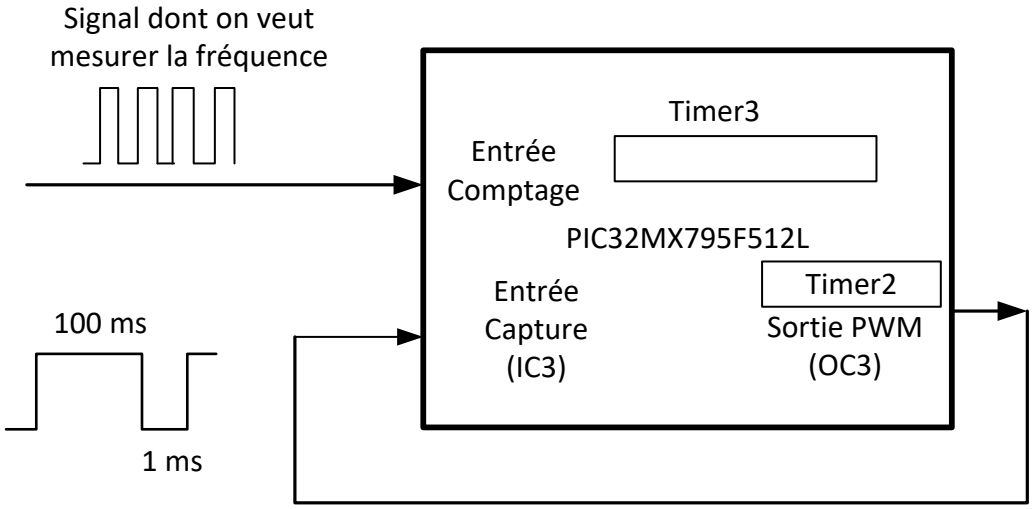
SOLUTION EXERCICE 6_2 PIC32MX

OBJECTIFS

Cet exercice a pour objectif de permettre aux étudiants d'utiliser concrètement les timers, captures et PWM.

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

On souhaite réaliser un fréquencemètre par comptage avec un PIC32MX795F512L configurer pour PB_FREQ = SYS_FREQ = 80 MHz. Utilisation du timer 3 comme compteur d'impulsions et d'une sortie PWM comme signal de base de temps pour le fréquencemètre. La base de temps du signal PWM est réalisée par le timer2.



Le signal utilisé pour le déclenchement de la capture doit provenir du module OC3 en mode PWM (base de temps timer2). Il doit être au niveau haut pendant 100 ms et au niveau bas seulement pendant 1 ms. La capture est effectuée avec le module IC3.

Il faut effectuer une capture au deux flancs et réagir à l'interruption de chaque flanc.

En utilisant le timer2 combiné au module OC3 et en capturant le timer 3 avec le module IC3, veuillez fournir les éléments suivants :

a) A quelles broches du pic32MX795F512L (noms et n°) correspondent les 3 éléments suivants :

Entrée de comptage :	**T3CK**/AC2TX/RC2	broche 7
Entrée de capture :	SCK1/**IC3**/PMCS2/PMA15/RD10	broche 70
Sortie du signal PWM :	**OC3**/RD2.....	broche 77

b) Déterminez quelles sont les fréquences min et max que l'on peut mesurer avec ce dispositif en partant du principe que le signal à mesurer est périodique et que son rapport cyclique est de 50%. En tenant compte que le timer 3 est configuré sans pré division.

Fréquence minimum : correspond au comptage d'une seule impulsion durant 100 ms, ce qui correspond à 10 impulsions pendant 1 sec donc 10 Hz.

Fréquence maximum : correspond à la capacité maximum du compteur 16 bits durant 100 ms, ce qui correspond à 65535 * 10 = 655'350 Hz.

c) Quelle est la précision (résolution) de mesure pour les fréquences suivantes :

10 Hz	résolution 10 Hz, précision de 100 %
100 Hz	résolution 10 Hz, précision de 10 %
1'000 Hz	résolution 10 Hz, précision de 1 %
10'000 Hz	résolution 10 Hz, précision de 0.1 %
100'000 Hz	résolution 10 Hz, précision de 0.01 %

d) Déterminez pour le timer2 la valeur de prescaler et de comparaison pour obtenir la période de 101 ms. Pour OC3, déterminez la valeur "PulseWidth" pour obtenir t_haut = 100 ms et t_bas 1 ms.

Recherche du diviseur :
On a $T_{COMPTAGE}$ = t_haut + t_bas
$NTICK_{PBCLK} = T_{COMPTAGE}/T_{PBCLK}$ = 101'000 us / 0.0125 us = 8'080'000
$PRESCALER_{MIN} = NTICK_{PBCLK} / NTICK_{MAX}$ = 8'080'000 / 65536 = 123.29
=> 128 qui n'existe pas donc 256.
Valeur comparaison = $(T_{COMPTAGE}/T_{TIMERCLK})$-1 = (101'000 us / (0.0125 * 256))-1 = 31'561.5
=> 31'562
Valeur PulseWidth = (t_haut / $T_{TIMERCLK}$)-1 = (100'000 us / (0.0125 * 256))-1 = 31'249

e) Quelle doit être la valeur de comparaison pour que le timer 3 compte jusqu'à son maximum ? Quelles sont les particularités de la configuration pour l'utilisation en comptage externe en vue de la capture.

Pour le timer 3 le maximum est 2^{16} -1 = 65535.

Il faut établir la synchronisation:

```
PLIB_TMR_ClockSourceExternalSyncEnable(TMR_ID_3);
```

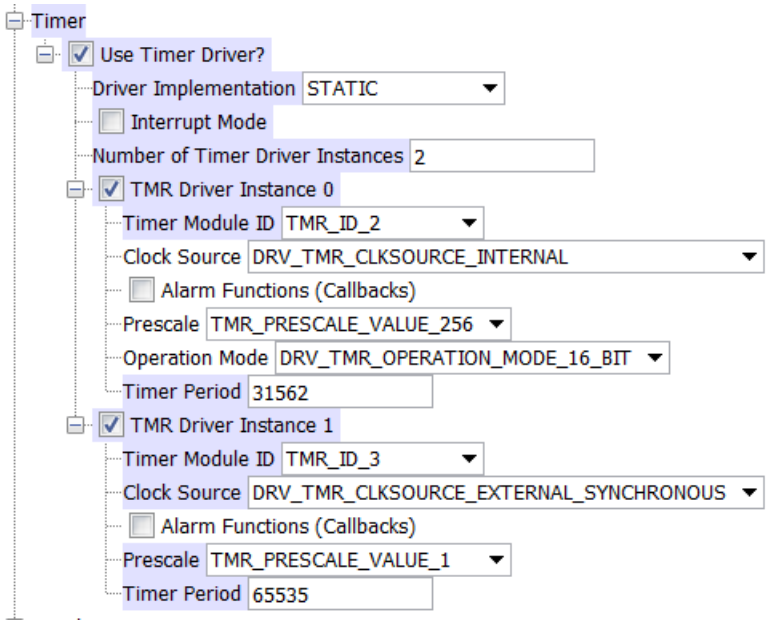
REALISATION PRATIQUE

Créez un projet nommé Ex6_2 en utilisant le MHC (vous pouvez utiliser une copie de l'exercice 6_1). Vous devez configurer les drivers suivants :

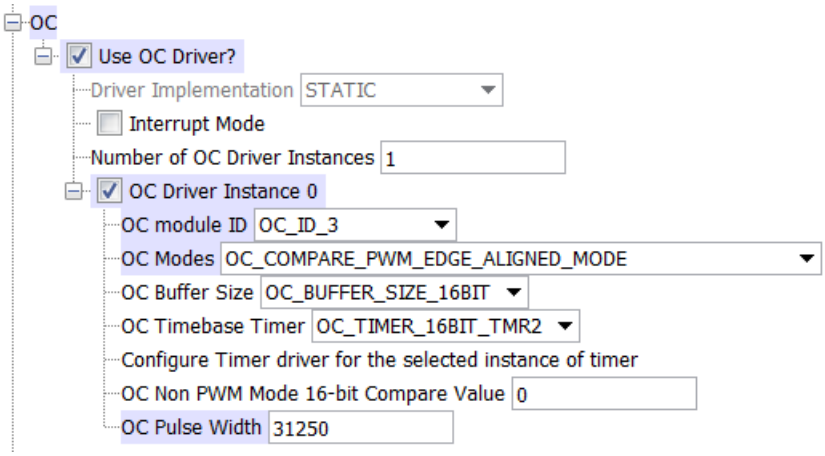
- Un driver timer statique sans interruption pour le timer 2, période 101 ms.
- Un driver timer statique sans interruption pour le timer 3, **en comptage externe**.
- Un driver OC statique pour fournir le signal demandé avec OC3.
- Un driver IC (avec IC3) avec interruption à chaque flanc, capture du timer 3, priorité 7.

  ☝ Il sera nécessaire de compléter la fonction DRV_IC0_Open() (fournie vide ! – état Harmony 2.06) : activation de l'IC, vidage du fifo des valeurs et activation interruption.

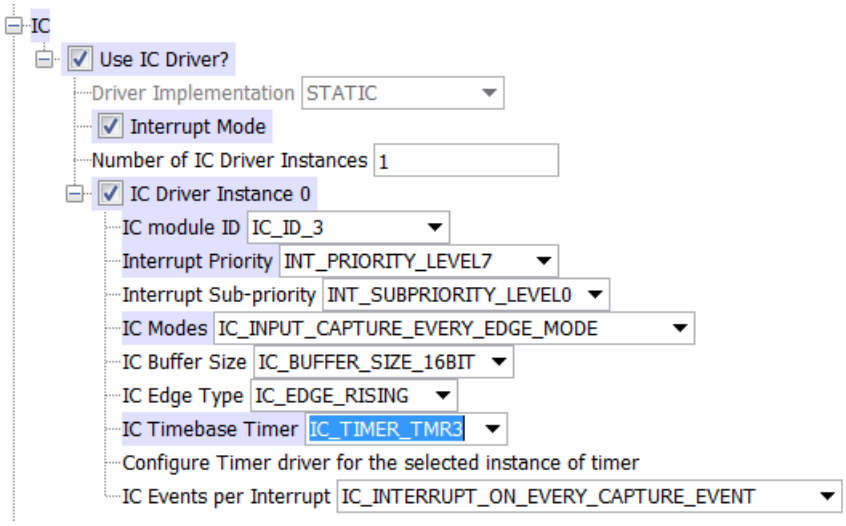
CONFIG DES 2 TIMERS



CONFIG OC3



CONFIG IC3



DETAILS DES CONFIGURATIONS OBTENUES ET COMPLETEE

CONFIGURATION DU TIMER2

Voici la fonction de configuration du timer2.

```
void DRV_TMR0_Initialize(void)
{
    /* Initialize Timer Instance0 */
    PLIB_TMR_Stop(TMR_ID_2); /* Disable Timer */
    /* Select clock source */
    PLIB_TMR_ClockSourceSelect(TMR_ID_2,
                  TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK);
    /* Select prescaler value */
    PLIB_TMR_PrescaleSelect(TMR_ID_2,
                            TMR_PRESCALE_VALUE_256);
    PLIB_TMR_Mode16BitEnable(TMR_ID_2); // 16 bit mode
    PLIB_TMR_Counter16BitClear(TMR_ID_2); // Clear counter
    PLIB_TMR_Period16BitSet(TMR_ID_2, 31562); //Set period
}
```

CONFIGURATION DE L'OC3

Voici la fonction de configuration de l'output compare 3, dont la base de temps est le timer2. L'appel de DRV_OC0_Start() sera effectué dans le case APP_STATE_INIT de l'application.

```
void DRV_OC0_Initialize(void)
{
    /* Setup OC0 Instance */
    PLIB_OC_Disable(OC_ID_3);  // Ajout CHR
    PLIB_OC_ModeSelect(OC_ID_3,
                       OC_COMPARE_PWM_EDGE_ALIGNED_MODE);
    PLIB_OC_BufferSizeSelect(OC_ID_3,
                             OC_BUFFER_SIZE_16BIT);
    PLIB_OC_TimerSelect(OC_ID_3, OC_TIMER_16BIT_TMR2);
    PLIB_OC_Buffer16BitSet(OC_ID_3, 0);
    PLIB_OC_PulseWidth16BitSet(OC_ID_3, 31249);
}
```

CONFIGURATION DU TIMER 3

Voici la fonction de configuration du timer 3. Il faut ajouter **ClockSourceExternalSyncEnable** pour que la capture fonctionne correctement avec le timer en comptage externe.

```
void DRV_TMR1_Initialize(void)
{
    /* Initialize Timer Instance0 */
    PLIB_TMR_Stop(TMR_ID_3); /* Disable Timer */
    /* Select clock source */
    PLIB_TMR_ClockSourceSelect(TMR_ID_3,
                       TMR_CLOCK_SOURCE_EXTERNAL_INPUT_PIN);
    /* Select prescaler value */
    PLIB_TMR_PrescaleSelect(TMR_ID_3,
                            TMR_PRESCALE_VALUE_1);
    PLIB_TMR_Mode16BitEnable(TMR_ID_3); // 16 bit mode
    PLIB_TMR_Counter16BitClear(TMR_ID_3); // Clear counter
    PLIB_TMR_Period16BitSet(TMR_ID_3, 65535); //Set period
    // Ajout pour la capture
    PLIB_TMR_ClockSourceExternalSyncEnable(TMR_ID_3);
    TRISCbits.TRISC2 = 1;  // T3CK en entrée
}
```

☝ Il reste un point à clarifier : la configuration du timer en comptage externe configure-t-elle la broche T3CK en entrée ?

Pour vérifier cela, il faut mettre en output avant la configuration et observer la valeur du bit TRISC2 après la configuration. Avec :

```
bool ValTrisC2;
uint32_t valTRISC;

void DRV_TMR1_Initialize(void)
{
```

```
// Test action sur entrée de comptage T3CK/AC2TX/RC2
TRISCbits.TRISC2 = 0;
/* Initialize Timer Instance1 */
```

```
6   // Test action sur entrée de comptage T3CK/AC2TX/RC2
7   TRISCbits.TRISC2 = 0;
8   /* Initialize Timer Instance1 */
9   /* Disable Timer */
0   PLIB_TMR_Stop(TMR_ID_3);
1   /* Select clock source */
2   PLIB_TMR_ClockSourceSelect(TMR_ID_3, TMR_CLOCK_SOURCE_EXTERNAL_INPUT_P
3   /* Select prescalar value */
4   PLIB_TMR_PrescaleSelect(TMR_ID_3, TMR_PRESCALE_VALUE_1);
5   /* Enable 16 bit mode */
6   PLIB_TMR_Mode16BitEnable(TMR_ID_3);
7   /* Clea
8   PLIB_TM
9   /*Set p
0   PLIB_TM
1   // Ajou
2   PLIB_TM
3
4   ValTris
    valTRISC = TRISC;
```

```
[TRISC]      15      14      13      12          4       3       2
       TRISC15 TRISC14 TRISC13 TRISC12 - TRISC4 TRISC3 TRISC2
             1       1       1       1 -       1       1       0

                     1
             TRISC1 -
                     0 -
```

On constate que TRISC2 = 0 donc output. La variable de type bool ValTrisc2 vaut bien aussi 0 `Address = 0xA0000224,  ValTrisC2 = 0x00`

☝ Il est donc nécessaire d'établir TRISC2 à 1 si dans l'initialisation du BSP cette broche est configurée en sortie.

Au niveau du Pin Setting :

Pin Setting Configuration

Pin	Name	Voltage Tolerance	Function	Direction (TRIS)	Latch (LAT)	Open Drain (ODC)	Mode (ADPCFG)	Change Notification	Pull Up (CNPUE)	Pull Down (CNPD)
1	RG15	5V	S_MINUS	In	Low		Digital			
2	VDD	5V		In	n/a		Digital			
3	RE5	5V	LCD_DB5	In	Low		Digital			
4	RE6	5V	LCD_DB6	In	Low		Digital			
5	RE7	5V	LCD_DB7	In	Low		Digital			
6	RC1	5V	BIN1_HBRID...	In	Low		Digital			
7	RC2	5V	T3CK	n/a	n/a		Digital			
8	RC3	5V	SD_DETECT	In	Low		Digital			
9	RC4	5V		In	n/a		Digital			
10	RG6	5V	LIGNE1	In	Low		Digital			
11	RG7	5V	LIGNE2	In	Low		Digital			

La direction n'est pas clairement établie, d'où la nécessité d'effectuer cette configuration (a été ajouté à l'initialisation du timer 3).

CONFIGURATION DE L'IC3

Voici la fonction de configuration de l'input capture 3, capturant le timer 3. L'enable sera effectué dans le case APP_STATE_INIT de l'application ainsi que l'autorisation de la source d'interruption, ceci en appelant la fonction **DRV_IC0_Open** qu'il est nécessaire de compléter.

A cause de la direction non établie, on ajoute la configuration en input.


```
void DRV_IC0_Initialize(void)
{
    PLIB_IC_Disable(IC_ID_3); // Ajout CHR
    TRISDbits.TRISD10 = 1 ;   // input

    /* Setup IC0 Instance */
    PLIB_IC_ModeSelect(IC_ID_3,
                       IC_INPUT_CAPTURE_EVERY_EDGE_MODE);
    PLIB_IC_FirstCaptureEdgeSelect(IC_ID_3,
                                   IC_EDGE_RISING);
    PLIB_IC_TimerSelect(IC_ID_3, IC_TIMER_TMR3);
    PLIB_IC_BufferSizeSelect(IC_ID_3,
                             IC_BUFFER_SIZE_16BIT);
    PLIB_IC_EventsPerInterruptSelect(IC_ID_3,
                     IC_INTERRUPT_ON_EVERY_CAPTURE_EVENT);

    /* Setup Interrupt */
   //        PLIB_INT_SourceEnable(INT_ID_0,
                                INT_SOURCE_INPUT_CAPTURE_3);
    PLIB_INT_VectorPrioritySet(INT_ID_0, INT_VECTOR_IC3,
                               INT_PRIORITY_LEVEL7);
    PLIB_INT_VectorSubPrioritySet(INT_ID_0, INT_VECTOR_IC3,
                                  INT_SUBPRIORITY_LEVEL0);
}
```

COMPLEMENT DU DRIVER IC

Voici le complément de la fonction **DRV_IC0_Open** qui est fournie vide par le driver.

```
// Modif CHR
void DRV_IC0_Open(void)
{
    PLIB_IC_Enable(IC_ID_3);
    while (!PLIB_IC_BufferIsEmpty(IC_ID_3))
    {
        PLIB_IC_Buffer16BitGet(IC_ID_3);
    }
    PLIB_INT_SourceFlagClear(INT_ID_0,
                             INT_SOURCE_INPUT_CAPTURE_3);
    PLIB_INT_SourceEnable(INT_ID_0,
                          INT_SOURCE_INPUT_CAPTURE_3);
}
```

☝ Il faut ajouter le prototype de la fonction dans le fichier drv_ic_static.h

ACTIONS DANS APP.C

Au niveau de l'application dans le case APP_STATE_INIT, il faut :

- lancer les timers,
- lancer l'OC,
- démarrer la capture et autoriser son interruption.

Il faut réaliser l'habituelle fonction APP_UpdateState et aussi APP_UpdateFreq qui permet de mettre à jour la variable globale uint32_t FrequenceSignal.

Dans le case APP_STATE_SERVICE_TASKS, il faut effectuer l'affichage de la fréquence.

SECTION CASE APP_STATE_INIT

Dans cette section, on trouve l'initialisation du LCD, un affichage initial et bien sur le start des driver TMR et OC. Utilisation du DRV_IC0_Open pour effectuer l'enable de IC3, la purge du buffer de capture, la mise à zéro du flag de l'interruptions de capture et finalement l'autorisation de l'interruption de capture.

```
case APP_STATE_INIT:
    lcd_init();// Init du LCD
    lcd_bl_on();

    printf_lcd("Solution Ex6_2 ");
    lcd_gotoxy(1,2);
    printf_lcd("C. Huber 24.01.2017");

    // Start les Timer
    DRV_TMR0_Start();
    DRV_TMR1_Start();
    // Start OC
    DRV_OC0_Start();
    // Action spéciale pour IC
    DRV_IC0_Open();

    appData.state = APP_STATE_WAIT;
break;
```

SECTION CASE APP_STATE_SERVICE_TASKS

Dans cette section, on trouve l'affichage de la fréquence mesurée. Avec la déclaration globale de :

```
APP_DATA appData;
uint32_t FrequenceSignal = 12345;

case APP_STATE_SERVICE_TASKS:
    lcd_gotoxy(1,3);
    printf_lcd("Freq %06d [Hz]", FrequenceSignal);

    BSP_LEDToggle(BSP_LED_2);
    appData.state = APP_STATE_WAIT;
break;
```

FONCTION APP_UPDATESTATE

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

FONCTION APP_UPDATEFREQ

```
void APP_UpdateFreq ( uint32_t NewFreq )
{
   FrequenceSignal = NewFreq;
}
```

ACTIONS DANS L'ISR DE LA CAPTURE

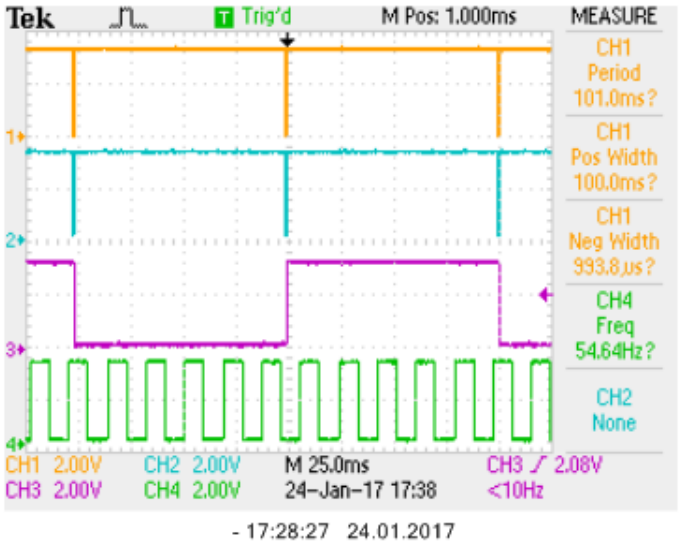
L'ISR générée est à corriger et à compléter. Lors de l'interruption au flanc descendant de la capture, il faut effectuer le calcul de la fréquence que l'on exprimera en Hz. Réalisation des calculs avec une variable locale, appels des fonctions APP_UpdateFreq et APP_UpdateState.

☝ include nécessaire :

```
#include "peripheral/ic/plib_ic.h"

void __ISR(_INPUT_CAPTURE_3_VECTOR, IPL7AUTO)
                          _IntHandlerDrvICInstance0 (void)
{
    uint16_t Capt3Falling;
    static uint16_t Capt3Rising;
    uint16_t FreqPeriodeTic;

    // IC3 correspond à RD10
    if (PORTDbits.RD10 == 1) {
        // Obtient et mémorise capture du flanc montant
        Capt3Rising = PLIB_IC_Buffer16BitGet(IC_ID_3);
        PLIB_INT_SourceFlagClear(INT_ID_0,
                                 INT_SOURCE_INPUT_CAPTURE_3);
        BSP_LEDToggle(BSP_LED_1);
    } else {
         // Obtient capture du flanc descendant
         Capt3Falling = PLIB_IC_Buffer16BitGet(IC_ID_3);
         // Calcul du nb impulsions comptées
        FreqPeriodeTic = Capt3Falling - Capt3Rising;
         // Fournit valeur à l'application
        APP_UpdateFreq(FreqPeriodeTic * 10);
        APP_UpdateState(APP_STATE_SERVICE_TASKS);

        PLIB_INT_SourceFlagClear(INT_ID_0,
                                 INT_SOURCE_INPUT_CAPTURE_3);
        BSP_LEDToggle(BSP_LED_1);
    }
}
```

Comme la durée d'échantillonnage est de 100 ms, la fréquence réelle est 10 fois plus élevée.

EXEMPLES DE RESULTATS

CH1 signal OC3, broche 77) CH2 signal LED1 (image int. IC2) CH3 signal LED2 (toggle dans application) CH4 signal à mesurer	

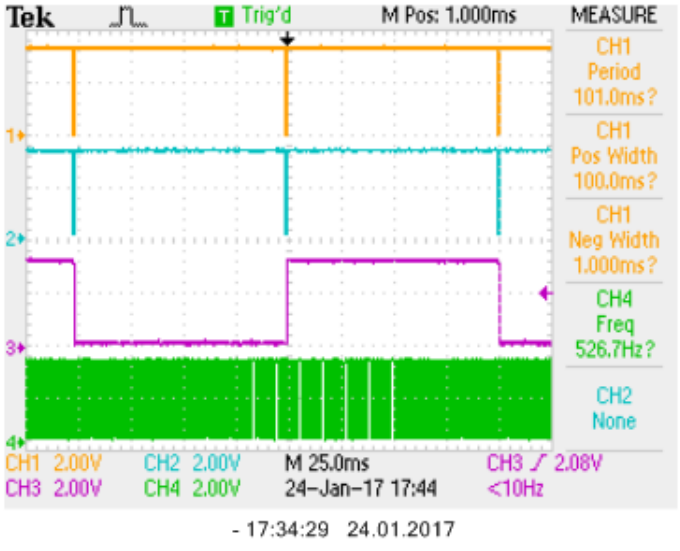
Avec les mesures sur CH1, on vérifie le signal généré par OC3 (période 101 ms, t_haut 100 ms et t_bas 1 ms.

Le signal sur LED1 nous montre que l'interruption de capture suit le signal en OC2.

Le signal sur LED2 nous indique à chaque flanc que l'application a été déclenchée pour réaliser l'affichage.

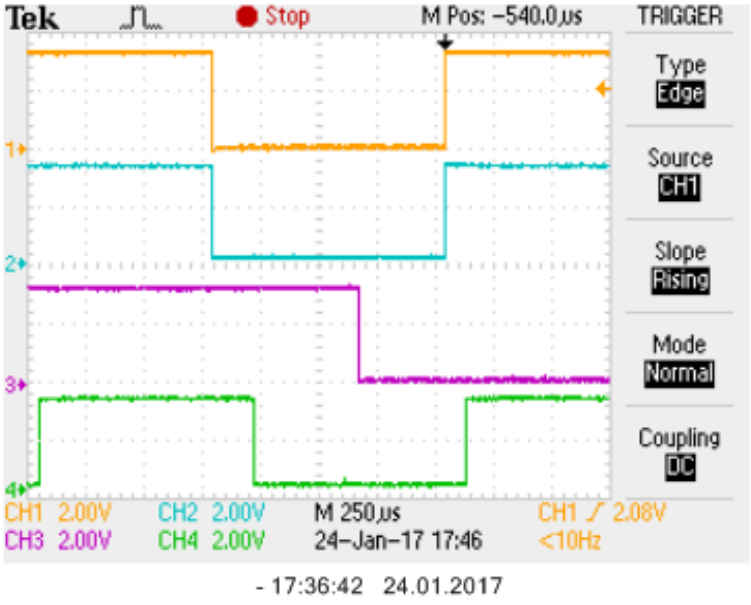
Avec le signal à mesurer d'environ 55 Hz, on obtiendra un comptage de 5 ou 6 impulsions, donc un affichage qui passe de 50 à 60 Hz.

SITUATION AVEC SIGNAL 545 HZ

CH1 signal OC3, broche 77) CH2 signal LED1 (image int. IC2) CH3 signal LED2 (toggle dans application) CH4 signal à mesurer	

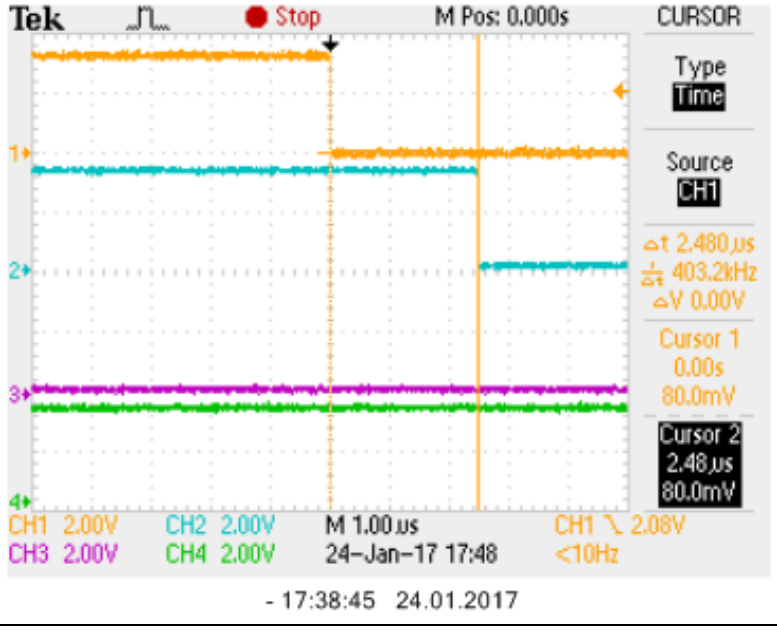
Avec le signal à mesurer de 545 Hz, on obtiendra un comptage de 54 ou 55 impulsions, donc un affichage qui passe de 540 à 550 Hz.

DETAIL SIGNAL OC3 ET LED1

CH1 signal OC3, broche 77) CH2 signal LED1 (image int. IC2) CH3 signal LED2 (toggle dans application) CH4 signal à mesurer	

On voit bien l'inversion du canal 2 à chaque flanc du canal 1 (sortie OC3).

Un changement de base de temps nous montre qu'il y a un temps de retard d'environ 2 us entre les 2 signaux.

CH1 signal OC3, broche 77) CH2 signal LED1 (image int. IC2) CH3 signal LED2 (toggle dans application) CH4 signal à mesurer	

Le temps de retard mesuré est 2,48 us. Cela correspond au temps de réaction de l'interruption et du traitement jusqu'a l'action sur la led (niveau optimisation 0).

RELATION FREQUENCE ET AFFICHAGE

f_{gen} = 2'225 Hz	f_{ocillo} = 2.22501 kHz	$f_{affichée}$ = 2'220 et 2'230
f_{gen} = 22'225 Hz	f_{ocillo} = 22.22501 kHz	$f_{affichée}$ = 22'220 et 22'230
f_{gen} = 222'225 Hz	f_{ocillo} = 222.226 kHz	$f_{affichée}$ = 222'220 et 222'230