DSPIC30F3011

PWM "push-pull" avec les structures Output Compare

Cette application est utilisée dans le projet "VAE" (Vélo à Assistance Électrique) pour piloter le convertisseur DC-DC de l'éclairage. Le module "moteur" du dsPIC est utilisé pour piloter le moteur brushless et est donc indisponible pour la réalisation de ce PWM "push-pull".

1. Étude du convertisseur DC-DC "push-pull"

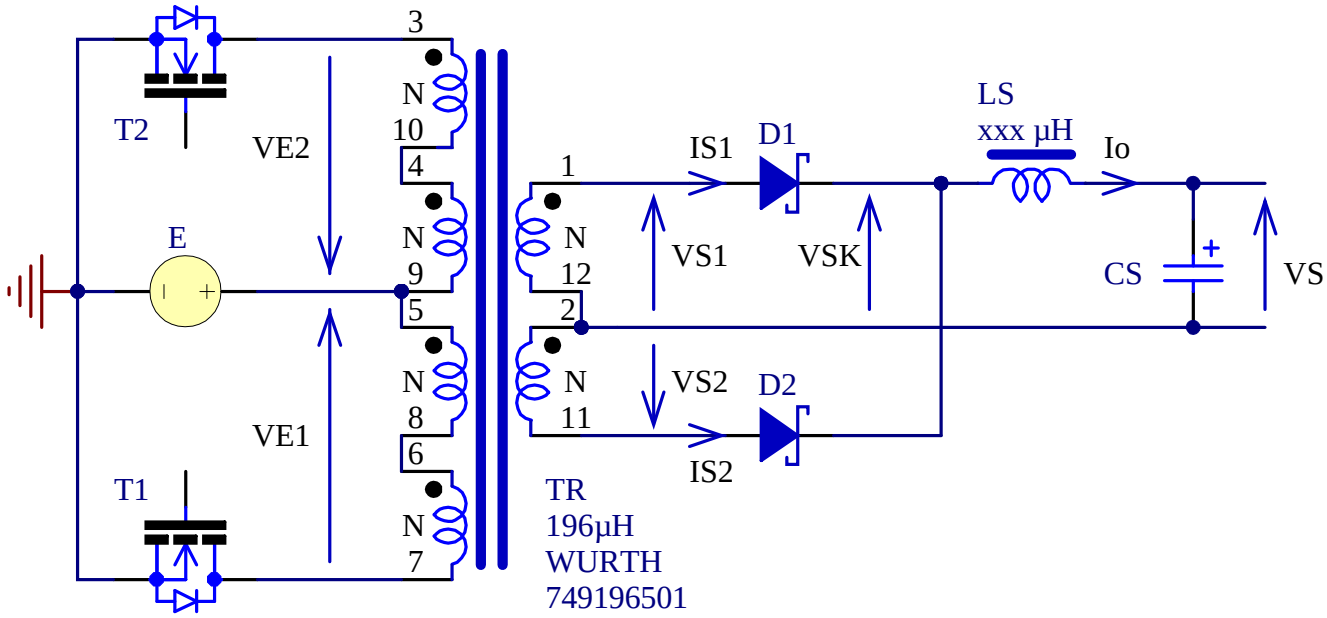
1.1 Cahier des charges

- **Entrée** : batterie "Ni-Mh", "Li-Ion" ou "Li-Po" de 36V à 72V nominal
- **Sortie** : tension continue au choix : 6V ou 12V, ondulations inférieures à 1V, puissance max : 10W
- Isolation galvanique
- Protégée contre les courts-circuits

1.2 Schéma simplifié

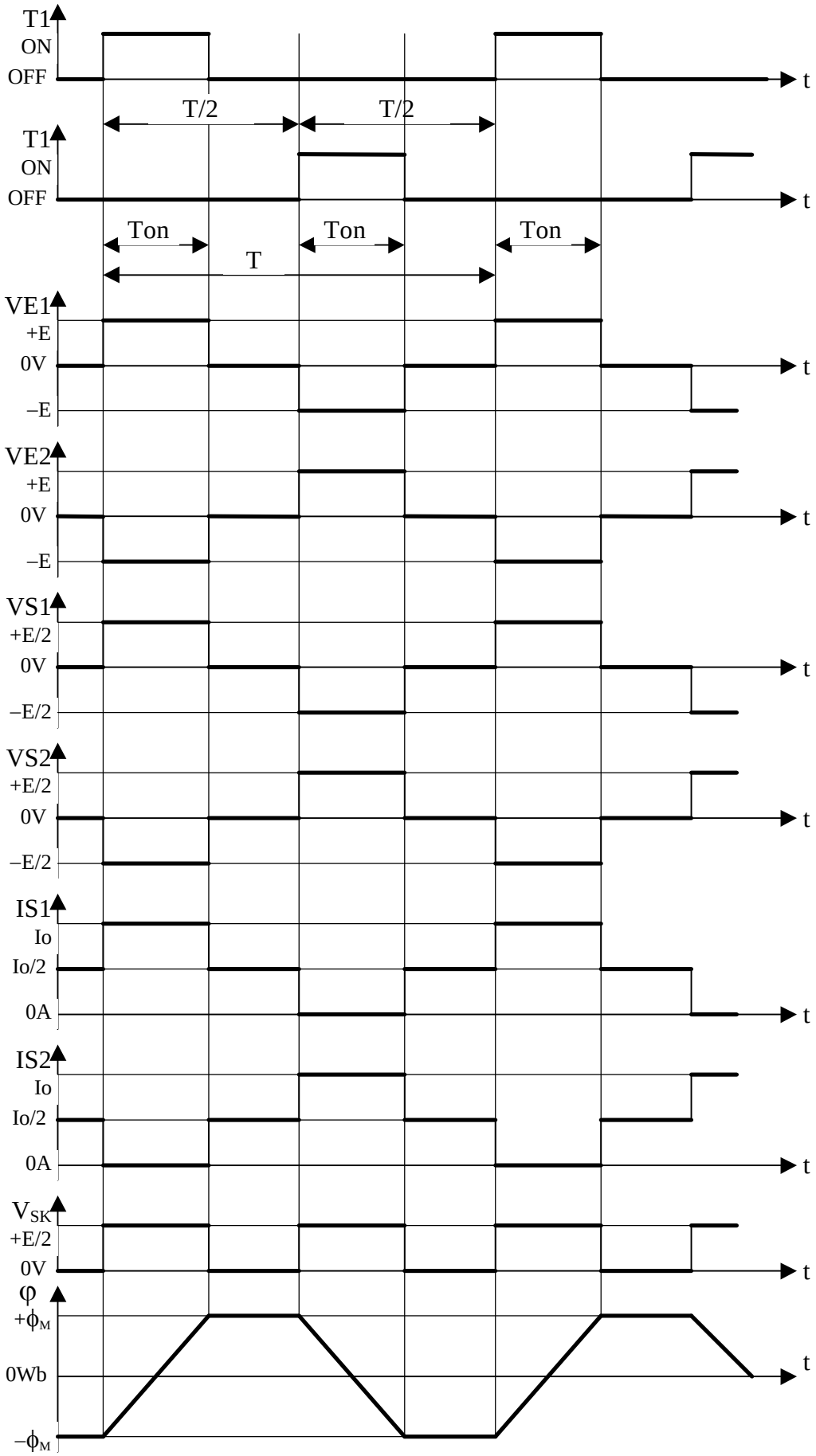
L'utilisation d'un transformateur s'impose pour respecter l'isolation galvanique. On choisit une commande symétrique pour minimiser la taille du transformateur.

→ La structure "**push-pull**" s'impose de fait.



Le transformateur représenté est celui choisi pour l'application VAE. Il s'agit d'un transformateur "universel" fabriqué en série (donc de coût faible) et comportant 6 bobinages identiques que l'on peut interconnecter à volonté (série et/ou parallèle). Le modèle utilisé ne comporte pas d'entrefer compte tenu de son mode de fonctionnement (pas d'accumulation d'énergie dans le noyau).

1.3 Chronogrammes typiques (avec des composants parfaits) :



Note : φ = flux magnétique dans le noyau du transformateur

Si les composants sont parfaits :

$$\boxed{V_{Smoy} = E \times 2 \times \frac{N2}{N1} \times \frac{T_{on}}{T}}$$ avec N1=2.N et N2=N (transfo Würth)

Pour le projet VAE, on a :

- V_E = 32V à 82V
- Transformateur : modèle 749196501 de Würth :
  - 6 bobinages indépendants de 196µH
  - N1 = 2.N et N2 = N $\Rightarrow$ N1/N2 = 2

La durée T_{on} doit être pilotée par le logiciel pour obtenir la tension de sortie choisie (6V ou 12V) pour toutes valeurs de V_E comprises entre 32V et 82V, soit :

$\frac{T_{on}}{T} = \frac{6}{82} = 7{,}3\%$ pour E = 82V et V_{Smoy} = 6V

$\frac{T_{on}}{T} = \frac{12}{32} = 37{,}5\%$ pour E = 32V et V_{Smoy} = 12V

Ces rapports cycliques seront un peu plus élevés dans le cas réel pour compenser les chutes de tension diverses (les diodes en particulier). **Attention** : on ne peut pas dépasser 50% avec un "push-pull".

1.4 Choix de la période T

Le transformateur impose le choix de la période T.

Pour limiter les pertes dans le transformateur et obtenir un bon rendement du convertisseur DC-DC, l'induction magnétique dans le transformateur doit rester notablement inférieure au seuil de saturation B_{max}.

Ce paramètre n'est pas fourni par le fabricant Würth, mais il en donne un autre qui lui est lié : $\int Udt$

Sa signification semble assez floue au premier abord. En fait, il permet justement de déterminer très rapidement la durée Ton à ne pas dépasser (et par conséquent la période T) :

- quand T1 est conducteur la f.é.m. E est appliquée entre les broches 5 et 7, soit V_{E1}=E
- le flux φ dans le transformateur est lié à V_{E1} par la relation : $V_{E1} = N1\frac{d\varphi}{dt}$
- V_{E1} étant constante pendant Ton, le flux φ s'accroît linéairement de $-\Phi_M$ à $+\Phi_M$ (voir chronogramme)
- on en déduit : $\Delta\Phi = 2 \times \Phi_M = \frac{E \times T_{on}}{N1}$ (intégration sur Ton, d'où la notation $\int Udt$)
- et pour un seul bobinage (transfo Würth) : $\Delta\Phi = 2 \times \Phi_M = \frac{\frac{E}{2} \times T_{on}}{N}$ car N1 = 2.N
- le paramètre $\int Udt$ donné par Würth est alors : $\boxed{\int Udt = 2 \times N \times \Phi_M}$ (pour un bobinage)
- soit $\boxed{\int Udt = \frac{E}{2} \times T_{on}}$

Mais cette relation n'est pas vraiment utilisable, car E varie dans de fortes proportions.

Par contre, la tension de sortie moyenne est connue et la relation qui lie V_{Smoy} et E permet d'écrire :

$$E \times T_{on} = V_{Smoy} \times \frac{1}{2} \times \frac{N1}{N2} \times T$$

Soit : $\int Udt = \frac{1}{2} \times V_{Smoy} \times \frac{1}{2} \times \frac{N1}{N2} \times T = \frac{1}{2} \times V_{Smoy} \times T$ car N1=2.N2

Et finalement : $\boxed{\boxed{T = \frac{2 \times \int Udt}{V_{Smoy}}}}$

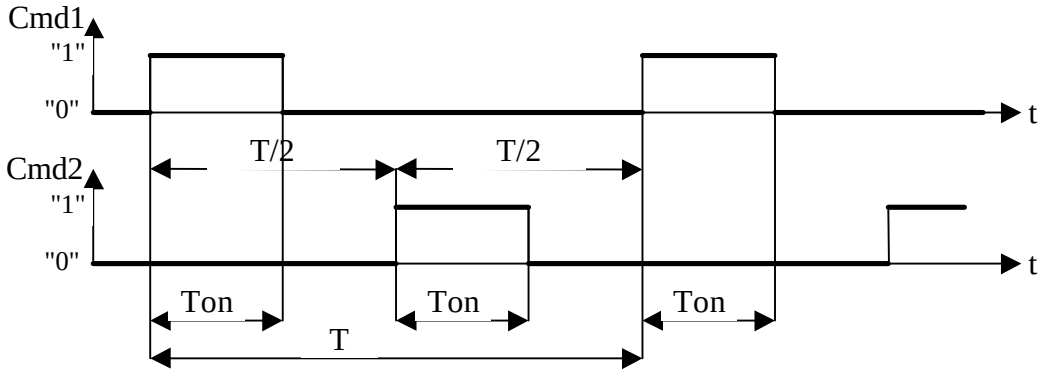
Le cas le plus défavorable ($\Delta\Phi$ maximum, donc $\int Udt$ max) est obtenu quant Ton est maximum (soit Ton = T/2), donc quand V_{Smoy} est régulée à 12V (V_{Smoy} est proportionnelle à Ton).

Il faudra donc respecter $T < \frac{2 \times 98{,}4 \times 10^{-6}}{12} = 16{,}4\mu S$ pour le transformateur 749196501 de Würth.

2. Production des signaux de commande

Les transistors T1 et T2 doivent être commandés pour que leurs états respectent les chronogrammes du §1.3. Avec des MOS, il faut produire 2 signaux logiques avec la même chronologie.

2.1 Caractéristiques des signaux de commande



- Période T : < 16,4µS (voir ci-dessus)
- Durée Ton : 0µS à T/2. En fait, la valeur T/2 ne sera jamais atteinte pour respecter des temps morts minimum pour T1 et T2 (T1 et T2 bloqués). On trouvera plus de détails à ce sujet dans le paragraphe des relevés.
- **Important** : les durées Ton des 2 signaux "Cmd1" et "Cmd2" doivent être absolument égales pour éviter une dérive de la composante continue du champ magnétique et saturer rapidement le noyau (le champ magnétique est proportionnel à l'*intégrale* de la tension appliquée).

2.2 Configuration du microcontroleur dsPIC

Le périphérique le plus adapté serait le module PWM à haute résolution. Mais dans l'application VAE, il est déjà utilisé pour commander le moteur brushless triphasé.
On utilise alors 2 structures "**Output Compare**" (sur 4 disponibles dans dsPIC30F3011), en l'occurrence OC3 et OC4.

2.2.1 Fonction d'initialisation "InitPushPull"

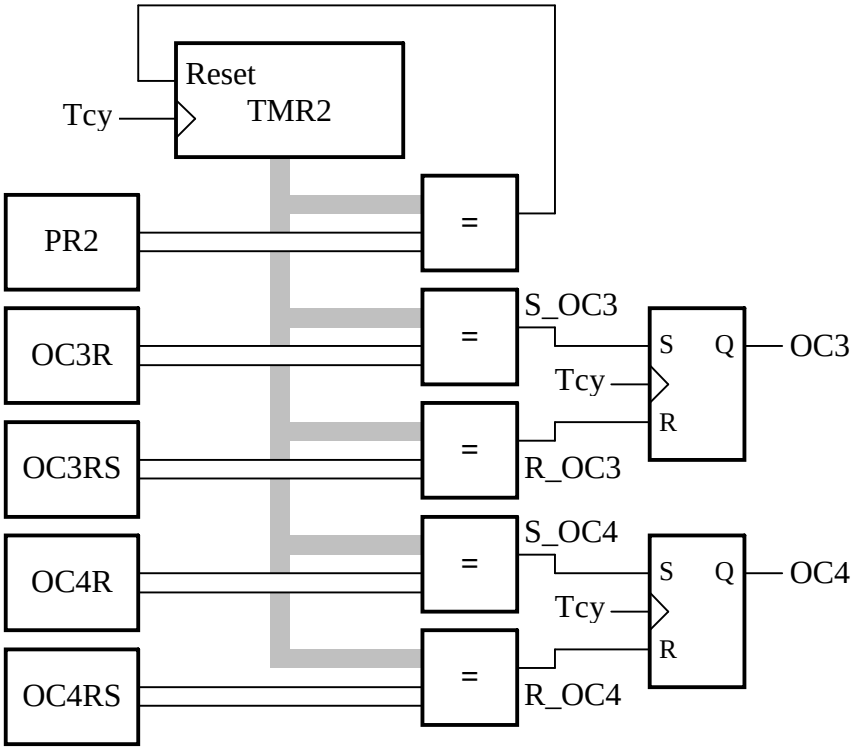
```
/*****************************************************************
  Function:    void InitPushPull(void)
  Description: Initialisation OC3 et OC4 pour piloter push pull éclairage
    - Période = 16µS
    - Rapport cyclique : 0%=0 50%=Pediode_PushPull/2
    - Protection par l'entrée d'interruption INT2 (active aux flancs descend.)
      -> sorties mises en Hiz (il faut des résistances "pull-up" ou "down")
*****************************************************************/
void InitPushPull(void)
{
 int i;
 PWM_PushPull=PWM_PP_Delay=2;// Rapport cyclique initial presque nul
 T2CON=0x8000; // Fonctionnement continu, horloge=TCY=16MHz,
               // compteur validé
 PR2=Periode_PushPull-1; // Période PWM = Periode_PushPullxTcy
 TMR2=-20;      // Pour produire les premiers changements d'états ci-dessous
 // Rapport cyclique initial sur OC3 presque nul
 //OC3RS=0;           // Sortie inversée
 //OC3R=PWM_PP_Delay; // Sortie inversée
 OC3R=0;             // Sortie non inversée
 OC3RS=PWM_PP_Delay; // Sortie non inversée
 // Seuils pour produire la 1° impulsion sur OC4 ci-dessous
 //OC4RS=30; // Sortie inversée
 //OC4R=32;  // Sortie inversée
```

```
 OC4R=30;  // Sortie non inversée
 OC4RS=32; // Sortie non inversée
 OC3CON=0x0005; // OCSIDL=0 : fonctionnement validé en mode "Idle"
                // OCFLT=0  : sélection "Timer 2"
                // OCM=5    : mode : chgt d'états de OC3 avec les 2 comparat.
 for (i=0;i<3;i++);//Petit retard pour laisser la 1° imp. sur OC3 se produire
 OC4CON=0x0005; // OCSIDL=0 : fonctionnement validé en mode "Idle"
                // OCFLT=0  : sélection "Timer 2"
                // OCM=5    : mode : chgt d'états de OC4 avec les 2 comparat.
 for (i=0;i<10;i++);//Petit retard pour laisser la 1° imp. sur OC4 se produire
 //Rapport cyclique initial sur OC4 presque nul
 //OC4RS=Periode_PushPull/2;             // Sortie inversée
 //OC4R=Periode_PushPull/2+PWM_PP_Delay; // Sortie inversée
 OC4R=Periode_PushPull/2;               // Sortie inversée
 OC4RS=Periode_PushPull/2+PWM_PP_Delay; // Sortie inversée
// Initialisation interruption INT2
 INTCON2bits.INT2EP=1; // Interruption aux flancs descendants
 IFS1bits.INT2IF=0;    // Raz de l'indicateur d'interruption
 IPC5bits.INT2IP=7;    // Priorité maximum
}
```

La fonction est assez complexe, son algorithme permet d'éviter la présence d'impulsions qui pourraient créer un dysfonctionnement. Le plus important est d'éviter la conduction simultanée de T1 et T2, même de façon transitoire, pour éviter un court circuit de la batterie.

Pour bien comprendre cette fonction et la production des signaux de commande, il est utile de dessiner un schéma simplifié des structures OC3 et OC4 configurées par "InitPushPull".

2.2.2 Schéma simplifié des structures OC3 et OC4 configurées par "InitPushPull"



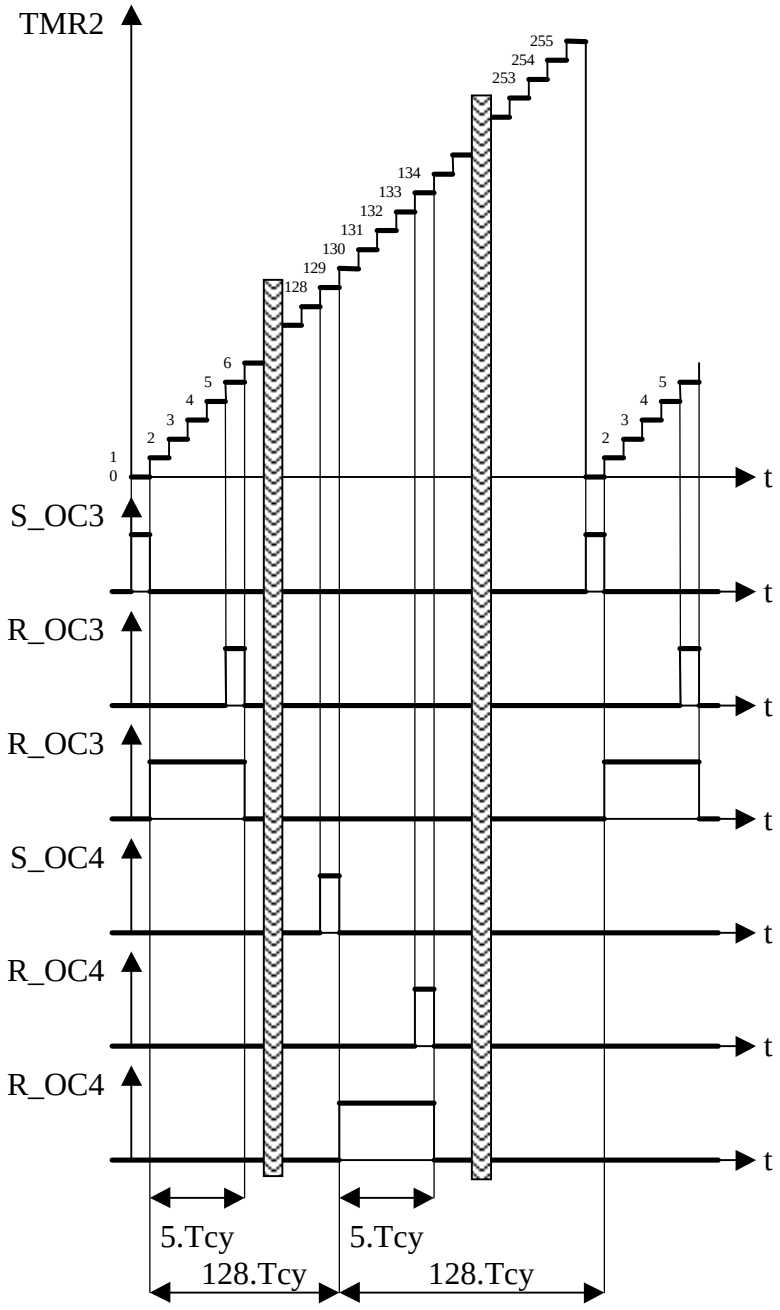
- Tcy est l'horloge système, soit Fcy=16MHz dans l'application VAE
- TMR2 est le compteur 16 bits synchrone du "Timer 2". L'entrée "Reset" est synchrone.
- PR2 est le registre "modulo" du "Timer 2" : le compteur TMR2 est remis à 0 quand son état atteint PR2.
- OC3R et OC3RS sont les registres de comparaison de la structure "Output Compare" OC3.
- OC4R et OC4RS sont les registres de comparaison de la structure "Output Compare" OC4.
- Leurs contenus sont constamment comparés à l'état de TMR2 :
  - S_OC3 est à "1" si TMR2=OC3R et à "0" dans le cas contraire
  - R_OC3 est à "1" si TMR2=OC3RS et à "0" dans le cas contraire
  - S_OC4 est à "1" si TMR2=OC4R et à "0" dans le cas contraire

  - R_OC4 est à "1" si TMR2=OC4RS et à "0" dans le cas contraire
- Ces signaux pilotent 4 bascules RS synchrones pour produire les signaux OC3 et OC4.

Chronogrammes typiques :

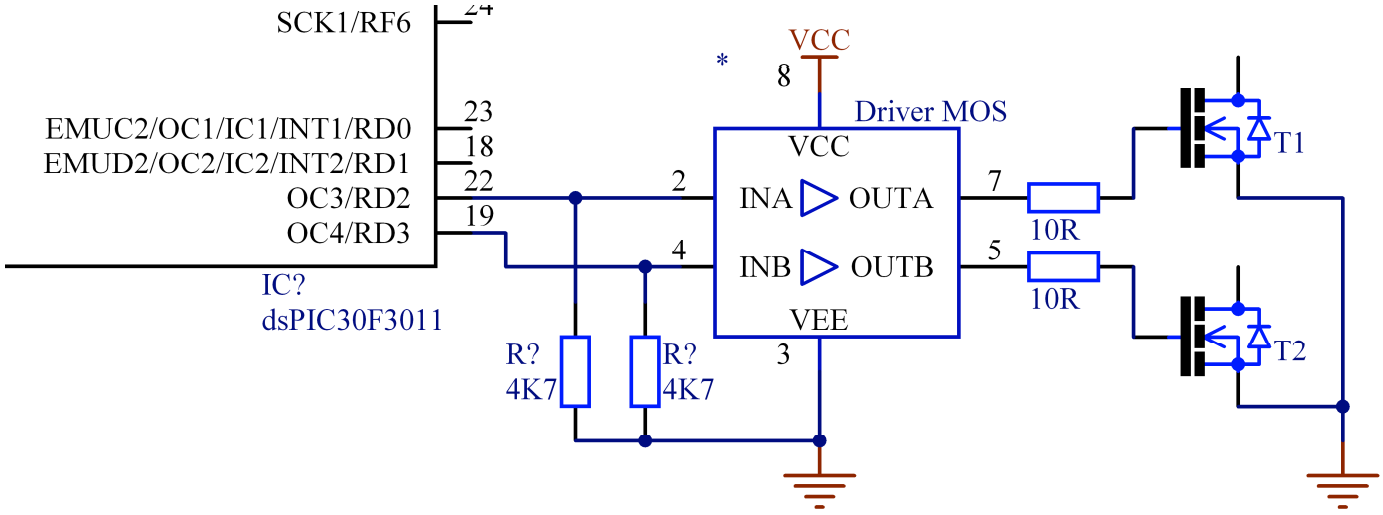
Pour ces chronogrammes, les registres sont affectés comme suit :

- PR2 = 255 pour obtenir une période T de 16µS avec Fcy = 16MHz
- OC3R = 0
- OC3RS = 5 pour obtenir Ton = 312,5nS (rapport cyclique de 0,3125/16 = 2%) sur OC3
- OC4R = 128 pour que OC4 passe à "1" à T/2
- OC4RS = 128+5 = 133 pour obtenir Ton = 312,5nS (rapport cyclique de 0,3125/16 = 2%) sur OC4



2.2.3 Déroulement de la fonction "InitPushPull"

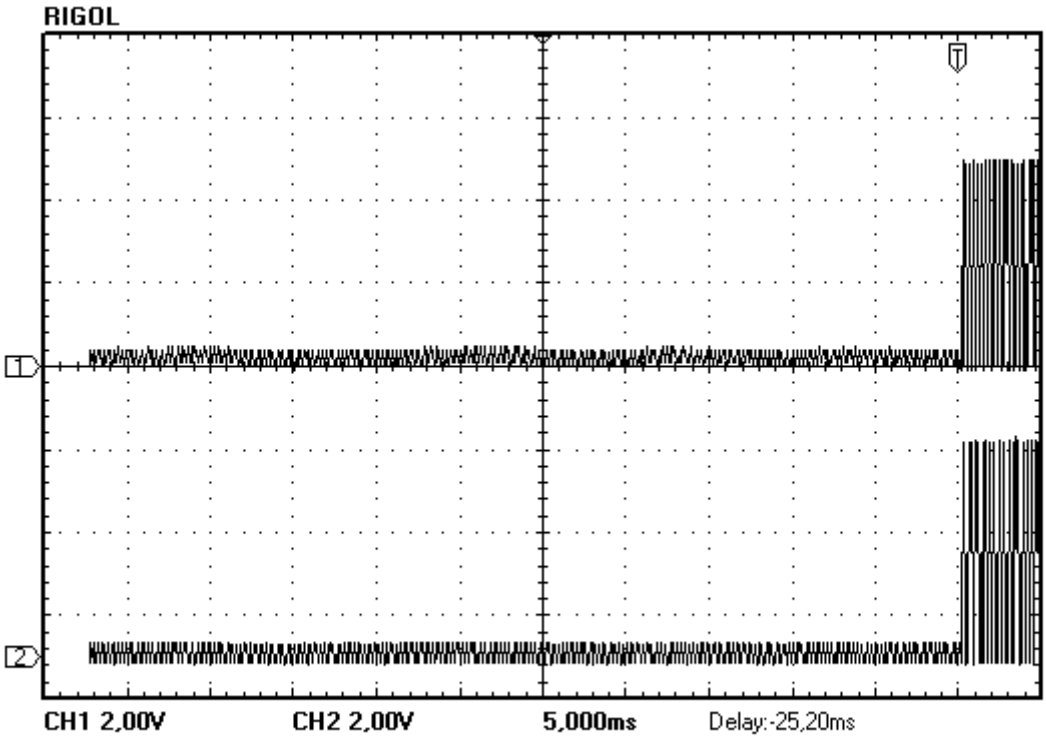
Les broches affectées aux sorties OC3 et OC4 ne sont pas encore configurées au "reset" du dsPIC. Ce sont encore des ports "I/O" classiques configurés en "entrée" dont les états sont "flottants".
Pour éviter la conduction simultanée des transistors T1 et T2 (court circuit de la batterie !), il faut absolument placer des résistances qui vont fixer l'état de OC3 et OC4 pendant la phase d'initialisation du dsPIC.
Si on utilise des "driver" de MOS non-inverseurs, cet état doit être "0" : les résistances sont alors de type "pull-down" (reliées à 0V) :



Dans ces conditions, les niveaux logiques sur OC3=RD2 et OC4=RD3 sont à "0" au début de l'exécution de la fonction "InitPushPull" :

- La fonction commence par configurer le "Timer 2" (registres T2CON et PR2) de façon à ce que le compteur TMR2 s'incrémente à chaque période Tcy.
- Le compteur TMR2 est alors affecté à –20 (ou 0xFFEC) pour que son passage par 0 se produise 20 cycles Tcy plus tard, soit après les affectations des registres OC3R, OC3RS et OC3CON (vérifié par simulation sur MPLAB)
- Les registres OC3R et OC3RS sont affectés avec les valeurs 0 et 2 pour produire une courte impulsion de durée 2xTcy sur OC3
- Le registre OC3CON est affecté pour configurer la structure "Output Compare 3" et affecter la sortie OC3 sur la broche correspondante.
- Un petit retard logiciel attend la production de cette impulsion (TMR2 > 2)
- La fonction affecte alors OC4R et OC4RS avec les valeurs 30 et 32 (avant que TMR2 n'atteigne la valeur 30) pour produire une courte impulsion de durée 2xTcy sur OC4
- Un nouveau petit retard logiciel attend la production de cette impulsion (TMR2 > 32)
- La fonction affecte alors les registres OC avec leurs valeurs finales pour obtenir un rapport cyclique pratiquement nul (2/256=0,78%)

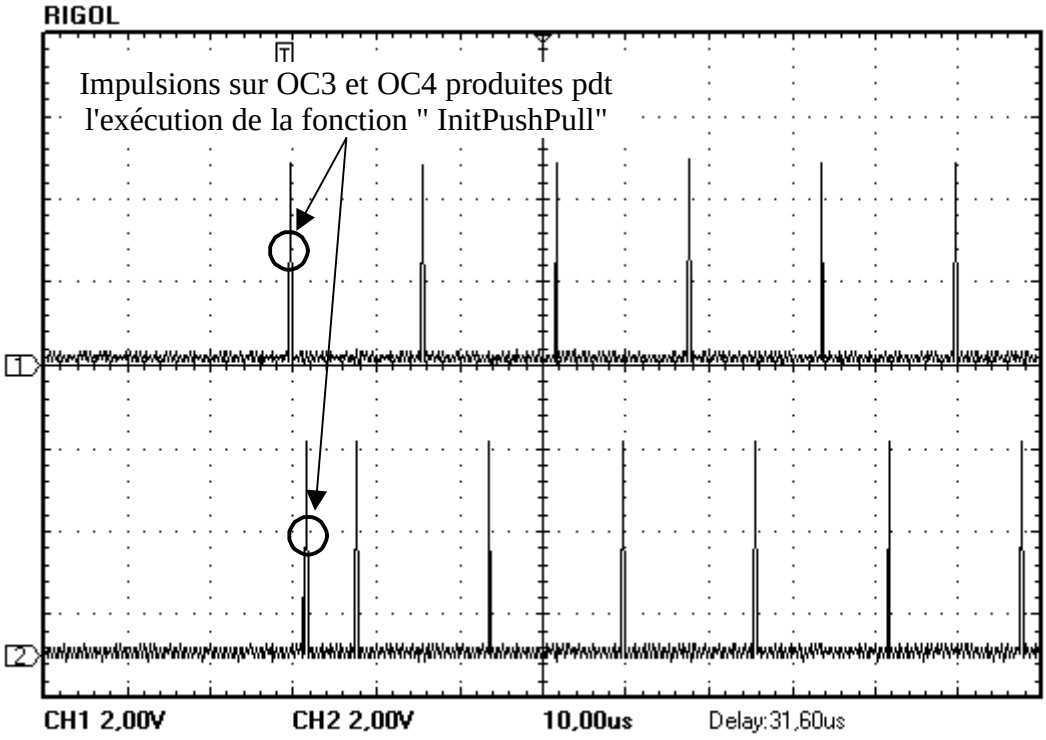
2.2.4 Relevés expérimentaux



OC3

OC4

On constate que les sorties OC3 et OC4 restent bien à "0" pendant la phase d'initialisation à la mise sous tension.

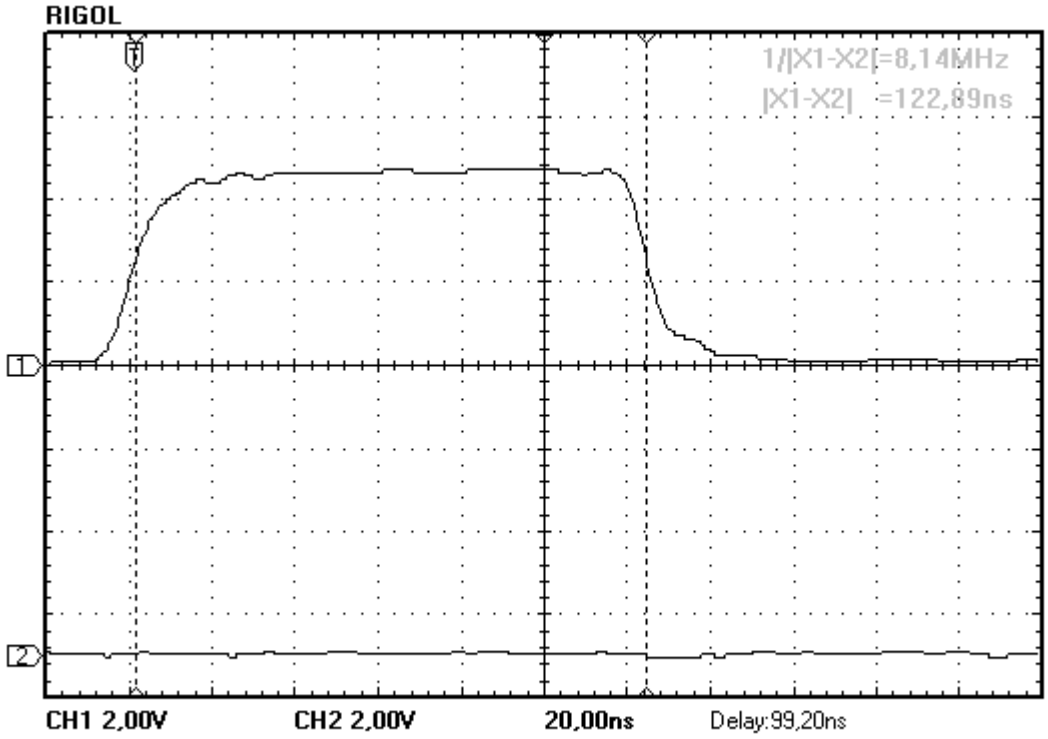


OC3

OC4

Un "zoom" sur les premières impulsions montre le bon déroulement de la fonction "InitPushPull". Celle-ci force les premières impulsions sur OC3 et OC4 pour éviter un aléa où elles seraient présentes simultanément.
Les impulsions suivantes sont produites conformément à l'analyse du paragraphe 2.2.2.

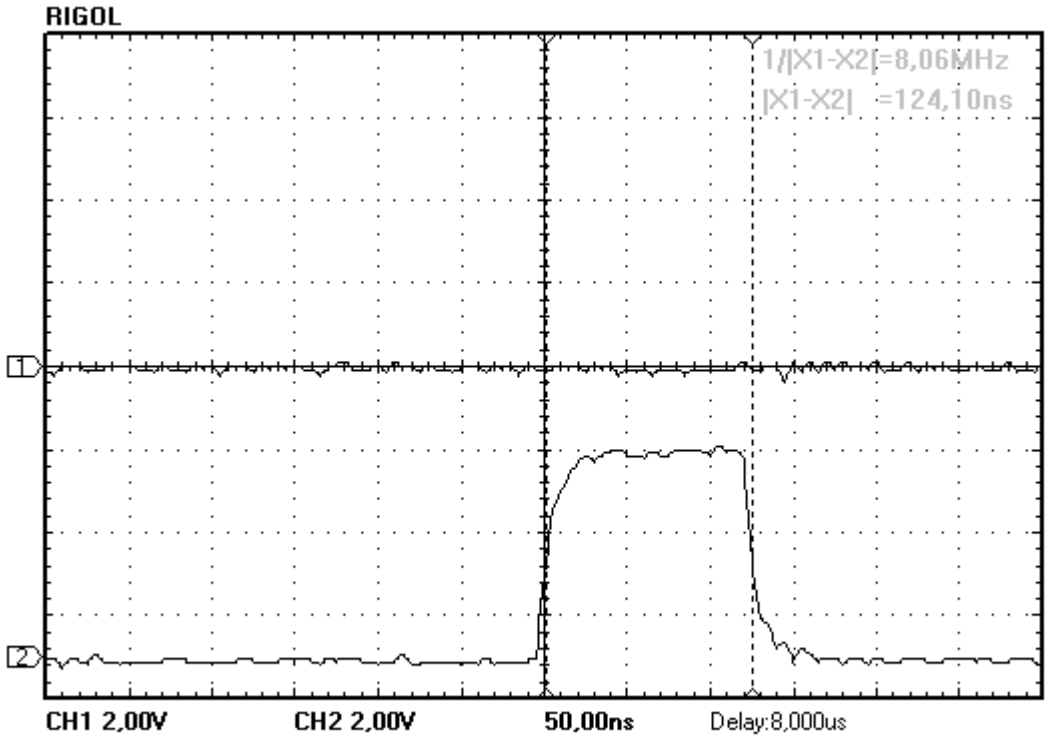
Un dernier relevé permet de vérifier la période et la durée des impulsions en régime établi :



OC3

OC4

La fréquence mesurée est de 65,500kHz (fonction "compteur" du RIGOL) et l'impulsion dure 123nS : les résultats sont conformes (avec OC3RS=2, soit 2 périodes Tcy).



OC3

OC4

Le délai entre les flancs montants de OC3 et OC4 est de 8,000µS : résultat conforme (demi-période T). La durée de l'impulsion l'est également (égale à celle sur OC3).

3. Régulation de la tension de sortie du convertisseur

Le principe de base consiste à prélever une image de la tension de sortie, de la comparer à une consigne et de commander le rapport cyclique pour rattraper tout écart. Cette solution a été rejetée pour les raisons suivantes :

- nécessité d'une isolation galvanique dans la boucle de retour (prélèvement de Vs),
- les performances de régulation exigées ne sont pas très élevées (il s'agit de l'éclairage du vélo)

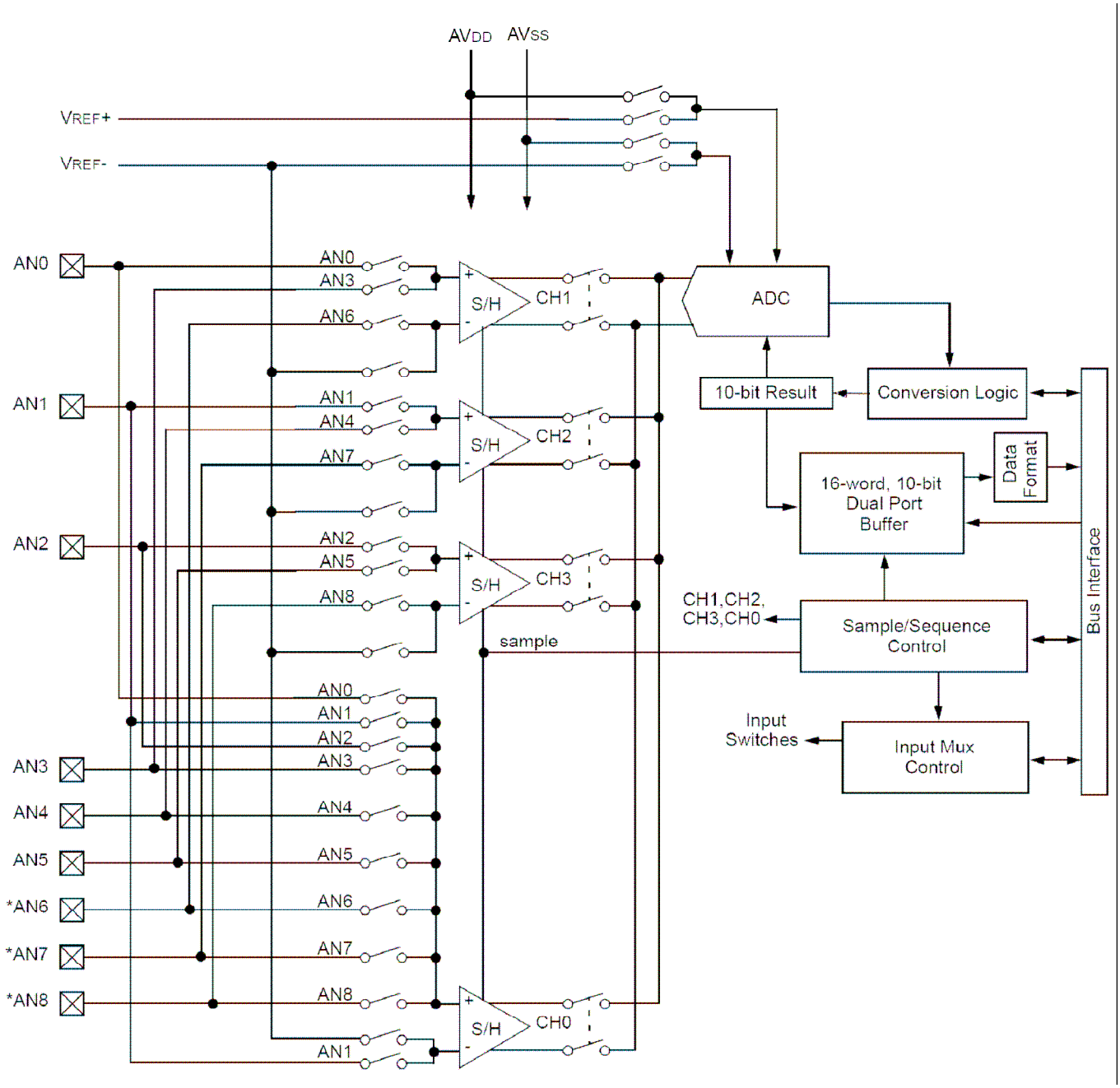
Par ailleurs, la relation qui lie la tension de batterie, la tension de sortie et le rapport cyclique est bien connue (voir §1.3). Le principe de régulation retenu est alors le suivant :

- mesure de la tension batterie (cette fonction existe déjà dans le VAE, donc pas d'augmentation du coût)
- calcul du rapport cyclique (en fait la durée Ton) pour obtenir la tension de sortie choisie (6V ou 12V) en fonction de la tension batterie avec la formule du §1.3.

De plus, pour les tests, on utilise la poignée de "gaz" pour commander l'éclairage; le seuil "OFF/ON" étant placé approximativement au milieu de la course.
La poignée de gaz est reliée sur l'entrée AN7 et produit une tension comprise entre 0,87V et 4,28V.

3.1 Conversion A/N

On utilise bien entendu le CAN 10 bits du dsPIC :



La structure comporte tous les éléments nécessaires à la conversion (sauf une vraie source de référence) :

- un multiplexeur analogique pour choisir la source de référence (AVDD ou externe)
- un multiplexeur pour choisir l'entrée à convertir (l'échantillonnage peut être simultané pour 4 entrées)
- un CAN rapide (jusqu'à 1M ech/s)
- un registre FIFO à double accès de profondeur 16 mots.

Tous ces éléments sont configurés via des registres 16 bits. On utilise la configuration de l'application VAE qui exploite toutes les entrées analogiques disponibles, sauf AN5. La fonction de contrôle de l'éclairage n'en utilise que 2 : VBAT et POT_Pos.

Register 17-5: ADPCFG: A/D Port Configuration Register

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PCFG15	PCFG14	PCFG13	PCFG12	PCFG11	PCFG10	PCFG9	PCFG8
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PCFG7	PCFG6	PCFG5	PCFG4	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

ADPCFG = `0x01DF` : AN0 à AN8 utilisées (sauf AN5)

Register 17-1: ADCON1: A/D Control Register 1

Upper Byte:							
R/W-0	U-0	R/W-0	U-0	U-0	U-0	R/W-0	R/W-0
ADON	—	ADSIDL	—	—	—	FORM<1:0>	
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0 HC, HS	R/C-0 HC, HS
SSRC<2:0>			—	SIMSAM	ASAM	SAMP	DONE
bit 7							bit 0

ADON = 1 : convertisseur opérationnel à la fin de la configuration
ADSIDL = 0 : module opérationnel en mode "idle"
FORM = 0 : résultat codé en binaire décalé (Bd)
SSRC = 3 : conversion déclenchée par le module PWM
SIMSAM = 1 : échantillonnage simultané de CH0, CH1, CH2 et CH3
ASAM = 1 : échantillonnage dès que la phase de conversion est terminée
SAMP = 0 :
DONE : indicateur de fin de conversion

ADCON1 = `0x006C` (ADON est mis à "1" à la fin de la configuration)

Register 17-2: ADCON2: A/D Control Register 2

Upper Byte:							
R/W-0	R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0
VCFG<2:0>			reserved	—	CSCNA	CHPS<1:0>	
bit 15							bit 8

Lower Byte:							
R-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BUFS	—	SMPI<3:0>				BUFM	ALTS
bit 7							bit 0

VCFG = 0 : VREF_H=AVDD et VREF_L=AVSS
CSCNA = 0 : pas de "scan" des entrées
CHPS = 2 : échantillonnage simultané de CH0, CH1, CH2 et CH3 (voir schéma précédent)
BUFS = 0 : sans effet car BUFM = 0
SMPI = 0 : interruption après chaque échantillonnage de CH0, CH1, CH2 et CH3

BUFM = 0 : registre FIFO configuré en mots de 16 bits
ALTS = 0 : utilisation de la configuration "MUX A"
ADCON2 = `0x0200`

Register 17-3: ADCON3: A/D Control Register 3

Upper Byte:							
U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	SAMC<4:0>				
bit 15							bit 8

Lower Byte:							
R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADRC	—	ADCS<5:0>					
bit 7							bit 0

SAMC = 0 : sans effet dans ce mode de fct
ADRC = 0 : horloge CAN issu de l'horloge système
ADCS = 5 : TAD = 3.Tcy = 187,5nS (conversion en 5x0,1875+4x12x0,1875=9,9375µS)
ADCON3 = `0x0005`

Register 17-4: ADCHS: A/D Input Select Register

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CH123NB<1:0>		CH123SB	CH0NB	CH0SB<3:0>			
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CH123NA<1:0>		CH123SA	CH0NA	CH0SA<3:0>			
bit 7							bit 0

CH123NB = 0: sans effet ici ("MUXB")
CH123SB = 0 : sans effet ici ("MUXB")
CH0NB = 0 : sans effet ici ("MUXB")
CH0SB = 0 : sans effet ici ("MUXB")
CH123NA = 0: entrée - de CH1, CH2, CH3 = VREF-
CH123SA = 0 : entrée + de CH1 = AN0 = IMOT_A
CH2 = AN1 = IMOT_B
CH3 = AN2 = IBAT
CH0NA = 0 : entrée négative de CH0 = VREF-
CH0SA = 7 : entrée + de CH0 = AN7 = poignée de gaz
ADCHS = `0x0007`

Register 17-6: ADCSSL: A/D Input Scan Select Register

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CSSL15	CSSL14	CSSL13	CSSL12	CSSL11	CSSL10	CSSL9	CSSL8
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CSSL7	CSSL6	CSSL5	CSSL4	CSSL3	CSSL2	CSSL1	CSSL0
bit 7							bit 0

Registre sans effet ici.

3.1.1 Fonction d'initialisation du CAN 10 bits

```
/*********************************************************************
  Function:        void InitADC10(void)
  Description :    Initialisation de l'ADC 10 bits :
                   1. AN6 = VBAT
                      AN7 = poignée d'accélération
                   2. Conversion déclenchée par PWM (toutes les 16 périodes)
                   3. Interruption EOC validée
*********************************************************************/
void InitADC10(void)
{
 ADPCFG=~0x01DF;// AN0 à AN8 utilisées (sauf AN5)
 ADCON1=0x006C; // Codage en Bd sur 10 bits
                // Déclenché par module PWM
                // Echantillonnage simultané de CH0, CH1, CH2 et CH3
                // entre 2 cycles de conversion
 ADCON2=0x0200; // VREF+=AVDD et VREF-=AVSS
                // CHPS<1:0>=10 : conversion de CH0, CH1, CH2 et CH3
                // SMPI<3:0>=0 : interruption après l'échantillonnage et les
                //               4 conversions de CH0, CH1, CH2 et CH3
                // Buffer 16 bits
 ADCON3=0x0005; // SAMC<4:0>=0 : sans effet
                // ADRC=0 -> Horloge système
                // ADCS<5:0>=5 -> Tad=3Tcy soit 187,5nS
                // Conversion en 5x0,1875+4x12x0,1875=9,9375µS
 ADCHS=0x0007;  // Entrée - de CH1, CH2, CH3 = VREF-
                // Entrée + de CH1 = AN0 = IMOT_A
                //             CH2 = AN1 = IMOT_B
                //             CH3 = AN2 = IBAT
                // Entrée - de CH0 = VREF-
                // Entrée + de CH0 = AN7 = poignée
 IFS0bits.ADIF=0;  // Clear ISR flag
 IEC0bits.ADIE=1;  // Enable interrupts
 ADCON1bits.ADON=1;// Turn ADC ON
}
```

Une interruption est provoquée après les 4 conversions de CH0, CH1, CH2 et CH3, donc à chaque échantillonnage. Les canaux CH1, CH2 et CH3 ne sont pas utilisés pour le contrôle de l'éclairage; on les ignore.

Le canal CH0 correspond à la voie sélectionnée par le multiplexeur analogique (AN3 à AN8 pour un dsPIC30F3011). Le multiplexeur analogique est commandé par les bits "CH0SA" du registre "ADCHS". Les 2 entrées analogiques utilisées par la fonction d'éclairage sont AN6 (VBAT) et AN7 (poignée de gaz). Le résultat de la conversion du canal CH0 est placé dans le registre ADCBUF0.

La fonction "_ADCInterrupt" correspondante à cette interruption affecte la bonne variable avec le résultat de la conversion et gère le multiplexeur analogique pour sélectionner la prochaine entrée analogique à convertir.

3.1.2 Fonction d'initialisation du module moteur PWM

Ce module n'est pas utilisé pour piloter le convertisseur DC-DC de l'éclairage, mais il déclenche chaque conversion du CAN 10 bits.
La fonction reproduite ci-dessous est une version simplifiée de celle de l'application VAE où seuls sont affectés les registres permettant le déclenchement du CAN 10 bits.

```
/*****************************************************************
  Function:        void InitMCPWM(void)
  Description:     Initialisation des 3 modules PWM comme suit :
                   1. FPWM = 20000 Hz
                   2. ADC déclenché par le "PWM special trigger"
*****************************************************************/
void InitMCPWM(void)
{
 PTPER=PTPERI;   // Moitié de la période PWM
 PWMCON2=0x0302; // Postscale = 4 : déclenchement du CAN toutes les 4 PWM
                 // Rafraichissement PWM synchrones période PWM
                 // Rafraichissement validé
 PTCON=0xA002;   // PWM validé en mode "centré"
                 // La base de temps PWM s'arrête en mode "veille"
}
```

Le registre "PWMCON2" est affecté pour que le module moteur déclenche le CAN 10 bits toutes les 4 périodes PWM "moteur", soit à la fréquence de 5kHz. Cette fréquence d'échantillonnage est largement suffisante pour convertir l'image de la tension batterie et la position de la poignée. Une fréquence trop élevée aurait provoqué un rythme d'interruptions inutilement élevé (il faut limiter l'occupation du CPU).

3.1.3 Fonction d'interruption "_ADCInterrupt"

Cette fonction est activée après la conversion des 4 canaux CH0, CH1, CH2 et CH3.
Il s'agit d'une version simplifiée de celle de l'application VAE dans laquelle on ne convertit que les 2 entrées analogiques associées au pilotage de l'éclairage : la tension "batterie" et la poignée de gaz (ces 2 entrées sont "multiplexées" vers le canal CH0).
La fonction "_ADCInterrupt" :

- affecte la bonne variable avec le résultat de la conversion (disponible dans ADCBUF0),
- et gère le multiplexeur analogique pour sélectionner la prochaine entrée analogique à convertir.

```
/*****************************************************************
  Function    : void _ISR _ADCInterrupt (void)
  Description : Programme d'interruption déclenché à chaque EOC du CAN 10 bits
  Résultat des conversions :
      - de AN6 rangé dans VBAT en Bd
      - de AN7 rangé dans POT_Pos en Bd
*****************************************************************/
void _ISR _ADCInterrupt (void)
{
 TEST1=1;
 IFS0bits.ADIF=0; // Raz indicateur interruption
 switch (ADCHSbits.CH0SA)
  {
   case 6 : VBAT=ADCBUF0;      // Mesure tension DC de la batterie
            ADCHSbits.CH0SA=7; // Prochaine conversion : POT_Pos
            break;
   case 7 : POT_Pos=ADCBUF0;   // Lecture position poignée
            ADCHSbits.CH0SA=6; // Prochaine conversion : VBAT
            break;
   default: break;
  }
 TEST1=0;
}
```

Pour mesurer la fréquence d'activation de ce programme (qui est le 1/4 de la fréquence d'échantillonnage), on ajoute les instructions "TEST1= 1;" au début et "TEST1=0;" à la fin de la fonction.

3.2 Commande du rapport cyclique

Le rapport cyclique des signaux OC3 et OC4 doit être rafraîchi régulièrement pour réguler la tension d'éclairage en fonction de la tension de la batterie. Un rythme supérieur à 10 rafraîchissements par seconde est largement suffisant.
Dans l'application VAE, le "Timer 1" est utilisé pour produire des interruptions RTI au rythme de 60 par seconde. La fonction d'interruption correspondante ("_T1Interrupt") est donc chargée du rafraîchissement du rapport cyclique.

3.2.1 Fonction d'initialisation du "Timer 1"

```
/*****************************************************************
  Function    : void InitTMR1(void)
  Description : Initialisation du Timer 1 (Timer A) pour provoquer
                une interruption à rythme régulier (RTI).
  Note : le rythme est choisi pour obtenir une fréquence des transmissions
         des paramètres sur TXD de 10Hz
*****************************************************************/
void InitTMR1(void)
{
 T1CON=0x0010;   // Fonctionnement continu, horloge=Fcy/8=2MHz
 TMR1=0;         // Raz du compteur pour décaler la 1° interruption
 PR1=(FCY/8)/(10*NbDtaTxd); // Fréquence RTI = 10Hz x NbDtaTxd
 T1CONbits.TON=1;// Validation timer 1
 IFS0bits.T1IF=0;// Raz indicateur Timer 1
 IEC0bits.T1IE=1;// Validation interruption Timer 1 (RTI)
}
```

3.2.2 Fonction d'interruption "_T1Interrupt"

Cette fonction est appelée à chaque boucle du compteur TMR1 du "Timer 1", soit "10*NbDtaTxd" fois par seconde. La constante "NbDtaTxd" vaut 6 dans la version actuelle de l'application VAE : le rythme RTI est donc de 60 appels par seconde.
Par ailleurs, cette fonction va donner une certaine inertie à la commande d'éclairage : le rapport cyclique n'est pas affecté immédiatement avec la valeur nominale mais l'atteindra progressivement. Cette fonction de service ne sera peut-être pas retenue dans la version finale.
Pour ce faire, la fonction utilise 2 variables :

– "PWM_PushPull" : valeur finale du rapport cyclique (en fait Ton)
– "PWM_PP_Delay" : valeur actuelle de Ton qui évolue vers la valeur finale avec une pente déterminée.

Ces variables représentent en fait la durée Ton en nombre de cycles Tcy. Le rapport cycle est facile à déduire, la période T étant constante (voir §2.2).

```
/*****************************************************************
  Function    : void _ISR _T1Interrupt (void)
  Description : Fréquence = 10Hz x NbDtaTxd
  Traitement :
    - Affectation PWM éclairage avec inertie
*****************************************************************/
void _ISR _T1Interrupt(void)
{
 TEST0=1;
 IFS0bits.T1IF=0; // Acquittement interruption
 Compt_T1_TXD++;
 if (Compt_T1_TXD==NbDtaTxd)
  {
   Compt_T1_TXD=0; // Compt_T1_TXD modulo NbDtaTxd
   Calc_PWM_Ecl(); // Calcul de PWM_PushPull pour obtenir
                   // la tension d'éclairage voulue (V_ECL)
  }
 // Inertie du PWM de l'éclairage (pente = NbDtaTxd x 60/seconde)
```

```
 if (PWM_PushPull > PWM_PP_Delay) PWM_PP_Delay++;
 if (PWM_PushPull < PWM_PP_Delay) PWM_PP_Delay--;
 // Affectation des registres PWM
 //OC3R=PWM_PP_Delay;                    // Sortie inversée
 //OC4R=Periode_PushPull/2+PWM_PP_Delay; // Sortie inversée
 OC3RS=PWM_PP_Delay;                    // Sortie non inversée
 OC4RS=Periode_PushPull/2+PWM_PP_Delay; // Sortie non inversée
 TEST0=0;
}
```

Commentaires :

- La variable "Compt_T1_TXD" est incrémentée modulo "NbDtaTxd" (6) à chaque appel de la fonction.
- L'opération modulo est simplement obtenue en comparant la valeur de "Compt_T1_TXD" à "NbDtaTxd" juste après son incrémentation et en l'affectant à 0 en cas d'égalité.
- Cette mise à 0 est donc faite 10 fois pas seconde. Ce rythme est suffisant pour recalculer le nouveau rapport cyclique si la tension batterie a évolué : la mise à 0 est donc suivie de l'appel à la fonction "Calc_PWM_Ecl" qui calcule la nouvelle valeur de "PWM_PushPull" en utilisant la formule du §1.3 (voir aussi le §3.2.3 ci-dessous)
- La variable "PWM_PP_Delay" est incrémentée ou décrémentée à chaque interruption pour "rattraper" la variable "PWM_PushPull".
- Les registres OC3RS et OC4RS sont finalement affectés par la valeur actuelle de "PWM_PP_Delay" pour rafraîchir le rapport cyclique.
- Le signal TEST0 permet d'observer à l'oscilloscope les phases d'activité de la fonction "_T1Interrupt"

Note : la fonction "Calc_PWM_Ecl" nécessite beaucoup de temps CPU pour calculer la valeur "PWM_PushPull", en raison notamment d'une opération de division. C'est pourquoi elle n'est appelée que 10 fois par seconde et non à chaque interruption. Ce rythme est largement suffisant car la tension batterie varie relativement lentement.

3.2.3 Fonction "Calc_PWM_Ecl"

```
/*****************************************************************
  Function:     void Calc_PWM_Ecl(void)
  Description : Calcul du rapport cyclique PWM_PushPull pour obtenir
                la tension V_ECL en fonction de la tension VBAT
                Appelée régulièrement par "_T1Interrupt"
  Traitement  : PWM_PushPull=(PARAMCONT.V_ECL x Periode_PushPullxK_PWM)/(VBAT)
                Le résultat est limité en valeurs haute et basse pour obtenir
                des rapports cycliques convenables
*****************************************************************/
#define K_PWM 20
void Calc_PWM_Ecl(void)
{
 // Affectation des registres PWM de OC3 et OC4 avec inertie
 if (PilotCont.Eclairage) // Commande d'éclairage du TdB ?
  {// Oui : activer l'éclairage
   if (VBAT > 483) // Soit 24V
   {// VBAT > 24V : activation de l'éclairage
    PWM_PushPull=(unsigned int)(PARAMCONT.V_ECL*Periode_PushPull*K_PWM)/VBAT;
    if (PWM_PushPull>((Periode_PushPull/2)-2)) PWM_PushPull=(Periode_PushPull/2)-2;
    if (PWM_PushPull<2) PWM_PushPull=2;
    Flags.Eclairage_ON=1; // Eclairage ON
   }
   else
   {// VBAT < 24V : pas d'éclairage si la batterie est déchargée
    PWM_PushPull=2;       // Consigne de rapport cyclique presque nulle
    Flags.Eclairage_ON=0; // Eclairage OFF
   }
  }
 else
  {// Non : couper l'éclairage
   PWM_PushPull=2; // Consigne de rapport cyclique presque nulle
```

```
    Flags.Eclairage_ON=0; // Eclairage OFF
   }
}
```

Commentaires :

- Le champ "V_ECL" de la variable "PARAMCONT" est la consigne de tension d'éclairage : soit les valeurs 6 ou 12, pour 6V ou 12V.
- Le champ "Eclairage" de la variable "PilotCont" est la commande d'éclairage : 0 → éclairage coupé, ≠0 → éclairage opérationnel. Ce champ est affecté dans la boucle sans fin de la fonction "main".
- Si "PilotCont.Eclairage" = 0 : la consigne de rapport cyclique est affectée avec 2, soit un rapport cyclique de 2/256 = 0,78%. On évite la valeur 0 pour écarter tout risque d'aléas dans les structures "Output Compare". Cela pourra être modifié dans la version finale à l'issue des essais.
- Si "PilotCont.Eclairage" ≠ 0 :
  - Si la tension batterie est inférieure à 24V (batterie déchargée) : l'éclairage reste éteint
  - Dans le cas contraire, il faut respecter $T_{on}=T\times\frac{V_{Smoy}}{E}$ pour que V_{Smoy} ait la valeur voulue.

    La tension de batterie E est mesurée via un pont diviseur "2,4K-22K" par le CAN du dsPIC. On en déduit : $VBAT=E\times\frac{2,4}{22+2,4}\times\frac{1024}{AV_{DD}}=E\times 20,144$ et $T_{on}=T\times\frac{V_{Smoy}}{VBAT}\times 20,144$ avec $AV_{DD}=5V$

    Or $T_{on}=T_{CY}\times PWM\_PushPull$ et $T=T_{CY}\times Periode\_PushPull$

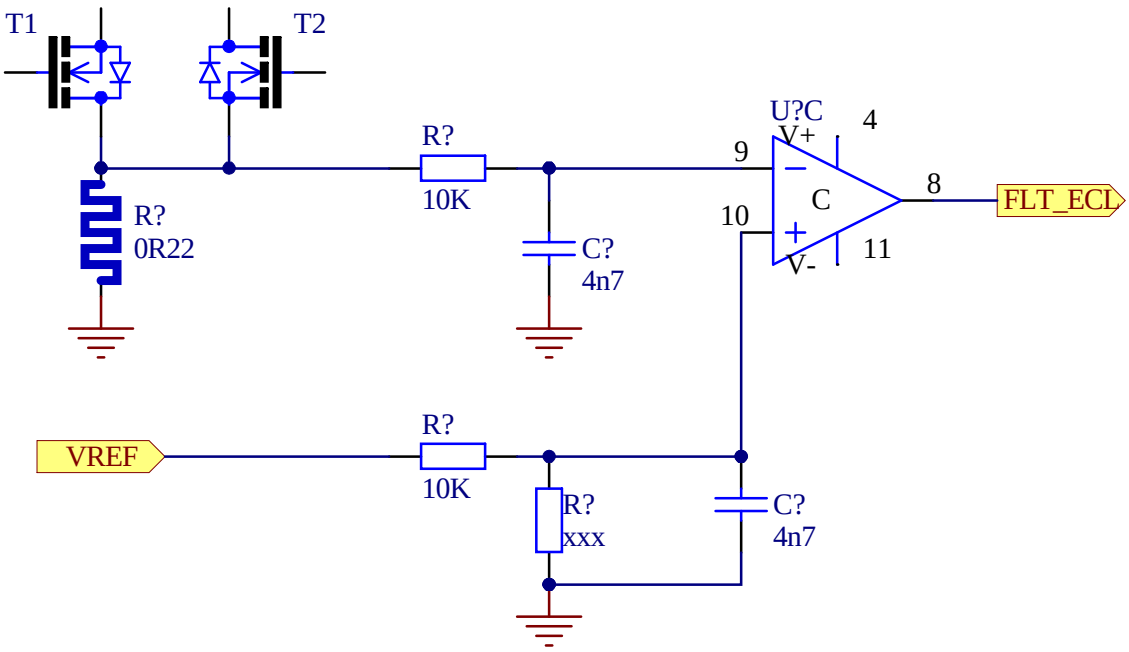
    soit $PWM\_PushPull=Periode\_PushPull\times\frac{V_{Smoy}}{VBAT}\times 20,144$

    On identifie la relation utilisée dans la fonction en notant que K_PWM vaut 20. C'est la valeur entière la plus proche, l'erreur est négligeable compte tenu de la précision recherchée.
  - La valeur de "PWM_PushPull" est alors limitée en valeurs haute et basse pour assurer un bon fonctionnement du convertisseur push-pull :
    - valeur mini = 2 pour écarter tout risque d'aléas dans les structures "Output Compare". Cela pourra être modifié dans la version finale à l'issue des essais
    - valeur max = "Periode_PushPull"-2 pour obtenir un temps mort minimum de 2 périodes Tcy.

4. Protection contre les surintensités

Cette fonction est indispensable pour prévenir une destruction de composant (en particulier les transistors du push-pull) et aussi stopper une destruction en chaîne en cas d'anomalie de fonctionnement. Celle-ci peut, par exemple, être provoquée par un court-circuit sur le câblage de l'éclairage.
La protection est réalisée par une structure mixte :

- **Matérielle** :



La résistance de 0,22Ω permet d'avoir une image du courant instantané dans les transistors du push-pull.
Le filtre RC qui suit atténue les pointes de courant très courtes et non dangereuses.
L'amplificateur opérationnel détecte toute surintensité en comparant l'image du courant à un seuil fixe.
Le signal logique "FLT_ECL" est mis à l'état bas en cas de surintensité.

– **Logicielle** : le signal "FLT_ECL" est relié à l'entrée INT2 du dsPIC. Cette entrée est configurée pour provoquer une interruption de priorité maximum aux flancs descendants.
La fonction correspondante est chargée de bloquer les 2 transistors du push-pull le plus rapidement possible, sans attendre la fin du cycle PWM en cours.

4.1 Configuration de l'interruption INT2

L'interruption INT2 est configurée à la fin de la fonction "InitPushPull" vue au §2.2.1

```
/*****************************************************************
  Function:    void InitPushPull(void)
  Description: Initialisation OC3 et OC4 pour piloter push pull éclairage
    - Période = 16µS
    - Rapport cyclique : 0%=0 50%=Pediode_PushPull/2
    - Protection par l'entrée d'interruption INT2 (active aux flancs descend.)
      -> sorties mises en Hiz (il faut des résistances "pull-up" ou "down")
*****************************************************************/
void InitPushPull(void)
{
 …
 …
// Initialisation interruption INT2
 INTCON2bits.INT2EP=1; // Interruption aux flancs descendants
 IFS1bits.INT2IF=0;    // Raz de l'indicateur d'interruption
 IPC5bits.INT2IP=7;    // Priorité maximum
}
```

Les commentaires dans le source sont suffisamment clairs.

4.2 Fonction d'interruption "_INT2Interrupt"

Cette fonction est chargée de bloquer les 2 transistors du push-pull le plus rapidement possible, sans attendre la fin du cycle PWM en cours.

```
/*****************************************************************
  Function:        void _ISR _INT2Interrupt (void)
  Description :    Programme d'interruption déclenché à l'activation de
                   l'entrée INT2, soit à la détection d'une surintensité
    - INT2 active aux flancs descendants
    - Sorties OC3 et OC4 mises en Hiz (il faut des résistances "pull-up")
*****************************************************************/
void _ISR _INT2Interrupt (void)
{
 OC3CON=0;             // Arrêt sortie PWM sur OC3 par Hiz
 OC4CON=0;             // Arrêt sortie PWM sur OC4 par Hiz
 Flags.Eclairage_ON=0; // Eclairage OFF
 IFS1bits.INT2IF=0;    // Raz de l'indicateur d'interruption
}
```

L'opération est simplement réalisée par un arrêt des fonctions OC3 et OC4. Les ports correspondants (RD2 et RD3) retrouvent leur configuration initiale : 2 entrées. Grâce aux résistances de "pull-down" (voir schéma du §2.2.3), les niveaux logiques de OC3/RD2 et OC4/RD3 passent immédiatement à "0".

Amélioration :

Cette fonction ne pourra pas être utilisée telle quelle dans l'application VAE. En effet, aucune procédure n'est prévue pour ré-initialiser les structures OC si la cause de la défaillance a disparu.

Il faudra prévoir 2 modes de protection :

– Lors des premières défaillances, l'éclairage est rétabli automatiquement après un court délai (1/10s par exemple)
– Si la défaillance persiste après plusieurs rétablissements (5 par exemple), la coupure devient définitive jusqu'à un réamorçage manuel de l'utilisateur.

5. Fonction "main"

```
                        /*****************************
                        *    Programme principal     *
                        *****************************/
int main (void)
{
 U2MODE=0;    // Inhiber l'UART à cause du bootloader
 TEST0_Dir=0; // Signal TEST0 en sortie
 TEST1_Dir=0; // Signal TEST1 en sortie
 InitTMR1();  // Initialisation Timer 1 pour RTI
 InitMCPWM(); // Initialisation PWM
InitADC10(); // Initialisation ADC 10 bits
 InitPushPull(); // Initialisat. des 2 O.C. utilisés pour piloter le push-pull
                 // de l'éclairage ainsi que le Timer 2
 // Interruption INT2 pour PWM éclairage
 IFS1bits.INT2IF=0; // Raz de l'indicateur d'interruption INT2
 IEC1bits.INT2IE=1; // Interruptions INT2 validées

 PARAMCONT.V_ECL=12; // Pour éclairage 12V

 while (1)
  {
   if (POT_Pos > 527) PilotCont.Eclairage=1;
                 else PilotCont.Eclairage=0;
  };
}
```

La fonction réalise :

- toutes les initialisations nécessaires au fonctionnement de l'éclairage,
- fixe la consigne d'éclairage à 12V,
- puis entre dans une boucle sans fin qui détermine simplement l'état de l'éclairage en fonction de la position de la poignée de gaz (le seuil 527 correspond environ au milieu de la course).

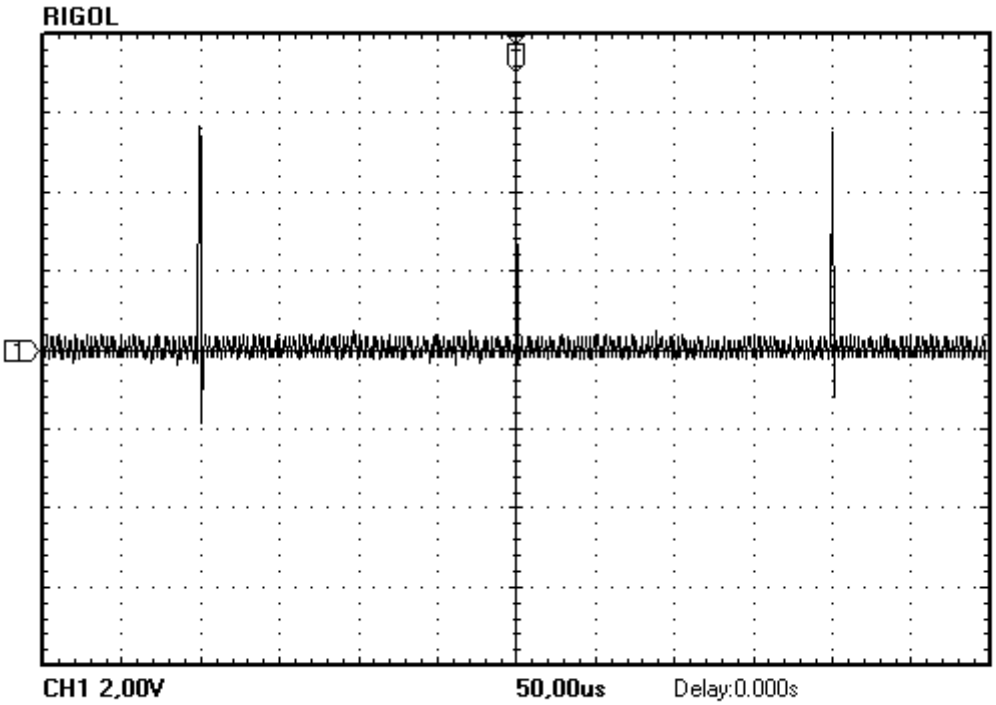
6. Relevés

La configuration des 2 structures "Output Compare" a déjà été validée au §2.2.4.
Il reste donc à vérifier :

- le bon fonctionnement du CAN 10 bits : fréquence d'échantillonnage et multiplexage
- la configuration du "Timer 1" et la fonction d'interruption "_T1Interrupt"
- la commande de l'éclairage avec la poignée de gaz
- les rapports cycliques min et max et les temps morts
- la fonction d'inertie sur la commande du rapport cyclique
- la fonction de régulation de la tension d'éclairage
- la fonction de protection

6.1 CAN 10 bits

La **fréquence d'échantillonnage** est simplement contrôlée en observant le signal TEST1 à l'oscilloscope. En effet, la fonction d'interruption "_ADCInterrupt" est appelée après la conversion des 4 canaux du CAN et elle comporte les 2 instructions qui provoquent une impulsion sur TEST1 (voir §3.1.3).

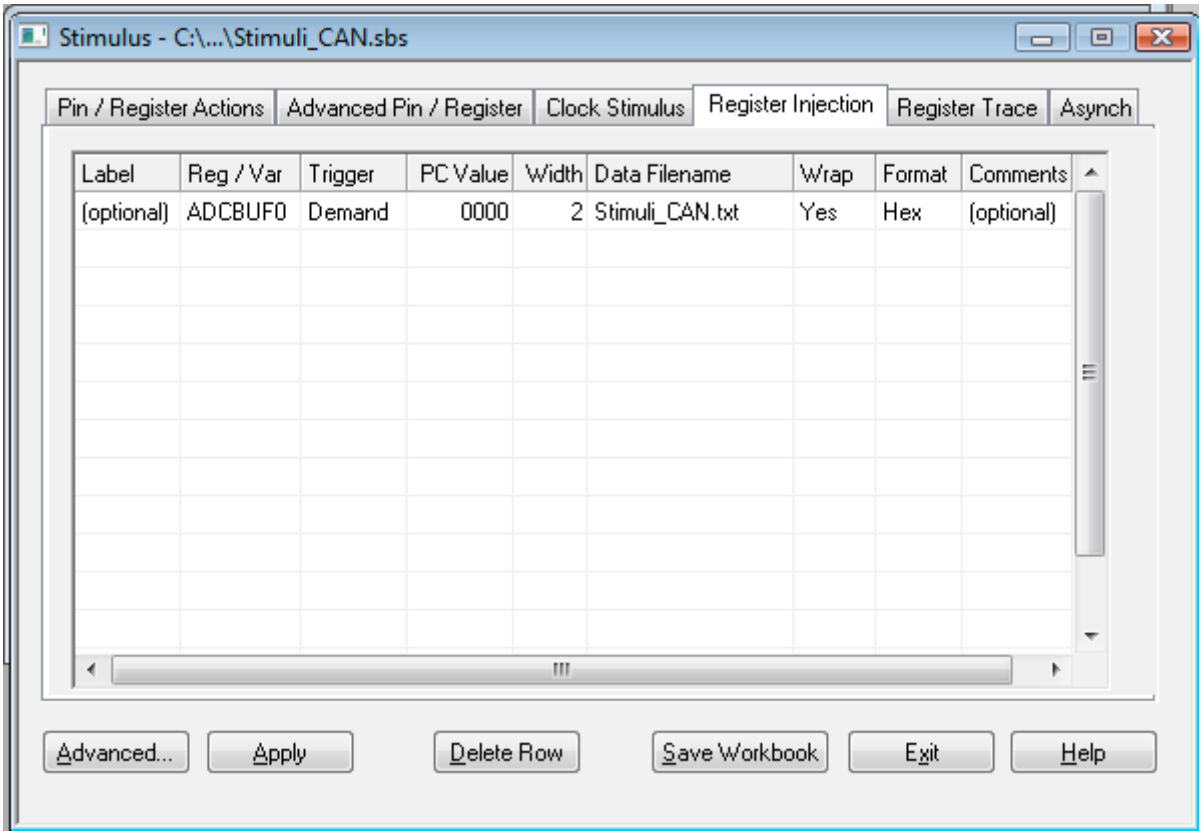


TEST1

La fréquence est mesurée avec la fonction "fréquencemètre" de l'oscilloscope RIGOL, donc avec une bonne précision (0,1%).
La valeur de 4,9912kHz est parfaitement correcte : elle correspond bien au 1/4 de la fréquence PWM des signaux du moteur, rapport choisi dans la fonction "InitMCPWM" ($F_{PWMmoteur}$ = 20kHz).

La **fonction multiplexage** est plus difficile à vérifier de façon expérimentale. On se contentera ici d'une simulation sous MPLAB. La confirmation expérimentale sera faite lors de la vérification de la fonction de régulation de la tension d'éclairage.
Configuration de MPLAB :

- Editer un fichier texte (par exemple "Stimuli_CAN.txt") avec une suite de valeurs (par exemple 10 20 30 40 etc.). Ces valeurs seront les résultats successifs du CAN 10 bits lors de la simulation : ADCBUF0 à ADCBUF3 sont affectés par les 4 valeurs suivantes à chaque interruption.
- Ouvrir une fenêtre de stimuli : "Debugger/Stimulus/New Workbook" :
  - Dans l'onglet "Register Injection" : associer le fichier "Stimuli_CAN.txt" au registre "ADCBUF0". Ce registre sera affecté avec les valeurs successives du fichier à chaque fin de conversion du CAN 10 bits.



– Cliquer sur "Apply" pour que cette association soit prise en compte

– Placer un point d'arrêt sur la première ligne de la fonction "_ADCInterrupt"

– Il suffit alors de lancer le programme et de vérifier l'affectation des bits "CH0SA" du registre "ADCHS" qui sélectionnent les entrées analogiques "VBAT" et de la poignée de gaz. Cela devrait affecter alternativement les variables "VBAT" et "POT_Pos" avec les valeurs du fichier de stimuli (1 sur 4).

6.2 Configuration du "Timer 1" et fonction "_T1Interrupt"

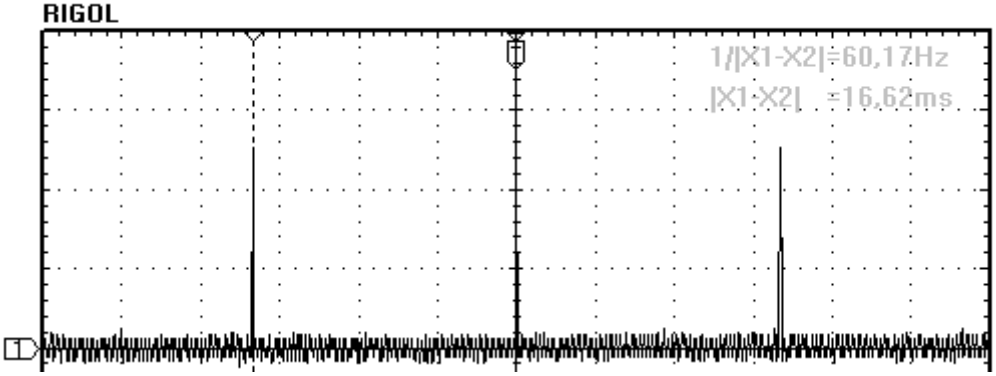
Il s'agit de vérifier la fréquence d'activation de la fonction "_T1Interrupt". Le signal TEST0 est affecté dans cette fonction dans ce but.

Ce test est réalisé pour 2 versions de "_T1Interrupt" :

– Version 1 :

```
void _ISR _T1Interrupt(void)
{
 TEST0=1;
 IFS0bits.T1IF=0; // Acquittement interruption
 Compt_T1_TXD++;
 if (Compt_T1_TXD==NbDtaTxd)
  {
   Compt_T1_TXD=0; // Compt_T1_TXD modulo NbDtaTxd
   etc.
```

La fréquence de TEST0 doit être de 60Hz (avec NbDtaTxd=6), ce qui est vérifié dans le relevé ci-dessous :

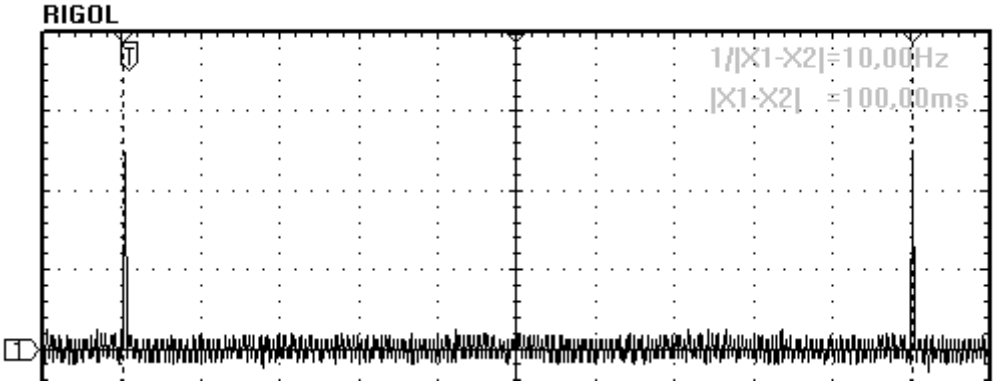


TEST0

– Version 2 :

```
void _ISR _T1Interrupt(void)
{
 IFS0bits.T1IF=0; // Acquittement interruption
 Compt_T1_TXD++;
 if (Compt_T1_TXD==NbDtaTxd)
  {
   TEST0=1;
   Compt_T1_TXD=0; // Compt_T1_TXD modulo NbDtaTxd
   etc.
```

On vérifie ici le bon fonctionnement du compteur "Compt_T1_TXD" modulo "NbDtaTxd". La fréquence de TEST0 doit être de 10Hz (avec NbDtaTxd=6), ce qui est vérifié dans le relevé ci-dessous :



TEST0

Les autres opérations de "_T1Interrupt" sont testées dans les paragraphes suivants.

6.3 Commande de l'éclairage

Cette opération est réalisée dans la boucle sans fin de la fonction "main".
La variable "POT_Pos" représente la position de la poignée de gaz. Celle-ci fournit une tension continue comprise entre 0,87V et 4,28V, ce qui correspond à des valeurs de "POT_Pos" comprises entre 178 et 876. Le seuil 527 correspond bien au milieu de la course.
La vérification expérimentale est réalisée en modifiant légèrement la boucle sans fin de la fonction "main" :

```
while (1)
  {
   if (POT_Pos > 527) {
                       PilotCont.Eclairage=1;
                       TEST1=1;
                      }
                 else {
                       PilotCont.Eclairage=0;
                       TEST1=0;
                      }
  };
```

Il faut bien entendu supprimer toutes les autres affectations de TEST1 dans le programme.

→ *Une observation de l'état de TEST1 confirme le bon fonctionnement de la commande d'éclairage : changement d'état au milieu de la course de la poignée, soit pour une tension de 2,56V (valeur théorique de "POT_Pot" = 1024*2,56/5 = 524 : CQFD)*

6.4 Les rapports cycliques min et max et les temps morts

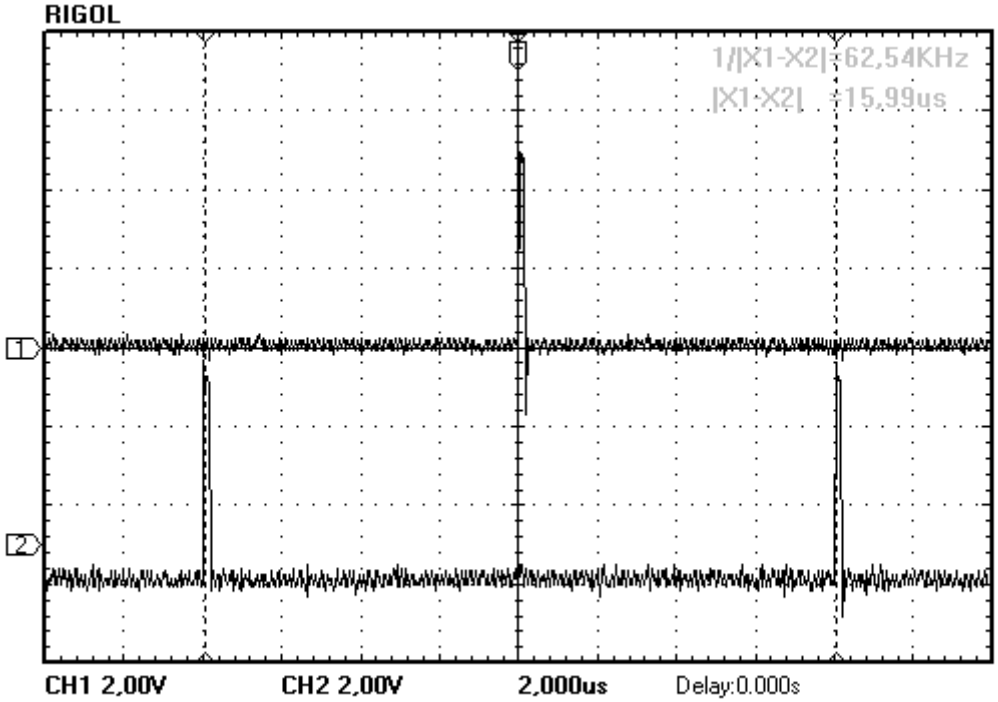
La fonction d'interruption est normalement chargée de calculer le rapport cyclique. Ce calcul est inhibé dans ce test pour affecter le rapport cyclique directement avec les valeurs min et max.

Fonction "_T1Interrupt" modifiée :

```
void _ISR _T1Interrupt(void)
{
 IFS0bits.T1IF=0; // Acquittement interruption
 Compt_T1_TXD++;
 if (Compt_T1_TXD==NbDtaTxd)
  {
   Compt_T1_TXD=0; // Compt_T1_TXD modulo NbDtaTxd
   PWM_PushPull=2; // Rapport cyclique minimum
   //PWM_PushPull=(Periode_PushPull/2)-2; // Rapport cyclique maximum
   //Calc_PWM_Ecl(); // Calcul de PWM_PushPull pour obtenir
                     // la tension d'éclairage voulue (V_ECL)
  }
 // Inertie du PWM de l'éclairage (pente = NbDtaTxd x 60/seconde)
 if (PWM_PushPull > PWM_PP_Delay) PWM_PP_Delay++;
 if (PWM_PushPull < PWM_PP_Delay) PWM_PP_Delay--;
 // Affectation des registres PWM
 //OC3R=PWM_PP_Delay;                    // Sortie inversée
 //OC4R=Periode_PushPull/2+PWM_PP_Delay; // Sortie inversée
 OC3RS=PWM_PP_Delay;                    // Sortie non inversée
 OC4RS=Periode_PushPull/2+PWM_PP_Delay; // Sortie non inversée
}
```

L'appel de la fonction "Calc_PWM_Ecl" est inhibé en plaçant la ligne en commentaires.
On ajoute alors la ligne "PWM_PushPull=2;" ou la ligne "PWM_PushPull=(Periode_PushPull/2)-2;" suivant le relevé à effectuer.

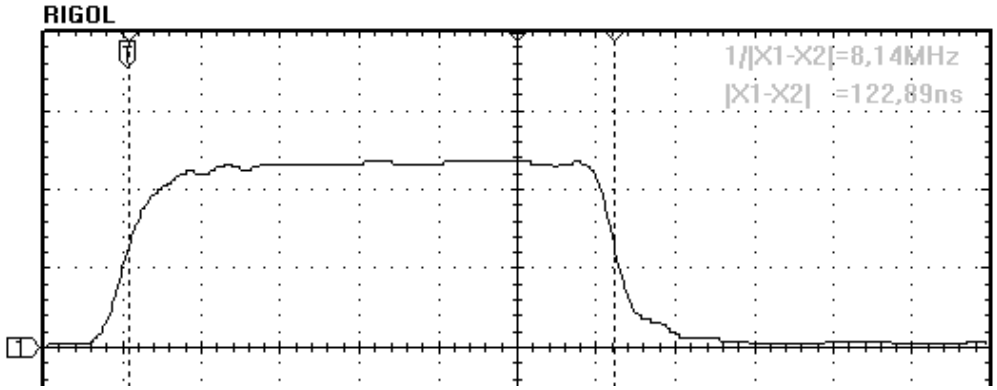
Rapport cyclique minimum :



OC3

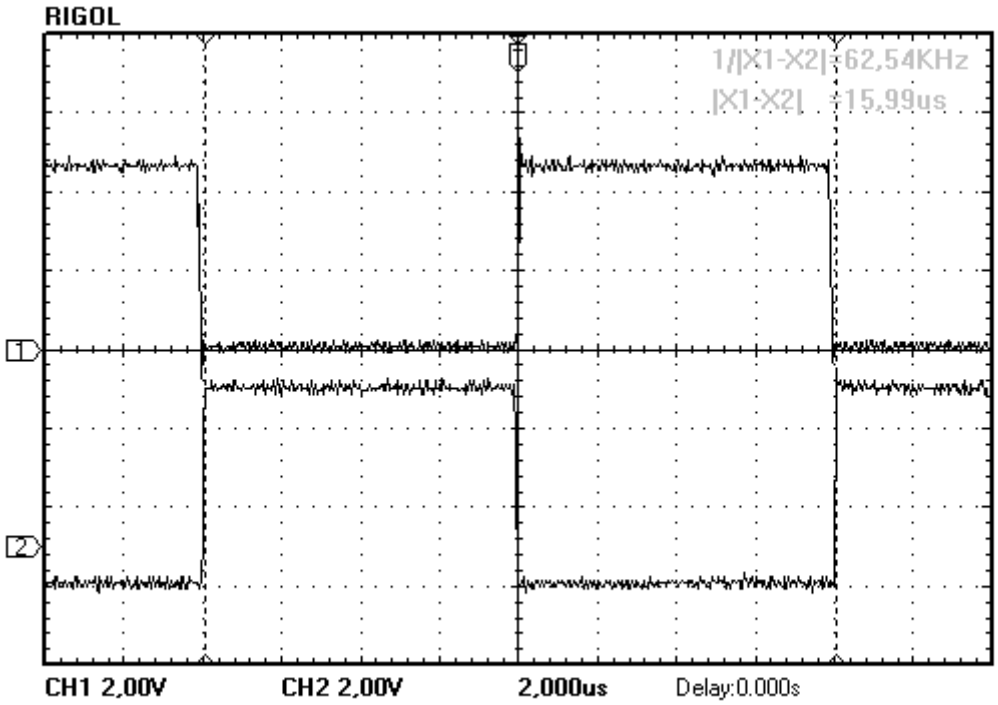
OC4

Un zoom sur les impulsions confirme la durée minimum (par exemple sur OC3): 125nS = 2.Tcy = 2/16MHz



OC3

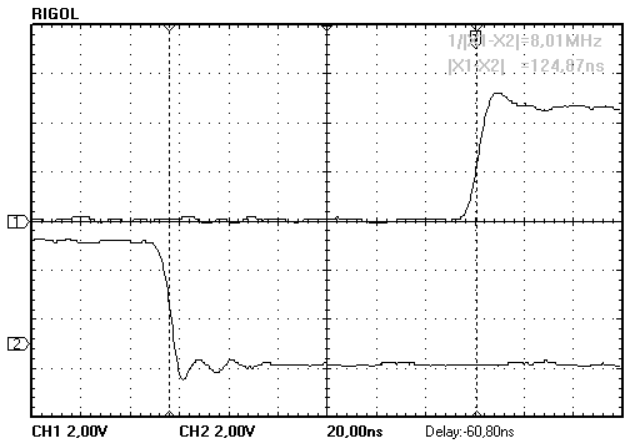
Rapport cyclique maximum :

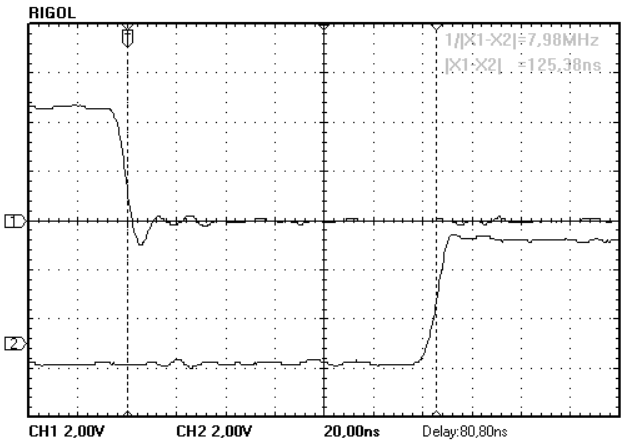


OC3

OC4

Un zoom sur les changements d'états confirme la durée du temps mort (les 2 transistors bloqués) :
125nS = 2.Tcy = 2/16MHz





Le temps mort peut être rallongé en cas de besoin (transistors très lents).

6.5 Inertie sur la commande de rapport cyclique

Il s'agit de vérifier les pentes montantes et descendantes et les fonctions de limitation.
La fonction d'interruption est une nouvelle fois modifiée pour réaliser ces tests.

```
void _ISR _T1Interrupt(void)
{
 IFS0bits.T1IF=0; // Acquittement interruption
 Compt_T1_TXD++;
 if (Compt_T1_TXD==NbDtaTxd)
  {
   Compt_T1_TXD=0; // Compt_T1_TXD modulo NbDtaTxd
   if (PilotCont.Eclairage) // Commande d'éclairage du TdB ?
        PWM_PushPull=(Periode_PushPull/2)+2;//Rapport cycl. supérieur au max.
   else PWM_PushPull=0; // Rapport cyclique inférieur au minimum
   if (PWM_PushPull>((Periode_PushPull/2)-2)) PWM_PushPull=(Periode_PushPull/2)-2;
   if (PWM_PushPull<2) PWM_PushPull=2;
   //Calc_PWM_Ecl(); // Calcul de PWM_PushPull pour obtenir
                     // la tension d'éclairage voulue (V_ECL)
  }
 // Inertie du PWM de l'éclairage (pente = NbDtaTxd x 60/seconde)
 if (PWM_PushPull > PWM_PP_Delay) PWM_PP_Delay++;
 if (PWM_PushPull < PWM_PP_Delay) PWM_PP_Delay--;
 // Affectation des registres PWM
 //OC3R=PWM_PP_Delay;                    // Sortie inversée
 //OC4R=Periode_PushPull/2+PWM_PP_Delay; // Sortie inversée
 OC3RS=PWM_PP_Delay;                   // Sortie non inversée
 OC4RS=Periode_PushPull/2+PWM_PP_Delay; // Sortie non inversée
}
```

L'appel de la fonction "Calc_PWM_Ecl" est inhibé en plaçant la ligne en commentaires.
Elle est remplacée par un test de la commande d'éclairage ("PilotCont.Eclairage") dont le résultat détermine la valeur de la consigne "PWM_PushPull". Les 2 valeurs possibles ont été choisies volontairement au-delà des valeurs min et max normales pour vérifier les fonctions de limitation.

Mode opératoire :
On observe les rapports cycliques de OC3 et OC4 à l'oscilloscope.
En agissant sur la poignée, on chronomètre le temps passé quand le rapport cyclique passe du minimum au maximum, et inversement.

→ *On mesure 2 secondes environ*

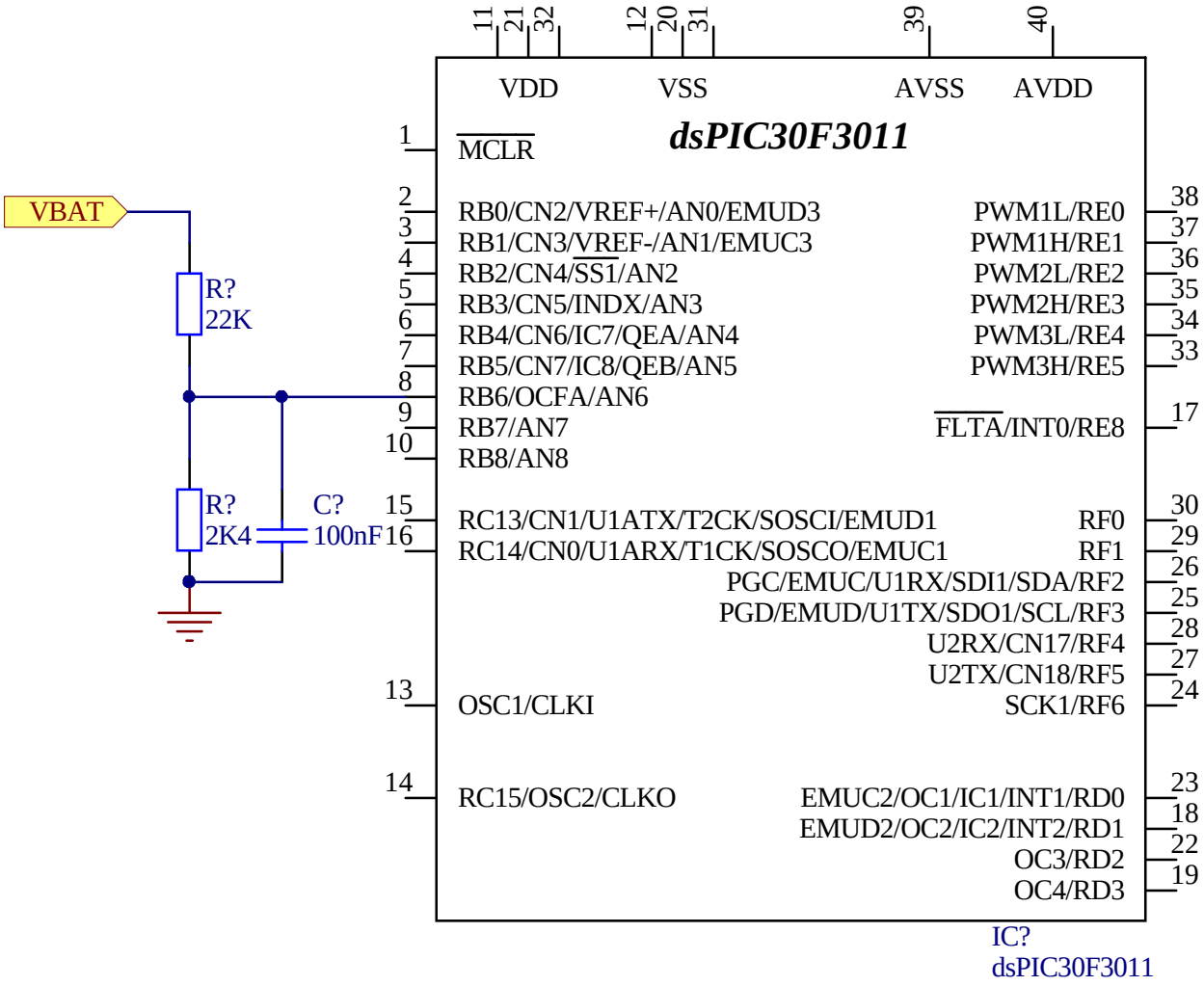
Théoriquement, la durée Ton en nombre de cycle Tcy évolue entre 2 (minimum) et 126 (maximum). La vitesse d'incrémentation devrait être de 60 par seconde : les durées mesurées devraient être donc égales à 2,07 secondes. *CQFD*.

On vérifie aussi les durées Ton de butée :

- minimum : 125nS, soit 2 cycles Tcy
- maximum : temps mort = 125nS, soit Ton_{max} = 128-2 cycles Tcy

6.6 Régulation de la tension d'éclairage

Il ne s'agit pas d'une vraie régulation : la fonction "Calc_PWM_Ecl" calcule le rapport cyclique nécessaire pour obtenir la tension d'éclairage choisie (12V ou 6V) en fonction de la tension batterie (voir §3).
Pour vérifier cette fonction, on simule la tension batterie avec une alimentation de laboratoire et on l'applique sur le dsPIC via un pont diviseur suivant le schéma suivant :



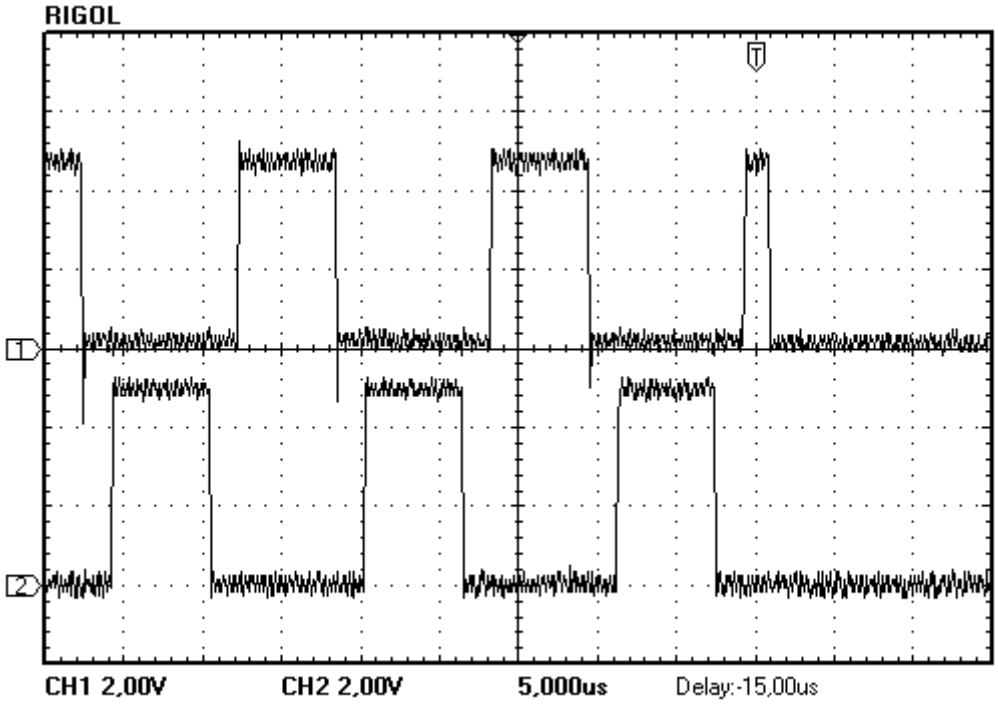
On mesure alors la tension batterie et le rapport cyclique sur plusieurs points pour vérifier la relation de dépendance.

PARAMCONT.V_ECL					
6			12		
VBAT=E	Ton (µS)	Vsmoy théo.	VBAT=E	Vsmoy théo.	Ton (µS)
24V	3,75	5,63V	24V	11,34V	7,56
25V	3,62	5,65V	25V	11,31V	7,24
26V	3,47	5,64V	26V	11,34V	6,98
27V	3,31	5,59V	27V	11,27V	6,68
28V	3,22	5,64V	28V	11,34V	6,48
29V	3,12	5,65V	29V	11,33V	6,25
30V	3	5,63V	30V	11,25V	6
31V	2,87	5,56V	31V	11,24V	5,8
32V	2,81	5,62V	32V	11,24V	5,62

Conclusion : la stabilisation est efficace. L'erreur constante constatée (5 à 6%) est due au pont diviseur de VBAT qui n'utilise pas exactement des résistances de précision.
Note : ce relevé devra être complété par une mesure directe de la tension d'éclairage VSmoy sur la maquette et/ou le prototype.

6.7 Protection contre les surintensité

On provoque une information de surintensité en forçant l'entrée INT2 à l'état bas (les relevés sont réalisés sur une maquette qui ne comporte pas la structure du §4). L'oscilloscope est déclenché au ↓ de INT2 et on observe OC3 et OC4.



OC3

OC4

On constate l'action très rapide du signal de surintensité (1µS environ après l'instant de déclenchement) qui place les signaux OC3 et OC4 dans leurs états de repos (transistors du push-pull bloqués).

Attention : ce programme de tests utilise très peu d'interruptions. Elles seront beaucoup plus nombreuses dans l'application VAE : il faudra répéter ce test pour vérifier la haute priorité de l'interruption INT2 pour permettre sa prise en compte rapide.

7. Relevés sur maquette DEFO

Modifications :
- Pont diviseur VBAT : 2K2/56K ⇒ K_PWM = 9 (voir §3.2.3)
- L1 et L2 = 100µH/1200mA (Farnell 3877516) pour indisponibilité des composants prévus
- Ajout d'une résistance pull-up de 5K6 sur FLT_ECL pour fixer un état de repos "1" si IC4 non soudé

Conditions :
- VCC=12V
- Régulateur 5V de la carte
- Tous les relevés avec sondes 1/10 (en choisissant le bon GND)

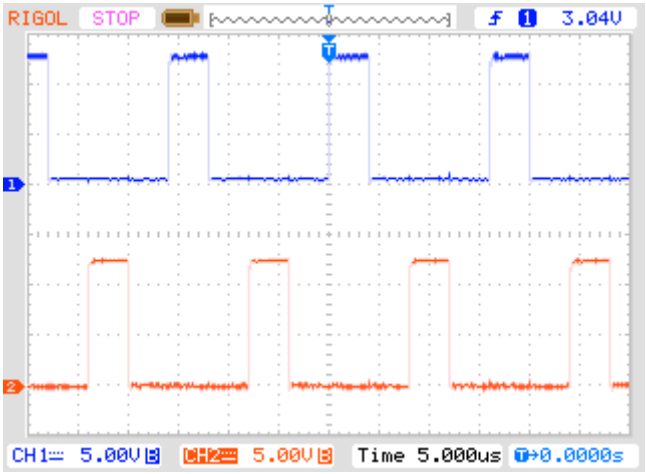
7.1 Rapports cycliques fixes

La fin de la fonction "_T1Interrupt" est modifiée comme suit :

```
// Affectation des registres PWM
 // OC2RS=PWM_PP_Delay;                    // Sortie non inversée
 // OC3RS=Periode_PushPull/2+PWM_PP_Delay; // Sortie non inversée
 OC2RS=Periode_PushPull/4;                   // Sortie non inversée
 OC3RS=Periode_PushPull/2+Periode_PushPull/4; // Sortie non inversée
```

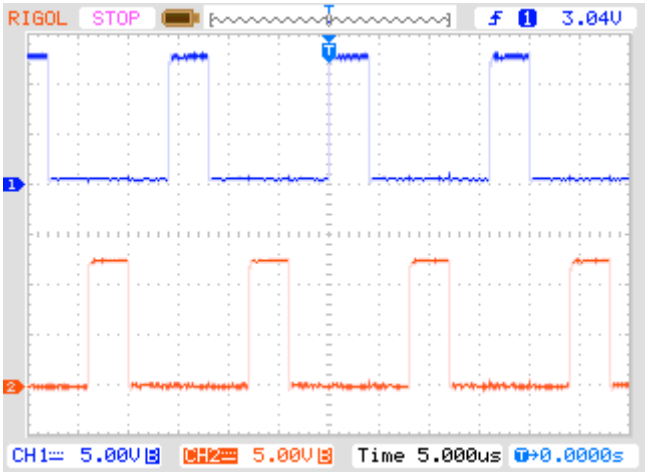
On produit ainsi 2 signaux PWM de rapport cyclique fixe de 25%.

7.1.1 PWM1ECL et PWM2ECL



Les signaux sont conformes aux attentes :
- période = 16µS
- rapport cyclique = 25%
- déphasage de 8µS (1/2 période) entre les 2 signaux

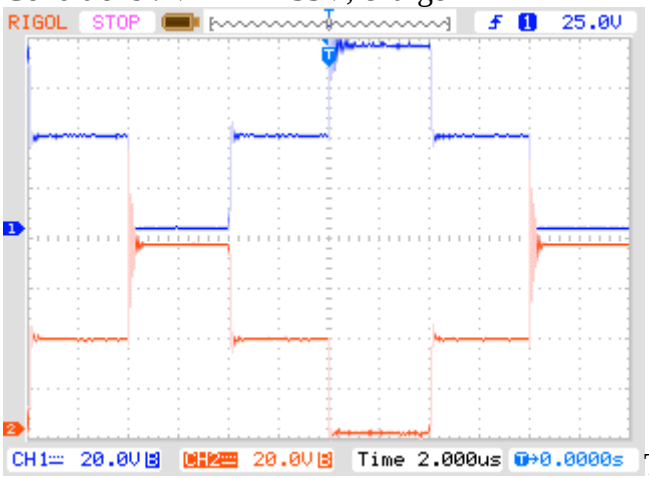
7.1.2 PT8 et PT9



Conformes : niveaux H = 12V
Restent à mesurer les temps de commutation

7.1.3 PT5 et PT6

Conditions : VBAT = 35V, charge = 27Ω



Temps de conduction = 4µS (25%)

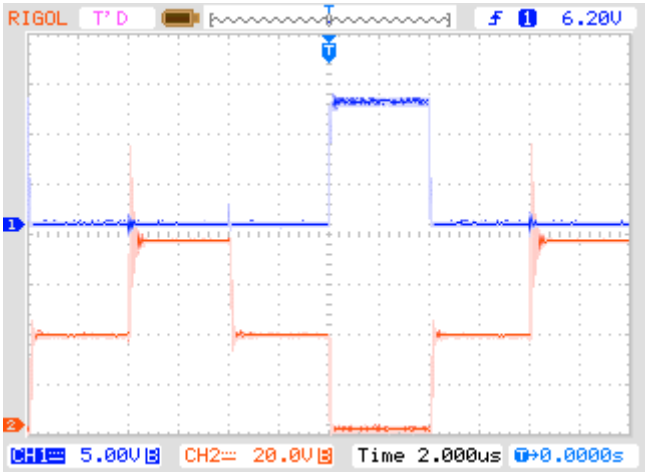
Conformes à la simulation :

– niveau "L" (qd le MOS concerné conduit) = 0V
– niveau intermédiaire : VBAT=35V (les 2 MOS bloqués)
– niveau H (qd l'autre MOS conduit) = 70V (2xVBAT)

Les oscillations parasites peuvent être atténuées avec les réseaux C20-R8 et C19-R7 non câblés

7.1.4 PT9 et PT6

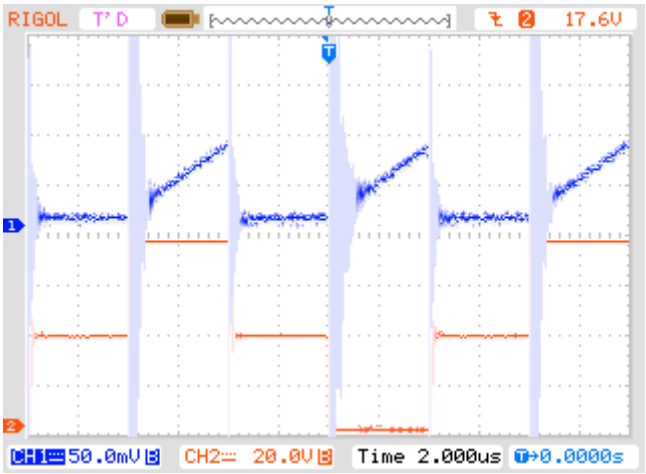
Mêmes conditions



VDS2 est bien nulle qd T2 est commandé

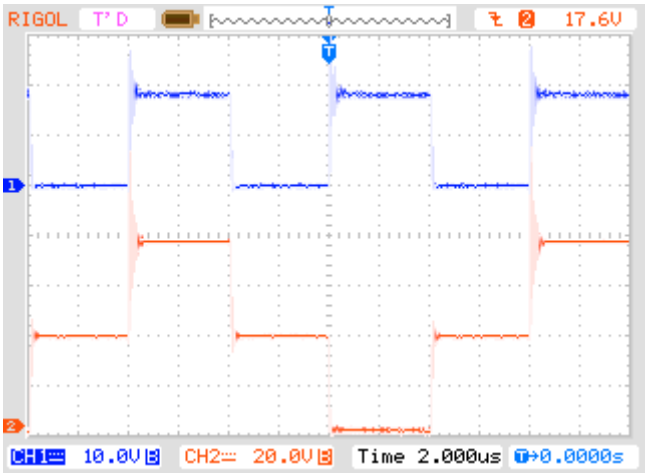
7.1.5 PT6 et PT7

Mêmes conditions



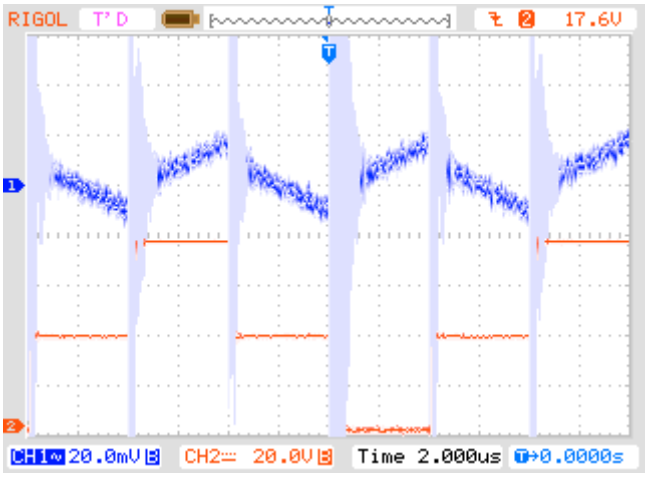
En ignorant les oscillations parasites, on relève une pente de 50mV/2µS, soit 230mA/2µS (R12=0,22Ω).
La pente théorique devrait être de ((VBATx(N2/N1)-VS)/L1)x(N2/N1) = (35x(1/2)-8,66)/100µH)x(1/2) =

7.1.6 Cathodes de D1 et D2 et PT6



On relève le niveau H au secondaire et en aval des diodes : 18V environ .
C'est conforme (théorie : VBAT/2 – chute de tension diode)

7.1.7 Ondulations VS et PT6



En ignorant les oscillations HF, l'ondulation c.c. est de 30mV environ. Parfait, compte-tenu de l'usage en éclairage

7.2 Régulation de VS

On supprime les lignes de programme fixant le rapport cyclique à 25% pour vérifier la fonction de régulation.
Compte tenu de l'équipement disponible (alim limitée à 35V), on a choisi une consigne de 6V ("PARAMCONT.V_ECL=6;" dans la fonction "main").
Conditions :

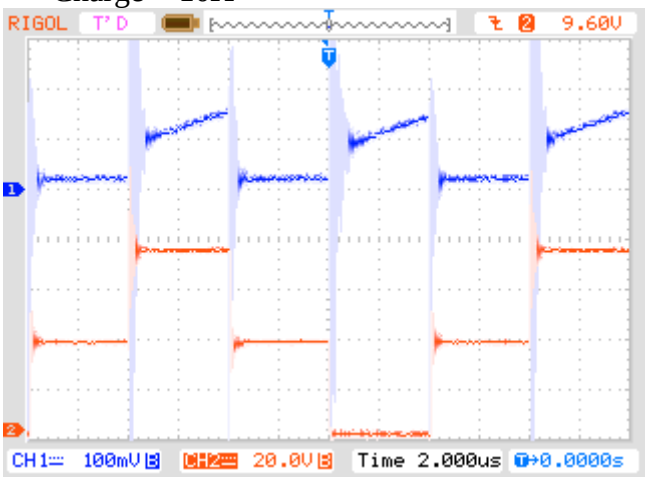
– Charge = 10Ω

7.2.1 PT6 et PT7

Conditions :

– VBAT = 35V

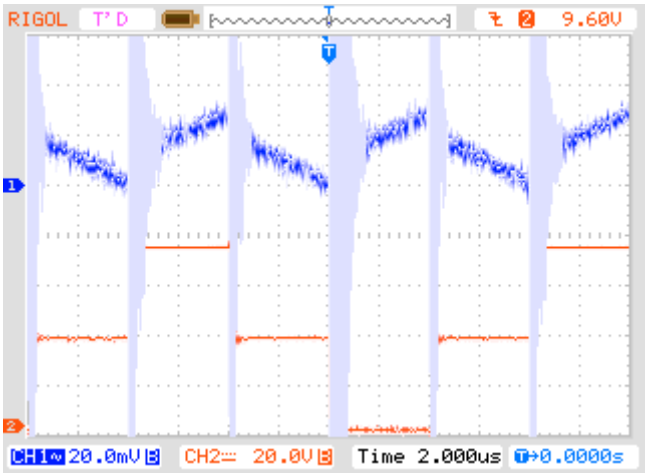
– Charge = 10Ω



Le courant max atteint 150mV/0,22Ω=0,68A (il sera plus faible avec une consigne 12V et la même puissance, mais il augmente si VBAT diminue).

Ce courant max est compatible avec les transistors MOS.

7.2.2 Ondulations VS et PT6



L'ondulation c.c. est de 30mV environ (en ignorant les oscillations HF).

7.2.3 Performance de la régulation

On relève les points suivants :

VBAT (V)	24,86	27,04	29	31	33	35	37,09
VS (V)	6,49	6,49	6,46	6,48	6,42	6,45	6,45
Rapport cy.	28,5%	26%	24%	22,5%	21%	20%	18,5%

La régulation est parfaite pour l'éclairage

Les tests doivent être complétés pour des tensions VBAT plus grandes

Note : le logiciel coupe l'éclairage si VBAT < 24V

7.3 Rendements

Les résultats ci-dessous sont très approximatifs compte tenu des instruments utilisés (galvanomètre de l'alim).

VBAT = 35V ,charge = 10Ω, rapport cyclique = 25% : Ibat = 200mA, VS=8,14V, IS=0,81A

$$\Rightarrow \Gamma=(8,14x0,81)/(35x0,2)=94\%$$

Très bon : à vérifier

SOMMAIRE