TP convertisseur analogique-numérique

É. Carry, J.-M Friedt

11 mars 2021

Questions sur ce TP :

1. Quels sont les trois modes de déclenchement d'un convertisseur analogique-numérique ?
2. L'ADC est un composant numérique qui doit donc être cadencé : quelle est la fréquence maximale de l'horloge alimentant un ADC d'Atmega32U4 ?
3. La référence de tension est un point clé d'une conversion analogique-numérique : quel est son impact sur l'exactitude de la mesure ?
4. ... quel est son impact sur la précision de la mesure ?
5. Quelles sources de références de tension sont disponibles sur un Atmega32U4 ?
6. Sur combien de bits sont codés les mots issus de l'ADC de l'Atmega32U4 ?
7. Peut-on acquérir simultanément les tensions sur deux canaux d'ADC (e.g ADC0 et ADC1) d'un Atmega32U4 ? Justifier.
8. Comment se nomme le registre qui permet de sélectionner la voie de conversion ? quel est l'effet d'écrire une valeur dans ce registre ? (consulter la *datasheet*).

1 Généralités sur le convertisseur analogique-numérique

Un convertisseur analogique-numérique fait le lien entre le monde des grandeurs continues (analogiques) et des grandeurs discrétisées (numériques). Il convertit une valeur V comprise entre 0 et V_{ref}, la tension de référence, en une valeur numérique b comprise entre 0 et $2^N - 1$ pour un encodage sur N bits, selon

$$b = \frac{V}{V_{ref}} \times (2^N - 1) \tag{1}$$

Dans le cas de l'Atmega32U4, $N = 10$.

Quelle est la résolution du convertisseur si $V_{ref} = 3,3$ V ?

De nombreux microcontrôleurs annoncent une multitude de canaux de conversion analogique numérique, alors qu'en pratique un unique convertisseur voit ses entrées multiplexées : seule *une* conversion peut avoir lieu à un instant donné. Des conversions multiples de divers canaux (entrées) sont nécessairement séquentielles, sauf dans le cas particulier de plusieurs convertisseurs physiques (cas du STM32F1xx qui comporte deux convertisseurs par exemple).

En s'inspirant du schéma de la carte électronique, identifier la broche correspondant au canal 0 du convertisseur analogique-numérique (ADC0). De même, quel canal de convertisseur est associé à la broche sérigraphiée A0 sur la carte Olimex ?.

7.4 Arduino shield pin holes

For your convenience the pads are named individually near each of them. Please take extra care about the numbering but consider that there might be offset.

Pad Name	Signal	Pad Name	Signal
POWER CON1		DIGITAL CON2	
RST	RESET	A0	PF7/ADC7/TDI
3V3	+3.3 V	A1	PF6/ADC6/TDO
5V	+5 V	A2	PF5/ADC5/TMS
GND	GROUND	A3	PF4/ADC4/TCK
GND	GROUND	A4	PF1/ADC1
VIN	V in	A5	PF0/ADC0

FIGURE 1: Relation entre la sérigraphie sur le circuit imprimé (compatible avec la bibliothèque Arduino) et les broches du microcontrôleur tel que annoncé à `https://www.olimex.com/Products/Duino/AVR/OLIMEXINO-32U`

Le convertisseur analogique-numérique équipant l'ATMega32U4 fonctionne par approximations successives. Son fonctionnement est détaillé dans le chapitre 24 de [1], tandis que la figure 24-1 de [1] illustre l'architecture générale du convertisseur. On y notera en particulier la présence d'un convertisseur numérique-analogique (DAC) chargé de générer, par rapport à la tension de référence AREF, une tension qui se compare à la tension inconnue à identifier. Une méthode de dichotomie – naturelle pour trouver le mot binaire représentant la tension inconnue – permet de programmer successivement le DAC avec des versions de plus en plus précises du mot qui finira par être l'estimation de la tension (ADCH/ADCL). Noter aussi la présence du multiplexeur routant l'unique sortie vers les 15 entrées possibles, incluant la masse, une référence de tension ou une diode faisant office de capteur grossier de température.

⚠ La référence interne de température des microcontrôleurs est tout juste bonne à détecter une variation relative de température ou une sortie de la gamme de fonctionnement. Elle ne peut en aucun cas être utilisée à des fins métrologiques [1, section 24.6.1].

Citer un autre type de convertisseur analogique-numérique. Quel est l'avantage de chaque méthode ? [1]

2 Affichage

Tout au long de ces exercices nous aurons besoin de formater un affichage d'une valeur codée sur 16 bits afin de l'afficher sur un terminal. Le nombre binaire `v` doit être convertir en chaîne de caractères `s` qui sera affichée par la fonction `fputs(char*, FILE*)`.
Proposer une fonction `affiche(unsigned short, char*)` **qui, de la valeur fournie comme entier codé sur 16 bits, génère la chaîne de caractères affichable qui représente cette valeur en hexadécimal.**

Une sonde de température LM335 [2] est connectée, sur le circuit supportant les afficheurs 7-segments, à la broche A0 de la carte Olimex. Nous désirons mesurer sa température. La tension délivrée par ce composant est de 10 mV/°C.

3 Mode *polling*

Le mode le plus simple du convertisseur est de lancer une conversion, attendre que la conversion s'achève, et lire le résultat dans le registre approprié. Dans cet exemple, nous configurons la tension de référence comme étant la tension d'alimentation. Il s'agit d'un choix confortable mais potentiellement peu judicieux en terme de stabilité de la mesure. En effet, une tension d'alimentation mal régulée induira une variation de b dans Eq. 1 si V_{ref} varie et ce, même en l'absence de fluctuation de V. L'alternative est d'utiliser une référence interne de 2,56 V – de stabilité indépendante de la qualité de l'alimentation mais de dynamique plus faible, ou une tension de référence externe fournie sur la broche AREF (42). Comme dans la majorité des microcontrôleurs actuels, une des voies du multiplexeur d'ADC (MUX=0x27 [1, p.293]) permet de lire une indication de température, ici en sélectionnant aussi la tension de référence interne de 2,56 V.

Listing 1 – Conversion sur ADC0 par *polling*

```
1  // potentiometre entre GND/3.3V pour les extremes et ADC0=A5
2  //   ou sonde de temperature de la carte 7-segments (LM235)
3  //http://maxembedded.com/2011/06/20/the-adc-of-the-avr/
4
5  #include <avr/io.h> //E/S ex PORTB
6  #define F_CPU 16000000UL
7  #include <util/delay.h>  // _delay_ms
8  #include "affiche.h"
9  #include "VirtualSerial.h"
10
11 extern USB_ClassInfo_CDC_Device_t VirtualSerial_CDC_Interface;
12 extern FILE USBSerialStream;
13
14 void adc_init()
15 {   DIDR0=0xff; // http://www.openmusiclabs.com/learning/digital/atmega-adc
16     ADMUX = (1<<REFS0); // AREF = AVcc, 2.5 V si (1<<REFS0|1<<REFS1)
17     // ADC Enable and prescaler of 128 : 16 MHz/128=125 kHz
18     ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
19 }
20
21 unsigned short adc_read(uint8_t ch)
22 {
23   ch &= 0x07;              // ch\in[0..7]
24   ADMUX=(ADMUX & 0xF8)|ch; // clears the bottom 3 bits before ORing
25
26   ADCSRA |= (1<<ADSC);       // start single convertion
27   while(ADCSRA & (1<<ADSC)); // poll jusqu'a fin de conversion
28   return (ADC);
29 }
30
31 int main(void){
32   unsigned short res=0;
33   char s[9];
34   SetupHardware();
35   CDC_Device_CreateStream(&VirtualSerial_CDC_Interface, &USBSerialStream);
36   GlobalInterruptEnable();
```

1. http://www.analog.com/library/analogdialogue/archives/39-06/architecture.html
2. www.ti.com/lit/ds/symlink/lm335.pdf

```
37
38    DDRB |=1<<PORTB5;
39    DDRE |=1<<PORTE6;
40    PORTB |= 1<<PORTB5;
41    PORTE &= ~1<<PORTE6;
42
43    adc_init();
44    s[0]='0';s[1]='x';s[6]='\r';s[7]='\n';s[8]=0;
45
46    while (1){
47      PORTB^=1<<PORTB5;PORTE^=1<<PORTE6;
48      _delay_ms(500);
49      res=adc_read(0);
50      affiche(res,&s[2]);
51      fputs(s, &USBSerialStream);
52  //  les 3 lignes ci-dessous pour accepter les signaux venant du PC
53      CDC_Device_ReceiveByte(&VirtualSerial_CDC_Interface);
54      CDC_Device_USBTask(&VirtualSerial_CDC_Interface);
55      USB_USBTask();
56    }
57    return 0;
58  }
```

1. **Proposer le `Makefile` qui exploite la fonction `affiche()` pour afficher le résultat de la conversion.**
2. Le choix de la référence de tension est un point crucial de la résolution d'un ADC. En effet, puisque $valeur \propto V/V_{ref}$, une source de référence V_{ref} instable se traduit par une $valeur$ fluctuante même si V reste stable. Une source est une référence interne, stable mais de valeur réduite, particulièrement appropriée pour la mesure de petits signaux. Une alternative pour accéder à une gamme de mesure plus large – nécessaire dans les exemples qui vont suivre – est d'utiliser la tension d'alimentation comme référence.
   Quelle est la référence de tension utilisée pour la mesure? ce choix est-il judicieux?

24.9 ADC Register Description

24.9.1 ADC Multiplexer Selection Register – ADMUX

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
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 7:6 – REFS1:0: Reference Selection Bits**

These bits select the voltage reference for the ADC, as shown in Table 24-3. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set). The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin.

Table 24-3. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AV_{CC} with external capacitor on AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor on AREF pin

3. **Quel est le canal d'ADC mesuré? Comment activer la lecture sur le canal 1 qui est connecté au jack qui va à la carte son?**

Une fois le convertisseur configuré, nous avons uniquement à lancer un ordre de début de conversion (bit `ADSC` – *ADC Start Conversion*) en prenant soin d'avoir défini le canal sur lequel se fait la mesure, puis d'attendre le message de fin de conversion.

Après avoir vérifié que la carte est alimentée en 5 V (cavalier à côté du port USB), placer un fil alternativement entre A0 et GND, A0 et 3,3V, A0 et 5 V. Quelles sont les valeurs renvoyées par l'ADC?

Afin d'afficher des séquences de mesures stockées en format hexadécimal (que Matlab et GNU/Octave ne savent pas lire par `load`, nous pourrons par exemple utiliser `f=fopen("fichier.dat"); d=fscanf(f,"%x");` (Fig. 2).

⚠Pour utiliser un ADC au dessus de 7, penser à activer la fonction analogique correspondante (registre DIDR2) et noter que MUX5 se trouve dans un autre registre que les 5 premiers bits de configuration du canal.

Configurer le convertisseur pour exploiter le canal de mesure de température interne (attention, MUX5 ne se trouve *pas* dans ADMUX), et démontrer une mesure de température. Quelle est la précision de la mesure, sachant qu'en sélectionnant comme tension de référence la source interne à 2,56 V, la valeur fournie par le convertisseur est directement la température en Kelvin Quelle est son exactitude (d'après la datasheet)?

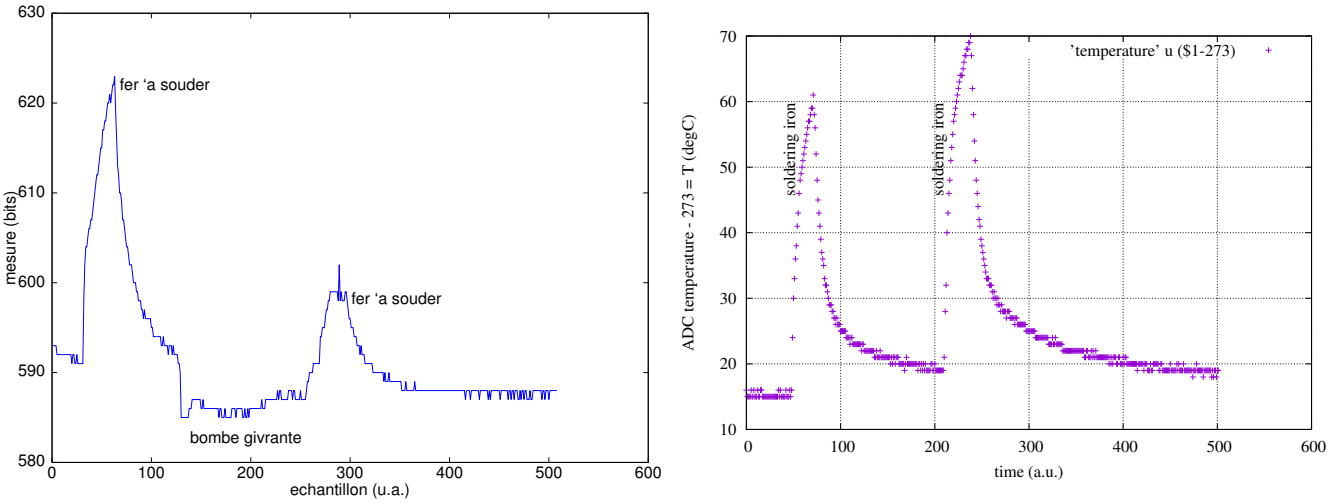


FIGURE 2 – Mesure de température par la sonde LM335 (gauche) et par la sonde interne à l'Atmega32U4 (droite).

4 Acquisition d'un signal « rapide » : utilisation de la carte son

Une carte son de PC est un générateur idéal de signaux périodiques, facilement accessible. La seule subtilité consiste à transposer le signal en tension puisque l'ADC de l'Atmega32U4 n'accepte que des tensions positives. Le circuit de la Fig. 3 propose une solution certes peu élégante, mais fonctionnelle.

Comment faudrait-il améliorer le circuit pour ne pas être dépendant de l'impédance de sortie de la carte son et de l'impédance d'entrée de l'ADC ?.

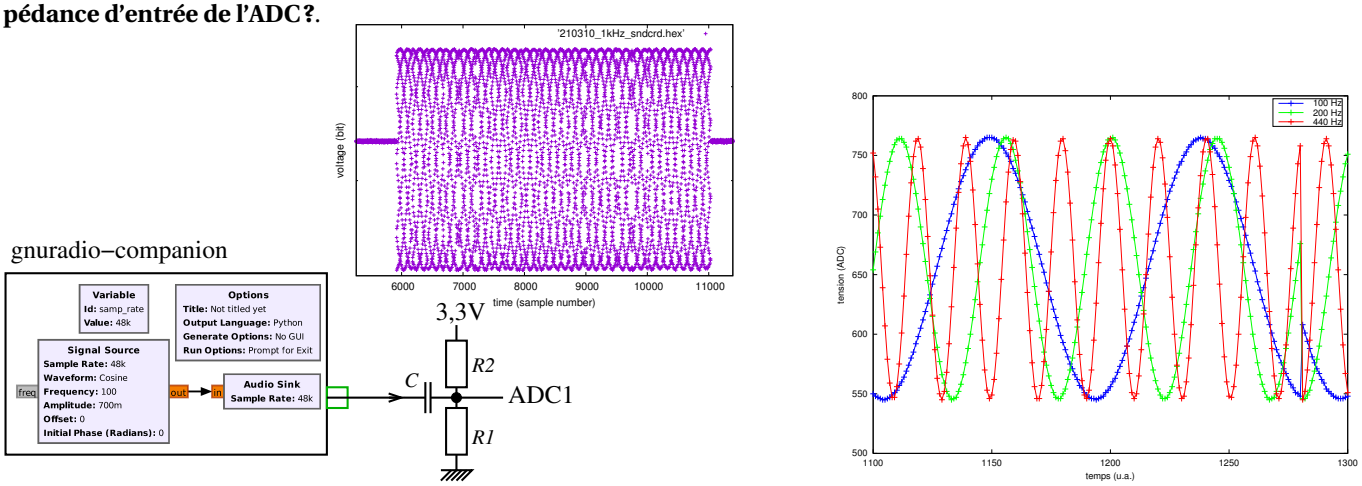


FIGURE 3 – Gauche : utilisation de `gnuradio-companion` pour générer un signal périodique à échantillonner par l'ADC1 de l'Atmega32U4. Droite : résultat de la mesure.

Le danger lors de l'utilisation d'un ADC est de ne pas tenir compte de sa **faible impédance** : [1, p.285] nous informe que l'impédance équivalente du convertisseur analogique-numérique est $Z_{ADC} \simeq 10\ \text{k}\Omega$. Nous devons donc prendre soin de ne pas effondrer la source de courant d'un capteur présentant une impédance plus élevée que Z_{ADC} (donc incapable de fournir suffisamment de courant, puisque $Z = U/I$). Par ailleurs, notre montage de T de polarisation est (volontairement) naïf car n'inclut aucun circuit d'adaptation d'impédance. Le cas idéal serait de considérer $R1$ et $R2$ comme pont diviseur de tension. Dans le cas idéal, le pont diviseur de tension tire la tension de sortie de la carte son, protégée contre le courant associé au potentiel continu par C, de $\frac{R2}{R1+R2}$. Le cas pratique n'est pas aussi simple, d'une part parceque l'ADC présente une impédance finie qui se place en parallèle de $R2$, et d'autre part parceque le condensateur qui bloque le retour de courant continu dans la carte son présente en alternatif une impédance de module $\frac{1}{C\omega}$. Ainsi, dans l'équation du pont diviseur, $R1$ devient $R1 - \frac{j}{C\omega}$ et $R2$ devient $\frac{R2 \cdot Z_{ADC}}{R2+Z_{ADC}}$.

Comment acquérir les données au plus vite afin d'identifier la fréquence d'échantillonnage ?
Quelle est la fréquence d'échantillonnage du signal ?

Un extrait du programme utilisé pour générer les courbes de la Fig. 3 (droite) est proposé ci-dessous. Il se base sur l'exemple en mode *polling* de gestion de l'ADC. On notera que la carte son est connectée à l'entrée ADC1.

```
1 [...]
```

```
int main(void){
  unsigned short res[256];
  int k;
  char s[7];

[...]
  while (1){
    PORTB^=1<<PORTB5;PORTE^=1<<PORTE6;
    _delay_ms(100);
    for (k=0;k<256;k++) res[k]=adc_read(1);
    for (k=0;k<256;k++) {affiche(res[k],s); fputs(s, &USBSerialStream); }
  }
  return 0;
}
```

En cas de dysfonctionnement, vérifier au moyen de `alsamixer` que l'amplitude du signal de sortie n'est pas au minimum et que la sortie audiofréquence est active. On pourra aussi s'affranchir de l'interface graphique de `gnuradio-companion` pour générer le signal audiofréquence sur la carte son en utilisant la commande `play -n synth 3 sin 440` pour générer continument une sinusoïde de fréquence 440 Hz pendant 3 secondes (l'argument de durée peut être omis pour émettre continûment).

Ce mode de fonctionnement – attendre la fin de conversion en sondant l'état d'un bit de statut (*polling*) – gaspille des ressources de calcul en bouclant pour attendre que la conversion prenne fin, alors que d'autres opérations pourraient prendre place entre temps, surtout avec un convertisseur aussi lent que celui annonçant un débit de 15 kS/s (ou 13 cycles d'horloge de l'ADC cadencé entre 50 et 200 kHz [1, p.279] – 200/13=15 kS/s). Nous allons donc nous intéresser au cas où nous lançons un ordre de conversion puis vaquer à nos occupations, le microcontrôleur nous prévenant par interruption de la disponibilité de la donnée.

5 Mode interruption

Plus efficace, au lieu de boucler en sondant le registre de statut du convertisseur analogique-numérique, nous sommes informés par une interruption de la fin de conversion. Ce mode de fonctionnement permet en particulier d'endormir le microcontrôleur en attendant la fin de conversion (économie d'énergie), ou de lancer d'autres tâches pendant que le convertisseur effectue son travail.

Listing 2 – Conversion sur ADC0 détectée par interruption

```
// http://www.avrfreaks.net/forum/tut-c-newbies-guide-avr-adc
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/power.h>
#include <avr/interrupt.h>
#include <avr/wdt.h>
#include <util/delay.h>  // _delay_ms
#include "affiche.h"
#include "VirtualSerial.h"

extern USB_ClassInfo_CDC_Device_t VirtualSerial_CDC_Interface;
extern FILE USBSerialStream;

#define N 256

volatile char flag=0;

ISR(ADC_vect) {flag=1;}

void adc_init()
{ADMUX  = (1<<REFS0); // Set ADC reference to AVCC
 ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0); // no MUX for ADC0
 ADCSRA |= (1 << ADIE); // Enable ADC Interrupt
}

int main (void)
{volatile unsigned short res[N],indice=0;
 char s[9];
 int k;
```

```
30  s[0]='0';s[1]='x';s[6]='\r';s[7]='\n';s[8]=0;
31
32  SetupHardware();
33  CDC_Device_CreateStream(&VirtualSerial_CDC_Interface, &USBSerialStream);
34  GlobalInterruptEnable();
35
36  adc_init();
37  sei();
38  ADMUX|=0x01; // ADC1
39  ADCSRA |= (1<<ADSC);          // start single conversion
40  while (1)
41    {if (flag!=0)
42       {res[indice]=ADC;flag=0; // fin de conversion par interrupt
43        indice++;
44        if (indice==N)
45           {for (k=0;k<N;k++)
46              {affiche(res[k],&s[2]);fputs(s, &USBSerialStream);
47 // est-ce malin de mettre USB dans la boucle plutot que de remplir
48 //   le descripteur de fichier et envoyer en une fois ?
49               CDC_Device_USBTask(&VirtualSerial_CDC_Interface);
50               USB_USBTask(); //  communication avec le PC
51              }
52            indice=0;
53           }
54        ADCSRA |= (1<<ADSC);  // start single conversion
55       }
56 //  _delay_ms(100);
57    }
58 }
```

Nous prenons soin de déclencher les interruptions en général (`sei()`), et de rapatrier la donnée lue dans le registre ADC (10 bits) lorsqu'un drapeau est transféré du gestionnaire d'interruption au programme principal indiquant la disponibilité de la donnée.

Vérifier le bon mode de fonctionnement de cette méthode de mesure

Estimer la fréquence d'échantillonnage en observant le nombre de points acquis sur un signal de référence émis par la carte son du PC. Modifier le programme pour maximiser la bande passante de mesure.

Comme pour la lecture sur une carte son, il est courant de charger de façon automatique un tableau de données et d'informer le programme principal de la disponibilité de cette masse de données plutôt qu'interrompre l'exécution du programme principal pour chaque nouvelle donnée acquise.

Modifier le gestionnaire d'interruption afin de remplir un tableau de $N = 256$ éléments et tester dans le programme principal l'indice du tableau pour n'en afficher le contenu que lorsque le Nième élément a été acquis.

6 Échantillonnage périodique sur *timer*

L'ADC peut être déclenché de façon périodique par un *timer* : ce mode de fonctionnement permet de respecter une condition fondamentale du traitement spectral du signal, à savoir l'acquisition périodique des échantillons. Diverses sources de déclenchement sont possibles, telles que décrites sur l'extrait de datasheet de la Fig. 4.

Listing 3 – Conversion sur ADC0 déclenchée par *timer*

```
1  // http://www.avrfreaks.net/forum/atmega32u4-starting-adc-conversion-timer4
2  #include <avr/io.h> //E/S ex PORTB
3  #include <avr/wdt.h>
4  #include <avr/power.h>
5  #include <avr/interrupt.h>
6  #define F_CPU 16000000UL
7  #include <avr/interrupt.h>
8  #include <util/delay.h>  // _delay_ms
9  #include "affiche.h"
10 #include "VirtualSerial.h"
11
12 extern USB_ClassInfo_CDC_Device_t VirtualSerial_CDC_Interface;
13 extern FILE USBSerialStream;
14
15 EMPTY_INTERRUPT(TIMER4_OVF_vect);
```

24.9.4 ADC Control and Status Register B – ADCSRB

Bit	7	6	5	4	3	2	1	0	
	ADHSM	ACME	MUX5	–	ADTS3	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R/W	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – ADHSM: ADC High Speed Mode**

Writing this bit to one enables the ADC High Speed mode. This mode enables higher conversion rate at the expense of higher power consumption.

- **Bit 5 – MUX5: Analog Channel Additional Selection Bits**

This bit make part of MUX5:0 bits of ADRCSRB and ADMUX register, that select the combination of analog inputs connected to the ADC (including differential amplifier configuration).

- **Bit 3:0 – ADTS3:0: ADC Auto Trigger Source**

If ADATE in ADCSRA is written to one, the value of these bits selects which source will trigger an ADC conversion. If ADATE is cleared, the ADTS3:0 settings will have no effect. A conversion will be triggered by the rising edge of the selected interrupt flag. Note that switching from a trigger source that is cleared to a trigger source that is set, will generate a positive edge on the trigger signal. If ADEN in ADCSRA is set, this will start a conversion. Switching to Free Running mode (ADTS[3:0]=0) will not cause a trigger event, even if the ADC Interrupt Flag is set.

Table 24-6. ADC Auto Trigger Source Selections

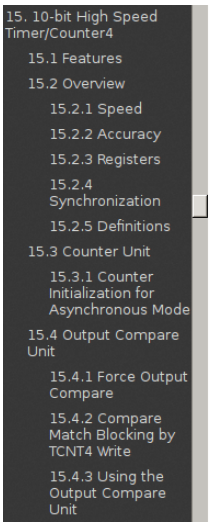
ADTS3	ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	0	Free Running mode
0	0	0	1	Analog Comparator
0	0	1	0	External Interrupt Request 0
0	0	1	1	Timer/Counter0 Compare Match
0	1	0	0	Timer/Counter0 Overflow
0	1	0	1	Timer/Counter1 Compare Match B
0	1	1	0	Timer/Counter1 Overflow
0	1	1	1	Timer/Counter1 Capture Event
1	0	0	0	Timer/Counter4 Overflow

FIGURE 4 – Sources de déclenchement de l'ADC – notamment liées aux divers *timers*.

```
16
17 volatile unsigned short adc,flag;
18
19 ISR(ADC_vect)
20 {adc = ADC;
21  flag = 1;
22 }
23
24 void adc_init()
25 { ADMUX = (1<<REFS0);   // quelle tension de reference ? quelle canal ?
26   ADCSRB = (1<<ADTS3); // quelle source de declenchement de la mesure ?
27   ADCSRA = (1<<ADEN) | (1<<ADSC) | (1<<ADATE) | (1<<ADIE) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);
28 }
29
30 int main(void)
31 { char s[9];
32   s[0]='0';s[1]='x'; s[6]='\r'; s[7]='\n'; s[8]=0;
33
34   SetupHardware();
35   CDC_Device_CreateStream(&VirtualSerial_CDC_Interface, &USBSerialStream);
36   GlobalInterruptEnable();
37
```

```
38   TCCR4B = (1<<CS43) | (1<<CS42) | (1<<CS41) | (1<<CS40); // quel prescaler ?
39   TIMSK4 = (1<<TOIE4); // quel taux de raffraichissement de l'interruption timer ?
40   TC4H = 3; OCR4C = 0xFF; // Set TOP (overflow limit) to 0x3FF (= 1023)
41
42   adc_init();
43   sei();
44
45   while (1)
46    {
47     if (flag)
48       {flag = 0;
49        affiche(adc,&s[2]); fputs(s, &USBSerialStream);
50 //  les 3 lignes ci-dessous pour accepter les signaux venant du PC
51        CDC_Device_ReceiveByte(&VirtualSerial_CDC_Interface);
52        CDC_Device_USBTask(&VirtualSerial_CDC_Interface);
53        USB_USBTask();
54       }
55    }
56 }
```

1. **Calculer et valider la fréquence d'échantillonnage périodique de la conversion. Pour ce faire, répondre aux diverses questions parsemant les commentaires du code.**
2. **Quelle est la source de déclenchement de la mesure (configuration de `ADCSRA` – quel *timer* et selon quelle configuration) ?**
3. **Modifier la fréquence de déclenchement du *timer* et vérifier l'impact sur la mesure.**



The Clock Select bits 3, 2, 1, and 0 define the prescaling source of Timer/Counter4.

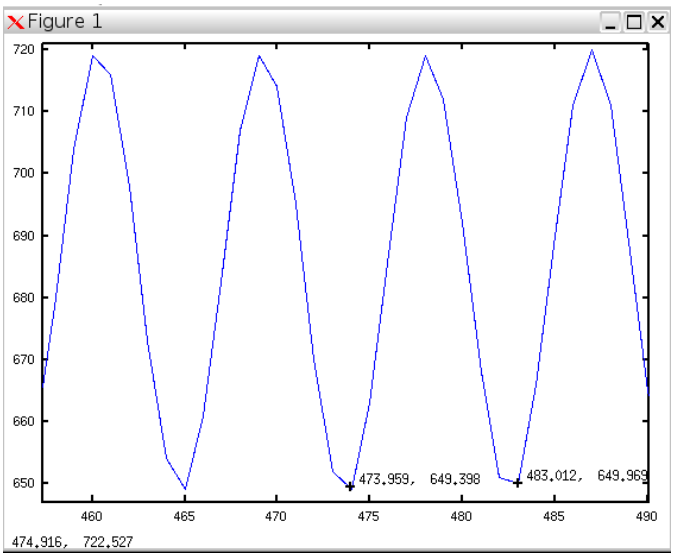
Table 15-15. Timer/Counter4 Prescaler Select

CS43	CS42	CS41	CS40	Asynchronous Clocking Mode	Synchronous Clocking Mode
0	0	0	0	T/C4 stopped	T/C4 stopped
0	0	0	1	PCK	CK
0	0	1	0	PCK/2	CK/2
0	0	1	1	PCK/4	CK/4
0	1	0	0	PCK/8	CK/8
0	1	0	1	PCK/16	CK/16
0	1	1	0	PCK/32	CK/32
0	1	1	1	PCK/64	CK/64
1	0	0	0	PCK/128	CK/128
1	0	0	1	PCK/256	CK/256
1	0	1	0	PCK/512	CK/512
1	0	1	1	PCK/1024	CK/1024
1	1	0	0	PCK/2048	CK/2048
1	1	0	1	PCK/4096	CK/4096
1	1	1	0	PCK/8192	CK/8192
1	1	1	1	PCK/16384	CK/16384

En particulier, configurer l'ADC pour échantillonner à 1 kHz et exciter une sinusoïde périodique à 100 Hz sur la carte son.

4. **Quelle est la fréquence maximale accessible par cette méthode ?**

Nous constatons cependant que la fréquence maximale accessible est 8 kHz. Dans l'exemple ci-dessous, nous avons choisi CS43=0, CS42=0, CS41=1 et CS40=1 pour obtenir une fréquence d'échantillonnage de 4 kHz, qui se traduit bien par 9 points/période lors de l'acquisition d'un signal représentant une sinusoïde à 440 Hz.



Compte tenu de l'instabilité de la bibliothèque de communication par USB, on sauvera les données dans le fichier `toto` au moyen de `minicom -C toto`.

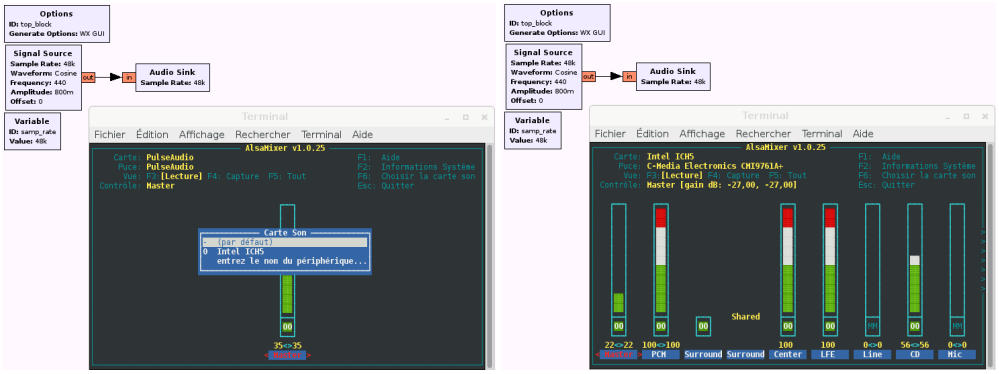
Au lieu de lire une unique valeur dans un scalaire `adc`, remplir dans l'interruption un tableau de valeurs, qui sera lu par le programme principal une fois plein. Cette opération a pour vocation de s'affranchir de la limitation sur la fréquence d'acquisition induite par le temps de communication.

Bonus : passer le microcontrôleur en mode veille entre deux conversions analogique-numériques afin de réduire la consommation électrique du microcontrôleur.
Pour ce faire, on s'inspirera de la documentation disponible à `http://www.nongnu.org/avr-libc/user-manual/group__avr__sleep.html`.

⚠Attention : l'interruption du *timer* 4 permet de sortir du mode veille mais entre en conflit avec la communication par USB. Penser, dans ce mode, à communiquer au travers de l'USART (port RS232).

Appendice : réglage du niveau sonore

L'utilisation de la carte son implique d'activer la sortie audio et de régler le niveau sonore. L'outil pour ce faire est `alsamixer`. Si *pulseaudio* est le pilote audio utilisé, il faut passer en mode `alsa` en sélectionnant la carte son par F6. On vérifiera en particulier que les sorties ne sont pas inhibées (`Mute`).



Références

[1] Atmel, 8-bit Microcontroller with 16/32K Bytes of ISP Flash and USB Controller – ATmega16U4 & ATmega32U4, disponible à `www.atmel.com/Images/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf` (version Atmel-7766J-USB-ATmega16U4/32U4-Datasheet_04/2016)