Projecten voor het werkveld 1

Bart Gysen

Graduaat Netwerkbeheer - Informatica

LID VAN ASSOCIATIE KU LEUVEN

THOMAS MORE

Inhoudsopgave

1 Het project

Voor dit vak moet er een project gemaakt worden waarin alle elementen worden samengevoegd die we het afgelopen jaar hebben geleerd bij Embedded Software & Embedded Devices.

In het project moet er gebruik worden gemaakt van verschillende hardware componenten die met elkaar worden verbonden. Deze componenten moeten met elkaar kunnen communiceren via de peripherals die gezien zijn. De software moet geschreven worden met de technieken die aangeleerd zijn.

1.1 Projectopdracht

De opdracht is het maken van een thermostaat met verwarmingselement, een OLED display die dat communiceert via wifi met een mqtt broker.

De zaken die de thermostaat allemaal moet kunnen:

- Het uitlezen van de temperatuur
- Gewenste temperatuur sturen via mqtt
- Actuele temperatuur naar een mqtt broker sturen
- Het schakelen van een verwarmingselement
- Actuele temperatuur weergeven op het OLED display
- Gewenste temperatuur weergeven op het OLED display
- Actuele temperatuur weergeven op visualisatie software

1.2 MQTT

MQTT (Message Queuing Telemetry Transport) is een lichtgewicht netwerkprotocol dat berichten tussen apparaten transporteert. Het is gebaseerd op het "publish-subscribe" concept. Het protocol werkt meestal via TCP/IP, maar elk netwerkprotocol dat geordende, verliesvrije, bidirectionele verbindingen biedt, kan MQTT ondersteunen. Het is ontworpen voor verbindingen met externe locaties waar er beperkte middelen zijn en de behoefte aan snelheid beperkt is.

Voor MQTT te gebruiken hebben we een broker nodig. De broker is de centrale spil voor alle verzonden berichten. De betrokken apparaten communiceren alleen met de broker en kennen elkaar onderling verder niet.

Data dat we willen sturen naar de broker moeten we publishen in een “topic”. Voor data dat we in de broker willen gaan halen, moeten we op dat topic publishen.

Voor de MQTT broker heb ik gekozen voor **Mosquitto Broker**, deze broker is een ADDON van Home Assistant.

1.3 Home Assistant

Home Assistant ga ik voor 3 zaken gebruiken:

- Als mqtt broker
- Een database waar ik data uit de broker ga halen en die daar in ga plaatsen
- Om data te visualiseren ga ik gebruik maken van Grafana in Home Assistant.

1.3.1 InfluxDB

Dit is een database waar we data kunnen ophalen van tijdreeksgegevens op gebieden zoals operationele monitoring, applicatiestatistieken, Internet of Things-sensorgegevens en realtime analyse.

Hier heb ik 2 databases aangemaakt:

- TEMPSENSOR voor het ophalen van de temperatuur die de thermostaat stuurt.
- SETPOINT om de gewenste temperatuur naar de thermostaat te kunnen sturen.

1.3.2 Grafana

Grafana is een visualisatie software waar er dashboards kunnen aangemaakt worden die data visualiseren in bijvoorbeeld grafieken. Ik ga dit ook enkel maar gebruiken om de actuele temperatuur dat de thermostaat stuurt in een grafiek te visualiseren en om het setpoint af te beelden.

De data dat ik gebruik is gelinkt aan de twee databases van InfluxDB.

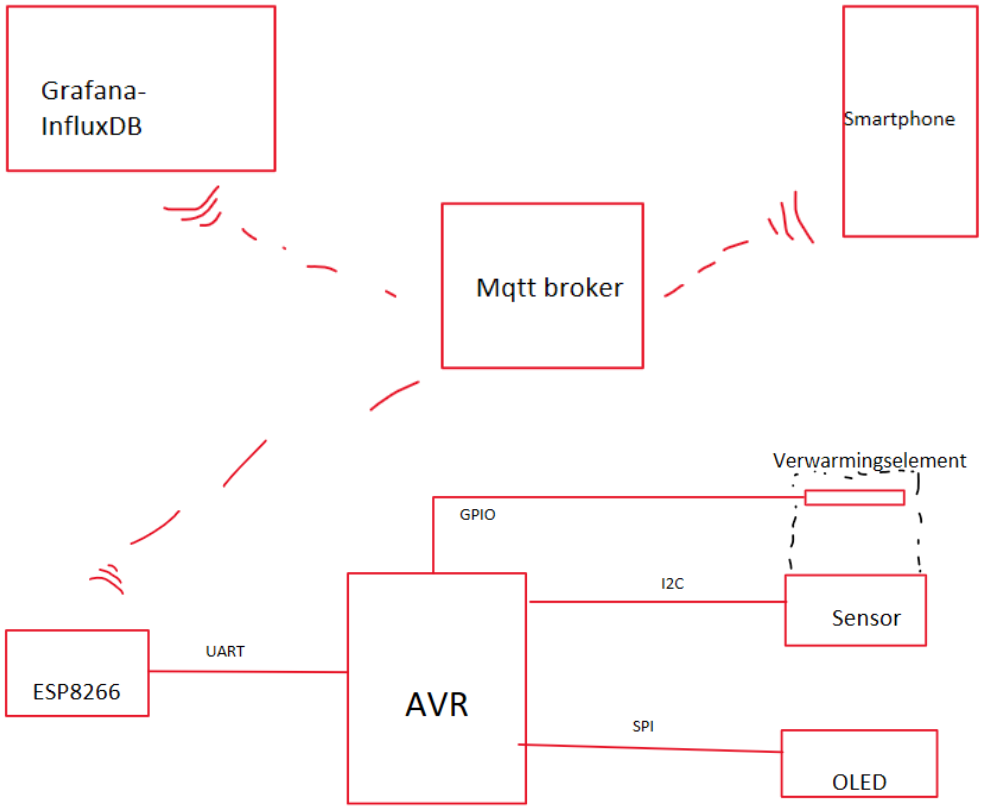
1.4 MQTT Dash

Om de gewenste temperatuur te sturen, alsook om de actuele temperatuur uit te lezen. Ik maak daarvoor gebruik van een App op mijn smartphone. Deze app noemt MQTT Dash, die is verbonden met de mosquitto broker uit Home Assistant en ik heb op beiden topics gesubscribed.

2 Hardware

2.1 Inleiding

Zoals ik eerder al kort heb aangehaald in Projectopdracht is het project het bouwen van een thermostaat. Die thermostaat moet kunnen verwarmen, communiceren met een wifi module, data met mqtt via wifi verzenden en ontvangen, alles kunnen weergeven op een display en data visualiseren in visualisatie software. In de afbeelding hieronder is een eenvoudig blokschema waar dat de elementen zichtbaar in zijn en volgens welke peripheral ze moeten communiceren.



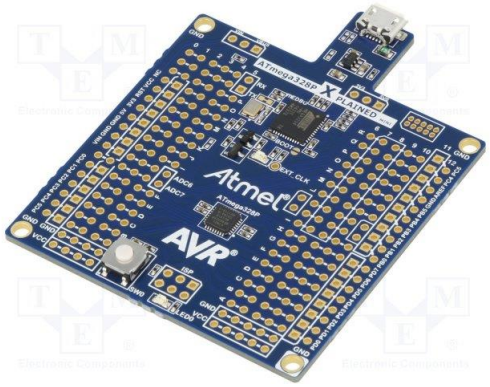
In dit project maken we gebruik van:

- AVR → microcontroller
- Wifi module → UART
- Verwarmingselement → GPIO
- Temperatuur sensor → I2C
- OLED display → SPI

2.2 Componenten

2.2.1 AVR (microcontroller)

Voor de microcontroller maken we gebruik van de Xplained mini 328pb.

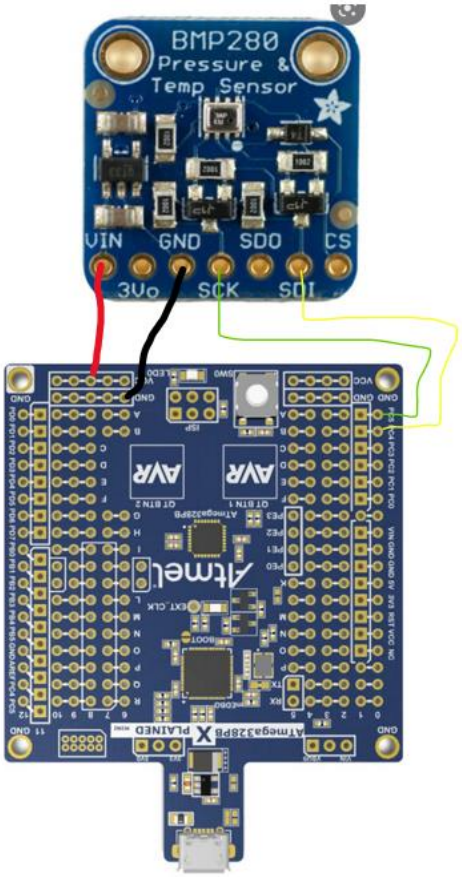


Dit is een laag voedende 8-bit microcontroller. Deze kunnen we rechtstreeks aansluiten aan onze PC via een USB-poort. Onze PC geeft instructies aan de MCU via het communicatie protocol UART. Deze microcontroller is geschikt voor onze thermostaat. Hij heeft de gewenste peripheral eigenschappen zoals de UART, I2C, SPI en PWM. Deze 4 zijn de gewenste communicatie protocollen waar we mee gaan werken. Nog andere eigenschappen van de microcontroller zijn dat hij werkt bij 1,8v tot 5,5v. Hij beschikt een temperatuur bereik van -40°C tot 105°C. Ook beschikt hij over een led en een drukknop. Vervolgens heeft hij een flashgeheugen van 32Kbyte en 2KByte SRAM.

2.2.2 BMP280 (Temperatuur sensor)

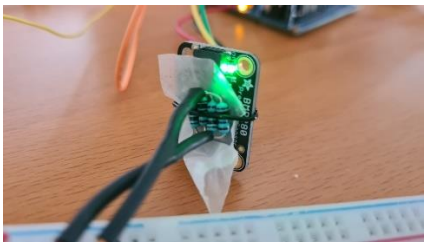
De sensor die we hiervoor gebruiken is de BMP280. Deze sensor kan zowel luchtdruk als temperatuur meten. De minimumwerking voltage bedraagt 1,71 Volt. Het heeft een drukresolutie van 0,16Pa en een temperatuur resolutie van 0,01°C. Het kan ook temperaturen meten van –40°C tot 85°C.

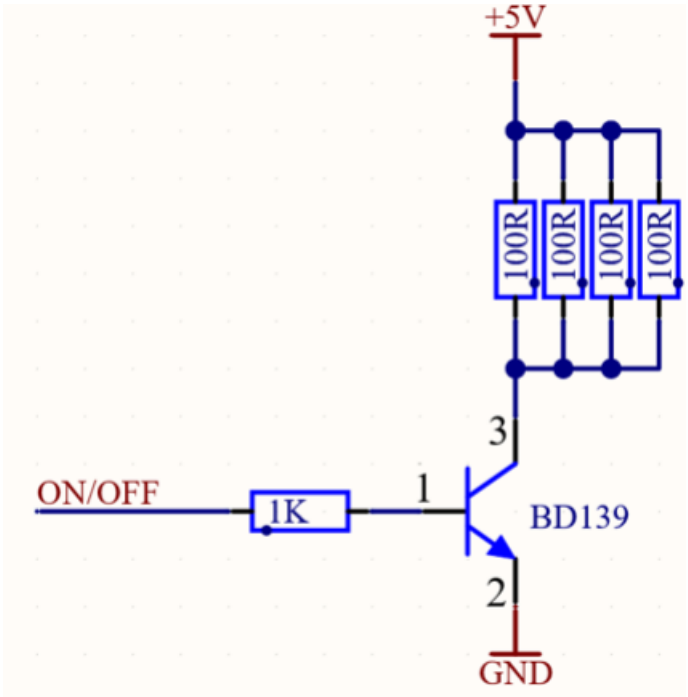
- De Vin wordt aangesloten op de VCC.
- De GND wordt aangesloten op de GND.
- De SCK wordt aangesloten op de PB5.
- De SDI wordt aangesloten op de PB4.



2.2.3 Verwarmingselement

Het verwarmingselement bestaat uit 4 x 100 ohm weerstanden in parallel. Samen is dit nog 25ohm in het circuit.



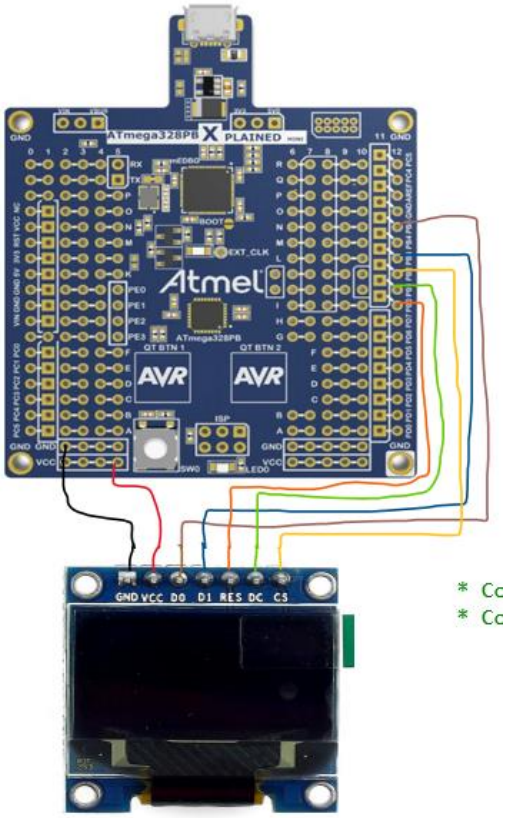


Onze verwarming wordt aangestuurd via een transistor. De aansluiting on/off word aangesloten op pin PD7 op onze AVR en op de basis van de transistor. Tussen deze aansluiting zit nog een weerstand van 1Kohm, dit is nodig aangezien de stroom anders te hoog gaat zijn voor de pin uitgang. Onze 4 weerstanden zijn aangesloten op de collector van de transistor en op de VCC van onze MCU. Deze kan alleen maar werken wanneer poort PD7 het signaal geeft naar onze transmitter om te schakelen. Als laatste hebben we ook nog de emitter die rechtstreeks aangesloten is op de grond.

2.2.4 OLED display

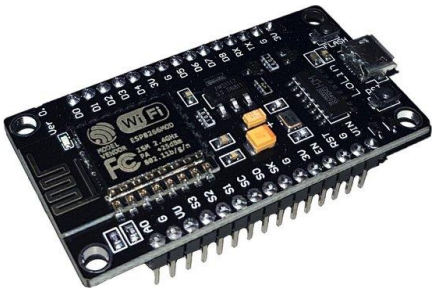
Het Oled display dat we gebruiken is ssd1306. Enkele kenmerken van deze display zijn, hij vraagt een spanning tussen 3.3 en 5 Volt. Resolutie van 128x64 en werkt tussen -40°C tot 70°C. Zoals je hierboven al kan zien heeft onze display 7 aansluitingen:

- GND is de aansluiting op de grond op onze MCU
- VCC is de aansluiting onze 5V van onze MCU
- D0 is de aansluiting van de klok op PB5 van onze MCU
- D1 is de aansluiting van de MOSI op PB3 van onze MCU
- RES is de aansluiting van de Reset op PB0 van onze MCU
- DC is de aansluiting van de Data command control pin op PB1 van onze MCU
- CS is de aansluiting van de chip select op PB2 van onze MCU



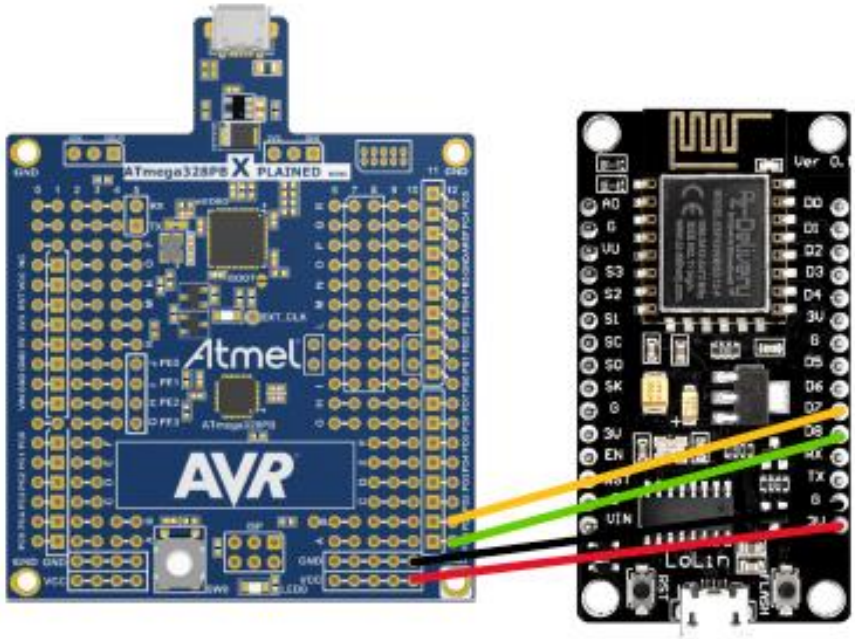
2.2.5 ESP8266 (wifi module)

De NodeMCU (Node MicroController Unit) is een open-source software- en hardwareontwikkelingsomgeving gebouwd rond een goedkope System-on-a-Chip (SoC) genaamd de ESP8266. De ESP8266, ontworpen en vervaardigd door Espressif Systems, bevat de cruciale elementen van een computer: CPU, RAM, netwerken (WiFi) en zelfs een modern besturingssysteem en SDK. Dat maakt het een uitstekende keuze voor allerlei soorten Internet of Things (IoT) projecten.



De NodeMCU heeft als specificaties:

- MCU: ESP-8266 32-bit
- Kloksnelheid: 80Mhz
- Bedrijfsspanning: 3.3V
- Flash-geheugen/SRAM: 4MB/ 64kB
- Wifi: 802,11 b/g/n
- Temperatuurbereik: -40°C – 125°C



3 Software

3.1 Inleiding

Voor elk onderdeel in de thermostaat wordt er een stuk software geschreven. De software wordt in lagen gemaakt.

1. Eerst wordt een laag geschreven dat "praat" met de hardware. In die laag gaan we er ook voor zorgen dat er data kan worden verstuurd en ontvangen via de hardware. Dit noemen we de **Hardware Abstraction Layer**, ook wel Low Level genoemd.

2. Dan hebben we nog een laag dat bepaald welke data er moet verzonden worden en wat er met ontvangen data moet gedaan worden. Dit noemen we **Middleware**, ook wel High level genoemd.

3. Uiteindelijk is er nog een laag dat gaat bepalen wanneer dat er iets moet verstuurd worden of wanneer dat we echts iets gaan doen met ontvangen data. Dit noemen we de **Application Layer**.

3.2 Hardware Abstraction Layer

De AVR (microcontroller) is het brein van het project. De eerste stap in het maken van de software is alle nodige peripherals configureren, anders is er geen communicatie naar de componenten mogelijk. Natuurlijk gaan we niet alle beschikbare peripherals gebruiken. We configureren enkel dat wat nodig is. Voor elke peripheral is er ook een apart .c bestand.

- **UART.c:** voor de communicatie met de wifi module
- **SPI.c:** voor de communicatie naar het OLED display
- **I2C.c:** voor de temperatuur sensor uit te lezen
- **HAL_GPIO.c:** voor de in- en uitgangen
- **TC1.c:** voor onderdelen in de software tijd gebaseerd te sturen.

3.2.1 UART.c

Om de UART te configureren heb ik gebruik gemaakt van een functie die eerder al is aangemaakt.

```
void init_UART(uint16_t l_UBRR, uint8_t l_doublespeed)
```

De functie kan worden aangepast door 2 waardes mee te geven.

- `uint16_t l_UBRR` Wordt gebruikt voor het instellen van de Baud Rate.
- `uint8_t l_doublespeed` Wordt gebruikt om de transfersnelheid te verdubbelen.

Om de UART te kunnen gebruiken moet in de registers worden geschreven.

In het **UBRR0** (USART Baud Rate Low and High byte) register:

Wordt gebruikt voor het instellen van de snelheid. Dit kan ofwel met een 8-bit getal of met een 16-bit. In geval van de 16-bit gebruiken we de 2 registers en anders enkel de Low byte.

- **UBRR0H:** High byte
- **UBRR0L:** Low byte

In het **UCSR0B** register:

- **Transmitter (TXEN0):** Om data te kunnen verzenden.
- **Receiver (RXEN0):** Om data te kunnen ontvangen
- **RX Complete Interrupt (RXCIE0):** Om de interrupt te triggeren bij ontvangst data.

In het **UCSR0B** register:

- **Stopbit (USB0):** Om in te stellen wat de stopbit is op het einde van de transitie.
- **Character size (UCSZ00):** Om de bit grote van de data te bepalen.

```
22  void init_UART(uint16_t l_UBRR, uint8_t l_doublespeed)
23  {
24      if (l_doublespeed)
25      {
26          UBRR0H = l_UBRR >> 8;
27          UBRR0L = l_UBRR;
28          UCSR0A |= (1<<U2X0);                                    // set double speed bit
29          UCSR0B |= (1<<TXEN0) | (1<<RXEN0) | (1<<RXCIE0);        // Enable RX, TX and RX interrupt
30          UCSR0C |= (0<<USBS0) | (3<<UCSZ00);                     // 8N1
31      }
32      else
33      {
34          UBRR0H = l_UBRR >> 8;
35          UBRR0L = l_UBRR;
36          UCSR0B |= (1<<TXEN0) | (1<<RXEN0) | (1<<RXCIE0);        // Enable RX, TX and RX interrupt
37          UCSR0C |= (0<<USBS0) | (3<<UCSZ00);                     // 8N1
38      }
39  }
```

3.2.1.1 UART_put..

In de UART hebben we 2 functies die iets versturen.

```
void putc_UART(uint8_t c)

void puts_UART(const uint8_t* s)
```

De functie `void putc_UART(uint8_t c)` gaat data die we meegeven versturen. In de functie gaan we door middel van het WHILE-statement wachten totdat USART Transmit Complete en USART Data Register Empty terug op 0 staan. Pas daarna gaan we de data dat is meegegeven aan de functie in het `UDR0` zetten.

```
41  void putc_UART(uint8_t c)
42  {
43      while(!(UCSR0A & (1<<UDRE0)));  // wait until sending is possible
44      UDR0 = c;                       // output character saved in c
45  }
46
47  void puts_UART(const uint8_t* s)
48  {
49      while(*s)
50      {
51          putc_UART(*s);
52          s++;
53      }
54  }
```

De andere functie wordt gebruikt om een string te versturen. In deze functie gaan we de functie `void putc_UART(uint8_t c)` gebruiken. Hier gaan we de string die wordt meegegeven blijven incrementeren tot dat we een terminated zero (0) tegen komen.

3.2.2 SPI.c

Voor de configuratie van de SPI peripheral heb ik ook gebruik gemaakt van een functie die al aangemaakt is in andere software.

```
void spi_init(uint8_t lsbfirst, uint8_t master, uint8_t mode, uint8_t
clkrate, uint8_t dblclk)
```

Ook deze functie kan je aanpassen door de waardes, die worden meegegeven, te veranderen.

- ***uint8_t* lsbfirst** Wordt gebruikt om te kiezen tussen MSB en LSB:
  MSB = 0
  LSB = 1
- *uint8_t* master Wordt gebruikt om te kiezen tussen master of slave:
  MASTER = 1
  SLAVE = 0
- *uint8_t* mode Wordt gebruikt om transfer mode in te stellen:
  MODE 0 = 0
  MODE 1 = 1
  MODE 2 = 2
  MODE 3 = 3
- *uint8_t* clkrate Wordt gebruikt om de clock rate in te stellen:
  CPUCLK/4 = 0
  CPUCLK/16 = 1
  CPUCLK/64 = 2
  CPUCLK/128 = 3
- *uint8_t* dblclk Wordt gebruikt om de clock rate te verdubbelen.
  DOUBLE CLOCK RATE = 1

In onderstaande afbeelding kan je de functie in detail zien.

```
44  void spi_init(uint8_t lsbfirst, uint8_t master, uint8_t mode, uint8_t clkrate, uint8_t dblclk)
45  {
46      //set outputs
47      __SPI_DDR |= ((1<<__SPI_MOSI) | (1<<__SPI_SCK) | (1<<__SPI_SS));
48      //set inputs
49      __SPI_DDR &= ~(1<<__SPI_MISO);
50      __SPI_PORT |= (1<<__SPI_MISO); //turn on pull-up resistor
51      //set SPI control register
52
53      SPCR0 = (
54          (1<<SPE) |                                  //enable SPI
55          ((lsbfirst & __SPI_LSBFIRST_MASK)<<DORD) |  //set msb/lsb ordering
56          ((master & __SPI_MASTER_MASK)<<MSTR) |      //set master/slave mode
57          ((mode & __SPI_MODE_MASK)<<CPHA) |          //set mode
58          (clkrate & __SPI_SPEED_MASK<<SPR0)          //set speed
59      );
60      //set double speed bit
61      SPSR0 = ((dblclk&__SPI_DBLCLK_MASK)<<SPI2X);
62  }
63
```

Figuur 1 spi_init HAL

Voor deze functie zijn een aantal defines aangemaakt en variabelen.

```
10  #define __SPI_PORT PORTB
11  #define __SPI_DDR DDRB
12
13  const uint8_t __SPI_MOSI = 3;
14  const uint8_t __SPI_MISO = 4;
15  const uint8_t __SPI_SCK = 5;
16  const uint8_t __SPI_SS = 2;
17  const uint8_t __SPI_LSBFIRST_MASK = 0b00000001;
18  const uint8_t __SPI_MASTER_MASK = 0b00000001;
19  const uint8_t __SPI_MODE_MASK = 0b00000011;
20  const uint8_t __SPI_SPEED_MASK = 0b00000011;
21  const uint8_t __SPI_DBLCLK_MASK = 0b00000001;
```

Figuur 2 spi_init defines variables

Als we de functie aanroepen gaan we eerst de IO instellen. Dat doen we door de volgende lijnen:

`__SPI_DDR |= ((1<<__SPI_MOSI) | (1<<__SPI_SCK) | (1<<__SPI_SS));`
In de defines zien we dat `__SPI_DDR` een vervanging is van `DDRB`. Verder gaan we hier gewoon PIN 3, 5 en 2 als output instellen.

`__SPI_DDR &= ~(1<<__SPI_MISO);` Hier gaan we PIN 4 instellen als input.

`__SPI_PORT |= (1<<__SPI_MISO);` En als laatste activeren we voor PIN 4 de pull-up weerstand.

Daarna gaan we onze configuratie uitvoeren in het `SPCR0` register te zien in onderstaande afbeelding.

```
53      SPCR0 = (
54          (1<<SPE) |
55          ((lsbfirst & __SPI_LSBFIRST_MASK)<<DORD) |
56          ((master & __SPI_MASTER_MASK)<<MSTR) |
57          ((mode & __SPI_MODE_MASK)<<CPHA) |
58          (clkrate & __SPI_SPEED_MASK<<SPR0)
```

Figuur 3 spi_init SPCR0

Hier gaan we eerst de peripheral activeren. De rest hangt af van de waardes die we de functie hebben meegegeven. De MASKS zorgen ervoor dat we niet meer bits kunnen activeren als waar ze voor bestemd zijn.

Als laatst gaan we eventueel de double speed instellen:
`SPSR0 = ((dblclk&__SPI_DBLCLK_MASK)<<SPI2X);` Hier hangt het ook weer van af wat we de functie hebben meegegeven.

3.2.3 I2C.c

Ook voor de configuratie van de I2C peripheral heb ik gebruik gemaakt van een functie die al aangemaakt is in andere software.

```
11  #define F_SCL 50000UL // SCL frequency
12  #define Prescaler 1
13  #define TWBR_val ((((F_CPU / F_SCL) / Prescaler) - 16 ) / 2)
14
15
16  void i2c_init(void)
17  {
18      TWBR0 = (uint8_t)TWBR_val;
19  }
```

TWBR selecteert de delingsfactor voor de bitrate-generator. De bitrate-generator is een frequentiedeler die de SCL-klokfrequentie genereert in de Master-modi. De waarde voor de delingsfactor word berekend in de define:

```
#define TWBR_val ((((F_CPU / F_SCL) / Prescaler) - 16 ) / 2)
```

3.2.4 TC1.c

Voor de configuratie van de Timer Counter peripheral heb ik gebruik gemaakt van een bestaande functie.

```
void init_TC1(unsigned int l_MaxCount)
```

In deze functie kan je het Output Compare Register aanpassen door de functie een ander waarde mee te geven.

In onderstaande afbeelding kan je de functie in detail zien.

```
20 void init_TC1(unsigned int l_MaxCount)
21 {
22     TCCR1B = (0<<CS12)|(1<<CS11)|(0<<CS10);     // Select CLKIO/8
23     OCR1A = l_MaxCount;                         // Set max count value to trigger interrupts
24     TIMSK1 = (1<<OCIE1A);                       // Enable output compare A match interrupt
25     sei();                                      // Enbale global interrupts
26 }
27
28
29 ISR(TIMER1_COMPA_vect)
30 {
31     TCNT1=0;                // Reset TC1 to 0
32     g_TickSensor++;             // General purpose tick timer
33     g_TickSetpoint++;
34     g_TickStatemachine++;
35     g_CheckSubRecv++;
36     g_LcdUpdateTimer++;     // Timer for updating LCD
37     g_DemoTimer++;          // Used for display demo
38 }
```

Figuur 4 init_TC1

`TCCR1B = (0<<CS12)|(1<<CS11)|(1<<CS10);`
In het TC Control Register 1 B gaan we de Clock Select bits instellen op /64.

`OCR1A = l_MaxCount;`
De waarde die we de functie hebben meegegeven zetten we in het Output Compare Register 1 A.

`TIMSK1 = (1<<OCIE1A);`
In het Timer/Counter 1 Interrupt Mask Register gaan we het Output Compare A Match Interrupt aangezet.

`sei();`
Hier gaan de globale interrupts aan zetten.

Als er een compare match interrupt heeft plaatsgevonden gaan we eerst Timer/Counter 1 Counter Value terug op nul zetten. Daarna gaan we alle variable incrementeren die we later in een hoger laag in de software gaan gebruiken.

3.2.5 HAL_GPIO.c

Ook voor de GPIO heb ik gebruik gemaakt van bestaande functies. In deze file heb ik wel een paar aanpassingen in gemaakt.

```
16  void init_GPIO()
17  {
18      DDRC = 0b00001000;          // Set PC3 as output, used for I2C display
19      DDRB = 0b00000111;          // Set PB0-2 as output, used for SPI display
20      DDRD = 0b10000000;          // Set PD7 as output, used for the heater
21  }
```

3.2.5.1 SetpinD

Bij de output configuratie heb ik het volgende toegevoegd:

```
DDRD = 0b10000000;
```

Hiermee stel ik bit 7 van het DDRD register als output in. Deze output ga ik gebruiken om mijn verwarmingselement te schakelen. Om dit te doen heb ik ook een functie bij aangemaakt, die universeel is voor pinD. Hiermee kan elke pinD geschakeld worden.

```
91  void setpinD(uint8_t l_pinnr, uint8_t l_status)
92  {
93      if (l_status)
94      {
95          PORTD = PORTD | (1<<l_pinnr);    // turn ON output
96      }
97      else
98      {
99          PORTD = PORTD &~(1<<l_pinnr);    // Turn OFF output
100     }
101 }
```

Met deze functie kunnen/ moeten we 2 waardes meegeven:

- *uint8_t* l_pinnr — Deze waarde wordt gebruikt om de Pin die we willen schakelen te selecteren.
- *uint8_t* l_status — Deze waarde wordt gebruikt om de pin AAN of UIT te schakelen.

3.3 Middleware

3.3.1 HAL_Init.c

Nu dat alle configuratie functies in de software zitten, gaan we ze ook moeten aanroepen. Om dat te doen is een aparte functie gemaakt `void HAL_Init()`. In deze functie worden ze één voor één aangeroepen. Hier gaan we de functies ook alle waardes meegeven die nodig zijn om de peripherals correct te configureren.

```
24  void HAL_Init()
25  {
26      init_GPIO();
27      init_TC1(20000);
28      init_UART(103, 0);
29      i2c_init();
30      spi_init (MSBFIRST, SPIMASTER, MODE3, 3, 0);
31
32      GPIO_OledReset(RESETSPI, 0);
33      GPIO_OledReset(RESETSPI, 1);
34      SSD1306_InitSPI();
35      BMP280_init();
36  }
37
```

Voor bepaalde functies hebben we waardes meegegeven:

- `init_TC1(20000)` 20000 wordt meegegeven voor een 10mS tick timer te hebben.
- `init_UART(103, 0);` 103 geeft een baud rate van 9600 aan. Dat is terug te vinden in de datasheet.

Baud Rate [bps]	f_{osc} = 16.0000MHz			
	U2Xn = 0		U2Xn = 1	
	UBRRn	Error	UBRRn	Error
2400	416	-0.1%	832	0.0%
4800	207	0.2%	416	-0.1%
9600	103	0.2%	207	0.2%

- `spi_init (MSBFIRST, SPIMASTER, MODE3, 3, 0)`
  - MSBFIRST → we gaan eerst de MSB sturen.
  - SPIMASTER → we gaan de SPI als master gebruiken.
  - MODE3 → we gebruiken mode 3
  - 3 → we gaan een snelheid van CPUCLK/128 gebruiken.
  - 0 → we gaan de doubleclk niet gebruiken.

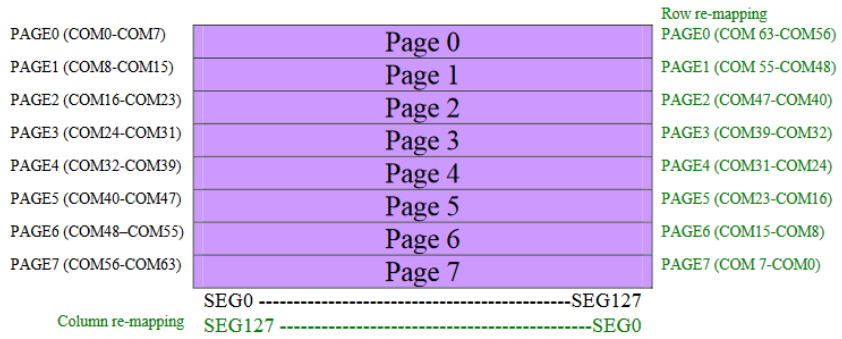
De functies op regels 32-35 zijn de drivers en configuraties van de sensor en het OLED display.

3.3.2 SSD1306.c

Dit is de driver van het OLED display. Hier ga ik maar 2 functies van gebruiken. Ik ga deze functies ook niet in detail bespreken omdat er ook niks in veranderd is. Aangezien ik deze functies wel ga gebruiken ga ik ze wel toelichten zodat duidelijk is waar ze voor dienen.

1. `void SSD1306_puts(uint8_t X, uint8_t Y, const char Text[], uint8_t FontSize, struct _SSD1306FrameBuffer* l_FrameBuffer);`

   Deze functie gebruik ik om tekst op het display te tonen. `uint8_t X` is de x-as van het display, de `uint8_t Y` is de y-as van het display.



   We hebben 8 pagina's, dat is onze Y-as. We hebben 127 segmenten en dat is onze X-as. Deze waarde kunnen we meegeven aan de functie.
   Het volgende wat we nog gaan meegeven is de array waar de data in staat dat we willen tonen `const char Text[]`. Dan moet er ook nog de fontsize worden meegegeven `uint8_t FontSize` . Daar hebben we keuze uit SMALL, MEDIUM en LARGE. En als laatste moeten we de pointer naar de framebuffer meegeven.
   Voor de fontsize zijn er defines aangemaakt in global.h

```
38  #define FONTSMALL   3
39  #define FONTMEDIUM  5
40  #define FONTLARGE   7
```

2. `void ProcessLCD(uint8_t l_Displaytype)`
   Deze functie gebruiken we enkel om het framebuffer naar het display te kopiëren.

3.3.3 Buffer.c

In de uiteindelijke software gaat er veel data worden ontvangen via de UART. Omdat de UART enkel maar byte per byte kan ontvangen moeten we onze ontvangen bytes ergens gaan stockeren zodat we ze naderhand kunnen raadplegen. Hier gaan we een buffer voor gebruiken. De buffer is aangemaakt in een struct.

```
12  #define RXBUFFERSIZE0 100
13
14  struct _RxBuffer
15  {
16      volatile unsigned char Buffer[RXBUFFERSIZE0];
17      volatile unsigned char WriteIndex;
18      volatile unsigned char IsDirty;
19      volatile unsigned char BufferOverflow;
20      unsigned char ReadIndex;
21      volatile uint8_t Ok;
22      volatile uint8_t CrLn;
23  };
```

Op lijn 16 hebben we een array aangemaakt met de naam Buffer. Voor de grote van de buffer hebben we een define: `RXBUFFERSIZE0 100` . De buffer heeft een grote van 100 bytes. Verder zijn er nog een aantal variabelen aangemaakt die van belang zijn voor de buffer.

- `WriteIndex` Om het adres aan te duiden waar de volgende byte mag komen.
- `BufferOverflow` Om te controleren als we het laatste adres van de buffer hebben bereikt.
- `Ok` Om aan te geven of er in de buffer 'OK' is binnengekomen.
- `CrLn` Om aan te geven of er in de buffer een Carriage return en een Line feed is binnengekomen.

Voor de buffer zijn ook nog 2 functionele functies.

```
27  signed int FindString(const uint8_t *l_String, struct _RxBuffer *l_UART_RxBuffer);
28
29  // Clear the buffer to all 0x0.
30  void ClearBuffer(struct _RxBuffer *l_UART_RxBuffer);
```

- `FindString` Om een string in de buffer te zoeken.
- `ClearBuffer` Om heel de buffer te wissen.

3.3.3.1 UART Interrupt

Van het moment dat er via de UART een byte binnenkomt wordt de interrupt getriggerd.

```
65  ISR (USART0_RX_vect)
66  {
67      UART_RxBuffer.Buffer[UART_RxBuffer.WriteIndex] = UDR0;
68
69      if (UART_RxBuffer.WriteIndex >= RXBUFFERSIZE0)
70      {
71          UART_RxBuffer.WriteIndex = 0;
72          UART_RxBuffer.BufferOverflow = 1;
73      }
74
75      if (UART_RxBuffer.Buffer[UART_RxBuffer.WriteIndex] == 0x0A)
76      {
77
78          if (UART_RxBuffer.Buffer[UART_RxBuffer.WriteIndex-1] == 0x0D)
79          {
80              UART_RxBuffer.CrLn = 1;
81          }
82
83      }
84
85      UART_RxBuffer.IsDirty=1;
86      UART_RxBuffer.WriteIndex++;
87  }
```

Als de interrupt is getriggerd gaan er regels code uitgevoerd worden:

1. **67:** Byte wordt in de buffer geplaatst
2. **69:** Controleren of de WriteIndex groter of gelijk is aan de buffergrote → Index terug op 0 gezet en een flag bufferoverflow op 1 zetten.
3. **75:** Controleren of de byte een Line feed is → controleren of de byte ervoor een Carriage return is → flag CrLn op 1 zetten.
4. **85:** Flag IsDirty op 1 zetten, wordt gebruikt om aan te geven dat de buffer veranderd is.
5. **86:** WriteIndex incrementeren.

3.3.4 Setheater

```
103 void setheater(uint8_t l_status)
104 {
105     setpinD(7, l_status);
106 }
```

Deze functie gaat de functie setpinD uitvoeren en pin 7 aan of uit zetten, naargelang wat we deze functie meegeven natuurlijk.

3.3.5 App_BMP280.c

Voor de temperatuur sensor hebben wij een driver gekregen. We moeten hier geen code voor schrijven. Ik ga hier dan ook niet dieper op in. Wat wel belangrijk is om te weten, is dat in `comp_data.temperature` de uitgelezen temperatuur staat. En dat de functie `get_data` ons verteld dat er communicatie is met de sensor. Alle andere functionaliteiten zoals het

verwarmingselement laten schakelen op de gewenste temperatuur, de temperatuur op het display te tonen, heb ik zelf moeten schrijven.

3.3.5.1 Sprintf

Sprintf is een functie uit de stdlib.h library die ik in de software raadpleeg.
`sprintf(char *__s, const char *__fmt, ...)`

Deze functie kan een bepaald datatype omzetten naar een string (array). Ik gebruik deze functie omdat het datatype van `comp_data.temperature` double is en `comp_data.setpoint` een uint16 is. Om het te tonen op het display hebben we een character array nodig.

3.3.5.2 Readsensor

Deze functie dient om te zien of er communicatie is met de sensor. Het enige wat ik hier doe is de bestaande functie uit de driver oproepen. Deze functie is alleen maar gemaakt om het duidelijk leesbaar te maken in de code.

```
132 uint8_t readsensor()
133 {
134     uint8_t l_result;
135 
136     l_result = get_data(meas_time, &conf, &dev, &comp_data);
137     return l_result;
138 }
```

Aan de functie hangt ook nog een define vast. Het resultaat van deze functie moet gelijk zijn aan deze define. Als dat niet zo is, is er geen communicatie.

```
106 /*! @name Success code*/
107 #define BMP2_OK                                    INT8_C(0)
```

3.3.5.3 Showsetpoint

```
31 void showsetpoint(struct bmp2_data *l_setpoint)
32 {
33     uint8_t l_setpointstring[5];
34 
35     sprintf(l_setpointstring, "%d", l_setpoint->setpoint);
36 
37     SSD1306_puts(0,3,"Setpoint:", FONTMEDIUM,&SSD1306FrameBuffer1);
38     SSD1306_puts(0,4,l_setpointstring, FONTMEDIUM,&SSD1306FrameBuffer1);
39 }
```

De functie Showsetpoint toont de gewenst temperatuur op het display.

De functie verwacht hier een pointer naar de struct `bmp2_data` .
Hier wordt er eerst een lokale array aangemaakt voor de conversie van

uint16_t naar een array. Die gaan we gebruiken in sprintf. Daarna gaan we via SSD1306_puts "setpoint: " tonen op het display en de gewenste temperatuur via de geconverteerde array. De werking van SSD1306_puts kan hier worden teruggevonden.

3.3.5.4 Showtemperature

De functie Showtemperature toont de actuele temperatuur op het display.

```
41  void showtemperature(struct bmp2_data *p_temp)
42  {
43      uint8_t l_temperaturestring[16];
44
45      // Convert the temperature and pressure from double to array of bytes
46      sprintf(l_temperaturestring, "%.5f", p_temp->temperature);
47
48      SSD1306_puts(0,0,"Temperature (degC):", FONTMEDIUM,&SSD1306FrameBuffer1);
49      SSD1306_puts(0,1,l_temperaturestring, FONTMEDIUM,&SSD1306FrameBuffer1);
50  }
```

De functie verwacht hier een pointer naar de struct bmp2_data .
In deze functie doen we hetzelfde als in Showsetpoint alleen converteren we de actuele temperatuur naar een array en tonen we de actuele temperatuur op het display.

3.3.5.5 Showsensorerror

De functie Showsensorerror toont de tekst "Sensor error" op het display.

```
52  void showsensorerror(void)
53  {
54      SSD1306_clear(&SSD1306FrameBuffer1);
55      SSD1306_puts(0,0,"Sensor error.     ", FONTMEDIUM,&SSD1306FrameBuffer1);
56  }
```

Hier gaan we eerst het display leeg maken met SSD1306_clear(&SSD1306FrameBuffer1) daarna gaan we met SSD1306_puts(0,0,"Sensor error.   ", FONTMEDIUM,&SSD1306FrameBuffer1) de tekst "Sensor error." op het display tonen.

3.3.5.6 Comparetemperature

De functie Comparetemperature gaat de actuele temperatuur vergelijken met de gewenste temperatuur.

```
59  uint8_t comparetemperature(struct bmp2_data *p_data)
60  {
61      uint8_t l_result;
62
63      if(p_data->temperature < p_data->setpoint)
64      {
65          l_result = 1;
66          SSD1306_puts(63,6,"                              ", FONTMEDIUM,&SSD1306FrameBuffer1);
67          SSD1306_puts(63,6,"ON", FONTMEDIUM,&SSD1306FrameBuffer1);
68      }
69      else
70      {
71          l_result = 0;
72          SSD1306_puts(63,6,"                              ", FONTMEDIUM,&SSD1306FrameBuffer1);
73          SSD1306_puts(63,6,"OFF", FONTMEDIUM,&SSD1306FrameBuffer1);
74      }
75      return l_result;
76  }
```

De functie verwacht hier een pointer naar de struct `bmp2_data` .
Eerst maken we een lokale variabelen aan `l_result.` Als op regel **63** de actuele temperatuur kleiner is dan de gewenste temperatuur dan zetten we op regel **65** `l_result` op 1. Door `SSD1306_puts` wordt het display leeg gemaakt op segment 63 van regel 6. Daarna gaan we op dezelfde plaats “ON” tonen. Als regel **63** niet “waar” is doen we hetzelfde alleen zetten we `l_result` op 0 en wordt op het display “OFF” getoond. Als laatste gaan we het resultaat met de functie terug sturen.

3.3.6 AT_commands_ESP8266.c

De ESP8266 wifi module communiceert met de microcontroller via UART. We moeten de module ook commando’s sturen om verbinden te maken met een wifi netwerk en gebruik te maken van het MQTT verhaal. Dit doen we door gebruik te maken van AT commando’s. AT-commando's zijn instructies die worden gebruikt om een modem te besturen. Maar we kunnen het ook gebruiken voor wifi en MQTT. AT is de afkorting van ATtention. Elke commando begint met "AT" gevolgd door een + teken en de commando dat nodig is. De commando moeten we ook versturen met als laatste een Carriage return en een Line feed. Voor elke commando dat ik naar de module wil sturen heb ik een aparte functie aangemaakt.

- AT_check → Om de communicatie met de module te controleren.
- AT_wificonnect → Om te connecteren met een wifi netwerk.
- AT_mqtt_userconf → Om de user gegevens te configureren.
- AT_mqtt_connection → Om te connecteren met een broker.
- AT_mqtt_subscribe → Om te subcriben op een topic.
- AT_mqtt_publish → Om data te publishen naar een topic.

3.3.6.1 Voorbereiding

Om deze functies universeel te kunnen gebruiken ga ik werken met pointers. Dan kan het wifi netwerk, de user gegevens voor mqtt, de mqtt broker, de topics allemaal worden aangepast buiten de functies. Hiervoor heb ik 2 structs aangemaakt.

```
20  //struct for the wifi data
21  struct _wifi
22  {
23      unsigned char WifiSSID[12];
24      unsigned char WifiPassword[11];
25
26  };
27
28  // struct for data to use mqtt
29  struct _mqtt
30  {
31      unsigned char username[6];
32      unsigned char password[6];
33      unsigned char broker_IP[14];
34      unsigned char client_ID[8];
35      unsigned char topic_pub[11];
36      unsigned char topic_sub[10];
37      unsigned char data[10];
38  };
```

Er is een struct specifiek voor het wifi netwerk en specifiek voor mqtt. Voor het wifi netwerk hebben we 2 zaken nodig:

- `WifiSSID` Voor het wifi netwerk waarmee we willen connecteren.
- `WifiPassword` Voor het paswoord van het wifi netwerk.

Dan hebben we nog 7 zaken nodig voor mqtt:

- `username` Een username als de broker dat verwacht.
- `password` Een paswoord van de user als de broker dat verwacht.
- `broker_IP` Het IP-adres van de broker.
- `client_ID` Het ID van de client, in dit geval de wifi module.
- `topic_pub` Een topic waar we op willen publishen.
- `topic_sub` Een topic waar we op willen subcriben.

3.3.6.2 AT_check

Deze functie stuurt het commando om de communicatie te controleren. De commando die we gaan sturen is **AT\r\n**”. Dit doen we door gebruik te maken van de functie puts_UART.

```
31  // Check if the interaction is OK
32  void AT_check(void)
33  {
34      puts_UART("AT\r\n");
35  }
```

3.3.6.3 AT_wificonnect

Deze functie stuurt het commando om te connecteren met een wifi netwerk. Deze functie verwacht 2 pointers. De 2 zijn allebei pointers naar de struct `_wifi`. We gaan die gebruiken zodat we de SSID en het paswoord kunnen aanpassen. Het commando dat wordt gestuurd om te connecteren is:

AT+CWJAP=" *`SSID` "," *`password` "\r\n

We gaan ook weer gebruik maken van de functie puts_UART. Dit wordt wel in enkele stappen gedaan. De waarde 34 die wordt gestuurd is de decimale waarde voor **"**.

```
37  // Connect to a wifi network
38 void AT_wificonnect(struct _wifi *SSID, struct _wifi *password)
39  {
40      puts_UART("AT+CWJAP=");
41      putc_UART(34);
42      puts_UART(SSID->WifiSSID);
43      putc_UART(34);
44      puts_UART(",");
45      putc_UART(34);
46      puts_UART(password->WifiPassword);
47      putc_UART(34);
48      puts_UART("\r\n");
49  }
```

3.3.6.4 AT_mqtt_userconf

Deze functie stuurt het commando om de user gegevens te configureren zodat we kunnen connecteren met de broker. Deze functie verwacht 3 pointers. De 3 zijn allemaal pointers naar de struct `_mqtt`. Die worden gebruikt voor de username, password en id. Het commando dat wordt gestuurd is:

AT+MQTTUSERCFG=0,1," *`id` "," *`user` "," *`password` ",0,0,""\r\n

```
51  // Set the user configuration
52 void AT_mqtt_userconf(struct _mqtt *user, struct _mqtt *password, struct _mqtt *id)
53  {
54      puts_UART("AT+MQTTUSERCFG=0,1,");
55      putc_UART(34); // show ASCII "
56      puts_UART(id->client_ID);
57      putc_UART(34);
58      puts_UART(",");
59      putc_UART(34);
60      puts_UART(user->username);
61      putc_UART(34);
62      puts_UART(",");
63      putc_UART(34);
64      puts_UART(password->password);
65      putc_UART(34);
66      puts_UART(",0,0,");
67      putc_UART(34);
68      putc_UART(34);
69      puts_UART("\r\n");
70  }
```

3.3.6.5 AT_mqtt_connection

Deze functie stuurt het commando om te connecteren met de mqtt broker. Deze functie verwacht 1 point naar de struct `_mqtt`. Die wordt gebruikt voor IP-adres van de mqtt broker Het commando dat wordt gestuurd om te connecteren is:

AT+MQTTCONN=0," *`ip` ",1883,0\r\n

```
72  // Connect to mqtt broker
73  void AT_mqtt_connection(struct _mqtt *ip)
74  {
75      puts_UART("AT+MQTTCONN=0,");
76      putc_UART(34);
77      puts_UART(ip->broker_IP);
78      putc_UART(34);
79      puts_UART(",1883,0\r\n");
80  }
```

3.3.6.6 AT_mqtt_publish

Deze functie stuurt het commando om data te publishen naar een topic. Deze functie verwacht 2 pointers naar de struct `_mqtt`. Die worden gebruikt voor het topic en de data. Het commando dat wordt gestuurd is:

AT+MQTTPUB=0," *`topic` "," * `data` ",1,0\r\n

```
82  // Publish to mqtt topic
83  void AT_mqtt_publish(struct _mqtt *topic, struct _mqtt *data)
84  {
85      puts_UART("AT+MQTTPUB=0,");
86      putc_UART(34);
87      puts_UART(topic->topic_pub);
88      putc_UART(34);
89      puts_UART(",");
90      putc_UART(34);
91      puts_UART(data->data);
92      putc_UART(34);
93      puts_UART(",1,0\r\n");
94  }
```

3.3.6.7 AT_mqtt_subcribe

Deze functie stuurt het commando om te subcriben op een topi. Deze functie verwacht één pointer naar de struct `_mqtt`. Die wordt gebruikt om het topic aan te geven waar we op willen subcriben. Het commando dat wordt gestuurd om te connecteren is:

AT+MQTTSUB=0," *`topic` ",1\r\n

```
96  // Subcribe to mqtt topic
97  void AT_mqtt_subscribe(struct _mqtt *topic)
98  {
99      puts_UART("AT+MQTTSUB=0,");
100     putc_UART(34);
101     puts_UART(topic->topic_sub);
102     putc_UART(34);
103     puts_UART(",1\r\n");
104 }
```

3.3.7 Statemachine.c

De AT commando's waar ik functies voor heb aangemaakt moeten één voor één verstuurd worden, in een bepaalde volgorde. Sommige functies hoeven ook maar één keer uitgevoerd te worden terwijl er andere meermaals achter elkaar worden uitgevoerd. De wifi module stuurt ook "OK" terug als de commando correct is uitgevoerd. Dat moet ook gecontroleerd worden alvorens een volgende functie wordt uitgevoerd. Om dit te verwezenlijken ga ik gebruik maken van een statemachine.

Een statemachine is een algoritme die een eindige reeks toestanden definieert: één toestand wordt gedefinieerd als de begintoestand. Wanneer we de statemachine beginnen uit te voeren, komt deze automatisch in deze toestand. Elke status kan acties definiëren die plaatsvinden wanneer een machine die status binnengaat of verlaat.

3.3.7.1 Voorbereiding

Voor een statemachine te kunnen maken moete we wat voorbereidingen maken. Er moeten een aantal defines worden aangemaakt om de verschillende statussen aan te geven. Deze defines geven aan wat er moet uitgevoerd worden. De defines maken we aan in de header file van de statemachine.

```
12  //defines used in the statemachine
13  #define ATCHECK             1
14  #define ATCHECKOK           2
15  #define ATWIFICONNECT       3
16  #define ATUSERCONF          4
17  #define ATMQTTCONNECTION    5
18  #define ATMQTTSUB           6
19  #define ATMQTTPUB           7
```

We hebben ook een aantal variabelen nodig in de statemachine. Die worden gebruikt om de huidige status te definiëren en de volgende status aan te geven. We gebruiken ook een timer in de statemachine.

```
33  uint8_t g_AT_State = ATCHECK;
34  uint8_t g_AT_NextState;
35  volatile uint8_t g_tickTimeOut = 0;
```

3.3.7.2 Strcpy

Dit is een functie uit de string.h library. Deze functie kopieert een string naar een array. Dit hebben we nodig omdat we de arrays in de structs nog een string moeten toekennen.

```
strcpy(*destination, "*string")
```

Bij destination geven we de array aan waar de string in moet komen, bij string geven we de string mee dat we willen kopiëren.

3.3.7.3 Switch – case

Dit statement gaan we gebruiken om aan te geven naar welke state we moeten naartoe gaan. De switch statement stelt ons in staat om één codeblok uit vele alternatieven uit te voeren.

Stel we hebben dit:

```
switch (3)
```

Dan zal de code worden uitgevoerd waarbij de `case` 3 is. Al de andere cases gaan niet uitgevoerd worden.

`break` dient om de case te stoppen en terug uit de functie te gaan. We gaan enkel de huidige state uitvoeren.

In de statemachine ziet het er zo uit:

```
41  void AT_statemachine(void)
42  {
43      switch (g_AT_State)
44      {
45          case ATCHECK:
46          AT_check();
47          g_AT_State = ATCHECKOK;
48          g_AT_NextState = ATWIFICONNECT;
49          break;
50
51          case ATWIFICONNECT:
52          strcpy(str_AT_wifi.WifiSSID, "dlink-5914");
53          strcpy(str_AT_wifi.WifiPassword, "cjvgu89628");
54          AT_wificonnect(&str_AT_wifi, &str_AT_wifi);
55          g_AT_State = ATCHECKOK;
56          g_AT_NextState = ATUSERCONF;
57          break;
```

De switch-statement gaat naar de case die in de variabele `g_AT_State` staat. De eerste keer dat de statemachine wordt uitgevoerd, gaat ATCHECK worden uitgevoerd. Dat komt omdat we dit hebben toegekend aan de variabele `g_AT_State = ATCHECK;`

3.3.7.4 Case ATCHECK

Deze staat gaat als eerste worden uitgevoerd. De bedoeling van deze state is dat de communicatie wordt gecontroleerd.

```
45          case ATCHECK:
46          AT_check();
47          g_AT_State = ATCHECKOK;
48          g_AT_NextState = ATWIFICONNECT;
49          break;
```

1. AT_check() uitvoeren.
2. De define `ATCHECKOK` in de variabele `g_AT_State` zetten.
3. De define `ATWIFICONNECT` in de variabele `g_AT_NextState` zetten om de volgende state te definiëren.
4. Case stoppen.

3.3.7.5 Case ATCHECKOK

In het begin van de titel statemachine heb ik al uitgelegd dat de wifi module OK terug stuurt als bevestiging dat de commando geslaagd is. Dit gaan we in deze state controleren. Dit doen we altijd voor dat we naar een volgende commando state gaan. Elke keer dat de communicatie is mislukt gaan we terug opnieuw beginnen met de communicatie te checken.

```
90          case ATCHECKOK:
91          if (UART_RxBuffer.Ok)
92          {
93              g_tickTimeOut = 0;
94              UART_RxBuffer.Ok = 0;
95              g_AT_State = g_AT_NextState;
96          }
97          else if (g_tickTimeOut++ > 10)
98          {
99              g_tickTimeOut = 0;
100             g_AT_State = ATCHECK;
101         }
102         break;
```

Wat we hier doen is controleren of er een flag `UART_RxBuffer.Ok` is. Als deze toestand waar is gaan we eerst `g_tickTimeOut` en `UART_RxBuffer.Ok` terug op 0 zetten, daarna gaan we de variabele `g_AT_NextState` in `g_AT_State` zetten zodat we verder kunnen naar een volgende state. Als de toestand niet waar is gaan we `g_tickTimeOut` incrementeren en controleren als de waarde groter is dan 10, is deze toestand waar is dan gaan we `g_tickTimeOut` terug op 0 zetten en ATCHECK als huidige state definiëren. Dat wil zeggen dat er iets mis is en dat we terug opnieuw moeten beginnen.

3.3.7.6 Case ATWIFICONNECT

Als Case ATCHECK gecontroleerd is gaan we deze case uitvoeren. Deze case gaat de verbinding met het wifi netwerk tot stand brengen.

```
51          case ATWIFICONNECT:
52          strcpy(str_AT_wifi.WifiSSID, "dlink-5914");
53          strcpy(str_AT_wifi.WifiPassword, "cjvgu89628");
54          AT_wificonnect(&str_AT_wifi, &str_AT_wifi);
55          g_AT_State = ATCHECKOK;
56          g_AT_NextState = ATUSERCONF;
57          break;
```

In deze state gaan we eerst de SSID en paswoord kopiëren naar de arrays door middel van strcpy . Daarna gaan we de functie AT_wificonnect uitvoeren. Dan gaan we weer terug de controle laten uitvoeren, en als volgende state de user configuratie laten uitvoeren.

3.3.7.7 Case ATUSERCONF

Als Case ATWIFICONNECT gecontroleerd is gaan we deze case uitvoeren. Deze case gaat de configuratie van de user en id voor de mqtt broker tot stand brengen.

```
59          case ATUSERCONF:
60          strcpy(str_AT_mqtt.username, "mqtt");
61          strcpy(str_AT_mqtt.password, "mqtt");
62          strcpy(str_AT_mqtt.client_ID, "ESP8266");
63          AT_mqtt_userconf(&str_AT_mqtt, &str_AT_mqtt, &str_AT_mqtt);
64          g_AT_State = ATCHECKOK;
65          g_AT_NextState = ATMQTTCONNECTION;
66          break;
```

Hier wordt ook weer eerst alle strings gekopieerd door middel van strcpy. Daarna gaan we de functie AT_mqtt_userconf uitvoeren. Dan gaan we weer terug de controle laten uitvoeren en als volgende state de mqtt broker connectie laten uitvoeren.

3.3.7.8 Case ATMQTTCONNECTION

Als Case ATUSERCONF gecontroleerd is gaan we deze case uitvoeren. Deze case gaat de connectie maken met de mqtt broker.

```
68          case ATMQTTCONNECTION:
69          strcpy(str_AT_mqtt.broker_IP, "192.168.0.102");
70          AT_mqtt_connection(&str_AT_mqtt);
71          g_AT_State = ATCHECKOK;
72          g_AT_NextState = ATMQTTSUB;
73          break;
```

De string van het IP adres wordt hier eerst gekopieerd, daarna wordt AT_mqtt_connection uitgevoerd. Daarna wordt de controle state en de volgende state gedefinieerd.

3.3.7.9 Case ATMQTTSUB

Als Case ATMQTTCONNECTION gecontroleerd is gaan we deze case uitvoeren. Deze case gaat subcriben op het topic "setpoint".

```
75          case ATMQTTSUB:
76          strcpy(str_AT_mqtt.topic_sub , "SETPOINT");
77          AT_mqtt_subscribe(&str_AT_mqtt);
78          g_AT_State = ATCHECKOK;
79          g_AT_NextState = ATMQTTPUB;
80          break;
```

De string van het topic wordt eerst gekopieerd. Daarna wordt AT_mqtt_subcribe uitgevoerd. En dan volgt de controle state en de volgende state die gedefinieerd worden.

3.3.7.10 Case ATMQTTPUB

Als Case ATMQTTCONNECTION gecontroleerd is gaan we deze case uitvoeren. Deze case gaat subcriben op het topic "setpoint".

```
82          case ATMQTTPUB:
83          strcpy(str_AT_mqtt.topic_pub , "TEMPSENSOR");
84          AT_mqtt_publish(&str_AT_mqtt, &str_AT_mqtt);
85          ClearBuffer(UART_RxBuffer.Buffer);
86          g_AT_State = ATMQTTPUB;
87          break;
```

De string van het topic wordt eerst gekopieerd. Nadien wordt AT_mqtt_publish uitgevoerd. Daarna wordt de functie ClearBuffer uitgevoerd om ervoor te zorgen dat de buffer niet overvol gaat zitten.

3.4 Application layer

In deze laag gaan we alles wat we hiervoor hebben gebouwd rechtstreeks en onrechtstreeks gebruiken. Hier voegen we alles samen zodat we tot één gehele functionele software komen. Deze laag regelt wanneer dat welke functies of algoritmes moeten uitgevoerd worden.

3.4.1 Main(void)

De main(void) functie is het begin van de werkende software. Als de software begint met runnen, dan zal die hier starten. Hier gaan we definiëren wat de volgorde is voor het uitvoeren van de functies of algoritmes.

Het eerst dat moet gebeuren is de configuratie van de peripherals. Dat doen we door da functie HAL_init() op te roepen. Als volgende gaan we in de While(1) loop. Dit is een oneindige lus, de software zal hier altijd code in blijven uitvoeren. Dit komt omdat de voorwaarde altijd waar is (1).

```
58  int main(void)
59  {
60      HAL_Init();
61
62      while (1)
63      {
64          if (UART_RxBuffer.CrLn == 1)
65          {
```

3.4.1.1 if (UART_RxBuffer.CrLn == 1)

Dit statement wordt gebruikt om te zien of er data is binnengekomen van een gesubcribed topic. Die data wordt dan uit de buffer gehaald en in een variabele gezet. Verder dient dit statement ook om de string OK te gaan zoeken in de buffer om de OK flag op 1 te zetten voor de statemachine.

```
62      while (1)
63      {
64          if (UART_RxBuffer.CrLn == 1)            // check if there is an flag Carriage return and line feed
65          {
66              UART_RxBuffer.CrLn = 0;             // Sets the flag back to 0
67
68              if ((FindString("+MQTTSUBRECV", &UART_RxBuffer)) >= 0)  // check if there is a match for mqtt subscribed received data
69              {
70                  g_SubRecvIndex = FindString("SETPOINT", &UART_RxBuffer); // check if the subcribed received data is a setpoint
71                  if (g_SubRecvIndex >= 0)
72                  {
73                      comp_data.setpoint = ((UART_RxBuffer.Buffer[g_SubRecvIndex+12]-48) *10) + (UART_RxBuffer.Buffer[g_SubRecvIndex+13]-48);
74                  }
75
76              }
77              else if (FindString("OK", &UART_RxBuffer))
78              {
79                  UART_RxBuffer.Ok = 1;
80              }
81              else
82              {
83                  ClearBuffer(&UART_RxBuffer);
84              }
85
86          }
```

In de interrupt van de buffer is er een flag aangemaakt die op 1 wordt gezet als er een Carriage return en een Line feed is binnengekomen in de buffer. Het eerste wat we doen is die flag terug op 0 zetten.

```
62      while (1)
63      {
64          if (UART_RxBuffer.CrLn == 1)            // check if there i
65          {
66              UART_RxBuffer.CrLn = 0;             // Sets the flag ba
67
68              if ((FindString("+MQTTSUBRECV", &UART_RxBuffer)) >= 0)
69              {
```

Daarna gaan we in een volgend if-statement via de functie `FindString` in de buffer zoeken naar de string `"+MQTTSUBRECV"` . Als we een match vinden dan zal deze toestand waar zijn en dan gaan we verder in het if-statement. Als er geen match is zal de functie een -1 terug geven. Vandaar dat de voorwaarde groter of gelijk moet zijn aan 0.

Stel de voorwaarde is waar, dan gaan we terug via diezelfde functie `FindString` zoeken naar de string `"SETPOINT"` . Als er een match is geeft deze functie de plaats weer in de buffer waar het eerst karakter van deze string staat. Die index wordt in de variabele `g_SubRecvIndex` geplaatst. Deze hebben we nodig om eerst te controleren of “setpoint” effectief in de buffer staat. Dat doen we weer in een if-statement.

```
70              g_SubRecvIndex = FindString("SETPOINT", &UART_RxBuffer); // check if the subcribed received data is a setpoint
71              if (g_SubRecvIndex >= 0)
72              {
73                  comp_data.setpoint = ((UART_RxBuffer.Buffer[g_SubRecvIndex+12]-48) *10) + (UART_RxBuffer.Buffer[g_SubRecvIndex+13]-48);
74              }
75
```

Daarna gaan we in de variabele `setpoint` van de struct `comp_data` de output van de volgende berekening zetten.

```
((UART_RxBuffer.Buffer[g_SubRecvIndex+12]-48) *10) +
(UART_RxBuffer.Buffer[g_SubRecvIndex+13]-48)
```

Op de positie `g_SubRecvIndex+12` gaat de eerste byte van de data dat we nodig hebben staan. Dit is een decimale waarde. Wij hebben het karakter nodig dus dan moet we -48 doen. Dit is een tiental getal, die moeten we dan vermenigvuldigen met 10. Op positie `g_SubRecvIndex+13` staat de tweede byte dat we nodig hebben. Hier moeten we ook weer 48 van af trekken en dan met elkaar optellen. We hebben onze setpoint nu.

Stel dat het statement `if ((FindString("+MQTTSUBRECV", &UART_RxBuffer)) >= 0)` niet waar is, dan gaan we verder naar het else if statement.

```
}
else if (FindString("OK", &UART_RxBuffer))
{
    UART_RxBuffer.Ok = 1;
}
else
{
    ClearBuffer(&UART_RxBuffer);
}
```

Hier gaan we dan zoeken naar OK in de buffer. Als OK is gevonden in de buffer gaan we de flag op 1 zetten want die is nodig in de controle van de statemachine. Anders gaan we de buffer leeg maken.

3.4.1.2 if (g_TickSensor > 100)

Hier werken we met een ticktimer, we gaan om de 100mS dit stuk code uitvoeren. Deze code dient om eerst te controleren als we communicatie hebben met de temperatuur sensor, zo niet gaan we op regel **109** showsensorerror() uitvoeren.

```
88          if (g_TickSensor > 100)
89          {
90              g_TickSensor = 0;
91              if (readsensor() == BMP2_OK)
92              {
93                  // Convert the temperature from uint16 to array of bytes
94                  sprintf(str_AT_mqtt.data, "%.5f", comp_data.temperature);
95
96                  showtemperature(&comp_data);
97                  showsetpoint(&comp_data);
98                  if (comparetemperature(&comp_data))
99                  {
100                     setheater(ON);
101                 }
102                 else
103                 {
104                     setheater(OFF);
105                 }
106             }
107             else
108             {
109                 showsensorerror();
110             }
111         }
```

Anders gaan we eerst op regel 94 sprintf uitvoeren zodat de actuele temperatuur wordt geconverteerd naar de array die we gebruiken om via de functie AT_mqtt_publish de data naar de broker te sturen. Daarna gaan we de functies showtemperature en showsetpoint uitvoeren.

Daarna voeren we in een if-statement de functie comparetemperature uit. Als de uitkomst waar is voeren we de functie setheater(ON) uit en anders voeren we setheater(OFF).

3.4.1.3 if (g_TickStatemachine > 500)

Hier gaan we om de 500mS het algoritme AT_statemachine oproepen.