Microcontroller Programmeren in C

MICPRG Les1

DE HAAGSE HOGESCHOOL

Werkvormen MICPRG

- MICPRG-co1 + MICPRG-pr1 = **84** SBU.
  - 14 uur theorie.
  - 14 uur practicum.
  - 14 uur toets + voorbereiding
  - 42 uur zelfstudie = **6 uur/week zelfstudie!**
- Toets:
  - Theorie:
    - Schriftelijke toets met open vragen beoordeeld met cijfer.
    - Je mag gebruik maken van al het studiematriaal + eigen aantekeningen.
  - Practicum:
    - 6 opdrachten beoordeeld met Voldoende/Onvoldoende.
    - Aanwezigheid verplicht.

DE HAAGSE HOGESCHOOL 2

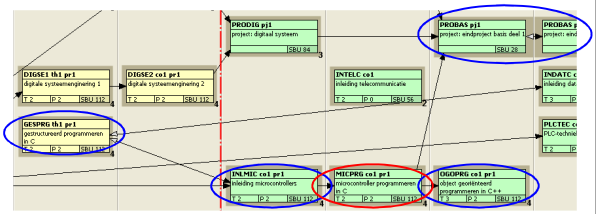
Inhoud

- Toepassingen van microcontrollers.
- Embedded software voor microcontrollers in **C**.
- Uitsturen en inlezen van digitale signalen (parallelle I/O).
- Het maken van een tijdvertraging (timer).
- Na een bepaalde tijd herhalen van bepaalde acties (timer).
- Interrupts in C.
- Inlezen van analoge signalen (ADC).
- Tellen van pulsen (counter).
- Meten van pulsduur (input capture timer).
- Opwekken van pulsen (output compare en PWM timer).
- Seriële communicatie (UART).
- Datastructuren (`struct`), textfiles, datum en tijd in **C**.
- Gebruik van Flash en EEPROM.
- Duurzaam gebruik van een microcontroller (sleep modes).

DE HAAGSE HOGESCHOOL 3

Plaats in curriculum

- Bouwt verder op GESPRG en INLMIC.
- Voorbereiding voor OGOPRG.
- Wordt toegepast bij PROBAS (eindproject Basis).

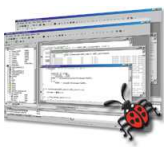


http://bd.eduweb.hhs.nl/semboek/2012/index.htm
http://bd.eduweb.hhs.nl/semboek/2012/duaal.htm

DE HAAGSE HOGESCHOOL 4

Leermiddelen

- Blackboard MICPRG.
- http://bd.eduweb.hhs.nl/micprg:
  - Sheets, handouts.
  - Studiewijzer.
  - Practicumopdrachten.
  - Stukken theorie.
- Boeken:
  - AVR - An Introductory Course van John Morton
  - De programmeertaal C, 4de vernieuwde editie van Al Kelley en Ira Pohl.
- Ontwikkelomgeving:
  - AVR Studio 4 met WinAVR plugin en simulator plugin.
  - STK 500 bord met JTAGICE mkII of Dragon.

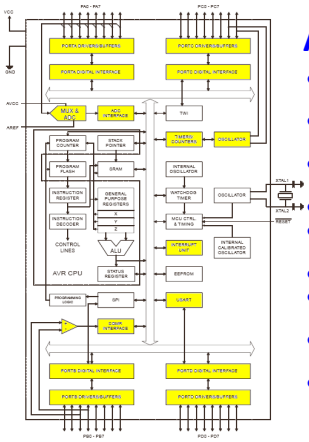


DE HAAGSE HOGESCHOOL 5

Microcontroller toepassingen

- Huis, tuin en keukenproducten:
  - Magnetron, broodbakmachine, video, DVD speler, speelgoed, CV ketel enz...
- Medische apparatuur:
  - MRI scanner, pacemaker, digitale thermometer, enz...
- Computer apparatuur:
  - DVD drive, printer, modem enz...
- Land- en tuinbouw:
  - Klimaatbeheersing, sorteermachine, weegschaal, koeherkennings- systeem enz...
- Auto:
  - Motor management systeem, ABS, airbag, radio, route informatie- systeem enz...
- Verkeer:
  - Stoplicht, overwegbeveiliging, flitspaal, enz...
- Energietechniek:
  - Smart energy meter, smart grid, motorcontroller, enz...
- ...

DE HAAGSE HOGESCHOOL 6



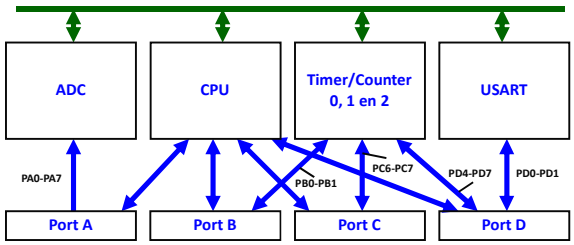
ATmega32

- Uitsturen en inlezen van digitale poorten.
- Het maken van een tijdvertraging (timer).
- Na een bepaalde tijd herhalen van bepaalde acties (timer).
- Interrupts in C.
- Inlezen van analoge signalen (ADC).
- Tellen van pulsen (counter).
- Meten van pulsduur (input capture timer).
- Opwekken van pulsen (output compare en PWM timer).
- Seriële communicatie (UART).

DE HAAGSE HOGESCHOOL 7

ATmega32 Blok diagram

- Belangrijkste onderdelen die behandeld worden:



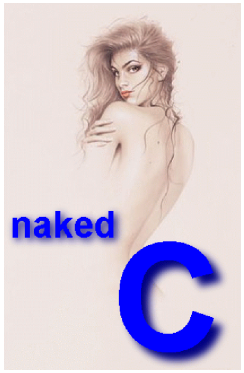
DE HAAGSE HOGESCHOOL 8

Voordelen C t.o.v. ASM

- Eenvoudiger te lezen en te begrijpen. Daardoor beter aan te passen en eenvoudiger uit te breiden.
- C programma's voor de AVR kunnen eenvoudiger worden omgezet naar een C programma voor een andere microcontroller. Zeker als de microcontroller specifieke delen van het programma zijn "verborgen" in een aantal specifieke functies.

DE HAAGSE HOGESCHOOL 9

MICPRG = Naked C



- Geen operating systeem en geen run-time environment. (B.v. geen stdio.h.)
- Beperkte library AVR-libc.
- Beperkt datageheugen 2K Bytes RAM.
- Beperkt programma-geheugen 32K Bytes Flash.

DE HAAGSE HOGESCHOOL 10

Naked C Voorbeeld

DE HAAGSE HOGESCHOOL 11

Programma

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

void wait(void) {
    uint8_t i;
    for (i = 0; i < 10; ++i)
        _delay_ms(25);
}
int main(void) {
    void wait(void);
    uint8_t c1, c2, i;
    DDRB = 0xFF;
    while (1) {
        c1 = 0x80;
        c2 = 0x01;
        for (i = 0; i < 4; i++) {
            wait();
            PORTB = ~(c1 | c2);
            c1 >>= 1;
            c2 <<= 1;
        }
    }
    return 0;
}
```

- Bitn… voor beginners:
  - Bitje veranderen:
    - Bitje setten.
    - Bitje clearen.
    - Bitje flippen.
  - Meerdere bitjes veranderen.
  - Bitje testen:
    - Is het bitje 1?
    - Is het bitje 0?
  - Meerdere bitjes testen.
  - Schuiven met bitjes.
  - Maskers en patronen samenstellen door een 1 naar links te schuiven.

DE HAAGSE HOGESCHOOL 12

Microcontroller Programmeren in C

MICPRG Les 2

DE HAAGSE HOGESCHOOL

Even wachten…

- in een programma voor de AVR:
  - Maak een lusje dat verder niets doet:

```
void wait(void) {
    volatile int i;
    for (i = 0; i < 30000; ++i)
        /*empty*/;
}
```

Denk aan **volatile**!

  - Gebruik een library routine:

```
#include <util/delay.h>

void wait(void) {
    uint8_t i;
    for (i = 0; i < 10; ++i)
        _delay_ms(25);
}
```

Denk aan beperkingen!

The maximal possible delay is 262.14 ms / F_CPU in MHz.

  - Gebruik een hardware **Timer/Counter**.

DE HAAGSE HOGESCHOOL 14

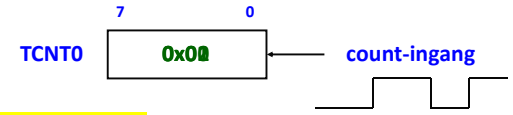
Timer/Counter

- De ATmega32 heeft 3 Timer/Counters:
  - 8 bits Timer/Counter**0**.
  - 16 bits Timer/Counter**1**.
  - 8 bits Timer/Counter**2**.
- Toepassingen:
  - Tellen van pulsen (counter).
  - Opwekken van pulsen (output compare en PWM).
  - Meten van pulsduur (input capture).
  - **Bepaalde tijd wachten.**

DE HAAGSE HOGESCHOOL 15

Timer/Counter0

- Normal mode. Voor eenvoudig gebruik.
- Timer/Counter0 is niets anders dan een 8 bits I/O register TCNT0 (Timer CouNT 0) waarvan de inhoud bij elke actieve flank van de count-ingang met 1 wordt verhoogd.



Je kunt het TCNT0 register uitlezen en beschrijven!

DE HAAGSE HOGESCHOOL 16

Timer/Counter0

- Welke signalen kun je als ingang gebruiken?
  - Extern signaal aangesloten op pin PB0 = T0.
  - Kloksignaal van de AVR / prescaler (= 1, 8, 64, 256 of 1024).
  - Instellingen in I/O register TCCR0 (Timer/Counter Control Register 0)

7	6	5	4	3	2	1	0	
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Mode	WGM01	WGM00	Timer/Counter Mode of Operation
0	0	0	Normal
1	0	1	PWM, Phase Correct
2	1	0	CTC
3	1	1	Fast PWM

DE HAAGSE HOGESCHOOL 17

Timer/Counter0

- Welke signalen kun je als ingang gebruiken?
  - Instellingen in TCCR0 (Timer/Counter Control Register 0)

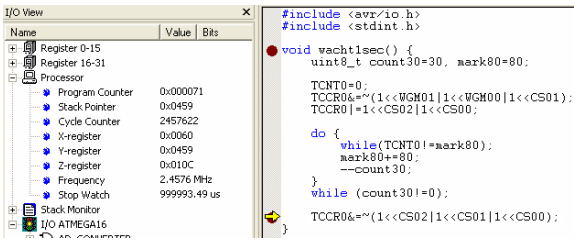
CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

DE HAAGSE HOGESCHOOL 18

Voorbeeld Timer/Counter0

- Zie AVR boek pagina 44 t/m 46.
- 1 seconde wachten bij F_{CLK} = 2.4576 MHz.
  - Gebruik prescaler van 1024 (waarom?)
  - $F_{Count\ ingang}$ = 2457600/1024=2400 Hz.
  - Dus 1 seconde = wachten tot timer/counter0 tot 2400 heeft geteld. Maar…
  - TCNT0 is maar **8** bits (max 255).
  - Oplossingen:
    - Gebruik Timer/Counter1 (16 bits).
    - Tel 30x tot 80.

1 seconde wachten:



- Waarom niet exact 1 seconde?
- Alternatieve oplossingen? **= Huiswerk!**

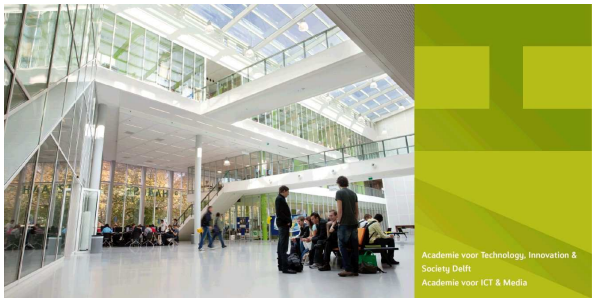
Alternatief (uitwerking huiswerk)

```
void wacht1sec() {
    uint8_t i;

    TCCR0 &= ~(1<<WGM01 | 1<<WGM00 | 1<<CS01);
    TCCR0 |= 1<<CS02 | 1<<CS00;

    for (i = 0; i < 10; i++) {
        TCNT0 = 0;
        while (TCNT0 != 240);
    }

    TCCR0 &= ~(1<<CS02 | 1<<CS01 | 1<<CS00);
}
```

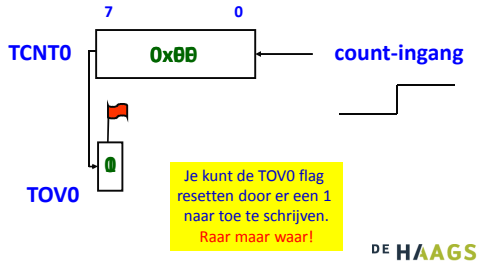
Microcontroller Programmeren in C

MICPRG Les 3

DE HAAGSE HOGESCHOOL

Timer/Counter0 overflow

- Als de timer overloopt dan wordt de TOV0 (Timer OVerflow 0) flag in I/O register TIFR (Timer Interrupt Flag Register) geset.



Huiswerk!

7	6	5	4	3	2	1	0	
OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

- Pas de functie wacht1sec aan zodat deze functie gebruik maakt van de TOV0 flag.
  - 2400 = 96 + 9 x 256
  - Laad TCNT0 met 256 – 96 = 160 en wacht tot TOV0 10x geset is.
  - Vergeet niet om TOV0 steeds te resetten voordat je gaat wachten (door er een 1 naar toe te schrijven).
  - Het nut van deze methode zal hierna pas blijken…

Alternatief (uitwerking huiswerk)

```
void wacht1sec() {
    uint8_t i;

    TCNT0 = 160;
    TCCR0 &= ~(1<<WGM01 | 1<<WGM00 | 1<<CS01);
    TCCR0 |= 1<<CS02 | 1<<CS00;

    for (i = 0; i < 10; i++) {
        TIFR = 1<<TOV0;
        while (!(TIFR & 1<<TOV0));
    }

    TCCR0 &= ~(1<<CS02 | 1<<CS01 | 1<<CS00);
}
```

- Waarom `TIFR = 1<<TOV0` en niet `TIFR |= 1<<TOV0` ?

DE HAAGSE HOGESCHOOL 25

LED7 laten knipperen (1 sec)

```
int main(void) {
    DDRB = 0xFF;
    PORTB = 0xFF; /* alle LEDs uit */
    while (1) {
        wacht1sec();
        PORTB ^= 1<<7; /* flip LED7 */
    }
    return 0;
}
```

- **Alle** implementaties van `wacht1sec` gebruiken busy waiting (spinning). Dat geeft problemen als je tijdens het wachten iets anders wilt doen.
- Bijv: LED7 moet knipperen en LED6 moet meteen gaan branden als SW6 ingedrukt wordt.
  - Kan alleen door `wacht1sec` aan te passen… maar dat is niet handig.

DE HAAGSE HOGESCHOOL 26

Interrupts in C

- Onderbreking van "normale" programma.
- Verschillende redenen:
  - Karakter ontvangen via seriële poort
  - Timer die overloopt
  - Bepaald ingangssignaal veranderd
  - Enz…
- Bij optreden interrupt:
  - Maak huidige machinecode instructie af.
  - Blokeer andere interrupts door I bit in SREG 0 te maken.
  - Reset de Flag die interrupt veroorzaakt heeft.
  - Spring naar een bij de interrupt behorende interrupt service routine (ISR) via vectortabel (compiler genereert deze code).
  - Save alle gebruikte registers op de stack (compiler genereert deze code).
- Bij einde ISR:
  - Restore registers (haal registers van de stack) (compiler genereert deze code).
  - RETI instructie (maakt I bit weer 1 = geef andere interrupts vrij).
  - Onderbroken programma gaat verder.

DE HAAGSE HOGESCHOOL 27

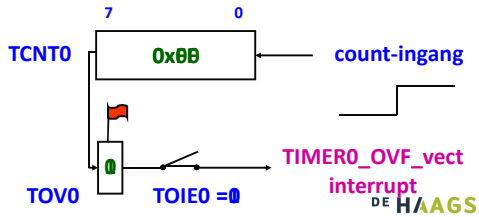
Interrupts

- Tijdens ISR wordt niet op andere interrupts gereageerd.
  - ISR moet dus snel zijn.
- Interrupt vector bevat JMP naar begin van ISR.
- ATmega32 heeft 21 interrupt vectoren (Flash 0x000 t/m 0x029).
  - ADC_vect, ANA_COMP_vect,
  - EE_RDY_vect,
  - INT0_vect, INT1_vect, INT2_vect,
  - SPI_STC_vect, SPM_RDY_vect,
  - TIMER0_COMP_vect, TIMER0_OVF_vect, TIMER1_CAPT_vect, TIMER1_COMPA_vect, TIMER1_COMPB_vect, TIMER1_OVF_vect, TIMER2_COMP_vect, TIMER2_OVF_vect,
  - TWI_vect,
  - USART_RXC_vect, USART_TXC_vect, USART_UDRE_vect

DE HAAGSE HOGESCHOOL 28

T/C0 overflow interrupt

- T/C0 kan een interrupt genereren als TCNT0 overloopt.
- Deze interrupt kun je aanzetten met het TOIE0 bit (Timer/Counter Overflow Interrupt Enable 0) in het TIMSK register (Timer MaSK)



DE HAAGSE HOGESCHOOL 29

Interrupts in C (gcc)

- avr/interrupt.h
  - #define waarmee ISR gedefinieerd kan worden:
    `ISR(naam_van_vect) { … }`
  - Functies waarmee het I bit in het SREG geset of gereset kan worden:
    `sei()` en `cli()`

DE HAAGSE HOGESCHOOL 30

TOV0 interrupt

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdint.h>

uint8_t i=0;

ISR(TIMER0_OVF_vect) {
    ++i;
    if (i==10) {
        i=0;
        TCNT0=160;
        PORTB^=1<<7;
    }
}

int main() {
    DDRB=0xFF;
    PORTB=0xFF; /* alle LEDs uit */
    DDRA=0x00;

    TCNT0=160;
    TCCR0&=~(1<<WGM01|1<<WGM00|1<<CS01);
    TCCR0|=1<<CS02|1<<CS00;
    TIMSK|=1<<TOIE0;

    sei();

    while(1) {
        if (PINA&1<<6)
            PORTB|=1<<6;
        else
            PORTB&=~(1<<6);
    }

    return 0;
}
```

TIMER0_OVF_vect interrupt

Microcontroller Programmeren in C

MICPRG Les 4

DE HAAGSE HOGESCHOOL

Globale variabele

- We gebruiken liever geen globale variabele.
  - Waarom eigenlijk niet?
- De variabele `i` in de ISR kan niet lokaal zijn.
  - Waarom eigenlijk niet?
- Oplossing: **static** lokaal (variabele wordt maar 1x aangemaakt en blijft daarna bestaan).

```
ISR(TIMER0_OVF_vect) {
    static uint8_t i = 0;
    ++i;
    if (i == 10) {
        i = 0;
        TCNT0 = 160;
        PORTB ^= 1<<7; /* flip LED7 */
    }
}
```

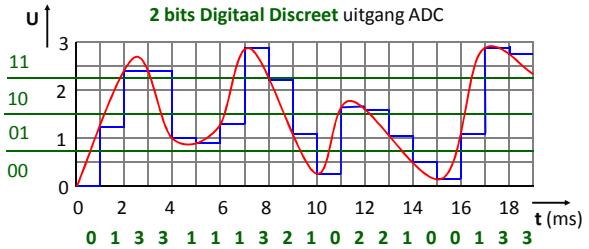
static
Scope (zichtbaarheid) = block (lokaal).
Livetime = tot einde programma.

ADC Analog Digital Converter

- Veel signalen in de buitenwereld zijn analoog en continue.
- Veel meetsensoren leveren een analoge uitgangsspanning:
  - Temperatuursensor
  - Druksensor
  - Lichtsensor
  - Microfoon
  - enz…
- Om deze signalen te kunnen inlezen met een µC (microcontroller) moeten ze discreet en digitaal gemaakt worden.
- Veel µC's hebben een ingebouwde ADC.

Bijv: ADC 0-3V => 2 bits

Analoog Continue afkomstig van sensor
Analoog Discreet na sample (1 ms) en hold
2 bits Digitaal Discreet uitgang ADC



Bijv: ADC 0-3V => 2 bits

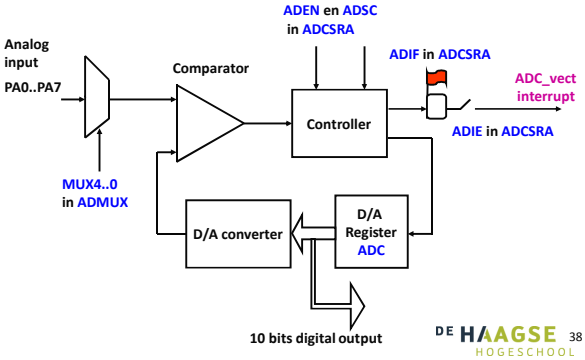
- Omrekenen:
  $U_{in} = U_{max} * (DIG_{out} + ½) / (DIG_{max} + 1)$
- Kwantiseringsfout:
  $±½ LSB = ±½ * U_{max} / (DIG_{max} + 1)$

Gemeten digitale waarde	Omgerekende analoge waarde	Min analoge waarde	Max analoge waarde
0	0.375 ± 0.375	0.00	0.75
1	1.125 ± 0.375	0.75	1.50
2	1.875 ± 0.375	1.50	2.25
3	2,625 ± 0.375	2.25	3.00

ADC

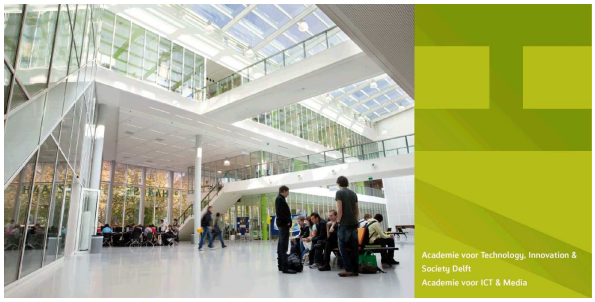
- Sample frequentie:
  - Volgens theorie (bemonsteringstheorema van Nyquist-Shannon) **2x** hoogst voorkomende frequentie in signaal.
    - spraak 4KHz => sample frequentie 8 KHz (telefoon => 8KHz)
    - muziek 20 KHz => sample frequentie 40 KHz (CD => 44,1 KHz)
    - oventemperatuur 10 Hz => sample frequentie 20Hz
    - omgevingstemperatuur 0,01Hz => sample frequentie 0,02 Hz
- maximale sample frequentie is afhankelijk van de conversiesnelheid van de ADC.
  - AVR: f_{ADC} = 50 – 200 KHz. Conversion time (single ended, free running) = **13,5 ADC clocks.** Dus max f_{sample} = **14,8** KHz.
- Signaal bewerkingen voor S&H:
  - Versterken of verzwakken.
  - Verschuiven.
  - Filteren (te hoge frequenties eruit = Anti-aliasing filter).
- Resolutie ADC:
  - AVR: **10 bits**

ADC eenvoudig blokschema



Huiswerk

- Bestudeer:
  - AVR boek blz. 97 t/m 99 (tot program K).
  - AVR boek blz. 109 t/m 112 (tot program M).
- Lees:
  - Hoofdstuk over de ADC in ATmega32A datasheets: http://www.atmel.com/Images/Atmel-8155-8-bit-Microcontroller-AVR-ATmega32A_Datasheet.pdf
    19 pagina's. Dit soort documentatie moet je aan het einde van het 2de jaar zelfstandig kunnen gebruiken!
    (Wordt volgende les uitgelegd.)
- Programmeer:
  - Maak een programma dat de spanning op PA0 op de LCD display laat zien. Maak gebruik van de LCD display library (zie BB)!



Microcontroller Programmeren in C

MICPRG Les 5

DE HAAGSE HOGESCHOOL

ADC ATmega32

- ADC gebruikt relatief veel energie daarom staat de ADC na reset uit. Aanzetten met bit ADEN (ADc ENable) in het ADCSRA (ADc Control and Status Register A) register.
- Alle ingangen van poort A (PA0 t/m PA7) kunnen als single-ended ingang van de ADC gebruikt worden. Meten t.o.v. GND (GrouND) en AREF (Analog REFerence).
- ADC heeft ook differential mode (wordt niet besproken).
- De ADC wordt bestuurd met behulp van het ADMUX (ADc MUltipleXer selection Register), ADCSRA en SFIOR (Special Function IO Register).

ADC ATmega32

- Schrijven van 1 naar ADSC (ADc Start Conversion) bit in ADCSRA start de conversie.
- Flag ADIF (ADc Interrupt Flag) van ADCSRA register wordt 1 als de conversie klaar is en resultaat in ADC register (16 bits) staat. Dit bit kun je resetten door er een 1 naar toe te schrijven.
- Als ADIE (ADc Interrupt Enable) bit in ADCSRA register geset is wordt een ADC_vect interrupt gegeven als ADIF geset wordt. ADIF wordt bij afhandelen van de interrupt automatisch gereset.

ADC ATmega32

- Automatisch starten van conversie (auto trigger).
- Set bit ADATE (ADc Auto Trigger Enable) in ADCSRA en kies een trigger source met de bits ADTS2 (ADc Trigger Source), ADTS1 en ADTS0 in het SFIOR register.
  - ADC Interrupt flag ADIF => Free Running Mode (continue bemonsteren).
  - Timer/Counter0 Overflow TOV0 => Equidistant bemonsteren.
  - Timer/Counter1 Overflow TOV1 => Idem.

ADMUX

7	6	5	4	3	2	1	0	
REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin X
1	0	Reserved X
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin X

Input Channel and Gain Selections

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			

Pas op!

Verkeerde waarde in REFS1 of REFS0 maakt ADC stuk!

ADCSRA

7	6	5	4	3	2	1	0	
ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Weet je het nog?

ADEN = ...
ADSC = ...
ADATE = ...
ADIF = ...
ADIE = ...

- Division Factor = f_{clk} / f_{ADC}
- Voor maximale (10 bits) nauwkeurigheid: 50 KHz ≤ f_{ADC} ≤ 200 KHz.

SFIOR

7	6	5	4	3	2	1	0	
ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	

ADC Auto Trigger Source Selections

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

Opdracht

- Maak een programma dat de spanning op PA0 op de LCD display laat zien. Maak gebruik van de LCD display library (Zie BB)!

```
void lcd_init(void); /* initialize LCD */
void lcd_cls(void); /* clear screen */
void lcd_cursor(bool cursorOn, bool cursorBlinks);

void lcd_home(void); /* place cursor in upper left corner) */
void lcd_goto(uint8_t row, uint8_t column);
uint8_t lcd_get_row(void); /* get cursor row position (0..1) */
uint8_t lcd_get_column(void); /* get cursor column pos (0..39) */

void lcd_putc(char c); /* write character c */
void lcd_puts(char* s); /* write string s */
```


Uitwerking

- Spanning als getal 0-1023.

```
#include <avr/io.h>
#include <stdio.h>
#include "lcd.h"
int main(void) {
    char buffer[5];
    lcd_init();
    lcd_cursor(false, false);
    // division factor fclk / fadc = 3686 / 200 = 18.4 Kies 32
    // fadc = 3686 / 32 = 115 KHz.
    ADMUX = 0;
    ADCSRA = 1<<ADEN | 1<<ADPS2 | 1<<ADPS0;
    while (1) {
        ADCSRA |= 1<<ADSC | 1<<ADIF;
        while (~ADCSRA & 1<<ADIF);
        lcd_home();
        snprintf(buffer, sizeof buffer, "%4d", ADC);
        lcd_puts(buffer);
    }
    return 0;
}
```

Uitleg **snprintf** zie BB!

Uitwerking

- Spanning in Volts.

Zie BB voor gebruik **float** met **snprintf**!

```
#include <avr/io.h>
#include <stdio.h>
#include "lcd.h"
int main(void) {
    char buffer[5];
    lcd_init();
    lcd_cursor(false, false);
    ADMUX = 0;
    ADCSRA = 1<<ADEN | 1<<ADPS2 | 1<<ADPS0;
    while (1) {
        ADCSRA |= 1<<ADSC | 1<<ADIF;
        while (~ADCSRA & 1<<ADIF);
        // 1 stapje = 5 / 1024 = 0.0048828125 V = 4.8828125 mV
        // fout = +/-2.44 mv. Geef antwoord met 2 cijfers achter de punt
        lcd_home();
        snprintf(buffer, sizeof buffer, "%4.2f", 5.0 * (ADC + 0.5) / 1024);
        lcd_puts(buffer);
    }
    return 0;
}
```

DE HAAGSE HOGESCHOOL 49