





Fiche guide 3	TS SI		P.P.E Effets de lumière et commande DMX512	 académie d'Orléans-Tours Éducation nationale enseignement supérieur recherche 
Analyse et synthèse	4h			
 Lycée Polyvalent PIERRE EMILE MARTIN	Réglage de la luminosité des LED du projecteur			

Nom(s) :	Classe :	Groupe :
----------	----------	----------

Objectifs

Etablir l'algorithme de la partie du programme, à implanter dans le projecteur, chargée du réglage de la luminosité des LED.

Matériels

1 carte ATMELESSI + 1 carte étude « LED haute luminosité » dans son boîtier de protection.

Logiciel

CodeVisionAVR.

Documentation

Schéma carte SSI + Schéma carte étude LED haute luminosité. Résumé sur le langage C. Documentation LM3404. Documentation de afficheur LCD.

Le présent document et la documentation sont téléchargeables sur le site WebGE à l'adresse <http://p.mariano.free.fr/> (rubrique PPE)

Sommaire

- A) Présentation
 - A1) Schéma et principe de fonctionnement
 - A2) Information sur le réglage de la luminosité des LED (commande PWM)
- B) Analyse d'une solution existante
 - B1) Etape 1 : Calcul des coefficients α (rapport cyclique)
 - B2) Etape 2 : Création d'un projet avec CVAVR
 - B3) Etape 3 : Ecriture du programme destiné à la carte d'étude
- C) Programmation
- D) Tests
- E) Synthèse : Algorithme du programme destiné au projecteur à réaliser

CONSIGNES DE SECURITE



ATTENTION AUX LED BRILLANTES

Ne jamais regarder directement/fixer les LED.

Le rayonnement très brillant des LED n'est pas seulement désagréable, il est aussi dangereux pour les yeux car il peut endommager la rétine.



A) Présentation

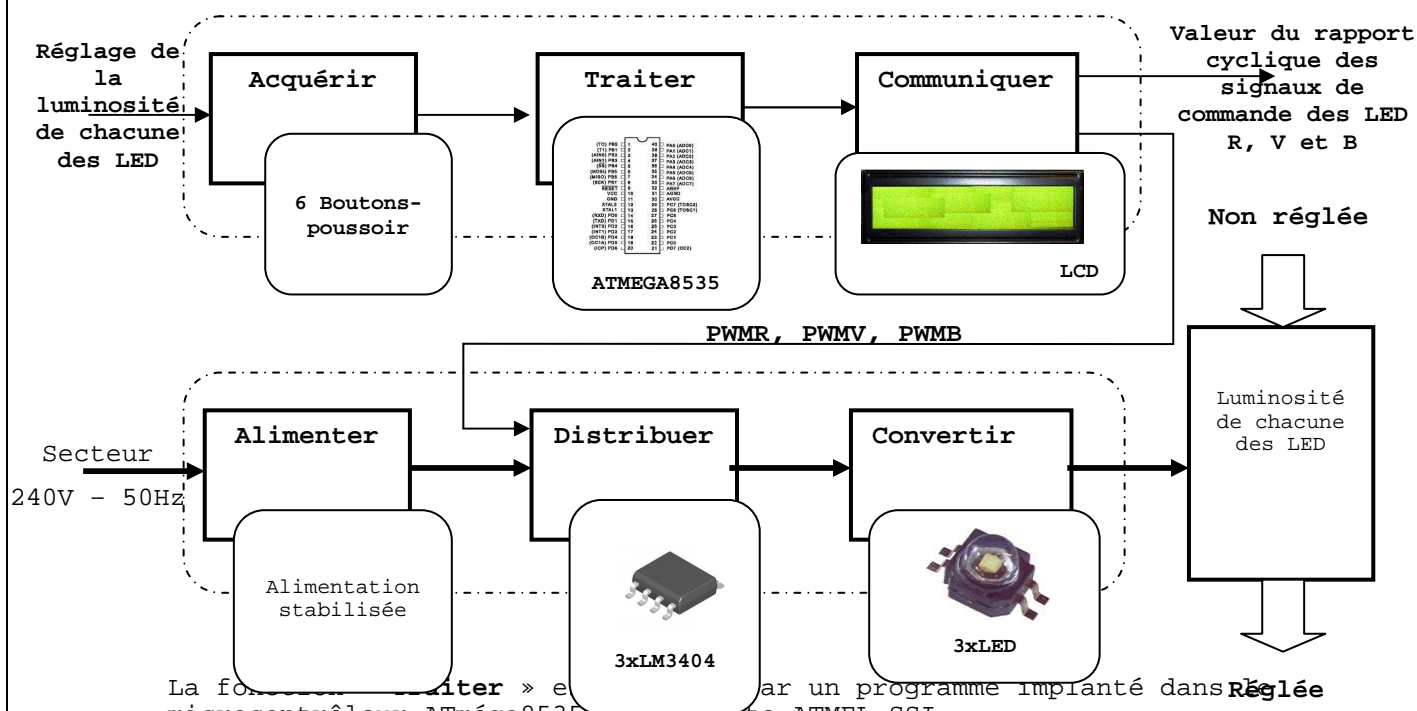
Afin d'écrire l'algorithme destiné au projecteur, vous allez tout d'abord **analyser** un programme permettant de faire varier la luminosité des LED implantées sur une carte d'étude.

Vous **réinvestirez** ce travail dans la dernière partie de ce document (§E **Synthèse**) pour répondre à l'objectif fixé en page de garde.

A1) Schéma et principe de fonctionnement

On souhaite **contrôler** la luminosité des LED rouge, verte et bleu de la carte d'étude (Annexe 1).

Pour cela, on propose de mettre en œuvre une structure correspondant au schéma ci-dessous.



La fonction « **Distribuer** » est assurée par un programme implanté dans le microcontrôleur ATmega8535 de la carte ATMEL SSI.

Les signaux PWM (**PWMR, PWMV, PWMB**) sont issus de structures, appelées « Timer », intégrées au microcontrôleur. PWMR, PWMV et PWMB permettent à la fonction « **Distribuer** » de **réglage la luminosité** des LED rouge, verte et bleu. Cette fonction est réalisée par un circuit intégré LM3404. Elle sera étudiée dans la fiche guide 4. La fonction « Communiquer » est assurée par un afficheur LCD (à processeur Hitachi).

L'ensemble des structures matérielles étant réunies sur les cartes « ATMELSSI » et « LED haute luminosité », votre travail va se limiter à la **réalisation du logiciel** à implanter dans le microcontrôleur.

Pour cela, vous allez **créer et configurer un projet** avec le magicien du cross-compileur **CodeVisionAVR**. Puis, vous complèterez la structure de ce projet avec les fonctions nécessaires à la mise en œuvre des LED et de l'afficheur.

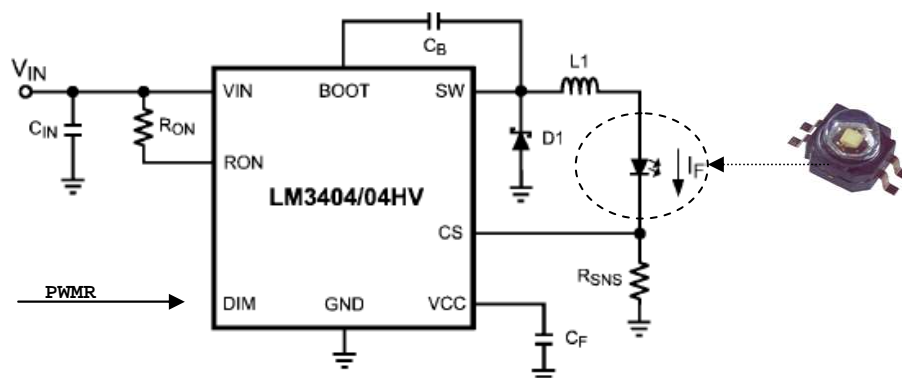
Le travail d'analyse développé dans les paragraphes A à D doit vous aider à écrire l'algorithme de réglage de la luminosité des LED pour le projecteur à réaliser (paragraphe E). La suite de ce document décrit le travail à réaliser étape par étape.



A2) Information sur le réglage de la luminosité des LED (commande MLI ou PWM)

Sur la carte d'étude « LED haute luminosité » chaque LED est commandée par un **régulateur de courant** de type LM3404 (spécialement conçu pour cette application). Il se charge de maintenir **constante l'intensité** I_F traversant la LED. Ce régulateur peut également régler sa luminosité si on applique un signal **modulé en largeur d'impulsions** (MLI ou PWM*) sur son entrée DIM (Dimming).

- Schéma de la fonction « distribuer » de la LED rouge (Doc National Semiconductor)



Vous devez donc être capable de produire des signaux MLI (PWM) avec un microcontrôleur. Le principe de ce type de commande est expliqué dans le paragraphe suivant. Vous serez ensuite guidé pour le mettre en œuvre !

- Génération d'un signal Modulé en Largeur d'Impulsion : MLI ou PWM* (principe)

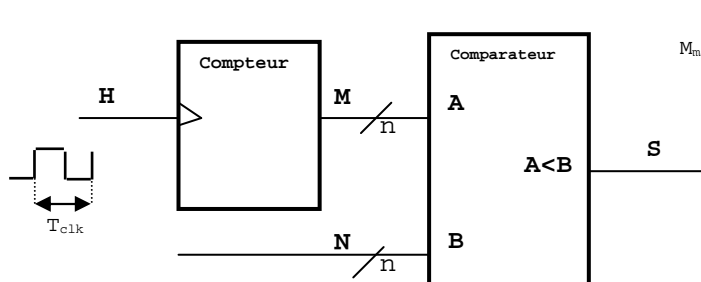
Un signal modulé en largeur d'impulsion peut être obtenu à partir d'un signal **périodique H** de fréquence fixe $F_{clk} = 1/T_{clk}$. En effet, en appliquant ce signal à l'entrée d'un **compteur**, on obtient un signal numérique M (codés sur n bit) capable d'évoluer entre 0 et $2^n - 1$.

La représentation de $M_{(t)}$ est appelée **rampe numérique**.

En appliquant $M_{(t)}$ et un signal constant $N_{(t)}$, codé sur n bit, à un **comparateur** numérique, on obtient un signal binaire $S_{(t)}$ de période $T = 2^n \cdot T_{clk}$ dont le temps t_1 (à l'état « 1 ») est réglé avec la valeur de N.

On appelle $\alpha = t_1/T$ le **rapport cyclique** du signal $S_{(t)}$. On montre que la **valeur moyenne** S_{moy} de $S_{(t)}$ est égale au produit de α par la valeur maximum de S (S_{max}).

On donne ci-dessous le schéma de principe d'une structure générant un signal M.L.I et les chronogrammes de $S_{(t)}$ pour **deux valeurs particulières** de N (N_1 et N_2).



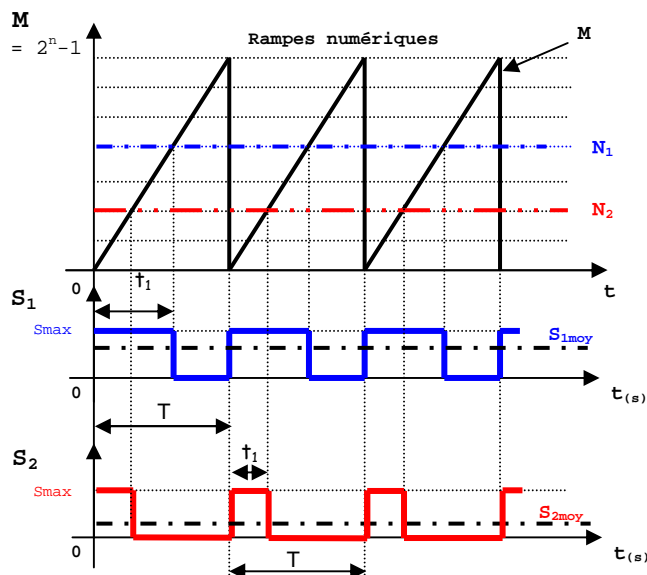
M : valeur numérique variable
($0 \leq M \leq 2^n - 1$)

N : valeur numérique constante

Comparateur : si (A < B) alors S = S_{max}
sinon S = 0

H : signal périodique de fréquence F_{clk}

n : nombre de bit de la structure logique



Dans les microcontrôleurs, les signaux modulés en largeur d'impulsion sont produits par une structure appelée **TIMER**. Celle-ci fonctionne selon le principe développé ci-dessus.

$$S_{moy} = \alpha \cdot S_{max}$$

*M.L.I : Modulation de largeur d'impulsion (P.W.M. : Pulse With Modulation)



- Génération d'un signal MLI avec le microcontrôleur ATMEGA8535
Le **Timer 1** de l'ATMEGA8535 permet de générer deux signaux modulés en largeur d'impulsion. Ces signaux sont identifiés par **OC1B** et **OC1A**. Ils sont accessibles sur les broches 4 et 5 du microcontrôleur (port D).

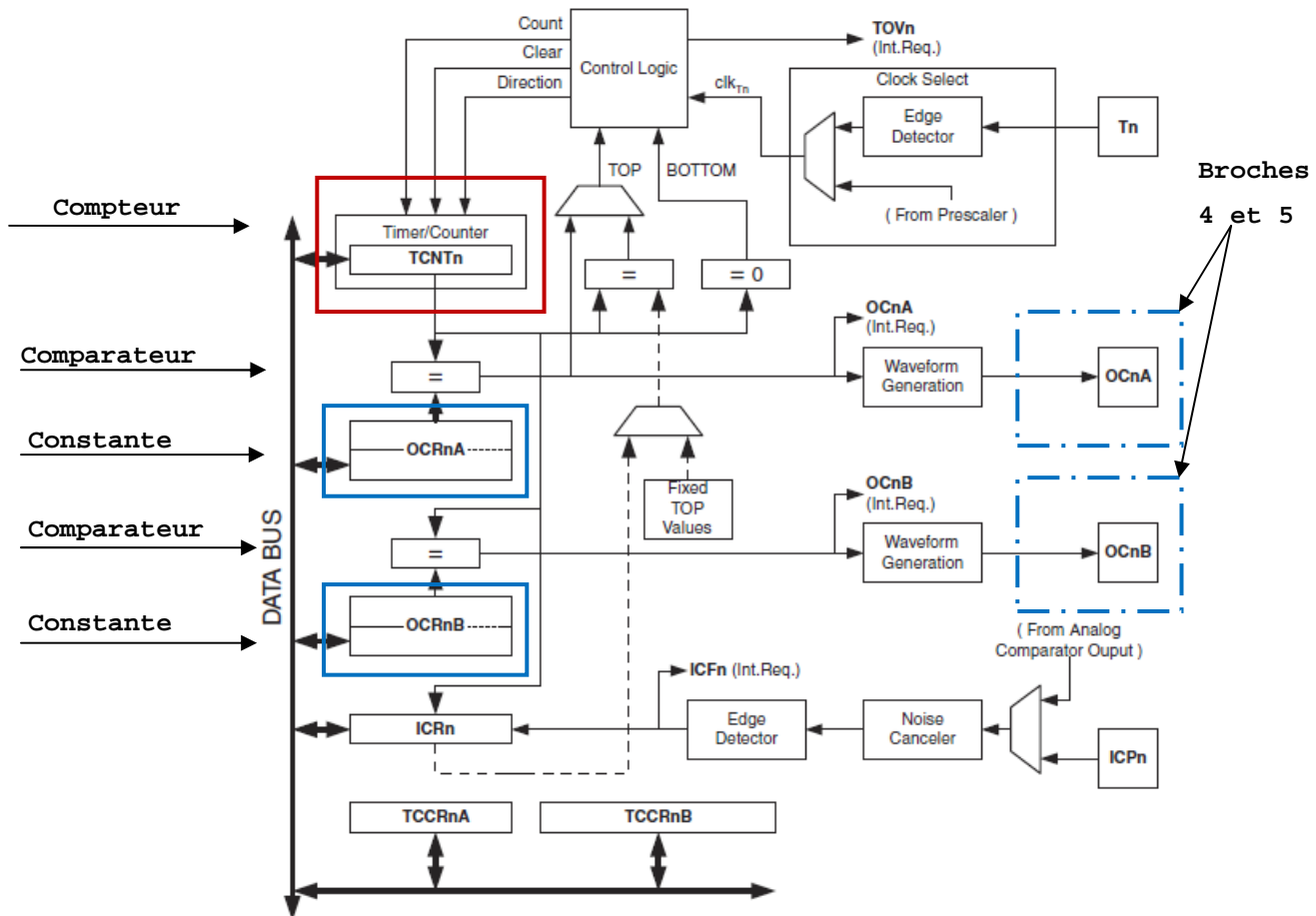


Le **Timer 1** intègre un **compteur**, **deux comparateurs** et divers registres.

En mode M.L.I. son fonctionnement répond au principe exposé dans le paragraphe précédent.

(T0) PB0	1	40	PA0 (ADC0)
(T1) PB1	2	39	PA1 (ADC1)
(AIN0) PB2	3	38	PA2 (ADC2)
(AIN1) PB3	4	37	PA3 (ADC3)
(SS) PB4	5	36	PA4 (ADC4)
(MOSI) PB5	6	35	PA5 (ADC5)
(MISO) PB6	7	34	PA6 (ADC6)
(SCK) PB7	8	33	PA7 (ADC7)
RESET	9	32	AREF
VCC	10	31	AGND
GND	11	30	AVCC
XTAL2	12	29	PC7 (TOSC2)
XTAL1	13	28	PC6 (TOSC1)
(RXD) PD0	14	27	PC5
(TXD) PD1	15	26	PC4
(INT0) PD2	16	25	PC3
(INT1) PD3	17	24	PC2
(OC1B) PD4	18	23	PC1
(OC1A) PD5	19	22	PC0
(ICP) PD6	20	21	PD7 (OC2)

Schéma fonctionnel du timer 1 de l'ATmega8535 (Doc ATMEL)



Le compteur TCNTn (**n=1 ici**) génère le signal numérique M. Les registres OCRnA et OCRnB contiennent chacun une valeur numérique N. Les broches OCnA et OCnB sont les sorties S des comparateurs.

Le Timer 1 contient donc **deux structures** dont le fonctionnement répond au principe exposé dans le paragraphe précédent.

Le Timer 0 contient également **une** structure dont le fonctionnement répond au principe exposé dans le paragraphe précédent.

Ces **trois structures** vont nous permettre de produire les signaux **PWMR, PWMV et PWMB**. Le registre OCR1A contiendra la valeur nécessaire à PWMR, le registre OCR1B celle nécessaire à PWMV et le registre OCR0 celle nécessaire à PWMB.

Dans le paragraphe suivant, vous allez **établir la relation** permettant de calculer la valeur à placer dans un registre **OCRx**, pour obtenir un **rapport cyclique α particulier** (donc un **éclairement particulier**) ($x=1A, 1B$ ou 0). On note cette relation **$OCRx = f(\alpha)$** .



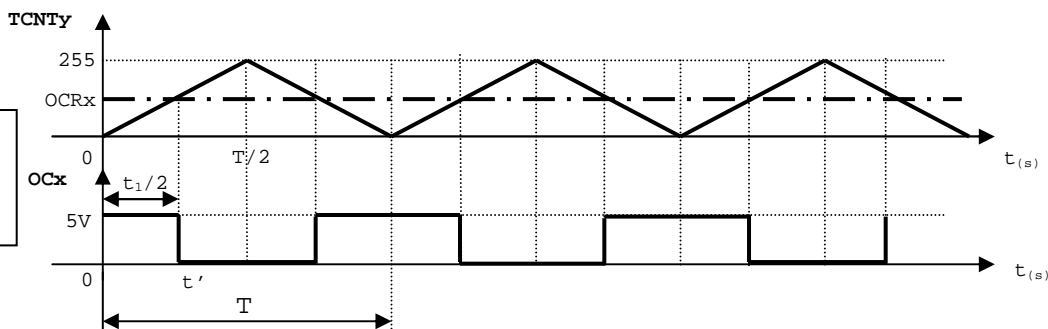
B) Analyse d'une solution existante

B1) Etape 1 : Calcul des coefficients correspondant à α (rapport cyclique)

Objectif : Déterminer les valeurs à placer dans les registres OCRx (x=1A, 1B ou 0) afin de régler la luminosité des LED avec un signal dont le rapport cyclique varie par pas de 10%.

Les chronogrammes ci-dessous sont extraits des « Datasheets » ATMEL.

Rappel : OCRx contient la valeur comparée à celle évoluant dans TCNTy.



Q1a) Exprimez $TCNTy = f(t)$ pour $t \in [0, T/2]$ Remarque : $y=1$ ou 0

Q1b) Exprimez $t' = f(t_1)$ (1)

Q1c) A l'instant $t = t'$, $TCNTy = OCRx$, exprimez $t' = f(OCRx)$ (2)

Q1d) Exprimez $OCRx = f(\alpha)$ à partir des expressions (1) et (2)

Q1e) Complétez le tableau ci-dessous (arrondissez à l'entier supérieur)

ATTENTION : Le rapport cyclique α est exprimé en pourcentage.

$\alpha(\%)$	0	10	20	30	40	50	60	70	80	90	100
OCRx											

Vous disposez maintenant des valeurs permettant de régler l'énergie appliquée au LED (par pas de 10%). Il vous reste à écrire le programme qui utilisera ces valeurs. C'est l'objectif des prochains paragraphes.



B2/ Etape 2 : Création et configuration d'un projet avec CVAVR

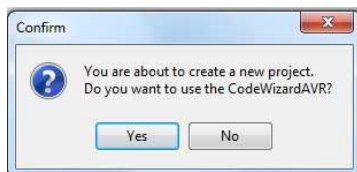
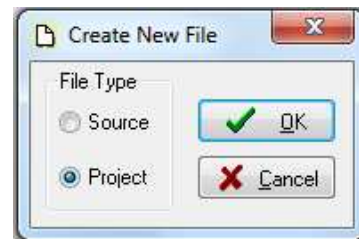
Lancez le logiciel CodeVisionAVR



(1) Création d'un nouveau projet

Dans la barre d'outils : « **File** » puis « **New** » pour obtenir la boîte de dialogue ci-contre.

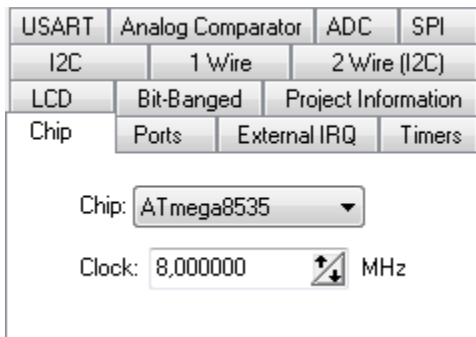
Cochez « **Project** » puis clic sur « **Ok** »



Ici « **Yes** »

(2) Sélection du composant cible

La boîte du « **Magicien** » ci-dessous s'ouvre. Choisissez le « **Chip** » ATMEGA8535 et réglez le signal d'horloge « **Clock** » à 8Mhz.



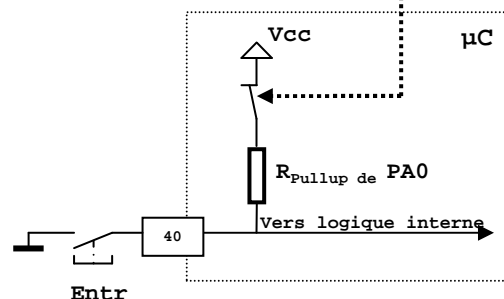
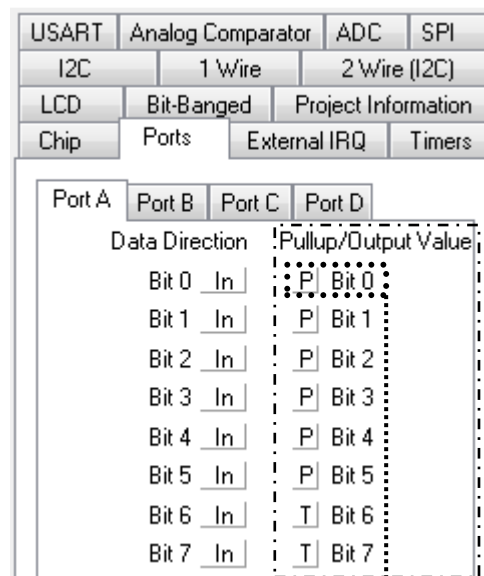
(3) Configuration des ports A, B et D

- Sélectionnez l'onglet « **Port A** ».
Les boutons-poussoirs de la carte ATMEL SSI sont connectés au Port A du microcontrôleur. (voir schéma structurel « **ATMELSSI V1** »)
- Configurez le Port A comme ci-contre.
Sélectionnez les résistances de « **Pullup** ».

Explications concernant la résistance de Pullup

Chaque broche reliée à un port d'entrée sortie du microcontrôleur est dotée d'une résistance dite de « **Pullup** ». Cette résistance est susceptible d'être reliée, par le logiciel, au potentiel positif de l'alimentation. Elle permet de fixer la valeur du potentiel sur la broche concernée.

Exemple : Sur la carte ATMELSSI, le bouton-poussoir <Entr> est relié à la broche 40 (PA0) du microcontrôleur. Lorsque la résistance de Pullup est connectée, cette broche « voit » un niveau logique « 1 » si <Entr> est ouvert et un niveau logique « 0 » si il est fermé. Sans cette résistance, <Entr> ouvert produirait un état logique « **aléatoire** » (« 0 » ou « 1 ») ;



- Sélectionnez l'onglet « **Port B** ».

Q2a) Configurez le bit 3 du port B après avoir déterminé son sens (entrée ou sortie).

- Sélectionnez l'onglet « **Port D** ».

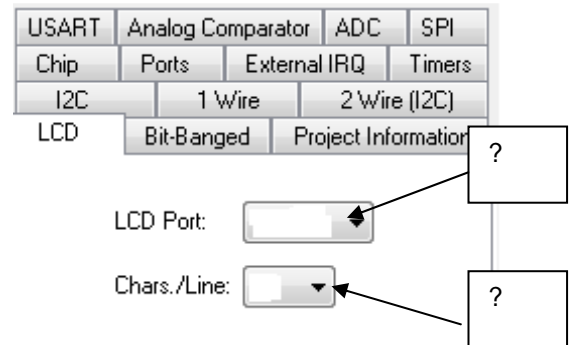
Q2b) Configurez les bit 4 et 5 du port D après avoir déterminé leur sens (entrée ou sortie).



(4) Choix de l'affichage

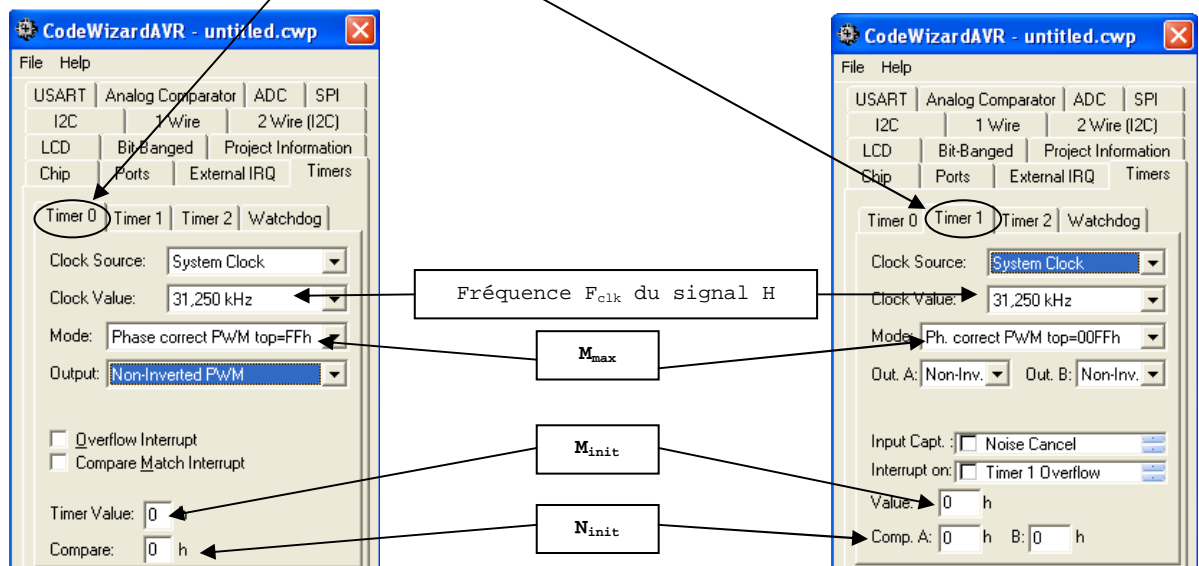
sélectionnez l'onglet « LCD ».

En étudiant le schéma de la carte « **ATMELSSI V1** », et la documentation de l'afficheur LCD, **déterminez** sur quel port est connecté l'afficheur LCD et le nombre de caractères par ligne que comporte cet afficheur.



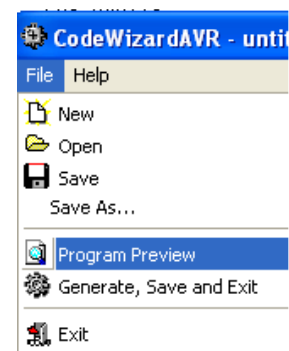
Q2c) Configurez les champs « **LCD Port** » et « **Chars./Line** » de la boîte de dialogue « **LCD** » ci-dessus.

(5) Configuration des timers 0 et 1



(6) Enregistrement du projet

(6) Sélectionnez « Program Preview ».



Si le projet est correctement configuré, les éléments suivants doivent se trouver dans le fichier source du programme.

```
#include <mega8535.h>

// Alphanumeric LCD Module functions
#asm
.equ __lcd_port=0x15
#endasm
#include <lcd.h>

// Declare your global variables here

void main(void)
{
// Declare your local variables here

// 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=P State4=P State3=P State2=P State1=P State0=P
PORTA=0x3F;
DDRA=0x00;

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

// Port D initialization
// Func0=In Func1=In Func2=In Func3=In Func4=Out Func5=Out Func6=In Func7=In
// State0=T State1=T State2=T State3=T State4=0 State5=0 State6=T State7=T
PORTD=0x00;
DDRD=0x30;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 31,250 kHz
// Mode: Phase correct PWM top=FFh
// OC0 output: Non-Inverted PWM
TCCR0=0x64;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 31,250 kHz
// Mode: Ph. correct PWM top=00FFh
// OC1A output: Non-Inv.
// OC1B output: Non-Inv.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0xA1;
TCCR1B=0x04;
TCNT1H=0x00;
TCNT1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// LCD module initialization
lcd_init(16);
```

Fermez la fenêtre.

Sélectionnez

→ File

→ « Generate, save and Exit ».

Donnez le nom **CHCOLO** à votre projet (demandé 3 fois) pour créer les trois fichiers de base du projet. (.c, .prj, .cwp)

ATTENTION : Le Magicien ne peut plus être utilisé pour modifier votre projet.
Voir le prof pour d'éventuelles corrections.



B3) Etape 3 : Ecriture du programme destiné à la carte d'étude

B31) Rappels

Dans ce paragraphe, vous allez compléter la partie **déclarative**...

```
void main(void)
{
// Declare your local variables here
```

... et la partie **exécutive** du programme.

```
while (1)
{
// Place your code here
};
```

On rappelle que la **partie déclarative** d'un programme est la zone dans laquelle sont **créées les variables** alors que la **partie exécutive** est la zone du **traitement** appliqué à ces variables.

B32) Démarche adoptée pour l'écriture du programme

B321) Ecriture de la partie exécutive du programme à réaliser

Le programme à réaliser peut être résumé par les actions suivantes:

(1) **Lire** les consignes de rapport cyclique

(2) **Traiter**

Modifier les valeurs des consignes de rapport cyclique en fonction des actions sur les boutons-poussoirs.

(3) **Ecrire**

Afficher les consignes de rapport cyclique sur un LCD.

Placer les valeurs correspondant aux rapports cycliques dans les registres OCR1A, OCR1B et OCR0.

(1) Lire (les consignes de rapport cyclique)

On souhaite modifier les valeurs des rapports cycliques des signaux PWMR, PWMV et PWMB avec les boutons-poussoirs {INC, DEC} {OK, SET} {ECHAP, ENTR} de la carte ATMELSSI. Les boutons-poussoirs INC, OK et DEC permettront d'incrémenter les rapports cycliques. DEC, SET et ENTR permettront de les décrémenter. L'action sur ces différents boutons-poussoirs est mémorisée dans les **variables Alpha_R, Alpha_V et Alpha_B**.

Exemple : A chaque action sur INC, on incrémente la variable Alpha_R avec la valeur 10.

A chaque action sur Dec, on la décrémente avec la valeur 10.

Un test est effectué pour que Alpha_R soit toujours compris entre 0 et 100%.

On propose la solution ci-dessous. **Complétez** le fichier source C.

```
// Initialisation de l'affichage
// -----
sprintf(display_buffer_ligne0,"Rouge Vert Bleu");
sprintf(display_buffer_ligne1,"%-u%% %-u%% %-u%%",Alpha_R, Alpha_V, Alpha_B);
Affiche_LCD(display_buffer_ligne0,display_buffer_ligne1);

while (1)
{
// début while
// ----- Lire les consignes de rapport cyclique-----
BP = Lire_BP();
// -----
```

(2) Traiter

On propose la solution ci-dessous. **Complétez** le fichier source C.



```
// -----
// ----- Traiter -----
switch(BP)
{
    case INC: if (Alpha_R < 90) Alpha_R = Alpha_R+10; else Alpha_R = 100; break;
    case DEC: if (Alpha_R > 10) Alpha_R = Alpha_R-10; else Alpha_R = 0; break;
    case OK: if (Alpha_V < 90) Alpha_V = Alpha_V+10; else Alpha_V = 100; break;
    case SET: if (Alpha_V > 10) Alpha_V = Alpha_V-10; else Alpha_V = 0; break;
    case ECHAP : if (Alpha_B < 90) Alpha_B = Alpha_B+10; else Alpha_B = 100; break;
    case ENTR : if (Alpha_B > 10) Alpha_B = Alpha_B-10; else Alpha_B = 0; break;
}
// -----
```

Q3) A quelle structure algorithmique correspond le switch ? Par quelle structure algorithmique pourrait-on le remplacer ? Pourquoi un switch a t'il été adoptée ici ? **Compléter** l'algorithme de l'annexe 2.

(3) Ecrire

On propose la solution ci-dessous. **Complétez** le fichier source C.

```
// ----- Ecrire -----
REGISTRE_PWM_R=Table_Alpha[Alpha_R/10]; // REGISTRE_PWM_R = OCR1AL
REGISTRE_PWM_V=Table_Alpha[Alpha_V/10]; // REGISTRE_PWM_V = OCR1BL
REGISTRE_PWM_B=Table_Alpha[Alpha_B/10]; // REGISTRE_PWM_B = OCR0

//On met à jour l'affichage si la valeur du rapport cyclique a été modifiée
if ((Alpha_R != Alpha_R_1) || (Alpha_V != Alpha_V_1) || (Alpha_B != Alpha_B_1))
{
    sprintf(display_buffer_ligne0,"Rouge Vert Bleu");
    sprintf(display_buffer_ligne1,"%-u%% %-u%% %-u%%",R,V,B);
    Affiche_LCD(display_buffer_ligne0,display_buffer_ligne1);
    Alpha_R_1 = Alpha_R; Alpha_V_1 = Alpha_V; Alpha_B_1 = Alpha_B;
}
// -----
```

Q4) **Complétez** l'algorithme de l'annexe 2.

Dans la commande « REGISTRE_PWM_R = Table_Alpha [Alpha_R/10]; », « Table_Alpha » est le nom d'un **tableau de valeurs** et « Alpha_R/10 » permet de calculer une **position** (indice) dans ce tableau. On donne le contenu de ce tableau ci-dessous.

En langage C, la première valeur d'un tableau est accessible à la position 0

Alpha_R/10	Table_Alpha(Alpha_R/10)
0	0
1	26
2	51
3	77
4	102
5	128
6	153
7	179
8	207
9	230
10	255

Lisez le paragraphe sur les vecteurs du document « **Résumé de langage C** ».

Q5) Quelle est la valeur placée dans REGISTRE_PWM_R si le rapport cyclique souhaité est Alpha_R=60 ?

Le tableau « Table_Alpha » réalise une fonction **transcodage** entre la valeur de α et la valeur correspondante à placer dans OCRx.

Malgré la complexité des structures à mettre en œuvre, le programme à réaliser reste relativement simple. Ceci est du à la « richesse » des **bibliothèques de fonctions** fournies avec le cross-compileur CodeVisionAVR.



L'écriture sur le LCD nécessite la fonction `printf()`. Cette fonction est située dans la bibliothèque `stdio` accessible par une référence au fichier `stdio.h`

Une référence à une bibliothèque se fait à l'aide de la directive de compilation `#include <nom bibliothèque>`.

Exemple : `#include <stdio.h>`

B322) Déclaration des bibliothèques de fonctions utilisées dans le programme

Vous devez rajouter une référence à la bibliothèque `ssi` (nécessaire pour accéder aux fonctions `Lire_BP` et `Affiche_LCD`), et à la bibliothèque `delay` (nécessaire pour réaliser une temporisation) à la suite de `#include <mega8535.h>`.

B323) Partie déclarative du programme à réaliser

Le programme ci-dessous utilise deux types de variables :

- **Table_Alpha** : tableau d'octets non signés,
- **Alpha_R, Alpha_R_1, Alpha_V, Alpha_V_1, Alpha_B, Alpha_B_1, BP** : octets non signés
- **display_buffer_ligne0, display_buffer_ligne1** : tableaux de caractères

Pour être reconnues, ces variables doivent être **déclarées** avant leur utilisation.

Complétez le fichier source C comme ci-dessous. **Remplacez** ??? par vos onze valeurs séparées par une virgule.

```
void main(void){
// Declare your local variables here
// -----
//type          nom          désignation
// -----
char display_buffer_ligne0[17]; // tampon ligne 0 de l'afficheur
char display_buffer_ligne1[17]; // tampon ligne 1 de l'afficheur
unsigned char Alpha_R=0, Alpha_R_1=0, Alpha_V=0, Alpha_V_1=0, Alpha_B_1=0, Alpha_B=0;
unsigned char BP; // Bouton-poussoir sélectionné

// Table des valeurs de réglage du rapport cyclique des signaux PWM
unsigned char Table_Alpha[11] = { ???};
```

D'autres parts, nous avons renommé `OCR1AL` en `REGISTRE_PWM_R`, `OCR1BL` en `REGISTRE_PWM_V` et `OCR0` en `REGISTRE_PWM_B`. Ceci est possible à condition d'avertir le cross-compileur CAVR par les lignes ci-dessous :

```
#define REGISTRE_PWM_R OCR1AL
#define REGISTRE_PWM_V OCR1BL
#define REGISTRE_PWM_B OCR0
```

A placer en tête du fichier, après les directives `#include`.

C) Programmation du composant

(1) Configurez le projet

- > Project
- > Configure
 - > Sélectionnez l'onglet "After Make"
 - > Cochez "Program the Chip"
 - > ok

(2) Programmez le composant

- > Icône "Make the Project"
- > "Program"



D) Test du programme

D1) Test sans la carte interface de puissance

Branchez un oscilloscope entre la douille PWMA de la carte SSI et le 0V, la douille PWMB et le 0V puis la douille PB3 et le 0V. Vous devez obtenir des signaux TOR dont le rapport cyclique varie par pas de 10%. Voir les exemples en annexe 3.

Appel prof

D2) Test avec la carte « LED haute luminosité »

Pour connecter la carte « LED haute luminosité » à la carte ATMEL SSI.

Appel prof

E) Synthèse : Algorithme « Dimming » destiné au projecteur à réaliser

La luminosité des LED du projecteur à réaliser ne sera pas réglée par des boutons-poussoirs mais par des **valeurs numériques lues sur la liaison DMX512**.

On note DMX_R, DMX_V et DMX_B les variables recevant ces valeurs.

Q6) Sachant que ces variables sont codées sur un octet et que les rapports cycliques Alpha_i doivent être proportionnels aux valeurs DMX_i, **modifiez l'algorithme** de l'annexe 2 pour qu'il réponde aux contraintes du projecteur. **Répondez ci-dessous.**

Remarque : $i = R, V$ ou B

Algorithme Dimming

```
// Variables
```

DMX_R, DMX_V et DMX_B, REGISTRE_PWM_R, REGISTRE_PWM_V, REGISTRE_PWM_ : *octets non signés*

```
// Constantes
```

```
Table_Alpha(11)={0,      } // Tableau d'octet
```

début

Répéter (toujours) _____

début // Répéter

```
// Lire
```

```
Lire(DMX_R, DMX_V, DMX_B) ; // Cette fonction vous sera donnée
```

```
// Traiter
```

[illegible]

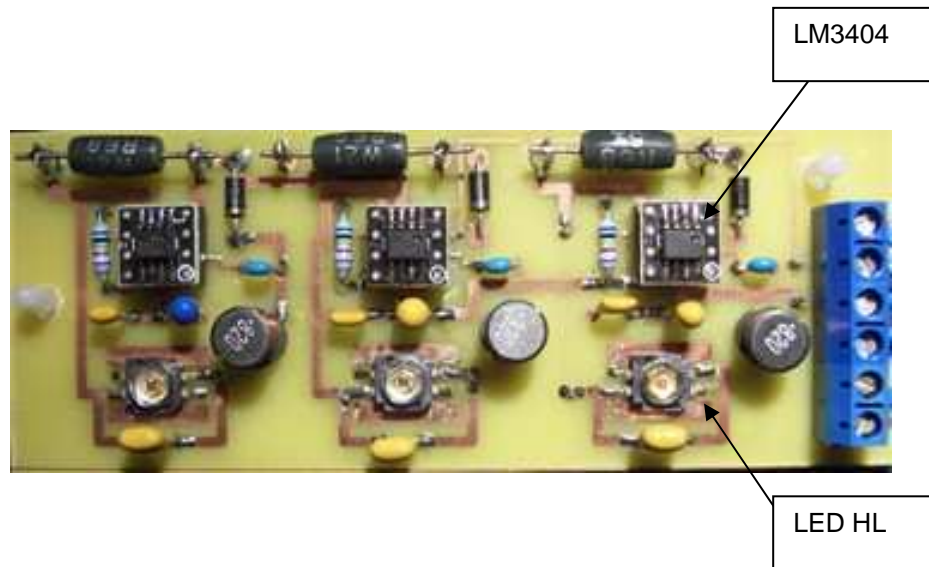
Appel prof

Pour faire vérifier l'algorithme

Q7) Comment doit-on modifier l'algorithme précédent pour que la luminosité varie par pas $< 1\%$? Quelle est la valeur du pas minimum ?



Carte d'étude



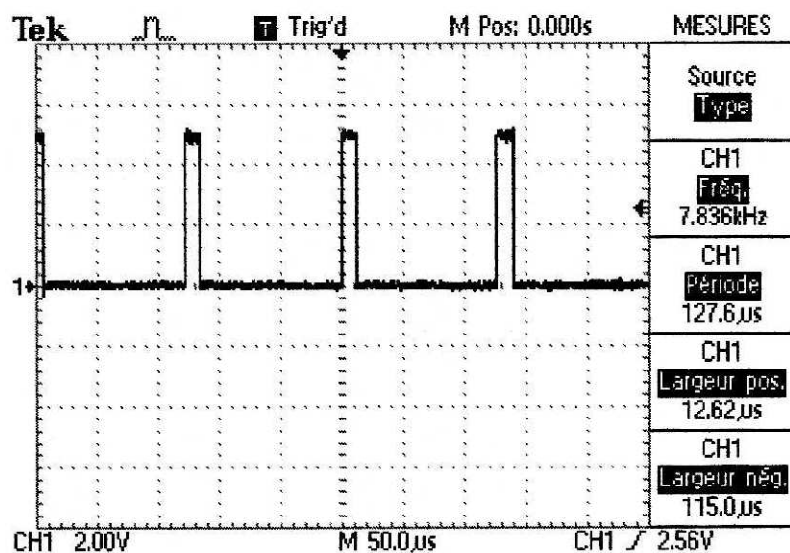
Carte dans son boîtier de protection



Annexe 2 : Programme C proposé et algorithme correspondant à compléter

Programme C (partiel)	Algorithme correspondant
<pre>while (1) { // début while // -----Lire les consignes de rapport cyclique----- BP = Lire_BP(); // ----- Traiter ----- switch(BP) { case INC: if (Alpha_R < 90) Alpha_R = Alpha_R+10; else Alpha_R = 100; break; case DEC: if (Alpha_R > 10) Alpha_R = Alpha_R-10; else Alpha_R = 0; break; case OK: if (Alpha_V < 90) Alpha_V = Alpha_V+10; else Alpha_V = 100; break; case SET: if (Alpha_V > 10) Alpha_V = Alpha_V-10; else Alpha_V = 0; break; case ECHAP : if (Alpha_B < 90) Alpha_B = Alpha_B+10; else Alpha_B = 100; break; case ENTR : if (Alpha_B > 10) Alpha_B = Alpha_B-10; else Alpha_B = 0; break; } // ----- Ecrire ----- REGISTRE_PWM_R = Table_Alpha[Alpha_R/10]; REGISTRE_PWM_V = Table_Alpha[Alpha_V/10]; REGISTRE_PWM_B = Table_Alpha[Alpha_B/10]; //On met à jour l'affichage si la valeur du rapport cyclique a été modifiée if ((Alpha_R != Alpha_R_1) (Alpha_V != Alpha_V_1) (Alpha_B != Alpha_B_1)) { sprintf(display_buffer_ligne0,"Rouge Vert Bleu"); sprintf(display_buffer_lignel,"%-u%% %-u%% %-u%%",R,V,B); Affiche_LCD(display_buffer_ligne0,display_buffer_lignel); Alpha_R_1 = Alpha_R; Alpha_V_1 = Alpha_V; Alpha_B_1 = Alpha_B; } // -----</pre>	<p>Répéter (toujours)</p> <p>début // Répéter Lire(BP) ; _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____</p> <p>Ecrire(Rapports cycliques) ;</p> <p>fin //Répéter</p>

Rapport cyclique $\alpha = 10\%$



Rapport cyclique $\alpha = 60\%$

