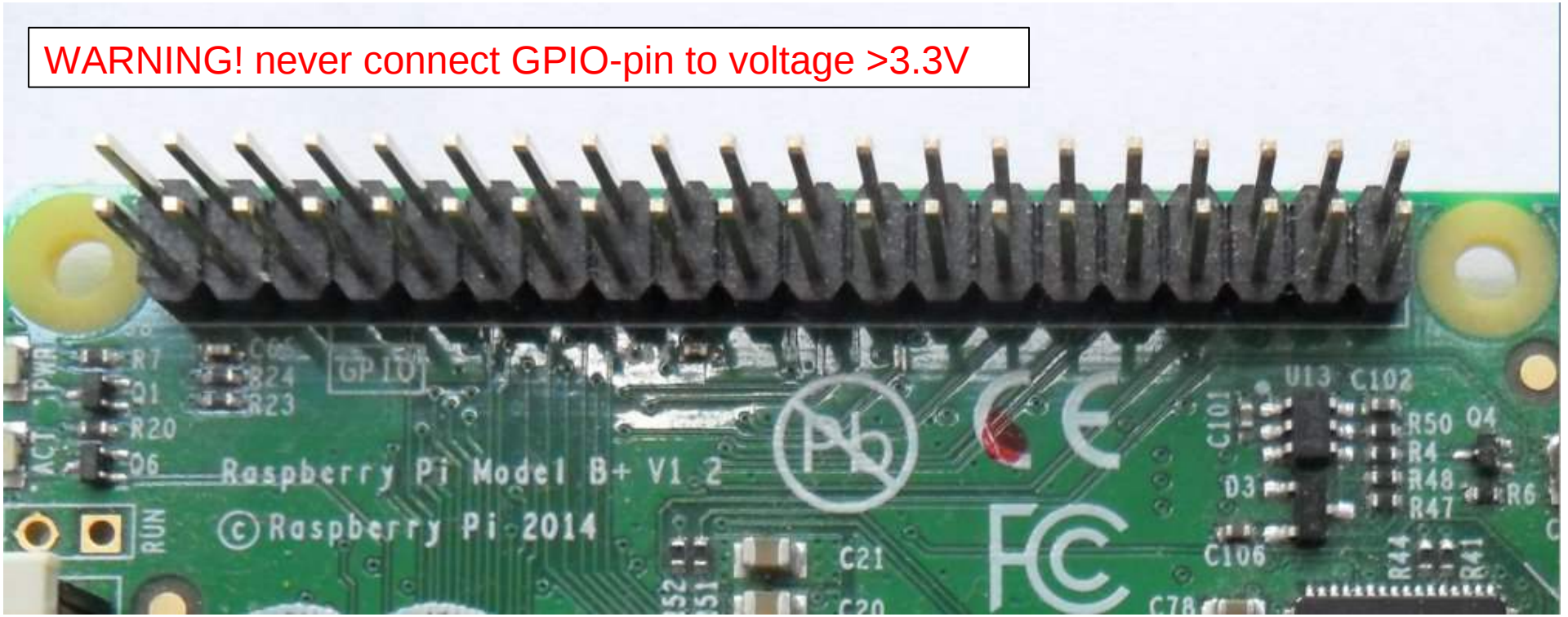
Using GPIOs of Raspberry Pi in pure Tcl - my way – Part II

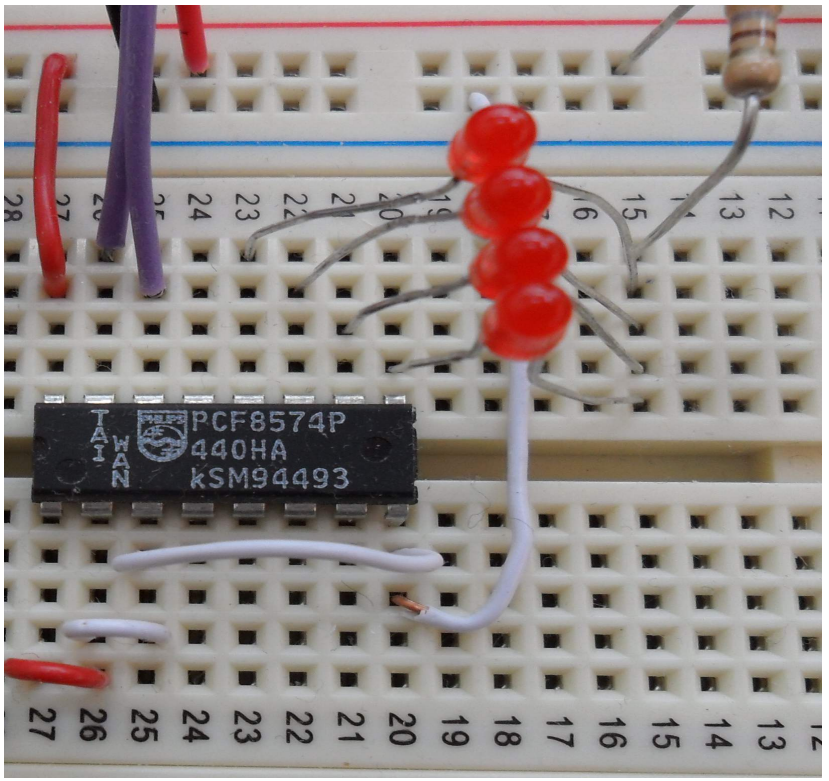
I2C-bus 8bit-port expander - LCD-module
SPI-bus programming ATtiny85 microcontroller



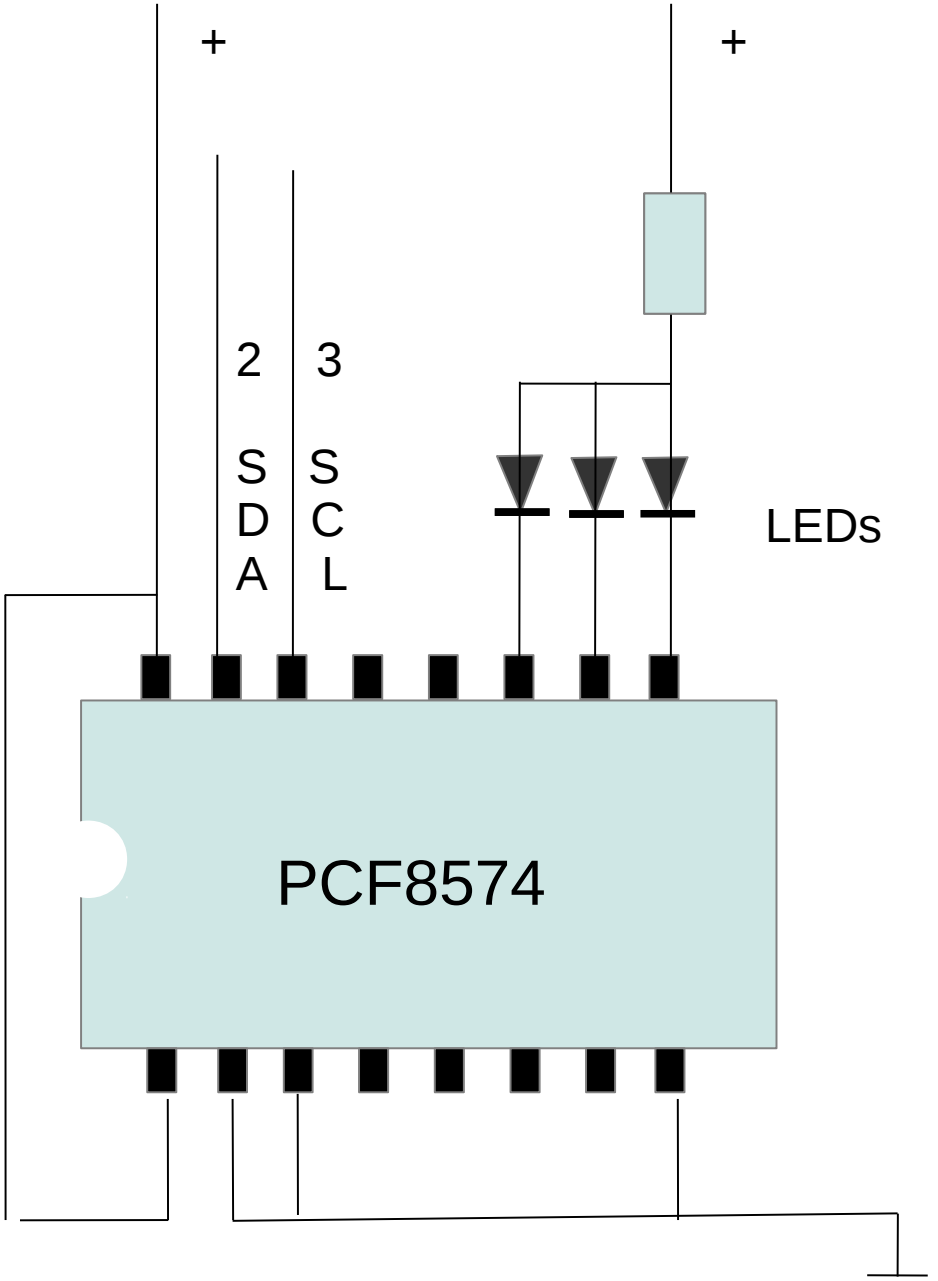
Preparation for i2c demo programs

On RASPI
- starting-i2c
- server8888

On your computer
wish i2c-demo



SDA & SCL are connected internally by pull-up resistors To 3.3V!



starting-i2c

```
# meti la komencajn kondicxojn por i2c

set h [open /sys/class/gpio/export w]
puts $h 2
after 1000
close $h

set h [open /sys/class/gpio/export w]
puts $h 3
after 1000
close $h
set x [exec ls /sys/class/gpio]
```

```
export
gpio2
gpio3
gpiochip0
unexport
```

unexporting

```
# unexporting GPIOs 2,3
set h [open /sys/class/gpio/unexport w]
catch {
puts $h 2
flush $h
}
after 1000
catch {
puts $h 3
close $h
}
set x [exec ls /sys/class/gpio]
```

```
export
gpiochip0
unexport
```

on raspberry-pi

Server

```
# start: exec tclsh server8888 &
proc akceptu {chan addr port} {
set x [gets $chan]
set y [eval $x]
puts $chan $y
close $chan
}
socket -server akceptu 8888
vwait forever
}
```

```
# scl low

set h [open /sys/class/gpio/gpio3/direction w]
puts $h out
flush $h
puts $h low
close $h
```

on computer side

Client

```
proc raspi {txt} {
set chan [socket 192.168.1.102 8888]
puts $chan $txt
flush $chan
set result [read $chan]
close $chan
return $result
}
```

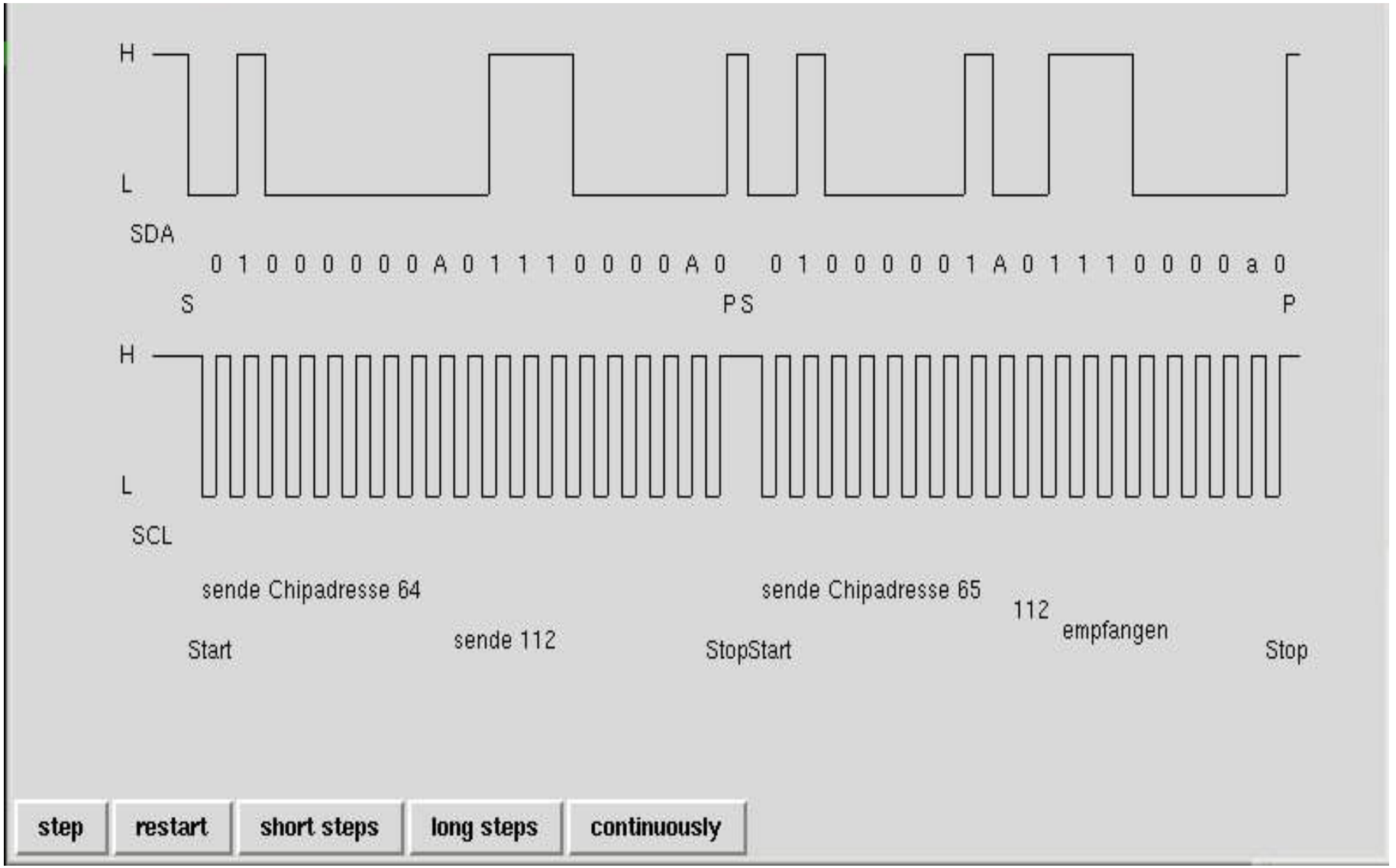
Communication

```
raspi "source sda-low"
set antwort [raspi "source read-sda"]
```

```
# read sda

set h [open /sys/class/gpio/gpio2/direction w]
puts $h in
close $h
set h [open /sys/class/gpio/gpio2/value r]
set x [read $h]
close $h
set y $x
```

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 acknowlege-bit (A,a,n)



I2c-schnell

procs:
init start stop sende empfange&a empfange beende
(init start stop send receive&a receive finish)

```
proc init {} {
# initialisiere
global h hh hhh
# scl
set h   [open /sys/class/gpio/gpio3/direction w]
# sda
set hh  [open /sys/class/gpio/gpio2/direction w]
set hhh [open /sys/class/gpio/gpio2/value r]
. . .
# read acknowlege - bit
puts $hh in
flush $hh
puts $h in
flush $h
seek $hhh 0
set x [read $hhh]
puts $h out
flush $h
return $x
```

```
proc sende {n} {
global h hh hhh
binary scan [binary format S* $n] b* bits
set li "out in"

# sende 7. bit
set b7 [lindex $li [string index $bits 15]]
puts $hh $b7
flush $hh
puts $h in
flush $h
puts $h out
flush $h
```

examples:

```
set y [sende 111]

setx [empfange]
```

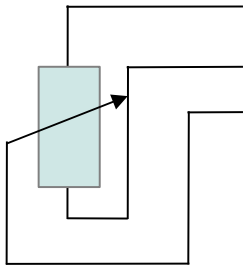
Preparation for lcd demo program

On RASPI

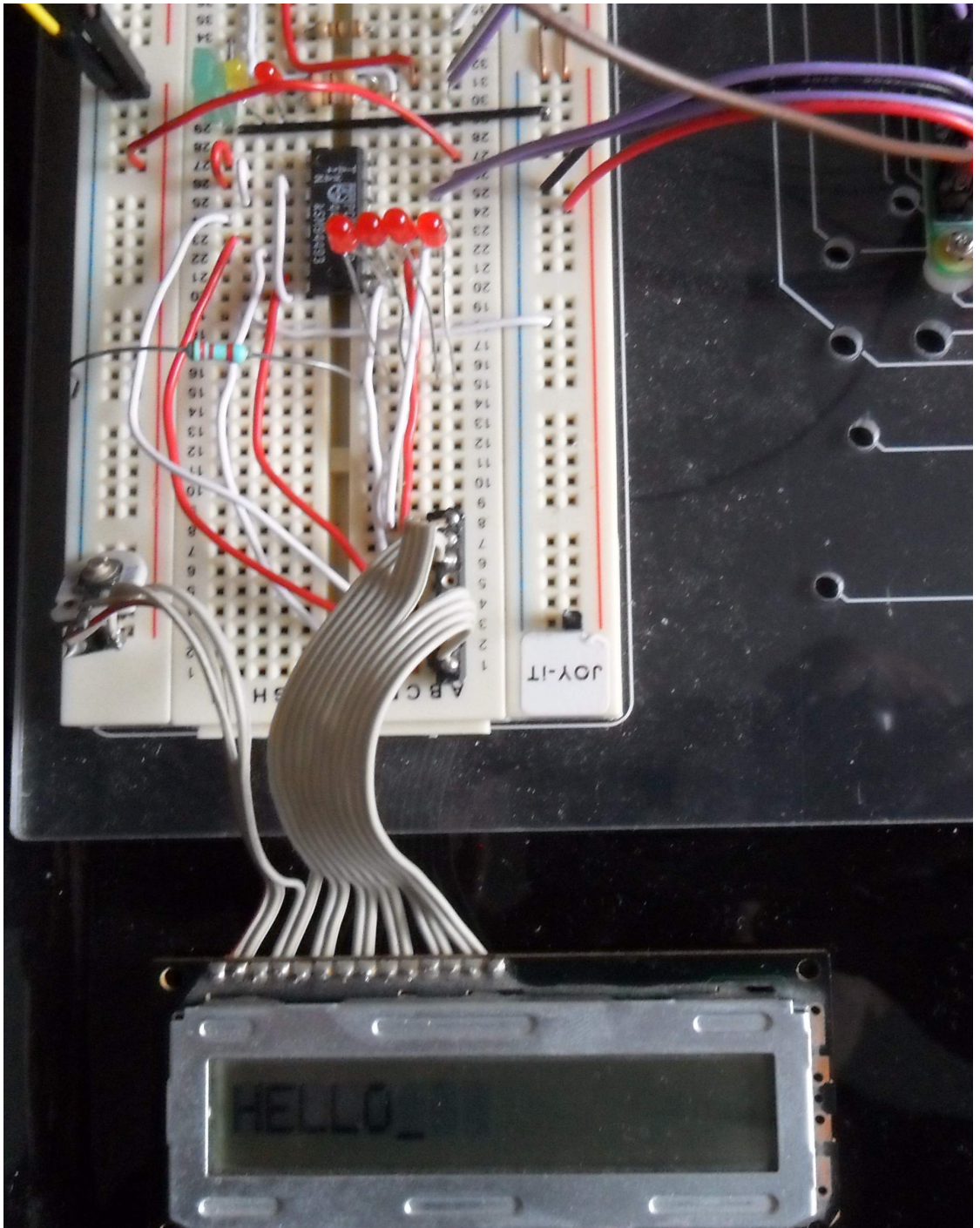
- starting-i2c
- server8888

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



Further information see

I2c

https://de.wikipedia.org/wiki/i2c

datasheets

A/D-converter ads1115 Texas Instruments

A/D-D/A-converter PCF8591 Philips/NXP

Clock and calender with 240x8-bit RAM PCF8583 Philips/NXP

8-bit I/O-expanderPCF8574 Philips/NXP

LCD-display

http://www.stefan-buchgeher.info/elektronik/lcd/lcd.html

datasheet

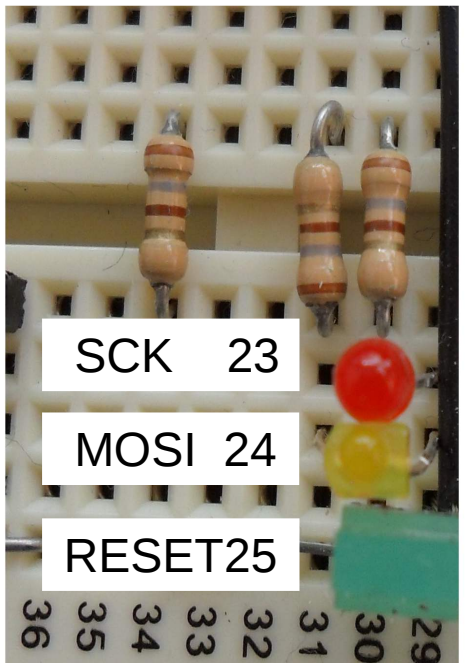
HD44780 Hitachi

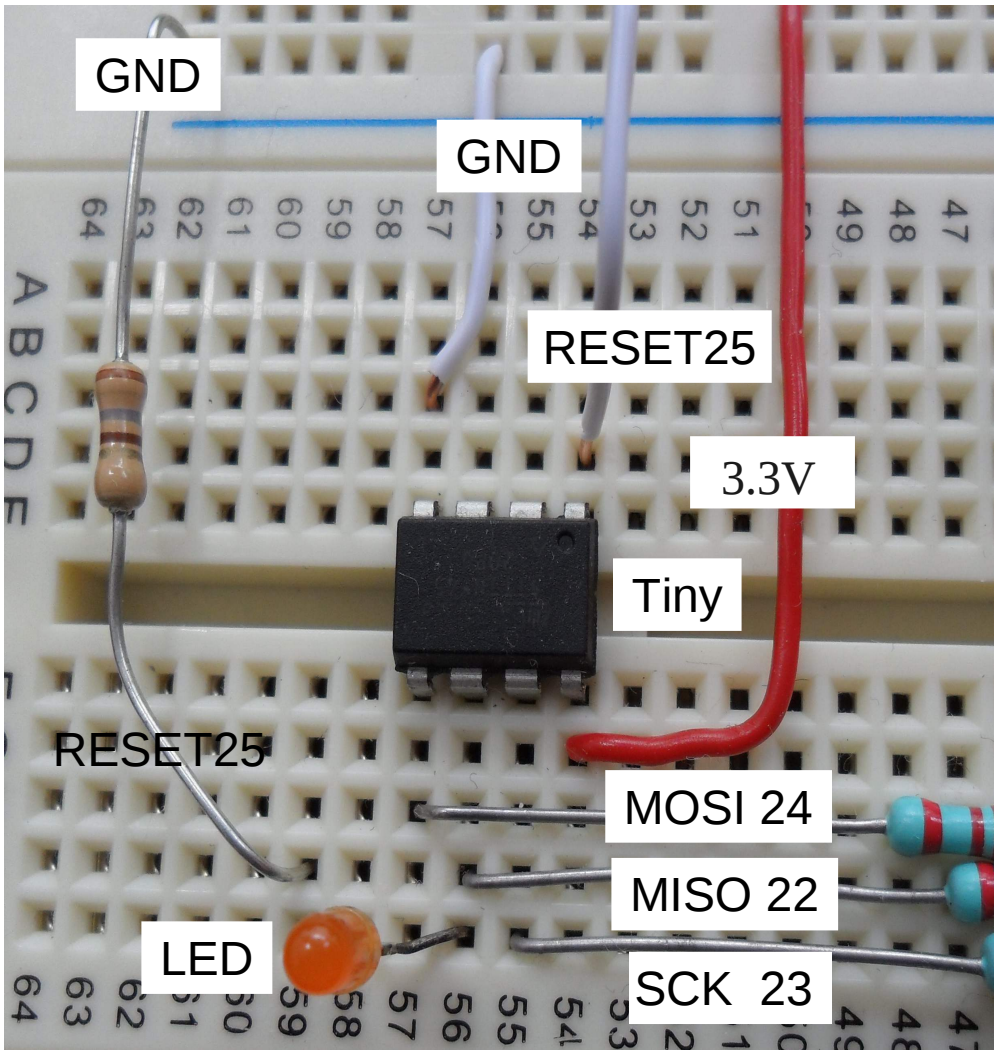
Preparation for spi demo programs

On RASPI
- starting
- server8888

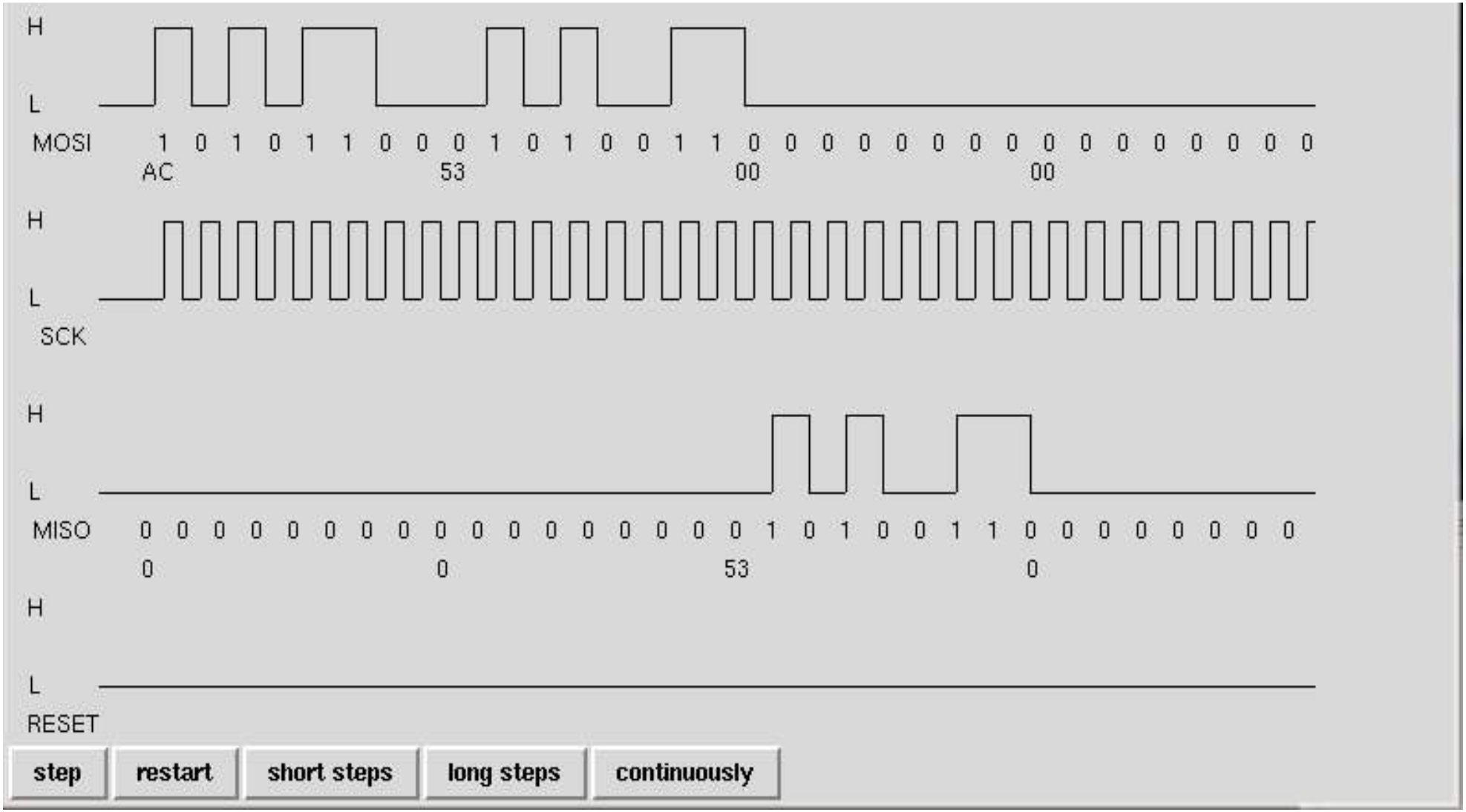
On your computer
wish spi-demo

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



code for blinking program

AC 53 00 00 programming enable

AC 80 00 00 chip erase

40 00 00 B9 sbi ddrb,1 1001 1010 AAAA Abbb set bit in i/o-register

48 00 00 9A 1001 1010 1011 1001 DDRB==17 lower byte first

40 00 01 00 ldi r16,16 1110 kkkk dddd kkkk load mmediate

48 00 01 E1 1110 0001 0000 0000 r16 == 0

40 00 02 0A out tccra,r16 1011 1AAr rrrr AAAA store register to i/o-location

48 00 02 BD 1011 1101 0000 1010 r16== 10 tccr0a == 2A

40 00 03 05 ldi r16,5 1110 kkkk dddd kkkk load immediate

48 00 03 E0 1110 0000 0000 0101

40 00 04 03 out tccrb,r16 1011 1AAr rrrr AAAA store register to i/o-location

48 00 04 BF 1011 1111 0000 0011 r16 == 10 tccr0b == 33

40 00 05 FF rjmp -1 1100 kkkk kkkk kkkk relative jump

48 00 05 CF 1101 1111 1111 1111 FFF == -1

4C 00 00 00 write buffer to flash

FF FF FF FF Ende

Spi-schnell

procs:

Initialisiere beende start-programming stop-programming sende sende-u-empfange

extract of initialisiere:

```
set h    [open /sys/class/gpio/gpio25/value w]
set hh   [open /sys/class/gpio/gpio24/value w]
set hhh  [open /sys/class/gpio/gpio23/value w]
set hhhh [open /sys/class/gpio/gpio22/value r]
```

```
reset output!
mosi  output
scl     output
miso  input
```

extract of sende-u-empfange:

```
set b7 [string index $bits 7]
puts  $hhh  0
flush $hhh
seek $hhhh 0
set x [read $hhhh]
if {$x == 1} {incr y 128}
puts  $hh   $b7
flush $hh
puts  $hhh  1
flush $hhh
```

Isolate bit 7 from byte that has to be sent
Set scl low

Read miso
If miso == 1 increment by 2^7 = 128
Set Mosi value of bit 7

Set scl high

Further reading:

Excellent tutorial:

http://www.elektronik-labor.de/AVR/KursAssembler/T13asm13.html

Datasheets (pdf):

AVR Instruction Set Manual – Microchip Technology Atmel/Microchip

attiny45 Atmel/Microchip