**Le Port GPIO**

GPIO (General Purpose Inputs/Outputs)

Configurer les horloges de périphériques, les GPIOS et la pin en lecture/écriture sur le GPIO.

Rappels de syntaxes :

Préfixe = la base

Les écritures en décimal :42 en octale :052 en hexadécimal : 0x2A en binaire :0b101010.

Suffixe = le codage machine

non signé suffixe u : 42u ->32bits

signé pas de suffixe :42 -> complément à deux 32bits

long suffixe l :42.0u -> complément à deux 64 bits

float suffixe f :42.0f -> IEE 754 sir 32 bits

double suffixe d:42.0d -> indisponible sur stm32f3

Opérations bits à bits (opération binaire des entiers)

Décalage Logique droite Décalage Logique gauche

01011110 -> <- 01011110

>>1 = 001011110 <<1 = 010111100

Décalage shift décale tous les chiffres et le vide est remplis avec des zéros.

Dans ce cas un décalage vers la gauche de n rangs équivaux à une multiplication par pour les non-signés ou entier positif.

Dans ce cas un décalage vers la droite de n rangs équivaux à une division par pour les entiers positifs. Ce n’est pas le cas pour les non-signés, on obtient un résultat qui n’a pas grand sens mathématiquement.

Ce type d’opération s’avère plus rapide que des instructions de multiplication ou division.

Détails : Lorsqu'on effectue un décalage à droite -, certains bits de notre nombre vont sortir du résultat et être perdus. Cela a une conséquence : le résultat est tronqué ou arrondi. Plus précisément, le résultat d'un décalage d’un décalage à droite de n rangs sera égal à la partie entière du résultat de la division par .

Décalage Arithmétique

Similaire au décalage logique à un détail près : Celui de droite, le bit de signe n’est pas modifié et on remplit les vides par le décalage avec le bit de signe.

Décalage Logique droite Décalage Logique gauche

11011110 100111100

>>1 = 111011110 <<1 = 10111100

Ces instructions sont équivalentes à une multiplication/division par que le nombre soit signé ou non. A un détail près l’arrondi n’est pas fait de la même façon pour les nombres positifs et négatifs (9/2=4 et (-9)/2=-5)

Rotation

Similaire mais les bits sortant sont réinjécté à la place libérée par le décalage. Utile en cryptographie algorithme de chiffrement ou si on doit manipuler des données bit par bit comme un calcul du nombre de bit à 1 dans un nombre.

Mais dans le code ?

En java il n’y a pas de distinction de décalage à gauche (logique ou arithmétique) ont le même effet donc (<<)

Droite (>>> logique) (>> arithmétique)

En C/C++ le type est déterminé par le type d’entier à décaler. Les entiers signés décalage arithmétique tandis que la logique est utilisée sur les entiers non signés.

Décalages

0b100101 0b101101

<< 3 = 0b100101000 >> 2 = 0b001101

Opérations bits-à-bits

Conjonctions & ET logique 0b1010 [0&0=0] [0&1=0] [1&1=1] (Test)

&0b1100

= 0b1000

Disjonction | OU 0b1010 [0|0=0] [0|1=1] [1|1=1] (Mettre)

|0b1100

= 0b1110

Xor ^ 0b1010 [0^0=0] [0|1=1] [1|1=0]

^0b1100

= 0b0110

Négation ~ 0b1010 = 0b0101

Les masques et les opérations sur les registres

GPIOA -> ODR = (1<<5) //on affecte 1 au bit du rang 5 sur le registre ODR du GPIOA.

GPIOA->MODER &= 0xFFFFF7FF ou GPIOA->MODER &= 0<<11

1010 1011 1111 1111 1111 1111 1111 1111 0XABFF FFFF

& 1111 1111 1111 1111 1111 0111 1111 1111 0xFFFF F7FF

-----------------------------------------------------------------------------

1010 1011 1111 1111 1111 **01**11 1111 1111

Pour une question d’écriture et de préférence :

GPIOA->ODR |= (1 << 5); /\* mise à 1 du bit n°5 \*/ moi je préfère

GPIOA->ODR |= 0x20; /\* mise à 1 du bit n°5 \*/ ce qui revient exactement au même.  
 OU 0000 0000

0x20 = 0010 0000 = 0010 0000

GPIOA->ODR &= ~(1 << 5); /\* mise à 0 du bit n°5 \*/ moi j'écrirai

GPIOA->ODR &= 0xDF; /\* mise à 0 du bit n°5 \*/ ce qui revient exactement au même.  
 ET 1111 1111

0xDF = 1101 1111 = 1101 1111

Directions et Configurations des ports

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Direction | Mode | Description | Valeur du champ | Détails |
| input | Analog | Permet une acquisition directe de la tension (l’état binaire n’est plus connu sur le registre) | 0000 |  |
| input | Floating input | Une entrée si aucune tension n’est appliquée, la valeur n’est pas connue | 0100 |  |
| input | With pull-up/pull-down | Entré pour lequel la valeur il est possible de fixer l’état par défaut à 0 (pulldown) 1(pull-up) | 1000 |  |
| input | Analog | Non utilisé | 1100 |  |
| output | Push-pull | Sortie dans lequel la tension appliquée à la broche est toujours forcée à 0 ou 1 | 0001  0010  0011 |  |
| output | Open-drain | Sortie dans lequel le niveau de tension haut est fixé par le circuit connecté à la broche | 0101  0110  0111 |  |
| output | Alternate function push-pull | Sortie identique au mode push-pull, mais l’état de la broche est contrôlé par un autre périphérique que le port | 1001  1010  1011 |  |
| output | Alternate function open-drain | Sortie identique en mode open-drain, mais l’état de la broche est contrôlé par un autre périphérique que le port. | 1101  1110  1111 |  |
| OUTPUT |  |  |  |  |
| Push-pull | mode de sortie dans lequel la tension appliquée à la broche est  toujours forcée à 0 ou 1 | | | |
| Open-drain | mode de sortie dans lequel le niveau de tension haut est fixé par le circuit connecté à la broche | | | |
| Alternate function push-pull | Mode de sortie identique au mode push-pull mais l’état de la broche est contrôlé par un autre périphérique que le port | | | |
| Alternate function open-drain | mode de sortie identique au mode open-drain, mais l’état de la broche est contrôlé par un autre périphérique que le port | | | |
| Mode  GPIO\_MODE\_OUTPUT\_PP :Output Push Pull Mode  GPIO\_MODE\_OUTPUT\_OD :Output Open Drain Mode  GPIO\_MODE\_AF\_PP :Alternate Function Push Pull Mode  GPIO\_MODE\_OUTPUT\_OD : Alternate Function Open Drain Mode  Comprendre la différence entre le Push-Pull et Open-Drain :  Si vous choisissez un drain ouvert, la broche de sortie peut être LOW (Gnd) ou flottant tandis  si vous choisissez Push Pull, la broche de sortie peut être HAUT ou BAS.  <https://controllerstech.com/stm32-gpio-output-config-using-registers/> | | |  |  |
|  |  |  |  |  |

Il y a 16 broches sur un port, il faut 4 bits par broche pour configurer e/s. soit 64 bits pour configurer l’ensemble des broches d’un port. Or un registre fait 32 bits. Il faut donc 2 registres pour configurer l’ensemble des broches du même port d’où l’existence de 2 registres : CRL et CRH.

CRL (L pour low) permet de configurer les broches de 0 à 7. CRH (H pour high) de 8 à 15.

Exemple de stm :

