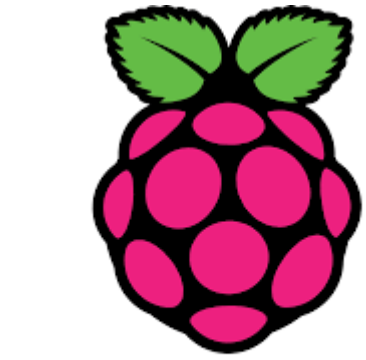
Raspberry Pi

GPIO

Table des matières

Le Raspberry Pi est un nano-ordinateur monocarte à processeur ARM conçu par le créateur de jeux vidéo David Braben, dans le cadre de sa fondation Raspberry Pi.



RaspberryPi

1. Entrées/sorties GPIO

Les ports GPIO[1] sont des ports qui sont très utilisés dans le monde des microcontrôleurs, en particulier dans le domaine de l'électronique embarquée. Selon la configuration, ces ports peuvent fonctionner aussi bien en entrée qu'en sortie.

Les périphériques GPIO comportent un ensemble de ports d'entrée/sortie qui peuvent être configurés pour jouer soit le rôle d'une entrée, soit le rôle d'une sortie.

La carte Raspberry Pi donne accès à des entrées et sorties numériques appelées GPIO contrôlées par le processeur ARM.

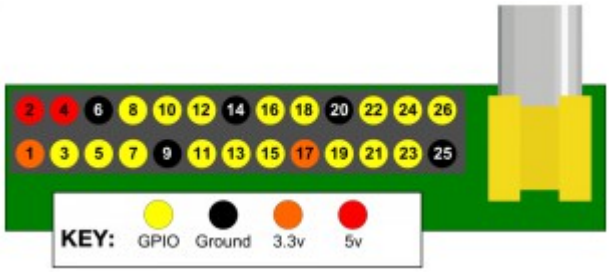
Elles sont à usage multiple :

- en entrée numérique tout ou rien, pour détecter un interrupteur par exemple
- en sortie numérique tout ou rien, pour activer un relais par exemple
- en sortie numérique MLI[2], pour contrôler la puissance moyenne d'une DEL[3] par exemple
- en protocole I2C, d'échanger avec une ou plusieurs puces
- en protocole SPI, idem
- en protocole UART, d'échanger avec une seule puce (ou un PC)

D'autres usages sont possibles (audio PCM, vidéo sur les connecteurs DSI et CSI).

Plusieurs connecteurs donnent accès aux GPIO, mais le principal est un connecteur comportant 2 rangées de 13 picots mâles.

Le connecteur GPIO ci-dessous est composé de 26 broches dont 17 sont dédiées au GPIO (permets des changements d'état on/off). Les autres broches ont le rôle d'alimentation (3.3V et 5V) et de masse (ground).



NB : le numéro de GPIO n'est pas sa position sur le connecteur, mais son numéro dans les registres de la puce ARM BCM2835. C'est donc celui qui sera utilisé dans la plupart des bibliothèques d'accès.

1 General Purpose Input/Output
2 Modulation à Largeur d'Impulsion
3 Diode Électro Luminescente

wiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	wiringPi Pin
–	–	3.3v	1 | 2	5v	–	–
8	0	SDA0	3 | 4	DNC	–	–
9	1	SCL0	5 | 6	0V	–	–
7	4	GPIO 7	7 | 8	TxD	14	15
–	–	DNC	9 | 10	RxD	15	16
0	17	GPIO 0	11 | 12	GPIO 1	18	1
2	21	GPIO 2	13 | 14	DNC	–	–
3	22	GPIO 3	15 | 16	GPIO 4	23	4
–	–	DNC	17 | 18	GPIO 5	24	5
12	10	MOSI	19 | 20	DNC	–	–
13	9	MISO	21 | 22	GPIO 6	25	6
14	11	SCLK	23 | 24	CE0	8	10
–	–	DNC	25 | 26	CE1	7	11
wiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	wiringPi Pin

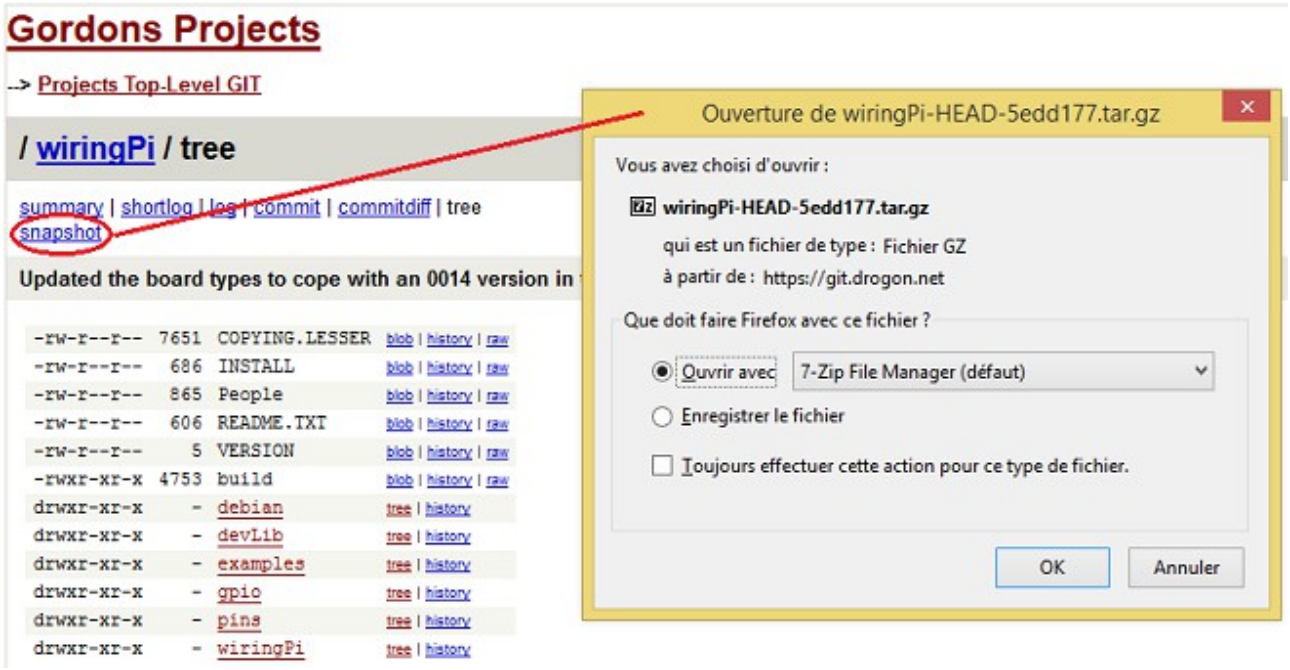
Les numéros centraux sont les positions physiques des PIN, les numéro de la colonne “wiringPI Pin” sont les correspondances à appeler dans les lignes de commandes.

2. Accéder aux GPIO

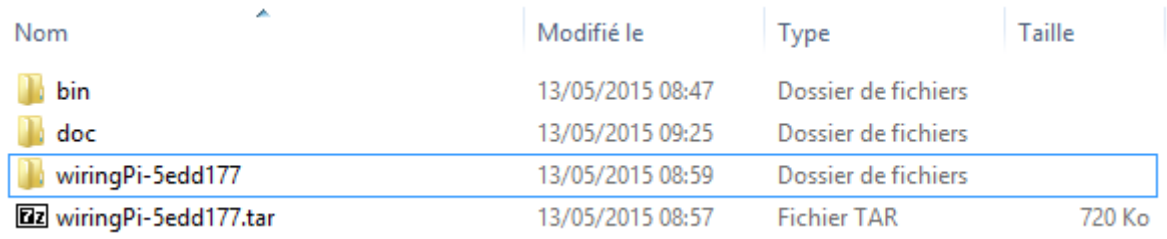
Pour accéder aux GPIO depuis le shell en ligne de commande, nous utiliserons WiringPi, une bibliothèque C et des outils de compilation.

2.1. Installation du programme gpio

Il faut d'abord télécharger l'archive depuis cette adresse. Puis cliquer sur le lien Snapshot.



Enregistrer ensuite cette archive, la décompresser (avec 7zip par exemple) puis transférer le dossier qui a été extrait sur la carte Raspberry (wiringPi-5edd177 dans notre cas).



Nom	Modifié le	Type	Taille
bin	13/05/2015 08:47	Dossier de fichiers	
doc	13/05/2015 09:25	Dossier de fichiers	
wiringPi-5edd177	13/05/2015 08:59	Dossier de fichiers	
wiringPi-5edd177.tar	13/05/2015 08:57	Fichier TAR	720 Ko

en décompressant l'archive on a extrait un dossier

Se connecter ensuite sur Raspberry avec PuTTY, se déplacer dans le dossier wiringPi-5edd177 et lancer la compilation du programme gpio.

```
cd wiring*
sudo sh ./build
```

A la fin de la compilation, un exécutable a été créé dans le dossier ~/wiringPi-5edd177/gpio.

Toutes les bibliothèques et les fichiers header ont également été créés et copier dans les bons répertoires.

```
pi@raspberrypi ~/wiringPi-5edd177/gpio $ ls -l
total 156
-rw-r--r-- 1 pi   pi     7651 mars   8 16:59 COPYING.LESSER
-rwxr-xr-x 1 root root 30289 mai    1 18:45 gpio
-rw-r--r-- 1 pi   pi     9534 mars   8 16:59 gpio.1
-rw-r--r-- 1 pi   pi    32755 mars   8 16:59 gpio.c
-rw-r--r-- 1 root root 22488 mai    1 18:44 gpio.o
-rw-r--r-- 1 pi   pi     2460 mars   8 16:59 Makefile
-rw-r--r-- 1 pi   pi     1134 mars   8 16:59 newVersion
-rw-r--r-- 1 pi   pi     1252 mars   8 16:59 pins.c
-rw-r--r-- 1 root root   1196 mai    1 18:44 pins.o
-rw-r--r-- 1 pi   pi     4084 mars   8 16:59 pintest
-rw-r--r-- 1 pi   pi     8856 mars   8 16:59 readall.c
-rw-r--r-- 1 root root   6140 mai    1 18:44 readall.o
-rw-r--r-- 1 pi   pi     1262 mars   8 16:59 test.sh
-rw-r--r-- 1 pi   pi       23 mars   8 16:59 version.h
```

Le programme a été créé avec comme propriétaire et groupe root. Il faut changer les droits par sécurité :

```
sudo chown pi *
sudo chgrp pi *
```

Puis tester le programme en affichant l'usage :

```
gpio -v
```

ou en affichant les PIN des entrées / sorties :

```
gpio readall
```

Selon l'explication donnée dans le manuel, on pourra utiliser de manière équivalente les instructions suivantes :

- gpio mode 4 out # définit la PIN 4 en sortie (écriture)
- gpio write 4 1 # positionne la PIN 4 à l'état haut
- gpio write 4 0 # positionne la PIN 4 à l'état bas
- gpio mode 7 in # définit la PIN 7 en entrée (lecture)
- gpio read 7 # lecture de la PIN 7
- gpio mode 1 pwm # définit la PIN 1 en sortie PWM
- gpio pwm 1 1023 # positionne la PIN 1 à 1023 (0 à 1023)

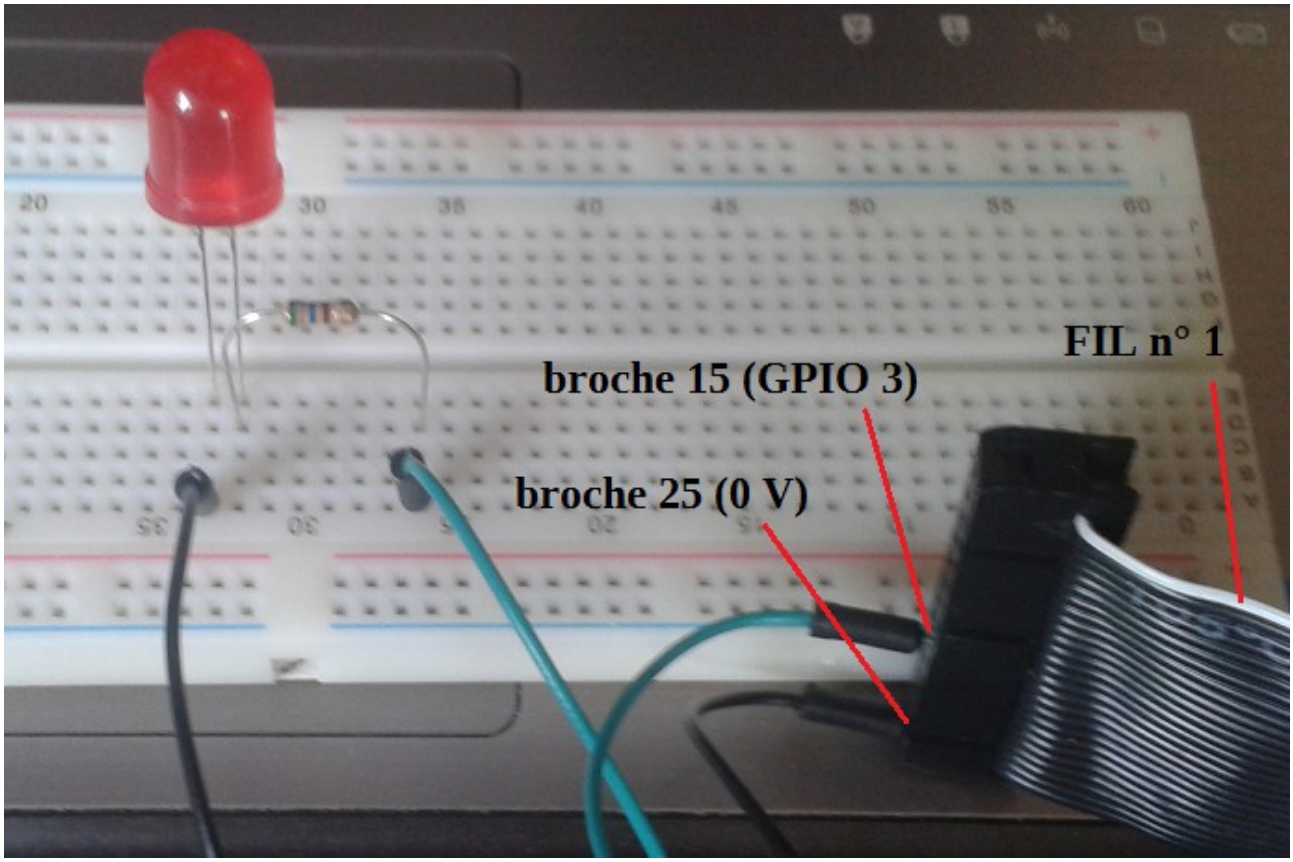
Pour accéder au manuel taper la commande suivante en mode console :

```
man gpio
```

2.2. Accéder au GPIO en shell

Exemple 1 : on connecte la Raspberry au montage ci-contre, avec un signal d'entrée/sortie (interrupteur ou LED) sur la patte 15 du connecteur, correspondant à la GPIO 3 de la Raspberry Pi. Le programme va faire clignoter la LED à une fréquence de 60 Hz (équivalent du programme blink sous Arduino).

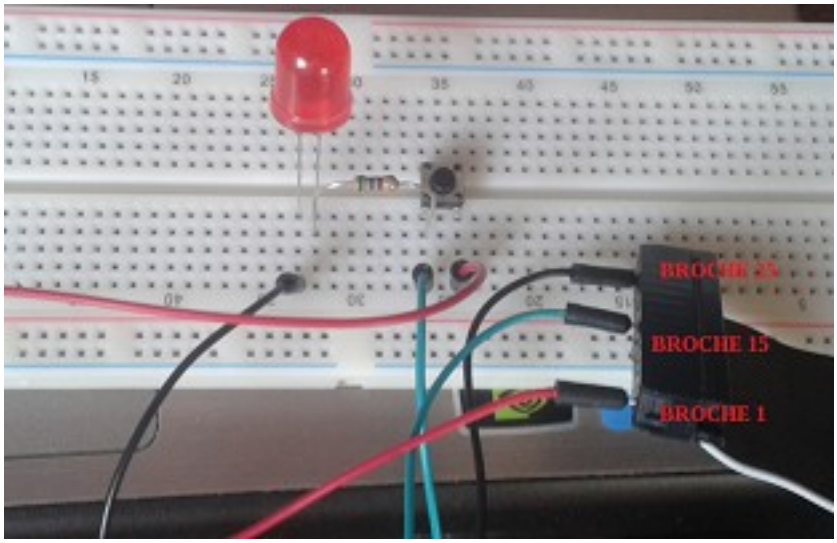
Voir la vidéo 

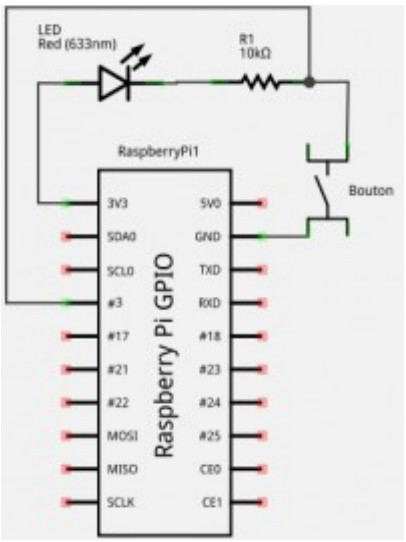


Voici le script en bash :

```
#!/bin/bash

#---- déclaration des variables globales ----#
PIN=3

#---- initialisation programme ----#
setup()
{
        # définit la broche 3 en sortie
        gpio mode $PIN out
        echo "setup done..."
}

#---- boucle du programme ----#
loop()
{
        gpio write $PIN 1
        sleep 0.5        # attente 500 ms
        gpio write $PIN 0
        sleep 0.5        # attente 500 ms
}

#==== programme principal ====#
setup

while true;
do
        loop
done
```

Exemple 2 : on connecte la Raspberry au montage ci-contre, avec un signal d'entrée/sortie (interrupteur ou LED) sur la patte 15 du connecteur, correspondant à la GPIO 3 de la Raspberry Pi. Le programme détecte si le bouton poussoir est pressé ou non ; une LED sert de témoin.





Voici le script en bash :

```
#!/bin/bash

#---- déclaration des variables globales ----#
PIN=3

#---- initialisation programme ----#
setup()
{
        # définit la broche 3 en lecture
        gpio mode $PIN in
        echo "setup PIN$PIN done..."
}

#---- boucle du programme ----#
loop()
{
        if [ `gpio read $PIN` = 0 ]
        then
                echo "wait for button"
        else
                echo "pressed"
        fi

        sleep 0.1         # attente 100 ms
}

#==== programme principal ====#
setup

while true;
do
        loop
done
```

Voir la vidéo 

2.3. Accéder au GPIO par programme

La bibliothèque fournit avec WiringPi permet également de programmer en langage C/C++. Voici un exemple du programme blink.

L'architecture du programme se calque sur celle utilisée par Arduino avec une fonction d'initialisation (setup) et une boucle qui se répète à l'infini (loop). Ce sont les corps de ces deux fonctions qu'il faut modifier en plus de la partie déclarative des constantes pour les entrée/sortie (PIN).

Le code source :

```
#include <stdio.h>
#include <wiringPi.h>

// définition des entrée/sortie
const int    LED = 3;     // LED Pin - wiringPi pin 3 is BCM_GPIO 15
```

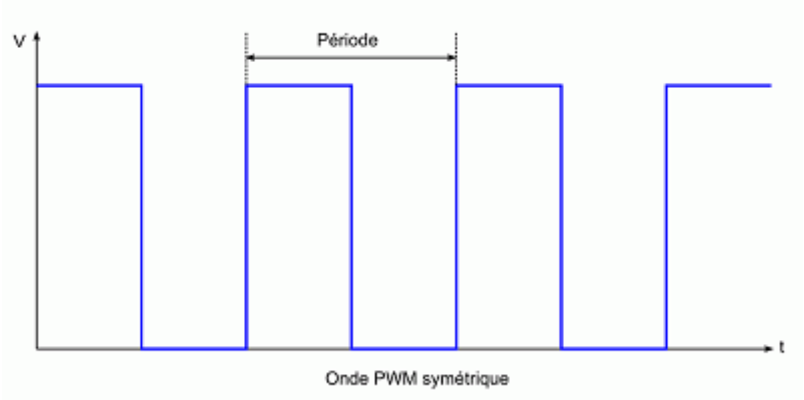
```
// s'exécute une seule fois
void setup()
{
	printf("Raspberry Pi blink\n") ;
	pinMode(LED, OUTPUT) ;
}

// s'exécute en boucle
void loop()
{
	digitalWrite(LED, HIGH);  // On
	delay (500);              // 500 ms
	digitalWrite(LED, LOW);   // Off
	delay (500);
}

//==== ne pas modifier ====//
int main (void)
{
	wiringPiSetup();    // doit toujours être déclarée

	setup();            // initialisation

	for (;;)
		loop();     // programme en boucle

  return;
}
```

Pour compiler le programme blink.c, il faut lancer la commande make et ensuite lancer l'exécution du programme compilé par sudo car la fonction `wiringPiSetup()` ne fonctionne qu'en mode super utilisateur.

```
make blink
sudo ./blink
```

3. PWM

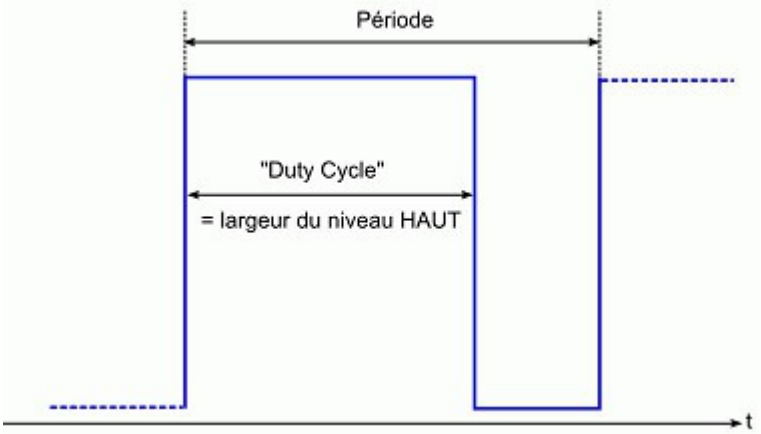
Une impulsion PWM[4] (ou MLI[5]) est une impulsion carrée qui se répète dans le temps de façon continue.



Onde PWM symétrique

4 Pulse Width Modulation
5 Modulation de Largeur d'Impulsion

Une telle onde, comme toute onde est caractérisée par sa période, c'est à dire la durée entre 2 impulsions. La définition d'une onde PWM se fait en modifiant la largeur du niveau HAUT par rapport à la durée de la période : la largeur du niveau HAUT est appelée « duty cycle ».



3.1. La librairie standard

La carte Raspberry dispose uniquement de sorties TOR et, contrairement à l'Arduino, Raspberry Pi ne comporte pas la moindre entrée analogique ; seule la PIN 1 (broche 12) supporte la PWM.

L'instruction pwmWrite(pin, largeur) permet de générer une impulsion PWM avec une largeur de « duty cycle » voulue :

- si valeur = 0, le duty cycle sera de 0% de la période
- si valeur = 1023, le duty cycle sera de 100% de la période
- si valeur = 512, le duty cycle sera de 50% de la période
- si valeur = n, le duty cycle sera de (n/255)*100% de la période.

Exemple : on fait varier la luminosité de la LED du 1er montage. La plage utilisée sera de 0 à 512 (sur un maximum de 1023) pour limiter le courant dans la diode.

Voici le code source :

```
#include <stdio.h>
#include <wiringPi.h>

// définition des entrée/sortie
const int    LED = 1;     // doit être la PIN 1

// s'exécute une seule fois
void setup()
{
      printf("PWM\n") ;
      pinMode(LED, PWM_OUTPUT) ;
}

// s'exécute en boucle
void loop()
{
      int i;
      for (i = 0; i < 512; i++) {
            pwmWrite(LED, i);   // plage 0..1023
            delay(2);                  // 2 ms
            }

      for (i = 512; i > 0; i--) {
```

```
            pwmWrite(LED, i);
            delay(2);                   // 2 ms
            }
}

//==== ne pas modifier ====
int main()
{
      wiringPiSetup();    // doit toujours être déclarée

      setup();            // initialisation

      for (;;)
            loop();       // programme en boucle

      return 0;
}
```

Voir la vidéo 

3.2. La bibliothèque PWM

WiringPi comprend un gestionnaire de PWM piloté par logiciel capable de fournir en sortie un signal PWM sur l'un des broches GPIO du Raspberry Pi.

Il y a quelques limitations... Pour maintenir une faible utilisation du CPU, la largeur d'impulsion minimum est de 100 µs.

Il faut inclure le header `<softPwm.h>` pour utiliser les fonctions de la libraire.

Deux fonctions suivantes sont disponibles :

- `int softPwmCreate(int broche, int initialValue, int pwmRange);`

  Cela crée une broche PWM contrôlée par logiciel. Vous pouvez utiliser toutes les broches GPIO. Si vous utilisez une plage de 100 pour la pwmRange, alors la valeur aller de 0 (désactivé) à 100 (valeur maxi) pour la broche.

  En cas de succès, la valeur de retour est 0 sinon vérifier la variable globale errno.

- `void softPwmWrite(int broche, int valeur);`

  Cela met à jour la valeur PWM sur la broche donnée. La valeur est vérifiée pour être dans la plage et les broches qui n'ont pas été initialisées via softPwmCreate seront ignorées.

Remarques :

- Chaque «cycle» de la sortie PWM prend 10ms avec une valeur de plage par défaut de 100, et essayer de changer la valeur de PWM plus de 100 fois par seconde serait futile.
- Chaque broche activée en mode softPWM utilise environ 0,5% du CPU.
- Il n'y a aucun moyen de désactiver la fonction softPWM sur une broche quand le programme est en cours d'exécution.
- Le programme doit constamment tourner pour maintenir une sortie PWM.

Le code précédent devient :

```
#include <stdio.h>
#include <wiringPi.h>
#include <softPwm.h>

// définition des entrée/sortie
const int    LED = 3;

// s'exécute une seule fois
void setup()
{
       printf("PWM\n");
       softPwmCreate(LED, 0, 512);
}

// s'exécute en boucle
void loop()
{
       int i;
       for (i = 0; i < 512; i++) {
              softPwmWrite (LED, i) ;
              delay(10);                // 10 ms
              }

       for (i = 512; i > 0; i--) {
              softPwmWrite (LED, i) ;
              delay(10);                // 10 ms
              }
}

//==== ne pas modifier ====
int main (void)
{
       wiringPiSetup();    // doit toujours être déclarée

       setup();            // initialisation

       for (;;)
              loop();      // programme en boucle

       return 0;
}
```