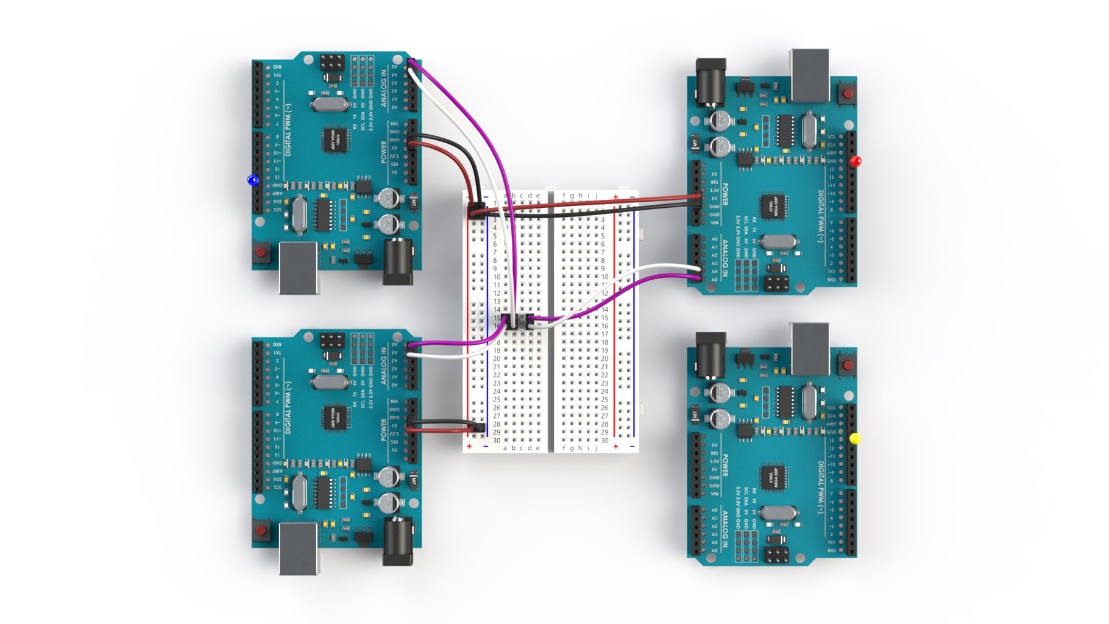
Sharing resources I2C

Tiemon Steeghs

Inhoud

[1. Probleemstelling 3](#_Toc137414802)

[2. Design 4](#_Toc137414803)

[3. Implementatie 5](#_Toc137414804)

[3.1. Versie 1: Status opvragen 5](#_Toc137414805)

[3.2. Versie 2: Lowcode data lezen 5](#_Toc137414806)

[3.3. Versie 3: Communicatie met arbitrage 6](#_Toc137414807)

[4. Evaluatie 9](#_Toc137414808)

[5. Bronvermelding 10](#_Toc137414809)

# Probleemstelling

Binnen I2C systemen gaat er veel data heen en weer. Maar wat nou als twee masters tegelijkertijd data willen versturen naar een slave? In deze challenge ga ik kijken hoe je ervoor zorgt dat dit geen problemen oplevert, zodat de communicatie vlekkeloos blijft verlopen. Een verdere beschrijving van mijn opstelling is in het volgende kopje beschreven.

# Design

Mijn systeem bestaat uit twee Arduino redboards (masters) en één Qwiic oled display (slave). Mijn systeem gaat als volgt te werk: De twee redboard masters en de Qwiic oled display zitten allemaal aan elkaar aan een I2C bus. Beiden masters kunnen data schrijven naar de display, om zo het gene wat afgebeeld wordt te veranderen. Voordat ze dit doen vragen ze eerst aan de andere master of deze niet al bezig is met het schrijven van data. Op het moment dat de master terugkrijgt van de andere master dat de bus beschikbaar is, zal hij data gaan schrijven. Het is belangrijk om op te merken dat de slave op zichzelf niet geprogrameerd wordt, dus alles wordt geregeld door de twee masters.

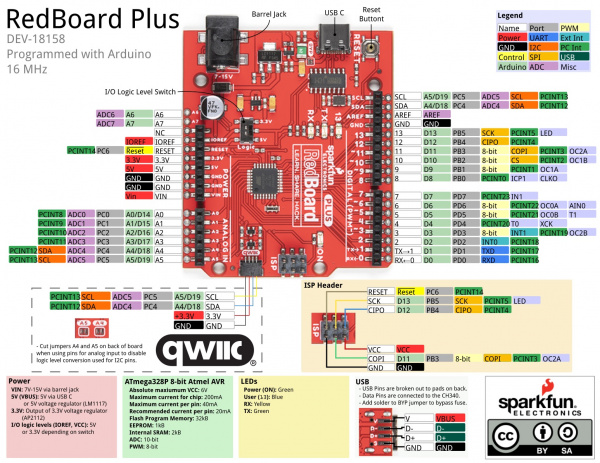
De beschreven opstelling is hieronder ook weergegeven. Bij de pijlen zie je dus ook staan wat de masters opvragen en verkrijgen. De data van de beschikbaarheid gaat overigens via de display omdat die tussen de masters in zit, echter doet de display hier niks mee.

Afbeelding met tekst, lijn, schermopname, diagram

Automatisch gegenereerde beschrijving

Figuur Diagram van het design

Om het systeem werkende te krijgen moeten de twee redboards steeds naar de SDA lijn kijken of deze hoog of laag is. Dit moet overheen komen met de bits die ze willen sturen. Als dit niet overeenkomt weten ze dus dan de ander bezig met schrijven. Om de SDA lijn te kunnen lezen moet ik eerst ontdekken of dit wel kan.



Figuur Pin layout van de Redboard

# Implementatie

Voor de daadwerkelijke implementatie ben ik door verschillende versies gegaan. Dit komt doordat ik erachter kwam dat mijn originele design enkele problemen met zich mee bracht, dit wordt hieronder verder beschreven.

## Versie 1: Status opvragen

Mijn eerste oplossing was doormiddel van de masters eerst een bericht naar de ander te laten sturen voor het opvragen van of de bus beschikbaar is.

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

Echter kwam ik erachter dat als de twee masters precies tegelijk de data zouden opvragen er problemen zullen optreden. Deze implementatie is daarom nog niet voldoende dus ben ik naar andere oplossingen gaan kijken.

## Versie 2: Lowcode data lezen

Hierna ben ik gaan kijken naar een lowcode oplossing om de SDA lijn af te lezen. Echter had deze oplossing ook nog een probleem. Deze methode leest alleen data af die naar het adres van de desbetreffende master wordt gestuurd. Dit betekent dus dat als de master data zou gaan sturen naar de slave op exact hetzelfde moment er nog steeds een probleem optreed.

Afbeelding met tekst, schermopname, Lettertype

Automatisch gegenereerde beschrijving

## Versie 3: Communicatie met arbitrage

### POC

Voor mijn derde versie ben ik gaan kijken naar het toevoegen van een arbitrage mechanisme. Door dit toe te voegen zou er geen probleem moeten optreden als beiden master tegelijk data willen schrijven naar de slave. Door direct de SDA-lijn uit te lezen kan ik zien of de andere master bezig is. Op het moment dat er geen data wordt geschreven word de lijn als hoog afgelezen, maar op het moment dat dus de andere master data gaat schrijven naar de slave en de lijn laag trekt, kan de master dit waarnemen. Vervolgens zou de master dan even moeten wachten en dan opnieuw proberen de data te verzenden.



Figuur Het lezen van de SDA-lijn

Afbeelding met tekst, schermopname

Automatisch gegenereerde beschrijving

Figuur Het resultaat van de lezing

Afbeelding met tekst, schermopname, software, Multimediasoftware

Automatisch gegenereerde beschrijvingVoor mijn kleine POC laat ik de masters een byte aan data sturen en niet een volledige afbeelding omdat dit erg veel bits zijn om te controleren. Voor deze POC gebruik ik daarom de data() methode voor het versturen van data naar de slave. Ik kan hiervoor niet de wire methodes gebruiken omdat deze niet samen gaat met de digitalRead() methode. “You can't use digitalRead() with I2C.” (*Arduino Forums*, 2018b)



Als je nu kijkt wat de andere master op de SDA leest kan hij vrij accuraat zien hoe de data wordt gestuurd. Echter is het nog niet volledig accuraat omdat de timing niet helemaal overeenkomt.

Om de timing te controlleren ben ik gaan kijken naar het I2C signaal met de logic analyzer. Ik stuur nu elke halve seconde data, wat je hieronder ook ziet.

Afbeelding met schermopname, Multimediasoftware, Grafische software

Automatisch gegenereerde beschrijving

Figuur Uitgezoomed beeld van de I2C data

Als we verder inzoomen zie je de negen bits die ik aan het sturen ben. Één start bit en acht data bits.

Afbeelding met schermopname, lijn, Rechthoek, Parallel

Automatisch gegenereerde beschrijving

Figuur Closeup van de gehele verzending waarin je de verschillende bits ziet

Als we nu kijken naar de tijd tussen elke bit zien we dat dit 30 microseconden is.

Afbeelding met schermopname, Multimediasoftware, software

Automatisch gegenereerde beschrijving

Figuur Closeup van de verzending van één losse bit

Een volledige bit verzending duurt ongeveer 317 microseconden. Door nu steeds deze tijd te wachten nadat ik elke bit ontvang kan nog accurater de SDA-lijn uitlezen.

In mijn code doe ik dit door eerst te kijken naar of de SDA-lijn laag is. Dit is eigenlijk de start bit waar ik op wacht. Nadat de startbit is ontvangen lees ik met een for loop de acht data bits uit met dus een vertraging van 317 microseconden/

Afbeelding met tekst, schermopname, Lettertype, ontwerp

Automatisch gegenereerde beschrijving

Figuur Stukje code waarin de uitlezing te zien is

### POC probleem

Ik kan nu dus redelijk accuraat uitlezen via de SDA-lijn wat de andere master aan het versturen is. Echter is er wel een probleem. Met het uitlezen van de SDA-lijn via de digitalRead methode of de pin registers kan je niet zien wat de master zelf stuurt. Voor de arbitrage betekent dit dat hij maar voor de helft zal werken. De logica achter mijn oplossing is correct dus daar ligt niet het probleem. “Arbitration proceeds bit by bit. During every bit, while SCL is HIGH, each controller checks to see if the SDA level matches what it has sent.” (nxp user manual, 2018)

Al zet master1 bijvoorbeeld de lijn hoog en master2 ook, dan zal dit correct worden waargenomen. Maar als master1 de lijn laag legt maar master2 doet niks, zal de lijn nog steeds als hoog worden uitgelezen. Ik dacht eerst dat de digitalRead() methode het probleem was, maar als ik handmatig het register van PC4 (Pin A4 oftewel SDA pin) probeer uit te lezen, krijg ik precies hetzelfde probleem. Het is daarom niet mogelijk om op deze manier de arbitrage toe te passen.



Figuur Het handmatig uitlezen van het SDA pin register, dit lost het probleem niet op

Als ik opzoek ga naar bronnen die dit probleem omschrijven vind ik eigenlijk maar weinig. ChatGPT zegt dit over het probleem: “You are correct that using digitalRead on the SDA line immediately after the master has sent data to the slave will not directly provide you with the transmitted data or allow you to observe the activity on the SDA line. The digitalRead function in the Arduino environment is designed to read the logic level of a digital input pin, but it is not suitable for directly monitoring the activity on the I2C bus. The I2C communication involves a specific protocol that includes addressing, start and stop conditions, and data packets. The digitalRead function alone cannot decode this protocol and provide you with the actual data being transmitted on the bus.” Dit komt ook overheen met mijn vondsten dus het lijkt mij zeer aannemelijk dat mijn techniek niet de manier is hoe je arbitrage moet toepassen.

# Evaluatie

Naar meerdere implementaties gemaakt te hebben is het uiteindelijk gelukt om het probleem gedeeltelijk op te lossen. Doormiddel van te kijken naar de SDA-lijn kan ik achterhalen of de andere master data aan het schrijven is en zo weet de master dus of hij moeten wachten of niet.

Echter omdat de master niet van zichzelf kan zien of hij data aan het schrijven is op de manier die ik heb geïmplementeerd werkt het arbitrage mechanisme maar gedeeltelijk. Of ik nou de digitalRead() methode gebruik of het register van de SDA pin handmatig uitlees maakt niet uit, ik kan niet zien van de master zelf of de SDA lijn laag of hoog is.

# Bronvermelding

*Qwiic Micro OLED Hookup Guide - SparkFun Learn*. (z.d.). <https://learn.sparkfun.com/tutorials/qwiic-micro-oled-hookup-guide/all>

*ATmega328P datasheet*

<https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf>

*IS THAT ABLE TO DIGITAL READ A I2C BUS SENSOR*. (2018, 24 december). Arduino Forum. <https://forum.arduino.cc/t/is-that-able-to-digital-read-a-i2c-bus-sensor/563734/8>

*Gammon Forum : Electronics : Microprocessors : I2C - Two-Wire Peripheral Interface - for Arduino*. (z.d.). <http://www.gammon.com.au/i2c>

*MultiMaster – I2C Bus*. (z.d.). <https://www.i2c-bus.org/multimaster/>

*I2C bus user manual* - <https://www.nxp.com/docs/en/user-guide/UM10204.pdf>