DS n°1 Module I2 2010

Durée 2H: Tous les Documents et calculettes autorisés :
3 parties indépendantes.
Vous devrez répondre aux questions **directement dans le sujet** (partie droite du sujet). Total brut sur 42 pts Note finale = note brute/2

Indiquez ici si vous disposez officiellement d'un tiers temps officiel :
☐ **Je dispose d'un tiers temps officiel** (cocher) :groupe =

Si c'est le cas, vous n'êtes pas obligé de traiter l'exercice 2 du DS (décodage d'adresse).

Nom ----------

Prenom ----------

Groupe ----------

1. Exercices sur les masquages et décalages /9

Veuillez indiquer ci après la valeur des résultats de masquage ou des masques ayant permis d'obtenir un certain résultat.

1.1. Résultats de Masquages

Donnez la valeur obtenue après masquage dans chacun des 3 exercices .

```
Ex1:Valeur initiale:0111 . 1100 . 0110 . 1001
    Masque ET      :0101 . 0101 . 0011 . 1001
  Résultat masquage:
```
0.5

```
EX2 Valeur initiale   :0x D . 3 . B . 6
    Masque OU         :0x 2 . C . 3 . 5
    Résultat masquage :0x
```
0.5

```
Ex3 Valeur initiale   :0x 0 . F . C . C
    Masque ET         :0x A . 7 . E . B
    Résultat masquage :0x
```
0.5
/1.5

1.2. Exemple de masquages.

On donne des valeurs d'un registre iReg qui a été masqué par 1, 2 ou 3masques.

- X désigne un bit qui n'a pas changé après les masquages.
- Ŷ désigne un bit qui a été complémenté à 1 par rapport à sa valeur d'origine après les masquages.
- 1 un bit qui a été forcé à 1 et
- 0 un bit qui a été forcé à 0 après les opérations de masquage.

Il s agit de retrouver quels masques ont permis d'aboutir au résultat de iReg.
Indiquez à chaque fois le type de masque et leur valeur qui ont été nécessaire pour obtenir le résultat final.
Les masques doivent pouvoir être appliqués dans n'importe quel ordre.
Indiquez en code 'C' les lignes de codes à écrire pour réaliser ces masquages.

Ex1:*Apres masquage* X11X.1X00.0011.X0X1
Type et valeur **ET** 0.5
Type et Masque **OU** 0.5
Code 'C' correspondant à ces 2 Masquages sur iReg
iReg= iReg 0.5
iReg= iReg

Ex2:*Apres masquage* 11X0.000X.X1X0.1010
Type et Masque 0.5
Type et Masque 0.5
Code 'C' correspondant à ces 2 Masquages sur iReg
iReg=iReg 0.5
iReg=iReg

Ex3:*Apres masquage* XŶ10.XŶXŶ.11ŶX.XŶ01
Type et Masque 0.5
Type et Masque 0.5
Type et Masque 0.5
Code 'C' correspondant à ces 3 Masquages sur iReg
iReg= iReg 1
iReg= iReg
iReg= iReg
/5.5

1.3. Décalages à droite et gauche

On donne une valeur codée hexa décimal. Donnez la valeur finale après les opérations de décalage à droite(>>) ou gauche(<<) effectuée par le uP.

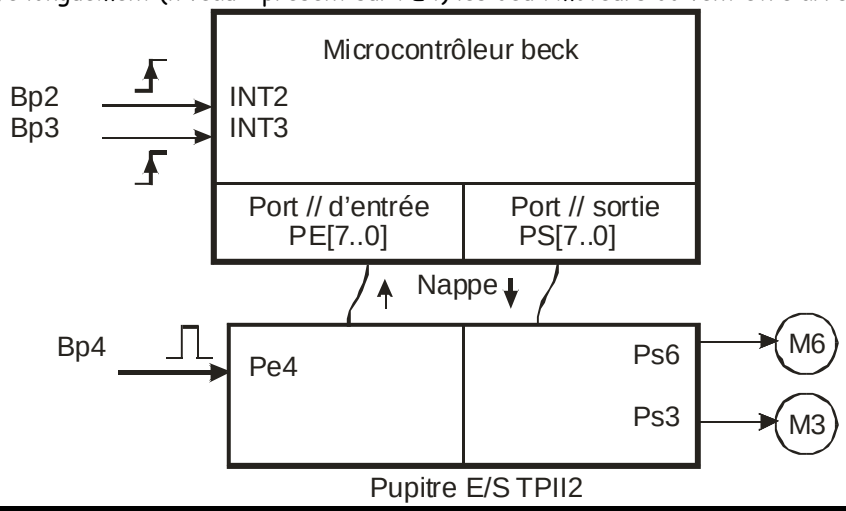
0x1234>>1 = 0.5
0x5678<<1= 0.5
((0xA874>>3)&0x0005= 1
/2

2. Décodage d'adresse /8

On dispose d'un processeur en architecture 8 bits dont le bus d'adresse externe est de largeur 14bits (A13..A0). On veut implanter dans son plan mémoire différents boitiers mémoire ram et rom ainsi qu'un périphérique d'E/S . Même si les adresses sont codées sur 14 bits nous utiliserons par commodité une notation de celles-ci sur 4 digits hexadécimaux. L'adresse de reset de ce uP est 0x0000. Les adresses désignent toujours une case mémoire de 1 octet de large.		
2.1. Taille de l'espace adressable Donnez la taille de (en nombre d'octets) de l'espace adressable de ce processeur. Indiquez en hexa décimal l'adresse de départ et l'adresse de fin de cette espace adressable.	Taille en Octet= Adresse de départ(hexa)= Adresse de fin (hexa)=	0.5 0.5 0.5 /1.5
2.2. Mise en place de la ram1 On veut décoder un boitier RAM1 de taille 4Ko à partir de l'adresse de l'@ 0x1000. Parmi A13..A0 quels bits d'adresse appartiennent à la partie fixe des adresses, quels bits appartiennent à la partie variable des adresses, quelle est la plage d'adresse (en hexa) occupée par ce boitier RAM1 dans l'espace adressable	Bits de la partie variable= Bits de la partie fixe= Plage d'adresse occupée:	0.5 0.5 0.5 /1.5
2.3. Equation de décodage d'adresse de RAM1. A partie de Q2.2, déterminez l'équation logique de sélection (/CS_RAM1) du boitier RAM1. On suppose que le signal de sélection de ce boitier (/CS_RAM1) est actif au niveau bas.	/CS_RAM1=	0.5 /0.5
2.4. Mise en place de ROM2 On a implanté dans le plan mémoire un deuxième boitier mémoire de type ROM dont on ne connaît pas la taille. On sait simplement que sa plage d'@ occupée va de 0x0000 à 0x07FF Déterminer sa taille (en octets), les bits de la partie fixe et variable et son équation logique de décodage en fonction de A13..A0. On suppose que le signal de sélection de ce boitier (/CS_ROM2) est actif au niveau bas.	Taille de ROM2 (en octets)= Bits de la partie variable= Bits de la partie fixe= Equation de /CS_ROM2=	0.5 0.5 0.5 0.5 /2
2.5. Mise en place d'un périphérique d'E/S. On doit localiser dans le plan mémoire un périphérique d'E/S qui possède 64 registres internes à localiser dans le plan mémoire. L'adresse du dernier registre de ce périphérique en mémoire (celui dont l'adresse est la plus élevée)=0x2F7F Il est sélectionné grâce à un signal /CS_IO actif au niveau bas. Déterminer les bits d'adresse appartenant à la partie fixe, à la partie variable, sa plage mémoire occupée dans l'espace adressable (en hexa) et l'équation de décodage du signal:/CS_IO.	Partie variables des adresses= Partie fixe des adresses= Plage mémoire occupée(en hexa)= Equation de /CS_IO=	0.5 0.5 0.5 1 /2.5

3. Utilisation du port // d'E/S et des interruptions externes du microcontrôleur Beck /15

On en vu en TP que la maquette Beck possédait un port de sortie parallèle PS[7..0] et un port d'entrée // PE[7..0].
Ces deux ports de 8 bits sont accessibles via les fonctions du pilote Port.c et Port.h (PIO_Init(), PIO_Read() et PIO_Write()).
On veut réaliser une application très simple pour piloter un téléski . On dispose de 3 boutons poussoirs BP2 (connecté à une broche d'interruption externe INT2), BP3 (connecté à une broche d'interruption externe INT3) et BP4 connecté sur le bit PE4 du port // d'entrée PE[7..0] .
L 'ascenseur est commandé par 2 moteurs M3 servant à faire monter et M6 servant à faire descendre le téléski . M3 est piloté par le bit PS3 du port // PS et M6 par le bit PS6. Les autres bits du port PS ne doivent en aucun cas être modifiés. Les deux moteurs M3 et M6 ne fonctionnent jamais en même temps.
Le fonctionnement est le suivant.
Si un front montant est généré par un appui bref sur BP2 le téléski monte (M31, M6=0). Si un front montant est généré par un appui bref sur BP3 , l'ascenseur descend (M3=0, M6=1).
Si le bouton BP4 est appuyé longuement (niveau 1 présent sur PE4) les deux moteurs doivent être arrêtés (M3=0 et M6=0).



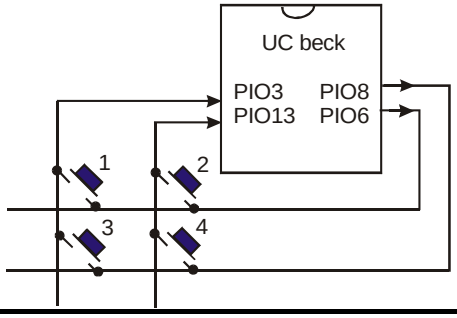
Pupitre E/S TPII2

3.1. Lecture de BP4 Donnez le code de la fonction **`int Read_BP4(void)`** qui lie le port d'entrée // PE[7..0] et qui retourne la valeur 1 si BP4 est appuyé (niveau 1 présent sur PE4) ou 0 si BP4 est relâché (niveau 0 sur PE4). Les autres bits de PE[7..0] peuvent présenter une valeur quelconque.		0.5 0.5 0.5 0.5 /2
3.2. Initialisation des IT externes Donnez le code de la fonction **`void Init_IT2_3(void)`** qui : Configure les broches INT2 et INT3 en tant que broche d'interruption externe. Les rends toutes deux actives sur front montant. Mets en place les deux fonctions d'interruption externe ISR2 et ISR3(void) liées respectivement aux broches INT2 et INT3.		0.5 0.5 1 0.5 0.5 /3

3.3. Commande des 2 moteurs. Donnez le code de la fonction `Void Com_Machine(int iCom)` qui reçoit en paramètre d'entrée un nombre iCom égal à : « 0 » pour arrêter les moteurs M3 et M6 « 3 » pour alimenter M3 et arrêter M6, « 6 » pour alimenter M6 et arrêter M3. On met en route un moteur en envoyant 1 sur son bit de commande correspondant. gbyValPS est une variable globale qui représente à tout instant la valeur présente sur le port //PS. Cette variable sera masquée puis écrite physiquement sur le port PS.		1 1 1 0.5 /3.5
3.4. fonction d'interruption ISR2 Donnez le code de la fonction d'interruption ISR2() qui permet de mettre en route le moteur M3 à chaque fois qu'un front montant est détecté par le uP sur la broche INT2 (utiliser la fonction Com_Machine() pour cela.		0.5 /0.5
3.5. fonction d'interruption ISR3 Donnez le code de la fonction d'interruption ISR3() qui permet de mettre en route le moteur M6 à chaque fois qu'un front montant est détecté par le uP sur la broche INT3 (utiliser la fonction Com_Machine() pour cela.		0.5 /0.5
3.6. void main (void) Donnez le code de la fonction main() qui déclare et initialise gbyValPS 0, initialise le Port de sortie PS[7..0], initialise les interruptions externes et qui ensuite, en boucle tant que l'utilisateur n'appuie pas sur une touche de la console série, vient regarder toutes les 200ms l'état du bouton BP4 pour demander à arrêter les deux moteurs . En cas d'appui sur une touche de la console, on désinstalle les deux fonctions d'interruptions ISR2() et ISR3 et on sort de l'application en ayant arrêté au préalable les deux moteurs M3 et M6. Le main() utilise les fonctions écrites précédemment.		0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 /5.5

4. Utilisation des broches d'E/S individuelles présentes sur le uC Beck /10

On en vu en TD4 et TP3 que le uC BECK possédait des broches individuelles d'E/S (des pio) qui sont commandables individuellement grâce à des fonctions de l'API Beck (pfe_enable_pio(), hal_write_pio() et hal_read_pio())
On a connecté un clavier à 4 touches (numérotées '1','2','3' et '4' sur 4 de ces broches.
Deux broches seront commandées en sortie (PIO6 et PIO8) et deux broches seront utilisées en entrée (PIO3 et PIO13) et seront doté dans le uC de deux résistances de pull down . Ainsi si les touches connectées à cette ligne sont relâchées, le niveau par défaut sur ces broches d'entrée sera égal à 0 (tiré par la résistance de pull down)



Le fonctionnement est le suivant.
Au départ on met PIO6=1 et PIO8=0 :

- si la touche '1' est enfoncée PIO3=1/PIO13=0
- si la touche '2' est enfoncée PIO13=1/PIO3=0.

Puis on met PIO6=0 et PIO8=1 :

- si la touche '3' est enfoncée PIO3=1/PIO13=0
- si la touche '4' est enfoncée PIO13=1/PIO3=0

Si aucune touche n est enfoncée, PIO3=0 et PIO13=0
Enfin on affiche le numéro de la touche sur la console Telnet du Beck.

4.1. Initialisation des pins d'E/S Donnez le code de la fonction void **`Init_ES(void)`** qui configure les broches PIO3 et PIO13 du Beck en entrée avec une résistance de pull down et configure les broches PIO6 et PIO8 du Beck en sortie avec comme valeur initiale '0'.		0.5 0.5 0.5 0.5 /2
4.2. Lecture du clavier avec verrouillage Donnez le code de la fonction int **`Read_Key(void)`** qui : Met PIO6 à 1 puis PIO8 à 1 et à chaque fois vient lire la valeur de PIO3 et PIO13 pour déterminer quelle touche est actuellement appuyée. Cette fonction retourne le numéro de la touche appuyée (en testant la valeur de PIO3 et PIO13) ou retourne 0 si aucune touche n'est appuyée On suppose dans cet exercice qu'un verrouillage mécanique empêche que plusieurs touches soient appuyées en même temps.		0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 /4
4.3. Lecture clavier sans verrouillage Maintenant donnez le code de la fonction int Read_Key3(void) dans le cas ou il n'y a pas de verrouillage mécanique (plusieurs touches peuvent être appuyées en même temps). Dans ce cas la valeur retournée est la somme des valeurs de touches appuyées. Une idée pourrait consister à affecter un poids (ex poids=1 pour la valeur de PIO3 et poids=2 pour la valeur de PIO13) de sommer la valeur de ces 2 poids pour déterminer quelle est la touche est appuyée lorsque que PIO6 est actif. Même idée pour PIO8 actif.		0.5 0.5 1 0.5 0.5 1 /4

`void` **`BIOS_Set_Focus`** `( IO_FOCUS focus );`

Set STDIO focus.

SC1X Parameters

Focus: Enumeration type:
FOCUS_SHELL = 1: Set focus to command shell
FOCUS_APPLICATION = 2: Set focus to application
FOCUS_BOTH = 3: Set focus to both application and command shell

Return Value: -- none --

Comments :This function sets the focus of STDIO to either console, application or both.

If your application requires input from the user, you should set the focus to the application. You should take care that only one application requests input from STDIO.
The user can change the focus by using the focus hot key (default Ctrl-F). Changing the focus clears the serial input and output queues immediately.

Important :
All buffered incoming and outgoing characters in the internal serial send and receive queues are lost after this call.
This function includes a one millisecond sleep.

`unsigned int` **`BIOS_kbhit`** `( void );`

BIOS_kbhit
Test if character is ready at standard input(s).

SC1X Parameters

-- none --

Return Value :Boolean: '1' if characters is ready, '0' if no character ready.

Comments: All devices selected as standard input (e.g. EXT, COM and Telnet) are polled for data ready.

`void` **`pfe_set_edge_level_intr`**`_mode ( unsigned char mode, unsigned short mask );`

Set edge/level interrupt mode for INTx.

SC1x Parameters

Mode:1 = active high, level-sensitive interrupt
0 = low-to-high, edge-triggered interrupt

mask : Bits set to designate interrupts affected:
Bit 0 = INT0
Bit 1 = don't care
Bit 2 = INT2
Bit 3 = INT3
Bit 4 = INT4
Bit 5..15 = don't care

Return Value -- none --.

SC1x Comments
Default for all interrupts is edge-triggered mode. In each case (edge or level) the interrupt pins must remain high until they are acknowledged.

void pfe_enable_int (unsigned char irq);	`InterruptHandler` **`hal_install_isr`** `( unsigned short irq, unsigned short count, InterruptHandler handler );`
Enable external interrupt requests INTx SC1x Parameters **irq** :External Interrupt number: 0: Enable INT0 1: N/A 2: Enable INT2 3: Enable INT3 4: Enable INT4 5: Enable INT5 6: Enable INT6 **Return Value:**-- none --. **SC1x Comments:** Used pins: INT0, INT[2..6] **Excluded pins**: if INT0: PIO13, TMROUT0, cascaded interrupt controller if INT2: PIO6, PCS2#, INTA#, PWD, SPI, hw flow control serial port 1 if INT3: PIO12, serial port 1 if INT4: PIO5, PCS3#, SPI, hw flow control serial port 1 if INT5: PIO1, DRQ0, default I2C-Bus pins, SPI if INT6: PIO0, DRQ1, default I2C-Bus pins, SPI	Install Interrupt Service Routine. SC1x Parameters **Count:**Number of interrupts generated before new user interrupt service routine is called. count = 0 disables the user ISR (same as a NULL in handler) **Handler:**far pointer to user interrupt service routine if pointer is NULL user ISR is disabled **irq**: HAL interrupt number from following list: 0 = INT0 (external) 1 = (*) Network controller (internal) 2 = INT2 (external) 3 = INT3 (external) 4 = INT4 (external) 5 = INT5 (external) / DMA Interrupt Channel 0 (if DMA is used) 6 = INT6 (external) / DMA Interrupt Channel 1 (if DMA is used) 7 = reserved 8 = Timer0 (internal) 9 = Timer1 (internal) 10 = (*) Timer 1ms (internal) 11 = (*) Serial port 0 (internal) 12 = (*) Serial port 1 (internal) 13 = (*) DMA Interrupt Channel 2 (internal, not on SC12) 14 = (*) DMA Interrupt Channel 3 (internal, not on SC12) 15 = NMI Powerfail Interrupt (internal/external) (*) = internal used by @CHIP-RTOS, not available for user interrupt service functions **Return Value**: Far pointer to old ISR handler **Comments:**The user-defined ISR is called from a system ISR with the interrupt identifier number in the BX CPU register, thus allowing for a single user ISR to handle multiple interrupt sources. The user ISR can be declared in either of the following forms: void huge My_ISR(void) ; // More efficient form

`void` **`pfe_enable_pio`**`(unsigned short pio,unsigned char mode );`	`unsigned char` **`hal_read_pio`** `( unsigned char pio );`
pfe_enable_pio Enable used programmable I/O pins. Define which pins are inputs and which are outputs. This function can be called several times for definition of different PIO pins. With repeated selection of the same pin, the definition made last is valid. The selection of a PIO pin can be cancelled by calling the appropriate PFE function that causes the respective PIO pin to be used for another purpose (e.g. function pfe_enable_pcs for PIO2). SC13 Parameters **pio** :PIO pin number, [0..13] **mode**: 1 = Input without pullup/pulldown 2 = Input with pullup (not PIO13) 3 = Input with pulldown (only for PIO3 and PIO13) 4 = Output init value = High 5 = Output init value = Low **Return Value:**-- none --. Comments Used pins: PIO[0..x] Excluded pins: All other functionality on the selected PIO pin.	hal_read_pio: Read specific I/O pin. SC13 Parameters **pio** :IPC@CHIP® PIO No. [0..13] **Return Value:** 0 PIO pin is low, 1 PIO pin is high **Comments:** Read specified programmable I/O pin.
	`void h`**`al_write_pio`**`( unsigned char pio,unsigned short value );`
	hal_write_pio: Write to specific I/O pin. SC13 Parameters **pio** :IPC@CHIP® PIO No. [0..13] **Value**: 0 ==> set PIO to low non-zero ==> set PIO to high **Return Value**: -- none --.