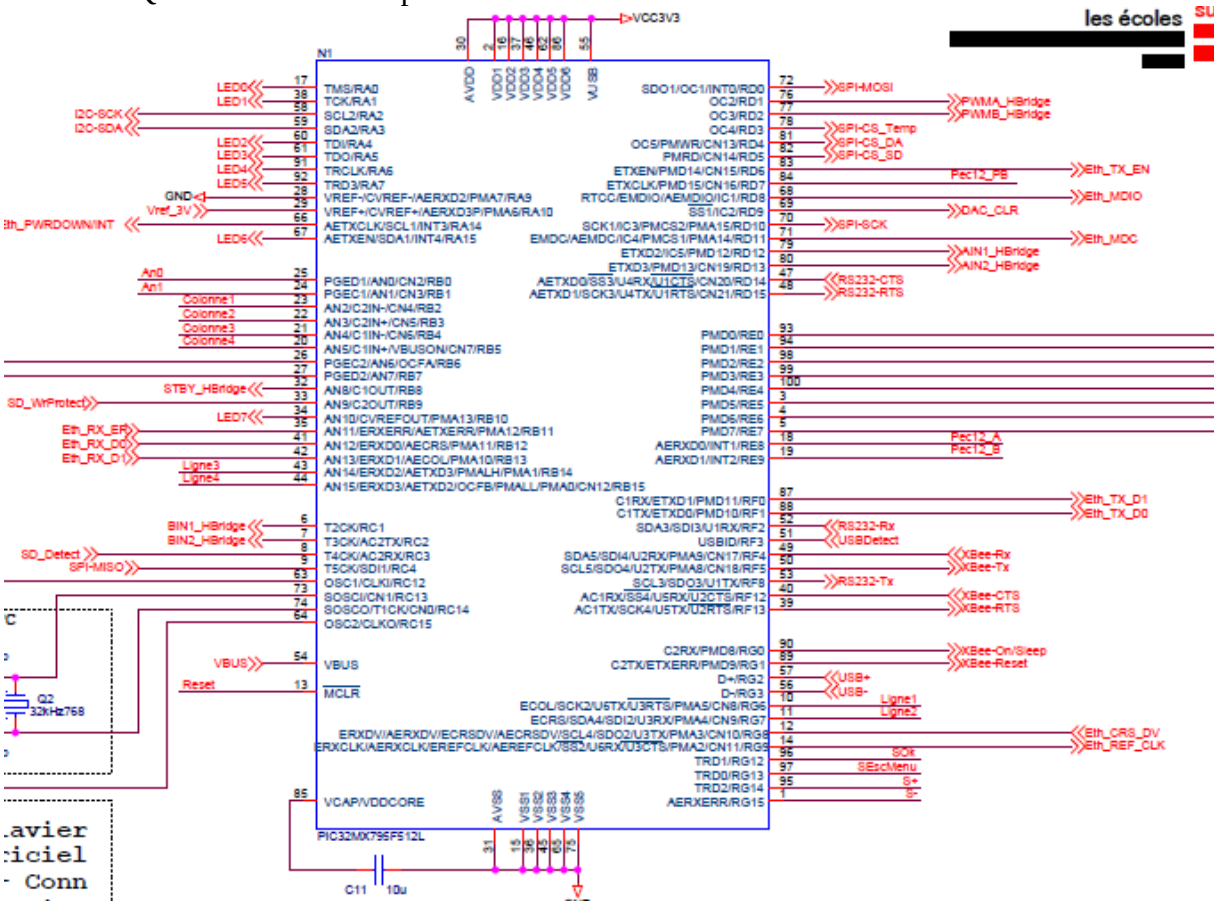
Solution exercice révision chapitres 2 à 6

Cette révision porte sur les chapitres 2 à 6 du cours "programmation des PIC32MX".

L'ensemble des cours de théorie et de laboratoire sont à disposition.

QUESTION 1

a) Pour effectuer un essai d'un module on souhaite réalisez un bus de 4 bits (les 4 lignes sur le même port ainsi que des lignes contiguës). Le port E est entièrement utilisé par l'affichage LCD. On souhaite conserver le plus de fonctionnalités du kit (ports série pour RS232, SPI, I2C, sortie OC et input capture. Par contre, il est possible de sacrifier les leds et le clavier matriciel. Quels sont les choix possibles avec le moins de conflits avec le hardware existant ?



On peut utiliser soit Led2 à Led5, soit colonne1 à colonne 4.
Led2 à Led5 correspond à RA4 à RA7
Colonne 1 à 4 correspond à RB2 à RB5

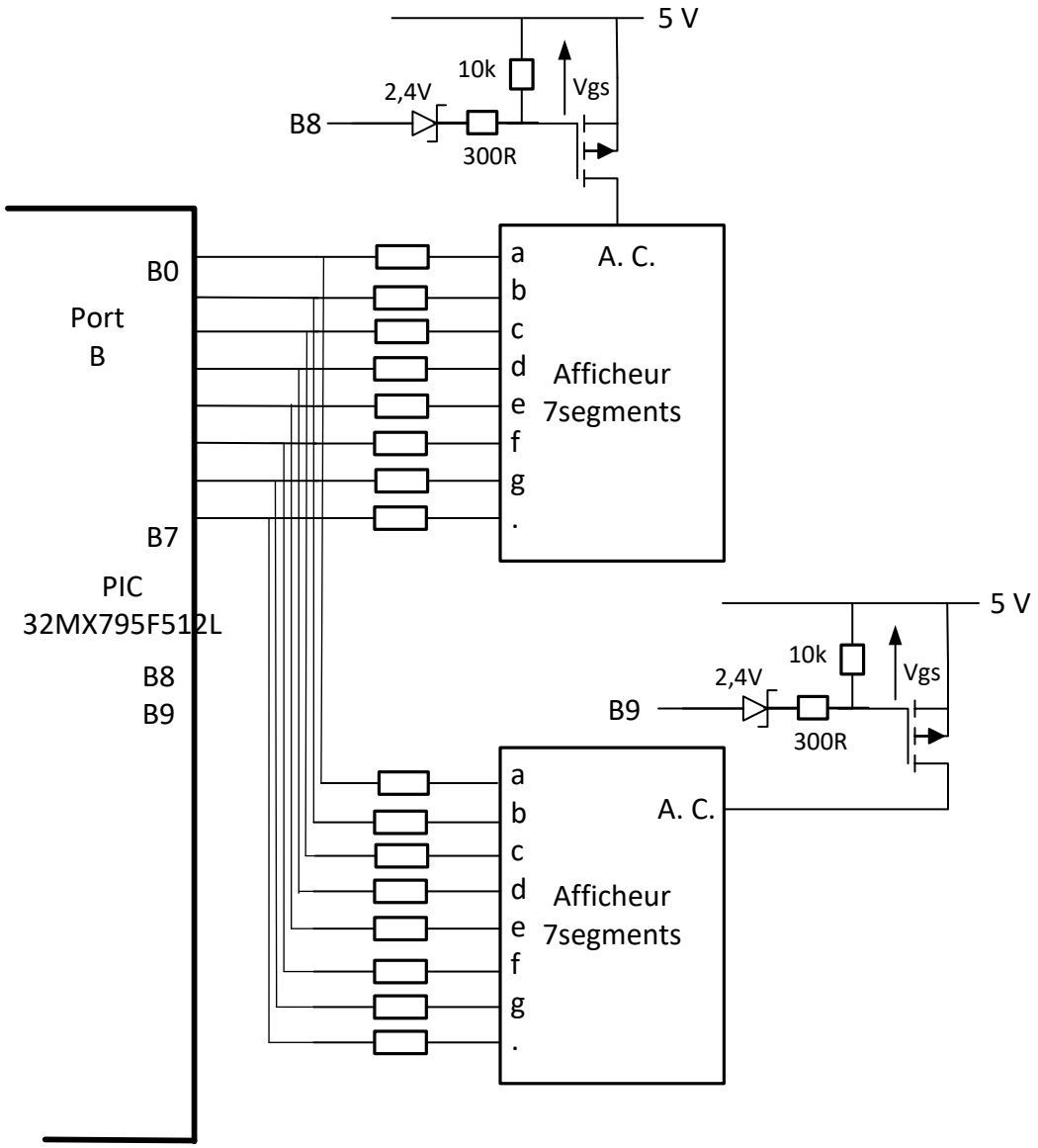
Choix de RA4 à RA7 dans notre exemple de solution.

b) Il est demandé, en utilisant directement les registres TRIS, PORT et LAT, de réaliser les actions de lecture et d'écriture sans entrer en conflit ni modifier le comportement des autres lignes du port. Il est en plus demandé d'effectuer les action d'écriture et de lecture globalement sur l'entier du port.

```
// Choix de Led2 à Led5 (RA4 à RA7)
#define mask_bus4 0x00F0


void main() {

    uint16_t ValBusIn;
    uint16_t ValBusOut = 0x0A;
    uint16_t tmp;

    // Modification de la direction
    // action sur l'entier de TRISA
    tmp = TRISA;
    tmp = tmp & ~mask_bus4;  // 0 pour output
    TRISA = tmp;
    // OU Variante action TRISAbits
    TRISAbits.TRISA4 = 0;
    TRISAbits.TRISA5 = 0;
    TRISAbits.TRISA6 = 0;
    TRISAbits.TRISA7 = 0;

    // Mise en place valeur BusOut dans image du portA
    // (pour écriture sur tous le portA)
    tmp = PORTA;
    tmp = tmp & ~mask_bus4;       // les 4 bits à 0
    tmp = tmp | (ValBusOut << 4); // ajout valBus
    LATA = tmp;   // Ecriture sur le bus 4 bits

    // Configuration pour Lecture du bus
    // (On souhaite les 4 bits du bus dans le poids
    // faible de ValBusIn)
    // Action sur l'entier de TRISA
    tmp = TRISA;
    tmp = tmp | mask_bus4;  // 1 pour input
    TRISA = tmp;
    // OU Variante action TRISAbits
    TRISAbits.TRISA4 = 1;
    TRISAbits.TRISA5 = 1;
    TRISAbits.TRISA6 = 1;
    TRISAbits.TRISA7 = 1;
   // Lecture PORTA et isolation du bus de 4 bits
    tmp = PORTA;
    ValBusIn = (tmp & mask_bus4) >> 4;
}
```

QUESTION 2

Le PIC est alimenté en 3V3, les afficheurs en 5V. Voici le câblage de 2 afficheurs 7seg. Un seul des afficheurs doit être alimenté. Donnez l'action sur la ligne B8 pour faire conduire le MOS (Vgs ≤ Vgs,threshold) et l'action pour le bloquer (Vgs = 0 V). Utilisation des fonctions bit de plib_ports de Harmony. On suppose que B8 et B9 sont configurés en sortie.

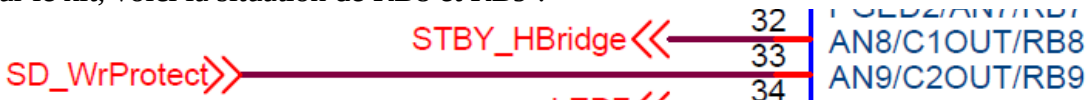



```
// Action sur B8 pour Vgs = -5 V
.............................................
.............................................
.............................................
.............................................
// Action sur B8 pour Vg = 0 V
.............................................
.............................................
.............................................
.............................................
```

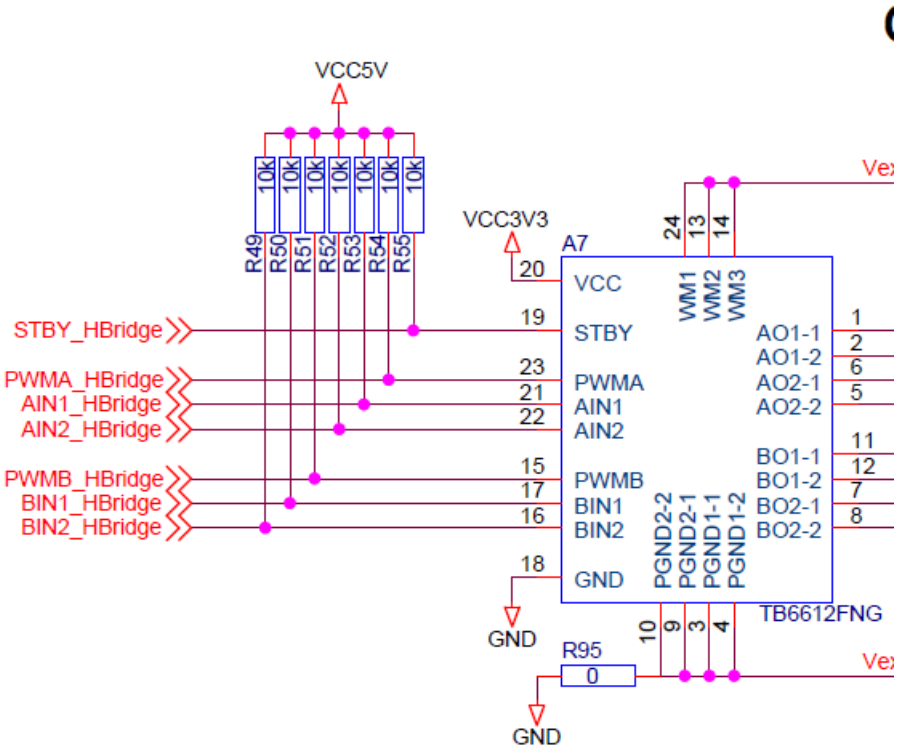
QUESTION 2- TEST PRATIQUE

Le test pratique ci-dessous a été réalisé sur le kit. Le schéma des pins connectées au +5V n'est pas identique à celui de la question.

Sur le kit, voici la situation de RB8 et RB9 :



On constate de STBY_Hbridge est câblé avec une pull-up au +5V, ce qui correspond à notre situation.

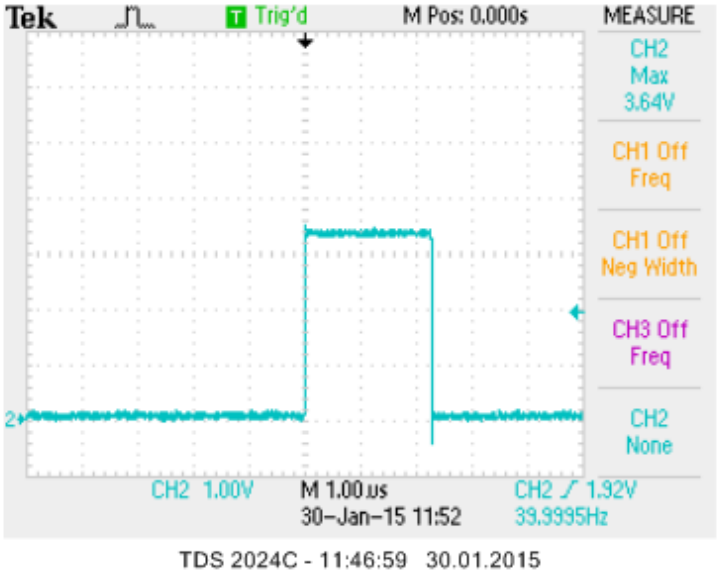


SITUATION EN MODE NORMAL

Configuration de la direction:

```
PLIB_PORTS_PinDirectionOutputSet( PORTS_ID_0,PORT_CHANNEL_B,
                                  PORTS_BIT_POS_8 );

// Action sur B8 pour Vg = -5 V
// Il faut imposer B8 low (clear)
PLIB_PORTS_PinClear(PORTS_ID_0, PORT_CHANNEL_B,
                                PORTS_BIT_POS_8);

// Action sur B8 pour Vg = 0 V
// Il faut établir B8 High (Set)
PLIB_PORTS_PinSet(PORTS_ID_0, PORT_CHANNEL_B,
                              PORTS_BIT_POS_8);
```

On constate que l'on atteint 3.64V ce qui est supérieur à la tension d'alimentation de 3V3.

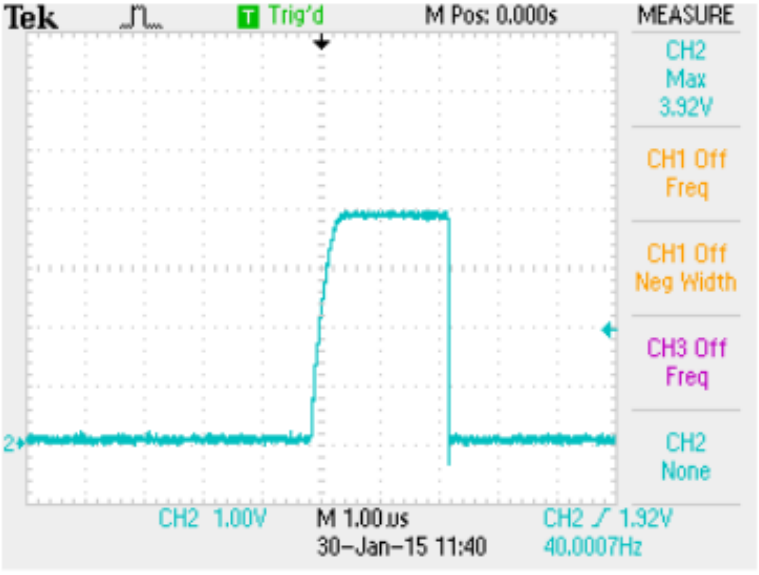


VARIANTE ACTION POUR ETAT HIGH

Si, au lieu d'imposer l'état à high, on force la broche en entrée (haute impédance) on obtient un meilleur résultat, car la broche se laisse tirer au 5 volts.

```
// Action sur B8 pour Vg = -5 V
// Il faut imposer B8 low
PLIB_PORTS_PinDirectionOutputSet( PORTS_ID_0,PORT_CHANNEL_B,
                                  PORTS_BIT_POS_8 );
PLIB_PORTS_PinClear(PORTS_ID_0, PORT_CHANNEL_B,
                                PORTS_BIT_POS_8);

// Action sur B8 pour Vg = 0 V
// Il faut établir état Hi-z (La pullUp assure le niveau haut)
PLIB_PORTS_PinDirectionInputSet( PORTS_ID_0, PORT_CHANNEL_B,
                                 PORTS_BIT_POS_8 );
```

On constate que l'on atteint 3.92V. Par contre le flanc montant présente l'effet du RC de la pullup de 10k et de la capacité d'entrée du circuit.



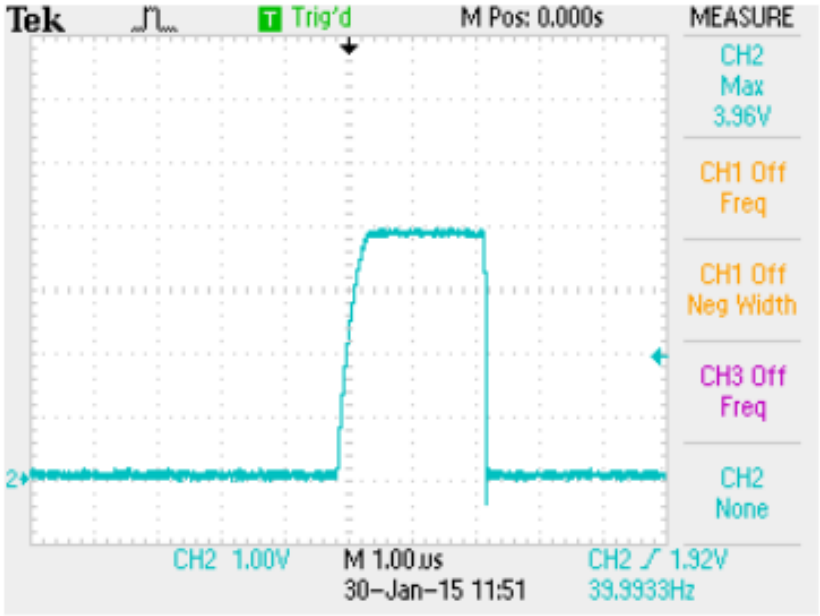
SITUATION AVEC CONFIGURATION OPEN DRAIN

Si lors de la configuration de la direction on ajoute une action **OpenDrainEnable**, il est possible d'utiliser l'action PinSet pour le niveau haut (donc haute impldance).

```
PLIB_PORTS_PinDirectionOutputSet( PORTS_ID_0,PORT_CHANNEL_B,
                                  PORTS_BIT_POS_8 );
PLIB_PORTS_PinOpenDrainEnable( PORTS_ID_0,PORT_CHANNEL_B,
                               PORTS_BIT_POS_8 );
```

On peut écrire de manière classique :

```
// Action sur B8 pour Vg = -5 V
// Il faut imposer B8 low
PLIB_PORTS_PinClear(PORTS_ID_0, PORT_CHANNEL_B,
                                PORTS_BIT_POS_8);

// Action sur B8 pour Vg = 0 V
// Il faut établir B8 High
PLIB_PORTS_PinSet(PORTS_ID_0, PORT_CHANNEL_B,
                              PORTS_BIT_POS_8);
```



On constate que l'on atteint 3.96V, par contre le flanc montant présente l'effet du RC de la pullup de 10k et de la capacité d'entrée du circuit.

☺ Cette solution présente l'avantage de ne pas modifier le princicpe des actions set et clear. C'est uniquement lors de la configuration qu'il faut agir.

QUESTION 3

Soit l'élément suivant de programme en C :

```
int32_t val;
```

On suppose que la variable val se situe à l'offset 4 par rapport à S8 (copie du SP)

```
val = 0;
val += 0x31;
```

Donnez l'équivalent en assembleur PIC32MX, en supposant que l'on commence à l'adresse 9D000014.

Adresse	Instruction en assembleur	Commentaire
9D000014	SW ZERO, 4(S8)	val = 0 par transfert reg 0;
9D000018	LW V0, 4(S8)	val -> V0
9D00001C	ADDIU V0, V0, 49	V0 = V0 + 49 49 => 0x31
9D000020	SW V0, 4(S8)	V0 -> val

Vérification pratique de la solution :

```
24:                     val = 0;
9D000014  AFC00004   SW ZERO, 4(S8)
25:                     val += 0x31;
9D000018  8FC20004   LW V0, 4(S8)
9D00001C  24420031   ADDIU V0, V0, 49
9D000020  AFC20004   SW V0, 4(S8)
```

QUESTION 4

Pour les 4 instructions PIC32MX ci-dessous :

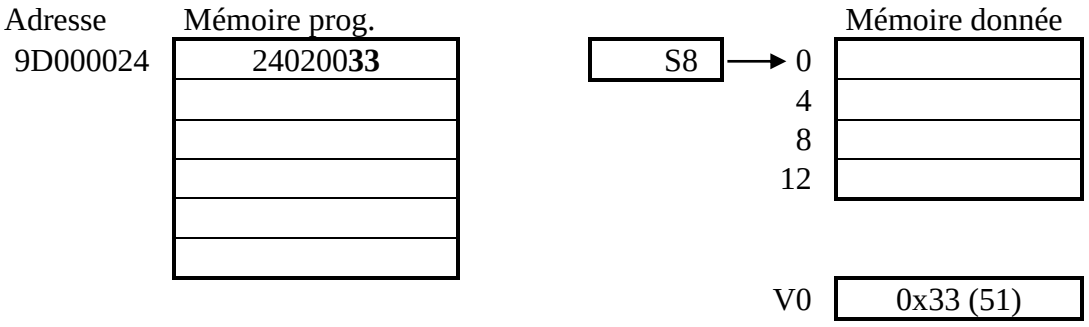
a) Donnez le code machine hexadécimal de l'instruction dans la mémoire programme, puis décrivez le mouvement des données pour chaque instruction en donnant la situation après l'exécution (mémoire DATA ou registre).

b) Donnez l'équivalent en langage C.

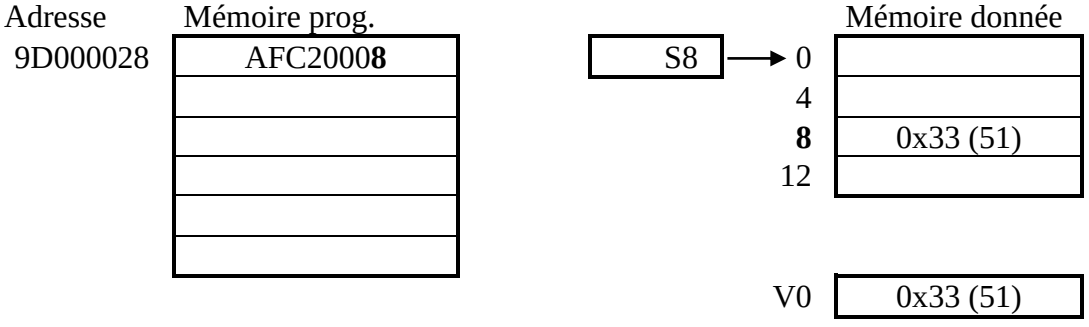
```
Start:      ADDIU V0, ZERO, 51
            SW V0, 8(S8)
            LW V0, 8(S8)
            SW V0, 12(S8)
```

On suppose que l'étiquette Start correspond à 9D000024 et 8(S8) correspond à la variable val1 ainsi que 12(S8) correspond à la variable val2.

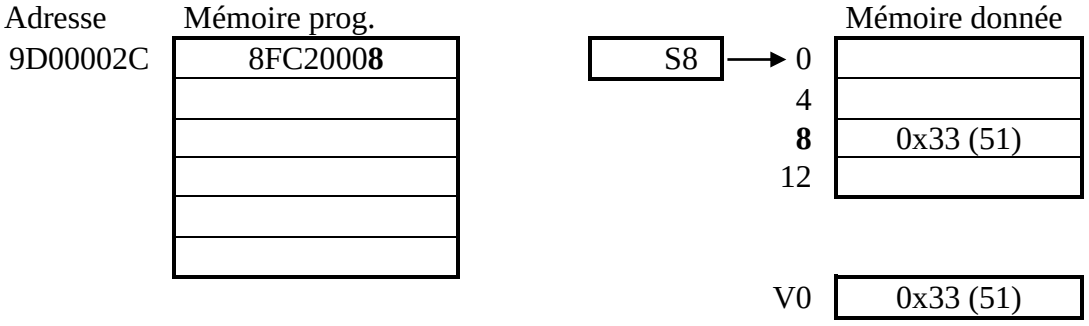
a)
ADDIU V0, ZERO, 51



SW V0, 8(S8)

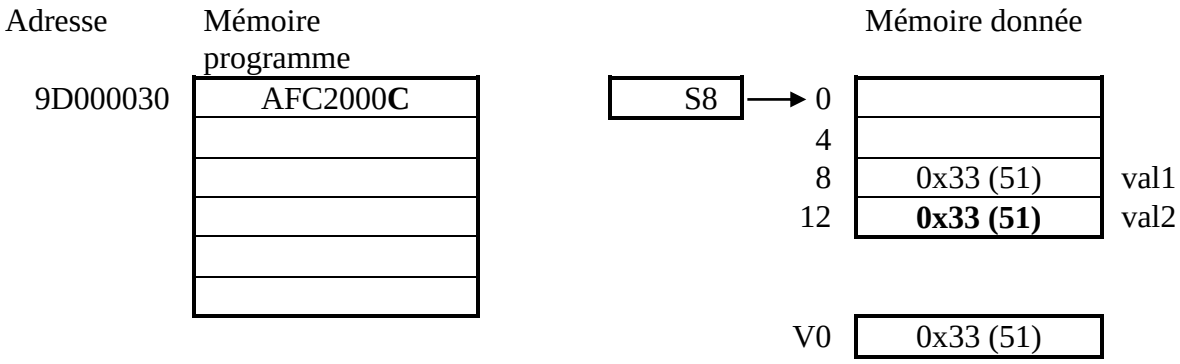


LW V0, 8(S8)



Remarque : V0 est reécrit avec la même valeur.

SW V0, 12(S8)



b) Équivalent en C

```
int32_t val1;
int32_t val2;

val1 = 0x33;
val2 = val1;
```

Vérification pratique (en partant du C et en observant le listing assembleur).

```
27:                         val1 = 0x33;
9D000024  24020033    ADDIU V0, ZERO, 51
9D000028  AFC20008    SW V0, 8(S8)
28:                         val2 = val1;
9D00002C  8FC20008    LW V0, 8(S8)
9D000030  AFC2000C    SW V0, 12(S8)
```

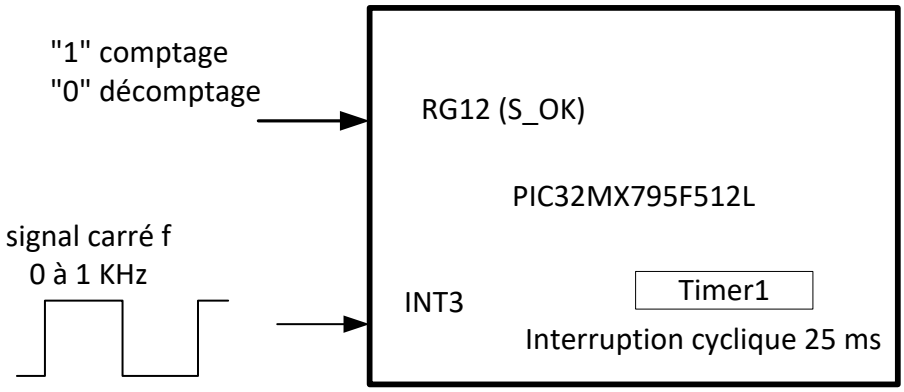
Adresses **Code machine**

Remarque : la documentation fournie ne permet pas d'obtenir le code machine, c'est en utilisant le désassemblage que l'on peut le connaitre.

Sous K:\ES\PROJETS\SLO\1102x_SK32MX775F512L\Data_sheets\PIC32 Family Reference Manual le fichier mips32v2_InstructionSet.pdf fournit la possibilité de reconstituer le code machine.

QUESTION 5

Voici le schéma de principe du système demandé :



Dans la réponse à l'interruption du Timer1, on inverse la LED0 pour permettre le contrôle de la période et on établit le statut de l'application à SERVICE_TASKS en utilisant la fonction APP_UpdateState.

Dans la réponse à l'interruption externe 3, on inverse la LED1 à chaque flanc montant pour contrôle. L'action principale consiste à incrémenter (si S_OK = 1) ou à décrémenter (si S_OK = 0) une variable globale 32 bits signée que l'on nommera countInt3.

Remarque : la variable countInt3 doit être déclarée dans l'application. Pour incrémenter il faut appeler la fonction APP_IncCountInt3 et pour décrémenter il faut appeler la fonction APP_DecCountInt3.

Niveau avancé (facultatif) :

Au lieu de prendre l'état de S_OK pour commander le comptage/décomptage, faire en sorte que chaque appui sur S_OK commute entre un mode d'incrémentation et autre mode de décrémentation.
Vous devrez alors réaliser un anti-rebond, ou vous aider de la librairie mc32debounce à disposition. Une variable booléenne indiquant si on est en mode d'incrémentation ou de décrémentation sera togglée à chaque appui.

QUESTIONS THÉORIQUES

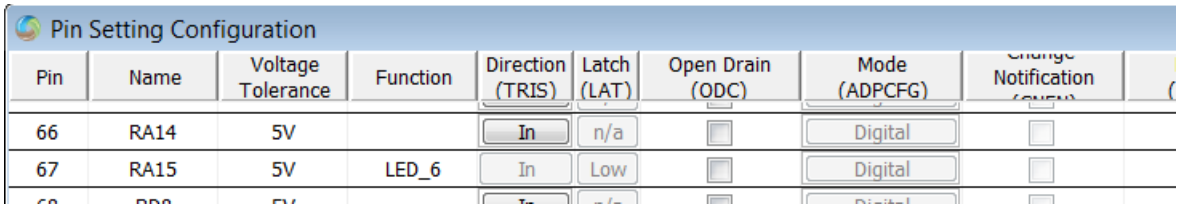
a) A quelles broches du PIC32MX795F512L (noms complet et No) correspondent les 4 éléments suivants :
LED_0 : TMS/**RA0** broche **17**
LED_1 : TCK/**RA0** broche **38**
INT3 : AETXCLK/SCL1/**INT3/RA14** broche **66** indication Eth_PWRDOWN/INT sur kit.
LED_6 : AETXEN/SDA1/**INT4**/RA15 broche **67**

b) Pour obtenir une interruption cyclique de 25 ms avec le timer1, quelle doit être sa période ainsi que le prescaler ?

Période du timer1 pour 25 ms : **31249**
Prescaler du timer1 : **64**
$NTICK_{PBCLK} = T_{COMPTAGE}/T_{PBCLK}$ = 25'000 us / 0,0125 us = 2'000'000
→ dépasse capacité timer 16 bits.
$PRESCALER_{MIN} = NTICK_{PBCLK} / NTICK_{MAX}$ = 2'000'000 / 65'536 = 30.52
→ 32 → 32 non disponible pour timer1 d'où 64.
D'où :
$NMAX= (T_{COMPTAGE}/T_{TIMERCLK})-1$ = (25'000 us / (0,0125 * 64))-1 = 31'249

c) Faut-il faire des modifications dans la configuration des E/S pour pouvoir fournir un signal sur INT3 ? Si oui nature de la modification, si non preuve.

Au niveau du *Pin Setting Configuration*, on constate que INT3/RA14 est bien en Input.



Donc il n'y a pas besoin de modification. Par contre si nous avions choisi INT4 nous serions en contradiction avec la configuration en sortie.

RÉALISATION PRATIQUE

Création d'un projet avec Harmony pour le kit PIC32MX

Utilisation du Timer1 avec interruption, prendre niveau 3.

Utilisation du bsp pic32mx_skes.

Ajout par le MHC de la configuration de l'interruption externe 3, prendre niveau 7. Configurer action au flanc montant.

MODIFICATION FICHIER SYSTEM_INIT.C

☝ Il est demandé de transformer le code généré dans system_init.c, en le plaçant dans une fonction à nommer **IntExt3_Initialize**.

MODIFICATION FICHIER SYSTEM_INTERRUPT.C

Compléter l'ISR du Timer1 avec l'inversion de la LED0. Utiliser la fonction BSP_LEDToggle. Ajouter l'appel à la fonction APP_UpdateState.

Créez l'ISR de INT3 avec l'inversion de la LED1. Utilisez la fonction BSP_LEDToggle.
Utilisez IPL7SRS.
Ajouter le comptage/décomptage de la variable countInt3.

MODIFICATION FICHIER APP.C

Effectuer l'initialisation de l'afficheur Lcd et afficher (section case INIT):
Ligne 1 : Ex Rev chap 2 a 6
Ligne 2 : Nom

Dans la section case SERVICE_TASK :
Affichez la valeur de la variable countInt3 sur la ligne 3.

Il faut aussi réaliser les 2 fonctions APP_IncCountInt3 et APP_DecCountInt3, ainsi que la classique fonction APP_UpdateState.

TEST DE FONCTIONNEMENT

Câbler un générateur sur l'entrée int3 en veillant à disposer d'un signal 0 à 3V3.

OBSERVATION

Mesurez à l'oscilloscope la période du Timer1.

Vérifiez que le signal sur Led1 soit de la moitié de la fréquence de celui fourni sur Int3.

En fournissant un signal de 100 Hz vérifiez l'évolution de l'affichage du compteur. En pressant sur S_OK vérifiez que le compteur diminue.

SOLUTION DE LA RÉALISATION PRATIQUE Q5

FONCTIONS DE CONFIGURATION APPELEES DANS SYSTEM_INIT.C

CONFIGURATION DU TIMER1

Voici la configuration du timer1 et de son interruption, que l'on trouve dans le fichier drv_tmr_static.c.

```
void DRV_TMR0_Initialize(void)
{
    /* Setup TMR0 Instance */
    PLIB_TMR_Stop(TMR_ID_1); /* Disable Timer */
    PLIB_TMR_ClockSourceSelect(TMR_ID_1,
                        TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK);
    PLIB_TMR_PrescaleSelect(TMR_ID_1, TMR_PRESCALE_VALUE_64);
    PLIB_TMR_Mode16BitEnable(TMR_ID_1);   /* 16 bit mode */
    PLIB_TMR_Counter16BitClear(TMR_ID_1); /* Clear counter */
    PLIB_TMR_Period16BitSet(TMR_ID_1, 31249); /*Set period */
    /* Setup Interrupt */
    PLIB_INT_SourceEnable(INT_ID_0, INT_SOURCE_TIMER_1);
    PLIB_INT_VectorPrioritySet(INT_ID_0, INT_VECTOR_T1,
                               INT_PRIORITY_LEVEL3);
    PLIB_INT_VectorSubPrioritySet(INT_ID_0, INT_VECTOR_T1,
                                  INT_SUBPRIORITY_LEVEL0);
}
```

CONFIGURATION DE INT EXT3

Voici la configuration de l'interruption externe 3. Cette fonction est réalisée manuellement. Il ne faut pas oublier de l'appeler dans la fonction SYS_Initialize().

```
void IntExt3_Initialize(void)
{
    // Etablit le flanc de Int3
    PLIB_INT_ExternalRisingEdgeSelect(INT_ID_0,
                                  INT_EXTERNAL_INT_SOURCE3);
    /* Setup Interrupt */
    PLIB_INT_SourceEnable(INT_ID_0, INT_SOURCE_EXTERNAL_3);
    PLIB_INT_VectorPrioritySet(INT_ID_0, INT_VECTOR_INT3,
                               INT_PRIORITY_LEVEL7);
    PLIB_INT_VectorSubPrioritySet(INT_ID_0,
                INT_VECTOR_INT3, INT_SUBPRIORITY_LEVEL0);
}
```

ISR DANS SYSTEM_INTERRUPT.C

ISR DU TIMER1

Voici la réponse à l'interruption du Timer1.

```
void __ISR(_TIMER_1_VECTOR, ipl3AUTO)
                              _IntHandlerDrvTmrInstance0(void)
{
    PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_TIMER_1);
    BSP_LEDToggle(BSP_LED_0);
    APP_UpdateState(APP_STATE_SERVICE_TASKS);
}
```

ISR DE INT EXT3

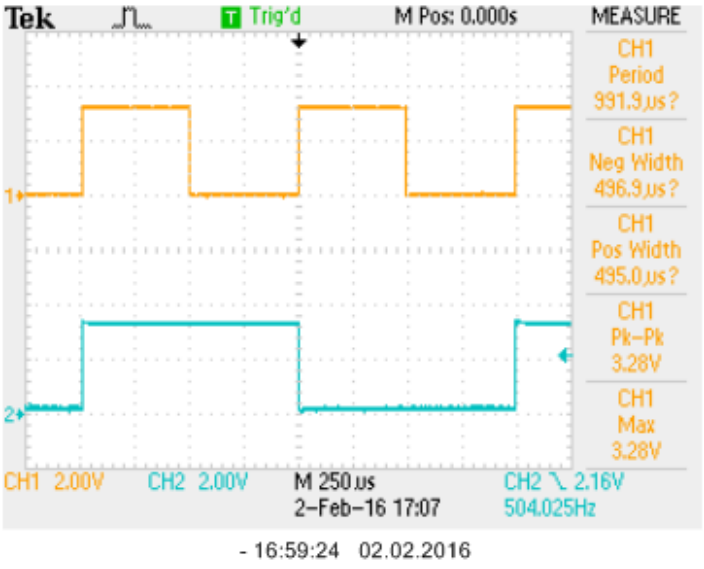
Voici la réponse à l'interruption externe 3. Le traitement de la touche OK est réalisé par la lecture directe d'un bit du port en utilisant la définition:

```
#define S_OK     PORTGbits.RG12

void __ISR(_EXTERNAL_3_VECTOR, IPL7SRS)
                                      IntHandlerIntExt3(void)
{
    BSP_LEDToggle(BSP_LED_1);
    PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_EXTERNAL_3);
    if (S_OK == 0){
        APP_DecCountInt3();
    } else {
        APP_IncCountInt3();
    }
}
```

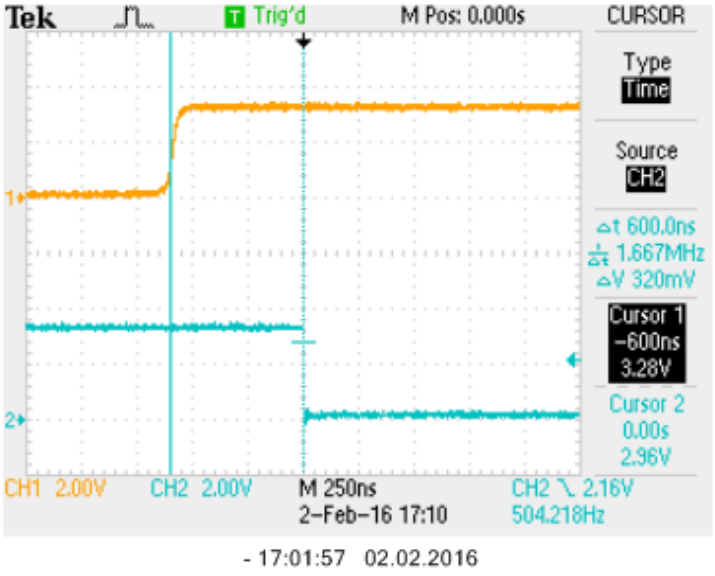
Voici une observation à l'oscilloscope montrant l'inversion de la LED_1 (canal 2) en réaction au signal sur l'entrée Int3Ext (canal 1), situation avec une fréquence de 1 KHz. Le compteur s'incrémente de 1000 à chaque seconde.

Canal1 : broche 66 INT3

Canal2 : LED_1



Voici une observation à l'oscilloscope montrant l'inversion de la LED_1 (canal 2) en réaction au signal sur l'entrée Int3Ext (canal 1), situation avec une fréquence de 1 KHz. Mesure du temps de réaction.

Canal1 : broche 66 INT3 Canal2 : LED_1	

On obtient 600 ns comme temps de réaction.

CONTENU APP.H

Voici le contenu de app.h sans les commentaires de Microchip.

```
#ifndef _APP_H
#define _APP_H

// Section: Included Files
#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdlib.h>
#include "system_config.h"
#include "system_definitions.h"

// Application's state machine's states.
typedef enum
{
   APP_STATE_INIT=0,
   APP_STATE_WAIT=1,
   APP_STATE_SERVICE_TASKS=2
} APP_STATES;

// Application Data
typedef struct
{
    APP_STATES state;
} APP_DATA;
```

```
// This routine must be called from the SYS_Initialize
function.
void APP_Initialize ( void );
//    This routine must be called from SYS_Tasks() routine.
void APP_Tasks ( void );

// Section: Application Callback Routines
void APP_UpdateState ( APP_STATES NewState ) ;
void APP_IncCountInt3  (void );
void APP_DecCountInt3  (void );
#endif /* _APP_H */
```

CONTENU APP.C

Voici un extrait du contenu de l'application :

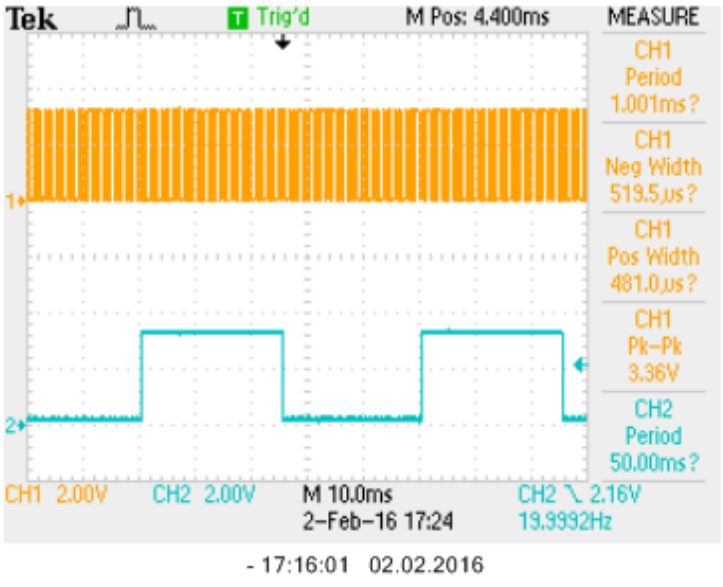
```
// variables globales
APP_DATA appData;
int32_t countInt3 = 0;

void APP_Initialize ( void )
{
    /* Place the App state machine in its initial state. */
    appData.state = APP_STATE_INIT;
}


void APP_Tasks ( void )
{
    /* Check the application's current state. */
    switch ( appData.state )
    {
        /* Application's initial state. */
        case APP_STATE_INIT:
        {
            lcd_init();
            lcd_bl_on();
            printf_lcd("SolQ5ExRev2a6      ");
            lcd_gotoxy(1,2);
            printf_lcd("C. Huber 02.02.2016");
            // Start le Timer
            DRV_TMR0_Start();
            appData.state = APP_STATE_WAIT;
            break;
        }

        case APP_STATE_WAIT :
          // nothing to do
        break;
```

```
        case APP_STATE_SERVICE_TASKS:
             lcd_gotoxy(1,3);
             printf_lcd("Count %010d", countInt3);
             appData.state = APP_STATE_WAIT;
         break;

        /* The default state should never be executed. */
        default:
        {
            break;
        }
    }
}

void APP_UpdateState ( APP_STATES NewState )
{
    appData.state = NewState;
}

void APP_IncCountInt3  ( void )
{
   countInt3 ++;
}

void APP_DecCountInt3  ( void )
{
   countInt3 --;
}
```

CONTROLE DE LA PERIODE DU TIMER1

Canal1 : broche 66 INT3

Canal2 : LED_0



Avec une période de 50 ms pour le canal 2, on a bien une inversion de la led_0 toutes les 25 ms.