

Industriële ICT

# Softwareontwikkeling

Programmeren van 8-bits Microchip  
AVR microcontrollers



Wim Baert  
THINKTWISE

## Inhoud

Wat is een microcontroller? .....	3
Waarom blijven we niet gewoon de Arduin Uno gebruiken? .....	3
Wat zijn de kenmerken van de ATmega328P-chip? .....	3
Startoefening assembleertaal en C .....	5
Wat is de functie van het PC-register in de microcontroller? .....	5
Hoe kan je de werking van de program-counter uittesten? .....	6
Wat betekent .org 0x0000 en .org 0x0100 ? .....	7
Hoe wordt een programma in het flashgeheugen opgeslagen? .....	8
Wat zijn subroutines?.....	12
Hoe worden subroutines afgehandeld? .....	13
Hoe kan ik de digitale IO van mijn AVR als output gebruiken? .....	15
Hoe laad ik met Atmel Studio een programma in een Arduino? .....	17
Programmeren van de Arduino vanuit Atmel Studio.....	18
Intermezzo : Command line argumenten bij programma's.....	21
Hoe maak ik van mijn Arduino Uno een AVR-programmer?.....	23
Wat is SPI? .....	23
Ik heb geen ISP-programmer! Wat nu? .....	25
Hoe gebruik ik mijn Arduino Uno bordje als ISP-programmer?.....	26
Hoe kan een AVR serieel communiceren met een PC?.....	29
Wat is RS232 (ANSI/EIA/TIA-232-F)? .....	29
Welke kenmerken heeft RS232 (ANSI/EIA/TIA-232-F)?.....	29
RS232 (ANSI/EIA/TIA-232-F) in de 21 <sup>e</sup> eeuw?.....	29
Signaalverloop bij RS232 .....	30
Hoe kan je bits manipuleren.....	32
Probleemstelling .....	32
Schuifoperaties .....	32
Bit-twiddeling .....	33
Hoe kan je in de AVR digitale ingangen gebruiken? .....	35
Hoe gebruik ik in de C-taal pointers als functieparameters?.....	37
Hoe kan ik met de AVR analoge signalen meten?.....	40
Wat zijn interrupts? .....	41
Probleemstelling .....	41
Verloop van interruptafhandeling .....	43
Normale programmaverloop : in main wordt een oneindige lus uitgevoerd	43
Uitvoering ISR : normale programmaverloop wordt onderbroken .....	44

Schema : afhandeling van een IRQ .....	45
Interrupt in de ATmega328p - mechanisme.....	46
Interrupt in de ATmega328p – configuratie .....	47
Interrupt in de ATmega328p – frame voor C-code .....	48
Hoe kan ik de Timer-modules in de AVR gebruiken? .....	49
Toepassingen van timers en tellers.....	49
De generieke Timer/Counter.....	50
Blokschema van een generieke timer .....	50

## Wat is een microcontroller?

- Een microcontroller ( $\mu$ C) is een *1-chip computer* met I/O-mogelijkheden.
- Op het Arduino Uno platform gebruikt men de ATmega328P  $\mu$ C.
- De ATmega328 is een 8-bits  $\mu$ C
- De ATmega328P is oorspronkelijk van Atmel maar zit nu onder de vleugels van Microchip.
- Ondersteuning via [www.microchip.com](http://www.microchip.com)

## Waarom blijven we niet gewoon de Arduino Uno gebruiken?

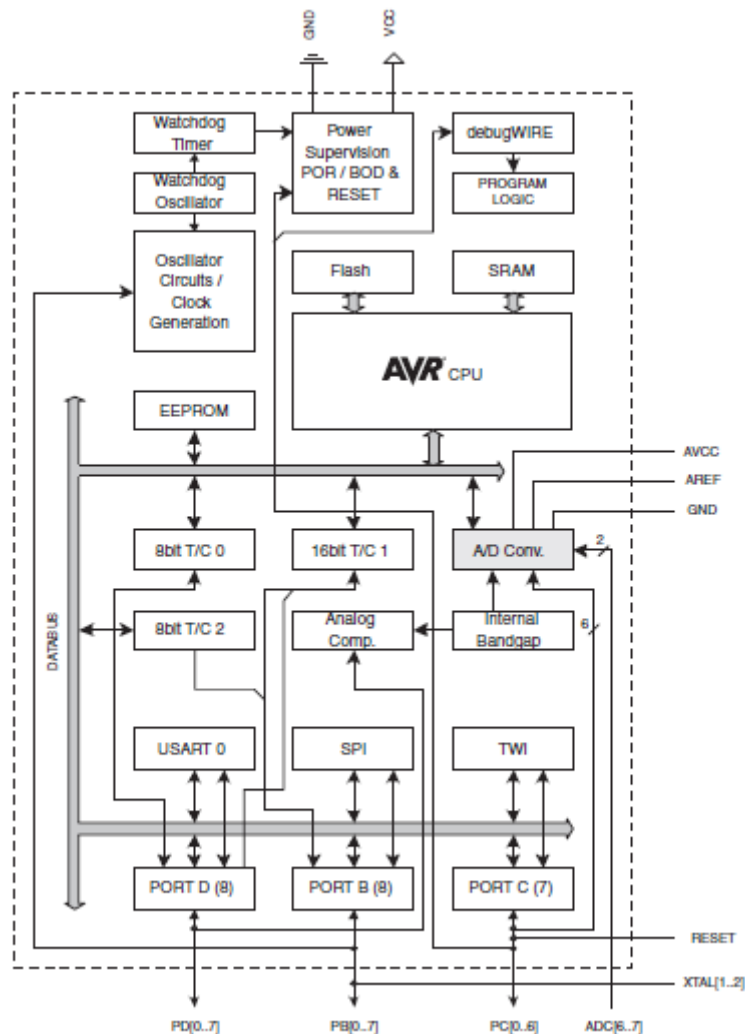
- Arduino begrensd de mogelijkheden die de ATmega328P biedt
- In een toepassing is het gebruik van de chip veel goedkoper dan het gebruik van een volledige Arduino
- De C-taal die we gebruiken om de ATmega328P te programmeren is superieur aan de Arduino-C

## Wat zijn de kenmerken van de ATmega328P-chip?

Om met een  $\mu$ C te leren werken moet je beschikken over het datablad. Dat bevat alle informatie die nodig is om het toestel in dienst te krijgen. Voor de ATmega328P kan je het datablad downloaden via de site van [Microchip](http://www.microchip.com) . Ga niet gewoon via zoeken in Google. Er circuleren nog oude versies van het datablad of versies die nog van Atmel afkomstig zijn. De huidige producent levert de meest recente informatie.

Volgende kenmerken zijn in deze fase van de cursus al belangrijk:

- Advanced **RISC** Architecture met **Harvard-computerarchitectuur**
- 131 assembleertaalinstructies waarvan de meeste in 1 kloktik worden uitgevoerd
- 32 vrij bruikbare registers (General Purpose Registers – **GPR's** - r0 tot r31)
- Snelheid van 20 **MIPS** bij een klok van 20MHz
- 32KBytes programmeergeheugen (Flash)
- 1KBytes **EEPROM**
- 2KBytes Internal **SRAM**
- 32 **digitale IO's**



### Opdrachten :

1. Leg uit wat men bedoelt met "...is een 8-bits  $\mu C$ "?
2. Leg de opbouw van de Harvard-computerarchitectuur uit
3. Bespreek de opbouw van de Von Neumann computerarchitectuur
4. Waarvoor staat de afkorting RISC?
5. Wat houdt RISC in? Vergelijk RISC met CISC
6. Welke computerarchitectuur wordt er vaak bij RISC gebruikt?
7. Waarvoor staat de afkorting MIPS
8. Wat zijn GPR's
9. Wat is SRAM? Vergelijk SRAM met DRAM.
10. Wat is EEPROM?
11. Wat is een digitale IO?
12. De ATmega328p heeft 2 kByte SRAM. Bereken het aantal standaard integers en doubles je kan opslaan in dit geheugen.
13. Installeer de Atmel Studio programmeeromgeving.
14. Wat is het resultaat wanneer we de microcontroller 255 + 6 laten berekenen? Heeft deze bewerking invloed op de bits van het statusregister? Zo ja, welke invloed?

## Startoefening assembleertaal en C

Een teller die bijgehouden wordt in r16 telt telkens af van 10 naar 0.

- Wanneer r16=0 wordt dan zal een tweede teller die wordt bijgehouden in r17 met één verminderen. Bij de start is r17=5.
- Wanneer r17=0 stopt het programma.

Tip : bekijk de brne-instructie

Hoe ziet de structuur van dit programma er in C uit?

Schrijf het programma neer in C.

## Wat is de functie van het PC-register in de microcontroller?

PC staat voor Program Counter. In dit register wordt automatisch bijgehouden op welk adres *de volgende uit te voeren instructie staat*.

Het programmeergeheugen van de Atmega328p omvat 32 kB (=32768 bytes). Elke instructie in de Atmega328P is 16 bits of 32 bits breed. Het programmeergeheugen van de ATmega328P is om die reden opgebouwd uit 16k lijnen van elk twee bytes breed. Om 16k programmalijnen te kunnen adresseren is een program counter van 14 bits breed nodig. Met 14 bits kan je immers  $2^{14} = 16384$  locaties adresseren.

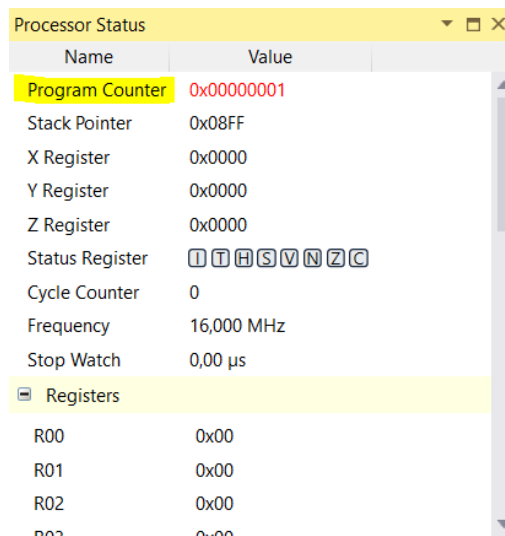
byteadres	PC	low byte	high byte
0000	0x0000	16-bits instructie. Vb. LDI r16,0x55	
0002	0x0001	32-bits instructie. Vb. JMP start	
0004	0x0002		
0006	0x0003		

- Een voorbeeld van een 16-bits instructie is LDI (load immediate)
- Een voorbeeld van een 32-bits instructie is JMP (onvoorwaardelijke sprong)

Bij het opstarten van de microcontroller is de program counter PC=0x0000. De eerste instructie die wordt uitgevoerd staat dus op adres 0x0000.

Spronginstructies zoals JMP, RJMP, CALL, BRNE,...veranderen de inhoud van de programcounter. Op die wijze is het mogelijk om sprongen en lussen in een programma te voorzien. De uitleg van de werking van de program counter is niet alleen geldig voor een microcontroller. Elke digitale computer beschikt over een program counter. Vaak kom je ook de benaming *Instruction Pointer* (IP) tegen.

In Atmel-studio kan je de program counter tijdens het debuggen (in simulatie) terugvinden in het processor status venster.



Name	Value
Program Counter	0x00000001
Stack Pointer	0x08FF
X Register	0x0000
Y Register	0x0000
Z Register	0x0000
Status Register	01101010
Cycle Counter	0
Frequency	16,000 MHz
Stop Watch	0,00 µs
Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00

Hoe kan je de werking van de program-counter uittesten?

Tik de onderstaande code in en test ze uit met de simulator.

```
;
; testPC.asm
;
; Schooljaar 2020-2021
; Testprogramma om de werking van de PC te bestuderen
; Author : Ing. Wim Baert

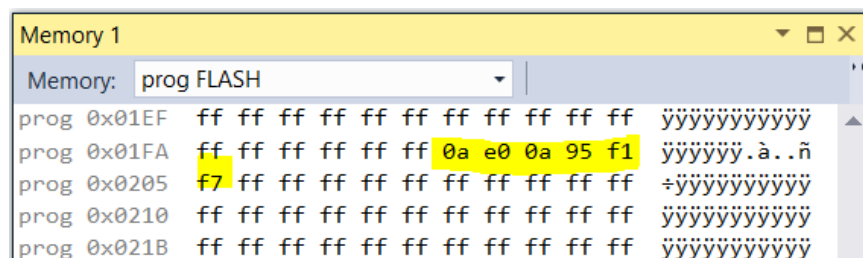
.org 0x0000
    jmp start

.org 0x0100
start:
    ldi r16,10
loop:
    dec r16
    brne loop
```

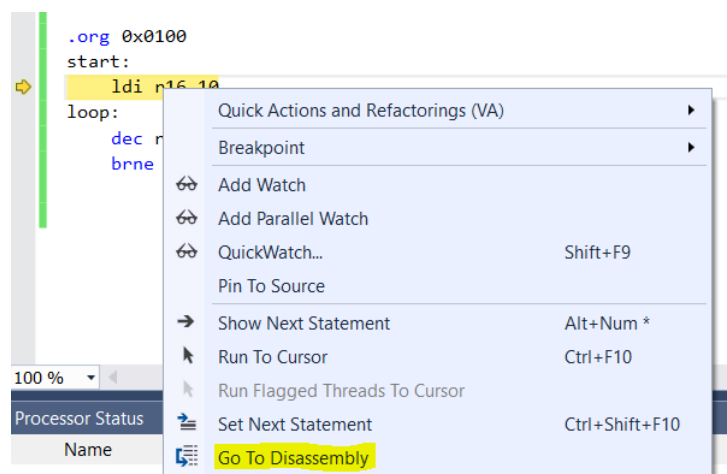
Wat betekent .org 0x0000 en .org 0x0100 ?

.org is een *directive*. Letterlijk vertaald is dit een richtlijn. We geven aan de assembler (= het programma dat onze code omzet naar machinecode) de opdracht om de jmp start instructie op lijn 0x0000 van het programmeergeheugen te zetten en de rest van de code vanaf lijn 0x0100 in het programmeergeheugen te zetten. Let op: de program-counter wijst naar een lijn in het programmeergeheugen. Het byteadres waar je de instructie in het programmeergeheugen kan terugvinden is tweemaal de waarde die in de PC staat.

De instructies die vanaf programmalijn 0x0100 in het programmeergeheugen staan kan je in het geheugenvenster terugvinden vanaf adres 0x200

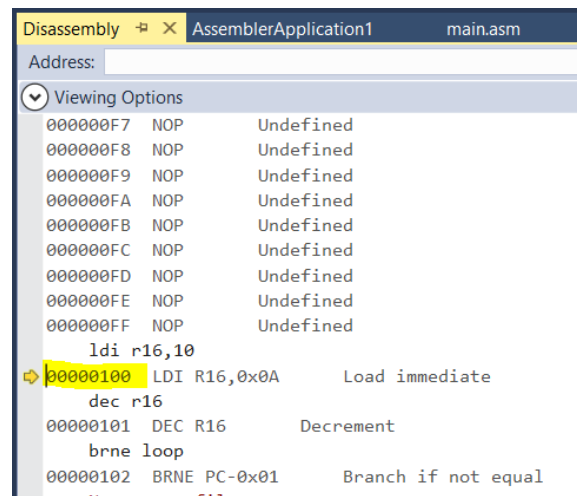


Wanneer je tijdens het debuggen rechts klikt op een instructie dan krijg je het volgende pop-up menu.

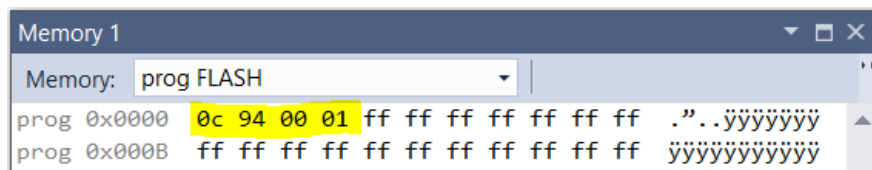




Kies voor *Go to disassembly*. Je krijgt dan de code te zien met de adressen van de programmalijnen:



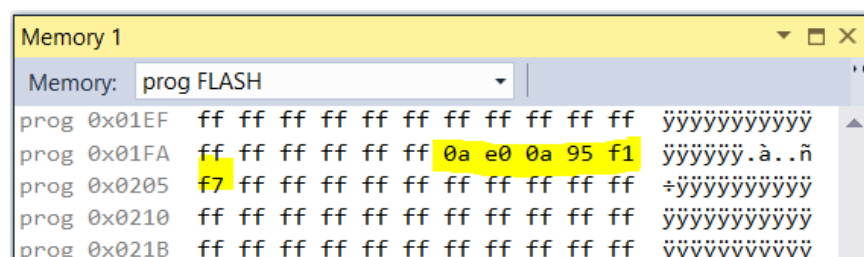
In het programmeergeheugen start de code vanaf byteadres 0x0000.



Het vervolg kan je – zoals in een eerdere afbeelding is getoond – terugvinden vanaf adres 0x0200.

Hoe wordt een programma in het flashgeheugen opgeslagen?

De *assembleertaal* is het laagste menselijke programmeerniveau. Instructies zoals LDI, RJMP, BNEQ,... noemt men *mnemotechnische codes* of kortweg *mnemonics*. Deze codes worden door een assembler omgezet naar binaire code. In het programmeergeheugen zie je dan de binaire code die de µP zal uitvoeren. Om het leesbaar te houden wordt deze binaire code neergeschreven in hexadecimale vorm.



De eerste van de instructies die starten op byteadres 0x200 is LDI r16,10. In het programmeergeheugen staat dit neergeschreven als "0a e0".

Om deze hexadecimale voorstelling van de binaire programmacode op te stellen is er gelukkig de assembler. Die doet dat gelukkig voor ons.

Niettemin is het interessant om te weten waar deze code vandaan komt.

Hiervoor gebruiken we de referentie-gids voor de assembleertaal-instructies.

In die gids staat in detail neergeschreven hoe elke instructie functioneert. We bekijken dit voor de LDI-instructie

## 73. LDI – Load Immediate

### 73.1. Description

Loads an 8-bit constant directly to register 16 to 31.

Operation:

(i)  $Rd \leftarrow K$

Syntax:

Operands:

Program Counter:

(i) LDI Rd,K

$16 \leq d \leq 31, 0 \leq K \leq 255$

$PC \leftarrow PC + 1$

16-bit Opcode:

1110	KKKK	dddd	KKKK
------	------	------	------

### 73.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
clr r31 ; Clear Z high byte
ldi r30,$F0 ; Set Z low byte to $F0
lpm ; Load constant from Program
; memory pointed to by Z
```

**Words** 1 (2 bytes)

**Cycles** 1

"Operation" legt uit wat de instructie precies doet. In dit geval " $Rd \leftarrow K$ ".

- Rd : destination Register (bestemmingsregister)
- K : een 8 bits constante. Dit noemt men ook een literal.
- $\leftarrow$  : Geeft de richting van de gegevensstroom aan. In dit geval wordt K opgeslagen in Rd

"Syntax" omschrijft hoe je de instructie in je programma moet neerschrijven. In het voorbeeldprogramma is dat *LDI r16,10*

- Na de instructie noteer je eerst de symbolische naam van het register waarin je het getal wil schrijven
- Vervolgens schrijf je de waarde neer. Dat kan decimaal (10), hexadecimaal (0x0A) of binair (0b00001010)

"Operands" geeft de grenzen aan van de operands.

- De LDI instructie kan je enkel gebruiken met de registers r16 tot r31
- K heeft een waarde in het interval [0,255]

"Program Counter" geeft aan wat er met de program counter gebeurt wanneer de LDI-instructie wordt uitgevoerd. In dit geval staat er " $PC \leftarrow PC+1$ ". Dat wil zeggen dat de PC met één wordt verhoogd. Na een LDI-instructie zal gewoon de volgende instructie worden uitgevoerd.

Op basis van de instructie en rekening houdend met de bovenstaande opmerkingen kan je de 16-bits binaire code voor de instructie LDI r16,10 samenstellen:

- Code voor de LDI-instructie = 0b1110

16-bit Opcode:

1110	KKKK	dddd	KKKK
------	------	------	------

- De constante die we willen opslaan is 10. In hexadecimale notatie is dat 0x0a. In binaire notatie is dat 0b00001010. Deze constante wordt in twee delen aan de instructie toegevoegd. De *nibble* van hogere orde komt op de linkse positie aangegeven met KKKK. De *nibble* van lagere orde komt op de rechtse positie aangegeven met KKKK.
- Het register waar we deze waarde in willen schrijven is r16. Er kunnen zestien verschillende registers worden gebruikt door de LDI-instructie. Om zestien dingen te onderscheiden heb je vier bits nodig. De registers r16 krijgen een volgnummer van 0b0000 tot 0b1111. Dit volgnummer wordt op de positie geschreven waar "dddd" staat in de opcode.

Met de bovenstaande gegevens kunnen we de 16-bits opcode voor de instructie LDR r16,10 neerschrijven:

1	1	1	0	K	K	K	K	d	d	d	d	K	K	K	K
1	1	1	0	0	0	0	0	0	0	0	0	1	0	1	0

Deze 16-bits opcode schrijven we hexadecimaal als

0xe0	0x0a
MSB	LSB

Dat is precies wat we in het FLASH-geheugen van de  $\mu$ C zien staan! Enkel de volgorde is omgekeerd. Dat komt omdat de ATmega voor het opslaan van gegevens gebruik maakt van "little endian".

"little endian" staat voor little end first : De byte van lagere orde (LSB) komt in de geheugenlocatie met het laagste adres. In het geheugen zie je dus eerst 0x0a en dan pas 0xe0. In de onderstaande tabel zie je dit duidelijk.

De eerste kolom geeft de waarde aan van de PC en de instructie waar de PC naar wijst. De tweede kolom bevat het overeenkomstige byteadres van de geheugenlocaties. In de derde kolom zie je wat er in de geheugenlocatie staat.

Program Counter	Byteadres	inhoud
0x00000100 LDI R16,0x0A	0x200	0x0a
	0x201	0xe0
0x00000101 DEC R16	0x202	0x0a
	0x203	0x95
0x00000102 BRNE PC-0x01	0x204	0xf1
	0x205	0xf7

Memory 4	
Memory: prog FLASH	Address: 0x0108,prog
prog 0x0108	ff ff
prog 0x0124	ff ff
prog 0x0140	ff ff
prog 0x015C	ff ff
prog 0x0178	ff ff
prog 0x0194	ff ff
prog 0x01B0	ff ff
prog 0x01CC	ff ff
prog 0x01E8	ff ff
prog 0x0204	f1 f7 ff
prog 0x0220	ff ff

Je weet nu hoe instructies in het FLASH-geheugen worden gezet en hoe je ze kan interpreteren. Naast 'little endian' kent men ook 'big endian'. In een big-endian toestel zal de MSB op het laagste geheugenadres staan. 'Endianess' is vooral belangrijk wanneer een little endian en big endian gegevens uitwisselen en opslaan in het geheugen voor latere verwerking. Bij de interpretatie van de

gegevens moet de programmeur weten dat de bytes bij de transfer in het geheugen van positie zijn omgewisseld.

## Wat zijn subroutines?

In de lessen over C++ heb je kennis gemaakt met functies. Subroutines zijn in assembleertaal het equivalent van dergelijke functies. Het zijn stukjes herbruikbare code die ervoor kunnen zorgen dat het programma meer leesbaar wordt.

De basisstructuur van een subroutine voor de ATmega328p ziet er als volgt uit:

```
;
; test_subroutine.asm
;
; Schooljaar 2020-2021
; Testprogramma om de afhandeling van subroutines te bestuderen
; Author : Ing. Wim Baert

; start = deel van het programma dat slechts één keer wordt
; uitgevoerd
start:

    ; code die éénmalig wordt uitgevoerd
    call setupIO
    ldi r16,0x00

;main = oneindige lus die de hoofdcode bevat
main:

    ; code die oneindig uitgevoerd wordt
    com r16
    out portb,r16
    call delay
    rjmp main

; setupIO = voorbeeldsubroutine
setupIO:

    ldi r16,0xFF
    out ddrb,r16
    ret

delay:

    ; code voor delay-routine

    ret
```

Een subroutine identificeer je met een label. Deze naam is vrij te kiezen. De laatste instructie in een subroutine is de ret-instructie (=return). Een subroutine roep je in assembleertaal aan met de call-instructie.

```
; setupIO = voorbeeldsubroutine
setupIO:

    ; code van de subroutine

    ret
```

### Hoe worden subroutines afgehandeld?

Wanneer een subroutine wordt aangeroepen dan moet de processor in de microcontroller een stukje code uitvoeren dat op een andere geheugenlocatie start dan de locatie waar de program counter (PC) naar wijst. De PC wijst immers altijd naar de volgende uit te voeren instructie.

Na de afhandeling van de subroutine moet het programma verder gaan met de instructie die net na de aanroep van de subroutine staat.

We bekijken het mechanisme aan de hand van het volgende codefragment:

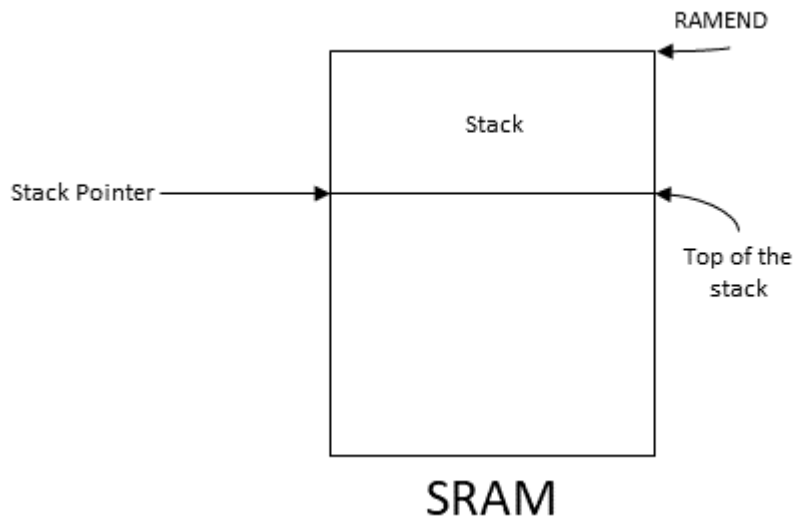
```
call setupIO
ldi r16,0x00
```

Stel dat de PC wijst naar de instructie `call setupIO`. Dat is dus de instructie die de processor als volgende instructie zal uitvoeren.

Wanneer de processor de `call`-instructie uitvoert dan zullen de volgende stappen doorlopen:

- De PC wordt met één verhoogd. Die wijst dus naar de instructie net na de `call`-instructie. Natuurlijk zal die instructie pas worden uitgevoerd nadat de subroutine is afgehandeld. De PC bevat dus niet de correcte waarde. De PC moet wijzen naar de eerste instructie van de subroutine.
- De huidige inhoud van de PC wordt opgeslagen in een geheugengebied dat men de **stack** noemt. Een stack is een stapel. Je kan er vanboven iets opleggen of afnemen. Wanneer je zaken op de stack legt dan groeit hij. Wanneer je er zaken afneemt dan krimpt hij. Iets op de stack zetten noemt men een push. Iets van de stack halen noemt men een pop. *In vaktermen wordt de PC dus op de stack gepushed.* In de AVR-controller start de stack op het hoogste SRAM-adres (RAMEND). RAMEND is bij de start van het programma de top van de stack (TOS). Elke stacklocatie neemt twee bytes in het geheugen in. Telkens er iets op de stack wordt geduwd zal het adres van de TOS met twee afnemen.

Het adres van de TOS wordt bijgehouden in een stackpointer-register. Dat is een 16-bits register.



- De PC wordt geladen met het adres van de eerste instructie van de subroutine. Dat is de eerste instructie die na het label staat waarmee we de subroutine identificeren.
- Vervolgens wordt de subroutine instructie per instructie uitgevoerd. De laatste instructie is de ret-instructie.

Op de stack is het adres van de instructie gepushed die na de call moet worden uitgevoerd. De ret-instructie zorgt ervoor dat aan het einde van de subroutine automatisch een pop van de stack wordt gedaan. De waarde die op de top van de stack staat wordt in de PC geladen. Het programma vervolgt dan met de eerste instructie na de call-instructie.

**Opdracht:** schrijf een testprogramma waarbij je in een subroutine een andere subroutine aanroept die op haar beurt nog een andere subroutine uitvoert. Leg uit hoe de processor er met behulp van een stack toch in slaagt om de uitvoering van het programma correct te laten verlopen.

## Hoe kan ik de digitale IO van mijn AVR als output gebruiken?

Wanneer je de digitale IO van de microcontroller als output configureert dan kan je met de microcontroller allerlei toestellen aansturen. Dat is uiteraard één van de hoofdtaken van de microcontroller.

Om te weten hoe je de IO als output kan gebruiken moet je weten hoe Data Direction Registers (DDRx) en Port-register (PORTx) functioneren.

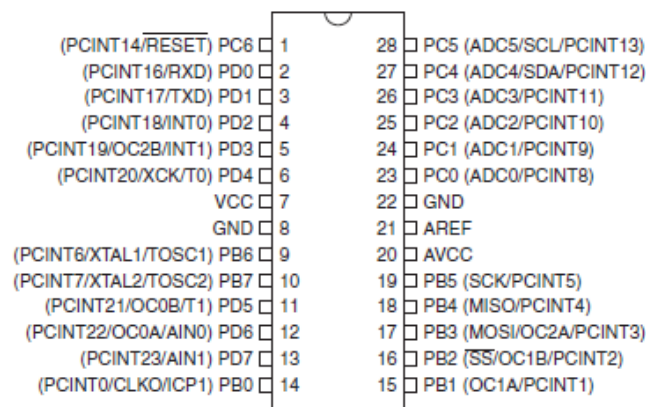
De digitale IO-pinnen in de microcontroller kunnen zowel als input als output werken. Dat is de werkmingsmodus van die pinnen. Je moet in de code aangeven in welke modus een pin werkt.

Om dat te doen is er voor elke PORTx-register een overeenkomstig DDRx-register voorzien. Het DDRx-register geeft voor elke IO van een poort aan in welke richting die IO werkt.

- Wanneer een bit in het DDRx-register op "1" staat dan is de overeenkomstige pin in het PORTx-register output.
- Wanneer een bit in het DDRx-register op "0" staat dan is de overeenkomstige pin in het PORTx-register input.

Let op : de x moet je vervangen door de letter van de poort die je gebruikt (A,B,C,D,...). Welke poorten je kan gebruiken hangt af van het typenummer van microcontroller.

We bekijken dit voor PORTB:





Het PORTB register is met de buitenwereld verbonden via de pinnen PB0..PB7 van de microcontroller. Voor elke pin van PORTB is er in het DDRB-register een bit voorzien:

PORTB							
PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0

DDRB							
b7	b6	b5	b4	b3	b2	b1	b0
1	0	0	1	1	0	1	1

In de bovenstaande tabel is DDRB=0b10011011, dat is in hex 0x9B

Deze instelling heeft tot gevolg dat de PORTB-pinnen de volgende werkingsmodus krijgen:

PIN	Werkingsmodus
PB7	output
PB6	input
PB5	input
PB4	output
PB3	output
PB2	input
PB1	output
PB0	output

Je kan in assembleertaal waarden in een DDRx register schrijven met de volgende assembleertaalcode

```
LDI r16,0x9B
OUT DDRB
```

Je kan bits van het PORT-register op 0 of 1 zetten met behulp van volgende assembleertaalcode:

```
LDI r16,0x55
OUT PORTB
```

In AVR C is het mogelijk nog makkelijker:

```
DDRB = 0x9B;
PORTB= 0b10011011;
```

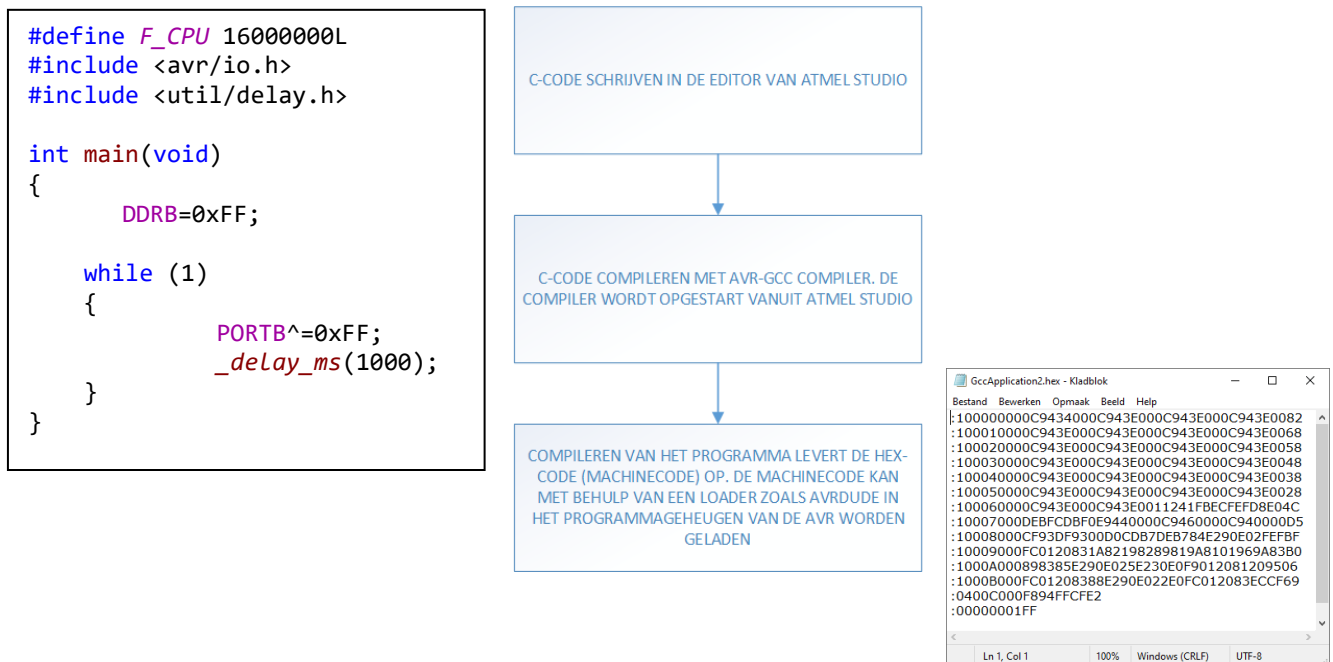
Let er wel op dat PB6, PB5 en PB2 inputs zijn. Je kan die niet vanuit de controller aansturen. Dat is iets waar we in één van de volgende lessen een oplossing voor zullen voorzien.

## Hoe laad ik met Atmel Studio een programma in een Arduino?

Tot nu toe heb je enkel met de simulator gewerkt die met Atmel Studio komt. We willen uiteraard met een echte controller aan de slag. Als alles goed is verlopen dan heb je in deze fase van de lessenreeks een werkend C-programma voor een PWM-sigitaal met een periode van 40 ms en een dutycycle van 50%.

Deze C-code gaan we met behulp van Atmel Studio naar de ATmega328 op het Arduino-printje schrijven.

Te volgen toolchain:



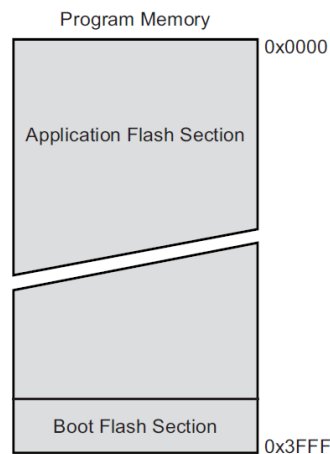
We gaan in twee fasen het programmeerproces bestuderen

1. Programmeren van de Arduino vanuit Atmel Studio. Hiervoor moeten er een aantal instellingen in Atmel Studio worden uitgevoerd. Dit is een makkelijke methode, maar daar programmeer je alleen de Arduino mee.
2. Programmeren van een lege chip met behulp van een programmer en Atmel Studio. We bouwen de Arduino om tot programmer. Dat is iets moeilijker omdat je de Arduino moet verbinden met een lege chip. Dit is wel de leukste werkwijze. Je kan de chip dan inbouwen in een eigen schakeling. Je kan ook een andere AVR kiezen die meer geschikt is voor jouw toepassing.

## Programmeren van de Arduino vanuit Atmel Studio.

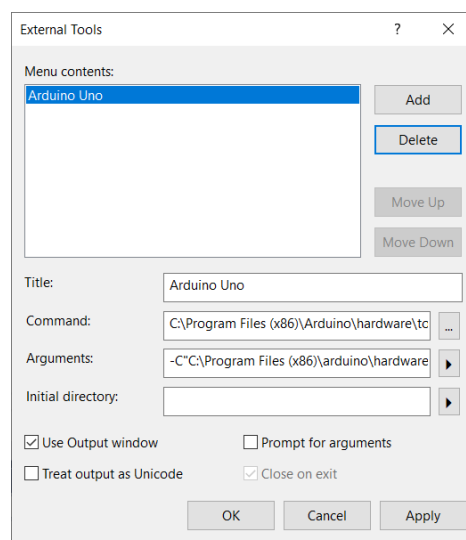
Hiervoor moeten er een aantal instellingen in Atmel Studio worden uitgevoerd.

De ATmega328 die op het Arduino-bordje zit is in het programmeergeheugen voorzien van een bootloader. Deze bootloader is een stukje code dat zich in een speciaal deel van het flashgeheugen van de controller bevindt:



De bootloader kan uitgevoerd worden na een reset van de controller of op basis van een commando dat via de UART of SPI-pinnen naar de controller wordt gestuurd. De bootloader zal communiceren met de loader en de code die de loader doorstuurt in het flash-geheugen schrijven (Application Flash Section). Wanneer het programma is ingeladen kan men de controller resetten. Eerst wordt de bootloader opgestart. Wanneer er geen loader communiceert met de bootloader dan zal vanuit de bootloader het programma dat in de Application Flash Section staat worden opgestart.

Om de AVR op het Arduino-bordje te programmeren moet je het programma avrdude.exe toevoegen als externe tool. Je doet dit via het menu "Tools":



Voeg met "Add" een tool toe. Als naam (title) kies je Arduino Uno.  
Bij "Command" komt het pad naar het avrdude.exe programma. Bij "Arguments" komen de command line argumenten die worden meegegeven bij het opstarten van avrdude.exe (zonder newline's)

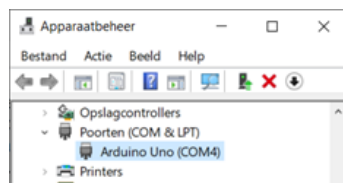
Command:

C:\Program Files (x86)\Arduino\hardware\tools\avr\bin\avrdude.exe

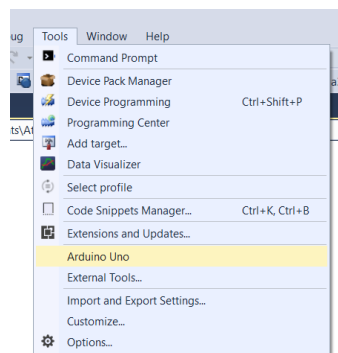
Arguments:

```
-C"C:\Program Files (x86)\arduino\hardware\tools\avr\etc\avrdude.conf" -v  
-patmega328p -carduino -PCOM4 -b115200 -D -  
Uflash:w:"$(ProjectDir)Debug\$(TargetName).hex":i
```

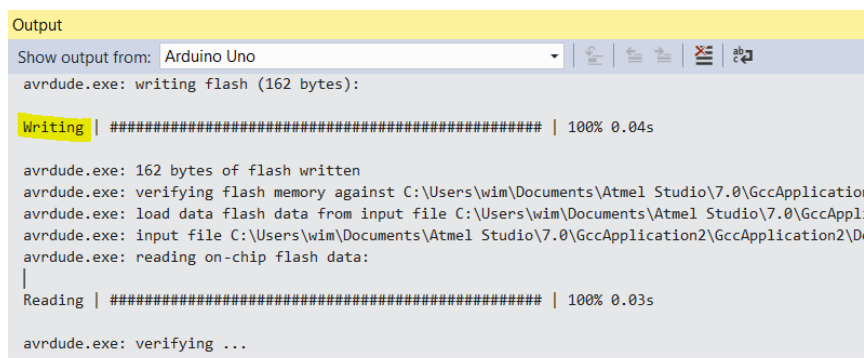
Het is mogelijk dat je de rood gemarkeerde zaken moet aanpassen voor jouw situatie. Wanneer je een andere microcontroller gebruikt dan wijzig je het -patmega328p argument. Wanneer de Arduino verbonden is met een andere seriële poort dan wijzig je het argument -PCOM4



Om het programma dat je hebt geschreven in de Arduino te laden kies je gewoon bij "Tools" voor de externe tool "Arduino Uno" die je hebt aangemaakt:



Als je alles correct hebt uitgevoerd dan zal in het outputvenster een rapport verschijnen van het programmeren:



Het programma staat nu in het flashgeheugen van de Atmega328p op het Arduino-bordje. Je kan de werking uittesten.



## Intermezzo : Command line argumenten bij programma's

Het avrdude-programma heeft behoorlijk wat argumenten. Wanneer je code naar de Arduino schrijft dan wordt avrdude.exe aangeroepen met zijn argumenten. Je hebt een gelijkaardig iets gezien bij het programma vboxmanage.exe dat gebruikt wordt bij het beheer van VirtualBox virtuele machines. Het toevoegen van argumenten aan een programma is niet moeilijk. Bekijk even de volgende C++-code. Maak een testproject aan in Visual Studio:

```
// argumentTest.cpp - C++ code voor PC-applicatie

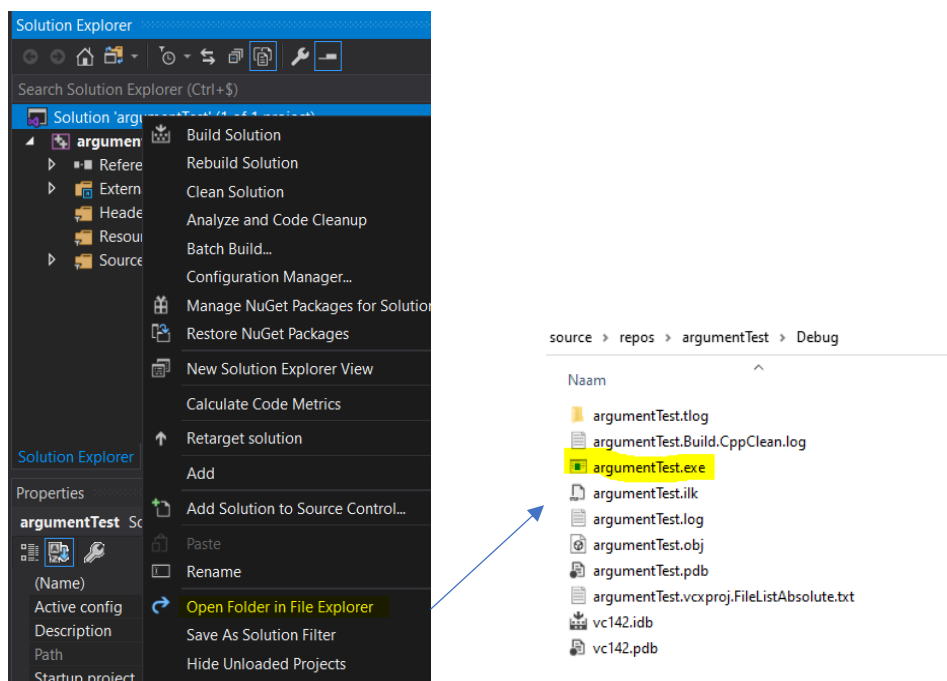
#include <iostream>

int main(int argc, char* argv[])
{
    std::cout << "Code met argumenten\n";

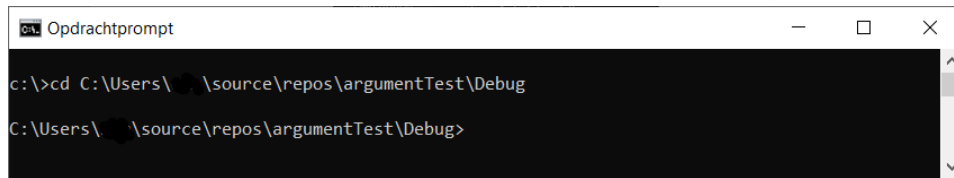
    for (int i = 0; i < argc; i++)
    {
        std::cout << i << " : " << argv[i]<<"\n";
    }

    return 0;
}
```

Compileer het programma en zoek het pad waarin het gecompileerde bestand staat. Je kan dit makkelijk vinden door in de solution explorer van Visual Studio rechts te klikken op de projectnaam en "Open folder in File Explorer" te kiezen. In de debug-map van het project kan je het uitvoerbaar .exe bestand terugvinden:

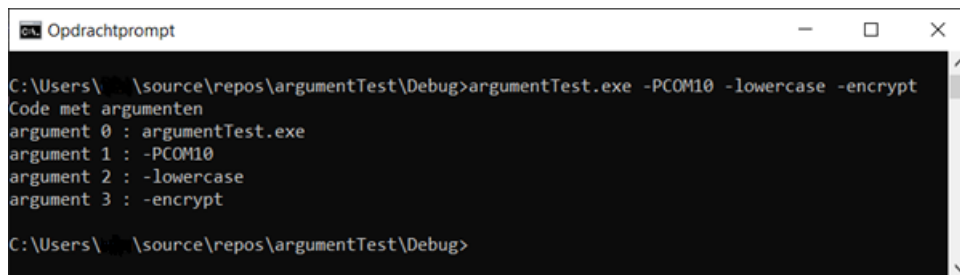


Om het programma uit te voeren open je een commandovenster. Je gaat met het cd-commando (change directory) naar de debug map.



```
Opdrachtprompt
c:\>cd C:\Users\...source\repos\argumentTest\Debug
C:\Users\...source\repos\argumentTest\Debug>
```

Je kan vanuit die map het programma uitvoeren. In het voorbeeld zijn *dummy*-argumenten gebruikt. De argumenten worden via een array doorgegeven aan de main-routine. Het nulde element is altijd de programmanaam. In het testprogramma worden in een lus de argumenten afgedrukt.



```
Opdrachtprompt
C:\Users\...source\repos\argumentTest\Debug>argumentTest.exe -PCOM10 -lowercase -encrypt
Code met argumenten
argument 0 : argumentTest.exe
argument 1 : -PCOM10
argument 2 : -lowercase
argument 3 : -encrypt
C:\Users\...source\repos\argumentTest\Debug>
```

## Hoe maak ik van mijn Arduino Uno een AVR-programmer?

Je kan nu Atmel Studio gebruiken in combinatie met Arduino. Die combinatie laat je toe om snel prototype-code in C te schrijven en die uit te testen op een AVR-microcontroller.

Het Arduino-bordje inbouwen in de uiteindelijke toepassing is vaak niet de beste strategie. Het maakt het eindproduct duur en vaak ook een pak duurder. Je kan na de prototype-fase een andere chip nemen uit de AVR 8-bits lijn en die chip gebruiken als hart van jouw toepassing. Die chip plaats je op een eigen PCB waarvan de vorm is aangepast aan de vorm van het eindproduct. Eens jouw PCB is afgewerkt kan je het programma inladen.

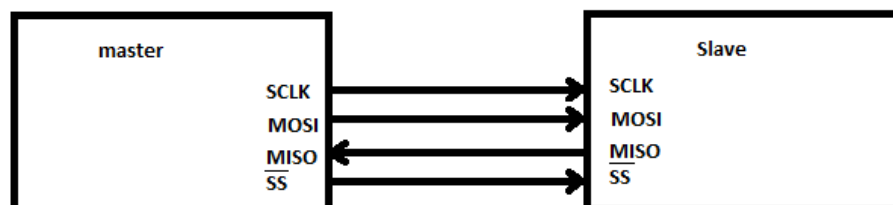
Omdat de AVR-chip al in het eindproduct zit moet je de chip In System kunnen programmeren. Dit principe noemt men In System Programming (ISP).

### Wat is SPI?

Om een AVR met ISP te programmeren kan je gebruik maken van het SPI-protocol. SPI staat voor Serial Peripheral Interface. Een SPI-interface is een verbinding tussen een zender en meerdere ontvangers. De zender is *master*. De master start en stopt de communicatie en bepaalt hoe snel er wordt gecommuniceerd. De ontvangers zijn *slaves*.

Om via SPI te communiceren zijn er vier verbindingen nodig:

- SCLK : een kloksignaal
- MOSI : datalijn Master Out Slave In
- MISO : datalijn Master In Slave Out
- $\overline{SS}$  : Slave Select. Dit is een *actief laag* signaal



SPI is een *synchrone* bus. Het kloksignaal wordt door de master opgewekt en via een SCLK-lijn verdeeld. De slaves gebruiken dit kloksignaal.

SPI is *full-duplex*. Er is gelijktijdige tweerichtingscommunicatie mogelijk aan hoge snelheid. De *bitrate* (bits/s) hangt af van de kwaliteit van de busverbinding en de snelheid die de slaves aankunnen.

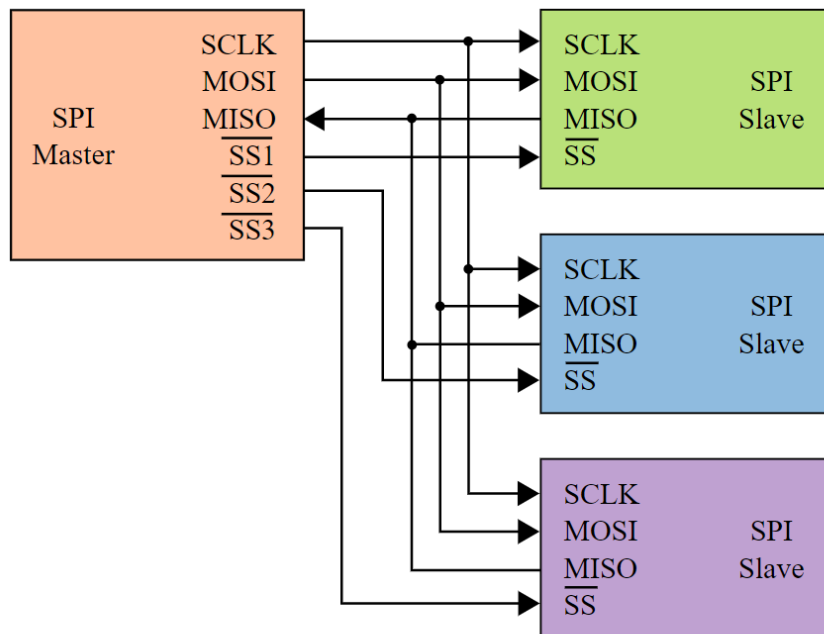
SPI is een *bussysteem*. Je kan meerdere slaves aansluiten aan dezelfde bus. Elke slave heeft wel zijn eigen Slave Select signaal nodig.



SPI is niet het enige seriële communicatiesysteem dat in de lessen de revue gaat passeren. Er zijn nog tal van andere systemen : RS232 (tegenwoordig heet dat ANSI/EIA/TIA-232-F), I<sup>2</sup>C, USB, RS485, CAN,...

Elk van die systemen heeft zijn toepassingsgebied, voor- en nadelen. Sommige van die systemen hebben een hoog abstractieniveau en hebben een hoge instapdrempel (zoals USB). Anderen zitten op een laag abstractieniveau (zoals RS232). Een aantal van die systemen is *asynchroon*. Ze hebben geen kloklijn. Er wordt geen kloksignaal meegestuurd. Bij *asynchrone communicatie* moeten de verschillende partijen in het communicatieschema overeenkomen hoe snel er wordt gecommuniceerd. De snelheid kan vast ingesteld worden. Dat laatste doe je bij RS232. Daar stel je de *baudrate* in.

Op de onderstaande afbeelding zie je een SPI master met drie slaves.



Laat je bij het lezen van databladen niet in de war brengen : SPI is een bussysteem. ISP is een manier om microcontroller te programmeren. Dezelfde letters, andere volgorde. In de lessen zullen we gebruik maken van SPI om aan ISP te doen...

Meer weten? [Analog Devices over SPI](#)

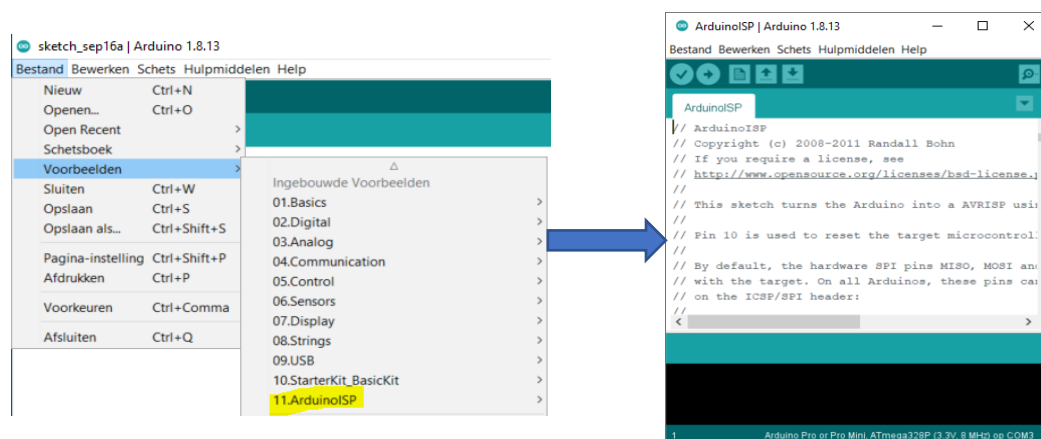
Ik heb geen ISP-programmer! Wat nu?

Heel eenvoudig. We vormen de Arduino Uno tot ISP-programmer. Voer hiervoor de volgende stappen uit.

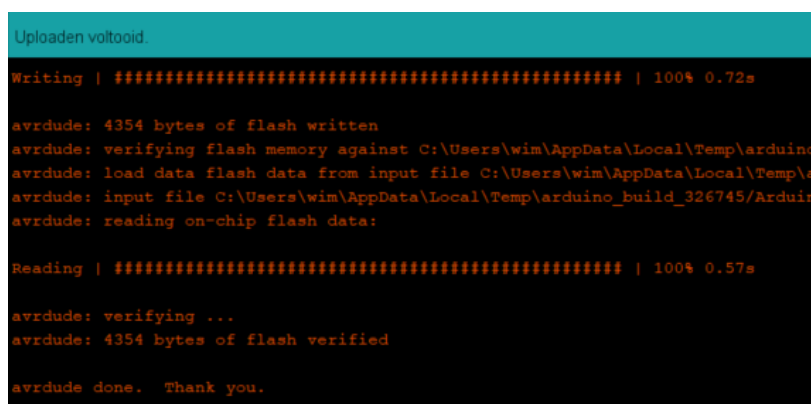
a. Start de Arduino IDE



b. Open de voorbeeldschets **11.ArduinoISP**



c. Compileer de schets en schrijf ze naar de Arduino Uno

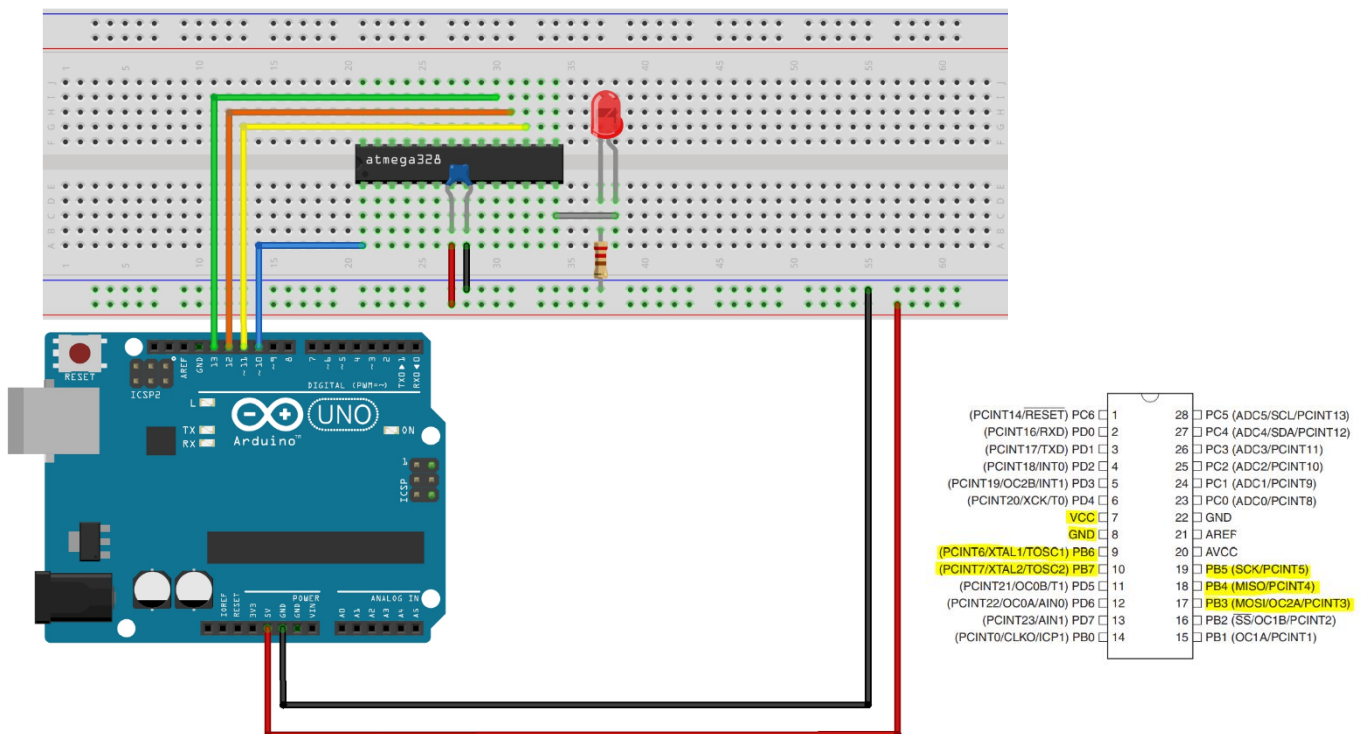


Als je de bovenstaande output ziet dan is jouw Arduino gepromoveerd tot ISP-programmer. We gaan deze programmer gebruiken in de Atmel Studio omgeving.

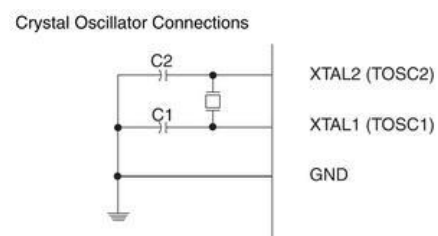
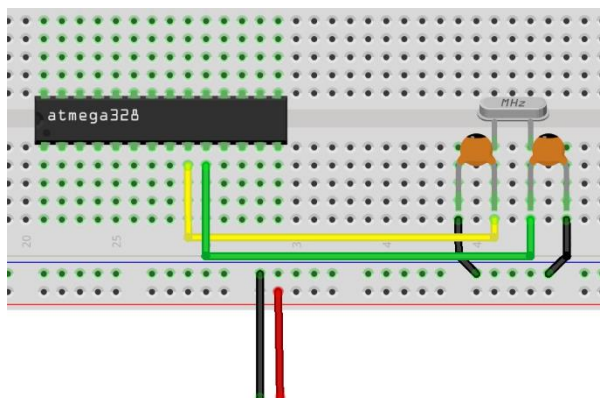
Hoe gebruik ik mijn Arduino Uno bordje als ISP-programmer?

Eerst verbindt je het Arduino-bordje met de losse AVR microcontroller. Een losse controller wordt geleverd met een geactiveerde 1 MHz interne klok. Wanneer dit niet zo is dan sluit je een extern kristal aan (bvb. 8 MHz).

Programmeren gebeurt via de SPI-bus. De aansluiting ziet er als volgt uit. Er is al een LED met PB0 verbonden om de code te testen:

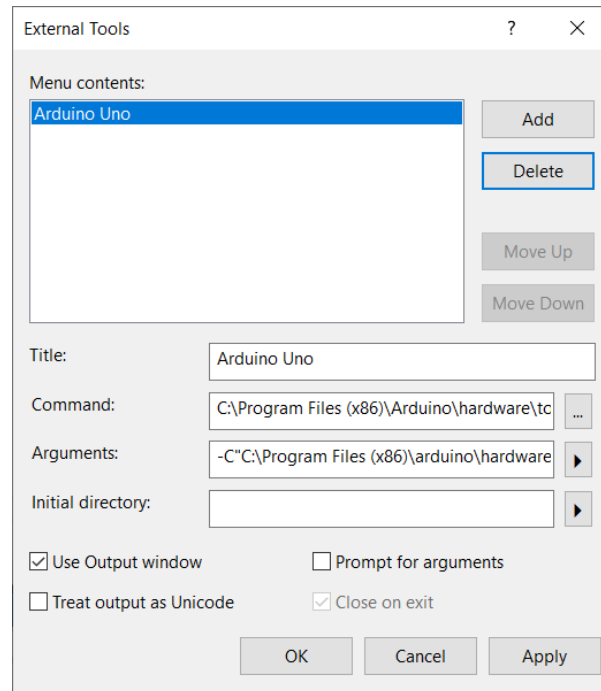


Als de interne klok van de controller die je wil programmeren niet is geactiveerd, dan sluit je een extern kristal aan. Op onderstaand schema is aangegeven hoe je dit doet. Voor de duidelijkheid is het kristal een eindje van de controller geplaatst. In werkelijkheid plaats je het kristal zo dicht mogelijk bij de controller. De keramische condensatoren zijn hebben een waarde van 22 pF.



Als de verbinding in orde is dan moeten we aan Atmel Studio duidelijk maken dat we de Arduino als programmeertoestel willen gebruiken. De Arduino met de ISP-sketch is een externe tool. We gaan deze tool zoals eerder in de lessenreeks gezien toevoegen via Tools.

Open het menu-item "Tools". Daar zie je al de eerder aangemaakte Arduino Uno staan.



Voeg met "Add" een tool toe. Als naam (title) kies je ArduinoISP. Bij "Command" komt het pad naar het avrdude.exe programma. Bij "Arguments" komen de command line argumenten die worden meegegeven bij het opstarten van avrdude.exe (zonder newline's)

Command:

C:\Program Files (x86)\Arduino\hardware\tools\avr\bin\avrdude.exe

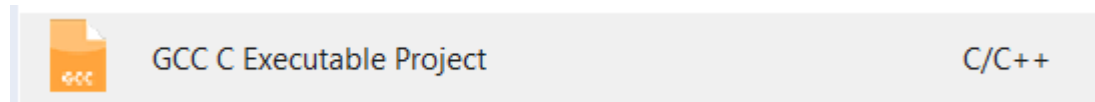
Arguments:

-U lfuse:w:0x62:m -U hfuse:w:0xd9:m -e -v -patmega328p -carduino -PCOM3 -b19200 -D -Uflash:w:"\$(ProjectDir)Debug\\$(ItemFileName).hex":i -C"C:\Program Files (x86)\arduino\hardware\tools\avr\etc\avrdude.conf"

Mogelijks moet je weer de COM-poort waarop de ArduinoISP is aangesloten aanpassen.

Wat mogelijks moet worden aangepast zijn de fuses (lfuse, hfuse). De lfuse is bij de argumenten ingesteld voor gebruik met een interne oscillator. Om deze waarden te bepalen bestaan er fuse calculators. Je kan zo'n calculator bekijken via volgende link : [AVR fuse calculator](#)

Maak vervolgens in Atmel Studio een nieuw project aan.



In het main.c bestand voorzie je de onderstaande code:

```
#define F_CPU 1000000
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    DDRB=0xFF;

    while (1)
    {
        PORTB^=0xFF;
        _delay_ms(250);
    }
}
```

Compileer deze code.

Kies in het "Tools" menu voor de ArduinoISP-tool. De RX en TX-leds van de ArduinoISP-programmer zouden moeten oplichten. Controleer vervolgens het output-venster in Atmel Studio. Als in het outputvenster geen foutmeldingen verschijnen dan is de chip correct geprogrammeerd. Je mag dan de SPI-verbinding met de ArduinoISP verbreken.

Heb je wel foutmeldingen dan ga je die stap voor stap moeten oplossen.

Op de Teams-site is een filmpje geplaatst. De L-LED op de ArduinoISP knippert mee met de rode LED die aangestuurd wordt door de pas geprogrammeerde ATmega328p. Kan je verklaren waardoor dit veroorzaakt wordt?

## Hoe kan een AVR serieel communiceren met een PC?

Bij het omvormen van het Arduino Uno bordje naar ISP-programmer hebben we al een *synchroon serieel protocol* bestudeerd : SPI (Serial Peripheral Interface).

In dit hoofdstuk bekijken we hoe we informatie kunnen uitwisselen tussen een AVR en een PC. De meest eenvoudige seriële communicatie die we daarvoor kunnen gebruiken is RS232 (ANSI/EIA/TIA-232-F).

### Wat is RS232 (ANSI/EIA/TIA-232-F)?

Oorspronkelijk (1969) een standaard voor trage seriële communicatie tussen terminals en modems

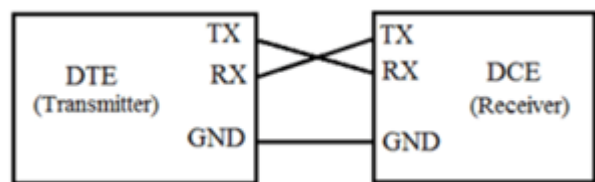


Terminal = Data Terminal Equipment (DTE)

Modem = Data Communications Equipment (DCE)

### Welke kenmerken heeft RS232 (ANSI/EIA/TIA-232-F)?

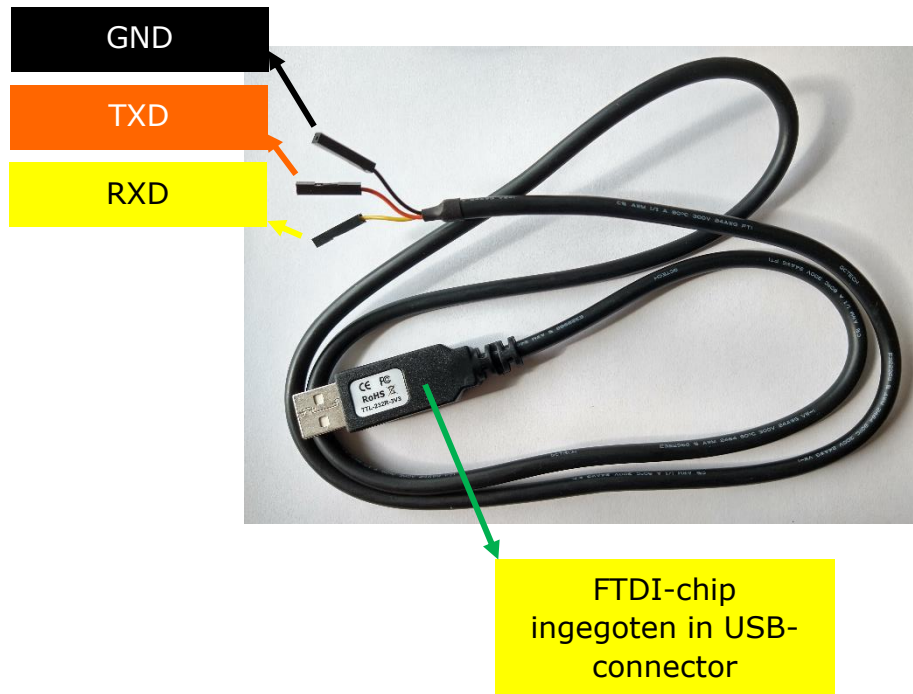
- Minimaal 3 draden
  - Zendlijn (TxD)
  - Ontvangstlijn (RxD)
  - Massa (GND)
- Bi-directioneel (**full duplex**)
- Lage snelheid ( max 115200 **baud**)
- Mogelijkheid tot **handshaking** via RTS/CTS lijnen



### RS232 (ANSI/EIA/TIA-232-F) in de 21<sup>e</sup> eeuw?

USB heeft RS232 verdrongen in PC-omgeving. Maar RS232 is verre van dood. Niet alle communicatie hoeft aan hoge snelheid te verlopen. RS232 blijft door zijn eenvoud een zeer interessant protocol. Jammer genoeg vind je nagenoeg geen PC's meer met een RS232-aansluiting

Het bedrijf Future Technology Devices International (FTDI) is op de kar gesprongen om RS232 een nieuw leven te geven. Ze bouwen FTDI RS232-USB omvormers. Zo kan je een microcontroller koppelen met een PC. De microcontroller communiceert via RS232 met een FTDI-chip. De FTDI-chip zet de communicatie om naar USB.



De drie draden van de FTDI kabel vormen de aansluiting van de DTE. Wanneer je hem aansluit aan een DCE ( $\mu$ C) dan kruis je de verbindingen :

TXD van DTE aan RXD van DCE

RXD van DTE aan TXD van DCE

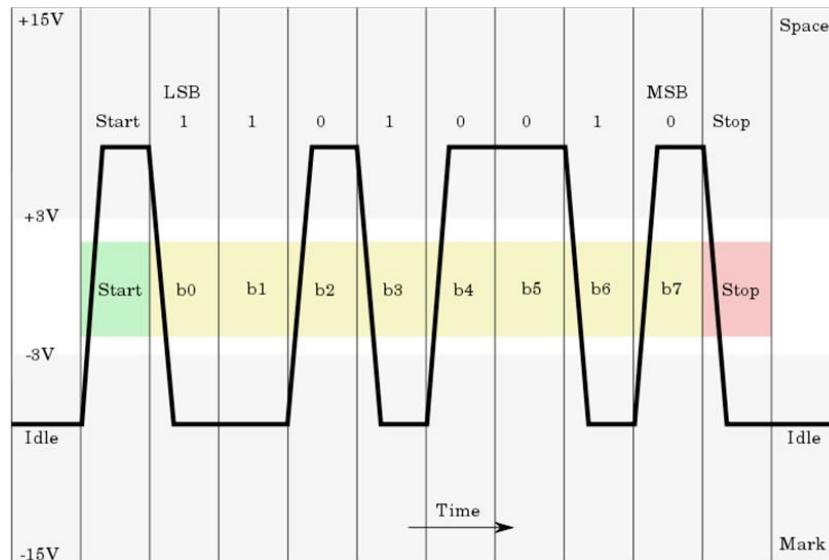
### Signaalverloop bij RS232

- Asynchroon → geen klok
  - zender en ontvanger moeten dezelfde transmissiesnelheid hanteren
  - Baudrate (baud)
  - Bitrate (bps)
  - Standaardsnelheden
  - Van 110 bps tot 115200 bps
  - Typische getallen: 2400bps, 9600 bps, 19200 bps
- NZR-principe (non return to zero)
  - Logische "1"= "mark", een negatieve spanning van -15 V
  - Logische "0"= "space", een positieve spanning van +15V
  - 0 V is geen geldig niveau
- Spanning tussen -3 V en +3 V zijn niet geldig
- De maximale spanning is +/- 25V
- UART (Universal Asynchronous Receiver Transmitter) zorgt voor opwekken van signalen

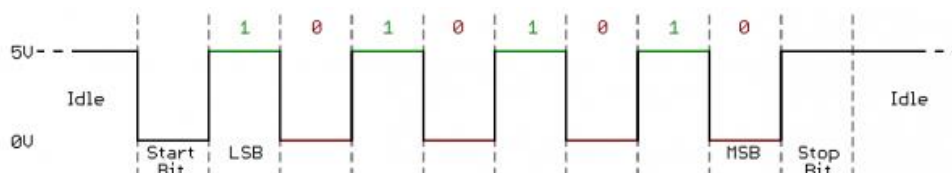
Voorbeeld:

Verzenden van één byte : letter 'K', ASCII-waarde = 0x4b = 0b01001011

- Communicatie start met een startbit
- Vervolgens 8 databits, LSb-first
- Communicatie wordt afgesloten met een stopbit



De AVR die we gebruiken werkt met TTL-niveaus (0 V, 5 V). Er wordt dus afgeweken van de originele standaard. In onderstaand tijddiagram zie je welke signaalniveaus er worden gebruikt. De idle-toestand is 0V. Een mark is 5V. Een space komt overeen met 0V. Welk ascii-karakter wordt er in onderstaand tijddiagram verzonden?



## Oefening

Op de Teams-site kan je een RS232-voorbeeldproject downloaden. Alle code om de AVR te laten communiceren met een PC is al voorzien. In het project is de code voorzien van voldoende commentaar zodat je snel de communicatie tussen AVR en PC op poten kan zetten.

In de opdracht op Teams (TSO\_AVR\_RS232\_LES3) wordt stap voor stap beschreven wat er van jou wordt verwacht.



## Hoe kan je bits manipuleren

### Probleemstelling

In de oefeningen heb je tot nu toe steeds gewerkt met bytes, woorden of andere datatypes. Wanneer je PORTB0 wou gebruiken als output voor je programma dan kon dit er in code als volgt uitzien:

```
...  
PORTB = 0b00000001;  
  
...  
PORTB = 0b00000000;
```

PINB0 wordt in het bovenstaande codefragment *hoog* gemaakt en enige tijd later terug laag. De andere bits van PORTB maak je telkens *laag*.

Wanneer je niet alleen PORTB0 gebruikt maar ook PORTB1 dan wordt het een probleem. Bij het schrijven van de ene pin ga je de toestand van de andere pin misschien veranderen. Dat is niet gewenst.

Hoewel de assembleertaal van de AVR in bit-instructies zoals CBI en SBI voorziet, beschikt de AVR-GCC C-taal niet over de mogelijkheden om individuele bits te manipuleren. Je moet er zelf voor zorgen dat dit goed komt. Gelukkig hebben we enkele instrumenten om onze code correct te laten werken:

- Schuifoperaties
- Bitgewijze logischeoperaties

### Schuifoperaties

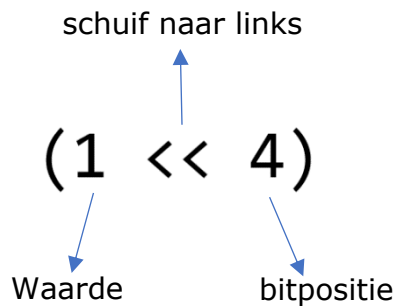
Stel dat ik in een byte op de vierde bitpositie van rechts een 1 wil hebben terwijl de rest van de bits 0 is, dan kan je de volgende code gebruiken:

```
uint8_t reg;  
  
reg = (1 << 4);
```

uint8\_t is een 8-bits unsigned type (een simpele byte dus). Let erop dat bitposities van rechts naar links genummerd worden. De meest rechtse bit in een byte, woord,... krijgt het volgnummer 0. De bitpositie komt overeen met de waarde die een bit in een byte, woord,... heeft. Onderstaande tabel

positie	7	6	5	4	3	2	1	0
bit	b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>
waarde	2 <sup>7</sup>	2 <sup>6</sup>	2 <sup>5</sup>	2 <sup>4</sup>	2 <sup>3</sup>	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>

We analyseren even de instructie (1<<4)



Je leest deze instructie als :

*"schuif de waarde 1 vanaf links naar positie 4 in een variabele."*

In het voorbeeld is de variabele een byte.

De waarde hoeft niet altijd een integer te zijn. Je kan ook andere getalvoorstellingen kiezen (hexadecimaal, binair). De onderstaande code doet hetzelfde als het vorige voorbeeld. Alleen geven we nu de waarde in binaire vorm mee.

```
uint8_t reg;  
  
reg = (0b1 << 4);
```

Je hoeft ook niet altijd de waarde 1 te schuiven. Ga na wat het effect is van de volgende instructies:

```
uint8_t reg;  
  
reg = (0b11 << 4);
```

## Bit-twiddeling

Hoe helpt dit bij het sturen van de IO van de microcontroller?

Stel dat we PORTB4 van de controller willen aansturen. De overige pinnen van PORTB gebruiken we ook, maar we willen alleen PORTB4 aansturen.

We hebben in de les gezien dat PORTB4 een macrodefinitie is. Je kan de macrodefinities voor de Atmega328p terugvinden in de volgende map:

C:\Program Files (x86)\Atmel\Studio\7.0\packs\atmel\ATmega\_DFP\1.3.300\include\avr\iom328p.h

Een fragmentje uit dit bestand ziet er als volgt uit:

```
#define PORTB_SFR_IO8(0x05)  
#define PORTB0 0  
#define PORTB1 1  
#define PORTB2 2  
#define PORTB3 3  
#define PORTB4 4  
#define PORTB5 5  
#define PORTB6 6  
#define PORTB7 7
```

PORTB4 is een macrodefinitie (#define). Telkens je PORTB4 in de code gebruikt zal onze compiler dit vervangen door het getal 4. Hierdoor wordt de code leesbaarder.

Stel dat alle bits van het PORTB-register als output zijn geconfigureerd en het PORTB-register de volgende waarde bevat:

```
PORTB = 0b11000111
```

Vijf uitgangen zijn hoog gestuurd (PORTB7, PORTB6, PORTB2, PORTB1 en PORTB0)

Wanneer we nu PORTB4 ook hoog willen sturen zonder de andere bits te raken dan kunnen we de volgende code gebruiken:

```
uint8_t reg;
DDRB = 0x11;

PORTB = 0b11000111;

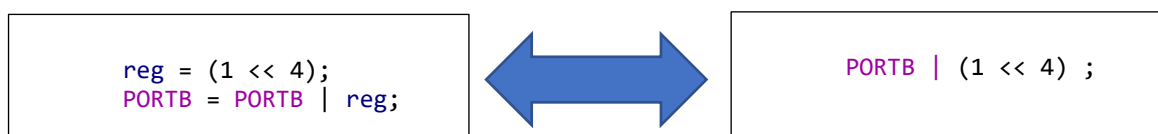
reg = (1 << 4);
PORTB = PORTB | reg;
```

In dit voorbeeld geven we PORTB de uitgangstoestand 0b11000111. Vervolgens zetten we in reg de bit op bitpositie 4 hoog. In het laatste statement voeren we een bitsgewijze of (bitwise OR) uit tussen PORTB en reg.

PORTB	1	1	0	0	0	1	1	1
reg	0	0	0	1	0	0	0	0
= PORTB	1	1	0	1	0	1	1	1

Op die wijze kunnen we één bit in een variabele gaan manipuleren terwijl de andere bits onberoerd blijven.

We kunnen de code nog vereenvoudigen:



De andere bitsgewijze operatoren zijn:

& → bitgewijze en (bitwise and)

^ → bitgewijze exor (bitwise xor)

Test eens de volgende statements uit :

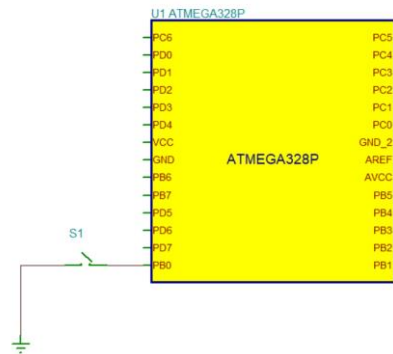
```
PORTB = 0b10101010;
PORTB ^= 0b11111111;
```

Het manipuleren van bits wordt in het Engels '*bit twiddeling*' genoemd.

## Hoe kan je in de AVR digitale ingangen gebruiken?

Wanneer je een pin van een poort van de AVR als digitale ingang wil gebruiken dan zijn er een aantal zaken waar je rekening mee moet houden.

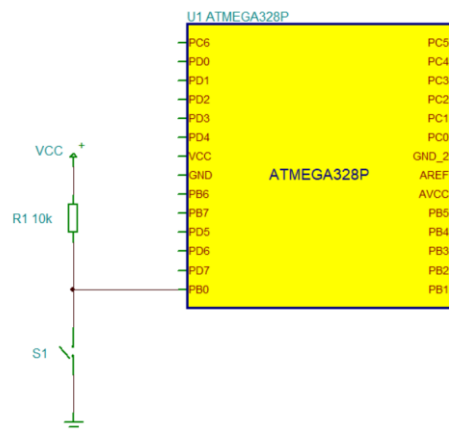
Als je een drukknop wil aansluiten aan een ingangspin dan zou je dit op de volgende wijze kunnen doen:



Als drukknop S1 gesloten wordt dan ligt PINB0 aan massa. Er is dan geen twijfel mogelijk over het signaal dat op PINB0 komt te staan.

Als drukknop S1 open is dan wordt het moeilijker. PINB0 is dan zwevend (Engels : 'floating'). Dat is geen goede toestand. De aansluiting naar de pin kan door ruis in een hoge of een lage toestand komen. Het is beter om een pin in een gekende toestand te hebben.

We kunnen zorgen dat er altijd een bekende toestand is door een *pull-up weerstand* te voorzien.



In de bovenstaande schakeling is R1 een pull-up weerstand. Wanneer S1 niet wordt ingedrukt is dan 'trekt' weerstand R1 PINB0 naar de positieve voedingsspanning. Vandaar de naam 'pull-up'.

Wanneer S1 wordt ingedrukt dan zal PINB0 via S1 verbonden worden met de massa.

Door gebruik te maken van een pull-up weerstand zal PINB0 altijd op een *ondubbelzinnig* spanningsniveau staan.

Voor de configuratie van een pin als ingang *volstaat het niet* om in het DDR-register van de betrokken poort de DDR-bit op '0' te zetten.

In de AVR zijn de poorten voorzien van een *interne pull-up weerstand*. Deze weerstand kan in- of uitgeschakeld worden. Het in- en uitschakelen gebeurt via het PORT-register. Je kan dus geen digitale inputs inlezen via het PORT-register.

Om digitale inputs in te lezen moeten we gebruik maken van het PIN-register van de betrokken poort.

Voorbeeld:

```
uint8_t teller;

//PINB0 als input configureren
DDRB &=0b11111110;

//Pull-up voor PINB0 inschakelen
PORTB=0b00000001;

//doe iets zolang PINB0 hoog is
while(PINB & (1<<PINB0))
{
    teller+=1; //plaats hier de code die je in de lus wil uitvoeren
}
```

In het bovenstaande voorbeeld zie je ook een toepassing van bit-twiddeling en van de schuifoperaties.

## Hoe gebruik ik in de C-taal pointers als functieparameters?

Je hebt pointers al in het vijfde jaar bestudeerd. We hebben dit concept in de les herhaald. In het onderstaande programmavoorbeeld kan je een aantal toepassingen van pointers terugvinden. In het labo zal je deze toepassing als oefening kunnen uittesten. Op die wijze leer je omgaan met het pointer-concept en kan je dit ook toepassen in je eigen programma's.

```
/*
 * main.h
 */

#include <avr/io.h>
#include "main.h"

int main(void)
{
    uint8_t ledbar = 0b00000001;
    char direction = 1;
    int teller = 0;
    char telrichting = 0xFF;

    initPorts();

    while (1)
    {
        /*
        TER INFO : WAT STELT 'PORTB' PRECIES VOOR IN DE CODE?

        PORTB gedefinieerd in "...\\Atmel\\Studio\\7.0\\packs\\atmel\\ATmega_DFP\\1.3.300\\include\\avr\\iom328p.h
        #define PORTB _SFR_IOR(0x05)

        _SFR_IOR() gedefinieerd in \\Atmel\\Studio\\7.0\\toolchain\\avr8\\avr8-gnu-toolchain\\avr\\include\\avr\\sfr_defs.h
        _SFR_IOR() vormt het I/O address om tot een geheugenaddress. Dit is een macro die de data op adres "io_addr +
        __SFR_OFFSET" retourneert

        #define __SFR_OFFSET 0x20
        #define _SFR_IOR(io_addr) _MMIO_BYTE((io_addr) + __SFR_OFFSET)

        _MMIO_BYTE() is een macro die de inhoud van een geheugenlocatie waar een volatile uint8_t naar wijst ophaalt (* =
        "dereferencing"). Deze macro is ook gedefinieerd in \\Atmel\\Studio\\7.0\\toolchain\\avr8\\avr8-gnu-
        toolchain\\avr\\include\\avr\\sfr_defs.h
        #define _MMIO_BYTE(mem_addr) (*(volatile uint8_t *) (mem_addr))

        De IO-registers zitten in het datageheugen van de Atmega328p controller vanaf adres 0x0020 tot 0x005F
        In het datablad DS40002061B op p.30 (8.5) wordt er uitgelegd welke assembleertaalinstructies werken op welke
        geheugenlocaties:
        "When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O
        Registers as data space using LD and ST instructions, 0x20 must be added to these addresses"

        In onze code gebruiken we de PORTB-macro om de inhoud van het register PORTB in de controller te manipuleren. De C-
        taal werkt met instructies die, uiteindelijk, uitwerking hebben op geheugenadressen. De PORTB-macro moet
        uiteindelijk de inhoud kunnen schrijven van een bepaalde geheugenlocatie. De C-compiler zet de C-code om naar
        machinecode die op de AVR kan draaien. Dat wil zeggen dat de compiler de C-code moet omzetten naar instructies die
        de AVR kan uitvoeren.
        */

        portChange(&PORTB, ledbar);

        //portChange((volatile uint8_t *) (0x25), ledbar); //zelfde uitwerking als
        portChange(&PORTB, ledbar);

        updateLedBar(&ledbar, &direction);

        if ( updateVariabele(&teller, 5, 10, telrichting) )
        {
            telrichting = ~(telrichting); //neem 1's complement van de waarde telrichting
        }
    }
}
```

```

/*char updateVariabele : increment of decrement van een variabele tussen een onder- en een bovengrens
de telrichting wordt aangegeven door de parameter richting.
richting = 0x00 : decrement
richting = 0xFF : increment

de returnwaarde geeft aan of de parameter tegen een grens is opgelopen
boundaryCheck = 1 : ondergrens bereikt
boundaryCheck = 2 : bovengrens bereikt
*/

char updateVariabele(int* variabele, int ondergrens, int bovengrens, char richting)
{
    char boundaryCheck=0;

    if (richting == 0x00)        //decrement
    {
        if (*variabele > ondergrens)
        {
            *variabele = *variabele - 1;
        }
        else
        {
            boundaryCheck = 1; //melding ondergrens bereikt
        }
    }

    if (richting == 0xFF)        //increment
    {
        if (*variabele < bovengrens)
        {
            *variabele = *variabele + 1;
        }
        else
        {
            boundaryCheck = 2; //melding bovengrens bereikt
        }
    }

    return boundaryCheck;
}

void portChange(volatile uint8_t *port, uint8_t value)
{
    *port = value;
}

void initPorts()
{
    DDRB=0xFF;        //PORTB alle pinnen output
}

void updateLedBar(uint8_t* ledbar, char* direction)
{
    if (*direction == 1)
    {
        if (!(*ledbar & 0b10000000))
        {
            *ledbar <<= 1;
        }
        else
        {
            *ledbar >>= 1;
            *direction = 0;
        }
    }
    else
    {
        if (!(*ledbar & 0b00000001))
        {
            *ledbar >>= 1;
        }
        else
        {
            *ledbar <<= 1;
            *direction = 1;
        }
    }
}

```

```
/*
 * main.h
 */

#ifndef MAIN_H_
#define MAIN_H_
#define F_CPU 8000000L

char updateVariabele(int* variabele, int ondergrens, int bovengrens, char richting);
void portChange(volatile uint8_t *, uint8_t);
void initPorts();
void updateLedBar(uint8_t* ledBar, char* direction);

#endif /* MAIN_H_ */
```



Hoe kan ik met de AVR analoge signalen meten?

## Wat zijn interrupts?

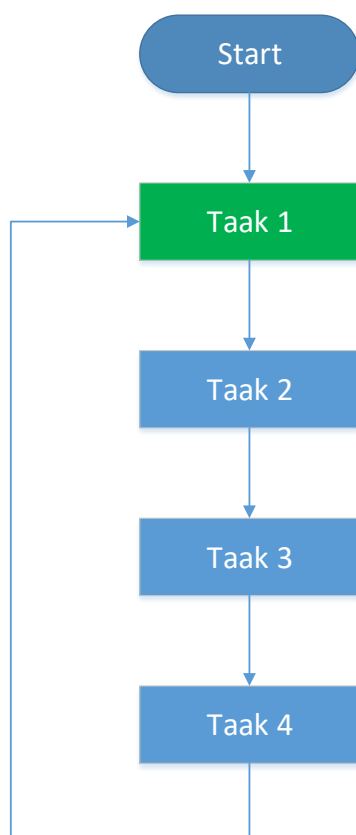
### Probleemstelling

Een microcontroller communiceert met meerdere randapparaten :

- Temperatuursensor waarvan de ADC elke 500 ms een nieuwe temperatuurwaarde heeft;
- Host die via seriële verbinding commando's stuurt naar de microcontroller;
- Drukknop die, zodra ingedrukt, een resetactie als gevolg moet hebben;
- LED die moet knipperen aan een frequentie van 1 Hz.

Elk randapparaat vraagt op regelmatige tijdstippen en gedurende een zekere tijd aandacht

De microcontroller kan in een oneindige lus de randapparaten gaan bevragen of besturen. In dit voorbeeld zijn er 4 taken uit te voeren:



- **Taak 1** : Als er een temperatuurwaarde ter beschikking is, lees die dan uit.
- **Taak 2** : Als er via de host een commando binnenkomt voer dit commando dan uit.
- **Taak 3** : Als de drukknop ingedrukt is voer dan de reset-actie uit.
- **Taak 4** : Laat de LED knipperen aan een frequentie van 1 Hz.

### Nadelen polling:

- Geen prioriteit in de taken.
- Taakuitvoering kan lang duren.
- Dringende acties krijgen geen gevolg.

→ **Polling** is goede oplossing voor toepassingen met gelijkwaardige en/of niet tijd kritische taken.

### Eisen gesteld aan een oplossing

- Taken moeten prioriteit kunnen krijgen.
- Randapparaat moet om aandacht van processor kunnen vragen.
- Microcontroller moet lopende taken kunnen onderbreken.
- Microcontroller moet code kunnen uitvoeren ten behoeve van randapparatuur

### OPLOSSING : Interrupt (=onderbrekingsaanvraag)

Wanneer de in de probleemstelling beschreven toepassing gebruikt maakt van interrupts dan zal de aangesloten randapparatuur aandacht kunnen vragen van de processor. De processor reageert hierop door het uitvoeren van code ten behoeve van de randapparatuur.

- De ADC geeft *een signaal* dat er een temperatuurwaarde klaar is om uitgelezen te worden.
- De USART geeft *een signaal* dat er data ontvangen is via RS232.
- Het indrukken van de drukknop geeft *een signaal* aan de microprocessor van de  $\mu C$ .
- Een interne timer geeft *een signaal* wanneer een bepaald tijdsinterval is afgelopen.

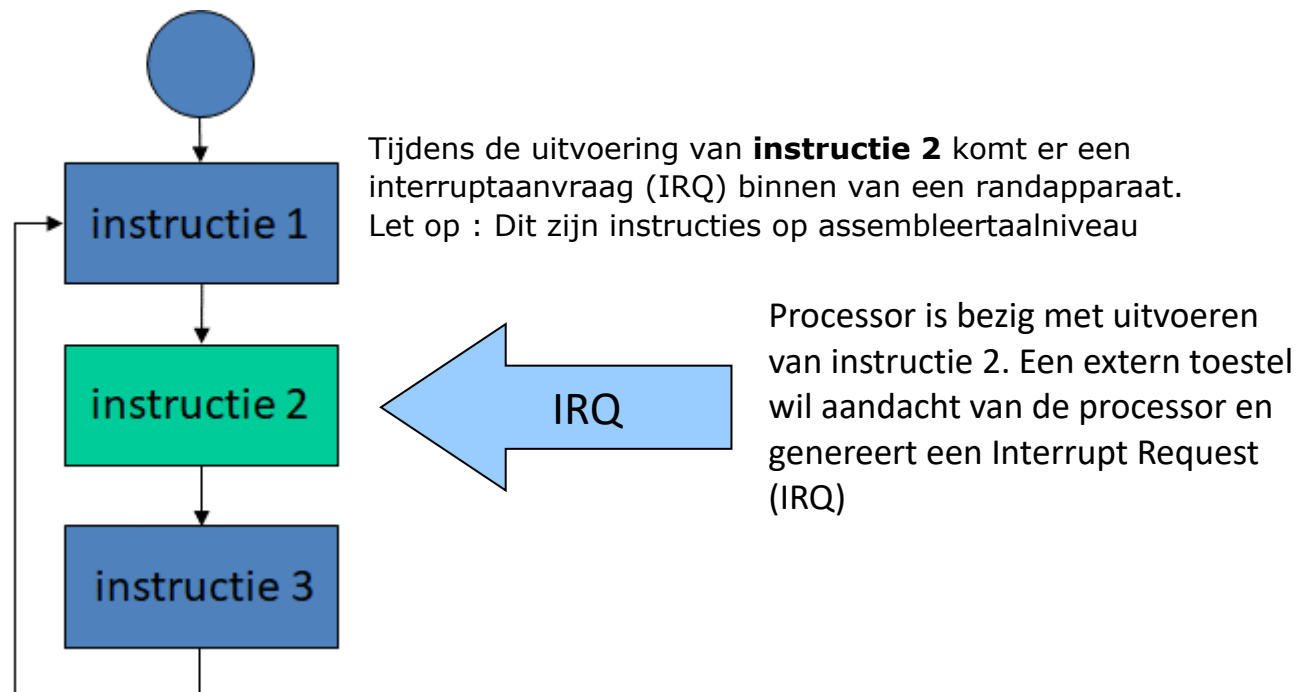
Randapparaten signaleren een verandering in hun toestand aan de microprocessor in de  $\mu C$ . De microprocessor reageert op deze signalen door zijn normale werking te onderbreken en de taak uit te voeren die voor het betreffende randapparaat in de code is voorzien:

Het **signaal** is een onderbrekingsaanvraag  
→ **interrupt request (IRQ)**

De **taak** die wordt uitgevoerd is een  
→ **interrupt service routine (ISR)**

## Verloop van interruptafhandeling

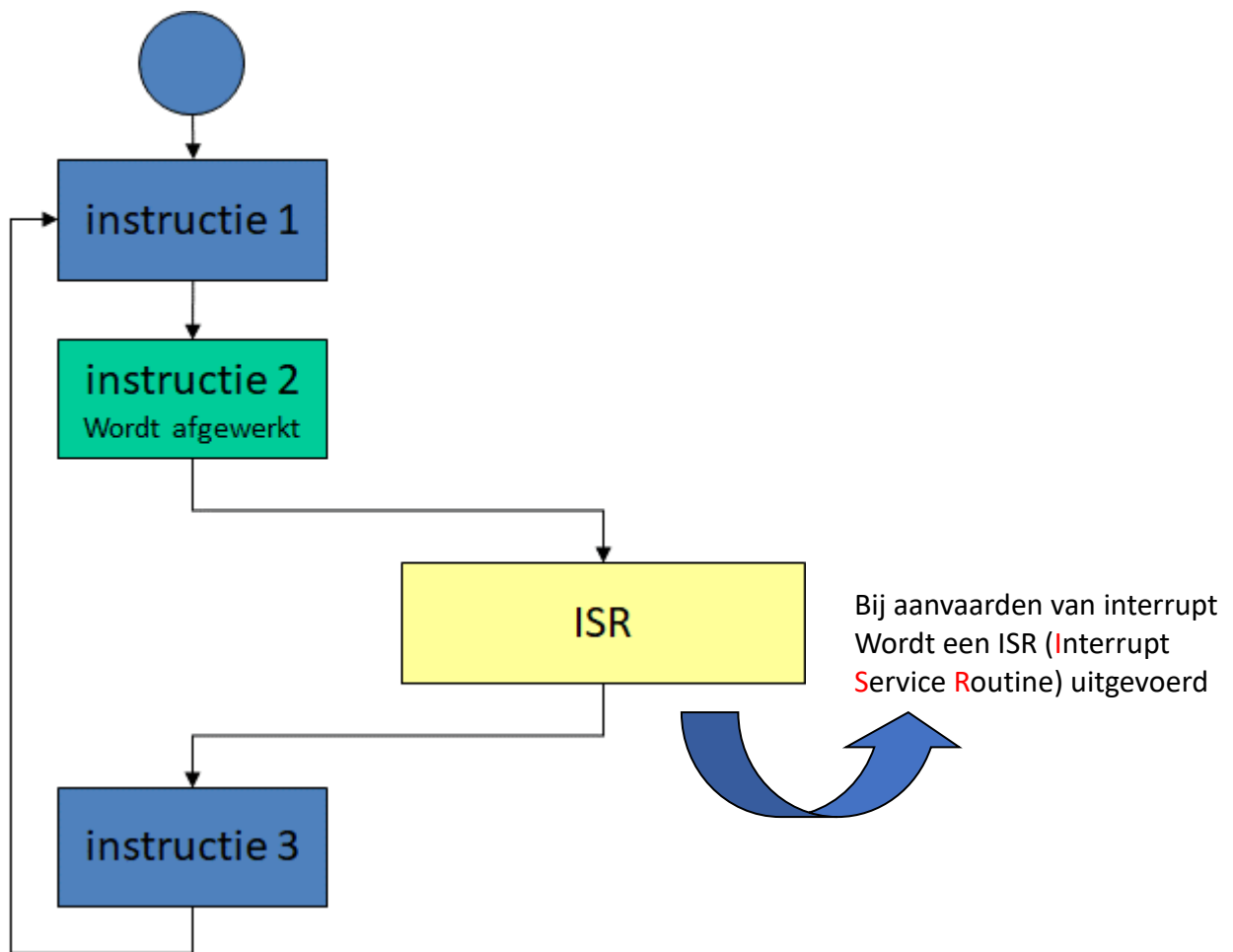
Normale programmaverloop : in main wordt een oneindige lus uitgevoerd



De microprocessor werkt de lopende instructie af. Wanneer de instructie is afgewerkt zal de microprocessor de ISR (Interrupt Service Routine) uitvoeren. Hiervoor wordt hetzelfde principe gebruikt als bij de afhandeling van een subroutine, functie of procedure.

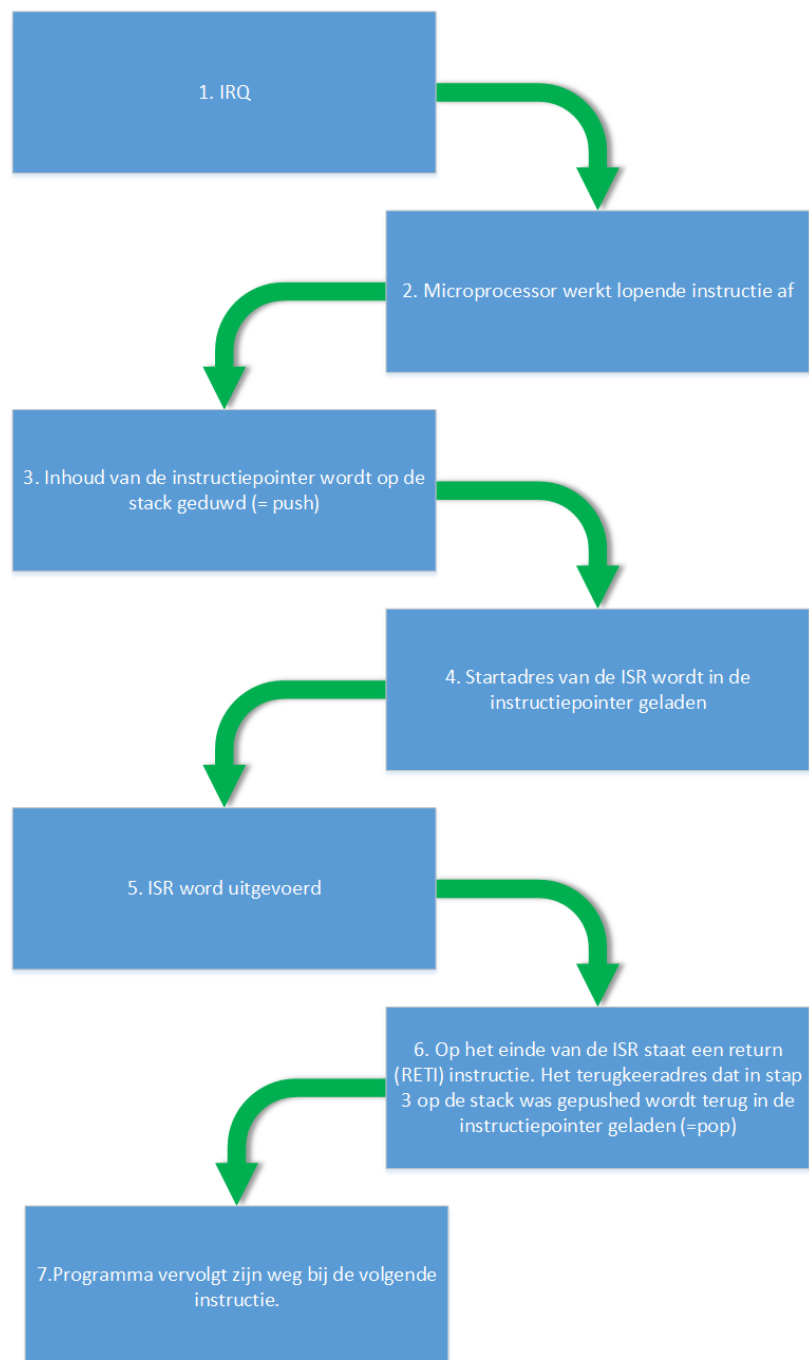
Er is echter *geen expliciete call* nodig. Het interruptmechanisme zorgt ervoor dat de bij het randapparaat horende ISR automatisch wordt uitgevoerd zodra de microprocessor er klaar voor is.

Uitvoering ISR : normale programmaverloop wordt onderbroken



De afhandeling van een interruptaanvraag van een randapparaat lijkt op de afhandeling van een subroutine. Een ISR wordt echter uitgevoerd zonder dat er een expliciete call-instructie naar de ISR in de code staat.

## Schema : afhandeling van een IRQ



## Interrupt in de ATmega328p - mechanisme

- Een **IRQ** is gekoppeld aan een **interrupt-vector**
- Een **interrupt-vector** is een **pointer**
- Het **adres** waar de interrupt-vector naar wijst is het startadres van de **interrupt service routine (ISR)**
- De ATmega328p heeft **26 interrupt-vectoren**

Table 12-6. Reset and Interrupt Vectors in ATmega328 and ATmega328P

VectorNo.	Program Address <sup>(2)</sup>	Source	Interrupt Definition
1	0x0000 <sup>(1)</sup>	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2_COMPA	Timer/Counter2 Compare Match A

De bovenstaande tabel toont enkel de eerste 8 interruptvectoren. In Tabel 12-6 van het datablad kan je alle andere interruptvectoren terugvinden.

	Address	Labels	Code	Comments
2	0x0002	INT0	0x0000 jmp RESET	; Reset Handler
3	0x0004	INT1	0x0002 jmp EXT_INT0	; IRQ0 Handler
4	0x0006	PCINT0	0x0004 jmp EXT_INT1	; IRQ1 Handler
5	0x0008	PCINT1	0x0006 jmp PCINT0	; PCINT0 Handler
			0x0008 jmp PCINT1	; PCINT1 Handler
			0x000A jmp PCINT2	; PCINT2 Handler

Als een randapparaat een IRQ verzendt dan zal de microcontroller de bij die IRQ horende ISR uitvoeren.

Voorbeeld:

Er is voor PD3 (=INT1) **door de programmeur (correct) een interrupt geconfigureerd** wanneer er zich op PD3 een stijgende flank voordoet. Men gebruikt dus interrupt-vector 3. Deze vector wijst naar het adres 0x0004 in het programmeergeheugen van de controller. Op dat adres zet de programmeur een sprong naar de betreffende interruptroutine.

(PCINT14/RESET) PC6 ☐ 1  
(PCINT16/RXD) PD0 ☐ 2  
(PCINT17/TXD) PD1 ☐ 3  
(PCINT18/INT0) PD2 ☐ 4  
(PCINT19/OC2B/INT1) PD3 ☐ 5  
(PCINT20/XCK/T0) PD4 ☐ 6  
VCC ☐ 7

```
EXT_INT1:
; de assembleertaalcode die hier staat wordt uitgevoerd.
; als je INT1 interrupt correct hebt ingesteld.

; ...

; De laatste instructie is de reti instructie. De programma-
; uitvoering gaat verder na de laatst uitgevoerde instructie
; in het hoofdprogramma.

reti
```

## Interrupt in de ATmega328p – configuratie

- De programmeur stelt de interrupts in : Dit komt neer op het *juist* instellen van de *juiste* bits in de *juiste* registers
- De programmeur schrijft de ISR : Een ISR is een subroutine (assembleertaal) of een functie (C, Arduino)

Voorbeeld:

Externe interrupt via PD3 (=INT1)

### EICRA – External Interrupt Control Register A

The External Interrupt Control Register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 13-1. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

### EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

```
sei();          /*set global interrupt enable bit
```



## Interrupt in de ATmega328p – frame voor C-code

```
/* de nodige headers toevoegen aan de code
#include ...

ISR(<vectornaam>)
{
    /* In dit codeblok komt de code die moet worden uitgevoerd wanneer de aan
    <vectornaam> gekoppelde interruptbron een IRQ geeft. Deze functie heeft als naam altijd
    ISR en neemt als argument de vectornaam van de interrupt*/
}

void init<vectornaam>(void)
{
    /* In dit codeblok regel je het juist instellen van de juiste bits in de juiste registers. De
    naam is vrij te kiezen. We nemen een combinatie van init + de vectornaam */
}

int main(void)
{
    /*initialisatieroutines voor IO, USART, ADC,...*/
    ...
    init<vectornaam>;
    sei();
    while(1)
    {
        /* in deze lus wordt er gewacht tot er zich een IRQ voordoet */
    }
}
```

## Hoe kan ik de Timer-modules in de AVR gebruiken?

### Toepassingen van timers en tellers

De AVR-microcontroller is voorzien van de nodige hardware om nauwkeurig tijdsintervallen te meten: De Timer/Counter modules. Afhankelijk van het type microcontroller heb je de beschikking over één of meerdere van dergelijke modules. Deze modules kan je onder andere gebruiken bij de volgende toepassingen:

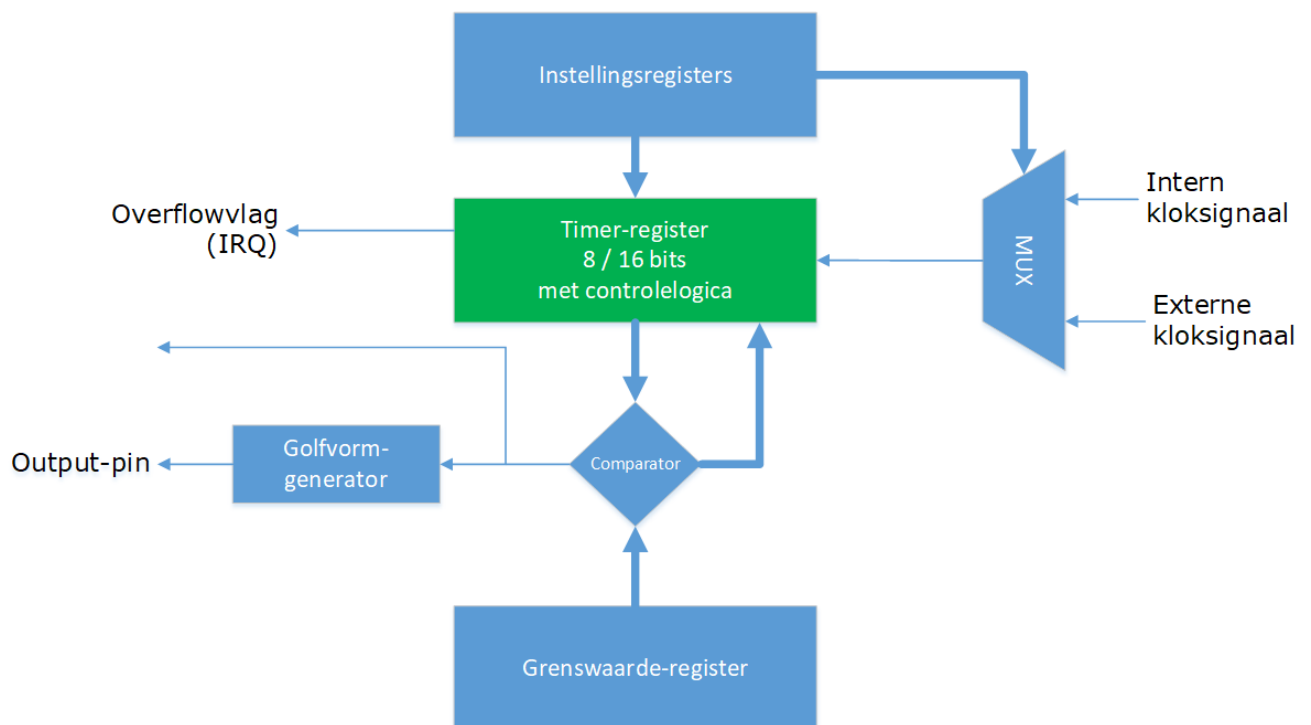
- Meten van verstreken tijd
  - Meten van de lengte van een puls
  - Digitale klok
  - Meten van de tijdsconstante van een condensator
- Uitvoeren van taken op tijdsbasis
  - Samplefrequentie bij meten van analoge signalen
  - Taakschema bij eenvoudig real time besturingssysteem
- Opwekken van golfvormen
  - Blok golf met variabele frequentie
  - PWM-signalen
- Tellen van pulsen
  - Een snelheidsmeter voor een fietscomputer (meten van puls/omwenteling)

## De generieke Timer/Counter

De timermodule in een microcontroller is een behoorlijk complexe module. Het valt buiten het bestek van deze cursus om elk detail ten gronde uit te leggen. Als alternatief is er gekozen om de werking van een *generieke* timer te beschrijven. Deze generieke beschrijving kan je dan gebruiken om voor een specifieke microcontroller de correcte werkingssmode en daarbij horende instellingen in het datablad op te zoeken.

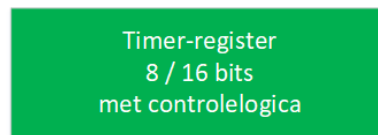
### Blokschema van een generieke timer

Het blokschema van een generieke timer geeft de verschillende elementen weer waaruit een microcontroller-timer is opgebouwd. Afhankelijk van de mogelijkheden van de timer kunnen bepaalde functies afwezig zijn of extra functies zijn toegevoegd. Het datablad geeft daarover meer informatie.



### Het Timer-register

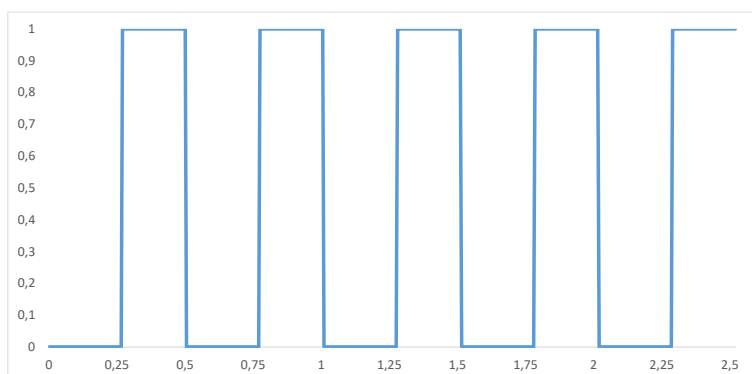
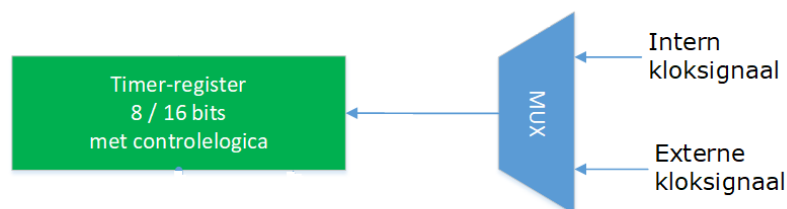
Centraal in de timermodule staat het Timer-register. De bitbreedte van dit register is typisch 8 of 16 bits. Een doorsnee microcontroller heeft meestal zowel een 8- als een 16 bits timer aan boord.



Een register is een geheugenlocatie in een microprocessor. Registergeheugen is in de regel uitgevoerd als snel geheugen (SRAM, statische RAM). In een register kan er opgeslagen worden. Uit een register kan er data gelezen worden. Een register heeft dus niet uit zichzelf de mogelijkheid om zomaar te veranderen. De inhoud van een register wijzigt als er iets wordt in weggeschreven.

### De controle-logica

Een Timer-register zal dus niet uit zichzelf de tijd of pulsen tellen. Hiervoor is nog controlelogica nodig. Deze controlelogica is in staat om de toestand van een intern of extern kloksignaal te monitoren. De module beschikt hiervoor over een multiplexer die aangestuurd wordt vanuit de instellingsregisters. Bij wijziging van hetingangssignaal, zoals een stijgende of een dalende flank, zal de controlelogica de inhoud van het timerregister wijzigen.



De blokgolf in de afbeelding kan dienst doen als kloksignaal voor de timer. Dit signaal heeft stijgende en dalende flanken. Een timer kan geconfigureerd worden om de stijgende of de dalende flanken te tellen.

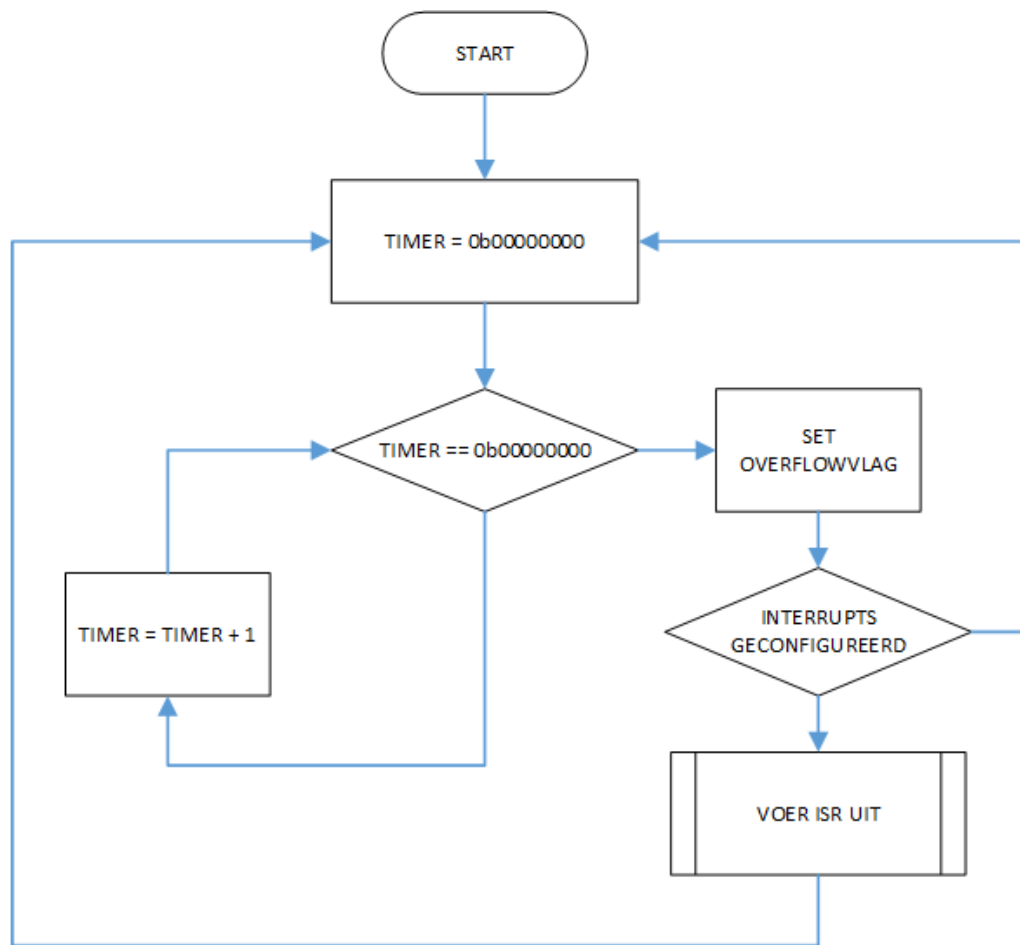
Bij elke stijgende flank zal de controlelogica de timer met één verhogen. In onderstaande tabel zijn de opeenvolgende timerwaarden weergegeven die een 8-bits timer zal aannemen bij het gegeven kloksignaal. De timer start bij de waarde 0b00000000 :

Tijd	TIMER
0,25	00000000
0,75	00000001
1,25	00000010
1,75	00000011
2,25	00000100
2,75	00000101
3,25	00000110
...	
124,75	11111001
125,25	11111010
125,75	11111011
126,25	11111100
126,75	11111101
127,25	11111110
127,75	11111111

#### Timer-overflow

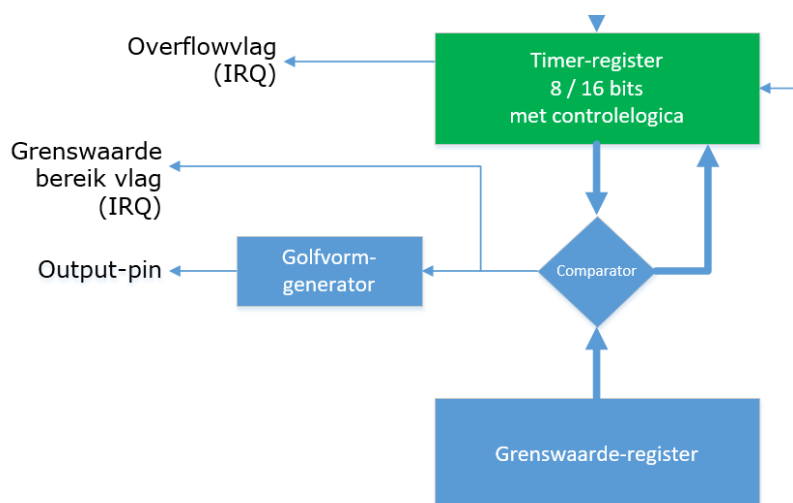


Wanneer de timer de waarde 0b11111111 bereikt dan zal de volgende kloktik leiden tot een overflow. De timer zal dan terug de waarde 0b00000000 hebben en de overflowvlag wordt geset. Deze vlag kan gebruikt worden in een polling routine of om een IRQ te activeren. De onderstaande flowchart geeft weer hoe het mechanisme werkt.



### Compare-module

In normale modus werkt een timer als free run counter. Hij telt van 0 tot een maximum telt en zet bij het omkappen naar 0 een overflowvlag. Vaak is dit onvoldoende en willen we ook de mogelijkheid om een vlag te zetten wanneer een andere waarde dan de maximale waarde wordt bereikt.



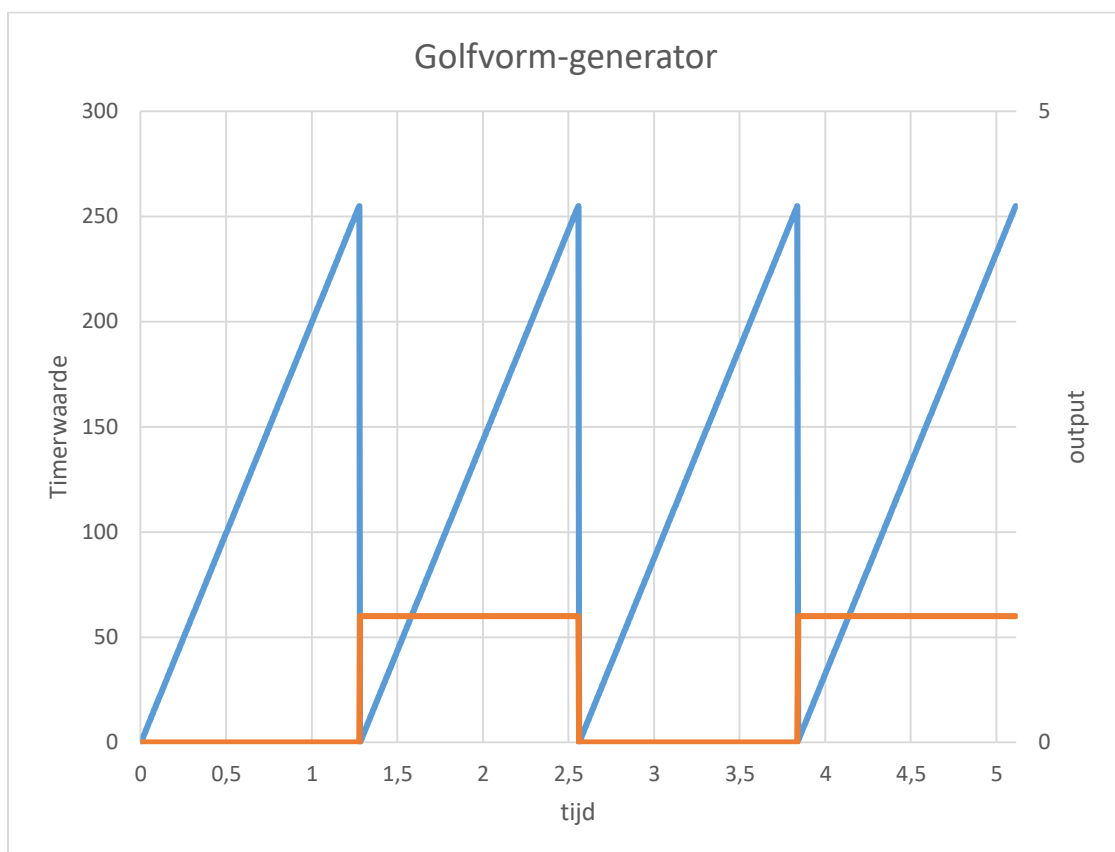
De comparatorfunctionaliteit is in de timermodule ingebouwd. Als programmeur kan je de timer configureren zodat de inhoud van het timerregister vergeleken wordt met de inhoud van een grenswaarde register. Wanneer de timerwaarde gelijk is aan de grenswaarde kan er een vlag geset worden. Deze vlag kan aanleiding geven tot een IRQ. Deze vlag kan ook gebruikt worden om een golfvorm-generator aan te sturen.

#### *Golfvorm generator*

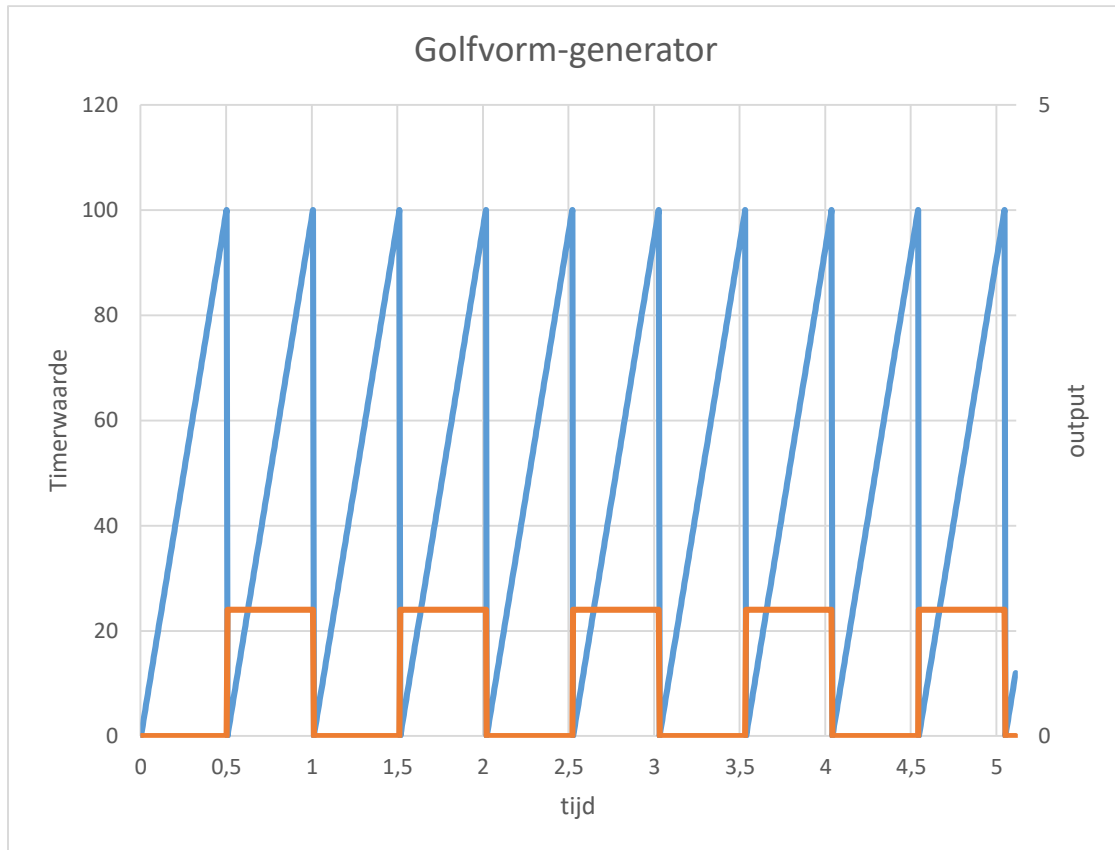
Een microcontroller kan gebruikt worden om golfvormen of PWM-signalen te genereren. Hiervoor kan gebruik gemaakt worden van een golfvorm-generator.

*Voorbeeld:*

*Men kan de timer zodanig configureren dat het timerregister gereset wordt zodra de grenswaarde wordt bereikt. Stel dat we een 8-bits timer hebben dan kan die timer – wanneer we hem vrij laten tellen – van 0 tot 255 tellen. Bij de overgang van 255 naar 0 wordt de overflow-vlag geset. Je kan de golfvorm-generator zodanig instellen dat de output-pin van de golfvormgenerator toggelt bij elke overflow van de timer. Dit is weergegeven in onderstaande grafiek.*



Wanneer er een grenswaarde wordt ingesteld die lager is dan de maximale waarde die de timer kan bevatten dan zal de door de golfvorm-generator opgewekte blokgolf een hogere frequentie hebben. In onderstaande afbeelding is als grenswaarde 100 ingesteld. De frequentie van het output-sigitaal van de golfvorm-generator zal dan toenemen.



#### Opdracht

Gebruik het Atmega328p-gegevensblad p. 74-115 en de kennis uit de les om een samenvatting te schrijven over hoe de timermodule geconfigureerd moet worden om een blokgolf met een variabele frequentie te genereren.

Een derde partij met kennis van stappenmotoren moet aan de hand van de samenvatting AVR-software kunnen schrijven waarmee de rotatiesnelheid van de motor kan worden geregeld.

Je kiest zelf of je gebruik maakt van een 8-bits of 16-bits timer. De frequentie-instelling wordt vanaf een toepassing op een host-PC via RS232 doorgestuurd naar de microcontroller.

Je meet het signaal met een oscilloscoop.