|  |  |
| --- | --- |
| Afbeeldingsresultaat voor atmega128  Rapportage document  Microcontrollers  29-01-2020  -Laatst aangepast op: 02-04-2020, versie 1.6 | Noah Walsmits & Vincent de Rooij  Periode 2.3 – microcontrollers |

Contents

[Versie beheer 2](#_Toc36729039)

[Rapportage week 1 3](#_Toc36729040)

[Opdracht A1 3](#_Toc36729041)

[Opdracht A2 4](#_Toc36729042)

[Opdracht B1 5](#_Toc36729043)

[Opdracht B2 9](#_Toc36729044)

[Opdracht B3 10](#_Toc36729045)

[Opdracht B4 10](#_Toc36729046)

[Opdracht B5 11](#_Toc36729047)

[Opdracht B6 11](#_Toc36729048)

[Opdracht B7a 12](#_Toc36729049)

[Rapportage week 2 13](#_Toc36729050)

[Opdracht A1 13](#_Toc36729051)

[Opdracht A2 14](#_Toc36729052)

[Opdracht B1 15](#_Toc36729053)

[Opdracht B2 15](#_Toc36729054)

[Opdracht B3 16](#_Toc36729055)

[Opdracht B4 17](#_Toc36729056)

[Opdracht B5 18](#_Toc36729057)

[Rapportage week 3 18](#_Toc36729058)

[Opdracht B1 18](#_Toc36729059)

[Opdracht B2 18](#_Toc36729060)

[Rapportage week 4 19](#_Toc36729061)

[Opdracht B1 19](#_Toc36729062)

[Opdracht B2 19](#_Toc36729063)

[Opdracht B3 20](#_Toc36729064)

[Rapportage week 5 20](#_Toc36729065)

[Opdracht C.a 20](#_Toc36729066)

[Opdracht C.b 21](#_Toc36729067)

[Opdracht C.c 22](#_Toc36729068)

[Opdracht C1a 23](#_Toc36729069)

[Opdracht C1b 25](#_Toc36729070)

# Versie beheer

|  |  |  |  |
| --- | --- | --- | --- |
| **Datum** | **Versie** | **Aanpassingen** | **Aanvullend commentaar** |
| 29-01-2020 | 1.0 | n.v.t./geen | Start versie |
| 05-02-2020 | 1.1 | Week 2 toegevoegd |  |
| 12-02-2020 | 1.2 | Week 3 toegevoegd | Verder gewerkt aan week 2, start met week 3 |
| 20-02-2020 | 1.3 | Toevoegingen aan week 3 |  |
| 04-03-2020 | 1.4 | Week 4 toegevoegd | Verder gewerkt aan week 3 |
| 11-03-2020 | 1.5 | Week 4 afgemaakt en week 5 toegevoegd |  |
| 12-03-2020 | 1.55 | Week 5 afgemaakt | Opdrachten afgemaakt, maar nog niet volledig verwerkt in het groeidocument, bewijs onderdelen zoals foto’s en video zijn er pas gedeeltelijk in verwerkt |
| 02-04-2020 | 1.6 | Layout aanpassingen, letter grote universeel gemaakt, onderdelen verplaatst, algemene code link per week geplaatst en uitleg week 2 opdracht B3 toegevoegd | Deze versie bevat nog niet alle bewijs materialen, zoals de filmpjes en foto’s |

# Rapportage week 1

**Code:** <https://github.com/VincentDeRooij/Microcontrollers-Avans2.3/tree/master/Microcontrollers/week1>

## Opdracht A1

**Essentie:**

De materialen verzamelen en klaarmaken. En door de user guide bladeren.

**Oplossing:**

Materiaal verzameld en opgezet.

**Reflectie:**

Ik had wat moeite met de MCU kaart maar voor de rest ging het wel goed met de hulp van de handleiding.

**Bewijs:**

<https://youtu.be/jI10g7l5yBI>

## Opdracht A2

**Essentie:**

Installeer AVR Studio, maak een nieuw project en gebruik de simulator.

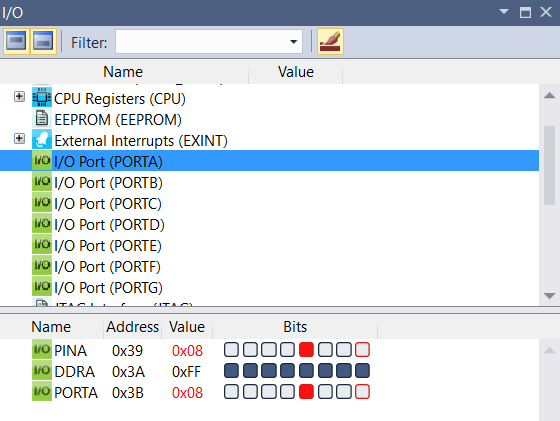
**Oplossing:**

Project gemaakt voor ATmega 128A, omdat die naam op de handleiding stond die kwam met het bord. Ik heb code van blackboard gekopieerd naar main.c. De simulator lijkt de code te draaien. Je kan de aansturing van de I/O bekijken met een knop bovenin. Uiteindelijk heeft het ook gewerkt op het echte bord.

**Reflectie:**

Atmel Studio werkt niet op een andere drive dan C:, dus ik heb het opnieuw moeten installeren. De I/O knop had ik niet zelf kunnen vinden.

**Bewijs:**



## Opdracht B1

**Essentie:**

Beantwoord deze vragen:

* Hoe groot is het program memory van de ATmega128?
* Wat is het adres van Data direction register van PORTE (DDRE)?
* Uit hoeveel byte bestaat de instructie ‘IN R3, PORTA’ ?
* Hoeveel RS232 poorten zitten er op het BIGAVR6 development board?
* Op welke pin van de microcontroller zit de ingang voor Analog digitaalconverter, channel 1?
* Hoe groot is het data geheugen van de microcontroller maximaal?
* Hoeveel I/O-registers zijn er op de ATmega128?
* De pinnen van PORTA kunnen met een weerstand naar 0 V (pull-down) of met een weerstand naar de +5V verbonden worden (pull-up). Hoe is dat standaard ingesteld op het BIGAVR6 development board?

**Oplossing:**

1: Het flash program memory is 128 kilobytes.

2: Onder de naam DDRE, het adres is 0x22.

3: De instructie bestaat uit één byte. De meeste register instructies worden in één cycle uitgevoerd. Op een 8-bit systeem kan die instructie dan niet meer dan een byte zijn.

4: Er zitten twee RS232 poorten op het bord.

5: De ingang voor analoog digitaal converter kanaal 1 zit op pin 60 of PF1.

6: De maximale grootte van het data geheugen is 64 kilobytes.

7: Er zijn 64 I/O registers.

8: Pull-up en pull-down staan standaard uit.

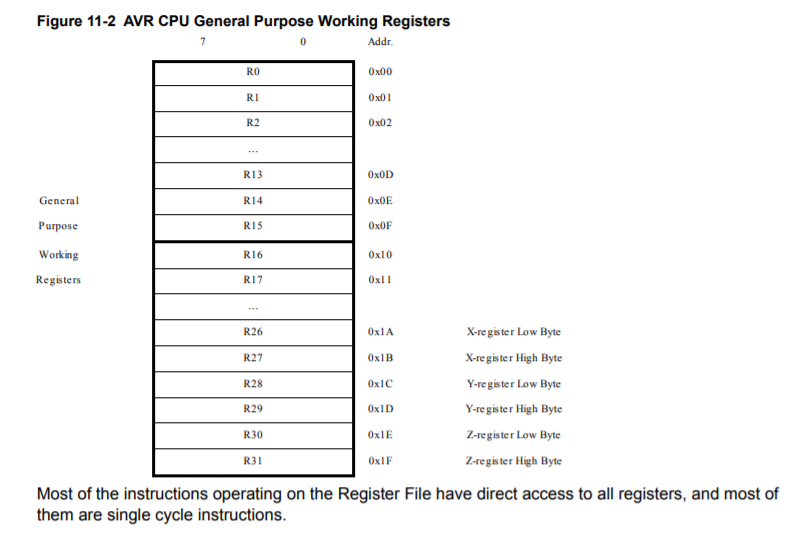
**Reflectie:**

De meeste antwoorden waren niet zo moeilijk om te vinden.

**Bewijs:**

Zie onderstaande foto’s

2:

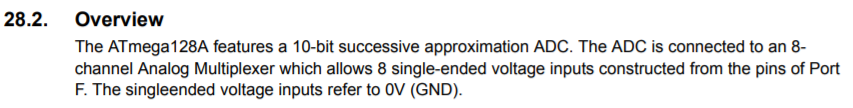


4:

A circuit board

Description automatically generated

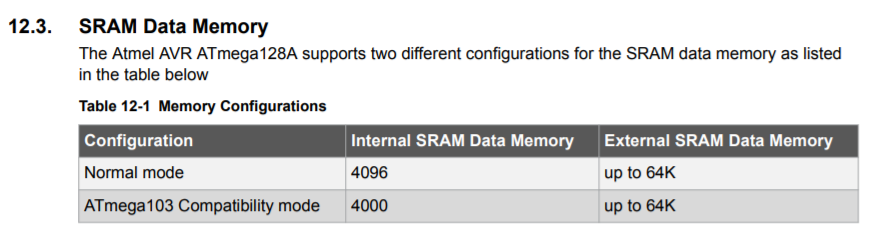
5:



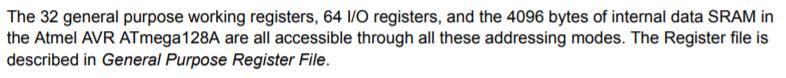
A close up of text on a white background

Description automatically generated

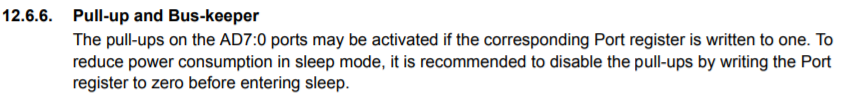
6:



7:



8:



**Bron:**

<https://www.tme.eu/Document/554724703930dfa21e190a7257787ec0/ATmega128A-DTE.pdf>

## Opdracht B2

**Essentie:**

Maak een applicatie die twee beurtelings twee LEDs om de 500ms laat knipperen.

**Oplossing:**

We passen de waarde van PORTD om de 500ms aan. Om een enkele bit op een positie te zetten hebben we een define gebruikt om dat sneller te doen

#define BIT(x) (1<<(x))

**Reflectie:**

Het gebruik van BIT(x) zorgt ervoor dat we makkelijk een pin of port op het bord aan kunnen geven. Port A5 is dan BIT(5) in plaats van 0b00100000.

**Bewijs:**

A circuit board

Description automatically generated

## Opdracht B3

**Essentie:**

Maak een applicatie die een LED laat knipperen als een knop in is gedrukt.

**Oplossing:**

Eerst wordt er gekeken of de knop in is gedrukt door naar de waarde van pin c te kijken. Als dat waar is wordt een LED aan en uit gezet.

if (PINC == 0x80)

**Reflectie:**

De waarde van DDRC is belangrijk als er met de knoppen gewerkt wordt, het veranderen hiervan hadden we eigenlijk in de methode van de opdracht moeten doen.

**Bewijs:**

Volgende versie

## Opdracht B4

**Essentie:**

Implementeer een looplicht met 50ms tussen elke verandering. Geef ook aan hoe er gemeten kan worden of er daadwerkelijk 50ms tussen iedere verandering zit.

**Oplossing:**

Met een for loop wordt iedere LED om de beurt even aangezet met steeds 50ms ertussen. Je kan de timing meten door op het begin van de cycle een timer te beginnen en die te stoppen aan het einde van de cycle. Door de tijd te delen door het aantal veranderingen is te vinden of er tussen iedere verandering 50ms zit.

**Reflectie:**

Om de wait methode goed te laten met de hardware hebben we de wachttijd vermenigvuldigd met 8 in de wait methode zelf.

**Bewijs:**

<https://youtu.be/jI10g7l5yBI?t=25>

## Opdracht B5

**Essentie:**

Implementeer een looplicht met een datastructuur zoals een C array.

**Oplossing:**

We hebben het patroon uit het voorbeeld gebruikt. We probeerden het patroon te gebruiken in een for loop maar we hadden moeite met het bepalen van de lengte van de array. Uiteindelijk volgden we het voorbeeld en hadden we een while loop gebruikt.

**Reflectie:**

Het bepalen van de lengte van een array zal waarschijnlijk binnenkort in de embedded programming lessen voorkomen.

**Bewijs:**

<https://youtu.be/jI10g7l5yBI?t=35>

## Opdracht B6

**Essentie:**

Maak een applicatie die een LED met 1 Hz laat knipperen. Als er op een knop gedrukt wordt moet de LED met sneller knipperen. Als er weer op de knop gedrukt wordt gaat de LED weer terug naar de oude snelheid.

**Oplossing:**

Wij hebben een delay variabele die eerst op 1000 wordt gezet. Terwijl het programma aan staat checkt het of de knop in is gedrukt en wacht het daarna met de tijd die de delay variabele aangeeft. Als de knop in is gedrukt en het delay 1000 is, wordt het delay op 250 gezet. Als de knop in gedrukt is en het delay niet 1000 is, wordt het delay weer op 1000 gezet.

if (PINC == 0b00000001) {

if (delay == 1000) {

delay = 250;

} else {

delay = 1000;

}

}

**Reflectie:**

De toggle werkt niet terwijl het programma wacht. Dit zou op te lossen zijn door de tijd te meten en met een if statement het knipperen uit te voeren.

**Bewijs:**

Volgende versie

## Opdracht B7a

**Essentie:**

Implementeer de applicatie zodat het voldoet aan het state machine diagram.

**Oplossing:**

Wij hebben twee enums gemaakt om de states en events te representeren. Voor iedere enum is ook een variabele om de huide waarde weer te geven.

Eerst gaat het programma door een switch case om de huidige waarde van het event te veranderen als de juiste knop in is gedrukt. Daarna gaat het programma door een tweede switch case om de acties van de huidige state uit te voeren en deze ook mogelijk te veranderen. Aan het einde is een wait methode gebruikt omdat het porgramma anders door meerdere states heen gaat als de knop even ingedrukt is. Dit maakt het wel moeilijker om te bedienen.

case START:

if (btnpressed == D5) {

currentstate = STATE2;

} else if (btnpressed == D6) {

currentstate = STATE1;

}

PORTA = 0b00000001;

break;

**Reflectie:**

Een switch case is misschien niet de meest onderhoudbare oplossing, maar in deze situatie heeft het goed gewerkt.

**Bewijs:**

<https://youtu.be/jI10g7l5yBI?t=47>

# Rapportage week 2

Code: <https://github.com/VincentDeRooij/Microcontrollers-Avans2.3/tree/master/Microcontrollers/week2>

## Opdracht A1

**Essentie:**

Lees drie artikels en hiervan beschrijf de kerngedachten.

**Oplossing:**

**The good:** Met interrupts kan je bepaalde gebeurtenissen afhandelen zodra ze gebeuren, zonder dat die gebeurtenissen afgewacht moeten worden door het programma. Interrupts kunnen gebruikt worden als de timing belangrijk is. Als de timing minder belangrijk is kan polling gebruikt worden. De prioriteit van een interrupt is in te stellen via de hardware.

**The bad:** Als je te veel interrupts gebruikt dan kan je ervoor zorgen dat het we werking van je programma stoort omdat je programma niet goed werkt tussen de interrupt calls. Want als je te veel code hebt in je interrupt zorgt het ervoor dat het programma wacht totdat de interrupt is voltooid. Dit zijn dan te lange interrupts. Ook kan je te veel kleine interrupts hebben wat voor hetzelfde effect zorgt.

**The ugly:** Wanneer variabelen in de code tussentijds veranderd worden door interrupts, kan dat voor problemen zorgen stel, je hebt een if statement die controlleert op een GETAL variabele dat niet ‘0’ mag zijn. Momenteel is de globale variabel waarde van GETAL = 8, dus de if statement is true en de code doorloopt de code van de if-statement, op dat moment is er een interrupt, waardoor de code wordt gepauzeerd. De globale variabel van GETAL wordt veranderd naar ‘0’, de code wordt vervolgens vervolgd, maar wel met de ‘0’ waarde die niet in de code mocht voorkomen. Dit kan gebeuren, mits je interrupts niet op de juiste manier afhandelt, dit kan je oplossen door een locale variabele te introduceren die de globale variabele check en overneemt aan het begin van de code. Hierdoor kan de code niet tussentijds veranderd worden.

**Reflectie:**

We hebben de artikels verdeeld en goed door gelezen wat er stond. Hierna hebben we met z’n gezamelijk alle belangrijke punten samengevat. Deze hebben we ieder appart opgeschreven en besproken.

## Opdracht A2

**Essentie:**

Gebruik de LCD module simulator om de werking te bestuderen. Realiseer het volgende in de simulator:

1. De initialisatiecode om het LCD in 4-bits mode te initialiseren (dia pagina 10). Welke

data, commando’s en instellingen moet je maken om het LCD te initialiseren? Maak een

tabel en vertaal deze handelingen in een C functie.

2. Verklaar de functie van de D0-D7 pinnen, de E pin en de RS pin.

3. Het afbeelden van enkele karakters op het display. Welke data, commando’s

en/of instellingen moet je maken? Vertaal deze handelingen in een C functie.

4. Het schuiven van alle tekst naar rechts. Welke data, commando’s en/of instellingen

moet je maken? Vertaal deze handelingen in een C functie

**Oplossing:**

0x20 om het in 4 bit modus te zetten. (wanneer het lcd display in 8 bits modus staat in de simulator)

0x00 voor de hoge nibble

0x0F voor de lage nibble om het scherm aan te zetten

0x00 voor de hoge nibble

0x02 cursor home, lage

2: D0/D7 wordt gebruikt om de bits van het commando door te geven. E wordt gebruikt om de input te verzenden. RS is gebruikt om van command modus naar character modus te veranderen.

3: Om een character te ‘’printen’’ op het LCD moet eerst de modus op RS gezet worden de upper bits verstuurd worden, daarna de lower bits, zo kan je elke willekeurige character maken.

4: Om een reeks aan characters naar rechts te verplaatsen moet er eerst in de upper nibble een command met hex waarde 0x01 gestuurd worden, daarna moet er in de lower nibble een command met de hex waarde 0x0C gestuurd worden.

**Reflectie:**

Achteraf zijn we hier heel lang tegen aangelopen. Het LCD scherm was een van de lastigere onderdelen. Vooral vanwege de complexiteit en de hoeveelheid verschillende commands waar we tegenaan zijn gelopen. In onze code is het dan ook niet goed gelukt om het LCD display werkende te krijgen.

**Bewijs:**

In filmpje

## Opdracht B1

**Essentie:**

De werking verklaren en onderzoeken van interrupts d.m.v. demo code te bestuderen.

**Oplossing:**

Er zijn twee interrupts gedefinieerd. Een interrupt zet het licht uit en de andere zet het licht aan. De interrupts gaan in werking als de juiste poort een signaal krijgt. Deze interrupts voeren code uit terwijl de main methode bezig is.

**Reflectie:**

**Bewijs:**

In filmpje

## Opdracht B2

**Essentie:**

Het werken met interrupts, d.m.v. het besturen van een lampje, hierdoor is het makkelijker te visualiseren.

**Oplossing:**

Er is een globale variable de de index bijhoud van het looplicht. Als ISR0 afgaat gaat de lamp op die index aan en de index gaat omhoog met 1 of teruggezet naar 0, als die al te hoog is.

**Reflectie:**

**Bewijs:**

In filmpje

## Opdracht B3

**Essentie:**

Onderzoeken hoe een 7-segmenten display werkt samen met knoppen.

**Oplossing:**

while (1) {

if (PINA & (1 << 0) && PINA & (1 << 1)) {

//up and down pressed

currentdigit = 0;

displayDigit(currentdigit);

} else if (PINA & (1 << 0)) {

//down pressed

currentdigit--;

displayDigit(currentdigit);

} else if (PINA & (1 << 1)) {

//up pressed

currentdigit++;

displayDigit(currentdigit);

}

wait(250);

}

Wanneer er in het PINA register bit 0 en bit 1 op hoog staan wordt de currentDigit variabele gereset. Wanneer bit 0 in het PINA register op hoog staat gaat de huidige waarde met 1 naar beneden. Wanneer bit 1 in het PINA register op hoog staat gaat de huidige waarde met 1 naar boven.

**Reflectie:**

We hebben voor deze code geen gebruik hoeven te maken van een interrupt. De oplossing voor deze opdracht was niet al te lastig te ontwikkelen. We moesten even kijken hoe we het 7-segmenten display aan konden sturen, maar daarna konden we vrij eenvoudig met deze oplossing komen.

**Bewijs:**

In filmpje

## Opdracht B4

**Essentie:**

Een leuke tussen opdracht met het 7-segmenten display.

**Oplossing:**

Door een Array van waardes te doorlopen d.m.v. een for loop kan elk bitje aangestuurd/veranderd worden. Hierdoor kan er een looplicht gemaakt worden.

const unsigned char

PATTERN[] =

{

// dP g f e d c b a

0b00000001,

0b00000010,

0b00000100,

0b00001000,

0b00010000,

0b00100000,

0b00010000,

0b00001000,

0b00000100,

0b00000010,

0b00000001,

void \_7Segment\_Pattern()

{

DDRC = 0b11111111; // PORTC all output

while (1)

{

for (int i = 0 ; i < sizeof(PATTERN) / sizeof(PATTERN[0]); i++ )

{

PORTC = PATTERN[i];

wait(100);

}

}

}

**Reflectie:**

Het werken met het 7-segmenten display ging in het begin niet zonder problemen, eerst hadden we problemen met de juiste aansluiting van de draden, hierdoor kregen we verkeerde waardes. Uiteindelijk is dit opgelost en hebben we de opdrachten met het 7-segmenten display goed kunnen afronden.

**Bewijs:**

--- Plak hier de link naar het filmpje/foto’s

## Opdracht B5

**Essentie:**

Het begrijpen van het LCD scherm en de onderliggende aansturing d.m.v. bit/hex waardes

**Oplossing:**

Deze opdracht was niet gelukt omdat wij moeilijkheden hadden met het LCD bord.

**Reflectie:**

**Bewijs:**

In filmpje

# Rapportage week 3

Code: <https://github.com/VincentDeRooij/Microcontrollers-Avans2.3/blob/master/Microcontrollers/week3>

## Opdracht B1

**Essentie:**

Geef weer hoe vaak er op een toets gedrukt wordt met het gebruik van een counter.

**Oplossing:**

We hebben de hoeveelheid binair weergegeven op de poort LED’s, omdat het ons niet gelukt was om het LCD scherm hiervoor te gebruiken. De counter is zo te initialiseren:

TCCR1B |= (1 << CS10); // Set up timer

Met TCNT1 kunnen wij de waarde van onze counter pollen en resetten.

if (TCNT1 >= 49999)

TCNT1 = 0; // Reset timer value

**Reflectie:**

**Bewijs:**

In filmpje

## Opdracht B2

**Essentie:**

**Oplossing:**

Deze opdracht was niet volledig gelukt, uiteindelijk hadden we er teveel tijd aan besteed en hebben we besloten door te gaan naar de volgende opdracht.

**Reflectie:**

**Bewijs:** In filmpje

# Rapportage week 4

Code: <https://github.com/VincentDeRooij/Microcontrollers-Avans2.3/blob/master/Microcontrollers/week4>

## Opdracht B1

**Essentie:**

Laat de 10 bits AD-waarde van kanaal 1 op de led’s van Poort A en B zien.

**Oplossing:**

Zet de jumper op het juiste kanaal en initialiseer de ADC.

ADMUX = 0b01100001; // AREF=VCC, result left adjusted, channel1 at pin PF1

ADCSRA = 0b11100110; // ADC-enable, no interrupt, start, free running, division by 64

Je kan de high en low bits van de ADC op de poorten zetten door ADCH en ADCL te gebruiken. Zorg ervoor dat de poort F op input staat en gebruik dan de draaiknop.

**Reflectie:**

We hadden eerst wat moeite met de ADC juist aan te sluiten. We wisten niet echt welk resultaat we nodig hadden. Na wat hulp van een docent was het wel gelukt.

**Bewijs:**

In filmpje

## Opdracht B2

**Essentie:**

Verander de code van de vorige opdracht om de AD-conversie uit te voeren op aanvraag en niet constant.

**Oplossing:**

We doen de initialisatie wanneer een knop ingedrukt is. De laatste drie bits van ADMUX kunnen we veranderen om het kanaal te kiezen. Terwijl de knop ingedrukt is worden de hoge bits op poort A afgebeeld. Als de knop niet ingedrukt is wordt de initialisatie ongedaan gemaakt.

**Reflectie:**

Het veranderen van het kanaal hadden we getest door de jumper te veranderen en dan te kijken of de lampjes veranderen met de draaiknop.

**Bewijs:**

In filmpje

## Opdracht B3

**Essentie:**

Sluit een temperatuursensor aan als analoge input. Lees dit dan uit met de ADC.

**Oplossing:**

Dit vereist geen code aanpassingen vergeleken met opdracht B1. We hebben wel het kanaal veranderd maar de rest is hetzelfde. Door de temperatuursensor met het easyproto bord aan te sluiten aan het bord wordt de output van de sensor weergegeven. Wanneer eerst de output van het draaiknopje gebruikt werd, werd nu de output van de sensor gebruikt.

**Reflectie:**

Goed opletten of de tekening van de sensor van boven of onder is afgebeeld, anders kan je de sensor verkeerd aansluiten. Let er ook op dat je het juiste kanaal gebruikt, anders meet je de verkeerde waardes.

**Bewijs:**

In filmpje

# Rapportage week 5

Code: <https://github.com/VincentDeRooij/Microcontrollers-Avans2.3/blob/master/Microcontrollers/week4>

## Opdracht C.a

**Essentie:**

Met deze opdracht moesten we met behulp van het SPI interface, wat enorm lijkt op I2C van philips communiceren met de 4 7-segmenten displays. Deze opdracht was een voorbereiding en kennismaking met SPI

**Oplossing:**

Door de register waarde van de scan limit (0x0B) op de hexcode van 0x03 te veranderen konden we alle vier de 7-segementen displays aandoen.

**Reflectie:**

We hebben aan de code nauwelijks onderdelen veranderd, bij het runnen van de code hebben we eerst gekeken wat er gebeurde, vervolgens konden wij zien dat alleen de eerste 2 aanstonden. Hierna de code onderzocht, om te kijken welke methode werd gebruikt om de displays aan te zetten.

**Bewijs:**

In filmpje

## Opdracht C.b

**Essentie:**

Na het onderzoeken van de voorbeeld code en de werking van de displays. Was het tijd om zelf aan de slag te gaan en code toe te voegen aan de gegeven voorbeeld code.

**void setValueForDigitDisplay(unsigned char** *digitDisplay***, unsigned char** *value***)**

**{**

spi\_slaveSelect(0); // Select display chip

spi\_write(*digitDisplay*); // Register 0B: Shutdown register

spi\_write(*value*); // -> 1 = Normal operation

spi\_slaveDeSelect(0); // Deselect display chip

**}**

**Oplossing:**

Door het volgende code blok te runnen (met ondersteuning van een hulp methode):

**void set\_1\_2\_3\_4\_onDigitDisplays()**

**{**

for (**char** i =4; i>0; i--)

**{**

setValueForDigitDisplay(*i, 5-i*);

wait(*500*);

**}**

**}**

Konden we het voor elkaar krijgen om de cijfers 1-2-3-4 op de segmenten displays te krijgen.

**Reflectie:**

Deze opdracht ging in het begin nogal lastig, omdat wij eerst dachten dat het communiceren naar de displays met bits aangegeven moest worden. Dus om display 1 aan te zetten moet bit 0 aangepast worden naar 1, maar achteraf was dit dus niet zo. We hoefde enkel alleen bij de spi\_write() methode de waarde in te vullen die overeenkomt met het display. Dus voor display 1 de waarde 1, voor display 2 de waarde 2, enz... Hierna konden we gemakkelijk de cijfers 1-2-3-4 op de displays krijgen.

**Bewijs:**

In filmpje

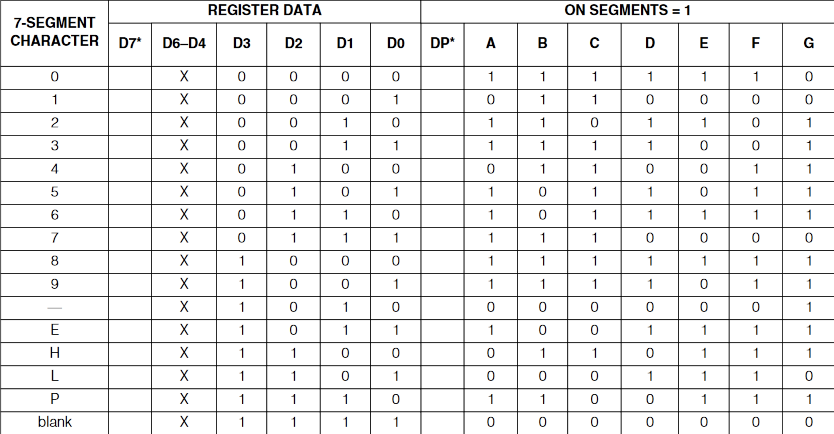
## Opdracht C.c

**Essentie:**

Ontdek meerdere functionaliteiten en onderzoek de datasheet verder van de MAX7219.

**Oplossing:**

Door te kijken naar de datasheet zijn we er achter gekomen hoe je verschillende standaard tekens op de 7-segmenten displays kan krijgen.



Het schrijven van characters (mits -decode modus aanstaat) kan gedaan worden door de waarde 10(-), 11(E), 12(H), 13(L), 14(P), 15( “BLANK”/NIKS ). Zodra decode modus uitstaat kan je 8 bits waardes sturen en je individueel de 7-segementen LED’s besturen.

**Reflectie:**

Wij hadden deze opdracht achteraf overgeslagen en waren gelijk door gegaan met opdracht C1.

**Bewijs:**

In filmpje

## Opdracht C1a

**Essentie:**

De eerste tussenopdracht om te zorgen dat de 7-segmenten displays samen kunnen werken.

**Oplossing:**

Deze opdrachten hebben wij een timer gemaakt die naar boven optelt. Dat hebben we voor elkaar gekregen met de volgende code stukjes (voor de volledige code zie week 5, methode countUp() ):

**char** indexFirst = *0*; // display 1

**char** indexSecond = *0*; // display 2

**char** indexThirth = *0*; // display 3

**char** indexFourth = *0*; // display 4

for (**char** i = 0; i < *9999*; i++ )

}

if (indexFirst == *9*)

{

indexFirst = *0*;

indexSecond++;

if (indexSecond == *9*)

{

indexSecond = *0*;

indexThirth++;

if (indexThirth == *9*)

{

.....

}

}

.....

}

indexFirst++;

setValueForDigitDisplay(*1, indexFirst*); // value for digit 1;

setValueForDigitDisplay(*2, indexSecond*);

setValueForDigitDisplay(*3, indexThirth*);

setValueForDigitDisplay(*4, indexFourth*);

wait(*100*);

}

**Reflectie:**

Het was even puzzelen met deze opdracht, vooral omdat we er later pas achter kwamen waarom niet alle displays wilde werken aan het begin. Dit kwam omdat we eerst communiceerde met displays 0-3, maar het moest zijn 1-4. Na dat we hier achter kwamen moesten we even kijken wat de beste aanpak was. Uiteindelijk is het ons gelukt om de methode countUp() te realiseren.

**Bewijs:**

In filmpje

## Opdracht C1b

**Essentie:**

De tweede tussenopdracht om te zorgen dat de 7-segmenten displays samen kunnen werken. Deze keer in een lastigere vorm dan opdracht c1a.

**Oplossing:**

Deze opdrachten hebben wij een timer gemaakt die naar van -999 boven optelt. Dat hebben we voor elkaar gekregen met de volgende code stukjes (voor de volledige code zie week 5, methode showMinusDemo() ):

**char** minusToken = *10***; // de waarde van het min teken.**

**char** indexFirst = *9*; **// display 1**

**char** indexSecond =*9*; **// display 2**

**char** indexThirth = *9*; **// display 3**

**char** indexFourth = *9*; **// display 4**

for (**char** i = 0; i < *999*; i-- )

}

if (i <= *99*)

{

clearDisplay(*4*);

indexThirth = minusToken;

}

else if (i <= *9*)

{

clearDisplay(*3*);

indexSecond = minusToken;

}

if (indexFirst == *0*)

{

indexFirst = *9*;

indexSecond--;

if (indexSecond == *0*)

{

indexSecond = *9*;

indexThirth--;

if (indexThirth == *0*)

{

}

}

}

indexFirst--;

setValueForDigitDisplay(*1, indexFirst*); // value for digit 1;

setValueForDigitDisplay(*2, indexSecond*);

setValueForDigitDisplay(*3, indexThirth*);

setValueForDigitDisplay(*4, indexFourth*);

wait(*100*);

}

**Reflectie:**

Ook hier was het even puzzelen met deze opdracht, we konden het uiteindelijk wel voor elkaar krijgen, maar gek genoeg werkte het niet volledig. Na een tijdje kwam die op de -70 en bleeft die vast haken. Wij zijn er niet achter kunnen komen waarom dit nou zo is.

**Bewijs:**

In filmpje