

Geïntegreerde Proef

Elektriciteit – Elektronica - ICT

Slimme Parkeergarage



Thian D'haene

Klas: 6ICT

Studierichting: Industriële ICT

Schooljaar: 2022-2023

Technische mentor: Mevr. Vervaeke

Taalmentor: Mevr. Cardinael

1. Voorwoord

In mijn laatste jaar moet ik een eindwerk of GIP maken. Dit houdt in dat ik eerst verschillende deelprojecten moet maken en hiervan telkens een duidelijke uitleg moet beschrijven. Hierna komen alle deelprojecten samen in een volledige opstelling. Ook moet ik een GIP dossier maken waarin verschillende delen van mijn GIP staan uitgelegd, bijvoorbeeld alle deelprojecten die beschreven worden en de code van mijn volledige opstelling.

Voor mijn eindwerk maak ik een slimme parkeergarage. Dit houdt in dat elke parkeerplaats een rood en groen lampje heeft als aanduiding of een parkeerplaats al dan niet bezet is, er een display is die het aantal vrije plaatsen toont, werkende slagbomen aan de in- en uitgangen en een applicatie waarin een overzicht van de volledige parking wordt getoond.

Ook wil ik nog verschillende mensen bedanken voor het helpen bij het realiseren van dit eindwerk. Als eerste wil ik Mr. Van Quickelberghe bedanken. Hij heeft mij zeer veel geholpen tijdens mijn GIP. Ook mijn klasgenoten wil ik bedanken voor de hulp. Als laatste wil ik mijn ouders bedanken voor de (financiële) steun.

Inhoud

1.	Voorwoord	3
2.	Inleiding.....	6
3.	Blokschema	7
4.	Hardware.....	8
	- Atmega 324P	8
	- MAX232	8
	- Servomotor MG90s 9G	8
	- Module PCF8574	8
	- LJ12A3-4-Z/BX.....	8
	- IC 74HC04 Hex Inverters.....	8
	- 7 Segment displays	8
5.	Software	9
	- Atmel Studio 7	9
	- hTerm	9
	- Logic.....	9
	- Visual Studio 2022	9
	- EasyEDA	9
	- Thinkercad	9
	- GitHub.....	9
	- Zelf geschreven software	9
6.	Voorontwerp.....	10
	- Aansturen Servomotor	10
	- Seriële Communicatie.....	11
	- 7-segment.....	14
	- Aansluiten sensoren.....	15
	- Aansluiten leds met I ² C.....	17
7.	Ontwerp hardware.....	18
	- Schema	18
	-Bespreking schema.....	19
	- Print	24
8.	Applicaties.....	26
	Initialisatie Microcontrollers	26

Main Microcontrollers	27
While(1) microcontrollers.....	27
Functies.....	28
C# programma	29
GUI	33
9. Componentenlijst & Kostprijsberekening.....	34
10. Besluit.....	35
11. Bronnen.....	36
12. Bijlagen	37

2. Inleiding

In dit GIP dossier zal ik mijn eindwerk, een slimme parkeergarage, presenteren. Dit doe ik in verschillende delen.

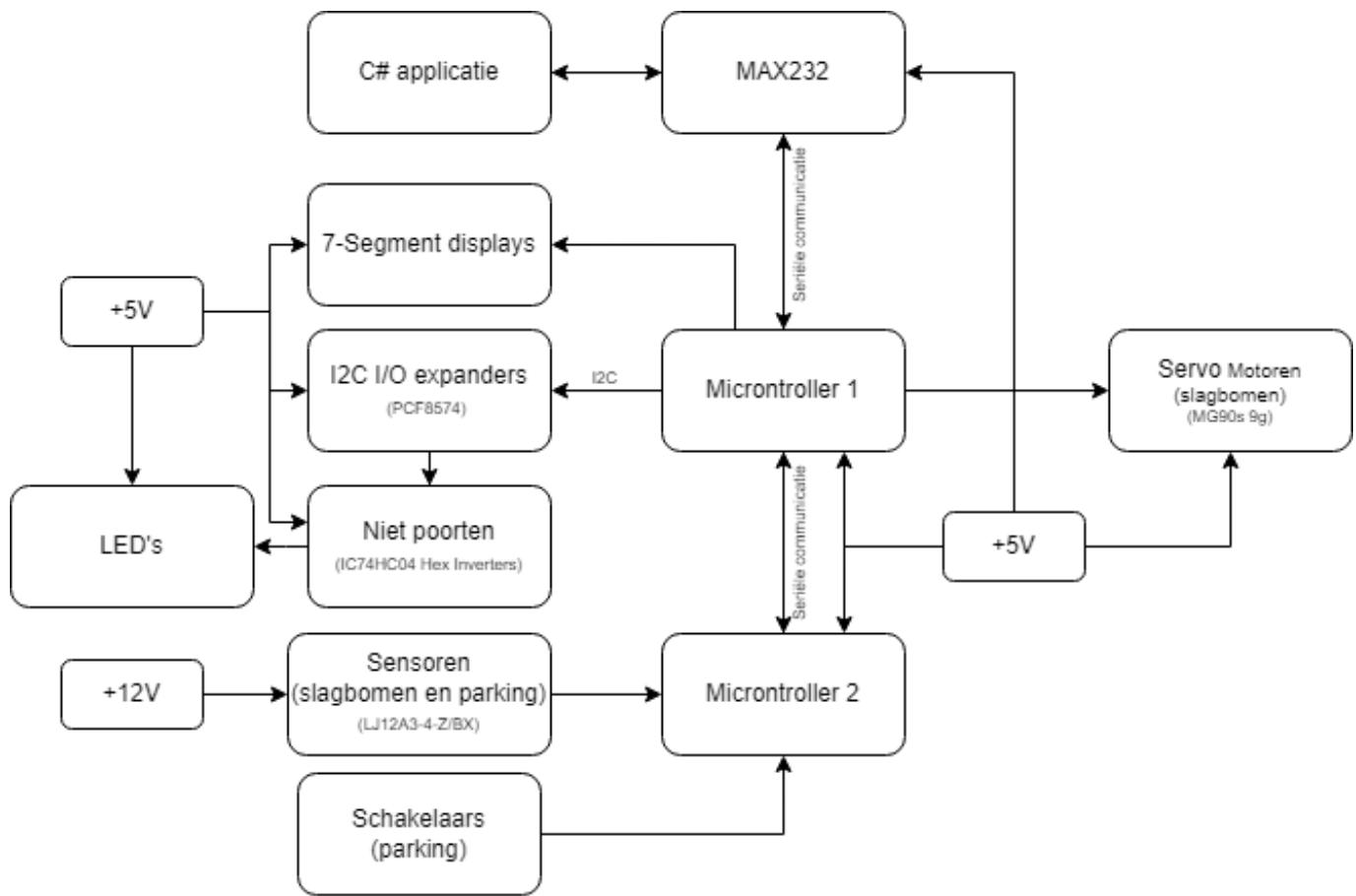
Eerst start ik met een blokschema. Dit is een schema waarin alle componenten en de verbindingen tussen die componenten wordt voorgesteld. Daarna leg ik de hardware uit. Dit zijn de componenten die belangrijk zijn in dit eindwerk met extra uitleg per component. Na de hardware leg ik de verschillende software uit die gebruikt is en waarom ik gekozen heb voor die software.

Vervolgens komt het voorontwerp. Dit zijn alle verschillende deelprojecten die ik tijdens de eerste fase van dit eindwerk heb gemaakt. Het eerste deelproject is het aansturen van de servomotoren. Deze servomotoren zijn nodig om mijn slagbomen precies te kunnen aansturen. Daarna komt het deelproject over de seriële communicatie, dit is nodig zodat de microcontroller en de pc serieel met elkaar kunnen communiceren. Het derde deelproject gaat over hoe het 7-segment display aangesloten wordt. Het vierde deelproject gaat over hoe de sensoren worden aangesloten en hoe ik dit berekend heb. Als laatste leg ik uit hoe ik alle leds zal verbinden met de microcontroller en met I2C modules. Per deelproject wordt een stuk code uitgelegd, is er een blokschema, een aansluitschema en indien nodig zijn er berekeningen.

Daarna leg ik het ontwerp van de hardware uit. Dit is de uitleg van het volledige schema en de pcb en hoe ik hiertoe gekomen ben. Een gelijkaardig deel is het achtste deel over de applicaties. Hierin leg ik alle software uit die in de finale opstelling gebruikt wordt.

Tenslotte is er nog een componentenlijst met per component de prijs. Ook staat de volledige prijs van mijn eindwerk in dit deel. Daarna is er nog een besluit waarin ik een evaluatie geef van een jaar GIP. Als laatste staan er nog enkele bronnen en bijlagen die ik gebruikt heb.

3. Blokschema



Dit is het blokschema van mijn eindwerk. Hierop worden alle verschillende verbindingen tussen de componenten afgebeeld. In dit deel van mijn GIP dossier zal ik deze verbindingen uitleggen.

Centraal staan de twee microcontrollers. Deze 2 microcontrollers zijn serieel verbonden met elkaar. Om met mijn C# applicatie te communiceren zal vanaf de eerste microcontroller serieel gecommuniceerd worden. Ook de I₂C modules zijn verbonden met de eerste microcontroller. Om met deze modules de LEDS aan te sturen worden deze nog met niet-poorten verbonden. Naast de I₂C modules zijn ook de servomotoren en de 7-segment displays op de eerste microcontroller aangesloten. Om alle sensoren en schakelaars te kunnen inlezen maak ik gebruik van een tweede microcontroller. De schakelaars werken gewoon op 5V maar de sensoren werken op 12V rechtstreeks van de voeding.

4. Hardware

- Atmega 324P

Dit is de microcontroller die ik voor mijn GIP zal gebruiken. Dit is een 8-bit microcontroller met veel extra functies die ik nodig heb zoals 2 PWM pinnen, pinnen voor seriële communicatie en aansluitingen voor i2c modules. Omdat ik een groot aantal pinnen nodig heb om alle schakelaars en sensoren aan te sluiten zal ik 2 microcontrollers gebruiken die met elkaar serieel communiceren.



- MAX232

Dit is de module die nodig is voor de seriële communicatie met de pc. De pinnen VCC en GND worden aangesloten op de gelijknamige pinnen op de interface. De RX pin en TX pin moeten aangesloten worden op pin 14 & pin 15. De module wordt hierna aangesloten via een seriële kabel op de pc. Nu kan er seriële communicatie tussen pc en microcontroller uitgevoerd worden.



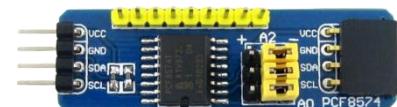
- Servomotor MG90s 9G

De MG90s is een servomotor die in principe hetzelfde is als de SG90 servomotor maar dan met metalen tandwielen. Deze servomotor zal ik in mijn GIP gebruiken. Met deze servomotor kan ik de 2 slagbomen openen aan de hand van een zelfgeschreven programma.



- Module PCF8574

Dit is de I2C module die ik zal gebruiken in mijn GIP. Deze module heb ik nodig om het aantal I/O pinnen te verhogen aangezien er veel LEDs aangestuurd moeten worden. Eventueel kan ik deze module ook gebruiken om mijn sensoren in te lezen.



- LJ12A3-4-Z/BX

Om te detecteren of een parkeerplaats bezet is en om te detecteren of er een wagen voor de slagboom staat, gebruik ik een inductieve benaderingssensor. Deze sensoren kunnen verschillende metalen (koper, aluminium, staal, ...) detecteren. Daarom zal ik onder mijn wagens een klein metalen plaatje moeten monteren.



- IC 74HC04 Hex Inverters

Dit zijn IC's die zorgen dat het inkomende signaal geïnverteerd wordt. Dit zal ik gebruiken voor de parkeerplaatsen zodat maar 1 pin van de I2C modules nodig is om 2 leds aan te sturen. Deze IC's werken op 5V en zullen op een aparte print worden gemonteerd.



- 7 Segment displays

Om weer te geven hoeveel parkeerplaatsen er nog beschikbaar zijn in de parking maak ik gebruik van 7-segment displays. Dit display bestaat uit 7 staafvormige segmenten die een 8 voorstellen. Hiermee kunnen alle mogelijk cijfers getoond worden. De module waarop deze displays zijn gemonteerd wordt later nog besproken.



5. Software

-Atmel Studio 7

Om de Atmega324p te kunnen programmeren, gebruik ik de IDE Atmel Studio 7. Hierin schrijf ik de code voor de Atmega324p. Deze code wordt geschreven in C. Atmel Studio 7 is een geïntegreerde ontwikkelomgeving (IDE) voor het programmeren van microcontrollers van Atmel, zoals de Atmega324p die wij gebruiken.



-hTerm

Om de seriële communicatie te kunnen testen, gebruik ik het programma hTerm. Hierin kan ik ontvangen data weergeven en ook data naar de microcontroller sturen. Later zal ik voor de seriële communicatie een zelfgemaakte C# applicatie gebruiken.



-Logic

Deze software gebruik ik in combinatie met de logic analyzer om de status van verschillende pinnen van de microcontroller te kunnen weergeven op de pc. Dit is net als hTerm een programma dat gebruikt wordt bij het maken van mijn eindwerk.



-Visual Studio 2022

Om de applicatie te maken die op een pc zal draaien, gebruik ik Visual Studio 2022. Hierin kan een form gemaakt worden dat geprogrammeerd wordt in C#. Visual Studio 2022 is een IDE die vooral gebruikt wordt om toepassingen te maken in C#.



-EasyEDA

In deze software heb ik mijn verschillende elektrische schema's ontworpen. Ook het pcb heb ik met deze software volledig getekend. EasyEDA is een programma waarin verschillende schema's en pcb kunnen ontworpen worden. Ik heb voor deze editor gekozen aangezien er veel componenten ingebouwd zijn en ik ook zelf een library kan toevoegen.



-Thinkercad

Hierin heb ik de behuizingen die gebruikt worden voor mijn slagbomen ontworpen. Dit is een makkelijke online 3D-modelleringsprogramma waarmee simpele 3D tekeningen gemaakt kunnen worden.



-GitHub

Om van al mijn software samen een back-up te maken, gebruik ik GitHub. Hierdoor heb ik ook alle verschillende versies van mijn software online ter beschikking en kan ik makkelijk mijn software aanpassen op verschillende locaties (school en thuis).



-Zelf geschreven software

Ik heb ook zelfgeschreven testsoftware voor elk deelproject. Deze zal ik ook verder gebruiken in de volledige opstelling.

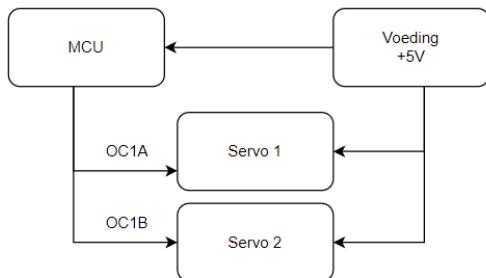
6. Voorontwerp

- Aansturen Servomotor

Doelstellingen:

De doelstelling van dit deelproject was het aansturen van 2 servomotoren via de microcontroller. Deze 2 servomotoren worden gebruikt voor de slagbomen bij de in- en uitgangen van de parkeergarage. Ook zijn er voor dit deelproject 2 behuizingen gemaakt waarin de servomotoren worden gemonteerd.

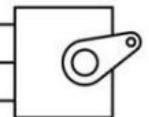
Blokschema:



Betekenis van de aansluitingen:

Ik sluit de servomotor aan op drie aansluitingen: VCC, GND en PWM. De VCC en GND verbind ik met een voeding die een spanning van 4,8 V tot 6 V levert. Ik gebruik een spanning van 5 V omdat dit dezelfde spanning is als die van de microcontroller. De PWM-aansluiting verbind ik met OC1A of OC1B op de microcontroller. Deze aansluitingspinnen sturen een PWM-signal dat de positie van de servomotor nauwkeurig kan instellen.

PWM=Orange (⊟⊟)
Vcc=Red (+)
Ground=Brown (-)



Broncode om de werking uit te leggen:

Dit is de code om de juiste positie van de servomotor in te stellen. Ik gebruik een functie die enkel het aantal graden ontvangt van het hoofdprogramma. OCR1A wordt aangepast naar het berekende aantal graden. Eén graad is gelijk aan 5.1222 die toegevoegd of verminderd moet worden van OCR1A. De waarde voor 0 graden is 231 en de waarde voor 180 graden is 1153.

In de registers moeten deze instellingen gemaakt worden. Hier wordt PWM mode 14 ingesteld en worden de COMBITS voor OCRA en OC RB ingesteld zodat het PWM signaal uitgestuurd wordt op deze uitgangen.

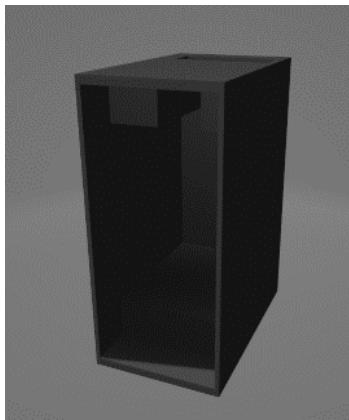
Reflectie:

Uiteindelijk is er een werkend programma. De servomotoren kunnen naar wens precies aangestuurd worden.

```
char Servo1(unsigned char graden)
{
    OCR1A=(231+(graden*5.1222));
}
//OCR1A INSTELLEN ALS UITGANG
DDRD|=(1<<DDRD5);
DDRD|=(1<<DDRD4);
//WGM BITS MODE 14
TCCR1A&=~(1<<WGM10);
TCCR1A|=(1<<WGM11);
TCCR1B|=((1<<WGM12)|(1<<WGM13));
//COMBITS A instellen
TCCR1A|=(1<<COM1A1);
TCCR1A&=~(1<<COM1A0);
//COMBITS B instellen
TCCR1A|=(1<<COM1B1);
TCCR1A&=~(1<<COM1B0);
//ICR1
ICR1 = 9215; // aantal telpulsen
//OCR1A
OCR1A = 231; //
OCR1B = 231; //
```

Behuizing:

Om de servomotoren te monteren in de parking heb ik 2 behuizingen gemaakt waarin de servomotoren zitten bevestigd. Deze heb ik in Thinkercad getekend en deze zijn met een 3D printer geprint.

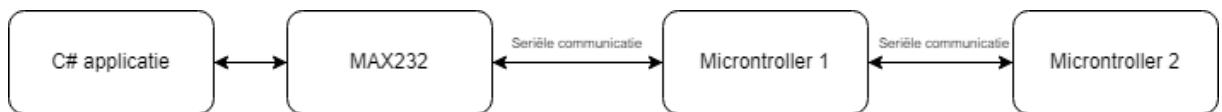


- Seriële Communicatie

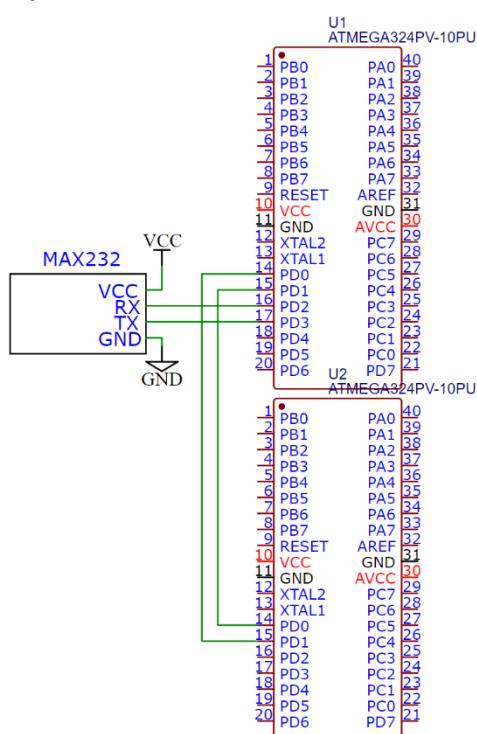
Doelstelling:

Schrijven van een programma voor seriële communicatie met een pc en een 2^{de} microcontroller.

Blokschema:



Schema:



Betekenis van de aansluitingen:

Om de communicatie tussen microcontroller 1 en 2 tot stand te brengen gebruik ik RX0 van MCU1 verbonden met TX0 van MCU2. TX0 van MCU1 wordt verbonden met RX0 van MCU2.

Voor de communicatie tussen de eerste microcontroller en de pc maak ik gebruik van RX1 en TX1. Deze worden beide aangesloten op hun overeenstemmende aansluitingen van de MAX232 module.

Broncode om de werking uit te leggen:

Dit is de nodige code om de seriële communicatie te initialiseren. Het zorgt ervoor dat de baudrate ingesteld wordt op 9600bps, dat er 8 databits zijn, dat er één stopbit is en geen pariteit. Ook het activeren van de interrupt bij ontvangst van data wordt in dit stukje code gedaan. Dit is voor de communicatie tussen de microcontrollers en de communicatie tussen de pc hetzelfde, wel moeten de registers aangepast worden zodat UDR0 en UDR1 gebruikt worden.

```
void serieel_init0(void)
{
    //de pin PD2 als ingang (RX), de pin PD3 als uitgang (TX) met een hoog niveau
    DDRD &=~(1<<DDRD0);
    DDRD |=(1<<DDRD1);

    //init seriële communicatie
    //USART0, 9600 baud, 8 databit, geen pariteit, 1 stopbit, geen interrupt werking
    UCSR0A &=~(1<<U2X0); //normale snelheid
    UBRR0 =23; //Instellen van Baudrate op 9600bps
    UCSR0C |= ((1<<UCSZ00)|(1<<UCSZ01)); //8 databits instellen
    UCSR0B &=~ (1<<UCSZ02); //8 databits instellen
    UCSR0C &=~ (1<<USBS0); //1 stopbit
    UCSR0C &=~(1<<UPM00)|(1<<UPM01); //geen pariteit
    UCSR0B |=((1<<RXEN0)|(1<<TXEN0)); //activeren zender en ontvanger
    UCSR0B |= (1<<RXCIE0); //activeren interrupt rx
}

void serieel_init1(void)
{
    //de pin PD2 als ingang (RX), de pin PD3 als uitgang (TX) met een hoog niveau
    DDRD &=~(1<<DDRD2);
    DDRD |=(1<<DDRD3);

    //init seriële communicatie
    //USART0, 9600 baud, 8 databit, geen pariteit, 1 stopbit, geen interrupt werking
    UCSR1A &=~(1<<U2X1); //normale snelheid
    UBRR1 =23; //Instellen van Baudrate op 9600bps
    UCSR1C |= ((1<<UCSZ10)|(1<<UCSZ11)); //8 databits instellen
    UCSR1B &=~ (1<<UCSZ12); //8 databits instellen
    UCSR1C &=~ (1<<USBS1); //1 stopbit
    UCSR1C &=~(1<<UPM10)|(1<<UPM11); //geen pariteit
    UCSR1B |=((1<<RXEN1)|(1<<TXEN1)); //activeren zender en ontvanger
    UCSR1B |= (1<<RXCIE1); //activeren interrupt rx
}
```

Interrupt USART0_RX_vect is de interrupt die geactiveerd wordt wanneer er data ontvangen wordt. In deze interrupt wordt alle data van de 2^{de} microcontroller ingelezen. Aangezien hier veel if's in staan kan deze interrupt volledig in de volledige code gevonden worden.

```
#ISR(USART0_RX_vect)
{
    char data = UDR0;
    //P1
    if(data==0x01 )
    {
        if(bezetteparkeerplaatsen[1]!=0)
        {
            bezetteparkeerplaatsen[1]=0;
            if(i2c1>0) i2c1-=1;
        }
    }
}
```

```

ISR(USART1_RX_vect)
{
    static unsigned char rx_ptr=0;
    rx_buf[rx_ptr]=UDR1;
    if(rx_buf[rx_ptr]=='\r')
    {
        rx_buf[rx_ptr]='\0';
        rx_ptr=0;
        msg=MSG_NEW;
    }
    else rx_ptr++;
}

```

Om data vanaf de pc te ontvangen wordt USART1_RX_vect gebruikt. De data van de C# applicatie wordt als strings verstuurd dus moet dit ook zo ingelezen worden. Hiervoor gebruik ik deze code.

Om deze strings te controleren wordt in de while(1) loop gecontroleerd of er al een nieuw bericht is binnengekomen. Daarna wordt met strstr de ingelezen string vergeleken en de nodige code uitgevoerd.

```

//Nieuwe seriele berichten verwerken
if(msg==MSG_NEW)
{
    PORTD |=(1<<PORTD7);
    msg=MSG_OLD;

    //ptr=strstr(rx_buf,"test");

    if(strstr(rx_buf,"slagboom1-0"))
    {
        slagboom1=0;
    }
    if(strstr(rx_buf,"slagboom1-1"))
    {
        slagboom1=1;
    }
    if(strstr(rx_buf,"slagboom1-2"))
    {
        slagboom1=2;
    }
}

```

Om data naar de C# applicatie te verzenden gebruik ik deze functie. Hierbij wordt ieder karakter van de string 1 voor 1 verzonden.

```

void sendChar1(char data)
{
    while( !(UCSR1A & 0x20) ); //Er word gewacht tot dat de vlag UDRE0 1 is en dus het register UDR0 leeg is
    UDR1 = data; //De te versturen data word in register UDR0 geplaatst
}

void sendString1(char s[])
{
    int i = 0;                //
    while(i < 64)             //Blijft lus doorlopen zolang i kleiner is dan 64 om niet over de maximum hoeveelheid data te gaan
    {
        if( s[i] == '\0' ) break;   //
        sendChar1(s[i++]);       //
    }
}

```

Berekeningen:

Om de seriële communicatie te kunnen gebruiken moet de baudrate berekend worden. Dit kan met de formule $UBBRn=fosc/16baud -1$.

Na deze berekening kom ik de waarde 23 uit om de baudrate op 9600 in te stellen.

Reflectie:

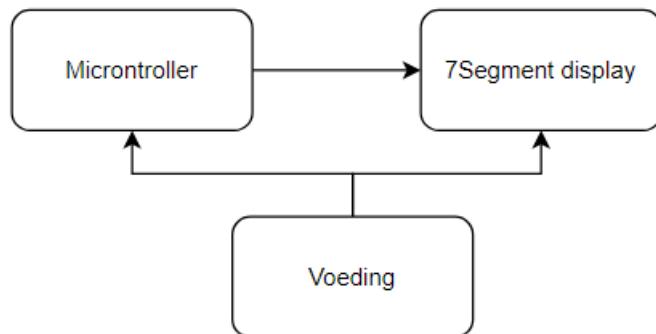
Er is een werkend programma om data te sturen en ontvangen.

- 7-segment

Doelstelling:

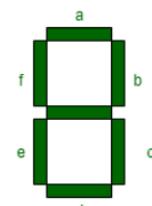
Het aansturen van 2 7-segment displays om het aantal vrij parkeerplaatsen te kunnen weergeven.

Blokschema:



Betekenis van de aansluitingen:

De twee 7-segment displays zijn op een bordje gemonteerd. Er zijn 7 pinnen aanwezig die elk een deeltje van de 7-segment aansturen. Via 2 andere pinnen op dit bordje wordt gekozen welke van de twee 7-segments displays wordt aangestuurd.



Stukje broncode om de werking uit te leggen:

```
ISR (TIMER0_COMPA_vect)
{
    PORTB ^= (1<<PORTB0)|(1<<PORTB1);
    if(linkrechts==1)
    {
        PORTC = array1[waarde_e];
        linkrechts=0;
    }
    else
    {
        PORTC = array1[waarde_t];
        linkrechts=1;
    }
}
```

Op de 2 pinnen PB0 en PB1 zijn de pinnen aangesloten om te wisselen van display. Deze wisselen via de timer. Afhankelijk van welk display aangestuurd wordt, wordt de correcte

waarde op PORTC aangestuurd. Alle mogelijke waarden zijn opgeslagen in een array om zo makkelijk de juiste waarde op te roepen.

Via de functie waarde7 kan ik de gewenste waarde voor de 7-segment display opsplitsen.

```
char waarde7(char waarde)
{
    waarde_e=waarde % 10;
    waarde_t=waarde /10;
}
```

Stel dat ik de waarde 23 stuur, dan zal in waarde_e het getal 3 worden opgeslagen en in waarde_t het getal 2. Deze variabelen kunnen door het volledige programma gebruikt worden en dus kunnen deze waarden in de interrupt van de timer gebruikt worden.

Berekeningen:

Om het getal op te splitsen in 10-tallen en eenheden gebruik ik de modulo. De modulo 10 zal de waarde delen door 10 en de rest wordt opgeslagen in het variabele waarde_e. Om de tientallen op te slaan in waarde_t wordt gewoon de waarde gedeeld door 10.

Technisch functionele beslissingen naar werkelijke realisatie:

Om dit werkelijk te kunnen aansturen zal er nog een aanpassing moeten gebeuren. De I2C modules die ik wil gebruiken, moeten aangesloten worden op PortC waar ook de 7-segment module wordt aangesloten. Daarom zal ik een de 7-segment display op een andere poort moeten aansluiten.

Print:

De 7-segment modules worden op een print gemonteerd met alle nodige weerstanden en transistoren. In bijlage staat het schema voor deze print.

- Aansluiten sensoren

Doelstelling:

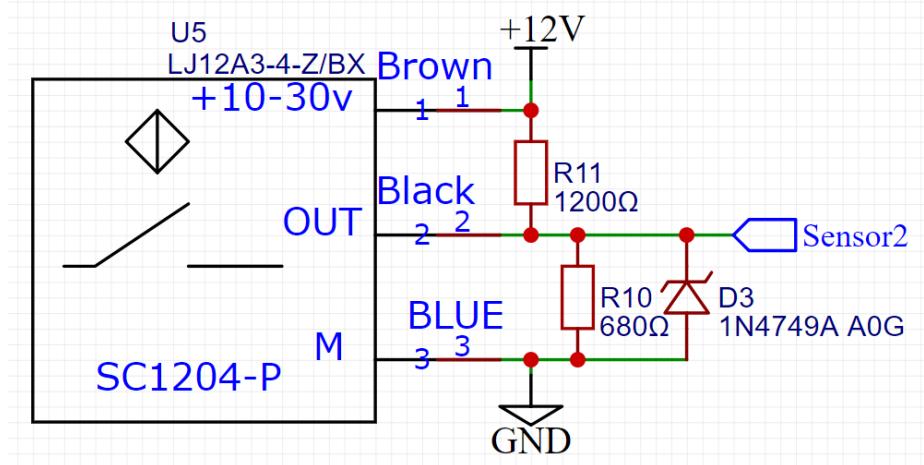
De doelstelling is om een sensor aan te sluiten voor de detectie van een wagen voor de slagbomen of op een parkeerplaats.

Keuze sensor:

In werkelijkheid gebruikt men een inductielus in de grond. Deze worden geplaatst waarna er dan beton over wordt gegoten. Bij een slagboom zijn er meerdere lussen gelegd zodat kan gecontroleerd worden wanneer een wagen toekomt en weg gaat, zodat de slagboom niet sluit wanneer er een wagen onder staat.

Een andere manier is om gebruik te maken van een ultrasone sensor. Ultrasone sensoren maken gebruik van geluidsgolven om afstand te meten. De sensor stuurt een ultrasoon geluidssignaal uit en meet de tijd die het signaal kost om terug te keren nadat het is weerkaatst op een object. Hierdoor kan de afstand tot het object worden berekend aan de hand van de tijd die het signaal heeft gekost om heen en weer te gaan. Zo kan ook gedetecteerd worden of er een wagen aanwezig is. De sensor die ik in mijn GIP zal gebruiken is de IJ12A3-4-Z/BX. Dit is een inductieve sensor die werkt met een NPN transistor. Het kan metalen detecteren op 4mm.

Schema:



Betekenis van de aansluitingen:

De sensor zelf heeft 3 aansluitingen: VCC op de bruine draad, GND op de blauwe kabel en OUT op de zwarte kabel. Om deze sensor op de microcontroller te kunnen aansluiten, moeten eerst weerstanden geplaatst worden (dit wordt later uitgelegd). Daarna is er 1 connector om op de microcontroller aan te sluiten.

Berekeningen:

$$R1 = 12 / 10 \text{mA} = 12 / (10 * 10^{-3}) = 1200\Omega$$

$$UR2 = 4.5V$$

$$UR1 = 12V - 4.5V = 7.5V$$

$$I = 7.5 / 1200 = 6.25 \text{mA}$$

$$R2 = 4.5 / (6.25 * 10^{-3}) = 4500 / 6.25 = 720\Omega \Rightarrow E12reeks = 680\Omega$$

Code:

Om deze sensor te gebruiken in een programma moet omgekeerd gedacht worden. Wanneer er detectie is, zal er een laag niveau gedetecteerd worden, wanneer er geen detectie is, is er een hoog niveau.

Metingen:

De uitgangsspanning van deze schakeling is 4.6V wanneer er geen detectie is. Wanneer er wel detectie is, zal er een spanning van 0.7V zijn.

Meetresultaten toelichten:

Deze resultaten tonen dat de sensor nu perfect kan aangesloten worden op de microcontroller.

Reflectie of conclusie over de haalbaarheid:

De sensor werkt samen met de microcontroller en kan correct worden aangesloten.

Technisch functionele beslissingen naar werkelijke realisatie:

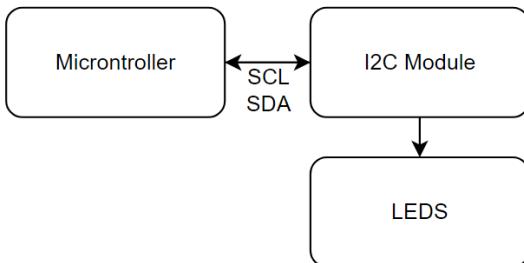
Omdat de sensor redelijk duur is, zal ik in mijn werkelijke opstelling een kleiner aantal sensoren gebruiken, de rest zal vervangen worden met gewone schakelaars.

- Aansluiten leds met I²C

Doelstelling:

Aansturen van leds via een I²C module.

Blokschema:

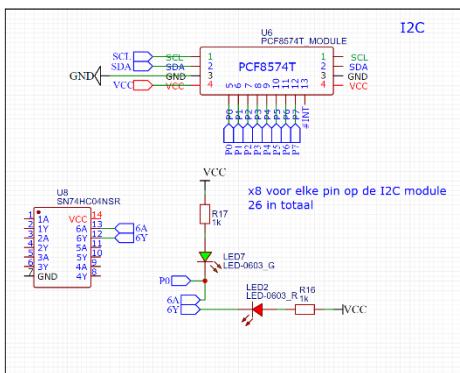


De I²C module wordt via de SCL en SDA pin van microcontroller aangestuurd.

Opstelling:

Om dit deelproject te realiseren heb ik gebruik gemaakt van een breadboard. Hierop heb ik de module en de verschillende leds geplaatst.

Schema:



Stukje broncode om de werking uit te leggen:

Om de module aan te sturen maak ik gebruik van een library. Een eerste nodige functie is de `twi_start()` functie. Dit is nodig om de communicatie tussen de module en de microcontroller te starten. Daarna moet het adres doorgegeven worden. Dit gebeurt met de functie `twi_write(adres);` en dit zal ervoor zorgen dat ik uitgangen kan aanpassen. Daarna moet terug de functie `twi_write()` gebruikt worden om de uitgang aan te sturen. Aangezien er een omgekeerde werking is, moet de waarde omgedraaid worden door 255-waarde te doen. Daarna moet de communicatie weer gesloten worden door `twi_close();` uit te voeren. Om alles makkelijker te maken heb ik een funtie geschreven die het gewenste adres en de waarde ontvangt.

Daarna kan de functie opgeroepen worden.

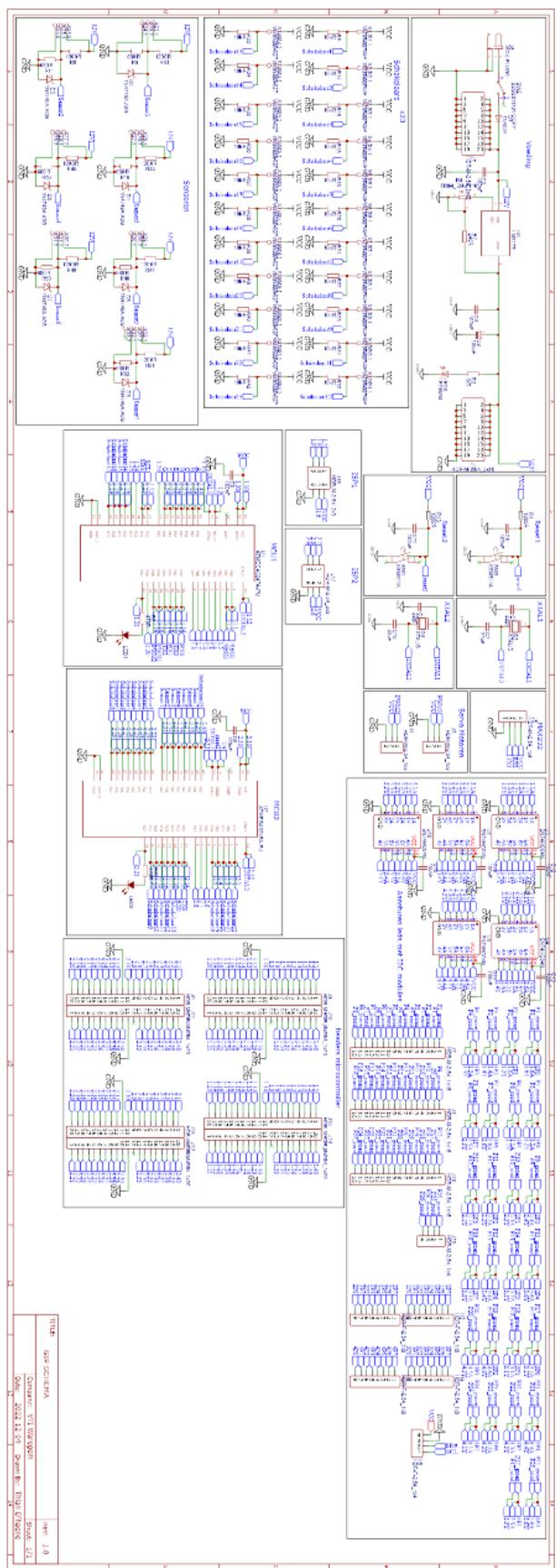
aansturen(0x40, 0);

```

void aansturen(char adres, char waarde)
{
    twi_start();
    twi_write(adres);
    twi_write(255-waarde);
    twi_stop();
}
  
```

7. Ontwerp hardware

- Schema

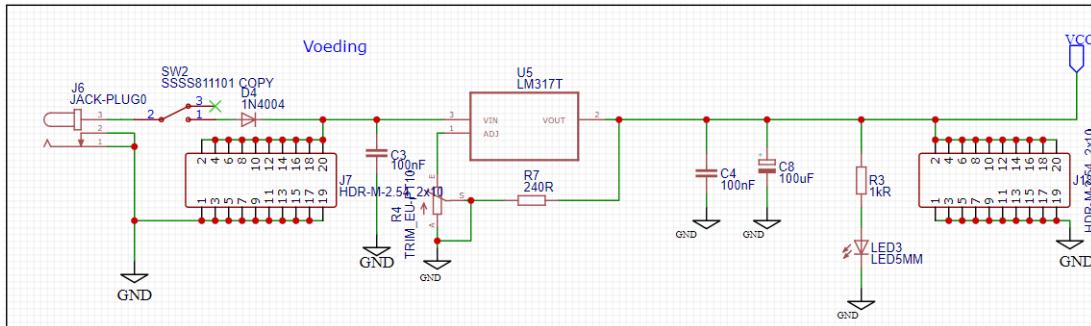


-Bespreking schema

Voeding

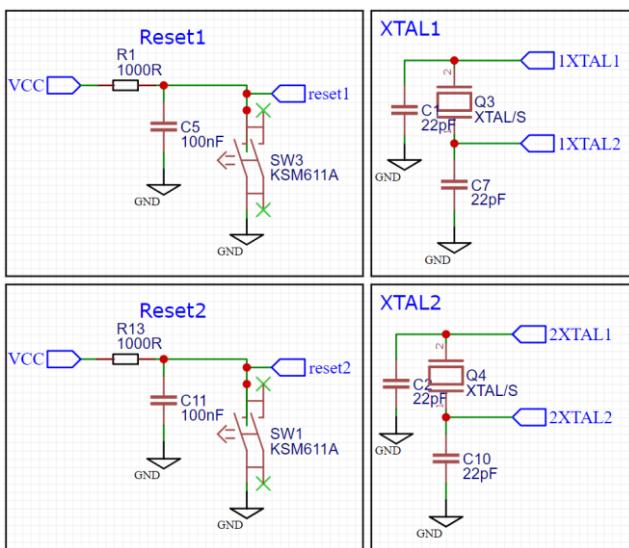
Het schema voor de print start bij de voeding. Hiervoor heb ik het voedingsschema gebruikt die ook gebruikt is voor het bordje dat we van mr. Van Quickelberghe hebben gekregen. De voeding zorgt ervoor dat de 12 V die binnenkomt van de adapter, omgezet wordt naar 5 V door gebruik te maken van de spanningsregelaar LM317. De spanning kan hierbij via een regelbare weerstand eenmalig aangepast worden naar 5V. De condensatoren zorgen ervoor dat onregelmatigheden weg gefilterd worden.

Hierbij heb ik er ook voor gezorgd dat er zeker voldoende headers aanwezig zijn om indien nodig extra aansluiting te kunnen doen met 12 V en 5 V.



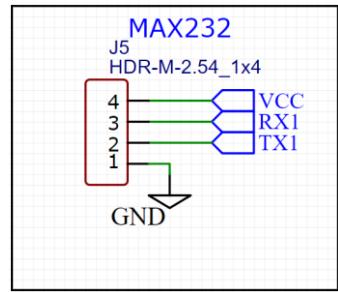
XTAL en Reset

Beide microcontrollers zijn aangesloten aan hun eigen reset. Wanneer de microcontrollers geprogrammeerd worden via de ISP worden ze gereset, maar ik heb ervoor gezorgd dat er ook een handmatige reset is via een drukknop. Ook worden beide microcontrollers aangesloten op hun eigen kristal. Dit zorgt ervoor dat er een kloksnelheid van 3.6864 MHz is.



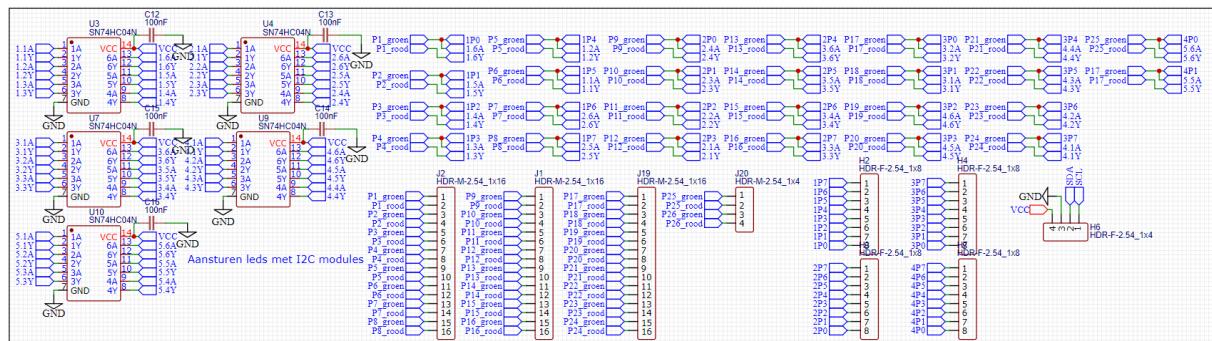
Seriële Communicatie

Om te communiceren met de pc maak ik gebruik van de MAX232. Dit is een IC die een TTL signaal omzet naar een RS-232 signaal. Deze IC is bevestigd op een module die kan aangesloten worden op de USB poort van de pc. Aan de andere zijde van de module wordt VCC en GND aangesloten en worden ook TX en RX op de pinnen van de microcontroller aangesloten. Hiervoor zijn headers voorzien.



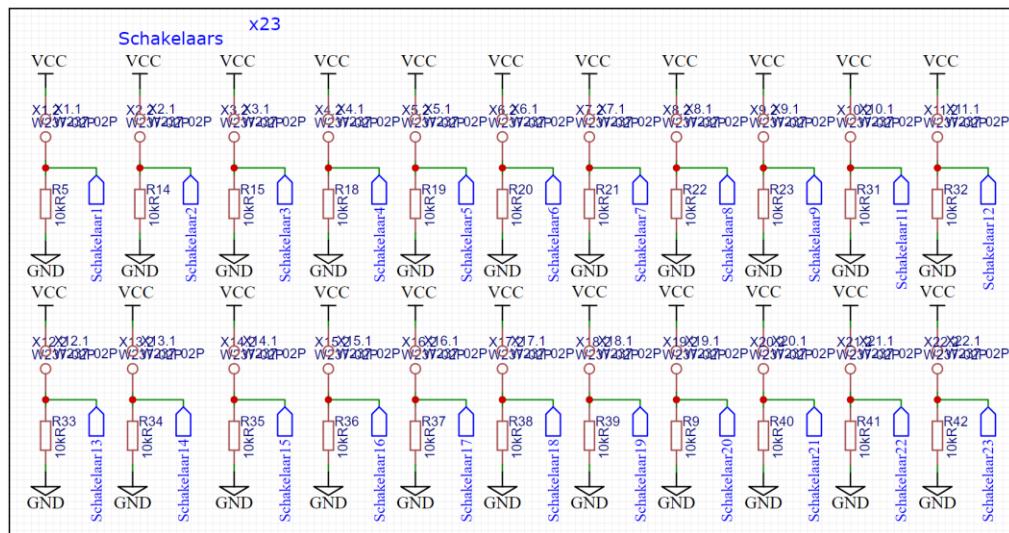
I2C en 74HC04

Om alle Leds te kunnen aansturen maak ik gebruik van I2C modules PCF8574_PCF8574A. Deze modules hebben elk een adres dat de microcontroller kan gebruiken om een specifieke pin aan te sturen. Om pinnen te besparen maak ik gebruik van 74HC04 modules. Dit zijn niet-poorten of inverterende poorten. Hierdoor kan de groene of de rode led altijd branden.



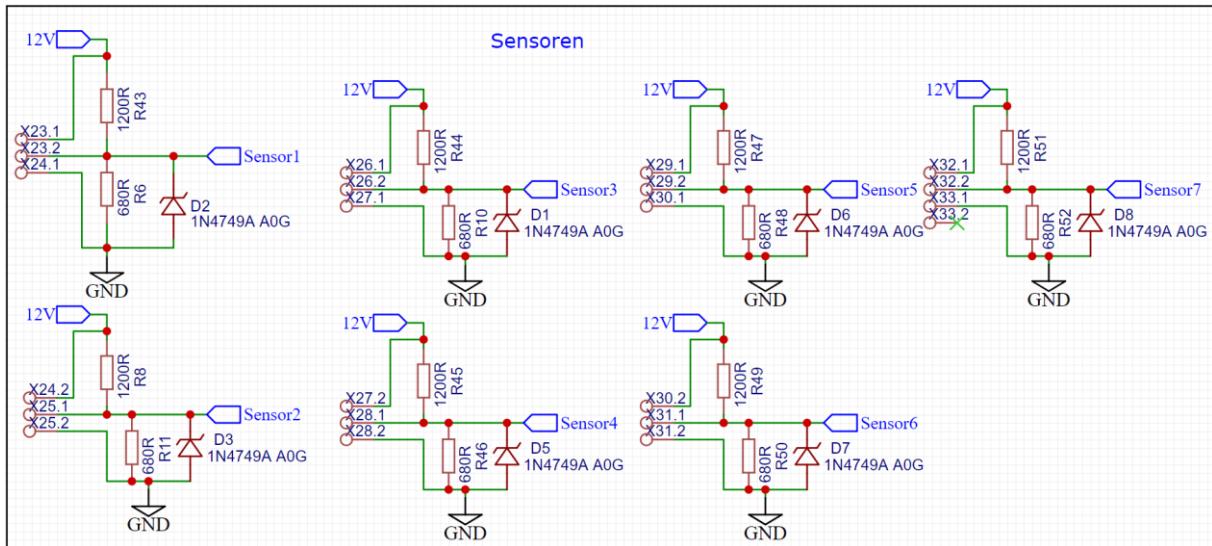
Schakelaars

Omdat de sensoren te veel zouden kosten maak ik gebruik van schakelaars. Deze worden aangesloten volgens het schema dat hieronder gezien kan worden. De sensoren worden aangesloten op schroefconnectoren.



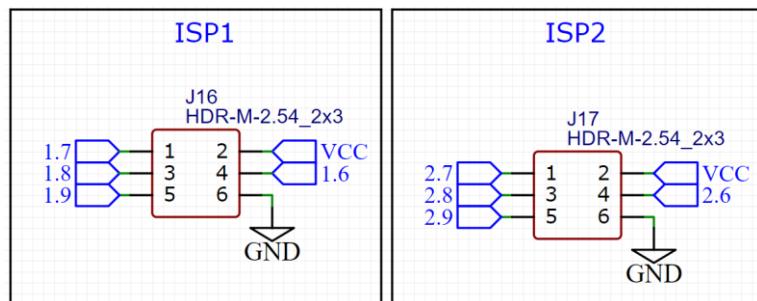
Sensoren

De sensoren worden ook aangesloten via schroefconnectoren. De weerstanden worden aangesloten zoals in het deelproject van de sensoren is berekend en uitgelegd. Ook is er een zener-diode.

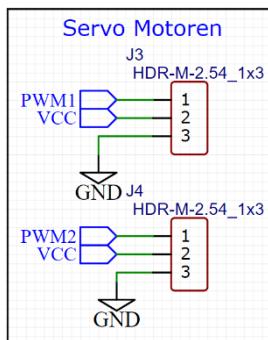


ISP's

De ISP's voor beide microcontrollers bestaan uit 2 rijen van 3 headers naast elkaar. Hiermee kan de microcontroller geprogrammeerd worden. Dit wordt gewoon rechtstreeks verbonden op de microcontroller.



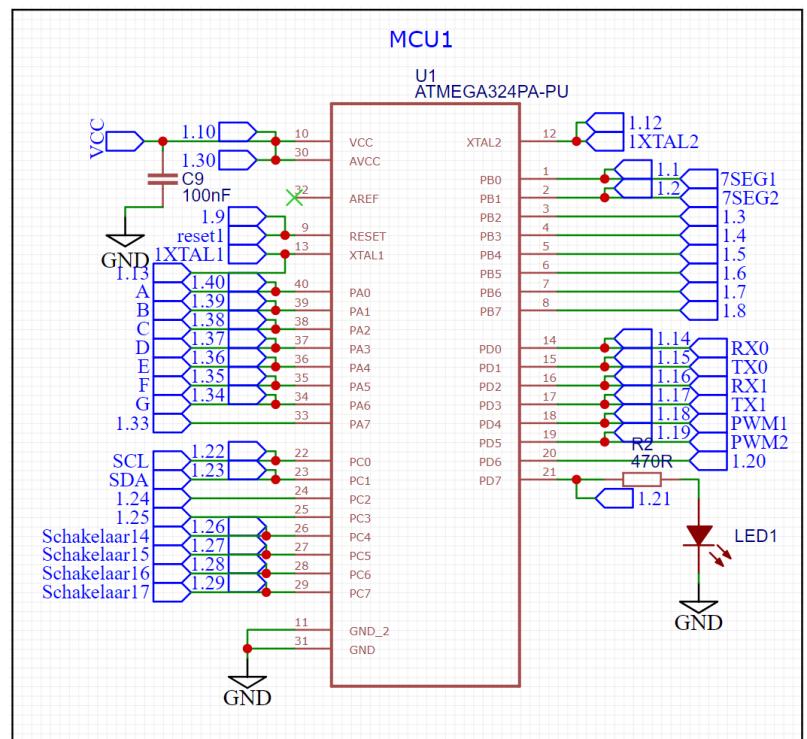
Servomotoren



De servomotoren worden ook aangesloten op headers. De verbinding met de microcontroller gebeurt ook rechtstreeks.

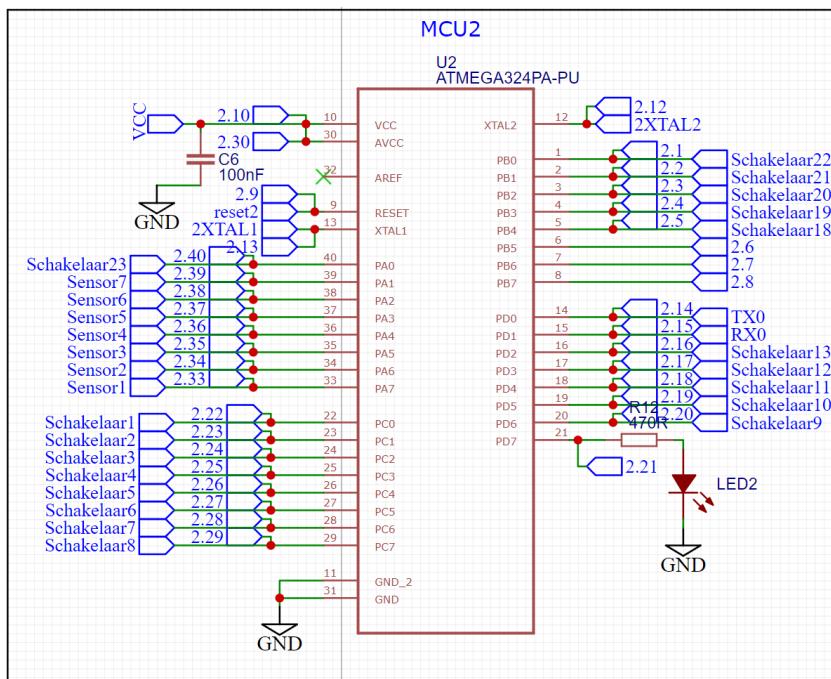
Microcontroller 1

De eerste microcontroller gebruik ik om 7-segment displays aan te sturen. De pinnen A-G van de 7-segment displays sluit ik aan op PORTA. De keuze tussen welk van de 2 displays wordt aangestuurd, kan gebeuren via pin 1 en 2. Pin 14 en 15 worden verbonden aan de andere microcontroller voor seriële communicatie. Op pin 16 en 17 wordt dan de seriële communicatie aangesloten via de MAX232 module. Pin 18 en 19 wordt gebruikt om de servomotoren aan te sturen. Omdat er pinnen te kort zijn op de 2^{de} microcontroller om de ISP aan te sluiten, moeten enkele schakelaars aangesloten worden op de eerste. Ook de I2C modules worden op deze microcontroller aangesloten. Dit is op pin 22 en pin 23. Voor de rest worden VCC, GND de reset en het kristal aangesloten. Ook is er nog een controle-led voorzien.



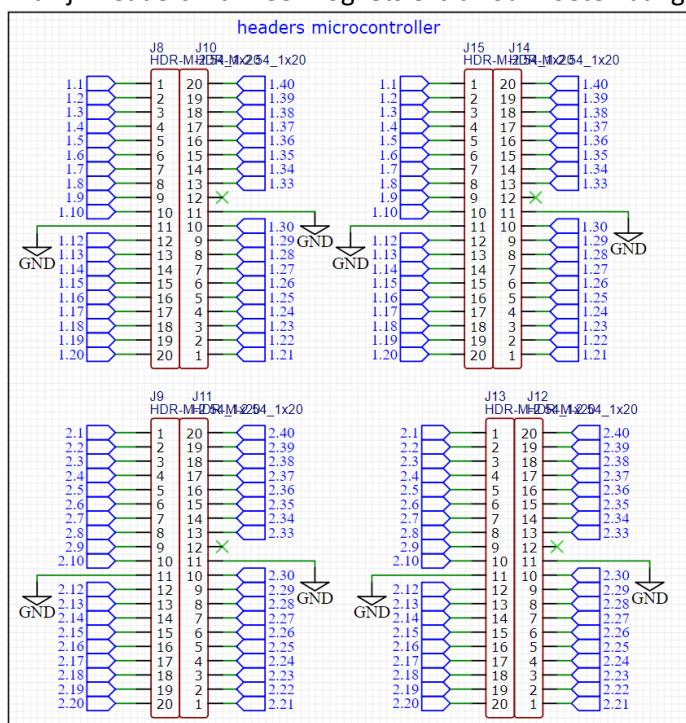
Microcontroller 2

De tweede microcontroller wordt gebruikt om alle sensoren en schakelaars in te lezen. Hierbij is opnieuw een controle-led voorzien. Ook VCC, GND, reset en het kristal zijn aangesloten.



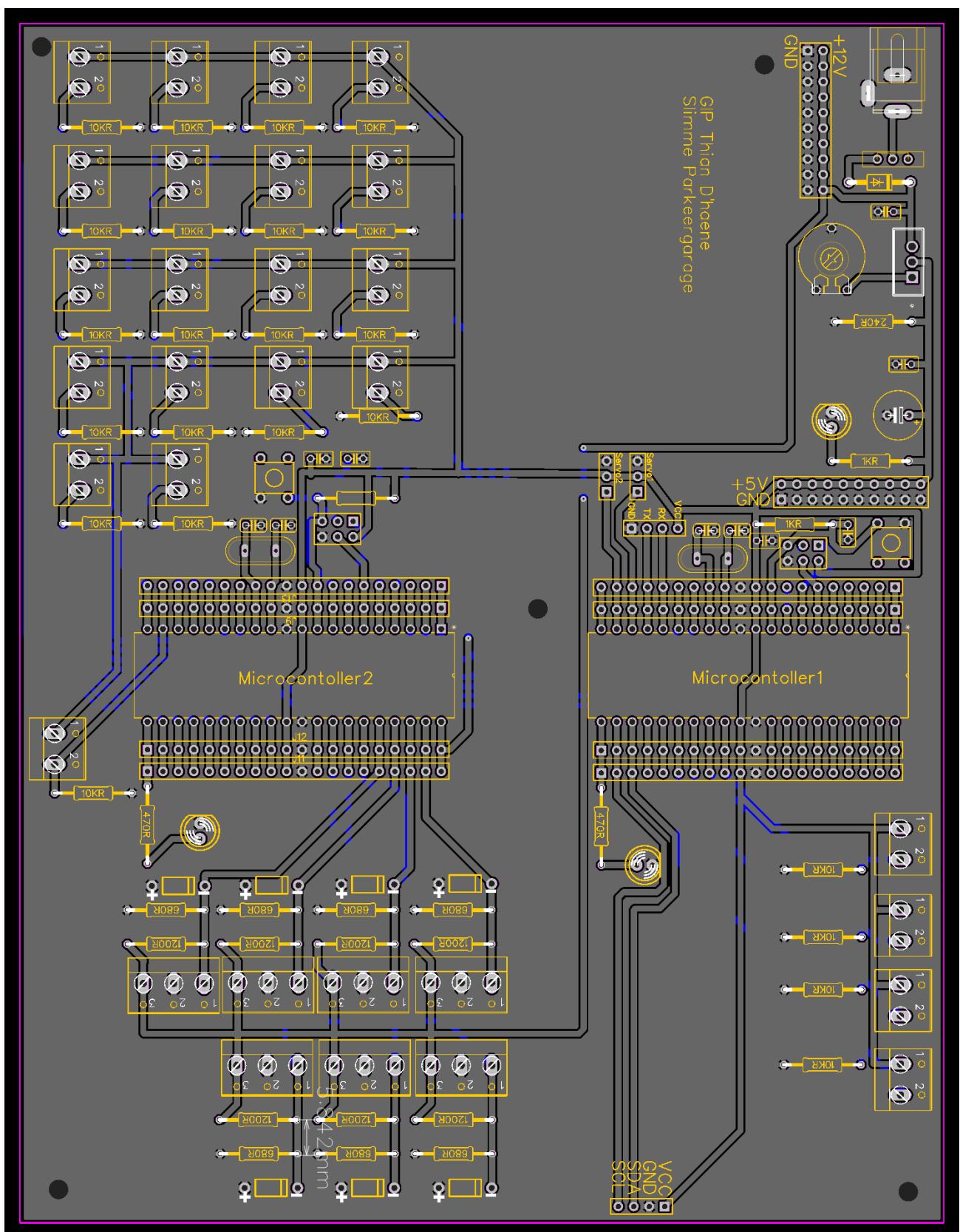
Headers

Dit zijn headers wanneer nog iets extra moet worden aangesloten op de microcontrollers.

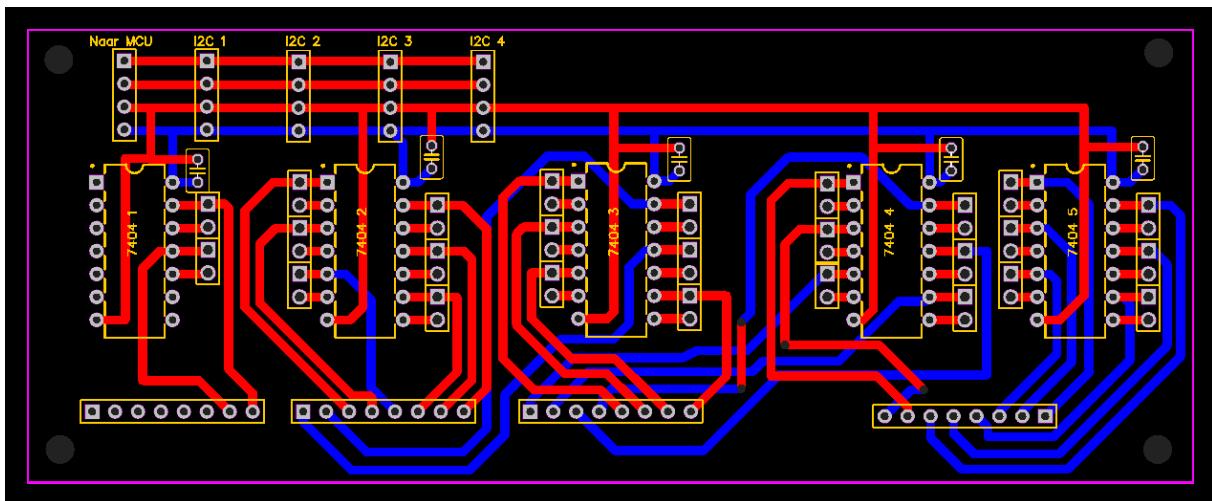


- Print

Print 1



Print 2



8. Applicaties

In dit hoofdstuk van mijn GIP dossier zal ik al mijn software uitleggen. Dit houdt in dat de software voor de eerste en tweede microcontroller wordt uitgelegd en de software voor de C# applicatie.

Initialisatie Microcontrollers

Eerst moeten verschillende libraries toegevoegd worden aan het programma. Een van de belangrijke libraries voor de eerste microcontroller is die voor het aansturen van de I2C modules. Andere libraries zijn nodig om bijvoorbeeld bewerkingen met strings te doen of om berekeningen te kunnen maken.

```
//INIT AANSTUREN SLAGBOMEN
char Servo1(unsigned char graden);
char Servo2(unsigned char graden);
void init_servo();
char slagboom1 = 0;
char slagboom2 = 0;

//INIT AANSTUREN 7SEGMENT
volatile unsigned char waarde_e;
volatile unsigned char waarde_t;
volatile unsigned char linksrechts;
volatile char array7[]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
void init_7seg(void);
char waarde7(char waarde);

//AANSTUREN I2C
void I2C(char adres, char );
volatile unsigned char i2c1 = 0;
volatile unsigned char i2c2 = 0;
volatile unsigned char i2c3 = 0;
volatile unsigned char i2c4 = 0;

//DOORSTUREN STATUS PARKING
volatile unsigned char ticks4s;
volatile unsigned char ticks64;
char bezetteparkeerplaatsen[27];
char buffer[5];
char bezetteplaatsen=0;

//INIT FUNCTIES USART0
void serieel_init0(void);
void sendChar0(char data);
void sendString0(char s[]);

//INIT FUNCTIES USART1
void serieel_init1(void);
void sendChar1(char data);
void sendString1(char s[]);

//INIT ONTVANGEN VIA C#
#define MSG_NEW 1
#define MSG_OLD 2
char rx_buf[160];
volatile unsigned char msg=MSG_OLD;
```

De tweede microcontroller wordt enkel gebruikt om de verschillende ingangen in te lezen en heeft dus veel minder functies.

```
//Toevoegen van alle libraries
#include <avr/io.h>
#define F_CPU 3686400L
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdbool.h>
#include "twi_master.h"
#include <stdlib.h>
#include <avr/pgmspace.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
```

Na het toevoegen van de libraries moeten eerst alle functies worden geïnitialiseerd. Dit zorgt ervoor dat deze functies verder in het programma gebruikt kunnen worden. Bij het initialiseren hiervan moeten bijvoorbeeld de meegegeven variabelen ingesteld worden of welk type de return waarde moet zijn. Aangezien de eerste microcontroller veel verschillende onderdelen moet aansturen, zijn hier veel functies nodig .

```
//INIT SERIEEL
void serieel_init0(void);
void sendString0(char s[]);
void sendChar0(char data);
char OmzettenNaarHex(char i, char status);

//INIT FUNCTIE INGANGEN
void init_ingangen(void);

//ARRAY BEZETTEPARKEERPLAATSEN
char bezetteparkeerplaatsen[27];

//INIT TIMER
volatile unsigned char ticks1s;
volatile unsigned char ticks16;
void init_timer(void);

//VARIABELEN SLAGBOMEN
char slagboom1 = 1;
char slagboom2 = 1;
```

Main Microcontrollers

Nadat alles is geïnitialiseerd kan de main betreden worden. Hier moeten de functies voor de verschillende onderdelen op te starten een eerste keer worden gebruikt. Bij de eerste microcontroller moeten er ook nog enkele ingangen en uitgangen worden ingesteld voor de sensoren en om gebruik te kunnen maken van de status-led.

Bij beide microcontrollers zorg ik er ook voor dat de status led knippert bij het opstarten, waardoor bij een fout direct duidelijk is dat de microcontroller herstart.

```
//Statusled knipperen
for (int i = 0; i < 10; i++) {
    PORTD |= (1<<PORTD7);
    _delay_ms(200);
    PORTD &=~(1<<PORTD7);
    _delay_ms(200);
    i++;
}
```

```
int main(void)
{
    //CONTROLE-LED ALS UITGANG
    DDRD |= (1<<DDRD7);

    //PORTA ALS UITGANG VOOR 7-SEGMENT
    DDRA = 0xFF;

    //PB0 EN PB1 ALS INGANG VOOR MULTIPLEXEN
    //PB0 LAAG EN PB1 HOOG OM TE STARTEN
    DDRB=(1<<DDRB0)|(1<<DDRB1);
    PORTB&=~(1<<PORTB0);
    PORTB|= (1<<PORTB1);

    //INIT 4 PARKEERPLAATSEN MCU1
    //P14 P15 P16 P17
    DDRC &=~ (1<<DDRC4)|(1<<DDRC5)|(1<<DDRC6)|(1<<DDRC7);

    //OPSTARTEN VERSCHILLENDEN COMPONENTEN
    //SERVOMOTOREN
    init_servo();
    //USART0 & USART1
    serieel_init0();
    serieel_init1();
    //I2C
    twi_init();
    //7-SEGMENT
    init_7seg();

    //SLAGBOMEN NAAR STANDAARD POSITIE ZETTEN
    Servo1(0);
    Servo2(90);
    _delay_ms(200);
    Servo1(90);
    Servo2(180);
}
```

While(1) microcontrollers

Nadat alles correct is opgestart kan de while(1) lus betreden worden. Hierin zal alles blijven herhaald worden tot de microcontroller zou worden uitgeschakeld. In het programma van de eerste microcontroller worden hier de 4 schakelaars die hierop zijn aangesloten ingelezen. Om de bijhorende positie van deze parkeerplaats in de array aan te passen wordt deze code gebruikt (voorbeeld van 1 parkeerplaats, dit is voor alle 25 andere parkeerplaatsen hetzelfde).

```
if (ticks4s)
{
    bezetteplaatsen=0;
    for (int i = 1; i <= 26; i++)
    {
        if(bezetteparkeerplaatsen[i]==1)
        {
            bezetteplaatsen++;
            sprintf(buffer, "PB%d\r\n",i);
        }
        if(bezetteparkeerplaatsen[i]==0)
        {
            sprintf(buffer, "PL%d\r\n",i);
        }
        sendString1(buffer);
        _delay_ms(20);
    }
    sprintf(buffer, "B%d\r\n",bezetteplaatsen);
    sendString1(buffer);
    ticks4s=0;
    waarde7(26-bezetteplaatsen);
}
```

```
//P14
if(!(PINC &(1<<PIN4)))
{
    if(bezetteparkeerplaatsen[14]!=0)
    {
        bezetteparkeerplaatsen[14]=0;
        if(i2c2>=32) i2c2-=32;
    }
}
if(PINC &(1<<PIN4))
{
    if(bezetteparkeerplaatsen[14]!=1)
    {
        bezetteparkeerplaatsen[14]=1;
        i2c2+=32;
    }
}
```

Om met de eerste microcontroller alle data serieel te versturen naar het C# programma maak ik gebruik van een for lus. Deze lus zal 26 keer doorlopen worden waardoor vanuit de array alle statussen van de parkeerplaatsen worden gecontroleerd, deze waarden worden omgezet naar een string en daarna de string serieel verstuurd wordt naar het C# programma. Ook wordt hierbij de waarde van het 7-segment display aangepast naar het momenteel aantal beschikbare parkeerplaatsen.

Bij de tweede microcontroller worden constant alle ingangen ingelezen. Afhankelijk van de status van de ingang zal de overeenkomende waarde in de array aangepast worden.

```
//Inlezen alle ingangen
//P1
if(PINA &(1<<PIN4)) {bezetteparkeerplaatsen[1]=0; }
if(!(PIN4 &(1<<PIN4))){bezetteparkeerplaatsen[1]=1; }

//P2
if(PINA &(1<<PIN4)) {bezetteparkeerplaatsen[2]=0; }
if(!(PIN4 &(1<<PIN4))){bezetteparkeerplaatsen[2]=1; }
```

In de while(1) lus van de tweede microcontroller worden ook om de seconde alle statussen van alle parkeerplaatsen serieel verstuurd naar de eerste microcontroller. Hierbij wordt gebruik gemaakt van een aparte functie die afhankelijk van de momenteel te verzenden parkeerplaats een hexadecimale waarde zal doorgeven om te verzenden.

```
if(ticks1s)
{
    for (int i = 0; i <= 26; i++)
    {
        sendChar0(OmzettenNaarHex(i,bezetteparkeerplaatsen[i]));
        _delay_ms(20);
    }
    ticks1s=0;
}
```

Functies

Om alle onderdelen te kunnen aansturen maak ik gebruik van verschillende zelfgeschreven functies. Een eerste functie is een functie om de slagbomen te kunnen aansturen. In het programma moet dan enkel het aantal graden van 1-360 gestuurd worden naar deze functie en zal dan de servomotor zichzelf positioneren naar deze hoek. Van deze functie is er ook een 2^{de} versie voor de 2^{de} slagboom.

```
char Servo1(unsigned char graden)
{
    OCR1A=(231+(graden*5.12222));
    return 1;
}
```

Om de I2C module aan te sturen maak ik ook gebruik van een zelfgeschreven functie. Wel maakt deze functie gebruik van een library, maar door deze functie kan mijn code een heel stuk koper geschreven worden. In deze functie wordt eerst de communicatie met de module gestart en wordt daarna de juiste module benaderd via het adres. Daarna wordt de juiste waarde doorgestuurd en wordt de communicatie gesloten.

```
void I2C(char adres, char waarde)
{
    twi_start();
    twi_write(adres);
    twi_write(255-waarde);
    twi_stop();
}
```

Bij de 2^{de} microcontroller is er slechts 1 functie die afhankelijk van de geselecteerde parkeerplaats een hex waarde zal terug sturen om naar de eerste microcontroller te sturen.

C# programma

Dan is er nog het C# programma. Hier worden bij het opstarten ook verschillende variabelen aangemaakt en ingesteld. Eerst zullen verschillende aanpassingen gebeuren aan de grote van het formulier zodat alles netjes gepositioneerd staat afhankelijk van het scherm. Daarna worden alle beschikbare seriële poorten in een array van strings geplaatst. Deze strings worden dan toegevoegd aan een lijst waarin de gebruiker kan kiezen welke seriële poort moet gebruikt worden. In de 2^{de} foreach wordt ervoor gezorgd dat van alle labels de parent wordt ingesteld om de pictureBox van de parking. Nadat dit gebeurd is kan de functie InitializeLabels() opgeroepen worden. Deze functie wordt hieronder uitgelegd.

```
private void Form1_Load(object sender, EventArgs e)
{
    this.WindowState = FormWindowState.Maximized;
    pbParking.SizeMode = PictureBoxSizeMode.Zoom;
    lstPoort.Width = pnlConnect.Width / 3;
    lstConsole.Width = pnlConnect.Width - lstPoort.Width;
    //alle seriële poorten voor dit toestel opvragen
    string[] ports = System.IO.Ports.SerialPort.GetPortNames();
    foreach (string poort in ports)
    {
        lstPoort.Items.Add(poort);
    }
    foreach (Label label in pbParking.Controls.OfType<Label>())
    {
        label.Parent = pbParking;
    }
    InitializeLabels(); //oproepen functie om alle labels op te slaan in de map
}
```

Om ervoor te zorgen dat elk label makkelijk kan benaderd worden in de code, wordt in een foreach aan elk label een tag toegewezen. Al deze tags worden dan opgeslagen in de Dictionary labelMap.

```
//InitializeLabels
//InitializeLabels zal van alle labels van elk parkeervak het nummer opslaan zodat automatisch het nummer later kan gebruikt worden
private Dictionary<int, Label> labelMap = new Dictionary<int, Label>();
private void InitializeLabels()
{
    //Eerst wordt een tag toegewezen aan elk label waarna dit wordt opgeslagen in een map
    foreach (Label label in pnlParkingEnStatus.Controls.OfType<Label>())
    {
        int number = int.Parse(label.Name.Substring(1));
        label.Tag = number;
        labelMap[number] = label;
        //label.Text = label.Tag.ToString();
    }
}
```

Nadat het programma volledig is opgestart kan de seriële poort geselecteerd worden. Dit gebeurt in het event SelectedIndexChanged van de lijst waar in het begin van het programma alle beschikbare poorten aan worden toegevoegd. De geselecteerde poort zal worden opgestart en indien dit succesvol gebeurt kan de seriële communicatie gebruikt worden. Wanneer hierbij een fout zou optreden zal dit worden opgevangen en zal een error getoond worden.

```
//Selecteren seriële poort
//Dit event wordt opgeroepen wanneer een keuze voor een seriële poort wordt gemaakt,
//hierna wordt indien mogelijk de verbinding opgezet
private void lstPoort_SelectedIndexChanged(object sender, EventArgs e)
{
    if (lstPoort.SelectedIndex == -1) return;
    string poort = lstPoort.SelectedItem.ToString();
    if (poort == serial.PortName) return; //zelfde poort geselecteerd
    serial.Close();
    try
    {
        serial.PortName = poort;
        serial.Open();
        if (serial.IsOpen == true)
        {
            lstConsole.Items.Insert(0, "Nieuwe poort geopend: " + poort);
            lstConsole.Items.Insert(0, "Start de garage op en wacht op bevestiging");
            lstPoort.Enabled = false;
        }
    }
    catch (System.Exception ex)
    {
        MessageBox.Show("Error: " + ex.Message);
        lstConsole.Items.Add(ex.Message.ToString());
    }
}
```

Wanneer de verbinding succesvol is opgestart kan er seriële data ontvangen worden van de microcontroller. In het event DataReceived() worden alle mogelijke serieel ontvangen strings gecontroleerd. Wanneer er een goede verbinding is zal dit getoond worden op de applicatie doormiddel van onderstaande code.

```
if (data.Contains("PARKING-OPGESTART"))
{
    tsConnectie.Text = "Verbonden";
    tsConnectie.ForeColor = Color.Green;
}
```

```

else if (data.StartsWith("PB")) //Parking.Bezet
{
    int parkeerplaats = int.Parse(data.Substring(2));
    if (parkeerplaatsen[parkeerplaats] != 1)
    {
        parkeerplaatsen[parkeerplaats] = 1;
        if (File.Exists(logboekLocatie))
        {
            StreamWriter streamwriter = new StreamWriter(logboekLocatie, true);
            streamwriter.WriteLine("[" + DateTime.Now.ToString() + "] " + parkeerplaats.ToString() + " BEZET");
            streamwriter.Close();
        }
    }
    AanpassenKleur(parkeerplaats, Color.Red);
}
else if (data.StartsWith("PL")) //Parking.Leeg
{
    int parkeerplaats = int.Parse(data.Substring(2));
    if (parkeerplaatsen[parkeerplaats] != 0)
    {
        parkeerplaatsen[parkeerplaats] = 0;
        if (File.Exists(logboekLocatie))
        {
            StreamWriter streamwriter = new StreamWriter(logboekLocatie, true);
            streamwriter.WriteLine("[" + DateTime.Now.ToString() + "] " + parkeerplaats.ToString() + " LEEG");
            streamwriter.Close();
        }
    }
    AanpassenKleur(parkeerplaats, Color.Green);
}

```

Wanneer er PB of PL ontvangen wordt wil dit zeggen dat de kleur van een van de labels aangepast moet worden. Dit gebeurt in deze if's. Ook wordt het momentele tijdstip samen met het nummer van de parkeerplaats opgeslagen in het log bestand. Via de functie AanpassenKleur wordt de kleur aangepast. Deze functie gebruikt het nummer van de parkeerplek en zal dan de eerder aangemaakt Directory labelMap gebruiken om het bijhorende label van kleur aan te passen.

```

//Aanpassen.Kleur.Labels
//Deze functie zorgt ervoor dat het label gekoppelt aan het nummer van de
//parkeerplaats veranderd van kleur afhankelijk of de parking bezet is of niet
private void AanpassenKleur(int nummer, Color color)
{
    if (labelMap.TryGetValue(nummer, out Label label))
    {
        label.BackColor = color;
    }
}

```

Om het logboek te kunnen gebruiken moet natuurlijk eerst het bestand gekozen worden. Dit gebeurt met een OpenFileDialog venster. Wanneer er succesvol een correct bestand is geselecteerd zal de locatie worden opgeslagen zodat deze op andere locaties in het programma gebruikt kan worden. Ook zal het tijdstip waarop het bestand gekozen is toegevoegd worden aan het logboek.

```
string logboekLocatie;
private void tsLogBoekLocatie_Click(object sender, EventArgs e)
{
    using (var fbd = new OpenFileDialog())
    {
        fbd.Filter = "txt files (*.txt)|*.txt";
        fbd.Title = "Kies of maak een .txt bestand";
        DialogResult result = fbd.ShowDialog();
        if (result == DialogResult.OK && !string.IsNullOrWhiteSpace(fbd.ToString()))
        {
            logboekLocatie = (fbd.FileName.ToString());
            System.Windows.Forms.MessageBox.Show("Logboek wordt opgeslagen in" + logboekLocatie);
            if (File.Exists(logboekLocatie))
            {
                StreamWriter streamwriter = new StreamWriter(logboekLocatie, true);
                streamwriter.WriteLine("[" + DateTime.Now.ToString() + "] " + "LOGBOEK GEOPEND");
                streamwriter.Close();
            }
        }
    }
}
```

Ook is er nog een knop aanwezig om de nummers van de parkeerplaatsen weer te geven op de afbeelding van de parkeergarage.

```
private void tsWeergevenNummer_Click(object sender, EventArgs e)
{
    //private Dictionary<int, Label> labelMap = new Dictionary<int, Label>();
    if (toggleNumbers == false)
    {
        foreach (Label label in pnlParkingEnStatus.Controls.OfType<Label>())
        {
            int number = int.Parse(label.Name.Substring(1));
            label.Tag = number;
            label.Text = " " + label.Tag.ToString() + " ";
        }
        toggleNumbers = true;
    }
    else
    {
        foreach (Label label in pnlParkingEnStatus.Controls.OfType<Label>())
        {
            int number = int.Parse(label.Name.Substring(1));
            label.Tag = number;
            label.Text = "     ";
        }
        toggleNumbers = false;
    }
}
```

GUI

Om alles netjes te kunnen bedienen is er natuurlijk ook een Graphical user interface waarin de gebruiker alles kan bedienen. In bijlage zijn hiervan enkele afbeeldingen aanwezig.

9. Componentenlijst & Kostprijsberekening

Naam	Aantal	Prijs per stuk	Totaal prijs
Atmega324P	2	€ 6,62	€ 13,24
DC Jack Connector Heba 21	1	€ 0,94	€ 0,94
Switch ESP101 Vertikal	1	€ 0,58	€ 0,58
Diode 1N4007	1	€ 0,02	€ 0,02
LM317	1	€ 0,79	€ 0,79
Resistor 270R	1	€ 0,03	€ 0,03
Trimmer	1	€ 0,26	€ 0,26
Condensator 100nF	11	€ 0,05	€ 0,55
Gepolariseerde condensator 100uF	1	€ 0,09	€ 0,09
Resistor 1kR	3	€ 0,16	€ 0,48
Led 5mm	3	€ 0,44	€ 1,31
Drukknop	2	€ 0,10	€ 0,20
Condensator 22pF	4	€ 0,02	€ 0,10
Kristal 3,6864MHz	2	€ 0,80	€ 1,60
Resistor 10kR	23	€ 0,64	€ 14,67
Resistor 680R	7	€ 0,22	€ 1,51
Resistor 1200R	7	€ 0,02	€ 0,17
Zener-diode	7	€ 0,39	€ 2,75
SN74HC04	5	€ 0,75	€ 3,75
Resistor 470R	2	€ 0,03	€ 0,05
Headers(per 10)	30	€ 0,20	€ 5,97
Heatsink voor LM317	1	€ 0,22	€ 0,22
Sensoren	7	€ 5,95	€ 41,65
I2C PCF8574	4	€ 6,95	€ 27,80
Servomotors	2	€ 5,15	€ 10,30
Schakelaars	23	€ 0,68	€ 15,64
Aansluitingskabels	1	€ 22,50	€ 22,50
PCB's	1	€ 99,38	€ 99,38
Montage materiaal MCU's	1	€ 32,45	€ 32,45
		Totaal:	€ 299,00

10. Besluit

Het werken aan mijn GIP heeft me veel geleerd over zelfstandigheid, planning en samenwerking. Vooral het plannen is voor mij een belangrijk werk punt, zodat ik niet telkens op het laatste moment nog iets moet afwerken. Ik heb ook geleerd hoe belangrijk het is om regelmatig te communiceren met mijn GIP Coördinator zodat er op tijd kon bijgestuurd worden. Ook het communiceren met medestudenten was belangrijk om te helpen denken bij problemen waar ik zelf niet uitkwam.

Technisch heb ik ook zeer veel bijgeleerd tijdens dit eindwerk. Ik heb geleerd hoe ik de I2C module kan aansturen, seriële communicatie tussen 2 microcontrollers kan gebruiken, servomotoren kan aansturen en meer. Dit zal ik eventueel later in mijn verdere studies kunnen gebruiken.

Het originele plan voor het plaatsen van mijn leds heb ik ook moeten aanpassen. Eerst wou ik alle leds plaatsen in een aluminium balk die boven de parkeerplaatsen gemonteerd werden. Dit heb ik aangepast naar een pcb waarop alle leds gemonteerd zijn. Bij het maken van de pcb hiervoor heb ik wel een fout gemaakt. De afmetingen klopten niet volledig dus heb ik de leds allemaal anders moeten positioneren. Dit zorgt ervoor dat het eindresultaat er anders uitziet dan eerst gepland.

Een fout die moeilijk op te lossen was zat in het programma voor het aansturen van mijn I2C modules die de leds aansturen. Hier had ik mijn eigen functie voor geschreven. De waarde die naar deze functie werd geschreven werd fout aangepast bij het ontvangen van de data van de tweede microcontroller. Dit heb ik kunnen oplossen.

Het eindresultaat is niet 100% wat ik bij de start voor ogen had. Door aanpassingen in de loop van het project zijn de LEDS en de software anders afgewerkt dan initieel de bedoeling was. Toch ben ik tevreden met de aangelegde weg.

11. Bronnen

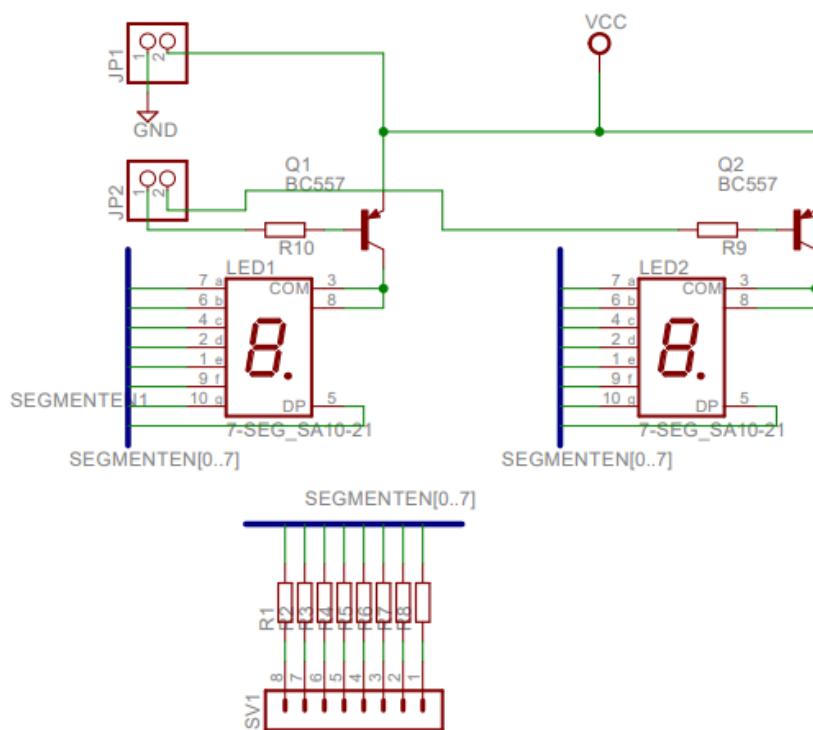
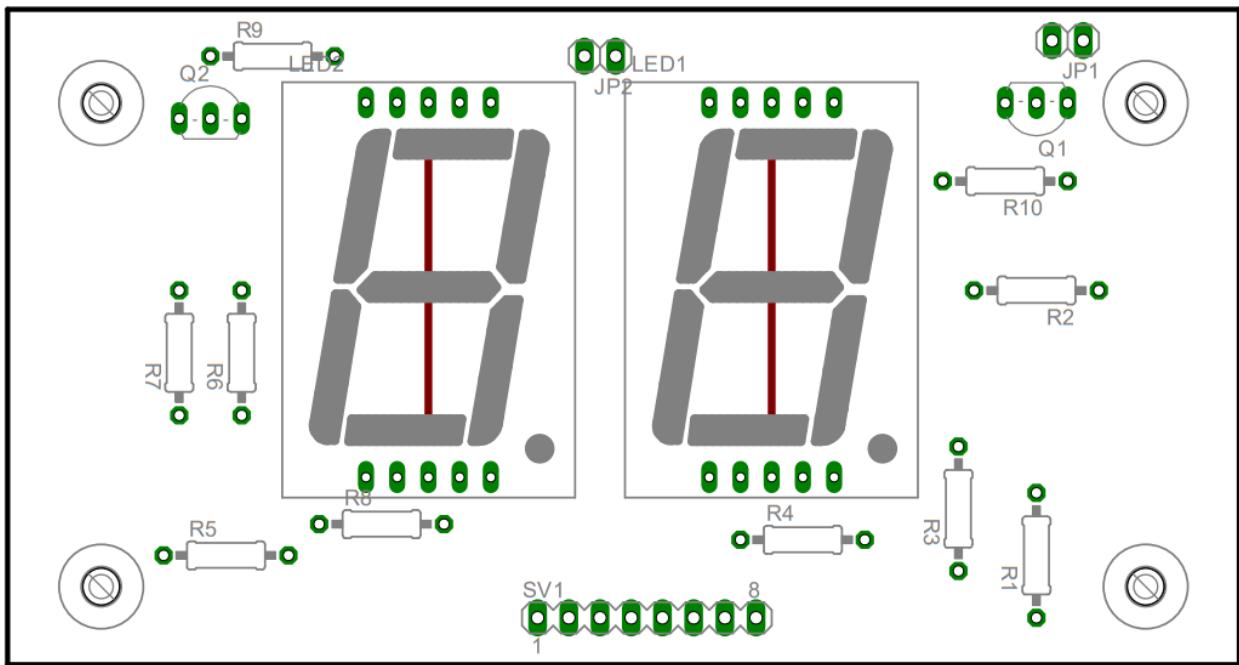
- Specificaties: <https://gipthiandhaene.wordpress.com/specificaties/>
- Cursussen en OneNote Mr. Van Quickelberghe (MCU)
- Cursussen en OneNote Mvr. Vervaeke (C#)

12. Bijlagen

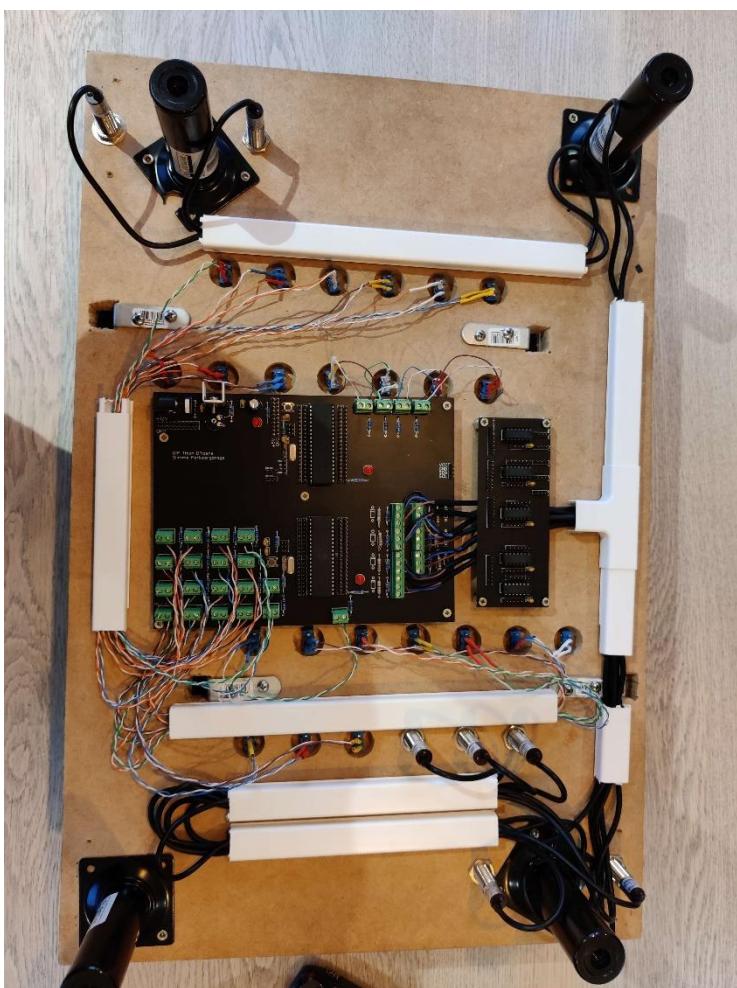
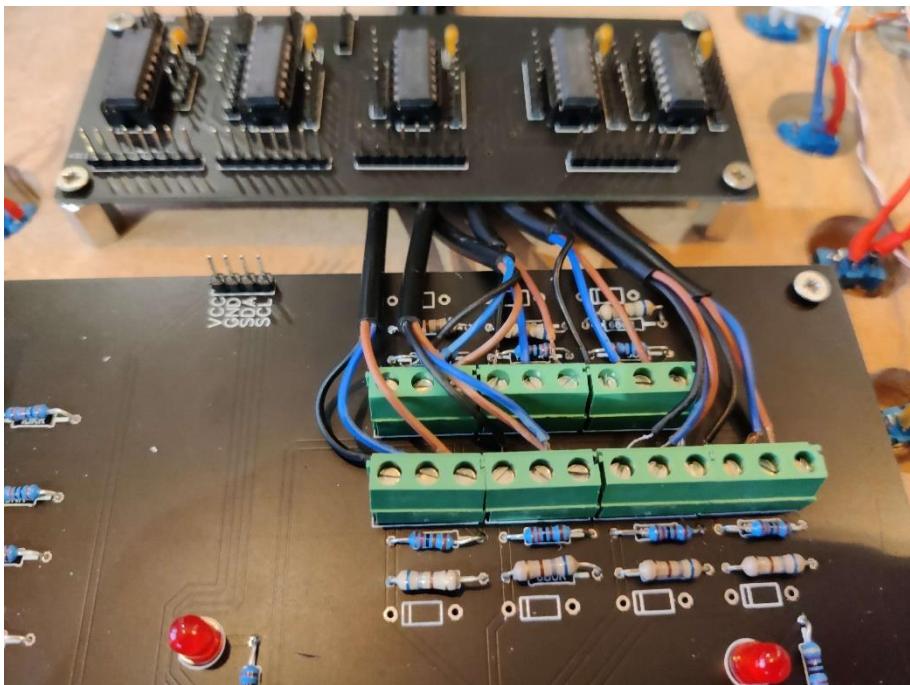
Zie specificaties op blog: <https://gipthiandhaene.wordpress.com/specificaties/>

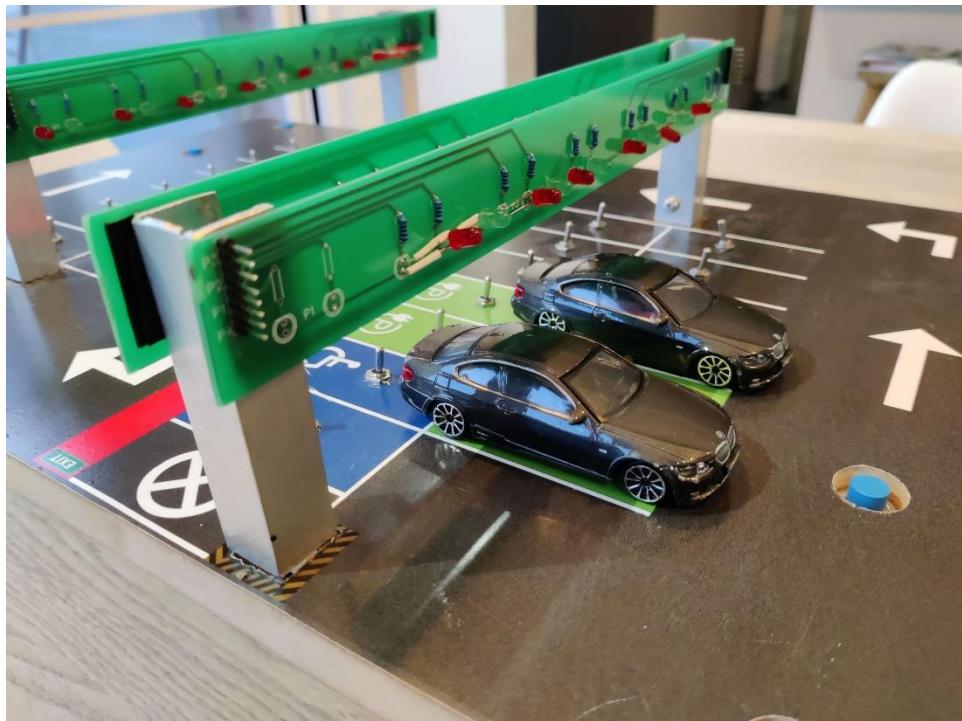
Alle code van mijn GIP programma kan op mijn GitHub gevonden worden:
<https://github.com/ThianDhaene/GIP-ThianDhaene/>

Schema's print 7-segment:

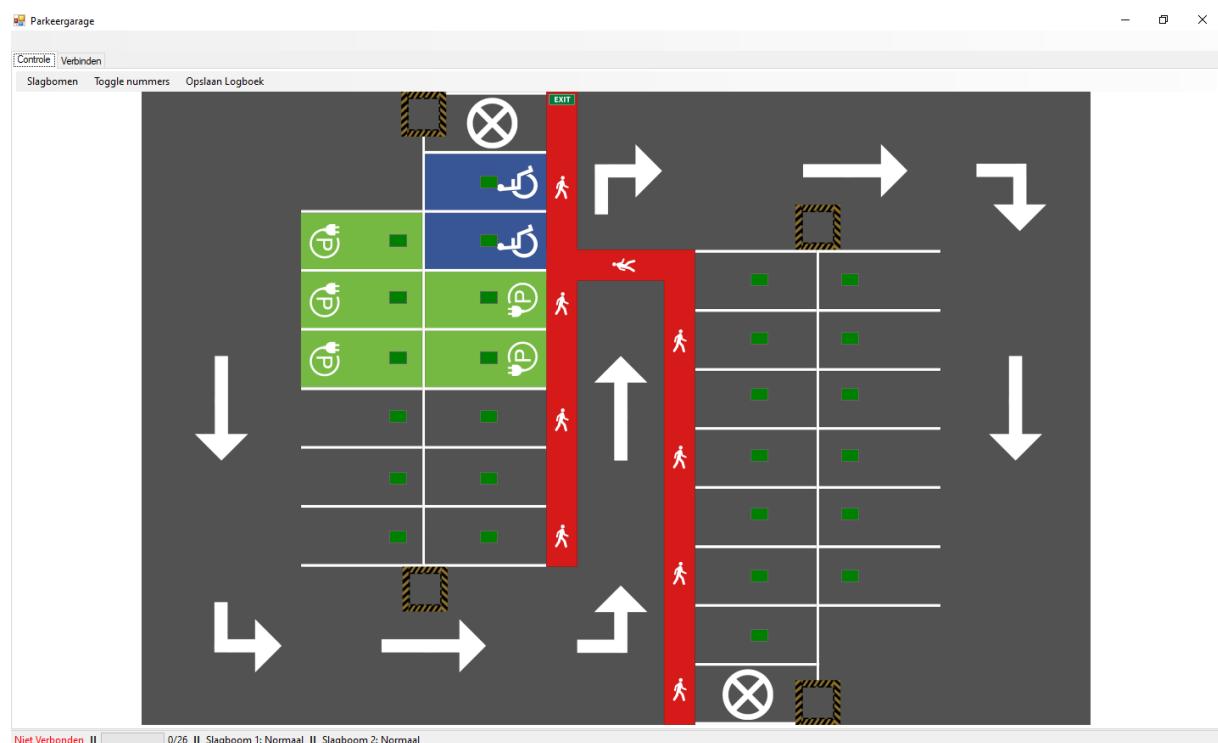


Foto's afgewerkte garage:

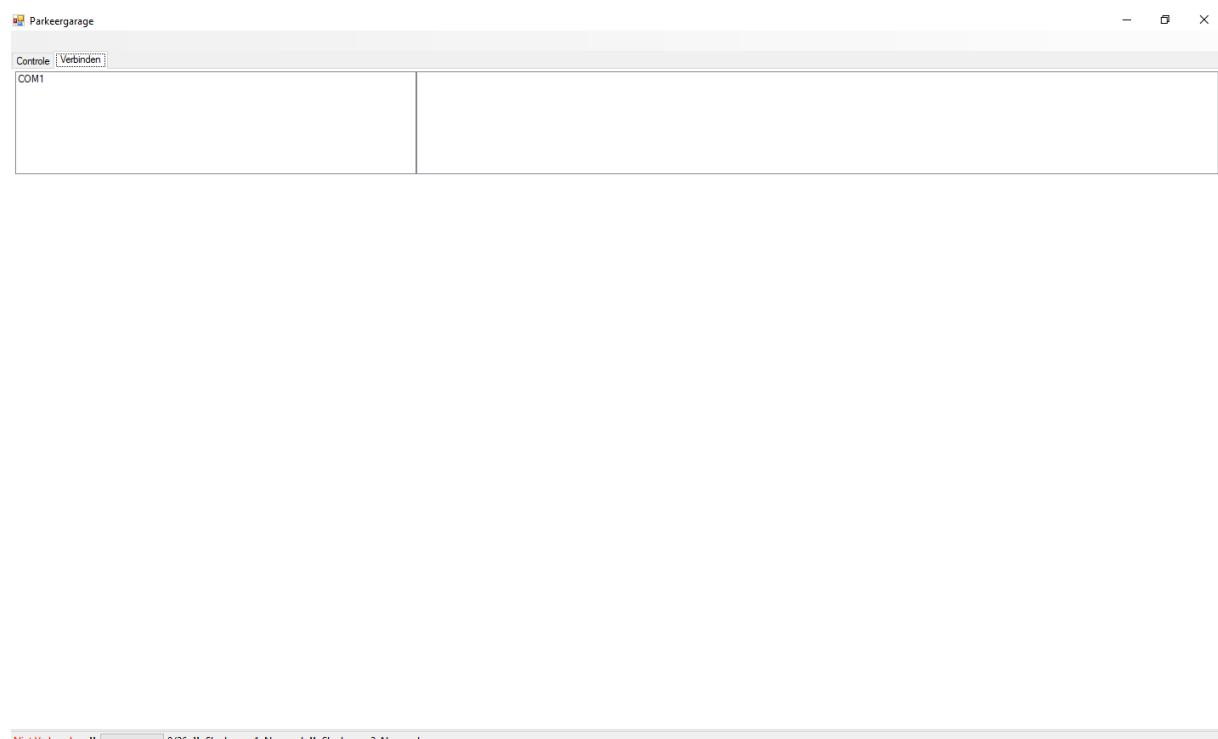




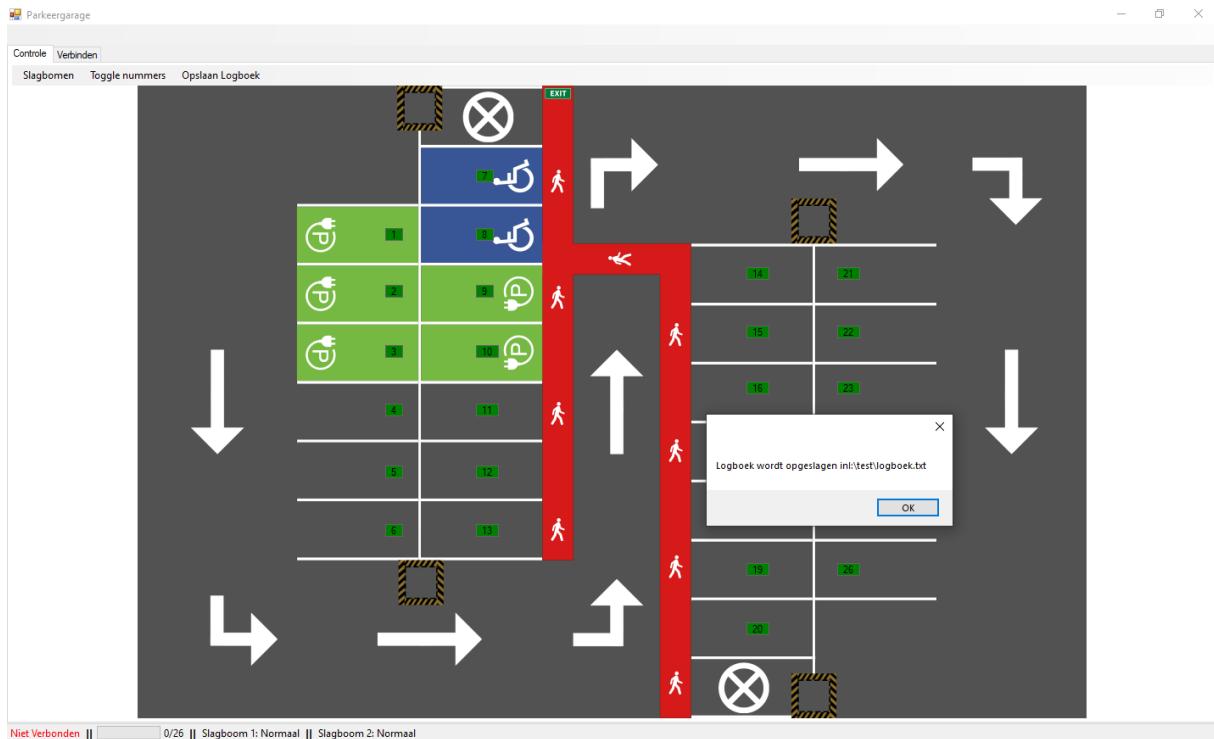
Foto's GUI:



Standaard scherm



Seriële verbinding opstarten



Weergave nummers ingeschakeld, gekozen locatie voor logboek