



Voorwoord

Ik ben Olivier Van den Eede.

olivier.vandeneede@student.thomasmore.be

Voor mijn opleiding Electronica-Ict fase1 aan Thomas More De Nayer heb ik een project gemaakt, namelijk de kamerthermostaat.

Ik vond dat het ofwel steeds te warm of steeds te koud was op mijn kamer, daarom ben ik begonnen denken hoe dit opgelost moet worden. Daarom ben ik begonnen om mijn eigen thermostaat te maken om de temperatuur automatisch te kunnen regelen.

Inhoudstafel

Voorwoord	2
Inhoudstafel	3
1 Introductie	5
1.1 Inleiding	5
1.2 Samenvatting	5
1.3 Probleem en doelstelling	5
1.4 Oplossingsstrategie	5
2 Bespreking blokschema	6
2.1 Blokschema Hoofdmodule	6
2.1.1 Microcontroller	6
2.1.1 Real time clock	6
2.1.3 Lcd-Controller	7
2.1.4 Port-Expander	7
2.1.5 Rotary encoder	8
2.1.6 Rf-controller	8
2.2 Blokschema Temperatuursensor	9
2.2.1 Microcontroller	9
2.2.2 Rf-controller	9
2.2.3 Temperatuursensor	9
3 Samenhang	10
3.1 Volledig schema	10
3.1.1 Thermostaat	10
3.1.2 Temperatuursensor en relais-module	11
3.2 Layout van de print	12
3.2.1 Thermostaat	12
3.2.2 Temperatuursensor en relais-module	13
3.3 Foto's	14
4 Broncode	16
4.1 Thermostaat	16
4.1.1 Uitlezen Rtc (rtc.h)	16
4.1.2 Inlezen Rotary encoder (rotary.h)	18
4.1.3 Het lcd (nokia_5110.h)	19
4.1.4 i2c-port expander met knoppen	21
4.1.5 Nrf24L01 (nRF24L01.h)	22
4.2 Relais-module	25
4.2.1 Nrf24L01	25
4.2.2 main	26
4.3 Temperatuursensor	28
4.3.1 Nrf24l01	28
4.3.2 HTU21D-temp sensor	29
4.4 Sleep-mode	31

5	Conclusie	32
5.1	Reflectie	32
5.2	Kostprijs	33
Bijlage 1:	Broncode Hoofdmodule	34
Bijlage 2:	Broncode Temperatuursensor	64
Bijlage 3:	Broncode Relais-module	73
Bijlage 4:	Datasheet features	76
Bijlage 5:	Referenties	85
Bijlage 6:	Projectvoorstel	86
Bijlage 7:	Plan van aanpak	88
Bijlage 8:	Logboek	91

1 Introductie

1.1 Inleiding

Het project de kamerthermostaat is ontworpen en gemaakt om de temperatuur in een kamer te meten met een draadloze temperatuursensor en een hoofdmodule die nagaat afhankelijk van de ingestelde temperatuur of de verwarming aan of uit moet. Dit gebeurd met 3 modules. De 3 modules communiceren met elkaar dmv. rf-modules.

1.2 Samenvatting

De eerste module, de temperatuursensor, bevat een µc, een rf-module en een temperatuursensor. Deze meet de temperatuur en stuurt deze door naar de hoofdmodule.

De 2de module, de thermostaat zelf, bevat een µc, een rf-module, een lcd scherm, een rotary encoder en een aantal knoppen. Op het lcd van de thermostaat wordt de huidige temperatuur weergegeven, en kunnen aan aantal dingen ingesteld worden met de knoppen en de rotary encoder.

De 3de module, de relais-module, bevat een µc en een rf-module. De module stuurt een relais aan die op zijn beurt het ventiel aanstuurt dat de verwarming regelt.

1.3 Probleem en doelstelling

Bij het initiële idee van de temperatuursensor was er geen spraken van een aparte temperatuursensor. Maar omdat de thermostaat op mijn bureau zou komen te staan zou de temperatuursensor te dicht tegen de verwarming staan.

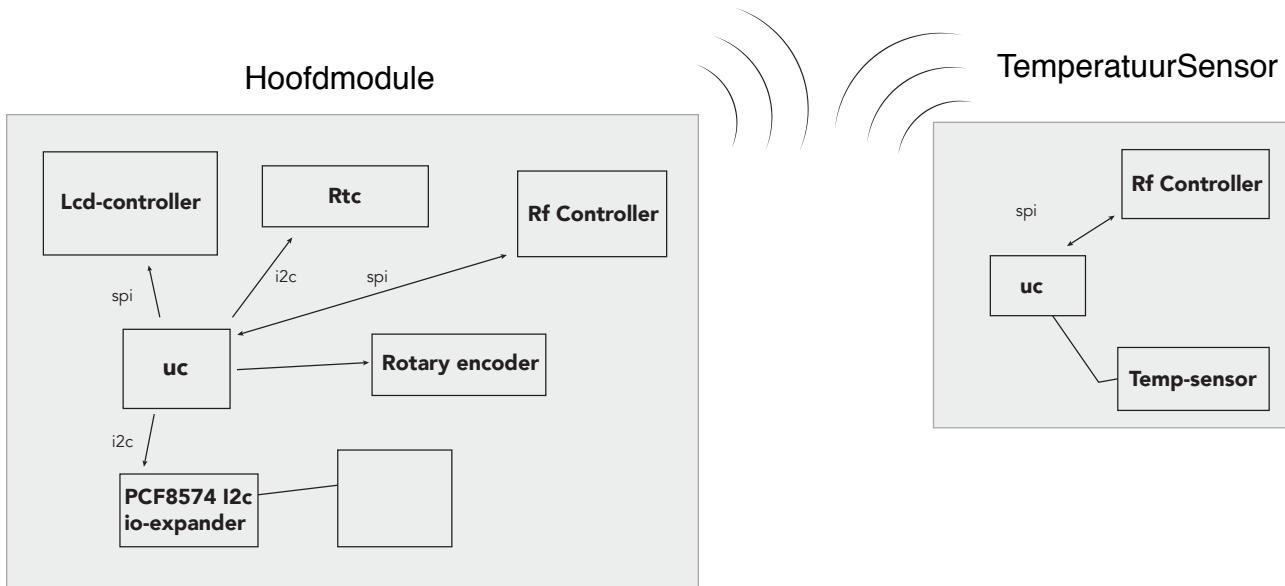
De afgezonderde modules zullen batterij gevoed worden dus moet er ook aan power-management gedacht worden.

1.4 Oplossingsstrategie

Ik heb dus gekozen om de temperatuursensor af te zonderen en draadloos te doen omdat er zo ook een uitbreiding mogelijk is om nog extra modules te koppelen achteraf.

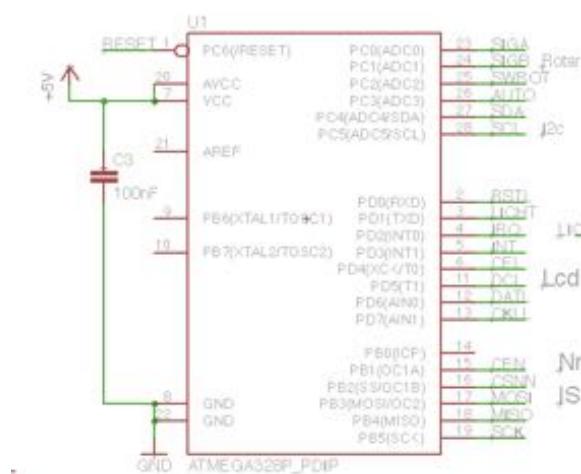
Voor het power management zal er gebruik gemaakt worden van 1 van de slaap functies van de µc om het batterijverbruik te verminderen en de levensduur te verlengen.

2 Bespreking blokschema



2.1 Blokschema Hoofdmodule

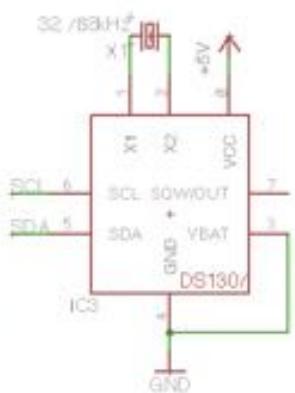
2.1.1 Microcontroller



De hoofdmodule wordt gestuurd door de microcontroller. Voor de microcontroller heb ik gekozen voor een Atmega 328p van atmel. De reden hiervoor is dat ik wou leren om ook met een andere controller te kunnen werken dan de controller die we in de lessen gezien hebben. De controller beschikt over een spi- en I₂C-bus die ik beide gebruik heb om tot een werkend geheel te komen. Mijn controller werkt op een interne klok van 8Mhz, dit kan omdat er geen nauwkeurig kristal nodig is, omdat er geen tijdkritische handelingen moeten gebeuren. De controller werkt hier op 5v.

2.1.1 Real time clock

In mij project wordt er gebruikgemaakt van een Real time clock. Deze wordt gebruikt om het uur bij te houden zodat de microcontroller hier geen tijd aan moet verliezen. Het bijhouden van het uur is belangrijk omdat de thermostaat ingesteld kan worden om enkel tussen bepaalde uren te werken, de controller kan dus op elk moment het uur uitlezen, zonder dat er een nauwkeurig kristal ,een timer en interrupt nodig zijn. Het Rtc heeft een eigen kristal van 32.768kHz en wordt gevoed met 5v.



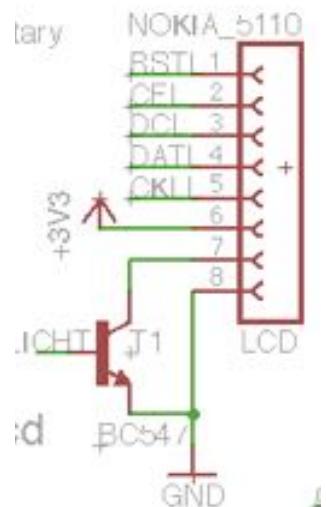
2.1.3 Lcd-Controller

In mijn Project maak ik gebruik van een klein grafisch lcd, een Nokia 5110. Bij dit lcd zit een PCD8544 lcd-controller. Dit zit samen op een breakout bord, waaruit 8 pinnen komen.

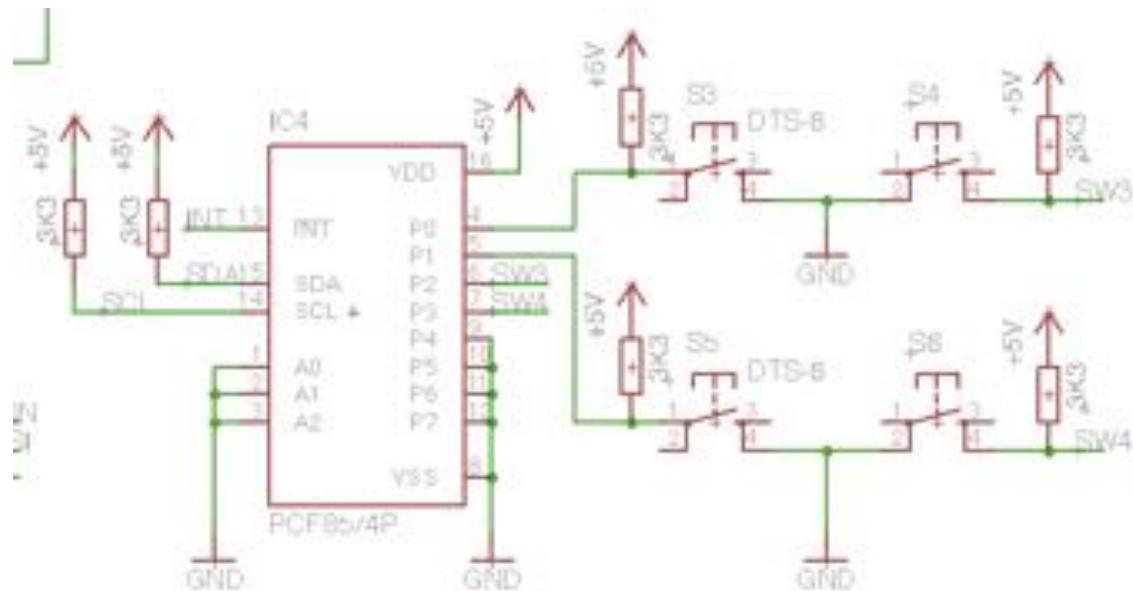
Deze pinnen zijn rechtstreeks aangesloten op de µc.

Het lcd-scherm werkt op 3,3v voedingsspanning, maar kan 5v signalen hebben op de spi aansluitingen. Om deze reden is er een Lm1117-regelaar opgenomen, zodat de voeding juist is.

Het lcd heeft een blauwe achtergrondverlichting, deze kan worden aangestuurd dmv. een BC547-transistor die aangestuurd wordt vanuit de µc. De aansluiting op het breakout bord voor de achtergrondverlichting moet aan de ground geschakeld worden.



2.1.4 Port-Expander



Er wordt gebruik gemaakt van een Philips - PCF8574 i2c port expander.

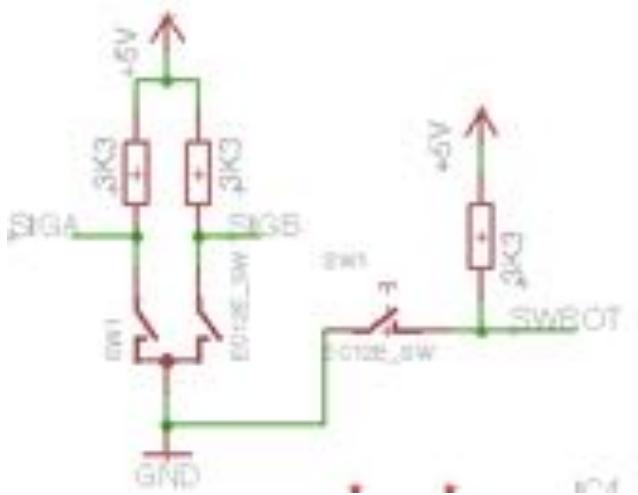
De expander wordt gebruikt omdat er niet genoeg aansluitingen zijn op de µc om alle componenten aan te sluiten, daarom heb ik er voor gekozen om de 4 drukknoppen in te lezen dmv. een port expander. Deze expander maakt gebruik van de i2c-bus die al nodig was voor het Rtc aan te sturen, dus zijn er geen extra pinnen gebruikt om 4 drukknoppen aan te sluiten.

De port expander werkt op 5v en maakt gebruik van de i2c-bus die ook op 5v werkt met de pull-up weerstanden. Elke knop apart heeft zijn eigen pull-up weerstand, en zit op een andere ingang op de expander.

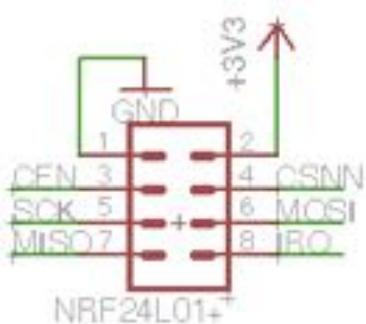
2.1.5 Rotary encoder

Ik heb voor de rotary-encoder gekozen omdat het een makkelijke manier is om op een scherm een temperatuur of tijd aan te passen. De encoder bezit ook een drukknop, wat ook weer extra mogelijkheden geeft.

De signaaluitgangen van de rotary, worden mbv. pull-up weerstanden verbonden met de pc die zo met een simpel programma kan nagaan in welke richting er wordt gedraaid.



2.1.6 Rf-controller



De gebruikte rf-controller is een nrf24l01+ op een breakout bordje. Op dit bordje komen er 8 pinnen naar buiten. Deze pinnen zijn de spi- en voedings-pinnen. De nrf werkt met een voedingsspanning van 3,3v. Dit is echter geen probleem omdat de spi pinnen weer wel 5v kunnen verdragen, en er al een 3,3v spanningsregelaar in het systeem is opgenomen voor het lcd.

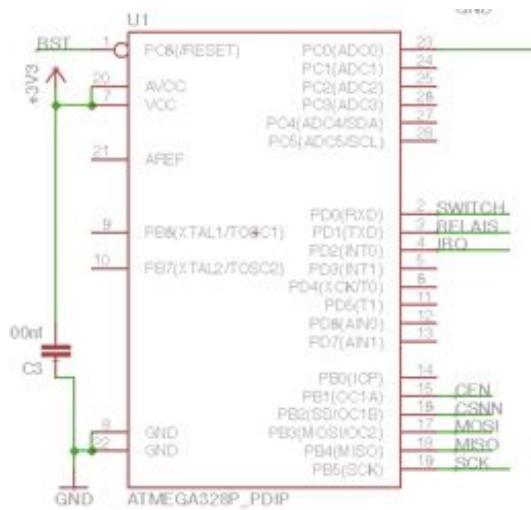
De antenne van de rf-module zit mee ingebouwd op het breakout bord.



2.2 Blokschema Temperatuursensor

2.2.1 Microcontroller

De gebruikte microcontroller in deze module is eveneens een Atmega 328p van atmel.
De reden voor het gebruik van deze controller is omdat deze over een i2c en een spi-bus beschikt. Deze bussen worden gebruikt om de Rf-module en de temperatuursensor aan te sturen. De μ c werkt in deze schakeling op 3,3v omdat de temp-sensor en de Rf-module dit nodig hebben. Er zijn in de schakeling dan ook geen andere componenten die wel 5v nodig hebben.



2.2.2 Rf-controller

De gebruikte rf-controller is een nrf24l01+ op een breakout bordje. De module wordt gebruikt om de temperatuur door te sturen naar de hoofdmodule.

2.2.3 Temperatuursensor



De gebruikte temperatuursensor is een HTU21D-i2c temperatuursensor op een breakout bord.

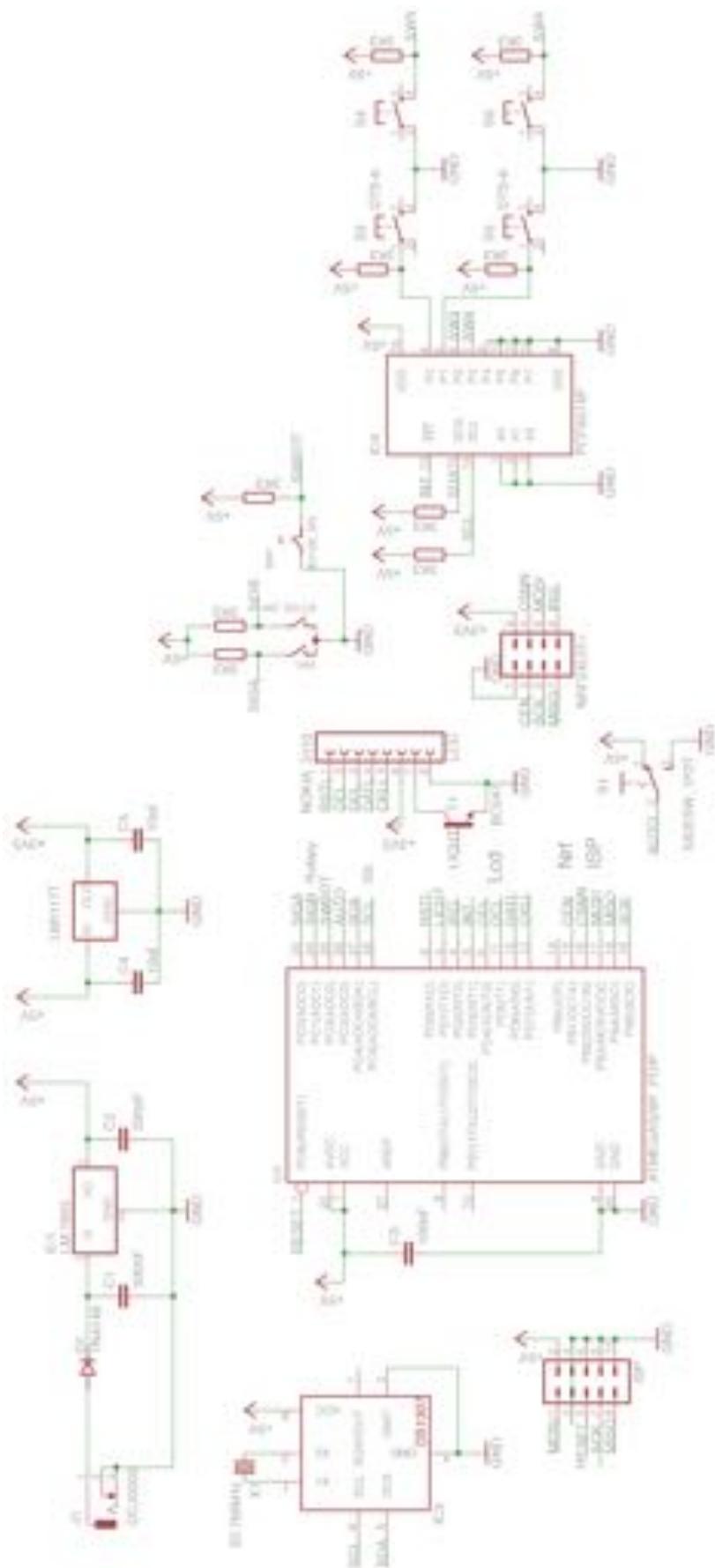
De sensor werkt op een spanning van 3,3v en wordt gestuurd via de i2c-bus van de μ c.

De sensor levert een 12-bit resultaat dat door de μ c omgezet wordt naar een temperatuur in °C.

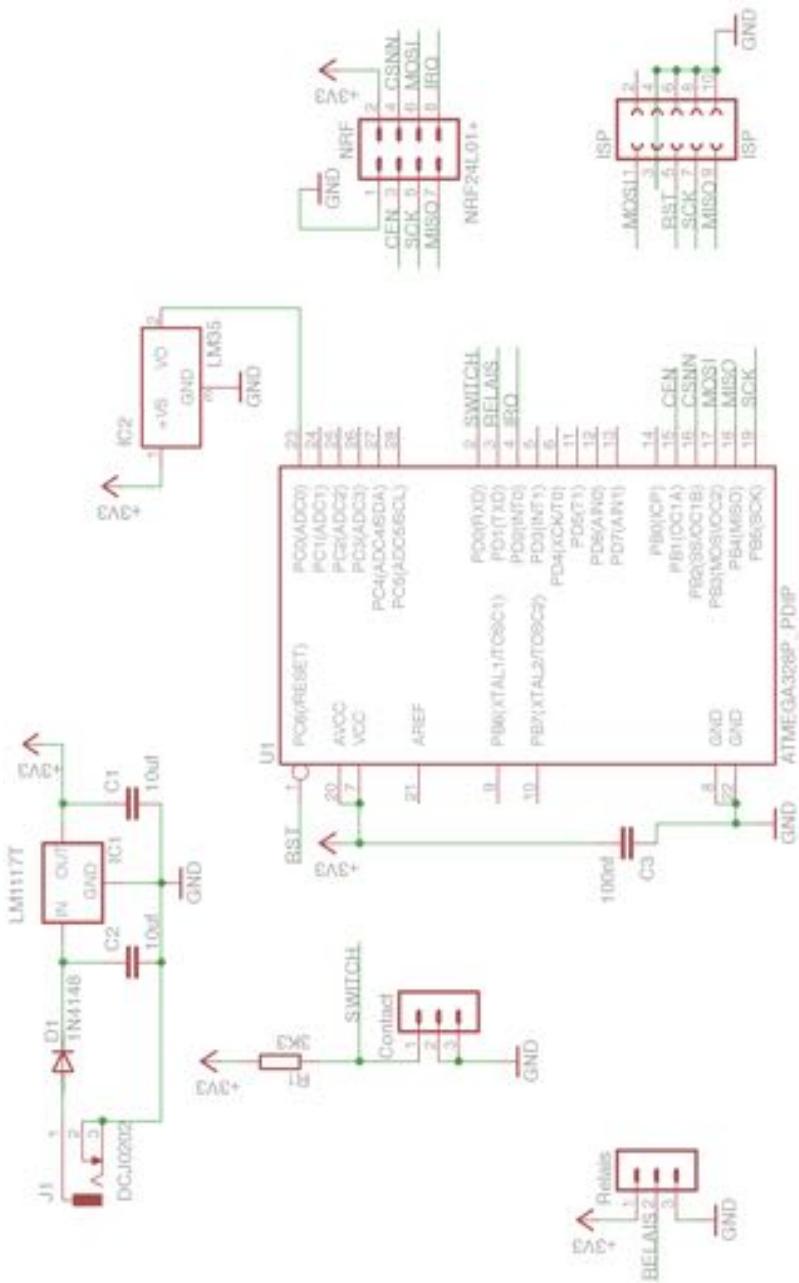
3 Samenhang

3.1 Volledig schema

3.1.1 Thermostaat



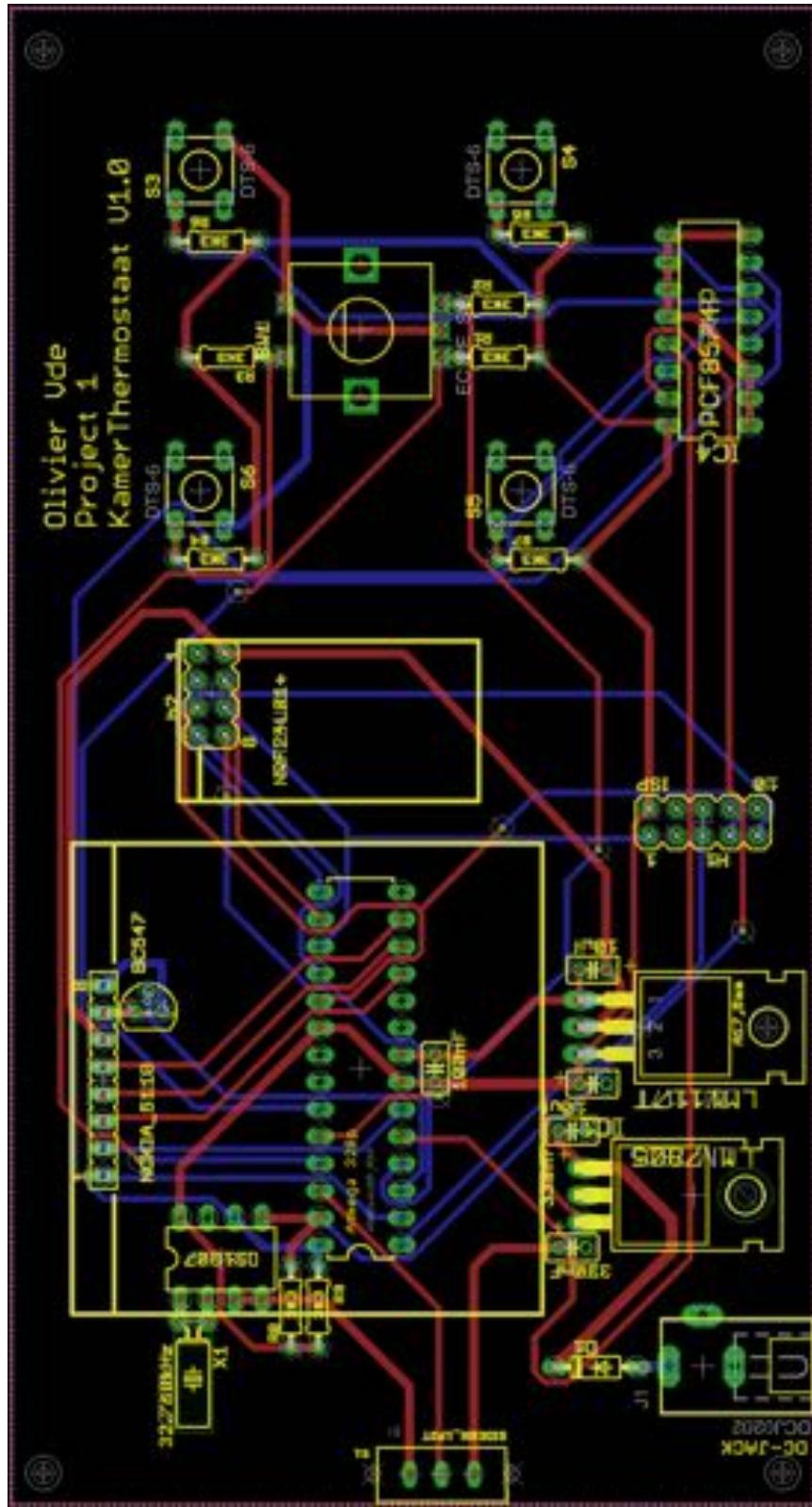
3.1.2 Temperatuursensor en relais-module



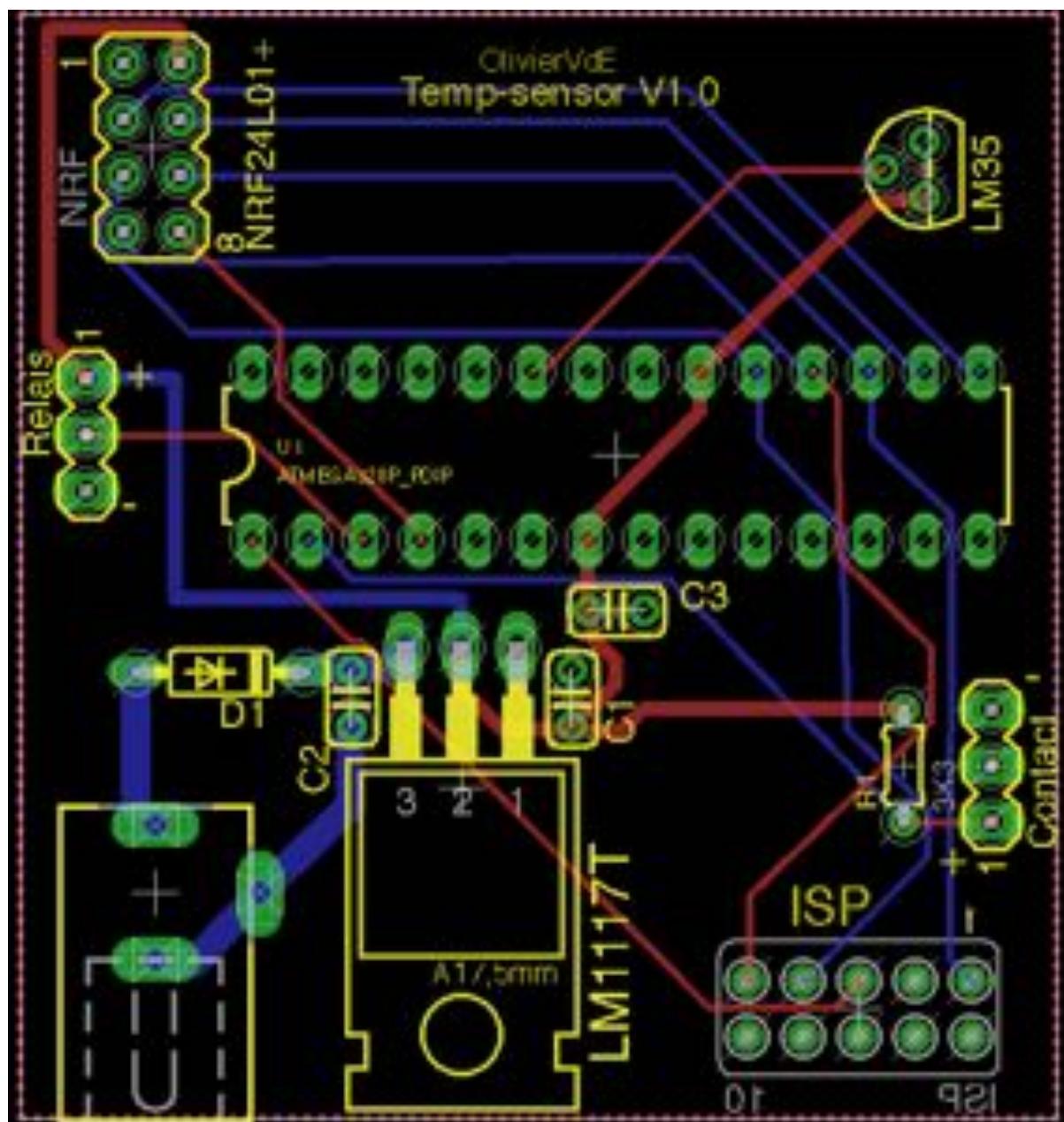
3.2 Layout van de print

3.2.1 Thermostaat

De 2 verschillende printplaten zijn gemaakt in eagle en opgestuurd naar seeedstudio voor fabricatie.



3.2.2 Temperatuursensor en relais-module



3.3 Foto's

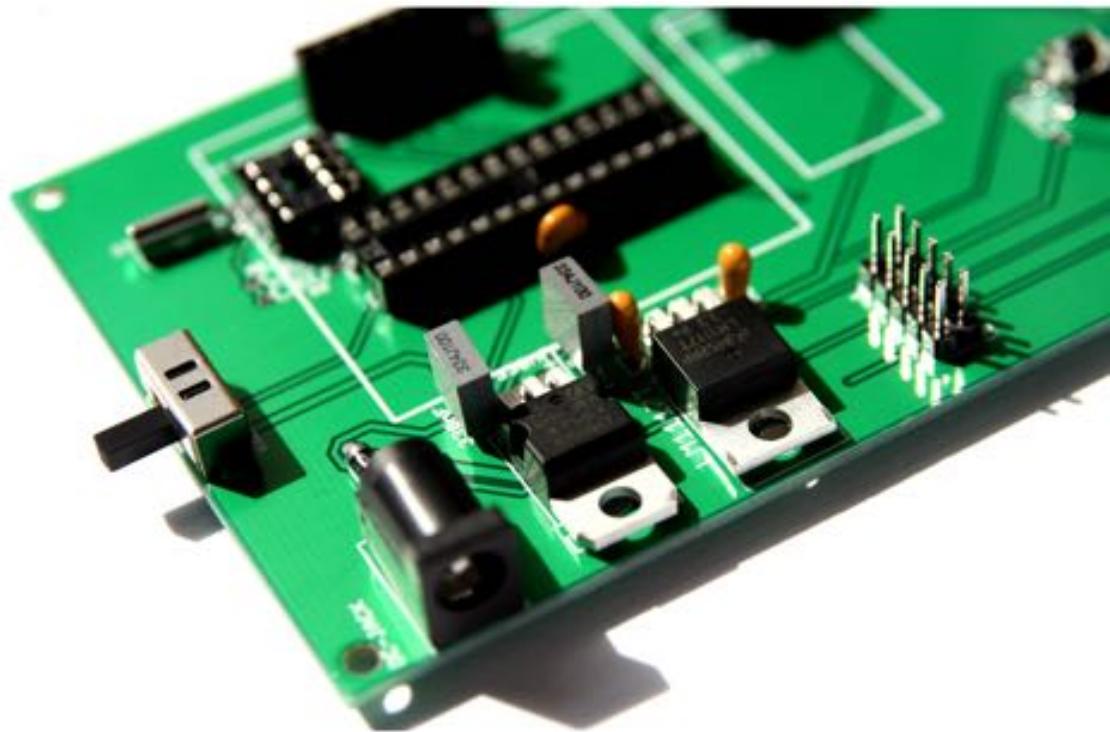


Foto1: Hoofdmodule close-up

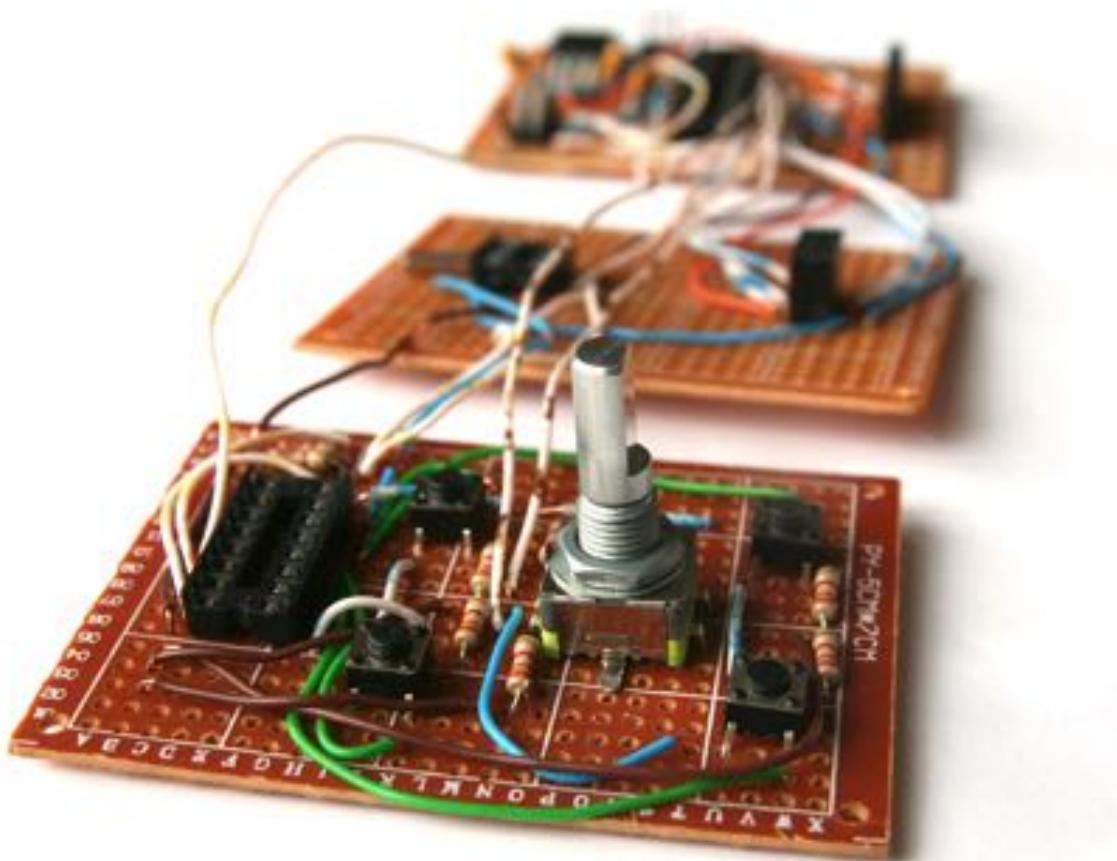


Foto2: Prototype hoofdmodule



Foto3: Temperatuursensor/Relais-module

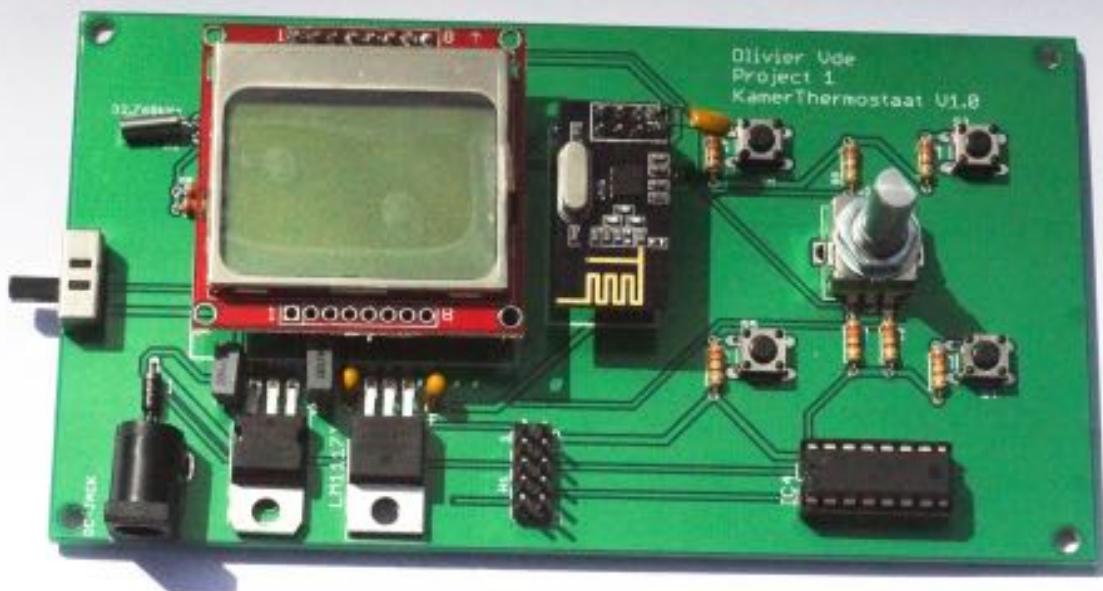


Foto4: Overzicht hoofdmodule met lcd

4 Broncode

4.1 Thermostaat

4.1.1 Uitlezen Rtc (rtc.h)

```
void RTC_write(uint8_t reg, uint8_t data)
{
    i2c_start();           //startconditie sturen
    i2c_send(0b11010000); // rtc aanspreken in write mode

    i2c_send(reg);        //het juiste register aanspreken
    i2c_send(data);       //de data sturen

    i2c_stop();           //stopconditie sturen
}
```

Functie1: Schrijven naar het Rtc

De eerste functie is gemaakt om een Byte weg te schrijven naar 1 van de registers in de Real-time clock. De argumenten zijn de te schrijven byte en het register waarnaar geschreven zal worden. De communicatie verloopt via i2c-functies.

```
uint8_t RTC_Read(uint8_t reg)
{
    uint8_t data;          //variabele om data te ontvangen

    i2c_start();           //startconditie sturen
    i2c_send(0b11010000); //rtc aanspreken in write mode
    i2c_send(reg);         //het juiste register aanspreken
    i2c_stop();

    i2c_start();           //nieuwe startconditie
    i2c_send(0b11010001); //opnieuw aanspreken rtc maar nu in read mode

    data = i2c_receive(0); //register uitlezen en data opslaan + nack

    i2c_stop();            //stopconditie sturen

    return data;           //data returnen
}
```

Functie2: lezen uit het rtc

De volgende functie kan een byte uit de registers van het rtc uitzelen. De functie heeft 1 argument, het te lezen register. De functie returnt de uitgelezen byte. Ook deze functie communiceert dmv. i2c met het rtc.

```
void get_uur(void)
{
    uint8_t uren = 0;
    uint8_t minuten = 0;

    minuten = RTC_Read(0x01);           //minuten lezen
    uren = RTC_Read(0x02);             //uren uitlezen

    BCD_To_ASCII(uren,uurRtc);
    BCD_To_ASCII(minuten,minRtc);
}
```

Functie3: uur uitlezen

Deze functie leest mbv. i2c het uur en minuten register van de rtc uit. Dit gebeurd met de functie `rtc_read()`. Na het uitlezen wordt het Bcd getal uit het rtc omgezet naar een string om later op het scherm af te drukken.

4.1.2 Inlezen Rotary encoder (rotary.h)

```
void RotaryCheckStatus(void)
{
    //Lezen van rotary encoder
    //Kijken of er links gedraaid word
    if(ROTA & (!ROTB)){
        while (bit_is_clear(ROTPIN, ROTPA));
        if (ROTB) rotarystatus=1; //als links draaiend, op 1 zetten

        else if(ROTB & (!ROTA)){
            while(bit_is_clear(ROTPIN, ROTPB));
            if (ROTA) rotarystatus=2;
                //als rechts draaiend, op 2 zetten
        }else if (ROTA & ROTB){
            while (bit_is_clear(ROTPIN, ROTPA));
            if (ROTB)
                rotarystatus=1;           //als links, op 1 zetten
            else rotarystatus=2; //als recht op 2
        }
    }
}
```

Functie1: draairichting lezen

De functie dient om de draairichting van de encoder te achterhalen. Voor deze functie zijn een aantal macro's gebruikt.

Door te kijken naar het verschil in opeenvolgende signalen kan er bepaald worden of er links of rechts gedraaid word. De variabele rotarystatus wordt dan op 1 of op 2 gezet.

Deze functie wordt uitgevoerd door de interrupt van timer0 die opgestart wordt bij het begin van het hoofdprogramma. De status van het register kan dan opgevraagd worden door de volgende functie.

```
#define ROTPORT PORTC
#define ROTDDR DDRC
#define ROTPIN PINC

#define ROTPA PC0
#define ROTPB PC1

#define ROTA !((1<<ROTPA)&ROTPIN)
```

Macro's rotary encoder

```
uint8_t RotaryGetStatus(void)
{
    return rotarystatus;      //status van de encoder opvragen
}
```

Deze functie retournd de waarde uit de globale variabele "rotarystatus" en kan zo in het hoofdprogramma verwerkt worden.

4.1.3 Het lcd (nokia_5110.h)

Voor de volgende functies worden een aantal simpele macro's gebruikt. Deze macro's zijn om de pinnen aan te duiden waarop het lcd-scherm op aangesloten is.

```
#define BIT(i) (1<<i)

#define SCLK_set PORTD|=BIT(7) //serial clock pin
#define SCLK_clr PORTD&=~BIT(7)

#define SDIN_set PORTD|=BIT(6) //serial data pin
#define SDIN_clr PORTD&=~BIT(6)

#define LCD_DC_set PORTD|=BIT(5) //data/commando
#define LCD_DC_clr PORTD&=~BIT(5)

#define LCD_CE_set PORTD|=BIT(4) //chip enable pin
#define LCD_CE_clr PORTD&=~BIT(4)

#define LCD_RST_set PORTD|=BIT(0) //reset pin
#define LCD_RST_clr PORTD&=~BIT(0)
```

Macro's lcd-scherm

```
void LCD_init(void)           //LCD initialiseren
{
    DDRD |= 0xF1;             //poorten als output zetten

    LCD_RST_clr;              //reset pin laag
    delay_1us();
    LCD_RST_set;              //reset pin hoog

    LCD_CE_clr;                //chip enable laag maken
    delay_1us();
    LCD_CE_set;                //chip enable hoog maken
    delay_1us();

    LCD_write_byte(0x21, 0);    // LCD mode instellen
    LCD_write_byte(0xc8, 0);    // bias instellen
    LCD_write_byte(0x06, 0);    // Temp-range instellen
    LCD_write_byte(0x13, 0);    // 1:48
    LCD_write_byte(0x20, 0);
    LCD_clear();                // Lcd leegmaken
    LCD_write_byte(0x0c, 0);    // lcd mode instellen op normal

    LCD_CE_clr;                //chip enable pin laag maken
}
```

De vorige functie dient om het lcd-scherm te initialiseren en alles juist in te stellen.

```
void LCD_write_byte(unsigned char dat, unsigned char command)
{
    unsigned char i;
    LCD_CE_clr;

    if (command == 0)          //moet het in het comando of in het data reg?
        LCD_DC_clr;
    else
        LCD_DC_set;

    for(i=0;i<8;i++)
    {
        if(dat&0x80)
            SDIN_set;      //als data 1 is, bit setten
        else
            SDIN_clr;     //als data 0 is, bit op 0

        SCLK_clr;         //klokpuls maken
        dat = dat << 1;   //data door shiften
        SCLK_set;         //klokpuls maken
    }
    LCD_CE_set;
}
```

Functie2: Een databyte doorsturen

Deze functie stuurt 1 databyte door naar het lcd, in ofwel het data of comando register. De functie heeft 2 argumenten, 1 met de data, en 1 met of het naar het data of comando register moet.

Als "Command" 0 is, wordt de data in het comando register geschreven, anders in het data-register. De functie stuurt de data door met een simpele vorm van spi.

```
void LCD_write_char(unsigned char c)
{
    unsigned char line;

    c -= 32;           //ascii codes omzetten naar de lookup table

    for (line=0; line<6; line++)
        LCD_write_byte(font6x8[c][line], 1);
}
```

Functie3: een karakter sturen.

In deze functie wordt een karakter in ascii-vorm omgezet naar de lookup-table die mee in de header file staat. In deze file staan alle tekens in een array op volgorde. De array noemt "font6x8".

4.1.4 i2c-port expander met knoppen

```
void lees_knoppen(void)          //leest de knoppen uit, en zet de waarde in
{                                //de globale array "knoppen[3]"
    uint8_t getal;

    i2c_start();
    i2c_send(0b01110001);        //expander aanspreken
    getal = i2c_receive(1);      //byte lezen

    getal = getal << 4;         //nibble switchen

    for (uint8_t i=0; i<4; i++)
    {
        uint8_t kopie = getal;
        kopie = kopie << i;      //i keer doorschuiven afhankelijk van de knop
        kopie = kopie & 0b10000000; //enkel hoogste bit overhouden

        if (kopie == 0b10000000)
        {
            switch(i)           //juiste knop op de juiste plaats in de array
            {
                case 0: knoppen[0] = 0; break;
                case 1: knoppen[3] = 0; break;
                case 2: knoppen[1] = 0; break;
                case 3: knoppen[2] = 0; break;
            }
        }
        else
        {
            switch(i)           //juiste knop op de juiste plaats in de array
            {
                case 0: knoppen[0] = 1; break;
                case 1: knoppen[3] = 1; break;
                case 2: knoppen[1] = 1; break;
                case 3: knoppen[2] = 1; break;
            }
        }
    }
}
```

Functie1: Knoppen inlezen

Deze functie maakt gebruik van i2c om de i2c-port expander uit te lezen. Op de expander zijn 4 knoppen aangesloten. Door een byte in te lezen en voor elke knop te checken of hij ingedrukt was of niet, kan de globale array "knoppen" gevuld worden met de juiste informatie over welke knoppen er ingedrukt zijn. Deze knoppen kunnen dan uitgelezen worden in het hoofdprogramma.

4.1.5 Nrf24L01 (nRF24L01.h)

Bovenaan de include file staan een aantal defines van de verschillende registers en instructies die in de nRF-chip beschikbaar zijn, deze defines zijn in een file gedownload van het internet en maken enkel het programma simpeler. De paarse woorden, zijn deze defines.

```
uint8_t * WriteToNrf(uint8_t ReadWrite, uint8_t reg, uint8_t *pack, uint8_t aantpack)
{
    // "Readwrite" = "W" of "R", reg = register om naar te sturen, pack = array met
    // de data, aantpack = aantal data in de array

    if (ReadWrite == W)          // als het w is, willen we schrijven
    {
        reg = W_REGISTER + reg;  // we voegen de write bit toe
                                   // aan het register
    }
    // we maken een array om op het einde de data te returnen
    static uint8_t ret[32];

    _delay_us(10);              // er zeker van zijn dat alle communicatie gedaan is
    CLEARBIT(PORTB,2);          // csn laag zetten, nrf luisterd nu
    _delay_us(10);
    WriteByteSPI(reg);          // nrf in lees of schrijfmode zetten voor het "reg" register
    _delay_us(10);

    for(uint8_t i = 0; i < aantpack; i++)
    {
        if (ReadWrite == R && reg != W_TX_PAYLOAD) // read comando?
        {
            // w_tx_payload is een uitzondering, deze kan niet het w bevatten
            ret[i] = WriteByteSPI(NOP);           // dummy bytes sturen
                                                   // om data binnen te halen
            _delay_us(10);
        }
        else
        {
            WriteByteSPI(pack[i]);             // stuur de data naar de nrf 1 per 1
            _delay_us(10);
        }
    }

    SETBIT(PORTB,2);              // csn terug hoog, nrf doet nu niks meer
    return ret;                  // de array terugsturen
}
```

Functie1: bytes zenden/ontvangen

De eerste functie is een functie met 2 mogelijkheden. De functie is in staat om een aantal bytes te sturen en te ontvangen van de nrf-module. De functie heeft 4 argumenten, een r/w-bit, het register om naar te sturen, de data om te sturen, en het aantal te sturen/ontvangen bytes.

De functie returnt een pointer naar de data die binnengehaald is.

```

void nrf24l01_init_RX(void)           //functie voor het initialiseren van de nrf in receiver mode
{
    _delay_ms(100);                  //wachten om er zeker van te zijn dat alle communicatie gedaan is

    uint8_t setup[5];               //een array om de writeToNrf functie te gebruiken

    //EN_AA = auto ack uitzetten voor alle pipes
    setup[0] = 0x00;                //de registerwaarde zetten
    WriteToNrf(W,EN_AA, setup,1);   //data sturen naar register

    //SETUP_RTR stelt het aantal pogingen in bij een mislukte transmittie
    setup[0] = 0x2f;                //0b0010 1111: elke 750us opnieuw proberen bij mislukking, 15keer opnieuw
                                    //proberen
    WriteToNrf(W,SETUP_RTR, setup, 1);

    //aantal datapipes instellen hier enkel dp0
    setup[0] = 0x01;
    WriteToNrf(W,EN_RXADDR, setup, 1); //data pipe 0 aanzetten

    //Rf-adres breedte instellen (1-5 in bytes)
    setup[0] = 0b00000011;          //0b0000 0011 = 5byte adres
    WriteToNrf(W,SETUP_AW, setup, 1);

    //rf kanaal instellen - frequentie 2.4 - 2.527ghz
    setup[0] = 0b01110000;          //kanaal 112, is een legaal kanaal buiten het bereik van bluethoot en wifi
    WriteToNrf(W,RF_CH, setup, 1);

    //RF_setup power down mode en data speed
    setup[0] = 0x07;
    WriteToNrf(W,RF_SETUP, setup, 1);

    //RX RF_adress setup met 5 bytes
    for (int i=0; i<5; i++)
    {
        setup[i] = 0x12;           //0x12 x 5 is het adres, het moet hetzelfde zijn op de receiver-
                                    //transmitter
    }
    WriteToNrf(W,RX_ADDR_P0, setup, 5); //we hebben datapipe0 gekozen, dus geven we deze een
                                    //adres

    //TX RF_adr setup met 5 bytes
    for (int i=0; i<5; i++)
    {
        setup[i] = 0x12;           //adres hetzelfde op receiver en transmitter
    }
    WriteToNrf(W, TX_ADDR, setup, 5);

    //breedte van de paketten instellen
    setup[0] = datlen;             //3 bytes sturen per transmittie
    WriteToNrf(W, RX_PW_P0, setup, 1);

    //Config register setup- hier kiezen we receiver of transmitter mode
    setup[0] = 0b00111111;          //receiver mode, enkel interrupt voor data ontvangen, power-up mode
    WriteToNrf(W, CONFIG, setup, 1);

    //het device heeft 1.5ms nodig om in standby te gaan
    _delay_ms(100);
}

```

De vorige functie Initialiseert de nrf module in receiver-mode. Er is ook een bijna identieke functie om een nrf in transmitter-mode te initialiseren, het enige verschil is de configuratie van het "CONFIG"-register. In de functie wordt er ingesteld dat er 4 pakketten per transmissie verstuurd worden. En het receiver adres wordt ingesteld op 0x1212121212. De interrupt wordt gegenereerd vanaf de data verzonden is.

```
void naar_tx_mode(void)
{
    /*eerst veranderen we het adres om met een andere module te
     *comuniceren*/
    uint8_t setup[5];
    //RX RF_adress setup met 5 bytes
    for (int i=0; i<5; i++)
    {
        setup[i] = 0x14;      //0x12 x 5 is het adres, het moet hetzelfde zijn op
                             //de reciever-transmitter
    }
    WriteToNrf(W, RX_ADDR_P0, setup, 5); //we hebben datapipe0 gekozen,
                                         //dus geven we deze een adres

    //TX RF_adr setup met 5 bytes
    for (int i=0; i<5; i++)
    {
        setup[i] = 0x14;      //adres hetzelfde op reciever en transmitter
    }
    WriteToNrf(W, TX_ADDR, setup, 5);

    /*Nu zetten we het om naar tx-mode*/

    //Config register setup- hier kiezen we receiver of transmitter mode
    setup[0] = 0b01011110;      //transmitter mode, power-up, enkel
                                //interupt als transmittie gelukt is
    WriteToNrf(W, CONFIG, setup, 1);

    //het device heeft 1.5ms nodig om in standby te gaan
    _delay_ms(100);
}
```

Functie3: mode omswitchen

In deze functie wordt er een andere modus opgestart, hier gaan we over van receiver-naar transmitter-mode. We veranderen ook het adres zodat de receiver nu naar een ander device kan transmitten. Het adres wordt hier 0x1414141414.

4.2 Relais-module

4.2.1 Nrf24L01

De nrf-modules moeten ook nog kunnen ontvangen, dit wordt gedaan door een combinatie van een functie en een interruptroutine.

```
ISR(INT0_vect)           //interruptroutine als er data ontvangen is
{
    nrf_int_disable();    //interrupts uitzetten, want de data is ontvangen
    CLEARBIT(PORTB,1);   //ce laag maken -> stop met luisteren naar data

    recData = WriteToNrf(R,R_RX_PAYLOAD, recData ,datlen);
    //data binnenhalen uit receive register
}
```

Interruptroutine receiver

We komen in de interruptroutine op de moment dat de nrf-module een datapakket ontvangt en een interrupt genegeerd op externe interrupt0. In de interruptroutine zetten we eerst de interrupt voor de nrf uit, omdat we het datapakket nu ontvangen hebben, en nog niet om een 2de gevraagd hebben. Daarna zetten we de Ce-pin op 0 om te zeggen aan de nrf dat hij mag stoppen met luisteren naar data.

In de volgende functie lezen we het "receive" register uit, dit doen we met de functie WriteToNrf, door het read(R) commando te geven. We geven hier ook een argument "recdata" mee, deze wordt echter niet gebruikt omdat het read-commando gegeven is, maar moet steeds wel een waarde bevatten omdat er anders een error volgt bij het compileren. De data die ontvangen is, wordt nu opgeslagen in de globale variabele "recdata".

```
void receive_Data(void)
{
    WriteToNrf(R, FLUSH_RX, NOP, 0);    //alle oude data verwijderen uit de
                                         //chip met flush
    nrf_int_enable();      //interrupt inschakelen om te kijken of er nieuwe data is

    SETBIT(PORTB,1);       //luisteren naar data
    _delay_ms(1000);       //1s luisten
    CLEARBIT(PORTB,1);     //stop met luisten

    nrf_int_disable();     //als er nu geen interrupt is, is er geen data, dus interrupt uit
}
```

Functie1: receive-data

In de receive functie wordt eerst het flush-commando uitgevoerd, dit maakt de data in het receive leeg. Daarna wordt de interrupt geactiveerd en wordt de ce-lijn hoog gezet, zo weet de nrf dat er geluisterd moet worden naar data. Dit wordt voor 1 seconde gedaan. Als er data wordt ontvangen wordt de interruptroutine opgestart en kan de data binnengehaald worden. Als er geen data beschikbaar is, wordt het allemaal gewoon terug afgezet.

4.2.2 main

```
uint8_t * recData = 0;      //variabele om nrf data te ontvangen

void verander_adres(void);

int main(void)
{
    InitSPI();
    nrf24l01_init_RX();
    verander_adres();          //het adres veranderen
    INT0_interrupt_init();
    sei();                      //interrupts aanzetten

    DDRD |= (1<<PD1); //als output schakelen, om de led aan te
                        sturen

    while(1)
    {
        nrf_reset();           //nrf ontvangstregister leegmaken
        recieve_Data();         //kijken of er data is

        if (recData[0] != 0)
        {
            if (recData[0] == 0x20) //dan is de data voor hier
            {
                if (recData[1] == 0x30) //als 0x30 dan moet de
                                //verwarming aan
                {
                    SETBIT(PORTD,1);
                }
                else
                {
                    CLEARBIT(PORTD,1);
                }
            }
        }
    }
}
```

Main functie

Deze main-functie is vrij simpel, er worden een aantal dingen geinitialiseerd, en vervolgens wordt er gewacht tot er data ontvangen wordt, en wordt hieruit opgemaakt of de uitgang hoog of laag moet zijn.

De interrupt bij dit programma is dezelfde als in het vorig hoofdstuk over de nrf beschreven is, hierin wordt de data binnengehaald.

```

void verander_adres(void)          //deze functie veranderd het adres
van de transciever om meerdere comunicaties te doen
{
    uint8_t setup[5];

    //RX RF_adress setup met 5 bytes
    for (int i=0; i<5; i++)
    {
        setup[i] = 0x14; //0x14 x 5 is het adres, het moet hetzelfde
                           zijn op de reciever-transmitter
    }
    WriteToNrf(W, RX_ADDR_P0, setup, 5);
    //TX RF_adr setup met 5 bytes
    for (int i=0; i<5; i++)
    {
        setup[i] = 0x14;      //adres hetzelfde op reciever en
                           transmitter
    }
    WriteToNrf(W, TX_ADDR, setup, 5);
}

```

Functie1: verander adres

Deze functie verandert het adres van 0x1212121212 naar 0x1414141414.

4.3 Temperatuursensor

4.3.1 Nrf24l01

In dit deel over de Nrf gaan we dieper in op het zenden van data. Ook dit gebeurd door een combinatie van een interruptroutine en een functie.

```
void transmit_Data(uint8_t * buff)
{
    WriteToNrf(R, FLUSH_TX, buff, 0);      //alle oude data verwijderen uit de
                                             chip met flush
    WriteToNrf(R, W_TX_PAYLOAD, buff, datlen); //stuur de data in de buffer
                                                naar de nrf

    nrf_int_enable();                      //interrupt opstarten om te kijken of transmissie
                                             kan worden voltooid.

    _delay_ms(20);                         //even wachten zodat de data er zeker inzit
    SETBIT(PORTB, 1);                     //ce hoogzetten = transmit data
    _delay_us(20);                        //even wachten
    CLEARBIT(PORTB,1);                   //ce terug naar omlaag -> transmittie stoppen
    _delay_ms(20);                        //even wachten voor einde

}
```

Functie1: Data verzenden

Deze functie heeft 1 argument, een datapakket om te verzenden. Eerst wordt het tx-register leeggemaakt met een flush, en daarna wordt de data erin geladen. Als we vervolgens de Chip-enable pin hoog maken, begint de chip met de transmissie van de data. De interruptroutine wordt geactiveerd op de moment dat de data verzonden is, dan wordt er een externe interrupt gegenereerd.

```
ISR(INT0_vect)                      //interrupt routine als data in de lucht is
{
    nrf_int_disable();
    CLEARBIT(PORTB,1);                //ce laag maken
}
```

In deze interruptroutine wordt de interrupt uitgezet, en de chip-enable pin terug laag gemaakt.

4.3.2 HTU21D-temp sensor

De temperatuursensor wordt hier gebruikt met een bitbang-i2c.

```
void initHTU21D(void)           //deze functie initialiseert de Htu21d-
{                                temperatuursensor in 12-bit mode
    uint8_t gegevens;           //een van om tijdelijk iets op te slaan
    i2c_Start();                //adres in write mode
    i2c_Write(0x80);            //soft reset
    i2c_Stop();
    i2c_Start();
    i2c_Write(0x80);            //adres met schrijfcomando
    i2c_Write(0xe7);            //lees user-register comando
    i2c_Start();
    i2c_Write(0x81);            //lees mode
    gegevens = i2c_Read(0);     //ontvangen met nack
    gegevens |= 0x00000001;      //12-bitmode aanzetten
    i2c_Start();
    i2c_Write(0x80);            //schrijf mode
    i2c_Write(0xE6);            //schrijven in user-register
    i2c_Write(gegevens);        //register juistzetten
    i2c_Stop();
}
```

Functie1: initialiseren

In deze functie initialiseren we de temperatuursensor in 12-bit mode. Dit wordt gedaan door eerst het user-register (config-reg) uit te lezen, de juiste bits te activeren en dat de nieuwe inhoud terug weg te schrijven.

```
temperatuur = read_temp();          //temperatuur meten
#define KOMMA (10)
temp1 = (int)temperatuur;           // opsplitsen van float in 2
                                    integers
temp2 = ((int)(temperatuur*KOMMA)%KOMMA);
```

Functie2: float omzetten naar 2 integers

In deze functie wordt de gemeten temp in float vorm omgezet naar 2 bytes, 1 voor de komma en 1 erna. Dit is omdat we byte per byte doorsturen en het ook zo verwacht wordt op de hoofdmodule.

```

float read_temp(void)           //deze functie leest de meting uit, en berekend de
                                temp in float vorm
{
    uint8_t getal1, getal2;
    uint16_t meting;
    float temperatuur, deling;

    i2c_Start();
    i2c_Write(0x80);          //adres in write mode
    i2c_Write(0xF3);          //temperatuurmeting in non-hold mode

    i2c_Start();

    while (i2c_Write(0x81) == 0) //kijken of de data al beschikbaar is,
                                //als niet blijven kijken
    {
        i2c_Start();
    }

    getal2 = i2c_Read(1);      //msb uitlezen
    getal1 = i2c_Read(1);      //lsb-uitlezen
    i2c_Read(0);              //check uitlezen maar niks mee doen
    i2c_Stop();

    getal1 = getal1 & 0b11111100; //laagste 2 bits zijn statusbits, dus
                                //maskeren
    meting = getal1 | (getal2 << 8); //msb en lsb samenvoegen tot 16-bit getal

    deling = (float)meting / (float) 65536; //de deling berekenen, met
                                              //typecast

    temperatuur = -46.85 + (175.72 * deling); //de temperatuur
                                                //berekenen, wordt een float.

    return temperatuur;          //return temp
}

```

Functie 3: Meten temp

In deze functie wordt de temperatuur gemeten. Eerst sturen we het "temp non-hold" commando, dit wil zeggen dat we vragen om de meting uit te voeren, maar de i2c bus vrij te laten. Hoelang de meting zal duren weten we niet, dus moeten we pollen of de meting al klaar is. Door een "start" en een "read" commando te sturen kunnen we aan de ack zien of de data al beschikbaar is. Als er geen ack volgt op de pol, dan proberen we gewoon nog een keer tot de data wel beschikbaar is.

Na de ack lezen we 3 bytes in, de eerste 2 zijn de databytes, de derde is een controlebyte die we niet nodig hebben. Na het inlezen van de data moeten we de 2 statusbits maskeren voordat we de temperatuur kunnen berekenen.

Na de berekening wordt de temperatuur in een float-vorm naar buiten gestuurd.

4.4 Sleep-mode

Om batterij te sparen wordt er gebruik gemaakt van 1 van de sleep-modes van de microcontroller. Door aan de controller het sleep commando te geven gaat deze in power save mode, en wordt alles behalve timer2 uitgezet. Timer2 en zijn interrupt dienen om de controller uit de sleep-mode te halen.

```
void init_sleep(void)
{
    init_timer2();           //timer 2 initialiseren
    SMCR |= (1<<SM1) | (1<<SM0);      //power-save mode
                                         inschakelen(datasheet)
}

void start_slaap(void)          //start de slaapmode op, het stopt
{                                als timer2 een overflow geeft
    SMCR |= (1<<SE);          //slaap mode activeren door se-bit hoog
                                te zetten
    timer2_start();            //timer 2 opstarten
    sleep_cpu();               //slaap instructie uitvoeren, nu gaan
                                we slapen
}

void stop_slaap(void)           //timer 2 stoppen
{
    SMCR &= ~(1<<SE);        //slaap mode uitzetten door se-bit laag
                                te maken
    timer2_stop();             //timer 2 wordt gestopt
}
```

De functies om in en uit slaapstand te gaan

In deze functies wordt de slaap-mode in en uit geschakeld door een aantal registers juist te zetten en de timer te stoppen of te starten.

```
for (uint16_t i=0; i<500; i++)
{
    start_slaap();
}
```

```
ISR (TIMER2_OVF_vect)          //interrupt vector voor de slaapfunctie;
{
    stop_slaap();
}
```

In de main

In de main staat een loop die 500 keer doorlopen wordt om 500 keer 2ms in slaapstand te gaan. Na 2ms gaat de timer in overflow en in interrupt. Dan stopt de slaapfunctie.

5 Conclusie

5.1 Reflectie

Na 13 weken werken aan het project is het volledig in orde gekomen en werkt alles van de vooropgestelde eisen. Het enige wat niet gelukt is, is het meten van de voedingsspanning met de adc. Hierdoor komt er dus geen melding op de hoofdmodule als de batterij van de temperatuur-module bijna leeg is. Het is echter wel te zien op het scherm als de temp-module niet beschikbaar is.

Een ander probleem dat zich heeft voorgedaan, is het meten van de temperatuur. Initieel was de bedoeling om dit analoog te doen met een lm35, maar door een designfout op de temperatuur-module pcb, kon dit niet meer. De reden hiervoor is, dat ik ervoor gekozen heb om het systeem te laten draaien op 3,3v omdat de RF-module deze voedingsspanning nodig heeft, en de microcontroller dit ook aankan. Maar ik heb toen er geen rekening mee gehouden dat de lm35 een minimumspanning van 4v nodig heeft. Daarom heb ik gekozen om de temperatuur te meten met een HTU21D sensor, die werkt met i2c.

Om het in de toekomst op te lossen, zou er een extra regelaar op de print moeten om ook 5v op de print aanwezig te maken.

Op de markt bestaan er al zeer veel regelbare en programmeerbare thermostaten, maar die werken allemaal met een ingebouwde temperatuursensor, en kunnen niet werken met een thermometer die op een logische plaats gepositioneerd is. De regelmodule moet op een goed bereikbare en toegankelijke plaats staan, maar deze plaats is meestal niet de ideale plaats om de temperatuur te meten, omdat deze meestal in de warmste ruimte staat, en zo worden de koudste ruimtes in huis nooit naar de ideale temperatuur verwarmd.

Het aansturen van een 1 verwarming is ook niet echt een ideale oplossing, maar mijn ontwerp is perfect instaat om een hele centrale verwarming aan te sturen moest dit nodig zijn.

Bij het ontwerpen van mijn systeem heb ik ook goed nagedacht over mogelijke uitbreiding die achteraf nog toegevoegd kunnen worden. Zo heb ik bv. de module om de temperatuur te meten ontworpen dat deze meerdere doelen kan hebben. Nu in de huidige versie wordt deze enkel gebruikt om in de ene module de temperatuur te meten en in de andere de verwarming aan en uit te sturen, maar de mogelijkheid bestaat om ook nog een aantal sensoren toe te voegen om bv. te detecteren of er nog ramen openstaan, of nog een extra temperatuursensor er bij op aan te sluiten. Door deze ontwerpstrategie is mijn project inzetbaar op meerdere plaatsen dan enkel waar het voor gemaakt is.

5.2 Kostprijs

Onderdeel	Aantal	Prijs	Totaal
Nrf24l01+	3	1 €	3 €
Atmega 328p	3	3,50 €	10,50 €
LM7805	1	0,23 €	0,23 €
LM1117	3	0,40 €	1,20 €
Dc-Jack	3	0,16 €	0,48 €
Switch	1	0,22 €	0,22 €
Nokia 5110	1	2,30 €	2,30 €
Rtc + Kristal	1	Gratis sample	0 €
Rotary encoder	1	1,30 €	1,30 €
Pushbutton	4	0,05 €	0,20 €
PCF8574P i/o-exp	1	0,65 €	0,65 €
HTU21D temp-sens	1	4,00 €	4,00 €
Cond + weerstanden	1	2,00 €	2,00 €
Pcb's	1	50,00 €	50,00 €
Totaal			76,08 €

De Kostprijs die verwacht was, waar de pcb's nog niet mee inbegrepen waren was 41€. De prijs is iets hoger geworden omdat ik beslist heb om zelf 2 pcb's te laten maken.

Bijlage 1: Broncode Hoofdmodule

- Main.c
- i2c.h
- Nokia_5110.h
- Rtc.h
- Rotary.h
- Knoppen.h
- Nrf24l01.h

```
/*
 * Hoofdmodule_v1.c
 *
 * Created: 28/03/2015 14:43:04
 * Author: Olivier
 */

#include <avr/io.h>
#define F_CPU 8000000
#include <util/delay.h>      //standaard includes
#include <string.h>
#include <avr/interrupt.h>

#include "i2c.h"
#include "nokia_5110.h"
#include "Rtc.h"                //eigen include files
#include "rotary.h"
#include "knoppen.h"
#include "nRF24L01.h"

char uurRtc[2];
char minRtc[2];           //2 var om het uur uit te lezen en af te printen

uint8_t tempH1 = 0;        //2 var voor de huidige temp
uint8_t tempH2 = 0;
uint8_t tempIn1 = 20;
uint8_t tempIn2 = 5;       //2 var voor de ingestelde temp

uint8_t uurhUur = 22;
uint8_t uurhMin = 00;
uint8_t uurlUur = 11;     //een aantal var om bij te houden wanneer het aan
    of uit moet
uint8_t uurlMin = 00;

uint8_t * recData = 1;      //variabele om nrf data te ontvangen
uint8_t data_relais[4] = {0x20,0x30,0,0}; //data om te sturen
uint8_t trxMode = 1;        //variabele om aan int door te geven of we
    zenden of ontvangen 1=rx;
uint8_t relais = 1;         //var om bij te houden of relais aan of uit
    moet

int main(void)
{
    LCD_init();           //lcd initialiseren
    LCD_clear();

    InitSPI();
    nrf24l01_init_RX();   //init nrf
    INT0_interrupt_init();

    init_RTC();           //rtc initialiseren
    RTC_write(0x02,0x11);
    RTC_write(0x01,0x04);

    RotaryInit();         //rotary initialiseren
    RotaryResetStatus();  //register leegmaken
    Timer0_Start();        //timer 0 opstarten
```

```
uint8_t knop2 = 0;           //een var om bij te houden of de knop ingedrukt
                            geweest is
uint8_t knop3 = 0;
uint8_t knop4 = 0;

uint8_t aanUit = 1;          //een var om te kijken of de module aan of uit
                            moet staan
uint8_t uurOk = 0;           //een var om de uurstatus bij te houden
//uint8_t uurtc = 1,mintc = 1;
uint8_t batterij = 0;        //een var om de batterijstatus bij te houden

SETBIT(DDRD,1);             //lcd-lichtje als uitgang zetten

while(1)
{
    /*In dit deel van het programma printen we al het nodige naar het lcd
     scherm.
     Daarvoor lezen we eerst het uur uit het RTC*/
    LCD_clear();
    get_uur();                  //uur lezen uit rtc komt in 'uurRtc' en 'minRtc'

    LCD_write_string(0,0,uurRtc);
    LCD_write_string(12,0,":");      //uur naar scherm printen
    LCD_write_string(18,0,minRtc);

    if (aanUit == 1) LCD_write_string(66,0,"Aan"); //aan of uit printen
                                                op lcd
    else LCD_write_string(66,0,"uit");           //afhankelijk van de
                                                aan/uit-knop

    LCD_write_int(30,2,tempH1);
    //LCD_write_int(20,2,tempH1);
    LCD_write_string(42,2,"");                //de huidige temperatuur
                                                printen op het lcd
    LCD_write_int(48,2,tempH2);

    LCD_write_int(60,5,tempIn1);
    LCD_write_string(72,5,"");                //de ingestelde temperatuur op
                                                het lcd tonen
    LCD_write_int(78,5,tempIn2);

    if (knop3 == 1)      //enkel als knop ingedrukt is tonen
    {
        LCD_write_int(0,4,uurhUur);
        LCD_write_string(12,4,":");          //het maximum aan-uur
        LCD_write_int(18,4,uurhMin);
    }

    if (knop4 == 1)      //enkel als knop ingedrukt is tonen
    {
        LCD_write_int(0,4,uurlUur);
        LCD_write_string(12,4,":");          //het minimum aan-uur
        LCD_write_int(18,4,uurlMin);
    }

    if (batterij == 0) LCD_write_string(0,5,"bat"); //als batterij
                                                leeg is, melding op het scherm geven
```

```
/*In dit deel lezen we de 4 knoppen uit uit de i/o expander
er worden ook functies aantoe gekend */

lees_knoppen();           //knoppen lezen

//hier wordt de temp ingesteld
if (knoppen[3] == 1 || knop2 == 1)      //als knop 2 ingedrukt is, kan
er gedraaid worden
{
    if (((PIN & (1<<PIN2)) == 0) && knop2 == 1)          //als er op de
        rotary encoder gedrukt wordt,
    {
        registreren
        knop2 = 0;
    }
    else
    {
        knop2 = 1;
        knop3 = 0;
        knop4 = 0;
    }
    switch (RotaryGetStatus())      //rotarystatus opvragen(wordt
        verwerkt met interrupttimer)
    {
        case 1: tempMin(); RotaryResetStatus();      //als 1, dan wordt er
            naar links gedraaid, dus min
            break;
        case 2: tempPlus(); RotaryResetStatus();      //als 2, dan wordt er
            naar rechts gedraaid, dus plus
            break;
    }
}
//hier wordt het uitschakeluur ingesteld
if (knoppen[0] == 1 || knop3 == 1)      //als knop 2 ingedrukt is, kan
er gedraaid worden
{
    if (((PIN & (1<<PIN2)) == 0) && knop3 == 1)          //als er op de
        rotary encoder gedrukt wordt,
    {
        registreren
        knop3 = 0;
    }
    else
    {
        knop3 = 1;
        knop2 = 0;
        knop4 = 0;
    }
    switch (RotaryGetStatus())      //rotarystatus opvragen(wordt
        verwerkt met interrupttimer)
    {
        case 1: uurhmin(); RotaryResetStatus();      //als 1, dan wordt
            er naar links gedraaid, dus min
            break;
        case 2: uurhplus(); RotaryResetStatus();      //als 2, dan wordt
            er naar rechts gedraaid, dus plus
            break;
    }
}
```

```
//hier wordt het inschakeluur ingesteld
if (knoppen[1] == 1 || knop4 == 1)          //als knop 2 ingedrukt is, kan
    er gedraaid worden
{
    if (((PINC & (1<<PIN2)) == 0) && knop4 == 1)      //als er op de
        rotary encoder gedrukt wordt,
    {
        registreren
        knop4 = 0;
    }
    else
    {
        knop4 = 1;
        knop2 = 0;
        knop3 = 0;
    }
    switch (RotaryGetStatus())           //rotarystatus opvragen(wordt
        verwerkt met interrupttimer)
    {
        case 1: uurlmin(); RotaryResetStatus();      //als 1, dan wordt
            er naar links gedraaid, dus min
        break;
        case 2: uurlplus(); RotaryResetStatus();      //als 2, dan wordt
            er naar rechts gedraaid, dus plus
        break;
    }
}
```

/*In dit deel gebruiken we de nrf module om de temperatuur te
ontvangen*/

```
nrf_reset();                      //nrf ontvangstregister leegmaken
recieve_Data();                    //kijken of er data is

if (recData[0] != 0)              //als data niet 0 is, mag het verwerkt
    worden
{
    tempH1 = recData[0];          //de eerste data is het eerste deel
    van de temp
    tempH2 = recData[1];          //tweede deel
    batterij = recData[3];        //4de data is de status van de
    batterij
}
```

/*Hier kijken we of de aanuitschakelaar aan of uit staat*/

```
if ((PINC & (1<<PIN3)) == 0) aanUit = 0;
else aanUit = 1;

if ((PINC & (1<<PIN2)) == 0) SETBIT(PORTD,1);
else CLEARBIT(PORTD,1);
```

/*Hier gaan we kiezen of het ventiel aangetrokken moet worden of
niet*/

```
if (aanUit == 1 && uurOk == 1)      //enkel als aanuit 1 is, mag het
    werken
{
```

```
if (tempH1 < tempIn1)           // ingestelde temp voor de komma groter
    dan huidig?
{
    relais = 1;                 // zo ja, relais aan
}
else if (tempH1 == tempIn1)     // ingestelde temp voor de komma gelijk
    aan huidig?
{
    if (tempH2 < tempIn2)      // achter de komma groter?
    {
        relais = 1;             // zo ja, relais aan
    }
    else
    {
        relais = 0;             // zo nee, relais uit
    }
}
else if (tempH1 > tempIn1)     // als het warmer is dan ingesteld,
    niks doen
{
    relais = 0;                // relais uit
}
}
else
{
    relais = 0;
}

/* Hier gaan we kijken of het huidige uur wel tussen de juiste waardes
   valt */

if (uurtc > uurlUur && uurtc < uurhUur)
{
    uurOk = 1;
}
else if (uurtc == uurlUur)
{
    if (mintc > uurlMin) uurOk = 1;
    else uurOk = 0;
}
else if (uurtc == uurhUur)
{
    if (mintc < uurhMin) uurOk = 1;
    else uurOk = 0;
}
else
{
    uurOk = 0;
}

// hier gaan we de receiver omzetten in een transmitter om data te
verzenden naar de relais module

        // data om te sturen
if (relais == 1)
{
    data_relais[1] = 0x30;
}
else
```

```
{  
    data_relais[1] = 0x40;  
}  
  
naar_tx_mode();           //we zetten de tranciever in tx-mode op een  
                        ander adres  
  
nrf_reset();  
transmit_Data(data_relais); //data versturen  
_delay_ms(100);  
  
naar_rx_mode();           //terug in normale mode  
  
}  
}  
  
ISR(TIMER0_OVF_vect)      //bij timer0 overflow een interrupt  
{  
    RotaryCheckStatus();   //de richting van de rotary encoder lezen  
}  
  
ISR(INT0_vect)            //interruptroutine als er data ontvangen is  
{  
    nrf_int_disable();     //interrupts uitzetten, want de data is  
                        ontvangen  
    CLEARBIT(PORTB,1);    //ce laag maken -> stop met luisteren naar data  
  
    if (trxMode == 1)      //kijken of we wel in rx-mode zitten, anders moeten we  
                        niks binnenhalen  
    {  
        recData = WriteToNrf(R,R_RX_PAYLOAD, recData ,datlen); //data  
                        binnenhalen uit receive register  
    }  
  
}  
  
void tempMin(void)         //deze functie vermindert te ingestelde temp  
{  
    if (tempIn2 > 0)  
    {  
        tempIn2 = tempIn2-1;  
    }  
    else if (tempIn2 == 0)  
    {  
        if (tempIn1 > 15)  
        {  
            tempIn1 = tempIn1 - 1;  
            tempIn2 = 9;  
        }  
        else if(tempIn1 == 15)  
        {  
            tempIn1 = 30;  
            tempIn2 = 9;  
            LCD_clear();  
        }  
    }  
}
```

```
}

void uurhmin(void)      //deze functie vermindert het ingestelde uur
{
    if (uurhMin > 0)
    {
        uurhMin = uurhMin-1;
    }
    else if (uurhMin == 0)
    {
        if (uurhUur > 0)
        {
            uurhUur = uurhUur - 1;
            uurhMin = 59;
        }
        else if(uurhUur == 0)
        {
            uurhUur = 23;
            uurhMin = 59;
            LCD_clear();
        }
    }
}

void uuurlmin(void)      //deze functie vermindert het ingestelde uur
{
    if (uuurlMin > 0)
    {
        uuurlMin = uuurlMin-1;
    }
    else if (uuurlMin == 0)
    {
        if (uuurlUur > 0)
        {
            uuurlUur = uuurlUur - 1;
            uuurlMin = 59;
        }
        else if(uuurlUur == 0)
        {
            uuurlUur = 23;
            uuurlMin = 59;
            LCD_clear();
        }
    }
}

void tempPlus(void)      //deze functie verhoogt te ingestelde temp
{
    if (tempIn2 < 9)
    {
        tempIn2++;
    }
    else if (tempIn2 == 9)
    {
        if (tempIn1 < 30)
        {
            tempIn1++;
            tempIn2 =0;
        }
    }
}
```

```
    else if (tempIn1 == 30)
    {
        tempIn1 = 15;
        tempIn2 = 0;
        LCD_clear();
    }
}

void uurhplus(void)      //deze functie verhoogt te ingestelde temp
{
    if (uurhMin < 59)
    {
        uurhMin++;
    }
    else if (uurhMin == 59)
    {
        if (uurhUur < 23)
        {
            uurhUur++;
            uurhMin =0;
        }
        else if (uurhUur == 23)
        {
            uurhUur = 0;
            uurhMin = 0;
            LCD_clear();
        }
    }
}

void uuurlplus(void)      //deze functie verhoogt te ingestelde temp
{
    if (uuurlMin < 59)
    {
        uuurlMin++;
    }
    else if (uuurlMin == 59)
    {
        if (uuurlUur < 23)
        {
            uuurlUur++;
            uuurlMin =0;
        }
        else if (uuurlUur == 23)
        {
            uuurlUur = 0;
            uuurlMin = 0;
            LCD_clear();
        }
    }
}

void naar_tx_mode(void)
{
    /*eerst veranderen we het adres om met een andere module te communiceren*/
}
```

```
uint8_t setup[5];

//RX RF_adress setup met 5 bytes
for (int i=0; i<5; i++)
{
    setup[i] = 0x14;      //0x12 x 5 is het adres, het moet hetzelfde zijn
                          op de reciever-transmitter
}
WriteToNrf(W, RX_ADDR_P0, setup, 5); //we hebben datapipe0 gekozen, dus
                                      geven we deze een adres

//TX RF_adr setup met 5 bytes
for (int i=0; i<5; i++)
{
    setup[i] = 0x14;      //adres hetzelfde op reciever en transmitter
}
WriteToNrf(W, TX_ADDR, setup, 5);

/*Nu zetten we het om naar tx-mode*/

//Config register setup- hier kiezen we reciever of transmitter mode
setup[0] = 0b01011110;      //transmitter mode, power-up, enkel interrupt
                           als transmittie gelukt is
WriteToNrf(W, CONFIG, setup, 1);

//het device heeft 1.5ms nodig om in standby te gaan
_delay_ms(100);
}

void naar_rx_mode(void)
{
    /*We veranderen het adres terug naar origineel*/

    uint8_t setup[5];

    //RX RF_adress setup met 5 bytes
    for (int i=0; i<5; i++)
    {
        setup[i] = 0x12;      //0x12 x 5 is het adres, het moet hetzelfde zijn
                              op de reciever-transmitter
    }
    WriteToNrf(W, RX_ADDR_P0, setup, 5); //we hebben datapipe0 gekozen, dus
                                         geven we deze een adres

    //TX RF_adr setup met 5 bytes
    for (int i=0; i<5; i++)
    {
        setup[i] = 0x12;      //adres hetzelfde op reciever en transmitter
    }
    WriteToNrf(W, TX_ADDR, setup, 5);

    /*Nu zetten we het om naar tx-mode*/

    //Config register setup- hier kiezen we reciever of transmitter mode
    setup[0] = 0b00111111;      //transmitter mode, power-up, enkel interrupt
                               als transmittie gelukt is
    WriteToNrf(W, CONFIG, setup, 1);

    //het device heeft 1.5ms nodig om in standby te gaan
    _delay_ms(100);
}
```

```
/*
 * i2c.h
 *
 * Created: 27/03/2015 8:26:45
 * Author: Olivier
 */

void i2c_start(void)
{
    TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);      //startconditie maken
    while((TWCR & (1<<TWINT)) == 0);      //wachten tot transmittie voltooid is
}

void i2c_stop(void)
{
    TWCR = (1<<TWINT) | (1<<TWST0) | (1<<TWEN);      //stuur stopconditie
}

void i2c_send(uint8_t DataByte)
{
    TWDR = DataByte;          //byte in i2c register laden
    TWCR = (1<<TWINT) | (1<<TWEN);      //interrupt wegdoen en transmittie
    starten
    while((TWCR & (1<<TWINT)) == 0);      //wachten tot transmittie voltooid is
}

uint8_t i2c_receive(uint8_t ack)
{
    if (ack == 1)
    {
        TWCR = (1<<TWINT) | (1<<TWEA) | (1<<TWEN); //interrupt clearen en
        transmittie starten met ack
    }
    else
    {
        TWCR = (1<<TWINT) | (1<<TWEN);      //interrupt clearen en transmittie
        starten zonder ack
    }

    while((TWCR & (1<<TWINT)) == 0); //wachten tot data binnen is
    return TWDR;          //ontvangen byte returnen
}
```

```
//  
#define BIT(i) (1<<i)  
  
#define SCLK_set PORTD|=BIT(7) //serial clock pin  
#define SCLK_clr PORTD&=~BIT(7)  
  
#define SDIN_set PORTD|=BIT(6) //serial data pin  
#define SDIN_clr PORTD&=~BIT(6)  
  
#define LCD_DC_set PORTD|=BIT(5) //data/commando  
#define LCD_DC_clr PORTD&=~BIT(5)  
  
#define LCD_CE_set PORTD|=BIT(4) //chip enable pin  
#define LCD_CE_clr PORTD&=~BIT(4)  
  
#define LCD_RST_set PORTD|=BIT(0) //reset pin  
#define LCD_RST_clr PORTD&=~BIT(0)  
  
void LCD_write_byte(unsigned char, unsigned char);  
void LCD_write_string(unsigned char, unsigned char, char*);  
void LCD_write_char(unsigned char);  
void LCD_set_XY(unsigned char, unsigned char);  
void LCD_clear(void);  
void LCD_init(void);  
  
// 6 x 8 font  
// 1 pixel space at left and bottom  
// index = ASCII - 32  
const unsigned char font6x8[][6] =  
{  
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }, // sp  
    { 0x00, 0x00, 0x00, 0x2f, 0x00, 0x00 }, // !  
    { 0x00, 0x00, 0x07, 0x00, 0x07, 0x00 }, // "  
    { 0x00, 0x14, 0x7f, 0x14, 0x7f, 0x14 }, // #  
    { 0x00, 0x24, 0x2a, 0x7f, 0x2a, 0x12 }, // $  
    { 0x00, 0x62, 0x64, 0x08, 0x13, 0x23 }, // %  
    { 0x00, 0x36, 0x49, 0x55, 0x22, 0x50 }, // &  
    { 0x00, 0x00, 0x05, 0x03, 0x00, 0x00 }, // '  
    { 0x00, 0x00, 0x1c, 0x22, 0x41, 0x00 }, // (   
    { 0x00, 0x00, 0x41, 0x22, 0x1c, 0x00 }, // )  
    { 0x00, 0x14, 0x08, 0x3e, 0x08, 0x14 }, // *  
    { 0x00, 0x08, 0x08, 0x3e, 0x08, 0x08 }, // +  
    { 0x00, 0x00, 0x00, 0xa0, 0x60, 0x00 }, // ,  
    { 0x00, 0x08, 0x08, 0x08, 0x08, 0x08 }, // -  
    { 0x00, 0x00, 0x60, 0x60, 0x00, 0x00 }, // .  
    { 0x00, 0x20, 0x10, 0x08, 0x04, 0x02 }, // /  
    { 0x00, 0x3e, 0x51, 0x49, 0x45, 0x3e }, // 0  
    { 0x00, 0x00, 0x42, 0x7f, 0x40, 0x00 }, // 1  
    { 0x00, 0x42, 0x61, 0x51, 0x49, 0x46 }, // 2  
    { 0x00, 0x21, 0x41, 0x45, 0x4b, 0x31 }, // 3  
    { 0x00, 0x18, 0x14, 0x12, 0x7f, 0x10 }, // 4  
    { 0x00, 0x27, 0x45, 0x45, 0x45, 0x39 }, // 5  
    { 0x00, 0x3c, 0x4a, 0x49, 0x49, 0x30 }, // 6  
    { 0x00, 0x01, 0x71, 0x09, 0x05, 0x03 }, // 7  
    { 0x00, 0x36, 0x49, 0x49, 0x49, 0x36 }, // 8  
    { 0x00, 0x06, 0x49, 0x49, 0x29, 0x1e }, // 9  
    { 0x00, 0x00, 0x36, 0x36, 0x00, 0x00 }, // :  
    { 0x00, 0x00, 0x56, 0x36, 0x00, 0x00 }, // ;  
    { 0x00, 0x08, 0x14, 0x22, 0x41, 0x00 }, // <  
    { 0x00, 0x14, 0x14, 0x14, 0x14, 0x14 } // =
```

```
{ 0x00, 0x00, 0x41, 0x22, 0x14, 0x08 }, // >
{ 0x00, 0x02, 0x01, 0x51, 0x09, 0x06 }, // ?
{ 0x00, 0x32, 0x49, 0x59, 0x51, 0x3E }, // @
{ 0x00, 0x7C, 0x12, 0x11, 0x12, 0x7C }, // A
{ 0x00, 0x7F, 0x49, 0x49, 0x49, 0x36 }, // B
{ 0x00, 0x3E, 0x41, 0x41, 0x41, 0x22 }, // C
{ 0x00, 0x7F, 0x41, 0x41, 0x22, 0x1C }, // D
{ 0x00, 0x7F, 0x49, 0x49, 0x49, 0x41 }, // E
{ 0x00, 0x7F, 0x09, 0x09, 0x09, 0x01 }, // F
{ 0x00, 0x3E, 0x41, 0x49, 0x49, 0x7A }, // G
{ 0x00, 0x7F, 0x08, 0x08, 0x08, 0x7F }, // H
{ 0x00, 0x00, 0x41, 0x7F, 0x41, 0x00 }, // I
{ 0x00, 0x20, 0x40, 0x41, 0x3F, 0x01 }, // J
{ 0x00, 0x7F, 0x08, 0x14, 0x22, 0x41 }, // K
{ 0x00, 0x7F, 0x40, 0x40, 0x40, 0x40 }, // L
{ 0x00, 0x7F, 0x02, 0x0C, 0x02, 0x7F }, // M
{ 0x00, 0x7F, 0x04, 0x08, 0x10, 0x7F }, // N
{ 0x00, 0x3E, 0x41, 0x41, 0x41, 0x3E }, // O
{ 0x00, 0x7F, 0x09, 0x09, 0x09, 0x06 }, // P
{ 0x00, 0x3E, 0x41, 0x51, 0x21, 0x5E }, // Q
{ 0x00, 0x7F, 0x09, 0x19, 0x29, 0x46 }, // R
{ 0x00, 0x46, 0x49, 0x49, 0x49, 0x31 }, // S
{ 0x00, 0x01, 0x01, 0x7F, 0x01, 0x01 }, // T
{ 0x00, 0x3F, 0x40, 0x40, 0x40, 0x3F }, // U
{ 0x00, 0x1F, 0x20, 0x40, 0x20, 0x1F }, // V
{ 0x00, 0x3F, 0x40, 0x38, 0x40, 0x3F }, // W
{ 0x00, 0x63, 0x14, 0x08, 0x14, 0x63 }, // X
{ 0x00, 0x07, 0x08, 0x70, 0x08, 0x07 }, // Y
{ 0x00, 0x61, 0x51, 0x49, 0x45, 0x43 }, // Z
{ 0x00, 0x00, 0x7F, 0x41, 0x41, 0x00 }, // [
{ 0x00, 0x55, 0x2A, 0x55, 0x2A, 0x55 }, // 55
{ 0x00, 0x00, 0x41, 0x41, 0x7F, 0x00 }, // ]
{ 0x00, 0x04, 0x02, 0x01, 0x02, 0x04 }, // ^
{ 0x00, 0x40, 0x40, 0x40, 0x40, 0x40 }, // -
{ 0x00, 0x00, 0x01, 0x02, 0x04, 0x00 }, // '
{ 0x00, 0x20, 0x54, 0x54, 0x54, 0x78 }, // a
{ 0x00, 0x7F, 0x48, 0x44, 0x44, 0x38 }, // b
{ 0x00, 0x38, 0x44, 0x44, 0x44, 0x20 }, // c
{ 0x00, 0x38, 0x44, 0x44, 0x48, 0x7F }, // d
{ 0x00, 0x38, 0x54, 0x54, 0x54, 0x18 }, // e
{ 0x00, 0x08, 0x7E, 0x09, 0x01, 0x02 }, // f
{ 0x00, 0x18, 0xA4, 0xA4, 0xA4, 0x7C }, // g
{ 0x00, 0x7F, 0x08, 0x04, 0x04, 0x78 }, // h
{ 0x00, 0x00, 0x44, 0x7D, 0x40, 0x00 }, // i
{ 0x00, 0x40, 0x80, 0x84, 0x7D, 0x00 }, // j
{ 0x00, 0x7F, 0x10, 0x28, 0x44, 0x00 }, // k
{ 0x00, 0x00, 0x41, 0x7F, 0x40, 0x00 }, // l
{ 0x00, 0x7C, 0x04, 0x18, 0x04, 0x78 }, // m
{ 0x00, 0x7C, 0x08, 0x04, 0x04, 0x78 }, // n
{ 0x00, 0x38, 0x44, 0x44, 0x44, 0x38 }, // o
{ 0x00, 0xFC, 0x24, 0x24, 0x24, 0x18 }, // p
{ 0x00, 0x18, 0x24, 0x24, 0x18, 0xFC }, // q
{ 0x00, 0x7C, 0x08, 0x04, 0x04, 0x08 }, // r
{ 0x00, 0x48, 0x54, 0x54, 0x54, 0x20 }, // s
{ 0x00, 0x04, 0x3F, 0x44, 0x40, 0x20 }, // t
{ 0x00, 0x3C, 0x40, 0x40, 0x20, 0x7C }, // u
{ 0x00, 0x1C, 0x20, 0x40, 0x20, 0x1C }, // v
{ 0x00, 0x3C, 0x40, 0x30, 0x40, 0x3C }, // w
{ 0x00, 0x44, 0x28, 0x10, 0x28, 0x44 }, // x
{ 0x00, 0x1C, 0xA0, 0xA0, 0xA0, 0x7C }, // y
```

```
{ 0x00, 0x44, 0x64, 0x54, 0x4C, 0x44 },    // z
{ 0x14, 0x14, 0x14, 0x14, 0x14 }      // horiz lines
};

void delay_1us(void)                      //delay 1us
{
    _delay_us(1);
}

void delay_1ms(void)                      //delay 1ms
{
    _delay_ms(1);
}

void delay_nms(uint8_t n)                 //delay nms
{
    for(uint8_t i; i<n; i++)
    {
        _delay_ms(1);
    }
}

void LCD_init(void)                       //LCD initialiseren
{
    DDRD |= 0xF1;                         //poorten als output zetten

    LCD_RST_clr;                         //reset pin laag
    delay_1us();
    LCD_RST_set;                         //reset pin hoog

    LCD_CE_clr;                          //chip enable laag maken
    delay_1us();
    LCD_CE_set;                          //chip enable hoog maken

    delay_1us();

    LCD_write_byte(0x21, 0);             // LCD mode instellen
    LCD_write_byte(0xc8, 0);             // bias instellen
    LCD_write_byte(0x06, 0);             // Temp-range instellen
    LCD_write_byte(0x13, 0);             // 1:48
    LCD_write_byte(0x20, 0);             // bias comando gebruiken
    LCD_clear();                        // Lcd leegmaken
    LCD_write_byte(0x0c, 0);             // lcd mode instellen op normal

    LCD_CE_clr;                         //chip enable pin laag maken
}

void LCD_clear(void)                     // lcd leegmaken
{
    unsigned int i;

    LCD_write_byte(0x0c, 0);
    LCD_write_byte(0x80, 0);

    for (i=0; i<504; i++)
    {
```

```
    LCD_write_byte(0, 1);
}

/*
Deze functie zet het coordinaat van de pointer in het lcd

input parameter -X coordinaat 83 max
          -Y coordinaat  #5 max

*/
void LCD_set_XY(unsigned char X, unsigned char Y)
{
    LCD_write_byte(0x40 | Y, 0);           // kolommen
    LCD_write_byte(0x80 | X, 0);           // rijen
}

/*
Deze functie stuurt 1 karakter naar het lcd
*/
void LCD_write_char(unsigned char c)
{
    unsigned char line;

    c -= 32;                //asci codes omzetten naar de lookup table

    for (line=0; line<6; line++)
        LCD_write_byte(font6x8[c][line], 1);
}

/*
LCD_write_String : een string naar het scherm sturen
*/
void LCD_write_string(unsigned char X,unsigned char Y,char *s)
{
    LCD_set_XY(X,Y);                  //coordinaat juistzetten
    while (*s)
    {
        LCD_write_char(*s);
        s++;
    }
}

/*
Deze functie schrijft een data byte naar het lcd, comando argument is om te
kiezen tussen comando of data register. 0=cmd, 1=data
*/
void LCD_write_byte(unsigned char dat, unsigned char command)
{
    unsigned char i;
    LCD_CE_clr;

    if (command == 0)
        LCD_DC_clr;
    else
        LCD_DC_set;

    for(i=0;i<8;i++)
    {
```

```
    if(dat&0x80)
        SDIN_set;
    else
        SDIN_clr;
    SCLK_clr;
    dat = dat << 1;
    SCLK_set;
}
LCD_CE_set;
}

/*
Deze functie stuurt een getal naar het scherm
*/
void LCD_write_int(unsigned char X,unsigned char Y,uint8_t getal)
{
    char string[4];
    itoa(getal,string,10);
    LCD_write_string(X,Y,string);
}
```

```
/*
 * Rtc.h
 *
 * Created: 27/03/2015 10:44:54
 * Author: Olivier
 */

/* Hier staan de functies beschreven voor de i2c en rtc

    void init_RTC(void)
    void RTC_write(uint8_t reg, uint8_t data)
    uint8_t RTC_Read(uint8_t reg)

    void BCD_To_ASCII(unsigned char bcd_value, char * p_ascii_text)

*/

void init_RTC(void);
void RTC_write(uint8_t reg, uint8_t data);
uint8_t RTC_Read(uint8_t reg);
void get_uur(void);
uint8_t BCDToDecimal (uint8_t);

void BCD_To_ASCII(unsigned char bcd_value, char * p_ascii_text);

char uurRtc[2];
char minRtc[2];
uint8_t uurtc,mintc;

//hier de functies voor het rtc te besturen

void init_RTC(void)
{
    RTC_write(0b00000000,0x00);      //seondenregister op 0
    RTC_write(0x00000001,0x00);    //minutenregister op 0
    RTC_write(0x00000010,0x00);    //urenregister op 0
}

void RTC_write(uint8_t reg, uint8_t data)
{
    i2c_start();                  //startconditie sturen
    i2c_send(0b11010000);        // rtc aanspreken in write mode

    i2c_send(reg);               //het juiste register aanspreken
    i2c_send(data);              //de data sturen

    i2c_stop();                  //stopconditie sturen
}

uint8_t RTC_Read(uint8_t reg)
{
    uint8_t data;                //variabele om data te ontvangen

    i2c_start();                  //startconditie sturen
    i2c_send(0b11010000);        //rtc aanspreken in write mode
    i2c_send(reg);               //het juiste register aanspreken
```

```
i2c_stop();
i2c_start();           //nieuwe startconditie
i2c_send(0b11010001); //opnieuw aanspreken rtc maar nu in read mode

data = i2c_receive(0); //register uitlezen en data opslaan

i2c_stop();           //stopconditie sturen

return data;          //data returnen

}

void get_uur(void)
{
    uint8_t uren = 0;
    uint8_t minuten = 0;

    minuten = RTC_Read(0x01);      //minuten lezen
    uren = RTC_Read(0x02);        //uren uitlezen

    BCD_To_ASCII(uren,uurRtc);
    BCD_To_ASCII(minuten,minRtc); //omzetten naar string voor lcd

    uurtc = BCDToDecimal(uren);   //ook omzetten naar int voor de
                                  berekeningen
    mintc = BCDToDecimal(minuten);
}

//deze functie zet een bcd getal om naar een string

void BCD_To_ASCII(unsigned char bcd_value, char * p_ascii_text)
{
    //-----
    // BCD contains digits 0 .. 9 in the binary nibbles
    //-----
    *p_ascii_text++ = (bcd_value >> 4) + '0';
    *p_ascii_text++ = (bcd_value & 0x0f) + '0';
    *p_ascii_text = '\0';
    return;
}

uint8_t BCDToDecimal (uint8_t bcdByte)
{
    return (((bcdByte & 0xF0) >> 4) * 10) + (bcdByte & 0x0F);
}
```

```
/*
 * rotary.h
 *
 * Created: 27/03/2015 11:01:30
 * Author: Olivier
 */

#define ROTPORT PORTC
#define ROTDDR DDRC          //de poort waar de rotary encoder op aangesloten is
#define ROTPIN PINC

#define ROTPA PC0
#define ROTPB PC1          //rotary encoder pinnen
#define ROTBUTTON    PC2

#define ROTA !((1<<ROTPA)&ROTPIN)
#define ROTB !((1<<ROTPB)&ROTPIN)      //macro's
#define ROTCLICK !((1<<ROTPUTTON)&ROTPIN)

void RotaryInit(void);
void RotaryCheckStatus(void);        //functies
uint8_t RotaryGetStatus(void);
void RotaryResetStatus(void);

uint8_t rotarystatus;           //variabele om stand van rotary uit te lezen

void RotaryInit(void)
{
    //pinnen als input zetten
    ROTDDR &= ~((1<<ROTPA)|(1<<ROTPB)|(1<<ROTPUTTON));
}

void RotaryCheckStatus(void)
{
    //Lezen van rotary encoder en knop
    //Kijken of er links gedraaid word
    if(ROTA & (!ROTB)){
        while (bit_is_clear(ROTPIN, ROTPA));
        if (ROTB) rotarystatus=1;    //als links draaiend, op 1 zetten

        //check if rotation is right
    }else if(ROTB & (!ROTA)){
        while(bit_is_clear(ROTPIN, ROTPB));
        if (ROTA)
            rotarystatus=2;      //als rechts draaiend, op 2 zetten
        }else if (ROTA & ROTB){
            while (bit_is_clear(ROTPIN, ROTPA));
            if (ROTB)
                rotarystatus=1;    //als links, op 1 zetten
            else rotarystatus=2;  //als recht op 2
    }
}

uint8_t RotaryGetStatus(void)
{
    return rotarystatus;        //status van de encoder opvragen
```

```
}

void RotaryResetStatus(void)
{
    rotarystatus=0;           //register resetten
}

void Timer0_Start(void)
{
    TCCR0B|=(1<<CS01); //prescaller 8~122 interrupts/s
    TIMSK0|=(1<<TOIE0); //Enable Timer0 Overflow interrupts
    sei();
}
```

```
/*
 * knoppen.h
 *
 * Created: 28/03/2015 14:40:45
 * Author: Olivier
 */

/*
knop layout:          knop 1          knop 3
                      knop 2          knop 4
*/

uint8_t knoppen[3];           //een array om de knoppen in te lezen

void lees_knoppen(void)      //leest de knoppen uit, en zet de waarde in de
    globale array "knoppen[3]"
{
    uint8_t getal;

    i2c_start();
    i2c_send(0b01110001);        //expander aanspreken
    getal = i2c_receive(1);      //byte lezen

    getal = getal << 4;         //nibble switchen

    for (uint8_t i=0; i<4; i++)
    {
        uint8_t kopie = getal;
        kopie = kopie <<i;       //i keer doorschuiven afhankelijk van de knop
        kopie = kopie & 0b10000000; //enkel hoogste bit overhouden

        if (kopie == 0b10000000)
        {
            switch(i)
            {
                case 0: knoppen[0] = 0; break;
                case 1: knoppen[3] = 0; break;           //juiste plaats in de array
                case 2: knoppen[1] = 0; break;
                case 3: knoppen[2] = 0; break;
            }
        }
        else
        {
            switch(i)
            {
                case 0: knoppen[0] = 1; break;
                case 1: knoppen[3] = 1; break;           //juiste plaats in de array
                case 2: knoppen[1] = 1; break;
                case 3: knoppen[2] = 1; break;
            }
        }
    }
}
```

```
uint8_t datlen = 4;

/* Copyright (c) 2007 Stefan Engelke <mailto:stefanengelke.de>
   Deze defines zijn gedownload van internet om het programma makkelijker te
   maken
 */

/* Memory Map */

#define CONFIG      0x00
#define EN_AA       0x01
#define EN_RXADDR   0x02
#define SETUP_AW    0x03
#define SETUP_RETR  0x04
#define RF_CH       0x05
#define RF_SETUP    0x06
#define STATUS      0x07
#define OBSERVE_TX 0x08
#define CD          0x09
#define RX_ADDR_P0  0x0A
#define RX_ADDR_P1  0x0B
#define RX_ADDR_P2  0x0C
#define RX_ADDR_P3  0x0D
#define RX_ADDR_P4  0x0E
#define RX_ADDR_P5  0x0F
#define TX_ADDR     0x10
#define RX_PW_P0    0x11
#define RX_PW_P1    0x12
#define RX_PW_P2    0x13
#define RX_PW_P3    0x14
#define RX_PW_P4    0x15
#define RX_PW_P5    0x16
#define FIFO_STATUS 0x17
```

```
#define DYNPD      0x1C
#define FEATURE    0x1D

/* Bit Mnemonics */

#define MASK_RX_DR  6
#define MASK_TX_DS  5
#define MASK_MAX_RT 4
#define EN_CRC      3
#define CRC0        2
#define PWR_UP      1
#define PRIM_RX     0
#define ENAA_P5     5
#define ENAA_P4     4
#define ENAA_P3     3
#define ENAA_P2     2
#define ENAA_P1     1
#define ENAA_P0     0
#define ERX_P5     5
#define ERX_P4     4
#define ERX_P3     3
#define ERX_P2     2
#define ERX_P1     1
#define ERX_P0     0
#define AW          0
#define ARD         4
#define ARC         0
#define PLL_LOCK   4
#define RF_DR      3
#define RF_PWR     6
```

```
#define RX_DR      6
#define TX_DS      5
#define MAX_RT      4
#define RX_P_NO      1
#define TX_FULL      0
#define PLOS_CNT      4
#define ARC_CNT      0
#define TX_REUSE      6
#define FIFO_FULL      5
#define TX_EMPTY      4
#define RX_FULL      1
#define RX_EMPTY      0
#define DPL_P5      5
#define DPL_P4      4
#define DPL_P3      3
#define DPL_P2      2
#define DPL_P1      1
#define DPL_P0      0
#define EN_DPL      2
#define EN_ACK_PAY    1
#define EN_DYN_ACK    0
```

```
/* Instruction Mnemonics */
#define R_REGISTER    0x00
#define W_REGISTER    0x20
#define REGISTER_MASK 0x1F
#define ACTIVATE      0x50
#define R_RX_PL_WID   0x60
#define R_RX_PAYLOAD   0x61
#define W_TX_PAYLOAD   0xA0
```

```
#define W_ACK_PAYLOAD 0xA8
#define FLUSH_TX      0xE1
#define FLUSH_RX      0xE2
#define REUSE_TX_PL   0xE3
#define NOP           0xFF

/* Non-P omissions */

#define LNA_HCURR    0

/* P model memory Map */
#define RPD          0x09

/* P model bit Mnemonics */
#define RF_DR_LOW    5
#define RF_DR_HIGH   3
#define RF_PWR_LOW   1
#define RF_PWR_HIGH  2

/* Vanaf hier start de eigen gemaakte code */

/* De pinout voor de nrf zit op poortB voor de spi-bus
 - CE --> PB1
 - CSN --> PB2
 - MOSI --> PB3
 - MISO --> PB4
 - SCK --> PB5
 - INT --> PD2 (INT0)

*/
/* Een paar defines om de code makkelijker te maken*/
#define BIT(x) (1 << (x))
#define SETBITS(x,y) ((x) |= (y))
#define CLEARBITS(x,y) ((x) &= (~y)))
#define SETBIT(x,y) SETBITS((x), (BIT((y))))
#define CLEARBIT(x,y) CLEARBITS ((x), (BIT((y))))
```

```
#define W 1      //write is 1
#define R 0      //read is 0

// hier staan de functie declaraties

void InitSPI(void);
uint8_t WriteByteSPI(uint8_t);
uint8_t GetReg(uint8_t);
uint8_t *WriteToNrf(uint8_t, uint8_t, uint8_t*, uint8_t);
void nrf24l01_init_TX(void);
void nrf24l01_init_RX(void);
void transmit_payload(uint8_t * buff);

//de functies zelf

void InitSPI(void)
{
    //SCK,MOSI,CSN en ce als output zetten
    DDRB |= (1<<DDB5) | (1<<DDB3) | (1<<DDB2) | (1<<DDB1);

    //SPI aanzetten in mastermode en klok= fck/16.
    SPCR |= (1<<SPE) | (1<<MSTR);

    SETBIT(PORTB,2);    //CSN hoogzetten om mee te starten
    CLEARBIT(PORTB,1); //ce laag om mee te starten
}

uint8_t WriteByteSPI(uint8_t Dbyte)
{
    //byte laden in dataregister
    SPDR = Dbyte;

    //wachten tot versturen voltooid is
    while (!(SPSR & (1<<SPIF)));

    //returnen wat er terugkomt uit de spi-bus
    return SPDR;
}

uint8_t GetReg(uint8_t reg)
{
    _delay_us(20); //zeker zijn dat vorige communicatie gedaan is
    CLEARBIT(PORTB,2); //csn laag zetten, nrf wacht nu op comando
    _delay_us(20);
    WriteByteSPI(R_REGISTER + reg); //R_register = nrf in read mode zetten,
    // "reg" = het register dat wordt teruggestuurd
    _delay_us(20);
    reg = WriteByteSPI(NOP); //dummy byte sturen om data terug binnen
    // te lezen uit "reg" register
    _delay_us(20);
    SETBIT(PORTB,2); //csn hterug hoog, nrf doet terug niks

    return reg; //return van het gelezen register
}

uint8_t *WriteToNrf(uint8_t ReadWrite, uint8_t reg, uint8_t *pack, uint8_t
aantpack)
{
    //Readwrite" = "W" of "R", reg = register om naar te sturen, pack = array
```

```

        met de data, aantalpack = aantal data in de array

    if (ReadWrite == W)      //als het w is, willen we schrijven
    {
        reg = W_REGISTER + reg;      //we voegen de write bit toe aan het
                                       register
    }

//we maken een array om op het einde de data te returnen
static uint8_t ret[32];

_delay_us(10);      //er zeker van zijn dat alle communicatie gedaan is
CLEARBIT(PORTB,2); //csn laag zetten, nrf luisterd nu
_delay_us(10);
WriteByteSPI(reg); //nrf in lees of schrijfmode zetten voor het "reg"
                   register
_delay_us(10);

for(uint8_t i = 0; i<aantpack; i++)
{
    if (ReadWrite == R && reg != W_TX_PAYLOAD) //read comando?
    {                                              //w_tx_payload is een
        uitzondering, deze kan niet het w bevatten
        ret[i] = WriteByteSPI(NOP);           //dummy bytes sturen om data
                                               binnen te halen
        _delay_us(10);
    }
    else
    {
        WriteByteSPI(pack[i]);           //stuur de data naar de nrf 1 per 1
        _delay_us(10);
    }
}

SETBIT(PORTB,2);      //csn terug hoog, nrf doet nu niks meer
return ret;          //de array terugsturen

}

void transmit_Data(uint8_t * buff)
{
    WriteToNrf(R, FLUSH_TX, buff, 0); //alle oude data verwijderen uit de
                                         chip met flush
    WriteToNrf(R, W_TX_PAYLOAD, buff, datlen); //stuur de data in de buffer
                                                naar de nrf

    nrf_int_enable();      //interrupt opstarten om te kijken of transmissie
                           kan voltooien

    _delay_ms(20);         //even wachten zodat de data er zeker inzit
    SETBIT(PORTB, 1);     //ce hoogzetten = transmit data
    _delay_us(20);         //even wachten
    CLEARBIT(PORTB,1);   //ce terug naar omlaag -> transmissie stoppen
    _delay_ms(20);         //even wachten voor einde

}

void nrf_reset(void)      //reset alle interrupt messages in de nrf
{

```

```
_delay_us(20);
CLEARBIT(PORTB, 2); //csn laag maken
_delay_us(20);
WriteByteSPI(W_REGISTER + STATUS); //write to status reg
_delay_us(20);
WriteByteSPI(0x70); //reset alle irq in status reg
_delay_us(20);
SETBIT(PORTB,2); //csn terug hoog
}

void INT0_interrupt_init(void) //interrupt initialiseren
{
    DDRD &= ~(1<<DDD2); //INT0 als input zetten

    EICRA |= (1<<ISC01); //int falling edge
    EICRA &= ~(1<<ISC00); //falling edge

    EIMSK |= (1<<INT0); //Int0 enable
}

void nrf_int_enable(void)
{
    EIMSK |= (1<<INT0); //interrupt aan
}

void nrf_int_disable(void)
{
    EIMSK &= ~(1<<INT0); //interrupt uitzetten
}

void recieve_Data(void)
{
    WriteToNrf(R, FLUSH_RX, NOP, 0); //alle oude data verwijderen uit de
        chip met flush
    nrf_int_enable(); //interrupt inschakelen om te kijken of er
        nieuwe data is

    SETBIT(PORTB,1); //luisteren naar data
    _delay_ms(1000); //1s luisten
    CLEARBIT(PORTB,1); //stop met luisten

    nrf_int_disable(); //als er nu geen interrupt is, is er geen
        data, dus int terug uit
}

void nrf24l01_init_RX(void) //functie voor het initialiseren van de nrf in
    receiver mode
{
    _delay_ms(100); //wachten om er zeker van te zijn dat alle communicatie
        gedaan is

    uint8_t setup[5]; //een array om de writeToNrf functie te gebruiken

    //EN_AA = auto ack uitzetten voor alle pipes
    setup[0] = 0x00; //de registerwaarde zetten
    WriteToNrf(W,EN_AA, setup,1); //data sturen naar register

    //SETUP_RETR stelt het aantal pogingen in bij een mislukte transmissie
    setup[0] = 0x2f; //0b0010 1111: elke 750us opnieuw proberen bij
        mislukking, 15keer opnieuw proberen
}
```

```
WriteToNrf(W, SETUP_RETR, setup, 1);

//aantal datapipes instellen hier enkel dp0
setup[0] = 0x01;
WriteToNrf(W, EN_RXADDR, setup, 1); //data pipe 0 aanzetten

//Rf-adres breedte instellen (1-5 in bytes)
setup[0] = 0b00000011; //0b0000 0011 = 5byte adres
WriteToNrf(W, SETUP_AW, setup, 1);

//rf kanaal instellen - frequentie 2.4 - 2.527ghz
setup[0] = 0b01110000; //kanaal 112, is een legaal kanaal buiten het
bereik van bluethoot en wifi
WriteToNrf(W, RF_CH, setup, 1);

//RF_setup power down mode en data speed
setup[0] = 0x07;
WriteToNrf(W, RF_SETUP, setup, 1);

//RX RF_adress setup met 5 bytes
for (int i=0; i<5; i++)
{
    setup[i] = 0x12; //0x12 x 5 is het adres, het moet hetzelfde zijn
    op de reciever-transmitter
}
WriteToNrf(W, RX_ADDR_P0, setup, 5); //we hebben datapipe0 gekozen, dus
geven we deze een adres

//TX RF_adr setup met 5 bytes
for (int i=0; i<5; i++)
{
    setup[i] = 0x12; //adres hetzelfde op reciever en transmitter
}
WriteToNrf(W, TX_ADDR, setup, 5);

//breedte van de paketten instellen
setup[0] = datlen; //3 bytes sturen per transmittie
WriteToNrf(W, RX_PW_P0, setup, 1);

//Config register setup- hier kiezen we reciever of transmitter mode
setup[0] = 0b00111111; //reciever mode, enkel interupt voor data
ontvangen, power-up mode
WriteToNrf(W, CONFIG, setup, 1);

//het device heeft 1.5ms nodig om in standby te gaan
_delay_ms(100);
}

void nrf24l01_init_TX(void) //funcite voor het initialiseren van de nrf in
transmitter mode
{
    _delay_ms(100);

    uint8_t setup[5]; //een array om de writeToNrf functie te gebruiken

    //EN_AA = auto ack uitzetten voor alle datapipes
    setup[0] = 0x00; //de registerwaarde zetten
    WriteToNrf(W, EN_AA, setup, 1); //data sturen

    //SETUP_RETR stelt het aantal pogingen in bij een mislukte transmittie
```

```
setup[0] = 0x2f;           //0b0010 1111: elke 750us opnieuw proberen bij
                           mislukking, 15keer opnieuw proberen
WriteToNrf(W,SETUP_RETR, setup, 1);

//aantal datapipes instellen hier enkel dp0
setup[0] = 0x01;
WriteToNrf(W,EN_RXADDR, setup, 1); //data pipe 0 aanzetten

//Rf-adres breedte instellen (1-5 in bytes)
setup[0] = 0b00000011;        //0b0000 0011 = 5byte adres
WriteToNrf(W,SETUP_AW, setup, 1);

//rf kanaal instellen - frequentie 2.4 – 2.527ghz
setup[0] = 0b01110000;        //kanaal 112 – is een legaal kanaal buiten het
                           berijk van wifi
WriteToNrf(W,RF_CH, setup, 1);

//RF_setup power down mode en data speed
setup[0] = 0x07;
WriteToNrf(W,RF_SETUP, setup, 1);

//RX RF_adress setup met 5 bytes
for (int i=0; i<5; i++)
{
    setup[i] = 0x12;          //0x12 x 5 is het adres, het moet hetzelfde zijn
                           op de reciever-transmitter
}
WriteToNrf(W,RX_ADDR_P0, setup, 5); //we hebben datapipe0 gekozen, dus
                           geven we deze een adres

//TX RF_adr setup met 5 bytes
for (int i=0; i<5; i++)
{
    setup[i] = 0x12;          //adres hetzelfde op reciever en transmitter
}
WriteToNrf(W, TX_ADDR, setup, 5);

//breedte van de paketten instellen
setup[0] = datlen; //3 bytes sturen per transmittie
WriteToNrf(W, RX_PW_P0, setup, 1);

//Config register setup- hier kiezen we reciever of transmitter mode
setup[0] = 0b01011110;      //transmitter mode, power-up, enkel interrupt
                           als transmittie gelukt is
WriteToNrf(W, CONFIG, setup, 1);

//het device heeft 1.5ms nodig om in standby te gaan
_delay_ms(100);
}
```

Bijlage 2: Broncode Temperatuursensor

- Main.c
- Slapen.h
- I2c_htu21d.h

```
/*
 * nrf_transm.c
 *
 * Created: 21/03/2015 15:55:18
 * Author: Olivier
 */

#include <avr/io.h>
#define F_CPU 1000000
#include <util/delay.h>
#include <avr/interrupt.h>

#include "nRF24L01.h"
#include "slapen.h"
#include "i2c_htu21d.h"

int main(void)
{
    InitSPI();
    init_sleep();
    nrf24l01_init_TX();
    INT0_interrupt_init();

    uint8_t data[4] = {0,0,0,0};           //de var om data te verzenden via nrf
    uint8_t temp1,temp2;                 //2 var om temp in op te slaan
    uint8_t cont;                      //de controlevar
    uint8_t batterij;                  //de batterijvar

    //temp1 = 21;                      //testwaardes
    //temp2 = 7;
    batterij = 3;

    float temperatuur;                //de var om de gemeten temp in op te slaan

    i2c_Init();                       //Init i2c en htu
    initHTU21D();

    while(1)
    {
        temperatuur = read_temp();      //temperatuur meten

        #define KOMMA (10)
        int temp1 = (int)temperatuur;    // opsplitsen van float in 2
                                         integers
        int temp2 = ((int)(temperatuur*KOMMA)%KOMMA);

        cont = (~ temp1);

        data[0] = temp1;
        data[1] = temp2;               //data op juiste manier in datapaket steken
        data[2] = ~ data[0];
        if (batterij < 2)
        {
            data[3] = 0;
        }
    }
}
```

```
else
{
    data[3] = 1;
}

nrf_reset();
transmit_Data(data);           //data versturen
_delay_ms(500);

for (uint16_t i=0; i<500; i++)
{
    start_slaap();
}
}

ISR(INT0_vect)           //interrupt routine als data in de lucht is
{
    nrf_int_disable();
    CLEARBIT(PORTB,1);      //ce laag maken
}

ISR (TIMER2_OVF_vect)       //interrupt vector voor de slaapfunctie;
{
    stop_slaap();
}
```

```
/*
 * slapen.h
 *
 * Created: 22/04/2015 15:51:08
 * Author: Olivier
 */

#include <avr/interrupt.h>
#include <avr/sleep.h>

/* Deze header file bevat een aantal functies om een avr in slaap mode te
zetten. De slaap mode die hier geactiveerd
word, is power-save, omdat deze gewekt kan worden met timer2. Timer 2 is
 ingesteld om om de 2ms een overflow te
genereren.

De te gebruiken functies zijn: - init_sleep();
                                - start_slaap();           //cpu gaat slapen
                                              en wordt wakker door t2
                                - stop_slaap();
```

Er moet ook een interruptroutine opgestart worden:

```
ISR (TIMER2_OVF_vect)
{
    stop_slaap();
}

void init_timer2(void)      // timer initialiseren om het slapen te kunnen
    onderbreken
{
    TCCR2A = 0x00;          //geen compare match
    TIMSK2 |= (1<<TOIE2); //interrupt bij overflow
    sei();
}

void timer2_start(void)     //start de timer op
{
    TCNT2 = 0x00;           //timer leegmaken
    TCCR2B |= (1<<CS21); //prescaler op 8
}

void timer2_stop(void)      //stop de timer
{
    TCCR2B = 0x00;           //prescaler op oneindig, dus timer uit
}

//vanaf hier volgen de slaapfuncties

void init_sleep(void)
{
    init_timer2();          //timer 2 initialiseren
    SMCR |= (1<<SM1) | (1<<SM0); //power-save mode
        inschakelen(datasheet)
}
```

```
void start_slaap(void)          //start de slaapmode op, het stopt als timer2 een
    overflow geeft
{
    SMCR |= (1<<SE);          //slaap mode activeren door se-bit hoog te zetten
    timer2_start();              //timer 2 opstarten
    sleep_cpu();                //slaap instructie uitvoeren, nu gaan we slapen
}

void stop_slaap(void)           //timer 2 stoppen
{
    SMCR &= ~(1<<SE);         //slaap mode uitzetten door se-bit laag te maken
    timer2_stop();              //timer 2 wordt gestopt
}
```

```
/* Deze header file bevat een aantal functies voor bitbang-i2c en de benodigde
   functies om te temp uit
een htu21d uit te lezen

void i2c_Init(void)
void i2c_Start(void)
void i2c_Stop(void)
unsigned char i2c_Write(uint8_t c)
uint8_t i2c_Read(unsigned char ack)
void initHTU21D(void)
float read_temp(void)

*/

// Poorten voor de i2c
#define I2C_DDR DDRD
#define I2C_PIN PIND
#define I2C_PORT PORTD

// Pinnen die gebruikt worden voor de klok en de data
#define I2C_CLK 0 //contact, klok, geel
#define I2C_DAT 1 //relais, data, groen

#define I2C_DATA_HI()\nI2C_DDR &= ~ (1 << I2C_DAT);\nI2C_PORT |= (1 << I2C_DAT);\n#define I2C_DATA_LO()\nI2C_DDR |= (1 << I2C_DAT);\nI2C_PORT &= ~ (1 << I2C_DAT);

#define I2C_CLOCK_HI()\nI2C_DDR &= ~ (1 << I2C_CLK);\nI2C_PORT |= (1 << I2C_CLK);\n#define I2C_CLOCK_LO()\nI2C_DDR |= (1 << I2C_CLK);\nI2C_PORT &= ~ (1 << I2C_CLK);

void delay(uint8_t num)      //voor de delay
{
    _delay_ms(1);
}

void i2c_WriteBit(unsigned char c)      //1 bit uiststyren
{
    if (c > 0)                      //data 0 of 1?
    {
        I2C_DATA_HI();
    }
    else
    {
        I2C_DATA_LO();
    }

    I2C_CLOCK_HI();      //klokpuls maken
    delay(1);

    I2C_CLOCK_LO();
}
```

```
delay(1);

if (c > 0)           //data terug laag maken
{
    I2C_DATA_L0();
}

delay(1);
}

unsigned char i2c_ReadBit(void)      //1 bit binnenhalen
{
    I2C_DATA_HI();

    I2C_CLOCK_HI();           //klokpuls maken
    delay(1);

    unsigned char c = I2C_PIN;        //data lezen

    I2C_CLOCK_L0();
    delay(1);

    return (c >> I2C_DAT) & 1;      //Data returnen
}

// Initialiseren van de bitbang-i2c
//
void i2c_Init(void)
{
    I2C_PORT &= ~ ((1 << I2C_DAT) | (1 << I2C_CLK));      //poorten juistzetten

    I2C_CLOCK_HI();
    I2C_DATA_HI();        //startconditie

    delay(1);
}

// een startconditie sturen
void i2c_Start(void)
{
    // Bijde tegelijk op 1 zetten
    I2C_DDR &= ~ ((1 << I2C_DAT) | (1 << I2C_CLK));
    delay(1);

    I2C_DATA_L0();        //eerst data laagmaken, dan de clok
    delay(1);            //dat is een startconditie

    I2C_CLOCK_L0();
    delay(1);
}

// Stopconditie sturen
void i2c_Stop(void)
{
    I2C_CLOCK_HI();       //eerst clock naar omhoog, dan pas data
    delay(1);

    I2C_DATA_HI();
    delay(1);
}
```

```
// Een byte sturen naar de slave
unsigned char i2c_Write(uint8_t c)
{
    for (char i = 0; i < 8; i++)
    {
        i2c_WriteBit(c & 128);      //bit per bit versturen

        c <<= 1;
    }
    if (i2c_ReadBit() == 1)      //kijken of er een ack of een nack volgt
    {
        return 0;                //als bit is 1, was er een nack
    }
    else
    {
        return 1;                //als bit is 0, was er een ack
    }
}

// Een byte lezen van de slave
// 
uint8_t i2c_Read(unsigned char ack)
{
    uint8_t res = 0;            // een var voor het resultaat

    for (char i = 0; i < 8; i++)
    {
        res <<= 1;              //bit per bit lezen
        res |= i2c_ReadBit();
    }

    if (ack > 0)              //kijken of er een ack of een nack gestuurt moet
        worden
    {
        i2c_WriteBit(0);
    }
    else
    {
        i2c_WriteBit(1);
    }

    delay(1);

    return res;
}

void initHTU21D(void)          //deze functie initialiseert de Htu21d-
    temperatuursensor in 14-bit mode
{
    uint8_t gegevens;         //een van om tijdelijk iets op te slaan

    i2c_Start();
    i2c_Write(0x80);           //adres in write mode
    i2c_Write(0xFE);           //soft reset
    i2c_Stop();

    i2c_Start();
    i2c_Write(0x80);           //adres met schrijfcomando
```

```
i2c_Write(0xe7);           //lees user-register comando

i2c_Start();                //lees mode
i2c_Write(0x81);            //ontvangen met nack
gegevens = i2c_Read(0);      //14-bitmode aanzetten
gegevens |= 0x00000001;

i2c_Start();                //schrijf mode
i2c_Write(0x80);            //schrijven in user-register
i2c_Write(gegevens);         //register juistzetten

i2c_Stop();                 

}

float read_temp(void)        //deze functie leest de meting uit, en berekend de
    temp in float vorm
{
    uint8_t getal1,getal2;
    uint16_t meting;
    float temperatuur;

    i2c_Start();                //adres in write mode
    i2c_Write(0x80);            //temperatuurmeting in non-hold mode
    i2c_Start();

    while (i2c_Write(0x81) == 0) //kijken of de data al beschikbaar is,
        als niet blijven kijken
    {
        i2c_Start();
    }

    getal2 = i2c_Read(1);        //msb uitlezen
    getal1 = i2c_Read(1);        //lsb-uitlezen
    i2c_Read(0);                //check uitlezen maar niks mee doen
    i2c_Stop();

    getal1 = getal1 & 0b11111100; //laagste 2 bits zijn statusbits, dus
        maskeren
    meting = getal1 | (getal2 << 8); //msb en lsb samenvoegen tot 16-bit
        getal

    float deling = (float)meting / (float) 65536; //de deling berekenen,
        met typecast

    temperatuur = -46.85 + (175.72*deling); //de temperatuur berekenen

    return temperatuur;          //return temp
}
```

Bijlage 3: Broncode Relais-module

- main.c

```
/*
 * relais_moduleV1.c
 *
 * Created: 24/04/2015 8:21:09
 * Author: Administrator
 */

#include <avr/io.h>
#define F_CPU 1000000
#include <util/delay.h>
#include <avr/interrupt.h>

#include "nRF24L01.h"

uint8_t * recData = 0;           //variabele om nrf data te ontvangen

void verander_adres(void);

int main(void)
{
    InitSPI();
    nrf24l01_init_RX();
    verander_adres();           //het adres veranderen
    INT0_interrupt_init();
    sei();

    DDRD |= (1<<PD1);         //als output schakelen, om de led aan te sturen
    SETBIT(PORTD,1);
    _delay_ms(1000);
    CLEARBIT(PORTD,1);
    _delay_ms(1000);

    while(1)
    {
        nrf_reset();           //nrf ontvangstregister leegmaken
        recieve_Data();         //kijken of er data is

        if (recData[0] != 0)
        {
            if (recData[0] == 0x20)     //dan is de data voor hier
            {
                if (recData[1] == 0x30)   //als 0x30 dan moet de verwarming
                    aan
                {
                    SETBIT(PORTD,1);
                }
                else
                {
                    CLEARBIT(PORTD,1);
                }
            }
        }
    }
}

ISR(INT0_vect)      //interruptroutine als er data ontvangen is
{
    nrf_int_disable();       //interrupts uitzetten, want de data is
    ontvangen
```

```
CLEARBIT(PORTB,1);           //ce laag maken -> stop met luisteren naar data

recData = WriteToNrf(R,R_RX_PAYLOAD, recData ,datlen); //data binnenhalen
    uit recieve register
}

void verander_adres(void)      //deze functie veranderd het adres van de
    transciever om meerdere comunicaties te doen
{
    uint8_t setup[5];

    //RX RF_adress setup met 5 bytes
    for (int i=0; i<5; i++)
    {
        setup[i] = 0x14;      //0x12 x 5 is het adres, het moet hetzelfde zijn
        op de reciever-transmitter
    }
    WriteToNrf(W,RX_ADDR_P0, setup, 5); //we hebben datapipe0 gekozen, dus
        geven we deze een adres

    //TX RF_adr setup met 5 bytes
    for (int i=0; i<5; i++)
    {
        setup[i] = 0x14;      //adres hetzelfde op reciever en transmitter
    }
    WriteToNrf(W, TX_ADDR, setup, 5);
}
```

Bijlage 4: Datasheet features

- Maxim DS1307 Rtc
- Philips PCD8544 lcd-controller
- Atmel Atmega 328p microcontroller
- Philips PCF8574 i2c-I/O-expander
- Nordic nRF24L01+ Rf-controller
- HTU21D temperatuursensor
- Fairchild LM 7805 spanningsregelaar
- Texas Instruments LM1117 spanningsregelaar



maxim
integrated™

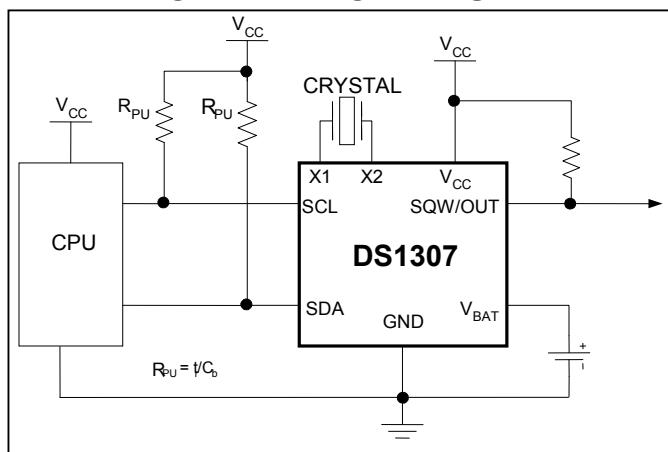
DS1307

64 x 8, Serial, I²C Real-Time Clock

GENERAL DESCRIPTION

The DS1307 serial real-time clock (RTC) is a low-power, full binary-coded decimal (BCD) clock/calendar plus 56 bytes of NV SRAM. Address and data are transferred serially through an I²C, bidirectional bus. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with AM/PM indicator. The DS1307 has a built-in power-sense circuit that detects power failures and automatically switches to the backup supply. Timekeeping operation continues while the part operates from the backup supply.

TYPICAL OPERATING CIRCUIT



ORDERING INFORMATION

PART	TEMP RANGE	VOLTAGE (V)	PIN-PACKAGE	TOP MARK*
DS1307+	0°C to +70°C	5.0	8 PDIP (300 mils)	DS1307
DS1307N+	-40°C to +85°C	5.0	8 PDIP (300 mils)	DS1307N
DS1307Z+	0°C to +70°C	5.0	8 SO (150 mils)	DS1307
DS1307ZN+	-40°C to +85°C	5.0	8 SO (150 mils)	DS1307N
DS1307Z+T&R	0°C to +70°C	5.0	8 SO (150 mils) Tape and Reel	DS1307
DS1307ZN+T&R	-40°C to +85°C	5.0	8 SO (150 mils) Tape and Reel	DS1307N

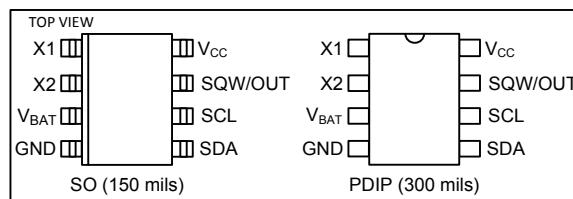
*Denotes a lead-free/RoHS-compliant package.

*A "+" anywhere on the top mark indicates a lead-free package. An "N" anywhere on the top mark indicates an industrial temperature range device.

FEATURES

- Real-Time Clock (RTC) Counts Seconds, Minutes, Hours, Date of the Month, Month, Day of the week, and Year with Leap-Year Compensation Valid Up to 2100
- 56-Byte, Battery-Backed, General-Purpose RAM with Unlimited Writes
- I²C Serial Interface
- Programmable Square-Wave Output Signal
- Automatic Power-Fail Detect and Switch Circuitry
- Consumes Less than 500nA in Battery-Backup Mode with Oscillator Running
- Optional Industrial Temperature Range: -40°C to +85°C
- Available in 8-Pin Plastic DIP or SO
- Underwriters Laboratories (UL) Recognized

PIN CONFIGURATIONS



48 × 84 pixels matrix LCD controller/driver**PCD8544****1 FEATURES**

- Single chip LCD controller/driver
- 48 row, 84 column outputs
- Display data RAM 48 × 84 bits
- On-chip:
 - Generation of LCD supply voltage (external supply also possible)
 - Generation of intermediate LCD bias voltages
 - Oscillator requires no external components (external clock also possible).
- External $\overline{\text{RES}}$ (reset) input pin
- Serial interface maximum 4.0 Mbits/s
- CMOS compatible inputs
- Mux rate: 48
- Logic supply voltage range V_{DD} to V_{SS} : 2.7 to 3.3 V
- Display supply voltage range V_{LCD} to V_{SS}
 - 6.0 to 8.5 V with LCD voltage internally generated (voltage generator enabled)
 - 6.0 to 9.0 V with LCD voltage externally supplied (voltage generator switched-off).
- Low power consumption, suitable for battery operated systems
- Temperature compensation of V_{LCD}
- Temperature range: -25 to +70 °C.

4 ORDERING INFORMATION

TYPE NUMBER	PACKAGE		
	NAME	DESCRIPTION	VERSION
PCD8544U	-	chip with bumps in tray; 168 bonding pads + 4 dummy pads	-

2 GENERAL DESCRIPTION

The PCD8544 is a low power CMOS LCD controller/driver, designed to drive a graphic display of 48 rows and 84 columns. All necessary functions for the display are provided in a single chip, including on-chip generation of LCD supply and bias voltages, resulting in a minimum of external components and low power consumption.

The PCD8544 interfaces to microcontrollers through a serial bus interface.

The PCD8544 is manufactured in n-well CMOS technology.

3 APPLICATIONS

- Telecommunications equipment.

ATMEL 8-BIT MICROCONTROLLER WITH 4/8/16/32KBYTES IN-SYSTEM PROGRAMMABLE FLASH

DATASHEET

Features

- High Performance, Low Power Atmel® AVR® 8-Bit Microcontroller Family
- Advanced RISC Architecture
 - 131 Powerful Instructions – Most Single Clock Cycle Execution
 - 32 x 8 General Purpose Working Registers
 - Fully Static Operation
 - Up to 20 MIPS Throughput at 20MHz
 - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
 - 4/8/16/32KBytes of In-System Self-Programmable Flash program memory
 - 256/512/1KBytes EEPROM
 - 512/1K/1K/2KBytes Internal SRAM
 - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
 - Data retention: 20 years at 85°C/100 years at 25°C⁽¹⁾
 - Optional Boot Code Section with Independent Lock Bits
 - In-System Programming by On-chip Boot Program
 - True Read-While-Write Operation
 - Programming Lock for Software Security
- Atmel® QTouch® library support
 - Capacitive touch buttons, sliders and wheels
 - QTouch and QMatrix® acquisition
 - Up to 64 sense channels
- Peripheral Features
 - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
 - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
 - Real Time Counter with Separate Oscillator
 - Six PWM Channels
 - 8-channel 10-bit ADC in TQFP and QFN/MLF package
 - Temperature Measurement
 - 6-channel 10-bit ADC in PDIP Package
 - Temperature Measurement
 - Programmable Serial USART
 - Master/Slave SPI Serial Interface
 - Byte-oriented 2-wire Serial Interface (Philips I²C compatible)
 - Programmable Watchdog Timer with Separate On-chip Oscillator
 - On-chip Analog Comparator
 - Interrupt and Wake-up on Pin Change

- Special Microcontroller Features
 - Power-on Reset and Programmable Brown-out Detection
 - Internal Calibrated Oscillator
 - External and Internal Interrupt Sources
 - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
 - 23 Programmable I/O Lines
 - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
 - 1.8 - 5.5V
- Temperature Range:
 - -40°C to 85°C
- Speed Grade:
 - 0 - 4MHz@1.8 - 5.5V, 0 - 10MHz@2.7 - 5.5V, 0 - 20MHz @ 4.5 - 5.5V
- Power Consumption at 1MHz, 1.8V, 25°C
 - Active Mode: 0.2mA
 - Power-down Mode: 0.1µA
 - Power-save Mode: 0.75µA (Including 32kHz RTC)

Remote 8-bit I/O expander for I²C-bus

PCF8574

1 FEATURES

- Operating supply voltage 2.5 to 6 V
- Low standby current consumption of 10 µA maximum
- I²C-bus to parallel port expander
- Open-drain interrupt output
- 8-bit remote I/O port for the I²C-bus
- Compatible with most microcontrollers
- Latched outputs with high current drive capability for directly driving LEDs
- Address by 3 hardware address pins for use of up to 8 devices (up to 16 with PCF8574A)
- DIP16, or space-saving SO16 or SSOP20 packages.



The device consists of an 8-bit quasi-bidirectional port and an I²C-bus interface. The PCF8574 has a low current consumption and includes latched outputs with high current drive capability for directly driving LEDs. It also possesses an interrupt line (\bar{INT}) which can be connected to the interrupt logic of the microcontroller. By sending an interrupt signal on this line, the remote I/O can inform the microcontroller if there is incoming data on its ports without having to communicate via the I²C-bus. This means that the PCF8574 can remain a simple slave device.

The PCF8574 and PCF8574A versions differ only in their slave address as shown in Fig.10.

2 GENERAL DESCRIPTION

The PCF8574 is a silicon CMOS circuit. It provides general purpose remote I/O expansion for most microcontroller families via the two-line bidirectional bus (I²C-bus).

3 ORDERING INFORMATION

TYPE NUMBER	PACKAGE		
	NAME	DESCRIPTION	VERSION
PCF8574P; PCF8574AP	DIP16	plastic dual in-line package; 16 leads (300 mil)	SOT38-4
PCF8574T; PCF8574AT	SO16	plastic small outline package; 16 leads; body width 7.5 mm	SOT162-1
PCF8574TS; PCF8574ATS	SSOP20	plastic shrink small outline package; 20 leads; body width 4.4 mm	SOT266-1

nRF24L01+

Single Chip 2.4GHz Transceiver

Product Specification v1.0

Key Features

- Worldwide 2.4GHz ISM band operation
- 250kbps, 1Mbps and 2Mbps on air data rates
- Ultra low power operation
- 11.3mA TX at 0dBm output power
- 13.5mA RX at 2Mbps air data rate
- 900nA in power down
- 26µA in standby-I
- On chip voltage regulator
- 1.9 to 3.6V supply range
- Enhanced ShockBurst™
- Automatic packet handling
- Auto packet transaction handling
- 6 data pipe MultiCeiver™
- Drop-in compatibility with nRF24L01
- On-air compatible in 250kbps and 1Mbps with nRF2401A, nRF2402, nRF24E1 and nRF24E2
- Low cost BOM
- ±60ppm 16MHz crystal
- 5V tolerant inputs
- Compact 20-pin 4x4mm QFN package

Applications

- Wireless PC Peripherals
- Mouse, keyboards and remotes
- 3-in-1 desktop bundles
- Advanced Media center remote controls
- VoIP headsets
- Game controllers
- Sports watches and sensors
- RF remote controls for consumer electronics
- Home and commercial automation
- Ultra low power sensor networks
- Active RFID
- Asset tracking systems
- Toys

All rights reserved.

Reproduction in whole or in part is prohibited without the prior written permission of the copyright holder.
September 2008

Digital Relative Humidity sensor with Temperature output



- **DFN type package**
- **Relative Humidity and Temperature Digital Output, I²C interface**
- **Fully calibrated**
- **Lead free sensor, reflow solderable**
- **Low power consumption**
- **Fast response time**

DESCRIPTION

The HTU21D(F) is a new digital humidity sensor with temperature output by MEAS. Setting new standards in terms of size and intelligence, it is embedded in a reflow solderable Dual Flat No leads (DFN) package with a small 3 x 3 x 0.9 mm footprint. This sensor provides calibrated, linearized signals in digital, I²C format.

HTU21D(F) digital humidity sensors are dedicated humidity and temperature plug and play transducers for OEM applications where reliable and accurate measurements are needed. Direct interface with a micro-controller is made possible with the module for humidity and temperature digital outputs. These low power sensors are designed for high volume and cost sensitive applications with tight space constraints.

Every sensor is individually calibrated and tested. Lot identification is printed on the sensor and an electronic identification code is stored on the chip – which can be read out by command. Low battery can be detected and a checksum improves communication reliability. The resolution of these digital humidity sensors can be changed by command (8/12bit up to 12/14bit for RH/T).

With MEAS' improvements and miniaturization of this sensor, the performance-to-price ratio has been improved – and eventually, any device should benefit from its cutting edge energy saving operation mode.

Optional PTFE filter/membrane (F) protects HTU21D digital humidity sensors against dust and water immersion, as well as against contamination by particles. PTFE filter/membranes preserve a high response time. The white PTFE filter/membrane is directly stuck on the sensor housing.

FEATURES

APPLICATIONS

- | | |
|--|--|
| <ul style="list-style-type: none"> • Full interchangeability with no calibration required in standard conditions • Instantaneous desaturation after long periods in saturation phase • Compatible with automatized assembly processes, including Pb free and reflow processes • Individual marking for compliance to stringent traceability requirements | <ul style="list-style-type: none"> • Automotive: defogging, HVAC • Home Appliance • Medical • Printers • Humidifier |
|--|--|



September 2014



LM78XX / LM78XXA

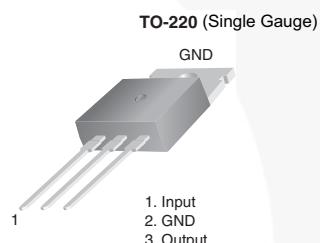
3-Terminal 1 A Positive Voltage Regulator

Features

- Output Current up to 1 A
- Output Voltages: 5, 6, 8, 9, 10, 12, 15, 18, 24 V
- Thermal Overload Protection
- Short-Circuit Protection
- Output Transistor Safe Operating Area Protection

Description

The LM78XX series of three-terminal positive regulators is available in the TO-220 package and with several fixed output voltages, making them useful in a wide range of applications. Each type employs internal current limiting, thermal shut-down, and safe operating area protection. If adequate heat sinking is provided, they can deliver over 1 A output current. Although designed primarily as fixed-voltage regulators, these devices can be used with external components for adjustable voltages and currents.



Ordering Information⁽¹⁾

Product Number	Output Voltage Tolerance	Package	Operating Temperature	Packing Method
LM7805CT				
LM7806CT				
LM7808CT				
LM7809CT				
LM7810CT				
LM7812CT				
LM7815CT				
LM7818CT				
LM7824CT				
LM7805ACT	±4%	TO-220 (Single Gauge)	-40°C to +125°C	Rail
LM7809ACT				
LM7810ACT				
LM7812ACT				
LM7815ACT	±2%		0°C to +125°C	

Note:

1. Above output voltage tolerance is available at 25°C.

LM1117-N/LM1117I 800mA Low-Dropout Linear Regulator

Check for Samples: [LM1117-N](#), [LM1117I](#)

FEATURES

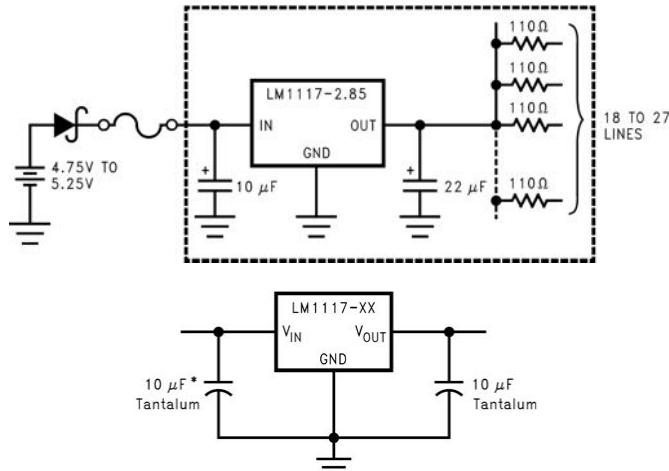
- Available in 1.8V, 2.5V, 2.85V, 3.3V, 5V, and Adjustable Versions
- Space Saving SOT-223 and WSON Packages
- Current Limiting and Thermal Protection
- Output Current 800mA
- Line Regulation 0.2% (Max)
- Load Regulation 0.4% (Max)
- Temperature Range
 - LM1117-N: 0°C to 125°C
 - LM1117I: -40°C to 125°C

APPLICATIONS

- 2.85V Model for SCSI-2 Active Termination
- Post Regulator for Switching DC/DC Converter
- High Efficiency Linear Regulators
- Battery Charger
- Battery Powered Instrumentation

TYPICAL APPLICATION

Active Terminator for SCSI-2 Bus



* Required if the regulator is located far from the power supply filter.

Figure 1. Fixed Output Regulator

DESCRIPTION

The LM1117-N is a series of low dropout voltage regulators with a dropout of 1.2V at 800mA of load current. It has the same pin-out as Texas Instruments' industry standard LM317.

The LM1117-N is available in an adjustable version, which can set the output voltage from 1.25V to 13.8V with only two external resistors. In addition, it is also available in five fixed voltages, 1.8V, 2.5V, 2.85V, 3.3V, and 5V.

The LM1117-N offers current limiting and thermal shutdown. Its circuit includes a zener trimmed bandgap reference to assure output voltage accuracy to within $\pm 1\%$.

The LM1117-N series is available in WSON, PFM, SOT-223, TO-220, and TO-263 DDPAK packages. A minimum of $10\mu\text{F}$ tantalum capacitor is required at the output to improve the transient response and stability.



Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

All trademarks are the property of their respective owners.

Bijlage 5: Referenties

FLORIAN SCHÄFFER (2011). *AVR:Hardware en c-programmering in de praktijk.* (2de druk). Amersfoort: Wilco.

<http://stackoverflow.com/questions/3577784/byte-bcd-to-ascii-conversion-optimization>

<http://bytes.com/topic/c/answers/212851-decimal-bcd-long-int>

http://www.nongnu.org/avr-libc/user-manual/group_avr_interrupts.html

http://www.nongnu.org/avr-libc/user-manual/group_avr_sleep.html

Datasheets: Maxim DS1307, Philips PCD8544, Atmel Atmega328p, Philips PCF8574, Nordic nRF24L01+, Fairchild LM 7805, Texas Instruments LM1117, HTU21D.

Datum goedkeuring	Versie	Datum herziening		Redacteur(s)

Projectvoorstel

1 **Teamleden :** Olivier Van den Eede

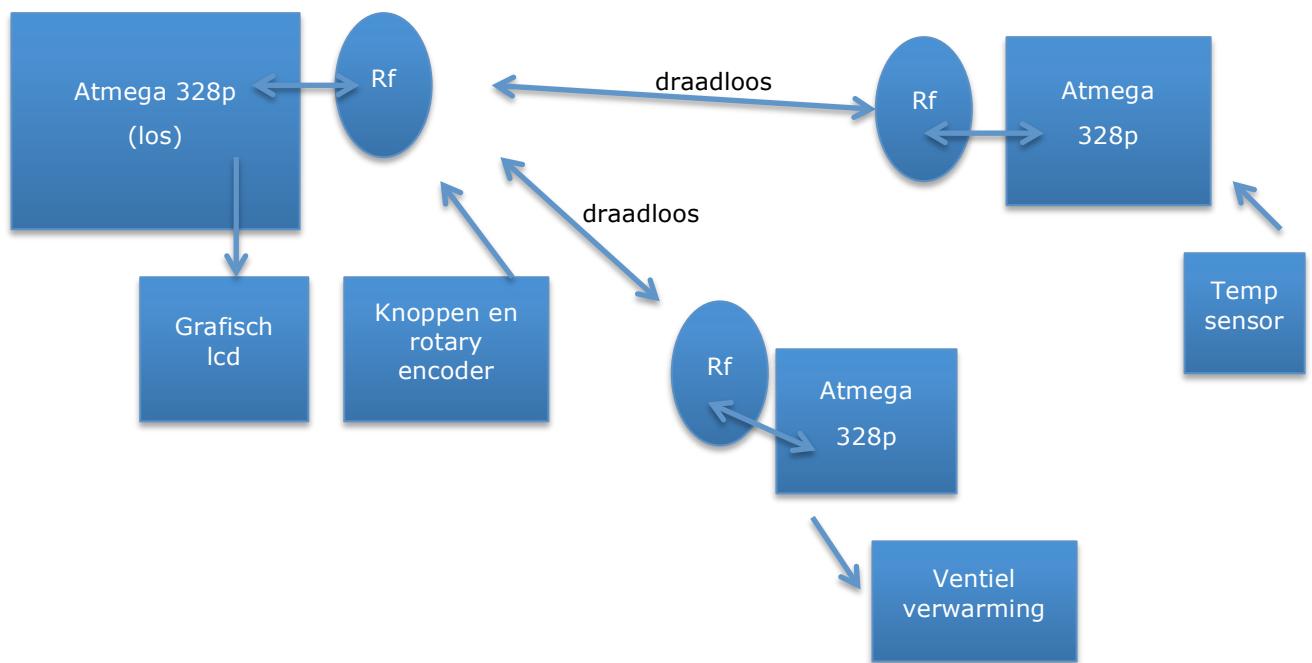
2 **Titel project.** Kamerthermostaat

3 **Korte omschrijving van het project.**

Een thermostaat om de verwarming van mijn kamer te regelen, met een grafisch lcd, een rotary encoder, en een draadloze temperatuursensor.

Dit bestuurd een elektrisch radiatorventiel die ook draadloos bestuurd kan worden.

4 **Blokschema.**



5 Een budgetraming (mini).

Naam	Prijs
lcd(grafisch)	2,00 €
rotary encoder	1,00 €
atmega 4x	10,00 €
NRF24L01+ 4x	8,00 €
weerstanden+condensatoren	8,00 €
rtc + kristal	2,00 €
knoppen en switches	4,00 €
3.3v voltage regulators	3,00 €
temp sensor	1,00 €
5v voeding(lm + cond.)	2,00 €
totaal	41,00 €

Plan van aanpak

Titel van het project: Kamerthermostaat
Projectmedewerkers: Olivier Van den Eede

Externe partners/opdrachtgevers:

Doel & ontwerpspecificaties

Doel

Een thermostaat maken die de temperatuur van mijn kamer kan regelen. Dit adhv. Een microcontroller, grafisch lcd, een rotary-encoder, en een draadloze temperatuursensor.

Ontwerpspecificaties

De gebruiker kan via de rotary encoder en de verschillende toetsen een aantal instellingen doen op de hoofdmodule zoals de temperatuur en een aantal tijden. Dit is om alles te kunnen instellen.

De hoofdmodule communiceert via nRF24L01+ rf-modules met de andere microcontrollers. Deze controllers sturen de temperatuursensor en het ventiel voor de verwarming aan.

Grootte hoofdmodule: een microcontroller, een grafisch lcd, rf-module en een rotary encoder.

Grootte temp-sensor: een microcontroller, een temperatuursensor en een rf-module.

Prijs: +-45 euro

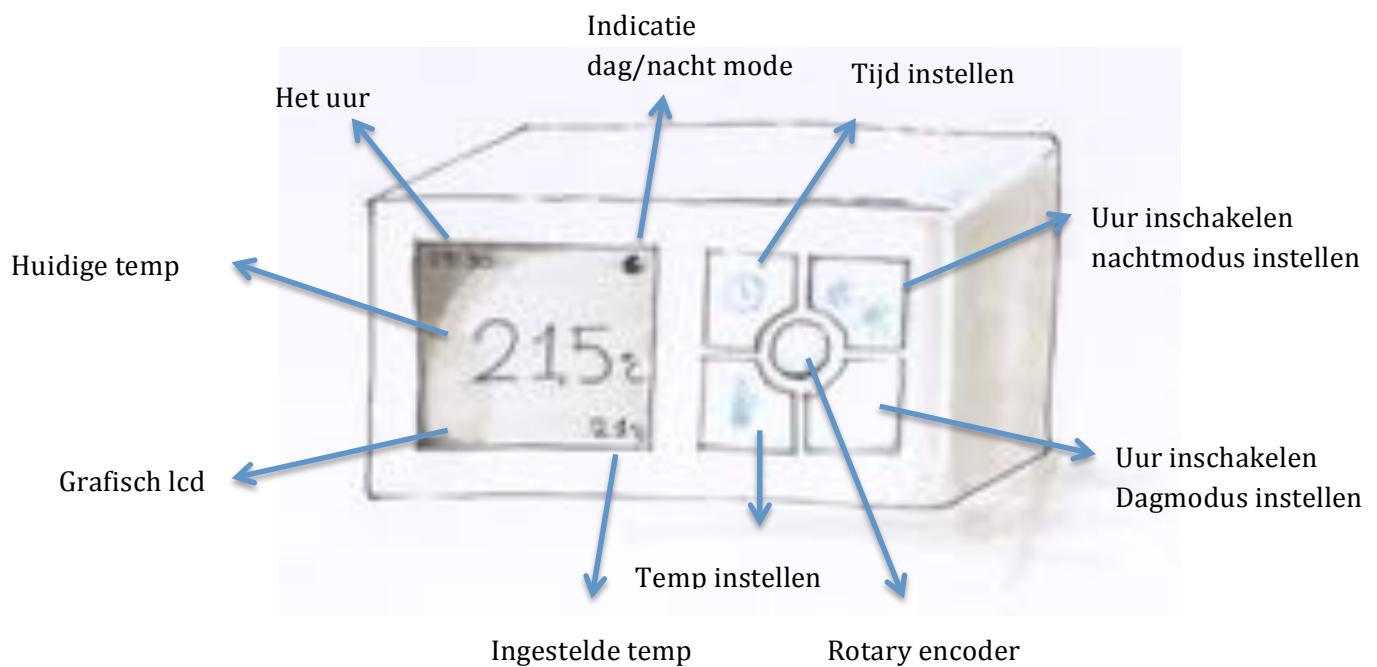
Tijd: n/a

Levensduur: enkele jaren

Functionele analyse

Kamerthermostaat:

- Gebruiker stelt een temperatuur en de tijden waartussen de verwarming moet werken in
- Microcontroller staagt deze gegevens op
- Microcontroller wacht tot de temperatuursensor de gemeten waarde uitstuurt
- Microcontroller geeft deze waarde weer op het scherm
- Microcontroller vergelijkt deze waarde met de ingestelde waarde, en schakelt de verwarming in of uit
- Als de in of uitschakeltijd overschreden word, schakelt de controller de verwarming uit
- De gebruiker kan op elk moment de temperatuur en de tijden bijregelen dmv. De knoppen en de rotary encoder.
- Als de temperatuursensor niet aan het meten is, gaat de atmega in slaapstand



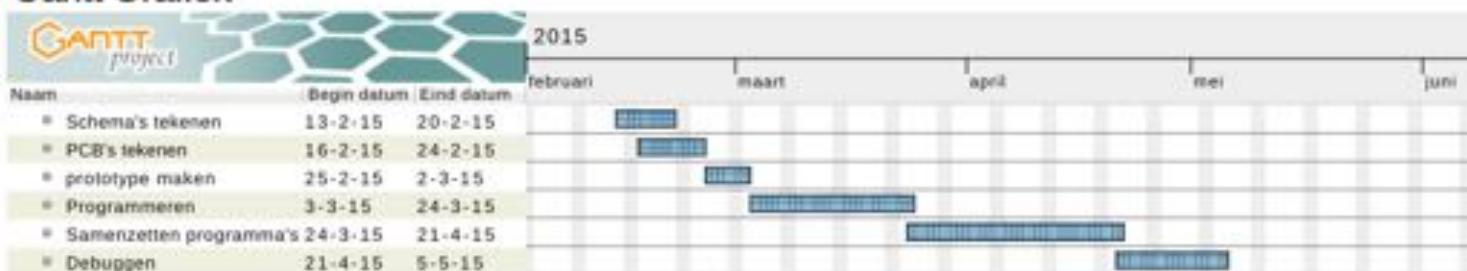
Work Breakdown Structure (WBS)

1. Schema's en pcb's tekenen	2 weken
1.1 Hoofdmodule thermostaat	1 week
1.2 Bijmodules (temp-sens,raamcont,ventiel)	1 week
2. Prototype pcb's solderen	2 weken
2.1 Hoofdmodule thermostaat	1 week
2.2 Bijmodule	1 week
3. Programmeren	5 weken
3.1 Grafisch lcd	1 week
3.2 Nrf12l01+ modules	1 week
3.3 Rotary encoder, button's en menu	1 week
3.4 Alles samen laten werken	2 weken
3. Testing en Troubleshooting	1 à 2 weken
4. Behuizing	1 week

Gantt-chart

Untitled Gantt Project

Gantt Grafiek



Datum	Taak	Tijd
13 feb. 15	Maken projectvoorstel	1u
16 feb. 15	Beginnen tekenen schema's	2u
17 feb. 15	Opmaak plan van aanpak	1u
20 feb. 15	Afmaken schema's + beginnen pcb	4u
27 feb. 15	verder tekenen pcb + testen lcd en lm35	4u
06 mrt. 15	Tekenen 2de pcb	4u
13 mrt. 15	testen i2c port expander	2u
18 mrt. 15	Testen nrf-comunicatie	5u
20 mrt. 15	Testen nrf-comunicatie	5u
21 mrt. 15	Succesvolle communicatie nrf-module	3u
23 mrt. 15	Start maken prototype hoofdmodule	2u
25 mrt. 15	Verder maken prototype + testen eerste deel	3u
26 mrt. 15	Prototype afwerken	2,5u
27 mrt. 15	i2c-rtc + rotary encoder programma	4u
28 mrt. 15	Beginnen samenvoegen rtc,lcd,rotary en i2c-exp	3u
28 mrt. 15	Beginnen samenvoegen rtc,lcd,rotary en i2c-exp	3u
03 apr. 15	temp regelen met rotary + temp doorsturen	4u
12 apr. 15	solderen hoofdmodule	2u
13 apr. 15	programmeren hoofdmodule	4u
14 apr. 15	programmeren hoofdmodule	3u
15 apr. 15	programmeren hoofdmodule	5u
16 apr. 15	uitzoeken t2+sleep mode	2u
22 apr. 15	timer2 en sleep mode programmeren	2u
22 apr. 15	solderen relais module	1u
24 apr. 15	programmeren communicatie relais+hoofdmodule	4u
01 mei 15	programmeren temp-sensor	3u
08 mei 15	programmeren temp-sensor	4u
12 mei 15	Project klaar	