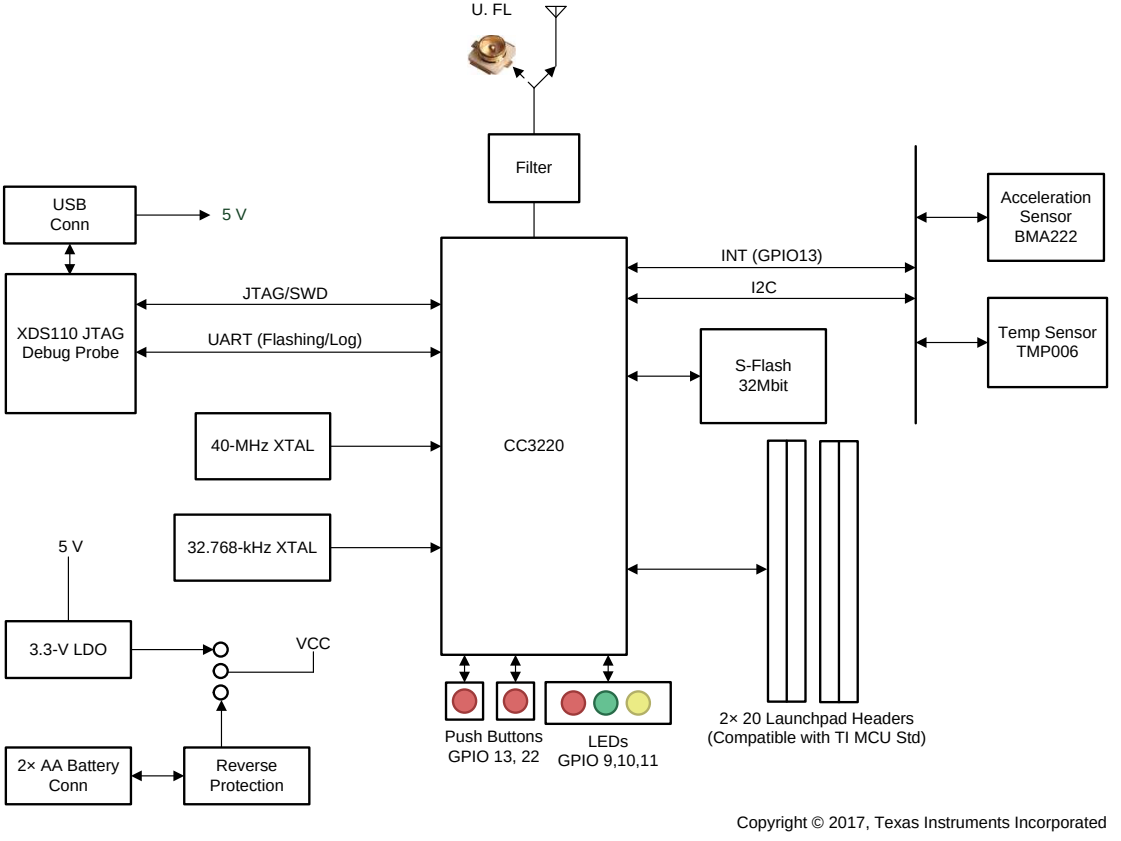
Opdrachten week 3 lab 1 – GPIO (General Purpose Input Output) en UART-driver

Je kennis van de programmeertaal C is de afgelopen weken flink opgefrist en uitgebreid. Je hebt ook geleerd hoe een microcontroller werkt op een lager niveau.

Bij dit lab wordt er weer gebruik gemaakt van de SimpleLink™ Wi-Fi® CC3220S LaunchPad™. Je gaat leren:

- uit welke onderdelen de CC3220S is opgebouwd;
- leds aan te sturen op verschillende abstractieniveaus;
- de UART-driver te gebruiken;
- 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 MSP430 zijn weergegeven in tabel 1.

Tabel 1: Verschillen tussen de MSP430 en CC3220S.

Onderdeel	MSP430	CC3220S
Architectuur CPU	16 bit MSP	32 bit ARMv7
Snelheid CPU	16 MHz	80 MHz
Aantal µCs	1	2
Flash	16 kB	0[3]
RAM	512 B	256 kB

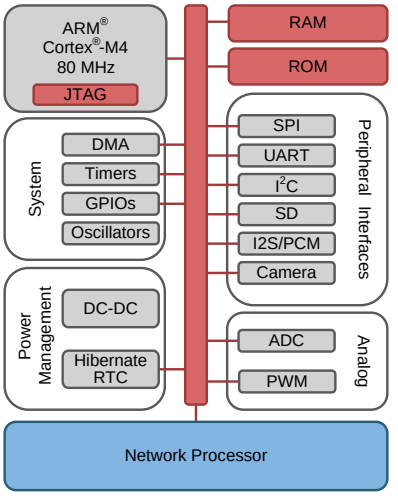
De CC3220S bevat, zoals gezegd, twee processoren. Er is een processor die voor het gebruikersprogramma beschikbaar is (APP: Application Processor). Maar de CC3220S heeft ook een processor special voor wifi- en netwerkgerelateerde zaken (NWP: Network Processor), zie figuur 2. Deze opzet zorgt ervoor dat 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.

In latere labs wordt gekeken hoe deze communicatie over wifi kan worden opgezet. In dit lab gaan we weer terug naar de basis en kijken we hoe we de APP kunnen programmeren om een ledje aan te sturen!

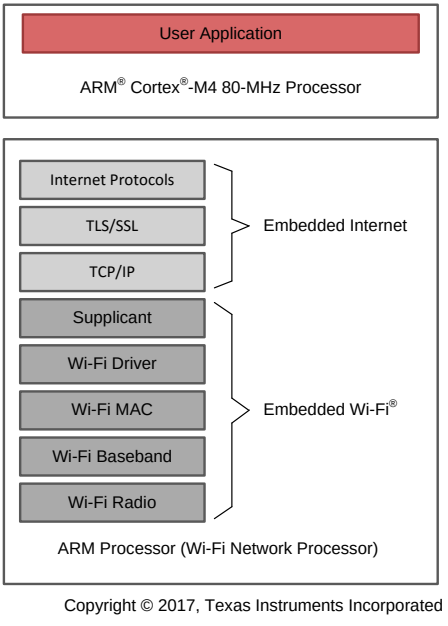
[1] Zie eventueel `http://www.ti.com/product/CC3220S`.

[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.

Je ziet in figuur 2 dat de CC3220S veel peripherals bevat die we in EMS10 hebben leren kennen bij het programmeren van de MSP430. Hopelijk herken je de Timers, GPIOs[4], UART[5],

[4] GPIO = General Purpose Input Output.

I^{2}C[6] en ADC[7]. Hoewel de functionaliteit van deze peripherals vergelijkbaar is met die van hun naamgenoten in de MSP430 is de implementatie volstrekt anders. De I/O-registers van de CC3220S zijn volledig anders van opbouw en naam dan die van de MSP430.

leds aansturen met registers via debug-view

3.1.1 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 menuoptie Project > New CCS Project... wordt afgeraden door TI[8]. 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/ems20_ccs_projecten en importeer `CC3220S_leeg_project` naar je workspace, via de menu optie Project > Import CCS Projects.... Mocht het project al eerder aan je workspace zijn toegevoegd, dan kun je die versie gebruiken.

3.1.2 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 Register view window van de debugger, moeten we de offset van het DATA-register aanpassen, later zul je leren waarom.

3.1.3 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.

---

[5] UART = Universal Asynchronous Receiver Transmitter.

[6] I^{2}C = Inter-Integrated Circuit.

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

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

3.1.4 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. Dit lukt niet[9]! 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 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!

3.1.5 Om nu de rode led aan te zetten, moeten we wel weten welke GPIO-module we moeten hebben!

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?[11]

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 juiste bits, zou de rode 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).

[9] Als je inderdaad het NetworkTerminal project op de flash hebt geschreven

[10] ARCM = Application Reset-Clock Manager

[11] Inderdaad, bit 1 (niet 0!) in GPIO module 1

3.1.6 Stuur nu de groene en de gele led aan door het `GPIO_DATA`- en het `GPIO_DIR`-register aan te passen in de Register view.

leds aansturen met registers via code (optioneel[12])

Ga naar verder met opdracht 3.1.8 als je deze opdracht niet doet.

De in de headerbestanden aanwezige definities waarmee je de registers van de CC3220S vanuit een C-programma kunt benaderen moet je op een iets andere wijze gebruiken dan bij de MSP430. Waar je bij de MSP430 bijvoorbeeld `P1DIR` zou neerzetten, zou je nu `HWREG(P1DIR)` moeten gebruiken. `HWREG(x)` is een zogenoemde macro die `x` vervangt door `(*((volatile uint32_t *)(x)))`[13]. 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.

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.

3.1.7 Maak gebruik van `HWREG()` en de eerder genoemde headerbestanden om in code alles te doen wat bij opdracht 3.1.4 t/m opdracht 3.1.6 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 3.1.4 hebt gedaan). Het basisadres van de ARCM-module kun je vinden

[12] Deze opdrachten hoe je niet te maken, maar zijn leerzaam als beter wilt begrijpen hoe dit koppelt aan de manier die je voor de MSP430 hebt geleerd.

[13] 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.

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`.

- 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 3.1.5 en opdracht 3.1.6 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 er simpelweg bij 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`. Als je niet uitkomt, zie footnote [14].

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.

leds aansturen met de TI-Drivers abstractielaag

Zoals je waarschijnlijk wel merkt is het programmeren van de CC3220S andere koek dan het programmeren van de MSP430. De registers zijn groter, de peripherals zijn complexer en de support om op registerniveau te programmeren is minder.

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: http://dev.ti.com/tirex/explore/content/simplelink_cc32xx_sdk_6_10_00_05/docs/drivers/doxygen/html/index.html.

[14] Als je er niet uit komt kun je HWREG(GPIOA1_BASE + GPIO_O_GPIO_DATA + (0xE≪2)) gebruiken

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

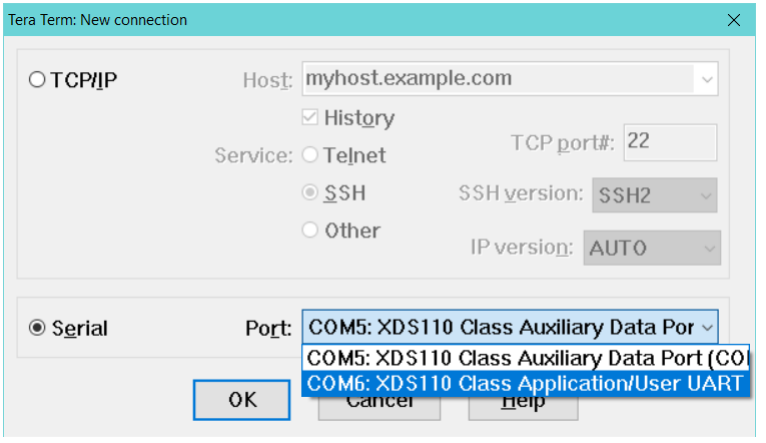
3.1.8 Om gebruik te maken van de TI Drivers is het makkelijker om een bestaand project te importeren, dan het huidige project aan te passen. Als eerste gaan we een project aanmaken dat geschikt is om met behulp van de TI Drivers te gaan programmeren.

- Kies Project > Import CCS Projects... en klik op Browse. Navigeer naar het directory: `C:\ti\simplelink_cc32xx_sdk_6_10_00_05\examples\nortos\CC3220S_LAUNCHXL\drivers\uart2echo\ccs`. Klik op Finish.
- Klik met de rechtermuisknop op het projectnaam en kies *Rename...*. Hernoem het project naar `leds_CC3220S_LAUNCHXL_nortos_ccs`.
- Hernoem `uart2echo.c` (op dezelfde wijze) naar `leds.c`.
- Hernoem `uart2echo.syscfg` naar `leds.syscfg`.
- Kies Project > Properties en selecteer (indien nodig) CCS General. Klik op de knop Manage Configurations... en maak de Debug configuratie Active. Delete vervolgens de MCU+Image configuratie[15]. Klik tot slot op OK en op Aply and Close.
- Verwijder `main_nortos.c`, `Board.html`, `image.syscfg`, `MCU+Image`, `userFiles`, `README.html` en `README.md` door op elk bestand te klikken met de rechtermuisknop en te kiezen voor *Delete*.
- Vervang de code in `leds.c` door de code die gegeven is in listing 1. Je kunt deze code eenvoudig kopiëren met behulp van deze link: `leds.c`. Dit programma

[15] Deze configuratie is nodig als we het programma in het Flash geheugen willen laden. In dit geval willen we het programma alleen in RAM laden om het te kunnen debuggen (in de Debug configuratie).

initialiseert een UART (Universal Asynchronous Receiver-Transmitter) zodat we karakters kunnen versturen naar en ontvangen van de pc via een seriële verbinding. Zie ems10 week 8 les 1 als je niet meer weet wat een UART is. Het programma stuurt elk ontvangen karakter weer terug en toggled de rode led telkens als een karakter wordt ontvangen.

- Start het terminalprogramma Tera Term[16] en maak verbinding met de virtuele COM-poort *XDS110 Class Application/User UART*, zie figuur 4.
- Debug en start het CCS-project `leds_CC3220S_LAUNCHXL_nortos_ccs` en test de werking door iets in te typen in het terminalprogramma Tera Term.



Figuur 4: Open een seriële verbinding met Tera Term.

We bekijken het programma gegeven in listing 1 nu wat beter. Op regel 20 tot en met 27 wordt de UART ingesteld. We maken daarbij geen gebruik van registers maar van een zogenoemde *driver*. Je zou de UART ook via registers in kunnen stellen, maar dat is veel meer uitzoekwerk. Op regel 20 wordt de variabele `uartParams` aangemaakt van het type `UART2_Params`. In de documentatie van de UART2-driver[17] kun je vinden dat dit een **`struct`**[18] is die verschillende datavelden (Engels: data fields) bevat. Op regel 21 geven we het *adres van*[19] deze variabele door aan de functie `UART2_Params_init`. Deze functie vult de **`struct`**

[16] Je kunt het programma downloaden van `https://osdn.net/projects/ttssh2/releases/`. De installatie wijst zichzelf.

[17] Zie: `http://dev.ti.com/tirex/explore/content/simplelink_cc32xx_sdk_6_10_00_05/docs/drivers/doxygen/html/_u_a_r_t2_8h.html`.

[18] Zie Week 2 Lab 1.

[19] Zie Week 1 Lab 2.

```c
/*
 * Copyright (c) 2024, Rotterdam University of Applied Sciences
 * All rights reserved.
 */

#include <stdint.h>
#include <string.h>
// Driver Header files
#include <NoRTOS.h>
#include <ti/drivers/GPIO.h>
#include <ti/drivers/UART2.h>
// Driver configuration
#include "ti_drivers_config.h"

int main(void) {
    Board_init();
    NoRTOS_start();
    // Turn off red LED
    GPIO_write(CONFIG_GPIO_LED_0, CONFIG_GPIO_LED_OFF);
    // Init UART
    UART2_Params uartParams;
    UART2_Params_init(&uartParams);
    uartParams.baudRate = 9600;
    UART2_Handle uart = UART2_open(CONFIG_UART2_0, &uartParams);
    if (uart == NULL) {
        while (1); // UART2_open() failed
    }

    const char echoPrompt[] = "Echoing characters:\r\n";
    size_t bytesWritten;
    UART2_write(uart, echoPrompt, strlen(echoPrompt), &bytesWritten);

    while (1) { // Loop forever echoing
        char input;
        size_t bytesRead;
        UART2_read(uart, &input, 1, &bytesRead);
        UART2_write(uart, &input, 1, &bytesWritten);
        GPIO_toggle(CONFIG_GPIO_LED_0);
    }
}
```

Listing 1: Echoput die de rode led laat knipperen, zie leds.c

met default waarden. Het enige veld van de **struct** dat, in dit geval, aangepast hoeft te worden is `uartParams.baudRate`. Dit gebeurt op regel 22. Op regel 23 wordt de UART-verbinding geopend. De `UART2_Handle` die opgeslagen wordt in de variabele `uart` kunnen we later gebruiken om karakters via de UART te verzenden en te ontvangen, zie regel 30, 35 en 36. Als het openen niet lukt, dan blijft het programma 'hangen' op regel 25.

De led wordt aangestuurd op regel 18 en 37. Wat opvalt is dat de GPIO-pin waarop de led is aangesloten en de UART-pinnen niet geconfigureerd worden in het programma. Maar hoe weten deze functies welke pinnen gebruikt moeten worden en hoe die geconfigureerd moeten worden?

Deze informatie is te vinden in de files `ti_driver_config.h` en `ti_driver_config.c`. In Code Composer Studio is een zogenoemde Systeem Configuratie tool beschikbaar waarmee deze files aangemaakt kunnen worden, zie figuur 5. Hiermee kun je gemakkelijk de TI Drivers en betreffende pinnen configureren.

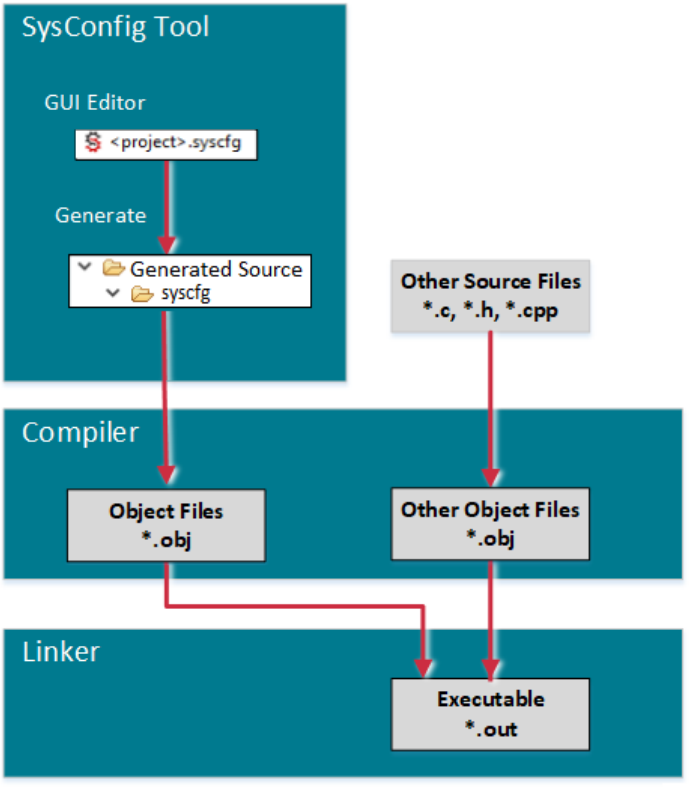
Als je in de *Project Explorer* op `leds.syscfg` dubbelklikt, wordt de Systeem Configuratie tool geopend, zie figuur 6.

Je ziet dat de naam `CONFIG_GPIO_LED_0` die we in listing 1 gebruikt hebben om de led aan te sturen hier gedefinieerd is. Deze naam is gekoppeld aan de pin waarop de rode led is aangesloten op de CC3220S LaunchPad. Je ziet dat het met behulp van deze tool heel eenvoudig is om een pin te configureren. Zo kun je bijvoorbeeld bij *Mode* kiezen voor *Input*, *Output* of *Dynamic*. *Dynamic* wil zeggen dat het in het programma zelf wordt geconfigureerd met behulp van de functie `GPIO_setConfig()`.

Door op het knopje *Hardware* (aan de linkerkant) te klikken kun je alle hardware die op de CC3220S LaunchPad aanwezig is eenvoudig configureren, zie figuur 7.

Door op het knopje *Show Board View* (rechts boven) te klikken kun je zien welke pinnen geconfigureerd zijn om met de drivers te gebruiken, zie figuur 8.

3.1.9 Gebruik de SysConfig tool om de pinnen zo te configureren dat de gele en groene led op de CC3220S Launchpad ook met de GPIO-driver aan te sturen zijn. Zorg ervoor dat de leds worden aangestuurd volgens tabel 2. Telkens als een karakter binnenkomt via de UART moeten de leds aangestuurd worden zoals aangegeven in de volgende regel uit de tabel. Als het einde van de tabel bereikt is, moet weer bovenaan worden begonnen.



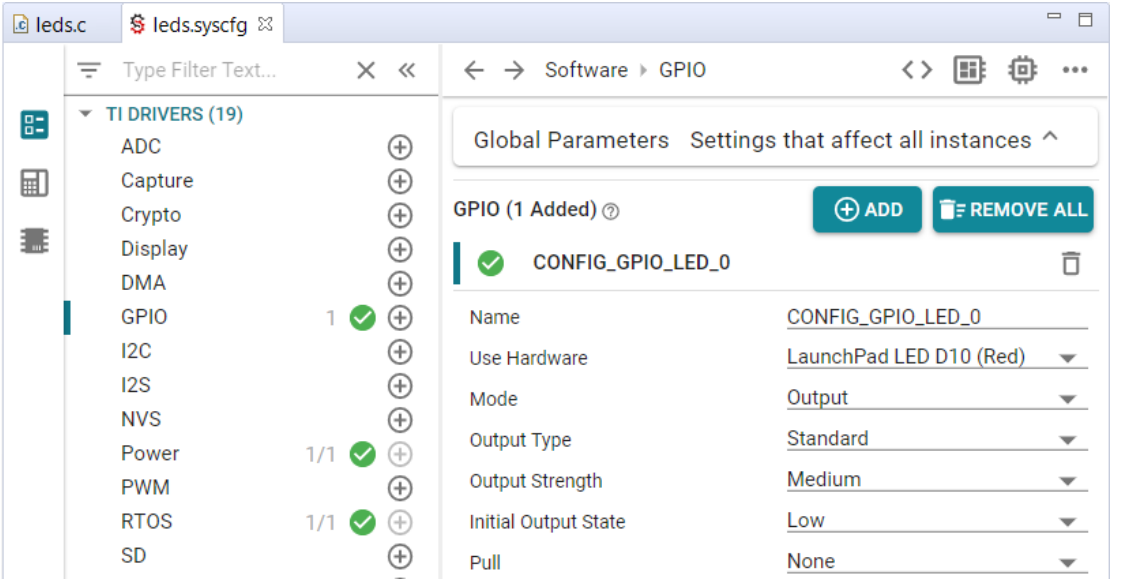
Figuur 5: De SysConfig tool genereert de bestanden die door de drivers gebruikt worden om de microcontroller te configureren.

In het volgend lab leer je hoe je de TI Drivers kunt gebruiken om via I^2C de sensoren op het bordje uit te lezen.

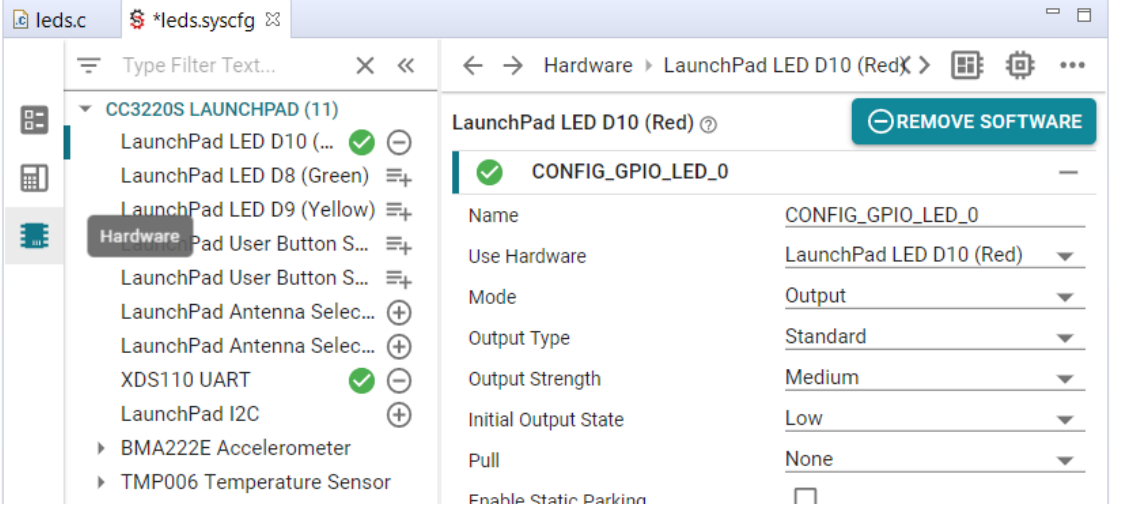
De volgende opgave behoort niet tot de verplichte stof voor EMS20 en is bedoeld voor studenten die zich verder in het programmeren van de leds en knoppen op de CC3220S LaunchPad willen verdiepen.

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

3.1.10 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



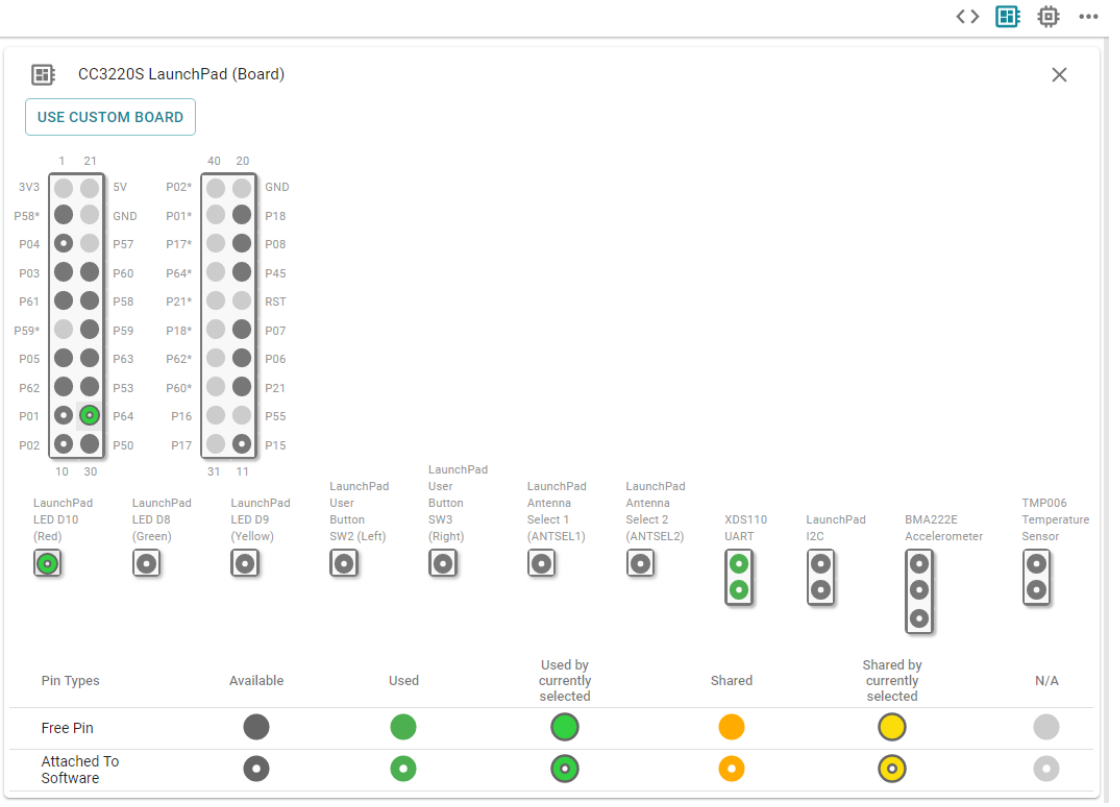
Figuur 6: De SysConfig tool.



Figuur 7: De standaard op de CC3220S Launchpad aanwezig hardware.

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;



Figuur 8: Een overzicht van de geconfigureerde pinnen op de CC3220S Launchpad.

- 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.