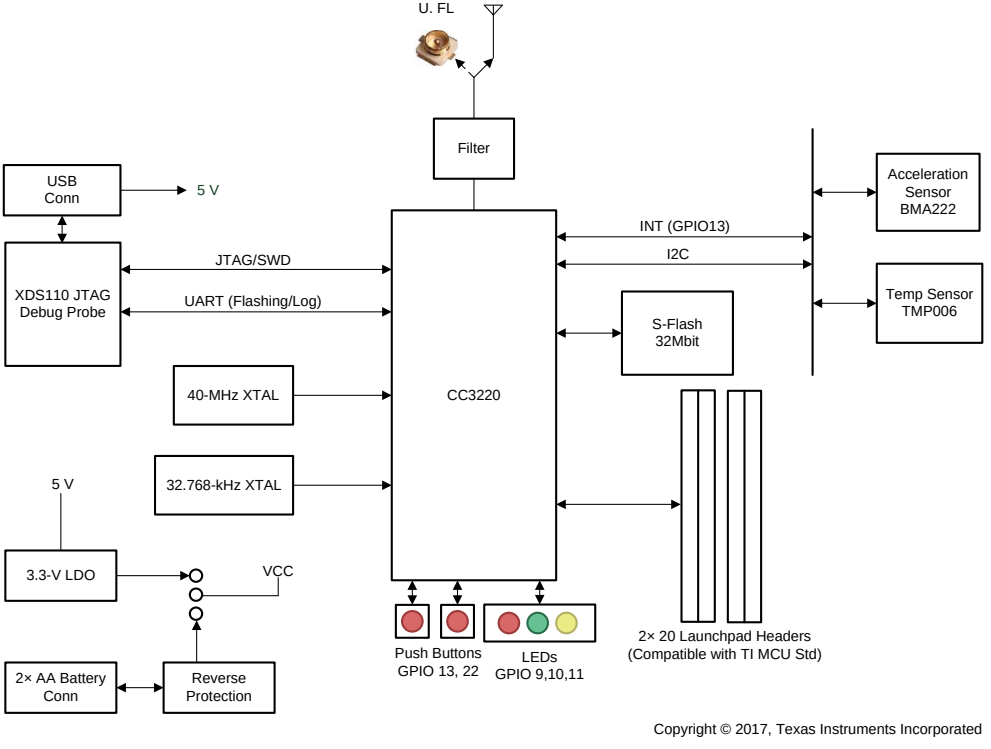
Opdrachten week 1 – Blinking leds

Bij dit vak wordt er weer gebruik gemaakt van de SimpleLink™ Wi-Fi® CC3220S LaunchPad™ die je (misschien) ook al bij het vak EMS20 hebt gebruikt. Je gaat deze les leren:

- uit welke onderdelen de CC3220S is opgebouwd;
- leds aan te sturen op verschillende abstractieniveaus;
- de System Configuration tool te gebruiken.



Figuur 1: Opbouw van de TI SimpleLink™ Wi-Fi® CC3220S LaunchPad™.

Het is tijd om iets meer te leren over de opbouw van de CC3220S[1]. De CC3220S is een zogenoemde SoC[2] die twee microcontrollers en een wifiradio bevat. Een van deze twee microcontrollers, een ARM® Cortex®-M4, is op de gebruikelijke wijze te programmeren. De andere microcontroller is een zogenoemde netwerk processor die alle wifi- en Internetprotocollen implementeert en de wifiradio aanstuurt. Deze laatste processor kan niet geprogrammeerd worden door de gebruiker. De verschillen tussen de CC3220S en de ATmega328P, die wordt gebruikt in de Arduino Uno, zijn weergegeven in tabel 1.

Tabel 1: Verschillen tussen de ATmega328P en CC3220S.

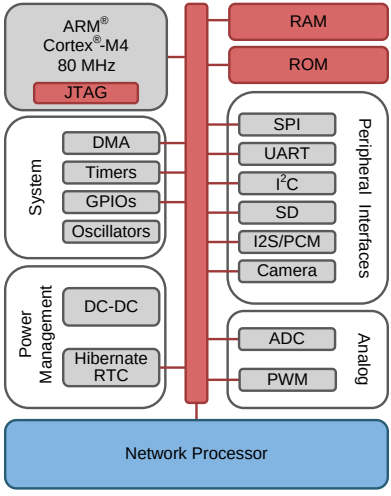
Onderdeel	ATmega328P	CC3220S
Architectuur CPU	8 bit ATmega	32 bit ARMv7
Snelheid CPU	16 MHz	80 MHz
Aantal µCs	1	2
Flash	32 kB	0[3]
RAM	2 kB	256 kB

De CC3220S bevat, zoals gezegd, twee processoren. Er is een processor die voor het gebruikersprogramma beschikbaar is (APP: Application Processor). Maar de CC3220S heeft ook een processor special voor wifi- en netwerkgerelateerde zaken (NWP: Network Processor), zie figuur 2. Deze opzet zorgt ervoor dat de APP zich niet druk hoeft te maken over wifi, TCP/IP, encryptie enzovoorts (figuur 3). Dit scheelt natuurlijk enorm veel overhead. De µC heeft nu tijd om de applicatie uit te voeren en tegelijkertijd veilig te communiceren met de buitenwereld via wifi.

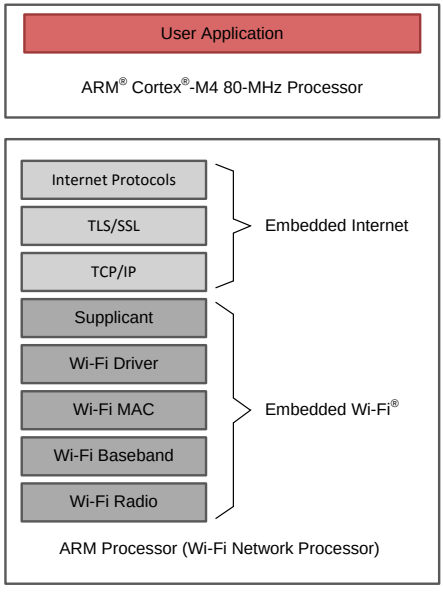
[1] Zie eventueel http://www.ti.com/product/CC3220S.

[2] Een SoC (System on Chip) is een chip waarop een heel systeem geïntegreerd is. Vaak is dit een combinatie van digitale en analoge elektronica.

[3] De CC3220S heeft geen intern flashgeheugen. Het LaunchPad bord bevat 4 MB (extern) flashgeheugen. Er is een variant van de CC3220S beschikbaar die wel intern flashgeheugen heeft. Deze CC3220SF bevat 1 MB flashgeheugen.



Figuur 2: Hardwarematige architectuur van de CC3220S.



Figuur 3: Softwarematige architectuur van de CC3220S.

Bij deze minor maken we geen gebruik van communicatie over wifi. Als je EMS20 niet hebt gevolgd en het leuk vind kun je, in je eigen tijd, kijken hoe communicatie over wifi kan worden opgezet[4]. In deze les gaan we weer terug naar de basis en kijken we hoe we de APP kunnen programmeren om een ledje te laten knipperen!

Je ziet in figuur 2 dat de CC3220S veel peripherals bevat die je misschien al kent. Hopelijk herken je de Timers, GPIOs[5], UART[6], I^2C[7] en ADC[8]. Hoewel de functionaliteit van deze peripherals vergelijkbaar is met die van hun naamgenoten in de andere microcontrollers is de implementatie anders. De I/O-registers van de CC3220S zijn anders van opbouw en naam dan die van andere microcontrollers.

1.1 We maken bij deze opdracht gebruik van Code Composer Studio (CCS) versie 10 en van de SimpleLink CC32XX SDK versie 5.20.00.06. Als je deze software nog niet hebt geïnstalleerd, dan kun je hier vinden hoe dat moet.

1.2 We gaan nu kijken hoe we een ledje kunnen laten knipperen. Als eerste doen we dit op registerniveau en vervolgens doen we dit nogmaals, maar dan maken we gebruik van de *TI Drivers*. Een nieuw project aanmaken via de menu optie Project › New CCS Project... wordt afgeraden door TI[9]. Door de docent is een project gemaakt dat geschikt is om op registerniveau te gaan programmeren. Clone of pull *de laatste versie van* de projectenrepository van `https://bitbucket.org/HR_ELEKTRO/ros01_ccs_projecten` en importeer `CC3220S_leeg_project` naar je workspace, via de menu optie Project › Import CCS Projects.... Als dit project al bestaat moet je deze (oude) versie eerst uit je workspace verwijderen.

[4] Zie: `https://bytebucket.org/HR_ELEKTRO/ems20/wiki/Opdrachten/Opdrachten_Week_5_Lab_1_Home.pdf`

[5] GPIO = General Purpose Input Output.

[6] UART = Universal Asynchronous Receiver Transmitter.

[7] I^2C = Inter-Integrated Circuit.

[8] ADC = Analog-to-Digital Converter.

[9] Zie paragraaf 5.4 van de Quick Start Guide for SimpleLink™ CC32xx SDK.

1.3 Build het project en kijk of je de software kunt debuggen op je CC3220S LaunchPad. Pas eventueel de compiler-versie aan naar degene die bij jouw installatie is meegeleverd. Sluit CCS af!

Om dadelijk de ledjes te kunnen controleren via het debug register view, moeten we de offset van het DATA register aanpassen, later zul je leren waarom.

1.4 Binnen de CCS installatiemap staan een hoop bestanden, we zullen het bestand `ccs\ccs_base\common\targetdb\Modules\cc32\gpio.xml` moeten aanpassen. Open dit met bijv. Notepad++. Het eerste register is het DATA register, deze heeft standaard een offset van 0x0. Pas deze offset aan naar 0x38 en sla het bestand op.

De programmeeromgeving is nu opgezet, debuggen is mogelijk en de registers zijn te bekijken en aan te passen. Wat nu rest is daadwerkelijk code schrijven om de registers aan te passen en een led te laten knipperen.

1.5 Voordat we code gaan schrijven gaan we eerst wat bitjes aanpassen in de registers. Start CCS opnieuw en start het project `CC3220S_leeg_project` in de debugger. Open het register view en probeer register `GPIOA0` te bekijken. Afhankelijk van de firmware die je op het apparaat hebt geladen, zal dit wel of niet lukken. Bekijk nu het `GPIOA1`-register, deze is wel te lezen!

Binnen de CC3220S kun je individuele peripherals aan- en uitschakelen om op deze manier de µC zo zuinig mogelijk in te stellen. Als we module 0 niet nodig hebben hoeft hij ook geen stroom te verbruiken! GPIOA1 is ingeschakeld door het standaard programma van de LaunchPad maar GPIOA0 niet. Om deze module ook in te schakelen moet je er een kloksignaal naar toe sturen wat je kunt doen door een 1 te schrijven naar het `GPIO_A_CLK_GATING`-register binnen de ARCM[10]-module. Om dit via de debugger te doen kun je

[10] ARCM = Application Reset-Clock Manager

klikken op de waarde van het `GPIO_A_CLK_GATING`-register, er een 1 neerzetten en op enter drukken. Laat de debugger een stapje maken (F5) om de nieuwe instelling uit te schrijven/lezen. Kun je nu `GPIOA0` wel bekijken?

Elke peripheral binnen de CC3220S moet op deze manier worden ingeschakeld voordat je de registers ervan kunt benaderen. Een sequentiële digitale schakeling functioneert immers niet zonder een kloksignaal!

1.6 Om nu een led aan te zetten, moeten we wel weten welke GPIO-module we moeten hebben! Welke naam heeft de rode led volgens figuur 1?

Hoofdstuk 5 van de Technical Reference Manual beschrijft hoe de GPIO-modules in elkaar zitten. Als er 32 GPIO-pinnen zijn en 4 modules van 8-bit breed. Welke module valt de rode led dan onder en om welke bit gaat het?

Test jouw bevinding door `DATA` en `DIR` van de juiste module aan te passen in de register view. Pas eventueel het "number format" aan van de Value binnen de Register view, om het register binair weer te geven en de aanpassing makkelijker te maken. Na het aanzetten van de bit, zou de led direct aan moeten gaan op het bord. Als je de nieuwe inhoud van het register wilt zien, moet je de debugger (soms) een stapje laten maken (F5).

1.7 Zet ook de groene en de gele led uit door het `GPIO_DATA` en het `GPIO_DIR`-register aan te passen in de register view. Als dit niet lukt voer dan opdracht 1.8 uit om de pinnen aan de GPIO-module te koppelen. Als dit wel is gelukt hoef je opdracht 1.8 alleen maar door te lezen, zodat je snapt hoe de pinnen aan de GPIO-module gekoppeld zijn.

1.8 Afhankelijk van de firmware die je op het apparaat hebt geladen, zijn GPIO10 en GPIO11 al dan niet aan de GPIO-module gekoppeld. Deze leds zijn dan gekoppeld aan de I^2C-peripheral. Dit is aan te passen met behulp van de Pin Multiplexer!

In de register view zijn de multiplexer configuratieregisters GPIO_PAD_CONFIG_X genoemd onder OCP_SHARED.

Wanneer je hier kijkt in de register view, dan zie dat de registers niet leesbaar zijn. Om deze leesbaar te krijgen kunnen we het volgende doen. Als je de debugger hebt gestart, ga naar Tools > GEL Files. Een nieuw venster opent. Binnen dit venster vind je "GEL Files". Dubbelklik op CC3220.gel en voeg het volgende toe in memorymap_init():

```
GEL_MapAddStr(0x4402E0A0, 0, 0x00000120, "R|W", 0); /* OCP */
```

Sla het bestand op, en sluit CCS! Start CCS weer op en debug opnieuw je project. Nu kun je het register wel benaderen.

Je kunt de registerwaarde van pin 9 (GPIO_PAD_CONFIG_9) overnemen voor pin 10 en 11 zodat ook die gekoppeld zijn aan de GPIO-module. In paragraaf 16.8.1.1.1 van de CC3220 Technical Reference Manual is eventueel te lezen wat de waarde van het register betekent.

Koppel de leds aan de GPIO-module en test wederom de werking door het GPIO_DATA en het GPIO_DIR-register aan te passen in de register view.

De in de headerbestanden aanwezige definities waarmee je de registers van de CC3220S vanuit een C-programma kunt benaderen moet je met behulp van de macro HWREG(x) gebruiken. Als je deze macro gebruikt, dan wordt x vervangen door `(*((volatile uint32_t *)(x)))`[11]. Op deze manier kun je geheugenadressen benaderen als zijnde een variabele (of eigenlijk derefereer je een pointer naar het gegeven adres). Deze manier van werken stelt je in staat om dezelfde registerdefinitie te gebruiken voor verschillende basis-adressen (modules). Dus met bijvoorbeeld HWREG(GPIOA1_BASE + GPIO_O_GPIO_DIR) benader je het DIR-register van GPIO-module 1, en met HWREG(GPIOA0_BASE + GPIO_O_GPIO_DIR) benader je het DIR-register van GPIO-module 0.

[11] Hier wordt een adres gecast naar een pointer naar een unsigned 32 integer. Deze pointer wordt vervolgens gederefereerd zodat het register achter het adres aangepast kan worden.

De registernamen zijn te vinden in de Project Explorer onder het mapje `inc`. Je kunt een header file openen door te dubbelklikken op de filenaam in de Project Explorer. Je maakt bij de volgende opdracht gebruik van de volgende header files:

- In `inc/hw_memmap.h` zijn alle definities voor de basis-adressen te vinden.
- In `inc/hw_gpio.h` zijn alle definities voor de GPIO-module te vinden.
- In `inc/hw_apps_rcm.h` zijn alle definities voor de ARCM-modules te vinden.
- In `inc/hw_ocp_shared.h` zijn alle definities voor de pin multiplexer te vinden.

1.9 Maak gebruik van `HWREG()` en de eerder genoemde headerbestanden om in code alles te doen wat bij opdracht 1.5 t/m opdracht 1.8 is gedaan (dat wordt hieronder stap voor stap uitgelegd).

Je programma moet achtereenvolgens de volgende drie stappen uitvoeren:

- De klok van module GPIOA1 moet enabled worden (zoals je handmatig in opdracht 1.5 hebt gedaan). Het basisadres van de ARCM-module kun je vinden in de header file `inc/hw_memmap.h`. De offset naar het juiste register kun je vinden in de header file `inc/hw_apps_rcm.h`. De offset naar register GPIO1CLKEN heet in deze header file vreemd genoeg `APPS_RCM_O_GPIO_B_CLK_GATING`. De verschillende GPIO-modules worden in de header file aangeduid met `A` t/m `D` in plaats van met `0` t/m `3`.
- De pin multiplexer moet goed ingesteld worden voor alle drie de pinnen waarop de leds zijn aangesloten (zoals je handmatig in opdracht 1.8 hebt gedaan). Het basisadres van de GPIO-multiplexer-module wordt in de header file `inc\hw_memmap.h` `OCP_SHARED_BASE` genoemd. De offsets naar het juiste registers kun je vinden in de header file `inc/hw_ocp_shared.h`.
- Tot slot moeten de `GPIO_DIR` en `GPIO_DATA` van de juiste GPIO-module de juiste waarden krijgen om de leds aan te schakelen (zoals je handmatig in opdracht 1.6 hebt gedaan). De GPIO-module maakt voor het DATA register gebruik van een adresmasker om te bepalen welke bits

mogen worden aangepast. De `GPIO_O_GPIO_DATA` definitie heeft dit masker op 0 staan (en dat is niet de juiste waarde om de leds aan te kunnen sturen). Lees paragraaf 5.2.1.2 om te kijken welke waarden bit 2-9 van het adres moeten hebben. Bedenk dat `HWREG()` een adres verwacht. Je kunt het masker simpelweg erbij optellen: `HWREG(adres + (masker<<2))`.

Vergeet niet om het juiste masker te gebruiken als je schrijft naar het register `GPIO_DATA`, zoals hierboven uitgelegd. Het basisadres van de juiste GPIO-module kun je vinden in de header file `inc/hw_memmap.h`. De offsets naar het juiste registers kun je vinden in de header file `inc/hw_gpio.h`.

Zorg ervoor dat de leds worden aangestuurd volgens tabel 2. Elke regel uit de tabel moet ongeveer een seconde duren. Als het einde van de tabel bereikt is, moet weer bovenaan worden begonnen. Gebruik een **`for`**-loop (busy waiting) om de vertraging te realiseren. Het is dus *niet* de bedoeling dat je de hardware timers en/of interrupts gebruikt. We stellen dat nog even uit tot volgende week.

Tabel 2: Gewenste sequentie waarmee de leds aangestuurd moeten worden.

groen	geel	rood
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Zoals je waarschijnlijk wel merkt is het programmeren van de CC3220S op registerniveau vrij complex. Om die reden gaan we gebruik maken van de *TI Drivers* die veel eenvoudiger te gebruiken zijn. Deze bibliotheek is gebouwd bovenop de *driverlib* en voegt nog een abstractielaag toe. De documentatie van deze bibliotheek kun je vinden op: https://dev.ti.com/tirex/explore/content/simplelink_cc32xx_sdk_5_20_00_06/docs/drivers/doxygen/html/index.html.

1.10 In de laatste versies van Code Composer Studio heeft TI een zogenoemde Systeem Configuratie tool ingebouwd. Hiermee kun je gemakkelijk de TI Drivers en betreffende pinnen configureren. Gelukkig voorziet TI in een goede tutorial hiervoor: https://dev.ti.com/tirex/explore/node?node=AO3dBAhu05HBWXd.SPxt0g__fc2e6sr__LATEST. Voer deze tutorial uit **tot en met Task 7**. Maak daarbij gebruik van het project dat je kunt vinden in de SDK in `..\examples\rtos\CC3220S_LAUNCHXL\drivers\empty\ti-rtos\ccs`.

1.11 Gebruik je opgedane kennis van de tutorial om de leds weer aan te sturen zoals gegeven in tabel 2 maar nu door gebruik te maken van de Systeem Configuratie tool en de TI Drivers. Begin daarbij weer met het lege project (empty). Je moet dan het bestaande project `empty_CC3220S_LAUNCHXL_tirtos_ccs` eerst hernoemen of verwijderen. Elke regel uit de tabel moet ongeveer een seconde duren. Als het einde van de tabel bereikt is, moet weer bovenaan worden begonnen.

De GPIO-driver is bedoeld om digitale I/O-pinnen mee aan te sturen of uit te lezen. Specifiek voor leds en buttons zijn er gespecialiseerde drivers beschikbaar. Zie LED driver en Button driver

1.12 Schrijf een programma met behulp van de LED en Button drivers waarin de leds aangestuurd worden volgens tabel 2. In dit programma wordt gebruik

gemaakt van twee knoppen, die hier knop 1 en knop 2 genoemd worden. Als het programma begint zijn alle leds uit.

Met knop 1 bepalen we de volgende stap:

- bij een klik worden de leds aangestuurd volgens de *volgende* regel in de tabel;
- bij een dubbele klik worden de leds aangestuurd volgens de *vorige* regel in de tabel;
- als de knop langer dan 2 seconden wordt ingedrukt, worden de leds aangestuurd volgens de *eerste* regel in de tabel (alle leds uit).

Met knop 2 bepalen we de helderheid van de leds in 4 stappen, van zwak naar helder:

- bij een klik gaan de leds een stapje *minder* helder branden;
- bij een dubbele klik gaan de leds een stapje *helderder* branden;
- als de knop langer dan 2 seconden wordt ingedrukt, worden de leds met de maximale helderheid aangestuurd.