Analog digital converter (ADC)

1. Présentation

L'ADC- 12bits est un convertisseur analogique numérique à approximations successives. Ce convertisseur admet 19 entrées multiplexés dont 3 réservées pour la mesure de température, de la tension de référence interne et de la tension de batterie. Le résultat de conversion est chargé dans un registre 16bits.

L'ADC admet plusieurs modes de fonctionnement, on va se limiter dans ce cours aux modes « Regular » et « Injected ».

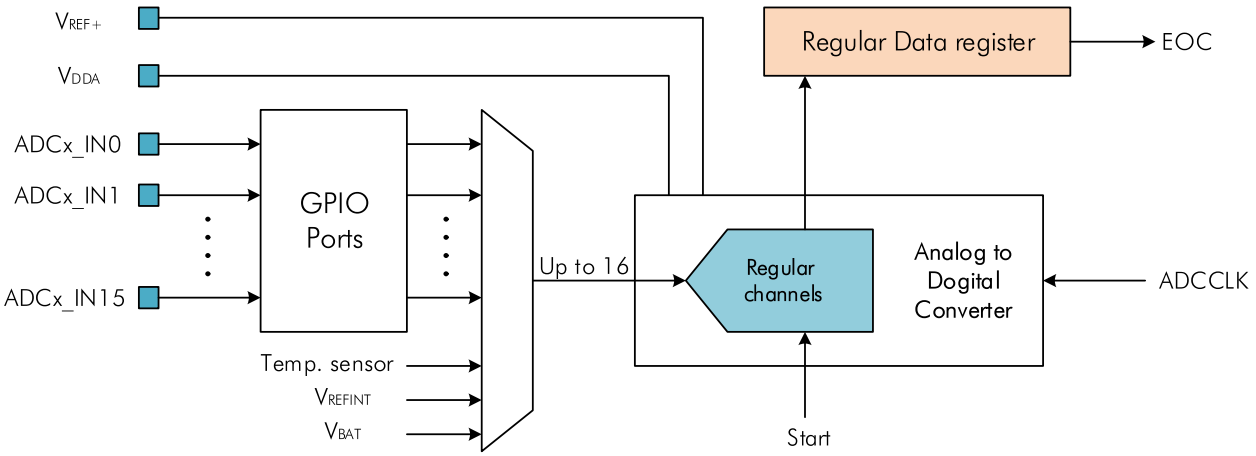


Figure 1 : schéma bloc simplifié de l'ADC

Le convertisseur est alimenté quand le bit ADON du registre ADC_CR2 est mis à 1. Le début de conversion est signalé de façon matérielle (par l'un des timer) ou de façon logicielle, par la mise à 1 du bit SWSTART du registre ADC_CR2.

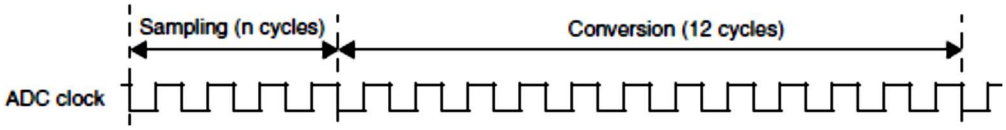
L'horloge du convertisseur (ADCCLK) est dérivée à partir de l'horloge du bus ABP2 divisée par 2, 4, 6 ou 8. La fréquence d'horloge maximale de l'ADC atteint 36Mhz, une fréquence de 30 Mhz est considérée typique.

2. Détermination du temps d'acquisition

La conversion d'un signal analogique en équivalent numérique passe par deux phases :

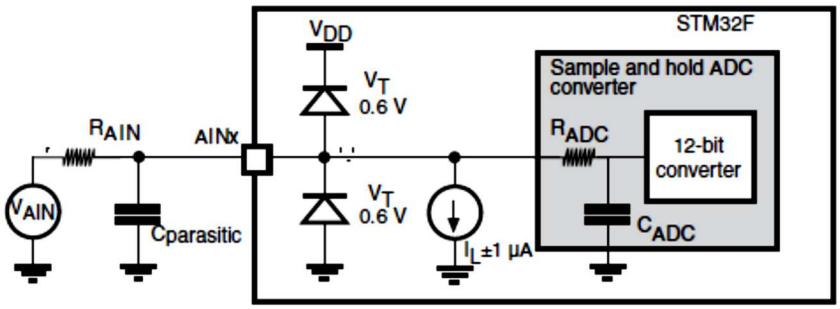
- l'échantillonnage blocage (sample and hold). Cette opération consiste à connecter l'entrée à convertir à un condensateur interne, qui va se charger à travers une résistance interne jusqu'à la tension appliquée. Ce temps est appelé temps d'acquisition ou temps d'échantillonnage (Simpling Time).

- Après l'épuisement du temps d'échantillonnage programmé ; la source de tension externe est déconnectée, pour procéder à la conversion de la tension aux bornes du condensateur.



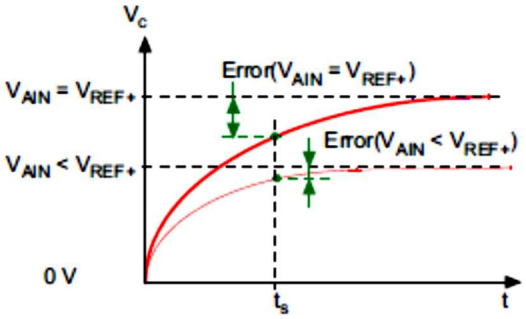
Temps d'acquisition

C'est le temps nécessaire pour que le condensateur interne atteigne une tension proche de la tension à convertir. Cette charge s'effectue à travers la résistance interne et la résistance de la source connectée à l'entrée de l'ADC.



En négligeant la capacité parasite, la tension aux bornes du condensateurs C_{ADC} : $V_C = V_{AIN}(1 - e^{\frac{-t}{(R_{AIN}+R_{ADC})C_{ADC}}})$.

Supposons que l'erreur maximal est de $\frac{1}{2}LSB$, nous allons calculer la résistance de la source correspondante. On a donc : $V_{AIN} - V_C = \frac{1}{2}LSB$.



On remarque que l'erreur maximal correspond à $V_{AIN} = V_{REF+}$.

Le temps d'acquisition, $t_s = T_s \cdot T_{ADC} = T_s / F_{ADC}$; T_s représentation le temps d'acquisition évalué en nombre de cycle d'horloges.

Prenant un T_s qui correspond à l'erreur maximale ($V_{AIN} = V_{REF+}$). Et cherchons la résistance R_{AIN} maximale.

$Erreur = V_{REF+} - V_{REF+}\left(1 - e^{\frac{-t_s}{(R_{AIN\max}+R_{ADC\max})C_{ADC}}}\right) = \frac{1}{2} \cdot \frac{V_{REF+}}{2^N}$, avec N est la résolution du convertisseur $N = 12$.

D'où $e^{\frac{-t_s}{(R_{AIN\max}+R_{ADC\max})C_{ADC}}} = \frac{1}{2^{N+1}} \Rightarrow t_s = (R_{AIN\max} + R_{ADC\max})C_{ADC} \cdot \ln(2^{N+1})$. Le nombre de cycle T_s : $\boxed{T_s = (R_{AIN\max} + R_{ADC\max})F_{ADC} \cdot C_{ADC} \cdot \ln(2^{N+1})}$

Il faut vérifier aussi la condition de Shannon : $F_{AIN} < 2 \cdot F_{SMP}$

F_{AIN} : Fréquence du signal d'entrée

F_{SMP} : Fréquence d'échantillonnage.

Le temps de conversion total : $t_{convtotal} = t_s + t_{conv} = (T_s + T_{conv}) \cdot T_{ADC} \leq T_{SMP}$ d'où $\boxed{T_s + T_{conv} \leq \frac{F_{ADC}}{F_{SMP}}}$

Pour une conversion sur 12 bits : $T_{conv} = 12 cycles$.

Pour les microcontrôleurs STM32F4xx, T_s peut prendre l'une des valeurs suivantes : **3** cycles, **15** cycles, **28** cycles, **56** cycles, **84** cycles, **112** cycles, **144** cycles ou **480** cycles.

Pour calculer le temps d'acquisition, il faut consulter le document constructeur.

Symbole	Paramètre	Conditions	Min	Type	Max	Unité
V_{DDA}	Alimentation		1.8	-	3.6	V
V_{REF+}	Tension de référence positive		1.8	-	V_{DDA}	V
f_{ADC}	Fréquence d'horloge de l'ADC	V_{DDA} = 1,8 à 2,4 V	0.6	15	18	MHz
		V_{DDA} = 2,4 à 3,6 V	0.6	30	36	MHz
R_{AIN}	impédance d'entrée externe		-	-	50	kΩ
R_{ADC}			-	-	6	kΩ
C_{ADC}	Capacité de l'interrupteur d'échantillonnage		-	4	-	pF

Calculons T_s pour $R_{AIN} = 10k\Omega$; $R_{ADC} = 6k\Omega$; $C_{ADC} = 4pf$; $F_{ADC} = 30Mhz$;

$T_s = 16 \cdot 10^3 \times 30 \cdot 10^6 \times 4 \cdot 10^{-12} \times 9 = 17,28 cycles$ soit $T_s = 28 cycles$.

Le temps de conversion total : $t_{convtotal} = (T_s + T_{conv}) \cdot T_{ADC} = 40/30\,\mu s = 1,34\mu s$

On peut calculer la fréquence d'échantillonnage maximale : $F_{SMP} = \frac{F_{ADC}}{T_s + T_{conv}} = \frac{30Mhz}{28+12} = 750Khz$

3. Modes de fonctionnes des ADCs

Les microcontrôleurs STM32 possède 3 convertisseurs analogiques numériques identiques. Qui peuvent fonctionner en ***modes indépendants*** ou en ***modes combinés***. Dans ce cours nous allons nous limiter aux modes indépendants.

Chaque ADC admet 16 entrées multiplexées ; la conversion de ces entrées peut être organisée en deux groupes. Un groupe consiste en une séquence de conversions pouvant être effectuées sur n'importe quel canal et dans n'importe quel ordre. Par exemple, il est possible d'implémenter la séquence de conversion dans l'ordre suivant : ADC_IN3, ADC_IN8, ADC_IN2, ADC_IN2, ADC_IN0, ADC_IN2, ADC_IN2, ADC_IN15.

- Un ***groupe régulier*** « *Regular* » : composé de 16 conversions maximum. Le nombre de canaux normaux, leur ordre dans la séquence de conversion doivent être indiqué. A la réception du signal de début de conversion, les canaux sont convertis automatiquement l'une à la suite de l'autre. Après chaque conversion, le correspondant numérique est chargée dans le registre de données ADC_DR.
- Un ***groupe injecté*** « *Injected* » : il diffère du groupe régulier, par le nombre de conversion qui est limité à 4 conversions au maximum et, que chaque canal admet son propre registre résultat ADC_JDRx (x : 1 à 4).

Note : Les deux groupes peuvent être sélectionnés ensemble, dans ce cas la priorité est accordée au groupe *Injected*.

3.1. Modes de conversion

J'expose dans cette partie les modes conversion attachées au mode indépendant.

3.1.1. Un seul canal / Une seule conversion

C'est le mode ADC le plus simple. Après la réception de l'ordre du début de conversion, l'ADC effectue la conversion unique (échantillon unique) du canal sélectionné et s'arrête après la conversion. La procédure de conversion passe par trois étapes principales :

- L'ordre de début de conversion, déclenché par la mise à 1 du bit SWSTART en mode régulier (ou JSWSTART en mode injecté) ou par un trigger externe généré par un Timer.
- Une fois la conversion terminée, le résultat de conversion est chargé dans le registre ADC_DR en mode régulier ou le registre ADC_JDR1 en mode injecté.
- La fin de conversion est signalée par la mise à un du bit EOC (End Of Convert) en mode régulier ou JEOC en le mode injecté.

3.1.2. Multi-canaux avec balayage (scan) / Une seule conversion

En mode régulier, après l'ordre de début de conversion, les canaux de la séquence sont balayés automatiquement l'un à la suite de l'autre (scan mode), la fin de conversion d'un canal ou de la totalité de la séquence est signalée par le bit EOC. Pour ne pas solliciter le CPU et éviter la perte des données, il est fortement recommandé d'utiliser le DMA pour transférer après chaque conversion.

4. Module ADC avec la librairie HAL

Le librairie HAL déclare les périphériques ADC avec des structures similaires à celles des GPIO. La structure ***ADC_ TypeDef*** contient différents registres de l'ADC. La structure ***ADC_ InitTypeDef*** contient les membres suivants :

```
typedef struct
{
  uint32_t ClockPrescaler;        /* définie l'horloge de l'ADC (ADCCLK = APB2/ ClockPrescaler)
                                   ClockPrescaler :
                                                ADC_CLOCK_SYNC_PCLK_DIV2
                                                ADC_CLOCK_SYNC_PCLK_DIV4
                                                ADC_CLOCK_SYNC_PCLK_DIV6
                                                ADC_CLOCK_SYNC_PCLK_DIV8 */.
  uint32_t Resolution;            /* configure la résolution du convertisseur */
                                                ADC_RESOLUTION_12B
                                                ADC_RESOLUTION_10B
                                                ADC_RESOLUTION_8B
                                                ADC_RESOLUTION_6B
  uint32_t DataAlign;             /* Alignement du résultat à droite ou à gauche */
                                                ADC_DATAALIGN_RIGHT
                                                ADC_DATAALIGN_LEFT
  uint32_t ScanConvMode;          /* conversion successif de tous les entrées */
                                                ENABLE ou DISABLE
  uint32_t EOCSelection;          /* spécifie la mode de fin de conversion */
```

```
                                                ADC_EOC_SEQ_CONV
                                                ADC_EOC_SINGLE_CONV
  uint32_t NbrOfConversion;          /* définie le nombre d'entrées à convertir */
                                                de 1 à 16
  uint32_t NbrOfDiscConversion;     /* spécifie le nombre de conversion en mode discontinu */
  uint32_t ExternalTrigConv;         /* définie la source de l'entrée de début de conversion */
                                      si ADC_SOFTWARE_START est spécifiée, le déclenchement
                                      de début de conversion externe est désactivé.
  uint32_t ExternalTrigConvEdge;      /* spécifie la source et le front d'activation de la source
                                        externe de début de conversion, */
                                              ADC_EXTERNALTRIGCONVEDGE_NONE;
  FunctionalState DMAContinuousRequests; /* spécifie le mode de DMA. */
                                                ENABLE ou DISABLE
}ADC_InitTypeDef;
```

La structure ***ADC_ HandleTypeDef*** est définie comme suit :

```
typedef struct {
  ADC_TypeDef           *Instance;              /* Pointe sur les adresses des ADC */
                                                    ADC1, ADC2 ou ADC3
  ADC_InitTypeDef        Init;                  /* Pour définir les paramètres de l'ADC  */
    . . .
}ADC_HandleTypeDef;
```

Les paramètres de la structure ADC_HandleTypeDef;sont affectés aux registres correspondants par le biais de la fonction « ***HAL_ StatusTypeDef HAL_ ADC_ Init(ADC_ HandleTypeDef* hadc)*** »

La structure ***ADC_ ChannelConfTypeDef*** définie les paramètres de chaque canal

```
typedef struct {
  uint32_t Channel;                 /*!< spécifie le canal à convertir */
                                        ADC_CHANNEL_0
                                        ADC_CHANNEL_1
                                        ADC_CHANNEL_2
                                         . . .
  uint32_t Rank;                     /* spécifie pour le groupe régulier le rang du canal à convertir */
                                          de  1 à  16
  uint32_t SamplingTime;            /* définie le temps d'acquisition */
                                        ADC_SAMPLETIME_3CYCLES
                                        ADC_SAMPLETIME_15CYCLES
                                        ADC_SAMPLETIME_28CYCLES
                                        ADC_SAMPLETIME_56CYCLES
                                        ADC_SAMPLETIME_84CYCLES
                                        ADC_SAMPLETIME_112CYCLES
                                        ADC_SAMPLETIME_144CYCLES
                                        ADC_SAMPLETIME_480CYCLES
}ADC_ChannelConfTypeDef;
```

Les paramètres de la structure ADC_ChannelConfTypeDef sont affectés aux registres correspondants par le biais de la fonction « ***HAL_StatusTypeDef HAL_ADC_ConfigChannel (ADC_HandleTypeDef* hadc, ADC_ChannelConfTypeDef* sConfig)*** »

Le logiciel CubeMx à travers la librairie HAL Driver, génère les initialisations nécessaires. Le convertisseur ADCx est connu à travers sa structure principale **hadcx** de type **`ADC_HandleTypeDef`.**

Fonction de début de conversion, cette fonction valide la mise en service du convertisseur.

```
HAL_StatusTypeDef HAL_ADC_Start(ADC_HandleTypeDef* hadc)
```

Paramètre d'appel : `hadc1 ou hadc2 ou hadc3`

Valeur de retour : `HAL_OK`.

Lecture du résultat de conversion :

```
uint32_t HAL_ADC_GetValue(ADC_HandleTypeDef* hadc) ;
```

Paramètre d'appel : `hadc1 ou hadc2 ou hadc3`

Valeur de retour : `résultat de conversion`.

Attente de fin de conversion de fin de conversion

```
HAL_StatusTypeDef HAL_ADC_PollForConversion(ADC_HandleTypeDef* hadc, uint32_t
Timeout) ;
```

Paramètres :

`hadc : hadc1 ou hadc2 ou hadc3`

`Timeout` : en milliseconde.

Valeur de retour : `HAL_OK, HAL_TIMEOUT, HAL_ERROR`.