# Opdracht: Two Wire Interface en Integratie

# 

Opdracht: Two Wire Interface en Integratie 1

Inleiding 2

Leerdoelen 3

Leerdoelen deel A 3

Leerdoelen deel B 3

Leerdoelen deel C 3

Leerdoelen deel D 3

Deel A.1: Schrijven naar de slave 4

Beschrijving van de opdracht 4

Stappenplan 4

Tips 4

Te veel kletsen tijdens I2C communicatie 4

Deel A.2: Schrijven en lezen naar de slave 5

Beschrijving van de opdracht 5

Stappenplan 5

Tips 5

Deel B: Het simuleren van Hardware Registers 6

Beschrijving van de opdracht 6

Stappenplan 6

Deel C: Accelerometer 7

Beschrijving van de opdracht 7

Stappenplan 7

Tips 8

Tips – MMA7455 8

Deel D: Accelerometer waarden integreren 9

Beschrijving van de opdracht 9

Stapppenplan 10

Tips 10

Submission 11

## Inleiding

In deze opdracht zullen we micro-controllers met verbinden met andere devices. Dit doen we door gebruik te maken de Two Wire Interface (I2C).

De opdracht bestaat uit de volgende delen:

1. Twee Arduino’s met elkaar verbinden en bytes versturen/lezen.
2. Een Arduino hardware registers laten simuleren.
3. Een I2C device (accelerometer) configuren en uitlezen.
4. Accelerometer waarden integreren

Bij de beschrijving van elke deel vind je de volgende delen:

* Beschrijving opdracht
* Stappenplan
* Tips

**Lees deze steeds helemaal door voordat je de opdracht gaat uitvoeren.** Het stappenplan geef je hints om de opdracht succesvol te kunnen maken. Je zult echter af en toe met behulp van de tips zelf alle details ontdekken.

## Leerdoelen

### Leerdoelen deel A

* Het kunnen verbinden van twee Arduino’s via I2C (juiste draadjes aansluiten).
* Het begrijpen en configureren van het I2C adres van de slave.
* Het kunnen versturen van data van de master naar de slave.
* Het laten terugsturen van data van de slave naar de master. (Tip: als de master daarom vraagt!)

### Leerdoelen deel B

* De betekenis van hardware registers begrijpen.
* Een microcontoller programma maken met registers waar naar toe geschreven en gelezen kan worden.
* De register waarden aanpassen afhankelijk van de betekenis van elke register.

### Leerdoelen deel C

* Uit de datasheet van een I2C device kunnen achterhalen wat de betekenis van de hardware registers is.
* Het kunnen achterhalen wat adres van het I2C device is.
* Het kunnen schrijven naar registers.
* Het kunnen lezen van registers.

### Leerdoelen deel D

* Het ‘integrator’ algoritme kunnen implementeren op een embedded system.

## Deel A.1: Schrijven naar de slave

### Beschrijving van de opdracht

Je verbindt de Arduino-master en Arduino-slave met elkaar. De master stuurt een byte naar de slave. Zowel de master als slave laten de verstuurde en/of ontvangen bytes via seriële communicatie zien.

### Stappenplan

* Verbind de Arduino’s met elkaar door de data (SDA), clock (SCL) en ground met elkaar te verbinden.
* Het is mogelijk om via TinkerCad twee Arduino devices te simuleren. Een demo hiervan vind je via: <https://www.tinkercad.com/things/eeoiN0GklWb>
* Gebruik de voorbeelden File 🡪 Examples 🡪 Wire 🡪 [Slave Receiver](http://arduino.cc/en/Tutorial/MasterWriter) en Master Writer als basis. Gebruik hierbij de volgende code in de receiveEvent functie (het voorbeeld was onnodig complex door apart naar de laatst ontvangen byte te kijken):

void receiveEvent(int howMany)

{

while (0 < Wire.available()) {

char c = Wire.read();

//TODO: doe iets met de ontvangen bytes…

}

* Configureer de Arduino-slave code zodat deze als slave het adres [42](http://en.wikipedia.org/wiki/42_(number)) heeft.
* Pas de basis code zo aan dat de master elke seconde een byte naar de slave stuurt.
* De slave laat de ontvangen byte via de seriële poort zien.

### Tips

#### Te veel kletsen tijdens I2C communicatie

Vaak gaat het mis als er gedurende het ontvangen en/of verzenden van I2C berichten te veel logging op de Arduino plaatsvindt via seriële communicatie. Waarom denk je dat dit komt? Omdat hij te druk bezig is met printen en niet meer kan lezen.

Het is daarom een alternatief om de seriële logging niet vanuit de ‘receiveEvent’ functie te doen, maar vanuit de ‘loop’ functie.

#include <Wire.h>

#define SLAVE\_ID (42)

int led = 7;

byte value = 0;

void setup()

{

Serial.begin(115200);

pinMode(led, OUTPUT);

Wire.begin(SLAVE\_ID);

Wire.onReceive(receiveEvent);

Wire.onRequest(requestEvent);

}

void receiveEvent(int howMany)

{

/\* Make sure Wire buffer is empty, otherwhise

\* communication will non-function in TinkerCad. \*/

while(Wire.available() > 0)

{

value = Wire.read();

}

}

void requestEvent() {

const byte someValue = 4;

Wire.write(value + someValue);

}

void loop()

{

Serial.print("Value: "); Serial.println(value);

digitalWrite(led, (value % 2) ? HIGH : LOW);

delay(250);

}

## Deel A.2: Schrijven en lezen naar de slave

### Beschrijving van de opdracht

Dit is een uitbreiding van de vorige opdracht. Laat de slave het getal 2 terug sturen als de opgestuurde byte groter dan 100 is. Als dat niet zo is dan wordt het getal 4 teruggestuurd.

### Stappenplan

Breid de code uit zodat de master om een byte vraagt, en de slave ook de correcte byte terugstuurd.

### Tips

* De Arduino slave mag pas data terugsturen als de master hier om vraagt (readByte). Om dit te doen is het nodig om de Wire.onRequest(….) functie te gebruiken. Zie bijvoorbeeld deze [link](https://www.arduino.cc/en/Reference/WireOnRequest) over Wire.onRequest op de Arduino website.

/\*+--------------------------------+\*/

/\*| S l a v e |\*/

/\*+--------------------------------+\*/

#include <Wire.h>

#define SLAVE\_ID (42)

int led = 7;

byte value = 0;

void setup()

{

Serial.begin(115200);

pinMode(led, OUTPUT);

Wire.begin(SLAVE\_ID);

Wire.onReceive(receiveEvent);

Wire.onRequest(requestEvent);

}

void receiveEvent(int howMany)

{

/\* Make sure Wire buffer is empty, otherwhise

\* communication will non-function in TinkerCad. \*/

while(Wire.available() > 0)

{

value = Wire.read();

}

}

void requestEvent() {

const byte someValue = 4;

if (value >= 100)

{

Wire.write(2);

}

else

{

Wire.write(4);

}

}

void loop()

{

Serial.print("Value: "); Serial.println(value);

digitalWrite(led, (value % 2) ? HIGH : LOW);

delay(250);

}

Master

#include <Wire.h>

#define SLAVE\_ID (42)

void setup()

{

Wire.begin();

Serial.begin(115200);

delay(3000);

}

void loop()

{

static byte value = 0;

Wire.beginTransmission(SLAVE\_ID);

Wire.write(value);

int result = Wire.endTransmission(true /\*repeatedStop\*/);

Serial.print("Result: "); Serial.println(result);

Serial.print("Value:");Serial.println(value);

const int numberOfBytesForRequest = 1;

Wire.requestFrom(SLAVE\_ID, numberOfBytesForRequest);

while(Wire.available() == 0) {

//Waiting for reply

//Todo: handle non-replying slave... timeout?

}

int receivedValue = Wire.read();

Serial.print("Received value:");Serial.println(receivedValue);

value++;

value %= 105;

delay(1000);

}

* Voor de detailed Wire library reference zie: <https://playground.arduino.cc/Main/WireLibraryDetailedReference>
* Studenten die moeite hebben met het voor elkaar krijgen van een "repeated start" op de I2C bus voor het lezen van register data kunnen hints vinden op: <https://www.arduino.cc/en/Reference/WireEndTransmission>
* Gebruik een logic analyzer zoals de Analog Discovery 2 om de i2C bus te “sniffen”. Dit helpt je bij het debuggen en het begrijpen van het gedrag van zowel het I2C protocol je code.

## Deel B: Het simuleren van Hardware Registers

Een [hardware register](https://en.wikipedia.org/wiki/Hardware_register) is een geheugen adres met een specifieke betekenis.

Voorbeelden van register zijn bijvoorbeeld het kunnen configureren van een device en het uitlezen van meetwaarden. In onderstaande voorbeeld zijn er 4 registers:

|  |  |  |  |
| --- | --- | --- | --- |
| **Register-adres** | **Betekenis** | **Byte size** | **I/O** |
| 21 | variabele a | 1 | Schrijven + Lezen |
| 22 | variabele b | 1 | Schrijven + Lezen |
| 23 | min(a,b) | 1 | Lezen |
| 24 | max(a,b) | 1 | Lezen |

* Op register adressen (21, 22) is het mogelijk om de een waarde van de variabelen a en b te schrijven én lezen.
* Op register adressen (23, 24) is de minimale en maximale waarde van a en b te lezen.
* Lezen betekent dat dit register alleen te lezen is vanuit de I2C master.

Bijvoorbeeld: a = 12, b = 4 🡪 min(a,b) =4, max(a,b) = 12.

### Beschrijving van de opdracht

Implementeer bovenstaande min/max machine op de Arduino.

### Stappenplan

* Om naar een register te schrijven is het de bedoeling dat je 2 bytes naar de slave stuurt. De slave interpreteert het 1ste byte als het register adres, en het 2de byte als de register waarde.
* Om een register uit te lezen is het noodzakelijk om aan de slave te vertellen welke register adres je wilt lezen (**tip: write vanuit master**). De slave zal vervolgens de register waarde voor dit adres opsturen (**tip: request door de master om communicatie te initiëren**).

## /\*+--------------------------------+\*/

## /\*| M a s t e r |\*/

## /\*+--------------------------------+\*/

## #include <Wire.h>

## #define SLAVE\_ID (42)

## void setup()

## {

## Wire.begin();

## Serial.begin(115200);

## delay(3000);

## }

## void WriteRegister(int reg, byte value)

## {

## Wire.beginTransmission(SLAVE\_ID);

## Wire.write(reg);

## Wire.write(value);

## Wire.endTransmission();

## }

## byte ReadRegister(int reg)

## {

## Wire.beginTransmission(SLAVE\_ID);

## Wire.write(reg);

## Wire.endTransmisson();

## Wire.requestFrom(SLAVE\_ID, 1);

## while( Wire.available() == 0) {};

## return Wire.read();

## }

## void loop()

## {

## int reg;

## 

## byte value;

## Serial.println("Fill in register");

## reg = Serial.read();

## Serial.println("Fill in value");

## value = Serial.read();

## WriteRegister(reg, value);

## value = ReadRegister(reg);

## Wire.beginTransmission(SLAVE\_ID); }

/\*+--------------------------------+\*/

/\*| S l a v e |\*/

/\*+--------------------------------+\*/

#include <Wire.h>

#define SLAVE\_ID (42)

int reg;

byte a = 0;

byte b = 0;

void setup()

{

Wire.begin(SLAVE\_ID);

Wire.onReceive(receiveEvent);

Wire.onRequest(requestEvent);

}

void receiveEvent(int howMany)

{

/\* Make sure Wire buffer is empty, otherwhise

\* communication will non-function in TinkerCad. \*/

if (howMany == 1)

{

reg = Wire.read();

}

else if(howMany == 2)

{

reg = Wire.read();

if (reg == 21)

{

a = Wire.read();

}

else if (reg == 22)

{

b = Wire.read();

}

}

}

void requestEvent() {

if (reg == 21)

{

Wire.write(a);

}

else if (reg == 22)

{

Wire.write(b);

}

else if (reg == 23)

{

Wire.write(min(a, b));

}

else if (reg == 24)

{

Wire.write(max(a, b));

}

}

void loop()

{

}

## Deel C: Accelerometer

|  |  |
| --- | --- |
| MiniIMU-9\_v3\_imu01c\_2014: LSM303d | MiniIMU-9\_v5\_imu04a\_2015: LSM6 |

### Beschrijving van de opdracht

De accelerometer (MMA7455L, LSM303d of LSM6) kan via I2C als slave worden verbonden. Configureer de accelerometer en lees de acceleratie meetwaarden uit in 3-dimensies (x,y,z).

### Stappenplan\

* Gebruik het project ‘accelerometer\_reader’ als basis.
* Sluit de acceleratie sensor aan door gebruik te maken van de data-sheet van de gebruikte accelerometer.
* Het is mogelijk om een tweede Arduino als accelerometer simulator te gebruiken mocht je geen beschikking hebben over de echte hardware. Details zijn te vinden in het project ‘accelerometer\_reader\_mma7455L\_slave\_simulation’.
* Implementeer onderstaande functies (<device> is het door jouw gebruikte device):

void accelerometer\_<device>\_begin();

void accelerometer\_<device>\_set\_measurement\_mode();

Om deze functies te implementeren maak je gebruik van configuratie register(s). Stel het meetbereik zo in dat maximaal een versnelling van 2g kan worden gemeten (1g is de versnelling van een voorwerp dat je laat vallen op aarde = 9,81 m/s2).

* Voor de configuratie-registers zie de datasheet:
  + MMA7455L: MCTL register.
  + LSM303d en LSM6: zie CTRL registers

* Lees de acceleraties waarden in X,YZ richting uit.

void accelerometer\_<device>\_measure(

accelorometer\_data \*data);

Je maakt hierbij gebruik van de registers x,y,z meetwaarden.

* Laat de acceleratie meetwaarden (x,y,z) via seriële communicatie zien.

### Tips

* Acceleratie is de toename van de snelheid (m/s) per seconde (1/s).
* Een acceleratie van 10 m/s2 betekent dat de snelheid elke seconde met 10 m/s toeneemt.
* In 2,8 s wordt bijvoorbeeld een snelheid van 28m/s bereikt (= 100 km/h).
* De I2C master initiëert het ophalen van de x,y,z waarden. Is het mogelijk voor de master om het resultaat meteen terug te lezen? Hoe weet de master of de bytes al opgestuurd zijn door de slave, en dus [beschikbaar](https://translate.google.com/#nl/en/beschikbaar) zijn?
* Bekijk in wat de nauwkeurigheid is van de x,y,z waarden en hoeveel bits je hiervoor nodig hebt.
* Schrijf eerst het register adres wat je wilt gaan lezen (write) en dan pas lees je de waarde uit (request). Dit is precies gelijk aan het uitlezen van een register bij deel B (de min/max machine).
* Zie de #defines in accelerometer.h voor de verschillende register-adressen.
* De registers zijn 1 byte groot. Elke bit in deze byte heeft een unieke betekenis die je kan terugvinden in de datasheet. Je kunt de waarde voor elke bit handig zetten door gebruik te maken van binaire notatie: het (decimale) getal 10 in binaire notatie is 0b1010. Zie ook bijvoorbeeld ook <https://gcc.gnu.org/onlinedocs/gcc/Binary-constants.html>.
* Voor de detailed Wire library reference zie: <https://playground.arduino.cc/Main/WireLibraryDetailedReference>
* Studenten die moeite hebben met het voor elkaar krijgen van een "repeated start" op de I2C bus voor het lezen van register data kunnen hints vinden op: <https://www.arduino.cc/en/Reference/WireEndTransmission>
* Gebruik een logic analyzer zoals de Analog Discovery 2 om de i2C bus te “sniffen”. Dit helpt je bij het debuggen en het begrijpen van het gedrag van zowel het I2C protocol als de accelerometers.

### Tips – MMA7455

* In de NMA7455 datasheet vind je op pagina 21 een overzicht van de hardware registers van de accelerometer.
* De betekenis van de bits in het MCTL register vind je in de datasheet (deze vind je bij het lesmateriaal, zie pagina 9).
* De 8-bits meetwaarden de accelerometer vind je op registers  
   XOUT8, YOUT8, ZOUT8.

void accelerometer\_lsm303\_begin()

{

Wire.begin();

delay(500);

}

void accelerometer\_lsm303\_set\_measurement\_mode()

{

Wire.beginTransmission(SENSOR\_SLAVE\_ADDRESS);

Wire.write(0x20);

Wire.write(0b00010111);

Wire.endTransmission(false);

}

int calculatingX()

{

Wire.beginTransmission(SENSOR\_SLAVE\_ADDRESS);

Wire.write(OUT\_X\_L\_A);

Wire.endTransmission(false);

Wire.requestFrom(SENSOR\_SLAVE\_ADDRESS, 2);

int value = Wire.read();

return value;

}

int calculatingY()

{

Wire.beginTransmission(SENSOR\_SLAVE\_ADDRESS);

Wire.write(OUT\_Y\_L\_A);

Wire.endTransmission(false);

Wire.requestFrom(SENSOR\_SLAVE\_ADDRESS, 2);

int value = Wire.read();

return value;

}

int calculatingZ()

{

Wire.beginTransmission(SENSOR\_SLAVE\_ADDRESS);

Wire.write(OUT\_Z\_L\_A);

Wire.endTransmission(false);

Wire.requestFrom(SENSOR\_SLAVE\_ADDRESS, 2);

int value = Wire.read();

return value;

}

void accelerometer\_lsm303\_measure(accelerometer\_data\* data)

{

data->x = calculatingX();

data->y = calculatingY();

data->z = calculatingZ();

}

## Deel D: Accelerometer waarden integreren

### Beschrijving van de opdracht

Voor deze opdracht gebruiken de library ‘accelerometer\_simulator.h’ die gesimuleerde acceleratie meetwaarden teruggeeft.

De ‘accelerometer\_simulator’ geeft de gemeten acceleratie waarden van een sportauto die van 0 tot tot ruim 300 [km/h] accelereert.



Bron: Arial Atom V8: [https://www.autocar.co.uk](https://www.autocar.co.uk/sites/autocar.co.uk/files/styles/gallery_slide/public/ariel-atom-front.jpg?itok=jNneRZs_)   
  
De meting wordt gestart via de functie ‘accelerometer\_simulator\_begin()’. Op dit moment is de snelheid van de sportauto 0 [km/h] en begint de auto te accelereren.

### Stapppenplan

* Gebruik het project ‘accelerometer\_reader’ als basis.
* Vul als oefening onderstaande tabel in door op elke rij in te vullen wat de snelheid is bij de gevegeven acceleratie-waarden.  
    
  Bij de start van de meting is de snelheid 0 [m/s]. Je mag er gemakshalve vanuit gaan dat de acceleratie een constante waarde heeft in onderstaande intervallen.

|  |  |  |
| --- | --- | --- |
| **Tijdstip-meting [ms]** | **Acceleratie [m/s2]** | **Snelheid [m/s]** |
| 0 | 10 | 0 |
| 500 | 10 | 5 |
| 1000 | 9 | 9.5 |
| 1500 | 9 | 14 |
| 2000 | 8 | 18 |
| 2500 | 8 | 22 |
| 3000 | x | 22 + x |

* Configureer de meegeleverde voorbeeld code zodat dat je accelerometer\_simulator gebruikt. Dit doe je door ‘useSimulationMode’ op ‘true’ te zetten.
* Voer het voorbeeld programma uit en kijk op de serial monitor wat de acceleratie waarden zijn. Na een bepaalde tijd worden de simulatie accelearatie-waarden 0.
  + Hoe lang duurt het voordat de acceleratie waarden 0 worden?
  + In welke richting(en) (x,y,z) wordt er een acceleratie gemeten?

X en 8526 miliseconden

* Schrijf een Arduino programma waarmee je de volgende waarden kunt laten zien via seriele communicatie:
  + De tijdsduur van de volgende intervallen: 0-100, 0-200, 0-300 [km/h].  
    Let op: dus niet van bv 0-106.5 [km/h].
  + De tijdsduur van de volgende intervallen: 80-120, 100-200 [km/h].
  + Afgelegde afstand van de auto sinds de start bij de volgende snelheden: 0, 100, 200 km/h.
* **Extra opdracht:** laat in een grafiek de acceleratie [m/s2], snelheid [km/h] en afgelegde afstand [km] zijn als functie van de tijd in [ms]. De tijd loopt totdat de sportauto 300 [km/h] gaat.

### Tips

* De library heeft de acceleratie waarden in [m/s2] vermenigvuldigd met een factor 1000. Een waarde van 10000 betekent een acceleratie van 10 [m/s2].
* De snelheid wordt gevraagd in [km/h]. Een snelheid van 1 [m/s] betekent dat je 3600 [m] in 1 [uur] aflegt.

# Submission

Als submission lever je alle code in die je gemaakt hebt voor onderdelen B,C en D. Geef ook een demo aan je docent van deze onderdelen.   
  
Eventueel de extra opdracht voor onderdeel D graag in leveren als pdf-file.