+1/1/60+

Systèmes Temps-Réel

Examen de Session 1

Durée totale : 1h30

Toute communication (orale, téléphonique, par messagerie, etc.) avec les autres étudiants est interdite. Les notes de cours et de TP sont autorisées. Les pdf du cours sont autorisés (support électronique permis), si ils ont été téléchargés au préalable)

Vous rendrez le sujet complet dans une copie d'examen classique. Vous reporterez votre **NUMÉRO DE COPIE** qui **n'est PAS votre numéro étudiant** sur la première page (ci-dessous).

— Pour la partie QCM, plusieurs réponses peuvent être valides à chaque question, on souhaite avoir **toutes** les réponses valides. Chaque question admet au moins une réponse valide et au moins une réponse incorrecte. Il n'y a aura pas de point négatif.

— Pour les parties rédigées, vous répondrez obligatoirement dans les parties prévues pour, et seulement en cas d'extrême drame sur votre feuille d'examen.

Consignes :

- Utilisez un **stylo à bille noir ou bleu**.
- **Noircir ou bleuir** la/les cases, sans dépasser !
- Pour corriger (dernier recours) : effacez proprement la case.
- Ne pas oublier de noter votre **numéro de copie**.

Numéro de votre copie d'examen :

- Notez-le ici : .....
- Encodez-le ci-contre (chiffre des unités tout à droite).

☐0	☐0	☐0	☐0	☐0	☐0
☐1	☐1	☐1	☐1	☐1	☐1
☐2	☐2	☐2	☐2	☐2	☐2
☐3	☐3	☐3	☐3	☐3	☐3
☐4	☐4	☐4	☐4	☐4	☐4
☐5	☐5	☐5	☐5	☐5	☐5
☐6	☐6	☐6	☐6	☐6	☐6
☐7	☐7	☐7	☐7	☐7	☐7
☐8	☐8	☐8	☐8	☐8	☐8
☐9	☐9	☐9	☐9	☐9	☐9

1 Questions de cours/TP

Question 1 ♣ Un système temps-réel dur est :

- ☐ un système rapide
- ☐ un système prévisible
- ☐ un système où toutes les échéances sont respectées
- ☐ un système "bare-metal"
- ☐ un gestionnaire d'événements

Question 2 ♣ L'ordonnancement des tâches temps réel doit permettre :

- ☐ de permettre une exécution rapide des tâches
- ☐ le respect des contraintes temporelles pour le plus grand nombre de tâches
- ☐ de privilégier les tâches les plus critiques

Question 3 ♣ Les algorithmes d'ordonnancement hors ligne permettent :

- ☐ de construire la séquence d'ordonnancement avant l'exécution
- ☐ de construire la séquence d'ordonnancement après l'exécution
- ☐ de construire la séquence d'ordonnancement pendant l'exécution

Question 4 ♣ Le critère de Liu et Layland :

- ☐ permet de prouver la non ordonnançabilité d'un jeu de tâches avec RM
- ☐ est une condition nécessaire d'ordonnançabilité
- ☐ est une condition suffisante d'ordonnançabilité
- ☐ permet de prouver l'ordonnançabilité d'un jeu de tâches avec RM

Question 5 ♣ L'ordonnancement EDF :

- ☐ privilégie toujours les tâches de plus courte période
- ☐ est un algorithme statique
- ☐ privilégie toujours les tâches de plus proche échéance

Question 6 ♣ La programmation asynchrone :

- ☐ permet d'éviter l'utilisation des verrous
- ☐ n'est utilisable qu'en Python
- ☐ rend tous les appels système non bloquants
- ☐ utilise une thread par fichier en attente
- ☐ fait tourner des tâches (non système) préemptibles

Question 7 ♣ Il est intéressant de lancer une thread pour chaque "traitement" quand :

- ☐ le traitement fait principalement des entrées sorties
- ☐ les traitements sont courts
- ☐ il y a de très nombreux traitements en parallèle
- ☐ on veut assurer l'isolation mémoire des traitements entre-eux

Question 8 ♣ Les plateforme de type arduino sont adaptées à du temps-réel car :

- ☐ leur modèle de programmation est une boucle réactive
- ☐ on peut les programmer sans système d'exploitation
- ☐ les entrées-sorties sont facilement commandables

Question 9 ♣ Pour calculer le pire-temps d'exécution d'un bloc de base pour le temps-réel :

- ☐ on écrit un programme linéaire
- ☐ on considère les caractéristiques architecturales de la plateforme d'exécution
- ☐ on a besoin du nombre de tours de boucles de chaque boucle du programme.
- ☐ on exécute le programme symboliquement

Question 10 ♣ Dans Trampoline, l'ordonnancement des tâches est :

- ☐ Toujours non-préemptif
- ☐ préemptif ou non-préemptif, selon la valeur de SCHEDULE
- ☐ basé sur les priorités
- ☐ Toujours préemptif
- ☐ préemptif ou non-préemptif, selon la valeur de ACTIVATION

Question 11 ♣ Un programme exécutant 10 fois une tâche dont chaque instance prend 0.5 seconde de CPU, avec une période de 1 seconde s'exécutera en :

- ☐ 5.5 secondes
- ☐ 15 secondes
- ☐ 10.5 secondes
- ☐ 5 secondes
- ☐ 10 secondes

2 Ordonnancement

Les notations de cette sections sont les notations classiques du cours : C capacité (wcet), T (ou P) période, D deadline, N nombre de tâches à ordonnancer.

2.1 Exercice 1

On considère l'ordonnancement Rate Monotonic (RM). On rappelle le critère de Liu et Layland :

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \leq N(2^{1/N} - 1)$$

et le jeu de tâches suivant (avec Période = Échéance)

Tâche	*Période*(T)	*Temps d'exécution*(C)
1	50	10
2	40	10
3	30	15

Question 12 Pour ce jeu de tâches, que peut-on dire de l'ordonnancement RM avec le critère de Liu Et Layland ?

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

.....

.....

.....

Question 13 Dessiner un ordonnancement RM correct. ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

2.2 Exercice 2 - RM et EDF

Tâche	*Période*(P)	*Temps d'exécution*(C)	*Deadline*/Échéance(D)
1	9	3	9
2	18	5	18
3	12	4	12

Question 14 En dessinant la trace d'exécution pour les 18 premières périodes d'exécution, montrez que RM ne respecte pas les contraintes, donc ne permet pas un ordonnancement faisable.

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

Question 15 En dessinant la trace d'exécution pour les 36 premières périodes d'exécution, montrez que l'algorithme d'ordonnancement EDF est valide. . . ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

Question 16 Calculer le pourcentage d'utilisation du CPU pour votre solution ci-dessus.

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

. . . . . .

. . . . . .

. . . . . .

3 Arduino

On veut écrire un programme Arduino qui réalise la tâche périodique (de fréquence 100Hz) suivante : lire 3 entrées successives sur `digital0` et écrire (après un délai de 3 périodes) sur `digital1` la valeur de "majorité" ('0' si deux '0', '1' si deux '1' sur les trois), et ce sur une "fenêtre glissante".

```
void main(){
   init();
   while(1) {
     step();
     _delay_ms(xx); // attend xx ms.

   }
}
```

On demande d'utiliser la librairie Arduino de bas niveau (PORT, calcul booléen), dont on fournit une documentation en annexe.

Question 17 Que vaut xx ?

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

.....

Question 18 Écrire la fonction `init`.

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

.....

.....

.....

Question 19 Écrire un pseudo-code pour la fonction `step`.

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

.....

.....

.....

.....

Question 20 Implémenter la fonction `step`. ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

.....

.....

.....

.....

.....

.....

4 Programmation Lustre et Vérification

FIGURE 1 – Modélisation Lustre

Dans cet exercice, on va modéliser des tâches temps-réel par des noeuds LUSTRE auquel on passera en plus un signal venant de l'ordonnanceur (cf Figure 1). L'ordonnanceur, qui est un noeud LUSTRE aussi, générera 3 signaux qui simulent l'ordonnancement par priorité fixe ; en considérant que la tâche 1 est plus prioritaire que la tâche 2, elle même plus prioritaire que la tâche 3. L'ordonnanceur doit ici ordonnancer 3 tâches de période 10 et de wcet 3.

Question 21 Dessiner l'ordonnancement RR à priorité fixe des trois tâches sur une période de 10 ticks. ..... ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

Question 22

Écrire un noeud Lustre qui produit cet ordonnancement "à l'infini" (après la première période de 10, il reproduit le même). Ce noeud , go_1, go_2, go_3, booléens, $go_i(t)$ étant `T` (true) ssi la tâche i a un accès CPU à la date t. Voici un exemple d'exécution pour `ordo()` ! On pourra utiliser une variable auxiliaire qui "décompte" le temps depuis le début d'une période de 10.

go1	*T*	*T*	*T*	*F*	*F*	*F*	*F*	*F*	*F*	*F*
go2	*F*	*F*	*F*	*T*	*T*	*T*	*F*	*F*	*F*	*F*
go3	*F*	*F*	*F*	*F*	*F*	*F*	*T*	*T*	*T*	*F*

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

.....

.....

.....

.....

.....

Question 23

Écrire un observateur qui permet de s'assurer qu'une seule des tâches obtient le CPU à chaque unité de temps. Attention aux paramètres d'entrée et de sortie de ce noeud Lustre.

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

.....

.....

.....

.....

Question 24 Écrire en LUSTRE le code de `mytask` :

```
node mytask (const period, wcet :int; go : bool)
returns ( clockx, nbp : int; awake : bool)
```

qui modélise une tâche de période `period` (entrée constante), de capacité `wcet` (entrée constante), sous l'influence de l'ordonnanceur : `go` est une entrée qui dit que l'ordonnanceur a donné l'accès CPU à cette tâche (comme à la question précédente). Vous définirez les sorties suivantes :

— l'entier `clockx` correspond à la taille de l'intervalle de temps discret qui s'écoule entre deux réveils de la tâche (réveil au sens début de l'intervalle de temps de durée `period`) ;
— l'entier `nbp` correspond à la taille de l'intervalle de temps discret où la tâche a eu accès à la ressource depuis le dernier réveil (au sens précédent) ;
— le booléen `awake` est vrai si la tâche n'a pas fini de s'exécuter dans l'intervalle de temps.

Voici un exemple d'exécution pour `mytask(5,2,go)` :

go	*T*	*T*	*T*	*T*	*T*	*T*	*F*	*F*	*T*	*T*
wake	*T*	*T*	*F*	*F*	*F*	*T*	*T*	*T*	*T*	*F*
clockx	*0*	*1*	*2*	*3*	*4*	*0*	*1*	*2*	*3*	*4*
nbp	*0*	*1*	*2*	*2*	*2*	*0*	*1*	*1*	*1*	*2*

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

5 Calcul de pire temps d'exécution

On considère le programme suivant :

```
1   int a[100],t[100];
2   i=0;
3   while (i<97){
4     b=random(0,1); //0 or 1 value
5     if (b)
6       t[i] = 100;
7     else
8       t[i] = 200+t[i-1];
9     t[i] = t[i] + 42;
10    a[i] = 1515;
11    if (b)
12       a[i] = 100;
13  }
```

On considère qu'un accès tableau en lecture coûte 150, en écriture 200, une opération élémentaire (affectation de variable, test) 100.

Question 25 Dessiner l'AST du programme et calculer une estimation du WCET de ce programme.

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Question 26 Dessiner le graphe de flot du programme (1 ligne de code max par bloc, numéroter les blocs à l'aide des numéros de ligne du code) et écrire le programme linéaire dont la résolution donne une estimation du WCET.

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Question 27 Ajouter une contrainte qui permet de prendre en compte (partiellement) le fait que le test sur b donne toujours le même résultat dans une même boucle. Expliquer.

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

.....

.....

.....

Annexe 1 : Arduino Port Manipulation

Port Registers

Port registers allow for lower-level and faster manipulation of the i/o pins of the microcontroller on an Arduino board. The chips used on the Arduino board (the ATmega8 and ATmega168) have three ports:

- B (digital pin 8 to 13)
- C (analog input pins)
- D (digital pins 0 to 7)

Each port is controlled by three registers, which are also defined variables in the Arduino language. The DDR register, determines whether the pin is an INPUT or OUTPUT. The PORT register controls whether the pin is HIGH or LOW, and the PIN register reads the state of INPUT pins set to input with pinMode(). The maps of the ATmega8 (http://www.arduino.cc/en/Hacking/PinMapping) and ATmega168 (//www.arduino.cc/en/Reference/Atmega168Hardware) chips show the ports. The newer Atmega328p chip follows the pinout of the Atmega168 exactly.

DDR and PORT registers may be both written to, and read. PIN registers correspond to the state of inputs and may only be read.

PORTD maps to Arduino digital pins 0 to 7

DDRD - The Port D Data Direction Register - read/write
PORTD - The Port D Data Register - read/write
PIND - The Port D Input Pins Register - read only

PORTB maps to Arduino digital pins 8 to 13 The two high bits (6 & 7) map to the crystal pins and are not usable

DDRB - The Port B Data Direction Register - read/write
PORTB - The Port B Data Register - read/write
PINB - The Port B Input Pins Register - read only

PORTC maps to Arduino analog pins 0 to 5. Pins 6 & 7 are only accessible on the Arduino Mini

DDRC - The Port C Data Direction Register - read/write
PORTC - The Port C Data Register - read/write
PINC - The Port C Input Pins Register - read only

Each bit of these registers corresponds to a single pin; e.g. the low bit of DDRB, PORTB, and PINB refers to pin PB0 (digital pin 8). For a complete mapping of Arduino pin numbers to ports and bits, see the diagram for your chip: ATmega8 (//www.arduino.cc/en/Hacking/PinMapping), ATmega168 (//www.arduino.cc/en/Hacking/PinMapping168). (Note that some bits of a port may be used for things other than i/o; be careful not to change the values of the register bits corresponding to them.)

Examples

Referring to the pin map above, the PortD registers control Arduino digital pins 0 to 7.

You should note, however, that pins 0 & 1 are used for serial communications for programming and debugging the Arduino, so changing these pins should usually be avoided unless needed for serial input or output functions. Be aware that this can interfere with program download or debugging.

DDRD is the direction register for Port D (Arduino digital pins 0-7). The bits in this register control whether the pins in PORTD are configured as inputs or outputs so, for example:

```
DDRD = B11111110;  // sets Arduino pins 1 to 7 as outputs, pin 0 as input
DDRD = DDRD | B11111100;  // this is safer as it sets pins 2 to 7 as outputs
                          // without changing the value of pins 0 & 1, which are RX & TX
```

```
 //See the bitwise operators reference pages and
```
The Bitmath Tutorial (http://www.arduino.cc/playground/Code/BitMath) in the Playground

PORTD is the register for the state of the outputs. For example;

```
 PORTD = B10101000; // sets digital pins 7,5,3 HIGH
```

You will only see 5 volts on these pins however if the pins have been set as outputs using the DDRD register or with pinMode().

PIND is the input register variable It will read all of the digital input pins at the same time.

Why use port manipulation?

From The Bitmath Tutorial (http://www.arduino.cc/playground/Code/BitMath)

Generally speaking, doing this sort of thing is **not** a good idea. Why not? Here are a few reasons:

- The code is much more difficult for you to debug and maintain, and is a lot harder for other people to understand. It only takes a few microseconds for the processor to execute code, but it might take hours for you to figure out why it isn't working right and fix it! Your time is valuable, right? But the computer's time is very cheap, measured in the cost of the electricity you feed it. Usually it is much better to write code the most obvious way.
- The code is less portable. If you use digitalRead() and digitalWrite(), it is much easier to write code that will run on all of the Atmel microcontrollers, whereas the control and port registers can be different on each kind of microcontroller.
- It is a lot easier to cause unintentional malfunctions with direct port access. Notice how the line DDRD = B11111110; above mentions that it must leave pin 0 as an input pin. Pin 0 is the receive line (RX) on the serial port. It would be very easy to accidentally cause your serial port to stop working by changing pin 0 into an output pin! Now that would be very confusing when you suddenly are unable to receive serial data, wouldn't it?

So you might be saying to yourself, great, why would I ever want to use this stuff then? Here are some of the positive aspects of direct port access:

- You may need to be able to turn pins on and off very quickly, meaning within fractions of a microsecond. If you look at the source code in lib/targets/arduino/wiring.c, you will see that digitalRead() and digitalWrite() are each about a dozen or so lines of code, which get compiled into quite a few machine instructions. Each machine instruction requires one clock cycle at 16MHz, which can add up in time-sensitive applications. Direct port access can do the same job in a lot fewer clock cycles.
- Sometimes you might need to set multiple output pins at exactly the same time. Calling digitalWrite(10,HIGH); followed by digitalWrite(11,HIGH); will cause pin 10 to go HIGH several microseconds before pin 11, which may confuse certain time-sensitive external digital circuits you have hooked up. Alternatively, you could set both pins high at exactly the same moment in time using PORTB |= B1100;
- If you are running low on program memory, you can use these tricks to make your code smaller. It requires a lot fewer bytes of compiled code to simultaneously write a bunch of hardware pins simultaneously via the port registers than it would using a for loop to set each pin separately. In some cases, this might make the difference between your program fitting in flash memory or not!

See

- Pin Mapping of Atmega 168/328 (http://arduino.cc/en/Hacking/Atmega168Hardware)