NOM : **PRENOM :**

DEVOIR SURVEILLE
Systèmes Numériques à Microprocesseurs
(Durée 2h00 Documents interdits, calculatrice simple autorisée)

ATTENTION, vous devez répondre directement sur le sujet
N'oubliez pas d'indiquer vos noms et prénoms sur TOUTES les feuilles
Pensez à écrire lisiblement

Vous allez devoir répondre à 12 questions ou petits exercices **SIMPLES** qui correspondent à l'évaluation des compétences minimales devant être acquises à la fin du module Systèmes à Microprocesseurs. Chaque question compte pour 1 point.

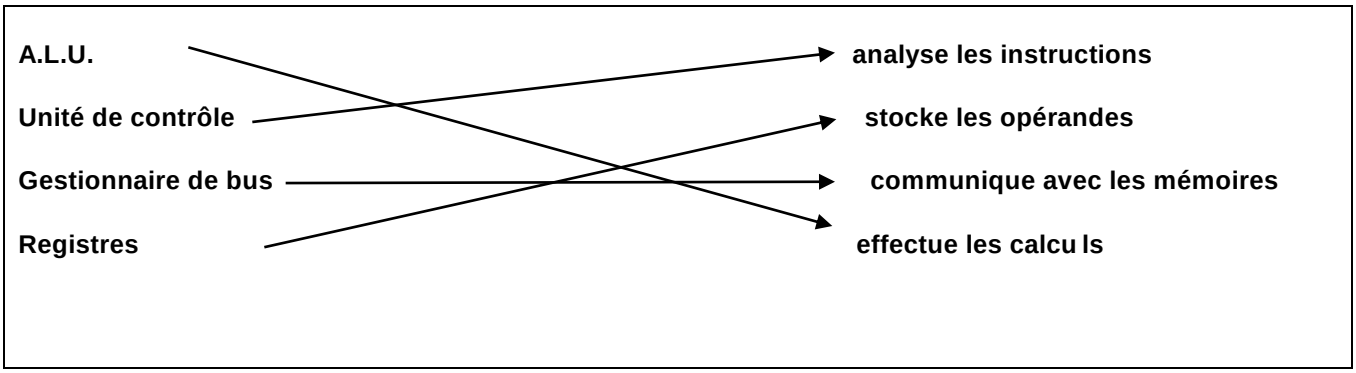
FAITES BIEN ATTENTION EN LISANT LES QUESTIONS

TOUS LES MOTS SONT IMPORTANTS

NOM : **PRENOM :**

PREMIERE PARTIE :

QUESTION 1 :
Associez les 4 éléments de la colonne de gauche avec les quatre affirmations de la colonne de droite



QUESTION 2 :
Si l'on veut alimenter le MSP430G2553 avec une pile, a-t-on intérêt à le faire fonctionner à sa vitesse d'horloge maximale ? Pourquoi ?

A priori non, car la consommation d'un processeur augmente avec sa fréquence de fonctionnement. Donc si l'on veut avoir une autonomie importante, il faut limiter la consommation au maximum.

Conclusion : A moins que l'algorithme de l'application impose une vitesse de fonctionnement élevée, il vaut mieux limiter la fréquence de fonctionnement du MSP430.

NOM : **PRENOM :**

QUESTION 3 :
Il existe dans le debugger de Code Composer Studio une fonctionnalité appelée BREAKPOINT. A quoi sert-elle ?

Le BREAKPOINT (ou point d'arrêt) sert à stopper l'exécution du programme sur une ligne précise de celui-ci.
On l'utilise en débogage pour exécuter des parties de programmes à vitesse réelle, puis vérifier que cette exécution a donné les résultat escomptés.

QUESTION 4 :
Vous avez ci-dessous une fonction d'initialisation du registre P1DIR du MSP430.
En supposant que le registre P1DIR contient **0x76** avant l'exécution, indiquez quelles seront les lignes du port P1 en entrées et celles qui seront en sorties après l'exécution de la fonction. (Il n'y a pas d'erreur de frappe dans la fonction).

```
void init_P1DIR( void )
{
  P1DIR &= ~(BIT5| BIT4 | BIT3 | BIT2);
  P1DIR |= (BIT7 | BIT5 | BIT1 | BIT0);
}
```

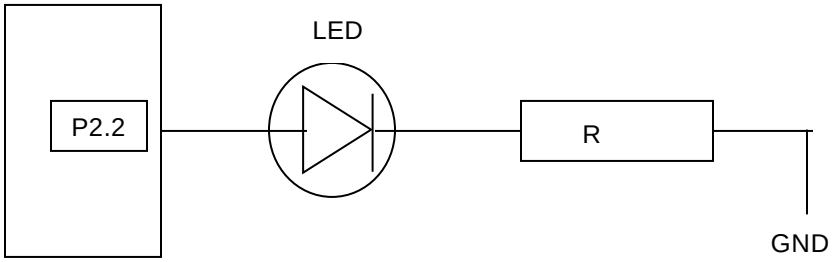
0x76 = 0 1 1 1 0 1 1 0
P1DIR &= ~(BIT5| BIT4 | BIT3 | BIT2); 0 1 **0 0 0 0** 1 0
P1DIR |= (BIT7 | BIT5 | BIT1 | BIT0); **1** 1 **1** 0 0 0 **1 1**

Résultat final 0b 1110 0011

Donc les lignes 7, 6 , 5, 1 et 0 du Port1 sont en sorties
Les lignes 4, 3 et 2 du Port1 sont en entrées.

NOM : **PRENOM :**

QUESTION 5 :
En considérant le schéma électrique suivant :



Ecrire le programme qui allume la led pendant 2 secondes. On suppose l'existence d'une fonction **void tempo(int duree)** dans laquelle durée représente des millisecondes.

```
Void gestion_led ()
{

P2DIR |= BIT2 ;
P2OUT |= BIT2 ;
tempo (2000) ;
P2OUT &=~BIT2 ;
}
```

NOM : **PRENOM :**

QUESTION 6 :
Quelles sont les initialisations nécessaires pour créer un délai de 1000ms avec le Timer du MSP430 ? On n'utilisera pas les interruptions et l'horloge du MSP430 est à 1MHz.

Tsmck = 1µs
Mode UP/DOWN
Prédiv = 8
TACCR0 = 62500

Est la seule solution possible. Tous les autres cas aboutissent à un TACCR0 > 65535

QUESTION 7 :
Quelle sera la durée de comptage du timer initialisé de la manière suivante. **Précisez** votre calcul :

MODE UP/DOWN
TACCR0 : 40000
Horloge SMCLK (2MHz)
Prédiviseur par 2

Durée= Tsmclk x MODE (1ou2) x prédiv (1,2,4ou8) x TACCR0

Durée = 0,5 µs x 2 x 2 x 40000 = 80000µs = 80 ms = 0.08 s

NOM : **PRENOM :**

QUESTION 8 :
Quels sont les rôles des registres P1IE et P1IES. Donnez un exemple d'utilisation.

P1IE sert à autoriser les interruptions sur une ligne Tout ou rien du Port1
P1IES sert à choisir le déclenchement des interruptions sur front montant ou descendant

Exemple : si un capteur de présence renvoie une impulsion positive lorsqu'il détecte un objet et que l'on connecte ce capteur sur la ligne P1.4 du MSP430 ; on pourra déclencher une interruption dès qu'un objet sera vu en plaçant le BIT4 du P1IE à 1 et le BIT4 de P1IES à 0.
Ceci évitera au programme de devoir sans arrêt tester le capteur pour savoir si un objet est présent.

QUESTION 9 :
Le convertisseur analogique-numérique (ADC) du MSP430 travaille sur **10** bits.
On le configure pour qu'il travaille avec des tensions de référence comprises entre **0 et 2.5 V**
Quelle valeur de tension d'entrées permet d'obtenir 0x1A0 comme valeur de sortie ?

10 bits donnent des valeurs entre 0 et 1023
0 correspondra à 0 Volt et 1023 à 2,5 Volts

0x1A0 vaut 256 + 160 = 416

Pour avoir 416 comme résultat de conversion il faut donc (2.5 / 1023) * 416 = 1.016 Volts en entrée

NOM : **PRENOM :**

QUESTION 10 :
Vous avez ci-dessous une fonction utilisant des lignes d'entrées-sorties digitales du MSP430. Décrire l'opération **matérielle** réalisée par cette fonction.

```
void action()
{
  unsigned char a ;
  P2DIR = 0x0F;
  a =P2IN ;
  a = (a >> 4);
  P2OUT = a;
}
```

P2DIR = 0x0F; les lignes 7, 6, 5 et 4 sont en entrées, les lignes 4, 3, 2 et 1 en sorties
a =P2IN ; On lit l'état du port2 on récupère les état des entrées 7,6,5 et 4
a = (a >> 4); On décale de 4 bits à droite la variable a: les bits 7-6-5-4 se retrouvent en position 3-2-1-0
P2OUT = a; On écrit sur le port2, les bits 3-2-1-0 du port prennent les valeurs continues dans a

Au final, on a recopié sur les 4 sorties 3-2-1-0 du Port2 l'état des 4 entrées 7-6-5-4 lues sur ce même port.

NOM : **PRENOM :**

QUESTION 11 :

Peut-on concevoir un système à base de microprocesseur sans mémoire ROM ? Pourquoi ?

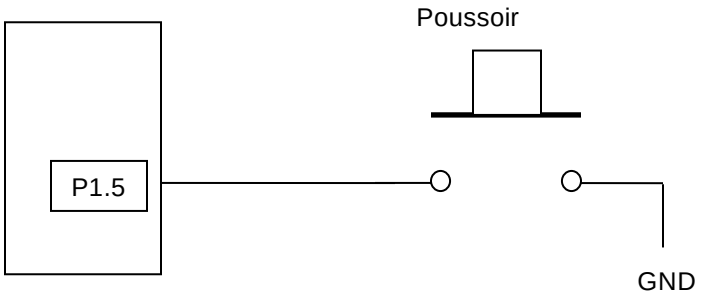
La mémoire ROM est une mémoire non volatile qui ne perd pas son contenu hors tension.

Lorsqu'un système à Microprocesseurs est mis sous tension, il faut qu'il puisse trouver le programme de démarrage. Il faut donc que celui-ci soit stocké dans une mémoire qui ne s'efface pas hors alimentation.

Donc il est nécessaire qu'il y ait une mémoire de type ROM sur un système à microprocesseur pour que celui-ci puisse démarrer lors de la mise sous tension.

NOM : **PRENOM :**

QUESTION 12 :
Ecrire le programme d'initialisation ci-dessous pour qu'il soit capable de gérer les interruptions sur appui du bouton poussoir ?




```
void Init_poussoir_IT()
{
P1DIR &=~BIT5 ;
P1REN |= BIT5 ;
P1OUT |= BIT5 ;
P1IE |= BIT5 ;
P1IES &=~BIT5 ;

}
```

NOM : **PRENOM :**

DOCUMENTATIONS TECHNIQUES

8.2 Digital I/O Operation

The digital I/O is configured with user software. The setup and operation of the digital I/O is discussed in the following sections.

8.2.1 Input Register PxIN

Each bit in each PxIN register reflects the value of the input signal at the corresponding I/O pin when the pin is configured as I/O function.

Bit = 0: The input is low

Bit = 1: The input is high

Note: Writing to Read-Only Registers PxIN

Writing to these read-only registers results in increased current consumption while the write attempt is active.

8.2.2 Output Registers PxOUT

Each bit in each PxOUT register is the value to be output on the corresponding I/O pin when the pin is configured as I/O function, output direction, and the pull-up/down resistor is disabled.

Bit = 0: The output is low

Bit = 1: The output is high

If the pin's pull-up/down resistor is enabled, the corresponding bit in the PxOUT register selects pull-up or pull-down.

Bit = 0: The pin is pulled down

Bit = 1: The pin is pulled up

8.2.3 Direction Registers PxDIR

Each bit in each PxDIR register selects the direction of the corresponding I/O pin, regardless of the selected function for the pin. PxDIR bits for I/O pins that are selected for other functions must be set as required by the other function.

Bit = 0: The port pin is switched to input direction

Bit = 1: The port pin is switched to output direction

8.2.4 Pullup/Pulldown Resistor Enable Registers PxREN

Each bit in each PxREN register enables or disables the pullup/pulldown resistor of the corresponding I/O pin. The corresponding bit in the PxOUT register selects if the pin is pulled up or pulled down.

Bit = 0: Pullup/pulldown resistor disabled

Bit = 1: Pullup/pulldown resistor enabled

NOM : **PRENOM :**

Interrupt Edge Select Registers P1IES, P2IES

Each PxIES bit selects the interrupt edge for the corresponding I/O pin.

Bit = 0: The PxIFGx flag is set with a low-to-high transition

Bit = 1: The PxIFGx flag is set with a high-to-low transition

Note: Writing to PxIESx

Writing to P1IES, or P2IES can result in setting the corresponding interrupt flags.

PxIESx	PxINx	PxIFGx
0 → 1	0	May be set
0 → 1	1	Unchanged
1 → 0	0	Unchanged
1 → 0	1	May be set

Interrupt Enable P1IE, P2IE

Each PxIE bit enables the associated PxIFG interrupt flag.

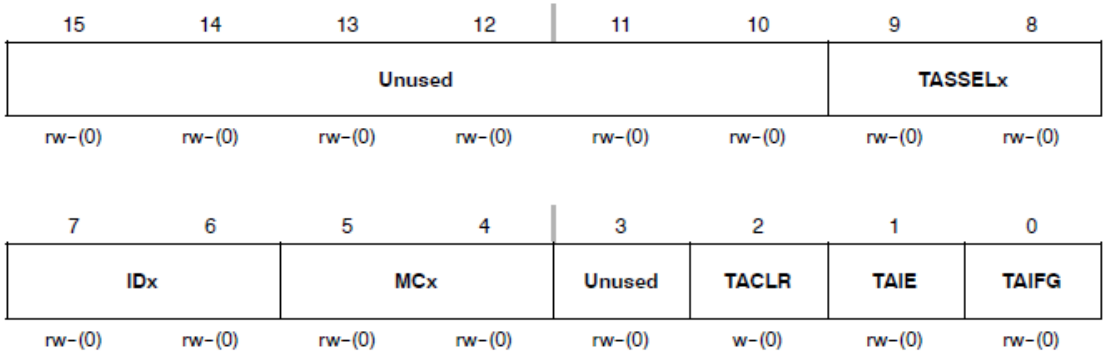
Bit = 0: The interrupt is disabled.

Bit = 1: The interrupt is enabled.

NOM : **PRENOM :**

Timer_A Registers

TACTL, Timer_A Control Register



Unused Bits 15-10 Unused

TASSELx Bits 9-8 Timer_A clock source select
- 00 TACLK
- 01 ACLK
- 10 SMCLK
- 11 INCLK (INCLK is device-specific and is often assigned to the inverted TBCLK) (see the device-specific data sheet)

IDx Bits 7-6 Input divider. These bits select the divider for the input clock.
- 00 /1
- 01 /2
- 10 /4
- 11 /8

MCx Bits 5-4 Mode control. Setting MCx = 00h when Timer_A is not in use conserves power.
- 00 Stop mode: the timer is halted.
- 01 Up mode: the timer counts up to TACCR0.
- 10 Continuous mode: the timer counts up to 0FFFFh.
- 11 Up/down mode: the timer counts up to TACCR0 then down to 0000h.

Unused Bit 3 Unused

TACLR Bit 2 Timer_A clear. Setting this bit resets TAR, the clock divider, and the count direction. The TACLR bit is automatically reset and is always read as zero.

TAIE Bit 1 Timer_A interrupt enable. This bit enables the TAIFG interrupt request.
- 0 Interrupt disabled
- 1 Interrupt enabled

TAIFG Bit 0 Timer_A interrupt flag
- 0 No interrupt pending
- 1 Interrupt pending

NOM : **PRENOM :**

TACCRx, Timer_A Capture/Compare Register x

15	14	13	12	11	10	9	8
TACCRx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
TACCRx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

TACCRx Bits 15-0 Timer_A capture/compare register.
Compare mode: TACCRx holds the data for the comparison to the timer value in the Timer_A Register, TAR.
Capture mode: The Timer_A Register, TAR, is copied into the TACCRx register when a capture is performed.

TACCTLx, Capture/Compare Control Register

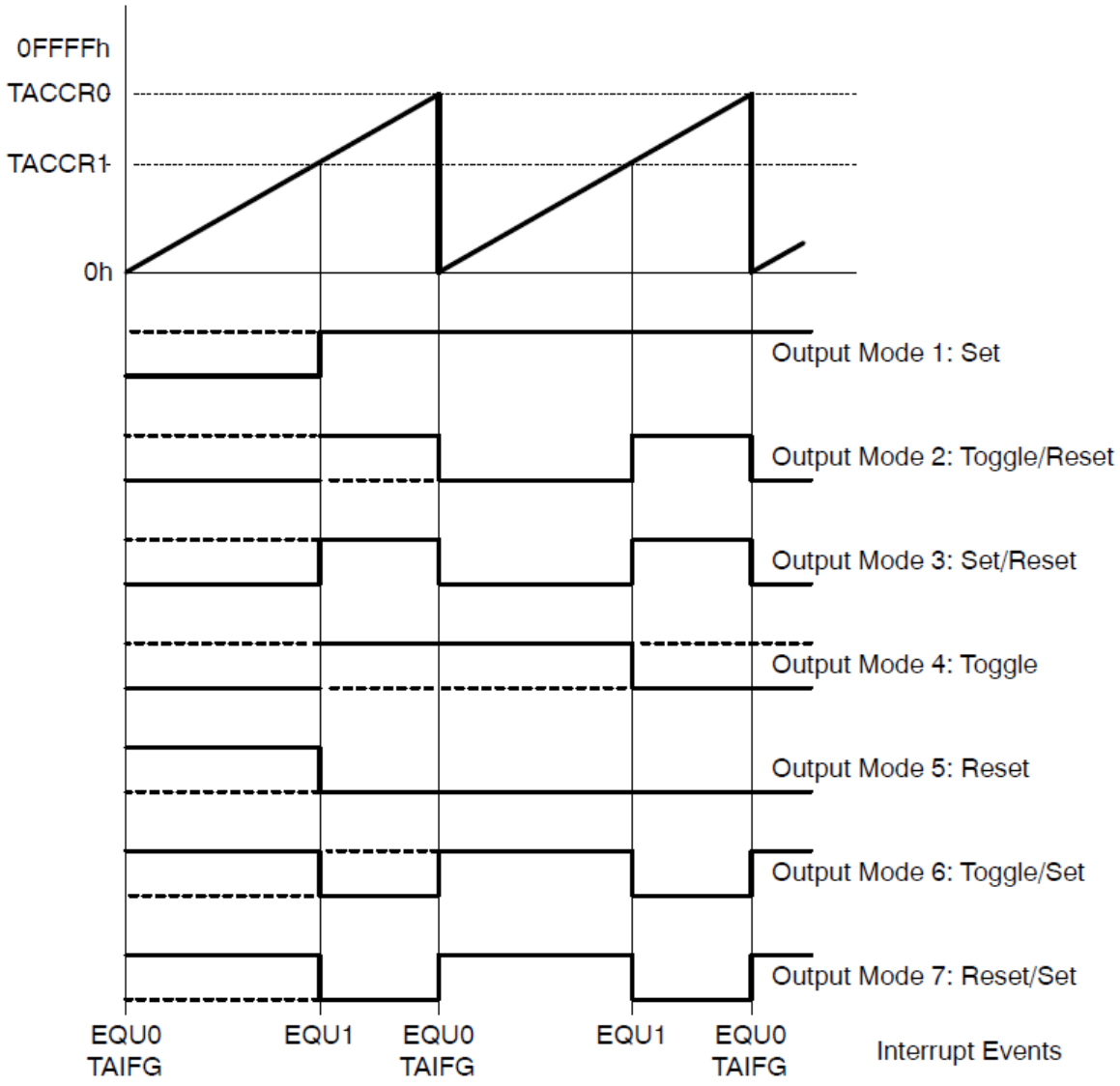
15	14	13	12	11	10	9	8
CMx		CCISx		SCS	SCCI	Unused	CAP
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	r0	rw-(0)

7	6	5	4	3	2	1	0
OUTMODx			CCIE	CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

OUTMODx Bits 7-5 Output mode. Modes 2, 3, 6, and 7 are not useful for TACCR0 because EQUx = EQU0.

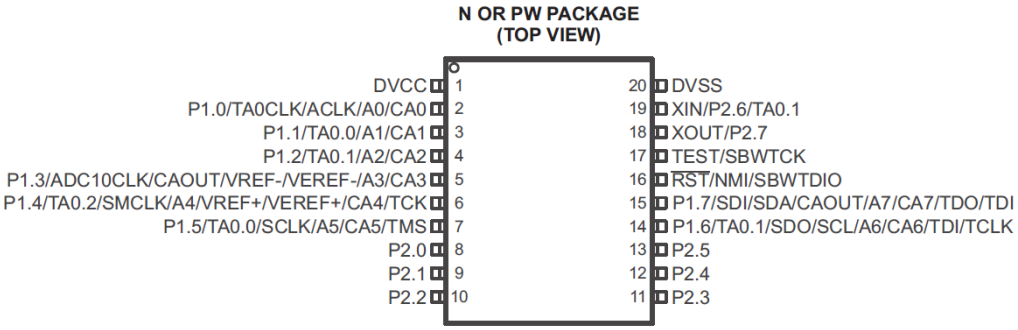
000 OUT bit value
001 Set
010 Toggle/reset
011 Set/reset
100 Toggle
101 Reset
110 Toggle/set
111 Reset/set

NOM : **PRENOM :**


0FFFFh
TACCR0
TACCR1
0h
Output Mode 1: Set
Output Mode 2: Toggle/Reset
Output Mode 3: Set/Reset
Output Mode 4: Toggle
Output Mode 5: Reset
Output Mode 6: Toggle/Set
Output Mode 7: Reset/Set
EQU0 TAIFG
EQU1
EQU0 TAIFG
EQU1
EQU0 TAIFG
Interrupt Events

NOM : **PRENOM :**

ANNEXE 1: Brochage MSP430G2452



NOTE: ADC10 pin functions are available only on MSP430G2x52.

Symboles du langage C correspondant aux opérateurs :

&	ET
|	OU
^	OU-exclusif
~	inversion
<< (>>)	décalage à gauche (resp. droite)