UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Simon Fabčič

Priklop perifernih naprav na Raspberry Pi

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: izr. prof. dr. Branko Šter

Ljubljana 2014

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika dela:

Opišite, kako na enoploščni računalnik Raspberry Pi povezujemo periferne naprave. Uporabite naslednjo periferijo oz. vodila: tipka in LED dioda, I2C, 1Wire, UART, SPI in znakovni LCD zaslon. Za vsako vrsto priklopa opišite protokol prenosa podatkov, električno vezalno shemo, potrebne nastavitve na Raspberry Pi-ju in ustrezno programsko kodo v jeziku Python.

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani Simon Fabčič,

z vpisno številko 63070222,

sem avtor diplomskega dela z naslovom:

Priključitev perifernih naprav na Raspberry Pi.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Branka Štera,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Planini pod Golico, dne 30. 6. 2014 Podpis avtorja:

Zahvaljujem se staršem za razumevanje in podporo v času celotnega študija. Za strokovno pomoč, nasvete in strojno opremo, ki sem jo potreboval za izdelavo diplomskega dela, se zahvaljujem zaposlenim v podjetju AX elektronika.

Posebna zahvala je namenjena mentorju, izr. prof. dr. Branku Šteru.

Kazalo

Povzetek

Moje diplomsko delo obravnava povezovanje perifernih naprav na Raspberry Pi. Najprej je predstavljena oprema, ki sem jo potreboval pri pisanju. Kot glavno strojno opremo sem uporabil enoploščni računalnik Raspberry Pi (model B) in nanj namestil operacijski sistem Raspbian. Na njem je bilo že pred naloženo razvojno okolje za programski jezik Python, ki sem ga tudi uporabljal. Nato sem prikazal, kako na Raspberry Pi na različne načine in prek raznih protokolov povežemo periferne naprave ter digitalne senzorje. Diplomska naloga je sestavljena tako, da je za vsak način priklopa najprej opisan protokol prenosa podatkov, električna vezalna shema z opisom, potrebne nastavitve na Raspberry Pi-ju in Python 3.x programska koda, ki omogoči komunikacijo z oddaljenimi napravami. Poleg opisa priklopa enostavnega vhoda (tipka) in izhoda (LED dioda) sem priključil še naprave na vodila I2C, 1Wire, UART in SPI. Da sem pokazal, kako se napiše nestandarden protokol, sem priključil tudi znakovni LCD zaslon s Hitachi krmilnikom HD44780 in za povezovanje z njim napisal modul. Pri protokolu I2C sem moral namesto predvidene strojne opreme uporabiti nizko nivojski »bit banging«, ki mi je omogočil do bita natančno komunikacijo. Nekateri I2C sužnji namreč zahtevajo ponovljen start bit, ki pa ga z uporabo modulov nisem mogel zagotoviti.

Ključne besede: Raspberry Pi, Python 3, I2C bit banging, 1Wire, UART, SPI,

Abstract

My thesis deals with connecting peripheral devices to the Raspberry Pi. At first I presents the equipment I needed for programming. For the general hardware I used the single-board Raspberry Pi (model B) and installed the Raspbian operating system. There was a development environment for the Python programming language already installed on the computer, which I used.

In the subsequent passages I presented how to connect the Raspberry Pi with peripheral devices and digital sensors using different ways and various protocols. This thesis is composed in such a way that there is first described for each type of connection the data transmission protocol, electrical wiring diagram with a description, the necessary settings for the Raspberry Pi-Rom, and the Python 3.x software code which enables communication with remote devices. Besides the description of the connection of the simple input (button) and output (LED diode), I added devices on the I2C bus, 1Wire, UART and SPI. To show how to write a non-standard protocol, I added a character LCD screen with a Hitachi HD44780 controller and wrote a module for connecting with it. With the I2C protocol I had to use low-level "bit banging", which enabled accurate communication with every bit. Certain I2C slaves require a repeated start bit, which I wasn't able to provide using the libraries.

Keywords: Raspberry Pi, Python 3, I2C bit banging, 1Wire, UART, SPI,

Poglavje 1

Uvod

Kot študent računalništva in informatike s poudarkom na strojni opremi in kot ljubitelj elektronike sem želel usvojiti znanja o povezovanju perifernih naprav prek različnih protokolov. To je osnova za vsak bolj kompleksen projekt. Srečo sem imel, da sem v podjetju, kjer sem opravljal praktično izobraževanje, po dogovoru z mentorjem delal ravno na tej tematiki. Ker tako kot vsem elektronikom, niti meni ne zmanjka idej o projektih, ki bi jih lahko naredil, mi je bilo pisanje diplomskega dela v veselje. Naučil sem se veliko novega – tako na področju programske kot tudi strojne opreme.

V prvem delu sem opisal glavno opremo, ki sem jo uporabljal pri pisanju diplomskega dela. To je enoploščni računalnik Raspberry Pi in programski jezik Python 3.x. Zakaj sem kljub slabemu podprtju z moduli vseeno uporabil Python 3.x namesto 2.7? Zato, ker želim biti v koraku s časom in ker je različica 3.x sedanjost in prihodnost programskega jezika Python. Raspberry Pi sem izbral, ker celotno programiranje poteka kar na njem. Ne potrebujemo nobenega dodatnega razvojnega programskega okolja, prevajalnikov, programatorjev in povezovalnih kablov, da bi napisano kodo lahko zagnali in ravno zato je zelo primerno orodje tudi za tehnično manj podkovane uporabnike. Poleg naštetih dobrih lastnosti ima že implementiran grafični vmesnik v visoki ločljivosti, povezovanje s svetovnim spletom, zaznavanje USB naprav, avdio izhod in na drugi strani GPIO priključke za nizkonivojsko komunikacijo, ki služijo za delo s strojno opremo.

Odčitavanje raznih fizikalnih veličin in komunikacija med sistemi sta postala praktično vsakdan našega življenja. Zato sem v nadaljevanju opisal ravno to. Uporabil sem že integrirane protokole prenosa podatkov. Dva sta sinhrona (I2C in SPI) in dva asinhrona (1Wire in UART). Malo sem se pomudil tudi pri električnih vezavah – torej kaj in kako je potrebno vezati elemente pri načrtovanju tiskanega vezja (PCB).

Poglavje 2

Orodja

V tem poglavju bom predstavil glavni del strojne opreme – Raspberry Pi, na katerega sem nato priključeval ostale periferne naprave.

Večinski delež napisane kode je v programu Python, katerega opis sledi v nadaljevanju.

2.1 Glavna strojna oprema – Raspberry Pi

Raspberry Pi je enoploščni računalnik velikosti kreditne kartice. Razvit je bil v Veliki Britaniji z namenom spodbujanja učenja osnovnih računalniških znanj v šolah. Želja je bila ponuditi dve različici; okrnjeno (model A) in bolj izpopolnjeno (model B). Sam sem si za delo izbral model B.

Slika 1: Raspberry Pi – mini računalnik[1]

[1] Vir slike: http://www.scratchmypi.co.uk/courses/using-the-raspberry-pi-in-the-primary-classroom/.

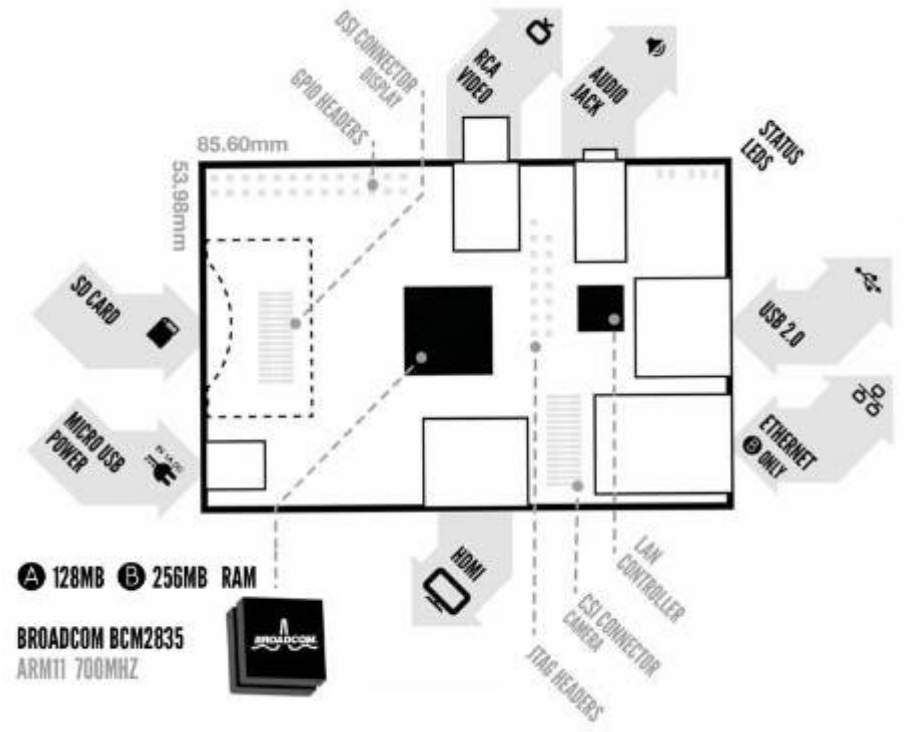
Ta računalnik ne vključuje trdega diska, ampak je za namestitev operacijskega sistema in shranjevanje podatkov predvidena SD (angl. Secure Digital) kartica oz. mikro SD prek adapterja. Namestitev operacijskega sistema je zelo preprosta, saj so nam na domači strani Raspberry Pi-ja za prenos na voljo podatkovne slike, ki jih nato enostavno namestimo na SD kartico. Če želimo še enostavnejšo namestitev, lahko uporabimo Berry Boot. To je preprost program, ki ga prekopiramo na SD kartico in z njegovo pomočjo namestimo operacijski sistem. Za namestitev imamo na voljo kar nekaj distribucij:

- Raspbian,
- Pidora,
- Openelec,
- Raspbmc,
- Risc os,
- Arch linux.

Izbral sem Raspbian, ki ima veliko programskih orodij za splošno rabo že pred nameščenih. Strojna oprema, ki nam je na voljo in nam omogoča delo na Raspberry Pi-ju, je zbrana v nadaljevanju:

Sistem na čipu	Broadcom BCM2835
Centralna procesna enota	700 MHz ARM1176JZF-S core
Delovni pomnilnik (SDRAM)	512 MB (deljen z grafično kartico)
Video vhod	CSI priključek omogoča priklop kamere
Video izhodi	Kompozitni RCA (PAL in NTSC) HDMI (1.3 in 1.4) (14 resolucij: 640 × 350 do 1920 × 1200) DSI vodilo za LCD
Avdio izhodi	3.5 mm avdio jack HDMI I^2S (možno tudi kot avdio vhod)
Spomin na osnovni plošči	SDIO (Secure Digital Input Output)
Omrežje na osnovni plošči	10/100 Mbit/s Ethernet (8P8C) USB adapter na tretjem priključku USB hub-a
Nizkonivojska periferija	8 x GPIO – splošno namenski priključki UART I2C SPI z dvema priključkoma CS I^2S

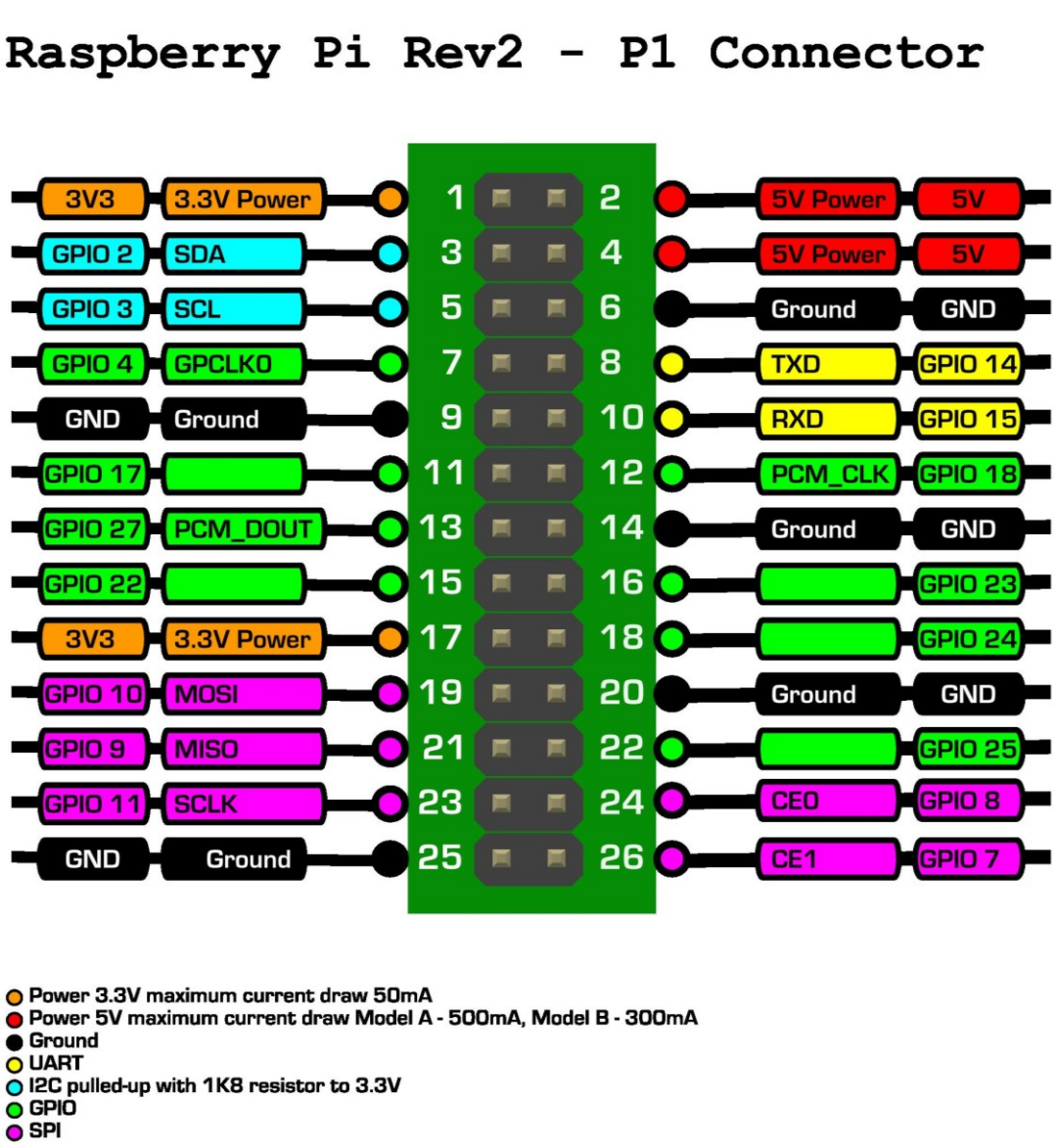
Nazivna moč	700 mA (3.5 W)
Vir napajanja	5 V preko MicroUSB 5 V preko GPIO vodila
Velikost	85.60 mm × 56 mm
Teža	45 g



Slika 2: Priključki Raspberry Pi-ja[2]

Seveda moram omeniti še priključke, ki sem jih uporabljal za priključitev perifernih naprav. Priklapljal sem jih preko GPIO konektorja, ki nam poleg običajnih digitalnih vhodov/izhodov omogoča tudi uporabo mnogih komunikacijskih protokolov. Omeniti še velja, da Raspberry Pi nima niti analognih vhodov niti analognih izhodov. Če potrebujemo tovrstno strojno opremo, jo priključimo prek standardnih protokolov prenosa podatkov. Veliko zmede v programe prinese tudi način označevanja priključkov. Na voljo imamo namreč dva načina. Prvi po oznakah GPIO, drugi pa po oznakah BOARD, ki so na Slika 3 prikazane na notranji strani zaporedno od 1 do 26 in ki sem jih tudi uporabil.

[2] Vir slike: http://www.thegeekscafe.com/tech-blogeeks/8-tech-blogeeks-hardware-category/chips-and-technology/109-35-pc-raspberry-pi-released.html.



Slika 3: Raspberry Pi priključni konektor GPIO[3]

[3] Vir slike: http://www.combinatorialdesign.com/boards/Raspberry_Pi/P1.

2.2 Glavna programska oprema – Python 3.x

Programski jezik Python je svojo zgodovino začel okrog leta 1990 in je že od začetka odprtokoden. Ima popolnoma dinamične podatkovne tipe in samodejno upravlja s pomnilnikom. Omogoča tudi objektno usmerjeno programiranje. Kot posebnost velja omeniti, da za določanje obsega zank ne uporablja oklepajev in zaklepajev, ampak to določamo z zamiki (angl. white space).

Čeprav je bila različica 3.x objavljena že leta 2008, je z moduli še dokaj slabo podprta. Nekateri uporabniki Python-a so ravno zaradi (zaenkrat še) slabe podpore še vedno zagovorniki različice 2.7.

Koda, napisana v programskem jeziku Python, je prenosljiva med operacijskimi sistemi, torej platformsko neodvisna.

Na nameščenem operacijskem sistemu Raspbian na Raspberry Pi, sta bili že nameščeni obe različici Python-a – 2.7 in 3.4, kar mi je na začetku povzročalo nekaj težav. Moduli, ki sem jih želel namestiti za novejšo različico, so se po privzetih nastavitvah namestili za starejšo.

Slika 4: Logotip programskega jezika Python[4]

[4] Vir slike: https://www.python.org/community/logos/.

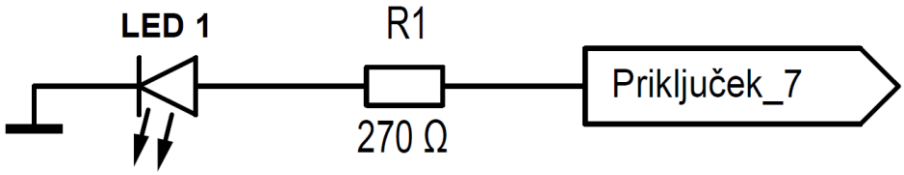
Poglavje 3

Načini povezovanja periferne strojne opreme na Raspberry Pi

Na Raspberry Pi sem najprej priključil tipko in LED diodo. Nato sem priključil znakovni LCD zaslon, za katerega je bilo potrebno implementirati protokol prenosa – napisati modul. Za tem so opisani še standardni protokoli prenosa podatkov, kot so I2C, 1Wire, UART in SPI.

3.1 Enostaven vhod/izhod

Prikazan je priklop LED diode kot primer enostavnega izhoda. Priključil sem jo na nožico 7 GPIO konektorja, kot kaže Slika 5. Z uporom R1 sem omejil tok skozi diodo na 12 mA. Diodo bom torej prižigal s pozitivno logiko.



Slika 5: Priključitev LED diode na priključek 7

Programska koda je enostavna. V program Python uvozimo modul za delo z GPIO konektorjem:

```
import RPi.GPIO as GPIO
```

Nastavimo način naslavljanja priključkov na »board« in priključek 7 določimo kot izhod:

```
GPIO.setmode(GPIO.BOARD)
GPIO.setup(7, GPIO.OUT)
```

Izhode lahko postavljamo v '1' na tri enakovredne načine:

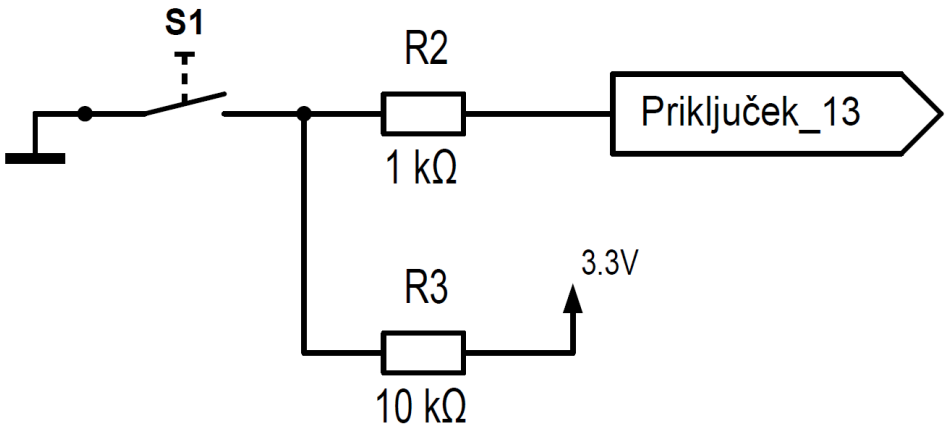
```
GPIO.output(7, GPIO.HIGH)
GPIO.output(7, True)
GPIO.output(7, 1)
```

Ravno tako imamo tri ukaze za postavljanje v '0':

```
GPIO.output(7, GPIO.LOW)
GPIO.output(7, False)
GPIO.output(7, 0)
```

Na tako preprost način lahko vklapljamo in izklapljamo izhode.

Za primer zaznavanja preprostega vhoda sem tipko vezal na priključek 13, tako kot kaže Slika 6. Da bi se znebil nedefiniranega stanja oz. stanja visoke impedance, sem uporabil 10 kΩ »pull up« upor (R3). Upor, ki je s stikalom vezan zaporedno (R2), je namenjen zaščiti priključka na Raspberry Pi-ju. Težava bi se namreč pojavila, če bi priključek 13 definirali kot izhod, ga nastavili v visok nivo in nato pritisnili tipko. V primeru brez upora bi prišlo do kratkega stika in posledično do uničenja priključka. Ko tipka ni pritisnjena, je torej vhod v logični '1', če pa je tipka pritisnjena, bo na vhodu logična '0'. Tu sem uporabil negativno logiko. Če je torej pogoj »if not vhod« izpolnjen, pomeni, da je tipka pritisnjena.



Slika 6: Priključitev tipke na priključek 13

V Python program zopet uvozim modul za delo z vhodi in izhodi – GPIO, nastavim način naslavljanja in priključek 13 določim kot vhod:

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setup(13, GPIO.GPIO.IN)
```

Tipko lahko enostavno zaznavam v toku programa:

```
if not GPIO.input(13):
  …
```

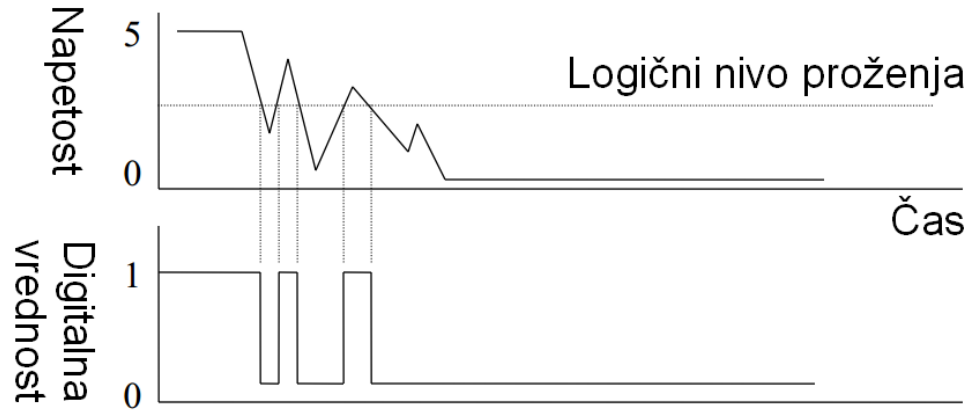
Taka uporaba tipke je v večini primerov neuporabna, ker tipko lahko pritisnemo, ko je program ne preverja. V iskanju rešitve sem naletel na funkcijo, ki čaka na rob signala:

```
GPIO.wait_for_edge(13, GPIO.FALLING)
```

Tudi ta rešitev se ni izkazala kot najboljša. Tok programa se namreč ustavi, ko čaka rob signala iz vhoda. Rešitev s prekinitvami se je izkazala kot najbolj elegantna. Prekinitev sem nastavil tako:

```
GPIO.add_event_detect(13, GPIO.FALLING, callback=metoda,
    bouncetime=25)
```

Tukaj že v definiciji zaznavanja dogodka nastavimo, katera metoda naj se ob pritisku pokliče in kakšen naj bo »bouncetime«. To je čas med dvema vzorčenjema. Težava se namreč pojavi pri mehanskem pritisku tipke, ko pride do prehodov med visokim in nizkim napetostnim nivojem (Slika 7). Z »bouncetime-om« torej določim, čez koliko časa po zaznani spremembi zopet preverja stanje na vhodu.

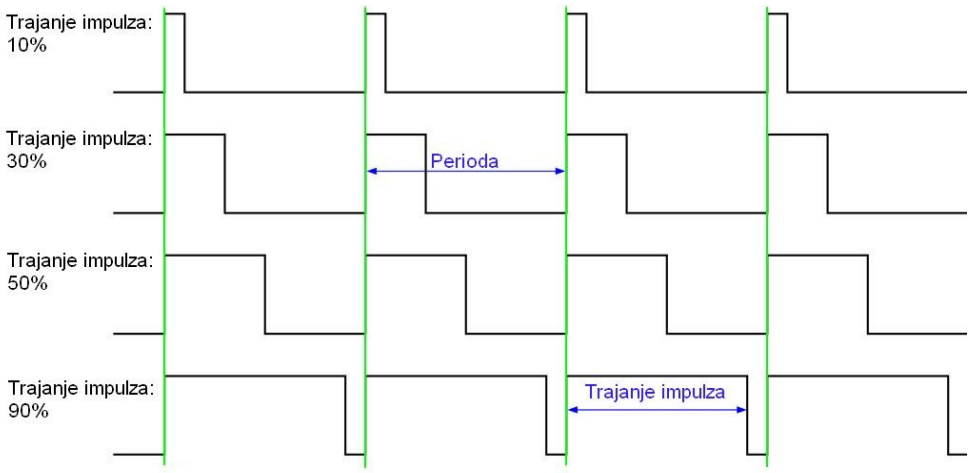


Slika 7: Prehod signala iz visokega v nizek logični nivo

To sta bila čisto preprosta vhod in izhod. Nato sem želel LED diodo še zatemnjevati. To sem enostavno rešil s pulzno širinsko modulacijo oz. PWM-jem (angl. Pulse-Width Modulation). Zopet sem koristil funkcijo v GPIO modulu, ki mi je zelo olajšala delo. Pri uporabi te funkcije ne kličem le neke metode z določenim parametrom, ampak ustvarim objekt, ki je namenjen pulzno širinski modulaciji. Najprej določim priključek za izhod, nato pa nad tem priključkom ustvarim objekt za pulzno širinsko modulacijo, v mojem primeru s frekvenco 1000 Hz in ga nato zaženem:

```
GPIO.setup(19, GPIO.OUT)
p = GPIO.PWM(19, 1000)
p.start(50)
```

Atribut 50, ki je podan metodi »start«, predstavlja trajanje pozitivnega dela periode v odstotkih, kot kaže tretja vrstica na Slika 8.

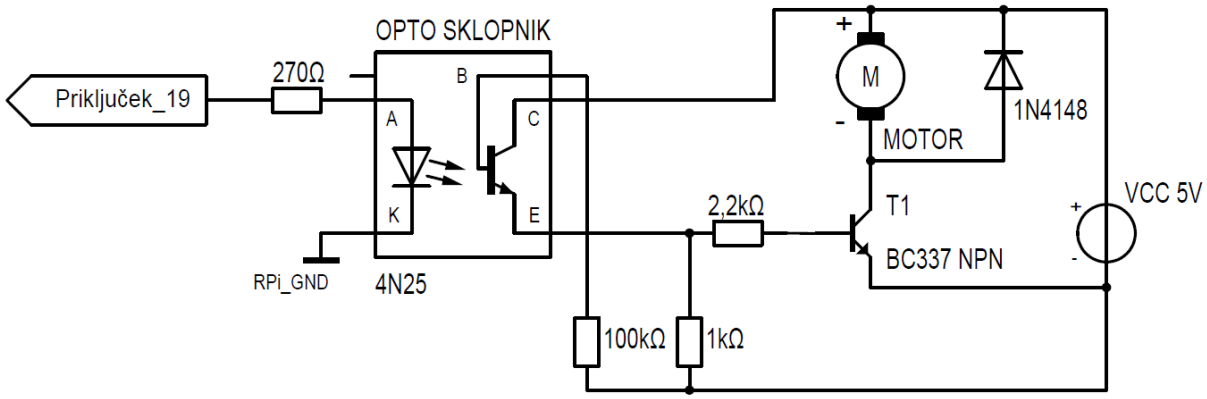


Slika 8: Pulzno širinska modulacija (PWM)

Seveda lahko dolžino trajanja pozitivnega dela periode tudi spreminjam s preprostim ukazom:

```
p.ChangeDutyCycle(TI)
```

Tukaj je »TI« številka med 0.0 in 100.0, ki predstavlja odstotek pozitivnega dela periode. S spreminjanjem trajanja impulza v visokem nivoju spreminjam tudi odstotek povprečne – efektivne napetosti, ki je enaka razmerju med stanjem impulza v visokem in nizkem nivoju. Pri visokih frekvencah se efektivna napetost odraža kot enosmerna. To sem preizkusil s priključitvijo enosmernega krtačnega motorja, kot kaže Slika 9.



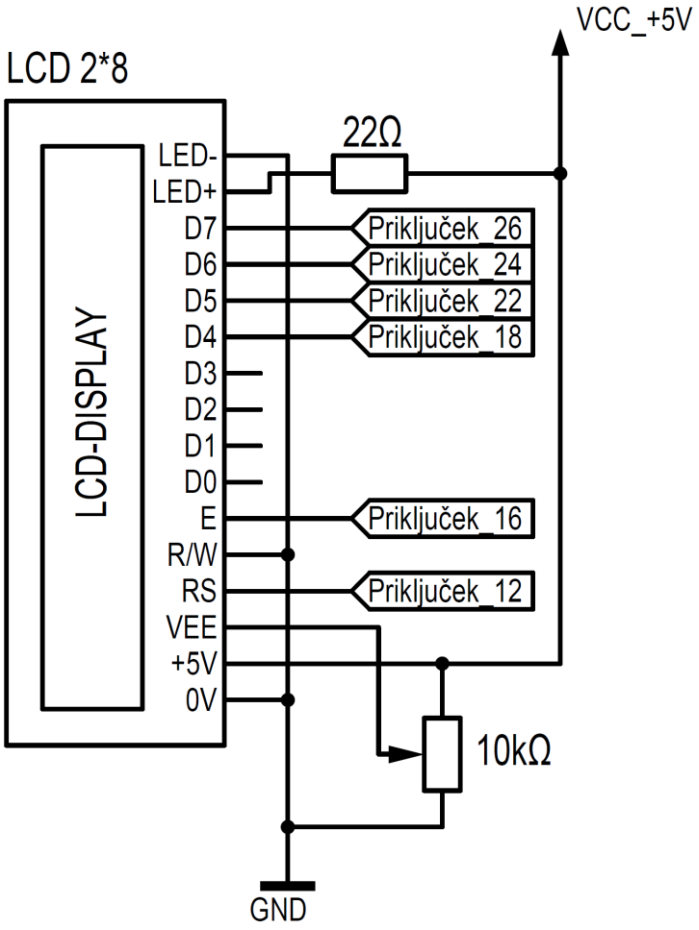
Slika 9: Priključitev enosmernega krtačnega motorja na Raspberry Pi

Da bi zaščitil priključek Raspberry Pi-ja, sem ga od 5V vezja ločil z optičnim sklopnikom. Ker je motor indukcijsko breme, sem vzporedno priključil nasprotno obrnjeno diodo, skozi katero se izničijo »špice«, ki nastanejo ob izklopu motorja in bi lahko privedle do preboja tranzistorja.

Na Raspberry Pi-ju sem spreminjal parameter trajanja pozitivnega dela impulza in s tem tudi hitrost vrtenja motorja.

3.2 Znakovni LCD

Pri znakovnem LCD zaslonu bom pokazal, kako je zasnovana komunikacija, za katero ni določenega standardnega protokola. V veliko tovrstnih zaslonov je vgrajen Hitachi-jev krmilnik HD44780, zato je upravljanje in priključevanje večine enako. Do razlik pride le med dimenzijami, saj so lahko zasloni eno, dvo, štiri ali osem vrstični. En HD44780 podpira naslavljanje 80 znakov. Ker so zasloni tudi večji, jih lahko upravljamo z dvema krmilnikoma, ki imata dodaten priključek – omogočen ali onemogočen. Priključil sem znakovni LCD, dimenzij 2 x 8, kot kaže Slika 10. Zaradi varčevanja s priključki na Raspberry Pi-ju sem namesto osem linijskega vodila uporabil štiri linijsko.



Slika 10: Priključitev znakovnega LCD-ja na Raspberry Pi

Nato sem na svetovnem spletu poiskal tehnično dokumentacijo, iz katere sem ugotovil, kako se LCD zaslonu pošilja ukaze in podatke. Pošiljati sem jih moral kot nibble in ne kot bajt. Tovrstno pošiljanje podatkov je seveda počasnejše, vendar se pri normalni uporabi tega niti ne opazi. Napisati sem torej moral funkcijo, ki LCD-ju pošilja nibble. To je potrebno narediti tako, da štiri podatkovne linije nastavimo na želeno vrednost, nato pa na priključek E (angl. Enable) damo najprej visok nivo, nato pa zopet nizkega, da se ob negativnem robu podatki iz vodila prenesejo v LCD krmilnik. Funkcijo za pošiljanje nibble-a sem napisal tako:

```
def send_nibble(data):
  GPIO.output(_LCD_D4, bool(data & 0x1))
  GPIO.output(_LCD_D5, bool(data & 0x2))
  GPIO.output(_LCD_D6, bool(data & 0x4))
  GPIO.output(_LCD_D7, bool(data & 0x8))
  pulse_enable_transmit()
```

Napisal sem tudi funkcijo, ki LCD-ju pošlje bajt. Ta funkcija dvakrat kliče »send_nibble« – s štirimi visokimi in štirimi nizkimi biti. Posebnost krmilnika je tudi priključek RS (angl. Register Select), s katerim določimo, ali so na vodilu ukazi ali podatki za prikaz na zaslonu.
Sedaj lahko napišem funkcijo za inicializacijo zaslona. V njej priključek RS nastavim najprej na nizek nivo (poslani biti predstavljajo ukaze), nato pa pošljem podatke tako, kot zahteva tehnična dokumentacija:

```
def init_display():
  GPIO.output(_LCD_RS, False) #pošiljali bomo ukaze
  send_nibble(0x3) #inicializacija
  send_nibble(0x3) #inicializacija
  send_nibble(0x3) #inicializacija
  send_nibble(0x2) #nastavimo na 4 bitni način
  send_byte(0x28) #4 bitni način, 2 vrstici, 5x7 matrica
  send_byte(0x0C) #vključimo LCD in izključimo kurzor
                   (0x0E za vključitev)
  send_byte(0x06) #pišemo od leve prot desni, brez
                   zamikanja
  send_byte(0x01) #počistimo zaslon
```

Sedaj je LCD pripravljen na sprejem znakov za prikazovanje na zaslonu. S funkcijo »send_char« lahko pošljem znak LCD krmilniku s tem, da z atributom »True« nastavim RS nožico na visok nivo, kar pomeni pošiljanje znakov za prikaz na zaslonu.

```
def send_char(ch):
  send_byte(ord(ch), True) #funkcija ord() vrne int
                            vrednost znaka
```

Kazalec se samodejno premika v desno; ko pošljem naslednji znak, se zapiše na naslednje mesto.

Omeniti moram še implicitno naslavljanje vrstic in stolpcev z znaki. Na točno določeno mesto se lahko postavimo tako, da LCD krmilniku pošljemo ukaz za želeno lokacijo. To naredim tako, da ukazu za nastavitev naslova (0 x 80) prištejem naslov vrstice, nato pa še naslov stolpca:

```
def go_to_location(line, column):
  column = column-1
  if line <= 1:
    send_byte(0x80+0x00+column)
```

```
elif line == 2:
  send_byte(0x80+0x40+column)  #0x80 za premik
            kurzorja, 0x40 za vrstico, column za stolpec
elif line == 3:
  send_byte(0x80+0x14+column)
else:
  send_byte(0x80+0x54+column)
```

S pomočjo tehnične dokumentacije sem sestavil protokol, ki mi omogoča pisanje na točno določeno lokacijo znakovnega LCD-ja.

3.3 I2C

I2C (angl. Inter-Integrated Circuit) znan tudi kot I^2C ali IIC in najpogosteje izgovorjen kot »I kvadrat C« je protokol podjetja Philips. Primeren je za komunikacijo pri nižjih hitrostih prenosa podatkov. Za komunikacijo potrebuje dve liniji – podatkovno (SDA) in takt ure (SCK), ki določa odčitavanje vrednosti iz podatkovne linije. Ker sužnji za logično '0' podatkovno linijo vežejo proti masi, za logično '1' pa jo pustijo nepovezano, morata biti obe liniji prek uporov dvignjeni na visok nivo. Tipična napetostna nivoja sta 5V in 3,3V, dovoljeni pa so tudi drugi. I2C vodilo ima naslovni prostor velikosti 7 ali 10 bitov, odvisno od uporabljenih naprav. Topologija, ki jo omogoča vodilo, je več gospodarjev in več sužnjev.

Da sem na Raspberry Pi-ju lahko uporabljal I2C vodilo, sem moral spremeniti nekaj nastavitev. Načrtovalci operacijskega sistema so namreč predvideli, da večina uporabnikov tovrstne komunikacije ne potrebuje, zato je v privzetih nastavitvah izključena. Omogočil sem jo tako, da sem v datoteko »/etc/modules« na konec dodal »i2c-dev«. Moral sem odstraniti še I2C vodilo iz t. i. črne liste »/etc/modprobe.d/raspi-blacklist.conf«. Nato sem imel možnost komuniciranja z I2C napravami. Za lažje delo z vodilom sem z ukazom

```
$ sudo apt-get install i2c-tools
```

namestil priročno orodje. Z njim lahko pišemo na vodilo, z vodila beremo, preberemo celoten pomnilnik sužnja in preberemo prisotne sužnje na vodilu. Slednje naredimo z ukazom:

```
sudo i2cdetect -y 1
```

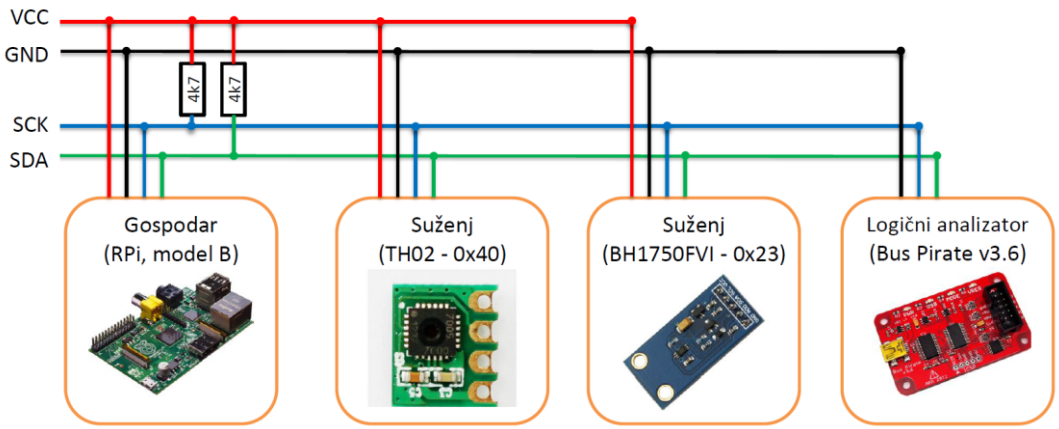
kjer je "1" številka vodila. Na Raspberry Pi-ju sta namreč na voljo dva I2C vodila – z oznako "0" in z oznako "1". Pri starejših različicah je za uporabo na voljo le eden, in sicer "0". Ker

sem pri testiranju linije imel priključen en TH02, z naslovom hex40, je sistem izpisal, kot kaže Slika 11.

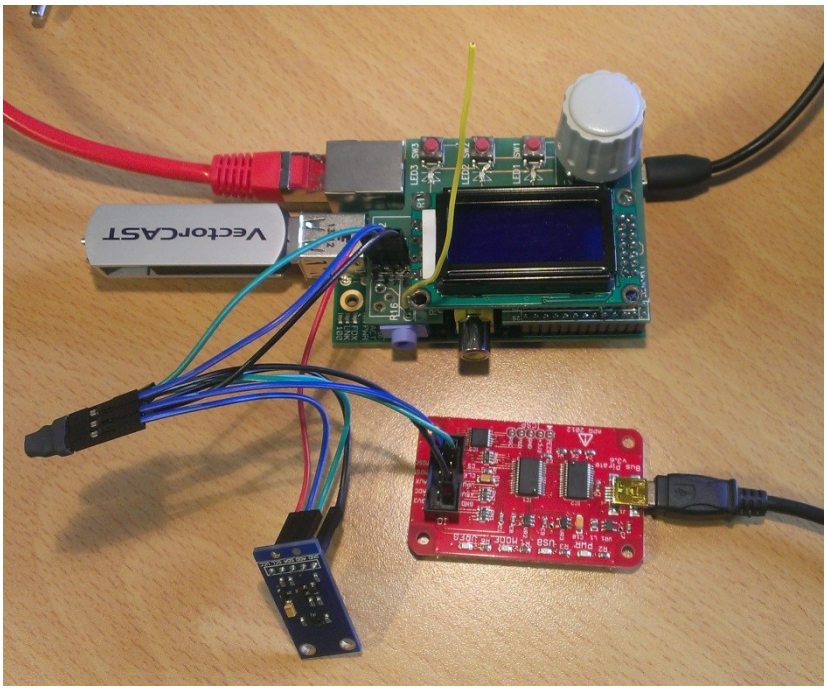
```
pi@raspberrypi ~ $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Slika 11: Izpis naslova sužnja TH02

I2C vodilo deluje. Na Raspberry Pi sem priključil še drugega sužnja, senzor osvetljenosti BH1750FVI. Zaradi nekaterih težav sem kasneje priključil tudi logični analizator Bus Pyrate. Vse skupaj sem povezal, kot kažeta Sliki Slika 12 in Slika 13.

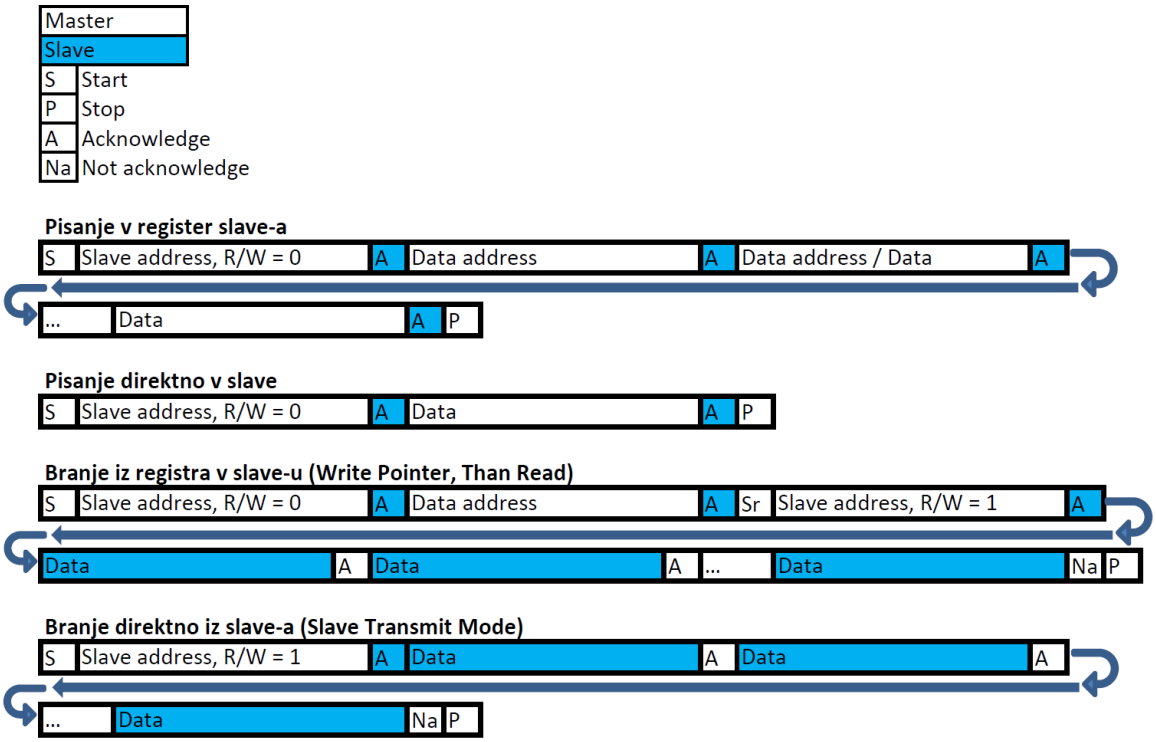


Slika 12: Shematski prikaz priključitve I2C naprav



Slika 13: Fizični priklop I2C naprav

Po spletu sem začel iskati module za delo s sužnji. Kljub velikemu naboru modulov, sužnji niso delovali oz. so delovali le delno. Ker napake nisem mogel odkriti, sem na linijo priključil logični analizator »Bus Pirate« in za prikaz stanja linije na osebnem računalniku uporabil »Logic Sniffer«. Niti strojna niti programska oprema nista avtorsko zaščiteni. Z analizatorjem sem ugotovil, da je v komunikaciji en stop bit preveč. Ko pošiljam zahtevek za branje iz registra sužnja, se namreč po naslovu registra ponovi start bit, brez predhodnega stop bita (Slika 14 – branje iz registra v slave-u).



Slika 14: Načini komunikacije I2C

Te napake z moduli s spleta nisem mogel odpraviti. Ugotovil sem, da mi ne preostane drugega kot »Bit banging«. To je tehnika, s katero komunikacijo implementiramo programsko, namesto da bi za to uporabili namensko strojno opremo. Težava je, da s tem načinom bolj obremenimo glavni procesor, vendar druge možnosti nisem imel. Napisal sem torej modul, ki ima zapis bitov ločen po funkcijah. Tako sem za start in stop bit napisal naslednji funkciji:

```
def start_bit():
  GPIO.output(_I2C_sda, True)
  sleep(_DELAY)
  GPIO.output(_I2C_scl, True)
  sleep(_DELAY)
  GPIO.output(_I2C_sda, False)
  sleep(_DELAY)
  GPIO.output(_I2C_scl, False)
  sleep(_DELAY)

def stop_bit():
  GPIO.output(_I2C_scl, False)
  sleep(_DELAY)
  GPIO.output(_I2C_sda, False)
```

```
  sleep(_DELAY)
  GPIO.output(_I2C_scl, True)
  sleep(_DELAY)
  GPIO.output(_I2C_sda, True)
  sleep(_DELAY)
```

Za funkcijo branja iz vodila sem moral podatkovno linijo spremeniti v vhod in nato v odvisnosti od takta ure odčitavati vrednosti na liniji:

```
def read_byte():
  GPIO.setup(_I2C_sda, GPIO.IN)
  sleep(_DELAY)
  vrednost = 0
  for j in range (8):
    vrednost = vrednost << 1
    GPIO.output(_I2C_scl, False)
    sleep(_DELAY)
    GPIO.output(_I2C_scl, True)
    sleep(_DELAY)
     if GPIO.input(_I2C_sda):
      vrednost = vrednost | 1
  GPIO.setup(_I2C_sda, GPIO.OUT)
  sleep(_DELAY)
  GPIO.output(_I2C_sda, True)
  sleep(_DELAY)
  GPIO.output(_I2C_scl, True)
  sleep(_DELAY)
  return vrednost
```

Pri pisanju na linijo suženj potrdi prejetje enega bajta s t. i. ACK bitom. To naredi tako, da podatkovno linijo veže na nizek nivo. Torej sem moral tudi tu podatkovno linijo spremeniti v vhod, preveriti njen nivo in jo nato zopet nastaviti kot izhod. Funkcija za pisanje na vodilo je naslednja:

```
def write_byte(data):
  if not (data > 0x00 or data < 0xFF):
    raise ValueError("Given 'data' argument (" + str(data)
        + ")is not byte")
  data = bin(data)
  data = data[2:]
  while len(data) < 8: #npr. iz "10011" dobim "00010011"
```

```
    data = "0"+data
  for i in data:
    bit = int(i)
    GPIO.output(_I2C_scl, False)
    sleep(_DELAY)
    GPIO.output(_I2C_sda, bit)
    sleep(_DELAY)
    GPIO.output(_I2C_scl, True) #branje ob pozitivni
                                 fronti
    sleep(_DELAY)
  GPIO.output(_I2C_scl, False)
  sleep(_DELAY)

  GPIO.setup(_I2C_sda, GPIO.IN)
  sleep(_DELAY)#ni nujno
  GPIO.output(_I2C_scl, False)
  sleep(_DELAY)
  GPIO.output(_I2C_scl, True)
  if not GPIO.input(_I2C_sda): #ali je slave pull down-al
                               linijo
    pass
  else:
    print("Slave didn't recive Byte", hex(int(data, 2)))
  sleep(_DELAY)
  GPIO.output(_I2C_scl, False)
  sleep(_DELAY)

  GPIO.setup(_I2C_sda, GPIO.OUT)
  sleep(_DELAY
```

Prikazal sem osnovne funkcije, s katerimi sem nato lahko komuniciral s sužnji. Ta dva sužnja se razlikujeta po tem, da imam pri TH02 dostop do pomnilnika, pri BH1705FVI pa le do nekakšnega medpomnilnika, v katerega pišem ukaze in iz katerega berem izmerjene vrednosti. Prikazal bom razliko v kodi med branjem iz enega in iz drugega.

TH02 (iz pomnilnika sužnja preberemo 2 bajta):

```
I2C.start_bit()
I2C.write_byte(0x80) #0x40 + write bit('1')
I2C.write_byte(0x01) #naslovljen register 0x01
I2C.start_bit() #ponoven start bit, ker bomo brali
I2C.write_byte(0x81) #0x40 + read bit
```

```
tmp_temp_humi = I2C.read_byte()
I2C.ack() #prvi bajt potrdimo
tmp_temp_humi = tmp_temp_humi << 8 #pripravimo prostor za
                                    drugi bajt
tmp_temp_humi = tmp_temp_humi | I2C.read_byte()
I2C.nack() #zadnjega bajta ne potrdimo
I2C.stop_bit()
return tmp_temp_humi
```

BH1705FVI (iz medpomnilnika sužnja preberemo 2 bajta):

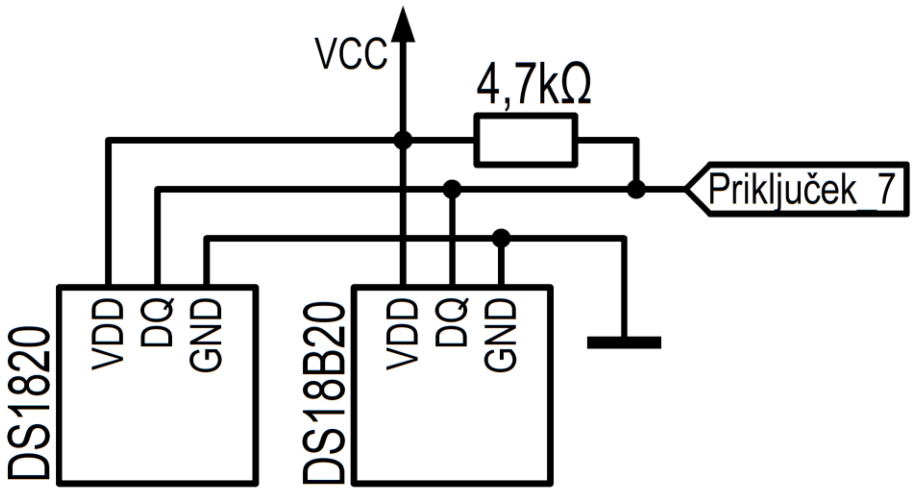
```
I2C.start_bit()
I2C.write_byte(0x46) #0x23 + write bit('1')
readed = I2C.read_byte()
I2C.ack()#prvi bajt potrdimo
readed = readed << 8
readed = readed | I2C.read_byte()
I2C.nack() #zadnjega bajta ne potrdimo
I2C.stop_bit()
return readed
```

Kot je vidno pri TH02, sem v četrti vrstici ponovil start bit, brez predhodnega stop bita, ki so ga dodajali moduli s spleta.

3.4 1Wire

1Wire je komunikacijski standard, zasnovan s strani podjetja Dallas Semiconudctors Corp. Zagotavlja nizkohitrostni prenos podatkov in napajanje po eni liniji. Namenjena je daljšim razdaljam in komunikaciji z manjšimi nizkocenovnimi napravami, kot so digitalni senzorji. Posebnost 1Wire vodila je, da omogoča možnost uporabe le dveh linij: podatkovne in mase. Da je to omogočeno, imajo naprave vgrajen 100 pF kondenzator, ki se polni v času mirovanja vodila – podatkovna linija je v visokem nivoju. Pri vodilu, daljšem od 1 m, se priporoča tudi priklop VCC priključka.

Na Raspberry Pi sem priključil Dallas-ova senzorja temperature DS18B20 in DS1820. Vezal sem ju na Raspberry Pi priključek 7, ki je predviden za tovrstno komunikacijo (Slika 15). Med VCC in podatkovno linijo je tudi »pull-up« upor 4,7 kΩ.



Slika 15: Vezava 1Wire sužnjev na RPi

Za delo z 1Wire sem na Raspberry Pi uvozil Linux model z ukazom:

```
$ sudo modprobe w1-gpio
```

Ko pa sem začel iskati module za Python, ki bi komunicirali s senzorjem temperature, sem zopet naletel na Linux modul in ga namestil z ukazom:

```
$ sudo modprobe w1-therm
```

Namestil sem Linux orodja. Uporabim jih tako, da v mapi »/sys/bus/w1/devices« poiščem moji napravi, ki ju predstavljata znakovni datoteki. Za prikaz bom izpisal vsebino datoteke senzorja DS1820:

```
$ cat /sys/bus/w1/devices/10-000802839836/w1_slave
31 00 4b 46 ff ff 07 10 8d : crc=8d YES
31 00 4b 46 ff ff 07 10 8d t=24312
```

Na koncu prve vrstice je oznaka »crc« (angl. Cyclic Redundancy Check), ki ima oznako »YES«, kar pomeni, da je branje uspelo. Ker je bil prenos podatkov v redu, je na koncu druge vrstice številka 24312, ki označuje prebrano vrednost. To številko moram le deliti s 1000 in dobim vrednost izmerjene temperature v stopinjah Celzija.
Pokazal bom, kako sem postopal pri dostopu do senzorjev iz Python-a. Najprej sem uvozil modul »os«, ki mi omogoča delo z Linux ukazi:

```
import os
```

Nato sem v Linux uvozil modula za delo z 1Wire in Dallas-ovimi temperaturnimi senzorji. Program lahko zaženem tudi, če prej nisem uvozil modul-ov v Linux-ovi lupini.

```
os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')
```

Naslove senzorjev, ki sem jih našel v mapi z 1Wire sužnji, sem shranil v spremenljivki »addr1« in »addr2«:

```
addr1 = "10-000802839836"
addr2 = "28-000002dca3de"
```

Najprej bom prikazal, na kakšen način berem vsebino datotek. V metodi »read_temp_raw(dev_file) « je v spremenljivki »dev_file« podana pot do datoteke senzorja, ki jo odprem za branje in njeno vsebino vrnem.

```
def read_temp_raw(dev_file):
  f = open(dev_file, 'r')
  lines = f.readlines()
  f.close()
  return lines
```

V glavni metodi »read_temp(addr) « kličem prej omenjeno metodo. Ta metoda dobi ime datoteke sužnja, katerega temperaturo želim. To ime nato sestavi v pot, ki jo posreduje metodi »read_temp_raw«. Vrnjeni dobim dve vrstici – v prvi preverim, če so zadnji trije znaki »YES« (če niso, branje ponovim), nato v drugi poiščem znaka »t=« in za njima odčitam temperaturo.

```
def read_temp(addr):
  dev_file = "/sys/bus/w1/devices/" + addr + "/w1_slave"
  lines = read_temp_raw(dev_file)
  while lines[0].strip()[-3:] != 'YES':
    time.sleep(0.2)
    lines = read_temp_raw(dev_file)
  equals_pos = lines[1].find('t=')
  if equals_pos != -1:
    temp_string = lines[1][equals_pos+2:]
    temp_c = float(temp_string) / 1000.0
    return temp_c
```

Vse pa kličem iz glavne »main()« funkcije:

```
def main():
  while True:
    print(read_temp(addr1))
    print(read_temp(addr2))
    time.sleep(1)
```

3.5 UART

UART (angl. Universal asynchronous receiver/transmitter) je del računalniške strojne opreme, ki skrbi za preslikavo med paralelno in serijsko komunikacijo. Pogosto se uporablja skupaj s standardoma RS-232 in RS-485. Ta predpisa določata stanja na linijah, torej fizikalno predstavitev visokega in nizkega logičnega nivoja. RS-232 se uporablja za krajše razdalje (do 15 m), RS-485 pa za daljše (tudi do 1200 m). Tu nas omejuje pravilo: daljša kot je linija, počasnejši mora biti podatkovni prenos.

UART vezje sestavi standardno zaporedje bitov, ki ga posreduje na vodilo. Slika 16 kaže obliko enega sporočila, kjer morajo biti najprej poslani manj pomembni (LSB, angl. Least Significant Bit) podatkovni biti.

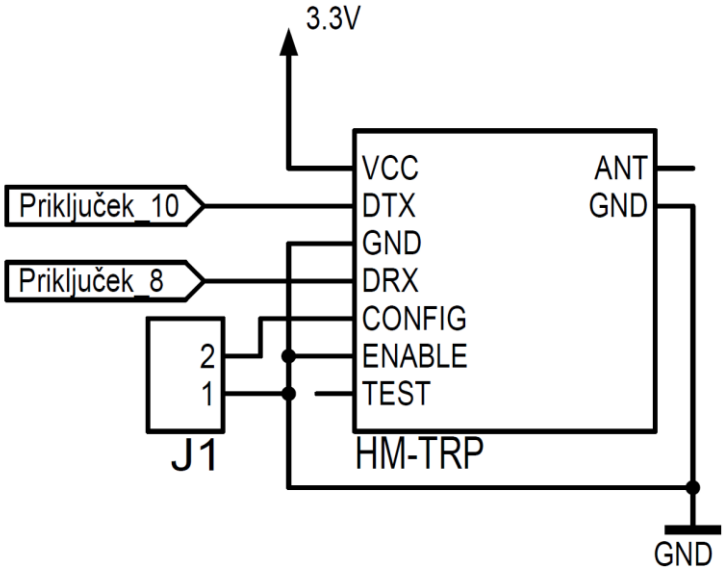
Št. bita	1	2	3	4	5	6	7	8	9	10	11
Namen bita	Start bit	5 - 8 podatkovnih bit-ov								Stop bit(a)	
Ime bita	Start	D0	D1	D2	D3	D4	D5	D6	D7	Stop	

Slika 16: Oblika UART sporočila

UART je običajno del nekega integriranega vezja, ki skrbi za komunikacijo s perifernimi napravami. Največkrat je že integriran v večini mikrokontrolerjev. Komunikacija poteka po dveh linijah. Naprave imajo priključek za oddajanje TX (angl. transmit data) in priključek za sprejemanje RX (angl. receive data) podatkov. Komunikacijsko vodilo nima takta ure, zato morajo biti parametri enako nastavljeni v vseh napravah. V kolikor imamo povezani le dve napravi, liniji le obrnemo (zakrižamo) TX –> RX, RX –> TX. Povezava je lahko:

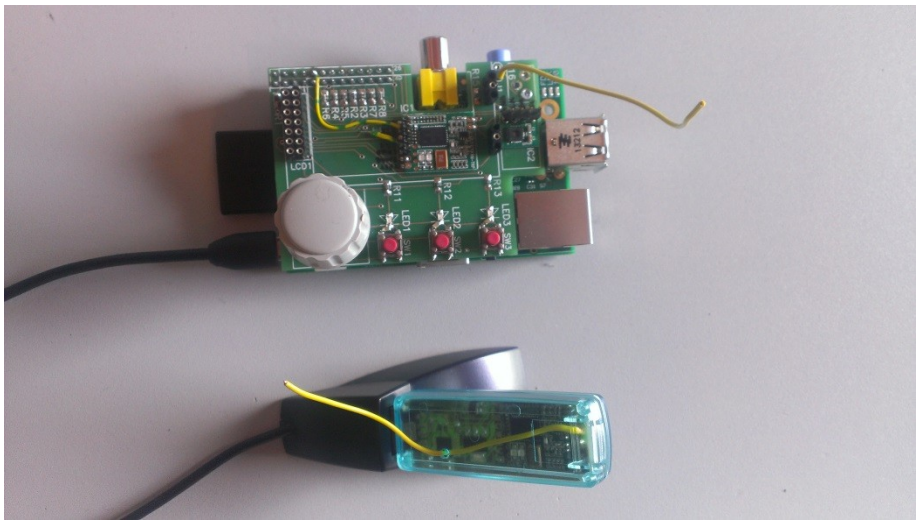
- Simplex – le naslavljanje oddaljene naprave, brez možnosti odgovora,
- Full Duplex – obe napravi lahko pošiljata in prejemata podatke istočasno,
- Half Duplex – najprej pošilja ena naprava, nato druga.

Uporabil sem Half Duplex. Ker sem za komunikacijo uporabil radijsko povezavo, Full Duplex ni bil mogoč, ker komunikacijski modul v času oddajanja sprejem ugasne. Uporabil sem radijsko-frekvenčni modul HM-TRP in ga priključil, kot kaže Slika 17.



Slika 17: Priklop HM-TRP na Raspberry Pi

J1 je kratkostičnik, s katerim omogočim spreminjanje nastavitev modula. Drug HM-TRP modul je bil kot USB -> HM-TRP adapter[5] priključen na osebni računalnik.



Slika 18: HM-TRP priključen na Raspberry Pi in osebni računalnik

Za razliko od ostalih načinov prenosa je UART že dostopen in omogoča prijavo v sistem prek konzole. UART vmesnik se nahaja kot znakovna datoteka »/dev/ttyAMA0«. Ta naprava pripada skupini »tty« v kateri nisem bil, zato sem se vanjo dodal z ukazom:

[5] Adapter HM TRP_UART_USB izdelan: http://trgovina.svet-el.si/productdetail.php?prodid=18182

```
$ sudo usermod -a -G tty pi
```

Privzeta hitrost za komunikacijo s HM-TRP-jem je 9600 bitov na sekundo. Na Raspberry Pi-ju sem moral v datoteki »/etc/inittab« spremeniti hitrost komunikacije za prijavo z 115200 na 9600 bitov na sekundo. Ravno tako sem moral spremeniti hitrost prenosa podatkov ob izpisovanju zagona sistema. Tako sem v datoteki »/boot/cmdline.txt« spremenil vrednosti z 115200 na 9600.
Na osebnem računalniku sem odprl terminalno okno, izbral prava vrata in hitrost prenosa podatkov nastavil na 9600 bitov na sekundo. Znova sem zagnal Raspberry Pi in v konzoli osebnega računalnika sem lahko odčitaval podatke, ki jih Raspberry Pi izpisuje ob zagonu. Po koncu nalaganja sistema se je izpisalo:

```
login as:
```

Lahko sem se prijavil v sistem in z zelo dobro uporabniško izkušnjo pregledoval datoteke. Seveda sem hotel z osebnim računalnikom komunicirati tudi prek programskega jezika Python. Da bi to dosegel, sem moral v prej omenjenih datotekah na Raspberry Pi-ju onemogočiti uporabo UART-a. Po ponovnem zagonu je bil UART na voljo za splošno uporabo. Najprej sem v Linux lupini nastavil hitrost vmesnika na 9600 bitov na sekundo in preveril njegovo delovanje z ukazom:

```
$ stty –F /dev/ttyAMA0 9600
$ echo "Hello PC" > /dev/ttyAMA0 #pisanje na UART vodilo
$ cat /dev/ttyAMA0 #branje iz UART vodila
```

UART preverjeno deluje. Nato sem namestil Python modul:

```
$ sudo apt-get install python3-serial
```

Zagnal sem urejevalnik kode za Python, uvozil pravkar nameščen modul in ustvaril nov objekt, ki mi bo služil za komunikacijo po UART vodilu ter mu nastavil osnovna parametra:

```
import serial
ser = serial.Serial("/dev/ttyAMA0")
ser.baudrate = 9600
ser.timeout = 1
```

Parameter »timeout« prejme sekunde in preprečuje predolgo čakanje, če na liniji ne bi bilo podatkov. Nize lahko pišem na linijo prek ustvarjenega objekta tako, da jih najprej pretvorim v ASCII kodo, nato pa v posamezne bajte:

```
ser.write(byte("Hello PC\n\r".encode("ascii"))
```

Branje je bolj preprosto:

```
ser.read() #prebere 1 bajt
```

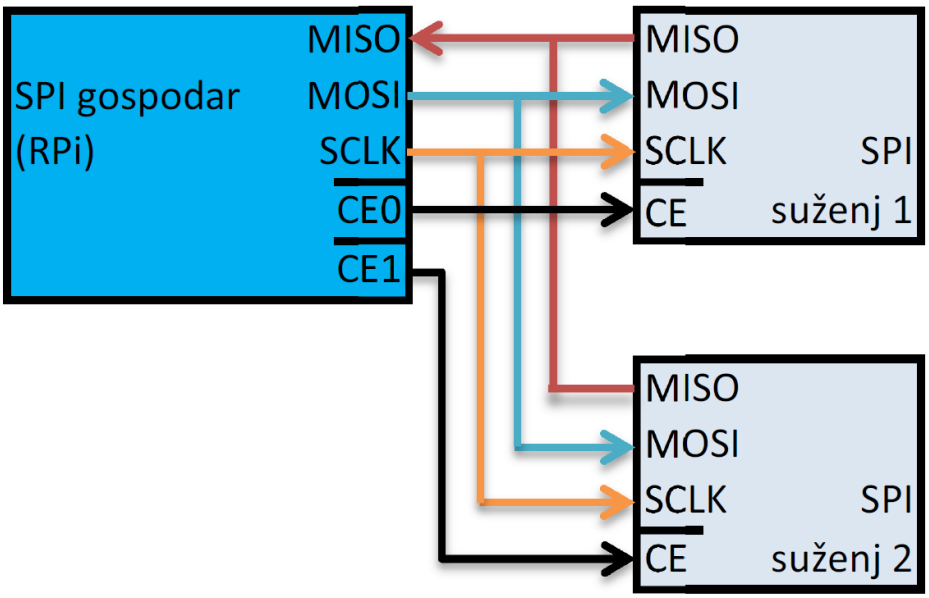
Seveda sem moral pred branjem na osebnem računalniku vpisati neko črko, ki sem jo lahko nato prebral iz medpomnilnika UART, vmesnika na Raspberry Pi-ju.

3.6 SPI

Za razliko od UART-a je SPI (angl. Serial Peripheral Interface) sinhrona komunikacija, poimenovana s strani Motorole. Deluje v Full Duplex načinu in je namenjena komunikaciji na krajših razdaljah z enim gospodarjem. Vsak suženj mora imeti svojo »chip enable« linijo (CE), po kateri ga gospodar omogoči tako, da jo spusti na nizek nivo. Če je suženj omogočen, lahko komunicira z gospodarjem, v nasprotnem primeru pa mora ignorirati dogajanje na liniji s taktom ure (SCLK), ki ga vedno daje gospodar, kot tudi dogajanje na podatkovnih linijah (MISO in MOSI). Gospodar izbere le enega sužnja naenkrat. Omenjene oznake niso edine. Za iste linije se uporabljajo naslednji sinonimi:

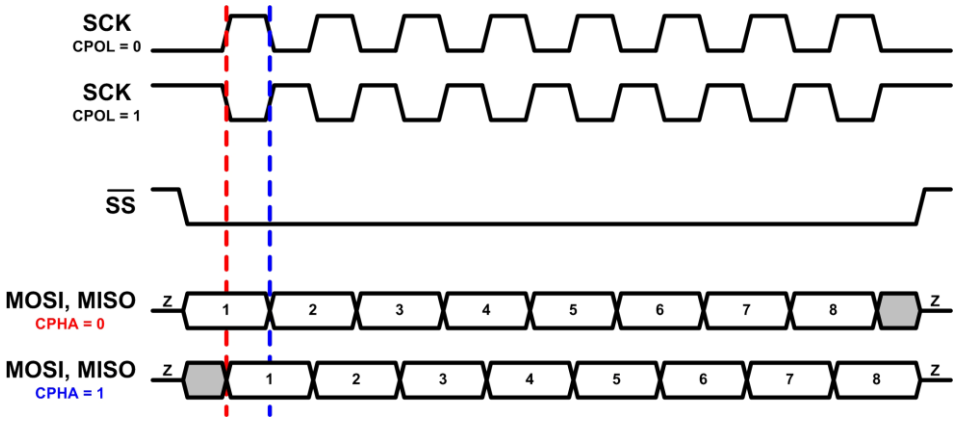
MISO (angl. Master In, Slave Out): SOMI, SDI, DI, DIN, SO, MRST.
MOSI (angl. Master Out, Slave In): SIMO, SDO, DO, DOUT, SI, MTSR.
SCLK (angl. Serial Clock): SCK, CLK.
CE (angl. Chip Enable): nCS, CS, CSB, CSN, nSS, SS, STE, SYNC.

Kot že kratica pove, liniji MISO in MOSI obrnemo, kot kaže Slika 19.



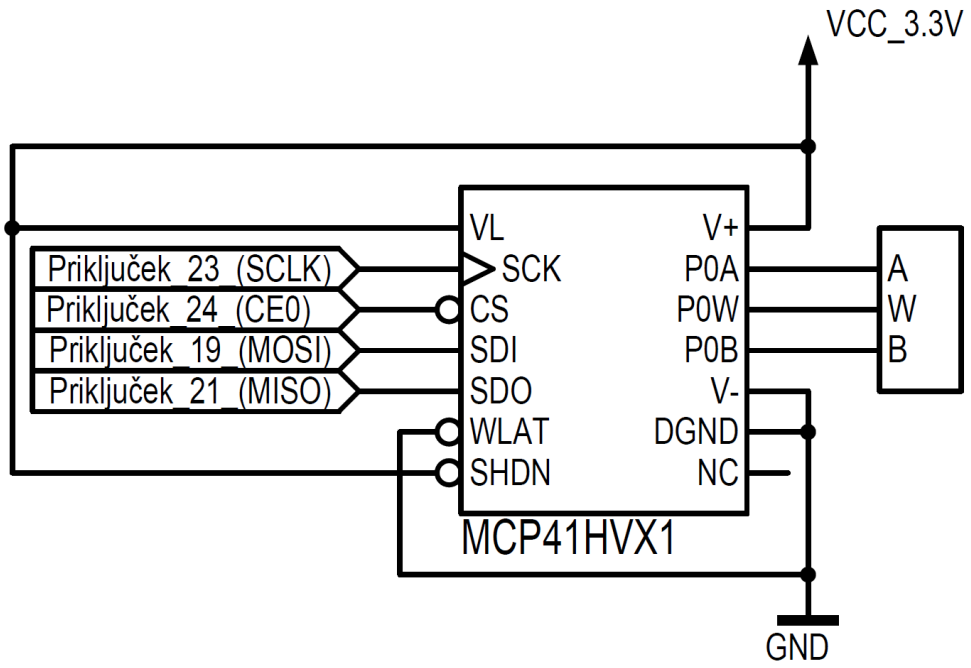
Slika 19: Priključitev sužnjev pri protokolu SPI

Omeniti moram tudi načine prenosa. Tukaj so možnosti (0, 0), (0, 1), (1, 0), (1, 1). To so štirje pari bitov CPOL (angl. Clock Polarity) in CPHA (angl. Clock Phase). Kot je razvidno s Slika 20, govorimo o načinu takta ure in načinu vzorčenja. Vzemimo za primer drugi par CPOL=0, CPHA=1. Iz diagrama na sliki 3 sledi, da CPOL=0 pomeni mirovno stanje takta ure v '0', CPHA=1 pa pomeni, da je spreminjanje podatkov ob pozitivnem robu, odčitavanje podatkov pa ob negativnem robu takta ure.



Slika 20: Časovni diagram SPI

Za prikaz delovanja SPI-ja na Raspberry Pi-ju sem uporabil digitalni potenciometer, Microchip-ov MCP41HV31, z največjo upornostjo 50 kΩ. Čip sem priključil, kot kaže Slika 21.



Slika 21: Priklop MCP41HV31 na Raspberry Pi

SPI priključke sem priključil kot določa tehnična dokumentacija RPi-ja; za omogočanje čipa sem izbral Raspberry Pi priključek "CE0". Funkcionalnost ostalih priključkov je sledeča:

- »WLAT« (angl. Write Latch): priključek omogoča ('0') ali onemogoča ('1') posodabljanje stanja potenciometra glede na vrednost v pomnilniku. Želeno vrednost v pomnilniku torej lahko spreminjamo, dejanska upornost pa se posodobi, ko je vrednost »WLAT« na logični '0'.
- »SHDN« (angl. Shutdown): v logični '1' deluje kot potenciometer, če je v logični '0', se priključek »P0A« oddvoji od uporovnega omrežja, priključka »P0B« in »P0W« se kratko skleneta.
- »VL« in »DGND«: napajanje krmilja uporovnega omrežja – potenciometra. Priključeno na 5V enosmerne napetosti in na GND.
- »V+« in »V-«: napajanje uporovnega omrežja – potenciometra. Priključeno na enak potencial kot napajanje krmilja. S tem smo določili, da na potenciometru ne bo višjih napetosti kot 5V.
- »P0A«, »P0W« in »P0B«: P0A in P0B sta skrajna priključka potenciometra, P0W pa je spremenljiv. Če želimo vrednost potenciometra sorazmerno vrednosti pomnilnika, priključimo ohm meter med priključka P0B in P0W.

Prikazal sem opis in priklop strojne opreme. Sedaj bom prikazal tudi, kaj je potrebno storiti na Raspberry Pi-ju, da lahko s potenciometrom komuniciramo iz Python-a. Najprej sem ga moral odstraniti s črne liste. To sem storil tako, da sem v datoteki »/etc/modprobe.d/raspi-

blacklist.conf« zakomentiral vrstico »blacklist spi-bcm2708«. Po ponovnem zagonu Linux-a sem preveril, če je SPI na voljo:

```
$ ls –l /dev/spidev*
```

Izpis je bil sledeč:

```
crw------- 1 root root 153, 0 Jan  1  1970 /dev/spidev0.0
crw------- 1 root root 153, 1 Jan  1  1970 /dev/spidev0.1
```

Na voljo imam dve SPI napravi. Razlikujeta se le v zadnji številki '0' in '1', ki predstavljata nožice Raspberry Pi konektorja »CS0« in »CS1«. Opazil sem tudi, da je zopet težava z lastništvom. Tokrat sem spremenil lastništvo datotek:

```
$ sudo chown `id -u`.`id -g` /dev/spidev0.*
```

Za delo s SPI-jem v Pythonu sem na spletu našel modul »quick2wire«. Ko sem ga namestil, sem iz njega najprej uvozil vse funkcije:

```
from quick2wire.spi import *
```

Nato sem v tehnični dokumentaciji poiskal ukaze za delo z MCP41HV31. Najbolj pomembne so se mi zdele naslednje štiri:

- povečaj vrednost v registru upornosti za ena,
- zmanjšaj vrednost v registru upornosti za ena,
- preberi trenutno vrednost registra upornosti,
- vpiši določeno vrednost v register upornosti.

Omenjene ukaze sem za lažje rokovanje shranil v spremenljivke:

```
incr_comm = 0b00000111
decr_comm = 0b00001011
read_comm = 0b00001111
writ_comm = 0b00000011
```

Naredil sem nov objekt, ki bo omogočal komunikacijo preko SPI-ja:

```
dev0 = SPIDevice(0, 0) #CE0
```

Digitalni potenciometer, ki sem ga priključil na Raspberry Pi, je sedembiten. To pomeni, da ima med 0 Ω in 50 kΩ 128 različnih vrednosti. Maksimalno upornost torej predstavlja šestnajstiška vrednost hex7F. Za testiranje potenciometra sem na priključka P0W in P0B priključil ohm meter. Najprej sem iz pomnilnika potenciometra prebral trenutno nastavljeno upornost z ukazom:

```
res = dev0.transaction(writing_bytes(read_comm),
      reading(1))
```

Vrednost se je ujemala z vrednostjo na ohm metru. Potem sem upornost povečal z nekajkrat poslanim ukazom:

```
dev0.transaction(writing_bytes(incr_comm))
```

Upornost naj bi se povečala za 393,7 Ω. Upornost seveda ni čisto taka, kot sem jo teoretično izračunal, ampak pride do odstopanj. Preizkusiti sem želel tudi velikokrat potrebno funkcijo nastavljanja potenciometra na točno določeno vrednost. To sem naredil s kodo:

```
dev0.transaction(writing_bytes(writ_comm, 0x3F))
```

Šestnajstiška vrednost hex3F predstavlja ravno polovico razpona, torej 25 kΩ. Tudi ohm meter je prikazal približno toliko. Odstopanja so bila okrog 300 Ω, kar pri tako velikih upornostih naj ne bi bilo kritično.

Poglavje 4

Sklepne ugotovitve

V diplomskem delu nisem opisal vseh možnosti za priklop perifernih naprav. Opisal sem le najpogosteje uporabljene. Za vsako dotično napravo je potrebno ugotoviti, katere registre moramo naslavljati.
Imel sem nekaj težav s strojno opremo. Na Raspberry Pi-ju mi je sistem večkrat zamrznil. Nekajkrat sem moral celo ponovno namestiti operacijski sistem. Menim, da je Raspberry Pi namenjen za manj kompleksne sisteme. Mogoče bi bilo bolje in manj napak, če programov ne bi zaganjal z grafičnega načina. Opazil sem namreč tudi težavo, da je v konzoli ob vpisu ukaza »ls« neko datoteko v mapi izpisalo, v grafičnem načinu pa je v oknu ni bilo. Če bi delal sisteme avtomatizacije, mislim, da bi bilo bolj smotrno uporabiti mikrokontroler (npr. AVR), ki bi upravljal vhode/izhode in bi komuniciral z Raspberry Pi-jem. Ta bi nato služil kot uporabniški vmesnik – bodisi grafični bodisi kot strežnik.

S perifernimi napravami sem bil v celoti zadovoljen. Nisem pričakoval, da bo z oddaljenimi napravami tako enostavno komunicirati in odčitavati njihovo stanje oz. izmerjene veličine. Senzorji so delovali brez napak.

Literatura

[1] 1-Wire. Dostopno na:
http://en.wikipedia.org/wiki/1-Wire (junij 2014)

[2] Hitachi HD44780 LCD controller. Dostopno na:
http://en.wikipedia.org/wiki/Hitachi_HD44780_LCD_controller (junij 2014)

[3] I2C. Dostopno na:
http://en.wikipedia.org/wiki/I%C2%B2C (junij 2014)

[4] Python(programming language). Dostopno na:
http://en.wikipedia.org/wiki/Python_(programming_language) (junij 2014)

[5] Quick2Wire. Dostopno na:
http://quick2wire.com/ (junij 2014)

[6] Raspberry Pi. Dostopno na:
http://en.wikipedia.org/wiki/Raspberry_Pi (junij 2014)

[7] Tehnična dokumentacija krmilnika Hitachi HD44780. Dostopno na:
https://www.sparkfun.com/datasheets/LCD/HD44780.pdf (junij 2014)

[8] Serial Peripheral Interface Bus. Dostopno na:
http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus (junij 2014)

[9] Universal asynchronous receiver/transmitter. Dostopno na:
http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter (junij 2014)