

Embedded C

Week 4: Sound – C: Structures - Timers

Overzicht

- Week 1: Hello Arduino - Basics - LED
- Week 2: Buttons - Interrupts
- Week 3: Potentiometer – the LED Screen
- Week 4: Timers - Sound
- Week 5: Integratie en individueel project
- Week 6: Integratie en individueel project

Content

- Overzicht Week 4
- Demo 1 – Play music
- Demo 2 – C: Structs
- Demo 3 – Timers
- Week project

Week4 - Embedded

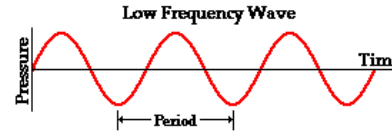
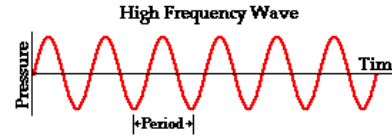
- ↪ Overzicht
- ↪ Tutorials
- ↪ Demos
- ↪ Oefeningen
- ↪ Weekproject

Week 3 - Overzicht



Overzicht

- De buzzer: muziek maken



- Timers: automatische tellers



- De programmeertaal C:
 - structs

THE
C
PROGRAMMING
LANGUAGE

Week 4 - Tutorials



Overzicht

1 – Interfacing met de Hardware

1.1 – Geluid- Muziek

1.2 – Timers

2 – De Programmeertaal C

2.1 – Structs

3 – Overige Tutorials

3.1- Een piëzo-elektrische luidspreker

Week 4 - Demos



Overzicht

1 – Play music

In deze demo leren we hoe je een muzieknoot uit de buzzer van je multifunctional shield krijgt.

2 – C: Structures

In deze demo maken we een struct KAART en gebruiken die om een boek van speelkaarten te maken en daaruit 10 willekeurige kaarten te trekken.

3 – C: Structures dynamisch

Als je de vorige demo "C: structures" goed begrepen hebt dan is het niet zo moeilijk om die code zodanig aan te passen dat de reservaties op de heap gebeuren. Dynamisch reserveren dus!

4 – Timers

Herinner je de oefening nog waarin je LEDs probeerde te laten dimmen? Hier een demootje waarin Timer 0 gebruikt wordt om LEDs te dimmen.

Week 4 - Oefeningen



Oefening



Overzicht

Mijn synthesizer

We kunnen de Arduino gebruiken om een willekeurige toon te produceren, onze eigen kleine synthesizer.

Wolfgang Arduino Mozart

We schrijven een programmaatje dat een muziekstukje genereert. Druk je op een knop, dan genereert je Arduino een willekeurig muziekje en speelt het 4 keer na elkaar af!

Stopwatch maken

We willen een stopwatch bouwen die begint te tellen vanaf er op S1 gedruwd wordt totdat er op S2 gedruwd wordt. S3 reset de stopwatch. De minuten worden getoond op de linker twee display segmenten en de seconden op de rechter twee display segmenten.

We gaan Timer 0 gebruiken om exact 1 seconde (1000 ms) te genereren.

Week 4 - Weekproject



Opdracht










Overzicht

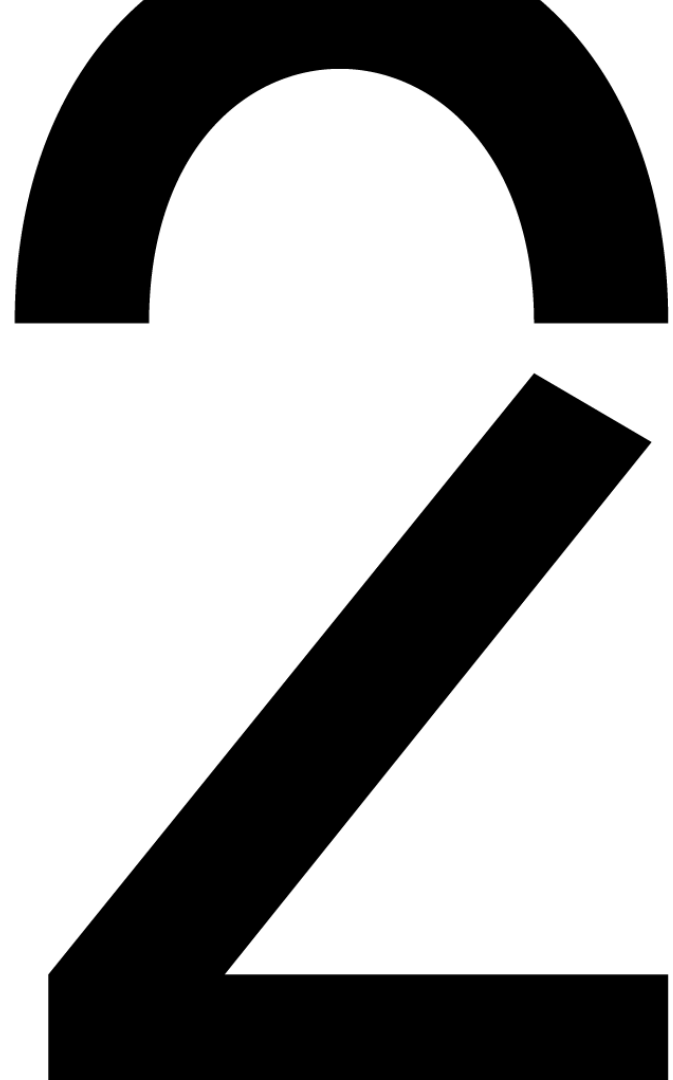
*In het kader van hun Artemis programma, startte NASA in mei 2020 een 'wedstrijd' tussen Blue Origin, SpaceX en Dynetics om de next generation Lunar Landers te mogen bouwen. Midden april 2021 koos NASA voor SpaceX en hun Human Landing System (HLS) -gebaseerd op hun herbruikbaar raketsysteem 'Starship'-. Blue origin en Dynetics stribbelen nog wat tegen waardoor begin mei 2021 het project even werd stilgelegd, maar uiteindelijk kon SpaceX aan de slag.. Na je informatica-studies solliciteerde je bij SpaceX en kort daarop werd je door SpaceX geselecteerd om deel uit te maken van het team dat met de '**SpaceX HLS 1.0**' in 2024 een bezoekje brengt aan de maan. Alles verloopt prima, de automatische landing op de maan werd ingezet, maar op 9999 meter hoogte boven het maanoppervlak, valt de computer die de 'SpaceX HLS 1.0' veilig aan de grond moest zetten uit. Een softwarebug? Een losse netwerk- of stroomkabel? Geen tijd om de software te herstarten noch te debuggen of de hardware te controleren, je moet de landing manueel en met maar enkele gegevens tot een goed eind brengen! Iedereen aan boord rekent op jou!*



Niet vergeten: Portfolio

- ▼  arduino
 - >  libraries
 - >  Project Week 1 - MorseCode
 -  Project Week 2 - Simon
 -  Project Week 3
 -  Project Week 4
 -  Project Week 5 & 6

Demo 1 – Play music with Buzzer



Demo 1 – Play music - C-code

```
#define __DELAY_BACKWARD_COMPATIBLE__
#include <avr/io.h>
#include <led.h>
#include <usart.h>
#include <util/delay.h>

#define DURATION 250 //DURATION IS IN MILLISECONDEN

//FREQUENCIES VAN DE VERSCHILLENDE NOTEN
#define C5 523.250
#define D5 587.330
#define E5 659.250
#define F5 698.460
#define G5 783.990
#define A5 880.000
#define B5 987.770
#define C6 1046.500

void enableBuzzer() {
    DDRD |= (1 << PD3); //buzzer hangt op PD3
}
```

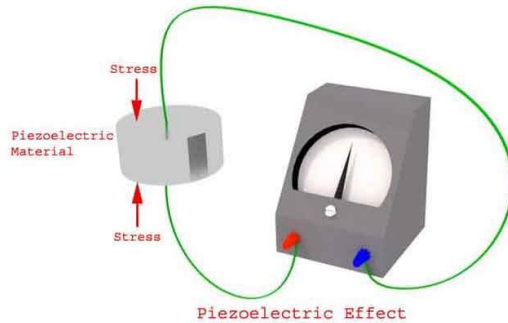
Demo 1 – Play music - C-code

```
void playTone(float frequency, uint32_t duration) {
    uint32_t periodInMicro = (uint32_t)(1000000/frequency); //we berekenen de periode in microseconden uit de frequency
    uint32_t durationInMicro = duration * 1000; //we willen duration in microseconden
    for (uint32_t time = 0; time < durationInMicro; time += periodInMicro) { //Zie tutorial over muziek voor meer uitleg!
        PORTD &= ~(1 << PD3); //aanzetten van de buzzer
        _delay_us(periodInMicro / 2); //helpt van de periode wachten
        PORTD |= (1 << PD3); //afzetten van de buzzer
        _delay_us(periodInMicro / 2); //weer helpt van de periode wachten
    }
}

int main() {
    initUSART();
    uint32_t frequencies[] = {C5, D5, E5, F5, G5, A5, B5, C6}; // do-re-mi...
    enableBuzzer(); enableOneLed(1); lightDownOneLed(1);
    while (1) {
        for (int note = 0; note < 8; note++) {
            playTone(frequencies[note], DURATION);
            printf("Period: %lu\n", frequencies[note]);
            lightUpOneLed(1); _delay_ms(DURATION);
            lightDownOneLed(1);
        }
    }
    return 0;
}
```

Demo 1 – Play music - Achtergrond

De werking van de **buzzer** is gebaseerd op het **piëzo-elektrisch effect**:

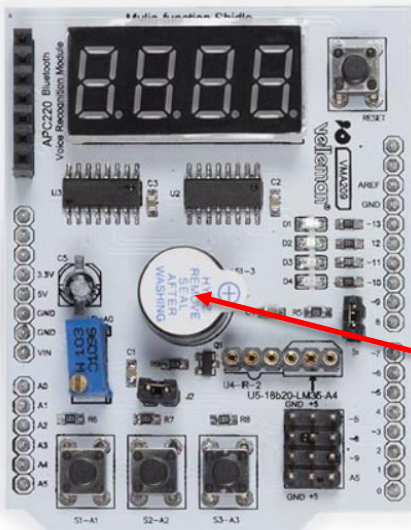


een keramisch materiaal kan onder invloed van een elektrische spanning vervormen.

Een buzzer is een **piëzo-elektrische luidspreker** dat een plaatje van keramisch materiaal bevat. Door afwisselend wel of geen spanning erop te zetten, zal dit materiaal gaan trillen.



Demo 1 – Play music - Achtergrond

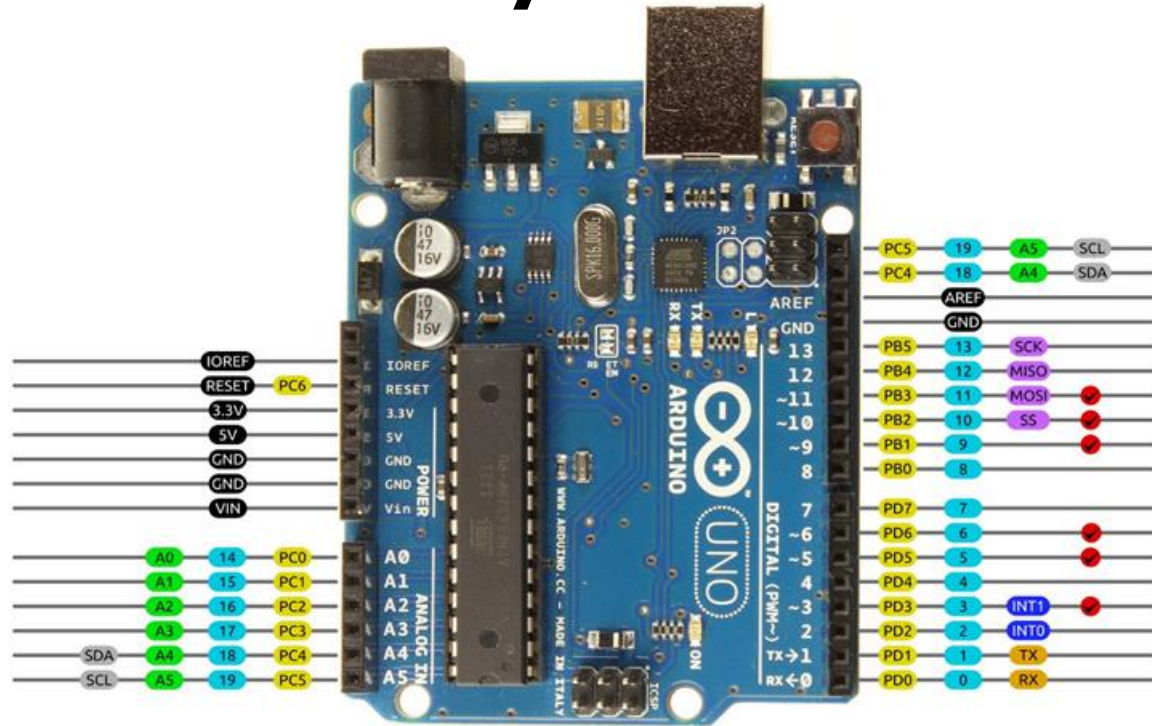


4 red LEDs
 3 buttons + reset button
 potentiometer (10 kΩ)
 4-digit, 7-segment LED tube driven by 74HC595
 buzzer
 socket for IR receiver (remote control)
 socket for temperature sensor LM35 or DS18B20 (polarity!)
 header for APC220 shield
 free pins (PWM)

10, 11, 12, 13
 A1, A2, A3
 A0
 latch 4, clock 7, data 8
 3 (digital on-off)
 2
 A4
 GND, +5V, 0, 1 (RX/TX)
 5, 6, 9, A5

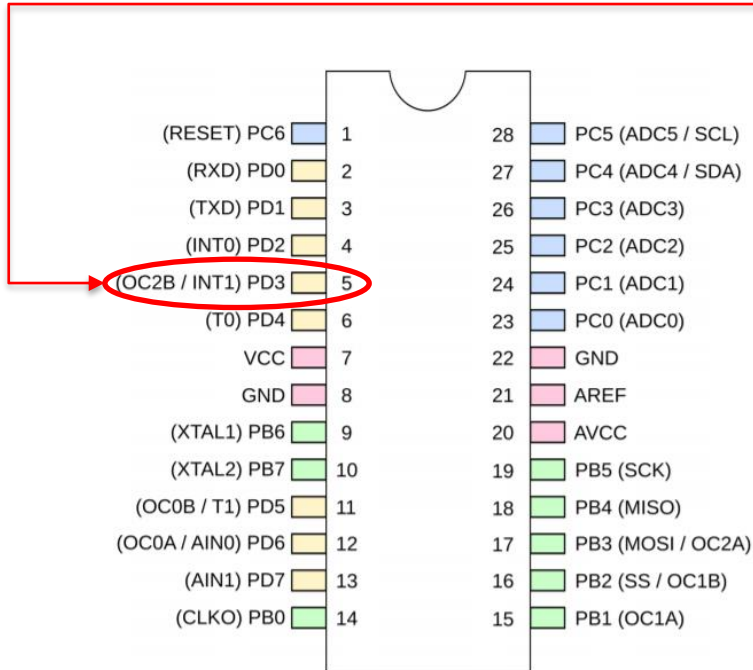


Demo 1 – Play music - Achtergrond



AVR DIGITAL ANALOG POWER SERIAL SPI I2C PWM INTERRUPT

Demo 1 – Play music – Achtergrond

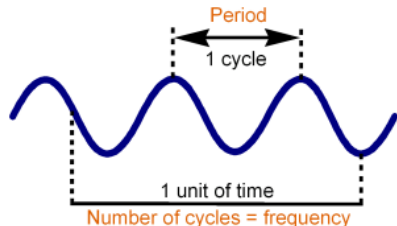


Multi-Function shield V2	Digital pin (Dn)	Analog pin (An)	AVR pin	AVR port	AVR function(s)	AVR PWM
Bluetooth header - tx	0		2	PD0	RxD	
Bluetooth header - rx	1		3	PD1	TxD	
IR receiver	2		4	PD2	INT0	
Buzzer	3		5	PD3	INT1, OC2B	Yes
7-seg display Pin latch	4		6	PD4	T0, XCK	
header	5		11	PD5	T1	Yes
header	6		12	PD6	AIN0	Yes
7-seg display clock	7		13	PD7	AIN1	
7-seg display data	8		14	PB0	CLK0, ICP1	
	9		15	PB1	OC1A	Yes
Red LED	10		16	PB2	OC1B, SS	Yes
Red LED	11		17	PB3	OC2A, MOSI	Yes
Red LED	12		18	PB4	MISO	
Red LED	13		19	PB5	SCK	
Variable resistor	14	0	23	PC0		
Switch 1	15	1	24	PC1		
Switch 2	16	2	25	PC2		
Switch 3	17	3	26	PC3		
DS18B20 header	18	4	27	PC4	SDA	
header	19	5	28	PC5	SCL	

Demo 1 – Play music - Achtergrond

Wat is muziek?

- Een **muzieknoot** is een **druk golf** die zich door de lucht verplaatst: periodes van hogere en lagere dichtheid wisselen zich af.
- **Drukwijzigingen** ervaart ons oor als **geluid**.
- Het aantal keer dat de golf per seconde op en neer gaat, bepaalt de hoogte van de noot. Dit is de **frequentie** (f) van de noot en wordt uitgedrukt in *herz* (= aantal keer op en neer per seconde)
Vb De frequentie van de middelste la op een piano bedraagt 440 herz (de geluidsgolf gaat 440 keer per seconde op en neer)
- De **periode** (T) van een golf is de lengte (in tijd) van 1 volledige op en neer beweging.
Vb De periode van de middelste la op een piano is 1/440e van een seconde

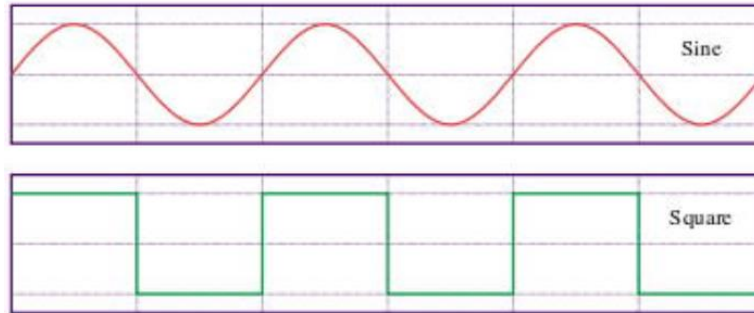


$$f = \frac{1}{T}$$

Demo 1 – Play music - Achtergrond

Muziek met de buzzer

- Willen we een bepaalde muziknoot afspelen op de buzzer, dan moeten we proberen de juiste geluidsgolf te genereren.
- We zullen dat doen door de geluidsgolf te benaderen met een **squarewave**.



- Buzzer afwisselend een tijdje aan en dan weer uit \Rightarrow squarewave!
 - ☐ Periode van een golf = duur van een volledige op en neer beweging.
 - ☐ Dus: golf benaderen met een squarewave door de buzzer een halve periode aan en dan een halve periode uit.
 - ☐ Voldoende herhalen binnen een seconde (frequentie) geeft een noot

Week 3 - Oefening



Oefening

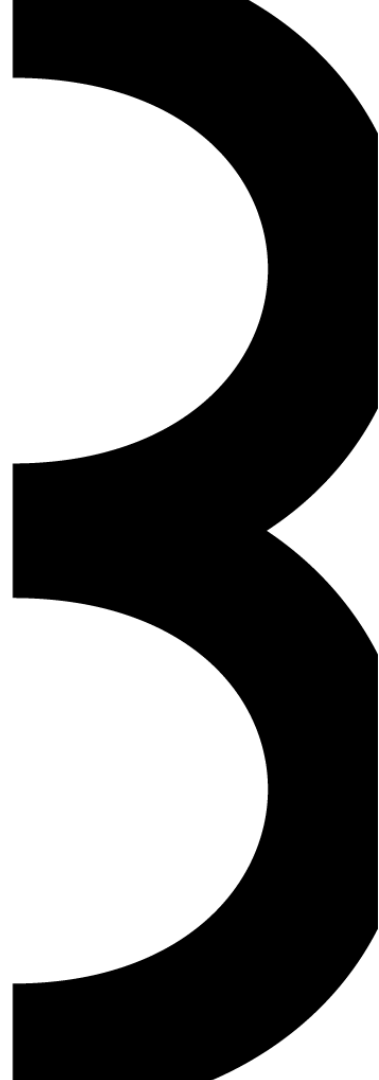
Mijn synthesizer

We kunnen de Arduino gebruiken om een willekeurige toon te produceren, onze eigen kleine synthesizer.

- Je zorgt ervoor dat de gebruiker door aan de potentiometer te draaien een frequentie kan kiezen. De frequentie wordt getoond op de display.
- Zodra je op een knop drukt wordt de toon afgespeeld. Gebruik interrupts!
- Druk je nogmaals op de knop, dan stopt de toon.



Demo 2 – C: Structures



Demo 2 – C: Structures - C-code

```
#include <avr/io.h>
#include <stdlib.h>
#include <usart.h>
#include <util/delay.h>
#include <string.h>
```

```
typedef struct { // Hier definiëren we een eigen datatype: KAART met 2 velden: waarde en kleur
    int waarde;
    char kleur[10];
} KAART;
```

```
void vulBoek(KAART boek[52]) { // krijgt als parameter de array van kaarten mee en vult die met 4 * 13 speelkaarten
    char kleuren[4][10] = {"harten", "ruiten", "schoppen", "klaveren"};
    int index;
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 13; j++) {
            index = (i * 13) + j;
            boek[index].waarde = (j + 1);
            strcpy(boek[index].kleur, kleuren[i]);
        }
    }
}
```

Demo 2 – C: Structures - C-code

```
void toonKaart(KAART deKaart) {
    switch (deKaart.waarde) {
        // de speciale kaarten:
        case 1:
            printf("%s aas", deKaart.kleur);
            break;
        case 11:
            printf("%s boer", deKaart.kleur);
            break;
        case 12:
            printf("%s dame", deKaart.kleur);
            break;
        case 13:
            printf("%s heer", deKaart.kleur);
            break;
        default: // alle andere kaarten:
            printf("%s %d", deKaart.kleur, deKaart.waarde);
    }
}
```

```
KAART trekKaart(KAART boek[]) {
    // De random generator wordt wel NIET geseed,
    // dus bij elke run: zelfde reeks kaarten
    int willek = rand() % 52;
    return boek[willek];
}

int main() {
    initUSART();
    KAART boek[52]; // een array van structures
    vulBoek(boek);
    // Trek 10 willekeurige kaarten en toon ze:
    for (int i = 0; i < 10; i++) {
        KAART kaart = trekKaart(boek);
        toonKaart(kaat);
        printf("\n");
    }
    return 0;
}
```

Demo 2 – C: Structures - Achtergrond

Zie slides beschikbaar in de tutorial ***C: Structures***

Demo 3 – C: Structures dynamisch



Demo 3 – C: Structures dynamisch - C-code

```
#include <avr/io.h>
#include <stdlib.h>
#include <usart.h>
#include <util/delay.h>
#include <string.h>
```

```
typedef struct {
    int waarde;
    char* kleur; //opgelet: pointer, want we willen juist genoeg reserveren
} KAART;
```

```
void vulBoek(KAART* boek) {
    char kleuren[4][10] = {"harten", "ruiten", "schoppen", "klaveren"};
    int index;
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 13; j++) {
            index = (i * 13) + j;
            //TODO: gebruik malloc om juist genoeg ruimte te reserveren voor de kleur
            //TODO: kopieer de kleur naar de gealloceerde ruimte
            //TODO: stel de waarde van de kaart in
        }
    }
}
```

Demo 3 – C: Structures dynamisch - C-code

// OPGELET: er komt een adres van een KAART binnen:

```
void toonKaart(KAART* deKaart) {  
    switch (deKaart->waarde) {  
        //TODO: pas onderstaande code aan:  
        case 1:  
            printf("%s aas", deKaart.kleur);  
            break;  
        case 11:  
            printf("%s boer", deKaart.kleur);  
            break;  
        case 12:  
            printf("%s dame", deKaart.kleur);  
            break;  
        case 13:  
            printf("%s heer", deKaart.kleur);  
            break;  
        default: // alle andere kaarten:  
            printf("%s %d", deKaart.kleur, deKaart.waarde);  
    }  
}
```

// OPGELET: de returnwaarde is een pointer!

```
KAART* trekKaart(KAART boek[]) {  
    int willek = rand() % 52;  
    //TODO: retourneer het adres van de willekeurige kaart  
}
```

```
int main() {  
    initUSART();  
    //TODO: Gebruik calloc om de boek kaarten aan te maken  
    vulBoek(boek);
```

// TODO: pas onderstaande code aan zodat ze compileert

```
for (int i = 0; i < 10; i++) {  
    KAART kaart = trekKaart(boek);  
    toonKaart(kaart);  
    printf("\n");  
}
```

//TODO: geef alle dynamisch gereserveerde ruimte weer vrij:

```
    return 0;  
}
```

Demo 3 – C: Structures dynamisch - C-code

Zie slides beschikbaar in de tutorial ***C: Structures***

Week 4 - Oefeningen



Oefening

Wolfgang Arduino Mozart



We schrijven een programmaatje dat een muziekstukje genereert. Druk je op een knop, dan genereert je Arduino een willekeurig muziekje en speelt het 4 keer na elkaar af!

We gebruiken structures om een SONG en een NOTE te definiëren. Vervolgens schrijven we de functies:

- `void playNote(NOTE* note) :` speel een NOTE af op de buzzer. Heeft de NOTE een frequency van nul, dan spelen we een rust.
- `SONG* generateSong(char* name, uint16_t length) :` genereer een SONG bestaande uit een reeks willekeurige noten. Zorg dat alles netjes op de HEAP gealloceerd wordt!
- `void playSong(SONG* song) :` speel de volledige song af.

Volg voor het uitwerken van deze oefening de instructies die je op Canvas kan terug vinden. **Maar voeg de functie die ontbreekt zelf toe!!!**

Demo 4 – Timers

Demo 4 – Timers - C-code

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

```
uint32_t counter = 0;
```

```
void initTimer0() {
    // STAP 1: kies de WAVE FORM en dus de Mode of Operation
    TCCR0A |= _BV(WGM00) | _BV(WGM01); // WGM00 = 1 en WGM01 = 1 --> Fast PWM Mode: TCNT0 telt steeds tot 255

    // STAP 2: stel *altijd* een PRESCALER in. De snelheid van tellen wordt bepaald door de CPU-klok / PRESCALER
    TCCR0B |= _BV(CS02) | _BV(CS00); // CS02 = 1 en CS00 = 1 --> prescalerfactor is nu 1024 (=elke 64µs)

    // STAP 3: enable INTERRUPTs voor twee gevallen: TCNT0 == TOP en TCNT0 == OCR0A
    TIMSK0 |= _BV(TOIE0);           // overflow interrupt enablen
    TIMSK0 |= _BV(OCIE0A);          // OCRA interrupt enablen
    sei();                          // interrupts globaal enablen
}

ISR(TIMER0_COMPA_vect) { // deze ISR runt telkens wanneer TCNT0 gelijk is aan de waarde in het OCRA-register
    counter++;
    PORTB &= ~(_BV(PB2) | _BV(PB3) | _BV(PB4) | _BV(PB5)); // lightsDownAllLeds ();
}
```

Vervolg op volgende slide

Demo 4 – Timers - C-code

```
ISR(TIMER0_OVF_vect) { // deze ISR runt telkens wanneer TCNT0 gelijk is aan de waarde TOP-waarde (255)
    counter++;
    PORTB |= _BV(PB2) | _BV(PB3) | _BV(PB4) | _BV(PB5); // lightsUpAllLeds ();
}

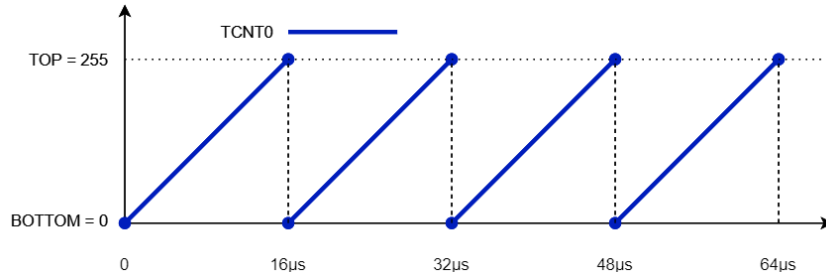
int main() {
    initUSART(); // initialiseer USART
    initTimer0(); // initialiseer Timer 0
    DDRB |= _BV(PB2) | _BV(PB3) | _BV(PB4) | _BV(PB5); // enableAllLeds ();
    while (1) {
        printf("*****\n");
        printf("***** Tik Tak Demo *****\n");
        printf("*****\n");
        printf("Timer 0 is ingesteld om elke 64 µs zijn TCNT0-register te verhogen tot maximaal 255.\n");
        printf("Momenteel is de waarde van het TCNT0-register %d\n\n", TCNT0);
        printf("We gaan een OCR0A-waarde instellen waarmee TCNT0 continu vergeleken zal worden.\n");
        printf("Kijk daarna goed naar je LEDs...\n");
        printf("Geef OCR0A-waarde in m.b.v. 3 cijfers (000-255):");
        OCR0A = getNumber(); // OCR0A kan je steeds vrij kiezen tussen 0 en 255
        printf("Timer 0 genereert nu *achter de schermen* continu twee interrupts:\n");
        printf("\t*) wanneer jouw ORCOA (%d) bereikt wordt.\n", OCR0A);
        printf("\t*) wanneer zijn TOP (%d) bereikt wordt.\n\n", 255);
        printf("Ondertussen zitten we in totaal al aan %d interrupts.\n\n", counter);
    }
    return 0;
}
```


Demo 4 – Timers - Achtergrond

De ATmega328 heeft 3 timers:

Timer	Aantal bits	TOP-waarde	Configuratie-registers	Compare-registers	Interrupt-registers	Verbonden pinnen	Teller
Timer 0	8-bit	255	TCCR0A / TCCR0B	OCR0A / OCR0B	TIMSK0 / TIFR0	5 en 6	TCNT0
Timer 1	16-bit	16635	TCCR1A / TCCR1B	OCR1A / OCR1B	TIMSK1 / TIFR1	9 en 10	TCNT1
Timer 2	8-bit	255	TCCR2A / TCCR2B	OCR2A / OCR2B	TIMSK2 / TIFR2	3 en 11	TCNT2

- Beschouw een timer als een **automatische for-loop** waarbij een teller telt van 0 (BOTTOM-waarde) tot de TOP-waarde (zie tabel) om daarna te herstarten.



ATmega328-klocksnelheid: 16Mhz

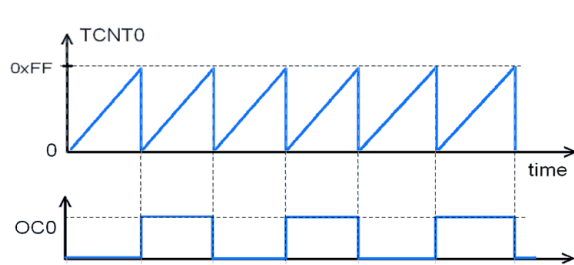
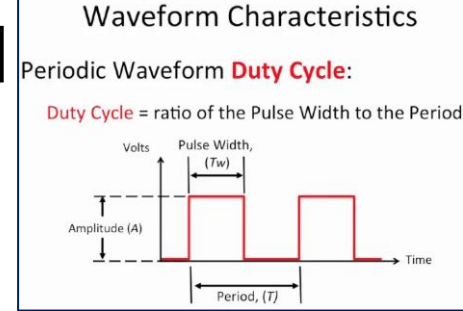
⇒ klokperiode van 62.5ns (= 1/16000000 Hz)

⇒ Timer 0 verhoogt TCNT0 elke 62.5 ns met 1.

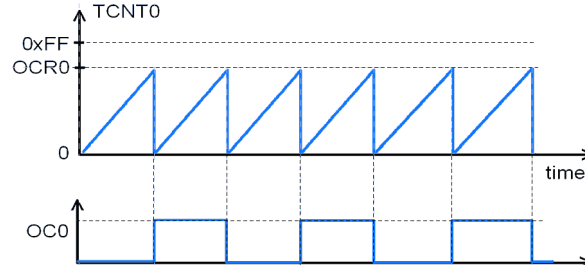
⇒ van 0 t.e.m. 255 in 16 μs (= 256 x 62.5 ns)

Demo 4 – Timers - Achtergrond

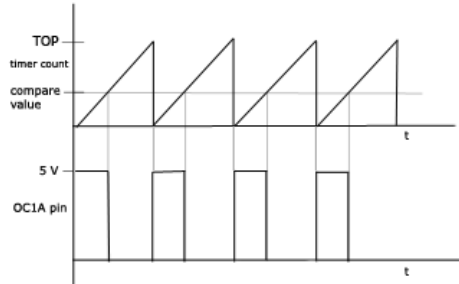
Door op bepaalde momenten een bit HIGH of LOW te zetten, creëren we met een timer een **golfpatroon**. Via de Output Compare Registers (OCRxx) kunnen we het golfpatroon beïnvloeden:



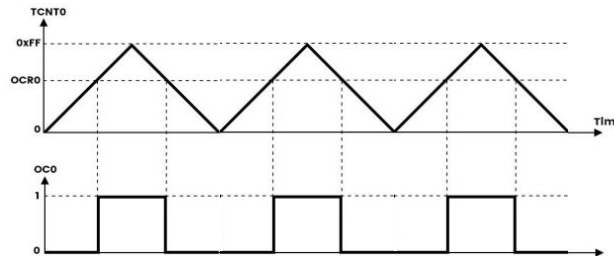
Waveform Generation using **Normal Mode**



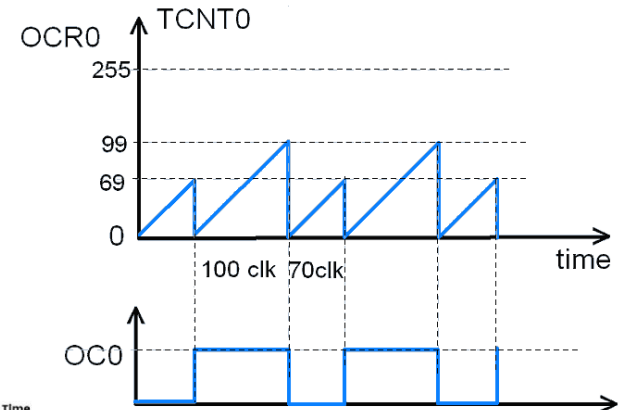
Waveform Generation using **CTC Mode**
(Clear Timer on Compare match)



Waveform Generation using **Fast PWM Mode**
(Pulse Width Modulation)



Waveform Generation using **phase correct PWM mode**



CTC Mode (with OCR0 alternating between 2 values)

Opmerking: Er zijn nog heel wat mogelijkheden!!

Demo 4 – Timers - Achtergrond

Timer 0 heeft heel wat registers.
We bekijken stap voor stap hoe
we Timer 0, kunnen configureren en
gebruiken.

Timer/Counter Control Register A (TCCR0A)

COM0A1	COM0A0	COM0B1	COM0B0	---	---	WGM01	WGM00
7							0

Timer/Counter Control Register B (TCCR0B)

FOC0A	FOC0B	---	---	WGM02	CS02	CS01	CS00
7							0

Timer/Counter Register (TCNT0)

7							0

Output Compare Register A (OCR0A)

7							0

Output Compare Register B (OCR0B)

7							0

Timer/Counter Interrupt Mask Register 0 (TIMSK0)

---	---	---	---	---	OCIE0B	OCIE0A	TOIE0
7							0

Timer/Counter Interrupt Flag Register 0 (TIFR0)

---	---	---	---	---	OCF0B	OCF0A	TOV0
7							0

Demo 4 – Timers - Achtergrond

Configureren van een timer (hier Timer 0):

➤ Stap 1: Mode of Operation instellen

Verschillende golfpatronen zijn mogelijk. Keuze heeft impact op o.a. de manier van tellen, het resetten teller,.. van daar de benaming 'Mode of Operation'

Mode	WGM2	WGM1	WGM0	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on ⁽¹⁾⁽²⁾
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Timer/Counter Control Register A (TCCR0A)

COM0A1	COM0A0	COM0B1	COM0B0	---	---	WGM01	WGM00
7							0

Timer/Counter Control Register B (TCCR0B)

FOC0A	FOC0B	---	---	WGM02	CS02	CS01	CS00
7							0

```
TCCR0A |= _BV(WGM00) | _BV(WGM01);
```

Opmerking: we maken gebruik in ons programma van het feit dat de default setting voor WGM02 = 0 is!

Demo 4 – Timers - Achtergrond

➤ Stap 2: Telsnelheid instellen

De snelheid waarmee de timers tellen (waarmee ze dus de for-loop doorlopen) hangt af van de snelheid van de **CPU-klok**. We kunnen de snelheid verlagen m.b.v. een schalingsfactor, een **prescaler** (cfr. delingsfactor bij de AnalooG Digitaal Conversie).

CS02	CS01	CS00	Schalingsfactor
0	0	0	0 -> timer zal niet tellen (!)
0	0	1	1
0	1	0	8
0	1	1	64
1	0	0	256
1	0	1	1024
1	1	0	Externe bron
1	1	1	Externe bron

ATmega328-kloksnelheid: 16Mhz

⇒ Prescaler = 1024

⇒ Kloksnelheid van 15,625 kHz (= 16 MHz/1024)

⇒ klokperiode van 64 μs (= 1/15625 Hz)

⇒ Timer 0 verhoogt TCNT0 elke 64 μs met 1

⇒ van 0 t.e.m. 255 in 16,384 ms (= 256 x 64 μs)

Timer/Counter Control Register B (TCCR0B)

FOC0A	FOC0B	---	---	WGM02	CS02	CS01	CS00
7							0

$TCCR0B = _BV(CS02) | _BV(CS00);$

Opmerking: we maken gebruik in ons programma van het feit dat de default setting voor CS01 = 0 is!

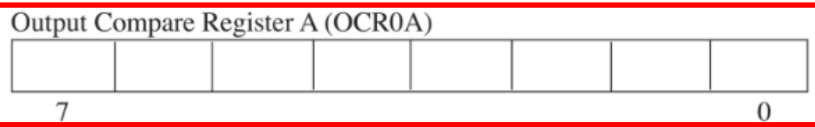
Demo 4 – Timers - Achtergrond

➤ Stap 3: Interrupts instellen

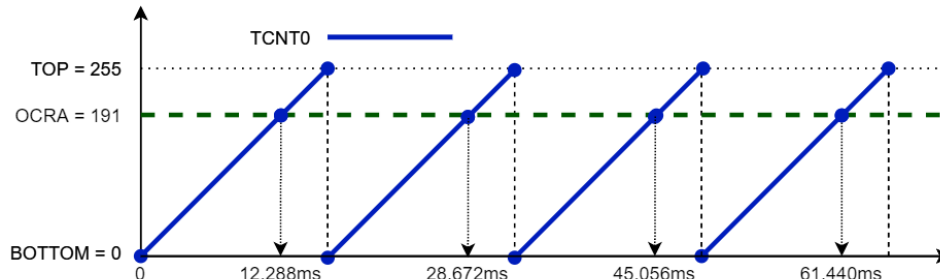
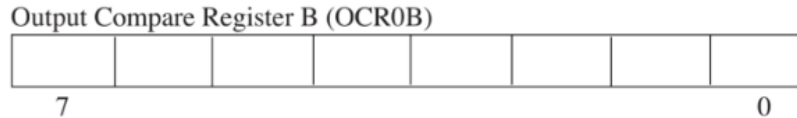
Elke Timer heeft twee Output Compare Registers (OCRxA en OCRxB).

Voor Timer 0 kan je in OCR0A een waarde tussen 0 en 255 stockeren.

Timer 0 vergelijkt continu zijn TCNT0-register met de waarde in dit OCR0A-register.



`OCR0A = getNumber();`

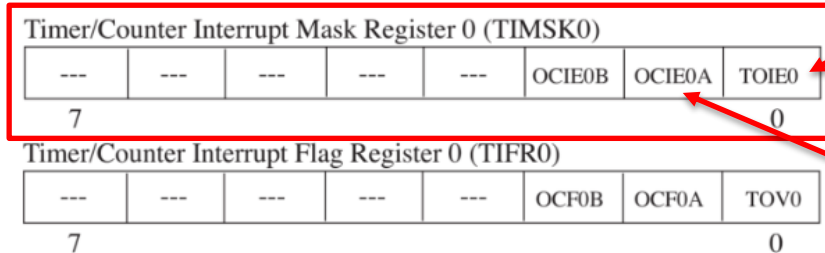


Op het moment dat TCNT0 gelijk wordt aan OCR0A en op het moment dat de TCNT0 gelijk wordt aan de TOP-waarde kunnen we eigen code uitvoeren door gebruik te maken van **Timer interrupts**.

Demo 4 – Timers - Achtergrond

➤ Stap 3: Interrupts instellen

Om op de momenten dat TCNT0 gelijk wordt aan OCR0A en aan de TOP-waarde een interrupt te gegenereerd, moeten we het **Timer Interrupt Mask Register (TIMSK)** configureren.



Overflow interrupt (TCNT0 = TOP-waarde) enablen:

`TIMSK0 |= _BV(TOIE0);`

OCRA interrupt (TCNT0 = OCR0A) enablen:

`TIMSK0 |= _BV(OCIE0A);`

```
ISR(TIMERO_OVF_vect) { // deze ISR runt telkens wanneer TCNT0 gelijk is aan de waarde TOP-waarde (255)
    //code
}
ISR(TIMERO_COMPA_vect) { // deze ISR runt telkens wanneer TCNT0 gelijk is aan de waarde in het OCRA-register
    //code
}
```

Week 4 - Oefeningen



Oefening

Stopwatch maken

We willen een stopwatch bouwen die begint te tellen vanaf er op S1 gedrukt wordt totdat er op S2 gedrukt wordt. S3 reset de stopwatch. De minuten worden getoond op de linker twee display segmenten en de seconden op de rechter twee display segmenten.

We gaan Timer 0 gebruiken om exact 1 seconde (1000 ms) te genereren.

We gebruiken een Timer en niet `_delay_ms` omdat een Timer deze telling parallel kan uitvoeren met de main-code.

Hoe pak je dit aan? Zie instructies op Canvas:

DEEL 1 - Timer configuratie

DEEL 2 - Stopwatch maken

DEEL 3 - Displayen

Weekproject



Week 4 - Weekproject



Opdracht



Overzicht

Lunar Lander 1.0: Op de display krijg je de afstand tot het oppervlakte te zien (startwaarde 9999 meter). De led-lampjes duiden de stand van de brandstof aan. Eén lampje komt overeen met een brandstoftank dat 25% van de start hoeveelheid aan brandstof (hier 1500 liter) bevat. Wanneer een brandstoftank nog volledig vol is, brand het lampje, wanneer er al een gedeelte gebruikt werd, knippert het lampje. Hoe meer er al uit de tank is, hoe sneller het lampje gaat knipperen.

Elke seconde krijg je een update van de afstand tot het oppervlakte en van de stand van de brandstoftanks en kan je door de middelste button in te drukken brandstof verbranden (een burst) om de snelheid van de '**SpaceX HLS 1.0**' te verminderen. Hoe langer je de knop tijdens deze seconde ingedrukt houdt, hoe meer brandstof je verbrand tijdens de burst. Door het drukken op de knop kan je niet meer dan 50 liter gebruiken (per seconde natuurlijk).

Lees aandachtig de richtlijnen en tips op Canvas







Black
board



Cursus/
Boek



Definitie



Discussie



Evaluatie



Inleiding



Leer
doelen



Let op



Lezen



Luisteren



Oefening



Online



Opdracht



Overzicht



Presentatie



Reflectie



Schrijven



Spreeken



Studeer
aanwijzing



Tip



Toets



Tussen
doortje



Verdieping



Verwerking



Video



Voorbeeld



Voorkennis



Zelfstudie