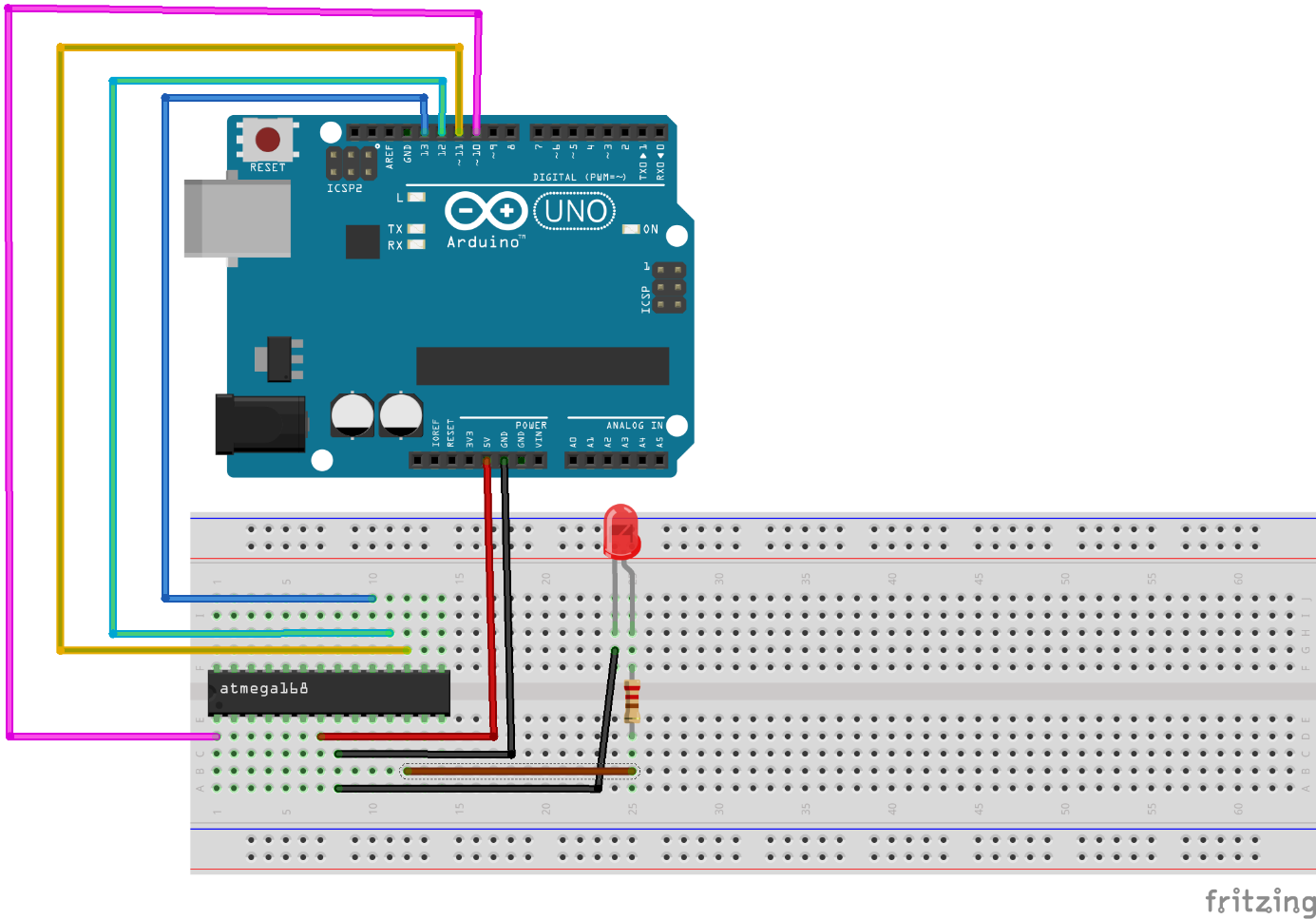
|  |  |
| --- | --- |
| IMTHE VErslag  Versie 1.0 | Duncan Pronk – s1071770  2014 |

Inhoud

# Week 1

## Installeren omgeving

Na mijn ervaring van de eerste poging, wist ik dat de omgeving opzetten niet heel moeilijk was, zolang je maar aan het stappenplan hield. Nadat ik Eclipse, USBASP, WINAVR en alle plug-ins had geïnstalleerd, had ik de test opstelling gebouwd. Het ledje deed het, maar heel vaag. Met een simpele blinkled code bleek dat Eclipse een foutmelding gaf. Na onderzoek bleek dit een probleem te zijn met de AVR-GCC toolchain en windows 8. Er stonden een aantal oplossingen op, maar die werkte allemaal niet. Ik heb toen besloten om mijn Arduino te gebruiken als programming header. Arduino ISP. Het enige wat mistte was een Makefile. Ik heb deze toen zelf geschreven. De setup voor het programmeren ziet er nu als volgt uit.



De Makefile heb ik als bijlage toegevoegd. Na een beetje research naar andere voorbeelden, bleek dat ik iemands werk bijna helemaal had overgedaan. Ik heb de comments van hem overgenomen, daarom hierbij: Makefile comments zijn gemaakt door Michael Holachek.

Ik zou fritzing gebruiken voor lastige aansluitingen. Tevens gebruik ik github samen met Sourcetree om mijn code te synchroniseren tussen verschillende werkstations. Mijn github is te vinden op <https://github.com/crazytim412/imthe> . Daarnaast zet ik alle filmpjes op Youtube. De link hier naar toe is: <http://www.youtube.com/playlist?list=PLMN6M1F9SxV10igUPme-DtYLGgaE6KJpT> .

## Verschillen met Java

|  |  |  |
| --- | --- | --- |
| Verschil | Java | C |
| Halve klassen | Nee | Ja |
| Enums in de interface | Ja | Nee |
| Pointers | Nee | Ja |
| Multidimensionale arrays | Nee | Ja |
| Overflow controle | Nee | Ja |
| Migratie | Ja | Nee |

## Overeenkomsten met Java

|  |
| --- |
| Overeenkomsten tussen Java en C |
| Signed 8, 16, 32 en 64 bits integers |
| Data abstractie |
| Data sealing |
| Klasse opbouw (c file, h file) |
| Constanten |
| Deprecation |

# Week 2

## Blink Led

Video:

#include <avr/io.h> // Laadt de IO library, deze bevat de poorten

#include <util/delay.h> // Laadt de Delay library, nu kan je een delay instellen

#include <stdio.h> // Laadt de stdio library, deze vat de meeste nodige onderdelen

int main() // Hoofd functioe

{

DDRB = 0x00000001; // Zet de ricthing van poort PORTB0 op output

while(1) // Doe iets voor altijd

{

PORTB = 0x00000001; // Zet PORTB0 hoog, dus led gaat aan

\_delay\_ms(200); // wacht 200 miliseconden

PORTB = 0x00000000; // Zet PORTB0 laag, dus led gaat uit

\_delay\_ms(200); // wacht 200 miliseconden

}

}

## Fuses

Fuses zijn van te voren vastgestelde code in de BIOS. De BIOS van een microcontroller zijn meestal bits uit de flash memory of uit het EEPROM. Fuses zijn bijvoorbeeld welke clock de microcontroller moet gebruiken of als de watchdog altijd aan moet staan.

Een handige site die ik al eerder heb gelinkt is: <http://www.engbedded.com/fusecalc> . Je kan hierop alle fuses zien en wat ze doen per microcontroller.

## Teller 7-segment display

### Alleen cijfers

// De poorten van de 7-segment display heb ik geteld van 0 tot en met 18

// Links onderin is poort 0, rechtsboven is poort 10

#define F\_CPU 1000000 // Snelheid van de CPU, met dit kristal en deze waarde, is het precies 1 seconde

#include <avr/io.h> // Laadt de IO library, deze bevat de poorten

#include <util/delay.h> // // Laadt de Delay library, nu kan je een delay instellen

#include <stdio.h> // Laadt de stdio library, deze vat de meeste nodige onderdelen

int main() // Hoofd functie

{

DDRD = 0b11111111; // Alle poorten van D op output

DDRB = 0b00000011; // B1, B2, B3 worden op output gezet

PORTB = 0b00000000; // Zet alle poorten van B op laag

// Poorten 13 en 14 zijn de common anode

// Poort 4 en 9 zijn de dp en worden niet gebruikt

// PD0 = linksonderin | Pin 1 en 5

// PD1 = Midden | Pin 7 en 17

// PD2 = rechtsonderin | Pin 3 en 8

// PD3 = Beneden | Pin 2 en 6

// PD4 = linksboven | Pin 18 en 12

// PD5 = boven | Pin 16 en 11

// PD6 = rechtsboven | Pin 15 en 10

char cijfer[10]; // Array van 16 plaatsen voor de cijfers

cijfer[0] = 0b00000010; // Nul

cijfer[1] = 0b00111011; // Een

cijfer[2] = 0b00010100; // Twee

cijfer[3] = 0b00010001; // Drie

cijfer[4] = 0b00101001; // Vier

cijfer[5] = 0b01000001; // Vijf

cijfer[6] = 0b01000000; // Zes

cijfer[7] = 0b00011011; // Zeven

cijfer[8] = 0b00000000; // Acht

cijfer[9] = 0b00000001; // Negen

void nummerLinks() // Linker nummer functie

{

PORTB = 0b00000010; // Zet alleen PB1 aan

}

void nummerRechts() // Rechter nummer functie

{

PORTB = 0b00000001; // Zet alleen PB0 aan

}

void welkCijfer(int eerste, int tweede) // Functie voor het laten zien van de cijfers

{

int teller; // Initieer de teller cijfer ( c99 modus voor de arduinoISP)

for(teller = 0; teller < 200; teller++) // Voor teller is kleiner dan 200, en elke keer als programma hier komt, teller + 1

{

PORTD = cijfer[eerste]; // Zet de portD poorten op het linker cijfer

nummerLinks(); // Voer functie nummerLinks uit

\_delay\_ms(1); // Wacht 1 ms

PORTD = cijfer[tweede]; // Zet portD poorten op het rechter cijfer

nummerRechts(); // Voer functie nummerRechts uit

\_delay\_ms(1); // Wacht 1 ms

}

}

while(1) // Doe iets voor altijd

{

int tellerTwee; // Initieer de tellerTwee voor het tellen om het cijferr te bepalen ( c99 modus voor de arduinoISP)

for(tellerTwee = 0; tellerTwee < 128; tellerTwee++) // Voor tellerTwee is kleiner dan 256, en elke keer als programma hier komt, tellerTwee + 1

{

int eerste = tellerTwee / 10; // Eerste cijfer is tellerTwee gedeeld door 10

int tweede = tellerTwee%10; // Tweede cijfer is tellerTwee modulus 10

welkCijfer(eerste, tweede); // Voer de functie welkCijfer uit

}

}

}

### Cijfers en letters

// De poorten van de 7-segment display heb ik geteld van 0 tot en met 18

// Links onderin is poort 0, rechtsboven is poort 10

#define F\_CPU 1000000 // Snelheid van de CPU, met dit kristal en deze waarde, is het precies 1 seconde

#include <avr/io.h> // Laadt de IO library, deze bevat de poorten

#include <util/delay.h> // // Laadt de Delay library, nu kan je een delay instellen

#include <stdio.h> // Laadt de stdio library, deze vat de meeste nodige onderdelen

int main() // Hoofd functie

{

DDRD = 0b11111111; // Alle poorten van D op output

DDRB = 0b00000011; // B1, B2, B3 worden op output gezet

PORTB = 0b00000000; // Zet alle poorten van B op laag

// Poorten 13 en 14 zijn de common anode

// Poort 4 en 9 zijn de dp en worden niet gebruikt

// PD0 = linksonderin | Pin 1 en 5

// PD1 = Midden | Pin 7 en 17

// PD2 = rechtsonderin | Pin 3 en 8

// PD3 = Beneden | Pin 2 en 6

// PD4 = linksboven | Pin 18 en 12

// PD5 = boven | Pin 16 en 11

// PD6 = rechtsboven | Pin 15 en 10

char cijfer[16]; // Array van 16 plaatsen voor de cijfers en letters

cijfer[0] = 0b00000010; // Nul

cijfer[1] = 0b00111011; // Een

cijfer[2] = 0b00010100; // Twee

cijfer[3] = 0b00010001; // Drie

cijfer[4] = 0b00101001; // Vier

cijfer[5] = 0b01000001; // Vijf

cijfer[6] = 0b01000000; // Zes

cijfer[7] = 0b00011011; // Zeven

cijfer[8] = 0b00000000; // Acht

cijfer[9] = 0b00000001; // Negen

cijfer[10] = 0b00001000; // A

cijfer[11] = 0b01100000; // B

cijfer[12] = 0b01000110; // C

cijfer[13] = 0b00110000; // D

cijfer[14] = 0b01000100; // E

cijfer[15] = 0b01001100; // F

void nummerLinks() // Linker nummer functie

{

PORTB = 0b00000010; // Zet alleen PB1 aan

}

void nummerRechts() // Rechter nummer functie

{

PORTB = 0b00000001; // Zet alleen PB0 aan

}

void welkCijfer(int eerste, int tweede) // Functie voor het laten zien van de cijfers en letters

{

int teller; // Initieer de teller cijfer ( c99 modus voor de arduinoISP)

for(teller = 0; teller < 200; teller++) // Voor teller is kleiner dan 200, en elke keer als programma hier komt, teller + 1

{

PORTD = cijfer[eerste]; // Zet de portD poorten op het linker cijfer

nummerLinks(); // Voer functie nummerLinks uit

\_delay\_ms(1); // Wacht 1 ms

PORTD = cijfer[tweede]; // Zet portD poorten op het rechter cijfer

nummerRechts(); // Voer functie nummerRechts uit

\_delay\_ms(1); // Wacht 1 ms

}

}

while(1) // Doe iets voor altijd

{

int tellerTwee; // Initieer de tellerTwee voor het tellen om het cijfer of letter te bepalen ( c99 modus voor de arduinoISP)

for(tellerTwee = 0; tellerTwee < 256; tellerTwee++) // Voor tellerTwee is kleiner dan 256, en elke keer als programma hier komt, tellerTwee + 1

{

int eerste = tellerTwee / 16; // Eerste cijfer is tellerTwee gedeeld door 16

int tweede = tellerTwee%16; // Tweede cijfer is tellerTwee modulus 16

welkCijfer(eerste, tweede); // Voer de functie welkCijfer uit

}

}

}

## RGB LED met PWM

Video:

#include <avr/io.h> // Laadt de IO library, deze bevat de poorten

#include <util/delay.h> // Laadt de util library en dan Delay onderdeel, deze is nodig voor de delay functie

#include <math.h> // Laadt de reken library, nu kunnen we rekenen.

typedef struct // We willen wat dingen definiëren

{

float rood; //0 of 1

float groen; //0 of 1

float blauw; //0 of 1

} SColorRGB; // We noemen deze define: SColorRGB omdat deze de kleuren bevat

typedef struct // We willen wat dingen definiëren

{

float hue; //0 tot en met 360, kleurverloop

float saturation; //0 of 1

float brightness; //0 of 1

} SColorHSB;

void rgbFromHSB(SColorHSB \_\_hsb,SColorRGB \* \_\_rgb) // functie rgb from HSB

{

if (\_\_hsb.saturation==0)

{

\_\_rgb->rood = \_\_hsb.brightness; // Zet rood op de brightness

\_\_rgb->groen = \_\_hsb.brightness; // Zet groen op de brightness

\_\_rgb->blauw = \_\_hsb.brightness; // Zet blauw op de brightness

}

else

{

float max = \_\_hsb.brightness; // Zet de maximale brightness

float verschil = \_\_hsb.brightness \* \_\_hsb.saturation; // Zet het verschil tussen de sat en de brightness

float min = \_\_hsb.brightness - verschil; // Zet het minimale brightness neer

if (\_\_hsb.hue < 60) // Als de hue minder is dan 60

{

\_\_rgb->rood = max; // Rood wordt de wordt het maximale hue

\_\_rgb->groen = \_\_hsb.hue \* verschil / 60 + min; // Groen wordt de hue \* het verschil gedeeld door 60 + het minimale

\_\_rgb->blauw = min; // Blauw wordt de minimale hue

}

else if (\_\_hsb.hue < 120) // Als de hue minder is dan 120

{

\_\_rgb->rood = -(\_\_hsb.hue - 120) \* verschil / 60 + min; // Rood wordt de negatieve kant van de hue-120 \* het verschil gedeeld 60 + het minimale

\_\_rgb->groen = max; // Groen wordt het maximale hue

\_\_rgb->blauw = min; // Blauw wordt de minimale hue

}

else if (\_\_hsb.hue < 180) // Als de hue minder is dan 180

{

\_\_rgb->rood = min; // Rood wordt de minimale hue

\_\_rgb->groen = max; // Groen wordt het maximale hue

\_\_rgb->blauw = -(\_\_hsb.hue - 120) \* verschil / 60 + min; // Blauw wordt de negatieve kant van de hue - 120 \* het verschil gedeeld 60 + het minimale

}

else if (\_\_hsb.hue < 240) // Als de hue minder is dan 240

{

\_\_rgb->rood = min; // Rood wordt de minimale hue

\_\_rgb->groen = -(\_\_hsb.hue - 240) \* verschil / 60 + min; // Groen wordt de negatieve kant van de hue - 240 \* het verschil gedeeld 60 + het minimale

\_\_rgb->blauw = max; // Blauw wordt het maximale hue

}

else if (\_\_hsb.hue < 300)// Als de hue minder is dan 300

{

\_\_rgb->rood = -(\_\_hsb.hue - 240) \* verschil / 60 + min; // Rood wordt de negatieve kant van de hue - 240 \* het verschil gedeeld 60 + het minimale

\_\_rgb->groen = min; // Groen wordt de minimale hue

\_\_rgb->blauw = max; // Blauw wordt het maximale hue

}

else if (\_\_hsb.hue <= 360) // Als de hue minder is dan 360

{

\_\_rgb->rood = max; // Rood wordt de wordt het maximale hue

\_\_rgb->groen = min; // Groen wordt de minimale hue

\_\_rgb->blauw = -(\_\_hsb.hue - 360) \* verschil / 60 + min; // Blauw wordt de negatieve kant van de hue - 360 \* het verschil gedeeld 60 + het minimale

}

else

{

\_\_rgb->rood = 0; // Zet rood op nul

\_\_rgb->groen = 0; // Zet groen op nul

\_\_rgb->blauw = 0; // Zet blauw op nul

}

}

}

int main (void)

{

DDRB = 0b00001000; // PB3 output

DDRD = 0b01100000; // PD5 en PD6 outputs

SColorHSB hsb; // Nemen de definitie over in een variabelen

SColorRGB rgb;

hsb.hue = 0; // Zetten hue op 0

hsb.saturation = 1; // zetten saturation op 1

hsb.brightness = .5; // zetten helderheid op 0.5

TCCR0A = \_BV(COM0A1) | \_BV(COM0B1) | \_BV(WGM01) | \_BV(WGM00); // Niet inverterende snelle PWM modus

TCCR0B = \_BV(CS00); // Geen prescaling

TCCR2A = \_BV(COM2A1) | \_BV(COM2B1) | \_BV(WGM21) | \_BV(WGM20); // Niet inverterende snelle PWM modus

TCCR2B = \_BV(CS20); // Geen prescaling

while (1) // Doet het voor altijd

{

rgbFromHSB(hsb,&rgb); // Omrekenen van rood, blauw, groen naar hue, brightness en saturation

OCR0A = (unsigned char)(255\*rgb.rood); // Timer 1 is rood

OCR0B = (unsigned char)(185\*rgb.groen); // Timer 2 is groen

OCR2A = (unsigned char)(150\*rgb.blauw); // Timer 3 is blauw

\_delay\_ms(10); // wacht 10 miliseconden

if (hsb.hue==359) // Als de heu 359 is

hsb.hue=0; // Zet de hue op 1

else // anders

hsb.hue++; // Hue = oude Hue + 1

}

return(0); // geef 0 terug

}

## Bronnen

<http://www.protostack.com/blog/2010/09/timer-interrupts-on-an-atmega168/>

De timer interrupts, benamingen en welke poorten door welke timers wordt bestuurd.

<http://expand.xs4all.nl/avr/rgb.html>

Voorbeeld code van hoe je de interrupts gebruikt en wat de fast modus activeerd.

# Week 3

## Lichtknop

#define F\_CPU 1000000 // Snelheid van de CPU, met dit kristal en deze waarde, is het precies 1 seconde

#include <avr/io.h> // Laadt de IO library, deze bevat de poorten

#include <util/delay.h> // // Laadt de Delay library, nu kan je een delay instellen

#include <stdio.h> // Laadt de stdio library, deze vat de meeste nodige onderdelen

int main() // Hoofd functie

{

DDRB = 0b00000011; // Zet B0 en B1 op input

PORTB |= (1<<0); // Activeer de pull up op B1

int pull = 0; // Zet een cijfer genaamd pull op 0

while(1) // Doe iets voor altijd

{

if(PINB & (1 << 0)) // Kijk als PB een input krijgt (digitaal, dus 0 of 1)

{

if(pull == 0) // Als pull gelijk is aan 0

{ // Dan

PORTB = 0b00000000; // Zet poort B1 hoog

pull = 1; // Zet pull op 1

\_delay\_ms(500); // wacht 1000 ms (button debouncing)

}

else // anders

{

PORTB = 0b00000010; // Zet alles laag

pull = 0; // Zet pull op 0

\_delay\_ms(500); // Wacht 1000 ms (button debouncing)

}

}

}

}

## Dendergedrag

### Met delay

// De poorten van de 7-segment display heb ik geteld van 0 tot en met 18

// Links onderin is poort 0, rechtsboven is poort 10

#define F\_CPU 1000000 // Snelheid van de CPU, met dit kristal en deze waarde, is het precies 1 seconde

#include <avr/io.h> // Laadt de IO library, deze bevat de poorten

#include <util/delay.h> // // Laadt de Delay library, nu kan je een delay instellen

#include <stdio.h> // Laadt de stdio library, deze vat de meeste nodige onderdelen

#define BUTTON\_PORT PORTD // Definitie voor welke PORT er gelezen moet worden

#define BUTTON\_PIN PIND // Definitie voor welke PIN er gelezen moet worden

#define BUTTON\_BIT PD7 // Definitie voor welke BIT er gelezen moet worden

int main() // Hoofd functie

{

DDRD = 0b11111111; // Alle poorten van D op output

DDRB = 0b00000111; // B1 en B0 worden op output gezet

BUTTON\_PORT |= \_BV(BUTTON\_BIT); // Zet pd7 op input

PORTB = 0b00000000; // Zet alle poorten van B op laag

int cijferRechts = 0; // int voor het rechter cijfer

int cijferLinks = 0; // int voor het linker cijfer

// Poorten 13 en 14 zijn de common anode

// Poort 4 en 9 zijn de dp en worden niet gebruikt

// PD0 = linksonderin | Pin 1 en 5

// PD1 = Midden | Pin 7 en 17

// PD2 = rechtsonderin | Pin 3 en 8

// PD3 = Beneden | Pin 2 en 6

// PD4 = linksboven | Pin 18 en 12

// PD5 = boven | Pin 16 en 11

// PD6 = rechtsboven | Pin 15 en 10

char cijfer[10]; // Array van 16 plaatsen voor de cijfers

cijfer[0] = 0b00000010; // Nul

cijfer[1] = 0b00111011; // Een

cijfer[2] = 0b00010100; // Twee

cijfer[3] = 0b00010001; // Drie

cijfer[4] = 0b00101001; // Vier

cijfer[5] = 0b01000001; // Vijf

cijfer[6] = 0b01000000; // Zes

cijfer[7] = 0b00011011; // Zeven

cijfer[8] = 0b00000000; // Acht

cijfer[9] = 0b00000001; // Negen

void nummerLinks() // Linker nummer functie

{

PORTB = 0b00000010; // Zet alleen PB1 aan

}

void nummerRechts() // Rechter nummer functie

{

PORTB = 0b00000001; // Zet alleen PB0 aan

}

while(1) // Doe iets voor altijd

{

if (!bit\_is\_clear(BUTTON\_PIN, BUTTON\_BIT)) // Kijk als de bit niet gezet is (de ! betekent niet)

{ // Zo ja

cijferRechts = cijferRechts + 1; // CijferRechts is oud cijferRechts + 1

if(cijferRechts > 9) // Als cijferRechts groter is dan 9

{ // Dan

cijferRechts = 0; // CijferRechts wordt 0

if(cijferLinks < 10) // Als cijferLinks kleiner is dan 10

{ // dan

cijferLinks = cijferLinks + 1; // cijferLinks is oud cijferLinks + 1

}

else // anders

{

cijferLinks = 0; // cijferLinks wordt 0

}

}

}

PORTD = cijfer[cijferLinks]; // Zet de portD poorten op het linker cijfer

nummerLinks(); // Voer functie nummerLinks uit

\_delay\_ms(5); // Wacht 5 Miliseconden

PORTD = cijfer[cijferRechts]; // Zet portD poorten op het rechter cijfer

nummerRechts(); // Voer functie nummerRechts uit

\_delay\_ms(5); // Wacht 5 miliseconden

}

}

### Zonder delay

// De poorten van de 7-segment display heb ik geteld van 0 tot en met 18

// Links onderin is poort 0, rechtsboven is poort 10

#define F\_CPU 1000000 // Snelheid van de CPU, met dit kristal en deze waarde, is het precies 1 seconde

#include <avr/io.h> // Laadt de IO library, deze bevat de poorten

#include <util/delay.h> // // Laadt de Delay library, nu kan je een delay instellen

#include <stdio.h> // Laadt de stdio library, deze vat de meeste nodige onderdelen

#define BUTTON\_PORT PORTD // Definitie voor welke PORT er gelezen moet worden

#define BUTTON\_PIN PIND // Definitie voor welke PIN er gelezen moet worden

#define BUTTON\_BIT PD7 // Definitie voor welke BIT er gelezen moet worden

int main() // Hoofd functie

{

DDRD = 0b11111111; // Alle poorten van D op output

DDRB = 0b00000111; // B1 en B0 worden op output gezet

BUTTON\_PORT |= \_BV(BUTTON\_BIT); // Zet pd7 op input

PORTB = 0b00000000; // Zet alle poorten van B op laag

int cijferRechts = 0; // int voor het rechter cijfer

int cijferLinks = 0; // int voor het linker cijfer

// Poorten 13 en 14 zijn de common anode

// Poort 4 en 9 zijn de dp en worden niet gebruikt

// PD0 = linksonderin | Pin 1 en 5

// PD1 = Midden | Pin 7 en 17

// PD2 = rechtsonderin | Pin 3 en 8

// PD3 = Beneden | Pin 2 en 6

// PD4 = linksboven | Pin 18 en 12

// PD5 = boven | Pin 16 en 11

// PD6 = rechtsboven | Pin 15 en 10

char cijfer[10]; // Array van 16 plaatsen voor de cijfers

cijfer[0] = 0b00000010; // Nul

cijfer[1] = 0b00111011; // Een

cijfer[2] = 0b00010100; // Twee

cijfer[3] = 0b00010001; // Drie

cijfer[4] = 0b00101001; // Vier

cijfer[5] = 0b01000001; // Vijf

cijfer[6] = 0b01000000; // Zes

cijfer[7] = 0b00011011; // Zeven

cijfer[8] = 0b00000000; // Acht

cijfer[9] = 0b00000001; // Negen

void nummerLinks() // Linker nummer functie

{

PORTB = 0b00000010; // Zet alleen PB1 aan

}

void nummerRechts() // Rechter nummer functie

{

PORTB = 0b00000001; // Zet alleen PB0 aan

}

while(1) // Doe iets voor altijd

{

if (!bit\_is\_clear(BUTTON\_PIN, BUTTON\_BIT)) // Kijk als de bit niet gezet is (de ! betekent niet)

{ // Zo ja

cijferRechts = cijferRechts + 1; // CijferRechts is oud cijferRechts + 1

if(cijferRechts > 9) // Als cijferRechts groter is dan 9

{ // Dan

cijferRechts = 0; // CijferRechts wordt 0

if(cijferLinks < 10) // Als cijferLinks kleiner is dan 10

{ // dan

cijferLinks = cijferLinks + 1; // cijferLinks is oud cijferLinks + 1

}

else // anders

{

cijferLinks = 0; // cijferLinks wordt 0

}

}

}

PORTD = cijfer[cijferLinks]; // Zet de portD poorten op het linker cijfer

nummerLinks(); // Voer functie nummerLinks uit

PORTD = cijfer[cijferRechts]; // Zet portD poorten op het rechter cijfer

nummerRechts(); // Voer functie nummerRechts uit

}

}

## Potentiometer op 7-segment

#define F\_CPU 1000000 // Snelheid van de CPU, met dit kristal en deze waarde, is het precies 1 seconde

#include <avr/io.h> // Laadt de IO library, deze bevat de poorten

#include <util/delay.h> // // Laadt de Delay library, nu kan je een delay instellen

#include <stdio.h> // Laadt de stdio library, deze vat de meeste nodige onderdelen

int adc\_value; // Variabelen om het resultaat op te slaan

void nummerLinks() // Linker nummer functie

{

PORTB = 0b00000010; // Zet alleen PB1 aan

}

void nummerRechts() // Rechter nummer functie

{

PORTB = 0b00000001; // Zet alleen PB0 aan

}

int main(void) // Hoofd functie

{

DDRD = 0b11111111; // Alle poorten van D op output

DDRB = 0b00000011; // B1, B2, B3 worden op output gezet

PORTB = 0b00000000; // Zet alle poorten van B op laag

// Poorten 13 en 14 zijn de common anode

// Poort 4 en 9 zijn de dp en worden niet gebruikt

// PD0 = linksonderin | Pin 1 en 5

// PD1 = Midden | Pin 7 en 17

// PD2 = rechtsonderin | Pin 3 en 8

// PD3 = Beneden | Pin 2 en 6

// PD4 = linksboven | Pin 18 en 12

// PD5 = boven | Pin 16 en 11

// PD6 = rechtsboven | Pin 15 en 10

char cijfer[16]; // Array van 16 plaatsen voor de cijfers en letters

cijfer[0] = 0b00000010; // Nul

cijfer[1] = 0b00111011; // Een

cijfer[2] = 0b00010100; // Twee

cijfer[3] = 0b00010001; // Drie

cijfer[4] = 0b00101001; // Vier

cijfer[5] = 0b01000001; // Vijf

cijfer[6] = 0b01000000; // Zes

cijfer[7] = 0b00011011; // Zeven

cijfer[8] = 0b00000000; // Acht

cijfer[9] = 0b00000001; // Negen

cijfer[10] = 0b00001000; // A

cijfer[11] = 0b01100000; // B

cijfer[12] = 0b01000110; // C

cijfer[13] = 0b00110000; // D

cijfer[14] = 0b01000100; // E

cijfer[15] = 0b01001100; // F

int teller;

ADCSRA |= ((1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)); // Prescaler is 128, dus een 125 kHz

ADMUX |= (1<<REFS0); // Zeg dat hij 5 volt moet meten

ADMUX &= ~(1<<REFS1); // Zet de maximum op 5 volt

ADCSRB &= ~((1<<ADTS2)|(1<<ADTS1)|(1<<ADTS0)); // Zet de ADC in free-running mode

ADCSRA |= (1<<ADATE); // Signaal bron om te meten

ADCSRA |= (1<<ADEN); // Start de ADC

ADCSRA |= (1<<ADSC); //Start de conversie

while(1) // doet iets voor altijd

{

adc\_value = ADCW; // Lees de ADC

if(adc\_value >= 0 && adc\_value < 127) // 1 bit / 1 decimaal

{ // 1 hexidecimaal

for(teller = 0; teller < 100; teller++) // Voor teller is kleiner dan 100, en elke keer als programma hier komt, teller + 1

{

PORTD = cijfer[0]; // Zet de portD poorten op 0

nummerLinks(); // Voer functie nummerLinks uit

\_delay\_ms(1); // Wacht 1 ms

PORTD = cijfer[1]; // Zet portD poorten op 1

nummerRechts(); // Voer functie nummerRechts uit

\_delay\_ms(1); // Wacht 1 ms

}

}

else if(adc\_value >= 127 && adc\_value < 254) // 2 bit / 2 decimaal

{ // 2 hexidecimaal

for(teller = 0; teller < 100; teller++) // Voor teller is kleiner dan 100, en elke keer als programma hier komt, teller + 1

{

PORTD = cijfer[0]; // Zet de portD poorten op 0

nummerLinks(); // Voer functie nummerLinks uit

\_delay\_ms(1); // Wacht 1 ms

PORTD = cijfer[2]; // Zet portD poorten op 1

nummerRechts(); // Voer functie nummerRechts uit

\_delay\_ms(1); // Wacht 1 ms

}

}

else if(adc\_value >= 254 && adc\_value < 381) // 3 bit / 4 decimaal

{ // 4 hexidecimaal

for(teller = 0; teller < 100; teller++) // Voor teller is kleiner dan 100, en elke keer als programma hier komt, teller + 1

{

PORTD = cijfer[0]; // Zet de portD poorten op 0

nummerLinks(); // Voer functie nummerLinks uit

\_delay\_ms(1); // Wacht 1 ms

PORTD = cijfer[4]; // Zet portD poorten op 1

nummerRechts(); // Voer functie nummerRechts uit

\_delay\_ms(1); // Wacht 1 ms

}

}

else if(adc\_value >= 381 && adc\_value < 508) // 4 bit / 8 decimaal

{ // 8 hexidecimaal

for(teller = 0; teller < 100; teller++) // Voor teller is kleiner dan 100, en elke keer als programma hier komt, teller + 1

{

PORTD = cijfer[0]; // Zet de portD poorten op 0

nummerLinks(); // Voer functie nummerLinks uit

\_delay\_ms(1); // Wacht 1 ms

PORTD = cijfer[8]; // Zet portD poorten op 1

nummerRechts(); // Voer functie nummerRechts uit

\_delay\_ms(1); // Wacht 1 ms

}

}

else if(adc\_value >= 508 && adc\_value < 635) // 5 bit / 16 decimaal

{ // 10 hexidecimaal

for(teller = 0; teller < 100; teller++) // Voor teller is kleiner dan 100, en elke keer als programma hier komt, teller + 1

{

PORTD = cijfer[1]; // Zet de portD poorten op 0

nummerLinks(); // Voer functie nummerLinks uit

\_delay\_ms(1); // Wacht 1 ms

PORTD = cijfer[0]; // Zet portD poorten op 1

nummerRechts(); // Voer functie nummerRechts uit

\_delay\_ms(1); // Wacht 1 ms

}

}

else if(adc\_value >= 635 && adc\_value < 762) // 6 bit / 32 decimaal

{ // 20 hexidecimaal

for(teller = 0; teller < 100; teller++) // Voor teller is kleiner dan 100, en elke keer als programma hier komt, teller + 1

{

PORTD = cijfer[2]; // Zet de portD poorten op 0

nummerLinks(); // Voer functie nummerLinks uit

\_delay\_ms(1); // Wacht 1 ms

PORTD = cijfer[0]; // Zet portD poorten op 1

nummerRechts(); // Voer functie nummerRechts uit

\_delay\_ms(1); // Wacht 1 ms

}

}

else if(adc\_value >= 762 && adc\_value < 889) // 7 bit / 64 decimaal

{ // 40 hexidecimaal

for(teller = 0; teller < 100; teller++) // Voor teller is kleiner dan 100, en elke keer als programma hier komt, teller + 1

{

PORTD = cijfer[4]; // Zet de portD poorten op 0

nummerLinks(); // Voer functie nummerLinks uit

\_delay\_ms(1); // Wacht 1 ms

PORTD = cijfer[0]; // Zet portD poorten op 1

nummerRechts(); // Voer functie nummerRechts uit

\_delay\_ms(1); // Wacht 1 ms

}

}

else if(adc\_value >= 889 && adc\_value < 1016) // 8 bit / 128 decimaal

{ // 80 hexidecimaal

for(teller = 0; teller < 100; teller++) // Voor teller is kleiner dan 100, en elke keer als programma hier komt, teller + 1

{

PORTD = cijfer[8]; // Zet de portD poorten op 0

nummerLinks(); // Voer functie nummerLinks uit

\_delay\_ms(1); // Wacht 1 ms

PORTD = cijfer[0]; // Zet portD poorten op 1

nummerRechts(); // Voer functie nummerRechts uit

\_delay\_ms(1); // Wacht 1 ms

}

}

else // Error catching, zet FF op de display (waardes tussen de 1016 en 1023)

{

for(teller = 0; teller < 100; teller++) // Voor teller is kleiner dan 100, en elke keer als programma hier komt, teller + 1

{

PORTD = cijfer[15]; // Zet de portD poorten op 0

nummerLinks(); // Voer functie nummerLinks uit

\_delay\_ms(1); // Wacht 1 ms

PORTD = cijfer[15]; // Zet portD poorten op 1

nummerRechts(); // Voer functie nummerRechts uit

\_delay\_ms(1); // Wacht 1 ms

}

}

}

return 0; // geef 0 terug

}

## Bronnen

De datasheet van de atmega uitgelegd:

<http://maxembedded.com/2011/06/20/the-adc-of-the-avr/>

Sheets van ELO

Lezen over bit\_is\_clear methode

<http://www.avrfreaks.net/index.php?name=PNphpBB2&file=printview&t=27233&start=0>

# Week 4

## PWM ISR

## RGB met Pot en knop

# Eindopdracht

# Bijlage 1: Makefile

# Name: Makefile

# Author: Duncan Pronk and Michael Holachek

# Copyright: 2014, editable by everyone

# License: This work fall under the Creative Common License

# DEVICE ....... The AVR device you compile for

# CLOCK ........ Target AVR clock rate in Hertz

# OBJECTS ...... The object files created from your source files. This list is

# usually the same as the list of source files with suffix ".o".

# PROGRAMMER ... Options to avrdude which define the hardware you use for

# uploading to the AVR and the interface where this hardware

# is connected.

# FUSES ........ Parameters for avrdude to flash the fuses appropriately. <http://www.engbedded.com/fusecalc>

DEVICE = atmega88pa

CLOCK = 8000000

PROGRAMMER = -c arduino -P com5 -b 19200

OBJECTS = main.o

FUSES = -U lfuse:w:0x62:m -U hfuse:w:0xdf:m -U efuse:w:0xf9:m

######################################################################

######################################################################

# Tune the lines below only if you know what you are doing:

AVRDUDE = avrdude $(PROGRAMMER) -p $(DEVICE)

COMPILE = avr-gcc -Wall -Os -DF\_CPU=$(CLOCK) -mmcu=$(DEVICE)

# symbolic targets:

all: main.hex

.c.o:

$(COMPILE) -c $< -o $@

.S.o:

$(COMPILE) -x assembler-with-cpp -c $< -o $@

# "-x assembler-with-cpp" should not be necessary since this is the default

# file type for the .S (with capital S) extension. However, upper case

# characters are not always preserved on Windows. To ensure WinAVR

# compatibility define the file type manually.

.c.s:

$(COMPILE) -S $< -o $@

flash: all

$(AVRDUDE) -U flash:w:main.hex:i

fuse:

$(AVRDUDE) $(FUSES)

install: flash fuse

# if you use a bootloader, change the command below appropriately:

load: all

bootloadHID main.hex

clean:

rm -f main.hex main.elf $(OBJECTS)

# file targets:

main.elf: $(OBJECTS)

$(COMPILE) -o main.elf $(OBJECTS)

main.hex: main.elf

rm -f main.hex

avr-objcopy -j .text -j .data -O ihex main.elf main.hex

# If you have an EEPROM section, you must also create a hex file for the

# EEPROM and add it to the "flash" target.

# Targets for code debugging and analysis:

disasm: main.elf

avr-objdump -d main.elf

cpp:

$(COMPILE) -E main.c