Bacheloroppgåve

NTNU
Noregs teknisk-naturvitskaplege universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for elektroniske system

Ellen Iren Johnsen

Utvikling and prototyping av IoT node for distribusjon av fastvare

Development and prototyping of IoT node for deployment of firmware

Bacheloroppgåve i Elektronikk og Sensorsystemer
Rettleiar: Arne Midjo
Medrettleiar: Carsten Wulff
Mai 2024

NTNU
Kunnskap for ei betre verd

Ellen Iren Johnsen

Utvikling and prototyping av IoT node for distribusjon av fastvare

Development and prototyping of IoT node for deployment of firmware

Bacheloroppgåve i Elektronikk og Sensorsystemer
Rettleiar: Arne Midjo
Medrettleiar: Carsten Wulff
Mai 2024

Noregs teknisk-naturvitskaplege universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for elektroniske system

NTNU

Kunnskap for ei betre verd

NTNU

Tittelside Bacheloroppgave BIELEKTRO

Oppgåvetittel (norsk og engelsk):
NO: Utvikling and prototyping av IoT node for distribusjon av fastvare

EN: Development and prototyping of IoT node for deployment of firmware

Forfattarar:
Ellen Iren Johnsen

Prosjektnummer:	E2418-0001
Innleveringsdato:	21.05.2024
Gradering:	[x] Open [ ] Lukka

Studium:	Elektroingeniør - BIELEKTRO
Studieretning:	Elektronikk og Sensorsystem
Rettleiar internt:	Arne Midjo
Institutt:	Institutt for elektroniske systemer
Oppdragsgiver:	Nordic Semiconductor
Kontaktperson og Teknisk rettleiar:	Carsten Wulff

Sammendrag (norsk og engelsk):
Nordic Semiconductor og NTNU eksperimenterer med å nytte nRF Development Kits som nodar i eit nettverk for å orkestrera kanalmålingar i større skala. Under utviklinga kan eit slikt system gå gjennom mange iterasjonar av fastvare for noden. Fastvare er tradisjonelt permanent programvare innebygd i ei elektronisk eining. Nyare lagringsmetodar som Flash kan likevel omskrivast eksternt. Målet med dette prosjektet er å utvikla og prototype ei utviding til nRF52833DK som kan lasta ned ny fastvare trådlaust og skriva ho til internminnet til nRF52833DK ved hjelp av SWD-grensesnittet

Nordic Semiconductor and NTNU are experimenting with using nRF Development Kits as nodes in a network to orchestrate large scale channel sounding measurements. During development such a system may go through many iterations of firmware on the nodes. Firmware is traditionally permanent software embedded into an electronic device. However, newer storage methods like Flash can be externally rewritten. The objective with this project is to develop and prototype an extension to the nRF52833DK that can download new firmware wirelessly and write it to the internal memory of the nRF52833DK using the SWD-interface

Stikkord norsk:	**Stikkord engelsk:**
SWD	SWD
Debugger	Debugger
Fastvare	Firmware
ESP32	ESP32
nRF52833DK	nRF52833DK
Tingenes Internett	Internet of Things

Samandrag

Kanalmåling er ein teknikk som kan nyttast til å måle distansar mellom objekt. Nordic Semiconductor i samarbeid med NTNU utviklar prosjektet Massive Distributive Channel Sounding Equipment(MaDChaSE) som skal skape eit større nettverk av nodar av typen nRF52833DK for innsamling av kanalmålingsdata. Eit nettverk av denne typen med opp-til fleire hundre nodar presenterer fleire problem som må utforskast før eventuellt oppsett.

Målet med dette bachelorprosjektet er å skape ein prototype som trådlaus kan distribuere fastvare til nRF52833DK, utan å påverke målingar. Vidare er det ynskjeleg at resultatet skal være enkelt i oppsett, kan kontrollere noder individuelt, kunne overføre måledata frå noden trådlaust til ein sentral server og er mindre økonomisk ressurskrevjande enn ei tilsvarnde løysing med Raspberry Pi 3+ kopla til kvar enkelt nRF52833DK eining.

Det blir tatt utgangspunkt i mikrokontrollaren ESP WROOM32 med Development Board. Denne eininga har nok lagringsplass til å handtere filene krevd for fastvareoppdatering av nRF52833DK gjennom SWD. Den har vidare innebygd WiFi-modul som gjev den høve til å gjennomføre nedlasting av fastvarefiler og IoT-basert kontroll av eininga. Det blir utvikla programvare som gjer at ESP32-eininga både fungerar som SWD-grensesnitt opp mot nRF52833DK og som IoT-node kopla til eit større nettverk.

Det er vidare tatt utgangspunkt i at IoT-nettverket er av typen lokalt trådlaust nettverk, og at alt skal kunne kontrollerast av ei styringseining som kan være ei personleg data-maskin. Det blir frå styringseininga utvikla eit IoT-system basert på MQTT-meldingsoverføringar, og sett opp ein enkel HTTP-server for å tilgjengleggjere filer for nedlasting til ESP32.

Det ferdigstilte prosjektet oppfyller ynskja gjevne, og er samansett av ein IoT-node med ESP32DB og nRF52833DK og eit IoT-nettverk som kan settast opp med ei personleg datamaskin og ein mobiltelefon. Prosjektet kan potensielt samanførast med MaDChaSE, og det er identifisert fleire forbetringsområder som bør utviklast før større system blir laga.

Abstract

Channel sounding is a technique that can be used to measure distances between objects. Nordic Semiconductor in collaboration with NTNU is developing the Massive Distributive Channel Sounding Equipment (MaDChaSE) project to create a larger network of nRF52833DK nodes used to collect channel sounding data. A larger network of nodes such as this presents several problems that need to be explored before any setup.

The goal of this bachelor project was to create a prototype that can wirelessly distribute firmware to the nRF52833DK, without affecting measurements. Furthermore, it is desirable for the prosject result to be easy to set up, able to control nodes individually, able to transfer measurement data from the node wirelessly to a central server and be less resource-intensive than a similar solution with Raspberry Pi 3+ connected to each individual nRF52833DK unit.

The starting point is the ESP WROOM32 Development Board microcontroller. This device has enough storage space to handle the files required for firmware updates of nRF52833DK through SWD. It also has a built-in WiFi module that enables downloading of firmware files and IoT-based control of the unit. Software is being developed to enable the ESP32 device to function both as a SWD interface to the nRF52833DK and as an IoT node connected to a larger network.

It is further assumed that the network is a local wireless network, and that everything can be controlled by a control unit that can be a personal computer. The control unit will develop an IoT system based on MQTT message transfers, and set up a simple HTTP server to make files available for download to the ESP32.

The completed project fulfils the wishes given and is composed of an IoT node with ESP32DB and nRF52833DK and an IoT network that can be set up with a personal computer and a mobile phone. The project can in the future be combined with MaDChaSE, but several areas of improvement have been identified that should be developed before a larger system is created.

Forord

Denne prosjektrapporten markerer ferdigstilling av den tre årige utdanninga Bachelor i Ingeniørfag: Elektronikk og Sensorsystemer ved Norges Teknisk-Naturvitenskaplige Universitet i Trondheim. Prosjektarbeidet har til tider vore utfordrande, men òg lærerikt. Arbeidet demonstrerer i sum ferdigheiter og dugleik utvikla i utanningsprosessen innan ingeniørfaget.

Oppdragsgjevar for prosjektarbeidet er Nordic Semiconductor ASA, i samarbeid med NTNU. Det har vært svært gjevande å bruke denne moglegheiten til å sette seg inn i ulike teknologiar og skape eit produkt som potensielt kan vidareførast under utvikling av prosjektet MaDChaSE.

Spesielt ynskjer eg her å takke intern rettleiar Arne Midjo, og teknisk rettleiar Carsten Wulff. Begge har stilt med engasjement, gode faglege råd og rettleiing for arbeidet gjennom heile prosjektperioden. Vidare bør Adrian Heyerdahl ved Company of Things takkast for gode innspel og råd i samband med organisering av IoT-kommunikasjon.
Tusen takk for moglegheita til å gjennomføre denne oppgåva!

Ellen Iren Johnsen
Dato: 21.05.2024

Innhald

Figurar

Tabellar

Kodesnuttar

1 Terminologi

.bin	Binærfil lagrar data i binært format sekvensielt
.hex	ASCII tekstfil som følgjer Intel HEX filformat
Debugger	Eit verktøy brukt for å finne og rette opp feil i programvare
DRAM	Dynamic Random Access Memory - Dynasmisk arbeidsminne
Flash	Varig minnetype som kan omskrivast
IoT	Internet of Things - Tingenes Internett
GPIO	General Purpose Input Output - Genrell Ingong/Utgong
MaDChaSE	Massive Distributed Channel Sounding Equipment
RAM	Random Access Memory - Type arbeidsminne
ROM	Read Only Memory - Dataminne
SoC	System on a Chipc- System på ei mikrobrikke
SRAM	Static Random Access Memory - Statisk arbeidsminne
SWD	Serial Wire Debug
URL	Uniform Resource Locator - Unik nettadresse

2 Innleiing

2.1 Bakgrunn

MaDChaSE (Massive Distributed Channel Sounding Equipment) er eit prosjekt i regi av NTNU og Nordic Semiconductor, der målet er å skape måleutstyr for å gjennomføre kanalmålingar i eit nettverk med opptil fleire hundre nodar som samarbeider.
Kanalmåling er ein prosess der ein målar fysiske kommunikasjonskanalar for å hente ut informasjon som distanse. Det er vist at vi i dag kan måle avstanden mellom einingar med stor nøyaktigheit når ein nyttar kanalmåling[18][25].
Målet med prosjektet MaDChaSE er å vurdere om det er mogleg å nøyaktig måle avstanden til andre objekt i målingsmiljøet dersom ein gjennomførar kanalmåling med fleire nodar i eit målingsnettverk. Eit slikt målingsnettverk har tidlegare vore vanskeleg å realisere, gitt ressurskrava til å sette opp kanalmåling, men i dag kan alt av fastvare som er naudsynt for kanalmåling køyre på nRF52833 Development Kits.

Grunnlaget for denne oppgåva er ynskjet om å skape eit verktøy som kan gjennomføre trådlaus oppdatering av fastvare og avlesing av minneregister på nRF52833 DK. Verktøyet skal gjere oppsett og utvikling for MaDChaSE prosjektet meir effektivt og mindre ressurskrevjande. Vidare skal prosjektet kunne vidareutviklast til å handtere målingsorkestrering og andre arbeidsoppgåver.

2.2 Avgrensing og mål

I oppgåveforslaget utgitt ved NTNU står det at oppgåva skal dekke prototyping og utvikling av noder som kan gjennomføre kanalmåling i nettverk av hundrevis av nodar[31]. Oppgåveforslaget stiller seg åpen til utforsking og ulike løsninger.
Vidare i oppgåveforslaget heiter det at ei potensiell arbeidsoppgåve kan være å "utvikle verktøy for oppdatering av fastvare på nRF-Development Kits". Gitt at bachelorprosjektet har ein student, stiller rettleiarar seg positive til at fokus på denne arbeidsoppgåva vil resultere i eit bachelorprosjekt av passande omfang.
Etter undersøking av liknande prosjekt, blir det vurdert at det er potensial for å nytte tilkoplingspunktet for eksterne debuggere på nRF52833DK for å oppdatere fastvaren. Dette tilkoplingspunktet støtter kommunikasjon med JTAG og SWD, som kan brukast til å manipulere fastvaren til nRF52833DK eininga. Prosjektet skal prototype ei løysing som brukar dette punktet til trådlaus fastvareoppdatering og kan gjennomføre trådlaus minneavlesing.

2.2.1 Mål

Hovudmål for prosjektet er originalt skildra i Endringsnotat for forprosjekt, som er lagt ved som vedlegg. Måla skildrar kort kva prosjektet skal oppnå av resultat, verknadar og prosess.
Effektmål: Det skal være enklare for framtidige utviklarar av MaDChaSE prosjektet å sette opp oppdaterbare nodenettverk for testing
Resultatmål: Det skal produserast ein nRF-basert node som kan bruke metoder for debugging for å oppdatere fastvare
Prosessmål: Deltakar får brukt og vist tidlegare tileigna kunnskapar frå studiet, i tillegg til å vise eignaheit innan sjølvstendig læring og prosjektutvikling
Bærekraftsmål: Det vurderast naturleg å kople dette prosjektet opp mot FNs-bærekraftsmål: Industri, Innovasjon og Infrastruktur[26], grunna bakgrunnen knytta til vidare forsking

2.2.2 Prosjektkrav

Basert på oppgåveforslaget, satte mål og innspel frå rettleiar er det utvikla ei rekke minimumskrav som skal sikre moglegheit for bruk og vidareutvikling.

- Fastvareoppdatering skal kunne gjennomførast trådlaust
- Systemet skal kunne organisere oppdatering av fastvare
- Systemet skal være mindre økonomisk ressurskrevjande enn tilsvarande alternativ
- Systemet skal bruke komponent som er lett tilgjengleg
- Systemet skal kunne settast opp av person med programmeringskunnskapen til ein gjennomsnittleg ingeniørstudent
- Produksjon av fastvare skal kunne gjennomførast med Nordic Semiconductor sin eksisterande verktøykjede

2.3 Utforming av rapporten

Teori Formålet med dette kapittelet er å gjere lesar kjend med relevant teori som blir brukt i resten av rapporten
Metode Kapittelet skildrar tilgjenglege ressursar, planleggingsarbeid og testmetodikk
Implementasjon Kapittelet skildrar kva som blei gjort, korleis og kvifor
Resultat Kapittelet legg fram resultat av testar
Drøfting Kapittelet vurderar implementasjon og resultat satt opp mot skildra mål og vedlagd teori. Vidare drøftast potensiell vidareutvikling av prosjektet
Konklusjon Kapittelet oppsummerar kort innhaldet i rapporten og gir eit definitivt svar på problemstillinga

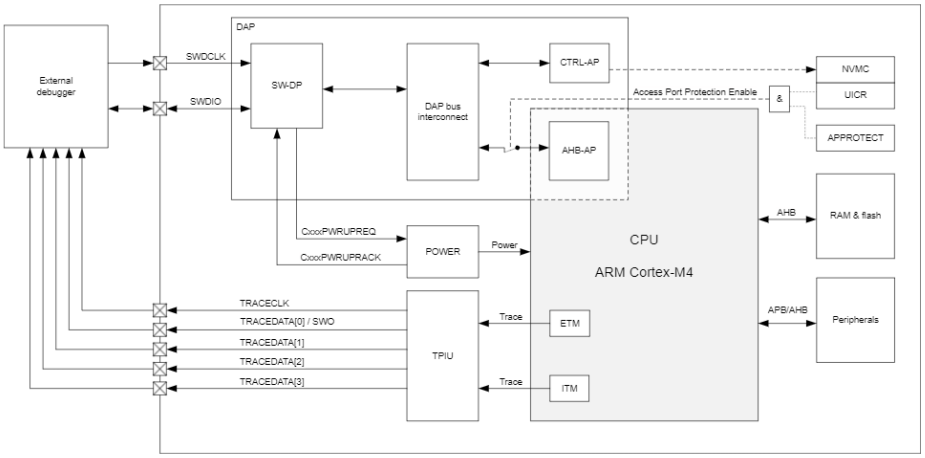
3 Teoretisk Grunnlag

3.1 SWD

Serial Wire Debug er eit alternativ til JTAG-grensesnittet og JTAG-protokollen utvikla av ARM[6]. I oppsettet skil ein mellom 'kontrolleren' som leser av frå og skriver data til 'målet'.

3.1.1 Fysisk grensesnitt

Det fysiske grensesnittet til SWD brukar to pinnar, SWDCLK og SWDIO. Pinnane er direkte kopla til SWD-Debug Port som kan gje vidare tilgong til ein eller fleire Access Ports. Figur:1 syner implementasjonen av arkitekturen på eininga nRF52833.



Figur 1: Implementasjon av SWD-grensesnitt på nRF52833[21]

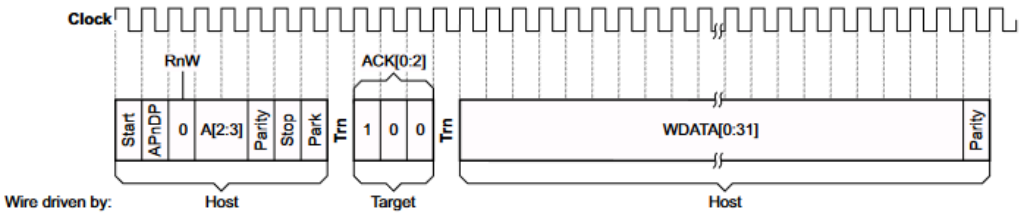
3.1.2 Protokoll og Kommunikasjon

SWD brukar ein eigen protokoll for seriell kommunikasjon. Dataoverføring gjennomførast synkront med bruk av SWDCLK styrt av kontrolleren. Kvar enkelt overføring av data brytast ned i tre steg:

1. Pakkeførespurnad (Packet Request): Kontroller sender ein førespurnad om overføring til målet
2. Annerkjenning av førespurnad (Acknowledge): Målet responderar med
3. Dataoverføring (Data Transfer phase): Ein 32-bit datapakke overførast i retning angitt i førespurnad og overføringa fullførast med paritetsbit for datapakken

Det må være ein mellomperiode der ingen av eininga kontrollerar datalinja før linja skifter retning, denne perioden kallast for TRN. Dette forhindrar konflikter på datalinja. Figur 2 syner ein gjennomført skrive-operasjon for protokollen.
Ein ser her at førespurnaden inneheld all relevant informasjon om dataoverføringa som skal finne stad. Den angjer om det skal skrivast eller lesast(RnW), om ein ynskjer tilgong til DP eller AP(APnDP), og registeradressa til transaksjonen(A). Ein kan basert på enkeltbit inndele protokollen i fire sentrale operasjonar for SWD-kommunikasjon, den einaste variabelen som gjenstår å definere er då registeraddressa for operasjonen.

- Skriv til AP
- Les frå AP
- Skriv til DP
- Les frå DP



Figur 2: Suksessful SWD skrive-operasjon[6]

3.2 HTTP-server

Ein server er eit program som gjennomførar tenester for andre einingar, eller klientar[1]. Vidare er ein HTTP- eller nettserver ein type server som nyttar førespurnadar og responsar saman med HTTP-protokollen. Serveren tilgjengleggjer ressursar som filer. Klienten bruker førehandsdefinerte kommandoar og sender ein førespurnad, serveren tolkar innhaldet i førespurnaden og svarar med ein respons.

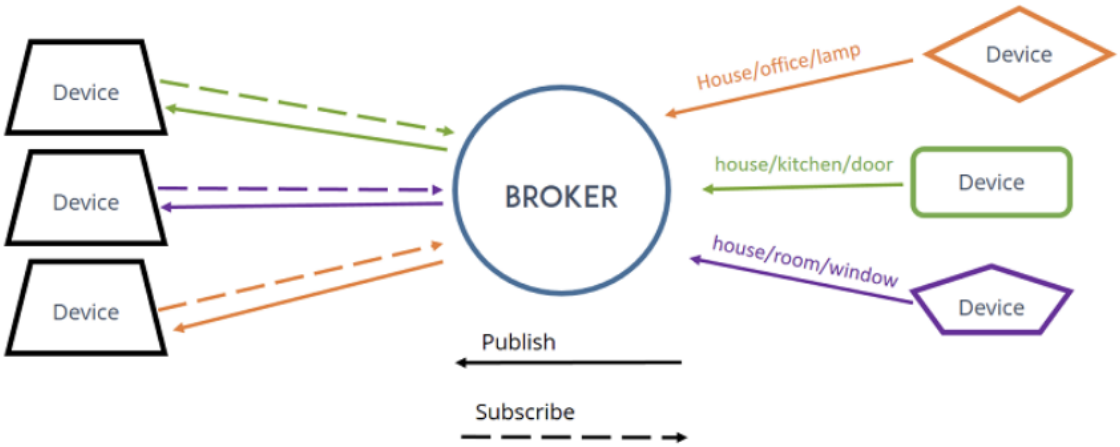
Vanlege HTTP metodar for førespurnadar[16]:

- **GET**: Metoden brukast for å spør om tilgong til ein ressurs. Målet er her å motta data
- **HEAD**: Metoden er ein form for GET, men ein mottar her berre headeren til ressursen utan medfølgande data
- **POST**: Metoden er brukt for å laste opp data til serveren, gjennom å oppdatere ein ressurs som allereie er til stades

- **PUT**: Metoden er brukt for å erstatte ein ressurs til stades på serveren, men følgande data.
- **DELETE**: Metoden brukast til å slett ein spesifikk ressurs på serveren
- **CONNECT**: Metoden brukast til å etablere ein to-vegs kommunikasjonstunnel til serveren
- **OPTIONS**: Metoden brukast til å definere kommunikasjonsalternativ
- **TRACE**: Metoden brukast til å gjennomføre ein loop-back test langs vegen til ynskja ressurs
- **PATCH**: Metoden brukast til å gjennomføre delvise modifikasjonar på ein ressurs

3.3 MQTT

Message Queueing Telemetry Transport(MQTT) er eit lettvektig meldingssytem designa for å kunne handtere låg bandbreidde og ustabile nettverk, dette gjer den eigna for IoT-applikasjon[27]. Systemet baserer seg på at ein kan abonnere på eller publisere meldingar til ulike "topics". Hjernen til systemet er ein MQTT-server, som tar i mot, filtrerar og sender medlingar til klientar som er tilkopla. Figur:3 syner korleis ein server mottek data frå ulike klientar og vidareførar dataen til andre klientar igjen, all kommunikasjon går gjennom serveren. Klientane bestemmer sjølv kva for topics dei vil abonnere på eller publisere til, men ein kan avgrense tilgongen til ulike topics for ulike nodar på server-sida.

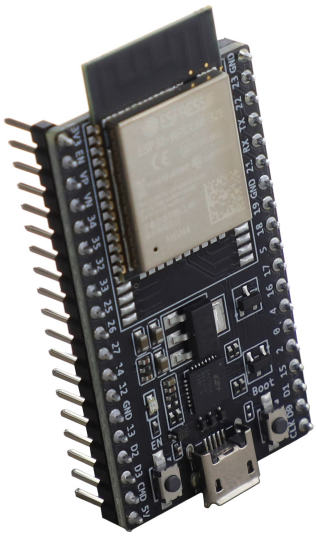


Figur 3: MQTT-server kontrollerar dataflyt til klientar med topics[27]

3.4 Komponenter

3.4.1 ESP32 Development Board

ESP32 er ein serie med mikrokontrollerer utvikla av Espressif[14][28]. Dei er populære gitt sin låge innkjøpskostnad, lågt straumforbruk og påbygd kapabilitet for WiFi og Bluetooth. Vidare er mikrokontrollerene kompatible med Arduino C++ og MicroPython som er vanlege startpunkt for mange. Det er vidare mogleg å kjøpe fleire ferdiginnstallerte ESP32 mikrokontrollerer på utviklarbrett, eller Development Boards. Eit typisk slikt utviklarbrett er vist i Figur 4. Utviklarbrett kjem som regel med alt ein trenger av komponent får å programmere eininga, tilkople den til datamaskin, og bruke WiFi-modulen. Eit utviklarbrett er derfor å føretrekke over enkeltkontroller for arbeidsoppgåver som testing og prototyping.



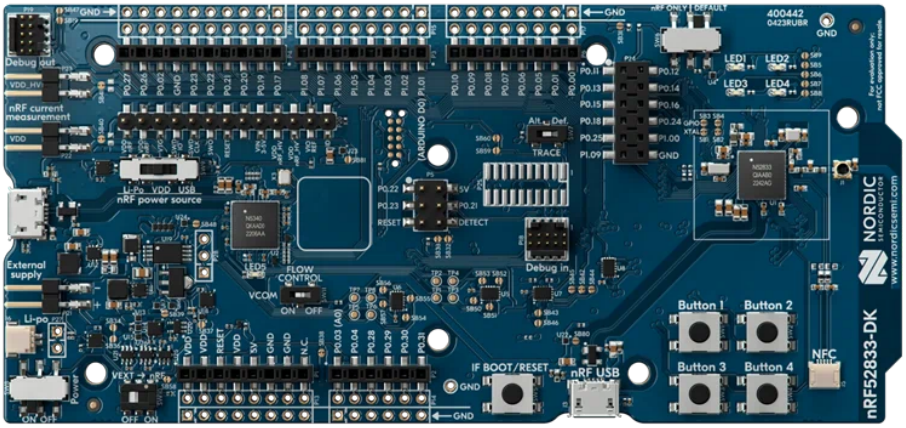
Figur 4: ESP32 Development Board[10]

3.4.2 nRF52833 Development Kit

nRF52833 er eit System On a Chip(SoC) eigna for generelle formål med ein integrert Bluetooth Radio[19]. Brikka er bygd rundt ein 64MHz ARM Cortex-M4, og har i minsteforpakning 512kB med flashminne og 128kB RAM-minne tilgjengleg.

Ein kan i likheit med ESP32 kjøpe ein versjon a denne eininga på eit førehandsinnstallert utviklarbrett[24]. Utviklarbrettet har i tillegg til nRF-eininga, tilkoplingspunkt tilpassa Arduino standard, tilkoplingspunkt for debugger, innebygd LED og knappekrinsar og

USB tilkopling for programmering, sjå Figur 5. Utvikling, bygging og debugging av applikasjonar kan vidare gjerast med det integrerte utviklingsverktøyet nRF Connect for VS Code[20]. Der utviklarbrettet er vedlagt som ei førehandsdefinert eining.



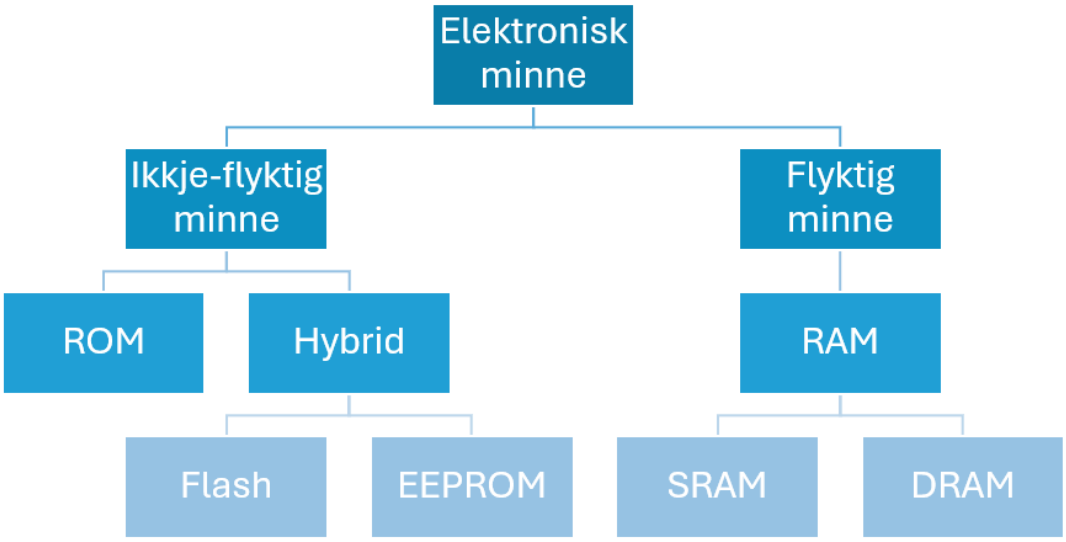
Figur 5: nRF52833 DK[24]

3.5 Elektronisk Minne

Elektronisk minne kan bli inndelt i to hovudkategoriar basert på korleis dei lagrar informasjon: varig minne og flyktig minne[17], Figur: 6.

Det varige minnet er karakterisert av at det tek vare på data sjølv ved eit straumbrot. Det er derfor typisk brukt til lagring av fastvare, verdiar som trengst ved oppstart og konstante verdiar. I mikroelektronikk er ofte denne minnetypen brukt til å handtere lagring av fastvare, konfigurasjonsdata og andre kritiske variablar som ein ynskjer å behalde i tilfelle eit straumbrot. Vidare kan varig minne inndelast i omskrivbart og ikkje-omskrivbart minne. Omskrivbare varige minner, òg kalla hybrid-minner, er nyttige då dei brukast til å lagre fastvare, som er mogleg å redigere eller oppdatere sjølv etter produksjon. Det er derimot viktig å merke at sjølv omskrivbare varige minner er avgrensa i mengda omskrivingar som kan gjennomførast i løpet av produktets levetid, til dømes har nRF52833 garanti opptil 10'000 omskrivngar [23]. To vanlege typar omskrivbare minner er Flash og EEPROM.

Flyktige minner går òg under det engelske tilnamnet RAM(Random Access Memory). I kontrast med varige minner vil flyktige minner miste all data ved eit straumbrot. Den interne oppbygginga til flyktige minner gjer skrive- og lesehastigheita til minnet generelt raskare enn det varige minnet. Flyktig minne er derfor ofte brukt til å handtere data som må kunne hentast eller skrivast hurtig. I dag eksisterar fleire typar flyktig minne, men i hovudsak skilar vi mellom statisk og dynasmisk flyktig minne.



Figur 6: Sentrale elektroniske minnetypar

4 Material og Metode

4.1 Planlegging

I samband med både forprosjekt og endringsnotat til forprosjekt, blei det gjennomført skissering av arbeidsoppgåver, tidsestimat og krav til ynskja produkt, endringsnotat er tilgjengleg som vedlegg. Under blir hovudpunkta frå endringsnotatet skildra.

4.1.1 Arbeidsoppgåver og tidsplanlegging

Prosjektet skulle i utgongspunktet gjennomførast med ei gruppe med fire medlem, men blei overført til ei oppgåve med eit medlem etter den 15.03.2024. Ved overføring til enkeltoppgåve blei dokumenta "Endringsnotat til forprosjekt" og "Arbeidsnotat" skapt for å omorganisere, dokumentere overførtbart arbeid og definere det nye omfanget til oppgåva. I samband med utvikling av endringsnotatet blei planlegging av det nye prosjektetomfanget gjennomført. Her blei arbeidsoppgåver, framdriftsplan og mål for prosjektet definert. Framdriftsplanen er illustrert som eit Gannt-diagram avhengig av arbeidsoppgåver. Endringsnotat, arbeidsnotat, framdriftsplan i form av Gannt-diagram og resulterande timeliste er lagt ved som vedlegg.

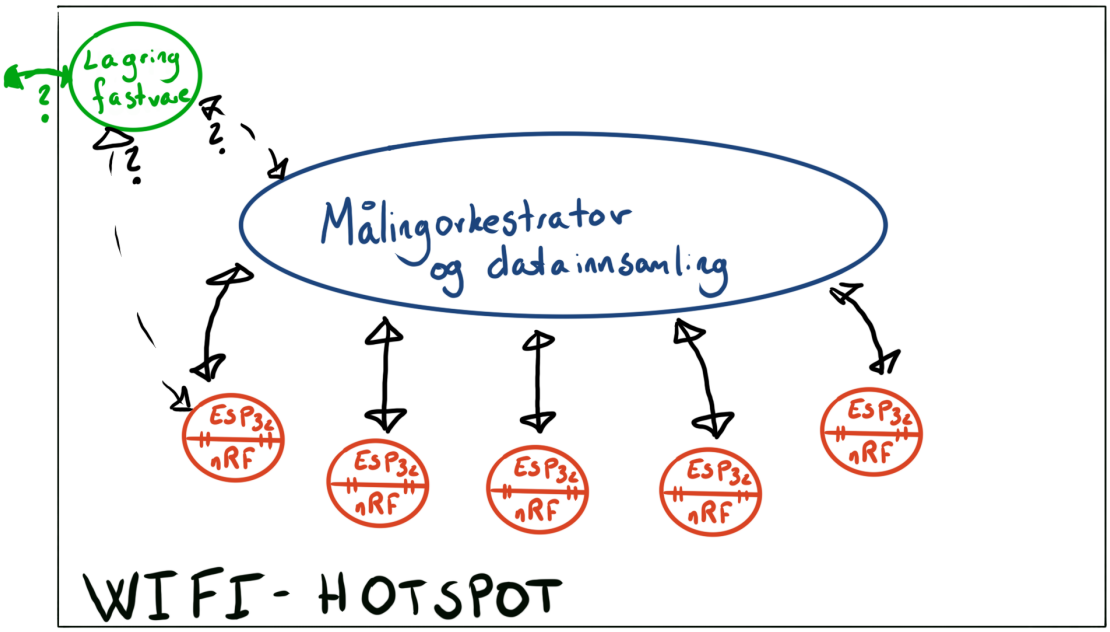
4.1.2 Arbeidsprosess

Framdriftsplanen illustrert som Gannt diagram er inndelt etter arbeidskategoriane: administrasjonsarbeid, prototyping og nettverk. Administrasjonsarbeid dekker jamnlege statusoppdateringsmøter med veileder, skriving av tovekersrapportar, andre dokument og produksjon av prosjektrapport. Prototyping innebærar alle stega for å skape ein debugger som kan endre fastvaren til eit nRF 52833 dk. Nettverk er vidareutbygging som lar debuggeren gjennomføre oppgåver som ein funksjonell IoT-node. Framdriftsplanen innebærte å linært gjennomføre oppgåver innan prototyping og deretter nettverk, mens administrasjonsarbeid gjennomførast parallelt etter behov med skift til fullt fokus på prosjektrapport etter fullføring av dei førenevnte kategoriane.

For å organisere arbeidet undervegs har fleire vektøy blitt teke i bruk. Gannt diagrammet fungerar vidare som ein arbeidslogg der ein kan markere ferdige oppgåver. I tillegg til framdriftsplanen dette blei vektøya **GitHub** og **Discord** brukt for å stadfeste progresjon. GitHub har blitt brukt til versjonskontroll og for å laste opp kildekode ved ferdigstilling av relevante milepælar i utviklinga. Plattformen Discord har blitt brukt til å loggføre mindre oppdateringar og spørsmål, lagre kodesnuttar, kjelder og relevante skjermbilete.

4.1.3 Systemplan

Basert på prosjektkrav vart det i endringsnotatet skissert ein systemplan som skildrar hovudtanken bak prosjektet, denne er vist i Figur: 9. Figuren syner at hovudtanken er å bruke ESP32 både til fastvareoppdatering av ein nRF og til dataoverføring og trådlaus kommunikasjon i eit IoT-system. Denne fungerer som rettleiar for systemutviklinga, sjølv om hovudarbeidet i prosjektet er å skape ein ESP32-basert debugger.



Figur 7: Tidlig skisse av systemdiagram

4.2 Komponentval

4.2.1 Delliste - Bill of Materials

Komponent	Skildring	Mengde
ESP-WROOM32-DEVKITV1	ESP32 med utviklerbrett	1
nRF52833 DK (PCA10100)	nRF 52833 med utviklerbrett	1
Analog Discovery 2	Multifunksjonelt måleintrument brukt som straumforsyning og logikkanalysator	1
Personleg Datamaskin(PC)	Standard 64-bit Windows datamaskin	1
Mikro-USB 2.0 kabel	USB A til Mikro-USB	2
SOP24PIN	Kretskort for overføring mellom TCSD og ESP32 pinner	1
TCSD-05-D-12.00-01-N(2X5)	Flatkabel som passer Debug IN på nRF 52833 DK	1

Table 1: BOM for prosjektarbeidet

4.2.2 Utstyr frå Nordic Semiconductor

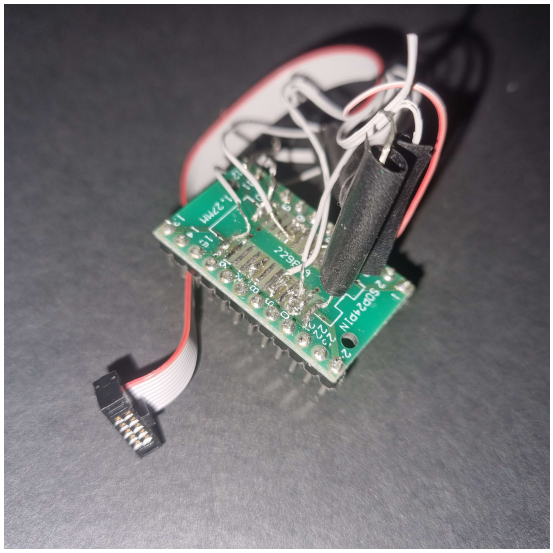
Ved oppstart av oppgåva vart prosjektmedlem tildelt nRF 52833 Development Kit. Dette grunna tilgjengelegheit og planlagd vidareutvikling av MaDChaSE prosjektet, som trenger støtte til kanalmåling. Målet er å kunne oppdatere fastvaren på ein slik eining utan å måtte gjennomføre fysisk oppkobling av datamaskin.

4.2.3 Val av mikrokontrollar

Ved oppstart av enkeltprosjektet med endringsnotat måtte ein mikrokontroller velgast som kunne fungere som debugger for nRF-eininga. Fleire aspekt blei vurdert i dette valet. Mikrokontrollaren burde være eigna for IoT applikasjon, ha lågare prisklasse enn tidlegare løysing med Raspberry Pi 3+, være brukarvennleg i oppsett og vidareutvikling. Den måtte òg ha nok lagringsplass til å handtere .bin filene som skal overførast til fastvare på nRF-eininga.
Valet vart eit ESP WROOM-32 Development Board. Den spesifikke produktkoden er delvis utfasa og vart vald grunna kort leveransetid og ein snarleg oppstart var naudsynt i høve endringsnotatet. Produktet ESP32 er godt dokumentert og det er mogleg å overføre program frå modell til modell. Mikrokontrollaren ESP WROOM-32 har innebygd støtte for WiFi og Bluetooth brukt til IoT applikasjonar, støttar Arduino-programmeringsspråk og inneheld 4MB flash som kan oppgraderast. Tilsvarande eining med utviklingskort har ein kostnad på ~110kr, hjå leverandør DigiKey [10].

4.2.4 Debugger-adapter

Einiga nRF 52833dk er utstyrt med ein debugger kontakt av typen ARM CoreSight 10 Interface[22][13]. For å kople ESP32 utviklarkortet til denne treng vi ein overgang. I dette tilfellet vart tilgjengeligheit og raskt oppstart av prototyping prioritert. Derfor blir ein flatkabel som passar kontakten splitta opp og leidningane lodda på eit kretskort eigna for å kople ein SOP til pinnar med 1.00 mm diameter, vist i Figur:8.. Ved høve for lengre bestillingstid ville nok eit meir eigna ferdigprodusert adapter vore nytta.



Figur 8: Sjølvprodusert overgang mellom debugger kontakt of ESP32

4.2.5 Logikkanalysator

Logikkanalysator brukt i prosjektet er Analog Discovery 2 med den tilhøyrande programvara Waveforms. Dette verktøyet vart tildelt på utlån frå NTNU.

4.3 Testmetodikk

Testane i prosjektet er praktiske målsjekkar som demonstrerer ulike funksjonalitetar i systemet. Gitt den lagvise oppbygginga til ESP32 biblioteka har det vært eit naturleg samanfall mellom testane og funksjonaliteten til kvart bibliotek.

4.3.1 Test av SWD debugger

Ein viktig grunnstein i prosjektet er å kunne gjennomføre dei fire grunnleggande SWD-operasjonane: skriv til AP/DP og les av AP/DP. Prosjektet bygger på desse SWD-

operasjonane gjennom å lenke dei saman i større system, og det er derfor avgjerande at dei blir tolka som tiltenkt. Gitt samanfallet mellom funksjonane sin verknadsmåte blir ein test av registeravlesing gjennomført.

Returnering av ID-CODE registeret
Korrekt returnering av innhaldet i ID-registeret demonstrerer at debuggeren korrekt kan starte opp SWD-grensesnittet, sende ein førespurnad, handtere mellomfasane, motta korrekt ACK-respons og deretter lese av innkommande data med paritetssjekk. ID-CODE registeret er ein del av Debug Port og er fabrikkinnstilt. Den skal derfor alltid returnere same svar. Ved korrekt svar viser ein dermed at ein kan gjennomføre avlesing av Debug Port. For å gjennomføre denne testen kan ein anten bruke ei kjede av andre funksjonar, eller bruke oppstartsfunksjonen `begin()` funnet i swd_interface

4.3.2 Test av nRF grensesnitt

nRF grensesnittet bygger på SWD debuggeren og sender større mengder data, som fort kan bli uoversiktleg i ein logikkanalysator. Derfor brukar ein kode som kan gje eit visuelt inntrykk for å indikere suksess. Det er her gjennomført to testar for funksjonalitet.

Endre fastvara til nRF
For å demonstrere at ein kan endre fastvara til nRF-eininga, blir ein sekvens der ein bytter mellom å skrive eit visuelt verfiserbar program og å slette nRF-eininga sitt innhald fastminne ved gitte intervall.
Det visuelt verifiserbare programmet er ved dette høvet ein versjon av eksempelkoden "blinky" funnet og bygd i Connect SDK. Deretter blir .bin fila til programmet overført til ESP32 gjennom lån av HTTP-nedlasting funnet i trådlaus grensesnitt, deretter køyrer sekvensen med skifting mellom sletting og skriving av fastvare. Koden vist i Kodesnutt 1 skildrar korleis ein kan bruke biblioteket nrf_interface til å gjennomføre sekvensen og skrive fastvare til nRF eininga.

```
void loop(){
    nrf.begin_if();
    nrf.flash_write_from_file("/firmware.bin", 0);
    nrf.end_if();
    Serial.println("Blinky is running");
    delay(5000);
    nrf.begin_if();
    nrf.ctrl_erase_all();
    nrf.end_if();
    Serial.println("nRF is erased");
    delay(5000);
}
```

Kodesnutt 1: Manipulering av nRF-fastvare

Gjennomføre ikkje-forstyrrande lesing av data

Ikkje-forstyrrande datavlesing er ein teknikk utvikla spesielt for dette prosjektet. På nRF-einingа blir programmet 'non_interfering_meas' førehandsinstallert. Dette programmet skal fungere i samspel med funksjonen: `measurement()` frå nrf_interface biblioteket. Kvar gong ei måling blir gjennomførd vil programmet skrive resultatet av fem kvadrattal til register i området 0x2000100-0x200110. Deretter skal ESP32 ved korrekt køyring kunne lese av dei gitte registera og tolke data. Kodesnutt 2 syner oppsettet brukt på ESP32 sida for å gjennomføre testen.

```
void loop(){
    uint32_t temp[5] = {0};
    nrf.measurement(temp, 0x2000100, 5);
    for(int i = 0; i < 5; i++){
        Serial.println(temp[i]);
    }
    delay(1000);
}
```

Kodesnutt 2: Gjennomføre ikkje-forstyrrande måling

4.3.3 Test av trådlaus kommunikasjon

Trådlaus kommunikasjon inneberer kommunikasjon gjennom MQTT og nedlasting av fastvare via HTTP. For å verifisere at desse funksjonane fungerer blir to testar gjennomført.

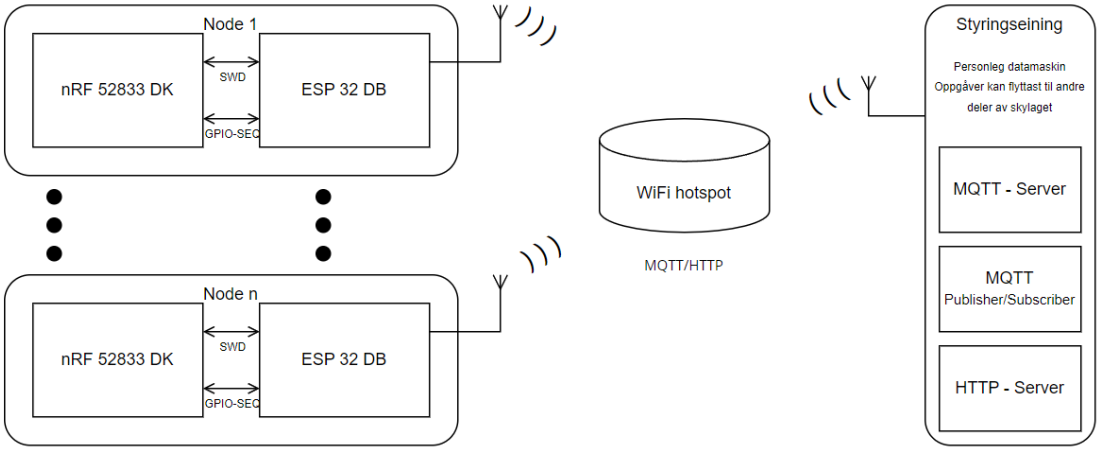
Verifisering av HTTP-nedlasting
ESP32-biblioteket HTTPClient handterer sjølv feil i HTTP-kommunikasjon, men det er naudsynt å verifisere at metoden fungerer. HTTP-server vert sett opp lokalt på ei personleg datamaskin ved hjelp av den innebygde Python3 funksjonen `http.server`. Dersom ESP32 er tilkopla ein Seriell Monitor vil det ved suksessfull nedlasting skrivast ei melding som syner storleiken på den nedlasta fila, dette vært samanlikna med filstørrelsen angitt i HTTP-serveren. Vidare kan ein bruke visuelt verifiserbare program. Ein lastar fyrst ned eit program med ein gitt blinkefrekvens og skriv det til det ikkje-flyktige minnet, deretter lastat ein ned eit program med ein annleis eller ingen blinkefrekvens og skriv dette til det ikkje-flyktige minnet. Dette syner at ulike program kan lastast ned korrekt og skrivast til nRF-einingа ved hjelp av debuggeren.

MQTT kommunikasjon
MQTT-kommunikasjon er testa gjennom å sette opp ein MQTT-server av typen Mosquitto på ei datamaskin. Dette programmet vert sett opp slik at tilkopla einingar og overførte meldingar kontinuerleg vert synt i kommandovindauget. Ein kan her kontrollere at program- og meldingsflyten er som forventa. MQTT-kommunikasjon er i fleire tilfelle brukt for å kontrollere einingane i dei andre testane, og ein observerer gjennom desse kva for funksjonalitetar som fungerer som tiltenkt.

5 Implementasjon

5.1 Overordna arkitektur

Endestrukturen for prosjektet liknar i stor grad på den orginale skissa vist i Figur 7, og er skildra i Figur 9. Ein ser her at det blir nytta eit lokalt trådlaust nettverk for å kople saman ei ugitt mengde nodar og ei styringseining med fleire underprogram. Nodane er sett saman av eit ESP WROOM32 Development Board og eit nRF 52833 DK. Dei to einingane er kopla saman på to måtar. Gjennom tilkoplingspunktet for debuggere, som nyttar SWD-grensesnittet. Og gjennom bruk av to GPIO-tilkoplingar, som nyttar ein original sekvens for å handtere målingsorkestrering. Vidare er ESP32 einingane kopla til det trådlause nettverket og kommunisera med styringseininga ved hjelp av protokollane HTTP og MQTT. Styringseininga køyrer derfor samtidig program for å vedlikehalde ein MQTT-server, ein HTTP-server og abonnering og publisering til MQTT topics.



Figur 9: Systemarkitektur

5.2 ESP32 basert debugger

5.2.1 Programvare ESP32

Programvare for ESP32 er skapt for å kunne implementerast med utviklingsplattformen Arduino IDE, som er ein begynnarvennleg plattform med stor brukarbase. Programvaren har ein grunnstruktur med tre bibliotek som bygger på kvarandre. Kvar bibliotek legg trinnvis til funksjonalitet, dette er inspirert av eit liknande prosjekt [9]. Dette er gjort for å tillate brukar å bytte ut eller forkaste delar av prosjektet som ikkje er naudsynt å inkludere i ulike brukstifelle, i tillegg til å "skjule" delar av prosjektet som ikkje er i bruk. Biblioteka er vidare organisert som klassar med tilhøyrande private og offentlege

funksjonar for å gjere bruk og endring brukarvennleg mogleg [4][5]. Bruken av klassar gjer det vidare enkelt å inkludere føreliggande bibliotek i neste trinn og meir intuitivt for brukar å organisere eigen kode. For å tilrettelegge for vidare endring eller bruk er alle klassar, funksjonar og parameterer forklart og redegjort for i kommentarar på engelsk. Biblioteka, installasjonsretningslinjer og brukseksempel kan finnast under Vedlegg B: Kildekode.

SWD-Grensesnitt
SWD-grensesnittet fungerar som grunnsteinen i prosjektet. Dette biblioteket handterar det grunnleggande i SWD-protokollen. Dette er å sende førespurnadar, handtere trn-fasar, motta ACK, dataoverføring og paritetkontroll. For å gjennomføre funksjonane kontroller biblioteket vidare linjene SWDIO og SWDCLK sin logiske status. Basert på dette er det skapt fem offentlege funksjonar i biblioteket:

1. begin() - Startar SWD tilkopling og returnerar ID-CODE registeret
2. DP_read() - Leser av eit register frå ei gitt addresse i Debug Port
3. DP_write() - Skriver data til eit register med gitt addresse i Debug Port
4. AP_read() - Leser av eit gitt register frå ei gitt addresse i Access Port
5. AP_write() - Skriver data til eit register med gitt addresse i Access Port

Det er skild mellom funksjonar for Debug Port og Access Port for å gjere det enklare å aktivt velje riktig funksjon ved meir avansert bruk, som i nRF-grensesnitt.

For å ta i bruk biblioteket må ein fyrst inkludere .h fila til biblioteket. Deretter skape eit objekt av klassen, med fire instansvariablar: pinnenummer for SWDCLK, SWDIO, periodetid for klokkesignalet, og boolen debug angir om Serielle feilmeldingar skal sendast. Eit kort eksempel som viser to metodar for å hente ut ID-CODE registeret er vist i Kodesnutt 3

```
#include "swd.h"

#define pin_SWDIO 19
#define pin_SWDCLK 21
#define clockperiod 5
#define debug true
uint32_t temp = 0;

SWD_IF swd(pin_SWDCLK, pin_SWDIO, clockperiod, debug);

void setup(){
    Serial.begin(115200);
}
void loop(){
    //Both print-statements should return the same answer
```

```
    temp = swd.begin();
    Serial.println(temp);
    swd.DP_read(temp, 0x00)
    Serial.println(temp);
    delay(2000);
}
```

Kodesnutt 3: Enkel bruk av SWD-grensesnitt biblioteket

nRF-Grensesnitt

nRF-grensesnittet bygger på SWD-grensesnittet gjennom å bruke dei grunnleggande funksjonane i større kjeder og operasjonar, og inkluderar derfor eit objekt av typen `SWD_IF` ved oppstart. Biblioteket fungerar som eit grensesnitt mellom einingane ESP32 og nRF52833. Grensesnittet handterar å gjennomføre kjernestopp og -start, sletting av fastvare, skrive til fastvare frå fil, avlesing av fastvare til fil, og den spesialbygde sekvensen ikkje-forstyrrande måling ved hjelp av funksjonar frå SWD_IF.

Eit eksempel på struktur og oppbygging innad i biblioteket er kommandoen `begin_if()` som er vist i Kodesnutt 4. Her brukast funksjonar frå SWD_IF for å samanlikne ID-CODE med standard for nRF52 einingar, deretter brukast DP-registeret til å skru på system og debugger-grensesnitt internt. Kjerna stoppast og fabrikkinformasjon avlesast, som vidare er funksjonar basert på SWD_IF. For å gjere vidare arbeid enklare returnerar funksjonen ein bool som verifiserar tilkoplingstilstanden og ein kan skru på serielle feilmeldingar.

Biblioteket er spesielt tilpassa eininga nRF52833 DK, men kan sannsynlegvis brukast med andre einingar i familien nRF52 og tilpassast andre einingar som har SWD-grensesnitt. Sjølv om nokre av funksjonane ikkje er teke i bruk av det trådlause-grensesnittet

```
/*
 * Starts up an SWD-session, checks the DP-ID register
 * and requests startup of debug and system
 */
bool NRF_IF::begin_if(){
  _swd.init();
  nrf_ficr.core_id = _swd.begin();
  if(nrf_ficr.core_id == 0x2ba01477){ //Standard Core-ID for nRF
   52 devices
    if(_debug){
      Serial.print("Connected to device of type nRF52");
    }
    uint32_t temp;
    _swd.DP_write(0x0, DP_SELECT);       //Choose AP0
    _swd.DP_write(0x50000000, DP_CTRL); //Request powerup
    _swd.DP_read(temp, DP_CTRL);         //Read ack for request
    _swd.AP_write(0x23000052, MEM_AP_CSW);//Read info on MEM-AP
    core_halt();
    read_info();
    return true;
```

```
  }
  else{
    if(_debug){
      Serial.print("Unable to connect to device");
    }
    return false;
  }
}
```

Kodesnutt 4: Funksjonen begin_if() frå biblioteket nRF-grensesnitt

Trådlaust-Grensesnitt
Det trådlause grensesnittet bruker funksjonane skapt i nRF-grensesnittet og legger til trådlaus kommunikasjon i form av MQTT og HTTP for å kontrollere oppgåver og dataflyt. Det er eit poeng her at mest mogleg på ESP32 sida skal fungere med automatikk, alt skal kunne kontrollerast frå ekstern styringseining. Dette biblioteket har derfor berre to offentlege funksjonar: `interface_init()` og `interface_run()`.
Funksjonen `interface_init()` skal i eit standard Arduino program køyrast i `void setup()` og initialiserar pinner, trådlaust internett tilkopling, og tilkopling til MQTT server. Vidare brukast funksjonen `interface_run()` til å vedlikehalde tilkoplingane, kontrollere ulike oppgåver og kontrollere den forstyrrande funksjonen callback som brukast til å motta MQTT-meldingar. Denne plasserast typisk i `void loop()` i eit standard Arduino program.

5.2.2 Programvare nRF52833

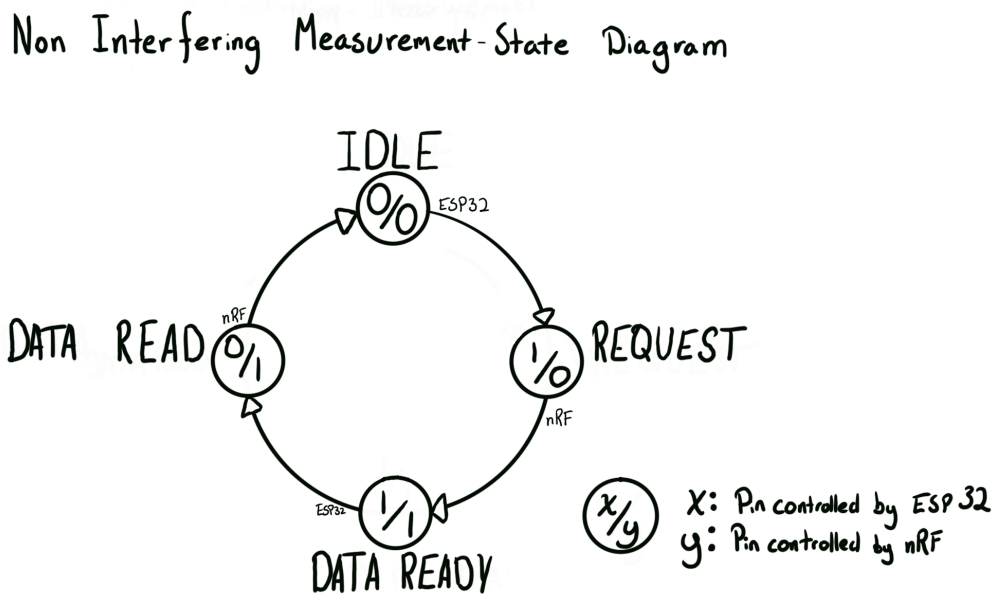
For nRF-eininga er det lagd tre testprogram for køyring. To er redigerte versjonar av eksempelprogrammet `blinky`, og den siste er ein test av den ikkje-forstyrrande målingssekvensen som produserar varierte men forutsigbare verdiar.
Dei to versjonane av blinky har satt ulike frekvensar for å visuelt kontrollere at oppdatering av fastvare fungerar. Programmet tilhøyrande ikkje-forstyrrande måling er satt saman av tre delar: sekvenskontroll for måling, produksjon og minneplassering av verdiar, og ein oppstartsfunksjon for å skru av nRF sin innebygde `APPROTECT`-funksjonalitet i programvare [30][8].
Programkode og resulterande binærfiler er vedlagt som kildekode. Alle filer er produsert i NRF Connect for VS Code, SDK v2.5.0. Konfigurasjon for bygging er standard, med vald brett: nrf52833dk_nrf52833.

5.2.3 Ikkje-forstyrrande måling

Ikkje-forstyrrande måling er ein førehandsdefinert sekvens som krev programmvare både på nRF52833 og ESP32 sida. Målet er her å bruke to GPIO-tilkoplingar for å skape ein sekvens der: ESP32 ber om å få hente data, nRF52833 gjennomførar måling, ESP32 leser av data direkte gjennom nRF-grensesnittet, og tilkopling avsluttast. Eit engelsk

tilstandsdiagram blei skapt i samband med Vedlegg B: Kildekode, sjå Figur: 10. Figuren viser den logiske tilstanden til dei to GPIO-pinnane brukt, ein kontrollert av ESP32 og ein kontrollert av nRF52833. Ein ser at det skiljast mellom fire tilstandar som må gjennomførast i sekvens.

1. IDLE (0/0): Ingen av dei logiske pinnane har tilstand 1
2. REQUEST (1/0): ESP32 har aktivert sin pinne og nRF kan gjennomføre måling i denne tilstanden.
3. DATA READY(1/1): nRF har fullført måling og ESP32 kan lese av data frå SRAM i denne tilstanden.
4. DATA READ(0/1): ESP32 har gjennomført avlesing av data og venter til nRF er klar, satt tilstand 0, før neste avlesing



Figur 10: Tilstandsdiagram for ikkje-forstyrrande måling

5.3 IoT-system

IoT-systemet skapt i prosjektet har som mål å både være mogleg å sette opp enkelt for utprøving, men vidare være skalerbart og justerbart for meir avansert bruk. Det er derfor brukt programvare og protokollar som kan køyre på ei personleg datamaskin.

5.3.1 HTTP-server oppsett og kommunikasjon

HTTP-serveren er brukt i prosjektet for å tilgjengleggjere fastvarefiler som skal lastast ned av dei ulike nodane. HTTP-serveren er sett opp med den innebygde Python3 funksjonen `http.server`. Kodesnutt 5, visar korleis ein kan starte ein slik server ved hjelp av kommandovindauget. HTTP-serveren vil då være tilgjengleg gjennom datamaskina si IP-adresse på det lokale nettverket med port 8000.

```
PS C:\> cd path\to\directory\with\soucecode
PC C:\yourpath> python -m http.server 8000
```

Kodesnutt 5: Oppsett av HTTP-server med Python3

5.3.2 MQTT-server oppsett og kommunikasjon

MQTT-serveren er sett opp med programmet Mosquitto versjon 2.0.18 for Windows. Kodesnutt 6 syner korleis ein kan sette opp serveren ved hjelp av eit kommandovindauge. Anonym kommunikasjon er her tillat for å forhindre potensielle feil ved autentisering for ein fyrstegongsbrukar, men det er potensial for å sette opp meir avansert autensitering som gjev ulike tilkopla einingar ulike roller. Vidare er Mosquitto nytta til å publisere og motta meldingar over MQTT. I biblioteket er det førehandsdefinert ei rekke topics og kommandoar som ESP32 lyttar etter og publiserer til. Eksempel på køyring er vedlagt som videomateriale, i tillegg skildrar Figur 14 og Figur 13 under resultat to eksempel på MQTT-køyring frå personleg datamaskin.
Topics brukt for å kontrollere dataflyt:

- command: Meldinga angir ein kommando ESP32 skal gjennomføre.
- offset: Meldinga blir sendt frå styreeining og blir omdanna til eit tal som endrar den globale variabelen offset.
- length: Meldinga blir sendt frå styreeining og blir omdanna til eit tal som endrar den globale variabelen span.
- url: Meldingen frå styreeining inneheld ei nettadresse der fastvarekode kan lastast ned.
- ack: ESP32 sender statusrapporteringsmeldingar her.
- data: ESP32 sender avlest data frå ikkje-forstyrrande måling her.

Kommandoar ESP32 kan gjennomføre:

- FW-Flash Write: Skriv noværande lagra fil på ESP32 til fastvara til nRF, bruker variabelen `offset` for å avgjere startaddressa for skriving.

- FR-Flash Read: Leser av fastvareminnet til nRF og lagrar det til fil på ESP32. Brukar variablane `offset` og `span` for å avgjere startadresse for avlesing og mengde data som skal avlesast
- DF-Download File: ESP32 lastar ned fil frå angitt nettadresse i den globale variabelen `url` gjennom HTTP GET.
- UF-Upload File: ESP32 lastar opp den lokalt lagra fila til nettaddresse angitt i variabelen `url` gjennom HTTP POST.
- ME-Measurement: Gjennomfører ei ikkje-forstyrrande måling, nyttar variablane `offset` og `span` for å avgjere startadresse for avlesing og mengde verdiar som skal avlesast
- ER-Erase all: Sletter all data i minnet

```
PS C:\> cd yourpath\Program Files\mosquitto
PS C:\mosquitto> notepad mosquitto.conf
#Add these lines to the .conf file
listener 1883
allow_anonymous true
#Save your changes and close
PS C:\mosquitto> mosquitto.exe -c mosquitto.conf -v
```

Kodesnutt 6: Start av MQTT-server med Mosquitto

6 Resultat

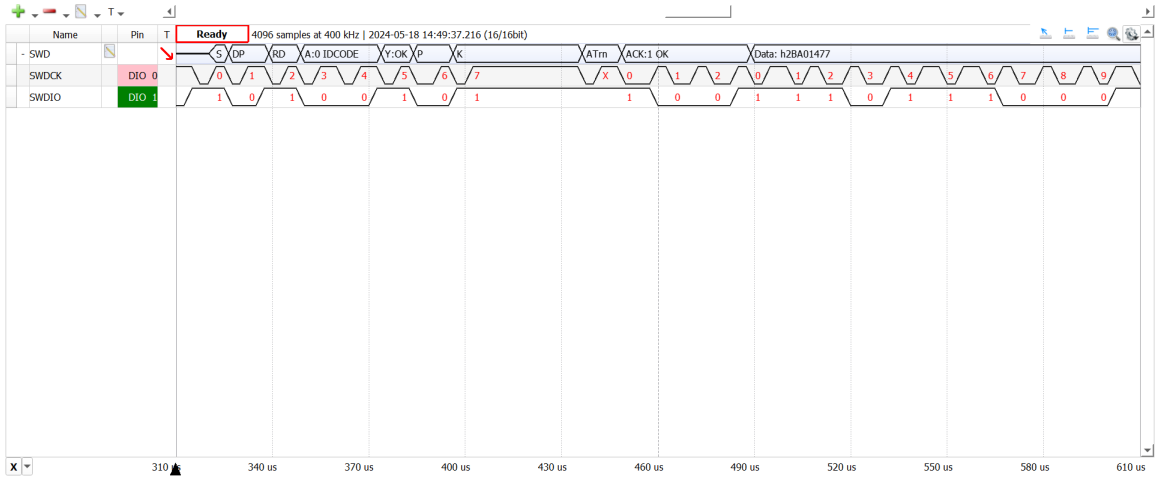
6.1 Testresultat

Testresultata er delt inn i underkapittel "SWD-debugger", "nRF-grensesnitt" og "Trådlaus kommunikasjon" for betre oversikt.

6.1.1 SWD debugger

Returnering av ID-CODE registeret
ID-CODE registeret skal i nRF52 familien innehalde 0x2BA01477. Det er observert at ESP32 eininga klarer å lese av denne dataen og logge det til Seriell Monitor, vist i Figur: 11. Figuren skildrar programmet som køyrer på ESP32-eininga saman med resultat logga til Seriell Monitor som viser den heksadesimale verdien 0x2BA1477. Transaksjonen blir vidare observert med logikkanalysator som synar samsvarande resultat som tidlegare. Figur: 12, synar utklipp av den fullstendige transaksjonen der ID-CODE registeret avlesast.

Figur 11: Enkelt program som kan returnere ID-CODE registeret frå nRF ESP32

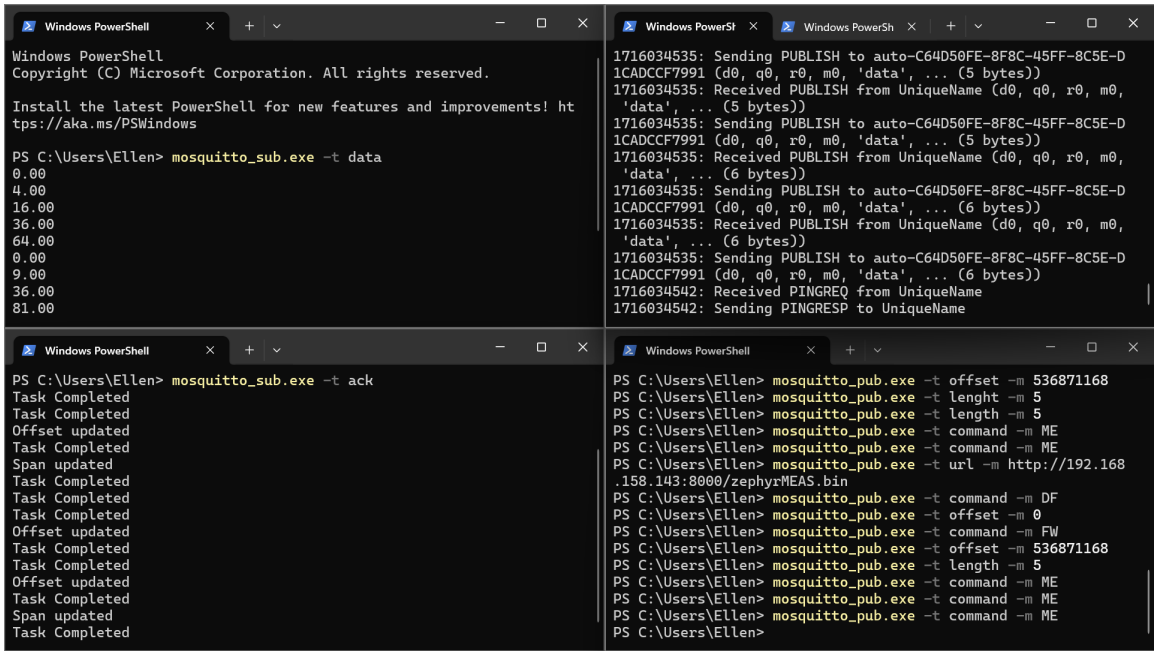


Figur 12: Utklipp frå logikkanalysator viser DP avlesning av ID-CODE registeret

6.1.2 NRF grensesnitt

Endre fastvara til nRF
Denne testen er i praksis berre visuelt verifiserbar. Eksempelet vist i Kodesnutt 1 blir køyrd utan endringar, men med tillegg for å laste ned programvare. Det blir observert at nRF-eininga oppfører seg som forventa i tilfellet. Dette visast vidare i videoen "Fastvare-oppdatering med MQTT" under Vedlegg G: Videomateriale.

Ikkje-forstyrrande avlesing av data
Denne testen blir gjennomført saman med trådlaus test. I praksis vil dette sei at Kodesnutt:2, er integrert som ein funksjon i det trådlause grensesnittet, og at resultatet blir returnert over MQTT i staden for logga med Seriell Monitor. Lengda av avlesinga og startadressa for avlesning er sett som variablar som kan redigerast over MQTT. Figur: 13, syner resultatet frå avlesning av data viser at forventa verdiar er avlest frå forventa addresser.

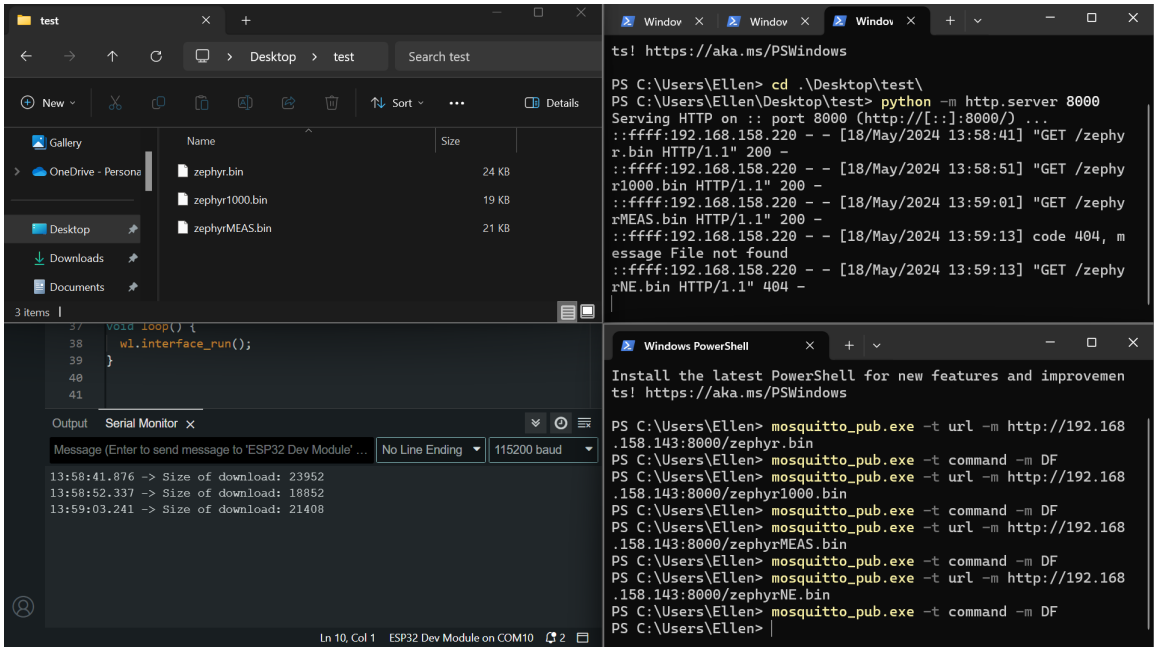


Figur 13: MQTT-kommandoar og resultat for ikkje-forstyrrande avlesing av data

6.1.3 Trådlaust grensesnitt

Verifisering av HTTP nedlasting

For å verifisere at det er mogleg for ESP32-eininga å laste ned fastvarefiler korrekt frå ein HTTP-server, vert det sett opp ein lokal HTTP-server med tre førebudde prøvefiler. I tillegg settast MQTT-systemet opp for å kunne bruke kommandoer for kontroll. Figur 14 syner filene gjort tilgjengled på HTTP-serveren, driftsmeldingar på serveren, ESP32 seriell informasjon og MQTT kommandoar køyrt. Ein ser at filstorleiken angitt i Seriell Monitor samsvarar med filstorleiken angitt i HTTP-serveren. Vidare observerar ein at HTTP-serveren gjev suksessfull respons når nettadressa er angitt korrekt, medan respons for klientfeil er angitt når fila som skal nås ikkje er tilstades. Prøvefiler er lagt ved som vedlegg.



Figur 14: MQTT-kommandoer og resultat for ikkje-forstyrrande avlesning av data

Test av MQTT-kommunikasjon
MQTT-kommunikasjon blir brukt til å handtere testane for HTTP og ikkje-forstyrrande overføring av data. Ein bruker derfor resultata vist i Figur 13 og Figur 14 som grunnlag for testresultat innan MQTT kommunikasjon. I løpet av testane har alle topics og kommandoer fungert som skildra, forruten kommandoen "Upload File". HTTP-serveren brukt i arbeidet støttar ikkje metoden HTTP-POST, og det var derfor ikkje høve for å fullføre denne funksjonaliteten. Under kildekode er derfor denne funksjonaliteten kommentert ut og er **ikkje** del av endeleg produkt.

6.2 Rekneskap innkjøp

Det er gjennomført to omtrentlege rekneskap for innkjøpspris av noder samansett av fabrikklare komponenter, basert på bulkprisar henta frå leverandørane DigiKey og ELFA [12][11]. Ein for systemet utvikla i prosjektet, vist i Tabell 2. Det andre for eit tilsvarande system der Raspberry Pi koplast til nRF-eininga for å gjennomføre fastvareoppdateringar. vist i Tabell 3. Resultatet visar ein prisskilnad på ~180NOK per node.

Komponent	**Delnummer**	**Stk**	**Pris[kr]**
ESP32 DB	ESP32-DEVKITC-32D	1	110
nRF52833DK	NRF52833-DK	1	660
Mikro-USB kabel	Qualtech 3025034-10	2	50
SWD overgang	ARM-JTAG-20-10	1	60
		Total	**930**

Table 2: Prisestimat for utvikla node

Komponent	**Delnummer**	**Stk**	**Pris[kr]**
Raspberry Pi 3	SC0022	1	400
nRF52833DK	NRF52833-DK	1	660
Mikro-USB kabel	Qualtech 3025034-10	1	50
		Total	**1110**

Table 3: Prisestimat for tilsvarande node

7 Drøfting

7.1 Vurdering av resultat

Testresultata og videovedlegg syner at det er mogleg å gjennomføre fastvareoppdatering trådlaust og organisert gjennom bruk av MQTT-kommandoar. På den andre sida kan organiseringa av fastvareoppdatering forbetrast. Gjeldande løysing for MQTT-server skil ikkje mellom ulike nodar i nettverket, som vil bety at alle nodane vil prøve å gjennomføre oppdatering samstundes. Dette kan føre til overbelasting i nettverket og vidare feil i nedlasting av ny fastvare.

Vidare viser reknskapen at systemet er økonomisk mindre ressurskrevjande i innkjøpspris enn ei tilsvarande løysing med Raspberry Pi, som er ein positiv indikator på at prosjektet oppfyller kravet om å være mindre ressurskrevjande enn tilsvarande alternativ. Det er derimot usikkert kva for løysing som i lengda vil krevje minst ressursar i form av tid. Oppsettstid, tilpassing av systemet, feilsøking og utvikling av nye tillegg er arbeidsoppgåver som vidare må vurderast. Tid er ein viktig økonomisk ressurs i mange utviklingsprosjekt og dette kan påverke endeleg økonomisk resultat betrakteleg.

Systemet brukar i dag komponentar skildra i dellista. Dette er eit ESP WROOM32 DB, nRF52833 DK, ein overgong for å kople saman debugger-kontakten og ESP32-pinner, og ei personleg datamaskin. Frå komponentlista er overgong frå debugger-kontakt til ESP32-pinner spesielt vanskeleg å få tak i ferdigprodusert. Løysinga var i implementasjonen å sjølv lodde saman ein overgong. I rekneskapen er eit ferdigprodusert alternativ framstilt, men dette er ikkje kontrollert for kvalitet, bestillingstid eller potensial for større innkjøpsmengde.

Vidare vert det vurdert at systemet kan settast opp av ein gjennomsnittleg ingeniørstudent. Kildekode til prosjektet er skrive i språka C og C++, og er der mogleg forma etter industristandard, i tillegg er eit større kommenteringsarbeid gjennomført for å gjere det enkelt å orientere seg i kildekoden. Saman med kildekoden på GitHub er det skrive eit omfattande oppsettsnotat på engelsk som forklarar korleis systemet er satt opp og kva det gjer. For å bruke systemet dette prosjektet har produsert vil det være naudsynt å kunne orientere seg i kode skrive C\C++, og ein bør være kjend med utviklingsplattformane Arduino IDE og nRF Connect SDK. Dette er vurdert som kunnskap ein gjennomsnittleg ingeniørstudent kan tileigne seg.

Systemet brukar i dag filer av typen .bin for å overføre og laste opp fastvare. Desse blir produsert når ein i dag bygger ein applikasjon med nRF Connect SDK for VS Code. Fila kan som regel direkte hentast under `ApplikasjonsMappe\builds\zephyr\zephyr.bin`. Det er fullt mogleg, og gjerne naudsynt, å bruke verktøykjeda til Nordic Semiconductor for å produsere ny fastvare, men ein bør vurdere å utforme ein funksjon på ESP32-sida som sjølv kan konvertere ei full .hex fil til binør data .

Det følgjer at systemet oppfyller dei gjevne minstekrava, og vidare oppfyller resultatmålet til prosjektet. Det er derimot fortsett noko underliggande problematikk som kan være til hinder for å oppnå det gitte effektmålet. Ein kan derfor kalle dagens implementasjon ein versjon Alpha, implementasjonen *kan* brukast av nodar i prosjektet MaDChaSE, men det bør gjennomførast forbetringar før eventuell full integrasjon.

7.2 Potensial for vidareutvikling

Det er fleire ulike aspekt ved løysinga som kan forbetrast og utviklast. Under er to potensielle utviklingsområde skildra med problemstillingar saman med idéar for løysing.

7.2.1 Oppsett av bedre trådlaus kommunikasjon

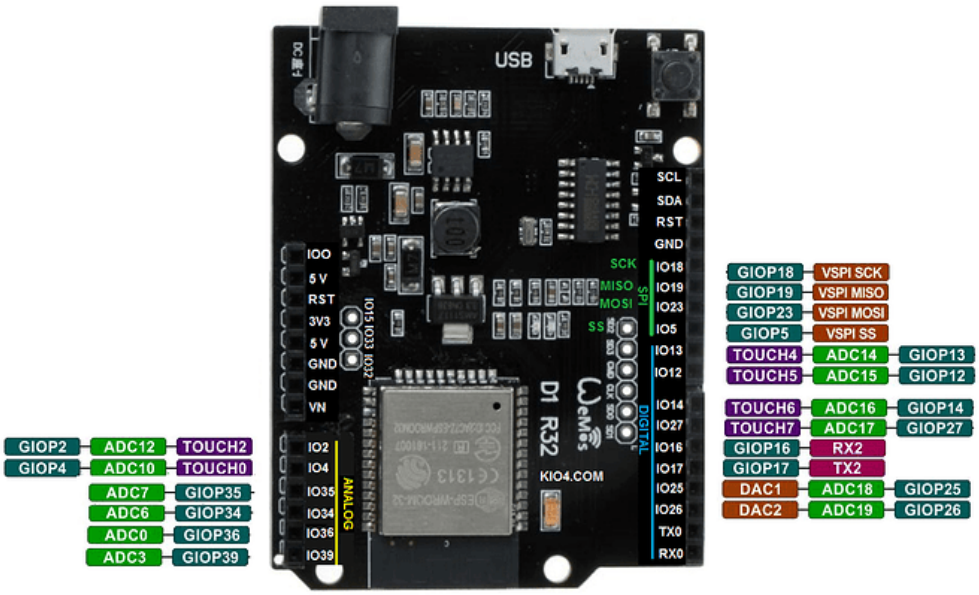
Sikkerheit og kryptering
Sånn systemet er sett opp no er det manglar innan sikkerheit og kryptering, både innan MQTT- og HTTP-kommunikasjon. Sidan MQTT-serveren tillét anonyme einingar å kople til er enkelt for uynskja einingar å kople til og avlytte transaksjonar eller potensielt sende kommandoar som er skadelege for programflyten. MQTT er vidare i grunn overført i rein tekst, utan noko form for kryptering, sjølv utan tilkopling er det mogleg å avskjere datapakkane som overførast. Det er fleire metodar for å betre sikre MQTT-kommunikasjonen i systemet. Ein kan mellom anna implementere autentisering av klientar, ulike autoritasjonslag, kryptering av dataoverføring, isolering av IoT-nettverket og strengare kontroll over tilkopla einingar[7][15].
HTTP-serveren er vidare eit svakt punkt i kommunikasjonskjeda. Igjen blir data overført i rein tekst utan noko form for kryptering. Ein bør derfor starte med å bytte over til HTTPS-server med innebygd kryptering gjennom bruk av sertifikat som endrast ved gitte intervall. Det er fleire måtar vidare sikre ein HTTPS-server og ein kan bruke verktøy som for SSL Labs sin SSL test kontrollere sikkerheitskarakteren til serveren[2].
Vurdering av kva for steg som bør implementerast for å sikre dei trådlause kommunikasjonskanalane i systemet bør gjennomførast for kvart prosjekt. Dette vil avhenge av: isolasjon av nettverket, talet nodar som skal koplast saman, implementasjon av monitorering og sensitivtetsnivået på dataen som blir overført m.m.

Handtering av endring av IP-addresser
Systemet som er bygd no inkluderer ingen form for opprettingsmetode dersom MQTT-serveren eller ved endring av SSID eller passord til det trådlause nettverket. Det er her ynskjeleg å bygge eit system som er sjølvopprettande ved endringar i desse variablane. Ein kan til dømes sette opp eit nettverkssystem som nyttar MAC-filtering for tilkopling [3]. Her trenger ikkje klienten noko anna informasjon enn SSIDen "NTNU-IOT" for tilkopling, tilgang blir kontrollert på nettverkssida. Vidare er det mogleg å køyre ein Domain Name System-server på eit lokalt nettverk. Ein slik server lar deg kontrollere ulike domenenamn og kva dei førar til i nettverket. Ein kan derfor gje opp eit statisk domenenamn til alle

relevante klientar og heller endre IP-addressene på DNS-sida.

7.2.2 Forbetring av oppkopling av sensornode

Tilpassa ESP32 til nRF52833 DK
nRF52833DK har allereie pinnerader som er tilpassa tilkopling einingar som følgjer Arduino Standard[20]. nRF52833DK kan derfor fungere som eit skjold for einingar som har denne arkitekturen. Det er derfor mogleg å legge ein ESP32 modul på eit tilpassa kretskort som følgjer Arduino standard, som for eksempel vist i Figur 15. Ein kan vidare forlenge kretskortet og inkludere tilkopling til Debug Input på nRF52833DK. Eit kretskort tilpassa på denne måten vil forenkle utvikling, oppsett og straumforsyning.



Figur 15: ESP32 modul på Arduino Standard kretskort

Felles straumforsyning
Implementasjonen av prosjektet dekkjer ikkje problemstillinga rundt felles straumforsyning for dei to einingane som samankopla for å danne noden. Begge einingar støttar fleire metodar for straumforsyning, men det må tilpassast for straumtrekk for begge einingar. Dersom ein til dømes implementererar å bruke nRF52833DK som eit skjold for eit spesiallaga ESP32 kretskort kan ein bruke tilpasse ein spenningsregulator på kretskortet og kople til nRF gjennom VDD og GND utgangane. Ellers vil det for testing være tilstrekkeleg å bruke eit veggadapter med to separate mikro-USB kablar til kvar eining.

7.3 Konsekvens av arbeidet

Systemet skapt i prosjektet er funksjonelt og visar positive resultat. Ein kan gjennom trådlaus kommunikasjon og vidare SWD-kommunikasjon oppdatere fastvare og innhenting av målingsdata. Prototypen syner gjennom desse resultata potensial for vidare utvikling og integrering med prosjektet MaDChaSE. Gjennom integrering med prosjektet MaDChaSE kan ein bygge eit større nodenettverk, med definerte roller, betre dataflyt og automatisering av kommunikasjonen. Ein kan vidare bruke den trådlause kommunikasjonen til å orkestrere målingar. Prosjektet vil i eit slikt tilfelle gå frå å være ein enkel manuelt kontrollerbar prototype, til å potensielt være eit fullt automatisert IoT-nettverk. Effektiv innsamling av data gjennom eit IoT-systemet kan potensielt bidra til å kunne "sjå" verda gjennom bruk av kanalmålingar[31].

7.3.1 Bærekraft og bærekraftsmål

Dette prosjektet rettar seg i hovudsak mot bærekraftsmålet "Industri, Innovasjon og Infrastruktur"[26]. Bærekraftsmålet inneberer mellom anna å utvikle infrastruktur, styrke forsking og utvikling av teknologi. Prosjektet er enno i ein prototypefase, men planen med å integrere prosjektet under MaDChaSE inneberer at proskjektet potensielt bidreg til vidare forsking på metoden kanalmåling og i eit lengre perspektiv kunne nyttast til å sette opp infrastruktur for kanalmåling. På den andre sida er det viktig å merke at det ikkje er gjort studier eller rekneskap som vuderer om oppsett av denne infrastrukturen vil være kunne gjennomførast på ein bærekraftig måte. Det er difor per i dag umogleg å konkludere fast om prosjektet vil ha positiv eller negativ utslagseffekt under dette eller andre bærekraftsmål. Vidare utvikling av prosjektet vil potensielt utgreie og tydeleggjere desse effektane. For å avgjere om nyutvikla teknologi er bærekraftig kan ein sjå på fleire typiske markørar: vurdering av materialtyper og -kostnad, kontroll av potensiell levetid, måling av straumtrekk, moglegheit for resirkulering og utslepp kopla til produksjon og transport[29].

8 Konklusjon

Føremålet med prosjektet var å skape verktøy for å distribuere fastvare til ei eining av typen nRF52833 DK. Denne distribusjonen av fastvare skal vidare ha potensial til å bli brukt innan det større prosjektet MaDChaSE.

Prosjektet har produsert ein SWD-debugger tilpassa den tildelte komponenten nRF52833 DK, tilhøyrande IoT-system og system for å vidare kunne nytte SWD-debuggeren til bruk i avlesing av data. Løysinga som presentert dekkar i stor grad prosjektkrava skildra i innleiinga, men manglar å gjennomføre ein større systemtest med fleire nodar.

Det er vidare potensial for vidareutvikling av fleire funksjonalitetar, samansetting med prosjektet MaDChaSE, sikring av IoT kommunikasjon og forenkle oppsettsprosessen. For å mogleggjere vidareutvikling har prosjektet blitt publisert offentleg på sida GitHub og rettleiiar for installasjon er vedlagt på engelsk.

Løysinga prosjektet presenterer er derfor samla vurdert til å være eigna for funksjonelle testar av ei mindre samling noder, til dømes. Prosjektet er ein positiv indikator for moglegheita til å skape eit større IoT nettverk med fleire nodar som kan gjennomføre fastvareoppdatering, men vidareutvikling bør gjennomførast. Spesielt bør automatisering, identifisering av noder og sikring av IoT kommunikasjon prioriterast før prosjektet eventuelt integrerast i MaDChaSE, som i prinsippet skal kunne handtere fleire hundre nodar i eit nettverk.

9 Referanseliste

[1] "An overview of http." Henta: 19. mai 2024, mdn web docs. (2024), [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview.

[2] K. Annoh. "Http vs https – what's the difference?" Henta: 20. mai 2024. (2022), [Online]. Available: https://www.freecodecamp.org/news/http-vs-https/.

[3] K. Annoh. "Mac filtering: What is it and should i enable it?" Henta: 21. mai 2024. (2023), [Online]. Available: https://www.whatismyip.com/mac-filtering/.

[4] Arduino.cc, *Writing a library for arduino*, Henta: 19. mai 2024, 2023. [Online]. Available: https://docs.arduino.cc/learn/contributions/arduino-creating-library-guide/.

[5] Arduino.cc, *Arduino style guide for creating libraries*, Henta: 19. mai 2024, 2024. [Online]. Available: https://docs.arduino.cc/learn/contributions/arduino-library-style-guide/.

[6] *Arm debug interface: Architecture specification*, 5th, Henta: 06. april 2024, 10 Fulbourn Road Cambridge, England CB1 9NJ, 2006. [Online]. Available: https://developer.arm.com/documentation/ihi0031/a.

[7] L. Ballejos. "How to secure iot devices: 5 best practices." Henta: 10. mai 2024. (2024), [Online]. Available: https://www.ninjaone.com/blog/how-to-secure-iot-devices-5-best-practices/.

[8] V. Berg, *Response: Trying to disable approtect for debugging*, Henta: 21. mai 2024, 2022. [Online]. Available: https://devzone.nordicsemi.com/f/nordic-q-a/85671/trying-to-disable-approtect-for-debugging.

[9] A. Christophel, *Github repository: Esp32$_n$rf52$_s$wd*, Henta: 19. mai 2024, 2023. [Online]. Available: https://github.com/atc1441/ESP32_nRF52_SWD/tree/main.

[10] DigiKey. "Esp32-devkitc-32e." Henta: 21. mai 2024. (2024), [Online]. Available: https://www.digikey.no/en/products/detail/espressif-systems/ESP32-DEVKITC-32E/12091810.

[11] DigiKey. "Searchable products." Henta: 18. mai 2024. (2024), [Online]. Available: https://www.digikey.no/en/products.

[12] E. Distrelec. "Søkbare produkter." Henta: 18. mai 2024. (2024), [Online]. Available: https://www.elfadistrelec.no/no/.

[13] "Documentation: Coresight 10." Henta: 20. mai 2024, ARM Developer. (2011), [Online]. Available: https://developer.arm.com/documentation/dui0499/d/ARM-DSTREAM-Target-Interface-Connections/CoreSight-10.

[14] "Esp32." Henta: 21. mai 2024, Espressif. (2024), [Online]. Available: https://www.espressif.com/en/products/socs/esp32#:~:text=ESP32%20is%20highly%2Dintegrated%20with,Hybrid%20Wi%2DFi%20%26%20Bluetooth%20Chip.

[15] HiveMQ, *Mqtt security fundamentals*, Henta: 20. mai 2024, 2024. [Online]. Available: https://www.hivemq.com/mqtt/mqtt-security-fundamentals/.

[16] "Http request methods." Henta: 21. mai 2024, Mozilla Corperation. (2023), [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods.

[17] B. B. Larsen, *Store norske leksikon: Mikrokontroller*, Henta: 9. april 2024, 2021. [Online]. Available: https://snl.no/mikrokontroller.

[18] J. N. S. Lojan, "Improving accuracy of mcpd distance measurements," M.S. thesis, Norwegian University of Science and Technology, 2023.

[19] "Nrf52833." Henta: 21. mai 2024, Nordic Semiconductor ASA. (2024), [Online]. Available: https://www.nordicsemi.com/Products/nRF52833.

[20] "Nrf52833 product specification: Connector interface." Henta: 20. mai 2024, Nordic Semiconductor ASA. (2020), [Online]. Available: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fug_nrf52833_dk%2FUG%2Fdk%2Fhw_debug_in_trace.html.

[21] "Nrf52833 product specification: Debug and trace." Henta: 20. mai 2024, Nordic Semiconductor ASA. (2020), [Online]. Available: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fps_nrf52833%2Fmemory.html.

[22] "Nrf52833 product specification: Memory." Henta: 20. mai 2024, Nordic Semiconductor ASA. (2020), [Online]. Available: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fps_nrf52833%2Fmemory.html.

[23] *Nrf52833dk objective product specification*, Nordic Semiconductor ASA, 2019. [Online]. Available: https://infocenter.nordicsemi.com/pdf/nRF52833_OPS_v0.7.pdf.

[24] "Product: Nrf52833dk." Henta: 20. mai 2024, Nordic Semiconductor ASA. (2024), [Online]. Available: https://www.nordicsemi.com/Products/Development-hardware/nRF52833-DK.

[25] R. M. Saltnes, *Presentation: Introduction to nordic distance toolbox*, Henta: 20. mai 2024, 2022. [Online]. Available: https://devzone.nordicsemi.com/cfs-file/__key/communityserver-discussions-components-files/4/Measuring_5F00_distance_5F00_with_5F00_Nordic_5F00_Distance_5F00_Toolbox_5F00_slides.pdf.

[26] FN-sambandet. "Bærekraftsmål: Industri, innovasjon og infrastruktur." Henta: 17. mai 2024. (2023), [Online]. Available: https://fn.no/om-fn/fns-baerekraftsmaal/industri-innovasjon-og-infrastruktur.

[27] R. Santos. "What is mqtt and how it works." Henta: 19. mai 2024, Random nerd Tutorials. (2021), [Online]. Available: https://randomnerdtutorials.com/what-is-mqtt-and-how-it-works/.

[28] R. Santos. "Getting started with the esp32 development board." Henta: 20. mai 2024. (2023), [Online]. Available: https://randomnerdtutorials.com/getting-started-with-esp32/.

[29] V. I. Solutions. "Kpis to measure the environmental impact of your technologyr." Henta: 17. mai 2024. (2023), [Online]. Available: https://virtu.net/5-kpis-measure-the-environmental-impact-of-technology/.

[30] D. Veilleux, *Working with the nrf52 series' improved approtect*, Henta: 21. mai 2024, 2022. [Online]. Available: https://devzone.nordicsemi.com/nordic/nordic-blog/b/blog/posts/working-with-the-nrf52-series-improved-approtect.

[31] C. Wulff, *Oppgaveforslag bacheloroppgave*, Blackboardressurs, Henta: 20. mai 2024, Norges teknisk-naturvitenskapelige universitet., 2024.

10 Vedlegg

Vedlegg A: Oppgåvetekst

NTNU
Institutt for elektroniske systemer

Oppgaveforslag bacheloroppgave elektroingeniør (BIELEKTRO) i Trondheim, vårsemester 2024

Navn bedrift: **NTNU / Nordic Semiconductors**	**Kontaktperson: Carsten Wulff** **Epost: carsten@wulff.no** **Telefon/mobil:**	
Tittel på oppgave: **Massive Distributed Channel Sounding Equipment**		
Hvilke studieretninger passer oppgaven for? (kryss av for alle aktuelle retninger; flervalg er mulig):	**Automatisering og robotikk**	☐
	Elektronikk og sensorsystemer	☒
	Elkraft og bærekraftig energi	☐
Er oppgaven reservert for noen bestemte studenter? (skriv navnene på studentene inn til høyre)	Eryk Bartlomiej Siejka <erykbs@stud.ntnu.no> Ellen Iren Johnsen <ellen.i.johnsen@ntnu.no> Krzysztof Karol Morylowski <krzysztm@stud.ntnu.no> Michal Stankiewicz <michsta@stud.ntnu.no>	
Har dere arbeidsplass for studentene	☐ ja ☐ nei ☒ usikker?	
Er dette en lukket oppgave? Dvs. at sluttrapporten ikke kan publiseres fordi den inneholder sensitiv informasjon.	☐ ja ☒ nei ☐ ikke enda bestemt	

Kort beskrivelse av oppgaven med problemstilling.

The process of measuring a physical communication channel is called "Channel Sounding". Where we used to need long cables, antennas, and a Vector Network Analyzer (at the price of a large car) the firmware for channel sounding now runs on nRF52833 development kits.

With 100's of devices observing the world at 2.4 GHz, what could we see? We know we can accurately measure distance between devices, but could we see the presence of walls, indoor location, whether there is people moving around. Could we see people breathe?

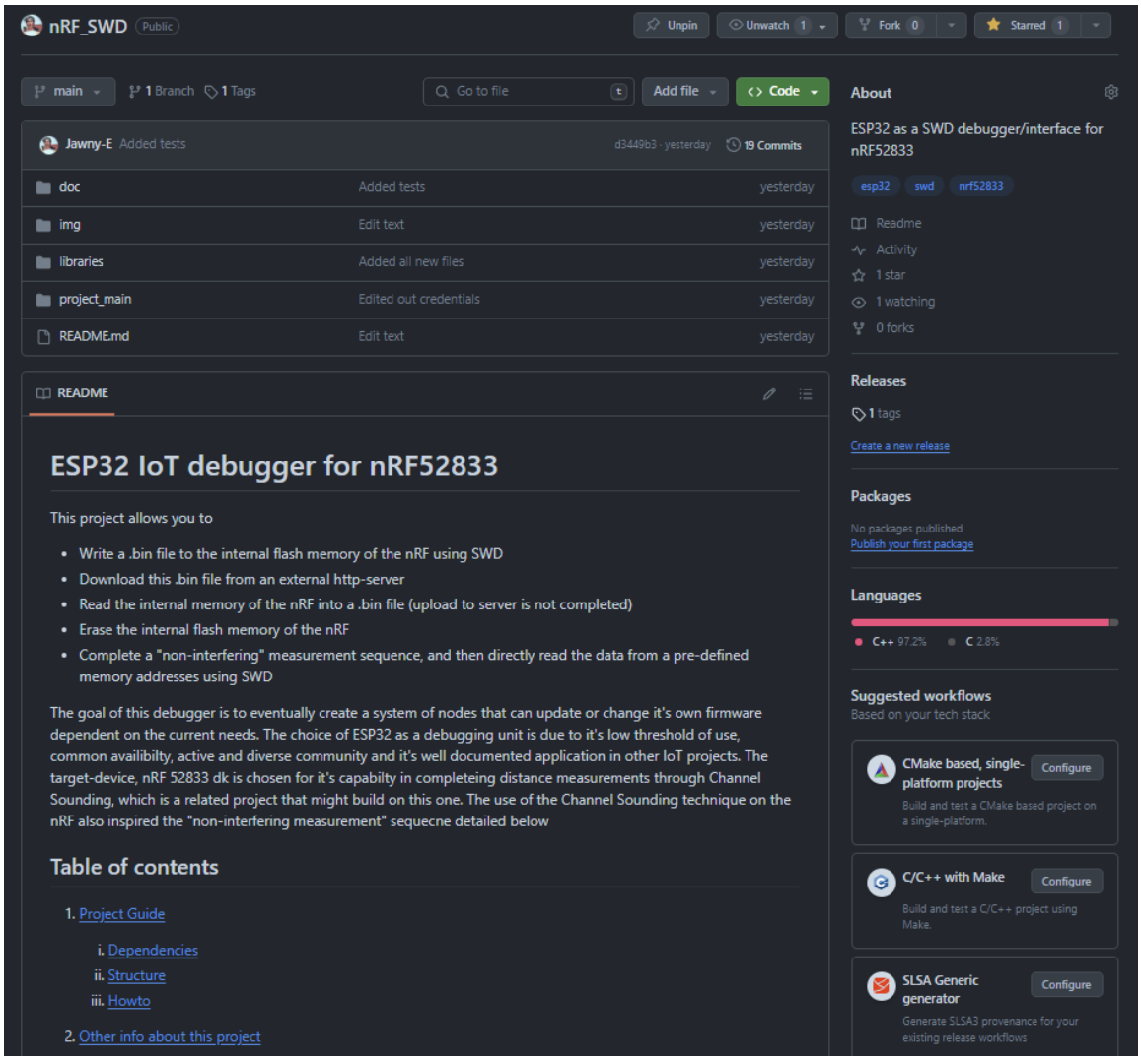
100's of devices in a network present challenges. How do we roll out firmware if the old firmware hard-faulted? How do we power devices? How do we collect data? How do we orchestrate measurement? How do we commission devices onto network? How do we give students access?

The assignment is to prototype and develop nodes, capable of channel sounding, that can be scaled to hundreds of nodes where students can have access to roll out firmware. The tasks could be:
- Prototype multiple nodes (for example, raspberryPI and nRF52833 DK running off AC)
- Create tools for deployment of firmware to nRF52833 DKs
- Create tools for commissioning of new nodes onto WiFi network
- Create tools for channel sounding measurement orchestration and data collection.

Vedlegg B: Kildekode

Kildekoden produsert i prosjektet er publisert på tjenesten GitHub, der det har blitt oppretta eit offentleg repository der ein kan finne ESP32 bibliotek, testkode for nRF, fullstendig prosjekt og retningslinjer for installasjon. Framsida til repositoriet visast i Figur: 16

https://github.com/Jawny-E/nRF_SWD



Figur 16: Framside GitHub

Vedlegg C: Endringsnotat forprosjekt

NTNU

Institutt for elektroniske systemer

Prosjektskildring:

MaDChaSE

Massive Distributed Channel Sounding Equipment -
Microcontroller based Firmware depolyment

Bacheloroppgave elektronikk og sensorsystemer
IELET2900
Trondheim, Vår 2024

Vedlegg C: Endringsnotat forprosjekt

Prosjektskildring 2024 NTNU

KANDIDATER: Johnsen, Ellen Iren			
DATO: 17.03.2024	**FAGKODE:** IELET2900	**PROSJEKT NR:** E2418	**SIDER/BILAG:** 10/ X
Veileder: Arne Morten Midjo			
TITTEL: Massive Distributed Channel Sounding Equipment			

2

Vedlegg C: Endringsnotat forprosjekt

Innhold

Vedlegg C: Endringsnotat forprosjekt

1 Innledning

1.1 Oppgaveteksten

Oppgåveteksten utdelt fra oppdragsgjevar omhandlar å prototype og utvikle noder som kan utføre kanalmåling. Denne utskilte delen av oppgaven skal spesielt fokusere på å skape billegare og brukarvennlege verktøy for distribusjon av fastvare til systemet. Dette arbeidet vil innvolvere:

- Skape verktøy for distribusjon av fastvare
- Skape verktøy for å legge til nye enheter til ett WiFi nettverk
- Skape verktøy for orkestrering av datainnsamling

1.2 Rapportens oppbygging

Vedlegg C: Endringsnotat forprosjekt

2 Teknisk del

2.1 Problemstilling

Kanalmåling er i dag en teknikk som brukest til å måle distansen mellom to enheter. Ofte involverer dette dyrt utstyr og teknisk personell til oppsett. Versjon 0.0x av systemet som skal skapast er samansett av ein Raspberry Pi 3 og eit nRF 52833 development kit. Målet med dette prosjektet er ideelt å komplett erstatte Raspberry Pien med ein meir lettvektig mikrokontroller, som vist i Figur 1.

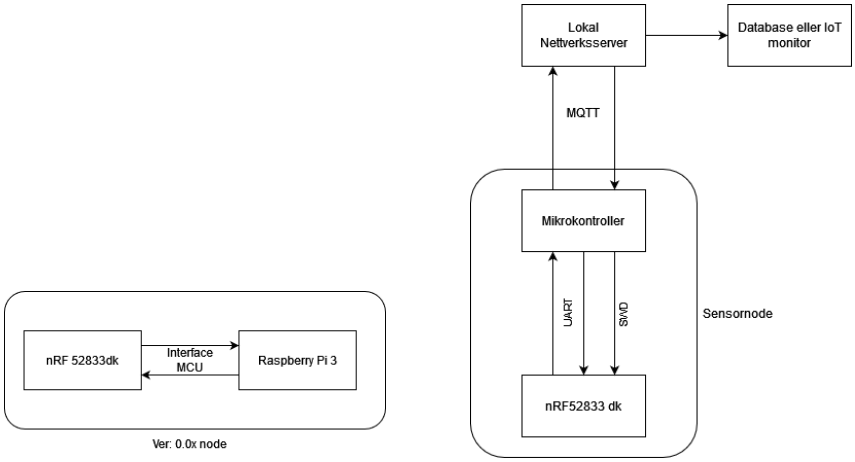


Figure 1: Systemet før og etter erstatting av mikrokontroller

2.2 Prosjektmål

Prosjektmåla skal tydeleg skildre kva utfallet av prosjektoppgåva skal være. Godt definerte og oppnåelege mål dannar eit godt fundament for vidare arbeid. Gitt omfanget til prosjektet er det hensynsmessig å skilje måla vidare inn i: effekt-, resultat- og prosessmål.

2.2.1 Effektmål

Effektmåla for prosjektet kan oppsummerast som "framtidige vidareutviklarar eller brukarar av MaDChaSE-prosjektet skal enklare kunne produsere eigne sensornodar og sette opp kommunikasjonssytem"

Vedlegg C: Endringsnotat forprosjekt

2.2.2 Resultatmål

Resultatmålet av prosjektet er at det skal produserast ein sensrnode som skildra i Figur: 1. Vidare er det ynskjeleg å levere eit utlegg for eit ferdig oppsatt IoT system med fleire fungerande sensornodar.

2.2.3 Prosessmål

Prosessmålet for prosjektet vil være at deltakar får brukt og vist tidlegare tileigna kunnskapar i tillegg til at prosjektet skal fungere som ein læringsmoglegheit innan: debuggerprotokollar, nRF 52 serien og mikrokontrollerstruktur.

2.3 Prosjektskildring

Prosjektet kan då delast inn i to hovudarbeid med tilhøyrande dokumentasjon. Fyrste hovudarbeid vil være å skape ein mikrokontrollerprototype som kan gjennomføre arrbeidsoppgåvene som tidlegare høyrde til Raspberry Pi. Andre hovudarbeid vil være å skape kommunikasjonsflyt mellom einingane i systemet på ein slik måte at prosjektet seinare vil være enkelt å ta i bruk. For dette prosjektet vil det å skape gjennomførd dokumentasjon være eit sentralt delmål. Spesielt gitt den seine oppstarten og omfanget til prosjektet vil det være viktig å dokumentere alt arbeid gjennomførd på ein slik måte at det er enkelt å vidareføre.

2.4 Problemområdar

Det største problemområdet identifisert av deltakar er å skaffe kompetanse innan programminnebehandling og debuggingsprotokollar. Usikkerheiten her kan ha store utslag for resten av prosjektgjennomføringa, og er derfor markert med "høg risiko" i tidsplanlegginga. For å vege opp dette har kandidaten konsekvent vald å bruke kjende teknologiar og metodar i andre områder av prosjektet, vidare leggast låg terskel for å kontakte arbeidsgjevar eller andre fagpersonar med spørsmål.

Vedlegg C: Endringsnotat forprosjekt

3 Arbeidspakker

3.1 Introduksjonspakker

3.1.1 Skape testkode for nRF 52833dk

For å teste om systemet fungerar er det nyttig å ha ulike typar testkode som kan overførast til nRF 52833. Det er her planlagd ein enkel led-blinkekode, ein kode som har hentar GPIO data og overførar det med liknande UART-meldingsstruktur som kanalmålingskoden, og mogleg endeleg kode som kan gjennomføre kanalmålingar

3.2 Prototypepakker

3.2.1 Vurder oppdateringsmetoder for fastvare

Gitt oppgåvestarten har kandidat i røynda valdt å ta utgongspunkt i SWD for å overføre fastvare til nRF52833, men det skal i dette steget utforskast nærmare korleis denne metoden gjennomførast, kva andre metodar innebærar, og planlegge korleis ein kan gå fram for å bygge eit SWD-grensesnitt på mikrokontroller

3.2.2 Val av mirkokontroller

Her er det i praksis to moglege alternativ gitt krav til: WiFi, tilgjenglegheit og kostnad: Raspberry Pi Pico W og ESP32. Begge alternativ har fordelar og baksider, men det er her viktig å nevne at kandidat har betydelg større erfaring med utvikling for ESP32. I løpet av denne korte arbeidspakken skal det vurderast om Rasperry Pi Pico har nokon fordelar over ESP32 som kan oppvege for differansen i kompetanse.

3.2.3 Skape SWD-prototype

Arbeidspakken dekkar å skape eit SWD-grensesnitt mellom ESP32 og nRF 52833 dk. Dette vil være ein hjørnestein i prosjektet og vidare arbeid med andre arbeidspakkar vil være tilnærma lik umogleg før dette blir gjennomført. Det skal her skapast verktøy som lar mikrokontrolleren laste ned programmeringsfil frå lokal server for å deretter bruke denne til å oppdatere fastvaren på nRF 52833 dk.

Vedlegg C: Endringsnotat forprosjekt

3.2.4 Sett opp mikrokontrollerkode som eget bibliotek

Etter ein SWD-prototype er skapt vil neste arbeidspakke være å omforme mikrokontrollerprogrammet til ein form for nedlastbart bibliotek som er enkelt å ta i bruk. Denne arbeidspakken vil være ei sentral brikke for vidareføring av prosjektet.

3.2.5 Verifiser at mikrokontrolleren fortsatt kan brukest til IoT formål

Denne arbeidspakken omfattar å sette saman SWD-prototypeprogrammet med tiltenkte kommunikasjonsprotokollar som vil kontrollere oppførselen til systemet. Mikrokontrolleren skal etter dette steget ha to "modusar": Fastvareoppdatering og IoT-node.

3.3 Nettverkspakker

3.3.1 Oppsett av lokal nettverksserver

For å gjøre det enkelt å redigere hvilke noder som er del av distansemålingsnettverket som videre skal utviklast vurderer kandidaten at systemet best organiserest av en hovedserver som de ulike sensornodene kan koble til og sende data til. Denne serveren kan vidare organisere og synkronisere fastvareoppdateringar til nodane.

3.3.2 Full systemtest med fleire nodar

Dette vil være den siste arbeidspakken gjennomført i dette prosjektet. Her skal fleire sensornodar settast opp og koplast til den lokale nettverksserveren. Systemet skal både klare å overføre data i målingsmodus og gjennomføre fastvareoppdateringer.

Vedlegg C: Endringsnotat forprosjekt

4 Tidsplan

For å styre prosjektet er det satt opp to hovudverktøy for tidsplanlegging, som ligger tilgjenleg i PA-permen. Ett timeestimat basert på normal arbeidsveke, som legger opp til ca. 50 arbeidstimar med bachelorarbeid per veke. Resultatet av estimatet er eit S-diagram som ser forhaldsvis linært ut gitt den korte tidsperioden, Figur: 2. Vidare blei ett Gantt-diagram produsert, inndelt arbeidspakkane som skal leverast. Diagrammet kunne tydlegare markert avhengigheitar, men sidan arbeidet blir utført av enkeltkandidat vil utviklinga i stor grad være linær som skildra, Figur: 3. Gitt denne utsatte oppstarten av prosjektet er det lite slingringsrom i tidsplanane. Det er derfor i tillegg produsert ei omfattande timeliste for å halde oversikt over arbeid som har blitt gjennomført og timar som er lagt ned, sjå Figur: 4.

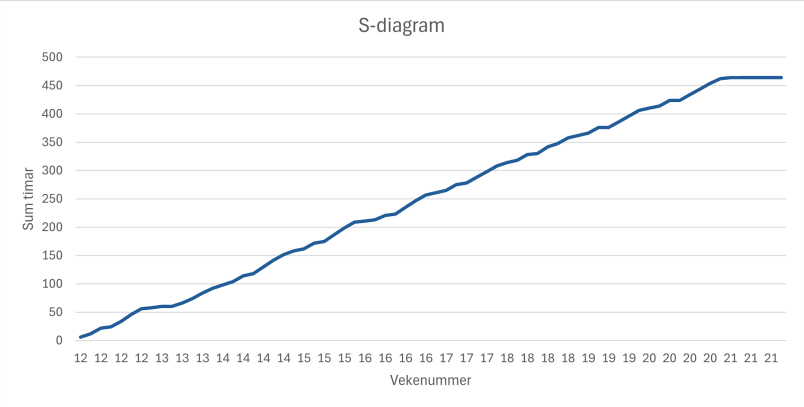


Figure 2: S-diagram tidsestimat

Vedlegg C: Endringsnotat forprosjekt

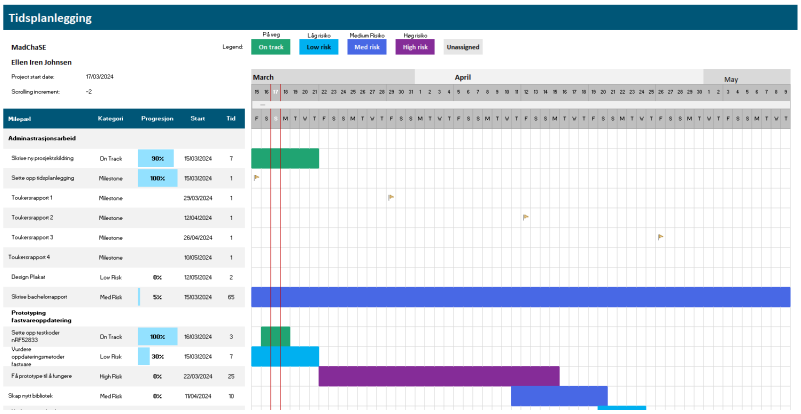


Figure 3: Utdrag av Gantt-diagram

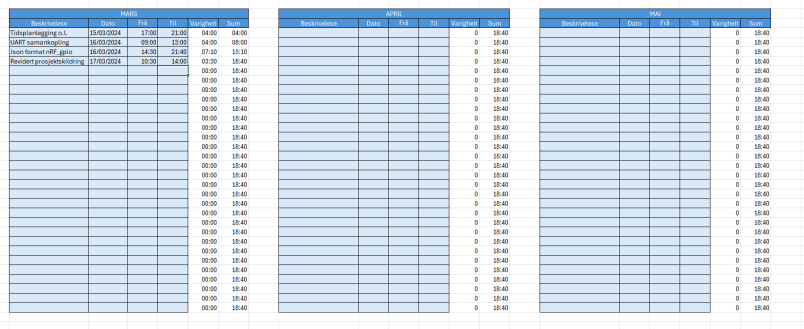


Figure 4: Utrag av timelistedokument

Vedlegg D: Arbeidsnotat

Arbeid gjennomført før 12.02.2024:

Merk! Dette er i stor grad eit estimat då det ikkje blei loggført timar på ein høveleg måte i perioden

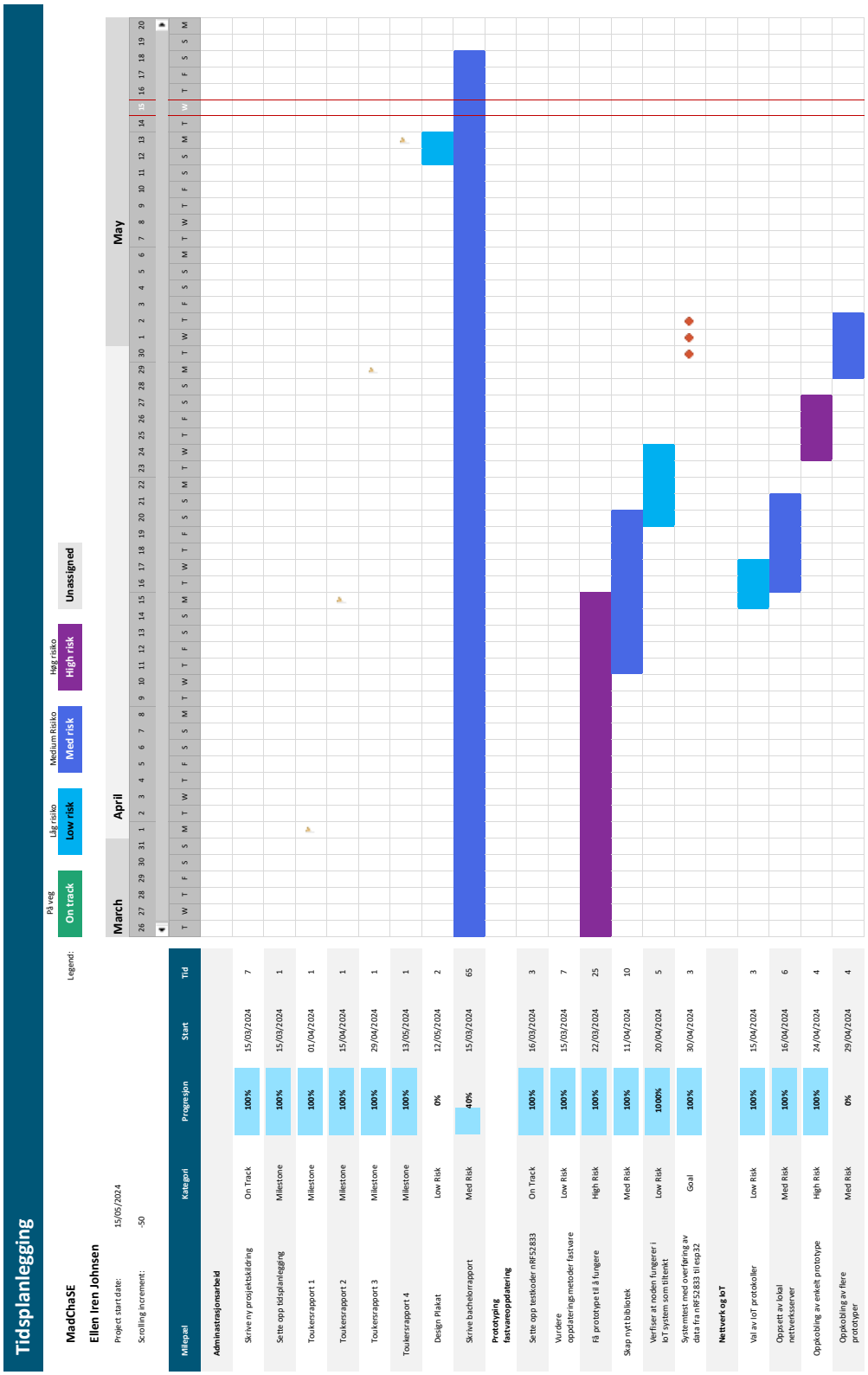
Arbeid	Timar	Vidareføringsplan
Oppsett internt forum(Discord)	2	Overført til resten av gruppa
Oppsett GitHub med samankopling til internt forum og skrive “standard” for kode	6	Overført til resten av gruppa, men vil bruke deler f.eks ReadMe.md
Oppsett av Raspberry Pi einingar (4)	2	Overført til gruppa, men kunnskap kan gjenbrukast
Kople einingar til NTNU sitt IoT nettverk og loggføre dette	3	Overført til gruppa
Prøvde å sette opp Toolchain for kompilering av kode til nRF52833 på Raspberry Pi (Linux)	10	Feilet, men relevant kompetanse
Oppsett og test av Segger J-Link på Raspberry Pi	6	Overført til gruppa
Oppsett av UART kommunikasjon mellom nRF og Raspberry Pi	4	Overført til gruppa
Skrive intern dokumentasjon om Raspberry Pi oppsett, J-Link og UART	3	Overført til resten av gruppa med forum
Gå gjennom Nordic Academy kurs (Fundamentals og BLE)	20	Kompetanseoppbygging
Skrive internt hjelpedokument for nRF-programmering	2	Tilgjengleg for begge partar
Sette opp testkode til bruk med Raspberry Pi	2	Tilgjengleg for begge partar
Oppsett og utprøving av ESP32-biblioteket: https://github.com/atc1441/ESP32_nRF52_SWD	2	Hovudarbeidet eg skal fortsette med
Produksjon av testutstyr til ESP32-nRF52833	5	Skal ta med vidare
Møtetid med forbereding	11	-
Skrive møtereferat	5	Tilgjengleg for begge partar
Skrive forprosjekt	8	Tilgjengleg for begge partar
Estimat anna lesetid	12	Kompetanseoppbygging

Samla tidsestimat: 103 timar

Vedlegg E: Timeliste

MARS						APRIL						MAI					
Beskrivelse	Dato	Frå	Til	Varighet	Sum(t)	Beskrivelse	Dato	Frå	Til	Varighet	Sum	Beskrivelse	Dato	Frå	Til	Varighet	Sum
Tidsplanlegging o.l.	15/03/2024	17:00	21:00	4.00	4.00	Debugging	02/04/2024	14:40	21:30	6.83	102.83	Fikse struktur teil flash-skriving	01/05/2024	13:00	23:15	10.25	320.50
UART samankopling	16/03/2024	09:00	13:00	4.00	8.00	Adminastrasjonsarbeid	03/04/2024	09:00	12:30	3.50	106.33	Les av flash	02/05/2024	09:30	19:30	10	330.50
Json format nRF_gpio	16/03/2024	14:30	21:40	7.17	15.17	Oppsett bachelorrapport	04/04/2024	10:00	16:00	6.00	112.33	Langdag, variert	03/05/2024	10:00	22:00	12	342.50
Revidert prosjektskilding	17/03/2024	10:30	14:15	3.75	18.92	Bachelorskriving	05/04/2024	06:00	12:30	6.50	118.83	Research/test SRAM	04/05/2024	08:00	18:00	10	352.50
SWD undersøking og dokumentarbeid	17/03/2024	15:15	21:00	5.75	24.67	Møtetid (med forbereding)	05/04/2024	13:00	14:15	1.25	120.08	IoT oppsett	05/05/2024	08:00	18:00	10	362.50
Produksjon arbeidsnotat	18/03/2024	08:50	10:00	1.17	25.83	Bachelorskriving	05/04/2024	15:00	19:00	4.00	124.08	IoT ferdigstilling	06/05/2024	10:00	22:00	12	374.50
Undersøking SWD protokoll	18/03/2024	10:00	12:00	2.00	27.83	Langdag, variert	06/04/2024	09:15	21:30	12.25	136.33	Data-avlesing	07/05/2024	10:00	22:00	12	386.50
Forts.	18/03/2024	12:40	18:00	5.33	33.17	Langdag, variert	07/04/2024	08:00	18:00	10.00	146.33	Test	08/05/2024	10:00	22:00	12	398.50
Bygge SWD klokketiming	19/03/2024	08:00	12:00	4.00	37.17	Bachelorskriving	08/04/2024	10:00	12:00	2.00	148.33	Test	09/05/2024	10:00	22:00	12	410.50
Forts.	19/03/2024	12:00	15:00	3.00	40.17	Bachelorskriving	08/04/2024	12:00	14:00	2.00	150.33	Bachelorskriving	10/05/2024	10:00	18:00	8	418.50
SWD-bibliotek restrukturering	20/03/2024	12:00	16:00	4.00	44.17	Bachelorskriving	09/04/2024	08:30	12:00	3.50	153.83	Bachelorskriving	11/05/2024	10:00	18:00	8	426.50
SWD-bibliotek read/write	21/03/2024	09:00	14:00	5.00	49.17	Bachelorskriving	10/04/2024	21:30	22:30	1.00	154.83	Bachelorskriving	12/05/2024	10:00	18:00	8	434.50
Test/fiks read/write	22/03/2024	10:00	14:40	4.67	53.83	Filsystem ESP32	11/04/2024	09:40	16:00	6.33	161.17	Bachelorskriving	13/05/2024	10:00	18:00	8	442.50
SWD-bibliotek reqeuest-melding	23/03/2024	07:00	09:30	2.50	56.33	WiFi + Filsystem	11/04/2024	19:15	21:00	1.75	162.92	Bachelorskriving	14/05/2024	10:00	18:00	8	450.50
Test/fiks request melding	24/03/2024	08:00	12:00	4.00	60.33	Langdag, variert	12/04/2024	11:40	23:40	12.00	174.92	Bachelorskriving	15/05/2024	10:00	18:00	8	458.50
Bytte klokke fra interrupt til delay	25/03/2024	11:30	14:00	2.50	62.83	Langdag, variert	13/04/2024	08:00	20:00	12.00	186.92	Bachelorskriving	16/05/2024	10:00	18:00	8	466.50
Lese dokumentasjon SWD/nRF	26/03/2024	09:00	18:00	9.00	71.83	Langdag, variert	14/04/2024	08:00	18:00	10.00	196.92	Bachelorskriving	17/05/2024	10:00	18:00	8	474.50
Organisere og laste opp kode til GitHub	27/03/2024	08:00	10:00	2.00	73.83	Programmering flashminne	15/04/2024	18:00	21:30	3.50	200.42	Bachelorskriving	18/05/2024	10:00	18:00	8	482.50
SWD-bibliotek ACK og Tms handtering	28/03/2024	10:00	14:40	4.67	78.50	Bachelorskriving	16/04/2024	08:00	12:00	4.00	204.42	Bachelorskriving	19/05/2024	10:00	18:00	8	490.50
Test/fiks ACK handtering	29/03/2024	12:00	16:30	4.50	83.00	Bachelorskriving	17/04/2024	17:00	20:00	3.00	207.42	Bachelorskriving	20/05/2024	10:00	18:00	8	498.50
SWD-bibliotek DP-read	30/03/2024	13:00	19:00	6.00	89.00	Langdag, variert	18/04/2024	11:30	21:00	9.50	216.92	Ferdigstilling/Innlevering	21/05/2024	08:00	12:00	4	502.50
Finne kilder	31/03/2024	16:00	23:00	7.00	96.00	Nettverksplanlegging, skriving	19/04/2024	10:00	16:00	6.00	222.92		22/05/2024			0	502.50
				0.00	96.00	Langdag, variert	20/04/2024	08:00	18:00	10.00	232.92	Forberede presentasjon	23/05/2024	10:00	14:30	4.5	507.00
				0.00	96.00	Langdag, variert	21/04/2024	10:30	22:00	11.50	244.42	Forberede presentasjon	24/05/2024	10:00	14:30	4.5	511.50
				0.00	96.00	NVMC + Bachelorskriving	22/04/2024	19:00	21:00	2.00	246.42		25/05/2024			0	511.50
				0.00	96.00	Fikse testutstyr + ERASEALL	23/04/2024	18:00	23:00	5.00	251.42		26/05/2024			0	511.50
				0.00	96.00	Omstrukturering + testing	24/04/2024	18:00	22:00	4.00	255.42	Gjennomføre prestasjon	27/05/2024	12:00	14:00	2	513.50
				0.00	96.00	Lesedag	25/04/2024	12:00	16:30	4.50	259.92					0	513.50
				0.00	96.00	Les av fabrikkinformasjon	26/04/2024	10:00	18:00	8.00	267.92					0	513.50
				0.00	96.00	Skriv til flash	27/04/2024	11:20	22:00	10.67	278.58					0	513.50
						Langdag, variert	28/04/2024	08:00	20:30	12.50	291.08					0	513.50
						Skriv til flash	29/04/2024	12:30	23:10	10.67	301.75					0	513.50
						Finne kilder og vurdere IoT	30/04/2024	08:00	16:30	8.50	310.25					0	513.50

Vedlegg F: Gannt Diagram



Vedlegg G: Videomateriale

Videomateriale er skapt for å demonstrere verkemåten til systemet i liten skala. Det er skapt tre videoar for å vise ulike sentrale aspekt ved prosjektet.

- Full gjennomgang av ikkje-forstyrrande måling med MQTT: Viser gjennom skjermklipp alle kommandoer og resultat frå ikkje-forstyrrande måling
- Fastvareoppdatering med MQTT: Viser frekvensendring etter ny versjon av blinky har blitt lasta ned. Bruker MQTT for å kontrollere tid/flyt
- Avlesing av fastvare og gjenoppretting av fastvare

Materialet ligger under linktilgjengleg mappe på One Drive assosiert med prosjektet fram til 17.Juni 2024.
Ved delingsfeil eller ynskje om tilgong etter perioden: kontakt Ellen Iren Johnsen.

Vedlegg H: Teknisk Plakat

Bacheloroppgåve

Ellen Iren Johnsen

Institutt for elektronisk system

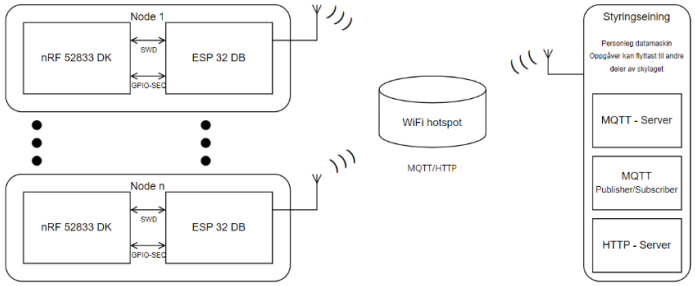
Bakgrunn

MaDChaSE er eit prosjekt i regi av NTNU og Nordic Semiconductor, der målet er å skape måleutstyr for å gjennomføre kanalmålingar i eit nettverk med opptil fleire hundre nodar av typen nRF52833 Development Kits som samarbeider for å hente inn data. Grunnlaget for denne oppgåva er ynskjet om å skape eit verktøy som kan gjennomføre trådlaus oppdatering av fastvare til nRF52833 DK gjennom den påbygde debugger-porten.

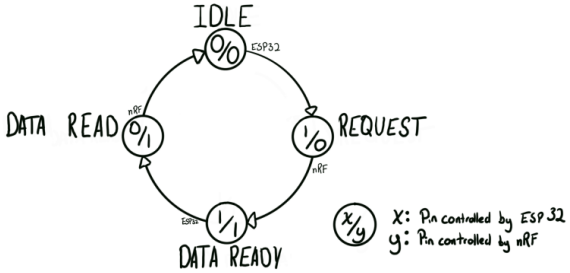
Prosjektkrav

- Fastvareoppdatering skal kunne gjennomførast trådlaust
- Systemet skal ha verktøy for organisering av fastvareoppdateringar
- Systemet skal bruke lett tilgjenglege komponent
- Produksjon av fastvarefiler skal kunne gjennomførast med nRF Connect SDK

Implementasjon



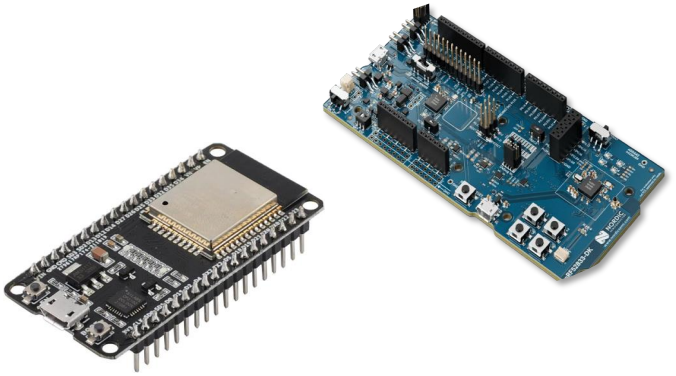
Systemet er sett saman av ei styringseining og ei udefinert mengde nodar som kommuniserar ved hjelp av dei trådlause protokollane MQTT og HTTP.
Nodane er sett saman av ein ESP32 og eit nRF52833DK. ESP32 handterar all trådlaus kommunikasjon, nRF fungerar i praksis her som ein programmerbar sensor.

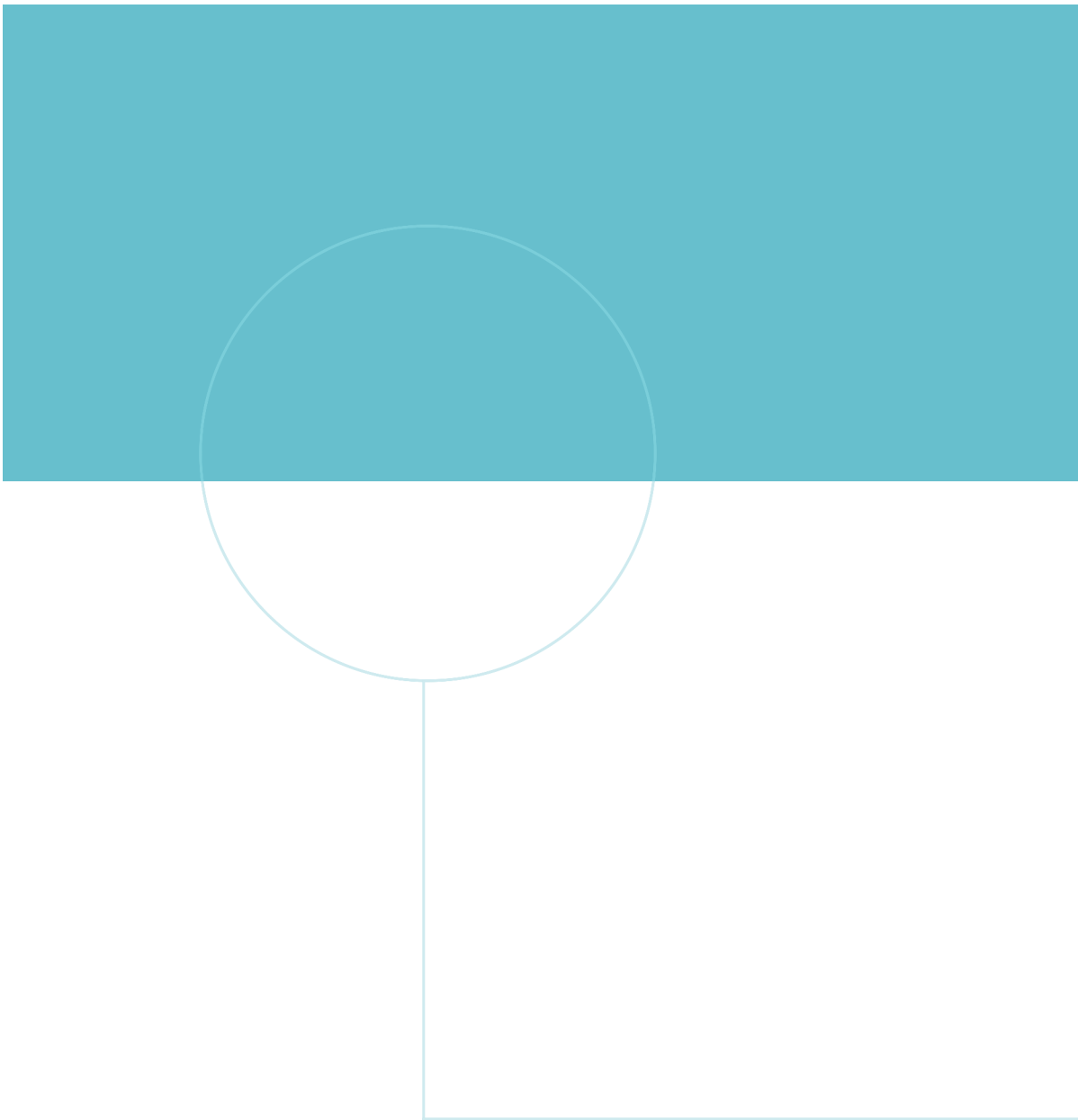


I tillegg til å utføre fastvareoppdatering kan nemleg SWD-grensesnittet brukast til å direkte lese av minneregister på nRF-einingа. Det er utbygd ein eigen digital-logisk sekvens som tillater ESP32 å hente ut data ved fullførte måling uten å forstyrre programflyten til nRF528333DK

Konklusjon

Prosjektet oppnår krava skildra og fungerar som ein positiv indikator for vidare utvikling. Neste steg vil være å forbetre sikkerheit i trådlaus kommunikasjon, forbetre nettverksarkitektur med f.eks. DNS-server og gjere nye komponentval





NTNU

Kunnskap for ei betre verd