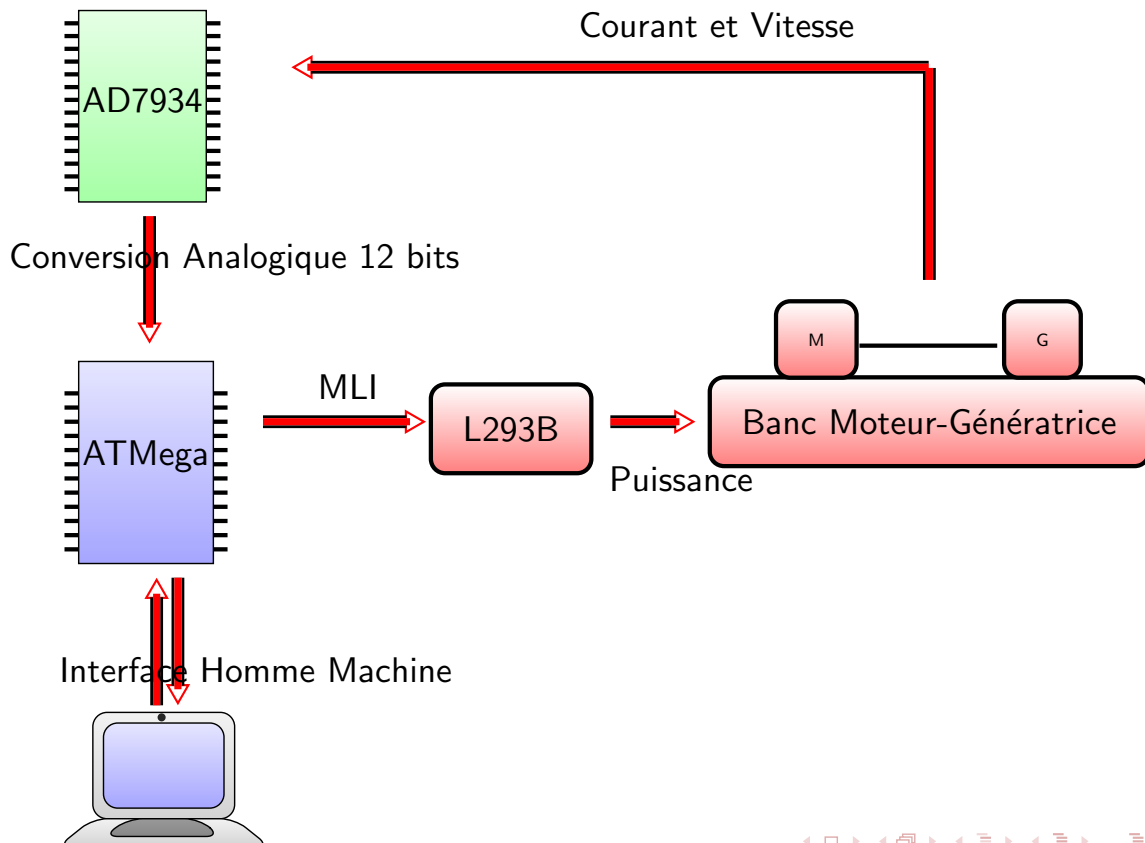


Micro-Projet d'Informatique Industrielle S7

Nadia Aït Ahmed, David Delfieu

October 8, 2024

Commande en boucle ouverte d'une MCC



1 Conversationnel sous Arduino

- Bibliothèque serial
- Moniteur Serie
- Algorithme du conversationnel
- Traceur serie

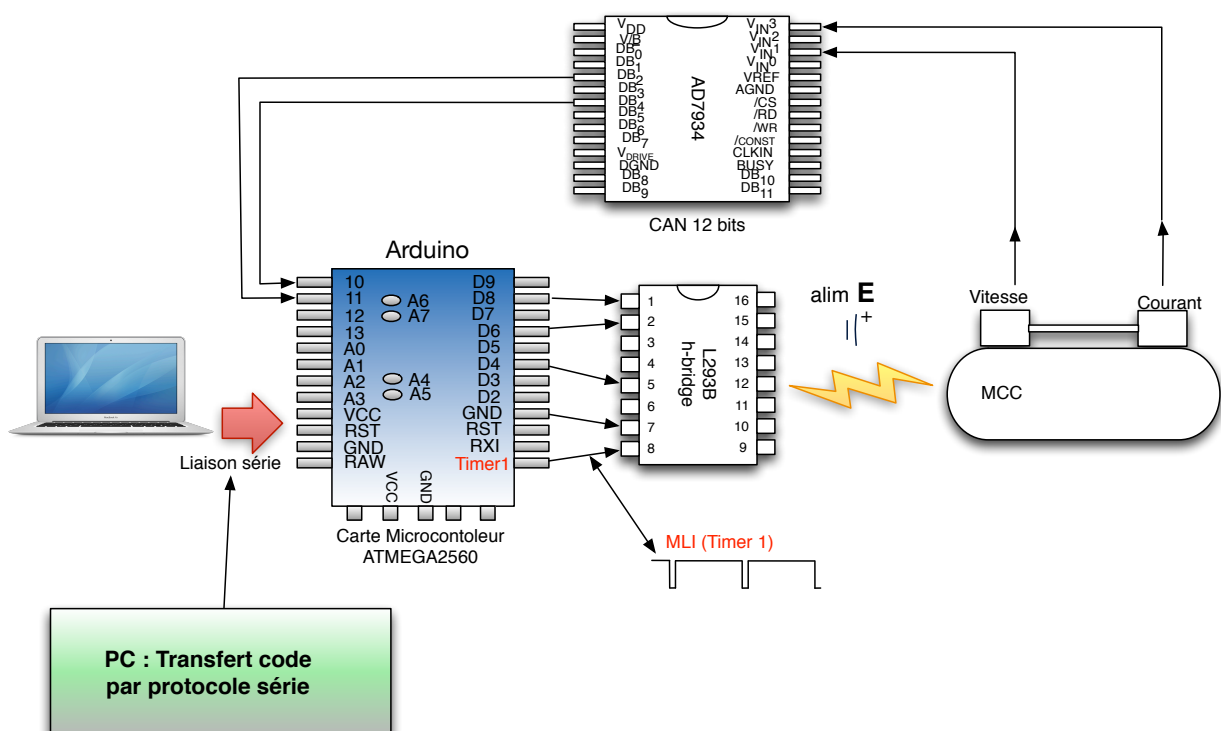
2 Génération de la MLI

- Hacheur et MCC
- Programmation du Timer de l'ATMEGA2560
- Les Timers

3 Le Convertisseur 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 RS232-USB
- RS232 ⇒ bibliothèque *Serial*
- Pins en TTL (5V ou 3.3V):
 - Transmission TxD (PD3)
 - Reception RxD (PD2)
- ATMEGA₂₅₆₀ : 3 ports séries additionnels:

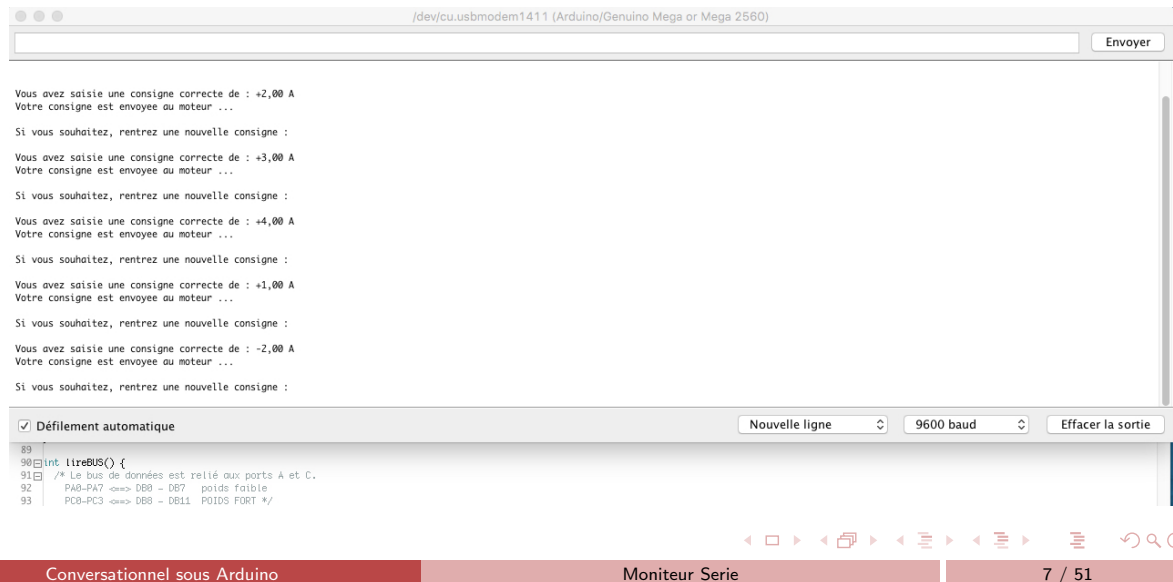
Le Moniteur Serie

- 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);

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 un saut de ligne"  
Serial.print("\n Voici la valeur de A = ");  
Serial.print(A);
```

Serial.read

```
S=Serial.read();  
A=Serial.read();  
B=Serial.read();  
Si l'utilisateur tapes +15 alors  
S vaut \x2B : code ascii de +  
A vaut \x31 : code ascii de 1 (48 en base 10)  
B vaut \x35 : code ascii de 5 (53 en base 10)
```

Lecture sur le bus série

- Le bus série permet une communication de type asynchrone :
 - ▶ 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.

readCar()

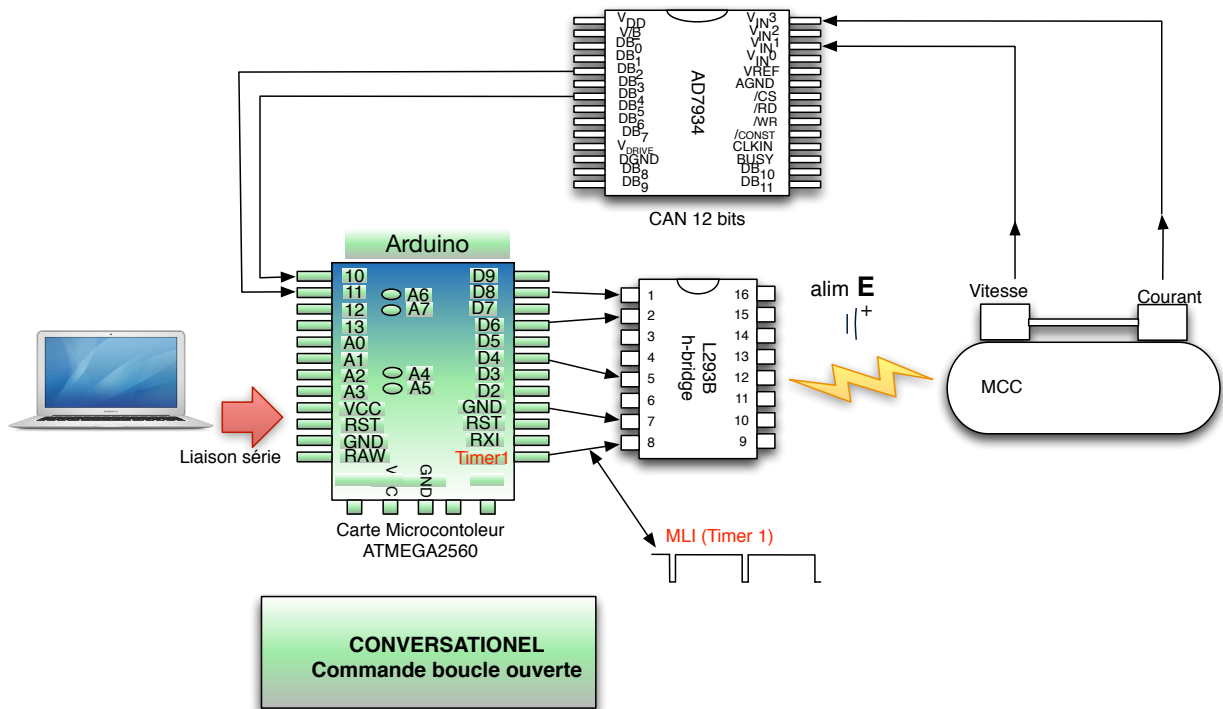
- ReadCar() récupère un caractère dès qu'il apparaît sur le bus :

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

- Utilisation de readCar() :

```
int carLU = 0; // caractère lu sur le bus série  
  
void setup() {Serial.begin(9600);}   
  
void loop() {  
    carLU = readCar(); // On lit un caractère sur le bus  
    Serial.print("J'ai reçu : ");  
    Serial.println(carLU, DEC);  
}
```

Conversationnel

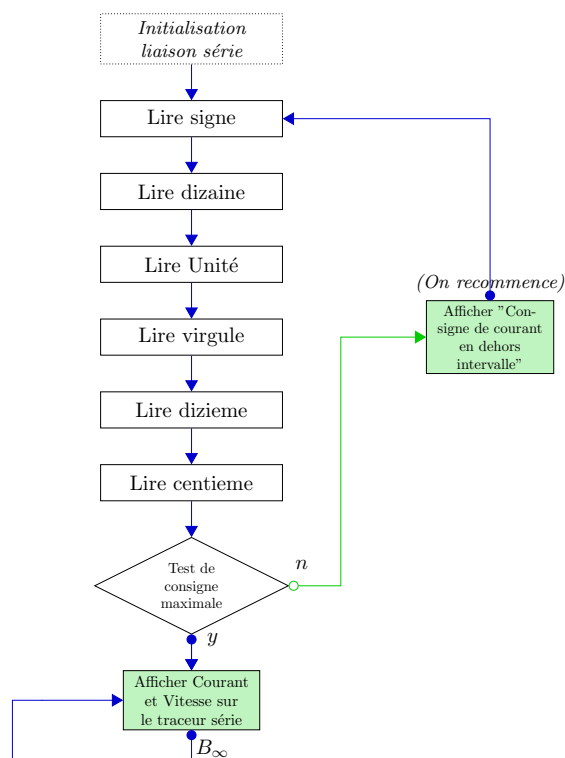


Conversationnel

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

- A l'aide de la fonction `readCar()`, sur le modèle de `lectureSigne()` du squelette
- Ecrire la fonction `lireChiffre()`
- A l'aide de cette fonction, écrire l'algorithme du slide suivant.
- L'algorithme devra produire la consigne en format flottant.

Organigramme simple, non robuste



Traceur serie

TraceurCourbe | Arduino 1.8.9

Fichier Édition Croquis **Outils** Aide

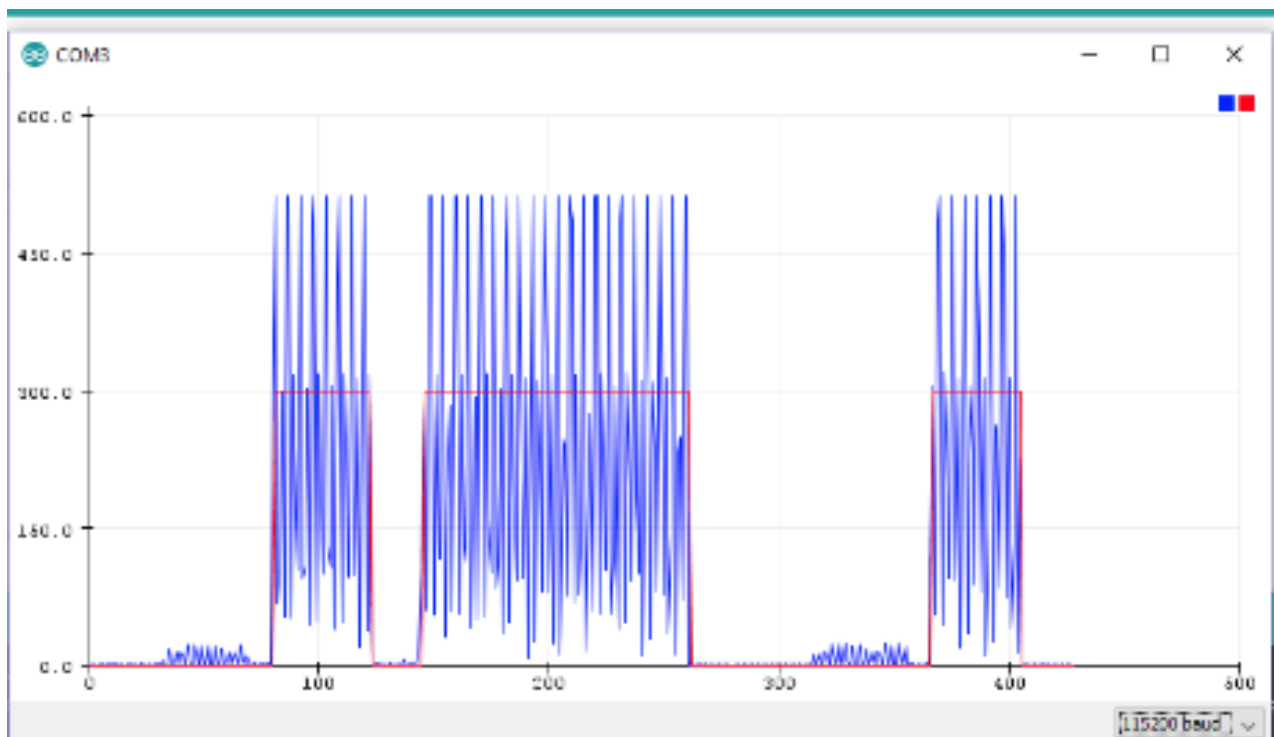
```
149 {
150   Val
151 }
152 else
153 {
154   Val
155 }*/
156 // Ve
157 Val_T
158
159 // At
160 delay(20);
161 }
162
```


Traceur serie

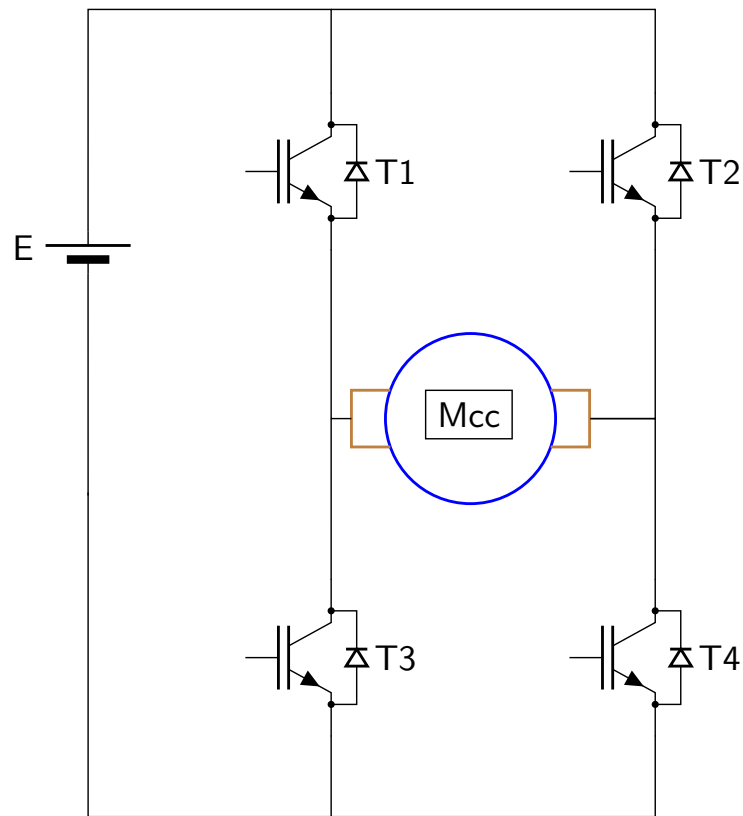
- Dans le code les valeurs des courbes sont séparées par une virgule.
- La dernière valeur d'une courbe est suivie par le retour à la ligne.
- Exemple avec 2 courbes *Courant* et *Vitesse* :

```
void setup() {  
  Serial.begin(9600);  
}  
  
void loop() {  
  Serial.print(Courant);  
  Serial.print(',');  
  Serial.print(Vitesse);  
  Serial.print("\n");  
}
```

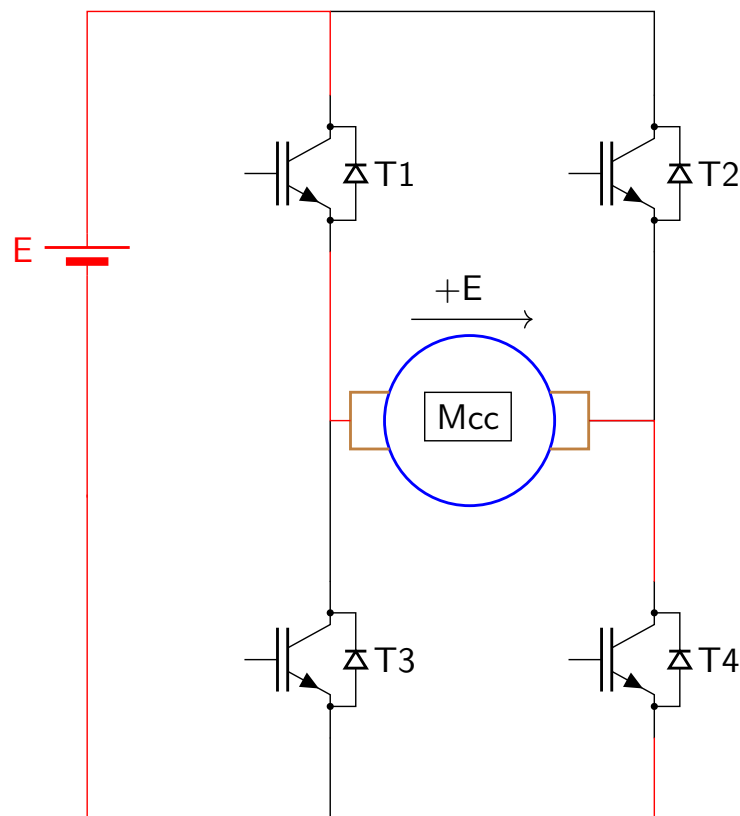
Traceur serie



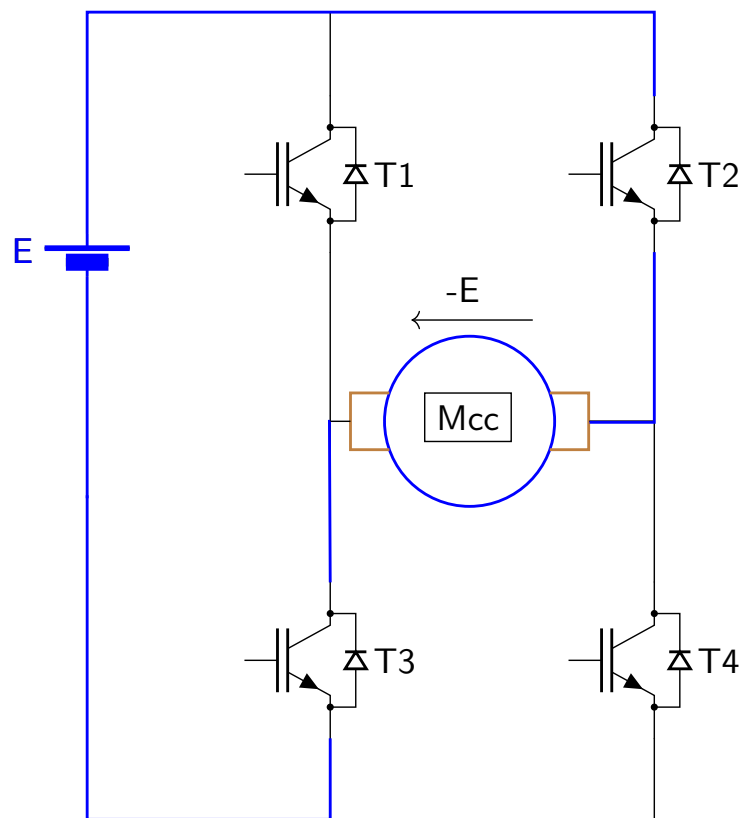
Principes de commande du hacheur (L293 H-Bridge)



Principes de commande du hacheur

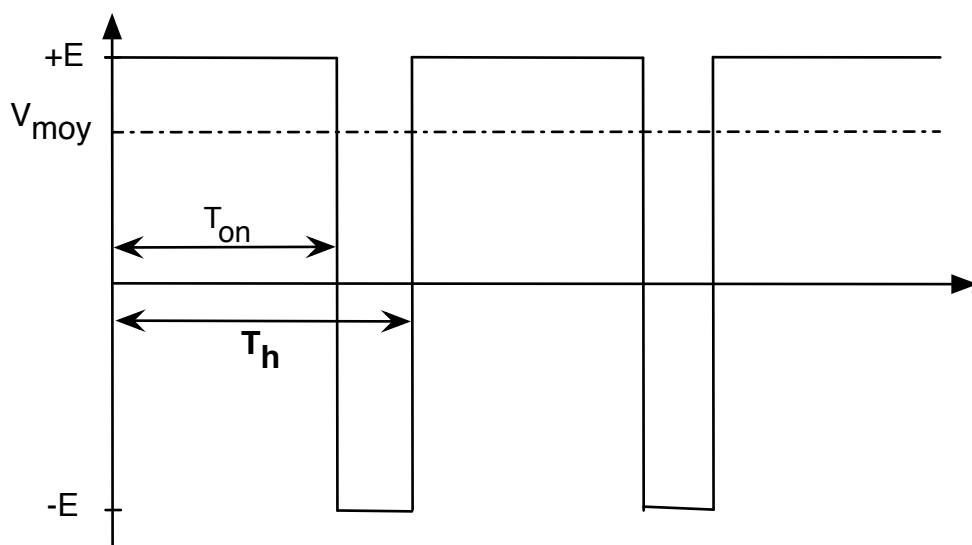


Principes de commande du hacheur



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

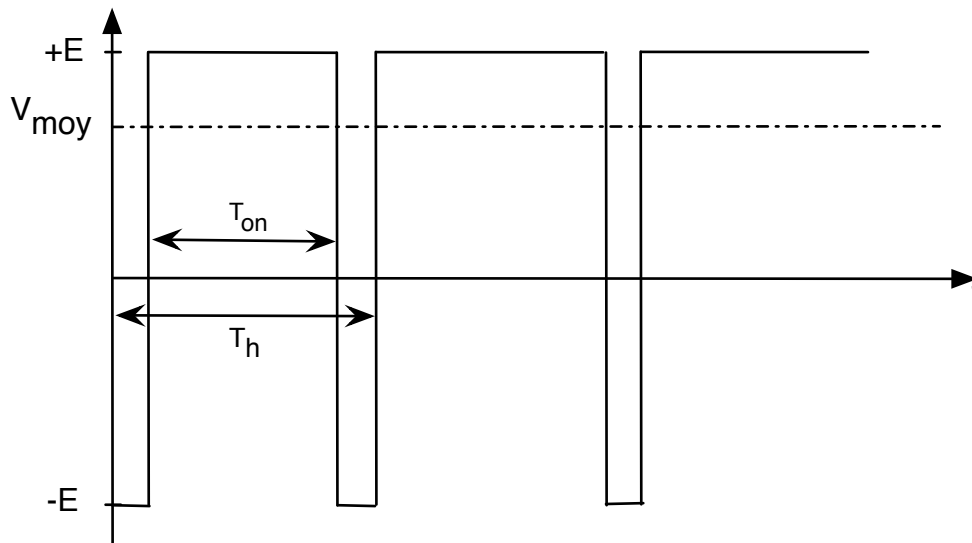
Consigne de tension hachée : T_{on} non centré



T_{on} non centré.

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

T_{on} centré sur T_h



T_{on} centré.

Numérisation du Calcul

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

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

Conversationnel en virgule fixe : V_{moy}

C'est le conversationnel qui fournit V_{moy} :

- Si on considère que l'utilisateur a tapé +21,37
- E est connu : $E = 35$ v.
- T_h est fixé pour ne pas être dans les fréquences audibles de vos oreilles on pourra prendre $f_h \geq 13$ kHz
- D'après l'équation précédente :

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

- Donc pour produire V_{moy} avec un timer je vais devoir déterminer T_{on} :

Calcul de V_{moy}

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

- Calculer cette intégrale et exprimer V_{moy} en fonction de E , T_{on} , T_h
- Calculer l'expression littérale du rapport $\frac{T_{on}}{T_h}$

Mise en oeuvre de la MLI

- Timer produit T_h^N (OCR_{1A}) et T_{on}^N (OCR_{1B})
- Timer devra générer une MLI centrée
- Trouver dans [ATmega2560_datasheet](#) la table des modes du timer 1
- Quel terme est utilisé dans cette documentation pour désigner une MLI centrée ?

Les Timers

Dans la documentation [ATmega2560_datasheet](#) :

- Trouver dans la documentation la formule qui permet d'établir T_h^N . Donner la référence.
- A partir de la figure 17.1 de la page 134, retrouver les 4 registres qui sont importants pour la génération d'une MLI sur la sortie B, c.a.d $OC1B$.
- Rechercher dans la doc les rôles des 4 registres et donner les références.
- Dans la 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} ?

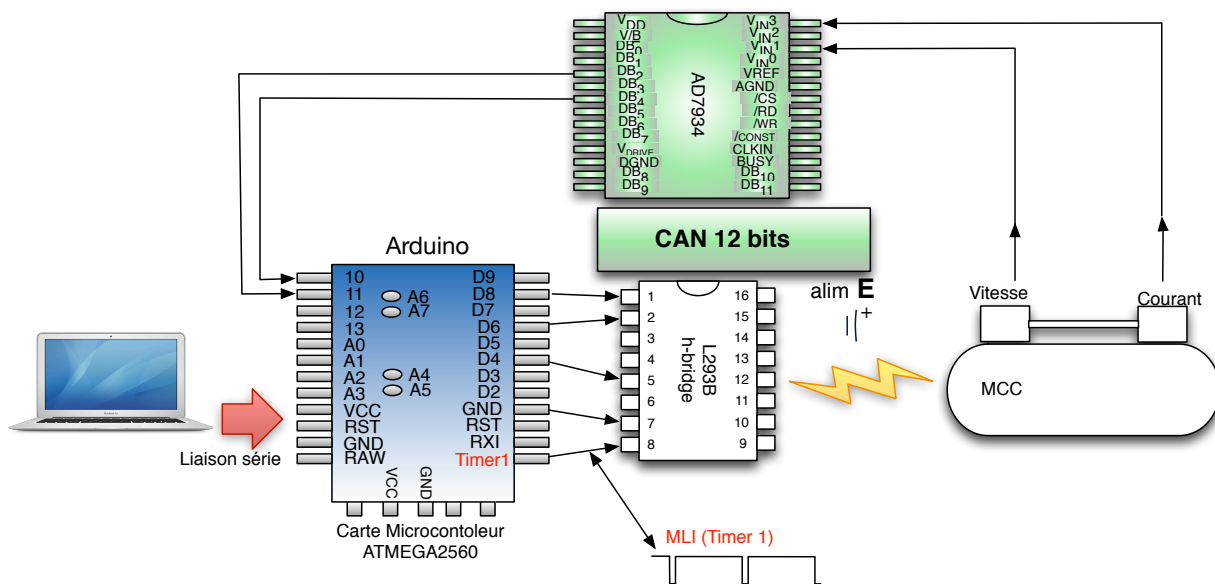
Rôles de OCR_{1B} et OCR_{1A}

- Trouver la figure qui montre dans un chonogramme 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^{ème} ligne de ce tableau.

Initialisations des registres $TCCR_{1A}$ et $TCCR_{1B}$

- Donner les lignes d'initialisations des registres $TCCR_{1A}$ et $TCCR_{1B}$
Exemple d'initialisation du Mode 11 :
 $TCCR1A = (1 \ll WGM11) + (1 \ll WGM10);$
 $TCCR1B = (1 \ll WGM13) + (1 \ll CS10);$
- 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

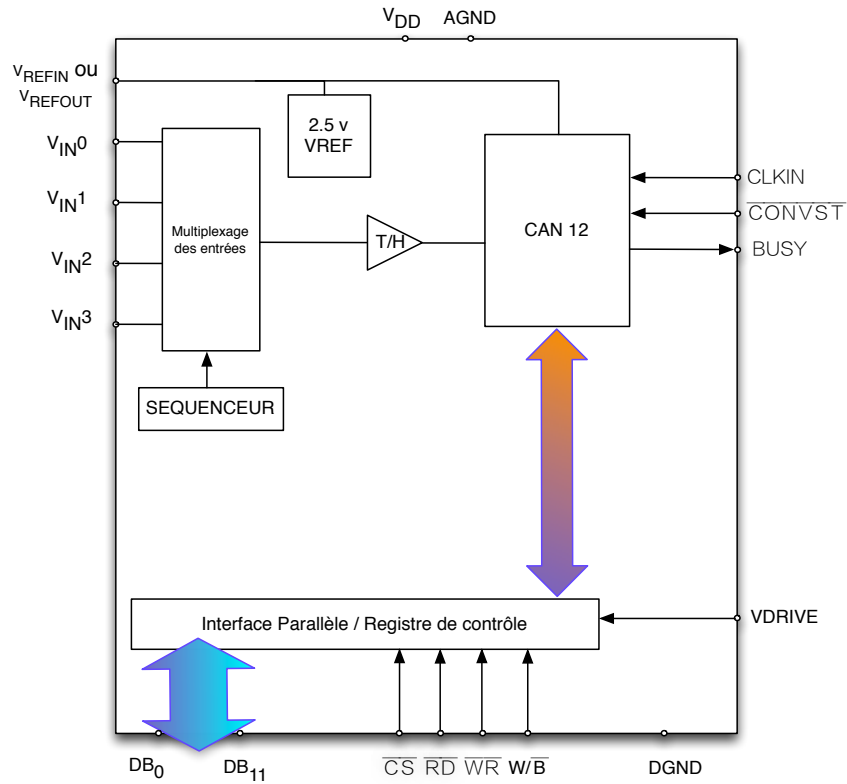
Convertisseur AD7934



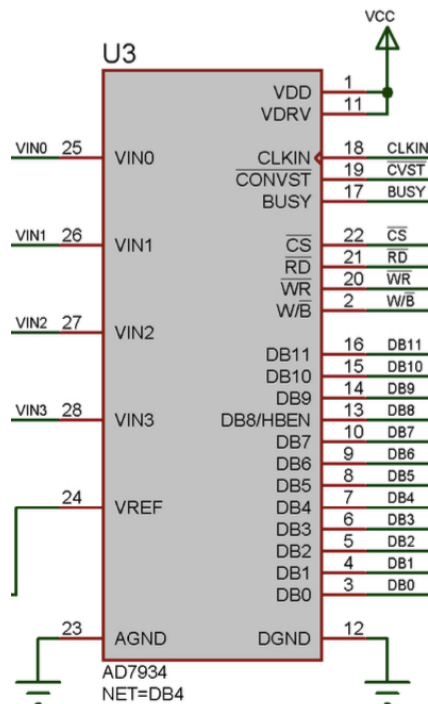
Convertisseur AD7934

- CAN 12 bits rapide 1.5 MSPS (Million Samples Per Second)
- 4 entrées analogiques séquençables en mode simple ou en mode différentiel
- Possibilité de connecter l'AD7934 à un bus d'adresse (\overline{CS} , \overline{RD} , \overline{WR} , $\overline{W/B}$) et un bus de données (DB_0, \dots, DB_{11}) pour une lecture rapide de la conversion.
- Possibilité d'utiliser une référence interne précise de 2.5 v ou une référence externe (V_{REF}).
- Echantillonnage déclenché par le signal \overline{CONVST}

Convertisseur AD7934

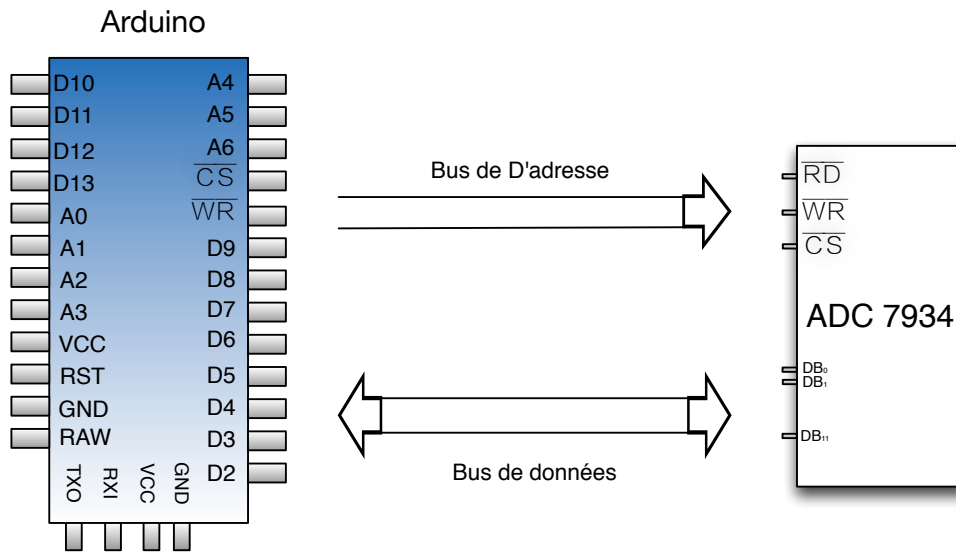


Présentation



Bus d'adresses - Bus de données

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

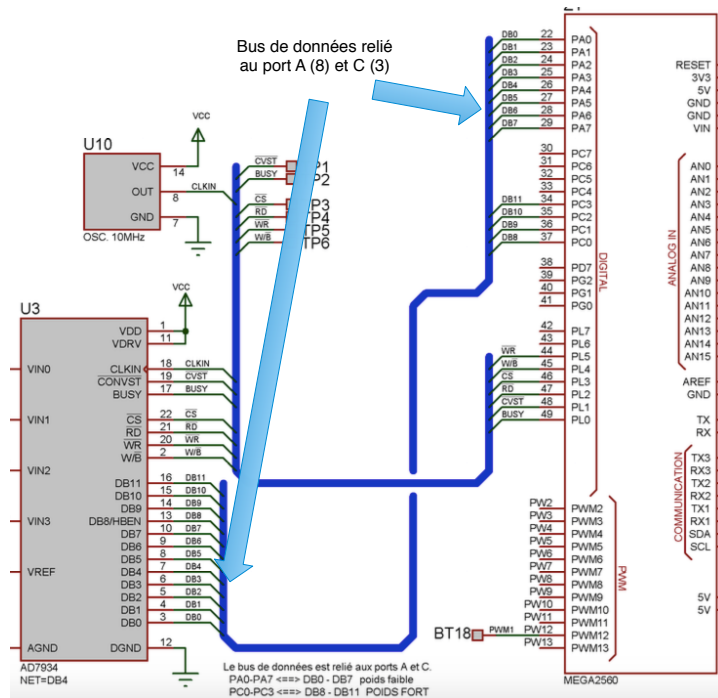


Bus de données

Bus de données (Data Bus) : DB_0, \dots, 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



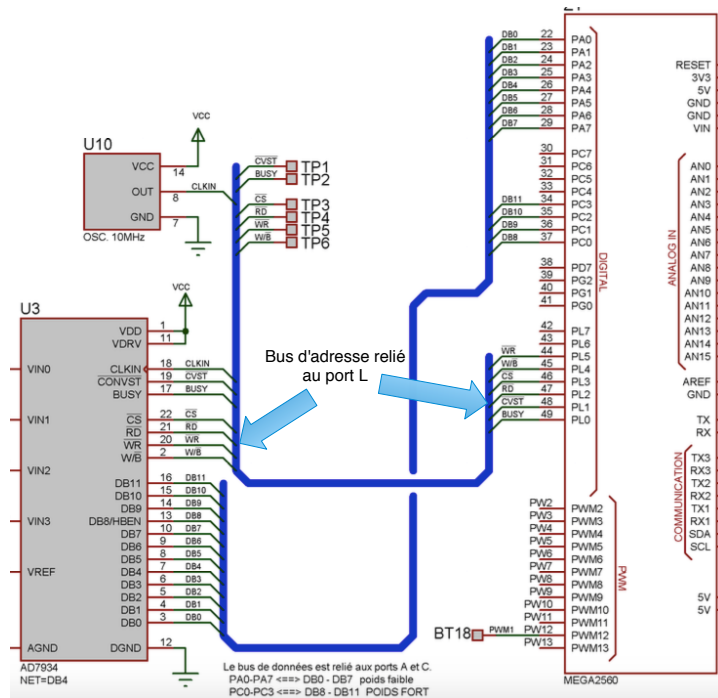
Bus d'adresse et de contrôle

Bus d'adresse et de contrôle :

- \overline{CS} , \overline{WR} , \overline{RD} et W/\overline{B} servent à adresser le composant
- $BUSY$, \overline{CONVST} et $CLKIN$ servent à contrôler le composant



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 référence (numéro de page).
 - ▶ Exemple pour 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 analogiques peuvent être utilisées suivant 4 modes différents.
- En examinant ce montage, quel est le mode utilisé ?

Entrées différentielles

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

PM₁ PM₀ CODING REF ZERO ADD₁ ADD₀ MODE₁ MODE₀ SEQ₁ SEQ₀ RANGE

- Donner le rôle de ces bits par fonctionnalités
- 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 conversion sera faite
- Fonction *int lireAD7934(void)*: Permet de lire le résultat de conversion de l'**AD7934** sur le canal prédéfini par l'autre fonction

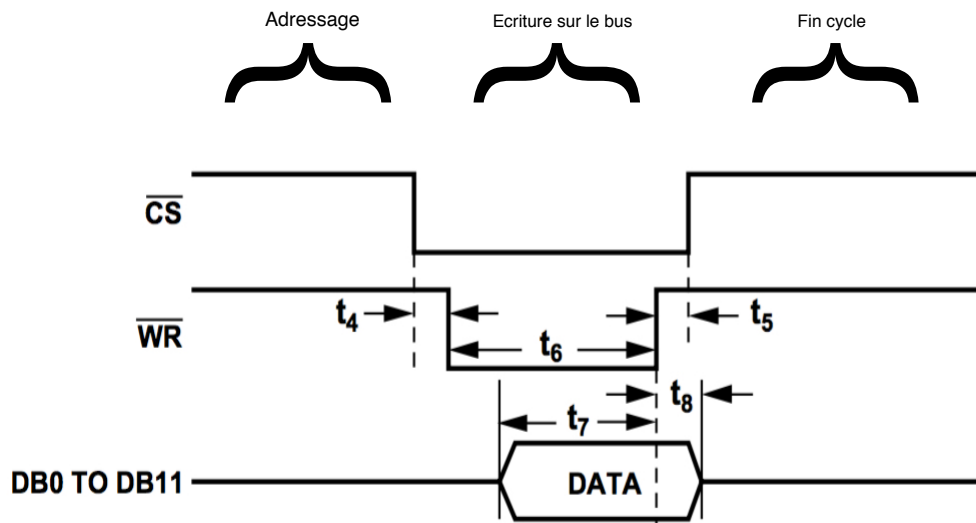
Fonction *void programAD7934(int voie)*: Ecriture du mot de contrôle l'**ATMega** sur l'**AD7934**

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/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} soit 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
 - ▶ Compléter 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

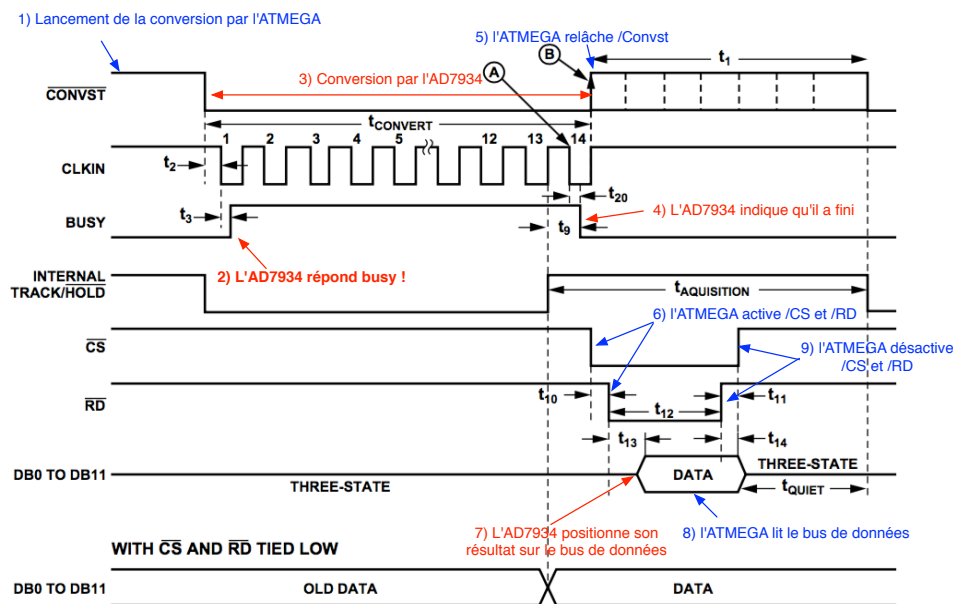


Figure 34. AD7933/AD7934 Parallel Interface—Conversion and Read Cycle in Word Mode ($W/\overline{B} = 1$)

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.