Opdrachten week 6 les 2 – Timers

In de vorige les heb je geleerd om met interrupts te werken en heb je geleerd hoe je de MSP430G2553 in een low-power mode kunt zetten. Tot nu toe heb je, als je een bepaalde tijd wilde wachten, gebruik gemaakt van de standaardfunctie `__delay_cycles`. In deze functie telt de CPU in de µC het als argument meegegeven aantal klokcycli af. Deze manier van wachten wordt *busy waiting* genoemd. We kunnen de µC in dit geval dus *niet* in een low-power mode zetten tijdens het wachten, omdat dan het tellen van cycli stopt en het wachten dus nooit meer eindigt. Om deze reden zijn de meeste microcontrollers voorzien van specifieke peripherals, timers genaamd, die gebruikt kunnen worden om cycli te tellen zonder dat daarvoor de CPU gebruikt hoeft te worden.

Deze les implementeer je een aantal programma's die allemaal een ledje met een frequentie van 1 Hz laten knipperen (0,5 s uit en 0,5 s aan). In het begin gebruiken we de CPU nog om de tijd af te tellen, maar in de uiteindelijke applicatie hebben we de CPU helemaal niet meer nodig. Bij het schrijven van deze programma's zul je zelf de benodigde gegevens in de MSP430x2xx Family User's Guide en in de MSP430G2x53 Datasheet moeten opzoeken.

In deze les leer je hoe je de timer peripheral van de MSP430G2553 kunt gebruiken om:

- tijd af te tellen;
- met een vaste regelmaat een interrupt op te wekken;
- met een vaste regelmaat een pin te toggelen.

Lees nu hoofdstuk 7 van het handboek[1].

De MSP430G2553 bevat, zoals je op pagina 1 van de MSP430G2x53 Datasheet[2] kunt lezen, twee zogenoemde 16-bit Timer_A peripherals elk met drie capture/compare registers.

Lees nu paragraaf 12.1 en 12.2 tot 12.2.3.1 van de MSP430x2xx Family User's Guide[3].

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

[2] Deze kun je vinden in de Resource Explorer van CCS of online op: http://www.ti.com/lit/ds/symlink/msp430g2553.pdf.

[3] Deze kun je vinden in de Resource Explorer van CCS of online op: http://www.ti.com/lit/ug/slau144k/slau144k.pdf.

Je hebt net gelezen dat je, als kloksignaal voor de timer, kunt kiezen uit twee interne signalen (ACLK of SMCLK) of uit twee externe signalen (TACLK of INCLK). In dit geval willen we gebruik maken van een interne klok.

Lees nu paragraaf 5.1 van de MSP430x2xx Family User's Guide.

Je hebt net gelezen dat je voor ACLK kunt kiezen uit LFXT1CLK of VLOCLK en dat je voor SMCLK kunt kiezen uit LFXT1CLK, VLOCLK, XT2CLK (indien beschikbaar on-chip), of DCOCLK.

Na een reset van de µC is het SMCLK signaal gelijk aan het DCOCLK signaal, zoals je in de eerste zin van paragraaf 5.2 van de MSP430x2xx Family User's Guide kunt lezen.

In de vorige lessen heb je geleerd hoe je de DCO-klokfrequentie in kunt stellen. Als we de instellingen van de Basic Clock Module+ niet wijzigen, dan is het SMCLK signaal gelijk aan het DCOCLK signaal.

Je kunt het gekozen kloksignaal van de timer nog delen door 1, 2, 4 of 8, zoals je hebt gelezen in paragraaf 12.2.1.1 van de MSP430x2xx Family User's Guide.

Lees nu paragraaf 12.2.3.3 van de MSP430x2xx Family User's Guide.

Je gaat nu Timer_A0 gebruiken in de Continuous mode om een halve seconde af te tellen.

Let op! De MSP430x2xx Family User's Guide bevat een algemene beschrijving van Timer_A. De MSP430G2553 bevat twee van deze timers: Timer_A0 en Timer_A1 genaamd. Voor alle in de MSP430x2xx Family User's Guide genoemde timer register moeten we zelf een `0` of `1` toevoegen om aan te geven welke timer we willen gebruiken. Bijvoorbeeld in paragraaf 12.3.2 van de MSP430x2xx Family User's Guide wordt gezegd dat je de tellerstand van Timer_A kunt lezen/schrijven via het register genaamd `TAR`. Concreet betekent dat in ons geval dat je tellerstand van Timer_A0 kunt lezen/schrijven via het register genaamd `TA0R`. De tellerstand van Timer_A1 kun je lezen/schrijven via het register genaamd `TA1R`.

6.2.1 Maak een nieuw project aan en noem dit `opdr_6.2.1`. Zorg ervoor dat de RGB-led is aangesloten op P2.0 (Rood), P2.1 (Groen) en P2.2 (Blauw). Stel de DCO in op een klokfrequentie van 1 MHz. Zet de pinnen waar de led op aangesloten is op output. Stel Timer_A0 nu als volgt in met behulp van het register `TA0CTL`, zie paragraaf 12.3.1 van de MSP430x2xx Family User's Guide:

- Kies SMCLK als clock source. Dit kloksignaal is gelijk aan het DCOCLK signaal dat je hebt ingesteld op 1 MHz.

- Kies een input divider van /8. Hiermee wordt de klokfrequentie waarmee Timer_A telt gelijk aan $1/8 = 0{,}125\,\text{MHz}$. Een klokperiode van deze timer komt dus overeen met $1/0{,}125\,\text{MHz} = 8\,\mu\text{s}$. Als we dus $0{,}5\,\text{s} = 500\,000\,\mu\text{s}$ willen wachten dan moeten we Timer_A0 dus 500 000 / 8 = 62 500 kloktikken laten tellen. Timer_A0 is een 16-bit timer die tot maximaal $2^{16} - 1 = 65\,535$ kan tellen. Dus tellen tot 62 500 is geen probleem.
- Kies Continuous mode.
- De rest van alle bits in register `TA0CTL` moeten op nul worden gezet.

De code van de **`while`**`(1)`-loop van het programma is gegeven in listing 1. Je ziet dat in deze code het timer register van Timer_A0 op `0` wordt gezet. Vervolgens wordt gewacht tot dit register de waarde `62500` heeft bereikt (dit duurt als het goed is 0,5 s). Dan wordt de rode led geïnverteerd en begint de **`while`**-loop weer opnieuw. Als je de timer correct geconfigureerd hebt, dan laat dit programma de rode led knipperen met een frequentie van 1 Hz.

```
while (1)
{
    TA0R = 0;
    while (TA0R < 62500)
    {
        /* wacht */
    }
    P2OUT ^= 1<<0; // toggle rode led
}
```

Listing 1: De **`while`**`(1)`-loop bij opdracht 6.2.1.

```
TA0CTL = 0b0000001011100000;
//                ^^          10 = SMCLK
//                  ^^        11 = /8
//                    ^^      10 = Continuous mode
```

Listing 2: Een mogelijke manier om Timer_A0 te configureren.

Misschien heb je Timer_A0 wel geconfigureerd zoals gegeven in listing 2. Hoewel de programmeur van deze code geprobeerd heeft om in commentaar duidelijk te maken wat de betekenis is van de verschillende bits in dit register, is deze code toch niet erg goed leesbaar. In de standaard include file `msp430.h` zijn een flink aantal constante waarden gedefinieerd die je kunt gebruiken om de peripherals van de MSP430G2553 te configureren. Als je met

de cursor op de filenaam `msp430.h` gaat staan in CCS en je drukt op F3, dan wordt deze file geopend. Je ziet dat in deze file een specifieke include file wordt geopend afhankelijk van de gebruikte microcontroller. In ons geval is dat `msp430g2553.h`. Zoek deze filenaam op en ga er met de cursor op staan en druk nogmaals op F3. De file `msp430g2553.h` wordt nu geopend in de editor van CCS. Als je de inhoud van deze file bekijkt dan zie je dat er allerlei constanten zijn gedefinieerd die je in je programma's kunt gebruiken. De constanten `BIT0` tot en met `BITF` kun je bijvoorbeeld gebruiken als maskers. Dus in plaats van:

```
P2OUT ^= 1<<0;
```

Kun je dus ook de volgende code gebruiken:

```
P2OUT ^= BIT0;
```

Onder het kopje `Timer0_A3` vind je een aantal constanten die je kunt gebruiken om de timers te configureren. Zo kun je bijvoorbeeld de constante `MC_2` gebruiken om de Continious mode in te stellen.

6.2.2 Configureer nu het register `TA0CTL` met behulp van de in de includefile `msp430g2553.h` gedefinieerde constanten.

Als het goed is heb je Timer_A0 nu geconfigureerd zoals gegeven in listing 3.

```
TA0CTL = TASSEL_2 | ID_3 | MC_2;
// TASSEL_2 = Timer A clock source select: 2 - SMCLK
// ID_3 = Timer A input divider: 3 - /8
// MC_2 = Timer A mode control: 2 - Continous up
```

Listing 3: Een betere manier om Timer_A0 te configureren.

Het telkens opnieuw op `0` zetten van het register `TA0R` is *niet* nodig als je een andere mode van de timer kiest.

Lees nu paragraaf 12.2.3.1 van de MSP430x2xx Family User's Guide.

6.2.3 Pas het programma nu zo aan dat Timer_A0 in de Up mode werkt. Welke waarde moet je in het register `TA0CCR0` laden om er voor te zorgen dat de timer na elke halve seconde weer op `0` begint? Omdat de bit `TAIFG` in het register `TA0CTL` nu elke keer

geset wordt als de timer weer naar nul gaat, kun je deze flag[4] pollen om te kijken of er al een halve seconde is verstreken. De benodigde code voor de `while(1)`-loop is gegeven in listing 4

```
while (1)
{
    while ((TA0CTL & TAIFG) == 0)
    {
        /* wacht op TAIFG */
    }
    TA0CTL &= ~TAIFG; // reset TAIFG
    P2OUT ^= BIT0; // toggle rode led
}
```

Listing 4: De `while(1)`-loop bij opdracht 6.2.3.

Tot nu toe zijn we nog niet zoveel opgeschoten met het gebruik van de timer peripheral. De CPU moet nog steeds in de gaten houden of er al een halve seconde verlopen is. Maar we zijn al veel verder dan je misschien denkt. In plaats van de bit `TAIFG` te pollen kunnen we ook de timer een interrupt laten genereren als deze flag geset wordt. De CPU kan dan iets anders gaan doen (of gaan slapen als er niets anders te doen is). Op het moment dat er een halve seconde is verlopen zal de timer de interrupt genereren waardoor de CPU onderbroken (of wakker gemaakt) wordt en de ISR wordt uitgevoerd. Na afloop van de ISR gaat de CPU weer verder waar die mee bezig was.

6.2.4 Configureer de timer nu zo dat er een interrupt gegenereerd wordt als de flag `TAIFG` wordt geset. Schrijf ook een ISR met als vector `TIMER0_A1_VECTOR` die wordt aangeroepen als de flag `TAIFG` wordt geset. In deze ISR moet de rode led worden getoggled en moet de flag `TAIFG` gereset worden. De `while(1)`-loop in het hoofdprogramma is nu helemaal leeg.

Omdat we nu niets meer te doen hebben in de functie `main`, kunnen we de microcontroller in de slaapstand zetten in plaats van de `while(1)`-loop uit te voeren.

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

[4] Een bit die door een peripheral geset wordt om een bepaalde conditie mee aan te geven, wordt een vlag (Engels: flag) genoemd. Deze naam is afkomstig uit de scheepvaart. Daar worden zogenoemde seinvlaggen gehesen om bepaalde condities aan te geven, zie https://nl.wikipedia.org/wiki/Seinvlag.

6.2.5 Vervang de `while(1)`-loop in het programma van opdracht 6.2.4 met een aanroep naar een zo zuinig mogelijke low-power mode.

We kunnen de CPU nog meer werk uit handen nemen. De timer peripheral kan ook zelf pinnen aansturen.

Lees nu paragraaf 12.2.5 tot en met 12.2.5.2 van de MSP430x2xx Family User's Guide.

Als je Output Mode 4:Toggle gebruikt met register `TACCR0` dan wordt het OUT0-signaal exact op het juiste moment getoggled en hebben we helemaal geen interrupt meer nodig. De CPU kan dan continue slapen (of nuttig dingen doen). Het OUT0-signaal moet dan nog wel aan een bepaalde pin worden gekoppeld. Zoek in de datasheet van de MSP430G2553 op aan welke pinnen het OUT0-signaal gekoppeld kan worden[5]. Kies één van deze pinnen en sluit er een led op aan (via een weerstand). Zoek in de datasheet van de MSP430G2553 hoe je het OUT0-signaal (in de datasheet TA0.0 genoemd) koppelt aan de gekozen pin. Afhankelijk van je keuze, vind je deze informatie op pagina 43 of 49 van de datasheet.

6.2.6 Pas het programma uit opdracht 6.2.5 nu zo aan dat er geen interrupt meer gegenereerd wordt. Zorg ervoor dat de timer het OUT0-signaal aanstuurt door Output Mode 4: Toggle te gebruiken met register `TACCR0`. Dit doe je door het register `TA0CCTL0` in te stellen. Zie paragraaf 12.3.3 van de MSP430x2xx Family User's Guide. Zorg er ook voor dat het OUT0-signaal gekoppeld wordt met de juiste output pin (naar keuze P1.1 of P1.5). De CPU kan nu aan het einde van het programma gaan slapen en hoeft ook niet meer wakker gemaakt te worden. De Timer_A0 peripheral zal geheel zelfstandig de led laten knipperen.

Als we de Timer_A0 op een lagere klokfrequentie (ACLK) laten draaien in plaats van op SMCLK, dan kunnen we de microcontroller nog dieper laten slapen.

Je hebt al eerder gelezen dat je voor ACLK kunt kiezen uit LFXT1CLK of VLOCLK **Lees nu paragraaf 5.2.2 van de MSP430x2xx Family User's Guide.**

De interne Very-Low-Power Low-Frequency Oscillator (VLO) genereert dus een kloksignaal van 12 kHz. Dit signaal kunnen we selecteren als ACLK en vervolgens kunnen we dit signaal gebruiken als klokingang voor Timer_A0.

[5] Op pagina 6 en verder vind je dat je kunt kiezen voor P1.1 of P1.5.

6.2.7 Zorg ervoor dat de uitgang van de VLO gekoppeld wordt met ACLK. Gebruik ACLK als clock input voor Timer_A0. Welke waarde moet je nu in het register `TA0CCR0` laden om er voor te zorgen dat de timer na elke halve seconde weer op `0` begint? Eindig het programma met een aanroep naar een zo zuinig mogelijke low-power mode.

Verdieping

De volgende opdrachten geven je meer inzicht in het energieverbruik van de MSP430G2553.

6.2.8 Bereken hoeveel stroom je in opdracht 6.2.5 ongeveer bespaart door een zo zuinig mogelijke low-power mode te gebruiken. Vergeet niet om het stroomgebruik van de led ook mee te rekenen. Gebruik de EnergyTrace in CCS om je berekeningen te verifiëren.

6.2.9 Bereken hoeveel stroom je in opdracht 6.2.7 ongeveer bespaart door een zo zuinig mogelijke low-power mode te gebruiken. Vergeet niet om het stroomgebruik van de led ook mee te rekenen. Gebruik de EnergyTrace in CCS om je berekeningen te verifiëren.

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

6.2.10 Pas het codeslot dat je hebt gemaakt bij opdracht 6.1.11 aan zodat de benodigde vertragingstijden met behulp van een timer gerealiseerd worden. Maak gebruik van de interne Very-Low-Power Low-Frequency Oscillator (VLO) om er voor te zorgen 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.