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

RAPPORT DE PROJET 2ÈME ANNÉE

GESTION D'ALLUMAGE DES PHARES DU KART ÉLECTRIQUE

Sébastien GUIBOUT
Kéram OUZOUNIAN
Groupe K4B

Enseignants :
LEQUEU Thierry
AUGER Philippe

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

RAPPORT DE PROJET 2ÈME ANNÉE

GESTION D'ALLUMAGE DES PHARES DU KART ÉLECTRIQUE

Sébastien GUIBOUT
Kéram OUZOUNIAN
Groupe K4B

Enseignants :
LEQUEU Thierry
AUGER Philippe

Sommaire

Introduction

Dans le cadre du semestre 4, notre projet d'études et réalisation est de programmer l'allumage des feux du kart électrique. Ce projet est proposé par M. LEQUEU.

Le support de notre projet est le micro-contrôleur ATMega8535. C'est ce composant qui gérera la commande des différents phares. La carte électronique sur laquelle est monté le micro-contrôleur a été réalisée au semestre 3 par Cheikh Abba DIEME.

Lors de ce semestre, nous allons nous concentrer sur la partie programmation. Nous avons cinq modes de feux à prévoir : les feux de code, les pleins feux, les clignotants, les feux de détresse et les feux de recul. Pour programmer, nous utiliserons le logiciel CodeVisionAVR.

Dans un premier temps, nous élaborerons un cahier des charges. Ensuite nous aborderons la gestion des feux. Nous présenterons le micro-contrôleur ATMega8535, le logiciel CodeVisionAVR et la configuration nécessaire; puis nous commenterons le code, qui sera en langage C, et les ordinogrammes correspondant. Pour finir cette deuxième partie, nous parlerons des tests effectués sur le programme et des éventuelles erreurs.

Ensuite, nous validerons la carte électronique réalisée précédemment. Si celle-ci fonctionne correctement, nous ferons part de nos observations afin de l'améliorer.

Enfin, la dernière partie sera consacrée à l'intégration de la carte et du boîtier de commande. Au départ, le boîtier de commande était simplement une boîte comportant cinq interrupteurs. Nous remplacerons ce boîtier par un commodo de voiture.

1. Cahier des charges

Ce projet consiste à réaliser le contrôle des feux de signalisation avant et arrière pour un kart électrique. Il s'agit d'allumer progressivement les feux stop en fonction de la pédale de frein et de pouvoir allumer les feux avant, graduellement, en tenant compte de la luminosité ambiante. Enfin, il faudra paramétrer les clignotants et les feux de détresse.

Dans un premier temps, nous nous occuperons de la partie informatique du projet. Nous programmerons les feux selon plusieurs modes : feux de code (éclairage faible, permet d'indiquer sa position), pleins feux (éclairage fort pour voir la route, revenir en feux de code pour ne pas éblouir les autres conducteurs), clignotants avant et arrière, feux de détresse (tous les clignotants sont actifs) et feux de recul.

Ensuite, nous réaliserons un boîtier de commande fonctionnel qui sera installé directement sur le kart. Il est composé de trois interrupteurs à deux positions : un premier interrupteur pour la commande marche/arrêt du boîtier, un deuxième pour les feux de détresse et le dernier pour le passage en mode manuel ou automatique. Deux autres interrupteurs, à trois positions, permettront de gérer les clignotants et d'allumer ou d'éteindre les pleins feux ou les feux de codes. Nous essaierons également de remplacer le boîtier par un commodo de voiture.

Finalement, nous reprendrons la carte électronique réalisée au semestre précédent, permettant de contrôler les feux du kart. Nous vérifierons son bon fonctionnement et, si besoin est, elle sera modifiée puis imprimée à nouveau.

Si la carte est à rééditer, nous en profiterons pour tenter d'ajouter des LED's sur le boîtier de commande, l'objectif étant d'indiquer l'état de chaque phare.

Ce projet présente deux contraintes matérielles. Nous devons utiliser le micro-contrôleur ATMega 8535. Le programme gérant les feux sera rédigé sur ordinateur, puis implanté dans ce composant qui fonctionnera ainsi en autonomie. L'autre contrainte est de reprendre la carte électronique. Nous devons vérifier ce projet, valider son fonctionnement. Si la carte convient, nous proposerons quelques modifications.

Planning prévisionnel										
	37	38	39	40	41	42	43	44	45	46
Prise de connaissance du sujet	P R									
Elaboration du cahier des charges et du planning	P R									
Formation Orcad		P R								
Programmation et test	P R	P R	P R	P R	P R	 R				
Réalisation du boîtier de commande						P R				
Validation de la carte électronique				P R	P	P				
Test final du projet								P		
Intégration au kart								P R	P R	
Rédaction du rapport	P R	P R	P R	P R	P R	P R	P R	P R		
Remise du rapport								P R		
Soutenance orale									P R	
	P	Prévisionnel				R	Réel			

Illustration 1: planning prévisionnel et réel

2. Gestion de l'éclairage

Dans cette partie, nous allons présenter les éléments de programmation. L'élément physique, le composant où sera implanté le programme final, est un ATMega 8535. Le logiciel de programmation est CodeVisionAVR. Nous exposerons son interface et la configuration nécessaire à ce projet. Ensuite, les ordinogrammes et le code seront expliqués pas à pas, ainsi que le programme final. Enfin, nous montrerons et expliciterons les phases de tests.

2.1. Présentation de l'ATMega

Nous utilisons un micro-contrôleur Atmega8535, imposé par M. LEQUEU. Ce composant convient parfaitement à notre projet car il possède un nombre d'entrées/sorties satisfaisant. La carte est alimentée avec une tension continue de 12V et est directement connectée sur la batterie du kart électrique ; l'ATMega est alimenté en 5V continu à partir du 12V grâce à une alimentation à découpage de type Buck.

Le micro-contrôleur est composé de quatre ports d'interface parallèles, chacun comprenant huit entrées ou sorties, ainsi que d'un port de programmation.

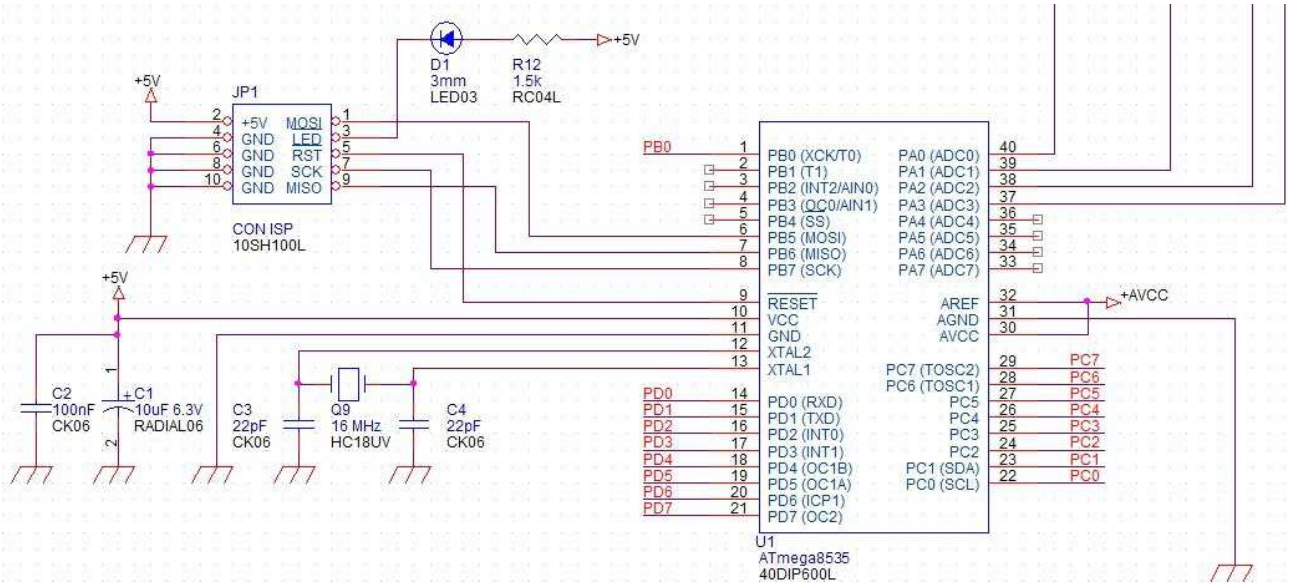

Illustration 2: schéma de l'ATMega

A droite du schéma se situe l'ATMega, avec ses quatre ports de A à D et l'alimentation (pins 30 à 32). Le port de programmation est en haut à gauche, il est connecté aux pins 6 à 9, à l'alimentation 5V et à la LED D1 qui s'allume pendant le transfert du programme de l'ordinateur vers le composant. Le montage à quartz se trouve en bas à gauche et est relié aux pins 10 à 13. Il fournit une base de temps au micro-contrôleur.

Pour l'instant, les entrées numériques – c'est-à-dire les clignotants, les feux de code et pleins phares, les warnings, le fonctionnement automatique ou manuel et la pédale de frein – sont gérées à partir d'un boîtier possédant cinq interrupteurs à deux ou trois positions. Elles sont reliées au port C.

Comme sur le schéma ci-dessous :

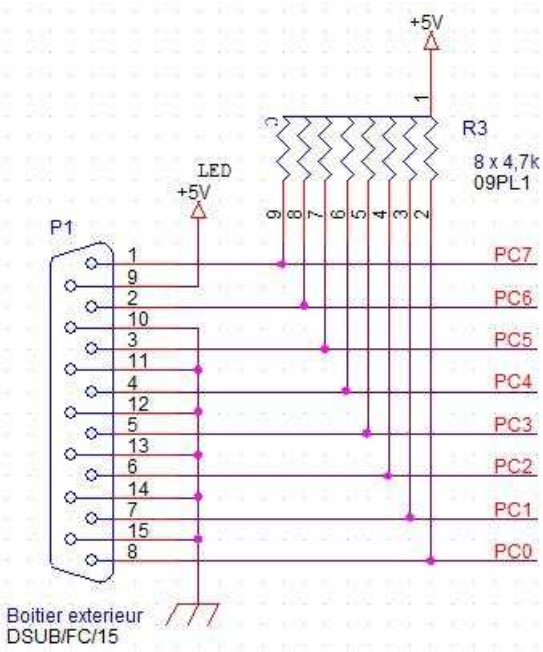

Illustration 3: entrées de l'ATMega

Cependant pour améliorer l'intégration au kart, nous l'équiperons d'un commodo qui est plus simple à manipuler.

Les entrées analogiques – c'est-à-dire la détection de la marche avant et arrière, le potentiomètre de la pédale de frein et la photodiode – sont connectées sur le port A.

Voici le tableau répertoriant les entrées de l'ATMega8535 :

Port d'entrée	Numéro	Mnémonique	Explications
Port A	0	Amavant	Détection de la marche avant
	1	Amarrière	Détection de la marche arrière
	2	Aphdiode	Capteur de luminosité, allume les phares en fonction de la lumière extérieure
	3	Apedale	Allume progressivement les feux stop en fonction de l'appui sur la pédale de frein.
Port C	0	Eauto	Switch entre le fonctionnement automatique et manuel
	1	Ewarn	Gestion des warnings
	2	EclignoD	Clignotants droits
	3	EclignoG	Clignotants gauches
	4	Ecode	Allume les feux de code
	5	Epp	Allume les pleins phares
	6	Efrein	Allume progressivement les feux stop en fonction de l'appui sur la pédale de frein.

Illustration 4: tableau récapitulatif des entrées

Nous utilisons huit sorties du composant. Ces sorties commandent des transistors MOSFET de référence IRL2203NPBF. Lorsqu'une sortie est à 1, la Grille du transistor reçoit cette impulsion à travers une résistance, le transistor devient ainsi passant. C'est par ce moyen que s'allument les phares du kart.

Voici le tableau des sorties de l'ATMega :

Port d'entrée	Numéro	Mnémonique	Explications	Position sur le kart
Port D	0	SclignoD	Clignotants droits	Avant et arrière
	1	SclignoG	Clignotants gauches	Avant et arrière
	2	Srecul	Feux de recul	A l'arrière
	3	ScodeAV	Feux de code	A l'avant
	4	Sblanc1	Cadre de LED blanches	A l'avant
	5	ScodeAR	Feux de code	A l'arrière
	6	Svert	Feux de code, cadre plein vert	A l'avant
	7	Sblanc2	Pleins feux	A l'avant

Illustration 5: tableau récapitulatif des sorties

Exceptées les sorties Sblanc1 et Svert, toutes les autres sont reliées à deux lampes à LED de type GU5,3 / 12V / 1W – la sortie ScodeAR est reliée à 4 lampes. Les clignotants sont de couleur orange, les feux stop et de code arrière sont rouges. Tous les autres sont blancs.

2.2. Présentation du logiciel de programmation

Le logiciel de programmation que nous utilisons est CodeVisionAVR. Il s'agit d'un environnement de développement intégré, c'est un compilateur en langage C. Il est largement utilisé par les entreprises et les universités pour la programmation de micro-contrôleur, comme l'ATMega 8535. Dans cette partie, nous présenterons la configuration du logiciel.

2.2.1. Interface

L'interface de CodeVisionAVR est l'image ci-dessous :

Illustration 6: CodeVisionAVR, principaux outils

Le cercle noir désigne le navigateur, il permet de naviguer entre les différents fichiers du projet. Le cercle rouge regroupe les quatre fonctions de base d'un logiciel : créer un nouveau fichier ou projet, ouvrir un fichier existant, enregistrer et imprimer. Le cercle jaune réunit cinq autres importantes possibilités : annuler, restaurer, couper, copier et coller.

Les cercles vert, bleu et violet sont les plus intéressants. Le premier icône[1] est la

1 D'après l'édition 2007 du Robert, l'icône informatique est du genre masculin.

compilation. Elle permet de compiler le code et de vérifier les éventuelles erreurs et « warning [2]». L'icône désigné en bleu compile dans un premier temps, puis implante le programme dans le composant. Enfin, le cercle violet, l'engrenage, ouvre une fenêtre de paramétrage et génère automatiquement un code en fonction des paramètres. Ce code est ensuite à compléter.

2.2.2. Configuration

Dans cette partie, nous allons aborder la configuration du logiciel CodeVisionAVR afin de pouvoir programmer correctement le micro-contrôleur. Toutes les étapes suivantes sont importantes et doivent être exécutées rigoureusement.

Tout d'abord, il faut créer un nouveau projet, en cliquant sur l'icône « Create new file » à droite de la barre cerclée de rouge dans la partie précédente. La fenêtre suivante apparaît :

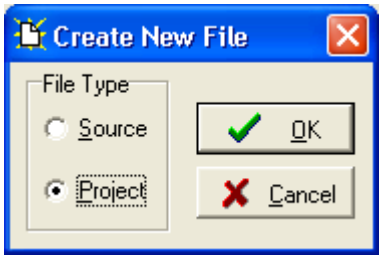

Illustration 7: création d'un nouveau projet

Après un appui sur la touche « OK » pour valider, le programme ouvre une fenêtre de confirmation. Il faut également valider en cliquant sur « Yes ».

Ensuite, la fenêtre de paramétrage du composant s'affiche – on peut également l'obtenir en cliquant sur l'engrenage vu dans la partie 2.2.1 Interface. Il faut remplir les champs comme suit :

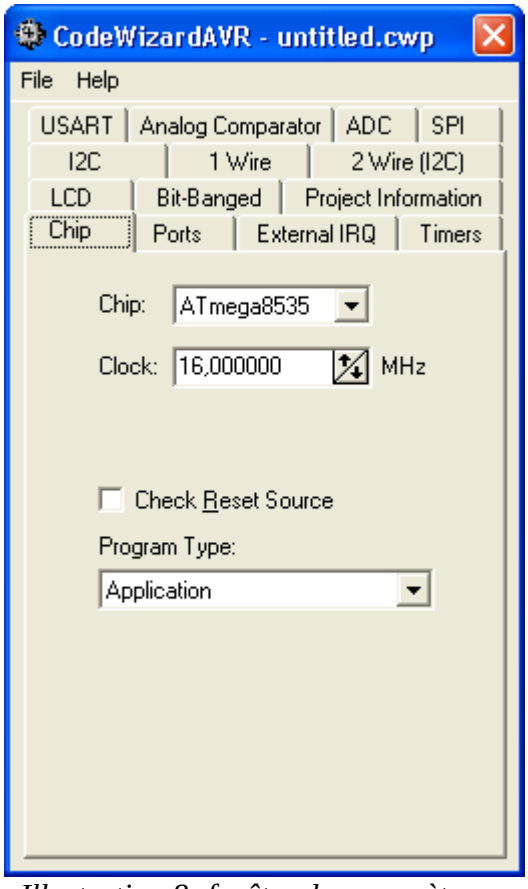

Illustration 8: fenêtre de paramètrage

2 Il s'agit d'une erreur n'empêchant pas l'exécution du programme.

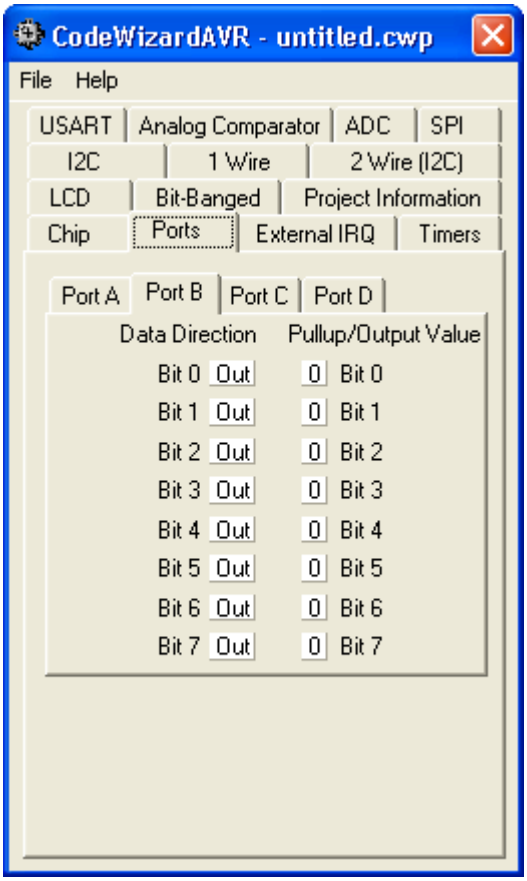

Illustration 9: programmation des ports d'entrées / sorties

Puis en passant dans l'onglet « Ports », on peut programmer les quatre ports A, B, C et D selon qu'ils sont en entrée ou en sortie. Par défaut, tous les ports sont en entrée, tous les bits sont alors indiqués en « In » pour les « Data Direction » et les « Pullup/Output Value » sont à 1. Il suffit de cliquer sur « In » pour que les bits passent en « Out » et à 0.

Ensuite, nous avons programmé les timers dans l'onglet correspondant. Ces compteurs vont servir à gérer les clignotants. La fréquence d'oscillation est de 15,625 kHz. Le « Comp. A » est la valeur jusqu'à laquelle le timer compte, puis revient à zéro. Ce paramètre est réglé en validant « Compare A Match ». La valeur $(1e84)_h$ vaut 7812 soit la moitié de la fréquence du timer. Ainsi, nous obtenons une période de clignotement d'une seconde : une demi-seconde à l'état haut (le clignotant est allumé) et l'autre demi-seconde à l'état bas (le clignotant est éteint).

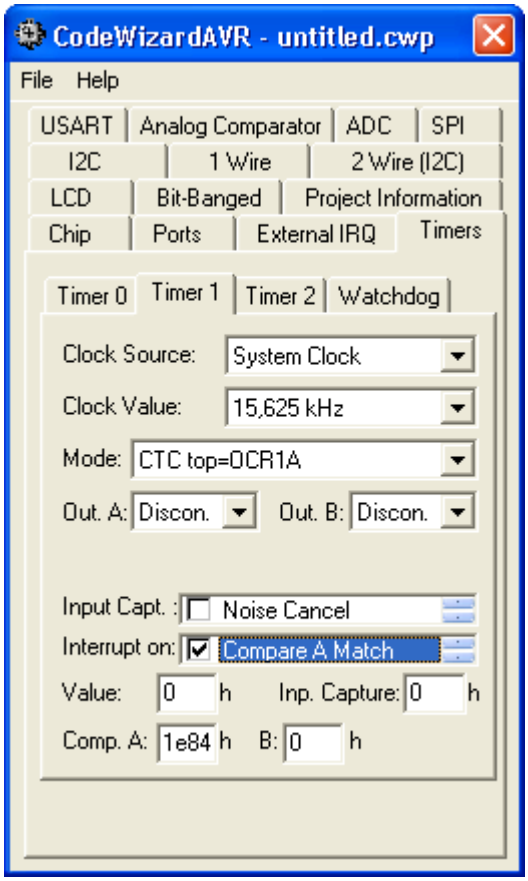

Illustration 10: programmation des timers

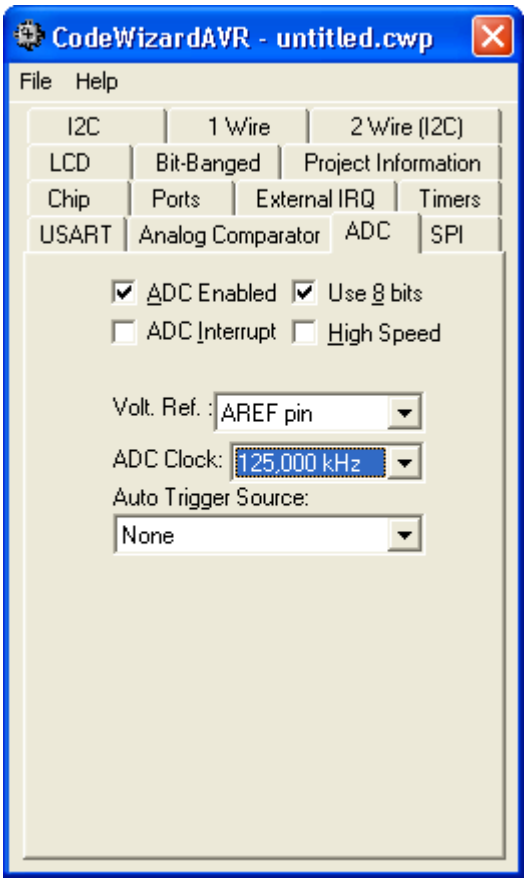

Illustration 11: paramétrage des entrées analogiques

Dans l'onglet ADC, nous indiquons à l'ATMega qu'il y a des entrées analogiques à gérer. Il faut alors cocher les cases « ADC Enabled » pour autoriser les entrées analogiques et « Use 8 bits », dans le but de préciser que la valeur entrante sera codée sur un octet. Si cette case n'est pas validée, la valeur sera codée sur dix bits.

Le logiciel est maintenant bien paramétré. Pour retrouver cette configuration dans le programme, afin qu'elle prenne effet, le logiciel génère du code qui est ensuite à compléter.

C'est la commande « Generate » qui donne ce code dans une nouvelle fenêtre. Le programme demande la sauvegarde du logiciel. Enfin la fenêtre de configuration est fermée.

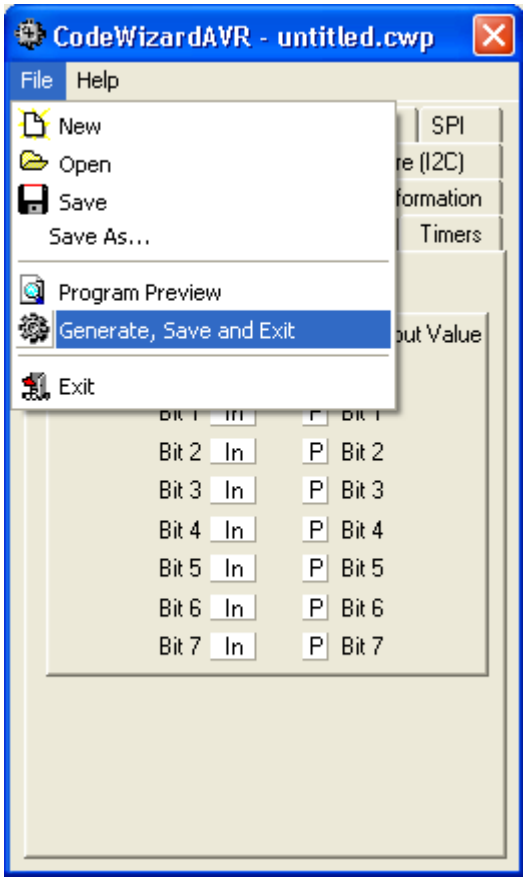

Illustration 12: génération du code

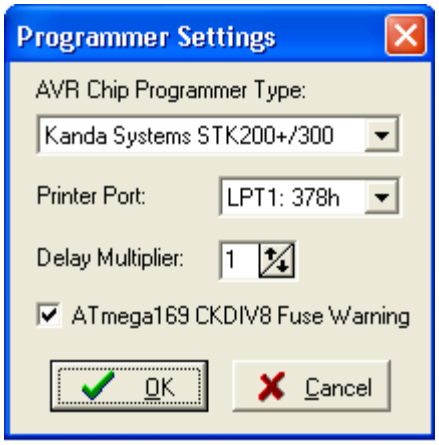

Illustration 13: "programmer settings"

Il reste maintenant deux éléments à paramétrer. La fenêtre « Programmer Settings », ci-dessous, s'obtient en suivant le chemin Settings->Prommager. Par défaut, le premier champ est « AtmelSTK500/AVRISP »; il faut alors sélectionner « Kanda Systems STK200+/300 ».

Enfin, il reste la configuration du projet à faire : il faut suivre le chemin Project->Configure. Une fois dans l'onglet « After Make », il suffit de cocher les cases « Program the Chip » et « Execute User's Program ».

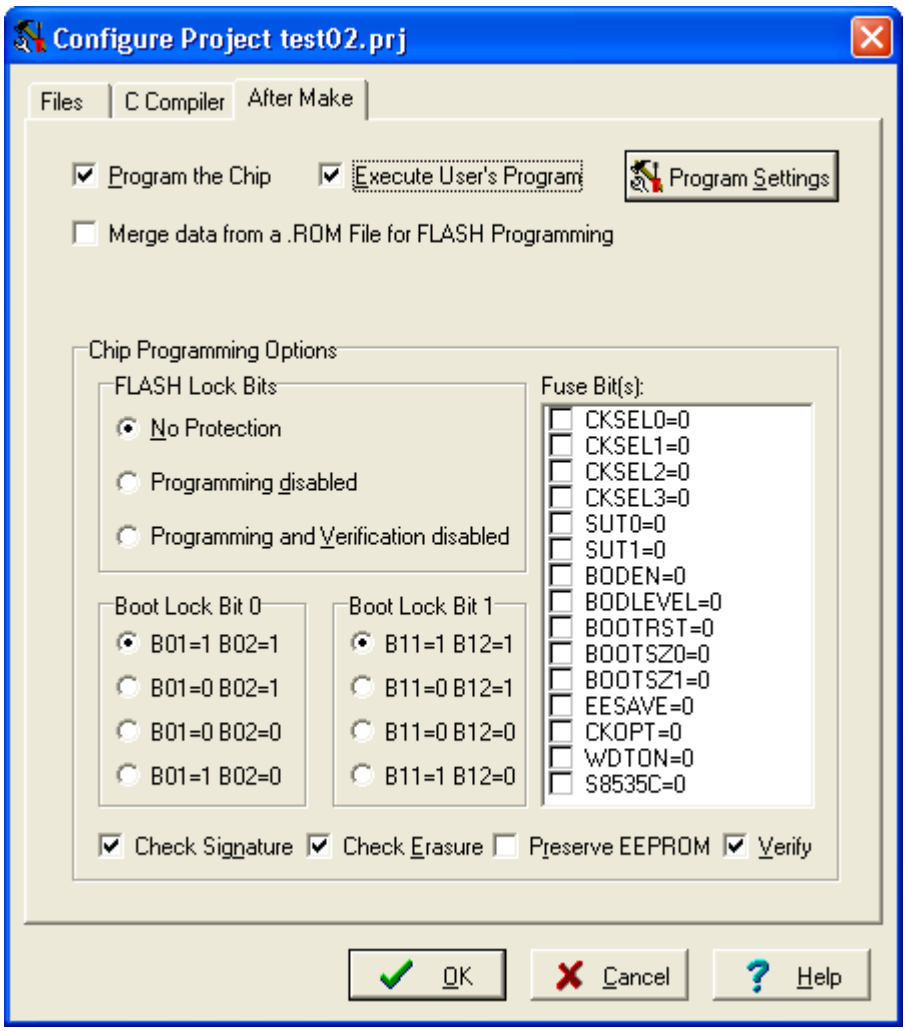

Illustration 14: "after make"

2.3. Programmation

Dans cette partie, nous atteignons le cœur du projet. Après avoir configuré le logiciel CodeVisionAVR et utilisé l'outil automatique de génération de code, nous pouvons passer à la programmation du composant. Dans un premier temps, nous présenterons les fonctions principales du programme, leur ordinogramme qui constitue une première approche, puis le code associé. Enfin, nous verrons le programme final, son ordinogramme et le code. Nous commenterons et expliquerons le code. L'intégralité du code sera disponible en annexes.

2.3.1. Ordinogrammes et code

Cette sous-partie est consacrée à l'explication des fonctions que nous avons créées. Ces fonctions sont au nombre de trois : la première concerne la gestion des clignotants et du mode warning, la deuxième traite le mode manuel d'allumage des feux – c'est le conducteur qui choisit quels feux enclencher, la dernière s'occupe du mode automatique – les feux principaux s'illuminent grâce à une photodiode. Les entrées sont en logique inversée : un zéro indique l'état haut.

2.3.1.1. Fonctions Clignotants et Timer

Tout d'abord, regardons l'ordinogramme de la fonction Clignotants :

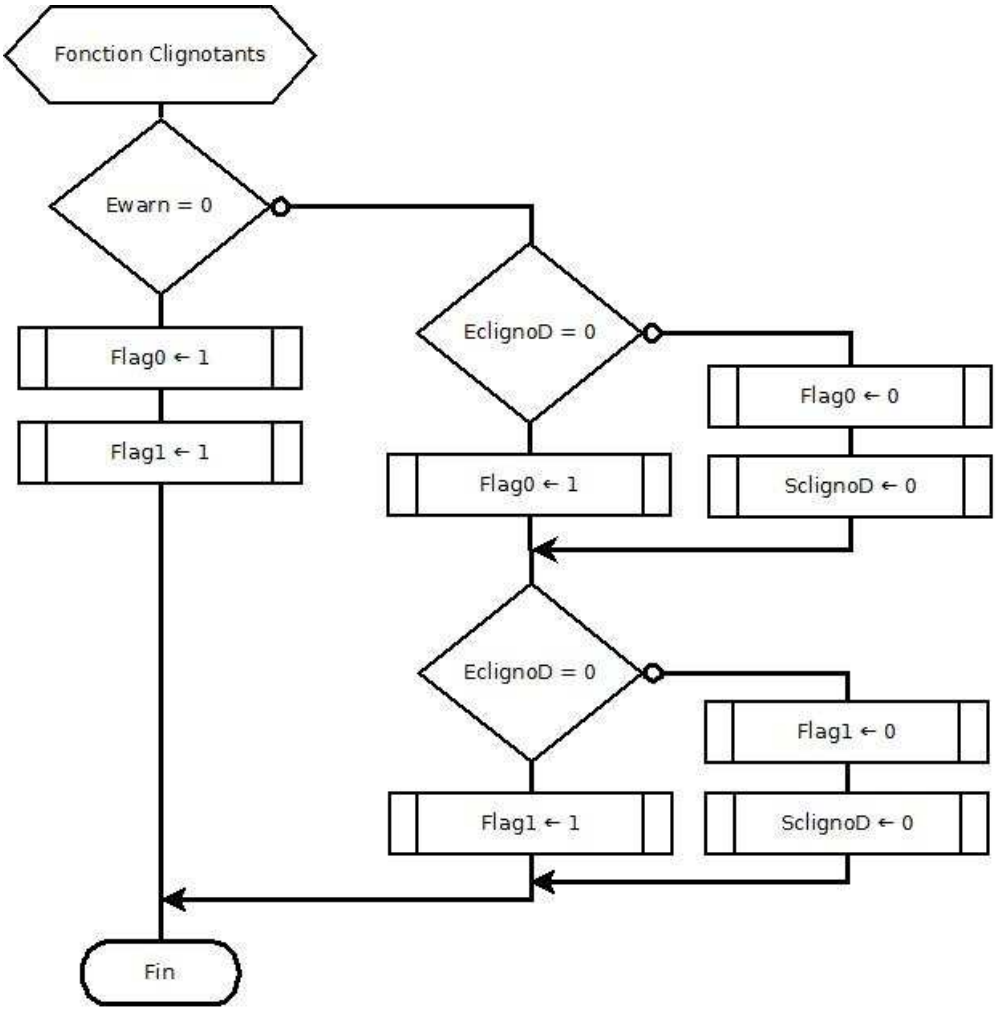

Illustration 15: ordinogramme de la fonction "Clignotants"

Les entrées détectées sont Ewarn, EclignoD et EclignoG. Cette fonction ne gère pas directement de sortie mais des « flags ». Il s'agit de drapeaux. Ce sont des variables internes déclarées en global, c'est-à-dire qu'elles sont utilisables dans tout le programme, pas seulement dans la fonction dans laquelle elles sont déclarées. Ces flags permettent donc de stocker un état (1 ou 0) qui est ensuite utilisé dans un timer.

Le code correspondant est :

```
if(Ewarn == 0)

{

        Flag0 = 1;
        Flag1 = 1;
}
else
{
        if (EclignoD == 0) Flag0 =1;
        else
        {
                Flag0 = 0;
                SclignoD = 0;
        }
        if (EclignoG == 0) Flag1 = 1;
        else
        {
                Flag1 = 0;
                SclignoG = 0;
        }
}
```

Comme nous venons de l'évoquer, la fonction Clignotants change l'état des flags et c'est la fonction Timer qui s'occupe réellement des sorties. Ce timer est en fait une fonction d'interruption interne. Elle est générée automatiquement par le composant, sans intervention extérieure (à l'aide d'un bouton poussoir, par exemple) à la différence d'une interruption externe. Elle s'exécute toutes les 500 ms environ de manière à avoir une période de clignotement d'une seconde.

Voici le code et l'ordinogramme :

```
interrupt [TIM1_COMPA] void timer1_compa_isr(void)

{
        if (Flag2 == 1)
        {
            if (Flag0 == 1) SclignoD = 1;
            if (Flag1 == 1) SclignoG = 1;
            Flag2 = 0;
        }
        else
        {
            SclignoD = 0;
            SclignoG = 0;
            Flag2 = 1;
        }
}
```

Les sorties sont SclignoD et SclignoG et nous retrouvons les deux flags. Cependant un nouveau flag apparaît, il est initialisé à 1. Celui-ci est testé à chaque début de l'interruption. Au premier passage, le programme passe le « else », éteint les clignotants et met Flag2 à 1. Ainsi nous obtenons une demi-seconde où les clignotants ne sont pas actionnés. Par conséquent, la demi-seconde suivante, c'est-à-dire au deuxième tour de la fonction, le programme entre dans le « if » et allume un clignotant, les deux ou aucun, selon la demande du conducteur. Puis le Flag2 est remis à 0 et le timer s'exécute à nouveau.

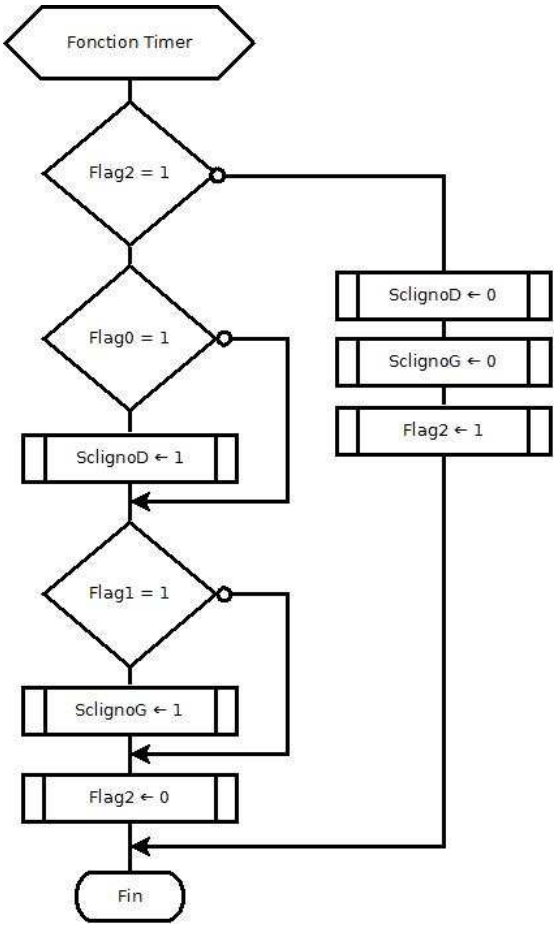

Illustration 16: ordinogramme du timer

2.3.1.2. Fonction Mode_Manuel

La fonction Mode_Manuel permet au conducteur d'allumer lui-même les feux de codes ou les pleins phares. De plus, ce code est prévu pour être utilisé avec le commodo. Ceci dit, il fonctionne également avec le boîtier de test.

```
void Mode_Manuel(void)
{
          if(Ecode == 0)
          {
            ScodeAV = 1;
            ScodeAR = 1;
            Svert = 1;
          }
          else
          {
            if(Epp == 1)
            {
                  ScodeAV = 0;
                  ScodeAR = 0;
                  Svert = 0;
            }
          }
```

```
            if(Epp == 0)
            {
                  ScodeAV = 1;
                  ScodeAR = 1;
                  Svert = 1;
                  Sblanc1 = 1;
                  Sblanc2 = 1;
            }
            else
            {
            if(Ecode == 1)
            {
                  ScodeAV = 0;
                  ScodeAR = 0;
                  Svert = 0;
                  Sblanc1 = 0;
                  Sblanc2 = 0;
            }
          }
}
```

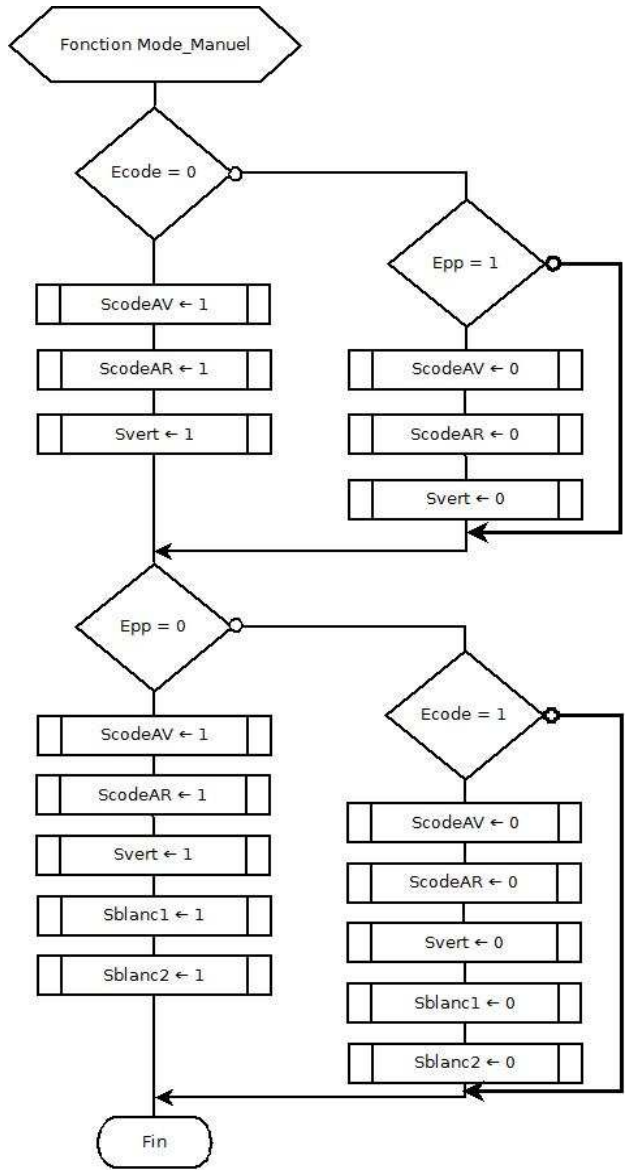

Illustration 17: ordinogramme de la fonction Mode_Manuel

Nous rappelons que nous sommes en logique inversée.

Cette fonction s'effectue de la manière suivante : nous faisons un test sur l'entrée Ecode. Si elle est active les feux de code s'allument. Sinon, un test sur l'entrée Epp (pleins phare) est fait. Si Epp est à l'état bas, les feux de code sont éteints, sinon rien ne se passe.

Le même schéma est reproduit ensuite : nous testons en premier Epp puis Ecode si Epp à 1.

Lors des premiers essais, il n'y a pas de test dans les « else ». Les feux de code sont éteints si l'entrée est à l'état bas (de même pour les pleins phares). Comme les codes font partit des pleins phares, nous n'avions pas tous les feux voulus lors des pleins phares, les codes restant éteints. Le schéma ci-contre permet de résoudre ce problème.

2.3.2. Programme final

Dans cette partie, nous allons présenter le reste du programme, c'est-à-dire ce qui se trouve en dehors des fonctions que nous devons décrire. Nous préciserons ainsi les fichiers à inclure, les définitions des pins d'entrée/sortie et leur mnémonique, la fonction de lecture des entrées analogiques, puis l'initialisation des ports, des timers, des entrées analogiques et des feux.

Tout d'abord, il y a un fichier indispensable à inclure dans le projet pour obtenir toutes les fonctions nécessaires à ce projet. Un second fichier peut venir compléter le premier mais il est surtout utile pour les tests. La syntaxe est la suivante :

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

Ensuite, nous avons renommé les pins d'entrée et de sortie pour augmenter la lisibilité du programme. Ainsi, nous n'avons pas à nous reporter sans cesse aux tableaux. Il faut utiliser la commande #define comme dans le code ci-dessous :

```
// Définition des entrées :
// Entrées du commodo
#define Eauto PINC.0          // Switch fonctionnement automatique / manuel
#define Ewarn PINC.1          // Warning
#define EclignoD PINC.2       // Clignotant droit
#define EclignoG PINC.3       // Clignotant gauche
#define Ecode PINC.5          // Feux de code
#define Epp PINC.4            // Pleins phares
#define Efrein PINC.6         // Pédale de frein

// Entrées analogiques
#define AMavant PINA.0        // Détection marche avant
#define AMarriere PINA.1      // Détection marche arrière
#define Aphdiode PINA.2       // Photo diode
#define Apfrein PINA.3        // Potentiomètre de la pédale de frein

// Définitions des sorties :
// Clignotants
#define SclignoD PORTD.0      // Clignotants droit
#define SclignoG PORTD.1      // Clignotants gauche

// Feux de codes
#define ScodeAV PORTD.3       // Feux de codes avant
#define ScodeAR PORTD.5       // Feux de codes arrière
#define Svert PORTD.6         // Feux de codes cadre vert

// Pleins Feux
#define Sblanc1 PORTD.4       // Pleins feux avant cadre blanc
#define Sblanc2 PORTD.7       // Pleins feux  avant blancs

// Feux de recul
#define Srecul PORTD.2        // Feux de recul
```

Lorsque nous avons paramétré les timers et les entrées analogiques dans le logiciel, ce dernier a généré du code que nous décomposerons ici.

La dernière fonction en dehors du « main » est la fonction de lecture des entrées analogiques. Elle se présente sous la forme de trois blocs. Le premier concerne l'initialisation du comparateur analogique, le deuxième l'initialisation de la fonction de lecture et le troisième est la fonction elle-même. Attention : les deux premiers blocs se situent dans le main.

```
// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
// Analog Comparator Output: Off
ACSR=0x80;
SFIOR=0x00;

// 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;

#define ADC_VREF_TYPE 0x20
// Read the 8 most significant bits
// of the AD conversion result
// Read the AD conversion result
unsigned int read_adc(unsigned char adc_input)
{
ADMUX = adc_input | (ADC_VREF_TYPE & 0xff);
// Start the AD conversion
ADCSRA |= 0x40;
// Wait for the AD conversion to complete
while ((ADCSRA & 0x10) == 0);
ADCSRA |= 0x10;
return ADCW;
}
```

Maintenant, nous entrons dans la fonction « main ». C'est la fonction principale, tout programme en langage C en possède une. Elle appelle toutes les autres fonctions et initialise les ports, les timers et les feux.

Commençons par l'initialisation des ports :

```
// 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=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
PORTB=0x00;
DDRB=0xFF;
```

```
// Port C 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
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;
```

Nous remarquerons que la valeur du port est 0 qu'il soit en entrée ou en sortie ; en revanche le DDR change selon l'état des bits (State) : en entrée il vaut 0 et 1 pour les ports de sortie.

Passons maintenant aux timers. L'ATMega 8535 en compte trois qui apparaissent automatiquement dans le programme même s'ils ne sont pas utilisés, comme le timer 2 dans notre cas. Toutes ses valeurs sont alors à zéro. Les deux autres possèdent des valeurs correspondantes à celles que nous avons choisies lors du paramétrage.

```
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 16000,000 kHz
// Mode: Phase correct PWM top=FFh
// OC0 output: Non-Inverted PWM
TCCR0=0x61;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 15,625 kHz
// Mode: CTC top=OCR1A
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x00;
TCCR1B=0x0D;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x1E;
OCR1AL=0x84;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;
```

Les lignes suivantes sont importantes pour le bon fonctionnement des timers. Il faut d'abord autoriser les interruptions, qu'elles soient internes ou externes :

```
// Global enable interrupts
#asm("sei")
```

Puis initialiser les compteurs :

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

Maintenant, nous précisons que les interruptions externes sont désactivées :

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

Ensuite, nous initialisons les feux afin qu'ils soient tous éteints lorsque nous envoyons le programme dans le composant :

```
//Initialisation des feux
ScodeAV = 0;
ScodeAR = 0;
Svert = 0;
Sblanc1 = 0;
Sblanc2 = 0;
SclignoD = 0;
SclignoG = 0;
Srecul = 0;
```

Enfin, nous atteignons le « while ». Il s'agit d'une boucle infinie dans laquelle sont appellées les fonctions décrites précédemment :

```
while (1)
{
    Clignotants();

    if(Eauto == 0) Mode_Manuel();

    else Mode_Automatique();
}
}
```

La fonction Mode_Automatique concerne la gestion d'allumage des feux selon les entrées analogiques. Nous n'avons pas pu mettre en place cette partie. Il semble que les codes testés soient corrects et cohérents. Cependant lors de l'implantation dans le programmation, les tests avec la pédale de frein n'ont pas été concluants. Les entrées analogiques sont très sensibles aux perturbations électriques. Pour utiliser ces fonctions de l'ATMega, il faut une carte électronique de bonne facture. Nous pouvons maintenant passer à la partie concernant cette carte.

3. Carte électronique

La carte électronique sur laquelle est implantée l'ATMega8535 a été réalisée au semestre précédent par l'étudiant Cheikh Abba DIEME. Il s'agit d'une partie indispensable au projet car il faut veiller à ce que tous les composants soient correctement reliés sans qu'ils perturbent les éléments voisins. Les problèmes de compatibilité électromagnétique (CEM) peuvent rapidement intervenir empêchant alors le bon fonctionnement.

Les premiers tests informatiques sur la carte ont été concluants. Il s'agissait de tester les entrées et sorties numériques de la carte. Le port de programmation fonctionne ainsi que la LED associée. Nous avons relevé une erreur de routage : la sortie Sstop est à utiliser avec la pin PB3 et non PB0.

Cependant, nous avons relevé quelques modifications à apporter. Tout d'abord, nous avons remarqué que les condensateurs associés au quartz en sont éloignés. Il est lui-même trop distant de l'ATMega. Le circuit d'horloge est basé sur le quartz et ne fonctionne qu'avec des courants et des tensions très faibles. Le signal d'horloge est alors très sensible aux perturbations venant des autres éléments de la carte. Il faut que le montage de l'horloge soit le moins exposé à ces perturbations.

Ensuite, nous avons observé que l'ATMega n'a pas de condensateurs de découplage entre les pins d'alimentation (+5V et GND). Grâce à un oscilloscope, nous nous sommes aperçus que la tension d'alimentation présentait une ondulation de plus ou moins 450 mV, alors que le signal devrait être du 5V en continu. Le composant étant un élément sensible aux variations de tension à ses bornes et autour, il n'est pas recommandé de lui imposer de telles fluctuations. Nous avons alors soudé des condensateurs de 10 nF chacun entre les pins concernées permettant de se rapprocher d'une tension continue. Nous sommes ainsi passés de plus ou moins 450 mV à plus ou moins 230 mV. Cependant cette ondulation est encore trop forte.

Lors des tests des entrées analogiques, nous avons remarqué que le micro contrôleur avait quelques problèmes pour recevoir correctement les signaux analogiques. Ces entrées acceptent une tension variant entre 0 et 5V. Il faut encore une fois, faire attention à ce qu'elles ne soient pas perturbées par d'autres éléments de la carte. Pour cela, le bornier d'entrée de la carte doit être le plus proche possible des entrées analogiques de l'ATMega, afin d'éviter que les pistes ne parcourent trop de distances créant ainsi des boucles inductives et plus généralement des problèmes de CEM. De plus, dans l'objectif de réduire ces problèmes de compatibilité, il faut ajouter un plan de masse entre les entrées de la carte et du composant.

Enfin, une résistance devra être ajoutée à chaque transistor entre la Grille et la Source (la Source étant reliée à la masse), permettant une transition de l'état 1 à 0 plus rapide. Pour économiser de l'énergie, cette résistance peut être assez élevée. Une meilleure esthétique pourra être apportée en espaçant régulièrement les transistors.

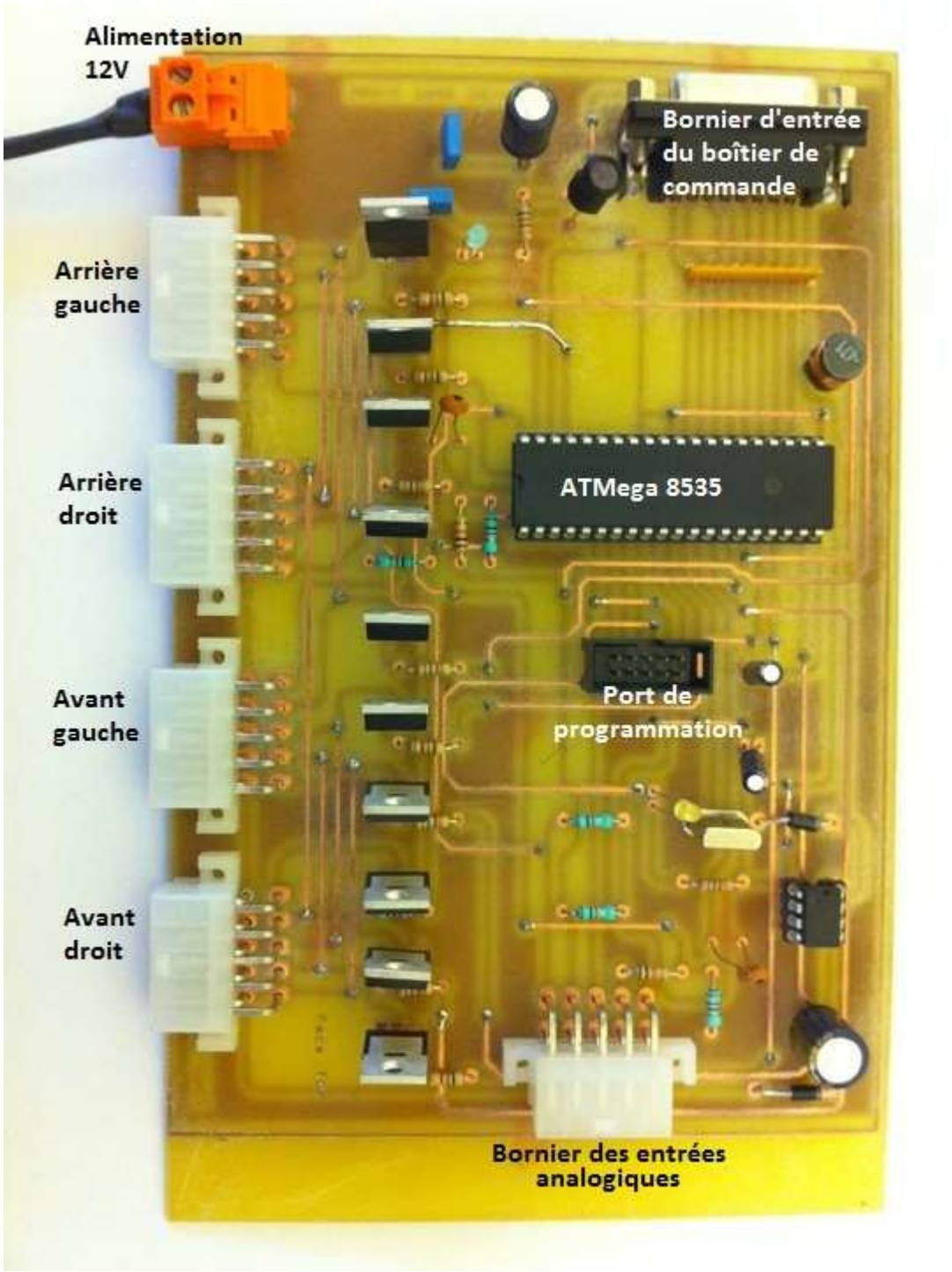

Illustration 18: photo de la carte électronique

4. Intégration au kart électrique

Dans cette dernière partie, nous allons aborder les modifications à mettre en place pour que la carte électronique et le programme puissent être implantés au kart électrique. Nous avons également récupéré un commodo de voiture qui sera à adapter pour servir de commande à la carte électronique.

La première partie de l'intégration concerne le câblage. La première carte incluant l'ATMega ne possède pas les mêmes borniers que celle que nous avons programmée. Nous devons alors modifier les borniers sur le kart électrique, en veillant à ce que les sorties de la carte correspondent bien aux lampes que l'on souhaite allumer. Les fils doivent avoir un embout femelle, car les borniers sont mâles. Il faut utiliser la pinces à sertir Molex rouge (référence 638190900A), avec la taille AWG 24.

Illustration 19: pince molex

Les fils connectés aux borniers ont été numéroté. Pour les feux arrières, les numéros 1 se trouvent à l'extérieur, les numéros croissent en se déplaçant vers l'intérieur.

Nous avons remplacé les lampes afin qu'il y ait les bonnes couleurs au bon endroit.

Dans toutes les voitures, il existe une IHM (Interface Homme Machine) permettant à l'utilisateur d'informer l'ordinateur de bord de ses choix. Cette acquisition passe sur les modèles récents par des centaines de capteurs mais aussi tous les boutons qui entourent le poste de conduite et les places passagers, ainsi que les pédales et les commodos tous raccordés à des bus de capteur. Cependant le karting n'a pas besoin d'acquérir autant d'informations, ce qui nous permet de remplacer l'ordinateur de bord par un simple micro-contrôleur. La commande de l'utilisateur est réduite à un commodo et la pédale de frein. Nous n'utiliserons pas non plus de bus car toutes les entrées et sorties seront câblées jusqu'à la carte électronique.

Le commodo ne doit pas communiquer sur bus CAN, mais par un système classique de contacts. Il a donc été récupéré sur une Renault Laguna I dans une casse. Il dispose de sept fonctionnalités en plus d'un état de repos où aucune requête n'est formulée par l'utilisateur. Aucun document technique n'a pu être trouvé en raison du secret industriel très présent dans ce secteur. Nous alors avons dû mettre en place un protocole de test permettant de déterminer l'utilité de chaque patte. Les tests ont été réalisés grâce à une platine de prototypage et de test Arduino Mega

basé sur un ATMega 2560. Cette carte disposant de 54 entrées et sorties numériques, nous avons pu connecter toutes les fiches du connecteur directement sur la carte.

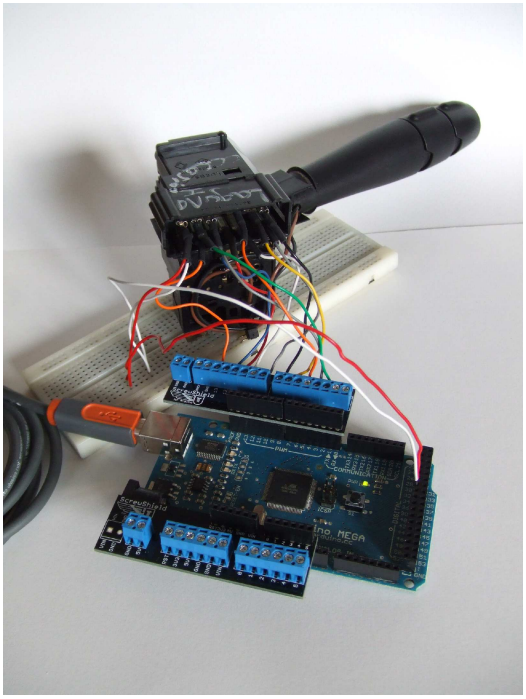

Illustration 20: test du commodo

Le programme de test est très simple d'utilisation. Le numéro de la patte à tester est envoyé à l'ATMega via le port série par l'outil série du logiciel de développement. Une fois l'information acquise par la carte, la patte concernée passe en sortie logique active (5V), les autres pattes sont initialisées en entrées. Le programme balaie ensuite ces entrées et retourne celles à 1 puis le programme attend une nouvelle demande de balayage. Pendant chaque balayage l'utilisateur doit utiliser une fonction du commodo afin de pouvoir déterminer l'utilité de chaque patte.

Les résultats des tests ont été très surprenants. Certaines pattes d'entrées changeant d'état après deux balayages identiques. Les entrées où nous avons rencontré ce genre de problème sont commentées comme fonctionnant « par intermittence ». Pour discerner les contacts francs des autres nous avons effectué des salves d'au moins cinq mesures sur chaque patte. Nous pensons que le problème vient des contacts internes trop oxydés.

Conclusion

Après avoir réalisé plusieurs projets d'électronique au cours des semestres précédents, nous souhaitions nous lancer dans un projet d'informatique. La programmation d'un micro-contrôleur a parfaitement répondu à nos attentes.

Ce projet a été bénéfique dans le sens où il nous a permis d'appliquer plusieurs compétences acquises pendant notre formation. Tout d'abord, nous avons mis en œuvre notre connaissance du langage C. La programmation d'un composant diffère de la création d'une application. La découverte d'un logiciel et sa configuration sont un exercice intéressant bien que difficile si l'on ne possède pas d'aide ou de guide.

Nous avons réalisé un programme qui fonctionne. Cependant, le cahier des charges n'a pas pu être entièrement respecté. La partie concernant les entrées analogiques n'a pas été programmée car nos tests n'étaient pas concluants, à cause de la mauvaise facture de la carte. Nous nous sommes alors penchés sur les autres points essentiels du projet, comme l'intégration au kart.

Nous avons aussi appris à reprendre un projet. Comme le nôtre consiste à implanter un programme, nous sommes totalement dépendants de la conception de la carte accueillant le composant. Nous avons dû la vérifier, faire quelques tests. Après modifications et dépannages, la carte fonctionne en grande partie. La partie analogique est, en revanche, à revoir.

L'intégration au kart est terminée. Les borniers des lampes correspondent à ceux de la carte. Toutes les lampes s'allument, nous avons juste un problème avec les panneaux de LED à l'avant. Les panneaux ne s'allument pas en totalités. Le test du commodo n'ayant pas été concluant, nous ne nous y sommes pas attardés.

Le travail en équipe est un élément essentiel dans l'approche d'un projet. Il permet le développement de plusieurs qualités, autant personnelles que professionnelles. Cette condition nous a permis de travailler sur la communication, l'organisation du projet et le partage des tâches.

Tables des illustrations et des tableaux

Bibliographie

Rapport de projet (2009/2010) d'Étude et Réalisation, sur la carte électrique, de Cheikh Abba DIEME.

AVR033: Getting Started with the CodeVisionAVR C Compiler

Annexes

Code complet

```
/*****************************************************
This program was produced by the
CodeWizardAVR V1.24.2c Professional
Automatic Program Generator
© Copyright 1998-2004 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.ro
e-mail:office@hpinfotech.ro

Project : Gestion des feux du kart électrique
Version :
Date    : 06/10/2010
Author  : F4CG
Company : F4CG
Comments:

Chip type           : ATmega8535
Program type        : Application
Clock frequency     : 16,000000 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 128
*****************************************************/

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

// Définition des entrées :
// Entrées du commodo
#define Eauto PINC.0         // Switch fonctionnement automatique / manuel
#define Ewarn PINC.1         // Warning
#define EclignoD PINC.2      // Clignotant droit
#define EclignoG PINC.3      // Clignotant gauche
#define Ecode PINC.5         // Feux de code
#define Epp PINC.4           // Pleins phares
#define Efrein PINC.6        // Pédale de frein

// Entrées analogiques
#define AMavant PINA.0       // Détection marche avant
#define AMarriere PINA.1     // Détection marche arrière
#define Aphdiode PINA.2      // Photo diode
#define Apfrein PINA.3       // Potentiomère de la pédale de frein

// Définitions des sorties :
// Clignotants
#define SclignoD PORTD.0      // Clignotants droit
#define SclignoG PORTD.1      // Clignotants gauche

// Feux de codes
#define ScodeAV PORTD.3         // Feux de codes avant
#define ScodeAR PORTD.5     // Feux de codes arrière
#define Svert PORTD.6         // Feux de codes cadre vert

// Pleins Feux
#define Sblanc1 PORTD.4       // Pleins feux avant cadre blanc
#define Sblanc2 PORTD.7       // Pleins feux  avant blancs
```

```
// Feux stop et de recul
#define Sstop PORTB.3          // Feux Stop
#define Srecul PORTD.2         // Feux de recul

// Declare your global variables here
int Flag0 = 0;
int Flag1 = 0;
int Flag2 = 0;

// Interruption 0.5 Hz
interrupt [TIM1_COMPA] void timer1_compa_isr(void)
{
        if (Flag2 == 1)
        {
            if (Flag0 == 1) SclignoD = 1;
            if (Flag1 == 1) SclignoG = 1;
            Flag2 = 0;
        }
        else
        {
            SclignoD = 0;
            SclignoG = 0;
            Flag2 = 1;
        }
}

void Mode_Manuel(void);
void Clignotants(void);

#define ADC_VREF_TYPE 0x20
// Read the 8 most significant bits
// of the AD conversion result
// Read the AD conversion result
unsigned int read_adc(unsigned char adc_input)
{
ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);
// Start the AD conversion
ADCSRA|=0x40;
// Wait for the AD conversion to complete
while ((ADCSRA & 0x10)==0);
ADCSRA|=0x10;
return ADCW;
}

void Mode_Manuel(void)
{
          if(Ecode == 0)
          {
            ScodeAV = 1;
            ScodeAR = 1;
            Svert = 1;
          }
          else
          {
            if(Epp == 1)
            {
                  ScodeAV = 0;
                        ScodeAR = 0;
                  Svert = 0;
            }
          }
```

```
            if(Epp == 0)
            {
                  ScodeAV = 1;
                  ScodeAR = 1;
            Svert = 1;
            Sblanc1 = 1;
                  Sblanc2 = 1;
            }
            else
            {
            if(Ecode == 1)
            {
                ScodeAV = 0;
                        ScodeAR = 0;
                  Svert = 0;
                  Sblanc1 = 0;
                        Sblanc2 = 0;
            }
          }
}

void Clignotants(void)
{
        if(Ewarn == 0)
        {
            Flag0 = 1;
            Flag1 = 1;
        }
        else
        {
            if (EclignoD == 0) Flag0 =1;
            else
            {
                  Flag0 = 0;
                  SclignoD = 0;
            }
            if (EclignoG == 0) Flag1 = 1;
            else
            {
                  Flag1 = 0;
                  SclignoG = 0;
            }
        }

}

void main(void)
{
// Declare your local variables here
unsigned int Afrein;

// 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=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
PORTB=0x00;
DDRB=0xFF;

// Port C 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
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: 16000,000 kHz
// Mode: Phase correct PWM top=FFh
// OC0 output: Non-Inverted PWM
TCCR0=0x61;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 15,625 kHz
// Mode: CTC top=OCR1A
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x00;
TCCR1B=0x0D;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x1E;
OCR1AL=0x84;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
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
// Analog Comparator Output: Off
ACSR=0x80;
SFIOR=0x00;

// 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;

// Global enable interrupts
#asm("sei")

//Initialisation des feux
ScodeAV = 0;
ScodeAR = 0;
Svert = 0;
Sblanc1 = 0;
Sblanc2 = 0;
SclignoD = 0;
SclignoG = 0;
Sstop = 0;
Srecul = 0;

Afrein = read_adc(3);

while (1)
{
    Clignotants();

    if(Eauto == 0) Mode_Manuel();

    else Mode_Automatique();
}
}
```