

Embedded C

Week 3: Analog to Digital - LED Display
- C: Geheugenmanagement

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 3
- Demo 1 – Potentiometer
- Demo 2 – Using the display library
- Demo 3 – C: Geheugenmanagement
- Week project

Week3 - Embedded

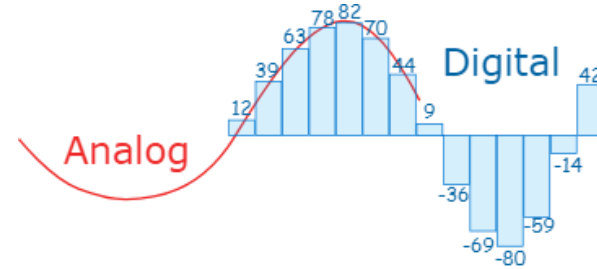
- ↪ Overzicht
- ↪ Tutorials
- ↪ Demos
- ↪ Oefeningen
- ↪ Weekproject

Week 3 - Overzicht



Overzicht

- De potentiometer:
Analoog naar digitaal conversie
- Het LED Display met een shiftregister
- De programmeertaal C:
 - Geheugenmanagement



THE
C
PROGRAMMING
LANGUAGE

Week 3 - Tutorials



Overzicht

- 1 – Interfacing met de Hardware
 - 1.1 – Analooog Digitaal Conversie
 - 1.2 – Het LED display met zijn shiftregister
 - 1.3 – Wat is een potentiometer?

- 2 – De Programmeertaal C
 - 2.1 – Geheugenmanagement

- 3 – Overige Tutorials
 -

Week 3 - Demos



Overzicht

1 – Potentiometer

*Hoe de **A**nalog to **D**igital **C**onvertor die in de AVR zit, gebruiken om een analoog signaal uit te lezen en te converteren naar een getal. We gebruiken de potentiometer als leverancier van het analoge signaal.*

2 – Using the display library

Hoe het gebruik van de display library combineren met de ADC.

3 – C: Geheugenmanagement

We gebruiken allocatiefuncties om geheugen op de heap te alloceren. We tonen hoe je zelf een array van strings kan 'nabouwen' op de heap en opvullen en gebruiken

Week 3 - Oefeningen



Oefening



Overzicht

Scrolling numbers

Schrijf een programma dat de cijfers van rechts naar links laat binnen 'scrollen'. Dus eerst verschijnt '1234' en één seconde later: '2345' en vervolgens: '3456', enz... Opgelet: na '7890' willen we dat '8901' verschijnt en daarna '9012' enz... Dus het programma blijft eindeloos verdergaan.

Alphabet on Display

We gaan de display-library uitbreiden zodat we ook letters kunnen tonen.

Potentiometer on Display

Deze oefening combineert het gebruik van de display library en de ADC.

C: Stack en Heap – Strafwerk

Week 3 - Weekproject

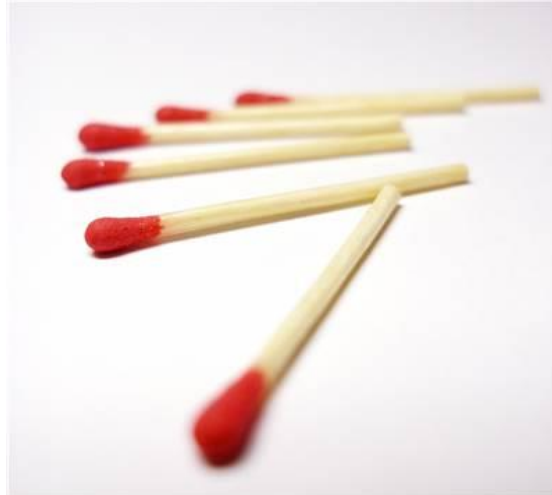


Opdracht










Overzicht

Nim is een spel dat je met twee spelers speelt. Het basisspel verloopt als volgt: er liggen 21 lucifers op een tafel en om beurten nemen de 2 spelers minimaal 1, maximaal 3 lucifers weg. Wie de laatste lucifer neemt, verliest het spel.



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 – Potentiometer

Demo 1 – Potentiometer- C-code

Hoe de Analog To Digital Convertor die in de AVR zit gebruiken om een analoog signaal uit te lezen en te converteren naar een getal. We gebruiken de potentiometer als leverancier van het analoge signaal.

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

void initADC(){
    ADMUX |= (1<<REFS0);           //instellen van de reference voltage. We kiezen voor de 5V als reference voltage
    ADCSRA |= (1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0); //sample rate door een deelfactor (hier 128) in te stellen.
    ADCSRA |= (1<<ADEN);           //Enable de ADC
}

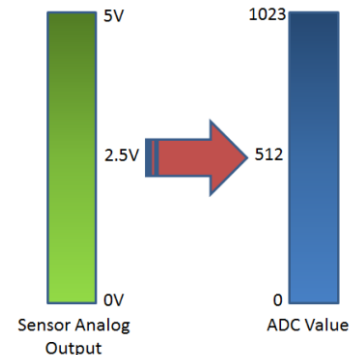
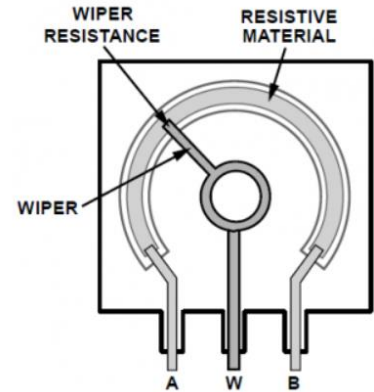
int main(){
    initUSART();
    initADC();
    while (1) {
        ADCSRA |= (1<<ADSC);       //start de conversie analoog -> digitaal
        loop_until_bit_is_clear(ADCSRA,ADSC); //wacht tot conversie gebeurd is
        uint16_t value = ADC;       //lees het resultaat uit
        printf("Value:%d\n", value);
    }
    return 0;
}
```

Demo 1 – Potentiometer - Achtergrond

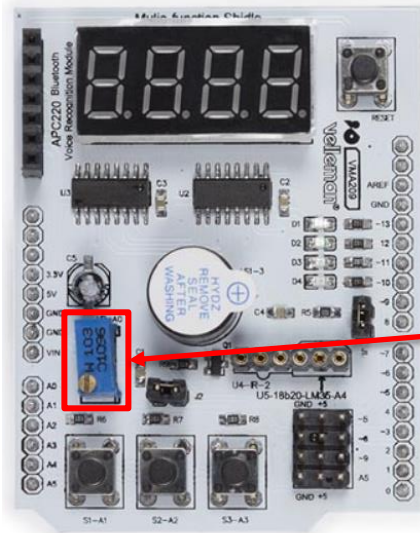
Potentiometers zijn instelbare weerstanden: door de 'wiper' naar links of rechts te draaien, wordt er meer of minder spanning gecreëerd.

De waarde van de **analoge poort** is een waarde tussen de 0 (0v) en 1023 (5v). Dit is afhankelijk van hoeveel spanning er loopt via de potentiometer.

Een potentiometer kan gebruikt worden om bv. een LEDje aan en uit te schakelen met een tijdsinterval of om deze zacht en feller te doen branden.



Demo 1 – Potentiometer - Achtergrond

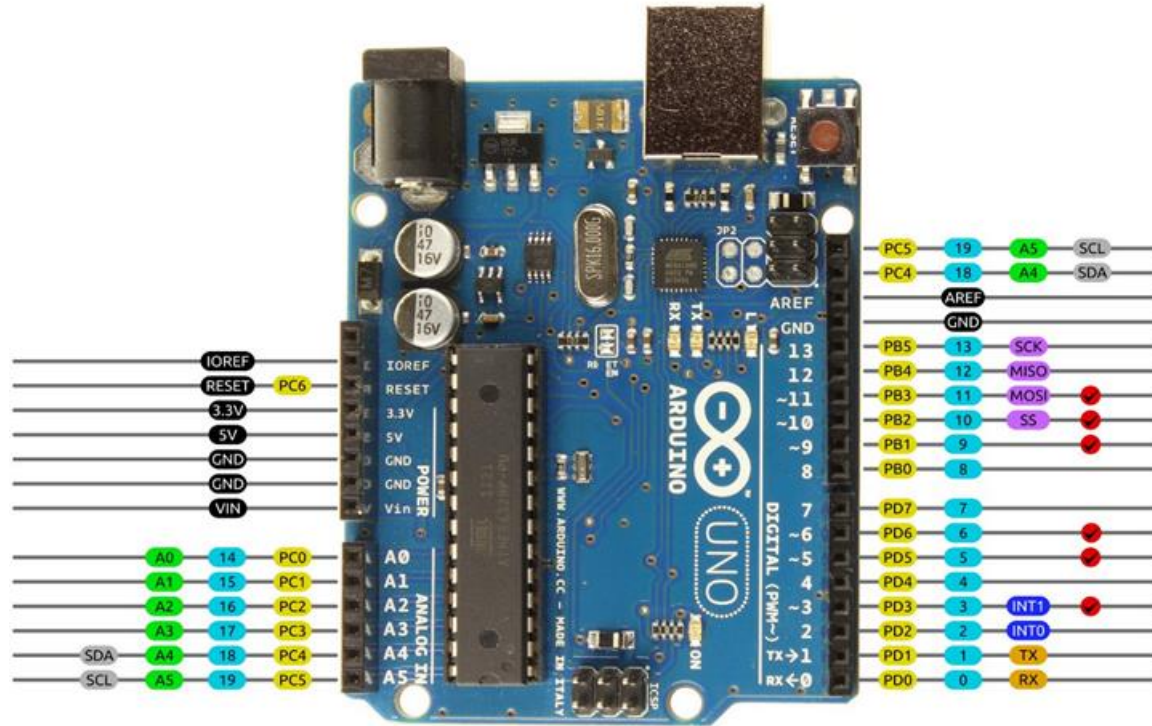


4 red LEDs
3 buttons + reset button
potentiometer (10 kΩ)
4-digit, 7-segment LED tube driven by 74HC595
buzzer
socket for IR receiver (remote control)
socket for temperature sensor LM35 or DS18B20 (polarity!)
header for APC220 shield
free pins (PWM)

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

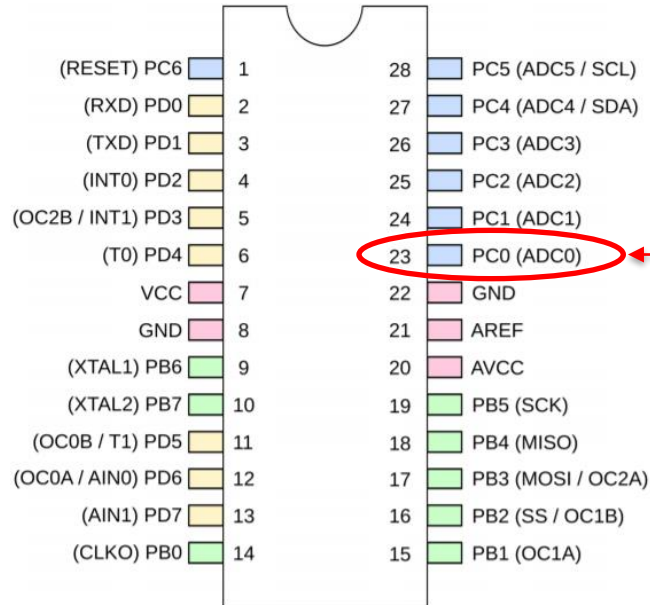


Demo 1 – Potentiometer - Achtergrond



AVR DIGITAL ANALOG POWER SERIAL SPI I2C PWM INTERRUPT

Demo 1 – Potentiometer - Achtergrond



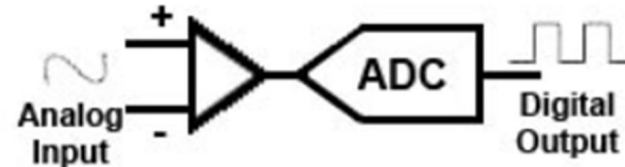
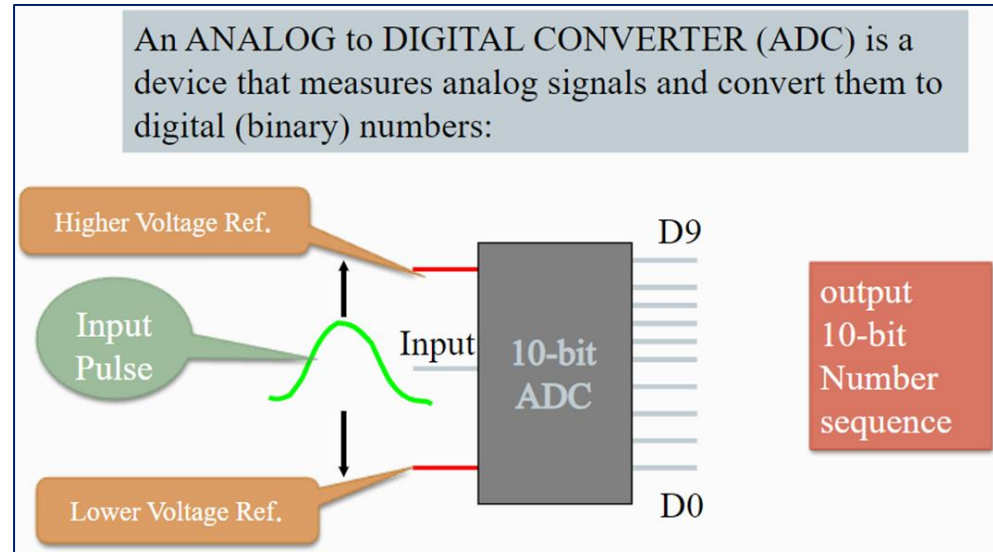
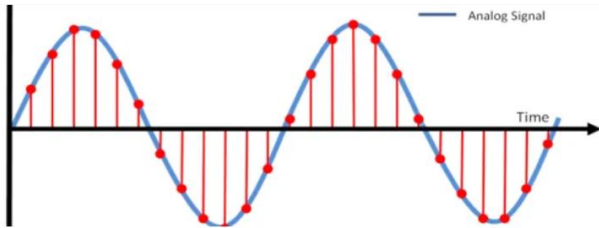
Multi-Function shield V2	Digital pin (Dn)	Analog pin (An)	AVR pin	AVR port	AVR function(s)	AVR PWM
Bluetooth header - tx	0		2	PD0	RxD	
Bluetooth header - rx	1		3	PD1	TxD	
IR receiver	2		4	PD2	INT0	
Buzzer	3		5	PD3	INT1, OC2B	Yes
7-seg display Pin latch	4		6	PD4	T0, XCK	
header	5		11	PD5	T1	Yes
header	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 – Potentiometer - Achtergrond

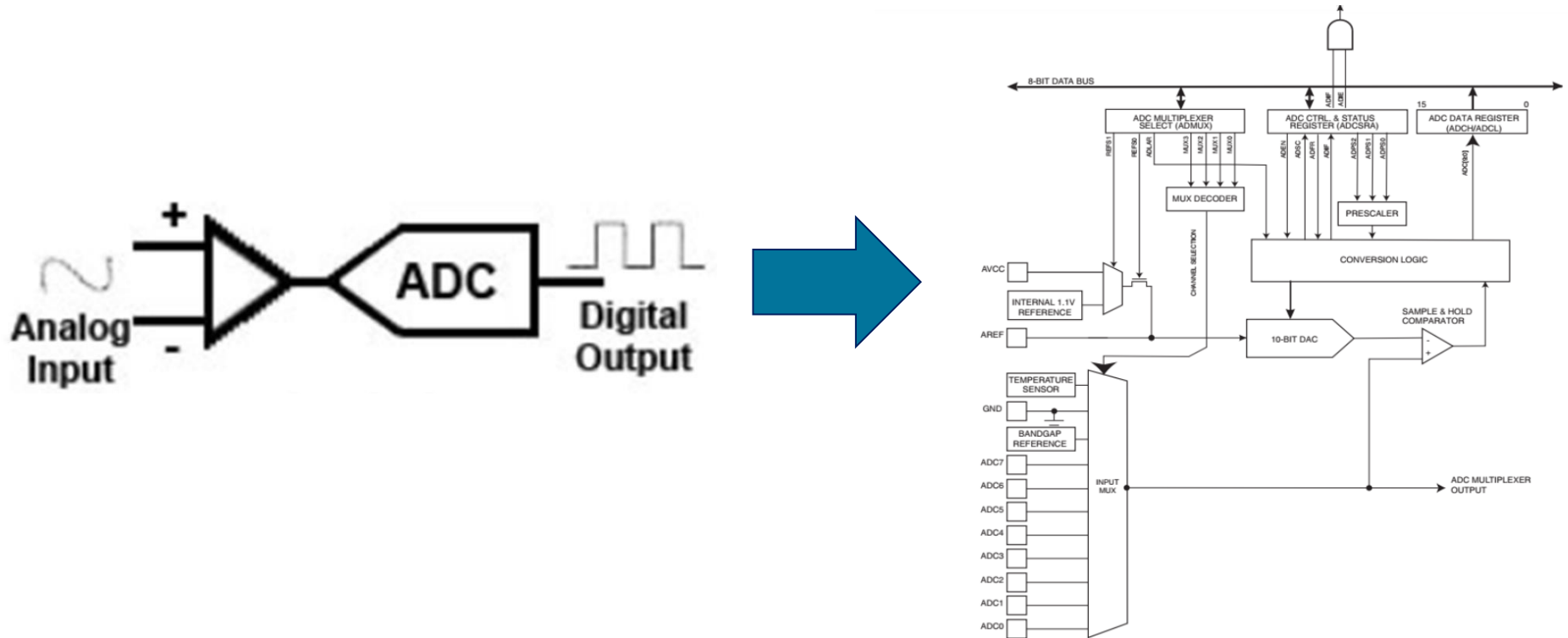
Hoe moeten we nu de waarde opmeten die we hebben ingesteld op onze potentiometer?

Kijken naar 0° bit van PORT C (PC0)?

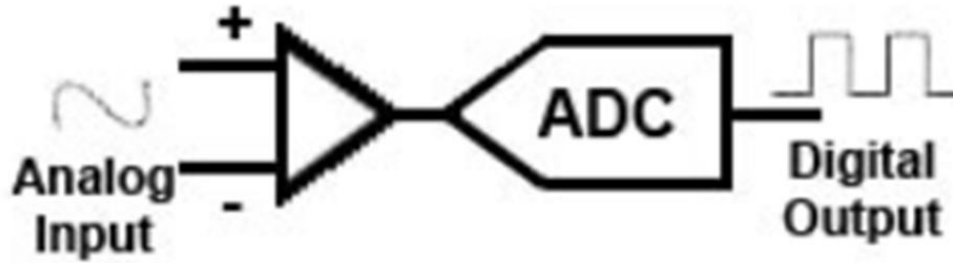
Hoe dan wel?



Demo 1 – Potentiometer - Achtergrond



Demo 1 – Potentiometer - Achtergrond



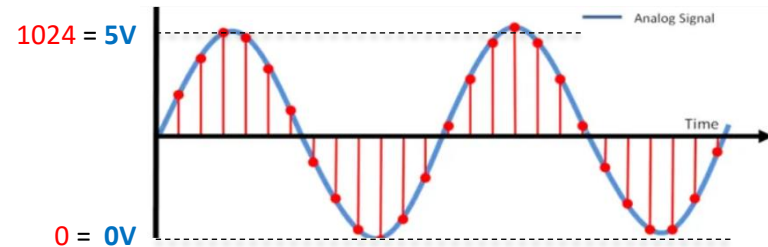
Maximale spanning: $V_{\text{hoog}} = 5 \text{ volt}$

Minimale spanning: $V_{\text{laag}} = 0 \text{ volt}$

Volledige meetspanning (EFSR) = $V_{\text{hoog}} - V_{\text{laag}} = 5\text{V} - 0 \text{ V} = 5\text{V}$

ADC-resolutie (N) = 10 bits = $2^{10} = 1024 \text{ niveaus}$

Daarom is ADC-spanningsresolutie (Q) = $\text{EFSR} / N = 5/1024 = 0,00488 \text{ V}$ of 4,88 mV.

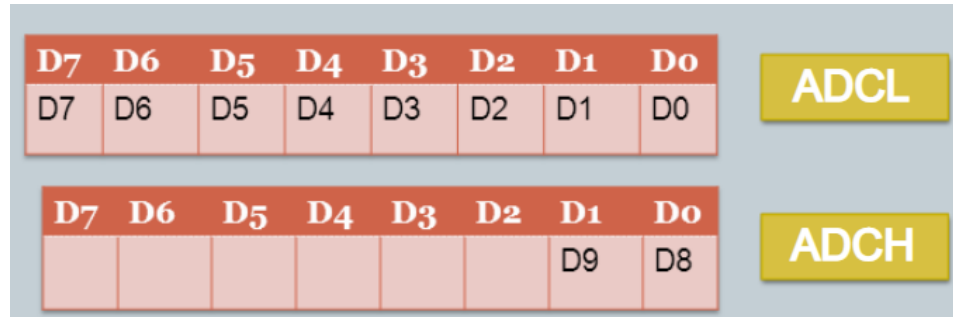


Demo 1 – Potentiometer - Achtergrond

De 10-bit ADC ontvangt input en kan gecontroleerd worden door 4 registers:

- ADMUX: **ADC Multiplex** Register
 - ADCSRA: **ADC Control/Status** Register
 - ADCL: Low byte of converted value
 - ADCH: High byte of converted value
- } 10 bits

`uint16_t value = ADC;`



Demo 1 – Potentiometer - Achtergrond

⇒ ADMUX: ADC Multiplex Register

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Reference voltage :

00 – externe voltage referentie via AREF-pin

01 – AVCC-pin (maw zelfde als power supply nl 5 V) `ADMUX |= (1<<REFS0);`

11 – interne voltage referentie

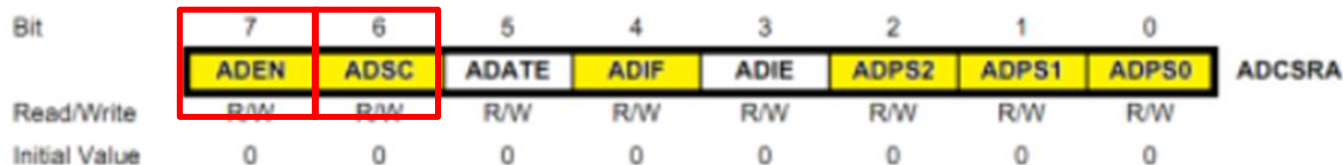
worden gebruikt om de pin
op PORTA op te pikken:
0000 voor PC0,..., 0101 voor PC5

Opmerking: in onze code maken we gebruik van de initiële waarden van het ADMUX-register!!

Demo 1 – Potentiometer - Achtergrond

- ⇒ Ook het activeren (enabelen) van de ADC vereist een wijziging in het ADCSRA register, nl. bit 7 (**ADEN** – **ADC EN**able) op 1 zetten: `ADCSRA |= (1<<ADEN);`
- ⇒ Om de analoog naar digitaal conversie te starten dient bit 6 (ADSC) op 1 gezet te worden: `ADCSRA |= (1<<ADSC);`
- ⇒ Om de volgende conversie/meting te hebben dient er gewacht te worden tot deze nieuwe conversie/meting plaats vond (sample rate): `loop_until_bit_is_clear(ADCSRA,ADSC);`

Opmerking: Na een analoog naar digitaal convertie komt bit 6 (ADSC) terug op nul te staan. Dus terug op 1 zetten om een nieuwe meeting/convertie te verkrijgen.



Bit	7	6	5	4	3	2	1	0
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

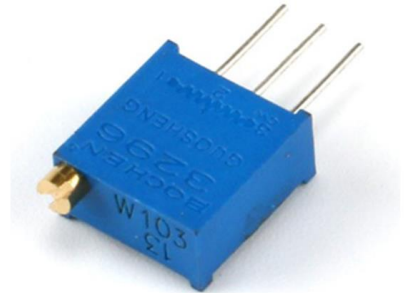
Demo 1 – Potentiometer

TIP:

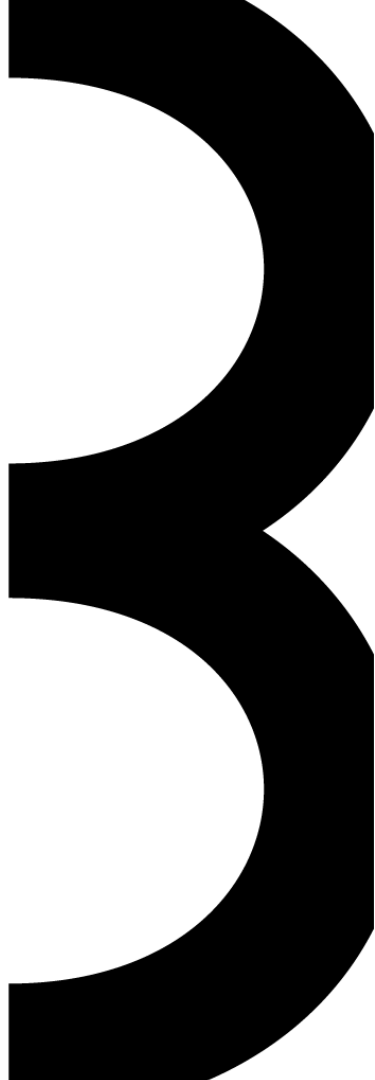
Maak een (kleine) potentiometer library

```
void startPotentiometer();
```

```
uint16_t readPotentiometer();
```



Demo 2 – Using the display library



Demo 2 – Using the display library- C-code

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

int main(void) {
    initUSART();
    initDisplay();
    while (1) {
        for (int i = 0; i < 4; i++) {
            writeNumberToSegment(i, 8);    // functie om een cijfer op één van de LEDdisplays te tonen
            _delay_ms(1000);
        }
        writeNumber(1974);                 // functie om een getal van 4 cijfers tonen
        _delay_ms(1000);
        for (int i = 0; i < 10000; i++) {   // multiplexing
            writeNumber(1974);
        }
        writeNumberAndWait(1974, 1000);    // functie om een getal van 4 cijfers tonen met multiplexing en duurtijd
    }
}
```

Demo 2 – using the display library- Achtergrond

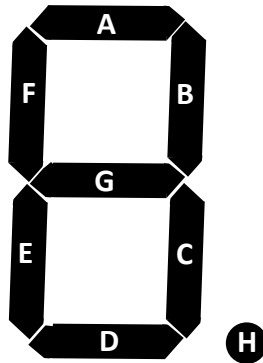
1 byte = 8 bits

= 0bxxxxxxxx met x = 0 (= LED aan) of 1 = (LED uit)

↓↓↓↓↓↓↓↓

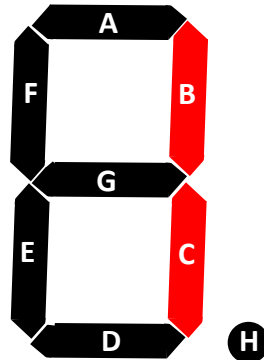
HGFEDCBA

0b11111111



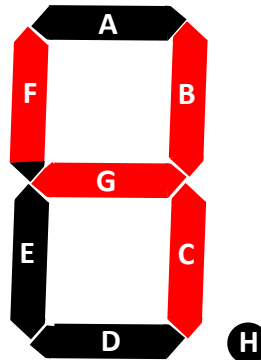
H

'1'
0xF9
0b11111001



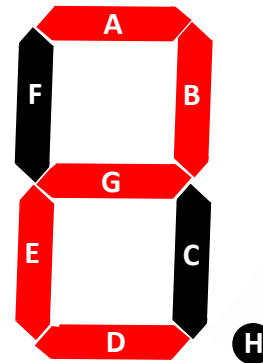
H

'4'
0x99
0b10011001



H

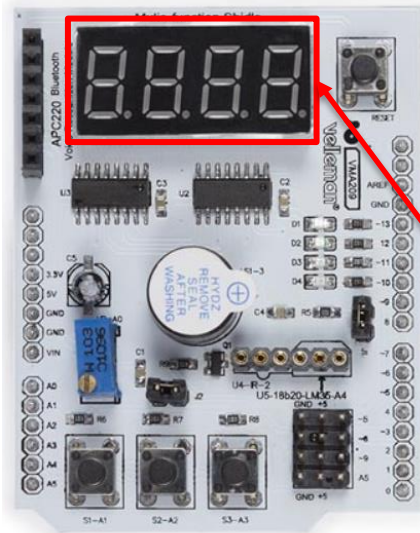
'2'
0xA4
0b10100100



H

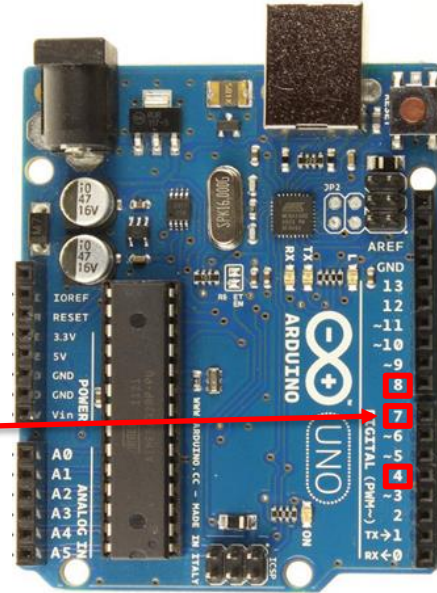


Demo 2 – using the display library- Achtergrond

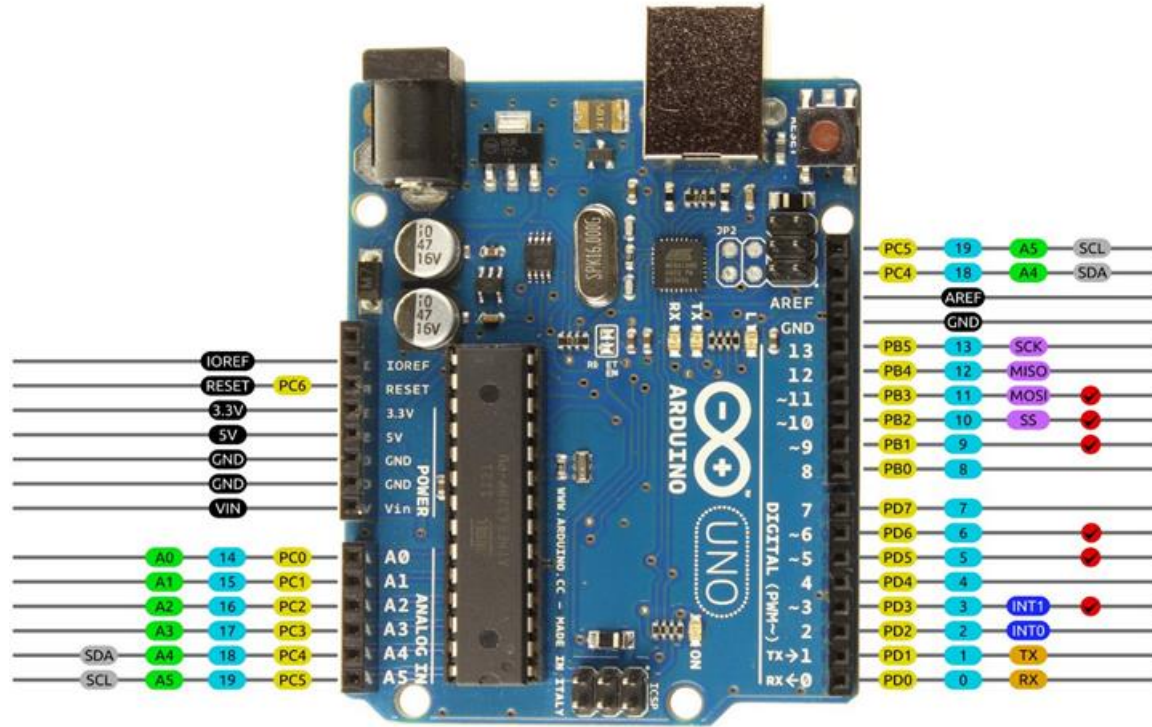


4 red LEDs
3 buttons + reset button
potentiometer (10 kΩ)
4-digit, 7-segment LED tube driven by 74HC595
buzzer
socket for IR receiver (remote control)
socket for temperature sensor LM35 or DS18B20 (polarity!)
header for APC220 shield
free pins (PWM)

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

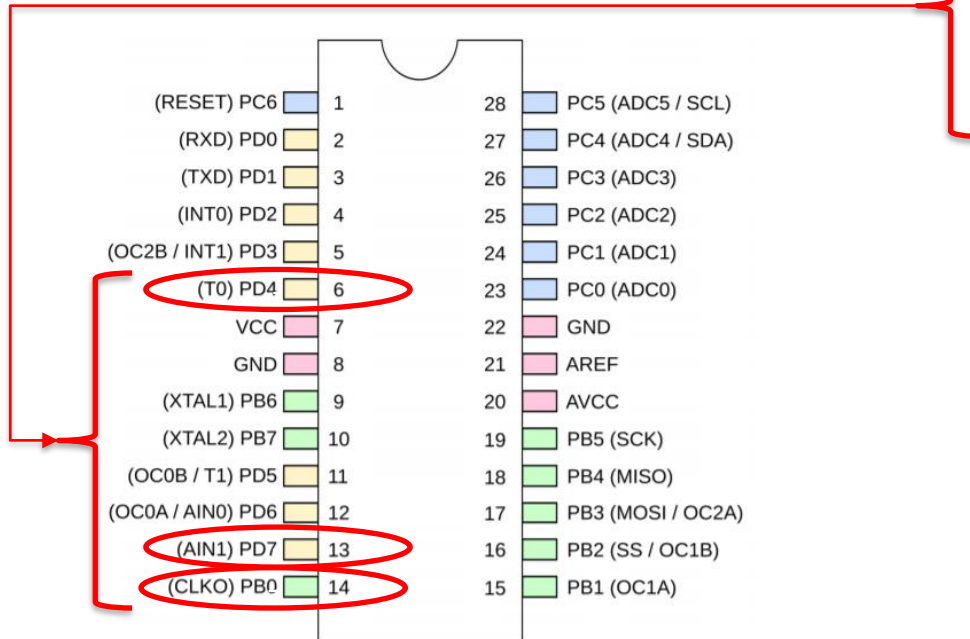


Demo 2 – using the display library- Achtergrond



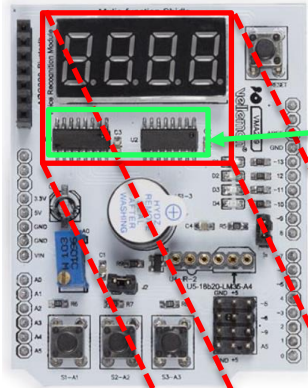
AVR DIGITAL ANALOG POWER SERIAL SPI I2C PWM INTERRUPT

Demo 2 – using the display library- Achtergrond

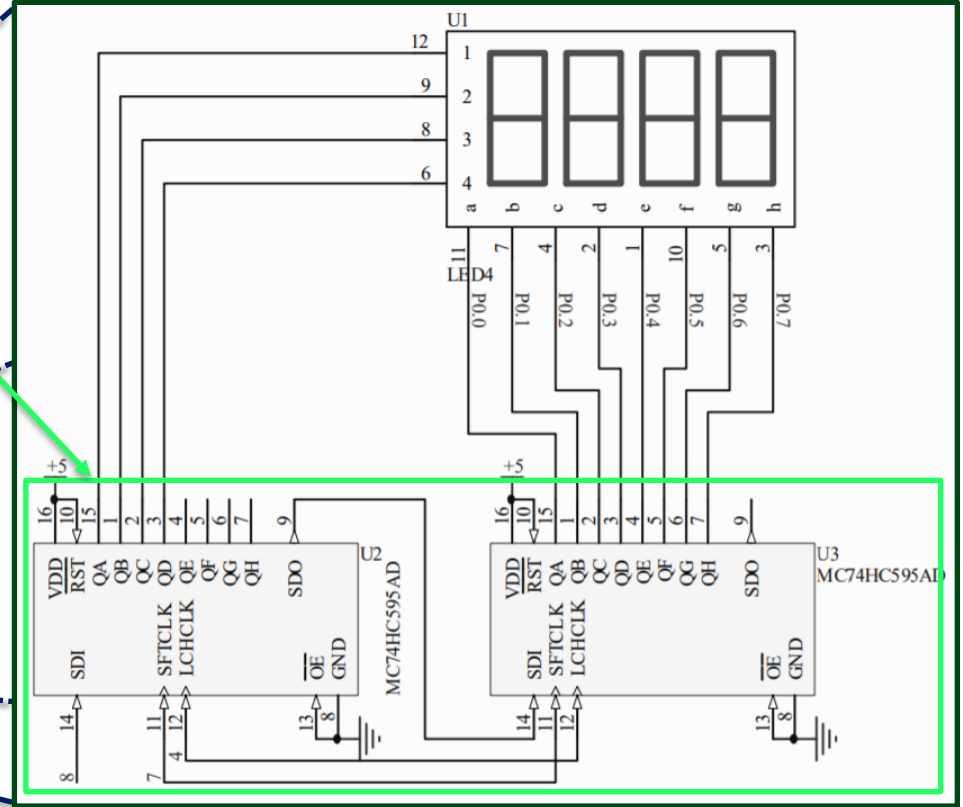
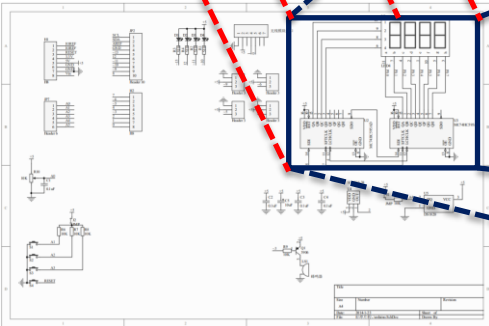


Multi-Function shield V2	Digital pin (Dn)	Analog pin (An)	AVR pin	AVR port	AVR function(s)	AVR PWM
Bluetooth header - tx	0		2	PD0	RxD	
Bluetooth header - rx	1		3	PD1	TxD	
IR receiver	2		4	PD2	INT0	
Buzzer	3		5	PD3	INT1, OC2B	Yes
7-seg display Pin latch	4		6	PD4	T0, XCK	
header	5		11	PD5	T1	Yes
header	6		12	PD6	AIN0	Yes
7-seg display clock	7		13	PD7	AIN1	
7-seg display data	8		14	PB0	CLKO, 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 2 – using the display library- Achtergrond



Shiftregister



Demo 2 – using the display library- Achtergrond

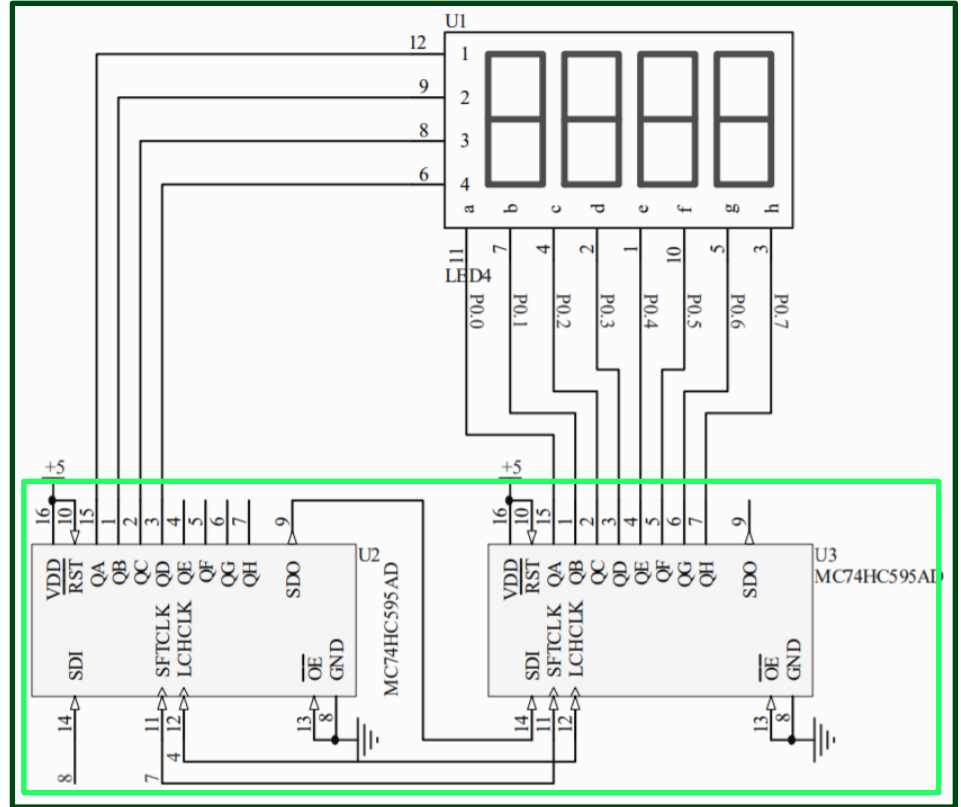
Het **shiftregister** op het extention shield horende bij de 4 digit LED display omvat 2 bytes:

- 1° byte: welke van de (8) LEDs moeten oplichten
- 2° byte: welk van de 4 displays wordt er gebruikt

In een shiftregister "schuift" men via de **serial input pin** de bits van een byte, die de verschillende pins van bv. het display moet aanspreken, binnen. Daarna wordt de byte parallel op de 8 pinnen van het display geduwd.

Probleem: je kan hierdoor maar één van de 4 digit LED display aanspreken.

Oplossing: **multiplexing** – in een lus snel na elkaar de 4 digital LED displays aanspreken



Demo 2 – using the display library- Achtergrond

Hoe de twee bytes in het shiftregister krijgen?

Het 'shift-en' van een byte verloopt bit per bit.

Hierbij wordt gestart vanaf ofwel:

- Most Significant Bit First (MSBF) = uiterst linkse bit eerst
- Least Significant Bit First (LSBF) = uiterst rechtse bit eerst

Via de **latch pin** wordt aangeduid of de chip informatie (bits) dient te ontvangen.

Om beurten wordt een bit uit de byte op de **data pin** geschreven waarna de **clock pin** 'pulseert' (naar high en terug naar low) om aan te duiden dat de bit beschikbaar is.

Bestudeer nu de functies *writeNumberToSegment* en *shift* in de display library.

Demo 2 – using the display library- Achtergrond

```
#define LOW 0
#define HIGH 1
#define LATCH_DIO PD4
#define CLK_DIO PD7
#define DATA_DIO PB0
#define LSBFIRST 0
#define MSBFIRST 1
#define NUMBER_OF_SEGMENTS 8
#define sbi(register, bit) (register |= _BV(bit))
#define cbi(register, bit) (register &= ~_BV(bit))
const uint8_t SEGMENT_MAP[] = {0xC0, 0xF9, 0xA4,
                                0xB0, 0x99, 0x92, 0x82, 0xF8, 0X80, 0X90}
const uint8_t SEGMENT_SELECT[] = {0xF1, 0xF2,
                                   0xF4, 0xF8};
```

```
//Schrijft cijfer naar bepaald segment.
//Segment 0 is meest linkse.
void writeNumberToSegment(uint8_t segment,
                           uint8_t value) {
    cbi(PORTD, LATCH_DIO);
    shift(SEGMENT_MAP[value], MSBFIRST);
    shift(SEGMENT_SELECT[segment], MSBFIRST);
    sbi(PORTD, LATCH_DIO);
}
```

```
// loop through eight segments of LED display and
// shift the correct bits in the data register
void shift(uint8_t val, uint8_t bitorder) {
    uint8_t bit;
    for (uint8_t i = 0; i < NUMBER_OF_SEGMENTS; i++) {
        if (bitorder == LSBFIRST) {
            bit = !(val & (1 << i));
        } else {
            bit = !(val & (1 << (NUMBER_OF_SEGMENTS-1 -i)));
        }
        // write bit to register
        if (bit == HIGH)
            sbi(PORTB, DATA_DIO);
        else
            cbi(PORTB, DATA_DIO);
        // Trigger the clock pin so the display updates
        sbi(PORTD, CLK_DIO);
        cbi(PORTD, CLK_DIO);
    }
}
```

Week 3 - Oefening



Oefening

Display library gebruiken

```
void initDisplay();  
void writeNumberToSegment(uint8_t, uint8_t);  
void writeNumber(int);
```

⇒ Kan deze functie anders/beter?

```
void writeNumberAndWait(int, int);
```

⇒ Kan deze functie anders/beter?

```
void writeCharToSegment(uint8_t, char);  
void writeString(char* str);  
void writeStringAndWait(char*, int);
```

⇒ Zie oefening Alfabet on display



Week 3 - Oefening



Oefening

```
void writeNumberAndWait(int number, int delay) {
    if (number < 0 || number > 9999) return;
    for (int i = 0; i < delay / 16; i++) {
        writeNumberToSegment(0, number/1000);
        _delay_ms();
        writeNumberToSegment(1, (number/100)%10);
        _delay_ms(4);
        writeNumberToSegment(2, (number/10)%10);
        _delay_ms(4);
        writeNumberToSegment(3, number%10);
        _delay_ms(4);
    }
}
```

Wat is het fundamenteel verschil tussen deze twee implementaties van de `writeNumberAndWait`-functie?

```
void writeNumberAndWait(int number, int delay) {
    int maxFactor=1;
    for (int j = 0; j < NUMBER_OF_DISPLAYS; j++) {
        maxFactor *= 10;
    }
    if (number < 0 || number > maxFactor-1) return;
    int segment;
    int factor;
    for (int i = 0; i < delay / 20; i++) {
        factor = maxFactor / 10;
        for (segment=0; segment < NUMBER_OF_DISPLAYS;
              segment++) {
            writeNumberToSegment(segment,
                                  (number / factor) % 10);
            factor = factor / 10;
            _delay_ms(20/NUMBER_OF_DISPLAYS);
        }
    }
}
```

Week 3 - Oefeningen



Oefening

Scrolling numbers

Schrijf een programma dat de cijfers van rechts naar links laat binnen 'scrollen'. Dus eerst verschijnt '1234' en één seconde later: '2345' en vervolgens: '3456', enz...

Opgelet: na '7890' willen we dat '8901' verschijnt en daarna '9012' enz... Dus het programma blijft eindeloos verdergaan.

TIP: natuurlijk gebruik je een for-loop met een teller en de modulo-operator!

Week 3 - Oefeningen



Oefening

Alphabet on Display

We gaan de display-library uitbreiden zodat we ook letters kunnen tonen:

In display.h:

```
void writeCharToSegment(uint8_t segment, char character);  
void writeString(char* str);  
void writeStringAndWait(char* str, int delay);
```

In display.c, de implementaties van bovenstaande functies samen met:

```
const uint8_t ALPHABET_MAP[] = {0x88, 0x83, 0xC6, 0xA1, 0x86, 0x8E, 0xC2, 0x89, 0xCF, 0xE1, 0x8A, 0xC7,  
                                0xEA, 0xC8, 0xC0, 0x8C, 0x4A, 0xCC, 0x92, 0x87, 0xC1, 0xC1, 0xD5, 0x89, 0x91, 0xA4};
```

Week 3 - Oefeningen



Oefening

Potentiometer on Display

Deze oefening combineert het gebruik van de display library en de ADC.

Opgave 1:

Lees de waarde van de potentiometer uit en toont deze op de display. Maak net zoals in de demo gebruik van de 5V als referentiewaarde.

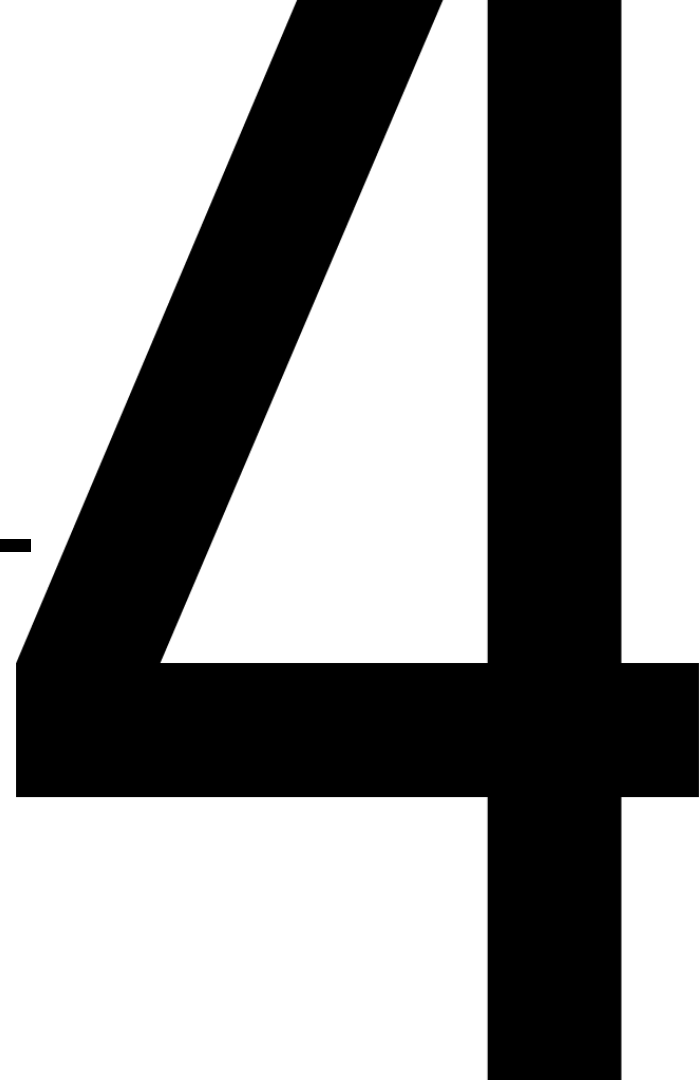
Opgave 2:

Pas de oefening aan: maak nu gebruik van 2.5V als referentiewaarde. Zoek op hoe je dit doet in het artikel onderaan de tutorial rond Analog To Digital Conversie: [1. Analooog Digitaal Conversie](#)

Opgave 3:

Aangezien de knoppen aangesloten zijn op PC1, PC2 en PC3, kan je de input van de knoppen ook door de ADC laten gaan. Probeer dat eens door de ADMUX aan te passen en bestudeer het (vrij zinloze!) resultaat...

Demo 3 – C: Geheugen- management



Demo 3 – C: Geheugenmanagement - C-code

```
#include <stdlib.h>
#include <usart.h>
#include <string.h>
```

```
#define MAX 5
#define AANTAL 7
```

```
void drukNamen(char** namen, int aantal) {
    printString("\nNamen: \n");
    for (int i = 0; i < aantal; i++) {
        printf("de %d° naam is: ", i);
        int j = 0;
        while (namen[i][j] != '\0') {
            printf("%c", namen[i][j]);
            j++;
        }
        printString("\n");
    }
}
```

Demo 3 – C: Geheugenmanagement - C-code

```
int main() {
    initUSART();

    int* tabc;
    tabc = calloc(MAX, sizeof(int));
    printString("\nDe inhoud van de array tabc:");
    for (int i = 0; i < MAX; i++) {
        printf(" %d ", tabc[i]);
    }

    int* tabm;
    tabm = malloc(MAX * sizeof(int));
    printString("\nDe inhoud van de array tabm:");
    for (int i = 0; i < MAX; i++) {
        printf(" %d ", *(tabm + i));
    }
    int* p;
    p = tabm;
    for (int i = 0; i < MAX; i++) {
        *p = i * i;
        p++;
    }

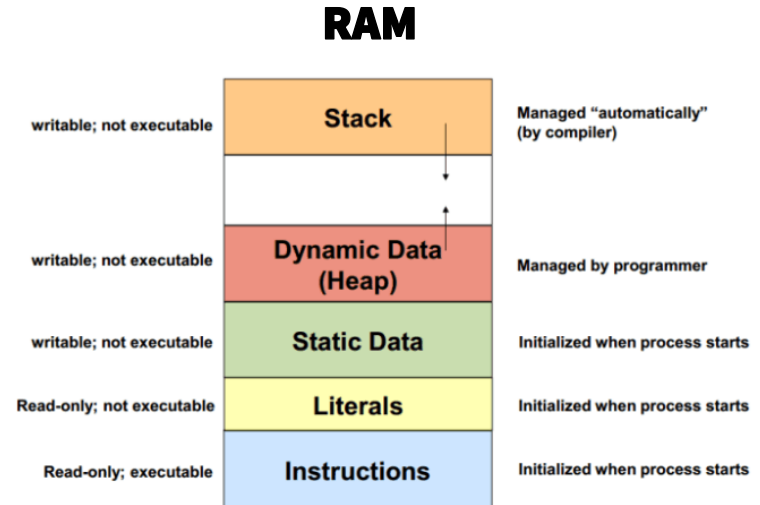
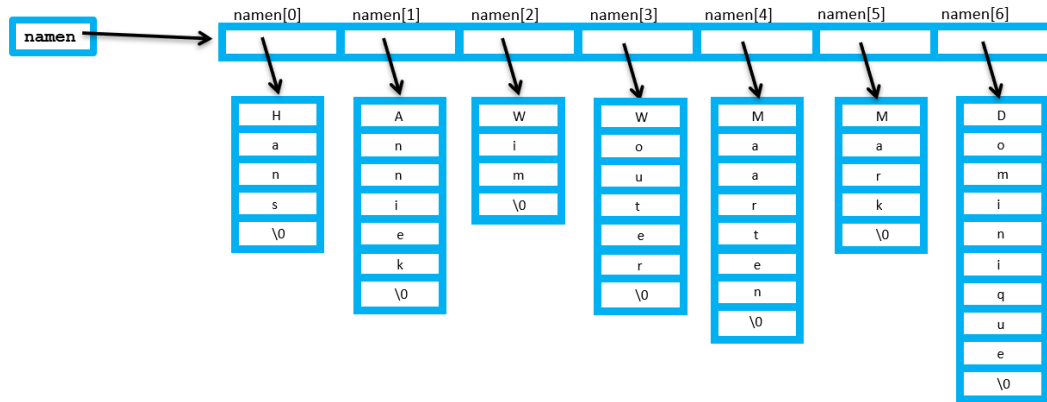
    free(tabc);
    free(tabm);
}
```

```
char namen[AANTAL][10] = {"Hans", "Anniek", "Wim",
                           "Wouter", "Maarten", "Mark", "Dominique"};
char* pnamen[AANTAL];
for (int i = 0; i < AANTAL; i++) {
    pnamen[i] = malloc(strlen(namen[i]) + 1);
    strcpy(pnamen[i], namen[i]);
}
drukNamen(pnamen, AANTAL);
for (int i = 0; i < AANTAL; i++) {
    free(pnamen[i]);
}

char** ppnamen = calloc(AANTAL, sizeof(char*));
for (int i = 0; i < AANTAL; i++) {
    *(ppnamen + i) = malloc(strlen(namen[i]) + 1);
    strcpy(*(ppnamen + i), namen[i]);
}
drukNamen(ppnamen, AANTAL);
for (int i = 0; i < AANTAL; i++) {
    free(*(ppnamen + i));
}
free(ppnamen);
return 0;
}
```

Demo 3 – C: Geheugenmanagement - Achtergrond

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



Week 3 - Oefeningen



Oefening

C: Stack en Heap – Strafwerk



Opgave 1

Schrijf een eenvoudig programmaatje dat 100 keer de zin "Ik mag niet praten in de les" naar de Serial Monitor stuurt, zodra er op de eerste knop van de Arduino gedrukt wordt. Je maakt geen gebruik van interrupts voor deze oefening.

Opdracht 2

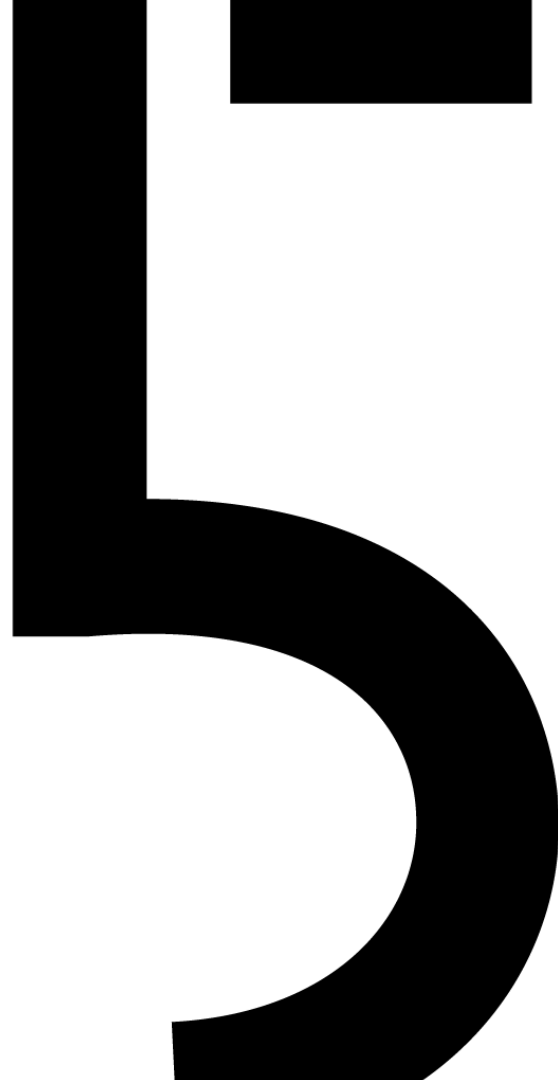
Schrijf een functie `void schrijfStrafOpHeap(char zin[])` die nu de zin op de heap gaat plaatsen:

- Je allocceert juist voldoende plek op de heap om de zin te kunnen bevatten. Je moet hiervoor de lengte van de string vinden, daarvoor kan je stringfuncties uit `string.h` gebruiken. Vergeet de sluitnul niet!
- Je gebruikt een stringfunctie uit `string.h` om de meegegeven string naar de ge-allocceerde plaatst op de heap te kopiëren.

Pas nu je hoofdlus (in de main) aan, zodat bovenstaande functie gebruikt wordt om de string telkens naar de heap te kopiëren. Wat merk je?

Extra: probeer te weten te komen hoeveel plaats er eigenlijk op de heap effectief is. Je kan dit bijvoorbeeld doen door in een globale variabele bij te houden hoeveel plek je juist gereserveerd hebt, en dit elke keer af te printen...

Weekproject



Week 3 - Weekproject



Opdracht



Overzicht

Nim is een spel dat je met twee spelers speelt. Het basisspel verloopt als volgt: er liggen 21 lucifers op een tafel en om beurten nemen de 2 spelers minimaal 1, maximaal 3 lucifers weg. Wie de laatste lucifer neemt, verliest het spel.

In deze implementatie van het spel, speel jij tegen je Arduino. De informatie mbt het verloop van het spel toon je op de 4 digit LED-displays van het extension shield:

- De twee meest rechtse digit LED-displays gebruiken we om het aantal nog beschikbare lucifers (tussen 1 en startAantal) weer te geven.
- De meest linkse digit LED-display gebruiken we om het aantal lucifers weer te geven dat een speler wens weg te nemen (tussen1 en maxAantal).
- De 2de digit LED-display van links, gebruiken we om ofwel de letter 'C' of de letter 'P' te tonen. 'C' geeft aan dat het beurt is van de computer, 'P' duidt aan dat het aan jouw beurt is (P van Player).

Lees aandachtig de spelomschrijving op Canvas:

- Start van het spel
- Verloop spel
- Strategie computer
- Einde spel
- Uitbreiding







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