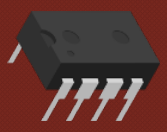


Les servo-moteurs avec PwmWrite.h

Comme pour les autres classes de MODULE, **PwmWrite.h** n'est pas dédiée à une application en particulier (bien que l'exemple qui suit concerne les servo-moteurs).

Plus généralement, j'utilise la génération de signaux **PWM** pour faire **varier la luminosité des diodes électroluminescentes** (voir l'exemple plus bas), ou encore pour piloter mon élévateur de tension à découpage de courant (qui prend en entrée un signal carré de 2000Hz) afin d'alimenter mon tube Geiger en +400V.

> *Avec la classe PwmWrite.h, **tous les ports de l'automate programmable** peuvent servir à la génération de signaux PWM (voir plus loin: Le choix du PWM matériel ou logiciel).*

L'exemple suivant permet de faire fonctionner un servo-moteur sur le port numéro **1** de l'automate programmable à une fréquence PWM de **50Hz**. Dans la boucle, le palonnier du servo-moteur se positionnera à **1000µs**, puis 3 secondes plus tard, se positionnera à **2000µs**, pour ensuite attendre de nouveau 3 secondes à cette position, après quoi le programme bouclera.

Exemple d'utilisation de PwmWrite.h:

```
#include "../module/1284p/PwmWrite.h"
#include "../module/1284p/Timer.h"

int main()
{
        PwmWrite myServo = PwmWrite (1);

        PwmWrite::start (50);

        while (true)
        {
                myServo.us (1000);
                Timer::pause (3000);
                //pause de 3 secondes

                myServo.us (2000);
                Timer::pause (3000);
                //pause de 3 secondes
        }

        return 0;
}
```

Un objet **myServo** de type **PwmWrite** est déclaré, en paramètre est indiqué d'utiliser le port numéro **1** de l'automate programmable en sortie. Puis la fonction statique **start** est appelée prenant en paramètre la fréquence du PWM, ce qui démarre la génération PWM à la fréquence de **50Hz** dans cet exemple.

> *La fréquence PWM indiquée en paramètre avec la fonction statique **start** sera toujours appliquée à l'ensemble des ports choisis (un seul compteur 16 bits du microcontrôleur étant dédié à cette fonction).*

Ensuite dans la boucle, dans un premier temps l'objet **myServo** appelle la fonction **us** qui prend en paramètre **1000**, la largeur d'impulsion en microsecondes, ce qui vient positionner le palonnier du servo-moteur à cet endroit. Puis 3000ms (3 secondes) plus tard, positionne le servo-moteur à **2000µs**.

> *Il vous est possible d'appeler la fonction **us** avant ou après avoir démarré et choisi la fréquence du PWM avec la fonction statique **start**. Ceci n'a en effet aucune incidence sur le fonctionnement puisque la fonction **us** retient les valeurs indiquées en paramètre lorsque que le PWM n'est pas encore démarré.*

La fonction **us** attend en paramètre un **nombre décimal** (float), ou entier si vous le souhaitez. Libre à vous de déterminer dans vos montages électroniques si vous avez besoin de générer des signaux PWM avec une largeur d'impulsion précise à la fraction de microseconde (c'est le cas de certains servo-moteurs ou contrôleurs de moteurs sans charbons), ou si vous n'avez besoin que d'une précision arrondie à l'entier.

> *À noter que la largeur d'impulsion des objets créés est à une valeur par défaut de **0 microseconde** si aucun changement de rapport cyclique n'a encore été effectué avec la fonction **us**.*

Ce programme est simple, mais il permet en quelques lignes de comprendre facilement les fonctions de la classe PwmWrite.h.

Faire varier la luminosité d'une del:

L'exemple suivant va vous montrer comment modifier la luminosité d'une del grâce au changement de la largeur d'impulsion d'un signal PWM.

En effet, notre rétine étant sensible à la persistance rétinienne, une simple **variation du temps d'exposition à la lumière** à une fréquence supérieure au rafraîchissement de la rétine, sera perçue par le cerveau comme une **variation de la luminosité**.

Exemple de variation de la luminosité d'une del avec un potentiomètre:

```
#include "../module/1284p/PwmWrite.h"
#include "../module/1284p/AnalogRead.h"
#include "../module/1284p/Math.h"

int main()
{
	PwmWrite led = PwmWrite (13);
	AnalogRead potentiometer = AnalogRead (25);

	PwmWrite::start (1000);

	while (true)
	{
		potentiometer.read();
		led.us (Math::curve (0, potentiometer.value, 1023, 0, 1000, 0));
	}

	return 0;
}
```

Dans l'exemple, la fréquence du PWM est de **1000Hz** (bien plus rapide que le rafraîchissement de la rétine), ce qui donne un rapport cyclique (largeur d'impulsion) pouvant varier de **0µs** (0%) à **1000µs** (100%).

La fonction statique **curve** de la classe **Math.h** permet d'interpoler la valeur 10 bits (0 à 1023) du potentiomètre en valeurs de **0µs** à **1000µs**. À une valeur de **500µs** (rapport cyclique de 50%), la del éclairerait à une luminosité d'environ 50%.

Le choix du PWM matériel ou logiciel:

Tous les ports de l'automate programmable peuvent générer des signaux PWM sur une précision théorique de 16 bits avec la classe PwmWrite.h (ce qui offre une grande souplesse d'utilisation), mais la façon dont ils sont générés (purement matérielle, ou partiellement logicielle) ainsi que les limites techniques (fréquences et précision) seront **différentes suivant les ports que vous utiliserez**.

Ports des automates programmables concernés par la génération de PWM 16 bits purement matérielle:

> ***Automate programmable MODULABLE M20:***
> *- Port 10 (PB1)*
> *- Port 11 (PB2)*
>
> ***Automate programmable MODULABLE M32:***
> *- Port 13 (PD4)*
> *- Port 14 (PD5)*

> *La sélection d'un autre port que ceux indiqués ci-dessus passera la génération PWM en mode **logiciel**.*

En mode **logiciel**, la génération de signaux PWM s'effectuant de manière **séquentielle** (ports après ports), **la fréquence PWM choisie** ainsi que **le nombre de ports utilisés** permettent de déterminer la largeur d'impulsion maximale possible avec une telle configuration.

Exemple de calcul du rapport cyclique maximum suivant la fréquence et le nombre de signaux PWM à générer en mode logiciel:

> *Fréquence du PWM = **1000Hz***
> *Nombre de signaux PWM à générer (ports utilisés) = **4 ports***
>
> ***Limites maximales:***
> *Rapport cyclique = 1000000µs / (1000Hz * 4 ports) = **250µs** (max)*

Dans cette configuration, la **largeur d'impulsion maximale** qui pourra être générée sera de **250 microsecondes** (rapport cyclique de 100%). Si vous indiquez un rapport cyclique supérieur à 250µs, ceci n'aura aucun impact sur le bon fonctionnement du PWM (celui-ci sera bloqué à 250µs par port utilisé dans tous les cas).

De façon matérielle, **PwmWrite.h** permet de générer des signaux PWM à des fréquences très élevées (supérieures à 1Mhz), mais vous devez prendre conscience que le nombre d'états possibles (précision) à de telles fréquences sera plus faible qu'à des fréquences inférieures. Ceci est dû aux contraintes même du matériel, en particulier du compteur 16 bits, de la fréquence d'horloge de 16Mhz, et des diviseurs d'horloge du microcontrôleur.

Exemple de calcul du rapport cyclique maximum suivant la fréquence PWM choisie en mode matériel:

> *Fréquence du PWM = **1000Hz***
>
> ***Limites maximales:***
> *Rapport cyclique = 1000000µs / 1000Hz = **1000µs** (max)*

En mode matériel, la largeur d'impulsion maximale n'est pas dépendante du nombre de ports utilisés.

> *À noter que la classe PwmWrite.h calculera **le diviseur d'horloge le plus adapté** en rapport avec la fréquence PWM choisie, ceci afin de garantir une précision de la largeur d'impulsion **la plus élevée possible**.*

Cette précision de 16 bits théorique maximum est bien entendu égale et valable pour les deux méthodes de génération du PWM (matérielle et logicielle). Seul des limites en terme de fréquence maximale du PWM distinguent l'une ou l'autre méthode.

> *Rappelez-vous que c'est la classe PwmWrite.h qui par vos choix en terme de ports utilisés, **sélectionne automatiquement pour vous** le mode de génération PWM matériel ou logiciel.*

Dans tous les cas, si un problème vient à se poser à vous en terme de fréquence maximale, de précision, ou de largeur d'impulsion maximale possible, il convient alors avant toute chose de comprendre et de s'interroger sur la logique matérielle du composant, et de relire ce que j'ai indiqué plus haut afin d'en maîtriser les contraintes et les limites techniques (ceci ne doit en aucun cas être un frein à vos projets).

Ports des automates programmables concernés par la génération du PWM:

Automate programmable MODULABLE M20:

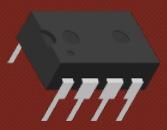
- Port 1 (PD0)
- Port 2 (PD1)
- Port 3 (PD2)
- Port 4 (PD3)
- Port 5 (PD4)
- Port 6 (PD5)
- Port 7 (PD6)
- Port 8 (PD7)
- Port 9 (PB0)
- Port 10 (PB1)
- Port 11 (PB2)
- Port 12 (PB3)
- Port 13 (PB4)
- Port 14 (PB5)
- Port 15 (PC0)
- Port 16 (PC1)
- Port 17 (PC2)
- Port 18 (PC3)
- Port 19 (PC4)
- Port 20 (PC5)

Automate programmable MODULABLE M32:

- Port 1 (PB0)
- Port 2 (PB1)
- Port 3 (PB2)
- Port 4 (PB3)
- Port 5 (PB4)
- Port 6 (PB5)
- Port 7 (PB6)
- Port 8 (PB7)
- Port 9 (PD0)
- Port 10 (PD1)
- Port 11 (PD2)
- Port 12 (PD3)
- Port 13 (PD4)
- Port 14 (PD5)
- Port 15 (PD6)
- Port 16 (PD7)
- Port 17 (PC0)
- Port 18 (PC1)
- Port 19 (PC2)
- Port 20 (PC3)
- Port 21 (PC4)
- Port 22 (PC5)
- Port 23 (PC6)
- Port 24 (PC7)
- Port 25 (PA7)
- Port 26 (PA6)
- Port 27 (PA5)
- Port 28 (PA4)
- Port 29 (PA3)
- Port 30 (PA2)
- Port 31 (PA1)
- Port 32 (PA0)

Récapitulatif des fonctions de cette classe:

```
PwmWrite (const unsigned char PIN);
static void start (const unsigned long FREQUENCY);
void us (const float US);
```



www.sylvainmahe.xyz

LE BLOG
de Sylvain Mahé
contact@sylvainmahe.xyz

Retour
Suite