Développement sur le port d'entrées-sorties de la raspberry PI4

1) But

Programmer les broches d'usages générales en entrées sorties et gérer les interruptions sur celles-ci .

2) Objectifs

L'étudiant devra être capable :

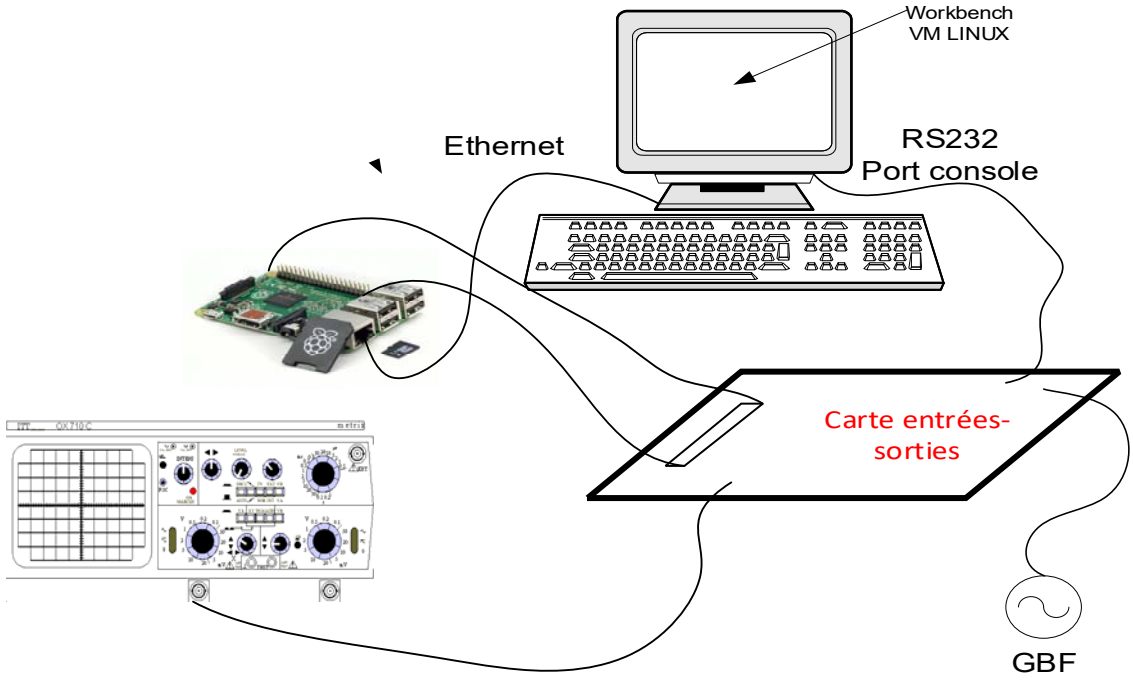
- D'élaborer des programmes avec des routines en langage C qui nous permettent de gérer les entrées sorties sur la raspberry PI4 sous l'environnement VxWorks.
- De mettre en œuvre les dispositifs de test afin de vérifier la conformité des programmes élaborés.

3) Pré-requis

- Cours sur VxWorks notamment le GPIO et les interruptions

- Cours sur le langage C.

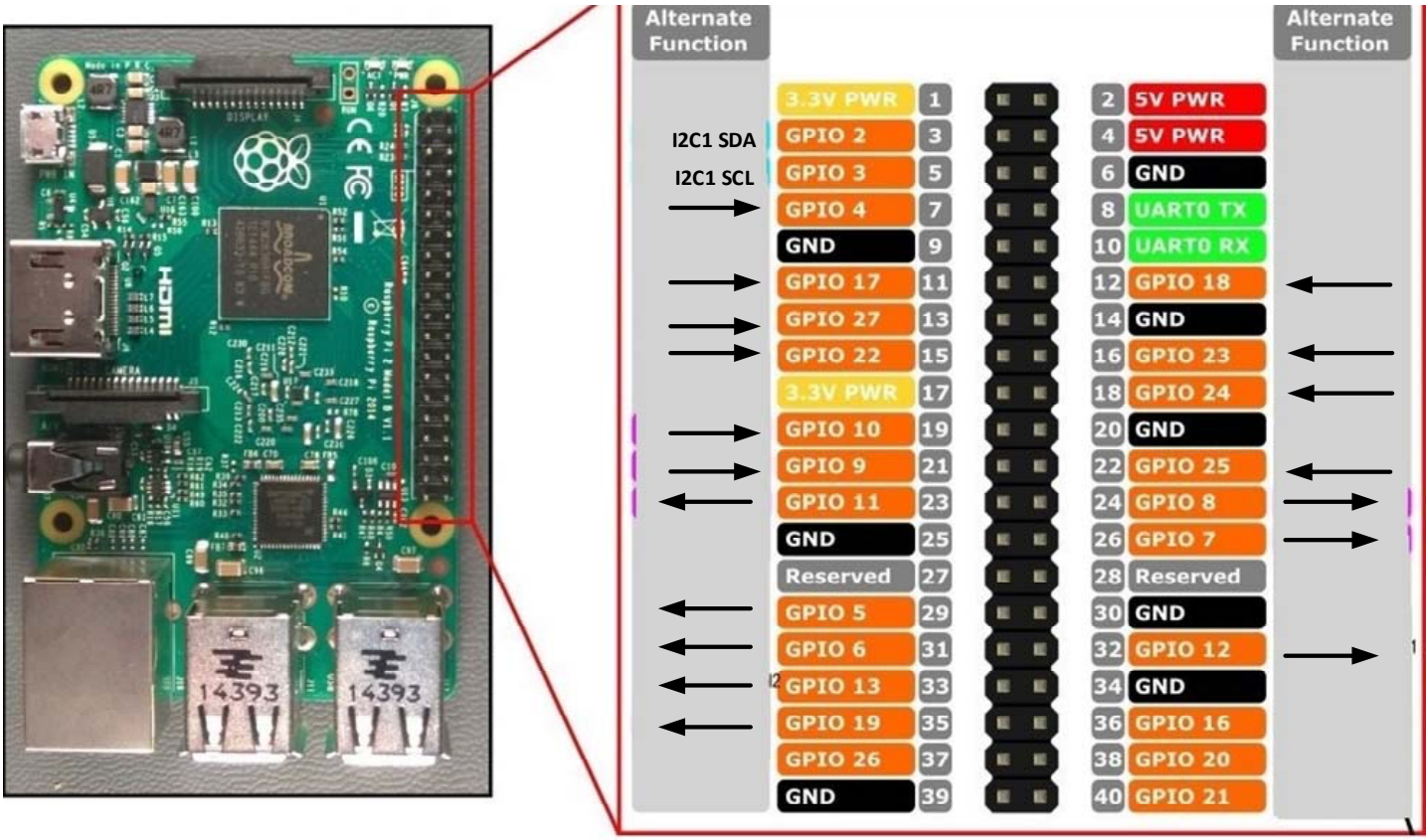
4) Schéma de câblage

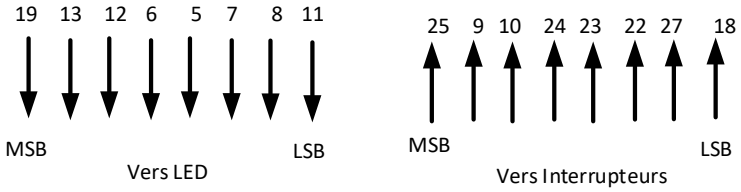


-

5) Configuration du port d'entrées sorties

On Souhaite dans la première partie configurer le port d'extension avec les broches d'entrées-sorties programmées de la manière suivante :

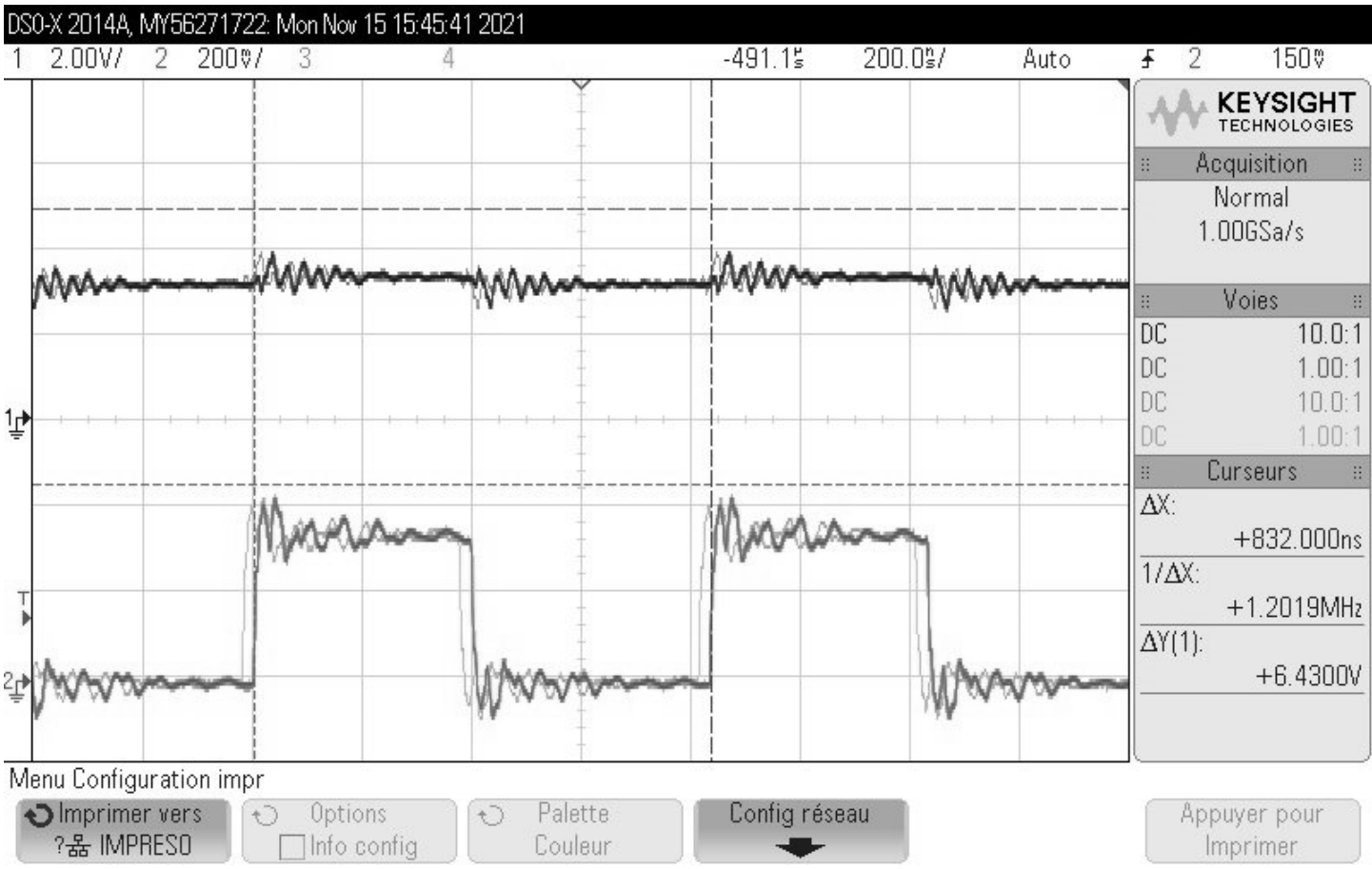




5.1) En vous servant des fonctions du fichier gpiolib.h (document annexe N°1), complétez le programme LED.c document réponse N°1 afin de faire clignoter la LED N°2 (broche 11) à l'infini

5.2) Testez votre programme et mesurer la fréquence de clignotement de la LED

✍ Fréquence=



5.3) On souhaite développer 3 fonctions qui vont nous permettre d'exploiter la carte d'entrées sorties :

- ✓ La fonction init_GPIO qui permet de configurer les ports d'entrées sorties

  void init_gpio(void)

- ✓ La fonction interrupteur qui retourne un octet image de l'état logique des interrupteurs

```
unsigned char interrupteur(void)
```

- ✓ La fonction affichage qui reçoit un octet qui permettra d'allumer ou d'éteindre les LEDs

```
void affichage(unsigned char c)
```

5.3) On donne document annexe N°2 la fonction init_GPIO, une erreur s'est glissée dans cette fonction trouvez et modifiez cette erreur

5.4) On donne document annexe N°3 la fonction affichage, en vous aidant de cette fonction, codez la fonction interrupteur

5.5) Proposez un programme document réponse N°2 (testgpio) qui permet de saisir un entier au clavier et d'afficher sa valeur en binaire sur les LEDs. Ce programme nous permettra aussi d'afficher par un printf l'image binaire des interrupteurs en héxadécimal

6) Configuration du port d'entrées sorties

On souhaite déclencher une interruption sur la broche 4 du GPIO. On associera une routine d'interruption ISR nommée `routineIT`

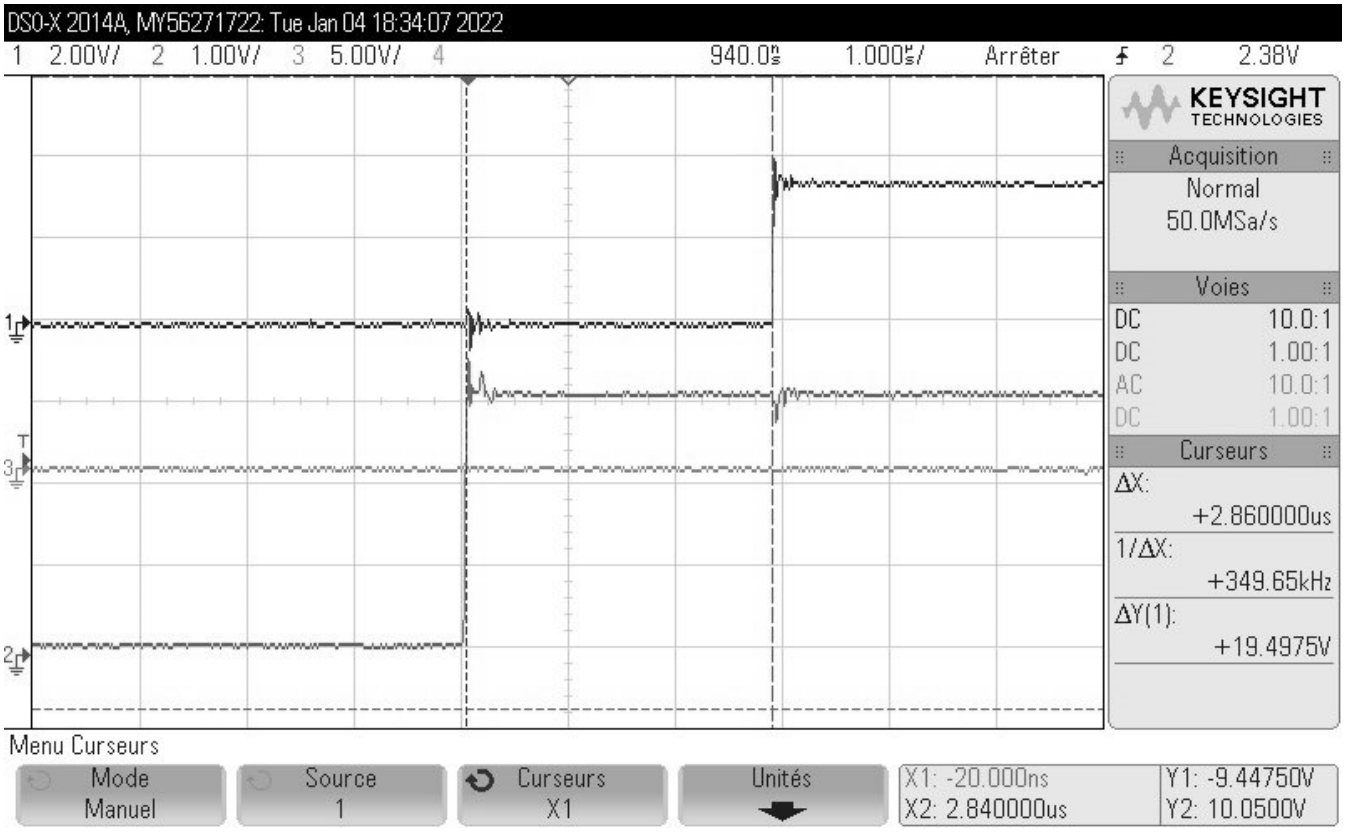
L'interruption et cette routine seront déclenchées sur un front montant grâce à un signal créneau de 3.3V sur la broche DO1.

A chaque déclenchement la routine permettra d'allumer ou d'éteindre successivement le LED LD2

6.1) En analysant une partie du device tree sous Workbench4 (document annexe N°4), sur quel type de front déclenche l'interruption ? Quels sont les numéros de vecteur associés aux contrôleur GPIO

6.2) En vous aidant du fascicule VxWorks 7 (chapitre interrution), complétez le programme swint.c afin de répondre à l'application envisagée.

6.3) Mesure le temps de latence d'une interruption (temps qui sépare le déclenchement de l'interruption de sa prise en compte par la routine d'interruption)



✍ temps de latence interruption=

7) Préparation

Complétez les programmes fournis

Document réponse N°1

```
#include <vxWorks.h>
#include <unistd.h>
#include <stdio.h>
#include <timers.h>
#include "gpiolib.h"

#define PIN 11


void tache_A(void);
void tache_B(void);


void led1(void)
{
	TASK_ID tid_a, tid_b;

	pinMode(_____________________________);

		tid_a=taskSpawn("tache_A",101,0,5000,(FUNCPTR)tache_A,0,0,0,0,0,0,0,0,0,0);
	if(tid_a==ERROR)
		{
		perror("pb taskspawn 1\n");
		exit(0);
		}
	exit(0);
}


void tache_A(void)
{


// A complétez




	}
gpioFree(PIN);
exit(0);}
```

Document réponse N°2

```
#include <vxWorks.h>
#include <unistd.h>
#include <stdio.h>
#include <timers.h>
#include "gpiolib.h"


void init(void);
unsigned char interrupteur(void);
void affichage(unsigned char c);
void start(void);


void swint(void)
{
	TASK_ID tid_a, tid_b;



	 tid_a=taskSpawn("tachea",101,0,5000,(FUNCPTR)start,0,0,0,0,0,0,0,0,0,0);
	if(tid_a==ERROR)
		{
		perror("pb taskspawn 1\n");
		exit(0);
		}

	taskSuspend(0);


}







void start(void) {
	char c=0;
	int d=0;


	init_GPIO();
	while(1){
		printf("tapez y pour continuer n pour sortie \n");
				fflush(stdin);

			scanf(" %c",&c);
			if (c=='n'){
				printf("sortie \n");
				exit(0);

			}

	printf("Programmez une valeur pour le port de sortie de 0 a 255 \n");

// A compléter
```

```
// A complétee




      }



}
```

Document réponse N°3

```
#include <vxWorks.h>
#include <unistd.h>
#include <stdio.h>
#include <timers.h>
#include "gpiolib.h"
int temoin=0;
void tache_A(void);


void swint(void)
{
      TASK_ID tid_a, tid_b;

      pinMode(PIN, IN);
      pinMode(PIN1, OUT);
      printf("switches 101\n\n Off");
        tid_a=taskSpawn("tache_A",101,0,5000,(FUNCPTR)tache_A,0,0,0,0,0,0,0,0,0,0);
      if(tid_a==ERROR)
            {
            perror("pb taskspawn 1\n");
            exit(0);
            }
}


void routineIT(void)
{
//affiche("interruption");

// A compléter

}

void tache_A(void)
{
       printf("tache A\n\n On");
//vxbGpioIntConfig (PIN, 1, 1);
vxbGpioIntConnect (_____________________________________________);
vxbGpioIntConfig (_____________________________________________);
vxbGpioIntEnable (_____________________________________________);
//vxbGpioIntConfig (PIN, 1, 1);
```

```
        sleep(500);// endormissement 500s
vxbGpioIntDisable(PIN,(VOIDFUNCPTR)routineIT1, 0);
vxbGpioIntDisconnect(PIN,(VOIDFUNCPTR)routineIT1, 0);
gpioFree(PIN);
//exit(0);
}
```

Document annexe N°1

```
#include <vxWorks.h>
#include <unistd.h>
#include <stdio.h>
#include <timers.h>
#include <kernelLib.h>
#define FOREVER    for (;;)
#define OUT 1
#define IN 0

#define HIGH 1
#define LOW 0

int gpioWrite(UINT32 gpioNum, UINT32 value)
{
    return (vxbGpioSetValue(gpioNum, value));
}

int gpioRead(UINT32 gpioNum)
{
    return vxbGpioGetValue(gpioNum);
}

int pinMode(UINT32 gpioNum, UINT32 value)
{
    return vxbGpioSetDir(gpioNum, value);
}

int gpioFree(UINT32 gpioNum)
{
    return vxbGpioFree(gpioNum);
                                                    }
```

Document annexe N°2

```
void init_GPIO(void)
{
	pinMode(PIN11, OUT);
	pinMode(PIN8, OUT);
	pinMode(PIN7, OUT);
	pinMode(PIN5, OUT);
	pinMode(PIN6, OUT);
	pinMode(PIN12, OUT);
	pinMode(PIN13, OUT);
	pinMode(PIN19, OUT);

	pinMode(PIN18, IN);
	pinMode(PIN27, IN);
	pinMode(PIN22,IN);
	pinMode(PIN23, IN);
	pinMode(PIN24, IN);
	pinMode(PIN10, IN);
	pinMode(PIN9, IN);
	pinMode(PIN25, IN);

	pinMode(PIN4, OUT);
	pinMode(PIN17, IN);
}
```

Document annexe N°3

```
void affichage(unsigned char c)
{
if ((c&0x01)==0x01)
gpioWrite(PIN1, HIGH);
else
gpioWrite(PIN1, LOW);
if ((c&0x02)==0x02)
gpioWrite(PIN2, HIGH);
else
gpioWrite(PIN2, LOW);
if ((c&0x04)==0x04)
gpioWrite(PIN3, HIGH);
else
gpioWrite(PIN3, LOW);
```

```
if ((c&0x08)==0x08)
gpioWrite(PIN4, HIGH);
else
gpioWrite(PIN4, LOW);
if ((c&0x10)==0x10)
gpioWrite(PIN5, HIGH);
else
gpioWrite(PIN5, LOW);
if ((c&0x20)==0x20)
gpioWrite(PIN6, HIGH);
else
gpioWrite(PIN6, LOW);
if ((c&0x40)==0x40)
gpioWrite(PIN7, HIGH);
else
gpioWrite(PIN7, LOW);
if ((c&0x80)==0x80)
gpioWrite(PIN8, HIGH);
else
gpioWrite(PIN8, LOW);}
```

Document annexe N°4

```
gpio: gpio@fe200000
      {
      compatible = "brcm,bcm2711-gpio";
     gpio-controller;
      #gpio-cells = <4>;
      reg = <0x0 0xfe200000 0x0 0x00f4>;
      interrupts = <145 0 1
                    146 0 1>;
      interrupt-parent = <&intc>;
      pinmux-0 = <&gpio_pins0>;
      status = "okay";
      };
```