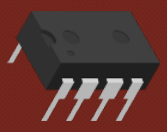

www.sylvainmahe.xyz

LE BLOG
de Sylvain Mahé
contact@sylvainmahe.xyz

Les interruptions avec InterruptRead.h

Les interruptions permettent **d'interrompre le déroulement normal d'un programme** afin d'effectuer une tâche à un moment précis, puis de continuer l'exécution du programme la ou il c'était arrêté.

La classe **InterruptRead.h** permet de détecter un **front montant et/ou descendant** (passage de 0V à +5V ou inversement) sur les broches du microcontrôleur concernées par les interruptions.

Ports des automates programmables concernés par les interruptions:

> ***Automate programmable MODULABLE M20:***
> *- Port 3 (PD2)*
> *- Port 4 (PD3)*
>
> ***Automate programmable MODULABLE M32:***
> *- Port 3 (PB2)*
> *- Port 11 (PD2)*
> *- Port 12 (PD3)*

L'utilité d'un tel système est de pouvoir **déceler un changement d'état d'un port du microcontrôleur au moment même ou il se produit**. Il n'est alors plus nécessaire d'attendre une lecture de l'état du port concerné au sein même d'une boucle de vérification, comme c'est le cas avec la classe **GpioRead.h** (cette dernière solution produisant une latence suivant la vitesse d'exécution totale de votre boucle de calcul).

> *Grâce à ce système, il vous est par exemple possible de quantifier et de mesurer précisément la périodicité d'un phénomène physique externe qui changerait spontanément d'état.*

Exemple d'utilisation de InterruptRead.h:

```
#include "../module/1284p/InterruptRead.h"

void event()
{
	//un front montant a été détecté sur le port 3
}

int main()
{
	InterruptRead myInterrupt = InterruptRead (3);

	myInterrupt.start (event, true, false);

	while (true)
	{
	}

	return 0;
}
```

Dans cet exemple, un objet **myInterrupt** de type **InterruptRead** est déclaré, en paramètre est indiqué d'utiliser le port numéro **3** de l'automate programmable en entrée, puis afin de démarrer la lecture du port concerné, cet objet **myInterrupt** appelle la fonction **start** prenant plusieurs paramètres:
- Le 1er paramètre **event** est le nom de la fonction déclarée en amont qui va servir à exécuter du code quand l'interruption interviendra.
- Le 2ème paramètre **true** permet de détecter les fronts montants sur le port.
- Le 3ème paramètre **false** indique de ne pas détecter les front descendants sur le port.

> *Même si vous n'avez pas besoin d'exécuter des instructions dans la boucle **while**, cette dernière reste importante car elle permet au microcontrôleur de continuer l'exécution du code (qu'il soit une interruption ou pas).*

> *Dans le cas contraire, l'exécution terminerait après **return 0;**, et le microcontrôleur s'arrêterait la de sorte que plus aucune interruption ne pourrait être exécutée !*

Cette classe dispose également d'une fonction **stop** qui permet de stopper la lecture des fronts montants et/ou descendants sur le port concerné.

Récapitulatif des fonctions de cette classe:

```
InterruptRead (const unsigned char PIN);
void start (void functionJump(), const bool RISING, const bool FALLING);
void stop();
```

Mesure de la périodicité d'un phénomène:

La spontanéité du principe même des interruptions (quelques cycles d'horloges tout au plus) permet de mesurer avec précision un phénomène périodique.

Exemple de mesure d'un phénomène périodique:

```
#include "../module/1284p/InterruptRead.h"
#include "../module/1284p/Period.h"

Period _myPeriod = Period();

void event()
{
        _myPeriod.state();
}

int main()
{
        InterruptRead myInterrupt = InterruptRead (3);

        _myPeriod.start();
        myInterrupt.start (event, true, false);

        while (true)
        {
                //_myPeriod.us est la valeur mesurée avec une précision d'1 microsecond
        }

        return 0;
}
```

Dans cet exemple j'utilise la classe **Period.h** qui est dédiée à ce genre d'opération spécifique, mais il est tout à fait possible d'utiliser la classe **Timer.h** qui via un simple calcul du temps mesuré associé à une petite logique, remplirait la même fonction. **Period.h** offre l'avantage d'avoir ce principe intégré dans son fonctionnement ce qui simplifie la programmation.

> *Ce programme pourrait être associé à un capteur à effet hall connecté au port 3 de l'automate programmable, ce qui permettrait par exemple de mesurer la période de rotation d'un moteur électrique ou thermique (et d'en déduire le nombre de tours par minute).*

De nombreuses applications et montages sont possibles avec les interruptions, pour ne citer encore qu'un exemple, c'est avec ce principe que je détecte la fermeture du tube sur mon compteur Geiger, ce qui permet ensuite d'extrapoler et de quantifier la radioactivité par mesure du temps écoulé.
