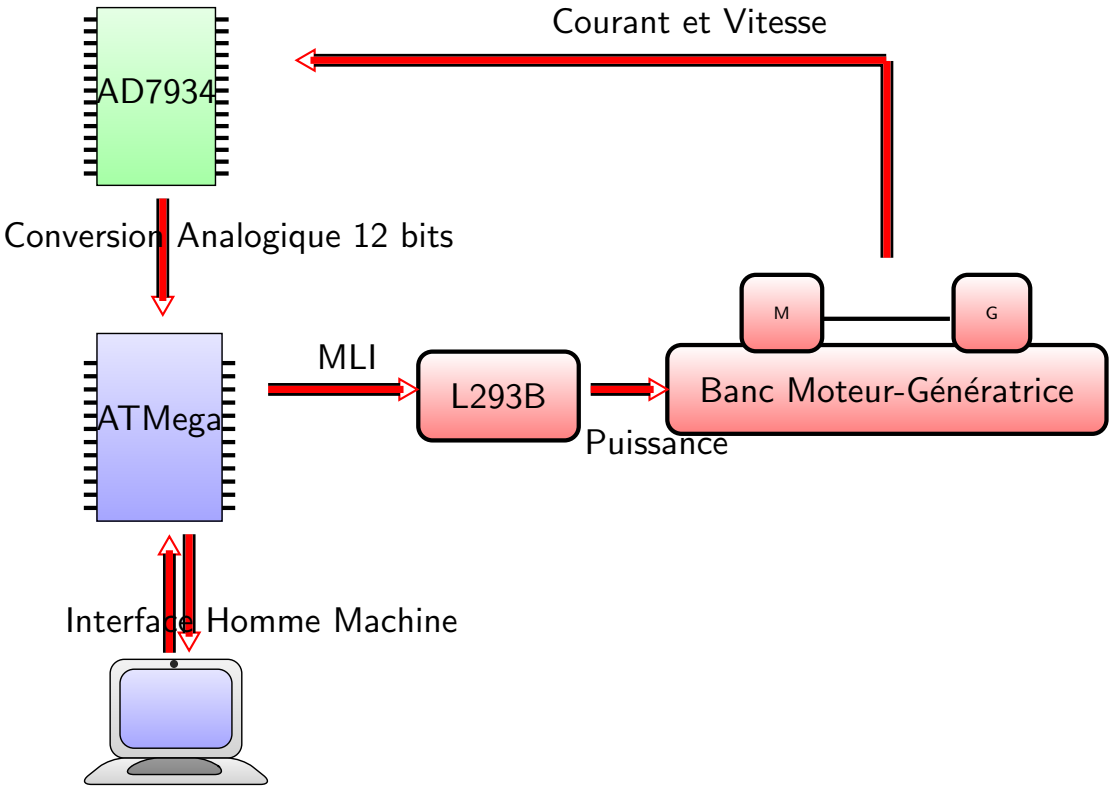
Micro-Projet d'Informatique Industrielle S7

Nadia Aït Ahmed, David Delfieu

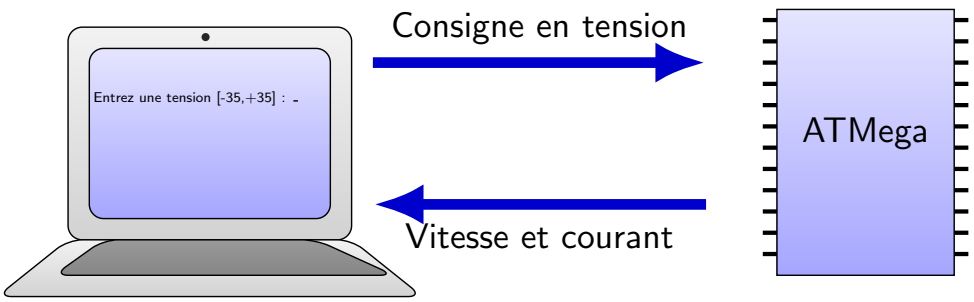
September 30, 2025

Commande en boucle ouverte d'une *MCC*



1. Conversationnel sous Arduino
   - Bibliothèque serial
   - Moniteur Serie
   - Algorithme du conversationnel
   - Traceur serie
2. Génération de la MLI
   - Hacheur et *MCC*
   - Programmation du Timer de l'$ATMEGA_{2560}$
3. Etude et écriture du driver de l'ADC AD7934
   - Le brochage
   - Entrées du CAN
   - L'interface parallèle
   - Programmation de l'*AD7934*

Conversationnel sous Arduino



Protocole série

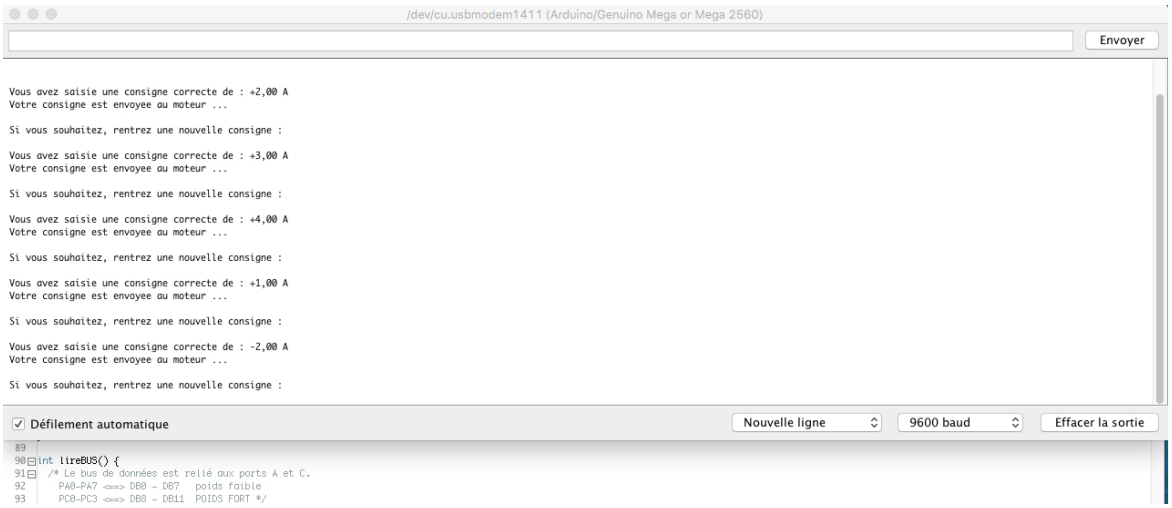
- Protocole RS232 série : Transmission d'octet
- Physique : 3 fils, émission, transmission, masse.
- Arduino : pont USB-RS232
- RS232 : bibliothèque *Serial*
- Pins en TTL (5V ou 3.3V):
  Transmission TxD (PD3)
  Reception RxD (PD2)
- $ATMEGA_{2560}$: 3 ports séries additionnels:

Le Moniteur Serie

- Plusieurs liaisons série
- Outil "Moniteur série".

Le Moniteur Serie

- Une fenêtre avec deux panneaux
- Cette fenêtre affichée sur le PC modélise les entrées sorties de l'Arduino : Clavier/écran
  - Une ligne de saisie
  - Une écran d'affichage



Configuration du moniteur série

- Baud rate : 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, ou 115200.
- Second argument :

  bit de parité,
  bit de stop.
- Réglage 8N1 : 9600 bauds, 8 bits, pas de bit de parité,bit de stop.
- Instruction : Syntaxe Serial.begin(speed)
  Syntaxe Serial.begin(9600, SERIAL_8N1);

Lecture et écriture sur le bus série

- Le bus série se comporte comme un tube/pipeline bi-directionnel FIFO.
- Le bus série permet une communication de type asynchrone : L'émetteur et le récepteur émettent ou retirent de l'information de façon non synchrone.
  - Celui qui écrit, envoie des données dans le bus des données qui se remplit
  - Celui qui lit, consomme des données dans le bus qui se vide alors
- La lecture sur le bus se fait en mode caractère et produit donc des codes ASCII.
- Le process d'écriture : Arduino $\Rightarrow$ Ecran du PC
- Le process de lecture : Clavier du PC $\Rightarrow$ Arduino

Ecriture : Arduino $\Rightarrow$ Ecran du PC : Serial.print()

- Affiche à l'écran du PC, les données envoyées par l'Arduino qui transitent par le bus USB au format ASCII.
- Les nombres sont affichés caractères par caractères en ASCII.

```
Serial.print(78);              // affiche "78"
Serial.print(1.23456);         // affiche "1.23"
Serial.print(byte(78));        // affiche "N" (dont le code ASCII est 78)
Serial.print('N');             // affiche "N"
Serial.println("Hello world."); // affiche Hello world avec saut de ligne
Serial.print("\n Voici la valeur de A = ");
Serial.print(A);
```

Lecture : clavier du PC ⇒ Arduino : Serial.read()

Attention Serial.read() renvoie des codes ASCII en base 10 !

```
S=Serial.read();
A=Serial.read();
B=Serial.read();

Si l'utilisateur tapes +15 alors S= 43, A=49, B=53
 Le code ASCCI de + est $2B{16} soit 43 en base 10
 Le code ASCII de 1 est $31{16} soit 49 en base 10
 Le code ASCII de 5 est $35{16} soit  53 en base 10
```


readCar()

- La fonction ReadCar() prélève un caractère sur le bus :

```
int readCar() {
   while (Serial.available() == 0); //Attente TQ il n'y a rien sur bus
   return(Serial.read());
}
```

- Utilisation de readCar() :

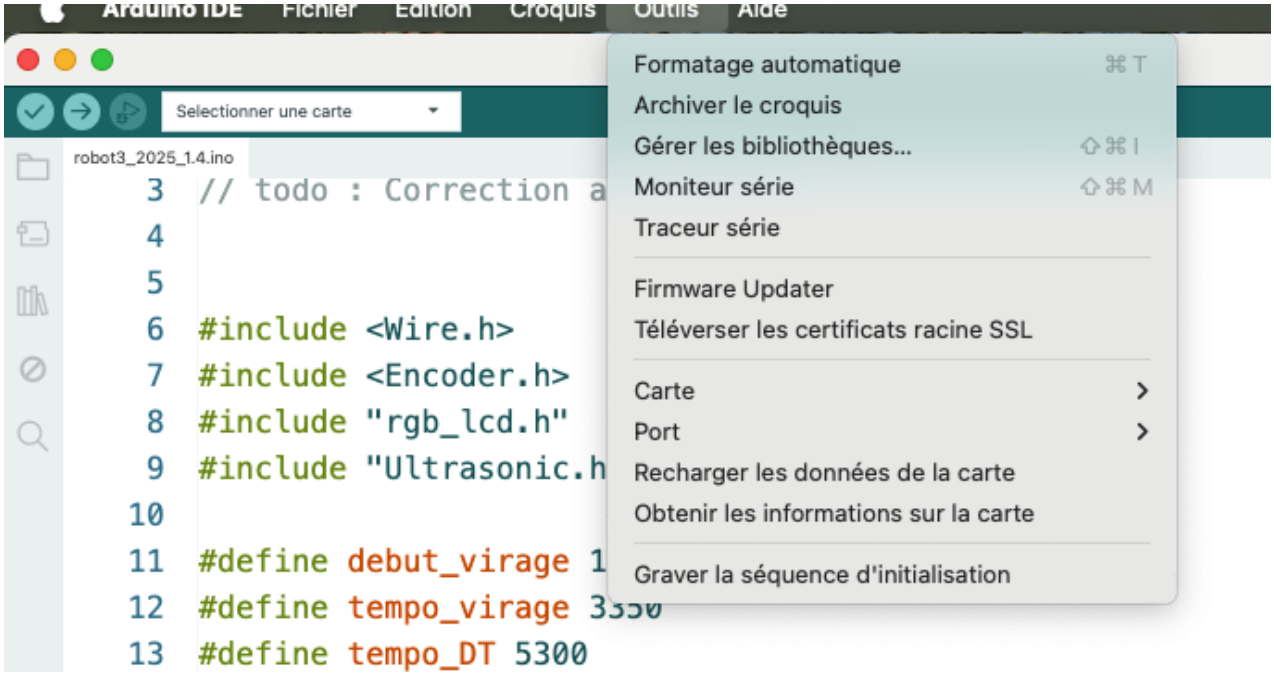
```
int carLU = 0;

void setup() {Serial.begin(9600);}

void loop() {
    carLU = readCar(); // Prélève un caractère sur le bus
    Serial.print("J'ai reçu : ");
    Serial.println(carLU, DEC);
}
```

Interface conversationnelle

Interface conversationnelle entre le micro-contrôleur et le moniteur sériel :

- A l'aide de la fonction readCar, écrire et tester individuellement :
  - lireSigne(),
  - lireChiffre(),
  - lireVirguleouPoint().
- A l'aide de ces fonctions, dessiner un algorithme qui permet la saisie.

Traceur serie

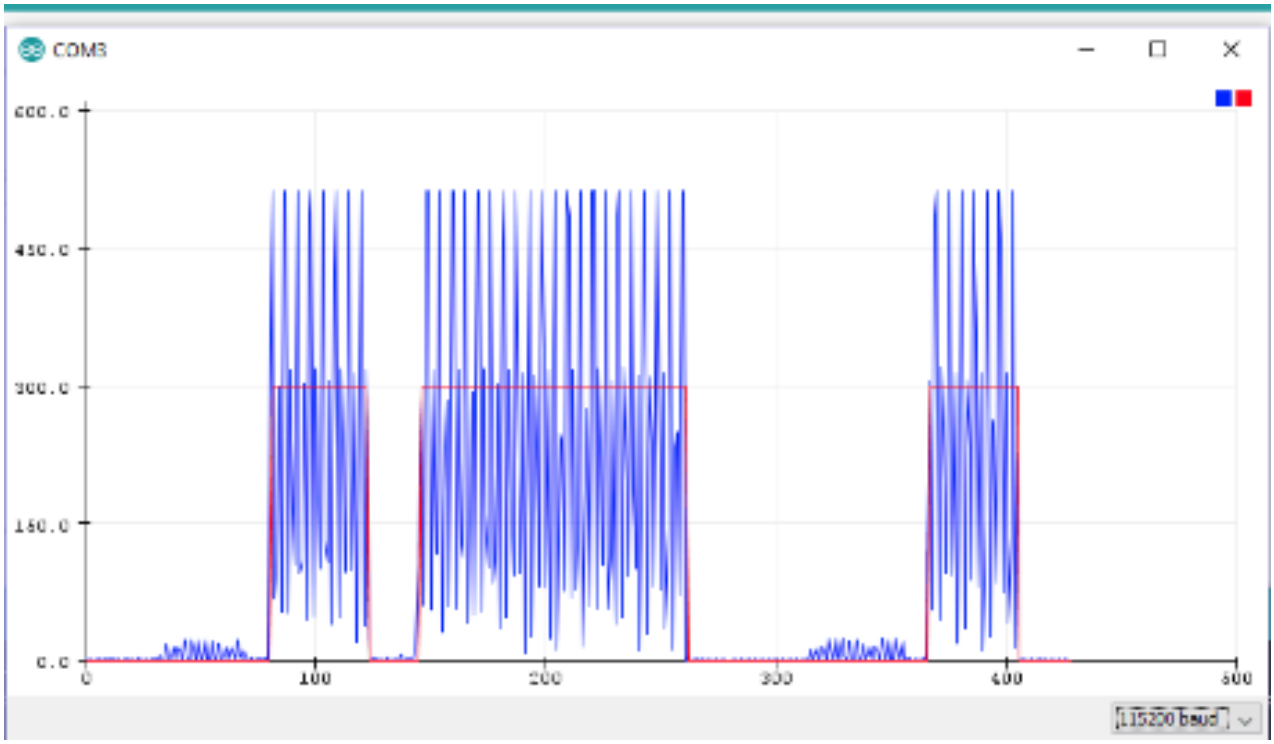


Traceur serie

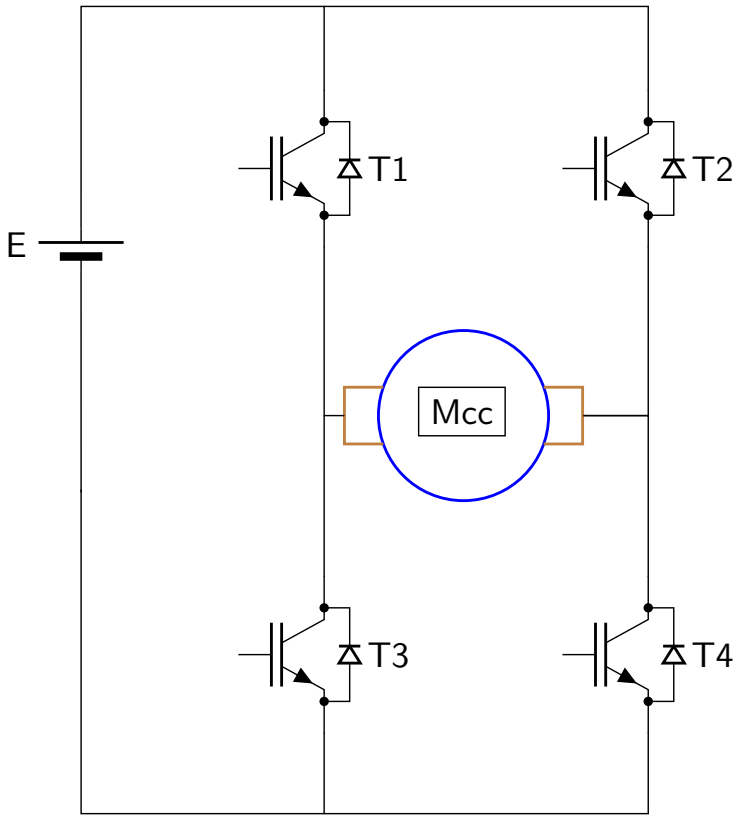
- Dans le code, les valeurs des courbes sont séparées par une virgule.
- La dernière valeur d'une courbe est suivie par le retour à la ligne.
- Exemple avec 2 courbes *Courant* et *Vitesse* :

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.print(Courant);
  Serial.print(',');
  Serial.print(Vitesse);
  Serial.print("\n");
}
```


Traceur serie



Hacheur L293 H-Bridge



+E aux bornes de la *MCC*

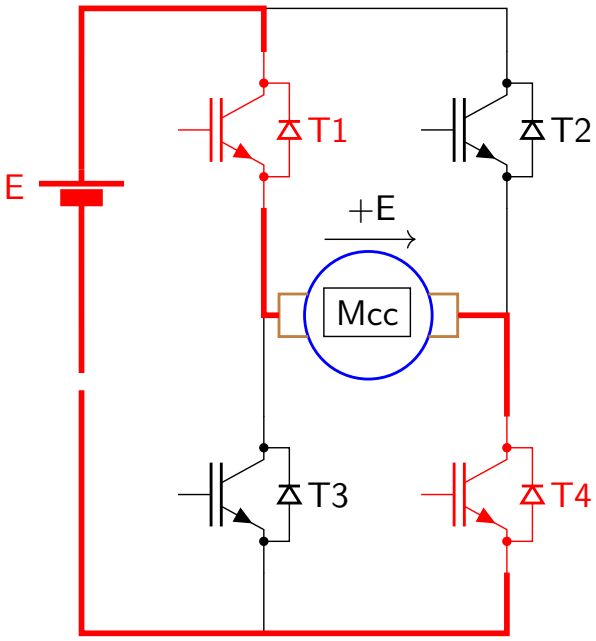


Figure: Hacheur

- T_1 et T_4 passant on a $V = +E$ aux bornes de la *MCC*

-E aux bornes de la *MCC*

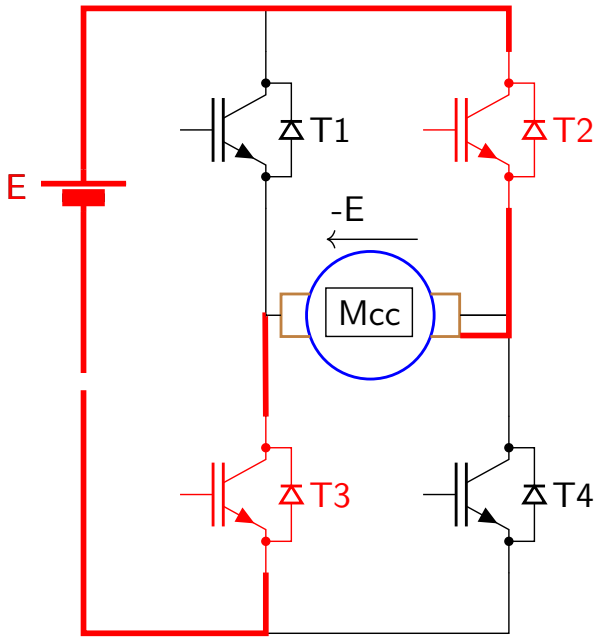
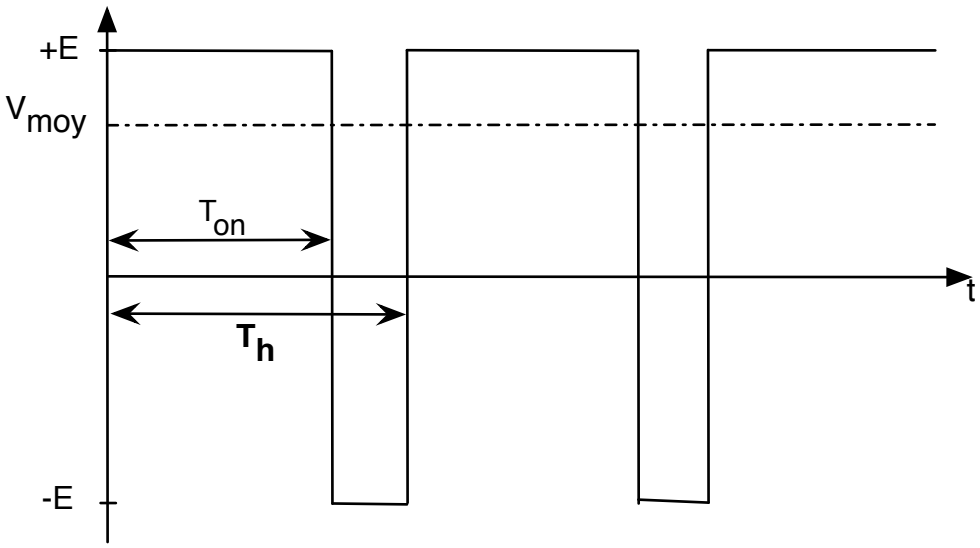


Figure: Hacheur

- T_2 et T_3 passant on a $V = -E$ aux bornes de la *MCC*

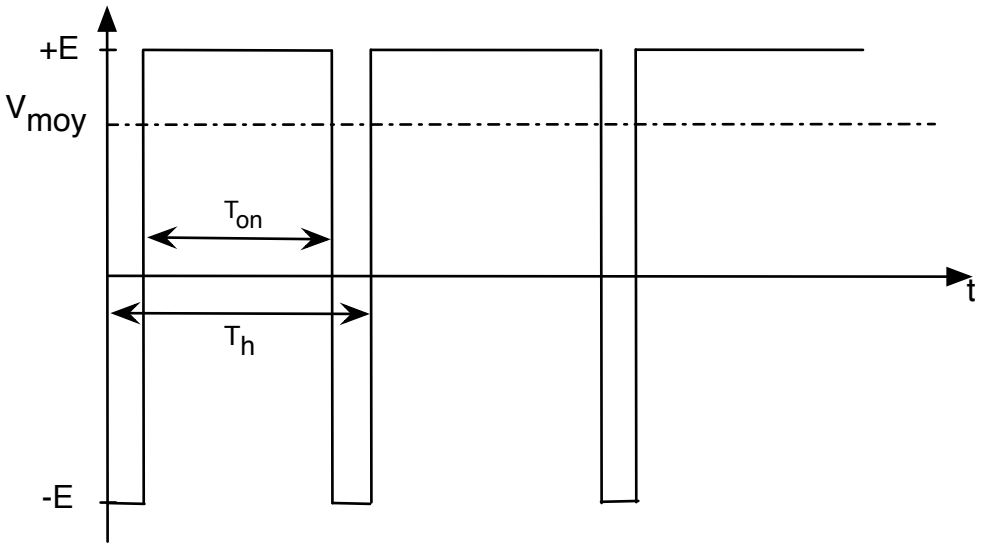
Consigne de tension hachée : T_{on} non centré



T_{on} non centré.

$$V_{moy} = \frac{1}{T_h}\left(\int_0^{T_{on}} E dt + \int_{Ton}^{T_h} -E dt\right)$$

T_{on} centré sur T_h



T_{on} centré.

$$V_{moy} = \frac{1}{T_h}\left(\int_0^{\frac{T_h - T_{on}}{2}} -E dt + \int_{\frac{T_h - T_{on}}{2}}^{\frac{T_h + T_{on}}{2}} E dt + \int_{\frac{T_h + T_{on}}{2}}^{T_h} -E dt\right)$$

Numérisation du Calcul

- V_{moy} est exprimé en volt,
- T_{on}, T_h, sont exprimés en secondes (microsecondes).
- dans un micro-contrôleur les registres sont sans unités
- Numérisation : Passer de variables typées à des registres
- Notation : Numérisation une variable $V \Rightarrow V^N$

$$\frac{T_{on}}{T_h} = \frac{T_{on}^N}{T_h^N} = \frac{OCR_{1A}}{OCR_{1B}}$$

Calcul d'expressions littérales

V_{moy} étant fourni par me conversationnel :

- Calculer les deux intégrales précédentes.
- En déduire l'expression littérale de V_{moy}
- En déduire l'expression littérale de T_{on}
- Calculer l'expression littérale du rapport $\frac{T_{on}}{T_h}$

Etude de la documentation du Timer (1/3)

A partir de la documentation ATmega2560_datasheet :

- Quels registres du timer 1 correspondent respectivement à T_h^N et à T_{on}^N
- Trouver dans ATmega2560_datasheet la table des modes du timer 1
- A quel mode du timer 1 correspond à une MLI centrée ?
- Choisissez T_h pour ne pas être dans vos fréquences audibles ($f_h >= 13kz$)
- Trouver la formule qui permet d'établir T_h^N. Donner la référence.
- Fig. 17.1, p 134, retrouver les 4 registres pour MLI sur la patte $OC1B$.

Etude de la documentation du Timer (2/3)

- Rechercher les rôles des 4 registres qui contrôlent le timer 1.
- Table des 16 modes des timers : déterminer quels sont les modes possibles et qu'est-ce qui les différencie ?
- Quelles valeurs doit-on mettre dans WGM_{13} WGM_{12} WGM_{11} WGM_{10} ?
- Trouver la figure qui montre dans un chronogramme les rôles des registres OCR_{1A} et OCR_{1B} en mode phase et fréquence correcte (MLI Centrée).
- Trouver la table qui définit comment déterminer les bits COM_{1B1} et COM_{1B0} et donner la référence.
- Traduire et reformuler la 3^{ieme} ligne de ce tableau.

Etude de la documentation du Timer (3/3)

- Donner les lignes d'initialisations des registres $TCCR_{1A}$ et $TCCR_{1B}$

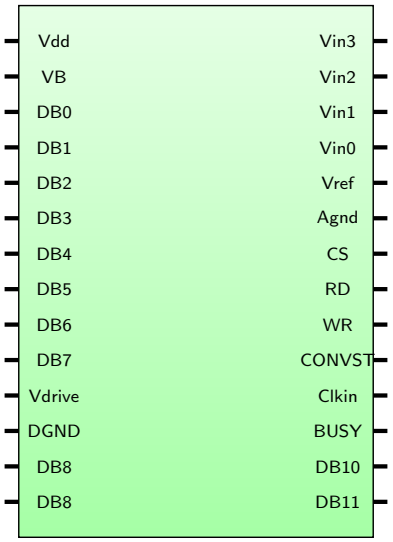
```
Exemple d'initialisation du Mode 11 :
   TCCR1A = (1<<bit3) + (1<< bit5);
   TCCR1B = (1<< bit1)+(1<<bit0);
```

- Compléter alors la fonction *initTimer*(), de façon à avoir une fréquence de hachage de $15khz$ et un rapport cyclique initial de 0.5
- Tester cette fonction. Tester d'autres fréquences et indiquer quelles est la fréquence la plus haute que vous pouvez discerner.

Convertisseur AD7934

Nous allons étudier et écrire un driver pour l'*AD*7934 :

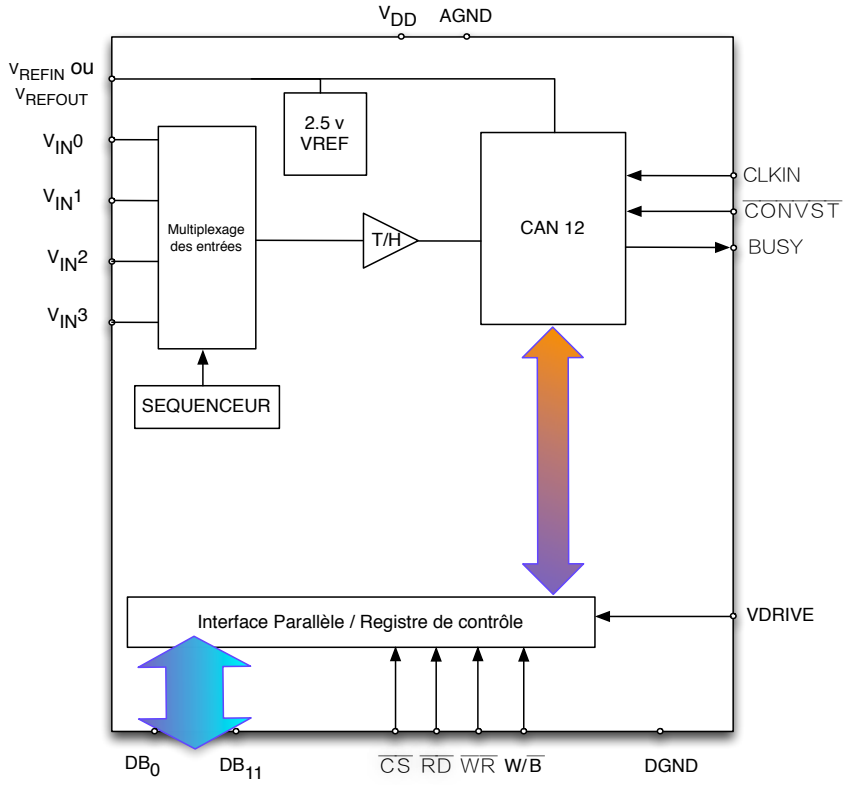
AD7934



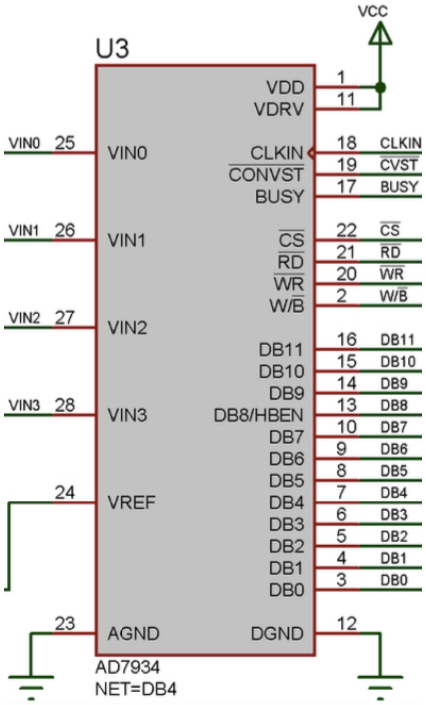
Convertisseur AD7934

- CAN 12 bits rapide 1.5 MSPS (Million Samples Per Second)
- 4 entrées analogiques $Vin_0, Vin_1, Vin_2, Vin_3$ en mode simple ou en mode différentiel
- Possibilité de connecter l'*AD*7934 à un bus d'adresse ($\overline{CS}, \overline{RD}, \overline{WR}, W/\overline{B}$) et un bus de données ($DB_0, ..., DB_{11}$) pour une lecture rapide de la conversion.
- Possibilité d'utiliser une référence interne précise de 2.5 v ou une référence externe ($VREF$).
- Echantillonnage déclenché par le signal $\overline{CONVST}$

Convertisseur AD7934

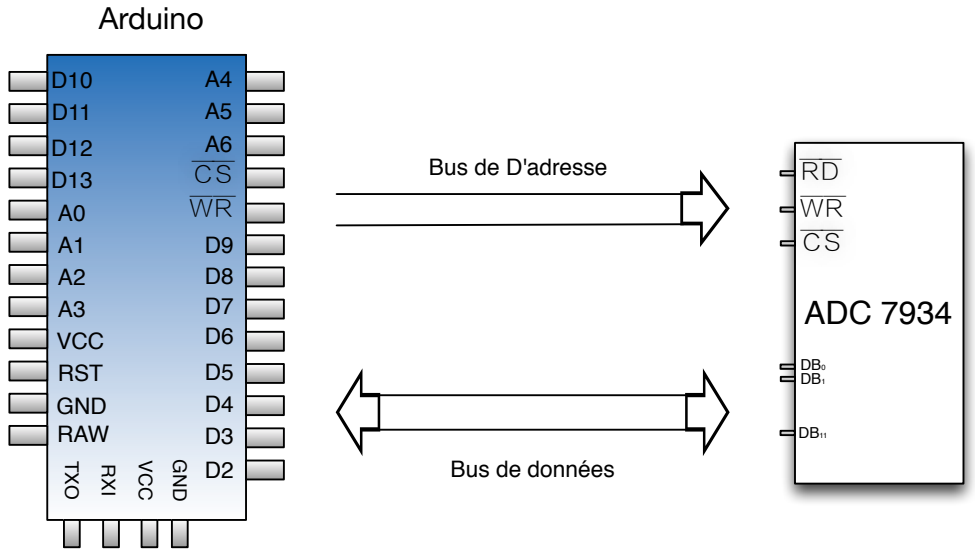


Etude du cablage



Bus d'adresses - Bus de données

Tout composant relié à un processeur peut être relié de la façon suivante :

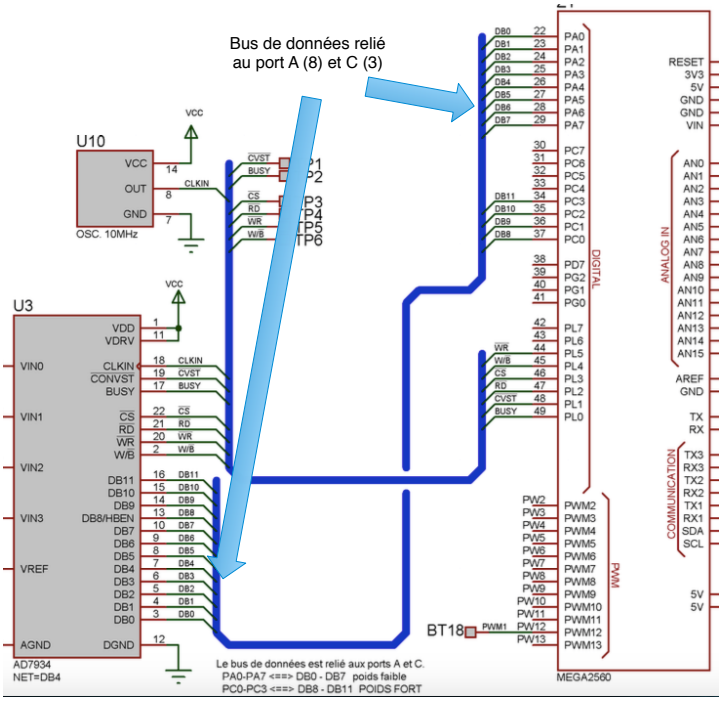


Bus de données

Bus de données (Data Bus) : $DB_0, ..., DB_{11}$

- Bus bidirectionnel.
- Initialisation : Bus en écriture sur : *ATMega* $\Rightarrow$ *AD7934*:
  Informations pour programmer les modes
- Conversion : Bus en lecture sur : *AD7934* $\Rightarrow$ *ATMega*:
  Resultat de conversion
- Il doit être connecté à des ports de l'*ATMega*

Bus de données

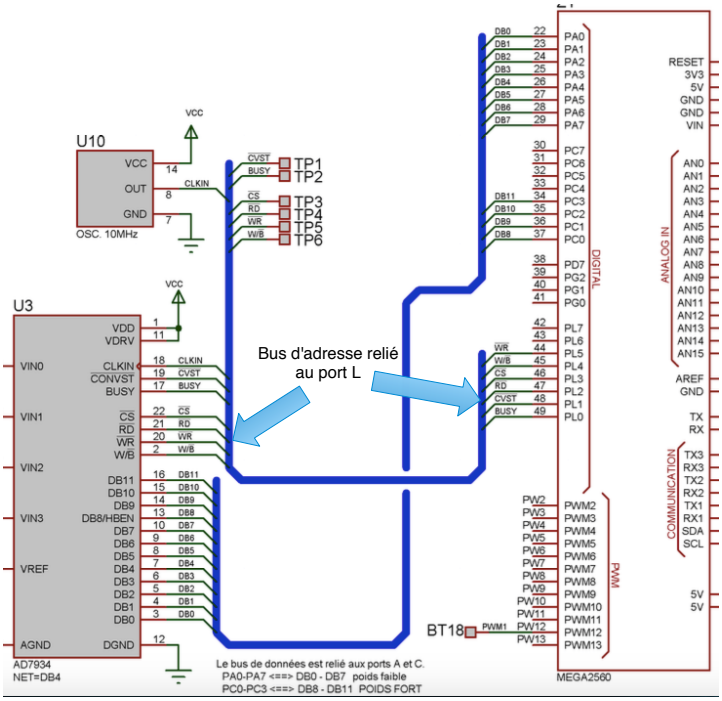


Bus d'adresse et de contrôle

Bus d'adresse et de contrôle :

- $\overline{CS}$: Chip Select
- $\overline{WR}$: Write
- $\overline{RD}$: Read
- $W/\overline{B}$: Write Byte
- $BUSY$: Conversion en cours
- $\overline{CONVST}$: Conv Start
- $CLKIN$: Horloge

Bus d'adresses (et de contrôle)



Analyse des liaisons entre l'*AD7934* et l'*ATMega*

A partir de la documentation CarteTP_ATMega :

- Identifier chaque liaison connectant une pin de l'*ATMega* à une pin de l'*AD7934*.
- Pour chacune des pins identifiées de l'*AD7934*, à partir de la documentation AD7934Bruz déterminer :
  - Le sens de circulation de l'information
  - Leurs rôles : Une simple phrase accompagnée de la réference (numéro de page).
  - Exemple pour $\overline{CONVST}$: Un front descendant de ce signal démarre une conversion (p 23, AD7934Bruz) .

Entrées analogiques de l'*AD7934*

A partir de la documentation CarteTP_ATMega :

- Identifier les entrées analogiques du convertisseur.
- A partir de la documentation AD7934Bruz expliquer comment ces entrées anlogiques peuvent être utilisées suivant 4 modes différents.
- En examinant ce montage, quel est le mode utilisé ?

Entrées differentielles

A partir de la documentation CarteTP_ATMega et de AD7934Bruz :

- Les *bnc* J_1 et J_2 acceptent des tensions sur l'intervalle $[-10, +10v]$
- Le composant $U8$ (Traco) convertit les tensions dans la plage $[-12v, +12v]$ qui est nécessaire pour les *AOP*
- Puis SUB_1 et SUB_2 vont produire deux tensions antagonistes positives centrées autour de $VREF = 2.5v$
- Expliquer pourquoi SUB_1 et SUB_2 produisent de telles sorties.
- Chercher l'opération arithmétique (indiquez la référence dans la documentation) que va faire l'*AD7934* sur chaque paire de signaux (VIN_0, VIN_1) et (VIN_2, VIN_3) .
- Quel est l'intérêt de cette opération arithmétique.
- Donner un exemple de tension sur V_A, calculer la paire VIN_0, VIN_1, et donner le résultat de l'opération de l'*AD7934* avant la conversion.
- Expliquer ce qui se passe.

Registre de contrôle

- Le registre de contrôle est un mot de 12 bits.
- Il permet de programmer différents modes de fonctionnement.
- On l'initialise au début de programme.

Mot de contrôle

PM_1 PM_0 $CODING$ REF $ZERO$ ADD_1 ADD_0 $MODE_1$ $MODE_0$ SEQ_1 SEQ_0 $RANGE$

- Donner le rôle de ces bits par fonctionalités
- En déduire la valeur du mot pour la conversion du courant
- En déduire la valeur du mot pour la conversion de la vitesse

Ecriture du driver AD7934

Le driver va se faire en codant deux fonctions :

- Fonction *void programAD7934(int voie)*: Permet de définir le mode de fonctionnement de *AD7934* et de définir le canal sur lequel la convertsion sera faite
- Fonction *int lireAD7934(void)*: Permet de lire le résultat de conversion de l'*AD7934* sur le canal prédéfini par l'autre fonction

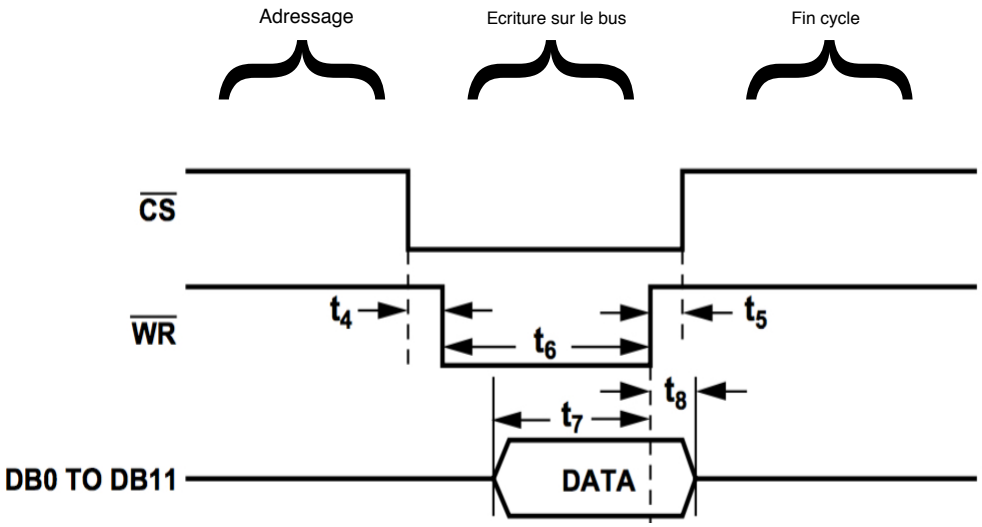
Fonction *void programAD7934(int voie)*: Ecriture du mot de contrôle l'*ATMega* sur l'*AD7934*

La programmation de l'*AD7934* se réalisera alors par 3 phases :

- Adressage : Le positionnement d'une valeur sur le bus d'adresse ($PORT_L$) qui permet d'activer et d'écrire sur le composant sur le bus d'adresse (avec insertion éventuelle de délais)
- Ecriture sur le bus de données ($PORT_A$ et $PORT_C$)
- Fin de cycle : Le positionnement d'une nouvelle valeur sur le bus d'adresse qui désactive le composant et le mode écriture du composant sur le bus d'adresse

Chronogramme correspondant à la fonction *void programAD7934(int voie)*

Voici le chronogramme d'écriture de l'*AD7934* :



Description du chronogramme précédent

Dans la figure précédente :

- C'est la conjonction de $\overline{WR}$ et $\overline{CS}$ qui précède l'écriture d'un mot de 12 bits sur le bus de données. Attention vous devez positionner une valeur sur tous les bits d'adressage $\overline{WR}$, $\overline{RD}$, $W/\overline{B}$, $\overline{CONVST}$, $\overline{CS}$.
- Détaillons la figure précédente :
  - $\overline{CS}$ et $\overline{WR}$ active l'*AD7934*
  - La data est écrite dans l'*AD7934* après que le signal $\overline{WR}$ soir retombé au niveau bas.
  - L'écriture nécessite un temps de maintien t_7 avant la remontée de $\overline{WR}$.
  - $\overline{CS}$ et $\overline{WR}$ sont théoriquement liés par t_4 et t_5
  - Cherchez dans AD7934Bruz, la table des temporisations qui donne les valeurs de t_4 et t_5
  - Completer maintenant la fonction *void programAD7934(int voie)* qui, selon la voie (COURANT ou VITESSE), initie les modes de fonctionnement de l'*AD7934*

Fonction *int lireAD7934(void)*() : Lancement de conversion sur l'*AD7934* et lecture du résultat

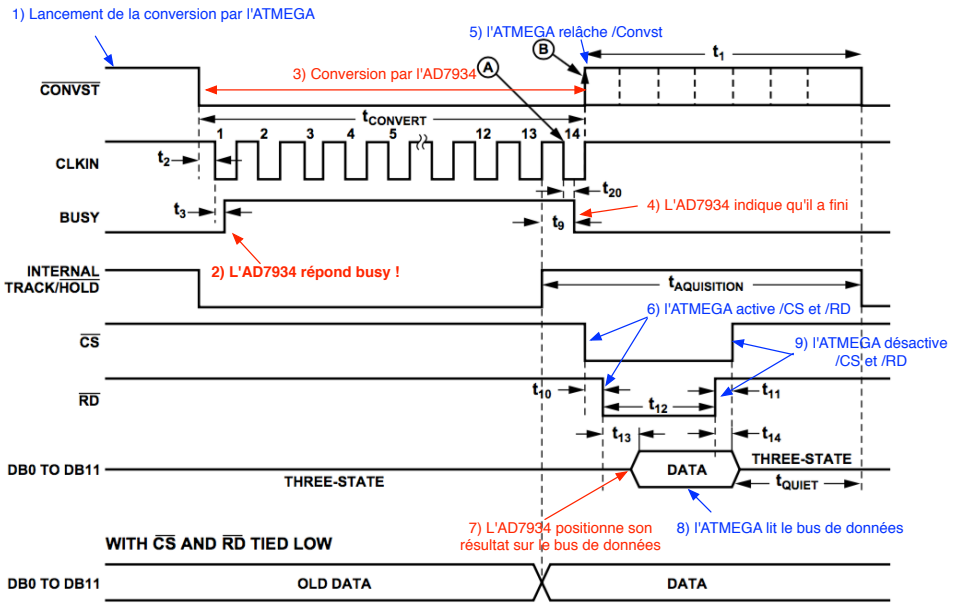


Figure 34. AD7933/AD7934 Parallel Interface—Conversion and Read Cycle in Word Mode ($W/\overline{B}$ = 1)

Description du chronogramme précédent

Voici les phases principales du chronogramme précédent :

1. *ATMega* envoie le signal $\overline{CONVST}$ qui lance la conversion
2. *AD7934* active *BUSY* pour indiquer qu'une conversion est en cours
3. $\overline{CONVST}$ et *BUSY* restent à l'état haut pendant tout le cycle.
   * Le temps de conversion dure 14 cycles d'horloge ($t_{convert}$)
   * Entre 2 conversions $\overline{CONVST}$ est à l'état haut pendant t_1
4. Fin de la conversion *AD7934* relâche *BUSY* pour informer l'*ATMega*
5. *ATMega* relâche $\overline{CONVST}$ et commence la lecture du résultat de la conversion
6. *ATMega* active $\overline{CS}$ et $\overline{RD}$: Adressage de l'*AD7934*
   * $\overline{CS}$ et $\overline{RD}$ qui sont théoriquement liés par t_{10} et t_{11}
   * Retrouver dans AD7934Bruz ces valeurs
7. Données placées par *AD7934* sur bus de données après retombée de $\overline{CS}$ et $\overline{RD}$
8. *ATMega* lit le résultat de la conversion sur le bus de données
9. *ATMega* désactive $\overline{CS}$ et $\overline{RD}$

Test de *int lireAD7934(void)* et *void programAD7934(int voie)*

- Tester maintenant les fonctions *void programAD7934(int voie)* et *int lireAD7934(void)* en lisant alternativement le courant et la vitesse que vous afficherez sur le traceur série.
- Appliquer une charge sur le moteur : Expliquer l'évolution des courbes.