
# Microcontroller Programmeren in C

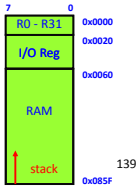
**MICPRG Les 13**

DE HAAGSE HOGESCHOOL

## AVR geheugen

- Werkgeheugen (RAM en registers):
  - Inhoud verdwijnt als spanning wegvalt.
  - Gebruikt voor opslag variabelen.
    - Lokale variabelen op de stack (in RAM) of in registers.
    - Globale en static variabelen op van tevoren gereserveerde plaatsen.
  - Direct benaderbaar vanuit ASM en C programma's.
  - Snel te lezen en te schrijven.
  - Oneindig vaak te beschrijven.

ATmega32:
- 2048 bytes RAM.
- 32 bytes CPU registers.
- 64 bytes I/O registers.

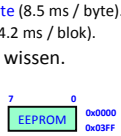


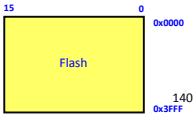

139

## AVR geheugen

- Achtergrondgeheugen (Flash en EEPROM):
  - Inhoud blijft bewaard als spanning wegvalt.
  - Gebruikt voor:
    - Opslag programma en constanten (Flash).
    - Opslag fabrieksinstellingen (Flash).
    - Opslag voorkeursinstellingen (EEPROM).
    - Opslag belangrijke variabelen (EEPROM).
  - Niet direct benaderbaar vanuit C programma's.
  - Moet voor schrijven eerst gewist worden. Dit is traag.
    - EEPROM wissen per byte (8.5 ms / byte).
    - Flash wissen per blok (4.2 ms / blok).
  - Beperkt aantal keer te wissen.
    - Flash 10000x
    - EEPROM 100000x

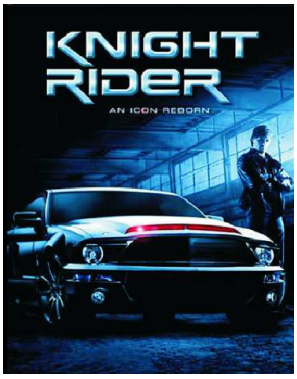
ATmega32:
- 16384 x 16 bits Flash.
  256 blokken
- 1024 bytes EEPROM.




140

## Knight Rider Pro ;-)




- Heen en weer gaand patroon (eenvoudig in de code aan te passen).
- Instelbare snelheid.
- Ingestelde snelheid onthouden bij spanningsuitval.
- Minimaal RAM gebruik.

DE HAAGSE HOGESCHOOL 142

## Knight Rider Pro 1

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdint.h>
#include <stdbool.h>
int main(void) {
    uint8_t delay = 2;
    bool swSnel, swTraag, oldswSnel = false, oldswTraag = false;
    volatile uint16_t i;
    DDRB = 0xFF; PORTB = 0xFF; DDRA = 0x00;
    OCR0 = 36 * delay; TCCR0 = 0x0D;
    TIMSK |= 1<<OCIE0; sei();
    while (1) {
        swTraag = ~PINA & 0x08; /* SW3 */
        if (swTraag != oldswTraag) {
            oldswTraag = swTraag;
            if (swTraag && delay < 5) OCR0 = ++delay * 36;
        }
        swSnel = ~PINA & 0x04; /* SW2 */
        if (swSnel != oldswSnel) {
            oldswSnel = swSnel;
            if (swSnel && delay > 1) OCR0 = --delay * 36;
        }
        for (i = 0; i < 3000; ++i); /* anti-dender vertraging */
    }
    return 0;
}
```

DE HAAGSE HOGESCHOOL 143

## Knight Rider Pro 1

```
ISR(TIMER0_COMP_vect) {
    static uint8_t patroon[] = {
        0x00, 0x80, 0xC0, 0x40, 0x60, 0x20, 0x30, 0x10,
        0x18, 0x08, 0x0C, 0x04, 0x06, 0x02, 0x03, 0x01, 0x00
    };
    static uint8_t j = 0;
    static bool heen = true;
    PORTB = ~patroon[j];
    heen = heen ? ++j != sizeof patroon - 1 : --j == 0;
}
```

Geheugengebruik -O0:
- 548 bytes Flash.
- 19 bytes RAM +
- 7 bytes op stack in main +
- 12 bytes op stack in ISR.

Alle static variabelen

Alle lokale variabelen

Zie C boek voor uitleg operator ? :

Geheugengebruik -O2:
- 444 bytes Flash.
- 19 bytes RAM +
- 2 bytes op stack in main +
- 10 bytes op stack in ISR.

Alle niet volatile lokale variabelen zijn in registers gezet

DE HAAGSE HOGESCHOOL 144

## Verbeterpunten

- De waarde van de variabele delay moet behouden blijven als de spanning wegvalt. Dit kan door deze variabele in EEPROM op te slaan.
- De array patroon staat in het Flash en wordt naar RAM gekopieerd. Dit zou niet nodig zijn als we Flash kunnen uitlezen.



## EEPROM in C

- Include file: avr/eeprom.h.
- Variabele die in EEPROM moet worden geplaatst moet gemarkeerd worden met EEMEM.
- Functies om in EEPROM te schrijven:
  - `void eeprom_write_byte(const uint8_t* EEMEM, uint8_t)`
  - `void eeprom_write_word(const uint16_t* EEMEM, uint16_t)`
  - `void eeprom_write_block(const void*, void* EEMEM, size_t)`
- Functies om uit EEPROM te lezen:
  - `uint8_t eeprom_read_byte(const uint8_t* EEMEM)`
  - `uint16_t eeprom_read_word(const uint16_t* EEMEM)`
  - `void eeprom_read_block(void*, const void* EEMEM, size_t)`



## Knight Rider Pro 2

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/eeprom.h>
#include <avr/pgmspace.h>
#include <stdint.h>
#include <stdbool.h>
int main(void) {
    uint8_t delay EEMEM;
    bool swSnel, swTraag, oldswSnel = false, oldswTraag = false;
    volatile uint16_t i;
    if (eeprom_read_byte(&delay) < 1 || eeprom_read_byte(&delay) > 5) {
        eeprom_write_byte(&delay, 2);
    }
    DDRB = 0xFF; PORTB = 0xFF; DDRA = 0x00;
    OCR0 = 36 * eeprom_read_byte(&delay); TCCR0 = 0x0D;
    TIMSK |= 1<<OCIE0; sei();
    while (1) {
        swTraag = ~PINA & 0x08; /* SW3 */
        if (swTraag != oldswTraag) {
            oldswTraag = swTraag;
            if (swTraag && eeprom_read_byte(&delay) < 5) {
                eeprom_write_byte(&delay, eeprom_read_byte(&delay) + 1);
                OCR0 = eeprom_read_byte(&delay) * 36;
            }
```

Nodig voor lezen program space (zie verderop)

Rest van main op dezelfde wijze aanpassen.



## Flash

- Include file: avr/pgmspace.h.
- Data die niet naar RAM moet worden gekopieerd moet gemarkeerd worden met PROGMEM.
- Functies om Flash uit te lezen:
  - `uint8_t pgm_read_byte(const uint8_t* PROGMEM)`
  - `uint16_t pgm_read_word(const uint16_t* PROGMEM)`
  - `uint32_t pgm_read_dword(const uint32_t* PROGMEM)`
- C-string constante die niet naar RAM moet worden gekopieerd: `PSTR("Hallo")`.
- Functies om C-string constante te gebruiken:
  - `strcpy_P` enz.



## Flash

- LCD library heeft speciale functie voor het schrijven van een C-string uit Flash:
  - `lcd_puts_P(PSTR("Hallo"));`
- AVR stdio library heeft speciale functies voor het gebruik met C-strings uit Flash:
  - `snprintf_P(buffer, sizeof buffer, PSTR("i = %6d"), i);`



## Knight Rider Pro 2

```
ISR(TIMER0_COMP_vect) {
    static uint8_t patroon[] PROGMEM = {
        0x00, 0x80, 0xC0, 0x40, 0x60, 0x20, 0x30, 0x10,
        0x18, 0x08, 0x0C, 0x04, 0x06, 0x02, 0x03, 0x01, 0x00
    };
    static uint8_t j = 0;
    static bool heen = true;
    PORTB = ~pgm_read_byte(&patroon[j]);
    heen = heen ? ++j != sizeof patroon - 1 : --j == 0;
}
```

Geheugengebruik –O0:
- 876 bytes Flash.
- 3 bytes RAM +
- 16 bytes op stack in main +
- 15 bytes op stack in ISR.

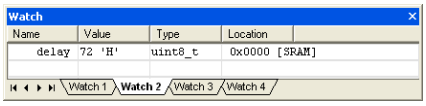
Alle static variabelen in RAM + ?? (avr-libc)

Geheugengebruik –O2:
- 574 bytes Flash.
- 3 bytes RAM +
- 2 bytes op stack in main +
- 10 bytes op stack in ISR.

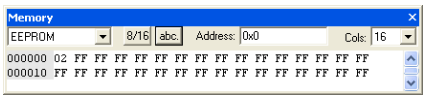
Alle niet volatile lokale variabelen zijn in registers gezet

## Flash een EEPROM debug

- AVR Studio Watch window werkt niet correct met Flash en EEPROM.




- Je moet een Memory window gebruiken om Flash en EEPROM inhoud te zien.







## Microcontroller Programmeren in C

**MICPRG Les 14**

DE HAAGSE HOGESCHOOL

## Duurzaam AVR gebruik

- De AVR ATmega32A is de opvolger van de ATmega32. De A versie is ongeveer 40% zuiniger.
- De AVR ATmega32 heeft 6 slaapstanden waarin de CPU en bepaalde andere delen uitgeschakeld zijn.
- De AVR wordt weer wakker door een interrupt of door een Reset. `sei()` niet vergeten voordat je gaat slapen.
- Assembler: gebruik de machinecode instructie SLEEP en het MCUCR I/O register.
- C: gebruik avr/sleep.h.
  - `void set_sleep_mode(uint8_t mode);`
  - `void sleep_mode(void);`




## Slaapstanden

| mode | CPU | I/O | ADC | T/C2 (async) | System clock |
|---|---|---|---|---|---|
| SLEEP_MODE_IDLE | ✘ | ✔ | ✔ | ✔ | ✔ |
| SLEEP_MODE_ADC | ✘ | ✘ | ✔ | ✔ | ✔ |
| SLEEP_MODE_EXT_STANDBY | ✘ | ✘ | ✘ | ✔ | ✔ |
| SLEEP_MODE_STANDBY | ✘ | ✘ | ✘ | ✘ | ✔ |
| SLEEP_MODE_PWR_SAVE | ✘ | ✘ | ✘ | ✔ | ✘ |
| SLEEP_MODE_PWR_DOWN | ✘ | ✘ | ✘ | ✘ | ✘ |

- Dieper slapen betekent:
  - Minder hardware beschikbaar.
  - Minder stroomverbruik.
  - Langere tijd nodig om wakker te worden.



## Slaapstanden

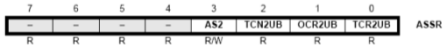
- Voorbeeld ATmega32A $Icc_{MAX}$ bij $f_{CPU}$=8 MHz en Vcc=5 V. Active: 15 mA, Idle: 8 mA, Power Down: 20 µA.

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

ISR(TIMER1_COMPA_vect) {
    PORTB ^= 0x01;
}

int main(void) {
    DDRB = 0xFF;
    PORTB = 0xFF;
    OCR1A = 1799;
    TCCR1B = 0x0D;
    TIMSK |= (1<<OCIE1A);
    sei();
    set_sleep_mode(SLEEP_MODE_IDLE);
    while (1) {
        sleep_mode();
    }
    return 0;
}
```



## T/C2 Asynchronous mode

- Timer/Counter2 kan in een asynchrone mode gebruikt worden.
  - Wordt dan geklokt met apart horloge kristal 32.768 kHz aangesloten op TOSC1 = PC6 en TOSC2 = PC7.
  - Ons STK500 practicumbord heeft zo'n kristal.




  - AS2 = 1 : Asynchrone mode
  - *XXXX*UB (Update Busy) = 1 : schrijven *XXXX* register is nog niet klaar.

## T/C2 Asynchronous mode

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

ISR(TIMER2_OVF_vect) {
    PORTB ^= 0x01;
}

int main(void) {
    DDRB = 0xFF;
    PORTB = 0xFF;
    ASSR |= (1<<AS2);
    TCCR2 = 0x04;
    TIMSK |= (1<<TOIE2);
    sei();
    set_sleep_mode(SLEEP_MODE_PWR_SAVE);
    while (1) {
        sleep_mode();
    }
    return 0;
}
```


DE HAAGSE HOGESCHOOL 157

## Interrupt en volatile

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

volatile uint8_t ready = 0;

ISR(TIMER1_OVF_vect) {
    //...
    ready = 1;
}

int main(void) {
    //...
    sei();
    while (1) {
        if (ready == 1) {
            //...
        }
    }
    return 0;
}
```

Waarom moet ready volatile zijn?

Zie: http://bd.eduweb.hhs.nl/micprg/volatile.htm#interrupts

DE HAAGSE HOGESCHOOL 158