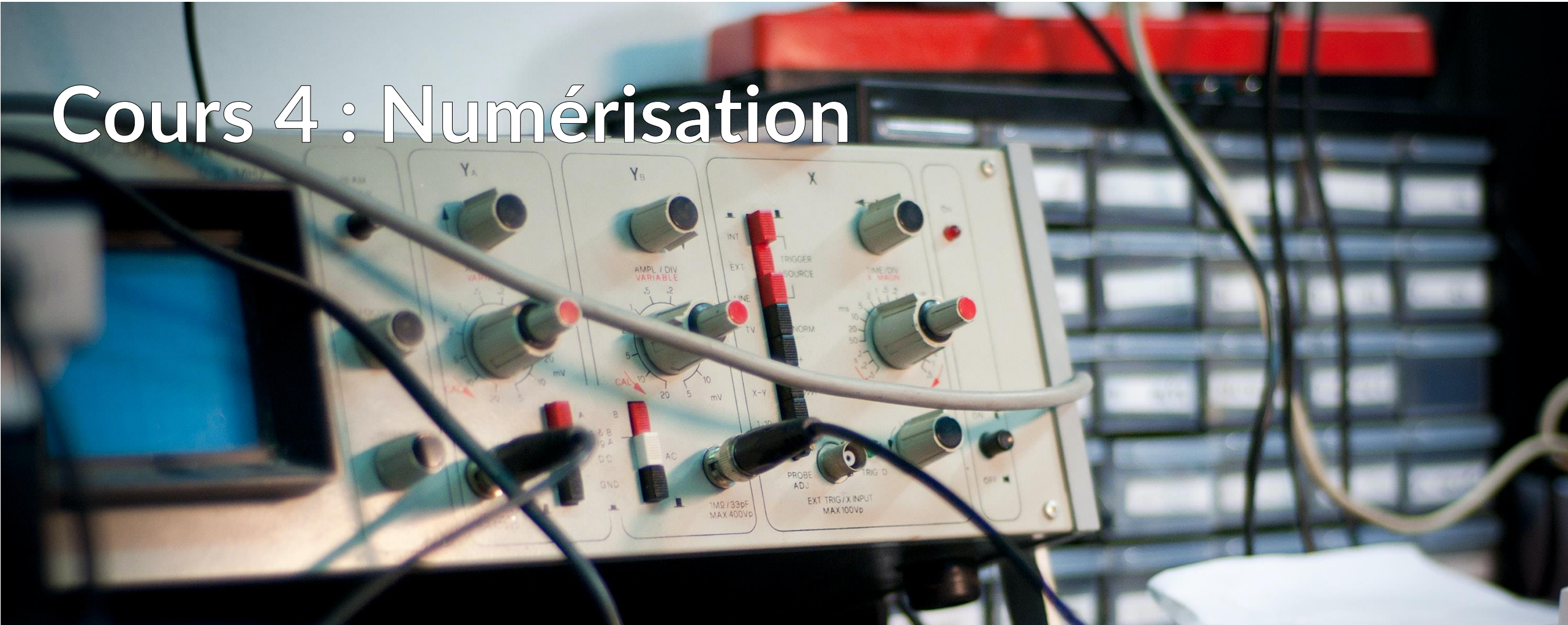


Cours 4 : Numérisation



Enjeu de la numérisation

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

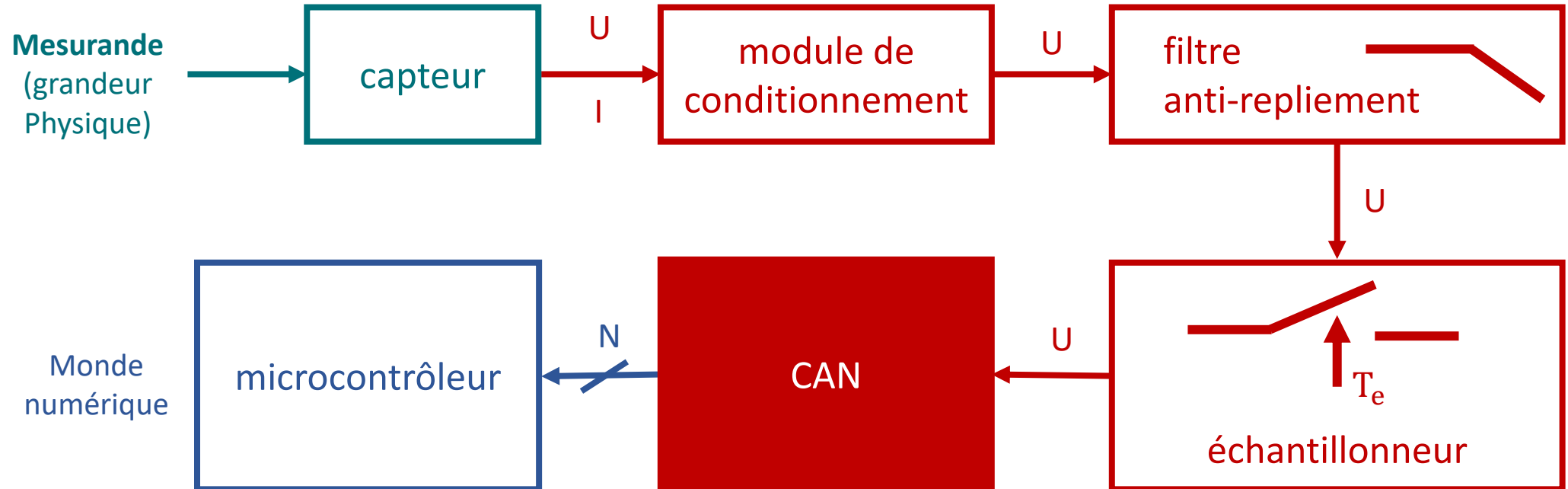
II. CAN sur Arduino

- A. `analogRead()`
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

■ La plupart des systèmes électroniques sont numériques...



→ Comment le signal est-il numérisé ?

Cours 4 : Numérisation

I. Le convertisseur analogique numérique

- A. Le convertisseur analogique numérique
- B. Résolution
- C. Temps de conversion
- D. Défauts d'un CAN
- E. Rapport signal sur bruit
- F. Les différents CAN

II. Conversion analogique-numérique avec Arduino

- A. Conversion sur l'Arduino
- B. Améliorer la précision
- C. Adapter la plage de tension
- D. CAN de l'ATMega328P
- E. Conversion plus rapide

III. Étude de cas : enregistreur portable

- A. Numérisation du signal
- B. Restitution du signal
- C. Échantillonneur bloqueur

1 Le convertisseur analogique numérique

Le convertisseur analogique numérique (1 / 4)

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

■ La conversion comporte deux étapes :

- La **quantification**
 - Conversion de la tension v à convertir en une tension multiple du quantum q , encore appelé « pas » de quantification. La tension v s'écrit alors sous la forme Nq .
- Le **codage**
 - Association d'un mot binaire au nombre N obtenu précédemment.
 - Le nombre N associé à la tension v est tel que $Nq - \frac{q}{2} \leq v \leq Nq + \frac{q}{2}$

Convertisseur Analogique - Numérique

Un Convertisseur – Analogique – Numérique (CAN, ou ADC pour *Analog to Digital Converter*) est un dispositif qui convertit une grandeur analogique, le plus souvent une tension v , en un mot numérique de n bits $b_0b_1...b_n$, sa valeur décimale N .

Codage -> associer une valeur d'amplitude en bit, à chaque échantillon.

Le convertisseur analogique numérique (2 / 4)

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

1ère étape : la quantification

- Il s'agit de «découper» l'amplitude du signal en valeurs discrètes : pour N bits, on a 2 valeurs possibles.
- Le nombre de bits sur lequel est codé l'information est appelée «résolution» du CAN

EXEMPLE : Quantification d'un signal sur 2 bits

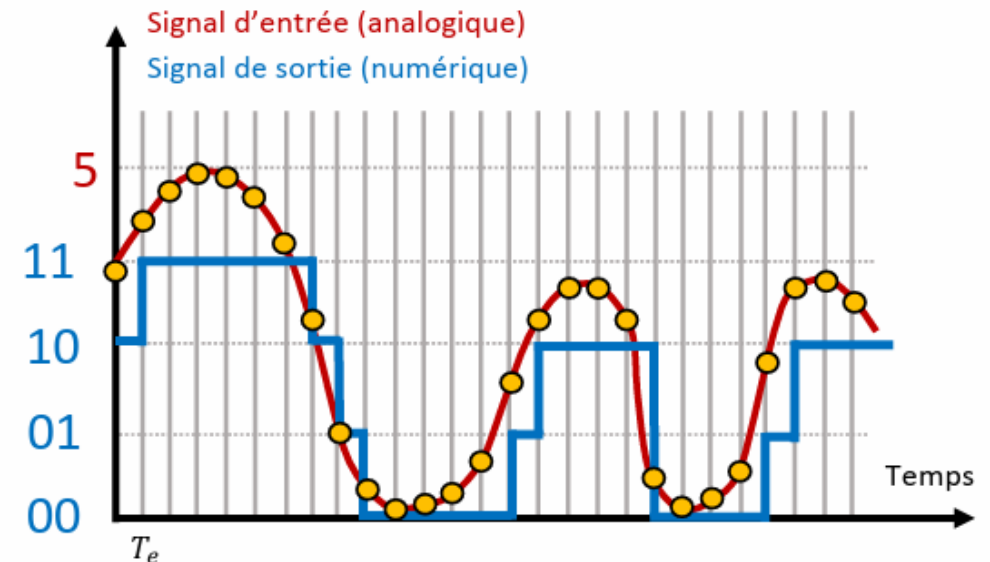
La plage du CAN est de 5 V.

La résolution du CAN est $n=2$ bits

$n = 2$ donc $2^2 = 4$ états : 00, 01, 10 et 11

Le pas :

$$q = \frac{5}{2^2} = \frac{5}{4} = 1,25 \text{ V}$$



Le convertisseur analogique numérique (3 / 4)

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

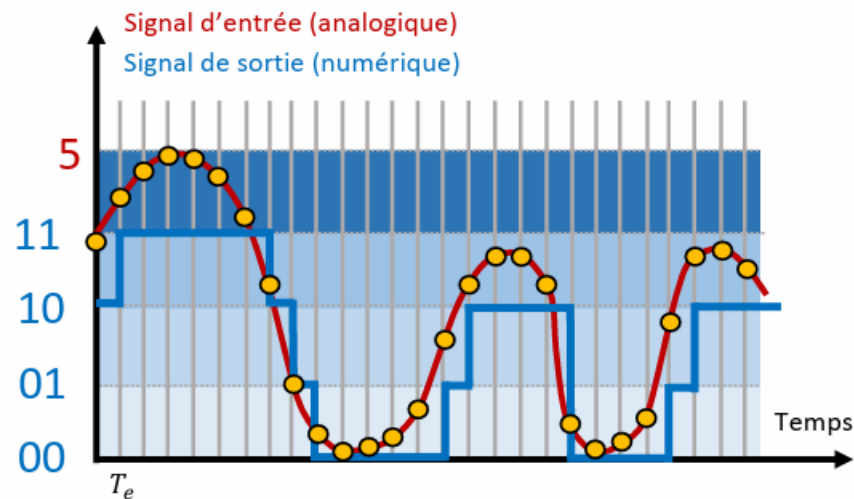
III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

2nde étape : le codage

- On associe à chaque intervalle un nombre binaire.
- Et pour chaque échantillon, on associe le nombre binaire correspondant

EXEMPLE : Codage d'un signal sur 2 bits



Pour une tension entre :

- 3,75 et 5 V → 11
- 2,5V et 3,75 V → 10
- 1,25 V et 2,5 V → 01
- 0 et 1,25 V → 00

Signal numérique résultant :

10	11	11	11	11	11	11	10	01	00	00	...
----	----	----	----	----	----	----	----	----	----	----	-----

Le convertisseur analogique numérique (4 / 4)

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

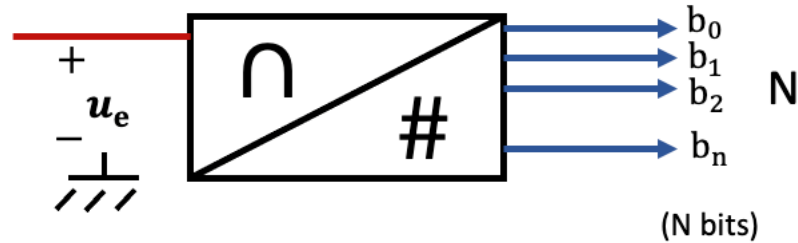
II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

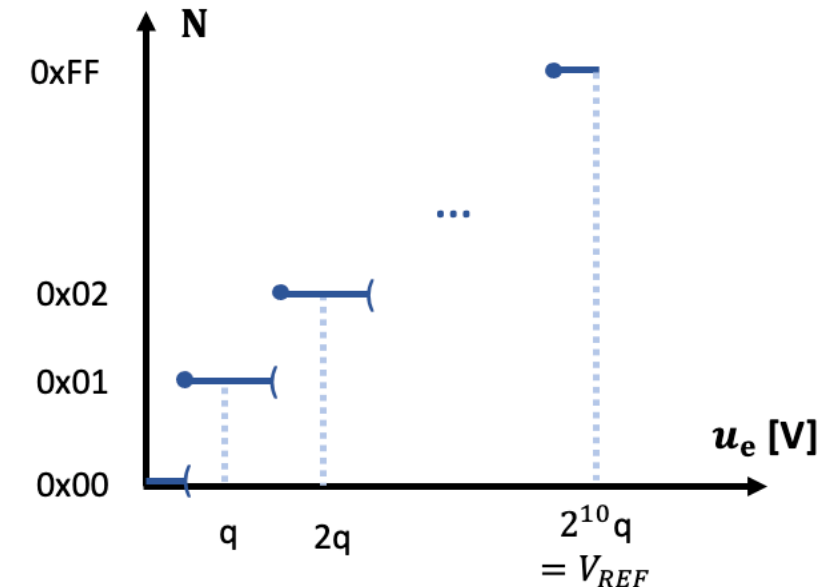
Pour une entrée analogique sur Arduino Nano, la résolution est sur 10 bits, donc valeurs N de 0 à 1023



■ La caractéristique de transfert d'un CAN représente le mot numérique de sortie $b_0 b_1 \dots b_n$ en fonction de la tension analogique d'entrée u_e .

■ La caractéristique est formée de paliers de largeur q , sauf le premier et le dernier qui ont une largeur de $q/2$ et $1,5 q$.

■ On appelle **quantum** ou LSB (*Less Significant Bit*) la variation de tension pour passer d'un palier N à un palier N+1



Quantum

$$q = \frac{V_{REF}}{2^n}$$

Résolution

I. Le CAN

- A. Conversion
- B. **Résolution**
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

- La quantification est liée à la **résolution** N du CAN : le nombre de bits sur lesquels le signal de sortie est codé.
- Plus la résolution d'un CAN est élevée, plus la sortie numérique est une image précise du signal analogique d'entrée.

EXEMPLE 1

Quantum d'un CAN en fonction de sa résolution pour $V_{REF} = 5\text{ V}$ et 3 V

V_{REF}	8 bits	10 bits	12 bits	14 bits
5 V	19,5 mV	4,8 mV	1,22 mV	305 μV
3 V	11,7 mV	2,93 mV	732 μV	183 μV

→ **Abaisser la tension de référence ou augmenter la résolution d'un CAN le rendent plus précis.**

EXEMPLE 2

Déterminer le nombre d'octets qu'occupe 1h de musique sur un CD audio ($f_e = 44,1\text{ kHz}$) en stéréo.

- Le nombre d'octets Nb pour décrire 1 s de musique est :

$$Nb = f_e \frac{n}{8} p \text{ avec } p \text{ le nombre de voies (mono : 1, stéréo : 2)}$$

- Pour 1h de musique en stéréo sur un CD, cela donne donc :

$$Nb = 44\,100 \cdot \frac{16}{8} \cdot 2 \cdot 3\,600 = 635\,040\,000 \text{ octets}$$

Puisque 1 ko = 1 024 octets et 1 Mo = 1024 ko, **Nb = 605,6 Mo**

support	N
CD	16
DVD	24
téléphonie	8
Radio numérique	8
Oscilloscope numérique classique	8
CAN du microcontrôleur de l'Arduino Uno	10

Temps de conversion

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

- Après l'échantillonnage, une phase de maintien est nécessaire pour laisser le temps au CAN de convertir.
- Son temps doit être supérieur au temps de conversion t_{CONV} .

$$T_e > t_{conv}$$

Incidence du temps de conversion sur le choix de F_e

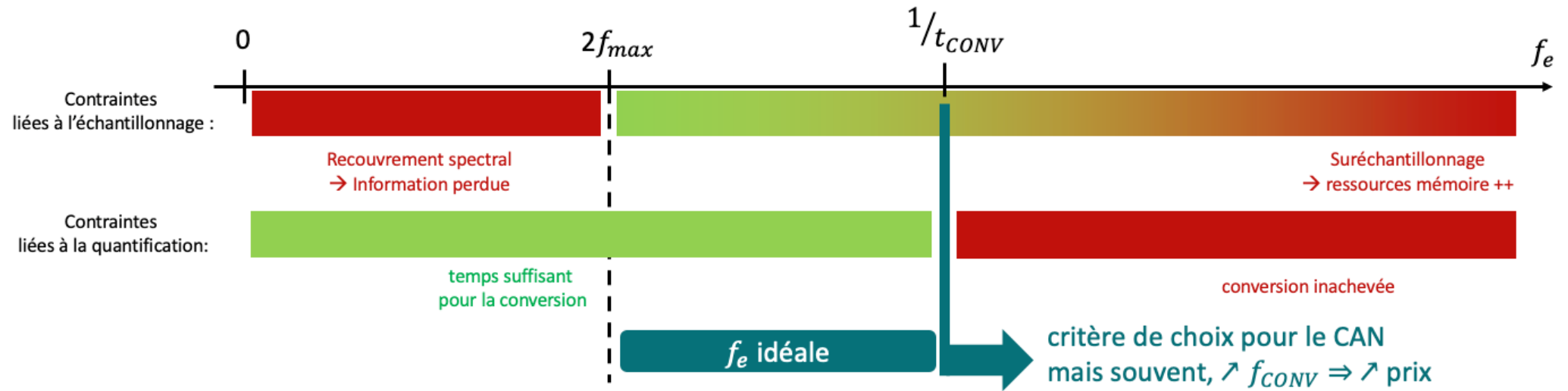
$$F_e < 1/t_{CONV}$$

II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur



Défauts d'un CAN

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts**
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

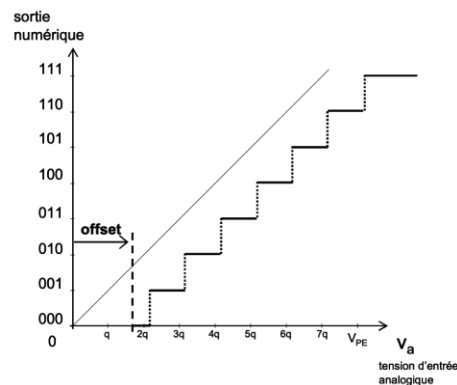
- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

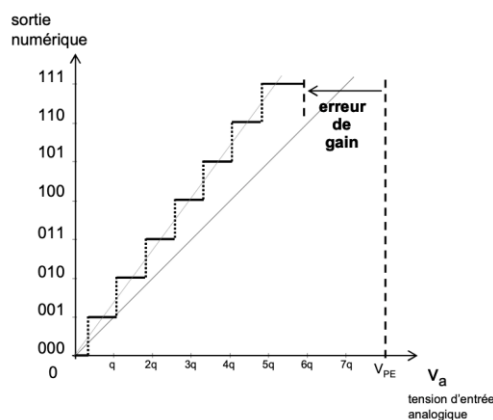
Erreur d'offset

- décalage horizontal de la caractéristique de transfert ;
- exprimée usuellement en LSB.



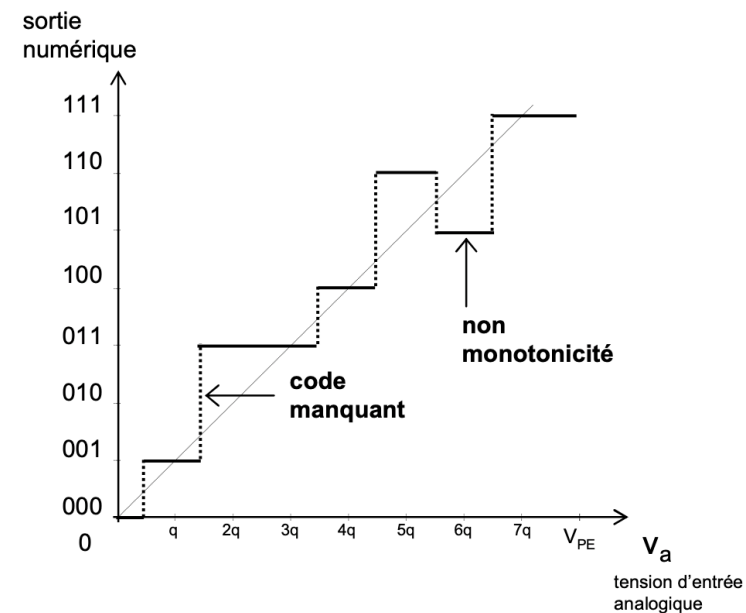
Erreur de gain

- Écart entre la pente de la caractéristique idéale de transfert et la pente de la caractéristique réelle obtenue par régression linéaire des centres des paliers.



Erreur de code manquant

- un des codes de sortie n'apparaît jamais quel que soit la valeur de la tension analogique d'entrée.



Erreur de non monotonicité

- Les codes numériques en sortie ne se succèdent pas de façon croissante pour un signal d'entrée croissant.

Erreur de quantification

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

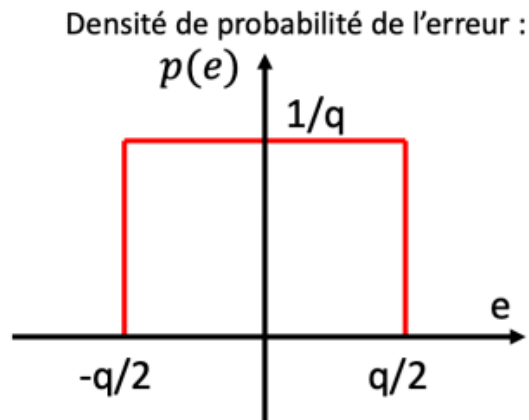
- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

■ La largeur des paliers n'est pas nécessairement identique : tout se passe comme s'il y avait une tension résiduelle appelée **bruit de quantification** b_q .

■ L'**erreur de quantification** ε_q est assimilée à un bruit dont on calcule la puissance moyenne pour une résistance normalisée de 1Ω :



$$\begin{aligned}\varepsilon_q^2 = P_{BRUIT} &= \int_{-\infty}^{\infty} e^2 p(e) de = \int_{-\frac{q}{2}}^{\frac{q}{2}} e^2 p(e) de = \int_{-\frac{q}{2}}^{\frac{q}{2}} e^2 \frac{1}{q} de \\ 2 \cdot \int_0^{\frac{q}{2}} e^2 \frac{1}{q} de &= \frac{2}{q} \left[\frac{e^3}{3} \right]_0^{\frac{q}{2}} = \frac{2}{q} \frac{e^3}{3 \cdot 8} = \frac{q^2}{12}\end{aligned}$$

Erreur de quantification

$$\varepsilon_q^2 = \frac{q^2}{12}$$

■ Résultat valable pour une résolution $N \geq 6$. Sinon, on prend par défaut $P_{BRUIT} = \frac{q^2}{3}$

Rapport signal sur bruit

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit**
- F. Structures

II. CAN sur Arduino

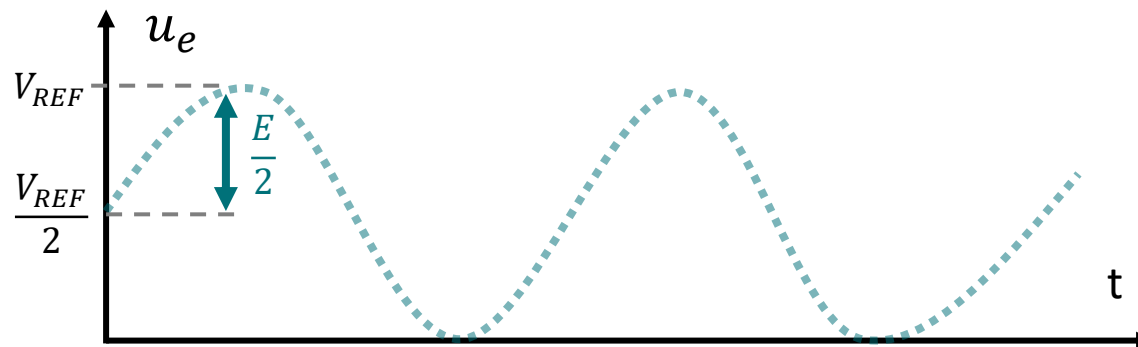
- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

■ Le **rapport signal sur Bruit** γ (SNR, Signal over Noise Ratio) d'un CAN idéal est défini pour une entrée sinusoïdale pleine échelle. C'est le quotient entre la valeur efficace du signal et l'erreur de quantification : $\gamma = \frac{u_{EFF}}{\varepsilon_q}$

■ On redresse (offset) une sinusoïde que l'on place en entrée afin qu'elle oscille entre 0 et V_{REF} :



$$u_{EFF} = \frac{V_{REF}}{2\sqrt{2}} = \frac{2^N q}{2\sqrt{2}} = \frac{2^{N-1} q}{\sqrt{2}}$$

$$\text{d'où } \gamma = \frac{2^{N-1} q / \sqrt{2}}{q / \sqrt{12}} = \frac{2^{N-1} q \sqrt{12}}{\sqrt{2} q} = \sqrt{6} \cdot 2^{N-1}$$

$$\text{et } \gamma_{dB} = 20 \log(\gamma) = 6,02 N + 1,76$$

Rapport signal sur bruit

$$\gamma_{dB} = 6,02 N + 1,76$$

→ Le SNR d'un CAN augmente avec sa résolution (gain de 6 dB par bit supplémentaire).

CAN Flash (1 / 3)

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

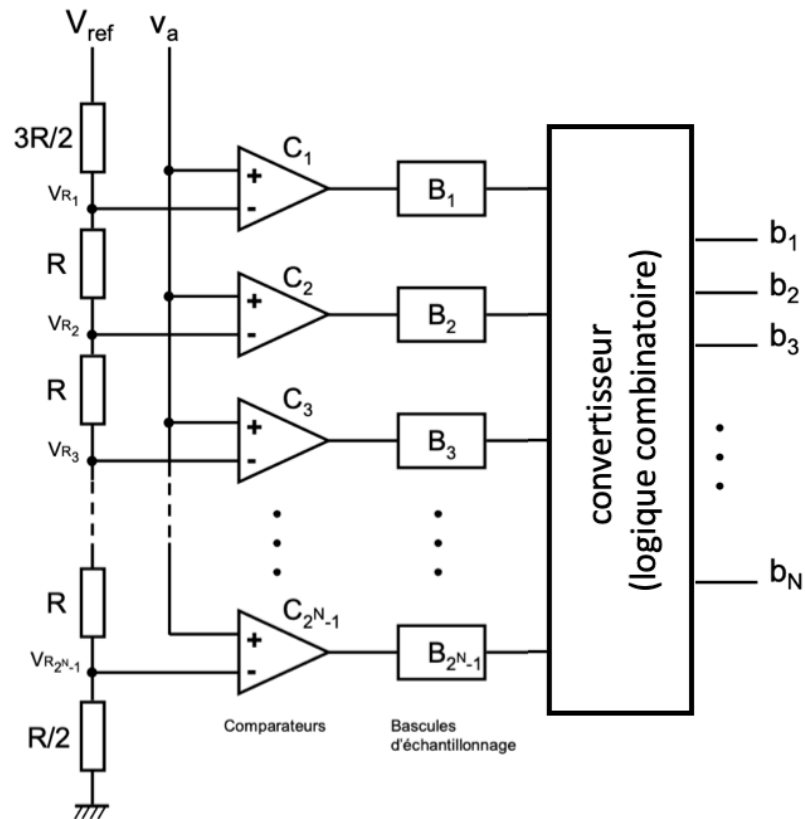
II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

■ Pour un CAN de résolution N bits, l'entrée analogique à convertir est **comparée simultanément à $2^N - 1$ tensions de seuils**.



Pas de rétroaction sur les entrées non-inverseuses
 → Régime saturé → comparateurs

■ Ces tensions de seuil sont obtenues par des ponts diviseurs comportant **$2N$ résistances** connectées en série entre V_{REF} et la masse.

■ La résistance connectée à la masse est prise égale à $R/2$ et celle connectée à V_{REF} égale à $3R/2$.

■ Le CAN comporte **$2N - 1$ comparateurs** (un pour chaque seuil à comparer), **$2N - 1$ bascules d'échantillonnage** et une **logique de conversion**.

- Les comparateurs de Ck à C_{2N-1} dont les tensions de seuils associées sont **inférieures** à v_a délivrent en sortie un **1** logique ;
- les comparateurs de $Ck - 1$ à C_1 dont les tensions de seuils associées sont **supérieures** à v_a délivrent en sortie un **0** logique.

■ La conversion est réalisée en un seul cycle d'horloge, ce type de convertisseur est donc par essence extrêmement rapide.

■ Le coût résultant en termes de surface ($2^N - 1$ comparateurs, $2^N - 1$ bascules), pour une résolution élevée limite leur emploi à une douzaine de bits.

CAN Flash (2 / 3)

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

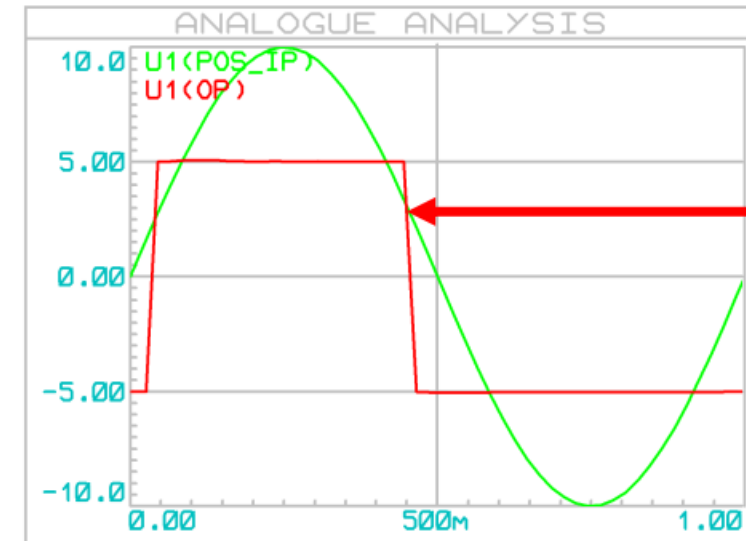
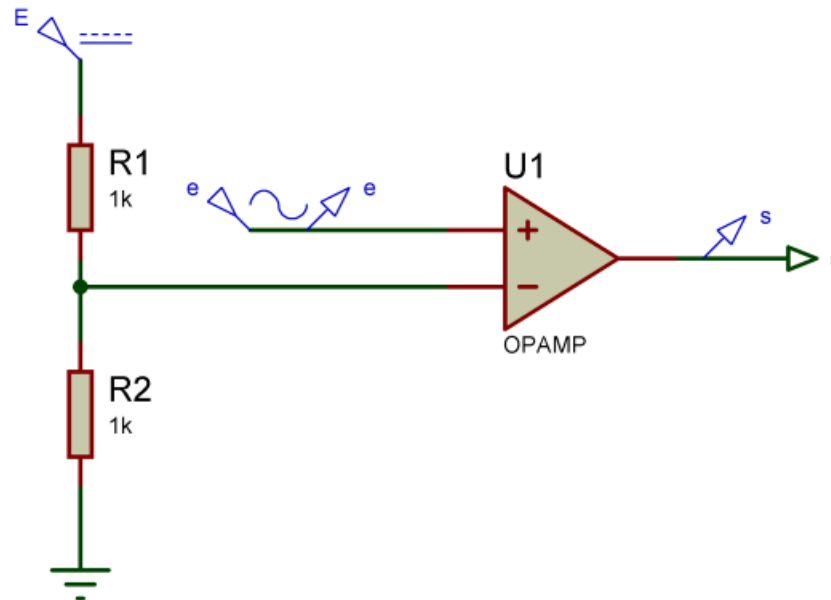
EXEMPLE 1 : Comparateur simple à pont diviseur de tension

II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur



$$\text{Seuil } \frac{R_2}{R_1 + R_2} E$$

CAN Flash (3 / 3)

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

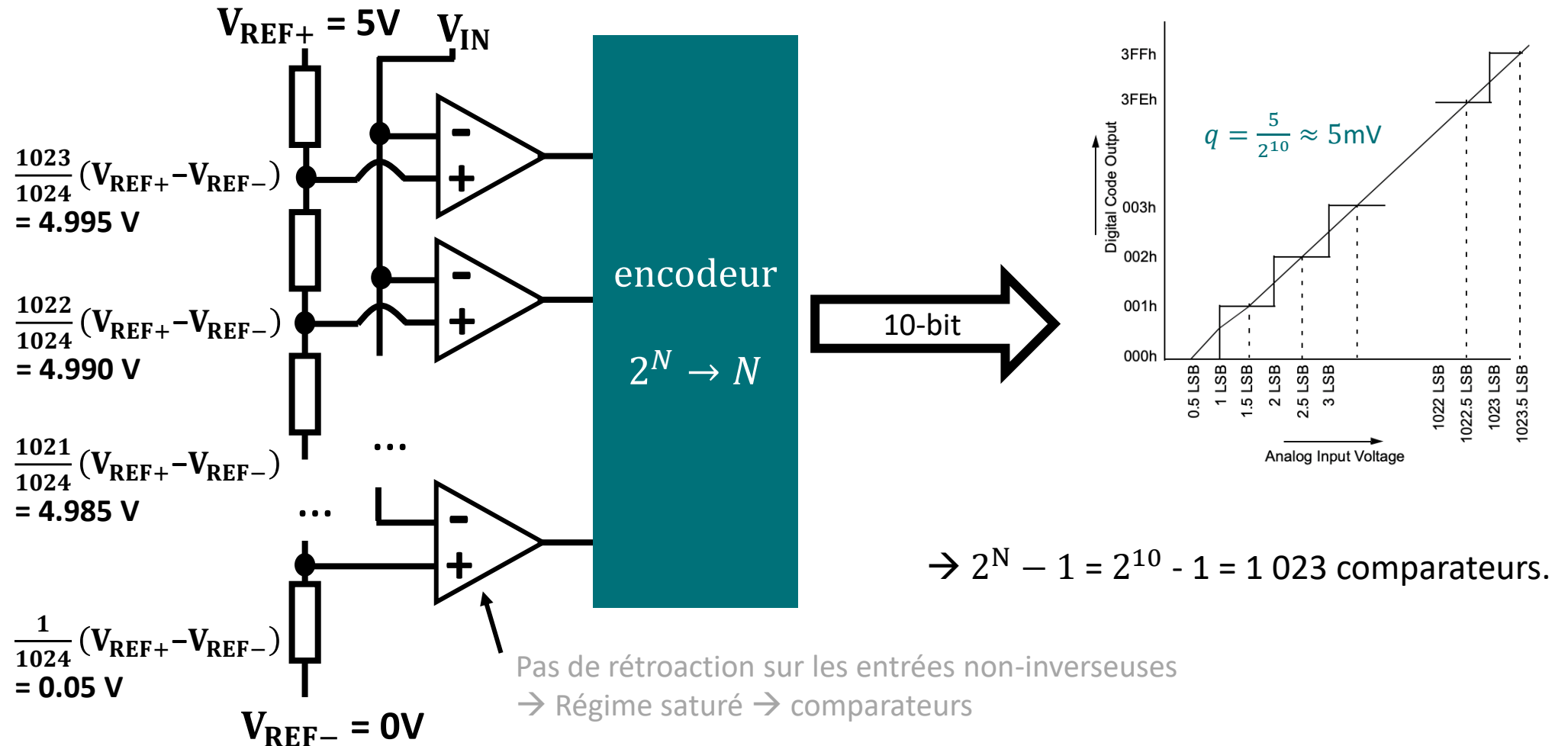
- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

EXEMPLE

Le CAN du microcontrôleur PIC18F a une résolution de 10 bits. De combien d'AOP comparateur dispose-t'il ?



CAN à Approximations successives (SAR)

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

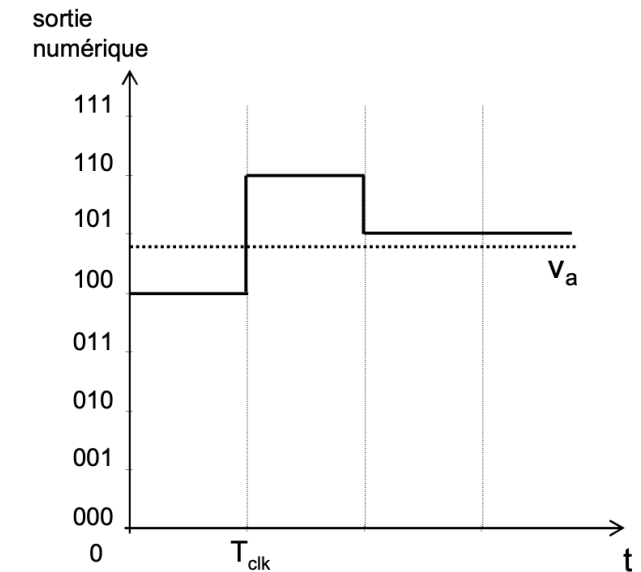
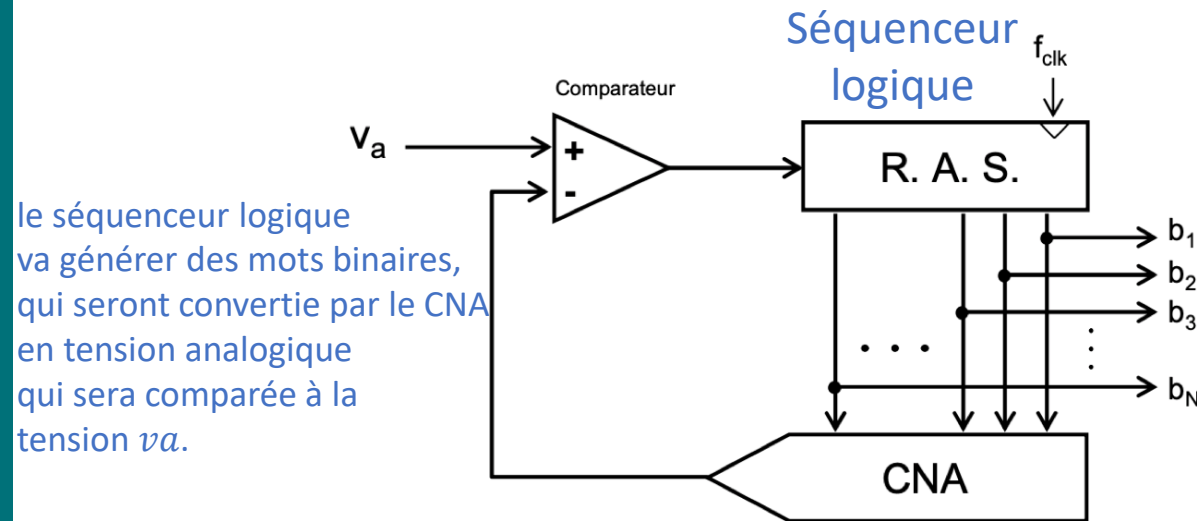
II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

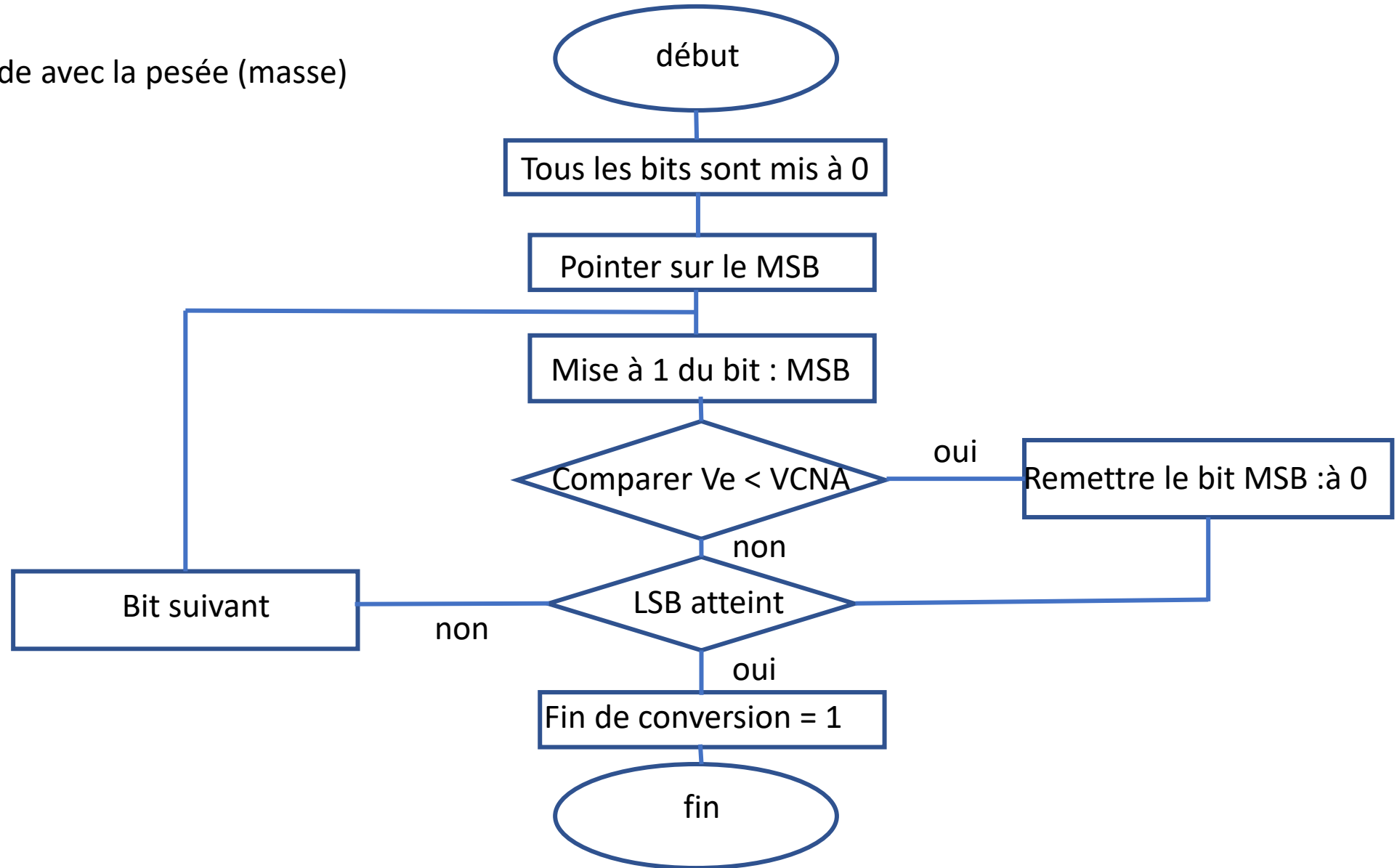
- Le CAN possède une boucle de rétroaction, constituée d'un CNA de même résolution et d'un comparateur qui commande un **Registre à Approximation Successive** (RAS, qui donne son nom à cette architecture).



- Le principe de conversion est basé sur une recherche du code de sortie par **dichotomie** :
 - à chaque coup d'horloge l'intervalle de recherche est divisé par 2.
 - En début de conversion tous les bits de sortie (RAS et CAN) sont positionnés à zéro à l'exception du MSB, b_1 , qui est fixé à '1'.
 - Le mot binaire correspondant (100...0) est présenté au CNA qui délivre en sortie une tension $V_{REF}/2$ et cette dernière au comparateur
 - Si v_a est inférieur à $V_{REF}/2$ alors b_1 passe à zéro, dans le cas contraire il reste à un ; dans les deux cas il s'agit de la valeur finale
 - Tous les bits de sortie jusqu'au LSB sont testé successivement sur le même principe.
- En première approximation, le temps de conversion est $t_{CONV} = N \cdot T_{CLK}$
- C'est une architecture de conception ancienne mais encore très répandue.

CAN à Approximations successives (SAR)

Similitude avec la pesée (masse)



I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

- A. `analogRead()`
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

CAN à Approximations successives (SAR)

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

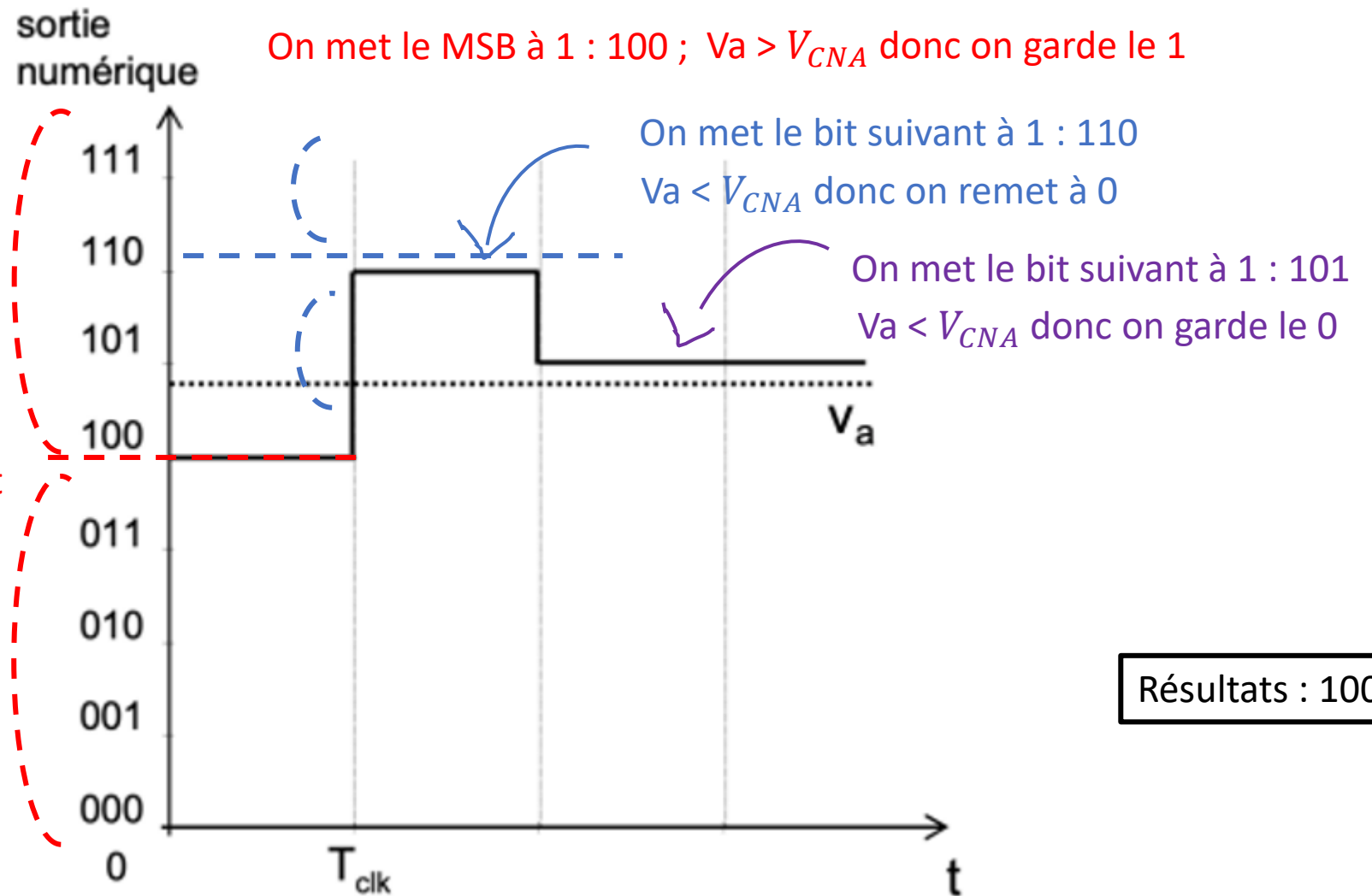
III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

EXEMPLE

$n = 3$ bits

On procède par dichotomie en sélectionnant le MSB à 1, et en divisant l'intervalle en deux intervalles égaux.



Comment choisir un CAN ?

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

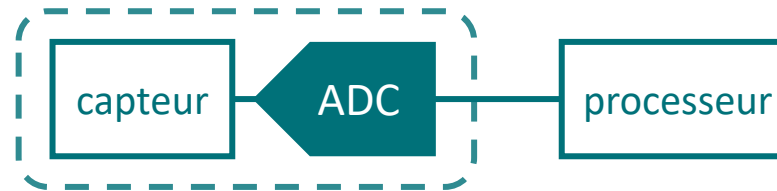
III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

■ Le CAN peut être :

• Intégré au capteur

- Capteur complexe qui donne directement un résultat numérique
- Meilleure intégration, souvent un MEMS (MicroSystème ElectroMécanique)



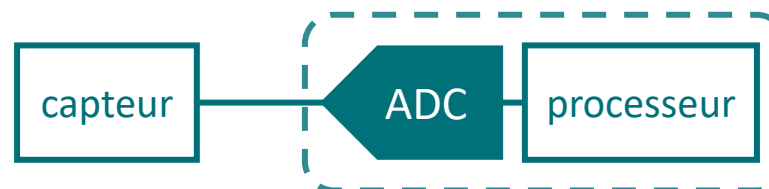
• Externe

- Recherche de performances (résolution élevée, temps de conversion court, etc.)



• Intégré à l'unité de calcul numérique (microcontrôleur, microprocesseur, FPGA, etc.)

- Cas général, pas les meilleures performances
- Proche de l'unité de calcul et donc meilleure synchronisation



Comparatif des CAN

I. Le CAN

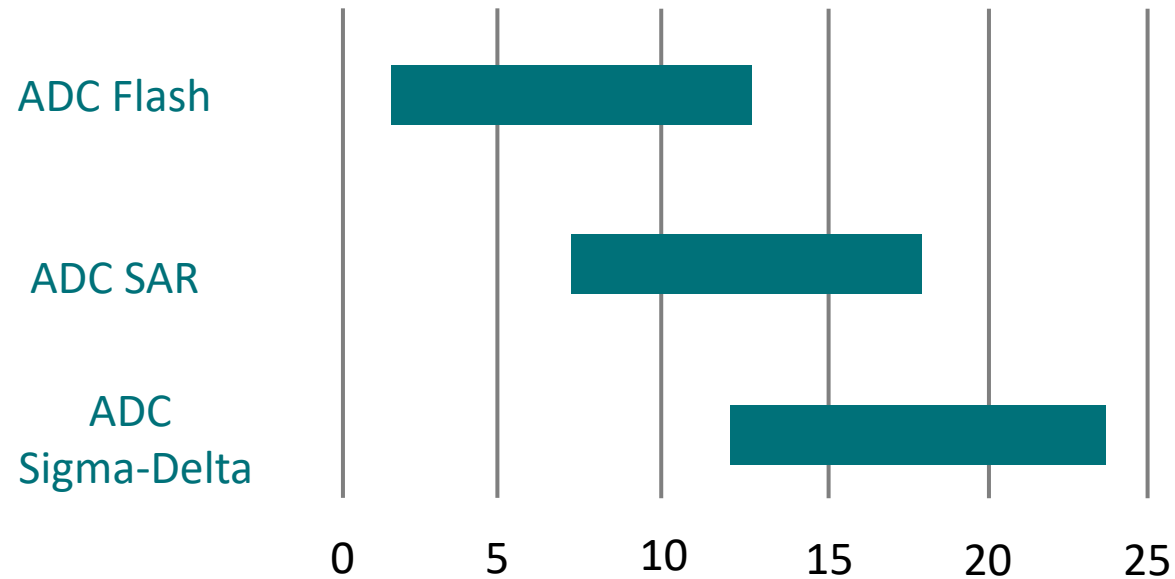
- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur



Temps de conversion	Consommation
$\sim ns$	1 W
$\sim \mu s$	10 mW
$\sim s$	100 mW

2 Conversion analogique-numérique avec Arduino

Conversion analogique numérique avec Arduino

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

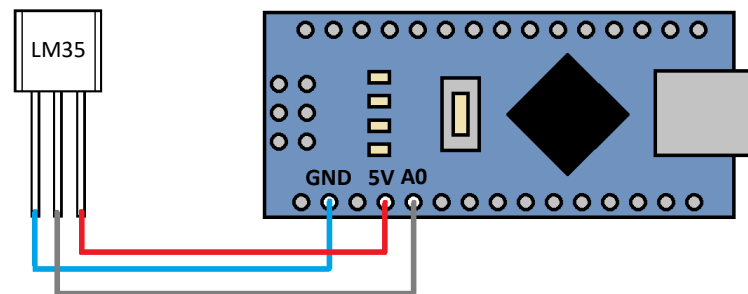
- A. `analogRead()`
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

- La plupart des microcontrôleurs, tels que l'ATMega328P de l'Arduino Nano, possèdent un CAN intégré.
- Celui de l'Arduino Nano a les caractéristiques suivantes :
 - Résolution de **10 bits** → résultats de 0 à $2^{10} - 1 = 1023$
 - Tension de référence par défaut : **5 V** → quantum $q = \frac{5}{2^{10}} = \mathbf{4,88\ mV}$
 - $t_{CONV} \approx \mathbf{115\ \mu s}$ donc $f_{e,max} \approx 9\ kHz$
 - Laisser une broche non connectée revient à avoir une antenne, et donc mesurer une tension sur ces broches non connectées retourne des valeurs de l'ordre de 300 à 500

EXEMPLE



```
int adcvalue ;
float temperatureCelsius;

void setup()
{
    pinMode(A0, INPUT);
}

void loop()
{
    adcvalue = analogRead(A0); // retourne un int de 0 à 1023
    temperatureCelsius = adcvalue * 5.0 / 1023.0 * 100.0 ;
    // conversion en volt puis conversion (10 mV = 1°C)
}
```

Float : 32 bit
Int : 16 bit

Circuit interne (1 / 3)

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

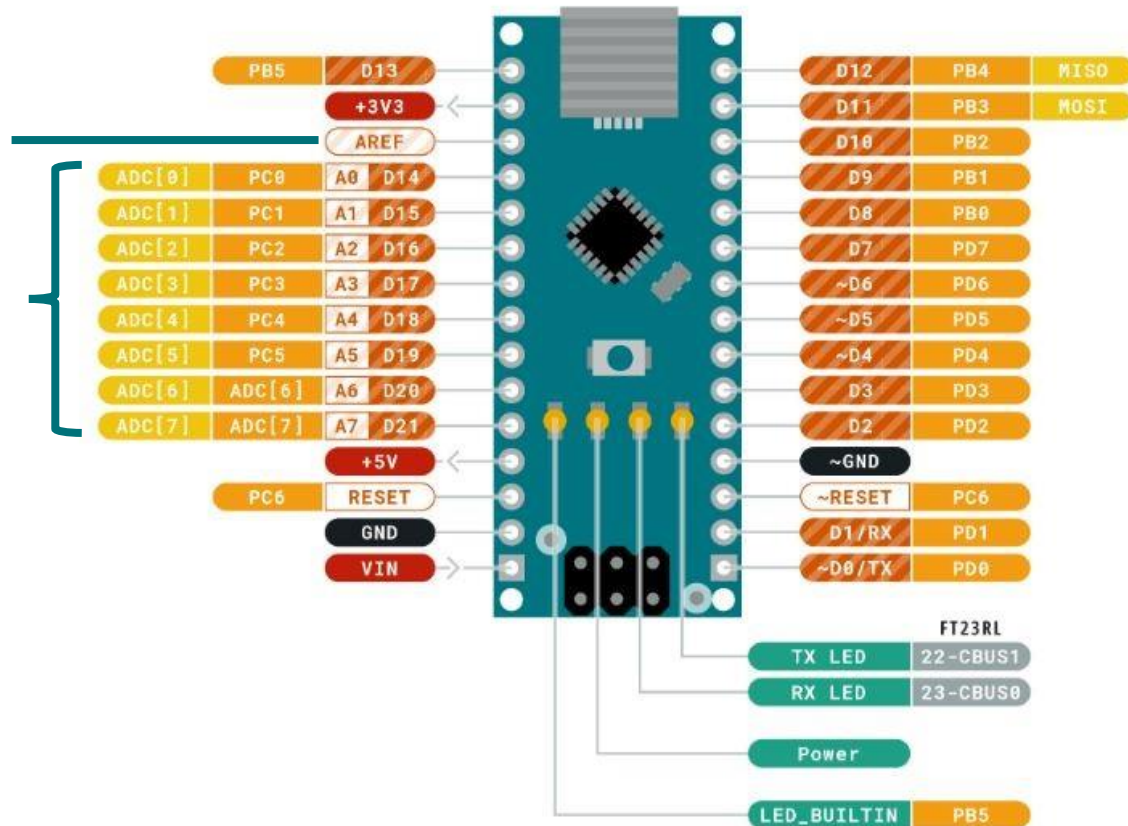
- A. `analogRead()`
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

Tension de référence

8 pins analogiques



Niveau d'abstraction



Carte électronique

Circuit interne (2 / 3)

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

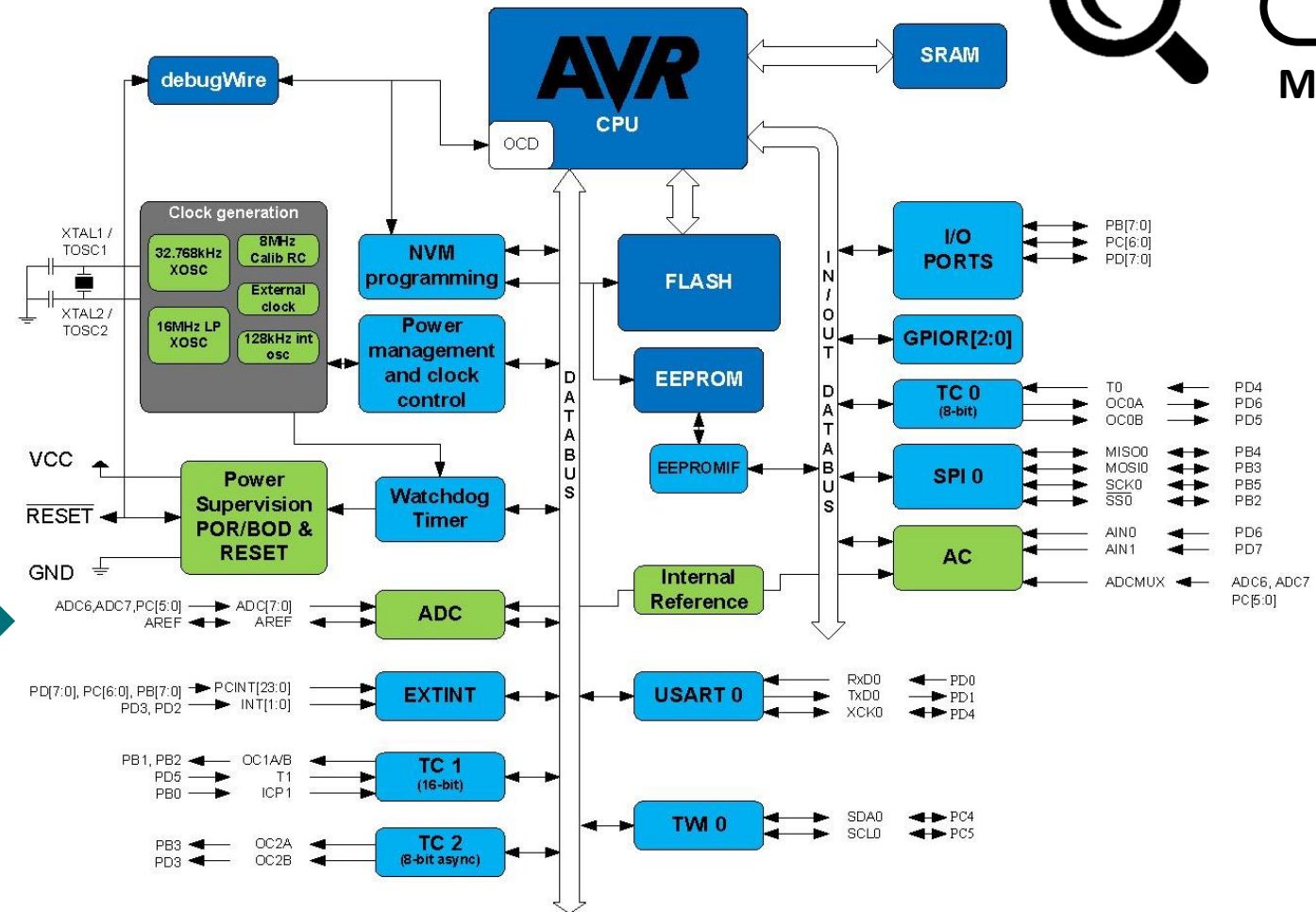
II. CAN sur Arduino

- A. `analogRead()`
- B. Améliorer la précision
- C. Améliorer la rapidité

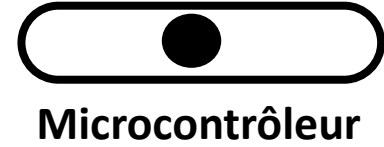
III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

1 module ADC



Niveau d'abstraction



Circuit interne (3 / 3)

I. Le CAN

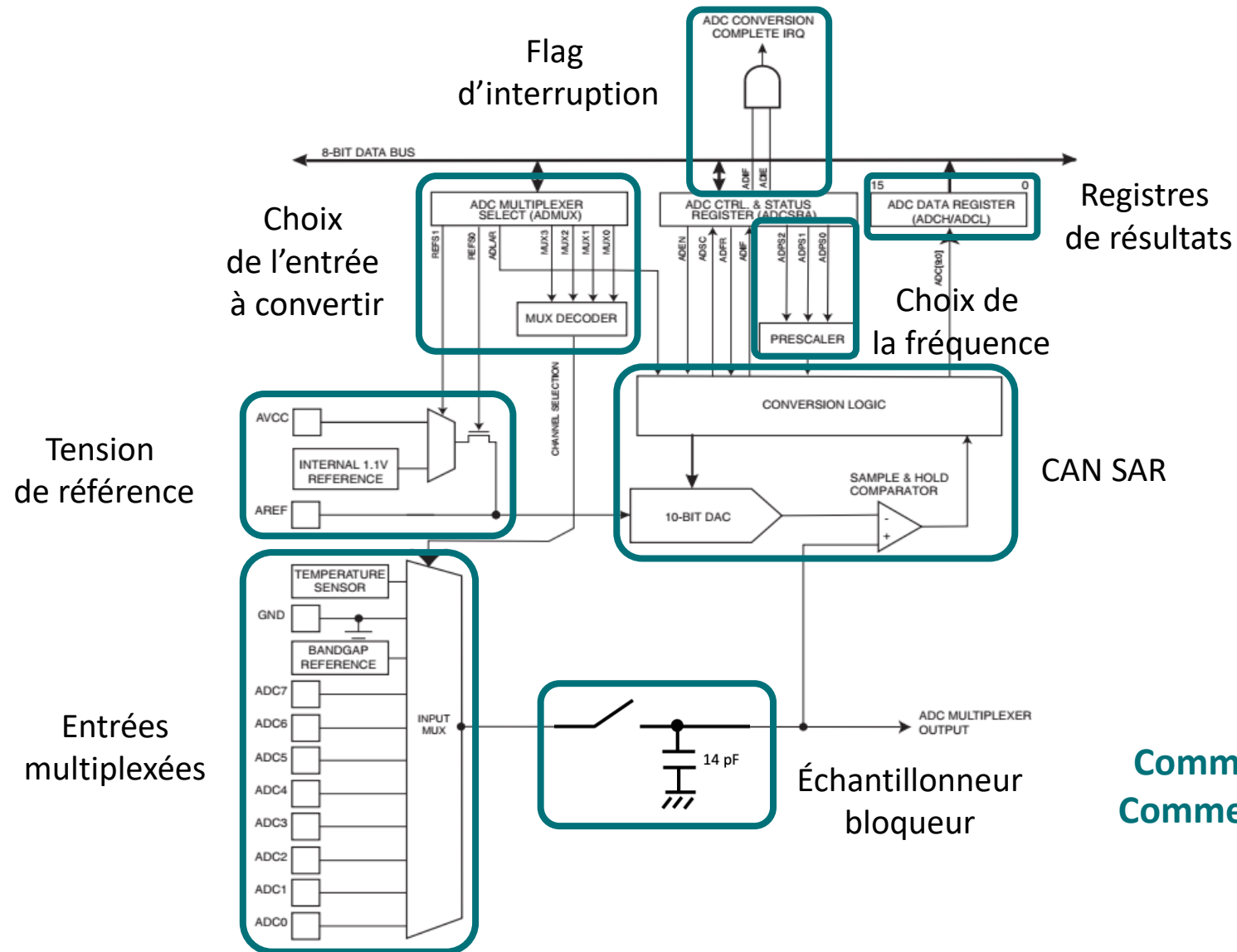
- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

- A. `analogRead()`
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur



Niveau d'abstraction



Module ADC

Registre ADCH
ADC result High

0 0 0 0 0 0 □ □

Registre ADCL
ADC result Low

□ □ □ □ □ □ □ □

Comment améliorer la précision ?
Comment échantillonner plus vite ?

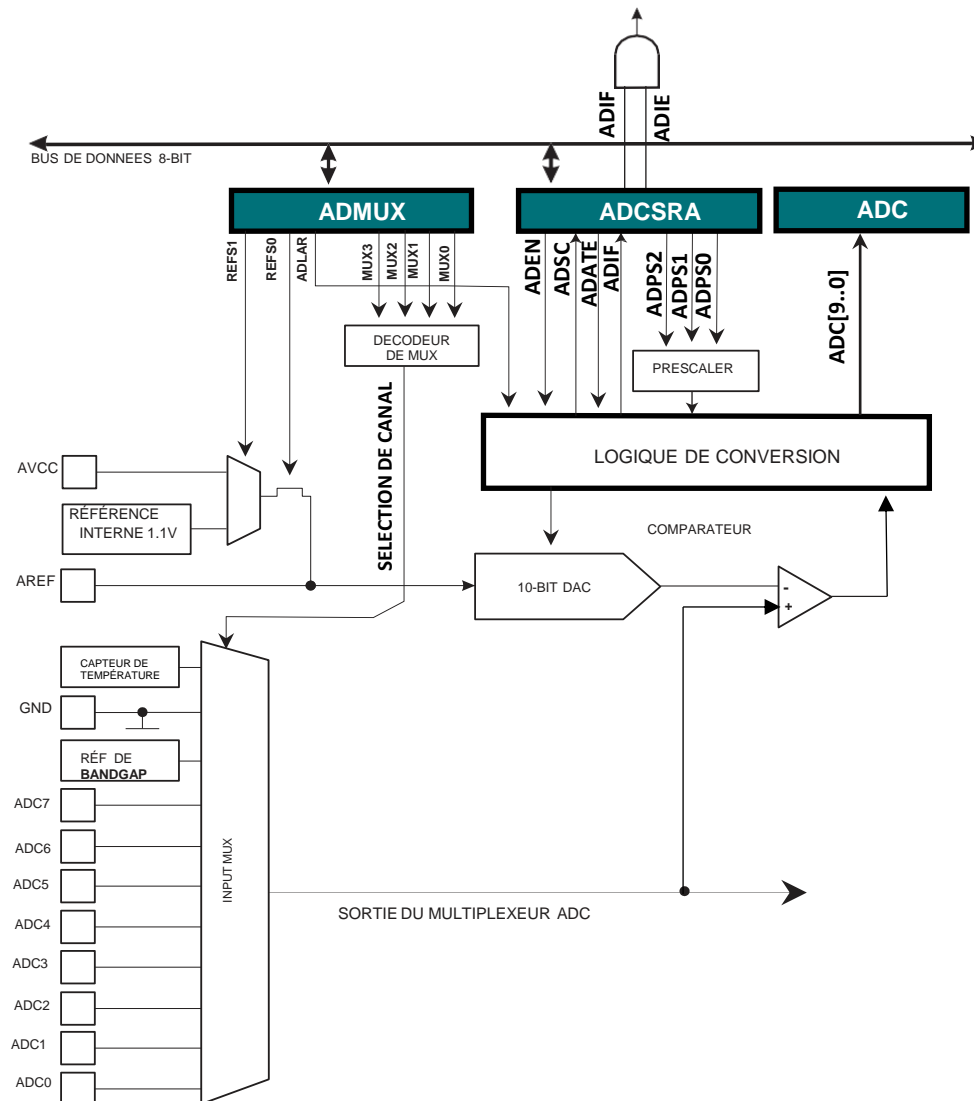
- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

- A. `analogRead()`
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur



Niveau d'abstraction



Module ADC

AnalogRead ()

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

3 registres à connaître :

Registre
ADMUX

Registre
ADCSRA

Registres
de résultats
ADC



Vitesse d'exécution



$$t_{conv} \approx 115 \mu s$$

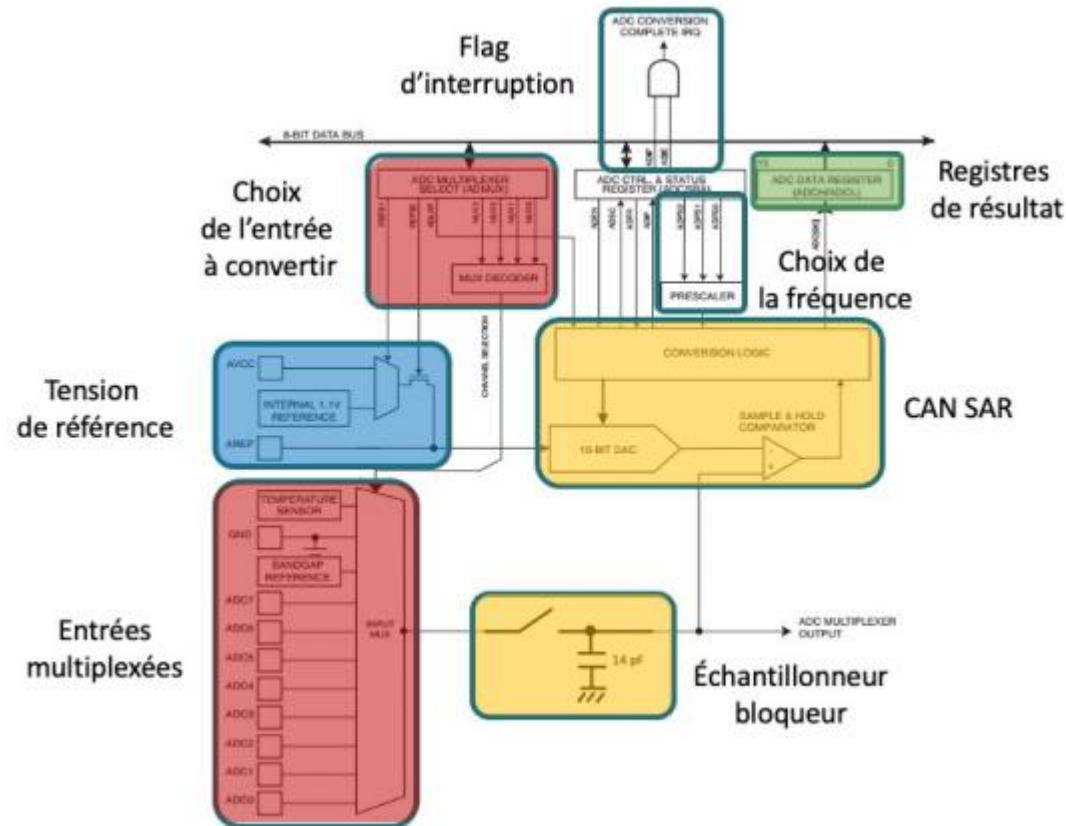
$$f_{e,max} \approx 9 kHz$$

II. CAN sur Arduino

- A. `analogRead()`
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur



Registre ADCH
ADC result High

0 0 0 0 0 0 0 0

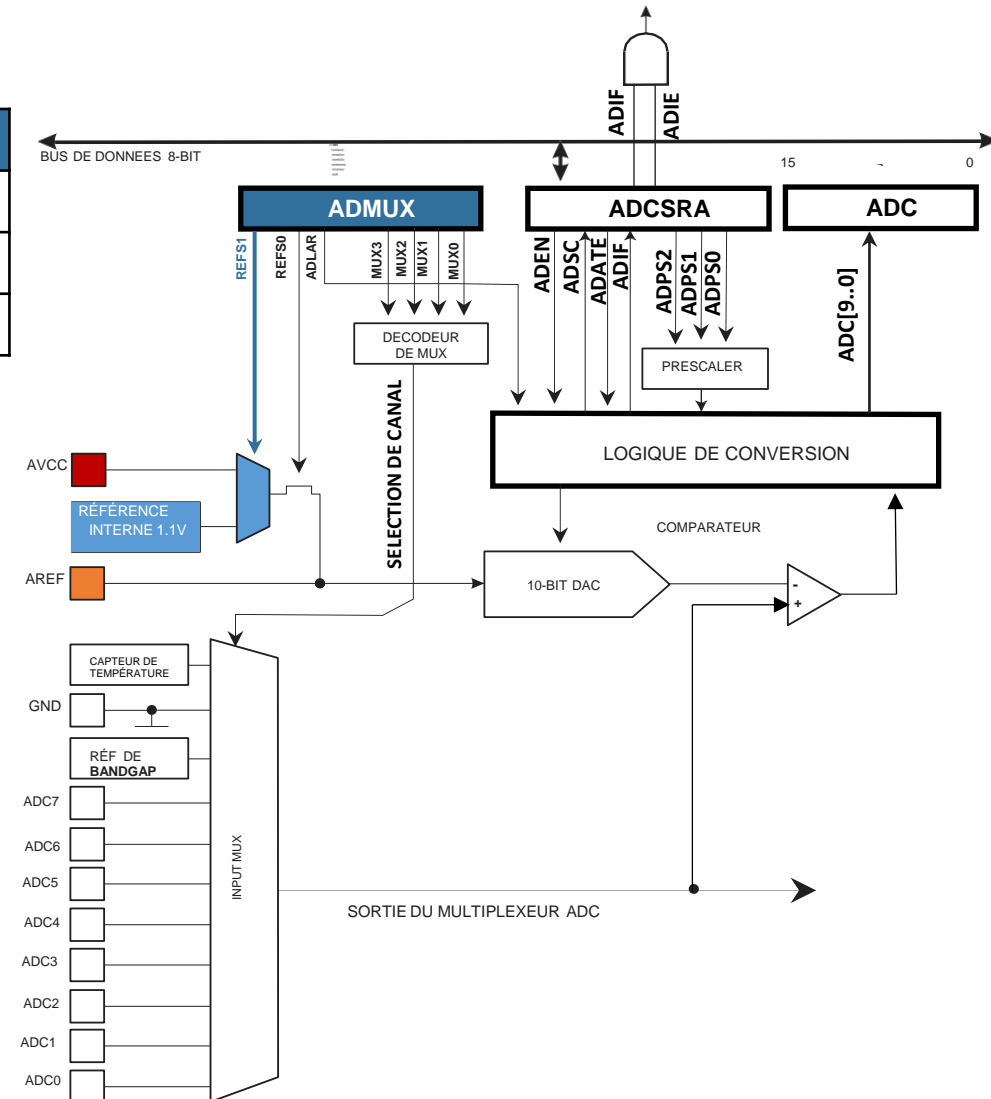
Registre ADCL
ADC result Low

0 0 0 0 0 0 0 0

Résultats sur 10 bits
(À compléter sur
2 registres de 8 bits)

- A. Numérisation
- B. Restitution
- C. Bloqueur

REF1	REF0	Action
0	0	AREF, Vref interne désactivée
0	1	AVCC avec condensateur externe à la broche AREF
1	0	Réservé
1	1	Tension interne de 1,1 V avec un condensateur externe à la pin AREF



Choix de l'entrée à convertir (2/4)

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

- A. `analogRead()`
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

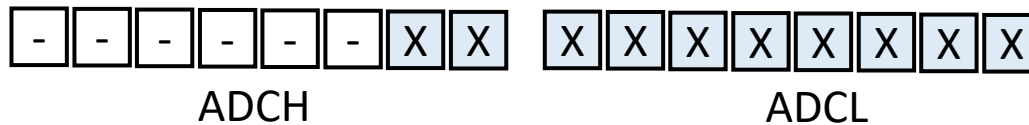
Registre de multiplexage des entrées analogiques : ADMUX

Bit	7	6	5	4	3	2	1	0
0x7C	REFS1	REFS0	ADLAR	-	MUX3	MUX2	MUX1	MUX0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

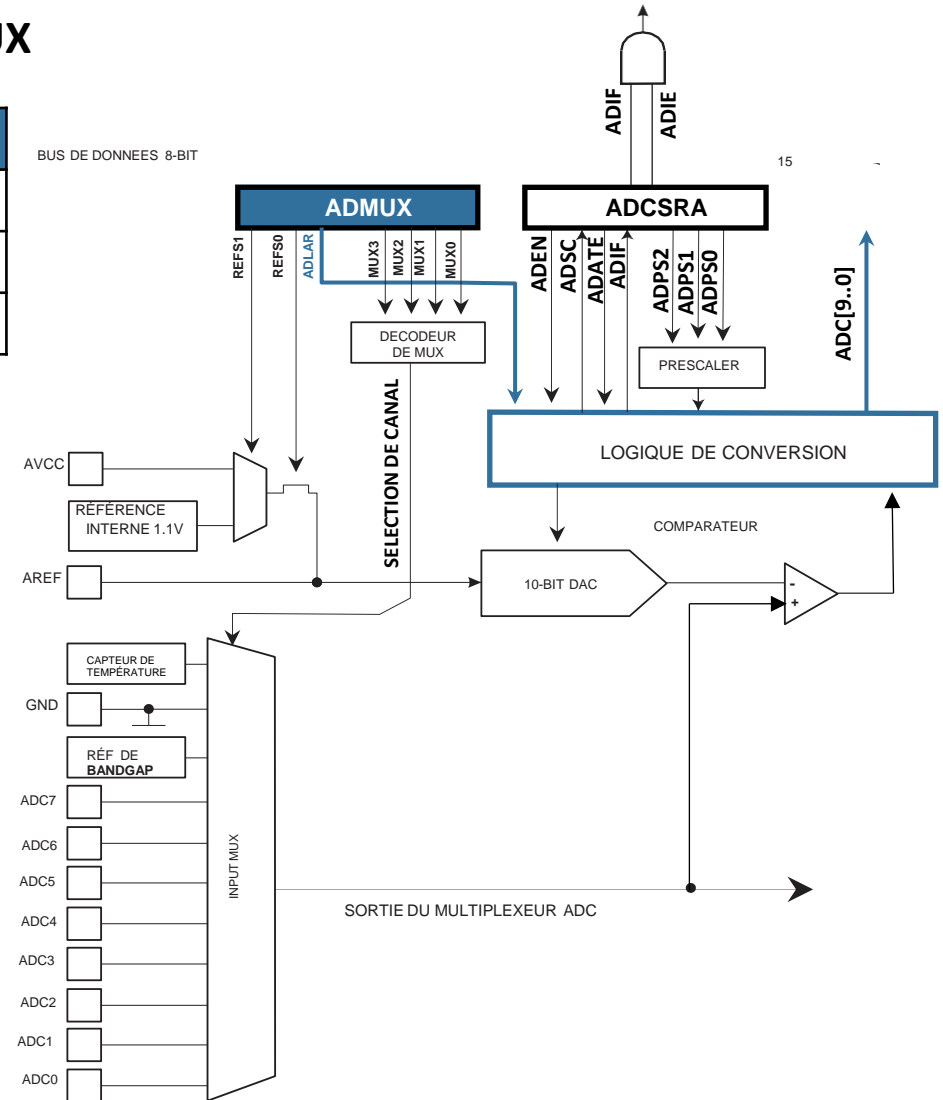
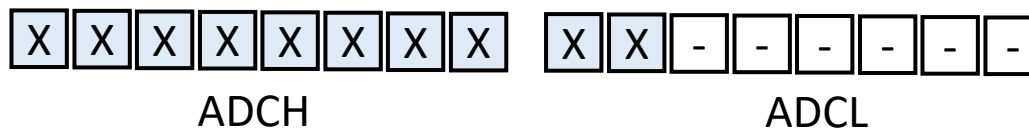
■ Bit 5 : ADLAR (ADC Left Adjust Result)

- Il permet de choisir si l'on veut une distribution des bits de résultat en 2-8 ou 8-2 :

- ADLAR = 0



- ADLAR = 1



- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

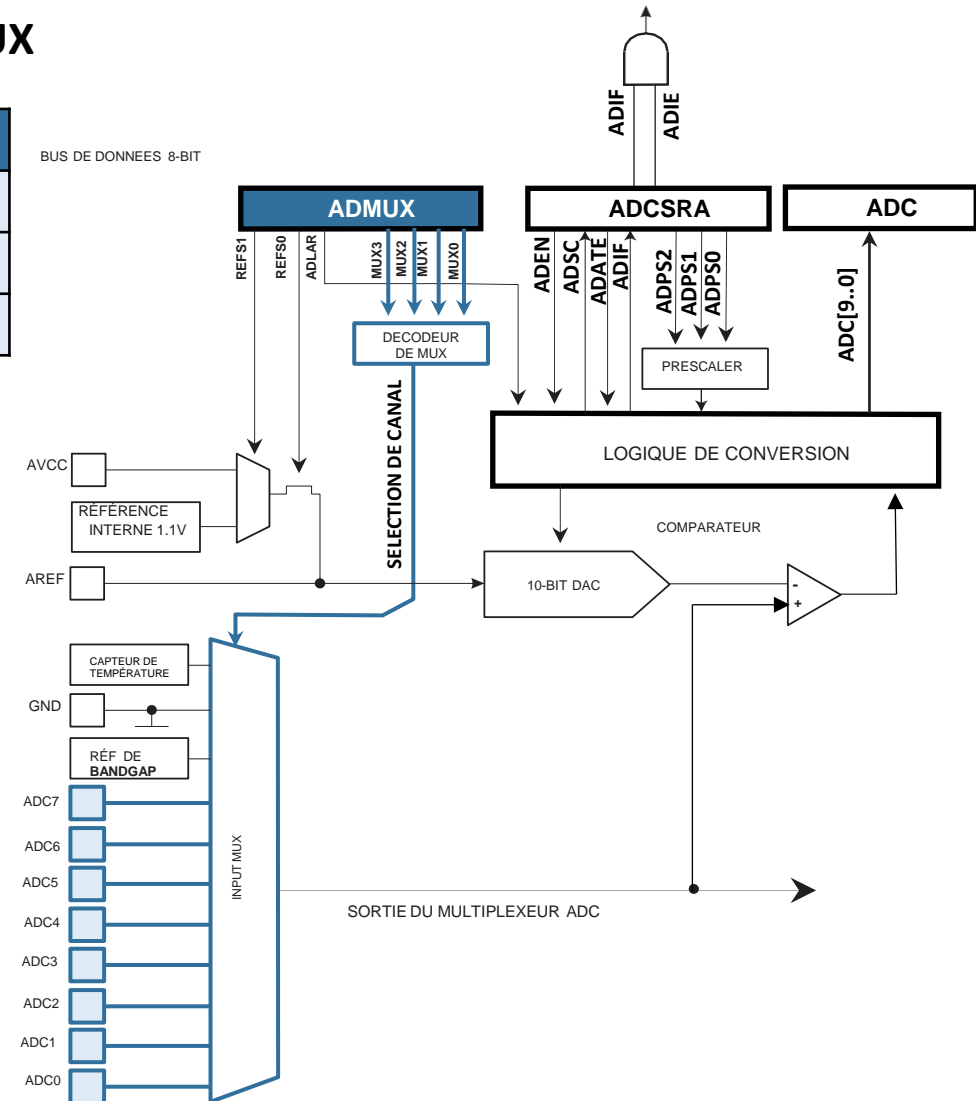
Registre de multiplexage des entrées analogiques : ADMUX

Bit	7	6	5	4	3	2	1	0
0x7C	REFS1	REFS0	ADLAR	-	MUX3	MUX2	MUX1	MUX0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

EXAMPLE

Sélectionner l'entrée ADC0

```
void adcSelectCh(uint8_t ch)
{
    ADMUX &= 0xF0;
    ADMUX |= (ch & 0x0F);
}
```



Choix de l'entrée à convertir (4/4)

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

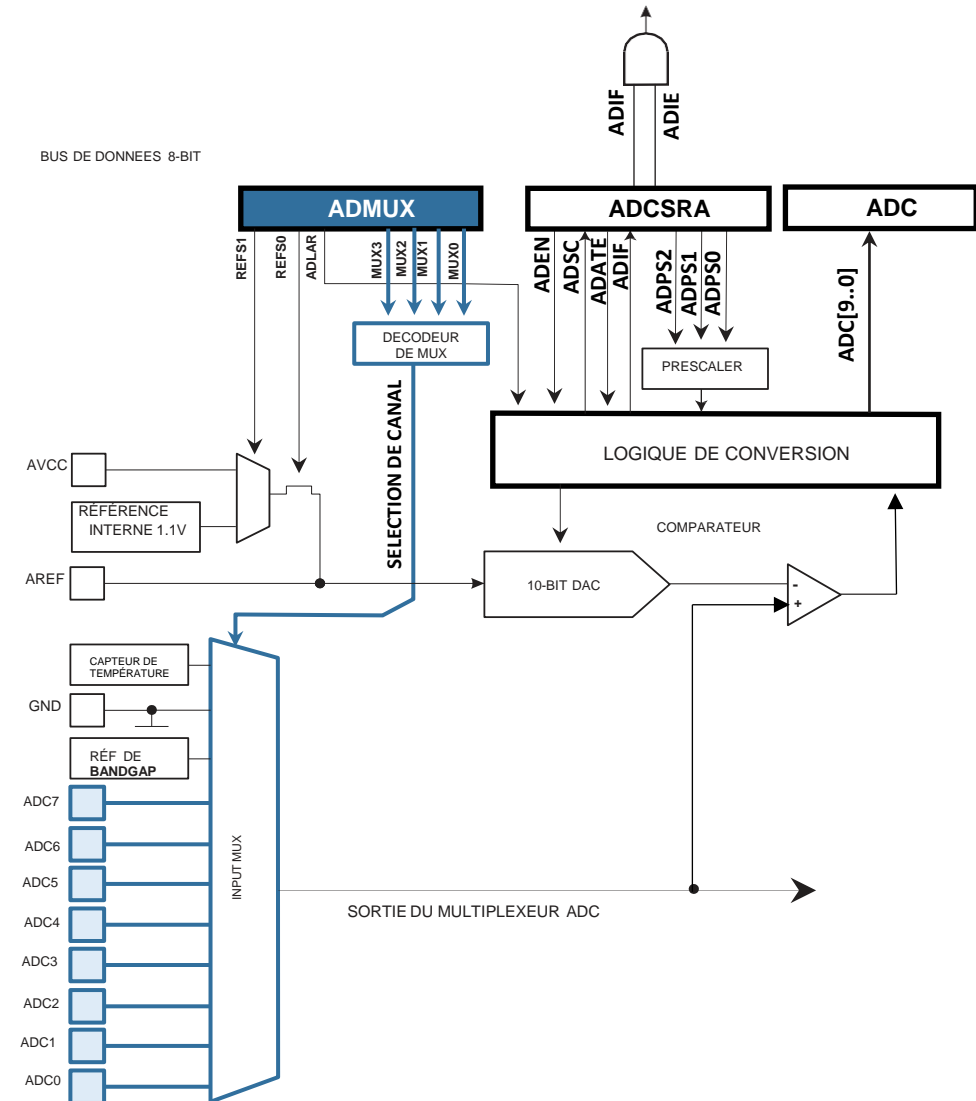
II. CAN sur Arduino

- A. `analogRead()`
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

MUX[3:0]	Entrée Analogique sélectionnée
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	Capteur de Température
1001	Réservé
1010	Réservé
1011	Réservé
1100	Réservé
1101	Réservé
1110	VBG
1111	GND



analogRead()

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

- A. `analogRead()`
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

```
int analogReadNew(uint8_t pin)
{
    // Définition de la référence de tension
    ADMUX |= (1 << REFS0);

    // On sélectionne notre pin
    ADMUX |= pin & 0x07;

    // On lance la conversion
    sbi(ADCSRA, ADSC);

    // Le bit sera désactivé à la fin de la conversion
    while(bit_is_set(ADCSRA, ADSC));

    // Lecture d'ADC result LOW
    uint8_t low = ADCL;

    // renvoie le résultat
    return (ADCH << 8) | low;
}
```

AVCC

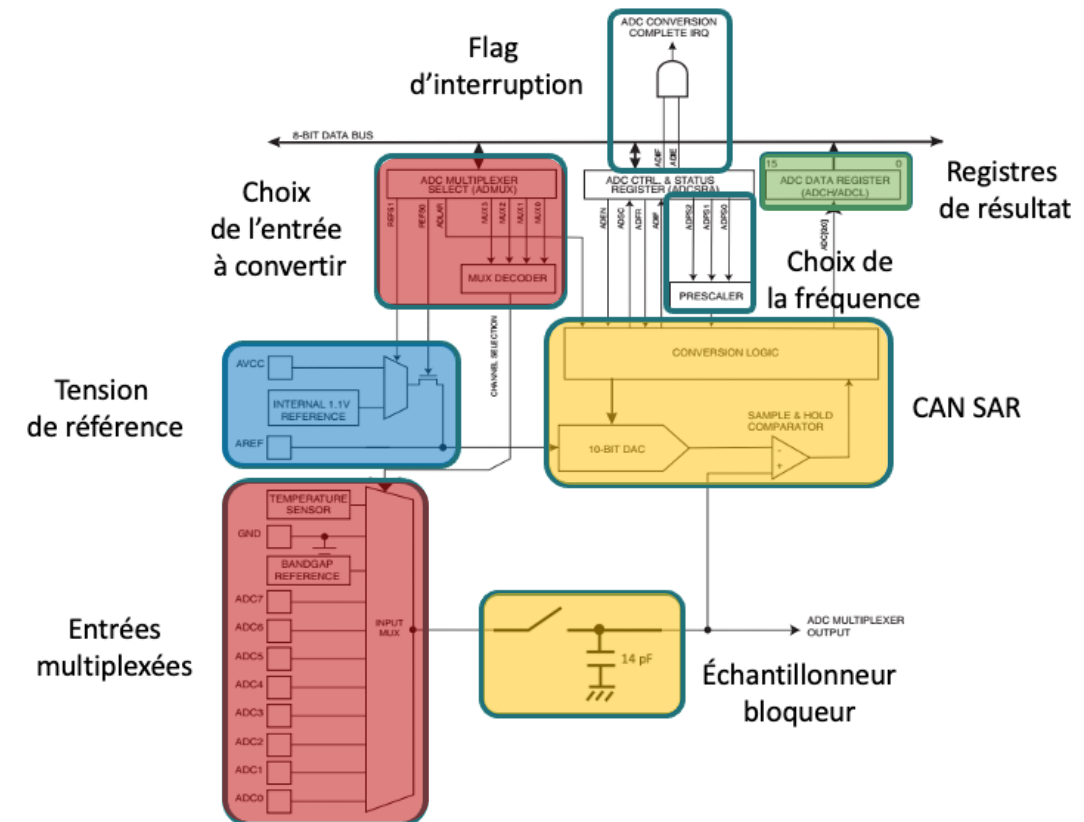
Décalage à gauche

Vitesse d'exécution



$$t_{conv} \approx 115 \mu s$$

$$f_{e,max} \approx 9 kHz$$



Abaissier la tension de référence

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

■ on peut améliorer la précision des mesures en abaissant la tension de référence du CAN.

- Par défaut $V_{REF} = 5V : q = \frac{5}{2^{10}} = 4,88 \text{ mV}$

■ La référence de tension se choisit à l'aide de la fonction `analogReference()`

- Mode **EXTERNAL**
 - Brancher la pin AREF à la tension voulue
 - $q = \frac{3,3}{2^{10}} = 3,23 \text{ mV}$
- Mode **INTERNAL**
 - Utilise une tension stabilisée interne de 1,1 V.
 - $q = \frac{1,1}{2^{10}} = 1,01 \text{ mV}$

REFS1	REFS0	Tension de référence
0	0	Tension sur la pin AREF
0	1	AV_{CC} (par défaut)
1	0	/
1	1	Tension interne



Le changement de référence est effectif après quelques millisecondes.



Ne rien connecter à AREF sans être en mode **EXTERNAL** sinon, cela court-circuite le microcontrôleur.

```
int adcvalue ;
float temperatureCelsius;

void setup()
{
    pinMode(A0, INPUT);
    analogReference(INTERNAL);
    delay(100);
}

void loop()
{
    adcvalue = analogRead(A0); // retourne un int de 0 à 1023
    temperatureCelsius = adcvalue * 1.1 / 1023.0 * 100.0 ;
    // conversion en volt puis conversion (10 mV = 1°C)
}
```

Augmenter la résolution du CAN

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

■ une autre approche (valide que sur certaines cartes) est d'augmenter la résolution du convertisseur.

- Par défaut CAN 10 bits : $q = \frac{5}{2^{10}} = 4,88 \text{ mV}$

■ La résolution se choisit à l'aide de la fonction `analogResolution()`

- Arduino Zero, Due, Nano 33 (BLE & IOT) : jusqu'à **12 bits**

- $q = \frac{5}{2^{12}} = 1,22 \text{ mV}$

- Arduino Portenta H7 : jusqu'à **16 bits**

- $q = \frac{5}{2^{16}} = 76,3 \mu\text{V}$



La conversion se fait donc différemment puisque la fonction `analogRead` retourne alors un int de 0 à $2^{12} = 4096$ ou $2^{16} = 65536$

```
int adcvalue ;
float temperatureCelsius;

void setup()
{
    pinMode(A0, INPUT);
    analogReadResolution(12);
}

void loop()
{
    adcvalue = analogRead(A0); // retourne un int de 0 à 4096
    temperatureCelsius = adcvalue * 5.0 / 4096.0 * 100.0 ;
    // conversion en volt puis conversion (10 mV = 1°C)
}
```

Écriture directe dans les registres (2 / 2)

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

```
#include <avr/io.h>

bool converted;

void setup()
{
    ADCSRA = 0b10000111; // Activation du CAN, prescaler = 128 ( $f_e = 9\text{ kHz}$ )
    ADMUX = 0b01000000; // Tension de référence 5V et lecture sur A0
    converted = true;
}

void loop()
{
    if (converted)
    {
        bitSet(ADCSRA, ADSC); // Démarrage de la conversion
        converted = false;
    }
    if (bit_is_clear(ADCSRA, ADSC)) // Si ADSC = 0 (conversion terminée)
    {
        int value = ADC; // Lecture du résultat
        converted = true;
    }
}
```



Vitesse d'exécution



$t_{CONV} \approx 115\ \mu s$

$f_{e,max} \approx 9\text{ kHz}$

■ idée : l'écriture / lecture dans les registres est plus rapide que l'appel de fonctions, qui sont parfois mal optimisées en langage bas niveau.

Vers une conversion temps réel

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

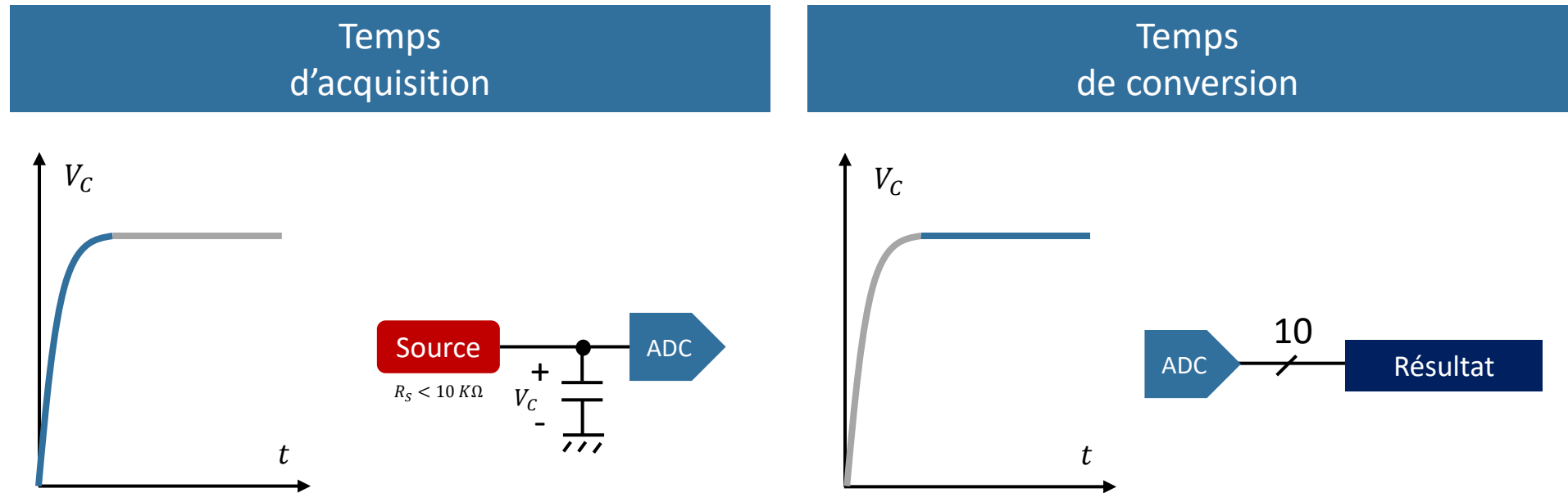
II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

- La numérisation d'un échantillon comprend deux étapes :



- Bien que le temps de conversion puisse être réduit en augmentant la fréquence d'horloge de l'ADC, le caractère séquentiel du microcontrôleur prend du temps et empêche l'exécution d'autres tâches, comme par exemple son traitement :



→ Comment optimiser gestion du temps afin de pouvoir traiter en « temps réel » des échantillons ?

Recours aux interruptions sur CAN (1 / 2)

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

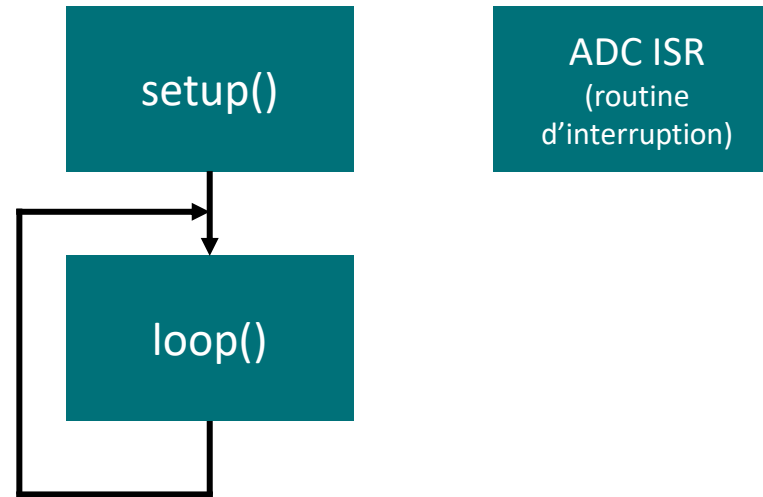


Vitesse d'exécution



$$t_{CONV} \approx 115 \mu s$$

$$f_{e,max} \approx 9 kHz$$



■ chronogramme :

setup()	loop()	loop()	lo	ADC ISR	op()
---------	--------	--------	----	---------	------

■ une fois la conversion terminée, on interrompt le programme pour récupérer le résultat.

■ avantage : ne pas bloquer le processeur le temps de la conversion qui est longue ($115 \mu s > \text{quelques } \mu s$)

Recours aux interruptions sur CAN (2 / 2)

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

```
#include <avr/io.h>

volatile bool converted;
volatile bool newValue;
volatile int value;

void setup()
{
    ADCSRA = 0b10001111; // Activation du CAN, prescaler = 128 ( $f_e = 9\text{ kHz}$ )
    ADMUX = 0b01000000; // Tension de référence 5V et lecture sur A0
    sei(); // Activation des interruptions
    converted = true;
    newValue = false;
}

void loop()
{
    if (newValue) newValue = false;

    if (converted)
    {
        ADCSRA |= bit(ADSC) | bit(ADIE); // Démarrage de la conversion
        converted = false;
    }
}

ISR(ADC_vect)
{
    value = ADC;
    converted = true;
    newValue = true;
}
```



Vitesse d'exécution



$t_{CONV} \approx 115\ \mu s$

$f_{e,max} \approx 9\text{ kHz}$

Activation du CAN (1/2)

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

Registre de Contrôle d'état du CAN : ADCSRA

Bit	7	6	5	4	3	2	1	0
0x7A	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

■ Bit 7 : ADEN (ADC Enable)

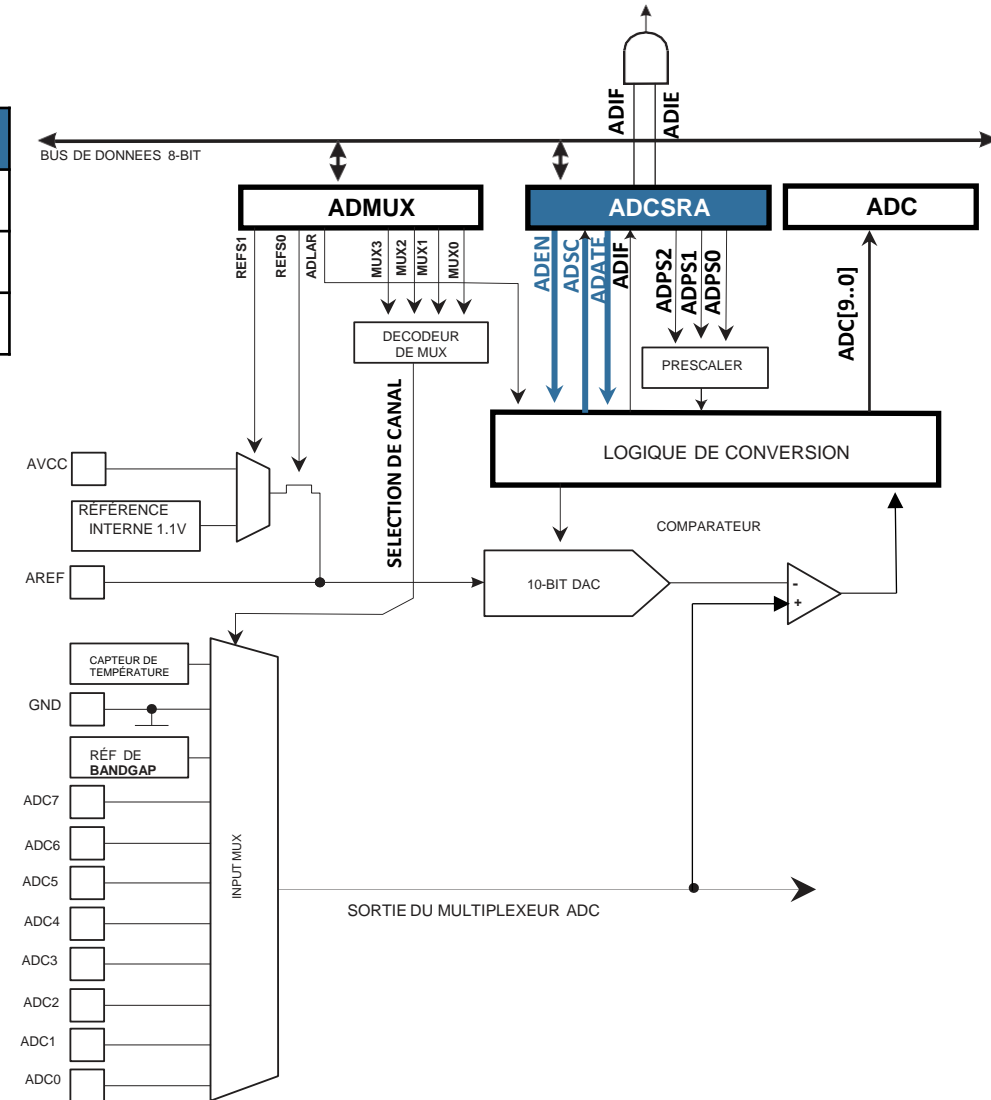
- ▶ Active /désactive le CAN

■ Bit 6 : ADSC (ADC Start Conversion)

- ▶ En mode simple conversion il faut remettre à 1 à chaque nouvelle conversion.
- ▶ En mode libre, la première conversion dure 25 cycles puis les suivantes 15, il n'est pas nécessaire de remettre le bit à 1 à chaque conversion.

■ Bit 5 : ADATE (ADC Auto Trigger Enable)

- ▶ L'activation de ce bit permet de mettre CAN en fonction d'un déclencheur.
- ▶ La sélection du déclencheur est faite avec le bit **ADTS** du registre **SFIOR**.



Activation du CAN (2/2)

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

Registre de Contrôle d'état du CAN : ADCSRA

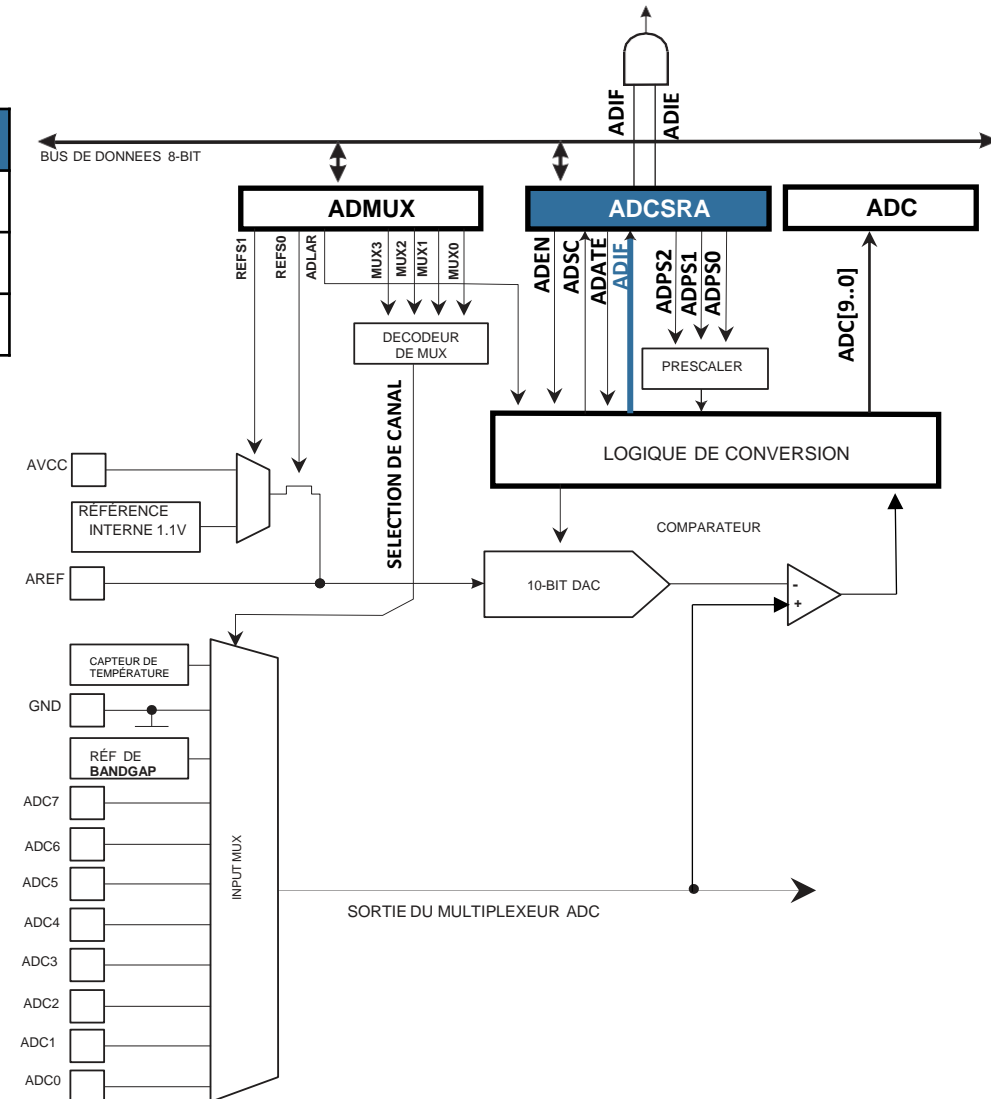
Bit	7	6	5	4	3	2	1	0
0x7A	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

■ Bit 4 : ADIF (ADC Interrupt Flag)

- Passe à 1 une fois la conversion terminée et déclenche l'interruption si **ADIE** = '1', repasse à 0 lors du traitement de la routine d'interruption.

■ Bit 3 : ADIE (ADC Interrupt Enable)

- Validation de l'interruption CAN, déclenché lors du passage à 1 de **ADIF**.



Horloge du CAN

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

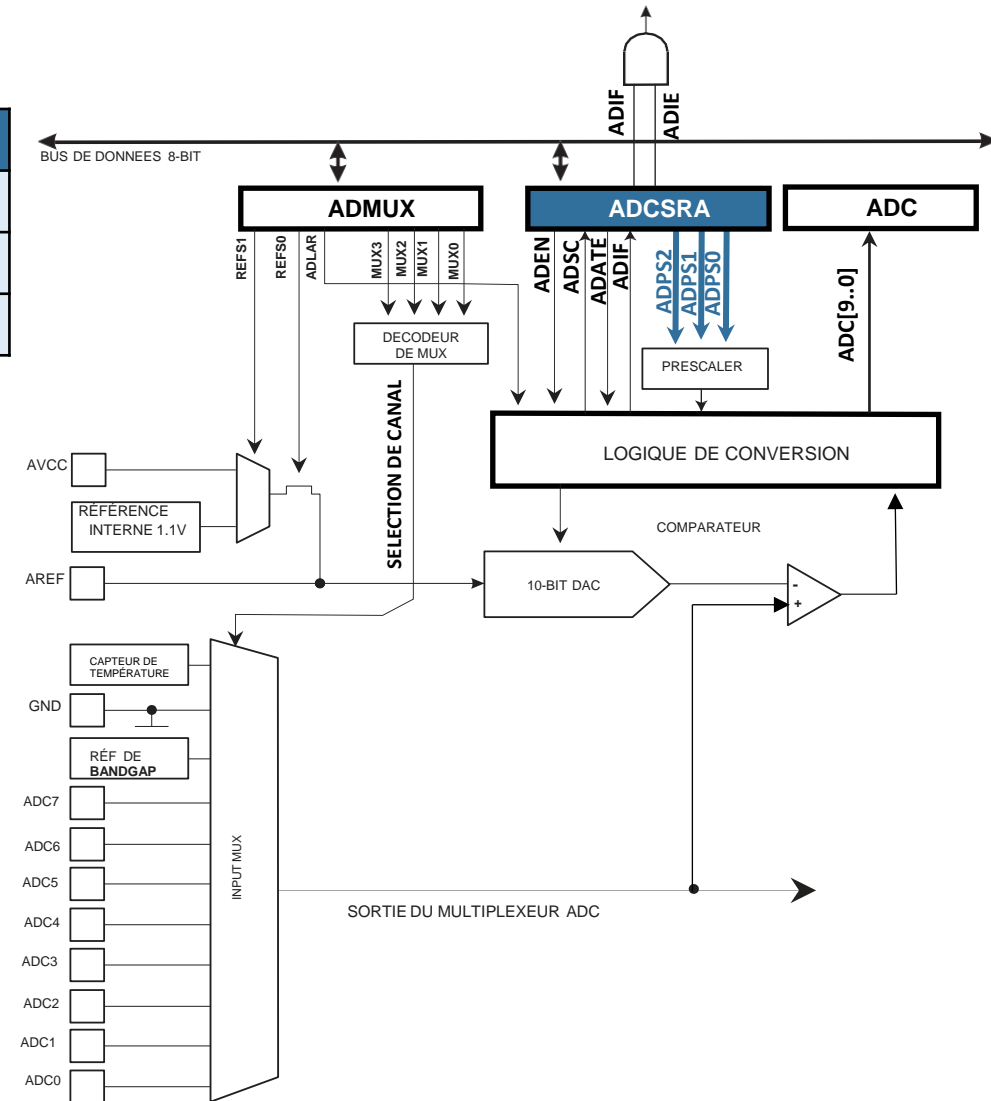
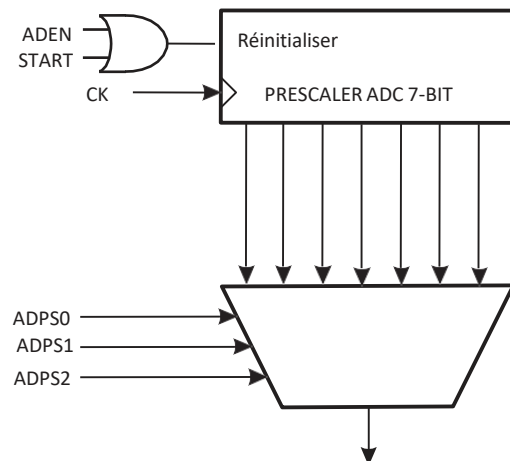
Registre de Contrôle d'état du CAN : ADCSRA

Bit	7	6	5	4	3	2	1	0
0x7A	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

■ Bits 2:0 : ADSP[2..0] (*Prescaler Select Bits*)

- Sélection du facteur de pré-division de l'horloge interne du convertisseur en fonction du quartz:

ADSP2	ADSP1	ADSP0	Facteur de division
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

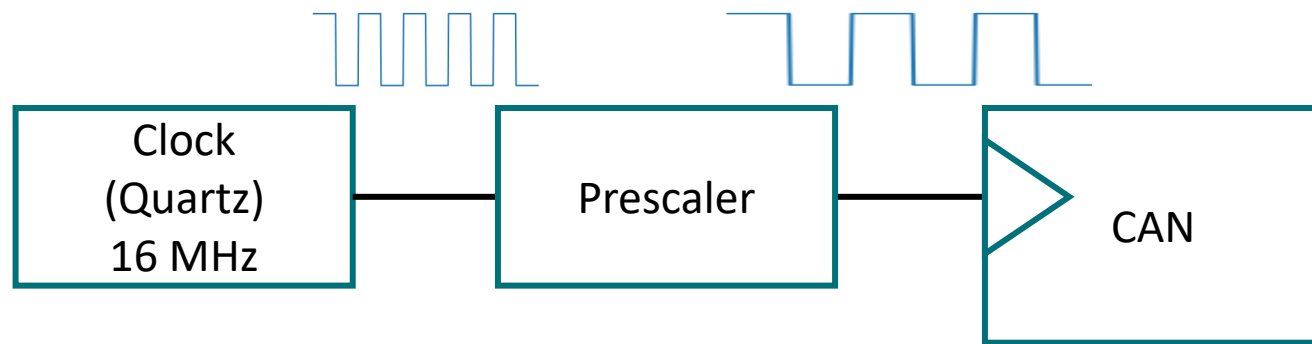


Fast sampling (1 / 2)

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

■ idée : moins ralentir la clock (16 MHz) en utilisant un prescaler (diviseur d'horloge) plus petit



Vitesse d'exécution



$$t_{CONV} \approx 8 \mu s$$

$$f_{e,max} \approx 125 kHz$$

$$t_{conv} = 14 \text{ cycles} * 128 / 16 MHz$$


$$t_{conv} = 112 \mu s$$

II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

ADPS2	ADPS1	ADPS0	division	t_{CONV}	f_e	
1	1	1	/128	115 μs	9 kHz	Précision maximale
1	1	0	/64	60 μs	17 kHz	
1	0	1	/32	35 μs	30 kHz	
1	0	0	/16	20 μs	56 kHz	 Perte de précision
0	1	1	/8	15 μs	72 kHz	
0	1	0	/4	10 μs	100 kHz	
0	0	1	/2	8 μs	125 kHz	

```
// set (→ 1) bit ADPS2 du registre ADCSRA
sbi(ADCSRA, ADPS2);

// clear (→ 0) bit ADPS1 du registre ADCSRA
cbi(ADCSRA, ADPS1);

// clear (→ 0) bit ADPS0 du registre ADCSRA
cbi(ADCSRA, ADPS0);
```

si on diminue le temps on perd en précision autant travailler sur 8bits

Fast sampling (2 / 2)

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

II. CAN sur Arduino

- A. analogRead()
- B. Améliorer la précision
- C. Améliorer la rapidité

III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

```
#include <avr/io.h>

volatile bool converted;
volatile bool newValue;
volatile int value;

void setup()
{
    // Désactivons l'ADC pour l'instant
    cbi(ADCSRA, ADEN);

    // Activation du free-running mode
    ADCSRB = 0x00;

    // On sélectionne notre pin (A0)
    ADMUX |= 0 & 0x07;

    // Important : préciser la référence de tension (ici,
    // équivalent de DEFAULT)
    ADMUX |= (1 << REFS0);

    // Choix de la division du prescaler (ici, facteur 8)
    cbi(ADCSRA, ADPS2);
    sbi(ADCSRA, ADPS1);
    sbi(ADCSRA, ADPS0);

    // Ce bit doit être passé à 1 pour prendre en compte le
    // free-running
    sbi(ADCSRA, ADSC);

    // Demande d'une interruption à la fin de la conversion
    sbi(ADCSRA, ADIE);
}
```



Vitesse d'exécution



$t_{conv} \approx 8 \mu s$

$f_{e,max} \approx 125 kHz$

```
// Réactivons l'ADC
sbi(ADCSRA, ADEN);

// On lance la première conversion
sbi(ADCSRA, ADSC);

// activation des interruptions matérielles
sei();
}}

ISR(ADC_vect) {
    int value = (ADCH << 8) | ADCL;
}

void loop()
{
}
```

Résultat de la conversion

I. Le CAN

- A. Conversion
- B. Résolution
- C. Temps de conversion
- D. Défauts
- E. Rapport signal sur bruit
- F. Structures

■ Le résultat de la conversion est sur 10 bits mais ne peut pas être contenu dans un unique registre (microcontrôleur 8-bit), mais dans deux registres, soit dans un emplacement mémoire de 16 bits.

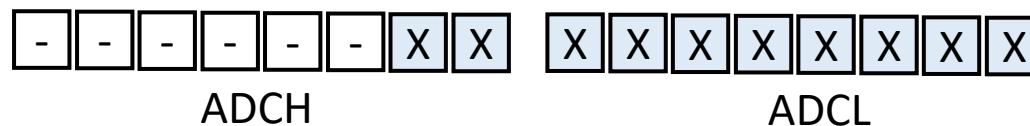
II. CAN sur Arduino

- A. `analogRead()`
- B. Améliorer la précision
- C. Améliorer la rapidité

Pour la combinaison avec $ADLAR = 0$

► Le registre **ADCL (ADC Low)** contient par défaut les 8 bits de droite (bits 0 à 7) ;

► Le registre **ADCH (ADC High)** contient par défaut les 2 bits de gauche (bits 8 et 9).



III. Étude de cas

- A. Numérisation
- B. Restitution
- C. Bloqueur

