UNIVERSITÉ
CAEN
NORMANDIE

ESIX NORMANDIE CAEN
MÉCATRONIQUE ET SYSTÈMES NOMADE

COMPTE-RENDU
TP de Microcontrôleur

BEHILIL Mohamed-Yassine ; DEVAUX Benoît 04/05/2023

TP1 : Ordonnanceur

Questions :

1. La variable "TEST" est associée à la broche P1.0, c'est-à-dire que sa valeur est directement liée à l'état de cette broche.

2. En utilisant un scheduler qui fonctionne avec une périodicité de 10 ms, chaque tâche est appelée toutes les 10 ms. Si nous avons 10 tâches, cela signifie que chaque tâche est exécutée toutes les 100 ms. La périodicité d'appel de la tâche 1 est de 100 ms, car elle est appelée toutes les 10 périodes de scheduler, soit toutes les 100 ms.
   On peut pratiquement vérifier ce temps en mettant un point d'arrêt sur la fonction d'interruption. Cela permet de s'assurer que chaque tâche est effectivement appelée toutes les 100 ms.

3. TMR3RL est un registre de 16 bits qui utilise en conjonction les registres TMR3RLL et TMR3RLH pour former un compteur de 16 bits qui compte le nombre de cycles d'horloge depuis le dernier overflow du Timer3. Lorsque ce compteur atteint sa valeur maximale de 65535, il se remet automatiquement à zéro et génère une interruption (si cette fonctionnalité est activée).
   Pour un microcontrôleur fonctionnant avec une fréquence d'horloge de 10 MHz et une période de timer désirée de 10 ms, il faut calculer la valeur initiale du timer en utilisant la formule suivante :
   On a val initial = (10 ms) x (10/12 MHz) = 8333
   Ainsi : TMR3RL = [65536 (= FFFFH+1 [overflow])] - 8333
   TMR3RL = DF73 Hex
   Enfin TMR3RLH = 0xDF Hex
   Et TMR3RLL = 0x73 Hex
   Sans oublier à programmer Time Overflow Frequency à 1/(10ms) qui est **100 Hz**

4. L'adresse de la première instruction exécutée du programme d'interruption du Timer 3 est le vecteur d'interruption du timer 3 se trouvant à l'adresse 0x73. “”C'est une LJMP vers TIMER3_ISR.””
   La dernière instruction assembleur de la fonction TIMER3_ISR() est RETI (Return Interrupt).

5. TIMERS ISR :

ANL 91H, #7FH ; ⇔ TMR3CN&=~0x80;

/* En utilisant l'opérateur AND (ANL) avec la valeur complémentaire de 0x80 (qui est 0x7F), l'instruction masque tous les bits du registre TMR3CN sauf le bit 8 (qui correspond à TF3H qui est mis à zéro). Cela efface le flag d'interruption du overflow et permet au Timer 3 de commencer à compter à nouveau. */

SETB 0H; ⇔ new_task=1;

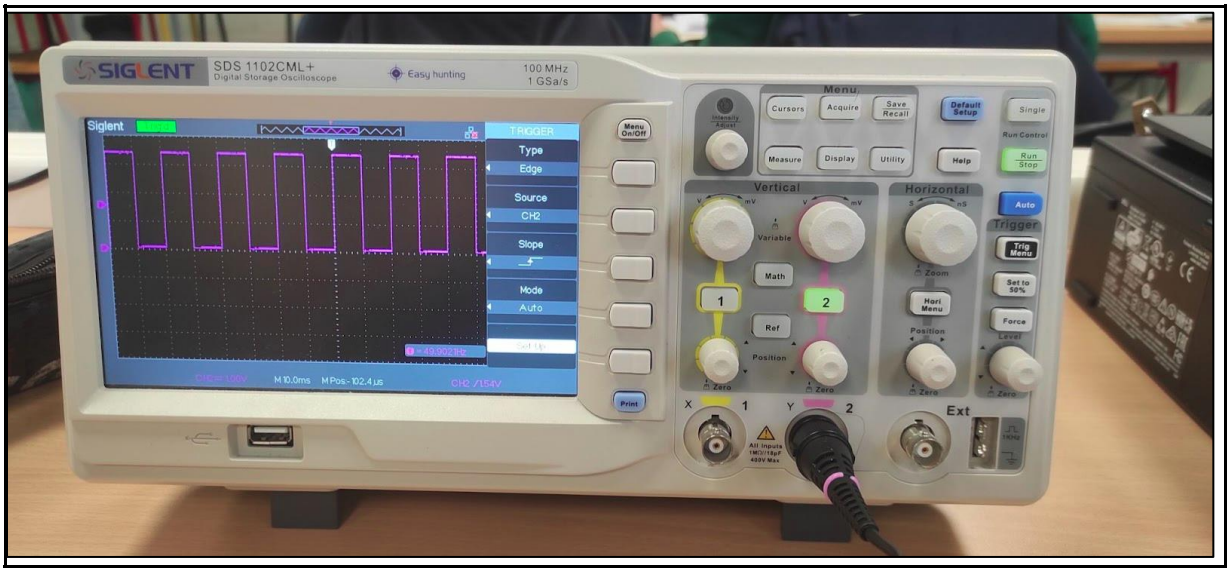
/* La ligne "SETB 0H" permet de mettre la variable **new_task à 1**, indiquant qu'une nouvelle tâche est prête à être exécutée par le gestionnaire de tâches. Cela permet de sortir de la boucle

while(new_task == 0) qui attend l'activation du gestionnaire de tâches.*/

// La variable **new_task** est une variable globale qui est stockée dans la banque de registres 0.

CPL P1.0;⇔ TEST!=TEST;

/* Lorsque la fonction d'interruption est appelée toutes les 10ms, elle change la valeur de la variable "TEST", qui est ensuite utilisée pour inverser l'état de la broche P1.0 avec l'instruction "CPL P1.0". Ainsi, si la broche P1.0 est connectée à un oscilloscope, on peut observer cette inversion toutes les 10ms. Cela permet de confirmer que la fonction d'interruption est bien appelée à la fréquence attendue et que la base de temps fonctionne correctement.



TP2 : Ordonnanceur avec Watchdog

Question 1 :

Étape 1 : Modifier votre ordonnanceur de manière à ce que le watchdog ne provoque pas de reset.

Il est impératif de relancer le watchdog dans la fonction scheduler() en mettant PCA0CPH5 à zéro, car cela nous permet de contrôler son redémarrage après chaque tâche. Autrement, il se relancerait de manière aléatoire après que les 11ms se soient écoulées.

Étape 2 :

Lors du 50ème appel du cycle 7 de l'ordonnanceur, nous avons décidé de créer un problème en ajoutant une attente de 12 ms. Cependant, cette durée est supérieure à la période de notre watchdog, ce qui va entraîner un reset de la machine.

Notez que nous pouvons identifier quelle tâche a provoqué le reset en utilisant un point d'arrêt sur NOP(), qui sera exécuté si nous avons un reset.

Pour identifier quelle tâche a provoqué un reset, il est nécessaire de stocker cette information dans une zone de mémoire qui ne sera pas effacée lors d'un reset. Comme la RAM interne du microcontrôleur est effacée à chaque interruption, il est important de stocker cette variable dans une zone de mémoire non volatile, comme la XRAM.

Ainsi **scheduler_ct** sera déclaré comme variable globale **extern** dans le fichier d'en-tête "**variable_globale.h**" et de la définir dans le fichier source "**variable_globale.c**" .

Question 2 :

Pour la création d'une attente de 12 ms lors du 50ème appel du cycle 7 de l'ordonnanceur:

Solution :

```
void scheduler(void){
    static uint8_t nb_appels=0;
    uint8_t i;

    PCA0CPH5 = 0;
    switch(scheduler_ct) {
        case 0: {
            delay(5);
            break;
        }
        case 1: {
            delay(5);
            break;
                }
        case 7: {

            nb_appels++;
            if (nb_appels==50){
                for(i=0;i<100;i++){
                    delay(120); //attente de 12 ms de""l'etape 2""
                }
            }
            break;
        }
    }
```

Figure 2 : Code pour une attente de 12ms

La variable nb_appels est déclarée avec le spécificateur de classe static pour une durée de vie globale car la variable conserve sa valeur entre les appels successifs de la fonction **scheduler()**.
Problème détecter à :

(x)= scheduler_ct	unsigned char	7 ('\007')	XRAM:0x0

Preuve :

```
    if ((RSTSRC &0x02)==0x00){
        if ((RSTSRC & 0x08)==0x08){
            NOP(); //point d'arret pour detecter un reset provoque par le watchdog
        }
    }
```

Question 3 :

Est-il possible d'effectuer l'attente de 12 ms en appelant une seule fois la fonction **delay()** avec un paramètre égal à 12000 ?

L'input de la fonction **delay()** est un caractère "**char**" de 8 bits, qui peut prendre des valeurs allant de 0 à 255, alors il ne serait pas possible d'effectuer l'attente de 12 ms en appelant une seule fois la fonction delay() avec un paramètre égal à 12000.
il faut absolument passer par une boucle.

TP3 : Scrutation d'un périphérique (clavier) dans l'ordonnanceur

Étape 1 : Vérifier le fonctionnement de la fonction *getchar()*.

Après l'ajout de l'initialisation, il a été observé que la variable "key" récupérait correctement la valeur saisie au clavier une fois que la touche était enfoncéea clavier.

Étape 2 : L'objectif est de scanner le clavier et stocker la frappe d'une touche dans la variable key_memory une fois relâchée.
Pour ce faire on ajoute le code suivant:

```
void scheduler(void){
    uint8_t key;
    static uint8_t key_memory=0;
    uint8_t tmp ;

    PCA0CPH5=0;

    switch(scheduler_ct) {
        case 0: {

                key = getchar();

                if(key != 0){
                    tmp = key;
                }

                if(key == 0){
                    key_memory = tmp ;
                    tmp=0;
                }

                if(key_memory != 0){
                    key_memory = 0;
                    NOP();
                }

                break;
            }

        }
```

Nous avons déclaré une nouvelle variable de type uint8_t appelé *tmp* qui permettra de détecter quand l'utilisateur aura relâché la touche. Il faut attendre d'avoir un 0 saisi au clavier pour pouvoir rentrer dans la deuxième condition. Cette dernière récupère dans la variable *key_memory* la valeur de notre nouvelle variable.

Question: Quand l'application fonctionnera correctement autorisez le watchdog (période 11ms) et vérifiez qu'aucun reset n'est provoqué par son overflow.

Nous avons placé un point d'arrêt à l'endroit indiqué, mais aucun reset n'a été détecté.

```
if((RSTSRC & 0X02) == 0x00 ){
    if((RSTSRC & 0X08) == 0x08)
        NOP(); // point d'arret pour detecter un reset provoqué par le Watchdo
}
```

un point d'arrêt pour détécter un reset provoqué par le Watchdog

TP4 : Décomposition d'une tâche en sous-process

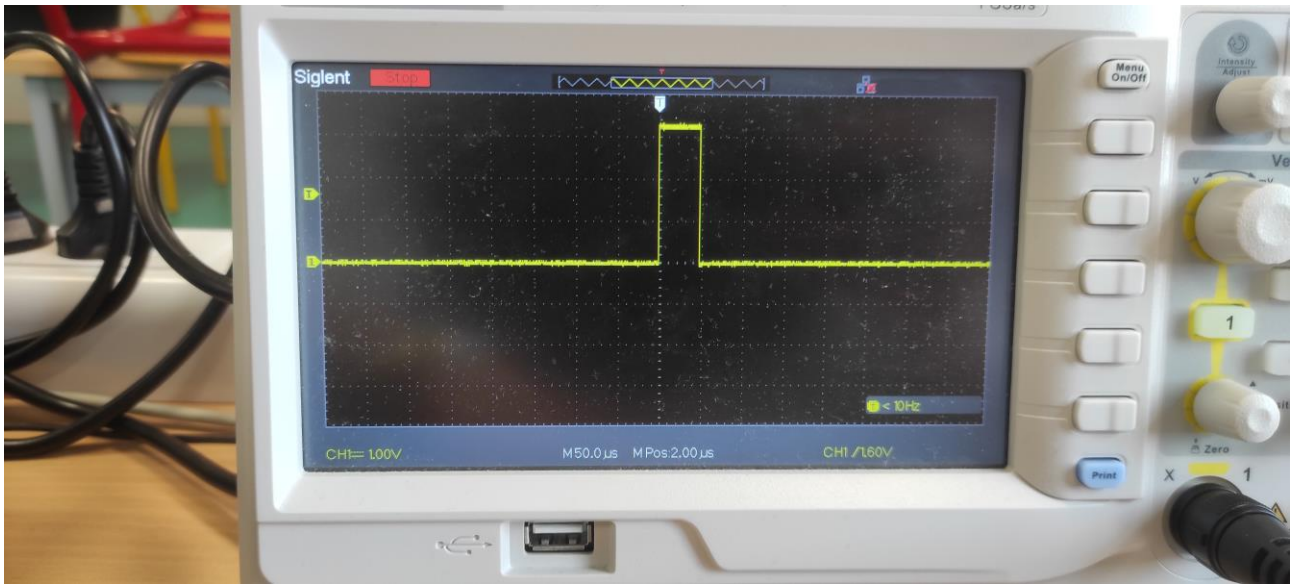
Étape 1 :

Les deux variables *flag_affi_init* et *flag_affi_1* sont à déclarer en static pour les conserver en dehors du scheduler et ainsi qu'elles aient une durée de vie globale.

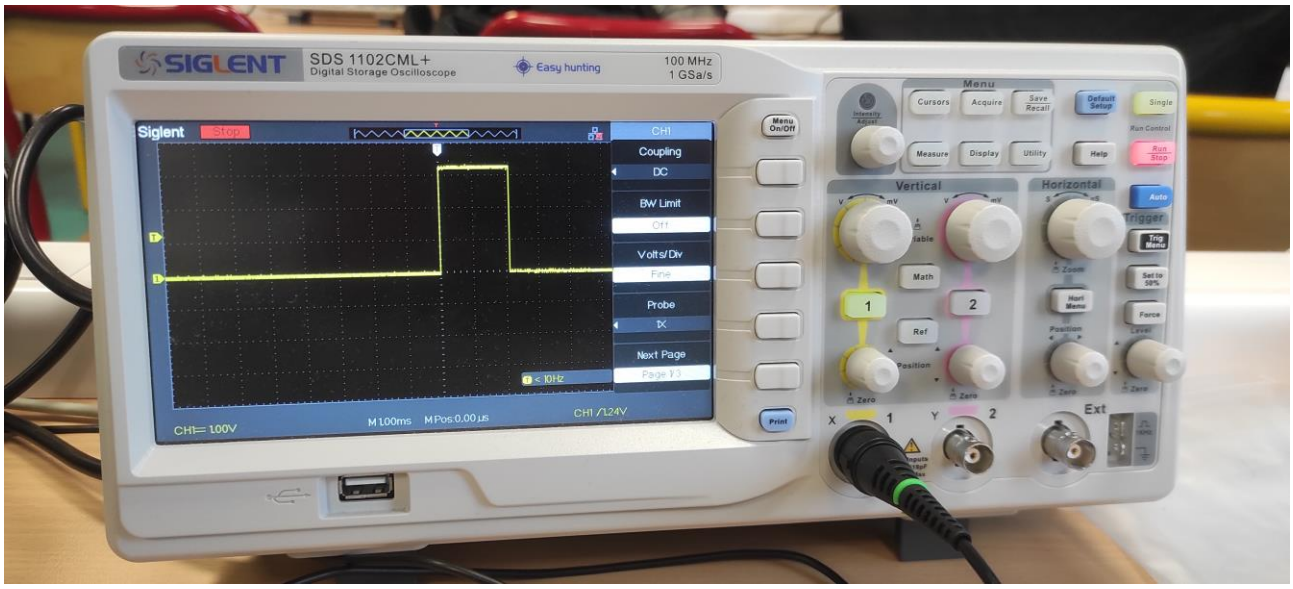
Questions :

Mesurer la durée de la fonction putchar() et le temps d'affichage de deux lignes de caractères (rafraîchissement de l'écran). Est-ce compatible avec votre ordonnanceur ?

la durée de la fonction putchar() est 50 microseconde comme suit:



Le temps d'affichage de deux lignes de caractères est 2,1ms :



Cela est cohérent car en multipliant le nombre de caractères par ligne (17) par le temps de transmission d'un caractère (50 microsecondes), on obtient 0,85 millisecondes, soit 1,7 millisecondes pour les deux premières lignes. Cependant, en ajoutant une troisième ligne de 8 caractères avec la commande printf("\nBonjour"), on obtient un temps total de 2,1 ms, ce qui correspond exactement au résultat précédent.

Est-ce compatible avec votre ordonnanceur ? Oui,car inférieure a 10ms.

Etape 2 :

Modifier le programme pour que la touche '*' provoque l'affichage d'un message et que la touche '#' efface la totalité de l'écran.

```
static unsigned char flag_affi_init = 0;
static bit flag_affi_1 = 0;
static bit flag_clr = 0;
```

Et

```
if(key_memory == '*'){
    key_memory = 0;
    flag_affi_1 = 1;
}
if(key_memory == '#'){
    key_memory = 0;
    flag_clr = 1;
}
```

Question :

Expliquer le fonctionnement de la fonction clrscr_LCD().

La fonction "clrscr_LCD()" permet de supprimer tout ce qui est affiché sur l'écran LCD. Elle utilise la commande "sendByte_LCD()" pour envoyer le code 0x1, qui correspond à l'effacement de l'écran selon la documentation , lorsque AFFICHEUR_RS est mis à 0 pour sélectionner la commande. Ensuite, elle utilise une boucle "for" pour générer un délai de 1,64 millisecondes, qui correspond au temps nécessaire pour que la commande soit exécutée. Enfin, remettre AFFICHEUR_RS = 1 pour revenir en mode affichage.

Etape 3 :Autoriser le watchdog avec une période de 11ms.

Modifier le programme pour éviter le reset provoqué par le watchdog en scindant l'étape problématique en deux, et compléter la couche logicielle LCD avec une fonction de mise sous tension pour assurer le bon fonctionnement du programme avec le watchdog.

la gestion de l'afficheur est confiée à une étape de l'ordonnanceur (cycle 1) qui est scindée en 2 processus :

●Mise sous tension et initialisation de l'afficheur.

●Affichage des messages.

→ un seul processus est exécuté à chaque passage dans le cycle.

→ le premier processus est effectué une seule fois après le reset.

Nous allons ajouter le programme :

```
case 1 :{
    if(flag_affi_init == 0){
        poweron_LCD();
        //init_LCD();
        flag_affi_init = 1;
    }
    else if(flag_affi_init == 1){
        init_LCD();
        flag_affi_init = 2;
    }
    else{
        //init_LCD();
        if(flag_affi_1 == 1){
            flag_affi_1 = 0;
            TEST = 1;
            printf("\r0123456789ABCDEF");
            printf("\n0123456789ABCDEF");
            printf("\nBonjour");
            TEST = 0;
        }
        if(flag_clr == 1){
            clrscr_LCD();
        }
        if(flag_affichrono == 1){
            flag_affichrono=0;
            printf("\rmin=%02bd s=%02bd ", min,sec);
        }
    }
break;}
```

```
void poweron_LCD(void){
    AFFICHEUR_ON;
    AFFICHEUR_RS=0;
}
```

Étape 4 : Réaliser un chronomètre qui affiche les minutes et les secondes et qui démarre après que la touche ‘1’ ait été enfoncée.

On note les variables que nous utilisons dans le code ci-dessous :

static bit flag_startchrono = 0;

static bit flag_affichrono = 0;

unsigned char min,sec;

static uint8_t nb_appels;

```
case 3 :{
    if(flag_startchrono == 1){
        nb_appels++;
        if(nb_appels == 10){
            nb_appels = 0;

            if(sec < 60){
                sec++;
            }
            if(sec == 60){
                min++;
                sec = 0;
            }
            flag_affichrono=1;
        }
    }
break;
}
```

L'incrémentation des minutes et des secondes se fait dès lors que le nombre d'appels atteint 10 pour avoir 1 s.

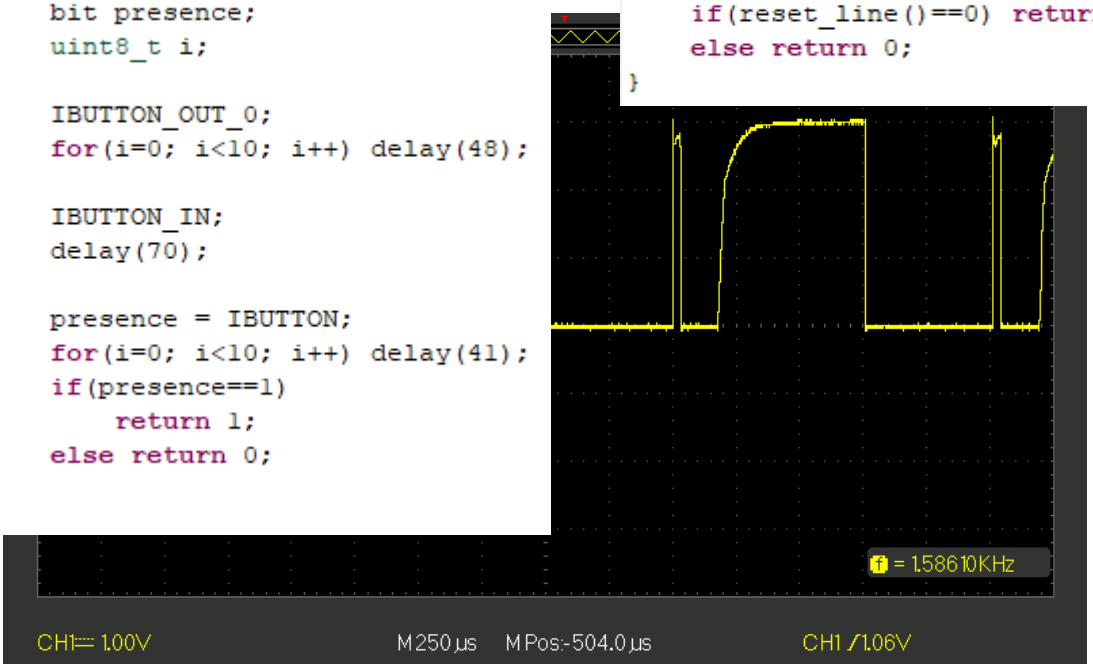
TP5 : Ibutton

Etape 1 : Programmer la fonction reset_line() dans le fichier one_wire_bus.c. Puis compléter la fonction reset_IBUTTON() en utilisant

reset_line().

```
bit reset_line(void){
    bit presence;
    uint8_t i;

    IBUTTON_OUT_0;
    for(i=0; i<10; i++) delay(48);

    IBUTTON_IN;
    delay(70);

    presence = IBUTTON;
    for(i=0; i<10; i++) delay(41);
    if(presence==1)
        return 1;
    else return 0;
}
```

```
bit reset_IBUTTON(void) {
    if(reset_line()==0) return 1;
    else return 0;
}
```



Etape 2 :

Ecrire la fonction lecture_ID() qui lit l'identificateur (ID) du ibutton connecté. Cette fonction sera

appelée dans la tâche 9 de l'ordonnanceur. Dans Simplicity Studio, relever l'ID du Ibutton lu par le

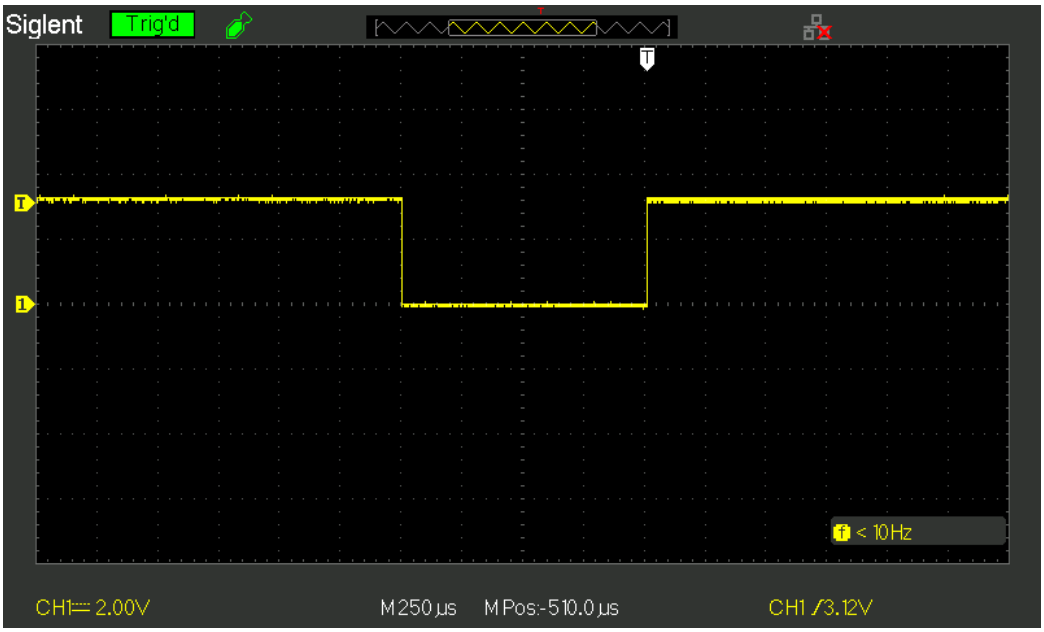
microcontrôleur et comparer le avec celui inscrit sur le composant.

Questions : Quel est la durée de la fonction lecture_ID() ? Conclusion.

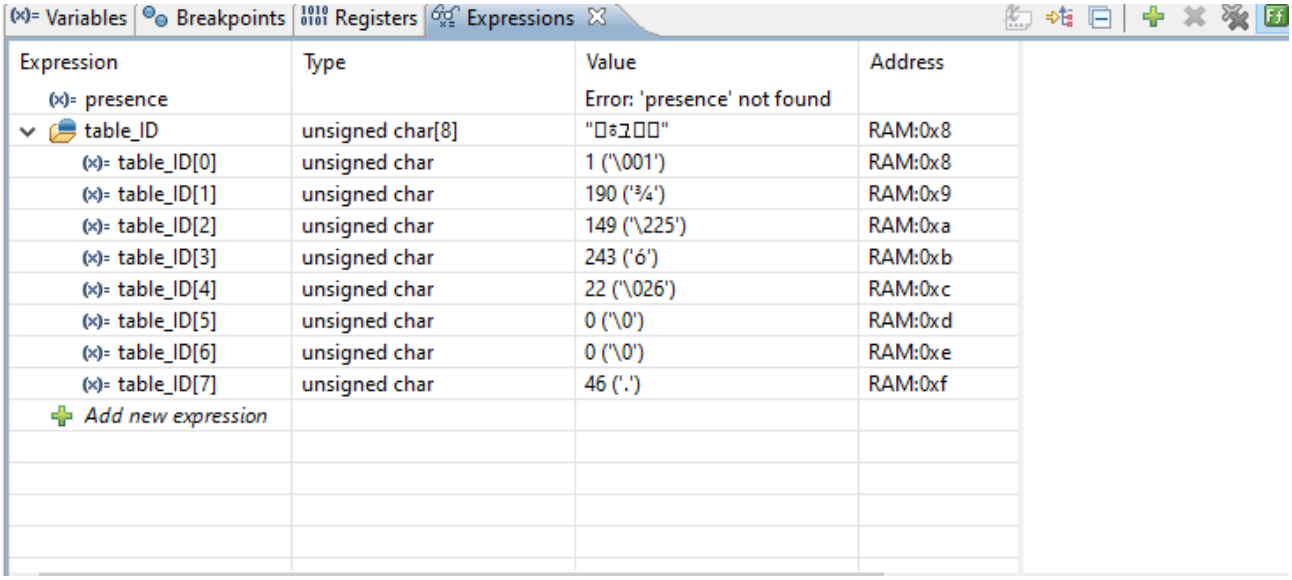
```
bit lecture_ID(uint8_t table_IBUTTON[8]){
    uint8_t i;

    if(reset_IBUTTON()==1){
        write_Byte(0X33);

        for(i=0;i<8;i++){
            table_IBUTTON[i] = read_Byte();
        }

        if(test_CRC(table_IBUTTON)==1)  return 1;
        else    return 0;
    }
    else    return 0;

}
```

la durée de la fonction lecture_ID() : approximativement 1 ms

```
case 9 : {
          TEST = 1;
          lecture_ID(table_ID);
          TEST = 0;
          break;
      }
```



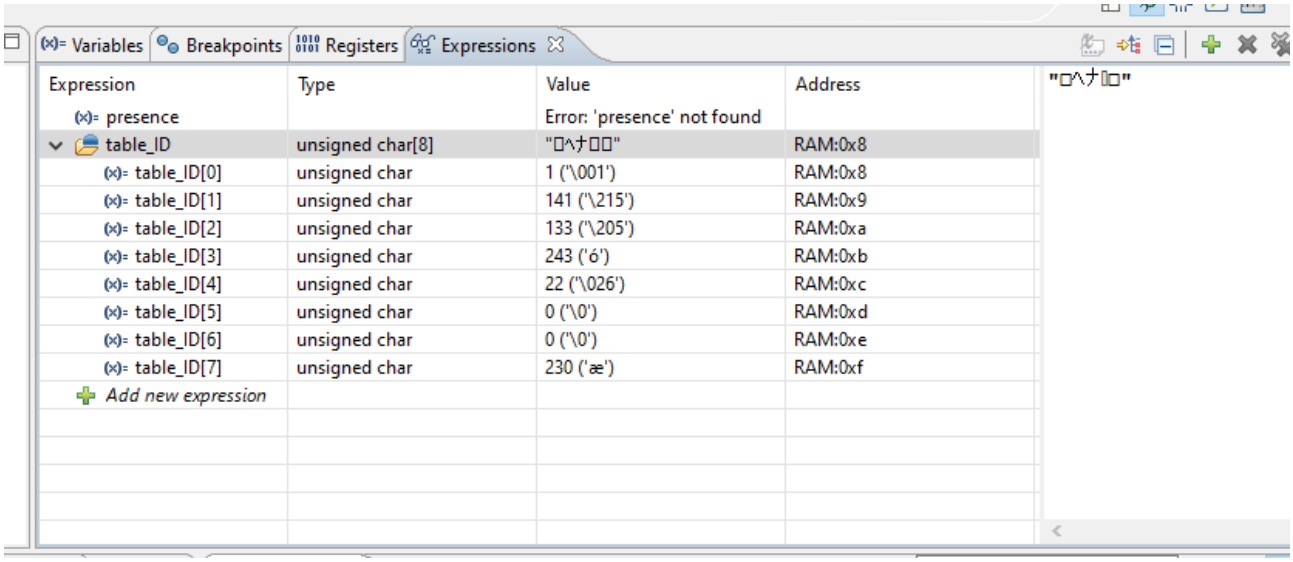
Relever l'ID du Ibutton lu par le microcontrôleur et comparer le avec celui inscrit sur le



Expression	Type	Value	Address
presence		Error: 'presence' not found	
table_ID	unsigned char[8]	"□ȣ□□"	RAM:0x8
table_ID[0]	unsigned char	1 ('\001')	RAM:0x8
table_ID[1]	unsigned char	190 ('¾')	RAM:0x9
table_ID[2]	unsigned char	149 ('\225')	RAM:0xa
table_ID[3]	unsigned char	243 ('ó')	RAM:0xb
table_ID[4]	unsigned char	22 ('\026')	RAM:0xc
table_ID[5]	unsigned char	0 ('\0')	RAM:0xd
table_ID[6]	unsigned char	0 ('\0')	RAM:0xe
table_ID[7]	unsigned char	46 ('.')	RAM:0xf
Add new expression			

composant:

Ce qui est identique au boutton dans on dispose :



Expression	Type	Value	Address
presence		Error: 'presence' not found	
table_ID	unsigned char[8]	"□ヘナ□□"	RAM:0x8
table_ID[0]	unsigned char	1 ('\001')	RAM:0x8
table_ID[1]	unsigned char	141 ('\215')	RAM:0x9
table_ID[2]	unsigned char	133 ('\205')	RAM:0xa
table_ID[3]	unsigned char	243 ('ó')	RAM:0xb
table_ID[4]	unsigned char	22 ('\026')	RAM:0xc
table_ID[5]	unsigned char	0 ('\0')	RAM:0xd
table_ID[6]	unsigned char	0 ('\0')	RAM:0xe
table_ID[7]	unsigned char	230 ('æ')	RAM:0xf
Add new expression			

Tp 6 : UART

Implémenter les paramètres d'initialisation suivants :

Activer l'UART0 dans péripheral Mapping et configurer les broches Rx et TX

Configurer l'UART0.

Configurer le timer 1 pour que l'UART0 fonctionne à 9600 bauds

Timer 1 Mode 2: 8-bit Counter/Timer	
Clock Source Frequency	2.500 MHz
Clock Source Period	400.000 ns
Target Overflow Frequency	19200 (0x4B00)
Timer Init Overflow After	102.400 us
Timer Init Value	0 (0x0)
Timer or Counter	Timer mode
Timer Reload Overflow Frequency	19.231 kHz
Timer Reload Overflow Period	52.000 us
Timer Reload Value	126 (0x7E)
Timer 1 Overflow for Peripherals	
SMBus0 SCK Frequency	6.410 kHz
UART0 Baud Rate	9.615 kHz

pour etre le plus proche de la valeur voulue on change la fréquence d'incrementation du Timer 1 à Sysclock/4

Properties of TIMER Setup

TIMER Setup

Property	Value
Clock Control	
Timer 0/1 Prescale	SYSCLK / 4
Timer 0	
Clock Frequency	2.500 MHz
Clock Source	Use the SCA prescale clock
Interrupt 0 Type Select	Level-triggered
Mode	Mode 0, 13-bit Counter/Timer
Timer or Counter	Timer mode
Timer Running State	Stopped
Timer Switch 1: Run Control	Stop
Timer Switch 2: Gate Control	Disabled
Timer 0 Firmware Control	
Timer Interrupt Cleared by	By Hardware
Timer Interrupt Enable Flag	ET0(IE.1)
Timer Interrupt Pending Flag	TF0(TCON.5)
Timer Overflow Flag	TF0(TCON.5)
Timer Start or Stop	TR0(TCON.4)
Timer 1	
Clock Frequency	2.500 MHz
Clock Source	Use the SCA prescale clock
Interrupt 1 Type Select	Level-triggered
Mode	Mode 2, 8-bit Counter/Timer with Auto-Reload
Timer or Counter	Timer mode
Timer Running State	Timer is Running
Timer Switch 1: Run Control	Start
Timer Switch 2: Gate Control	Disabled

UART 0

Property	Value
Stop Bit Transmitting	Logic 1

Properties of P0.4

P0

Property	Value
Settings	
Drive Strength	Low drive
IOMode	Digital Push-Pull Output
Latch	High
Mask Value	Ignored
Match Value	High
Skip	Not skipped

Pour la configuration des broches P0.4 en push_pull car sortie et P0.5 en open-drain car entrer:

Properties of P0.5

P0

Property	Value
Settings	
Drive Strength	Low drive
IOMode	Digital OpenDrain I/O
Latch	High
Mask Value	Ignored
Match Value	High
Skip	Not skipped

Etape 1:

Le programme fourni envoie sur la liaison UART0 le code 0x33 a chaque fois que la touche '4' du

clavier matriciel est appuyée.

Vérifier avec un point d'arrêt qu'une interruption marquant la fin de le transmission est réalisée.

```
if(key_memory == '4'){
    key_memory = 0;
    SBUF0 = 0x33; //SCON0_TI = 1;
```

```
    if(SCON0_TI==1){
        SCON0_TI=0;

    }
```

Questions :

1) Relever avec l'oscilloscope la trame de transmission (Tx sur P0^4). Faut-il mieux configurer Tx

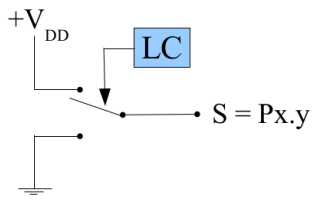
en push-pull ou en open-drain ? Pourquoi ?

2) Interpréter chaque bit de la trame. En déduire la vitesse de communication.

Réponse :

1) il mieux configurer Tx en push-pull car sortie

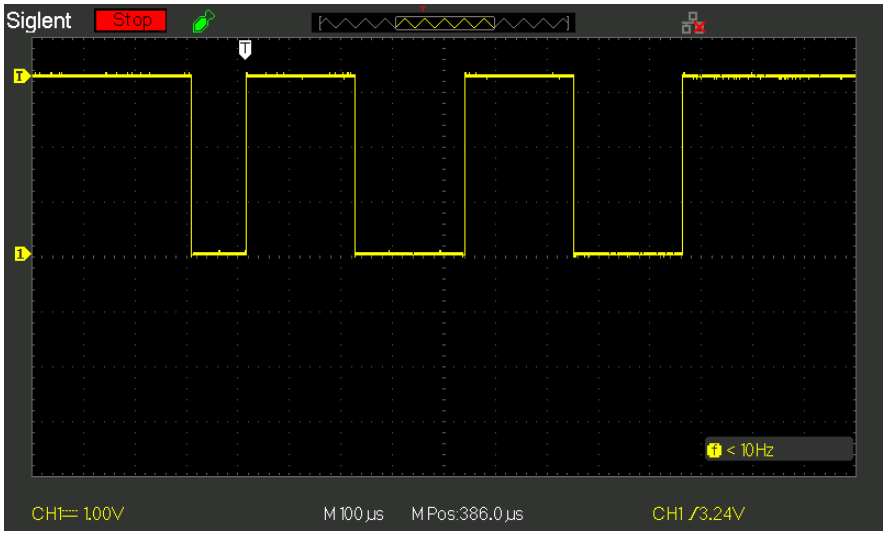
PxMDOUT . y = 1 : Px.y en push-pull



Utilisation : sortie
Avantages : vitesse & courant

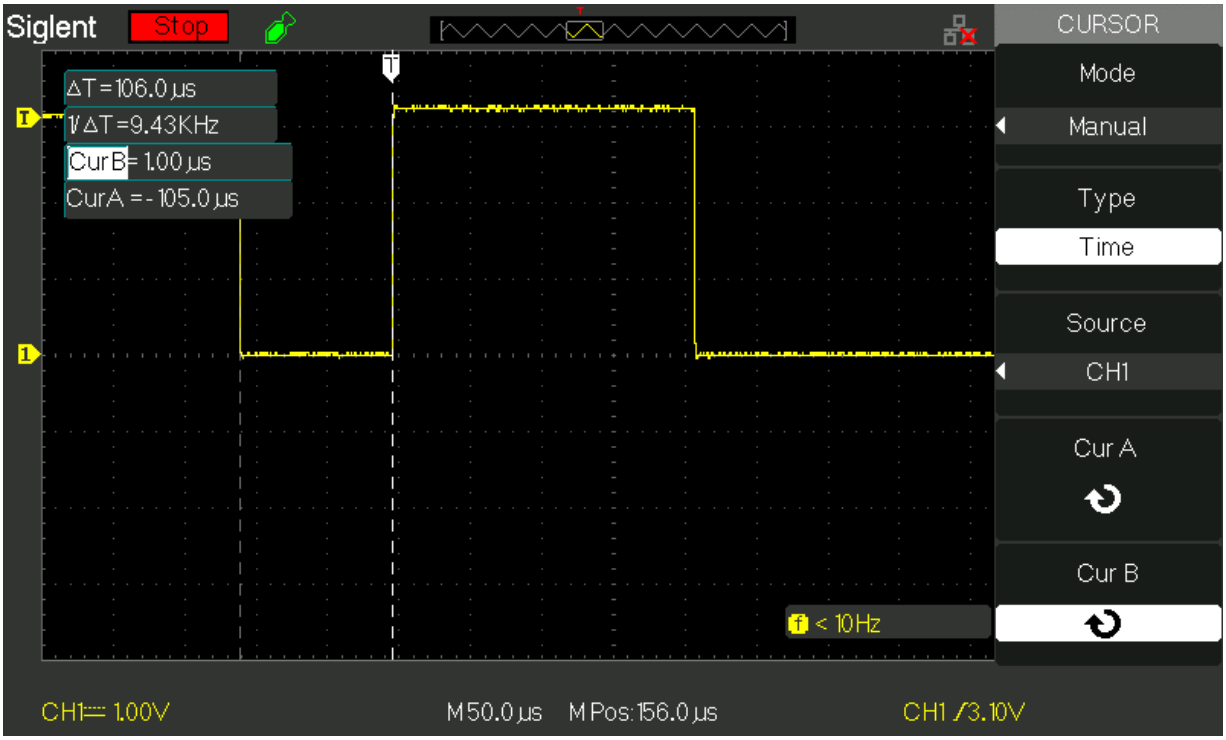
Inconvénient : ne pas relier des sorties entre-elles sinon possibilité de c-c alimentation-masse.

Ainsi que l'oscilloscope la trame de transmission:



2) Interprétation chaque bit de la trame

donne 11001100 et a l'inverse c'est 00110011 sui est 0x33 en hexadécimal.



La durée d'un bit est de 104 microsecondes (en réalité 106 microsecondes sur l'oscilloscope en raison du curseur, mais en pratique c'est bien 104 microsecondes), ce qui donne une vitesse de communication de **9615 bauds, qui correspond exactement à la configuration choisie.**

Etape 2 :

Remplacer dans la tâche 0 l'écriture dans SBUF0 par un appel soft au programme d'interruption

(passage de SCON0_TI à '1'). Modifier la routine UART0_ISR pour transmettre un seul et unique

caractère à chaque fois que la touche '4' du clavier est frappée.

Solution pour ne pas rester dans une boucle car la source du problème est l'interruption déclarée par SBUF0 est :

```
if(key_memory == '4'){
    key_memory = 0;
    SCON0_TI = 1; //SBUF0 = 0x33
```

et on change le programme d'interruption en ajoutant un compteur.

```
INTERRUPT(UART0_ISR, UART0_IRQn){
    static uint8_t nb_entrer_inter;

    if(SCON0_TI==1){
        SCON0_TI=0;
        if(nb_entrer_inter == 0){
            nb_entrer_inter++;
            SBUF0 = 0x33;
        }
    }
    else if(SCON0_RI==1){
        SCON0_RI=0;
        }
    }
```

Etape 5 : Quand le PC envoie le code ASCII de la lettre M, le microcontrôleur transmet au PC le

contenu d'un tableau de caractères (Bytes_ToSend). Ce tableau sera remplit par la fonction sprintf()

avec les codes ASCII d'un message laissé à votre discrétion mais dont la taille n'est pas connue à

priori. Pour transmettre l'intégralité du message, il sera judicieux de détecter la fin du tableau

marquée par le caractère '\0' qui est ajouté par la fonction sprintf().

Pour ce faire:

```
INTERRUPT(UART0_ISR, UART0_IRQn){
    static uint8_t nb_entrer_inter;
    static unsigned char recue;
    uint8_t Bytes_ToSend[10];
    static uint8_t len=0;

    if(SCON0_TI==1){

        if(nb_entrer_inter <= len){
            nb_entrer_inter++;
            SBUF0 = Bytes_ToSend[nb_entrer_inter-1];
        }
        SCON0_TI=0;
    }
    else if(SCON0_RI==1){
        SCON0_RI=0;
        recue = SBUF0;
        if(recue == 'M'){
            sprintf(Bytes_ToSend,"test\n");

            while(Bytes_ToSend[len] != '\0'){
                        len++;
            }
            SCON0_TI = 1;
        }
    }
}
```

TP 7 : ADC 10 bits – Voltmètre

Implémenter les paramètres d'initialisation suivants :

Ports : valider le Crossbar et configurer les I/O (P1.0 en push-pull et P0^6 en analogique)

Configurer l'ADC0.

Configurer le timer 2 avec une base de temps de 18ms.

pour ce faire:

Dans cet exercice l'ADC0 du microcontrôleur est configuré de la manière suivante :

• Conversion sur 10 bits de la tension sur la broche P0^6.

• Horloge de conversion SARclk de 3.33MHz (pas de Burst ni de Delay Tracking Mode).

• Tension de référence Vref de 1.65V.

• Gain de l'atténuateur égal à 0.5.

• Conversion lancée par le timer 2 toutes les 18ms sans accumulation des mesures.

Properties of ADC 0

ADC 0

Property	Value
View	
View	Advanced
Control	
Enable ADC	Enabled
Enable Burst Mode	Disabled
Start of Conversion	Timer 2 overflow
Timer 2 Overflow Frequency	55.000 Hz
Multiplexer Selection	
Positive Input Selection	Ground
Configuration	
SARCLK (Actual)	3.333 MHz
SAR Clock Source	10.000 MHz
SAR Clock Divider	3 (0x3)
Enable 8-Bit Mode	Normal mode (10-bit)
Gain Control	0.5x gain
Result Shift and Justify	Right justified
Track Mode	Normal tracking
Throughput	
Conversion Time	3.900 us
Minimum Tracking Time	1.500 us
Maximum Throughput	185.172 ksps
Burst Mode Control	
Burst Mode Repeat Count	1 conversion
Accumulator Configuration	
Enable Accumulate	Disabled
Window Compare	
Greater-Than Value	0 (0x0)
Less-Than Value	0 (0x0)

Configuration Timer 2:

TIMER 2

Property	Value
Control	
Clock Source	SYSCLK / 12
Clock Source Frequency	833.333 kHz
Clock Source Period	1.200 us
Enable Low Byte Interrupt	Disabled
Mode	One timer, 16-bit auto-reload
Run Control	Start
Timer Running State	Timer is Running
Init and Reload Value	
Target Overflow Frequency	56 (0x38)
Timer Init Overflow After	18.000 ms
Timer Init Value	50536 (0xC568)
Timer Reload Overflow Frequency	56.000 Hz
Timer Reload Overflow Frequency(Low Byte)	25.253 kHz
Timer Reload Overflow Period	17.857 ms
Timer Reload Overflow Period(Low Byte)	39.600 us
Timer Reload Value	50655 (0xC5DF)
Timer Overflow for Peripherials Clock Source	
ADC0 Conversion Start Signal	56.000 Hz
SMBus0 SCK Frequency	18.667 Hz

La touche '0' doit lancer le début des mesures:

```
if(key_memory == '0'){
    key_memory = 0;
    TMR2CN_TR2 = 1;
}
```