

Opdrachten week 5 les 1 – Functies en interne klok

In de vorige lessen heb je geleerd om een led aan te sturen en een knop in te lezen met de MSP430G2553. Daarbij heb je gebruik gemaakt van de zogenoemde bitwise operatoren die beschikbaar zijn in de programmeertaal C. Je hebt ook kennis gemaakt met het **if**- en **while**-statement die in de programmeertaal C gebruikt kunnen worden.

In deze les leer je:

- een functie maken in C;
- een eenvoudige manier van tijdsbeheer toepassen;
- om de interne klok in te stellen van de microcontroller;
- wat het probleem is van het denderen van een knop;
- hoe je een knop kunt ontdenderen.

In de volgende les wordt gekeken hoe je functies eenvoudig kunnen hergebruiken door ze op te nemen in een apart bestand. Daarbij ga gebruik maken van git, een versiebeheersysteem.

Lees **paragraaf 1.5 van het boek**¹.

5.1.1 Maak een nieuw project aan en noem dit opdr_5.1.1. Zorg ervoor dat de RGB led is aangesloten op P2.0 (Rood), P2.1 (Groen) en P2.2 (Blauw). Maak daarbij gebruik van weerstanden met de waarden die je in **opdracht 4.2.12** hebt berekend. Kopieer de code uit **listing 1** in het bestand main.c. In de functie main wordt de standaardfunctie `__delay_cycles` aangeroepen die het als argument meegegeven aantal klokcycli wacht. De interne klok van de MSP430G2553 is, na een reset, ongeveer 1,1 MHz dus als je 1100000 klokcycli wacht, is er ongeveer een seconde verstreken. Als het goed is, begrijp je alle overige code die in de functie main gegeven is. Vraag de docent om uitleg als dit niet het geval is! Boven de functie main is de functie `zet_led_aan` gegeven. Deze functie heeft 1 parameter genaamd lednummer van het type **int**. De functie heeft geen return waarde (**void**). Het doel is om led 1 (Rood), 2 (Groen) of 3 (Blauw) aan te zetten met deze functie. Vul nu de daarvoor benodigde code in de functie in. Maak slim gebruik van de `<<` bitwise shift-operatie om led 1, 2 of 3 aan te zetten op basis van de parameter lednummer. Als het goed is gaat de rode led branden een seconde nadat je het programma hebt gestart. Een seconde later gaat ook de groene led aan

¹ Carl Burch. *C for Python programmers*. 2011. URL: <http://www.cburch.com/books/cpy/>

(de rode blijft aan). Nog een seconde later gaat tot slot ook de blauwe led aan, zodat alle leds tegelijkertijd branden.

```
#include <msp430.h>

void zet_led_aan(int lednummer)
{
    // vul hier je code in
}

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD; // stop de watchdog timer
    P2DIR |= 1<<2 | 1<<1 | 1<<0; // zet pins P2.2, P2.1 en P2.0 op ←
    ↪ output
    P2OUT &= ~(1<<2 | 1<<1 | 1<<0); // maak pins P2.2, P2.1 en ←
    ↪ P2.0 laag

    while (1)
    {
        __delay_cycles(1100000); // wacht ongeveer een seconde
        zet_led_aan(1);
        __delay_cycles(1100000); // wacht ongeveer een seconde
        zet_led_aan(2);
        __delay_cycles(1100000); // wacht ongeveer een seconde
        zet_led_aan(3);
    }

    return 0;
}
```

Listing 1: Functie om leds aan te zetten. Zie [opdr5.1.1.c](#)

5.1.2 Vervang de code uit de **while**-loop nu door de code die gegeven is in [listing 2](#). Schrijf nu ook de functie `zet_led_uit()` die de led met het als argument meegegeven led-nummer uitzet (zonder de overige leds te beïnvloeden). Als het goed is branden achtereenvolgens de rode, groene en blauwe één voor één gedurende ongeveer een seconde.

```
__delay_cycles(1100000); // wacht ongeveer een seconde
zet_led_uit(3);
zet_led_aan(1);
__delay_cycles(1100000); // wacht ongeveer een seconde
zet_led_uit(1);
zet_led_aan(2);
__delay_cycles(1100000); // wacht ongeveer een seconde
zet_led_uit(2);
zet_led_aan(3);
```

Listing 2: Functie om leds uit te zetten. Zie [opdr5.1.2.c](#)

5.1.3 Test de `zet_led_uit()` functie nog iets beter door de led met nummer 3 niet uit te zetten in de `while`-loop. De blauwe led moet dan, nadat hij is aangezet, aan blijven.

Net als in Python kunnen functies in de taal C een returnwaarde hebben. Er moet echter wel van tevoren worden aangegeven van welk type deze returnwaarde is. Je kunt zo'n returnwaarde gebruiken om de toestand van een knop terug te geven.

5.1.4 Sluit een drukknop aan op pin P1.0. Kies voor de pull-down configuratie. Stel de juiste registers in om de pin als input te kunnen gebruiken en om de pull-down weerstand in te schakelen in de functie `main` voor de `while`-loop. Maak nog een functie aan boven de `main`-functie. Deze functie moet de waarde 1 teruggeven als de knop is ingedrukt of de waarde 0 als de knop niet is ingedrukt. Noem de functie `knop_is_ingedrukt`. Deze functie heeft geen parameter (`void`). Test de functie `knop_is_ingedrukt` door de `while(1)`-loop te vervangen door de code uit [listing 3](#). Als het goed is gaan de leds één voor één aan zolang de knop wordt ingedrukt. Snap je waarom het programma niet meteen reageert als je de knop loslaat? Snap je waarom de blauwe led blijft branden als je de knop loslaat?

De MSP430G2553 kan gebruik maken van een aantal klokbronnen: externe kristallen en interne oscillatoren. Een oscillator is een schakeling die een kloksignaal kan genereren. Een (piëzo-elektrisch) kristal is een materiaal wat op een vaste frequentie oscilleert wanneer een spanning wordt aangebracht en weggenomen. Interne oscillatoren hebben als voordeel dat het goedkoop is om ze toe te passen, er zijn immers geen externe componenten nodig. Oscillatoren op basis van een extern kristal hebben als voordeel dat ze preciezer en nauwkeuriger zijn dan de volledig interne oscillatoren. In de MSP430G2553 zit ook een

```
while (1)
{
    while (knop_is_ingedrukt())
    {
        __delay_cycles(1100000); // wacht ongeveer een seconde
        zet_led_uit(3);
        zet_led_aan(1);
        __delay_cycles(1100000); // wacht ongeveer een seconde
        zet_led_uit(1);
        zet_led_aan(2);
        __delay_cycles(1100000); // wacht ongeveer een seconde
        zet_led_uit(2);
        zet_led_aan(3);
    }
}
```

Listing 3: Functie om een drukknop in te lezen. Zie [opdr5.1.4.c](#)

digitaal bestuurbare oscillator (DCO = Digitally Controlled Oscillator). De DCO kan worden ingesteld om te oscilleren op verschillende frequenties. Wij gaan gebruik maken van een frequentie van 16 MHz.

5.1.5 Gebruik de [online resource explorer](#) om voorbeeldcode te vinden voor het instellen van de DCO. Ga naar [MSP430 microcontrollers](#) » [Embedded Software](#) » [MSP430Ware](#) » [Devices](#) » [MSP430G2XX](#) en kies voor [MSP430G2553](#) » [Peripheral Examples](#) » [Register Level](#) en klik op `mcp430g2xx3_dco_calib.c`². Kopieer het stuk code om de DCO in te stellen op 16 MHz naar jouw functie main. Als het goed is brandt elke led nog maar heel kort, zolang de drukknop is ingedrukt.

5.1.6 Pas het main-programma nu zo aan dat elke led weer een seconde brandt, zolang de drukknop is ingedrukt, bij een klokfrequentie van 16 MHz.

5.1.7 Schrijf een functie genaamd `zet_klok_op_MHz` die de interne klokfrequentie instelt op de als argument meegegeven waarde (in MHz). De enige geldige waarden voor het argument zijn 1, 8, 12 en 16. De functie geeft de waarde 0 terug als het niet gelukt is om de frequentie aan te passen. De functie geeft de waarde 1 terug als het wel gelukt

² Zie: https://dev.ti.com/tirex/explore/node?node=A__APHjs06D5Cc3ElncZl6MVg__msp430ware__IOGqZri__LATEST.

is. Test de functie `zet_klok_op_MHz` door gebruik te maken van de standaardfunctie `__delay_cycles` en een led.

Het is soms handig als een programma reageert op het aantal maal dat op een bepaalde knop is gedrukt. In [listing 4](#) is een programma gegeven waarbij de led die aangesloten is op P2.0 moet gaan branden nadat driemaal op de drukknop die is aangesloten op P1.0 is gedrukt. Er wordt daarbij vanuit gegaan dat de drukknop een hoog signaal geeft bij het indrukken.

5.1.8 Maak een nieuw project aan genaamd `opdr_5.1.8` en kopieer het programma gegeven in [listing 4](#) in `main.c`. Test of dit programma aan de verwachtingen voldoet door telkens de knop 3 maal in te drukken. Hoogst waarschijnlijk is dit niet het geval!

Lees [hoofdstuk 5 van het handboek](#)³ tot [paragraaf 5.1](#).

5.1.9 Verklaar waarom het programma gegeven in [listing 4](#) niet aan de verwachtingen voldoet.

Een knop bestaat uit een veercontact mechanisme. Als de knop wordt ingedrukt wordt er verbinding gemaakt en als de knop wordt losgelaten zorgt de veer ervoor dat het contact wordt verbroken. Helaas heeft dit wel eens tot effect dat de veer een aantal maal op- en neergaat en hierbij een aantal extra keer contact wordt gemaakt; Dit wordt ook denderen genoemd (Engels: Bouncing). Zie [figuur 1](#). Dit geeft problemen, als je wilt tellen hoe vaak een knop wordt ingedrukt, zoals je waarschijnlijk hebt gezien bij [opdracht 5.1.9](#). De microcontroller werkt namelijk zo snel dat er extra knopdrukken geregistreerd zullen worden.

Er zijn grofweg twee verschillende opties voor het werken met denderende contacten: hardware- of softwarematig oplossen.

Het denderen is in essentie een hardware probleem. Dit kan alsnog in de software worden opgelost door bijvoorbeeld een tijdje te wachten tussen twee verschillende controles. Zoals te zien is in [figuur 1](#) duurde het bij deze knop ongeveer 2 ms. Het signaal kan ook softwarematig ontdeinderd worden door het te integreren. Hier gaan [paragraaf 5.2](#) en [5.3](#) over van het handboek.

³ Daniël Versluis. *Microprocessor Programmeren in C*. 2018. URL: https://bytebucket.org/HR_ELEKTRO/ems10/wiki/Handboek/EMS10_handboek_ebook.pdf

```

#include <msp430.h>

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD; // stop de watchdog timer
    P2DIR |= 1<<2 | 1<<1 | 1<<0; // zet pins P2.2, P2.1 en P2.0 op ←
    ↪ output
    P2OUT &= ~(1<<2 | 1<<1 | 1<<0); // maak pins P2.2, P2.1 en ←
    ↪ P2.0 laag
    P1DIR &= ~(1<<0); // zet pin P1.0 op input
    P1REN |= 1<<0; // zet interne weerstand aan bij pin P1.0
    P1OUT &= ~(1<<0); // selecteer pull down weerstand op pin P1.0.

    while (1)
    {
        while ((P1IN & 1<<0) == 0) /* leeg */; // wacht tot P1.0 ←
    ↪ wordt ingedrukt
        while ((P1IN & 1<<0) != 0) /* leeg */; // wacht tot P1.0 ←
    ↪ wordt losgelaten
        while ((P1IN & 1<<0) == 0) /* leeg */; // wacht tot P1.0 ←
    ↪ wordt ingedrukt
        while ((P1IN & 1<<0) != 0) /* leeg */; // wacht tot P1.0 ←
    ↪ wordt losgelaten
        while ((P1IN & 1<<0) == 0) /* leeg */; // wacht tot P1.0 ←
    ↪ wordt ingedrukt
        P2OUT ^= 1<<0; // inverteer de led
        while ((P1IN & 1<<0) != 0) /* leeg */; // wacht tot P1.0 ←
    ↪ wordt losgelaten
    }

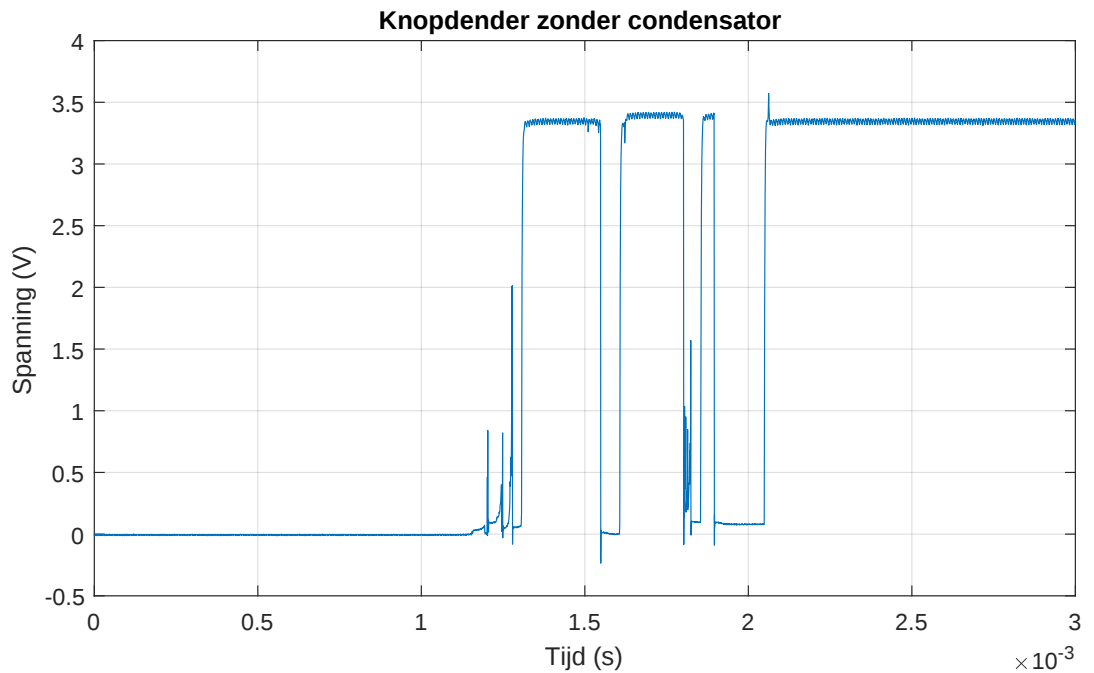
    return 0;
}

```

Listing 4: Na drie keer drukken invertiert de led. Zie [opdr5.1.8.c](#)

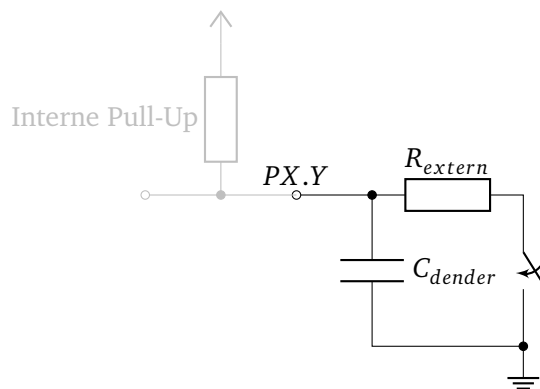
Wij kiezen nu echter voor een hardwarematige oplossing.

5.1.10 Het signaal van [figuur 1](#) beweegt te veel. Het simpele RC netwerk van [schakeling 1](#) zou dit kunnen verhelpen. Een condensator is bijna volledig opgeladen na 5 RC-tijden.



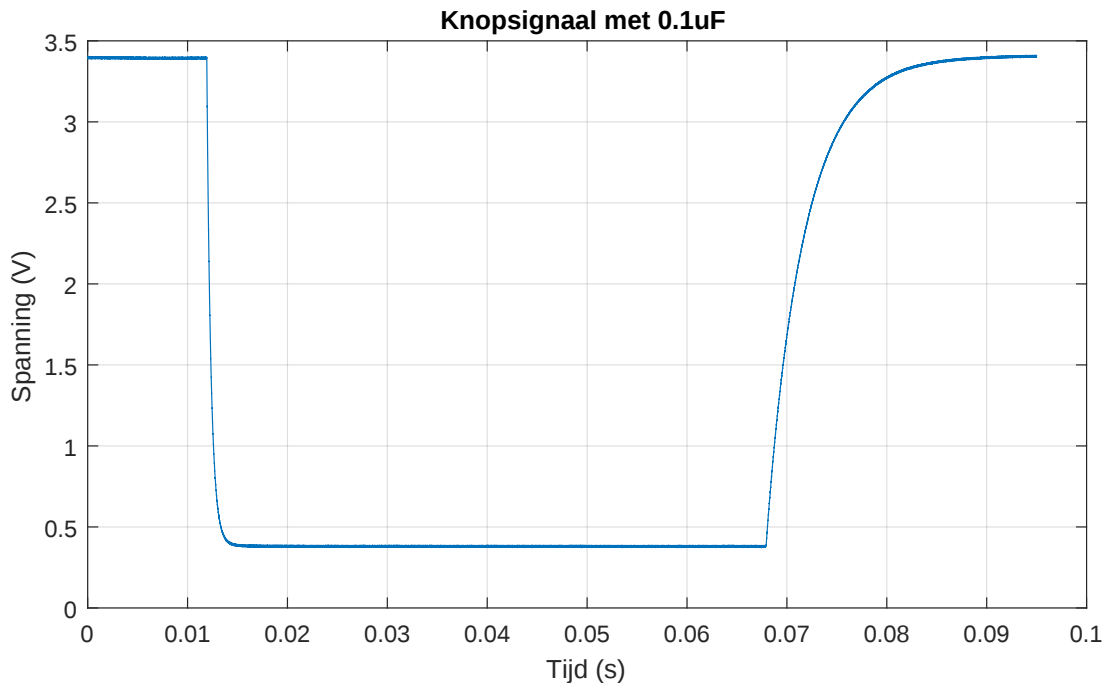
Figuur 1: Dender gemeten bij een knop van EMS10.

Geef een expressie voor de ont- en oplaadtijd van [schakeling 1](#) uitgedrukt in R_{pull} , R_{extern} en C_{dender} .



Schakeling 1: Een RC ontenderschakeling

5.1.11 Zoek in de datasheet van de MSP430G2553 op wat de waarde is van R_{pull} . In het zakje met componenten zit een condensator van $0,1\ \mu\text{F}$. Bereken de benodigde R_{extern} om de ontlaadcurve van [figuur 2](#) te krijgen.



Figuur 2: RC schakeling gemeten bij een knop van EMS10 in pull-up configuratie.

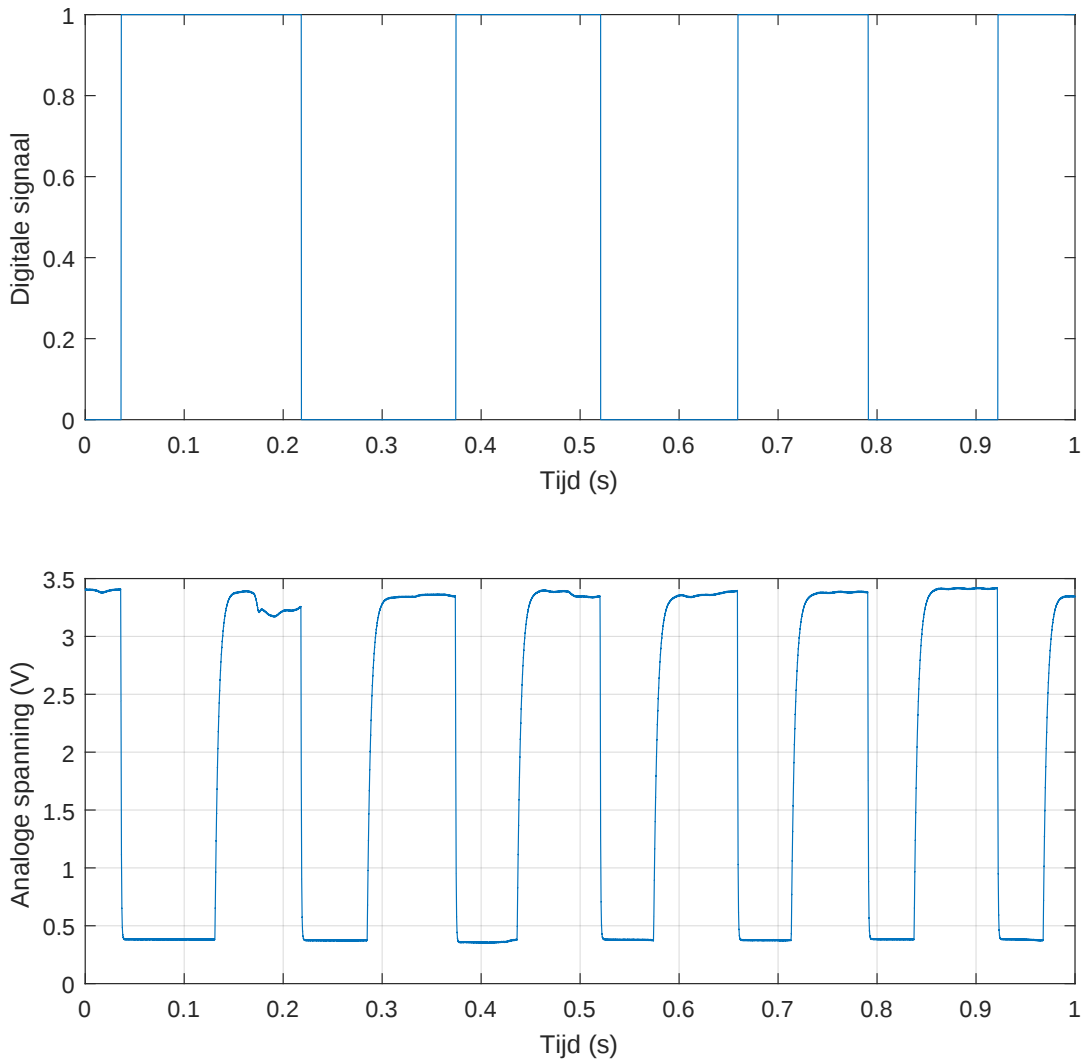
5.1.12 Teken voor jezelf de schakeling die je zou gebruiken bij een pull-down configuratie. De componentwaarden zullen hierbij niet veranderen. Bouw de nu getekende schakeling op om de knop die aangesloten is op P1.0 te ontdenderen.

5.1.13 Test opnieuw de functionaliteit van [opdracht 5.1.9](#). Als alles goed is gegaan zal het nu wel werken!

Lees nu [paragraaf 5.1 van het handboek](#).

Het filteren van het knopsignaal is slechts het begin van een knop inlezen, er kleeft namelijk nog een groot nadeel aan de code van [listing 4](#); Er wordt actief gewacht. Wat als je tussendoor nog wat andere code wilt uitvoeren? Een mogelijke oplossing hiervoor is om periodiek naar

het knopsignaal te kijken en alleen iets te doen als dit signaal is veranderd sinds de vorige keer. Voor het signaal van [figuur 2](#) zou bijvoorbeeld elke 20 ms naar de knop gekeken kunnen worden om te zien of het signaal is veranderd. Deze periode van 20 ms kan later op basis van de hardwarematige timer gebeuren. Voor nu wordt hier een softwarematige vertraging voor gebruikt.



Figuur 3: Het digitale signaal wordt geschakeld bij de neergaande flanken van het (gefilterde) knopsignaal.

5.1.14 Maak een nieuw project aan genaamd opdr_5.1.14 en kopieer het programma gegeven in [listing 5](#) in `main.c`. Test het programma door een breakpoint te zetten op regel 21 en op  te drukken. Doe dit herhaaldelijk waarbij je soms de knop ingedrukt houdt en soms loslaat. Bekijk telkens de oude en de nieuwe `knop_waarde`. Naar welke flank van het knopsignaal wordt gekeken?

5.1.15 Pas het programma aan zodat de led schakelt wanneer er 5 flanken zijn geweest (beide op- en neergaande flanken).

5.1.16 Hoe kort mag de vertraging aan het einde van de **while**-loop worden om de functionaliteit nog te behouden? Waar hangt dit van af?

```

1  #include <msp430.h>
2
3  int main(void)
4  {
5
6      WDTCTL = WDTPW | WDTHOLD; // stop de watchdog timer
7
8      P2DIR |= 1<<2 | 1<<1 | 1<<0; // zet pins P2.2, P2.1 en P2.0 op ↔
↪ output
9      P2OUT &= ~(1<<2 | 1<<1 | 1<<0); // maak pins P2.2, P2.1 en ↔
↪ P2.0 laag
10     P1DIR &= ~(1<<0); // zet pin P1.0 op input
11     P1REN |= 1<<0; // zet interne weerstand aan bij pin P1.0
12     P1OUT &= ~(1<<0); // selecteer pull down weerstand op pin P1.0.
13     int knop_waarde = 0, knop_waarde_oud = 0, knop_teller = 0;
14
15     while (1)
16     {
17         // welke waarde heeft bit0 nu?
18         knop_waarde = (P1IN & (1<<0));
19
20         // als de knop nu niet ingedrukt is EN de vorige keer de ↔
↪ knop wel was ingedrukt.
21         if (knop_waarde == 0 && knop_waarde_oud != 0)
22         {
23             knop_teller++; // hoog teller op

```

```
24
25     if (knop_teller == 3)
26     {
27         P2OUT ^= 1<<0; // inverteer de led
28         knop_teller = 0; // reset de teller
29     }
30 }
31
32 knop_waarde_oud = knop_waarde;
33 __delay_cycles(20 * 1100); // ongeveer 20ms wachten
34 }
35
36 return 0;
37 }
```

Listing 5: Na drie keer drukken invertiert de led. Zie [opdr5.1.14.c](#).

Oefening

Je hebt deze les geleerd hoe je functies in C kunt gebruiken om pinnen aan te sturen en in te lezen. Het is verstandig om een en ander uit je hoofd te leren. Flashcards kunnen je daarbij helpen. Er is een stok Anki flashcards beschikbaar zodat je ze regelmatig kunt herhalen. Deze stok kun je hier downloaden: [EMS10 Week 5 Les 1.apkg](#).

Je kunt een video bekijken waarin een deel van de stof van deze les wordt uitgelegd, zie [tabel 1](#).

Tabel 1: Video over de stof van week 5 les 1.

Link	Tijd	Beschrijving
Narcistische getallen in Python en C	32:00	Handig om kennis te maken met C variabelen en functies.

Als extra oefening kun je de theorie van vorige week nogmaals herhalen. Hiervoor kun je de Anki flashcards van week 4 gebruiken:

- [EMS10 Week 4 Les 2.apkg](#);
- [EMS10 Week 4 Les 3.apkg](#).

Daarnaast kun je ook video's bekijken waarin de stof van week 4 wordt uitgelegd, zie [tabel 2](#).

Tabel 2: Video's over de stof van week 4.

Link	Tijd	Beschrijving
Code Composer Studio	06:11	Legt uit hoe je CCS kunt gebruiken, layout, views, debuggen enz.
Opbouw van een microcontroller programma	04:51	Standaard opbouw microcontrollerprogramma wordt besproken. Waar zet je welke code neer?
Uitleg GPIO pinnen	09:54	Input, output, pull-up, pull-down wordt hier uitgelegd.
Output pin deel 1	05:09	In deel 1 van deze presentatie wordt uitgelegd hoe je een pin van de MSP430 als digitale output kan configureren.
Output pin deel 2	01:44	In deel 2 van deze presentatie wordt uitgelegd hoe je een output pin hoog kan maken.
Output pin deel 3	02:54	In deel 3 van deze presentatie wordt uitgelegd hoe je een output pin laag kan maken.
Output pin deel 4	02:45	In deel 4 van deze presentatie wordt uitgelegd hoe je een output pin kan inverteren.
Input pin deel 1	03:04	In deel 1 van deze presentatie wordt uitgelegd hoe je een pin van de MSP430 als digitale input kan configureren.
Input pin deel 2	03:54	In deel 2 van deze presentatie wordt uitgelegd hoe je kan bepalen of een digitale input pin hoog of laag is.
Pull-up of pull-down weerstand	05:14	Wanneer heb je een pull-up of pull-down weerstand nodig en hoe configureer je, in C-code, een input pin met een interne pull-up of pull-down weerstand?
Gebruik van alle bronnen	12:29	Met de diversiteit aan bronnen, waar vind je welke informatie? <i>N.B. bladwijzers in browsers werken vandaag de dag prima. Geen noodzaak meer voor een aparte PDF reader zoals in de video wordt verteld.</i>

Verdieping

Alle details over functies in C-code vind je hier: https://bitbucket.org/HR_ELEKTRO/cprog/wiki/Dictaat-C_ebook.pdf#chapter.7.

Alle details over het maken van herhalingen in C-code vind je hier: https://bitbucket.org/HR_ELEKTRO/cprog/wiki/Dictaat-C_ebook.pdf#chapter.4.