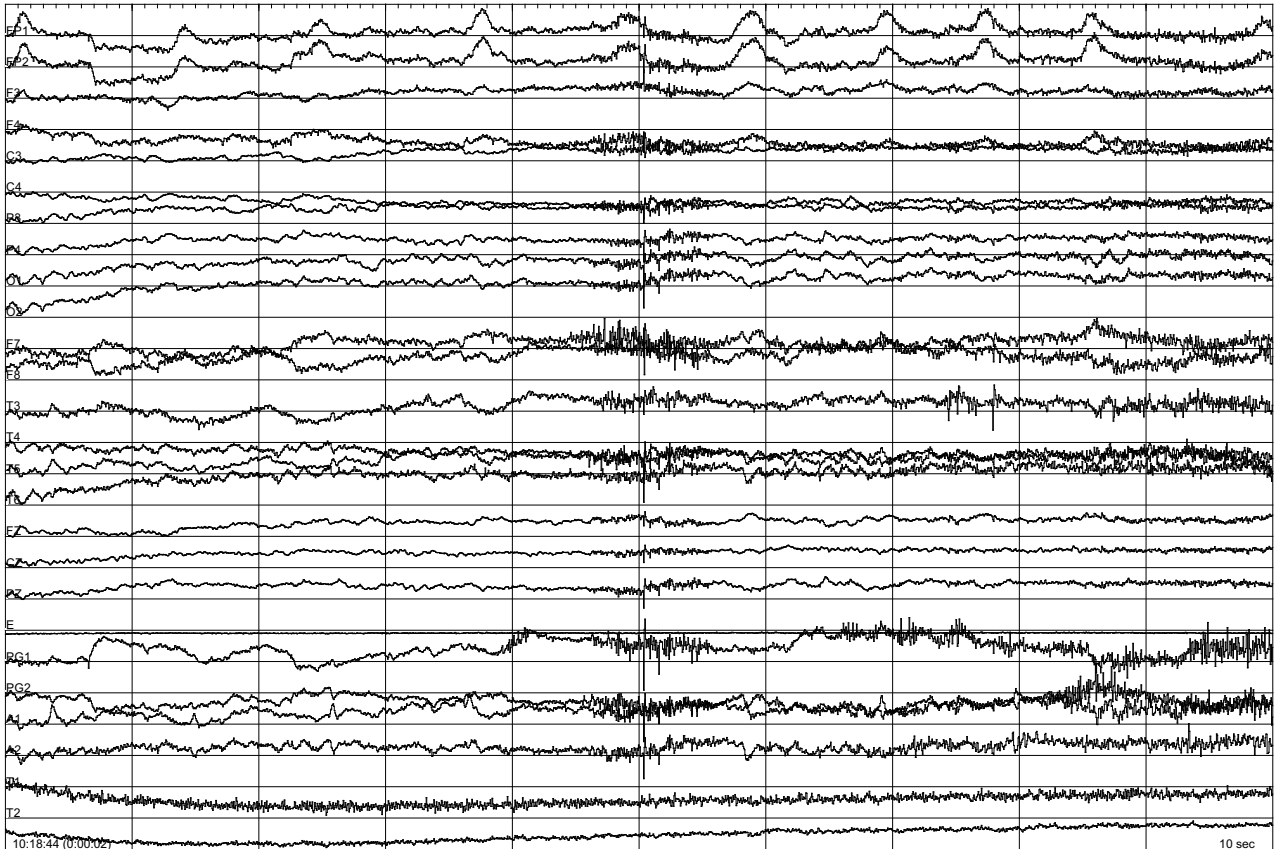




CHALMERS

ma0844az 1-1+.edf 1234567 M 09-APR-1955 L. Smith 15 sep 2005 2 Kesteren Nihon Kohden EEG-1100C V01.00



Mobilt Elektroencefalografi

Examensarbete i Data och -Informationsteknik

JOHN CROFT
CHRISTOFFER OLSSON

EXAMENSARBETE

Mobilt Elektroencefalografi

JOHN CROFT
CHRISTOFFER OLSSON

Institutionen för Data och -Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET

Göteborg, Sverige 2019

Mobilt Elektroencefalografi
JOHN CROFT
CHRISTOFFER OLSSON

© JOHN CROFT, CHRISTOFFER OLSSON, 2019

Examinator: Peter Lundin

Institutionen för Data och -Informationsteknik
Chalmers Tekniska Högskola/Göteborgs Universitet
SE-412 96 Göteborg
Sverige
Telefon: +46 (0)31-772 1000

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet. The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

Omslag:
Typisk elektroencefalogram.

Institutionen för Data och -Informationsteknik
Göteborg, Sverige 2019

Mobilt Elektroencefalografi

JOHN CROFT

CHRISTOFFER OLSSON

Institutionen för Data och -Informationsteknik

Chalmers Tekniska Högskola/Göteborgs Universitet

Examensarbete

SAMMANFATTNING

Dagens teknologi fortsätter alltjämt att utvecklas. För att fortsätta denna trend på samma sätt, så ämnar detta projekt att skapa en produkt för att möjliggöra en mobil implementation av ett EEG. Projektet är gjort hos *Institutionen för Data- och Informationsteknik* på Chalmers.

Ett elektroencefalografi (EEG) används för att detektera och mäta jonströmmar i hjärnans nervceller. Dessa kan mätas genom att fästa spänningskänsliga elektroder vid skalpen, och används för att fastställa olika tillstånd i hjärnan så som epilepsi, koma och hjärndödhet.

Tanken bakom projektet är att dess mobilitet och lättillgänglighet kan öppna upp möjligheten till ett stort utbud av data. Denna datamängd kan potentiellt användas för att analysera hur människan katalogiserar det hon lär sig och hur olika extern stimuli påverkar denna process.

Grundkrav inkluderar att det ska fungera precis som ett vanligt EEG, samt att det ska finnas ett enkelt sätt att interagera med mjukvaran. Det skall också finnas ett sätt att spela in ljud och bild från användarens perspektiv.

Projektet har följaktligen utvecklat en hårdvarudesign för att samla in EEG-data, samt utvecklat ett GUI för att lätt använda systemet. Projektets utveckling är tyvärr en ofulländad produkt, och en del av de mål som sattes från början har ej uppnåtts. Utvecklingen av projektet har resulterat i kunskap om EEG och datainsamlingssystem, och de begränsningar som finns med programmeringsspråk och hårdvara i att realisera ett digitalt system med realtidskrav.

Resultaten har använts som kunskapsbas till ett annat projekt, och ämnar användas till flera likasinnade utvecklingar i framtiden.

Nyckelord: EEG, Python, Raspberry Pi, tkInter, ADC, EDF+

ABSTRACT

Today's technology is constantly undergoing new innovations and evolutions. In keeping with this trend, this project aims to take the well-established measuring technique of electroencephalography (EEG) and create a mobile implementation. The project has been performed at Chalmers.

An Electroencephalography (EEG) is used to detect and measure ion currents in the brains nerve cells. These can be measured by attaching voltage sensitive electrodes on the scalp, and is used to determine difference conditions in the brain, such as epilepsy, coma and brain death.

The idea behind the project is that its mobility and availability can open an opportunity for large amounts of data. This large data set can potentially be used to analyse how humans catalogue what she learns, and how varying external stimuli affects this process.

Basic requirements include that it should function as a normal EEG, as well as there should exist a simple way to interact with the software.

The project has consequently developed hardware to collect EEG-data, as well as created a GUI for easy use of the system. The project development is unfortunately an unfinished product, and some of the initial goals have not yet been completed. The results of the project has resulted in a more in-depth knowledge of what an EEG is, and also the limitations that exists in programming languages and hardware when one tries to create a real time system.

The results have been used as a knowledge base for another project, and aims to be used for more like-minded projects in the future.

Keywords: EEG, Python, Raspberry Pi, tkInter, ADC, EDF+

TERMINOLOGI

$\Sigma\Delta$-modulation	Metod som gör det möjligt att skifta bruskomponenter från lägre frekvenser till högre frekvenser, utanför signalbandet, där de sedan kan filtreras bort med ett digitalt filter.
ADC	Analog-to-Digital Converter. Omvandlar analoga signaler till en digital representation.
ADS1299	Analog-till-digital omvandlare från Texas Instruments avsedd för biopotentiala mätningar.
ALSA	Advanced Linux Sound Architecture. En linux kernelmodul som ger tillgång till ett programmeringsgränssnitt för ljudkortsdrivrutiner.
Baud	Ett mått på dataakt. Enheten är <i>symboler</i> per sekund, där en symbol är en unik signalkodning. I digitala system används uteslutande binär representation på information och därmed kan baud tolkas som bitar per sekund i sådana sammanhang.
BOM	Bill of Materials. Lista över komponenter som ingår i en design. Genereras ofta automatiskt av EDA verktyg och kan eventuellt innehålla mer information såsom pris, länk till återförsäljare mm.
CAD	Computer Aided Design. Ett generellt begrepp på datorprogram som används för att skapa tekniska ritningar eller modeller i 2D och/eller 3D.
CSI	Camera Serial Interface. Standardiserad gränssnitt mellan en kameramodul och en processor.
EDA	Electronic Design Automation. Ett löst begrepp på olika datorprogram vars syfte är att underlätta design av elektroniska system, speciellt PCB design.
EDF+	European Data Format. Filformat som används för lagring av bioelektriska signaler, speciellt elektroencefalografiska signaler.
EEG	Elektroencefalografi. Mätning av elektriska signaler i hjärnan.
EKG	Elektrokardiografi. Mätning av elektriska signaler i hjärtat.
GUI	Grafiskt användargränssnitt (eng. Graphical user interface)
Half-cell potential	DC-spänning som utvecklas över gränssnittet mellan hud och elektrod.
MCU	Mikrokontroller. Förkortning på engelskans <i>microcontroller unit</i> .
OS	Operativsystem.
PCB	Mönsterkort (eng. Printed circuit board).
PGA	Programmerbar förstärkare (eng. Programmable Gain Amplifier). En förstärkare där förstärkningskvoten kan ställas in digitalt.
RPi	Raspberry Pi. En s.k. <i>enkortsdator</i> med processor och all kringutrustning på en och samma krets. Kör ett fullfjädrat Linux operativsystem (Debian).
SNR	Signal-brusförhållande (eng. Signal-to-noise ratio)
SPI	Serial Peripheral Interface. Ett seriellt, synkront, två-vägs kommunikationsprotokoll
SSH	Secure Shell. Ett protokoll för säker anslutning till fjärrdatorer över internet.

INNEHÅLL

Sammanfattning	i
Abstract	ii
Terminologi	iii
Innehåll	v
1 Inledning	1
1.1 Bakgrund	1
1.2 Syfte	1
1.3 Mål	1
1.4 Avgränsningar	2
2 Teknisk Bakgrund	3
2.1 Hårdvara	3
2.1.1 Raspberry Pi	3
2.1.2 Mikrokontroller	3
2.1.3 Analog till Digital Omvandlare (ADC)	4
2.1.4 Batteri	4
2.1.5 Kamera	4
2.1.6 Mikrofon	4
2.1.7 PCB	5
2.2 Mjukvara	6
2.2.1 Python	6
2.2.2 Tk Interface (TkInter)	6
2.2.3 European Data Format (EDF)	6
2.2.4 Linuxprogram	6
2.2.5 Python EDF Library (PyEDFlib)	6
3 Metod	7
3.1 Agila metoder	7
3.2 Git och GitHub	7
3.3 Kunskapsinhämtning	8
4 Teoretisk Bakgrund	9
4.1 Elektroencefalografi	9
4.2 Biopotentialer	9
4.3 Biopotentiala Elektroder	10
4.4 Kliniskt EEG	10
5 Genomförande	12
5.1 Förberedelse	12
5.1.1 Allmänt arbetssätt	12
5.2 Systemdesign	13
5.2.1 Systembeskrivning	13
5.2.2 Datakommunikation inom systemet	14
5.2.2.1 SPI gränssnitt på ADC ADS1299	14
5.2.3 ADC	15
5.2.4 PCB Design	18
5.2.4.1 EDA verktyg och Prototyp-PCB	18
5.3 Mjukvarudesign	20
5.3.1 MobileEEG-app	20
5.3.2 AnnotationWindow	20
5.3.3 HeaderWindow	20

5.3.4	StoppableThread	20
5.3.5	Record	21
5.3.6	Sync	21
5.3.7	EDFFunction	21
5.3.8	ReadEDF	21
5.3.9	Videoplay	21
5.3.10	Videostop	21
5.3.11	Popup	21
5.3.12	PyEDFlib	21
5.3.13	Inspelning	22
5.3.14	Start och Stopp	22
5.3.15	Synkronisering av ljud och bild	22
5.3.16	Uppspelning	22
5.3.17	Annotering	23
5.3.18	Metadata	24
6	Resultat	25
6.1	Summering av svar på frågeställningar	25
6.1.1	Vilket programmeringsspråk bör användas?	25
6.1.2	Vilken samplingsfrekvens bör användas för EEG signalerna.	25
6.1.3	Vad finns det för dataformat som är lämpade för EEG signaler? Vilket bör användas i denna tillämpning?	25
6.1.4	Klarar systemet alla beräkningsbehov?	25
6.2	Mål som uppnåtts	26
6.3	Mål som ej uppnåtts	26
6.4	Modell av lösning på överförning	26
6.5	Användning av Mobilt EEG	27
7	Diskussion	28
7.1	Diskussion av resultat	28
7.2	Genomgång av problem	28
7.2.1	Antaganden	29
7.2.1.1	Biologiska problem	29
7.2.1.2	Kunskap om hjärnan	29
7.2.1.3	Programmeringsspråk inte spelar någon roll	29
7.2.1.4	Pröva olika ADC:er	29
7.2.1.5	Databas	29
7.2.1.6	Externt beräkningsprogram	30
7.2.1.7	Streama Data	30
7.2.2	Oförutsedda problem	30
7.2.2.1	Raspberry Pi ej lämpad	30
7.2.2.2	Python ej lämpad	30
7.2.2.3	Saknad av färdig ADC-lösning	31
7.2.2.4	Extra hårdvara	31
7.2.2.5	Huvudbonad	31
7.3	Miljö och Etik	31
7.4	Övriga tankar	32
7.5	Möjliga Lösningar	32
7.5.1	En enda mikrokontroller	32
7.5.2	Interrupt knapp som startar båda hårdvaran samtidigt	32
7.5.3	Extern lagring	32
7.5.4	Extern buffer	33
7.5.5	DMA (Direct Memory Access)	33
7.5.6	Trådlös överföring	33
7.5.7	Realtids-OS för RPi	33
7.5.8	Realtidstillägget CONFIG_PREEMPT_RT för Linux	33

8	Framtida Utveckling	34
9	Bilagor	35
9.1	PCB-kretsschema	35
	Referenser	37

1

Inledning

1.1 Bakgrund

Nuförtiden används *elektroencephalografi* (EEG) för att, ofta i samband med andra metoder, diagnostisera och karakterisera olika hjärnsjukdomar som, framförallt, epilepsi men också hjärninflammation och koma. Andra diagnostiska metoder som MRI (*Magnetic Resonance Imaging*) och CT (*Computed Tomography Scan*) är i många avseenden överlägsna, men är begränsade rent praktiskt av tyngden och storleken på utrustningen och kan oftast bara utföras vid en fast installation. Eftersom EEG inte nödvändigtvis drabbas av detta problem finns det potential att skapa en mobil implementation.

Ett mobilt EEG av detta slag skulle teoretiskt kunna genomföras under andra omständigheter och på andra platser än på ett sjukhus, vilket eventuellt kan öppna upp outforskad data för framtida forskning inom biofysik och andra relevanta fält. Förhoppningsvis så kommer den mobila aspekten även medföra en minskning i den tid som det tar att applicera elektroderna på en patient.

Exakt hur människan lär sig, hur vi tar in och katalogiserar information, är fortfarande relativt okänt och är ett aktivt forskningsområde. Ett mobilt EEG skulle exempelvis kunna användas för att försöka anpassa studietekniker, utbildning och miljö för optimala förhållanden.

Projektet genomfördes hos *Institutionen för Data- och Informationsteknik* på Chalmers.

1.2 Syfte

Projektet syftar till att besvara frågan hur man på ett lämpligt sätt kan konstruera ett mobilt system för att utföra EEG-test. Systemet skall designas utifrån vissa förutsättningar för att sedan konstrueras till ett fungerande system.

1.3 Mål

Målet med projektet är att utveckla ett EEG-system som är lättanvänt, funktionellt samt mobilt. Man vill att systemet samtidigt skall spela in extern stimuli genom användningen av en monterad kamera och mikrofon, samt logga all data för alla anslutna elektroder. Denna information skall samlas parallellt med varandra och sedan tidsstämplas för varje sampel, för att korrelera hjärnaktivitet med extern stimuli.

Följande delmål gäller initialt för projektet:

- Att ha ett simpelt interaktivt GUI som har funktionalitet för att starta och stoppa inspelning.
- Köras på batteri. Det ska även fungera med sladd.
- Läs in EEG-data från elektroder med känd testsignal för att bekräfta funktionalitet hos ADC och eventuellt kalibrera den.
- Läs in EEG-data från elektroder anslutna till *occipitalloben* på en patient. Occipitalloben ger av sig en stark, tydlig signal som kan användas för att bekräfta funktionalitet av elektroder och eventuell filtrering. [31, s.172]
- Läs in EEG-data från elektroder anslutna till övriga hjärnområden.
- Skriva data på EDF-formatet.

- Fungerande ljud- och bildinspelning samt omkodning till lämplig container¹-format.
- Förena hårdvaran så att det blir en produkt, i form av en mössa för elektroder, samt ett paket för resterande hårdvara som kan placeras någonstans på kroppen.

Med detta kan man ställa vissa frågor som skall besvaras.

- Vilket programmeringsspråk bör användas för mjukvaruutvecklingen?
- Vilken samplingsfrekvens bör användas för EEG signalerna?
- Vad finns det för dataformat som är lämpade för EEG signaler? Vilket bör användas i denna tillämpning?
- Klarar systemet alla beräkningsbehov?

1.4 Avgränsningar

De avgränsningar som finns för detta projekt är:

- Det ingår ej i projektet att tolka EEG-datan, då speciell kunskap samt träning krävs.
- Det ingår ej i projektet att filtrera eller på annat sätt behandla eventuell digital data.

¹En *container* (sv. behållare) är en *meta*filtyp som beskriver hur olika slags data struktureras i en datafil. I det här fallet gäller det ljud- och videodata. Typiska exempel är **.mp4** och **.avi** filer.

2

Teknisk Bakgrund

I detta kapitel redovisas de olika subsystem som projektet består av. Dessa delas upp logiskt i hårdvara och mjukvara.

2.1 Hårdvara

Här presenteras en översikt över de större hårdvarumodulerna som systemet i projektet består av. Detta ska ge en förståelse över vad de är och varför de är lämpade för ändamålet. En fullständig systembeskrivning hittas i kap. 5.2.1.

2.1.1 Raspberry Pi

En Raspberry Pi (RPI) är en *enkortsdator*¹ som kör, bland annat, linuxbaserade operativsystem (OS) och därmed tillåter stor frihet att programmeras till användarens behov. Det används brett av många olika individer samt organisationer, och är utmärkt när det kommer till små eller mobila produkter som kräver beräkningskraft och/eller ett fullfjädrat operativsystem. Typiska användningar för en RPI är som egengjord säkerhetskamera eller som strömsnål webbserver.

I praktiken så kan RPI användas både som en 'vanlig' dator med en grafisk miljö och periferikopplingar eller på ett antal andra sätt. Ett sätt att använda en RPI är som ett strömsnål inbyggt system som kan styra utrustning på låg nivå med inbyggda hårdvarumoduler där kommunikationsprotokoll som SPI och seriell finns inbyggt i processorn eller för att styra en stor uppsättning GPIO (eng. *General-Purpose Input/Output*) pinnar.

RPI har uppdaterats med nya versioner sedan den första lanseringen 2012, och versionen som planerades att användas i detta projekt var Raspberry Pi 3B+, som till skillnad från tidigare versioner har inbyggt WiFi och Bluetooth samt en variant på Cortex-A53 processorn, som implementerar ARMv8-A instruktionsuppsättning, med högre nominell klockfrekvens[26].

2.1.2 Mikrokontroller

En mikrokontroller eller *microcontroller unit* (MCU) är en programmerbar krets med hög flexibilitet som kan användas i många generella sammanhang där det inte förekommer extrema prestandakrav. Uppgifter som kräver extrem prestanda, till exempel grafikprocessering, betjänas oftast bättre ur ett tekniskt och ekonomiskt perspektiv av mer dedikerad hårdvara, som är per definition mindre flexibel.

En stor skillnad mellan en MCU och en mer traditionell dator som RPI:en är att den inte kör något operativsystem eller kernel om inte man programmerar en på egen hand. Koden man skriver kompileras till instruktioner på maskinnivå och på en MCU utan OS kommer instruktionerna att exekveras direkt utan eventuella avbrott från ett OS. Denna närhet mellan mjukvara och hårdvara gör framförallt att man minskar den processortid som används till *overhead*, med andra ord operationer som inte direkt används för den uppgiften man vill utföra. På en MCU är det sällan man önskar arbeta på flera uppgifter parallellt. Ett undantag är flerkärniga MCU processorer där komplexiteten ökar avsevärt. Man undviker därmed overhead på grund av att processorn ska byta mellan olika program på grund av pseudo-parallell exekvering vilket innebär minskad latens² för IO

¹En enkortsdator (eng. *single-board-computer*) skiljer sig från en vanlig dator i med att all hårdvara, även sådant som brukar finnas som insticksmoduler, som primärminne exempelvis, finns på samma kretskort. Detta har naturligtvis fördelen att hårdvaran kan minimeras, vilket gör den mer portabel.

²Latens är den tidsfördröjning mellan en insignal och motsvarande utsignal och är skild från datatakt. Om man, till exempel, önskar en låg svarstid för en tidskänslig uppgift så skall man ha låg latens, oberoende av övrig prestanda.

(eng. Input/Output) operationer. Allt detta gör MCU:er lämpade för system med *realtidskrav*, som hanterar tidskänsliga uppgifter.

En MCU kan användas för att komplettera ett OS-baserat system, som en RPi, med ett låglatens 'realtids-gränssnitt' och kan då ses som en buffer mellan realtids- och icke-realtidssystem. Den betjänar realtidssystemet enligt tidskraven och kan sedan överföra informationen till icke-realtidssystemet när både den och MCU:n har ledig processortid. Detta beskrivs mer utförligt i kap. 5.2.1.

Båda projektmedlemmarna har tidigare erfarenhet av Arduino hårdvaran samt utvecklingsplattformen och därför valdes mikrokontrollern Microchip ATmega 328P[15]. För att göra den elektriskt kompatibel med RPi så matas den med 3.3V systemspänning vilket även gör att klockfrekvensen måste sänkas 8 MHz från nominella 16MHz vid 5V. För detta projekt är det värdefullt att notera att den har inbyggda hårdvarumoduler för SPI, seriell kommunikation och externa interrupts, då det underlättar kommunikering med andra hårdvarumoduler. Ytterligare protokoll som inte har hårdvarustöd kan implementeras i mjukvara genom så kallad 'bit-banging', det vill säga att ett kommunikationsprotokoll kan implementeras helt i mjukvara genom direkt styrning av GPIO pinnar.

2.1.3 Analog till Digital Omvandlare (ADC)

En ADC är en krets avsedd att omvandla analoga signaler till en digital representation för att kunna bearbetas digitalt.

Att mäta EEG signaler är av sin natur relativt svårt att göra eftersom de har en låg amplitud och lätt kan störas ut av brus, vilket ställer höga krav på ADC:n samt resten av den analoga datavägen. För att kunna göra systemet så robust och bruståligt som möjligt samtidigt som man integrerar så mycket av den analoga elektroniken som möjligt valdes en *integrerad krets* (IC) från *Texas Instruments*, ADS1299 [1], som enligt tillverkaren är lämpad för just mätningar inom medicin och i synnerhet EKG och EEG.

De flesta MCU:er har flertalet inbyggda ADC:er, men de är oftast begränsade i sin prestanda jämfört med dedikerade kretsar.

2.1.4 Batteri

En viktig del i systemet är batteriet och dess kapacitet och kraftleveransförmåga. RPi:n beräknades vara den mest strömkrävande delen i systemet. Den använder relativt lite ström men kan, beroende av processorlasten och vilken slags kringutrustning som är ansluten, som kamera och mikrofon, dra mer ström än vad en 'vanlig' powerbank klarar att leverera. För att en RPi inte ska överbelasta strömkällan, så behöver den klara ca. 2A, vilket batteriet skulle behöva kunna tillföra. Batteriet som valts är av Deltaco:s modell, har en kapacitet på 6000mAh, och har en output på 5V och 2A. Spänningsregleringen på RPi kortet står för omvandling till 3.3V.

2.1.5 Kamera

Kameran, som i systemet används för att spela in video, ansluts direkt till RPi:n via ett *CSI* (Camera Serial Interface)³ gränssnitt och är av typen *Camera Module V2*. Fördelen med detta gränssnitt är att man slipper vissa mellanliggande kommunikationsprotokoll innan datan från kameran når processorn, vilket ger en stor prestandavinst.

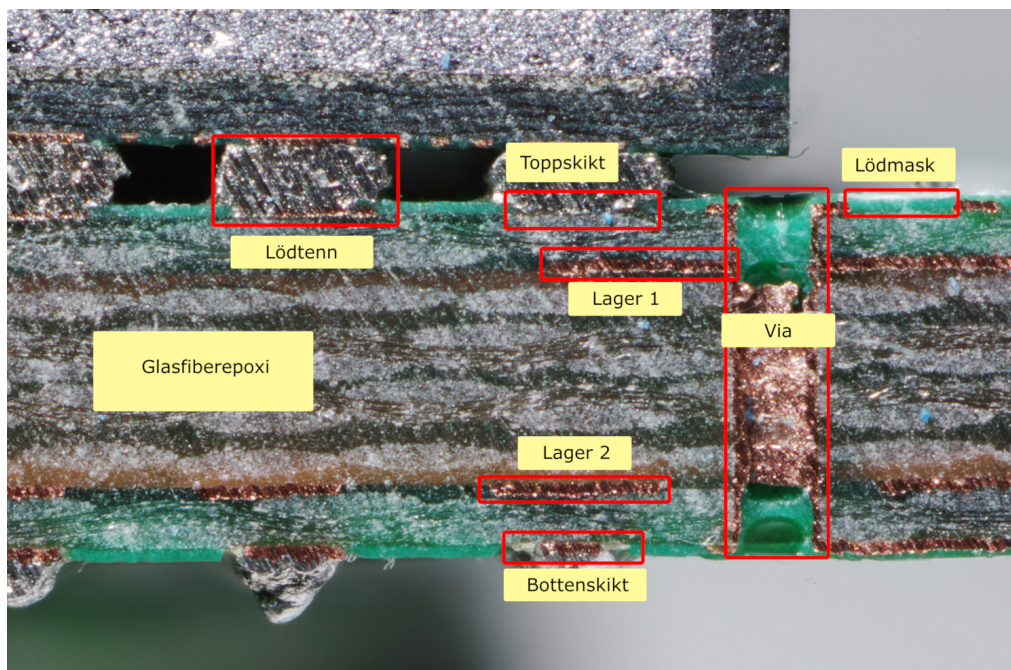
2.1.6 Mikrofon

Alla versioner av RPi saknar analog mikrofonanslutning och därför krävs en ADC krets, också kallad ljudkort i när det gäller en insticksmodul för ljudsignaler, för att omvandla mellan analoga signaler från mikrofonens 3,5mm kontakt till en digital signal som kan överföras via RPi:ns USB port. En enkel headsetmikrofon har använts i projektet då det varit oviktigt vilken typ eller modell som används.

³CSI är en specifikation av MIPI (Mobile Industry Processor Interface) Alliance.

2.1.7 PCB

Printed circuit board eller PCB (*sv. mönsterkort*), är i enklaste fallet ett lager med isolerande material, oftast glasfiberepoxi, och ett tunt skikt med kopparfolie i ett visst mönster som utgör elektriska ledare i en krets. Kopparskikt kan finnas på båda sidorna av ett kretskort och även inne i det isolerande materialet för att få *flerlayerskort*. Syftet med att använda en PCB gentemot andra former av prototypkrets såsom 'breadboards' och 'perfboards' är att det underlättar avsevärt i mera komplexa kretsar där komponenterna är mindre, har fler elektriska kopplingar och har en allmänt tätare integration. Genom att använda *electrical design automation* (EDA) verktyg för PCB design så kan man också genom *design rule checks* (DRC) verifiera kretsen på olika sätt innan den tillverkas.



Figur 2.1: Tvärsnitt av ett 4-lagers PCB med väsentliga områden markerade.

Photo: Rainer Knäpper, Free Art License (<http://artlibre.org/licence/lal/en/>), https://commons.wikimedia.org/wiki/File:Bga_und_via_IMG4531_wp.jpg

I fig. 2.1 ser man en typisk flerlayers PCB. Den har ett topp- och bottenskikt och två inre lager som inte syns utifrån. Olika lager kan kopplas genom att använda en *via*, ett kopparplatterad borrhål som utgör en lodrät ledare. I figuren ser man hur lager 1 och bottenskiktet kopplas ihop. En via som enbart kopplar inre lager kallas för en 'buried' (sv. begravd) via då den inte kan komma åt utifrån. Dessa brukar undvikas då de kan försvåra felsökning och dessutom inte kan åtgärdas ifall ett tillverkningsfel skulle uppstå.

2.2 Mjukvara

Detta kapitel går igenom de olika program mjukvaruverktyg som använts i projektet.

2.2.1 Python

Python är ett objektorienterat, interpreterad⁴ programmeringsspråk som är lättlärt och högt modulärt i sin dynamiska funktionalitet. Det finns många bibliotek i diverse tillämpningar. På grund av dess struktur i hur det är formerat, med dynamisk typsättning och det faktum att man inte behöver kompilera källkoden, gör att det är attraktivt som bland annat skriptspråk. På grund av de olika biblioteken som stöds, så är Python oerhört adaptivt i dess användning, vilket var en av anledning till att det valdes som språk för projektet. Tanken var att oavsett sätt som valdes att implementera lösningar, så skulle det vara relativt simpelt med ett språk som Python. [2]

2.2.2 Tk Interface (TkInter)

TkInter är ett Python API för Tk GUI toolkitet. Det är en defakto standard för pythonbaserade GUI-applikationer och är det mest använda av sitt slag. Det är använt i programmet för att utveckla ett GUI för lättillgänglig användarvänlighet i programmets styrning. [3]

2.2.3 European Data Format (EDF)

EDF är ett relativt enkelt och flexibelt format för hantering av multikanaliga bioelektriska signaler. Det skapades för att kunna applicera sina sömnanalysalgoritmer på sin respektive data och så att resultaten av dessa sedan kunde jämföras med varandra. Sedan 1992 så har EDF varit standard för bland annat EEG data, samt Polysomnografi (PSG) data, men kan även användas för att lagra annan bioelektrisk data. Sedan den introducerades har formatspecifikationen uppdaterats till en ny version, EDF+, och har numera stöd för funktioner som anteckningar med inbyggda tidsstämplar och möjligheten att kunna pausa inspelning av data. [4] [20]

2.2.4 Linuxprogram

Ett flertal program som installerats under RPi:ns linuxmiljö har använts, till exempel, `raspivid` och `arecord` för att spela in video, respektive audio och `OMXPlayer` för att spela upp video. Dessa program anropas av huvudprogrammet, som är skriven i Python, vilket gör att programmet för tillfället är linuxbundet. Pythonkoden kan relativt lätt anpassas till att använda vad som finns till hands i den aktiva miljön i framtida utveckling.

En viktig del med `OMXPlayer` är att den är hårdvaruaccelererad, vilket innebär att den kan jobba emot dedikerad avkodningshårdvara och är ett krav om video skall kunna spelas upp i korrekt hastighet. Alternativet är att RPi:n avkodar videoströmmen med mjukvarubibliotek som ger mycket mer 'overhead' samtidigt som RPi:n saknar processorkraften att utföra sådana komplexa beräkningar i realtid[23].

2.2.5 Python EDF Library (PyEDFlib)

PyEDFlib är ett pythonbaserat EDF bibliotek som innehåller en mängd funktioner för att lätt kunna skriva Pythonbaserade applikationer som använder sig av EDF för att hantera signaler. Detta bibliotek är en inkapsling av det ursprungliga biblioteket EDFlib, skriven i C[5].

⁴Ett interpreterat programspråk behöver inte kompileras utan 'tolkas' under själva körningen.

3

Metod

Arbetet har utförts i en dedikerad labbsal på Chalmers, vilket gjort det lätt för handledare att mötas upp och diskutera projektets fortsatta utveckling.

I början av projektet så lades stor tid på att försöka planera, läsa på och förstå ämnet. Det lades även tid på att lära sig programspråket Python, något som ingen av projektets medlemmar var särskilt bekant med sedan tidigare. Till en mindre grad lades även tid på att lära sig CAD-programmet KiCad, för att kunna skapa ett mönsterkortprototyp.

Efter att ha fått en viss bekantskap med ämnet, startade arbetet en process av idéer och beslut gällande olika aspekter av den färdiga produkten. Beslut fattades om saker som vilken hårdvara och mjukvara som skulle användas, hur GUI:t skulle se ut och vilka funktionaliteter som programmet skulle ha och i vilken ordning de skulle implementeras.

3.1 Agila metoder

Agila metoder är ett speciellt arbetssätt vilket främjar utveckling och förbättring i olika ändamål. Detta tillämpas då det sker regelbundna möten med både medlemmar i arbetsgruppen, samt produktägare, för att få en kontinuerlig ström av feedback och information som används för att den tidigare nämnda utvecklingen ska fortskrida på ett bra sätt och inte stagnera.

Dessa regelbundna mötet har skett med handledare, för att diskutera utvecklingen just då, samt bestämma ett lämpligt nästa steg. I många fall så har dessa tillfällen använts för att diskutera de aktuella problemen, då det var främst de som stod i vägen för en fortskrida utveckling.

3.2 Git och GitHub

Projektets versionshantering har skett med programmet `git` med GitHub som hostingplattform¹. Att versionshantera innebär att man har en metodik kring hur man håller reda på förändringar i ett system eller i kod. Detta utförs vanligen genom att man har en huvudstam med den aktuella koden, och utvecklingsgrenar. Dessa grenar skapas från stammen när man vill utveckla nya funktionaliteter, och när den specifika funktionen anses klar så sammanfogar man den grenen med den fungerande koden. På detta sätt så har man en centralt fungerande kod, samtidigt som flera olika utvecklare kan sitta i separata utvecklingsgrenar, beroende på mål eller funktion. Det kan också användas som backup om något skulle gå snett vid en integrering av en gren, då man lätt kan backa tillbaka till den versionen av koden som fungerade innan man försökte sammanfoga grenen med stammen, eller ännu längre bak i utvecklingen om man så vill det.

Där finns *Master* branchen, samt de olika utvecklingsfokuserade brancher som använts under utveckling. Med hjälp av Github så kan man se hur utvecklingen gått framåt, vilka delar som utvecklats när och i vilken ordning. Det finns en fullständig versionshantering av projektet, vars versioner uppdaterats när något mindre mål uppnåts. Vid standardinställningar är kommentarer obligatoriska vid varje uppdatering av koden vilket uppmuntrar till tydlighet.

¹GitHub är, enkelt sammanfattat, en central server som distribuerade användare jobbar emot genom Git.

3.3 Kunskapsinhämtning

När det kommer till kunskap om EEG och hjärnan generellt så finns det ett enormt utbud av information, men samtidigt inte så många svar som man skulle vilja ha. Det finns fortfarande mycket som är okänt när det kommer till hjärnan, och att kunna veta hur den därmed fungerar kan vara en svår sak. Däremot så finns det relativt väldokumenterat om de joniska strömmarna i hjärnan, vilket är vad EEG använder sig av. Det finns konstant aktivitet i hjärnan som genererar joniska strömmar, därmed så kan det vara svårt att veta exakt vilka strömmar som speglade vad en individ såg eller upplevde under någon tidpunkt.

För att förstå de bioelektriska mekanismer i kroppen som ger upphov till elektriska signaler och hur man på bästa sätt mäter dem, användes en del litteratur om bioelektriska signaler och dess klassificeringar, samt bioelektriska mätsystem med beskrivningar av verkliga implementationer. I synnerhet har boken *Medical Instrumentation: application and design* av J. G. Webster och J. W. Clark[31] varit till stor hjälp under arbetet för att förstå ämnet.

För att förstå olika tekniska koncept som används i ADC kretsar, som $\Sigma\Delta$ -modulering, har man använt tidigare kursböcker, speciellt *Elektriska Mätsystem och Mätmetoder* av Lars Bengtsson[16], samt så kallade *Application Notes*, beskrivande texter ämnade åt ingenjörer som ofta innehåller implementationsdetaljer utöver den abstrakta konceptbeskrivningen.

Information har även samlats in från olika vetenskapliga och tekniska journaler när det gäller specifika frågor eller koncept.

Vid programmering av Raspberry Pi har man utnyttjat det officiella forumet[25] i många fall för att besvara praktiska frågor och vid programmering av mer maskinorienterade enheter som mikrokontroller har man använt sig av relevanta datablad samt även där utnyttjat relevanta forum såsom Arduino forumet[14] och Texas Instruments 'data converter' forum[28].

4

Teoretisk Bakgrund

Detta avsnitt beskriver den underliggande teorin för hur EEG fungerar och hur det utförs i praktiken.

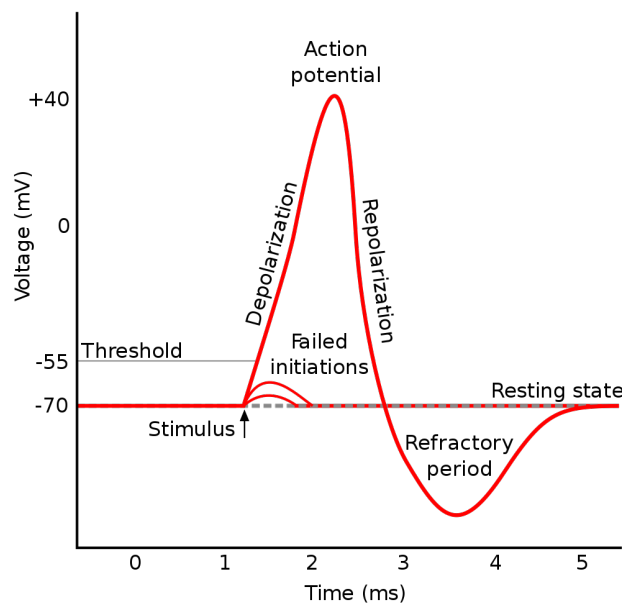
4.1 Elektroencefalografi

Elektroencefalografi (EEG) används för att detektera och mäta jonströmmar i hjärnans nervceller, särskilt så kallade *neurala oscillationer*, genom att fästa elektroder vid skalpen, vanligtvis med en ledande elektrolytgel och eventuellt någon slags huvudbonad som fysiskt håller dem på plats. Vågformerna kan sedan användas för att göra bedömningar kring en individs hjärna och dess tillstånd, så som att fastställa nivå på koma, epilepsi eller om en patient är hjärndöd. Mätning sker ofta med en sampelfrekvens på mellan 200-500 Hz, men man kan mäta upp till flera kHz, beroende på kapabilitet av systemet och önskad frekvensupplösning.

4.2 Biopotentialer

När man talar om olika bioelektriska signaler i kroppen och mätningarna därav så brukar man förknippa dem med olika organ: elektroencefalografi (EEG) syftar på mätningar av biopotentialer i hjärnan medans elektrokardiografi (EKG) gäller mätningar i hjärtat. Övriga mätningar som ofta är av intresse är elektroneurografi (ENG) direkt i nerver, elektromyografi (EMG) i muskler och elektroretinogram (ERG) i näthinnan.[31]

Bioelektriska potentialer, eller biopotentialer, skapas av en elektrokemisk process som förekommer hos en viss sorts celler, huvudsakligen hos nervceller. Dessa celler är i ett 'vilotillstånd' tills de blir tillräckligt stimulerade och genererar en 'aktionpotential'. En sådan cell har en elektrisk potential mellan den intracellulära och extracellulära miljön hos cellen. Detta kallas för *membranpotentialen* och är i vilotillstånd ca -70mV.



Figur 4.1: Spänningstransienter i en nervcell. Cellmembranet övergår från 'vilotillståndet' -70mV, till en aktionpotential när den tar emot stimuli som når ett visst tröskelvärde. Cellmembranet depolariseras, och depolariseringen propagerar längs nerven. Efter en kort tid återgår membranpotentialen till vilotillståndet.

Fig. 4.1 visar att när en nervcell blir stimulerad med en potential som överskrider ett visst tröskelvärde så orsakas en *aktionpotential*, en transient i membranpotentialen, även kallad *depolarisering* på grund av att membranpotentialen byter riktning, som utbreder sig längs membranet.

Enstaka nervceller är givetvis svåra att mäta och ointressanta sett till hela organet. Det är när många nervcellers aktionpotentialer inträffar samtidigt som man kan få mätbara resultat med EEG. På hjärnans ytmembran kan man mäta upp till 10mV; utanför kraniet ligger dock magnituden oftast inom spannet 10-100 μ V

4.3 Biopotentiala Elektroder

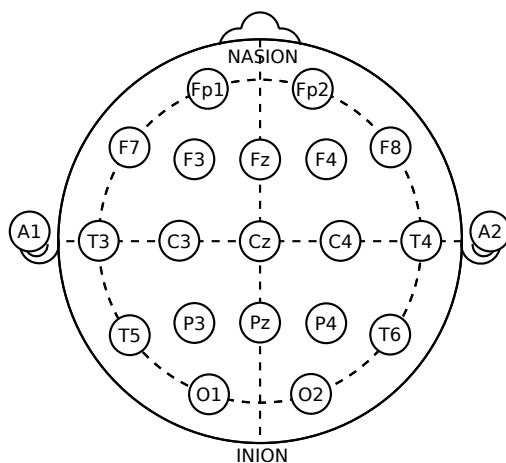
Elektroder i biotekniska sammanhang används för att omvandla *joniska* strömmar orsakade av aktionpotentialer i vävnad till elektrisk ström, med elektroner som laddningsbärare, som används i elektroniken som ska mäta den. En elektrod är med andra ord en *transducer*, som omvandlar mellan två olika former av energi.

Ström passerar gränssnittet mellan elektrolyt, huden och eventuellt elektrolytgel för att minska impedansen, och elektrod genom en kemisk reaktion. Atomer i elektroden oxiderar och skapar fria elektroner och positiva joner (cationer) som förflyttas till elektrolyten. Elektrolytens negativa joner (anioner) förflyttas samtidigt mot elektroden och förser den med elektroner.

En viktig sidoeffekt av den kemiska reaktionen är att en spänning skapas över elektrolyt-elektrod gränssnittet, som kallas *half-cell potential*. Detta uppenbarar sig som en DC spänning i EEG resultaten. Av denna anledning används ofta elektroder av materialet *silver/silver-klorid* (Ag/AgCl) som har en låg half-cell potential på ca. 220mV.[21] Det är viktigt att minimera denna DC spänning då den påverkar den användbara spänningsomfånget i ADC:n som mäter signalerna.

4.4 Kliniskt EEG

För att skapa reproducerbara och jämförbara EEG resultat så måste man ha ett enhetligt system för hur man placerar elektroder, i förhållande till huvudets orientering och i förhållande till andra elektroder. Den dominerande kliniska standarden för elektrodplacering är 10-20 systemet, som visas i fig. 4.2, som använder vissa anatomiska referenspunkter, nasion (vid näsan) och inion (vid nacken), för att placera ut elektroder och kompensera för olika huvudstorlekar. Namnet syftar på att mellanrummet mellan intilliggande elektroder är antingen 10% eller 20% av det totala avståndet från nasion till inion.[31, p. 175] Ju högre upplösning som önskas, desto kortare procentuellt avstånd mellan elektroder behöver man ha. Det har getts förslag på att utöka standardupplösningen till 5% och 10%, i praktiken ett '5-10 system'. [24]



Figur 4.2: Illustration på hur elektroder placeras på huvudet vid användning av 10-20 systemet samt namnkonventionerna för individuella elektroder. Sedd ovanifrån.

När elektroder väl är utplacerade så finns det frihet i hur man använder dem för att utföra mätningar, till exempel, om man vill mäta skillnaden mellan $Fp1$ och $Fp2$ eller mäta båda två med avseende på $A1$ (se fig. 4.2 för elektrodpositioner). Den valda definitionen kallas för en *montage* inom 10-20 systemet. I en *bipolär* montage så mäts skillnaden mellan två intilliggande elektroder, i en *referentiell* montage så mäts skillnaden mellan en elektrod och en referenselektrod, vanligtvis på örsnibben. Det finns även mer avancerade montager som mäter med avseende på den genomsnittliga spänningen av alla intilliggande elektroder.

För att få så låg impedans mellan huden och elektroden som möjligt så bör man förbehandla huden för att få bort torrt eller gammalt hud. Högre impedans ger en större DC spänning i mätningarna. Med bra hudkontakt så brukar ligga impedansen på ca. $5\text{ k}\Omega$ men kan i vissa fall ligga så högt som $500\text{ k}\Omega$.

5

Genomförande

5.1 Förberedelse

Början av projektet bestod till stora delar av påläsning om ämnet. Ämnet inkluderade delar så som vad EEG var, hur strömmarna i hjärnan agerar och existerar, samt hur man kan använda sig av denna information för medicinska ändamål. Det var under denna period som det upptäcktes att det finns en mängd specifika tillvägagångssätt när det kommer till mätning av hjärnaktivitet.

Bland annat om vilken spänningsnivå det handlar om, på vilka ställen på ett huvud man ska göra mätningar, samt vilka svårigheter som finns när det kommer till att mäta otroligt små spänningar på ett så känsligt organ som hjärnan. En relevant del av dessa svårigheter gäller rörelse. Det är nämligen så att det är svårt att uppfatta korrekt data från EEG om muskler rör på sig, då den spänning som används för att manipulera musklerna genererar större spänningsnivåer än vad hjärnaktivitet gör, vilket orsakar stora störningar.

En annan faktor är att det handlar om just små spänningsnivåer, som fyller väldigt lite av det dynamiska omfånget (eng. *dynamic range*) av en typisk ADC och kan därmed inte omvandlas till en meningsfull digital representation utan att först förstärkas. Helst ska denna ADC även inkludera Delta-Sigma modulation för att motverka brus.

5.1.1 Allmänt arbetssätt

Arbete har främst skett genom samarbete på de flesta fronter, förutom vissa data- eller elektrospecifika delar av projektet, till exempel PCB-design, vilket hade kunskapskrav som fanns relativt strikt inom elektroteknik.

Relativt sent under projektets gång var man tvungen att byta RPi på grund av en kortslutning. All mjukvara och eventuella hårdvaruinställningar som fanns på sekundärminnet, ett microSD kort, kunde flyttas till en RPi av tidigare modell, en Raspberry Pi 2 Modell B Version 1.2. Man var dock tvungen att använda en trådad internetuppkoppling. Utvecklingsarbetet kunde med andra ord försätta obehindrat efter detta.

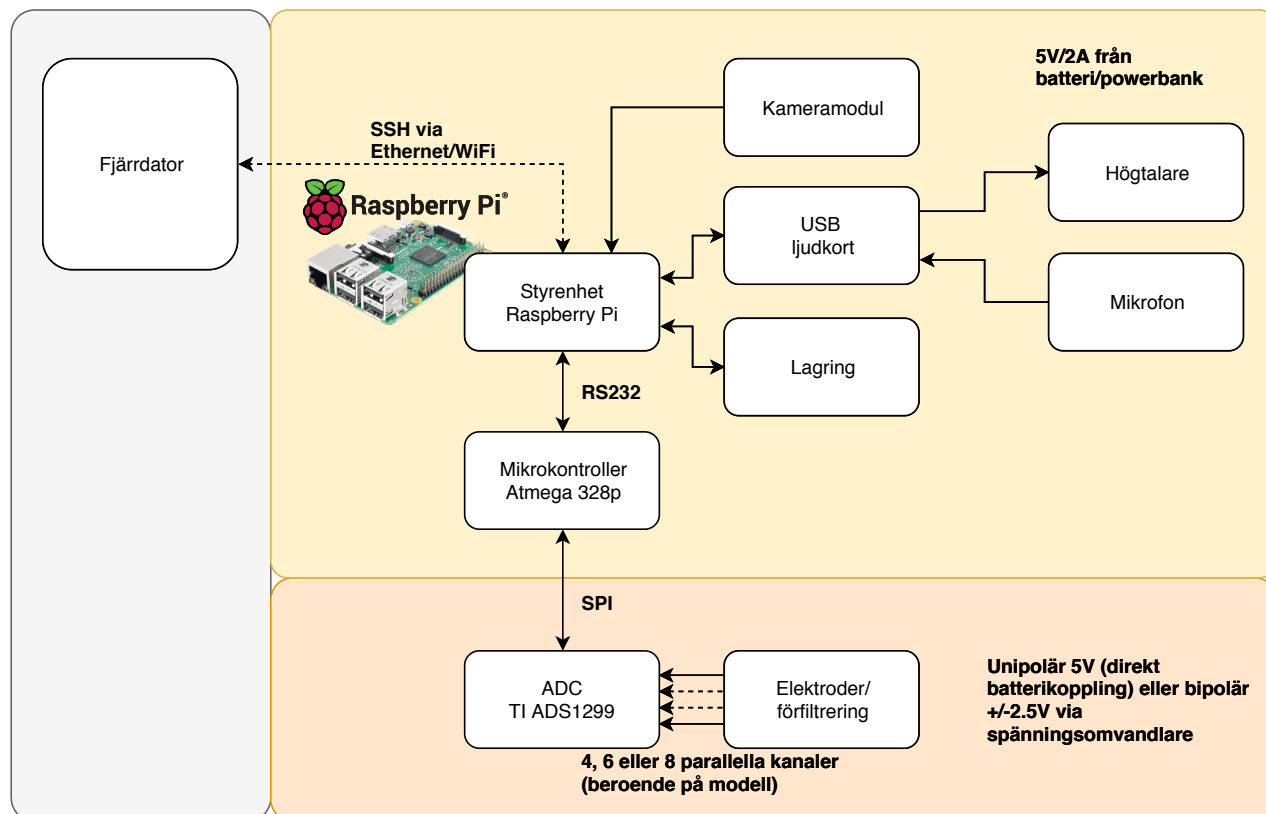
Under projektets gång så har det skett periodiska möten med projekthandledaren för att förbättra och strukturera projektets utveckling. Förutom de regelbundna mötena så har det även funnits kontakt vid akuta problem. Ett sådant var exempelvis när den originella RPi:n kortslöts och en ny behövdes akut.

Projektet består till stor del av egenutvecklad mjukvara, och till det var det lämpligt att använda Git och GitHub för versionshantering. Detta tillät en iterativ designmetod där man kunde implementera en viss funktionalitet, testa den och sedan sammanfoga den nya koden med kodbasen på Github. Man kunde också återställa koden till en tidigare version ifall något fel skulle upptäckas vid ett senare tillfälle.

5.2 Systemdesign

5.2.1 Systembeskrivning

I fig. 5.1 kan man se en övergripande bild av testsystemet som planerades. Man har en central styrenhet, i detta fall en Raspberry Pi, som styr flera periferienheter och som kan själv styras utifrån, genom ett trådlöst gränssnitt så som Bluetooth eller WiFi. Det är viktigt att skilja mellan olika spänningar i systemet då viss kringutrustning, speciellt analog elektronik som ADCs, kan behöva spänningskällor som inte finns i resten av systemet.



Figur 5.1: Systembeskrivning. Pilarna beskriver dataflödet i systemet. Systemet är uppdelat med avseende på systemspänning, då en ADC kan kräva en bipolär spänningskälla för att även kunna mäta negativa spänningar.

Den grundläggande funktionaliteten som beskrivs i diagrammet är att ADC:n samplar in upp till 8 EEG-signaler samtidigt och annonserar för MCU:n att data finns att hämta, varvid MCU:n läser in datan via SPI-protokollet (Serial Peripheral Interface) och lagrar den. Innan nästa sampel skickar MCU:n datan vidare till RPi:n via en seriell länk (RS232). När data väl tas emot av RPi:n skall den använda PyEDFlib för att skriva datan löpande till en fil i EDF+ format.

Samtidigt som EEG data läses in så skall video och ljud spelas in vid RPi:ns kameramodul och USB-ljudkort med ansluten mikrofon, respektive.

Alla submoduler ska kunna styras från ett GUI gränssnitt på RPi:n.

Enligt 10-20 systemet (se fig. 4.2) så ingår 19 mätpunkter i en standard montage (öronloberna, A1 och A2, används enbart som en gemensam referenspunkt). Systemet som beskrivs i denna rapport är dock begränsad till 8 kanaler av den valda ADC:n. Varje kanal har en positiv och negativ input vilket innebär att 16 elektroder kan användas. Eftersom hjärnvågor är relativt lokaliserade så behövs dock inte en full uppsättning elektroder för att erhålla meningsfull data, givet att man placerar ut dem där man önskar mäta.

Ett fortsatt arbete skulle vara att utöka antalet kanaler för att täcka hela 10-20 specifikationen. Detta beskrivs

mer utförligt i kap. 8.

5.2.2 Datakommunikation inom systemet

Mycket av designarbetet i projektet lades på att försöka lösa hur data skulle överföras mellan olika delar av systemet.

Det finns huvudsakligen två olika sätt att överföra digital information: parallellt och seriellt. Parallell överföring innebär att man använder flera kanaler för att skicka olika delar av meddelandet samtidigt, seriell överföring innebär att man använder en enda kanal för att skicka hela meddelandet, bit för bit. Det är uppenbart att man kan öka datatakten med parallell överföring, dock på bekostnad av hårdvaruresurser, men i detta projekt är datatakterna såpass låga (54 kbaud, enligt nedan) att det alternativet valdes bort tidigt som onödigt.

5.2.2.1 SPI gränssnitt på ADC ADS1299

ADS1299 använder sig av 4-wire SPI, med en klocksignal (SCLK), en chip-select signal (\overline{CS}) och två envägs datasignaler (DIN och DOUT). ADS1299 är en slavenhet i SPI-sammanhang och kräver därmed att masterenheten (MCU:n Atmega 328p) utför läsningen och eventuella skrivningar. Den kan med andra ord inte 'skicka' den inlästa datan till masterenheten utan måste aktivt läsas genom att skicka en pulståg på SCLK signalen.

SPI-protokollet som ADS1299 använder har dock kompletterats med en extra flödeskontrollsignal, DRDY (Data Ready), för att annonsera för *master*-enheten att data finns tillgängligt att läsa. ADS1299 har en data buffer i form av ett skift-register där datan lagras efter varje inläsning. Detta måste läsas innan nästa inläsning, annars skrivs den över och kan ej återskapas. Eftersom detta inte får ske, ställer det *hårda*¹ tidskrav på systemet.[29, kap. 1.1]

När ADS1299 annonserar sin data måste alltså masterenheten läsa av hela buffern seriellt inom en viss tidsram som bestäms av sampeltakten. I nominella fallet är datatakten 250 Hz. Detta innebär att hela läsningen måste utföras inom 4ms från den tidpunkt att datan blev tillgänglig.

Den seriella läsningen av varje sampel har också en datatakt som bestäms av enheten som utför läsningen. Detta kan kallas för bithastigheten, eller *baudrate* på engelska, som innebär hur många informationssymboler som kommuniceras per tidsenhet.

ADS1299 har 8 inläsningskanaler med 24 bitars upplösning på varje kanal. Datan från kanalerna skickas tillsammans med ett 'status' meddelande, även den 24 bitar lång. Tillsammans innebär det att varje sampel som skickas innehåller 216 bitar (27 bytes), och om detta ska skickas 250 gånger i sekunden motsvarar detta en 'throughput' 54 kbaud.

Här bör det påminnas om att SPI överföringen styrs enbart av *master*-enheten, i det här fallet MCU:n. MCU:n som användes, AtMega328p har en klockfrekvens på 8 MHz vid 3.3V, och har därför en högsta möjliga SPI-klocka på $f_{osc}/2 = 4$ MHz, enligt databladet. Då tar överföringen av en sampel minst

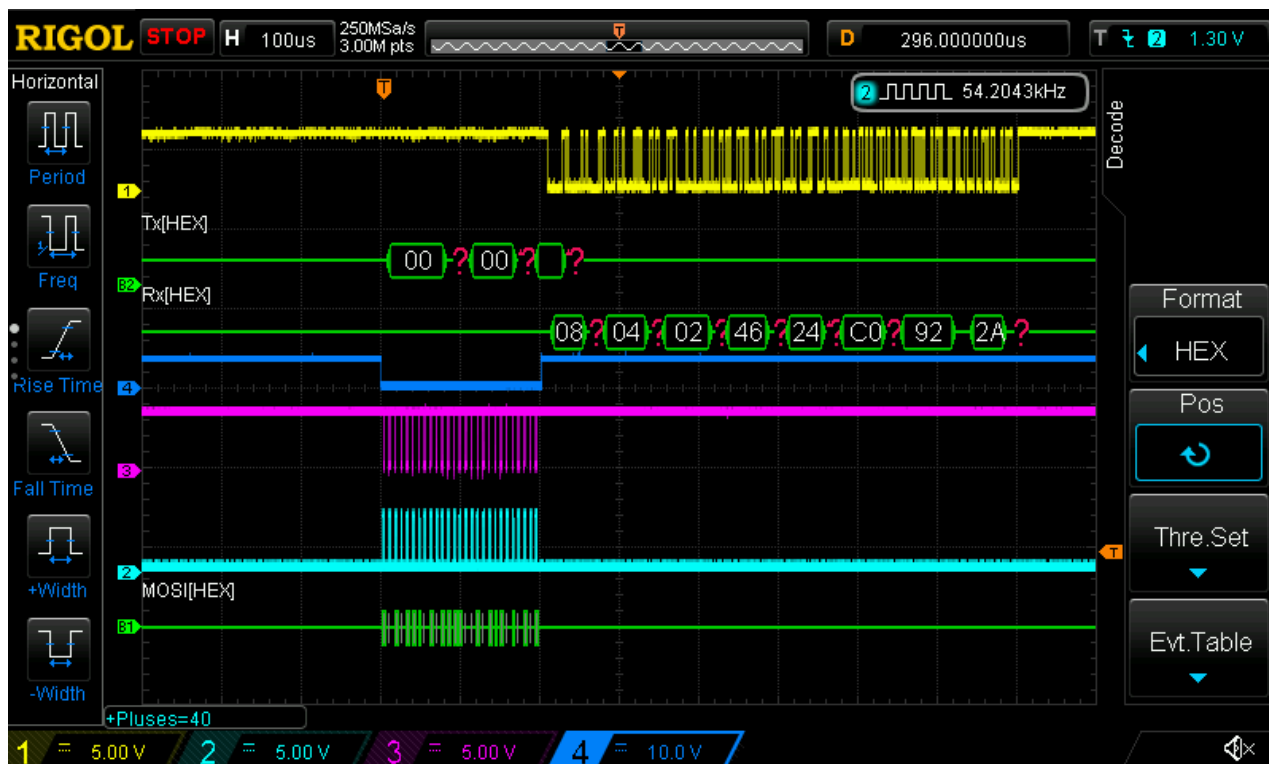
$$T_{osc} \cdot 27\text{bytes} = 108\mu s$$

där de första 3 bytes är statusmeddelandet och resterande 24 bytes är data. Statusmeddelandet innehåller information såsom tillståndet hos den inbyggda "lead-off" detektorn², och kan eventuellt förkastas om den inte används. Eftersom den skickas innan datan så måste den ovillkorligen läsas.

MCU SPI hårdvaran måste dock buffra och skicka en byte i taget, vilket ger en förhållandevis hög overhead. I praktiken har man, genom ett praktiskt experiment med testdata som ska motsvara det som ADC:n skickar i nominell datatakt, och som visas i fig. 5.2, uppmätt en överföringstid på ungefär 200 μs . Detta motsvarar ungefär 1/20 av tidsbudgeten per sampel ($T_s = 250^{-1}\text{Hz} = 4\text{ms}$) vilket ger god marginal för att sedan skicka datan vidare till RPi:n via seriellänken.

¹I facklig litteratur ang. realtidssystem skiljer man mellan tidskrav som är *hårda* och *mjuka*. I system med hårda tidskrav så har en viss data bara värde inom en viss tid, och blir värdelös om *deadline* överskrids. Med mjuka tidskrav så tappar datan värde enligt någon tidsfunktion. I detta fall, om *deadline* överskrids så samplar man fel data, därav hårda tidskrav.

²*Lead-off detection* är en mekanism för att övervaka den elektriska impedansen hos en elektrod. Om impedansen är för hög så försämrats signalkvaliten och beror oftast på den fysiska kopplingen mellan elektroden och huden. Impedansen kan av olika anledningar öka över tid vilket fordrar en periodisk läsning av tillståndet[18].



Figur 5.2: Vågformerna 2, 3 och 4 visar SPI signalerna SCLK, data och \overline{CS} , respektive. Vågform 1 visar samma data överfört via seriellänken. Övrig data i bilden, B1 och B2, är tolkning av datan i verifieringssyfte och kan bortses från.

5.2.3 ADC

På flera sätt den viktigaste komponenten i systemet är analog-till-digital omvandlaren ADC:n. Det är komponenten som ska läsa in information från den yttre miljön, i detta fall elektroder på huvudet, och omvandla den till en digital representation som kan behandlas av resten av systemet.

Det finns många olika sätt att implementera en ADC och de flesta metoder har både för- och nackdelar. Utmaningen blir då att välja den implementationen som har egenskaper som gynnar den specifika tillämpningen. Speciellt när man ska mäta känsliga, i den bemärkelsen att signalen lätt kan störas av brus, signaler, så är det mycket viktigt att välja en lösning som noggrant kan mäta signalen utan att själv införa för mycket brus.

Tidigt så bestämdes det att man skulle använda en lösning som innefattar lågpas filter, analog *instrumentförstärkare* och en *sigma-delta* ($\Sigma\Delta$) ADC. Resonemanget är att LP-filtret sällar bort all information utanför frekvensbandet av intresse vid ca. 0-120Hz. Signalen som blir kvar för-förstärks sedan till en nivå som enklare kan mätas av en ADC. Ofta så har en ADC en fix spänningssving; ju mer av spannet signalen utnyttjar, desto noggrannare blir mätningen, sett till signal-to-noise ratio (SNR).

Det ADC som valdes till systemet var Texas Instruments (TI) ADS1299. Den har enligt fig. 5.3 upp till 8 parallella inläsningskanaler, med varsin inställningsbar PGA, programmerbar förstärkare, (eng. programmable gain amplifier) och $\Sigma\Delta$ -omvandlare. Varje kanal har 24-bitars upplösning. Varje sampel läser alltså in upp till 8 oberoende signaler med 24 bitars upplösning. ADC:n skickar dessutom ett extra dataord på 24 bitar för varje sampel som innehåller status på olika subsystem som exempelvis 'lead-off detection'. Detta innebär en 'throughput' (sv. genomströmning, alltså data som strömmas genom systemet per tidsenhet) på

$$9 \cdot 24b = 216b$$

per sampel, där B är byte³. Vid en nominell sampelhastighet på 250Hz, så innebär detta en throughput på

$$54\text{kbaud} = 6.75\text{kB}$$

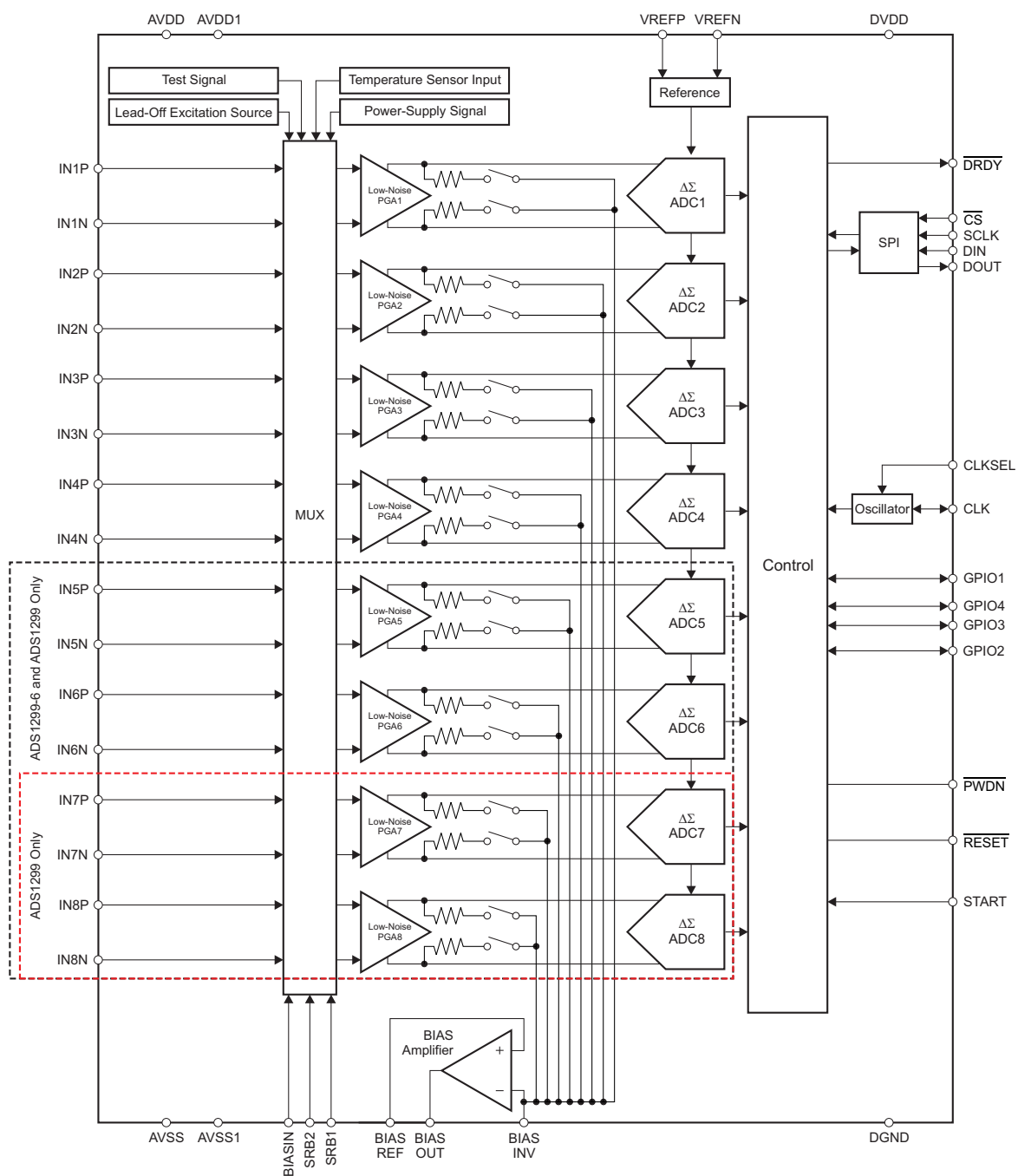
$\Sigma\Delta$ använder *oversampling*, alltså att ADC:n samplar i en betydligt högre frekvens än Nyquistfrekvensen⁴. Fördelen med oversampling är att det gör att inherenta brusällor, som *kvantiseringsbrus*, kraftigt minskas.

Kvantiseringsbrus är den momentana skillnaden mellan den riktiga analoga signalen och dess digitala representation i ADC:n. Om Δ är skillnaden mellan två digitala 'nivåer' så är det maximala kvantiseringsfelet alltså $\frac{\Delta}{2}$. Brus som uppstår på det här viset kan anses vara 'vitt', alltså att det lägger sig uniformt på över alla frekvenser inom frekvensbandet av intresse. Det som $\Delta\Sigma$ metoden ger möjlighet till är *noise shaping*, att skifta bruskomponenter från lägre frekvenser till högre frekvenser, utanför signalbandet, där de sedan kan filtreras bort med ett digitalt filter, eftersom signalen är samplad vid det här laget.[22, kap. 6]

³En byte rymmer som bekant 8 bitar. Det är dock viktigt att skilja på bit och byte när man sedan använder prefixer. En kB eller kilobyte är alltså 8 kb eller kilobit.

⁴Nyquistfrekvensen är enligt Nyquist-Shannon teoremet den samplingsfrekvens som krävs för att kunna återskapa signalen, vilket gör att ingen signalinformation går förlorad. Nyquistfrekvensen är två gånger högre än den högsta frekvenskomponenten i signalen.

$$f_N > 2 \cdot f_{s_{\max}}$$



Figur 5.3: Blockdiagram över I/O på ADS1299. Från vänster till höger: differentiella inputkanaler, multiplexer med valbara bias- och testsignaler, programmerbara förstärkare, biaseringssignalmultiplexer, $\Sigma\Delta$ -omvandlare och SPI gränssnitt. Hämtad från datablad[1].

5.2.4 PCB Design

I början av arbetet var förutsättningen att all hårdvara skulle byggas på *breadboards*/prototypkort med diskreta, hålmonterade komponenter. Detta visade sig dock inte vara möjligt då komplexiteten och tätheten hos elektroniken, i synnerhet ADC:n och dess många kringkomponenter, skulle visa sig kräva en mer sofistikerad lösning. En annan avgörande faktor var signalintegritet. För ett system som mäter små, bruskänsliga signaler så vill man försöka minimera effekterna av induktiv och kapacitiv koppling så mycket som möjligt. Detta görs oftast genom att orientera komponenter och ledningar rätt och minska arean och omkretsen hos ledare. Detta kan styras mycket mer noggrant på en PCB än på andra sorters prototypkort.

I detta projekt var man begränsad både av tid men också kostnad. För att tillverkningskostnaden för en PCB ska hållas inom rimliga gränser så måste huvudparametrarna så som storlek, antal lager och strömkapacitet i ledare inskränkas. Det beslutades därför innan designarbetet påbörjades att PCB:n skulle vara dubbelsidig, alltså ha två lager. Detta är vanligen det minsta antalet som behövs för att icke-triviala kretsar ska kunna kopplas utan att behöva *jumpers*, vilket är korta ledare som monteras på i efterhand och som 'hoppas över' andra ledare. Jumpers försvårar ofta designarbetet och kan möjligtvis sänka signalintegriteten.

5.2.4.1 EDA verktyg och Prototyp-PCB

De första försöken till en PCB design gjordes med ett browser-baserat program vid namnet *EasyEDA*[6] som ägs av företaget *Shenzhen JLC Electronics* som även driver PCB-tillverkningsföretaget *JLCPCB*[7] och elektronikkomponentåterförsäljaren *LCSC*[8]. Detta gör att EasyEDA har kunnat integrera ett komponentbibliotek över komponenter från LCSC, samt automatiskt skapa en *bill of materials* (BOM) och en direktbeställning genom JLCPCB. Det förstnämnda är speciellt intressant då ett bibliotek över verifierade komponent 'footprints', koptarmönster som stämmer överens med ledarna på komponenten, kan avsevärt minska designtiden. Alternativet är att manuellt försöka hitta dessa 'footprints' hos komponenttillverkaren eller rita/infoga dem direkt från komponentdatabladets paketbeskrivning.

Tyvärr fanns det begränsningar i EasyEDA som gjorde det svårt eller omöjligt att växla mellan olika rutnät för olika komponenter, skapa 'keep-out' områden, där alla form av ledare exkluderas, och skapa icke-typiska hålformer genom PCB:n. Den sistnämnda funktionen behövdes för att kunna leda kameramodulens flatkabel igenom kortet enligt designen för Raspberry Pi HATs [9].

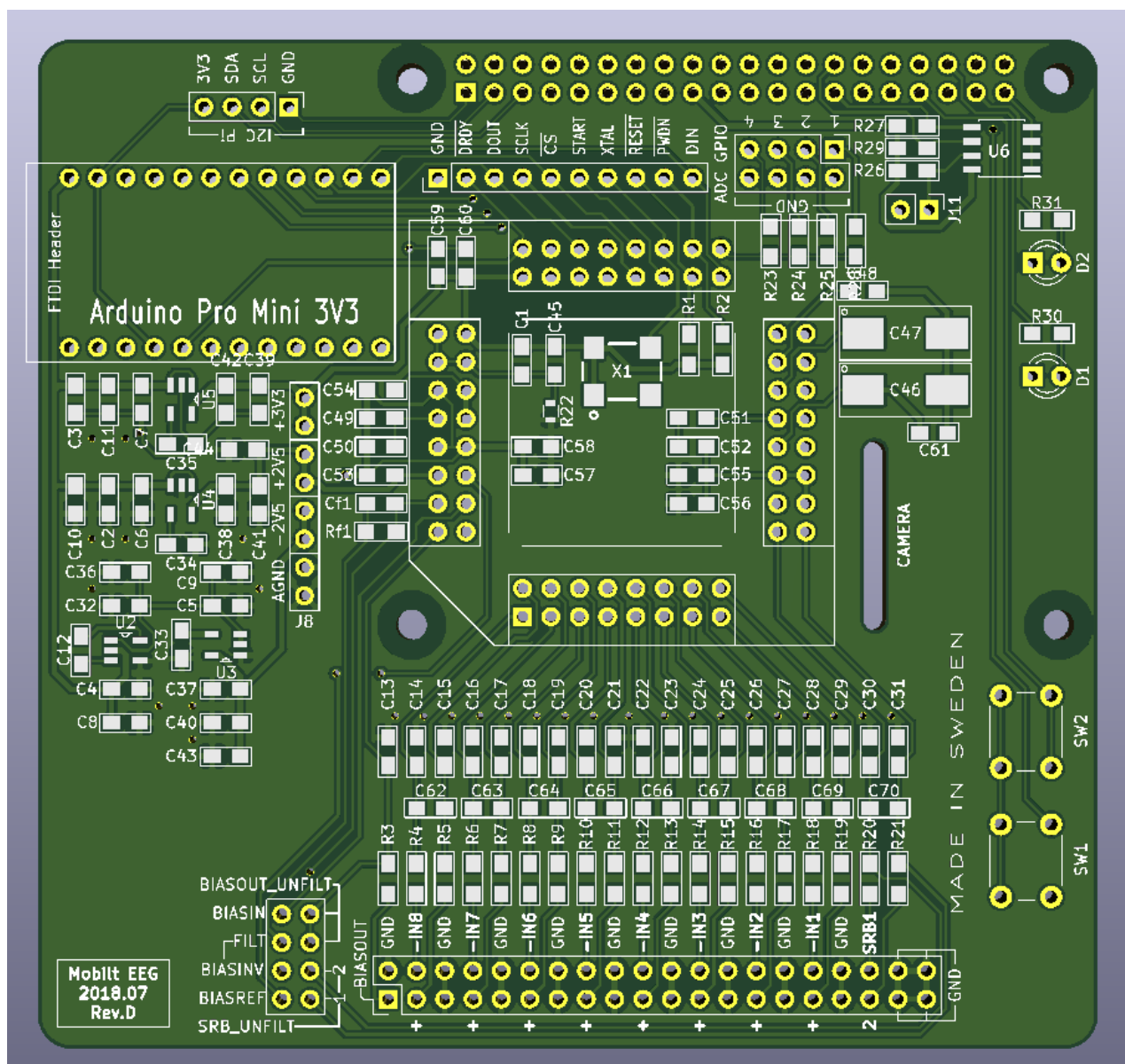
För att åtgärda dessa problem byttes designarbetet till programmet KiCad[10]. KiCad är en open-source PCB design programsvit som innefattar följande kärnprogram: schemaritningsverktyget Eeschema, layoutverktyget PcbNew och Gerbervisaren GerbView. Det finns även program som utökar funktionaliteten, som en speciell komponenteditor för att själv skapa och infoga komponenter utifrån ritningar. Ytterligare funktionalitet kan erhållas med insticksmoduler i form av Python-skript

KiCad har sammanfattningsvis funktioner som gör den jämförbar i vissa fall med rent kommersiell mjukvara utan den ingående komplexiteten som program som riktar sig åt experter har, som Altium Designer, både på gott och ont.

KiCad användes för att skapa en prototyp på ett kretskort som skulle kunna monteras på RPi:n på samma sätt som en HAT och förbinda den med ADC:n, elektroder, MCU och eventuellt ytterligare periferienheter. Kretsschemat finns i bilaga 9.1. En 3D-rendering på PCB:n finns i fig. 5.4.

Versionshantering av KiCad filer med Git KiCad har, i version 5.0.1 som användes under detta arbete, inget stöd för att välja sparfil, utan man har jobbat ständigt mot en och samma sparfil genom hela projektet. Detta gör det omöjligt att spara projektet vid ett visst tillstånd eller jobba på flera olika instanser samtidigt. För detta problem har Git varit en naturlig lösning, med *commits* som fångar projektets tillstånd vid en viss tidpunkt och *branches* som tillåter flera olika 'förgreningar' av samma grundtillstånd.

KiCad sparar sin data i läsbar form, precis som källkod, vilket gör att Git även kan använda funktioner för att lösa konflikter mellan inkompatibla filversioner.



Figur 5.4: 3D-rendering av PCB-prototypen. Anslutningspunkter för ADC modul mitterst, ADC in- och ut signaler med filtrering nederst, ADC spänningsreglering till vänster och Atmega-328p modul högst upp till vänster. Monteringshål, hål för kameramodulens flatkabel, EEPROM och RPi GPIO anslutning finns, enligt RPi HAT standarden, längst upp till höger.

5.3 Mjukvarudesign

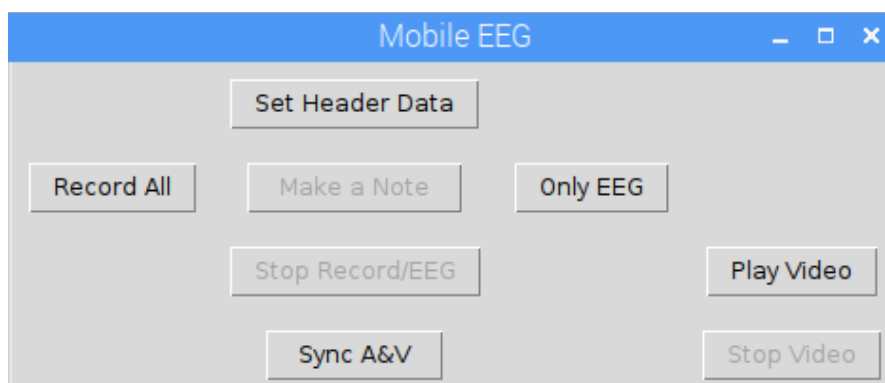
Detta kapitel ämnar att beskriva programmet och dess komponenter och hur dessa fungerar och agerar. De huvudsakliga delarna av programmet är följande:

- Initialisering av data och vissa inställningar samt tilldelning av vissa startvärden.
- Inspelning av ljud och bild.
- Inläsning av EEG-data.
- Dataskrivning, fokuserat kring interaktion med PYEDFlib och med den senast skapade filen.
- Videouppspelning via OMXplayer för att visa flödande bild och ljud

Programmet är uppdelat i ett par olika delar beroende på klass, men det mesta är fokuserat i GUI-klassen och dess funktioner. De olika knapparna i GUI:t aktiverar de primära funktionerna, så som starta inspelning, skriva annotering och sätta upp patientdata, också kallad headerdata, samtidigt som de primära funktionerna använder sig av flera av de mindre funktionerna, för att hålla systemet igång på rätt sätt.

5.3.1 MobileEEG_app

MobileEEG_app är huvudklassen för själva GUI:t samt alla dess relevanta funktioner så som Record, Sync och Videoplay. Den är uppbyggd genom att tillämpa TkInter för att sätta upp det fönster man får upp när programmet startas, samt koden för de funktioner som knapparna i fönster aktiverar.



Figur 5.5: Bild på hur vad hur programmet ser ut när det startas.

5.3.2 AnnotationWindow

AnnotationWindow är en simpel klass, vars mål är att verka som struktur och uppbyggnad av fönstret som kommer fram när man vill göra en annotering.

5.3.3 HeaderWindow

HeaderWindow sätter upp fönstret för datahämtning från användaren gällande den information som man vill att EEG-inspelningen ska ha.

5.3.4 StoppableThread

Klass som lägger in en Trace i en tråd som håller koll på när tråden skall stoppas/avslutas.

5.3.5 Record

En av GUI:ts huvudfunktioner, den som innehåller koden för att starta inspelning för ljud och bild, samt skapar subprocesser för dem, och även kallar på ReadEDF för insamlingen av EEG-datan. Resultatet av Record är tre separata filer, en för ljud, en för bild och en för EEG-datan.

5.3.6 Sync

Sync använder sig av avconv, för att sammanföra ljud- och bildfilerna till en enda mp4-fil. Detta gör så att ljud och bild spelas upp tillsammans, som en video. Avconv är en video- och audiokonverterare som finns tillgängligt i Linux.

5.3.7 EDFFunction

Detta är en av huvudfunktionerna, den spelar in endast EEG-data och inte varken ljud eller bild. Funktionen är till för den som endast vill använda produkten som ett EEG, och inte är intresserad av något annat än EEG datan.

5.3.8 ReadEDF

En simpel funktion som ser till att det finns headerdata, och som startar en tråd av EDFFunction, för att kunna köras pseudoparallellt med ljud och bild subprocesserna.

5.3.9 Videoplay

Videoplays uppgift är att använda omxplayer för att spela upp den senaste sammanfogade versionen av ljud och bild, samt att den spelas upp i en viss storlek och position på skärmen.

5.3.10 Videostop

Videostop är till för att stoppa den video som spelas upp för tillfället, den gör detta genom att stänga av alla processer som körs med omxplayer.

5.3.11 Popup

Funktion som sätter upp en instans av ett annoteringsfönster, för att kunna skriva en annotering.

5.3.12 PyEDFlib

Det är med hjälp av det tidigare etablerade pythonbaserade biblioteket PyEDFlib, med sina EDF-specifika funktioner, som programmet lätt kan skriva data enligt EDF-formatet.

Funktionerna som används har för uppgift att bland annat; skriva den data som användaren av GUI:t anger som headerdata till filen, skriva annoteringar till filen med annoteringsdatan, samt att sätta upp allting enligt EDF-formatet.

5.3.13 Inspelning

Inspelning av information, både ljud, bild och EEG data, är en central del av projektet. För detta så har det utvecklats flera olika funktioner, som har hand om varsin del, och som körs parallellt med hjälp av trådhantering. Både ljud och bild var relativt lätta att implementera, då det fanns färdig funktionalitet i operativsystemet Debian för RPi's. Då detta är Linux-baserat så finns det standard Linux-verktyg tillhands att implementera och nästla tillsammans med GUI-paketet TkInter.

Till EEG-inspelningen så var det en klart större och svårare uppgift, då detta handlar om att samla biofysisk data från elektroder. Detta behövs göras via en ADC för omvandling från analog till digital, samt en mikrokontroller för att kunna skicka denna datan på ett pålitligt, snabbt och säkert sätt. Som tidigare nämnts, så är detta den del som inte blev klar, däremot så finns det både design och prototyp för hur det skulle kunna fungera.

Ett problem för projektet var att det inte fanns en färdig lösning för EEG-inspelning att använda sig av tillsammans med den utvalda ADC:n. Detta gjorde att mycket fick göras från grunden, så som design av ett PCB-kretskort för just ADS1299, vilket inte var något som planerades från start, och gjorde, tillsammans med andra problem, att projektet inte blev klart i tid.

5.3.14 Start och Stopp

Huvudfunktionen i GUI:t är att kunna starta samt stoppa en inspelning av såväl ljud och bild som data. Implementationerna av de olika inspelningarna genererar processer samt trådar, vilket gör att för att kunna avsluta dem krävs det att man hanterar processerna samt trådarna på rätt sätt.

Det finns många sätt för systemet att stanna, när den bestämda tiden tagit slut, att man klickat på stopp eller om programmet skulle krascha. Oavsett anledning så behöver systemet hantera detta och avsluta programmet på ett korrekt sätt för att inte någon data ska bli korrupt, att processer inte ligger kvar i bakgrunden och kör eller att minnesläckor orsakas.

Genom att analysera programmet och se vilka platser i koden där programmet ska eller kan avslutas, så kan införa hantering av både trådar och processer och se till att de stängs av på rätt sätt.

5.3.15 Synkronisering av ljud och bild

För att maximera prestandan vid inspelning av olika slags data så används olika program som är speciellt lämpade för ändamålet. För att spela in från kameran så används programmet *raspivid* som är speciellt anpassat för hårdvaran i RPi:en och tillåter viss hårdvaruaccelerering. För att spela in ljud används programmet *arecord* som finns att användas i samband med *ALSA* ljudkortsdrivrutinen.

Båda programmen är relativt 'lågnivå' och genererar data i 'raw'-format, 'wav' för ljud och 'h264' for bild, vilket skulle kunna användas direkt, men eftersom minneskapaciteten på RPi:en är begränsat så sammanflätas de olika dataströmmarna till en komprimerad 'mp4'-fil. Detta görs med programmet *avconv*.

5.3.16 Uppspelning

Från början var tanken att man direkt skulle kunna se EEG-datan, samt video samtidigt i en nästlat videospelare, tillsammans med ett sätt att interagera med datan. Under utvecklingen så kom insikten att det skulle bli mycket jobb, för en visuell del som inte är högst relevant för projektet, utan snarare extra funktionalitet. Detta gjorde att det lades endast fokus under korta perioder, främst för att få en simpel videouppspelning att fungera, och inte mer.

En fördel är att alla filer kan nås genom att ansluta till RPi:en via till exempel SSH, och sedan kan de föras över till en PC eller laptop. Det finns inget komplicerat databassystem, utan de skapas som enkla filer som kan hanteras fritt av användare.

Uppspelningen är relativt primitiv, då det inte finns alternativ att spola eller hoppa fram och tillbaka i uppspelningen eller att välja fil, utan endast start och stop. Det funktionen knappen kallar på gör är att den

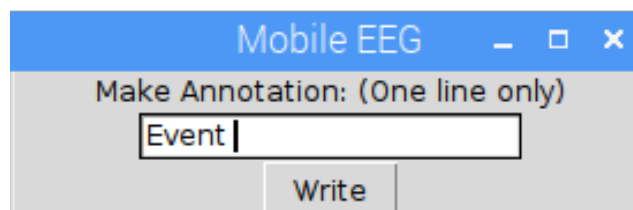
letar upp den senast skapade videofilen, den som är resultatet av synkroniseringen av ljud och bild, och spelar den.

5.3.17 Annotering

Annotering är att man kan lägga in kommentarer under inspelningen av EEG-datan. Detta gör att den som hanterar GUI:t kan, om någon specifik extern stimulus till synes påverkar datan, lägga in en kommentar om vad som orsakade det eller någon annan relevant information.

Detta implementerades genom att använda bakgrundstrådar, samt popoprutor från TkInter. Tanken var först att man skulle lägga in en kommentar samtidigt som man tryckte på annoteringsknappen, men denna ide frångicks av två anledningar: först så var tanken att en popopruta potentiellt skulle distrahera från datan som spelades in under tiden man skrev kommentaren, samt att det skulle bli komplicerat att implementera. Komplikationen är att specifikationer i EDF-formatets implementation i biblioteket som användes tillåter inte att annoteringar läggs in under inspelning. Istället gjordes beslutet om att man får trycka på knappen, så sparas en tidsstämpel för när annoteringen ville läggas in, och efter inspelning så får man upp en popopruta för varje tryck man gjort, då man senare kan skriva de kommentarer man velat.

I efterhand så skulle förmodligen en implementation där man får skriva kommentaren direkt vara bättre, då en EEG-inspelning brukar vara minst 30 minuter och vanligen 60 minuter, under vilken tid man lätt kan glömma vad man ville skriva vid en specifik tidpunkt. Samt att det är potentiellt inte så relevant eller skadligt om man blir distraherad under den korta period det tar att skriva en kommentar, då EEG-data kollas bäst i efterhand då man kan gå igenom datan noga med hög precision.

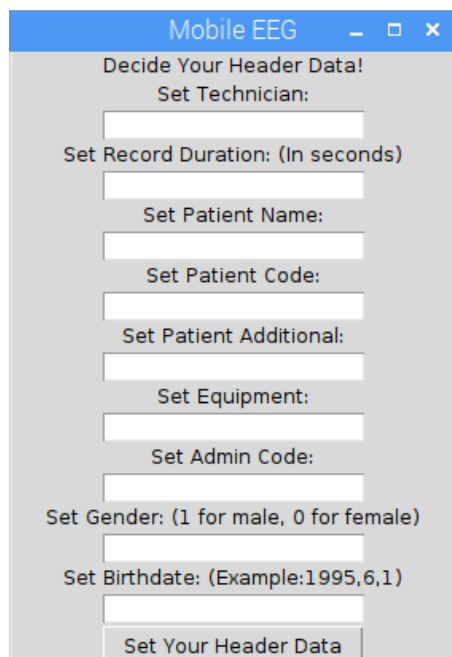


Figur 5.6: Här visas den rutan som visas när man ska göra en annotering efter en inspelning.

5.3.18 Metadata

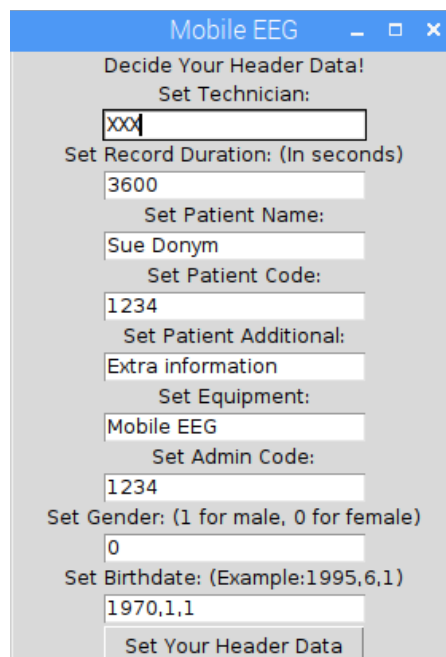
EDF-formatet vill gärna ha en rad olika information, så som namn på patient, namn på den som utför EEG-testet, patientkod, bakgrundsinformation, och många fler. Det bestämdes därmed att detta skulle implementeras.

Det realiserades genom att skapa en egen popurprute-klass för TkInter, vilket hade textfält för all data. Information skrivs in i EDF-formatet via funktioner från PyEDFlib, dock så är det ett krav från funktionerna att detta görs innan inspelning påbörjas för att datan ska skrivas in korrekt. Följden av detta kravet, tillsammans med den mänskliga faktorn, gjorde att det implementerades en simpel säkerhetsåtgärd så att knappar som startar processer som kan störa det som körs för tillfället deaktiveras.



The screenshot shows a window titled "Mobile EEG" with a blue header bar. Below the header, the text "Decide Your Header Data!" is displayed. The form contains several input fields with labels: "Set Technician:", "Set Record Duration: (In seconds)", "Set Patient Name:", "Set Patient Code:", "Set Patient Additional:", "Set Equipment:", "Set Admin Code:", "Set Gender: (1 for male, 0 for female)", and "Set Birthdate: (Example:1995,6,1)". All input fields are empty. At the bottom, there is a button labeled "Set Your Header Data".

(a) Dialogruta med tomma fält.



The screenshot shows the same "Mobile EEG" window, but with example data entered into the input fields. The "Set Technician:" field contains "XXX". The "Set Record Duration: (In seconds)" field contains "3600". The "Set Patient Name:" field contains "Sue Donym". The "Set Patient Code:" field contains "1234". The "Set Patient Additional:" field contains "Extra information". The "Set Equipment:" field contains "Mobile EEG". The "Set Admin Code:" field contains "1234". The "Set Gender: (1 for male, 0 for female)" field contains "0". The "Set Birthdate: (Example:1995,6,1)" field contains "1970,1,1". The "Set Your Header Data" button is still at the bottom.

(b) Dialogruta med exempel på giltig indata.

Figur 5.7: Bilden till vänster visar hur det ser ut när man skriver in sin metadata. Den högra bilden visar ett exempel på hur datan ser ut när den är ifylld. Dialogrutor för att fylla i headerdata i EDF+ filen.

6

Resultat

Mycket arbete har utförts för att få en fungerande prototyp, dock ej fulländad, av ett mobilt EEG. Funktionalitet som saknas inkluderar implementation av kretskort (PCB) och därmed en fulländad produkt, samt design och produktion av en form av huvudbonad för produkten. Kretskortet för att sammanfoga Raspberry Pi:n med ADC:n var tvungen att designas och produceras på egen hand, vilket det inte fanns tid till att göra vid denna tidpunkt då det var i slutet av tiden för projektet när detta blev relevant. Projektet har tyvärr inte uppnått alla delmål som sattes vid projektets start, vilket kommer att diskuteras och förklaras i följande kapitel.

6.1 Summering av svar på frågeställningar

6.1.1 Vilket programmeringsspråk bör användas?

För detta projekt så valdes programmeringsspråket Python, då det troddes kunna uppfylla de behov som fanns. Behov som fanns var bland annat realtidskrav, samt förmåga att implementera ett GUI för användarvänlighet. Språk valdes efter användarvänlighet och flexibilitet istället för uppfyllande av systemkrav, vilket fick vissa konsekvenser för utvecklingen, främst i form av behov av extra lösningar för att hantera problem. Python är ett 'långsamt' språk, vilket innebär att det tar längre tid att exekvera jämfört med andra språk och därmed gjort det svårt att realisera en lösning med hjälp av Python. Ett annat språk bör ha valts, så som C som är mer maskinnära och är ett språk anpassat till uppgifter med realtidskrav. Detta tas upp mer detaljerat i kap. 7, Diskussion.

6.1.2 Vilken samplingsfrekvens bör användas för EEG signalerna.

Ofta när man tänker på mätning av dynamiska signaler så kan man ha den naiva föreställning att signalen skall mätas så ofta som möjligt för att maximera den informationen man kan urvinna. I litteraturen som studerades inför projektet upptäcktes det att de vågformsklasser som är relevanta för EEG endast har en frekvensinnehåll på < 100 Hz, vilket innebär enligt Shannon-Nyquist samplingsteoremet att man endast måste läsa signalen med en frekvens på ca. 200 Hz för att den ursprungliga signalen skall helt kunna återskapas digitalt. Det visade sig senare att det lägsta samplingsfrekvensen som hade stöd i ADC:n var 250 Hz.

6.1.3 Vad finns det för dataformat som är lämpade för EEG signaler? Vilket bör användas i denna tillämpning?

Det finns många dataformat, men det som valdes är EDF-formatet. Detta för att det är en defakto standard, och det finns dedikerade bibliotek för programmeringsspråk som C och Python för att enkelt applicera data till det formatet. EDF-formatet beskrivs i mer detalj i kap. 2, Teknisk Bakgrund.

6.1.4 Klarar systemet alla beräkningsbehov?

Teoretiskt sätt så uppfyller systemet alla beräkningsbehov, dock så finns det problem i den praktiska implementationen som gör det svårt. Den mest uppenbara anledningen till detta är, i härledning till detta projektet, hur ADC:n kommunicerar när den gör redo sin data, samt att en RPi inte är förutsägbar i sin interruptsvarstid. Med detta menas att den är inte deterministisk, och kan därmed inte realisera de hårda tidskraven på att hämta datan i tid. Detta har gjort att det system som projektet resulterade i, inte är en färdig produkt. Det finns en prototyp av en lösning, som dock inte är integrerad med resten av systemet. Mer förklaring kring problemet finns i Teknisk Bakgrund för RPi, ADC samt MCU och även i Diskussion i dess respektive problemsektioner.

6.2 Mål som uppnåts

- Ett simpelt men funktionellt GUI.
- Kan köras med hjälp av batteri.
- Skriva data i EDF-formatet.
- Bild- och ljudinspelning.
- Både design och prototyp av EEG inläsning.
- Enkla säkerhetsåtgärder.

6.3 Mål som ej uppnåts

- En komplett implementation av EEG-inläsningen.
- En design och konstruktion av en huvudbonad samt hållare för all hårdvara.

6.4 Modell av lösning på överförning

Kretskortet för att sammanfoga RPi:n med ADC:n var tvunget att designas och produceras på egen hand.

Att läsa data från ADC:n till Raspberry Pi:n visade sig vara ett omfattande problem då varje sampel som ADC:n producerar måste läsas innan en vis tid innan den skrivs över med nästa sampel. Sampelhastigheten som behövdes för att infånga tillräckligt med data för att inte orsaka informationsförluster sattes enligt Nyquist-Shannons samplingsteorem i ett tidigt skede till 250 Hz. Detta krävde alltså att ADC:n lästes av 250 gånger i sekunden, eller var 4:e millisekund. Det förutsattes att RPi:n, med en systemklocka på 1 GHz skulle klara detta krav, men när man väl försökte implementera kommunikationen så uppenbarades det att datorsystem med linuxbaserade operativsystem behandlar interrupt signaler efter en kort men varierande tid beroende last och processer i kerneln. Experimentellt mättes en sämsta responstid på tiotals millisekunder. Linux är i sitt grundutförande alltså inget realtidssystem och lämpar sig inte för tidskänslig IO behandling.

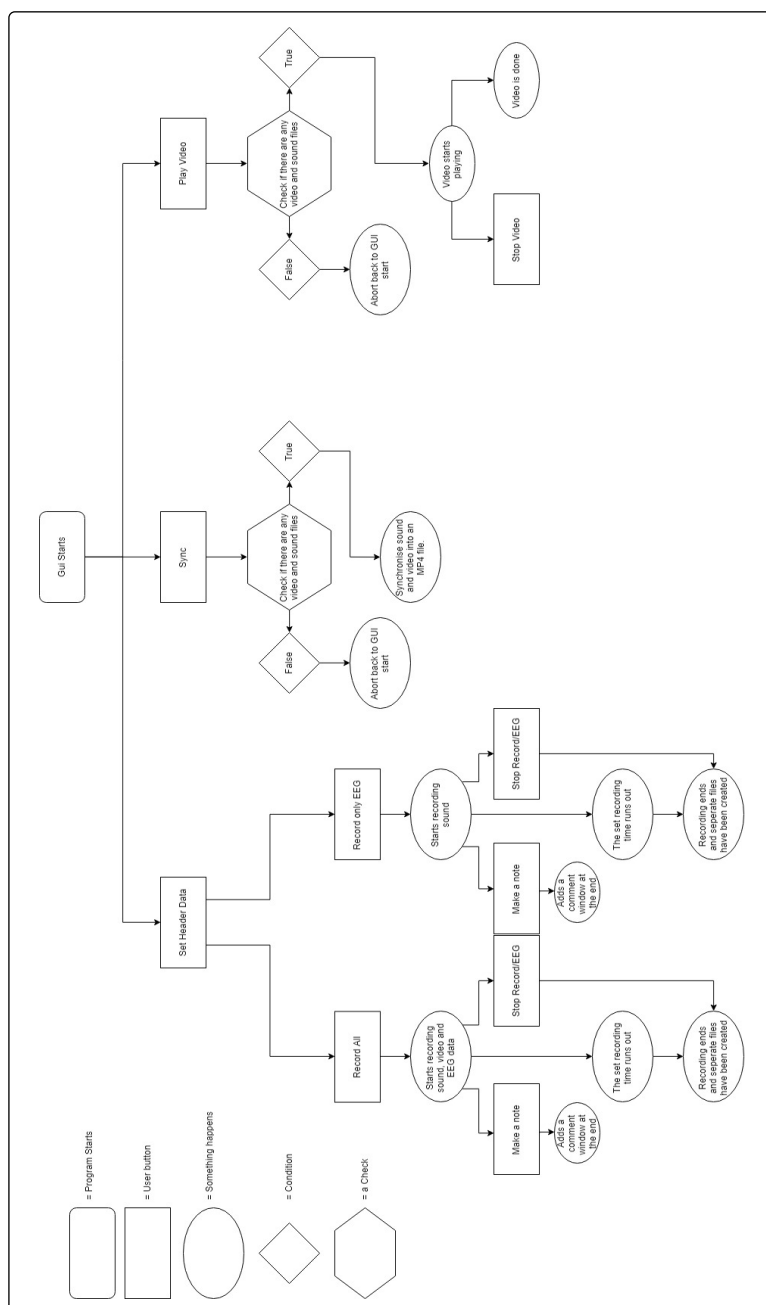
Lösningen på detta var att implementera en slags buffer, en enhet som hade realtidsmöjligheter och som samtidigt kunde kommunicera med RPi:n via ett icke-tidskänsligt protokoll. För detta valdes mikrokontrollern Atmega 328-PU. Detta beskrivs i detalj i kap. 5.2.

Ett alternativ som hittills inte tagits upp är att modifiera själva Linuxkerneln för att stödja realtidsmekanismer och på så sätt få en minskad interruptlatens. Det är just detta som kernelltillägget `CONFIG_PREEMPT_RT`[19] ämnar sig att göra och beskrivs närmare i kap. 7.5.8.

Detta alternativ behandlades inte i detta projekt men är en intressant möjlig lösning till problemet med interruptlatens, och kanske en källa till fortsatt arbete.

6.5 Användning av Mobilt EEG

Den produkt som utvecklats är den mjukvaran och implementation som finns på projektets RPi. För att kunna använda programmet så har ett användargränssnitt skapats. Detta användargränssnitt använder sig av funktionalitet från Pythonbiblioteket PyEDFlib för att kunna utföra specifika dataskrivningar, samt andra funktioner, till skapade EDF-filer. I den aktuella prototypen så är datan simulerad med Pythonbiblioteket NumPy[11], men ska i framtiden kunna hämta in data från en mikrokontroller, och skriva den informationen på EDF-formatet.



Figur 6.1: Flödesdiagram över Mobilt EEG programmet.

Fig. 6.1 visar vyn efter man startat programmet. Det som visas är programmets användargränssnitt. De olika knapparna är enkelt markerade med text för att hänvisa till sin funktion. Som exempel så måste man ha fyllt i metadata via SetHeaderData innan man kan klicka på Record eller Record EEG.

7

Diskussion

Resultatet kommer att diskuteras och därefter de val som gjorts under projektets gång. Kapitlet avslutas med etiska och miljöaspekter av projektet och några förslag på möjliga lösningar på de problem som tas upp här.

7.1 Diskussion av resultat

Ett fungerande Mobilt EEG var huvudmålet med projektet, men den slutgiltiga produkten saknar den viktiga funktionen av att kunna läsa signaler från elektroder och överföra denna data till Raspberry Pi:n. Detta på grund av de problem som uppstått under projektets gång, samt en relativt överklig bild av hur mycket tid olika delar av projektet skulle ta.

Under utvecklingen har vissa beslut tagits, främst för att få arbetet att kunna fortskrida och inte stanna av. Utvecklingen av ett mobilt EEG anses ej vara klart. Däremot så kan det anses vara på grund av gjorda val, och inte en omöjlighet i sig, vilket gör att denna utveckling istället kan användas för att främja framtida utveckling istället för att realisera en egen produkt.

För att förstå varför inte en komplett produkt är klar, så finns det flera saker som behöver undersökas innan det finns ett passande svar.

I början av åtagandet så var visionen helt annorlunda än vad verkligheten tillät. Det fanns vissa delar av utvecklingen som antogs enkla eller inte så tidskrävande, delar som missades helt och även en del problem som direkt resulterades från val som gjorts under utvecklingsprocessen.

7.2 Genomgång av problem

Projektet var öppet för val om hur man skulle utveckla produkten. Början av projekttiden spenderades på att välja ett passande språk, lämplig hårdvara och försöka planera vad man potentiellt skulle behöva göra under utvecklingsprocessen.

Det fanns flera saker som, vid början av projektet, antogs skulle vara sanna.

- Det skulle finnas problem på grund av den biologiska naturen av projektet, speciellt med avseende på brus.
- Det skulle behövas förståelse om hur neural aktivitet påverkar de biologiska processer som ger upphov till de joniska strömmar som ett EEG mäter.
- På grund av att man har inbyggd hårdvarustöd för I/O operationer så påverkar inte operativsystem eller programspråk tidsaspekter i I/O operationer nämnvärt.
- Det borde finnas tid till att pröva flera olika ADC:er
- Det borde skapas en funktionell databas för att lagra de olika resulterande filerna och hålla ordning på tidsstämpling.
- Använda ett externt beräkningsprogram som Matlab eller Mathematica för att avlasta signalhantering/-processering av data, vilket skulle underlätta för Raspberry Pi samt ADC.
- Kunna streama data till en extern display.

Bara en av dessa visade sig stämma, att det skulle finnas problem på grund av biologin hos en människa.

7.2.1 Antaganden

7.2.1.1 Biologiska problem

Det skulle finnas problem på grund av den biologiska naturen av projektet.

Detta är korrekt, men det består främst av hur man sätter upp elektroder enligt 10/20 systemet, samt att rörelse av muskler orsakar starkt brus på den nivå att de joniska strömmarna för EEG blir näst intill oläsbara. Då projektet inte lyckades att pröva det fysiskt så hann detta heller inte bli ett direkt problem, utan är ett problem för framtida utveckling. Det finns metoder för att mildra detta, men av allt att döma är det ett stort arbete i sig.

7.2.1.2 Kunskap om hjärnan

Det skulle behövas förståelse om hur neural aktivitet påverkar de biologiska processer ger upphov till de joniska strömmar som ett EEG samplar.

Det var ett stort missförstånd att det skulle behövas mycket påläsning om hur det fungerar. På grund av detta gick många dagar till spillo, då tiden spenderades på att läsa i böcker och på internet efter information, istället för att arbeta på projektet.

7.2.1.3 Programmeringsspråk inte spelar någon roll

På grund av att man har inbyggd hårdvarustöd för I/O operationer så påverkar inte operativsystem eller programspråk tidsaspekter i I/O operationer nämnvärt. När det undersöktes hur hårdvarukrävande projektet skulle vara, så sa EEG litteraturen att signalerna kunde innehålla frekvenser uppemot 100Hz, vilket innebär en sampeltakt på ca. 200Hz för att infånga hela signalen. Då en processor i en Raspberry Pi modell 2 går mot en processorhastighet mot nästan 1 GHz, och senare modeller är uppe i 1,4 GHz, och har inbyggt hårdvarustöd för externa interrupts samt en mängd kommunikationsprotokoll, så antogs det att oavsett hur långsamt ett språk än var så skulle en snabb processor och hårdvarustöd tillsammans göra så att det inte längre skulle vara ett problem.

7.2.1.4 Pröva olika ADC:er

Det skulle finnas tid till att pröva flera olika ADC:er. Tanken var från början att flera ADC:er skulle jämföras i hur bra det var för data-överföring, men det slutade med att endast ett beställdes och började arbetas på.

ADC:n är en integrerad del i systemdesignen att det skulle bli ett omfattande arbete att anpassa systemet till flera olika ADC:er på grund av skillnader i kommunikationsprotokoll och drivrutin, med andra ord, hur man programmerar ADC:n och tolkar datan man läser på bit-nivå. Andra ADC:er skulle eventuellt även kräva anpassning av övrig hårdvara som spänningsregulatorer för att erhålla andra matningsspänningar.

7.2.1.5 Databas

Det skulle skapas en funktionell databas.

En databas var originellt en bra tanke, då det skulle genereras en mängd data som skulle behöva sparas någonstans. Problemet var att det skulle potentiellt bli komplicerat att lägga in och hämta data. Under utvecklingens gång, när funktioner för att spela in testades, så skapade dessa funktioner filer, som lagrades enkelt i den mapp som programmet körde i. Vi bestämde oss därmed att istället för att skapa en databas för alla filer, så skulle det räcka att de lagras direkt i mappen, med specifika namn beroende på datatyp. Detta för att spara tid, gör det enkelt att komma åt data, och underlätta för de egna funktionerna.

EDF-formatet som används för lagring av EEG signaler har redan en pseudo-databasstruktur och tjänar ett av huvudsyftena med att använda en databas, nämligen tidsstämpling av data.

7.2.1.6 Externt beräkningsprogram

Använda Matlab eller Mathematica för att avlasta signalhantering/processering av data, vilket skulle underlätta för Raspberry Pi samt ADC.

Det fanns en tro att ett dedikerat beräkningsprogram skulle lösa de problem om att hantera signaldatan och även en enkel grafisk representation för datan.

Detta visade sig senare vara fel. Det skulle antingen behövas ytterligare hårdvara endast för att köra beräkningsprogram för att inte överbelasta RPi:n, vilket skulle komplicera kommunikationsprocessen samt överföring, eller så skulle det köras direkt på RPi:n, men då det inte längre ske någon avlastning.

Istället så skulle det finnas ett simpelt system för hantering av datan. ADC läser av, och skickar den till en mikrokontroller. Mikrokontrollern kommunicerar med RPi:n och skapar en länk med två interface för dataöverföring och därmed hanterar kommunikationen mellan ADC och RPi. RPi får in data, och lagrar den i EDF-formatet med hjälp av den utvecklade mjukvaran. Detta är vad som i slutändan finns design och simulerad fungerande prototyp av, men ej en fysisk implementation.

7.2.1.7 Streama Data

Kunna streama data till en extern display.

Att kunna skicka datan till en extern skärm var och är en möjlighet, däremot så bestämdes det relativt tidigt i utvecklingsskedet att det är en relativt onödig funktion. Det skulle behöva en hel del kommunikation mellan olika enheter för att överföra datan, vilket ansågs överflödigt för så simpel mjukvara. Det skulle även kräva implementationen av ett grafiskt ramverk för uppvisning av vågformen, ett betydande arbete i sig.

Dock så kan det påpekas att detta aldrig var ett krav eller mål från början, bara en intressant idé på något som skulle kunna utvecklas.

7.2.2 Oförutsedda problem

Det finns även en lista av problem som inte förutsågs.

- Raspberry Pi ej så lämpad för ändamålet.
- Python ej lämpat för snabb responstid gentemot hårdvara-mjukvara.
- Saknad av färdig lösning för vald ADC (PCB, kommunikationsstruktur etc).
- Extra hårdvara utifall något kortslöts eller gick sönder.
- På grund av tidsbrist så prioriterades huvudbonadens modellering bort.

7.2.2.1 Raspberry Pi ej lämpad

Raspberry Pi ej så lämpad för ändamålet.

Problemet är att en RPi är för generell, och är inte specialbyggd för signalprocessering. Med detta så menas att den begränsade beräkningskraft och processhastighet som RPi:n besitter, är skapad för att vara en generell lösning. Nackdelen är att signalhantering kräver inte ett datorkort med anständig kompetens i det generella perspektivet, utan har specifika krav som kommunikationshastighet mellan signalinhämtning, processering och lagring. RPi:n är därmed inte så lämpad för just detta projektet.

7.2.2.2 Python ej lämpad

Python ej lämpat för snabb responstid gentemot hårdvara-mjukvara Även om hårdvaran i Raspberry Pi är utrustad med hårdvarustöd för interrupts så orsakar Linux kerneln en viss latens tills en interrupt kan behandlas

av program som kör i så kallad *userspace*, program som körs av användare. Valet av programmeringsspråk visade sig lägga till en extra latens som kunde göra att systemet inte längre klarade sina tidskrav. Python har, av sin natur som interpreterad språk, en relativt hög sådan latens. Ett interpreterat språk innebär att källkoden inte kompileras utan evalueras vid körning.

Istället för Python så kunde ett mer maskinnära programmeringsspråk ha valts, så som C. Detta hade gjort det potentiellt svårare att skapa samma funktionalitet som i Python, men det skulle inte ha samma statiska fördröjning, vilket skulle förmildra problemet med att överföringen av data skulle ta för lång tid.

7.2.2.3 Saknad av färdig ADC-lösning

Saknad av färdig lösning för vald ADC (PCB, kommunikationsstruktur etc).

Vid projektets början så var det ett flertal ADC:er som passade projektet, men den som ansågs passa bäst var den som valdes, TI ADS1299. Trots teknisk lämplighet, var det ett omfattande arbete att försöka få en fungerande prototypdesign då den krävde mycket kringutrustning. Man var också tvungen att skriva en drivrutin för att hantera dess ovanliga tidskrav.

För att hantera komplexiteten i ADC-relaterad hårdvara, kom man fram till insikten att ett PCB skulle behövas då komponenterna var för små och för många för att koppla ihop för hand inom rimlig tid. PCB design är en icke-trivial uppgift med en förhållandevis lång inlärningsfas, vilket tog tid ifrån projektet.

7.2.2.4 Extra hårdvara

Extra hårdvara utifall något kortslöts eller gick sönder.

Raspberry Pi 3 Modell B+ som beställdes blev kortsluten under testning av I/O pinnar, och bara en hade beställts. Den fick ersättas av en Raspberry Pi 2, en modell äldre men som fortfarande var funktionellt duglig, i brist på tid på att vänta på en ny beställning.

7.2.2.5 Huvudbonad

På grund av tidsbrist så prioriterades huvudbonadens modellering bort.

Planen var att en 3D-printer på Chalmers skulle användas till att tillverka en huvudbonad för att hålla elektroder på plats, som beskrivet i kap. 4.1, som skulle skapas tillsammans med utvecklingen av projektet. Med tanke på tidsbristen som fanns, tillsammans med det faktum att ingen av projektets medlemmar tidigare skapat tekniska 3D-ritningar med CAD, fick detta låg prioritet.

7.3 Miljö och Etik

Syftet med ett EEG är att samla in och tolka hjärnvågor. En mobil implementation, som är lättillgänglig, lättanvänd och kan användas oberoende på plats öppnar upp möjligheten till ett stort utbud av data.

Eftersom EEG data kan innehålla information om medicinska och mentala tillstånd, så måste de betraktas som känsliga uppgifter. Enligt en studie[17], så har man även kunnat, med hög precision, identifiera individer med EEG, precis som med andra biometriska metoder som ansikts- och fingeravtrycksigenkänning. Detta gör det allt viktigare att se till att dessa uppgifter skyddas ordentligt från intrång, då det potentiellt skulle öppna upp ett nytt sätt för identitetsstöld.

Lättillgängligheten som nämndes innan har därmed en negativ aspekt i att dessa känsliga uppgifter då kan lagras av privatpersoner som inte nödvändigtvis respekterar detta ansvar.

Ett stort utbud på EEG-data skulle, å andra sidan, potentiellt kunna användas för att lära sig mer om hur individer tar in och katalogiserar information i en utbildningsmiljö, för att kunna anpassa studietekniker, utbildning och en bra miljö för optimala förhållanden för inläring. Detta skulle i sin tur kunna leda till

en potentiellt högre allmän utbildning världen över, då mycket av exakt hur människan lär sig och tar in information fortfarande är relativt okänt. På samma sätt skulle man även kunna lära sig om och anpassa sociala miljöer, vilket skulle kunna användas för att reducera stressnivå, då en miljö anpassad efter att hitta lugn hos en person med hjälp av EEG kan identifiera saker som påverkar en individ på det sättet.

7.4 Övriga tankar

I grund och botten så är ett av de större problemen att projektet siktade på att skapa en lösning för ett specifikt fall, med generella lösningar. Det som hände var att de generella lösningarna var för generella för att kunna uppfylla de specifika krav som visade sig finnas för ett EEG med komplex signalhantering.

- Valde Raspberry Pi, skulle valt en enhet med låg och förutsägbar interruptlatens, som en Arduino eller annan mikrokontroller.
- Valde Python, skulle valt något mer maskinnära som C eller C++.

Man kan därmed beskriva projektet som att det redan från början såg mörkt ut. Å andra sidan så finns det en produkt, med en prototyp på signalhämtning och leverans till programmet. Den stora nackdelen är dock att den förmodligen inte skulle vara snabb nog för stora mängder data även om målen blev uppfyllda.

7.5 Möjliga Lösningar

Här kommer ett par alternativa lösningar presenteras. Dessa är tankar som möjligtvis hade antingen gjort att projektet lyckats, eller löst samma problem på ett annat sätt.

7.5.1 En enda mikrokontroller

Genom att använda en betydligt kraftigare mikrokontroller kan man möjligtvis integrera många funktioner som RPi:n utför, såsom kamera- och mikrofoninspelning.

Många mer moderna mikrokontrollers har dessutom stöd för USB och kan därmed kommunicera i en högre hastighet.

7.5.2 Interrupt knapp som startar båda hårdvaran samtidigt

Man skulle kunna dela upp systemet i två oberoende delar, en som utför inspelning av ljud och bild och en annan som styr och lagrar data från ADC:n. Datan skulle då extraheras och sammanfogas 'för hand' efter en körning. En nackdel är att en interruptsignal kan mottas med olika fördröjningar på olika system och under olika körningstillstånd, vilket gör att datan från de olika delarna blir något osynkroniserat. Detta går troligtvis att mildra men inte eliminera. Man kan troligtvis försumma interruptfördröjningen på en mikrokontroller under de flesta omständigheter. RPi:n har en typisk interruptfördröjningstid för program i *userspace*, till skillnad från *kernel space* som användare inte kan påverka, som man skulle kunna utnyttja. Denna fördröjningstid har dock stor spridning, alltså att den med låg sannolikhet avviker avsevärt från normalvärdet, vilket gör den mindre pålitlig [12].

7.5.3 Extern lagring

För att undvika att behöva kommunicera mellan mikrokontroller och RPi under tidsbegränsning så kan mikrokontrollern istället skriva sensordata direkt till någon form av extern lagring, som ett SD-kort. Detta har fördelen att man inte är beroende av något annat system och kan garantera att all data sparas.

7.5.4 Extern buffer

En rimlig lösning på problemet med asynkron överföring av relativt stora datamängder är att använda en buffer mellan den skrivande (skapande) och den läsande (konsumerande) enheterna. En vanlig typ av buffer är en cirkulär buffer[30], som implementerar en Linked-List som skriver över första minnesadressen när den allokerade minnet tar slut. Den skriver alltså över den äldsta datan först och på så sätt kan aldrig överskrida det allokerade minnet, vilket är viktigt på minnesbegränsade enheter som mikrokontrollers. Som exempel på minnestyper som kan användas finns *EEPROM*- och *Flash*minne. Eftersom minnet blir en delad resurs som kräver *mutual exclusion* (sv. ömsesidig uteslutning), det vill säga att bara en enhet får använda resursen i taget, så måste någon form av åtkomstprotokoll implementeras.

7.5.5 DMA (Direct Memory Access)

DMA är ett sätt att läsa primärminnet på en enhet utan att processorn behöver vara inblandad. Följden är att man inte binder upp processorn som därmed kan utföra andra tidskänsliga funktioner. Detta skulle vara ett effektivt sätt att hantera envägs kommunikation (simplex) av stora mängder med data. Det krävs dock, som i de flesta realtids sammanhang, att man är försiktig så att primärminnet inte förändras under tiden man läser den. Dedikerad DMA hårdvara finns på många moderna mikrokontrollers.

7.5.6 Trådlös överföring

Inbyggd hårdvara för trådlös kommunikation, som Bluetooth, IEEE 802.11 WiFi, finns hos i princip alla moderna mobiltelefoner. Dessa standarder är beprövade och robusta och skulle kunna användas för att styra en mikrokontroller, förutsatt att även mikrokontrollern har inbyggt stöd eller kan kompletteras med en adapter av något slag. Dessa protokoll har fördelen att de är mycket mer sofistikerade än både SPI och RS-232 och därmed möjligtvis kan abstrahera en del av designarbetet, genom att använda verifierade bibliotek.

7.5.7 Realtids-OS för RPi

Realtidsoperativsystem för RPi:n undersöktes inte som alternativ under själva arbetet. Det finns dock exempel på detta som ChibiOS/RT[13] som påstås implementera ett fullfjädrat realtidskernel på RPi. Med det skulle man kunna ta bort MCU:n från designen helt och garantera att RPi uppfyller alla realtidskrav. Av större betydelse är dock det man tappar: linuxmiljön och drivrutiner för periferienheter. Det skulle också innebära ett stort paradigm-skifte för programmeringen, något som kostar i tid och arbete. Det kan också argumenteras att, ska ett RTOS användas i systemet, så är en enhet som RPi grovt överpresterande för ändamålet.

7.5.8 Realtidstillägget CONFIG_PREEMPT_RT för Linux

Något likt förslaget ovan, så skulle man kunna modifiera själva Linuxkerneln för att stödja realtidsmekanismer. På så sätt kan man bevara tidigare funktionalitet och drivrutiner medans man får realtidsbeteende.

Med CONFIG_PREEMPT_RT realtidstillägget för Linuxkerneln så kan man lägga till stöd för prioriteter. Detta kräver en omkompilering av kerneln och innebär att processer med högre prioritet kan 'ta över' processorn från processer med lägre prioritet (i vad som, på engelska, kallas *preemption*) med syftet att processer med högst prioritet alltid skall betjänas först.

Detta skall kunna leda till lägre latenser om man förutsätter att IO operationer, som interrupts, har en hög prioritet och därmed exekveras innan andra, lägre prioriterade, processer.

Den latensminskning som man kan få i verkligheten med realtidstillägget varierar givetvis beroende på processorhårdvara, arkitektur, kernelversion och fler faktorer, men enligt den officiella hemsidan så är det möjligt att få interruptlatenser på under en millisekund[19], vilket skulle kunna vara tillräckligt lågt för att kommunicera med ADC:n i denna rapport. Ett exempel på en RPi som kör Linux 4.19.8 med realtidstillägget visar med testprogrammet `cyclictst` en minskning av den maximala latenstiden från 5ms till 200 μ s[27].

8

Framtida Utveckling

Då projektet inte uppfyllt de mål som sattes från början, så kan mycket av den framtida utvecklingen tas från att uppnå de målen. Däremot så finns det många saker man kan tänka sig att programmet kan göra i framtiden, för att antingen göra den mer dynamisk, lättanvänd eller bättre.

- Fullända end-to-end samt även inställning av vilka punkter i 10-20 systemet man skulle vilja mäta mellan.
- Nästla in videouppspelning, att man kan se EEG-data och se den spelas upp samtidigt, kunna klicka och stanna under uppspelning för att se var i videon EEG-data spelades in, alltså se kopplingen mellan extern stimuli och EEG-data, live i uppspelningen.
- Välja fil/filer som spelas upp i uppspelaren tillsammans, samt implementera spola fram och tillbaka, och även en 'tidsaxel' man kan klicka på för att hoppa fram och tillbaka i uppspelningen.

På härvarusidan är man främst begränsad av antalet ADC kanaler. Ett framtida arbete skulle kunna vara att utveckla designen för att stödja fler samtida ADC-mätningar. Detta skulle kräva antingen att man bytte ADC till en med fler inbyggda kanaler eller använde fler av det befintliga ADS1299 chippet. För den sistnämnda lösningen skulle man kunna åtgärda den extra kommunikationen antingen genom att "kaskadkoppla" flera enheter, det vill säga använda samma SPI-buss men adressera enheterna separat med olika \overline{CS} signaler, eller använda den inbyggda daisy-chain- mekanismen[1, s. 65] för att kunna läsa av flera ADC:er på samma sätt som man gör med en. Båda lösningar kräver en större omdesign av PCB:n.

9

Bilagor

9.1 PCB-kretsschema

Se nästa sida.

Referenser

- [1] 2018. URL: <http://www.ti.com/product/ADS1299>.
- [2] 2018. URL: <https://www.python.org/> (hämtad 2018-08-16).
- [3] 2018. URL: <https://wiki.python.org/moin/TkInter> (hämtad 2018-08-16).
- [4] 2018. URL: <https://www.teuniz.net/> (hämtad 2018-08-16).
- [5] 2018. URL: <https://www.teuniz.net/edflib> (hämtad 2019-04-26).
- [6] 2019. URL: <https://easyeda.com/> (hämtad 2019-04-28).
- [7] 2019. URL: <https://jlcpcb.com/> (hämtad 2019-04-28).
- [8] 2019. URL: <https://lcsc.com/> (hämtad 2019-04-28).
- [9] 2019. URL: <https://github.com/raspberrypi/hats> (hämtad 2018-08-16).
- [10] 2019. URL: <http://kicad-pcb.org/> (hämtad 2019-04-28).
- [11] 2019. URL: <https://www.numpy.org/> (hämtad 2019-04-28).
- [12] 2019. URL: https://www.tablix.org/~avian/blog/archives/2016/04/measuring_interrupt_response_times/ (hämtad 2019-02-28).
- [13] 2019. URL: <https://www.stevebate.net/chibios-rpi/GettingStarted.html> (hämtad 2019-04-28).
- [14] *Arduino Forum*. 2019. URL: <https://forum.arduino.cc/> (hämtad 2019-04-04).
- [15] *AtMega 328p - Datasheet*. 2019. URL: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf (hämtad 2019-04-16).
- [16] L. Bengtsson. *Elektriska mätsystem och mätmetoder*. Studentlitteratur, 2012.
- [17] *Brain waves can be used to detect potentially harmful personal information*. 2019. URL: <https://www.sciencedaily.com/releases/2016/10/161003130904.htm> (hämtad 2019-03-17).
- [18] A. Calabria. *Understanding Lead-Off Detection in ECG*. Tekn. rapport SBAA196A. Texas Instruments, 2012. URL: <http://www.ti.com/lit/an/sbaa196a/sbaa196a.pdf> (hämtad 2019-05-03).
- [19] *CONFIG PREEMPT RT Patch*. 2019. URL: https://rt.wiki.kernel.org/index.php/CONFIG_PREEMPT_RT_Patch (hämtad 2019-05-05).
- [20] B. Kemp och J. Olivan. European data format ‘plus’ (EDF+), an EDF alike standard format for the exchange of physiological data. *Clinical Neurophysiology* **114**.9 (2003), 1755–1761. ISSN: 1388-2457. DOI: [https://doi.org/10.1016/S1388-2457\(03\)00123-8](https://doi.org/10.1016/S1388-2457(03)00123-8). URL: <http://www.sciencedirect.com/science/article/pii/S1388245703001238>.
- [21] S. Lee och J. Kruse. *Biopotential Electrode Sensors in ECG/EEG/EMG Systems*. 2008. URL: https://www.analog.com/media/en/technical-documentation/white-papers/ecg-eeeg-emg_final.pdf (hämtad 2019-03-17).
- [22] F. Maloberti. *Data Converters*. Springer, 2007. ISBN: 9780387324852.
- [23] *OMXPlayer: An accelerated command line media player*. 2019. URL: <https://www.raspberrypi.org/documentation/raspbian/applications/omxplayer.md> (hämtad 2019-04-23).
- [24] R. Oostenveld och P. Praamstra. *The five percent electrode system for high-resolution EEG and ERP measurements*. 2001.
- [25] *Raspberry Pi Forums*. 2019. URL: <https://www.raspberrypi.org/forums/> (hämtad 2019-04-04).
- [26] *Raspberry Pi Model 3B+ Specifications*. 2019. URL: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/> (hämtad 2019-04-16).
- [27] *Real-Time Linux on the Raspberry Pi*. 2019. URL: <https://www.get-edi.io/Real-Time-Linux-on-the-Raspberry-Pi/> (hämtad 2019-05-05).
- [28] *Texas Instruments - Data Converters - Forum*. 2019. URL: <http://e2e.ti.com/support/data-converters/f/73> (hämtad 2019-04-04).
- [29] K. Tindell och H. Hansson. Real time systems and fixed priority scheduling. *Technical Report Tech. rept.* (1995).
- [30] K. Wada. *Ring buffer basics*. 2019. URL: <https://www.embedded.com/electronics-blogs/embedded-round-table/4419407/The-ring-buffer> (hämtad 2019-02-28).
- [31] J. G. Webster och J. W. Clark. *Medical instrumentation : application and design*. Hoboken, NJ : John Wiley & Sons, cop. 2010., 2010. ISBN: 0-471-67600-4.