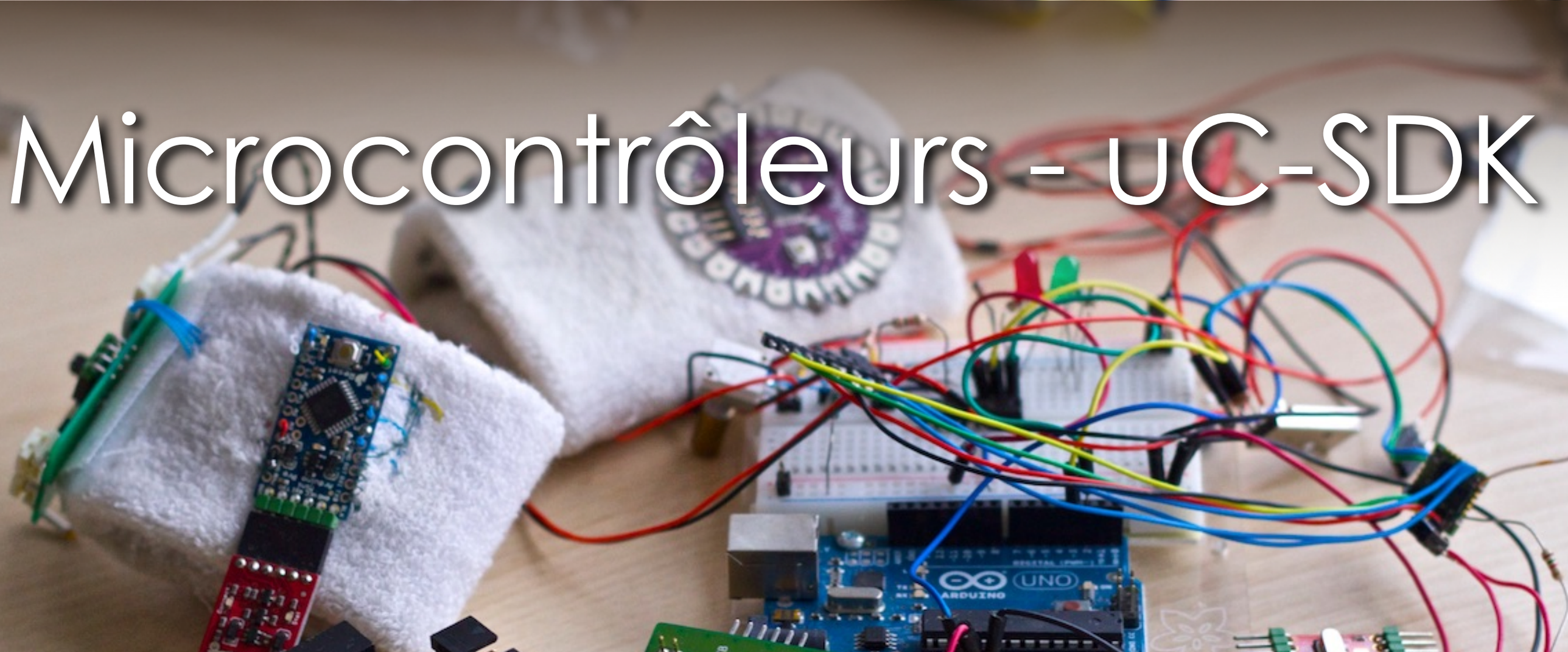
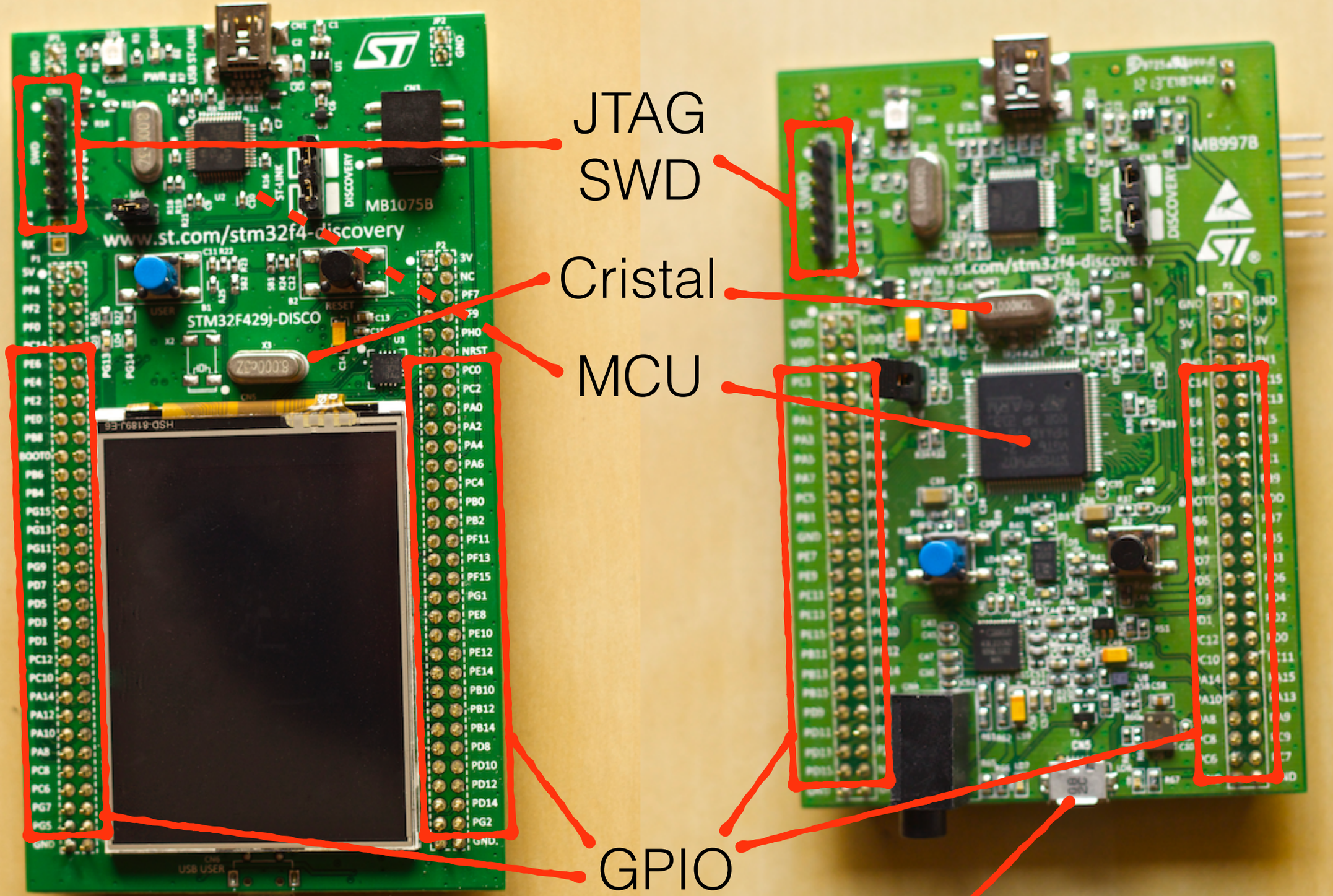
Microcontrôleurs - uC-SDK

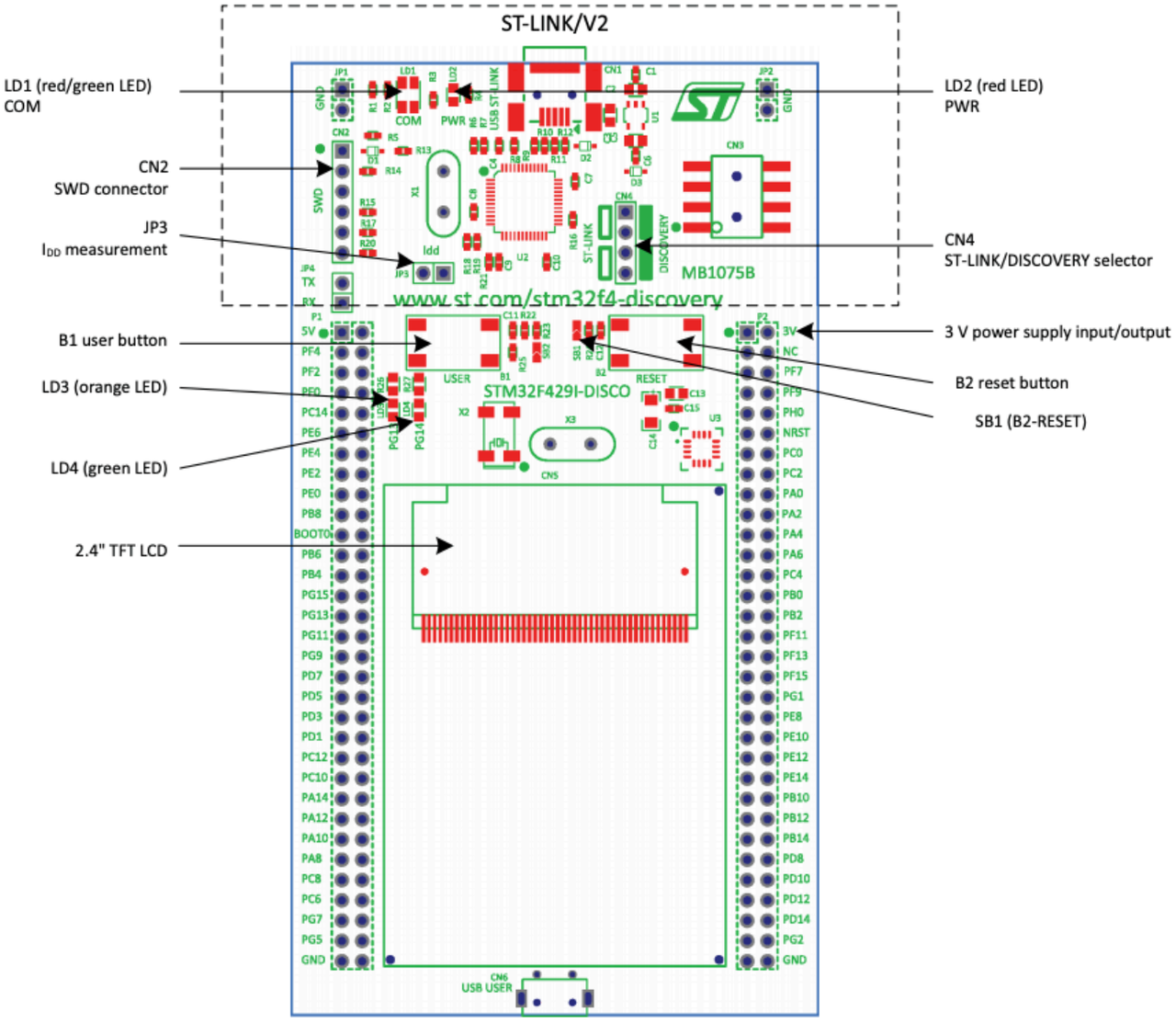


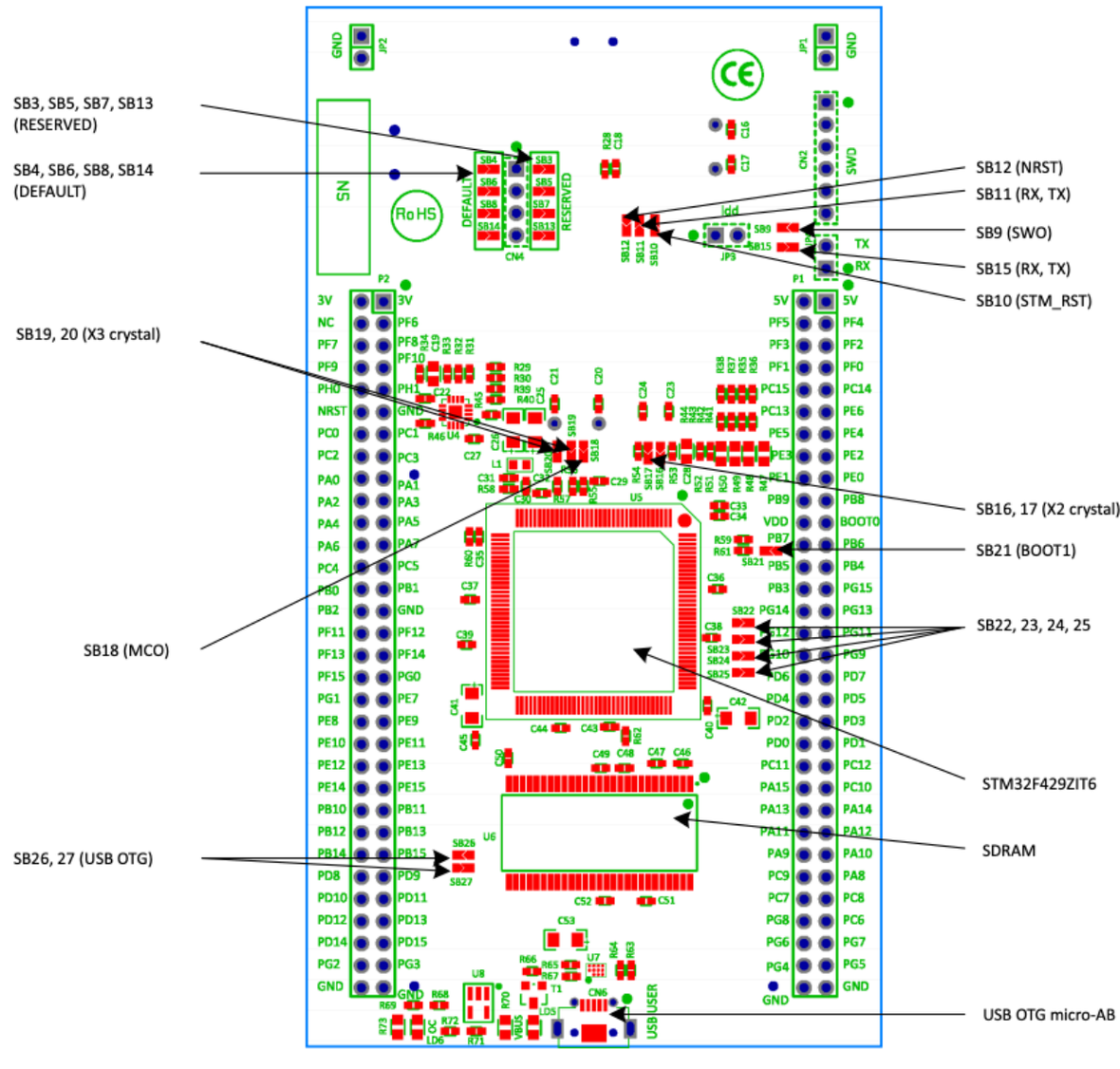
Thomas Pietrzak
Master 2 Informatique — RVA

Université de Lille

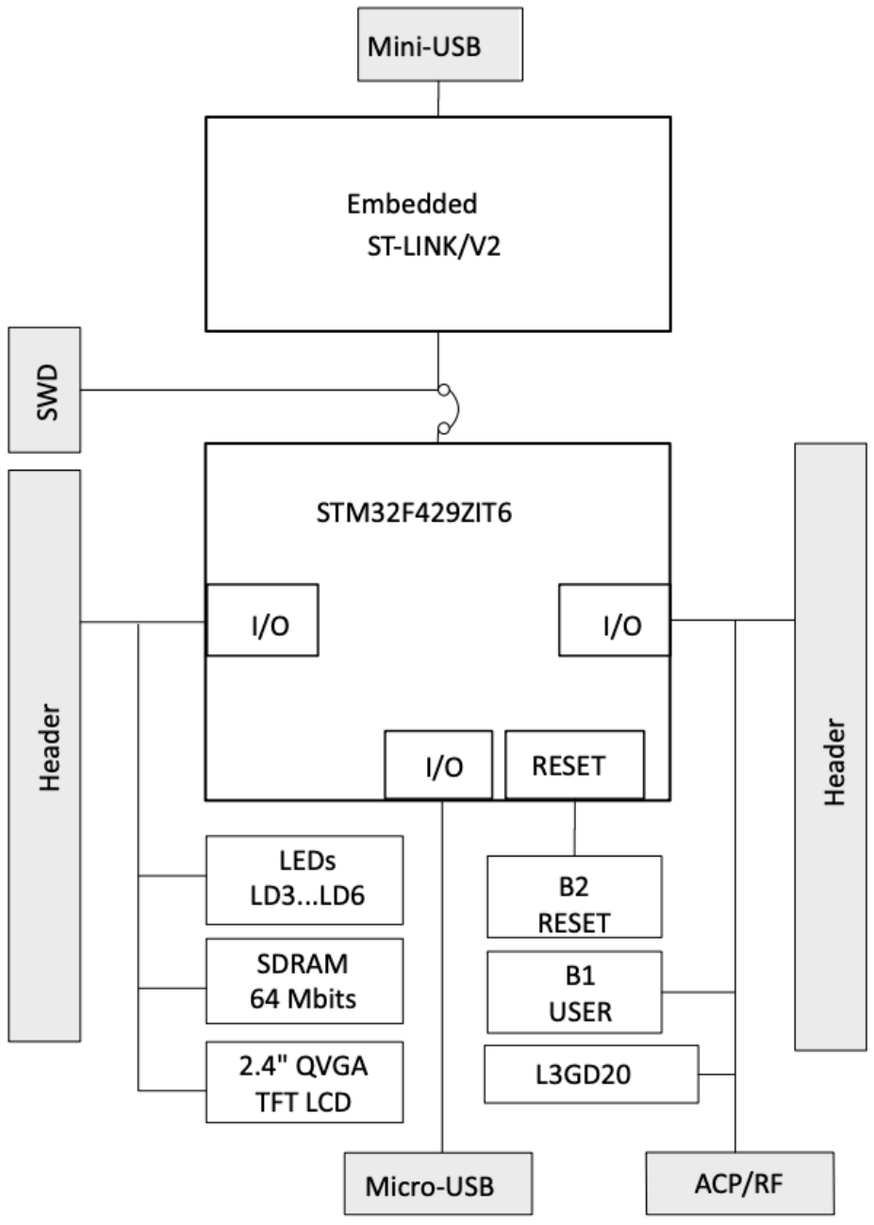

JTAG
SWD
Cristal
MCU
GPIO
Prise USB
STM32F429J-DISCO
MB1075B
MB9978
www.st.com/stm32f4-discovery

Carte STM32f429 discovery





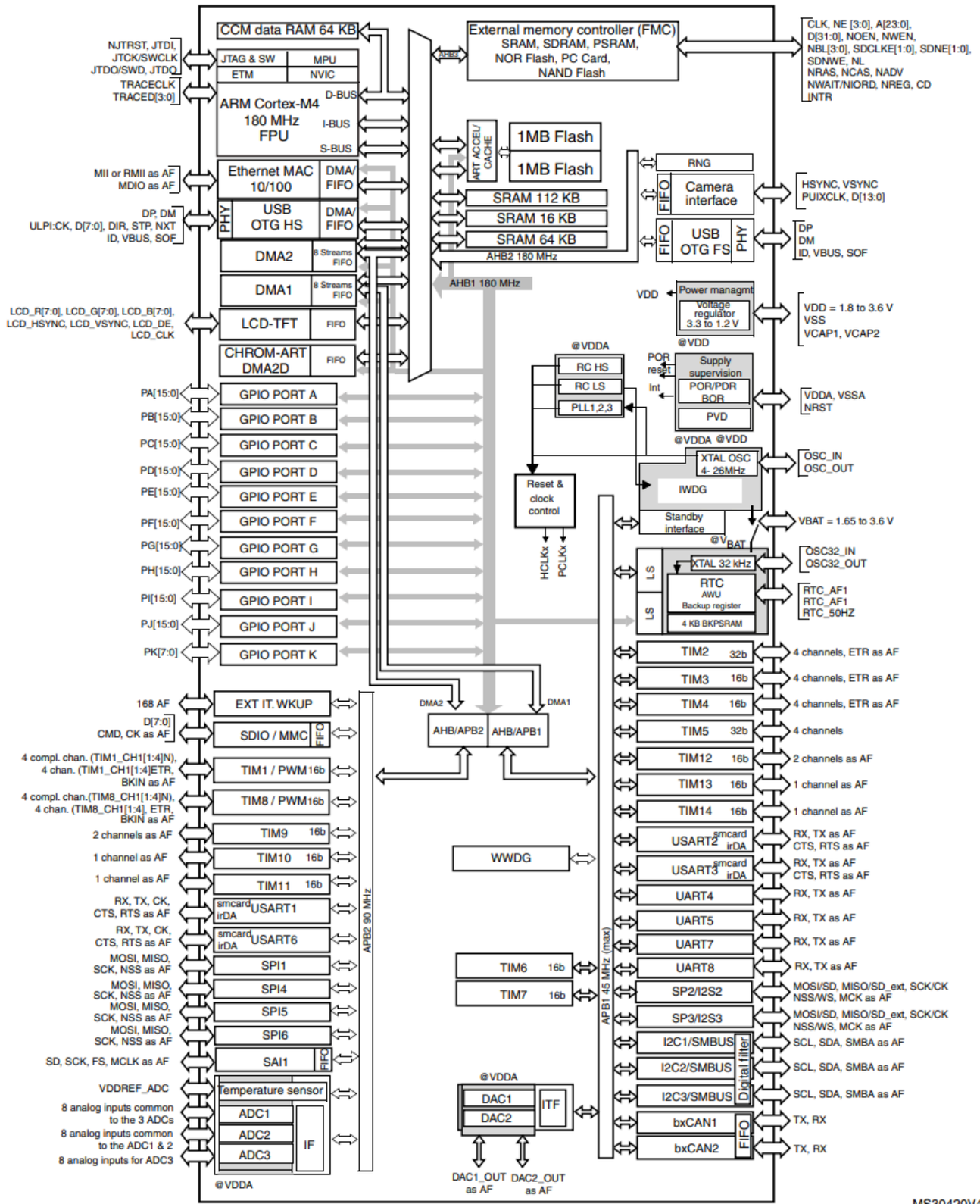
Carte STM32f429 discovery



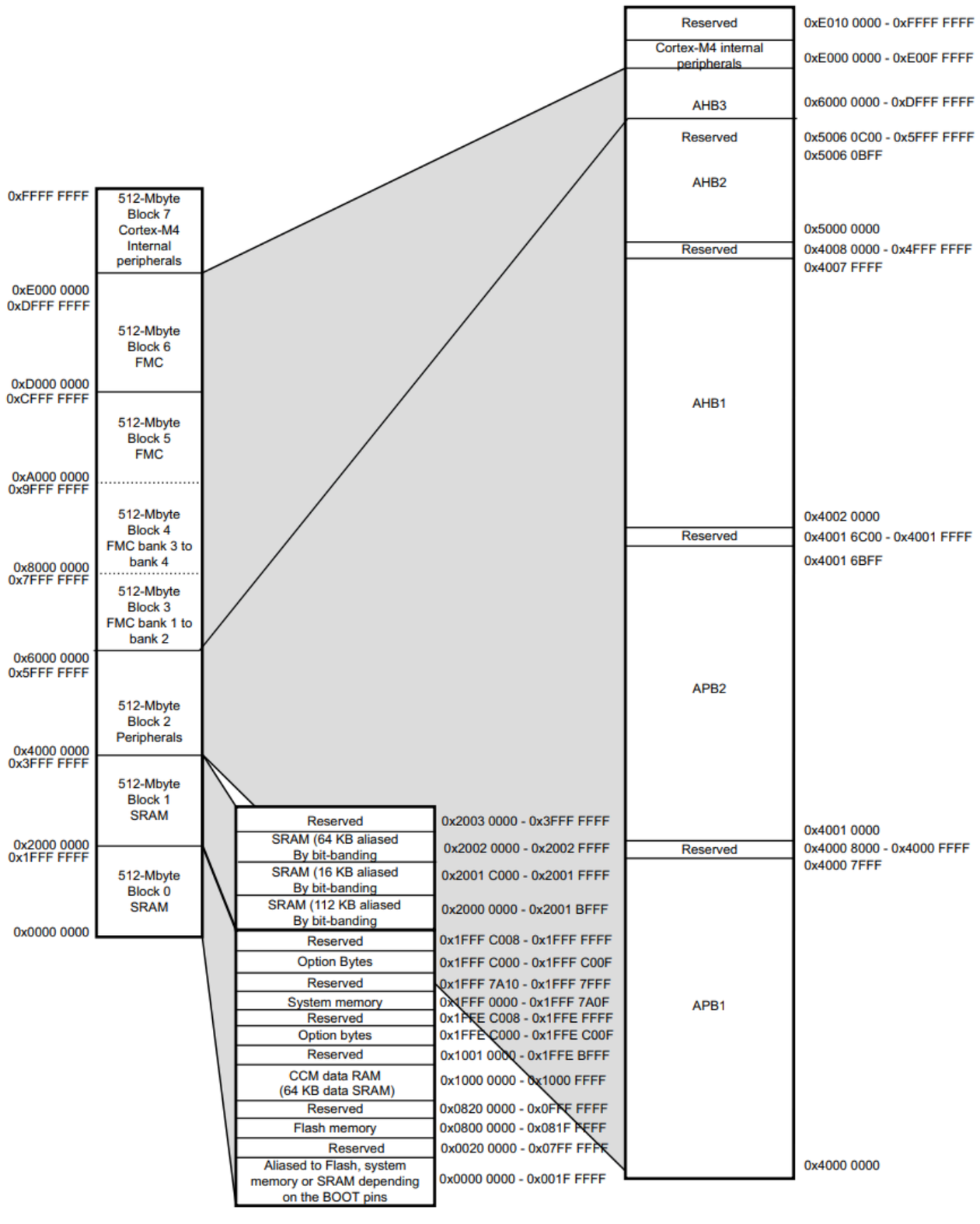
Pins

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

Clocks


CCM data RAM 64 KB
NJTRST, JTDI, JTCK/SWCLK JTDO/SWD, JTDO
TRACECLK TRACED[3:0]
JTAG & SW
ETM
MPU
NVIC
ARM Cortex-M4 180 MHz FPU
D-BUS
I-BUS
S-BUS
External memory controller (FMC) SRAM, SDRAM, PSRAM, NOR Flash, PC Card, NAND Flash
AHBS
CLK, NE [3:0], A[23:0], D[31:0], NOEN, NWEN, NBL[3:0], SDCLKE[1:0], SDNE[1:0], SDNWE, NL NRAS, NCAS, NADV NWAIT/NIORD, NREG, CD INTR
ART ACCEL/ CACHE
1MB Flash
1MB Flash
SRAM 112 KB
SRAM 16 KB
SRAM 64 KB
AHB2 180 MHz
RNG
FIFO
Camera interface
HSYNC, VSYNC PUIXCLK, D[13:0]
FIFO
USB OTG FS
PHY
DP DM ID, VBUS, SOF
MII or RMII as AF MDIO as AF
Ethernet MAC 10/100
DMA/ FIFO
DP, DM ULPI:CK, D[7:0], DIR, STP, NXT ID, VBUS, SOF
PHY
USB OTG HS
DMA/ FIFO
DMA2
8 Streams FIFO
DMA1
8 Streams FIFO
AHB1 180 MHz
LCD_R[7:0], LCD_G[7:0], LCD_B[7:0], LCD_HSYNC, LCD_VSYNC, LCD_DE, LCD_CLK
LCD-TFT
FIFO
CHROM-ART DMA2D
FIFO
VDD
Power managmt
Voltage regulator 3.3 to 1.2 V
@VDD
VDD = 1.8 to 3.6 V VSS VCAP1, VCAP2
@VDDA
RC HS
RC LS
PLL1,2,3
POR reset
Int
Supply supervision
POR/PDR BOR
PVD
@VDDA @VDD
VDDA, VSSA NRST
PA[15:0]
PB[15:0]
PC[15:0]
PD[15:0]
PE[15:0]
PF[15:0]
PG[15:0]
PH[15:0]
PI[15:0]
PJ[15:0]
PK[7:0]
GPIO PORT A
GPIO PORT B
GPIO PORT C
GPIO PORT D
GPIO PORT E
GPIO PORT F
GPIO PORT G
GPIO PORT H
GPIO PORT I
GPIO PORT J
GPIO PORT K
Reset & clock control
HCLKx
PCLKx
XTAL OSC 4- 26MHz
IWDG
Standby interface
@VBAT
OSC_IN OSC_OUT
VBAT = 1.65 to 3.6 V
LS
LS
XTAL 32 kHz
RTC AWU Backup register
4 KB BKPSRAM
OSC32_IN OSC32_OUT
RTC_AF1 RTC_AF1 RTC_50HZ
168 AF
EXT IT. WKUP
D[7:0] CMD, CK as AF
SDIO / MMC
FIFO
4 compl. chan. (TIM1_CH1[1:4]N), 4 chan. (TIM1_CH1[1:4]ETR, BKIN as AF
TIM1 / PWM 16b
4 compl. chan.(TIM8_CH1[1:4]N), 4 chan. (TIM8_CH1[1:4], ETR, BKIN as AF
TIM8 / PWM 16b
2 channels as AF
TIM9 16b
1 channel as AF
TIM10 16b
1 channel as AF
TIM11 16b
RX, TX, CK, CTS, RTS as AF
smcard irDA
USART1
RX, TX, CK, CTS, RTS as AF
smcard irDA
USART6
MOSI, MISO, SCK, NSS as AF
SPI1
MOSI, MISO, SCK, NSS as AF
SPI4
MOSI, MISO, SCK, NSS as AF
SPI5
MOSI, MISO, SCK, NSS as AF
SPI6
SD, SCK, FS, MCLK as AF
SAI1
FIFO
APB2 90 MHz
DMA2
DMA1
AHB/APB2
AHB/APB1
WWDG
TIM6 16b
TIM7 16b
APB1 45 MHz (max)
TIM2 32b
4 channels, ETR as AF
TIM3 16b
4 channels, ETR as AF
TIM4 16b
4 channels, ETR as AF
TIM5 32b
4 channels
TIM12 16b
2 channels as AF
TIM13 16b
1 channel as AF
TIM14 16b
1 channel as AF
USART2 smcard irDA
RX, TX as AF CTS, RTS as AF
USART3 smcard irDA
RX, TX as AF CTS, RTS as AF
UART4
RX, TX as AF
UART5
RX, TX as AF
UART7
RX, TX as AF
UART8
RX, TX as AF
SP2/I2S2
MOSI/SD, MISO/SD_ext, SCK/CK NSS/WS, MCK as AF
SP3/I2S3
MOSI/SD, MISO/SD_ext, SCK/CK NSS/WS, MCK as AF
I2C1/SMBUS
I2C2/SMBUS
I2C3/SMBUS
Digital f ilter
SCL, SDA, SMBA as AF
SCL, SDA, SMBA as AF
SCL, SDA, SMBA as AF
bxCAN1
bxCAN2
FIFO
TX, RX
TX, RX
VDDREF_ADC
8 analog inputs common to the 3 ADCs
8 analog inputs common to the ADC1 & 2
8 analog inputs for ADC3
Temperature sensor
ADC1
ADC2
ADC3
IF
@VDDA
@VDDA
DAC1
DAC2
ITF
DAC1_OUT as AF
DAC2_OUT as AF
MS30439V4

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
0xA000 0000
0x9FFF FFFF
512-Mbyte Block 4 FMC bank 3 to bank 4
0x8000 0000
0x7FFF FFFF
512-Mbyte Block 3 FMC bank 1 to bank 2
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
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
Reserved
0x2003 0000 - 0x3FFF FFFF
SRAM (64 KB aliased By bit-banding
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
CCM data RAM (64 KB data SRAM)
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

Pins

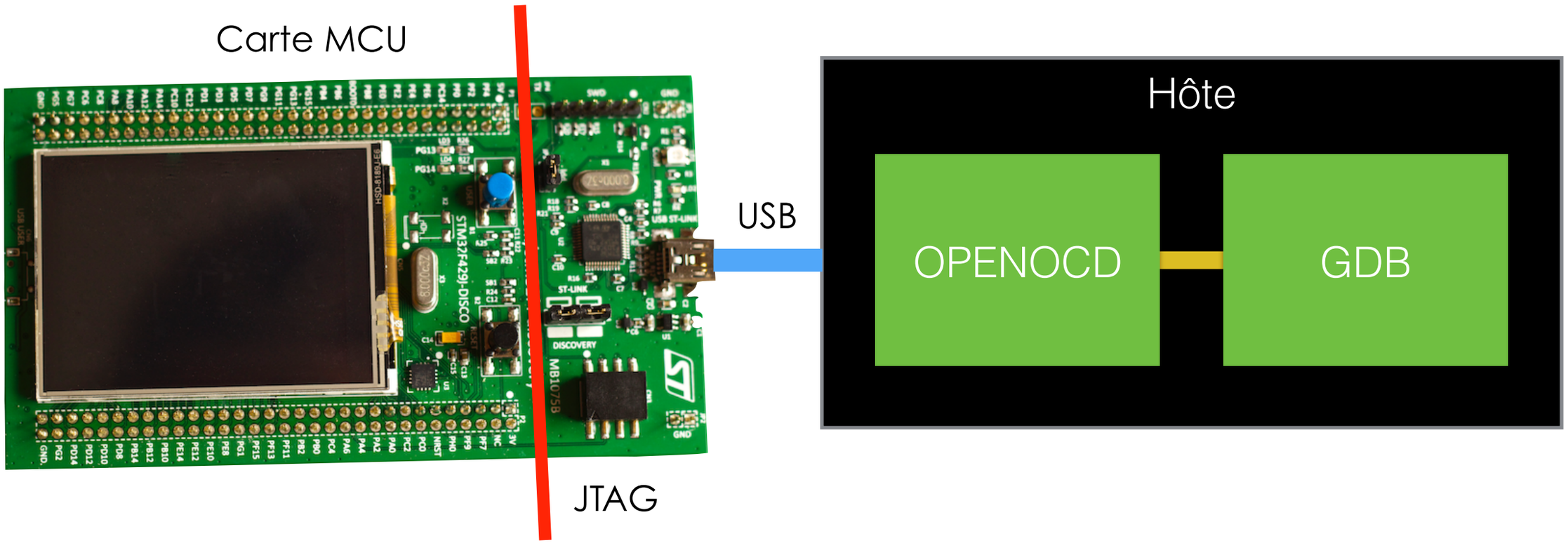
Table 10. STM32F427xx and STM32F429xx pin and ball definitions (continued)

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

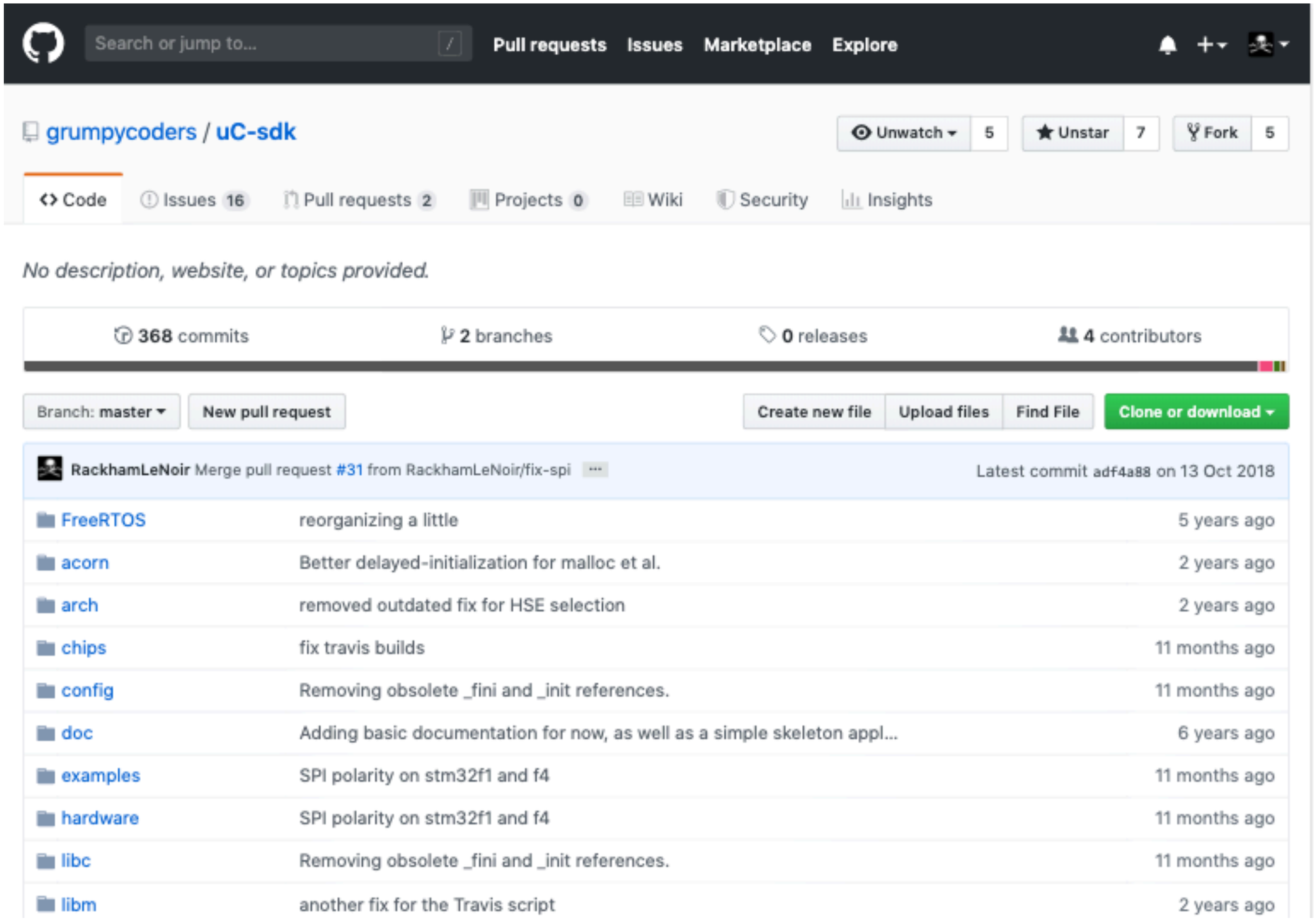
JTAG/SWD

- Programmation
- Débuggage pas à pas

Débuggage



uC-SDK

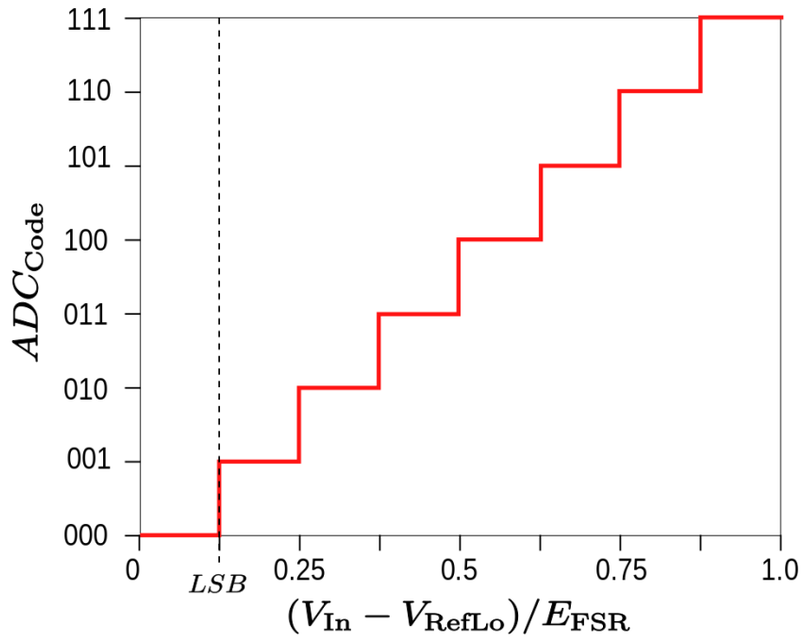

Search or jump to...
Pull requests
Issues
Marketplace
Explore
grumpycoders / uC-sdk
Unwatch 5
Unstar 7
Fork 5
Code
Issues 16
Pull requests 2
Projects 0
Wiki
Security
Insights
No description, website, or topics provided.
368 commits
2 branches
0 releases
4 contributors
Branch: master
New pull request
Create new file
Upload files
Find File
Clone or download
RackhamLeNoir Merge pull request #31 from RackhamLeNoir/fix-spi
Latest commit adf4a88 on 13 Oct 2018
FreeRTOS reorganizing a little 5 years ago
acorn Better delayed-initialization for malloc et al. 2 years ago
arch removed outdated fix for HSE selection 2 years ago
chips fix travis builds 11 months ago
config Removing obsolete _fini and _init references. 11 months ago
doc Adding basic documentation for now, as well as a simple skeleton appl... 6 years ago
examples SPI polarity on stm32f1 and f4 11 months ago
hardware SPI polarity on stm32f1 and f4 11 months ago
libc Removing obsolete _fini and _init references. 11 months ago
libm another fix for the Travis script 2 years ago

uC-SDK

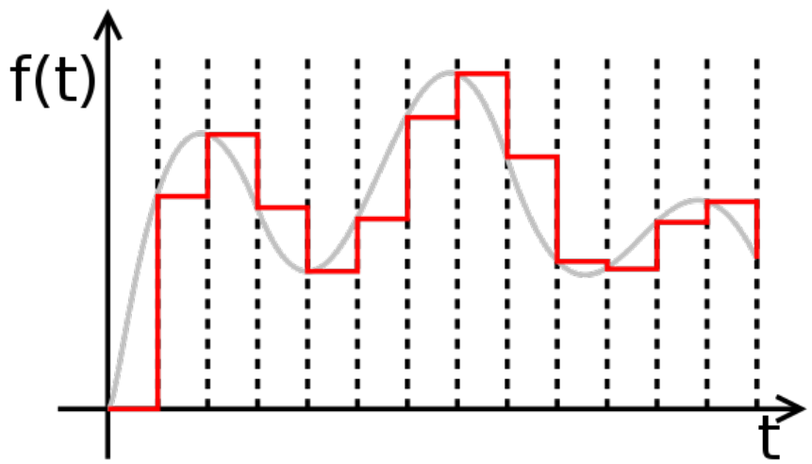
- FreeRTOS
  - threads
  - mutex
  - timers software
- Interface hardware
  - ADC, DAC, GPIO, I2C, SPI, timers hardware, UART
- Implémentations
  - LPC17xx, STM32f10x, stm32f4xx
- libm

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 2800 - 0x4002 2BFF	GPIOK
0x4002 2400 - 0x4002 27FF	GPIOJ
0x4002 2000 - 0x4002 23FF	GPIOI
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

Configuration

```c
typedef struct {
    uint8_t port;
    uint8_t pin;
} pin_t;

typedef enum {
    pin_dir_read = 0,
    pin_dir_write = 1,
} pin_dir_t;

typedef enum {
    pull_none = 0,
    pull_up = 1,
    pull_down = 2,
} pull_t;


typedef enum {
    gpio_port_a,
    gpio_port_b,
    gpio_port_c,
    gpio_port_d,
    gpio_port_e,
    gpio_port_f,
    gpio_port_g,
    gpio_port_h,
    gpio_port_i,
    gpio_port_j,
} gpio_port_t;
```

```c
void gpio_config(pin_t pin, pin_dir_t dir, pull_t pull) {

    RCC_AHB1PeriphClockCmd(1 << pin.port, ENABLE);

    GPIO_InitTypeDef def;
    def.GPIO_Pin = 1 << pin.pin;
    def.GPIO_Mode = dir;
    def.GPIO_Speed = GPIO_Speed_100MHz;

    if (dir)
        def.GPIO_OType = GPIO_OType_PP; //output : Push Pull
    else
        def.GPIO_OType = GPIO_OType_OD; //input : Open Drain

    def.GPIO_PuPd = pull;

    GPIO_Init(stm32f4xx_gpio_ports[pin.port], &def);
}
```

Activer le port

Configurer le pin

Initialiser le pin

```c
static __inline__ pin_t make_pin(gpio_port_t port, uint8_t pin) { pin_t p = { port, pin }; return p; }
#define PIN_NULL { .port = 0×ff, .pin = 0×ff }
static __inline__ bool valid_pin(pin_t pin) { return pin.port < 0×ff || pin.pin < 0×ff; }
```

Utilisation

```
void gpio_set(pin_t pin, int enabled) {
    if (enabled)
        GPIO_SetBits(stm32f4xx_gpio_ports[pin.port], 1 << pin.pin);
    else
        GPIO_ResetBits(stm32f4xx_gpio_ports[pin.port], 1 << pin.pin);
}
```

Exemple

```c
#include <gpio.h>

int main() {
    //Initialize the pin_t structure with the pin port and number
    //On this board there is a LED on PG13
    pin_t pin = make_pin(gpio_port_g, 13);

    //configure the pin for output.
    gpio_config(pin, pin_dir_write, pull_down);

    //set the pin to HIGH
    gpio_set(pin, 1);

   return 0;
}
```

Interruptions

```c
void gpio_irq_init(pin_t pin, void (*cb)(), irq_trigger_t tt)
{
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);

    SYSCFG_EXTILineConfig(pin.port, pin.pin);

    EXTI_InitTypeDef exti;
    exti.EXTI_Line = 1 << pin.pin;
    exti.EXTI_LineCmd = ENABLE;
    exti.EXTI_Mode = EXTI_Mode_Interrupt;
    switch(tt)
    {
        case rising:
            exti.EXTI_Trigger = EXTI_Trigger_Rising;
            break;
        case falling:
            exti.EXTI_Trigger = EXTI_Trigger_Falling;
            break;
        case change:
            exti.EXTI_Trigger = EXTI_Trigger_Rising_Falling;
            break;
    }
    EXTI_Init(&exti);

    NVIC_InitTypeDef nvic;
    nvic.NVIC_IRQChannel = gpio_irq_channels[pin.pin];
    nvic.NVIC_IRQChannelPreemptionPriority = 0x01;
    nvic.NVIC_IRQChannelSubPriority = 0x01;
    nvic.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&nvic);

    EXTI_ClearITPendingBit(1 << pin.pin);

    exti_irq_callback[pin.pin] = cb;
}
```

Configurer la ligne

Attention : un seul interrupt par ligne

Configurer le channel

```c
typedef enum
{
    rising = 0,
    falling = 1,
    change = 2,
} irq_trigger_t;
```

Exemple 2

```c
#include <gpio.h>
#include <stdio.h>

static volatile int status = 0;

void toggleled(){
    status ^= 1;
}

int main() {
    //declare the pin structures for the led and the button
    pin_t led, button;

    //Initialize the pin_t structure with the pin port and number
    //On this board there is a button on PA0
    button = make_pin(gpio_port_a, 0);

    //configure the pin for input.
    gpio_config(button, pin_dir_read, pull_down);

    //attach the callback to the rising trigger on this pin
    gpio_irq_init(button, toggleled, rising);

    //Initialize the pin_t structure with the pin port and number
    //On this board there is a LED on PG13
    led = make_pin(gpio_port_g, 13);

    //Configure the pin for output.
    gpio_config(led, pin_dir_write, pull_down);

    //Loop
    while (1)
        gpio_set(led, status);

    return 0;
}
```

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

Table 6. Timer feature comparison

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

On veut 10Hz

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

$$prescale \simeq 300$$

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

$$count = 59999$$

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

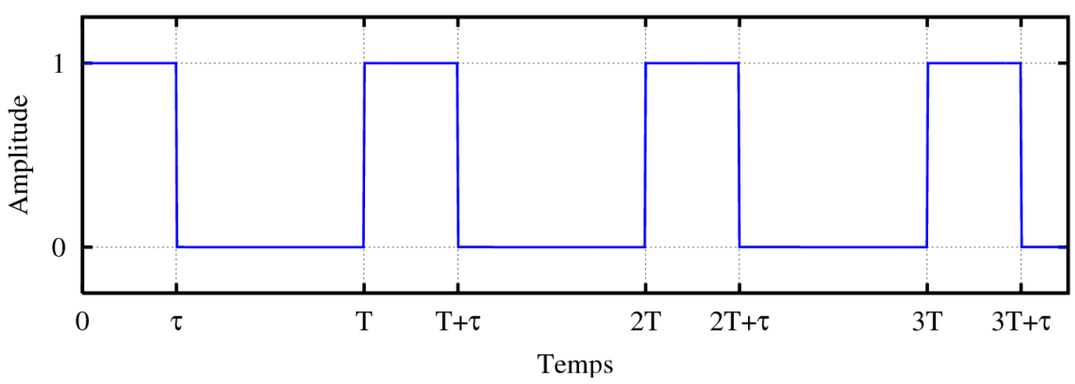
ou

prescale = 1200, count = 14999

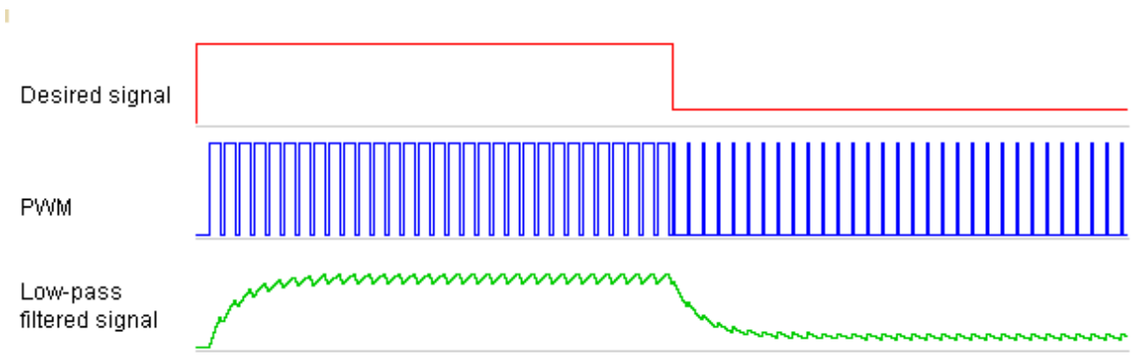
...

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

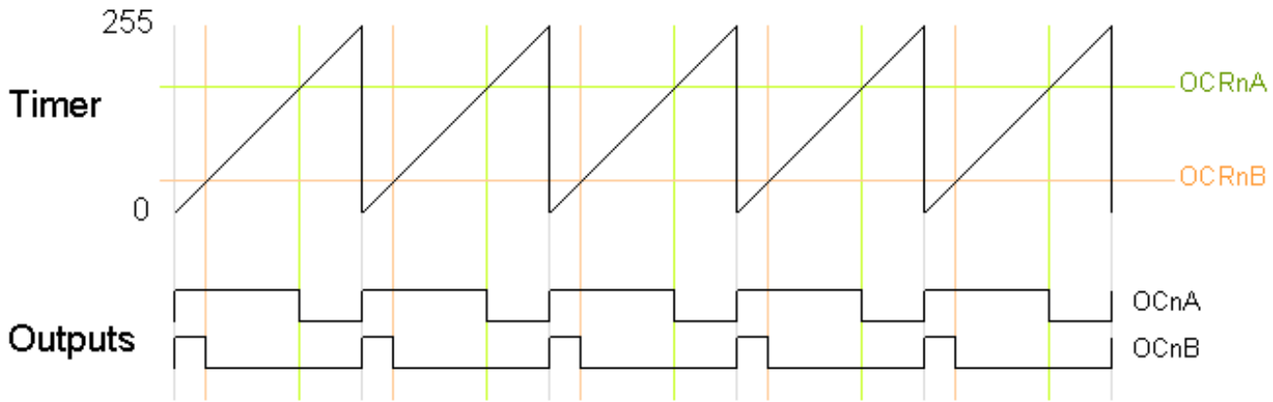
registres de comparaison

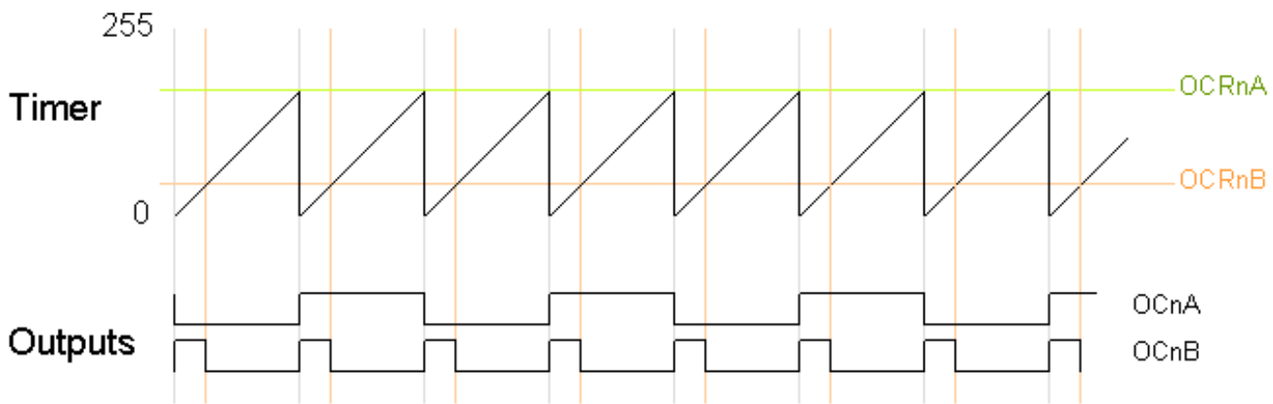
OCnA/OCnB

sorties

Rapide

La phase varie



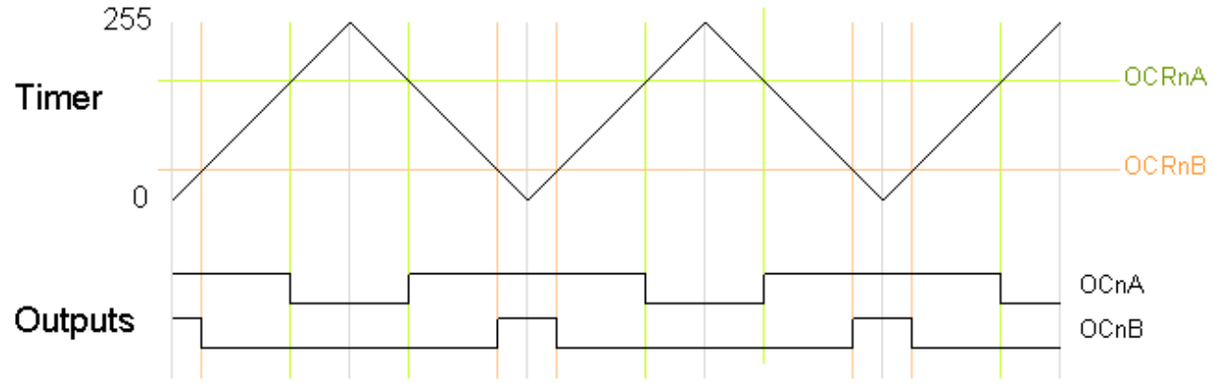


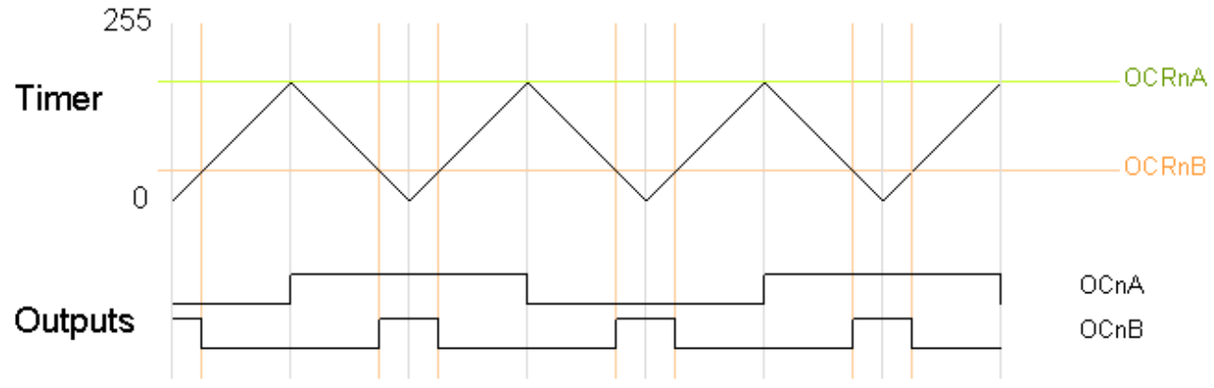
Toggle OCnA on Compare Match

Phase correct PWM

Plus lent

La période est fixe





Configuration

```c
typedef struct {
    timer_t timer;
    uint8_t channel;
} timer_channel_t;
```

```c
typedef enum {
    timer_0,
    timer_1,
    timer_2,
    timer_3,
    timer_4,
    timer_5,
    timer_6,
    timer_7,
    timer_8,
    timer_9,
    timer_10,
    timer_11,
    timer_12,
    timer_13,
    timer_14,
    timer_15,
    timer_16,
    timer_17
} timer_t;
```

```c
void timer_config(timer_t timer, uint16_t prescale, uint32_t period)
{
    TIM_TypeDef *id = timers[timer];
    if (timer < timer_1 || timer > timer_14)
        return;
    //only TIM2 and TIM5 are 32 bits
    if (timer ≠ 2 && timer ≠ 5 && period > 0×ffff)
        return;
    //clock the timer
    switch(timer)
    {
        case timer_1:
            RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);
            break;
        case timer_2:
            RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
            break;
        …
        default:
            return;
    }

    TIM_TimeBaseInitTypeDef def;
    def.TIM_Prescaler = prescale;
    def.TIM_CounterMode = TIM_CounterMode_Up;
    def.TIM_Period = period;
    def.TIM_ClockDivision = TIM_CKD_DIV1;
    def.TIM_RepetitionCounter = 0;          //TIM1 and TIM8 only
    TIM_TimeBaseInit(id, &def);
    TIM_Cmd(id, ENABLE);
}
```

Activer le port

Configurer le timer

Initialiser le timer
Activer le timer

Initialiser un channel

```
void timer_pwmchannel_init(
   timer_channel_t timer_port,
   pin_t pin,
   uint32_t pulse)
{
    timer_t timer = timer_port.timer;
    uint8_t channel = timer_port.channel;

    TIM_TypeDef *id = timers[timer];

 if (timer < 1 || timer > 14 || timer == 6 ||
     timer == 7 || channel < 1 || channel > 4)
     return;

 //only TIM2 and TIM5 are 32 bits
 if (timer ≠ 2 && timer ≠ 5 && pulse > 0×ffff)
     return;

if (timer == 1 || timer == 2)
    gpio_config_alternate(pin, pin_dir_write, pull_none, 1);
else if (timer ⩾ 3 && timer ⩽ 5)
    gpio_config_alternate(pin, pin_dir_write, pull_none, 2);
else if (timer ⩾ 8 && timer ⩽ 11)
    gpio_config_alternate(pin, pin_dir_write, pull_none, 3);
else if (timer ⩾ 9 && timer ⩽ 14)
    gpio_config_alternate(pin, pin_dir_write, pull_none, 9);
```

```
TIM_OCInitTypeDef def;
def.TIM_OCMode = TIM_OCMode_PWM1;
def.TIM_OutputState = TIM_OutputState_Enable;
def.TIM_OutputNState = TIM_OutputState_Enable; //Only TIM1 and TIM8
def.TIM_Pulse = pulse;
def.TIM_OCPolarity = TIM_OCPolarity_High;
def.TIM_OCNPolarity = TIM_OCPolarity_High;     //Only TIM1 and TIM8
def.TIM_OCIdleState = TIM_OCPolarity_High;
def.TIM_OCNIdleState = TIM_OCPolarity_High;    //Only TIM1 and TIM8
switch(channel)
{
    case 1:
        TIM_OC1Init(id, &def);
        TIM_OC1PreloadConfig(id, TIM_OCPreload_Enable);
        break;
    case 2:
        TIM_OC2Init(id, &def);
        TIM_OC2PreloadConfig(id, TIM_OCPreload_Enable);
        break;
    case 3:
        TIM_OC3Init(id, &def);
        TIM_OC3PreloadConfig(id, TIM_OCPreload_Enable);
        break;
    case 4:
        TIM_OC4Init(id, &def);
        TIM_OC4PreloadConfig(id, TIM_OCPreload_Enable);
        break;
    default:
        return;
    }
}
```

Utilisation

```
void timer_enable(timer_t timer)
{
    TIM_Cmd(timers[timer], ENABLE);
}

void timer_disable(timer_t timer)
{
    TIM_Cmd(timers[timer], DISABLE);
}

uint32_t timer_get_count(timer_t timer)
{
    return TIM_GetCounter(timers[timer]);
}

void timer_set_count(timer_t timer, uint32_t value)
{
    TIM_SetCounter(timers[timer], value);
}
```

Exemple

```c
#include <timer.h>

int main() {
    //Initialize the pin_t structure with the pin port and number
    //Pin PB0 is connected to Timer 3 channel 3.
    //Plug an LED on this pin
    pin_t pin = make_pin(gpio_port_b, 0);

    //Initialize Timer 3, channel 3
    timer_channel_t timer = { .timer = timer_3, .channel = 3 };

    //180000000/(18000 * 10000) = 1Hz
    timer_config(timer_3, 18000, 10000);

    //Creates a PWM signal on the pin, duty cycle 8/10
    timer_pwmchannel_init(timer, pin, 8000);

    while(1){}

    return 0;
}
```

Interruption

```
typedef enum {
    event_update,
    event_commutation,
    event_trigger,
    event_break,
    event_output_capture,
}irq_timer_event_t;
```

```
void timer_irq_init(timer_channel_t timer_port, irq_timer_event_t event, void (*cb)());

void timer_irq_deinit(timer_t timer, irq_timer_event_t event);
```

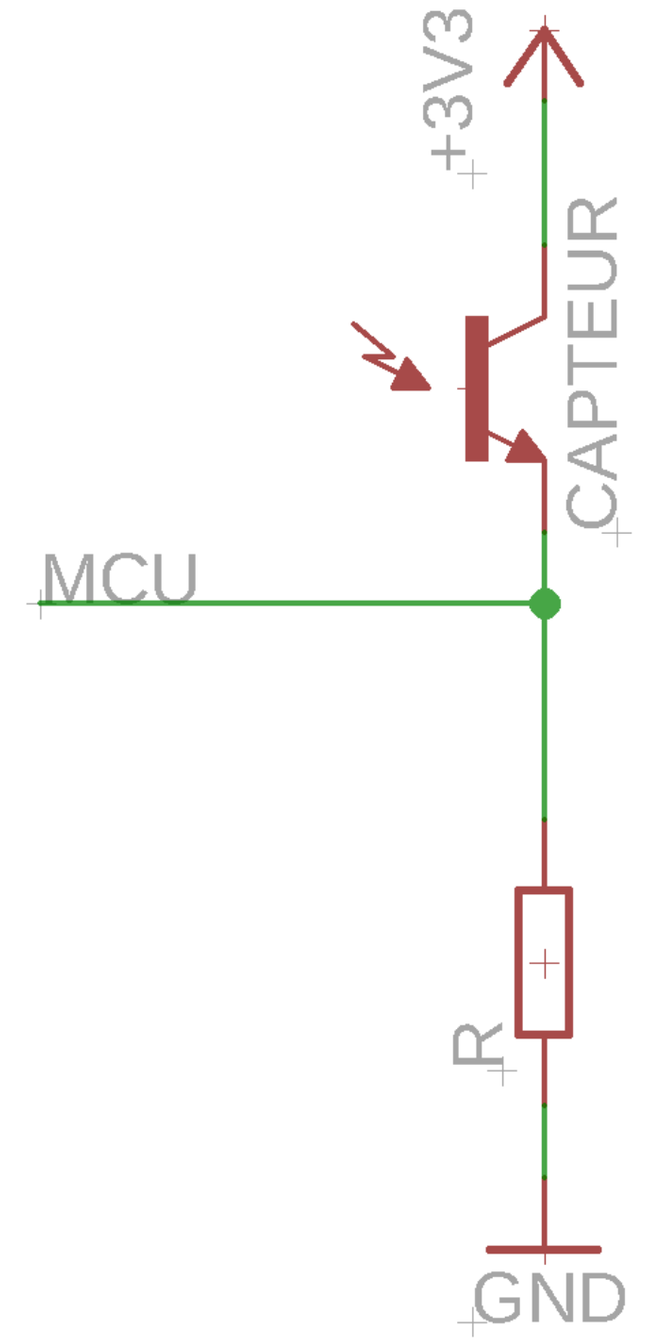
Exemple

```
#include <timer.h>

static uint8_t toggle = 0;

void update() {
    toggle ^= 1;
}

int main() {
    //Initialize the pin_t structure with the pin port and number
    //On this board there is a LED on PG13
    pin_t pin = make_pin(gpio_port_g, 13);

    //configure the pin for output.
    gpio_config(pin, pin_dir_write, pull_down);

    //Initialize Timer 1, channel 1
    timer_channel_t timer = { .timer = timer_1, .channel = 1 };

    //180000000/(18000 * 10000) = 1Hz
    timer_config(timer_1, 18000, 10000);

    //Call the update callback at each timer  update event
    timer_irq_init(timer, event_update, update);

    while(1)
        gpio_set(pin, toggle);

    return 0;
}
```

ADC

- Convertisseur analogique → numérique

- Capteurs simples
  - Micro
  - Potentiomètre
  - Joystick
  - Photorésistance
  - …



Configuration

```
typedef enum {
    adc_0,
    adc_1,
    adc_2,
    adc_3,
} adc_t;
```

```
void adc_config_single(adc_t adc, uint8_t channel, pin_t pin)
{
    if (adc < adc_1 || adc > adc_3 || channel > 18)
        return;

    gpio_config_analog(pin, pin_dir_read, pull_none);

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 << (adc - 1), ENABLE);

    ADC_InitTypeDef def;
    ADC_StructInit(&def);
    def.ADC_DataAlign = ADC_DataAlign_Right;
    def.ADC_Resolution = ADC_Resolution_12b;
    def.ADC_ContinuousConvMode = DISABLE;
    def.ADC_ExternalTrigConv = 0;
    def.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
    def.ADC_NbrOfConversion = 1;
    def.ADC_ScanConvMode = ENABLE;
    ADC_Init(adclist[adc - 1], &def);

    ADC_RegularChannelConfig(adclist[adc - 1], channel, 1, ADC_SampleTime_144Cycles);

    ADC_Cmd(adclist[adc - 1],ENABLE);
}
```

Activer le port

Configurer l'ADC

Initialiser l'ADC

Activer l'ADC

Utilisation

```
uint16_t adc_get(adc_t adc)
{
    if (adc < adc_1 || adc > adc_3)
        return 0;

    ADC_SoftwareStartConv(adclist[adc - 1]);
    while(!ADC_GetFlagStatus(adclist[adc - 1], ADC_FLAG_EOC)){}
    uint16_t res = ADC_GetConversionValue(adclist[adc - 1]);
    ADC_ClearFlag(adclist[adc - 1], ADC_FLAG_EOC);

    return res;
}
```

Exemple

```c
#include <gpio.h>
#include <adc.h>
#include <stdio.h>

int main() {
    //declare the pin structures for the sensor
    //We plug our first sensor on PA0
    pin_t sensor1 = make_pin(gpio_port_a, 0);
    //and the second sensor on PA1
    pin_t sensor2 = make_pin(gpio_port_a, 1);

    //configure the ADC
    //PA0 is connected to ADC 1, 2 or 3, channel 0
    //We use ADC 1
    adc_config_single(adc_1, 0, sensor1);
    //PA1 is connected to ADC 1, 2 or 3, channel 1
    //We use ADC 2
    adc_config_single(adc_2, 1, sensor2);

    //Loop
    while (1)
        printf("x=%4d y=%4d\n", adc_get(adc_1), adc_get(adc_2));

    return 0;
}
```

Configuration avec DMA

Pointeur vers destination

```
void adc_config_continuous(adc_t adc, uint8_t *channel, pin_t *pin, uint16_t *dest, uint8_t nb);
```

Exemple

```c
#include <gpio.h>
#include <adc.h>
#include <stdio.h>

int main() {
    //declare the pin structures for the sensor
    pin_t sensors[2];
    //We plug our first sensor on PA0
    sensors[0] = make_pin(gpio_port_a, 0);
    //and the second sensor on PA1
    sensors[1] = make_pin(gpio_port_a, 1);

    //declare an array where the DMA will put the values
    uint16_t values[] = {0, 0};

    //configure the ADC
    //PA0 is connected to ADC 1, 2 or 3, channel 0
    //PA1 is connected to ADC 1, 2 or 3, channel 1
    //We use ADC 1, with channels 0 and 1
    uint8_t channels[] = {0, 1};
    adc_config_continuous(adc_1, channels, sensors, values, 2);

    //Loop
    while (1)
        printf("x=%4d y=%4d\n", values[0], values[1]);

    return 0;
}
```

Bus

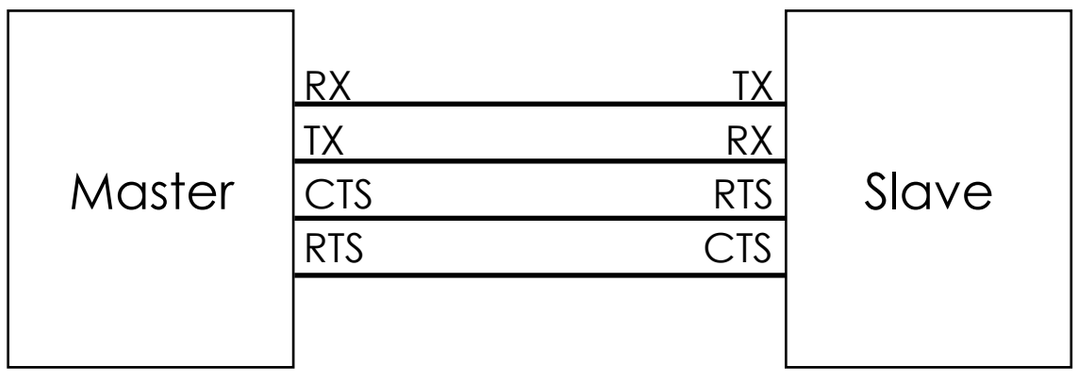
- Communication entre MCU
- Communication avec un PC
- Sérialisation / parallélisation

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

Configuration

```c
typedef struct {
    pin_t rx;
    pin_t tx;
    uart_t uart;
} uart_port_t;
```

```c
typedef enum {
    uart_port_0,
    uart_port_1,
    uart_port_2,
    uart_port_3,
    uart_port_4,
    uart_port_5,
    uart_port_6,
    uart_port_7,
    uart_port_8,
} uart_t;
```

```c
void uart_config(uart_port_t uart_port, uint32_t baudrate)
{
    if (uart_port.uart > 6)
        return;
    uart_t uart = uart_port.uart;
    pin_t rx = uart_port.rx;
    pin_t tx = uart_port.tx;

    USART_TypeDef * id = uarts[uart];

    switch (uart) {
    case uart_port_1:
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
        break;
    …
    default:
        return;
    }

    if (uart_port.uart ⩽ 3){
        gpio_config_alternate(rx, pin_dir_write, pull_up, 7);
        gpio_config_alternate(tx, pin_dir_write, pull_up, 7);
    }
    else{
        gpio_config_alternate(rx, pin_dir_write, pull_up, 8);
        gpio_config_alternate(tx, pin_dir_write, pull_up, 8);
}

    USART_InitTypeDef uartdef;
    uartdef.USART_BaudRate = baudrate;
    uartdef.USART_WordLength = USART_WordLength_8b;
    uartdef.USART_StopBits = USART_StopBits_1;
    uartdef.USART_Parity = USART_Parity_No;
    uartdef.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    uartdef.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
    USART_Init(id, &uartdef);
```

Activer le port

Configurer les pins

Configurer l'UART

Initialiser l'UART
Activer l'UART

Utilisation

```
void uart_send_char(uart_t uart, uint8_t c)
{
    USART_TypeDef * id = uarts[uart];

    while (USART_GetFlagStatus(id, USART_FLAG_TXE) == RESET);
    USART_SendData(id, c);
}

uint8_t uart_receive_char(uart_t uart)
{
    USART_TypeDef * id = uarts[uart];

    while (USART_GetFlagStatus(id, USART_FLAG_RXNE) == RESET);
    return (uint8_t) USART_ReceiveData(id);
}
```

Exemple

```c
#include <uart.h>

void BoardConsoleInit() {
    pin_t rx = make_pin(gpio_port_a, 9);
    pin_t tx = make_pin(gpio_port_a, 10);

    uart_port_t uart = { .uart = uart_port_1, .rx = rx, .tx = tx };
    uart_config(uart, 115200);
}

void BoardConsolePutc(int c) {
    if (c == '\r') return;
    if (c == '\n') c = '\r';

    uart_send_char(uart_port_1, c);

    if (c == '\r') {
        c = '\n';
        uart_send_char(uart_port_1, c);
    }
}
```

Cf arch/arm/stm32f4xx/stm32f429discovery/BoardConsole.c

SPI

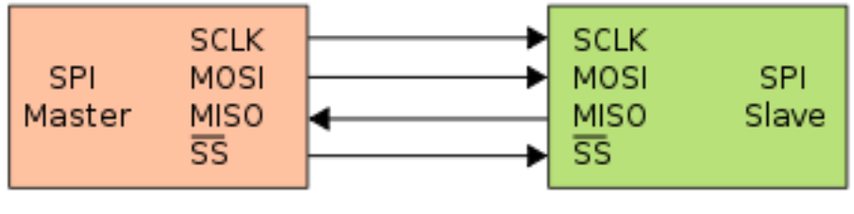
Serial Peripheral Interface

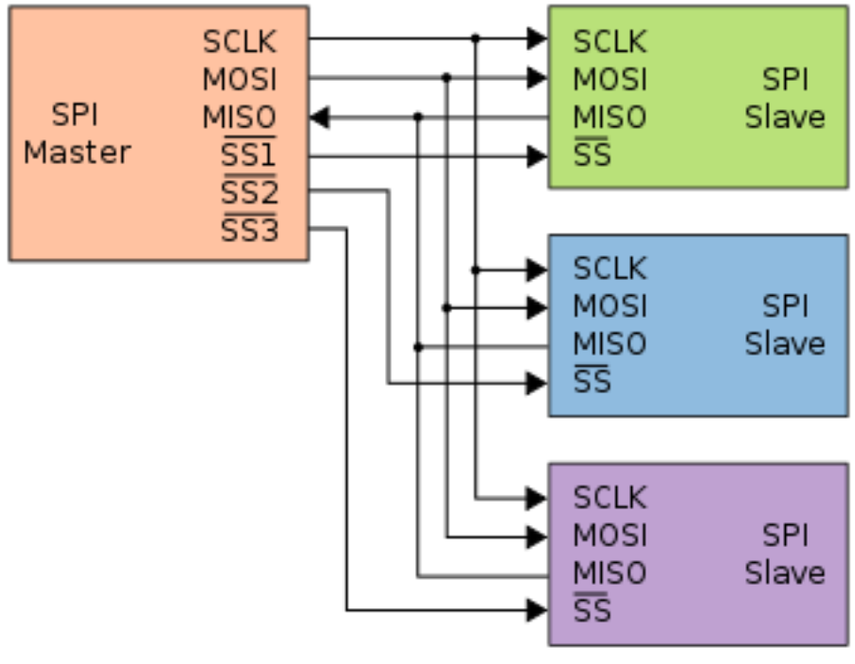
- SS/CS : slave select
- SCK : horloge
- MISO/SDO : master in slave out
- MOSI/SDI : master out slave in

Rapide

Bi-directionnel

Petites distances





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

Configuration

```
typedef struct {
    pin_t sclk;
    pin_t mosi;
    pin_t miso;
    pin_t ss;
    ssp_mode_t mode;
    ssp_polarity_t polarity;
    ssp_t ssp;
} ssp_port_t;


typedef enum {
    ssp_0,
    ssp_1,
    ssp_2,
    ssp_3,
    ssp_4,
    ssp_5,
    ssp_6,
} ssp_t;


typedef enum {
    ssp_slave,
    ssp_master,
} ssp_mode_t;


typedef enum {
    ssp_polarity_mode_0, // CPOL=0; CPHA=0
    ssp_polarity_mode_1, // CPOL=0; CPHA=1
    ssp_polarity_mode_2, // CPOL=1; CPHA=0
    ssp_polarity_mode_3, // CPOL=1; CPHA=1
} ssp_polarity_t;
```

```
void ssp_config(ssp_port_t ssp_port, uint32_t clock);
```

Utilisation

```
uint8_t ssp_readwrite(ssp_t ssp, uint8_t value) {
    SPI_TypeDef * id = spis[ssp];

    while (SPI_I2S_GetFlagStatus(id, SPI_I2S_FLAG_TXE) == RESET);
    SPI_I2S_SendData(id, value);
    while (SPI_I2S_GetFlagStatus(id, SPI_I2S_FLAG_RXNE) == RESET);
    return SPI_I2S_ReceiveData(id);
}

static __inline__ void ssp_write(ssp_t ssp, uint8_t value) { (void) ssp_readwrite(ssp, value); }
static __inline__ uint8_t ssp_read(ssp_t ssp) { return ssp_readwrite(ssp, 0×ff); }
```

Exemple master

```c
#include <gpio.h>
#include <ssp.h>
#include <stdio.h>

pin_t led1, led2;

uint8_t rled = 0, tled = 0;

void buttonpress(){
    //master write
    ssp_write(ssp_4, 0x42);
    //toggle the transmit LED
    tled ^= 1;
    gpio_set(led1, tled);

    //master read
    volatile uint8_t r = ssp_read(ssp_4);
    printf("received %x\n", r);
    //toggle the receive LED
    rled ^= 1;
    gpio_set(led2, rled);
}
```

```c
int main(){
    //LEDs on PG13 and PG14
    led1 = make_pin(gpio_port_g, 13);
    led2 = make_pin(gpio_port_g, 14);
    gpio_config(led1, pin_dir_write, pull_down);
    gpio_config(led2, pin_dir_write, pull_down);

    //SPI4
    pin_t sclk = make_pin(gpio_port_e, 2);
    pin_t miso = make_pin(gpio_port_e, 5);
    pin_t mosi = make_pin(gpio_port_e, 6);

    ssp_port_t master = {
        .ssp = ssp_4,
        .sclk = sclk,
        .mosi = mosi,
        .miso = miso,
        .ss = PIN_NULL,
        .mode = ssp_master,
        .polarity = ssp_polarity_mode_0
    };
    ssp_config(master, 8000000);

    //user button on PA0
    pin_t button = make_pin(gpio_port_a, 0);
    gpio_config(button, pin_dir_read, pull_down);
    gpio_irq_init(button, buttonpress, rising);

    while(1);

    return 0;

}
```

Slave : IRQ

```
typedef enum {
    event_read,
    event_write,
}irq_ssp_event_t;
```

```
void ssp_irq_init(ssp_t ssp, irq_ssp_event_t event, void (*cb)());
void ssp_slave_start_read(ssp_t ssp);
void ssp_slave_stop_read(ssp_t ssp);
void ssp_slave_start_write(ssp_t ssp);
void ssp_slave_stop_write(ssp_t ssp);
```

Exemple slave

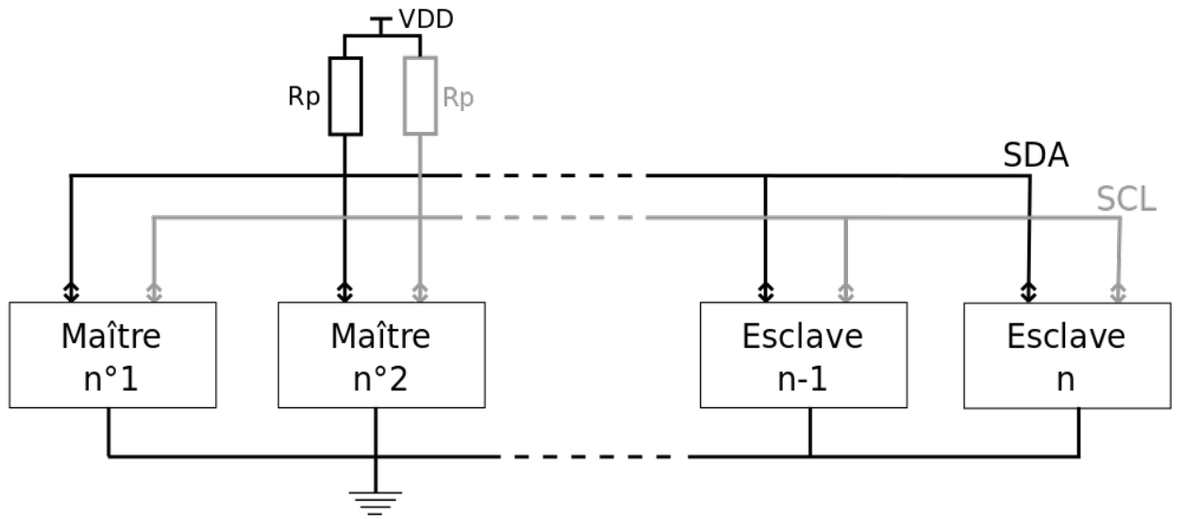
```c
#include <gpio.h>
#include <ssp.h>
#include <stdio.h>

pin_t led1, led2;
uint8_t rled = 0, tled = 0;

void slavereceived(){
    volatile uint8_t r = ssp_read(ssp_4);
    printf("received %x\n", r);
    //toggle the receive LED
    rled ^= 1;
    gpio_set(led2, rled);

    ssp_slave_stop_read(ssp_4);
    //slave write
    ssp_slave_start_write(ssp_4);
}

void slavesent(){
    ssp_write(ssp_4, 0x28);
    //toggle the transmit LED
    tled ^= 1;
    gpio_set(led1, tled);

    ssp_slave_stop_write(ssp_4);
    //slave read
    ssp_slave_start_read(ssp_4);
}
```

```c
int main(){
    //LEDs on PG13 and PG14
    led1 = make_pin(gpio_port_g, 13);
    led2 = make_pin(gpio_port_g, 14);

    gpio_config(led1, pin_dir_write, pull_down);
    gpio_config(led2, pin_dir_write, pull_down);

    //SPI4
    pin_t sclk = make_pin(gpio_port_e, 2);
    pin_t miso = make_pin(gpio_port_e, 5);
    pin_t mosi = make_pin(gpio_port_e, 6);

    ssp_port_t slave = {
        .ssp = ssp_4,
        .sclk = sclk,
        .mosi = mosi,
        .miso = miso,
        .ss = PIN_NULL,
        .mode = ssp_master,
        .polarity = ssp_polarity_mode_0
    };
    ssp_config(slave, 8000000);

    //SSP slave callbacks
    ssp_irq_init(ssp_4, event_read, slavereceived);
    ssp_irq_init(ssp_4, event_write, slavesent);

    //slave read
    ssp_slave_start_read(ssp_4);

    while(1);

    return 0;

}
```

I2C

Inter Integrated Circuit

- SDA : données
- SCL : horloge



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

Masse commune

Plus lent que SPI

Peu de fils

Jusqu'à 256 périphérique par bus

Configuration

```c
typedef struct {
    pin_t scl;
    pin_t sda;
    i2c_t i2c;
} i2c_port_t;
```

```c
typedef enum {
    i2c_0,
    i2c_1,
    i2c_2,
    i2c_3,
} i2c_t;
```

```c
void i2c_config(i2c_port_t i2c_port, uint32_t speed)
{
    i2c_t i2c = i2c_port.i2c;
    pin_t scl = i2c_port.scl;
    pin_t sda = i2c_port.sda;

    I2C_TypeDef *id = i2cs[i2c];

    RCC->APB1ENR |= RCC_APB1Periph_I2C1 << (i2c - 1);

    gpio_config_alternate(sda, pin_dir_read, pull_down, 4);
    gpio_config_alternate(scl, pin_dir_read, pull_down, 4);

    //Init I2C
    I2C_InitTypeDef i2cdef;
    i2cdef.I2C_Mode = I2C_Mode_I2C;
    i2cdef.I2C_DutyCycle = I2C_DutyCycle_2;
    i2cdef.I2C_OwnAddress1 = 0x00;
    i2cdef.I2C_Ack = I2C_Ack_Enable;
    i2cdef.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
    i2cdef.I2C_ClockSpeed = speed;

    I2C_Init(id, &i2cdef);
    I2C_Cmd(id, ENABLE);
}
```

Activer le port

Configurer les pins

Configurer l'I2C

Initialiser l'I2C

Activer l'I2C

Utilisation typique

```
void i2c_read_register(i2c_t i2c, uint8_t device, uint8_t reg, uint8_t *buffer, uint8_t nb) {
    if(nb > 1)
        reg |= 0x80;
    i2c_wait(i2c);
    i2c_start_write(i2c, device);
    i2c_write(i2c, &reg, 1);
    i2c_start_read(i2c, device);
    i2c_read(i2c, buffer, nb);
    i2c_stop(i2c);
}

void i2c_write_register(i2c_t i2c, uint8_t device, uint8_t reg, uint8_t *buffer, uint8_t nb) {
    if(nb > 1)
        reg |= 0x80;
    i2c_start_write(i2c, device);
    i2c_write(i2c, &reg, 1);
    i2c_write(i2c, buffer, nb);
    i2c_stop(i2c);
}
```

Cf lsm303dlhc dans chips/

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

…

Gyroscope

```c
#include <stdio.h>
#include <l3gd20.h>

pin_t led_w, led_e;
l3gd20_t l3gd20;
ssp_port_t ssp;

void init() {
    led_w = make_pin(gpio_port_g, 13);
    led_e = make_pin(gpio_port_g, 14);

    gpio_config(led_w, pin_dir_write, pull_up);
    gpio_config(led_e, pin_dir_write, pull_up);
    gpio_set(led_w, 0);
    gpio_set(led_e, 0);


    pin_t sclk = { .port = gpio_port_f, .pin = 7 };
    pin_t mosi = { .port = gpio_port_f, .pin = 9 };
    pin_t miso = { .port = gpio_port_f, .pin = 8 };
    pin_t cs   = { .port = gpio_port_c, .pin = 1 };

    ssp = {
        .ssp = ssp_5,
        .sclk = sclk,
        .mosi = mosi,
        .miso = miso,
        .mode = ssp_master,
        .polarity = ssp_polarity_mode_0 };

    if (!l3gd20_init_ssp(&l3gd20, ssp, cs))
    {
        printf("Cannot initialize gyroscope");
        return 0;
    }
}

int main() {
    init();

    float axis[3];
    int e = 0, w = 0;
    while (1) {
        l3gd20_read(&l3gd20, axis);

        if (axis[1] ⩾ 3000.0f) {
            e = 1;
            w = 0;
        } else if (axis[1] ⩽ -3000.0f) {
            e = 0;
            w = 1;
        } else if (axis[1] ⩾ -2000.0f && axis[1] ⩽ 2000.0f) {
            e = 0;
            w = 0;
        }

        gpio_set(led_e, e);
        gpio_set(led_w, w);
    }

    return 0;
}
```

FreeRTOS

- Threads (Tasks)
- Timers software
- Sémaphores
- …

FreeRTOS - threads

```c
#include <malloc.h>
#include <FreeRTOS.h>
#include <task.h>

#include <gpio.h>

void callback(void *parameter) {
    uint8_t status = 0;
    //Cast the parameter back to its origin type
    pin_t *pin = (pin_t *)parameter;
    while(1) {
        status ^= 1;
        gpio_set(*pin, status);
        //sleep
        vTaskDelay(1000);
    }
}
```

```c
int main() {
    //Initialize the pin_t structure with the pin port and number
    //On this board there is a LED on PG13
    pin_t *pin = malloc(sizeof(pin_t));
    *pin = make_pin(gpio_port_g, 13);

    //configure the pin for output.
    gpio_config(*pin, pin_dir_write, pull_down);

    //Create the task
    xTaskCreate(callback,           //callback function
        (const signed char *)NULL,  //task name
        configMINIMAL_STACK_SIZE,   //stack size
        (void *)pin,                //parameter (cast it to void *)
        tskIDLE_PRIORITY,
        NULL);

    //Run the tasks
    vTaskStartScheduler();

    return 0;
}
```