

Auslesen der Version des 328-Programms

```
cd ~/program_328
sudo python src/test_version.py
```

Die Anzeige sollte '0x93' sein, die zweite Zahl (minor version) kann z.B. 0x0C sein.

Firmware: LED-Kommandos

SET_LED_0	0x84	drive Port PD7 (Pin13) low.
SET_LED_1	0x88	drive Port PD7 (Pin13) high.

Testprogramm.
Die LED blinken lassen, lange hell und kurz dunkel.
Der Unterschied zum Blinkprogramm oben ist jetzt, dass der Controller Befehle vom Rpi entgegennimmt und ausführt.

```
cd ~/program_328
sudo python src/test_blink.py
```

Led on, off

```
cd ~/program_328
sudo python src/test_led_on.py
sudo python src/test_led_off.py
```

Firmware: Lesen der ADC Werte

Es werden zwei ADC-Kanäle unterstützt.

ADC-Conversions müssen eingeschaltet werden.

GET_ADC_0,0,0	0x81	get MSB,LSB ADC 0

GET_ADC_1,0,0	0x82	get MSB,LSB ADC 1
GET_ADC_CONFIG,0,0	0x91	Konfiguration der AD-Wandler auslesen Default ist 0x03, 0x03 für AD0, AD1
SET_ADC_CONFIG, x, x	0x92	Konfiguration der AD-Wandler schreiben. Erstes Byte AD0, zweites Byte AD1 Bit 1 gesetzt: Auflösung 3.3V Bit 1 gelöscht: Auflösung 1.1V Bit 0 gesetzt: aktiv Bit 0 gelöscht: inaktiv

Auslesen des adc_0-Wertes. Das Programm liest einige hundert Werte aus und zeigt diese auf der Konsole an.

```
cd ~/program_328

sudo python src/test_adc_0.py
```

Ist kein Signal angeschlossen, wird der Wert '0' angezeigt. Oder auch mal sehr kleine Werte, da immer etwas Rauschen im Spiel ist.

ADC, Anbindung an Scratch

Wenn die Tests funktionieren, kann der Aufbau mit Scratch in Betrieb genommen werden. Herunterladen und installieren der scratchClient-Software von heppg.de.

```
cd ~/scratchClient
sudo python src/scratchClient.py -config config/328_steckbrett.xml
```

Die Werte der AD-Wandler werden als 'adc_0' und 'adc_1' an Scratch geschickt, Wertebereich 0..1023.
Befehle von Scratch an den Controller sind 'led_0_off', 'led_0_on'.

Die Konfiguration der Kanäle, Auflösung und polling-Rate wird in der Konfigurationsdatei vorgenommen.

Firmware: Ereigniszähler, Frequenzmessung

Die verschiednen Funktionen zur ereigniszählung oder zur Frequenzmessung schliessen sich gegenseitig aus, da diese alle auf dem 16-Bit-Zähler Timer1 beruhen.

Die Funktionen benutzen entweder T1 oder ICP1 des Controllers, Pin 11, 14. Es wird empfohlen, diese Pins miteinander zu verbinden. Das erlaubt grössere Flexibilität für die Software.

Bei der TIMER-Funktion dürfen die Eingangsfrequenzen nicht sehr hoch sein, da sonst der Prozessor die Interrupts nicht mehr abarbeiten kann. Bis 20kHz sollte das funktionieren.

Um einen grossen Frequenzbereich messen zu können empfiehlt sich eine Steuerung host-seitig.

Hierzu muss T1 und ICP1 verbunden sein.
Bestimme die Frequenz grob mit den COUNTER und zwei Messungen im Abstand von ca 1 sec. Bei Frequenzen > 20kHz sollte die COUNTER-Funktion verwendet werden.
Bei Frequenzen > 122 Hz und < 20kHz ist die TIMER-Funktion geeignet.
Frequenzen < 122 Hz können nicht mehr im Messfenster 10ms gemessen werden, hier sind die Messdauern grösser.

Firmware: Ereigniszählung COUNTER

Der Controller kann Ereignisse zählen. Der interne Timer1, 16 Bit wird dazu um zwei Überlaufregister ergänzt. Es stehen also ein 32Bit Zähler zur Verfügung.
Input Pin is T1.

Das Auslesen der Zählwerte löscht das Register nicht.
Die Ereigniszählung belastet den Controller kaum, da nur selten ein Überlauf den Timer1 auftritt.

Firmware: Frequenzmessung TIMER

Der Controller kann Frequenzen bestimmen.
Anwendbar für Frequencen grösser 200Hz, kleiner 20 kHz.

Es wird ICP1 als Eingangspin verwendet.
Die Messung erfolgt mit der Capture-Funktion des Timer1.

```
N = 4
Wiederhole fortlaufend

      Setze Timer 1 zurück
      Warte auf steigende Flanke des Mess-Signals, Bestimme t0-Wert
      Warte N Perioden ab, bestimme TN-Wert.
      Wenn Overflow aufgetreten ist, verwerfe die Messung und verkleinere N
      Speichere den Messwert und N für die Kommunikation zum Host.
      Wenn Messwert > 0xF000, verkleinere N
      Wenn Messwert < 0xD000, vergrössere N
```

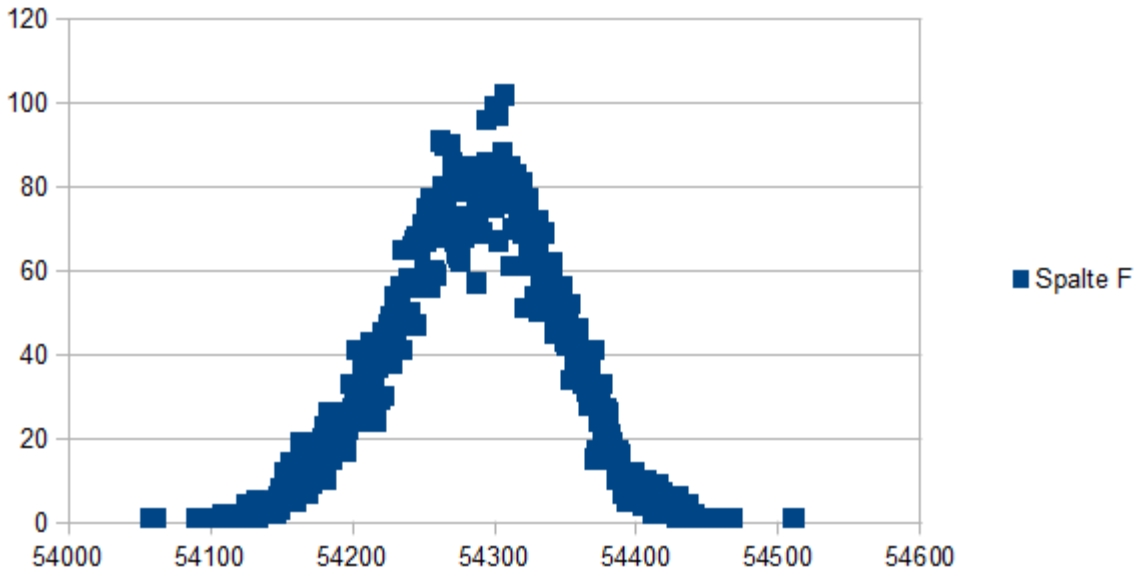
Der Timer1 benutzt die Taktfrequenz zur Messung. Bei 8MHz dauert eine Messung dann 0xffff/8MHz = 8,2ms.
An den Host wird der Wert ((tN-t0), 16Bit und der Wert N übergeben.

Bestimmung der Frequenz:
```
f = 8000000.0 *period / value
```

Überprüfen der Genauigkeit

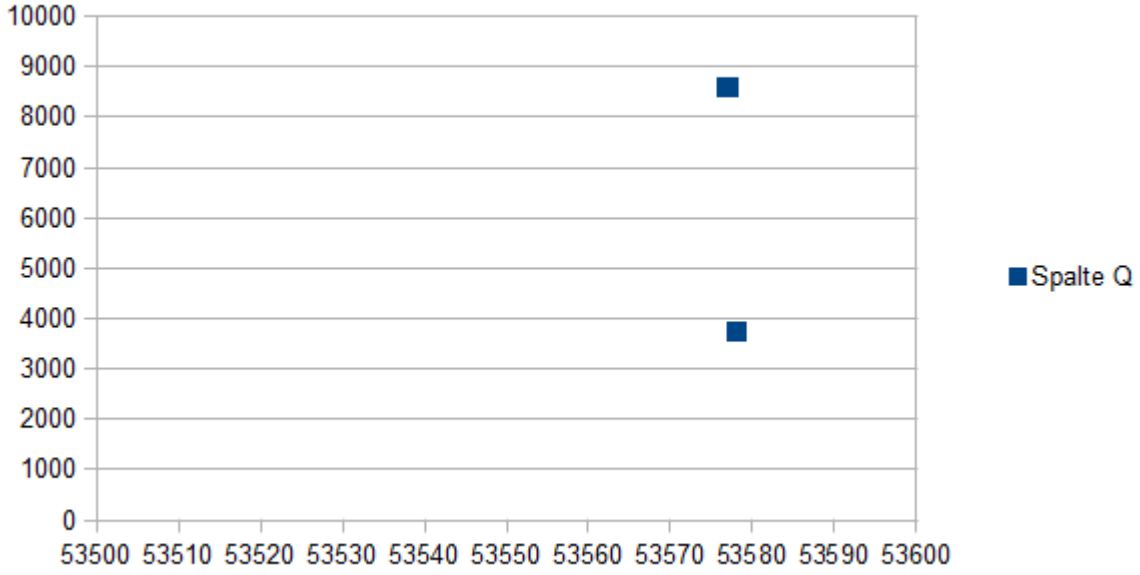
Ich wollte herausfinden wie genau die Messungen sind und überprüfen ob es Fehler in der Firmware gibt. Hier die Ergebnisse.

Die Messgrundlage for die Frequenzmessung ist die Zeitgenauigkeit der internen Oszillators.
Zur Bestimmung der Genauigkeit wurde eine präzise Eingangsfrequenz verwendet und etwa 10000 Messungen ausgeführt. Trägt man die verschiedenen ermittelten Werte mit den Häufigkeiten auf, so erging sich das folgende Bild.der mit den Wertestreuungen.



Die Grafik zeigt die aufsummierten Periodenzeiten für 90 Eingangsperioden, insgesamt 10000 Messungen. Die Streuung ergibt sich durch die Ungenauigkeit des internen RC-Oszillators. Die Genauigkeit ist also ca +- 2 Promille.

Wird statt des internen Oszillators ein Quartzoszillator verwendet ist die Streuung sehr gering, praktisch ergeben sich nur zwei Werte, die sich durch ein Bit unterscheiden.



Bitte die Skalierung beachten. Die beiden Messpunkte entsprechen zusammen über 10000 Messungen.

SPI-Befehle für die Frequenzmessung der Version 0x93 sind

`GET_TIMER, 0, 0, 0, 0`	0x87	Frequenzmessung auslesen. Byte [1]: Messung high Byte [2]: Messung low Byte [3]: Anzahl Signalperioden in der Messung Byte [4]: Fehler, 0 = ok

SET_COUNTER_CONFIG, 0, 0	0x85	Byte[1].0 Zähler benutzen Byte[1].1 Zähler starten Byte[1].2 Zähler auf 0 setzen Byte[1].3 Frequenzmessung benutzen Byte[1].4 Noise canceller für Timer oder Zähler einschalten, empfohlen..
GET_COUNTER_CONFIG, 0, 0	0x86	Liest die aktuelle Konfiguration

Firmware: Frequenzmessung mit einer Zeitbasis 10 ms, 20ms TIMEDCOUNTER

Für höherfrequente Signale als ca 50kHz ist die Messung mit einer internen Zeitbasis geeignet.
Es wird die Anzahl der Ereignisse innerhalb eines Zeitfensters von 10ms oder 20ms ermittelt.
Precisely, the 10ms is more like 9,98ms.
Bei 16Bit werden theoretisch Werte bis 6.5MHz gemessen.

SPI-Befehle für die Frequenzmessung der Version 0x93 sind

GET_TIMEDCOUNTER, 0, 0	0x89	Frequenzmessung auslesen. Byte [1]: Messung high Byte [2]: Messung low
SET_COUNTER_CONFIG, 0, 0	0x85	Byte[1].4 Noise canceller für Timer oder Zähler einschalten, empfohlen.. Byte[1].5 Frequenzmessung einschalten. Byte[1].6 0: 10ms Zeitfenster 1: 20ms Zeitfenster
GET_COUNTER_CONFIG, 0, 0	0x86	Liest die aktuelle Konfiguration

Die Messergebnisse sind die Anzahl Events innerhalb des Zeitfensters von 10ms bzw. 20ms. Die Genauigkeit wird durch die Genauigkeit des Oszillators begrenzt und durch ggf. während der Messung vorkommende (Kommunikations-) Interrupts während der Messung.

Firmware: Infrarot Fernsteuerung

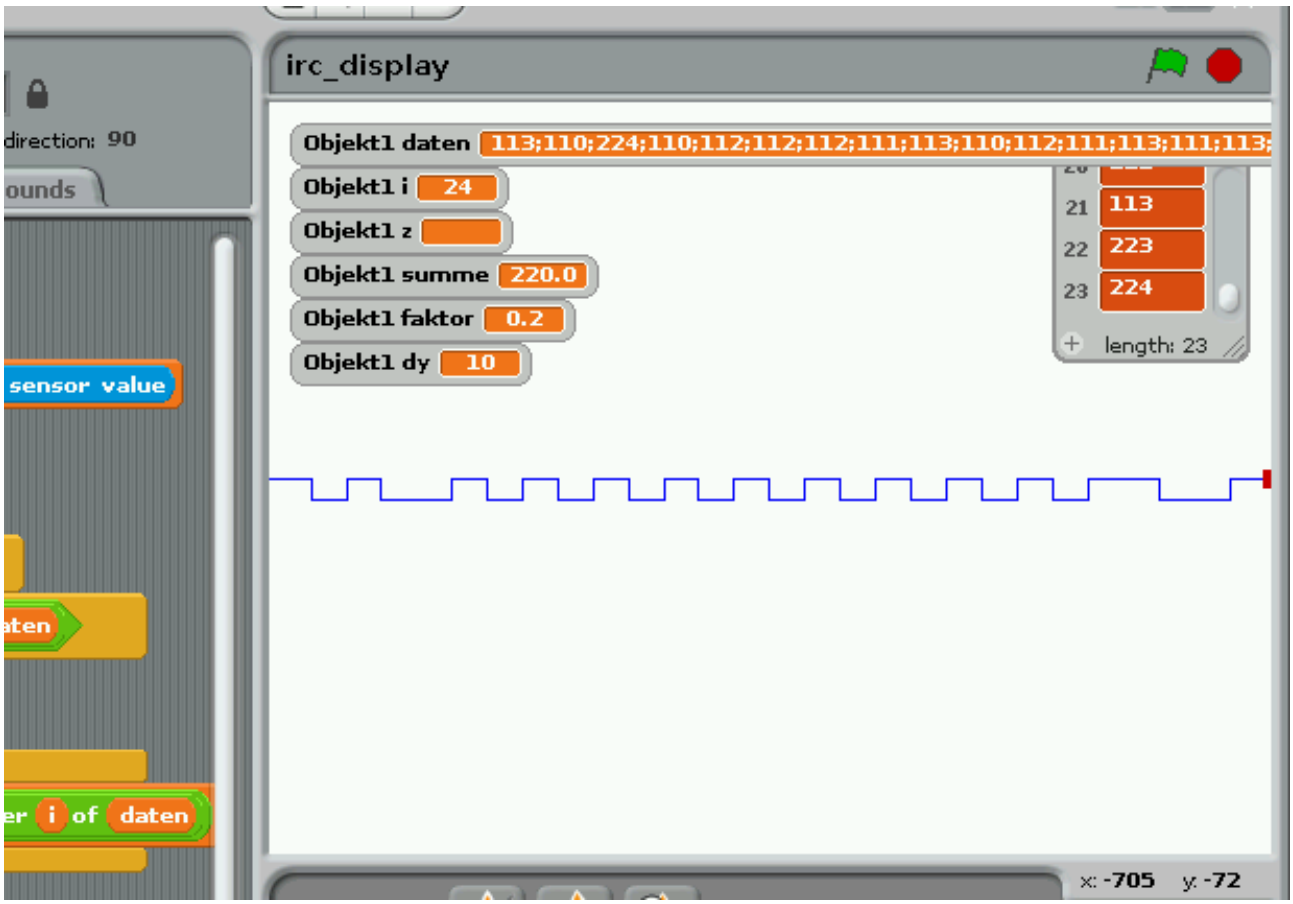
Fernsteuersignale können erfasst werden.
Ein TSOP34838 an PD6, PIN 12 empfängt die Signale einer IR-Fernsteuerung.

Der Controller decodiert die Signale nicht selber, sondern misst die Zeiten von Signalflanke zu Signalflanke und der Host kann diese auslesen und auswerten.
Die Funktion wird aktiviert über das Setzen der Konfiguration.
Ist eine Signalerfassung komplett, so meldet der Kontroller über PD4 an den GPIO#23 (active low) das an den Host.
Man kann auch das Statussignal periodisch auslesen, um festzustellen ob ein Signal verfügbar ist.
Der Start der Aufzeichnung beginnt mit einer negativen Flanke (Flanke 0).
Es werden maximal 128 Flanken berücksichtigt, jeweils 16 Bit, die Auflösung (beim 8MHz-Prozessor) ist 8us. Werden nach 12,3 ms keine Flanken mehr festgestellt, so wird die Sequenz als beendet betrachtet. Typische Pausen zwischen Zeichen sind etwa 60 bis 100 ms.

`SET_IRC_CONFIG, 0,`	0x8A	Konfiguration einstellen. Byte [1].0: 1 = enable Byte [1].1: 1 Messung starten
`GET_IRC_DATA_0, 0, 0..(64 Bytes)`	0x8B	Block 0 der Ergebnisdaten Byte[1] 1. Flanke MSB Byte[2] 1. Flanke LSB Byte[3] 2, Flanke MSB Byte[4] 2. Flanke LSB
`GET_IRC_DATA_1, 0, 0 (64 Bytes)`	0x8C	Block 1 der Ergebnisdaten
`GET_IRC_DATA_2, 0, 0 (64 Bytes)`	0x8D	Block 2 der Ergebnisdaten
`GET_IRC_DATA_3, 0, 0 (64 Bytes)`	0x8E	Block 3 der Ergebnisdaten
`GET_IRC_STATUS, 0, 0, 0`	0x8F	Status auslesen. Alle Datenbytes '0': keine Daten verfügbar Byte 1: Anzahl Messwerte, MSB Byte 2: Anzahl Messwerte, LSB Byte 3: 2 == Zeichenfolge erkannt 1 == Maximale Anzahl erreicht, Sequenz u.U. unvollständig.

Das Testprogramm test_get_irc.py startet eine Erfassung, und versucht die Daten anschliessend als RC5 zu interpretieren. Ältere Philips-Fernbedienungen benutzen dieses Protokoll. Bei nicht passenden Fernbedienungen wird es Fehlermeldungen geben.

Es gibt ein scratch-Programm zusammen mit dem scratchClient, welches die Signale grafisch darstellt.



Das Beispiel ist in der scratchClient-Distribution enthalten.
Konfiguration: /scratchClient/scratch/irc_atmel328/config_irc_atmel328.xml
Scratch-Programm /scratchClient/scratch/irc_atmel328/irc_atmel328.sb

In meinem Aufbau verwendet ich den TSOP34838, da er von meinem Händler verfügbar war. Es gibt viele vergleichbare IC. Die Frequenz(38kHz), Spannung (3,3V, also z.B. 2,5 bis 5V) muss passen; die Anschlüsse mit dem Datenblatt überprüfen.
Die orange eingezeichnete Verbindung von pin6 zu GPIO#23 ist über einen 1kOhm-Widerstand angeschlossen. Dieser sorgt dafür, dass wenn GPIO#23 aus Versehen als Ausgang konfiguriert wird, dann keine Schäden entstehen.

Anhang: GPIO#4 des RPi als Taktgenerator für den 328

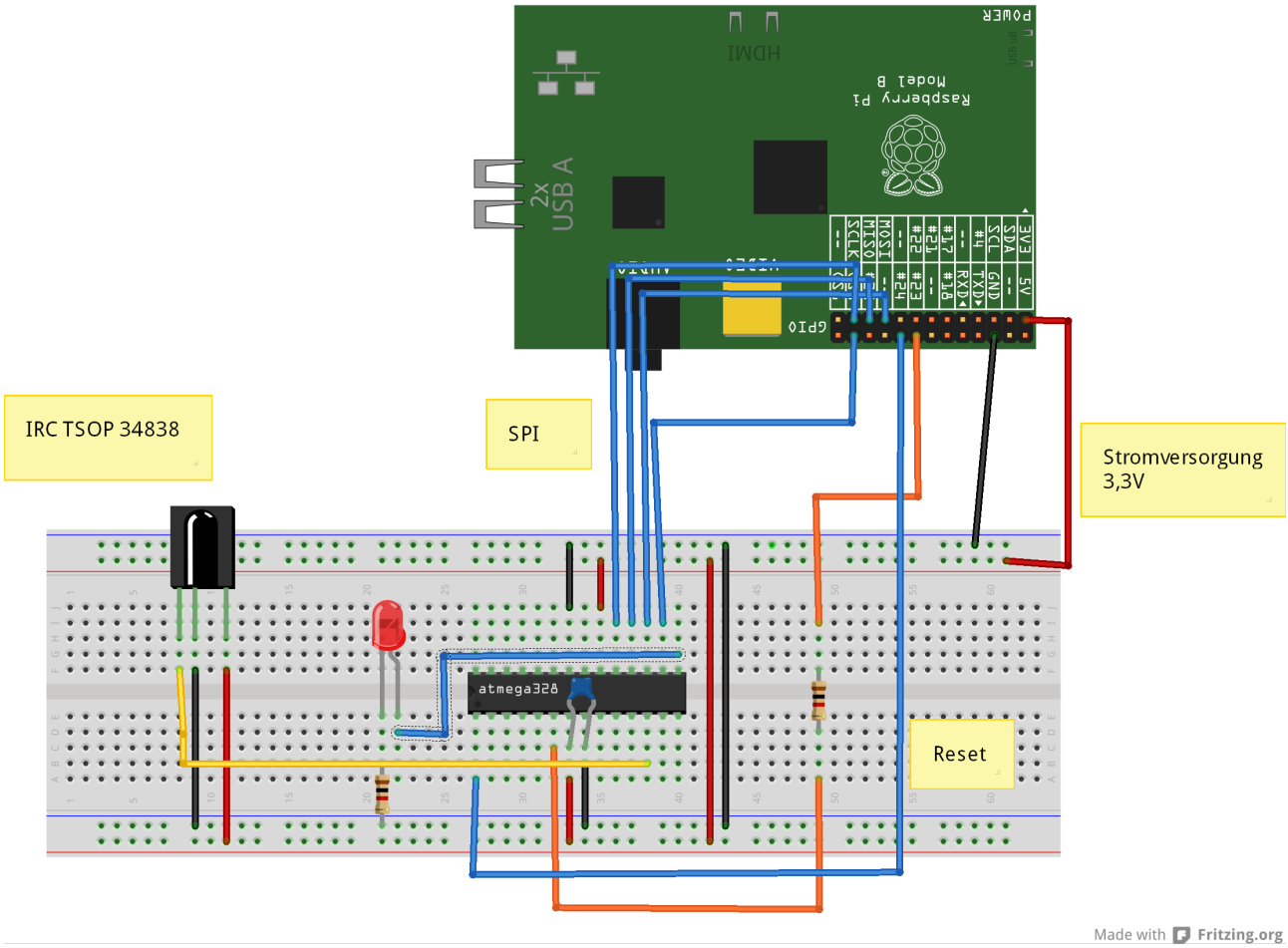
Der GPIO#4-Pin kann als Taktausgang konfiguriert werden. Das dafür notwendige Programm ist im bin/-Verzeichnis enthalten, Es basiert auf Code von guzunty.org.

GPIO#4 mit den XTAL1-Eingang des Controllers verbinden, Pin 9.

Dann das Programm starten, welches den Taktausgang aktiviert. Das erledigt man in einem zweiten Terminal.

```
cd ~/program_328/bin
chmod +x gz_clock_1.92MHz
sudo ./gz_clock_1.92MHz
```

Die Ausgabefrequenz ist ca 1.92MHz.



Während das Programm läuft, das Auslesen und Schreiben der Fuses ausführen.
Dann die Verbindung von GPIO#4 zum Prozessor wieder trennen und das Programm beenden.