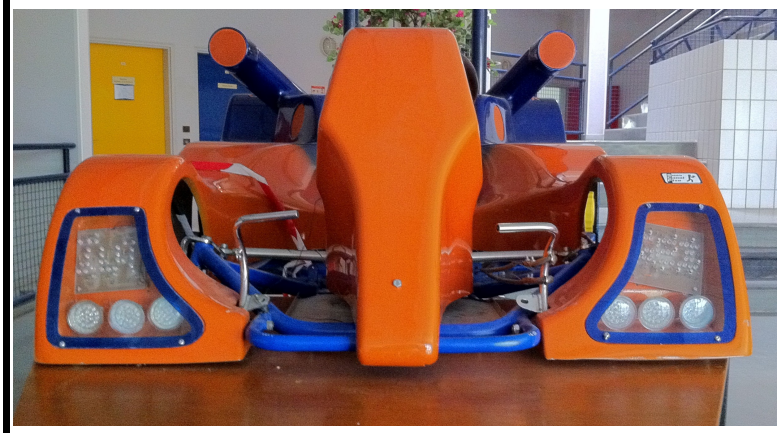
Université François-Rabelais de Tours
Institut Universitaire de Technologie de Tours
Département Génie Électrique et Informatique Industrielle

UNIVERSITE FRANCOIS-RABELAIS
TOURS
Institut Universitaire de Technologie
Département
GENIE ELECTRIQUE ET
INFORMATIQUE INDUSTRIELLE

Projet tuteuré 2ème année – S4



Gestion de l'allumage des feux du Kart électrique GEII

Guillaume ESCARIEUX
Thomas SOUFFEZ
2ème Année – Q1
Promotion 2010/2012

Enseignants :
M. LEQUEU Thierry
M. GLIKSOHN Charles

Université François-Rabelais de Tours
Institut Universitaire de Technologie de Tours
Département Génie Électrique et Informatique Industrielle

UNIVERSITE FRANCOIS-RABELAIS
TOURS
Institut Universitaire de Technologie
Département
GENIE ELECTRIQUE ET
INFORMATIQUE INDUSTRIELLE

Projet tuteuré - S4

Gestion de l'allumage des feux du Kart électrique GEII

Guillaume ESCARIEUX
Thomas SOUFFEZ
2ème Année – Q1
Promotion 2010/2012

Enseignants :
M. LEQUEU Thierry
M. GLIKSOHN Charles

Sommaire

Introduction

Dans le cadre du cours d'Études et Réalisations, nous avons réalisé un projet pratique. Ce projet a pour but de mettre en pratique nos compétences acquises à l'IUT en œuvre dans une situation concrète et de nous préparer à la rédaction et la présentation du futur rapport de stage.

Notre projet porte sur l'éclairage du kart électrique mis à notre disposition par l'IUT. Nous allons devoir effectuer plusieurs tâches afin d'optimiser son fonctionnement. Pour cela nous allons seulement intervenir sur la partie informatique du système en programmant l'ATmega8535 présent sur la carte qui nous est fournie. Nous devrons donc travailler sur les feux avants progressifs (en fonction de la lumière ambiante), les feux de stop progressifs (en fonction de la pression imposée sur la pédale de frein) ainsi que les clignotants et feux de détresse.

Ce rapport est le résultat de notre travail. Vous y trouverez, après une explication du cahier des charges, l'ensemble de nos recherches concernant la carte électronique du kart GEII électrique ainsi que la programmation de l'ATMega8535. Vous trouverez enfin notre programme visant à contrôler l'éclairage du kart, et toutes les informations relatives au déroulement du projet.

1. Cahier des charges

1.1. Objectifs

Notre objectif sera donc d'implanter un programme dans le microcontrôleur (un ATMega8535) de la carte électronique fournie pour obtenir l'éclairage souhaité. Les leds des clignotants devront clignoter quand on actionne les clignotants ou les feux de détresse. Les feux de stop arrières seront progressifs en fonction de la pression exercée sur la pédale et les feux avant seront progressifs en fonction de la luminosité ambiante. Pour cela, nous allons devoir faire une analyse adaptée des caractéristiques des lampes, de la pédale de frein et du capteur de lumière.

1.2. Contraintes

Notre projet est un travail d'amélioration. Cela signifie que nous devrons nous adapter à un environnement déjà existant. Nous devrons implanter notre programme dans la carte prévue pour être incorporée dans le Kart GEII. Les composants (dont le microcontrôleur : l'ATmega8535), le boîtier qui commande l'éclairage et différents capteurs (de luminosité ou de pression) sont donc imposés. La seconde contrainte sera de gérer notre temps car la réalisation du projet devra se faire dans un temps imparti de 8 semaines, ce qui est peu en comparaison au projet du semestre 3. Et enfin la dernière contrainte va être de pouvoir gérer l'allumage des lampes grâce à un signal MLI créé par le microcontrôleur, chose que nous n'avons encore jamais réalisée au cours de notre formation.

1.3. Déroulement

Nous allons donc devoir mettre en place des fonctions qui vont nous permettre de définir une suite logique dans la réalisation du projet :

✔ FP1 : Détecter la pression sur la pédale.

✔ FP2 : Détecter la luminosité ambiante à l'aide d'un Luxmètre.

✔ FP3 : Traiter les données passées par les capteurs ainsi que par le boîtier de commande.

✔ FP4 : Mettre en forme un signal de type MLI pour gérer l'allumage des feux.

1.4. Schéma fonctionnel

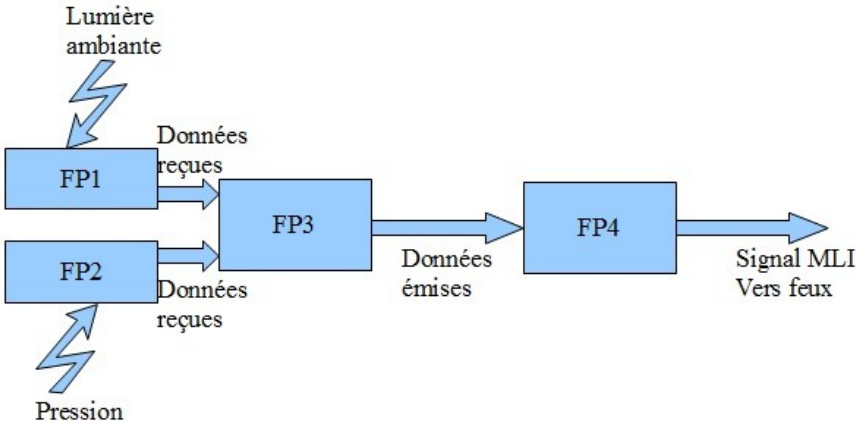


Illustration 1: Schéma fonctionnel des fonctions principales à réaliser[1]

2. Analyse du projet

La première étape de notre travail a été de faire des recherches afin d'étudier en détail les éléments fournis pour notre projet. Nous présenterons donc dans cette partie le fonctionnement de la carte électronique du kart ainsi que ses composants.

2.1. La carte électronique

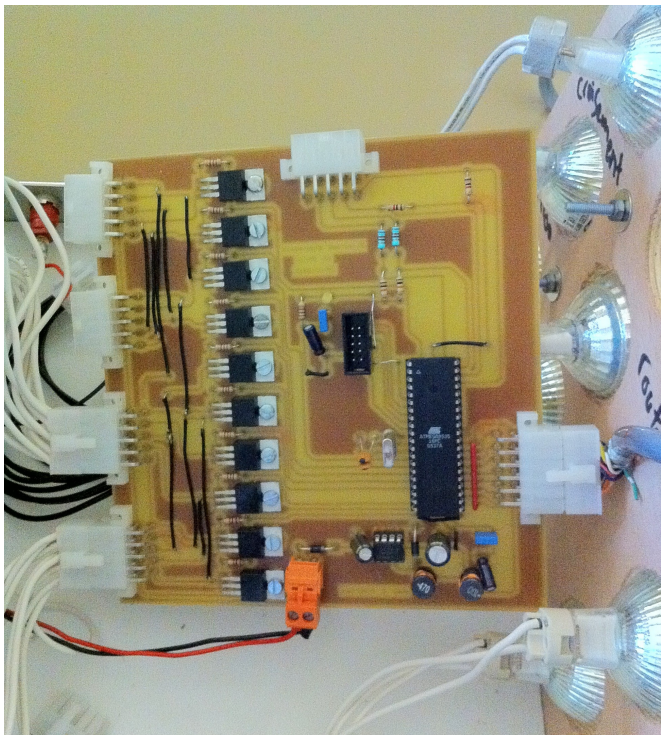


Illustration 2: La carte électronique du Kart électrique GEII[1]

La carte électronique du kart est composée principalement de :

➔ Un microcontrôleur : L'ATmega8535 (alimenté en 5V)

➔ Un circuit de régulation à découpage avec LM8574N qui crée une tension de 5V à partir d'une tension d'entrée comprise entre 7V et 40V

➔ Des transistors MOSFET de type L2203N (voir Illustration 3) , commandés par les sorties de l'ATmega8535, qui permettent en se fermant d'alimenter les lampes en 12V

➔ Des connecteurs pour relier les capteurs, le boîtier de commande et les sorties.

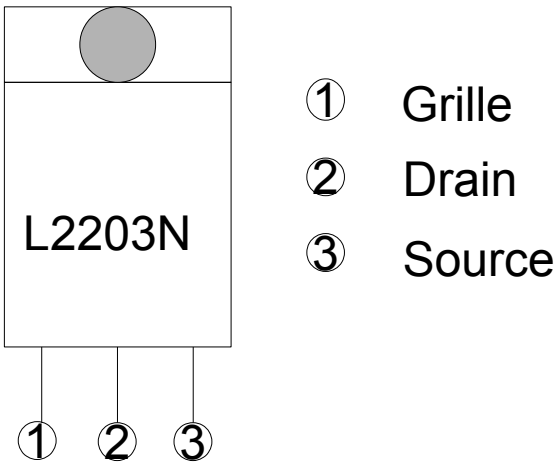


Illustration 3: Vue de face du transistor MOSFET L2203N[1]

Voici un schéma simplifié de la carte pour visualiser les précédents éléments, ainsi qu'une représentation des différents connecteurs présents sur la carte.

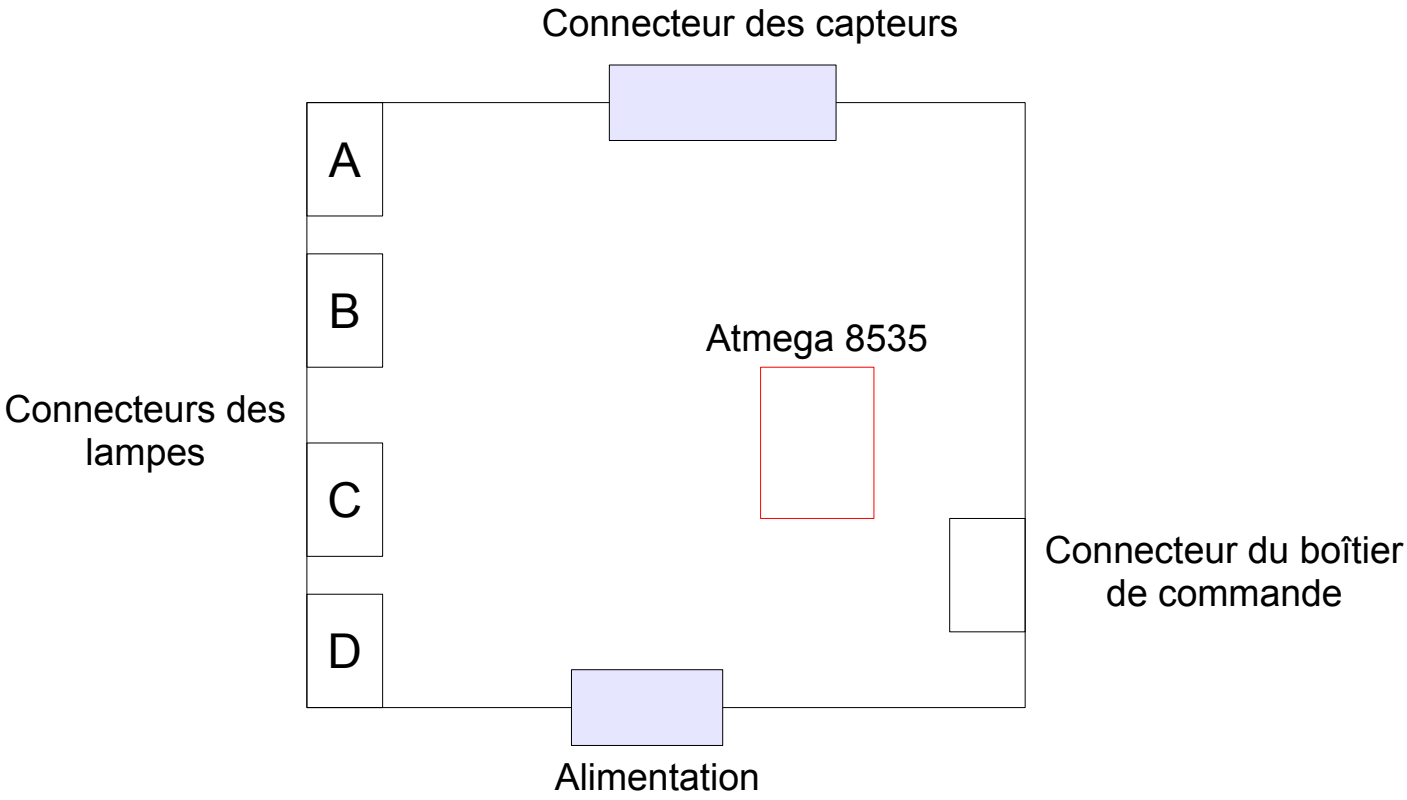


Illustration 4: Schéma simplifié de la carte électronique du kart[1]

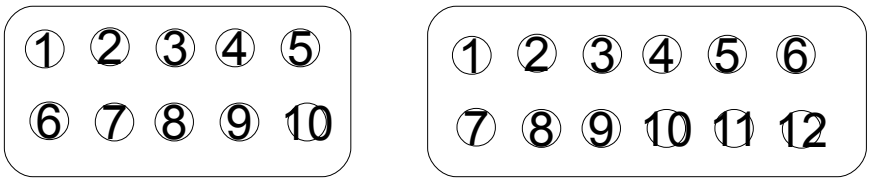


Illustration 5: Vue de faces des connecteurs des capteurs et des lampes (à gauche) et du boîtier de commande (à droite) [1]

2.2. Branchement des connecteurs

2.2.1. Connecteur des capteurs

Broches	Désignation des broches	Affectation sur les ports de l'ATmega
1	Capteur de frein	PA3
2	Luxmètre	PA2
3	Non utilisée	PA1
4	Non utilisée	PA0
5 + 8 à 10	Masse	-
6	Masse capteur de frein	-
7	Masse Luxmètre	-

2.2.2. Connecteur du boîtier de commande

Broches	Désignation des broches	Couleur des fils du connecteur	Couleur des fils dans le boitier	Affectation sur les ports de l'ATmega
1	Warning	Orange vif	Jaune	PC1
2	Clignotants droits	Orange terne	Marron	PC2
3	Clignotants gauches	Noir	Violet	PC3

4	Feux de route	Violet	Vert	PC4
5	Feux de code	Blanc	Bleu	PC5
6	ON / OFF	Jaune	Gris	PC6
7	Manu / Auto	Bleu	Blanc	PC0
8	Led témoin	Rouge	-	Sortie du régulateur
9 à 11	Masse	Noir	Noir	Masse
12	Non utilisée	Marron	-	PC7

2.3. Étude des lampes

Les lampes utilisées en guise de feux sont crées à partir de la technologie LED[1]. Pour faciliter la programmation en ce qui concerne les feux progressifs, on va considérer que l'éclairage des feux sera proportionnel à l'information envoyée par les capteurs (capteur de la pédale de frein et le luxmètre). Il nous faut donc étudier les caractéristiques des lampes pour essayer d'en dégager une zone linéaire.

Pour tester la linéarité des lampes, il va falloir reprendre le schéma de principe utilisé sur la carte, mais pour une seule lampe (voir schéma ci-dessous).

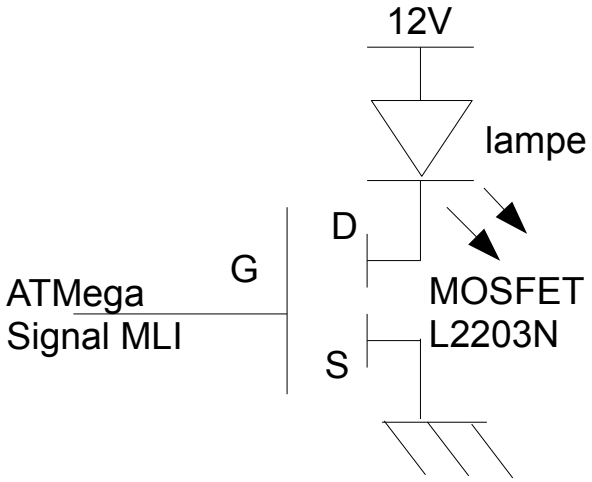


Illustration 6: Schéma de principe d'alimentation d'une lampe[1]

Selon la valeur de la MLI (voir 3.2), le courant traversant la lampe sera plus ou moins important, ce qui fera varier l'éclairage de cette dernière. On va donc devoir déterminer la courbe de

1 LED, en français DEL : Diode ÉlectroLuminescente

l'intensité lumineuse en fonction de la valeur de la MLI en espérant obtenir une droite afin de lui affecter une équation (voir illustration 9 et 10).

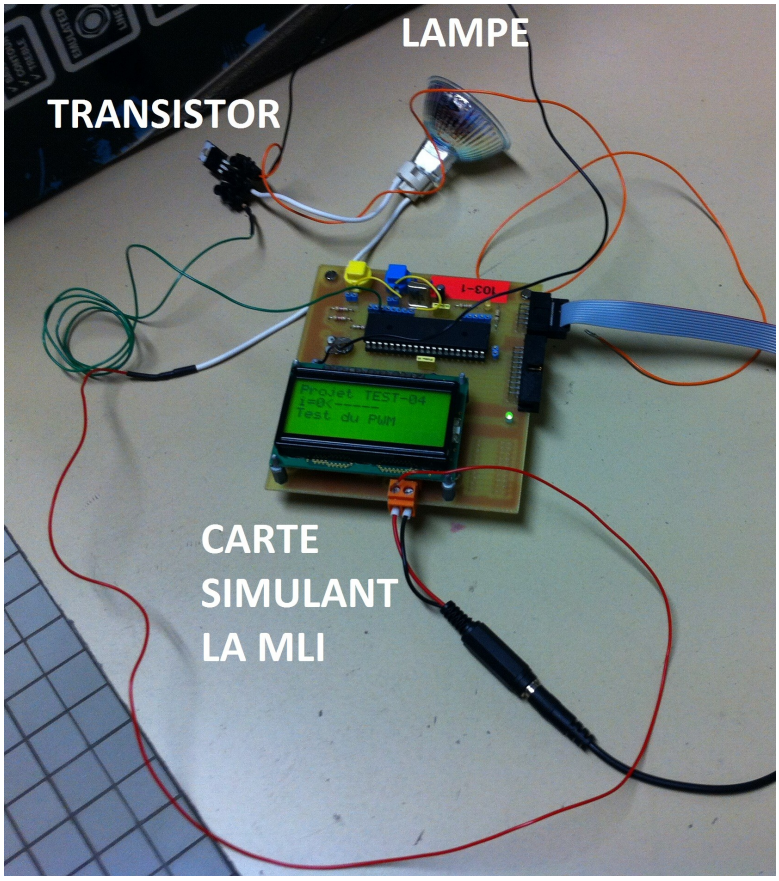


Illustration 7: Carte de test pour la caractérisation des lampes[1]

Pour procéder aux tests, on utilise une « boîte noire », une lampe à LED blanche et une rouge. On utilise une carte possédant une sortie analogique variable pour simuler la MLI et un luxmètre pour mesurer l'intensité lumineuse.

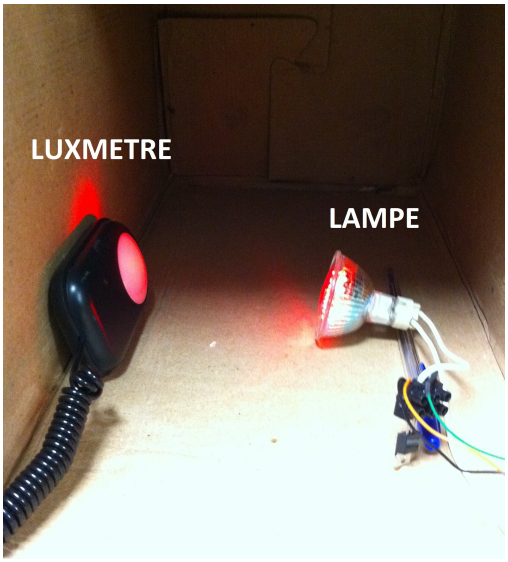


Illustration 8: Test de caractérisation des lampes à LED [1]

Caractéristiques des lampes rouges (pour les feux de stop) :

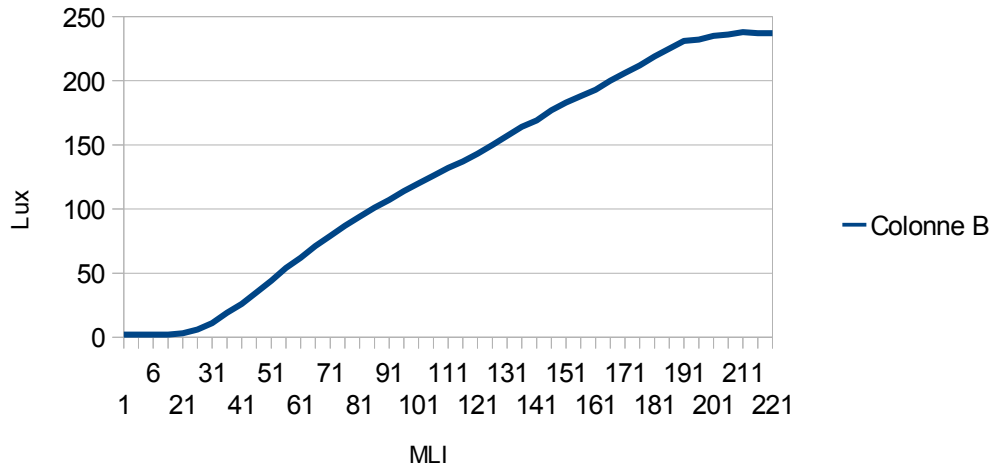


Illustration 9: Caractéristiques des lampes rouges[1]

Caractéristiques des lampes blanches (pour les feux de route) :

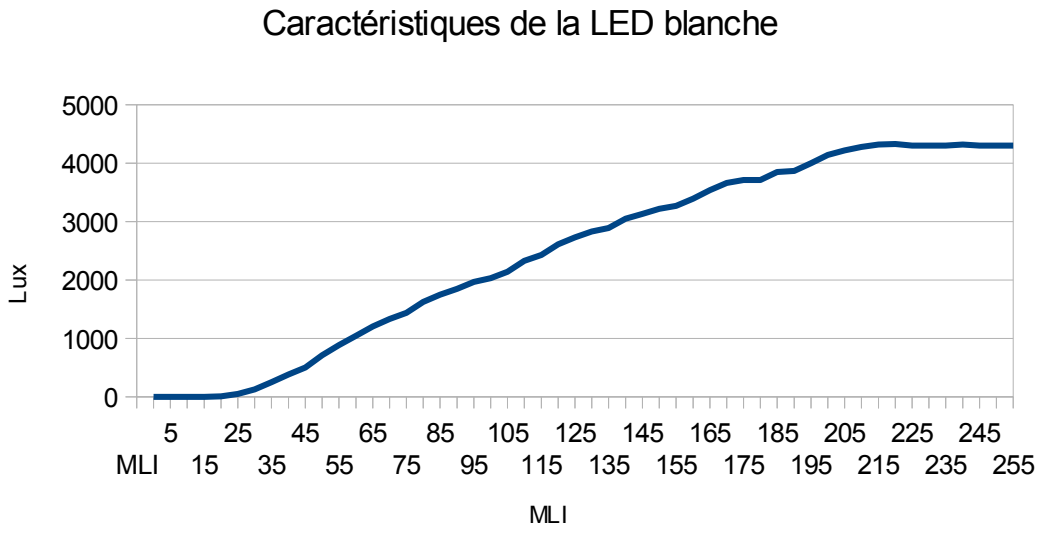


Illustration 10: Caractéristiques des lampes blanches[1]

Nous avons pu retenir de ces courbes la partie linéaire des caractéristiques des lampes pour les extremums de la MLI. Pour la lampe rouge, la MLI variera entre 25 et 185, valeurs qui correspondent aux intensités minimales et maximales des lampes. Quand à la lampe blanche, la MLI variera entre 25 et 200. Nous pouvons remarquer qu'il y a une grande différence entre les caractéristiques des deux lampes, la lampe à LED blanche peut éclairer beaucoup plus que la rouge (4300 Lux contre 237 Lux).

3. Réalisation du projet

Pour répondre au cahier des charges, nous allons programmer le microcontrôleur mis à disposition sur la carte électronique fournie. La gestion des feux de route et des feux de stop nécessitant une intensité variable, nous avons choisi d'utiliser la Modulation à Largeur d'Impulsion (MLI) pour retranscrire cela en langage informatique. Après s'être penché un peu plus sur l'ATMega8535, nous expliquerons comment créer de la MLI et enfin l'utilité et le fonctionnement de notre programme.

3.1. L'ATMega8535

L'ATmega8535 est un composant électronique programmable de type microcontrôleur. Il est composé de 4 ports parallèles de 8 broches chacun soit un total de 32 broches programmables. Le microcontrôleur possède de nombreuses fonctions comme un comparateur, un convertisseur, des horloges internes, des interruptions et d'autres fonctions.

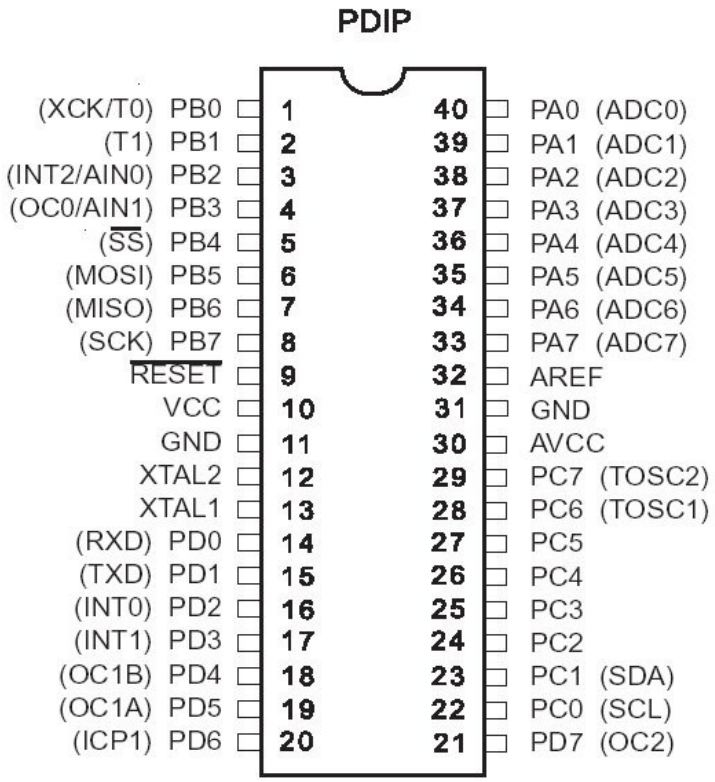


Illustration 11: Broches de l'ATMega8535[2]

Nous pouvons regrouper les affectations des broches de notre projet dans un tableau :

ENTREES	PORT A	PA2	Tension de sortie du luxmètre
		PA3	Tension de sortie du capteur de la pédale de frein
	PORT C	PC0	Manu / Auto
		PC1	Warning
		PC2	Clignotants gauches
		PC3	Clignotants droits
		PC4	Feux de route
		PC5	Feux de position
		PC6	ON / OFF
		PC7	-

SORTIES	PORT B	PB3	Feux de Stop arrières
	PORT D	PD0	Feux de recul
		PD1	Feux rouges de position
		PD2	Feux de croisement avants
		PD3	Clignotants droits
		PD4	Clignotants gauches
		PD5	-
		PD6	Feux de position avants
		PD7	Feux de route avants

3.2. La Modulation à Largeur d'Impulsion (MLI)

La modulation à largeur d'impulsion est le principe utilisé pour créer en sortie du microcontrôleur une tension variable, afin d'alimenter les lampes pour un allumage progressif. La MLI sera modélisée à l'aide du **Phase Correct PMW Mode** de l'ATMega8535 (voir documentation constructeur en annexe). Le principe est basé sur la comparaison de deux signaux. Un compteur, représenté par un signal en dent de scie de base (TCNTN sur le schéma ci-dessous) est comparé avec un échelon modifiable (OCRN sur le schéma ci-dessous).

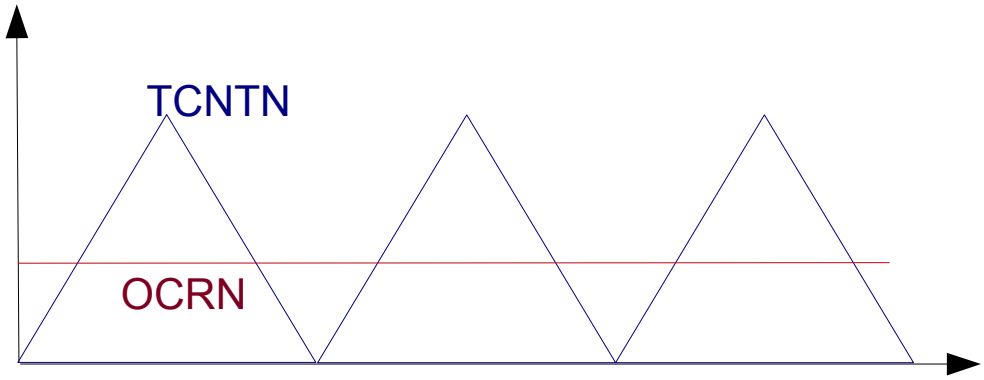


Illustration 12: Comparaison de la MLI[1]

Notre but va donc être de « jouer » sur la valeur du paramètre OCRN afin de modifier la sortie. En effet, à chaque fois que la valeur TCNTN atteint OCRN, que ce soit en phase ascendante ou descendante, la sortie OCN change d'état (passe à 1 si elle valait 0 et vice versa). En faisant varier OCRN, la caractéristique du signal de sortie va donc être modifiée : son rapport cyclique sera différent pour une valeur de OCRN grande ou petite. Un exemple illustré sera plus facile à comprendre :

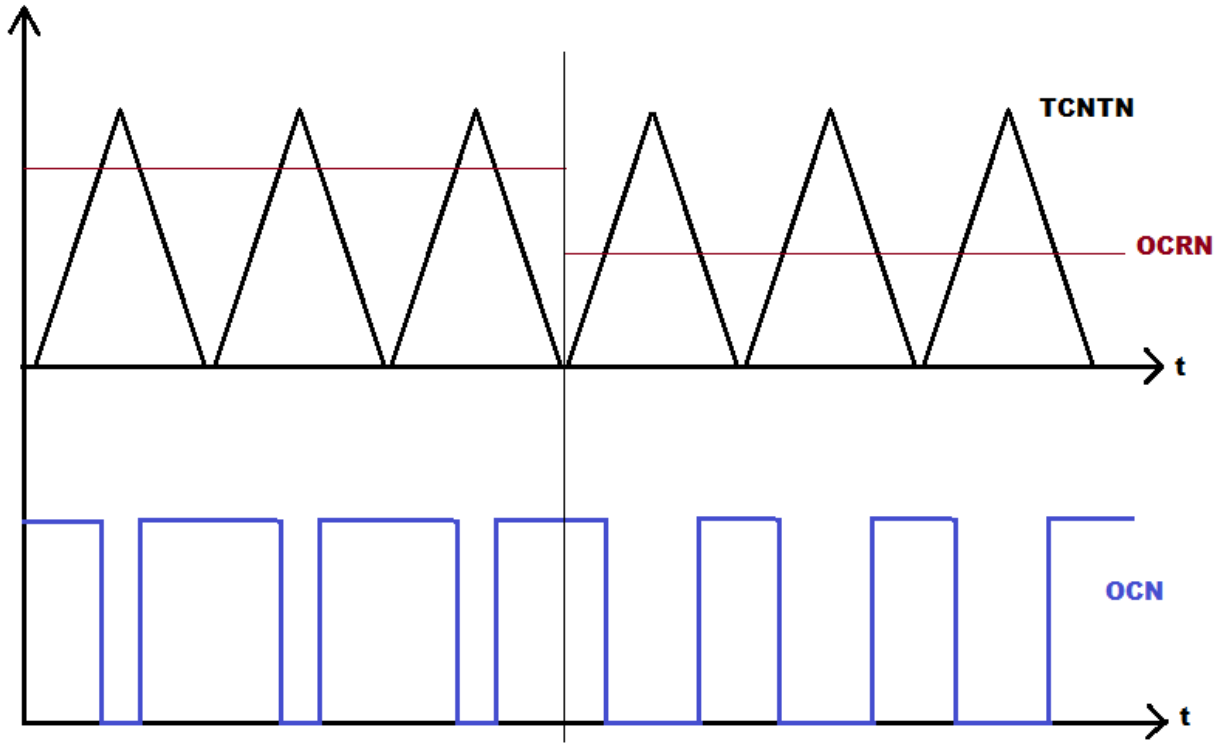


Illustration 13: Visualisation de la sortie MLI en fonction du paramètre OCRN[1]

Notre tâche va donc être de faire varier le paramètre OCRN en fonction de la tension d'entrée lue par l'ATMega8535 et envoyé par les capteurs de luminosité et de pression.

Si la tension de sortie du capteur est forte, le signal OCRN appliqué sera fort donc le rapport cyclique du signal de sortie OCN sera grand : les feux auront une forte intensité et brilleront d'avantage.

3.2.1. La MLI appliquée aux feux de stop

Pour obtenir la valeur de la tension envoyée par le capteur et indispensable à la MLI, il va falloir utiliser le convertisseur analogique/numérique de l'ATMega8535 pour convertir la tension de sortie du capteur de la pédale de frein en une donnée numérique (binaire). Ensuite, il nous faudra adapter la variation de cette donnée grâce à l'étude linéaire des lampes vue au 2.5.

La plage de tension de sortie du capteur de la pédale de frein est [0 ;4,22] V, donc à 0V, il n'y a pas d'éclairage et à 4,22V, l'éclairage des feux de stop est maximal. Le CAN[2] du microcontrôleur convertit des tensions de 0 à 5V en nombre de 0 à 511. La tension maximale renvoyée par la pédale de frein est de 4,22V, ce qui correspond à un valeur numérique de 431 (car 5V correspond à 511). Selon l'étude de la lampe rouge (voir 2.5), la MLI (OCRN) doit varier de 25 à 185 pour avoir un éclairage progressif linéaire. On peut déduire du schéma suivant l'équation donnant la valeur MLI à envoyer en fonction de la valeur reçue du capteur.

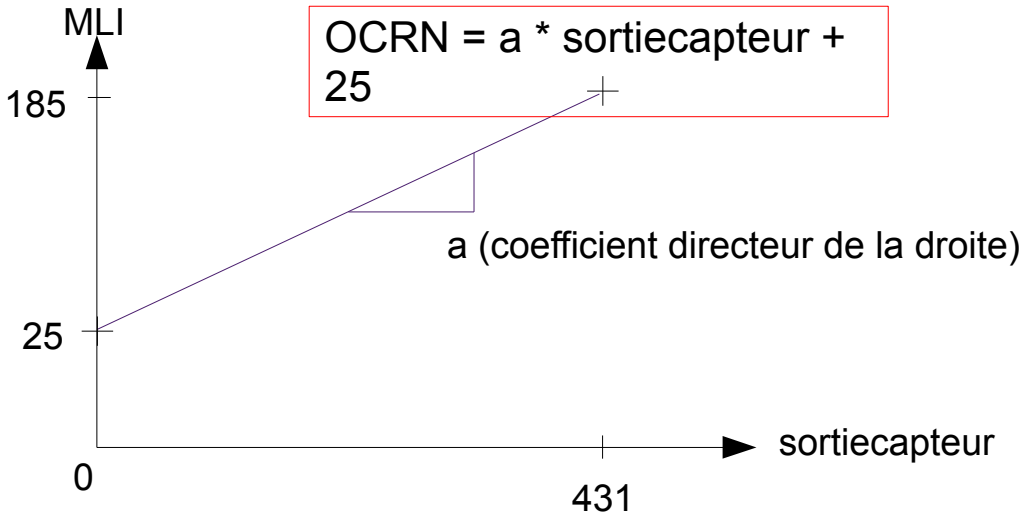


Illustration 14: Valeur de OCRN pour les feux de stop[1]

On calcule le coefficient directeur a :

$$a = \frac{185 - 25}{431} = 0{,}37$$

Donc on obtient au final :

$$OCRN = 0{,}37 \times Sortiecapteur + 25$$

2 CAN : Convertisseur analogique-numérique

3.2.2. La MLI appliquée aux feux de route

Pour les feux de route, la méthode est la même que pour les feux de stop. La plage de tension de sortie du luxmètre est [0 ; 5] V, donc à 0V, il n'y a pas d'éclairage et à 5V, l'éclairage des feux de stop est maximal. Le CAN du microcontrôleur convertit des tensions de 0 à 5V en nombre de 0 à 511. Le maximum d'éclairage étant atteint à 5V, la valeur de sortie du capteur après conversion (*sortielux* dans le schéma ci-dessous) varie donc de 0 à 511. Selon l'étude de la lampe blanche (voir 2.5), la MLI doit varier de 25 à 200 pour avoir un éclairage progressif linéaire. On peut déduire du schéma suivant la formule de la MLI, soit la tension à envoyer aux lampes des feux de route.

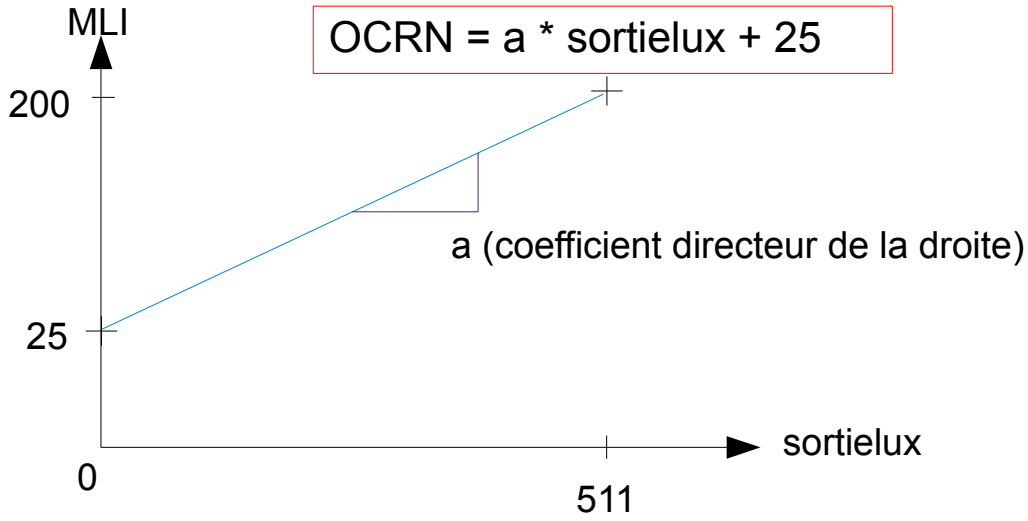


Illustration 15: Valeur de la MLI pour les feux de route[1] [1]

Donc on calcule la pente :

$$a = \frac{200 - 25}{511} = 0{,}34$$

Au final on obtient :

$$OCRN = 0{,}34 \times Sortielux + 25$$

Résumons comment passer d'un signal de capteur à une tension de contrôle d'un feux :

- Lire la tension de sortie du capteur
- Convertir cette tension analogique en numérique (entre 0 et 511) grâce au CAN du micro-contrôleur

– entre 0 et 431 pour le capteur de frein

– entre 0 et 511 pour le luxmètre

- Adapter la valeur de OCRN en fonction de la valeur renvoyée par le capteur.

– entre 25 et 185 pour les feux de stop

– entre 25 et 200 pour les feux de route

3.3. Programmation

Dans cette partie, nous allons étudier les sous fonctions dont nous avons parlé plus tôt dans le rapport. Ces sous fonctions réaliseront plusieurs tâches :

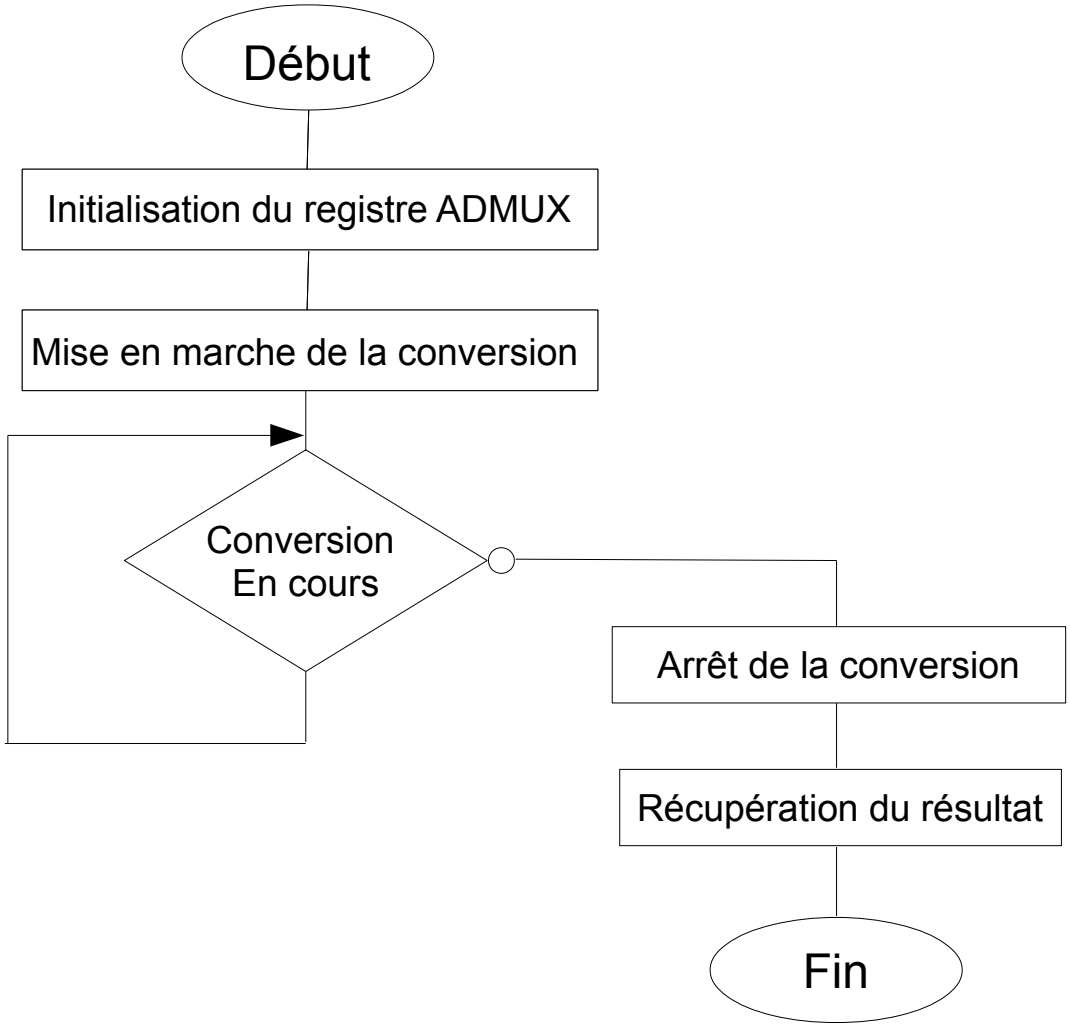
✔ La première aura pour but de faire la conversion analogique numérique qui nous sera d'une grande utilité pour le reste de la programmation.

✔La seconde sera l'étude de la fonction *main* qui sera le corps principal de notre programme.

✔ La troisième partie concernera l'allumage des différents feux en fonction du mode de fonctionnement demandée par le boîtier de commande.

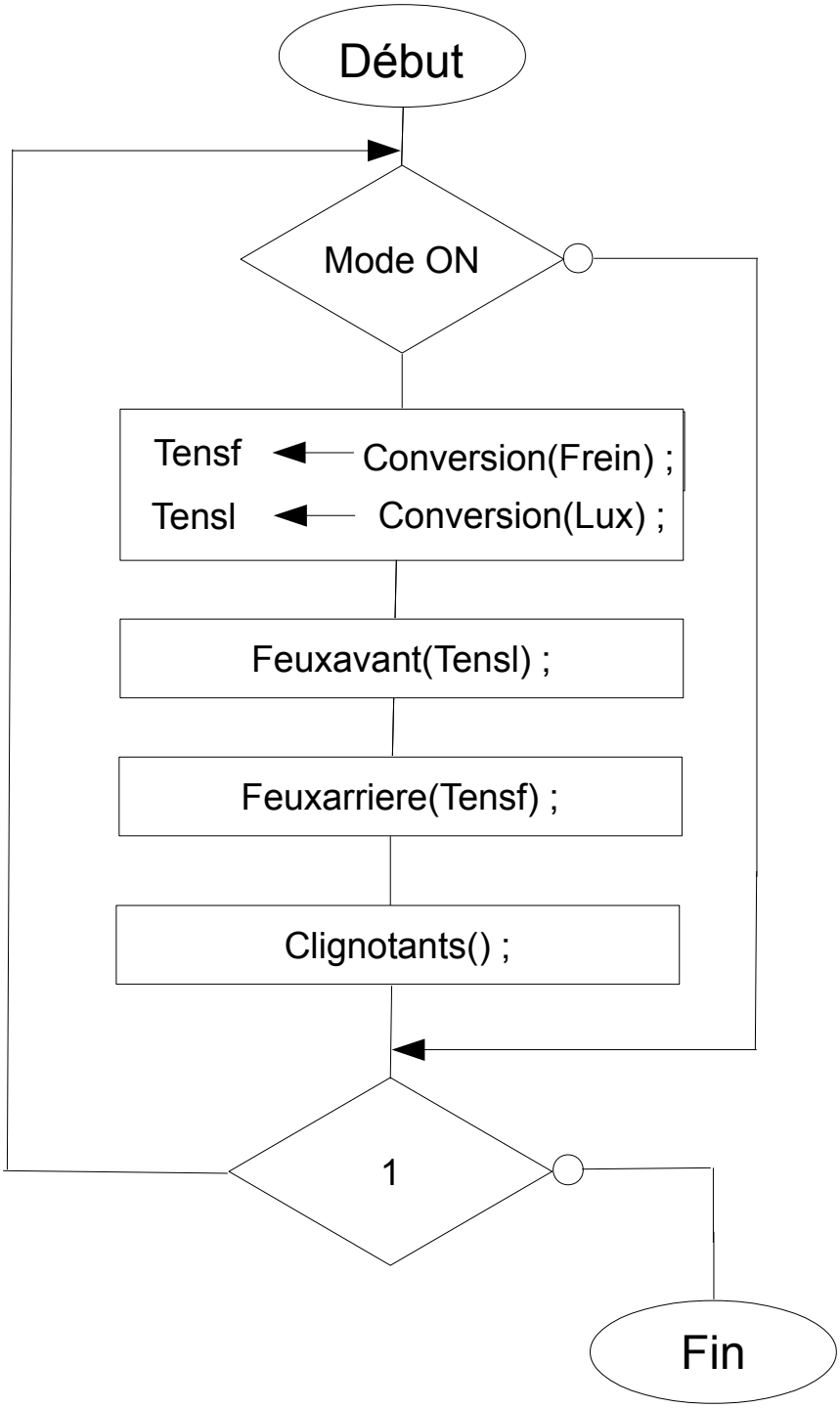
3.3.1. La conversion analogique-numérique

Cette fonction était complètement nouvelle pour nous car les registres utilisés nous étaient inconnus et donc leur étude nous a pris un peu de temps. Au final, elle marche très bien. Voici un organigramme de son fonctionnement :



Concernant le registre ADMUX, c'est un registre qui permet de définir la forme sous laquelle sera le résultat mais aussi d'indiquer sur quelle patte de l'ATMega8535 se situe la valeur à convertir. L'autre registre utilisé est le registre ADCSRA qui permet lui de s'occuper des paramètres de la conversion mais aussi de la démarrer, de savoir si la conversion est terminée, de l'arrêter et de récupérer le résultat.[1]

3.3.2. Le programme principal



Ce programme principal n'a pas grande utilité mais il est nécessaire pour organiser le programme. Il va effectuer les deux conversions et les stocker dans deux variables nommées *Tensf* et *Tensl* (respectivement pour Tension Frein et Tension Luxmètre). Une fois ces conversions faites il va faire appel aux fonctions *Feuxavant* , Feuxarriere et *Clignotants*, dont je vais décrire le fonctionnement juste après, en leur passant en argument les tensions numériques.

3.3.3. Les fonctions Feuxarriere, Feuxavant et Clignotants.

Ces trois fonctions vont permettre de gérer respectivement l'allumage des feux arrières, celui des feux avants et celui des clignotants/warnings qui se situent à la fois à l'avant mais aussi à l'arrière.

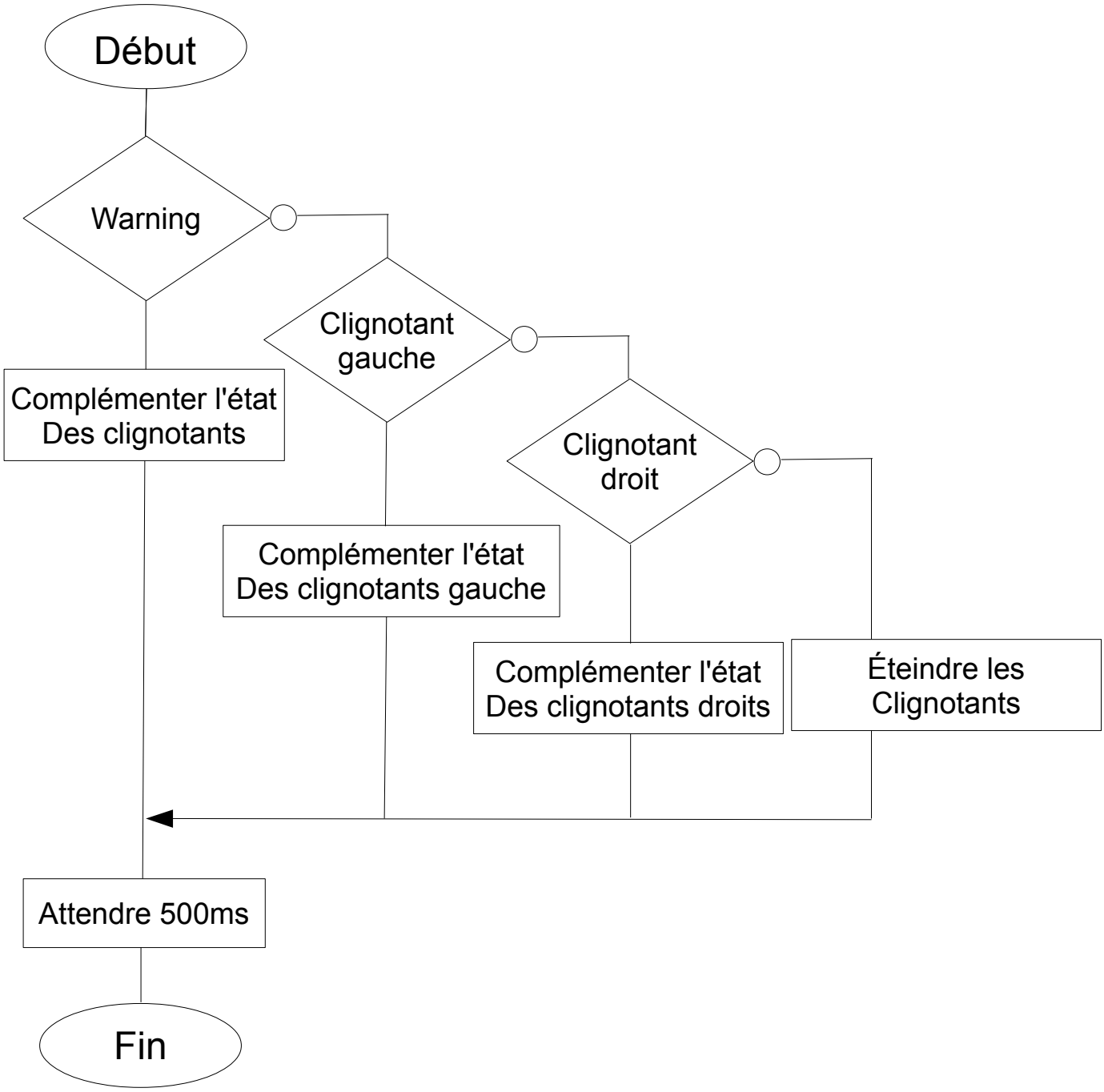
Concernant les feux clignotants, il y a trois boutons à prendre en compte : les warnings, le clignotant gauche ainsi que le clignotant droit. Le programme va d'abord traiter l'information concernant les warnings car il est inutile de traiter les autres informations sachant que les deux clignotants seront déjà allumés. En suite si le mode warning n'est pas activé, on va observer l'état de l'information clignotant gauche , puis si celui-ci n'est pas activé, l'état du clignotant droit. Enfin si aucune des trois entrées n'est active, nous allons éteindre tous les feux clignotants. Pour gérer le clignotement des lampes lorsque c'est nécessaire, nous allons utiliser une fonction d'attente que nous allons configurer à 500ms.

Ensuite, nous verrons la gestion des feux avants. C'est une fonction qui se verra passer en argument la valeur numérique correspondant à la tension envoyée par le Luxmètre. Tout d'abord, nous allons gérer l'allumage des feux de croisement et de position qui marchent indépendamment du mode de fonctionnement du kart (automatique ou manuel). Ensuite, nous nous occuperons des pleins phares. Ceux-ci auront deux fonctionnement différents : un progressif, en mode automatique, dont l'intensité lumineuse dépendra de la valeur numérique reçue et un fonctionnement TOR[3] en mode manuel : dés l'appui sur la commande « pleins phares », la lampe éclaire à pleine intensité, sinon elle n'éclaire pas. A noter que les feux de croisement s'allument automatiquement lorsque les pleins phares sont actifs même si l'information provenant du boîtier de commande indique le contraire.

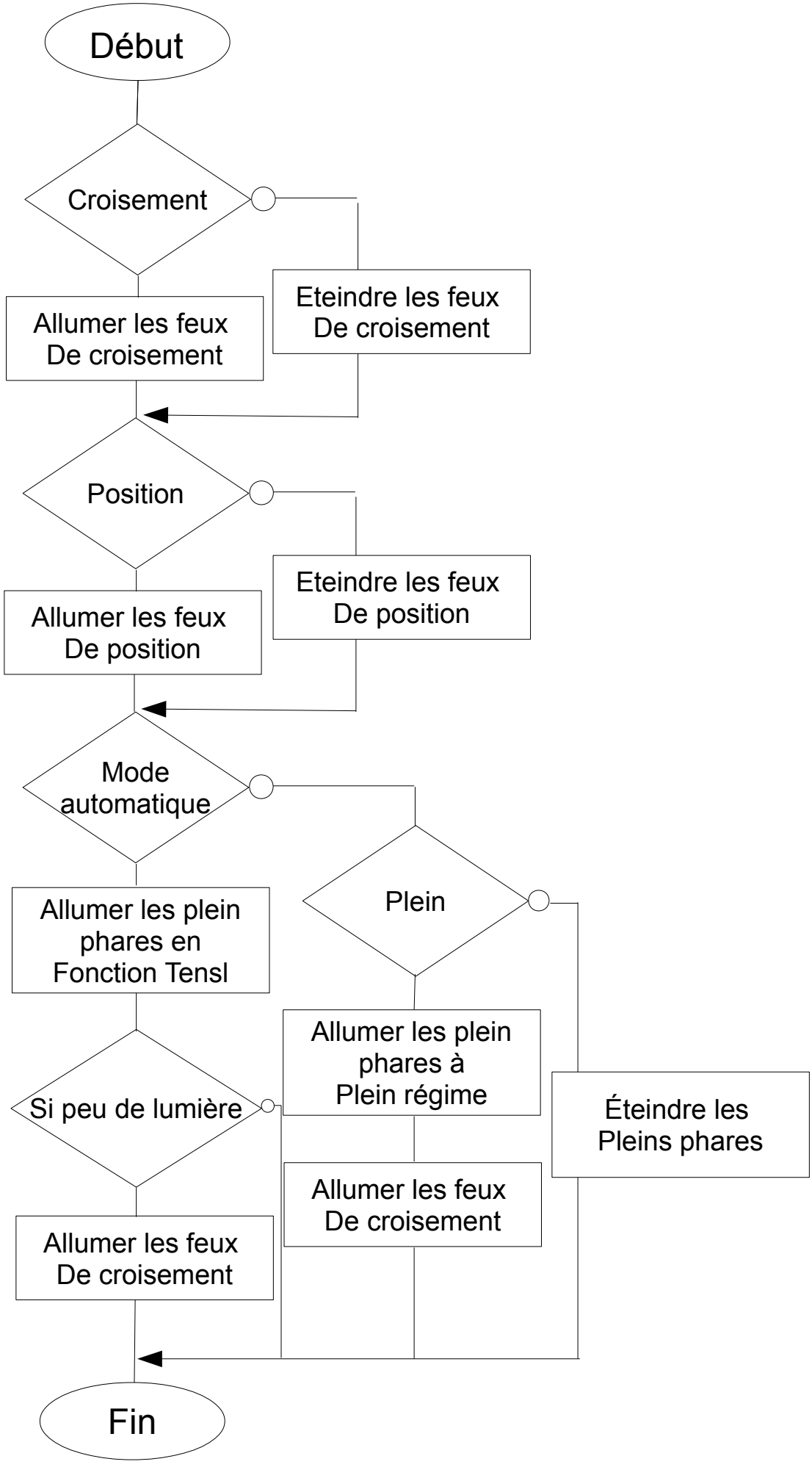
Et enfin, nous programmerons la fonction gérant les deux feux arrières : les feux témoins et les feux de stop. Les feux témoins seront allumés dés que l'un des trois feux situés à l'avant du kart le sera. Concernant les feux de stop, ils fonctionnent un peu de la même façon que les pleins phares : en mode automatique, ils éclaireront proportionnellement à la valeur numérique correspondant à la tension reçue de la pédale de frein alors qu'en mode manuel, des appui sur la pédale de frein la lampes éclaire de manière maximale et sinon est éteinte.

3 TOR : Tout Ou Rien

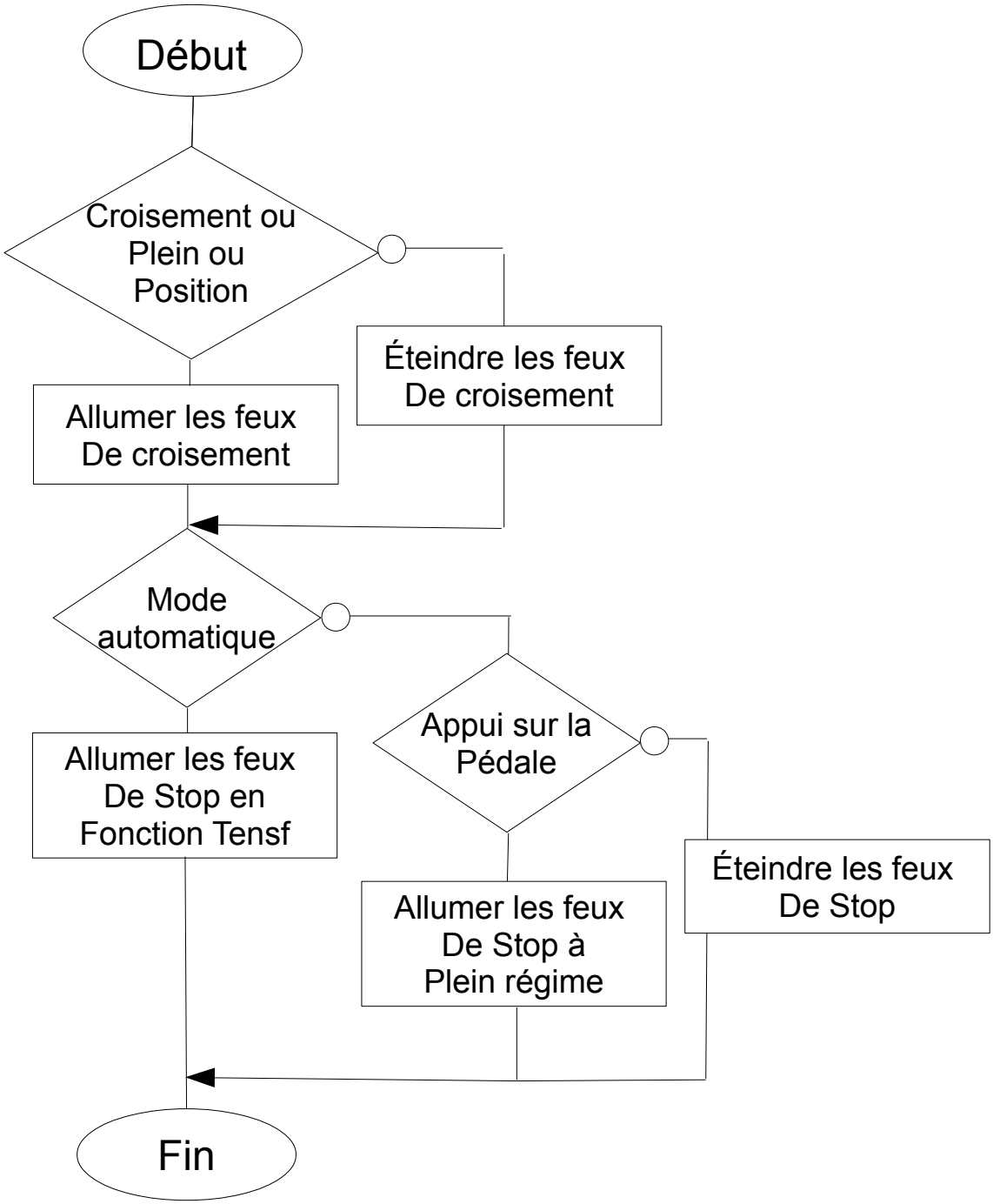
Voici d'abord l'organigramme de la fonction Clignotants.



Voici l'organigramme du programme gérant les feux avant :



Reste enfin la fonction gérant l'éclairage des feux arrières :



Concernant les feux arrières, les feux témoins s'allument lorsque les feux de croisement ou les pleins phares ou les feux de position sont allumés. Quant aux feux de stop, ils dépendent du mode de fonctionnement : ils s'allument progressivement en mode automatique et de manière optimale en mode manuel si jamais il y a appui sur la pédale de frein.

L'ensemble des organigrammes qui vous ont été présentés constituent la première partie de la programmation. La seconde est d'écrire le programme en langage C et de l'implanter dans la microcontrôleur. Vous pouvez retrouver ce programme dans les annexes.

3.4. Déroulement du projet

Dans le but de s'organiser, et ensuite d'observer notre progression tout au long du projet, nous avons établi un planning prévisionnel / réel (voir tableau ci-dessous).

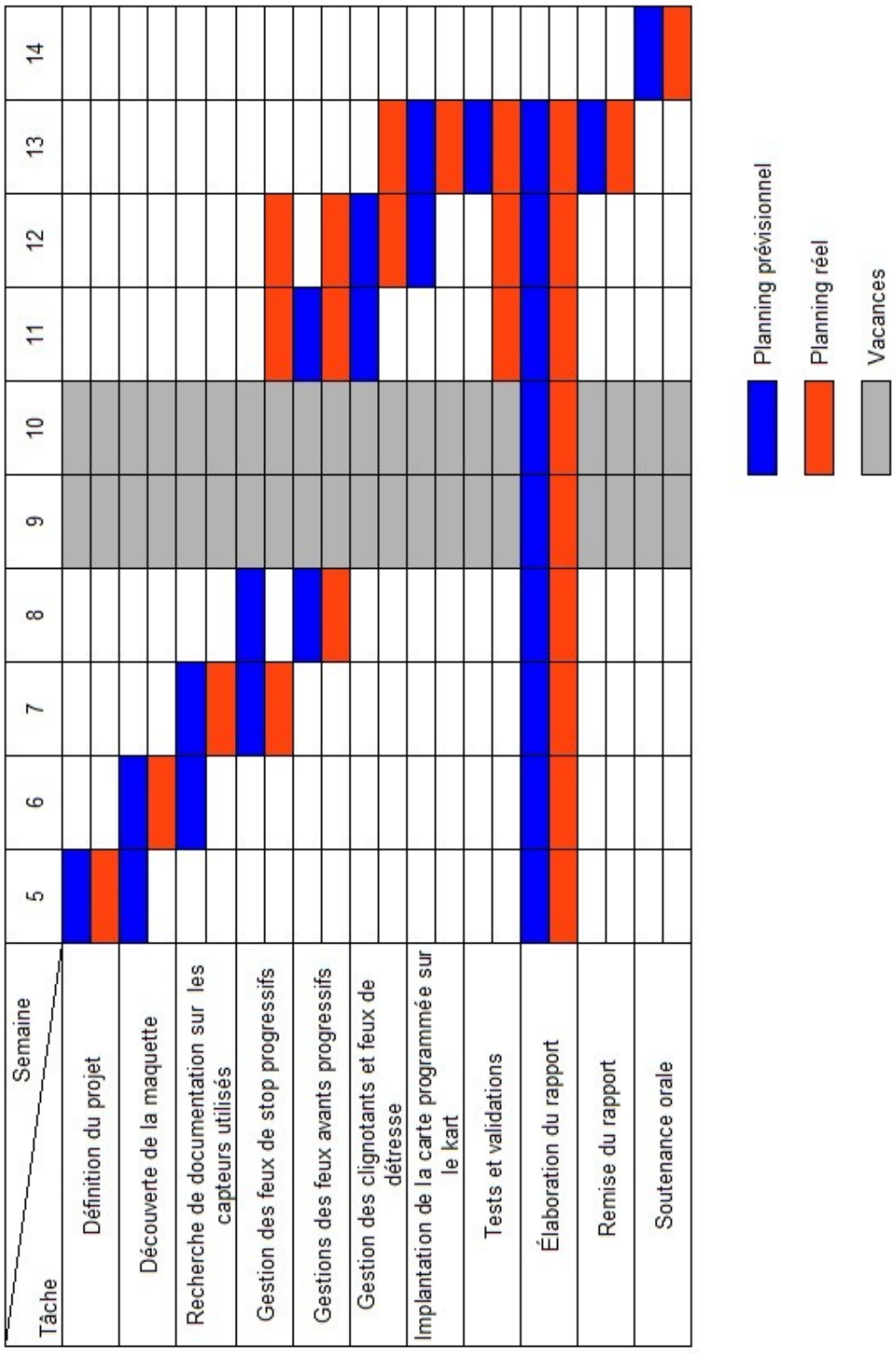


Illustration 16: Planning prévisionnel et réel[1]

Nous pouvons remarquer dans ce planning que nous avons relativement bien suivi le déroulement prévu, la partie programmation étant difficilement prévisible, puisqu'un problème technique peut vite survenir.

Conclusion

Lors de ce projet, nous avons dû effectuer un travail de recherche sur les différents éléments qui nous ont été mis à notre disposition.

En fonction de ces éléments, nous avons dû mobiliser nos connaissances et élaborer une solution répondant au mieux au cahier des charges, dans le temps imparti au projet.

À l'issue de ce travail, nous sommes capables de faire l'analyse d'un système déjà conçu et de l'améliorer, mais aussi d'un composant et de l'adapter à une situation précise, de faire face à une panne et de la résoudre, de s'organiser dans un projet et de retransmettre nos idées dans un rapport.

Ce projet s'est montré captivant, d'une part parce que le dispositif et le sujet sont intéressant, et d'autre part parce que le temps imparti était réduit, ce qui nous a donné un bon aperçu de la gestion de projet en entreprise. On peut conclure que ce projet a été instructif tout en étant ludique, et que cela a été un bon entraînement pour le stage de fin d'année d'IUT.

Résumé

Ce projet a été découpé en plusieurs tâches. La première a été de faire l'étude des éléments mis à notre disposition tels que la carte électronique, les différents capteurs et les lampes utilisées. Nous nous sommes rendus compte que les fonctionnement des lampes à LED n'était pas linéaire, il nous a donc fallu établir de nouvelles équations pour linéariser leur fonctionnement et ainsi obtenir un éclairage optimal. Une fois ceci terminé, nous avons du nous pencher sur la conversion analogique numérique intégrée à l'ATMEga8535, car c'est une chose qui nous était totalement inconnue. A cette étape, nous étions capables de gérer l'éclairage des pleins phares et feux stop en fonction des valeurs de tensions envoyées par les deux capteurs. La plus grosse partie du projet était terminée ! Il ne nous restait plus qu'à gérer l'éclairage des feux « normaux », c'est à dire des feux ne dépendant pas de capteurs et donc fonctionnant de manière Tout ou Rien. Au final nous obtenons donc le fonctionnement souhaité puisque en mode manuel, chaque lampe éclaire ou non en fonction du boîtier de commande fourni alors qu'en mode automatique, les feux stop et les plein phares s'éclairent progressivement. Nous considérons donc ce projet achevé et pleinement réussi.

Nombre de mots : 214 mots

Table des illustrations

Bibliographie

1: Les auteurs : SOUFFEZ, Thomas et ESCARIEUX, Guillaume, Rapport projet tuteuré S4, 2012
3: ATMEL CORPORATION, ATMEGA 8535L, 10/06
4: Frank SAURET, ATMega 8535. Français. , 24/06/05
2: TUILARD, Benjamin et MEUKENS, Pierre, Modification du kart électrique, 2011

ANNEXE 1 : Programme complet[1]

```
#include <mega8535.h>
#include<stdio.h>
#include<delay.h>

// Declaration des variables globales

#define Mode PINC.0
#define Warning PINC.1
#define CD PINC.2
#define CG PINC.3
#define Plein PINC.4
#define Croisement PINC.5
#define ON_OFF PINC.6
#define Position PINC.7



#define Feux_Temoin PORTD.1
#define Feux_Croisement PORTD.2
#define Feux_CD PORTD.3
#define Feux_CG PORTD.4
#define Feux_Position PORTD.6
```

```
#define Feux_Plein PORTD.7
#define Feux_Stop PORTB.3

#define ADC_VREF_TYPE 0x20

void Feuxavants(long T);
void Feuxarrieres(long T);
void Clignotants(void);

unsigned int Lecture_Tension(unsigned char adc_input) //Fonction conversion analogique numérique
{
ADMUX=adc_input|ADC_VREF_TYPE; //Justification à gauche du résultat et définition de l'entrée à convertir
ADCSRA|=0x40; //Début de la conversion
while ((ADCSRA & 0x10)==0); //Attente de la fin de conversion
ADCSRA|=0x10;//Arrêt de la conversion
return ADCH; //Récupération du résultat
}

void main(void)
{
// Declare your local variables here
   int Tensfrein, Tenslux; //Tension en sortie du capteur de la pédale de frein
   //unsigned int Tenslux; // Tension en sortie du capteur luxmètre

// Input/Output Ports initialization
// Port A initialization
```

```
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTA=0x00;
DDRA=0x00;

// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=Out Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;
DDRB=0x08;

// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
PORTC=0x00;
DDRC=0x00;

// Port D initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out Func0=Out
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
PORTD=0x00;
DDRD =0xFF;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 16 MHz
// Mode: Générateur MLI
```

```
TCCR0=0x61;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: 16 MHz
// Mode: Générateur MLI

ASSR=0x00; //horloge du timer 2 interne
TCCR2=0x61;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
MCUCR=0x00;
MCUCSR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x10;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
```

```
ACSR=0x80;
SFIOR=0x00;

#asm("sei")

// ADC initialization
// ADC Clock frequency: 125,000 kHz
// ADC Voltage Reference: AREF pin
// ADC High Speed Mode: Off
// ADC Auto Trigger Source: None
ADMUX=ADC_VREF_TYPE;
ADCSRA=0x87;
SFIOR&=0xEF;



while (1)
{
        if(ON_OFF==0) //Interrupteur sur ON
        {

        Tensfrein=Lecture_Tension(4); //Conversion de la tension issue de la pédale
        Tenslux=Lecture_Tension(2); //Conversion de la tension issue du Luxmètre
        Feuxavants(Tenslux); //Appel de la fonction Feuxavants
                Feuxarrieres(Tensfrein); //Appel de la fonction Feuxarriere
                Clignotants(); //Appel de la fonction Clignotants
      }
        else //Interrupteur sur OFF ->On étein tout.
        {
```

```
                Feux_Temoin=0;
                Feux_Croisement=0;
                Feux_CD=0;
                Feux_CG=0;
                Feux_Position=0;
                OCR0=0;
                OCR2=0;
        }

}
}
void Feuxavants(long T)
{
        //Allumage des feux de croisement en mode manuel
        if(Croisement==0 & Mode==1)
        {
                Feux_Croisement=1;
        }
        //Arret des feux de croisement
        else if(Croisement==1 & Mode==1)
        {
                Feux_Croisement=0;
        }
        //Allumage des feux de position
        if(Position==0)
        {
                Feux_Position=1;
        }
```

```
//Arret des feux de position
else
{
	Feux_Position=0;
}

//Mode Automatique
if(Mode==0)
{

	//Allumage des feux stop en fonction de la pédale de frein
	OCR2=T;

	//Allumage des feux de croisement si peu de lumière ou si Croisement=1
	if(T>120|Croisement==0)
	{
		Feux_Croisement=1;
		Feux_Temoin=1;

	}
	else
	{
		Feux_Croisement=0;
		Feux_Temoin=0;

	}
}
//Mode Manuel
```

```
        else
        {
                //Allumage des pleins phares manuel
                if(Plein==0)
                {
                        OCR2=255;
                        Feux_Croisement=1;
                }
                else
                {
                        OCR2=0;
                }
        }
}

void Feuxarrieres(long T)
{
        //Mode automatique
        if(Mode==0)
        {
                OCR0=T; //Allumage des feux stop en fonction de la tension du Luxmètre
        }
        //Mode manuel
        else
        {
                if(T>30) //Si appui sur la pédale
                {
                        OCR0=255; //Allumage à fond des feux stop
```

```
                }
                else
                {
                        OCR0=0;
                }
        }
        if(Mode==1&(Plein==0|Croisement==0|Position==0))
        {
                Feux_Temoin=1;
        }
        else if(Mode!=0)
        {
                Feux_Temoin=0;
        }
}

void Clignotants(void)
{
        if(Warning==0)//Mode warning
        {
                Feux_CD=!Feux_CD;
                Feux_CG=!Feux_CG;
        }
        else if(CD==0)//Mode clignotant droit
        {
                Feux_CD=!Feux_CD;
                Feux_CG=0;
        }
```

```
        else if(CG==0)//Mode clignotant gauche
        {
                Feux_CG=!Feux_CG;
                Feux_CD=0;
        }
        else
        {
                Feux_CG=0;
                Feux_CD=0;
        }
        delay_ms(500); //Attente de 500ms
}
```

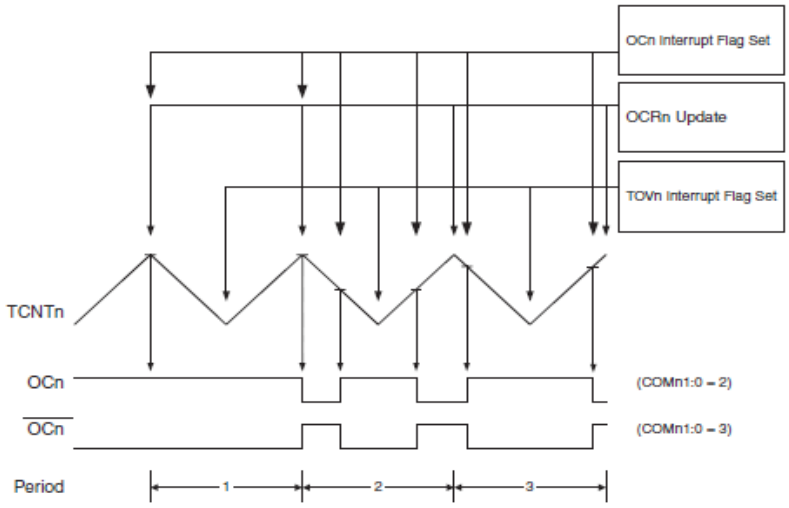
ANNEXE 2 : Phase Correct PMW Mode[2]

Phase Correct PWM Mode

The phase correct PWM mode (WGM01:0 = 1) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is based on a dual-slope operation. The counter counts repeatedly from BOTTOM to MAX and then from MAX to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC0) is cleared on the Compare Match between TCNT0 and OCR0 while up-counting, and set on the Compare Match while down-counting. In inverting Output Compare mode, the operation is inverted. The dual-slope operation has lower maximum operation frequency than single slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

The PWM resolution for the phase correct PWM mode is fixed to eight bits. In phase correct PWM mode the counter is incremented until the counter value matches MAX. When the counter reaches MAX, it changes the count direction. The TCNT0 value will be equal to MAX for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 33. The TCNT0 value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT0 slopes represent compare matches between OCR0 and TCNT0.

Figure 33. Phase Correct PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV0) is set each time the counter reaches BOTTOM. The interrupt flag can be used to generate an interrupt each time the counter reaches the BOTTOM value.

In phase correct PWM mode, the compare unit allows generation of PWM waveforms on the OC0 pin. Setting the COM01:0 bits to two will produce a non-inverted PWM. An inverted PWM output can be generated by setting the COM01:0 to three (See Table 42 on page 84). The actual OC0 value will only be visible on the port pin if the data direction for the port pin is set as output. The PWM waveform is generated by clearing (or setting) the OC0 Register at the Compare Match between OCR0 and TCNT0 when the counter increments, and setting (or clearing) the OC0 Register at Compare Match between

OCR0 and TCNT0 when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

$$f_{OCnPCPWM} = \frac{f_{clk\_I/O}}{N \cdot 510}$$

The "*N*" variable represents the prescale factor (1, 8, 64, 256, or 1024).

The extreme values for the OCR0 Register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR0 is set equal to BOTTOM, the output will be continuously low and if set equal to MAX the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values.

AT the very start of period 2 in Figure 33 OCn has a transition from high to low even though there is no Compare Match. The point of this transition is to guarantee symmetry around BOTTOM. There are two cases that give a transition without Compare Match.

- OCR0 changes its value from MAX, like in Figure 33. When the OCR0 value is MAX the OCn pin value is the same as the result of a down-counting Compare Match. To ensure symmetry around BOTTOM the OCn value at MAX must correspond to the result of an up-counting Compare Match.
- The timer starts counting from a value higher than the one in OCR0, and for that reason misses the Compare Match and hence the OCn change that would have happened on the way up.

ANNEXE 3 : Programmation des registres pour la conversion CAN[3]

Le CAN de l'ATMega 8535.

1. Caractéristiques :

- Résolution : 10 bits,
- 8 voies multiplexées,
- 7 voies différentielles,
- 2 voies différentielles avec gain ajustable 10x à 200x (Non garantit pour les boîtiers PDIP et PLCC, seulement pour les boîtiers TQFP et MLF),
- Ajustement optionnel à gauche pour le résultat sur ADC,
- Non linéarité : inférieur à 1/2 LSB,
- Erreur absolue de ± 2 LSB,
- Erreur à zéro volt : 1 LSB,
- Temps de conversion : réglable de 65 à 260 µs (plus le temps est long, plus le résultat et précis), soit 15384 échantillons/secondes,
- Tension de référence externe ou interne (conversion de 0 à VCC),
- Acquisitions simples ou continues,
- Démarrage auto sur entrée d'interruption.

2. Fonction de transfert :

La fonction de transfert pour une conversion unique est :

$$ADC = \frac{Vin}{Vref} \times 1024 \quad \text{et inversement : Tension mesurée} = \frac{ADC \times Vref}{1024}$$

Pour une conversion différentielle c'est :

$$ADC = \frac{(VPos-VNeg) \times Gain \times 512}{Vref} \quad \text{Et inversement : Tension mesurée} = \frac{ADC \times Vref}{Gain \times 512}$$

Le résultat est signé (complément à 2) de -512 à +511. Le signe est dans ADC9 (1 = - et 0 = +).

3. Atténuation du bruit de conversion:

Afin de réduire au maximum les erreurs de conversions dues à la logique interne du contrôleur, plusieurs solutions peuvent êtres mis en oeuvre afin d'éviter ce désagrément :

- mettre en sommeil l'unité centrale avant le lancement d'une conversion
- découpler soigneusement l'alimentation en respectant le schéma présenté ci-après
- respecter les règles élémentaires du routage de circuit imprimé en analogiques (connexions courtes, plan de masse...)
- dans tout les cas, effectuer toujours un traitement numérique des résultats (amortis,moyenne...)

4. Les registres :

Ils sont aux nombres de 5 :

4.1. ADCL et ADCH : Registres de résultats de la conversion.

	15	14	13	12	11	10	9	8	
ADCH							ADC9	ADC8	ADLAR = 0
ADCL	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	
	7	6	5	4	3	2	1	0	

	15	14	13	12	11	10	9	8	
ADCH	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADLAR = 1
ADCL	ADC1	ADC0							
	7	6	5	4	3	2	1	0	

ADMUX : Sélection de la voie de conversion (sur le port A, configuration des pins en entrées sans résistance de rappel)

Adresse	7	6	5	4	3	2	1	0
$07	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

REFS1	REFS0	Tension de référence sélectionnée
0	0	ARef. Vref interne est off
0	1	AVCC avec un condo externe sur AREF
1	0	//
1	1	2,56 V interne avec un condo externe sur AREF

ADLAR : Change la présentation du résultat. 1 : Justification à gauche ; 0 : Justification à droite. Le changement est immédiat.

MUX4..0	Entrée mono	Entrée diff. positive	Entrée diff. négative	Gain
00000	ADC0			
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4	N/A		
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000		ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010		ADC0	ADC0	200x
01011		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110		ADC2	ADC2	200x
01111		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010	N/A	ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
11101		ADC5	ADC2	1x
11110	1.22V (V_{BG})	N/A		
11111	0V (GND)			

4.3.

ADCSRA : Registre de contrôle et statut A.

Adresse	7	6	5	4	3	2	1	0
$06	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

- **ADEN** : Validation du CAN.
- **ADSC** : Lancement de la conversion sélectionnée (retourne à 0 en fin de conversion).
- **ADATE** : Déclenchement auto de la conversion (1) par la source sélectionnée par ADTS du SFIOR.
- **ADIF** : passe à 1 une fois la conversion terminée et déclenche l'interruption si ADIE =1 et le bit I du SREG = 1.
- **ADIE** : Validation de l'interruption **ADC** , déclenché lors du passage à 1 de ADIF.
- **ADSP2...ADSP0** : Sélection du facteur de pré division de l'horloge interne du convertisseur :

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

4.4. SFIOR : Registre de fonction spéciale.

Adresse	7	6	5	4	3	2	1	0
$30	ADTS2	ADTS1	ADTS0	-	ACME	PUD	PSR2	PSR10

ADTS2..0 : Ne fonctionnent que si ADATE (ADCSRA) = 1.

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Libre
0	0	1	Comparateur Analogique
0	1	0	Interruption externe 0. Permet la conversion continue.
0	1	1	Timer/Counter0 Comparaison OK
1	0	0	Timer/Counter0 Dépassement
1	0	1	Timer/Counter1 Comparaison OK B
1	1	0	Timer/Counter1 Dépassement
1	1	1	Timer/Counter1 Capture d'évenement

5. En résumé.

On peut choisir : La tension de référence, la fréquence de fonctionnement, la voie (ou les voies) à convertir et le déclencheur de la conversion.

Exemple (conversion rapide sur 10 bit avec Vref externe) :

1. Avant de lancer la conversion : ADMUX = 0x00, ADCSRA = 0x80, SFIOR = 0x00.
2. Lancement de la conversion : ADSC=1 (ADCSRA = ADCSRA ou 0x40).
3. Attendre que ADIF=1 (ADCSRA= ADCSRA ou 0x10).
4. Lire le résultat dans ADC (ADCH – ADCL).
5. Eteindre le CAN ADEN=0 (ADCSRA=ADCSRA & !0x80)