

Rapport de laboratoire

**Ecole supérieure
Électronique**

Laboratoire MINF
Salle R110

PIC32MX – PWM et AD

Réalisé par :

Jérémie Jean-Elie
Diego Savary

A l'attention de :

Philippe Bovey
Serge Castoldi

Dates :

Début du laboratoire :
Fin du laboratoire : 14 janvier 2025

Table des matières :

PIC32MX – PWM et AD	1
1 Cahier des charges	4
2 Dimensionnement des timers	4
2.1 Timer 1	4
2.2 Timer 2	4
2.3 Timer 3	5
2.4 Timer 4	5
3 Configuration des OC	6
3.1 OC2	6
3.2 OC3	6
4 Programmation	7
5 Mesures	7
5.1 Mesures pourcentages sur moteur DC et PWMSoft	7
5.1.1 Schéma de mesures	7
5.1.2 Tableau des résultats du moteur DC	7
5.1.3 Tableau des résultats du PWMSoft	8
5.2 PWM à 0	9
5.3 Mesures d'angles	9
5.3.1 Schéma de mesures	9
5.3.2 Tableau des résultats	9
5.4 Commande du pont en H	10
6 Problèmes rencontrés	10
6.1 Réglage de priorités des interruptions	10
6.2 Mesures des priorités des interruptions	13
6.2.1 Schématique et instrumentation	13
6.2.2 Méthode de mesures	13
6.2.3 Résultats des mesures	14
6.2.3.1 Timer 4 et 1 à la priorité 4	14
6.2.3.2 Timer 4 à la priorité 1 et le timer 1 à la priorité 4	15
6.2.3.3 Timer 4 à la priorité 4 et le timer 1 à la priorité 1	16
7 Conclusion	17
8 Annexes	18
8.1 PWM à 5%, 50% et 95%	18
8.1.1 Moteur DC	18
8.1.1.1 Mesure à 5% et -5%	18
8.1.1.2 Mesure à 50% et -50%	19
8.1.1.3 Mesure à 95% et -95%	20
8.1.2 PWMSoft	21
8.1.2.1 Mesure à 5% et -5%	21
8.1.2.2 Mesure à 50% et -50%	22
8.1.2.3 Mesure à 95% et -95%	23
8.2 PWM à 0	24
8.2.1 Moteur DC	24
8.2.2 PWMSoft	24
8.3 Mesures d'angle	25
8.3.1 -90°	25
8.3.2 0°	25
8.3.3 +90°	26

1 Cahier des charges

Voir donnée en annexe

2 Dimensionnement des timers

2.1 Timer 1

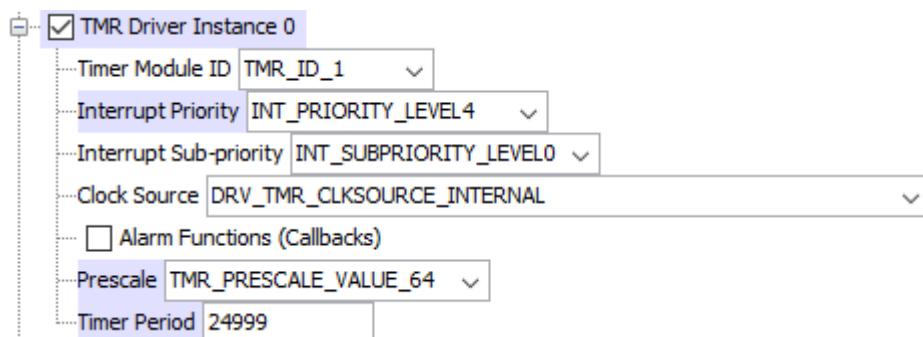
Comme indiqué sur la donnée le timer 1 doit avoir une période de 20ms, avec une priorité d'interruption de niveau 4, pour ce faire nous avons comme clock interne 80MHz.

Timer Period = $20'000\mu s * 80 = 1'600'000$ dépasse la valeur max pour un timer 16 bits

Besoin d'un prescaler de $1'600'000 / 65'536 = 24.41 \Rightarrow 64$

Il faut choisir un prescaler de 64 car le timer 1 ne supporte que 1, 8, 64, 256.

Timer Period = $(20'000\mu s * 80 / 64) - 1 = 24'999$

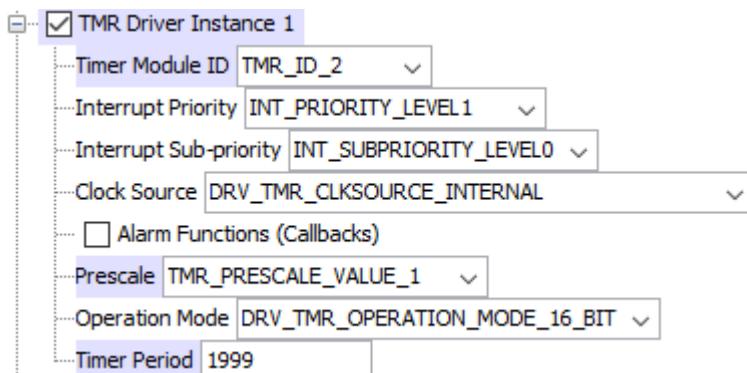


2.2 Timer 2

Configurer le timer 2 pour avoir un PWM à 40kHz, donc une période de 25us.

Pour ce timer, il n'y a pas besoin de mettre de prescaler, il faut donc prendre 1.

Timer Period = $(100\mu s * 80) - 1 = 1'999$

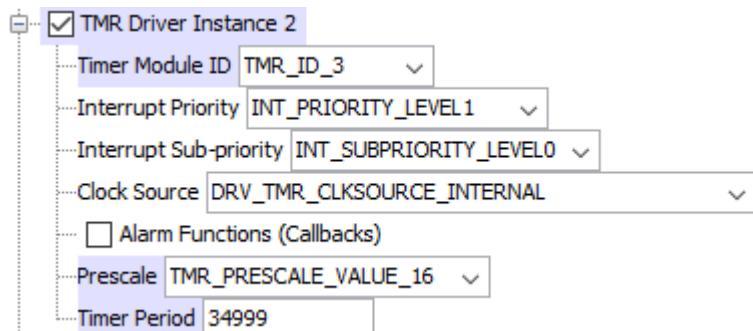


2.3 Timer 3

Le timer doit générer un PWM d'une période de 7ms. Il n'y a pas besoin d'une interruption.

Timer Period = $7'000\mu\text{s} * 80 = 560'000$ dépasse la valeur maximale pour 16 bits.
Besoin d'un prescaler de $560'000 / 65536 = 8.55 \Rightarrow 16$

Timer Period = $(7'000\mu\text{s} * 80 / 16) - 1 = 34'999$

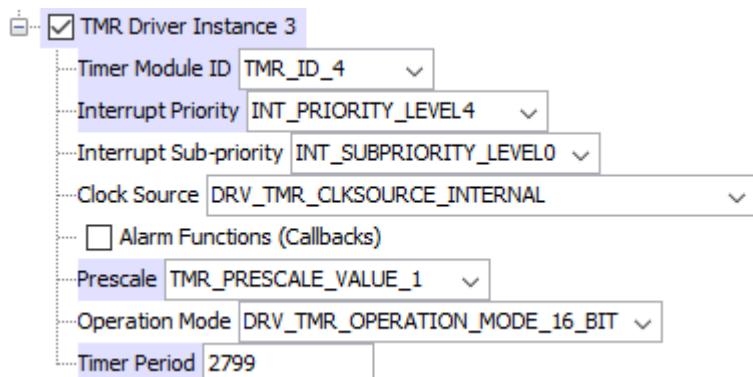


2.4 Timer 4

Ce timer doit être configuré pour générer une interruption toutes les $35\mu\text{s}$ avec une priorité d'interruption 4 au départ.

Timer Period = $35\mu\text{s} * 80 = 2'800$ pas besoin de prescaler

Timer Period = $(35\mu\text{s} * 80) - 1 = 2'799$

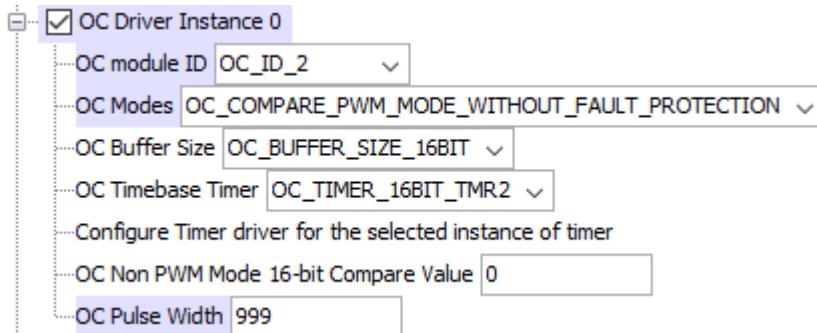


3 Configuration des OC

3.1 OC2

Le module OC2 doit être configuré pour travailler en PWM, en générant un signal d'une fréquence de 40kHz.

Le choix de la longueur d'impulsion correspond à un PWM de 50%.
 OC Pulse Width = (Timer2 Period / 2) – 1 = (2'000 / 2) – 1 = 999

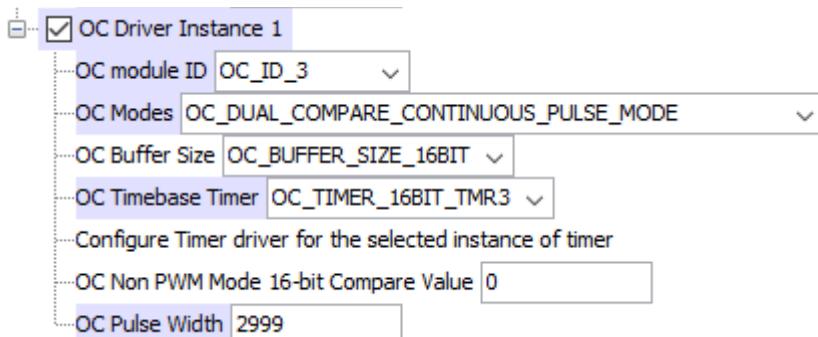


3.2 OC3

Le module OC3 doit être configuré pour travailler en modulation de largeur d'impulsion en relation avec le timer 3, en générant des impulsions d'une largeur modulable entre 0,6ms et 2,4ms.

Le choix de la longueur d'impulsion correspond à une impulsion de 0,6ms en se basant sur la période du timer 3 qui est de 34'999 pour 7ms.

$$\text{OC Pulse Width} = \left(\frac{35000}{7} * 0.6 \right) - 1 = 2'999$$



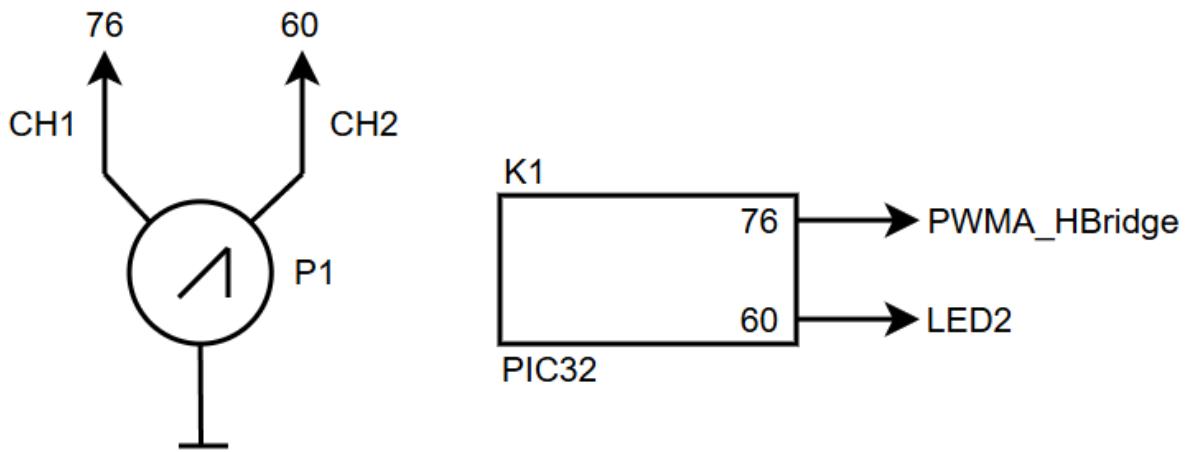
4 Programmation

Tous nos .c et .h sont sur Github

5 Mesures

5.1 Mesures pourcentages sur moteur DC et PWMSoft

5.1.1 Schéma de mesures



5.1.2 Tableau des résultats du moteur DC

Pourcentage sur l'écran	Durée haute théorique [μs]	Durée haute mesurée [μs]
-95%	23.75	23.75
-50%	12.5	12.64
-5%	1.25	1.276
5%	1.25	1.275
50%	12.5	12.64
95%	23.75	24.05

Pour trouver le temps théorique :

$$t_{haut} = t * Pourcentage = 25\mu s * Pourcentage$$

Les écarts de valeurs peuvent s'expliquer par le positionnement des curseurs sur l'oscilloscope. En regardant attentivement les oscillogrammes, on remarque que les flancs du signal ne sont pas droits. Ceci est du au fait que les temps d'impulsions sont assez petits (1.25μs à 23.75μs). Il est donc normal qu'il puisse y avoir de petites erreurs de mesures dues au positionnement des curseurs.

À part ça, les valeurs restent dans la plage attendue (max. 0.3μs d'erreur).

5.1.3 Tableau des résultats du PWMSoft

Pourcentage sur l'écran	Durée haute théorique[ms]	Durée haute mesurée [ms]
-95%	3.325	3.33
-50%	1.75	1.752
-5%	0.175	0.175
5%	0.175	0.175
50%	1.75	1.752
95%	3.325	3.33

Pour trouver le temps théorique :

$$t_{haut} = t * Pourcentage = 3.5ms * Pourcentage$$

Toutes les valeurs mesurées correspondent aux valeurs attendues. Cette fois-ci, les signaux étaient beaucoup plus faciles à mesurer car les temps d'impulsions étant plus long qu'avant (ici en ms au lieu de μ s), les flancs mesurés étaient nets. Il était donc facile de placer les curseurs sur ceux-ci.

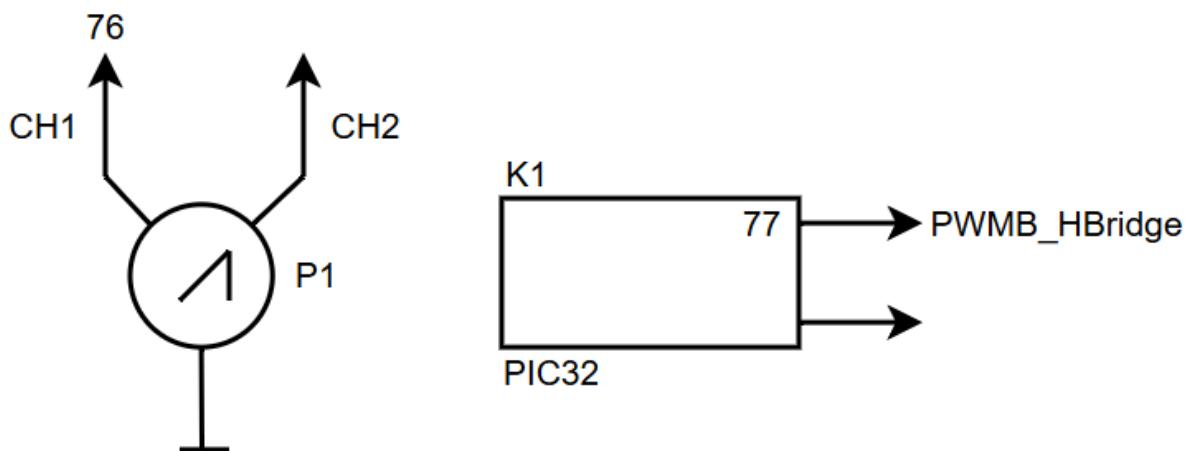
5.2 PWM à 0

Lorsque la mesure du moteur DC est fait avec un PWM à 0, celle-ci présente un signal à 0V. Le moteur ne tourne donc pas.

Lorsque l'on mesure le PWMSoft, la tension en sortie est de ~3,3V. Ceci est tout à fait normal car les LEDs sont actives basses. Ce qui veut dire qu'elles sont allumées quand la sortie du PIC32 est à 0V. Il est donc tout à fait logique que ces LEDs soient éteintes avec un PWM à 0.

5.3 Mesures d'angles

5.3.1 Schéma de mesures



5.3.2 Tableau des résultats

Angle sur l'écran	Durée haute théorique [ms]	Durée haute mesurée [ms]
-90	0.6	0.6
0	1.5	1.5
90	2.4	2.4

Pour trouver le temps théorique à l'angle 0 :

$$t_{haut} = \frac{t_{max} + t_{min}}{2} = \frac{2.4ms + 0.6ms}{2} = 1.5ms$$

Les temps mesurés correspondent exactement aux temps attendus. Les signaux sur l'oscilloscope étant propres, il était donc facile de venir poser le curseur sur les flancs des signaux.

5.4 Commande du pont en H

Pour commander le pont en H, il suffit de regarder le tableau mis dans la donnée du TP.

Input				Output			
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode	
H	H	H/L	H	L	L	Short brake	
L	H	H	H	L	H	CCW	
		L	H	L	L	Short brake	
H	L	H	H	H	L	CW	
		L	H	L	L	Short brake	
L	L	H	H	OFF (High impedance)		Stop	
H/L	H/L	H/L	L	OFF (High impedance)		Standby	

Il faut se concentrer sur les modes CCW (Counter ClockWise) et CW (ClockWise). À remarquer que dans ces deux modes, l'entrée STBY (Standby) doit être à l'état haut. Pour faire tourner le moteur en sens horaire (CW), l'entrée IN1 doit être à l'état haut et l'entrée IN2 doit être à l'état bas. Pour faire tourner le moteur en sens antihoraire (CCW), il suffit d'inverser l'état des deux entrées.

Pour freiner le moteur, il est possible de mettre les 2 entrées à l'état haut.

6 Problèmes rencontrés

6.1 Réglage de priorités des interruptions

Avant de reciter le problème voici comment fonctionne Harmony, lorsqu'un changement se fait sur son interface et qu'on appuie sur generate code  , il va changer le code entier avec les changements voulus. Dans notre cas, c'est la priorité des interruptions, donc il va changer directement dans le drv_tmr_static.c, la photo suivante est le timer 1 pour représentation :

```

void DRV_TMR0_Initialize(void)
{
    /* Initialize Timer Instance0 */
    /* Disable Timer */
    PLIB_TMR_Stop(TMR_ID_1);
    /* Select clock source */
    PLIB_TMR_ClockSourceSelect ( TMR_ID_1, TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK );
    /* Select prescalar value */
    PLIB_TMR_PrescaleSelect(TMR_ID_1, TMR_PRESCALE_VALUE_64);
    /* Enable 16 bit mode */
    PLIB_TMR_Mode16BitEnable(TMR_ID_1);
    /* Clear counter */
    PLIB_TMR_Counter16BitClear(TMR_ID_1);
    /*Set period */
    PLIB_TMR_Period16BitSet(TMR_ID_1, 24999);
    /* Setup Interrupt */
    PLIB_INT_VectorPrioritySet(INT_ID_0, INT_VECTOR_T1, INT_PRIORITY_LEVEL4);
    PLIB_INT_VectorSubPrioritySet(INT_ID_0, INT_VECTOR_T1, INT_SUBPRIORITY_LEVEL0);
}

```

Comme indiqué dans le carré rouge, la priorité est au niveau quatre, donc le changement est bien effectué ici.

Mais il y a un deuxième endroit où il doit changer la priorité, et c'est ce deuxième endroit où Harmony ne le fait pas, dans le système_interrupt.c, voici un screen démontrant ceci :

```

void __ISR(_TIMER_1_VECTOR, ipl4AUTO) IntHandlerDrvTmrInstance0(void)
{
    BSP_LEDOn(BSP_LED_0);

    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_1);

    static uint8_t compteur = 0;

    if(compteur == 150)
    {
        APP_UpdateState(APP_STATE_SERVICE_TASKS);

        GPWM_GetSettings(&PWMData);
        GPWM_DisableSettings(&PWMData);
        GPWM_ExecPWM(&PWMData);
    }
    else
    {
        compteur++;
    }

    BSP_LEDOff(BSP_LED_0);
}

```

Voici l'ISR du timer 1, ici nous pouvons aussi voir dans quel niveau de priorité d'interruption nous sommes avec l'encadré bleu. Ici, vous voyez qu'il correspond au screen précédent et avec le changement Harmony. Mais ceci est dû au fait qu'il a été changé à la main. En effet, il semble que lorsque nous chargeons Harmony pour la première fois et que nous changeons ces priorités il gardera quand même les paramètres lors de la première génération du code. Dans notre code, tous les timer avaient une priorité d'interruption de 1 donc lorsque nous voulions changer cela tout restait à 1, une Example ci-dessous des ISR des autres timer.

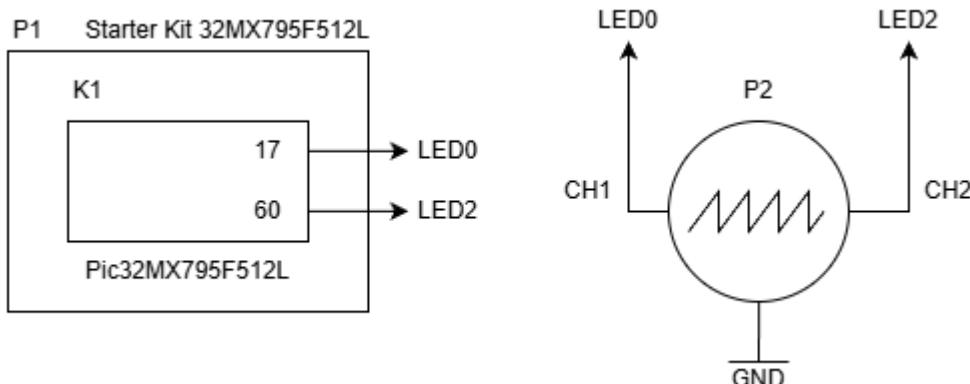
```
void __ISR(_TIMER_2_VECTOR, ip11AUTO) IntHandlerDrvTmrInstance1(void)
{
    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_2);
}
void __ISR(_TIMER_3_VECTOR, ip11AUTO) IntHandlerDrvTmrInstance2(void)
{
    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_3);
}
void __ISR(_TIMER_4_VECTOR, ip14AUTO) IntHandlerDrvTmrInstance3(void)
```

C'est embêtant car dans la donnée il nous est demandé que le timer 2 et 3 n'ai pas d'interruption et de changer les priorités pour les timer 4 et 1 pour voir ce que cela fait.

La solution à été de tout simplement supprimer à la main les interruptions des timers 2 et 3 et de changer à chaque fois l'encadré bleu mentionné précédemment aussi à la main avec la priorité voulue, mais attentions il faut toujours changer les priorités dans Harmony.

6.2 Mesures des priorités des interruptions

6.2.1 Schématique et instrumentation



Nom	Marque	Modèle	Type	N°ETML-ES
G1	GwInsteek	GPS-3303	Alimentation	ES.SLO2. 00.00.30
G2	Keysight	33500B series	Générateur de fonction	ES.SLO2.R110-134
P1	Rohde & Schwarz	RTBB2004	Oscilloscope	ES.SLO2.05.01.01

6.2.2 Méthode de mesures

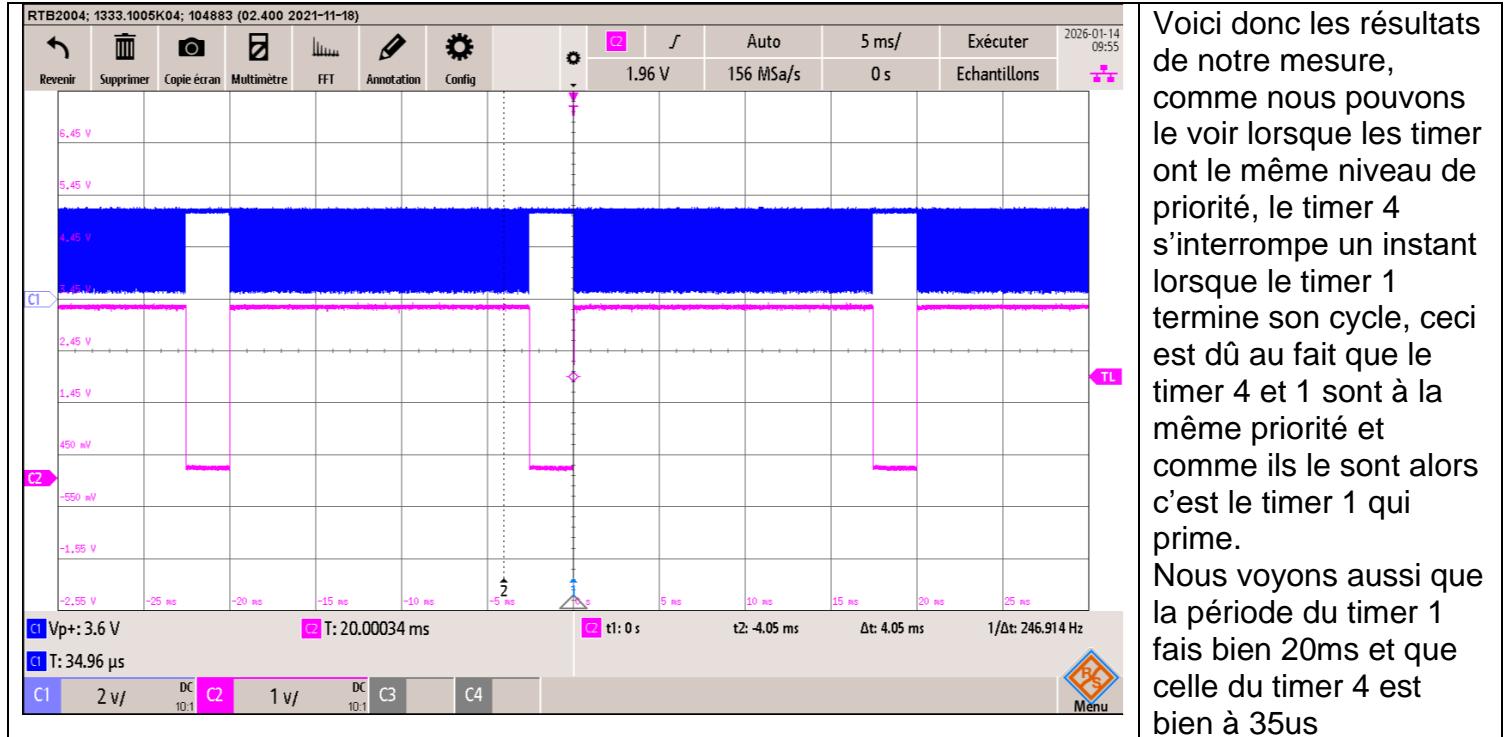
Pour cette mesure la première chose à faire est de rajouter quelque ligne de code, ces lignes irons dans le système_interruption.c et dans l'ISR des timer, pour le timer 1 écrive au début de l'interruption, la ligne permettant d'allumer la LED 0, quand l'interruption est terminée éteignez la LED 0 et fait de même pour le timer 4 et la LED 2.

Une autre chose à faire est de changer les niveaux de priorité des interruptions, voici les changements à faire : pour la première mesure nous allons mettre les priorités des deux interruptions au niveau 4, pour la deuxième mesure le timer 1 restera en priorité 4 mais le timer 4 changera en priorité 1, et pour la troisième mesure ce sera l'inverse, le timer 1 sera à la priorité 1 et le timer 4 à la priorité 4.

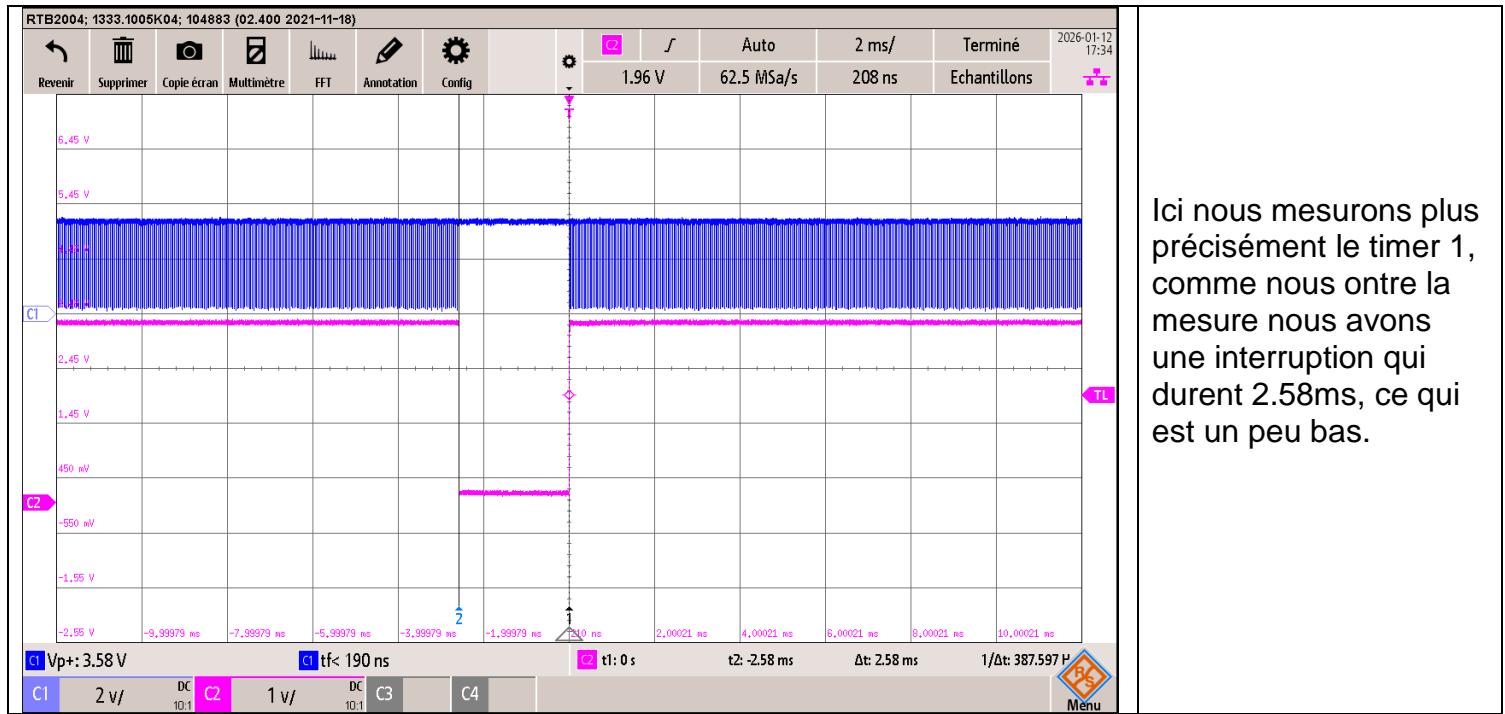
6.2.3 Résultats des mesures

6.2.3.1 Timer 4 et 1 à la priorité 4

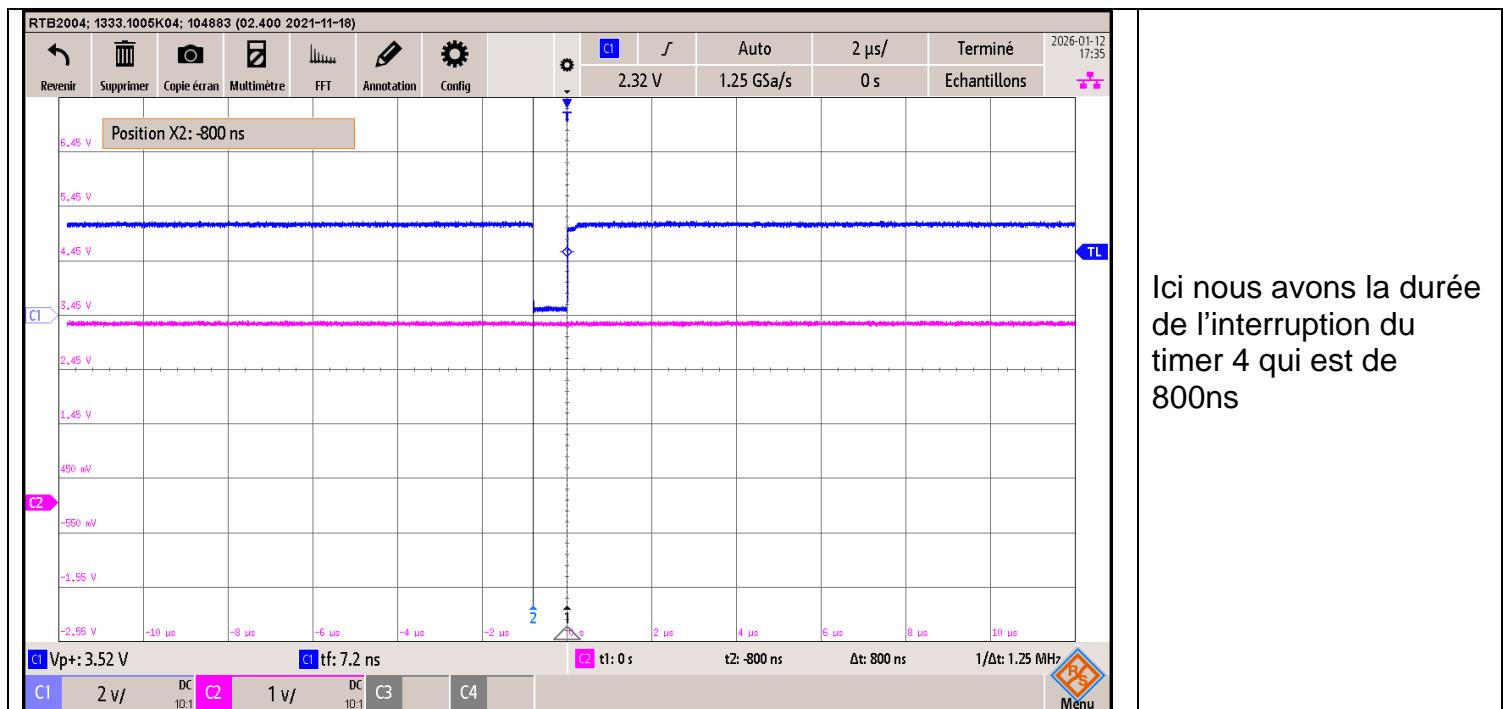
Vue globale des courbes :



Mesure de l'interruption du timer 1



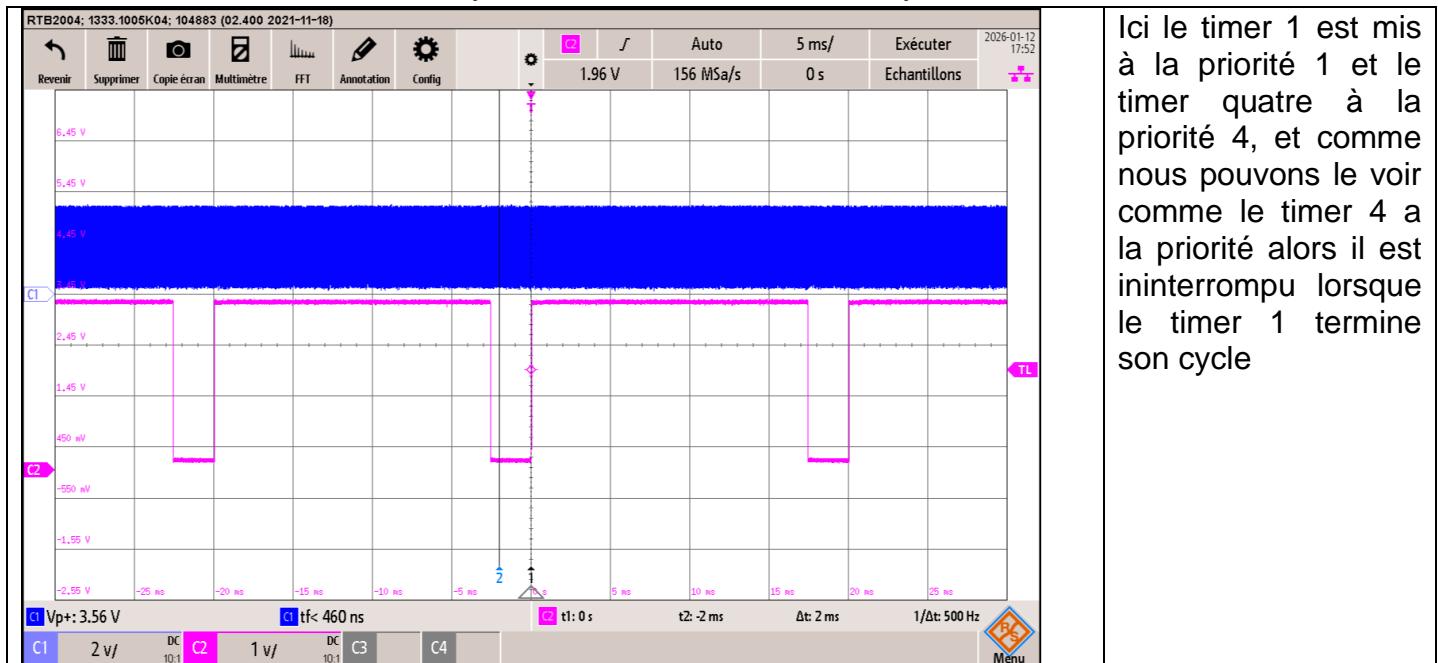
Mesure de l'interruption du timer 4



6.2.3.2 Timer 4 à la priorité 1 et le timer 1 à la priorité 4



6.2.3.3 Timer 4 à la priorité 4 et le timer 1 à la priorité 1



Ici le timer 1 est mis à la priorité 1 et le timer quatre à la priorité 4, et comme nous pouvons le voir comme le timer 4 a la priorité alors il est ininterrompu lorsque le timer 1 termine son cycle

7 Conclusion

Pour les mesures de PWM, toutes les valeurs mesurées correspondaient aux valeurs attendues. Les seules erreurs trouvées étaient dues à des erreurs de mesures. Les flancs n'étant pas parfaitement droits, il était difficile de placer précisément les curseurs sur ceux-ci pour trouver une valeur correcte.

Les PWM du moteur DC, du servomoteur ainsi que le PWMSoft étaient tous corrects. La seule difficulté rencontrée lors de cette partie était avec le configurateur graphique Harmony. Pour résumer, à la première génération du code, tous les paramètres générés étaient corrects. Mais à la 2^{ème} génération, certains paramètres mis à jour dans le configurateur graphique ne pouvaient pas être mis à jour dans le code. Dans ce cas, il fallait donc modifier manuellement le code mais quand même garder la configuration désirée sur Harmony. Ce problème a été rencontré à deux reprises. La première, lors de la modification des priorités d'interruption. Le code inscrit dans le fichier « system_interrupt.c » restait tel qu'il était lors de la première génération. La deuxième fois était pour simplement désactiver des interruptions sur 2 timers qui avaient été mises en priorité 1 par mégarde à la première configuration. Il a fallu là aussi modifier manuellement le fichier « system_interrupt.c » pour enlever les ISR de ces deux timers.

Lieu : Lausanne

Date : 14.01.2025

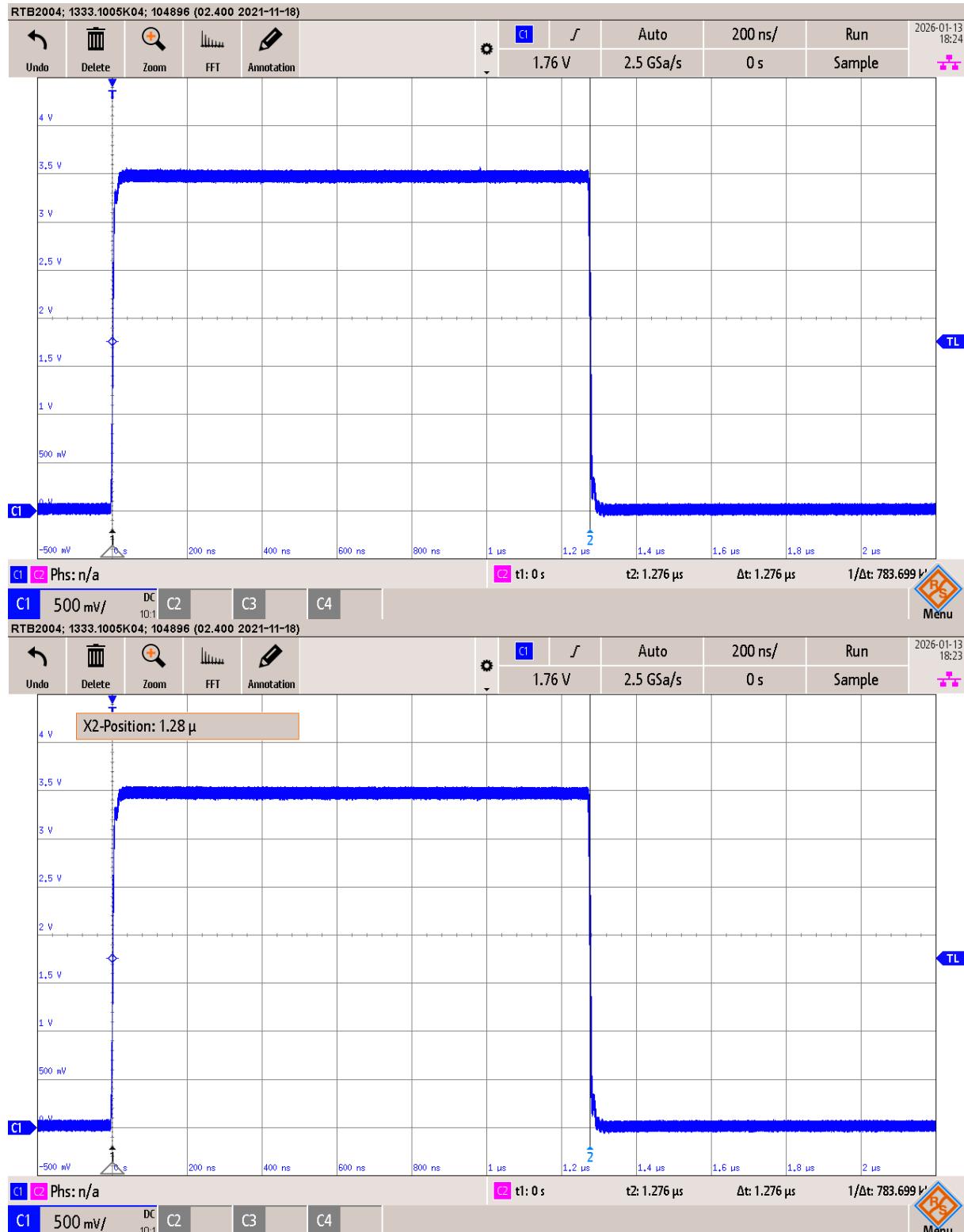
Signatures :

8 Annexes

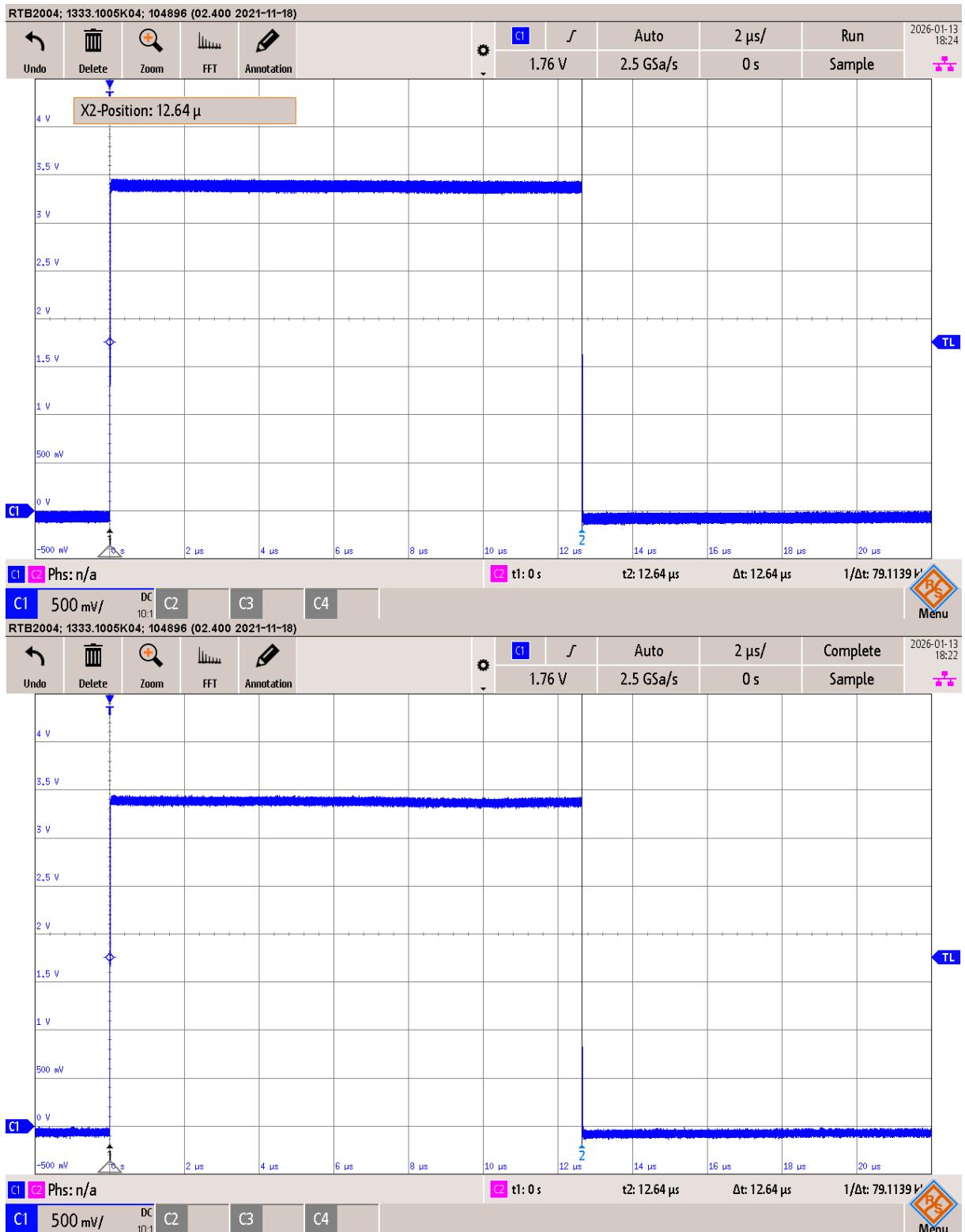
8.1 PWM à 5%, 50% et 95%

8.1.1 Moteur DC

8.1.1.1 Mesure à 5% et -5%

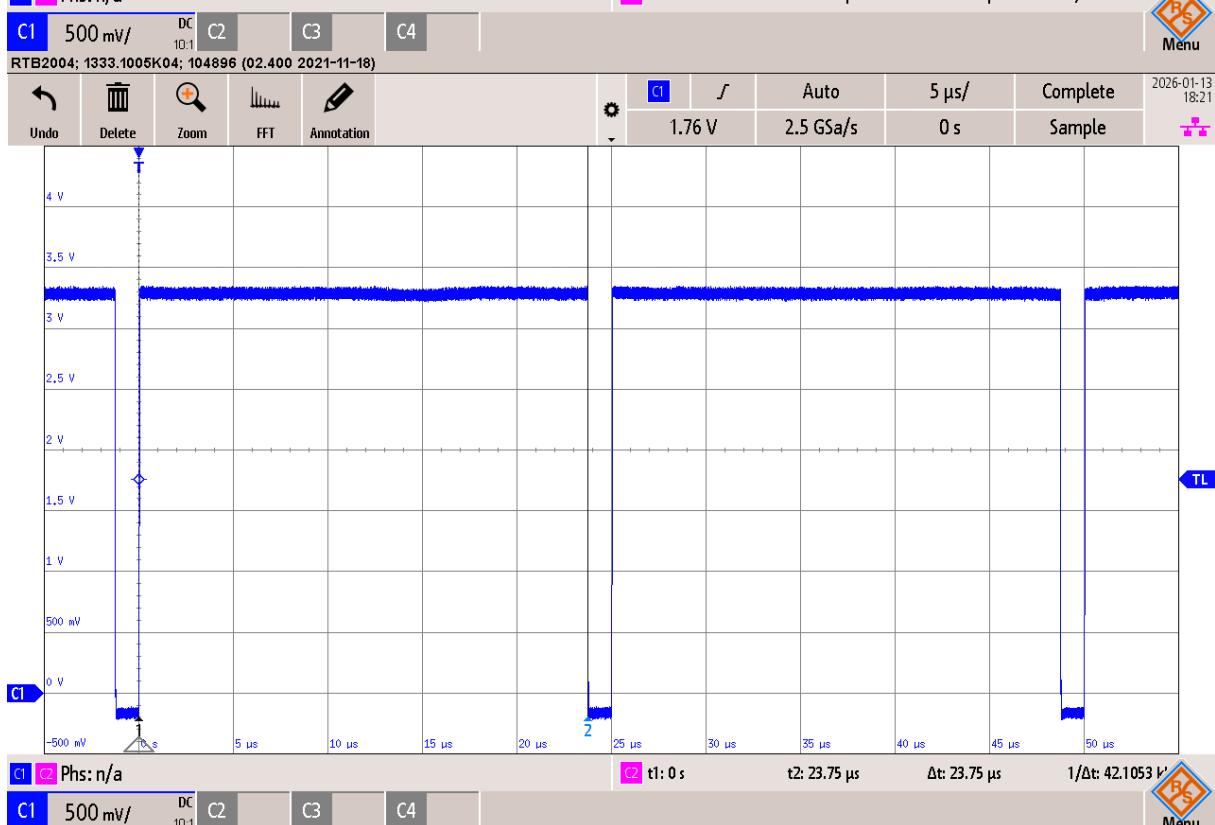
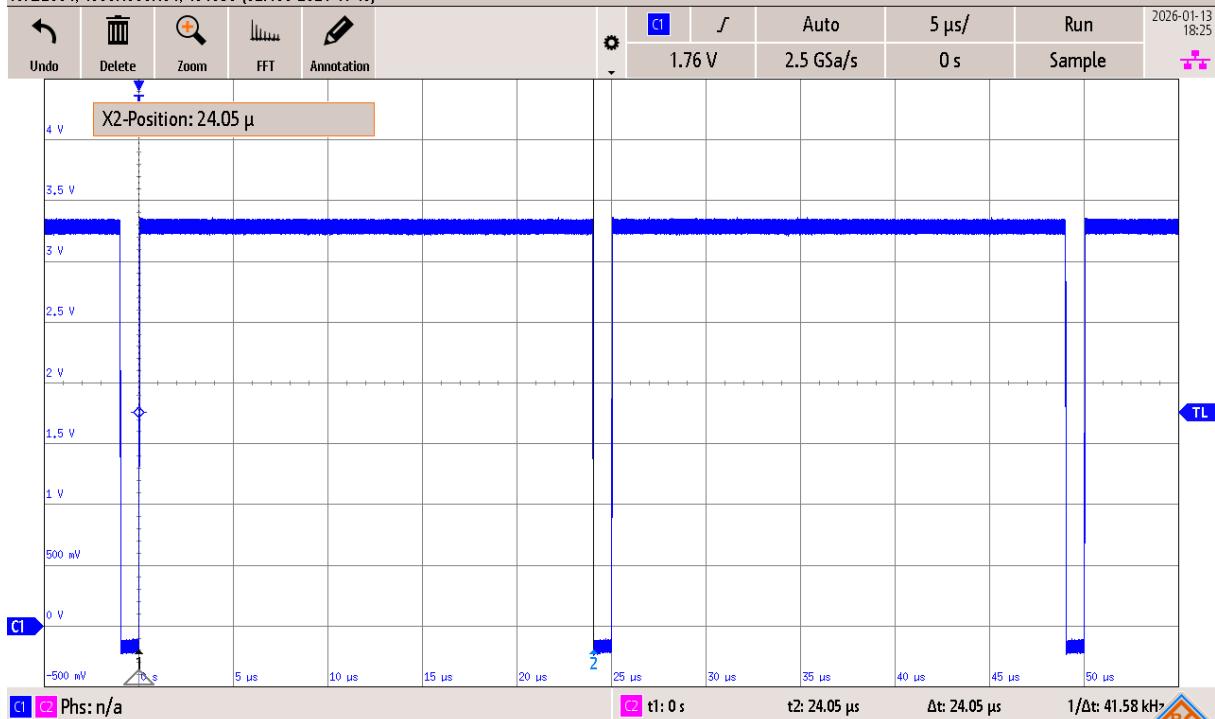


8.1.1.2 Mesure à 50% et -50%



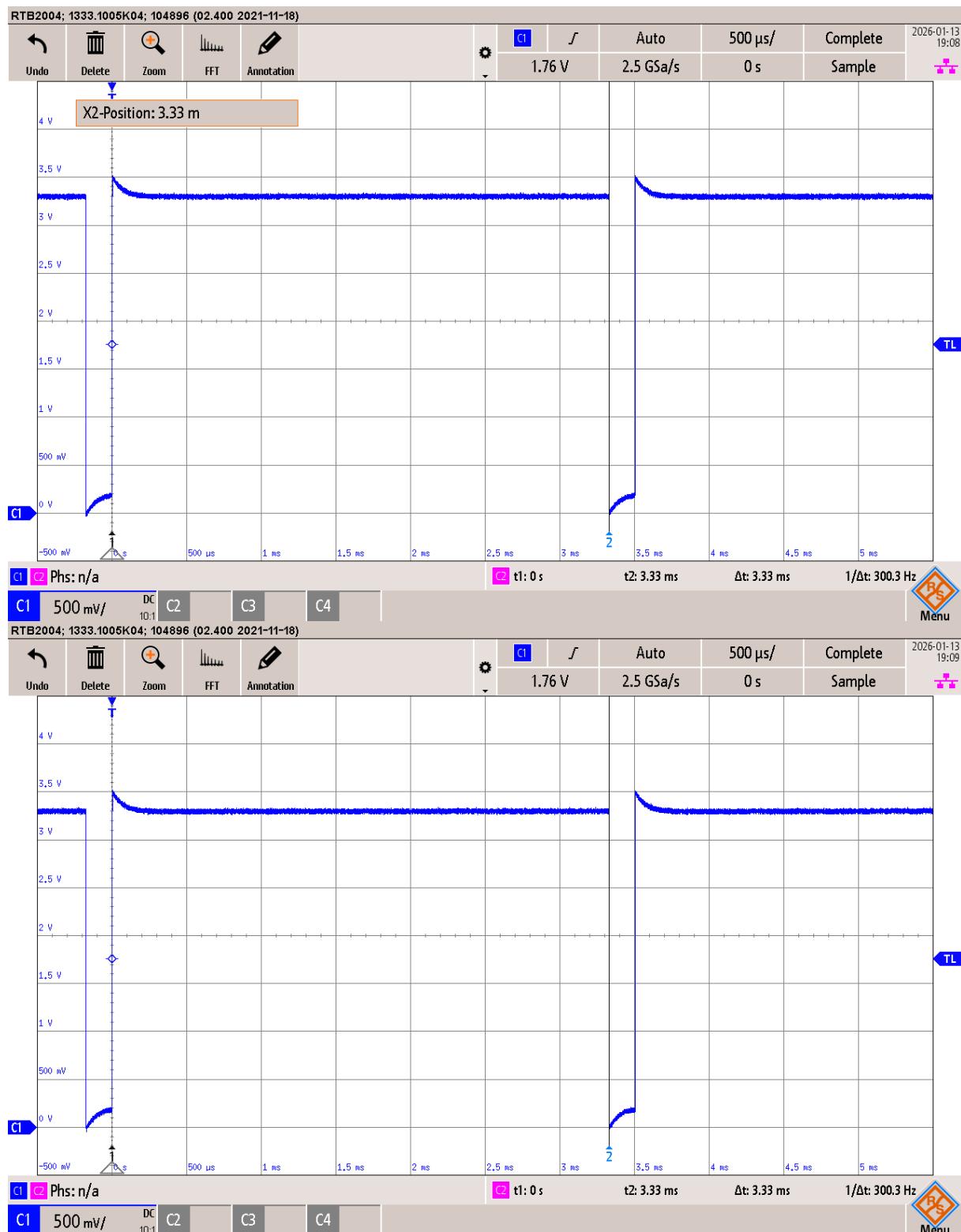
8.1.1.3 Mesure à 95% et -95%

RTB2004; 1333.1005K04; 104896 (02.400 2021-11-18)

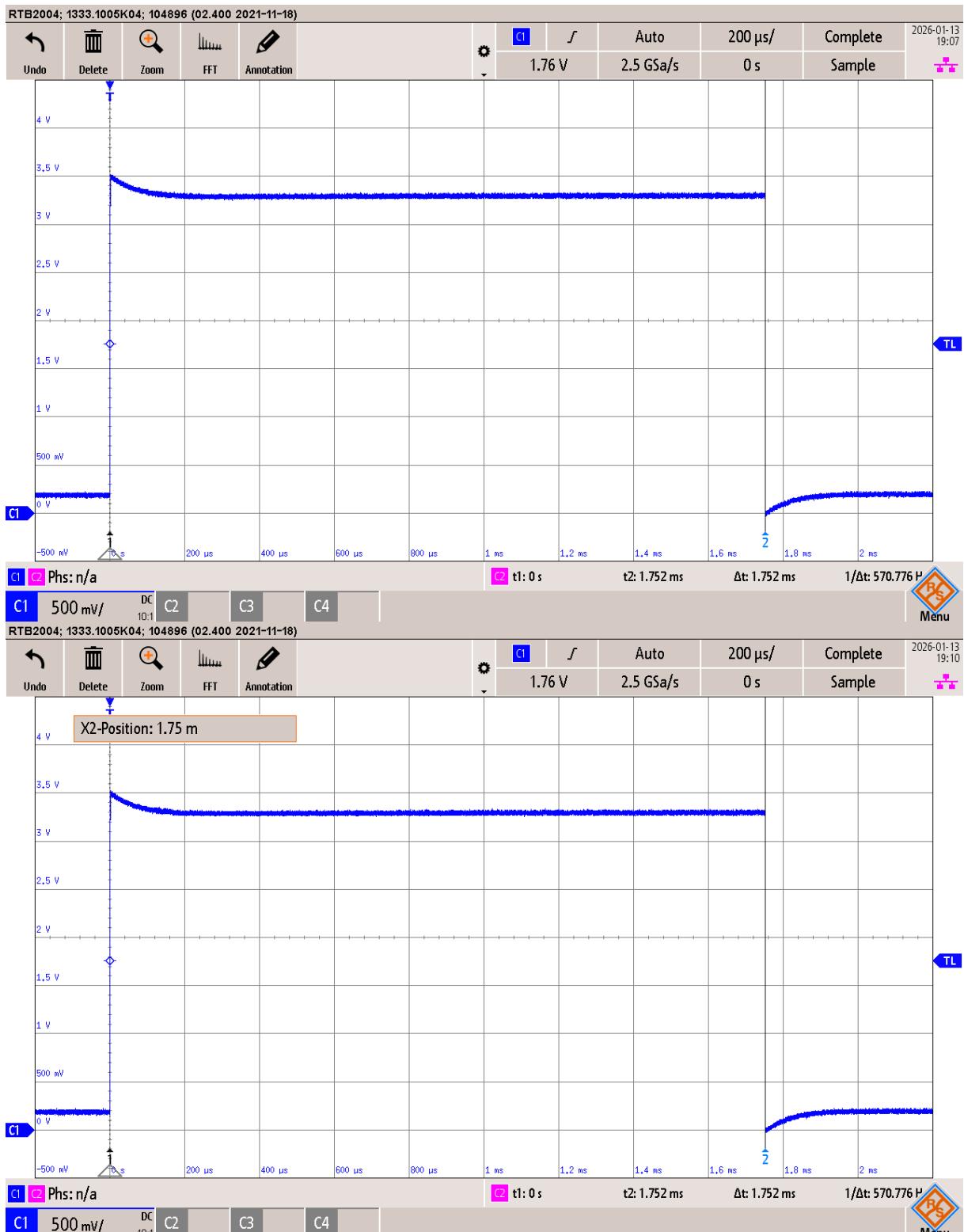


8.1.2 PWMSoft

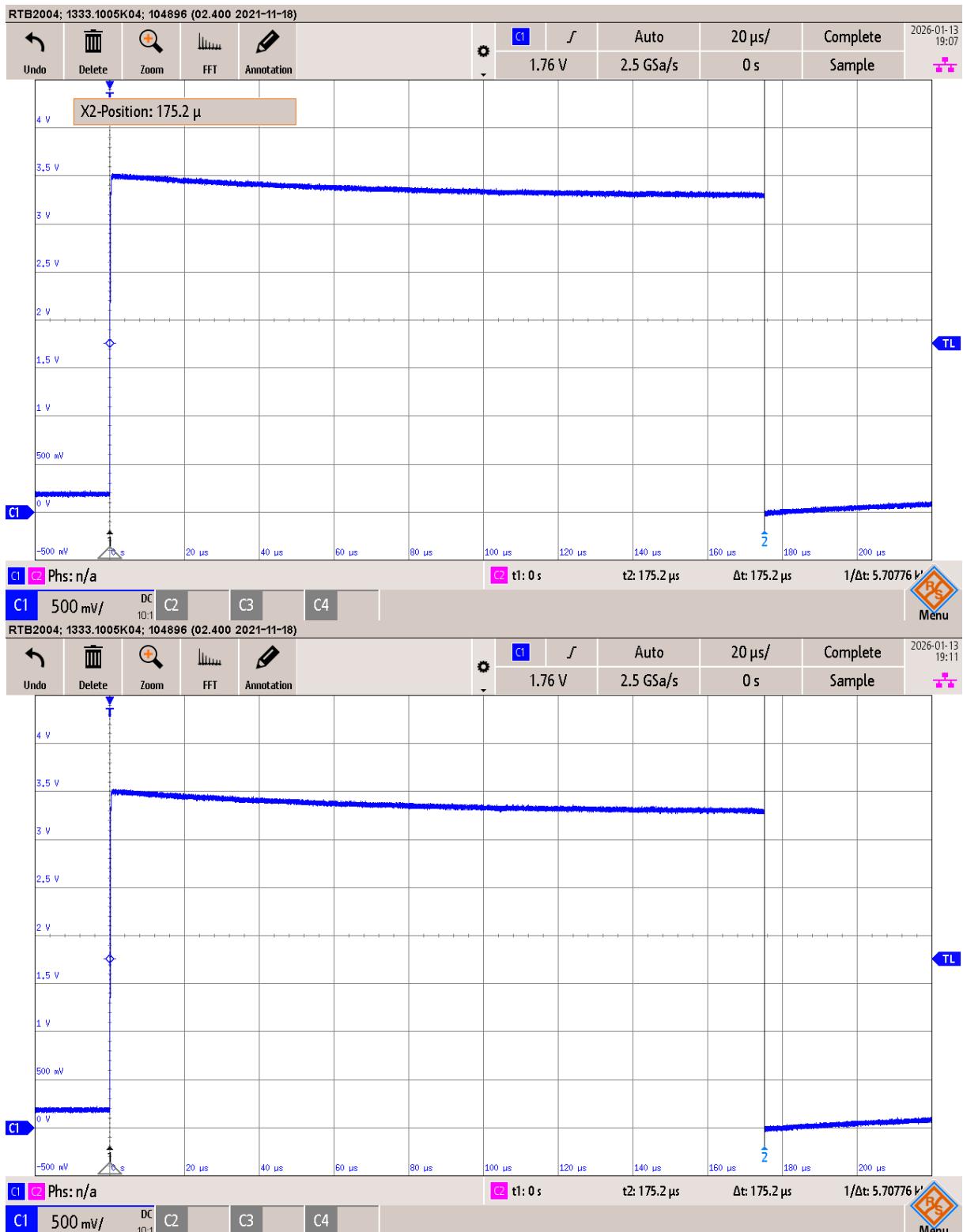
8.1.2.1 Mesure à 5% et -5%



8.1.2.2 Mesure à 50% et -50%

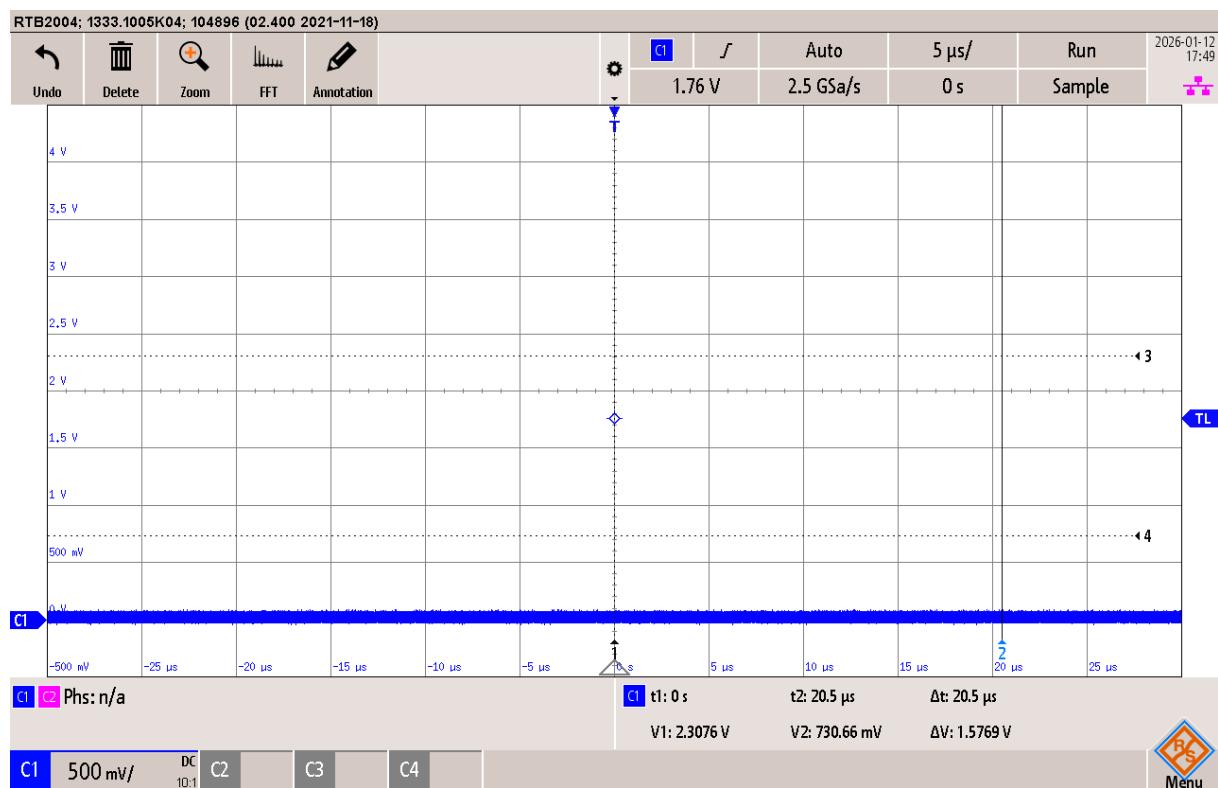


8.1.2.3 Mesure à 95% et -95%

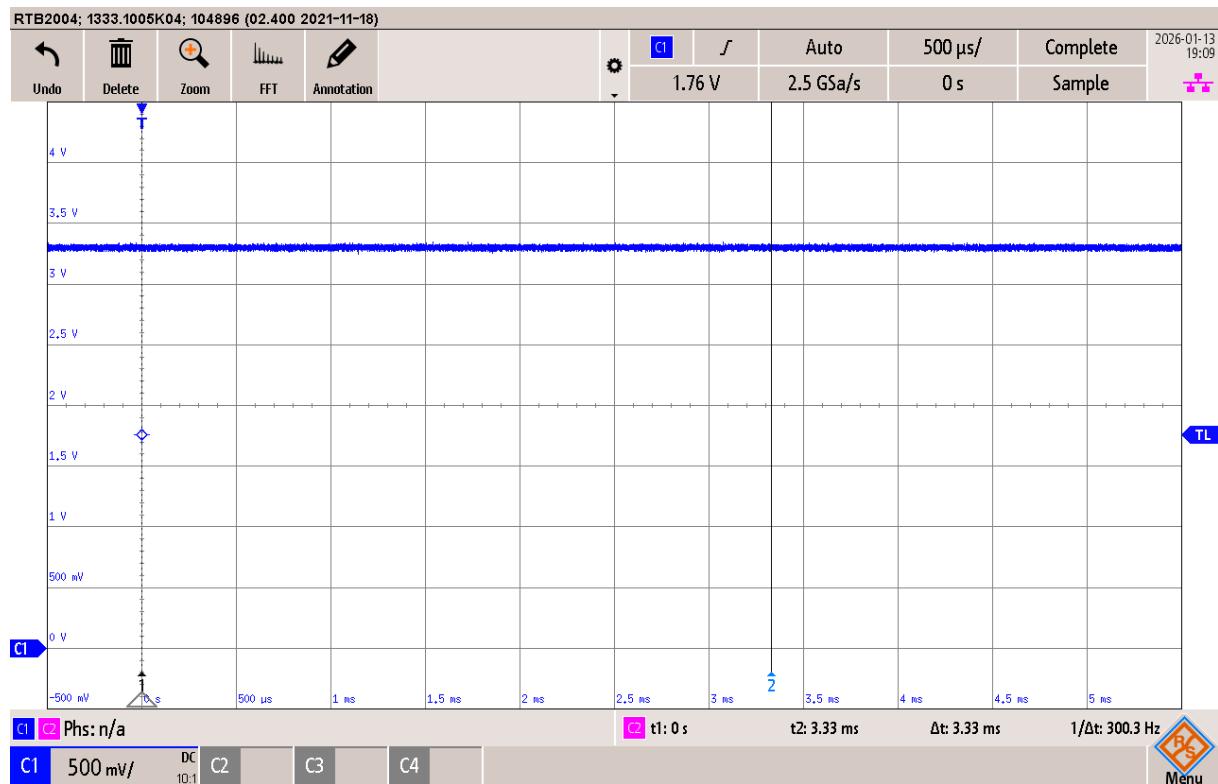


8.2 PWM à 0

8.2.1 Moteur DC

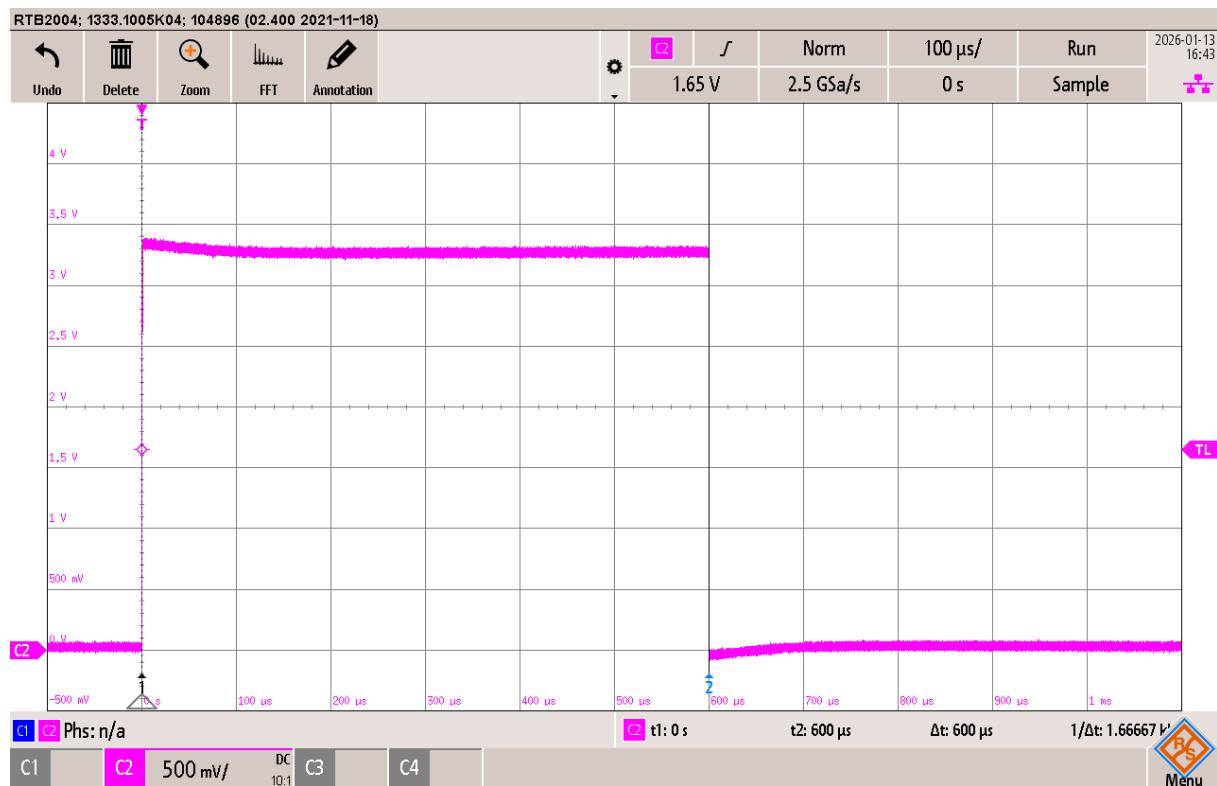


8.2.2 PWMSoft

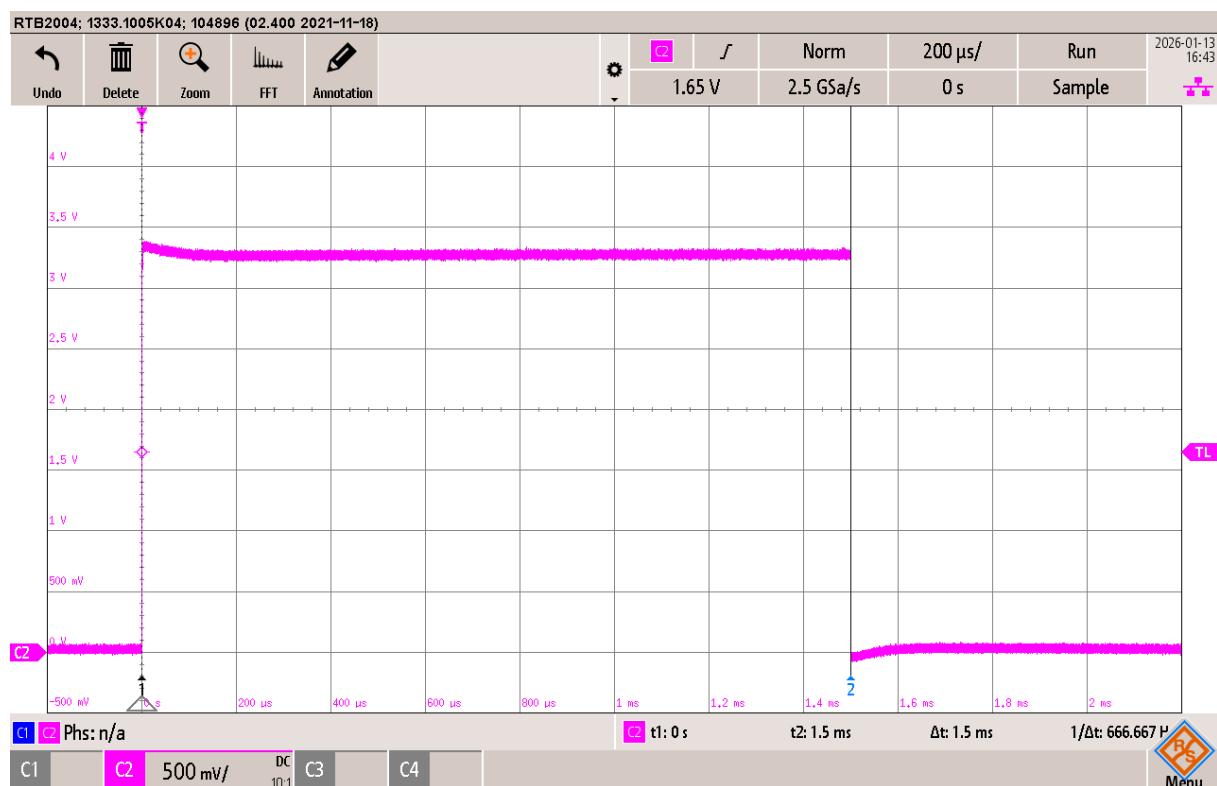


8.3 Mesures d'angle

8.3.1 -90°



8.3.2 0°



8.3.3 +90°

