

Exercice 1

La structure des GPIOs de la famille F4 est différente de celle de la famille F1. Les bits de configuration de chacun des 16 pins ne sont pas regroupés ensemble, mais sont séparés dans plusieurs registres.

Ainsi, on trouve pour chaque pin

- 2 bits pour la configuration du mode (Input, Output ou Alternate Function) dans le registre **GPIOx_MODER**
- 2 bits pour la configuration de la vitesse pour chaque pin dans le registre **GPIOx_OSPEEDR**
- Etc... (Voir Annexe)

Q1- Donner la partie à ajouter au fichier `stm32f4xx.h` qui devrait contenir le memory map. (voir annexe- partie1 pour adresses et offset /)

GPIOx	A	B	C	D
@	0x4002 0000	0x4002 0400	0x4002 0800	0x4002 0C00

Registre	MODER	OTYPER	OSPEEDR	PUPDR	IDR	ODR	BSRR	
Offset	0x00	0x04	0x08	0x0C	0x10			

Réponse : Au niveau de ce fichier il faut ajouter le memory map c.à.d une description des registres de l'interface ainsi que leurs adresses.

@ de ① } #define GPIOA_Base 0x40020000
Base des }
GPIO } #define GPIOB_Base 0x40020400

② } Structure des GPIO (Regs) {
typedef struct
{
uint32_t MODER;
uint32_t OTYPER;
uint32_t OSPEEDR;
uint32_t PUPDR;
uint32_t IDR;
uint32_t ODR;
uint32_t BSRR;
}
GPIO_TypeDef

// il faut respecter l'ordre
// (les offsets)

③ On définit des pointeurs sur la structure `GPIO_TypeDef`

```
#define GPIOA ((GPIO_TypeDef*) GPIOA_Base)
```

```
#define GPIOB ((GPIO_TypeDef*) GPIOB_Base)
```

// ainsi on peut accéder à un Register en utilisant
// `GPIOx → Reg` (exple: `GPIOA → IDR`).

Q2 – Ecrire un programme qui permet de faire clignoter une Led connectée au pin PD12 en utilisant l'accès direct aux registres (voir annexe- partie 2 pour BSRR).

GPIO port bit set/reset register (GPIOx_BSRR) (x = A..I/J/K)

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 **BRy**: Port x reset bit y (y = 0..15)

These bits are write-only and can be accessed in word, half-word or byte mode. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: Resets the corresponding ODRx bit

Note: If both BSx and BRx are set, BSx has priority.

Bits 15:0 **BSy**: Port x set bit y (y = 0..15)

These bits are write-only and can be accessed in word, half-word or byte mode. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: Sets the corresponding ODRx bit

Réponse

On remarque que contrairement au GPIO de la famille F1, les GPIO de F4 ont un unique Reg pour les opérations Set et Reset: les bits 0..15 pour le Set
— 16..31 pour le Reset

⇒ pour mettre à 1 le Pin PD12 ⇒ Mettre à 1 bit 12
— 0 — ⇒ — 1 bit 28

Ainsi le programme serait

```
while(1)
{
    GPIOB → BSRR = 0x1000;
    delay(...);
    GPIOB → BSRR = 0x10000000;
    delay(...);
}
```

Q3- Donner le code à ajouter au fichier stm32f4xx_gpio.h et qui permet de décrire les paramètres de configuration du GPIO (une structure).

Réponse Le GPIO contient (d'après l'annexe) 4 Registres de config.

Ainsi le type GPIO_InitTypeDef qui décrit les propriétés des pins du GPIO contient 4 paramètres en plus du type GPIO_Pin.

⇒ typedef struct

Ces 4 types relatifs aux 4 param. doivent être définis en fait des structures des Regs (voir ci-dessous)

```

uint8_t  GPIO_Mode;
          GPIO_Output;
          GPIO_Speed;
          GPIO_PuPd;
uint16_t  GPIO_Pin;
}
GPIO_InitTypeDef
    
```

erreur qui concerne GPIO_Pin
ça sera comme pin FA:

```

#define GPIO_Pin_0 0x0001
#define           1 0x0002
...
#define           15 0x8000
    
```

GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]	MODER14[1:0]	MODER13[1:0]	MODER12[1:0]	MODER11[1:0]	MODER10[1:0]	MODER9[1:0]	MODER8[1:0]								
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]	MODER6[1:0]	MODER5[1:0]	MODER4[1:0]	MODER3[1:0]	MODER2[1:0]	MODER1[1:0]	MODER0[1:0]								
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 MODERy[1:0]: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.

00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

```

uint8_t input = 0x00;
        GP_output = 0x01;
        AF       = 0x02;
        Analog   = 0x03;
    
```

doit les différentes valeurs possibles de la propriété GPIO_Mode.

GPIO port output type register (GPIOx_OTYPER) (x = A..I/J/K)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **OTy**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the output type of the I/O port

0: Output push-pull (reset state)

1: Output open-drain

`vint8_t OutPP = 0x00;`

`vint8_t OutOD = 0x01;`

GPIO port output speed register (GPIOx_OSPEEDR) (x = A..I/J/K)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OSPEEDR7[1:0]		OSPEEDR6[1:0]		OSPEEDR5[1:0]		OSPEEDR4[1:0]		OSPEEDR3[1:0]		OSPEEDR2[1:0]		OSPEEDR1 [1:0]		OSPEEDR0 [1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **OSPEEDRy[1:0]**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O output speed.

00: Low speed

01: Medium speed

10: Fast speed

11: High speed

`vint8_t Lowspd = 0x00;`

`—— Medspd = 0x01;`

`—— Fastspd = 0x02;`

`—— Highspd = 0x03;`

GPIO port pull-up/pull-down register (GPIOx_PUPDR) (x = A..I/J/K)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 PUPDRy[1:0]: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O pull-up or pull-down

00: No pull-up, pull-down

01: Pull-up

10: Pull-down

uint8_t NoPU_PD = 0x00;

— PU = 0x01;

— PD = 0x02;

4 – Donner le code de la fonction GPIO_Init qui prend en paramètre le GPIO à configurer ainsi que les paramètres de la structure

// On rappelle que cette fct sera appelée au niveau du main.c
// pour l'initialisation du GPIO. Par exple :

// GPIO_InitTypeDef GPIO_InitStructure;

// GPIO_InitStructure.GPIO_Pin = GPIO_Pin_i | GPIO_Pin_j | ... ;

— .GPIO_Mode = ... ;

— .GPIO_Output = ... ;

— .GPIO_Speed = ... ;

— .GPIO_PuPd = ... ;

// **GPIO_Init**(GPIO , &GPIO_InitStructure);

Cas 1 : Tous les bits des registres à configurer sont à 0

La fonction

```
void GPIO_Init (GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_Struct) ;
```

```
{
```

```
// On commence à initialiser à 0 des variables qui contiendront les configs des différentes
```

```
// paramètres (mode, speed, pupd et outtype)
```

```
uint32_t      mode, speed, pupd, outtype;
```

```
mode = speed = pupd = outtype = 0 ;
```

```
For (int i =0 ; i<=15 ; i++)      // Tester la valeur de chacun des 16 bits de « GPIO_Struct -> GPIO_Pin »
```

```
//afin de détecter les pins concernés par la configuration.
```

```
{ int  pos = (GPIO_InitStruct-> GPIO_Pin) & (1<<i) ;
```

```
if (pos !=0) { // si pin i est à configurer
```

```
    mode |= GPIO_InitStruct -> Mode << 2*i ;
```

```
    speed |= GPIO_InitStruct -> OSPEED << 2*i ;
```

```
    pupd |= GPIO_InitStruct -> PuPd << 2*i ;
```

```
    outtype |= GPIO_InitStruct -> outtype << i ;
```

```
    }
```

```
}
```

```
GPIOx-> MODER |= mode;      // imposer les valeurs des paramètres « les 1 »
```

```
GPIOx-> OSPEEDR |= speed;
```

```
GPIOx-> PUPDR |= pupd;
```

```
GPIOx-> OTyPER |= outtype;
```

```
}
```

Cas 2: Les valeurs des bits sont inconnues

```
void GPIO_Init (GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_Struct);
```

```
{
```

```
// On commence à initialiser à 0 des variables qui contiendront les configs des différentes
```

```
// paramètres (mode, speed, pupd et outtype)
```

```
uint32_t      mode, speed, pupd, outtype, maskzeros32, maskzeros16 ;
```

```
mode = speed = pupd = outtype = 0 ;
```

```
For (int i=0 ; i<=15 ; i++)      // Tester la valeur de chacun des 16 bits de « GPIO_Struct -> GPIO_Pin »  
                                // afin de détecter les pins concernés par la configuration.
```

```
{ int  pos = (GPIO_InitStruct-> GPIO_Pin) & (1<<i) ;
```

```
if (pos !=0) { // si pin i est à configurer
```

```
    maskzeros32 |= 0x3 << 2*i ; // 0x3 = (11)2
```

```
    maskzeros16 |= 0x1 << i ;
```

```
    mode |= GPIO_InitStruct -> Mode << 2*i ;
```

```
    speed |= GPIO_InitStruct -> OSpeed << 2*i ;
```

```
    pupd |= GPIO_InitStruct -> PuPd << 2*i ;
```

```
    outtype |= GPIO_InitStruct -> outtype << i ;
```

```
    }
```

```
}
```

```
GPIOx-> MODER &= ~ maskzeros32 ; // imposer les 0 au niveau des bits à configurer
```

```
GPIOx-> MODER |= mode;          // imposer les valeurs des paramètres « 1 »
```

```
GPIOx-> OSPEEDR &= ~ maskzeros32;
```

```
GPIOx-> OSPEEDR |= speed;
```

```
GPIOx-> PUPDR &= ~ maskzeros32;
```

```
GPIOx-> PUPDR |= pupd;
```

```
GPIOx-> OTYPER &= ~ maskzeros16;
```

```
GPIOx-> OTYPER |= outtype;
```

```
}
```


Q5 – Donner les fonctions GPIO_SetBits et GPIO_ResetBits qui permettent de mettre à 1 (resp à 0) un pin ou un ensemble de pins d'un GPIO donné.

// Les bits 0..15 du registre BSRR étant utilisés pour la mise à 1 (voir 2), on a :

```
Void GPIO_SetBits (GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{
    GPIOx->BSRR=GPIO_Pin;
}
```

// Les bits 16..31 du registre BSRR étant utilisés pour la mise à 0 (voir 2), on a :

```
Void GPIO_ResetBits (GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{
    GPIOx->BSRR= (GPIO_Pin<<16);    // Décalage de 16 positions
}
```