Mécanisme anti-rebond

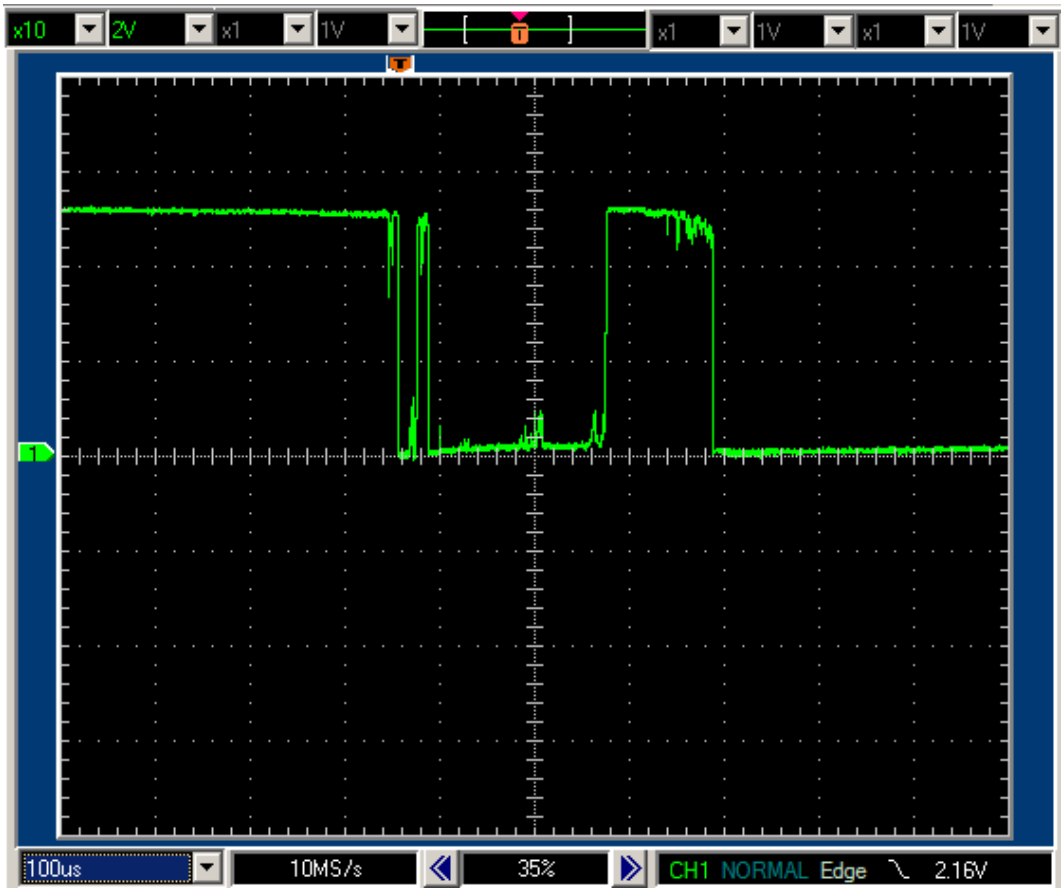
Dans ce complément, nous allons aborder le principe de la gestion des rebonds sous l'aspect théorique ainsi que sur la base d'une réalisation appliquant la théorie.

Les mécanismes anti-rebond s'appuient sur une lecture périodique du signal. Il est donc important de mesurer la durée d'instabilité pour choisir une période d'échantillonnage.

REBOND D'UNE TOUCHE

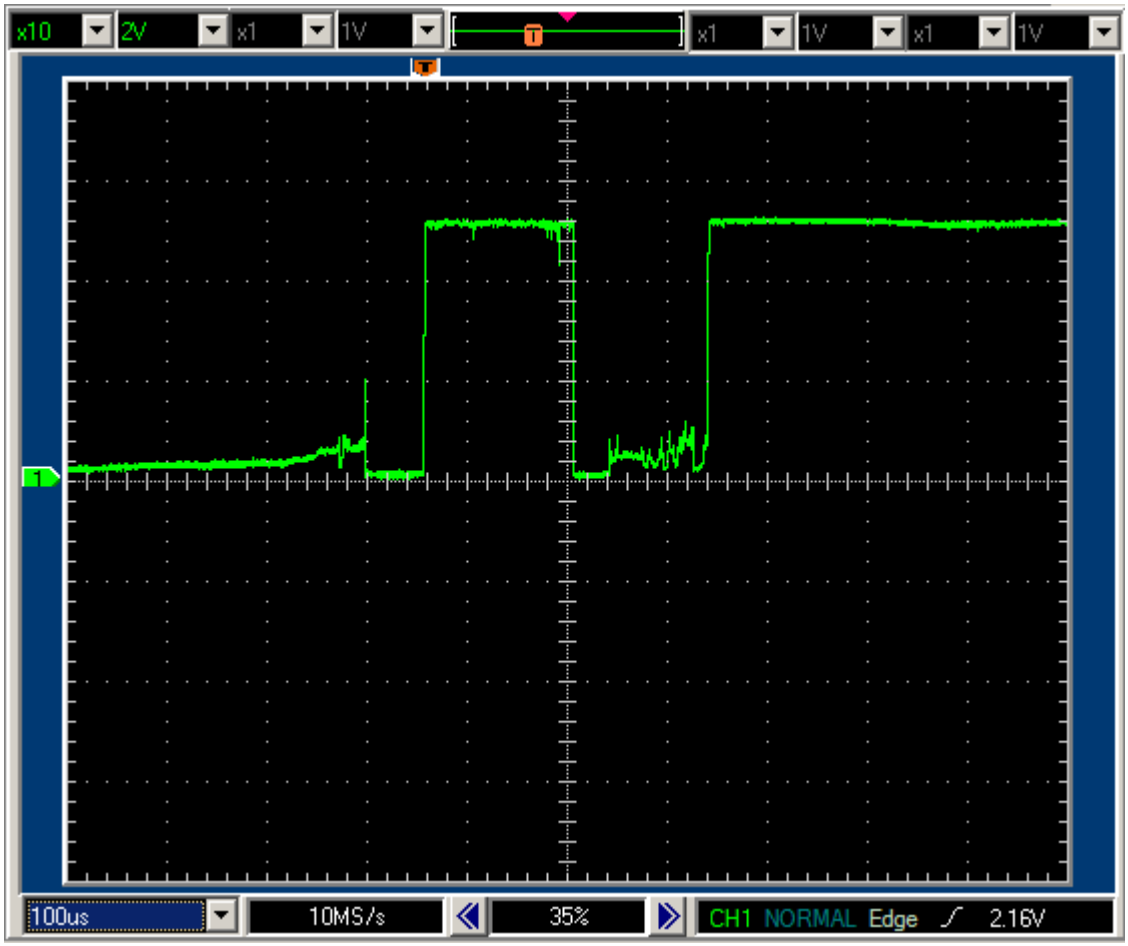
Avant de s'intéresser au mécanisme permettant d'éliminer les rebonds il est important d'avoir une idée de l'importance des rebonds. Voici 2 mesures réalisées avec une des touches du kit PIC32.

REBOND LORS DE L'APPUI



On constate que la zone d'instabilité est relativement courte. Après 500 µs, le signal est stable.

REBOND LORS DU RELÂCHEMENT



On constate lors du relâchement une durée d'instabilité de l'ordre de 500 µs.

CHOIX DE LA PERIODE D'ECHANTILLONNAGE

Avec le système mesuré, une période de 1 ms et un comptage de 3 à 5 états stables est suffisant.

Si on augmente la période, on garantit une meilleure élimination des rebonds, mais on augmente le temps de réaction.

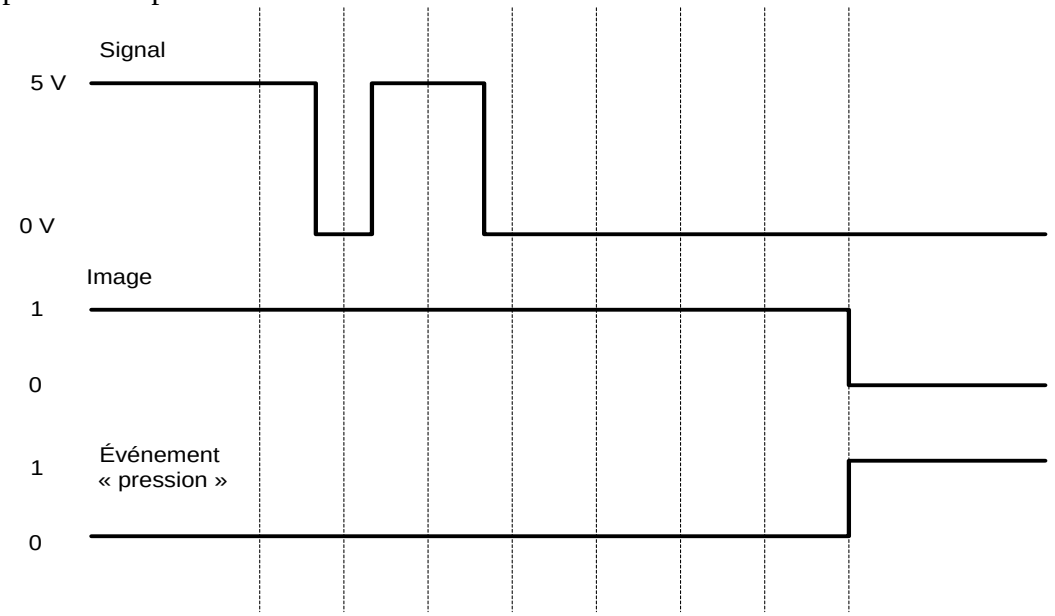
PRINCIPE DE L'ANTI REBOND

L'élimination des rebonds s'appuie sur la vérification de la stabilité du nouvel état en fournissant une image filtrée du signal ou de l'état de la touche.
En plus de fournir une image filtrée du signal, il peut être utile de fournir un indicateur d'événement, soit la pression ou le relâchement de la touche.

Les diagrammes qui suivent illustrent le comportement que l'on souhaite obtenir.

SITUATION DE PRESSION SUR LA TOUCHE

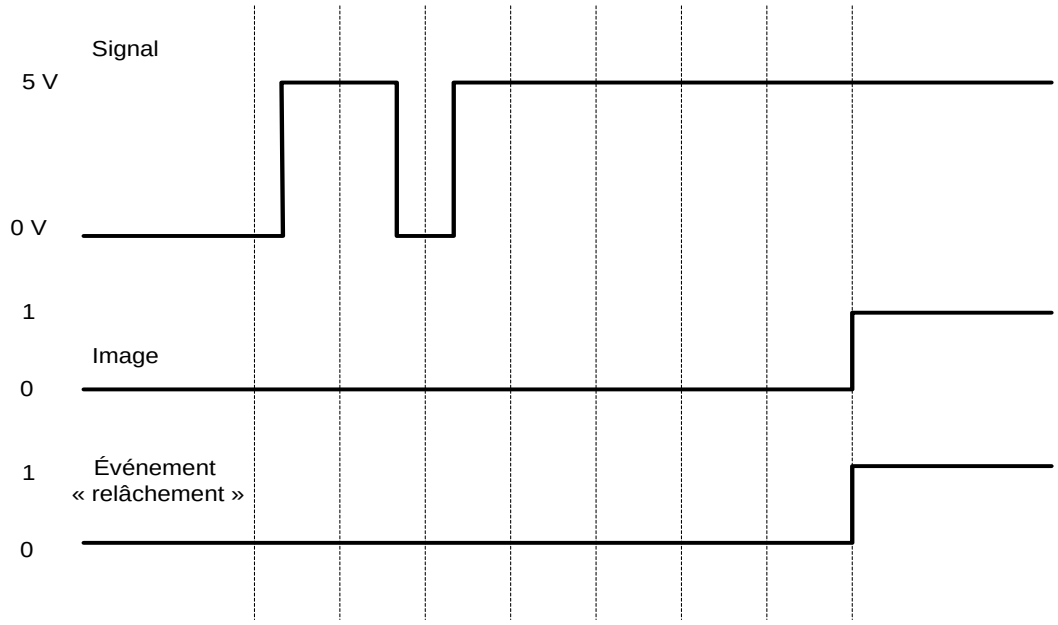
Le diagramme ci-dessous illustre le signal qui apparaît lors de la pression sur la touche, l'image du signal filtré créée par le programme, ainsi que l'indicateur d'événement.
Remarque : il est nécessaire de remettre à zéro l'indicateur d'événement lorsqu'il a été pris en compte.



Les traitillés verticaux indiquent les instants d'échantillonnage. Lorsque le système est stable (5 lectures avec la même valeur), alors on obtient le changement d'état sur l'image ainsi que l'événement.

SITUATION DE RELÂCHEMENT DE LA TOUCHE

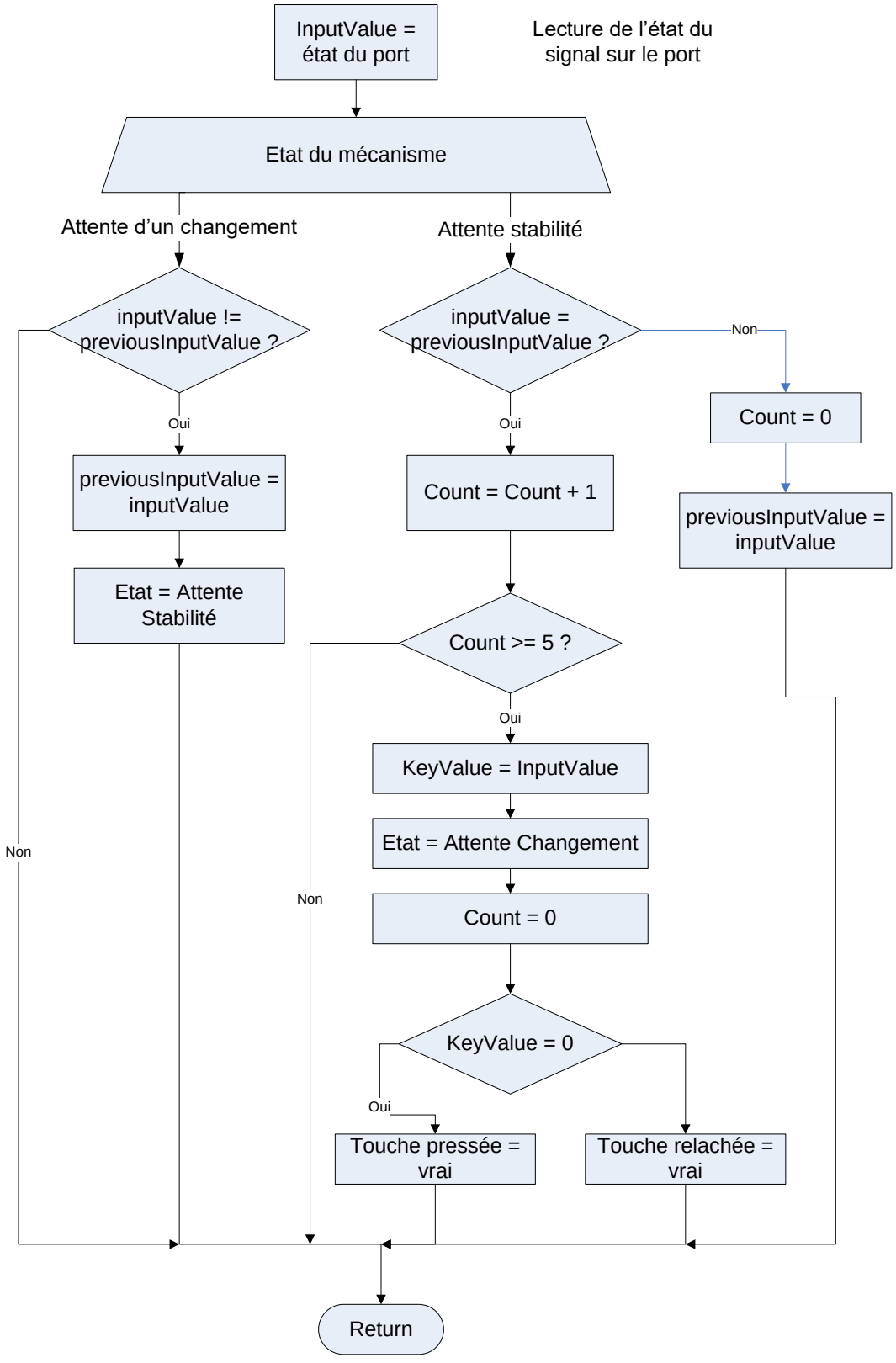
Le diagramme ci-dessous illustre la situation qui apparaît lors du relâchement de la touche.



ALGORITHME DE TRAITEMENT

Voici l'organigramme du traitement anti-rebond. La fonction décrite doit être appelée cycliquement et à intervalle fixe.

Les variables évoquées dans l'organigramme sont en réalité les champs d'une structure.



DESCRIPTION DE L'ALGORITHME

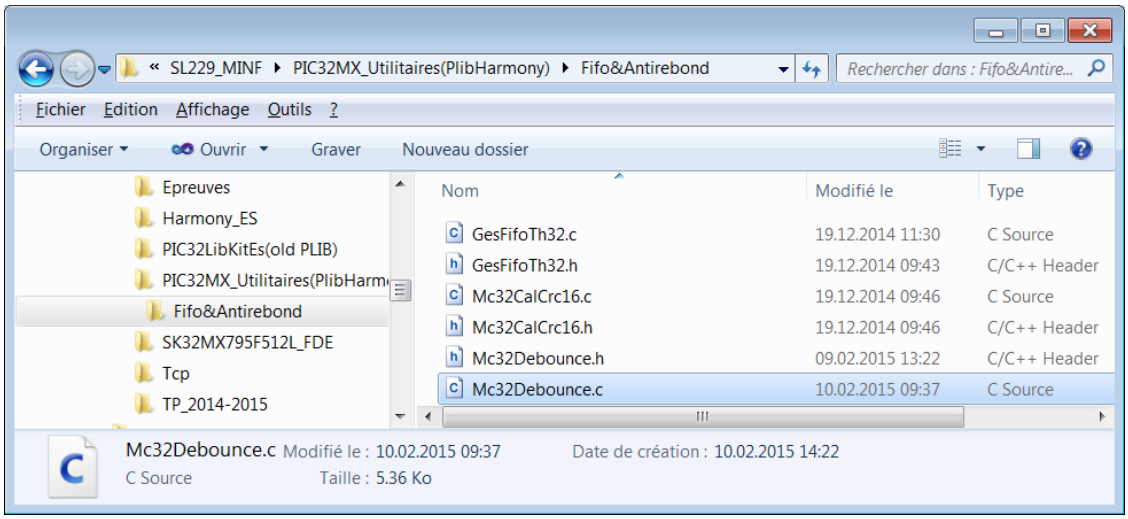
L'algorithme utilise deux états : l'attente d'un changement et l'attente de la stabilité du nouvel état.

Le système est normalement en attente de changement. Lorsque qu'un changement est détecté par la différence entre la valeur précédente et la nouvelle valeur, le système bascule dans le mode attente de la stabilité.

En mode attente de stabilité, il y a comptage si le nouvel état se maintient (nouvelle valeur = valeur précédente). Dès que le compteur atteint 5, on considère que la nouvelle valeur est stable. Dans ce cas il y a mise à jour de l'image de la touche, ainsi que mise à vrai de l'événement relâchement ou pression. Le mode est remis sur attente de changement.
Si durant le comptage la valeur change, le compteur est remis à 0, afin de recommencer un comptage pour vérifier la stabilité.

UTILITAIRE D'ANTI REBOND

La fonction DoDebounce est réalisée conformément à l'algorithme. Cette fonction utilise un descripteur qui est une structure, afin que plusieurs touches puissent être gérées en appelant la fonction avec des descripteurs différents.



Les fichiers Mc32Debounce.c et Mc32Debounce.c se trouvent sous …\Maitres-Eleves\SLO\Modules\SL229_MINF\PIC32MX_Utilitaires(PlibHarmony)\Fifo&Antirebond.

DÉCLARATION POUR DODEBOUNCE

```
// Etats du système antirebond
typedef enum {
      DebounceWaitChange,
      DebounceWaitStable,
} E_DebouceStates;

// Structure pour les flag
typedef struct {
   uint8_t  KeyPressed :1;     // événement touche pressée
   uint8_t  KeyReleased : 1;   // événement touche relâchée
   uint8_t  KeyPrevInputValue : 1;    // valeur précédente
   uint8_t  KeyValue : 1;          // valeur finale (image)
                                   // de la touche
 } s_bits;

// structure du descripteur
typedef struct {
    E_DebouceStates DebounceState;  // état de l'antirebond
    uint8_t     DebounceCounter;    // compteur
    s_bits bits;
} S_SwitchDescriptor;

// Constante comptage stabilité de l'anti-rebond
const int8_t MaxDebounceCount = 5;
```

FONCTION DODEBOUNCE

```
void DoDebounce (S_SwitchDescriptor *Descriptor,
                                        bool InputValue)
{
   bool PrevInputValue;
   s_bits tmp;

   tmp = Descriptor->bits;

   // Traitement selon état du mécanisme
   switch (Descriptor->DebounceState)
   {
      case DebounceWaitChange :
         PrevInputValue = tmp.KeyPrevInputValue;
         if ( ! (InputValue == PrevInputValue) ) {
             tmp.KeyPrevInputValue = InputValue;
             Descriptor->DebounceState =
                                        DebounceWaitStable;
         }
      Break;

      case DebounceWaitStable :
         PrevInputValue = tmp.KeyPrevInputValue;
         if ( InputValue == PrevInputValue )  {
            Descriptor->DebounceCounter++;
            if (Descriptor->DebounceCounter >=
                                        MaxDebounceCount)  {
                // Mise à jour du nouvel état
                tmp.KeyValue = InputValue;
                if (tmp.KeyValue == 0) {
                    tmp.KeyPressed = true;
                } else {
                    tmp.KeyReleased = true;
                }
                Descriptor->DebounceState =
                                        DebounceWaitChange;
                Descriptor->DebounceCounter = 0;
            }
         } else {
            Descriptor->DebounceCounter = 0;
            tmp.KeyPrevInputValue = InputValue;
         }
      Break;
   } // end switch
   Descriptor->bits = tmp;
} // end DoDebounce
```

LES FONCTIONS COMPAGNES DE DODEBOUNCE

Afin d'éviter de devoir connaître en détail le descripteur, plusieurs fonctions ont été réalisées pour obtenir certaines valeurs.

```
// Initialisation du descripteur
void DebounceInit (S_SwitchDescriptor *pDescriptor) {
    pDescriptor->DebounceState     = DebounceWaitChange;
    pDescriptor->debounceCounter   = 0;
    pDescriptor->bits.KeyPressed        = 0;
    pDescriptor->bits.KeyReleased       = 0;
    pDescriptor->bits.KeyPrevInputValue = 1;
    pDescriptor->bits.KeyValue          = 1;
}

// DebounceGetInput  fourni état touche après anti-rebond
bool DebounceGetInput (S_SwitchDescriptor *pDescriptor) {
   return (pDescriptor->bits.KeyValue);
}

// DebounceIsPressed  true indique que l'on vient
// de presser la touche
bool DebounceIsPressed (S_SwitchDescriptor *pDescriptor) {
   return (pDescriptor->bits.KeyPressed);
}

// DebounceIsReleased   true indique que l'on vient
// de relacher la touche
bool DebounceIsReleased (S_SwitchDescriptor
                                          *pDescriptor) {
   return (pDescriptor->bits.KeyReleased);
}

// DebounceClearPressed    annule l'indication de pression
// sur la touche
void DebounceClearPressed (S_SwitchDescriptor *pDescriptor)
{
   pDescriptor->bits.KeyPressed = false;
}
// DebounceClearReleased   annule l' indication
// de relâchement de la touche
void DebounceClearReleased  (S_SwitchDescriptor
                                           *pDescriptor) {
   pDescriptor->bits.KeyReleased = false;
}
```

EXEMPLE D'UTILISATION DE L'ANTI-REBOND

Voici un exemple reposant sur la gestion du codeur PEC12.

```
#include "Mc32Debounce.h"

// Descripteur des signaux
S_SwitchDescriptor DescrA;
S_SwitchDescriptor DescrB;
S_SwitchDescriptor DescrPB;
```

INITIALISATION DU MÉCANISME ANTI-REBOND

Dans la fonction Pec12Init, on procède à l'initialisation pour les 3 signaux du PEC12.

```
void Pec12Init (void)
{
   // Initialisation des descripteurs de touches Pec12
   DebounceInit(&DescrA);
   DebounceInit(&DescrB);
   DebounceInit(&DescrPB);

   // Initialisation de la structure PEC12
    Pec12.Inc = 0;              // événement incrément
    Pec12.Dec = 0;              // événement décrément
    Pec12.OK = 0;               // événement action OK
    Pec12.ESC = 0;              // événement action ESC
    Pec12.NoActivity = 0;       // Indication d'activité
    Pec12.PressDuration = 0;    // Pour durée pression du PB
    Pec12.InactivityDuration = 0; // Durée inactivité
} // Pec12Init
```

UTILISATION DU MÉCANISME ANTI-REBOND

Dans la fonction ScanPec12, nous allons utiliser les fonctions d'anti-rebond pour déterminer la situation d'incrément ou de décrément.

```
void ScanPec12 (bool ValA, bool ValB, bool ValPB)
{
   // Traitement anti-rebond sur A, B et PB
   DoDebounce (&DescrA, ValA);
   DoDebounce (&DescrB, ValB);
   DoDebounce (&DescrPB, ValPB);
```

```
// Dans le sens horaire CW:
//                 ________________
// A:   __________|                |_____________________
//                    ____________________
// B:   _____________|                    |_____________
// Dans le sens anti-horaire CCW:
//                       ____________________
// A:   ________________|                    |____________
//                 ________________
// B:   __________|                |_______________________

    // Détection incrément / décrément

    // Détection flanc descendant sur signal B
    if ( DebounceIsPressed(&DescrB) ) {
        // Quittance de l'événement
        DebounceClearPressed(&DescrB);
        if ( DebounceGetInput (&DescrA) == 0) {
            // Si A = 0 : situation CW = incrément
            Pec12.Inc = 1;
        } else {
            Pec12.Dec = 1;
        }
    }
    // Traitement du Push Button...

    // Gestion inactivité...

} // ScanPec12
```

Dans la fonction ScanPec12, qui est appelée cycliquement, on commence par appeler la fonction **DoDebounce** pour chacun des signaux du Pec12.

Ensuite il est possible d'utiliser les fonctions permettant de savoir si un flanc a été détecté. Si oui, il faut quittancer la fonction.

La fonction **DebounceGetInput** permet d'obtenir l'image filtrée de l'entrée physique après application de l'anti-rebond.

APPEL CYCLIQUE DE LA FONCTION SCANPEC12

Pour que le mécanisme d'anti-rebond fonctionne, il est indispensable que la fonction ScancPec12 soit appelée cycliquement. Dans notre cas, par exemple dans une interruption à chaque ms.

```
void __ISR(_TIMER_1_VECTOR, ipl3)
                        _IntHandlerDrvTmrInstance0(void)
{
    PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_TIMER_1);
    BSP_LEDToggle(BSP_LED_1); // pour contrôle
    ScanPec12(PEC12_A, PEC12_B, PEC12_PB );
}
```

On fournit l'état des entrées lors de l'appel de la fonction.