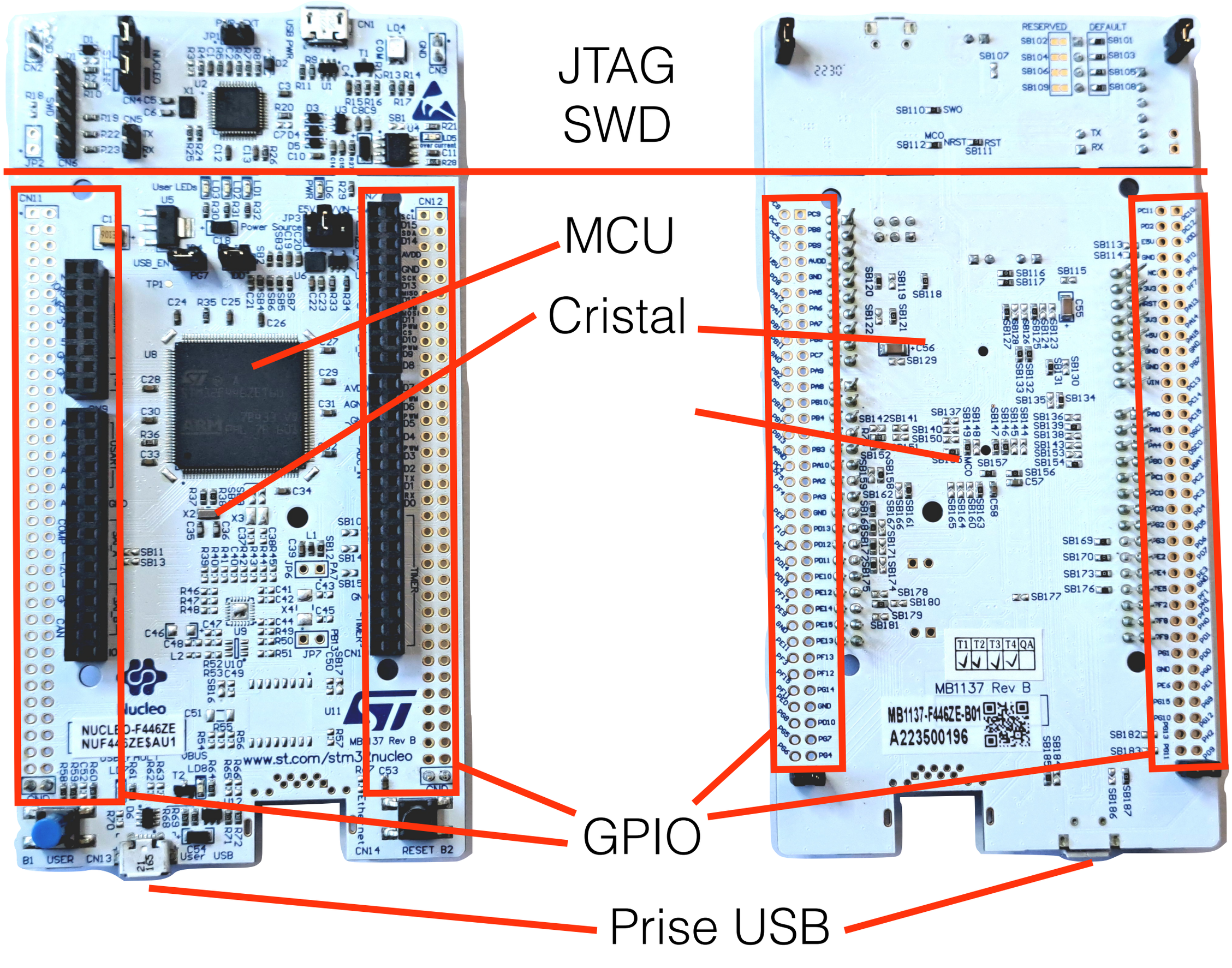
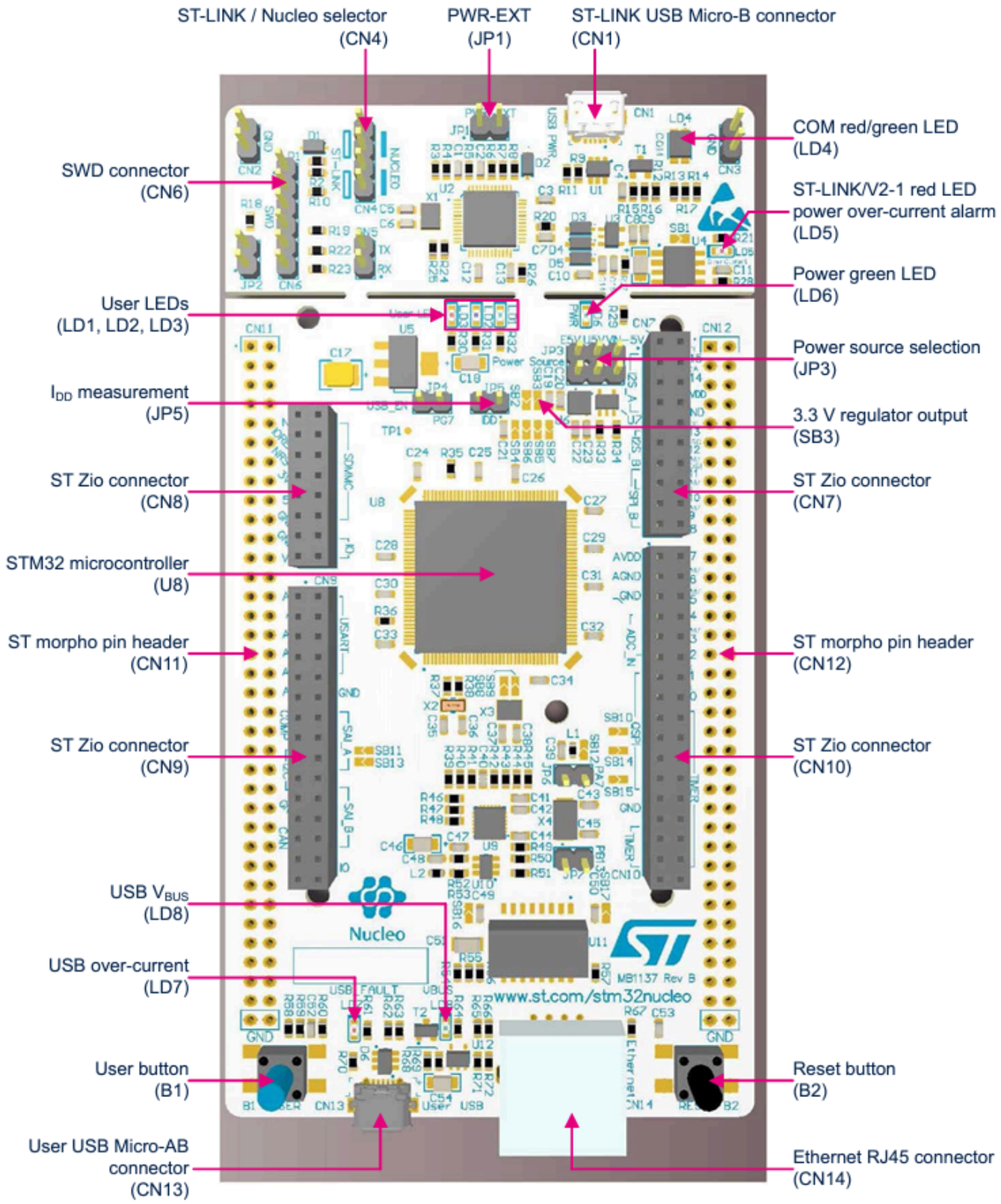
Microcontrôleurs - Arduino

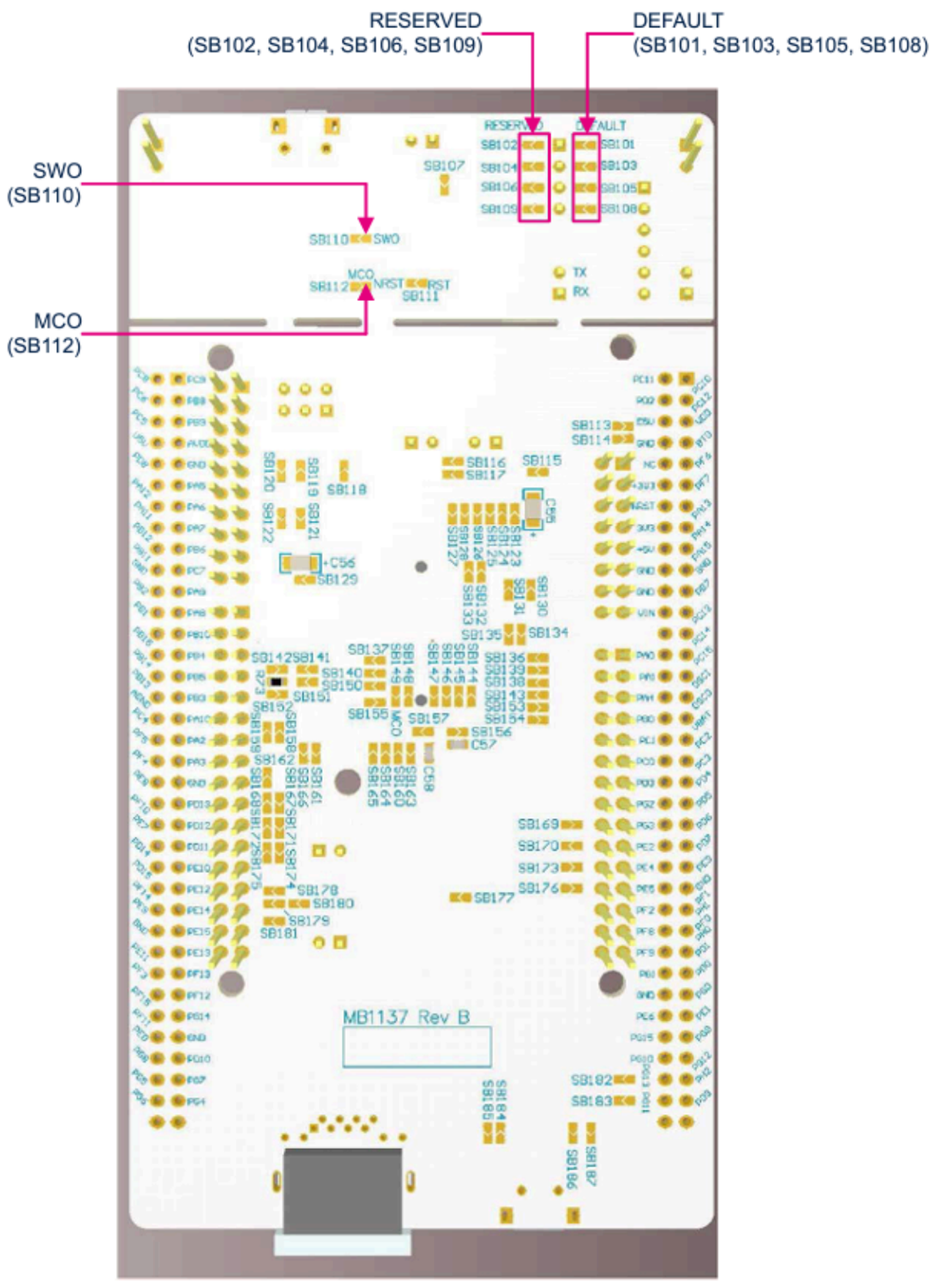
Thomas Pietrzak
Master 2 Informatique — RVA

Université de Lille

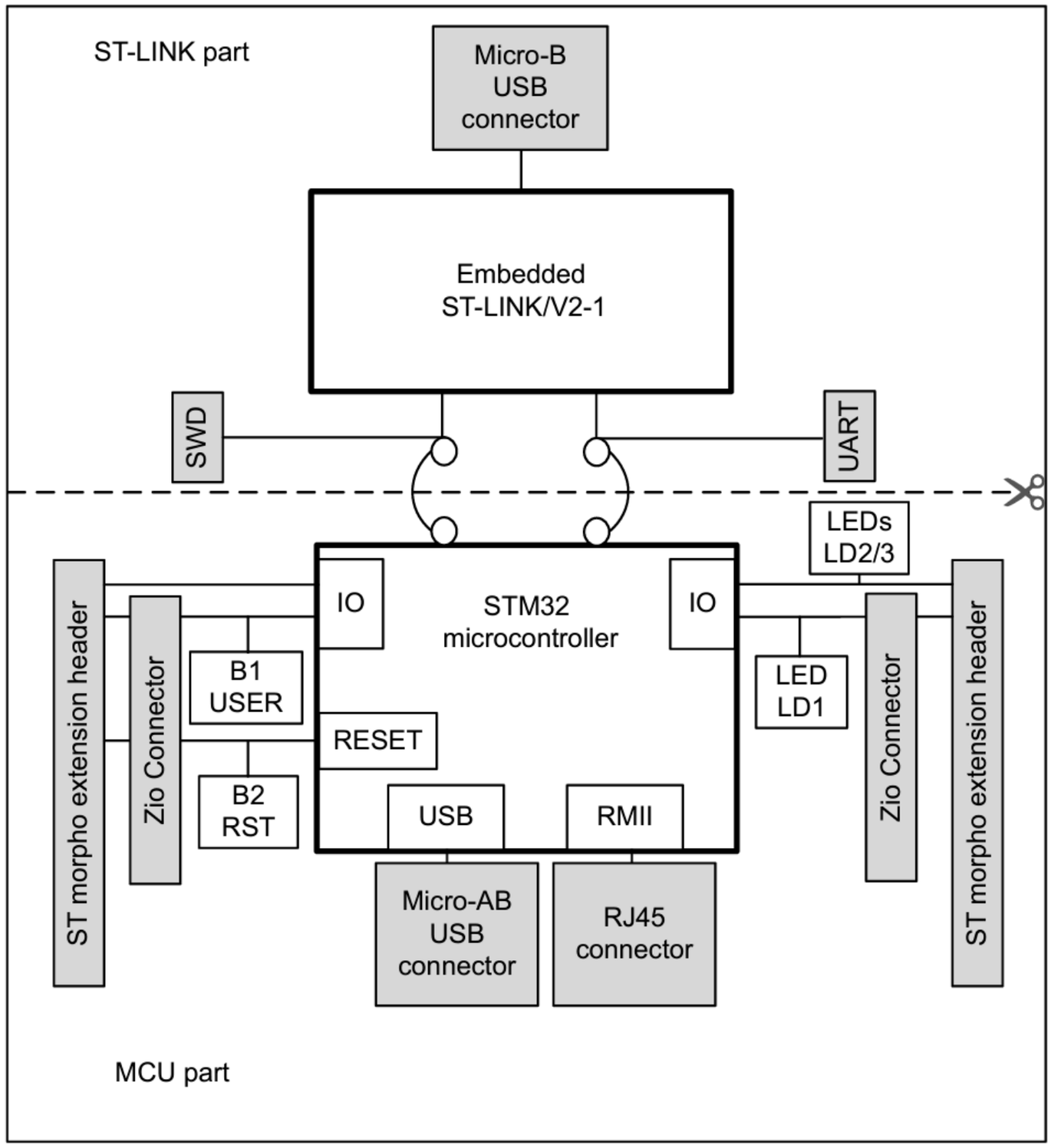

JTAG
SWD
MCU
Cristal
GPIO
Prise USB

Carte NUCLEO 144 - F446ZE





Block diagram



LD1 : vert, PB0/PA5

LD2 : bleu, PB7

LD3 : rouge, PB14

B1 : PC13

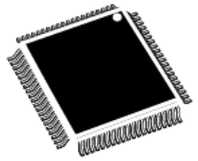
Pins

- VCC, VDD, V+ : alimentation (+)
- VEE, VSS, V-, GND : alimentation (-) ou masse
- BOOT0/BT0 : sélection de zone de démarrage
- RESET/NRESET : reset du circuit
- PXN (X = lettre, N = chiffre) : GPIO

Microcontrôleur

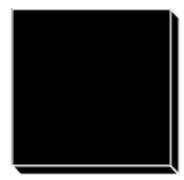
STM32F446RE

- Arm® 32-bit Cortex®-M4 CPU with FPU
- 512 Ko mémoire flash, 128 Ko SRAM
- 3 × ADC 12-bit, jusqu'à 24 channels
- 2 × DAC 12-bit
- 17 timers
- JTAG/SWD
- 114 GPIO avec interrupt
- 4 × I2C
- 4 × USART + 2 × UART
- 4 × SPI
- 2 × SAI
- 2 × CAN
- USB 2.0 device/host/OTG
- RTC
- …



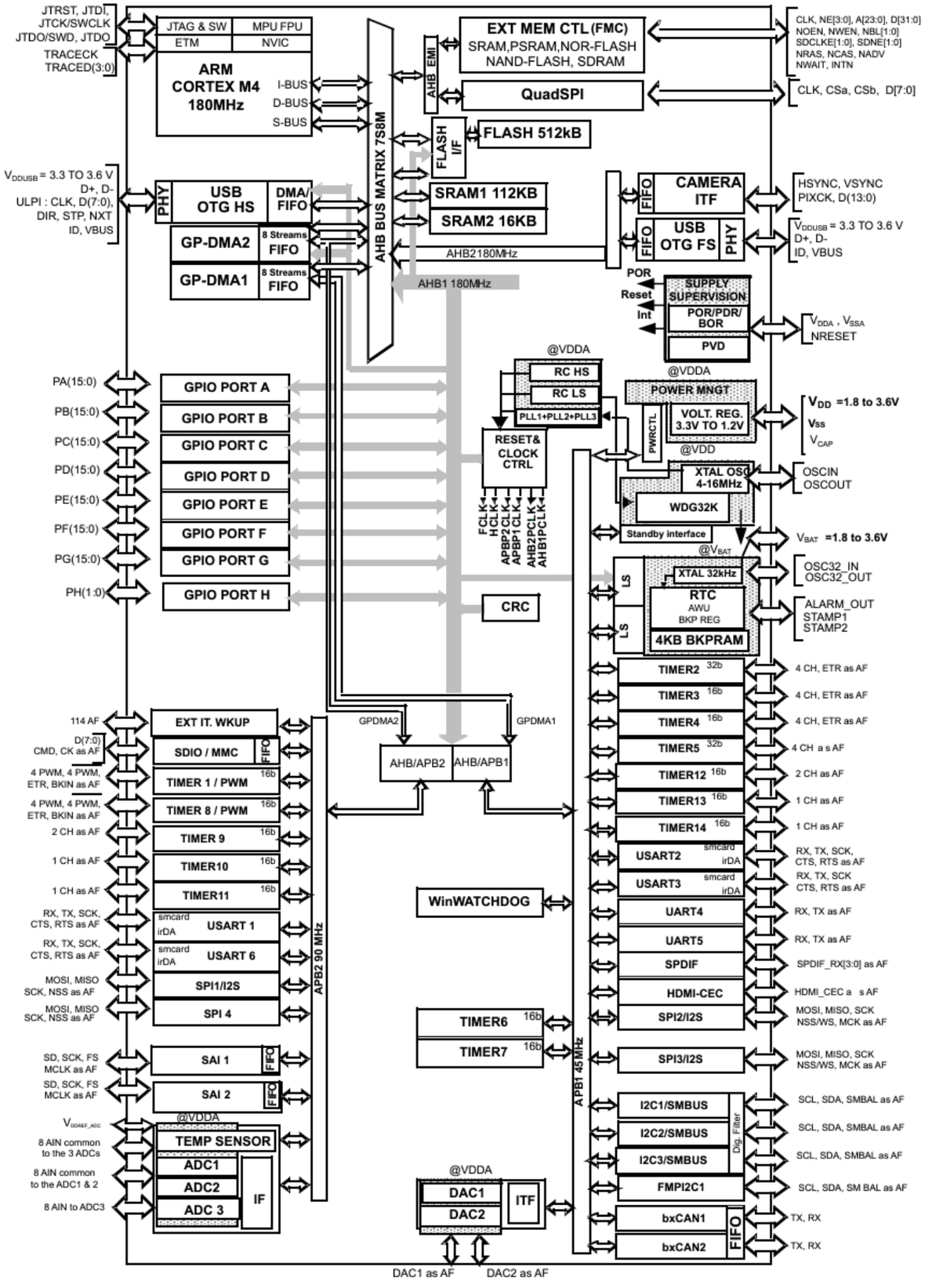
LQFP64 (10 × 10 mm)
LQFP100 (14 × 14 mm)
LQFP144 (20 x 20 mm)

WLCSP 81

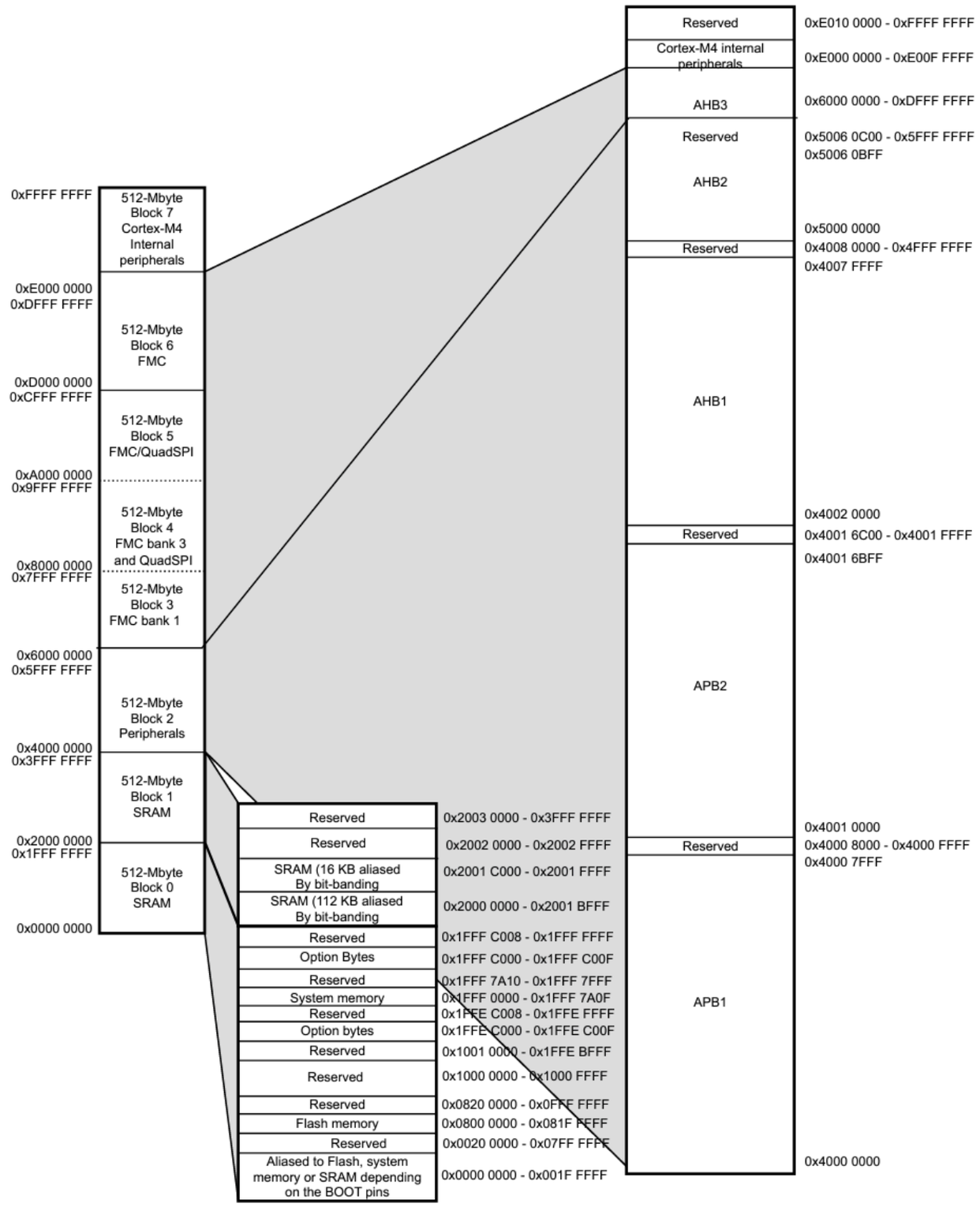


UFBGA144 (7 x 7 mm)
UFBGA144 (10 x 10 mm)

Clocks


JTRST, JTDI, JTCK/SWCLK JTDO/SWD, JTDO TRACECK TRACED(3:0)
JTAG & SW
ETM
MPU FPU
NVIC
ARM CORTEX M4 180MHz
I-BUS
D-BUS
S-BUS
AHB BUS MATRIX 7S8M
AHB EMI
EXT MEM CTL (FMC) SRAM,PSRAM,NOR-FLASH NAND-FLASH, SDRAM
CLK, NE[3:0], A[23:0], D[31:0] NOEN, NWEN, NBL[1:0] SDCLKE[1:0], SDNE[1:0] NRAS, NCAS, NADV NWAIT, INTN
QuadSPI
CLK, CSa, CSb, D[7:0]
FLASH I/F
FLASH 512kB
VDDUSB = 3.3 TO 3.6 V D+, D- ULPI : CLK, D(7:0), DIR, STP, NXT ID, VBUS
PHY
USB OTG HS
DMA/ FIFO
SRAM1 112KB
SRAM2 16KB
GP-DMA2
8 Streams FIFO
GP-DMA1
8 Streams FIFO
AHB2180MHz
AHB1 180MHz
FIFO
CAMERA ITF
HSYNC, VSYNC PIXCK, D(13:0)
FIFO
USB OTG FS
PHY
VDDUSB = 3.3 TO 3.6 V D+, D- ID, VBUS
POR
Reset
Int
SUPPLY SUPERVISION
POR/PDR/ BOR
PVD
VDDA , VSSA NRESET
@VDDA
RC HS
RC LS
PLL1+PLL2+PLL3
RESET& CLOCK CTRL
@VDDA
POWER MNGT
VOLT. REG. 3.3V TO 1.2V
PWRCTL
VDD =1.8 to 3.6V VSS VCAP
@VDD
XTAL OSC 4-16MHz
WDG32K
OSCIN OSCOUT
Standby interface
@VBAT
VBAT =1.8 to 3.6V
LS
LS
XTAL 32kHz
RTC AWU BKP REG
4KB BKPRAM
OSC32_IN OSC32_OUT
ALARM_OUT STAMP1 STAMP2
FCLK HCLK APBP2 CLK APBP1 CLK AHB2PCLK AHB1PCLK
PA(15:0)
PB(15:0)
PC(15:0)
PD(15:0)
PE(15:0)
PF(15:0)
PG(15:0)
PH(1:0)
GPIO PORT A
GPIO PORT B
GPIO PORT C
GPIO PORT D
GPIO PORT E
GPIO PORT F
GPIO PORT G
GPIO PORT H
CRC
TIMER2 32b
TIMER3 16b
TIMER4 16b
TIMER5 32b
TIMER12 16b
TIMER13 16b
TIMER14 16b
4 CH, ETR as AF
4 CH, ETR as AF
4 CH, ETR as AF
4 CH a s AF
2 CH as AF
1 CH as AF
1 CH as AF
114 AF
D(7:0) CMD, CK as AF
4 PWM, 4 PWM, ETR, BKIN as AF
4 PWM, 4 PWM, ETR, BKIN as AF
2 CH as AF
1 CH as AF
1 CH as AF
RX, TX, SCK, CTS, RTS as AF
RX, TX, SCK, CTS, RTS as AF
MOSI, MISO SCK, NSS as AF
MOSI, MISO SCK, NSS as AF
EXT IT. WKUP
SDIO / MMC
FIFO
TIMER 1 / PWM
TIMER 8 / PWM
TIMER 9
TIMER10
TIMER11
smcard irDA
USART 1
smcard irDA
USART 6
SPI1/I2S
SPI 4
GPDMA2
GPDMA1
AHB/APB2
AHB/APB1
APB2 90 MHz
WinWATCHDOG
TIMER6 16b
TIMER7 16b
APB1 45 MHz
USART2 smcard irDA
USART3 smcard irDA
UART4
UART5
SPDIF
HDMI-CEC
SPI2/I2S
RX, TX, SCK, CTS, RTS as AF
RX, TX, SCK CTS, RTS as AF
RX, TX as AF
RX, TX as AF
SPDIF_RX[3:0] as AF
HDMI_CEC a s AF
MOSI, MISO, SCK NSS/WS, MCK as AF
SD, SCK, FS MCLK as AF
SD, SCK, FS MCLK as AF
SAI 1
FIFO
SAI 2
FIFO
SPI3/I2S
MOSI, MISO, SCK NSS/WS, MCK as AF
VDDREF_ADC
@VDDA
8 AIN common to the 3 ADCs
8 AIN common to the ADC1 & 2
8 AIN to ADC3
TEMP SENSOR
ADC1
ADC2
ADC 3
IF
I2C1/SMBUS
I2C2/SMBUS
I2C3/SMBUS
FMPI2C1
Dig. Filter
SCL, SDA, SMBAL as AF
SCL, SDA, SMBAL as AF
SCL, SDA, SMBAL as AF
SCL, SDA, SM BAL as AF
@VDDA
DAC1
DAC2
ITF
bxCAN1
bxCAN2
FIFO
TX, RX
TX, RX
DAC1 as AF
DAC2 as AF

Memory map

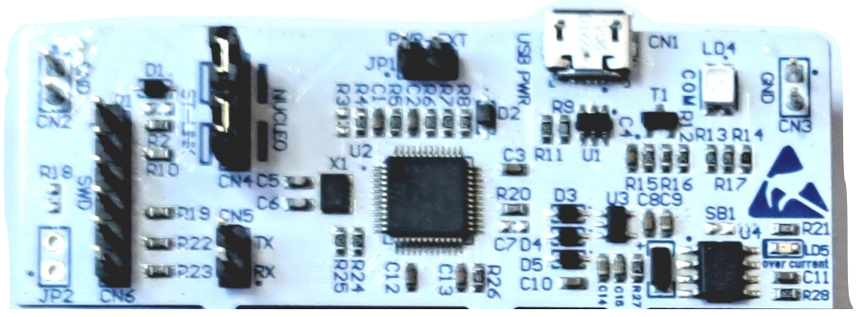

0xFFFF FFFF
512-Mbyte Block 7 Cortex-M4 Internal peripherals
0xE000 0000
0xDFFF FFFF
512-Mbyte Block 6 FMC
0xD000 0000
0xCFFF FFFF
512-Mbyte Block 5 FMC/QuadSPI
0xA000 0000
0x9FFF FFFF
512-Mbyte Block 4 FMC bank 3 and QuadSPI
0x8000 0000
0x7FFF FFFF
512-Mbyte Block 3 FMC bank 1
0x6000 0000
0x5FFF FFFF
512-Mbyte Block 2 Peripherals
0x4000 0000
0x3FFF FFFF
512-Mbyte Block 1 SRAM
0x2000 0000
0x1FFF FFFF
512-Mbyte Block 0 SRAM
0x0000 0000
Reserved
0x2003 0000 - 0x3FFF FFFF
Reserved
0x2002 0000 - 0x2002 FFFF
SRAM (16 KB aliased By bit-banding
0x2001 C000 - 0x2001 FFFF
SRAM (112 KB aliased By bit-banding
0x2000 0000 - 0x2001 BFFF
Reserved
0x1FFF C008 - 0x1FFF FFFF
Option Bytes
0x1FFF C000 - 0x1FFF C00F
Reserved
0x1FFF 7A10 - 0x1FFF 7FFF
System memory
0x1FFF 0000 - 0x1FFF 7A0F
Reserved
0x1FFE C008 - 0x1FFE FFFF
Option bytes
0x1FFE C000 - 0x1FFE C00F
Reserved
0x1001 0000 - 0x1FFE BFFF
Reserved
0x1000 0000 - 0x1000 FFFF
Reserved
0x0820 0000 - 0x0FFF FFFF
Flash memory
0x0800 0000 - 0x081F FFFF
Reserved
0x0020 0000 - 0x07FF FFFF
Aliased to Flash, system memory or SRAM depending on the BOOT pins
0x0000 0000 - 0x001F FFFF
Reserved
0xE010 0000 - 0xFFFF FFFF
Cortex-M4 internal peripherals
0xE000 0000 - 0xE00F FFFF
AHB3
0x6000 0000 - 0xDFFF FFFF
Reserved
0x5006 0C00 - 0x5FFF FFFF
0x5006 0BFF
AHB2
0x5000 0000
Reserved
0x4008 0000 - 0x4FFF FFFF
0x4007 FFFF
AHB1
0x4002 0000
Reserved
0x4001 6C00 - 0x4001 FFFF
0x4001 6BFF
APB2
0x4001 0000
Reserved
0x4000 8000 - 0x4000 FFFF
0x4000 7FFF
APB1
0x4000 0000

Pins

Table 10. STM32F427xx and STM32F429xx pin and ball definitions (continued)

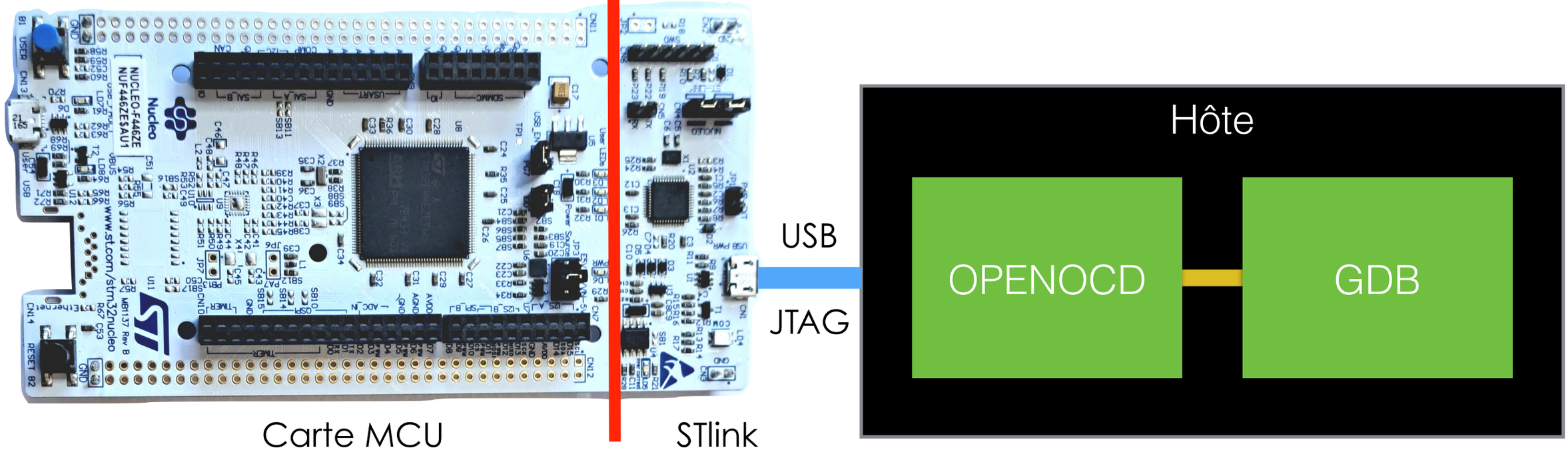
Pin number								Pin name (function after reset)[1]	Pin type	I / O structure	Notes	Alternate functions	Additional functions
LQFP100	LQFP144	UFBGA169	UFBGA176	LQFP176	WLCSP143	LQFP208	TFBGA216						
-	10	F2	E2	16	F11	16	D2	PF0	I/O	FT		I2C2_SDA, FMC_A0, EVENTOUT	
-	11	F3	H3	17	E9	17	E2	PF1	I/O	FT		I2C2_SCL, FMC_A1, EVENTOUT	
-	12	G5	H2	18	F10	18	G2	PF2	I/O	FT		I2C2_SMBA, FMC_A2, EVENTOUT	
-	-	-	-	-	-	19	E3	PI12	I/O	FT		LCD_HSYNC, EVENTOUT	
-	-	-	-	-	-	20	G3	PI13	I/O	FT		LCD_VSYNC, EVENTOUT	
-	-	-	-	-	-	21	H3	PI14	I/O	FT		LCD_CLK, EVENTOUT	
-	13	G4	J2	19	G11	22	H2	PF3	I/O	FT	(5)	FMC_A3, EVENTOUT	ADC3_IN9
-	14	G3	J3	20	F9	23	J2	PF4	I/O	FT	(5)	FMC_A4, EVENTOUT	ADC3_ IN14
-	15	H3	K3	21	F8	24	K3	PF5	I/O	FT	(5)	FMC_A5, EVENTOUT	ADC3_ IN15
10	16	G7	G2	22	H7	25	H6	V_{SS}	S				
11	17	G8	G3	23	-	26	H5	V_{DD}	S				
-	18	NC (2)	K2	24	G10	27	K2	PF6	I/O	FT	(5)	TIM10_CH1, SPI5_NSS, SAI1_SD_B, UART7_Rx, FMC_NIORD, EVENTOUT	ADC3_IN4

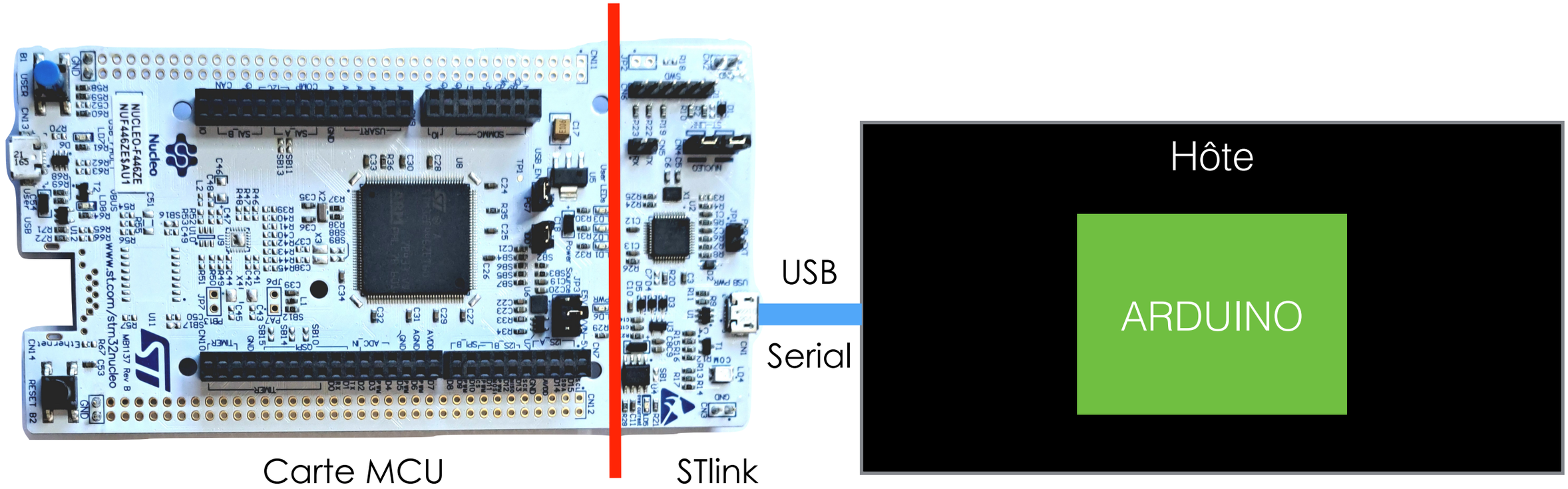
JTAG/SWD



- Programmation
- Débuggage pas à pas

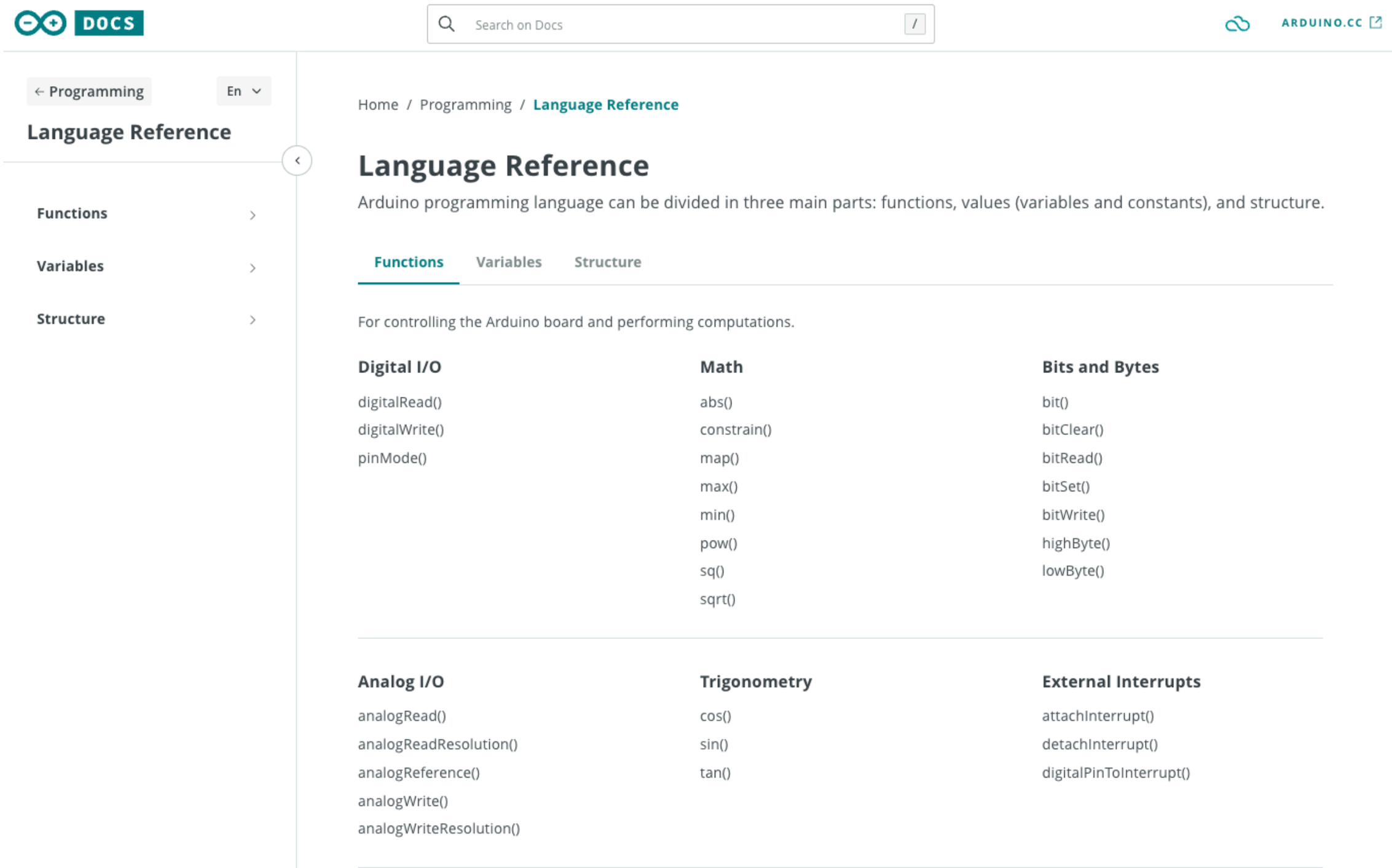
Débuggage


Carte MCU
STlink
USB
JTAG
Hôte
OPENOCD
GDB

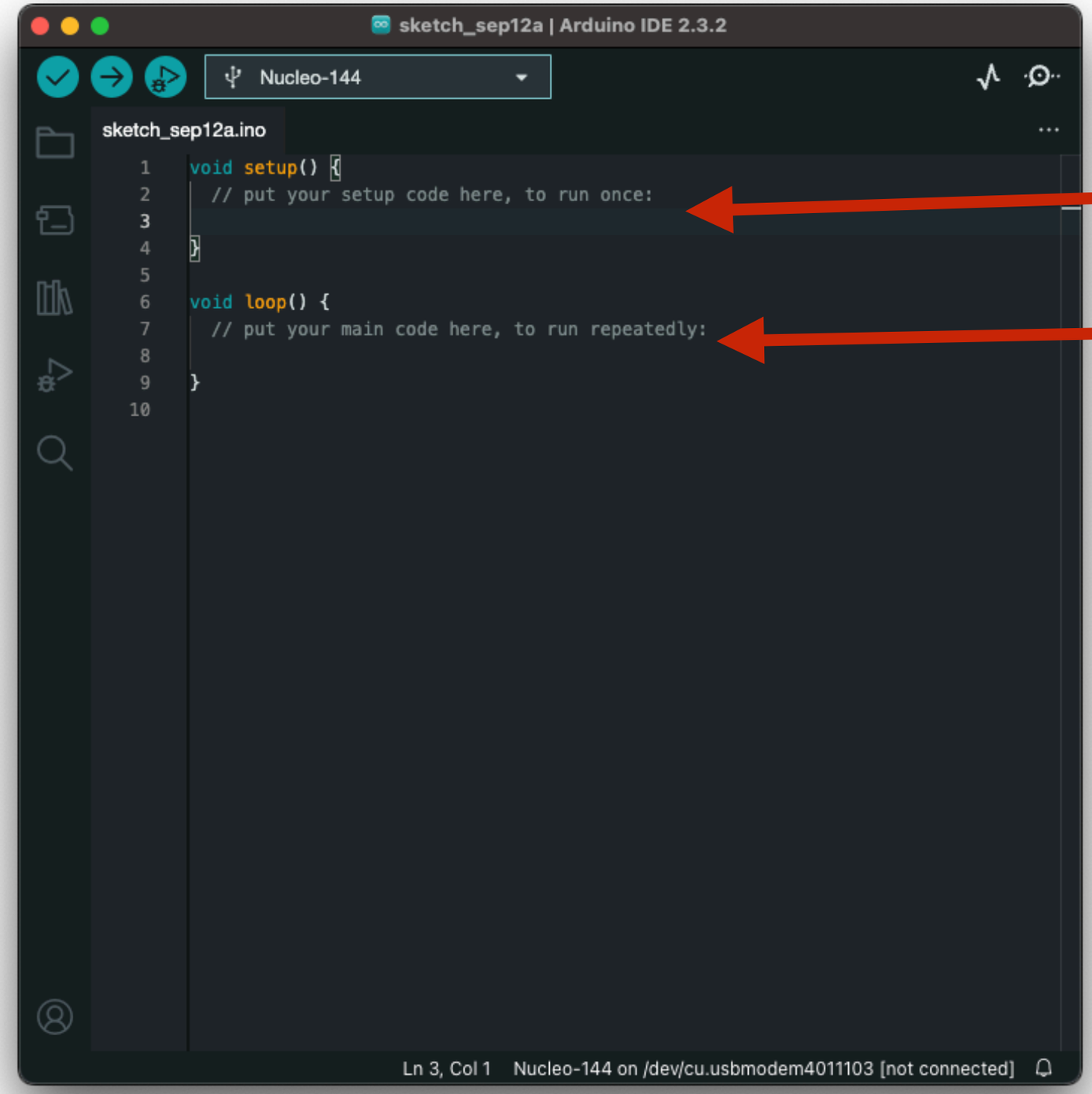

Carte MCU
STlink
USB
Serial
Hôte
ARDUINO

Arduino SDK

https://docs.arduino.cc/language-reference/



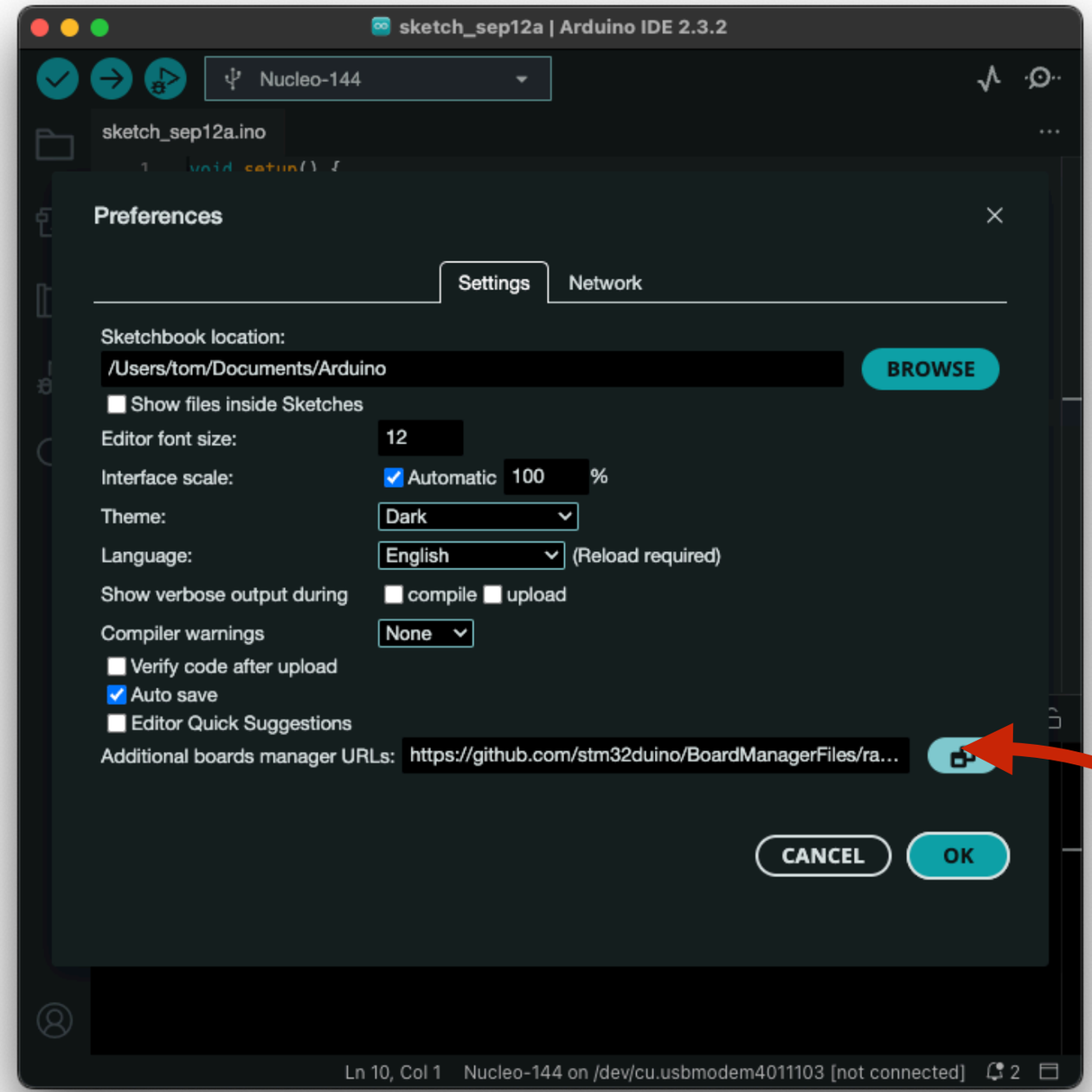
Arduino IDE



Code d'initialisation

Code qui s'exécute en boucle

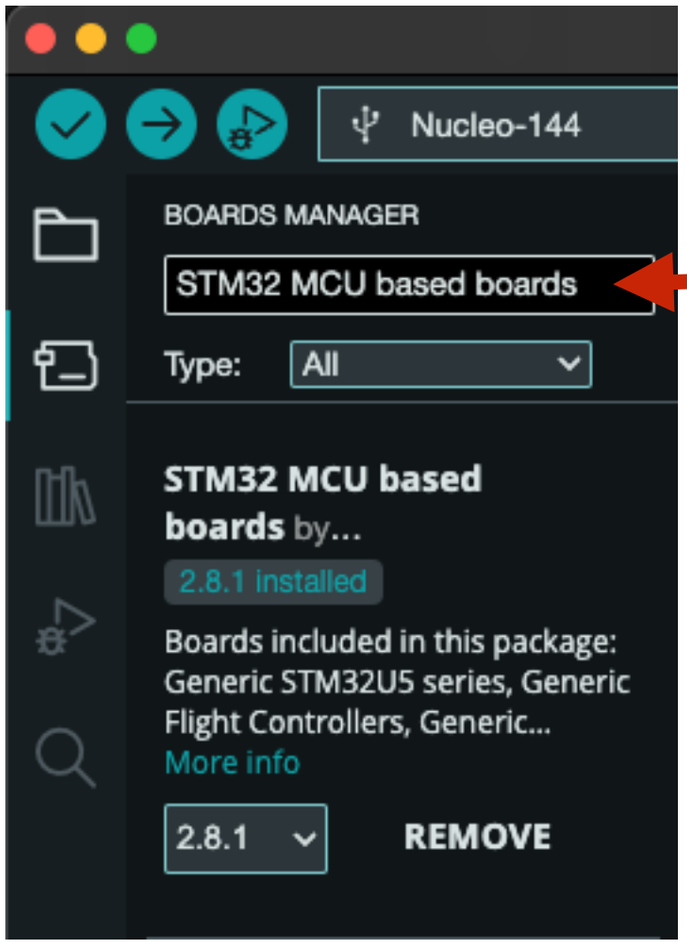
Librairie



Ajouter

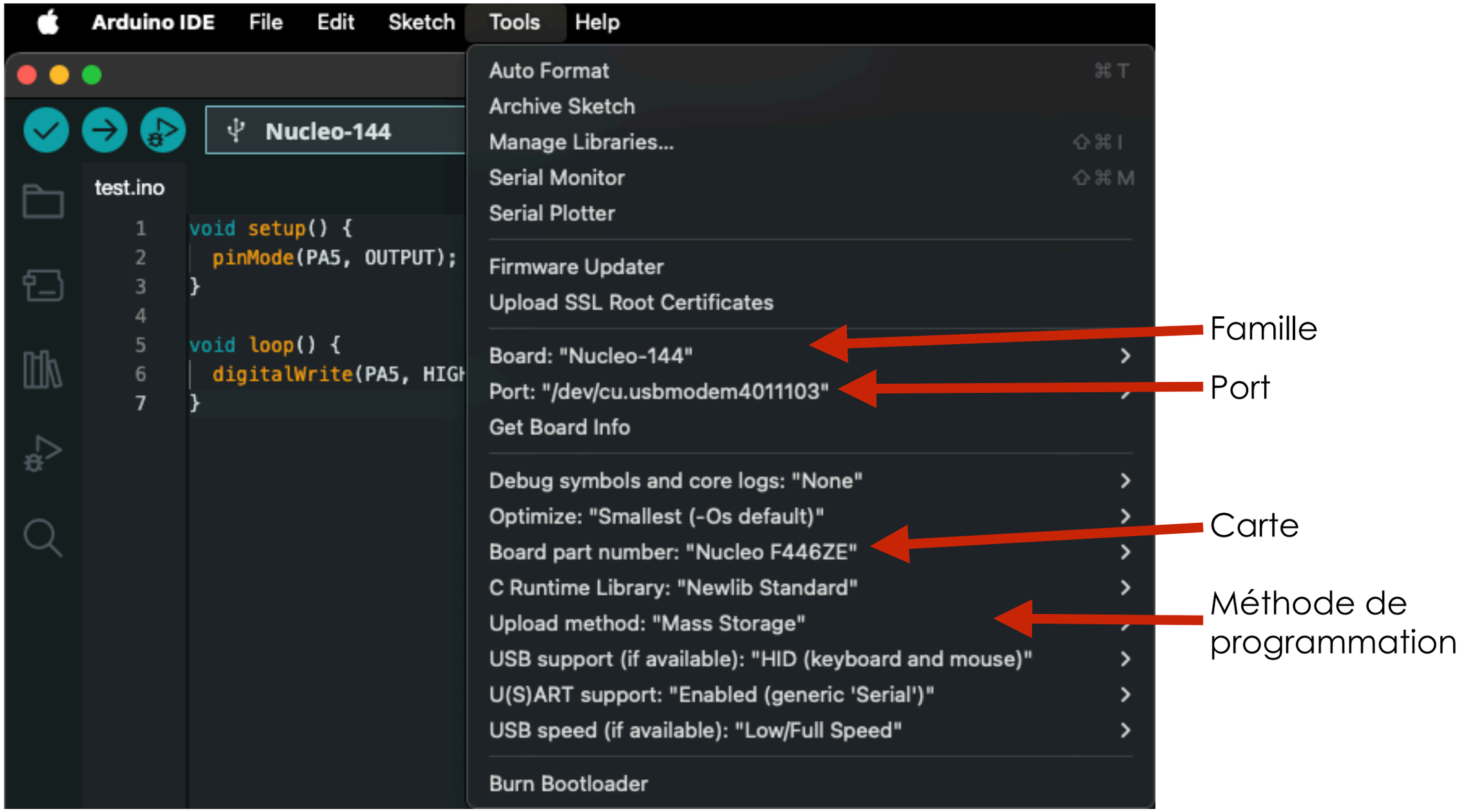
`https://github.com/stm32duino/BoardManagerFiles/raw/main/package_stmicroelectronics_index.json`

Installer la collection de cartes



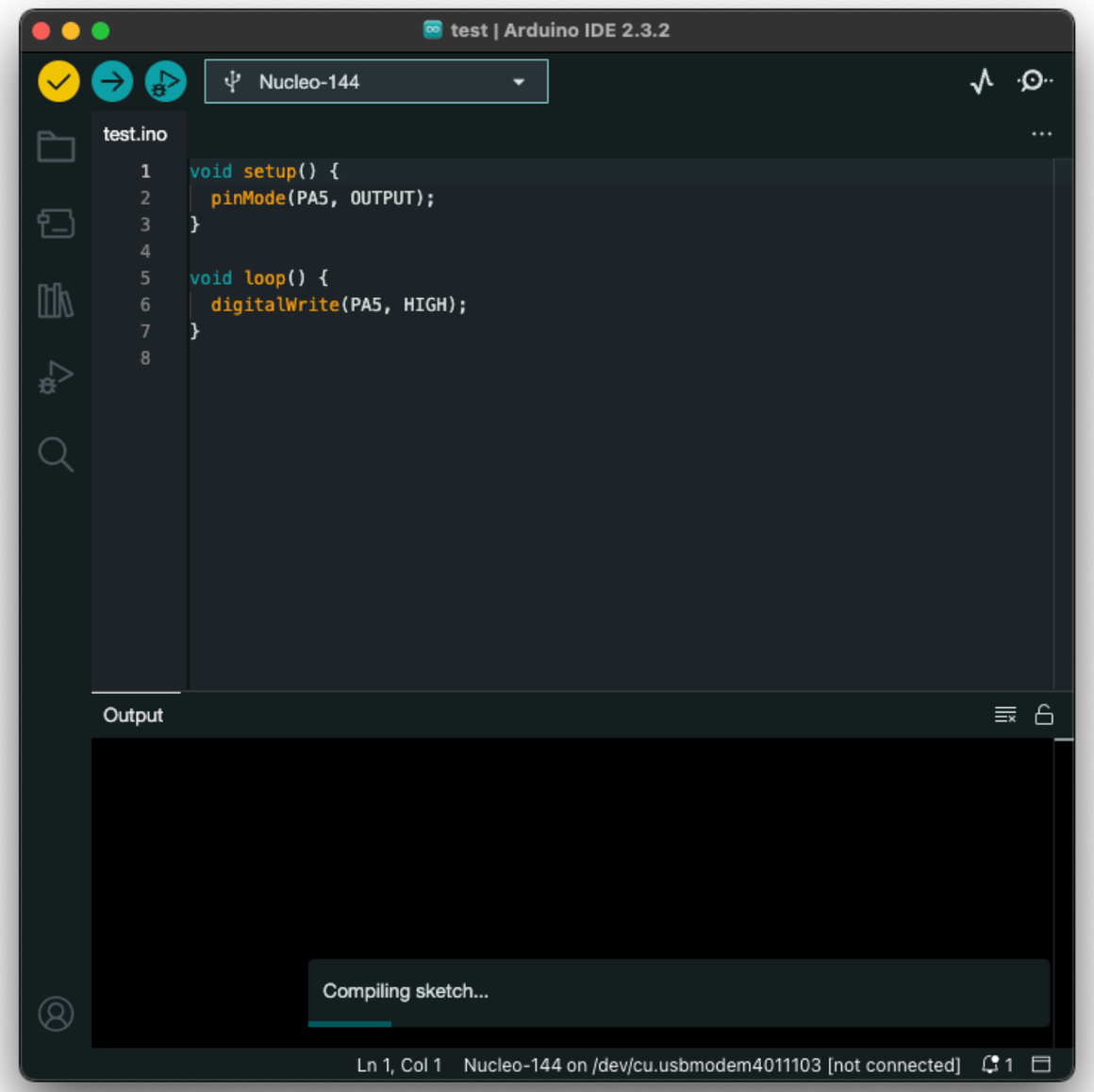
Cherchez ceci

Configuration de la carte

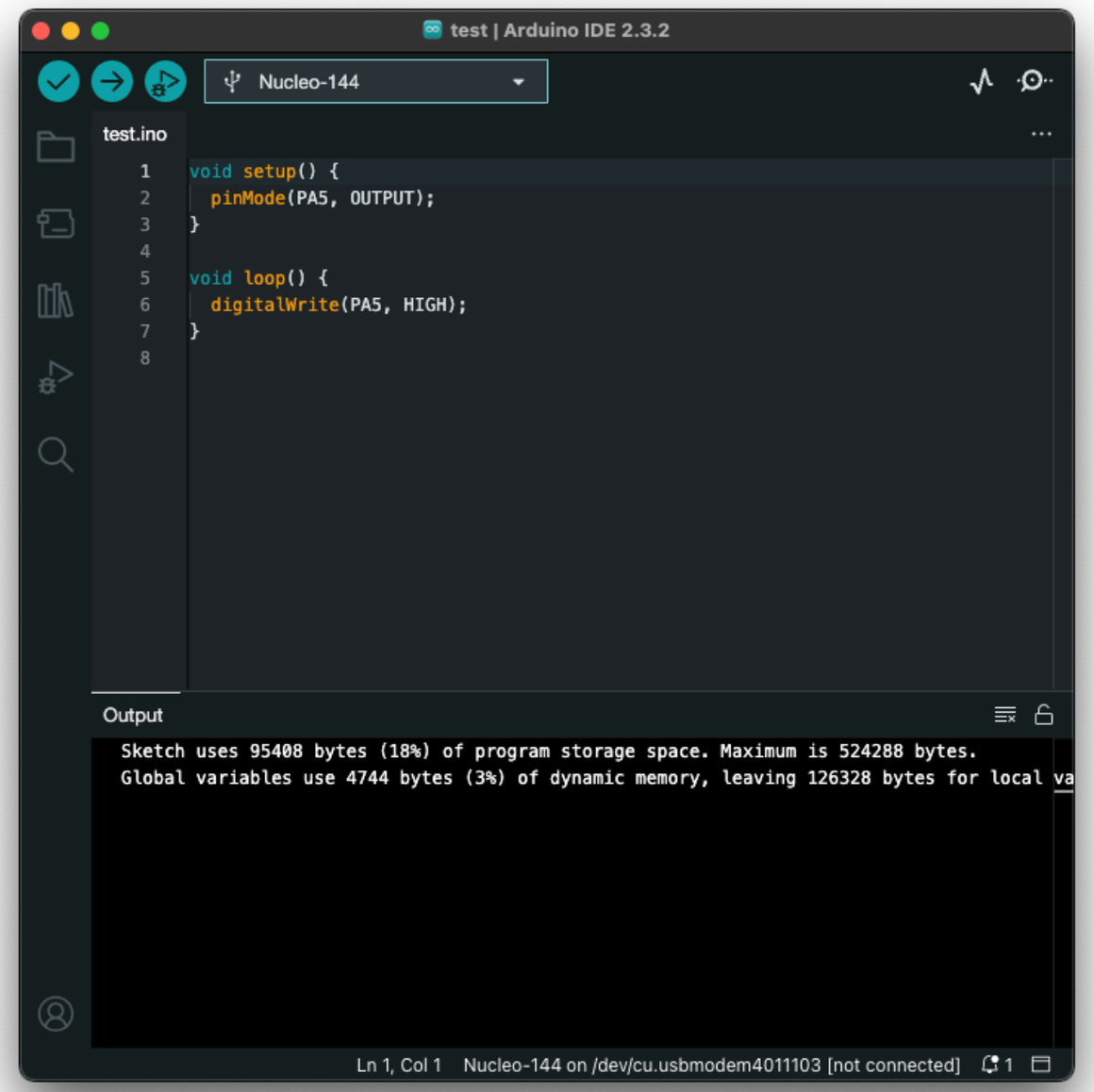


Exemple

Compilation

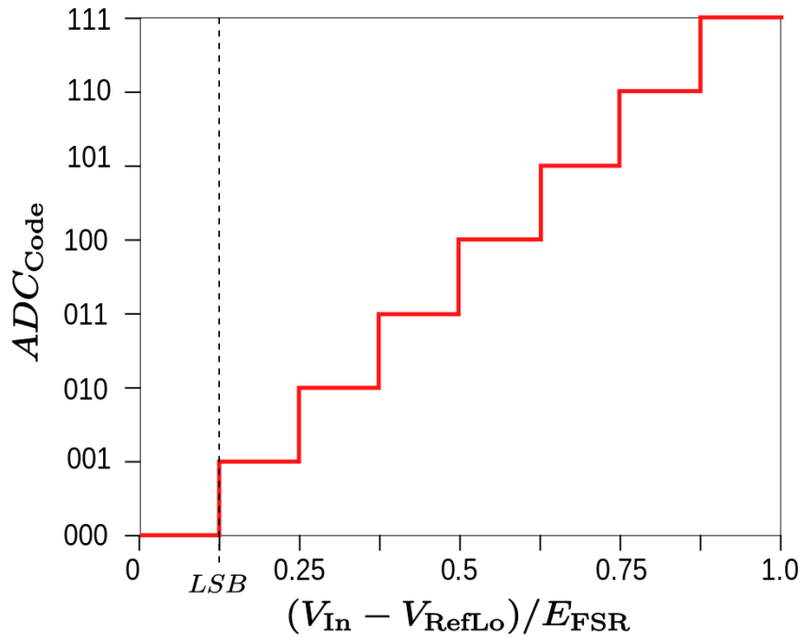


Ça compile

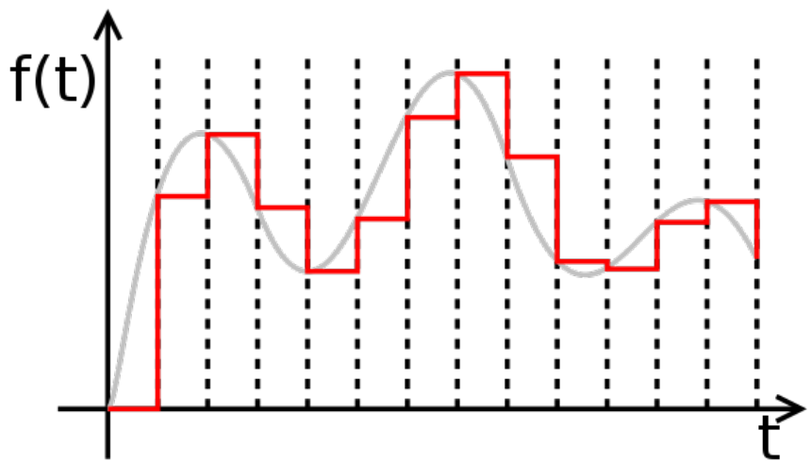


GPIO

- General purpose Input Output
- **Digital** : bit en *Input* ou *Output*
- **Analogique** :
  - Input : ADC
  - Output : DAC



http://en.wikipedia.org/wiki/Analog-to-digital_converter



http://en.wikipedia.org/wiki/Digital-to-analog_converter

Ports et registres

- Data
  - Input
  - Output
- Configuration
  - Mode
  - Output type
  - Output speed
  - Pull up/pull down
  - Write
  - Lock
  - Alternate function

Bus AHB1

0x4002 1C00 - 0x4002 1FFF	GPIOH
0x4002 1800 - 0x4002 1BFF	GPIOG
0x4002 1400 - 0x4002 17FF	GPIOF
0x4002 1000 - 0x4002 13FF	GPIOE
0X4002 0C00 - 0x4002 0FFF	GPIOD
0x4002 0800 - 0x4002 0BFF	GPIOC
0x4002 0400 - 0x4002 07FF	GPIOB
0x4002 0000 - 0x4002 03FF	GPIOA

Utilisation - Arduino

Configuration

```
pinMode(pin, mode)
```

- mode : INPUT, OUTPUT, ou INPUT_PULLUP

Écriture

```
digitalWrite(pin, value)
```

- Value : HIGH ou LOW

Lecture

```
digitalRead(pin)
```

- retourne HIGH ou LOW

Interruption

```
attachInterrupt(digitalPinToInterrupt(pin), callback, mode)
```

- mode : LOW, CHANGE, RISING, ou FALLING

Exemple output : LED

```
void setup() {
  pinMode(PB0, OUTPUT);
}

void loop() {
  digitalWrite(PB0, HIGH);
}
```

Exemple input : bouton

```
const int buttonPin PC13;
const int ledPin PB0;

int buttonState = 0;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  buttonState = digitalRead(buttonPin);
  digitalWrite(ledPin, buttonState);
}
```

Exemple input : bouton

```
const int buttonPin PC8;
const int ledPin PB0;

bool buttonPressed = false;

void buttonCallback() {
  buttonPressed ^= true; // toggle
}

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(buttonPin), buttonCallback, FALLING);
}

void loop() {
  digitalWrite(ledPin, buttonPressed?HIGH:LOW);
}
```

Autres fonctions

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/11 CEC	I2C1/2/3 /4/CEC	SPI1/2/3/4	SPI2/3/4/ SAI1	SPI2/3/ USART1/2/3 /UART5/ SPDIFRX	SAI/ USART6/ UART4/5/ SPDIFRX	CAN1/2 TIM12/13/ 14/ QUADSPI	SAI2/ QUADSPI/ OTG2_HS/ OTG1_FS	OTG1_FS	FMC/ SDIO/ OTG2_FS	DCMI	-	SYS
A	PA0	-	TIM2_CH1/ TIM2_ETR	TIM5_CH1	TIM8_ETR	-	-	-	USART2_ CTS	UART4_ TX	-	-	-	-	-	-	EVENT OUT
	PA1	-	TIM2_CH2	TIM5_CH2	-	-	-	-	USART2_ RTS	UART4_ RX	QUADSPI_ BK1_IO3	SAI2_ MCLK_B	-	-	-	-	EVENT OUT
	PA2	-	TIM2_CH3	TIM5_CH3	TIM9_CH1	-	-	-	USART2_ TX	SAI2_ SCK_B	-	-	-	-	-	-	EVENT OUT
	PA3	-	TIM2_CH4	TIM5_CH4	TIM9_CH2	-	-	SAI1_ FS_A	USART2_ RX	-	-	OTG_HS_ ULPI_D0	-	-	-	-	EVENT OUT
	PA4	-	-	-	-	-	SPI1_NSS/I 2S1_WS	SPI3_NSS / I2S3_WS	USART2_ CK	-	-	-	-	OTG_HS_ SOF	DCMI_ HSYNC	-	EVENT OUT
	PA5	-	TIM2_CH1/ TIM2_ETR	-	TIM8_ CH1N	-	SPI1_SCK/I 2S1_CK	-	-	-	-	OTG_HS_ ULPI_CK	-	-	-	-	EVENT OUT
	PA6	-	TIM1_ BKIN	TIM3_CH1	TIM8_ BKIN	-	SPI1_MISO	I2S2_ MCK	-	-	TIM13_CH1	-	-	-	DCMI_ PIXCLK	-	EVENT OUT
	PA7	-	TIM1_ CH1N	TIM3_CH2	TIM8_ CH1N	-	SPI1_MOSI / I2S1_SD	-	-	-	TIM14_CH1	-	-	FMC_ SDNWE	-	-	EVENT OUT
	PA8	MCO1	TIM1_CH1	-	-	I2C3_ SCL	-	-	USART1_ CK	-	-	OTG_FS_ SOF	-	-	-	-	EVENT OUT
	PA9	-	TIM1_CH2	-	-	I2C3_ SMBA	SPI2_SCK /I2S2_CK	SAI1_ SD_B	USART1_ TX	-	-	-	-	-	DCMI_D0	-	EVENT OUT
	PA10	-	TIM1_CH3	-	-	-	-	-	USART1_ RX	-	-	OTG_FS_ ID	-	-	DCMI_D1	-	EVENT OUT
	PA11	-	TIM1_CH4	-	-	-	-	-	USART1_ CTS	-	CAN1_RX	OTG_FS_ DM	-	-	-	-	EVENT OUT
	PA12	-	TIM1_ETR	-	-	-	-	-	USART1_ RTS	SAI2_ FS_B	CAN1_TX	OTG_FS_ DP	-	-	-	-	EVENT OUT
	PA13	JTMS-SWDIO	-	-	-	-	-	-	-	-	-	-	-	-	-	-	EVENT OUT
	PA14	JTCK-SWCLK	-	-	-	-	-	-	-	-	-	-	-	-	-	-	EVENT OUT
	PA15	JTDI	TIM2_CH1/ TIM2_ETR	-	-	HDMI_ CEC	SPI1_NSS/ I2S1_WS	SPI3_ NSS/ I2S3_WS	-	UART4_RT S	-	-	-	-	-	-	EVENT OUT

Timers

- Compteurs sous forme de registre
  - Fréquence (fixe)
  - Pre-scale
  - Taille de registre : propre à chaque timer
- Usages :
  - PWM : générer un signal sur un pin
  - Output compare : programmer des actions

Timers

Timer type	Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/ compare channels	Complementary output	Max interface clock (MHz)	Max timer clock (MHz)(1)
Advanced-control	TIM1, TIM8	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	Yes	90	180
General purpose	TIM2, TIM5	32-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	45	90/180
	TIM3, TIM4	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	45	90/180
	TIM9	16-bit	Up	Any integer between 1 and 65536	No	2	No	90	180
	TIM10, TIM11	16-bit	Up	Any integer between 1 and 65536	No	1	No	90	180
	TIM12	16-bit	Up	Any integer between 1 and 65536	No	2	No	45	90/180
	TIM13, TIM14	16-bit	Up	Any integer between 1 and 65536	No	1	No	45	90/180
Basic	TIM6, TIM7	16-bit	Up	Any integer between 1 and 65536	Yes	0	No	45	90/180

1. The maximum timer clock is either 90 or 180 MHz depending on TIMPRE bit configuration in the RCC_DCKCFGR register.

Timers

$$F = \frac{F_T}{\text{prescale} \times (\text{count} + 1)}$$

- Chaque signal de clock incrémente le compteur de prescale
- Si le compte est atteint, retour à 0 et incrémentation du count
- Si le compte est atteint, retour à 0, interruptions, etc.

Prescale = 2, count = 4

Clock	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Prescale	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
Count	0	0	1	1	2	2	3	3	4	4	0	0	1	1	2	2	3

Timers

$$F = \frac{F_T}{\text{prescale} \times (\text{count} + 1)}$$

Exemple

F_T = 180MHz

Taille de registre 16 bits :

count max : 65535

prescale max : 65535

On veut 10Hz

Timers

$$F = \frac{F_T}{\text{prescale} \times (\text{count} + 1)}$$

Exemple

F_T = 180MHz

Taille de registre 16 bits :

count max : 65535

prescale max : 65535

$$prescale \geq \frac{180.10^6}{10 \times 2^{16}}$$

$$prescale \simeq 300$$

On veut 10Hz

$$count = \frac{180.10^6}{300 \times 10} - 1$$

$$count = 59999$$

Timers

$$F = \frac{F_T}{\text{prescale} \times (\text{count} + 1)}$$

Exemple

F_T = 180MHz

Taille de registre 16 bits :

count max : 65535

prescale max : 65535

On veut 10Hz

$$prescale \geq \frac{180.10^6}{10 \times 2^{16}}$$

$$prescale \simeq 300$$

$$count = \frac{180.10^6}{300 \times 10} - 1$$

$$count = 59999$$

prescale = 300, count = 59999

ou

prescale = 600, count = 29999

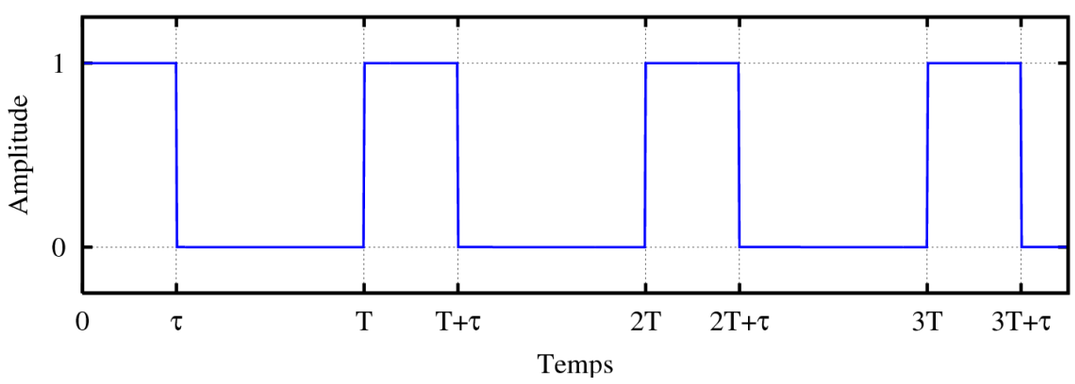
ou

prescale = 1200, count = 14999

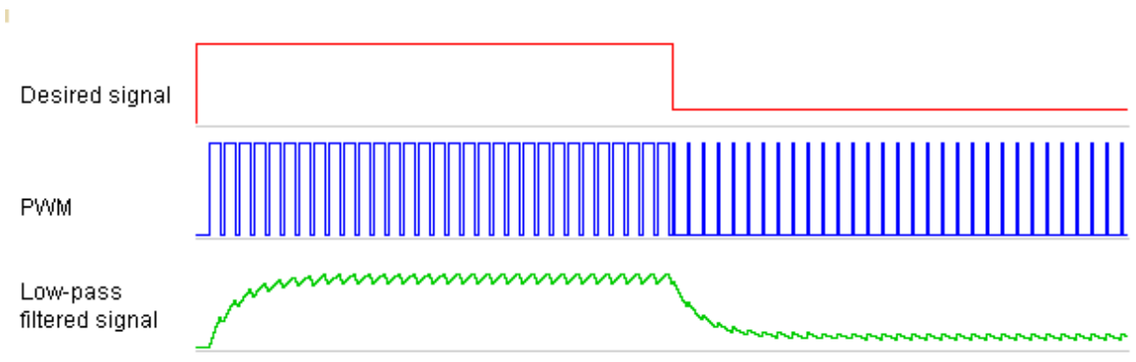
...

PWM

- Pulse-Width Modulation (Modulation de Largeur d'Impulsion)
- Rapport cyclique : ON / (ON + OFF)
- DAC en ajoutant un filtre passe-bas
  - un haut parleur a une impédance suffisante pour se passer de filtre passe-bas



http://fr.wikipedia.org/wiki/Rapport_cyclique



Fast PWM

OCRnA/OCRnB

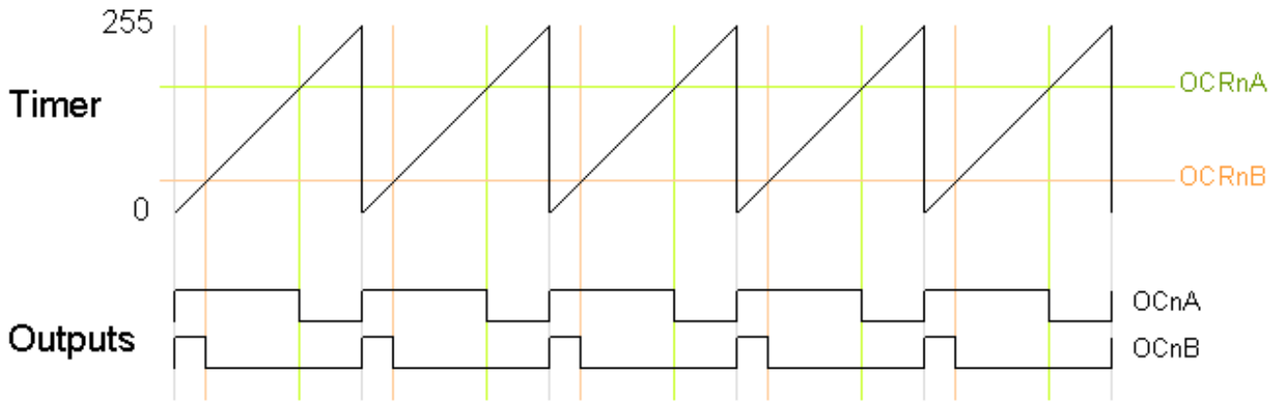
registres de comparaison

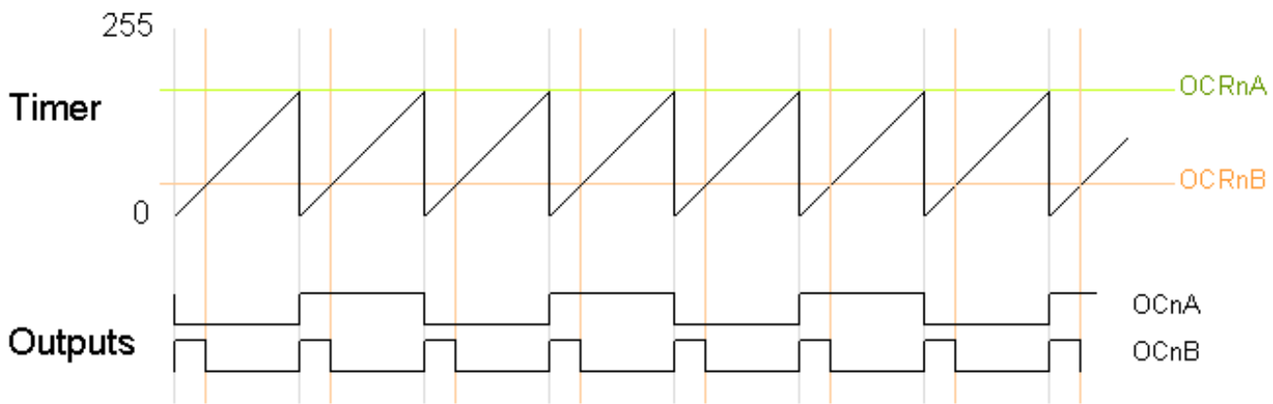
OCnA/OCnB

sorties

Rapide

La phase varie



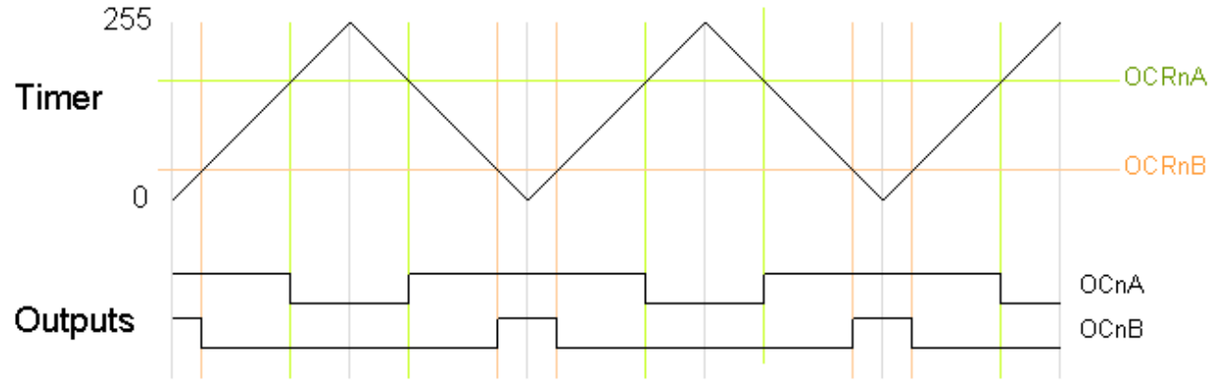


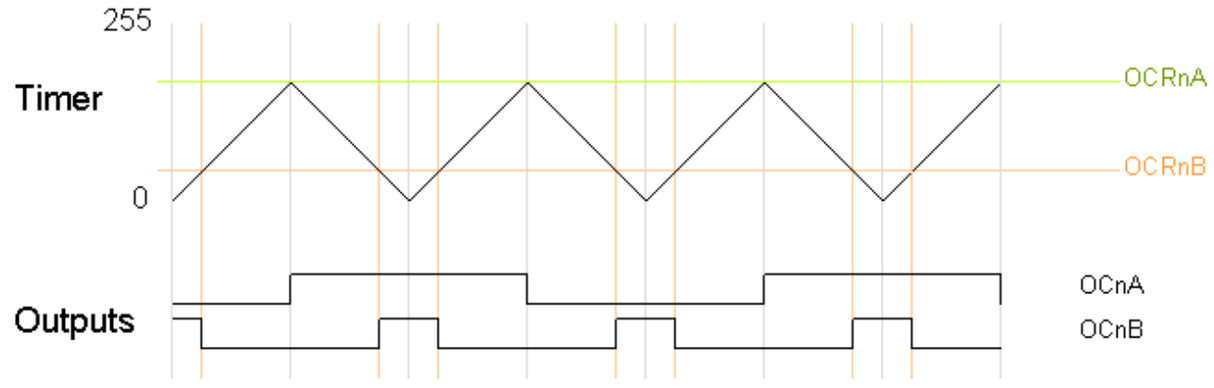
Toggle OCnA on Compare Match

Phase correct PWM

Plus lent

La période est fixe





Utilisation - Arduino

PWM (simple)

```
analogWrite(pin, value)
```

- valeur : entre 0 et 255

Librairie HardwareTimer

https://github.com/stm32duino/Arduino_Core_STM32/wiki/HardwareTimer-library

```
HardwareTimer(timer)
```

- timer : TIM1, TIM4, TIM7, TIM8, TIM12, TIM13, ou TIM14

PWM (détaillé)

```
setPWM(channel, pin, frequency, dutyCycle, periodCallback, compareCallback)
```

- dutyCycle : entre 0 et 100
- periodCallback et compareCallback : optionnels (interrupts)

Beaucoup d'autres fonctionnalités

- Prescale, count, pause/resume, …
- Cf doc

Exemple PWM - simple

```
const int pwmPin = PB7;

void setup() {
  pinMode(pwmPin, OUTPUT);
}

void loop() {
  analogWrite(pwmPin, 64);
  delay(1000);
  analogWrite(pwmPin, 255);
  delay(1000);
}
```

Exemple PWM - détaillé

```
#include <HardwareTimer.h>

const int pwmPin = PB7; //Timer 4 Channel 2

HardwareTimer *MyTim = NULL;

void setup() {
  MyTim = new HardwareTimer(TIM4);
  MyTim->setPWM(2, pwmPin, 10, 50);
}

void loop() {
}
```

Exemple interrupt

```
#include <HardwareTimer.h>

const int pwmPin = PB7; //Timer 4 Channel 2
bool ledOn = false;

HardwareTimer *MyTim = NULL;
```

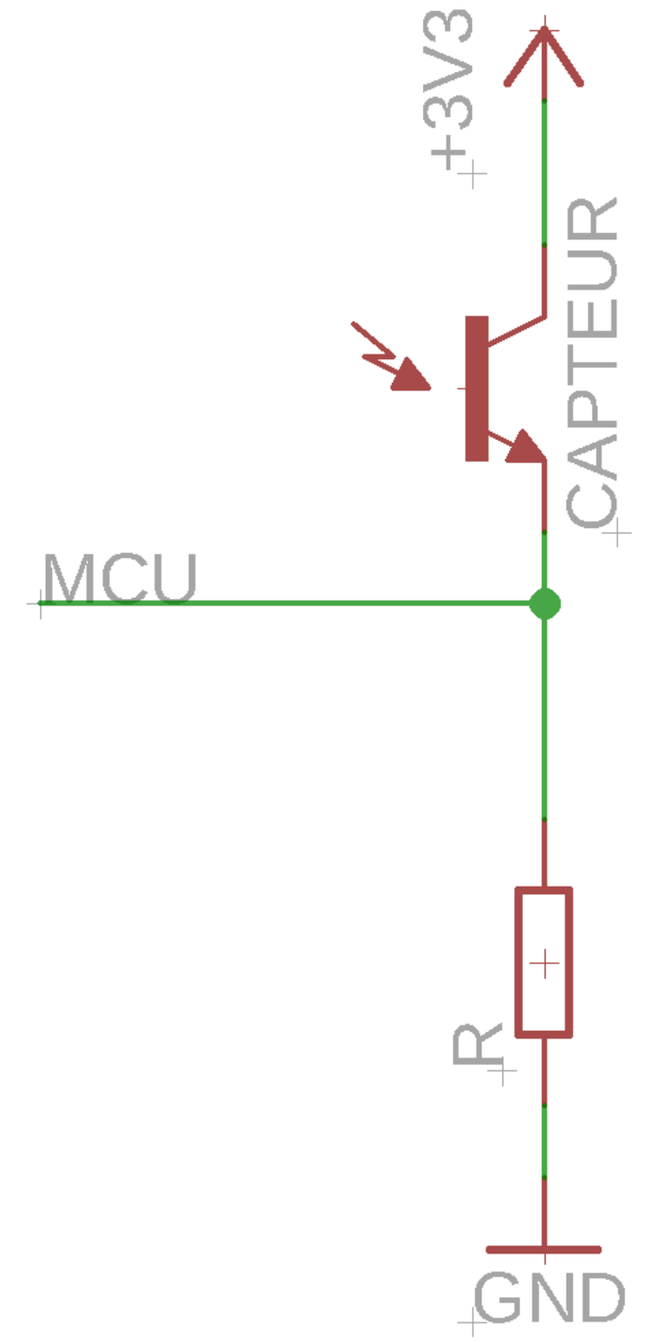
```
void ledCallback() {
  ledOn ^= true; // toggle
}

void setup() {
  MyTim = new HardwareTimer(TIM4);
  MyTim->setPWM(1, PB6, 10, 50, ledCallback);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  digitalWrite(ledPin, buttonPressed?HIGH:LOW);
}
```

Exemple interrupt 2

```
#include <HardwareTimer.h>

const int pwmPin = PB7; //Timer 4 Channel 2
bool ledOn = false;

HardwareTimer *MyTim = NULL;

void on() {
  ledOn = true;
}

void off() {
  ledOn = false;
}

void setup() {
  MyTim = new HardwareTimer(TIM4);
  MyTim->setPWM(1, PB6, 2, 80, on, off);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  digitalWrite(ledPin, buttonPressed?HIGH:LOW);
}
```

ADC

- Convertisseur analogique → numérique
- Capteurs simples
  - Micro
  - Potentiomètre
  - Joystick
  - Photorésistance
  - …



Utilisation - Arduino

Lecture

```
analogRead(pin)
```

- retourne une valeur uint32_t

Précision

```
analogResolution(bits)
```

Exemple ADC

```
const int inputPin = PA3;
const int ledPin = PB7;

void setup() {
  pinMode(inputPin, INPUT);
  pinMode(ledPin, OUTPUT);
  analogReadResolution(8);
}

void loop() {
  uint32_t value = analogRead(inputPin);
  analogWrite(ledPin, value);
  delay(100);
}
```

Bus

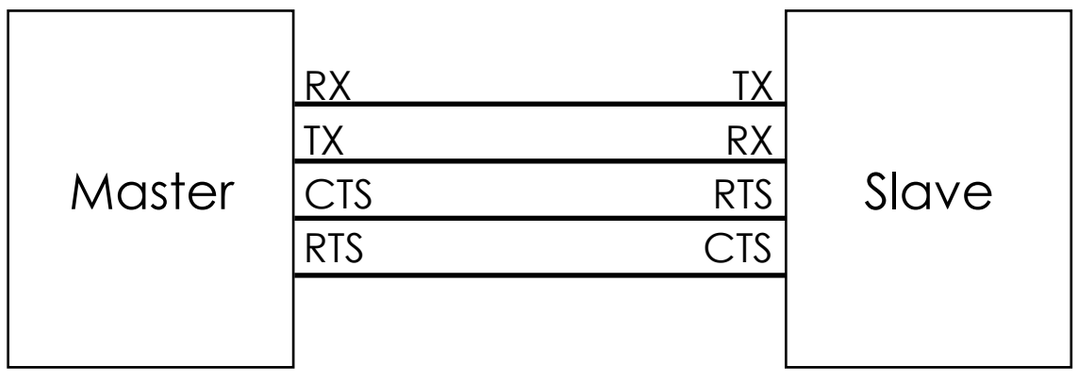
- Communication entre MCU
- Communication avec un PC
- Sérialisation / parallélisation

UART/USART

Universal (Synchronous/)Asynchronous Receiver Transceiver

Protocole RS 232 (port série PC)

- RX : réception
- TX : transmission
- RTS : prêt à écouter
- CTS : prêt à envoyer



RTS/CTS optionnels

Le maître et l'esclave doivent être configurées de la même façon

Utilisation - Arduino

Initialisation

```
Serial.begin(speed)
```

- Speed : typiquement 9600

Test

```
Serial.available()
```

- Retourne le nombre d'octets disponibles

Lecture

```
Serial.read()
```

- Retourne le prochain octet, -1 si erreur

```
Serial.readBytes(buffer, length)
```

- buffer : tableau où insérer les octets lus
- length : nombre max d'octets à lire
- Retourne le nombre d'octets écrits

Écriture

```
Serial.write(byte)
```

- Retourne -1 si erreur

```
Serial.print(val, format) / Serial.println(val, format)
```

- val : plusieurs types possibles
- Format : DEC, HEX, OCT, BIN, pour les nombres seulement
- La version ln ajoute \r\n

Exemple UART - écriture

```
int i = 0;

void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.println(i++);
  delay(100);
}
```

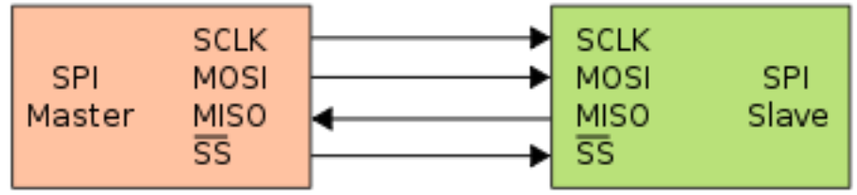
Exemple UART - lecture/écriture

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  if (Serial.available()) {
    int v = Serial.read();
    Serial.write(v);
  }
  delay(100);
}
```

SPI

Serial Peripheral Interface

- SS/CS : slave select
- SCK : horloge
- MISO/SDO : master in slave out
- MOSI/SDI : master out slave in



Rapide

Bi-directionnel

Petites distances

Nouvelles conventions de nommage

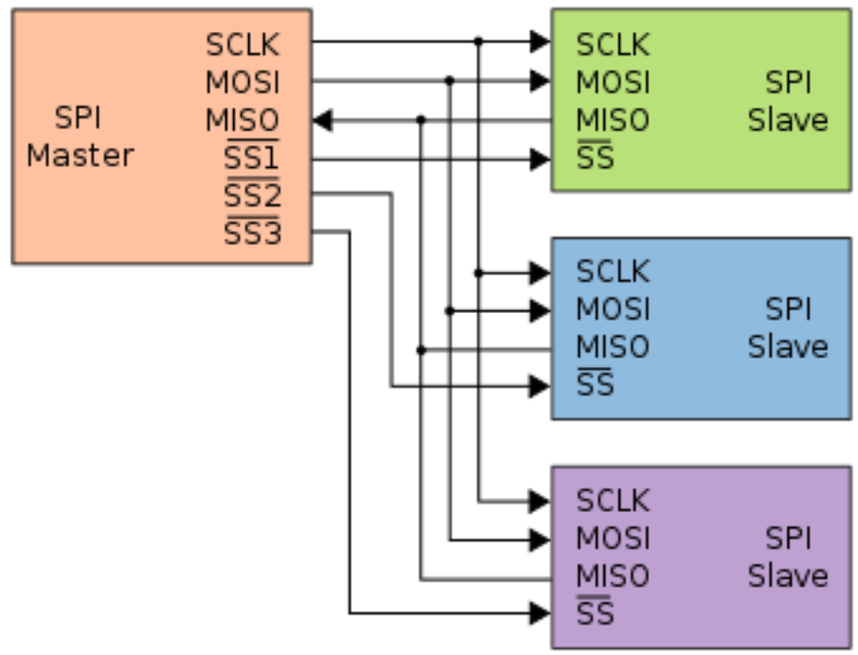
Master → Controller

Slave → Peripheral

MISO → CIPO : Controller In, Peripheral Out

MOSI → COPI : Controller Out Peripheral In

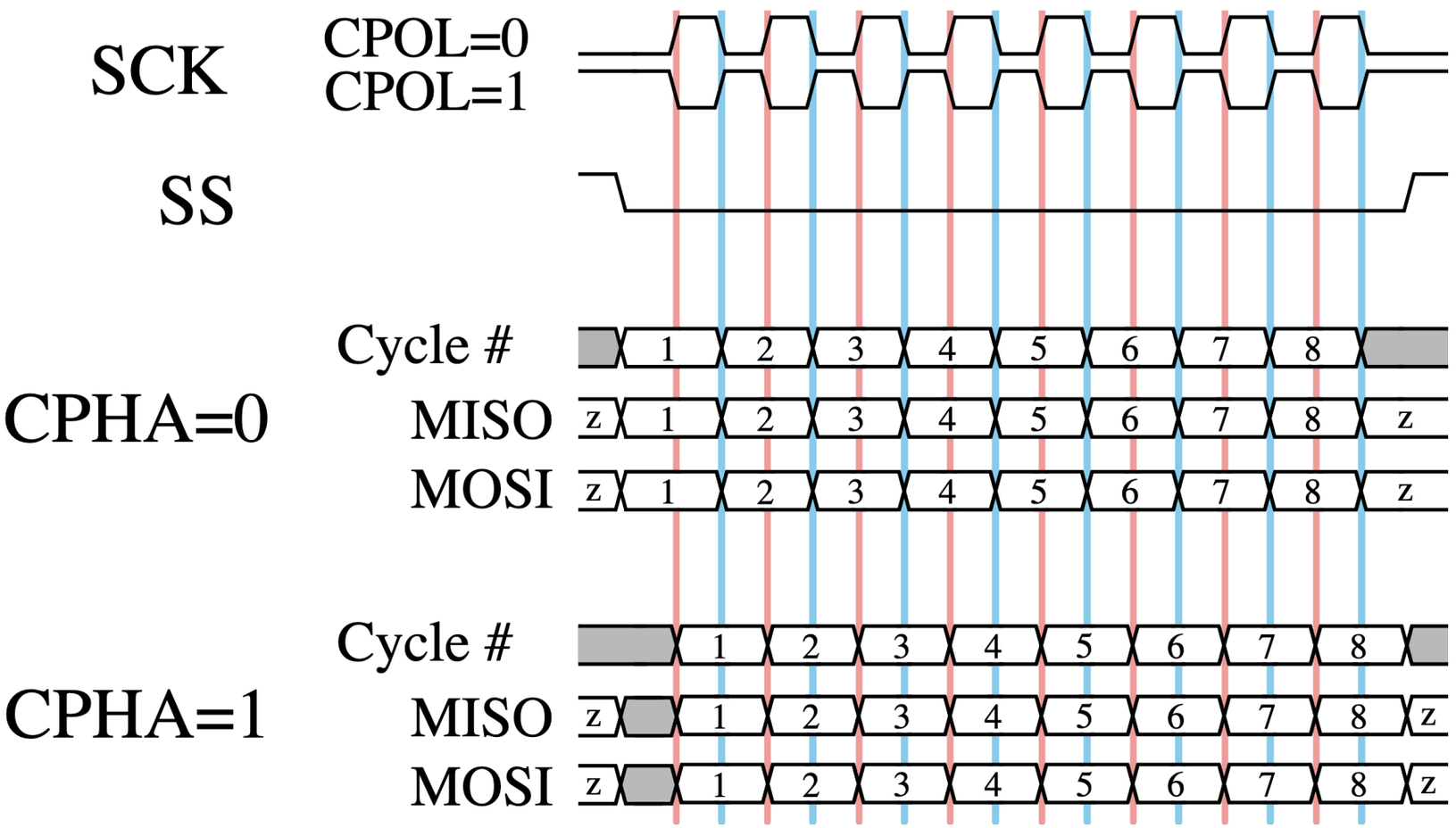
SS → CS : Chip Select



https://fr.wikipedia.org/wiki/Serial_Peripheral_Interface

SPI : phases

Mode	CPOL	CPHA
0 (0,0)	0	0
1 (0,1)	0	1
2 (1,0)	1	0
3 (1,1)	1	1



Utilisation - Arduino

Initiatlisation

```
SPI.begin()
```

Configuration

```
SPI.setMOSI(pin), SPI.setMISO(pin) SPI.setSCLK(pin), SPI.setSSEL(pin)
```

```
SPI.setClockDivider(divider)
```

- divider : SPI_CLOCK_DIVx (x = 2, 4, 8, 16, 32, 64, ou 128)

```
SPI.setDataMode(dataMode)
```

- dataMode : SPI_MODE0, SPI_MODE1, SPI_MODE2, ou SPI_MODE3

```
SPI.setBitOrder(dataOrder)
```

- dataOrder : MSBFIRST oo LSBFIRST

```
SPI.beginTransaction(SPISettings(speedMaximum, dataOrder, dataMode))
```

Lecture + écriture

```
SPI.transfer(val)
```

- val : valeur à envoyer
- Retourne la valeur lue

Exemple SPI - controller / CS hardware

```
#include <SPI.h>

const int ledPin = PB7;
int i = 0;

void setup() {
  pinMode(ledPin, OUTPUT);

  SPI.setMOSI(PA7);
  SPI.setMISO(PA6);
  SPI.setSCLK(PA5);
  SPI.setSSEL(PC9);

  SPI.begin();
  SPI.setClockDivider(SPI_CLOCK_DIV16);
  SPI.setDataMode(SPI_MODE0);
  SPI.setBitOrder(MSBFIRST);
}

void loop() {
  SPI.transfer(i++);
  delay(100);
}
```

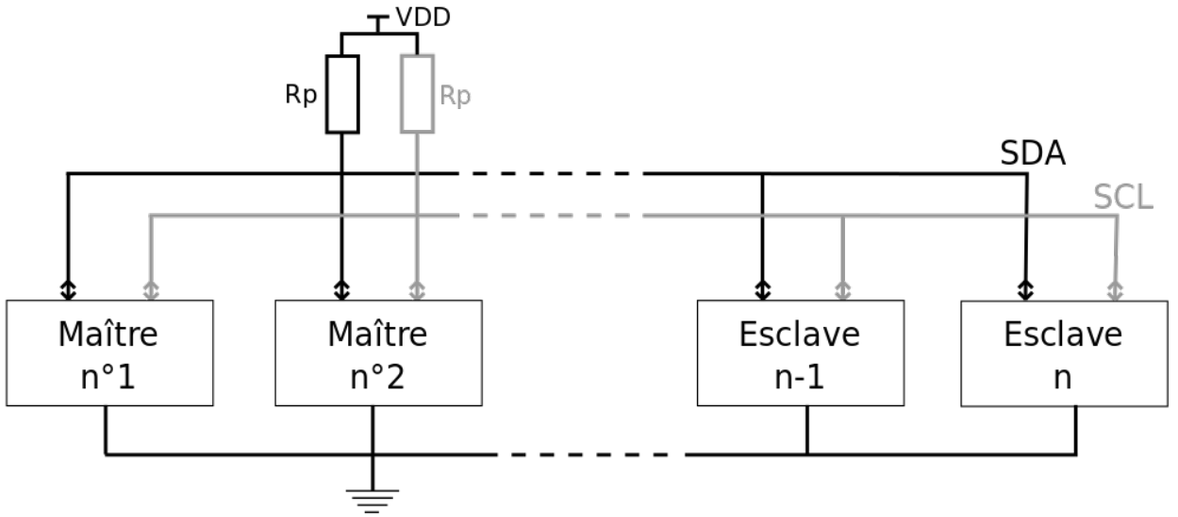
Exemple SPI - controller / CS software

```
#include <SPI.h>

const int ledPin = PB7;
const int CSPin = PC9;
int i = 0;

void setup() {
  pinMode(ledPin, OUTPUT);

  SPI.setMOSI(PA7);
  SPI.setMISO(PA6);
  SPI.setSCLK(PA5);
  pinMode(CSPin, OUTPUT);
  digitalWrite(CSPin, HIGH);

  SPI.begin();
  SPI.setClockDivider(SPI_CLOCK_DIV16);
  SPI.setDataMode(SPI_MODE0);
  SPI.setBitOrder(MSBFIRST);
}

void loop() {
  digitalWrite(CSPin, LOW);
  SPI.transfer(i++);
  digitalWrite(CSPin, HIGH);
  delay(100);
}
```

I2C

Inter Integrated Circuit

- SDA : données
- SCL : horloge



http://fr.wikipedia.org/wiki/I2C

Masse commune

Plus lent que SPI

Peu de fils

Adresses

Jusqu'à 256 périphérique par bus

Utilisation - Arduino

Initiatlisation

```
Wire.begin(address)
```

- Address : périphérique, inutile pour un contrôleur

Configuration

```
Wire.setClock(frequency)
```

- frequency : 100000 (standard), 400000 (fast)

Test

```
Wire.available()
```

- Retourne le nombre d'octets disponibles

Communication contrôleur

```
Wire.beginTransmission(address)
```

- Address : destination

```
Wire.endTransmission()
```

```
Wire.requestFrom(address, nb)
```

- Address : destination

Communication périphérique

```
Wire.onReceive(callback), Wire.onRequest(callback)
```

Lecture

```
Wire.read()
```

- Retourne le prochain octet, -1 si erreur

Écriture

```
Wire.write(byte)
```

- Retourne -1 si erreur

Exemple I2C - controller

```
#include <Wire.h>

const int ledPin = PB7;

void setup() {
  pinMode(ledPin, OUTPUT);

  Wire.begin();
  Wire.setClock(400000);
}
```

```
void loop() {
  Wire.beginTransmission(0×42);
  Wire.write(0×01);
  Wire.endTransmission();

  Wire.requestFrom(0×42, 1);

  if (Wire.available()) {
    char test = Wire.read();
    digitalWrite(ledPin, test==0×01?HIGH:LOW);
  }
  delay(100);
}
```

Exemple I2C - peripheral

```
#include <Wire.h>

const int ledPin = PB7;

void setup() {
  pinMode(ledPin, OUTPUT);

  Wire.begin(0×42);
  Wire.onReceive(receiveEvent);
  Wire.onRequest(requestEvent);
}

void receiveEvent() {
  if (Wire.available()) {
    char test = Wire.read();
    digitalWrite(ledPin, test==0×01?HIGH:LOW);
  }
}

void requestEvent() {
  Wire.write(0×01);
}

void loop() {
delay(100);
}
```

USB

Universal Serial Bus

- VCC
- GND
- D+
- D-

Classes

Mass storage

Media Transfer Protocol

Human Interface Devices (HID) : claviers, souris, joysticks, etc.

Virtual com port

…

ICM-20649

6-DoF IMU Accelerometer and Gyro

- https://github.com/adafruit/Adafruit_Sensor
- https://github.com/adafruit/Adafruit_ICM20X/

Handle

```
Adafruit_ICM20649 icm;
```

Initialisation

```
icm.begin_I2C()
```

Sensibilité

```
icm.getAccelRange(), icm.setAccelRange(range)
```

- range : ICM20649_ACCEL_RANGE_x_G (x = 4, 8, 16, 30)

```
icm.getGyroRange()
```

- range : ICM20649_GYRO_RANGE_x_DPS (x = 500, 1000, 2000, 4000)

```
icm.getAccelRateDivisor(), icm.setAccelRateDivisor(rate)

icm.getGyroRateDivisor(), icm.setGyroRateDivisor(rate)
```

Lecture

```
icm.getEvent(accel, gyro, temp)
```

- Accel, gyro, temp : sensors_event_t

```
accel.acceleration.x

accel.acceleration.y

accel.acceleration.z

gyro.gyro.x

gyro.gyro.y

gyro.gyro.z

temp.temperature
```