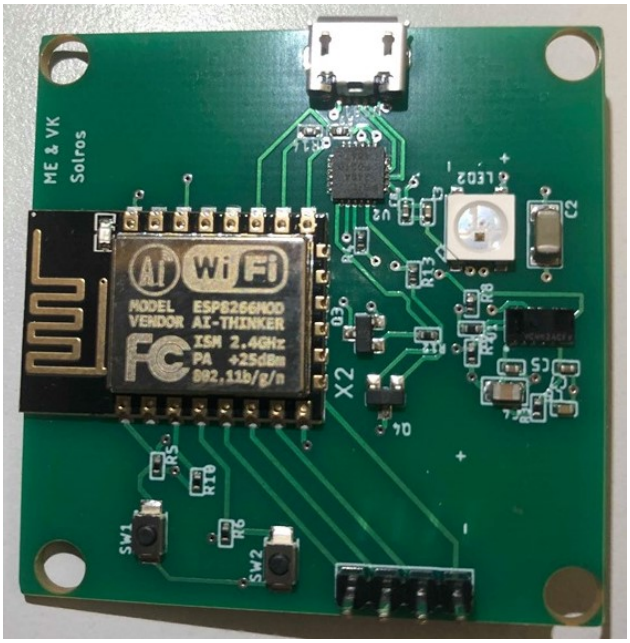
CHALMERS
UNIVERSITY OF TECHNOLOGY



Kretskortsdesign & programmering av mikrokontroller

Kandidatarbete i Elektroteknik

Michael Emmerstorfer
Victor Klint

Institutionen för Elektroteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2019

Kretskortsdesign & programmering av mikrokontroller

Michael Emmerstorfer
Victor Klint

CHALMERS
UNIVERSITY OF TECHNOLOGY

Institutionen för Elektroteknik
Chalmers Tekniska Högskola
Göteborg, Sverige 2019

Kretskortsdesign & programmering av mikrokontroller

Handledare: Thomas Rylander, Institutionen för Elektroteknik
Handledare: Jon Ramstedt, Solros AB (ESSIQ AB)
Examinator: Thomas Rylander, Institutionen för Elektroteknik

Institutionen för Elektroteknik
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Typset i LaTeX
Tryckt av Chalmers Reproservice
Göteborg, Sverige 2019

Sammanfattning

Solros är ett system som leder solljus från en mottagarenhet placerad utomhus till rum inomhus med hjälp av optiska fibrer. Systemet består av en parabol som fokuserar solljuset på en mottagarapertur kopplad till optiska fibrer vilka sedan leder ljuset vidare in i byggnaden. Med hjälp av sensorer och ett styrsystem så följer parabolen solen med stor precision. För att låta användaren styra och få information trådlöst från Solrossystemet så ska en prototyp konstrueras där kretskortet ska designas och tillverkas men även mjukvaran till mikrokontrollern på kretskortet ska programmeras. Kretskortet ska kunna få plats bakom en lampknapp på väggen.
Hårdvaran designades i EAGLE och tillverkades av Seeed studios i Kina. Mjukvaran programmerades i MicroPython som är anpassat för mikrokontrollers och även är användarvänligt. Projektet kan utvecklas vidare genom att implementera en skärm för att visa information från Solrossystemet. Mjukvaran kan även programmeras om så att kretskortet kan användas för att styra andra liknande system.

Abstract

Solros is a system which guides sunlight from the outdoor environment to indoor rooms through fiber optic cables. A parabolic dish focuses the sunlight onto a receiving aperture connected to optical fibers that guides the sunlight into the building. With the help of sensors and a control system, the parabolic dish follows the sun with great precision. In order for the user to be able to control and receive information from the Solros system, a prototype is constructed, which requires a new design for a PCB as well as the software for the microcontroller on the PCB. The PCB should be able to fit in a lamp button on the wall. The hardware is designed in Eagle and manufactured by Seeed Studios in China. The software is programmed in MicroPython since it is suitable for microcontrollers as well as being user friendly. The prototype that was made to met most of the requirements that are set. The project can be developed further through the implementation of a screen to show the user information regarding the Solros system. The software could also be reprogrammed so that the PCB can be used to control different systems of the same kind.

Förord

Denna rapport är ett examensarbete för utbildningen högskoleingenjör inom elektroteknik vid Chalmers Tekniska Högskola. Utbildningen omfattar 180 högskolepoäng och rapporten är utbildningens avslutande moment på 15 högskolepoäng. Projektet har till större del utförts i Essiqs lokaler därav vill vi tacka både Marcus Ekerhult och Fredrik Persson för möjligheten att göra vårt examensarbete hos Essiq. Vi vill även rikta ett stort tack till Jon Ramstedt på Essiq som har agerat handledare och varit till stor hjälp under projektets gång. Ett sista tack går till Thomas Rylander som är examinator/handledare för detta examensarbete.

Michael Emmerstorfer & Victor Klint, Göteborg, Maj 2019

Innehåll

Figurer

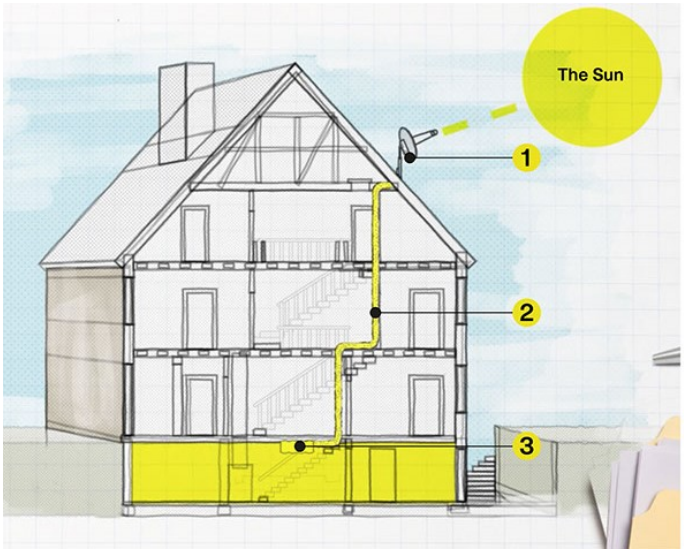
1

Introduktion

I detta avsnitt behandlas rapportens bakgrund, syfte, avgränsningar, precisering av mål samt säkerhet.

1.1 Bakgrund

Solros är ett projekt startat av Jon Ramstedt. Projektet går ut på att utveckla ett system som leder solljus från en mottagarenhet placerad utomhus till rum inomhus med hjälp av optiska fibrer och på så sätt går det att förbättra ljuset i rum där det inte finns något naturligt ljus. Systemet består av en parabol som fokuserar solljuset på en mottagarapertur kopplad till optiska fibrer vilka sedan leder ljuset vidare in i byggnaden. Med hjälp av sensorer och ett styrsystem så följer parabolen solen med stor precision. Finns det direkt solljus så kommer systemet att alltid leda solljus genom de optiska fibrerna som är kopplade till systemet. Solrossystemet är inte helt slutfört men förväntas att bli färdigställt inom kort. I figur 1.1 så visas en bild på hur hela Solrossystemet är uppbyggt.



Figur 1.1: Exempel på hur Solros system kan användas: (1) mottagarenhet med parabol som fokuserar solljus på en mottagarapertur kopplad till en optisk fiber; (2) optisk fiber som leder ljuset in i byggnaden; och (3) ljuskälla i ett rum som saknar fönster och därmed behöver belysning.

1.2 Syfte

Som konsument så är det inte alltid lämpligt att ljuset är på hela tiden utan att det ska kunna stängas av när det önskas. Därmed behövs en kontroll- och informationsenhet som kan starta och stänga av systemet trådlöst. Enheten ska designas så att den kan monteras på väggen och inte vara större än en vanlig lampknapp. Enheten ska även användas som en informationsenhet där parabolen ska kunna skicka ut information kring systemet.

1.3 Avgränsningar

Projektet kommer att leverera en prototyp av en kontroll- och informationsenhet som ska kunna uppfylla kraven som är ställda. Under projektet kommer det inte att tas hänsyn till ekonomiska faktorer när det kommer till designen av prototypen. ESP8266 kommer att användas som mikrochipp och kommer vara integrerad i en WiFi modul som heter ESP-12E på kretskortet, inga övriga alternativ kommer att övervägas.

1.4 Precisering av mål

Målet med arbetet är att utveckla ett koncept för en kontroll- och informationsenhet. Arbetet inkluderar:

- Design och konstruktion av ett kretskort med en mikrokontroller.
- Utveckla och implementera mjukvara till mikrokontrollern.

Följande är kraven som kontroll- och informationsenheten ska uppfylla:

- Den skall kunna anslutas trådlöst till ett Solrossystem. Anslutningen ska kunna ske:
  - Via ett lokalt WiFi som Solros också är uppkopplat på.
  - Direkt till själva Solrossystemet.
- Användaren ska fysiskt kunna sätta på och stänga av solljuset genom att aktivera en närhetssensor på kretskortet.
- Det skall finnas ett sätt för användaren att se om det finns något solljus tillgängligt på kontroll- och informationsenheten. Detta ska indikeras med en LED-lampa eller en skärm.

1.5 Säkerhet

Då det är tänkt att enheten ska monteras på en vägg och försörjas med ström från husets el-system så kommer el-säkerhet att bearbetas under rapporten. IT-säkerheten kommer också att bearbetas då kommunikation kommer att ske via WiFi och det är inte önskvärt att andra personer kan få åtkomst till systemet eller kommunikationen.

2
Hårdvarudesign

I detta kapitel så kommer designen av hårdvaran för kontroll- och informationsenheten att bearbetas. Detta kapitel kommer att inkludera både metod samt teori för att få en djupare förståelse av designen och valen som har gjorts.

2.1 Förstudie

Arbetet började med att höja kunskapen inom de områden som projektet berör samt hitta rätt programvaror för de olika delarna i projektet. Detta genomfördes genom att läsa olika studier samt hitta liknande projekt och sedan analysera detta för att få en djupare förståelse för projektet och dess arbetsgång.

Då det redan var bestämt att ESP8266 skulle användas så spenderades det ingen tid på att överväga andra alternativ. Diverse datablad studerades även för att få en bredare förståelse av både ESP8266 men även övriga komponenter som krävs för systemet.

2.2 Mikrochipp

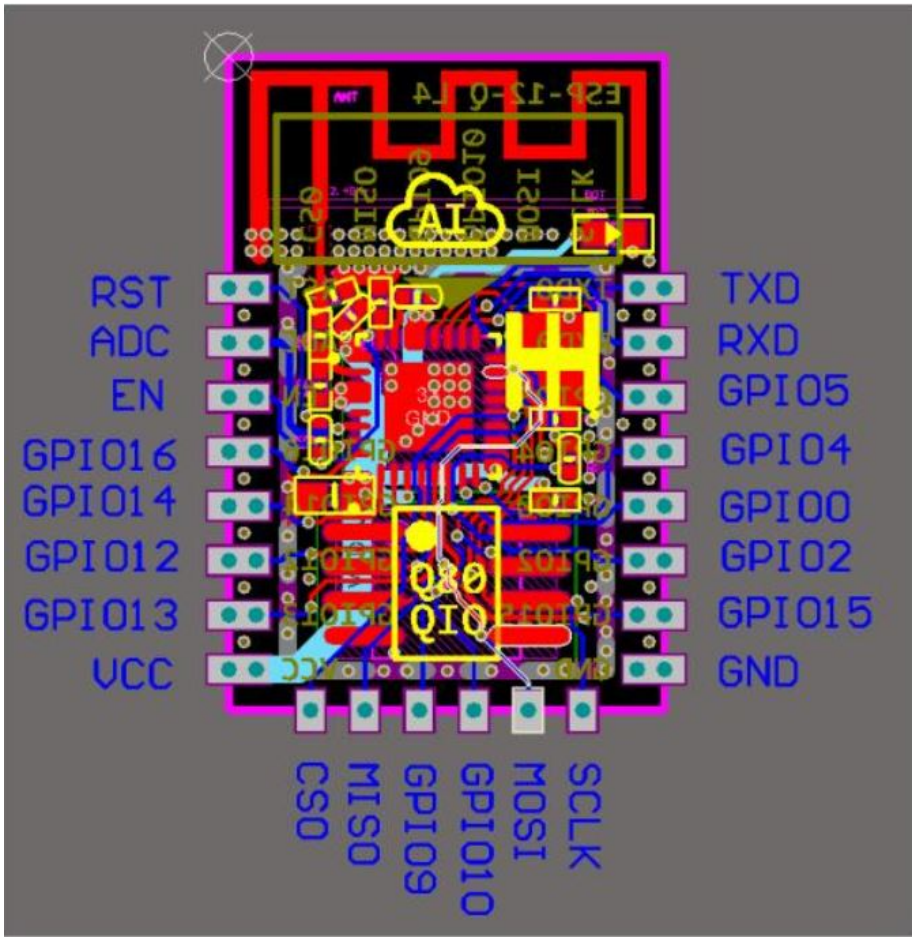
Det mikrochipp som är valt för detta projekt är ESP8266 som innehåller en 32-bit mikrokontroller som är WiFi kapabel. För att kunna integrera WiFi-kommunikation i systemet så krävs det ett mikrochipp som är WiFi kompatibelt och kan hantera antennomkompplare, där båda dessa krav är uppfyllda av ESP8266. ESP8266 används till största del som en integrerad komponent i diverse moduler och sällan som enskild komponent i ett system. För att kunna utnyttja dess WiFi funktion så krävs det även en antenn för att kunna skapa en uppkoppling emot WiFi[1].

ESP-12E är en WiFi modul som innehåller ESP8266, en antenn och många fler komponenter som stödjer övriga funktioner. Modulen stödjer standarden IEEE802.11 b/g/n protokoll som krävs för att kunna kommunicera via ett WiFi-nätverk. Användare kan koppla upp modulen till befintliga nätverk men modulen kan även etablera ett eget nätverk. ESP-12E behöver en matningsspänning på 3.3VDC. Huvudfunktioner som ESP-12E har listas nedan[2]:

- 802.11 b/g/n protokoll

- Integrerad 32-bit MCU (ESP8266)
- Integrerad 10-bit A/D-converter
- TCP/IP protokoll stack
- Stödjer antenn
- WiFi 2.4 GHz, stödjer WPA/WPA2
- SPI, I2C, PWM, GPIO

Pin-layouten av ESP-12E visas i figur 2.1. Olika pins har redan förinstallerade funktioner kopplade till dem. GPIO12-GPIO15 är till för SPI kommunikation som ofta används vid användning av skärmar. GPIO4 och GPIO5 är till för I2C kommunikation. Det går dock att ändra så att olika pins hanterar olika funktioner om det önskas.



Figur 2.1: ESP-12E stift utläggning, från [2].

2.2.1 Insignaler

Följande avsnitt kommer att redovisa samtliga insignaler till ESP-12E. Avsnittet kommer även att redovisa vilka matningsspänningar som krävs för att kretskortet ska fungera och hur de uppnås. Insignalerna är huvudsakligen till för att användaren ska kunna styra Solrossystemet men de är även till för att kunna programmera mikrokontroller.

2.2.1.1 Matningsspänning

Olika komponenter i system kräver matningsspänning och då enheten ska monteras på väggen och försörjas med ström från husets el-system så har säkerhet varit en faktor vid designen av en strömförsörjningskrets. Matningsspänningar som behövs för kretskortet är 5VDC och 3.3VDC för de olika komponenterna. Spänningen som tas från el-systemet är 230VAC och den behöver både transformeras till lämpliga nivåer och AC/DC omvandlas.
Till en början så var tanken att designa ett helt eget kretsschema för att hantera detta problem men efter några veckors arbete så gjordes bedömningen att det skulle kunna bli för osäkert och komplicerat att fortsätta på detta spår. Arbetet gick sedan ut på att hitta en modul som transformerade ner spänningen till önskade nivåer.

HLK-5M05 och HLK-PM03 valdes att användas som moduler för spänningsomvandlingen. Modulen är en isolated switch-mode AC/DC-omvandlare. Spänningen omvandlas först från AC till DC via en bridge-rectifier och sedan växlas spänning väldigt fort och vidare till en flyback transformer som transformerar ner spänningen, mellan allt detta så finns det även filter som består av induktorer och kondensatorer för att få bort eventuella störningar i signalen [3].

Spänningen som tas ut ur modulerna är 5VDC och strömmen är 1A respektive 3.3VDC och strömmen 0,9A, vilket är tillräckligt för kretskortet. I figur 2.3 visas HLK-5M05 modulen, HLK-PM03 har ett likadant utseende. Modulerna kommer inte att monteras på ett kretskort utan ska enbart testas på ett kopplingsdäck för att säkerställa ström och spänningsnivåerna.

Figur 2.2: HLK-5M05 modulen som kan monteras på kopplingsdäck.

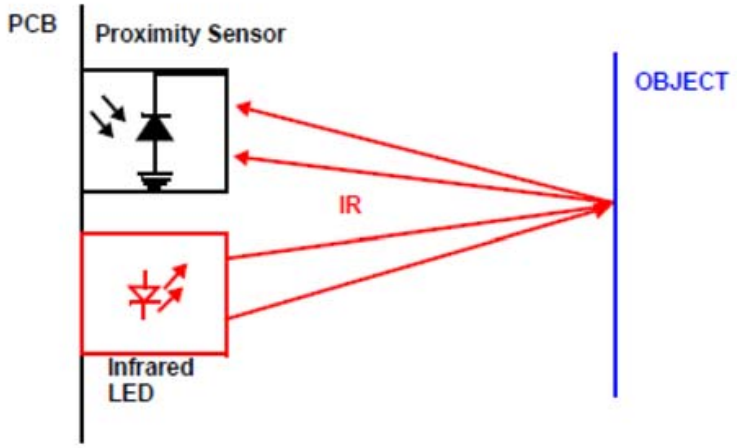
2.2.1.2 Seriell kommunikation

För att kunna kommunicera och överföra mjukvara till ESP-12E så krävs det en USB till UART (Universal asynchronous receiver-transmitter) komponent. UART kommunikation sker seriellt med start- och stoppbit och kräver en motpart på ESP-12E där dataöverföringen sker via de två signalerna Tx och Rx. Där Tx skickas till Rx på mottagaren och Tx från mottagaren skickas till Rx på sändaren [4]. Komponenten skapar via USB, en virtuell COM-port på datorn som sedan kan användas för att programmera ESP-12E. CP2104 som är en vanlig USB till UART komponent har använts i detta projekt, i detta fallet så har Tx skickats till RXD på ESP-12E och TXD på ESP-12E har skickats till Rx på CP2104.

2.2.1.3 Närhetssensor

För att användaren ska kunna sätta på och stänga av solljuset så ska en närhetssensor användas. Därmed behöver inte användaren trycka på en knapp för att styra ljuset. Användaren ska kunna placera sin hand framför kretskortet för att sätta på alternativt stänga av ljuset.

Närhetssensorn som valdes var VCNL-4020. Sensorn är en integrerad närhetssensor baserad på infrarött (IR) ljus. Sensorn ska kunna känna av objekt upp till 200mm ifrån den. Det finns en integrerad krets med signalbehandling och den stödjer I2C kommunikation [5]. IR dioden kommer att avge IR strålning och då den träffar ett objekt så kommer strålningen att reflekteras tillbaka mot sensorn. Den reflekterande IR strålningen som träffar sensorns fotodiod där strålningen konverteras till en elektrisk signal. [6]. Denna signalbehandlas av enheten och visas sedan i ett register inom I2C kommunikationen.



Figur 2.3: Principen bakom närhetssensor med IR-sensor, från [7].

Då kommunikationen mellan ESP-12E och VCNL-4020 kommer att ske via I2C så krävs det två pins som är dedikerade till I2C på ingången till ESP-12E, SCL och SDA. Som nämnt i avsnitt 2.2 så är GPIO4 och GPIO5 på ESP-12E till för I2C kommunikationen. Det går även att koppla flera enheter som stödjer I2C till GPIO4 och GPIO5 då det senare i programmeringen går att urskilja dessa enheter från varandra eftersom varje enhet har en specifik adress.

2.2.2 Utsignaler

Samtliga utsignaler är till för att ge användaren feedback och information kring systemet. Detta kommer huvudsakligen att ske via en NeoPixel som är en smart LED RGB-diod. NeoPixel kan programmeras till alla olika färger och med hjälp av detta så kan systemets tillstånd visas i form av olika färger.

2.2.2.1 Skärm

Möjligheterna för en skärm diskuterades i början av projektet men valdes sedan att inte tas med under detta projekt på grund av tidsbegränsning. Däremot så finns möjligheten för vidareutveckling och implementation av en skärm då det finns 4 pinnar på kretskortet som är dedikerade till SPI-kommunikation vilket generellt används av de flesta LED-skärmar. För att kunna kommunicera med en skärm så är det bara ytterligare programmering som krävs.

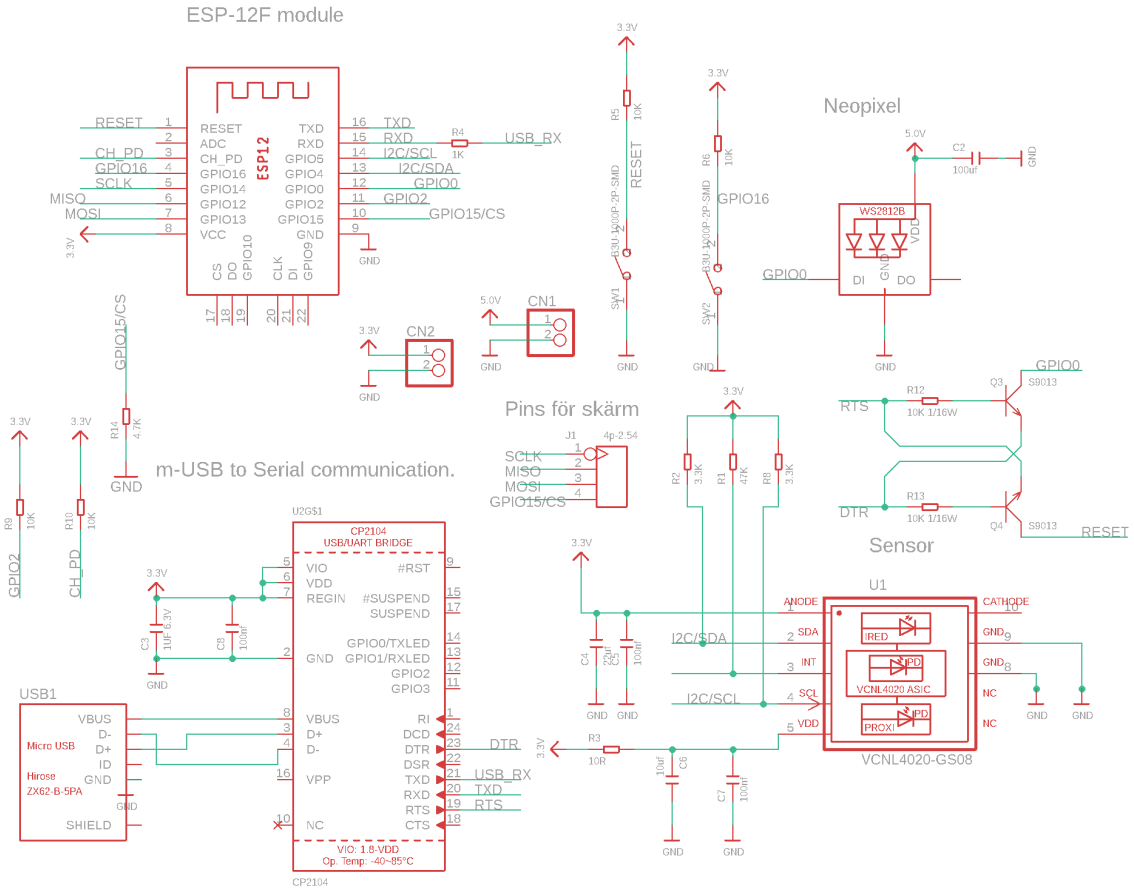
2.3 Autodesk Eagle

Easily Applicable Graphical Layout Editor (Eagle) är ett Electronic Design Automation (EDA) program som används för att designa kretsscheman, kretskortsutformning och skapa filer för Computer-Aided Manufacturing (CAM) [8]. Eagle har även övriga funktioner som t.ex autorouting som underlättar vid designen av kretskortet.

2.3.1 Kretsschema

För att kunna använda olika komponenter så behövs dessa finnas i ett bibliotek på Eagle och om de inte finns så måste en egen komponent skapas i Eagle med hjälp av komponentens datablad. För att underlätta konstruktionen så användes till mesta del färdiga komponenter som finns i diverse bibliotek som kan laddas in i Eagle. Ett bibliotek som användes mest var leverantörens egna bibliotek med komponenter som de har i sitt sortiment. Genom att använda dessa komponenter så var det en större chans till att leverantören kunde tillverka kretskortet utan några problem.

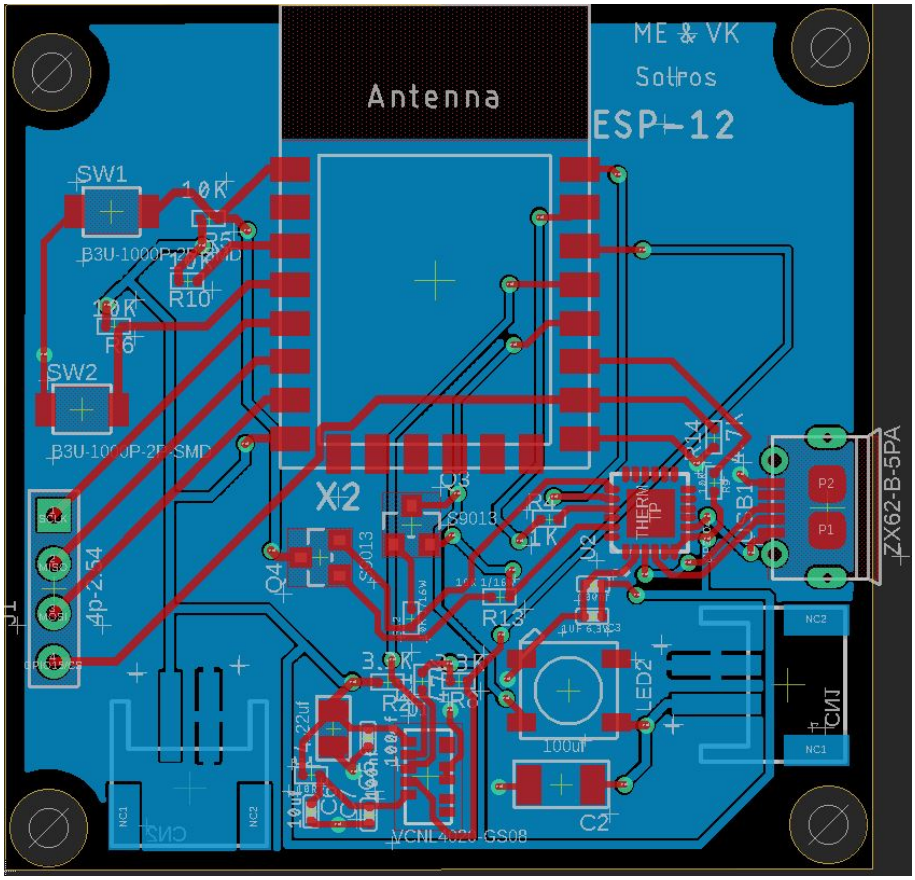
När samtliga komponenter hade valts ut så kopplades komponenterna ihop via ledningar. Det går även att namnge två portar till samma namn och då förstår programmet att det ska vara en ledning mellan dessa två. Hela kretsschemat visas i figur 2.4.



Figur 2.4: Kretsschemat för kretskortet där sammankopplingen mellan samtliga komponenter visas.

2.3.2 Kretskorts layout

När kretsschemat var klart så påbörjades andra delen av konstruktionen där kretskortets layout skulle designas. Komponenterna placerades ut på kretskortsytan i programmet där tillhörande komponenter placerades i närheten av varandra. Möjligheten till autorouting användes inte då det lätt kan bli fel med dimensioner på ledningarna. Matningsspänningens ledningar tillsammans med jord lades på undersidan av kretskortet och resterande signalledningarna lades på övre sidan av kretskortet. Vissa ledningar korsade varandra så att det krävdes en via som är ett hål på kretskortet för att kunna komma till undre lagret och där passera den andra ledningen. Kretskortsutformningen visas i figur 2.5, Allt som är blått på layouten ligger på det undre lagret och allt som är rött ligger på det övre lagret.



Figur 2.5: Utformning av mönsterkort.

2.4 Tillverkning av kretskort

Tillverkningen av kretskortet gjordes av Seeed Studio som är baserade i Kina. Seeed Studio erbjuder anpassade tillverkningsmöjligheter för kretskort. För att tillverkaren ska kunna konstruera ett kretskort så behöver de ritningen i det så kallade Gerber formatet. Gerber formatet är ett filformat som används av maskiner inom kretskortstillverkning för att systemet ska kunna tolka kretsschemat och layouten och producera ett verkligt kretskort utav det. Eagle har en inbyggd funktion som

genererar Gerber filer från de kretskortsfiler som redan finns.

Om tillverkaren även ska löda på samtliga komponenter så krävs det en fil för "Bill of Materials" (BOM) som specificerar alla komponenters nummer på layouten, "Manufacturer Part Number" (MPN) och antalet komponenter. I figur 2.6 så visas BOM-filen som skickades till Seeed Studio tillsammans med Gerber filerna.

Parts	MPN	Qty	Device
CN2	S2B-PH-SM4-TB(LF)(SN)	1	JST_2PIN
CN1	S2B-PH-SM4-TB(LF)(SN)	1	JST_2PIN
C5, C7, C8	CC0402KRX7R8BB104	3	CERAMIC-100NF-25V-10%-X7R(0402)
C2	GRM31CR60J107ME39L	1	CERAMIC-100UF-6.3V-20%-X5R(1206)
R5, R6, R9, R10	RC0402FR-0710KL	4	SMD-RES-10K-1%-1/16W(0402)
R12, R13	RC0402FR-0710KL	2	SMD-RES-10K-1%-1/16W(0402)
R3	RC0402FR-0710RL	1	SMD-RES-10R-1%-1/16W(0402)
C6	CC0402MRX5R5BB106	1	CERAMIC-10UF-6.3V-20%-X5R(0402)
R4	RC0402JR-071KL	1	SMD-RES-1K-5%-1/16W(0402)
C3	CC0402KRX5R5BB105	1	CERAMIC-1UF-6.3V-10%-X5R(0402)
C4	CC0805MKX5R5BB226	1	CERAMIC-22UF-6.3V-20%-X5R(0805)
R2, R8	RC0402JR-073K3L	2	SMD-RES-3.3K-5%-1/16W(0402)
R14	RC0402JR-074K7L	1	SMD-RES-4.7K-5%-1/16W(0402)
R1	RC0402JR-0747KL	1	SMD-RES-47K-5%-1/16W(0402)
J1	P125-1104A0BS116A1	1	DIP-BLACK-MALE-HEADER-VERT(4P-2.54)
SW1, SW2	B3U-1000P-2P-SMD	2	SMD-BUTTON(2P-3.0X2.5X1.2+0.4MM)-B3U-
U2	CP2104-F03-GMR	1	CP2104
X2	ESP-12E	1	ESP-12E
Q3, Q4	S9013	2	SMD-TRANSISTORS-NPN-25V-500MA-S9013
U1	VCNL4020-GS08	1	VCNL4020-GS08
LED2	1655	1	WS2812B5050
USB1	7X62-B-5PA	1	MICRO-USB-SMD(7X62-B-5PA)

Figur 2.6: Lista över komponenter som BOM-fil.

3
Mjukvarudesign

I detta kapitel kommer mjukvarudesignen att beskrivas. Detta kapitel kommer att inkludera både metod samt teori för att få en djupare förståelse av designen och valen som har gjorts.

3.1 Förstudie

Då kompetensen inom mjukvarudesgin var relativ låg så krävdes en stor del förstudier för att kunna genomföra projektet. Information och kunskap om programvara som behövdes har tillhandahållit av Jon Ramstedt. Då det var bestämt att WiFi modulen ESP-12E skulle användas så undersöktes dess upplägg och funktioner för att kunna få en överblick av hur designen skulle se ut. I uppstarten av designen av mjukvaran så undersöktes olika funktioner och bibliotek för att få en grund i hur programspråket fungerade. Dessa undersökningar kunde testas och verifieras på utvecklingskortet Adafruit ESP-8266 HUZZA. Utvecklingskortet användes under tiden då kretskortet ej var färdigställt.

3.2 MicroPython

MicroPython valdes som programmeringsspråk för att det är ett språk som är anpassat för mikrokontrollers och för att det anses vara snabbt och ha en enkel användarvänlighet.
MicroPython använder sig av en större del av Pythons standardbibliotek och är ett väldigt effektivt sätt att implementera och kontrollera hårdvara utan att använda sig av lågnivåprogrammeringsspråk så som C eller C++. Det som urskiljer MicroPython ifrån andra inbyggda system är att den kan använda sig utav ett interaktivt Read-Eval-Print Loop (REPL)[9]. Det som menas med interaktivt REPL är då att det går att exekvera kod utan att kompilera eller att ladda upp på mikrochippet. Detta gör att det går att experimentera och testa kod direkt med återkoppling om hur mikrokontrollern behandlar programmet. En till sak som skiljer MicroPython från andra inbyggda system är det stora omfattande mjukvarubiblioteket. I MicroPythons bibliotek finns det till exempel sätt att analysera och bearbeta JavaScript Object (JSON) Notation filer och även uppkoppling och kommunikation av nätverk [9].

3.3 Programvara

För att kunna utveckla, överföra och testa kod så behövs fyra olika programvaror.

3.3.1 Atom

Atom är namnet på den text- och källkodsredigerare som används för att skriva och redigera kod.

3.3.2 Kommandotolken

Kommandotolken en ett skalprogram som finns inuti Windows operativsystem och är till för att köra enklare program så som skript. I detta fallet används kommandotolken som ett verktyg för att kunna installera MicroPython på mikrochippet men även för att kunna ladda upp program på mikrochippet. Om mjukvarufel skulle uppstå på mikrochippet så finns det också en funktion för att kunna återställa mikrochippet helt.
För att utföra dessa kommandon så behövs två saker. Först behöver man installera Python 3 på datorn och sedan behöver man installera Adafruit MicroPython tool (Ampy). Ampy är den funktionen som gör det tillgängligt att ladda upp program på mikrochippet men ger också möjligheten att testa kod direkt i kommandotolken.

3.3.3 PuTTy

PuTTy är ett program som kan sätta upp en seriell konsol mellan datorn och mikrochippet. Denna seriella kommunikationen sker via USB och tilllåter programmeraren att se vad som händer på mikrochippet. PuTTy fungerar även som ett interaktiv REPL, vilket är beskrivet ovan.

3.3.4 Postman

Postman är namnet på den programvara som används för att testa kommunikationen mellan enheten och Solrossystemet. Postman fungerar som en virtuell simulering av Solrossystemet. Postman kan både skicka och ta emot information ifrån ett nätverk. Detta gör att själva Solrossystemet inte behövs användas vid test av enhetens funktionalitet.

3.3.5 Wireshark

Wireshark är en programvara som liknar Postman. Istället för bara skicka och ta emot data så kan Wireshark övervaka ett nätverk och se all trafik på nätverket. Detta används för att kontrollera när ett kommando skickas till Solrossystemet.

3.4 Nätverksuppkoppling

Nätverksuppkopplingen är en grundläggande del för att enheten ska kunna kommunicera med Solrossystemet. Informationen som skickas mellan enheten och Solrossystemet är över WiFi. Detta betyder alltså att både Solrossystemet och enheten som konstrueras i detta examensarbete måste vara uppkopplade emot samma lokala nätverk som till exempel en router.

Som det nämndes under delen om MicroPython så finns det olika smarta bibliotek som är anpassade för inbyggda system. Ett av dessa bibliotek heter Network och det är en modul som konfigurerar WiFi-uppkoppling till ett lokalt nätverk. Mikrochippet kommer agera som en station vilket innebär att den kommer att koppla upp sig till ett existerande nätverk.
Nätverksuppkopplingen fungerar så att en station aktiveras på mikrochippet och sen kollar programmet om det redan finns någon uppkoppling. Annars kopplar den upp sig till det nätverket som angetts i koden. Programmet kollar ständigt om det finns någon uppkoppling eller inte, och återupptar uppkoppling om det inte finns någon. Denna programkod kommer laddas upp på mikrochippet vid namnet boot.py för att den ska köras direkt när mikrochippet startar om.

```
1  import network
2
3  sta_if = network.WLAN(network.STA_IF)
4  sta_if.active(True)
5  if not sta_if.isconnected():
6      print('connecting to network...')
7      sta_if.connect('Solros', 'Victorkung')
8      while not sta_if.isconnected():
9          pass
10 print('network config:', sta_if.ifconfig())
```

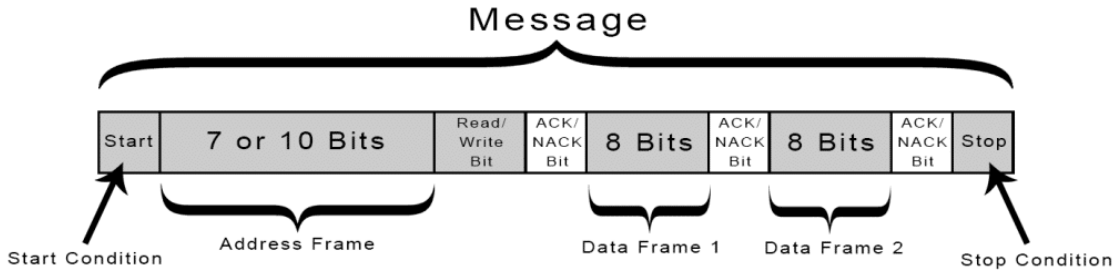
Figur 3.1: Kod för nätverksuppkoppling.

3.5 Närhetssensor

3.5.1 I2C-kommunikation

Närhetssensorn kommunicerar med ESP-12E via I2C kommunikation och som nämnt i avsnitt 2.2.1.3 så används GPIO4(SDA) och GPIO5(SCL) för I2C. I2C är en synkron seriell databuss som används för att överföra data. I2C är inte den snabbaste databussen men väger upp det genom att vara enkel och inte ta upp många pinnar på mikrokontrollern[10]. Varje enhet som är kopplad till I2C bussen har en specifik adress för att kunna urskilja dessa mellan varandra. För att skicka data till en enhet så måste adressen specificeras innan meddelandet skickas [10]. Vissa enheter

har även interna register vars adresser måste specificeras för att kunna styra diverse funktioner som enheten har. I figur 3.2 visas en sekvens där ett meddelande skickas till en enhet, där adressen till enheten först specificeras och sedan skickas meddelandet.



Figur 3.2: Sekvens för att skicka iväg ett meddelande via I2C.

3.5.2 VCNL4020

Enligt VCNL4020s datablad så måste några interna register initieras innan sensorn kan användas. I figur 3.3 visas adresserna i hexadecimal form där 0x13 är adressen till VCNL4020 på I2C bussen och 0x80, 0x82, 0x83 och 0x84 är adresserna till de interna register där dessa register initieras enligt databladet [5]. Resultatet av närhetssensorns mätningar hamnar i ett register på adress 0x87 och 0x88 då resultatet är 16 bitar. Resultaten adderas till en variabel prox som sedan ökar om det finns ett objekt i närheten av sensorn.

```
27          i2c = machine.I2C(scl=machine.Pin(5), sda=machine.Pin(4))
28
29          i2c.writeto_mem(0x13, 0x80, bytearray([255]))
30          i2c.writeto_mem(0x13, 0x82, bytearray([7]))
31          i2c.writeto_mem(0x13, 0x83, bytearray([15]))
32          i2c.writeto_mem(0x13, 0x84, bytearray([157]))
33
34          data = i2c.readfrom_mem(0x13, 0x87, 2)
35          prox = data[0] * 256 + data[1]
36
37          await asyncio.sleep(0.01)
```

Figur 3.3: Initiering av närhetssensorn VCNL4020.

3.6 Av/På funktion

Av/På funktion är den delen där användaren kan sätta på och stänga av enheten. Hårdvarumässigt så är Av/På funktionen en närhetssensor och inte en vanlig strömbrytare. Därför krävs det mjukvarukontroll för att den ska kunna hamna antingen i ett av eller på läge.

Närhetssensorn skickar ett värde till en variabel som ligger runt 3000-3100 när den ej är aktiv. När väl närhetsensorn är aktiv så mottages ett ökande värde över 3000-3100 beroende på hur långt bort handen är ifrån sensorn. Som nämndes tidigare så sparas detta värdet i variabeln prox. För att funktionen ska kunna bestämma ett Av/På läge så finns det en ytterligare variabel vid namn ButtonState. Denna variabel är till för att veta vilket läge enheten är i. Funktionen kollar värdet på prox variabeln och om det värdet överstiger 3150 så kommer ButtonState variabeln att byta värde ifrån av till på eller tvärtom. Efter funktionen har bytt läge så startar en timer på 3 sekunder. Denna timer är till för att funktionen inte ska kunna skifta läge för snabbt.

```
25      async def button(self):
26
27          i2c = machine.I2C(scl=machine.Pin(5), sda=machine.Pin(4))
28
29          i2c.writeto_mem(0x13, 0x80, bytearray([255]))
30          i2c.writeto_mem(0x13, 0x82, bytearray([7]))
31          i2c.writeto_mem(0x13, 0x83, bytearray([15]))
32          i2c.writeto_mem(0x13, 0x84, bytearray([157]))
33
34          await asyncio.sleep(0.01)
35
36          while True:
37              data = i2c.readfrom_mem(0x13, 0x87, 2)
38              prox = data[0] * 256 + data[1]
39              await asyncio.sleep(0.1)
40              if prox > 3150:
41                  self.buttonState = not self.buttonState
42                  print(self.buttonState)
43                  await asyncio.sleep(3)
```

Figur 3.4: Kod för Av/På knapp

En uppkoppling till Solrossystemet skapas när enheten byter läge. Denna uppkoppling är till för att kunna skicka information till Solrossystemet. Den information som skickas från enheten är en JSON sträng som innehåller information om huruvida Solrossystemet ska sättas igång eller inte. Hur denna kommunikation fungerar beskrivs i kapitlet 3.7.1.
Förutom att skicka iväg information när enheten växlar läge så meddelas även användaren om enhetens skifte av läge. Detta görs via en NeoPixel. NeoPixeln visar grönt när enheten är i På-läge och rött när den är i Av-läge.

3.7 Asynchronous I/O

Asynchronous I/O även kallat asyncio är ett bibliotek för att kunna skriva kod som exekveras samtidigt. Asyncio är också användbart för att skapa ett ramverk för högpresterande nätverk och webservrar. Asyncio har en I/O-bunden karakteristik vilket betyder att den väntar på att det ska ske en förändring på antingen en input eller en output för att en operation ska köras. Dessa är egenskaper som är grundläggande för att koden ska kunna fungera då det finns funktioner som måste kunna exekveras samtidigt.

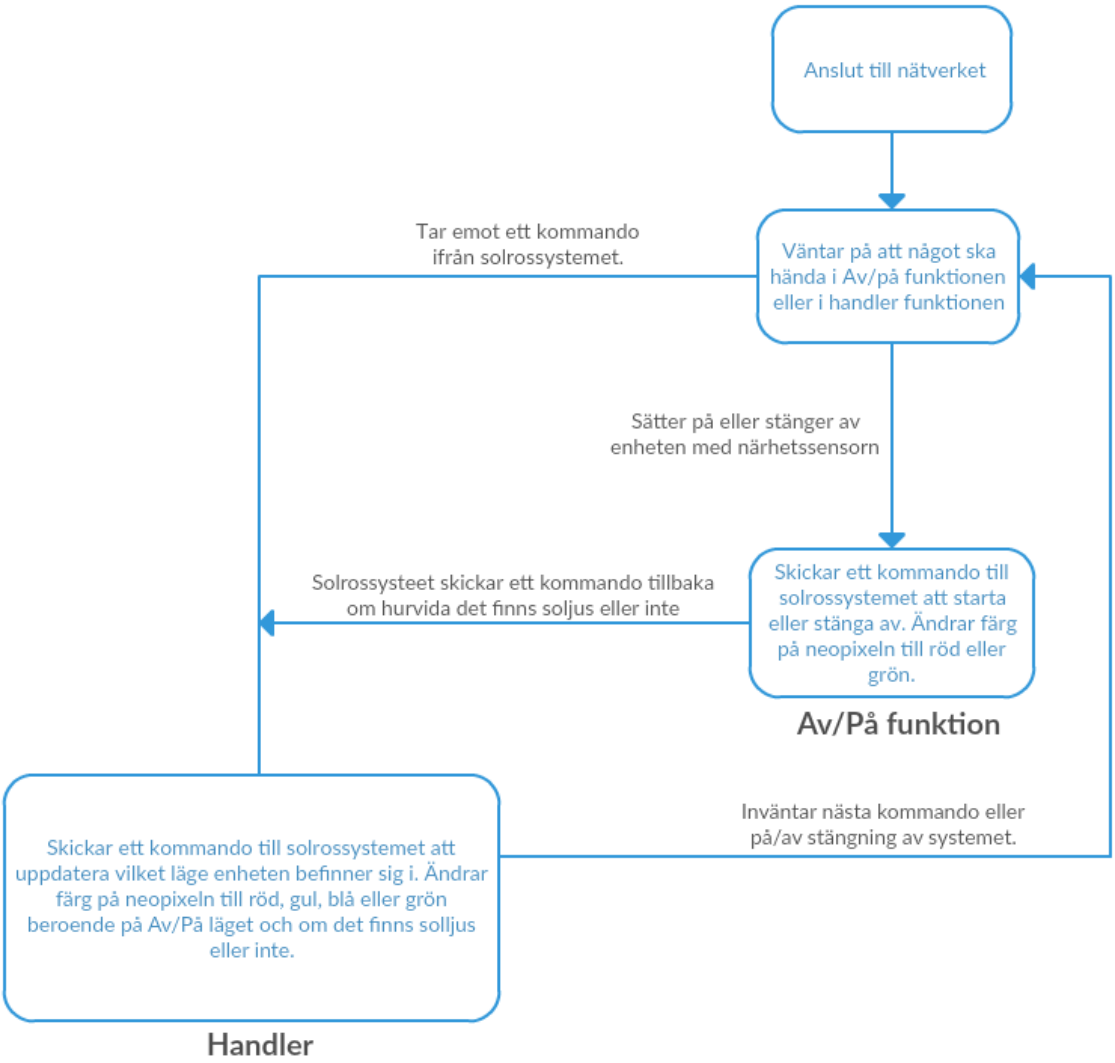
Asyncio används för att enheten ska kunna skicka information över nätet samtidigt som det ändrar Av/På läge. Programmet innehåller två funktioner, en är Av/På funktionen som förklarades tidigare i avsnitt 3.6 och den andra är en funktion vid namn handler. Handler funktionen behandlar mottagningen av information ifrån Solrossystemet men även informationen till användaren. Dessa funktioner använder sig av två olika sorters av kommunikation.

För att båda funktionerna ska kunna köras samtidigt och att de ska starta när någon händer inom funktionerna så används asyncio funktionen get_event_loop. Detta gör att Av/På funktionen startar när den känner av en ändring på en I/O port och handler funktionen startar när ett meddelande (kommando) skickas till enheten. Call_soon är en asyncio funktion som kan ses i figur 3.5. Denna funktion gör att när antingen Av/På funktionen eller handler funktionen är klar, kollar den efter nästa händelse.

```
123  loop = asyncio.get_event_loop()
124  loop.call_soon(asyncio.start_server(Solros.handler, '192.168.43.127', 8888))
125  loop.call_soon(Solros.button())
126  loop.run_forever()
127  loop.close()
```

Figur 3.5: Kod för asyncios loop funktion.

Flödesförloppet förklaras översiktligt i figur 3.6, där kan man se hur funktionerna relaterar till varandra och vad det är som gör att olika funktioner startar.



Figur 3.6: Tillståndsdiagram för hela programmet.

3.7.1 JSON formatet

Vid överföring av information mellan de två enheterna så används ett JSON text-format. JSON används för att det är enkelt för människor att läsa och skriva men även för datorer att analysera och generera koden. Den största anledningen till varför JSON formatet används är för att det är helt oberoende av vilket programspråk som används. Detta betyder att den informationen som skickas kan tas emot av en annan enhet med ett annat programspråk.
Då JSON filen ska skickas eller tas emot måste den först konverteras mellan en sträng variabel och ett objekt. När en JSON fil tas emot så används en Python funktion vid namn loads som konverterar filen från en sträng till ett objekt och när ett objekt skickas så används Python funktionen dumps för att göra konverteringen åt andra hållet [11].

3.7.2 Kommunikationen i Av/På funktionen

Kommunikationen som används i Av/På funktionen är en Python funktion som heter open_connection. Open_connection är en del av asyncio biblioteket och fungerar så att den öppnar upp ett dataflöde emot ett specifikt nätverk, i detta fallet är det Solrossystemets nätverk. För att öppna uppkopplingen så används den IP adress som Solrossystemet tillhandahåller.
Den information som skickas ifrån Av/På funktionen är två olika JSON objekt beroende på vilket läge enheten är i. I figure 3.6 visar rad 39 och 40 att de två olika JSON variabler som ska skickas är turn_system_on och turn_system_off. Dessa variabler är dock i ett objektformat och då används Python-funktionen dumps som nämndes tidigare till att omvandla variabeln till en sträng så att den blir redo för att skickas. Eftersom JSON strängen skickas över nätet så behövs det även grundläggande kod för HTTP för att kommandot ska kunna läsas av rätt.
Som det nämndes innan så skickas respektive JSON sträng beroende på vilket läge enheten är i. Asyncio funktionen awrite skickar ut JSON strängen ut på nätverket. Direkt efter kommandot har skickat så stängs uppkopplingen. När Solrossystemet tagit emot detta kommando så kommer den skicka tillbaka ett statuskommando till handler funktionen om huruvida det finns solljus att användas eller inte.

```
52                  print('open connection')
53                  reader, writer = await asyncio.open_connection(host='192.168.43.198', port=8888)
54                  print('connected')
55
56                  sendCommandOn = {"command":"turn_system_on"}
57                  sendCommandOff = {"command":"turn_system_off"}
58                  jsonMessEnON = json.dumps(sendCommandOn)
59                  jsonMessEnOff = json.dumps(sendCommandOff)
60                  httpJsonMessOn = "GET HTTP/1.0 200 OK\r\n\r\n" + jsonMessEnON +"\r\n"
61                  httpJsonMessOff = "GET HTTP/1.0 200 OK\r\n\r\n" + jsonMessEnOff +"\r\n"
62
63                  if self.buttonState == True:
64                      print('write turn on message')
65                      np[0] = (0,20,0)
66                      np.write()
67                      await writer.awrite(httpJsonMessOn)
68                      print('message written')
69                      await writer.aclose()
70                  elif self.buttonState == False:
71                      print('write turn off message')
72                      np[0] = (20,0,0)
73                      np.write()
74                      await writer.awrite(httpJsonMessOff)
75                      print('message written')
76                      await writer.aclose()
77                  await asyncio.sleep(3)
```

Figur 3.7: Kod för kommunikationen i Av/På funktionen.

3.7.3 Kommunikationen för handler

Handler funktionen bearbetar kommandon och upprättar kommunikation mellan enheten och Solrossystemet. Den kommer att ta emot information om det finns något solljus ute att använda och skicka tillbaka vilket Av/På läge enheten är i. Denna kommunikationen sker på ett annorlunda sätt jämfört med kommunikatio-

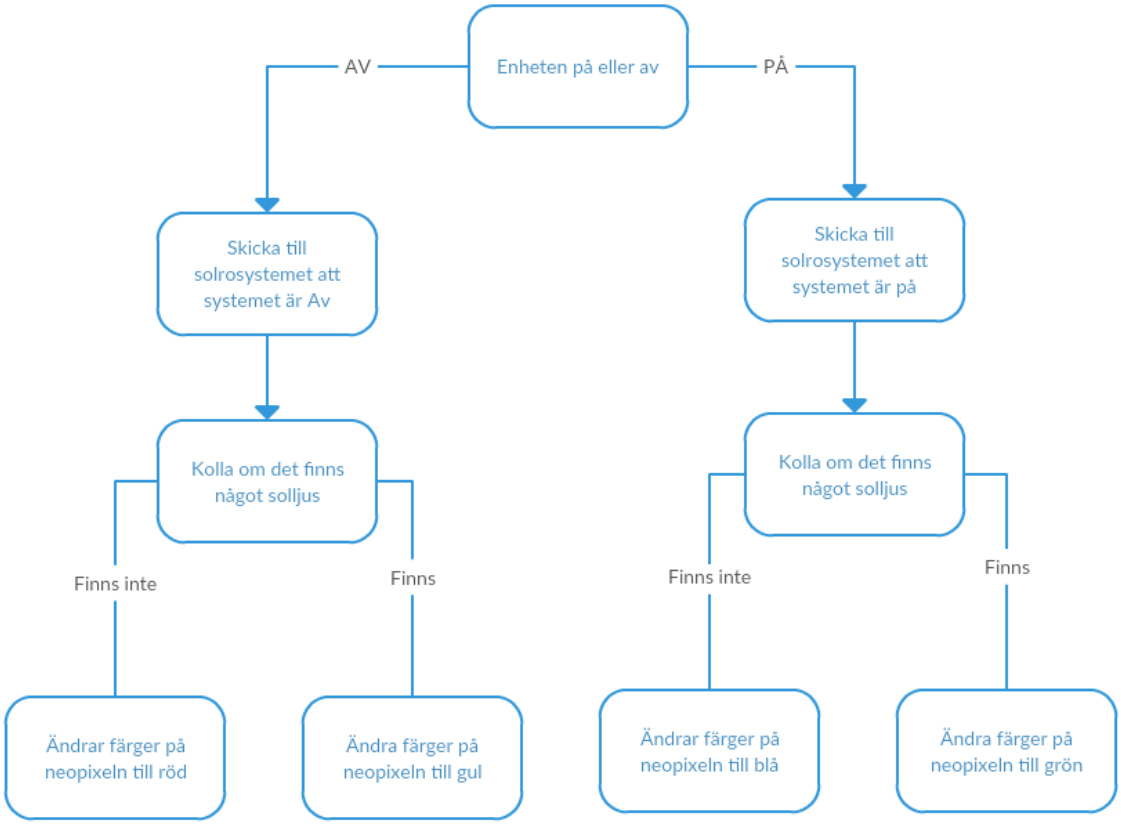
nen i Av/På funktionen gör. I handlern så etableras en uppkoppling av dataflöde upp emot en server. Detta gör att kommunikationen kan ske åt båda hållen samtidigt mellan enheten och Solrossystemet. Enheten kan alltså att ta emot ett kommando och även skicka iväg ett.

Handler funktionen fungerar så att den kopplas upp emot en server och börjat läsa av all data som finns på enhetens IP adress. Datan som lästs av kommer att avkodas för att sen behandlas av Python funktionen loads som förklarades tidigare. Efter att loads funktionen använts så finns det en variabel med ett meddelande (kommando) som erhåller information om huruvida det finns solljus att använda eller inte. Beroende på om det finns solljus eller inte så kommer NeoPixeln att ändra färg för att enkelt informera användaren vilket läge systemet befinner sig i. De olika lägena som enheten kan anta visas i tabellen nedan.

Tabell 3.1: Färgkod för NeoPixel.

	Inget solljus	Solljus
System av	röd	gul
System på	blå	grön

Som nämndes tidigare så användes en server för att både kunna skicka och ta emot information. Innan handler funktionen skickar iväg ett meddelande så kollar den om enheten är på eller av och då skickar den ett kommando till Solrossystemet vilket läge enheten har. Att skicka iväg detta meddelande fungerar på samma sätt som det gör i Av/På funktionen dock så skickar den en annan JSON sträng. I figur 3.8 finns flödesschemat för handler funktionen.



Figur 3.8: Flödesschemat för handler funktionen.

3.8 Klasser

En klass används i programmet för att göra Av/På funktionens buttonState variabel global. Med detta menas då att variabeln som vet om enheten är av eller på går att använda i alla funktioner. Detta är till för att handler funktionen ska kunna ta in och veta när enheten är på eller av för att kunna skicka sitt meddelande (kommando) till Solrossystemet.

Inuti klassen så finns det en funktion som heter ___init___. Denna funktion exekveras så fort klassen initieras. ___init___ funktionen innehåller buttonState variabeln och den deklareras till False till en början.
Eftersom ButtonState variabeln används i båda funktionerna så måste den inkluderas i call_loop funktionerna också [12]. Detta gör att när ButtonState variabeln byter värde så kommer värde följa med in i dem andra funktionerna.

```
8  class solros(object):
9      def __init__(self):
10         self.buttonState = False
```

Figur 3.9: Kod för som gör ButtonState vaiabeln blir global.

4
Resultat

4.1 Kontroll av olika delfunktioner

För att säkerställa att samtliga funktioner i systemet fungerar så har tester av de olika delfunktionerna utförts. Majoriteten av funktionerna fungerade som det var tänkt och funktionerna som inte fungerade redovisas i kapitel nedan.

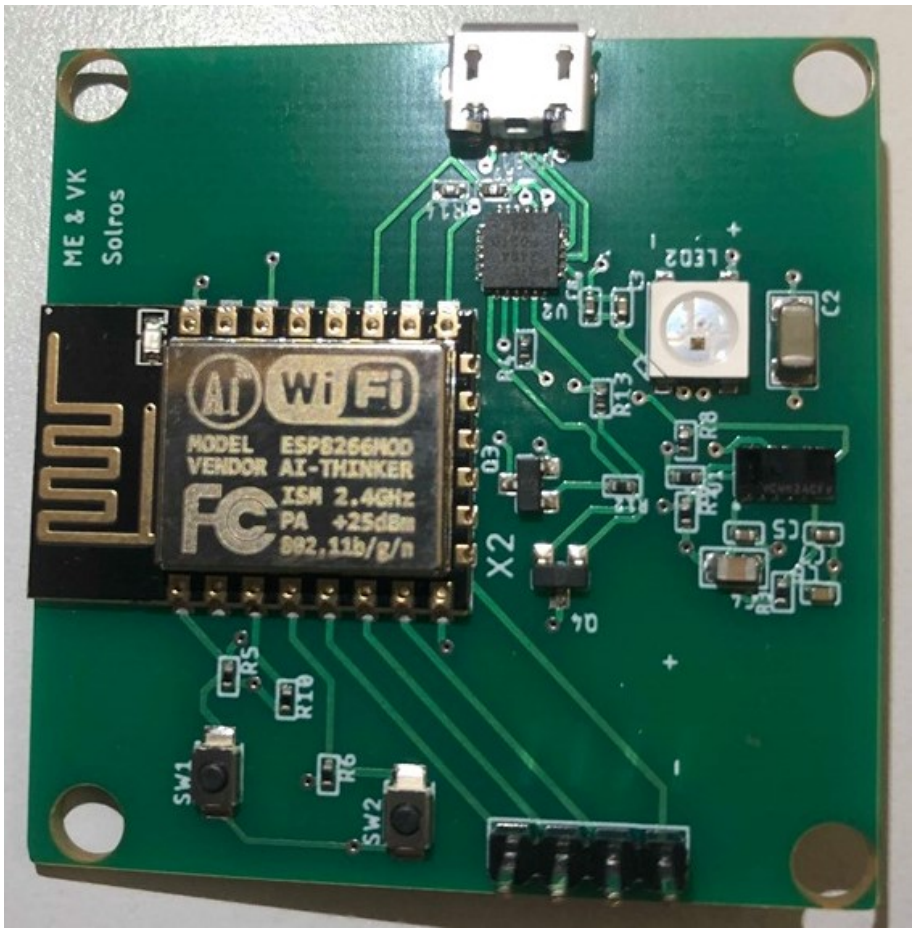
4.1.1 USB till UART

Första testet som utfördes var att försöka programmera mikrokontrollern. När m-USB kopplades in på kretskortet så etablerades ingen COM-port på datorn. Första försöket till att lösa problemet bestod av att uppdatera samtliga drivrutiner för CP2104 USB till UART men problemet löstes inte.

Efter diverse försök med att ändra inställningar på datorn så skiftades fokuset mot kretskortet för att felsöka möjliga faktorer som kan påverka detta. Problemet hittades relativt snabbt då DATA+ signalen från m-USB hade kopplats till DATA- på CP2104. Problemet löstes genom att klippa en m-USB kabel och skala bort det yttre gummiskalet för att sedan löda fast samtliga koppartrådar från båda ändar. DATA- tråden från ena änden kopplades ihop med DATA+ på andra änden. När detta var gjort så kontrollerades kontakten i kabeln genom att med en multimeter kolla resistansen mellan ingången och utgången på kabeln.

Vid testet av den modifierade kabeln så funkade allting och CP2104 etablerade en COM-port på datorn och då fungerade det även att programmera mikrokontrollern.

4.2 Prototyp



Figur 4.1: Kretskortet som används i systemet där huvudkomponenterna ESP-12E, CP2104 och VCNL4020 visas.

I figur 4.1 visas kretskortet som har tillverkats under projektet. Vid test av prototypen så funkar nästan allting som det ska. Ett problem som uppstod var att när Av/På funktionen ska skicka iväg ett meddelande så blev det något fel med open_connection funktionen. Det gjorde att meddelandet inte kommer fram som en JSON sträng utan det blir en krypterat sträng. Detta medför att Solrossystmet bara kommer att få ett Av/På kommando om det skickas ett meddelande till enheten. Återkopplingen som fås av Solrossystemet visas sedan på NeoPixel enligt de färger som visas i tabell 3.1.

Kraven som var uppsatta i början av projektet uppfylls till viss del. Enheten kan anslutas trådlöst via WiFi, kommunicera med övriga system och användaren får återkoppling via en LED-lampa.

5
Diskussion och slutsats

5.1 Arbetsgång

Arbetet under detta projekt delades upp i två delar i början av projektet där delarna var hårdvara och mjukvara. Det fungerade bra och då har båda parterna ett område att fokusera på och kan ägna all sin tid åt det. Då det var tänkt att designa och tillverka ett eget kretskort så var den delen tvungen att bli klar i god tid för att ha tillräckligt med tid för att fixa eventuella fel på kretskortet. Det skulle även finnas tid för att implementera mjukvaran och testa den.

Då det fanns tillgång till Adafruits ESP8266 kretskort med samma ESP-12E modul som använts för vårt kretskort så gick det att testa mjukvaran även om kretskortet inte var klart. I början av mjukvaruprogrammeringen så skapades en del program för att få en djupare förståelse av MicroPython. Därav tog det även längre tid för mjukvaran att bli färdig. Adafruits ESP8266 kretskort hade inte alla funktioner som vårt kretskort har och därför gick det inte att testa vissa funktioner innan vårt kretskort blev klart.

5.2 Kretskort

Då kretskortet skulle vara klart i god tid så uppstod det även slarvfel vid designen av kretsscheman som i efterhand förmodligen tog mer tid att fixa än om mer tid skulle spenderats på att säkerställa att kretsschemat var rätt från början. Vid framtida projekt så är det bra att ha en kompromiss mellan dessa och kunna avgöra om man behöver spendera mer tid för att säkerställa att allting är rätt istället för att stressa fram ett resultat.

5.3 Mjukvara

Under tiden som kretskortet designades så lades större del av tiden ner på att undersöka och testa olika funktioner på utvecklingskortet. Denna tid var mycket längre än förväntan med små framgångar. Det som skulle gjorts bättre var att hålla bättre kontakt med personalen på företaget för att snabbare komma framåt. De största framgångarna gjordes först efter kretskortet hade varit klart i några veckor och då märktes att det som gjorts under tiden kretskortet designades hade lite betydelse

för hela programmet.

Resultatet var inte riktigt det som förväntades då en funktion inte riktigt fungerande som den skulle. Det fanns inte en full insikt i hur Solrossystemet fungerade. Beroende på hur ofta Solrossystemet skickar ett kommando till enheten så kommer enheten att fungera olika bra. Om Solrossystemet skulle skicka ett kommando med ett kort intervall på några sekunder så kommer enheten att fungerar exakt enligt målen. Dock om Solrossystemet skickar kommandon med långa intervallen så kommer det ta längre tid för enheten att byta Av/På läge.

5.4 Slutsats

Målen som var uppsatta i början av projektet har devis uppnåtts men projektet kan vidareutvecklas genom att lägga till ytterligare funktioner som underlättar för användaren. Ett exempel på vidareutveckling är att implementera en skärm på kretskortet då det finns dedikerade pinnar till skärmen på kretskortet. Då inga ekonomiska faktorer togs i åtanke vid konstruktionen av denna enhet så kan även kostanden förbättras genom att välja billigare komponenter. På grund av tidsbegränsande anledningar så lades det inte mycket tid på att undersöka detta. Kretskortet kan ha många användningsområden men det krävs då bara att mjukvaran programmeras om för det specifika systemet.

Litteraturförteckning

[1] S. Saha and A. Majumdar, "Data centre temperature monitoring with esp8266 based wireless sensor network and cloud based dashboard with real time alert system," in *2017 Devices for Integrated Circuit (DevIC)*. IEEE, 2017, pp. 307–310.

[2] *ESP-12E WiFi Module datasheet*, AI-thinker, Apr. 2015, rev. 1.0.

[3] G. Barbehenn and H. W. Rice, "Simplified ac-dc switching converter with output isolation," Jun. 22 1999, uS Patent 5,914,865.

[4] U. Nanda and S. K. Pattnaik, "Universal asynchronous receiver and transmitter (uart)," in *2016 3rd International Conference on Advanced Computing and Communication Systems (ICACCS)*, vol. 01, Jan 2016, pp. 1–5.

[5] *Fully Integrated Proximity and Ambient Light Sensor WithInfrared Emitter, I2C Interface, and Interrupt Function*, Vishay Semiconductors, Mar. 2018, rev. 1.5.

[6] L. Joseph, "6.2 working with the ir proximity sensor," pp. 144–155, 2015. [Online]. Available: https://app.knovel.com/hotlink/khtml/id:kt00UCAOE5/learning-robotics-using/working-with-ir-proximity

[7] "Infrared proximity sensing: Building blocks, mechanical considerations, & design trade-offs," hämtad: 2019-05-21. [Online]. Available: https://www.edn.com/design/industrial-control/4010399/Infrared-proximity-sensing-Building-blocks-mechanical-considerations--design-trade-offs

[8] Autodesk, "Eagle." [Online]. Available: https://www.autodesk.com/products/eagle/overview

[9] "Overview — MicroPython 1.10 documentation," hämtad: 2019-05-21. [Online]. Available: https://docs.micropython.org/en/latest/index.html

[10] N. Semiconductors, "Um10204 i2c-bus specification and user manual," *User Manual*, vol. 4, 2014.

[11] "JSON," hämtad: 2019-05-21. [Online]. Available: https://www.json.org/

[12] "Python Classes," hämtad: 2019-05-21. [Online]. Available: https://www.w3schools.com/python/python_classes.asp

A

Appendix 1

```
import uasyncio as asyncio
import ujson as json
import time
import machine
from machine import Pin, Signal
from neopixel import NeoPixel

class solros(object):
    def __init__(self):
        self.buttonState = False


    async def button(self):

        i2c = machine.I2C(scl=machine.Pin(5), sda=machine.Pin(4))

        i2c.writeto_mem(0x13, 0x80, bytearray([255]))
        i2c.writeto_mem(0x13, 0x82, bytearray([7]))
        i2c.writeto_mem(0x13, 0x83, bytearray([15]))
        i2c.writeto_mem(0x13, 0x84, bytearray([157]))

        await asyncio.sleep(0.01)

        while True:
            data = i2c.readfrom_mem(0x13, 0x87, 2)
            prox = data[0] * 256 + data[1]
            await asyncio.sleep(0.1)
            if prox > 3100:
                self.buttonState = not self.buttonState
                print(self.buttonState)

                np = NeoPixel(Pin(0),1)

                print('open connection')
                reader, writer = await
                    asyncio.open_connection(host='192.168.43.198',
```

```
                port=8888)
            print('connected')

            sendCommandOn = {"command":"turn_system_on"}
            sendCommandOff = {"command":"turn_system_off"}
            jsonMessEnON = json.dumps(sendCommandOn)
            jsonMessEnOff = json.dumps(sendCommandOff)
            httpJsonMessOn = "GET HTTP/1.0 200 OK\r\n\r\n" +
                jsonMessEnON +"\r\n"
            httpJsonMessOff = "GET HTTP/1.0 200 OK\r\n\r\n" +
                jsonMessEnOff +"\r\n"

            if self.buttonState == True:
                print('write turn on message')
                np[0] = (0,20,0)
                np.write()
                await writer.awrite(httpJsonMessOn)
                print('message written')
                await writer.aclose()
            elif self.buttonState == False:
                print('write turn off message')
                np[0] = (20,0,0)
                np.write()
                await writer.awrite(httpJsonMessOff)
                print('message written')
                await writer.aclose()
            await asyncio.sleep(3)


async def handler(self, reader, writer):


    np = NeoPixel(Pin(0),1)

    # Json commands
    sendCommandOn = {"command":"system_on"}
    sendCommandOff = {"command":"system_off"}
    jsonMessEnON = json.dumps(sendCommandOn)
    jsonMessEnOff = json.dumps(sendCommandOff)
    httpJsonMessOn = "HTTP/1.0 200 OK\r\n\r\n" + jsonMessEnON +"\r\n"
    httpJsonMessOff = "HTTP/1.0 200 OK\r\n\r\n" + jsonMessEnOff +"\r\n"


    # Reading data
    print(reader, writer)
    readData = await reader.read()
    data = readData.decode().split('\r\n')
    message = json.loads(data[-1])
```

```
        print('\r\n -------Message data ----------- ')
        print(message)
        print('\r\n ----------- Message data ----------- ')

        print('\r\n ----------- App data ----------- ')
        if self.buttonState == True:
            await asyncio.sleep(0.01)
            await writer.awrite(httpJsonMessOn)
            print("Close the connection")
            await writer.aclose()
            print(jsonMessEnON)
            if message["command"] == 'light_on':
                print('System ON and Light ON')
                np[0] = (0,20,0)
                np.write()
            elif message["command"] == 'light_off':
                print('System ON and Light OFF')
                np[0] = (0,0,20)
                np.write()
        elif self.buttonState == False:
            await asyncio.sleep(0.01)
            await writer.awrite(httpJsonMessOff)
            print("Close the connection")
            await writer.aclose()
            print(jsonMessEnOff)
            if message["command"] == 'light_on':
                print('System OFF and Light ON')
                np[0] = (20,20,0)
                np.write()
            elif message["command"] == 'light_off':
                print('System OFF and Light OFF')
                np[0] = (20,0,0)
                np.write()
        print('\r\n ----------- App data ----------- ')


Solros = solros()


loop = asyncio.get_event_loop()
loop.call_soon(asyncio.start_server(Solros.handler, '192.168.43.127',
    8888))
loop.call_soon(Solros.button())
loop.run_forever()
loop.close()
```