

Inleiding

Vorige week heb je de user leds van het STM32F411E-DISCO ontwikkelbord aangestuurd met behulp van Pinky assembly code. Deze week ga je deze leds aansturen vanuit de hogere programmeertaal C. Je gaat dit doen op verschillende abstractieniveaus:

- zonder gebruik te maken van een API (Application Programmers Interface);
- met gebruik van de CMSIS (Common Microcontroller Software Interface Standard) API van Arm;
- met gebruik van de LL (Low-Layer) API van STM;
- met gebruik van de HAL (Hardware Abstraction Layer) API van STM.

Ook ga je met behulp van deze APIs en de STM32CubeMX configuratietool van STM de klokfrequentie van de STM32F411VET6 microcontroller aanpassen.

Aan het einde van deze week moet je een kort [verslag](#) van deze opdracht inleveren in .pdf-formaat in een daarvoor aangemaakte opdracht in Brightspace. In dit verslag bespreek je de door jouw geschreven code en beantwoord je de in de deelopdrachten gestelde vragen. Daarnaast maak je deze week ook de [assembly assignment](#).

Opdrachten week 2 – Microcontrollerarchitectuur en programmeren in C

Je gaat deze week leren hoe je:

- de user leds van het STM32F411E-DISCO ontwikkelbord met behulp van de programmeertaal C kunt aansturen:
 - zonder gebruik van een API;
 - met behulp van de CMSIS API van Arm;
 - met behulp van de STM32CubeMX configuratietool en de LL en HAL APIs van STM.
- “bit-banding” met behulp van de programmeertaal C kunt gebruiken om een enkele bit in de IO-registers van de STM32F411VET6 microcontroller te benaderen.
- het keyword **volatile** gebruikt om er voor te zorgen dat een programma ook na optimalisatie nog correct werkt;
- de klokfrequentie van de STM32F411VET6 microcontroller kunt aanpassen met behulp van:

- de CMSIS API van Arm;
- de STM32CubeMX configuratietool van STM.
- de Adaptive Real-Time (ART) memory accelerator van de STM32F411VET6 microcontroller kunt gebruiken om het flashgeheugen zonder wait states uit te lezen;
- een vertraging implementeert die onafhankelijk is van de klokfrequentie en de optimalisatie-instellingen met behulp van de LL en HAL APIs van STM.

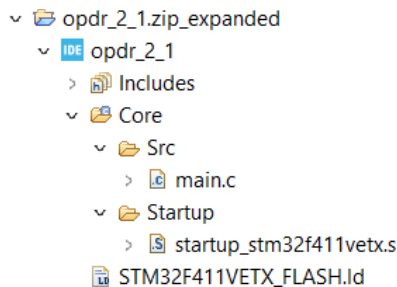
Leds aansturen in C zonder API

Om goed te begrijpen hoe je de IO-registers van een microcontroller kunt configureren vanuit C begin je met een C programma dat *geen* gebruikt maakt van een API.

2.1 A Download het project [opdr_2_1.zip](#).

B Importeer dit project in STM32CubeIDE met de menu-optie **File** > **Import...**, **General** > **Projects from Folder or Archive**, **Next**, **Archive**, selecteer het bestand `opdr_2_1.zip`, **Open** en **Finish**.

C Open in de “Project Explorer” het mapje `opdr_2_1.zip_expanded` ▶ `Opdr_2_1` ▶ `Core` ▶ `Src` en `Core` ▶ `Startup`, zie [figuur 1](#).



Figuur 1: De bestanden in het project `opdr_2_1.zip`.

D Het mapje `Includes` wordt automatisch aangemaakt bij het importeren en kun je negeren. Je ziet dat dit project verder slechts drie bestanden bevat.

Open het bestand `STM32F411VETX_FLASH.ld`. Dit bestand bevat het linkerscript dat er voor zorgt dat de GNU linker (`ld`)¹ alle code in het flashgeheugen en alle

¹ <https://sourceware.org/binutils/docs/ld.pdf>

data in het RAM-geheugen plaatst. Je ziet dat het linkerscript uitgebreider is dan het linkerscript dat je voor een assembler project hebt gebruikt. Kijk bijvoorbeeld nog even naar het linkerscript van het project `opdr_1_5`. Het linkerscript voor een C programma is uitgebreider omdat de C compiler meerdere sections gebruikt. Onder andere:

- `.text`
in deze section worden de machinecode-instructies opgeslagen;
- `.data`
in deze section worden de globale en **static** variabelen² die geïnitieerd worden met een waarde opgeslagen;
- `.bss`
in deze section worden de globale en **static** variabelen die *niet* geïnitieerd worden opgeslagen³;
- `.rodata`
in deze section worden constanten en string literals opgeslagen.

Dit linkerscript is, zoals je in het commentaar kunt lezen, automatisch aangemaakt door de STM32CubeIDE.

Open het bestand `startup_stm32f411vetx.s`. Dit assembly code bestand bevat de startup code die wordt uitgevoerd, na een reset, voordat de functie `main` wordt aangeroepen. Deze code voert onder andere de volgende acties uit:

- de stackpointer wordt geïnitieerd en wijst net voorbij het einde van het RAM-geheugen⁴;
- de data section wordt vanuit het flashgeheugen gekopieerd naar het RAM-geheugen zodat de globale en **static** variabelen na een reset opnieuw geïnitieerd worden⁵;
- de bss section wordt gevuld met nullen zodat de globale en **static** variabelen die *niet* geïnitieerd zijn na een reset opnieuw nul gemaakt worden⁶;

² Lokale variabelen worden opgeslagen in registers of op de stack.

³ Deze variabelen worden bij de start van het programma nul gemaakt.

⁴ Het `SP` register wordt geladen met de waarde `_estack` die in het linkerscript gedefinieerd is.

⁵ In dit deel van de startup code wordt gebruik gemaakt van de symbolen `_sdata`, `_edata` en `_sidata` die in het linkerscript gedefinieerd zijn.

⁶ In dit deel van de startup code wordt gebruik gemaakt van de symbolen `_sbss` en `_ebss` die in het linkerscript gedefinieerd zijn.

- de functie `main` wordt aangeroepen.

Open het bestand `main.c`. De inhoud van dit bestand is ook weergegeven in [listing 1](#). Dit programma maakt geen gebruik van een API en include alleen het

```

1 #include <stdint.h>
2
3 #define RCC_AHB1ENR_BIT_GPIODEN (*(volatile uint32_t*)(0x42000000 + ↵
    ↵ 0x00023830 * 32 + 3 * 4)
4 #define GPIOD_BASE 0x40020C00
5 #define GPIOD_MODER (*(volatile uint32_t*)(GPIOD_BASE + 0x00)
6 // There is something missing here ...
7
8 int main(void)
9 {
10     // GPIO Port D Clock Enable
11     RCC_AHB1ENR_BIT_GPIODEN = 1;
12     // GPIO Port D Pin 15 down to 12 Push/Pull Output
13     GPIOD_MODER = 0x55000000;
14     // Set green and red LEDs
15     GPIOD_ODR = 0x5000;
16     // Do forever:
17     while (1)
18     {
19         // Wait a moment
20         for (volatile int32_t i = 0; i < 1000000; i++);
21         // Flip all LEDs
22         GPIOD_ODR ^= 0xF000;
23     }
24 }
```

Listing 1: De inhoud van het bestand `main.c` in het project `opdr_2_1`.

bestand `<stdint.h>` waarin onder andere het type `uint32_t` gedefinieerd is.

Je ziet op regel 13 dat er naar het `GPIOD_MODER` register⁷ van de STM32F411-VET6 microcontroller wordt geschreven alsof dit een gewone variabele is. Dit is mogelijk dankzij de definities op regel 4 en 5. Op regel 4 wordt symbool `GPIOD_BASE` gedefinieerd als `0x40020C00`, het basisadres van de GPIOD module⁸.

⁷ Zie [paragraaf 8.4.1](#) van RM0383 als je niet meer weet wat de functie van dit IO register is.

⁸ Zie [paragraaf 2.3](#) van RM0383.

Op regel 5 wordt de offset van het `GPIO_MODER` register (`0x00`) bij dit basisadres opgeteld en wordt deze waarde gecast naar een (**volatile**⁹) `uint32_t` pointer. De pointer moet van het type `uint32_t*` zijn omdat het `GPIO_MODER` register 32 bits zonder tekenbit bevat. Vervolgens wordt het symbool `GPIO_MODER` gelijk gemaakt aan waar deze pointer naar wijst door de pointer dereference operator (*) toe te passen. Nadat de preprocessor¹⁰ de defines heeft verwerkt ziet regel 13 er als volgt uit:

```
*(volatile uint32_t*)(0x40020C00) = 0x55000000;
```

Dus er wordt een pointer aangemaakt die wijst naar de `uint32_t` waarde op adres `0x40020C00`, het adres van `GPIO_MODER` register. Vervolgens wordt naar de locatie waar deze pointer naar wijst, het `GPIO_MODER` register, de waarde `0x55000000` geschreven.

Op regel 11 wordt de IO port D clock enabled¹¹ door gebruik te maken van “bit-banding”¹² zoals je in [opdracht 1.9](#) geleerd hebt. De berekening die nodig is om het alias adres van de bit `GPIOEN` in het `AHB1ENR` register te berekenen is gedefinieerd op regel 3.

Als je het project build, dan geeft de compiler een foutmelding omdat het symbool `GPIO_ODR` niet gedefinieerd is. Voeg op regel 6 de definitie van dit symbool toe zodat je dit kunt gebruiken om naar het `GPIO_ODR` register¹³ te schrijven. Debug het aangepaste programma totdat de groene en rode led afwisselend knipperen met de oranje en blauwe led met een frequentie van ongeveer 1,5 Hz.

In het vervolg van deze opdracht ga je zien waarom het keyword **volatile** nodig is in [listing 1](#).

E Kopieer het project `opdr_2_1` naar `opdr_2_1_no_volatile` in de Project Explorer van STM32CubeIDE. Verwijder in het *nieuwe* project het Debug mapje en het `.launch` bestand. Verwijder in `main.c` alle **volatile** keywords. Build en debug het programma.

⁹ Later, in [deelopdracht E](#), leer je waarom deze pointer **volatile** moet zijn.

¹⁰ Zie eventueel <https://gcc.gnu.org/onlinedocs/cpp/>.

¹¹ Zie eventueel [paragraaf 6.3.9](#) van RM0383.

¹² Zie eventueel [paragraaf 2.3.3](#) van RM0383 en [paragraaf 2.2.5](#) van Cortex-M4 Devices Generic User Guide van Arm.

¹³ Zie eventueel [paragraaf 8.4.6](#) van RM0383.

Zoals je ziet, werkt het project nog steeds correct. Je zou nu de *foutieve* conclusie kunnen trekken dat het gebruik van **volatile** niet nodig is. Zet het optimalisatielevel van de compiler op “Optimize (-O1)”¹⁴. Build en debug het programma opnieuw. Je ziet dat de groene en rode led nu continue branden. Zet nu in het project opdr_2_1 het optimalisatielevel van de compiler ook op “Optimize (-O1)”. Build en debug dit project opnieuw. Als het goed is, functioneert dit programma wel correct. Omdat de code van de **for** loop geoptimaliseerd is, knipperen de leds wel iets sneller. Je kunt dus concluderen dat het keyword **volatile** wel degelijk nodig is.

Zoek zelf op wat de betekenis is van het keyword **volatile** in C en verklaar, in je verslag, waarom dit gebruikt moet worden op regel 3 tot en met 6 en regel 20 van [listing 1](#). Geef ook een nette bronvermelding.

Leds aansturen in C met gebruik van de CMSIS API van Arm

Het zelf definiëren van de **volatile** pointers naar de registers van de STM32F411VET6 microcontroller is bewerkelijk en foutgevoelig omdat je de basisadressen en de offsets van de registers die je wilt configureren op moet zoeken in [RM0383](#).

Onder andere om het gebruik van IO-registers eenvoudiger te maken heeft Arm de CMSIS (Common Microcontroller Software Interface Standard) gedefinieerd¹⁵. Deze API is zeer uitgebreid en bevat ook een DSP (Digital Signal Processing), NN (Neural Network) en RTOS (Real-Time Operating System) library. Lees de informatie op de webpagina https://arm-software.github.io/CMSIS_5/General/html/index.html door en zorg dat je de voordelen van het gebruik van CMSIS begrijpt.

Omdat elke fabrikant van microcontrollers zijn eigen peripherals heeft moet de fabrikant zelf CMSIS compatibele header files aanbieden waarmee de IO-registers benaderd kunnen worden. Arm biedt hiervoor wel templates en een tool aan die registerbeschrijvingen in het zogenoemde SVD (System View Description)¹⁶ automatisch om kan zetten naar een CMSIS header file¹⁷. STMicroelectronics levert deze CMSIS compatible headerfile mee met de STM32CubeIDE.

¹⁴ Zie eventueel [paragraaf 2.3.2](#) van UM2609 (STM32CubeIDE user guide).

¹⁵ <https://developer.arm.com/tools-and-software/embedded/cmsis>

¹⁶ Zie eventueel https://arm-software.github.io/CMSIS_5/SVD/html/svd_Format_pg.html.

¹⁷ Zie eventueel https://arm-software.github.io/CMSIS_5/Core/html/group__peripheral__gr.html.

2.2 A Download het project [opdr_2_2.zip](#).

B Importeer dit project in STM32CubeIDE met de menu-optie **File** **Import...**, **General** **Projects from Folder or Archive**, **Next**, **Archive**, selecteer het bestand `opdr_2_2.zip`, **Open** en **Finish**.

C Open in de “Project Explorer” het mapje `opdr_2_2.zip_expanded` **Opdr_2_2** **Core** **Src**.

D Je ziet dat dit project een mapje `opdr_2_2.zip_expanded` **Opdr_2_2** **Core** **CMSIS** bevat. Hierin bevinden zich de CMSIS header files. De bestanden `STM32F411VETX_` en `startup_stm32f411vetx.s` zijn ongewijzigd ten opzichte van [opdracht 2.1](#).

```

1 #include <stdint.h>
2 #include <stm32f4xx.h>
3
4 int main(void)
5 {
6     // GPIO Port D Clock Enable
7     RCC->AHB1ENR = RCC_AHB1ENR_GPIODEN;
8     // GPIO Port D Pin 15 down to 12 Push/Pull Output
9     GPIOD->MODER = GPIO_MODER_MODER12_0 | GPIO_MODER_MODER13_0 | ↵
↵ GPIO_MODER_MODER14_0 | GPIO_MODER_MODER15_0;
10    // Set green and red LEDs
11    /* There is something missing here ...*/ = /* ... and here */;
12    // Do forever:
13    while (1)
14    {
15        // Wait a moment
16        for (volatile int32_t i = 0; i < 1000000; i++);
17        // Flip all LEDs
18        /* There is something missing here ...*/ ^= /* ... and ↵
↵ here */;
19    }
20 }
```

Listing 2: De inhoud van het bestand `main.c` in het project `opdr_2_2`.

Open het bestand `main.c`. De inhoud van dit bestand is ook weergegeven in [listing 2](#). Dit programma maakt gebruik van de CMSIS API die gedefinieerd is in het bestand `<stm32f4xx.h>`.

Je ziet op regel 9 dat er naar het `GPIO_MODER` register van de STM32F411-VET6 microcontroller wordt geschreven alsof dit een element (Engels: member) genaamd `MODER` is van een **struct** waar de pointer genaamd `GPIO` naar wijst¹⁸. Dit is mogelijk dankzij de definities in de CMSIS header file(s). De header file `stm32f4xx.h` include met behulp van een in het project gedefinieerd symbool (`STM32F411xE`) de juiste header file (`stm32f411xe.h`) voor de specifieke STM32F4 microcontroller die wordt gebruikt.

Zet de cursor op het symbool `GPIO` in `main.c` en druk op **F3**. De include file `stm32f411xe.h` wordt nu geopend op de regel waar het symbool `GPIO` gedefinieerd is. Je ziet dat dit een pointer is naar het type `GPIO_TypeDef`. Met een typecast is ervoor gezorgd dat deze pointer wijst naar `GPIO_BASE`. Als je de definitie van `GPIO_BASE` opzoekt (regel 696 in `stm32f411xe.h`), dan zie je dat deze gedefinieerd is als `AHB1PERIPH_BASE + 0x0C00UL`. `AHB1PERIPH_BASE` is op regel 653 gedefinieerd als `PERIPH_BASE + 0x00020000UL`. `PERIPH_BASE` is op regel 638 gedefinieerd als `0x40000000UL`. Als je dit doorrekent, dan is `GPIO_BASE` gelijk aan `0x40020C00` en in [paragraaf 2.3](#) van RM0383 kun je vinden dat dit inderdaad het basisadres van de `GPIO` module is.

Als je de definitie van `GPIO_TypeDef` opzoekt (regel 265 tot en met 276 van `stm32f411xe.h`), dan zie je dat dit inderdaad een **struct** type is met een `uint32_t` member voor elk register in de `GPIO` module¹⁹. Nu rest alleen nog de vraag wat de definitie van het symbool `__IO` is, dat bij alle velden van het struct type `GPIO_TypeDef` wordt gebruikt. Op regel 140 van `stm32f411xe.h` wordt `core_cm4.h` 'included'. Op regel 222 van dit bestand wordt `__IO` gedefinieerd als **volatile**. Als het goed is, heb je inmiddels begrepen waarom dat noodzakelijk is²⁰.

Als je schrijft naar het member `MODER` van de **struct** waar de pointer `GPIO` naar wijst (met de syntax `GPIO->MODER`), dan schrijf je dus naar het `GPIO_MODER` register. In [listing 1](#) werd het 'magic number' `0x55000000` naar het `GPIO_MODER` register geschreven. In [listing 2](#) wordt in plaats van een magic number gebruik gemaakt van bit maskers die in `stm32f411xe.h` (zie regel 2462 en verder) gedefinieerd zijn. Dit maakt de code (iets) leesbaarder.

¹⁸ Zie [paragraaf 8.2](#) van "De programmeertaal C — een overzicht" als je dit niet snapt.

¹⁹ De volgorde van de members in de **struct** moet natuurlijk overeenkomen met de volgorde van de registers in de module. In [paragraaf 8.4](#) van RM0383 kun je zien dat dit inderdaad het geval is.

²⁰ Zie opdracht 2.1 [deelopdracht E](#).

Op regel 7 wordt enkel de IO port D clock enabled. Het is met de huidige versie van CMSIS voor de STM32F4 niet mogelijk om “bit-banding” te gebruiken²¹. Het gebruik van het bit masker `RCC_AHB1ENR_GPIODEN` in plaats van het ‘magic number’ `0x08` maakt deze code (veel) leesbaarder.

Als je het project build, dan geeft de compiler een foutmelding omdat er op vier plaatsen nog iets mist. Voeg op regel 11 en 18 de juiste code toe om naar het `GPIOD_ODR` register te schrijven met behulp van de CMSIS API. Maak geen gebruik van ‘magic numbers’ maar van de bit masks die in de CMSIS API gedefinieerd zijn. Debug het aangepaste programma totdat de groene en rode led afwisselend knipperen met de oranje en blauwe led met een periodetijd van ongeveer 1,5 s.

E Vergelijk de assembly code die de compiler produceert voor regel 13 van [listing 1](#) met de assembly code die de compiler produceert voor regel 9 van [listing 2](#). Beide regels C-code doen functioneel hetzelfde. Je kunt de door de compiler geproduceerde assembly code eenvoudig bekijken door het programma te openen en de debugger en via het menu `Window` `Show View` `Disassembly` het “Disassembly” window te openen. Wat constateer je en welke conclusie trek je hieruit?

Leds aansturen in C met gebruik van de LL en HAL APIs van STM

STMicroelectronics levert zelf ook een aantal verschillende APIs waaronder de LL (Low-Layer) API en de HAL (Hardware Abstraction Layer) API²². Daarnaast stelt STM ook een configuratietool beschikbaar genaamd STM32CubeMX²³. Deze tool is opgenomen in de STM32CubeIDE en genereert C-code die gebruik maakt van de LL en/of HAL API. Beide API's worden met elkaar vergeleken in een presentatie van STM²⁴ en een kort overzicht van deze vergelijking vind je in [figuur 2](#).

²¹ In principe is dit wel mogelijk door een `struct` te definiëren in de peripheral bit-band alias memory region. Zie eventueel https://github.com/HR-ELEKTRO/rtS1/wiki/Blinky-CMSIS_BB.md.

²² STM levert ook de SPL (Standard Peripheral Libraries) API maar deze is verouderd en wordt niet meer verder ontwikkeld.

²³ <https://www.st.com/en/development-tools/stm32cubemx.html>

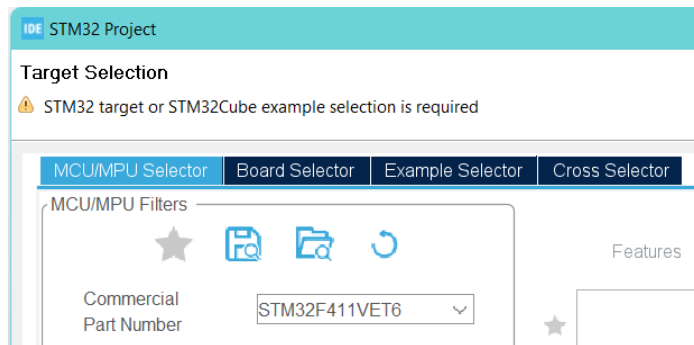
²⁴ https://www.st.com/content/ccc/resource/sales_and_marketing/presentation/product_presentation/37/55/ff/bc/a8/71/4f/c5/stm32_embedded_software_offering.pdf/files/stm32_embedded_software_offering.pdf/jcr:content/translations/en.stm32_embedded_software_offering.pdf

	Portability	Optimization (Memory & Mips)	Easy	Readiness	Hardware coverage
HAL APIs	+++	+	++	+++	+++
LL APIs	+	+++	+	++	++

Figuur 2: Een vergelijking tussen de HAL en LL APIs van STM.

2.3 In deze opdracht ga je STM32CubeMX gebruiken om de pinnen waarmee de user leds aangestuurd worden te configureren en ga je de LL API gebruiken om de leds aan te sturen.

- A** Kies in STM32CubeIDE de menu-optie `File > New > STM32 Project`. Na enige tijd opent het “Target Selection” window van STM32CubeMX.
- B** Selecteer bij “Commercial Part Number” de STM32F411VET6, zie [figuur 3](#).



Figuur 3: Selecteer STM32F411VET6.

- C** Selecteer nu in de “MCUs/MPUs List” ook de STM32F411VET6, zie [figuur 4](#).

MCUs/MPUs List: 2 items

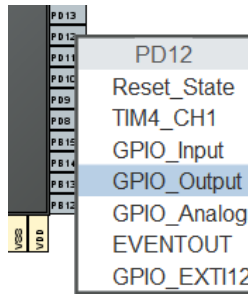
+

Display similar items

*	Commercial Part No	Part No	Refere...	Mark... X	Unit P...
☆	STM32F411VET6	STM32...	STM32...	Active	4.5319
☆	STM32F411VET6TR		STM32...	Active	4.5319

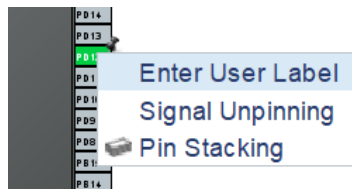
Figuur 4: Selecteer STM32F411VET6.

- D** Klik op en vul bij Project Name: opdr_2_3 in klik op . De eerste keer zal worden gevraagd of je de “associated view” wilt openen. Druk dan op . Het bestand opdr_2_3.ioc wordt geopend en je ziet de “Pinout view”. Hierin kun je de pinnen waarop de user leds zijn aangesloten configureren.
- E** Klik op pin PD12 en selecteer “GPIO_Output”, zie [figuur 5](#). Klik vervolgens met

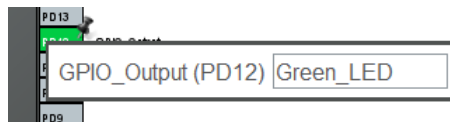


Figuur 5: Selecteer GPIO_Output.

de rechtermuisknop nogmaals op pin PD12 kies “Enter User Label”, zie [figuur 6](#). Vul als label Green_LED in, zie [figuur 7](#).

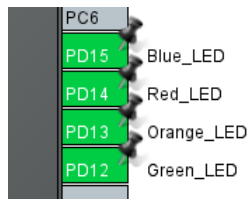


Figuur 6: Definieer een label voor pin PD12.



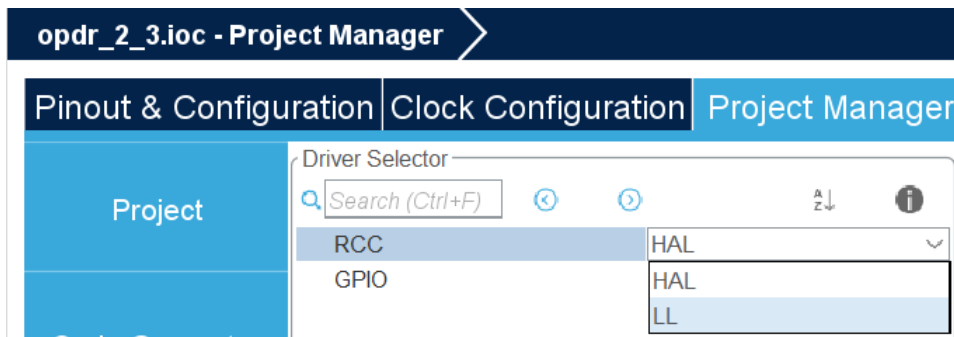
Figuur 7: Definieer het label Green_LED voor pin PD12.

- F** Doe hetzelfde voor de pinnen PD13 (Orange_LED), PD14 (Red_LED) en PD15 (Blue_LED), zie [figuur 8](#).
- G** Klik op de tab “Project Manager” en klik vervolgens aan de linkerkant op “Advanced Settings”. Klik in de “Driver Selector” bij de regel die begint met “RCC” op



Figuur 8: Alle pinnen voor de user leds zijn geconfigureerd als output pin en zijn van een label voorzien.

het woord “HAL” en selecteer “LL”, zie [figuur 9](#). Doe hetzelfde bij de regel die



Figuur 9: Kies voor de LL API.

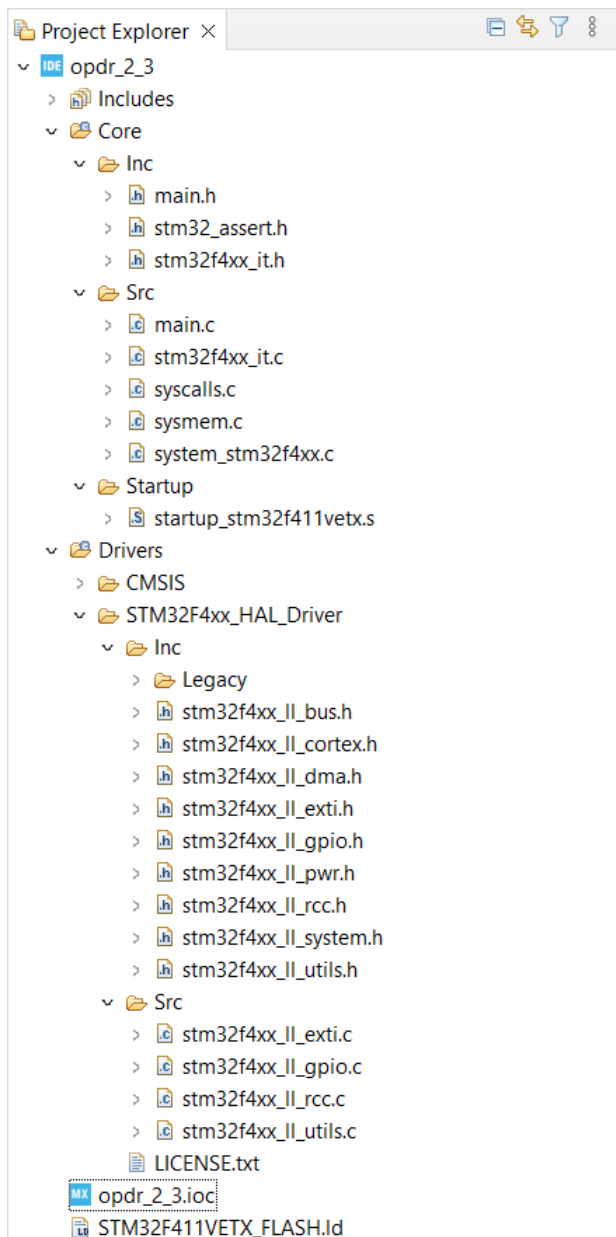
begint met “GPIO”.

H Kies in het hoofdmenu voor de optie **File > Save**. De eerste keer zal worden gevraagd of je de code wilt genereren. Kies voor **Yes**. In een aantal gevallen moet er extra software gedownload worden om de code te genereren. In dat geval moet je via de menuoptie **myST** inloggen of een account aanmaken. De configuratiecode wordt nu gegenereerd. Bekijk de gegenereerde bestanden in de “Project Explorer”, zie [figuur 10](#).

I Open het bestand `main.c`. In de code staat met commentaar aangegeven waar je code kunt toevoegen. Het is belangrijk om alleen code toe te voegen op de aangegeven plaatsen zodat de code behouden blijft als je de configuratietool opnieuw gebruikt om aanpassingen te maken.

Voeg achter `/* USER CODE BEGIN 2 */` de volgende code toe:

```
LL_GPIO_SetOutputPin(Green_LED_GPIO_Port, ←
↪ Green_LED_Pin | Red_LED_Pin);
```



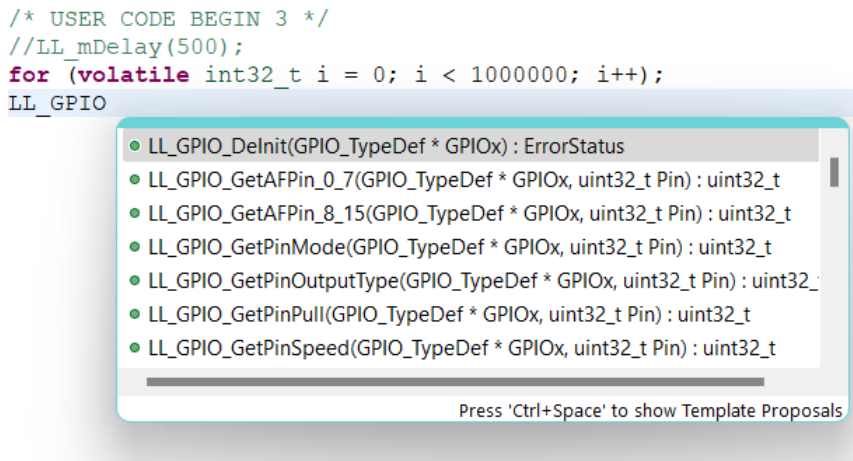
Figuur 10: De gegenereerde bestanden voor project opdr_2_3.

Je maakt hier gebruik van de functie `LL_GPIO_SetOutputPin` uit de LL API²⁵. De symbolen zoals `Green_LED_Pin` die als argumenten aan deze functie meegegeven worden zijn gedefinieerd (door STM32CubeMX) in `main.h`.

Voeg achter `/* USER CODE BEGIN 3 */` de volgende code toe:

```
for (volatile int32_t i = 0; i < 1000000; i++);
// Flip all LEDs
/* There is something missing here ...*/
```

Vul de benodigde code in om de signalen op alle leds te inverteren. Maak daarbij gebruik van een functie uit de LL API. Je kunt dit opzoeken in [UM1725](#). Het is handig om (ook) gebruik te maken van “code completion”. Als je in de editor bijvoorbeeld `LL_GPIO` intypt en op `Ctrl` + `Spatie` drukt, dan opent een dropdownmenu met alle identifiers die beginnen met `LL_GPIO`, zie [figuur 11](#).



Figuur 11: Code completion.

J Als je gebruik maakt van de LL API, dan maak je gebruik van functies in plaats van dat je zelf de IO-registers van de STM32F411VET6 microcontroller gebruikt. Dit geeft natuurlijk enige overhead. Als je een programma compileert in STM32-CubeIDE, dan kun je rechtsonder in het “Build Analyzer” window in het “Memory Details” tabblad zien hoe groot de `.text` section is (die alle machinecode van het programma bevat) door op het “Refresh” knopje  te drukken, zie [figuur 12](#).

²⁵ Zie [hoofdstuk 82](#) van UM1725 (Description of STM32F4 HAL and low-layer drivers user manual).

Build Analyzer × Static Stack Analyzer

opdr_2_2.elf - /opdr_2_2/Debug

Memory Regions Memory Details

Selection: 316 B

Search

Name	Run address (V...	Load address (LM...	Size
FLASH	0x08000000		512 KB
> .isr_vector	0x08000000	0x08000000	408 B
> .text	0x08000198	0x08000198	316 B
.rodata	0x080002d4	0x080002d4	0 B
.preinit_array	0x080002d4	0x080002d4	0 B
.init_array	0x080002d4	0x080002d4	4 B

Figuur 12: De grootte van de .text section van opdracht 2.2.

Vul tabel 1 verder in.

Tabel 1: De grootte van de .text section.

Opdracht	Optimization level	Grootte .text section
2.2	None (-O0)	316 B
	Optimize for size (-Os)	304 B
2.3	None (-O0)	 B
	Optimize for size (-Os)	 B

Wat constateer je en welke conclusie trek je hieruit?

2.4 In deze opdracht ga je nogmaals STM32CubeMX gebruiken om de pinnen waarmee de user leds aangestuurd worden te configureren, maar ga je, in plaats van de LL API, de HAL API gebruiken om de leds aan te sturen.

- A** Herhaal opdracht 2.3 [deelopdrachten A](#) tot en met [C](#).
- B** Herhaal opdracht 2.3 [deelopdracht D](#), maar kies nu als Project Name: opdr_2_4.
- C** Herhaal opdracht 2.3 [deelopdrachten E](#) en [F](#).
- D** opdracht 2.3 [deelopdracht G](#) sla je nu over, zodat de HAL API in plaats van de LL API gebruikt wordt.

E Herhaal opdracht 2.3 [deelopdracht H](#).

F Open het bestand `main.c`. Voeg achter `/* USER CODE BEGIN 2 */` de volgende code toe:

```
HAL_GPIO_WritePin(Green_LED_GPIO_Port, Green_LED_Pin | ↔
↪ Red_LED_Pin, GPIO_PIN_SET);
```

Je maakt hier gebruik van de functie `HAL_GPIO_WritePin` uit de HAL API²⁶. De symbolen zoals `Green_LED_Pin` die als argumenten aan deze functie meegegeven worden zijn gedefinieerd (door STM32CubeMX) in `main.h`.

Voeg achter `/* USER CODE BEGIN 3 */` de volgende code toe:

```
for (volatile int32_t i = 0; i < 1000000; i++);
// Flip all LEDs
/* There is something missing here ...*/
```

Vul de benodigde code in om de signalen op alle leds te inverteren. Maak daarbij gebruik van een functie uit de HAL API. Je kunt dit opzoeken in [UM1725](#) en gebruik maken van “code completion”.

G Als je gebruik maakt van de HAL API, dan maak je gebruik van functies met een hoger abstractieniveau dan de functies uit de LL API. Dit geeft natuurlijk enige overhead. Vul [tabel 2](#) verder in.

Tabel 2: De grootte van de `.text` section.

Opdracht	Optimization level	Grootte <code>.text</code> section
2.3	None (-O0)	... kB
	Optimize for size (-Os)	1,0 kB
2.4	None (-O0)	... kB
	Optimize for size (-Os)	... kB

Wat constateer je en welke conclusie trek je hieruit?

²⁶ Zie [hoofdstuk 31](#) van UM1725 (Description of STM32F4 HAL and low-layer drivers user manual).

Aanpassen klokfrequentie

Tot nu toe heb je bij alle opdrachten de standaard instellingen van de klokfrequentie van de STM32F411VET6 microcontroller gebruikt. De processor en alle interne bussen werken dan op de zogenoemde High Speed Internal (HSI) klok. Dit is een 16 MHz RC-oscillator. Deze oscillator is niet erg nauwkeurig en wordt na productie gekalibreerd door STM op een nauwkeurigheid van 1 % bij een omgevingstemperatuur T_A van 25 °C.

Als je een nauwkeurigere klokfrequentie wilt, dan kun je gebruik maken van een extern aangesloten kristal de zogenoemde High Speed External (HSE) klok. Op het STM32F411E-DISCO ontwikkelbord is een 8 MHz kristal verbonden met de pinnen PH0 en PH1 van de STM32F411VET6 microcontroller zie [UM1842](#) (Discovery kit with STM32F411VE MCU user manual). Deze HSE kan intern met behulp van een Phase-locked loop (PLL) nog verhoogd worden tot maximaal 100 MHz.

In dit deel van de opdracht ga je de STM32F411VET6 microcontroller op de maximale klokfrequentie laten draaien. Dit doe je eerst met behulp van de CMSIS API en daarna met behulp van de STM32CubeMX configuratietool, die gebruik maakt van de LL of HAL API.

Klokfrequentie aanpassen met behulp van de CMSIS API van Arm

2.5 In deze opdracht maak je eerst een kopie van het project `opdr_2_2` en voer je daarna stap voor stap alle code in die nodig is om de STM32F411VET6 microcontroller op 100 MHz te laten draaien. Alle benodigde informatie zoek je op in [RM0383](#) (STM32F411xC/E advanced Arm-based 32-bit MCUs reference manual).

A Kopieer het project `opdr_2_2` naar `opdr_2_2_100MHz` in de Project Explorer van STM32CubeIDE. Verwijder in het *nieuwe* project het Debug mapje en het `.launch` bestand.

Om te beginnen ga je de processor laten draaien op de HSE klok van 8 MHz in plaats van op de HSI klok op 16 MHz.

B Lees [paragraaf 6.2.1](#) van RM0383. Zoals al eerder vermeld, is op het STM32F411E-DISCO ontwikkelbord een 8 MHz kristal aangesloten. Om dit kristal te kunnen gebruiken moet:

- de HSEON bit in het `RCC_CR` register één worden gemaakt;

- gewacht worden tot de HSERDY flag in het RCC_CR register één is geworden.

De benodigde C code is als volgt:

```
// Enable HSE
RCC->CR |= RCC_CR_HSEON;
// Wait until HSE is stable
while ((RCC->CR & RCC_CR_HSERDY) == 0);
```

Lees [paragraaf 6.2.6](#) en [paragraaf 6.3.3](#) van RM0383 en zoek op hoe je de HSE oscillator kunt selecteren als systeemklok en hoe je daarna kunt wachten tot dit gelukt is. De benodigde C code heeft de volgende structuur:

```
// Select HSE as the system clock
RCC->CFGR |= /* ... */;
// Wait until HSE used as the system clock
while ((*/* ... */ & RCC_CFGR_SWS_HSE) == 0);
```

Nu kan (indien gewenst) de HSI klok uitgeschakeld worden omdat deze niet meer wordt gebruikt. De benodigde C code is als volgt:

```
// Disable HSI
RCC->CR &= ~RCC_CR_HSION;
```

Voeg de benodigde C-code toe aan het begin van `main.c` om de HSE klok in plaats van de HSI klok te gebruiken als systeemklok. Build en debug het programma. Als het goed is, knipperen de leds nu met een periodetijd van 3 s. Twee keer zo langzaam dan bij opdracht 2.2 [deelopdracht D](#).

We gaan nu de Phase-locked loop (PLL)²⁷ configureren om de systeemklok te verhogen naar 100 MHz. Om de STM32F411VET6 microcontroller op 100 MHz te laten werken moeten ook de Power controller (PWR) en Flash interface instellingen aangepast worden.

- C** Verwijder de code waarin overgeschakeld wordt van HSI naar HSE maar laat de code waarin de HSE wordt aangezet staan.
- D** Lees [paragraaf 3.4.1](#) van RM0383. Om de processor op 100 MHz te laten draaien moet dus `VOS[1:0] = 0x11`. Vervolgens kunnen je de stappen volgen die onder het kopje “Increasing the CPU frequency” staan opgesomd. Dit wordt hieronder stap voor stap toegelicht.

²⁷ Zie eventueel https://en.wikipedia.org/wiki/Phase-locked_loop.

De VOS[1:0] bits bevinden zich in de Power controller (PWR), zie [hoofdstuk 5](#) van RM0383. Om deze module te kunnen gebruiken, moet de module worden voorzien van een kloksignaal. Zoek in [RM0383](#) op hoe je dit doet²⁸. De benodigde C code heeft de volgende structuur:

```
// Power interface clock enable
/* ... */ |= /* ... */;
```

Zoek op hoe je de “voltage scaling output selection” kunt aanpassen. De benodigde C code heeft de volgende structuur:

```
// Regulator voltage scaling output selection Scale 1 ↔
↪ mode <= 100 MHz
/* ... */ |= /* ... */;
```

Als gegeven is dat de STM32F411VET6 microcontroller op het STM32F411E-DISCO ontwikkelbord voorzien is van een voedingsspanning van 3 V²⁹, hoeveel wait states moeten er dan ingesteld worden in het FLASH_ACR register volgens [paragraaf 3.4.1](#) van RM0383? De benodigde C code heeft de volgende structuur:

```
// Use ... wait states when reading Flash at 100 MHz.
FLASH->ACR = /* ... */;
// Wait until ... wait states are used
while ((FLASH->ACR & /* ... */) == 0);
```

Nu moet de PLL ingesteld worden om een frequentie van 100 MHz te genereren. Lees [paragraaf 6.2.3](#) van RM0383. Je moet dus de benodigde waarden van PLLN, PLLM en PLLP bepalen³⁰, volgens de formules:

$$f_{(\text{VCO clock})} = f_{(\text{PLL clock input})} \times (\text{PLLN}/\text{PLLM})$$

$$f_{(\text{PLL general clock output})} = f_{(\text{VCO clock})}/\text{PLLP}$$

Omdat je HSE als input voor de PLL gaat gebruiken, geldt:

$$f_{(\text{PLL clock input})} = 8 \text{ MHz}$$

²⁸ Zoek naar “Power interface clock enable”.

²⁹ https://www.st.com/content/ccc/resource/technical/layouts_and_diagrams/schematic_pack/group1/2e/42/7b/ea/56/55/40/25/MB1115-F411VE-D01_schematic/files/MB1115-F411VE-D01_Schematic.pdf/jcr:content/translations/en.MB1115-F411VE-D01_Schematic.pdf

³⁰ PLLQ is niet relevant omdat je de USB OTG FS en SDIO niet gebruikt.

Omdat je een klokfrequentie van 100 MHz wilt genereren, geldt:

$$f_{(\text{PLL general clock output})} = 100 \text{ MHz}$$

Zoals je in [paragraaf 6.2.3](#) hebt gelezen, wordt de PLL input klok gedeeld door PLLM en vervolgens gebruikt als input voor de voltage-controlled oscillator (VCO). Deze frequentie moet $1 \text{ MHz} \leq f \leq 2 \text{ MHz}$ zijn en er wordt geadviseerd om 2 MHz te kiezen. Dus moet je $\text{PLLM} = 4$ kiezen.

Bepaal nu de benodigde waarden van PLLN en PLLP. Hou daarbij rekening met het feit dat PLLP alleen 2, 4, 6 en 8 kan zijn en dat $100 \text{ MHz} \leq f_{(\text{VCO clock})} \leq 432 \text{ MHz}$ ³¹. Je hebt nu alle informatie om het RCC_PLLCFGR register te configureren. De benodigde C code heeft de volgende structuur:

```
// Use PLL to generate 100 MHz clock
RCC->PLLCFGR = 0x20000000
    | RCC_PLLCFGR_PLLSRC_HSE
    | (/* select P */ << RCC_PLLCFGR_PLLP_Pos)
    | (/* N */ << RCC_PLLCFGR_PLLN_Pos)
    | (4 << RCC_PLLCFGR_PLLM_Pos);
```

Het gebruik van de constante 0x20000000 zorgt ervoor dat de gereserveerde velden in het register hun reset waarde behouden (zoals gespecificeerd in [paragraaf 6.2.3](#)).

Vervolgens moet de PLL enabled worden en moet gewacht worden tot deze “locked” is. De benodigde C code heeft de volgende structuur:

```
// Enable PLL
RCC->CR |= /* ... */;
// Wait until PLL is locked
while ((RCC->CR & /* ... */) == 0);
```

Zoek in [paragraaf 6.3.3](#) van RM0383 op hoe je de PLL output kunt selecteren als systeemklok en hoe je daarna kunt wachten tot dit gelukt is. Daarbij moet je wel een prescaler van 2 kiezen voor de APB1 klok omdat deze bus slechts op maximaal 50 MHz kan draaien³². De benodigde C code heeft de volgende structuur:

³¹ Er zijn meerdere (twee) combinaties van PLLN en PLLP die het gewenste resultaat opleveren.

³² Zie eventueel [hoofdstuk 2](#) van RM0383 om te zien welke peripherals op deze bus zijn aangesloten.

```

    // Select PLL as the system clock. AHB clock divided ↩
    ↪ by 2.
    RCC->CFGR |= /* ... */ | RCC_CFGR_PPRE1_DIV2;
    // Wait until PLL used as the system clock
    while ((RCC->CFGR & /* ... */) == 0);

```

Nu kan (indien gewenst) de HSI klok uitgeschakeld worden omdat deze niet meer wordt gebruikt. De benodigde C code is als volgt:

```

// Disable HSI
RCC->CR &= ~RCC_CR_HSION;

```

Voeg de benodigde C-code toe aan main.c om de PLL klok in plaats van de HSI klok te gebruiken als systeemklok. Build en debug het project. Je zou misschien verwachten dat de leds nu $100/16 = 6,25$ keer zo snel knipperen dan bij opdracht 2.2 [deelopdracht D](#). Dus met een periodetijd van $1,5/6,25 = 0,24$ s. Als je de periodetijd meet met een logic analyzer, dan vind je een periodetijd van 0,32 s. Verklaar waarom de periodetijd langer is dan verwacht. Tip: vergelijk het aantal CPU klokcycles tussen het inverteren van de signalen op de leds bij deze opdracht met opdracht 2.2 [deelopdracht D](#).

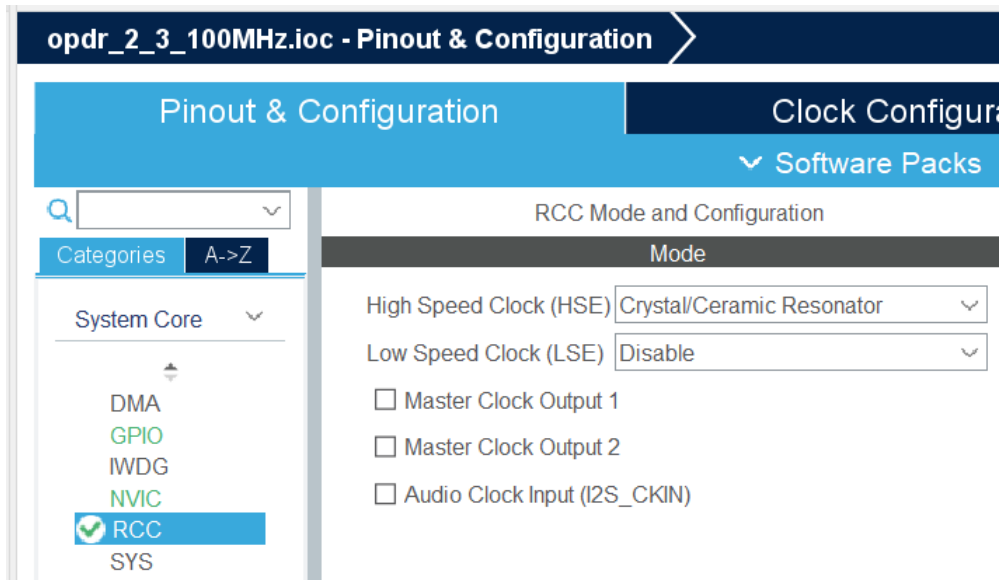
E Lees [paragraaf 3.4.2](#) van RM0383. Kopieer het project opdr_2_2_100MHz naar opdr_2_2_100MHz_ART in de Project Explorer van STM32CubeIDE. Verwijder in het *nieuwe* project het Debug mapje en het .launch bestand. Pas de code aan zodat alle features van de ART Accelerator (data cache, instruction cache en prefetch) aangezet worden. Build en debug het project. Wat is nu het aantal CPU klokcycles tussen het inverteren van de signalen op de leds? Met welke periodetijd knipperen de leds nu?

Klokfrequentie aanpassen met behulp van de STM32CubeMX configuratietool

2.6 In deze opdracht maak je eerst een kopie van het project opdr_2_3 en voer je daarna stap voor stap alle code in die nodig is om de STM32F411VET6 microcontroller op 100 MHz te laten draaien.

A Kopieer het project opdr_2_3 naar opdr_2_3_100MHz in de Project Explorer van STM32CubeIDE. Verwijder in het *nieuwe* project het Debug mapje en het .launch bestand. Rename het bestand opdr_2_3.ioc naar opdr_2_3_100MHz.ioc.

- B** Open het bestand `opdr_2_3_100MHz.ioc`. Kies aan de linkerkant onder “System Core” voor RCC en selecteer bij “High Speed Clock (HSE)” voor “Crystal/Ceramic Resonator”, zie zie [figuur 13](#). Je ziet dat de pinnen PH0 en PH1 nu ook geconfigureerd zijn.



Figuur 13: Configureer HSE voor het gebruik van een kristal.

- C** Klik nu op het tabblad “Clock Configuration” en:

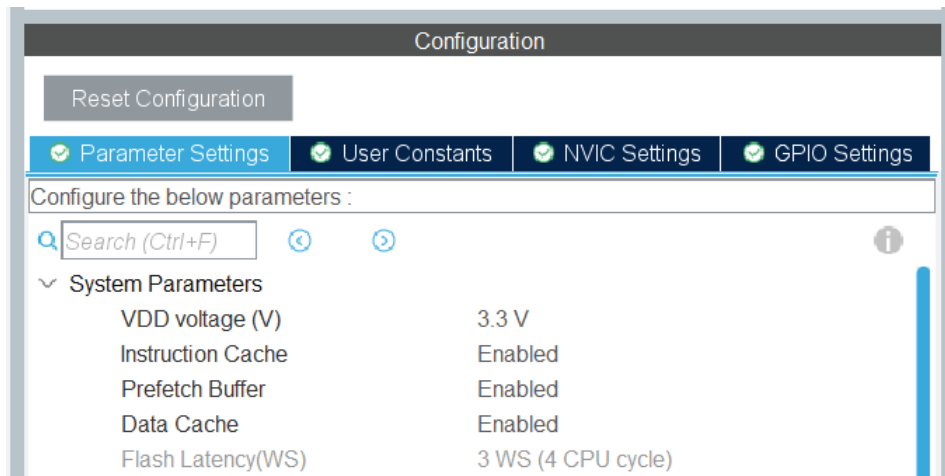
- maak de “Input frequency” van de HSE gelijk aan 8 MHz;
- selecteer HSE als input voor de PLL;
- selecteer PLLCLK als systeemklok;
- klik op “Resolve Clock Issues”;
- selecteer PLLCLK nogmaals als systeemklok;
- klik nogmaals op “Resolve Clock Issues”.

Kies in het hoofdmenu voor de optie `File > Save`. Build en debug het project.

- Met welke periodetijd knipperen de leds nu?
- Welke conclusie trek je hieruit?

Open het bestand `opdr_2_3_100MHz.ioc`. Kies aan de linkerkant onder “System Core” voor RCC en kijk bij “Configuration” naar de “Parameter Settings”, zie

figuur 14. Vreemd genoeg komt dit (in ieder geval bij mij) *niet* overeen met



Figuur 14: De system parameters.

de gevonden periodetijd³³. Er lijkt hier sprake te zijn van een bug want als de HAL API in plaats van de LL API wordt gebruikt, dan is de periodetijd wel zoals verwacht³⁴.

2.7 Welke opdracht vond je eenvoudiger, [opdracht 2.5](#) of [opdracht 2.6](#)? Welke conclusie trek je hieruit?

Vertraging onafhankelijk van de klokfrequentie

In de voorgaande opdrachten was de frequentie waarop de leds knipperden afhankelijk van de klokfrequentie van de processor en de optimalisatie-instellingen van de compiler. In de praktijk wil je dit meestal niet. De LL en HAL APIs bieden hier een oplossing voor in de vorm van “delay” functies.

2.8 **A** Pas [opdracht 2.6](#) aan zodat de leds knipperen met een frequentie van 1 Hz (0,5 s aan en 0,5 s uit). Maak daarbij gebruik van een functie uit de LL API.

³³ En ook niet met de instellingen van het FLASH_ACR register.

³⁴ Wie het beter weet, mag het zeggen.

- B** Pas [opdracht 2.4](#) aan zodat de leds knipperen met een frequentie van 1 Hz (0,5 s aan en 0,5 s uit). Maak daarbij gebruik van een functie uit de HAL API.

Verslag week 1-2

Ter afsluiting van het eerste deel van deze cursus moet een verslag geschreven worden. Daarnaast maak je deze week ook de [assembly assignment](#). Je mag alle bronnen gebruiken die je kunt vinden, zolang je er maar op de juiste manier naar verwijst in je verslag. Wij geven de voorkeur aan de [IEEE reference style](#). Je mag ook Copilot, ChatGPT of een ander groot taalmodel gebruiken om je te helpen met deze opdracht, maar je bent verplicht om dit per geval in je verslag te vermelden³⁵.

Dit verslag moet uit twee delen bestaan:

- de relevante programmacode van de opdrachten van week 2 en een korte uitleg per opdracht;
- de uitwerking van de [assembly assignment](#).

Document

Het document mag in het Engels of in het Nederlands geschreven worden. De relevante en aangepaste broncode moet worden bijgevoegd met behulp van kleurcodering. Het document moet het volgende bevatten:

- titelpagina met de namen van de studenten, studentnummers en de naam van deze cursus;
- informatie over de opdrachten van week 2 met broncodes;
- de antwoorden op de vragen die in deze opdrachten gesteld worden;
- een beschrijving van je testmethode en testresultaten.
- de assembly functies die je geschreven hebt voor de [assembly assignment](#);
- de C programma's die je gebruikt hebt om deze assembly functies te testen en de resultaten van die tests;
- de antwoorden op de vragen die gesteld zijn in de [assembly assignment](#).

³⁵ Zie: https://hrelektrotechniek.bitbucket.io/LLM/11m_ebook.pdf.

Inleveren

Het verslag wordt ingeleverd als één PDF-document. De bestandsnaam heeft de volgende conventie: studentnummer_achternaam_studentnummer_achternaam.pdf

Bijvoorbeeld: 1012345_Dijkstra_1054321_Hoare.pdf

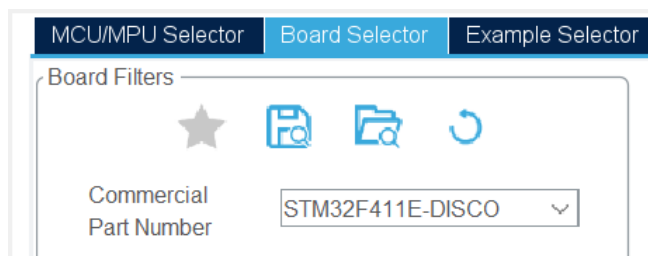
Dit bestand moet geüpload worden in de betreffende opdracht in de RTS1 Brightspace cursus. Slechts één van de twee samenwerkende studenten hoeft het bestand te uploaden.

Let op! Het volgende (laatste) deel van deze opdracht is niet verplicht en hoeft niet in het verslag opgenomen te worden.

Het configureren van het gehele STM32F411E-DISCO ontwikkelbord met STM32CubeMX

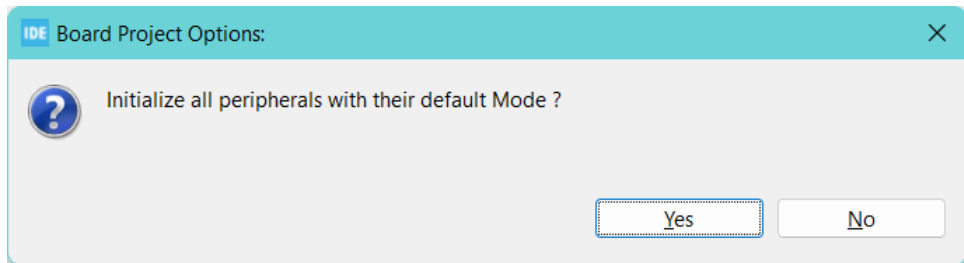
De STM32CubeMX configuratietool heeft ook een mogelijkheid om alle pinnen en peripherals van een STM ontwikkelbord in één keer te configureren. Dit kan erg handig zijn als je het STM32F411E-DISCO ontwikkelbord behalve voor deze cursus ook nog voor iets anders wilt gebruiken.

- 2.9 A** Kies in STM32CubeIDE de menu-optie `File > New > STM32 Project`. Na enige tijd opent het “Target Selection” window van STM32CubeMX.
- B** Klik op het tabblad “Board Selector” en selecteer bij “Commercial Part Number”: STM32F411E-DISCO, zie [figuur 15](#). Selecteer het bord ook in de “Boards List”



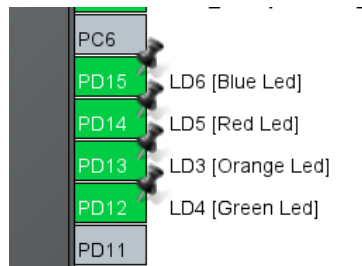
Figuur 15: Selecteer STM32F411E-DISCO.

in het midden van het window. Klik op **Next >**. Kies als projectnaam `Opdr_2_9`. Klik op **Finish**. Beantwoord de vraag “Initialize all peripherals with their default Mode?” met “Yes”, zie [figuur 16](#).



Figuur 16: Initialiseer alle peripherals.

Alle pinnen zijn geconfigureerd zoals aangesloten op het STM32F411E-DISCO ontwikkelbord. Bijvoorbeeld de pinnen waarop de user leds zijn aangesloten, zie [figuur 17](#). De labels die gebruikt worden (LD3 tot en met LD6) komen overeen met de namen die



Figuur 17: De pinnen waarop de user leds zijn aangesloten zijn als volgt geconfigureerd.

in [UM1842](#) (Discovery kit with STM32F411VE MCU user manual) gebruikt worden.

A Kies in het hoofdmenu voor de optie **File > Save**. Open het bestand `main.c`. Voeg achter `/* USER CODE BEGIN 2 */` de volgende code toe:

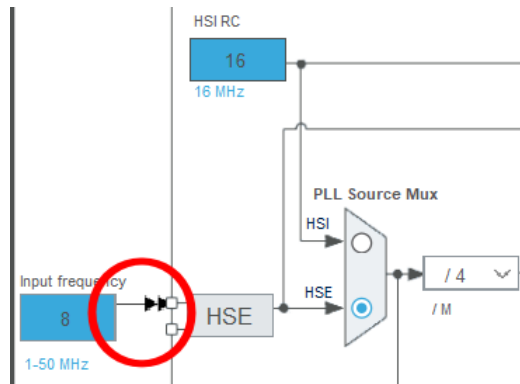
```
HAL_GPIO_WritePin(LD4_GPIO_Port, LD4_Pin | LD5_Pin, ↔
↪ GPIO_PIN_SET);
```

Voeg achter `/* USER CODE BEGIN 3 */` de volgende code toe:

```
HAL_Delay(500);
HAL_GPIO_TogglePin(LD4_GPIO_Port, LD4_Pin | LD5_Pin | ↔
↪ LD4_Pin | LD5_Pin);
```

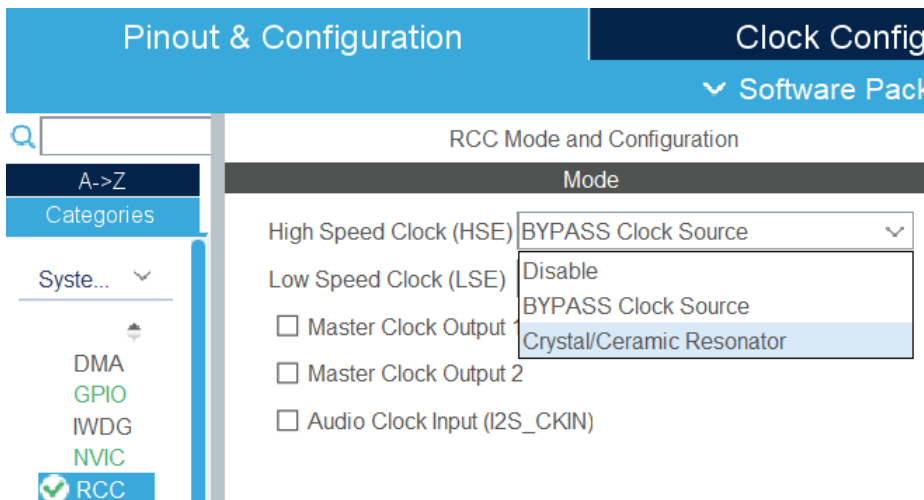
Build en debug het project en constateer dat dit *niet* werkt.

- B** Open in het bestand `opdr_2_9.ioc` de “Clock Configuration” en constateer dat de HSE niet correct geconfigureerd is, zie [figuur 18](#). Klik op het tabblad “Pinout



Figuur 18: De HSE is niet correct geconfigureerd.

& Configuration” Selecteer onder “System Core” RCC. Selecteer bij “High Speed Clock (HSE)” “Crystal/Ceramic Resonator” in plaats van “BYPASS Clock Source”, zie [figuur 19](#).

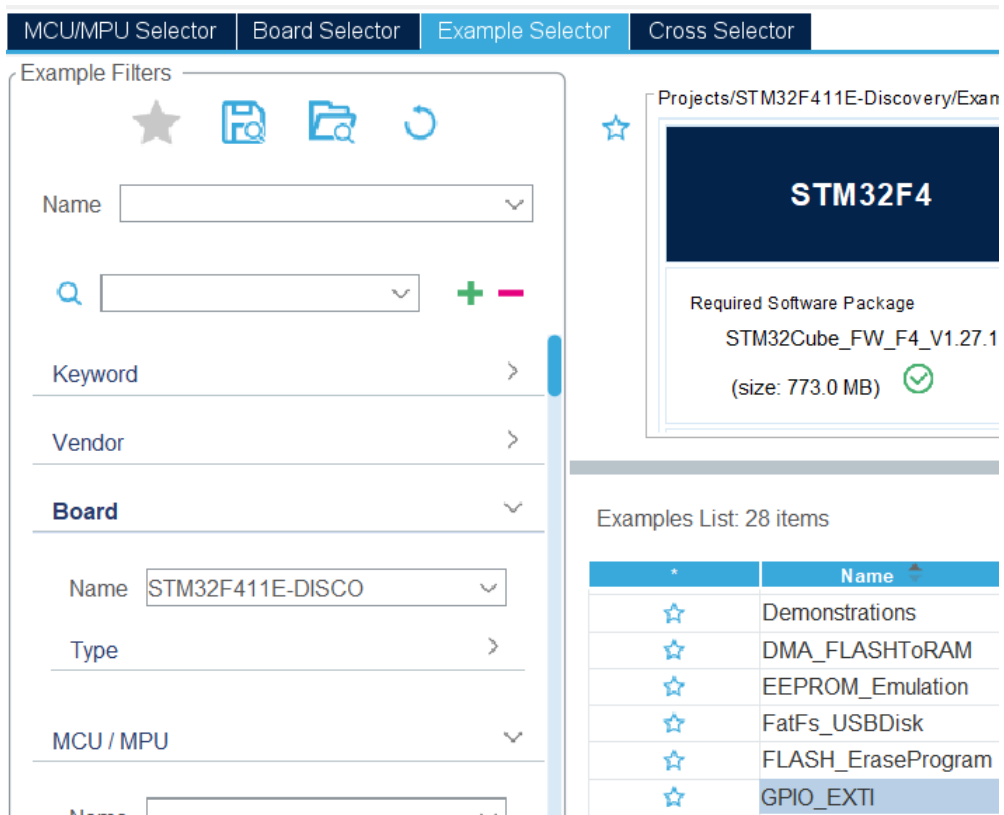


Figuur 19: De correcte configuratie van de HSE clock.

Power Cycle het bord. Build en debug het project en constateer dat het nu *wel* werkt.

Voorbeeldprojecten voor het STM32F411E-DISCO ontwikkelbord

Als je een nieuw STM32 project aanmaakt, dan kun je in het “Target Selection” window van STM32CubeMX ook kiezen voor “Example Selector”. Als je vervolgens bij “Board” STM32F411E-DISCO selecteert, dan kun je diverse voorbeeldprojecten voor dit ontwikkelbord selecteren, zie [figuur 20](#). Kies bijvoorbeeld voor GPIO_EXTI. Zoals je ziet, als je dit



Figuur 20: Voorbeeldprojecten voor het STM32F411E-DISCO ontwikkelbord.

project opent, maakt dit project³⁶ geen gebruik van de STM32CubeMX configuratietool. Er wordt gebruik gemaakt van de Board Support Package (BSP) API³⁷ die gebruik maakt van de

³⁶ Dit geldt ook voor de overige voorbeeldprojecten.

³⁷ <https://github.com/HR-ELEKTRO/rts1/wiki/Books/dm00440740-stm32cube-bsp-drivers-development-guidelines-stmicroelectronics.pdf>

HAL API. De documentatie is beschikbaar op <https://documentation.help/STM32F411E-Discovery-BSP-Drivers/>.

De voorbeeldprojecten zijn ontwikkeld met behulp van een oude IDE van STM genaamd “System Workbench”³⁸ die inmiddels is vervangen door STM32CubeIDE. Het project wordt bij het openen omgezet naar een STM32CubeIDE project, zie het bestand `GPIO_EXTI_converter.log`. Dit zorgt ervoor dat dit project niet kunt hernoemen of kopiëren zonder dat er fouten ontstaan. Het is wel mogelijk om de BSP API te gebruiken in een ‘echt’ STM32CubeIDE project, zie <https://community.st.com/t5/stm32-mcus/how-to-add-a-bsp-to-an-stm32cubeide-project/ta-p/49812>.

³⁸ <https://www.st.com/en/development-tools/sw4stm32.html>