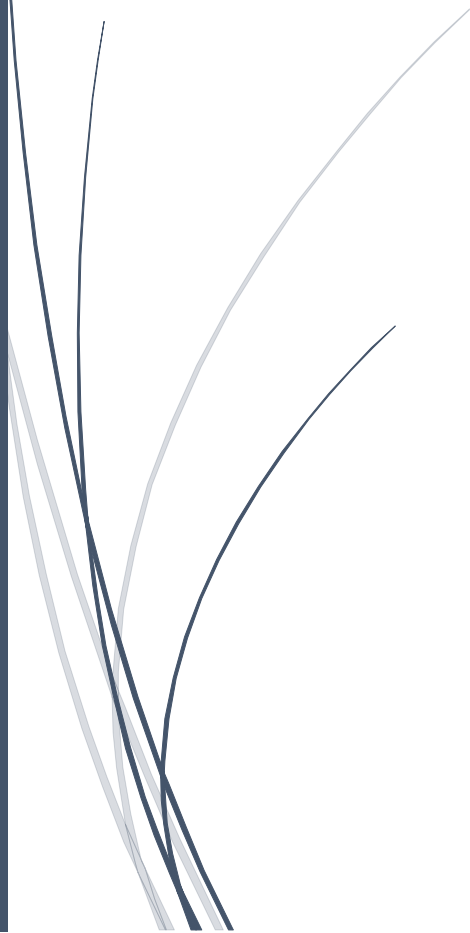


A dark blue vertical bar runs down the left side of the page. A blue arrow points to the right from this bar, containing the date.

27-6-2016

# Microcontrollers

## Groeidocument

Several thin, curved lines in dark blue and light grey originate from the bottom left and sweep upwards and to the right.

Jasper van Megroot  
Jeroen van den Bergh  
Kevin van den Akkerveken

## Voorwoord

Dit is het groeidocument voor het vak microcontrollers. Ik heb in het tweede jaar samengewerkt met Jasper van Megroot en Kevin van den Akkerveken. Zij hebben destijds het vak afgesloten en ik (Jeroen van den Bergh) niet. Een deel van de uitwerkingen in dit document zijn samen gemaakt, vandaar dat hun namen hier ook bijstaan om misverstanden te voorkomen.

### Opdrachten lesweek 1:

Bestudeer de AVR128 documentatie. Raak er vertrouwd mee en zorg dat je vlot de relevantie informatie kan vinden die nodig is. Dit levert tijdswinst op.

Neem hier de tijd voor. Wat is er allemaal, waar staat wat, ...!

Kijk bijvoorbeeld eens wat voor documentatie Atmel (de fabrikant van de ATmega128A) op de website heeft staan.

Kijk eens naar de lijst van application notes over de AVR.

Kijk ook eens wat voor informatie je op de AVR Freaks website ([avrfreaks.net](http://avrfreaks.net)) kunt vinden, bijvoorbeeld op het forum van AVR Freaks.

#### 1.3

Uit de literatuur:

a) Hoe groot is het program memory van de ATmega128?

**128Kbytes.**

b) Wat is het adres van Data direction register van PORTE (DDRE)?

**\$03 (\$23).**

c) Uit hoeveel byte bestaat de instructie 'IN R3, PORTA' ?

**uit 2 bytes.**

d) Hoeveel RS232 poorten zitten er op het BIGAVR6 development board?

**2 stuks**

e) Op welke pin van de microcontroller zit de ingang voor Analog digitaalconverter, channel 1?

**Op pin 60.**

f) Hoe groot is het data geheugen van de microcontroller maximaal?

**64 kB.**

g) Hoeveel I/O-registers zijn er op de ATmega128?

**Er zijn 6 I/O-Registers op de Atmega128.**

h) 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?

**Standaard is dit ingesteld op pull-down dus 0 V.**

## 1.5

Vervolg op het programma Blinky.

Maak een nieuw programma dat beurtelings het LED op PORTD, pin 7 (PORTD.7) en het LED op PORTD, pin 6 (PORTD.6) laat branden (per seconde).

Bekijk eerst het effect met de debugger-functie.

```
#define F_CPU 8000000UL
#define BIT(x) (1<<(x))

#include <avr/io.h>
#include <util/delay.h>

void wait( int ms );

int main(void)
{
    DDRD = 0b11111111;

    PORTD |= BIT(7);
    PORTD &= ~BIT(6);

    for(;;)
    {
        PORTD^=BIT(7);
        PORTD^=BIT(6);
        wait(1000);
    }
    return 1;
}

void wait( int ms )
{
    for (int i=0; i<ms; i++)
    {
        _delay_ms( 1);           // library function (max 30 ms at 8MHz)
    }
}
```

## 1.6

Oefening met de C-omgeving: Toggle PORTD.7

Wijzig Blinky zodat PORTD.7 knippert als drukknop PORTC.0 laag (0) is (ingedrukt) en niet knippert als PORTC.0 hoog (1) is. (niet ingedrukt)

```
#define F_CPU 8000000UL
#define BIT(x) (1<<(x))

#include <avr/io.h>
#include <util/delay.h>

void wait( int ms );

int main(void){
    DDRD = 0b11111111;
    DDRC = 0b11111110;

    PORTD &= ~BIT(7);

    while(1){
        if(PINC & 1){
            PORTC = 0;
            PORTD^= BIT(7);
            wait(200);
        }else{
            PORTD = 0;
        }
    }
    return 1;
}

void wait( int ms )
{
    for (int i=0; i<ms; i++)
    {
        _delay_ms( 1);           // library function (max 30 ms at 8MHz)
    }
}
```

## 1.7

Vervolg op het programma Toggle PORTD.7

Maak een programma dat beurtelings een LED op PA.7 laat knipperen of een LED op PORTD.6 laat knipperen afhankelijk van de waarde op PORTC.0:

PORTC.0 = 0: PORTD.7 knippert

PORTC.0 = 1: PORTD.6 knippert

```
#define F_CPU 8000000UL
#define BIT(x) (1<<(x))

#include <avr/io.h>
#include <util/delay.h>

void wait( int ms );

int main(void){

    DDRD = 0b11111111;
    DDRC = 0b11111110;

    PORTD &= ~BIT(7);
    PORTD &= ~BIT(6);

    while(1){
        if(PINC & 1){
            PORTD &= ~BIT(7);
            PORTD^= BIT(6);
        }else{
            PORTD &= ~BIT(6);
            PORTD^= BIT(7);
        }
        wait(200);
    }
    return 1;
}

void wait( int ms )
{
    for (int i=0; i<ms; i++)
    {
        _delay_ms( 1);           // library function (max 30 ms at 8MHz)
    }
}
```

## 1.8

Oefening met de C-omgeving: RunningLight (zie bijlage 3)

Maak een nieuw programma dat een looplicht maakt.

Laat de LEDs heen en weer lopen, maar wijzig het programma zó dat ze daarbij een LED overslaan, dus: 0 -> 2 -> 4 -> 6 -> 7 -> 5 -> 3 -> 1 -> 0 -> ...

```
#define BIT(x) (1 << (x))
#define F_CPU 8000000UL

#include <avr/io.h>
#include <util/delay.h>

int main( void )
{
    DDRD = 0b11111111;
    while (1)
    {
        for ( int i = 0b00000001; i<0b10000000; i<=<2 )
        {
            PORTD = i;
            wait(1000);
        }
        for ( int i = 0b10000000; i>0b00000001; i>=>2 ) {
            PORTD = i;
            wait(1000);
        }
    }
    return 1;
}

void wait( int ms )
{
    for (int i=0; i<ms; i++)
    {
        _delay_ms( 1 );
    }
}
```

## 1.9

Programmeer een lichtslang die loopt over twee bytes.

De lichtslang start bij PA0, loopt vervolgens naar PA7, dan naar PB7, terug naar PB0, en tenslotte weer naar PA0. Telkens in het rond met een kort tijdsinterval van 400 ms per ronde (25 ms tussen elk stukje).

(Tip: programmeer het looplicht in stukken, dat gaat gemakkelijker dan dat je alles in één keer wilt doen.)

```
#define BIT(x) (1 << (x))
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>

int main( void )
{
    DDRA = 0b11111111;
    DDRB = 0b11111111;
    while (1)
    {
        for ( int i = 0b00000001; i<=0b10000000; i<<=1 )
        {
            PORTB = 0;
            PORTA = i;
            wait(25);
        }
        for ( int i = 0b10000000; i>=0b00000001; i>>=1 ) {
            PORTA = 0;
            PORTB = i;
            wait(25);
        }
    }
    return 1;
}

void wait( int ms )
{
    for (int i=0; i<ms; i++)
    {
        _delay_ms( 1 );
    }
}
```

### 1.10

Maak een programma dat voor elke pen van PORT A de aangesloten LED laat wisselen van aan naar uit of omgekeerd, telkens als de drukknop aangesloten op die pen wordt ingedrukt. Geef in je verslag aan hoe je programma werkt, en welke problemen je hebt moeten oplossen.

Hints:

- Je kunt niet tegelijk de drukknop uitlezen en de LED laten branden, dus hoe los je dat op?
- De LED moet alleen van toestand wisselen als de knop van niet ingedrukt naar wel ingedrukt gaat, dus je moet ook de vorige toestand van de knop kennen en gebruiken.
- Na het omschakelen van output naar input moet je heel even wachten tot de toestand op de PORT pennen stabiel is geworden. Anders lees je de oude output waarde.
- Als je afwisselt tussen LEDs laten branden en knoppen uitlezen, denk dan goed na welk van deze stappen de meeste (wacht)tijd moet krijgen en waarom.

Tips voor een stapsgewijze aanpak:

- Begin eenvoudig: de LEDS van PORT B geven direct de toestand weer van de knoppen van PORT A (dus nog niet afwisselen tussen aan en uit)
- Volgende stap: de LEDS van PORT B tussen aan en uit omschakelen met de knoppen van PORT A (dus PORT A is alleen voor input, PORT B alleen voor output)
- Tenslotte: de LEDS van PORT A omschakelen met de knoppen van PORT A



## Opdrachten lesweek 2:

### Opdracht 2.1

*Als bijlage 2.1 vind je het programma met digitale I/O met interrupts uit de theorieles. Laat het programma draaien en ga na dat alles goed werkt.*

Het werkte correct zoals wij uit de code verwacht hadden. Het lampje op port C7 ging knipperen. Lampje op port C0 en C3 bleven constant aan.

### Opdracht 2.2

*Combineer het programma van 2.1 met opdracht 1.9: Verander het programma zó dat de lichtslang rechtsom loopt ('met de klok mee') bij een laaghoog ('raising edge') interrupt op EX0 en linksom ('tegen de klok in') bij een laag-hoog interrupt op EX1.*

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#define BIT(x) (1 << (x))
unsigned char toggle = 1;
#define F_CPU 8000000UL
int draailinksom( void );
void wait( int ms );
int draairechtsom( void );
int status = 0;
// wait(): busy waiting for 'ms' millisecond
// Used library: util/delay.h
void wait( int ms ) {
    for (int i=0; i<ms; i++)
        _delay_ms( 1 ); // library function (max 30 ms at 8MHz)
}
// Interrupt service routine External Input0
ISR( INT0_vect ) {
    if(status == 0){
        status = 1;
    } else {
        status = 0;
    }
}
// Initialisation external interrupt EX0 and EX1
void exIntrInit( void ) {
    DDRD = DDRB & 0b11111100; // PD1 en PD2 for input
    EICRA |= 0x0F; // EX0, EX1: rising edge
    EIMSK |= 0x03; // turn_on EI0, EI1
    SREG |= 0x80; // turn_on intr all
}
// Main program
int main( void ) {
    DDRA = 0b11111111; // port A output
    DDRB = 0b11111111; // port B output
    exIntrInit(); // initialize EXT_INT0 en EXT_INT1
    while(1){
        if (status == 0){
            draailinksom();
        }
        else {
            draairechtsom();
        }
    }
}
```

```

        return 1;
    }

    int draailinksom( void ) {
        for ( int i = 0b00000001; i<=0b10000000; i<<=1 ) {
            PORTB = 0; // port B alles uit
            PORTA = i; // lampje op port A positie i aan rest uit
            wait(500);
        }
        for ( int i = 0b10000000; i>=0b00000001; i>>=1 ) {
            PORTA = 0; // port A alles uit
            PORTB = i; // lampje op port B positie i aan rest uit
            wait(500);
        }
        return 1;
    }

    int draairechtsom( void ) {
        for ( int i = 0b10000000; i>=0b00000001; i>>=1 ) {
            PORTB = 0; // port B alles uit
            PORTA = i; // lampje op port A positie i aan rest uit
            wait(500);
        }
        for ( int i = 0b00000001; i<=0b10000000; i<<=1 ) {
            PORTA = 0; // port A alles uit
            PORTB = i; // lampje op port B positie i aan rest uit
            wait(500);
        }
        return 1;
    }
}

```

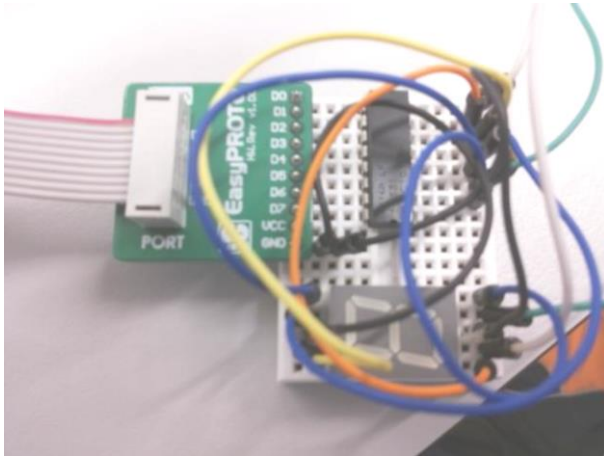


Op deze foto is te zien dat er een lampje brand het laten lopen was niet als afbeelding weer te geven

### Opdracht 2.3:

a) Zoek eerst in het bijgaande programma van bijlage 2.2 uit welke poort gebruikt wordt en welke pinnummer overeenkomt met de aansturing van welk segment. Zoek in de datasheet uit wat de aansluitingen van de display zijn. Gebruik een easy-proto-board. Dit is een adapter van 10 pins IDC naar breadboard. Bouw de bijgaande schakeling op het breadboard. Gebruik hiervoor een weerstandsarray van 330  $\Omega$ . Dit zijn 8 weerstanden in een IC-behuizing.

Dit hebben wij gedaan



b) Laat het programma van bijlage 2.2 draaien en zorg dat alles werkt. Verander het programma dat er naast de decimale cijfers ook de aanvullende hex-cijfers weergegeven worden (dus: 0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f).

bij het initialiseren van numbers was numbers met een N geschreven bij het gebruik met een n dit hebben we aangepast. Daarna werkte hij.

```
#include <avr/io.h>
#include <util/delay.h>
const unsigned char numbers [16] =
{
    // Pgfedcba
    0b00111111, // 0
    0b00000110, // 1
    0b01011011, // 2
    0b01001111, // 3
    0b01100110, // 4
    0b01101101, // 5
    0b01111101, // 6
    0b00000111, // 7
    0b01111111, // 8
    0b01101111, // 9
    0b01011111, // a
    0b01111100, // b
    0b00111001, // c
    0b01011110, // d
    0b01111001, // e
    0b01110001 // f
};
// wait(): busy waiting for 'ms' millisecond
// Used library: util/delay.h
void wait( int ms ){
    for (int i=0; i<ms; i++){
```

```

        _delay_ms( 1 ); // library function (max 30 ms at 8MHz)
    }
}
// main programm
int main (){
    DDRB=0xFF; // poortB output
    while (1){
        for (int i=0; i<=15; i++){ // show the numbers 0..9 sequentially
            PORTB = numbers[i];
            wait (8000);
        }
    }
    return 1;
}

```

c. Maak een functie die een cijfer van aanroep laat verschijnen en schrijf een testprogramma hiervoor.

```

#include <avr/io.h>
#include <util/delay.h>

void aanroep(char text);

const unsigned char numbers [16] =
{
    // Pgfedcba
    0b00111111, // 0
    0b00000110, // 1
    0b01011011, // 2
    0b01001111, // 3
    0b01100110, // 4
    0b01101101, // 5
    0b01111101, // 6
    0b00000111, // 7
    0b01111111, // 8
    0b01101111, // 9
    0b01011111, // a
    0b01111100, // b
    0b00111001, // c
    0b01011110, // d
    0b01111001, // e
    0b01110001 // f
};
// wait(): busy waiting for 'ms' millisecond
// Used library: util/delay.h
void wait( int ms )
{
    for (int i=0; i<ms; i++)
    {
        _delay_ms( 1 ); // library function (max 30 ms at 8MHz)
    }
}
// main programm
int main ()
{
    DDRB=0xFF; // poortB output

```

```

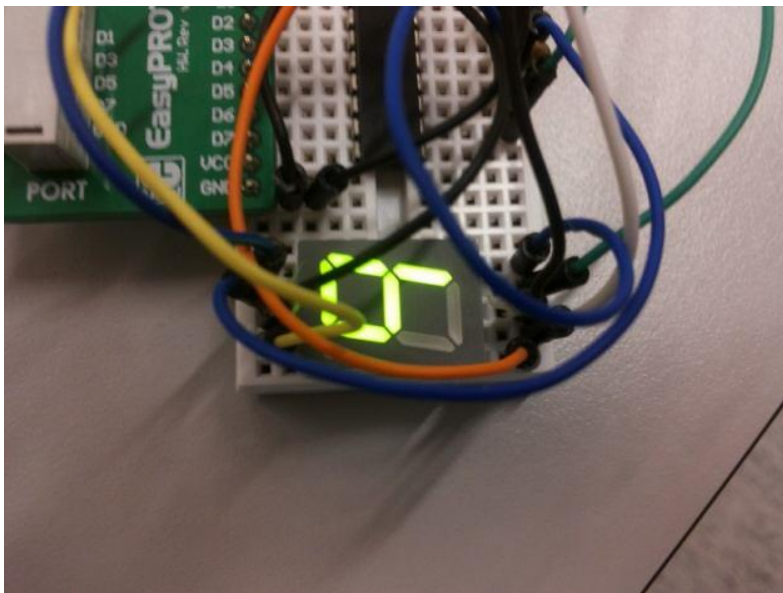
while (1)
{
    aanroep('a');
}
return 1;
}

void aanroep(char text){

    switch(text){
        case '0': PORTB = numbers[0]; break;
        case '1': PORTB = numbers[1]; break;
        case '2': PORTB = numbers[2]; break;
        case '3': PORTB = numbers[3]; break;
        case '4': PORTB = numbers[4]; break;
        case '5': PORTB = numbers[5]; break;
        case '6': PORTB = numbers[6]; break;
        case '7': PORTB = numbers[7]; break;
        case '8': PORTB = numbers[8]; break;
        case '9': PORTB = numbers[9]; break;
        case 'a': PORTB = numbers[10]; break;
        case 'b': PORTB = numbers[11]; break;
        case 'c': PORTB = numbers[12]; break;
        case 'd': PORTB = numbers[13]; break;
        case 'e': PORTB = numbers[14]; break;
        case 'f': PORTB = numbers[15]; break;

    }
}

```



Zoals op bovenstaande foto te zien is worden op het 7 segmenten display alle hexadecimale getallen van 1 t/m f weergegeven.

## Opdracht 2.4:

a. Zoek in het schema van BIGAVR6 op, op welke poort en pinnen de LCD module aangesloten is. Deze informatie heb je nodig om de LCD-module te kunnen programmeren. Controleer ook of de gebruikte module HD44780 compatible is.

De LCD is op poort C en op pin 2,3,4,5,6 en 7 aangesloten, de module is compatible.

b. Maak een project aan en Run het in bijlage 2.4 geleverde LCD programma. Ga na en leg uit in detail hoe de functies `LcdWriteData()` en `LcdCommand()` werken.

LcdWriteData:

1. Hoge nibble wordt op "dat & 0XF0" gezet
2. Data (RS=1), start (EN = 1)
3. Wacht 1 milliseconden.
4. Stopt
5. Lage nibble wordt op "(dat & 0X0F) << 4" gezet
6. Data (RS=1), start (EN = 1)
7. Wacht 1 milliseconden
8. Stopt

LcdCommand:

1. Hoge nibble wordt op "dat & 0XF0" gezet
2. Data (RS=1), start (EN = 1)
3. Wacht 1 milliseconden.
4. Stopt
5. Lage nibble wordt op "(dat & 0X0F) << 4" gezet
6. Data (RS=1), start (EN = 1)
7. Wacht 1 milliseconden
8. Stopt

c. Maak een 'library' `lcd.c` en `lcd.h` waarin de functies uit programma 2.4 m.b.t. het lcd-display worden geplaatst. Neem deze files op in het project en verander het main\_programma van bijlage 2.4 zo dat de lcd-functies worden aangeroepen.

Het werkte met de include en de source file.



Dit is de uitkomst van de test die we gedaan hebben met de tekst: dit is een demo van LCD-display.

## Opdracht 2.5:

a. Maak een functie met een commando dat de tekst laat knippen en voeg deze functie toe aan `lcd.c` en `lcd.h`

Met behulp van de `lcd_knipper()` methode wacht die 750 milliseconden voordat die weer iets anders aanroept waardoor je een knipper effect krijgt:

```
#define F_CPU 8000000UL

#include <avr/io.h>
#include <util/delay.h>

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

char regel1[] = "Het is mooi weer";
char regel2[] = "buiten ";
char regel3[] = " ";

int main (void)
{
    DDRC = 0xFF; // PORT C is output to LCD
    display
    init_lcd(); // initialize LCD

    while(1){
        lcd_knipper();
        lcd_writeLine1(regel3);
        lcd_writeLine2(regel3);
        lcd_knipper();
        lcd_writeLine1(regel1);
        lcd_writeLine2(regel2);
    }

    return 1;
}/

In de lcd.c file:
void lcd_knipper(void){
    wait(750);
}
```

*b. Breid het programma uit zodanig dat je alle tekst leesbaar naar rechts laat scrollen en daarna naar links. Neem bijvoorbeeld eerst 10 posities, later kan je dat uitbreiden.*

Doormiddel van de variabel "verlaging" wordt het telkens verlaagd met -1 zodat het display verschuift naar links.

```
#define F_CPU 8000000UL

#include <avr/io.h>
#include <util/delay.h>

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

char regel1[] = "Komend weekend is het lekker weer in Nederland";

int verlaging = -1;

int main (void)
{
    DDRC = 0xFF;                // PORT C is output to LCD
    display_init_lcd();         // initialize LCD

    lcd_writeLine1(regel1);
    while(1){
        wait(1000);
        lcd_shift(verlaging--);
    }

    return 1;
} // end program
```



c. Maak een programma dat alleen op de onderste regel een tekst van rechts naar links laat lopen, een soort lichtkrant.

```
#define F_CPU 8000000UL

#include <avr/io.h>
#include <util/delay.h>

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

char regel1[] = "Komend weekend is het lekker weer in Nederland";

int verlaging = -1;

int main (void)
{
    DDRC = 0xFF;           // PORT C is output to LCD
    display_init_lcd();    // initialize LCD

    lcd_writeLine2(regel1);

    int i
    while(1){
        wait(1000);
        lcd_shift(verlaging--);
    }

    return 1;
} // end program
```

d. Breid het programma uit met een tekst van meer dan 20 karakters per regel.

```
#define F_CPU 8000000UL

#include <avr/io.h>
#include <util/delay.h>

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

char regel1[] = "Komend weekend is het lekker weer in Nederland";

int verlaging = -1;

int main (void)
{
    DDRC = 0xFF;           // PORT C is output to LCD display
    display_init_lcd();    // initialize LCD

    lcd_writeLine2(regel1);

    int i
    while(1){
        wait(1000);
        lcd_shift(verlaging--);
    }

    return 1;
} // end program
```

## Opdracht 2.6:

*b. Maak een zelf gedefinieerde karakter (bijvoorbeeld het graden rondje) en laat deze op een vaste locatie weergeven. Bijvoorbeeld voorlaatste positie op de bovenste regel de °. Op de laatste positie kan dan 'C' weergegeven worden, in totaal: °C*

```
#define BIT(x) (1 << (x))
char regel1[] = "dit is een demo ";
char regel2[] = "van LCD-display ";
static unsigned char charset[] = //let op: maximaal 8 karakters
{
    0x02,0x05,0x05,0x02,0x00,0x00,0x00,0x00, // char 0: graden symbool
    0x04,0x0E,0x1F,0x04,0x04,0x00,0x00,0x00, // char 1: pijl omhoog
    0x00,0x0a,0x0a,0x00,0x11,0x0e,0x06,0x00 // char 2: smile
};
void wait( int ms )
{
    for (int i=0; i<ms; i++)
    {
        _delay_ms( 1 );
    }
}
void lcd_command ( unsigned char command )
{
    PORTC = command & 0xF0;
    PORTC = PORTC | 0x08;
    wait(1);
    PORTC = 0x00;
    PORTC = (command & 0x0F) << 4;
    PORTC = PORTC | 0x08;
    wait(1);
    PORTC = 0x00;
}
void lcd_writeChar( unsigned char dat )
{
    PORTC = dat & 0xF0;
    PORTC = PORTC | 0x0C;
    _delay_ms(1);
    PORTC = 0x04;
    PORTC = (dat & 0x0F) << 4;
    PORTC = PORTC | 0x0C;
    _delay_ms(1);
    PORTC = 0x00;
}
void init_lcd(void)
{
    lcd_command( 0x02 );
    lcd_command( 0x28 );
    lcd_command( 0x0C );
    lcd_command( 0x06 );
    lcd_command( 0x80 );
}
void lcd_writeln1 ( char text1[] )
```

```

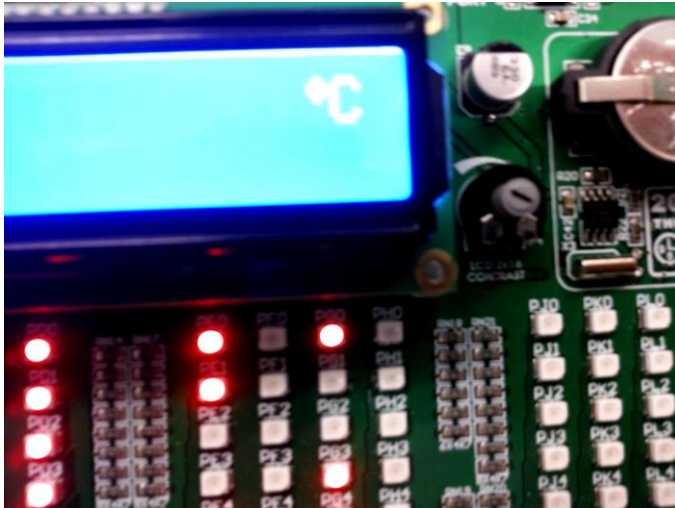
{
    lcd_command(0x80);
    for ( int i=0; i<16; i++ )
    {
        lcd_writeChar( text1[i] );
    }
}
//
// write second line
void lcd_writeLine2 ( char text2[] )
{
    lcd_command(0xC0);
    for ( int i=0; i<16; i++ )
    {
        lcd_writeChar( text2[i] );
    }
}
void lcd_shift(int displacement)
{
    int number=displacement<0?-displacement:displacement;
    for (int i=0; i<number; i++)
    {
        if (displacement <0)
            lcd_command(0x18);
        else
            lcd_command(0x1C);
    }
}
void lcd_writeToPosL1(unsigned char dat, int pos)
{
    lcd_command(0x80);
    int number=pos<0?-pos:pos;
    for (int i=0; i<number; i++)
        lcd_command(0x14);
    lcd_writeChar(dat);
}
void lcd_writeToPosL2(unsigned char dat, int pos)
{
    lcd_command(0xC0);
    int number=pos<0?-pos:pos;
    for (int i=0; i<number; i++)
        lcd_command(0x14);
    lcd_writeChar(dat);
}
void lcd_clear()
{
    lcd_command(0x01);
}
void lcd_fillCGrom (unsigned char* charmap)
{
    lcd_command(0x40);
    for (int ch=0; ch< sizeof(charmap); ch++)
    {
        lcd_writeChar( charmap[ch] );
    }
}
int main (void)

```

```

{
    DDRC = 0xFF;
    init_lcd();
    lcd_clear();
    lcd_fillCGrom( charset );
    lcd_writeToPosL1( 0, 14);
    lcd_writeToPosL1('C', 15);
    return 1;
}

```



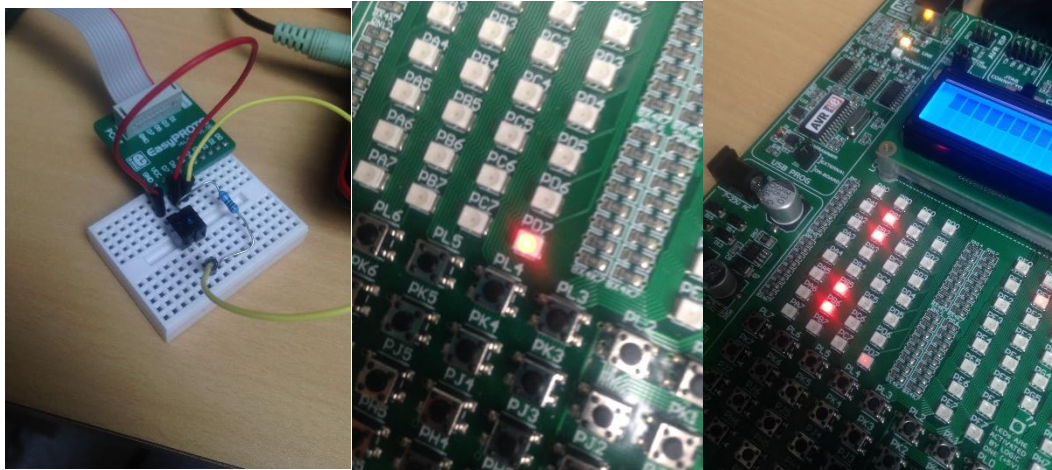
Zoals in de code omschreven word het graden symbool op positie 15 weergegeven en de hoofdletter C op positie 16.

## Opdrachten lesweek 3:

### Opdracht 3.1 – Counter op T2:

a) Het programma van de theorieles 3.1 telt pulsen met counter2 op Poort D.7 (=T2) en laat het resultaat van TCNT2 op poortB zien. Maak een project met het programma 3.1 en run het programma. Eerst met de simulator, daarna in het BIGAVR board.

Zodra je met een infrarood afstandsbediening op de counter ging mikken ging het lampje van PD7 knipperen en de lampjes op port B:



b) Pas het programma aan zodat je het aantal getelde pulsen op het lcd display laat zien. Het deel van het hoofdprogramma staat ook in bijlage 1. Voeg in het project de code file voor de lcd-functies toe ('lcd.c' en 'lcd.h').

Deze methode werkt samen met de lcd.c en lcd.h van vorige week. Als resultaat werden de pulsen op het lcd weergegeven.

```
int main( void )
{
    TCCR2 = 0x07;
    DDRD &= ~BIT(7);
    DDRB = 0xFF;
    DDRC = 0xFF;
    lcd_init();
    int idx = 0;
    char puls[16];
    while (1)
    {
        PORTB = TCNT2;
        idx++;
        sprintf(puls, "%i ", idx);
        lcd_writeline1(str);
        wait(10);
    }
    return 0;
}
```

c) De pulsen die geteld worden met counter2 op Poort D.7 (=T2) worden gegenereerd door een 'reflective optic sensor'. Deze sensor is vergelijkbaar met de sensor die bij de Boebot is gebruikt om een zwarte lijn te volgen. Het bestaat uit een infrarood LED en een infrarood sensor (een fototransistor). Wanneer het licht van de LED wordt weerkaatst door een voorwerp geleidt de fototransistor, ander niet. Controleer de werking (met een wit papiertje op enige afstand van de sensor).

Voor deze opdracht werd er gebruik gemaakt van de code op BlackBoard. Als resultaat werd er het aantal pulsen weergegeven op het scherm.

```
int main( void )
{
    char count;
    char countOnDisplay[3];
    TCCR2 = 0x07;
    DDRD &= ~BIT(7);
    DDRB = 0xFF;
    DDRC = 0xFF;
    lcd_init();
    while (1)
    {
        count = TCNT2;
        PORTB = count;
        sprintf(countOnDisplay, "Counter: %d ", count);
        lcd_writeline1(countOnDisplay, 10);
        wait(10);
    }
    return 0;
}
```

d) De afstandbediening van de radio (of de boebot van vorig jaar) zendt ook infra-rood-pulsen uit. Druk eens op een cijferknop en noteer het aantal pulsen dat ontvangen is.

Het aantal pulsen wat werd ontvangen was 11.

### Opdracht 3.2 – T2 met preset en overflow mode:

a) Pas het programma aan zodat elke 25e puls een overflow wordt gegenereerd (een 'vijf-entwintig-teller').

De TimerStart veranderd naar -25.

```
#define F_CPU 8000000UL

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

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

#define TimerStart -25

unsigned int tenthValue = 0;

// wait(): busy waiting for 'ms' millisecond
// Used library: util/delay.h
void wait( int ms )
{
    for (int i=0; i<ms; i++)
    {
```

```

        _delay_ms( 1 );           // library function (max 30 ms at 8MHz)
    }
}

// Interrupt routine timer2 overflow
//
ISR( TIMER2_OVF_vect )
{
    TCNT2 = TimerStart;           // Preset value
    tenthValue++;                 // Increment counter
}

// Initialize timer 2: counting, preset, interrupt on overflow
void timer2Init( void )
{
    TCNT2 = TimerStart;           // Preset value of counter 2
    TIMSK |= BIT(6);             // T2 overflow interrupt enable
    SREG |= BIT(7);              // turn_on intr all
    TCCR2 = 0x07;                // Initialize T2: ext.counting, rising edge
}

// Main program: Counting on T2
int main( void )
{
    DDRD &= ~BIT(7);             // set PORTD.7 for input
    DDRB = 0xFF;                 // set PORTB for output (shows countregister)
    DDRC = 0xFF;                 // set PORTC for output (shows tenthvalue)
    timer2Init();

    while (1)
    {
        PORTB = TCNT2;           // show value counter 2
        PORTC = tenthValue;      // show value tenth counter
        wait(10);                // every 10 ms
    }
}

```

b) Pas het programma aan zodat zowel de waarde van de counter als de waarde van de vijftientig-teller (ook) op het lcd-display getoond worden.

De lcd.c en lcd.h files zijn weer toegevoegd aan de onderstaande code.

```

int main( void )
{
    DDRD &= ~BIT(7); // set PORTD.7 for input
    DDRB = 0xFF; // set PORTB for output (shows countregister)
    DDRC = 0xFF; // set PORTC for output (shows tenthvalue)
    lcd_init();
    timer2Init();
    char timer[15];
    while (1)
    {
        PORTB = TCNT2; // show value counter 2
        PORTC = tenthValue; // show value tenth counter
        int i = -TimerStart-(255-TCNT2);
        sprintf(timer,"%d - %d " ,i,tenthValue);
        lcd_writeline1(timer);
        wait(10); // every 10 ms
    }
}

```

### Opdracht 3.3 – T2 in compare-mode

De tiende puls is ook de detecteren met de compare-functie van de timer. Dat gebeurt in het programma 3.3. Het effect van dit programma is gelijk aan dat van 3.2, maar gebaseerd op een ander principe: hier door de compare-functie en in het programma 3.2 door een preset en een overflow.

- Maak een project met het programma 3.3 en run het programma. Gebruik eerst de simulator, daarna het BIGAVR board.
- Laat ook hier de waarden op het lcd-display zien.
- Maak er ook een 25-teller van.

Lcd.c en lcd.h zijn weer toegevoegd.

Alle 3 opgaves samen:

```
int main( void )
{
    DDRD &= ~BIT(7); // set PORTD.7 for input
    DDRA = 0xFF; // set PORTA for output (shows countregister)
    DDRB = 0xFF; // set PORTB for output (shows tenth value)
    DDRC=0xFF; // PORT C is output
    lcd_init(); // initialize LCD
    timer2Init(); // Initialize timer 2
    char puls1[17];
    char puls2[17];
    while (1)
    {
        PORTA = TCNT2; // show value counter 2
        PORTB = tenthValue; // show value tenth counter
        sprintf(puls1, "puls: %i ",TCNT2);
        sprintf(puls2, "tiende puls: %i ",tenthValue);
        lcd_writeline1(line1);
        lcd_writeline2(line2);
        wait(10);
    }
}
```

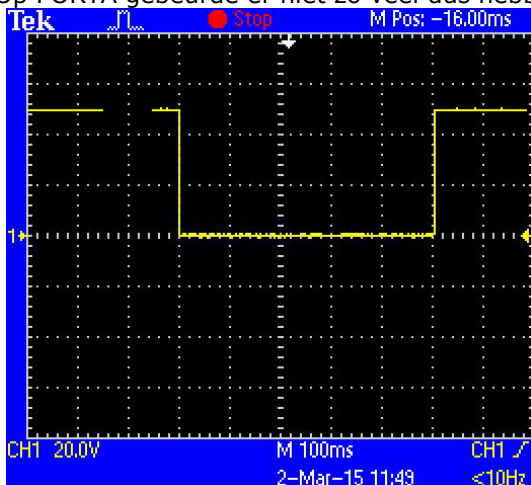
Als je 10 keer drukte werd de waarde met 1 verhoogd.

### Opdracht 3.4 – T2 als timer

We hebben in de voorgaande programma's steeds externe pulsen geteld. Nu gaan we gebruik maken van de interne clockpulsen en kunnen we tijd instellen. Dat is een betere oplossing dan de functie wait() die we voorheen gebruikten: veel preciezer en geen 'busy form of waiting' meer.

- In het programma van bijlage 3.4 wordt elke 250 ms een bit van PORTC getoggeld. Laat het programma draaien en controleer de tijd door met een oscilloscoop naar de overeenkomsten pen van PORTA te kijken. Meet de ingestelde tijd en neem dit op in je verslag.

Op PORTA gebeurde er niet zo veel dus hebben we maar eentje gemaakt van portC.7





b) Pas het programma aan, zodat nu elke 1000 ms (1 s) het bit van PORTV wordt getoggeld.

```
unsigned char count = 0;

void wait( int ms )
{
    for (int i=0; i<ms; i++)
    {
        _delay_ms( 1 ); // library function (max 30 ms at 8MHz)
    }
}

ISR( TIMER2_COMP_vect )
{
    count++; // Increment ms counter
    if ( count == 250 )
    {
        PORTC ^= BIT(0); // Toggle bit 0 van PORTC
        count = 0; // Reset ms_count value
    }
}

// Initialize timer 2: counting, preset, interrupt on overflow
void timer2Init( void )
{
    OCR2 = 250; // Compare value of counter 2
    TIMSK |= BIT(7); // T2 compare match interrupt enable
    SREG |= BIT(7); // turn_on intr all
    TCCR2 = 0b00001011; // Initialize T2: timer, prescaler=32,
}

// Main program: Counting on T2
int main( void )
{
    DDRC = 0xFF; // set PORTC for output (shows toggle bit)
    timer2Init();
    while (1)
    {
        wait(1000); // every 250 ms (busy waiting)
        PORTC ^= BIT(7);
    }
}
```

De wait methode aanpassen naar 1000 en je krijgt het effect.

### Opdracht 3.5 – T1 als timer - klok:

Het programma 3.5 is een soort real time clock met een digitaal display.

a) Maak een project met het programma 3.5 en run het programma.

Werkt

b) Voor een testfase is het handiger als het programma sneller verloopt. Pas het programma aan zodat het 60 x sneller loopt (elke minuut komt dan overeen met 1 seconde).

-

c) Controleer de ingestelde tijd voor een compare\_interrupt (hoe?) neem het resultaat op in je verslag.

```
char tijd[15];
sprintf(tijd,"Tijd: %d " ,OCR1A);
lcd_writeline1(tijd);
```

d) Geef de seconden – minuten – uren op het lcd-display weer

De code is een klein beetje aangepast:

```
char tijd[15];
sprintf(tijd,"Tijd: %d-%02d-%02d " ,hours,minutes,seconds);
lcd_writeline1(tijd);
```

e) Pas het programma aan, zodat je ook de dagen kunt weergeven.

De ISR is veranderd om de dagen op te laten lopen en via de main methode wordt het afgebeeld op het scherm.

```
ISR( TIMER1_COMPA_vect )
{
    seconds++; // Increment s counter
    PORTC ^= BIT(0); // Toggle bit 0 van PORTC
    if ( seconds == 60 ) // Every 1 minute:
    { //
        minutes++; // Increment minutes counter
        seconds = 0; // Reset s-counter
        if ( minutes == 60 ) // Every hour:
        { //
            minutes = 0; // Reset min-counter
            hours++; // Increment hours
            if ( hours == 24 ){ // Every day:
                hours = 0; // reset hours
                days++;}
        }
    }
}

int main( void )
{
    DDRC = 0xFF; // set PORTC for output (shows s, min, h)
    timer1Init();
    lcd_init();
    char tijd[15] = "";
    char dag[15] = "";
    while (1)
    {
        sprintf(tijd,"Tijd: %02d-%02d-%02d " ,hours,minutes,seconds);
        lcd_writeline1(tijd);
        wait(10);
        sprintf(dag, "Dag: %02d", days);
        lcd_writeline2(dag);
        wait(10);
    }
}
```

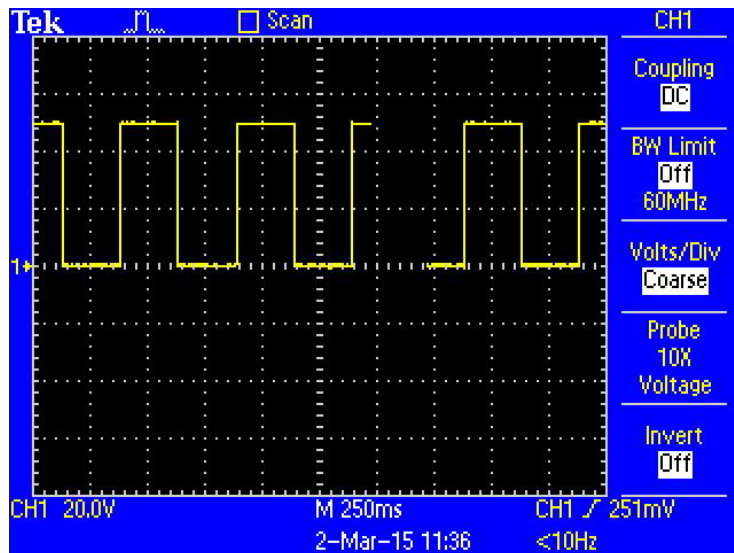
f) Pas het programma aan zodat de waarden op het lcd-display hexadecimaal worden weergegeven  
De decimaal in hexadecimaal veranderd.

```
sprintf(tijd,"Tijd: %02x-%02x-%02x " ,hours,minutes,seconds);
lcd_writeline1(tijd);
wait(10);
sprintf(dag, "Dag: %02x", days);
lcd_writeline2(dag);
```

## Opdrachten lesweek 4:

### Opdracht 4.1

- a) Meet met een oscilloscoop het pulsbreedte en de frequentie. Noteer deze, en maak een foto of print van het beeld voor jouw verslag.



de gemete pulsbreedte is 250ms te zien onder de middelste Y-as (M 250ms). De frequentie kan dan berekend worden via de formule  $f = 1/T$  waarin T de tijd in seconden is. Invullen:  $f = 1/0,25$  levert op:  $f = 4$  Hz oftewel, de puls wisselt 4x per seconde.

- b) Pas het programma aan, zodat je de grote C (C4) laat horen.  
Zoek eerst op: Welke frequentie heeft deze grote C, en welke periodetijd?

Grote C = 261,6 Hz  
 $T = 1/261,6 = 0,003822629969419$   
 $0,003822629969419 * 1.000.000 = 3822$

Op het moment dat het programma gestart word laat de buzzer een grote C horen

Code:

```

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define BIT(x)      (1 << (x))
#define INTERVAL    3822

unsigned int sCount=0, minutes=0, hours=0;

// wait(): busy waiting for 'ms' millisecond
// Used library: util/delay.h
void wait( int ms )
{
    for (int i=0; i<ms; i++)
    {
        _delay_ms( 1 );           // library function (max 30 ms at 8MHz)
    }
}

// Initialize timer 1: fast PWM at pin PORTB.6 (hundredth ms)
void timer1Init( void )
{
    ICR1 = INTERVAL;              // TOP value for counting = INTERVAL*us
    OCR1A = INTERVAL/2;           // compare value in between
    TCCR1A = 0b100000010;         // timer, compare output at OC1A=PB5
    TCCR1B = 0b00011010;          // fast PWM, TOP = ICR1, prescaler=8 (1MHz), RUN
}

// Main program: Counting on T1
int main( void )
{
    DDRB = 0xFF;                  // set PORTB for compare output
    DDRA = 0xFF;                  // set PORTA for output in main program
    timer1Init();                 // it is running now!!

    while (1)
    {
        // do something else
        wait(100);                // every 100 ms (busy waiting)
        PORTA ^= BIT(7);          // toggle bit 7 PORTA
    }
}

```

## Opdracht 4.2

- a) Sluit de drie kleuren LED's aan op de OCR-uitgangen van timer1;  
OCR1A = red; OCR1B = green; OCR1C = blauw.

Zoek uit welke poortpinnen dat zijn.

Voor elke LEDkleur is een voorschakelweerstand nodig: 180  $\Omega$  voor rood (mag ook wel 150  $\Omega$  zijn) en 100  $\Omega$  voor groen en blauw.

Gebruik het 10 pins -> breadboard adapter en maak de schakeling op het breadboardje.

Laat het programma van bijlage 4.2 draaien.

(BB – dag4\_2.c).

- b) Het programma van bijlage 4.2 geeft alleen de rode kleur weer.

Maak een programma dat elke kleur van de LED's afzonderlijk kan aansturen met PWM en elke afzonderlijke LED kan laten 'faden' (de lichtintensiteit geleidelijk laten toenemen van tot maximaal en ook weer geleidelijk laten afnemen).

```
void wait( int ms ) {
    for (int tms=0; tms<ms; tms++) {
        _delay_ms( 1 );
    }
}
// Initialize timer 1: fast PWM at pin PORTB.6 (hundredth ms)
void timer1Init( void ) {
    OCR1A = 0; // RED, default, off
    OCR1B = 0; // GREEN, default, off
    OCR1C = 0; // BLUE, default, off
    TCCR1A = 0b10101001; // compare output OC1A,OC1B,OC1C
    TCCR1B = 0b00001011; // fast PWM 8 bit, prescaler=64, RUN
}
void setRed( unsigned char red ) {
    OCR1A = red;
}
void setGreen( unsigned char green ) {
    OCR1B = green;
}
void setBlue( unsigned char blue ) {
    OCR1C = blue;
}
int main( void ) // Main program: Counting on T1
{
    DDRB = 0xFF; // set PORTB for compare output
    timer1Init();
    wait(100);
    loop();
}
void loop()
{
    while (1)
    {
        int tmp = 1;
        setRed( 0); setGreen(0); setBlue(0); // set rgb off
        // fade Red in
        for (int red = 0; red<=255; red+=tmp)
        {
            setRed( red );
        }
    }
}
```

```

        tmp += 2;
        wait(100);
    }
    // fade Red out
    for (int red = 255; red>=0; red-=tmp)
    {
        setRed( red );
        tmp -= 2;
        wait(100);
    }
    setRed( 0 ); // red off
    tmp = 1;
    wait(100);
    // fade Green in
    for (int gr = 0; gr<=255; gr+=tmp)
    {
        setGreen( gr );
        tmp += 2;
        wait(100);
    }
    // fade Green out
    for (int gr = 255; gr>=0; gr-=tmp)
    {
        setGreen( gr );
        tmp -= 2;
        wait(100);
    }
    setGreen(0); // green off
    tmp = 1;
    wait(100);
    // fade blue in
    for (int blue = 0; blue<=255; blue+=tmp)
    {
        setBlue(blue);
        tmp += 2;
        wait(100);
    }
    // fade blue out
    for (int blue = 255; blue >=0; blue-=tmp)
    {
        setBlue(blue);
        tmp+=2;
        wait(100);
    }
    setBlue(0); // blue off
    tmp = 1;
    wait(100);
}
}

```

c) *Maak een programma dat de drie kleuren vloeiend in elkaar laat overlopen over het hele mogelijk bereik (miljoenen kleurmogelijkheden).*

```

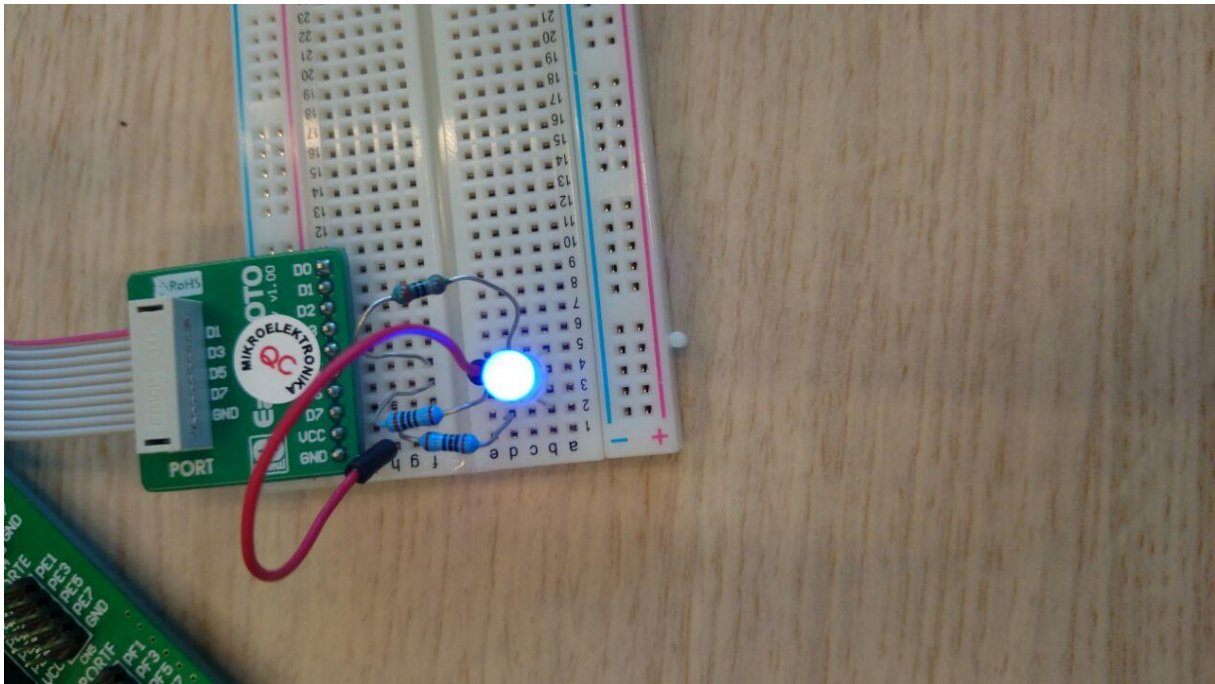
void wait( int ms ) {
    for (int tms=0; tms<ms; tms++) {
        _delay_ms( 1 );
    }
}

```

```

}
// Initialize timer 1: fast PWM at pin PORTB.6 (hundredth ms)
void timer1Init( void ) {
    OCR1A = 0; // RED, default, off
    OCR1B = 0; // GREEN, default, off
    OCR1C = 0; // BLUE, default, off
    TCCR1A = 0b10101001; // compare output OC1A,OC1B,OC1C
    TCCR1B = 0b00001011; // fast PWM 8 bit, prescaler=64, RUN
}
void setRed( unsigned char red ) {
    OCR1A = red;
}
void setGreen( unsigned char green ) {
    OCR1B = green;
}
void setBlue( unsigned char blue ) {
    OCR1C = blue;
}
int main( void ) // Main program: Counting on T1
{
    DDRB = 0xFF; // set PORTB for compare output
    timer1Init();
    wait(100);
    loop();
}
void loop()
{
    setRed(0); setGreen(0); setBlue(0);
    unsigned int rgbColour[3];
    while(1) {
        rgbColour[0] = 255; //red
        rgbColour[1] = 0; //green
        rgbColour[2] = 0; //blue
        for (int decColour = 0; decColour < 3; decColour += 1) {
            int incColour = decColour == 2 ? 0 : decColour + 1;
            for(int i = 0; i < 255; i += 1) {
                rgbColour[decColour] -= 1;
                rgbColour[incColour] += 1;
                setRed(rgbColour[0]);
                setGreen(rgbColour[1]);
                setBlue(rgbColour[2]);
                wait(5);
            }
        }
    }
}

```



Op deze afbeelding kan je zien dat de led blauw van kleur is. Deze verandert voortdurend van kleur.

### Opdracht 4.3

*Het programma 'dag4\_3.c' (zie Blackboard en bijlage 4.3) laat de 10 bits AD-waarde van kanaal 1 op de LED's van Poort A en B zien. Run het programma. Verander het programma zodanig dat kanaal 3 gelezen wordt en verder alleen de hoogste 8 bits worden getoond (op poort A). Door middel van een jumper kan je kiezen welk kanaal met de potentiometer verbonden is.*

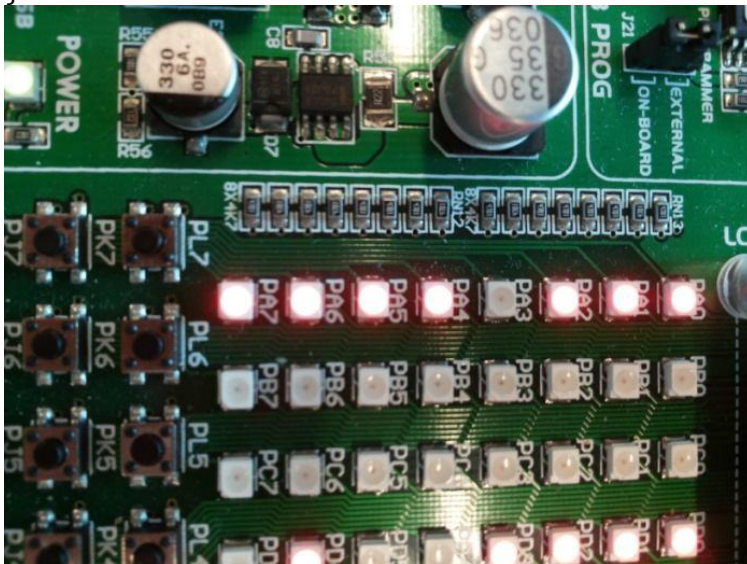
```
#define BIT(x) (1 << (x))
void wait( int ms )
{
    for (int tms=0; tms<ms; tms++)
    {
        _delay_ms( 1 );
    }
}
void adcInit( void )
{
    ADMUX = 0b01100011;
    ADCSRA = 0b11100110;
}
int main( void )
{
    DDRF = 0x00; // set PORTF for input (ADC)
    DDRA = 0xFF; // set PORTA for output
    DDRB = 0xFF; // set PORTB for output
    adcInit(); // initialize ADC
    while (1)
    {
        PORTA = ADCH;
        wait(100);
    }
}
```



## Opdracht 4.4

In het programma van opdracht 4.3 is de ADC ingesteld op de free-running mode. Verander het programma (voor kanaal 3 en voor 8 bits) zodat het alleen een AD-conversie uitvoert als jij dat wilt, dus op aanvraag. Maak daarvoor in main() een eindeloze lus met een wachtfunctie en een start voor de ADC (zie BB-dag4\_4.c).

```
#define BIT(x) (1 << (x))
void wait( int ms )
{
    for (int tms=0; tms<ms; tms++)
    {
        _delay_ms( 1 );
    }
}
void adcInit( void )
{
    ADMUX = 0b11100011;
    ADCSRA = 0b10000110;
}
int main( void )
{
    DDRF = 0x00; // set PORTF for input (ADC)
    DDRA = 0xFF; // set PORTA for output
    adcInit(); // initialize ADC
    while (1)
    {
        ADCSRA |= BIT(6); // Start ADC
        while ( ADCSRA & BIT(6) ); // Wait for completion
        PORTA = ADCH; // Show MSB (bit 9:2) of ADC
        wait(500); // every 50 ms
    }
}
```



Op deze foto is te zien dat de ad conversie op aanvraag word uitgevoerd.

## Opdracht 4.5

Het is mogelijk ADC op interruptbasis te laten plaatsvinden. Dit wil zeggen als de ADC klaar is met ADConversie genereert het een interrupt. Zie hiervoor ADCSRA register bit 4 en 3. Het is daarbij wenselijk dat de ADC gestart wordt door de code van een timer: Als de ingestelde tijd van de timer verstreken is wordt er een interrupt gegenereerd. In de ISR kan je de ADC laten starten.

Ondertussen kan je programma iets anders doen. Op BB staat een voorbeeldprogramma dat elke 500 ms op interruptbasis een AD waarde van kanaal 1 op de LEDs laat zien. Pas dat programma aan:

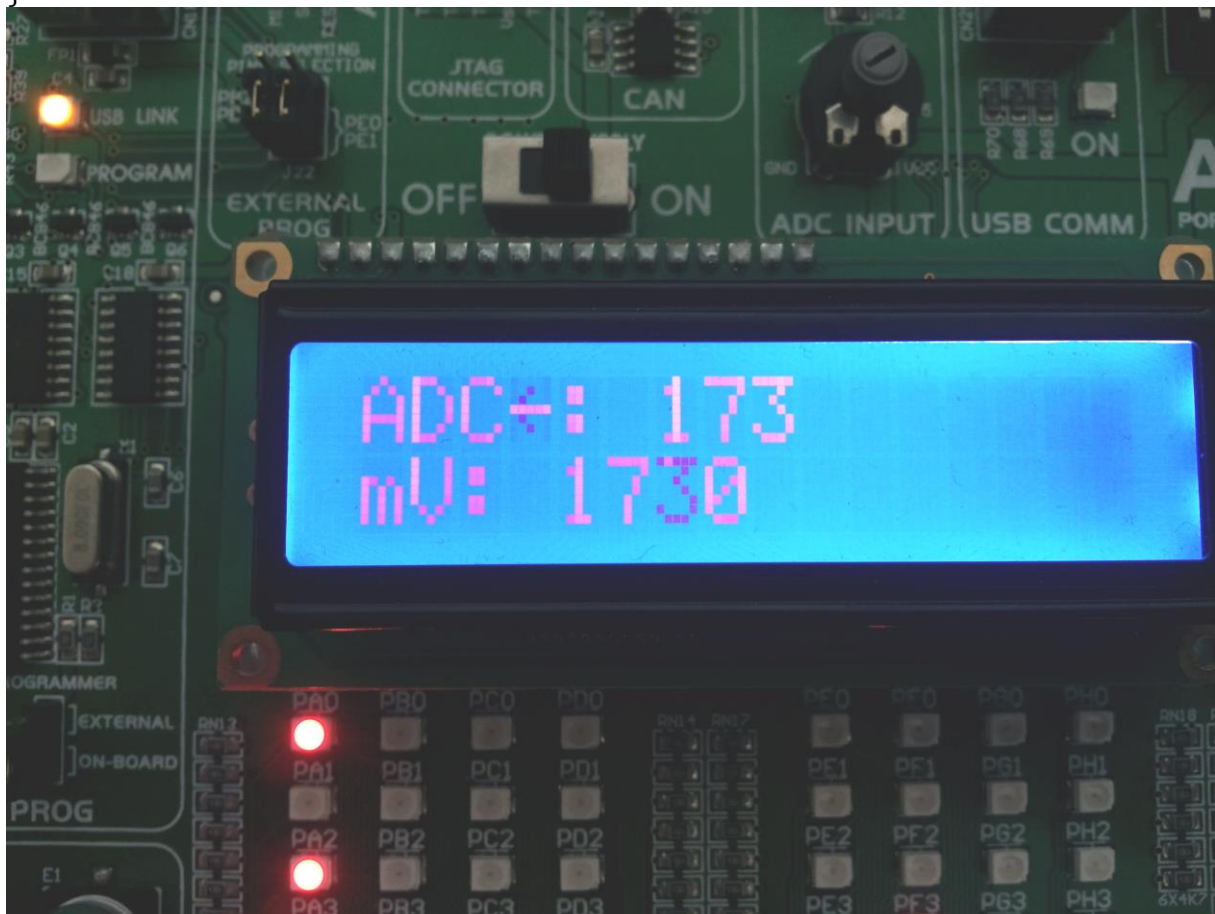
- elke seconde een sample van kanaal 3.
- toon de waarde op het LCD display (gebruik lcd.h en lcd.c van week 2).
- toon ook de waarde op het LCD-display omgerekend in mV.

```
#define BIT(x) (1 << (x))
#define INTERVAL 50
unsigned int tmp = 0;
void wait( int ms )
{
    for (unsigned int ms_counter=0; ms_counter<ms; ms_counter++)
    {
        _delay_ms( 1 );
    }
}
void adcInit( void )
{
    ADMUX = 0b11100011;
    ADCSRA = 0b10001110;
}
void adcStart( void )
{
    ADCSRA |= BIT(6);
}
ISR( ADC_vect )
{
    PORTA = ADCH;
}
ISR( TIMER2_COMP_vect )
{
    tmp++;
    if ( tmp == INTERVAL )
    {
        adcStart();
        tmp = 0;
        PORTC ^= BIT(7);
    }
}
void timer2Init( void )
{
    OCR2 = 250;
    TIMSK |= BIT(7);
    SREG |= BIT(7);
    TCCR2 = 0b00001011;
    disconnected, CTC, RUN
}
int main( void )
{
    DDRF = 0x00; // set PORTF for input (ADC)
    DDRC = 0xFF; // set PORTC for output
    DDRA = 0xFF; // set PORTA for output
}
```

```

adcInit();
timer2Init();
init_lcd();
char output[] = "";
while (1)
{
    PORTC ^= BIT(0);
    PORTA.0
    sprintf(output, "ADCH: %i", ADCH);
    lcd_writeLine1(output);
    wait(1000); // every 1000 ms
}
}

```

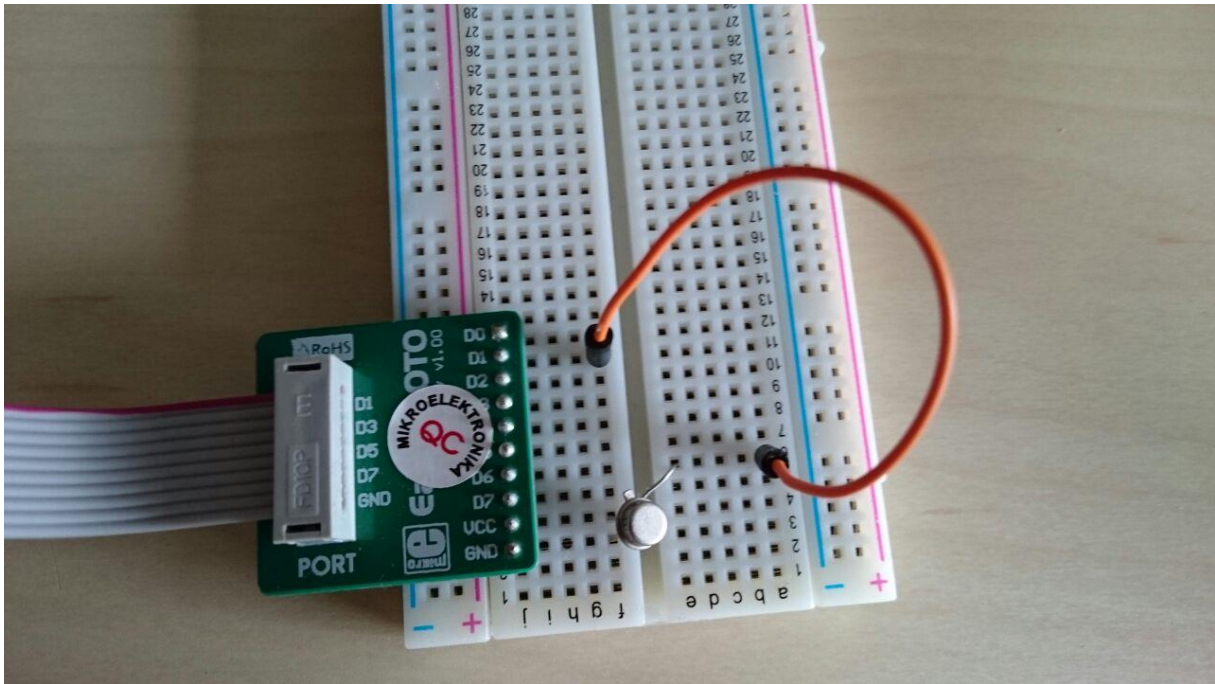


Op deze afbeelding is te zien wat er op het beeldscherm weergegeven word.

## Opdracht 4.6

LM35 is een temperatuurgevoelige sensor met een lineaire spanningsafgifte van  $10\text{mV}/^{\circ}\text{C}$ . Als de deze LM35 aansluit op een ADC kan je de temperatuur aflezen. Gebruik het easy proto board. Dit is de adapter van 10 pins IDC naar breadboard. Bouw de schakeling op het breadboard. De sensor is rechtstreeks aan te sluiten op VCC(+5V), ground (0V) en de ADC-ingang. Zie het schema hieronder en ook de foto die op Blackboard staat. LM35 geeft een spanning af van  $10\text{mV}$  per  $^{\circ}\text{C}$ . Bij een temperatuur van bijvoorbeeld  $35^{\circ}\text{C}$  geeft deze sensor een spanning af van  $350\text{mV}$ . Stel de ADC zo in dat voor de referentiespanning de interne spanning van  $2,56\text{ V}$  wordt gebruikt. Als je dan de hoogste 8 bits van de ADC gebruikt, dan komt 1 bit precies overeen met  $10\text{ mV} = 1^{\circ}\text{C}$ . Laat de waarde van ADCH op de LED's van een poort zien. Laat de waarde van temperatuur op het LCD-display verschijnen.

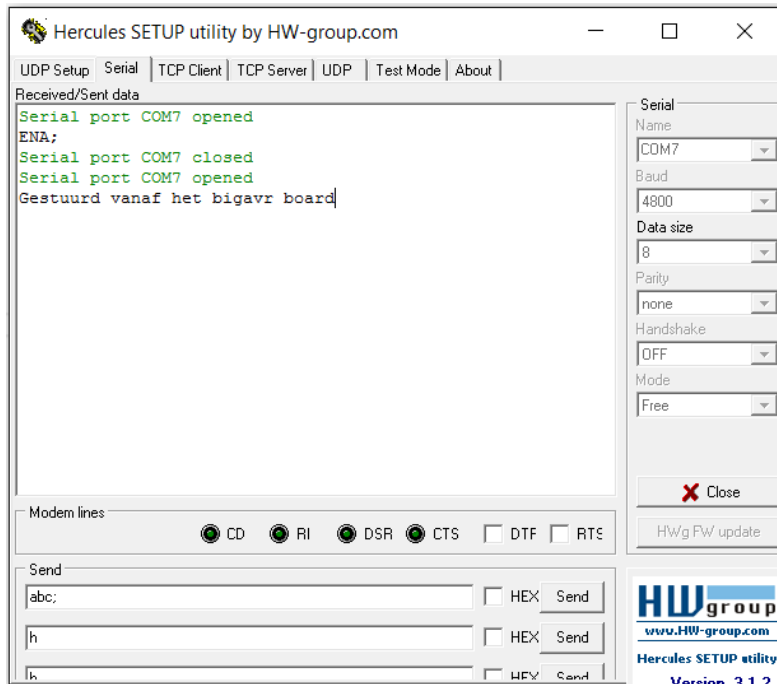
Deze opdracht lukte niet, we kregen de waarde niet op het LCD display te zien.



Dit is de opstelling die we gebruikt hebben voor de temperatuur sensor.

## Opdracht 5.1

Zorgt dat de RS232-verbinding op het bord werkt..



## Opdracht 5.2

a) Run het programma `dag5_1a.c` (zie Blackboard en bijlage 5.1). Het leest een karakter van de seriële poort (usart0) en stuurt het weer terug. Wanneer je de BIGAVR aansluit op de PC functioneert deze op dezelfde manier als het draadje van opdracht 5.1.

werkt

b) Verander de baudrate in het venster van 'Hercules' naar 4800 baud en bekijk het resultaat. Ga na wat je moet veranderen in het programma `dag5_1a.c` om ook het BIGAVR-board te laten functioneren op 4800 baud.

De constante baudrate in het programma moet aangepast worden.

c) Het BIGAVR-board is natuurlijk meer dan een draadje: Pas het programma zo aan het van elke kleine letter die het ontvangt een hoofdletter maakt en deze dan terugstuurt.

### Toelichting:

Als we een willekeurige letter nemen in het ascii tabel dan kunnen we die omzetten in bits. Vervolgens is het zo dat als we de 6<sup>e</sup> bit toggelen, elke kleine letter een hoofdletter wordt (en elke kleine letter een hoofdletter). Laten we daar nu net een functie voor hebben genaamd **BIT()**.

**Code:**

```
character ^= BIT(5);

int main( void ){
    DDRB = 0xFF;
    DDRA = 0xFF;
    usart0_init();
    usart0_start();
    while (1){
```

```
        wait(50);  
        PORTB ^= BIT(7);  
        character = uart0_receiveChar();  
        PORTA = character;  
        uart0_sendChar(character);  
    }  
}
```

d) Maak een codefile voor een module `uart0.c`, waarvan de headerfile, `uart0.h`, in bijlage 5.2 staat. Voeg deze `uart0.c` en `uart0.h` file toe aan je project en pas de code van het programma van opdracht 5.2c aan, zodat het gebruik maakt van deze `uart0`-module.

```

#include "uart0.h"

void wait( int ms )
{
    for (int tms=0; tms<ms; tms++)
    {
        _delay_ms( 1 ); // library function (max 30 ms at 8MHz)
    }
}

void usart0_init( void ) // initialize USART0 - receive/transmit
{
    int ubrr = MYUBRR;
    UBRR0H = ubrr>>8; // baudrate register, hoge byte
    UBRR0L = ubrr; // baudrate register, lage byte
    UCSR0C = 0b00000110; // asynchroon, 8 data - 1 stop - no parity
    UCSR0B = 0b00000000; // receiver & transmitter enable
}

void usart0_start( void ) // receiver & transmitter enable
{
    UCSR0B |= BIT(RXEN)|BIT(TXEN); // set bit RXEN = Receiver enable and TXEN = Transmitter enable
}

int uart0_sendChar( char ch )
{
    while (!(UCSR0A & BIT(UDRE0))) ; // wait until UDRE0 is set: transmit buffer is ready
    UDR0 = ch; // send ch
    return 0; // OK
}

char uart0_receiveChar( void )
{
    while (!(UCSR0A & BIT(RXC0))) ; // if RX0 is set: unread data present in buffer
    return UDR0; // read ch
}

#ifndef _UART
#define _UART
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define LF 0x0a // ascii code for Line Feed
#define CR 0x0d // ascii code for Carriage Return
#define BIT(x) (1 << (x))
#define UART0_BAUD 9600 // Baud rate USART0
#define MYUBRR F_CPU/16/UART0_BAUD - 1 // My USART Baud Rate Register

void usart0_init( void ) ; // initialize USART0 - receive/transmit

void usart0_start( void ) ; // UART0 receiver & transmitter enable
char uart0_receiveChar( void ) ; // UART0: receive ch
int uart0_sendChar( char ) ; // UART0: send ch
int uart0_receiveString( char* ); // UART0: receive string until LF
void wait( int );
#endif

```



## Opdracht 5.3

- a) Sluit de dB-meter aan op je PC (met serial converter en programma Hercules). De seriële poort van de dB-meter werkt op: 2400 baud, 8 databits, 1 stopbit, no parity, DTR on. Dat laatste is nodig om de dB-meter van de juist spanning te voorzien om RS232-signalen te kunnen leveren. In het venster van Hercules dien je daarom ergens een vinkje te zetten: de dB-meter levert meetwaarden, afsloten met één of meerdere karakters. Hercules kan alle karakters laten zien: Stel dat in door op de rechter muisknop te klikken wanneer je het beeldvenster aanwijst. Noteer het formaat en de gehele inhoud van de datastring die de dB-meter verstuurt.

Via PuTTY komen de juiste meetwaarden eruit. Verder komt er netjes de meetwaarde uitgerold.

- b) Run het programma dag5\_2a.c op de microcontroller. Dit programma leest de data van de RS232-poort, toont de bytes op de led's en slaat ze op in een buffer. Verander het programma zo, dat de meetwaarde op het lcd-display getoond wordt. Maak daarbij gebruik van de module 'lcd', met code file lcd.c en headerfile lcd.h die al eerdere gebruikt zijn. Deze modules staan op Blackboard; je kan het aanpassen/aanvullen waar nodig.

De lcd.h wordt weer geïmporteerd in het project.

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "lcd.h"
#include "uart0.h"
#define BIT(x) (1 << (x))
// Baud rate USART0
#define MYUBRR F_CPU/16/USART0_BAUD - 1 // My USART Baud Rate Register

char character;

int main( void )
{
    char buffer[13] = "12345676"; // declare string buffer
    DDRB = 0xFF;
    DDRA = 0xFF; // set PORTB for outputbuf

    lcd_init(); // initialize LCD-display
    uart0_init(); // initialize USART0
    uart0_start();

    while (1)
    {
        wait(50); // every 50 ms (busy waiting)
        PORTB ^= BIT(7); // toggle bit 7 for testing

        uart0_receiveString( buffer ); // receive string from uart
        PORTA = buffer[0];
        lcd_writeLine1(buffer); // write string to LCD display
    }
}
```





## Opdracht 5.5

- a) Sluit het 4-digit display aan op poort B van het ontwikkelbord.

PB0 = slave select

PB1 = SCLK

PB2= MOSI

In bijlage 5.4 staat de code voor het schrijven naar dit display, maar slechts voor de eerste twee digits. Laat het programma draaien en constateer dit.

Het programma wordt uitgevoerd en het schrijven naar het display werkt inderdaad alleen maar voor de eerste twee digits.

- b) Vul de code aan met de aanpassingen zoals hieronder aangegeven en test het na elke aanpassing.

Geef in je verslag duidelijk aan waar je de code hebt aangepast en waarom en hoe. Geef ook commentaar in de code. Lat ook de uitvoer zien. Bekijk de Datasheet voor de aansturing van de registers.

b1. Stuur alle vier digits aan en laat daar de cijfers 1-2-3-4 verschijnen.

b2. Zet het display op maximale intensiteit.

Om al de digits aan te sturen werd de waarde van het scannen op drie gezet.

### Code:

```
#define F_CPU 8000000
#include <avr/io.h>
#include <util/delay.h>
#define BIT(x) ( 1<<x )
#define DDR_SPI DDRB // spi Data direction register
#define PORT_SPI PORTB // spi Output register
#define SPI_SCK 1 // PB1: spi Pin System Clock
#define SPI_MOSI 2 // PB2: spi Pin MOSI
#define SPI_MISO 3 // PB3: spi Pin MISO
#define SPI_SS 0 // PB0: spi Pin Slave Select

void wait(int ms)
{
    for (int i=0; i<ms; i++)
        _delay_ms(1);
}

void spi_masterInit(void)
```

```

{
    DDR_SPI = 0xff; // All pins output: MOSI, SCK,
    SS, SS_display
    DDR_SPI &= ~BIT(SPI_MISO); // except: MISO input
    PORT_SPI |= BIT(SPI_SS); // SS_ADC == 1: deselect slave
    SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR1); // or: SPCR = 0b11010010;
    // Enable spi, MasterMode, Clock rate fck/64
    // bitrate=125kHz, Mode = 0: CPOL=0, CPPH=0
}
// Write a byte from master to slave
void spi_write( unsigned char data )
{
    SPDR = data; // Load byte --> starts transmission
    while( !(SPSR & BIT(SPIF)) ); // Wait for transmission complete
}
// Write a byte from master to slave and read a byte from slave - not used
here
char spi_writeRead( unsigned char data )
{
    SPDR = data; // Load byte --> starts transmission
    while( !(SPSR & BIT(SPIF)) ); // Wait for transmission complete
    data = SPDR; // New received data (eventually, MISO)
    in SPDR
    return data; // Return received byte
}
// Select device on pinnr PORTB
void spi_slaveSelect(unsigned char chipNumber)
{
    PORTB &= ~BIT(chipNumber);
}
// Deselect device on pinnr PORTB
void spi_slaveDeSelect(unsigned char chipNumber)
{
    PORTB |= BIT(chipNumber);
}
// Initialize the driver chip (type MAX 7219)
void displayDriverInit()
{
    spi_slaveSelect(0); // Select display chip (MAX7219)
    spi_write(0x09); // Register 09: Decode Mode
    spi_write(0xFF); // -> 1's = BCD mode for all digits
    spi_slaveDeSelect(0); // Deselect display chip
    spi_slaveSelect(0); // Select display chip
    spi_write(0x0A); // Register 0A: Intensity
    spi_write(0x0F); // -> Level 4 (in range [1..F])
    spi_slaveDeSelect(0); // Deselect display chip
    spi_slaveSelect(0); // Select display chip
    spi_write(0x0B); // Register 0B: Scan-limit
    spi_write(0x03); // -> 1 = Display digits 0..3
    spi_slaveDeSelect(0); // Deselect display chip
    spi_slaveSelect(0); // Select display chip
    spi_write(0x0C); // Register 0B: Shutdown register
    spi_write(0x01); // -> 1 = Normal operation
    spi_slaveDeSelect(0); // Deselect display chip
}
// Set display on ('normal operation')
void displayOn()
{
    spi_slaveSelect(0); // Select display chip
    spi_write(0x0C); // Register 0B: Shutdown register
    spi_write(0x01); // -> 1 = Normal operation
}

```

```

    spi_slaveDeSelect(0); // Deselect display chip
}
// Set display off ('shut down')
void displayOff()
{
    spi_slaveSelect(0); // Select display chip
    spi_write(0x0C); // Register 0B: Shutdown register
    spi_write(0x00); // -> 1 = Normal operation
    spi_slaveDeSelect(0); // Deselect display chip
}
// Write a word = address byte + data byte from master to slave
void spi_writeWord( unsigned char address, unsigned char data ){
    spi_slaveSelect(0); // Select display chip
    spi_write(address); // digit address: (digit place)
    spi_write(data); // digit value: 0
    spi_slaveDeSelect(0); // Deselect display chip
}
int main()
{
    DDRB=0x01; // Set PB0 pin as output for
display select
    spi_masterInit(); // Initialize spi module
    displayDriverInit(); // Initialize display chip
    // clear display (all zero's)
    for (char i =1; i<=4; i++) // For modules 1 to 4
    {
        spi_writeWord(i, 0); // write a word
    }
    wait(1000);
    // write 4-digit data
    for (char i =1; i<=4; i++) // For modules 1 to 4
    {
        spi_writeWord(i, 5-i); // write a word
        wait(1000);
    }
    wait(1000);
    return (1);
}

```

### Resultaat:

Het display geeft de waarden weer met de maximale intensiteit.

- c) Voor het schrijven van een waarde naar het 4-digit display zijn telkens 4 acties nodig:

Slave select

Schrijven van een adres naar het display

Schrijven van een waarde naar de inhoud op dit adres

Slave deselect

Bijv.:

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

spi\_write(0x0A); // Register 0A: Intensity

spi\_write(0x04); // -> Level 4 (in range [1..F])

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

Maak een functie die deze vier acties achtereenvolgens uitvoert. Deze functie heeft de volgende

header:

// Write a word = address byte + data byte from master to slave

void spi\_writeWord( unsigned char address, unsigned char data )

**Toelichting:**

De code uit de main kan hergebruikt worden.

```
spi_slaveSelect(0);
spi_write(address);
spi_write(data);
spi_slaveDeSelect(0);
```

Alleen zijn de parameters niet hardcoded

**Code:**

```
#define F_CPU 8000000
#include <avr/io.h>
#include <util/delay.h>
#define BIT(x) ( 1<<x )
#define DDR_SPI DDRB // spi Data direction register
#define PORT_SPI PORTB // spi Output register
#define SPI_SCK 1 // PB1: spi Pin System Clock
#define SPI_MOSI 2 // PB2: spi Pin MOSI
#define SPI_MISO 3 // PB3: spi Pin MISO
#define SPI_SS 0 // PB0: spi Pin Slave Select
// wait(): busy waiting for 'ms' millisecond - used library: util/delay.h
void wait(int ms)
{
    for (int i=0; i<ms; i++)
        _delay_ms(1);
}
void spi_masterInit(void)
{
    DDR_SPI = 0xff; // All pins output: MOSI, SCK,
    SS, SS_display
    DDR_SPI &= ~BIT(SPI_MISO); // except: MISO input
    PORT_SPI |= BIT(SPI_SS); // SS_ADC == 1: deselect slave
    SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR1); // or: SPCR = 0b11010010;
    // Enable spi, MasterMode, Clock rate fck/64
    // bitrate=125kHz, Mode = 0: CPOL=0, CPPH=0
}
// Write a byte from master to slave
void spi_write( unsigned char data )
{
    SPDR = data; // Load byte --> starts transmission
    while( !(SPSR & BIT(SPIF)) ); // Wait for transmission complete
}
// Write a byte from master to slave and read a byte from slave - not used
here
char spi_writeRead( unsigned char data )
{
    SPDR = data; // Load byte --> starts transmission
    while( !(SPSR & BIT(SPIF)) ); // Wait for transmission complete
    data = SPDR; // New received data (eventually, MISO)
    in SPDR
    return data; // Return received byte
}
// Select device on pinnumber PORTB
void spi_slaveSelect(unsigned char chipNumber)
{
    PORTB &= ~BIT(chipNumber);
```

```

}
// Deselect device on pinnumber PORTB
void spi_slaveDeSelect(unsigned char chipNumber)
{
    PORTB |= BIT(chipNumber);
}
// Initialize the driver chip (type MAX 7219)
void displayDriverInit()
{
    spi_slaveSelect(0); // Select display chip (MAX7219)
    spi_write(0x09); // Register 09: Decode Mode
    56
    spi_write(0xFF); // -> 1's = BCD mode for all digits
    spi_slaveDeSelect(0); // Deselect display chip
    spi_slaveSelect(0); // Select display chip
    spi_write(0x0A); // Register 0A: Intensity
    spi_write(0x0F); // -> Level 4 (in range [1..F])
    spi_slaveDeSelect(0); // Deselect display chip
    spi_slaveSelect(0); // Select display chip
    spi_write(0x0B); // Register 0B: Scan-limit
    spi_write(0x03); // -> 1 = Display digits 0..3
    spi_slaveDeSelect(0); // Deselect display chip
    spi_slaveSelect(0); // Select display chip
    spi_write(0x0C); // Register 0B: Shutdown register
    spi_write(0x01); // -> 1 = Normal operation
    spi_slaveDeSelect(0); // Deselect display chip
}
// Set display on ('normal operation')
void displayOn()
{
    spi_slaveSelect(0); // Select display chip
    spi_write(0x0C); // Register 0B: Shutdown register
    spi_write(0x01); // -> 1 = Normal operation
    spi_slaveDeSelect(0); // Deselect display chip
}
// Set display off ('shut down')
void displayOff()
{
    spi_slaveSelect(0); // Select display chip
    spi_write(0x0C); // Register 0B: Shutdown register
    spi_write(0x00); // -> 1 = Normal operation
    spi_slaveDeSelect(0); // Deselect display chip
}
// Write a word = address byte + data byte from master to slave
void spi_writeWord( unsigned char adress, unsigned char data ){
    spi_slaveSelect(0); // Select display chip
    spi_write(adress); // digit adress: (digit place)
    spi_write(data); // digit value: 0
    spi_slaveDeSelect(0); // Deselect display chip
}
int main()
{
    DDRB=0x01; // Set PB0 pin as output for
    display select
    spi_masterInit(); // Initialize spi module
    displayDriverInit(); // Initialize display chip
    // clear display (all zero's)
    for (char i =1; i<=4; i++) // For modules 1 to 4
    {
        spi_writeWord(i, 0); // write a word
    }
}

```

```

    wait(1000);
    // write 4-digit data
    for (char i =1; i<=4; i++) // For modules 1 to 4
    {
        spi_writeWord(i, 5-i); // write a word
        wait(1000);
    }
    wait(1000);
    return (1);
}

```

De code in de for loops is vervangen door de method en alles werkt net als voorheen.

## Opdracht 5.6

- a) Maak een functie dat de waarde van een niet-negatieve integer van maximaal 4 cijfers op het display laat zien, bijv. met de header: `void writeLedDisplay( int value );` // toont de waarde van value op het 4-digit display. Geef een toelichting bij de code en test deze functie.
- b) Pas de functie van b3) aan zo dat ook negatieve gehele getallen van maximaal drie cijfers getoond kunnen worden. Geef een toelichting bij de code en test deze functie.

Om de getallen op te splitsen in digits maken we gebruik van modulo. Door steeds modulo 10 te doen kan elke digit los maken van een getal en daarna naar het display schrijven. Als dat gebeurt is en het getal is negatief dan schrijven we een min teken naar het eerste display segment. Hierdoor kan je gebruik maken van negatieve getallen.

### Code:

```

#define F_CPU 8000000
#include <avr/io.h>
#include <util/delay.h>
#define BIT(x) ( 1<<x )
#define DDR_SPI DDRB // spi Data direction register
#define PORT_SPI PORTB // spi Output register
#define SPI_SCK 1 // PB1: spi Pin System Clock
#define SPI_MOSI 2 // PB2: spi Pin MOSI
#define SPI_MISO 3 // PB3: spi Pin MISO
#define SPI_SS 0 // PB0: spi Pin Slave Select
// wait(): busy waiting for 'ms' millisecond - used library: util/delay.h
void wait(int ms)
{
    for (int i=0; i<ms; i++)
        _delay_ms(1);
}
void spi_masterInit(void)
{
    DDR_SPI = 0xff; // All pins output: MOSI, SCK,
    SS, SS_display
    DDR_SPI &= ~BIT(SPI_MISO); // except: MISO input
    PORT_SPI |= BIT(SPI_SS); // SS_ADC == 1: deselect slave
    SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR1); // or: SPCR = 0b11010010;
    // Enable spi, MasterMode, Clock rate fck/64
    // bitrate=125kHz, Mode = 0: CPOL=0, CPPH=0
}
// Write a byte from master to slave
void spi_write( unsigned char data )
{

```

```

        SPDR = data; // Load byte --> starts transmission
        while( !(SPSR & BIT(SPIF)) ); // Wait for transmission complete
    }
    // Write a byte from master to slave and read a byte from slave - not used
    here
    char spi_writeRead( unsigned char data )
    {
        SPDR = data; // Load byte --> starts transmission
        while( !(SPSR & BIT(SPIF)) ); // Wait for transmission complete
        data = SPDR; // New received data (eventually, MISO)
        in SPDR
        return data; // Return received byte
    }
    // Select device on pinnumber PORTB
    void spi_slaveSelect(unsigned char chipNumber)
    {
        PORTB &= ~BIT(chipNumber);
    }
    // Deselect device on pinnumber PORTB
    void spi_slaveDeSelect(unsigned char chipNumber)
    {
        PORTB |= BIT(chipNumber);
    }
    // Initialize the driver chip (type MAX 7219)
    void displayDriverInit()
    {
        spi_slaveSelect(0); // Select display chip (MAX7219)
        spi_write(0x09); // Register 09: Decode Mode
        spi_write(0xFF); // -> 1's = BCD mode for all digits
        spi_slaveDeSelect(0); // Deselect display chip
        spi_slaveSelect(0); // Select display chip
        spi_write(0x0A); // Register 0A: Intensity
        spi_write(0x0F); // -> Level 4 (in range [1..F])
        spi_slaveDeSelect(0); // Deselect display chip
        spi_slaveSelect(0); // Select display chip
        spi_write(0x0B); // Register 0B: Scan-limit
        spi_write(0x03); // -> 1 = Display digits 0..3
        spi_slaveDeSelect(0); // Deselect display chip
        spi_slaveSelect(0); // Select display chip
        spi_write(0x0C); // Register 0B: Shutdown register
        spi_write(0x01); // -> 1 = Normal operation
        spi_slaveDeSelect(0); // Deselect display chip
    }
    // Set display on ('normal operation')
    void displayOn()
    {
        spi_slaveSelect(0); // Select display chip
        spi_write(0x0C); // Register 0B: Shutdown register
        spi_write(0x01); // -> 1 = Normal operation
        spi_slaveDeSelect(0); // Deselect display chip
    }
    // Set display off ('shut down')
    void displayOff()
    {
        spi_slaveSelect(0); // Select display chip
        spi_write(0x0C); // Register 0B: Shutdown register
        spi_write(0x00); // -> 1 = Normal operation
        spi_slaveDeSelect(0); // Deselect display chip
    }
    // Write a word = address byte + data byte from master to slave
    void spi_writeWord( unsigned char adress, unsigned char data ){

```

```

    spi_slaveSelect(0); // Select display chip
    spi_write(address); // digit address: (digit place)
    spi_write(data); // digit value: data
    spi_slaveDeSelect(0); // Deselect display chip
}
// toont de waarde van value op het 4-digit display
void writeLedDisplay( int value ){
    int number=value<0?-value:value;
    int address = 1;
    if(value > -1000 && value < 10000){
        while (number > 0){
            spi_writeWord(address, number % 10);
            number /= 10;
            adress++;
        }
        if (value < 0)
        {
            spi_writeWord(4, 10);
        }
    }
    else{
        for (address; address <= 4; adress++){
            spi_writeWord(address, 8);
        }
    }
}
int main()
{
    DDRB=0x01; // Set PB0 pin as output for
display select
    spi_masterInit(); // Initialize spi module
displayDriverInit(); // Initialize display chip
//clear display (all zero's)
for (char i =1; i<=4; i++) // For modules 1 to 4
{
    spi_writeWord(i, 0); // write a word
}
wait(1000);

writeLedDisplay(3);
wait(1000);
return (1);
}

```

### Resultaat:

Er worden alleen getallen weergegeven die op het scherm passen. Wanneer dit niet het geval is word het getal 8888 getoond.

