1 Qu'est-ce que PSoC ? PSoC contient :

1.1 Un cœur micro-processeur

1.2 Des interfaces variés : liaisons normalisées, pad tactile, cap sense, ADC DAC, timers, filtres numériques DMA .....

1.3 Les fonctions logiques de base : portes, bascules, registres ....

1.4 Des composants analogiques : ampli, comparateurs, multiplieurs...

2 Pourquoi utiliser PSoC

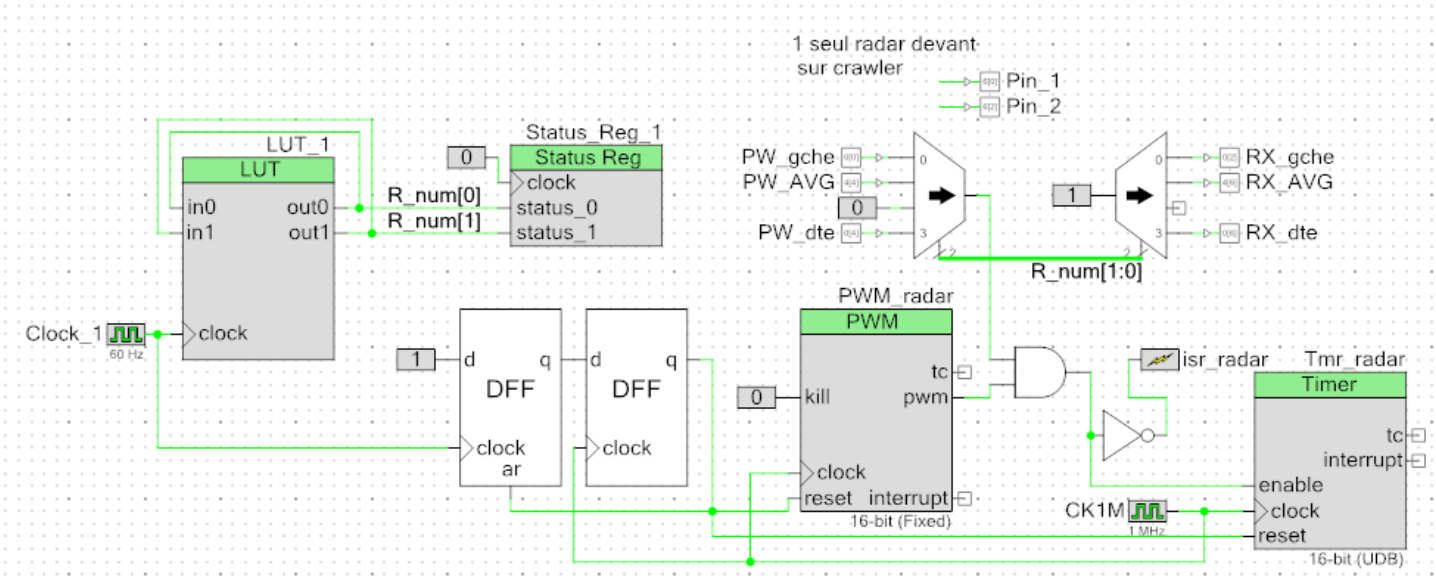
2.1 L'outil de développement et la doc sont très bien faits et gratuits.

2.2 Ajouter des sécurités câblées autour d'un projet micro-programmé.

2.3 Câbler les taches répétitives ou ayant besoin de synchroniser sur un signal pour simplifier le programme

exemple : Le projet ci-dessous gère un ensemble de 3 ou 4 radars ultra-son.

- Les tirs des radars sont déclenchés par les fronts montants (sur les sorties RX_ gche, RX_AVG, RX_dte) en envoyant 1 sur l'entrée du *multiplexeur* piloté par les 2 bits R_num[1:0].
- Le composant LUT_1 (look up table) a été configuré en *compteur binaire* 2 bits (on peut prendre direct un compteur à partir de la version 2.2 de PsoC creator) Il évolue à la fréq de 60Hz (vu son horloge, donc toutes les 16,6ms, on aiguille le 1 du mux vers un autre radar.
- => **aucune ligne de code pour déclencher les tirs radar** (et fonctionne même si le programme plante)



- Les 2 bascules D transforment le front montant issu de clock_1 en impulsion pour le reset de timer et PWM, ainsi on a une RAZ auto sur chaque tir radar (soft plus simple).
- PWM associé à la porte ET assure un front descendant si le radar ne répond pas (cible trop loin)
- Timer fait la mesure.
- le status_reg_1 permet de lire l'état de ce qu'on a câblé (les 2 bits qui choisissent le radar).

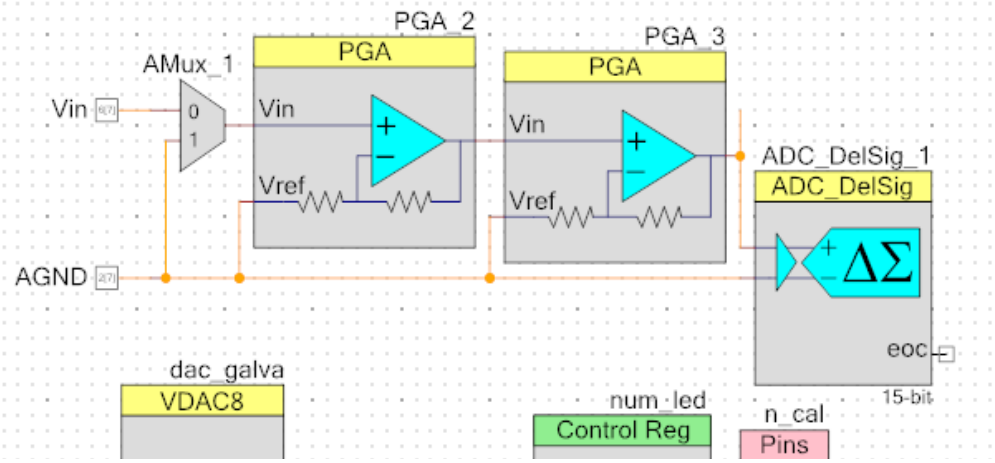
- voici la totalité du code de l'inter radar 4 lignes!

```
uint8 numrad;      // numero du radar
uint16 period;     //nombre periodes timer
numrad=Status_Reg_1_Read();          //lecture numero du radar actif
period=Tmr_radar_ReadCounter();      // recup valeur timer
period = 0xFFFF-period;        //convertir en nombre T horloge car il décompte
radar[numrad]=period/6;   // /6 convertit en mm, valeur ds tableau des resultats
```

Montrer une des vidéos du projet

2.4 PsoC peut réaliser un volmètre « autorange » avec moins d'une page de listing

- On peut remarquer que l'ADC est différentiel et le résultat est signé par rapport à AGND qui ici est à VCC/2
- Ici le sytème n'est pas au plus simple car on a une mesure d'offset et 2 amplis à gain variable (meilleure précision sur les gain faibles avec les PGA)




```
int16 i,offset;             // resultats intermédiaires (en quantum = calibre / 2^14)
uint8 j;                    // indice / compteur
uint16 FSadc = 2048;        // full scale ADC en mV
uint16 calibre;           // unité mV
//uint8 Tpga_[]={1,2,4,8,16,24,32,48,50}; // tableau des gains possibles PGA
uint8 Tpga[]={1,2,4,8,16,32,64};      // tableau de gain des paires PGA
uint8 Tval1[]={0,1,1,2,2,3,3,4,4};    //valeurs préférées paires en 2^n pour le réglage
uint8 Tval2[]={0,0,1,1,2,2,3,3,4};       // PGA + précis sur les gains faibles
PGA_2_SetGain(0) ;                      // gain mini
PGA_3_SetGain(0) ;                      // gain mini
AMux_1_Select(0) ;                      // mesure du signal
i=ADC_DelSig_1_GetResult16() ;          // écarter un résultat précédent
ADC_DelSig_1_StartConvert() ;           // démarre une conversion
if(!ADC_DelSig_1_IsEndConversion(ADC_DelSig_1_WAIT_FOR_RESULT)) // attends fin Conversion
      return 0;                       // si pb abandonner
i=ADC_DelSig_1_GetResult16() ;       // lecture ici
if(i<0)  i = -i;                        // i en positif
j=0;
if(!(i&0x4000) )    // overrange (ADC en 15 bits) laisser j=0
{                   //donc si pas over  autorange
  while((i&0x2000)==0)     // recherche ordre de grandeur b14 = 0 ?
  {    i<<=1;                    // doubler i
       j++;                 // nbre de crans d'ampli nécessaires en 2^n
       if (j==5) break;     // Gain maxi 2^6 si jmax=6
  }
}
PGA_2_SetGain(Tval1[j]) ;         // réglage des gains
PGA_3_SetGain(Tval2[j]) ; // le calcul suivant laisse un peu de temps pour que PGA change de
gain
calibre = FSadc/Tpga[j];    // en mV quantum = calibre/2^14 (16384 points en pos autant en
neg)
ADC_DelSig_1_StartConvert() ;       // démarre une conversion après réglage du gain
if(!ADC_DelSig_1_IsEndConversion(ADC_DelSig_1_WAIT_FOR_RESULT)) // attends Conversion
      return 0;                     // si pb abandonner
i=ADC_DelSig_1_GetResult16() ;
AMux_1_Select(1) ;  // mesure offset
ADC_DelSig_1_StartConvert() ;           // démarre une conversion
if(!ADC_DelSig_1_IsEndConversion(ADC_DelSig_1_WAIT_FOR_RESULT)) // attends Conversion
      return 0;                     // si pb abandonner
offset=ADC_DelSig_1_GetResult16() ;
i-=offset;              // correction enlever offset
mV=(float)i*calibre;    //conversion
mV/=16384;                          // réultat final en mV
//mV*=0.76;               // correction finale étage d'éntrée
AMux_1_Select(0) ;
return 1;
```

2.5 Filtre numérique

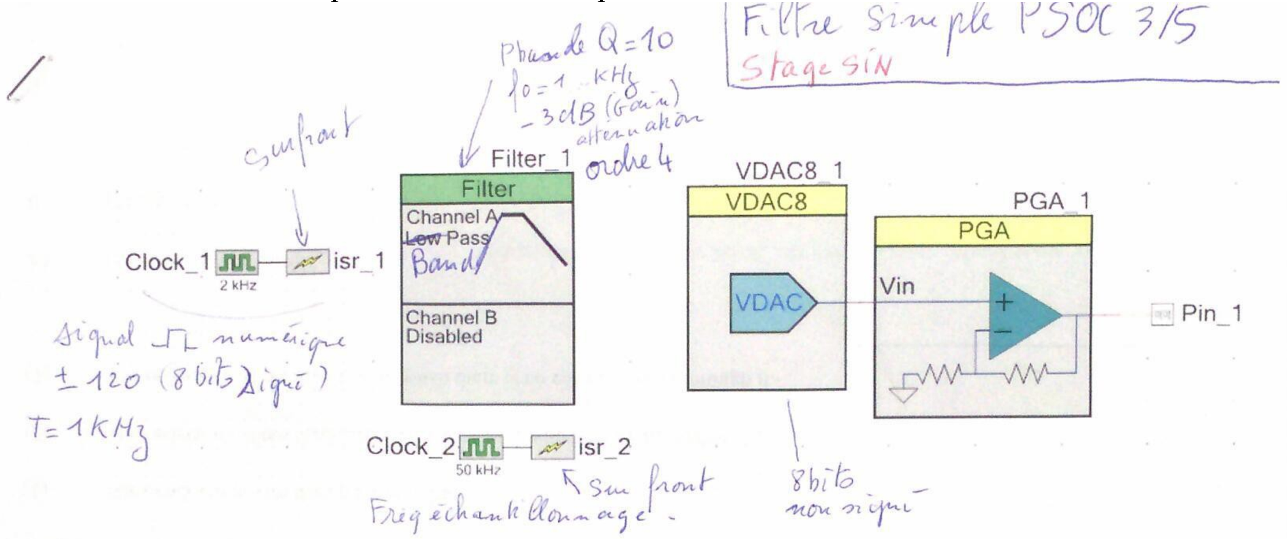
Le filtre numérique paraît complexe à utiliser (interruption ou DMA imposé pour respecter Féchantillonnage) mais sur les phénomènes lents où on peut faire un traitement numérique, le rapport signal/bruit et la sélectivité d'un passe bande peuvent être énormes.

- Pas de décalage de fréq centrale lié à l'incertitude des composants (si horloge PsoC de qualité)
- Sélectivité ahurissante mais attention au temps de réponse.
- Pas de précaution à prendre en sortie du filtre : on est en numérique, et le bruit de quantification est infime par rapport au bruit ds un circuit analogique.

3 TP#1 voltmètre
TP TSSE N°2
4 TP#2 fréquencemètre
TP TSSE N°4

5 TP#3 Filtrage

Un « carré » numérique => filtre numérique




```
#include <device.h>

int8 fltr_in ;    // non signé est possible

void main()
{

    Filter_1_Start() ;
    VDAC8_1_Start()  ;
    PGA_1_Start() ;
    isr_1_Start() ;
    isr_2_Start() ;
    CyGlobalIntEnable;  /* Uncomment this
    for(;;)
    {
        /* Place your application code here
    }
```

```
CY_ISR(isr_1_Interrupt)
{
    /*  Place your Interrupt code here. */
    /* `#START isr_1_Interrupt` */

    if(fltr_in== -120) fltr_in=120;
    else fltr_in= -120;

    /* `#END` */
```

```
CY_ISR(isr_2_Interrupt)
{
    /*  Place your Interrupt code here. */
    /* `#START isr_2_Interrupt` */


    uint8 i;
    Filter_1_Write8(Filter_1_CHANNEL_A, fltr_in) ;
    i=Filter_1_Read8(Filter_1_CHANNEL_A) ;
    VDAC8_1_SetValue(i+128);               // code signé => décalé


    /* `#END` */
```

CyDelay(100); isr1 après pour voir la réponse à la mise sous tension

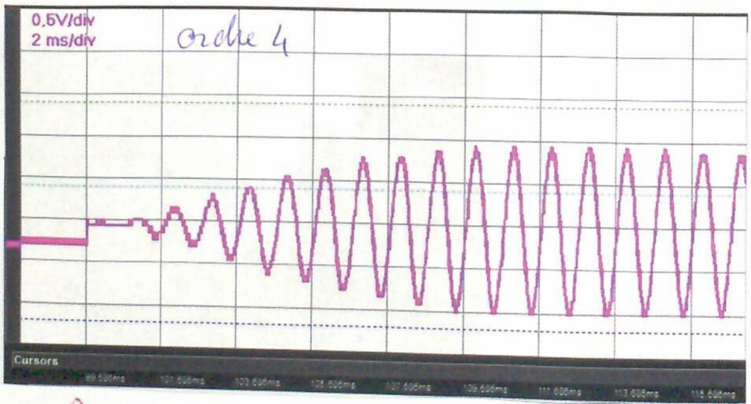
carré num en entrée du Filtre

échantillon en entrée du filtre

envoie sur DAC code décalé.

lecture du résultat Filtré

essayer avec ordre 2 tu devrait

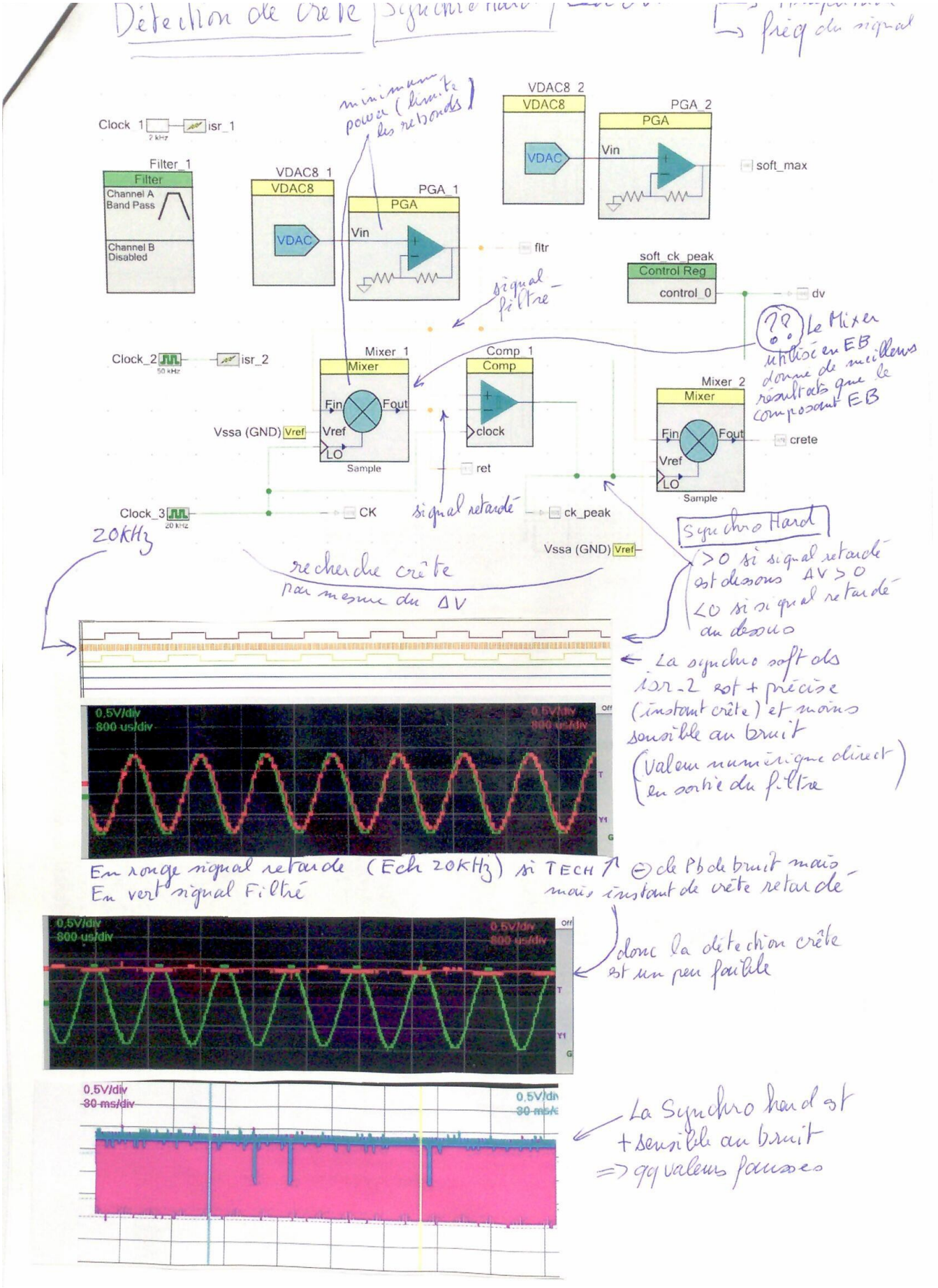


un bounded I/O → des allow

6 TP#4 extraire valeur crete et période d'un signal analogique

Traitement num et/ou soft sur le résultat du filtre

6.1 solution tout hard



6.2 solution syncho soft