Systèmes Embarqués 1 : Travail écrit no 2. 5,6 / 5,2

Nom : Zamban **Prénom :** Yannick **Classe :** T-2/I-2 **Date :** 18.01.2018

Problème n° 1 (10 points) (Travaux pratiques, passage d'arguments et scope (portée) des variables et méthodes)

a. Citez les deux techniques utilisées pour le passage d'arguments à des fonctions. Illustrez-les avec une fonction de 4 paramètres. Pour chaque technique, on donnera un paramètre en lecture seule (RO) et un en lecture/écriture (RW).

- passage par valeur : `void f(int a, const int b);` (RW, RO)

- passage par valeur : `void g(int* a, const int* b);` (RW, RO)

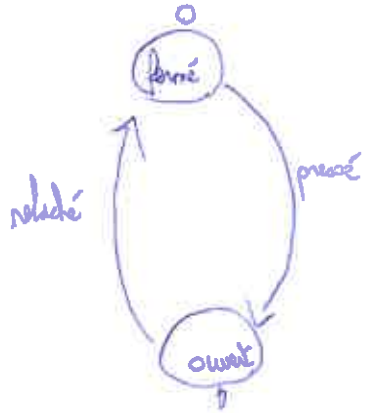
b. Ecrivez un code utilisant une variable globale au module de compilation, une variable locale à une méthode et une variable rémanente. Pour chacune de ces variables, indiquez la valeur initiale si le programmeur oublie de lui en assigner une.

```
extern int a = 0          // globale au programme : par défaut : 0
static void f(int x){
    static int y = 0;     // rémanente           : par défaut : 0
    int z = 0;            // locale à la méthode : par défaut : 0
}
```

Systèmes Embarqués 1 : Travail écrit no 2.

c. La carte d'extension du Beaglebone utilisée durant vos travaux pratiques est équipée de 3 boutons poussoir.
   a) Dessinez la machine d'état du bouton poussoir
   b) Donnez la table de vérité permettant de détecter les états stables et transitoires (ouvert, fermé, pressé et relâché)

a)



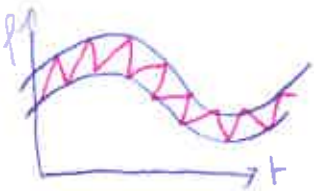
3

b)

	0	1
0	fermé	relâché
1	pressé	ouvert

d. Le µP AM3358 de TI dispose de plusieurs contrôleurs « PWM» (Pulse-Width Modulation).
   a) Décrivez, à l'aide d'une figure, le principe de fonctionnement et de génération d'un signal PWM
   b) Citez les 3 composants principaux du contrôleur PWM du µP AM3358 de TI

a)



Le résultat est une fonction sinusoïdale

µP doit

b) ePWM, timer,

Systèmes Embarqués 1 : Travail écrit no 2.

Problème n° 2 (10 points) (Interface C)

Implémentez en C l'interface de la bibliothèque « `am335x_spi.h` » permettant l'accès aux méthodes de son implémentation ci-dessous et adaptez le code C en conséquence.
L'interface doit être implémentée dans les règles de l'art et permettre son utilisation sans préconditions.

```c
/**
 * Copyright ...
 */

#include <stdint.h>
#include "am335x_clock.h"
#include "am335x_mux.h"

enum am335x_spi_controllers {AM335X_SPI0,  AM335X_SPI1};
enum am335x_spi_channels {AM335X_CHAN0, AM335X_CHAN1};


struct am335x_spi_channel {
        uint32_t chconf;        // 12c + (n * 0x14)
        uint32_t chstat;        // 130 + (n * 0x14)
        uint32_t chctrl;        // 134 + (n * 0x14)
        uint32_t tx;            // 138 + (n * 0x14)
        uint32_t rx;            // 13c + (n * 0x14)
};


#define CHSTAT_RXFFF    (1<<6)
#define CHSTAT_RXFFE    (1<<5)
#define CHSTAT_TXFFF    (1<<4)
#define CHSTAT_TXFFE    (1<<3)
#define CHSTAT_EOT      (1<<2)
#define CHSTAT_TXS      (1<<1)
#define CHSTAT_RXS      (1<<0)

#define CHCTRL_EN       (1<<0)


struct am335x_spi_ctrl {
        uint32_t revision;      // 000
        uint32_t res1[67];      // 004-10c
        uint32_t sysconfig;     // 110
        uint32_t sysstatus;     // 114
        uint32_t irqstatus;     // 118
        uint32_t irqenable;     // 11c
        uint32_t res2[1];       // 120
        uint32_t syst;          // 124
        uint32_t modulctrl;     // 128
        struct am335x_spi_channel channel[4];
        uint32_t xferlevel;     // 17c
};


#define SYSCONFIG_CLKACTIVITY_MASK      (0x3<<8)
#define SYSCONFIG_CLKACTIVITY_NONE      (0x0<<8)
#define SYSCONFIG_CLKACTIVITY_OCP       (0x1<<8)
#define SYSCONFIG_CLKACTIVITY_FUNC      (0x2<<8)
#define SYSCONFIG_CLKACTIVITY_BOTH      (0x3<<8)

#define SYSCONFIG_SIDLEMODE_MASK        (0x3<<3)
#define SYSCONFIG_SIDLEMODE_FORCEIDLE   (0x0<<3)
#define SYSCONFIG_SIDLEMODE_NOIDLE      (0x1<<3)
#define SYSCONFIG_SIDLEMODE_SMARTIDLE   (0x2<<3)
#define SYSCONFIG_SIDLEMODE_RESERVED    (0x3<<3)

#define SYSCONFIG_SRST                  (1<<1)
#define SYSCONFIG_AUTOIDLE              (1<<0)

#define SYSSTATUS_RDONE                 (1<<0)

#define SYSTEM_CLOCK                    48000000


static volatile struct am335x_spi_ctrl* spi_ctrl[] = {
        (struct am335x_spi_ctrl*)0x48030000,
        (struct am335x_spi_ctrl*)0x481A0000,
};

static const enum am335x_clock_spi_modules spi2clock[] = {
        AM335X_CLOCK_SPI0,
        AM335X_CLOCK_SPI1,
};

static const enum am335x_mux_spi_modules spi2mux[] = {
        AM335X_MUX_SPI0,
        AM335X_MUX_SPI1,
};


void am335x_spi_init(
        enum am335x_spi_controllers ctrl,
        enum am335x_spi_channels channel,
        uint32_t bus_speed,
        uint32_t word_len)
{
        volatile struct am335x_spi_ctrl* spi = spi_ctrl[ctrl];

```

```
92          am335x_clock_enable_spi_module (spi2clock[ctrl]);
93
94          am335x_mux_setup_spi_pins (spi2mux[ctrl]);
95
96          spi->sysconfig = SYSCONFIG_SRST;
97          while((spi->sysstatus & SYSSTATUS_RDONE) == 0);
98
99          spi->sysconfig = SYSCONFIG_SIDLEMODE_NOIDLE | SYSCONFIG_CLKACTIVITY_BOTH;
100
101         volatile struct am335x_spi_channel* chan = &spi->channel[channel];
102
103         uint32_t clkg = 0;
104         uint32_t clkd = 0;
105         uint32_t extclk = 0;
106         uint32_t ratio = SYSTEM_CLOCK / bus_speed;
107
108         if ((ratio & (ratio - 1)) != 0) {
109                 clkg = 1;
110                 clkd = (ratio - 1) & 0xf;
111                 extclk = (ratio - 1) >> 4;
112         } else {
113                 while (ratio != 1) {
114                         ratio /= 2;
115                         clkd++;
116                 }
117         }
118
119         chan->chconf =  (clkg<<29) | (2<<25) | (0<<21) | (0<<20) | (0<<18) | (1<<16)
120                       | (0<<14) | (0<<12) | ((word_len-1)<<7) | (1<<6)
121                       | (clkd<<2) | (0<<1) | (0<<0);
122         chan->chctrl = (extclk << 8);
123         spi->modulctrl =  (0<<8) | (0<<7) | (1<<4) | (0<<3) | (0<<2) | (0<<1) | (1<<0);
124 }
125
126 int am335x_spi_read(
127         enum am335x_spi_controllers ctrl,
128         enum am335x_spi_channels channel,
129         uint32_t cmd_word,
130         uint32_t* buffer,
131         uint32_t buffer_len)
132 {
133         (void)cmd_word;(void)buffer;(void)buffer_len;
134         volatile struct am335x_spi_ctrl* spi = spi_ctrl[ctrl];
135         volatile struct am335x_spi_channel* chan = &spi->channel[channel];
136         (void)chan;
137
138         return 0;
139 }
140
141 int am335x_spi_write(
142         enum am335x_spi_controllers ctrl,
143         enum am335x_spi_channels channel,
144         const uint32_t* buffer,
145         uint32_t buffer_len)
146 {
147         volatile struct am335x_spi_ctrl* spi = spi_ctrl[ctrl];
148         volatile struct am335x_spi_channel* chan = &spi->channel[channel];
149
150         spi->irqstatus = -1;
151
152         spi->xferlevel = (buffer_len << 16);
153         chan->chconf |= (2<<12) | (1<<27) | (1<<20);
154         chan->chctrl |= CHCTRL_EN;
155
156         while (buffer_len > 0) {
157                 while ((chan->chstat & CHSTAT_TXFFF) != 0);
158                 chan->tx = *buffer++;
159                 buffer_len--;
160         }
161         while ((spi->irqstatus & (1<<17)) == 0);
162         while ((chan->chstat & CHSTAT_EOT) == 0);
163         chan->chctrl &= ~CHCTRL_EN;
164         chan->chconf &= ~((2<<12) | (1<<27) | (1<<20));
165         spi->xferlevel = 0;
166
167         return 0;
168 }
169
170
```

Systèmes Embarqués 1 : Travail écrit no 2.

Fichier : am335x_spi.h

```
#pragma once
#ifndef _SPI_H_
#define _SPI_H_

#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>

enum am335x_spi_controllers {AM335X_SPI0, AM335X_SPI1};
enum am335x_spi_channels {AM335X_CHAN0, AM335X_CHAN1};
extern void am335x_spi_init(enum am335x_spi_controllers ctrl,
                            enum am335x_spi_channels channel,
                            uint32_t bus_speed,
                            uint32_t word_len);

extern int am335x_spi_read(enum am335x_spi_controllers ctrl,
                           enum am335x_spi_channels channel,
                           uint32_t cmd_word,
                           uint32_t* buffer,
                           uint32_t buffer_len);

extern int am335x_spi_write(enum am335x_spi_controllers ctrl,
                            enum am335x_spi_channels channel,
                            const uint32_t * buffer,
                            uint32_t buffer_len);

// structure dans le .c
#endif
```

Systèmes Embarqués 1 : Travail écrit no 2.

Problème n° 3 (10 points) (Programmation C)

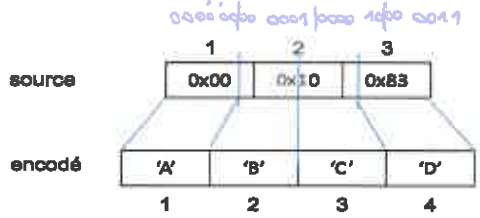
a) Implémentez en C, dans les règles de l'art, la fonction **base64** ci-dessous.

Description selon Wikipedia :

base64 est un codage de l'information utilisant 64 caractères, choisis pour être disponibles sur la majorité des systèmes.

Le processus de codage consiste à coder chaque groupe de 24 bits successifs de données par une chaîne de 4 caractères. On procède de gauche à droite, en concaténant 3 octets pour créer un seul groupement de 24 bits (8 bits par octet). Ils sont alors séparés en 4 nombres de seulement 6 bits (qui en binaire ne permettent que 64 combinaisons). Chacune des 4 valeurs est enfin représentée (codée) par un caractère de l'alphabet suivant :

"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"




```
/** base64 :
   @param dst  tampon (buffer) recevant les 4 octets convertis en base64
   @param src  les 3 octets à convertir
   @return     adresse de la prochaine case libre dans le tampon (dst+4)
*/
char* base64 (char* dst, const uint8_t src[3]) {
```

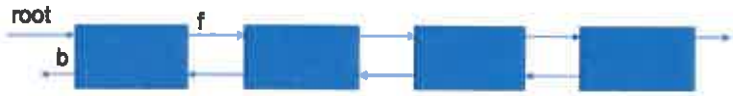
```
    char temp[64] = "ABCDEFGHIJ ... 6789+/";
    uint8_t firstChar = (src[0] & 1111 1100) >> 2;
    *dst = temp[firstChar];
    dst++;
    uint8_t secondChar = (src[0] & 0000 0011) << 4 + ((src[1] & 1111 0000) >> 4);
    *dst = temp[secondChar];
    dst++;
    uint8_t thirdChar = (src[1] & 0000 1111) << 2 + ((src[2] & 1100 0000) >> 6);
    *dst = temp[thirdChar];
    dst++;
    uint8_t fourthChar = (src[2] & 0011 1111);
    dst = temp[fourthChar];
    return dst;
}
```

Systèmes Embarqués 1 : Travail écrit no 2.

b) Implémentez la méthode «`void del_ele (struct element* *root, int key);`» permettant de supprimer le premier élément dont la clef est `key` de la liste doublement chaînée et de le détruire (rendre l'espace mémoire).

Pour rappel, les fins de listes sont indiquées par un pointeur NULL (0).

```
struct element {
        struct element* f;
        struct element* b;
        int key;
        // further attributes...
};
```




```
void del_ele (struct element* *root, int key) {
    if (root == NULL) { return; }
    if (root -> key == key) {
        root -> f -> b = root -> b;
        root -> b -> f = root -> f;   // comble la chaine, ôte virtuellement root;
        free (root);  return;
    }
    else {  root = root -> f;
            del_ele (root, key);  // on avance }

    return;
}
```

Systèmes Embarqués 1 : Travail écrit no 2.

Problème n° 4 (10 points) (Pilote de périphérique)

On souhaite réaliser un pilote pour un périphérique « timer watchdog (WDOG) » connecté à un µP ARM. La figure ci-dessous décrit sommairement les registres du contrôleur disponible sur le µP. Ce contrôleur est conçu pour une machine « little endian » et autorisent des accès 8, 16 et 32 bits sur chacun des registres.

Table 34-8. WDOG Register Summary

Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x1000_2000 (WCR)	R	WT								0	WOE	WDA	SRS	WRE	WDE	WDBG	WDZST
	W																
0x1000_2002 (WSR)	R	WSR															
	W																
0x1000_2004 (WRSR)	R	0	0	0	0	0	0	0	0	0	0	JRST	PWR	EXT	CMON	TOUT	SFTW
	W																

a. Définissez l'interface C (structure, constantes, etc.) pour le contrôleur ci-dessus permettant l'implémentation d'un pilote de périphérique en C.
Remarque : seuls les bits SRS et WOE du registre WCR et EXT du registre WRSR doivent être déclarés.

b. Déclarez la variable permettant d'accéder les registres du contrôleur WDOG situés à l'adresse 0x4001'2000

c. Initialisez le contrôleur en mettant le bit SRS à 1

d. Attendez que l'initialisation soit terminée (SRS à 0)

e. Configurez le champ WT à 10

f. Mettez à 0 le bit WOE

a)
```
#include <stdio.h>
#include <stdlib.h>

#define SRS (1<<4)
#define WOE (1<<6)
#define EXT (1<<3)

struct WDOG { uint16_t WCR; uint16_t WSR; uint16_t WRSR; };
```

b)
```
static volatile struct WDOG* timer = (struct WDOG*) 0x40012000;
```

c)
```
timer->WCR |= SRS;
```

d)
```
while ((timer->WCR & SRS) != 0);
```

e)
```
timer->WCR |= 0000 1010 0000 0000;
```

f)
```
timer->WRSR &= ~WOE;
```

Systèmes Embarqués 1 : Travail écrit no 2.

Problème n° 5 (10 points) (Pointeurs et pointeurs de fonctions)

a) Lors de la conception d'un système d'exploitation (OS), les développeurs souhaitent mettre en oeuvre un système de fichier virtuel (VFS), tel qu'on le connaît sous Linux par exemple. Hormis l'accès à des données stockées sur un disque physique, les VFS permettent de représenter les différents attributs de périphériques d'entrée/sortie sous forme de fichiers.
Les 4 services de base permettant d'échanger des données avec un fichier sont :

```
int open (const char* name, int flags, int mode);
int close(int fd);
int read (int fd, char* buf, int len);
int write(int fd, const char* buf, int len);
```

i. Déclarez les types pointeurs de fonction **`open_t`**, **`close_t`**, **`read_t`** et **`write_t`** pour les 4 services ci-dessus

```
typedef int (*open_t)  (const char*, int, int)
typedef int (*close_t) (int);
typedef int (*read_t)  (int, char*, int)
typedef int (*write_t) (int, const char*, int);
```

ii. Déclarez la structure **`struct file_operations`** permettant de regrouper tous ces opérations

```
struct file_operation { open_t  open;
                        close_t close;
                        read_t  read;
                        write_t write; };
```

iii. Définissez une variable **`sd_file_ops`**, selon le standard c11, pour le pilote d'une carte SD, qui fournit les méthodes **`sd_open`**, **`sd_close`**, **`sd_read`** et **`sd_write`** (ces méthodes sont déjà implémentées par le pilote)

```
struct file_operation sd_file_ops { sd_open, sd_close, sd_read, sd_write };

ou

struct file_operation sd_file_ops { .open = sd_open,
                                    .close = sd_close,
                                    .read = sd_read,
                                    .write = sd_write,
                                  };
```

Systèmes Embarqués 1 : Travail écrit no 2.

b) Complétez l'initialisation du tableau « fsm » afin que le code ci-dessous affiche sur la console le résultat suivant :

```
0) gama: 0
1) iota: 5
2) beta: 9
3) zita: 2
4) kapa: 14
```

Le code :

```
#define print(x) printf("%d) %s: %d\n", i, s->name, x)

typedef struct state {
   const char* name;
   struct state* (*handle)(struct state*, int, int);
   struct state* next;
} state_t;

state_t* s1(state_t* s, int i, int v) {print(v+i); return s->next;}
state_t* s2(state_t* s, int i, int v) {print(v-i); return s->next;}
state_t* s3(state_t* s, int i, int v) {print(v*i); return s->next;}
state_t* s4(state_t* s, int i, int v) {print(v%i); return s->next;}
state_t* s5(state_t* s, int i, int v) {print(v/i); return s->next;}

static struct state fsm[] = {
 [0] = {/* to be completed */
        .name = "kapa",
        .handle = s1,
        .next = 0
 },
 [1] = {/* to be completed */
        .name = "beta",
        .handle = s2,
        .next = &fsm[3],
 },
 [2] = {/* to be completed */
        .name = "gama",
        .handle = s3,
        .next = &fsm[4],
 },
 [3] = {/* to be completed */
        .name = "zita",
        .handle = s4,
        .next = &fsm[0],
 },
 [4] = {/* to be completed */
        .name = "iota",
        .handle = s5,
        .next = &fsm[1],
 },
};

static int v[] = {15,5,11,65,10};

int main() {
  int i = 0;
  state_t* state = &fsm[2];
  while (state != 0) {
     state = state->handle(state, i, v[i]);
     i++;
  }
}
```