Using GPIOs of Raspberry Pi in pure Tcl - my way

In the recent years the Raspberry Pi has become a popular low-cost mini-computer. When you run Debian-Wheezy-Linux on it, Tcl is already on-board.

Scripting in pure Tcl using simple commands like "open, puts, read, close" with /sys/class/gpio and /proc/interrupts, you can already toggle the GPIO lines several hundred times per seconds or count impulses. This is enough for a lot of uses of this little computer. Via WLAN interface and a small server script you can also load, store and source Tcl scripts without being directly connected.

There is already a library written in pure Tcl published by Gerhard Reithofer. In my talk instead, I want to demonstrate my way of programming the GPIO-lines of the Raspberry Pi

Uli Ender (uli@esperanto-muenchen.de)

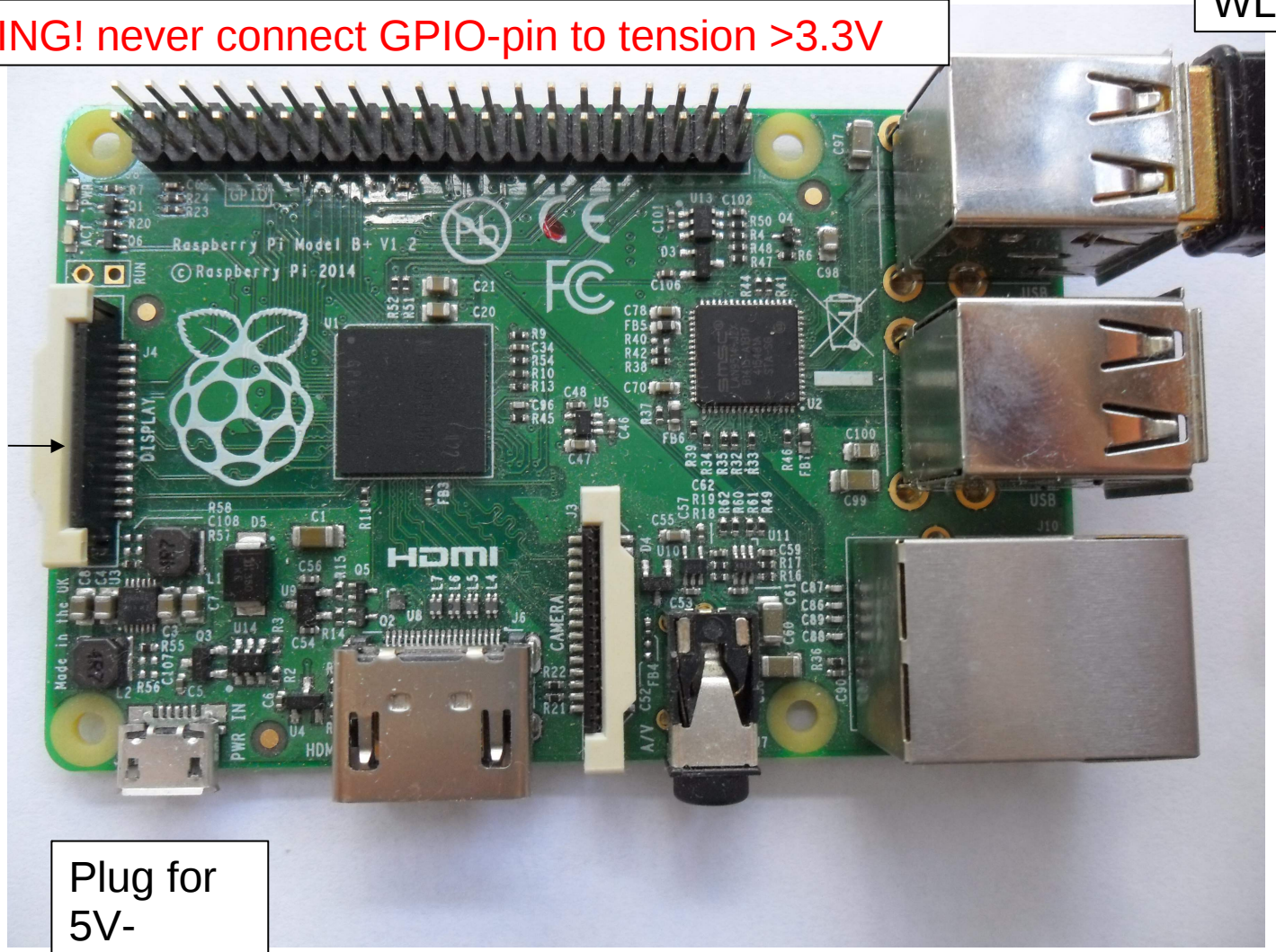
Raspberry Pi B+

GPIO-Port

WARNING! never connect GPIO-pin to tension >3.3V

WLAN-Stick

SD-Card With Raspbian (Linux)



Plug for 5V-power supply

Installation of necessary software

Transfer Raspbian from DVD to SD-card:

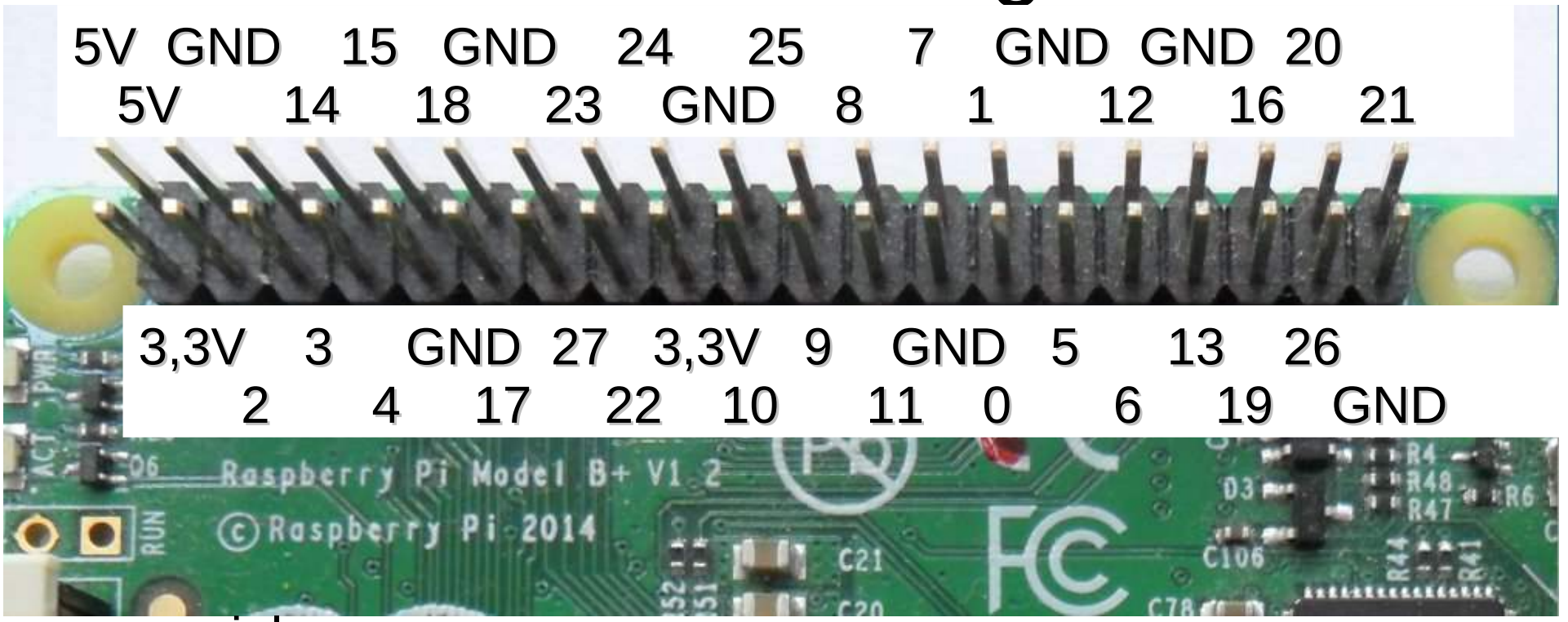
```
unzip 2015-05-05-raspbian-wheezy.zip
dd if=2015-05-05-raspbian-wheezy.img of=/dev/sdc bs=1M
```

to directly start via WLAN:

- rename or delete /etc/profile.d/raspi-config.sh
- configure WLAN in /etc/network/interfaces
- start HTML-server in /etc/rc.local

```
#! /bin/sh -e
/usr/bin/tclsh /home/pi/servilo.tcl &
exit 0
```

Part I GPIOs in general



some special uses

pin	use	pin	use
14	TxD	2	SDA
15	RxD	3	SCL
		4	1-wire-Temperature-sensor

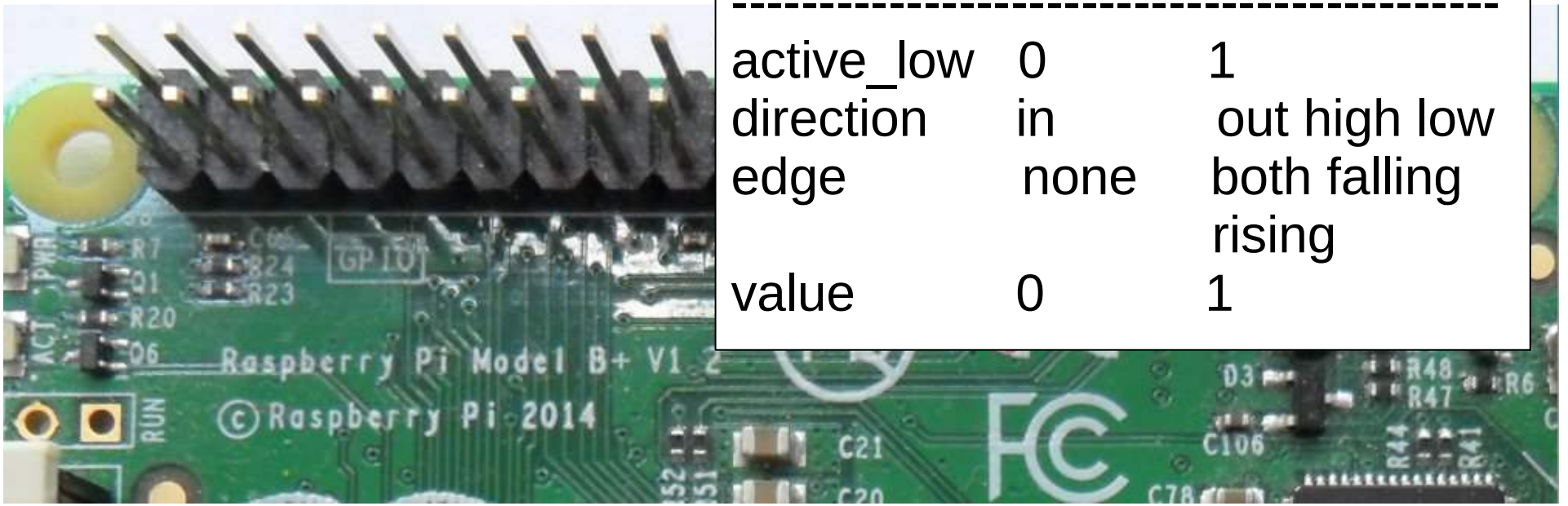
exec ls /sys/class/gpio ==> export gpiochip0 unexport

```
set h [open /sys/class/gpio/export w]
puts $h 23
close $h
```

exec ls /sys/class/gpio ==> export gpio23 gpiochip0 unexport
exec ls /sys/class/gpio/gpio23 ==> active_low device direction edge subsystem uevent value

23

file	default	alternative(s)
active_low	0	1
direction	in	out high low
edge	none	both falling rising
value	0	1



Demo HTML-server

---

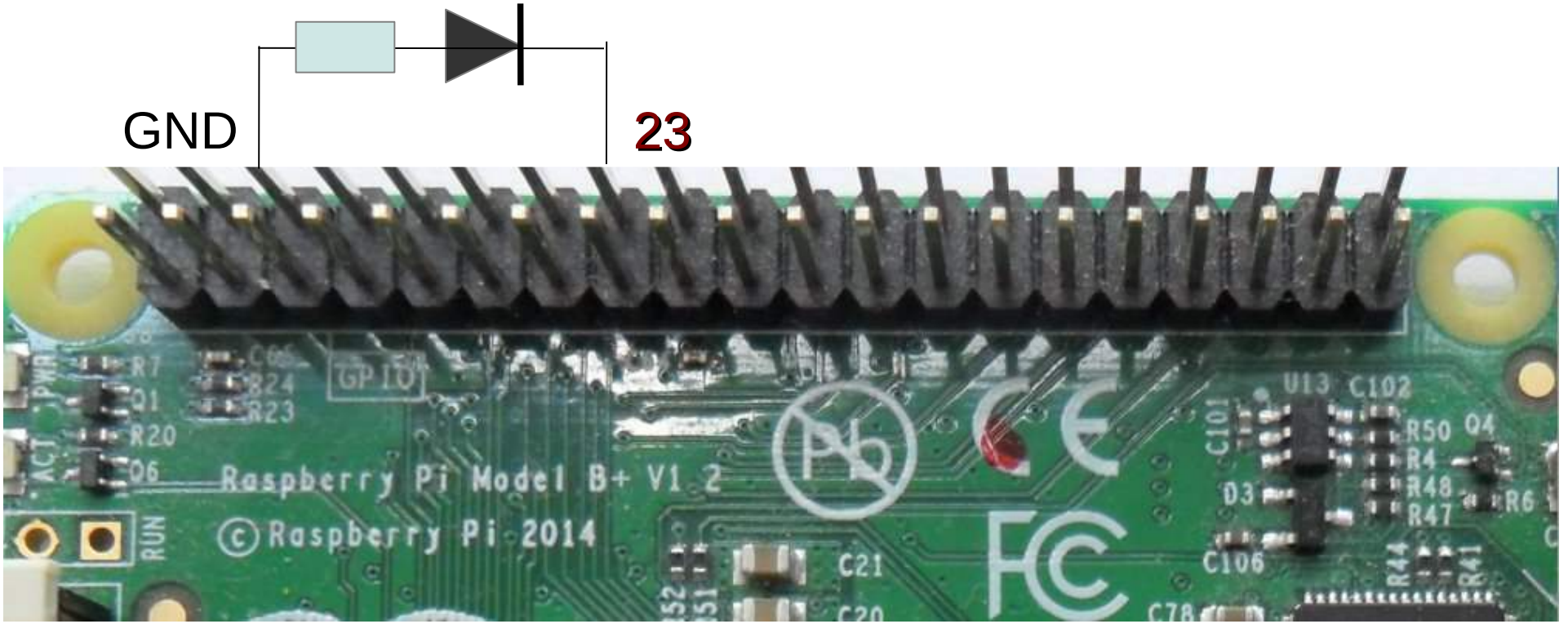
exec ls -l list files

l help list helpfile

l starting prepare GPIO 22 as input
GPIO 23,24,25 as outputs

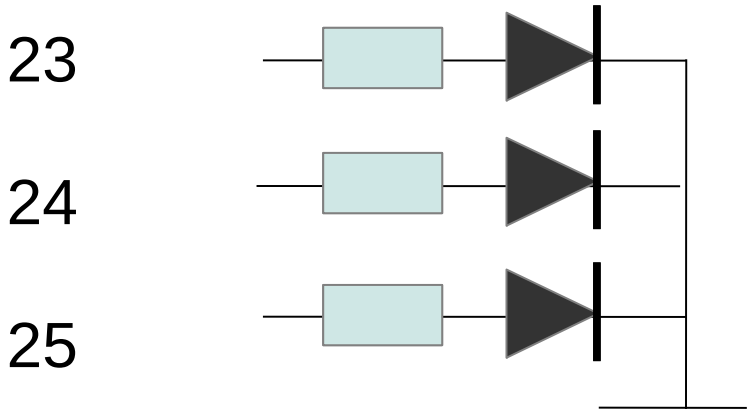
e starting

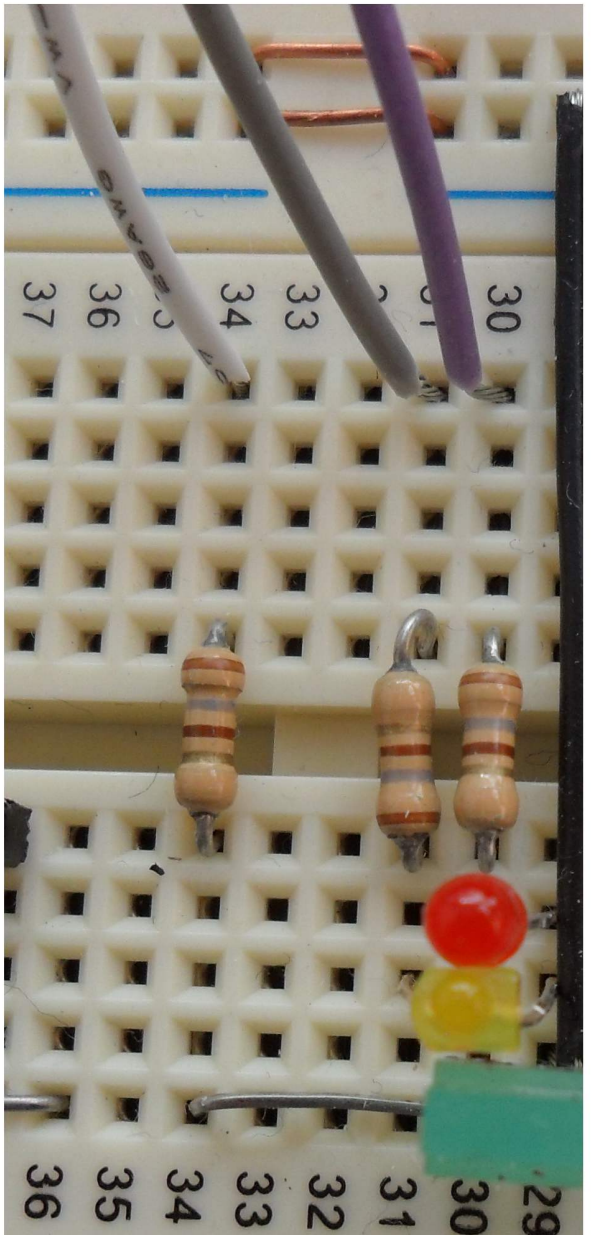
```
set h [open /sys/class/gpio/gpio23/direction  w]
puts $h out
close $h

set h [open /sys/class/gpio/gpio23/value w]
puts $h 1
flush $h                                          ==> LED on
after 10000
puts $h 0
close $h                                          ==> LED off
```



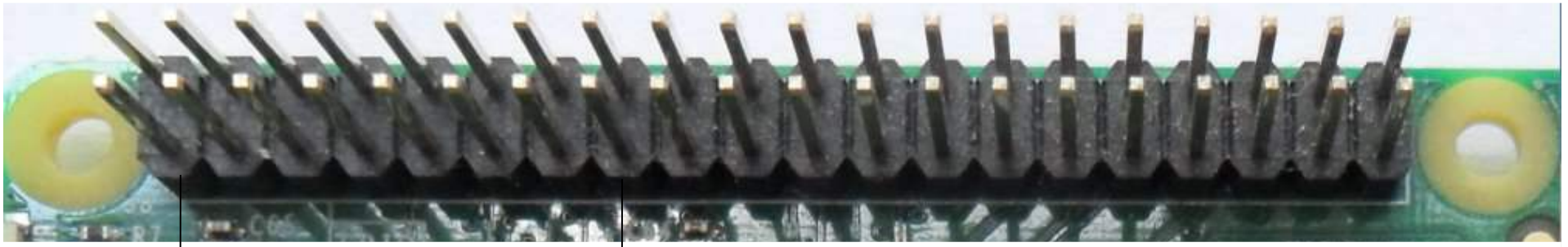
» DEMO OUTPUTS
» ----------

red-on
red-off
yellow-on
yellow-off
green-on
Green-off



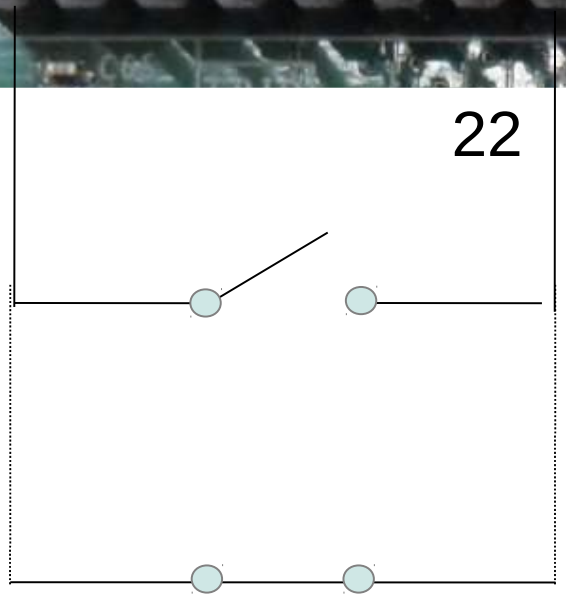


```
set h [open /sys/class/gpio/export w]
puts $h 22
close $h
after 1000

set h [open /sys/class/gpio/gplo22/value r]
set x [read $h 1]
close $h
```



3,3V

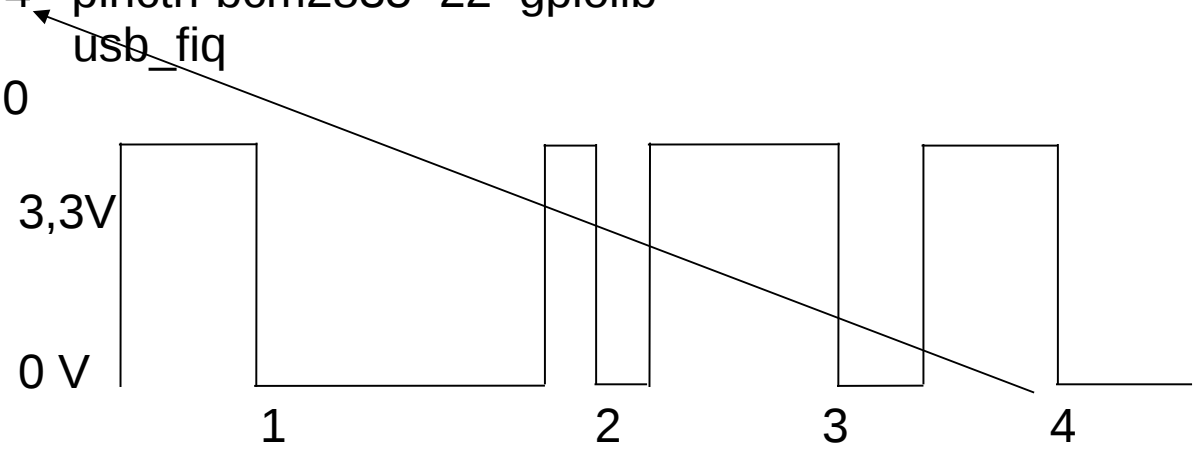
22



$x = 0

$x = 1

```
set h [open /sys/class/gpio/gpio22/edge  w]
puts $h falling
close $h

exec cat /proc/interrupts      ==>


           CPU0
 3:      11916   ARMCTRL   3  BCM2708 Timer Tick
16:          0   ARMCTRL  16  bcm2708_fb dma
24:        162   ARMCTRL  24  DMA IRQ
25:       1530   ARMCTRL  25  DMA IRQ
32:     154725   ARMCTRL  32  dwc_otg, dwc_otg_pcd, dwc_otg_hcd:usb1
49:          4   ARMCTRL  49  20200000.gpio:bank0
50:          0   ARMCTRL  50  20200000.gpio:bank1
65:         10   ARMCTRL  65  ARM Mailbox IRQ
66:          2   ARMCTRL  66  VCHIQ doorbell
75:          1   ARMCTRL  75
83:          4   ARMCTRL  83  uart-pl011
84:       5729   ARMCTRL  84  mmc0
416:         4   pinctrl-bcm2835  22  gpiolib
FIQ:             usb_fiq
Err:         0
```



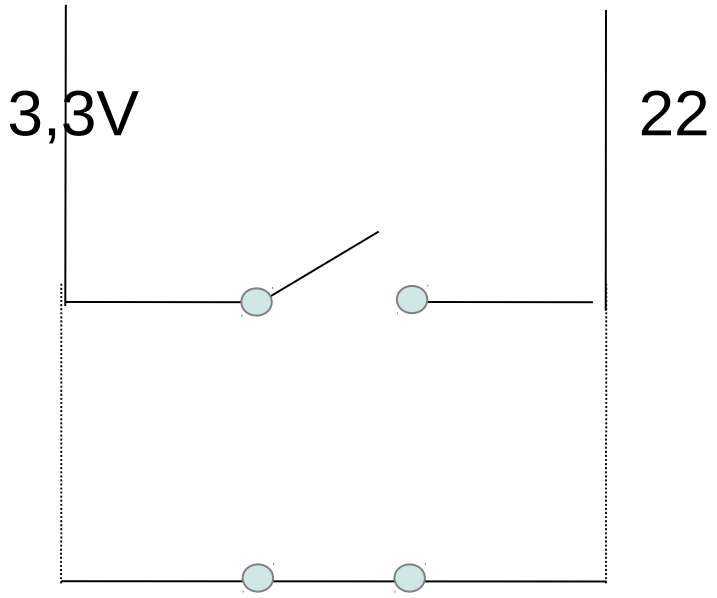
DEMO INPUTS

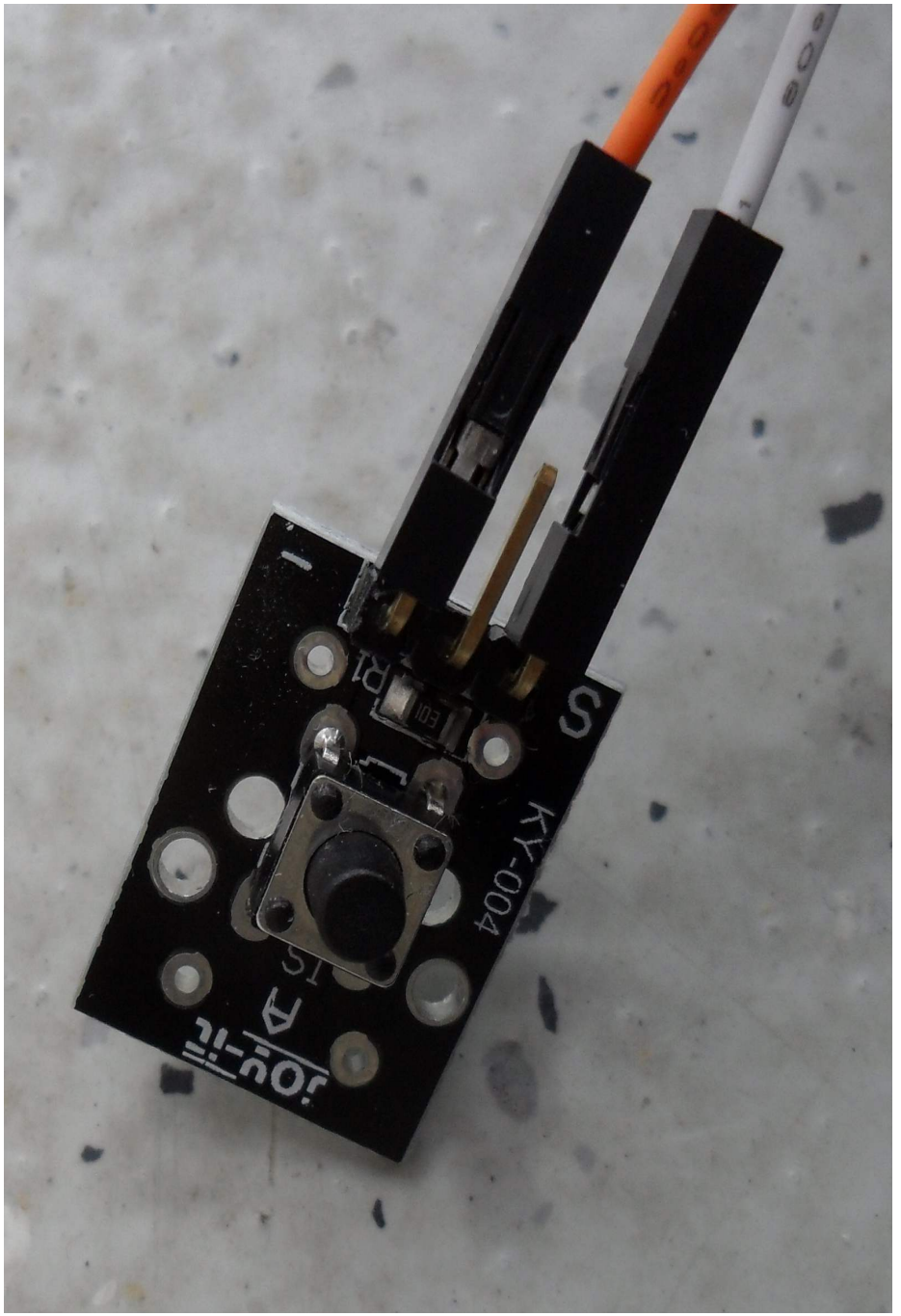
---

pressed

DEMO INTERRUPTS

inter



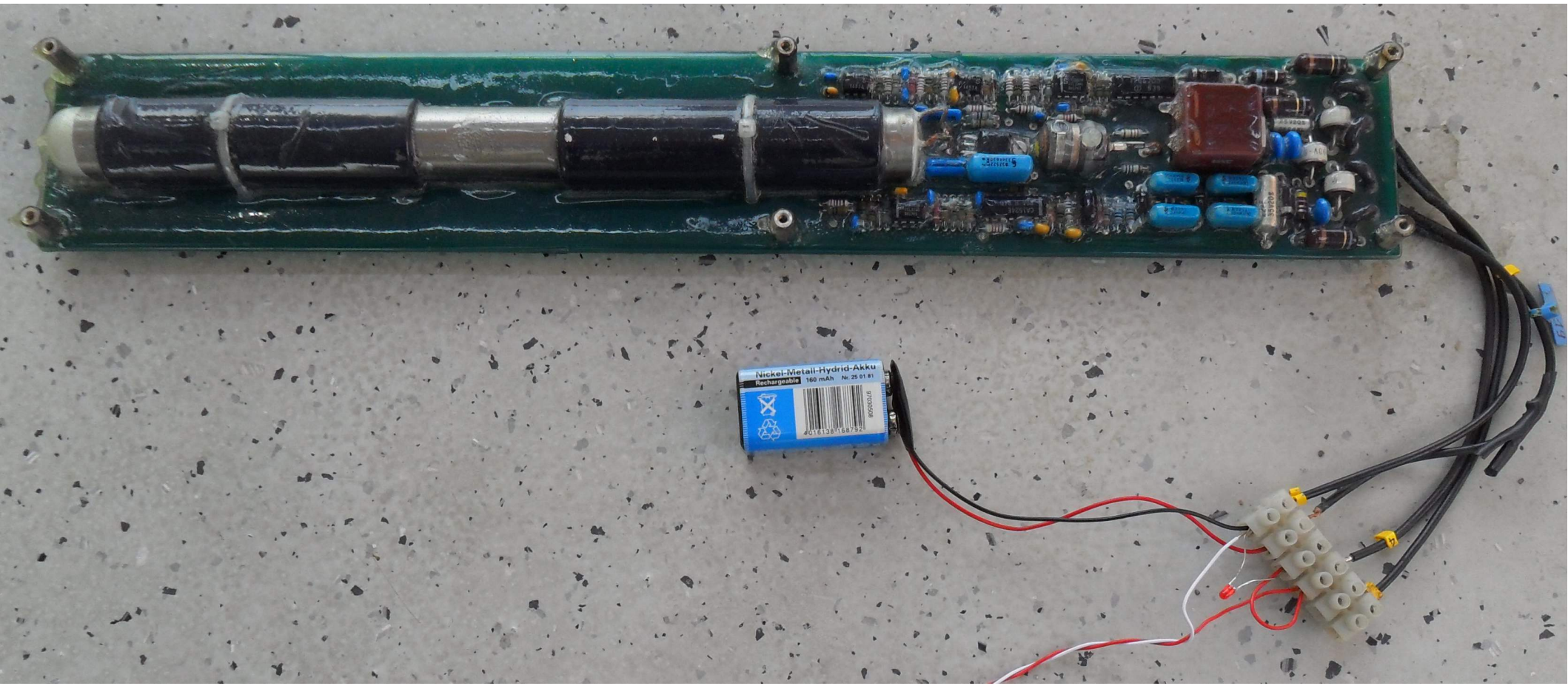


Example USE of INTERRUPTS

Counting Gamma-radiation with a Geiger-counter

Inter

White line GND
Red line GPIO-22



max. speed (Model B+ 700 Mhz)

DEMO SPEED schnellst with and without interrupts

write to GPIO-pin

---

puts $h 1

flush $h

(Inside a proc script < 1k
flush not seek used)

==> 25 µsec 40000 per sec

read value and store to file

---

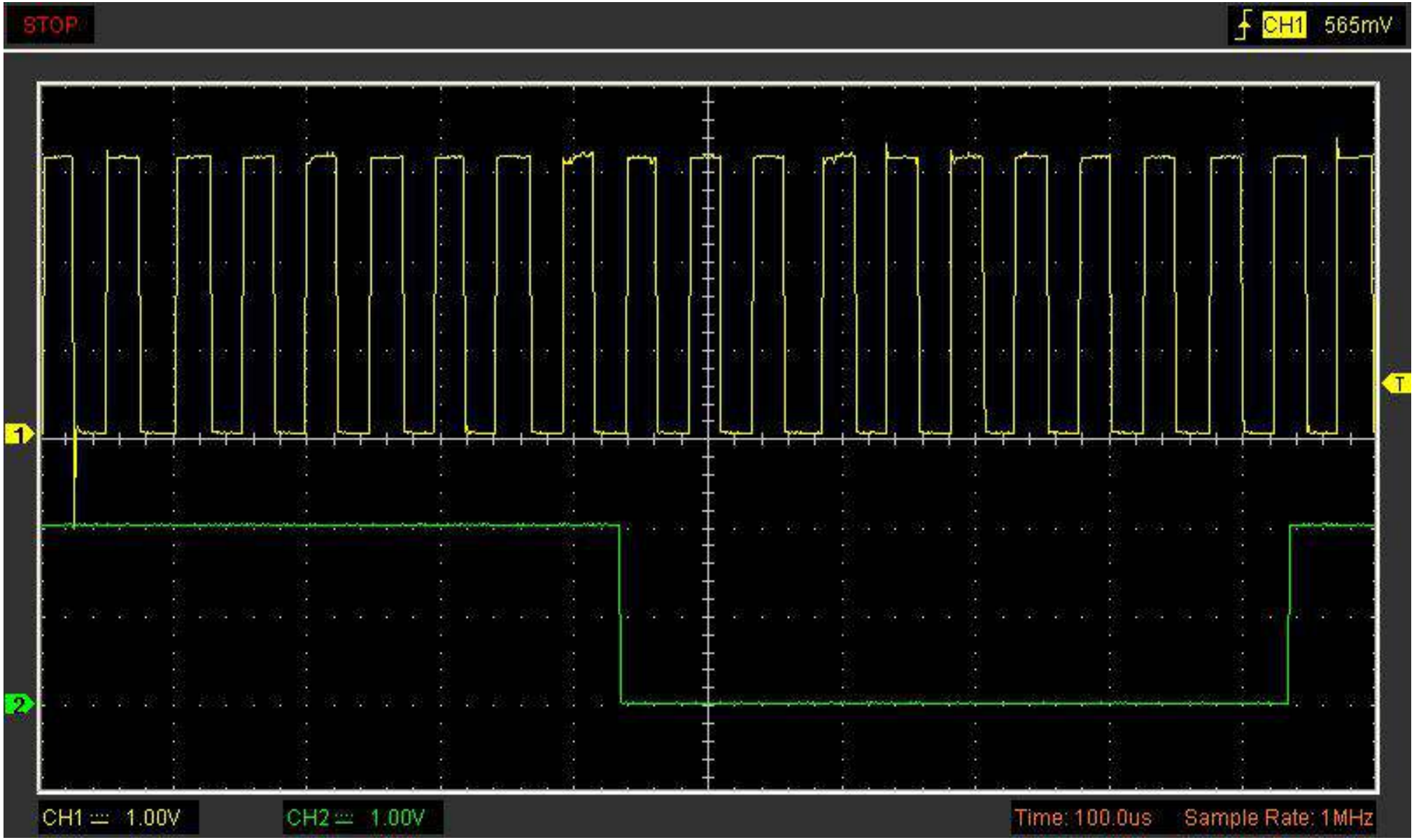
set x [read $h]

flush $h

puts -nonewline $hh $x

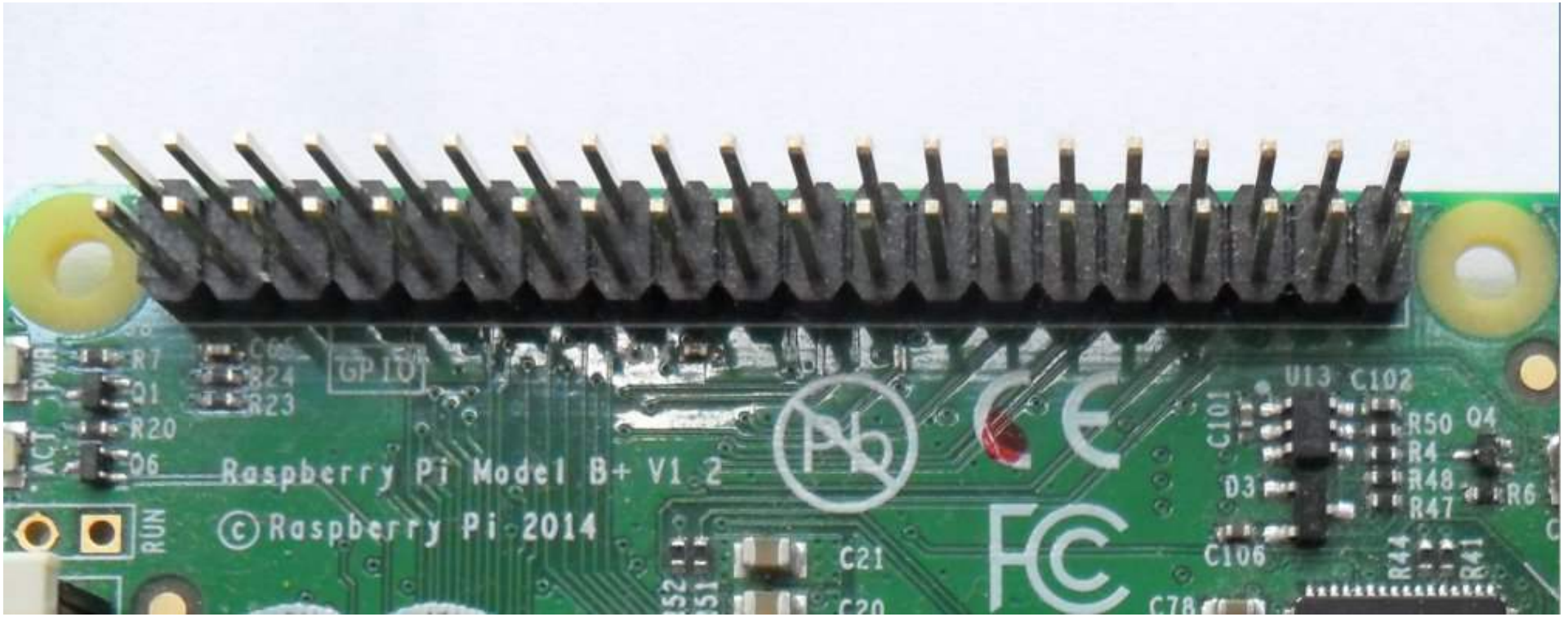
==> 125 µsec 8000 per sec

approx 20kHz wave generated by TCL compared to 1kHz calibration output of my oscilloscope



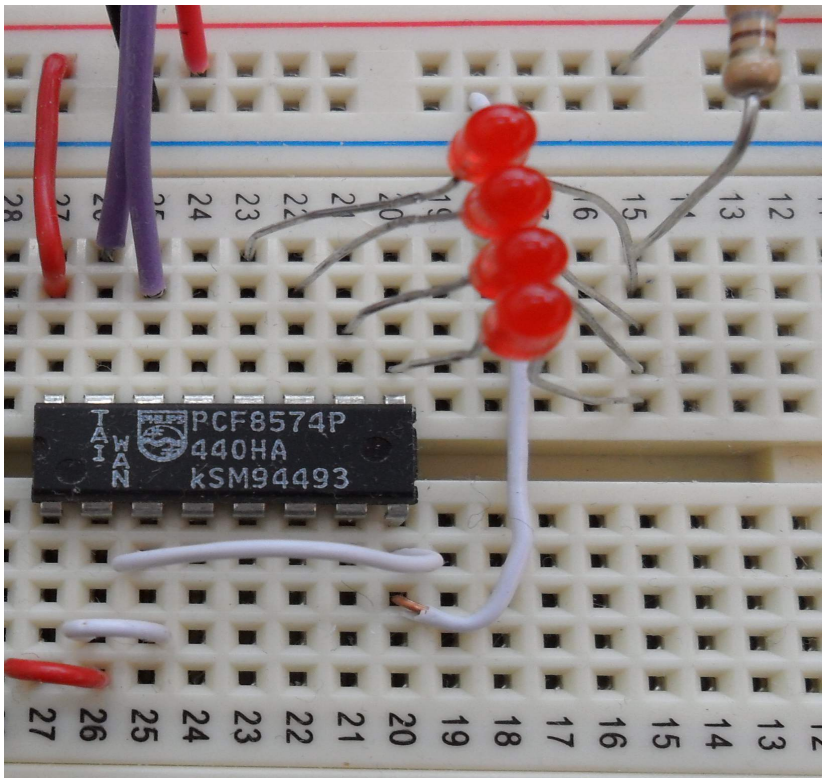
PART II

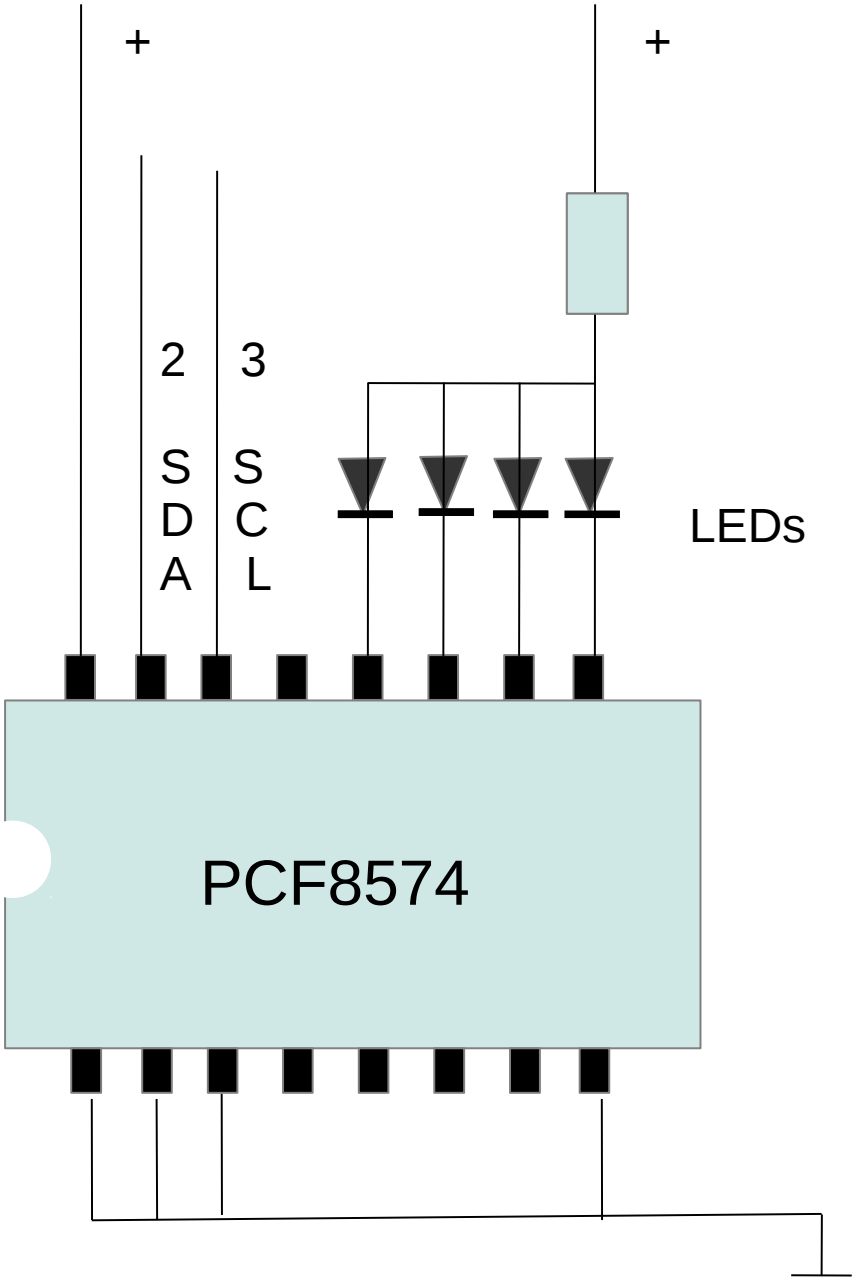
I2C-bus 8bit-port expander - LCD-module
SPI-bus programming ATtiny45 microcontroller



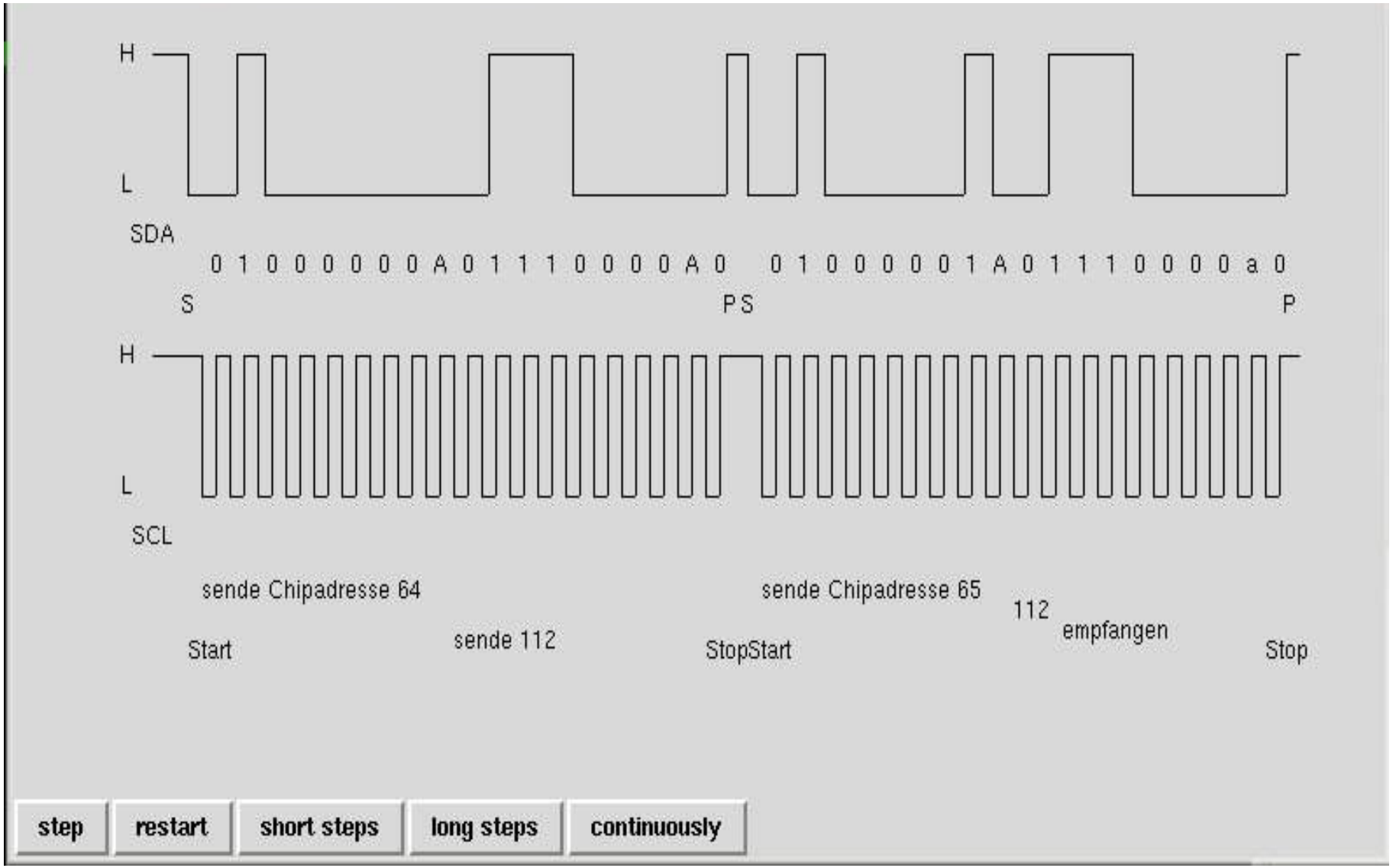
Preparation for i2c demo programs
On RASPI
- starting-i2c
- server8888

On your computer
wish i2c-demo





I2C-communication starts with a start-command (S) and ends with a stop-command (P) chipadress is sent first then the data-bytes
A bit becomes valid on rising clock
8th bit is the read/write-signal 9th bit is the aknowlege-bit (A,a)



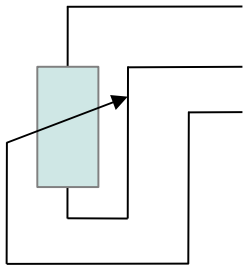
Preparation for lcd demo program

On RASPI

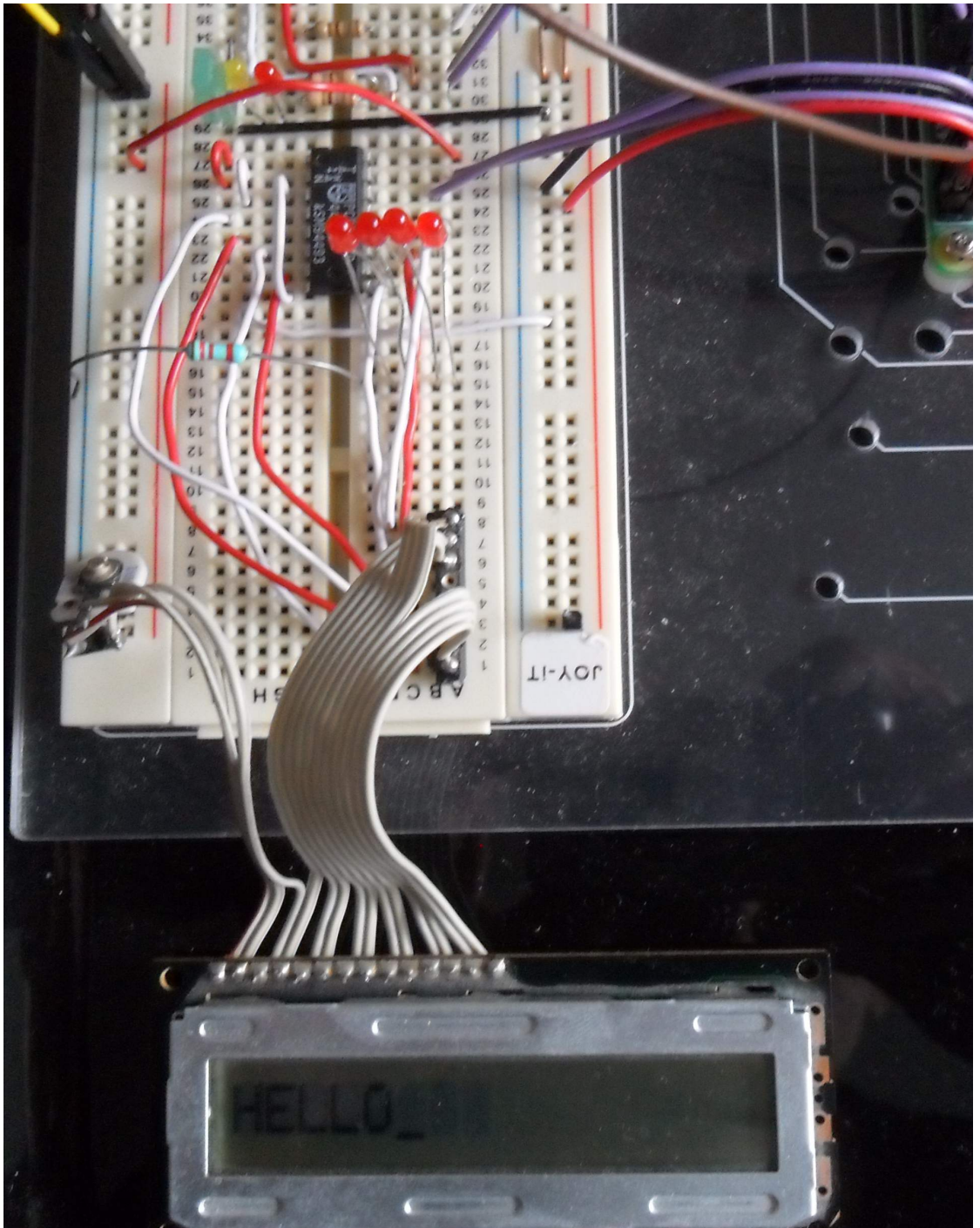
- starting-i2c
- server8888

On your computer

wish lcd-demo



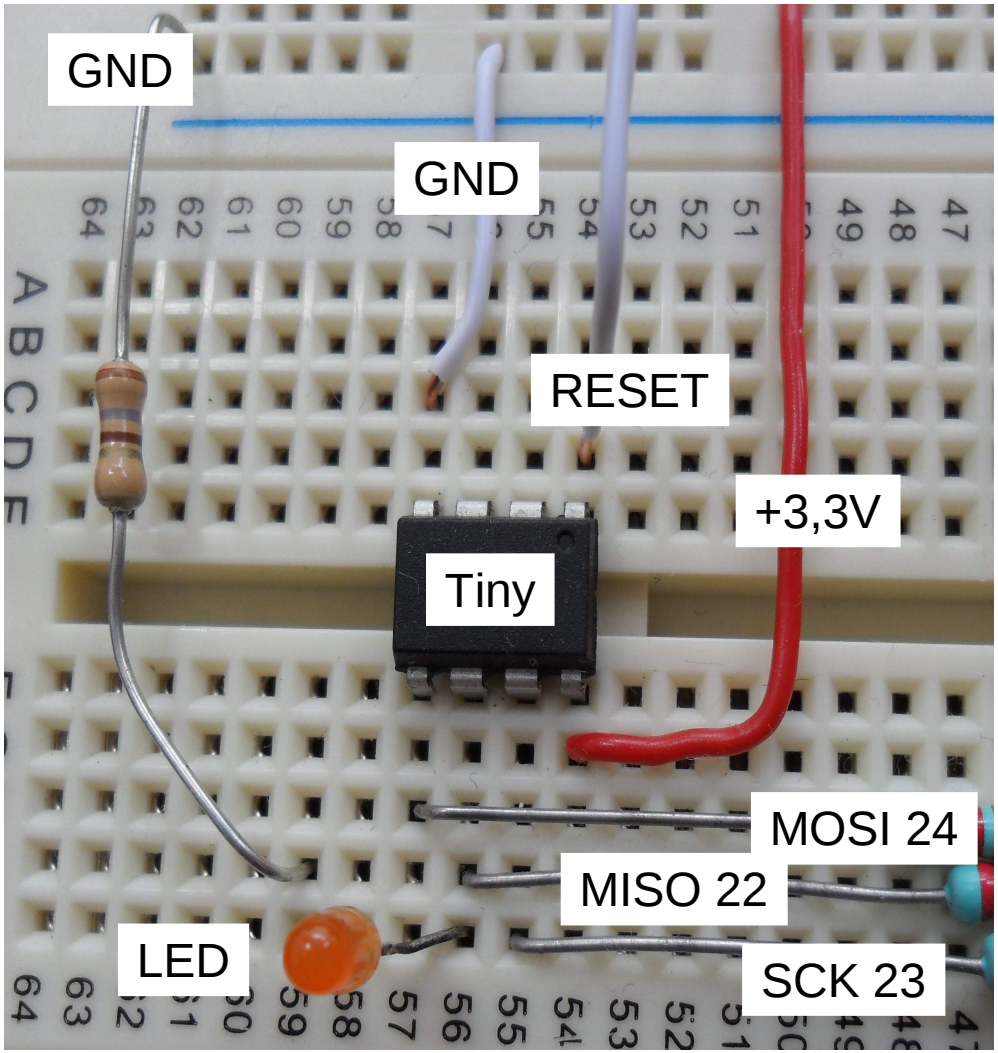
LCD-Module		PCF7483	
1	GND	8	GND
2	+5V	16	+5V
3	contrast		
4	RS	12	P7
5	R/W	11	P6
6	E	10	P5
7	-	-	
8	-	-	
9	-	-	
10	-	-	
11	D4	4	P0
12	D5	5	P1
13	D6	6	P2
14	D7	7	P3



Preparation for spi demo programs
On RASPI
- starting
- server8888

On your computer
wish spi-demos

RESET must
be connected
to GND during
programming!



For SPI-communication we need two outputs clock (sck) and data (mosi) and one inut (miso)
Mosi-bit becomes valid by rising clock
Miso bit becomes valid by falling clock-signal
Programming enable is shown here miso returns 53 in the third byte