SOLUTION EXERCICE 6_1 PIC32MX

OBJECTIFS

Cet exercice a pour objectif de permettre aux étudiants d'utiliser concrètement les timers et la génération de signaux PWM.

Après l'ébauche sur le papier, l'exercice sera réalisé pratiquement avec un kit PIC32MX795F512L, configuré pour SYS_CLK et PB_CLOCK = 80 MHz.

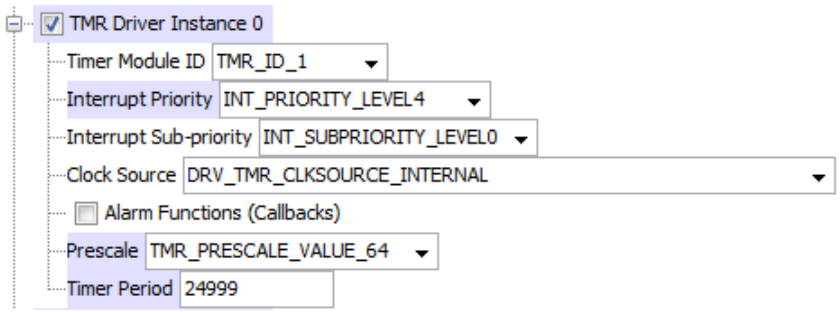
On souhaite réaliser les éléments suivants :

a) Générer une interruption cyclique toutes les ms en utilisant le CoreTimer. Niveau de priorité 5. L'action est un toggle de la LED0. Quelle est la valeur à fournir à la fonction OpenCoreTimer ?

Le CoreTimer utilise SYS_CLK / 2 comme horloge. $f_{CT} = f_{SYSCLK} / 2 = 80MHz / 2 = 40$ MHz.
Il faut effectuer le OpenCoreTimer avec :
$N_{MAX,CT} = T_{VOULU} / T_{CT} = 1000 / (0.0125 * 2) =$ **40'000**

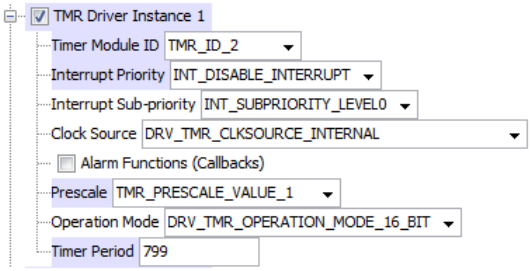
b) Générer une interruption cyclique toutes les 20 ms en utilisant le timer 1. Niveau de priorité 4. L'action est un toggle de la LED1.
Déterminez la valeur du prescaler et de la période.

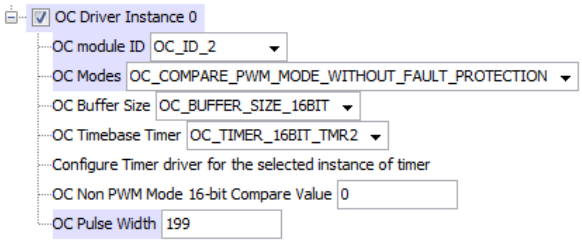
20'000 / 0.0125 = 1600000. Cette valeur dépasse largement 65535.
Le prescaler doit être > 1'600'000 / 65536 = 24.41 . La valeur supérieure existante la plus proche est **64**.
Valeur de la période :
$N_{MAX} = (T_{VOULU} / T_{TIMER1})-1 = (20'000 / (0.0125 * 64))-1 =$ **24999**



c) Configurer le timer 2 pour fournir une base de temps de 10 us. Avec OC2, générer un signal avec un rapport cyclique de 25%. Mode PWM.
Déterminez la valeur de la période pour le timer ainsi que la valeur de PulseWidth pour l'OC.

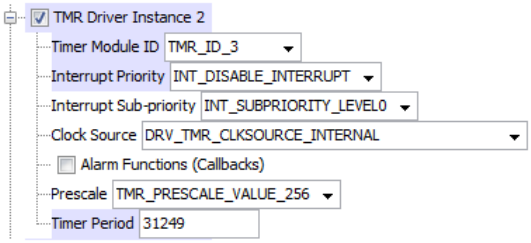
Periode Timer 2 :
$N_{MAX} = (T_{VOULU} / T_{TIMER2})-1 = (T_{VOULU} * f_{TIMER2})-1 = (10 * 80)-1 =$ **799** (prescaler de **1**)
Pulse Width OC2 :
$PW = (N_{CYCLES,TIMER} * r.c.)-1 = (800 * 0.25)-1 =$ **199**

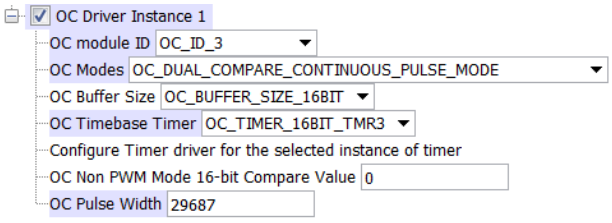




d) Configurer le timer 3 pour fournir une base de temps de 100 ms. Avec OC3, générer un signal avec un rapport cyclique de 95%. Mode CONTINUOUS PULSE.
Déterminez la valeur du prescaler et de la période pour le timer ainsi que la valeur de PulseWidth pour l'OC.

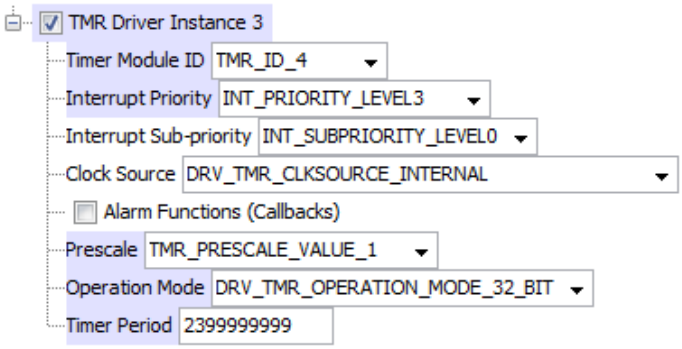
Période Timer 3 :
Le timer 3 doit compter (100'000 / 0.0125) = 8'000'000 cycles. Valeur trop grande pour 16 bits.
prescaler : 8'000'000 / 65536 = 122.07 => **256** car 128 n'est pas disponible.
$N_{MAX} = (T_{VOULU} / T_{TIMER3})-1 = (100'000 / (0.0125 * 256))-1 =$ **31249**
PulseWidth OC3 :
$PW = (N_{CYCLES,TIMER} * r.c.)-1 = (31250 * 0.95)-1 =$ **29687**





e) Générer une interruption cyclique toutes les 30 secondes en utilisant les timers 4 & 5. Niveau de priorité 3. L'action est un toggle de la LED4.
Déterminez la valeur du prescaler et de la période.

Période du Timer 4&5 :
$N_{MAX} = (T_{VOULU} / T_{TIMER45})-1 = (30'000'000 / 0.0125)-1 =$ **2'399'999'999**
$2^{32} =$ 4'294'967'296 donc un prescaler de **1** convient.



REALISATION PRATIQUE

- Création d'un projet avec Harmony pour le kit PIC32MX.
- Réaliser la configuration des timers et OC en utilisant les valeurs que vous avez déterminées.
- Réaliser manuellement la configuration du CoreTimer et de son ISR. Ajoutez les actions toggle dans les réponses aux interruptions des timers.
- Pour le CoreTimer, vous disposez des fonctions dans le BSP.

PROGRAMME A OBTENIR

CONTENU SYSTEM_INIT.C

INITIALISATION DU CORETIMER (PERIODE 1 MS)

Ajout de la fonction dans system_init.c et appel de la fonction.

```
void Init_CoreTimer(void)
{
   PLIB_INT_SourceEnable(INT_ID_0, INT_SOURCE_TIMER_CORE);
   PLIB_INT_VectorPrioritySet(INT_ID_0, INT_VECTOR_CT,
                              INT_PRIORITY_LEVEL5);
   PLIB_INT_VectorSubPrioritySet(INT_ID_0, INT_VECTOR_CT,
                                 INT_SUBPRIORITY_LEVEL0);

   OpenCoreTimer(40000);
}
```

Nécessaire d'ajouter :

```
#include "Mc32CoreTimer.h"
```

INITIALISATION DU TIMER1 (PERIODE 20 MS)

Généré à 100% sans modifications.

```
void DRV_TMR0_Initialize(void)
{
    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);
    PLIB_TMR_Counter16BitClear(TMR_ID_1);
    PLIB_TMR_Period16BitSet(TMR_ID_1, 24999);

    /* Setup Interrupt */
    PLIB_INT_VectorPrioritySet(INT_ID_0, INT_VECTOR_T1,
                               INT_PRIORITY_LEVEL4);
```

```
    PLIB_INT_VectorSubPrioritySet(INT_ID_0, INT_VECTOR_T1,
                                  INT_SUBPRIORITY_LEVEL0);
}
```

Avec la suite configuration interruption dans _DRV_TMR0_Resume.

```
static void _DRV_TMR0_Resume(bool resume)
{
    if (resume)
    {
        PLIB_INT_SourceFlagClear(INT_ID_0,
                                 INT_SOURCE_TIMER_1);
        PLIB_INT_SourceEnable(INT_ID_0,
                              INT_SOURCE_TIMER_1);
        PLIB_TMR_Start(TMR_ID_1);
    }
}
```

INITIALISATION DU TIMER2 ET DE OC2

Période 10 us et duty 25%. Généré à 100%, suppression autorisation interruption.

```
void DRV_TMR1_Initialize(void)
{
    PLIB_TMR_Stop(TMR_ID_2); /* Disable Timer */
    PLIB_TMR_ClockSourceSelect(TMR_ID_2,
                       TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK);
    PLIB_TMR_PrescaleSelect(TMR_ID_2,
                TMR_PRESCALE_VALUE_1);
    PLIB_TMR_Mode16BitEnable(TMR_ID_2);
    PLIB_TMR_Counter16BitClear(TMR_ID_2);
    PLIB_TMR_Period16BitSet(TMR_ID_2, 799); /*Set period */
    /* Setup Interrupt */
    PLIB_INT_VectorPrioritySet(INT_ID_0, INT_VECTOR_T2,
                               INT_PRIORITY_LEVEL1);
    PLIB_INT_VectorSubPrioritySet(INT_ID_0, INT_VECTOR_T2,
                                  INT_SUBPRIORITY_LEVEL0);
}
```

Mise en commentaire SourceEnable :

```
static void _DRV_TMR1_Resume(bool resume)
{
    if (resume)
    {
        PLIB_INT_SourceFlagClear(INT_ID_0,
                                 INT_SOURCE_TIMER_2);
        // PLIB_INT_SourceEnable(INT_ID_0,
        //                       INT_SOURCE_TIMER_2);
        PLIB_TMR_Start(TMR_ID_2);
    }
}
```

Généré à 100% , ajout PLIB_OC_Disable.

```
void DRV_OC0_Initialize(void)
{
    PLIB_OC_Disable(OC_ID_2);  // ajout CHR
    PLIB_OC_ModeSelect(OC_ID_2,
                       OC_COMPARE_PWM_EDGE_ALIGNED_MODE);
    PLIB_OC_BufferSizeSelect(OC_ID_2,
                             OC_BUFFER_SIZE_16BIT);
    PLIB_OC_TimerSelect(OC_ID_2, OC_TIMER_16BIT_TMR2);
    PLIB_OC_Buffer16BitSet(OC_ID_2, 0);
    PLIB_OC_PulseWidth16BitSet(OC_ID_2, 199); // 25%
}
```

INITIALISATION DU TIMER3 ET DE OC3

Période 100 ms et duty 95%. Généré à 100%, suppression autorisation interruption.

```
void DRV_TMR2_Initialize(void)
{
    PLIB_TMR_Stop(TMR_ID_3); /* Disable Timer */
    PLIB_TMR_ClockSourceSelect(TMR_ID_3,
                    TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK);
    PLIB_TMR_PrescaleSelect(TMR_ID_3,
                            TMR_PRESCALE_VALUE_256);
    PLIB_TMR_Mode16BitEnable(TMR_ID_3);
    PLIB_TMR_Counter16BitClear(TMR_ID_3);
    PLIB_TMR_Period16BitSet(TMR_ID_3, 31249);
    /* Setup Interrupt */
    PLIB_INT_VectorPrioritySet(INT_ID_0, INT_VECTOR_T3,
                               INT_PRIORITY_LEVEL1);
    PLIB_INT_VectorSubPrioritySet(INT_ID_0, INT_VECTOR_T3,
                                  INT_SUBPRIORITY_LEVEL0);
}
```

Mise en commentaire SourceEnable :

```
static void _DRV_TMR2_Resume(bool resume)
{
    if (resume)
    {
        PLIB_INT_SourceFlagClear(INT_ID_0,
                                 INT_SOURCE_TIMER_3);
        // PLIB_INT_SourceEnable(INT_ID_0,
        //                       INT_SOURCE_TIMER_3);
        PLIB_TMR_Start(TMR_ID_3);
    }
}
```

Généré à 100% , ajout PLIB_OC_Disable t.

```
void DRV_OC1_Initialize(void)
{
    PLIB_OC_Disable(OC_ID_3);  // ajout CHR
    PLIB_OC_ModeSelect(OC_ID_3,
                OC_DUAL_COMPARE_CONTINUOUS_PULSE_MODE);
    PLIB_OC_BufferSizeSelect(OC_ID_3,
                             OC_BUFFER_SIZE_16BIT);
    PLIB_OC_TimerSelect(OC_ID_3, OC_TIMER_16BIT_TMR3);
    PLIB_OC_Buffer16BitSet(OC_ID_3, 0);
    PLIB_OC_PulseWidth16BitSet(OC_ID_3, 29687);  // 95%
}
```

INITIALISATION DU TIMER 4&5 (PERIODE 30 S)

Généré à 100% sans modifications.

```
void DRV_TMR3_Initialize(void)
{
    PLIB_TMR_Stop(TMR_ID_4); /* Disable Timer */
    PLIB_TMR_ClockSourceSelect(TMR_ID_4,
                        TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK);
    PLIB_TMR_PrescaleSelect(TMR_ID_4,
                            TMR_PRESCALE_VALUE_1);
   /* Enable 32 bit mode */
    PLIB_TMR_Mode32BitEnable(TMR_ID_4);
    PLIB_TMR_Counter32BitClear(TMR_ID_4);
    PLIB_TMR_Period32BitSet(TMR_ID_4, 2399999999L);

    /* Setup Interrupt */
    PLIB_INT_VectorPrioritySet(INT_ID_0, INT_VECTOR_T5,
                               INT_PRIORITY_LEVEL3);
    PLIB_INT_VectorSubPrioritySet(INT_ID_0, INT_VECTOR_T5,
                                  INT_SUBPRIORITY_LEVEL0);
}

static void _DRV_TMR3_Resume(bool resume)
{
    if (resume)
    {
        PLIB_INT_SourceFlagClear(INT_ID_0,
                                 INT_SOURCE_TIMER_5);
        PLIB_INT_SourceEnable(INT_ID_0,
                              INT_SOURCE_TIMER_5);
        PLIB_TMR_Start(TMR_ID_4);
    }
}
```

☝ Configuration et start Timer4 (poids faible), mais interruption avec T5 (poids fort).

CONTENU SYSTEM_INTERRUPT.C

ISR CORETIMER (REALISATION MANUELLE)

```
// Réponse à l'interruption du Core Timer (cycle 1 ms)
void __ISR(_CORE_TIMER_VECTOR, ipl5SOFT)
                                    CoreTimerHandler(void)
{
    // clear the interrupt flag
    PLIB_INT_SourceFlagClear(INT_ID_0,
                             INT_SOURCE_TIMER_CORE);
    // Update the core timer (+1ms)
    UpdateCoreTimer(40000);
    // Inverse la Led 0
    BSP_LEDToggle(BSP_LED_0);
}
```

ISR TIMER1 (ADAPTATION CODE GENERE)

```
void __ISR(_TIMER_1_VECTOR, ipl4)
                         _IntHandlerDrvTmrInstance0(void)
{
    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_1);
     // Inverse la Led 1
    BSP_LEDToggle(BSP_LED_1);
}
```

ISR TIMER 4&5 (ADAPTATION CODE GENERE)

```
void __ISR(_TIMER_5_VECTOR, ipl3)
                          _IntHandlerDrvTmrInstance3(void)
{
    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_5);
    // Inverse la Led 4
    BSP_LEDToggle(BSP_LED_4);
}
```

☝ Les ISR pour le timers 2 et 3 ne sont pas présentées car elles ne sont pas utilisées !

CONTENU APP.C

Pas d'action dans l'application sauf les actions d'initialisation dans la fonction APP_Initialize().

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

    /* TODO: Initialize your application's state machine
             and other parameters.
    */
```

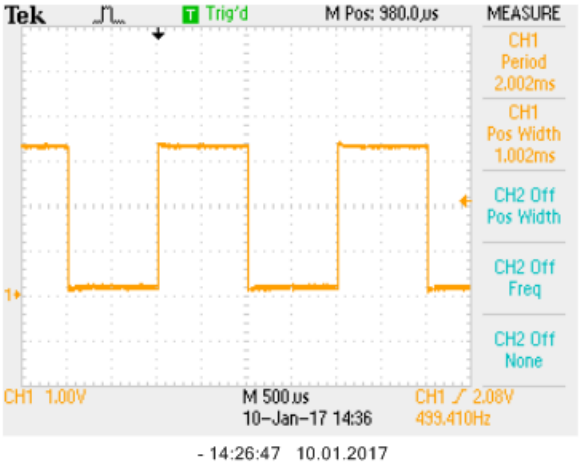
```
        lcd_init();
        lcd_bl_on();

        printf_lcd("SolEx6_1           ");
        lcd_gotoxy(1,2);
        printf_lcd("C. Huber 10.01.2017");

        // Start les Timer
        DRV_TMR0_Start();
        DRV_TMR1_Start();
        DRV_TMR2_Start();
        DRV_TMR3_Start();
        // Start les OC
        DRV_OC0_Start();
        DRV_OC1_Start();
}
```

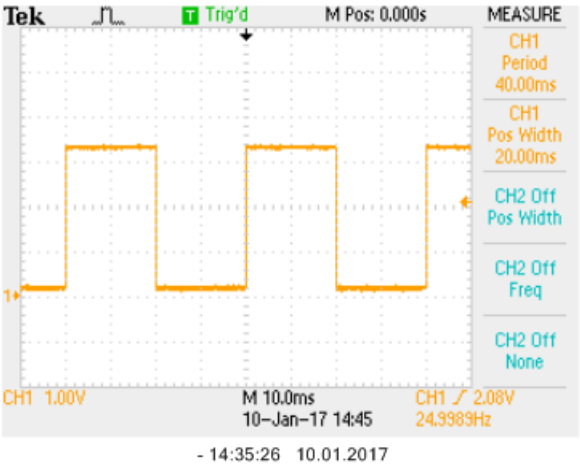
TEST DE FONCTIONNEMENT ET OBSERVATIONS

OBSERVATION INTERRUPTION 1 MS CORETIMER (LED0)



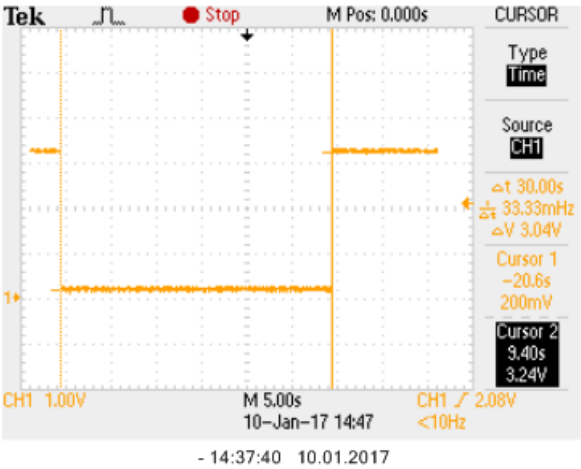
La periode de 2 ms confirme un cycle d'interruption à 1 ms.

OBSERVATION INTERRUPTION 20 MS TIMER1 (LED1)



La periode de 40 ms confirme un cycle d'interruption à 20 ms.

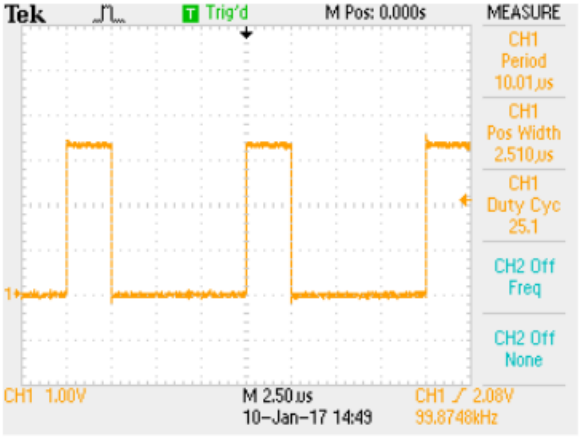
OBSERVATION INTERRUPTION 30 S TIMER4&5 (LED4)



On obtient bien les 30 secondes.

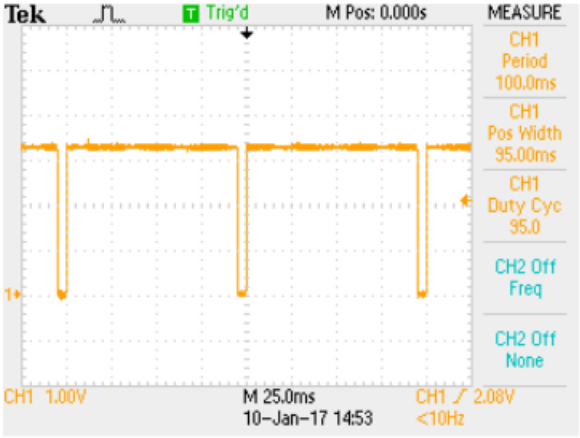
OBSERVATION SIGNAL OC2 PÉRIODE 10 US DUTY 25%

Observation sur la broche 76. Avec une période de 10.01 us, l'écart est de 0.1%. Le rapport cyclique est proche des 25%.



OBSERVATION SIGNAL OC3 PÉRIODE 100 MS DUTY 95%

Observation sur la broche 77. On obtient bien une période de 100 ms et le rapport cyclique est de 95%.



CONCLUSION

Les 3 interruptions fonctionnent et les 2 OC fournissent sans interventions et interruptions les signaux voulus.