S1098012@student.hsleiden.nl

S1098012

Imthe1

Door Rick van Megen

Inhoud

[Opdracht 1 - Blink once to accept 2](#_Toc528167983)

[Opdracht 2 – Count me in 4](#_Toc528167984)

[Opdracht 3 – De dobbelsteen 8](#_Toc528167985)

[Opdracht 3b – Potmeter met geluid 11](#_Toc528167986)

[Opdracht 4 – PWM interrupt op RGB 13](#_Toc528167987)

[Opdracht 4b – Lamp aan/uit op usart 15](#_Toc528167988)

[Opdracht 4c - RGB aan/uit op usart 17](#_Toc528167989)

[Opdracht 5a - ET Phone home 19](#_Toc528167990)

[Opdracht 5b - HD44780 Display 20](#_Toc528167991)

[Opdracht eindopdracht 22](#_Toc528167992)

# Opdracht 1 - Blink once to accept

**Youtube Video Link**   
<https://youtu.be/FdWvGPzlj20>

**Samenvatting**Opdracht 1 is de blinking led opdracht. Bij de blinking led opdracht is het de bedoeling dat ze om de beurt knipperen. Ik heb 2 ledjes aangesloten op port D12 en port D8 met samen een weerstandje, die vervolgens naar de GRND loopt.

Vervolgens in de code (in eclipse) PORTB aangeroepen als output pins, En daarna in een delay wordt de variabele ‘portb’ zijn binaire waarde gewijzigd. Waardoor er elke seconden een andere output op ‘hoog’ zal staan. Hierdoor zullen de lampjes om en om knipperen.

**Bronvermeldingen**<https://github.com/hexagon5un/AVR-Programming> – Voorbeelden van het boek

**Uitwerking code**

|  |
| --- |
| //include van librarys voor code  **#include** <avr/io.h>  **#include** <util/delay.h>  **int** **main** (**void**)  {  // alle pins worden op output gezet.  DDRB = 0xFF;    **while** (1) { // zichzelf herhalen met een while 1    PORTB = 0x0000001; //laatste pin van port b staat op aan  **\_delay\_ms**(1000); //delay 1s  PORTB = 0x0000010; //1-na laatste pin van port b staat op aan!  **\_delay\_ms**(1000); //delay 1s  }  **return** 0;  } |

|  |  |
| --- | --- |
| **Breadbord schema** | **Afbeelding setup** |
|  | C:\Users\Rick\AppData\Local\Microsoft\Windows\INetCache\Content.Word\279ef32c-a598-4252-bfbb-3970152bc494.jpg |

# Opdracht 2 – Count me in

**Youtube Video Link**   
[**https://youtu.be/-MT0tYCl8-Q**](https://youtu.be/-MT0tYCl8-Q)

**Samenvatting**Opdracht 2 ‘count me in’ gaat om het optellen in een 7 led display. Hierbij is het de bedoeling dat ‘bit shifting’ in de code gebruikt wordt. Het tweede gedeelte van de opdracht is dat er 2 “7 led displays“ werken en die op kunnen tellen, tot en met 99.

De uitvoering van de opdracht was wat lastig. Per display zitten er 7 vakjes op. Dit betekend dat er 7 input velden nodig zijn om een display aan te zetten. Op de arduino nano zitten er 3 pin layouts: “PORTB”,”PORTC” en “PORTD”. Per layout hebben ze maar 5 of 6 pins beschikbaar, dit betekend dat om de volle 7 vakjes te gebruiken voor beide led displays je “PORTB” en ”PORTC” gebruikt en “PORTD” gebruikt voor de pins die overgebleven zijn. Zo is het mogelijk om “PORTB” en ”PORTC” direct aan te spreken om de led te veranderen. Bij “PORTD” is er “bit-shifting” gebruikt zodat er geen conflicten ontstaan tussen de 2 componenten.

Voor de rest is er een functie uitgevoerd in een constante while loop. In deze while loop staat een tweetal tellers, die bij elke stap 1 omhoog gaan. Één is voor enkele getallen en één is voor tientallen. Als de tellers samen ‘99’ halen zal hij weer opnieuw starten.

**Bronvermeldingen**<https://github.com/hexagon5un/AVR-Programming> – Examples

<https://e-radionica.com/productdata/LD3361BS.pdf> -- Datasheet

**Uitwerking code**

|  |
| --- |
| //include van librarys voor de code  **#include** <avr/io.h>  **#include** <util/delay.h>  **int** **main** (**void**)  {  // pin setup  DDRB = 0xff;  DDRC = 0xff;  DDRD = 0xff;  //variable setup  **int** i = 0;  **int** j = 0;  **const** **int** iMax = 9;  **const** **int** jMax = 9;  **const** **int** delayTime = 100;  **void** **setCijfer**(**int** cijfer1, **int** cijfer2){    //functie om het eerste cijfer  //en  //het tweede cijfer in de 7 led display te veranderen.    **switch**(cijfer1){ //switch voor led display 1  **case** 0: //voor cijfer 0  PORTB = 0b11001000;//binaire code voor de output port b  PORTD &= ~(1 << 2);//Bitshifting van 1 bit.  **break**;  **case** 1: // voor cijfer 1  PORTB = 0b11111011;  PORTD &= ~(1 << 2);  **break**;  **case** 2: //etc..  PORTB = 0b11000001;  PORTD |= (1 << 2);  **break**;  **case** 3:  PORTB = 0b11010001;  PORTD &= ~(1 << 2);  **break**;  **case** 4:  PORTB = 0b11110010;  PORTD &= ~(1 << 2);  **break**;  **case** 5:  PORTB = 0b11010100;  PORTD &= ~(1 << 2);  **break**;  **case** 6:  PORTB = 0b11000100;  PORTD &= ~(1 << 2);  **break**;  **case** 7:  PORTB = 0b11111001;  PORTD &= ~(1 << 2);  **break**;  **case** 8:  PORTB = 0b11000000;  PORTD &= ~(1 << 2);  **break**;  **case** 9:  PORTB = 0b11010000;  PORTD &= ~(1 << 2);  **break**;  }  **switch**(cijfer2){ //switch voor cijfer 2  **case** 0: //als het cijfer 0 moet zijn  PORTC = 0b11001000;//Port c binaire waarden wordt anders  PORTD &= ~(1 << 3);//Port D bitshiften omdat het maar 1 getal mag veranderen  **break**;  **case** 1: // als het cijfer 1 is etc...  PORTC = 0b11111011;  //PORTD = 0b11110111;  PORTD &= ~(1 << 3);  **break**;  **case** 2:  PORTC = 0b11000001;  PORTD |= (1 << 3);  **break**;  **case** 3:  PORTC = 0b11010001;  //PORTD = 0b11110111;  PORTD &= ~(1 << 3);  **break**;  **case** 4:  PORTC = 0b11110010;  //PORTD = 0b11110111;  PORTD &= ~(1 << 3);  **break**;  **case** 5:  PORTC = 0b11010100;  PORTD &= ~(1 << 3);  **break**;  **case** 6:  PORTC = 0b11000100;  PORTD &= ~(1 << 3);  **break**;  **case** 7:  PORTC = 0b11111001;  PORTD &= ~(1 << 3);  **break**;  **case** 8:  PORTC = 0b11000000;  PORTD &= ~(1 << 3);  **break**;  **case** 9:  PORTC = 0b11010000;  PORTD &= ~(1 << 3);  **break**;  }  }  **while** (1) { //zichzelf herhalen  **\_delay\_ms**(delayTime); //delay volgens variabelen  setCijfer(i,j); //functie wordt uitgevoerd met de telbare variabelen  j++; //als er één actie is uitgevoerd dan telt hij j + 1.  **if**(j>jMax){ //als j zijn maximaal heeft bereikt  i++; //tel i + 1 op  **if**(i>iMax){ //als i zijn max heeft gehaald  i=0; //zet hem weer op 0  }  j=0;  }  }  **return** 0;  } |

|  |  |
| --- | --- |
| **Breadbord schema** | **Afbeelding setup** |
|  | C:\Users\Rick\AppData\Local\Microsoft\Windows\INetCache\Content.Word\b8542217-3d4b-4e49-b1b6-544ea05a21ab.jpg |

# Opdracht 3 – De dobbelsteen

**Youtube Video Link**   
[**https://youtu.be/YkEZafRhJiI**](https://youtu.be/YkEZafRhJiI)

**Samenvatting**Bij de opdracht 3a heb ik de dobbelsteen samengesteld, bestaand uit 9 ledjes die ieder met een weerstand is verbonden van een output naar een ground. Voor de button is er een “pull-down resistor” opstelling gemaakt waar de stroom weg kan, zodat de knop goed werkt en geen random waardes doorstuurt. Bij het indrukken van de knop zal er via de debounce functie (uit het boek) doorgegeven worden of de knop is ingedrukt(ook een middel om foutieve datat tegen te gaan). Als dit zo is zal door de continue while loop de setNumber functie uitgevoerd worden met een randomizer om zo een willekeurige dobbel te doen. Dit willekeurige getal zal door middel van een switch de verschillende outputs op hoog of laag zetten.

**Bronvermeldingen**

<https://github.com/hexagon5un/AVR-Programming> – Examples

**Uitwerking code**

|  |
| --- |
| //includes  **#include** <avr/io.h>  **#include** <util/delay.h>  //debounce functie  uint8\_t **debounce**(**void**){  **if**(bit\_is\_clear(PINC,PC0)){  **\_delay\_us**(1000);  **if**(bit\_is\_clear(PINC,PC0)){  **return** (1);  }  }  **return** (0);  }  //dobbelsteen functie  **void** **setNumber**(**int** nmbr){  **switch**(nmbr){  **case** 1:  PORTB = 0b00000010;  PORTD = 0b00000000;  **break**;  **case** 2:  PORTB = 0b00000000;  PORTD = 0b10100000;  **break**;  **case** 3:  PORTB = 0b00000010;  PORTD = 0b10100000;  **break**;  **case** 4:  PORTB = 0b00010000;  PORTD = 0b10110000;  **break**;  **case** 5:  PORTB = 0b00010010;  PORTD = 0b10110000;  **break**;  **case** 6:  PORTB = 0b00011001;  PORTD = 0b10110000;  **break**;  **case** 7:  PORTB = 0b00011011;  PORTD = 0b10110000;  **break**;  **case** 8:  PORTB = 0b00011101;  PORTD = 0b11110000;  **break**;  **case** 9:  PORTB = 0b01011101;  PORTD = 0b11110000;  **break**;  }  }  **int** **main** (**void**)  {  //init  uint8\_t buttonpressed = 0;  DDRB = 0b00011111; // setup 5 pinnen voor PORT B  DDRD = 0b11110000; // setup 4 pinnen voor PORT D  DDRC = 0x00; // setup input voor PORT C  //event  **while** (1) { // zichzelf herhalen met een while True loop  **if**(debounce()){ // Debounce functie volgens boek (voor indrukken knop)  setNumber(**rand**()% 9 + 1 ); // dobbelsteen functie aanroepen met randomizer meegeven  **\_delay\_ms**(500);  }  }  **return** 0;  } |

|  |  |
| --- | --- |
| **Breadbord schema** | **Afbeelding setup** |
|  |  |

## Opdracht 3b – Potmeter met geluid

**Youtube Video Link**<https://youtu.be/fgjFFqURi1Y>

**Samenvatting**Bij opdracht 3b is het de bedoeling dat er met een potmeter een geluidssignaal gecreëerd wordt, waar de toonhoogte veranderd als je aan de potmeter draait.

Als eerst heb ik de uitwerking van USART geïmplementeerd in mijn code, deze code is afkomstig van het github account van het boek. Ik heb dit gedaan omdat ik met de serial monitor de waarde van de potmeter wilde weten.

Vervolgens toen de waarde van de potmeter in beeld was, ben ik de speaker gaan aansluiten. Ik heb een capacitor aangesloten om de stroompieken van de speaker te kunnen afvangen, zodat er geen schade kan ontstaan aan de Nano.

**Bronvermeldingen**<https://github.com/hexagon5un/AVR-Programming> -- code uit het boek (via github)

**Uitwerking code**

|  |
| --- |
| /\*\*  //usart include  #include "main.h"  \*\*/  **#include** <avr/io.h>  **#include** <util/delay.h>  **static** **inline** **void** **initTimer**(**void**){  /\*  \* InitTimer voor het  \* gebruik van de speaker,  \* Hier, wordt de port gezet tot timer.  \*/  TCCR0A |= ( 1 << WGM01 );  TCCR0A |= ( 1 << COM0A0 );  TCCR0B |= ( 1 << CS00 ) | ( 1 << CS01 );  }  **static** **inline** **void** **initADC0**(**void**) {  /\*  \* Hier wordt de port A0 omgezet tot seriele port  \* Zodat een seriele waarden ontvangen kan worden.  \*/  ADMUX |= (1 << REFS0);  ADCSRA |= (1 << ADPS1) | (1 << ADPS0);  ADCSRA |= (1 << ADEN);  }  **int** **main**(**void**) {  /\* - USART init  //usart serial communication  initUSART();  printString("OK");  \*/  // -------- Inits --------- //  DDRD = (1 << PD6); // PD6 wordt aangezet als Timer voor de box  initADC0(); // Wordt functie: A0 aangezet als serieel ontvang  initTimer(); // Wordt functie: Timer wordt aangezet.  **while** (1) {  ADCSRA |= ( 1 << ADSC );  /\* instantieerd ADC. \*/  loop\_until\_bit\_is\_clear( ADCSRA , ADSC );  /\* wacht tot hij klaar is. \*/  OCR0A = ADC / 4;  //totale waarde van de ADC = maximaal 1024 & de maximale waarde is 255. -> Daarom / 4 (speelt geluid af)  **\_delay\_ms**(10);  /\*  //gebruik van USART.c  printByte(ADC/4);  printString(" -- \r\n");  ` \*/  }  **return** (0);  } |

|  |  |
| --- | --- |
| **Breadbord schema** | **Afbeelding setup** |
|  |  |

# Opdracht 4 – PWM interrupt op RGB

**Youtube Video Link**   
[**https://youtu.be/A2KkxqLYT9Y**](https://youtu.be/A2KkxqLYT9Y)

**Samenvatting**Bij deze opdracht ben ik erg lang bezig geweest, het begrijpen van de Pulse width modulation kostte wat tijd, gelukkig is het wel gelukt. In de code zal de led via de Interrupt schakelen tussen kleur. Dit kan je zien zien in de functie “ISR”. Voor de rest worden er in de main alleen maar variabelen en porten aangemaakt.

**Bronvermeldingen**https://github.com/hexagon5un/AVR-Programming

**Uitwerking code**

|  |
| --- |
| **#include** <avr/io.h>  **#include** <util/delay.h>  **#include** <avr/interrupt.h>  //vars  **unsigned** **char** **const** RGB\_Colors[] = {  0, 0, 0, 255, 0, 0,  127, 0, 255, 255, 0, 255,  127, 255, 0, 0, 255, 0,  0, 255, 127, 0, 255, 255,  0, 127, 255, 0, 0, 255,  127, 0, 255, 255, 0, 255,  255, 0, 127  };  ISR(TIMER2\_OVF\_vect) { // interrupt  **for** (**char** i=0;i<13;i++) // elke seconden een andere kleur  {  OCR1A = RGB\_Colors[i\*3];  OCR1B = RGB\_Colors[i\*3+1];  OCR2A = RGB\_Colors[i\*3+2];  **\_delay\_ms**(1000);  }  }  **void** **main**() {  // Timer 1 A,B  TCCR1A |= (1 << WGM10); /\* Fast PWM mode, 8-bit \*/  TCCR1B |= (1 << WGM12); /\* Fast PWM mode, pt.2 \*/  TCCR1B |= (1 << CS11); /\* PWM Freq = F\_CPU/8/256 \*/  TCCR1A |= (1 << COM1A1); /\* PWM output on OCR1A \*/  TCCR1A |= (1 << COM1B1); /\* PWM output on OCR1B \*/  // Timer 2  TCCR2A |= (1 << WGM20); /\* Fast PWM mode \*/  TCCR2A |= (1 << WGM21); /\* Fast PWM mode, pt.2 \*/  TCCR2B |= (1 << CS21); /\* PWM Freq = F\_CPU/8/256 \*/  TCCR2A |= (1 << COM2A1); /\* PWM output on OCR2A \*/  //Porten (B) enablen  DDRB |= (1<<PB1); //Blauw led aan  DDRB |= (1<<PB2); //Groen led aan  DDRB |= (1<<PB3); //Rood led aan  //uitvoeren van interrupts  TIMSK2 |= (1 << TOIE2); //TimerMask register enabled op TOIE2.  sei(); //zet interrupts aan.  **while** (1) {  }  } |

|  |  |
| --- | --- |
| **Breadbord schema** | **Afbeelding setup** |
|  |  |

## Opdracht 4b – Lamp aan/uit op usart

**Youtube Video link**

<https://youtu.be/DfKogh5QpvU>

**Samenvatting**Deze opdracht is eigenlijk vrij snel gegaan, het lastige van deze opdracht is het opzetten van de USART, daarbij moet je vanuit de teminal data sturen naar de microcontroller, waarbij het apparaat de data uitleest en op basis hiervan de led aan of uit zet. Het uitlezen van de data van de terminal was af en toe wat moeilijk ,omdat het lastig uit te vinden om wat voor datatypen het gaat.

**Bronvermeldingen**

**Uitwerking code**

|  |
| --- |
| // includes  **#include** <avr/io.h>  **#include** <util/delay.h>  //defines  **#define** F\_CPU 16000000UL //kloksnelheid processor  **#define** BAUD 9600 // baudrate  **#define** BAUD\_PRESCALLER (((F\_CPU / (BAUD \* 16UL))) - 1) // prescaler  //functions  **void** **USART\_send**( **unsigned** **char** data){  **while**(!(UCSR0A & (1<<UDRE0)));  UDR0 = data;  }  **unsigned** **char** **USART\_receive**(**void**){  **while**(!(UCSR0A & (1<<RXC0)));  **return** UDR0;  }  **void** **main**() {  //init usart  UBRR0H = (uint8\_t)(BAUD\_PRESCALLER>>8);  UBRR0L = (uint8\_t)(BAUD\_PRESCALLER);  UCSR0B = (1<<RXEN0)|(1<<TXEN0);  UCSR0C = ((1<<UCSZ00)|(1<<UCSZ01));  DDRB = 0xff; //init port B  **char** aan = '1'; //met knop ‘1’ aan, anders uit    **while** (1) {  **unsigned** **char** c = USART\_receive(); //ontvang de char (code wacht hier op)  **if** (c != '\0'){ //als hij niet null is  **if** (c == aan){ //if voor lamp aan  PORTB=0xFF;  USART\_send('o');  } **else** { //else voor lamp uit  PORTB=0x00;  USART\_send('x');  }  }  }  } |

|  |  |
| --- | --- |
| **Breadbord schema** | **Afbeelding setup** |
|  |  |

## Opdracht 4c - RGB aan/uit op usart

**Youtube Video Link**[**https://youtu.be/zPsoYqDKZJI**](https://youtu.be/zPsoYqDKZJI)

**Samenvatting**Door het instellen van mijn interrupt op de recieveUSART. Kon ik door middel van de toetsen de RGB led veranderen. De interrupt event luistert naar berichten die binnenkomen via de terminal. Vervolgens met een switch kijkt hij om wel character het gaat en zet hij een input op aan en uit.

**Bronvermeldingen**<http://www.fourwalledcubicle.com/AVRArticles.php>

<https://arduino.stackexchange.com/questions/43257/atmega328p-hangs-on-usart-rx-interrupt>

**Uitwerking code**

|  |
| --- |
| // includes  **#include** <avr/io.h>  **#include** <util/delay.h>  **#include** <avr/interrupt.h>  //defines  **#define** F\_CPU 16000000UL //kloksnelheid processor  **#define** BAUD 9600 // baudrate  **#define** BAUD\_PRESCALLER (((F\_CPU / (BAUD \* 16UL))) - 1) // prescaler  **void** **main**() {  //interrupt initiate  UCSR0B =(1 << RXCIE0) | (1 << RXEN0);  UCSR0C =(0 << UPM01) |(0 << UPM00) | (0 << UMSEL01) |(0 << UMSEL00) |(0 << USBS0) |(1 << UCSZ01) |(1 << UCSZ00);  sei();  //init usart  UBRR0H = (uint8\_t)(BAUD\_PRESCALLER>>8);  UBRR0L = (uint8\_t)(BAUD\_PRESCALLER);  DDRB = 0xFF; //init port B  PORTB = 0xFF;  **while** (1) {  }  }  ISR(USART\_RX\_vect)  {  **switch** (UDR0){  **case** 'r':  PORTB ^= (1<<3);  **break**;  **case** 'g':  PORTB ^= (1<<4);  **break**;  **case** 'b':  PORTB ^= (1<<2);  **break**;  }  } |

|  |  |
| --- | --- |
| **Breadbord schema** | **Afbeelding setup** |
|  |  |

# Opdracht 5a - ET Phone home

**ds18b20 temp sensor**Deze temperatuur sensor is een sensor die bepaalde standen heeft deze standen moeten doorgestuurd worden met een 64 bit hexadecimaal en hierdoor zal deze sensor van stand kunnen veranderen

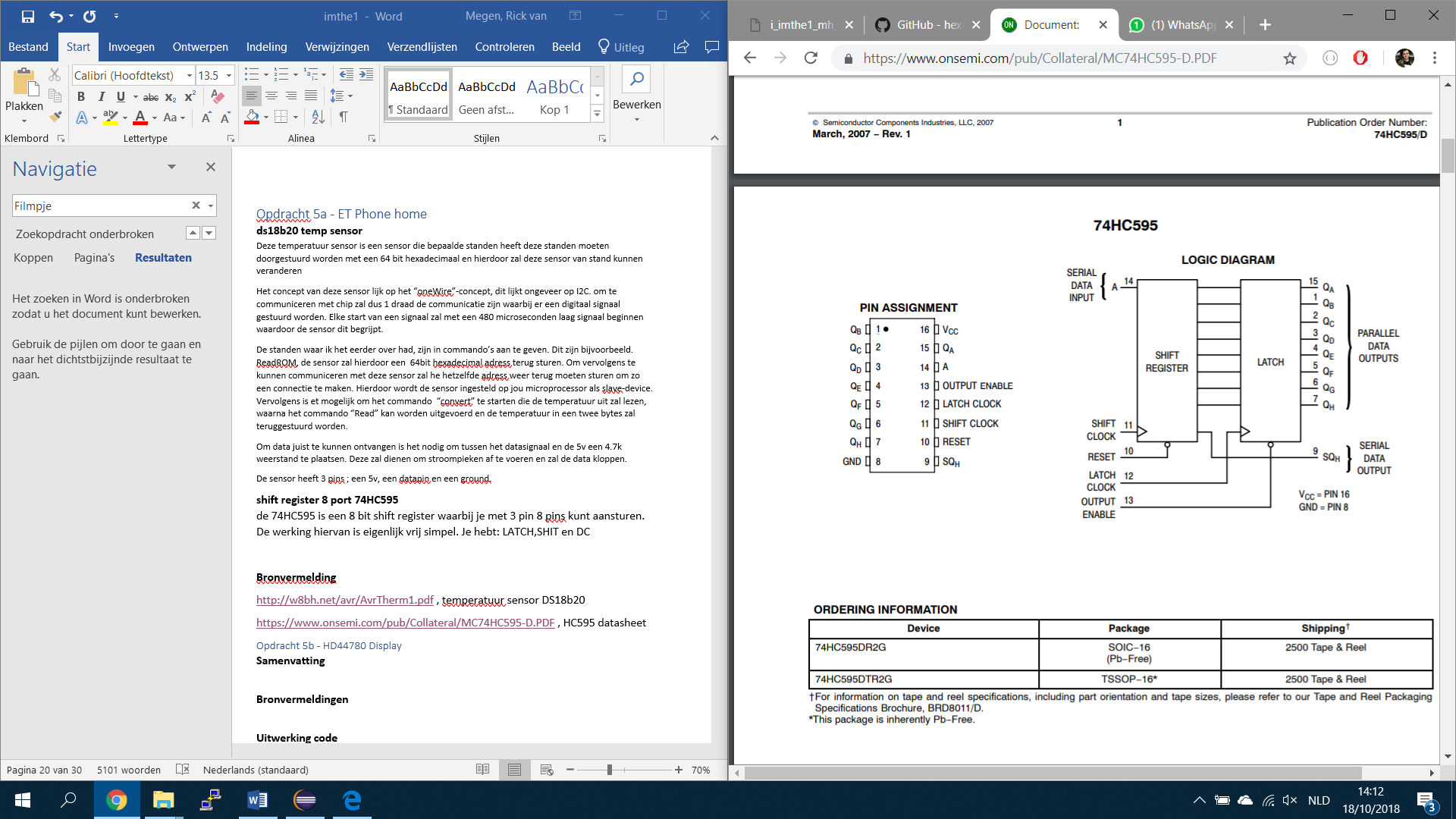
Het concept van deze sensor lijk op het “oneWire”-concept, dit lijkt ongeveer op I2C. om te communiceren met chip zal dus 1 draad de communicatie zijn waarbij er een digitaal signaal gestuurd worden. Elke start van een signaal zal met een 480 microseconden laag signaal beginnen waardoor de sensor dit begrijpt.

De standen waar ik het eerder over had, zijn in commando’s aan te geven. Dit zijn bijvoorbeeld. ReadROM, de sensor zal hierdoor een 64bit hexadecimal adress terug sturen. Om vervolgens te kunnen communiceren met deze sensor zal he hetzelfde adress weer terug moeten sturen om zo een connectie te maken. Hierdoor wordt de sensor ingesteld op jou microprocessor als slave-device. Vervolgens is et mogelijk om het commando “convert” te starten die de temperatuur uit zal lezen, waarna het commando “Read” kan worden uitgevoerd en de temperatuur in een twee bytes zal teruggestuurd worden.

Om data juist te kunnen ontvangen is het nodig om tussen het datasignaal en de 5v een 4.7k weerstand te plaatsen. Deze zal dienen om stroompieken af te voeren en zal de data kloppen.

De sensor heeft 3 pins ; een 5v, een datapin en een ground.

**shift register 8 port 74HC595**de 74HC595 is een 8 bit shift register waarbij je met 3 pin 8 pins kunt aansturen. De werking hiervan is eigenlijk vrij simpel. Je hebt: LATCH,SHIT en DC(A op de img)

De latch clock zal in de shift register werken als een start signaal, en de shift clock zal in de register werken als een push signaal. Tussen de latch en de shift zal je dus bij de DC een 8 bit/1byte signaal moeten sturen waardoor de pin 1 tot en met 8 hoog of laag gezet zal worden. Meer over de werking van deze shift register in de uitleg van de eindopdracht.

**Bronvermelding**

<http://w8bh.net/avr/AvrTherm1.pdf> , temperatuur sensor DS18b20

<https://www.onsemi.com/pub/Collateral/MC74HC595-D.PDF> , HC595 datasheet

## Opdracht 5b - HD44780 Display

**Youtube Video Link**[**https://youtu.be/bINzVkquyLo**](https://youtu.be/bINzVkquyLo)

**Samenvatting**Bij deze library (zie bron) was het eigenlijk nauwelijks nodig om enige code typen anders dan mijn naam , een verwijzing naar de 2e regel van de display en mijn studentnummer van de Hogeschool Leiden.

In de library was het nodig om bepaalde variabelen aan te passen, waarbij er ingesteld moest worden op welke pins de display is aangesloten. Nadat dit gedaan was werkte de code eigenlijk direct

Ik heb ook de potentiometer aangesloten op het scherm om zo de opacity van de tekst op het scherm te kunnen aanpassen.

**Bronvermeldingen**<https://community.atmel.com/projects/hd44780-library> -- guide LCD

**Uitwerking code**

|  |
| --- |
| **#include** "hd44780.h"  **int** **main**(){  lcd\_init(); //start  lcd\_clrscr(); //initiate  lcd\_puts("Rick van Megen");  lcd\_goto(0x40); //ga naar lijn 2  lcd\_puts("S1098012");    **while**(1){  //nothing to do  }  **return** 0;  } |

|  |  |
| --- | --- |
| **Breadbord schema** | **Afbeelding setup** |
|  |  |

# Opdracht eindopdracht

**Youtube Video Link**[**https://youtu.be/IuXEXfGe5Gk**](https://youtu.be/IuXEXfGe5Gk)

**Samenvatting**

Dit concept is een real-time temperatuur meter, hij meet direct de omgeving en zat dit weergeven op de led-matrix. De setting is geheel afhankelijk van de wensen van de gebruiker, ik raad wel aan dit niet onder of in het water te meten.

Er zijn van 3 soorten modules gebruik gemaakt:

**Ds18b20** temperatuur sensor  
(de code van de werking van deze module is onder het kopje ds18b20\_library van de code te vinden)

De temperatuur sensor werd door een zogehete “onewire” module. Dit is een techniek die door 1 digitaal signaal kan communiceren met de temperatuur sensor. Hardware technisch om geen verstoringen te krijgen in dit signaal is het belangrijk een 4.7k resistor aan te sluiten tussen de 5v en de digitale aansluiting.

In de datasheet van deze temperatuur sensor staat dat er een bepaalde 64 bit hexcode gestuurd moet worden naar de sensor om een bepaald soort stand te kunnen aanzetten. Zo is het bijvoorbeeld nodig om een ROM (unieke sleutel) te maken waardoor de microprocessor met de sensor kan communiceren. Vervolgens zal er een start signaal en stop signaal gestuurd worden, waartussen een bericht gestuurd kan worden.

**2x Hc595** – shift register -> led matrix   
(de code van de werking van deze module is onder het kopje hc595\_library van de code te vinden)

Elk shift register is aangesloten met 3 digitale verbindingen. Dit zijn de **ST\_CP** , de **SH\_CP** en de **DS.** De ST\_CP zal het latchen van het shift register zijn. Hij wacht hier op een signaal (dc) om geplaatst te worden in het shift register. Vervolgens als de invoer is aangegeven zal met het SH\_CP de data ingevoerd worden (geshift) als output.

Ik eb 2 van deze shift register aangesloten, waarbij één dient als input en één dient als output. Hiermee zal het mogelijk zijn om de 8x8 led matrix aan te sturen.

**8x8 led matrix**(aangezien deze led matrix geen eigen protocol heeft, maar enkel uit in en outputs bestaat zal deze door de shift-registers hc595 bestuurd worden. Hierdoor is de code ervan te vinden in de hc595 library onder het kope code. )Met de led matrix wordt het mogelijk om output van de temperatuur sensor te plaatsen met getallen . Wat is eigenlijk doe met de led matrix is het volgende; zodat de werking van mijn code duidelijk wordt.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Pulse 1 | | | | | | | |  | Pulse 2 | | | | | | | |  | Pulse 3 | | | | | | | |
|  | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |  | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |  |  | 0 |  |  |  |  |  |  |  |  | 0 |  |  |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |  |  | 0 |  |  |  |  |  |  |  |  | 0 |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  | 0 |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  | 0 |  |  |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |  |  | 0 |  |  |  |  |  |  |  |  | 0 |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |

\*geel = output (-), groen is input (+) – output afbeelding: “5”

De 2 shift registers zullen per keer een pulse ontvangen, zodat er een reeks lampjes gaan branden. Een aantal microsecondes later zullen de shift register een tweede puls ontvangen waardoor een andere set lampjes gaat branden. Gevolgd door de derde pulse.

De code voor de shift registers:

1. Pulse één : input: 10001101, output: 01000000
2. Pulse twee: input: 10101101, output: 00100000
3. Pulse drie: input 10100001, output: 00010000

Op de led matrix zal nu een “5” te zien zijn, dit process is gemaakt voor getal 1 t/m 99.

**Bronvermeldingen**<https://github.com/hexagon5un/AVR-Programming> – USART module

<http://w8bh.net/avr/AvrTherm1.pdf> , temperatuur sensor DS18b20

<https://www.onsemi.com/pub/Collateral/MC74HC595-D.PDF> ,shift register hc595

<https://cdn-shop.adafruit.com/datasheets/BL-M07C881.PDF>, 8x8 led matrix

**Uitwerking code**

|  |
| --- |
| **Main.c :** |
| **#include** <avr/io.h>  **#include** <util/delay.h>  **#include** "main.h"  **#define** F\_CPU 16000000L // run CPU at 16 MHz  **#include** "ds18b20\_library.h"  **#include** "hc595\_library.h"  **volatile** **int** graden = 0; //wordt aangepast door de ds18b20..  **volatile** **int** graden\_tiental = 0; //graden wordt opgesplitst in tientallen en decimalen voor de led matrix  **volatile** **int** graden\_decimal =0; // --  **volatile** **int** slowdown\_teller = 0; //slowdown teller voor de interrupt  **volatile** **int** tientallen\_switch =0; //switch voor het weergeven van de decimalen en tientallen op de led matrix  ISR(TIMER2\_OVF\_vect) { // interrupt  slowdown\_teller++; // teller  **if**(slowdown\_teller == 100){  //als teller 100 is, stuur een lichtpuls naar de ledmatrix  **if**(tientallen\_switch == 0 ){ schrijfGetal(graden\_tiental,0); tientallen\_switch++; } //omstebeurt een stuk van de tientallen en de decimalen  **else** **if**(tientallen\_switch == 1 ){ schrijfGetal(graden\_decimal,1); tientallen\_switch = 0;}  slowdown\_teller =0; //reset teller  }  }  **void** **deelGradenOpInTweeVariabelen**(){  uint8\_t positie = 0;  **int** value = graden; // de value = huidige\_gekregen graden  **while** (value > 0) {  **int** digit = value % 10; // % van 10 (tientallen eerst)  **if**(positie == 0){graden\_tiental = digit;} //als pos == 0 zet tiental  **if**(positie == 1){graden\_decimal = digit;} //als pos == 1 zet decimaal  positie++; //tientallen teller  value /= 10;  }  }  **int** **main**( )  {  //initiation  initUSART(); //usart for debugging  DDRC = 0x00; //PORTC activated for TEMP sensor 18B20  TCCR2B |= (1 << CS00); // prescaler wordt geinstantieerd  TIMSK2 |= (1 << TOIE2);//TimerMask register enabled op TOIE2. (timer 2)  sei(); //zet interrupts aan.  REG\_INPUTInit(); //initiate functie van de shift\_register\_output  REG\_OUTPUTInit(); //inituate functie van de shift\_reguster\_input  **while** ( 1 ){  geefCelcius(&graden); //haalt de graden op uit de ds18b20 module.  deelGradenOpInTweeVariabelen(); //deelt de vragen op in 2 variabelen voor de display  printByte(graden); //USART DEBUG functie (extra module)  printString(". \n"); // -  }  **return** 0; //its not possible to get here !  } |
| **ds18b20\_library.c :** |
| //#include <avr/io.h>  //#include <util/delay.h>  **#include** <avr/io.h>  **#include** <util/delay.h>  /\*\*/  **#define** THERM\_PORT PORTC  **#define** THERM\_DDR DDRC  **#define** THERM\_PIN PINC  **#define** THERM\_IO 0  **#define** ClearBit(x,y) x &= ~\_BV(y) // equivalent to cbi(x,y)  **#define** SetBit(x,y) x |= \_BV(y) // equivalent to sbi(x,y)  **#define** ReadBit(x,y) x & \_BV(y) // call with PINx and bit#  **#define** THERM\_INPUT() ClearBit(THERM\_DDR,THERM\_IO) // make pin an input  **#define** THERM\_OUTPUT() SetBit(THERM\_DDR,THERM\_IO) // make pin an output  **#define** THERM\_LOW() ClearBit(THERM\_PORT,THERM\_IO) // take (output) pin low  **#define** THERM\_HIGH() SetBit(THERM\_PORT,THERM\_IO) // take (output) pin high  **#define** THERM\_READ() ReadBit(THERM\_PIN,THERM\_IO) // get (input) pin value  /\* alle soorten modi die van deze sensor beschikbaar zijn\*/  **#define** THERM\_CONVERTTEMP 0x44  **#define** THERM\_READSCRATCH 0xBE  **#define** THERM\_WRITESCRATCH 0x4E  **#define** THERM\_COPYSCRATCH 0x48  **#define** THERM\_READPOWER 0xB4  **#define** THERM\_SEARCHROM 0xF0  **#define** THERM\_READROM 0x33  **#define** THERM\_MATCHROM 0x55  **#define** THERM\_SKIPROM 0xCC  **#define** THERM\_ALARMSEARCH 0xEC  /\*DE TE GEBRUIKEN ROM IS TE VINDEN ALS JE DE ROMREADERPROGRAM FUNCTIE UITVOER\*/  uint8\_t huidige\_rom[] = {0x28,0x5A,0x63,0x95,0x05,0x00,0x00,0xF5 };  uint8\_t **therm\_Reset**() // reset function van de stand van de sensor  {  uint8\_t i;  THERM\_OUTPUT(); // zet pins als output  THERM\_LOW(); // trek pin laag for 48 0microsecondes  **\_delay\_us**(480);  THERM\_INPUT(); // zet de pins als input  **\_delay\_us**(60); // wacht 60 microsecondes  i = THERM\_READ(); // geef pin waarde  **\_delay\_us**(420); // wachf 480 microsecondes  **return** i;  }  **void** **therm\_WriteBit**(uint8\_t bit)  {  THERM\_OUTPUT(); // zet pin als output  THERM\_LOW(); // zet pin laag voor 1 microseconden  **\_delay\_us**(1);  **if** (bit) THERM\_INPUT(); //schrijf en release  **\_delay\_us**(60);  THERM\_INPUT(); // wait 60uS & release pin  }  uint8\_t **therm\_ReadBit**()  {  uint8\_t bit=0;  THERM\_OUTPUT(); // zet pin als output  THERM\_LOW(); // trek pin laag for 1microsec  **\_delay\_us**(1);  THERM\_INPUT(); // release pin & wacht 14 microsec  **\_delay\_us**(14);  **if** (THERM\_READ()) bit=1; // lees pin value  **\_delay\_us**(45); // wacht voor 60microsecs  **return** bit;  }  **void** **therm\_WriteByte**(uint8\_t data) // functie die 8 keer writeBit schrijft  {  uint8\_t i=8;  **while**(i--){ // for 8 bits:  therm\_WriteBit(data&1);  data >>= 1; // shift all bits right  }  }  uint8\_t **therm\_ReadByte**() // functie die 8 keer een bit uitleest  {  uint8\_t i=8, data=0;  **while**(i--){  data >>= 1; // sbitshift  data |= (therm\_ReadBit()<<7); // krijg volgende bit (LSB first)  }  **return** data;  }  **void** **therm\_MatchRom**(uint8\_t rom[]){  therm\_WriteByte(THERM\_MATCHROM); //gaat naar de match-rom modus  **for** (uint8\_t i=0;i<8;i++)  therm\_WriteByte(rom[i]); //voert het rom 64-bit signaal in  }  **void** **leesRauweTemperatuurData**(uint8\_t id[], uint8\_t \*t0, uint8\_t \*t1){  therm\_Reset(); // skip ROM & start temp conversion  **if** (id) therm\_MatchRom(id);  **else** therm\_WriteByte(THERM\_SKIPROM);  therm\_WriteByte(THERM\_CONVERTTEMP);  **while** (!therm\_ReadBit()); // wait until conversion completed  therm\_Reset(); // read first two bytes from scratchpad  **if** (id) therm\_MatchRom(id);  **else** therm\_WriteByte(THERM\_SKIPROM);  therm\_WriteByte(THERM\_READSCRATCH);  \*t0 = therm\_ReadByte(); // first byte  \*t1 = therm\_ReadByte(); // second byte  }  uint8\_t **RomReaderProgram**(**int** i){ //  therm\_Reset();  therm\_WriteByte(THERM\_READROM); // zet de rom-modus aan, waardoor de sensor zijn 64bit rom code zal returnen  uint8\_t data[8]; //instantieert een array  **for** (uint8\_t i=0; i<8; i++){  data[i] = therm\_ReadByte();  }  **return** data[i]; // geeft gewenste stuk van de array terug  }  uint8\_t **geefCelcius**(**int** \*getal\_heel, **int** \*decimal){  /\*uint8\_t t0,t1;  leesRauweTemperatuurData(huidige\_rom,&t0,&t1);  \*getal\_heel = (t1 & 0x07) << 4; // grab lower 3 bits of t1  \*getal\_heel |= t0 >> 4; // and upper 4 bits of t0  \*/  uint8\_t t0,t1;  leesRauweTemperatuurData(huidige\_rom,&t0,&t1); // get temperature values  \*getal\_heel = (t1 & 0x07) << 4; // grab lower 3 bits of t1  \*getal\_heel |= t0 >> 4; // and upper 4 bits of t0  \*decimal = t0 & 0x0F; // decimals in lower 4 bits of t0  \*decimal \*= 625; // conversion factor for 12-bit resolution  **return** getal\_heel;  } |
| **ds18b20\_library.h** |
| **uint8\_t therm\_Reset();**  **void therm\_WriteBit(uint8\_t bit);**  **uint8\_t therm\_ReadBit();**  **void therm\_WriteByte(uint8\_t data);**  **uint8\_t therm\_ReadByte();**  **void therm\_MatchRom(uint8\_t rom[]);**  **void leesRauweTemperatuurData(uint8\_t id[], uint8\_t \*byte0, uint8\_t \*byte1);**  **uint8\_t RomReaderProgram();**  **uint8\_t geefCelcius();** |
| **hc595\_library.c** |
| **#include** <avr/io.h>  **#include** <util/delay.h>  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Configure Connections  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  //define port D en B voor shift register  **#define** REG\_INPUT\_PORT PORTB  **#define** REG\_OUTPUT\_PORT PORTD  **#define** REG\_INPUT\_DDR DDRB  **#define** REG\_OUTPUT\_DDR DDRD  //define pin setups voor shift register input en output  **#define** REG\_INPUT\_DS\_POS PB0 //Data pin (DS) pin location  **#define** REG\_INPUT\_SH\_CP\_POS PB1 //Shift Clock (SH\_CP) pin location  **#define** REG\_INPUT\_ST\_CP\_POS PB2 //Store Clock (ST\_CP) pin location  **#define** REG\_OUTPUT\_DS\_POS PD2  **#define** REG\_OUTPUT\_SH\_CP\_POS PD3  **#define** REG\_OUTPUT\_ST\_CP\_POS PD4  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Configure Connections \*\*\*ENDS\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  //Initialize REG\_INPUT System  **void** **REG\_INPUTInit**(){  //Make the Data(DS), Shift clock (SH\_CP), Store Clock (ST\_CP) lines output  REG\_INPUT\_DDR|=((1<<REG\_INPUT\_SH\_CP\_POS)|(1<<REG\_INPUT\_ST\_CP\_POS)|(1<<REG\_INPUT\_DS\_POS));  }  **void** **REG\_OUTPUTInit**(){  //Make the Data(DS), Shift clock (SH\_CP), Store Clock (ST\_CP) lines output  REG\_OUTPUT\_DDR|=((1<<REG\_OUTPUT\_SH\_CP\_POS)|(1<<REG\_OUTPUT\_ST\_CP\_POS)|(1<<REG\_OUTPUT\_DS\_POS));  }  //Low level macros to change data (DS)lines  **#define** REG\_INPUTDataHigh() (REG\_INPUT\_PORT|=(1<<REG\_INPUT\_DS\_POS))  **#define** REG\_OUTPUTDataHigh() (REG\_OUTPUT\_PORT|=(1<<REG\_OUTPUT\_DS\_POS))  **#define** REG\_INPUTDataLow() (REG\_INPUT\_PORT&=(~(1<<REG\_INPUT\_DS\_POS)))  **#define** REG\_OUTPUTDataLow() (REG\_OUTPUT\_PORT&=(~(1<<REG\_OUTPUT\_DS\_POS)))  //Sends a clock pulse on SH\_CP line  **void** **REG\_INPUTPulse**(){ // library functie voor het besturen van de shift register  //Pulse the Shift Clock  REG\_INPUT\_PORT|=(1<<REG\_INPUT\_SH\_CP\_POS);//HIGH  REG\_INPUT\_PORT&=(~(1<<REG\_INPUT\_SH\_CP\_POS));//LOW  }  **void** **REG\_OUTPUTPulse**(){ // library functie voor het besturen van de shift register  //Pulse the Shift Clock  REG\_OUTPUT\_PORT|=(1<<REG\_OUTPUT\_SH\_CP\_POS);//HIGH  REG\_OUTPUT\_PORT&=(~(1<<REG\_OUTPUT\_SH\_CP\_POS));//LOW  }  //Sends a clock pulse on ST\_CP line  **void** **REG\_INPUTLatch**(){ // library functie voor het besturen van de shift register  //Pulse the Store Clock  REG\_INPUT\_PORT|=(1<<REG\_INPUT\_ST\_CP\_POS);//HIGH  **\_delay\_loop\_1**(1);  REG\_INPUT\_PORT&=(~(1<<REG\_INPUT\_ST\_CP\_POS));//LOW  **\_delay\_loop\_1**(1);  }  **void** **REG\_OUTPUTLatch**(){  //Pulse the Store Clock  REG\_OUTPUT\_PORT|=(1<<REG\_OUTPUT\_ST\_CP\_POS);//HIGH  **\_delay\_loop\_1**(1);  REG\_OUTPUT\_PORT&=(~(1<<REG\_OUTPUT\_ST\_CP\_POS));//LOW  **\_delay\_loop\_1**(1);  }  **void** **REG\_INPUTWrite**(uint8\_t data){ //library code van de hc595 shift register om te schrijfen naar de led matrix  **for**(uint8\_t i=0;i<8;i++){  **if**(data & 0b10000000){  REG\_INPUTDataHigh();//MSB is 1 so output high  }  **else**{  REG\_INPUTDataLow(); //MSB is 0 so output high  }  REG\_INPUTPulse(); //Pulse the Clock line  data=data<<1; //Now bring next bit at MSB position  }  REG\_INPUTLatch();  }  **void** **REG\_OUTPUTWrite**(uint8\_t data){ //library code van de hc595 shift register om te schrijfen naar de led matrix  **for**(uint8\_t i=0;i<8;i++){  **if**(data & 0b10000000){  REG\_OUTPUTDataHigh();//MSB is 1 so output high  }  **else**{  REG\_OUTPUTDataLow(); //MSB is 0 so output high  }  REG\_OUTPUTPulse(); //Pulse the Clock line  data=data<<1; //Now bring next bit at MSB position  }  REG\_OUTPUTLatch();  }  **volatile** **int** tick = 0; // tick wordt gebruikt voor het eenmalig weergeven van 1 rij per aanroeping v/d functie  **void** **schrijfGetal**(**int** getal,**int** positie){  uint8\_t kolom[9] = { //input positie kolom gebruikt voor de switch  0, //positie 0 = empty  0b00000001, //1  0b00000010, //2  0b00000100, //3  0b00001000, //4  0b00010000, //5  0b00100000, //6  0b01000000, //7  0b10000000 //8  };  **switch**(positie){ //positie van getal  **case** 0://eerste gedeelte  **if**(getal == 0){  **if**(tick == 0){REG\_INPUTWrite( kolom[5] ); REG\_OUTPUTWrite( 0b10000001 );} //rij 1  **if**(tick == 1){REG\_INPUTWrite( kolom[7] ); REG\_OUTPUTWrite( 0b10000001 );} //rij 3  **if**(tick == 2){REG\_INPUTWrite( kolom[6] ); REG\_OUTPUTWrite( 0b10111101 );} //rij 2  }**if**(getal == 1){  **if**(tick == 0 || tick==1 || tick==2){REG\_INPUTWrite( kolom[6] ); REG\_OUTPUTWrite( 0b10000001 );} //rij 1  }**if**(getal == 2){  **if**(tick == 0){REG\_INPUTWrite( kolom[5] ); REG\_OUTPUTWrite( 0b10011011 );} //rij 1  **if**(tick == 1){REG\_INPUTWrite( kolom[6] ); REG\_OUTPUTWrite( 0b10101101 );} //rij 2  **if**(tick == 2){REG\_INPUTWrite( kolom[7] ); REG\_OUTPUTWrite( 0b10110011 );} //rij 3  }**if**(getal == 3){  **if**(tick == 0){REG\_INPUTWrite( kolom[5] ); REG\_OUTPUTWrite( 0b10110101 );} //rij 1  **if**(tick == 1){REG\_INPUTWrite( kolom[6] ); REG\_OUTPUTWrite( 0b10110101 );} //rij 2  **if**(tick == 2){REG\_INPUTWrite( kolom[7] ); REG\_OUTPUTWrite( 0b10000001 ); } //rij 3  }**if**(getal == 4){  **if**(tick == 0){REG\_INPUTWrite( kolom[5] ); REG\_OUTPUTWrite( 0b11110001 );} //rij 1  **if**(tick == 1){REG\_INPUTWrite( kolom[6] ); REG\_OUTPUTWrite( 0b11110111 );} //rij 2  **if**(tick == 2){REG\_INPUTWrite( kolom[7] ); REG\_OUTPUTWrite( 0b10000001 );} //rij 3  } **if**(getal == 5){  **if**(tick == 0){REG\_INPUTWrite( kolom[5] ); REG\_OUTPUTWrite( 0b10110001 ); } //rij 1  **if**(tick == 1){REG\_INPUTWrite( kolom[6] ); REG\_OUTPUTWrite( 0b10110101 ); } //rij 2  **if**(tick == 2){REG\_INPUTWrite( kolom[7] ); REG\_OUTPUTWrite( 0b10000101 ); } //rij 3  }**if**(getal == 6 ){  **if**(tick == 0){REG\_INPUTWrite( kolom[5] ); REG\_OUTPUTWrite( 0b10001101 ); }//rij 1  **if**(tick == 1){REG\_INPUTWrite( kolom[6] ); REG\_OUTPUTWrite( 0b10101101 ); } //rij 2  **if**(tick == 2){REG\_INPUTWrite( kolom[7] ); REG\_OUTPUTWrite( 0b10000001 ); } //rij 3  } **if**(getal == 7 ){  **if**(tick == 0){REG\_INPUTWrite( kolom[5] ); REG\_OUTPUTWrite( 0b11111101 ); } //rij 1  **if**(tick == 1){REG\_INPUTWrite( kolom[6] ); REG\_OUTPUTWrite( 0b11110101 ); } //rij 2  **if**(tick == 2){REG\_INPUTWrite( kolom[7] ); REG\_OUTPUTWrite( 0b10000001 ); } //rij 3  }**if**(getal == 8 ){  **if**(tick == 0){REG\_INPUTWrite( kolom[5] ); REG\_OUTPUTWrite( 0b10000001 ); } //rij 1  **if**(tick == 1){REG\_INPUTWrite( kolom[6] ); REG\_OUTPUTWrite( 0b10110101 ); } //rij 2  **if**(tick == 2){REG\_INPUTWrite( kolom[7] ); REG\_OUTPUTWrite( 0b10000001 ); } //rij 3  }**if**(getal == 9 ){  **if**(tick == 0){REG\_INPUTWrite( kolom[5] ); REG\_OUTPUTWrite( 0b10110001 ); } //rij 1  **if**(tick == 1){REG\_INPUTWrite( kolom[6] ); REG\_OUTPUTWrite( 0b10110101 ); } //rij 2  **if**(tick == 2){REG\_INPUTWrite( kolom[7] ); REG\_OUTPUTWrite( 0b10000001 ); } //rij 3  }  **break**;  **case** 1://tien-tallen  **if**(getal == 0){  **if**(tick == 0){REG\_INPUTWrite( kolom[1] ); REG\_OUTPUTWrite( 0b10000001 ); } //rij 1  **if**(tick == 1){REG\_INPUTWrite( kolom[3] ); REG\_OUTPUTWrite( 0b10000001 ); } //rij 3  **if**(tick == 2){REG\_INPUTWrite( kolom[2] ); REG\_OUTPUTWrite( 0b10111101 ); } //rij 2  }**if**(getal == 1){  **if**(tick == 0 || tick == 1 || tick==2 ){REG\_INPUTWrite( kolom[2] ); REG\_OUTPUTWrite( 0b10000001 ); } //rij 2  }**if**(getal == 2){  **if**(tick == 0){REG\_INPUTWrite( kolom[1] ); REG\_OUTPUTWrite( 0b10011011 ); } //rij 1  **if**(tick == 1){REG\_INPUTWrite( kolom[2] ); REG\_OUTPUTWrite( 0b10101101 ); } //rij 2  **if**(tick == 2){REG\_INPUTWrite( kolom[3] ); REG\_OUTPUTWrite( 0b10110011 ); } //rij 3  }**if**(getal == 3){  **if**(tick == 0){REG\_INPUTWrite( kolom[1] ); REG\_OUTPUTWrite( 0b10110101 ); } //rij 1 en 2  **if**(tick == 1){REG\_INPUTWrite( kolom[2] ); REG\_OUTPUTWrite( 0b10110101 ); } //rij 2  **if**(tick == 2){REG\_INPUTWrite( kolom[3] ); REG\_OUTPUTWrite( 0b10000001 ); } //rij 3  }**if**(getal == 4){  **if**(tick == 0){REG\_INPUTWrite( kolom[1] ); REG\_OUTPUTWrite( 0b11110001 ); } //rij 1  **if**(tick == 1){REG\_INPUTWrite( kolom[2] ); REG\_OUTPUTWrite( 0b11110111 ); } //rij 2  **if**(tick == 2){REG\_INPUTWrite( kolom[3] ); REG\_OUTPUTWrite( 0b10000001 ); } //rij 3  } **if**(getal == 5){  **if**(tick == 0){REG\_INPUTWrite( kolom[1] ); REG\_OUTPUTWrite( 0b10110001 ); } //rij 1  **if**(tick == 1){REG\_INPUTWrite( kolom[2] ); REG\_OUTPUTWrite( 0b10110101 ); } //rij 2  **if**(tick == 2){REG\_INPUTWrite( kolom[3] ); REG\_OUTPUTWrite( 0b10000101 ); } //rij 3  } **if**(getal == 6 ){  **if**(tick == 0){REG\_INPUTWrite( kolom[1] ); REG\_OUTPUTWrite( 0b10001101 ); } //rij 1  **if**(tick == 1){REG\_INPUTWrite( kolom[2] ); REG\_OUTPUTWrite( 0b10101101 ); } //rij 2  **if**(tick == 2){REG\_INPUTWrite( kolom[3] ); REG\_OUTPUTWrite( 0b10000001 ); } //rij 3  } **if**(getal == 7 ){  **if**(tick == 0){REG\_INPUTWrite( kolom[1] ); REG\_OUTPUTWrite( 0b11111101 ); } //rij 1  **if**(tick == 1){REG\_INPUTWrite( kolom[2] ); REG\_OUTPUTWrite( 0b11110101 ); } //rij 2  **if**(tick == 2){REG\_INPUTWrite( kolom[3] ); REG\_OUTPUTWrite( 0b10000001 ); } //rij 3  }**if**(getal == 8 ){  **if**(tick == 0){REG\_INPUTWrite( kolom[1] ); REG\_OUTPUTWrite( 0b10000001 ); } //rij 1  **if**(tick == 1){REG\_INPUTWrite( kolom[2] ); REG\_OUTPUTWrite( 0b10110101 ); } //rij 2  **if**(tick == 2){REG\_INPUTWrite( kolom[3] ); REG\_OUTPUTWrite( 0b10000001 ); } //rij 3  }**if**(getal == 9 ){  **if**(tick == 0){REG\_INPUTWrite( kolom[1] ); REG\_OUTPUTWrite( 0b10110001 ); } //rij 1  **if**(tick == 1){REG\_INPUTWrite( kolom[2] ); REG\_OUTPUTWrite( 0b10110101 ); } //rij 2  **if**(tick == 2){REG\_INPUTWrite( kolom[3] ); REG\_OUTPUTWrite( 0b10000001 ); } //rij 3  }  **break**;  }  tick++;  **if** (tick == 3) tick = 0;  } |
| **hc595\_library.h** |
| **void** **REG\_INPUTInit**();  **void** **REG\_OUTPUTInit**();  **void** **REG\_INPUTPulse**();  **void** **REG\_OUTPUTPulse**();  **void** **REG\_INPUTLatch**();  **void** **REG\_OUTPUTLatch**();  **void** **REG\_INPUTWrite**(uint8\_t data);  **void** **REG\_OUTPUTWrite**(uint8\_t data);  **void** **Wait**();  **void** **schrijfGetal**(**int** getal,**int** positie);  **void** **schrijfOpMatrix**(**int** value); |

|  |
| --- |
| **Breadbord schema** |
|  |
| **Afbeelding setup** |
| C:\Users\Rick\Desktop\imthe1\3591888f-26bf-4acf-82bd-d795e869b7d6.jpgC:\Users\Rick\Desktop\imthe1\3676b1bc-cfc2-4aa7-9503-e408318abf3f.jpg |