Tallinna Ülikool
Digitehnoloogiate Instituut

WI-FI-VÕRGUS TOIMIVATE ANDURITE LOOMINE JA SEADISTAMINE

Seminaritöö

Autor: Karl Talumäe
Juhendaja: Jaagup Kippar

Autor: ..... „ ..... „ 2016
Juhendaja: ..... „ ..... „ 2016
Instituudi direktor: ..... „ ..... „ 2016

Tallinn 2016

Autorideklaratsioon

Deklareerin, et käesolev seminaritöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

.....	.....
(kuupäev)	(autor)

Sisukord

Sissejuhatus

Käesolevas töös annan ülevaate, kuidas oleks nii teoreetiliselt ja ka praktiliselt võimalik luua Wi-Fi võrgus toimiv andur, kasutades võimalikult vähest ressurssi. Sarnased andurid on kasutusel paljudes tehastes ning süsteemides, kus juhtmevaba monitooring on tähtsal kohal.

Käesolev töö ei sisalda täielikku infot, kuidas luua töötav seade ja seda seadistada ning efektiivselt igapäevaselt kasutada. Pigem on tegemist eksperimentaalse katsetusega ning prototüübi loomisega. Töö eesmärgiks on leida sobivad seadmed, mis omavahel ühilduks ja lahendada võimalikult palju projektiga tekkivaid probleeme ning leida nendele mõistlik lahendus.

Kuna süsteemi südameks ehk kontrolleriks olen valinud Wi-Fi mooduli ESP8266, annan esmalt ülevaate seadmest ning selle võimekusest.

Järgmiseks annan ülevaate teiste loodud lahendustele ja katsetustele. Ma ei kasutanud neid kui juhendeid enda seadme loomiseks, vaid võtsin nendest inspiratsiooni ja üritasin teiste poolt tehtud vigu ise mitte korrata.

Töö järgmises osas on arutelu eri seadmete testimise ning koostamise teemadel ning töö käigus minu poolt loodud seadme erinevate arendusetappide kirjeldus. Kirjeldan ülevaatlikult iga seadme loomisega seotud etappi ja toon välja tähtsamad arengud ning leitud vead, mida protsessi käigus õppisin ning mida tuleks sarnast projekti tehes teisiti teha.

Töö vajalikkus on põhjendatud asjade interneti ja juhtmevabade andurite üha suureneva potentsiaaliga (Lisa 1.) ning isikliku huviga proovida luua soodsam seade võrreldes turul pakutavatega. Minu eesmärgiks oli kogeda seadme disainimise ning loomisprotsessi ja oma tööga nii endale kui teistele robootika ning monitooringuhuvilistele midagi pakkuda.

Kirjutis on eelkõige suunatud neile, kes plaanivad ise mõne monitooringlahenduse disainida ja ka koostamise ette võtta, aga ka neile, kes tahavad tutvuda, milline näeb välja ühe monitooringseadme disainimine ja prototüüpimine.

1. ESP8266 tutvustus

ESP8266 on üks kõike kompaktsemaid ja integreeritumaid Wi-Fi kiipe, mida turult leiab. Kiibi suuruseks on vaid 5mm x 5mm ning töötamiseks vajab väheseid kiibiväliseid lisakomponente, miinimum 7 lisakomponenti. Eelkõige on ESP8266 loodud kasutamiseks mobiilsetes seadmetes, kantavates elektroonikaseadmetes ning asjade interneti lahendustes. Kiibi südameks on 32-bitine Tensilica mikrokontroller, mille kõrvalt leiab veel standardse digitaalse välisseadme liidese, antenni lüliti, voolu võimendi, madala müratasemega vastuvõtja võimendi ning voolufiltrite ja toitehalduse moodulid. Mikrokontrolleri protsessori kiiruseks on 160MHz ning lisaks mikrokontrollerile on kiibil kuni 9 sisend-/väljundpesa, millest 4 on pulsilaiusmodulatsiooni võimelised. Võrreldes vanemate ESP8266 mudelitega on uuel versioonil 8 megabitti mälu. Kiibi eelisteks võrreldes teiste sarnaste toodetega on lai töötamistemperatuuri vahemik, milleks on -40ºC kuni +125ºC, ning väga madal voolu tarbimine (Espressif, 2016).

ESP8266 puhul on tegemist vaid Wi-Fi kiibiga, kuid kuna tavakasutajal ei ole kiibiga mugav erinevaid elektroonikalahendusi luua, pakutakse turul mitmeid trükkplaate, kus Wi-Fi kiip on juba integreeritud ning juurde on lisatud vajalikud komponendid kiibi edukaks toimimiseks.

Kõige levinum ESP8266 kiibiga trükkplaadi tootja on Al-Thinkers. Teised tuntumad tootjad on Olimex, Wireless-tag, Quilianer, Smarttime.cn, Espressif Systems.

Trükkplaadi versioone on palju erinevaid. Arvestades originaaltootjaid, leiab turult kuni 21 erinevat trükkplaadi versiooni. Kõige minimalistlikum ja odavam ESP8266 kiibiga trükkplaat on ESP-01, mille hind jääb keskmiselt vahemikku 2€ kuni 7€. Selle trükkplaadi laiuseks on 24.8mm ning pikkuseks 14.3mm ja pakub 2 sisend-/väljundpesa. Sellele vastandiks on kõige võimekam versioon nimega ESP-12E, mis pakub 11 sisend-/väljundpesa ning mille mõõtmeteks on 24mm x 16mm. Enamus mudelitel asub Wi-Fi antenn trükkplaadil, kuid on ka mudeleid, kus on võimalik antenne vahetada (stulander, 2016).

Tegemist on eelkõige asjade interneti ja kantavate elektroonikalahenduste jaoks loodud kiibiga, mis on odav ja mida on võimalik eri programmeerimiskeeltega seadistada. Soetades endale ühe ESP8266 mudeli ei tule kaasa peale trükkplaadi enda midagi ning kasutaja peab ise valima endale sobivad töövahendid. Kuna seadme populaarsus on ajaga kasvanud, on ühiskonnas tekkinud erinevad vabavara tarkvarad, millega seadet programmeerida.

Igasse sisend-/väljundpesasse on võimalik mõni digitaalset signaali väljastav elektroonikakomponent ühendada ja programmikoodis iga pesa kohta määrata, kas tegemist on sisendi või väljundiga. See annab kiibile teada, kas soovitakse lugeda sisse tulevaid andmeid või saata seadmele välja programmis kirjutatud käsklusi. Võimsamatelt ESP8266 trükkplaatidelt leiab analoog-digitaal signaali konverteri, mis teoorias võimaldab kasutada analoog andureid.

Kõige enam kasutatakse ESP8266 kiibi programmeerimiseks Lua keelt. Selle jaoks on loodud ka IDE taoline tarkvara nimega Lualoader (Jennings, 2014), mis võimaldab koodi laadida ESP8266 seadmele. Vaikimisi on ESP8266 kiibil AT tarkvara, mille programmeerimisvõimalused võrreldes Lua keelega on piiratud. Kiibile Lua tarkvara paigaldamiseks on loodud tarkvara NodeMCU Flasher (NodeMCU, 2015). Kuna ESP8266 seadmetel puudub USB tugi, on tarkvara ja koodi laadimiseks vajalik USB UART konverter, mis kirjutab kiibile andmeid ühe biti kaupa (Jimb0, 2010).

Üheks esimeseks katseks veendumaks et kiip ja trükkplaadi ühendused töötavad, on hea proovida LED-lambi süütamist. Eeldades, et kiibile on eelnevalt paigaldatud Lua tarkvara, on esmalt vajalik ühendada USB UART konverter arvutiga ja Lualoader tarkvaras valida sellele vaba port. Koodi laadimine ESP8266 seadmele võib võtta rohkem kui 250mA, seega on soovituslik kasutada USB UART konverteri toite asemel välist 3.3V toiteallikat. Sellega on tagatud stabiilne koodi laadimine seadmele. Kindlasti ei tohiks unustada, et kõik maandused peavad olema ühendatud ühte vooluringi, vastasel juhul jääb see poolikuks. Esmalt on vajalik Lua koodis defineerida LED-lambi pesa, peale mida määrata pesa väljundiks. Seejärel on võimalik lihtsa ühe käsuga määrata, et lamp põleks, kirjutades vastava väljundpesa väärtuseks HIGH. Laadides koodi ESP8266 seadmele ning ühendades pinge vooluringi, hakkab LED lamp põlema.

Sarnaselt on võimalik ühendada paljusid erinevaid elektroonikaseadmeid, mis mahuvad pinge- ja vooluvahemikku, milledeks on vastavalt 2.5-3.3V ja 12mA voolu ühe pesa kohta. Suuremate voolutugevuste korral, nagu ka minu projekti puhul, saab kasutada välist toiteallikat ja ESP8266 kiipi kasutada ainult loogika jaoks (Espressif_Kelly, 2015).

Lua keelel ei ole kindlat struktuuri, kuid peamine ülesehitus on sarnane iga teise programmeerimiskeelega kasutades muutujaid, konstante, funktsioone, tsükleid ja teisi tavapäraseid omadusi.

2. Teiste poolt ehitatud seadmed

Enne seadmete soetamist ja praktilise poolega alustamist uurisin mitmeid allikaid, et saada ülevaade, mida ESP8266 endast pakub ning mida on teised inimesed seadmega suutnud luua. Allikate puhul oli tegu nii foorumis leiduvate ehitusblogi tüüpi artiklitega kui ka Youtube keskkonnas olevate videotega. Käesolevas peatükis kirjeldan mõnda minu tööd enim mõjutanud teostust.

2.1. DHT22 andurilt temperatuuri ja niiskustaseme lugemine ning veebis kuvamine

Artiklis pealkirjaga *Making a NodeMCU Lua ESP8266 weather station* (Simon, 2015) on kirjeldatud protsessi, kuidas ühendada temperatuuri ja niiskustaseme andur ESP8266 kiibi külge ning suudab kuvada saadud andmeid veebilehel. On näha, et artikli autor kasutab seadmete ühendamiseks maketeerimislauda ja juhtmeid, mis tagab komponentide lihtsa ümberpaigutusvõimaluse. See omakorda tähendab, et tegemist on prototüübiga ning loodud seade ei ole mõeldud reaalses keskkonnas kasutamiseks.

Esmalt ühendab autor DHT22 anduri pesa *pin1* positiivsesse vooluvõrku, kus pingeks on 3.3v, *pin2* ühendab ESP8266 kiibiga, kasutades *gpio4* pesa ning *pin4* ühendab maandusesse. Peale ühenduste loomist on võimalik Lua koodi kirjutama asuda.

Esmalt defineeritakse koodis ära Wi-Fi võrgu parameetrid ehk võrgu nimi ning parool. Samuti määratakse ära IP, kust on hiljem võimalik andmeid näha. Seejärel antakse kiibile käsklus ühenduda ette võrku ja kuvada andmeid defineeritud IP peal. Järgmiseks on vaja määrata anduri sisend-/väljundpesa. Kuna andur vajab toimimiseks eraldi teeki, tuleb see sisse importida. Sellega on anduri väärtuste lugemiseks vajalik kood valmis ning on vaja käivitada defineeritud IP peal veebiserver. Eelviimaseks sammuks on DHT22 andurilt andmete lugemine. Viimasena luuakse lihtne HTML kood, mida kuvada veebiserveris koos andurilt saadud andmetega.

Tulemuseks on ühekordne andmete lugemine andurilt ja saadud info kuvamine staatilisel IP aadressil, mida on võimalik vaadata iga seadme pealt, mis on ühendatud samasse võrku.

Projektist sain teadmisi DHT22 ühendamise ja teegi kasutamise kohta. Samuti seda, kuidas

küsida andmeid andurilt ning millisel kujul need tagastatakse.

2.2. BMP180 õhurõhuanduri teek

Soovisin kasutada enda projektis õhurõhuandurit ning leidsin, et BMP180 on täpne ja sobilik andur minu projekti vajadusteks. Kahjuks eelnevaid projekte ESP8266 ja BMP180 anduri kohta internetiavarustest ei leidnud ning ainukeseks näiteks oli teek (NodeMCU Team, 2015), mida on vaja õhurõhuanduri kasutamiseks.

Uurides teeki lähemalt, sain teadmisi, mis formaadis andmeid on võimalik välja küsida ning kuidas neid erinevatesse õhurõhuühikutesse teisendada. Samuti paranes arusaam, milliseid protsesse on vaja teha, et füüsiliselt andurilt saadud info muuta inimloetavaks.

3. Wi-Fi võrgus toimiva ilmajaama loomine

Selles peatükis annan etapphaaval ülevaate sellest, kuidas määratlesin esmase plaani, kui palju see töö käigus muutus ning mida tegin teisiti võrreldes algul plaanituga. Alapeatükkides kirjeldan erinevaid protsessietappe, mida tuli läbida seadme loomiseks ja nendest saadud tulemusi.

3.1. Esmane plaan

Esialgseks plaaniks oli luua Wi-Fi võrgus töötav ilmajaam, mis edastab anduritelt saadud temperatuuri, niiskuse ning õhurõhu info andmebaasi ning kuvada saadud andmeid välja veebilehel. Seadmeks soovisin kasutada ESP8266 kiipi, mille külge ühendada vajalikud andurid ja toiteallikas. Juhul, kui seade töötab laitmatult, luua enda kavandatud trükkplaat, kuhu saaks kõik kasutatavad komponendid kokku ühendada. Tulemuseks oleks odav ning kompaktne Wi-Fi võrgus töötav ilmajaama prototüüp, mille andmeid oleks võimalik jälgida igalt sobivalt seadmelt. Töö ettevalmistusetapis otsisin esmalt vajalikku informatsiooni ning mõtlesin läbi töö teostamiseks vajalikud etapid.

Kõige esmalt oli vajadus soetada ESP8266 trükkplaat, et saaks seadme võimalustega lähemalt tutvuda. Valisin ilmajaama südameks just ESP8266 kiibi, kuna seade on sarnaste toodetega võrreldes üks kõige odavamaid ning sellel on Wi-Fi tugi. Kuna seadme mudeleid on üle 20 erineva oli vaja teha üpriski suuremahuline uuring, selgitamaks välja, millise mudeliga oleks hea alustada ning millist mudelit edasise arengu käigus kasutada. Samuti oli vaja välja selgitada, millist temperatuuri, niiskuse ning õhurõhuandurit kasutada, et tagada ESP8266 ja andurite omavaheline ühildumine.

Otsustasin, et esmalt üritan kõik kasutatavad komponendil üksikult tööle saada ning õnnestumise korral proovin kõik seadmed omavahel ühilduma saada.

3.2. Kasutatavad elektroonikakomponendid ja nende hankimine

Esmalt koostasin nimekirja vahenditest ja seadmetest, mida mul oleks esmalt vaja, et saaks ESP8266 katsetamistega alustada. Alustuseks, uurisin välja, milliseid seadmeid on võimalik Eesti elektroonikakauplustest hankida ning mis hinnaga. Raske oli valida enda projekti jaoks kõige sobivamat ESP8266 mudelit. Teadsin, et seade võiks vähemalt mõnekümne meetri

raadiuses suuta ennast Wi-Fi võrku ühendada. Leidsin, et esimesteks katsetusteks oleks sobiv seade ESP-01. Tegemist on kõige lihtsama ja odavama ESP8266 mudeliga. Kuna Hiinast tellides oleks seadme kättesaamiseni läinud oluliselt kauem aega, kui Eesti edasimüüjalt ostes, otsustasin teise valiku kasuks ning soetasin seadme Eestist 7€ eest.

Tarkvara ja koodi laadimiseks ESP8266 seadmele on vaja USB UART konverterit. Esmalt proovisin Tallinna Ülikooli robootika ruumis olevat konverterit nimega CP2102. Seadmel on 5 pesa, 3.3V, TXD, RXD, GND, +5V. Proovisin selle seadmega saada ühendust ESP8266 kiibiga, kuid edutult. Kahtlustasin, et asi on ühes kindlas seadmes, mis võib olla vigane ning proovisin teise konverteriga samast mudelist. Kahjuks ka selle seadmega ei saavutanud edukat ühendust. Otsustasin proovida mõnda teist USB UART konverterit. Leidsin seadme nimega FT232RL, mis oli küll eelmisest seadmest kallim, ning juba esimesel katsetusel sain seadet kasutades ESP8266 kiibiga eduka ühenduse.

Projekti komponentide testimine lükkus just USB UART konverteri probleemide tõttu pikalt edasi. Keeruline oli aru saada, millises komponendis oli viga. Kuna igat komponenti oli minul vaid üks eksemplar ning Eesti edasimüüjatelt ei olnud võimalik samu komponente soetada, muutus projekti algus üpriski frustreerivaks.

3.3. Andurid

Tahtsin kasutada kolme erinevat parameetrit välise keskkonna mõõtmiseks. Temperatuuri ja õhuniiskuse tase on väga kasulikud andmed, et hinnata nii mineviku, oleviku kui ka tuleviku ilma. Õhurõhu teadmine parandab tuleviku ilmaennustamise täpsust. Teadsin, et temperatuuriandur peab suutma mõõta temperatuuri alla 0ºC, kuna Eestis langeb igal talvel temperatuur alla selle. Samuti soovisin, et andur oleks võimalikult täpne. Leidsin enda jaoks sobiva anduri DHT22, mille puhul on tegemist temperatuuri ja õhuniiskuse mõõtmiseks mõeldud seadmega. Seade töötab pingevahemikus 3-5V, suudab lugeda õhuniiskuse taset vahemikus 0-100% täpsusega 2-5%, töötab ja mõõdab temperatuuri vahemikus -40ºC kuni +80ºC täpsusega ±0.5ºC ning andmete lugemise ajal kasutab maksimaalselt 2.5mA. Tähtis faktor oli ka anduri töötamiseks vajaminev pinge. Kuna ESP8266 seade vajab 3.3V, soovisin et andur töötaks sama pingega. Lähtuvalt sellest, et DHT22 andur väljastab nii temperatuuri kui ka õhuniiskuse andmeid, sain ühe anduriga kaks parameetrit kätte. Õhurõhu mõõtmiseks olin kahe anduri valimise vahel, BMP180 ja BMP085. Üks oli teisest kaks korda soodsam, kuid tänu

sellele ka ebatäpsem. Mõlemad andurid kasutavad sama teeki, seega otsustasin kallima ja täpsema anduri kasuks ja valisin BMP180. Andur on mõeldud eelkõige täpse õhurõhutaseme mõõtmiseks. Seade töötab 3-5V pingevahemikus, suudab lugeda õhurõhu taset vahemikus 300-1100 hektopaskalit täpsusega 0.03 hektopaskalit ning töötab temperatuurivahemikus -40ºC kuni +85ºC.

Kuna suuremat huvi pakkus temperatuuri lugemine, siis otsustasin esmalt soetada DHT22 anduri, tellides selle Suur-Britanniast.

3.4. Esmane ESP8266 katsetus

Kõige esimeseks katsetuseks tahtsin saada LED-lambi vilkuma, kasutades Lua koodi. Selleks ühendasin ESP8266 mudeli ESP-01 USB UART konverteri külge ning laadisin kiibile koodi (Koodinäide 3.1).

```
-- Määran lambi pesa
local pin = 5
local status = gpio.LOW
-- 1 sekundiline taimer
local duration = 1000
-- Lambi aktiveerimine
gpio.mode(pin, gpio.OUTPUT)
gpio.write(pin, status
-- Loon intervalli
tmr.alarm(0, duration, 1, function ()
if status == gpio.LOW then
status = gpio.HIGH
else
status = gpio.LOW
end
gpio.write(pin, status)
end)
```

Koodinäide 3.1: LED-lambi vilgutamine

Koodi eduka laadimise järel kiibile eemaldasin USB UART konverteri ning ühendasin ESP8266 *gpio1* pesasse LED-lambi pluss poole ja GND pesasse lambi maanduse. Peale ühenduste kontrollimist võis voolringi toitega ühendada. Toiteallikaks kasutasin USB UART

konverteri +3.3V pesa. Tulemuseks sain LED-lambi, mis vilgub ühe sekundilise intervalliga (Robo India, kuupäev puudub).

3.5. Temperatuuri ja niiskusanduri DHT22 katsetused

Peale DHT22 anduri kättesaamist soovisin koheselt proovida, kui hästi ühildub andur ESP8266 seadmega. Esimese sammuna tahtsin saada kätte õhutemperatuuri. Alguses tekitas palju raskusi töötava koodinäite leidmine, kuna Lua tarkvara uuendustega aegade jooksul on käsklused muutunud, tänu millele ei töötanud enamus koodinäiteid. Teiseks suureks probleemiks oli teegi leidmine. Artiklites ei olnud enamasti kirjas, et andur vajab töötamiseks eraldi teeki, mistõttu kulus palju aega ise erineva koodi loomisele ning testimisele, kui ka internetist leitud koodinäidete proovimisele. Leides lõpuks sobiva ning töötava teegi (Lisa 6.), tekkis probleem ESP8266 mälumahuga. Nimelt DHT22 anduri teek kasutas töö käigus kogu mälu ära, mille tulemusena jooksis kiip kokku ja temperatuuri lugemine andurilt oli edutu. Lahenduseks leidsin minimaliseeritud koodiga teegi, mis vajas töötamiseks umbes 3 korda väiksemat mälu hulka. Kolmandaks probleemiks oli anduri *pin2* ja vooluringi GND vahele takisti ühendamine. Paljude artiklite näidetes oli välja toodud, et takisti kasutamine on vajalik ning vastasel juhul ei ole võimalik täpseid andmeid andurilt saada. Sama palju oli ka artikleid, kus takistit ei kasutatud, tänu millele kulus väga palju aega ise erineva koodi ja vooluringide testimisele. Peale mitmeid päevi lugemist ja testimist leidsin lõpuks internetist sobiva artikli koos näidiskoodiga (jescarri, 2015), mille tulemusena sain koostada enda testfaili (Koodinäide 3.2) ning laadisin selle ESP8266 kiibile.

```
DHT= require("dht22_min")
DHT.read(5)
temperature = DHT.getTemperature()
humidity = DHT.getHumidity()
t1 = temperature / 10
t2 = temperature % 10
print("Temperature: "..t1.."."..t2.."C")
DHT = nil
package.loaded["dht22_min"]=nil
```

Koodinäide 3.2: DHT22 anduri esmane katsetus

Koodi eduka laadimise järel kiibile ühendasin anduri *pin0* vooluringi 3.3V, *pin2* ESP8266

gpio0 pesasse ning *pin4* GND pesasse ehk maandusesse. Kontrollisin ühendused vooluringis üle ning lülitasin pinge sisse. Avasin Lualoaderi tarkvaras konsoolivaate ning tegin ESP8266-le taaskäivituse. Mõne hetke pärast oli konsoolivaates näha andurist loetud temperatuur.

Järgmiseks sooviks oli kuvada konsoolis nii temperatuur kui õhuniiskus. Kuna anduri teek sisaldas endas nii temperatuuri kui ka õhuniiskuse välja küsimise võimalust, oli selleks vaja ainult koodi muuta. Defineerisin juurde õhuniiskuse muutuja, lisasin 2 muutujat, et arvutada õhuniiskus kümnendiku täpsusega ning viimaseks kuvasin temperatuuri alla õhuniiskuse taseme välja. Laadisin koodi kiibile ning avasin konsoolivaate. Positiivse üllatusena sain teise parameetri ilma suurema vaevata kätte.

DHT22 anduri tööle saamisega läks üpriski palju aega, kuid peale aega nõudvaid pingutusi jäi andur stabiilselt tööle, mida loen kindlasti positiivseks tulemuseks nii tarkvara kui ka riistvara suhtes.

3.6. Õhurõhuanduri BMP180 katsetused

Olles eelnevalt saanud selgeks DHT22 anduri kasutamise, oli uue anduriga juba lihtsam alustada. Esimeseks ülesandeks oli lugeda andurilt õhurõhu tase ning väljastada tulemus konsooli. Anduri ühendused olid väga sarnased DHT22 anduriga, tänu millele ei kulunud sellele palju aega. Esmalt otsisin internetist andurile sobiva teegi (Lisa 7.) ja näitekoodi. Kahjuks ei leidnud BMP180 andurile minimaliseeritud teeki ning olin sunnitud kasutama versiooni, mis kasutab üpriski suurt mälumahtu. BMP180 anduril oli lisaks õhurõhule temperatuuri väljastamise võimalus. Kuna ma kasutan temperatuuri jaoks aga DHT22 andurit ning BMP180 temperatuuri andur on ebatäpsem, otsustasin seda mitte kasutada. Peale näitekoodi (Koodinäide 3.3) muutmist vastavalt enda vajadustele laadisin selle kiibile.

```
OSS = 1
SDA_PIN = 5
SCL_PIN = 4
bmp180 = require("bmp180")
bmp180.init(SDA_PIN, SCL_PIN)
bmp180.read(OSS)
p = bmp180.getPressure()
print("Pressure: "..(p).." Pa")
print("Pressure: "..(p * 75 / 10000).."."..((p * 75 % 10000) / 1000).." mmHg")
bmp180 = nil
package.loaded["bmp180"]=nil
```

Koodinäide 3.3: BMP180 esmakatsetus

BMP180 anduril on andmete jaoks kaks pesa, SDA ja SCL. Ühendasin ühe *gpio0* pesasse ja teise *gpio1* pesasse, peale mida oli vaja vaid ühendada +3.3V ja GND vooluringi pluss osasse ning maandusesse. Kontrollisin ühendused üle ning lülitasin pinge sisse. Avades konsoolivaate Lualoader tarkvaras nägin õhurõhu mõõtmise tulemusi nii hektopaskalites, kui elavhõbedasamba millimeetrites (Autor puudub, 2015). Lugesin anduri esmast katsetust edukaks.

3.7. Analoogandurite katsetused

Projekti alguses oli plaanis kasutada vaid kolme väliskeskkonna parameetrit. Liikudes projektiga edasi, leidsin, et tuule kiiruse ja suuna teadmine ning andmete salvestamine oleks samuti huvitav ning aastaaegade võrdlemise suhtes vajalik.

Uurisin erinevaid andureid ning leidsin, et kõik laialdaselt kasutatavad tuule kiiruse ning suuna andurid väljastavad andmeid analoogformaadis. Kuna ma ei olnud eelnevalt katsetanud analoogandurite kasutamist ESP-12E seadmega, otsustasin enne tuuleandurite soetamist teha mõned katsetused olemasoleva valgustundliku dioodanduriga.

ESP-12E kiibil on olemas üks analoog-digitaal signaali konverter, mis teoorias peaks analooganduri kasutamise tegema lihtsaks ja võimalikuks. Uurisin välja kuidas kasutada NodeMCU funktsiooni *read.adc() (Random Nerd Tutorials, 2016)* ning asusin katsetama. Peale ühenduste loomist asusin testima erinevaid võimalusi, kuidas andurilt saadud tulemusi stabiilseks saada. Tulemused olid iga lugemise käigus väga erinevad. Veendumaks, et viga ei

ole anduris, testisin lugemist ka Arduino (Arduino, 2016) mikrokontrolleriga ning lugemise tulemused olid stabiilsed ja erinevad võrreldes ESP8266 seadmega. Lugedes valgusdioodi katsetamised ebaedukaks, otsustasin proovida teist analoogandurit, mis on mõeldud veetaseme mõõtmiseks. Esmalt ühendasin anduri valgusdioodiga sarnaselt ning laadisin testkoodi kiibile. Ühenduse saamine anduriga ebaõnnestus täielikult. Proovisin erinevaid ühenduse kombinatsioone ning koodinäiteid, kuid ESP-12E ei suutnud tagastada ühtegi tulemust veetaseme andurilt. Kindluse mõttes katsetasin erinevaid NodeMCU tarkvara versioone (NodeMCU, 2014), lootes, et kõige uuemas versioonis on midagi katki. Kasutades vanemat tarkvara ei õnnestunud siiski saada kumbagi andurit edukalt tööle.

Jättes ajutiselt katsetamise pooleli, otsustasin uurida lähemalt, milles seisneb probleem. Leidsin, et paljudel robootikahuvilistel on esinenud sama probleem ning põhjuseks konverteri töötamise pinge erinevus. Enamus andureid on mõeldud töötama pingega 5V, kuid ESP8266 analoogsignaali konverter suudab muuta anduri signaali vahemikus 1.8-3.6V, mis tähendab, et vajab töötamiseks spetsiaalseid andureid. Kuna tuulekiiruse ning suuna andurid väljastavad analoogsignaali ning töötavad enamasti pingega 5-12V, puudub võimalus mõõta tuuleparameetreid kasutades ESP8266 analoog-digitaal konverterit.

3.8. DHT22 ja BMP180 anduri paralleelne kasutamine

Jõudes faasi, kus olen edukalt mõlemad andurid tööle saanud, oli järgmiseks etapiks luua kood ja vooluring, et oleks võimalik temperatuuri, õhuniiskuse taset ning õhurõhku korraga väljastada. Kuna seni kasutataval ESP-01 mudelil on vaid 2 sisend-/väljundpesa, oli vajadus hankida rohkemate pesadega ESP8266 mudel. Leidsin, et ESP-12E on minule sobiv mudel ning tellisin selle Suur-Britanniast 11€ eest. Peale uue seadme kohale jõudmist laadisin NodeMCU Lua tarkvara kiibile ning alustasin koodi kokkupanemisega, mille peale kulus suhteliselt vähe aega.

Esmane katsetus ei läinud edukalt. ESP8266 jäi tsükliliselt taaskäivituma ning anduritelt ei saanud ma ühtegi vastust. Arvasin, et viga on koodis ning proovisin läbi mitmeid erinevaid muudatusi ja koodivariante, kuid edutult. Peale mitmeid päevi katsetusi leidsin internetist, et viga võib olla toiteallikas. Nimelt USB UART konverterit toiteallikana kasutades võib tekkida olukord, kus andurid kasutavad rohkem kui 250mA ning on vajalik kasutada võimsamat välist toiteallikat. Leidsin, et turul ei pakuta väga suures valikus +3.3V pingega toiteallikaid, mis

oleks suutelised väljastama vähemalt 1A. Kuna soovin kasutada kahte andurit, oli vähemalt 1A võimekusega toiteallikas vajalik. Õnneks leidsin ühe sobiva mudeli internetist ning tellisin selle ära.

Uue toiteallika saabudes disainisin uue vooluringi ning proovisin uuesti mõlemalt andurilt andmeid kätte saada. Üks suur viga, mille alguses tegin, oli kõigi maanduste mitte kokkuühendamine. Selle tulemusena ei olnud vooluring tervik ning anduritelt ei saanud andmeid kätte. Peale korrektse vooluringi loomist sain Lualoader tarkvara konsoolivaates temperatuuri, õhuniiskuse ning õhurõhu kätte.

3.9. Andmebaasiga ühendamine

Anduritelt saadud andmete hoidmiseks on loogiline kasutada andmebaasi. Kuna mul on enda server olemas, otsustasin luua sinna uue andmebaasi ning hoida igat parameetrit eraldi tabelis. Esimese sammuna soovisin saata temperatuuri andmed iga lugemise käigus andmebaasi. Lugesin nii Lua keele dokumentatsiooni kui ka artikleid erinevatest foorumitest ning koostasin testkoodi. Esmalt oli vaja ühendada ESP8266 minu Wi-Fi võrguga. Peale ESP8266 käivitamist otsib seade faili nimega init.lua ning selle olemasolul käivitab selle esimesena, mistõttu on mõistlik Wi-Fi ühendusega seotud kood just sinna paigutada. Koostasin testfaili (Koodinäide 3.4) ning laadisin kiibile.

```
wifi.setmode(wifi.STATION)
wifi.sta.config("Sputnik","satelliit")
wifi.sta.connect()
tmr.alarm(1, 1000, 1, function()
if wifi.sta.getip()== nil then
print("IP unavaiable, Waiting...")
else
tmr.stop(1)
print("Config done, IP is "..wifi.sta.getip())
end
end)
```

Koodinäide 3.4: ESP8266 Wi-Fi võrku ühendamine

Wi-Fi võrku ühendamine läks edukalt ning ei valmistanud probleeme. Järgmiseks sammuks oli

laadida kiibile andurite teegid ning fail, mis loeks anduritelt info ja kirjutaks andmebaasi. Koostasin funktsiooni (Koodinäide 3.5), mis saadab staatilisele IP aadressile temperatuuri, kasutades GET meetodit.

```
IP="46.101.62.130"
conn=net.createConnection(net.TCP, 0)
conn:on("receive", function(conn, payload) print(payload)
end)
conn:connect(80, IP)
conn:send("GET /insert.php?temperature="..t1.."
HTTP/1.1\r\n")
conn:send("Host: IP\r\n")
conn:send("Accept: */*\r\n")
conn:send("User-Agent: Mozilla/4.0 (compatible; esp8266 Lua;
Windows NT 5.1)\r\n")
conn:send("\r\n")
conn:on("sent",function(conn)
conn:close()
end)
conn:on("disconnection", function(conn)
```

Koodinäide 3.5: Temperatuuri edastamine

Selleks, et temperatuur andmebaasi kirjutada, koostasin PHP faili, mis loob ühenduse minu andmebaasiga ning kirjutab aadressirealt saadud temperatuuri andmebaasi tabelisse. Lõplik versioon failist on leitav (Lisa 8.).

Palju aega kulus selleks, et esimene andurist loetud tulemus andmebaasi kirjutada. Kuna Lua koodi näiteid oli vähe ning dokumentatsioon puudulik, kulus ülesande lahendamisele mitu nädalat.

Peale temperatuuri edukat andmebaasi kirjutamist oli järgmiseks loogiliseks sammuks kirjutada kõik parameetrid andmebaasi. Selleks laadisin mõlema anduri teegid kiibile ning muutsin koodi nii, et kõik kolm parameetrit saadetaks staatilisele IP aadressile. Samuti oli vaja muuta PHP faili, et ühe parameetri asemel kirjutataks andmebaasi kolm.

Ühendasin pinge vooluringi ning ootasin kannatlikult, et näha andmebaasis temperatuuri, õhuniiskuse ning õhurõhu andmeid, mis ilmusid mõne sekundi pärast. Katsetusest tegin

järelduse, et anduritelt andmete lugemiseks ja andmebaasi saatmiseks kulub keskmiselt umbes 2 sekundit.

Peale edukat andmete kirjutamist andmebaasi kordasin katset veel mitu korda, et testida, kui stabiilne on ESP8266 seade. Esmaste katsetuste järel tegin järelduse, et seade on üldjuhul stabiilne.

3.10. Andmete lugemine fikseeritud ajaperioodi tagant

Eesmärgiks oli luua seade, mis loeb anduritelt andmed ning salvestab andmebaasi kindla ajaperioodi tagant, luues andmekogumiku. Alustasin tsükli loomisega, mis käivitaks andurid iga 1 tunni tagant ning saadaks andmed andmebaasi. Esialgsel katsetusel koostasin taimeri sama faili sisse, kust käivitatakse andurid. Selle tulemusena tekkisid arusaamatutel põhjustel probleemid ESP8266 kiibiga ning katse ebaõnnestus. Otsustasin luua taimeri eraldi faili ning proovida uuesti, mis andis eduka tulemuse. Taimeri (Lisa 5.) puhul on tegemist failiga, kus fikseeritud aja tagant luuakse alarm, mis käivitab faili, mis loeb anduritelt andmed, saadab IP aadressile ning käivitab uuesti taimeri.

Esialgsete katsetuste käigus ei pannud ma tähele, et iga kord nihkub anduritelt andmete lugemine teatud ajaühiku võrra edasi. Selle põhjuseks on asjaolu, et tsükkel ei arvesta andurite töötamisaega ning aega, mis kulub andmete IP aadressile saatmiseks. Kasutades tsüklit lühikese ajaperioodi jooksul ning suurte ajavahedega, ei nihku tsükli käivitamine märkimisväärselt. Minu vajaduse puhul salvestada andmeid 1 tunni tagant liigub andmete täpsus liiga palju ning tsükli kasutamine ei ole mõistlik. Proovisin luua tsüklit (Lisa 4.), mis arvestaks andurite lugemiseks ja andmete saatmiseks kuluvat aega, kuid ESP8266 mälumaht ei ole piisavalt suur, et sellist tsüklit saaks kasutada.

Üritasin leida lahendust ning tulin mõttele kasutada välist kella moodulit. Plaaniks oli küsida moodulilt fikseeritud ajaperioodi tagant kellaaega ning täistunni korral, käivitada andurid ja saata andmed andmebaasi. Leidsin sobivaks seadmeks kella mooduli nimega DS3231.

Esimeseks eesmärgiks oli kellaaja väljastamine, kasutades ESP8266 kiipi. Ühendasin kella SDA ning SCL pesad *gpio* pesadega ning lülitasin seadme vooluringi. Leidsin internetist sobiva testkoodi ning proovisin saada kellaga ühendust. Peale esmaseid katsetusi sain teada, et kellamoodul vajab kasutamiseks nodemcu_integer_0.9.6 (NodeMCU, 2014) tarkvara. Ma olin siiani kasutanud sama tarkvara, aga nodemcu_float_0.9.6 versiooni kuna selleks, et saada

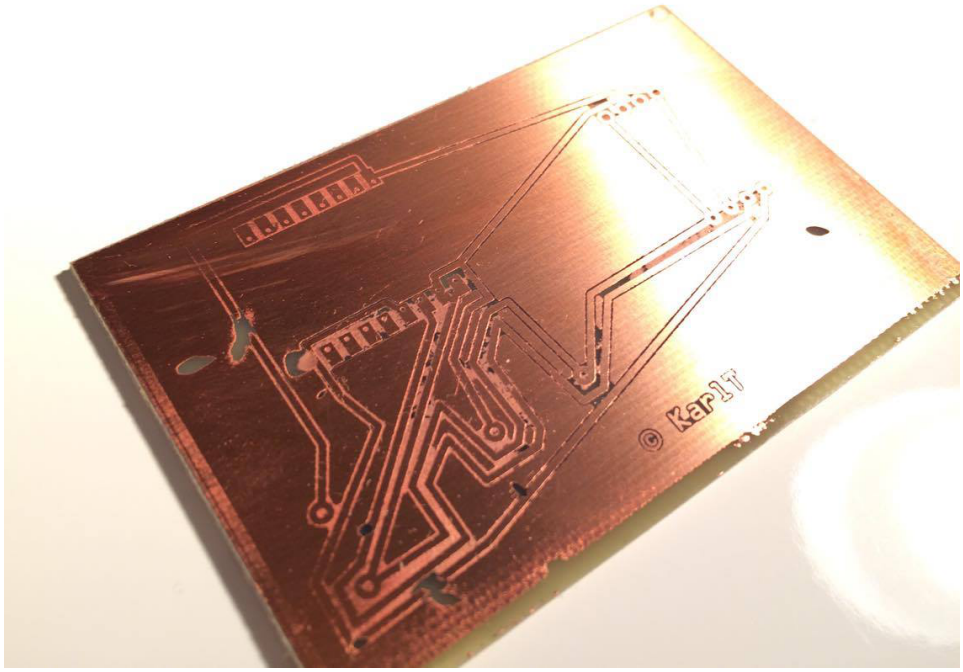
temperatuur kümnendiku täpsusega, on vaja kasutada muutujaid, millele saab omistada murdarve. Tänu sellele lihtsale avastusele ei õnnestunud kellamoodulit koos ESP8266 seadmega kasutada.

3.11. Trükkplaadi disainimine ja loomine

Peale tarkvaralisi katsetusi otsustasin, et oleks huvitav luua enda trükkplaat, kuhu saaks kõik elektroonikakomponendid kinni joota. Tegin uurimustööd selgitamaks välja, millist meetodit oleks mõistlik kodustes tingimustes kasutada ning milliseid vahendeid ma selleks vajan. Otsustasin kasutada fotolakkmeetodit, mis kujutab endast trükkplaati, mis on eelnevalt fotolakiga kaetud ning on valgusele tundlik (Leini, 2016).

Esimeseks sammuks oli disainida enda vajadustele vastav elektriskeem (Lisa 2.), mille saaks hiljem trükkplaadile kanda. Kuna mul on eelnevad kogemused tarkvaraga Fritzing (Fritzing, 2016), otsustasin taas seda kasutada. Ühendasin enda kasutatavad komponendid tarkvaras ning disainisin trükkplaadi skeemi endale meelepäraselt. Sain uusi teadmisi trükkplaadi disainimise kohta, näiteks kui laiad peaks olema trükkplaadi rajad, et ühendused läbi ei põleks. Üsna palju aega kulus selleks, et luua disain, mis vastaks enda ootustele ning oleks efektiivne komponentide paigutuse suhtes.

Peale eduka trükkplaadi disaini loomist (Lisa 3.) hankisin vajalikud vahendid, milleks olid fotolakiga kaetud trükkplaat, naatriumpersulfaat, seebikivi, fotoraam ning läbipaistev printerkile. Esmalt printisin enda trükkplaadi elektriskeemi peegelpildi läbipaistvale kilele ning lõikasin selle sobivasse mõõtu. Kleepisin kile fotoraami klaasi külge ning tegin tööruumi pimedaks, kuna fotolakiga kaetud trükkplaat on valgustundlik. Pimedas asetasin trükkplaadi kile alla ning sulgesin fotoraami ja kiiritasin trükkplaati valguse käes umbes 6 minutit. Kiiritamise ajal tegin seebikivi lahuse klaasnõusse, kuhu asetasin peale kiiritamist trükkplaadi. Seebikivi lahuses toimub edasine ilmutamine. Järgmiseks sammuks oli naatriumpersulfaadi lahuse tegemine, mida on vaja trükkplaadi söövitamiseks. Uputasin trükkplaadi lahusesse ning lasin söövitada umbes 4-5 minutit, peale mida pesin trükkplaati vee all. Tulemuseks sain hea välimusega trükkplaadi, kus kahjuks mõned rajad olid katki, tänu millele ei olnud esimene eksemplar (Illustratsioon 3.1) kasutatav.



Illustratsioon 3.1: Trükkplaadi esimene katsetus

Kordasin 2 korda trükkplaadi loomise protsessi, kuid ka nendel kordadel läksid mõned rajad katki ning plaate ei olnud võimalik kasutada. Järeldasin katsetustest, et täpse trükkplaadi loomine koduste vahenditega ei ole kõige lihtsam protsess ning hea tulemuse saavutamiseks on mõistlikum teha eritellimus enda trükkplaadi disainile.

Kokkuvõte

Planeeritud süsteemi loomine kujunes üldjoontes ootuspäraseks. Enamus elektroonikakomponente sobisid omavahel hästi nii riistavaraliselt kui ka tarkvaraliselt. Alguses arvasin, et koodiga tekib rohkem segadust, kuid positiivse üllatusena sain kõik kirjutatud nii nagu oli vaja.

Nagu alapeatükis 3.10 Andmete lugemine fikseeritud ajaperioodi tagant kirjutasin, tekkis probleem kindla ajaperioodi tagant andmeid lugedes. Andureid käivitav tsükkel nihkub iga käivitusega 2-3 sekundit edasi, tänu millele lükkub iga päeva möödudes ilma perimeetrite lugemine umbes minuti võrra edasi. Ilmajaama puhul on vajalik salvestada andmeid kindla ajaperioodi tagant, et saaks tulemusi analüüsida ning nende põhjal järeldusi teha erinevate aastakäikude kohta. Kuna tsükli kasutamine on ESP8266 kiibi puhul raskendatud tänu mäluprobleemidele ning välise kellamooduli jaoks ei olnud töö tegemise ajal teeki loodud, jõudsin järeldusele, et seade ESP8266 on liiga väikse jõudlusega, et tegeleda andmete kogumisega ning andmebaasi salvestamisega.

Projekti alguses olid ootused ESP8266 kiibile suured ning peale esimestest probleemidest ülesaamist tundus kiip sobivat kasutuseks. Ühendades seadmele juurde aina rohkem komponente ning muutes koodi pikemaks ja keerukamaks, hakkas kiip andma märku enda võimekuse piiridest. Oleksin tahtnud, et ESP8266 oleks ideaalselt tööle hakanud koos kõigi andurite ning elektroonikakomponentidega, kuid seade on loodud teostama ainult lihtsaid ning vähe ressursse nõudvaid ülesandeid.

Otsustasin mitte jätkata seadme edasist riistvaralist arendust, kuna ESP8266 kiip ei suutnud täita minu loodud ülesandeid. Kuna projekti eesmärgiks oli eelkõige uurida seadme võimekust ja katsetada seda reaalkeskkonnas ning eduka tulemuse puhul luua prototüüp, saan öelda, et uurimus ja testimine oli põhjalik ning edukas, vaatamata asjaolule, et seadme prototüüp ei valminud.

Uurimine, disainimine, katsetamine ja reaalkeskkonnas testimine andis mulle väga hea ülevaate ning kogemuse riistvara ja tarkvara kombineerimisest. Alustades sarnast uut projekti teeksin paljusid tööetappe teisiti ning kindlasti efektiivsemalt. Katsetaksin mitut erinevat seadet ning valiksin välja kõige sobivama, millega tagaksin parema projekti õnnestumise võimaluse.

Kõige huvitavam osa projekti puhul oli erinevate protsesside läbitegemine. Peale põhjalikku uurimustööd oli vaja seadmeid katsetada, disainida nii elektriskeeme kui ka koodi, leida lahendus jooksvalt tekkivatele riistvaraprobleemidele ning lõpuks saavutada korrektsus ja stabiilsus. Kõige raskemaks osaks oli kindlasti stabiilsuse loomine. Lihtne on lugeda andurilt andmeid ja neid edastada, kuid teha seda aastaid järjest täpse ajaperioodi tagant, kasutades laialt levinud elektroonikakomponente, osutus suureks katsumuseks. Tööstuslike seadmete definitsioon sai minu jaoks uue tähenduse, mõistes miks need seadmed on kordades kallimad võrreldes tavaseadmetega.

Projekt andis väga hea ülevaate, kui palju uurimist, ehitamist ning testimist kulub lihtsa seadme loomisele algusest lõpuni. Kõigile, kellel on huvi luua ise mõni elektroonikaseade, soovitan varuda palju kannatust ja aega ning pidada meeles, et testimist ei ole kunagi piisavalt. Kõige esimene mõte ei pruugi olla alati kõige parem ning tihti juhatab aeg kõige sobivama lahenduseni.

Kasutatud kirjandus

2016. () . Arduino. Loetud kuupäeval 05.03.2014 aadressil https://www.arduino.cc/

Autor puudub. (2015) . Õhurõhk. Loetud kuupäeval 16.03.2016 aadressil https://et.wikipedia.org/wiki/%C3%95hur%C3%B5hk

Espressif. (2016) . ESP8266EX Overview | Espressif Systems. Loetud kuupäeval 11.09.2015 aadressil https://espressif.com/en/products/hardware/esp8266ex/overview

Espressif_Kelly. (2015) . GPIO Maximum current Imax. Loetud kuupäeval 20.11.2015 aadressil http://bbs.espressif.com/viewtopic.php?t=139

Fritzing. (2016) . . Loetud kuupäeval 15.06.2015 aadressil http://fritzing.org/home/

jescarri. (2015) . DHT22 module. Loetud kuupäeval 07.03.2016 aadressil https://github.com/javieryanez/nodemcu-modules/tree/master/dht22

Jimb0. (2010) . RS-232 vs. TTL Serial Communication. Loetud kuupäeval 28.10.2015 aadressil https://www.sparkfun.com/tutorials/215

Mikk Leini. (2016) . UV fotoresistiga trükkplaatide valmistamine. Loetud kuupäeval 23.08.2016 aadressil https://www.robotiklubi.ee/juhendid/uv_trykkplaadi_valmistamine

NodeMCU. (2014) . nodemcu-firmware. Loetud kuupäeval 24.04.2016 aadressil https://github.com/nodemcu/nodemcu-firmware/releases

NodeMCU. (2015) . NodeMCU Flasher. Loetud kuupäeval 21.10.2015 aadressil https://github.com/nodemcu/nodemcu-flasher

NodeMCU Team. (2015) . BMP085 I2C module for NODEMCU. Loetud kuupäeval 05.02.2016 aadressil https://github.com/nodemcu/nodemcu-firmware/blob/2fbd5ed509964a16057b22e00aa8469d6a522d73/lua_modules/bmp085/bmp085.lua

Peter Jennings. (2014) . ESP8266 Lua Loader. Loetud kuupäeval 15.10.2015 aadressil http://benlo.com/esp8266/

Random Nerd Tutorials. (2016) . ESP8266 ADC – Reading Analog Values with NodeMCU. Loetud kuupäeval 13.06.2016 aadressil http://randomnerdtutorials.com/esp8266-adc-reading-analog-values-with-nodemcu/

Robo India. (Kuupäev puudub) . LED Blinking using ESP8266 by LUA. Loetud kuupäeval 01.11.2015 aadressil http://roboindia.com/tutorials/esp8266-led-blinking-lua

Simon. (2015) . Making a NodeMCU Lua ESP8266 weather station. Loetud kuupäeval 14.01.2016 aadressil http://www.beerandchips.net/2015/12/26/making-nodemcu-lua-esp8266-weather-station/

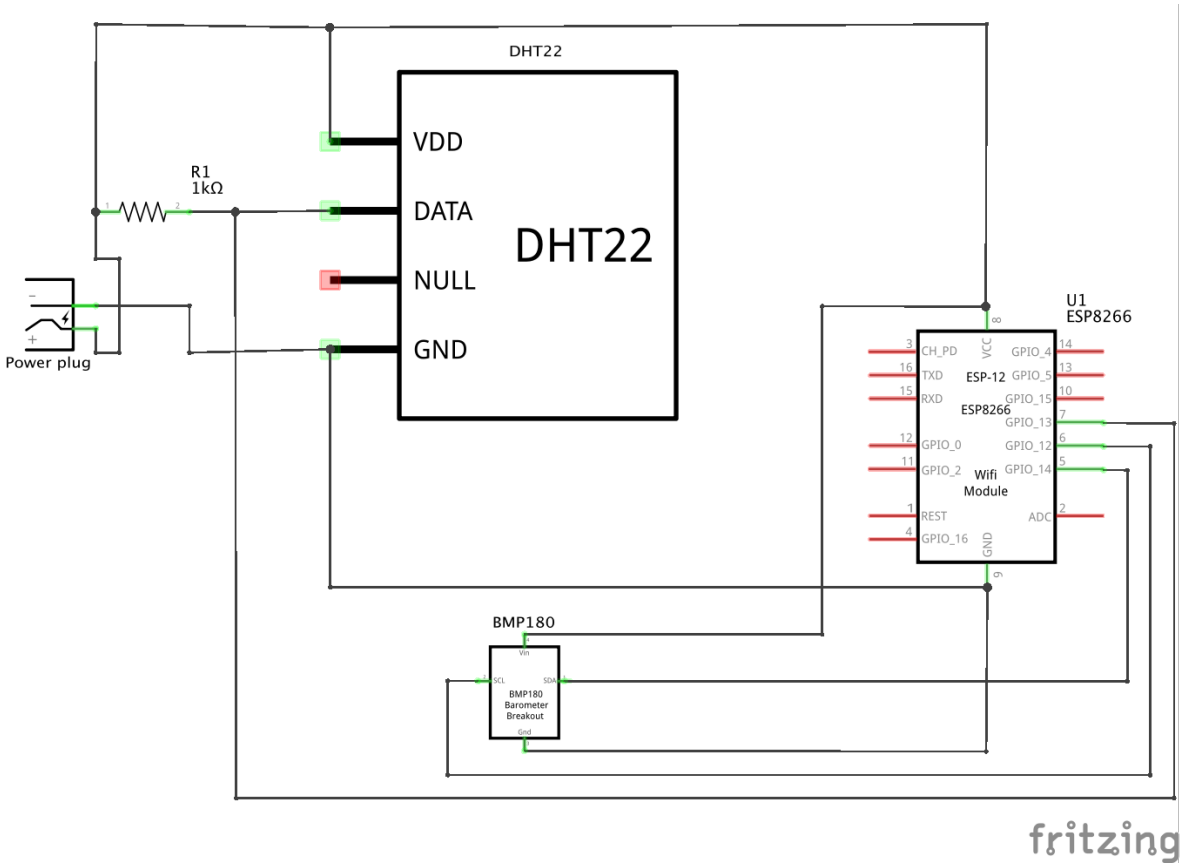
stulander. (2016) . esp8266-module-family [ESP8266 Support WIKI]. Loetud kuupäeval 11.09.2015 aadressil http://www.esp8266.com/wiki/doku.php?id=esp8266-module-family

LISAD

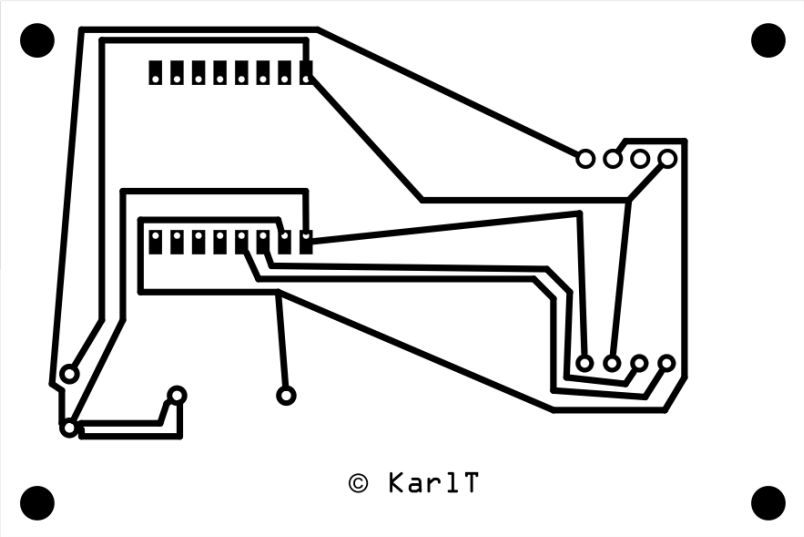
Lisa 1. Google Trends otsing „Internet of things”

Otsingutulemus kuupäeval 29.10.2016

Lisa 2. Elektroonikaskeem



Lisa 3. Trükkplaadi skeem



Lisa 4. Read-send-data.lua

```
DHT= require("dht22_min")
DHT.read(7)
temperature = DHT.getTemperature()
humidity = DHT.getHumidity()
t1 = (temperature / 10)..".."..(temperature % 10)
h1 = humidity / 10
if humidity == nil then
    print("Error reading from DHT22")
else
      print("Temperature: "..t1.."C")
      print("Humidity: "..h1.."%")
end
DHT = nil
package.loaded["dht22_min"]=nil
OSS = 1
SDA_PIN = 5
SCL_PIN = 6
bmp180 = require("bmp180")
bmp180.init(SDA_PIN, SCL_PIN)
bmp180.read(OSS)
p1 = bmp180.getPressure()
p2 = (p1 * 75 / 10000)
print("Pressure: "..p2.."mmHg")
bmp180 = nil
package.loaded["bmp180"]=nil
tmr.alarm(2, 1000, 1, function() sendData() end)
function sendData()
IP="46.101.62.130"
conn=net.createConnection(net.TCP, 0)
conn:on("receive", function(conn, payload) print(payload) end)
conn:connect(80, IP)
conn:send("GET /insert.php?
temperature="..t1.."&humidity="..h1.."&pressure="..p2.." HTTP/1.1\r\n")
conn:send("Host: IP\r\n")
conn:send("Accept: */*\r\n")
conn:send("User-Agent: Mozilla/4.0 (compatible; esp8266 Lua; Windows NT
```

```
5.1)\r\n")
conn:send("\r\n")
conn:on("sent",function(conn)
conn:close()
end)
conn:on("disconnection", function(conn)
end)
dofile("timer.lua")
end
```

Lisa 5. Timer.lua

```
tmr.alarm(2, 3600000, 1, function() dofile("read-send-data.lua") end)
-- 1m 60 000
-- 10m 600 000
-- 15m 900 000
-- 30m 1 800 000
-- 60m 3 600 000
```

Lisa 6. DHT22 teek

```
-- ***************************************************************
-- DHT22 module for ESP8266 with nodeMCU
--
-- Written by Javier Yanez
-- but based on a script of Pigs Fly from ESP8266.com forum
--
-- MIT license, http://opensource.org/licenses/MIT
-- ***************************************************************
local e=...
local a={}_G[e]=a
local o
local d
function a.read(e)local r
local t
o=0
d=0
r=0
local i=gpio.read
local l={}for e=1,40,1 do
l[e]=0
end
local a=0
gpio.mode(e,gpio.OUTPUT)gpio.write(e,gpio.HIGH)tmr.delay(100)gpio.write(e,
gpio.LOW)tmr.delay(40)gpio.mode(e,gpio.INPUT)while(i(e)==0)do end
local n=0
while(i(e)==1 and n<500)do n=n+1 end
while(i(e)==0)do end
n=0
while(i(e)==1 and n<500)do n=n+1 end
for d=1,40,1 do
while(i(e)==1 and a<10)do
a=a+1
end
l[d]=a
a=0
while(i(e)==0)do end
```

```
end
for e=1,16,1 do
if(l[e]>4)then
o=o+2^(16-e)end
end
for e=1,16,1 do
if(l[e+16]>4)then
d=d+2^(16-e)end
end
for e=1,8,1 do
if(l[e+32]>4)then
r=r+2^(8-e)end
end
t=(bit.band(o,255)+bit.rshift(o,8)+bit.band(d,255)+bit.rshift(d,8))t=bit.b
and(t,255)if d>32768 then
d=-(d-32768)end
if(t-r>=1)or(r-t>=1)then
o=nil
end
gpio.mode(e,gpio.OUTPUT)gpio.write(e,gpio.HIGH)
end
function a.getTemperature()return d
end
function a.getHumidity()return o
end
return a
```

Lisa 7. BMP180 teek

```
-- ***************************************************************
-- BMP180 module for ESP8266 with nodeMCU
-- BMP085 compatible but not tested
--
-- Written by Javier Yanez
--
-- MIT license, http://opensource.org/licenses/MIT
-- ***************************************************************
local moduleName = ...
local M = {}
_G[moduleName] = M

local ADDR = 0x77 --BMP180 address
local REG_CALIBRATION = 0xAA
local REG_CONTROL = 0xF4
local REG_RESULT = 0xF6

local COMMAND_TEMPERATURE = 0x2E
local COMMAND_PRESSURE = {0x34, 0x74, 0xB4, 0xF4}

-- calibration coefficients
local AC1, AC2, AC3, AC4, AC5, AC6, B1, B2, MB, MC, MD

-- temperature and pressure
local t,p

local init = false

-- i2c interface ID
local id = 0

-- 16-bit  two's complement
-- value: 16-bit integer
local function twoCompl(value)
 if value > 32767 then value = -(65535 - value + 1)
 end
```

```
 return value
end

-- read data register
-- reg_addr: address of the register in BMP180
-- lenght: bytes to read
local function read_reg(reg_addr, length)
  i2c.start(id)
  i2c.address(id, ADDR, i2c.TRANSMITTER)
  i2c.write(id, reg_addr)
  i2c.stop(id)
  i2c.start(id)
  i2c.address(id, ADDR,i2c.RECEIVER)
  c = i2c.read(id, length)
  i2c.stop(id)
  return c
end

-- write data register
-- reg_addr: address of the register in BMP180
-- reg_val: value to write to the register
local function write_reg(reg_addr, reg_val)
  i2c.start(id)
  i2c.address(id, ADDR, i2c.TRANSMITTER)
  i2c.write(id, reg_addr)
  i2c.write(id, reg_val)
  i2c.stop(id)
end

-- initialize module
-- sda: SDA pin
-- scl SCL pin
function M.init(sda, scl)
  i2c.setup(id, sda, scl, i2c.SLOW)
  local calibration = read_reg(REG_CALIBRATION, 22)

  AC1 = twoCompl(string.byte(calibration, 1) * 256 +
string.byte(calibration, 2))
  AC2 = twoCompl(string.byte(calibration, 3) * 256 +
```

```
string.byte(calibration, 4))
  AC3 = twoCompl(string.byte(calibration, 5) * 256 +
string.byte(calibration, 6))
  AC4 = string.byte(calibration, 7) * 256 + string.byte(calibration, 8)
  AC5 = string.byte(calibration, 9) * 256 + string.byte(calibration, 10)
  AC6 = string.byte(calibration, 11) * 256 + string.byte(calibration, 12)
  B1 = twoCompl(string.byte(calibration, 13) * 256 +
string.byte(calibration, 14))
  B2 = twoCompl(string.byte(calibration, 15) * 256 +
string.byte(calibration, 16))
  MB = twoCompl(string.byte(calibration, 17) * 256 +
string.byte(calibration, 18))
  MC = twoCompl(string.byte(calibration, 19) * 256 +
string.byte(calibration, 20))
  MD = twoCompl(string.byte(calibration, 21) * 256 +
string.byte(calibration, 22))

  init = true
end

-- read temperature from BMP180
local function read_temp()
  write_reg(REG_CONTROL, COMMAND_TEMPERATURE)
  tmr.delay(5000)
  local dataT = read_reg(REG_RESULT, 2)
  UT = string.byte(dataT, 1) * 256 + string.byte(dataT, 2)
  local X1 = (UT - AC6) * AC5 / 32768
  local X2 = MC * 2048 / (X1 + MD)
  B5 = X1 + X2
  t = (B5 + 8) / 16
  return(t)
end

-- read pressure from BMP180
-- must be read after read temperature
local function read_pressure(oss)
  write_reg(REG_CONTROL, COMMAND_PRESSURE[oss + 1]);
  tmr.delay(30000);
  local dataP = read_reg(0xF6, 3)
```

```
  local UP = string.byte(dataP, 1) * 65536 + string.byte(dataP, 2) * 256 +
string.byte(dataP, 1)
  UP = UP / 2 ^ (8 - oss)
  local B6 = B5 - 4000
  local X1 = B2 * (B6 * B6 / 4096) / 2048
  local X2 = AC2 * B6 / 2048
  local X3 = X1 + X2
  local B3 = ((AC1 * 4 + X3) * 2 ^ oss + 2) / 4
  X1 = AC3 * B6 / 8192
  X2 = (B1 * (B6 * B6 / 4096)) / 65536
  X3 = (X1 + X2 + 2) / 4
  local B4 = AC4 * (X3 + 32768) / 32768
  local B7 = (UP - B3) * (50000/2 ^ oss)
  p = (B7 / B4) * 2
  X1 = (p / 256) * (p / 256)
  X1 = (X1 * 3038) / 65536
  X2 = (-7357 * p) / 65536
  p = p +(X1 + X2 + 3791) / 16
  return (p)
end

-- read temperature and pressure from BMP180
-- oss: oversampling setting. 0-3
function M.read(oss)
  if (oss == nil) then
     oss = 0
  end
  if (not init) then
     print("init() must be called before read.")
  else
     read_temp()
     read_pressure(oss)
  end
end;

-- get temperature
function M.getTemperature()
  return t
end
```

```
-- get pressure
function M.getPressure()
  return p
end

return M
```

Lisa 8. Insert.php

```
 <?php
        $mysqli = new mysqli("localhost", "root", "Password", "esp");
        $temperature = $_GET["temperature"];
        $stmt= $mysqli->prepare("INSERT INTO temperature (temperature) VALUES
(?)");
        $stmt->bind_param("d", $temperature);
        $stmt->execute();

        $humidity = $_GET["humidity"];
        $stmt= $mysqli->prepare("INSERT INTO humidity (humidity) VALUES (?)");
                $stmt->bind_param("i", $humidity);
                $stmt->execute();

        $pressure = $_GET["pressure"];
        $stmt= $mysqli->prepare("INSERT INTO pressure (pressure) VALUES (?)");
                $stmt->bind_param("i", $pressure);
                $stmt->execute();

?>
```