

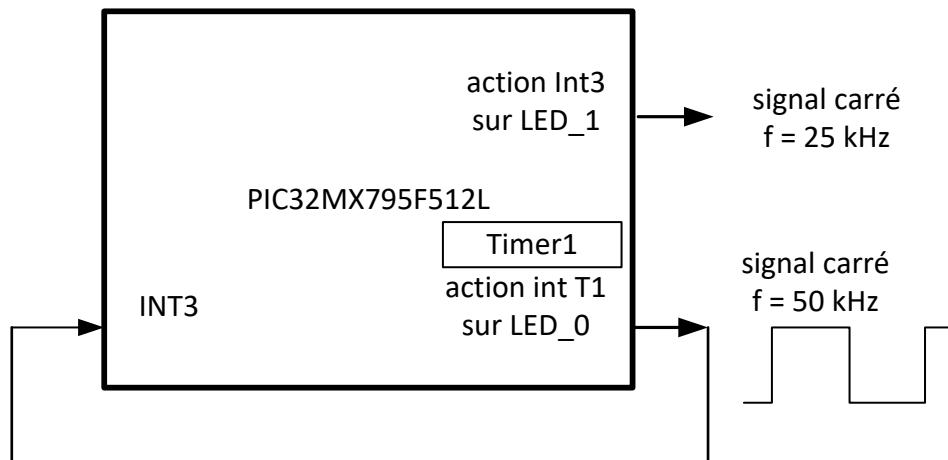
SOLUTION EXERCICE 5_1 PIC32

OBJECTIFS

Cet exercice a pour objectif de permettre aux étudiants de mettre en place des interruptions et de vérifier concrètement leur fonctionnement.

Après l'ébauche sur le papier, l'exercice sera réalisé pratiquement avec un kit PIC32MX795F512L.

Voici le schéma de principe du système demandé :



- Dans la réponse à l'interruption du timer 1, on inverse la LED0 de manière à obtenir un signal carré d'une fréquence de 50 kHz.
- Dans la réponse à l'interruption externe 3, on inverse la LED1 à chaque flanc montant, ce qui doit produire un signal carré d'une fréquence de 25 kHz.

a) A quelles broches du pic32MX795F512L (noms et No) correspondent les 3 éléments suivants :

LED_0 : TMS/RA0 broche 17

LED_1 : TCK/RA1 broche 38

INT3 : AETXCLK/SCL1/INT3/RA14 broche 66, indication Eth_PWRDOWN/INT sur kit

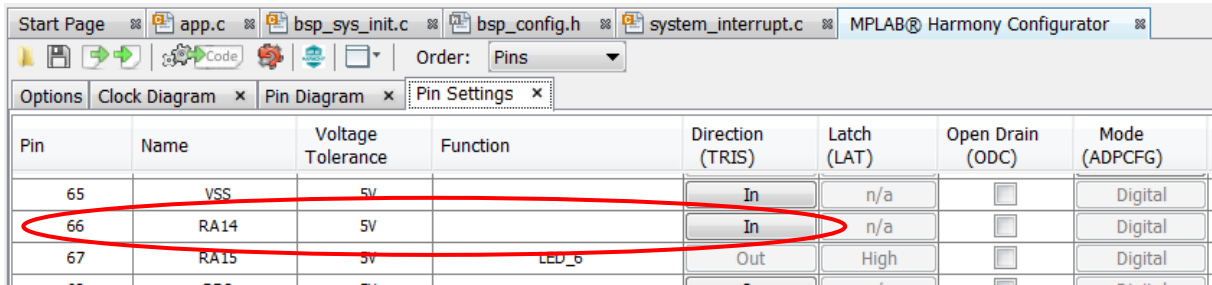
b) Pour obtenir un signal carré de 50 kHz en inversant LED0 à chaque interruption du timer 1, quelle doit être sa période? Déterminez également prescaler.

Période d'un signal carré 50 kHz : $1/50 \text{ kHz} = 0.02 \text{ ms}$ soit 20 μs

Période du timer 1 : $20 / 2 = 10 \text{ us}$ soit $10 * 80 = 800 \rightarrow 799$ avec un prescaler de 1.

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.

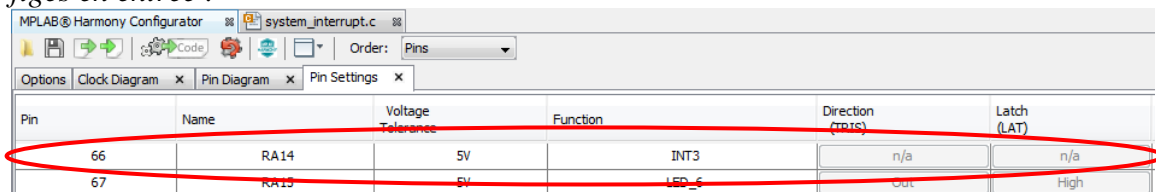
Il n'y a pas de modification à effectuer. Par défaut, on observe que RA14 est en entrée dans l'onglet "Pin Settings" du MHC :



The screenshot shows the MPLAB Harmony Configurator interface with the 'Pin Settings' tab selected. A red oval highlights the configuration for pin 66 (RA14), which is set to 'In' (Input) with a 'n/a' latch setting.

| Pin | Name | Voltage Tolerance | Function | Direction (TRIS) | Latch (LAT) | Open Drain (ODC) | Mode (ADPCFG) |
|-----|------|-------------------|----------|------------------|-------------|--------------------------|---------------|
| 65 | VSS | 5V | | In | n/a | <input type="checkbox"/> | Digital |
| 66 | RA14 | 5V | | In | n/a | <input type="checkbox"/> | Digital |
| 67 | RA15 | 5V | LED_6 | Out | High | <input type="checkbox"/> | Digital |

Si, de plus, on configure l'interruption externe dans le MHC, alors les réglages de la pin sont figés en entrée :



The screenshot shows the MPLAB Harmony Configurator interface with the 'Pin Settings' tab selected. A red oval highlights the configuration for pin 66 (RA14), which is set to 'In' (Input) with a 'n/a' latch setting and the 'INT3' function selected.

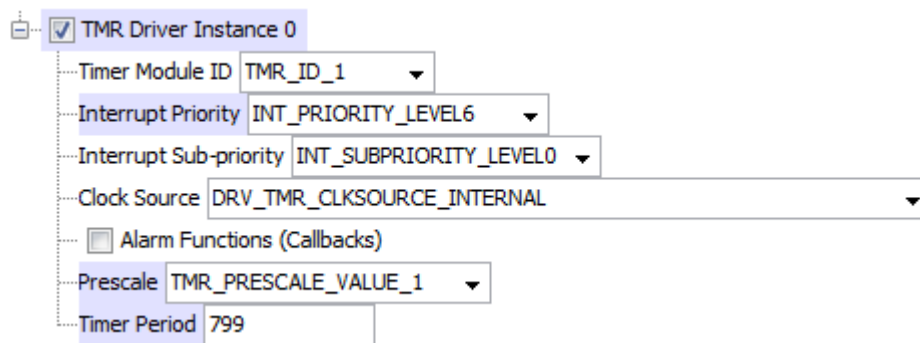
| Pin | Name | Voltage Tolerance | Function | Direction (TRIS) | Latch (LAT) |
|-----|------|-------------------|----------|------------------|-------------|
| 66 | RA14 | 5V | INT3 | In | n/a |
| 67 | RA15 | 5V | LED_6 | Out | High |

RÉALISATION PRATIQUE

- Création d'un projet avec Harmony pour le kit PIC32MX.
- Utilisation du timer 1 avec interruption, prendre niveau 6.
- Utilisation de l'interruption externe 3 au flanc montant, prendre niveau 7.
Dans le MHC, la configuration se trouve sous :
Harmony Framework Configuration > System Services > Interrupts > Use External Interrupts ?

CONFIGURATION DU MHC

Configuration du timer 1 :



Configuration de l'interruption externe 3 :

Use External Interrupts? ☒

Number of External Interrupt Instances

External Interrupt Instance 0 ☒

External Interrupt Module ID

Generate ISR Code? ☒

Interrupt Priority

Interrupt Sub-priority

Polarity

Enabled by System Service? ☒

SITUATION SYSTEM_INIT.C

Dans le code généré, on voit l'appel de `DRV_TMR0_Initialize`, ainsi que la configuration de l'interruption externe 3.

```
void SYS_Initialize ( void* data )
{
    .....

    /* Board Support Package Initialization */
    BSP_Initialize();

    /* Initialize Drivers */
    /*Initialize TMR0 */
    DRV_TMR0_Initialize();

    /* Initialize System Services */

    /*** Interrupt Service Initialization Code ***/
    SYS_INT_Initialize();

    /*Setup the INT_SOURCE_EXTERNAL_3 and Enable it*/
    SYS_INT_VectorPrioritySet(INT_VECTOR_INT3,
        INT_PRIORITY_LEVEL7);
    SYS_INT_VectorSubprioritySet(INT_VECTOR_INT3,
        INT_SUBPRIORITY_LEVEL0);
    SYS_INT_ExternalInterruptTriggerSet(
        INT_EXTERNAL_INT_SOURCE3,
        INT_EDGE_TRIGGER_RISING);
    SYS_INT_SourceEnable(INT_SOURCE_EXTERNAL_3);

    .....
}
```

MODIFICATION DU FICHIER SYSTEM_INTERRUPT.C

- Complétez l'ISR du timer 1 avec l'inversion de la LED0. Utilisez la fonction BSP_LEDToggle.
- Complétez l'ISR de INT3 avec l'inversion de la LED1. Utilisez la fonction BSP_LEDToggle. Configurez l'interruption de manière à utiliser le "Shadow Register Set" (attribut IPL7SRS au lieu de IPL7AUTO).

On obtient la situation suivante :

```
void __ISR(_TIMER_1_VECTOR, ipl6AUTO)
    _IntHandlerDrvTmrInstance0(void)
{
    BSP_LEDToggle(BSP_LED_0);
    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_1);
}

void __ISR(_EXTERNAL_3_VECTOR, ipl7SRS)
    IntHandlerIntExt3(void)
{
    BSP_LEDToggle(BSP_LED_1);
    PLIB_INT_SourceFlagClear(INT_ID_0,
                             INT_SOURCE_EXTERNAL_3);
}
```

COMPLÉMENT DE LA FONCTION APP_INITIALIZE DANS APP.C

Ajout de l'initialisation du LCD, affichage d'une message et démarrage du timer 1.

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

    //init LCD
    lcd_init();
    lcd_bl_on();
    printf_lcd("Solution Ex 5_1 ");
    lcd_gotoxy(1,2);
    printf_lcd("SCA 12.2017");

    DRV_TMR0_Start();
}
```

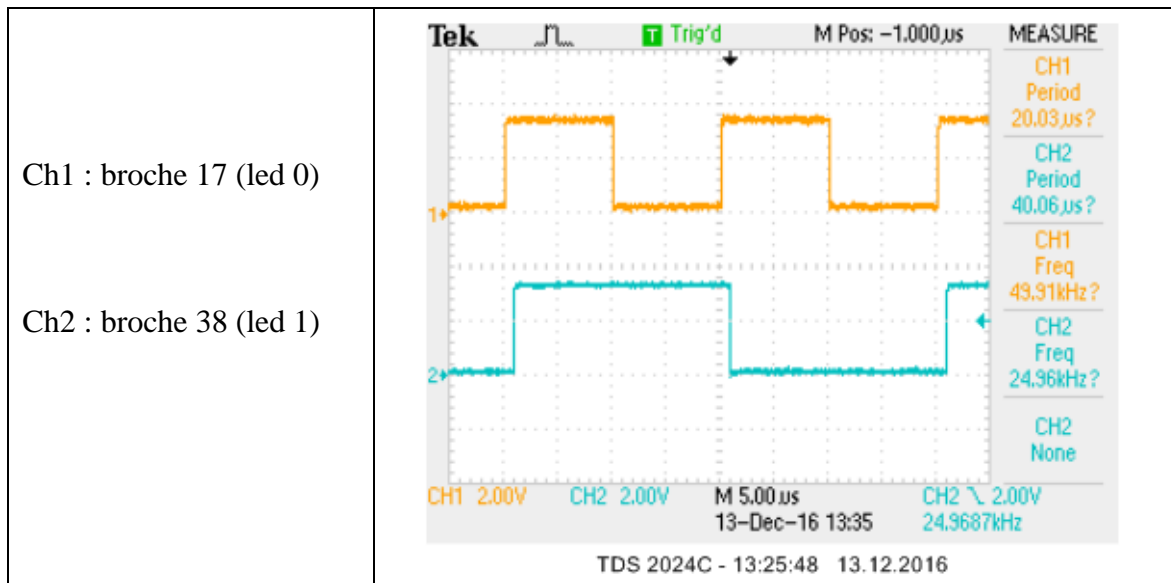
TEST DE FONCTIONNEMENT

- Réalisez le câblage entre la sortie correspondant à LED0 et INT3.
Il faut ponter la broche 17 (LED_0) avec la broche 66 (INT3)
- Vérifiez la présence des 2 signaux et leur fréquence.

OBSERVATIONS

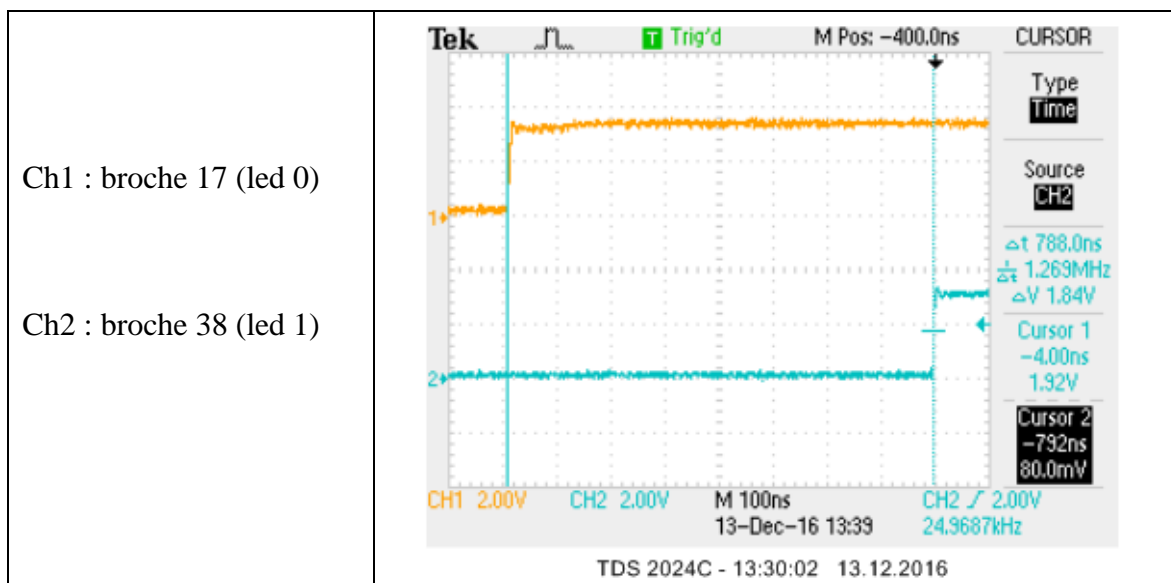
- Mesurez à l'oscilloscope le décalage entre le flanc montant du signal sur Int3 et le basculement de LED1.
- Observez en assembleur la réponse à l'int3, par rapport aux exemples du chapitre 5.
Y a-t-il davantage de sauvegardes ? Si oui, donnez les détails.

VUE D'ENSEMBLE



La relation entre les signaux correspond, la fréquence du canal 2 est moitié de celle du canal 1, soit ~25 kHz.

DÉCALAGE ENTRE LES SIGNAUX



On mesure un décalage de 788 ns entre le flanc montant de Led0 = Int3 et le flanc montant de Led1.

👉 Valeur assez élevée, mais niveau optimisation 0 et utilisation d'une fonction pour l'inversion de la Led.

ANALYSE DE LA RÉPONSE INT3 (ASSEMBLEUR)

```

80:                                void __ISR(_EXTERNAL_3_VECTOR, ipl7SRS)
81:                                IntHandlerIntExt3(void)
82:                                {
9D004C84  415DE800  RDPGPR SP, SP
9D004C88  401A7000  MFC0 K0, EPC
9D004C8C  401B6000  MFC0 K1, Status
9D004C90  27BDFFD8  ADDIU SP, SP, -40
9D004C94  AFBB0024  SW K1, 36(SP)
9D004C98  7C1B7844  INS K1, ZERO, 1, 15
9D004C9C  377B1C00  ORI K1, K1, 7168
9D004CA0  409B6000  MTC0 K1, Status
9D004CA4  00001012  MFLO V0
9D004CA8  AFA20014  SW V0, 20(SP)
9D004CAC  00001810  MFHI V1
9D004CB0  AFA30010  SW V1, 16(SP)
9D004CB4  03A0F021  ADDU S8, SP, ZERO
83:                                BSP_LEDToggle(BSP_LED_1);
9D004CB8  24040001  ADDIU A0, ZERO, 1
9D004CBC  0F4013B0  JAL BSP_LEDToggle
9D004CC0  00000000  NOP
84:                                PLIB_INT_SourceFlagClear(INT_ID_0,
85:                                INT_SOURCE_EXTERNAL_3);
9D004CC4  00002021  ADDU A0, ZERO, ZERO
9D004CC8  2405000F  ADDIU A1, ZERO, 15
9D004CCC  0F4015FD  JAL 0x9D0057F4
9D004CD0  00000000  NOP
86:                                }
9D004CD4  03C0E821  ADDU SP, S8, ZERO
9D004CD8  8FA20014  LW V0, 20(SP)
9D004CDC  00400013  MTLO V0
9D004CE0  8FA30010  LW V1, 16(SP)
9D004CE4  00600011  MTHI V1
9D004CE8  8FBB0024  LW K1, 36(SP)
9D004CEC  27BD0028  ADDIU SP, SP, 40
9D004CF0  41DDE800  WRPGPR SP, SP
9D004CF4  409B6000  MTC0 K1, Status
9D004CF8  42000018  ERET
    
```


DÉTAIL DE BSP_LEDTOGGLE

Pour expliquer le temps de réaction relativement long, il est nécessaire d'observer en assembleur la fonction **BSP_LEDToggle**.

```

378:                                void BSP_LEDToggle(BSP_LED led)
379:                                {
9D004EC0  27BDFFE8  ADDIU SP, SP, -24
9D004EC4  AFBF0014  SW RA, 20(SP)
9D004EC8  AFBE0010  SW S8, 16(SP)
9D004ECC  03A0F021  ADDU S8, SP, ZERO
9D004ED0  AFC40018  SW A0, 24(S8)
380:                                if (led == BSP_LED_7) {
9D004ED4  8FC30018  LW V1, 24(S8)
9D004ED8  2402000A  ADDIU V0, ZERO, 10
9D004EDC  14620008  BNE V1, V0, 0x9D004F00
9D004EE0  00000000  NOP
381:                                PLIB_PORTS_PinToggle(PORTS_ID_0,
                                                PORT_CHANNEL_B, led );
9D004EE4  00002021  ADDU A0, ZERO, ZERO
9D004EE8  24050001  ADDIU A1, ZERO, 1
9D004EEC  8FC60018  LW A2, 24(S8)
9D004EF0  0F401658  JAL 0x9D005960
9D004EF4  00000000  NOP
9D004EF8  0B4013C5  J 0x9D004F14
9D004EFC  00000000  NOP
382:                                } else {
383:                                PLIB_PORTS_PinToggle(PORTS_ID_0,
                                                PORT_CHANNEL_A, led );
9D004F00  00002021  ADDU A0, ZERO, ZERO
9D004F04  00002821  ADDU A1, ZERO, ZERO
9D004F08  8FC60018  LW A2, 24(S8)
9D004F0C  0F401658  JAL 0x9D005960
9D004F10  00000000  NOP
384:                                }
385:
386:                                }
9D004F14  03C0E821  ADDU SP, S8, ZERO
9D004F18  8FBF0014  LW RA, 20(SP)
9D004F1C  8FBE0010  LW S8, 16(SP)
9D004F20  27BD0018  ADDIU SP, SP, 24
9D004F24  03E00008  JR RA
9D004F28  00000000  NOP
    
```

On suppose que le temps de réaction plus grand (env. 300 ns) que dans l'exemple du chapitre 5 est sans doute dû à l'appel de la fonction **BSP_LEDToggle** qui entraîne la sauvegarde de V0 et V1 et des paramètres. Avec le niveau d'optimisation 0, les fonctions de la PLIB sont aussi appelées avec le JAL.

VARIANTE POUR RÉDUCTION DU TEMPS DE RÉACTION

Si on modifie la réponse à l'interruption externe 3 de la manière suivante :

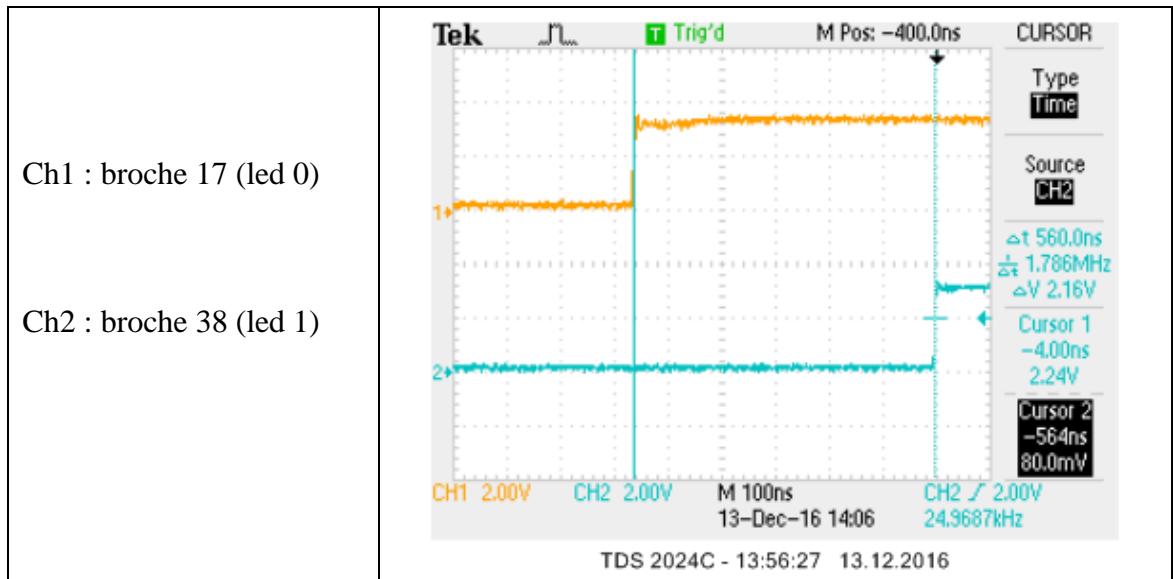
```
void __ISR(_EXTERNAL_3_VECTOR, ip17SRS)
    IntHandlerIntExt3(void)
{
    // BSP_LEDToggle(BSP_LED_1);
    LED1_W = !LED1_R;
    PLIB_INT_SourceFlagClear(INT_ID_0,
                             INT_SOURCE_EXTERNAL_3);
}
```

On obtient en assembleur :

```
80:                                void __ISR(_EXTERNAL_3_VECTOR, ip17SRS)
81:                                IntHandlerIntExt3(void)
82:                                {
9D0048D4  415DE800  RDPGPR SP, SP
9D0048D8  401A7000  MFC0 K0, EPC
9D0048DC  401B6000  MFC0 K1, Status
9D0048E0  27BDFFD8  ADDIU SP, SP, -40
9D0048E4  AFB00024  SW K1, 36(SP)
9D0048E8  7C1B7844  INS K1, ZERO, 1, 15
9D0048EC  377B1C00  ORI K1, K1, 7168
9D0048F0  409B6000  MTC0 K1, Status
9D0048F4  00001012  MFLO V0
9D0048F8  AFA20014  SW V0, 20(SP)
9D0048FC  00001810  MFHI V1
9D004900  AFA30010  SW V1, 16(SP)
9D004904  03A0F021  ADDU S8, SP, ZERO
83:                                // BSP_LEDToggle(BSP_LED_1);
84:                                LED1_W = !LED1_R;
9D004908  3C02BF88  LUI V0, -16504
9D00490C  8C426010  LW V0, 24592(V0)
9D004910  30420002  ANDI V0, V0, 2
9D004914  2C420001  SLTIU V0, V0, 1
9D004918  304400FF  ANDI A0, V0, 255
9D00491C  3C03BF88  LUI V1, -16504
9D004920  94626020  LHU V0, 24608(V1)
9D004924  7C820844  INS V0, A0, 1, 1
9D004928  A4626020  SH V0, 24608(V1)
85:                                PLIB_INT_SourceFlagClear(INT_ID_0,
86:                                INT_SOURCE_EXTERNAL_3);
9D00492C  00002021  ADDU A0, ZERO, ZERO
9D004930  2405000F  ADDIU A1, ZERO, 15
9D004934  0F401603  JAL 0x9D00580C
9D004938  00000000  NOP
87:                                }
9D00493C  03C0E821  ADDU SP, S8, ZERO
9D004940  8FA20014  LW V0, 20(SP)
9D004944  00400013  MTLO V0
```

```
9D004948 8FA30010 LW V1, 16(SP)
9D00494C 00600011 MTHI V1
9D004950 8FBB0024 LW K1, 36(SP)
9D004954 27BD0028 ADDIU SP, SP, 40
9D004958 41DDE800 WRPGR SP, SP
9D00495C 409B6000 MTC0 K1, Status
9D004960 42000018 ERET
```

Il n'y a pas de changement au niveau du stack. Par contre la manœuvre d'inversion se traduit par quelques instructions.



La durée du décalage est maintenant de 560 ns, donc un gain de $788 - 560 = 228$ ns.

Donc en optimisation 0, l'action BSP_LEDToggle est relativement lente.