Systèmes Embarqués 1 & 2: Travail écrit no 4.

Nom : Eambon **Prénom :** Yannick 4/9 **Classe : T-2/I-2** **Date : 14.06.2018**

3

Problème n° 1 (Entrées/Sorties)

Une équipe de développement logiciel dispose d'un pilote pour un contrôleur de port série. Ce pilote permet l'émission et la réception de caractères 8-bits. Par souci de simplification, il a été conçu en mode par scrutation (voir code ci-dessous). Pour améliorer la réactivité du système, l'équipe souhaiterait maintenant l'adapter pour travailler par interruption.

```
struct uart_ctrl {
        uint32_t data;          // 00 : data register
        uint32_t ier;           // 04 : interrupt enable register
        uint32_t lsr;           // 08 : line status register
};

// IER register definition
#define IER_RX_IT_ENABLED      (1<<0)       // 1 to enable the RX interrupts
#define IER_TX_IT_ENABLED      (1<<1)       // 1 to enable the TX interrupts

// LSR register definition
#define LSR_RX_IT_PENDING      (1<<0)       // if 1 then at least 1 character
                                            // has been received
#define LSR_TX_IT_PENDING      (1<<1)       // if 1 then character could be
                                            // transmitted
#define LSR_RX_FIFO_EMPTY      (1<<2)       // if 1 then rx fifo is empty
#define LSR_TX_FIFO_FULL       (1<<3)       // if 1 then tx fifo is full

static volatile struct uart_ctrl* uart = (struct uart_ctrl*)0x44e09000;

int uart_read() {
        while ((uart->lsr & LSR_RX_FIFO_EMPTY) == 0);
        return uart->data;
}

void uart_write(int c) {
        while ((uart->lsr & LSR_TX_FIFO_FULL) != 0);
        uart->data = c;
}
```

a) Le contrôleur du port série a été configuré pour des données de 8 bits avec 1 bit de start et 1 bit de stop, le tout sans parité. Le débit de la ligne a été fixé à 100kbps. Calculez la taille minimale du buffer de transmission afin que l'application ne doive pas attendre pour l'émission de 2 trames de 50 caractères, puis 1 trame de 10 caractères et 1 trame de 70 et ainsi que 1 trame 20 caractères ceci avec une période de 20ms.

100 kb/s → 100 bit/ms 1 caractère = 10 bits.

2 trames de 50 caractères : 500·2 = 1000 bits
1 trame de 10 caractère : 100 bits
1 trame de 70 caractères : 700 bits
1 trame de 20 caractères : 200 bits

1000+100+700+200 = 2000 bits en 20 ms, on a également un débit de possible de 2000bit/20ms.

Donc, à priori, un buffer n'est pas nécessaire, car en temps

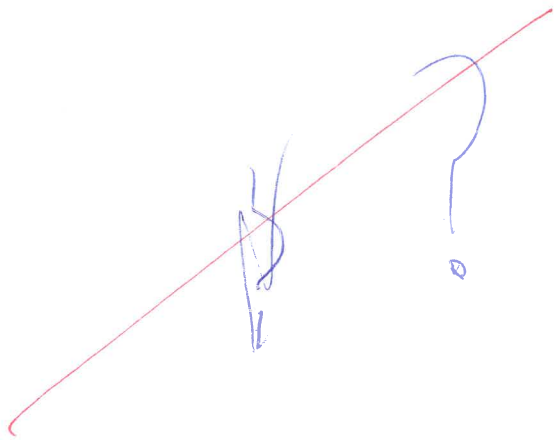
Systèmes Embarqués 1 & 2: Travail écrit no 4.

b) Définissez la structure permettant de stocker les caractères avant leur émission et instanciez-la.
c) Concevez la routine d'interruption pour l'émission de caractères.
d) Adaptez le code ci-dessus afin que l'émission de caractères se fasse par interruption (adapter la méthode « uart_write ».

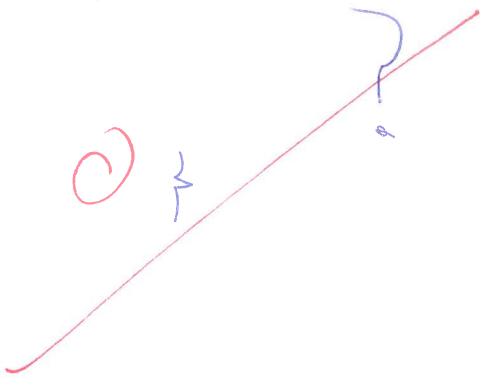
b) struct data_char { int size;
uint8_t character[size];
int current; }

struct data_char data { size = 100;
character[size] = {0};
current = 0; }

c)



d) void uart_write (int c) {



Systèmes Embarqués 1 & 2: Travail écrit no 4.

8

Problème n° 2 (MMU)

1. Citez les 3 fonctions principales (rôles) de la MMU

   - Translation d'adresse physique <—> virtuelle
   - isoler les adresses virtuelles des adresses physique (sécurité)
   - gérer le cache

2

2. Citez le contenu d'une TLB (Translation Lookaside Buffer), décrivez son utilité et son emplacement

   La TLB se situe dans la MMU. C'est une table qui associe une adresse physique et une adresse virtuelle dans chacune de ces lignes. (généralement des entrées très utilisées). Toutes les entrées d'une TLB sont parcourues en même temps. Cela permet de "court-circuiter" la translation usuel pour gagner du temps.

3. Pour effectuer la translation entre adresses virtuelles et adresses physiques, les MMU utilisent des tables de translation. Indiquer comme le processeur TI AM335x implémente ces tables.

   Il l'implémente en une table à 2 niveaux. Les 10 premiers bits permettent de trouver une entrée dans le premier niveau, et, le cas échéant, les 8 bits suivants indiquent l'emplacement de l'adresse dans la table de deuxième niveau.

1

4. Décrivez succinctement comment la MMU trouve l'adresse physique 0x8000'1004 correspondant à l'adresse virtuelle 0x1000'1004, sachant que cette adresse correspond à une page de 1MiB.

   - La MMU lit les 10 premier bits de l'adresse virtuelle et cherche l'entrée correspondante dans la table de premier niveau. L'entrée est marquée comme une page de 1 Mb, il n'y a donc pas de 2e niveau.
   - La MMU récupère l'adresse physique correspondante et y recopie l'offset de l'adresse virtuelle.

1

5. Indiquez comment la MMU procède pour gérer les accès à la mémoire cache

   1 bit indique si les données peuvent être mises dans le cache ou pas

0

6. Décrivez succinctement comment il est possible de protéger l'espace mémoire d'un processus avec une MMU.

   - La MMU génère des interruptions "page fault" si il y a un accès à une entrée de la table de translation non-mappée.
   - Avec ce système, on s'assure qu'une adresse physique n'existe qu'avec son adresse virtuelle correspondante.
   - La MMU isole les mémoire physique et virtuelle.

1/4

Systèmes Embarqués 1 & 2: Travail écrit no 4.

8

Problème n° 3 (Mémoire cache)

1. Décrivez succinctement l'architecture et le principe de fonctionnement d'une mémoire cache complètement associative et indiquez ses avantages et désavantages

Principe: Chaque entrée de la mémoire peut potentiellement remplacer n'importe quelle ligne de la mémoire cache.

2

+ : très rapide.

— : très couteux en silicium.

2. Quelle information est stockée avec les données placées dans la mémoire cache afin que cette dernière puisse les retrouver ? Quelle est l'utilité du bit D.

Les lignes de la mémoire cache possèdent un Tag qui indique les données qui sont stockées dans cette ligne. Une entrée peut être retrouvée avec ce Tag puis avec un offset de "mots" et un offset de byte.

2

Le flag D indique si l'µP a modifié l'entrée (dirty)

3. Deux principes sont à l'origine des mémoires caches, soit la localité temporelle et la localité spatiale. Donnez un exemple de localité temporelle. Si plusieurs variables sont utilisées, indiquez pour chacune d'elle le principe la régissant.

localité temporelle : si utilisé, très probable d'être réutilisé bientôt

localité spatiale : si utilisé, très probable que les voisins soient stockés à côté

2

Exemple temporel : `for (int i=0; i<1000; i++) { j = i+j; }`

register

i et j sont régis par une localité temporelle, elles sont amenées à être "utilisés" de nombreuse fois.

Systèmes Embarqués 1 & 2: Travail écrit no 4.

Pour le code ci-dessous :

a. Indiquez la raison pour laquelle celui-ci n'est pas optimal pour la mémoire cache
b. Proposez une implémentation alternative et argumentez votre choix

```
int a[2000];
int b[2000];
int c[2000];
int d[2000];

void foo() {
   for (int i=0; i<2000; i++) {
       a[i] = b[i] + c[i] * d[i];
   }
}
```

Les variable sont éloignées de & K;B

a) la mémoire cache préfère parcourir des boucles sur "deux" dimensions.
elle favorise également la lecture en ligne.

Or,

il faut faire un tableau de structure pour avoir des éléments contigus

b) ~~void foo() { for (int i=0; i<20~~

```
int a[5000][40];
int b[5000][40];
int c[5000][40];
int d[5000][40];

void foo() {
```

C'est déjà mieux mais pas optimal.

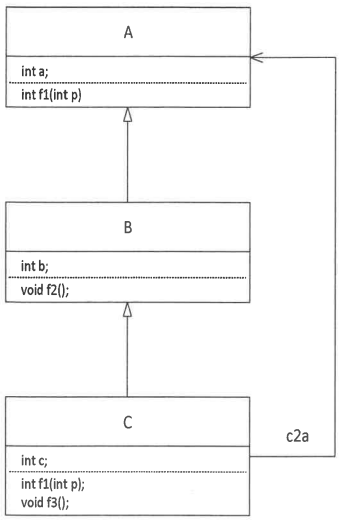
```
for (int i=0; i<5000; i++) {
    for (int j=0; j<10; j++) {
        a[i][j] = b[i][j] + c[i][j] * d[i][j]
```

On lit en ligne + les données sont mieux rangées.

Systèmes Embarqués 1 & 2: Travail écrit no 4.

Problème n° 4 (programmation orientée-objet)
Pour le diagramme de classes ci-contre :

1. Déclarez les classes A, B et C en langage C orienté-objet.
   Remarques : la méthode «`f1`» de la classe C surcharge celle de la classe A. La variable c2a indique une association dirigée de la classe C vers la classe A.
2. Implémentez la fonction «`f1`» de la classe C de manière à ce qu'elle retourne la somme « `p + c + b + a` »
   La macro « `container_of` » est à disposition.
3. Implémentez la méthode d'initialisation de la classe C «`init_c`».
   Remarque : la variable c doit être initialisée à 10;



1)
```
struct A { int a;
           int (*f1)(struct*A*, int); }

struct B { int b;
           void (*f2)(struct*B);
           struct A A_base; }

struct C { int c;
           void (*f3)(struct*C);
           struct B B_base;
           struct*A c2a; }
```

2)
```
void f1(struct*A aref, int p) { struct*C c_ref = container_of(aref, struct C, c2a)
    return p + c_ref->c + B_base.b + c2a->a; }
```

3)
```
void init_c(struct*C aref) { aref->c = 10;
    aref->f1 = f;
    aref->f3 = g;
    aref->B_base = b;
    aref->c2a = init_a(a_ref); }
```
init_b

struct B b, struct*A a_ref

Systèmes Embarqués 1 & 2: Travail écrit no 4.

Problème n° 5 (Toolchain + DMA)

1. Pour le Makefile ci-dessous, expliquez la fonction des lignes 1, 2, 3, 5, 8-9, 13-14, 19 et 21

```
1.  APP=app
2.  SRCS=main.c file1.c file2.c
3.  OBJS=$(SRCS:.c=.o)
4.  CC=gcc
5.  CFLAGS=-g -Wall -Wextra -O1 -std=gnu11 -c -MD
6.  LDFLAGS=-lm -lgcc -lc
7.
8.  .c.o:
9.      $(CC) $(CFLAGS) $< -o $@
10.
11. all: $(APP)
12.
13. $(APP): $(OBJS)
14.     $(LD) $(LDFLAGS) $^ -o $@
15.
16. clean:
17.     rm -f $(APP) $(OBJS)
18.
19. -include $(OBJS:.o=.d)
20.
21. .PHONY: all clean
```

1. définit le nom de l'exécutable

2. définit les sources nécessaires à la compilation du projet

3. Indique que les fichiers de sorties garderont le même nom et changerons leur extension en .o

5. flags (règles) utilisés pour la compilation.

8-9 : règle permettant de compiler les sources vers la cible

13-14 : règle permettant de linker les dépendances.

19 : permet d'inclure les fichiers avec extensions .d (donc main.d, file1.d et file2.d)

21 : indique que les règles "all" et "clean" sont factices et permet leur reconstruction.

Systèmes Embarqués 1 & 2: Travail écrit no 4.

2. Implémentez 3 tests unitaires permettant de valider/vérifier la spécification de la fonction « `strstr()` » de la librairie standard C (selon description ci-dessous).

```
/** locate a substring
 *
 * The strstr() function finds the first occurrence of the substring
 * needle in the string haystack. The terminating null bytes ('\0')
 * are not compared.
 *
 * @return Upon successful completion, strstr() shall return a pointer
 *         to the located string or a null pointer if the string is not
 *         found. If needle points to a string with zero length, the
 *         function shall return haystack.
 */
char *strstr(const char *haystack, const char *needle);
```

```
// zero length
CU_ASSERT( strstr("tristesse", "") == "tristesse" );
// no match
CU_ASSERT( strstr("moyenne", "de classe") == 0 );
// match
CU_ASSERT( strstr("les", "abeilles jaunes") == "les jaunes" );
```

3. Décrivez succinctement la fonction d'un contrôleur DMA dans un système à µP

Le DMA permet d'assurer l'échange de données entre les périphériques et la mémoire sans utiliser le CPU.

4. Citez les 4 phases principales d'un transfert DMA entre un périphérique et la mémoire principale