

Opdrachten week 4 les 3 – Leds en knoppen

In de vorige les is aandacht besteed aan het werken op de MSP430 Launchpad. Er is een Integrated Developers Environment geïnstalleerd (CCS) en het eerste programma is ingeladen om een led aan en uit te zetten. In deze les leer je meer over het aansturen van pinnen. Je leert hoe je:

- in C code een beslissing maakt;
- de MSP430G2xx Family User's Guide en MSP430G2x53 Datasheet kunt gebruiken om informatie op te zoeken;
- een breakpoint kunt instellen om de code te pauzeren op een specifieke regel;
- een knop kunt uitlezen.

In de volgende les leer je functies te maken om zo structuur aan te brengen in je programma.

Een embedded systeem draait autonoom en moet over lange tijd blijven opereren. Eigenlijk zijn deze systemen gemaakt om altijd aan te staan. In code resulteert dit in een oneindige loop. Deze oneindige loop wordt geïmplementeerd met het **while**-statement en binnen de 'body' hiervan zal het embedded programma staan.

Zoals te lezen in [paragraaf 2.2](#) van het boek werkt **while**(1) omdat 1 als true (waar) wordt gezien.

C does *not* have a Boolean type for representing true/false values. This has major implications for a statement like **if**, where you need a test to determine whether to execute the body. C's approach is to treat the integer 0 as *false* and all other integer values as *true*.¹

Je gaat nu de **while**-loop toepassen en kijken hoe hiermee de structuur van het programma wordt aangepast ten opzichte van vorige opdrachten. Zorg ervoor dat aan je microcontroller de RGB-led is aangesloten op P1.0, P1.1 en P1.2.

¹ Deze informatie is verouderd. In de C99-standaard uit 1999 is wel degelijk een type `bool` opgenomen. Om dit type te kunnen gebruiken moet je wel de regel:

```
#include <stdbool.h>
```

in je programma opnemen. In C99 kun je dus ook **while**(true) gebruiken in plaats van **while**(1). CCS staat vreemd genoeg per default ingesteld op een oude versie van de C-standaard: C89 (uit 1989). De laatste versie van de C-standaard die wordt ondersteund door CCS is C11 (uit 2011). In 2018 is nog een nieuwe versie van de C-standaard verschenen, maar die introduceert geen nieuwe features t.o.v. C11; er worden alleen een paar onduidelijkheden opgehelderd. Kies **Project** » **Properties** » **Build** » **MSP430 Compiler** » **Advanced Options** » **Language Options** en selecteer bij "C Dialect" de optie "Compile program in C11 mode".

```
1 #include <msp430.h>
2 #include <stdint.h>
3
4 int main()
5 {
6     WDTCTL = WDTPW | WDTHOLD; // Stop de watchdog timer
7
8     // Eerst alle code die 1 keer uitgevoerd moet worden
9     // Zoals bijvoorbeeld het instellen van de I/O
10    P1OUT = 0x00;
11    P1DIR = 0x07;
12    // en het declareren van variabelen
13    volatile uint16_t wachttijd;
14
15    while (1)
16    {
17        // Hier de rest van het programma
18        // dat continue uitgevoerd moet worden
19        P1OUT ^= 1<<1;
20
21        wachttijd = 60000;
22        while (wachttijd) {
23            wachttijd--;
24        }
25    }
26 }
```

Listing 1: Voorbeeld gebruik `while`-loop. Zie [les43_whileloop.c](#)

4.3.1 Maak een nieuw project aan en noem dit opdr_4.3.1. Vervang de code in `main.c` met [listing 1](#), compileer het project en voer het programma uit op de microcontroller. Wat gebeurt er?

4.3.2 Pas de code van [opdracht 4.3.1](#) aan om de led twee keer zo snel te laten knipperen.

Deze manier van programmaopbouw vereist een iets andere denkwijze. Onze code wordt namelijk niet eenmalig uitgevoerd, maar oneindig lang herhaald. Er is dus extra denkwerk nodig om ervoor te zorgen dat dingen niet te vaak gebeuren, niet te snel gebeuren, maar wel goed gebruik maken van de herhaling.

4.3.3 Je gaat de leds nu anders aansturen. Verwijder de code van regel 19. Maak een variabele genaamd `it` aan om het aantal iteraties van de `while(1)`-loop te tellen. Waar in de code zet je de variabele-declaratie neer? Vergeet niet om de variabele bij 0 te laten starten. Maak het P1OUT-register gelijk aan de `it` variabele. Verhoog de variabele `it` met 1 telkens als een loop eindigt. Regels 21 t/m 24 van [listing 1](#) mogen blijven staan, dit zorgt immers voor de vertraging tussen de verschillende iteraties. Waarom tellen de leds binair op? Waarom herhaalt het zich automatisch?

Zorg dat je hoofdstuk 4 tot met paragraaf 4.3 van het handboek² helemaal begrijpt. Vraag, indien nodig, de docent om hulp!

4.3.4 Je gaat de leds nu weer anders aansturen. Verwijder de regel `P1OUT = it;` uit je programma. Het doel van deze opdracht is om P1.0 en P1.1 nog steeds binair te laten optellen, zonder dat P1.2 hierdoor wordt aangepast. Dit is nog niet zo eenvoudig. We willen de laagste twee bits van de variabele `it` als het ware combineren met de hoogste 6 bits van het P1OUT-register.

Dit kun je als volgt doen:

- maak een integer variabele genaamd `var1` aan en geef deze de waarde van het P1OUT-register;
- maak de laagste twee bits van `var1` nul;
- maak een integer variabele genaamd `var2` aan en geef deze de waarde van de variabele `it`;
- maak de hoogste zes bits van `var2` nul;
- voer een OR-operatie uit op `var1` en `var2` en schrijf het resultaat naar het P1OUT-register.

De variabelen `var1` en `var2` zijn in de bovenstaande stappen gebruikt om de resultaten van de tussenstappen op te slaan. Deze variabelen zijn echter niet per se nodig. Indien gewenst kunnen de bovenstaande stappen allemaal in één regel C-code gecodeerd worden:

```
P1OUT = (P1OUT & 0b11111100) | (it & 0b00000011);
```

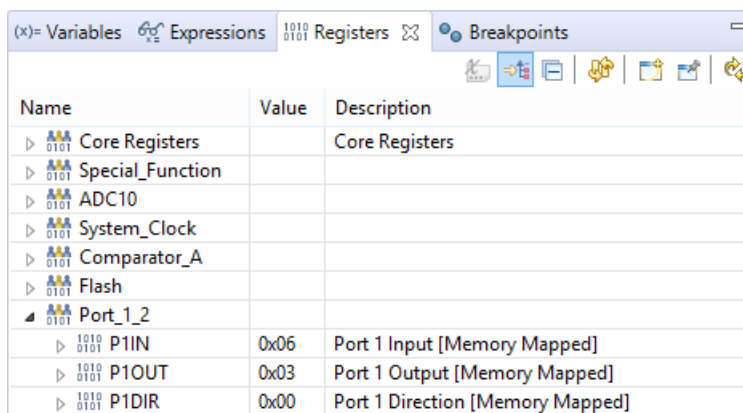
² Daniël Versluis. *Microprocessor Programmeren in C*. 2018. URL: https://bytebucket.org/HR_ELEKTRO/ems10/wiki/Handboek/EMS10_handboek_ebook.pdf

De stap-voor-stap-aanpak is echter vaak makkelijker te begrijpen en te debuggen en wordt dus aanbevolen.

4.3.5 Nu moet het programma uitgebreid worden zodat de led op P1.2 elke derde iteratie geschakeld worden. Deze led moet steeds gedurende drie iteraties aan en dan weer gedurende drie iteraties uit staan, zie [tabel 1](#). Je hebt hier een aantal tools voor nodig: het **if**-statement, de modulusoperator %, een correct samengestelde voorwaarde voor het **if**-statement en als laatste een stukje code om uit te voeren als deze voorwaarde waar is.

- Hoe kun je modulusoperator gebruiken om te kijken of de variabele `i` deelbaar is door drie? Verzin een expressie die waar wordt wanneer `i` deelbaar is door drie en gebruikt kan worden in een **if**-statement.
- Zet het **if**-statement op, inclusief de body ervan waarin de led geschakeld wordt.

4.3.6 Zet een breakpoint  op de regel waar `i` wordt opgehoogd. Dit doe je door op de betreffende regel te gaan staan en op `ctrl` + `↑` + `B` te drukken, je kunt ook links van het regelnummer dubbelklikken. Stap nu door de iteraties heen door herhaaldelijk op  te drukken. De uitvoer zal telkens stoppen bij het breakpoint. Als [opdracht 4.3.4](#) en [opdracht 4.3.5](#) goed zijn uitgevoerd moet het resultaat aan [tabel 1](#) voldoen. Is dit niet het geval ga dan terug naar [opdracht 4.3.4](#) of [opdracht 4.3.5](#) om het werkend te krijgen. Het is makkelijk om hierbij het *Registers* venster erbij te openen om de bits van P1OUT te bekijken via de menuoptie `Window >> Show View >> Registers`, zie [figuur 1](#).



Name	Value	Description
Core Registers		Core Registers
Special_Function		
ADC10		
System_Clock		
Comparator_A		
Flash		
Port_1_2		
P1IN	0x06	Port 1 Input [Memory Mapped]
P1OUT	0x03	Port 1 Output [Memory Mapped]
P1DIR	0x00	Port 1 Direction [Memory Mapped]

Figuur 1: Het Registers venster van CCS.

Tabel 1: Tijdsverloop leds.

Iteratie #	P1.2	P1.1	P1.0
0	1	0	0
1	1	0	1
2	1	1	0
3	0	1	1
4	0	0	0
5	0	0	1
6	1	1	0
7	1	1	1
8	1	0	0
9	0	0	1
10	0	1	0
11	0	1	1
⋮			

Een andere toepassing van de bitwise-operaties uit C is om juist te *kijken* naar een bitpositie in plaats van het *aanpassen* hiervan. Dit is bijvoorbeeld noodzakelijk bij het inlezen van een knop.

4.3.7 Sluit een knop aan op pin P2.3. Sluit de knop aan volgens [figuur 2a](#).

Lees nu paragraaf 8.2.2 en 8.2.4 van de user's guide. De user's guide is op het internet beschikbaar³.

4.3.8 Stel de pin waar de knop op is aangesloten in als een ingang. Om de pull-down weerstand te gebruiken moeten de registers P2OUT en P2REN ingesteld worden. Zoek aan de hand van de User's Guide uit welke waarde de bits voor de knop-pin moet krijgen en stel deze in. In de volgende opdracht ga je deze instellingen gebruiken om

³ User's Guide: <http://www.ti.com/lit/ug/slau144k/slau144k.pdf>.
 Datasheet: <http://www.ti.com/lit/ds/symlink/msp430g2553.pdf>.

de toestand van de knop in te lezen. Vraag eventueel aan je docent of je de registers P2OUT en P2REN correct hebt ingesteld.

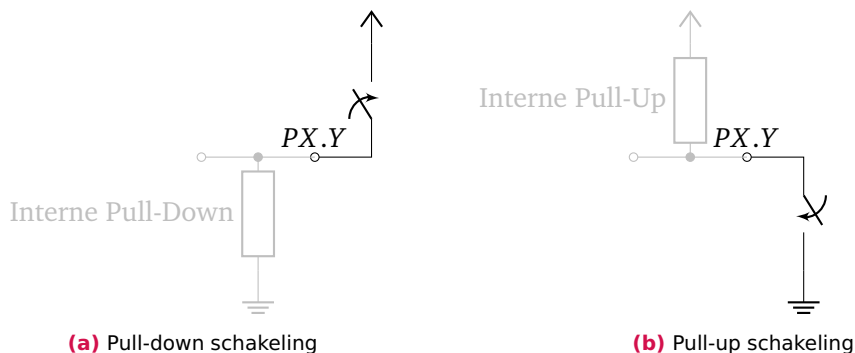
Lees nu **paragraaf 4.5 van het handboek⁴** waarin staat beschreven hoe een bit getest kan worden.

4.3.9 Gebruik deze informatie om P2IN te testen op het wel of niet indrukken van de knop. Doe dit als aanvulling op het programma van **opdracht 4.3.5**. Zorg ervoor dat alle leds uit zijn zolang (**while**) de knop is ingedrukt. Verwijder het breakpoint en test je programma. Waarom duurt het soms even een korte tijd voordat de leds uitgaan?

Lees **paragraaf 2.6 van het boek⁵**.

4.3.10 Becommentarieer de code van **opdracht 4.3.9** met een paar commentaarblokken. Pas nu de rest van de programmacode aan zodat de **while(1)** pas begint nadat de knop een keer is ingedrukt (en losgelaten).

4.3.11 Verander de knop aansluiting naar het alternatief in **figuur 2b**. Pas de code aan om je code weer werkend te krijgen. Welke van de opties in **figuur 2** vind je beter en waarom?



Figuur 2: Schakeling om een knop op de μC aan te sluiten

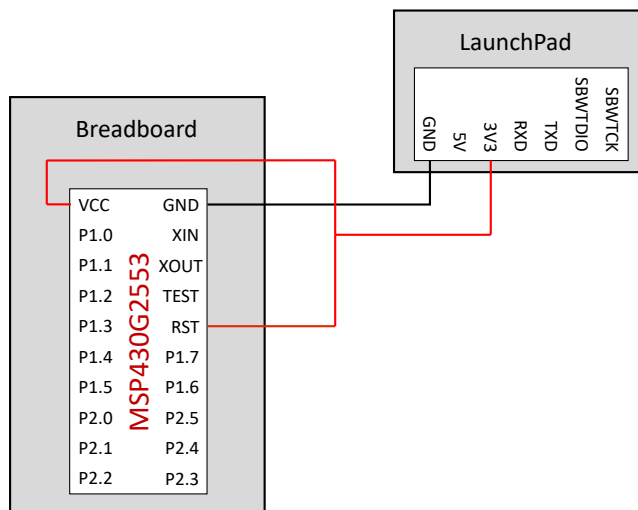
⁴ Daniël Versluis. *Microprocessor Programmeren in C*. 2018. URL: https://bytebucket.org/HR_ELEKTRO/ems10/wiki/Handboek/EMS10_handboek_ebook.pdf.

⁵ Carl Burch. *C for Python programmers*. 2011. URL: <http://www.cburch.com/books/cpy/>.

4.3.12 Sluit de knop nu aan op pin P1.3 in plaats van P2.3. Pas de code aan om je code weer werkend te krijgen. Je moet er nu op letten dat je bij het aansturen van de leds, niet per ongeluk de pull-up weerstand verandert in een pull-down weerstand.

In de praktijk wil je de geprogrammeerde MSP430G2553 gebruiken in je eigen hardware-schakeling zonder de LaunchPad te gebruiken. Het programma dat in de MSP430G2553 staat, wordt in flashgeheugen⁶ opgeslagen en blijft dus behouden ook als de spanning wegvalt. Om het programma op de MSP430G2553 uit te voeren zonder dat de LaunchPad wordt gebruikt, moet alleen een voedingsspanning en een resetsignaal aangesloten worden.

4.3.13 Verwijder alle verbindingen tussen de MSP-EXP430G2ET LaunchPad en je breadboard. Verbind daarna alleen de GND en de VCC pinnen van de MSP430G2553 met respectievelijk de GND en 3V3 van de LaunchPad. Verbind ook de RST pin van de MSP430G2553 met de VCC. Zie [figuur 3](#). Als het goed is, wordt nu het laatste in de MSP430G2553 geprogrammeerde programma uitgevoerd.



Figuur 3: Aansluiten geprogrammeerde microcontroller op breadboard met alleen de voedingsspanning.

⁶ Zie eventueel: <https://nl.wikipedia.org/wiki/Flashgeheugen>.

Oefening

Ook voor deze les is een stok Anki flashcards beschikbaar. Vergeet niet om de stok van de vorige les ook nog regelmatig te herhalen. De stok voor deze les kun je hier downloaden: [EMS10 Week 4 Les 3.apkg](#).

Daarnaast zijn er een aantal video's beschikbaar waarin de stof van deze les nogmaals wordt uitgelegd, zie [tabel 2](#).

Verdieping

Alle details over de bitwise-operaties in C vind je hier: https://bitbucket.org/HR_ELEKTRO/cprog/wiki/Dictaat-C_ebook.pdf#chapter.9.

Alle details over het maken van beslissingen in C-code vind je hier: https://bitbucket.org/HR_ELEKTRO/cprog/wiki/Dictaat-C_ebook.pdf#chapter.5.

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>