

Chapitre 3

GPIO : Les entrées/sorties à usage général

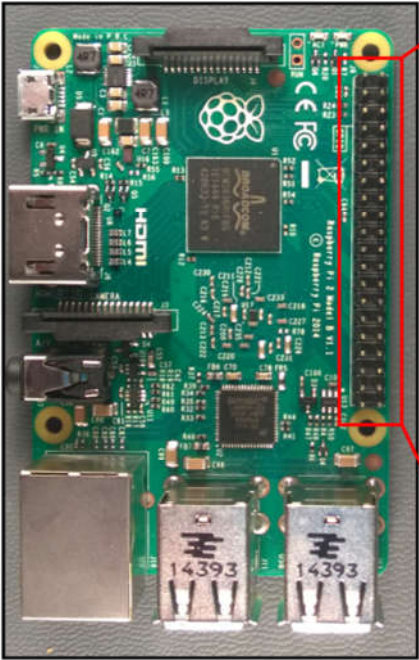
Dans le reste du chapitre Rspi désignera une raspberry pi, RspiX désignera le modèle X de la raspberry comme Rspi3 : raspberry pi 3

1. Introduction au GPIOs:

La Rspi est considéré comme un **Pocket PC** (PC de poche) muni d'un système d'exploitation GNU/Linux (Distribution officielle Raspbian par exemple), avec une interface graphique de type bureau (Desktop). Munie d'un moniteur/clavier/souris ou d'un moniteur tactile peut donc être utilisé **comme un PC ou une tablette**. Toutefois, ce qui fait la **différence** par rapport à un PC ou une tablette, à part la **taille**, la **consommation**, est le fait qu'elle met à la disposition de l'utilisateur **un connecteur J8 de 40 broches** (Fig.3.1), contenant :

- Des broches **d'entrée/sorties** à usage général (General Purpose Input/Output GPIO).
- Des broches d'alimentation (3.3V, 5V, GND).

qui lui procure la **capacité d'être intégrée** dans des projets de conception/prototypage rapide de **systèmes embarqués**.



Alternate Function	BCM Num	Physic Num	BCM Num	Alternate Function
	3.3V PWR	1	2	5V PWR
I2C1 SDA	GPIO 2	3	4	5V PWR
I2C1 SCL	GPIO 3	5	6	GND
	GPIO 4	7	8	UART0 TX
	GND	9	10	UART0 RX
	GPIO 17	11	12	GPIO 18
	GPIO 27	13	14	GND
	GPIO 22	15	16	GPIO 23
	3.3V PWR	17	18	GPIO 24
SPI0 MOSI	GPIO 10	19	20	GND
SPI0 MISO	GPIO 9	21	22	GPIO 25
SPI0 SCLK	GPIO 11	23	24	GPIO 8
	GND	25	26	GPIO 7
	Reserved	27	28	Reserved
	GPIO 5	29	30	GND
	GPIO 6	31	32	GPIO 12
	GPIO 13	33	34	GND
SPI1 MISO	GPIO 19	35	36	GPIO 16
	GPIO 26	37	38	GPIO 20
	GND	39	40	GPIO 21
				SPI1 CS0
				SPI1 CS1
				SPI1 CS0
				SPI1 MOSI
				SPI1 SCLK

Figure 3.1 Les GPIO de la Rspi avec leur numérotation BCM (Broadcom) et physique

La figure 3.2 donne le schéma du connecteur J8 avec une mise en garde contre l'utilisation de deux pins ID_SD et ID_SC (Broches 27 et 28) qui sont réservées au démarrage à l'identification d'une carte d'extension attaché à la Rspi.

NB : Comme le montre la figure 3.1 chaque GPIO peut être désignée par son numéro physique qui indique l'emplacement dans le connecteur, et son numéro sur le SOC (BCM choisi par le constructeur Broadcom). Les deux numéros ne sont pas identiques. Il faut spécifier la numérotation utilisée pour éviter toute ambiguïté. Exemple dans la figure suivante on utilise la numérotation Broadcom et on voit que les numéros ne respectent pas un ordre physique.

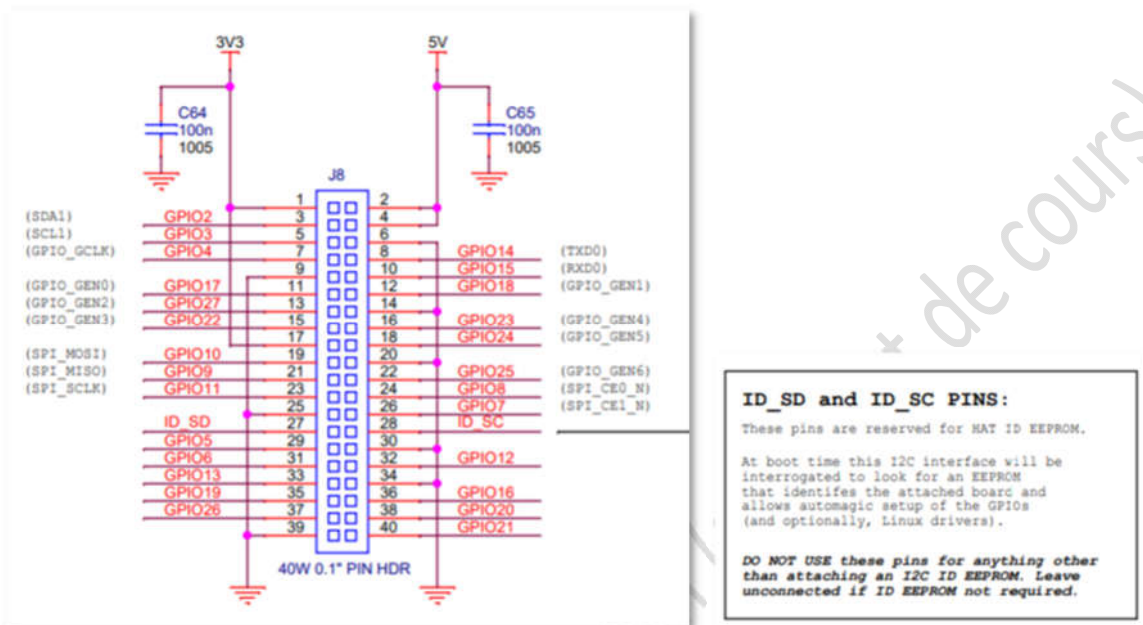
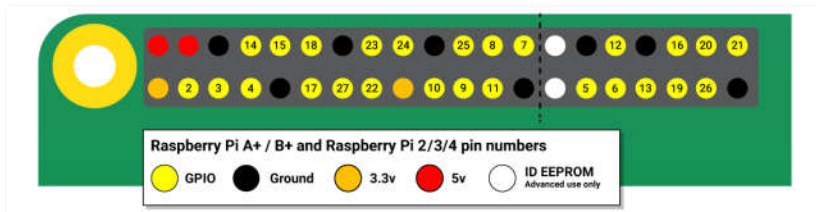


Figure 3.3 Schéma du connecteur J8 avec une mise en garde sur l'utilisation des pins ID_SD et ID_SC

NB : Quand une sortie GPIO est à 1 son potentiel est +3.3V et quand elle est à 0 son potentiel est à 0V. De même une entrée GPIO doit recevoir des potentiels +3.3V ou 0V. Une entrée attaquée par un niveau logique TTL de 5V peut détruire directement la puce SOC (System On Chip de Broadcom) BCM283X (X=5,6,ou 7 selon modèle de la Rspi) ou BCM2711 pour Rspi4 et rendre la Rspi inutilisable ! La figure 3.3 montre que le connecteur J8 est directement lié au SOC BCM2835, sans aucune protection supplémentaire !

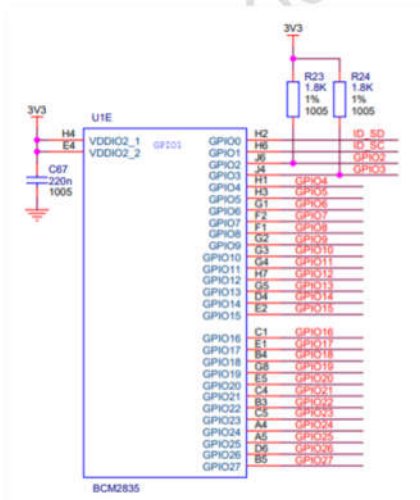


Figure 3.4 Connexions du SOC BCM2835 (utilisé sur la Rspi1) avec le connecteur J8

2. Fonction des GPIOs:

N'importe laquelle des broches GPIO peut **être désignée (par logiciel à travers des registres de configuration)** comme une broche **d'entrée ou de sortie**.

En plus du simple périphérique d'entrée et de sortie, les broches GPIO peuvent être **utilisées avec une variété de fonctions alternatives** (jusqu'à six fonctions alternatives pour certaines GPIO) comme :

- **PWM (modulation de largeur d'impulsion)**
 - **Hardware** PWM matériel disponible sur GPIO12, GPIO13, GPIO18, GPIO19, à ne pas confondre avec **Software** PWM disponible sur toutes les broches configurées en sortie, par programmation
- **SPI (Serial peripheral Interface)** (Fig.3.3)
 - SPI0: MOSI (GPIO10); MISO (GPIO9); SCLK (GPIO11); CE0 (GPIO8), CE1 (GPIO7)
 - SPI1: MOSI (GPIO20); MISO (GPIO19); SCLK (GPIO21); CE0 (GPIO18); CE1 (GPIO17); CE2 (GPIO16)
https://fr.wikipedia.org/wiki/Serial_Peripheral_Interface
- **I2C (Inter-Integrated Circuit)** (Fig.3.3)
 - Données: (GPIO2); Horloge (GPIO3)
 - Données EEPROM: (GPIO0); Horloge EEPROM (GPIO1)
<https://i2c.info/>
- **série (UART)**
 - TX (GPIO14); RX (GPIO15)

La liste complète des fonctions alternatives est donnée dans le document constructeur suivant, et un tableau récapitulatif est fourni en annexe 1 du chapitre :

<https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2835/BCM2835-ARM-Peripherals.pdf>

Attention ce document contient pas mal d'erreur, consultez la liste des erratas sur la page:

https://elinux.org/BCM2835_datasheet_errata

Un guide de brochage interactif et de fonctionnalité des GPIOs se trouve sur cette page :

<https://fr.pinout.xyz/>

La **figure 3.5** (extrait du datasheet constructeur), montre un exemple de **registre de sélection de fonction, GPFSEL0**, qui permet la configuration de certaines GPIO comme entrée ou sortie ou prenant une fonction alternative (alt0, à alt5) parmi celles possibles et décrites dans le tableau de l'annexe 1. Après un **Reset du circuit SOC** toutes les GPIO **sont configurées en entrées avec une résistance de mise à 0**, après le software **peut les reconfigurer** pour prendre d'autres fonctions.

Bit(s)	Field Name	Description	Type	Reset
31-30	---	Reserved	R	0
29-27	FSEL9	<u>FSEL9 - Function Select 9</u> 000 = GPIO Pin 9 is an input 001 = GPIO Pin 9 is an output 100 = GPIO Pin 9 takes alternate function 0 101 = GPIO Pin 9 takes alternate function 1 110 = GPIO Pin 9 takes alternate function 2 111 = GPIO Pin 9 takes alternate function 3 011 = GPIO Pin 9 takes alternate function 4 010 = GPIO Pin 9 takes alternate function 5	R/W	0
26-24	FSEL8	FSEL8 - Function Select 8	R/W	0
23-21	FSEL7	FSEL7 - Function Select 7	R/W	0
20-18	FSEL6	FSEL6 - Function Select 6	R/W	0
17-15	FSEL5	FSEL5 - Function Select 5	R/W	0
14-12	FSEL4	FSEL4 - Function Select 4	R/W	0
11-9	FSEL3	FSEL3 - Function Select 3	R/W	0
8-6	FSEL2	FSEL2 - Function Select 2	R/W	0
5-3	FSEL1	FSEL1 - Function Select 1	R/W	0
2-0	FSEL0	FSEL0 - Function Select 0	R/W	0

Figure 3.5 GPIO Function Select Register (GPFSEL0)

Pas **toutes les GPIO** du SOC sont attachées au connecteur **J8**, car certaines sont utilisées comme liaisons avec d'autres circuits de la carte Raspi; En somme, on peut avoir sur ce connecteur jusqu'à:

24x GPIO pins au max
2x SPI bus

2x Serial UART
1x I2C bus

2x 3.3V power pins
8x Ground pins
2x 5V power pins

En plus chaque broche peut être configurée pour être reliée par une résistance à +3.3V (Pull-Up) ou par une résistance à Gnd (Pull-Down), comme le montre la figure 3.6, **exception faite pour GPIO 2 et 3 (BCM)** car à ces deux pins sur la Raspi sont reliés deux résistances externes de PULL_UP de 1.8K, comme le montre la figure 3.3, qui empêchent la résistance de 50K interne de les tirer vers le potentiel 0.

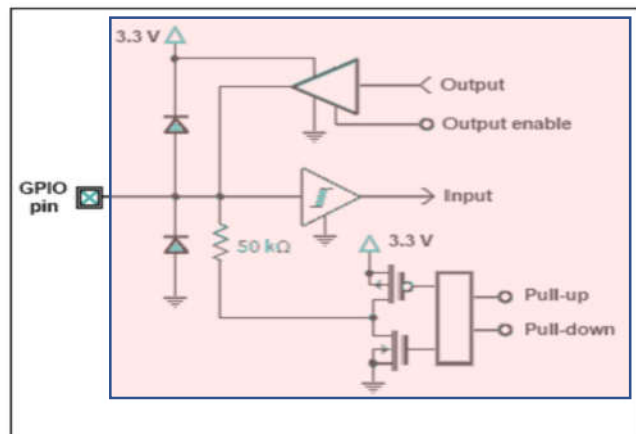


Figure 3.6 Diagramme fonctionnel interne simplifié d'une GPIO

Le tableau 3.1 suivant indique la configuration par défaut des broches après un démarrage de la Raspi, on peut donc voir que certaines GPIO sont configurées avec certaines fonctions alternatives, et aussi l'état des résistances Pull UP/Down :

Tableau 3.1 : Configuration des résistances PULL_UP/DOWN des GPIO après un démarrage de la Raspi

GPIO BCM	Power-on Pull	Alternate Functions	Header Pin
2	PullUp	I2C1 SDA	3
3	PullUp	I2C1 SCL	5
4	PullUp		7
5	PullUp		29
6	PullUp		31
7	PullUp	SPI0 CS1	26
8	PullUp	SPI0 CS0	24
9	PullDown	SPI0 MISO	21
10	PullDown	SPI0 MOSI	19
11	PullDown	SPI0 SCLK	23
12	PullDown		32
13	PullDown		33
16	PullDown	SPI1 CS0	36
17	PullDown		11
18	PullDown		12
19	PullDown	SPI1 MISO	35
20	PullDown	SPI1 MOSI	38
21	PullDown	SPI1 SCLK	40
22	PullDown		15
23	PullDown		16
24	PullDown		18
25	PullDown		22
26	PullDown		37
27	PullDown		13
35*	PullUp		Red Power LED
47*	PullUp		Green Activity LED

* = Raspi 2 seulement GPIO 35 & 47.

plus d'info sur les GPIO : <https://fr.pinout.xyz/>

3. Utilisation des GPIO :

L'utilisation des GPIO d'une Rspi peut être fait :

- En mode commandes et scripts shell, directement à partir de **l'espace utilisateur** :
 - Via **l'interface système /sys/class/gpio** : (système de fichiers virtuel sysfs → méthode ancienne, et en train d'être déconseillée, valable pour les **versions du noyau Linux < 4.8**)
 - ou via une **nouvelle interface système /dev/gpiochip***, en utilisant la librairie **libgpiod** (système de fichier **devtmpfs** monté sur le répertoire /dev → nouvelle méthode, disponible à partir de la version **4.8 du noyau Linux**).
- En mode commandes en utilisant des **utilitaires binaires** sans passer par l'interface système, comme **gpio** basé la librairie **wiringpi** ou l'utilitaire **pigs** installé avec la librairie **pigpio**, ou les plus récents outils associés à la librairie pigpio.
- Via des programmes écrits en différents langages (C, Python, ...) basés sur des librairies, comme :
 - **wiringpi** (voir <http://wiringpi.com/>) : librairie écrite en langage C, open source, installée par défaut sur raspbian. (son développeur a arrêté son maintien).
 - **pigpio** (voir <http://abyz.me.uk/rpi/pigpio/>): librairie écrite en langage C, open source, offre aussi un programme démon (Daemon), qui tourne sur une Raspi, et qui permet de commander les **GPIO à distance via un programme python** qui tourne localement sur la Raspi ou sur un PC par exemple, installée par défaut sur raspbian.
 - **Rpi.GPIO** (voir <https://pypi.org/project/RPi.GPIO/>): librairie en python (compatible python 2.7 et 3), installée par défaut dans Raspbian.

L'utilisation des GPIO sans passer par l'interface système implique des configurations bas niveau du Soc BCM283X ; elle nécessite les droits du super-utilisateur (root), et ne respecte pas les principes d'abstraction et d'arbitrage assurés par le noyau.

3.1. Mode commande : Utilitaire binaire gpio (accompagnant wiringPi) sous Raspbian

La distribution **Raspbian** inclut par défaut une installation de la librairie **wiringpi** et par conséquent elle inclut aussi un utilitaire appelé **gpio** qui peut être utilisé pour configurer et programmer les GPIO de la Rspi en **mode commandes**.

gpio est un **outil en ligne de commande** pour permettre à l'utilisateur de configurer/contrôler facilement les broches **GPIO** sur la **Raspi** et les **convertisseurs SPI A/D et D/A** sur les **cartes d'extension Gertboard** (qui étend les fonctionnalités de la Raspi).

Il est conçu à des fins de test, de diagnostic et de commandes simples, mais peut être utilisé dans des **scripts shell** pour un **contrôle général relativement lent** des broches GPIO, et à distance via **ssh** ou **vnc** ou interface web (par exemple en **domotique**).

Dans ce paragraphe on donne les principales caractéristiques, de cette utilitaire, avec quelques exemples. Pour toutes les options il faut consulter le manuel en ligne de commande (commandes : **man gpio** ou **gpio -h**), ou la documentation en ligne sur la page web de la librairie **wiringPi** (<http://wiringpi.com/>).

- Afficher la version de l'utilitaire, ainsi que des **caractéristiques de la carte Raspi** :

Syntaxe de la commande : gpio -v

- Lire et afficher l'état de toutes les GPIO (configuration et valeurs, voir fig.3.7) :

Syntaxe de la commande : gpio readall

NB: il y a trois numérotations des broches :

- la **numérotation physique** : position de la broche dans le connecteur de 1 à 40
- la **numérotation constructeur Broadcom BCM** : numéro sur le SOC
- la **numérotation établie par l'éditeur de la bibliothèque wiringpi : wPi**.

Exemple : la broche ayant le numéro physique 29, possède le numéro constructeur BCM 5 et le numéro **wPi** 21 ou le nom GPIO. 21.

BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM
2	8	3.3v			1	2		5v		
3	9	SDA.1	IN	1	3	4		5v		
4	7	SCL.1	IN	1	5	6		0v		
		GPIO. 7	IN	1	7	8	0	IN	TxD	15
		0v			9	10	1	IN	RxD	16
17	0	GPIO. 0	IN	0	11	12	0	IN	GPIO. 1	1
27	2	GPIO. 2	IN	0	13	14		0v		
22	3	GPIO. 3	IN	0	15	16	0	IN	GPIO. 4	4
		3.3v			17	18	0	IN	GPIO. 5	5
		0v			19	20		0v		
10	12	MOSI	IN	0	21	22	0	IN	GPIO. 6	6
9	13	MISO	IN	0	23	24	1	IN	CE0	10
11	14	SCLK	IN	0	25	26	1	IN	CE1	11
		0v			27	28	1	IN	SCL.0	31
0	30	SDA.0	IN	1	29	30		0v		
5	21	GPIO. 21	IN	1	31	32	0	IN	GPIO. 26	26
6	22	GPIO. 22	IN	0	33	34		0v		
13	23	GPIO. 23	IN	0	35	36	0	IN	GPIO. 27	27
19	24	GPIO. 24	IN	0	37	38	0	IN	GPIO. 28	28
26	25	GPIO. 25	IN	0	39	40	0	IN	GPIO. 29	29
		0v								

Figure 3.7 Etats des GPIOs lus et affichés par la commande gpio

- Configurer une broche **<pin>** donnée en numérotation **wPi**, en entrée (**in**), ou sortie(**out**), entrée avec résistance de Pull_up (**up**), ou avec résistance de Pull_down(**down**) ou sans résistance (**tri**) :

Syntaxe de la commande : gpio mode <pin> in|out|up|down|tri

- Configurer une broche **<pin>** donnée en numérotation **BCM**, en entrée (**in**), ou sortie(**out**), entrée avec résistance de Pull_up (**up**), ou avec résistance de Pull_down(**down**) ou sans résistance (**tri**) :

Syntaxe de la commande : `gpio -g mode <pin> in|out|up|down|tri`

- Configurer une broche **<pin>** donnée en numérotation **Physique**, en entrée (**in**), ou sortie(**out**), entrée avec résistance de Pull_up (**up**), ou avec résistance de Pull_down(**down**) ou sans résistance (**tri**) :

Syntaxe de la commande : `gpio -l mode <pin> in|out|up|down|tri`

- Mettre la broche **<pin>** configurée en sortie, à la valeur 0 ou 1 :

Syntaxe de la commande : `gpio [-l|-g] write <pin> 0|1`

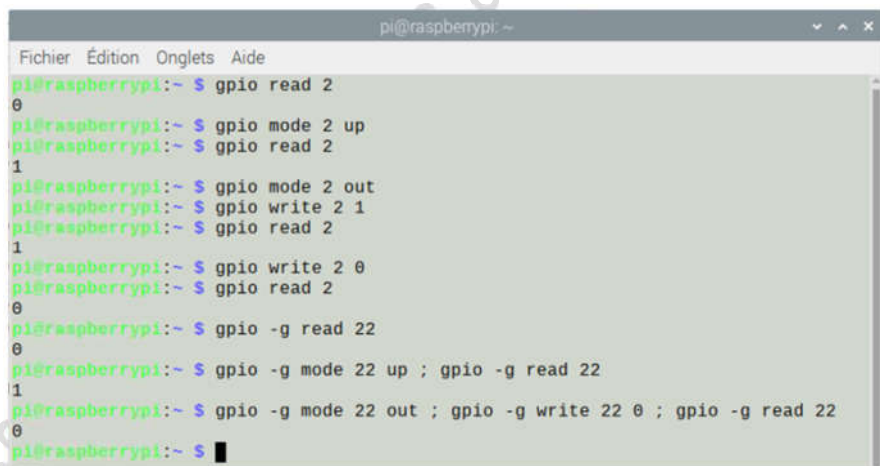
si aucune des options -l et -g n'est utilisée <pin> numéro wPi de la broche, si -l est utilisée <pin> numéro physique et si -g est utilisée <pin> numéro BCM de la broche.

- Lire et retourner l'état de la broche **<pin>** configurée en entrée :

Syntaxe de la commande : `gpio [-l|-g] read <pin>`

si aucune des options -l et -g n'est utilisée <pin> numéro wPi de la broche, si -l est utilisée <pin> numéro physique et si -g est utilisée <pin> numéro BCM de la broche.

Exemples d'utilisation :



```

pi@raspberrypi: ~
Fichier Édition Onglets Aide
pi@raspberrypi:~$ gpio read 2
0
pi@raspberrypi:~$ gpio mode 2 up
pi@raspberrypi:~$ gpio read 2
1
pi@raspberrypi:~$ gpio mode 2 out
pi@raspberrypi:~$ gpio write 2 1
pi@raspberrypi:~$ gpio read 2
1
pi@raspberrypi:~$ gpio write 2 0
pi@raspberrypi:~$ gpio read 2
0
pi@raspberrypi:~$ gpio -g read 22
0
pi@raspberrypi:~$ gpio -g mode 22 up ; gpio -g read 22
1
pi@raspberrypi:~$ gpio -g mode 22 out ; gpio -g write 22 0 ; gpio -g read 22
0
pi@raspberrypi:~$

```

- Changer l'état de la broche **<pin>** configurée en sortie :

Syntaxe de la commande : `gpio [-l|-g] toggle <pin>`

si aucune des options -l et -g n'est utilisée <pin> numéro wPi de la broche, si -l est utilisée <pin> numéro physique et si -g est utilisée <pin> numéro BCM de la broche.

- Faire **clignoter** l'état de la broche **<pin>** configurée en sortie, avec une **fréquence fixe de 1Hz**:

Syntaxe de la commande : `gpio [-l|-g] blink <pin>`

si aucune des options -1 et -g n'est utilisée <pin> numéro wPi de la broche, si -1 est utilisée <pin> numéro physique et si -g est utilisée <pin> numéro BCM de la broche.

- Configure la broche <pin> (wPi) en mode Modulation de largeur d'impulsion PWM (Pulse Width Modulation) :

Syntaxe de la commande : `gpio [-1|-g] mode <pin> pwm <dc>`

<dc> est la valeur du rapport cyclique (Duty cycle) : de 0 à Range (définie ci-dessous)

si aucune des options -1 et -g n'est utilisée <pin> numéro wPi de la broche, si -1 est utilisée

NB : Le Soc BCM2835 possède seulement deux blocs PWM : PWM0 et PWM1 ; pas toutes les GPIO sont utilisables en mode PWM, seules BCM GPIO12, GPIO13, GPIO18, GPIO19, GPIO40, GPIO41, GPIO45 (voir Annexe 1 liste des fonctions alternatives des GPIO extraite du BCM2835 datasheet (page 102 et 103)):

<https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2835/BCM2835-ARM-Peripherals.pdf>

ce qui correspond aux broches numéros (wPi) 23,26, 24, 1 les seules présentes sur le connecteur à 40 broches.

De plus le BCM2835 possède de mode de fonctionnement des modules PWM : **balanced Mode BM** et **Mark Space Mode MS**. Pour un rapport cyclique de 1/3:

- en **mode équilibré (BM)** on obtient au niveau de la sortie :
010010010010|010010010010|010010010010|010010010010|010010010010|...
<-----Range ---->|<-----Range ---->|
- en mode **marque-espace (BM)** on obtient au niveau de la sortie :
111100000000|111100000000|111100000000|111100000000|111100000000|...
<-----Range---->|<-----Range---->|
Avec Range est le nombre de bit par période, période $T_{pwm} = T_b * Range$, T_b période d'un bit, $F_b = 1/T_b = f_{base}/divisor$, $f_{base} = 19.2\text{Mhz}$, et divisor =1, 2, 4, 8, 16, ..., 4095, donc
 $F_{pwm} = 1/T_{pwm} = 1/(T_b * Range) = F_b / Range = f_{base} / (Range * divisor)$

mode, divisor, Range sont défini par software à travers des registres internes du BCM2835.

- Choisir le mode du PWM :

Syntaxe de la commande : `gpio pwm-bal/pwm-ms`

- Choisir Range pour le PWM :

Syntaxe de la commande : `gpio pwmr <range>`

Range de 0 à 65535.

- Choisir Divisor pour le PWM :

Syntaxe de la commande : `gpio pwmc <divider>`

Divisor de 0 à 4095

Exemple :

Pour contrôler un moteur avec un signal PWM en mode pwm-ms de fréquence d'environ 1KHz, on choisit : Divisor =16 et puisque $F_{pwm} = 1\text{khz} = f_{base}/(divisor * Range) = (19.2\text{Mhz}/16 * Range)$, ce qui donne Range=1200. Pour le rapport cyclique on choisit 50% : donc $dc = 50\% * Range = 600$

On vérifie le résultat à l'aide d'un oscilloscope software sur la figure 3.8a.

Si on prend maintenant un rapport cyclique de 25% soit $dc = 1200/4 = 300$, le résultat est sur la figure 3.8b. Et pour un rapport cyclique de 75%, le résultat est sur la figure 3.8c.


```

pi@raspberrypi:~ $ gpio -g mode 12 pwm
pi@raspberrypi:~ $ gpio pwm-ms ; gpio pwmr 1200 ; gpio pwmc 16;
pi@raspberrypi:~ $ gpio -g pwm 12 600 #rapp cyc 50%
pi@raspberrypi:~ $ gpio -g pwm 12 300 #rapp cyc 25%
pi@raspberrypi:~ $ gpio -g pwm 12 900 #rapp cyc 75%
pi@raspberrypi:~ $ █

```

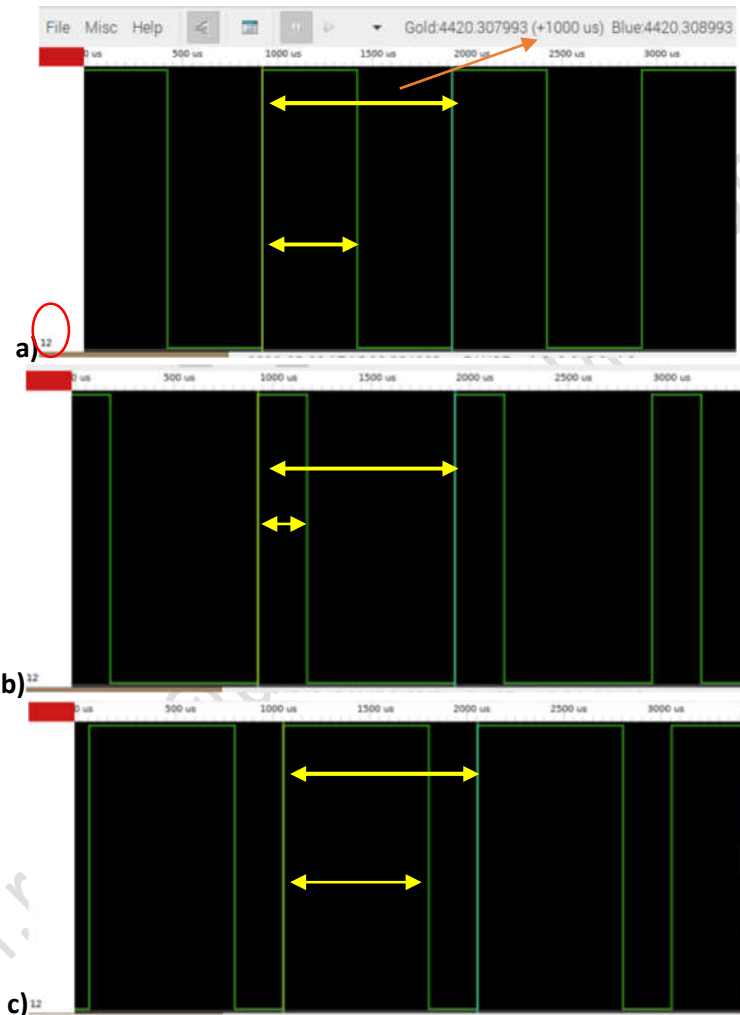


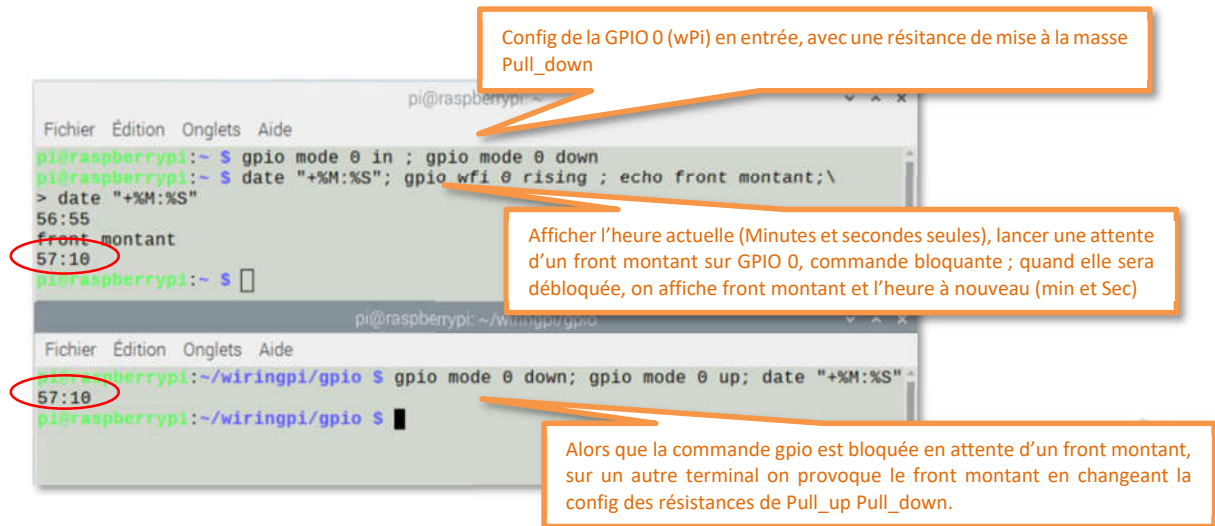
Figure 3.8 Capture par un Oscillo software le signal à la sortie de la broche BCM 12 pour un dc=50%, 25%, 75%

- Détecter une interruption par front montant(**rising**) ou descendant(**falling**) ou les deux(**both**) :

Syntaxe de la commande : `gpio wfi <pin> falling|rising|both`

wfi : wait for interruption : commande bloquante, ne retourne que si le front attendu sur la broche est apparu.

Exemple :



L'outil **gpio** est un programme qui peut s'exécuter par n'importe quel utilisateur avec les droits du super-utilisateur **root**, son bit setuid est positionné à 1 comme on peut le voir à l'aide de cette commande :

```
pi@raspberrypi ~$ ls -l /usr/bin
total 36
-rwsr-xr-x 1 root root 33336 mai  6 11:29 gpio
```

|__> rws (r lecture/w écriture/s exécution pour les autres utilisateurs avec les mêmes droits que root).

Ceci est obligatoire car ce programme ne s'appuie pas sur le noyau Linux pour accéder aux GPIO, et accède directement au matériel, les permissions root sont alors nécessaires.

3.2. Intégration de gpio dans des scripts shell:

Les commandes précédentes permettent de **configurer/contrôler/surveiller les GPIOs**, de manière manuelle. Si elles sont intégrées dans des scripts shell, on peut alors profiter de la puissance du langage de scripting shell, pour développer rapidement des **applications automatisant l'utilisation des GPIO**. Le langage de **scripting shell** GNU/Linux combine l'utilisation des commandes GNU/Linux et des instructions de contrôle (comme les tests, les boucles, ...). C'est **un langage interprété et non compilé**. Cela veut dire qu'un **script est un simple fichier texte** contenant une **série de commandes**, qui sera interprété, par l'interpréteur de commandes (Shell). Les détails des possibilités de ce langage peuvent être consultées :

dans le manuel de la commande bash

<http://manpagesfr.free.fr/man/man1/bash.1.html>

ou sur le site de documentation de Linux (plus avancé):

<https://tldp.org/LDP/abs/html/part2.html>

Exemple d'un script shell utilisant les GPIO:

Ce script configure une GPIO pour commander une LED et trois GPIO pour lire l'état de trois boutons poussoirs, puis après fait clignoter la led 5 fois, et rentre dans une boucle infinie qui surveille l'état des BP, chaque fois qu'un BP est appuyé, un message indique que la GPIO correspondante est passée à 0, ce message se répète tant que le BP reste appuyé.

3.3. Planification de contrôle des GPIO dans le temps : commande at et service cron

La **domotique** implique, des fois, de faire des tâches de **manière automatique et ponctuelle/répétitive**, dans **des heures et/ou jours**, bien précis. On peut alors, utiliser les commandes gpio combinées avec les commandes shell **at** ou l'outil de planification des tâches dans le temps **cron**.

3.3.1. Planification d'une tâche ponctuelle dans le temps par commande at :

La commande **at** permet d'exécuter des travaux (jobs) à une date/heure précise. Cette commande n'est pas installée par défaut, il faut commencer par l'installer :

```
pi@raspberrypi ~$ sudo apt-get install at
```

Exemples sur une Raspberry pi

```
pi@raspberrypi ~$ gpio mode 8 out      #config GPIO 8 en sortie
pi@raspberrypi:~$ date                 #vérifier date
jeudi 11 mai 2019, 09:26:12 (UTC+0100)
pi@raspberrypi:~$ at 9:30              #planifier jobs pour 9:30
warning: commands will be executed using /bin/sh
at> gpio write 8 1                      #Mettre à 1 GPIO 8
at> sleep 2m                           #attendre 2 min
at> gpio write 8 0                      #Remettre à 0 GPIO
at> <EOT>                              #Tapez Ctrl+D pour terminer la saisie
job 11 at Thu May 11 09:30:00 2019
```

une fois validée la saisie de la commande " **at 9:30** ", vous êtes invités à saisir les commandes à exécuter à la date/heure précisée dans la commande ; cette saisie se termine quand vous tapez Ctrl+D. Le job reçoit alors un numéro (dans l'exemple précédent il a le numéro 11).

Les commandes peuvent être réunies dans un fichier ; Editez un fichier texte avec **nano**

```
pi@raspberrypi:~$ nano planif.txt
```

et y saisissez les commandes suivantes



Exécutez les commandes suivantes qui planifient 4 jobs :

```
pi@raspberrypi:~$ at -f planif.txt 6:30 5/18/17
pi@raspberrypi:~$ at -f planif.txt 18:30 5/18/17
pi@raspberrypi:~$ at -f planif.txt 6:30 6/18/17
pi@raspberrypi:~$ at -f planif.txt 18:30 6/18/17
```

Le 18 Mai et 18 Juin à 6h30 et 18h30 la sortie GPIO 8 passe à l'état 1 et le reste pendant 1h seulement.

NB : la date est format MM/JJ/AA **le mois avant le jour !**

Pour voir le contenu de la file des jobs, utilisez la commande **atq**

```

pi@raspberrypi: ~
Fichier  Edition  Onglets  Aide
pi@raspberrypi:~$ atq
15      Sun Jun 18 18:30:00 2017 a pi
13      Thu May 18 18:30:00 2017 a pi
14      Sun Jun 18 06:30:00 2017 a pi
12      Thu May 18 06:30:00 2017 a pi
pi@raspberrypi:~$ █

```

On peut à tout moment annuler la planification d'un job, par exemple, pour supprimer le job 12 il faut taper la commande (`atrm rm` pour remove):

```

pi@raspberrypi:~$ atrm 12      #supprime le job 12
pi@raspberrypi:~$ atq         #afficher les jobs planifiés pour vérif

```

```

pi@raspberrypi: ~
Fichier  Edition  Onglets  Aide
pi@raspberrypi:~$ atq
15      Sun Jun 18 18:30:00 2017 a pi
13      Thu May 18 18:30:00 2017 a pi
14      Sun Jun 18 06:30:00 2017 a pi
12      Thu May 18 06:30:00 2017 a pi
pi@raspberrypi:~$ atrm 12
pi@raspberrypi:~$ atq
15      Sun Jun 18 18:30:00 2017 a pi
13      Thu May 18 18:30:00 2017 a pi
14      Sun Jun 18 06:30:00 2017 a pi
pi@raspberrypi:~$ █

```

On peut utiliser une planification relativement à l'instant présent (**now**) en utilisant la forme suivante :

```
pi@raspberrypi:~$ at -f planif.txt now +Numb <unit>
```

où Numb est un entier et <unit> peut être soit minutes ; hours ; days ; weeks ; months ; years

Exemples :

```

pi@raspberrypi:~$ at -f planif.txt now +5
#cmdes ds planif.txt seront exécutées après 5 min

pi@raspberrypi:~$ at now +2 d
warning: commands will be executed using /bin/sh
at> gpio write 8 1      #Mettre à 1 GPIO 8
at> <EOT>               #Tapez Ctrl+D pour terminer la saisie
#la mise à 1 de GPIO 8 sera faites d'ici 2 jours

```

La commande `sleep`, possède la syntaxe suivante : **sleep NUMBER[SUFFIX]** ; où [SUFFIX] peut être s pour secondes (par défaut), m pour minutes, h pour heures, d pour jours.

Plus de détails, consultez le manuel : `man at` et `man sleep`

3.3.2. Planification d'une tâche répétitive dans le temps par le service cron :

La commande `at` n'est pas très adaptée à la planification des tâches répétitives dans le temps ; on utilise pour ce type de tâche le service **cron**.

La planification des tâches répétitives se fait en **modifiant un fichier crontab**, via la commande `crontab -e` (edit), il faut choisir par la suite l'éditeur de texte à utiliser pour cette modification (on choisit nano comme éditeur de texte en validant par Enter).


```

pi@raspberrypi:~$ crontab -e
no crontab for pi - using an empty one

Select an editor. To change later, run 'select-editor'.
 1. /bin/ed
 2. /bin/nano <---- easiest
 3. /usr/bin/vim.tiny
Choose 1-3 [2]: █

```

Les tâches à planifiées sont à ajouter dans ce fichier après la ligne commentée (#) :

m h dom mon dow command

Cette ligne vous rappelle les champs à utiliser pour les lignes à ajouter dans ce fichier :

m : minutes (0 - 59) ; h : heures (0 - 23) ; dom (day of month) : jour du mois (1 - 31) ; mon (month) : mois (1 - 12) ; dow (day of week) : jour de la semaine (0-6) 0 pour dimanche ; command : c'est la commande à exécuter.

Chacun des 5 premiers champs peut contenir :

- un nombre valide par exemple 6 : la commande est exécutée quand ce champ prend la valeur 6 ;
- * : la commande est exécutée quelle que soit la valeur du champ
- suite de valeurs séparées par des virgules, exemple 3,5,10 : la commande est exécutée quand le champ prend les valeurs 3, 5 ou 10. **Ne pas mettre d'espace après la virgule ;**
- une plage de valeurs comme 3-7 : commande exécutée pour les valeurs 3 à 7 incluses ;
- */3 : commande exécutée pour toutes les valeurs multiples de 3 (par exemple à 0 h, 3 h, 6 h, 9 h...)

Quelques exemples à analyser :

m h dom mon dow command

	Commande sera exécutée
45 * * * * commande	Toutes les heures à 45 minutes exactement. Donc à 00 h 45, 01 h 45, 02 h 45, etc.
00 * * 0 commande	Tous les dimanches à minuit (dans la nuit de samedi à dimanche).
30 4 15 * * commande	Tous les 15 ^{èmes} jours du mois à 4 h30 du matin.
0 7 * 8 * commande	Tous les jours du mois d'Aout à 7h du matin.
0 * 4 12 * commande	Toutes les heures du 4 décembre.
* * * * * commande	Toutes les minutes
0 7 * * 1-5 commande	Tous les jours du Lun au vend à 7h (Un réveil par exple)

```

pi@raspberrypi: ~
Fichier  Édition  Onglets  Aide
GNU nano 2.2.6  Fichier : /tmp/crontab.gCdH2B/crontab
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow  command
* * * * * sudo gpio toggle 8

```

Cette commande va faire basculer GPIO 8 chaque minute et tous les jours !

Après sauvegarde du fichier (Ctrl+X, O, Enter), la commande **crontab** confirme l'installation de la table **crontab** et en informe le service **cron** des changements.

```
Select an editor. To change later, run 'select-editor'.
 1. /bin/nano      <---- easiest
 2. /usr/bin/vim.tiny
 3. /bin/ed

Choose 1-3 [1]: 1
crontab: installing new crontab
pi@raspberrypi:~$
```

Après avoir configuré GPIO 8 comme sortie, on vérifie le résultat de cette tâche en utilisant la une boucle qui lit l'état de GPIO 8 toutes les 20 secondes

```
pi@raspberrypi:~$ gpio mode 8 out
pi@raspberrypi:~$ while true; do echo -n `gpio read 8` ; sleep 20; done
0011100011100011
```

Exemple de synthèse:

```
GNU nano 3.2 /tmp/crontab.5RTxks/crontab

# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
* * * * * sudo gpio toggle 8
10 15 * * * sudo /home/pi/ch3script1 1>/home/pi/journal.txt 2>/home/pi/erreurs.txt
```

Exécution du script étudié précédemment avec redirection des sorties vers des fichiers, chaque jour à 15h50.
NB : il faut indiquer le chemin complet du script et aussi des fichiers

Résultats obtenus

```

pi@raspberrypi:~$ date "+%Hh:%Mm"
15h:50m
pi@raspberrypi:~$ V=24;gpio mode $V down ; sleep 0.3; gpio mode $V up
pi@raspberrypi:~$ V=21;gpio mode $V down ; sleep 0.3; gpio mode $V up
pi@raspberrypi:~$ V=25;gpio mode $V down ; sleep 0.3; gpio mode $V up
pi@raspberrypi:~$ V=21;gpio mode $V down ; sleep 0.3; gpio mode $V up
pi@raspberrypi:~$ cat journal.txt
fin de config
led clignotera 5 fois
LED reste allumée
GPIO24 à 0
GPIO24 à 0
GPIO24 à 0
GPIO21 à 0
GPIO21 à 0
GPIO21 à 0
GPIO25 à 0
GPIO25 à 0
GPIO25 à 0
GPIO21 à 0
GPIO21 à 0
GPIO21 à 0
pi@raspberrypi:~$ cat erreurs.txt
pi@raspberrypi:~$

```

A partir de 15h50

On simule des actions sur les BP

On analyse le contenu de journal.txt, et on retrouve la trace des messages indiquant le passage à zéro des GPIO

On analyse le contenu erreurs.txt

3.4. Contrôle des GPIO en Python:

Python est un langage qui a vite fait sa place parmi les langages les plus utilisés. Il a l'avantage d'être simple à utiliser, et dispose d'une riche librairie sous forme de modules qui le rendent un peu passe partout ; **et surtout il a été adopté par les développeurs pour raspberryPi.**

Python est un langage interprété et non pas compilé. Actuellement il y a deux versions majeures qui ne sont pas toujours compatibles **Python 3** et **python 2**. Les deux versions de python sont installées par défaut sur Raspbian.

```

pi@raspberrypi:~$ python
Python 2.7.16 (default, Oct 10 2019, 22:02:15)
[GCC 8.3.0] on linux2
Type "help", "copyright", "credits" or "license()"
>>> exit()
pi@raspberrypi:~$ python3
Python 3.7.3 (default, Dec 20 2019, 18:57:59)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
pi@raspberrypi:~$

```

On ouvre l'interpréteur python 2 en mode interactive

On quitte cette session interactive par exit()

NB : python pour python version 2 et python3 pour la version 3. Les deux interpréteurs utilisent l'outil GNU GCC (GNU collection compilers).

3.4.1. Utilisation du module RPi.GPIO:

Le module **RPi.GPIO** est l'une des premières librairies utilisées pour contrôler les GPIO des raspberry pi sous python. Ce module est installé par défaut sur Raspbian Buster (version 2020). On peut le vérifier dans une session python interactive : si l'importation du module ne signale aucune erreur, alors il est bien sinon il faut le réinstaller en suivant la procédure indiquée sur la page :

<https://sourceforge.net/p/raspberry-gpio-python/wiki/install/>

```

pi@raspberrypi: ~
Fichier  Édition  Onglets  Aide
pi@raspberrypi:~ $ sudo python
Python 2.7.9 (default, Sep 17 2016, 20:26:04)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import RPi.GPIO
>>> exit()
pi@raspberrypi:~ $

```

Vérification de la présence du module Rpi.GPIO

Le mode interactif de python suppose que l'on doit taper commande (instruction) après commande (instruction). Pour éditer des scripts python et les tester on doit disposer d'un (Environnement de Développement Intégré IDE (en anglais). Celui qui est choisi pour la suite du cours est **IDLE**. Sur la Raspbian Buster il n'est pas installé par défaut il faut l'installer pour les deux versions de python.

```

pi@raspberrypi:~ $ sudo apt search idle-python
En train de trier... Fait
Recherche en texte intégral... Fait
idle-python2.7/stable 2.7.16-2+deb10u1 all
  IDE for Python (v2.7) using Tkinter

idle-python3.5/stable 3.5.4-4 all
  IDE for Python (v3.5) using Tkinter

idle-python3.6/stable 3.6.8-1 all
  IDE for Python (v3.6) using Tkinter

idle-python3.7/stable 3.7.3-2+deb10u1 all
  IDE for Python (v3.7) using Tkinter

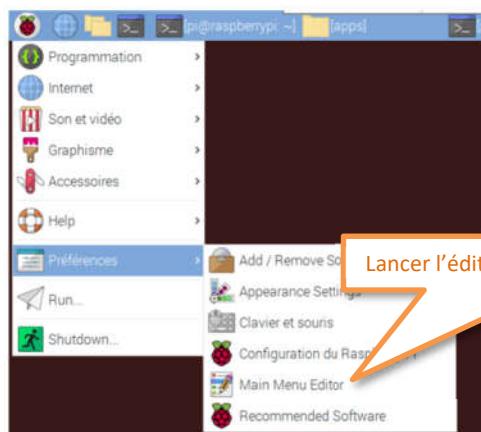
pi@raspberrypi:~ $ sudo apt-get install idle-python2.7 idle-python3.7
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Le paquet suivant a été installé automatiquement et n'est plus nécessaire :
  libmicrodns0
Veuillez utiliser « sudo apt autoremove » pour le supprimer.
Les NOUVEAUX paquets suivants seront installés :
  idle-python2.7 idle-python3.7

```

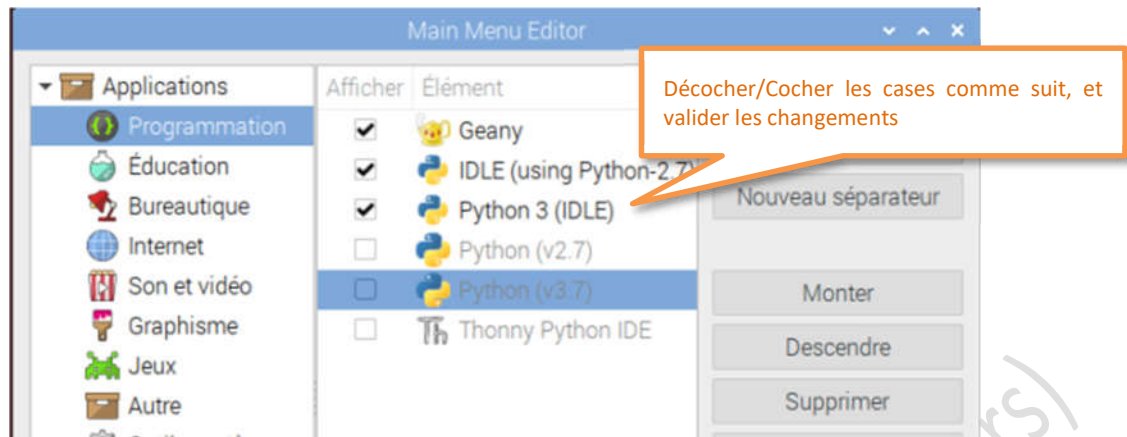
Recherche des packages pour idle python

installation des dernières versions

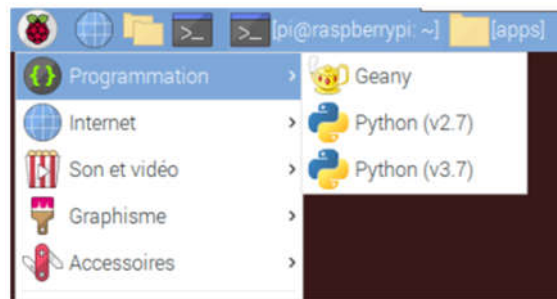
Une fois installés on ajoute des raccourcis sur le menu d'applications :



Lancer l'éditeur du menu principal



Dans le menu des applications de la Rspi on retrouve alors les deux lanceurs des interpréteurs python version 2 et 3.



Les fonctions de cette librairie sont décrites dans la page officielle suivante :

<https://sourceforge.net/p/raspberry-gpio-python/wiki/Examples/>

Dans la suite on donne quelques exemples d'utilisation de ces fonctions.

Script Python 1 utilisant les GPIO :

On lance l'IDE idle-python2, on crée un nouveau fichier un nouveau fichier et on y saisit le code suivant (les lignes commençant par # sont des commentaires peuvent ne pas être saisies) :

```

*Untitled*
File Edit Format Run Options Windows Help

import RPi.GPIO as GPIO
#importer le module RPi.GPIO sous le nom GPIO; ceci permet de précéder les objet
#du module Roi.GPIO par GPIO au lieu de RPi.GPIO
import time #pour pouvoir utiliser sleep(x) = blocage pendant x secondes
GPIO.setmode(GPIO.BCM)
#la numérotation utilisée sera celle de BCM; autre possibilité GPIO.BOARD
GPIO.setup(26, GPIO.OUT)
#configurer GPIO 26 (BCM) soit 37 physique, en sortie
while True:
    # boucle infinie
    #indentation pour marquer le bloc
    GPIO.output(26, GPIO.HIGH)
    #Mettre à 1 GPIO 26
    time.sleep(2)
    #blocage pendant 2 s
    GPIO.output(26, GPIO.LOW)
    #Mettre GPIO 26 à 0
    time.sleep(2)
    #blocage pendant 2 s

```


On sauvegarde le script(**File/Save as...**)sous le nom **clignoter26** (pas la peine d'ajouter l'extension py, elle sera automatiquement ajoutée).

On lance l'interprétation du script en cours (**Run/Run Module**)ou en utilisant le raccourcis clavier **F5**). Une nouvelle fenêtre **Python Shell** s'ouvre ; si elle n'affiche aucun message d'erreur c'est que le programme est en exécution. Pour le vérifier, ouvrez un terminal et tapez-y la suite de commandes :

```

pi@raspberrypi:~ $ while true; do gpio -g read 26; sleep 1; done
0
1
1
0
0
1
1
0
0
1
1
^C
pi@raspberrypi:~ $ while true; do gpio read 26; sleep 1; done
0
0
0
0
0
^C
pi@raspberrypi:~ $

```

NB l'utilisation de l'option **-g** car GPIO utilise la numérotation wPi si on ne spécifie pas **-g**
On voit bien que la sortie 26 (BCM) , soit 25 (wPi) clignote avec une période de 4 sec (2s On, 2s Off)

Erreur à ne pas commettre : **gpio read 26** donne l'état de 26 (wPi) soit 1 (BCM) qui est toujours à 0

On arrête script en fermant la fenêtre de Python shell qui a été ouverte (confirmez le kill du programme en exécution).

Script Python 2 utilisant les GPIO :

On Modifie le script précédent pour que GPIO 26 clignote tant que GPIO 0 est à 1 et le programme se termine si GPIO 0 passe à 0, après avoir remis GPIO 26 et GPIO 0 à leur configuration par défaut (IN et valeur=0).

```

*clignoter26_2.py - /home/pi/clignoter26_2.py (2.7.9)*
File Edit Format Run Options Windows Help

import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(26, GPIO.OUT)
GPIO.setup(0, GPIO.IN)
#config GPIO 0 comme entrée
GPIO.setup(0, GPIO.IN, pull_up_down=GPIO.PUD_UP)
#pull_up_down= GPIO.PUD_UP ou
#config GPIO 0 avec une résistance relié à 3.3V --> valeur de GPIO 0 =1
print('debut du programme')
while GPIO.input(0): #tant que GPIO 0 est à 1 faire clignoter GPIO 26
    GPIO.output(26, GPIO.HIGH)
    time.sleep(2)
    GPIO.output(26, GPIO.LOW)
    time.sleep(2)
GPIO.cleanup()
#réinitialiser la config de toutes les GPIO à l'état de démarrage
#possibilité de réinitialiser une seule GPIO cleanup(Num)
print('fin du programme')

```

On lance l'interprétation du script ; pour vérifier le bon fonctionnement on ouvre deux terminaux : dans le premier on prépare la série de commande qui permet de lire et visualiser l'état de GPIO 26 et

Pour résoudre ce problème (en partie) on modifie le script précédent pour qu'il scrute (polling) l'état de GPIO 0 plus fréquemment à l'intérieur de la boucle et quitte cette boucle quand il détecte un état bas. Le nouveau script est donné dans la figure suivante (**clignoter26_3.py**).

```

*clignoter26_3.py - /home/pi/clignoter26_3.py (2.7.9)*
File Edit Format Run Options Windows Help

import RPi.GPIO as GPIO
import time
import sys #pour sys.exit()
GPIO.setmode(GPIO.BCM)
GPIO.setup(26, GPIO.OUT)
GPIO.setup(0, GPIO.IN)
#config GPIO 0 comme entrée
GPIO.setup(0, GPIO.IN, pull_up_down=GPIO.PUD_UP)
#GPIO 0 à 1
print('debut du programme')

while True:
    compteur=0
    #basculer GPIO 26
    if (GPIO.input(26)==GPIO.LOW):
        GPIO.output(26, GPIO.HIGH)
    else:
        GPIO.output(26, GPIO.LOW)

    while GPIO.input(0)==GPIO.HIGH: #tant que GPIO est à 1
        time.sleep(0.1) #attendre 0.1s
        compteur+=1 #incrémenter compteur
        if (compteur==20): #2sec sont écoulées sans que GPIO 0 passe à 0
            break #quitter la boucle de scutation de GPIO 0
    if (compteur<20): #si GPIO 0 est passé à 1 avant les 2 sec
        print('fin du programme')
        GPIO.cleanup()
        sys.exit() #quitte le programme
    |

```

Un nouveau test du nouveau script dans les mêmes conditions que précédemment montre que le problème est résolu. A vrai dire le script sera aveugle pour toute impulsion de GPIO 0 de durée inférieure à 0.1s.

```

Python 2.7.9 Shell
File Edit Shell Debug Options Windows Help

Python 2.7.
[GCC 4.9.2]
Type "copyr
>>> =====
>>> debut du pr
>>> fin du prog
>>>

pi@raspberrypi:~ $ while true; do gpio -g read 26 ; sleep 1 ; done
^C
pi@raspberrypi:~ $ gpio -g mode 0 down ; sleep 0.5; gpio -g mode 0 up
pi@raspberrypi:~ $

```

1-Une seule tentative et le clignotement s'arrête et aussi le script

Autres fonctionnalités de la librairie RPi.GPIO :

<https://sourceforge.net/p/raspberry-gpio-python/wiki/Examples/>

Voir Utilisation à travers TD/TP.

3.4.2. Utilisation du module GPIOZERO:

Cette bibliothèque installée aussi par défaut dans la distribution Raspbian, est plus riche en fonctionnalités, bien documentée avec beaucoup d'exemples, et open source.

Son avantage est qu'elle est orientée Devices (périphériques) : Led, Sensor, Button, Buzzer, Motor,... et offre beaucoup de mécanismes pour implémenter une application réelle avec le moindre effort de développement. Mais elle nécessite un grand effort de documentation et un bon niveau en python, pour appréhender toutes ses fonctionnalités.

<https://gpiozero.readthedocs.io/en/stable>

3.5. Contrôle des GPIO en C:

Dans cette partie on utilisera la librairie wiringpi développée en C :

(voir <http://wiringpi.com/reference/>).

Tout programme C devant utiliser cette librairie doit contenir : **#include <wiringPi.h>**

Initialisation de la librairie :

Il existe 4 méthodes d'initialisation de la librairie wiringPi :

- **int wiringPiSetup (void)** : pour utiliser la numérotation wPi
- **int wiringPiSetupGpio (void)** : pour utiliser la numérotation Broadcom
- **int wiringPiSetupPhys (void)** : pour utiliser la numérotation Physique

Ces trois fonctions nécessitent le privilège root (mode kernel), car elles accèdent directement au matériel.

- **int wiringPiSetupSys (void)** : Initialise la librairie wiringPi mais utilise l'interface **/sys/class/gpio** plutôt que d'accéder directement au matériel. Elle peut être appelée par un utilisateur non root à condition que les broches GPIO aient été exportées avant, manuellement à l'aide de la commande gpio. La numérotation des broches dans ce mode est les nombres natifs Broadcom GPIO - pareils à wiringPiSetupGpio() ci-dessus.

Une des fonctions d'initialisation doit être appelée au début de votre programme ou votre programme ne fonctionnera pas correctement.

Configuration du mode de la GPIO :

void pinMode (int pin, int mode) :

Définit le mode d'une broche sur **INPUT**, **OUTPUT**, **PWM_OUTPUT** ou **GPIO_CLOCK**.

Le mode **PWM_OUTPUT** est valable seulement pour les GPIO (numérotation BCM) :

GPIO.12, GPIO.13, GPIO.18, GPIO.19, GPIO.40, GPIO.41, GPIO.45

Le mode **GPIO_CLOCK** est valable seulement pour les GPIO (numérotation BCM) :

GPIO.4, GPIO.20, GPIO.32, GPIO.34 (Clock 0)

GPIO.5, GPIO.21, GPIO.42, GPIO.44 (Clock 1)

GPIO.6, GPIO.43 (Clock 2)

Cette fonction n'a aucun effet après une initialisation **SetupSys**. Si on doit changer le mode broche, on peut le faire avec le programme **gpio** dans un script avant de lancer notre programme.

Commande d'une sortie en On/Off

void digitalWrite (int pin, int value);

Écrit la valeur **HIGH** ou **LOW** (1 ou 0) à la broche donnée qui doit avoir été précédemment définie comme une sortie. WiringPi traite n'importe quel nombre non nul comme **HIGH**, mais 0 est la seule représentation de **LOW**.

Exemple1 : myblink.c

Dans un éditeur de texte comme **nano** ou **mousepad** (accessible par la commande `mousepad` ou à partir du menu des application Accessoires/Text editor) , saisissez le programme C suivant et le sauvegarder sous le nom **myblink.c**:

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

#define PIN 8
int main (void)
{
    wiringPiSetup () ;    //adoption de la numérotation wPi
    pinMode (PIN, OUTPUT) ;    //config de PIN en sortie
    while (1)
    {
        digitalWrite (PIN, HIGH) ;    //mettre à 1 PIN
        delay (500) ;    //attendre 500ms
        digitalWrite (PIN, LOW) ;    //Mettre à 0 PIN
        delay (500) ;    //attendre 500ms
    }
    return 0 ;
}
```

La fonction `delay()` est fourni avec la librairie `wiringPi`.

La compilation du programme C doit explicitement indiquer de lier la librairie `wiringPi`.

```
pi@raspberrypi:~$ gcc -o myblink8 myblink.c -lwiringPi
```

Cette commande demande à gcc de compiler **myblink.c** et de le lier à la librairie `wiringPi` (option **-l**) et de produire (**-o pour output**) l'exécutable **myblink8** ; qui une fois lancer va faire clignoter GPIO 8.

L'exécutable généré doit être lancé avec **sudo**. Pour l'arrêter le programme il faut taper **Ctrl+C**, car c'est une boucle infinie :

```
pi@raspberrypi:~$ sudo ./myblink8
```

Pour voir le résultat de cette exécution, sur un autre terminal on tape la liste de commandes suivantes pour observer l'état de GPIO 8

```
pi@raspberrypi:~$ while true; do echo -n $((gpio read 8)) ; done
```

cette commande lit et affiche en permanence l'état de GPIO 8 ; on devrait voir GPIO 8 qui devrait changer d'état périodiquement.

`echo -n $((gpio read 8))` afficher sans retour à la ligne le résultat de la commande `gpio read 8`

Commande d'une sortie en PWM (Hardware)

void pwmSetMode (int mode);

Le générateur PWM peut fonctionner dans 2 modes BALANCED - "équilibré" et MARK_SPACE "marque_espace". Le premier est le mode par défaut dans la raspberry Pi. Vous pouvez basculer les modes en fournissant le paramètre mode: **PWM_MODE_BAL** ou **PWM_MODE_MS**.

void pwmSetRange (unsigned int range);

Ceci définit le registre de portée (rapport cyclique) dans le générateur PWM. La valeur par défaut est 1024 ; Valeurs possibles de 0 à 65535 (valeur possible sur 32bits)

void pwmSetClock (int divisor);

Ceci définit le diviseur pour l'horloge PWM.
Valeurs possibles de 0 à 4095.

void pwmWrite (int pin, int value) ;

Définit le rapport cyclique pour la sortie PWM indiquée par pin ; valeur possible de 0 à Range (100%).

Remarque: Les fonctions de commande PWM ne peuvent pas être utilisées en mode Sys.

Exemple2 : Commande rapport cyclique variable et fréquence fixe pwm1.c

Fpwm=1/Tpwm=Fb/Range=fbase/(Range*divisor)

pour fb=19.2MHz, Range =4096 , divisor=32, Fpwm=146,5Hz ; Tpwm=6,8ms

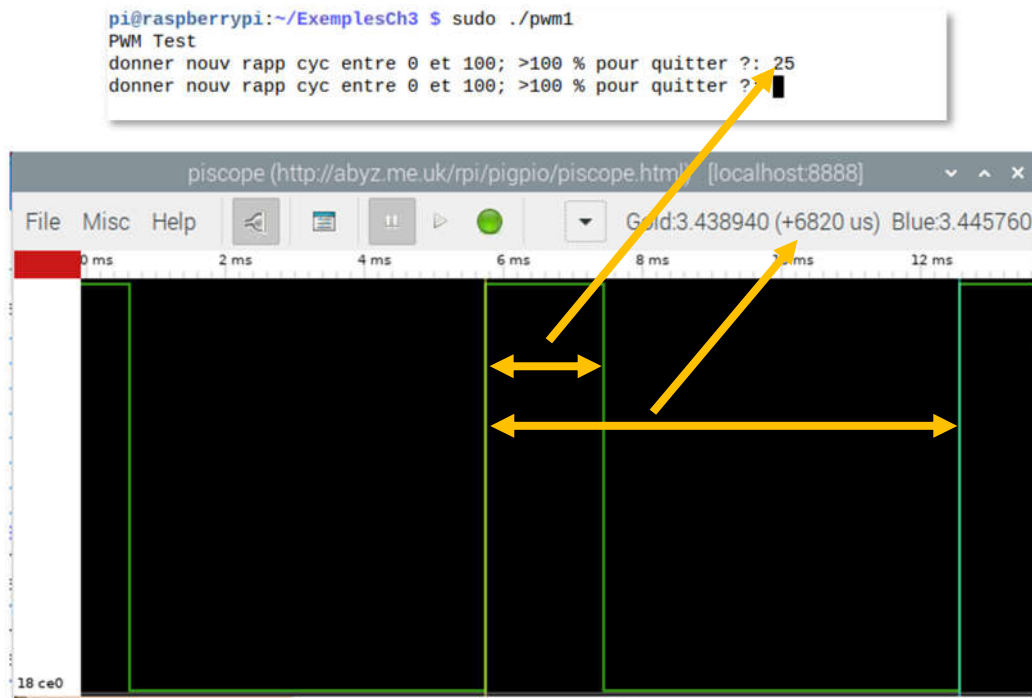
```
#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#define pin 12
#define Range 4096
#define Div 32
int main (void)
{
    int rapp_cyc=0 ;
    //config
    wiringPiSetupGpio(); //numérotation wPi
    pinMode (pin, PWM_OUTPUT); //GPIO 26 en sortie PWM
    printf("Rapport cyclique initial =100%% \n");
    pwmSetMode(PWM_MODE_MS); //Mode Mark space __|--|__|--|__|--|__
    pwmSetRange (Range) ;
    pwmSetClock (Div ) ;

    while(1){
        pwmWrite(pin, (long)rapp_cyc*Range/100) ;
        printf("donner nouv rapp cyc entre 0 et 100; >100 %% pour quitter ?: ");
        scanf("%d",&rapp_cyc);
        if ((rapp_cyc>100)|| (rapp_cyc<0)) break; //quit si valeur invalide
    }
    pwmWrite(pin, 0) ; //remet rapport cyc à 0%
    return 0 ;
}
```

Le programme doit être compilé en liant de manière statique la librairie wiringPi et exécuter avec sudo.

```
pi@raspberrypi:~$ gcc -o pwm1 pwm1.c -lwiringPi
```


A l'aide du l'oscilloscope logicielle piscope on peut vérifier le rapport cyclique est la fréquence :



Exemple3 : Commande progressive (en fad in fad) pwm2.c

```
#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#define pin 1 //wpi numero
#define Range 4096
#define Div 64 //→ fpwm=75Hz
int main (void)
{
    int rapp_cyc ; //rapport cyclique
    printf("PWM FadIn FadOut\n");
    wiringPiSetup(); //numérotation wPi
    pwmSetMode(PWM_MODE_MS); //Mode Mark space __|--|__|--|__|--|__
    pwmSetClock(Div) ;
    pwmSetRange(Range) ;
    pinMode (pin, PWM_OUTPUT); //GPIO 12 en sortie PWM
    for (;;) {
        //fad In
        for (rapp_cyc = 0 ; rapp_cyc < Range ; rapp_cyc++) {
            pwmWrite (pin, rapp_cyc) ; //config du rapport cyclique
            delay (1) ; //1ms -> duree fadin Range ms
        }
        //fad Out
        for (rapp_cyc= Range ; rapp_cyc >= 0 ; rapp_cyc--) {
            pwmWrite (pin, rapp_cyc) ;
            delay (2) ; //2ms -> duree fadout 2*Range ms
        }
    }
    return 0 ;
}
```

Le programme doit être compilé en liant de manière statique la librairie wiringPi et exécuter avec sudo.

```
pi@raspberrypi:~$ gcc -o pwm2 pwm2.c -lwiringPi
pi@raspberrypi:~$ sudo ./pwm2
```

Le programme précédent fait varier le rapport cyclique de 0 à Range puis de Range à 0, pour voir son effet ; dans un autre terminal on peut utiliser la boucle infinie suivante :

```
pi@raspberrypi:~$ while true; do echo -n $(gpio read 1) ; done
```

On observe alors une variation du nombre de 1 par rapport à celui de 0 (ce test n'est pas très précis mais ceci donne une idée sur ce qui se passe; un test avec un oscilloscope ou une led permet de voir la variation du rapport cyclique.

Lecture d'une entrée :

void pullUpDnControl (int pin, int pud);

Définit le mode de résistance de tirage (PULLUp/PULLDown) sur la broche donnée, qui doit être configurée comme entrée. Le paramètre pud devrait être : **PUD_OFF**, (sans tirage vers le haut / bas), **PUD_DOWN** (tirer vers la masse) ou **PUD_UP** (tirer vers 3.3v). Les résistances internes de tirage vers le haut / bas ont une valeur d'environ 50KΩ sur le Raspberry Pi.

Cette fonction n'a aucun effet sur les broches GPIO de Raspberry Pi en mode Sys. Si on doit activer un pull-up / pull-down, on peut le faire avec le programme gpio dans un script avant de lancer le programme.

int digitalRead (int pin);

Cette fonction renvoie la valeur lue à la broche donnée. Il sera **HIGH** ou **LOW** (1 ou 0) en fonction du niveau logique à la broche.

Détection de front sur une entrée :

int wiringPiISR (int pin, int edgeType, void (*function)(void)) ;

Cette fonction enregistre une fonction à appeler en cas d'interruption provoquée par un front sur la broche spécifiée. Le paramètre edgeType est **INT_EDGE_FALLING**, **INT_EDGE_RISING**, **INT_EDGE_BOTH** ou **INT_EDGE_SETUP**. Si c'est INT_EDGE_SETUP, aucune initialisation de la broche ne se produira ; il est supposé qu'on déjà configuré la broche ailleurs (p. Ex. Avec le programme gpio), mais si vous spécifiez l'un des autres types, la broche sera exportée et initialisée comme spécifiée. Cela se fait par un appel approprié au programme utilitaire gpio, il doit donc être disponible.

Le numéro de broche est fourni en mode actuel - modes de wiringPi natif, BCM_GPIO, physique ou Sys. Cette fonction fonctionnera dans n'importe quel mode et ne nécessite pas de privilèges root pour fonctionner.

La fonction fournie en paramètre, sera appelée chaque fois que l'interruption est déclenchée ; toutefois, si une interruption subséquente se déclenche avant de terminer l'exécution de cette fonction, elle est ignorée.

Une autre interruption déclenchée après avoir terminé l'exécution de cette fonction, exécute à nouveau cette fonction.

Cette fonction est exécutée à haute priorité (si le programme est exécuté en utilisant sudo ou en tant que root) et s'exécute simultanément avec le programme principal. Il a un accès complet à toutes les variables globales, les descripteurs de fichiers ouverts, etc.

Exemple3 : Détection de front sur GPIO0 int0.c

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

void affiche(){ //fonction appelée si Interruption
printf("une interruption !! \n");
if(digitalRead(2)==1) digitalWrite(2,0); else digitalWrite(2,1);
}

int main(){
wiringPiSetup(); //numérotation wPi
pinMode(0,INPUT); //GPIO 0 entrée
pullUpDnControl (0, PUD_UP); //résistance de PULLUP
pinMode(2,OUTPUT); GPIO 2 sortie pour indiquer interrup
wiringPiISR(0,INT_EDGE_FALLING,&affiche); //interrup sur front descendant
//Interrupt sur front descend sur GPIO 0
while(1); //pour maintenir le programme en vie
return 0;
}
```

L'exécution ne nécessite pas les privilèges root.

```
pi@raspberrypi:~$ gcc -o int0 int0.c -lwiringPi
pi@raspberrypi:~$ ./int0
```

Le programme attend un front descendant sur GPIO 0, et s'il se produit, il exécute la fonction affiche, cette dernière affiche un message et fait basculer GPIO2 (wPi) ; dans un autre terminal on exécute la série de commandes suivantes qui provoquent un front descendant, puis après une seconde lit la valeur de GPIO2 :

```
pi@raspberrypi:~$ gpio mode 0 up ; gpio mode 0 down ; sleep 1; gpio read 2
```

```
pi@raspberrypi: ~
Fichier Édition Onglets Aide
pi@raspberrypi:~ $ ./int0
une interruption !!
une interruption !!
une interruption !!
une interruption !!
□

pi@raspberrypi: ~
Fichier Édition Onglets Aide
pi@raspberrypi:~ $ gpio mode 0 up; gpio mode 0 down ; sleep 1; gpio read 2
1
pi@raspberrypi:~ $ gpio mode 0 up; gpio mode 0 down ; sleep 1; gpio read 2
0
pi@raspberrypi:~ $ gpio mode 0 up; gpio mode 0 down ; sleep 1; gpio read 2
1
pi@raspberrypi:~ $ gpio mode 0 up; gpio mode 0 down ; sleep 1; gpio read 2
0
pi@raspberrypi:~ $ █
```

Le multithreading:

int piThreadCreate (name) ; Cette fonction crée un thread (un processus léger : une tâche) qui exécute une autre fonction précédemment déclarée dans le programme principal, à l'aide de la déclaration **PI_THREAD**. Cette fonction est exécutée en même temps que le programme principal.

Un exemple peut être que cette fonction attend une interruption pendant que le programme continue d'effectuer d'autres tâches.

Le thread peut indiquer un événement ou une action en utilisant des variables globales pour communiquer avec le programme principal ou d'autres threads.

Les fonctions `_Thread` sont déclarées comme suit:

```
PI_THREAD (myThread)
{
    .. le code ici tourne en concurrence avec le programme principal,
       souvent une boucle infinie
}
```

un thread se basant sur cette fonction `_Thread` peut alors être démarré dans le programme principal comme suit :

```
x = pthreadCreate (myThread) ;
if (x != 0)
    printf ("le thread n'a pas démarré")
```

Exemple 4 : Clignotement de deux leds en concurrence thr1.c

```
#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>
#define LED1    26
#define LED2    2
int rep;
PI_THREAD (blinky26)
{
    fputs ("Thread blinky26 démarre", stderr);
    for (;;)
    { digitalWrite (LED1, HIGH) ; // On
      delay (2000) ;              // mS
      digitalWrite (LED1, LOW) ; // Off
      delay (2000) ;
    }
}
PI_THREAD (blinky2)
{
    fputs ("Thread blinky2 démarre", stderr);
    for (;;)
    { digitalWrite (LED2, HIGH) ; // On
      delay (1000) ;              // mS
      digitalWrite (LED2, LOW) ; // Off
      delay (1000) ;
    }
}
//*****
int main() {
    wiringPiSetup(); //numérotation wPi
    pinMode (LED1, OUTPUT) ;
    pinMode (LED2, OUTPUT) ;
    pinMode(0, INPUT); //GPIO 0 entrée BP
    pullUpDnControl (0, PUD_UP); //résistance de PULLUP

    if (pthreadCreate(blinky26) != 0)
        {printf ("impossible de démarrer blinky26");
         exit(1);}
    if (pthreadCreate(blinky2) != 0)
        {printf ("impossible de démarrer blinky26");
         exit(1);}
}
```

```
while (digitalRead(0)==1) {
    fputc('.',stderr); delay(1000);
}
return 0;
}
```

Ce programme créer deux threads, chacun fait clignoter une LED avec une fréquence différentes. Le programme principal (main) attend qu'on appuie sur un bouton poussoir pour s'arrêter. Ceci terminera aussi le clignotement des LEDs car les threads font partie du processus principal qui se termine.

Test du fonctionnement du programme :

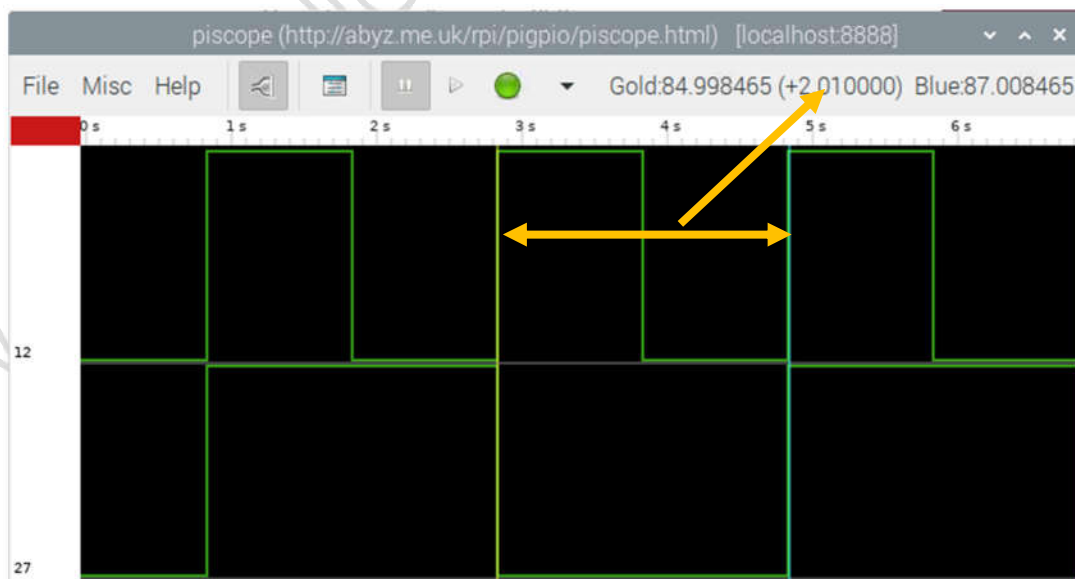
on efface thr1 s'il existe, on recompile thr1.c, et on le lance
Notez les messages des threads

affichage simultané des états des GPIO 2 et 26, avec une période de 500ms

On arrête le programme par une impulsion négative sur GPIO 0 réalisée par la modifications de la config des résistances de PULL UP/DOWN

Quand le programme s'arrête, les threads aussi s'arrêtent

La capture des signaux à l'aide de l'oscilloscope software piscope confirme le clignotement simultané des deux diodes avec les périodes de 4s et 2s



NB: la librairie wiringPI offre d'autres fonctions relatives aux cartes d'extensions qui s'adaptent à la raspberry pi ; à consulter sur le lien : wiringpi.com/Reference

Annexe 1

Fonctions alternatives des GPIO (extrait du datasheet BCM2835 ARM Peripherals)

Annexe 1 : Fonctions alternatives des GPIO, extrait du datasheet BCM2835 ARM Peripherals

	Pull	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
GPIO0	High	SDA0	SA5	<reserved>			
GPIO1	High	SCL0	SA4	<reserved>			
GPIO2	High	SDA1	SA3	<reserved>			
GPIO3	High	SCL1	SA2	<reserved>			
GPIO4	High	GPCLK0	SA1	<reserved>			ARM_TDI
GPIO5	High	GPCLK1	SA0	<reserved>			ARM_TDO
GPIO6	High	GPCLK2	SOE_N / SE	<reserved>			ARM_RTCK
GPIO7	High	SPI0_CE1_N	SWE_N / SDW_N	<reserved>			
GPIO8	High	SPI0_CE0_N	SD0	<reserved>			
GPIO9	Low	SPI0_MISO	SD1	<reserved>			
GPIO10	Low	SPI0_MOSI	SD2	<reserved>			
GPIO11	Low	SPI0_SCLK	SD3	<reserved>			
GPIO12	Low	PWM0	SD4	<reserved>			ARM_TMS
GPIO13	Low	PWM1	SD5	<reserved>			ARM_TCK
GPIO14	Low	TXD0	SD6	<reserved>			TXD1
GPIO15	Low	RXD0	SD7	<reserved>			RXD1
GPIO16	Low	<reserved>	SD8	<reserved>	CTS0	SPI1_CE2_N	CTS1
GPIO17	Low	<reserved>	SD9	<reserved>	RTS0	SPI1_CE1_N	RTS1
GPIO18	Low	PCM_CLK	SD10	<reserved>	BSCSL SDA / MOSI	SPI1_CE0_N	PWM0
GPIO19	Low	PCM_FS	SD11	<reserved>	BSCSL SCL / SCLK	SPI1_MISO	PWM1
GPIO20	Low	PCM_DIN	SD12	<reserved>	BSCSL / MISO	SPI1_MOSI	GPCLK0
GPIO21	Low	PCM_DOUT	SD13	<reserved>	BSCSL / CE_N	SPI1_SCLK	GPCLK1
GPIO22	Low	<reserved>	SD14	<reserved>	SD1_CLK	ARM_TRST	
GPIO23	Low	<reserved>	SD15	<reserved>	SD1_CMD	ARM_RTCK	
GPIO24	Low	<reserved>	SD16	<reserved>	SD1_DAT0	ARM_TDO	
GPIO25	Low	<reserved>	SD17	<reserved>	SD1_DAT1	ARM_TCK	
GPIO26	Low	<reserved>	<reserved>	<reserved>	SD1_DAT2	ARM_TDI	
GPIO27	Low	<reserved>	<reserved>	<reserved>	SD1_DAT3	ARM_TMS	
GPIO28	-	SDA0	SA5	PCM_CLK	<reserved>		
GPIO29	-	SCL0	SA4	PCM_FS	<reserved>		
GPIO30	Low	<reserved>	SA3	PCM_DIN	CTS0		CTS1
GPIO31	Low	<reserved>	SA2	PCM_DOUT	RTS0		RTS1
GPIO32	Low	GPCLK0	SA1	<reserved>	TXD0		TXD1
GPIO33	Low	<reserved>	SA0	<reserved>	RXD0		RXD1
GPIO34	High	GPCLK0	SOE_N / SE	<reserved>	<reserved>		
GPIO35	High	SPI0_CE1_N	SWE_N / SDW_N		<reserved>		
GPIO36	High	SPI0_CE0_N	SD0	TXD0	<reserved>		
GPIO37	Low	SPI0_MISO	SD1	RXD0	<reserved>		
GPIO38	Low	SPI0_MOSI	SD2	RTS0	<reserved>		
GPIO39	Low	SPI0_SCLK	SD3	CTS0	<reserved>		
GPIO40	Low	PWM0	SD4		<reserved>	SPI2_MISO	TXD1