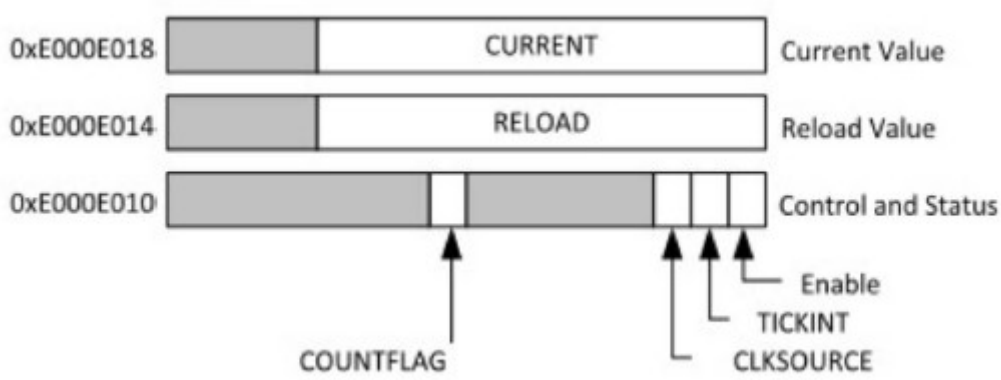
Fondements des systèmes embarqués
TD N°2 : Les microcontroleurs STM32

Exercice 1

Le SysTick est un périphérique du STM32 qui implémente un compteur cyclique programmable. Le périphérique est constitué de trois registres de 32 bits chacun comme le montre la figure suivante :



A chaque cycle d'horloge, le registre CURRENT est décrémenté de 1, lorsque ce registre atteint 0 :

- ✓Le bit COUNTFLAG du registre CONTROL est activé (mis à 1). Ce bit reste à 1 tant qu'on ne l'a pas lu
- ✓Une interruption est envoyée au processeur si le bit TICKINT est activé dans le registre CONTROL (sinon, pas d'envoi d'interruption)
- ✓CURRENT est rechargé par la valeur du registre RELOAD et le cycle de décomptage recommence immédiatement

Le bit ENABLE de CONTROL permet de désactiver/activer le périphérique en entier
Le bit CLKSOURCE permet de sélectionner comme horloge du compteur, soit l'horloge interne du processeur (CLKSOURCE = 1) soit une horloge externe de référence (CLKSOURCE = 0)

1) Définir le type de donnée structure SysTick_Type correspondant au périphérique SysTick
2) Ecrire la fonction C *void SysTick_Configure (SysTick_Type * p, int cycles, int source, int irq)* qui permet de configurer et démarrer le SysTick.
   La configuration revient à :
   - ✓ initialiser CURRENT à 0
   - ✓ initialiser RELAOD à la valeur du paramètre cycles
   - ✓ choisir l'horloge interne du CPU (source =1) ou une horloge externe (source =0)

- ✓ activer ou désactiver la génération d'interruption selon le paramètre irq

Dans ce qui suit on suppose que le SysTick est configuré avec l'horloge interne du processeur

3) Pour attendre le passage du SysTick par 0, un ingénieur écrit le code C suivant :

while (SYSTICK->CURRENT != 0) {}

a) Traduire ce code C en assembleur ARM
b) Calculer le nombre de cycles qui séparent deux lectures consécutives du registre CURRENT dans la boucle while, étant donnés les paramètres du tableau suivant :

Type d'instruction	Nombre de cycle d'horloge
UAL	1
Load/Store	2
Branchement	3

c) En déduire la raison pour laquelle le code ci-dessus n'est pas correct
d) Corriger le code C en question

Dans la suite, on suppose que la fréquence du processeur est F = 8 MHz

4) A combien doit-on initialiser RELOAD pour avoir un cycle (une interruption) chaque 1ms
5) On souhaite appeler deux fonctions *Task1* () et *Task2* () périodiquement chaque 20 et 15 ms respectivement. Pour cela, on utilise l'interruption du SysTick et la routine de gestion d'interruption correspondante appelé *SysTick_Handler ()*
   a) Configurer le périphérique sysTick en appelant *SysTick_Config* dans la fonction main en lui passant les bons paramètres.
   b) Dans la routine de gestion d'interruption *SysTick_Handler*, faites le traitement nécessaire pour implémenter l'appel périodique de *Task1* et *Task2* (remarque : vous pouvez déclarer des variables globales en cas de besoin)

Exercice 2

On considère un périphérique DMA (Direct Memory Access) qui implémente 8 canaux de transfert de données. Chaque canal peut être programmé pour effectuer le transfert d'un bloc de données de taille Z d'une adresse X vers une adresse Y en mémoire. Une fois programmé, le DMA se charge du transfert de données d'une façon autonome (sans intervention du processeur). Les registres qui définissent le périphérique DMA sont les suivants :

- Registre Adresse source (SAR) sur 32 bits (un registre par canal)
- Registre Adresse destination (DAR) sur 32 bits (un registre par canal)
- Registre taille du bloc en octet (DZR) sur 32 bits (un registre par canal)
- Registre configuration (CONFR) (Un seul registre pour les 8 canaux). Chaque canal peut être configuré
  - selon le mode : Circulaire / Non Circulaire
  - selon la taille d'un transfert atomique: 8 bits / 16 bits / 32 bits

Le DMA est mappé en mémoire à l'adresse de base 0xA0008000

1) Quelle est la taille minimale du registre de configuration ?
2) Proposer un arrangement de bits à l'intérieur de ce registre
3) Proposer un mapping mémoire du périphérique DMA. Indiquer clairement en hexadécimal l'adresse de chaque registre

On pose :

```
typedef enum {

DMA_MODE_CIRCULAR = 0,

GPIO_MODE_NON_CIRCULAR = 1

} DMA_MODE;

typeedef enum {

DMA_ATOM_8 = 0,

DMA_ATOM_16 = 1,

DMA_ATOM_32 = 2

} DMA_ATOMIC;
```

4) Écrire la fonction qui permet de configurer un canal particulier (spécifié par le paramètre canal qui doit être entre 0 … 7) :

void ConfigureDMA(uint32_t * base, int canal, uint32_t a_debut, uint32_t a_fin, uint32_t taille, DMA_MODE mode, DMA_ATOMIC atom)

Exercice 3

Le microcontrôleur STM32F4 possède 9 ports GPIO (GPIOA … GPIOI). Un port GPIO contient 16 pins : exemple le port GPIOA = {PA0 … PA15}

Les registres définissant un port GPIO sont :

- Les registres de configuration : chaque pin peut être configuré individuellement selon les paramètres suivants:
  - Mode: {Entrée, Sortie, Fonction Alternative, Analogique}
    - Le registre de configuration correspondant s'appelle MODER
  - Type de sortie: {Pushpull, OpenDrain}
    - Le registre de configuration correspondant s'appelle OTYPER
  - Fréquence de sortie: { Low , Medium , Fast , High }
    - Le registre de configuration correspondant s'appelle OSPEEDR
  - Résistance de rappel: {Rien, PullUp, PullDown}
    - Le registre de configuration correspondant s'appelle PUPDR

- Les registres de données
  - Un registre « Entrée de données » sur 32bits (IDR) accessible en lecture seule
  - Un registre « Sortie de données » sur 32 bits (ODR) accessible en lecture écriture
  - Un registre de Set/Reset de bit sur 32 bits (BSRR) accessible en écriture seule

1. Quelle la taille minimale de chaque registre de configuration
2. On suppose que les registres sont mappés en mémoire dans l'ordre MODER, OTYPER, OSPEEDR, PUPDR, IDR, ODR, BSRR à des adresses alignés sur 4 octets. Donner le plan mémoire de GPIOA
3. Fonction de lecture

Le registre IDR est accessible en lecture. Les valeurs des 16 bit les moins significatifs d'IDR correspondent aux valeurs des pins GPIOx du port concerné si ce port est configuré en lecture : IDR[i] = GPIOx[i], i=0…15

a) Ecrire la fonction GPIO_PortRead qui retourne la valeur du port spécifié par son adresse de base GPIOBASE : uint16_t GPIO_PortRead (unint32_t *GPIOBASE)
b) Ecrire la fonction GPIO_PinRead qui retourne la valeur du pin numéro num du port spécifié par son adresse de base GPIOBASE : uint32_t GPIO_PinRead (unint32_t *GPIOBASE, uint32_t num)

On définit le type suivant:

typedef struct

{ __IO uint32_t MODER;

__IO uint32_t OTYPER;

__IO uint32_t OSPEEDR;

__IO uint32_t PUPDR;

__IO uint32_t IDR;

__IO uint32_t ODR;

__IO uint16_t BSRRL;

__IO uint16_t BSRRH;

} GPIO_TypeDef;

define GPIOA ((GPIO_TypeDef *) GPIOA_BASE)

c) Réécrire la fonction PinRead avec la signature suivante :

uint32_t GPIO_PinRead (GPIO_TypeDef *GPIOx, uint32_t num)

4. Fonction d'écriture

Le registre ODR est accessible en écriture et en lecture. Les valeurs des 16 bits les moins significatifs de ODR se retrouvent sur les pins GPIOx du port concerné si ce port est configuré en sortie : GPIOx[i] = ODR[i], i=0...15

a) Ecrire la fonction GPIO_PortWrite qui écrit une valeur val dans le port GPIOx en appliquant un masque (seuls les bits à 1 dans le masque seront affectés) : void GPIO_PortWrite (GPIO_TypeDef *GPIOx, uint16_t mask, uint16_t val)

b) Supposons que le processeur est interrompu au milieu de cette fonction et qu'une autre fonction vient lire la valeur d'ODR. Pourquoi cette situation peut engendrer une corruption de données ?

5. Écriture atomique

Pour garantir une écriture atomique, on utilise le registre BSRR (Bit Set Reset Register) accessible en écriture seule

Mettre à 1 BSRR[i], i=0...15 engendre la mise à 0 (Reset) de ODR[i]

Mettre à 1 BSRR[16 + i], i=0...15 engendre la mise à 1 (Set) de ODR[i]

Un 0 dans BSRR n'a aucun effet sur ODR

La mise à 1 est plus prioritaire que la mise à 0 au niveau du même bit de ODRR

Ecrire la fonction GPIO_PinWrite qui écrit la valeur val (0 ou 1) dans pin numéro num du port GPIOx : void GPIO_PinWrite (GPIO_TypeDef *GPIOx, uint32_t num, uint32_t val)

6. Fonction de configuration

On définit les types énumérés suivant :

```
typedef enum _GPIO_MODE {
GPIO_MODE_INPUT = 0,
GPIO_MODE_OUTPUT = 1,
GPIO_MODE_AF = 2,
GPIO_MODE_ANALOG = 3
} GPIO_MODE;
```

```
typedef enum _GPIO_OUTPUT_SPEED {
GPIO_OUTPUT_SPEED_2MHz = 0,
GPIO_OUTPUT_SPEED_25MHz = 1,
GPIO_OUTPUT_SPEED_50MHz = 2,
GPIO_OUTPUT_SPEED_100MHz = 3
} GPIO_OUTPUT_SPEED;
```

```
typedef enum _GPIO_OUTPUT_TYPE {
GPIO_OUTPUT_PUSH_PULL = 0,
GPIO_OUTPUT_OPEN_DRAIN = 1
```

```
typedef enum _GPIO_PULL_UP_DOWN {
GPIO_NO_PULL_UP_DOWN = 0,
GPIO_PULL_UP = 1,
```

} GPIO_OUTPUT_TYPE;

GPIO_PULL_DOWN = 2

} GPIO_PULL_UP_DOWN;

Ecrire la fonction de configuration :

```
bool GPIO_PinConfigure(
GPIO_TypeDef *GPIOx,
uint32_t num,
GPIO_MODE mode,
GPIO_OUTPUT_TYPE otype,
GPIO_OUTPUT_SPEED speed,
GPIO_PULL_UP_DOWN pupd
)
```