**Startsemester  
Verdiepingsverslag  
Technology**

*<invoegen relevante afbeelding>*

**Studentnaam:  
Studentnummer:   
Klas:   
Vakdocent:  
  
Versie:   
Datum:**

**Versiebeheer**

|  |  |  |  |
| --- | --- | --- | --- |
| **Versienummer** | **Datum** | **Auteur** | **Veranderingen** |
| *V1.0* | *18-12-2022* | *Marijn Verschuren* | *TinkerCad design, Breadboard design, Arduino tester, blinking led, debounce, trafic control, IO techniques (digital, analog, UART/ USART)* |
| *V1.1* | *20-12-2022* | *Marijn Verschuren* | *CAN, I2C, SPI* |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Inhoudsopgave

[Inhoudsopgave 3](#_Toc122298951)

[1 Inleiding 4](#_Toc122298952)

[1.1 Aanleiding 4](#_Toc122298953)

[1.2 Onderwerp 4](#_Toc122298954)

[1.3 Leeswijzer 4](#_Toc122298955)

[2 Introductie 5](#_Toc122298956)

[3 Aantonen leerdoelen 6](#_Toc122298957)

[3.1 Sprint 1: Proof of concepts 6](#_Toc122298958)

[3.1.1 TinkerCAD Design: 6](#_Toc122298959)

[3.1.2 Breadboard Design: 7](#_Toc122298960)

[3.1.3 Arduino tester: 7](#_Toc122298961)

[3.1.4 Non-blocking blinking led: 9](#_Toc122298962)

[3.1.5 Non-blocking debounce + button function: 10](#_Toc122298963)

[3.1.6 Trafic control: 11](#_Toc122298964)

[3.1.7 I/O Techniques: 13](#_Toc122298965)

[3.1.7.1 Digital: 13](#_Toc122298966)

[3.1.7.2 Analoog: 13](#_Toc122298967)

[3.1.7.3 UART / USART: 14](#_Toc122298968)

[3.1.7.4 CAN: 15](#_Toc122298969)

[3.1.7.5 I2C: 15](#_Toc122298970)

[3.1.7.6 SPI: 15](#_Toc122298971)

[3.2 Sprint 2: Tussenproduct 15](#_Toc122298972)

[3.3 Sprint3: Eindproduct 15](#_Toc122298973)

[4 Reflectie / evaluatie 16](#_Toc122298974)

[4.1 Waar ben ik trots op? 16](#_Toc122298975)

[4.2 Wat doe ik een volgende keer anders? 16](#_Toc122298976)

[4.3 Welke formatieve indicatie zou ik mezelf geven voor de verdieping Technology? 16](#_Toc122298977)

# Inleiding

<Begin een verslag altijd met een inleiding. Voor jou als schrijver zijn onderstaande zaken misschien overduidelijk, maar voor de lezer wellicht niet. **Schrijf het verslag niet voor een docent maar zorg dat ook niet-ingewijden het kunnen volgen**>

## Aanleiding

*<Beschrijf hier waarom je dit verslag schrijft>*

## Onderwerp

*<Beschrijf hier waar dit verslag over gaat>*

## Leeswijzer

*<Beschrijf hier de opbouw van dit verslag>*

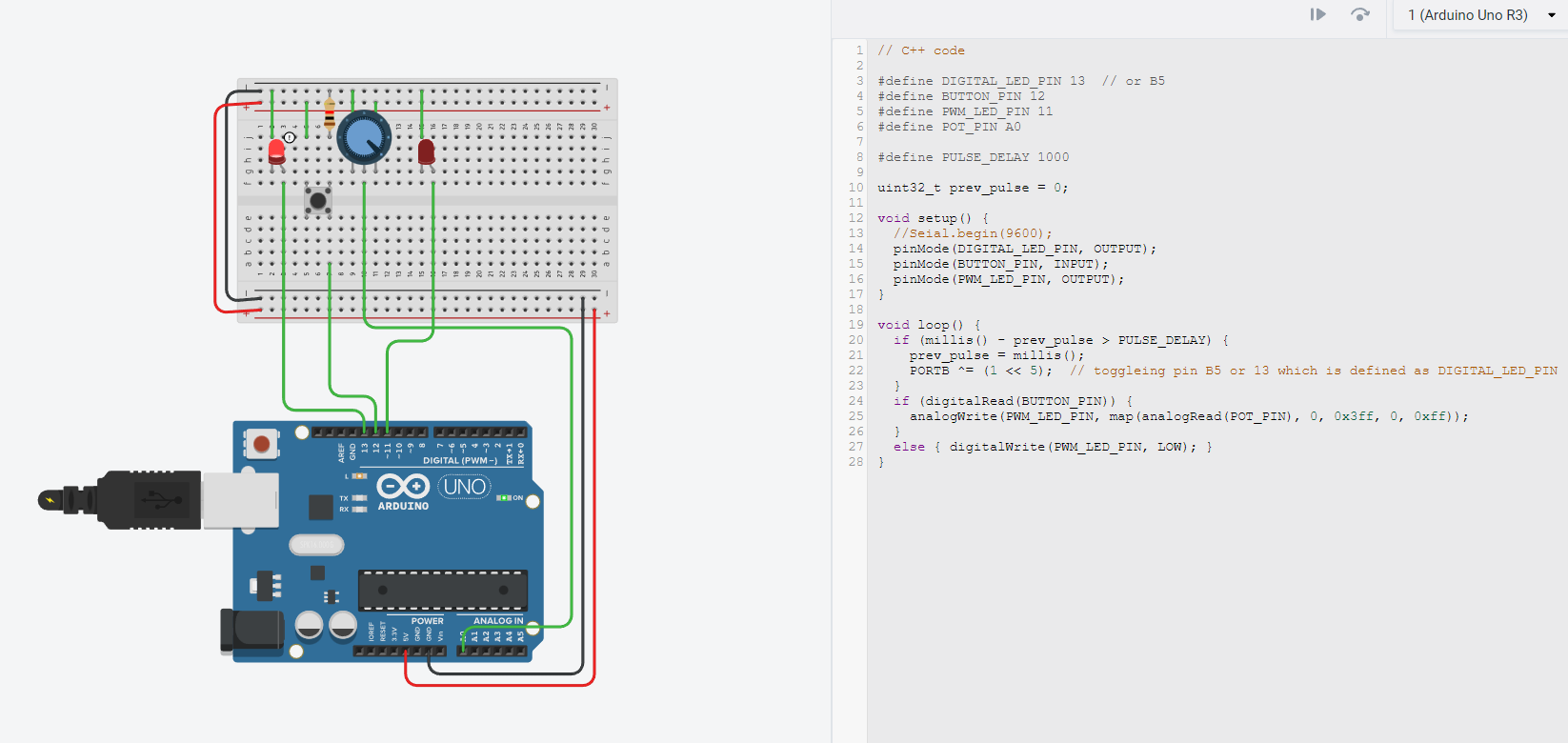
# Introductie

*<* *Beschrijf hier kort jullie project als groep en meer uitgebreid wat je individuele bijdrage daarin zal zijn>*

# Aantonen leerdoelen

## Sprint 1: Proof of concepts

### TinkerCAD Design:



De volgende twee regels in de code die enigszins bijzonder zijn:

|  |
| --- |
| PORTB ^= (1 << 5); // toggleing pin B5 or 13 which is defined as DIGITAL\_LED\_PIN |

In deze regel wordt een pin telkens hoog en laag geschakeld dit heb ik via een GPIO register gedaan omdat Arduino geen toggle functie heeft. In dit geval heb ik de port en het bit index van pin 13 opgezocht in het pinout diagram (PORTB, PIN5). Daarna word een xor uitgevoerd op bit 5 van PORTB hierdoor word pin13 geschakeld.

|  |
| --- |
| analogWrite(PWM\_LED\_PIN, map(analogRead(POT\_PIN), 0, 0x3ff, 0, 0xff)); |

In deze regel word de analoge pin die van de potmeter afgelezen dit geeft een waarde van 0 tot 1023 dit word omgerekend naar een waarde van 0 tot 255. Deze waarde wordt daarna uitgeschreven naar de PWM pin van de led.

### Breadboard Design:

A picture containing text

Description automatically generated

Het nagebouwde circuit had maar een verandering en dat was het bit nummer in PORTB omdat ik een Ardiuno Mega2560 gebruik.

### Arduino tester:

Voor deze pin tester heb ik geen externe onderdelen nodig gehad maar als ik een verzie zou maken met een led, button en potentiometer zou ik eerst de led aanzetten en vervolgens het programma laten wachten op een button press waarna een bericht gestuurd zal worden naar de serial console waar in staat wat de waarde van de potmeter was en welke pins nu getest kunnen worden. Ik heb in dit geval voor een simpele software oplossing gekozen waarin de pins alleen getest worden op digitale output (PWM inbegrepen) maar dit is alsnog boeiend voor analoge inputs omdat de pin simpelweg doorgebrand kan zijn hier word dus wel voor getest. Deze tester is dus niet geschikt voor het testen van de ADC peripheral.

Text

Description automatically generated

Deze code zet alle pins die niet in de exclude array staan op high en checkt of deze ook daadwerkelijk op high zijn gegaan door eerst de pin als output te configureren. Dit word gedaan door bits in de DDR registers high te zetten deze DDR registers bevinden zich op IO\_BASE + 1 + 3n. Vervolgens worden deze pins op high gezet door bits in de PORT register high te zetten deze PORT registers bevinden zich op IO\_BASE + 2 + 3n. Dan worden deze pins als input geconfigureerd. Ten slotte word de status afgelezen van de PIN register in IO\_BASE + 3n. Als de bits in de PIN registers niet hetzelfde zijn als in de PORT registers word er een bericht gestuurd naar de serial console.

Resultaat:





Door pin A1 (F1) laag te pullen met een 1k resistor word het volgende bericht gestuurd:



Error bij pin 2 in port F (er word geteld van 0) dus dat klopt met A1 (F1).

### Non-blocking blinking led:

Text

Description automatically generated

In dit geval is LED\_A\_FREQ 2 en LED\_B\_FREQ 5 (in Hz). Ook hier worden de pins getoggled via de PORT registers.

Resultaat: <https://youtube.com/shorts/ZkLfpoYSBno>

Als meerdere leds wil blinken zou ik een 2d array maken waarin telkens led\_pin, freq, last\_pulse in staat waardoor alle leds via éen functie geüpdatet kunnen worden door de index door te geven. Deze functie kan dan via een loop gecalled worden. Dit heb ik in dit geval niet gedaan omdat ik maar twee leds heb

### Non-blocking debounce + button function:

Voor het debounce circuit heb ik mosfets gebruikt omdat ik maar een werkende button heb voor de rest werkt alles hetzelfde. Text

Description automatically generated

Dit is de code snippet van de button class deze class houd elke variable voor het debouncen en een function pointer die gecalled word als de state veranderd.



In deze code snippet word de function pointer type defined voor de button class

A screenshot of a computer

Description automatically generated with low confidence

Hier worden de buttons geinializeerd met de pin number delay en functie die gecalled word als de state veranderd

Text

Description automatically generated

In de loop worden de buttons geupdated en de toggle functies defined. Hier zie je ook dat de leds alleen getoggled worden als de button\_state high is en word dus alleen uitgevoerd op rising edge.

Resultaat: [https://youtube.com/shorts/-94sOB9wIrg](https://youtube.com/shorts/-94sOB9wIrg?feature=share)

### Trafic control:

Voor de trafic control code heb ik dezelfde button class als bij de vorige challenge gebruikt samen met een nieuwe trafic light class die alle variables en functies voor kleuren veranderen bevat. Het oranje licht duurt 3.5 seconden en het automatiche switchen duurt 10 seconden. (in de code staat overal “Yellow” omdat ik alleen maar gele lampjes had)

Text

Description automatically generated

In dit code snippet zie je een enum met alle mogelijke states en alle variables van de class. De waardes van de states in de enum zijn ook gelijk de index van de pin in de array pins van de Trafic\_Light class. Dit geld alleen niet voor de NONE state omdat het knipperen van oranje meer code nodig heeft.

Text

Description automatically generated

Hier word de state en dus ook de pins aangepast. Hier zie je ook dat de NONE state word omgezet in oranje.

Graphical user interface, text

Description automatically generated with medium confidence

Hier worden alle variables voor transitie geïnstalleerd.

Text

Description automatically generated

En hier word gecheckt of er iets veranderd moet worden.

Text

Description automatically generated

Deze functies worden gecalled als de buttons (sensors) geactiveerd worden.

Text

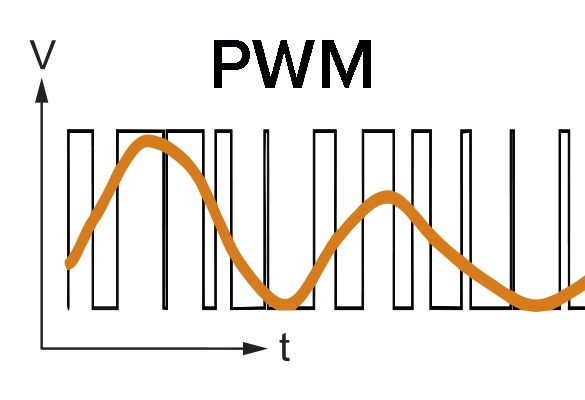
Description automatically generated

En ten slotte word er in de loop functie alle objects geupdated en gechecked of er een transitie nodig is.

Resultaat: <https://youtube.com/shorts/K1wBo79kHvg>

### I/O Techniques:

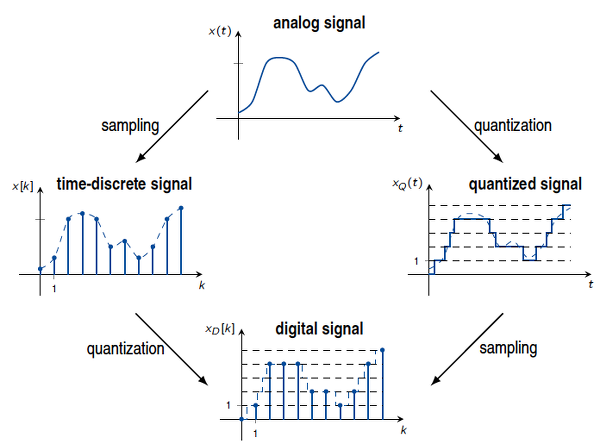
#### Digital:

Digital IO is de simpelste techniek en word ook het meest gebruikt met sensors omdat dit (bijna) geen processing nodig heeft en het snelste is. Het enige nadeel is dat het alleen een ON en OFF signaal kan sturen. Er bestaat wek een soort mix tussen Digital en Analoog en dat is PWM (pulse width modulation) met deze techniek kan je meer data versturen door een enkele pin door het signaal voor een gedeeltes van een tijdsperiode aan en uit te laten staan. Door het signaal voor een gedeelte aan of uit te laten staan kan je een soort analoog signaal nadoen (dit is geen echt analoog signaal daarom is het een mix tussen digital en analoog). 

hier zie je ook welk analoge signaal dit PWM signaal benadert

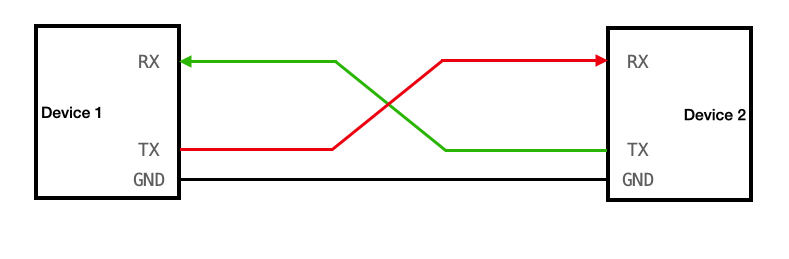
#### Analoog:

Een analoog signaal is een signaal met een bepaalde voltage tussen bijvoorbeeld 0v en 5v of 0v en 3.3v. deze voltage kan dan afgelezen worden doormiddel van de ADC peripheral die op meeste microcontrollers zit door de voltage range op te delen en te kijken waar de voltage ligt op deze verdeling.

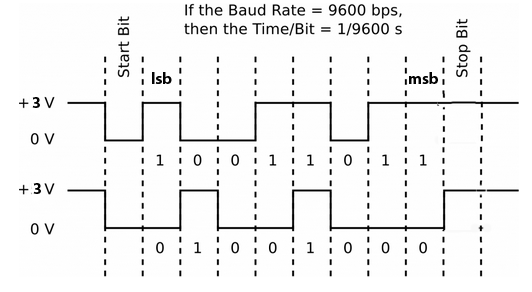


#### UART / USART:

UART (universal asynchronous receiver-transmitter) / USART (universal synchronous-asynchronous receiver-transmitter) is een full-duplex data protocol dat betekend dat deze data tegeleikertijd heen en weer kan sturen. Het sturen van deze data wordt bit voor bit via twee of drie draaden gedaan: TX, RX en CLK bij USART. De TX en RX draden moeten omgedraaid worden tussen devices omdat deze bij elk device op de zelfde manier gebruikt word namelijk: de TX voor het versturen van data (transmit) en RX om de data te ontvangen (receive). RX en TX zijn high als ze idle zijn.



De bits worden op een afgesproken snelheid verstuurd en ontvangen. Deze snelheid moet tussen beide devices gelijk zijn omdat omdat anders de bits op het verkeerde moment opgenomen worden deze afgesproken waarde heet de baud rate (of bit rate).



In het geval van USART is een gedeelde clock (CLK draad) toegevoegd waardoor de baud rate niet perse afgesproken hoeft te worden maar door een van de devices (meestal de master) geregeld te worden. Dit zorgt er voor dat er minder fouten gemaakt worden met het opnemen van de bits ook kan de clock uitgerekt worden als een van de microcontrollers het stuur moment niet haalt. Omdat deze clock line voor stabiliteit zorgt kan USART sneller werken dan UART terwijl deze betrouwbaar genoeg blijft.

Rectangle

Description automatically generated with medium confidence

Er zijn nog een paar andere afsplitsingen van het UART/USART protocol zoals bijvoorbeeld RS-232 maar dit wordt bijna niet meer gebruikt dus dit ga ik niet uitleggen.

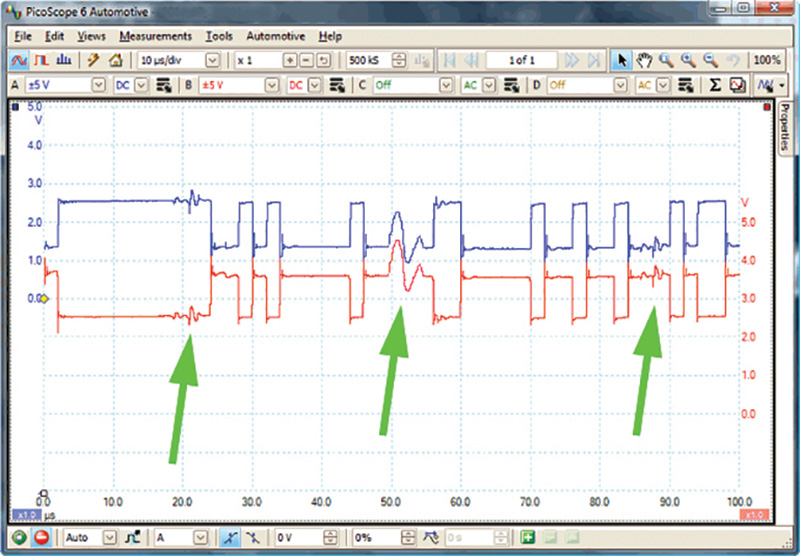
#### CAN:

Een CAN (controller area network) bus lijkt heel erg op UART omdat dit ook een asynchrone seriele protocol is. Het grootste verschil is dat CAN een high en low data pin heeft later wordt uitgelegd waarom. Ook is het half-duplex dat betekend dus dat de data maar een kant op kan per keer. Omdat er geen clock pin is moet de baud rate weer afgesproken worden.

A picture containing diagram

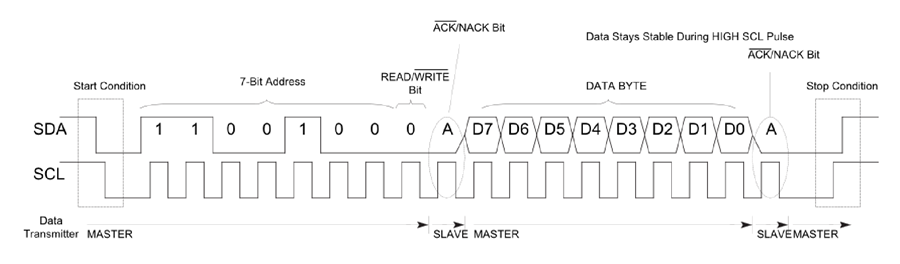
Description automatically generated

Doordat er data low en high pins zijn kunnen errors in het signaal makkelijk gedetecteerd en ontdaan worden door een van de signalen om te draaien en het gemmidelde uit te rekenen. Deze methode werkt omdat de meeste errors in signalen veroorzaakt worden door inductie (waarmee je je telefoon drradloos kan opladen) bij inductie wordt energie draadloos overgedragen door wisselende stroom in een andere draad. Hierdoor wordt dezelfde error op de low en high uitgeoefend omdat deze nu geen overgestelde van elkaar zijn wordt de error bij het omdraaien en optellen uit het signaal gefilterd.



#### I2C:

De I2C (inter intergrated circuit) bus lijkt heel erg op de USART omdat ze beide synchronous serial protcols zijn. Het vershil is dat I2C half-duplex is en dat elke transmissie begint met een device adres en een R/nW bit zodat dezelfde I2C bus voor meerdere devices gebruikt kan worden terwijl de per device communicatie behouden kan worden. De I2C bus bestaat uit 2 draaden SCL (serial clock) en SDA (serial data) SCL en SDA zijn high als ze idle zijn. Verder moet na elke byte een acknolagement bit komen van het ontvangende device.



A picture containing text, antenna

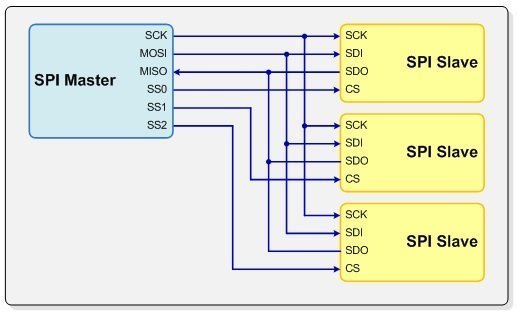
Description automatically generated

#### SPI:

SPI is een soort van full-duplex versie van I2C het enige verschil is dat er geen device adressen en R/nW bits gestuurd worden want in plaats van een device adres wordt de CS/SS (Chip Select / Slave Select) pin laag gezet waardoor de chip geselecteerd word. Voor de rest werkt het sturen van data over de twee data pins: MISO (master in slave out) en MOSI (master out slave in) hetzelfde als bij I2C. De data en clock pins zijn voor elke chip gedeeld maar de CS of SS pin is bij elke chip apart.

Text

Description automatically generated with low confidence



#### Sensors:

*<Hier beschrijf je de door jou gemaakte project POC’s, die je typisch doet in sprint 1 met daarin per POC de volgende onderdelen:*

* *Een beschrijving van de aanpak, eventueel met relevante code snippets.*
* *Indien van toepassing een flowchart*
* *Het testresultaat! Dit kan een beschrijving zijn of een screenshot van de Serial Monitor.*
* *De gebruikte bronnen.>*

**<**Daarnaast beschrijf je de POC’s die je hebt gedaan buiten het project om. Deze doe je vooral om de leerdoelen goed onder de knie te krijgen.>

## Sprint 2: Tussenproduct

*<Hier beschrijf je de door jou geleverde bijdrage aan het project tussenproduct met daarin de volgende onderdelen:*

* *Een beschrijving van de functionaliteit van het tussenproduct.*
* *Een tekening van de (voorlopige) hardware opstelling (bv. in Tinkercad, Fritzing, foto of op papier).*
* *Een (voorlopige) flowchart.*
* *Het testresultaat! Dit kan een beschrijving zijn of een screenshot van de Serial Monitor.*
* *De gebruikte bronnen*.>

**<**Daarnaast beschrijf je de POC’s die je hebt gedaan buiten het project om. Deze doe je vooral om de leerdoelen goed onder de knie te krijgen.>

## Sprint3: Eindproduct

*<Hier beschrijf je de door jou geleverde bijdrage aan het project eindproduct met daarin de volgende onderdelen:*

* *Een beschrijving van de functionaliteit van het eindproduct.*
* *Een tekening van de uiteindelijke hardware opstelling (bv. in Tinkercad, Fritzing, foto of op papier).*
* *De uiteindelijke flowchart.*
* *Het testresultaat! Dit kan een beschrijving zijn of een screenshot van de Serial Monitor.*
* *De gebruikte bronnen*.>

**<**Daarnaast beschrijf je de POC’s die je hebt gedaan buiten het project om. Deze doe je vooral om de leerdoelen goed onder de knie te krijgen.>

# Reflectie / evaluatie

## Waar ben ik trots op?

*<vul in>*

## Wat doe ik een volgende keer anders?

*<vul in>*

## Welke formatieve indicatie zou ik mezelf geven voor de verdieping Technology?

|  |  |  |
| --- | --- | --- |
| Leeruitkomst verdieping Technology (bouwt voort op de oriëntatie) | | |
| Onderdeel | Criterium | Rating |
| Interactieve embedded systemen | Je product kan communiceren met een ander systeem volgens een eigen gedefinieerd protocol inclusief parameters waarbij ongeldige berichten worden afgevangen. | *<vul in U S G O met motivatie>* |
| Programmeren | Je past alle imperatieve programmeer-concepten en de volgende OO concepten toe: objects, classes en encapsulation, d.w.z.: constructors, private fields, properties en methods. De focus hierbij is op leesbare (b.v. naamgeving, indentation) en onderhoudbare programma’s en robuustheid van het product. | *<vul in U S G O met motivatie>* |
| Sensoren en Actuatoren | Je past extra sensoren en actuatoren toe waarvan een eigen analyse is gedaan. | *<vul in U S G O met motivatie>* |
| Verschillende I/O technieken | Naast de genoemde I/O technieken kun je ook pulsbreedtemodulatie en analoge input interpreteren en toepassen. | *<vul in U S G O met motivatie>* |