Microprocesseurs 1 : Travail écrit no 2.

Nom : **Prénom :** **Classe : I/2** **Date : 13.01.2014**

Problème n° 1 (Processus de développement en C, passage d'arguments et scope (portée) des variables et méthodes)

a. Décrivez succinctement l'organisation/la structure des fichiers (des sources à l'exécutable) en C.

b. Indiquez le scope (portée) des 3 fonctions ci-dessous:

fichier : file1.c

```
extern void fnct1();

void fnct2(int, int){…}

static fnct3(float){…}
```

c. Indiquez la technique utilisée ainsi que le mode d'accès pour les 4 arguments de la fonction ci-dessous.

```
struct S foo (const struct S1 a1, long a2, struct S3* a3, const long* a4);
```

d. Indiquez le scope (portée) des variables v1 à v6 pour l'exemple de code ci-dessous:

```
static struct S1 v1;

bool v2;

extern struct S3* v3;

int bar (long v4) {

   static long v5 = 0;

   int v6 = 1;
   return 0;
}
```

Systèmes Embarqués 1 : Travail écrit no 2.

Problème n° 2 (Interface C)

Décrivez en C l'interface de la bibliothèque « error messages» (header file « err_msg.h ») permettant de gérer des messages d'erreurs. L'interface comprendra une méthode pour initialiser (**`init`**) la bibliothèque, une méthode pour ajouter un nouveau message d'erreurs (**`add_msg`**), une méthode pour lire/obtenir (**`get`**) les messages d'erreurs les uns après les autres et une méthode pour vider (**`flush`**) la liste des messages d'erreurs.

La méthode « **`add_msg`** » permettra de spécifier la sévérité de l'erreur (**`error`**, **`warning`**, **`info`**, **`debug`**) ainsi qu'un message (une chaine de caractères).

La méthode « **`get`** » prendra un paramètre permettant de spécifier si l'on désire obtenir le 1er message d'erreurs ou les suivants et retournera le message d'erreurs au moyen de la structure « **`struct err_msg_message`** » composée des champs suivants :

- Sévérité: énumération
- Timestamp : valeur entière 64-bit non-signée
- Message: chaîne de caractères, maximum 512 caractères.
  La taille maximale d'un message d'erreurs devra être défini par une contante (symbole).
- Attribut permettant de chaîner dynamiquement les messages dans une liste

Fichier : err_msg.h

Systèmes Embarqués 1 : Travail écrit no 2.

Problème n° 3 (Programmation C)

Implémentez dans les règles de l'art deux fonctions de la bibliothèque standard C afin de satisfaire aux spécifications ci-dessous :

```
/* Description
     This function copies up to n bytes from the memory region pointed to by in to the memory region pointed
     to by out. If a byte matching the endchar is encountered, the byte is copied and copying stops. If the
     regions overlap, the behavior is undefined.
   Returns
     memccpy returns a pointer to the first byte following the endchar in the out region. If no byte matching
     endchar was copied, then NULL is returned.
*/
void* memccpy(void *out, const void *in, int endchar, size_t n) {
```

```
/* Description
     strupr converts each character in the string at a to uppercase.
   Returns
     strupr returns its argument, a.
*/
char *strupr(char *a) {
```

Systèmes Embarqués 1 : Travail écrit no 2.

Problème n° 4 (Pilote de périphérique)

Le processeur i.MX27 de Freescale dispose de 6 timers. La figure ci-contre décrit sommairement les registres du contrôleur. Pour rappel, l'i.MX27 travaille en « little endian » et autorise des accès 8, 16 et 32 bits.

Name		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x1000_3000 (TCTL1)– 0x1000_F000 (TCTL6)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	0	0	CC	OM	FRR	CAP		CAP TEN	COMPEN	CLK SOURCE			TEN
	W	SWR															
0x1000_3004 (TPRER1)– 0x1000_F004 (TPRER6)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	0	0	PRESCALER										
	W																
0x1000_3008 (TCMP1)– 0x1000_F008 (TCMP6)	R	COMPARE VALUE															
	W																
	R	COMPARE VALUE															
	W																
0x1000_300C (TCR1)– 0x1000_F00C (TCR6)	R	CAPTURE VALUE															
	W																
	R	CAPTURE VALUE															
	W																
0x1000_3010 (TCN1)– 0x1000_F010 (TCN6)	R	COUNTER VALUE															
	W																
	R	COUNTER VALUE															
	W																
0x1000_3014 (TSTAT1)– 0x1000_F014 (TSTAT6)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CAPT	COMP
	W															W1C	W1C

a. Définissez l'interface C (structure, constantes, ...) pour le contrôleur ci-contre permettant l'implémentation d'un pilote de périphérique en C.
Remarque : seuls les bits utiles au code doivent être déclarés.

b. Déclarez la variable permettant d'accéder aux 3 premiers timers situés aux adresses 0x1000'3000, 0x1000'4000 et 0x1000'5000

c. Pour le 1er timer, écrivez le code permettant de:

i. Initialiser le contrôleur (mettre le bit SWR à 1 et TEN à 0), attendre que l'initialisation soit terminée (SWR à 0) et finalement mettre les bits CC et TEN à 1

ii. Poser le PRESCALER à 33

iii. Configurer le champ CLK_SOURCE à 4

Systèmes Embarqués 1 : Travail écrit no 2.

Problème n° 5 (Pointeurs et pointeurs de fonctions)

Définissez la structure « **`struct fnct`** » et le type « pointeur de fonction » pour les 3 operéations ci-dessous permettant de construire les variables « `fnct1` » à « `fnct6` » et « `fnct_list` ».

```
static long f1 (struct fnct* f, int i, int j) {return (f->v*i) + (j%5); }
static long f2 (struct fnct* f, int i, int j) {return (f->v/i) + (j<<1);}
static long f3 (struct fnct* f, int i, int j) {return (f->v-i) + (j*j); }
```

```
static struct fnct fnct1 = {1, "function1", f1, &fnct2};
static struct fnct fnct2 = {2, "function2", f2, 0};
static struct fnct fnct3 = {3, "function3", f3, &fnct4};
static struct fnct fnct4 = {4, "function3", f3, &fnct6};
static struct fnct fnct5 = {5, "function2", f2, &fnct1};
static struct fnct fnct6 = {6, "function1", f1, &fnct5};
static struct fnct* fnct_list = &fnct3;
```

Pour le code ci-dessous et pour chaque itération, indiquez la fonction appelée (f1, f2 ou f3) avec la valeur des arguments « **`i`** » et « **`j`** », ainsi que la valeur retournée et stockée dans le tableau « **`result`** ».

```
static long result[6] = {1,2,3,4,5,6};
int main () {
  struct fnct* f = fnct_list;
  int i = 0;
  while (f != 0) {
    int j = result[f->v%6];
    result[i] = f->fnct (f, i, j);
    f = f->next; i++;
  }
  return 0;
}
```

Iteration 0 :	Iteration 1 :	Iteration 2 :	Iteration 3 :	Iteration 4 :	Iteration 5 :
result[0] :	result[1] :	result [2] :	result [3] :	result [4]	result [5] :