Inhoudsopgave

[1 Het project 4](#_Toc106051698)

[1.1 Projectopdracht 4](#_Toc106051699)

[1.2 MQTT 4](#_Toc106051700)

[1.3 Home Assistant 5](#_Toc106051701)

[1.3.1 InfluxDB 5](#_Toc106051702)

[1.3.2 Grafana 5](#_Toc106051703)

[1.4 MQTT Dash 6](#_Toc106051704)

[2 Hardware 7](#_Toc106051705)

[2.1 Inleiding 7](#_Toc106051706)

[2.2 Componenten 8](#_Toc106051707)

[2.2.1 AVR (microcontroller) 8](#_Toc106051708)

[2.2.2 BMP280 (Temperatuur sensor) 9](#_Toc106051709)

[2.2.3 Verwarmingselement 10](#_Toc106051710)

[2.2.4 OLED display 11](#_Toc106051711)

[2.2.5 ESP8266 (wifi module) 12](#_Toc106051712)

[3 Software 13](#_Toc106051713)

[3.1 Inleiding 13](#_Toc106051714)

[3.2 Hardware Abstraction Layer 13](#_Toc106051715)

[3.2.1 UART.c 14](#_Toc106051716)

[3.2.1.1 UART\_put.. 15](#_Toc106051717)

[3.2.2 SPI.c 16](#_Toc106051718)

[3.2.3 I2C.c 18](#_Toc106051719)

[3.2.4 TC1.c 18](#_Toc106051720)

[3.2.5 HAL\_GPIO.c 20](#_Toc106051721)

[3.2.5.1 SetpinD 20](#_Toc106051722)

[3.3 Middleware 21](#_Toc106051723)

[3.3.1 HAL\_Init.c 21](#_Toc106051724)

[3.3.2 SSD1306.c 22](#_Toc106051725)

[3.3.3 Buffer.c 22](#_Toc106051726)

[3.3.3.1 UART Interrupt 23](#_Toc106051727)

[3.3.4 Setheater 24](#_Toc106051728)

[3.3.5 App\_BMP280.c 24](#_Toc106051729)

[3.3.5.1 Sprintf 26](#_Toc106051730)

[3.3.5.2 Readsensor 26](#_Toc106051731)

[3.3.5.3 Showsetpoint 26](#_Toc106051732)

[3.3.5.4 Showtemperature 27](#_Toc106051733)

[3.3.5.5 Showsensorerror 27](#_Toc106051734)

[3.3.5.6 Comparetemperature 28](#_Toc106051735)

[3.3.6 AT\_commands\_ESP8266.c 28](#_Toc106051736)

[3.3.6.1 Voorbereiding 29](#_Toc106051737)

[3.3.6.2 AT\_check 29](#_Toc106051738)

[3.3.6.3 AT\_wificonnect 30](#_Toc106051739)

[3.3.6.4 AT\_mqtt\_userconf 30](#_Toc106051740)

[3.3.6.5 AT\_mqtt\_connection 31](#_Toc106051741)

[3.3.6.6 AT\_mqtt\_publish 31](#_Toc106051742)

[3.3.6.7 AT\_mqtt\_subcribe 32](#_Toc106051743)

[3.3.7 Statemachine.c 33](#_Toc106051744)

[3.3.7.1 Voorbereiding 33](#_Toc106051745)

[3.3.7.2 Strcpy 33](#_Toc106051746)

[3.3.7.3 Switch – case 34](#_Toc106051747)

[3.3.7.4 Case ATCHECK 35](#_Toc106051748)

[3.3.7.5 Case ATCHECKOK 35](#_Toc106051749)

[3.3.7.6 Case ATWIFICONNECT 36](#_Toc106051750)

[3.3.7.7 Case ATUSERCONF 36](#_Toc106051751)

[3.3.7.8 Case ATMQTTCONNECTION 36](#_Toc106051752)

[3.3.7.9 Case ATMQTTSUB 37](#_Toc106051753)

[3.3.7.10 Case ATMQTTPUB 37](#_Toc106051754)

[3.4 Application layer 38](#_Toc106051755)

[3.4.1 Main(void) 38](#_Toc106051756)

[3.4.1.1 if (UART\_RxBuffer.CrLn == 1) 38](#_Toc106051757)

[3.4.1.2 if (g\_TickSensor > 100) 40](#_Toc106051758)

[3.4.1.3 if (g\_TickStatemachine > 500) 41](#_Toc106051759)

# 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.

## 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

## 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.

## 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.

### 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.

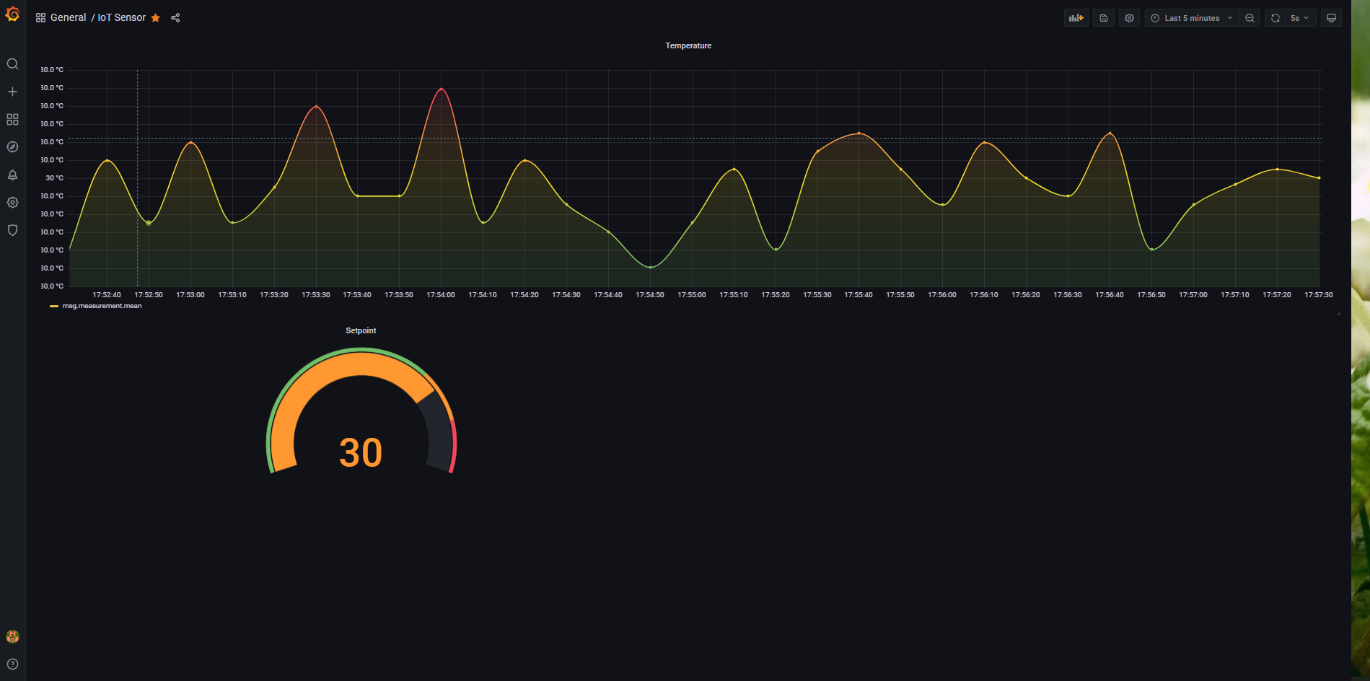
Hier heb ik 2 databases aangemaakt:

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

### Grafana

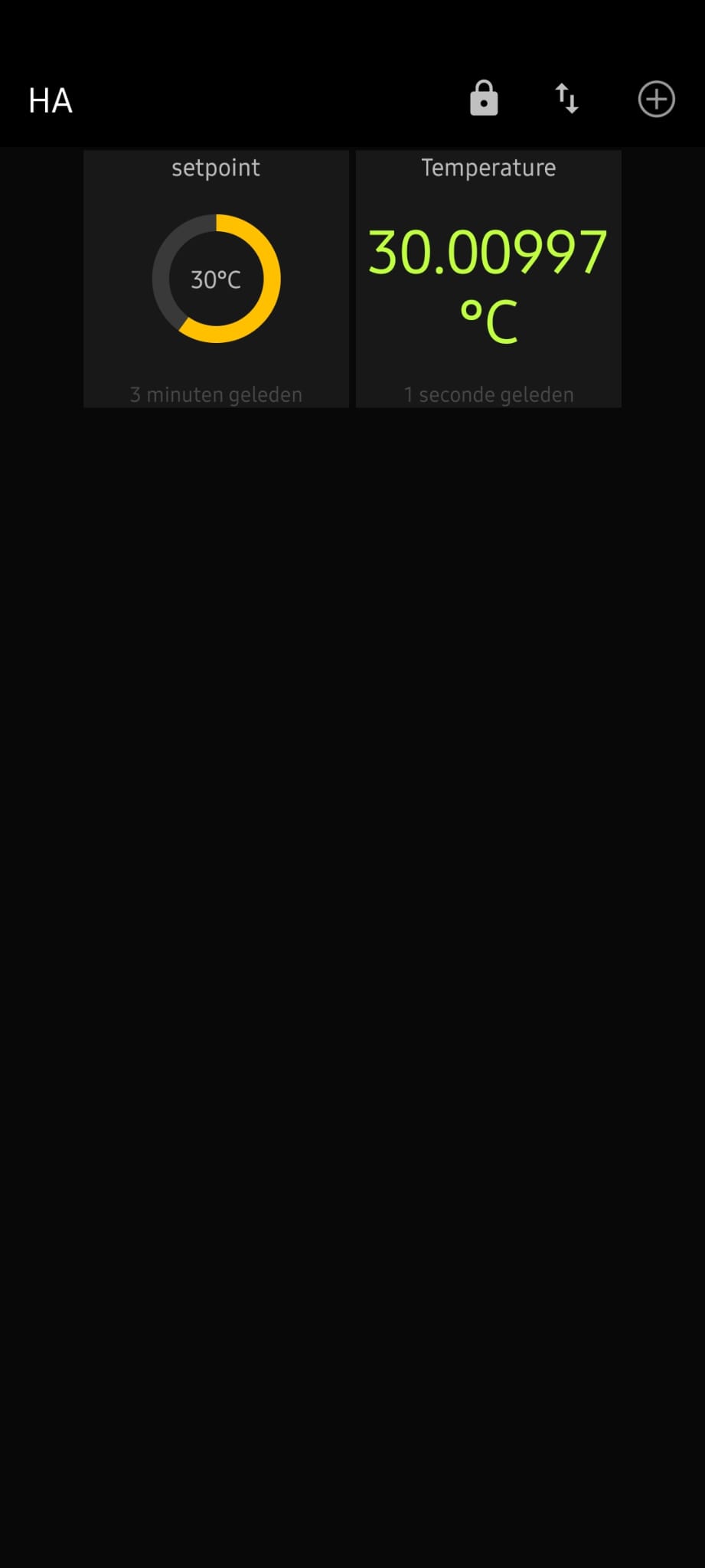
Grafana is een visualisatie software waar er dashboards kunnen aangemaakt worden die data visualiseren in bijvoorbeeld grafieken. Ik ga dit ook enkel maar gebruiken om de actuele temperatuur dat de thermostaat stuurt in een grafiek te visualiseren en om het setpoint af te beelden.

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



## MQTT Dash

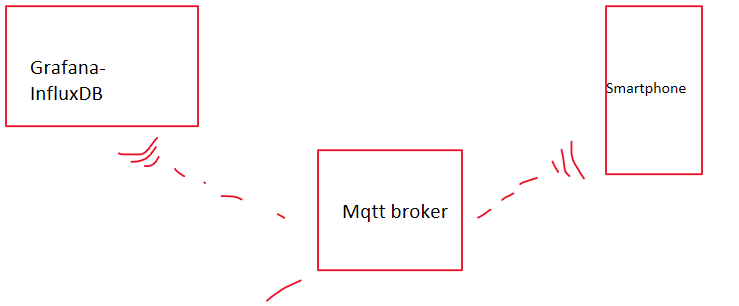
Om de gewenste temperatuur te sturen, alsook om de actuele temperatuur uit te lezen. Ik maak daarvoor gebruik van een App op mijn smartphone. Deze app noemt MQTT Dash, die is verbonden met de mosquitto broker uit Home Assistant en ik heb op beiden topics gesubscribed.

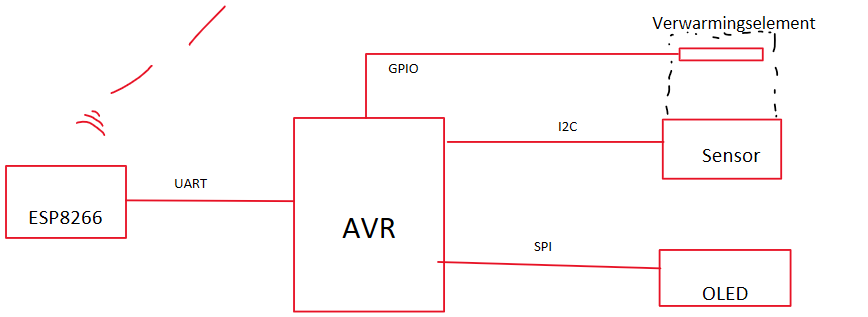


# Hardware

## 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.





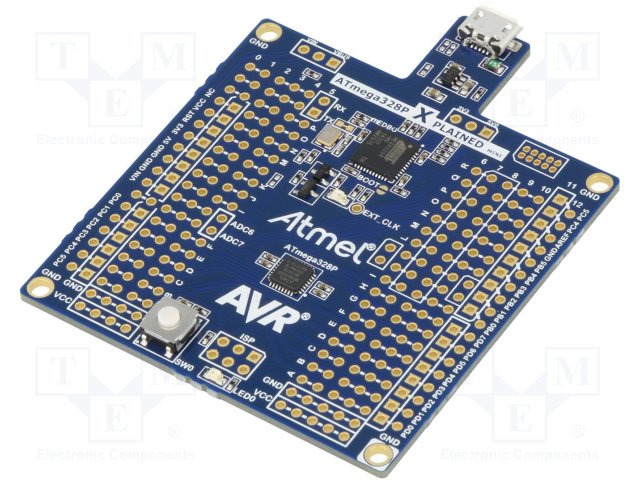
In dit project maken we gebruik van:

* AVR 🡪 microcontroller
* Wifi module 🡪 UART
* Verwarmingselement 🡪 GPIO
* Temperatuur sensor 🡪 I2C
* OLED display 🡪 SPI

## Componenten

### 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.

### 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.

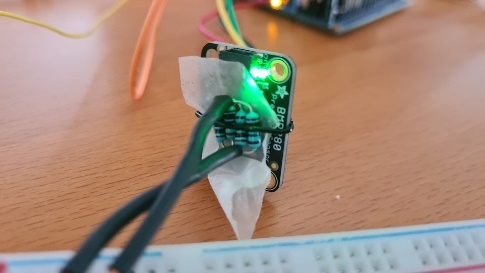
* De Vin wordt aangesloten op de VCC.
* De GND wordt aangesloten op de GND.
* De SCK wordt aangesloten op de PB5.
* De SDI wordt aangesloten op de PB4.

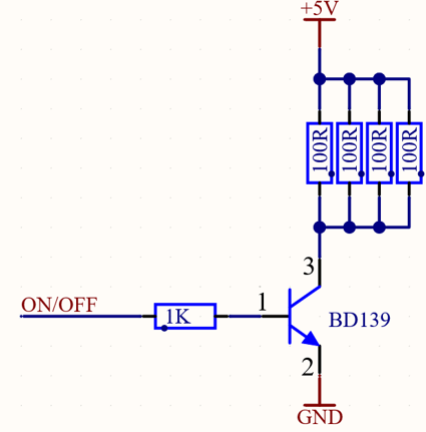
Afbeelding met tekst, elektronica, circuit

Automatisch gegenereerde beschrijving

### 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.

### 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:

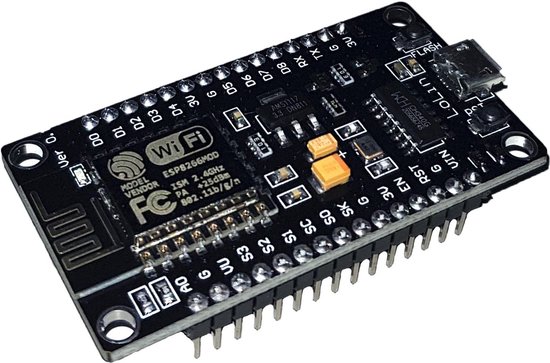
* GND is de aansluiting op de grond op onze MCU
* VCC is de aansluiting onze 5V van onze MCU
* D0 is de aansluiting van de klok op PB5 van onze MCU
* D1 is de aansluiting van de MOSI op PB3 van onze MCU
* RES is de aansluiting van de Reset op PB0 van onze MCU
* DC is de aansluiting van de Data command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

Afbeelding met tekst, elektronica

Automatisch gegenereerde beschrijving

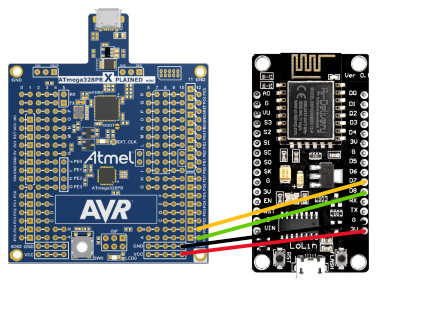
### ESP8266 (wifi module)

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



# Software

## 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**.

## 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.

### 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.

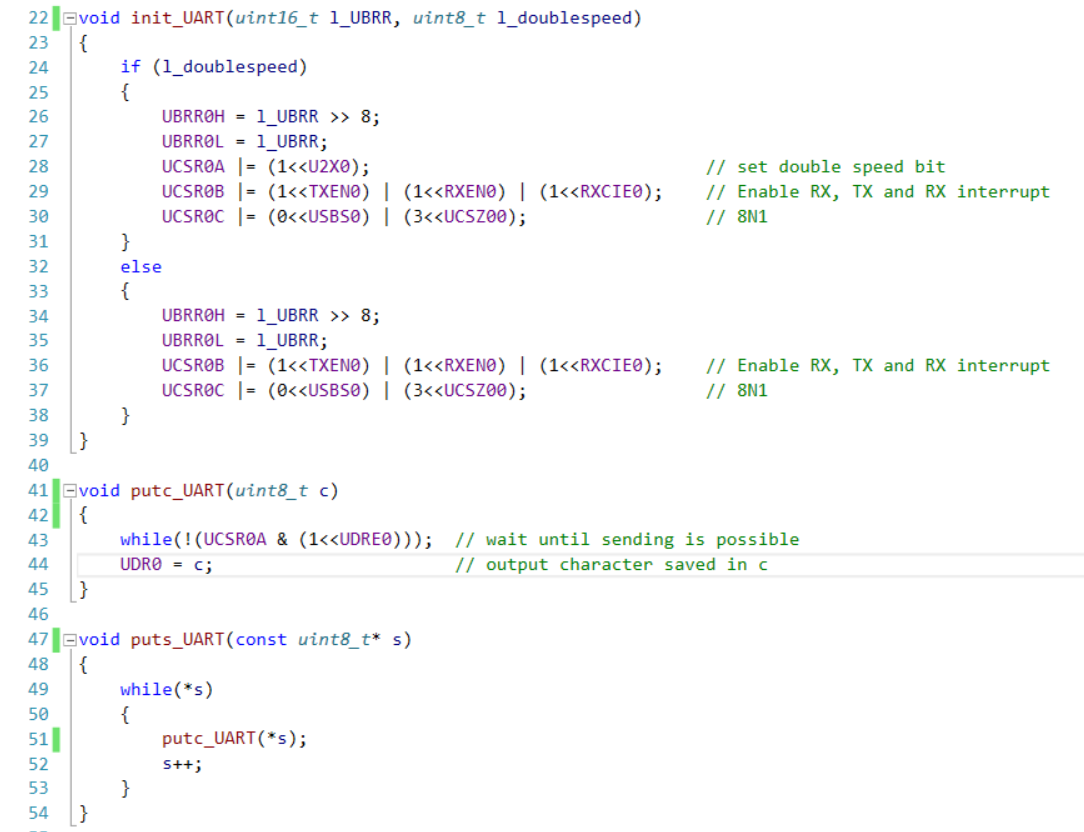
* **UBRR0H:** High byte
* **UBRR0L:** Low byte

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.



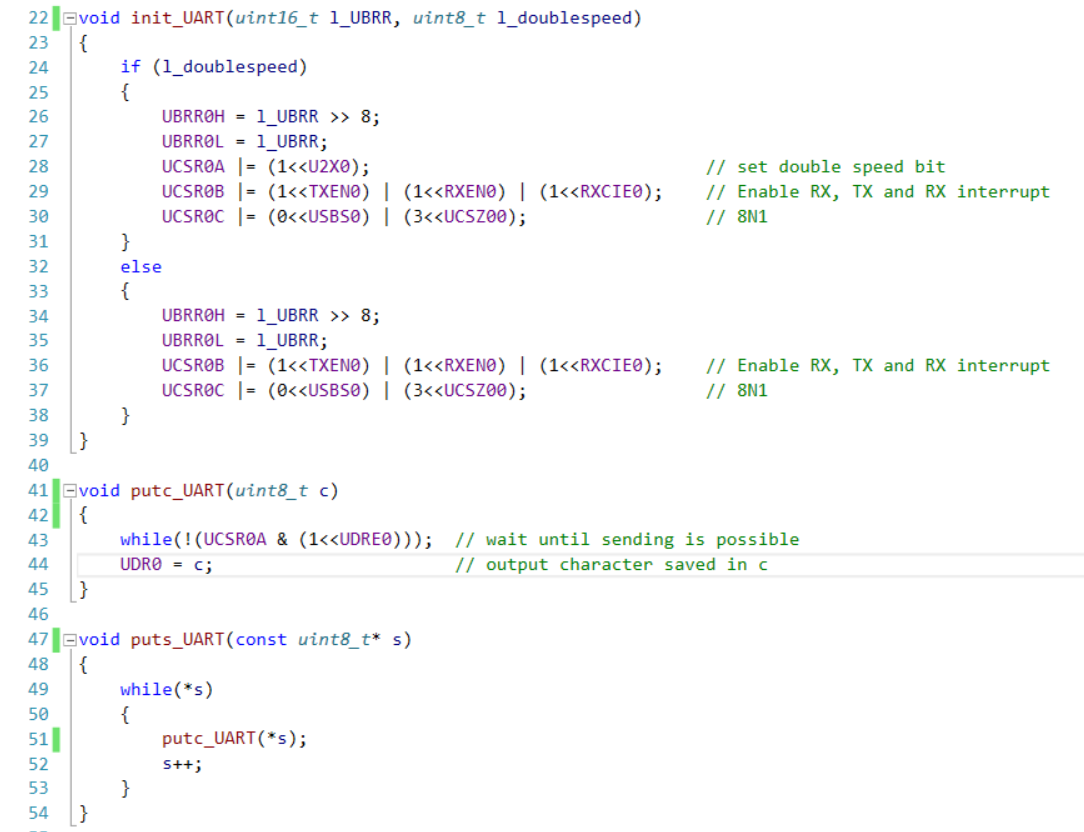
#### UART\_put..

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.



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.

### SPI.c

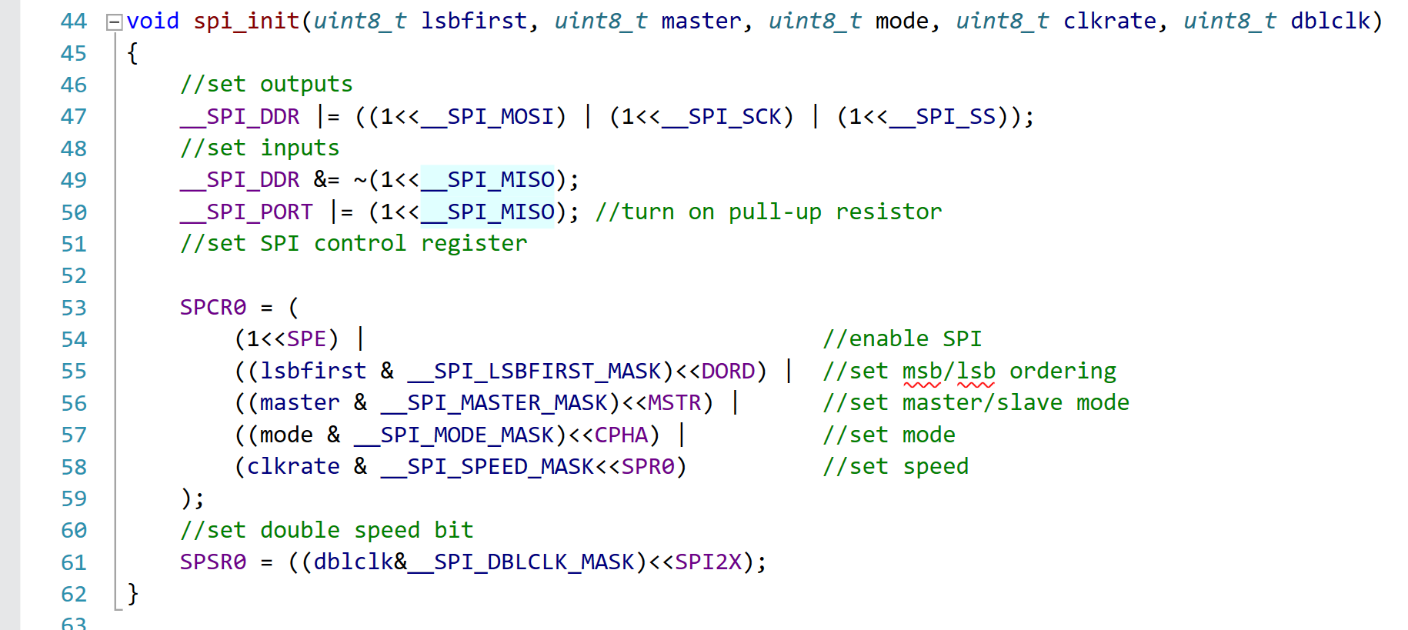
Voor de configuratie van de SPI peripheral heb ik ook gebruik gemaakt van een functie die al aangemaakt is in andere software.

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.

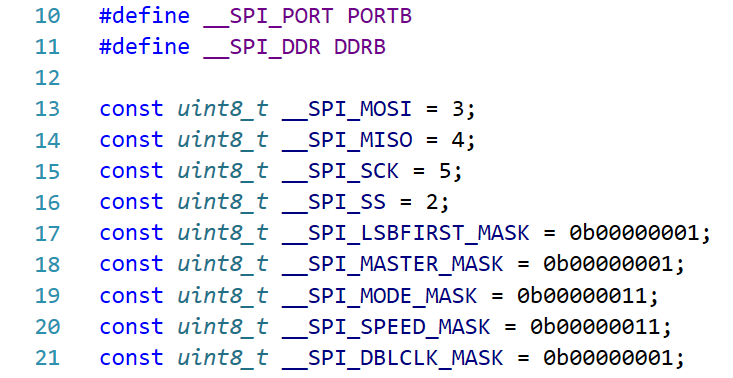
* ***uint8\_t* lsbfirst** Wordt gebruikt om te kiezen tussen MSB en LSB:  
   **MSB = 0  
   LSB = 1**
* *uint8\_t* master Wordt gebruikt om te kiezen tussen master of slave:  
   **MASTER = 1  
   SLAVE = 0**
* *uint8\_t* mode Wordt gebruikt om transfer mode in te stellen:  
   **MODE 0 = 0  
   MODE 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.



Figuur 2 spi\_init HAL

Voor deze functie zijn een aantal defines aangemaakt en variabelen.



Figuur 3 spi\_init defines variables

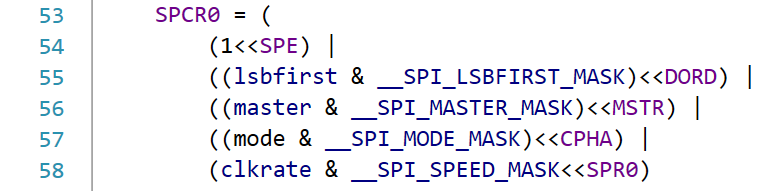
Als we de functie aanroepen gaan we eerst de IO instellen. Dat doen we door de volgende lijnen:

\_\_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.



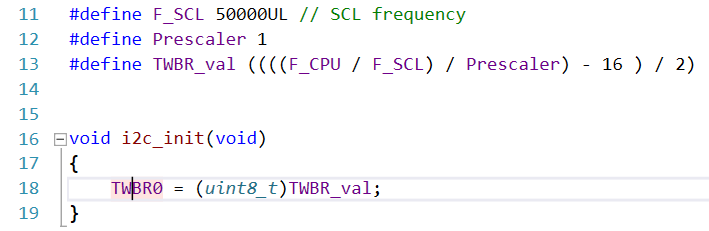
Figuur 4 spi\_init SPCR0

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.

### I2C.c

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



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)

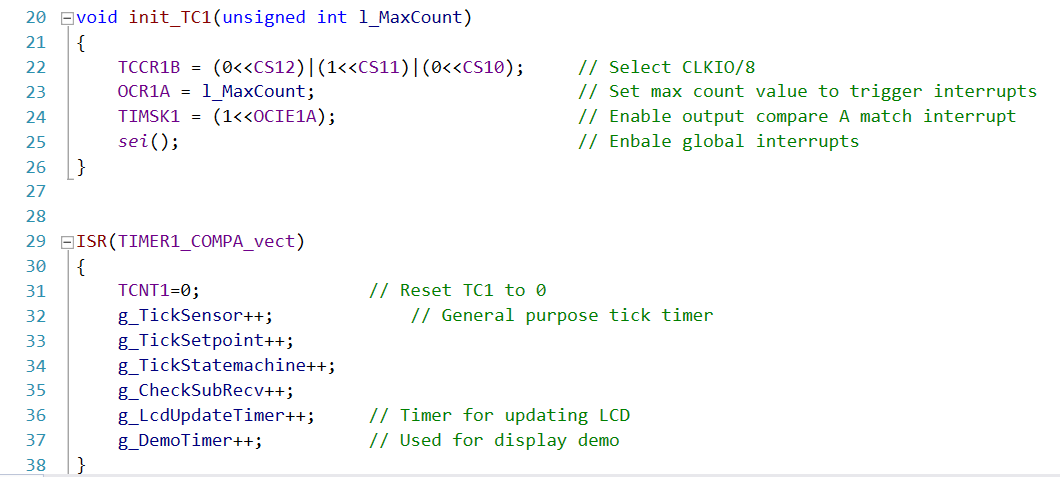
### TC1.c

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.



Figuur 5 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.

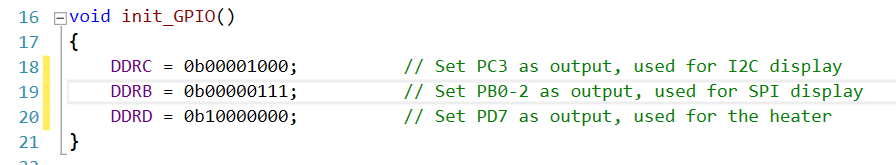
TIMSK1 = (1<<OCIE1A);   
In het Timer/Counter 1 Interrupt Mask Register gaan we het Output Compare A Match Interrupt aangezet.

*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.

### 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.

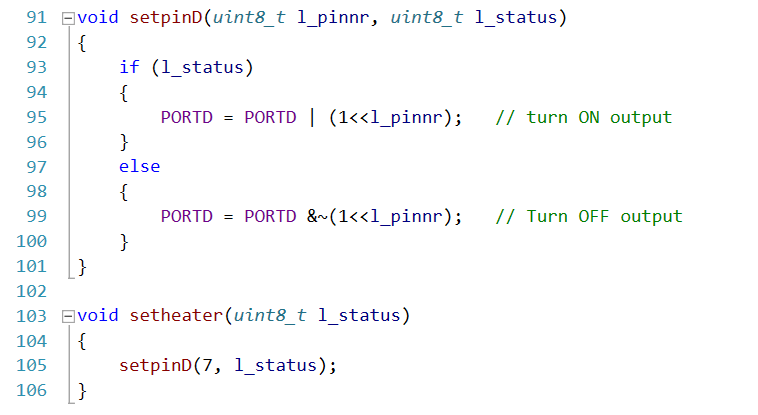


#### SetpinD

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.



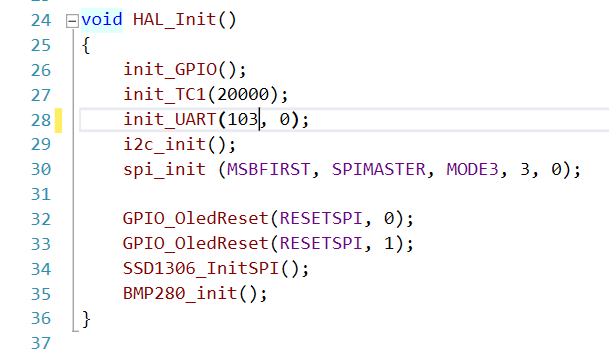
Met deze functie kunnen/ moeten we 2 waardes meegeven:

* *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.

## Middleware

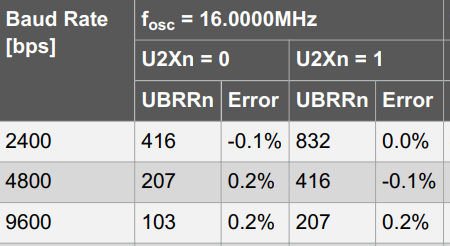
### HAL\_Init.c

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.



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.



* 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.

### 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* Yis de y-as van het display.

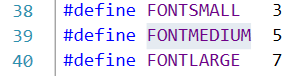
Afbeelding met tafel

Automatisch gegenereerde beschrijving

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.

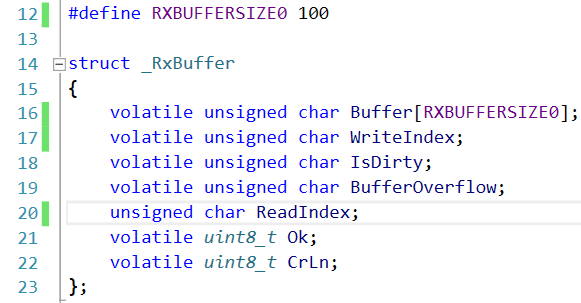
Voor de fontsize zijn er defines aangemaakt in global.h



1. void ProcessLCD(*uint8\_t* l\_Displaytype)   
   Deze functie gebruiken we enkel om het framebuffer naar het display te kopiëren.

### Buffer.c

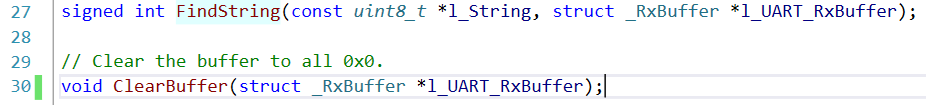
In de uiteindelijke software gaat er veel data worden ontvangen via de UART. Omdat de UART enkel maar byte per byte kan ontvangen moeten we onze ontvangen bytes ergens gaan stockeren zodat we ze naderhand kunnen raadplegen. Hier gaan we een buffer voor gebruiken. De buffer is aangemaakt in een struct.



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.

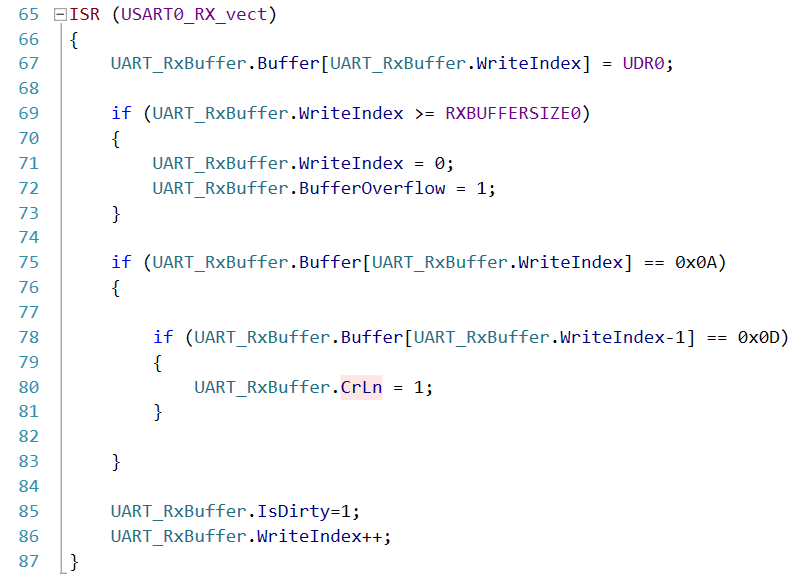
Voor de buffer zijn ook nog 2 functionele functies.



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

#### UART Interrupt

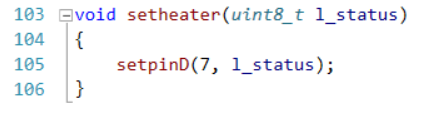
Van het moment dat er via de UART een byte binnenkomt wordt de interrupt getriggerd.



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.

### Setheater



Deze functie gaat de functie [setpinD](#_SetpinD) uitvoeren en pin 7 aan of uit zetten, naargelang wat we deze functie meegeven natuurlijk.

### App\_BMP280.c

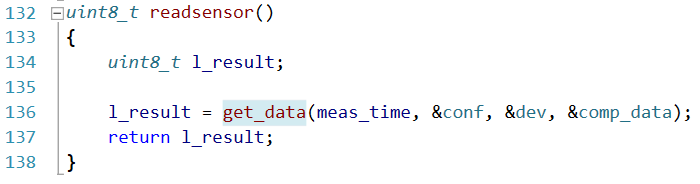
Voor de temperatuur sensor hebben wij een driver gekregen. We moeten hier geen code voor schrijven. Ik ga hier dan ook niet dieper op in. Wat wel belangrijk is om te weten, is dat in comp\_data.temperature de uitgelezen temperatuur staat. En dat de functie get\_data ons verteld dat er communicatie is 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.

#### 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.

#### 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.



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.



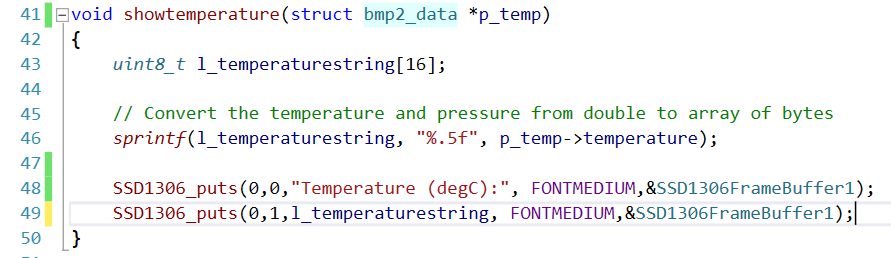
#### Showsetpoint

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](#_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](#_SSD1306.c) worden teruggevonden.

#### Showtemperature

De functie Showtemperature toont de actuele temperatuur op het display.



De functie verwacht hier een pointer naar de struct bmp2\_data .  
In deze functie doen we hetzelfde als in [Showsetpoint](#_Showsetpoint) alleen converteren we de actuele temperatuur naar een array en tonen we de actuele temperatuur op het display.

#### Showsensorerror

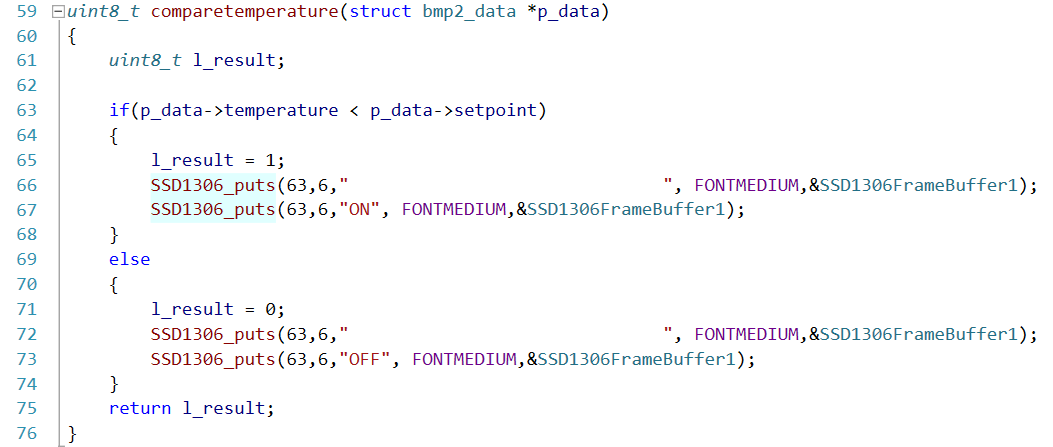
De functie Showsensorerror toont de tekst “Sensor error” op het display.



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.

#### Comparetemperature

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



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.

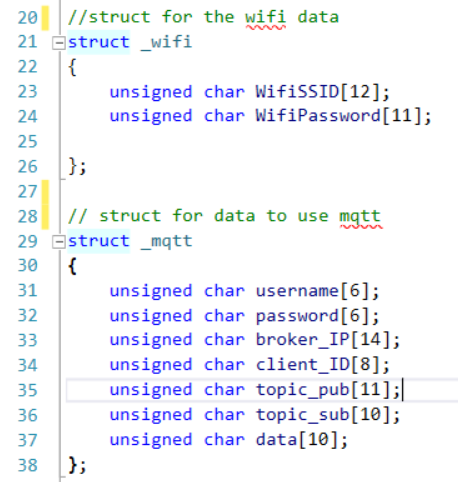
### AT\_commands\_ESP8266.c

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.

#### 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.



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

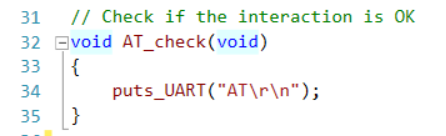
* WifiSSID Voor het wifi netwerk waarmee we willen connecteren.
* WifiPassword Voor het paswoord van het wifi netwerk.

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.

#### 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](#_UART_put..).

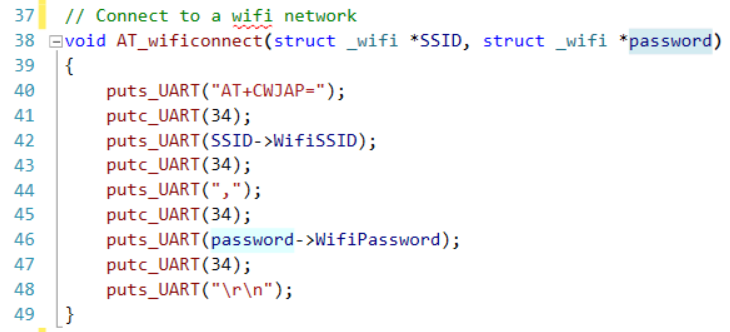


#### AT\_wificonnect

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](#_UART_put..). Dit wordt wel in enkele stappen gedaan. De waarde 34 die wordt gestuurd is de decimale waarde voor **“**.



#### 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:

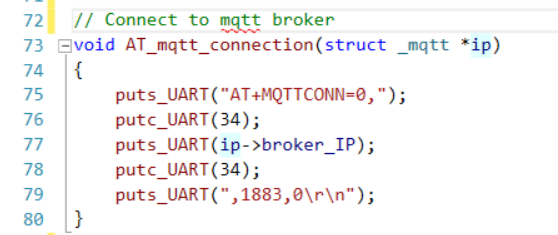
**AT+MQTTUSERCFG=0,1," \*id "," \*user "," \*password ",0,0,""\r\n**

****

#### 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**

****

#### 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:

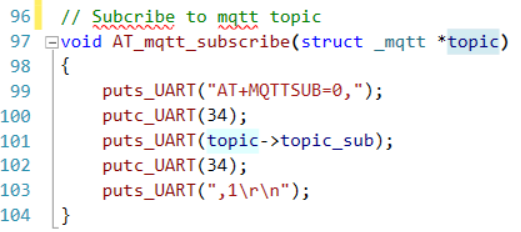
**AT+MQTTPUB=0," \*topic "," \* data ",1,0\r\n**



#### 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**



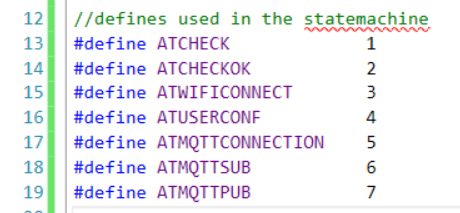
### Statemachine.c

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.

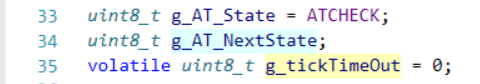
Een statemachine is een algoritme die een eindige reeks toestanden definieert: één toestand wordt gedefinieerd als de begintoestand. Wanneer we de statemachine beginnen uit te voeren, komt deze automatisch in deze toestand. Elke status kan acties definiëren die plaatsvinden wanneer een machine die status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

#### 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.



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.



#### 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.

#### 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.

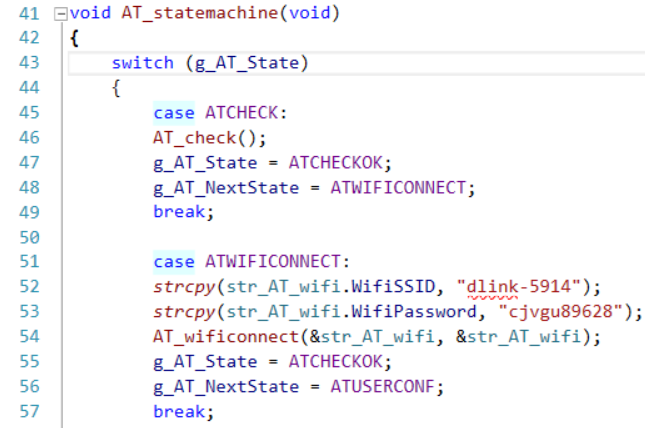
Stel we hebben dit:

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.

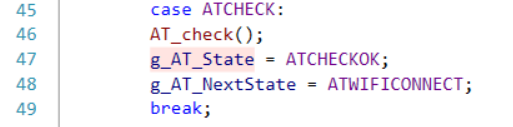
In de statemachine ziet het er zo uit:



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](#_Case_ATCHECK) worden uitgevoerd. Dat komt omdat we dit hebben toegekend aan de variabele g\_AT\_State = ATCHECK;

#### Case ATCHECK

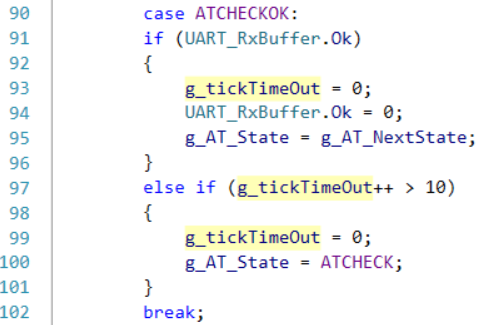
Deze staat gaat als eerste worden uitgevoerd. De bedoeling van deze state is dat de communicatie wordt gecontroleerd.



1. [AT\_check()](#_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.

#### Case ATCHECKOK

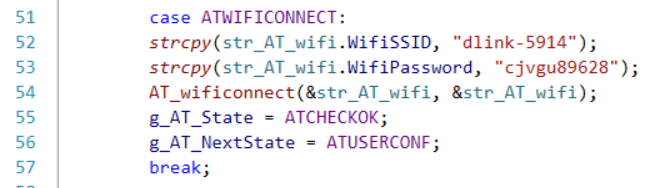
In het begin van de titel statemachine heb ik al uitgelegd dat de wifi module OK terug stuurt als bevestiging dat de commando geslaagd is. Dit gaan we in deze state controleren. Dit doen we altijd voor dat we naar een volgende commando state gaan. Elke keer dat de communicatie is mislukt gaan we terug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



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](#_Case_ATCHECK) als huidige state definiëren. Dat wil zeggen dat er iets mis is en dat we terug opnieuw moeten beginnen.

#### Case ATWIFICONNECT

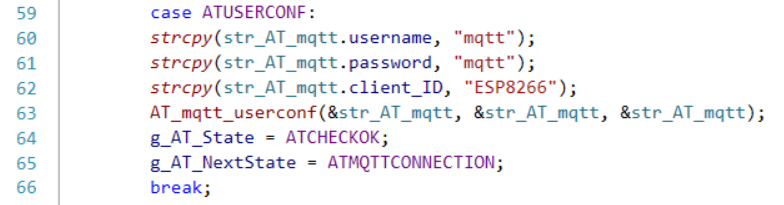
Als Case [ATCHECK](#_Case_ATCHECK) [gecontroleerd](#_Case_ATCHECKOK) is gaan we deze case uitvoeren. Deze case gaat de verbinding met het wifi netwerk tot stand brengen.



In deze state gaan we eerst de SSID en paswoord kopiëren naar de arrays door middel van [strcpy](#_Strcpy) . Daarna gaan we de functie [AT\_wificonnect](#_AT_wificonnect) uitvoeren. Dan gaan we weer terug de [controle](#_Case_ATCHECKOK) laten uitvoeren, en als volgende state de user configuratie laten uitvoeren.

#### Case ATUSERCONF

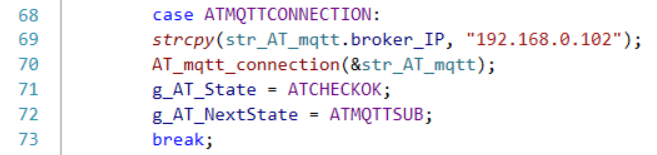
Als Case ATWIFICONNECT [gecontroleerd](#_Case_ATCHECKOK) is gaan we deze case uitvoeren. Deze case gaat de configuratie van de user en id voor de mqtt broker tot stand brengen.



Hier wordt ook weer eerst alle strings gekopieerd door middel van [strcpy](#_Strcpy). Daarna gaan we de functie [AT\_mqtt\_userconf](#_AT_mqtt_userconf) uitvoeren. Dan gaan we weer terug de [controle](#_Case_ATCHECKOK) laten uitvoeren en als volgende state de mqtt broker connectie laten uitvoeren.

#### Case ATMQTTCONNECTION

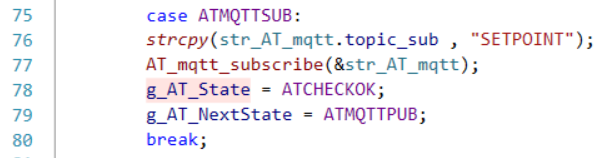
Als Case ATUSERCONF [gecontroleerd](#_Case_ATCHECKOK) is gaan we deze case uitvoeren. Deze case gaat de connectie maken met de mqtt broker.



De string van het IP adres wordt hier eerst gekopieerd, daarna wordt [AT\_mqtt\_connection](#_AT_mqtt_connection) uitgevoerd. Daarna wordt de [controle state](#_Case_ATCHECKOK) en de volgende state gedefinieerd.

#### Case ATMQTTSUB

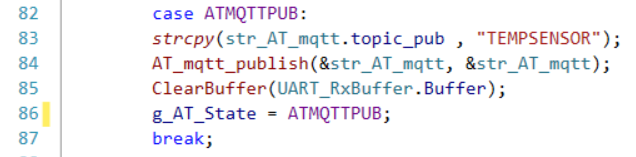
Als Case ATMQTTCONNECTION [gecontroleerd](#_Case_ATCHECKOK) is gaan we deze case uitvoeren. Deze case gaat subcriben op het topic “setpoint”.



De string van het topic wordt eerst gekopieerd. Daarna wordt [AT\_mqtt\_subcribe](#_AT_mqtt_subcribe) uitgevoerd. En dan volgt de [controle state](#_Case_ATCHECKOK) en de volgende state die gedefinieerd worden.

#### Case ATMQTTPUB

Als Case ATMQTTCONNECTION [gecontroleerd](#_Case_ATCHECKOK) is gaan we deze case uitvoeren. Deze case gaat subcriben op het topic “setpoint”.



De string van het topic wordt eerst gekopieerd. Nadien wordt [AT\_mqtt\_publish](#_AT_mqtt_publish) uitgevoerd. Daarna wordt de functie ClearBuffer uitgevoerd om ervoor te zorgen dat de buffer niet overvol gaat zitten.

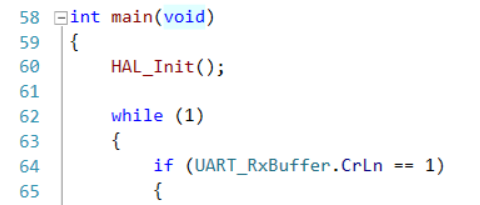
## Application layer

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.

### 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()](#_HAL_Init.c) 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).

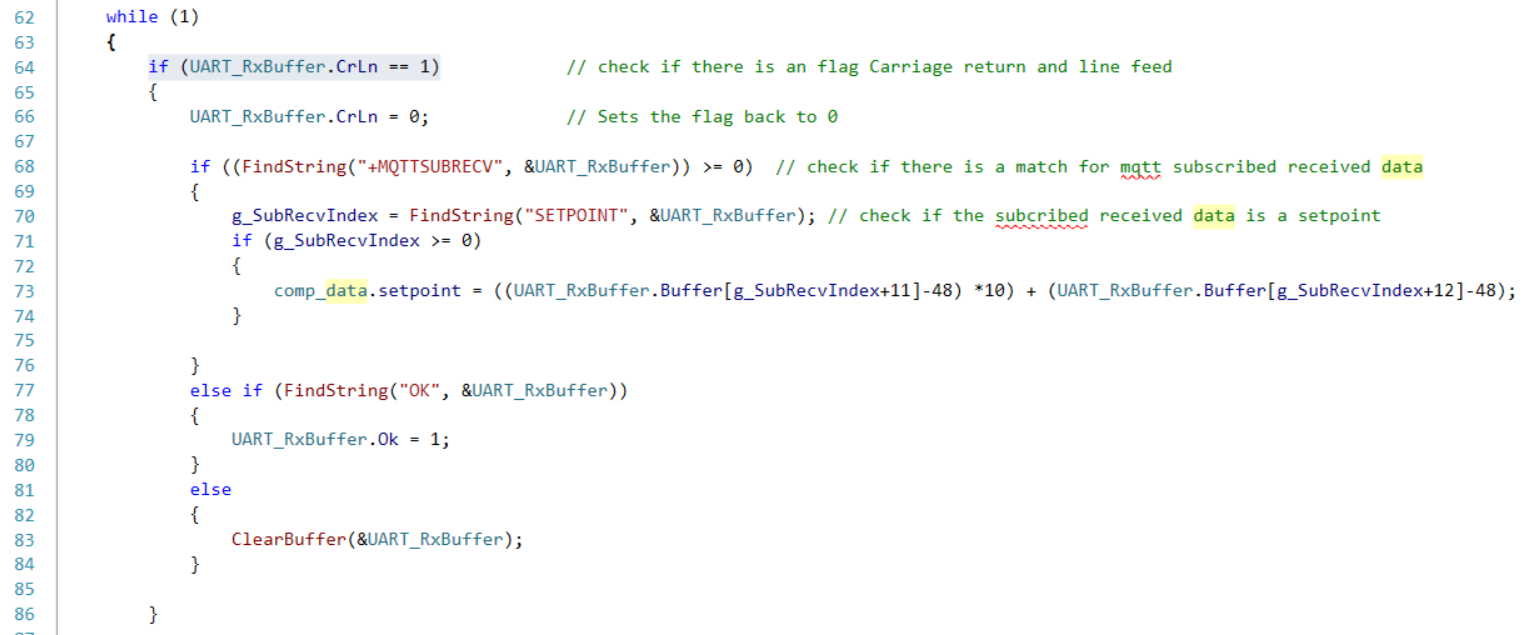


#### if (UART\_RxBuffer.CrLn == 1)

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.

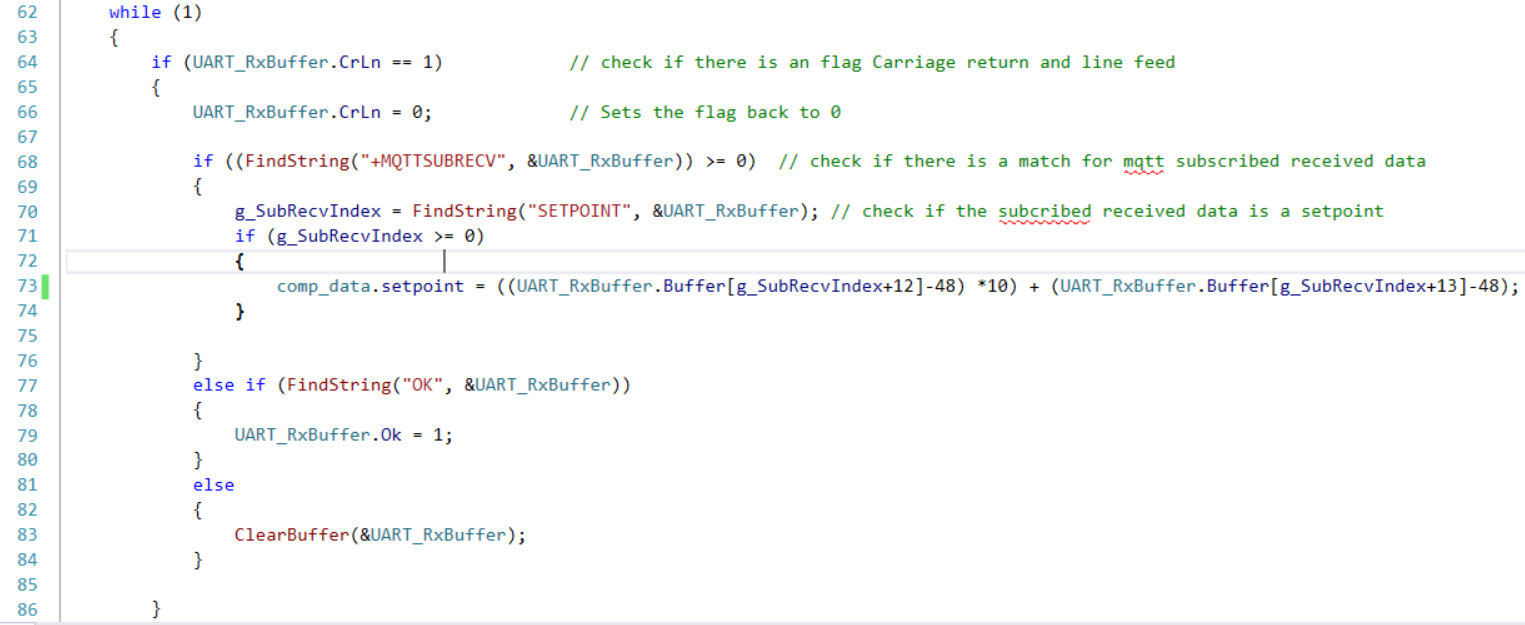


In de interrupt van de [buffer](#_UART_Interrupt) is er een flag aangemaakt die op 1 wordt gezet als er een Carriage return en een Line feed is binnengekomen in de buffer. Het eerste wat we doen is die flag terug op 0 zetten.



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.

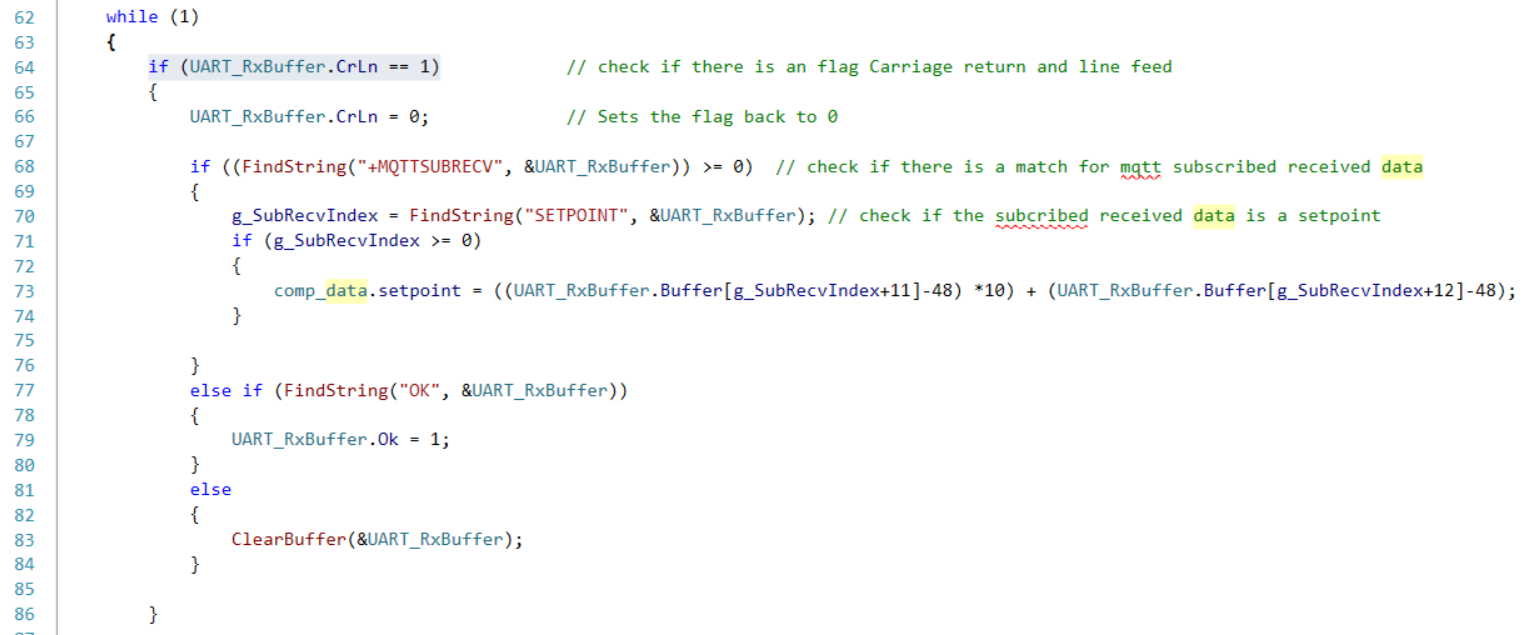


Daarna gaan we in de variabele setpoint van de struct comp\_data de output van de volgende berekening zetten.

((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.



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](#_Case_ATCHECKOK). Anders gaan we de buffer leeg maken.

#### 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()](#_Showsensorerror) uitvoeren.



Anders gaan we eerst op regel 94 [sprintf](#_Sprintf) uitvoeren zodat de actuele temperatuur wordt geconverteerd naar de array die we gebruiken om via de functie [AT\_mqtt\_publish](#_AT_mqtt_publish) de data naar de broker te sturen. Daarna gaan we de functies [showtemperature](#_Showtemperature) en [showsetpoint](#_Showsetpoint) uitvoeren.

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

#### if (g\_TickStatemachine > 500)

Hier gaan we om de 500mS het algoritme [AT\_statemachine](#_Statemachine.c) oproepen.