Systèmes embarqués
Correction TD sur les microcontroleurs STM32

Exercice 1

1)

```
typedef struct{
volatile int CONTROL ;
volatile int RELOAD ;
volatile int CURRENT ;
} SysTick_Type ;
```

2)

```
void SysTick_Configure(SysTick_Type * p, int cycles, int source, int irq){

/*il est recommande de désactiver le périphérique avant de le configurer, ce qui revient a mettre le premier bit de CONTROL a zero. La solution la plus simple est d'écrire zéro dans tout le registre sachant que les autres bits de configuration seront positionnes par la suite */

p→CONTROL = 0 ;
p→CURRENT = 0 ; // mettre CURRENT a zéro
p→RELOAD = cycles ; // initialiser RELOAD a cycles

/* la dernière étape est de forcer les bits Enable, TICKINT et CLKSOURCE du registre CONTROL aux valeurs 1, irq et source respectivement. Comme ces bits représentent la totalité des bits qui peuvent être écrits dans le registre, il est inutile de lire l'ancienne valeur du registre pour conserver d'éventuels autres bits. */
p→CONTROL = 1 | (irq  << 1) | (source << 2 );
/* l'autre solution (plus conservative) serait :
int tmp = p→CONTROL ;
tmp |=1 ;
tmp = tmp & ~(1<<1) | (irq <<1) ;
tmp = tmp & ~(1<<2) | (source<<2) ;
p→CONTROL = tmp ;
*/
}
```

3)

a)

```
LDR R0, =0xE000E018
test
LDR R1, [R0]
CMP R1, #0
BNE test
```

b) 1 LOAD + 1 UAL(CMP) + 1 Branchement = 2 + 1 + 3 = 6 cycles

c) La valeur du registre CURRENT est lue chaque 6 cycles d'horloge par le processeur alors que ce registre est décrémenté automatiquement par le périphérique chaque cycle d'horloge. Il est très probable que CURRENT passe par 0 et revient à sa valeur initiale RELOAD sans que le passage soit détecté par le processeur.

d) Il faut utiliser le bit COUNTFLAG de CONTROL

```
while( ! (SYSTICK→CONTROL & (1<<16) ) ) {}
```

4) 1 cycle d'interruption correspond à (RELOAD + 1) cycles d'horloges

1 ms = (RELOAD +1) * Tclk = (RELOAD+1) / F

RELOAD = 10^{-3} * 8 * 10^{6} – 1 = 8*10^{3} – 1 = 7999

5)

a)

```
int main(){

//on va utiliser un cycle d'interruption de 5ms (PGCD entre 15 et 20)

Systick_Config( (SysTick_Type*)0xE000E010, 39999, 1 , 1) ; }
```

b)

```
int counter = 0 ;
/* SysTick_Handler est une routine de gestion d'interruption (ISR) associée à l'interruption  du Systick. L'adresse de cette fonction a été au préalable écrite à la bonne position dans le vecteur d'interruption du processeur. La routine sera automatiquement exécutée chaque 5ms */
void SysTick_Handler(){
counter ++ ;
if(counter % 3 == 0)
Task2() ;
if(counter % 4 == 0)
Task1() ; }
```

Exercice 2

1) Pour chaque canal on a besoin de configurer le mode sur 1 bit (deux valeurs possibles) et la taille atomique sur 2 bits (3 valeurs possibles) ; donc dans le registre de configuration on a besoin de (1+2)* 8 canaux = 24 bits ce qu'on appelle la taille minimale.
2) La taille réelle de tous les registres ARM est 32 bits. **Une proposition** d'arrangement des bits dans le registre de configuration :

	Canal7			Canal1		Canal0	
8 bits non utilisés	2 bits Taille_atom	1 bit mode	....	2 bits Taille_atom	1 bit mode	2 bits Taille_atom	1 bit mode

3) **Proposition** d'un mapping mémoire du DMA :

Adresse en hexadécimal (du premier octet du registre concerné)	**Registres**
A0008060	DZR7
A000805C	DAR7
A0008058	SAR7
.....	...
A0008024	DZR2
A0008020	DAR2
A000801C	SAR2
A0008018	DZR1
A0008014	DAR1
A0008010	SAR1
A000800C	DZR0
A0008008	DAR0
A0008004	SAR0
A0008000	CONF

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

```
{  uint32_t tmp = *base ;
   Uint32_t *ptr = base ;
 tmp = tmp & ~(1<<3*canal) | (mode <<3*canal) ;
tmp = tmp & ~(3<<3*canal+1) | (atom <<3*canal+1) ;
*base = tmp;
ptr +=3*canal+1;
*ptr=a_debut;
ptr++;
*ptr = a_fin;
```

```
ptr++;
*ptr= taille;}
```

Exercice 3

1) Pour le mode on a besoin de 2 bits (4 valeurs possibles) donc 2*16 pins = 32 bits pour le registre MODER
   Pour le type de sortie on a besoin de 1 bit (2 valeurs possibles) donc 1*16 pins = 16 bits pour le registre OTYPER
   Pour la fréquence on a besoin de 2 bits (4 valeurs possibles) donc 2*16 pins = 32 bits pour le registre OSPEEDR
   Pour la résistance de rappel on a besoin de 2 bits (3 valeurs possibles) donc 2*16 pins = 32 bits pour le registre PUPDR

2) Plan d'adressage du GPIOA :

Adresse du premier octet du registre concerné (les valeurs ici en hexadécimal)	**Registres du GPIOA**
Adresse_base+18	BSRR
Adresse_base+14	ODR
Adresse_base+10	IDR
Adresse_base+C	PUPDR
Adresse_base+8	OSPEEDR
Adresse_base+4	OTYPER
Adresse_base	MODER