

AARHUS SCHOOL OF ENGINEERING

ELECTRONIC ENGINEERING

E4PRJ

---

# Detaljeret Software Design

---

*Author:*

Nicolai GLUD

Johnny KRISTENSEN

Rasmus LUND-JENSEN

Mick HOLMARK

Jacob ROESEN



4. december 2012

# Indholdsfortegnelse

---

<b>Kapitel 1</b>	<b>Indledning</b>	<b>4</b>
1.0.1	Formål . . . . .	4
1.0.2	Reference dokumentation . . . . .	4
<b>Kapitel 2</b>	<b>Kontrolinterface</b>	<b>5</b>
2.0.3	Modulets ansvar . . . . .	5
2.0.4	Klassediagram . . . . .	5
2.1	Metode- og klassebeskrivelser . . . . .	7
2.1.1	MainWindow . . . . .	7
2.1.2	Kontrolinterface . . . . .	11
2.1.3	DataSet . . . . .	11
2.1.4	manuDialog . . . . .	12
2.1.5	Styringsmodul . . . . .	13
2.1.6	VBTE . . . . .	14
2.1.7	Sensor . . . . .	14
2.1.8	RS232 . . . . .	14
<b>Kapitel 3</b>	<b>Databasen</b>	<b>16</b>
3.0.9	Modulets Ansvar . . . . .	16
3.0.10	Klassediagrammer . . . . .	16
3.0.11	Globale variabler . . . . .	18
3.0.12	Funktionsbeskrivelser . . . . .	18
3.1	Design . . . . .	20
3.1.1	Server . . . . .	20
3.1.2	TCP server . . . . .	21
3.1.3	Web-side . . . . .	22
3.2	Metodebeskrivelse . . . . .	23
3.2.1	TCP KI . . . . .	23
3.2.2	TCP database . . . . .	23
<b>Kapitel 4</b>	<b>SM</b>	<b>24</b>
4.1	Klassens ansvar . . . . .	24
4.2	Klassediagram . . . . .	24
4.3	Funktioner . . . . .	24
4.4	Variabler . . . . .	25
4.5	Funktionsbeskrivelser . . . . .	25
4.5.1	Init . . . . .	25
4.5.2	Levelsensor . . . . .	25
4.5.3	autoReg . . . . .	25
4.5.4	I2C_Kom . . . . .	26
4.5.5	KI_KOM . . . . .	26

4.6	Eventuelle Sekvensdiagrammer og state machines . . . . .	27
<b>Kapitel 5</b>	<b>VBTE</b>	<b>28</b>
5.1	Modulets ansvar . . . . .	28
5.2	Klassediagram . . . . .	28
5.3	Globale variabler . . . . .	29
5.4	Metode- og klassebeskrivelser . . . . .	29
5.4.1	Init . . . . .	29
5.4.2	Valve . . . . .	29
5.4.3	Dist . . . . .	30
5.4.4	I2CHandle . . . . .	30
5.4.5	PSoC-API::ADC . . . . .	31
5.4.6	State Machine . . . . .	31
5.4.7	Timing Diagram . . . . .	32
5.4.8	Interrupt rutiner . . . . .	32

# Indledning 1

---

Dette dokument beskriver det detaljerede SW-design for BROS, som er fastlagt ud fra dokumenterne kravspecifikation og systemarkitektur.

## 1.0.1 Formål

Formålet med dokumentet er:

- At fastlægge systemets detaljerede softwarestruktur udfra kravene specificeret i kravspecifikationen. Derudover beskrivelsen af softwarekomponenterne og deres grænseflader beskrevet i systemarkitektur-dokumentet.
- At fastlægge systemets softwareklasser og deres indbyrdes interaktioner.
- At beskrive de enkelte klassers vigtigste metoder.

## 1.0.2 Reference dokumentation

- Kravspecifikation for projektet.
- Systemarkitektur-dokument.

# Kontrolinterface 2

---

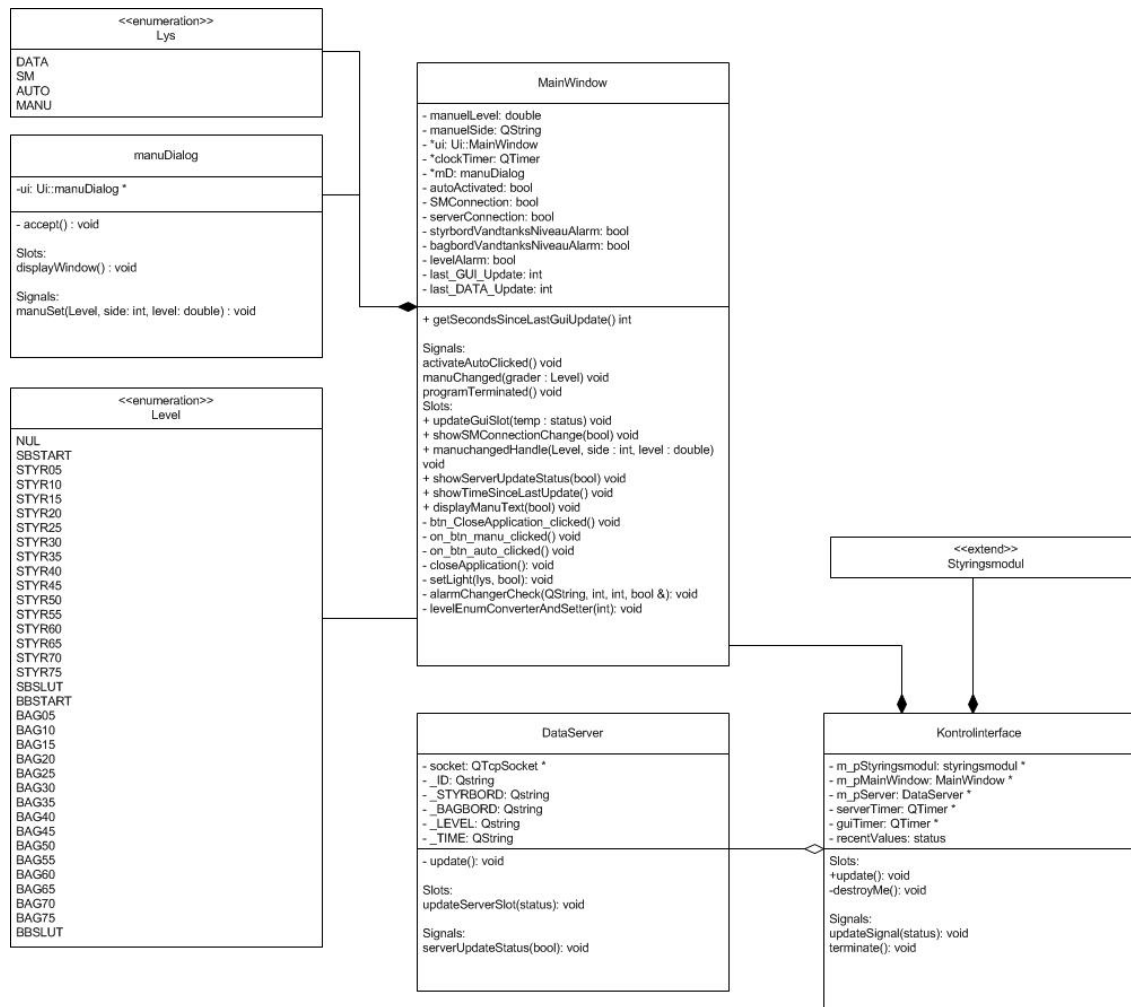
Nedenfor følger design af software til Kontrolinterfacet. Dette er lavet på baggrund af kravspecifikation og systemarkitektur.

## 2.0.3 Modulets ansvar

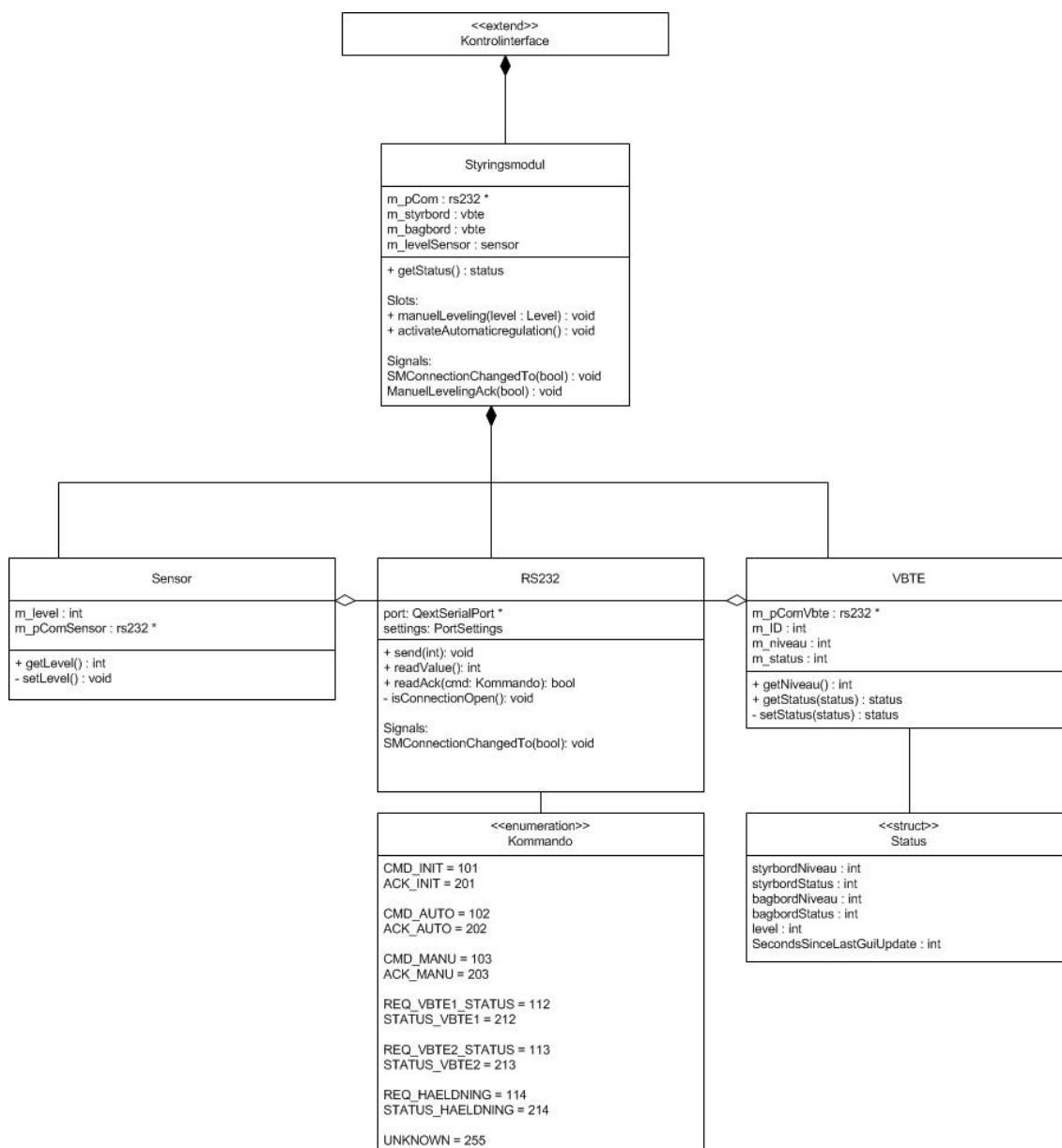
Kontrolinterfacet er brugerens primære kontaktflade til systemet. Programmet indeholder en brugergrænseflade der opfylder kravene i Kravspecifikationen. Her kan der også ses en prototype på den grafiske brugergrænseflade. Kontrolinterfacet står for at modtage inputs fra brugeren. Disse inputs sendes som kommandoer til Styringsmodulet. Det er også herfra at Kontrolinterfacet modtager de værdier, som sidenhen vises på den grafiske brugergrænseflade. Kontrolinterfacet står også for kommunikationen til den eksterne database. Her sendes en række parametre om skibet og dets status.

## 2.0.4 Klassediagram

Nedenfor ses klassediagrammet for Kontrolinterfacet. Bemærk at klassediagrammet er delt op i to. Skæringsstedet er mellem Kontrolinterface-klassen og Styringsmodul-klassen og er markeret med «extend».



**Figur 2.1.** På figuren ses klassediagrammet for KI - Kontrolinterface-delen



**Figur 2.2.** På figuren ses klassesdiagrammet for KI - Styringsmodul-delen

## 2.1 Metode- og klassebeskrivelser

### 2.1.1 MainWindow

#### Ansvar

Denne klasse indeholder de funktioner der er skrevet til Qt-formen MainWindow.ui hvori selve det grafiske er opbygget. Klassen indeholder de funktioner der anvendes i forbindelse med den grafiske brugergrænseflade. Det være sig når der kommer et input, eller der skal opdateres nogle værdier på skærmen.

**Funktionsbeskrivelser**

---

```
int getSecondsSinceLastGuiUpdate();
```

---

Beskrivelse: En simpelt get-funktionen der returnerer værdien af klasseattributten *last\_GUI\_Update*.

Parametre: Ingen

Returværdi: `int` secondsSinceLastGuiUpdate

---

```
void updateGuiSlot(status temp);
```

---

Beskrivelse: Bliver kaldt når GUI'en skal opdateres. Den modtager parameterent *temp* som er en struct af typen status. Ud fra denne struct hives de værdier ud, som skal vises på GUI'en. Værdierne vises ved de set-funktioner der er tilknyttet de anvendte widgets (og dermed en del af Qt-frameworket.) Når værdierne er opdateret vises det som en aktivitet i aktivitetsloggen

Parametre: `status` temp

Returværdi: Ingen

---

```
void showSMConnectionChange();
```

---

Beskrivelse: Kaldes hvis SM-forbindelsen til styringsmodulet ændres fra forbundet til mistet forbindelse eller omvendt. Det udløser en aktivitet i aktivitetsloggen. Derudover skiftes lyset på gui'en. Parameteren state er den status som forbindelsen har ændret sig til.

Parametre: `bool` state

Returværdi: Ingen

---

```
void manuChangedHandle(Level samlet, int side, double level);
```

---

Beskrivelse: Kaldes når brugeren har ændret i indstillingen til den manuelle hældning. Som parametre modtages hvilken side man ønsker at skibet skal hælde til (`int` side), hvor meget det skal hælde (`double` level) samt de to informationer samlet i en enum, Level samlet. Funktionen emitter signalet "manuChanged(Level temp). Det sætter også klasseattributterne manuelSide, manuelLevel samt autoActivated til deres rette værdier. Til sidst kaldes funktionen displayManuText(true)

Parametre: `Level` samlet

`int` side

`double` level

Returværdi: Ingen



---

```
void showServerUpdateStatus(bool state);
```

---

Beskrivelse: Kaldes hver gang signalet `DataSet::serverUpdateStatus()` udsendes. Funktionen undersøger om forbindelsen har ændret sig ved at sammenligne med attributen `serverConnection`. Hvis forbindelsen har ændret sig udsendes dette som en aktivitet. Lyset ændres også således at det passer ved hjælp af `setLight(DATA, serverConnection)`. Hvis vi modtager "true" vil `last_DATA_Update` opdateres til således til den nuværende værdi af sekunder siden epoch.

Parametre: Ingen

Returværdi: Ingen

---

```
void showTimeSinceLastUpdate();
```

---

Beskrivelse: Kaldes hvert sekund. Funktionen opdaterer antallet af sekunder siden sidste overførelse af data til serveren eller til SM. Når tiden er længere end tiden mellem hver opdatering vil dette tal skifte til rødt.

Parametre: Ingen

Returværdi: Ingen

---

```
void displayManuText(bool show);
```

---

Beskrivelse: Viser eller skjuler teksten med indstillingen af manuel hældning afhængig af parameteret `show`.

Parametre: Ingen

Returværdi: Ingen

---

```
void activateAutoClicked();
```

---

Beskrivelse: udsendes når der er blevet trykket på knappen *activateAutoClicked*

Parametre: Ingen

Returværdi: Ingen

---

```
void activateAutoClicked();
```

---

Beskrivelse: udsendes når der er blevet trykket på knappen *activateAutoClicked*

Parametre: Ingen

Returværdi: Ingen

---

```
void manuChanged(Level grader);
```

---

Beskrivelse: Udsendes når der er blevet ændret en manuel indstilling.

Parametre: `Level` grader

Returværdi: Ingen

---

```
void programTerminated();
```

---

Beskrivelse: Udsendes når programmet er blevet lukket ned.

Parametre: Ingen

Returværdi: Ingen

```
void btn_CloseApplication_clicked();
```

---

Beskrivelse: Kaldes når luk-knappen på GUI'en er blevet trykket. Udsender signalet `programTerminated()` hvis brugeren bekræfter valget

Parametre: Ingen

Returværdi: Ingen

```
void on_btn_manu_clicked();
```

---

Beskrivelse: Kaldes når der bliver trykket på knappen for manuel hældning. Viser dialogen "manuDialog".

Parametre: Ingen

Returværdi: Ingen

```
void on_btn_auto_clicked();
```

---

Beskrivelse: Kaldes når der trykkes på *Automatisk Hældnings-knappen*. Aktiverer automatisk styring og deaktiverer den manuelle.

Parametre: Ingen

Returværdi: Ingen

```
void setLight(lys id, bool state);
```

---

Beskrivelse: Sætter lyset i forhold til parametrene. `lys` er en enum der bestemmer hvilket element lyset skal ændres for. `state` er om lyset skal være tændt eller ej

Parametre: `lys id`  
`bool state`

Returværdi: Ingen

```
void alarmChangerCheck(QString sentence, int critical_point, int value, bool &earlier_state);
```

---

Beskrivelse: Tester om alarmerne har ændret sig. Sentence er starten af den sætning der skrives i aktivitetsloggen. `critical_point` er det kritiske punkt for det emne der arbejdes på. Value er den værdi den har. `Earlier_state` er hvilket stadie alarmen var i tidligere.

Parametre: `QString sentence`  
`int critical_point`  
`int value`  
`bool &earlier_state`

Returværdi: Ingen

```
void levelEnumConverterAndSetter(int level);
```

---

Beskrivelse: Konverterer en integer baseret på Level-enumeratoren om til en side og en vinkel.

Parametre: `int level`

Returværdi: Ingen

### 2.1.2 Kontrolinterface

#### Ansvar

Dette er hovedklassen hvori selve programmet lever. Oprettelsen af et objekt af denne klasse er derfor også det eneste der sker i main.

#### Funktionsbeskrivelser

---

```
void update();
```

---

Beskrivelse: Får en status-struct fra SM-klassen udfyldt med de nuværende værdier for systemet. Tilføjer antal sekunder siden sidste gui-update ved hjælp af `getSecondsSinceLastGuiUpdate`. Denne struct udsendes med signalet `updateSignal(status recentValues)`

Parametre: Ingen

Returværdi: Ingen

---

```
void updateSignal(status recentValues);
```

---

Beskrivelse: Signalet der sendes når gui og server skal updateres. Indeholder en struct med alle relevante værdier.

Parametre: `status recentValues`

Returværdi: Ingen

---

```
void terminate();
```

---

Beskrivelse: Udsendes når brugeren ønsker at terminere programmet.

Parametre: Ingen

Returværdi: Ingen

---

```
void destroyMe();
```

---

Beskrivelse: Muliggør nedlæggelse af klassen med et funktionskald.

Parametre: Ingen

Returværdi: Ingen

### 2.1.3 DataServer

#### Ansvar

Klassen står for al kommunikation med serveren via en TCP-forbindelse. Forbindelse oprettes og nedlægges hver gang der tages kontakt. DataServer-objektet opdaterer serveren når den får ordre om det fra Kontrolinterface-klassen.

**Funktionsbeskrivelser**

```
void onDelete();
```

---

Beskrivelse: Kaldes i destruktoren. Sender en besked til serveren om at programmet termineres .

Parametre: Ingen

Returværdi: Ingen

```
void updateServerSlot(status recentValues);
```

---

Beskrivelse: Kaldes når signalet updateSignal(status recentValues) udsendes. Funktionen sørger så for at der via TCP-forbindelsen bliver udsendt de relevante værdier til databasen.

Parametre: `status recentValues`

Returværdi: Ingen

```
void serverUpdateStatus(bool status);
```

---

Beskrivelse: Udsendes når databasen er blevet opdateret. Parameteren "status"indikerer hvorvidt overførelsen var succesfuld eller ej. Der ventes ikke noget svar fra databasen.

Parametre: `bool status`

Returværdi: Ingen

**2.1.4 manuDialog****Ansvar**

manuDialog-klassen står for håndtering af det vindue der åbnes ved tryk på *Manuel Hældningsregulering-knappen*.

**Funktionsbeskrivelser**

```
void manuSet(Level degrees, int side, double level);
```

---

Beskrivelse: Udsendes i funktionen accept(). Der medsendes de data som brugeren har valgt på dialogen.

Parametre: `Level degrees`

`int side`

`double level`

Returværdi: Ingen

---

```
void accept();
```

---

Beskrivelse: Kaldes når der trykkes på "OK-knappen på dialogen. Det indtastede omdannes til en værdi i forhold til enumeratoren "Level". Dialogen lukkes og signalet manuSet(...) udsendes

Parametre: `Level` degrees  
`int` side  
`double` level

Returværdi: Ingen

### 2.1.5 Styringsmodul

#### Ansvar

Klassen har samme rolle som det fysiske styringsmodul har i systemet. Det giver kontrolinterfacet adgang til sensor-værdier og vandstandsniveauer igennem sine underklasser, VBTE og Sensor.

#### Funktionsbeskrivelser

---

```
status getStatus();
```

---

Beskrivelse: Indhenter værdierne for systemet fra VBTE'er og Hældningssensor. Disse værdier sættes ind i structen temp som så returneres.

Parametre: Ingen

Returværdi: `status` recentValues

---

```
void manuelLeveling(Level level);
```

---

Beskrivelse: Sender kommando og vinkel til PSoC over RS232 vha. objektet m\_pCom.

Parametre: `Level` level

Returværdi: Ingen

---

```
void activateAutomaticRegulation();
```

---

Beskrivelse: Sætter hvorvidt automatisk regulering skal være aktiveret eller ej til styringsmodulet.

Parametre: Ingen

Returværdi: Ingen

---

```
void SMConnectionChangedTo(bool status);
```

---

Beskrivelse: Udsendes hver gang der har været en overførelse. Status indikerer hvorvidt overførelsen var succesfuld eller ej.

Parametre: `bool` status

Returværdi: Ingen

```
void ManuelLevelingAck(bool status);
```

---

Beskrivelse: Udsendes når der er blevet sendt kommandoen CMD\_MANU til styringsmodul. Bool status indikerer hvorvidt overførelsen var succesfuld ej.

Parametre: bool status

Returværdi: Ingen

### 2.1.6 VBTE

#### Ansvar

Håndterer værdierne for hver sin vandballasttankenhed. Kommunikerer til det fysiske styringsmodul ved hjælp af RS232-klassen.

#### Funktionsbeskrivelser

```
int getNiveau();
```

---

Beskrivelse: Kalder setNiveau og returnerer værdien af niveau

Parametre: Ingen

Returværdi: int niveau()

```
status getStatus(status temp);
```

---

Beskrivelse: Skaffer status vha. rs232-objektet og sætter niveau og status i structen temp som herefter returneres.

Parametre: status temp()

Returværdi: status temp()

### 2.1.7 Sensor

#### Ansvar

Håndterer værdierne for hældningssensoreren. Kommunikerer til det fysiske styringsmodul ved hjælp af RS232-klassen.

#### Funktionsbeskrivelser

```
int getLevel();
```

---

Beskrivelse: Skaffer værdien af hældningen på skibet vha. rs232-objektet og returnerer det.

Parametre: Ingen

Returværdi: int level

### 2.1.8 RS232

#### Ansvar

Håndterer kommunikationen til det fysiske styringsmodul ved protokollen der ses i enumeratoren "Kommando".

**Funktionsbeskrivelser**

---

```
void send(int cmd);
```

---

Beskrivelse: Sendefunktion. Sender den medsendte integer.

Parametre: `int` cmd

Returværdi: Ingen

---

```
int readValue();
```

---

Beskrivelse: Modtagerfunktion. Returnerer den læste værdi.

Parametre: Ingen

Returværdi: `int` receivedValue

---

```
bool readAck(Kommando cmd);
```

---

Beskrivelse: Kalder readValue() og sammenholder dennes returværdi med den værdi der er medsendt som parameter (cmd). Returnerer hvor vidt de to var identiske.

Parametre: `Kommando` cmd

Returværdi: `bool` status

---

```
void SMConnectionChangedTo(bool status);
```

---

Beskrivelse: Når der har været en overførelse udsendes dette signal. Status indikerer hvorvidt overførelsen var succesfuld eller ej.

Parametre: `bool` status

Returværdi: Ingen

---

```
void SMConnectionChangedTo(bool status);
```

---

Beskrivelse: Når der har været en overførelse udsendes dette signal. Status indikerer hvorvidt overførelsen var succesfuld eller ej.

Parametre: `bool` status

Returværdi: Ingen

---

```
bool isConnectedOpen();
```

---

Beskrivelse: Tester om der er forbindelse til PSoC

Parametre: Ingen

Returværdi: `bool` connectionState

# Databasen 3

---

Nedenfor følger design af software til databasen og dens interface. Dette er lavet på baggrund af kravspecifikation og systemarkitektur.

## 3.0.9 Modultes Ansvar

Databasen er her hvor havne terminalens personale kan aflæse data fra skibet. Disse data er sendt fra KI. Programmerne indeholder brugergrænseflader der opfylder kravene, beskrevet i kravspecifikationen. Her kan der også ses en prototyper på brugergrænsefladen. Database modulet har 3 dele. Severen, Web-siden og en mySQL database.

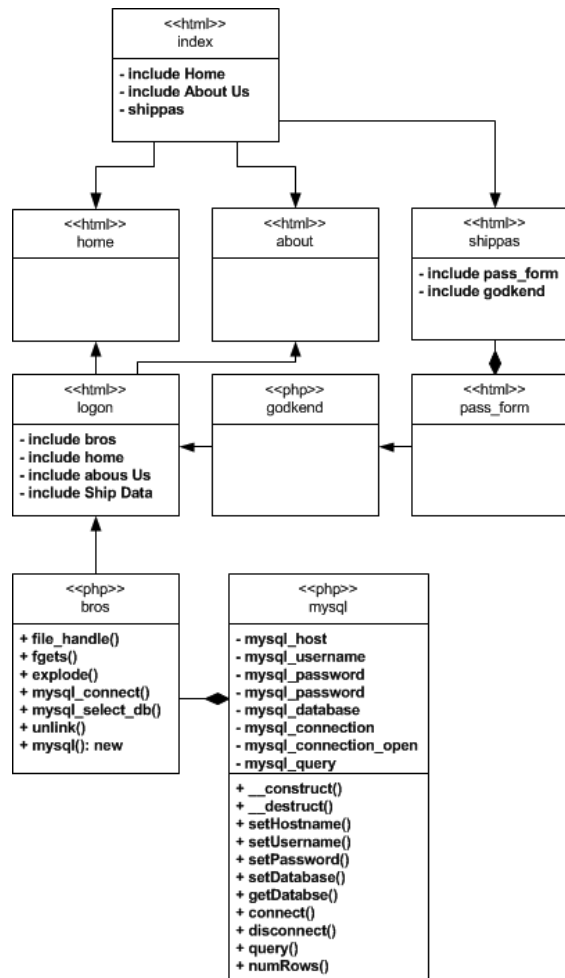
**Severen** står for kommunikationen imellem KI og Databasen. Severen modtager data fra KI og lager disse i en tekst fil

**Web-siden** giver brugeren mulighed for at se info om BROS samt at logge sig ind i BROS database hvorfra at data om skibe der er tilsluttet systemet kan aflæses. Web-sidens 3 vigtigste funktioner er at gemme ny data til mySQL databasen, slette den tekst fil som severen lavede og vise data for brugeren. Til at håndtere web-siden benyttes en apache server. **mySQL databasen** er en database som er installeret på computeren. Alle data som er sendt fra KI er lageret i mySQL databasen.

## 3.0.10 Klassediagrammer

Nedenfor ses klassediagrammerne for databasen. Bemærk database modulet er lavet som en server del og en web del

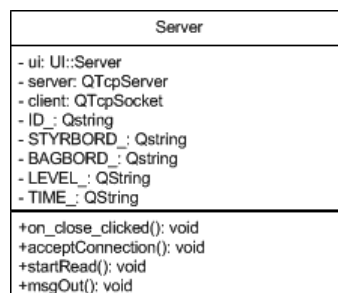




**Figur 3.1.** Klassedigram for databasens web-side.

1

tilføj dato()



**Figur 3.2.** Klassedigram for databasens server

<sup>1</sup>FiXme Note: check med kim, beskriv «html» og «php»

### 3.0.11 Globale variabler

### 3.0.12 Funktionsbeskrivelser

#### Server

Denne header har til ansvar at starte GUI og håndtere TCP forbindelse

---

`Void on_close_clicked( void )`

Beskrivelse: Ved klik forespørges brugeren om denne ønsker at lukke severen

Parametre:

Returværdi:

---

`Void acceptConnection( void )`

Beskrivelse: Står for at acceptere forbindelse fra KI og connecte.

Parametre: QTcpServer server

QTcpSocket\* client

Returværdi:

---

`Void startRead( void )`

Beskrivelse: Læser data fra socket. Data til fil og GUI

Parametre: QTcpSocket\* client

Returværdi:

#### Wep-side

Web-siden står for at fremvise skibs data grafisk for terminal personalet. Desuden står den for at lagre data og loade data fra mySQL databasen.

#### bro

---

`file_handle( )`

Beskrivelse: Læse hvilken adresse databasen ligger på

Parametre:

Returværdi:

---

`fgets ( )`

Beskrivelse: Læse hvilken adresse databasen ligger på

Parametre:

Returværdi:

---

`explode( )`

Beskrivelse: Læse hvilken adresse databasen ligger på

Parametre:

Returværdi:

`mysql_connect ( )`

---

Beskrivelse: Læse hvilken adresse databasen ligger på  
Parametre:  
Returværdi:

`mysql_select_db ( )`

---

Beskrivelse: Læse hvilken adresse databasen ligger på  
Parametre:  
Returværdi:

`unlink ( )`

---

Beskrivelse: Læse hvilken adresse databasen ligger på  
Parametre:  
Returværdi:

`new mysql ( )`

---

Beskrivelse: Læse hvilken adresse databasen ligger på  
Parametre:  
Returværdi:

## **mysql**

`setHostName ( )`

---

Beskrivelse: Læse hvilken adresse databasen ligger på  
Parametre:  
Returværdi:

`setUserName ( )`

---

Beskrivelse: Skriver brugernavn til databasen  
Parametre:  
Returværdi:

`setPassword ( )`

---

Beskrivelse: Skriver password til databasen  
Parametre:  
Returværdi:

`setDatabase ( )`

---

Beskrivelse: Fortæller mySQL hvilken database der skal benyttes  
Parametre:  
Returværdi:

---

```
getDatabase( )
```

---

Beskrivelse: Tager fat i databasen

Parametre:

Returværdi:

---

```
connect( )
```

---

Beskrivelse: Står for at samle localhost, username, password, database og connecte til databasen

Parametre:

Returværdi:

---

```
disconnect( )
```

---

Beskrivelse: Lukker database forbindelsen

Parametre:

Returværdi:

---

```
query( )
```

---

Beskrivelse: Opretter database kø og skriver data til skærm

Parametre:

Returværdi:

---

```
numRows( )
```

---

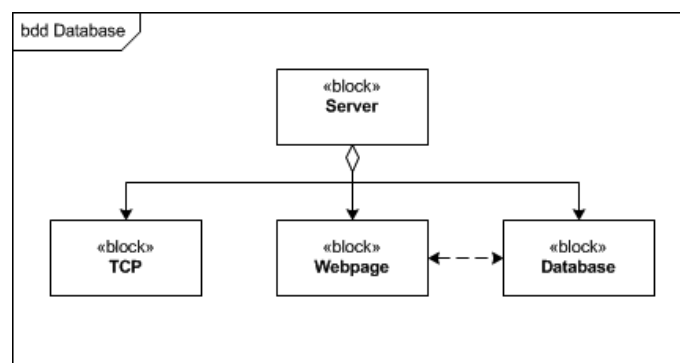
Beskrivelse: Checker hvor mange rækker der findes i databasen bruges desuden til at udskrive om databasen er tom.

Parametre:

Returværdi:

## 3.1 Design

### 3.1.1 Server

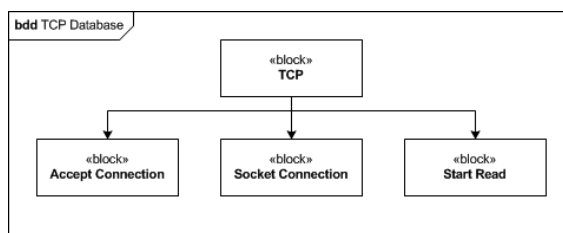


*Figur 3.3.* BDD server

Databasen er en server som har de 3 underblokke TCP, Database og Web-page som illustreret på figur 3.3

TCP er en dataforbindelse (Transmission Control Protocol). Denne protokol er benyttes til at sende data fra KI til serveren. Serveren vil modtage data og lagere dette i en midlertidig backup fil. TCP-forbindelsen er kodet i C++. For TCP-forbindelsen benyttes TCP - protocollen som tilbyder sikker data overførelse fra BROS. Databasen er en mySQL database som frit kan downloades og installeres på en Linux, Windows eller Mac. Man skal installere en server del og en client del. Server delen er den del der gør det muligt at håndtere og lagre data. Denne ligger under client delen og er nødvendig for at client kan fungere. mySQL client gør det muligt at en bruger kan tilgå og læse fra databasen uden at gøre ændringer i denne. Dette benyttes i web interfacet. mySQL kan tilgås direkte fra terminalen og giver mulighed for forskellige opsætninger af databaser og tabeller samt forskellige bruger rettigheder. **muligheder med mySQL** Web-pagen er udviklet i php som giver gode muligheder for kommunikation til og fra mySQL databasen. Web-pagen er implementeret ved hjælp af en apache server som er en web server. Web-pagen har en general information omkring BROS og et login til at tilgå databasen. Ved at anvende et web-interface gives der mulighed for at data kan aflæses fra andre pladser end fra den direkte server. Ved at kende ip eller host navn er det muligt at tilgå web-siden. Den web baserede database loader mySQL databasen og fremviser denne grafisk for brugeren.

### 3.1.2 TCP server



*Figur 3.4.* BDD TCP server(ikke færdig)

TCP protocollen er en af kerneprotokollerne på nutidens internet. Gennem TCP kan forskellige værtsmaskine igennem f.eks. internet ethernet og trådet forbindelse oprette forbindelse til hinanden og udveksle datapakker. Protokollen giver programmelt på værtsmaksine nogle vitale garantier for at disse datapakker afsendes og modtages ved:

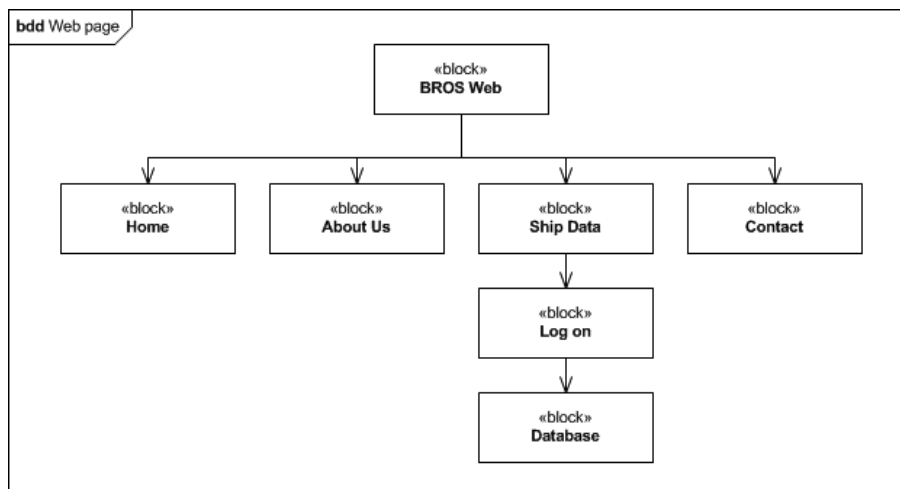
- Stabilitet: En pakke der går tabt bliver forsøgt afsendt igen
- Ordnet levering: En pakke kommer frem til modtageren i samme rækkefølge som de blev afsendt

Der ud over benytter TCP sig af forskellige port numre. Forskellige portnumre gør det muligt etablere flere forskellige datstrømme til og fra samme værtsmaskine.

Selve programmet er kodet op over socket programmering. Under opstart initialiseres socket, ip og porte. Når disse er succesfuldt initialiseret afventer TCP serveren at KI connecter. Efter connection modtager TCP serveren datapakken fra KI og gemmer denne i en textfil kaldt ship.txt. Denne fil bruges som en midlertidig sikkerhed for at data fra KI sikkert bliver inført i mySQL databasen.

sequuens diagram

### 3.1.3 Web-side



*Figur 3.5.* BDD Web-page BROS

For at web siden kan køre kræves der at der på serveren er installeret en web - server. Der er på denne server installeret apache som web - server.

Ved opstart af serveren bliver denne automatisk startet og start siden er **BROS**. Web siden fungere som bruger interfacet for havne terminalen. Web siden er opbygget som et dynamisk web page og kodet i php. Dette sikre minimal loading time ved hjælp af ajax. Alle styles på siden er styret af css. Siden har 4 forskellige under sider Home, About Us, Ship Data og Contact. Ved klik på Ship Data vil man blive bedt om at taste sit password som sikre at kun autoriserede personer får adgang til systemet.

Siden der håndtere skibs data starter med at connecte til mySQL databasen om ikke der kan connectes til databasen vil der blive udskrevet en error og siden vil igen forsøge at conencte til databasen. Efter connection til databasen vil den gemte fil fra TCP-serveren blive loadet ind i mySQL databasen og filen vil blive slettet. efter loading vil siden loade alle data i mySQL databasen og vise denne for brugeren. Data der kan vises for brugeren er:

- Skibs ID
- Højre tanks vandniveau
- Venstre tanks vandniveau
- Hældningsniveau
- Forbindelse til KI

Siden checker hvert 5 sekund om der er nu data. Hved ny data vil denne blvie placeret øverst på siden. Når brugeren er færdig med at benytte BROS databasen kan brugeren trykke log of i øverste højre hjørne.

## 3.2 Metodebeskrivelse

### 3.2.1 TCP KI

**Void msg(string, string, string, string, string);**

Håndtere data der skal sendes via tcp.

**Void socket();**

Opretter socket forbindelse for ctp client.

**Void connection();**

Kalder ip adresse og portnummer for TCP server. Overføre data.

### 3.2.2 TCP database

**Void socketConnect();**

**Void msgServer();**

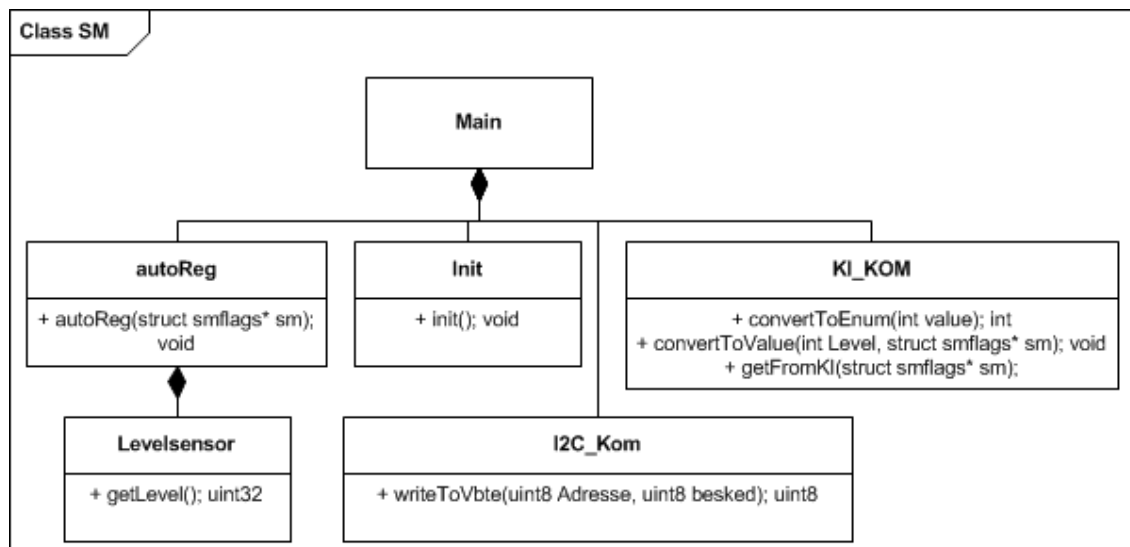
Dette afsnit beskriver designet af styringsmodulet, SM.

## 4.1 Klassens ansvar

Styringsmodulet har til ansvar at holde styr på levelsensoren og værdierne fra VBTE. Den kommunikerer med KI og VBTE med indbyggede API'er fra Cypress PSoC 5 biblioteker.

## 4.2 Klassediagram

Nedenfor ses klassediagrammet for SM. Bemærk at koden dog er i C men for overblikket er der lavet klassediagram.



*Figur 4.1.* På figuren ses klassediagrammet for SM

## 4.3 Funktioner

bla bla



## 4.4 Variabler

Variabel	Beskrivelse
autoflag	Denne variable er et flag der holder styr på automatisk regulering .
manuflag	Et flag til at holde styr på manuel regulering.
levelVal	En variable med vores level værdi.
VBTE1Niveau og VBTE2Niveau	Holder styr på vandniveauet i ballasttanke i %.
VBTE1Status og VBTE2Status	Holder styr op tilgængelighed for VBTE1 og 2.
vinkelVal	Indeholder værdien for manuel regulering.

Alle variabler er indkapslet i en struct navngivet "smflags".

## 4.5 Funktionsbeskrivelser

### 4.5.1 Init

#### Ansvar

Denne header har til ansvar at sørge for alle komponenter opretter og initieret. `void init( void );`

---

Beskrivelse: Funktionen anvender API'et fra Cypress componenter og står for at initiere og starte vores PSoC hardware. Den sætter også et register tilhørende vores Accelerometer.

Parametre: ingen

Returværdi: ingen

### 4.5.2 Levelsensor

#### Ansvar

Denne header har til ansvar at hente levelværdien ind fra vores accelerometer. `uint32 getLevel( void );`

---

Beskrivelse: Funktionen anvender API'et fra Cypress componenter og venter på at vores ADC henter convertere det analoge signal. Funktionskald for ADC ses i PSoC databladet.

Parametre: ingen

Returværdi: `uint32 levelVal`

### 4.5.3 autoReg

#### Ansvar

Denne header har til ansvar at styre automatisk regulering. `void autoReg( struct smflags* sm );`

---

Beskrivelse:	autoReg anvender værdier fra VBTE moduler samt KI til at holde systemet i et bestemt level. Funktionen starter med at checke på automatisk og manuel styrings flagene. Derefter kalder den getLevel agere ud fra niveauet. Funktionen vil altid tømme fra en tank før den begynder at fylde en anden.
Parametre:	<code>struct smflags* sm</code>
Returværdi:	ingen

#### 4.5.4 I2C\_Kom

##### Ansvar

Denne header har til ansvar at kommunikere med VBTE modulerne. `uint8 writeToVbte(uint8 Adresse, uint8 besked);`

---

Beskrivelse:	writeToVbte anvender I2C fra Cypress PSoC 5 API til at skrive til VBTE modulerne. Den tager adressen og beskeden man skal sende og sender til pågældende enhed. Derefter venter den på svar som den så returnere.
Parametre:	<code>uint8 Adresse</code> <code>uint8 besked</code>
Returværdi:	<code>uint8 VbteNiveau</code>

#### 4.5.5 KI\_KOM

##### Ansvar

Denne header har til ansvar at kommunikere med KI enheden.

`int convertToEnum(int value);`

---

Beskrivelse:	funktionen tager en level værdi ind for så at konvertere den til en Enum(integer) som den returnere.
Parametre:	<code>int value</code>
Returværdi:	<code>int Enum</code>

`void convertToValue(int Level, struct smflags* sm);`

---

Beskrivelse:	Funktionen tager en enum og en pointer som den så konvertere til en level værdi og sætter i sm structen.
Parametre:	<code>int Level,</code> <code>struct smflags* sm</code>
Returværdi:	ingen

`void getFromKI(struct smflags* sm);`

---

Beskrivelse: Funktionen anvender UART fra Cypress PSoC 5 API'en til at modtage en besked fra KI modulet som den så vurderer og agere på. Når den har modtaget noget sender den en ack tilbage til KI modulet. Derefter handler den og hvis det er nødvendigt sender data til KI.

Parametre: `struct smflags* sm`

Returværdi: ingen

## 4.6 Eventuelle Sekvensdiagrammer og state machines

Måske kommer de senere?

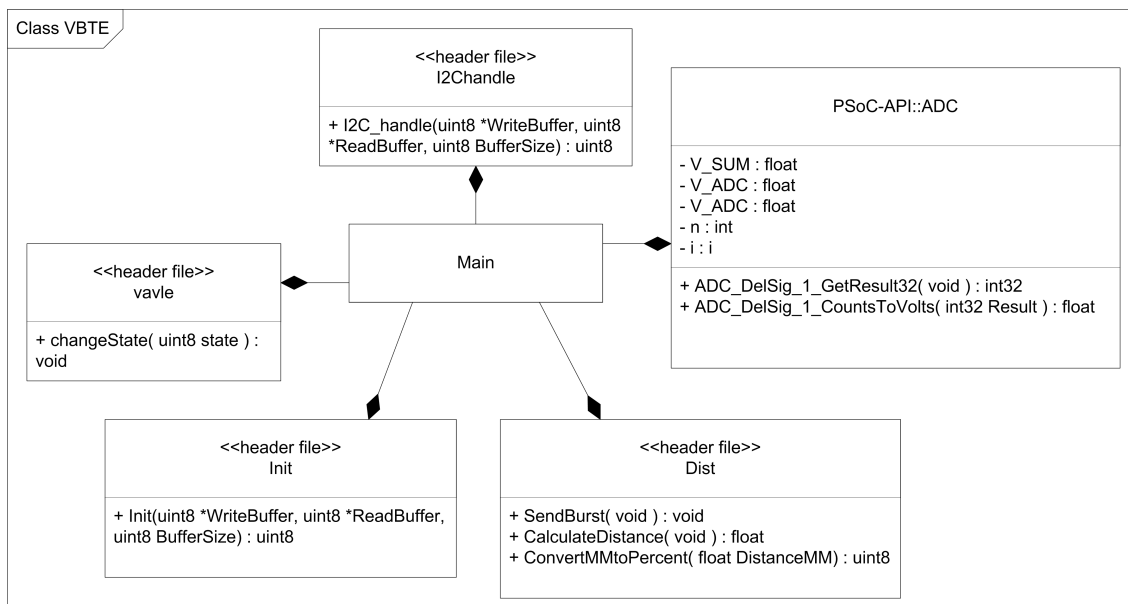
Nedenfor følger design af software til VBTE. Dette er lavet på baggrund af kravspecifikation og systemarkitektur. Bemærk der i dette design dokument blandt andet ikke er beskrevet mixer, pga osv. da deres eneste funktion er "Start()". Derudover er der en betydelig hardware del knyttet til dette modul, der refereres derfor til detaljeret hardware design for yderligere detaljer om VBTE modulet.

## 5.1 Modulets ansvar

Som beskrevet i systemarkitektur står VBTE'en for at måle vandniveauet i ballasttankene samt at lukke vand ind eller ud af ballasttankene. Hertil er der også en kommunikation med SM modulet indeholende instruktioner.

## 5.2 Klassediagram

Nedenfor ses klassediagrammet for VBTE. Bemærk at koden dog er i C men for overblikket er der lavet klassediagram.



**Figur 5.1.** På figuren ses klassediagrammet for VBTE **Billedet skal opdateres**

## 5.3 Globale variabler

Variabel	Beskrivelse
BurstLengthVal	Denne variabel er anvendt til at håndtere antallet af perioder burstet bliver sendt med.
WaitBurstVar	Bliver brugt til nonblocking delay til SendBurst funktionen.
BurstTimerVal	Holder på Timerens værdi når et burst er sendt.
DistanceTimerVal	Holder værdien på timeren når et burst er modtaget.
CalcDistFlag	Bliver sat når et burst er modtaget så en afstand kan blive beregnet.
BurstFlag	Bliver sat når et burst bliver sendt og hevet ned når et burst er modtaget. Dette sker for ikke at få flere detektioner på samme signal.

## 5.4 Metode- og klassebeskrivelser

### 5.4.1 Init

#### Ansvar

Denne header har til ansvar at initiere alle moduler og blokke på PSoC'en. Disse funktioner hentes fra PSoC'ens API.

#### Funktionsbeskrivelser

```
void Init( uint8* WriteBuffer, uint8* ReadBuffer, uint8 BufferSize )
```

Beskrivelse: Funktionen anvender API'et fra alle PSoC blokke anvendt i designet og kalder deres start funktion. Derudover initierer den også read- og writebuffer til I2C modulet.

Parametre: `uint8* WriteBuffer`  
`uint8* ReadBuffer`  
`uint8 BufferSize`

Returværdi: Ingen

### 5.4.2 Valve

#### Ansvar

Denne header har til ansvar at styre ventilerne ud fra "state-variablen modtaget fra I2C\_handle. Headeren benytter PSoC-API'et til kontrol registre..

## Funktionsbeskrivelser

---

```
void ChangeState( uint8 state )
```

---

Beskrivelse: Funktionen modtager state som indeholder indformationer om ventilerne skal være lukkede eller hvilken ventil der skal være åben. Den benytter PSoC5 API'et for kontrol register.

Parametre: `uint8 state`

Returværdi: Ingen

### 5.4.3 Dist

#### Ansvar

Denne header har til ansvar at sende burst, beregne afstanden samt at omregne afstanden til procent.

## Funktionsbeskrivelser

---

```
void SendBurst( void )
```

---

Beskrivelse: Denne metode aktiverer en 40kHz clock og tæller perioderne op til 10, lukker for burstet og ligger timerens værdi ind i den globale variabel BurstTimerVal. Herefter sættes BurstFlag'et.

Parametre: Ingen

Returværdi: Ingen

---

```
float CalculateDistance( void )
```

---

Beskrivelse: Denne metode anvender BurstTimerVal og DistanceTimerVal til at finde ud af hvor mange clocks der er gået fra burstet er blevet sendt til det igen er blevet registreret.

Parametre: Ingen

Returværdi: `float DistanceMM`

---

```
uint8 ConvertMMtoPercent( float )
```

---

Beskrivelse: Metoden modtager afstanden i millimeter og returnerer hvor mange % der er i tanken

Parametre: `float DistanceMM`

Returværdi: `uint8 DistancePercent`

### 5.4.4 I2CHandle

#### Ansvar

Denne header har til ansvar at håndtere I2C. Den kigger om der er kommet noget i writebufferen. Er der kommet noget i bufferen kigger den efter hvilket tilstand det er der skal ændres til og så smider den afstanden i procent i read bufferen.

## Funktionsbeskrivelser

```
uint8 I2C_handle( uint8* WriteBuffer, uint8* ReadBuffer, uint8 BufferSize )
```

Beskrivelse: Funktionen anvender API'et fra I2C blokken i PSoC miljøet. Med disse tjekker den om der er fyldt nyt i bufferen og aflæse dette. Herfer kalder den funktionen I2C\_decode til at afkode beskeden fra SM. Herefter klargøres readbufferen til at sende vandniveau tilbage.

Parametre: `uint8* WriteBuffer`  
`uint8* ReadBuffer`  
`uint8 BufferSize`

Returværdi: `uint8 State`

### 5.4.5 PSoC-API::ADC

#### Ansvar

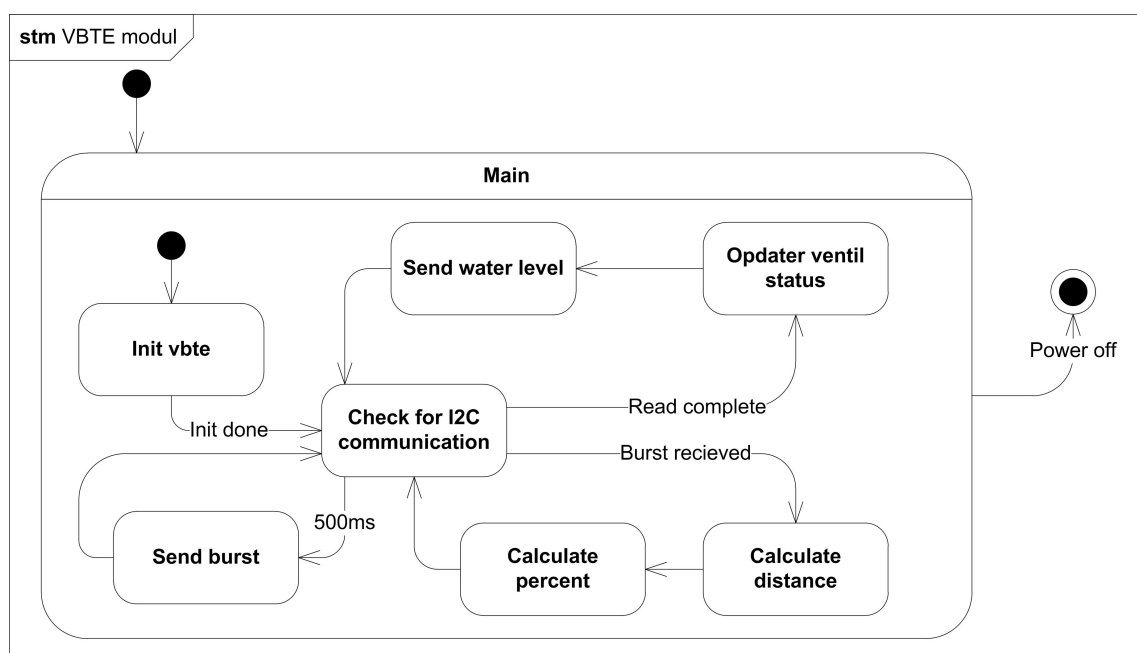
Denne header er kun beskrevet fordi der er implementeret en funktionalitet i dette API. Der er under ADC

#### Beskrivelse

Inde i ADC'ens interrupt header tilføjes funktionalitet så der, hver gang der bliver samplet, bliver valideret på om der er en detektion. Er der en detektion sættes flaget til udregning af afstand samt flaget for burst sættes til 0 igen. For at gøre dette anvendes funktionerne fra API'et for ADC'en.

### 5.4.6 State Machine

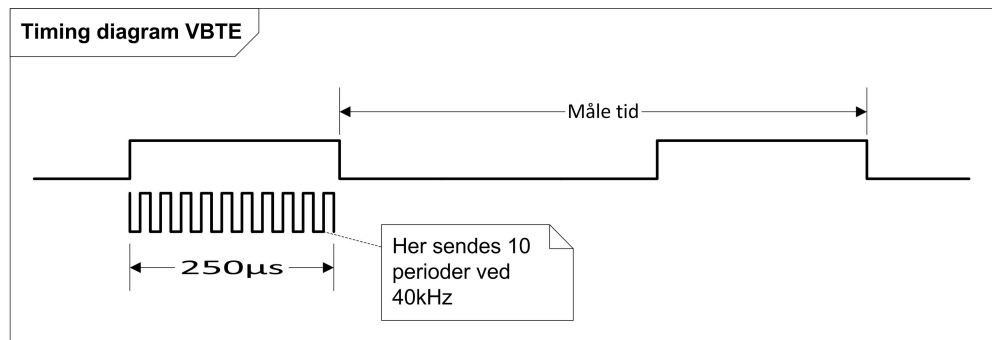
Nedenfor ses statemachine der beskriver det overordnede flow i VBTE programmet.



*Figur 5.2.* Statemachine for VBTE program

### 5.4.7 Timing Diagram

Nedenfor ses timing diagram for en ultralydspuls til afstandsmåling



*Figur 5.3.* Timing diagram for VBTE ultralydspuls

### 5.4.8 Interrupt rutiner

I dette afsnit beskrives interrupt rutinerne i VBTE programmet.

#### **isr\_dist**

Isr\_dist har til ansvar at tælle den globale variabel `WaitBurstVar` op. Den bliver triggeret af en clock på 1kHz. Hver gang variabelen havner over 500 bliver `SendBurst()`; funktionen kaldt og variabelen bliver nulstillet.

#### **isr\_counter**

Isr\_counter tæller variabelen `BurstLengthVal` op. Denne anvendes til at styre at der kun bliver sendt 10 peroder i et burst.



## Rettelser

Note: check med kim, beskriv «html» og «php» . . . . . 17