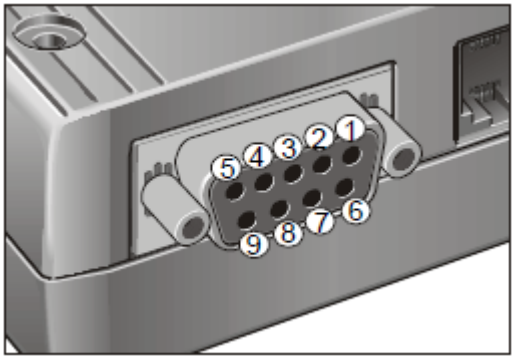
1 Tekniske overvejelser

1.1 RS232-kabel

Forbindelsen til GSM-modemet sker vha. et 9-pins kabel. Det er forbundet til STK-500's Spare port og GSM-modemets RS-232 interface.

Når der skrives til GSM-modemet køres dette igennem UART-driveren, som sender og modtager igennem RS-232 stikket. Interfacet på STK-500 og GSM-modemet er det samme, hvilket resulterer i, at et han-han kabel mellem de to vil få modtagerpindene sat sammen senderpindene sat sammen, hvilket resulterer i ingen kommunikation.

For at løse dette, designede vi et 9-pins kabel, hvor pin 2 og 3 var byttet, således at det passede med senderpindene blev sat til modtagerpindene.



Figur 1: 9-Pin hun stik for STK-500 Spare og GSM-modem

1.2 Drivers

Til projektet er der blevet brugt nogle drivere fra opgaverne fra AMS-forløbet, samt udleveret drivere til UART'en og LED'erne. Herunder er en liste over de drivere, som vi selv har arbejdet på.

1.2.1 LM75

Driveren til temperatursensoren, LM75, er baseret på en opgave fra AMS-forløbet. Den kommunikerer med STK-500 vha. I^2C-bussen og kan i princippet måle fra $-155°C$ til $+155°C$, hvilket vi dog ikke har test eller finder relevant for denne opstilling.

Fra tasken `void LM75SensorTask(void *pvParameters)` kaldes LM75-driverens funktion til, at modtage ny temperatur fra LM75-printet, vist i Listing 1.

Listing 1: LM75's metode til at spørge på ny data

```
int LM75_temperature(unsigned char SensorAddress)
{
  i2c_start();

  unsigned char address = ((0b01001000 | SensorAddress) << 1) | 0↩
     x01;
  i2c_write(address); //Address write

  unsigned char tempMSB = i2c_read(0x00);
  unsigned char tempLSB = i2c_read(0x01);
  i2c_stop();

  return (tempLSB>>7) | (tempMSB<<1);
}
```

Ved at give en adressen på slaven i I^{2}C-bussen som parameter, startes I^{2}C'en og der sendes en forespørgsel på denne slave, hvorefter denne sender sin data retur. Eftersom temperaturen er 9 bit lang, shiftes de to læsninger, således MSB og LSB står korrekt og returneres i en `int`.

1.2.2 LCD162

Driveren til Demo Board, der indeholder et LCD162 display, er baseret på en opgave fra AMS-forløbet. Denne kan udskrive tekst på displayet, hvilket benyttes til at vise den aktuelle temperatur.

Der udskrives hovedsagligt `strings` og `ints` på displayet. På Listing 2 og Listing 3 ses hvordan disse funktioner er skrevet.

Listing 2: LCD162 metode til visning af en string

```
void LCDDispString(char* str)
{
  for(int i = 0 ; i < 32 ; i++)
  {
    if(str[i] == '\0')
      break;
    sendData(str[i]);
  }
}
```

Listing 2 virker således, at den modtager en `char*`, hvor den steppes igennem `char` for `char` og udskrives indtil hele strengen er udskrevet, eller har skrevet flere tegn end der kan være på displayet.

Listing 3: LCD162 metode til visning af en integers

```
1 void LCDDispInteger(int i)
  {
3   char arr[3];
    itoa(i, arr, 10);
5   LCDDispString(arr);
  }
```

Listing 3 virker således, at den tager imod den ønskede `int`, som overføres til et array med plads til 3 `chars`[1], hvorefter `itoa`-funktionen skriver dataen in i arrayet i decimal format. Til sidst udskrives det vha. `LCDDispString`, vist i Listing 2.

1.3 Free RTOS

Vha. Free RTOS, Real Time Operating System, er systemet blevet bygget med flere tasks der kan køre sideløbende[2], sender beskeder til beskedkøer der håndteres vha. mutexes.

Systemet gør brug af to tasks, `LM75SensorTask` og `Lcd162Task`.

1.3.1 Lcd162Task

Denne task bruger en LM75 driver, som er udarbejdet i en lab øvelse. Vi har skrevet LM75_temperature, mens i2c_start, i2c_write og i2c_stop var skrevet på forhånd.

Denne task er tilknyttet en global queue, som vil indeholde de læste værdier fra LM75. Således er det muligt, at videregive data til andre tasks i systemet, samtidigt med at vi sørger for, at have en form for signalering. Den oprettede kø har nemlig indbygget, at receive funktionen kan sætte tasken i dvale, mens den venter på, at en værdi bliver lagt i køen.

1.3.2 Lcd162Task

Denne task henter temperatur værdier fra LM75 via en queue. Hvis denne queue er tom, går tasken i dvale og venter på en værdi skal ankomme. Den hentede værdi udskrives på LCD-displayet via funktionen `LCDDispInteger`, der er vist i Listing 3.

Som funktionen er lige nu, er der en simpel `if`-sætning, som holder øje med, om temperaturen overstiger en bestemt værdi, som sætter buzzeren og GSM-modemet i gang. Efter at have været sat i gang én gang, sættes et flag, således at der buzzes i intervaller af to sekunder ad gangen og GSM modem ikke sender beskeder konstant. Det var dog

[1] Her burde være 6 pladser, så der er plads til -32768 og op til +32767

[2] De kører vha. en scheduler, som afsætter lidt tid til hver og det forekommer derfor sideløbende

ikke den originale tanke, at disse to funktioner skulle kaldes her. Se afsnit 1.4 for bedre beskrivelse.

1.3.3 Buzzer

Buzzeren, der sidder på Demo Board'et, kan afspille toner som advarsel. Tilafspilningen kan der angives en bestemt frekvens, der er bestemt ved følgende formel:

$$frekvens = \frac{3686400Hz}{2 \cdot 32 \text{ prescale} \cdot (1 + OCRn)}$$
$$OCRn = \frac{57600}{frekvens} - 1$$

1.3.4 GSM-modemmet

Via UARTen på STK-500 kommunikerer vi med et GSM modul, MC35i, således vi kan sende beskeder, når der bliver målt en for høj temperaturværdi på LM75. Når vi kommunikerer med modemmet, skriver vi strenge til den via vores UART driver. Modemmet som standard, laver et echo til os, hvilket vil sige, at det vi skriver til den, skriver den tilbage til os, samt et respons. Så når vi sendte den en kommando og ventede på et respons vi kunne arbejde videre på, skulle vi være opmærksomme på, at det vi først får tilbage, er vores egen kommando. Men denne echo kunne konfigureres væk, ved at skrive en simple kommando til den[3].

Når en kommando blev udført korrekt på modemmet, ville der i de fleste tilfælde blive sendt et `OK` tilbage. I koden blev det derfor implementeret således, at hver gang en kommando blev sendt til modemmet, ville programmet vente på dette `OK`, før den næste kommando ville blive sendt. Dette var tanken, men det var ikke det som blev udført i praksis. I den endelig version af vores program, blev denne "wait for OK"erstattet med et delay, da vi aldrig fik `OK` tilbage når modemmet var tilsluttet STK-500.

Hvis vi testede præcis den samme kode, når modemmet var tilsluttet computeren via RS-232, fik vi altid det forventede respons tilbage. Men når vi kørte koden på STK-500, fik vi aldrig `OK` tilbage. Vi fik respons fra den, men det kom an på om `echo` var sluttet til eller fra. Hvis `echo` var sluttet til og vi sendte AT af sted, for at teste om der var forbindelse, ville vi forvente at få `AT\n OK` tilbage. Hvis `echo` var taget fra og vi igen sendte `AT` af sted, ville vi forvente kun at få `OK` tilbage. Men vi fik kun `\r\n` tilbage. Vi

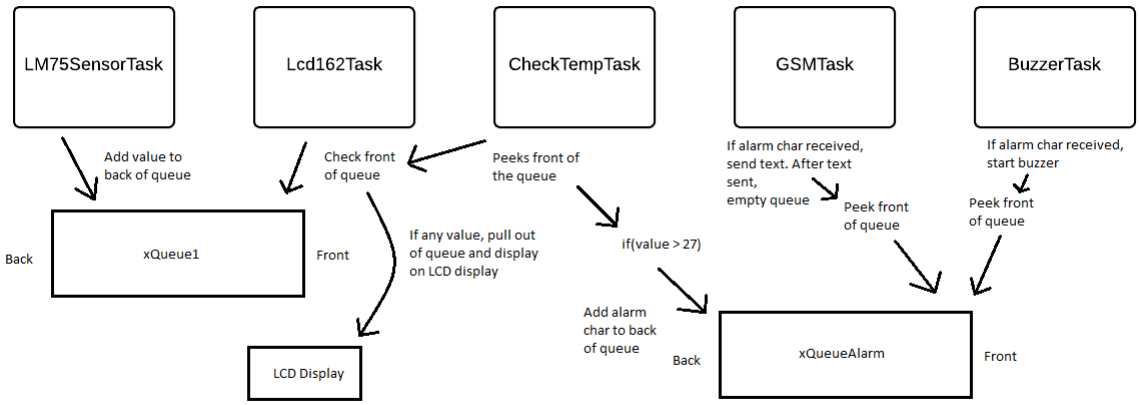
[3] `ATE0\r` - Alle strenge der blev skrevet til modemmet, skulle afsluttes med "`\r`", som er carriage return, for at eksekvere kommandoen.

mistænkte vores hjemmelavet 9-PIN kabel til at være problemet, men vi fik stadig echo tilbage og kunne læse dette.

Grundet manglede tid, blev systemet til sidst implementeret med delays, selvom det ikke er optimalt. Dog viste det sig funktionelt i vores tilfælde. Hvis systemet skulle udvides, var dette punkt et punkt af høj prioritet.

1.4 Ideel opbygning

Systemet var fra starten tænkt til, at køre flere tasks, således vi fik delt ansvaret ud så meget som muligt. Dvs. alle tilsluttede komponenter(LCD162, LM75, buzzer og modemmet) alle ville få deres egne tasks. Idet buzzeren og modemmet sjældent ville blive brugt, var det ideelt at holde dem i dvale, mens en anden task skulle sætte dem i gang, når de skulle bruges. Derfor blev `checkTempTask` et slags mellemled imellem temperatur-delen og alarmerings-delen af systemet. Figur 2 viser og beskriver meget godt det tænkte design af systemet.



Figur 2: Ideel opbygning af systemet

Desværre blev systemet i sidste ende ikke som først planlagt. Vi stødte tit ind i hukommelses problemer, idet systemet havde 5 tasks kørende, samt besked køer. Under `FreeRTOSConfig.h` blev der ændret på `TOTAL_HEAP_SIZE` og `MINIMAL_STACK_SIZE` for at kunne få programmet til at compile, men det ændrede stadig ikke på, at det ikke ville køre når overført til MEGA32.

For at kunne få noget til at virke, blev systemet modificeret, så i stedet for fem tasks, endte det med 2 tasks - nemlig `LM75SensorTask` og `Lcd162Task`.

Kigger man på figur 2 igen kan man se, at det endelige design består af den venstre del af figuren; de 2 tasks, xQueue1 og LCD displayet og `BuzzerTask` og `GSMTask` blev

lavet om til void funktioner.