

Opdrachten week 6 les 1 – Interrupts

In de opdrachten van de vorige weken heb je de MSP430G2553 leren programmeren in C met behulp van CCS. Je weet nu hoe je digitale pinnen van de MSP430G2553 kunt aansturen en inlezen. Je weet dat een schakelaar kan denderen, waarom dit een probleem kan zijn en hoe je deze dender kan wegfilteren. Je kan de interne klok van de MSP430G2553 instellen en je kunt de microcontroller een bepaalde tijd laten wachten. Als laatste heb je geleerd hoe je een toestandsmachine kunt implementeren als een programma in een microcontroller.

Deze les ga je leren hoe je zogenoemde interrupts kunt gebruiken om te reageren op een bepaalde gebeurtenis, zonder dat je de hele tijd zelf in de gaten hoeft te houden of deze gebeurtenis al heeft plaatsgevonden. Het draaiende programma kan namelijk onderbroken worden door een interrupt. Het Engelse werkwoord to interrupt betekent onderbreken. Na het afhandelen van de interrupt wordt het onderbroken programma weer hervat op de plaats waar het onderbroken is.

In plaats van interrupts te gebruiken kun je ook zelf in de gaten houden of een bepaalde gebeurtenis heeft plaatsgevonden, we noemen dit ‘pollen’. Na deze les weet je de verschillen tussen polling en interrupts en ken je van beide methoden de voor- en nadelen.

Tot slot van deze les leer je hoe je de MSP430G2553 in een zogenoemde low-power mode kunt schakelen, als er niets anders te doen is dan wachten op de volgende interrupt.

We beginnen deze les met een voorbeeldprogramma waarin de leerstof uit de vorige twee weken is toegepast. Daarbij wordt ervan uitgegaan dat er een RGB-led is aangesloten op P2.0 (Rood), P2.1 (Groen) en P2.2 (Blauw). We gebruiken dit programma als de basis voor de opdracht waarin je voor het eerst interrupts gaat toepassen.

Lees **hoofdstuk 6 van het handboek**¹.

6.1.1 In **listing 1** is een programma gegeven dat een rode led laat knipperen met een frequentie van 0,25 Hz (2 s uit en 2 s aan). De klokfrequentie van de MSP430G2553 wordt ingesteld op 1 MHz. De standaardfunctie `__delay_cycles` wordt gebruikt om 2 s te wachten. Test dit programma op jouw MSP430G2553.

¹ 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
    if (CALBC1_1MHZ == 0xFF)
    {
        while(1); // Doe niets
    }
    DCOCTL = 0; // Klokfrequentie = 1 MHz
    BCSCTL1 = CALBC1_1MHZ;
    DCOCTL = CALDCO_1MHZ;

    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(2000000);
        P2OUT ^= 1<<0; // inverteer pin P2.0
    }

    return 0;
}

```

Listing 1: programma dat de rode led laat knipperen met een frequentie van 0,25 Hz. Zie [opdr6.1.1.c](#)

6.1.2 Sluit een drukknop (SW0) aan op pin P1.0 van de MSP430G2553 in de pull-down configuratie. Zorg ervoor dat deze drukknop hardwarematig ontdenderd is.

Breid het programma uit [opdracht 6.1.1](#) nu uit zodat de groene led *meteen* gaat branden zodra drukknop SW0 wordt ingedrukt. De groene led moet blijven branden als SW0 wordt losgelaten en moet *meteen* doven als SW0 nogmaals wordt ingedrukt. Deze uitbreiding moet gerealiseerd worden *zonder* de `while(1)`-loop in het programma uit [opdracht 6.1.1](#) aan te passen. Dit kan door gebruik te maken van een zogenoemde pin-change interrupt.

Zoek zelf de benodigde informatie op in paragraaf 8.2.7 van de MSP430x2xx Family User's Guide².

Je moet dus in je C-code:

- er voor zorgen dat er een pin-change interrupt wordt gegenereerd als knop SW0 wordt ingedrukt;
- een Interrupt Service Routine (ISR) definiëren die de groene led inverteert en de interrupt flag reset;
- er voor zorgen dat de pin-change interrupt van P1.0 gekoppeld wordt aan de gedefinieerde ISR;
- alle interrupts aanzetten.

Zoek zelf de benodigde informatie op in hoofdstuk 6 en bijlage A van het handboek.

6.1.3 Sluit ook een drukknop (SW1) aan op de pin P1.1 van de MSP430G2553. Deze drukknop moet aangesloten worden in de pull-up configuratie. Zorg ervoor dat deze drukknop hardwarematig ontdenderd is. Het uitgangssignaal van SW1 zal dus nul worden als de knop wordt ingedrukt. Pas het programma uit opdracht 6.1.2 nu zo aan dat de blauwe led meteen gaat branden als SW1 wordt ingedrukt. De blauwe led moet blijven branden als SW1 wordt losgelaten en moet *meteen* doven als SW1 nogmaals wordt ingedrukt. Maak hierbij gebruik van een pin-change interrupt die reageert op een verandering op een van beide pinnen.

6.1.4 In listing 2 is een programma gegeven dat de hele statemachine van opdracht 5.2.3 realiseert in een pin pin-change interrupt. De `while(1)`-loop in het hoofdprogramma is nu volledig leeg. De huidige toestand is, tussen de interrupts door, bewaard in een zogenoemde `static` lokale variabele³. Download dit programma via deze link: [opdr6.1.4.c](#) en test dit programma op jouw MSP430G2553.

² Zie: <http://www.ti.com/lit/ug/slau144k/slau144k.pdf>.

³ Een normale lokale variabele wordt telkens als de functie wordt aangeroepen, opnieuw aangemaakt en aan het einde van de functie weer verwijderd. Op deze manier maak je, door gebruik te maken van lokale variabelen, optimaal gebruik van het beschikbare RAM-geheugen. In sommige gevallen kan het nodig zijn dat een lokale variabele zijn waarde behoudt nadat de functie waarin deze variabele is gedefinieerd is verlaten. De eerstvolgende keer dat die functie wordt aangeroepen is deze variabele, met zijn huidige waarde, dan weer beschikbaar. Om dit te realiseren kan in C het keyword `static` voor de betreffende variabele definitie worden geplaatst.

```
#include <msp430.h>

void zet_led_uit(int lednummer)
{
    P2OUT &= ~(1<<(lednummer - 1));
}

void zet_led_aan(int lednummer)
{
    P2OUT |= 1<<(lednummer - 1);
}

#pragma vector = PORT1_VECTOR
__interrupt void port1_isr(void)
{
    typedef enum {opslot, dicht, open} Toestand;
    static Toestand toestand = opslot;

    switch (toestand)
    {
    case opslot:
        if ((P1IFG & 1<<0) != 0)
        {
            P1IFG &= ~(1<<0); // clear interrupt flag
            zet_led_uit(1);
            zet_led_aan(3);
            toestand = dicht;
        }
        if ((P1IFG & 1<<1) != 0)
        {
            P1IFG &= ~(1<<1); // clear interrupt flag
        }
        break;
    case dicht:
        if ((P1IFG & 1<<0) != 0)
        {
            P1IFG &= ~(1<<0); // clear interrupt flag
            zet_led_uit(3);
            zet_led_aan(1);
            toestand = opslot;
        }
    }
```

```

    if ((P1IFG & 1<<1) != 0)
    {
        P1IFG &= ~(1<<1); // clear interrupt flag
        zet_led_uit(3);
        zet_led_aan(2);
        toestand = open;
    }
    break;
case open:
    if ((P1IFG & 1<<0) != 0)
    {
        P1IFG &= ~(1<<0); // clear interrupt flag
    }
    if ((P1IFG & 1<<1) != 0)
    {
        P1IFG &= ~(1<<1); // clear interrupt flag
        zet_led_uit(2);
        zet_led_aan(3);
        toestand = dicht;
    }
    break;
}
}

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD; // stop de watchdog timer
    if (CALBC1_1MHZ != 0xFF)
    {
        DCOCTL = 0; // Select lowest DCOx and MODx settings
        BCSCTL1 = CALBC1_1MHZ; // Set range
        DCOCTL = CALDCO_1MHZ; // Set DCO step + modulation*/

        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
        zet_led_aan(1);

        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.
    P1IE |= 1<<0; // zet interrupt aan bij pin P1.0
    P1IES &= ~(1<<0); // interrupt op opgaande flank
    P1IFG &= ~(1<<0); // clear interrupt flag

    P1DIR &= ~(1<<1); // zet pin P1.1 op input
    P1REN |= 1<<1; // zet interne weerstand aan bij pin P1.1
    P1OUT |= 1<<1; // selecteer pull up weerstand op pin P1.1.
    P1IE |= 1<<1; // zet interrupt aan bij pin P1.1
    P1IES |= 1<<1; // interrupt op neergaande flank
    P1IFG &= ~(1<<1); // clear interrupt flag

    __enable_interrupt(); // zet interruptsysteem aan

    while (1)
    {
    }

}
return 0;
}

```

Listing 2: Implementatie van de statemachine van [opdracht 5.2.3](#). Zie [opdr6.1.4.c](#)

De processor heeft nu niets anders meer te doen dan geduldig te wachten in een lege **while**(1)-loop tot de volgende interrupt zich aandient. Het is in zo'n geval mogelijk om de MSP430G2553 in een zogenoemde low-power mode te zetten waardoor de μ C bijna geen stroom meer gebruikt.

Lees nu paragraaf 2.3 van de van de MSP430x2xx Family User's Guide.

Om de MSP430G2553 in één van de low-power modes te zetten moeten bepaalde bits in het status register (SR) worden gezet. Dit kun je echter niet op de gebruikelijke manier doen omdat het SR geen I/O-register is maar een intern register in de μ C. Vanuit de programmeertaal C kun je de interne registers van een processor echter *niet* bereiken. Om nu toch vanuit een C-programma naar de low-power modes te kunnen gaan zijn na het includen van `msp430.h` de volgende functies beschikbaar:

- `__low_power_mode_0()`
- `__low_power_mode_1()`

- `__low_power_mode_2()`
- `__low_power_mode_3()`
- `__low_power_mode_4()`

6.1.5 Vervang de **while**(1)-loop in het programma van [opdracht 6.1.4](#) met een aanroep naar een zo zuinig mogelijke low-power mode.

6.1.6 Bedenk waarom het in de applicatie van [opdracht 6.1.4](#) niet zo veel uitmaakt dat de MSP430G2553, tussen de interrupts door, in een low-power mode wordt gezet. Bedenk welke component het meeste stroom gebruikt. Is dat de MSP430G2553?

Verdieping

De MSP-EXP430G2ET LaunchPad is uitgerust met zogenoemde Energy Trace Technology (vandaar de letters ET aan het einde van de naam van de LaunchPad).

6.1.7 Met behulp van de Energy Trace Technology kun je het energieverbruik van de schakeling op je breadboard meten. Je kunt dit doen door na het starten van de debugger met de -knop de menuoptie  EnergyTrace te gebruiken. Het 'EnergyTrace Technology' window verschijnt dan linksonder in het scherm. Als je op het tabblad 'Power' klikt, dan kun je (bijna) real-time het door de MSP430G2553 opgenomen vermogen zien nadat je het programma met de -knop hebt gestart. Als je na enkele seconden op SW0 drukt en nog een paar seconden later op SW1 dan verschijnt een grafiek zoals weergegeven in [figuur 1](#). Met behulp van het opgenomen vermogen kun je ook eenvoudig de opgenomen stroom bepalen. Je ziet dat elke led een ander stroomgebruik heeft (als alle serieweerstanden 1 kΩ zijn). Lees [opdracht 4.2.12](#) nogmaals door als je dit niet begrijpt.

6.1.8 Pas het programma uit [opdracht 6.1.5](#) aan, zodat er geen enkele led brandt als de deur dicht en op slot is. Debug het programma en bekijk het energieverbruik in het EnergyTrace window. Als je na enkele seconden kort op SW0 drukt en nog een paar seconden later nogmaals kort op SW0 drukt, dan verschijnt een grafiek zoals weergegeven in [figuur 2](#). Volgens [paragraaf 2.3](#) van de MSP430x2xx Family User's



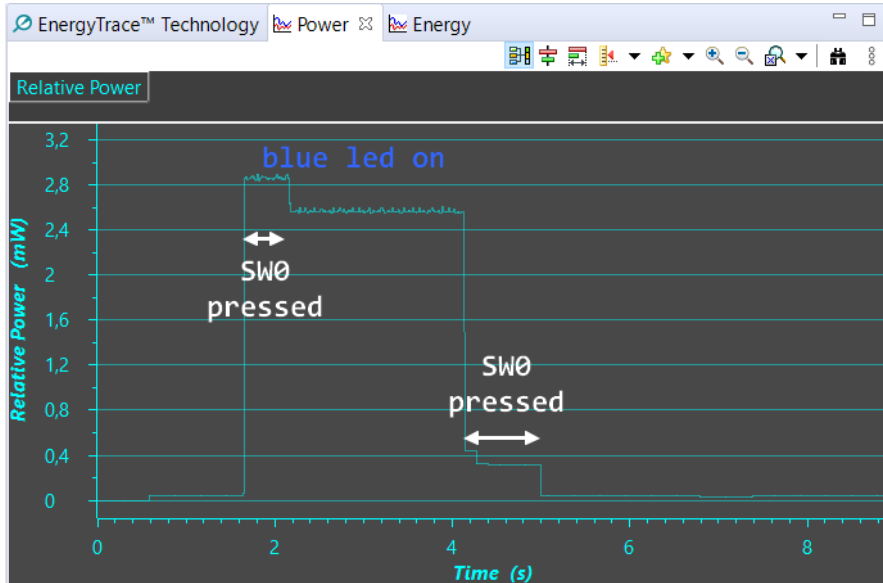
Figuur 1: Het energiegebruik van het programma uit opdracht 6.1.5.



Figuur 2: Het energiegebruik van het programma uit opdracht 6.1.8.

Guide zou het stroomgebruik in LPM4 gelijk moeten zijn aan $0,1 \mu\text{A}$. De gemeten stroom is echter $0,4\text{mW}/3,3\text{V} \approx 0,1\text{mA}$. Dat is een factor 1000 te hoog!

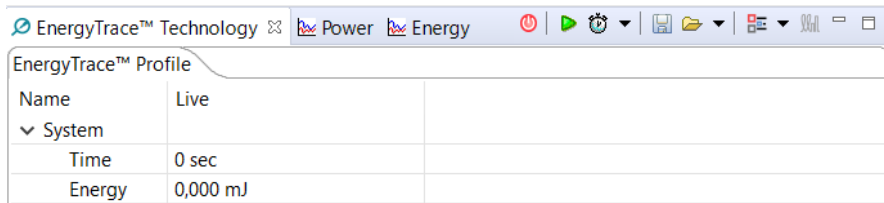
6.1.9 Het te hoge stroomgebruik blijkt veroorzaakt te worden door zwevende inputs. Dit wordt uitgelegd in [paragraaf 8.2.8](#) van de MSP430x2xx Family User's Guide. Pas het programma uit [opdracht 6.1.8](#) nu aan, zodat alle niet gebruikte pinnen als output geconfigureerd worden. Debug het programma en bekijk het energiegebruik in het EnergyTrace window. Als je na enkele seconden op SW0 drukt, die even kort ingedrukt houdt en nog een paar seconden later nogmaals op SW0 drukt en die weer even kort ingedrukt houdt, dan verschijnt een grafiek zoals weergegeven in [figuur 3](#). Snap je



Figuur 3: Het energiegebruik van het programma uit [opdracht 6.1.9](#).

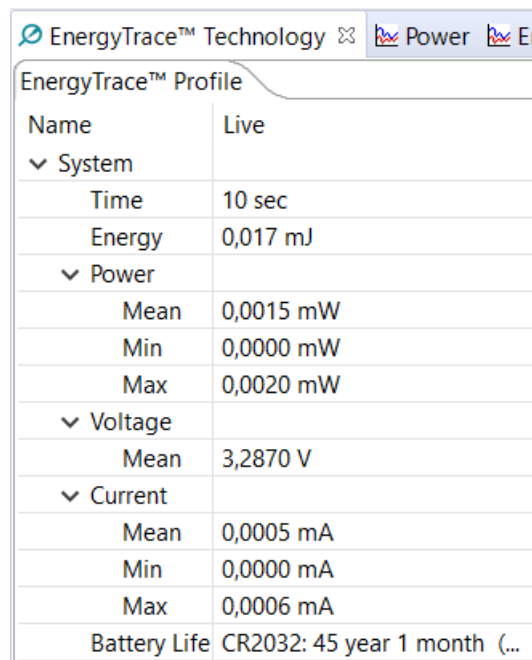
waarom er meer stroom opgenomen wordt gedurende de tijd dat de knop ingedrukt is? De gemeten stroom in de ruststand is nu ongeveer $0,05 \text{ mW} / 3,3 \text{ V} \approx 15 \mu\text{A}$. Nog steeds een factor 150 te hoog!

6.1.10 Het blijkt dat het uitvoeren van het programma in debug mode extra energie kost. Je moet dus een meting uitvoeren zonder het programma te debuggen. Dit doe je door in de edit mode op de -knop te klikken. Het 'EnergyTrace Technology' window verschijnt dan linksonder in het scherm, zie [figuur 4](#). Met de -knop kun je de duur van de meting instellen (staat standaard op 10 s). Met de -knop kun je de meting starten. In [figuur 5](#) zie je een mogelijk resultaat. Het gemeten stroomgebruik is nu $5 \mu\text{A}$ en komt in de buurt van de in de Family User's Guide genoemde waarde van $1 \mu\text{A}$.



Figuur 4: Het 'EnergyTrace Technology' window.

We zien dat de MSP430G2553 ruim 45 jaar in deze LPM4 kan blijven indien de chip gevoed wordt met een CR2032 knoopcel, zie [figuur 6](#).



Figuur 5: Het stroomgebruik in rust van het programma uit [opdracht 6.1.9](#).

In de [datasheet](#) van de MSP430G2553 is gegeven dat het stroomgebruik in low-power mode 4 typisch 1 μA is en maximaal 5 μA is bij kamertemperatuur (25 °C). Het stroomgebruik loopt op bij hogere temperaturen.

De volgende opdracht is bedoeld als extra oefening met de pin-change interrupts en low-power modes.



Figuur 6: Een CR2032 knoopcel batterij.

6.1.11 Pas het codeslot dat je hebt gemaakt bij [opdracht 5.3.2](#) aan zodat de knoppen ingelezen worden met pin-change interrupts. Zorg ervoor dat de MSP430G2553 zo min mogelijk energie gebruikt als het codeslot niet bediend wordt. Verifieer dit met behulp van het EnergyTrace window in Code Composer Studio.