ExpEYES-Junior

Manuel du programmeur

Ajith Kumar B.P

Inter-University Accelerator Centre

New Delhi 110 067

Version 1.1 (26-Oct-2013) - traduction Georges Khaznadar

http://expeyes.in

Contents

1 Introduction **4**

1.1 Le logiciel 4

2 Communication avec le matériel **6**

2.1 Sortie analogique 7

2.1.1 set_voltage(V) 7

2.2 Entrées numériques (IN1, IN2 et SEN) 8

2.2.1 get_state(num_canal) 8

2.3 Sortie numérique (OD1) 8

2.3.1 set_state(num_canal, etat) 8

2.4 Entrées analogiques (A1,A2,IN1,IN2 & SEN) 9

2.4.1 get_voltage(num_canal) 9

2.4.2 get_voltage_time(num_canal) 9

2.4.3 get_voltageNS(num_canal) 10

2.4.4 capture(ch, NP, tg) 10

2.4.5 capture2, capture3 & capture4 11

2.4.6 capture_hr(ch, NP, tg), capture2_hr(ch1, ch2, NP, tg) 11

2.5 Modificateurs de capture 11

2.5.1 set_trigger(trigval) 12

2.5.2 set_trigsource(num_canal) 12

2.5.3 enable_wait_high(num_canal), ..._low(...), ..._falling(...), ..._rising(...) 13

2.5.4 enable_set_high(num_canal), ..._low(...), ..._pulse_high(...), ..._low(...) 13

2.5.5 set_pulsewidth(duree) 14

2.6 Génération de signal 14

2.6.1 set_sqr1(freq), set_sqr2(freq) 14

2.6.2 set_sqrs(freq, <déphasage en pourcentage>) 14

2.6.3 set_sqr1_pwm(rc [, idx]), set_sqr2_pwm(rc[, idx]) 15

2.6.4 set_sqr1_dc(voltage), set_sqr2_dc(voltage) 15

2.6.5 get_frequency(pin) 15

2.7 Transmission infrarouge 16

2.7.1 irsend1(octet) 16

2.7.2 irsend4(byte, byte, byte, byte) 16

2.8 Mesures passives d'intervalle de temps . . . 16
2.8.1 r2ftime(pin1, pin2) , f2rtime(pin1, pin2) . . . 16
2.8.2 r2rtime(pin1, pin2), f2ftime(pin1, pin2) . . . 17
2.8.3 multi_r2rtime(num_canal, a_laisser) . . . 17
2.9 Mesures d'intervalles de temps actives . . . 17
2.9.0.1 set2rtime (<sortie numérique>, <entrée numérique>) . . . 18
2.9.0.2 htpulse2rtime(<sortie numérique>, <entrée numérique>) . . . 18
2.9.0.3 set_pulse_width(duree) . . . 18
2.10 Source de courant de 1mA . . . 18
2.11 Mesures de capacité . . . 19
2.11.1 measure_cap() . . . 19
2.11.2 measure_cv(num_canal, duree, courant) . . . 19
2.11.3 set_current(num_canal, courant) . . . 20
2.12 Mesures de résistance . . . 20
2.12.1 measure_res() . . . 20
2.13 Écriture sur le disque . . . 20
2.13.1 save_data . . . 20

3 Traitement des données **22**
3.0.1 fit_sine . . . 22
3.0.2 fit_dsine . . . 22
3.0.3 fit_exp . . . 23
3.0.3.1 fft . . . 23

4 Expériences **24**
4.1 Réponse transitoire d'un circuit LC . . . 24

Chapter 1

Introduction

La conception d'expEYES est schématisée à la figure 1.1, à côté de la sérigraphie du boîtier et du marquage des connexions d'entrée/sortie expliqué à la table 1.1. Les fonctions pour piloter expEYES, comme mesurer une tension ou une fréquence, régler une tension ou une fréquence, mesurer des intervalles de temps, etc. sont disponibles dans les langages Python et C. L'analyse de données et les fonctions graphiques viennent dans deux modules Python séparés. Les programmes d'application sont développés à l'aide de ces modules.

1.1 Le logiciel

Voici les modules principaux du paquetage expEYES :

- `eyesj.py` : communication avec le matériel
- `eyeplot.py` : fonctions graphiques basées sur le module `Tkinter`
- `eyemath.py` : analyse de données utilisant les modules `numpy` et `scipy`
- `ejlib.c` & `ejlib.h` : bibliothèque C et fichier d'en-tête

On peut les installer à l'aide des fichiers `.tgz` ou des paquetages `.deb` disponibles à http://expeyes.in.

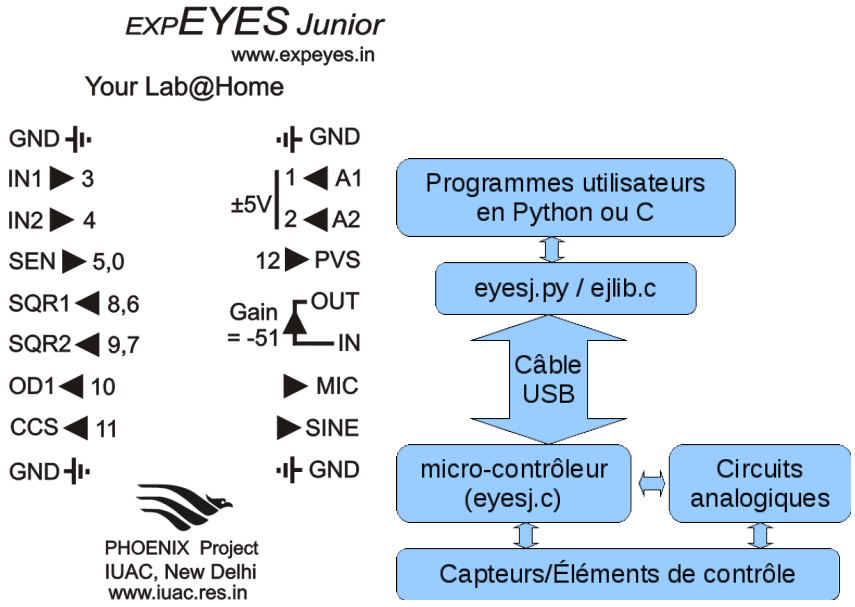


Figure 1.1: Sérigraphie du boîtier expEYES Junior et diagramme en blocs

Borne N°	Nom	Description
1	GND	Masse (Ground)
2	IN1	Entrée analogique/numérique 0 à 5 V, source de courant
3	IN2	Entrée analogique/numérique 0 à 5 V, source de courant
4	SEN	Entrée analogique/numérique 0 à 5V, polarisée avec $5k\Omega$, pour capteurs résistifs
5	SQR1	Sortie de signal carré de $0,7$ Hz à 200 kHz, résistance série 100 Ω
6	SQR2	Sortie de signal carré de $0,7$ Hz à 200 kHz, sans résistance série
7	OD1	Sortie numérique, sans résistance série
8	CCS	Source de courant constant 1 mA avec contrôle ON/OFF
9	GND	Masse (Ground)
10	GND	Masse (Ground)
11	SINE	Sortie sinusoïdale, 150 Hz environ, 4 V
12	MIC	Sortie du microphone, amplifiée 51 fois
13	IN	Entrée de l'amplificateur inverseur, gain maximum 51
14	OUT	Sortie de l'amplificateur (entrée en borne 13)
15	PVS	Sortie de tension programmable de 0 à 5 V
16	A2	Entrée analogique $\pm 5V$
17	A1	Entrée analogique $\pm 5V$
18	GND	Masse (Ground)

Table 1.1: Description des connexions d'entrée/sortie

Chapter 2

Communication avec le matériel

Le module `expeyes.py` contient toutes les fonctions nécessaires pour communiquer avec le matériel, plus quelques fonctions utilitaires. Les fonctions sont à l'intérieur d'une classe, et la fonction statique `open()` renvoie une instance de cette classe si un boîtier expEYES est détecté. Ensuite, les appels de fonctions pour accéder à expEYES sont faits à l'aide cette instance, comme le montre l'exemple ci-dessous.

```
import expeyes.eyesj      # importe la bibliothèque eyes
p = expeyes.eyesj.open()  # renvoie une instance si un boîtier est détecté
print p.get_voltage(1)    # affiche la tension à l'entrée A1
```

Un programme d'exemple en langage C est donné ci-dessous. Il convient de le compiler et de l'exécuter.

```
#include "ejlib.c"
int fd;
int main()
   {
   byte ss[10];
   fd = open_eyesj();
   if(fd < 0)
     {
     fprintf(stderr,"Échec à l'ouverture d'EYES\n");
     exit(0);
     }
   if(get_version(ss) != 0) exit(1);
   printf("%s\n",ss);
}
```

En cas d'erreur, les fonctions Python renvoient `None` (-1 dans le cas de mesures d'intervalles de temps). En cas de succès la donnée est renvoyée. Les fonctions C renvoient zéro en cas de succès, et un code d'erreur dans le cas contraire. La donnée est toujours renvoyée en utilisant

Canal N°	Nom
0	Sortie du comparateur analogique
1	A1
2	A2
3	IN1
4	IN2
5	SEN
6	retour de SQR1
7	retour de SQR2
8	sortie de SQR1
9	sortie de SQR2
10	sortie de OD1
11	contrôle de la sortie CCS
12	retour de PVS

Table 2.1: Signaux et numéros de canal

l'adresse passée à la fonction par le programme appelant. En Python comme en C, on a donné les mêmes noms aux fonctions. La différence principale est dans les résultats renvoyés. En C, il faut passer une adresse pour ça. La fonction ne renvoie que le statut de l'opération. Quelques-unes des fonctions C sont mentionnées ci-dessous. Le plus simple est d'examiner le fichier d'en-tête *ejlib.h*.

Pour chaque fonction, les versions Python et C sont décrites, mais il n'y a pas de code donné en exemple pour C. Chaque fonction communique avec le programme qui tourne sur le micro-contrôleur de la carte expEYES Junior. Les fonctions de communication avec le matériel peuvent être regroupées en gros en entrées analogiques, sorties analogiques, entrées numériques, sorties numériques, mesures d'intervalles de temps, génération de signaux, etc. Pour tracer des graphiques à partit de données d'expEYES, le paquet `python-matplotlib` est utilisé.

Dans les sections suivantes on introduira les fonctionnalités d'expEYES à l'aide d'exemples. Il est OBLIGATOIRE que les tensions appliquées restent dans les limites spécifiées. *Un numéro de canal est assigné pour identifier chaque signal analogique/numérique. Les appels de fonctions utilisent ce numéro pour y accéder.*

2.1 Sortie analogique

La source de tension programmable (Programmable Voltage Source, PVS) peut être réglée à volonté entre 0 et 5 V. La résolution est de 12 bits, ce qui signifie des échelons de 5000/4096, environ $1,25\ mV$.

2.1.1 set_voltage(V)

Règle la tension de sortie à la borne PVS. La valeur du paramètre de tension V doit être dans l'intervalle de 0 à 5. La fonction renvoie la valeur réglée en réalité, en la lisant en retour grâce à une entrée analogique (canal numéro 12).

```
print p.set_voltage(2.5)      # règle PVS à 2,5 V et affiche la valeur obtenue
```

fonction C : byte set_voltage(float v, float* vset); // vset reçoit le retour de PVS

2.2 Entrées numériques (IN1, IN2 et SEN)

On peut les connecter extérieurement à 0 (GND) ou à 5 V, pour fixer le niveau de tension à HAUT ou BAS. Tout ce qui est inférieur à 1 V est considéré comme BAS ou 0. Tout ce qui dépasse $2,5$ V est considéré comme HAUT ou 1. Ces bornes peuvent aussi être configurées comme entrées analogiques.

2.2.1 get_state(num_canal)

Renvoie 0 ou 1, selon le niveau de tension à la borne d'entrée

```
print p.get_state(3)    # affiche le niveau logique de IN1.  IN2 = 4, SEN = 5
print p.get_state(0)    # Renvoie « 1 » si SEN > 1,25 V
```

Le canal 0 représente la sortie du comparateur analogique. L'entrée non-inverseuse de ce comparateur doit être connectée à SEN, l'entrée inverseuse est connectée à $1,25$ V.

Une fonctionnalité puissante des entrées numériques est la possibilité de mesurer des intervalles de temps entre des changements de niveaux avec une résolution de l'ordre de la microseconde. On en reparlera plus loin.

fonction C : byte get_state(byte pin, byte *st); // la variable st renvoie 0 ou 1.

2.3 Sortie numérique (OD1)

On peut en régler le niveau de tension à HAUT ou BAS par programme. Si on y connecte une DEL, utiliser une résistance série de 1 $k\Omega$ pour limiter le courant.

2.3.1 set_state(num_canal, etat)

Cette fonction règle le canal spécifié à l'état « 0 » ou « 1 ».

```
p.set_state(10,1)    # Règle OD1 à HAUT. Le numéro de canal pour OD1 est 10.
```

Les sorties SQR1 (8) et SQR2 (9) peuvent aussi se comporter comme des sorties numériques, pour autant qu'elles ne soient pas utilisées pour générer une signal carré ou un signal PWM.

fonction C : byte set_state(byte pin, byte state); // la borne pin est réglée à 0 ou 1, selon la valeur de state.

2.4 Entrées analogiques (A1,A2,IN1,IN2 & SEN)

Les entrées analogiques A1 et A2 acceptent des tensions comprises entre $-5\ V$ et $+5\ V$. Les entrées IN1, IN2 et SEN acceptent des tensions dans l'intervalle 0 à $5\ V$. On peut lire la tension à chacune de ces entrées, par une lecture simple ou par une lecture multiple en un seul appel de fonction, normalement pour capturer un signal. L'intervalle de temps entre lectures consécutives lors d'une capture peut se régler à la microseconde près.

2.4.1 get_voltage(num_canal)

```
print p.get_voltage(1)  # tension en A1
print p.get_voltage(2)  # tension en A2
print p.get_voltage(3)  # tension en IN1
print p.get_voltage(4)  # tension en IN2
print p.get_voltage(5)  # tension en SEN
print p.get_voltage(6)  # tension à la sortie SQR1
print p.get_voltage(7)  # tension à la sortie SQR2
print p.get_voltage(12) # tension à la sortie PVS
```

Connecter PVS à A1 à l'aide d'un fil électrique et lancer le programme suivant plusieurs fois.

```
import expeyes.eyesj     p = expeyes.eyesj.open()
v = input('Entrer la tension (0 à 5)')
print p.set_voltage(v)   # affiche la tension affectée à PVS
print p.get_voltage(1)   # tension en A1
```

Si la tension est dans l'intervalle de 0 à $5\ V$, utiliser IN1 ou IN2 pour de meilleurs résultats. L'intervalle de tension $\pm 5\ V$ pour les entrées A1 & A2 est converti en un intervalle de tension de 0 à $5\ V$ à l'aide d'un calcul analogique. Les amplificateurs utilisés à cet effet souffrent de petites erreurs de gain et de décalage. De plus la résolution est divisée par deux quand l'intervalle de tension est doublé. L'entrée SEN possède une résistance de polarisation de $5\ k\Omega$ connectée à $5\ V$, pour y brancher des photo-transistors ou des capteurs résistifs.

fonction C : byte get_voltage(byte ch, float* v)

2.4.2 get_voltage_time(num_canal)

Cette fonction renvoie un timbre à date, issu de l'horloge du PC, et la tension dans un tuple. Elle est utile pour les applications de suivi de tension.

fonction C : byte get_voltage(byte ch, int* t, float* v)

2.4.3 get_voltageNS(num_canal)

La fonction get_voltage() mentionnée plus haut mesure le tension après avoir placé le micro-contrôleur en mode SLEEP, pour une meilleure précision. Ça a pour conséquence d'arrêter les signaux émis sur les sorties SQR1 et SQR2. Si on peut accepter ça pour une expérience particulière, on peut utiliser cette fonction.

```
print p.get_voltageNS(1)  # tension en A1
```

fonction C : byte get_voltageNS(byte ch, float* v)

2.4.4 capture(ch, NP, tg)

L'argument *ch* est le numéro de canal de l'entrée, *NP* est le nombre de mesures et *tg* est la durée entre deux mesures en microseconde. Deux listes sont renvoyées par cette fonction, qui contiennent le temps (en ms) et la tension (en V). Les appels à *capture* renvoient des données analogiques avec une résolution de 8 bit. La valeur maximale de NP est 1800, limitée par la RAM du micro-contrôleur.

La plus petite valeur de *tg* est 4 μs. Il faut choisir la valeur de *tg* selon le signal à capturer. Par exemple, une période d'un signal sinusoïdal dure 1000 μs. Une valeur de *tg* de 20 donnera 50 points de donnée par période.

Connecter SINE à A1 et lancer le programme suivant :

```
from pylab import * import expeyes.eyesj p = expeyes.eyesj.open()
t,v = p.capture(1,300,100)
plot(t,v)     # à l'aide de pylab
show()        # à l'aide de pylab
```

Borne	num_canal	Intervalle (V)
A1	1	-5 à +5
A2	2	-5 à +5
IN1	3	0 à 5
IN2	4	0 à 5
SEN	5	0 à 5
SQR1(lecture)	6	0 à 5
SQR2(lecture)	7	0 à 5

Si la tension à mesurer est dans l'intervalle de 0 à 5 V, utiliser IN1 ou IN2 pour une meilleure résolution. L'entrée SEN possède une résistance de polarisation de 5 $k\Omega$ connectée à 5 V. On peut calculer la valeur d'une résistance connectée entre SEN et GND, à partir de la tension mesurée, grâce à la loi d'Ohm.

fonction C : byte capture(int ch, int ns, int tg, float* data);

La variable *data* renvoie une table de 2**ns* éléments de type float, d'abord *ns* coordonnées de temps, puis *ns* coordonnées de tension. C'est de la responsabilité du programme appelant d'allouer une table de taille suffisante. *capture_hr()* renvoie aussi les données dans ce même format.

2.4.5 capture2, capture3 & capture4

Ces fonctions capturent plusieurs canaux en même temps, avec corrélation temporelle. La valeur maximale de *NP* pour *capture4* = 1800/4 = 450. La plus petite valeur de *tg* est de 4 μs par canal, *capture4* aura donc une valeur minimale de *tg* égale à 16.

```
t1,v1,t2,v2 = capture2(ch1, ch2, NP, tg)
t1,v1,t2,v2 = capture2_hr(ch1, ch2, NP, tg)
t1,v1,t2,v2,t3,v3 = capture3(ch1, ch2, ch3, NP, tg)
t1,v1,t2,v2,t3,v3,t4,v4 = capture4(ch1, ch2, ch3, ch4, NP, tg)
```

fonction C : byte capture2(int ch1, int ch2, int ns, int tg, float* data);

La variable *data* renvoie une table de 2(2**ns*) éléments de type float. Les premiers (2**ns*) sont le temps et la tension pour le canal *ch1* et les suivants (2**ns*) pour le canal *ch2*. Les fonctions *capture3* et *capture4* renvoient les données d'une façon similaire.

2.4.6 capture_hr(ch, NP, tg), capture2_hr(ch1, ch2, NP, tg)

Ces deux fonctions capturent les données avec un résolution supérieure (12 bit). Dans ce cas chaque valeur occupe deux octets et la valeur maximale de *NP* est 900 pour *capture_hr*, et 450 pour *capture2_ht*. Les versions en haute résolution ne sont PAS disponibles pour *capture3* et *capture4*.

```
t1,v1 = capture_hr(ch1, 900, 10)
t1,v1,t2,v2 = capture2_hr(ch1, ch2, 450, 20)
plot(t1,v1, t2,v2)
show()
```

On peut déterminer l'amplitude et la fréquence du signal d'entrée en réalisant un « fit » mathématique des données capturées à l'aide de l'équation d'un signal sinusoïdal $V = V_0 \sin(2\pi ft + \theta) + C$. En capturant 4 à 5 périodes, on peut aller jusqu'à obtenir la fréquences avec une incertitude de 0,1%.

2.5 Modificateurs de capture

Quand un signal périodique est capturé, le point de départ pourrait être à n'importe quelle tension, entre les valeurs extrémales. Pour implémenter une oscilloscope, il faut s'assurer que

Action	Code	Description
AANATRIG	0	Seuil (trigger) pour une entrée analogique
ASET	1	Force la sortie spécifiée à HAUT
ACLR	2	Force la sortie spécifiée à BAS
APULSEHT	3	Envoie une impulsion High True à une sortie
APULSELT	4	Envoie une impulsion Low True à une sortie
AWAITHI	5	Attend un niveau HAUT sur l'entrée spécifiée
AWAITLO	6	Attend un niveau BAS sur l'entrée spécifiée
AWAITRISE	7	Attend un front montant sur l'entrée spécifiée
AWAITFALL	8	Attend un front descendant sur l'entrée spécifiée

Table 2.2: Modificateurs de capture

le point de départ est prévisible, sans quoi la trace sera erratique. Ceci est un exemple simple de modificateur de capture. ExpEYES implémente plusieurs autres types de modificateurs de capture pour augmenter les fonctionnalités des fonctions de capture. L'idée de base est de réaliser une certaine action juste avant de démarrer la capture du signal. Les types importants de modificateurs (ou actions) sont :

- Seuil de déclenchement (trigger) analogique sur n'importe quel canal d'entrée, le seuil peut être choisi par l'utilisateur.
- Attente d'état HAUT, BAS, de front descendant ou montant sur les entrées IN1, IN2, SEN, SQR1 ou SQR2
- Mettre à HAUT, BAS, ou envoyer une impulsion sur une des sorties numériques, le plus souvent OD1. SQR1 & SQR2 peuvent aussi être utilisées comme sorties numériques si leur fréquence est réglée à zéro.

enable_action(<action>, <entrée/sortie spécifiée>) est l'appel de fonction à utiliser pour enregistrer des actions. L'appel de *disable_actions()* efface toutes les actions enregistrées et la capture revient à son fonctionnement par défaut (seuil analogique sur le canal capturé). Par commodité, on a défini des fonctions supplémentaires qui appellent en interne la fonction *enable_action()*.

2.5.1 set_trigger(trigval)

Règle le niveau du seuil (trigger) analogique, pour la fonction de capture. Si la valeur spécifiée de tension n'est pas trouvée à l'entrée durant la période d'attente, la capture est faite en ignorant la condition de seuil.

```
p.set_trigger(2048)   # 0 à 4095 est l'intervalle analogique
```

2.5.2 set_trigsource(num_canal)

La source d'entrée à utiliser pour le déclenchement par un seuil analogique. Ce n'est pas forcément l'entrée qui est capturée. Le code donné en exemple ci-dessous démontre l'effet de

cette fonction. Connecter SINE à A1 avant de lancer le programme.

```
from pylab import * import expeyes.eyesj p = expeyes.eyesj.open()
ts = 1      # relancer le programme après avoir remplacé ça par 2
p.set_trig_source(ts)
t,v = p.capture(1,300,50)
plot(t,v)
t,v = p.capture(1,300,50)
plot(t,v)
show()
```

Les traces ne se recouvriront pas si la source retenue pour le seuil est un autre canal quelconque, s'il n'y a pas de corrélation temporelle entre les deux entrées.

fonction C : byte set_trig_source(byte ch);

2.5.3 enable_wait_high(num_canal), ..._low(...), ..._falling(...), ..._rising(...

Quand on appelle cette fonction, tous les appels suivants de capture provoquent l'attente d'un état HAUT/BAS ou d'un front montant/descendant avant le début de la numérisation.

```
p.enable_action(1, 11)    # Démarrer CCS avant la capture
p.enable_wait_rising(3)   # Attendre un front montant sur IN1
p.disable_actions()       # effacer tous les modificateurs
```

fonction C : byte enable_wait_high(byte ch);

2.5.4 enable_set_high(num_canal), ..._low(...), ..._pulse_high(...), ..._low(...)

Dans certaines applications, il peut être nécessaire de forcer une sortie numérique à l'état HAUT ou BAS, ou émettre une impulsion, de largeur réglée par une autre fonction, avant que la numérisation commence. Capturer la tension aux bornes d'un condensateur durant sa charge / sa décharge est une application typique de cette fonctionnalité. Connecter un condensateur de 1 μF entre A1 et GND. Connecter une résistance de 1 $k\Omega$ entre OD1 et A1 puis lancer le programme suivant :

```
from pylab import * import expeyes.eyesj p = expeyes.eyesj.open()
p.set_state(10,1)       # Force OD1 à HAUT
p.enable_set_low(10)    # OD1 deviendra BAS avant la capture
t,v = p.capture(1,200,20)
plot(t,v)
show()
```

fonction C : byte enable_set_high(byte ch);

2.5.5 set_pulsewidth(duree)

Règle la *duree* de l'impulsion qui est envoyée sur la sortie numérique avant la capture, en microseconde, jusqu'à 250.

```
p.set_pulsewidth(100)  # règle la durée d'impulsion
```

fonction C : byte set_pulsewidth(u16 width);

2.6 Génération de signal

ExpEYES peut générer des signaux sur SQE1 et SQR2. La fréquence peut varier de $0,7\ Hz$ à $100\ kHz$. Toutes les valeurs intermédiaires ne sont PAS possibles comme le signal de sortie est généré par des temporisateurs et des comparateurs. La fonction renvoie la valeur effectivement réglée, au plus près de la valeur demandée. La sortie SQR1 a une résistance série de 100 Ω pour limiter le courant, mais SQR2 est connecté directement à la borne.

2.6.1 set_sqr1(freq), set_sqr2(freq)

Génère un signal carré, de rapport cyclique 50 %, sur SQR1/SQR2. La sortie SQR1 a une résistance série de 100 Ω. Régler *freq* = 0 force la sortie à HAUT et régler *freq* = -1 la force à BAS. Dans ces deux cas, le temporisateur/compteur est désactivé et la sortie est configurée comme une sortie numérique ordinaire.

```
import expeyes.eyesj     p = expeyes.eyesj.open()
print p.set_sqr1(1000)
```

fonction C : byte set_sqr1(float freq, float *fset);

La valeur désirée est passée par *freq*, et après l'appel *fset* contiendra la valeur réelle du réglage de fréquence.

2.6.2 set_sqrs(freq, <déphasage en pourcentage>)

Génère des signaux carrés de même fréquence sur SQR1 et SQR2. Le déphasage entre les deux signaux peut être réglé en pourcentage de la période.

```
p.set_sqrs(1000,50)    # Deux signaux en opposition de phase
```

fonction C : byte set_sqrs(float freq, float diff, float *fset);

2.6.3 set_sqr1_pwm(rc [, idx]), set_sqr2_pwm(rc[, idx])

SQR1 et SQR2 peuvent être configurés pour faire un signal carré modulé en largeur d'impulsion (Pulse Width Modulation, PWM). Le rapport cyclique est spécifié en pourcentage. La fréquence est 488 Hz par défaut, parce que le second argument est réglé à 14 par défaut. C'est l'index du bit du compteur qui est utilisé pour piloter le PWM. En spécifiant le deuxième argument, on peut changer la fréquence. Le diminuer de 1 double la fréquence et l'augmenter de 1 divise la fréquence par deux.

```
print p.set_sqr1_pwm(20)      # 488Hz, rapport cyclique 20%
print p.set_sqr1_pwm(50, 15)  # 244Hz, rapport cyclique 50%
print p.set_sqr1_pwm(50, 13)  # 976Hz, rapport cyclique 50%
```

fonction C : byte set_sqr1_pwm(byte dc);

2.6.4 set_sqr1_dc(voltage), set_sqr2_dc(voltage)

SQR1 et SQR2 peuvent être configurés pour générer une tension continue, grâce à un filtrage externe, depuis un signal PWM. La tension, entre 0 et 5 V, est spécifiée dans l'argument.

```
print p.set_sqr1_dc(2)   # 7.8 kHz, rapport cyclique 40%
```

Un filtrage du signal produira une tension continue. Connecter 10 $k\Omega$ entre SQR1 et IN1 et 100 μF entre IN1 et GND.

```
print p.get_voltage(3)   # tension en IN1
```

La tension de sortie dépend de la tension d'alimentation fournie par le câble USB. Régler cette tension à 3 V signifie en fait 60 % de la tension d'alimentation. La relecture par IN1 permet de connaître la valeur correcte.

fonction C : byte set_sqr1_dc(float volt)

2.6.5 get_frequency(pin)

Mesure la fréquence d'un signal carré entre 0 et 5 V connecté à IN1, IN2 ou SEN. On peut aussi mesurer la fréquence des sorties SQR1 et SQR2 en utilisant les canaux de numéros 6 et 7 respectivement. Connecter SQR1 à IN1 et lancer le programme suivant :

```
import expeyes.eyesj     p = expeyes.eyesj.open() p.set_sqr1(1000)
print p.get_frequency(3)      # fréquence du signal carré en IN1
print p.get_frequency(6)      # fréquence en SQR1, identique à l'autre
```

fonction C : byte get_frequency(byte pin, float *fr)

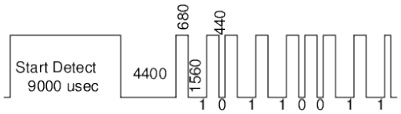
2.7 Transmission infrarouge

La sortie SQR1 supporte deux protocoles de transmission infrarouge à 38 kHz. L'un d'entre eux est une transmission non-standard à 1 octet, qui peut être reçu par un autre programme fonctionnant sur un micro-contrôleur ATmega32. On peut l'utiliser pour contrôler un appareil depuis expEYES Junior.

2.7.1 irsend1(octet)

Envoie l'*octet* par SQR1. Connecter simplement une DEL IR entre SQR1 et GND et lancer la commande.

Pour signifier un *start*, le signal normalement à 38 kHz est bloqué à l'état HAUT durant 9000 μs, puis à l'état BAS durant 440 μs. Ensuite, c'est un état HAUT durant 680 μs suivi d'un état BAS durant 1560 μs pour transmettre un 1, sinon un état BAS durant 440 μs pour transmettre un 0. Ce processus se répète 8 fois, en commençant par le bit le plus significatif (Most Significant Bit, MSB) à transmettre. La séquence se termine par une impulsion longue de 340 μs pour signifier un *end*. Voici ce que reçoit un programme[1] disponible sur le site web.



2.7.2 irsend4(byte, byte, byte, byte)

Les séquences *start* et *end* sont identiques à *irsend1()* mais au lieu d'un octet, quatre octets sont envoyés lors d'une seule transmission. En choisissant les nombres correctement, on peut télécommander des TV ou d'autres appareils ainsi.

2.8 Mesures passives d'intervalle de temps

Les entrées numériques peuvent être utilisées pour mesurer des intervalles de temps entre des transitions d'un niveau à l'autre avec une résolution temporelle de l'ordre de la microseconde. Les transitions qui définissent le début et la fin peuvent être sur la même borne ou sur des bornes différentes.

2.8.1 r2ftime(pin1, pin2) , f2rtime(pin1, pin2)

r2ftime renvoie le délai en microseconde entre un front montant sur la borne *pin1* et un front descendant sur la borne *pin2*, les numéros de canal correspondants étant donnés comme arguments. Les bornes peuvent être identiques ou distinctes. De la même façon, *f2rtime()* mesure le délai entre un front descendant et un front ascendant.

Connecter SQR1 à IN1 lancer le programme suivant, qui devrait afficher environ 500 μs.

[1] http://expeyes.in/sites/default/files/debs/recv.c

```
import expeyes.eyesj     p = expeyes.eyesj.open()
p.set_sqr1(1000)        # 1kHz, T=1 ms. demi-période = 500 µs.
print p.r2ftime(3,3)
```

fonction C : byte r2ftime(byte pin1, byte pin2, float *ti)

2.8.2 r2rtime(pin1, pin2), f2ftime(pin1, pin2)

r2rtime renvoie le délai en microseconde entre un front montant et un autre front montant. Les bornes ne doivent PAS être identiques. Le programme qui suit illustre comme utiliser ça pour mesurer un délai entre deux transitions.

```
import expeyes.eyesj     p = expeyes.eyesj.open()
p.set_sqrs(1000, 25)      # 1000Hz sur SQR1 & SQR2. Délai de 25%, soit 250 µs
print p.r2rtime(6,7)      # Les canaux 6 & 7 sont les retours de SQR1 et SQR2
```

2.8.3 multi_r2rtime(num_canal, a_laisser)

Mesure l'intervalle de temps entre deux fronts montants d'un signal appliqué à une entrée numérique. Le deuxième argument est le nombre de fronts montants à laisser passer entre les deux fronts montants pris en compte. De cette façon on peut décider du nombre de périodes à mesurer.

Connecter SQR1 à IN1 et lancer le programme suivant :

```
import expeyes.eyesj     p = expeyes.eyesj.open() p.set_sqr1(1000)
a = p.multi_r2rtime(3)   # durée pour 1 période, en µs
b = p.multi_r2rtime(3,9) # durée pour 10 périodes, en µs
print 10.0e6/a           # fréquence en Hz
```

Pour un signal d'entrée périodique *multi_ r2rtime* renvoie le temps pour dix périodes (9 fronts montants intermédiaires sont laissés de côté). On peut utiliser ça pour des mesures de fréquence. On peut augmenter la précision en mesurant un nombre de périodes plus grand.

fonction C : byte multi_r2rtime(byte pin, byte skip, float *ti)

2.9 Mesures d'intervalles de temps actives

Pendant certaines expériences, on veut initier une action et mesurer l'intervalle de temps jusqu'au résultat de cette action. Ces fonctions sont utiles dans des expériences comme la détermination de la gravité par temps de vol et la mesure de la vitesse du son à l'aide de disques pézoélectriques ultrasonores.

2.9.0.1 set2rtime (<sortie numérique>, <entrée numérique>)

Ça force la sortie numérique spécifiée à HAUT et attend l'arrivée d'un état HAUT sur l'entrée numérique. Connectez une résistance de 1 $k\Omega$ de OD1 à IN1 et un condensateur de 1 μF de IN1 à GND.

```
p.set2rtime(10, 3)
```

2.9.0.2 htpulse2rtime(<sortie numérique>, <entrée numérique>)

```
int htpulse2rtime(out, in)
```

Envoie une impulsion High True sur *out* (SQR1, SQR2 ou OD1) et attend un front montant/descendant sur *in* (IN1, IN2 ou SEN). La durée de l'impulsion est réglée par *set_pulsewidth()*. À l'allumage cette durée est de 13 μs. Le niveau logique initial de *out* devrait être réglé selon la catégorie d'impulsion.

De même, il existe *htpulse2ftime()*, *ltpulse2rtime()* et *ltpulse2ftime()*.

```
p.set_pulse_width(1)
print p.htpulse2rtime(10, 3)
```

mesure l'intervalle de temps depuis une impulsion High True de 1 μs jusqu'au front montant sur IN1.

2.9.0.3 set_pulse_width(duree)

Règle la largeur de l'impulsion, en microseconde, à utiliser avec les fonctions *htpulse2rtime()*, *htpulse2ftime()*, *ltpulse2rtime()*, *ltpulse2ftime()*.

```
p.set_pulse_width(10)
```

2.10 Source de courant de 1mA

La source de courant de 1 mA peut être mise dans l'état ON ou OFF par le canal de numéro 11, comme montré ci-dessous :

```
p.set_state(11, 1)    # met CCS dans l'état ON (allumé)
```

On peut tracer le graphique de la charge linéaire d'un condensateur de 1 μF en le connectant entre CSS et GND, et en lançant le programme suivant. Connecter aussi CSS à IN1 pour mesurer la tension.

```
from pylab import *
import expeyes.eyesj, time
p = expeyes.eyesj.open()
```

```
p.set_state(11,0)      # force CCS à l'état OFF (éteint)
time.sleep(1)          # attend la décharge complète du condensateur
p.enable_set_high(11)  # met en route CSS juste avant la capture
t1,v1= p.capture_hr(3,500,10)
plot(t1,v1)
show()
```

2.11 Mesures de capacité

La borne IN1 peut être utilisée pour mesurer la capacité d'un condensateur, dans un intervalle de cent à plusieurs millier de picofarad. C'est fait en utilisant une source de courant constant interne programmable.

2.11.1 measure_cap()

Connecter le condensateur entre IN1 et GND et lancer la fonction *measure_ cap()*.

```
import expeyes.eyesj
p = expeyes.eyesj.open()
print p.measure_cap()
```

La capacité est mesurée en chargeant le condensateur à l'aide d'une source de courant constant de $5,5\ \mu A$ pendant une durée fixe. La charge totale est donnée par $Q = It = CU$. Si U,I et t dont connus, C peut être calculé. La valeur de la source de courant peut être légèrement différente de $5,5\ \mu A$, et la borne nue, avec ses connexions, possède aussi un peu de capacité propre. On tient compte de ces erreurs en faisant une calibration à l'aide d'un condensateur de capacité connue. Les facteurs d'erreur sont enregistrés dans la mémoire EEPROM du microcontrôleur.

2.11.2 measure_cv(num_canal, duree, courant)

C'est une version plus souple de *measure_ cap*, qui autorise à ajuster la source de courant branchée à IN1 ou IN2. La source de courant est activée pour *duree* microseconde. Le dernier argument peut être au choix $0,55$; $5,5$; 55; 550 (μA). Cette fonction renvoie la tension à l'entrée sélectionnée après avoir appliqué le courant pendant la durée spécifiée.

Selon la valeur du condensateur branché, on doit sélectionner la durée et le courant de telle façon que la tension développée soit comprise entre 2 et 4 V, pour de bons résultats. Connecter un condensateur de 330 pF entre IN1 et GND puis lancer le programme suivant :

```
import expeyes.eyesj
p = expeyes.eyesj.open()
print p.measure_cv(3, 200, 5.5)  # on a trouvé 3,017 V
```

La capacité peut être calculée à l'aide de l'expression $Q = CU$ et $Q = I \times t$.

$$C = \frac{I \times t}{U} = \frac{5,5 \times 10^{-6} \times 200 \times 10^{-6}}{3,017} = 364 \times 10^{-12} = 364\ pF$$

Quand on soustrait la capacité à vide (32 pF), il reste 332 pF.

2.11.3 set_current(num_canal, courant)

Cette fonction active la source de courant interne sur IN1 ou IN2. Cette source de courant constant peut être utilisée pour pour mesurer le courant avec un autre appareil. Le retour de tension ne permet pas cette mesure directement. En utilisant un ampèremètre connecté entre IN1 et GND, on peut trouver par exemple des valeurs de courant de $5,5\ \mu A$, 47 μA et 470 μA, soit un peu moins que la spécification, dans les valeurs élevées.

2.12 Mesures de résistance

L'entrée SEN est connectée en interne à 5 V à travers une résistance de 5100 Ω. On peut calculer la valeur d'une résistance connectée entre SEN et GND en utilisant la loi d'Ohm. Cependant, la valeur de la résistance interne peut être légèrement différente de 5100 Ω à cause de la tolérance des composants.

2.12.1 measure_res()

L'entrée SEN est connectée en interne à 5 V à travers une résistance de 5100 Ω. Si on connecte une résistance externe entre SEN et GND on réalise un diviseur de tension. On peut calculer la valeur d'une résistance connectée entre SEN et GND en utilisant la loi d'Ohm.

Cette fonction renvoie la valeur d'une résistance connectée entre SEN et GND, à l'aide de l'équation :

$R_{ext} = R_{int} * U_{SEN}/(5.0 - U_{SEN})$

2.13 Écriture sur le disque

2.13.1 save_data

Les données mesurées peuvent être mises sous la forme [[x1,y1], [x2,y2],....] où x et y sont des vecteurs, pour les enregistrer dans un fichier texte.

Cette fonction enregistre les données renvoyées par les fonctions de capture dans un fichier texte. Le nom de fichier par défaut est 'plot.dat', on peut le modifier en utilisant un deuxième argument. Connecter SINE à A1 et lancer le programme suivant :

```
import expeyes.eyesj     p = expeyes.eyesj.open()
t,v = p.capture(1, 200, 100)
```

```
p.save([[t,v]], 'sine.dat')
```

ouvrir le fichier en utilisant la commande

```
$ xmgrace sine.dat
```

Chapter 3

Traitement des données

Les données acquises à l'aide d'expEYES sont analysées en utilisant diverses techniques mathématiques comme l'ajustement de modèle par la méthode des moindres carrés (fit), la transformation de Fourier, etc. Le module nommé `eyemath.py` le fait à l'aide de fonctions du paquet `scipy`. La plupart des fonctions acceptent les formats de données renvoyés par les fonctions de capture.

3.0.1 fit_sine

Modèle sinusoïdal.

Accepte deux vecteurs [x] et [y] et essaie de réaliser une modélisation des données avec l'équation $A\sin(2\pi ft+\theta)+C$, en ajustant par la méthode des moindres carrés. Renvoie les données modélisées et la liste de paramètres$[A, f, \theta, C]$. Connecter SINE à A1 lancer le programme suivant :

```
from pylab import *
import expeyes.eyesj, expeyes.eyemath as em p = expeyes.eyesj.open()
t,v= p.capture(1,400,100)
vfit, par = em.fit_sine(t,v)
print par        # A, f, θ, C
plot(t,v)        # les données brutes
plot(t,vfit)     # les données modélisées
show()
```

la fréquence par[1] est en kHz, comme le temps est donné en milliseconde.

3.0.2 fit_dsine

Modèle sinusoïdal amorti.

Accepte deux vecteurs [x] et [y] et essaie de modéliser les données par l'équation $A = A_0\sin(2\pi ft+\theta)\times exp(-dt)+C$, en ajustant par la méthode des moindres carrés. Renvoie les données modélisées et la liste de paramètres$[A, f, \theta, C, d]$. `par[1]` est la fréquence en kHz, comme le temps est donné en milliseconde, et `d` est le facteur d'amortissement.

3.0.3 fit_exp

Modèle exponentiel.

Accepte deux vecteurs [x] et [y] et essaie de modéliser les données par l'équation $A = A_0 \exp(kt) + C$, en ajustant par la méthode des moindres carrés. Renvoie les données modélisées et la liste de paramètres $[A, k, C]$. Connecter un condensateur de 1 μF entre A1 et GND, une résistance de 1 $k\Omega$ entre OD1 et A1 et lancer le programme suivant :

```
from pylab import * import expeyes.eyesj, expeyes.eyemath as em
p = expeyes.eyesj.open()
p.set_state(10,1)           # Force OD1 à HAUT
p.enable_set_low(1)         # OD1 deviendra BAS avant capture
t,v = p.capture(1,200,20)
plot(t,v)
vfit, par = em.fit_exp(t,v)
print par
plot(t,v)                   # les données brutes
plot(t,vfit)                # les données modélisées
show()
```

$\frac{-1}{par[1]}$ est la constante de temps RC en seconde.

3.0.3.1 fft

Réalise une transformée de Fourier sur un ensemble de données. L'intervalle d'échantillonage en milliseconde est le deuxième argument. Renvoie le spectre de fréquence, c'est à dire l'intensité relative de chaque composante fréquentielle. Connecter SINE à A1 et lancer le programme suivant :

```
from pylab import *
import expeyes.eyesj, expeyes.eyemath as em
ns = 1000                    # nombre de points à capturer
tg = 100                     # durée entre échantillons en µs
p = expeyes.eyesj.open() t,v= p.capture(1, ns, tg)
x,y = em.fft(v, tg * 0.001)  # tg en ms
plot(t,v)                    # Donnée brutes
plot(x,y)                    # spectre de fréquence
show()
```

Le spectre de fréquence devrait avoir un pic à la fréquence $f \simeq 150\ Hz$. Un petit pic à la fréquence double peut être visible.

Modifiez le montage pour montrer le spectre de fréquence d'un signal carré.

Chapter 4

Expériences

La plupart des expériences décrites dans le manuel de l'utilisateur peuvent être faites en écrivant quelques lignes de code Python.

4.1 Réponse transitoire d'un circuit LC

Connecter un solénoïde entre OD1 et A1, un condensateur entre A1 et GND.

```
NP = 200                        # nombre d'échantillons
tg = 10                         # délai entre ceux-ci, faire en sorte que NP*tg soit
from pylab import *
import expeyes.eyesj, expeyes.eyemath as em
p = expeyes.eyesj.open()
p.set_state(10,1)
p.enable_set_low(10)             # OD1 deviendra BAS avant capture
t,v = p.capture_hr(1,NP,tg)     # choisir NP*tg selon la constante de temps RC
plot(t,v)
vf, par = em.fit_exp(t,v)       # modèle exponentiel
plot(t, vf,'r')
print abs(1./par[1])            # affiche la valeur de RC
show()
```