Mise en œuvre des entrées/sorties sur STM32

Anthony Juton, janvier 2023

L'objet de ce guide est de mettre en œuvre les entrées sorties basiques de la carte microcontrôleur STM32. Un autre guide concerne la mesure de vitesse par la fourche optique.
On utilise ici l'environnement de développement gratuit, multiplateforme basé sur Eclipse STM32CubeIDE. On peut partir du projet avec les réglages par défaut pour la carte STM32L432KC.

1 Configuration des périphériques

Le schéma de la carte Hat de la voiture test CoVAPSy *Hat_CoVAPSy_v1re2.pdf* indique les connexions de la carte Nucleo.

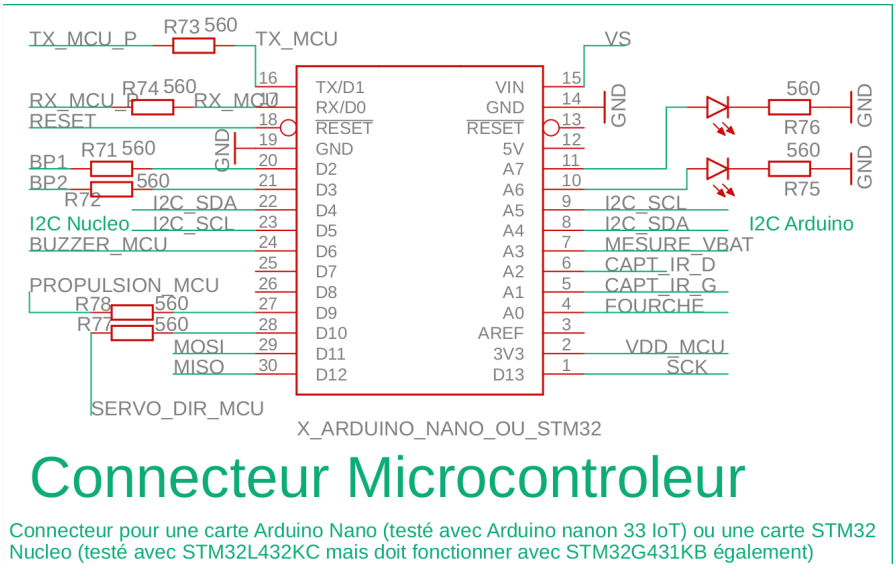

Figure 1: Extrait du schéma de la carte Hat_CoVAPSy_v1re2

Le tableau suivant résume les connexions qui concernées par ce guide :

Fonction	Nom Arduino/Nucleo32	Broche microcontrôleur	Type
BP1	D2	PA_12	TOR
BP2	D3	PB_0	TOR
LED3	A7	PA_2	TOR
LED4	A6	PA_7	TOR
PROPULSION	D9	PA_8	PWM
DIRECTION	D10	PA_11	PWM
MESURE_VBAT	A3	PA_4	analogique
CAPTEUR_IR_DROIT	A2	PA_3	analogique
CAPTEUR_IR_GAUCHE	A1	PA_1	analogique

1.1 Configuration des broches TOR

Depuis le volet *Pinout View* et le volet de configuration *GPIO,* associer les entrées/sorties TOR et leur donner un nom.

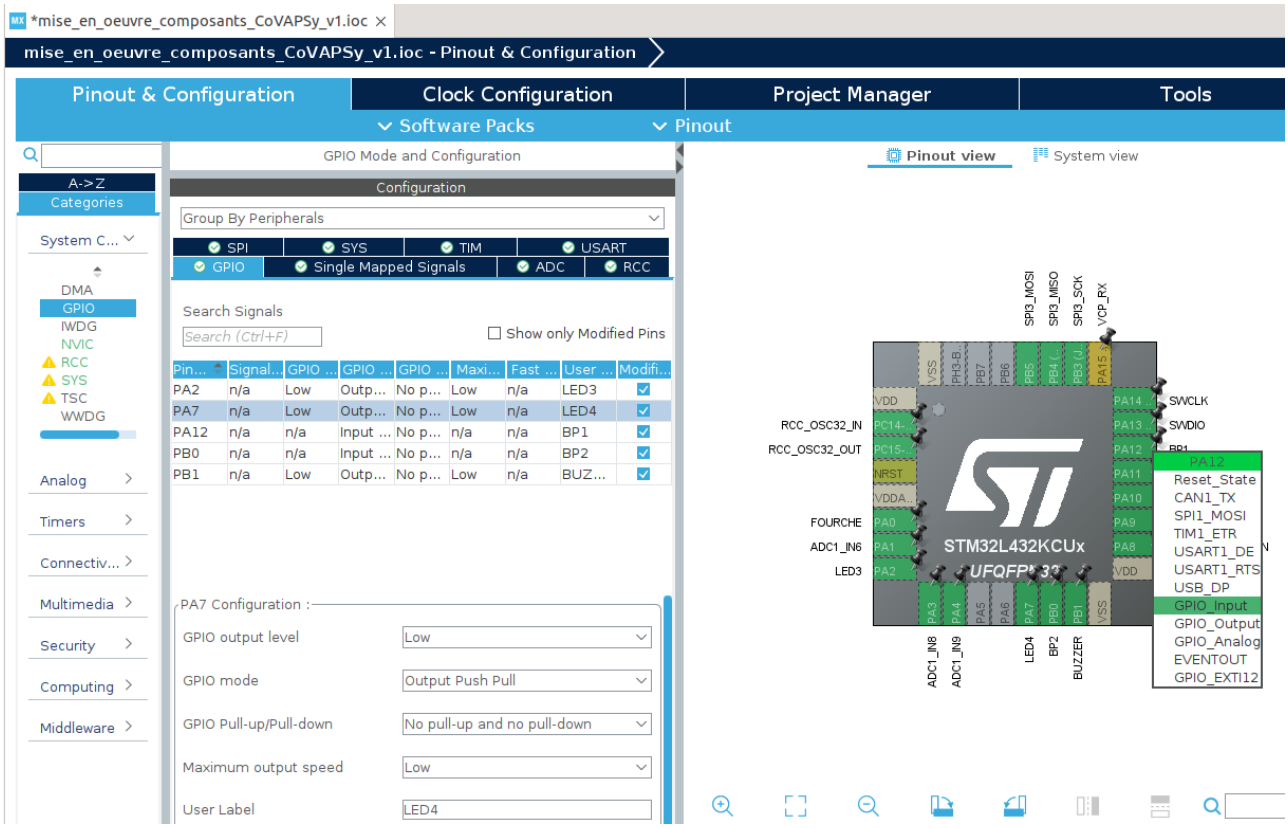

Figure 2: configuration des entrées TOR sous STM32CubeIDE

1.2 Configuration des broches PWM

Propulsion et direction sont deux sorties PWM, associées au Timer 1.

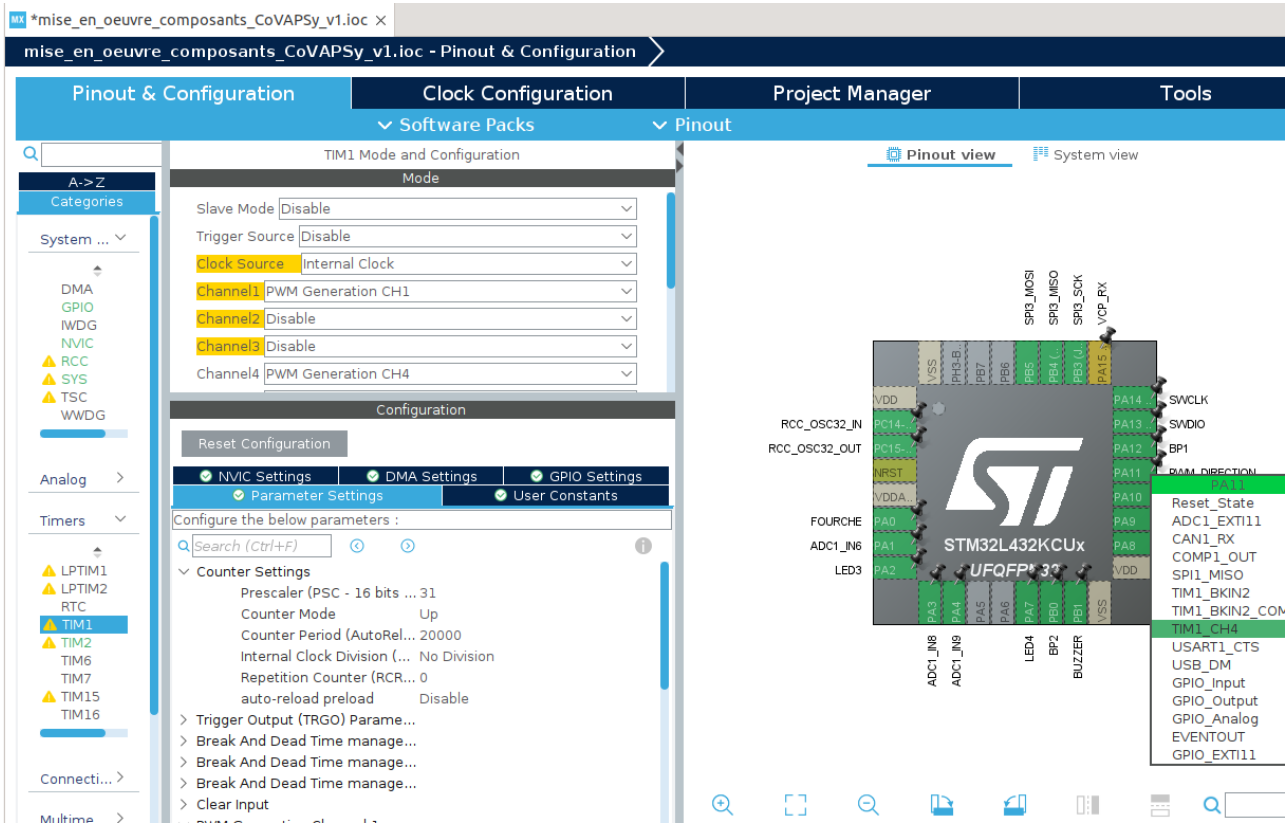

Figure 3: Configuration des sorties PWM depuis STM32CubeIDE

L'horloge du timer 1 étant à 32 MHz par défaut (réglable via l'onglet *Clock Configuration*). Dans la fenêtre de configuration du timer 1, on choisit 31 pour le préscalaire. L'horloge du timer est alors à 1 MHz.

Les PWM du servo-moteur de direction et du variateur de propulsion doivent avoir une période de 20 ms et une valeur au repos d'environ 1,5 ms. On configure donc la période du timer à 20000 (*Counter Period*) et la largeur d'impulsion des PWM à 1500 µs :

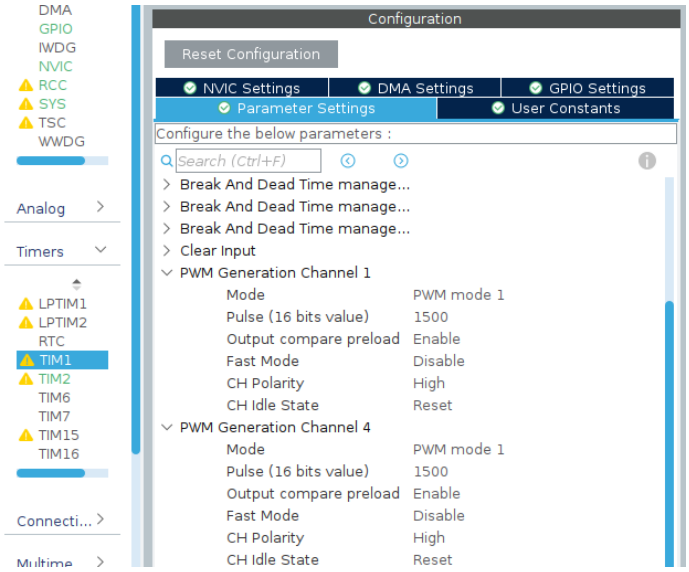

Figure 4: Configuration de la largeur d'impulsion de démarrage des PWM

1.3 Configuration des entrées analogiques

Les entrées analogiques se configurent de la même manière.

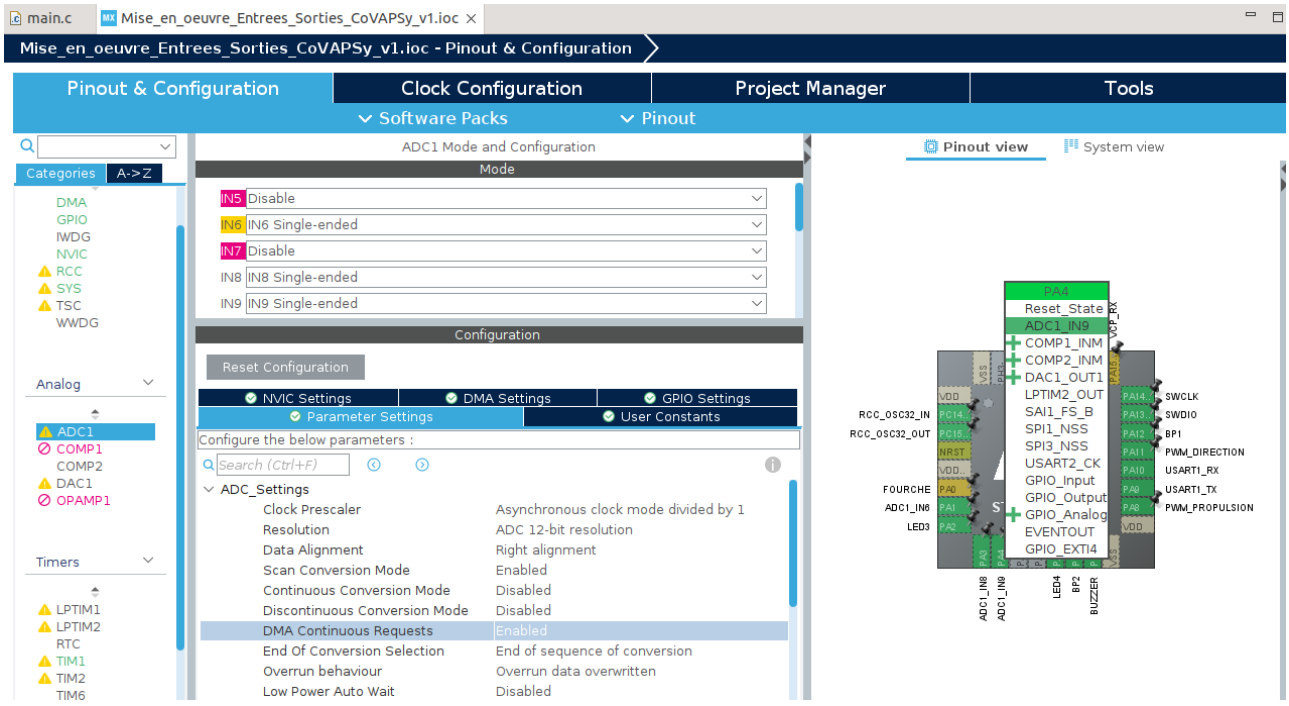

Figure 5: Configuration des entrées analogiques via STM32CubeIDE

Pour faire au plus simple (conversion automatique des 3 canaux), on choisit de travailler avec *Scan Conversion* activé (pour cela il faut d'abord avoir rempli les rangs des multiples conversions, comme expliqué ci-dessous), et *requêtes DMA* activées et la fin de conversion

activée lors de la fin de la séquence. Le DMA (*Direct Memory Access*) est le périphérique qui gère directement (sans passer par la CPU) la copie des sorties du convertisseur analogique-numérique dans la mémoire.

On configure ensuite les rangs des 3 voies à convertir :

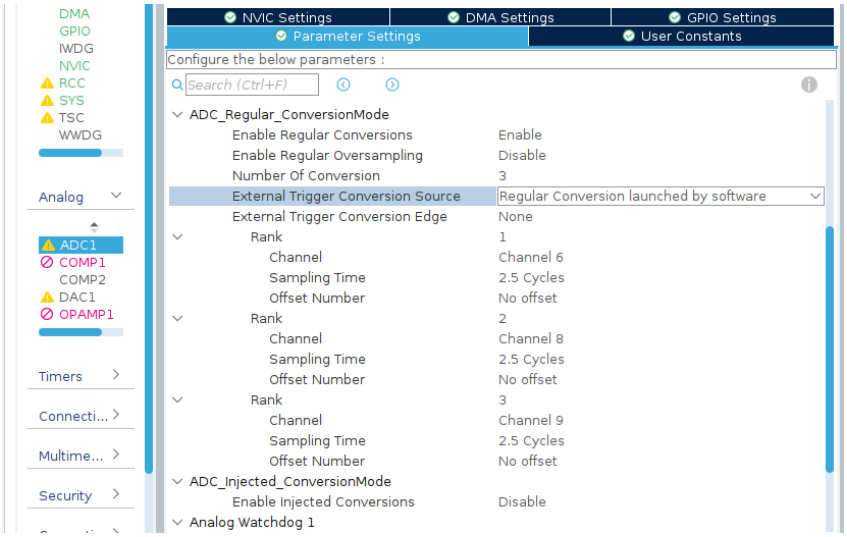

Figure 6: Configuration des rangs des 3 canaux utilisés pour la conversion multiple

Enfin, dans DMA settings, on ajoute un canal (bouton *Add*) et on y configure le mode circulaire (écrasement des anciennes valeurs) et un fonctionnement par mot (*Word*) de 32 bits.

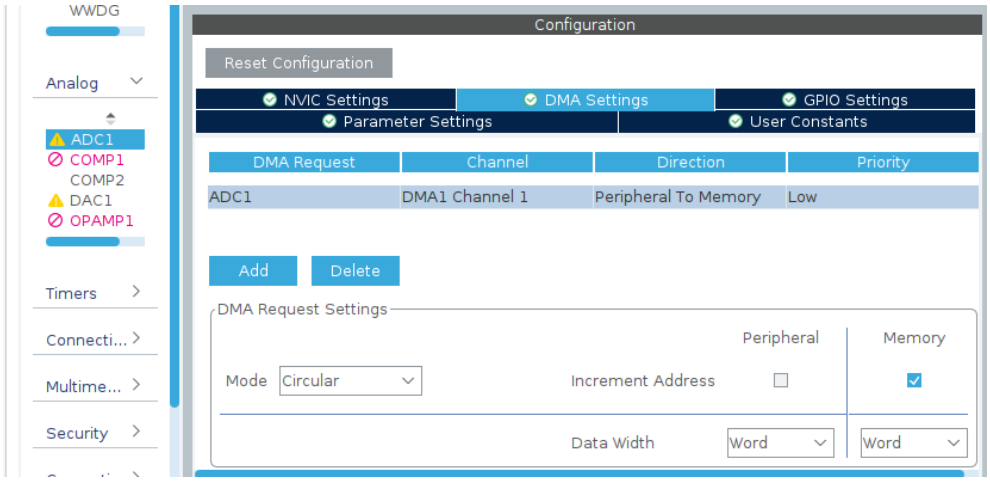

Figure 7: Ajout d'un canal DMA pour le convertisseur analogique numérique

1.4 Génération automatique du code

Une fois la configuration terminée, on génère le code associé : *Project > Generate Code*.

2 Code

Pour visualiser simplement le retour du convertisseur analogique-numérique, on affiche les résultats dans la fenêtre Live Expression de STM32CubeIDE, via des variables globales.

```
/* USER CODE BEGIN PV */
uint32_t lectures_ADC[3];
/* USER CODE END PV */
```

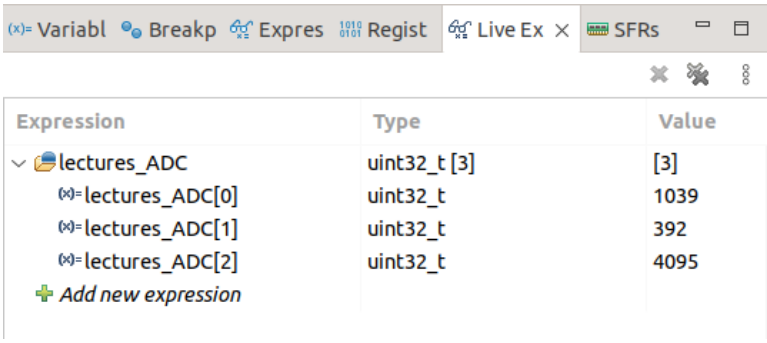

Figure 8: Supervision « live » des valeurs acquises par le convertisseur ADC

De plus, on crée quelques variables utiles pour le bon fonctionnement de ce programme de test.

```
int main(void)
{
  /* USER CODE BEGIN 1 */
      uint32_t bp1,bp2,bp1_old=0,bp2_old=0;
      uint32_t dc_propulsion=1500;
      uint32_t dc_direction=1500;
  /* USER CODE END 1 */
```

2.1 démarrage des périphériques

Au début du programme main, après les initialisations automatiques, on active le timer 1 et les sorties PWM 1 et 4 :

```
  /* USER CODE BEGIN 2 */
  HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
  HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_4);
  /* USER CODE END 2 */
```

2.2 La boucle principale

Dans la boucle principale, on lance la conversion analogique numérique, on lit les boutons et sur changement d'état de ces derniers, on modifie la commande de direction et la commande de vitesse.

Grâce au DMA, les valeurs converties sont automatiquement copiées dans la mémoire, dans la variable globale *lectures_ADC[3]*.


```
/* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
        //démarrage de la conversion ADC des 3 canaux
        HAL_ADC_Start_DMA(&hadc1, lectures_ADC, 3);

        //lecture des boutons
        bp1 = HAL_GPIO_ReadPin(BP1_GPIO_Port, BP1_Pin);
        bp2 = HAL_GPIO_ReadPin(BP2_GPIO_Port, BP2_Pin);

        //détection front descendant sur bp1
        if((bp1 == BP_ENFONCE) && (bp1_old == BP_RELACHE))
        {
              //changement d'état de la Led3
              HAL_GPIO_TogglePin(LED3_GPIO_Port, LED3_Pin);

              //propulsion un peu plus vite
              if(dc_propulsion < 1700)
              {
                          dc_propulsion += 10;
              }
              __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, dc_propulsion);

              //direction un peu plus à droite
              if(dc_direction < (BUTEE_DROITE-10))
              {
                    dc_direction += 10;
              }
              __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, dc_direction);
        }

        //détection front descendant sur bp2
        if((bp2 == BP_ENFONCE) && (bp2_old == BP_RELACHE))
        {
              //changement d'état de la Led4
              HAL_GPIO_TogglePin(LED4_GPIO_Port, LED4_Pin);

              //propulsion un peu moins vite
              if(dc_propulsion > 1500)
              {
                    dc_propulsion -= 10;
              }
              __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, dc_propulsion);

              //direction un peu plus à gauche
              if(dc_direction > (BUTEE_GAUCHE+10))
              {
                    dc_direction -= 10;
              }
              __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, dc_direction);
        }

        //attente de la fin de la conversion ADC, si jamais ce n'est pas encore
fini
        HAL_ADC_PollForConversion(&hadc1, 1);

        //sauvegarde des valeurs de bp1 et bp2 pour la détection des fronts
        bp1_old = bp1;
        bp2_old = bp2;
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
```

3 Amélioration du programme

Il faut encore ajouter les fonctions de conversion pour transformer les valeurs lues par le convertisseur ADC en distance ou en tension.

Un asservissement de la commande de vitesse est possible. Lire pour cela la fiche sur la mise en œuvre de la fourche.

La communication SPI entre le microcontrôleur et la raspberry Pi est expliquée dans une autre fiche.