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 au mode normal sans scan.

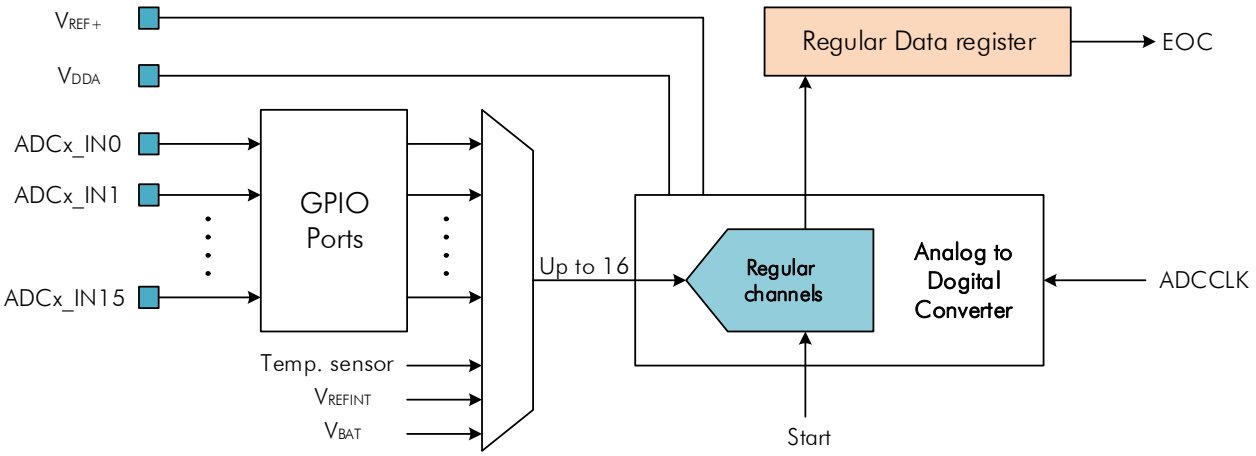


Figure 1 : schéma bloc simplifié de l'ADC

Le convertisseur est alimenté quand le bit ADON du bit 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.

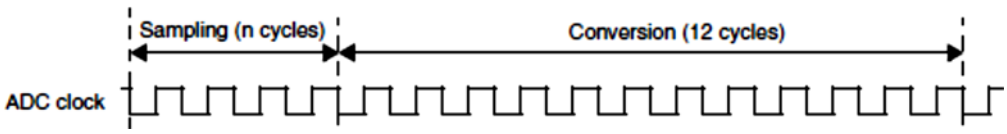
L'horloge du convertisseur (ADCCLK) est dérivée à partir de l'horloge du bus ABP2 divisée par 2, 4, 6 ou 8. La fréquence d'horloge maximale de l'ADC atteint 30Mhz quand la fréquence du bus ABP2 est de 60Mhz.

2. Détermination du temps d'acquisition

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

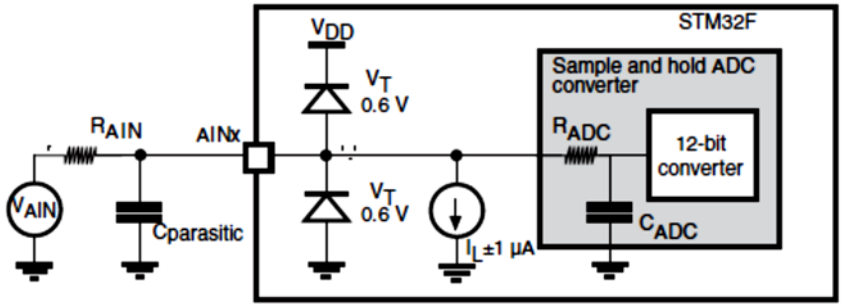
- l'échantillonnage blocage (sample and hold). Cette opération consiste à connecter l'entrée à convertir à un condensateur interne, qui va se charger à travers une résistance interne jusqu'à la tension appliquée. Ce temps est appelé temps d'acquisition ou temps d'échantillonnage (Simpling Time).

- Une fois le condensateur est chargé, l'ordre de début de conversion permet de déconnecter la tension appliquée, pour procéder à la phase ce conversion.



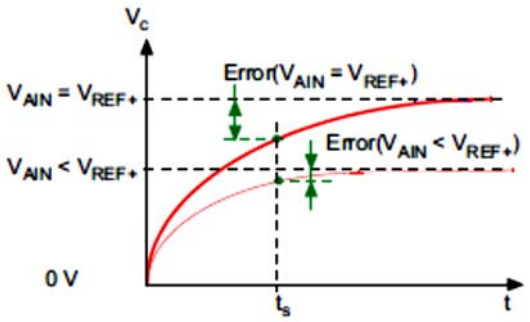
Temps d'acquisition

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 STM32F, 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	MHz
		V_{DDA} = 2,4 à 3,6 V	0.6	-	30	MHz
R_{AIN}	impédance d'entrée externe		-	-	50	kΩ
R_{ADC}			1.5	-	6	kΩ
C_{ADC}	Capacité de l'interrupteur d'échantillonnage		-	4	-	pF

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

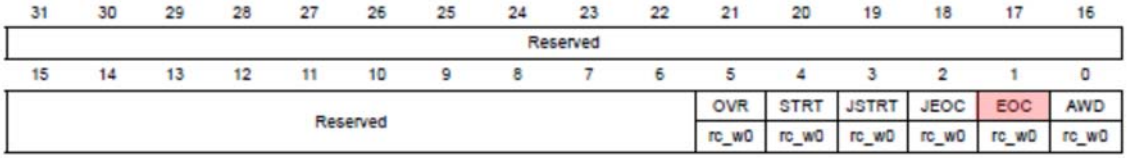
$T_s = 16 \cdot 10^3 \times 30 \cdot 10^6 \times 4 \cdot 10^{-12} \times 9 = 17,28 cycles$ soit $T_s = 28 cycles$.

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. Registres de l'ADC

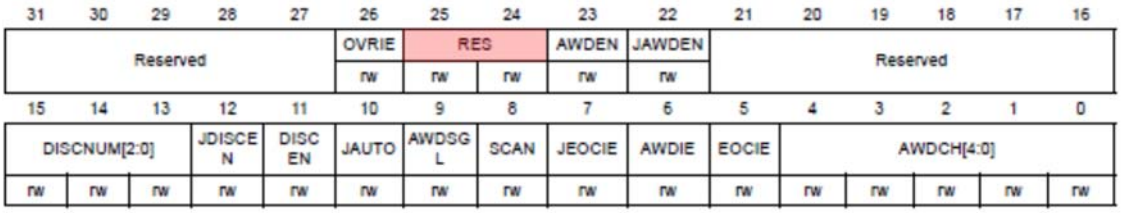
3.1. Registre ADC_SR (ADC status register)



Bit 1 : **EOC** : Regular channel end of conversion

Ce bit est mis à 1 à la fin de conversion par le matériel et peut être remis à zéro par logiciel ou après la lecture du registre de donnée ADC_DR.

3.2. Registre ADC_CR1 (ADC control register 1)



Bit 25,24 : **RES**[1-0] : Resolution

Le deux bits fixe la résolution du convertisseur

00 : conversion sur 12 bits.
01 : conversion sur 10 bits
10 : conversion sur 8 bits.
11 : conversion sur 6 bits

3.3. Registre ADC_CR2 (ADC control register 2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved	SWSTART	EXTEN		EXTSEL[3:0]				reserved	JSWSTART	JEXTEN		JEXTSEL[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved				ALIGN	EOCS	DDS	DMA	Reserved						CONT	ADON
				rw	rw	rw	rw							rw	rw

Bit 30 : SWSTART: Start conversion of regular channels

Ce bit est mis à 1 par logiciel pour démarrer une conversion, il se remet à zéro automatiquement lorsque la conversion démarre.

Bit 11 : ALIGN: Data alignment

0: résultat de conversion aligné à droite.

1: résultat de conversion aligné à gauche.

Bit 10 : EOCS: End of conversion selection

0: Le bit EOC sera positionné à la fin de conversion d'une séquence régulière

1: Le bit EOC sera positionné à la fin de conversion d'une entrée.

Bit 1 : CONT: Continuous conversion

0: conversion en mode Singulier

1: conversion en mode Continu.

Bit 0 : ADON: A/D Converter ON / OFF

0: ADC Désactivé

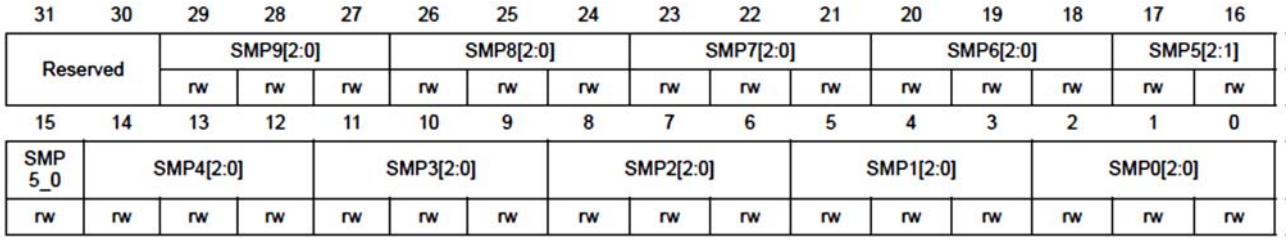
1: ADC Activé

3.4. ADC_SMPR1 et 2 (ADC sample time register 1 et 2)

ADC_SMPR1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					SMP18[2:0]			SMP17[2:0]			SMP16[2:0]			SMP15[2:1]	
					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMP15_0	SMP14[2:0]			SMP13[2:0]			SMP12[2:0]			SMP11[2:0]			SMP10[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

ADC_SMPR2



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		SMP9[2:0]			SMP8[2:0]			SMP7[2:0]			SMP6[2:0]			SMP5[2:1]	
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMP 5_0	SMP4[2:0]			SMP3[2:0]			SMP2[2:0]			SMP1[2:0]			SMP0[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

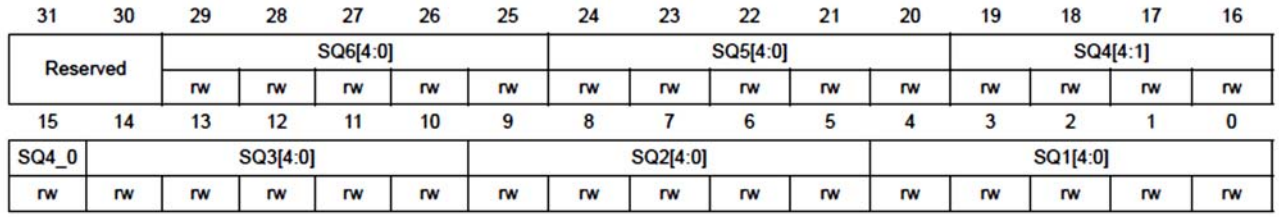
SMPx[2:0]: Channel x sampling time selection

Choix du temps de conversion en nombre de cycles.

000: 3 cycles
001: 15 cycles
010: 28 cycles
011: 56 cycles
100: 84 cycles
101: 112 cycles
110: 144 cycles
111: 480 cycles

3.5. ADC_SQRy (ADC regular sequence register y, y : 1 à 3)

ADC_SQR3



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		SQ6[4:0]					SQ5[4:0]					SQ4[4:1]			
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ4_0	SQ3[4:0]					SQ2[4:0]					SQ1[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

SQi[4:0]: i^{ème} séquence à convertir.

4. Fonction API

Le logiciel CubeMx à travers la librairie HAL Driver, génère les initialisations nécessaires. Nous nous limitons à l'étude du convertisseur ADC en mode indépendant. 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`.

//--