CAN Node

Gruppe 5

av

Morten Jacobsen og Mohammed Al-Rawi

i

DAT220 20V
Softwareutvikling 2

Veiledet av **Ken Henry Andersen**
Fakultet for teknologi og realfag
Universitetet i Agder

Grimstad, Desember 2020

Sammendrag

Målet til gruppen var å lage en CAN bus node som kunne motta, lese, tolke og dekode ønsket innholdt fra et utvalg av IDer og til slutt vise dataen fordelt på flere interaktive skjermer. Oppgaven ble delt opp i mindre deler og fordelt blant medlemmene, og disse delene ble publisert på individuelle brancher før de ble lagt sammen med master branchen. Oppgaven viste seg å være meget tidskrevende med tanke på igangsetting, fysiske endringer på kortet som måtte til samt innhenting av nødvendig dokumentasjon før produktet ble fungerende. Gruppen har tatt stor lærdom av oppgaven og ser seg fornøyde med det endelige produktet samt gjennomføring av oppgaven som helhet.

Obligatorisk gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

1.	Vi erklærer herved at vår besvarelse er vårt eget arbeid, og at vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	Ja
2.	**Vi erklærer videre at denne besvarelsen:** • Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • Ikke refererer til andres arbeid uten at det er oppgitt. • Ikke refererer til eget tidligere arbeid uten at det er oppgitt. • Har alle referansene oppgitt i litteraturlisten. • Ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	Ja
3.	Vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.	Ja
4.	Vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert.	Ja
5.	Vi er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk.	Ja
6.	Vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider.	Ja
7.	Vi har i flertall blitt enige om at innsatsen innad i gruppen er merkbart forskjellig og ønsker dermed å vurderes individuelt. Ordinært vurderes alle deltakere i prosjektet samlet.	Nei

Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. §2).
Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

Vi gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	Ja
Er oppgaven båndlagt (konfidensiell)?	Nei
Er oppgaven unntatt offentlighet?	Nei

Forord

Denne rapporten av utviklingsprosessen, er en del av DAT220 20H softwareutvikling 2 prosjektet.

Gruppen vil takke deres veiledere: Universitetslektor Ken Henry Andersen, Universitetslektor Christian Auby og Assistant Professor Geir Jevne som foreleser ved universitetet i Agder for riktig veiledning gjennom prosjektperioden.

Grimstad
18. desember 2020
Morten Jacobsen og Mohammed AlRawi.

Forkortelser

Forkortelse	Hele uttrykket
CAN	Controller Area Network
LIN	Local Interconnect Network
CANFD	Controller Area Network Flexible Data-rate
SoC	System On Chip
RTOS	Real Time Operating System
OBD	On-board diagnostics
TFT	Thin film transitor
UML	Unified Modeling Language
SDL	Specification and Description Language
DK	Development kit
OTA	Over the air
GUI	Graphical User Interface
PCB	Printed Circuit Board
SD	Secure Digital

Tabell 1: Tabell av forkortelser

Innhold

Figurer

Tabeller

Listinger

1 Innledning

1.1 Bakgrunn

1.2 Produktvisjon

Produktet vårt vil være en permanent skjerm som kan integreres i dashbordet på bilen, eller alternativt montert på en sugekopp plassert på feks. frontrute, dashbord eller andre flat overflate. Vi tror at produktet vårt vil vise ønsket informasjon på en mer organisert måte i forhold til tilgjengelige produkter på markedet. Det tillater også kunden å samhandle med produktet på en mer responsiv måte i sammenligning med en mobiltelefonapplikasjon eller en statisk skjerm.

1.3 Lignende produkter

For tiden eksisterer det mobiltelefonapplikasjoner som grensesnitt med en Bluetooth OBD2 adapter som har lignende funksjoner. I tillegg finnes det noen enkle Arduino prosjekter som viser CAN-bussinformasjon på en liten statisk skjerm.

2 Prosjekt organisering

2.1 Planlegging

2.2 Gruppe roller

Å jobbe med et produkt i en gruppe krever produktivitet og teamarbeid. Ved å dele problemene opp i mindre deler, kan hver del deles mellom gruppemedlemmene for å gjøre en klar fremgang. Produktet består av to hoveddeler, STM32H7B3I-DK del og GUI TouchGFX del. Alle gruppe medlemmer jobbet med begge deler. Roller som var fordelt var Kanban-master, koordinerings-ansvarlig, referent, møte-leder og Git-ansvarlig.

2.3 Møter

Det har blitt holdt ukentlige møter mellom gruppemedlemmer i elektronikklaboratoriet eller chattrommet *Prosjekt DAT220*, for å holde en jevn kommunikasjon om utviklingen. Gruppen jobbet på laboratoriet sammen hver mandag og onsdag og et møte hver mandag. Her ble det diskutere problemstillinger og fremdriften i prosjektet, og sammenslåing av funksjonskode fra de ulike delene.

2.4 Time log

Tidslogging ble gjort i JIRA-Kanban, der individuelle gruppemedlemmer kunne logge timer knyttet til et problem.Vi hadde to Story (System) og vi lagde subtasks underveis. En ny subtask ble opprettet hver uke etter møte og nye oppgaver ble lagt i kø. Hver subtask ble tildelt en ansvarlig, dette betydde ikke at dette medlemme skulle løse dette alene men at det medlemmet er ansvarlig for å følge denne subtasken opp å se at den holder fristen.

2.5 Versionskontroll

Vi brukte Git (BitBucket) for versjonskontroll, det er det mest brukte systemet innenfor programvare utviklingsmiljøet. Det er et verktøy som tillater flere utviklere å arbeide på samme kildekode samtidig. Hver utvikler som jobber med en blokk i systemet lager en branch til en spesifikk task. Etter en utvikler er ferdig med tasken blir koden pushet til serveren med en generert "Pull request". "Pull request" er for at andre medlemmer i teamet kan sjekke om de er enige i løsningen. Deretter merges koden med masteren og branchen slettes.

3 Teknisk bakgrunn

3.1 Hardware

3.1.1 STM32H7B3I-DK

Vi brukte et avansert utviklingskort fra STMicroelectronics med ARM Cortex-M7 prosessor og 480x272 LCD-TFT touch-skjerm. Det har innebygd CAN controller og CAN transciever preferienhet samt en rekke andre perifere enheter. For lagring av grafikk har kortet et eksternt minne (SDRAM) på 128Mib og 512Mib NOR Flash [9].

3.1.2 LCD-TFT

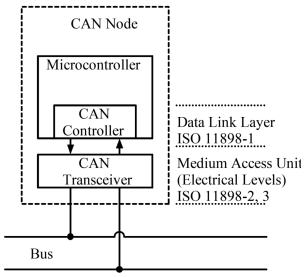
STM32H7B3I-DK kortet inkluderer en 4.3 tommers LCD touchskjerm kort koblet via RGB interface på STM32H7B3LIH6QU mikrokontrolleren. LCD kortet bruker RK043FN48H-CT672B TFT LCD fra Rocketch med driversystem, hvitt LED baklys, og capacitiv touch panel. Touchskjerm kontrolleren kommuniserer med STM32H7B3LIH6QU via I2C4 bus.

3.1.3 MCP2562FD

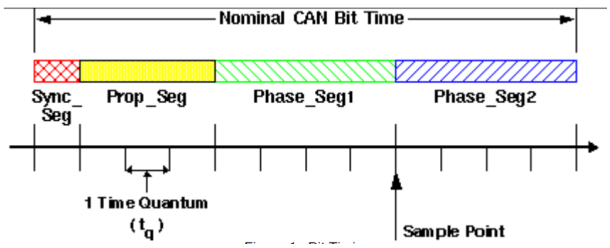
STM32H7B3I-DK kortet bruker MCP2562FD FDCAN Transceiver [9]for å kommunisere med andre noder på CAN busen. I vår applikasjon skal vi kun lese fra CAN bus så vi har konfiguert den til å være TX passiv.

3.1.3.1 CAN

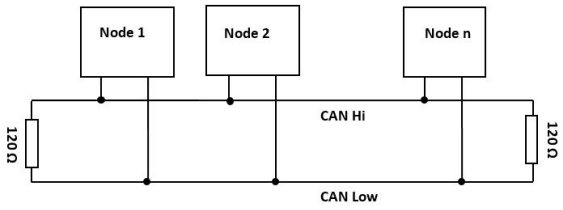
Controller Area Network er en serial asynchronous kommunikasjon som er designet for industrielle og automotive applikasjoner. En CAN bus kan bestå av to eller flere noder. Den type protokoll bruker bare et par kabler, CANH (High) og CANL (Low) med en 120 Ω ved terminering, see figure 3 [13]. Hver bit i en CAN melding består av flere tq (time quanta), time quanta er en periode av klokka frekvensen til CAN kontrolleren. En bit er delt opp i fire forskjellige deler, see figure 2 [3]. Selv om CAN kommunikasjon er asynchronous så må alle nodene som er koblet til en CAN bus sample hver bit på samme tid, og det skjer ved å konfiguere nominal bit timing på hver node til å ha samme antall tq(time quanta) for hver bit. En standard CAN er delt i to, Base Frame Format version 2.0A bruker en 11-bit melding ID men Extended Frame Format version 2.0B bruker opptil 29-bit melding ID, see figre 4 [1]. Hver node på en CAN Bus må inneholde en CAN Controller og Can Transciever, see figure 1 [25].



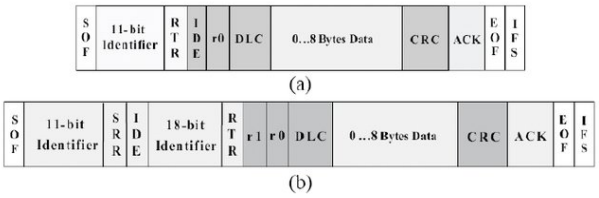
Figur 1: CAN Node



Figur 2: Nominal tq



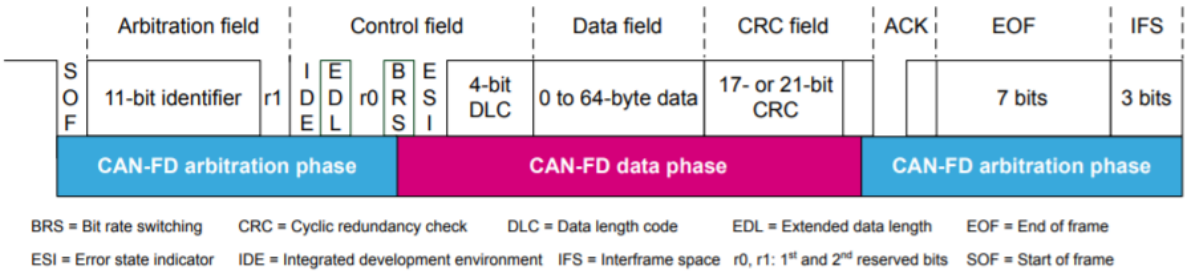
Figur 3: Can Wiring



Figur 4: Can2.0 frames

3.1.3.2 CAN FD

Controller Area Network Flexible Data-Rate versjon 1.0 ble introdusert i april 2012. FD-CAN er integrert i ISO 11898-1 hoved CAN standard [26] . FDCAN tillater mer enn 8 data bytes i en frame og gir muligheten til å overføre den sentrale delen (Data fasen) av en frame i høyere bit-rate enn den klassiske CAN kommunikasjonen. Figur [5] viser en standard FDCAN frame (11-bit meldingsID) mens en Extended FDCAN frame er lik standard FDCAN frame når vi legger til 18-bit identifier etter IDE bit'en i den første arbitration fasen [10].



Figur 5: CAN-FD frame

3.1.4 IXXAT USB-to-CAN

Dette er et HI-speed , USB 2.0 PC bus interface. Det er et hjelpemiddel for å kunne sende og motta CAN data fra/til PC til/fra en CAN Bus ved bruk av bus overvåker program [8].

3.2 Software

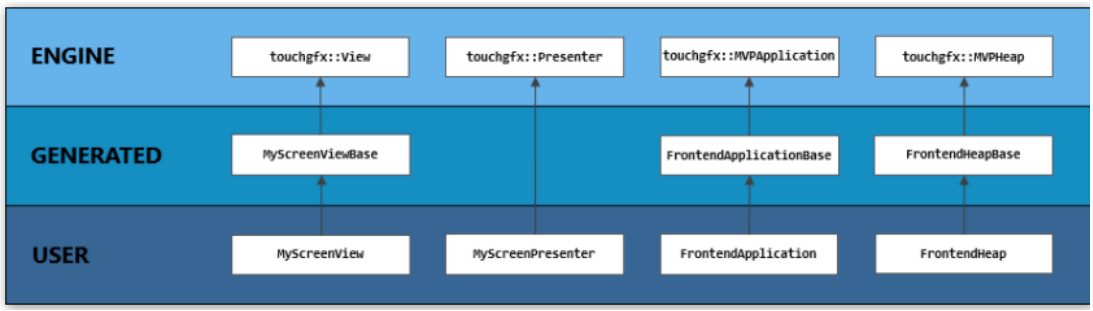
3.2.1 STM32CubeIDE

STM32CubeIDE er STM sin utviklingsplattform for embedded-systemer. Det er basert på Eclipse IDE, kompilerer med GCC og benytter GDB for debugging. STM sitt grensesnitt for innstilling av pinner og periferienheter Cube MX er også integrert. Det syr sammen en hel toolchain som gjør det lett å komme i gang med utvikling på STM sine mikrokontrollere. Selv om de individuelle delene har eksistert lenge er CubeIDE plattformen ganske

ny. Versjon 1.0 ble sluppet i fjor og vi er nå på versjon 1.4. Brukergrensesnittet fra Eclipse er beholdt uforandret og ser nokså utdatert ut. Det er også en del andre «utfordringer» i et realtivt nytt program som gjør prosessen unødvendig tungvinn. Blant annet sjekker det i noen tilfeller ikke for rettelser av feil i syntaks før man bygger hele prosjektet på nytt. Dette stjeler en del tid i utviklingsarbeidet. Til tross for noen svakheter er likevel STM32CubeIDE et meget godt alternativ til utvikling på STM sine produkter [11].

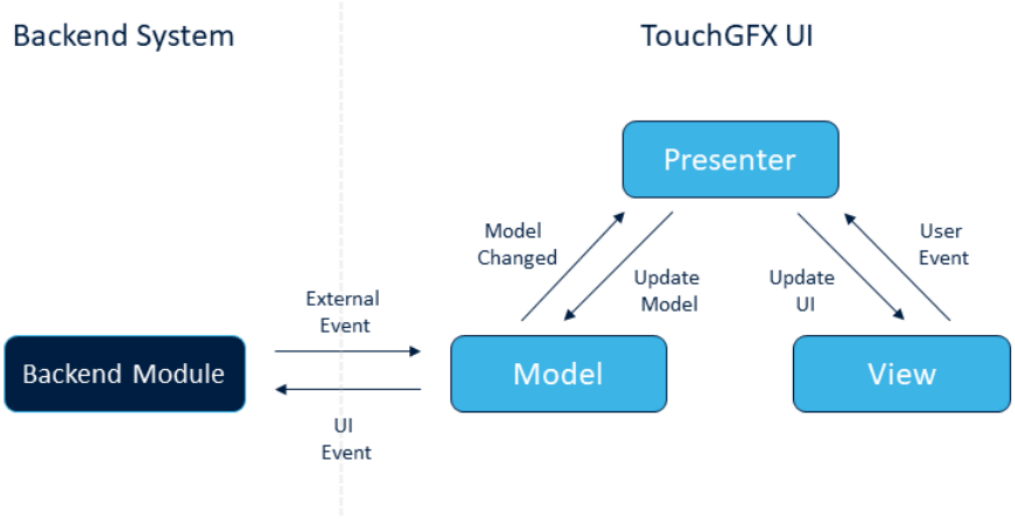
3.2.2 TouchGFX designer

TouchGFX er software som legger til rette for design av brukergrensesnitt på mikrokontrollere. Man kan enkelt legge til grafiske elementer på skjermområdet og knytte dem opp mot ønsket funksjon. Programmet genererer så (C++) kode som håndterer hvordan grafikken oppfører seg. Den genererte koden eksporteres til CubeIDE hvor vi har flere verktøy for feilsøking, bygging og flashing på mikrokontroller. Programmet kommer med noen standard grafiske elementer som kan tilpasses slik man ønsker. I dette prosjektet har vi tatt i bruk elementer som knapper, slidere, progresjonselementer og dynamiske tekstområder. Selve ikonene er endret for å sette vårt preg på designet. Dette tillater oss som embedded-utviklere å bruke tiden på implementering av systemets funksjonalitet i stedet for programmering av visuelle komponenter.



Figur 6: Klasse hierarki av engine, generert og user klasser [17]

Selve kommunikasjonen med back-end systemet, altså ikke-UI relatert kode, gjøres fra *model* klassen. Back-end systemet er en software komponent som mottar og sender hendelser til UIen som f.eks. målinger fra en sensor. I vårt tilfelle er dette hendelser eller meldinger som mottas på *CAN Bus* [20].



Figur 7: Model-View-Presenter and external communication

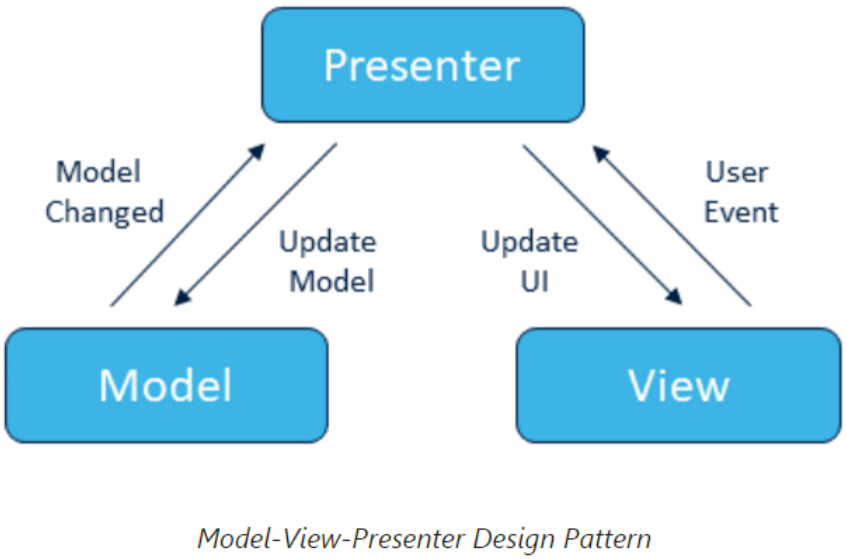
3.2.3 canAnalyser3 Mini

IXXAT canAnalyser3 Suite [7] er et bus overvåknings program som tillater overvåking av bus trafikk på CAN, LIN og CAN-FD bus, og transmisjon av enkelte bus meldinger. Avlesningen skjer ved fysisk tilkobling på busen via en Ixxat USB-to-CAN V2 interface [8] .

3.2.4 Design and architectural patterns

TouchGFX brukergrensesnitt er en arkitekturell pattern kjent som Model-View-Presenter(MVP) som er derivert av Model-View-Controller (MVC) pattern som er brukt ganske mye i brukergrensesnitt applikasjoner. I MVP pattern er det tre klasser:

- *Model* er en interface singleton klasse som alltid er aktiv. *Model* klassen får automatisk peker til den nåværende aktiv *Presenter*. Når det skjer en endring i *Model*, blir den nåværende aktiv *Presenter* varslet om endringen. Dette er gjort via *ModelListener* interface i applikasjonen. *Model* har to hoved formål:
  1. Lagrer tilstands informasjon for *UI*.
  2. Oppføre seg som en interface mot backend systemet, mottar/sender eventer fra/til den nåværende aktiv skjerm.
- *View* er en passiv interface som viser data (fra model) og sender bruker kommandoer (eventer) til presenter. *View* klasse er derivert fra **TouchGFX View** klasse. *View* klassen inneholder "widgets" som er vist som objekt medlemmer i denne *View'en*. Den inneholder også **setupScreen** og **tearDownScreen** funksjoner som blir automatisk kalt når den skjermen er startet/avsluttet.
  View har også en peker til den assosierte *Presenter*. Den pekeren er satt opp automatisk av "framework" (TouchGFX Engine).
- *Presenter* er bindeleddet mellom *View* og *Model*. *Presenter* klasse er derivert fra **TouchGFX Presenter** klasse. Den er ansvarlig for "buisness logic" av den nåværende aktive skjerm. I tillegg mottar den "backend" eventer fra *Model*, og UI eventer fra *View* og bestemmer hva den skal gjøre med de videre. [20]



Figur 8: MVP Design Pattern

3.2.5 GUI objekter

3.2.5.1 TextArea - with one wildcard

TouchGFX har en widget kalt TextArea som har baseklassen sin i TextArea.hpp. Denne baseklassen er abstrakt ved at flere funksjoner er rene virtuelle funksjoner. Baseklassen kan ikke brukes til å instansere objekter, og fungerer kun som et interface. TouchGFX Designer implementerer disse funksjonalitetene knyttet til TextArea for oss. I TextArea bruker vi Wildcards, som er gitt i formatet $< * >$, hvor * representerer et variabelnavn som ikke vil bli inkludert i teksten, men kun verdien denne variabelen får. Verdiene til wildcardet blir satt inn ved runtime i C++ koden. For å tvinge fontgeneratoren til å inkludere spesifikke karakterer, kan det brukes Wildcard Characters og Character Range i de ulike typografiene. For å legge til Wildcards velger Wildcard 1 eller 2, evnt. begge ved behov for flere wildcards. For å kunne oppdatere wildcardet i run-time trenger vi en Wildcard buffer for å lagre den dynamiske teksten. For å sette opp et wildcard buffer velger vi Use Wildcard Buffer i wildcard settings og setter ønsket bufferstørrelse. Ved behov for å minimere bruk av minnet bør man matche bufferen med behovet for antall karakterer så nært som mulig [22].

3.2.5.2 BoxProgress

I BoxProgress.hpp finner vi baseklassen til denne widgeten i Progress indicators gruppen. I likhet med TextArea, er denne baseklassen abstrakt ved at flere funksjoner er virtuelle. BoxProgress viser oss den nåværende progresjonen ved å bruke en enkelt boks på topp av et bakgrunnsbilde. Fargen, altså 'alphaen' til boksen og retningen til progresjonen samt posisjonering og maks/min rekkevidde kan konfigureres i TouchGFX Designer [14].

3.2.5.3 CircleProgress

Circleprogress viser den nåværende progresjonen ved å bruke en sirkel som progresjonsindikator på toppen av et bakgrunnsbilde. Baseklassen finnes i CircleProgress.hpp og er abstrakt i likhet med TextArea og Box Progress. Fargen (alpha), samt posisjon, steg, start/stopp vinkel, radius og strek bredde kan konfigureres i TouchGFX Designer. I tillegg er det mulig å lage en egendefinert bakgrunn. Sirkelen er basert på CanvasWidget og er meget prosesskrevende [16].

3.2.5.4 Slider

En slider bruker tre bilder for å vise slideren enten i vertikal eller horisontal orientering. Indikator bildet på slideren kan bli beveget på for å endre verdier som blir sendt gjennom callback funksjoner. I vårt prosjekt ble slideren brukt for å justere lysstyrken på skjermen. Slideren blir funnet i Widget gruppen Miscellaneous i TouchGFX Designeren. Navn, type, lokasjon, stil, bilde, posisjon og verdier kan bli endret i TouchGFX Designeren og baseklassen ligger i Slider.hpp. Slidere kan tilegnes triggere som utløser en gitt hendelse når den blir beveget på [21].

3.2.5.5 Buttons

Buttons finnes i widget gruppen *Buttons* i TouchGFX Designer. Her velger man så posisjon og ønsket ikon for den gjeldende knappen. En knapp i TouchGFX er bevisst på hendelser (den blir trykket ned og sluppet) og kan sende et callback når knappen blir sluppet. Hver tilstand, nedtrykket og sluppet, er assosiert med et eget bilde. Man kan endre navn, lokasjon, stil, bilde, utseende mm. på hver knapp. I tillegg kan knapper tilegnes en trigger som kan utløse hendelser når knappen blir trykket på [15].

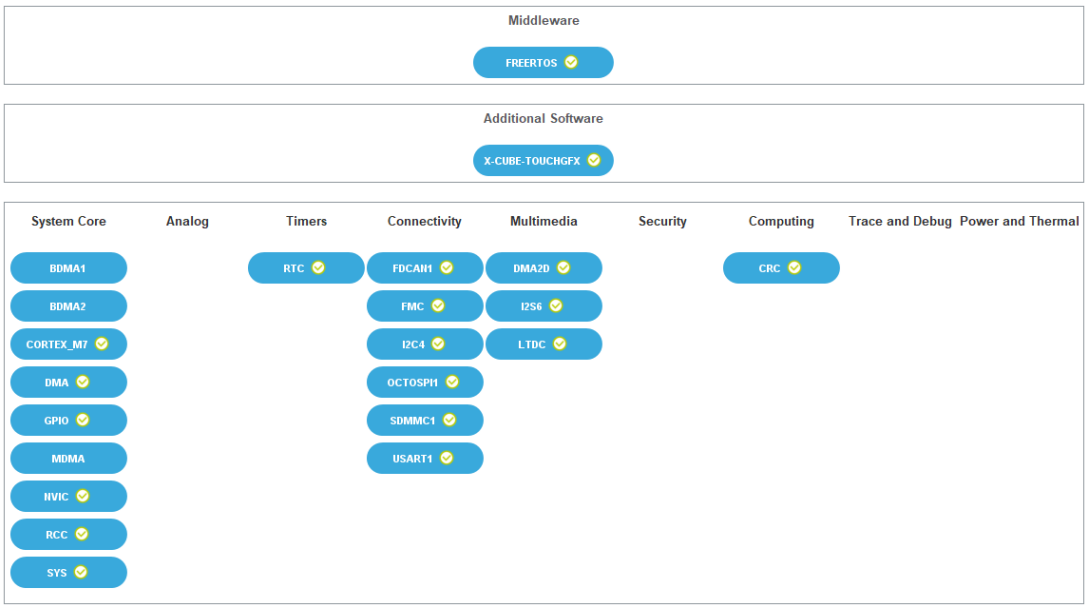
3.2.6 TouchGFX AL

TouchGFX Abstraction Layer (AL), i en TouchGFX applikasjon, er software komponenten som sitter mellom kortets initialiserings kode og TouchGFX Engine. Dens hoved oppgave er å binde sammen Engine med den underliggende hardwaren og operativ systemet. Det er gjort ved å abstraktere den underliggende hardwaren og OS slik at den kan bli behandlet i en ryddigere måte av Engine. AL består av to forskjellige deler, Hardware Abstraction Layer (HAL) og Operating System Abstraction Layer (OSAL). TouchGFX Engine følger grafikkprinsippene for beholdt modus. Kort sagt betyr dette at TouchGFX gir en modell som kan manipuleres av brukeren, og TouchGFX tar seg deretter av å oversette fra denne modellen til et optimalisert sett med gjengivelsesmetode kaller [18].

TouchGFX Engine har en uendelig main loop som utfører tre grunnleggende steg [19] [24]:

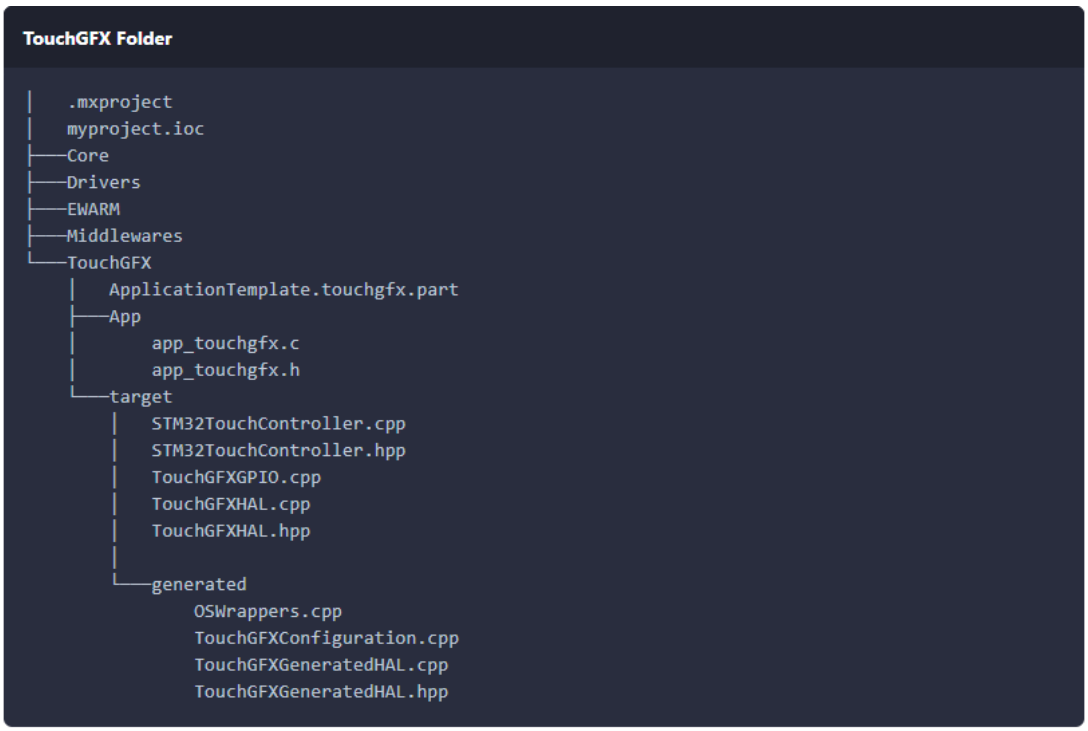
1. Samle input (Berørings koordinater, knapper osv.).
2. Oppdatere Scene Model (Scene Model er samling av forskjellige komponenter).
3. Render (gjengi) Scene Model til Framebufferet.

TouchGFX Engine blir inkludert i STM32CubeIDE som ekstra Software, se figur 9 .



Figur 9: System view- figuren hentet fra vår applikasjon i STM32CubeIDE

Når TouchGFX Generator er lagt til og enablet, vil CubeIDE lage TouchGFX mappen for prosjektet. Mappen inneholder de samme filene for alle prosjekter men innholdet på filene kan bli endret av utvikleren. Figur[10] viser innholdet i en CubeIDE prosjekt med TouchGFX Engine enablet. Figur 11 viser de forskjellige mappene i et project i STM32CubeIDE [23].



Figur 10: TouchGFX Engine

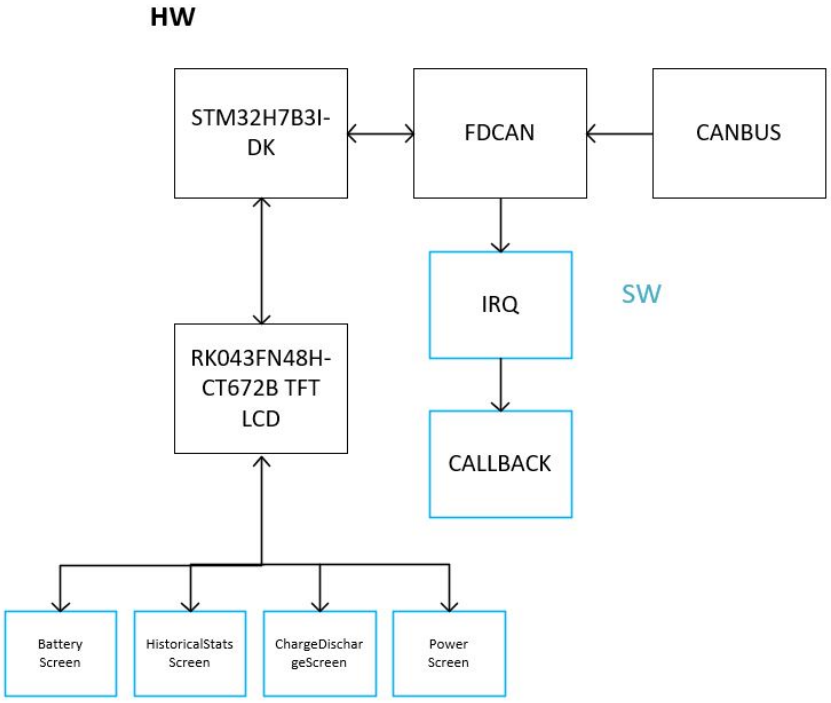
Folder	Resbonsibility
myproject.ioc	CubeIDE Project file
Core	main.c or main.cpp and startup code
Drivers	CMSIS and MCU family drivers
EWARM	IDE project folder. In our application it is STM32CubeIDE
Middlewares	Contains TouchGFX library/headerfiles and third party software like FreeRTOS
Template.touchgfx.part	The .part file is updated by CubeIDE with information that is relevant to TouchGFX Designer project, e.g. screen dimensions and bit depth
App	X-CUBE interface to CubeIDE. app_touchgfx.c contains definitions for the function MX_TouchGFX_Process(void) and MX_TouchGFX_Init(void) which are used to initilize TochGFX and start its main loop
target/generated	This sub-folder contains the read-only files that get overwritten by CubeMX when configurations change. TouchGFXGeneratedHAL.cpp is a sub-class of the TouchGFX class HAL and contains the code that CubeMX can generate based on its current configuration. OSWrappers.cpp (The OSAL) contains the functions required to synchronize with TouchGFX Engine, and finally TouchGFXConfiguration.cpp which contains the code to construct and configure TouchGFX, including the HAL.
target	Contains the bulk of files that can be modified by the user to extend the behavior of the HAL or to override configurations generated by CubeMX. STM32TouchController.cpp contains an empty touch controller interface. TouchGFXHAL.cpp defines a sub-class, TouchGFXHAL, of TouchGFXGeneratedHAL.

Figur 11: Project folders

3.2.7 FreeRTOS

FreeRTOS er et sanntidsoperativsystem (RTOS) laget for mikrokontrollere og embedded systemer som er gratis å bruke i kommersielle applikasjoner, og blir levert med kildekoden til STM32CubeIDE. FreeRTOS er et Real-Time operativsystem som støtter applikasjoner med ulike tjenester ved å fordele databehandlingsressurser og CPU tid til de ulike trådene som kjører. Ved å bruke et RTOS kan et prosjekt bli strukturert til å kjøre flere ulike oppgaver(tråder) i tilsynelatende parallell samtidig som disse oppgavene kan bli tildelt ønsket prioritering [5] [27].

4 Produkt og implementasjon

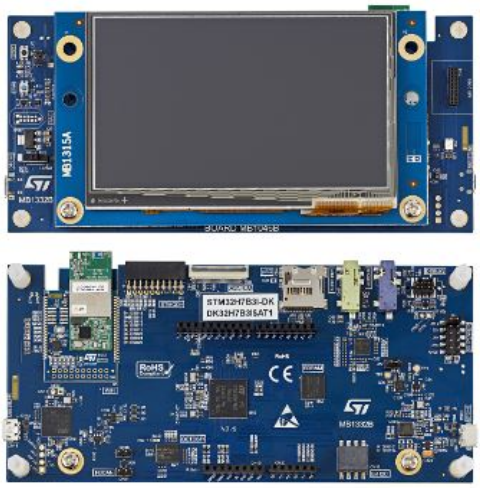


Figur 12: Hardware og software blokk diagram

4.1 Overordnet struktur

4.1.1 STM32H7B3I-DK

Innledende var det tilsiktet i dette prosjektet å anvende STM32F746NGH6, men da denne mikrokontrolleren ikke inneholdt en FDCAN periferienhet så gruppen det som nødvendig å anskaffe en annen type mikrokontroller, valget falt da på STM32H7B3I-DK som har en FDCAN periferienhet. STM32H7B3I-DK er en M32H7B3LIH6QU Arm® Cortex-M7 basert mikrokontroller og har ekstern 512Mbit Octo-SPI NOR FLASH samt ekstern 128Mbit SDRAM mm. som er tilfredsstillende i forhold til størrelsen på vårt prosjekt.

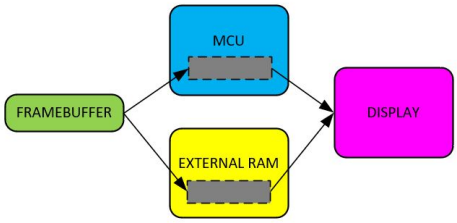


Figur 13: STM32H7B3I-DK med RK043FN48HCT627B TFT LCD display

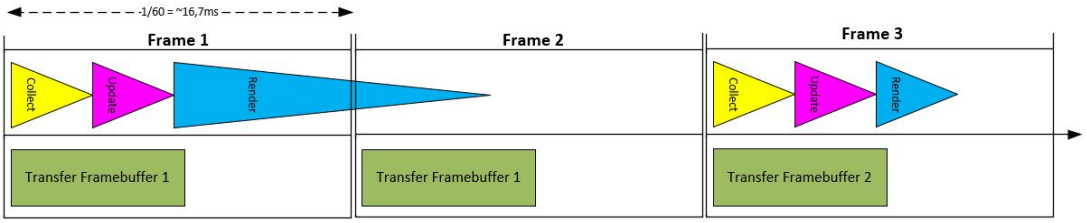
4.1.2 RK043FN48HCT627B TFT LCD

Figur 14: LCD TFT layers eksempel [12]

STM32H7B3I-DK kommer med et RK043FN48HCT627B TFT LCD display fra Rocktech med en oppløsning på 480x272 piksler og en fargedybde på 24bits (RGB888). Displayet mottar framebufferet fra mikrokontrolleren gjennom skjermdriveren som kjøres med Parallell RGB (LTDC). Framebufferet er en del av minnet som blir oppdatert av grafikk motoren til å inneholde neste bilde som blir vist på displayet. Framebufferet er en sammenhengende del av RAM av en gitt størrelse, og innholdet i framebufferent er det som til slutt blir overført til og vist på det fysiske displayet. Framebufferet kan bli plassert enten i intern RAM i MCUen for rask skrive- og lesetilgang, dog er den interne RAM ressursen begrenset (1.4MByte) og det ble derfor valgt å bruke ekstern RAM med en noe redusert lese og skrivehastighet men større kapasitet (128Mbit SDRAM). Da vi har rikelig med ekstern RAM tilgjengelig ble det valgt en dobbel framebuffer strategi, Den ene framebufferen brukes til å skrive det neste resulterende bildet, den andre framebufferen brukes til å overføre til skjermen (se figure 16), for bedre ytelse på bekostning av økt minnebruk. Denne doble framebuffer strategien tillater en lengre lesetid pr framebuffer.



Figur 15: Mulige lokasjoner for framebufferet

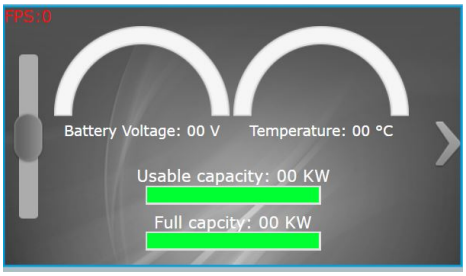


Figur 16: Dobbel framebuffer strategi

4.1.3 BatteryScreen

Battery Screen inneholder en grafisk fremstilling av fire ulike verdier tilknyttet hovedbatteriet i en Tesla Model S samt en frames per sekund (FPS) indikator. Disse variablene er:

- Main Battery Voltage (TextArea og Circleprogress)



Figur 17: BatteryScreen

- Negative Terminal Temperature (TextArea og Circleprogress)
- Usable capacity (TextArea og BoxProgress)
- Full capacity (TextArea og Boxprogress)
- FPS (TextArea)

4.1.3.1 BatteryScreen - TextArea

Realisering av en TextArea widget [3.2.5.1] i TouchGFX Designer finnes i gruppen *Miscellaneous*. I hver TextArea widget er det brukt wildcards med en bufferstørrelse på 10. I BatterySreenViewBase.hpp blir det generert TextArea wildcardbuffere til de ulike TextArea widgetene. I figur 18 ser vi ett eksempel på wildcard buffer oppsett og i figur 19 posisjonering av widgeten.

```
static const uint16_t BATTERYPACKVOLTAGE_TEXTBOX_SIZE = 10;
touchgfx::Unicode::UnicodeChar batterypackvoltage_textboxBuffer[BATTERYPACKVOLTAGE_TEXTBOX_SIZE];
```

Figur 18: Wildcards bufferet

```
batterypackvoltage_textbox.setPosition(50, 121, 190, 19);
batterypackvoltage_textbox.setColor(touchgfx::Color::getColorFrom24BitRGB(255, 255, 255));
batterypackvoltage_textbox.setLinespacing(0);
Unicode::snprintf(batterypackvoltage_textboxBuffer, BATTERYPACKVOLTAGE_TEXTBOX_SIZE, "%s", touchgfx::TypedText(T_SINGLEUSEID2).getText());
batterypackvoltage_textbox.setWildcard(batterypackvoltage_textboxBuffer);
batterypackvoltage_textbox.setTypedText(touchgfx::TypedText(T_SINGLEUSEID1));
```

Figur 19: TextArea oppsetting

Når så buffere og posisjonering er på plass konfigurerer vi funksjonaliteten til TextArea widgeten. Funksjoner og variabler blir deklarert i klassen til BatteryScreenView, se figure20.

Videre blir funksjonene implementert i BatteryScreenView.cpp fig. 22. Variablene tilhørende TextArea widgetene blir her oppdatert ved å peke mot en struct i CanDataStruct.hpp som tar imot dataene fra CANBUSen. Etter variablene har fått riktig verdi brukes det en snprintf funksjon fra Unicode klassen for å printe karakterene etterfulgt av en invalidate() funksjon for å tegne TextArea på nytt.

```
void BatteryScreenView::setData(struct canData* CANDATA_view){

    bpv = CANDATA_view->Model_B_Pack_Voltage; // Battery pack voltage
    temp = CANDATA_view->Model_Neg_Termi_Temp; // Negative terminal temperature
    ucr = CANDATA_view->Model_Usable_Remaining; // Usable Capacity Remaining
    ufc = CANDATA_view->Model_Usable_Full_Pack; // Usable Full Capacity

    CircleProgressBPV.setValue(bpv);
    circleProgressTemp.setValue(temp);
    UsableCapacityBoxProgress.setValue(ucr);
    UsableCapacityFullBoxProgress.setValue(ufc);

    Unicode::snprintf(batterypackvoltage_textboxBuffer, BATTERYPACKVOLTAGE_TEXTBOX_SIZE, "%02d", bpv);
    Unicode::snprintf(Temperature_textboxBuffer, TEMPERATURE_TEXTBOX_SIZE, "%02d", temp);
    Unicode::snprintf(Usable_remaining_textboxBuffer, USABLE_REMAINING_TEXTBOX_SIZE, "%02d", ucr);
    Unicode::snprintf(Usable_full_textboxBuffer, USABLE_FULL_TEXTBOX_SIZE, "%02d", ufc);

    batterypackvoltage_textbox.invalidate();
    Temperature_textbox.invalidate();
    Usable_remaining_textbox.invalidate();
    Usable_full_textbox.invalidate();

}
```

Figur 21: BatteryScreenView::SetData()

```
class BatteryScreenView : public BatteryScreenViewBase
{
public:
    BatteryScreenView();
    virtual ~BatteryScreenView() {}
    virtual void setupScreen();
    virtual void tearDownScreen();

    struct canData* CANDATA_view = NULL;
    void handleTickEvent();
    void setBrightnessScreen1(int value);
    void updateFPS();
    void setData(struct canData* CANDATA_view);

protected:
    uint16_t bpv; // Battery pack voltage
    uint16_t temp; // Negative terminal temperature
    uint16_t ucr; // Usable capacity remaining
    uint16_t ufc; // Usable full capacity

    bool BPV_increase = true;

    int frameSkippedCounter;
    int frames;
    int fps;

};
```

Figur 20: BatteryScreenView.hpp

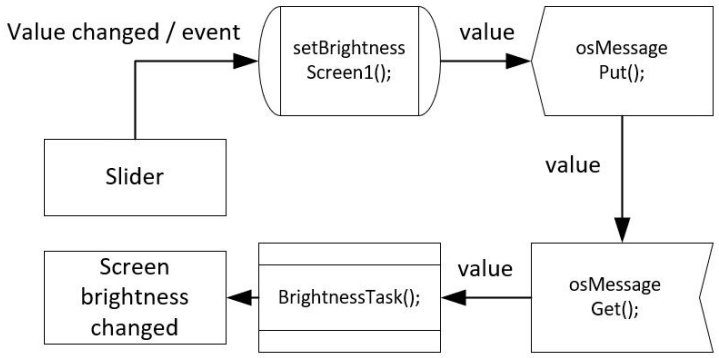
4.1.3.2 BoxProgress

BoxProgress widgeten finnes under *Progress Indicators* gruppen. TouchGFX Designer genererer kode i BatteryScreenViewBase.cpp i likhet med TextArea widgeten. Her settes bla. XY-posisjon, stil, rekkevidde, retning, steg, bakgrunn, farge og startverdi. For å gi verdier som skal visualiseres til BoxProgress widgeten bruker vi navnet gitt til widgeten og funksjonen *WidgetName.setValue* definert i BoxProgress.hpp, se figure 22.

4.1.3.3 CircleProgress

I likhet med BoxProgress kan CircleProgress finnes i *ProgressIndicators* gruppen, og ellers er funksjonaliteten nogenlunde lik. Posisjonering, progresjonsindikator, rekkevidde, senter, radius, strekbredde, start og stopp vinkel, bakgrunnsfarge og startverdi blir her konfigurert av TouchGFX Designer i BatteryScreenViewBase.cpp. Videre sendes verdien widgeten skal visualisere gjennom setValue() funksjonen definert i CircleProgress.hpp, se figure 22.

4.1.3.4 Slider - Brightness



Figur 22: BatteryScreenView::SetData()

Slideren på venstre side av BatteryScreen skjermen er ment for å justere lysstyrken på skjermen. Selve widgeten "Slider" er i vårt prosjekt satt opp med en interaction som trigger en virtuell funksjon (setBrightnessScreen1) som kan bli funnet i BatteryScreenViewBase.hpp. Denne funksjonen blir kalt inni callback-handleren til Slideren som tar imot de nye verdiene fra slideren og legger disse inn på en meldingskø. Denne meldingskøen blir motatt av brightnessTask() tråden i Brightness.cpp filen. I Brightness.cpp filen er det satt opp bruk av en tråd med RTOS, og i brightnessTask() tråden blir det satt delays samt on og off tider for å styre lysstyrken med pulsbreddemodulasjon-simulering . Vår implementasjon av lysstyrke justeringen er inspirert STM sin egen LCD driver der de bruker en hardware timer til å generere et PWM signal.

4.1.3.5 Buttons

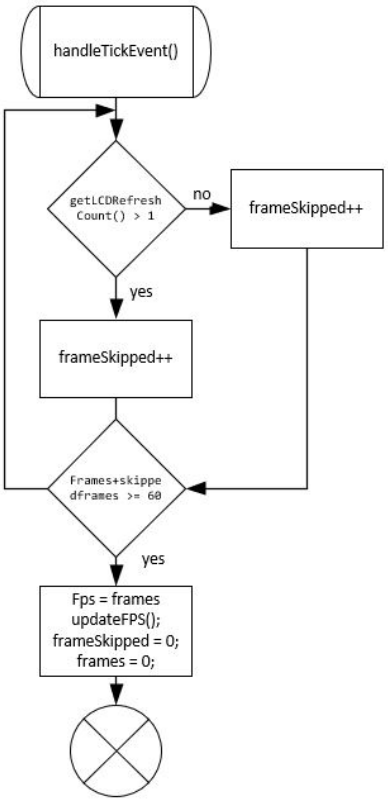
I dette prosjektet har knappene til hensikt å ta brukeren videre til neste eller forrige skjerm. Dette oppnås ved å opprette interaksjoner og triggere på de gjeldende knappene. Her setter man opp trigger på "button is clicked", velger så hvilken knapp dette skal gjelde (kilde). Deretter velger man hvilken handling triggeren medfører (i vårt tilfelle - skifte skjerm) og hvilken skjerm man går videre til. Til slutt velger man hvilken overgangsanimasjon man ønsker (Slide/Cover) og helt til slutt hvilken retning (øst/vest) man ønsker

at animasjonen skal gå i. Posisjon, utseende og funksjonalitet til knappene blir deklarert i BatteryScreenViewBase.hpp og funksjonaliteten satt i BatteryScreenViewBase.cpp.

4.1.3.6 Frames per sekund

For å kunne ha en enkel oversikt over ytelsen i vår mikrokontroller mens den kjører ble det valgt å implementere en frames per sekund med en TextArea widget som bruker Wildcard (se figure 23). Det blir så definert en funksjon updateFPS() med tre tilhørende variabler: frameSkipped, frames og fps.

Disse variablene blir initialisert til 0 ved oppstart i BatteryScreenView::BatteryScreenview(). Deretter blir funksjonen updateFPS(); kalt i BatteryScreenView::setupScreen(), og i HandleTickEvent() funksjonen sjekkes det etter antall Vsync interrupts mellom nåværende rendering og forrige. Dersom dette tallet er større enn 1, har vi nå hoppet over en frame og frameSkippedCounter blir inkrementert. Dersom antall Vsync interrupts er lik 1 blir variabelen frames inkrementert. Til slutt sjekkes det om total verdi av frames og framedSkipped er større eller lik 60 (oppdateringsfrekvensen). Dersom ja blir fps nå satt lik frames og frameSkipped og frames blir satt til null igjen og prosessen starter på nytt.



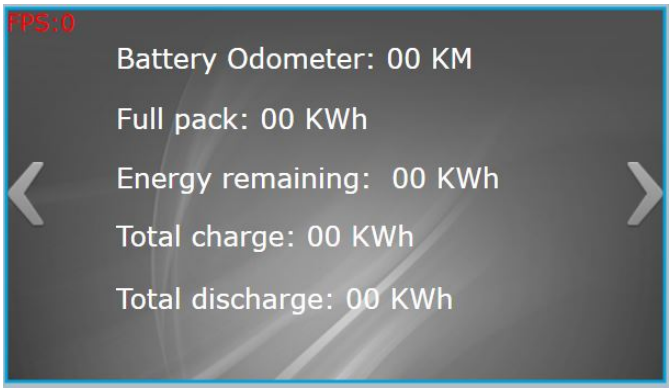
Figur 23: FPS

4.1.4 Historical Stats

Historical Stats skjermen viser en grafisk fremstilling av historiske data fra batteripakken i en Tesla Model S (se fig 24). De ulike dataene fremstiller kjørelengde på batteriet (kilometer), nominell full energi (kWh), gjenværende energi (kWh), total oppladning og utladning (kWh). Dataene er som følger:

- Battery Odometer (KM)
- Nominal Full Pack (kWh)
- Energy Remaining (kwh)
- Total Charge (kWh)
- Total Discharge (kWh)
- FPS

Visualiseringen av dataene på denne skjermen ble bestemt gjort kun via TextArea widgets da gruppen ikke fant det passende å visualisere dette på en annen måte. HistoricalStatsScreen består derfor kun av FPS (TextArea), de ulike dataene (TextArea) samt én knapp for forrige skjerm og én knapp for neste skjerm (Buttons).

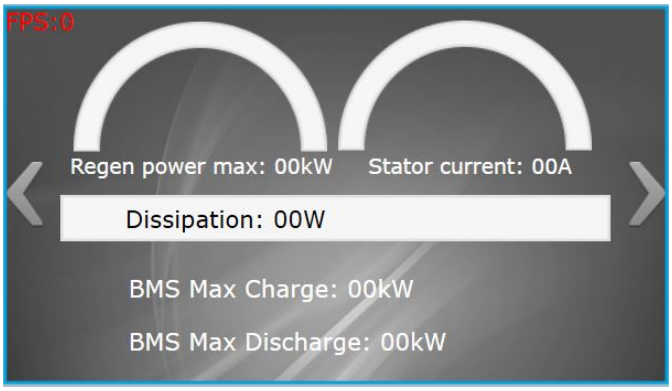


Figur 24: HistoricalStatsScreen

4.1.5 ChargeDischarge Screen

ChargeDischarge skjermen (se figure 25) visualiserer regenereringskraften til AC motoren som sitter i bilen samt strømmen i Statoren (huset til motoren), tap (dissipation) samt maks lading- og utladningeffekt. Her er dataene visualisert gjennom CircleProgress (RegenPowerMax, StatorCurrent), BoxProgress (Dissipation) samt TextArea (BMSMaxCharge, BMSMaxDischarge, FPS).

- Regen Power Max (kW)
- Stator Current (A)
- Dissipation (W)
- BMS Max Charge (kW)
- BMS Max Discharge (kW)
- FPS.



Figur 25: ChargeDischargeScreen

4.1.6 Power Screen

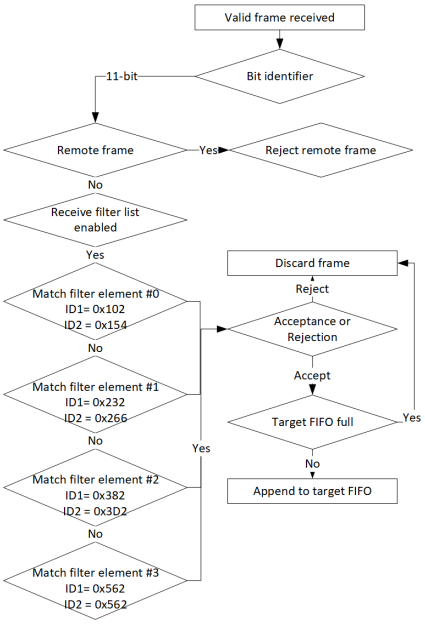
I PowerScreen (se figure 26) visualiseres kraft i systemet til Tesla Model S. Her er det brukt 4 CircleProgress widgets samt TextArea widget for FPS. Dataene representerer posisjonen til pedalen, kraften ut fra motoren, mekanisk kraft generert, dreiemoment produsert og strøm ut fra batteriet. De ulike dataene er som følger:

- Pedal Position (%)
- Drive Power (kW)
- Mech Power (kW)
- Rear Torque (Nm)
- Pack Current (A)
- FPS

Figur 26: PowerScreen

4.1.7 CAN FD

På STM32H7B3I-DK ligger det en CAN-FD Controller, og den bruker MCP2562FD CAN Transciever. Siden vi skal lese fra en Tesla Model S sin CAN Bus som har bit-rate på 500 Kbit/second, så har vi konfiguert vår node til å lese i samme bit-rate. Den Can Busen bruker samme bit-rate for både Arbitrary og Data fasene. Vi er interessert i å hente nyttig data fra CAN Bus 3, derfor bruker vi noen innebygde filtre for å ta imot de meldingene med ønsket ID. Vi har konfiguert CAN Controlleren til å se bort fra alle andre IDer og nekte Remote frames (Data forespørsel).



Figur 27: FDCAN Filters

4.1.8 CAN BUS

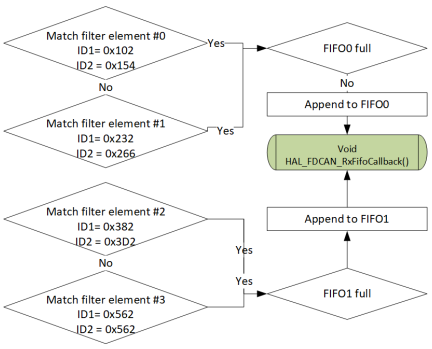
Tesla S har flere CAN Buser men vi er interessert i batteri informasjon og den bruker CAN 3. Kortet blir koblet til CAN 3 gjennom OBD2 konnektor. Kortet blir forsynt fra 12 V batteriet på bilen.

4.1.9 IRQ

Når en ønsket melding med ID som matcher en av filterne blir lagret i RX FIFO bufferet til FDCAN, blir det generert en Interrupt Request som igjen kaller en Callback function.

4.1.10 Callback

Denne blir kalt fra IRQ til FDCAN. I den funksjonen blir CAN medlingen lagret i Listen "CAN" som ny node "CANData".



Figur 28: FDCAN Filters

4.1.11 MVP Design

4.1.11.1 Model Class

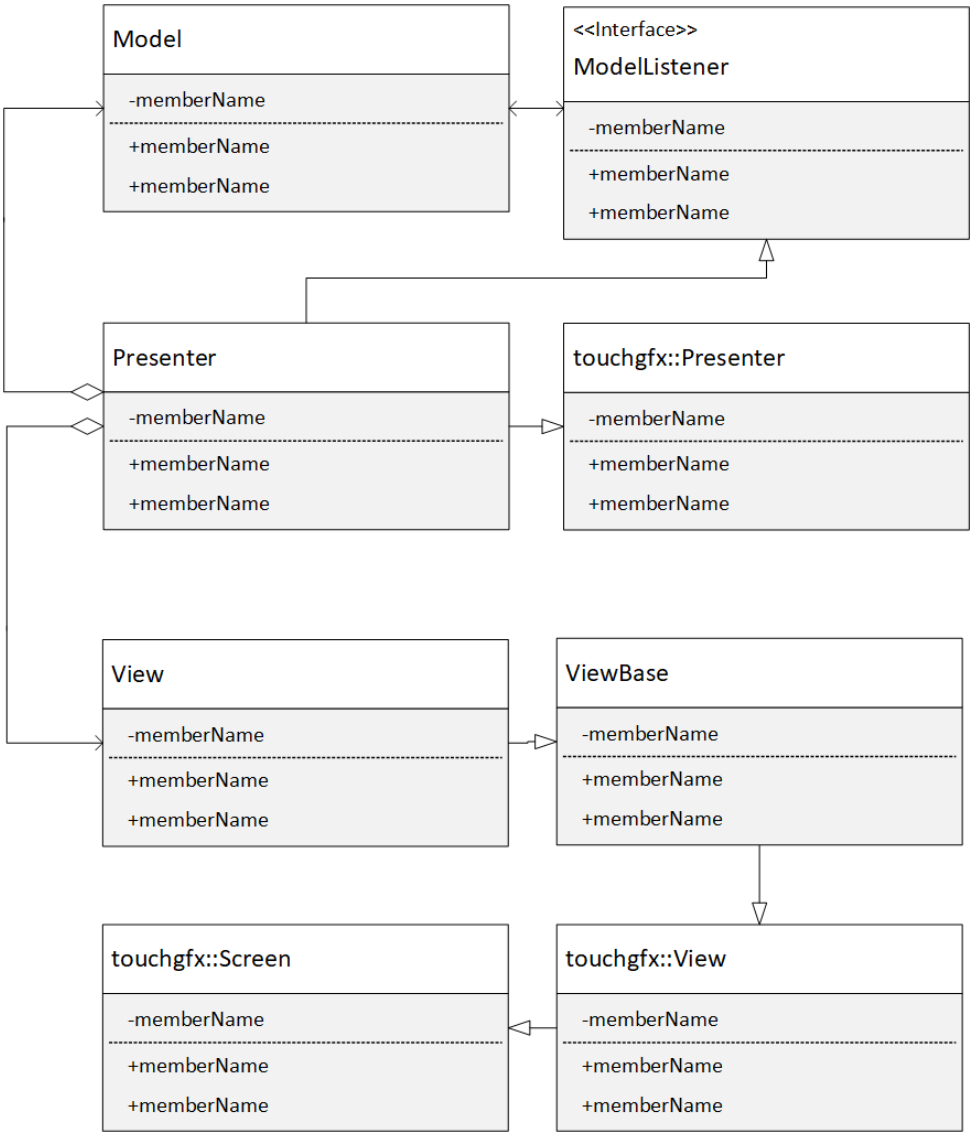
I CANClass.cpp er det definert en Node klasse (CANData) og en liste klasse (CAN). Meldingen som blir lest inn blir dyttet på listen ved hjelp av funksjonen push_back(). Deretter når meldingen skal leses, blir funksjonen switch_decode() kalt fra HandleTickEvent() i Model klassen for å dekode meldingene slik at den gir oss behandlet data. Når meldingen så er lest, blir den slettet av funksjonen remove(). Etter at Model klassen har hentet dataene fra listen, varsler han den nåværende aktive Presenteren om det har kommet nye data gjennom ModelListener Interface via en virtuell funksjon notifyNewCANDataChanged().

4.1.11.2 Presenter

Ved å bruke overnevnte implementasjon i Model klassen er det nå tilrettelagt for Presenter slik at denne blir varslet, slik at Presenter kan videresende denne informasjonen til View klassen ved implementering av notifyNewCANDataChanged() funksjonen. Etter dataene er mottatt må disse oppdateres til skjermen. Og det skjer ved å kalle setData() funskjon i view klassen.

4.1.11.3 View

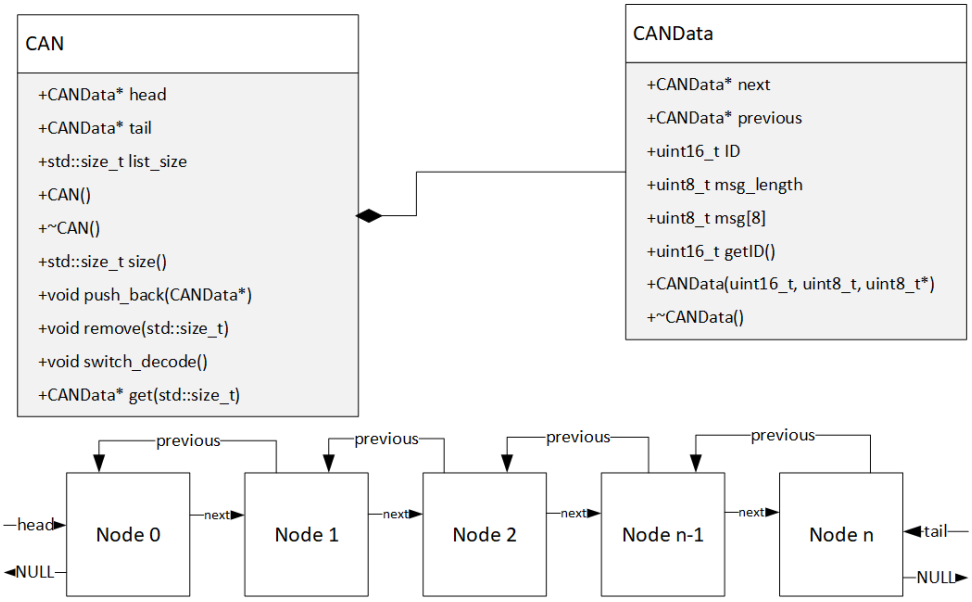
View klassen får sine UI elementer oppdatert gjennom Presenter klassen ved at presenter får en hendelsesvarsling fra Model klassen gjennom notifyNewCANDataChanged() funksjonen. I setData() funksjonen blir de ulike variablene knyttet til UI elementene satt samme verdienene pekerene i strukten peker til.



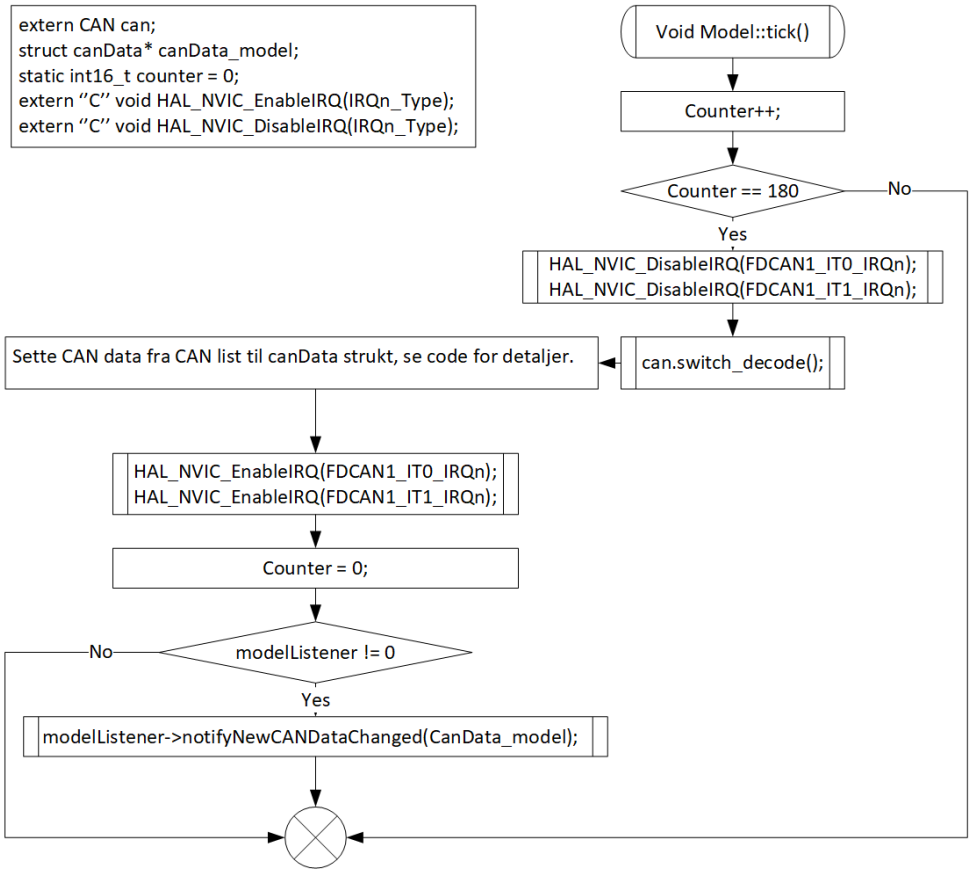
Figur 29: Generell UML av MVP i TouchGFX

4.2 Lese fra CANBus

Vi er interessert i syv IDer fra Tesla S sin CAN bus. Disse meldingene blir sendt til CAN Bus med forskjellige frekvenser mellom 10 og 100 Hz. På grunn av det, har vi satt en teller i Model::tick() funksjonen i Model klassen som igjen kaller can.switch_decode() for å dekode og så hente CAN meldingene hver tredje sekund, se figure 31. Vi kunne ha brukt FreeRTOS sine kommunikasjonsmekanismer som meldingskø og da tar FreeRTOS seg av allokering av minnet. Siden dette er et C++ prosjekt så brukte vi en doubly linked list for å lagre og dele rådataene, se figure 30, på den måten blir ressursene brukt på en effektiv måte. Før modellen får tilgang til listen, blir interrupt handleren til FDCAN slått av slik at det kun er modellen som får tilgang til listen.



Figur 30: UML til CAN Doubly List

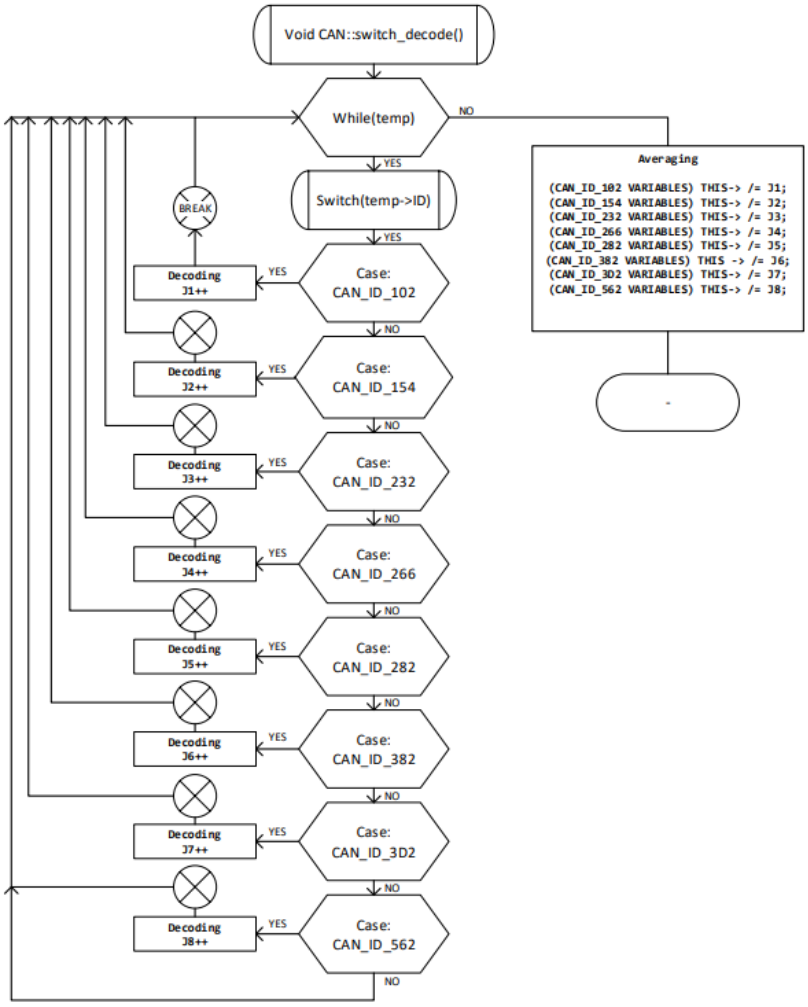


Figur 31: Model::tick()

De syv IDene på Tesla S CAN Bus 3 er:

1. ID 0x102: BMS Current and Voltage
2. ID 0x154: Rear drive unit measurment info
3. ID 0x232: Battery Power limits

4. ID 0x266: Rear drive unit power info
5. ID 0x382: Battery Energy Status
6. ID 0x3D2: Battery Lifetime Energy Status
7. ID 0x562: Battery Odometer



Figur 32: Switch Decoder SDL

4.3 Kode hierarki

I figuren under har vi listet de forskjellige filer som vi har laget/redigert i applikasjonen vår :

Folder	Subfolder(s)	File(s)
Core	Inc	FreeRTOSConfig.h main.h stm32h7xx_hal_conf.h stm32h7xx_it.h
	Src	freertos.c main.cpp stm32h7xx_hal_msp.c stm32h7xx_it.c
MyApplication		app_can.cpp app_can.hpp Brightness.cpp Brightness.hpp CANClass.cpp CANClass.hpp CANDataStruct.hpp timer.cpp timer.hpp
TouchGFX	gui/include/gui	BatteryScreenPresenter.hpp BatteryScreenView.hpp ChargeDischargeScreenPresenter.hpp ChargeDischargeScreenView.hpp HistoricalStatsPresenter.hpp HistoricalStatsView.hpp PowerScreenPresenter.hpp PowerScreenView.hpp Model.hpp ModelListener.hpp
	gui/src/gui	BatteryScreenPresenter.cpp BatteryScreenView.cpp ChargeDischargeScreenPresenter.cpp ChargeDischargeScreenView.cpp HistoricalStatsPresenter.cpp HistoricalStatsView.cpp PowerScreenPresenter.cpp PowerScreenView.hpp Model.cpp
	generated/gui_generated/include /gui_generated	BatteryScreenViewBase.hpp ChargeDischargeScreenViewBase.hpp HistoricalStatsViewBase.hpp PowerScreenViewBase.hpp
	generated/gui_generated/src	BatteryScreenViewBase.cpp ChargeDischargeScreenViewBase.cpp HistoricalStatsViewBase.cpp PowerScreenViewBase.cpp
	target	OSWrappers.cpp STM32DMA.cpp.cpp STM32DMA.hpp TouchGFXConfiguration.cpp TouchGFXGeneratedHAL.cpp TouchGFXGeneratedHAL.hpp STM32TouchController.cpp STM32TouchController.hpp TouchGFXGPIO.cpp TouchGFXHAL.cpp TouchGFXHAL.hpp

Figur 33: Kode hierarki

5 Testing og validering

Vi har brukt som hovedregel debuggeren på STM32 og STM32CubeIDE i tilleg til bruk av IXXAT USB-to-CAN analyser for å sende og motta CAN meldinger.

Funskjon	Test metode	Pass/Fail	Kommentar
Teste CAN kontrolleren og transcieveren på STM32	Internal loop ved å sende og motta samme medling på STM32 og debugge på STM32CubeIDE	PASS	vi brukte eksempel kode fra STM32H7B3I-DK firmware pakken
Lese CAN medlinger sendt fra canAnalyser Mini	sende og motta CAN meldinger ved bruk av IXXAT USB-to-CAN analyzer	PASS	
Lese CAN meldinger på Tesla S	koble til CAN Bus 3 på Tesla S bil	PASS	
Flashe assets på ekstern flash	Konfiguere den eksterne NOR OSPI Flash på STM32 kortet og flashe koden på kortet	PASS	Vi måtte bruke den eksterne flashen siden den interne var for liten for å lagre grafikken
Lagre framebufferne på ekstern SDRAM	Konfiguere den eksterne SDRAM og flashe koden på kortet	PASS	Vi brukte den eksterne SDRAM for å gi CPUen mer resurser og for å lagre opptil 65k CAN meldinger
Integriteten på dekodede dataer på skjermer	Ved å samenligne med dataer fra en lignende produkt ScanMyTesla	PASS	Det ble sammenlignet med ScanMyTesla app sine dekodede data på mobilappen
FDCAN filterne	Ved å sende CAN meldinger med andre IDer enn de som er satt opp til å tas imot i filterne ved hjelp av IXXAT USB-to-CAN analyzer	PASS	
Reject Remote Frames	Ved å sende CAN meldinger med forespørsel av data ved hjelp av IXXAT USB-to-CAN analyzer	PASS	
Lagring av rådata	Ved å fjerne remove funksjonen for å sjekke for mange CAN meldinger er det mulig å ta vare på	PASS	det ble lagret 65k CAN meldinger før CPUen gikk tom for RAM
Test av lysstyrke justering	Ved å flashe koden og teste slideren	PASS	

6 Diskusjon

6.1 Design

Hovedoppgaven i faget DAT220-G20H tillot deltakerne å velge mellom to alternativer: Game Development eller Embedded Development. Da gruppemedlemmene har erfaring fra embedded development fra tidligere fag og har et sterkt ønske om å jobbe videre innenfor dette feltet etter endt utdanning ble Embedded Development et naturlig valg.

Gruppen var klar over diverse utfordringer i forbindelse med realiseringen av dette prosjektet da det krever en kombinasjon av C og C++ programmering. Medlemmene har kompetanse innenfor C programmering fra tidligere fag og noe erfaring med C++ programmering fra de obligatoriske oppgavene innenfor faget DAT220-G20H, men ingen erfaring med å kombinere disse to språkene i ett og samme produkt med tanke på kompilering og linking.

Den innledende tiltenkte kundegruppen var eiere av Tesla Model S. Koden er derimot strukturert på en slik måte at den enkelt kan tilpasses andre bilmerker også. Vi har fokusert på å fremstille data som det tenkes at kunden kan nytte seg av men som ikke blir vist på bilens eget UI. Disse dataene blir fordelt utover kategorier som er separert på ulike skjermer. I tillegg er det på første skjerm lagt til en manuell lysstyrke justering gjennom en Slider Widget.

Da gruppen ikke har noen tidligere erfaring fra grafisk design av User Interfaces eller grafikk design ble det forsøkt etter beste evne å fremstille grafikken på en nøytral og logisk måte. Fargevalgene til de ulike elementene og bakgrunner er forsøkt gjort på en måte som tillater brukeren å enkelt lese verdiene som blir vist på skjermene samtidig som vi ønsket å være varsomme med å unngå for sterkt lys som potensielt kunne vært forstyrrende for føreren av kjøretøyet.

6.1.1 STM32H7B3I-DK

Da denne oppgaven ble gitt hadde gruppen ingen erfaring med å jobbe med STM32H7B3I-DK mikrokontrolleren eller STM32CubeIDE editoren. Oppgaven ble planlagt ut fra et CANBUS deciphering dokument [6] hvor dekodingen av de ulike IDene og dataene har blitt gjort. I hovedtrekk går prosjektet ut på å lese utvalgte data fra Tesla Model S' CANBUS for deretter å behandle og dekode meldingene og til slutt vise disse verdiene grafisk på en skjerm.

6.1.2 Product Vision

I gruppens Product Vision ble produktet rettet generelt mot El-bil markedet. Da gruppen kun hadde tilgang på én type El-bil (Tesla Model S) måtte produktet bli rettet mot dette spesifikke merket da ulike produsenter sender CANBUS meldinger med ulike IDer og meldingsformater. I hoved trekk var det hovedbatteriets temperatur som var av interesse for oss og våre potensielle kunder. Batteritemperaturen blir ikke vist i noen bilers UI og har stor påvirkning på effekten av hurtigladere.

6.1.3 RTOS

TouchGFX biblioteket bruker FreeRTOS. TouchGFX bruker en meldingskø for å synkronisere med display kontrolleren og en semafor for å beskytte aksessen til framebufferet, og dette er gjort ved bruk av OSWrappers klasse og den blir generert via TouchGFX Generator. Vi har brukt CMSIS v1 som benytter FreeRTOS i vår applikasjon. Alternativet

var å ikke ha noen RTOS, i stedet kunne vi bruke while-looper, men det var ikke aktuelt med tanke på bruker grensesnitt opplevelsen.

6.2 Utfordringer

6.2.1 STM32CubeIDE med TouchGFX Designer

Innledningsvis opplevde gruppen en del utfordringer i forbindelse med oppsett av STM32H73BI-DK mikrokontrolleren og STM32CubeIDE. Det ble brukt 3 hele dager med diverse forsøk på manuelt oppsett av .IOC filen og diverse konfigureringer til prosjektet for å få STM32CubeIDE og STM32H7B3I-DK til å kommunisere riktig. Mye av utfordringene her ligger i at både STM32CubeIDE og STM32H7B3I-DK er relativt nye tilskudd på markedet og mangelen på dokumentasjon rundt oppsett og konfigurasjon for vår kombinasjon av mikrokontroller og IDE. Det ble omsider funnet en kilde til et fungerende eksempel [2] som ble brukt som et utgangspunkt for prosjektet.

For å bli kjent med STM32CubeIDE, STM32H7B3I-DK og TouchGFX Designer ble det den første uken kun gjennomgått demoer, eksempler og uttesting av funksjonaliteter gruppen potensielt kunne nytte seg av i gjennomføringen av oppgaven.

I tillegg opplevde gruppen store utfordringer i forbindelse med oppsett av FDCAN perifer enheten. Gruppen leste i Refernce manual til mikrokontrolleren og studerte FDCAN eksempler som var tilgjengelig for andre STM32 kort (ST har ikke laget noen FDCAN eksempel kode for STM32H7B3I-DK). Vi slet med å få FDCAN til å fungere på grunn av forbindelser mellom FDCAN kontrolleren og FDCAN transcieveren var en åpen solder jumper som ligget under skjermen og det var ikke synlig. Vi fikk tips av faglæreren om å lodde de tre solder jumperne, og når dette var gjort var FDCAN fungerende.

6.2.2 Bruk av C og C++ i samme applikasjon

Det falt naturlig for gruppen å skrive funksjonene i C, i tillegg alle driverne til STM32 kortet er skrevet i C. Vi slet med å lenke de forskjellige source filene sammen, selv om vi hadde valgte C++ prosjekt i STM32CubeIDE ble main opprettet som main.c og det viste seg at vi må kompilere main med C++ kompileren [4]. Vi testet wrapping av C++ funksjoner for å kunne benytte de i C source filer men det ble uoversiktlig for oss og vi valgte å bruke .cpp source filer i stedet for .c .

6.2.3 Git

Atlassin BitBucket ble brukt i dette prosjektet for versjonskontroll og opplasting av filer til prosjektet. Hver bruker har her vært tiltenkt å laget en egen branch for sine tildelte sprints i Jira Kanban og merge disse med master branch når utviklingen av sin branch er ferdigstilt. Gruppen har noe erfaring med git fra tidligere fagemner i forbindelse med innlevering av oppgaver, men hadde liten til ingen erfaring med å bruke Git i større prosjekter. Det har derfor tidvis oppstått versjons og filkonflikter som har tatt en del tid å korrigere. I tillegg har det ved flere anledninger oppstått utilsiktet arbeid direkte på master branch, noe som skyldes manglende erfaring med arbeid i et slikt miljø. Under hele prosjektarbeidet har det blitt lagret lokale backuper av de ulike prosjektversjonene på gruppens eksterne harddisk for ekstra sikkerhet. Gruppen annerkjenner fordelen ved korrekt bruk av GIT i større prosjekter for versjonskontroll og korrektur eller tilbakemeldinger før merging i master branch.

6.2.4 Jira Kanban

Gruppen har valgt Jira Kanban for å opprette deloppgaver / sprints underveis. Gruppen har noe erfaring med bruk av Scrum for å sette opp ukentlige sprints og time-logging i tidligere prosjektarbeider. Arbeidsfordelingen ut fra gitte sprints har fungert bra med unntak i noen tidsestimater som ikke har blitt overholdt i forhold til planen.

6.2.5 Fremtidige forbedringspotensialer

6.2.5.1 TX

Gruppens løsning på denne oppgaven baserer seg utelukkende på lesing og dekoding av data fra CAN bus. Det sees også muligheter til å kunne utvide løsningen til også å sende (tx) meldinger på CAN busen i diagnoseøyemed samt muligheter for å potensielt kunne emulere front motor på modeller med kun én motor, sett i perspektiv av et tiltenkt fremtidig prosjekt hvor det tenkes å 'lure' kjøretøyet til å tro at det har 2 motorer for økt ytelse.

6.2.5.2 OTA Oppdateringer

Med nåværende løsning er gruppen avhengig av å fysisk koble seg på mikrokontrolleren for å oppdatere og flashe programvaren. En fremtidig mulighet her kan være å tilrettelegge for en WiFi periferienhet som i kombinasjon med en perfier SD enhet kunne koblet til en server satt opp av gruppen med den nye programvare oppdateringen, laste ned og til slutt flashe den på mikrokontrolleren. Da de fleste eiere av Tesla Model S har sin bil koblet til sitt eget WiFi nettverk hjemme sees denne løsningen å være praktisk gjennomførbar.

6.2.5.3 GUI

Nåværende GUI er satt opp på en enkel måte. Gruppen ser forbedringspotensialer i den visuelle oppbyggingen av skjermene i kombinasjon med ikoner, farger og lysstyrke.

6.2.5.4 Fysisk produktstørrelse

Nåværende produkt der vi bruker et Development Kit (DK) har unødvendig mange komponenter i forhold til det som er nødvendig for funksjonaliteten. Ved å innhente kun de nødvendige komponentene og designe eget PCB utlegg kan størrelsen på produktet reduseres betraktelig.

6.2.5.5 Lysstyrke

Ved videre utvikling av produktet er lysstyrke justering tiltenkt å foregå automatisk ved at en fotodiode registrerer lysstyrken i omgivenselene og systemet da automatisk justerer lysstyrken på skjermen.

6.2.5.6 Lagring

Det kan også være intressant for brukeren av produktet å kunne lagre ønskede verdier. Intressante verdier å lagre her kan være f.eks. Maks batteri temperatur, maks mekanisk kraft, maks strøm ut av batteri og lignende. Lagringen kunne blitt utført på en perfier SD enhet.

6.2.5.7 Meny og instillinger

Ved videre utvikling ville gruppen lagt til en ekstra 'meny' skjerm hvor brukeren kan endre på ulike instillinger. Herunder tenkes det at lysstyrke, tekststørrelse og fargeinstillinger mm. naturlig faller inn.

6.2.5.8 Bitbucket og Jira Kanban

Gruppen har etter beste evne brukt Bitbucket i kombinasjon med sprints på Jira Kanban for å planlegge og holde styr på løpende versjoner av programmet. Det annerkjennes her rollen korrekt planlegging og utføring av sprints og periodearbeid har for tidsbruken på et prosjekt. I fremtidige prosjekter vil gruppen dra med seg lærdommen ervervet fra både dette og andre prosjekter for kunne estimere tidsbruk mer nøyaktig.

7 Konklusjon

7.1 Prosess

Gruppen leverte inn sin 'Produkt Visjon' i faget Dat220-G20H 'Softwareutvikling 2' den 9.10.2020 og started umiddelbart med rollefordelinger. Oppgaven ble splittet opp i mindre deloppgaver og fordelt utover gruppemedlemmene. Rollefordelingen ble fordelt på følgende måte:

- Mohammed Salih Alrawi
  - Git master
  - Koordinerings ansvarlig
- Morten Jacobsen
  - Kanban master
  - Referent
  - Møte-leder

Det var en forutsetning fra faget om å bruke Bitbucket for opp- og nedlasting samt lagring av ulike software-versjoner. I tillegg ble det brukt Jira Kanban for å føre timer og lage ukentlige sprints utover i utviklingsfasen. Dette viste seg å fungere meget bra for medlemmene i gruppen og var strengt tatt helt nødvendig dersom campus skulle bli stengt ned som følge av COVID-19. Medlemmene har sett seg fornøyde med oppgavefordelingen hvor vi hver for oss har designet ulike blokket og testet disse før sammensying til et ferdig prosjekt. Gruppen innledet arbeidet med en ide om å motta og tolke meldinger fra en Tesla Model S CAN bus ved hjelp av STM32H7B3I-DK programmert via STM32CubeIDE. Innledningsvis ble det fokusert på å få kommunikasjonen mellom IDE og mikrokontroller og mikrokontroller og CANFD fungerende. Når kommunikasjonen var på plass ble fokuset satt over til funksjonalitet, effektivitet og flyt i brukeropplevelsen.

7.2 Produkt

Ønsket til gruppen var å representere utvalgte meldinger fra CAN på en enkel, oversiktelig og forståelige måte samtidig som brukeren fikk tilgang på informasjon som normalt ikke er synlig for eieren av kjøretøyet. I tillegg skal brukeren kunne interagere med produktet på en enkel måte for å selv bestemme hva slags informasjon som er synlig til en hver tid. Da Tesla Motors Inc. har for vane å holde teknisk informasjonen om sine produkter tett til brystet måtte gruppen ty til forumer og diverse artikler for å finne nødvendig dokumentasjon for å starte dekodingen.

Referanser

[1] Borhanuddin Mohd Ali. *CAN Evolution and Applications*. 01.01.2020. URL: https://www.researchgate.net/publication/228957774_Review_of_Researches_in_Controller_Area_Networks_Evolution_and_Applications/figures?lo=1.

[2] Jan Cumps. «STM32H7B3I - Create a TouchGFX Project with support for the Designer, CubeIDE and Debugger - Pt 1: Screen Works». I: (). 04.08.2020. URL: https://www.element14.com/community/groups/embedded/blog/2020/08/04/stm32h7b3i-create-a-touchgfx-project-with-support-for-the-designer-cubeide-and-debugger.

[3] Armin Bassemir Florian Hartwich og Robert Bosch GmbH. 04.11.1999. URL: https://www.kvaser.com/wp-content/uploads/2014/08/boschcia99-can-bit-timing.pdf.

[4] Standard C++ Foundation. 01.12.2020. URL: https://isocpp.org/wiki/faq/mixing-c-and-cpp.

[5] FreeRTOS. *FreeRTOS*. 01.11.2020. URL: https://www.freertos.org/.

[6] Jason Hughes. *Tesla Model S CAN Bus Deciphering*. 15.01.2016. URL: https://skie.net/uploads/TeslaCAN/Tesla.

[7] IXXAT. *canAnalyser 3 Suite*. 01.01.2020. URL: https://www.ixxat.com/products/products-industrial/tools-overview/cananalyser/cananalyser-lite-standard?ordercode=1.12.0133.31000.

[8] IXXAT. *USB-to-CAN V2 Active USB interface*. 01.01.2020. URL: https://www.ixxat.com/products/products-industrial/can-interfaces/usb-can-interfaces/usb-to-can-v2-professional?ordercode=1.01.0281.12001.

[9] ST. 01.12.2020. URL: https://www.st.com/en/evaluation-tools/stm32h7b3i-dk.html#documentation.

[10] ST. *FDCAN peripheral on STM32 devices AN5348*. 01.10.2019. URL: https://www.st.com/resource/en/application_note/dm00625700-fdcan-peripheral-on-stm32-devices-stmicroelectronics.pdf.

[11] ST. *STM32CubeIDE*. 01.11.2020. URL: https://www.st.com/en/development-tools/stm32cubeide.html.

[12] STMicroelectronics. *Display*. URL: https://support.touchgfx.com/docs/development/hardware-selection/hardware-components/hardware-selection-display.

[13] Pico Technology. *CAN and CAN FD bus decoding*. 01.01.2020. URL: https://www.picotech.com/library/oscilloscopes/can-bus-serial-protocol-decoding.

[14] TouchGFX. *BoxProgress*. 01.11.2020. URL: https://support.touchgfx.com/docs/development/ui-development/ui-components/progress-indicators/box-progress/.

[15] TouchGFX. *Buttons*. 01.11.2020. URL: https://support.touchgfx.com/docs/development/ui-development/ui-components/buttons/button/.

[16] TouchGFX. *CircleProgress*. 01.11.2020. URL: https://support.touchgfx.com/docs/development/ui-development/ui-components/progress-indicators/circle-progress/.

[17] TouchGFX. *Code structure*. 01.10.2019. URL: https://support.touchgfx.com/docs/development/ui-development/software-architecture/code-structure.

[18] TouchGFX. *Graphhics Engine*. 01.11.2020. URL: https://support.touchgfx.com/docs/basic-concepts/graphics-engine.

[19] TouchGFX. *Main Loop*. 01.11.2020. URL: https://support.touchgfx.com/docs/basic-concepts/graphics-engine#main-loop.

[20] TouchGFX. *ModelViewPresenter Design Pattern*. 01.10.2019. URL: https://support.touchgfx.com/docs/development/ui-development/software-architecture/model-view-presenter-design-pattern.

[21] TouchGFX. *Slider*. 01.11.2020. URL: https://support.touchgfx.com/docs/development/ui-development/ui-components/miscellaneous/slider/.

[22] TouchGFX. *Texts and Fonts*. 01.10.2019. URL: https://support.touchgfx.com/docs/development/ui-development/touchgfx-engine-features/texts-and-fonts#wildcards.

[23] TouchGFX. *TouchGFX Engine*. 01.11.2020. URL: https://support.touchgfx.com/docs/development/touchgfx-hal-development/touchgfx-generator.

[24] TouchGFX. *TouchGFXAL*. 01.11.2020. URL: https://support.touchgfx.com/docs/development/touchgfx-hal-development/touchgfx-al-development-introduction/.

[25] Wikipedia. 01.12.2020. URL: https://en.wikipedia.org/wiki/CAN_bus.

[26] Wikipedia. *CAN FD*. 20.11.2020. URL: https://en.wikipedia.org/wiki/CAN_FD.

[27] Wikipedia. *FreeRTOS-Wikipedia*. 01.11.2020. URL: https://en.wikipedia.org/wiki/FreeRTOS.