

— IoT Devices

Hoofdstuk 1: Inleiding

L. Espeel



(bron: grotendeels van R. Buysschaert)

— Even een terugblik ...

- Deze cursus gaat verder op de opgedane kennis uit het vak Microcontrollers in fase 1.
- Wat hebben we daar geleerd?
 - Werken met een Nucleo microcontrollerbord: NUCLEO-F091RC.
 - Programmeren in C-taal.
 - Basis input/output verwerken.
 - Communiceren via UART, SPI en I²C.
 - Gebruik maken van de **CMSIS-bibliotheken**.

— Doelstellingen in dit vak (1)

- Leren werken met de NUCLEO-L476RG.
 - Nog steeds toegang tot UART, SPI, I²C, timers, ADC, DAC, CAN, ...
 - Meer RAM-geheugen, flashgeheugen, kloksnelheden tot 80 MHz, extra ingebouwde controllers (bv. LCD, TSC, ...), ...
- Leren werken met de HAL-bibliotheken.
 - Vorig jaar gewerkt op register niveau met CMSIS-code.
 - Nu Hardware Abstraction Layer bibliotheek gebruiken.

3

— Doelstellingen in dit vak (2)

- De verschillende soorten IO-transfers inoefenen:
 - polling.
 - interrupts.
 - Direct Memory Access (DMA).
- Beveiligde WiFi-communicatie opzetten met een ESP32-C3-module via AT-commando's.
- Leren werken met een Realtime Operating System (RTOS) op een microcontroller.
- Wat dieper ingaan op correct gebruik van de C-taal.

4

— Nodige componenten

- Uitleenbaar in genummerd doosje:
 - Nucleo-L476RG met mini-USB kabel
 - Nucleo Extension Shield: indien het zelfgesoldeerde shield uit het 1e jaar werkt, gebruik dit.
 - Externe modules zoals keypad, SSD, MPU9250, APA102C, ESP32-C3
- Zelf voorzien:
 - F/F en M/F jumper kabels met vierkante connectors (15 à 20 cm lang; ≥ 20 stuks F/F en ≥ 10 stuks M/F)
 - Platte schroevendraaier van 2 mm

5

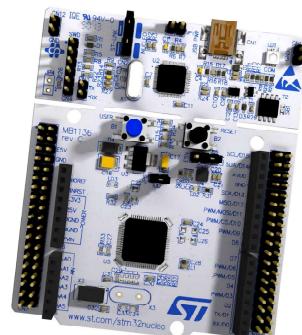


NUCLEO-L476RG

6

— NUCLEO-L476RG

- Ziet er hetzelfde uit als de andere Nucleo 64-borden.
- Kan dus ook uitgebreid worden met het Nucleo Extension Shield (knoppen, LED's, trimmer en 10-pin connector).
- Bezit de STM32L476RG microcontroller.
- Heeft ook een ST-link programmer on board.
- Kan geprogrammeerd worden via Keil µVision met behulp van C-taal.
- Wordt ondersteund door STM32CubeMX en de HAL-bibliotheken.



7

hogeschool
vives

— CMSIS versus HAL

8

hogeschool
vives

— CMSIS versus HAL (1)

- Werken met CMSIS-code is heel goed, omdat er een standaard manier van werken gehanteerd wordt: Cortex Microcontroller Software Interface Standard.
- Heel efficiënt tot op bit-niveau in de registers zaken gaan aanpassen.

```
// Clock voor GPIOA inschakelen.  
RCC->AHB2ENR = RCC->AHB2ENR | RCC_AHB2ENR_GPIOAEN;
```

- Echter: veranderen van type microcontroller is niet zo eenvoudig

...

9

— CMSIS versus HAL (2)

- Indien we vlotter van microcontroller willen veranderen, kan een Hardware Abstraction Layer (HAL) een voordeel bieden.
- De HAL-bibliotheek is een softwarelaag (C-taal) die bovenop de CMSIS-bibliotheken ligt, waardoor je onafhankelijk(er) wordt van het type microcontroller.



- Mogelijk gevaar: het maakt code soms wel minder efficiënt.

10

— CMSIS versus HAL (3)

Codevoorbeeld voor het toggelen van de toestand van een LED.

- CMSIS

```
GPIOC->ODR = GPIOC->ODR ^ GPIO_ODR_OD0;
```

- HAL

```
void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{
    uint32_t odr;
    /* Check the parameters */
    assert_param(IS_GPIO_PIN(GPIO_Pin));
    /* get current Output Data Register value */
    odr = GPIOx->ODR;
    /* Set selected pins that were at low level, and reset ones that were high */
    GPIOx->BSRR = ((odr & GPIO_Pin) << GPIO_NUMBER) | (~odr & GPIO_Pin);
}
```

Later bekijken we in detail wat hier gebeurt,
maar nu zie je al duidelijk dat er meer code nodig
is voor hetzelfde resultaat ...

11



—

Demo:
CMSIS template in Keil µVision

12



CMSIS template (1)

- Doel?
 - Het basiscode voorbeeld **Nucleo template files (no project)** om het Nucleo Extension Shield te testen, programmeren in een NUCLEO-L476RG.
- Hoe?
 - Gebruik Keil µVision om 'vanaf nul' een project op te bouwen.
 - Bekijk de reference manual en de device specifieke datasheet om de nodige registerinformatie op te zoeken.
- Vaststelling?
 - De opgedane kennis van in de module Microcontrollers, kan je rechtstreeks gebruiken om hier tot een goed resultaat te komen.

13

CMSIS template (2)

The screenshot shows the Keil µVision IDE interface with a project titled 'Nucleo-L476RG'. The project structure includes a 'Sources' folder containing 'main.c', 'buttons.c', 'leds.c', 'uart2.c', and 'adc.c'. A 'Headers' folder contains 'buttons.h', 'leds.h', 'uart2.h', 'adc.h', and 'main.h'. The 'CMSIS' and 'Device' folders also contain various header files. The main.c file is open in the editor, displaying the following code:

```
// Basiscode voor het starten van eenieder welk project op een Nucleo-L476RG met
// OPM: via 'Options for Target -> C/C++' zet je de compiler op C11, optimisaties
// Versie: 20230615
// Includes.
#include "stm32l476xx.h"
#include "stdio.h"
#include "stdbool.h"
#include "leds.h"
#include "buttons.h"
#include "uart2.h"
#include "adc.h"
#include "main.h"
// Functie prototypes.
void SystemClock_Config(void);
void InitIO(void);
void WaitForMs(uint32_t timespan);
// Variabelen aanmaken.
// OPM: het keyword 'static', zorgt ervoor dat de variabele enkel binnen dit bestand
// beschikbaar is.
static uint8_t count = 0;
```

The status bar at the bottom indicates 'Verify OK.', 'Application running ...', and 'Flash Load finished at 13:15:26'. The 'ST-Link Debugger' tab is visible at the bottom right.

1. Maak een nieuw project aan in µVision voor de **STM32L476RG** chip (met CMSIS Core en startup).
1. Voeg alle C- en H-bestanden toe (**Nucleo template files (no project)**).
2. Pas de projectinstellingen aan (ST-link, compiler, ...).
3. Compileer, test en kijk wat rond (in het project).

14

Demo: HAL template met STM32CubeMX en Keil µVision

15



— **HAL template (1)**

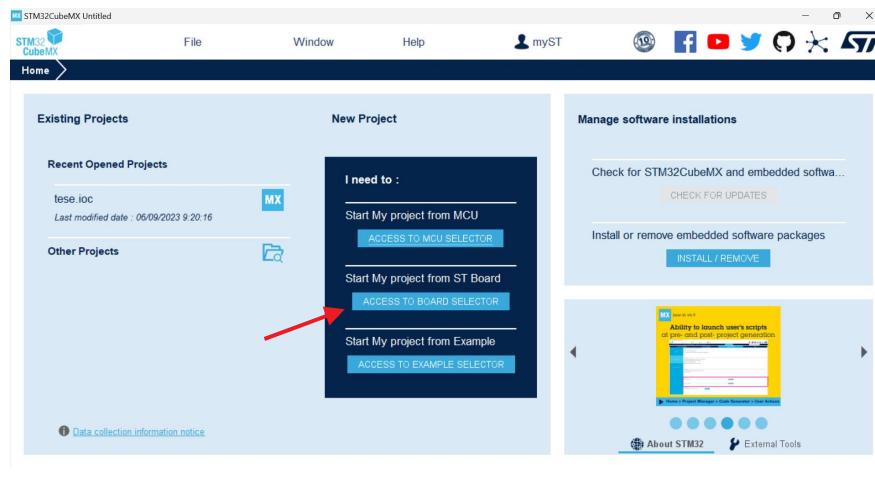
- Doel?
 - Hetzelfde basiscodevoorbeeld om het Nucleo Extension Shield te testen, programmeren in een NUCLEO-L476RG.
- Hoe?
 - Gebruik STM32CubeMX om een Keil µVision project aan te maken.
 - Zet daarbij de juiste instellingen om bijvoorbeeld de LED's en knoppen als uitgang en ingang te definiëren.
- Vaststelling?
 - We kunnen dus op een 'grafische' manier, ook de microcontroller instellen.

16



HAL template (2)

Start STM32CubeMX op en kies 'Start My project from ST Board'.

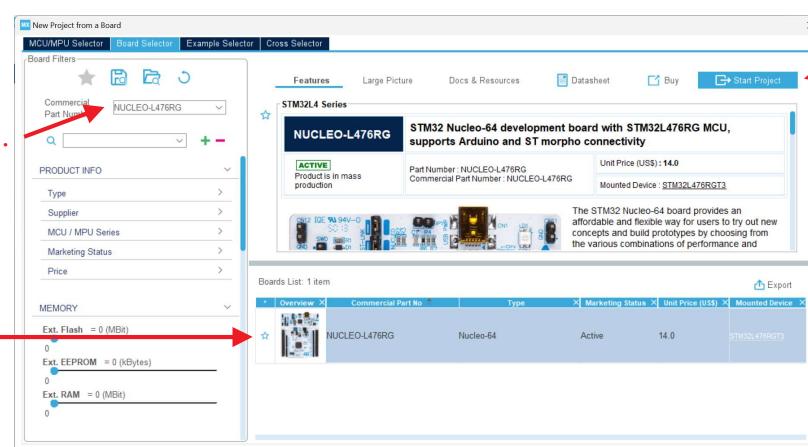


17

hogeschool
VIVES

HAL template (3)

Selecteer onze Nucleo-versie en klik op 'Start Project'.

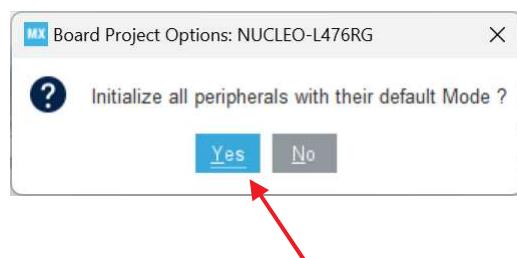


18

hogeschool
VIVES

— HAL template (4)

Initialiseer de randapparatuur met de standaardinstellingen.



19

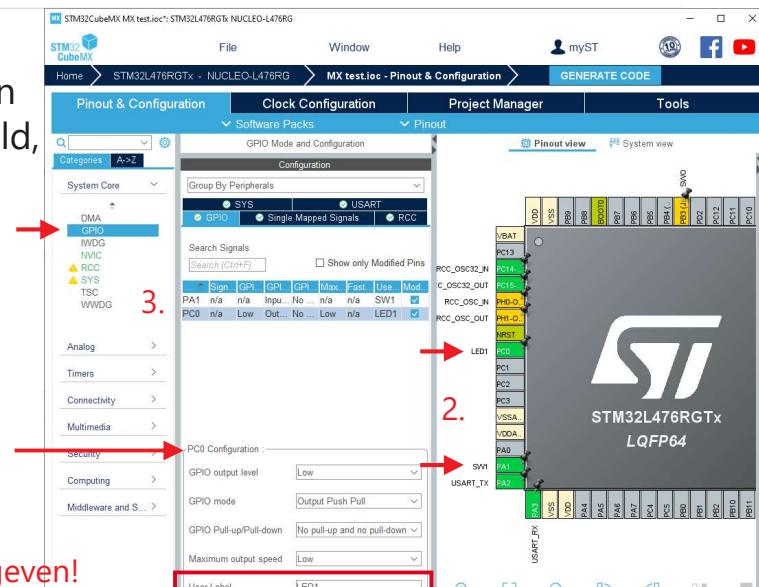
hogeschool
VIVES

— HAL template (5)

Zoek op in het schema van het Nucleo Extension Shield, waar LED1 en SW1 mee verbonden zijn.
Stel die twee IO's in zoals bedoeld.

1. →
3. →
2. →
4. →

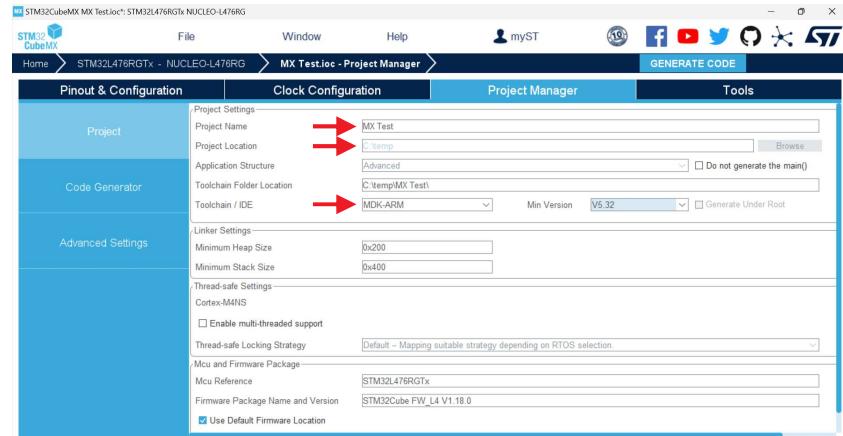
Eigen naam ingeven!
(bv. SW1, LED1, ...)



20

— HAL template (6)

Ga naar het tabblad 'Project Manager' en bepaal daar de projectnaam, -locatie en de IDE waarmee je wil werken (μ Vision = **MDK-ARM**).

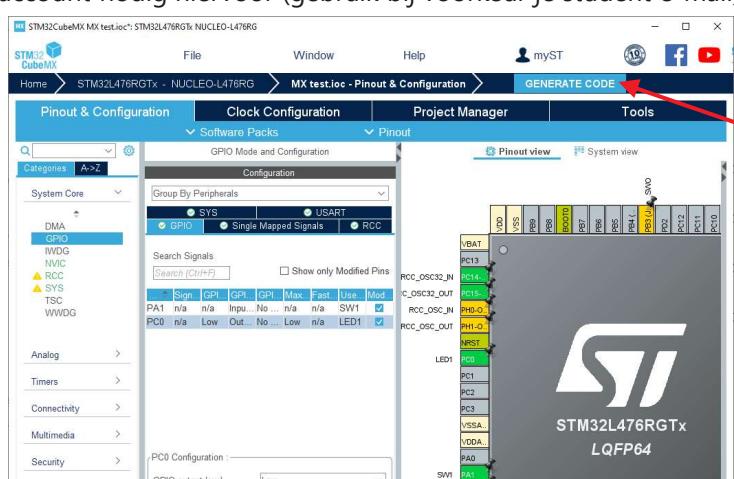


21

hogeschool
VIVES

— HAL template (7)

Klik op 'Generate Code' om een Keil μ Vision project aan te maken.
Opmerking: ST-account nodig hiervoor (gebruik bij voorkeur je student e-mail).



— HAL template (8)

Als de code succesvol is aangemaakt, klik dan op 'Open Project'.
Als alles goed gaat, opent nu Keil µVision.

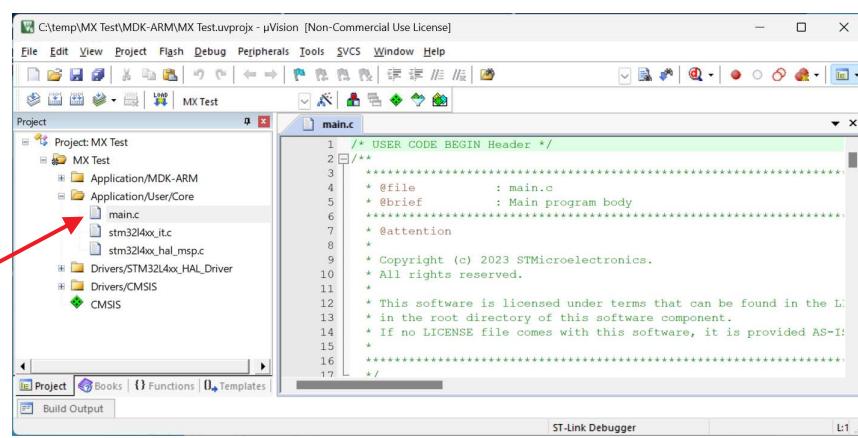


23

hogeschool
VIVES

— HAL template (9)

Het µVision project heeft een mappenstructuur zoals hieronder te zien. Open het *main.c* bestand en snuister eens rond.



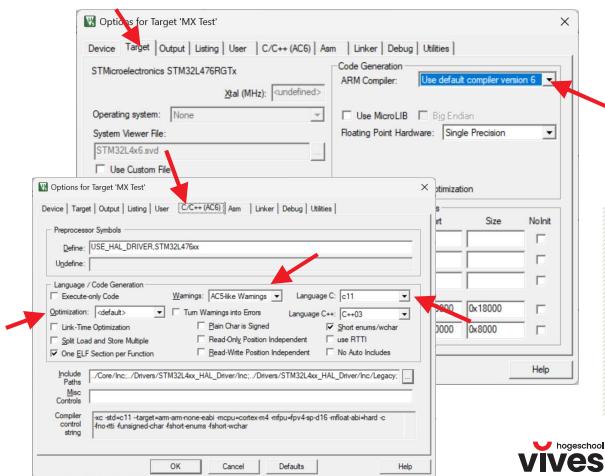
24

hogeschool
VIVES

HAL template (10)

Probeer het project te compileren. Lukt dat niet, dan moet je wellicht nog enkele instellingen voor de compiler goed zetten. Denk aan:

- User default compiler version 6.
- AC5-like warnings (optioneel).
- Optimization: default of O3.
- Language C: C11.
- Controleer dit even onder tab *Debug*:
 - ST-link Debugger.
 - Knop *Settings*: Connect under reset.
- Tab *Flash Download*: Reset and Run.
- ...



25

VIVES

HAL template (11)

De geladen code heeft nu nog geen zichtbare werking. Pas de *main.c* aan zodat je onderstaande krijgt. Wat verwacht je?

```
95  /* Infinite loop */
96  /* USER CODE BEGIN WHILE */
97  while (1)
98  {
99      if(HAL_GPIO_ReadPin(SW1_GPIO_Port, SW1_Pin) == GPIO_PIN_RESET)
100         HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_SET);
101     else
102     {
103         HAL_Delay(500);
104         HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin);
105     }
106
107    /* USER CODE END WHILE */
108
109    /* USER CODE BEGIN 3 */
110 }
111 /* USER CODE END 3 */
112 }
```

Merk op dat je slechts in bepaalde gedeeltes van de code zelf aanpassingen mag doen. De gedeeltes erbij kunnen worden automatisch gewist wanneer je STM32CubeMX het project opnieuw laat genereren!!!

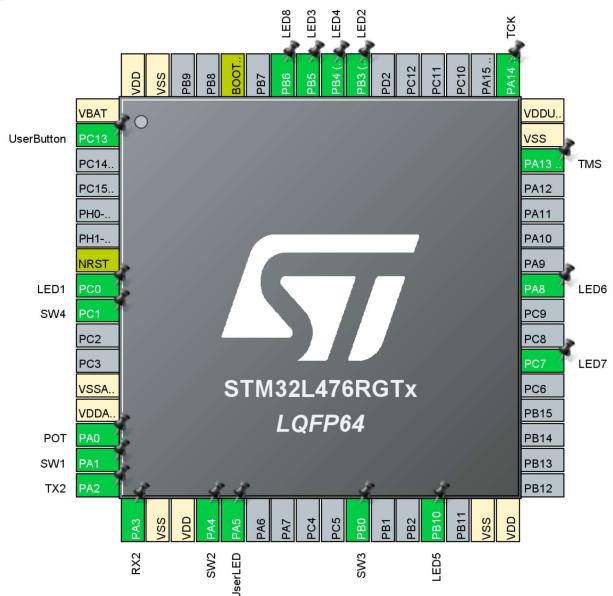
26

VIVES

HAL template (12)

De voorgaande code kan slechts één LED aansturen en één knop uitlezen. Pas nu alles verder aan in STM32CubeMX om de **8 LED's, 4 knoppen, User LED** en **User Button** te kunnen gebruiken.

Optioneel kan je ook de AD-converter en UART2 instellen.



HAL template (13)

Alle pinnen instellen kan een heel werkje zijn, daarom wordt ook het sjabloon **Nucleo template** ter beschikking gesteld.

Om de LED's aan te sturen en knoppen uit te lezen, werd ook reeds een *leds.c* en *buttons.c* bestand aangemaakt.

Alle bestanden samen vormen het sjabloon. Tijd om dat sjabloon samen te overlopen... Bekijk zowel STM32CubeMX als µVision.



—

HAL

29



—

HAL - info

- Waar kan ik info vinden over de HAL-bibliotheek?
 - Er is een goed uitgewerkte PDF met daarin de essentie.
 - https://www.st.com/resource/en/user_manual/um1884-description-of-stm32l4l4-hal-and-lowlayer-drivers-stmicroelectronics.pdf
(of PDF-document staat ook op Toledo)
- Gebruik een goede PDF-lezer om te navigeren via de bladwijzers. Per 'peripheral' is er een hoofdstuk.

 **UM1884**
User manual
Description of STM32L4/L4+ HAL and low-layer drivers

30



— HAL – Delay

Navigeer in de PDF naar de functie *HAL_Delay()* en zoek op wat die doet, welke parameters nodig zijn, wat de return value is, ...

HAL_Delay

Function name

void HAL_Delay (uint32_t Delay)

Function description

This function provides minimum delay (in milliseconds) based on variable incremented.

Parameters

- **Delay:** specifies the delay time length, in milliseconds.

Return values

- **None:**

Notes

- In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals where uwTick is incremented.
- This function is declared as `__weak` to be overwritten in case of other implementations in user file.

31



— HAL – Structs

De HAL-bibliotheek maakt veelvuldig gebruik van structs en pointers naar structs om de code te structureren.

- Weet je nog wat een struct is?
- Weet je nog wat een pointer is?
- Bekijk de *UART_HandleTypeDefDef* in een µVision project.

32



— HAL – Hetzelfde maar anders...

Wanneer we de HAL-bibliotheken gebruiken, kan je dikwijls op verschillende manieren je doel bereiken.

Je kan bijvoorbeeld de UART Enable bit uit het CR1-register op (minstens) 3 manieren zetten, via:

1. `MX_USART2_UART_Init();`
2. `__HAL_UART_ENABLE(huart);`
3. `huart2.Instance->CR1 |= USART_CR1_UE;`

Dit gebeurt ook zo in het project. Loop eens doorheen de code om dat uit te zoeken.

— Fouten opvangen – assert_param()

Je wilt uiteraard code schrijven die zo robuust mogelijk is (weinig fouten bevat).

Hier en daar gebruikt men daarvoor de *assert_param()* macro.

Deze macro/preprocessor directive, controleert of de parameters van een bepaald type zijn.

```
void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{
    uint32_t odr;
    /* Check the parameters */
    assert_param(IS_GPIO_PIN(GPIO_Pin));
    /* get current Output Data Register value */
    odr = GPIOx->ODR;
    /* Set selected pins that were at low level, and reset ones that were high */
    GPIOx->BSRR = ((odr & GPIO_Pin) << GPIO_NUMBER) | (~odr & GPIO_Pin);
}
```

MAAR: zolang de USE_FULL_ASSERT niet gedefinieerd wordt in het 'stm32l4xx_hal_conf.h' bestand, heeft de macro geen echte functie.

Voorlopig niet van belang dus ...

— Fouten opvangen – Error_Handler()

Een andere manier om fouten te ontdekken is het gebruik van de functie *Error_Handler()* in de HAL-bibliotheken.

Als een HAL-actie terugkeert met iets anders dan *HAL_OK*, wordt die functie (soms) opgeroepen...

```
352 |     * @brief This function is executed in case of error occurrence.  
353 |     * @retval None  
354 | */  
355 void Error_Handler(void) ←  
356{  
357     /* USER CODE BEGIN Error_Handler_Debug */  
358     /* User can add his own implementation to report the HAL error return state */  
359     __disable_irq();  
360     while (1)  
361     {  
362     }  
363     /* USER CODE END Error_Handler_Debug */  
364 }
```

MAAR: ook hier kan je zelf nuttige acties koppelen aan die foutmelding. Op dit moment schakelen we de interrupts uit en blijven we in een oneindig lange lus!



35

C-taal

36



C-taal – C-string

- Een 'tekst' of 'string' in C-taal wordt bewaard in een array van karakters.
- Om aan te geven dat de tekst stopt, wordt er op het einde een 'nulkarakter' toegevoegd ('\0').

V | I | V | E | S | \0

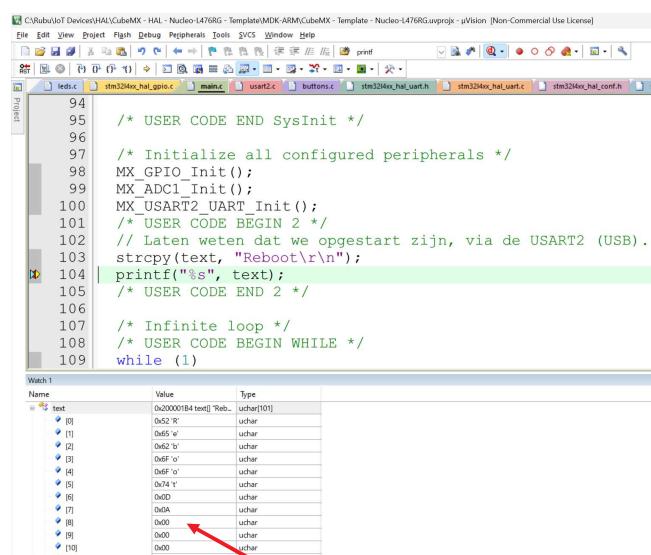
- Je kan dat nulkarakter gebruiken om bijvoorbeeld te stoppen met verzenden van karakters over de seriële poort.
- Een variabele type 'string' zoals in bijvoorbeeld C# (en C++) bestaat niet echt.

37

hogeschool
vives

C-taal – C-string

- Bekijk een voorbeeld waarbij je de tekst "Reboot\r\n" wil versturen via USART2.
- Zet een break point en bekijk het einde stringteken.
- Zoek ook de functie *fputc()* en bekijk hoe ze werkt...



```
94  /* USER CODE END SysInit */
95
96
97  /* Initialize all configured peripherals */
98  MX_GPIO_Init();
99  MX_ADC1_Init();
100 MX_USART2_UART_Init();
101 /* USER CODE BEGIN 2 */
102 // Laten weten dat we opgestart zijn, via de USART2 (USB).
103 sprintf(text, "Reboot\r\n");
104 printf("%s", text);
105 /* USER CODE END 2 */
106
107 /* Infinite loop */
108 /* USER CODE BEGIN WHILE */
109 while (1)
```

| Name | Type | Value |
|------|-----------|--------------------------------|
| text | uchar[10] | 0x200001B4 test]] 'Reboot\r\n' |
| [0] | uchar | 0x52 'R' |
| [1] | uchar | 0x65 'e' |
| [2] | uchar | 0x62 'b' |
| [3] | uchar | 0x6F 'o' |
| [4] | uchar | 0x6F 'o' |
| [5] | uchar | 0x74 't' |
| [6] | uchar | 0x0D |
| [7] | uchar | 0x0A |
| [8] | uchar | 0x00 |
| [9] | uchar | 0x00 |
| [10] | uchar | 0x00 |

38

hogeschool
vives

— C-taal – C-string

Merk op dat we in het vorig deel 'Microcontrollers', een andere functie gebruikten. In de plaats van *printf()* hadden we toen *StringToUsart2()*...

```
46 void StringToUsart2(char* string)
47 {
48     uint8_t indexer = 0;
49
50     while(string[indexer] != 0)
51     {
52         // Byte versturen.
53         USART2->TDR = (uint8_t)string[indexer++];
54
55         // Wachten tot byte vertuurd is.
56         while((USART2->ISR & USART_ISR_TC) != USART_ISR_TC);
57     }
58 }
```

39



— C-taal – structs

Bekijk het codevoorbeeld **Pointers** en zoek uit hoe alles werkt...

```
33 /* Private typedef -----*/
34 /* USER CODE BEGIN PTD */
35 // Maak een struct die de toestand van het Nucleo Extension Shield bijhoudt.
36 // Die struct zou je later kunnen gebruiken om bijvoorbeeld in een C#-app
37 // real time de NES-status weer te geven...
38 typedef struct{
39     uint8_t leds;
40     uint8_t buttons;
41     uint16_t adValue;
42 }NES;
43 /* USER CODE END PTD */
```

40



C-taal – pointers

```
65 // Maak een 8-bit unsigned integer.  
66 static uint8_t count = 0;  
67 // Maak een pointer (= plaats voor een verwijzing) naar een 8-bit unsigned integer.  
68 static uint8_t* countPointer;  
69  
70 // Maak een struct aan van het type NES.  
71 static NES nes;  
72 // Maak een pointer (= plaats voor een verwijzing) naar een struct van het type NES.  
73 static NES* nesPointer;  
74  
75 // Maak een array met plaats voor karakters. Merk op: als je enkel de naam van het array gebruikt,  
76 // kan je die bekijken als een pointer naar het begin van dat array...  
77 static char text[10];  
78 // Maak een pointer naar een karakter.  
79 static char* textPointer;  
80 // Maak plaats voor één karakter.  
81 static char character;  
82  
83 // Maak een pointer aan naar een void. Daarin kan je verwijzen naar eender welk type...  
84 static void* voidPointer;  
85 // Maak variabele aan die een AD-waarde kan bevatten.  
86 static uint16_t voidPointerAdValue;
```

41



C-taal – pointers naar structs

```
398 void UpdateNesState(NES* nes)  
399 {  
400     // De toestand van de LED's opvragen en opslaan in de struct.  
401     nes->leds = 0;  
402     if(HAL_GPIO_ReadPin(LED1_GPIO_Port, LED1_Pin) == GPIO_PIN_SET)  
403         nes->leds |= 0x01;  
404     if(HAL_GPIO_ReadPin(LED2_GPIO_Port, LED2_Pin) == GPIO_PIN_SET)  
405         nes->leds |= 0x02;  
406     if(HAL_GPIO_ReadPin(LED3_GPIO_Port, LED3_Pin) == GPIO_PIN_SET)  
407         nes->leds |= 0x04;  
408     if(HAL_GPIO_ReadPin(LED4_GPIO_Port, LED4_Pin) == GPIO_PIN_SET)  
409         nes->leds |= 0x08;  
410     if(HAL_GPIO_ReadPin(LED5_GPIO_Port, LED5_Pin) == GPIO_PIN_SET)  
411         nes->leds |= 0x10;  
412     if(HAL_GPIO_ReadPin(LED6_GPIO_Port, LED6_Pin) == GPIO_PIN_SET)  
413         nes->leds |= 0x20;  
414     if(HAL_GPIO_ReadPin(LED7_GPIO_Port, LED7_Pin) == GPIO_PIN_SET)  
415         nes->leds |= 0x40;  
416     if(HAL_GPIO_ReadPin(LED8_GPIO_Port, LED8_Pin) == GPIO_PIN_SET)  
417         nes->leds |= 0x80;
```

42



— C-taal – void pointers

```
// Deel 4: pointer naar een void.  
// Je kan verwijzen naar gelijk wat als je een void pointer gebruikt...  
// Bekijk de inhoud van voidPointer na onderstaand statement. Je zal er het adres  
// van 'nes' in terugvinden.  
voidPointer = &nes;  
// Na correct casten, kan je ook de velden van de struct aanspreken.  
voidPointerAdValue = ((NES*)voidPointer)->adValue;
```

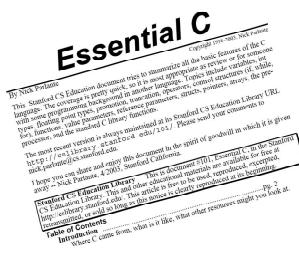
43



— C-taal – extra info

Het spreekt voor zich dat er nog veel meer te vertellen valt over de C-taal.

Wil je meer info, bekijk dan zeker de PDF op Toledo met als titel: "Essential C". Geschreven door Nick Parlante.



44



— IoT Devices

Hoofdstuk 2: Basis IO met HAL-bibliotheek

L. Espeel



(bron: grotendeels van R. Buysschaert)

—

HAL demo:
één knop – één LED

— HAL demo: één knop – één LED (1)

- Start van het sjabloon **Nucleo template** om één drukknop in te lezen en daarmee één LED aan te sturen...
- Open het STM32CubeMX-bestand en blader door de belangrijkste instellingen (System Core en Clock Configuration).
- Open het Keil µVision project en stel de firmware op... Maak twee versies:
 - met gebruik van de gekende functies '*SW1Active()*'.
 - met gebruik van de HAL-functies '*HAL_GPIO_ReadPin()*'.

3

— HAL demo: één knop – één LED (2)

```
111 //    // Versie 1: met gebruik van de gekende functies (die de HAL-functies 'wrappen').  
112 //    if(SW1Active())  
113 //        ByteToLeds(0x01);  
114 //    else  
115 //        ByteToLeds(0x00);  
116  
117 // Versie 2: met direct gebruik van de HAL-functies.  
118 if(HAL_GPIO_ReadPin(SW1_GPIO_Port, SW1_Pin) == GPIO_PIN_RESET)  
119     HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_SET);  
120 else  
121     HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);  
122  
123 // OPM: de tweede versie manipuleert enkel één LED, terwijl de eerste versie alle LED's instelt...  
124 //      Versie één kan echter ook op die manier uitgewerkt worden indien gewenst.
```

Als je dus in STM32CubeMX de LED's en knoppen verplaatst/aanpast, kan je vrij snel je project klaar maken voor andere hardware (of microcontroller)... Zo iets heet men soms 'migreren'.

4

— HAL demo: één knop – één LED (3)

Navigeer doorheen het Keil µVision project en zoek uit wat er allemaal gebeurt tijdens het opstarten...

Neem daarvoor je tijd! In principe moet ieder lijn code duidelijk zijn.

```
97  /* Initialize all configured peripherals */
98  MX_GPIO_Init();
99  MX_ADC1_Init();
100 MX_USART2_UART_Init();
101 /* USER CODE BEGIN 2 */
102 // Laten weten dat we opgestart zijn, via de USART2 (USB).
103 strcpy(text, "Reboot\r\n");
104 printf("%s", text);
105 /* USER CODE END 2 */
```

5



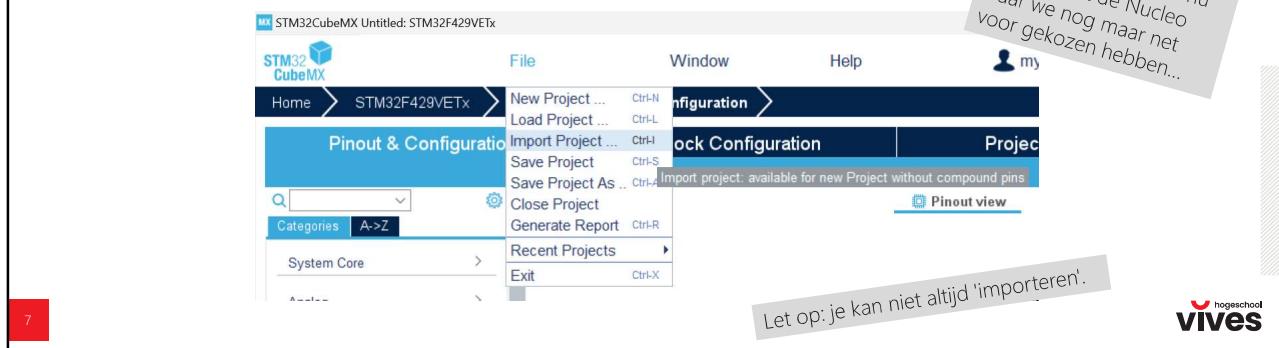
Migreren

6



Migreren

- Wil je effectief van de ene naar de andere microcontroller migreren, dan kan je dat met behulp van STM32CubeMX grotendeels automatiseren.
- Gebruik daarvoor de 'import project' optie...



7

HAL demo:
AD-converter en UART

8

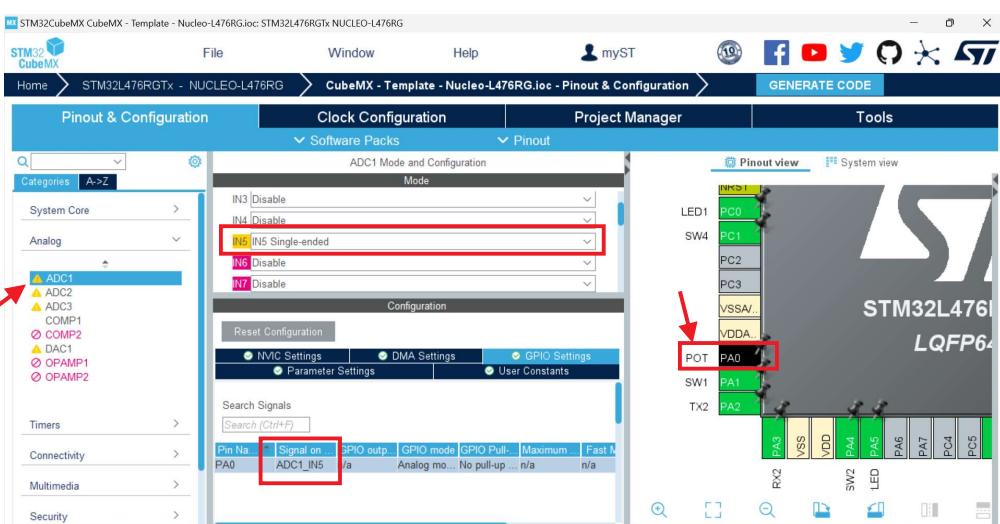
HAL demo: AD-converter en UART2⁽¹⁾

- Start van het sjabloon **Nucleo template** om de AD-waarde van de on board potentiometer in te lezen en de waarde op de LED's te zetten (via niveau aanduiding).
- Focus hierbij op de instellingen van de AD-converter in STM32CubeMX...

9

HAL demo: AD-converter en UART2⁽²⁾

10



— HAL demo: AD-converter en USART2⁽³⁾

- Pas de firmware aan zoals hieronder. Zoek uit hoe alles werkt!

```
106 |     while (1)
107 | {
108 |     // AD-conversie starten en in een blokkerende functie op het
109 |     // resultaat wachten.
110 |     adValue = GetAdValue(&hadcl);
111 |     adValue = adValue >> 4;    // 12-bit waarde die right aligned was, naar
112 |                             // 8-bit converteren.
113 |
114 |     // Waarde display'n
115 |     ByteToLevel((uint8_t)adValue);
116 |
117 |     // Waarde naar USART2 sturen (USB COM-poort).
118 |     printf("AD-value: %d.\r\n", adValue);
119 |
120 |     // Even wachten
121 |     HAL_Delay(25);
```

Merk op dat de printf omgeleid werd (redirect) naar USART2. Dat kan je zien in 'usart2.c'.

11



— HAL demo: AD-converter en USART2⁽⁴⁾

- printf()* wordt omgeleid naar *HAL_UART_Transmit()*, waarin het TDR-register opgevuld wordt met data. Daardoor wordt die byte verzonden over USART2. Zoek onderstaande code op...

```
1187 |     while (huart->TxXferCount > 0U)
1188 |     {
1189 |         if (UART_WaitOnFlagUntilTimeout(huart, UART_FLAG_TXE, RESET))
1190 |         {
1191 |             return HAL_TIMEOUT;
1192 |         }
1193 |         if (pdata8bits == NULL)
1194 |         {
1195 |             huart->Instance->TDR = (uint16_t)(*pdata16bits & 0x01FFU);
1196 |             pdata16bits++;
1197 |         }
1198 |         else
1199 |         {
1200 |             huart->Instance->TDR = (uint8_t)(*pdata8bits & 0xFFU);
1201 |             pdata8bits++;
1202 |         }
1203 |         huart->TxXferCount--;
1204 |     }
```

12



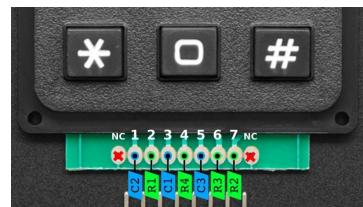
HAL demo: keypad inlezen

13

hogeschool
vives

— **HAL demo: keypad inlezen (1)**

- Lees een keypad in van 3 x 4 knoppen.

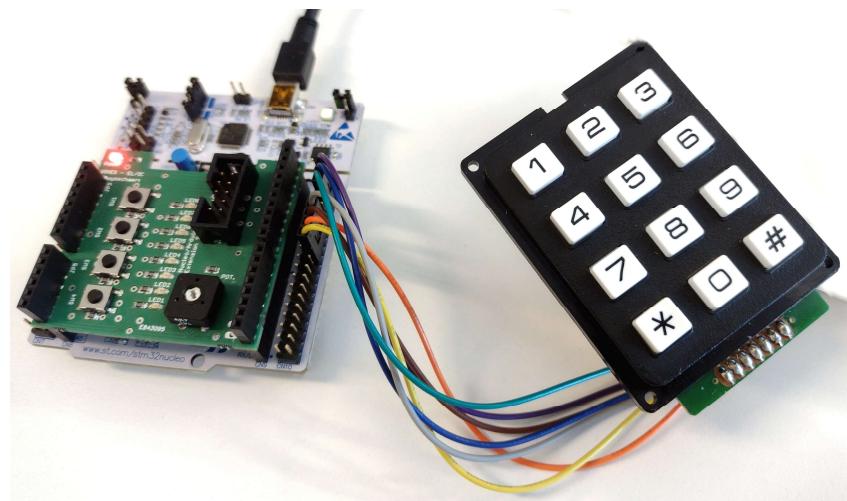


Afbeeldingen: <https://learn.adafruit.com/matrix-keypad/pinouts>

14

hogeschool
vives

— HAL demo: keypad inlezen (2)

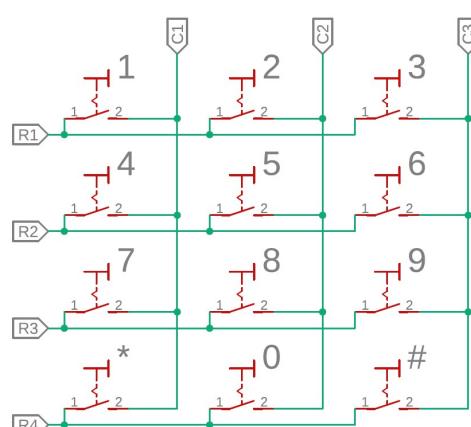


15

hogeschool
vives

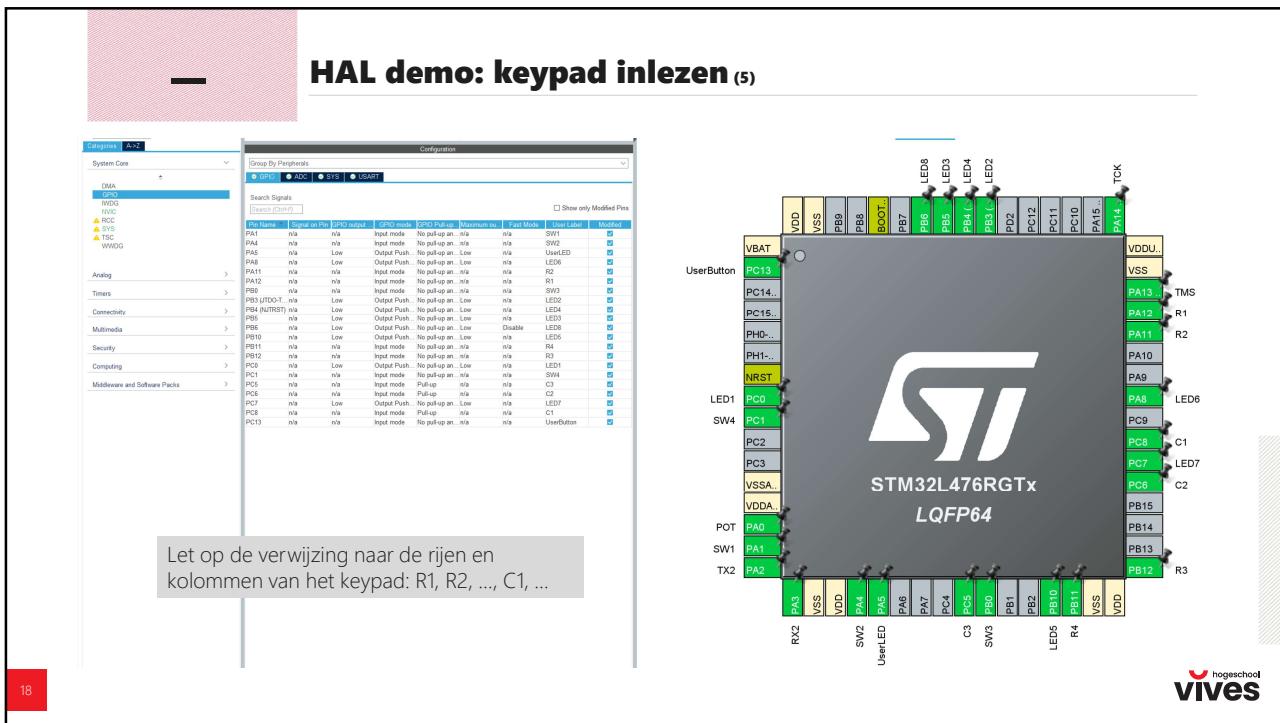
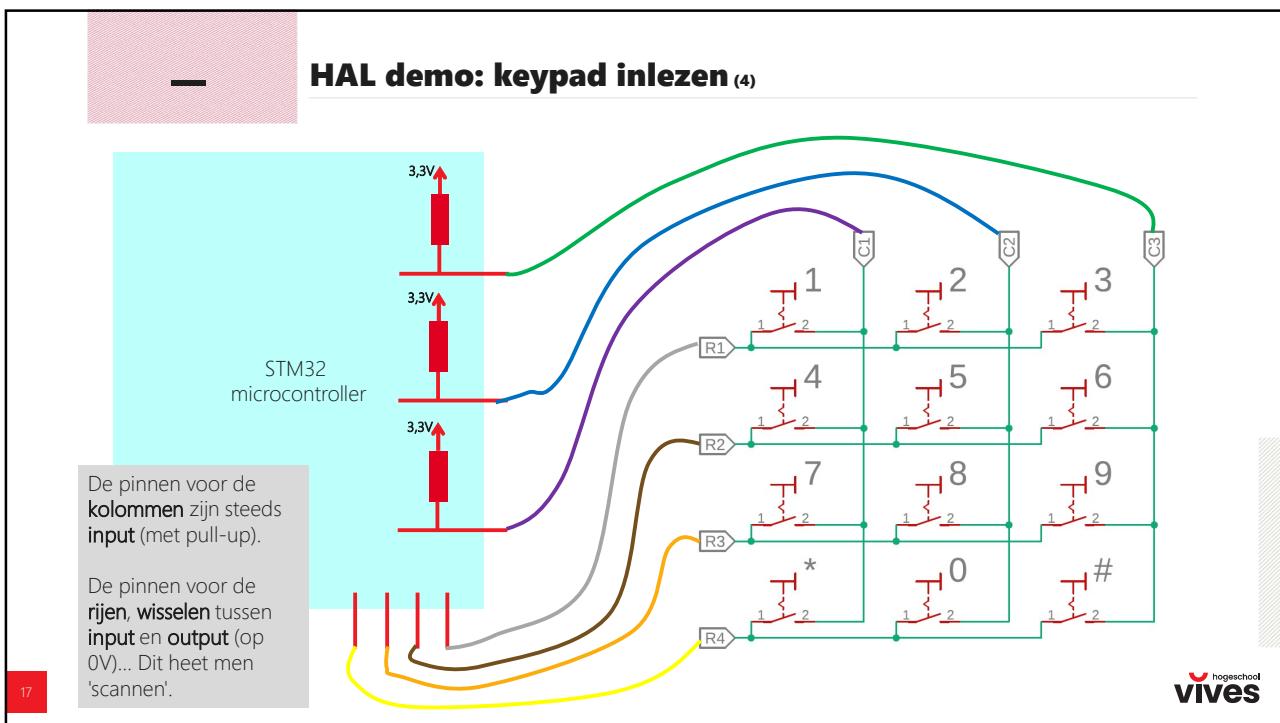
— HAL demo: keypad inlezen (3)

- Het interne schema ziet er zo uit...
- Waarom werkt men zo?
- Hoe zou je dat efficiënt kunnen gebruiken?
=> 'scanning'



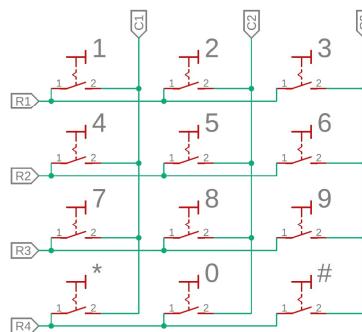
16

hogeschool
vives



— HAL demo: keypad inlezen (6)

Indien je wil werken met het codevoorbeeld uit de les, maak dan onderstaande verbindingen.



Connection suggestion to Nucleo board:
C1 => PC8
C2 => PC6
C3 => PC5
R1 => PA12
R2 => PA11
R3 => PB12
R4 => PB11

19

vives hogeschool

— HAL demo: keypad inlezen (7)

- Bekijk de 'scan'-werking.
- Van welk type is GPIO_ReInit?

```
26 // Rij 1 als output zetten en laag maken.  
27 GPIO_ReInit.Pin = R1_Pin;  
28 GPIO_ReInit.Mode = GPIO_MODE_OUTPUT_PP;  
29 GPIO_ReInit.Pull = GPIO_NOPULL;  
30 GPIO_ReInit.Speed = GPIO_SPEED_FREQ_LOW;  
31 HAL_GPIO_Init(R1_GPIO_Port, &GPIO_ReInit);  
32 HAL_GPIO_WritePin(R1_GPIO_Port, R1_Pin, GPIO_PIN_RESET);  
33  
34 // Knop 1 inlezen.  
35 if(HAL_GPIO_ReadPin(C1_GPIO_Port, C1_Pin) == GPIO_PIN_RESET)  
36 keys |= 0x0001;  
37  
38 // Knop 2 inlezen.  
39 if(HAL_GPIO_ReadPin(C2_GPIO_Port, C2_Pin) == GPIO_PIN_RESET)  
40 keys |= 0x0002;  
41  
42 // Knop 3 inlezen.  
43 if(HAL_GPIO_ReadPin(C3_GPIO_Port, C3_Pin) == GPIO_PIN_RESET)  
44 keys |= 0x0004;
```

20

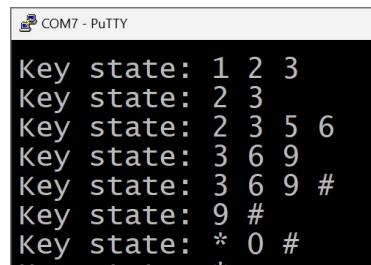
vives hogeschool

— HAL demo: keypad inlezen (8)

Sjabloon **Polling - Keypad template** is beschikbaar met code om het keypad in te lezen.

```
// Toestand van de knoppen opvragen. Eén bit per knop. Dus 12 bit nodig...
keys = GetKeys();

// Vul hier jouw eigen code verder aan...
```



The screenshot shows a terminal window titled "COM7 - PuTTY". It displays a series of lines of text, each starting with "Key state: ". The output is as follows:

```
Key state: 1 2 3
Key state: 2 3
Key state: 2 3 5 6
Key state: 3 6 9
Key state: 3 6 9 #
Key state: 9 #
Key state: * 0 #
```

—

HAL demo: 7-segment display aansturen

— HAL demo: 7-segment display aansturen (1)

- Stuur twee 7-segment displays aan.



Features include:

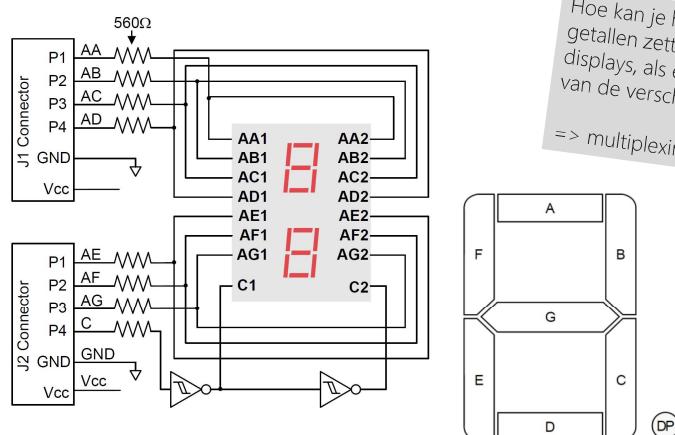
- Two-digit high brightness seven-segment display
- Easily view a counter or timer
- Common Cathode configuration
- Small PCB size for flexible designs 1.0" x 1.7" (2.5 cm x 4.3 cm)
- Two 6-pin Pmod connectors with GPIO interfaces
- Follows [Digilent Pmod Interface Specification](#) Type 1

Afbeelding: <https://digilent.com/reference/pmod/pmodssd/reference-manual>

23

vives hogeschool

— HAL demo: 7-segment display aansturen (2)

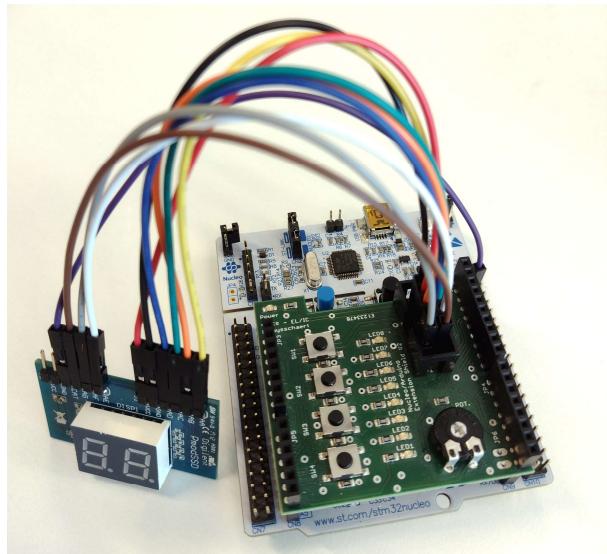


Afbeelding: <https://digilent.com/reference/pmod/pmodssd/reference-manual>

24

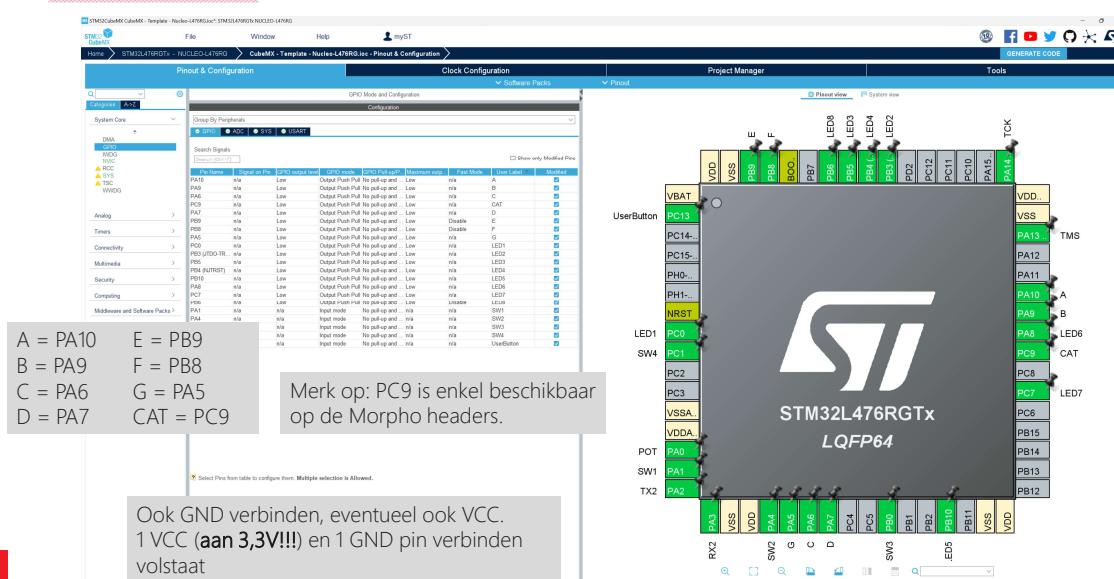
vives hogeschool

HAL demo: 7-segment display aansturen (3)



 hogeschool
VIVES

HAL demo: 7-segment display aansturen (4)



— HAL demo: 7-segment display aansturen (5)

- Bekijk hoe de 'multiplexing' tot stand komt in de code.

```
109 |     while (1)
110 | {
111 |     // Aansturen van het linker display.
112 |     SetSegments((count >> 4), left);
113 |     HAL_Delay(10);
114 |
115 |     // Aansturen van het rechter display.
116 |     SetSegments((count & 0x0F), right);
117 |     HAL_Delay(10);
```

27



— HAL demo: 7-segment display aansturen (6)

- Bekijk hoe de tekens gevormd worden in de code.

```
9 | // Segmenten aansturen.
10| switch(data)
11| {
12|     case 0:
13|         HAL_GPIO_WritePin(A_GPIO_Port, A_Pin, GPIO_PIN_SET);
14|         HAL_GPIO_WritePin(B_GPIO_Port, B_Pin, GPIO_PIN_SET);
15|         HAL_GPIO_WritePin(C_GPIO_Port, C_Pin, GPIO_PIN_SET);
16|         HAL_GPIO_WritePin(D_GPIO_Port, D_Pin, GPIO_PIN_SET);
17|         HAL_GPIO_WritePin(E_GPIO_Port, E_Pin, GPIO_PIN_SET);
18|         HAL_GPIO_WritePin(F_GPIO_Port, F_Pin, GPIO_PIN_SET);
19|         HAL_GPIO_WritePin(G_GPIO_Port, G_Pin, GPIO_PIN_RESET);
20|         break;
21|
22|     case 1:
23|         HAL_GPIO_WritePin(A_GPIO_Port, A_Pin, GPIO_PIN_RESET);
24|         HAL_GPIO_WritePin(B_GPIO_Port, B_Pin, GPIO_PIN_SET);
25|         HAL_GPIO_WritePin(C_GPIO_Port, C_Pin, GPIO_PIN_SET);
26|         HAL_GPIO_WritePin(D_GPIO_Port, D_Pin, GPIO_PIN_RESET);
27|         HAL_GPIO_WritePin(E_GPIO_Port, E_Pin, GPIO_PIN_RESET);
28|         HAL_GPIO_WritePin(F_GPIO_Port, F_Pin, GPIO_PIN_RESET);
29|         HAL_GPIO_WritePin(G_GPIO_Port, G_Pin, GPIO_PIN_RESET);
30|         break;
```

28



— HAL demo: 7-segment display aansturen (7)

- Bekijk hoe de tekens gevormd worden in de code.

```
173 // Linker of rechter display aansturen. Of alles uitdoven.  
174 if(display == left)  
175     HAL_GPIO_WritePin(CAT_GPIO_Port, CAT_Pin, GPIO_PIN_SET);  
176 else  
177     if(display == right)  
178         HAL_GPIO_WritePin(CAT_GPIO_Port, CAT_Pin, GPIO_PIN_RESET);  
179 else  
180 {  
181     HAL_GPIO_WritePin(A_GPIO_Port, A_Pin, GPIO_PIN_RESET);  
182     HAL_GPIO_WritePin(B_GPIO_Port, B_Pin, GPIO_PIN_RESET);  
183     HAL_GPIO_WritePin(C_GPIO_Port, C_Pin, GPIO_PIN_RESET);  
184     HAL_GPIO_WritePin(D_GPIO_Port, D_Pin, GPIO_PIN_RESET);  
185     HAL_GPIO_WritePin(E_GPIO_Port, E_Pin, GPIO_PIN_RESET);  
186     HAL_GPIO_WritePin(F_GPIO_Port, F_Pin, GPIO_PIN_RESET);  
187     HAL_GPIO_WritePin(G_GPIO_Port, G_Pin, GPIO_PIN_RESET);  
188 }
```

29



— HAL demo: 7-segment display aansturen (8)

Sjabloon **PmodSSD template** is beschikbaar met code om de 7-segment displays aan te sturen.

30



— IoT Devices

Hoofdstuk 3: Soorten IO-transfers

L. Espeel



1

(bron: grotendeels van R. Buysschaert)

— IO-transfers?

- Wanneer we willen communiceren met de buitenwereld (UART, SPI, I²C, knoppen, ...), kunnen we dat op drie manieren doen:
 - via **polling**.
 - via **interrupts**.
 - via **Direct Memory Access (DMA)**.

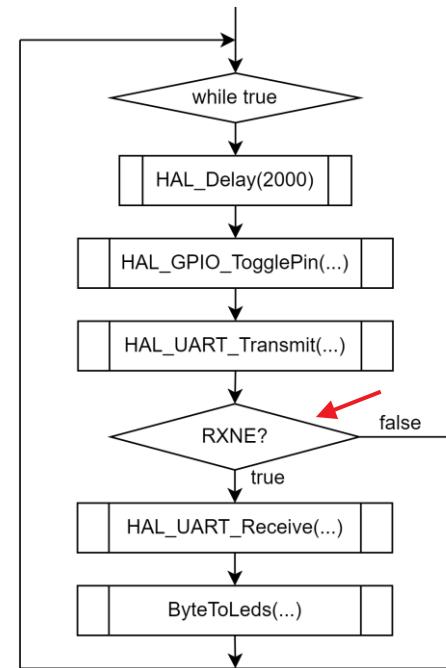
2



Polling

3

- **Polling** = de CPU moet **telkens opnieuw navragen** hoe het gesteld is met ...



4

Polling (2)

```
106 while (1)
107 {
108     // Even wachten.
109     HAL_Delay(2000); ←
110
111     // User LED toggle'n.
112     HAL_GPIO_TogglePin(UserLED_GPIO_Port, UserLED_Pin);
113
114     // Teken van leven geven.
115     HAL_UART_Transmit(&huart2, (uint8_t*)MESSAGE, strlen(MESSAGE), HAL_MAX_DELAY);
116
117     // Is er een byte ontvangen via de USART2?
118     // Lees de info in via 'polling' (= op gezette momenten vragen of er info is).
119     if(__HAL_UART_GET_FLAG(&huart2, UART_FLAG_RXNE)) ←
120     {
121         HAL_UART_Receive(&huart2, (uint8_t*)&receivedData, 1, HAL_MAX_DELAY);
122         ByteToLeds(receivedData);
123     }
```

Als er een zekere tijd verloopt tussen de momenten waarop je een 'vlag' controleert, **kan het zijn dat je data mist**. Dat is mogelijk rampzalig!

Bekijk zeker de werking van
HAL_UART_Transmit() en *HAL_UART_Receive()*.



5

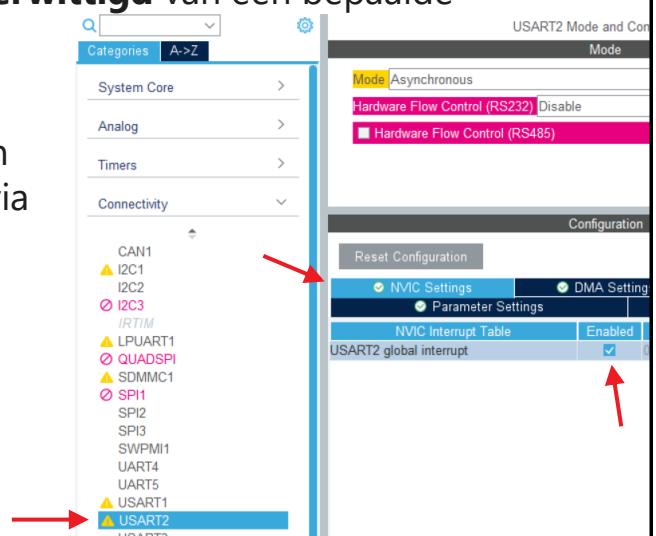
Interrupts



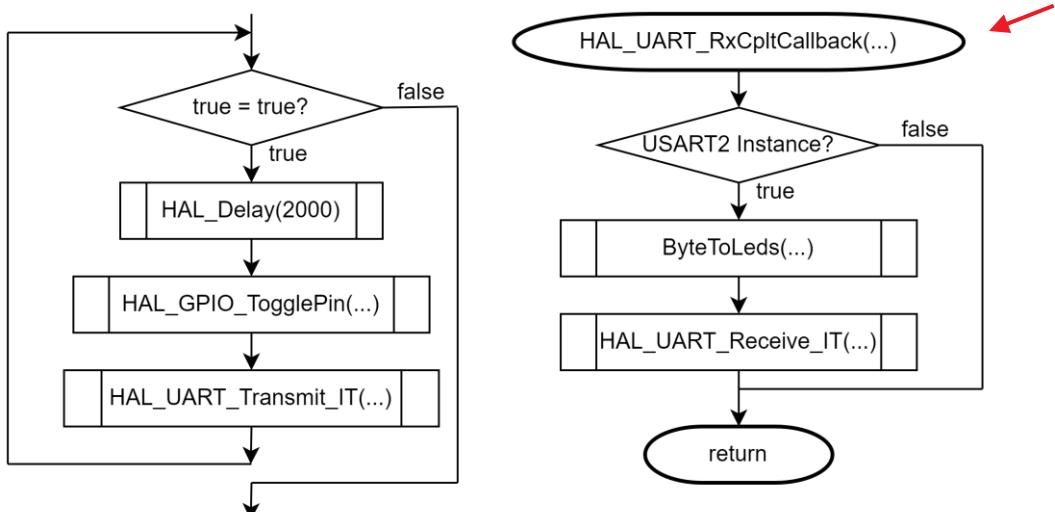
6

— Interrupts (1)

- **Interrupts** = de CPU **wordt verwittigd** van een bepaalde situatie via een interruptvlag.
- Om HAL interrupts toe te laten van bv. USART2, dien je deze via CubeMX in te schakelen (NVIC settings).



— Interrupts (2)



— Interrupts (3)

De hoofdlus bevat enkel nog code om UART-data te verzenden. Het ontvangen gebeurt elders: in de interrupt subroutine.

```
104 // Ontvangst van één byte via interrupts starten.  
105 HAL_UART_Receive_IT(&huart2, (uint8_t*)&receivedData, 1);  
106 /* USER CODE END 2 */  
107  
108 /* Infinite loop */  
109 /* USER CODE BEGIN WHILE */  
110 while (1)  
{  
    // Even wachten.  
    HAL_Delay(2000);  
    // User LED toggle'n.  
    HAL_GPIO_TogglePin(UserLED_GPIO_Port, UserLED_Pin);  
    // Teken van leven geven.  
    HAL_UART_Transmit_IT(&huart2, (uint8_t*)MESSAGE, strlen(MESSAGE));
```

Opmerking: ontvangen starten via USART2 interrupt vereist deze lijn code.

9



— Interrupts (4)

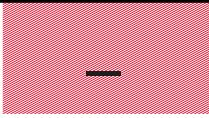
```
359 // Code hieronder wordt opgeroepen als alle data van HAL_UART_Receive_IT() ontvangen is...  
360 // Deze functie wordt opgeroepen vanuit HAL_UART_IRQHandler().  
361 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *UartHandle)  
362 {  
    // Is het data van UART2?  
    if(UartHandle->Instance == USART2)  
    {  
        // Ontvangen data op de LED's zetten.  
        ByteToLeds(receivedData);  
        // Ontvangst via interrupts opnieuw starten. Want RXNEIE wordt automatisch uitgeschakeld.  
        HAL_UART_Receive_IT(&huart2, (uint8_t*)&receivedData, 1);  
    }  
}
```

De interrupt code kan focussen op de ontvangst van data van de UART. Daardoor is de **kans minimaal dat je data misloopt!** Het nadeel is wel dat de CPU moet 'wegen' van de hoofdlus naar de interrupt code.

Hou interrupt code zo kort mogelijk (qua tijdsbestek)!

10

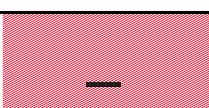




Direct Memory Access



11



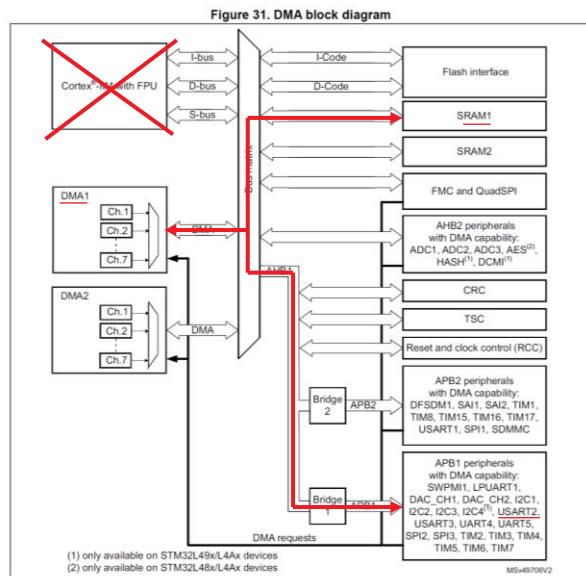
— Direct Memory Access (1)

- **Direct Memory Access (DMA)** = de CPU **wordt vrijgesteld** van het afhandelen van de communicatie. Dat doet de DMA controller.
- Onze microcontroller heeft twee DMA controllers. Dat zijn **systemen die onafhankelijk van de CPU**, ook gegevens kunnen uitwisselen tussen 'peripherals' en 'memories'.
- We kunnen dus bijvoorbeeld instellen dat er tien bytes opgevraagd worden van UART2 en naar het geheugen gekopieerd worden, **ZONDER dat de CPU daarvoor nodig is!**

12

— Direct Memory Access (2)

- Merk op: aan de linkerkant zie je de CPU, DMA1 en DMA2. Dat zijn drie 'bus masters'.
- Ze hebben toegang tot de 'Bus Matrix' en kunnen **zelfstandig data uitwisselen**.
- Er is wel **nog één interrupt nodig** om aan te geven dat de uitwisseling voorbij is.



Bron: STM32L476RG - Reference Manual

— Direct Memory Access (3)

- Zowel de CPU als de twee DMA controllers kunnen de **systeembussen gebruiken**. Het is de bus matrix die het 'round-robin' schema implementeert zodat alles mooi gestroomlijnd geraakt. Iedere deelnemer krijgt dus gelijkwaardige toegang tot de gedeelde bussen.

The DMA controller performs direct memory transfer by sharing the AHB system bus with other system masters. The bus matrix implements round-robin scheduling. DMA requests may stop the CPU access to the system bus for a number of bus cycles, when CPU and DMA target the same destination (memory or peripheral).

Bron: STM32L476RG - Reference Manual (11.4.1 DMA)

Zie ook: [https://nl.wikipedia.org/wiki/Round-robin_\(informatietechnologie\)](https://nl.wikipedia.org/wiki/Round-robin_(informatietechnologie))

— Direct Memory Access (4)

- Indien verschillende kanalen van één bepaalde DMA Controller gebruikt worden, kan je de **prioriteit** instellen via software en is er ook een hardware prioriteit indien nodig.

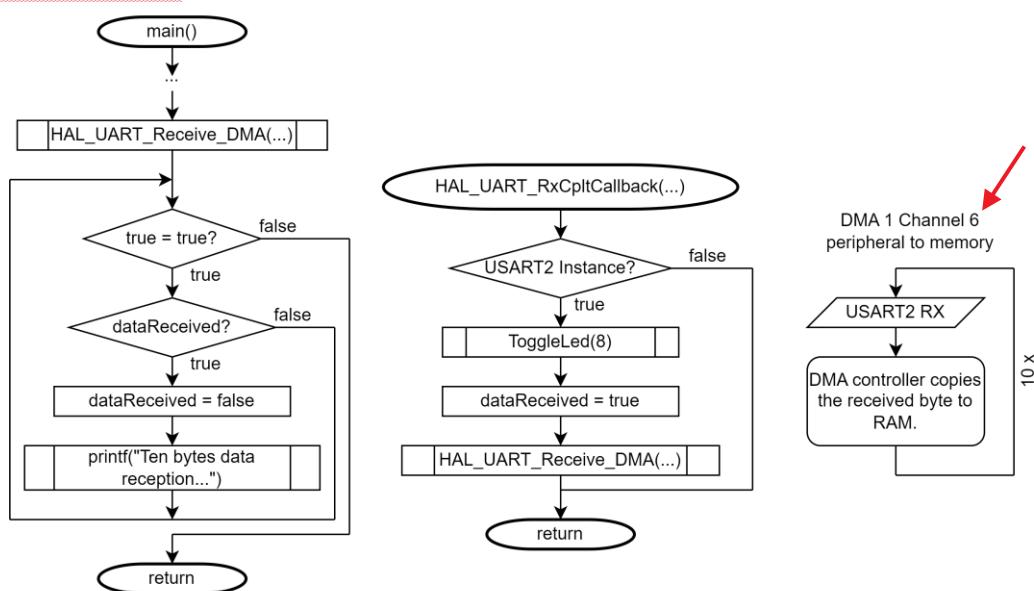
Priority between the requests is programmable by software (4 levels per channel: very high, high, medium, low) and by hardware in case of equality (such as request to channel 1 has priority over request to channel 2).

Bron: STM32L476RG - Reference Manual (11.2 DMA)

15

hogeschool
VIVES

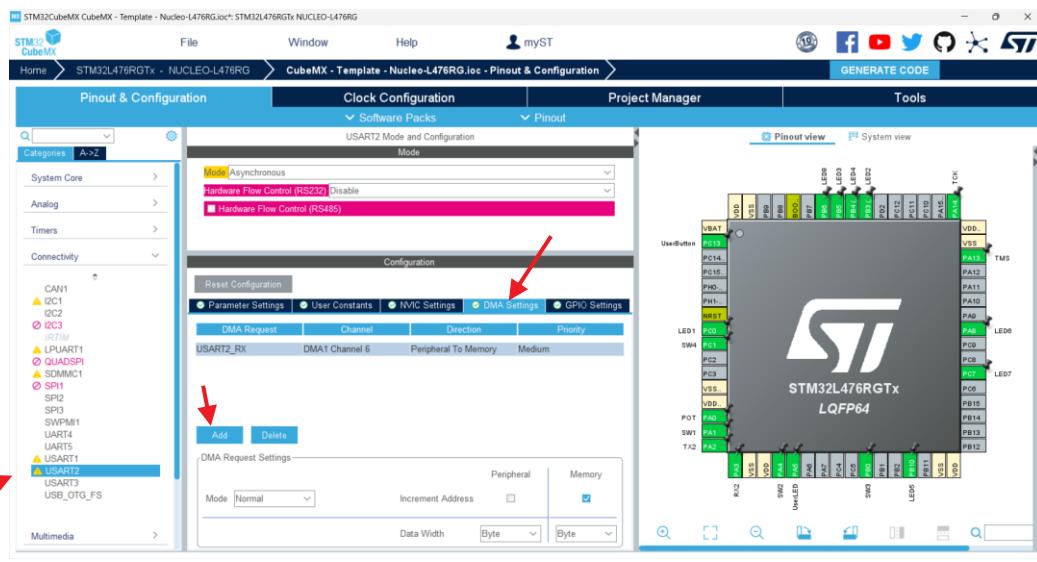
— Direct Memory Access (5)



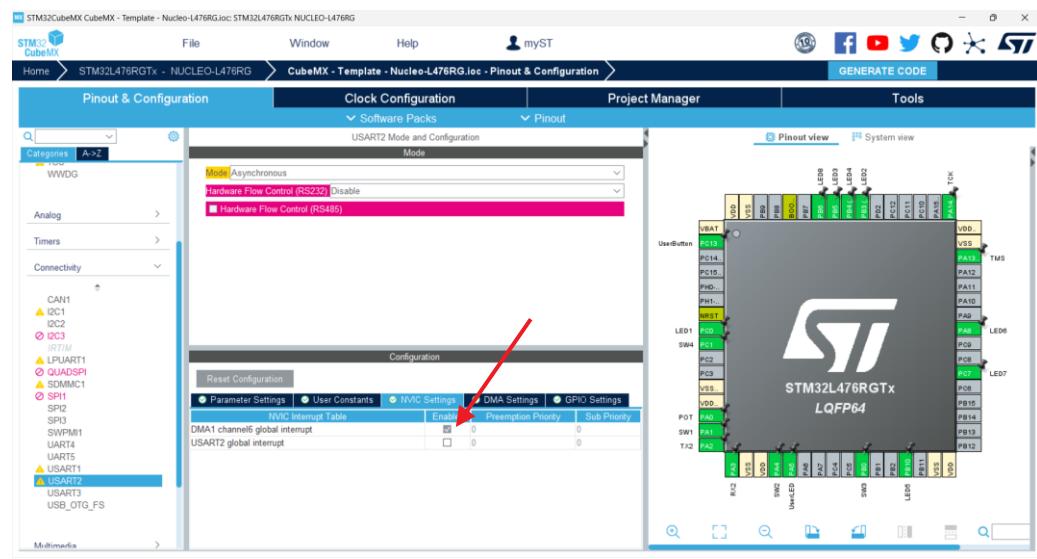
16

hogeschool
VIVES

Direct Memory Access (DMA)



Direct Memory Access (7)



— Direct Memory Access (8)

```
107 // Ontvangst van 10 bytes via DMA starten.  
108 HAL_UART_Receive_DMA(&huart2, destination, NUMBER_OF_BYTES_TO_RECEIVE);  
109 /* USER CODE END 2 */  
110 /* Infinite loop */  
111 /* USER CODE BEGIN WHILE */  
112 while (1)  
113 {  
114     // Is er data ontvangen via DMA?  
115     if(dataReceived)  
116     {  
117         // Melding wissen.  
118         dataReceived = false;  
119         // Gebruiker informeren (niet via DMA).  
120         printf("Ten bytes data reception complete...\r\n");  
121     }  
122 }
```

19



— Direct Memory Access (9)

```
380 // De code hieronder wordt opgeroepen als alle data van HAL_UART_Receive_DMA() ontvangen is...  
381 // Deze functie wordt opgeroepen vanuit:  
382 //     DMA1_Channel6_IRQHandler() -> HAL_DMA_IRQHandler() -> HAL_UART_RxCpltCallback().  
383 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *UartHandle)  
384 {  
385     // Is het data van USART2?  
386     if(UartHandle->Instance == USART2)  
387     {  
388         ToggleLed(8);  
389         dataReceived = true;  
390         // Teken van leven geven...  
391         // Melden aan de hoofdlus.  
392         // Ontvangst via DMA opnieuw starten. Want in het DMA1_Channel6->CCR register,  
393         // wordt de TCIE bit automatisch uitgeschakeld.  
394         // Dat gebeurt in: HAL_DMA_IRQHandler().  
395         HAL_UART_Receive_DMA(&huart2, destination, NUMBER_OF_BYTES_TO_RECEIVE);  
396     }  
397 }
```

20



— Direct Memory Access (10)

Tot slot:

- Je kan ook DMA gebruiken voor het **verzenden** van data.
- Er zijn **vele peripherals** die DMA ondersteuning hebben: UART, SPI, I²C, AD, DA, Timers, ...
- Zoek steeds op **welke 'channel'** van de DMA controller je kan gebruiken voor welke peripheral.

Table 44. DMA1 requests for each channel

| CxS[3:0] | Channel 1 | Channel 2 | Channel 3 | Channel 4 | Channel 5 | Channel 6 | Channel 7 |
|----------|-----------|-----------|-----------|------------------|------------------|------------------|------------------|
| 0000 | ADC1 | ADC2 | ADC3 | DFSDM1_- FLT0 | DFSDM1_- FLT1 | DFSDM1_- FLT2 | DFSDM1_- FLT3 |
| 0001 | - | SPI1_RX | SPI1_TX | SPI2_RX | SPI2_TX | SAI2_A | SAI2_B |
| 0010 | - | USART3_TX | USART3_RX | USART1_TX | USART1_RX | USART2_RX | USART2_TX |

Bron: STM32L476RG - Reference Manual



—

Vergelijking:
UART-data verzenden via polling,
interrupt en DMA.

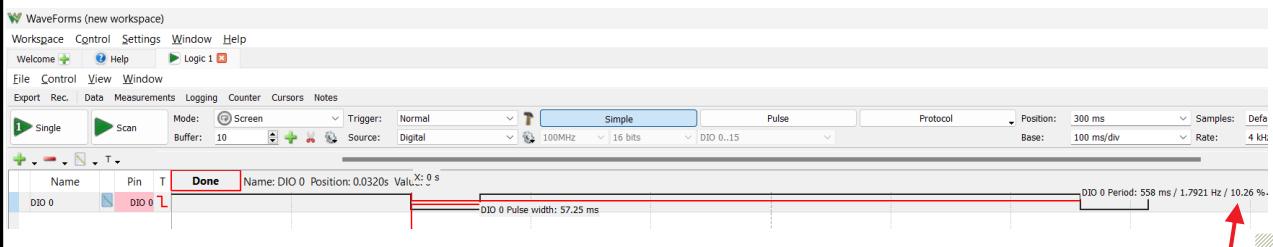
Vergelijking: UART TX polling versus DMA (1)

Polling

```
106 |     while (1)
107 |     {
108 |         // LED1 aanzetten. Dit geeft aan dat de hoofdlus begint met nuttige zaken...
109 |         // Breng de toestand van LED1 in beeld met een USB oscilloscoop.
110 |         HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_SET);
111 |
112 |         // Hier kan dan nuttige code komen...
113 |
114 |         // LED1 uitzetten. Dit geeft aan dat de hoofdlus begint met verzenden van
115 |         // de UART-data...
116 |         HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);
117 |
118 |         // UART-data verzenden starten (in een blokkerende functie).
119 |         HAL_UART_Transmit(&huart2, (uint8_t*)MESSAGE, strlen(MESSAGE), HAL_MAX_DELAY);
120 |
121 |         // LED1 aanzetten. Dit geeft aan dat de hoofdlus begint met nuttige zaken...
122 |         HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_SET);
123 |
124 |         // Hier kan ook nuttige code komen...
125 |
126 |         // Even wachten. Het wachten is dikwijls geen goed idee, maar
127 |         // hier helpt het om de zaken beter te begrijpen (hopelijk) ;-).
128 |         HAL_Delay(500);
129 |     /* USER CODE END WHILE */
```

Vergelijking: UART TX polling versus DMA (2)

Polling



Van de tijd die de hoofdlus ter beschikking heeft, gaat er dus meer dan **10%** 'verloren' aan het versturen van de UART-data. Dat is meer dan 57ms!

Die tijd kan wellicht nuttiger besteed worden door de CPU....

Vergelijking: UART TX polling versus DMA (3)

DMA

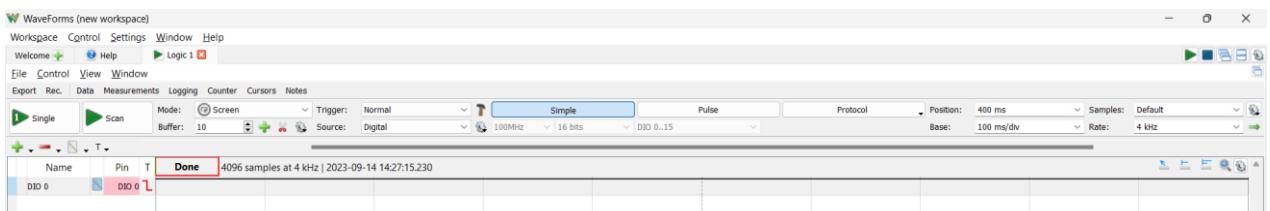
```
110 while (1)
111 {
112     // LED1 aanzetten. Dit geeft aan dat de hoofdlus begint met nuttige zaken...
113     // Breng de toestand van LED1 in beeld met een USB oscilloscoop.
114     HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_SET);
115
116     // Hier kan dan nuttige code komen...
117
118     // LED1 uitzetten. Dit geeft aan dat de hoofdlus begint met verzenden van
119     // de UART-data...
120     HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);
121
122     // UART-data verzenden starten (met DMA instellingen).
123     HAL_UART_Transmit_DMA(&huart2, (uint8_t*)MESSAGE, strlen(MESSAGE));
124
125     // LED1 aanzetten. Dit geeft aan dat de hoofdlus begint met nuttige zaken...
126     HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_SET);
127
128     // Hier kan ook nuttige code komen...
129
130     // Even wachten. Het wachten is dikwijls geen goed idee, maar
131     // hier helpt het om de zaken beter te begrijpen (hopelijk) ;-).
132     HAL_Delay(500);
```

25



Vergelijking: UART TX polling versus DMA (4)

DMA



De tijd waarin de CPU nu onbeschikbaar is, is zodanig kort dat hij bij een grote tijdsbasis niet zichtbaar is op de oscilloscoop ...

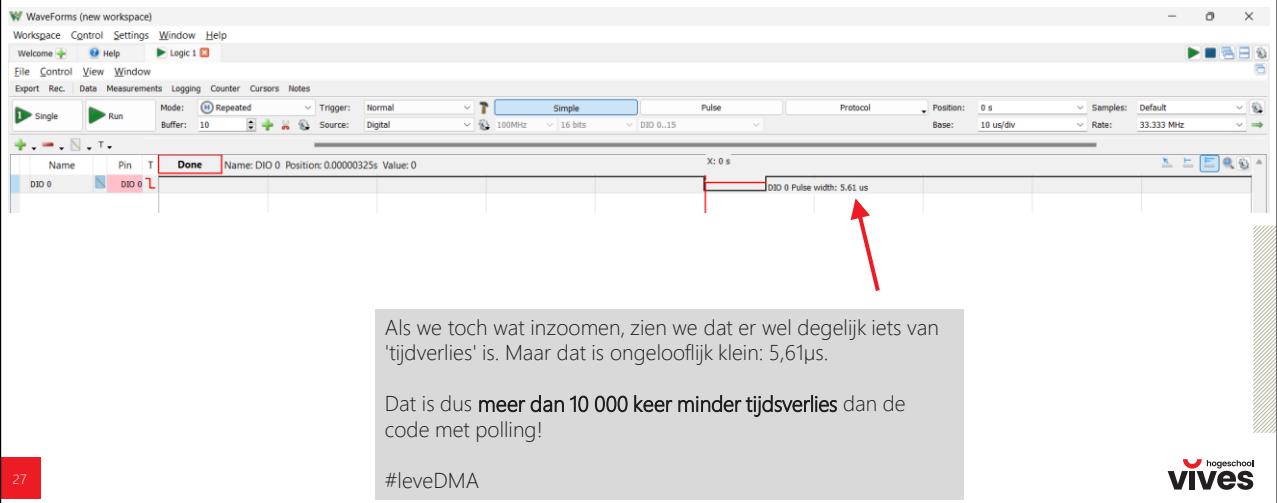
De CPU heeft dus quasi het volledige tijdsbestek beschikbaar om nuttige berekeningen te doen....

26



Vergelijking: UART TX polling versus DMA (5)

DMA

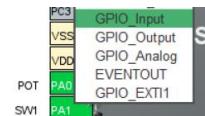


Interrupt van een knop



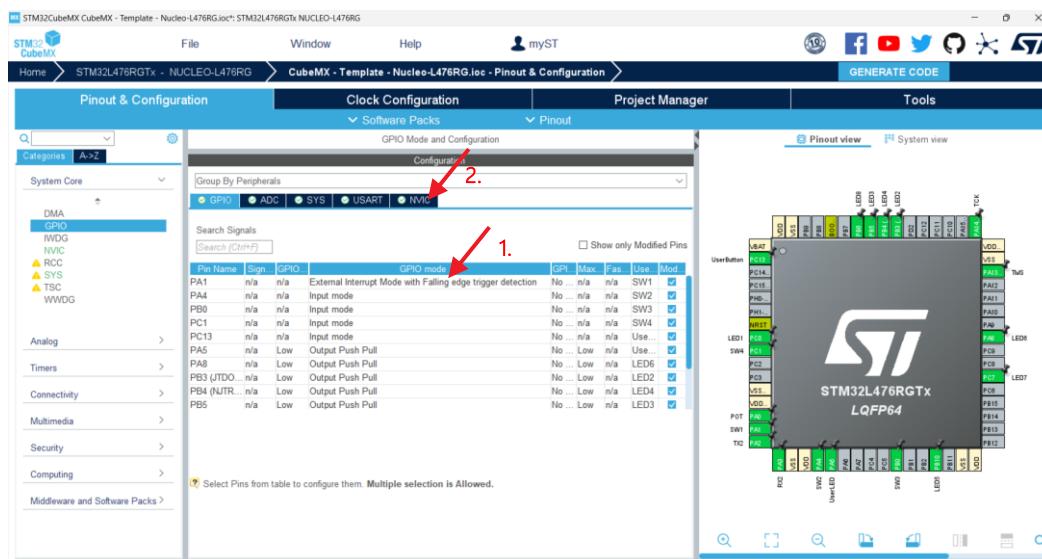
— Interrupt van een knop (1)

- Je kan uiteraard ook met **HAL-bibliotheken** een **interrupt** van een **knop** binnenlezen.
- Daarvoor gebruik je de microcontroller pinnen met **EXTI-mogelijkheden**.
 - Zie cursus 'Microcontrollers' voor CMSIS-voorbeelden.
- Klik in STM32CubeMX onder **GPIO** op de pin om een **GPIO_Input** te veranderen naar een **GPIO_EXTIx**:
 - Bv. voor PA1:
- Daarna stel je de gewenste **GPIO mode** in en geef je terug een passende user label.
- Vergeet ook niet om de interrupt op NVIC-niveau te activeren: ga naar tabblad **NVIC** en **activeer EXTI line interrupt**.



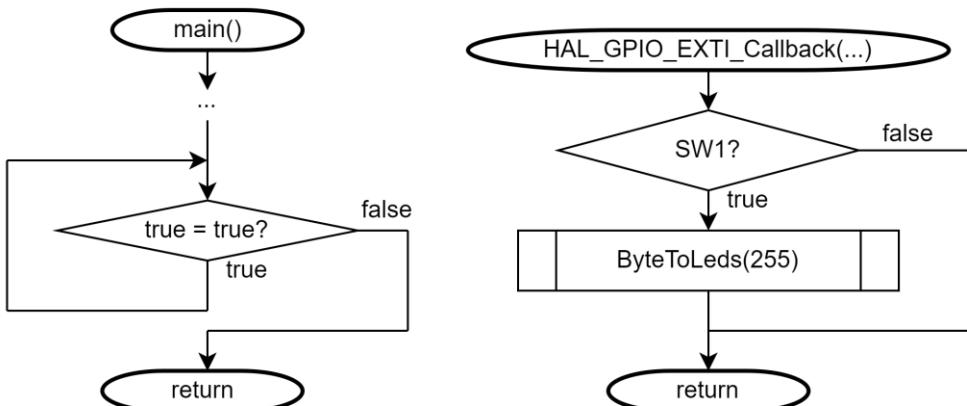
29

— Interrupt van een knop (2)



30

— Interrupt van een knop (3)



31

hogeschool
VIVES

— Interrupt van een knop (4)

```
352 // Als er een interrupt is, dan wordt de EXTI1_IRQHandler() 
353 // interrupthandler uitgevoerd in 'stm32l4xx_it.c'.
354 // In die functie, wordt de interruptvlag gereset en onderstaande
355 // callback functie opgeroepen...
356 // De instellingen om de interrupt code te maken, wordt gedaan in
357 // STM32CubeMX bij de juiste pin.
358 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
359 {
360     // Werd er op SW1 gedrukt?
361     if(GPIO_Pin == SW1_Pin)
362         ByteToLeds(255);
363 }
```

Hou ook hier (zoals ALTIJD) de interrupt code zo kort mogelijk (qua tijdsbestek)!

32

hogeschool
VIVES

Demo: automatische kofferdeur

33

hogeschool
vives

— **Automatische kofferdeur (1)**

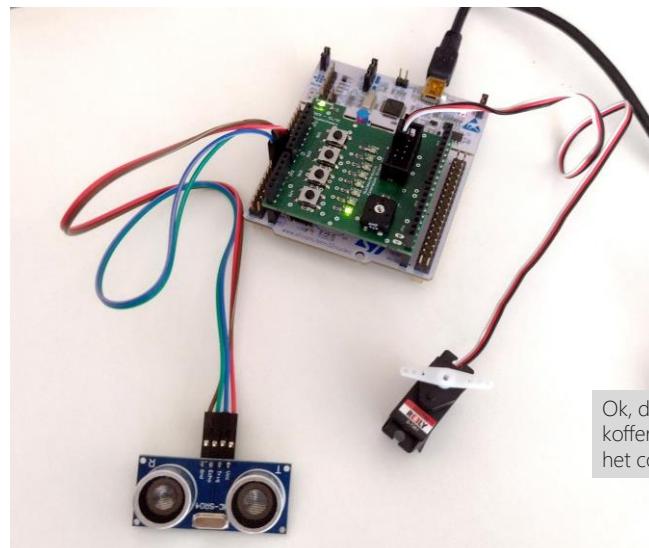
- Kunnen we een **automatische kofferdeur** simuleren?
- Gebruikte **onderdelen**:
 - Nucleo-L476RG.
 - HC-SR04 ultrasone sensor.
 - modelbouwservo.
- Gebruikte **technieken**:
 - Timer Input Capture.
 - Timer PWM.



34

hogeschool
vives

Automatische kofferdeur (2)



Ok, dit lijkt nog niet op een kofferdeur... Maar dit is louter het concept.

hogeschool VIVES

35

Automatische kofferdeur (3)

STM32CubeMX - Template - Nucleo-L476RG.ioc STM32L476RGx NUCLEO-L476RG

File Window Help myST GENERATE CODE

Pinout & Configuration Clock Configuration Project Manager Tools

Categories A-Z

System Core > Analog > Timers > TIM1 > TIM2

1. TIM2_CH1

2. TIM1_CH2

3. Prescaler set to 79

Prescaler ingesteld op 79, dit om klokfrequentie van timer 2 te laten tellen per 1 µs.
Uitwerking: zie p. 1184 in STM32L476RG Reference Manual:

Counter clock frequency (CK_{CNT}) = $\frac{f_{CK_PSC}}{(PSC + 1)} = \frac{80MHz}{(79 + 1)} = 1MHz = 1\mu s$

36

Automatische kofferdeur (4)

```
143 // While loop.
144 {
145     // Trigger uitsturen zodat de meting start.
146     HAL_GPIO_WritePin(TRIGGER_GPIO_Port, TRIGGER_Pin, GPIO_PIN_SET);
147     HAL_Delay(1);
148     HAL_GPIO_WritePin(TRIGGER_GPIO_Port, TRIGGER_Pin, GPIO_PIN_RESET);
149
150     // Even wachten op het onhoorbaar geluid.
151     HAL_Delay(100);
152
153     // De tijd wordt gemeten in microseconden. Bereken daaruit de afstand.
154     // 1cm (moet dubbel afgelegd worden) = 0.02m/340m/s = ~59 µs.
155     // Enkel de waarden boven de 2 cm en onder de 120 cm vertrouwen.
156     distance = capture/ONE_CM_SOUND_TRAVELLING_US_TIME;
157     if((distance >= CM_DISTANCE_MINIMUM_RELIABLE_THRESHOLD) && (distance <= CM_DISTANCE_MAXIMUM_RELIABLE_THRESHOLD))
158     {
159         // JSON-data versturen met daarin de gemeten afstand in centimeters.
160         printf("(\"distance\":%d)\r\n", distance);
161         ByteToLevel(distance*2);
162
163         // Kofferdeksel openen door de PWM voor de servo te wijzigen.
164         if(distance < CM_DISTANCE_TO_OPEN_DOOR)
165             __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, PWM_OFFSET + PWM_SETPOINT);
166     }
167     else
168     {
169         // Ongeldige afstand gemeten. Verstuur -1 via JSON.
170         printf("(\"distance\":-1)\r\n");
171         ByteToLevel(0);
172     }
173
174     // Kofferdeksel terug sluiten wanneer er op de User Button gedrukt wordt.
175     if(UserButtonActive())
176         __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, PWM_OFFSET);
```

37



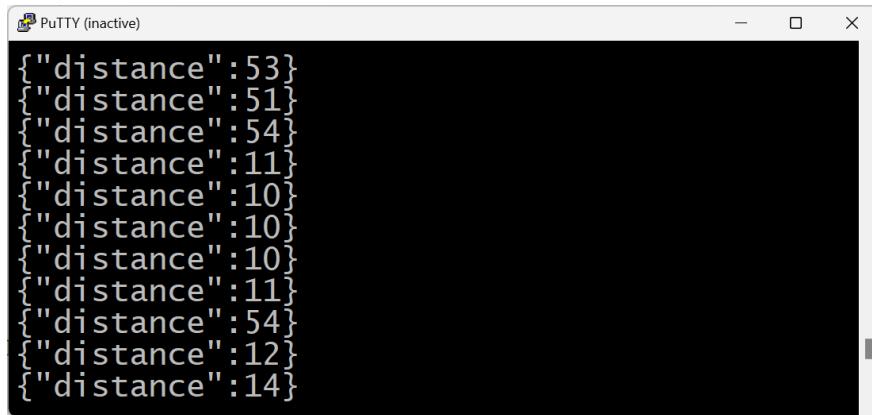
Automatische kofferdeur (5)

```
525 // Callback voor iedere interrupt van Timer 2 ontvangen.
526 void HAL_TIM_IC_CaptureCallback (TIM_HandleTypeDef *htim)
527{
528     // Is het Timer 2?
529     if(htim->Instance == TIM2)
530     {
531         // Kijken wat de staat van de ECHO-pin is. Afhankelijk daarvan weet je of je een dalende of stijgende flank had.
532         if(HAL_GPIO_ReadPin(ECHO_GPIO_Port, ECHO_Pin) == GPIO_PIN_SET)
533         {
534             // Stijgende flank gezien. Begin van de meting.
535
536             // Interruptvlag resetten.
537             __HAL_TIM_CLEAR_IT(htim, TIM_IT_CC1);
538
539             // Teller resetten.
540             __HAL_TIM_SET_COUNTER(&htim2, 0);
541
542             // Vanaf nu wachten op een falling edge.
543             __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_1, TIM_INPUTCHANNELPOLARITY_FALLING);
544
545             // Visueel aangeven dat de stijgende flank er is.
546             SetUserLed(true);
547         }
548     }
549     else
550     {
551         // Dalende flank gezien. Einde van de meting.
552
553         // Gemeten tijd opslaan. Hierdoor wordt ook de interruptvlag gereset.
554         capture = __HAL_TIM_GET_COMPARE(htim, TIM_CHANNEL_1);
555
556         // Visueel aangeven dat de dalende flank er is.
557         SetUserLed(false);
558
559         // Wachten op rising edge.
560         __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_1, TIM_INPUTCHANNELPOLARITY_RISING);
561     }
562 }
```

38



Automatische kofferdeur (6)



A screenshot of a PuTTY terminal window titled "PuTTY (inactive)". The window displays a JSON array of 12 objects, each containing a "distance" key with a numerical value. The values are: 53, 51, 54, 11, 10, 10, 10, 11, 54, 12, and 14.

```
{"distance":53}, {"distance":51}, {"distance":54}, {"distance":11}, {"distance":10}, {"distance":10}, {"distance":10}, {"distance":11}, {"distance":54}, {"distance":12}, {"distance":14}
```

— IoT Devices

Hoofdstuk 4: PC

L. Espeel



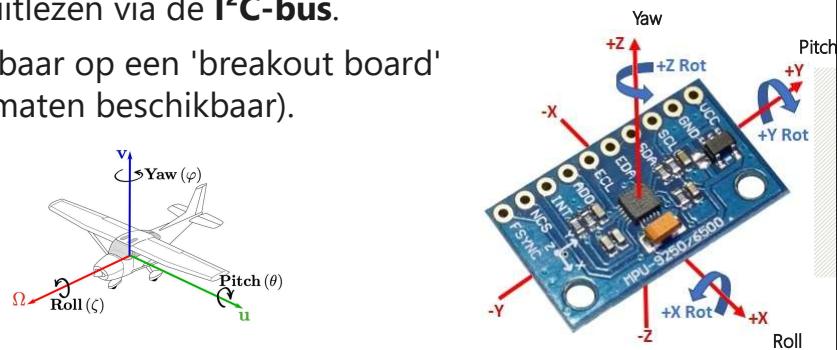
(bron: grotendeels van R. Buysschaert)

—

MPU-9250
algemene info

— MPU-9250 (1)

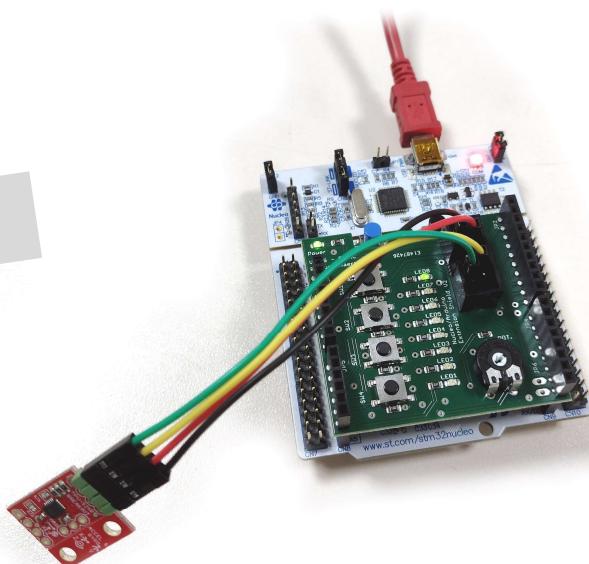
- De MPU-9250 is een **Inertia Measurement Unit (IMU)** en bevat:
 - **Gyroscoop:** meten van hoeksnelheid ($^{\circ}/\text{s}$)
 - **Accelerometer:** meten van lineaire versnelling (m/s^2)
 - **Magnetometer:** meten van magnetische veldsterkte (μT)
- Je kan deze sensor uitlezen via de **I²C-bus**.
- De sensor is beschikbaar op een 'breakout board'
(in verschillende formaten beschikbaar).



3

— MPU-9250 (2)

Je kan de sensor verbinden met
'jumper wires'. Let erop dat je de
GND en 3V3 correct aansluit!

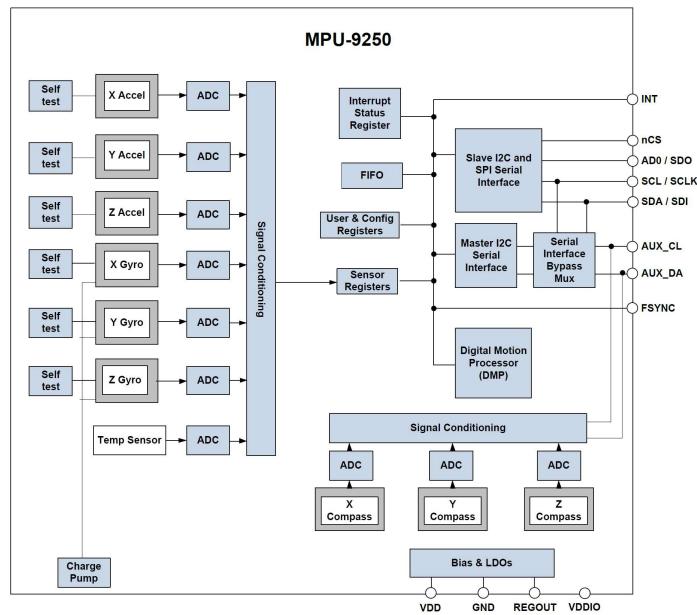


4

MPU-9250 (3)

Zoek de accelerometer en gyrocoop voor de drie assen...

5



Bron: PS-MPU-9250A-01-V1.pdf

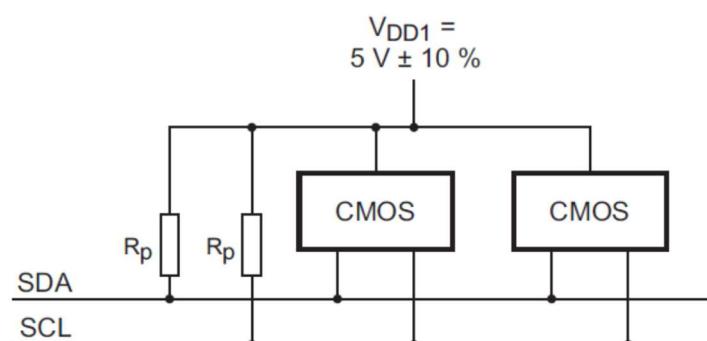
hogeschool
VIVES

I²C-bus

Let op de **pull-up weerstanden!** Ze zorgen ervoor dat de SDA en SCL, door verschillende gebruikers veilig 'laag' gemaakt kunnen worden zonder dat er schade ontstaat door te hoge stromen. Alle gebruikers moeten wel werken met een 'open collector' of 'open drain' output.

OPM: hier wordt 5V als busspanning aangegeven, wij gaan echter **meestal 3,3V gebruiken!**

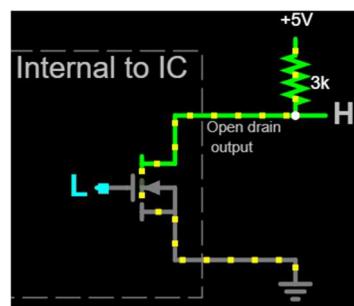
6



hogeschool
VIVES

— Open drain

De **werking van een open drain output**, wordt heel bevattelijk voorgesteld op Wikipedia.



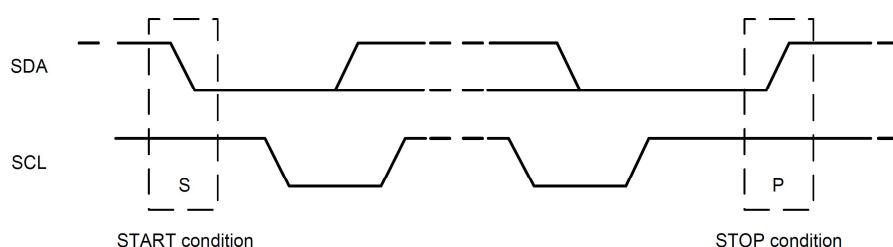
https://upload.wikimedia.org/wikipedia/commons/4/47/Animated_open_drain_output.gif

7

hogeschool
VIVES

— I²C-principe (1)

- De open drain (of open collector) outputs moeten aangewend worden om de I²C-communicatie te realiseren.
- Daarbij zijn de **start- en stopconditie** de basis. Daartussen heb je natuurlijk ook nog de verzonden **data** en de **acknowledge** (of not acknowledge).



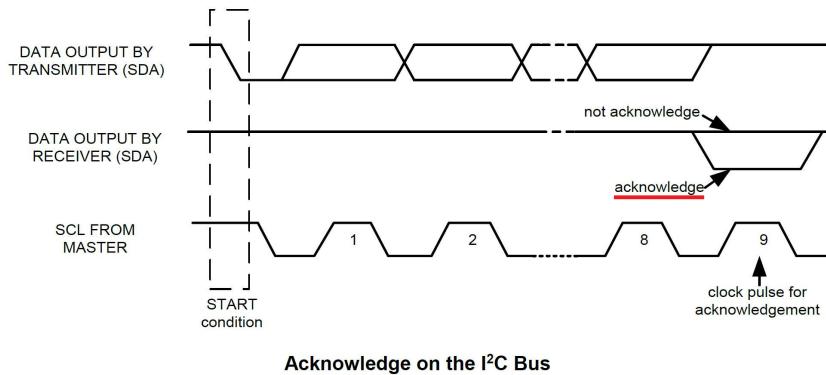
8

hogeschool
VIVES

I²C-principe (2)

Data Format / Acknowledge

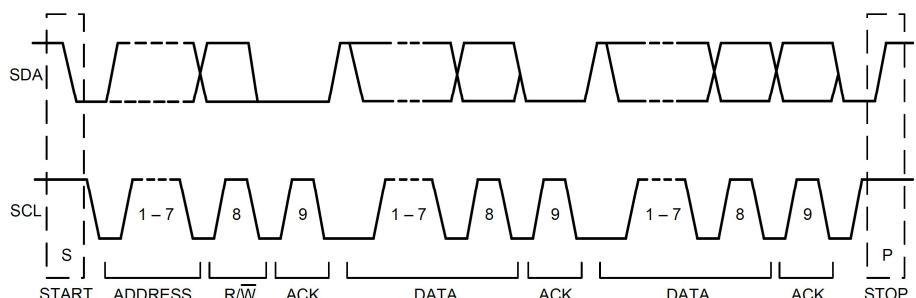
I²C data bytes are defined to be 8-bits long. There is no restriction to the number of bytes transmitted per data transfer. Each byte transferred must be followed by an acknowledgement (ACK) signal. The clock for the acknowledgement signal is generated by the master, while the receiver generates the actual acknowledgement signal by pulling down SDA and holding it low during the HIGH portion of the acknowledgement clock pulse.



Bron: PS-MPU-9250A-01-v1.1.pdf

I²C-principe (3)

Een volledig stuk I²C-communicatie kan er dan zo uitzien...



Bron: PS-MPU-9250A-01-v1.1.pdf

I²C-principe (4)

Je kan data **schrijven** van microcontroller (master) naar sensor (slave).

Single-Byte Write Sequence

S = start, AD = slave address, W = write bit (0), R = read bit (1), ACK = acknowledge, RA = (internal) register address, P = stop, NACK = Not ACK, ...

| | | | | | | | | |
|--------|---|------|-----|----|-----|------|-----|---|
| Master | S | AD+W | | RA | | DATA | | P |
| Slave | | | ACK | | ACK | | ACK | |

Burst Write Sequence

| | | | | | | | | | | |
|--------|---|------|-----|----|-----|------|-----|------|-----|---|
| Master | S | AD+W | | RA | | DATA | | DATA | | P |
| Slave | | | ACK | | ACK | | ACK | | ACK | |

11

I²C-principe (5)

Maar je kan ook data **lezen** van sensor (slave) naar microcontroller (master). Merk op dat je hier toch eerst een 'schrijfgedeelte' hebt!

Single-Byte Read Sequence

S = start, AD = slave address, W = write bit (0), R = read bit (1), ACK = acknowledge, RA = (internal) register address, P = stop, NACK = Not ACK, ...

| | | | | | | | | | | | |
|--------|---|------|-----|----|-----|---|------|-----|------|------|---|
| Master | S | AD+W | | RA | | S | AD+R | | | NACK | P |
| Slave | | | ACK | | ACK | | | ACK | DATA | | |

Burst Read Sequence

| | | | | | | | | | | | | | |
|--------|---|------|-----|----|-----|---|------|-----|------|-----|------|------|---|
| Master | S | AD+W | | RA | | S | AD+R | | | ACK | | NACK | P |
| Slave | | | ACK | | ACK | | | ACK | DATA | | DATA | | |

12

—

MPU-9250 inlezen via polling

13



— **MPU-9250 inlezen via polling (1)**

Eerder werd reeds aangehaald dat we op drie manieren kunnen communiceren over digitale bussen:

1. via polling (= telkens vragen of we een stap verder kunnen).
2. via interrupts (= we krijgen een melding als we de volgende stap moeten zetten).
3. via DMA (= we krijgen een melding als de laatste stap net gezet is).

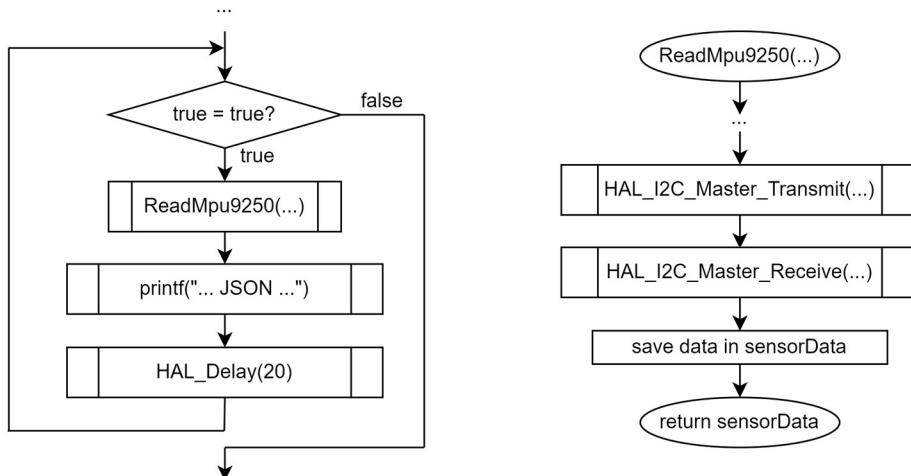
Polling is de meest eenvoudige, maar minste efficiënte.

14



MPU-9250 inlezen via polling (2)

OPM: hier zijn de transmit en receive functies 'blokkerend'!



15

vives hogeschool

MPU-9250 inlezen via polling (3)

```
115 | while (1)
116 | {
117 |     // IMU uitlezen en JSON dumpen.
118 |     sensordata = ReadMpu9250(&hi2c1);
119 |     printf("{\"accx\":%d, \"accy\":%d, \"accz\":%d, \"gyrox\":%d, \"gyroy\":%d, \"gyroz\":%d}\r\n",
120 |             sensordata.accX, sensordata.accY, sensordata.accZ, sensordata.gyroX,
121 |             sensordata.gyroY, sensordata.gyroZ);
122 |
123 |     HAL_Delay(20);
```

16

vives hogeschool

— MPU-9250 inlezen via polling (4)

```
59 Mpu9250 ReadMpu9250(I2C_HandleTypeDef* hi2c)
60{
61    Mpu9250 sensordata;
62    uint8_t data[14];
63
64    // Register 0x3B: Acc X High byte.
65    data[0] = 0x3B;
66    HAL_I2C_Master_Transmit(hi2c, (SLAVE_ADDRESS_GYRO_ACC << 1), data, 1, 100);
67
68    // 14 bytes aan IMU-data lezen.
69    HAL_I2C_Master_Receive(hi2c, (SLAVE_ADDRESS_GYRO_ACC << 1), data, 14, 100);
70
71    // Waardes uitrekenen.
72    sensordata.accX = (int16_t)(data[0] * 256 + data[1]);
73    sensordata.accY = (int16_t)(data[2] * 256 + data[3]);
74    sensordata.accZ = (int16_t)(data[4] * 256 + data[5]);
75    sensordata.temp = (int16_t)(data[6] * 256 + data[7]);
76    sensordata.gyroX = (int16_t)(data[8] * 256 + data[9]);
77    sensordata.gyroY = (int16_t)(data[10] * 256 + data[11]);
78    sensordata.gyroZ = (int16_t)(data[12] * 256 + data[13]);
79
80    return sensordata;
81}
```

17



— MPU-9250 inlezen via polling (5)

Hoe kan je weten dat je met 'blokkerende' functies werkt? Zoek het op in de HAL User Manual (UM1884).

HAL_I2C_Master_Transmit

Function name

```
HAL_StatusTypeDef HAL_I2C_Master_Transmit(I2C_HandleTypeDef* hi2c, uint16_t DevAddress, uint8_t
* pData, uint16_t Size, uint32_t Timeout)
```

Function description

Transmits in master mode an amount of data in blocking mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

18



— MPU-9250 inlezen via polling (6)

HAL_I2C_Master_Receive

Function name

```
HAL_StatusTypeDef HAL_I2C_Master_Receive (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t_t Size, uint32_t Timeout)
```

Function description

Receives in master mode an amount of data **in blocking mode**.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

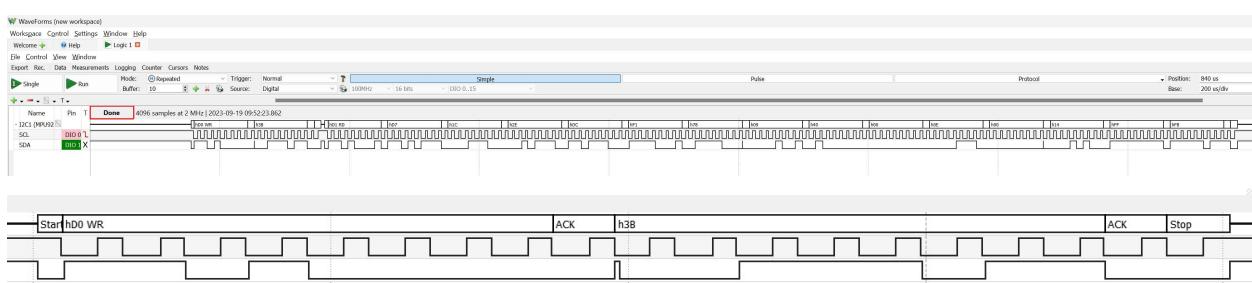
- **HAL:** status

19



— MPU-9250 inlezen via polling (7)

Kan je de voorspelde communicatie hier zien in onderstaand scoopbeeld?



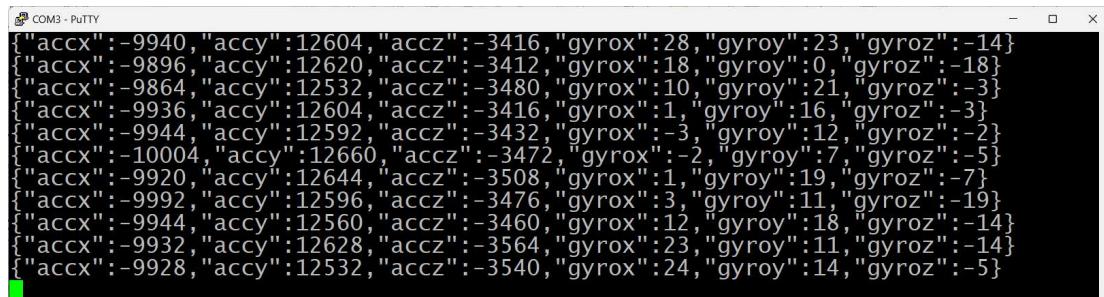
Het 7-bit **MPU-adres** is 0x68. Als je dat één bit naar links **opschuift** kom je uit op 0xD0... Daarbij komt dan de lees- of schrijfbit.

20



— MPU-9250 inlezen via polling (8)

De code stuurt de gemeten data in JSON-formaat door via USART2. Met die info kan je dus bijvoorbeeld een C#-applicatie maken om eender wat te doen ...



```
{"accx": -9940, "accy": 12604, "accz": -3416, "gyrox": 28, "gyroy": 23, "gyroz": -14}  
{"accx": -9896, "accy": 12620, "accz": -3412, "gyrox": 18, "gyroy": 0, "gyroz": -18}  
{"accx": -9864, "accy": 12532, "accz": -3480, "gyrox": 10, "gyroy": 21, "gyroz": -3}  
{"accx": -9936, "accy": 12604, "accz": -3416, "gyrox": 1, "gyroy": 16, "gyroz": -3}  
{"accx": -9944, "accy": 12592, "accz": -3432, "gyrox": -3, "gyroy": 12, "gyroz": -2}  
{"accx": -10004, "accy": 12660, "accz": -3472, "gyrox": -2, "gyroy": 7, "gyroz": -5}  
 {"accx": -9920, "accy": 12644, "accz": -3508, "gyrox": 1, "gyroy": 19, "gyroz": -7}  
 {"accx": -9992, "accy": 12596, "accz": -3476, "gyrox": 3, "gyroy": 11, "gyroz": -19}  
 {"accx": -9944, "accy": 12560, "accz": -3460, "gyrox": 12, "gyroy": 18, "gyroz": -14}  
 {"accx": -9932, "accy": 12628, "accz": -3564, "gyrox": 23, "gyroy": 11, "gyroz": -14}  
 {"accx": -9928, "accy": 12532, "accz": -3540, "gyrox": 24, "gyroy": 14, "gyroz": -5}
```

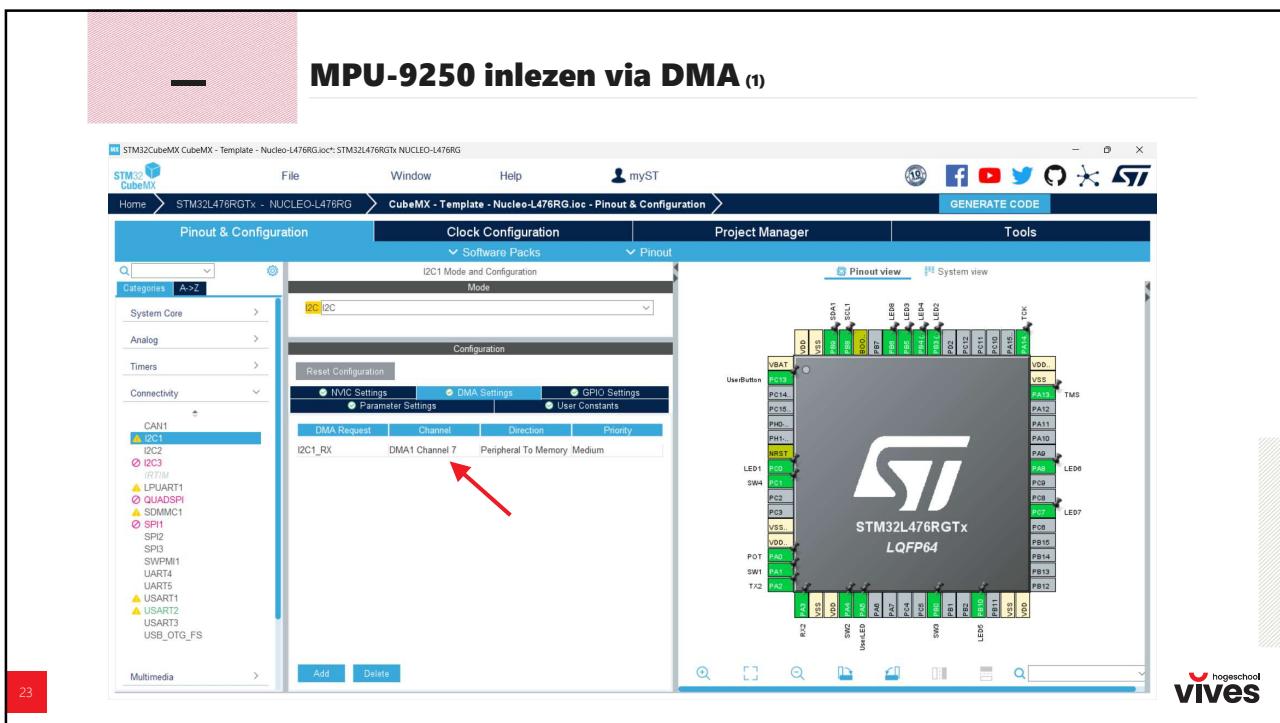
21



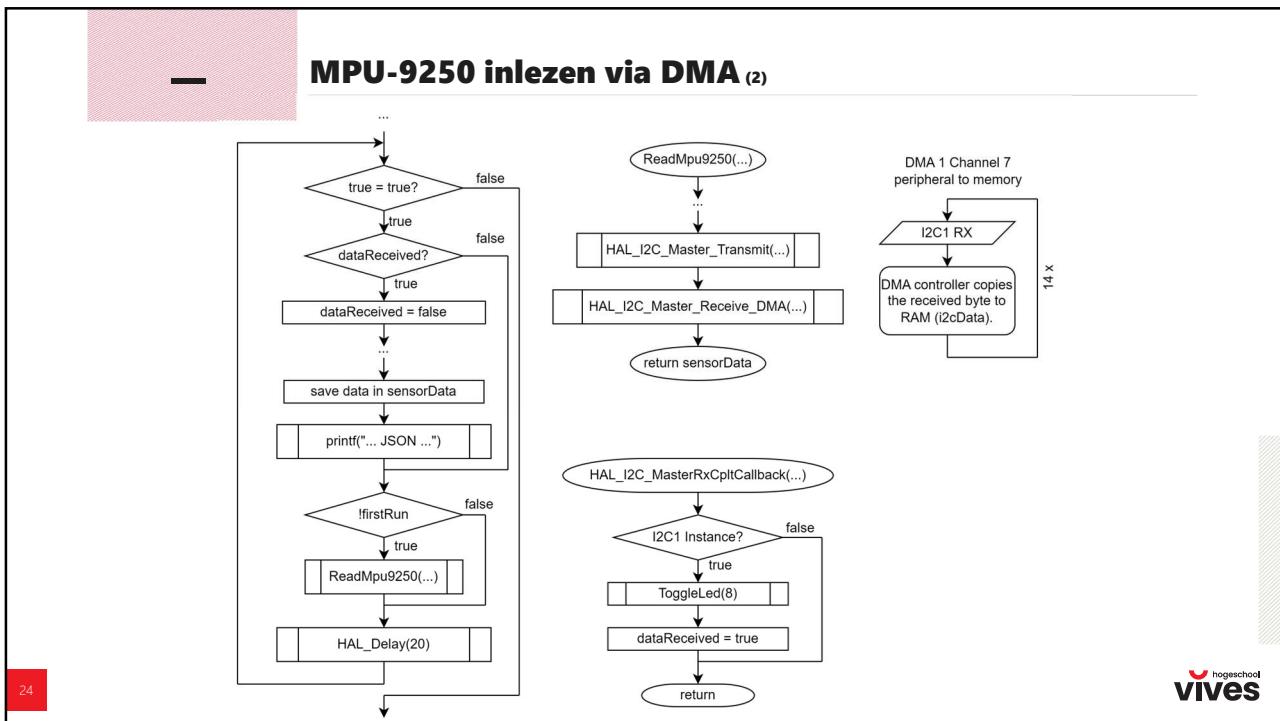
— MPU-9250 inlezen via DMA

22





23



24

MPU-9250 inlezen via DMA (3)

```
127 |     while (1)
128 |     {
129 |         // Is er data ontvangen via DMA, toon ze...
130 |         if(dataReceived)
131 |         {
132 |             dataReceived = false;
133 |
134 |             if(firstRun)
135 |             {
136 |                 firstRun = false;
137 |
138 |                 // Ontvangen Who-am-I-data uitlezen.
139 |                 whoAmI = i2cData[0];
140 |
141 |                 printf("Who am I: %d.\r\n", whoAmI);
142 |             }
143 |             else
144 |             {
145 |                 // Ontvangen IMU-data uitlezen.
146 |                 sensordata.accX = (int16_t)(i2cData[0] * 256 + i2cData[1]); // Casten naar 16 bit signed integer.
147 |                 sensordata.accY = (int16_t)(i2cData[2] * 256 + i2cData[3]);
148 |                 sensordata.accZ = (int16_t)(i2cData[4] * 256 + i2cData[5]);
149 |                 sensordata.temp = (int16_t)(i2cData[6] * 256 + i2cData[7]);
150 |                 sensordata.gyroX = (int16_t)(i2cData[8] * 256 + i2cData[9]);
151 |                 sensordata.gyroY = (int16_t)(i2cData[10] * 256 + i2cData[11]);
152 |                 sensordata.gyroZ = (int16_t)(i2cData[12] * 256 + i2cData[13]);
153 |
154 |                 printf("{\"accx\":%d,\"accy\":%d,\"accz\":%d,\"gyrox\":%d,\"gyroy\":%d,\"gyroz\":%d}\r\n",
155 |                     sensordata.accX, sensordata.accY, sensordata.accZ, sensordata.gyroX, sensordata.gyroY, sensordata.gyroZ);
156 |             }
157 |         }
158 |
159 |         // IMU uitlezen starten via DMA en JSON dumpen indien de 'Who am I' reeds gekend is.
160 |         if(!firstRun)
161 |             ReadMpu9250(&hi2c1);
162 |
163 |         // Uitlezen van de I2C-bus duurt ongeveer 1,5ms (2 + 14 bytes).
164 |         HAL_Delay(20);
165 }
```

25



MPU-9250 inlezen via DMA (4)

```
58 void ReadMpu9250(I2C_HandleTypeDef* hi2c)
59 {
60     extern uint8_t i2cData[NUMBER_OF_BYTES_TO_RECEIVE];
61
62     // Register 0x3B: Acc X High byte.
63     i2cData[0] = 0x3B;
64     HAL_I2C_Master_Transmit(hi2c, (SLAVE_ADDRESS_GYRO_ACC << 1), i2cData, 1, 100);
65
66     // 14 bytes aan IMU-data lezen.
67     HAL_I2C_Master_Receive_DMA(hi2c, (SLAVE_ADDRESS_GYRO_ACC << 1), i2cData, 14);
68 }
```



```
461 // MERK OP: om met HAL I2C en DMA te kunnen werken, moet in STM32CubeMX 'I2C1 event interrupt' aangevinkt staan!
462 // Anders wordt HAL_I2C_MasterRxCallback() niet opgeroepen.
463 void HAL_I2C_MasterRxCallback(I2C_HandleTypeDef *hi2c)
464 {
465     // Is de ontvangst van I2C1?
466     if(hi2c->Instance == I2C1)
467     {
468         ToggleLed(8); // Teken van leven geven...
469         dataReceived = true; // Melden aan de hoofdlus.
470     }
471 }
```

26

Houd het hier kort!



— MPU-9250 inlezen via DMA (5)

Hoe kan je weten dat je met 'niet-blokkerende' functies werkt?
Zoek het op in de HAL User Manual (UM1884).

HAL_I2C_Master_Receive_DMA

Function name

```
HAL_StatusTypeDef HAL_I2C_Master_Receive_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress,  
uint8_t * pData, uint16_t Size)
```

Function description

Receive in master mode an amount of data **in non-blocking mode** with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

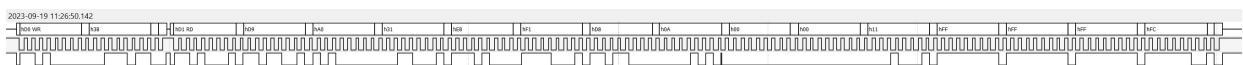
- **HAL:** status

27



— MPU-9250 inlezen via DMA (6)

De scoopbeelden moet er gelijkaardig uitzien als die van de 'polling' versie, maar de CPU heeft zelf veel meer tijd om nuttige berekeningen te maken!



De code in de *main()* ziet er wel complexer uit, maar toch benut deze DMA-versie de microcontroller op een veel betere manier.

28



—

'Head tracking' demo

29

 hogeschool
VIVES

—

Head tracking demo ⁽¹⁾

- In sommige gevallen kan het handig (of leuk) zijn dat een camera mee beweegt met de draaiing van je hoofd. Zeker in FPV-situaties kan dat van belang zijn (FPV-vliegen, -rijden, -varen,...).



<https://www.youtube.com/watch?v=G5nRet-r62s>

30

 hogeschool
VIVES

— Head tracking demo (2)

- Kan je met behulp van een IMU, een modelbouwservo en onze Nucleo, zo'n wannabee 'head tracking' systeem maken?
Natuurlijk!!
- Wat hebben we nodig?
 - I²C-bus (best met DMA),
 - timer met PWM-uitgang,
 - camera, microcontroller, ...

31

— Head tracking demo (3)

```
569 // Verwerken van de sensor data nadat de DMA een signaal (interrupt) geeft.  
570 // MERK OP: om met HAL I2C en DMA te kunnen werken, moet in STM32CubeMX 'I2C1 event interrupt' aangevinkt staan!  
571 // Anders wordt HAL_I2C_MasterRxCpltCallback() niet opgeroepen.  
572 // Het starten van de meting gebeurt in de SysTick_Handler() iedere 4ms.  
573 void HAL_I2C_MasterRxCpltCallback(I2C_HandleTypeDef *hi2c)  
574 {  
575     // Is de ontvangst van I2C1?  
576     if(hi2c->Instance == I2C1)  
577     {  
578         // Teken van leven geven...  
579         ToggleLed(8);  
580  
581         // Gyro Y-data verwerken.  
582         sensordata.gyroY = (int16_t)(i2cData[10] * 256 + i2cData[11]);  
583  
584         // Nieuwe hoek berekenen: hoek = draaisnelheid * draaitijd.  
585         // TODO: nog beter zou zijn dat er met SENSOR FUSION gewerkt wordt (gyrocoop + magnetometer).  
586         // Op die manier kan je de 'drift' uit het signaal filteren.  
587         angle = angle - ((float)(sensordata.gyroY - gyroYOffset)/16.375f) * (float)SENSOR_READ_INTERVAL/1000.0f;  
588         // of  
589         //angle = angle - ((float)(sensordata.gyroY - gyroYOffset)/5458.3f);  
590  
591         // PWM-waarde voor de servo berekenen, herschalen en begrenzen.  
592         pwmValue = PWM_MINIMUM + PWM_MIDPOINT + angle * ANGLE_TO_PWM_SCALE - servoTrim;  
593         if(pwmValue < PWM_MINIMUM)  
594             pwmValue = PWM_MINIMUM;  
595         if(pwmValue > PWM_MAXIMUM)  
596             pwmValue = PWM_MAXIMUM;  
597  
598         // PWM-waarde opslaan in het CCR2-register (als het kalibreren voorbij is).  
599         if(calibrationok)  
600             __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, pwmValue);  
601     }  
602 }
```

32

4 ms (zie stm32l4xx_it.c)

—

Head tracking demo (4)



33

hogeschool
VIVES

— IoT Devices

Hoofdstuk 5: SPI

L. Espeel



(bron: grotendeels van R. Buysschaert)

—

SPI
algemene info

— SPI (1)

- SPI = Serial Peripheral Interface
- Een communicatiebus die data **serieel** verstuurt, is net als I²C een **synchrone** bus. Ze heeft dus een kloksignaal.
Het ritme van de klok bepaalt het ritme van de datatransfert.
- Wordt, naast I²C, typisch gebruikt om te communiceren tussen IC's op éénzelfde print.

3

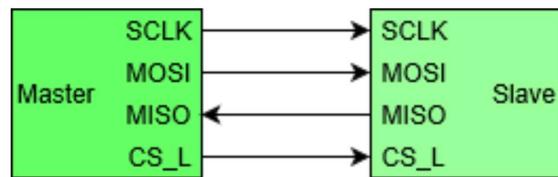
— SPI (2)

- Veelgebruikte naamgevingen:
 - MOSI = Master Output Slave Input
 - MISO = Master Input Slave Output
 - CS = Chip Select
 - SCK = Serial Clock
- Alternatieve naamgevingen:
 - SDO = Serial Data Out (= MOSI)
 - SDI = Serial Data In (= MISO)
 - (N)SS = Slave Select (= CS)

4

SPI (3)

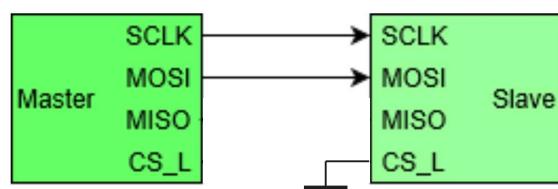
Verschillende opstellingen:



SPI Bus – Simple Point-to-Point
Topology

SPI (4)

Verschillende opstellingen:



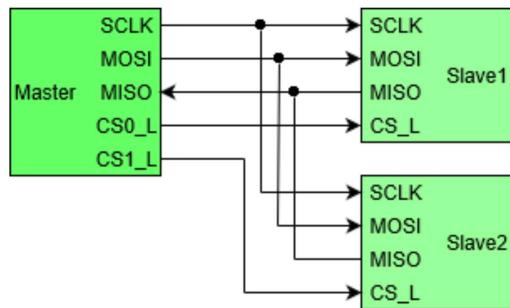
SPI Bus – Simple Point-to-Point
Topology

Opmerking: je kan in sommige gevallen ook enkel data van master naar slave sturen en de chip select constant laag houden...

Sommige IC's hebben zelfs geen chip select (APA102C RGB LED).

SPI (5)

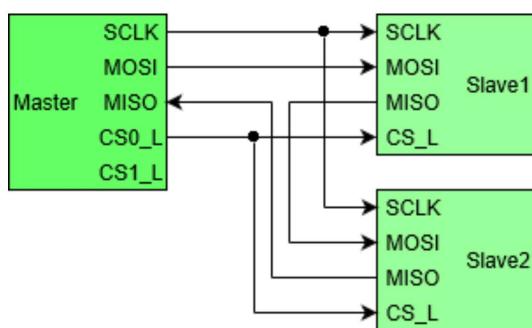
Verschillende opstellingen:



SPI Bus – Star Topology

SPI (6)

Verschillende opstellingen:



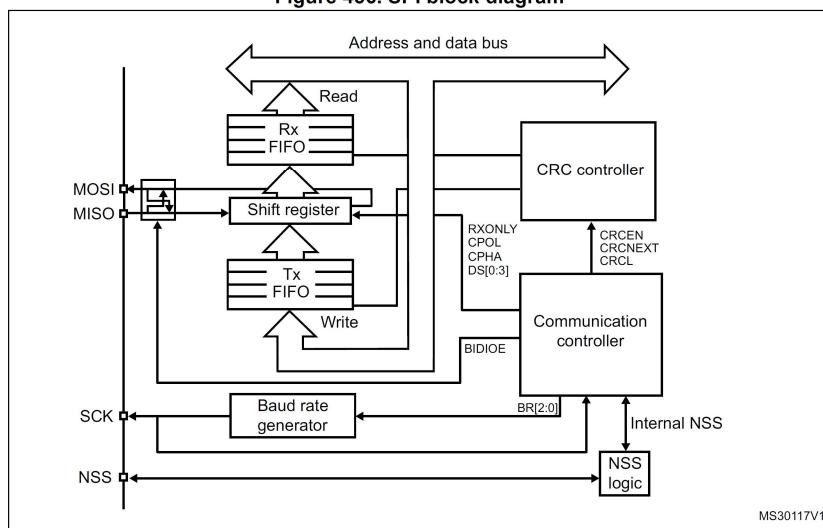
SPI Bus – Daisy Chain Topology

SPI in de STM32L476RG

9

SPI in de STM32L476RG (1)

Figure 456. SPI block diagram

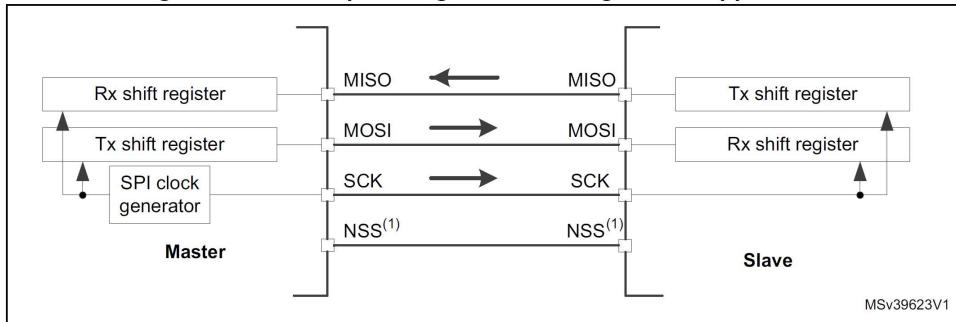


Bron: RM0351 STMicroelectronics.

10

SPI in de STM32L476RG (2)

Figure 457. Full-duplex single master/ single slave application



Bron: RM0351 STMMicroelectronics

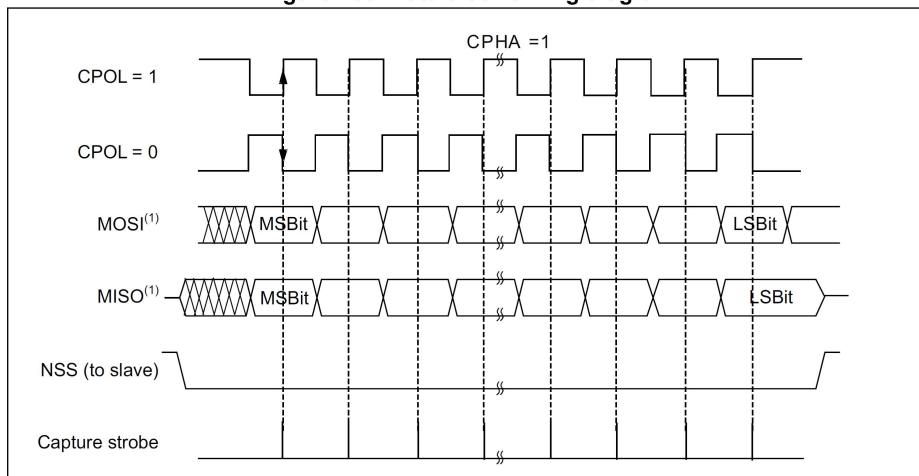
1. The NSS pins can be used to provide a hardware control flow between master and slave. Optionally, the pins can be left unused by the peripheral. Then the flow has to be handled internally for both master and slave. For more details see [Section 42.4.5: Slave select \(NSS\) pin management](#).

SPI in de STM32L476RG (3)

- SPI kan in **4 verschillende modes** communiceren. Die mode wordt ingesteld met de **clock polarity (CPOL)** en **clock phase (CPHA)**.
- Als CPOL nul is, dan is het kloksignaal laag indien er geen data verstuurd wordt. Als CPOL één is, dan is het kloksignaal hoog indien er geen data verstuurd wordt.
- Als CPHA nul is, wordt de eerste flank (stijgend of dalend) van de klok gebruikt om data in te klokken. Als CPHA één is, wordt de tweede flank van de klok gebruikt om data in te klokken.
- Zie afbeelding volgende slides.

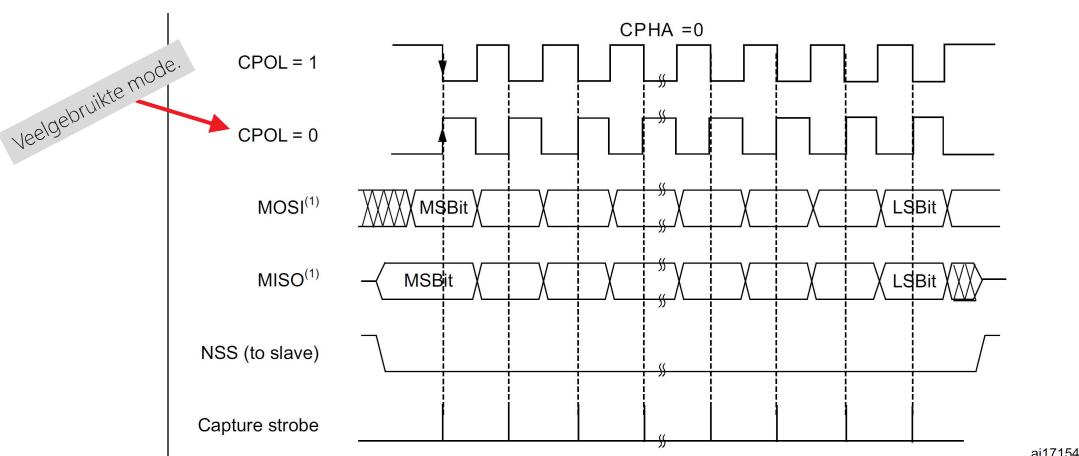
SPI in de STM32L476RG (4)

Figure 463. Data clock timing diagram



Bron: RM0351 STMicroelectronics.

SPI in de STM32L476RG (5)



Bron: RM0351 STMicroelectronics.

SPI in de STM32L476RG (6)

- CPOL en CPHA kan je instellen in het SPIx->CR1 register.

42.6.1 SPI control register 1 (SPIx_CR1)

Address offset: 0x00

Reset value: 0x0000

Waarvoor zou BR[2:0] dienen?

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-----------|--------|--------|----------|------|---------|-----|-----|-----------|-----|---------|----|----|------|------|------|
| BIDI MODE | BIDIOE | CRC EN | CRCN EXT | CRCL | RX ONLY | SSM | SSI | LSB FIRST | SPE | BR[2:0] | | | MSTR | CPOL | CPHA |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bit 1 **CPOL:** Clock polarity

0: CK to 0 when idle

1: CK to 1 when idle

Note: This bit should not be changed when communication is ongoing.

This bit is not used in SPI TI mode except the case when CRC is applied at TI mode.

Bit 0 **CPHA:** Clock phase

0: The first clock transition is the first data capture edge

1: The second clock transition is the first data capture edge

Note: This bit should not be changed when communication is ongoing.

This bit is not used in SPI TI mode except the case when CRC is applied at TI mode.

Bron: RM0351 STMMicroelectronics

15



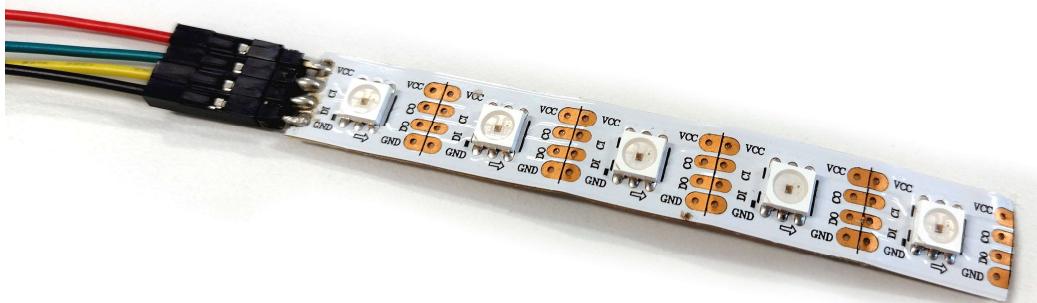
APA102C

16



— APA102C (1)

- De APA102C is een RGB LED die aanstuurbare is via SPI.
- Je kan er één aparte LED mee aansturen of een volledige LED-strip.
- Werkt met een 'start frame' en een 'end frame' (zie datasheet).



17

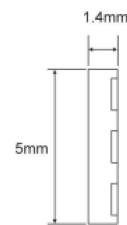
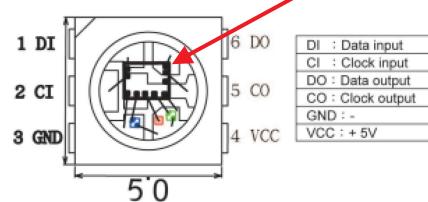
hogeschool
VIVES

— APA102C (2)

- In ieder LED zit een chip die de SPI-communicatie en LED-sturing op zich neemt.
- De uitgang van de ene LED kan gekoppeld worden met de ingang van de andere LED.



PHYSICAL DIMENSIONS



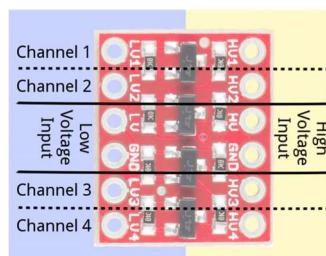
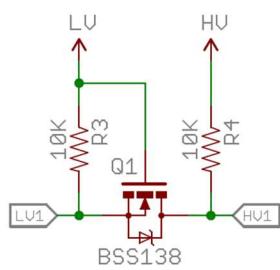
Bron: RM0351 shili-led.com

18

hogeschool
VIVES

Nucleo met APA102C

- Probleem: de LED-strip is 5V gevoed, ook de SPI-lijnen moeten best 5V niveaus behandelen \Leftrightarrow microcontroller waar alles op 3V3 werkt.
- Oplossing: logic level converter tussen microcontroller en LED-strip. Werking?



Hoe verbinden?

Microcontroller -> logic level converter:

- * MOSI met LV1
- * CLK met LV2
- * 3V3 met LV
- * GND's

Logic level converter -> LED-strip:

- * HV1 met DI
- * HV2 met CI
- * HV met 5V
- * GND's

Opmerking:

nog extra aftakking nodig naar 5V voeding!

APA102C aansturen via polling

— APA102C aansturen via polling (1)

- De eenvoudigste manier om de LED's aan te sturen is via blokkerende functies, via polling dus.

HAL_SPI_Transmit

Function name

```
HAL_StatusTypeDef HAL_SPI_Transmit (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size,  
uint32_t Timeout)
```

Function description

Transmit an amount of **data** in blocking mode.

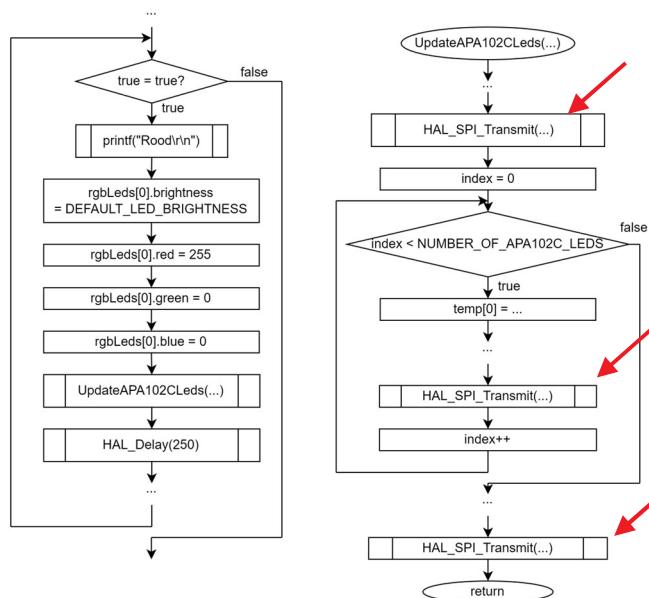
Parameters

- hspi**: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- pData**: pointer to data buffer
- Size**: amount of data to be sent
- Timeout**: Timeout duration

Return values

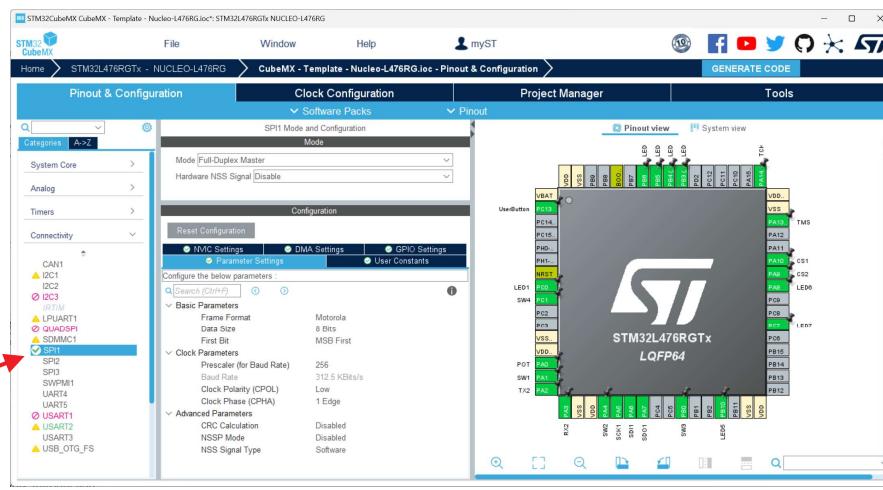
- HAL**: status

— APA102C aansturen via polling (2)



APA102C aansturen via polling (3)

- Stel de SPI-module in via STM32CubeMX:



APA102C aansturen via polling (4)

```
1 #include "stm32l4xx_hal.h"
2 #include "apa102c.h"
3
4 // Verstuur de data over SPI1. Let op: zorg dat SPI1 reeds geïnitialiseerd is.
5 void UpdateAPA102CLeds(SPI_HandleTypeDef* hspi, APA102C led[])
6{
7     uint8_t index = 0;
8     uint8_t temp[] = {0, 0, 0, 0};
9
10    HAL_SPI_Transmit(hspi, temp, sizeof(temp), 100);      // Start frame.
11
12    for(index = 0; index < NUMBER_OF_APAL02C_LEDS; index++)
13    {
14        temp[0] = lcd[index].brightness | 0xE0;           // 5-bit intensiteit.
15        temp[1] = lcd[index].blue;                         // Blauw.
16        temp[2] = lcd[index].green;                        // Groen.
17        temp[3] = lcd[index].red;                          // Rood.
18
19        HAL_SPI_Transmit(hspi, temp, sizeof(temp), 100);
20    }
21
22    temp[0] = 255;
23    temp[1] = 255;
24    temp[2] = 255;
25    temp[3] = 255;
26    HAL_SPI_Transmit(hspi, temp, sizeof(temp), 100);      // End frame.
27}
```

— APA102C aansturen via DMA

25



— APA102C aansturen via DMA (1)

- De beste manier om de LED's aan te sturen is via DMA-functies, niet-blokkerende functies dus.

`HAL_SPI_Transmit_DMA`

Function name

`HAL_StatusTypeDef HAL_SPI_Transmit_DMA (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)`

Function description

Transmit an amount of data in non-blocking mode with DMA.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent

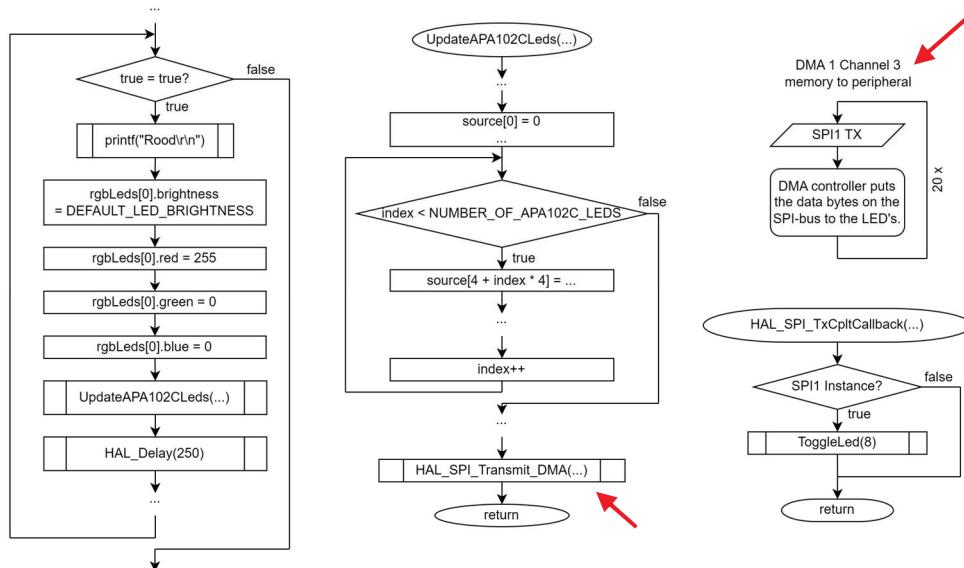
Return values

- **HAL:** status

26



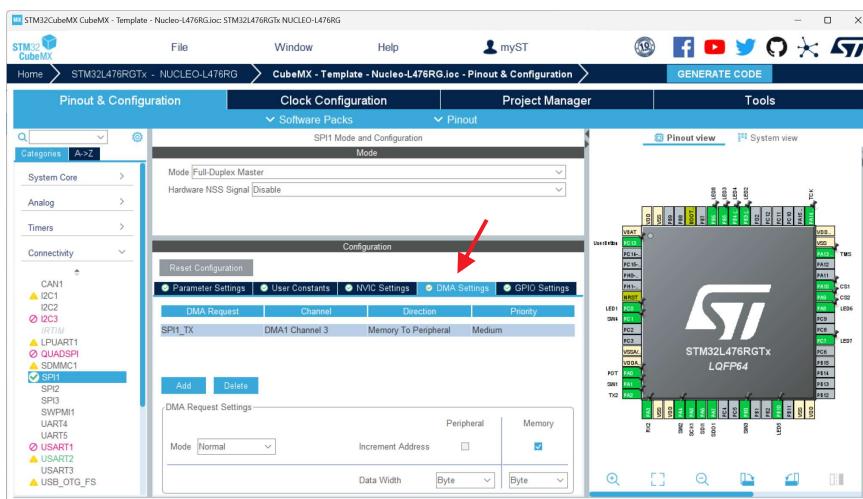
APA102C aansturen via DMA (2)



27

APA102C aansturen via DMA (3)

- Stel de SPI-module in via STM32CubeMX:



28

APA102C aansturen via DMA (4)

```
1 #include "stm3214xx_hal.h"
2 #include "apa102c.h"
3
4 // Verstuur de data over SPI1. Let op: zorg dat SPI1 reeds geinitialiseerd is.
5 void UpdateAPA102CLeds(SPI_HandleTypeDef* hspi, APA102C led[])
6{
7     uint8_t index = 0;
8     extern uint8_t source[4 + NUMBER_OF_APAL02C_LEDS * 4 + 4];
9
10    source[0] = 0;      // Start frame.
11    source[1] = 0;
12    source[2] = 0;
13    source[3] = 0;
14
15    for(index = 0; index < NUMBER_OF_APAL02C_LEDS; index++)
16    {
17        source[4 + index * 4] = led[index].brightness | 0xE0;           // 5-bit intensiteit.
18        source[5 + index * 4] = led[index].blue;                         // Blauw.
19        source[6 + index * 4] = led[index].green;                        // Groen.
20        source[7 + index * 4] = led[index].red;                          // Rood.
21    }
22
23    source[4 + NUMBER_OF_APAL02C_LEDS * 4] = 255;                      // End frame.
24    source[5 + NUMBER_OF_APAL02C_LEDS * 4] = 255;
25    source[6 + NUMBER_OF_APAL02C_LEDS * 4] = 255;
26    source[7 + NUMBER_OF_APAL02C_LEDS * 4] = 255;
27
28    // Alle data verzenden via DMA.
29    HAL_SPI_Transmit_DMA(hspi, source, (4 + NUMBER_OF_APAL02C_LEDS * 4 + 4));
30 }
```

29

APA102C aansturen via DMA (5)

```
597 /* USER CODE BEGIN 4 */
598 void HAL_SPI_TxCpltCallback(SPI_HandleTypeDef * hspi)
599{
600     if(hspi->Instance == SPI1)
601     {
602         ToggleLed(8);                                // Teken van leven geven...
603     }
604 }
605 /* USER CODE END 4 */
```

30

APA102C aansturen via DMA (6)

- Hieronder het scoopbeeld voor het aansturen van 5 APA102C LED's.
- Kan je het 'protocol' erin terugvinden?

