

# MICROCONTRÔLEURS MICROCONTRÔLEURS ET SYSTÈMES NUMÉRIQUES TRAVAIL PRATIQUE NO 2

MT GROUPE A	MT Groupe B	EL		
No du Groupe	Premier Etudiant	Second Etudiant	Evaluation	Visa Correcteur
093	Nathann Morand	Felipe Ramirez		

## 2. LA CARTE STK-300, LES PORTS D'ENTRÉE/SORTIE, LES BOUCLES D'ATTENTE ET LA GÉNÉRATION DE SON

Ce travail pratique présente une introduction au travail au moyen de Atmel Studio 7.0 et de la carte STK-300 et ses ports. Les boucles d'attente sont étudiées sur l'exemple d'un périphérique émettant du son.

Veuillez vous connecter sur les PCs avec votre compte EPFL habituel, puis créer un répertoire tp02 dans lequel vous créerez les projets et placerez vos fichiers. Ce répertoire est local à votre PC afin de garantir une rapidité d'accès aux fichiers maximale. A la fin de la session, veuillez copier ce répertoire dans votre partition EPFL personnelle pour sauvegarde (my documents), et enlever tous les fichiers présents sur le PC local.

### 2.1 LECTURE DES INTERRUPEURS, ET AFFICHAGE SUR LES LEDs

Créez un nouveau projet dans le répertoire tp02, en suivant la méthode appliquée au TP01. Chargez le programme inout1.asm en Figure 2.2 en cliquant par le bouton de droite de la souris ("SBD") sur AssemblerApplication1 qui apparaît dans la fenêtre "Solution Explorer" située sur la droite, puis dans le menu déroulant qui apparaît, sélectionnez Add→Existing Item ... .

Les fichiers nécessaires aux travaux pratiques sont disponibles sur le serveur \\stiiitcgen0\cours\Micro contrôleur, dans le répertoire "Micro contrôleurs" MICRO210-EE208→TP02 et/ou sur le site Moodle du cours.

Rendez le fichier actif en cliquant (SBD) sur le fichier dans le Solution Explorer, puis en choisissant Set as Entry File dans le menu déroulant. Double-cliquez le fichier afin de le faire apparaître dans une fenêtre d'édition.

Effectuez une simulation pas à pas et observez l'évolution des valeurs stockées dans les registres.

Ouvrez la fenêtre permettant d'observer les registres d'entrées/sorties par Debug→Window→I/O. Observez les valeurs des registres associés aux port B et port D. Cliquez sur I/O Port (Port B) (connecté aux LEDs) et I/O Port (Port D) (connecté aux bouton-poussoirs) pour faire apparaître les trois registres associés à un port d'entrée/sortie (Figure 2.3).

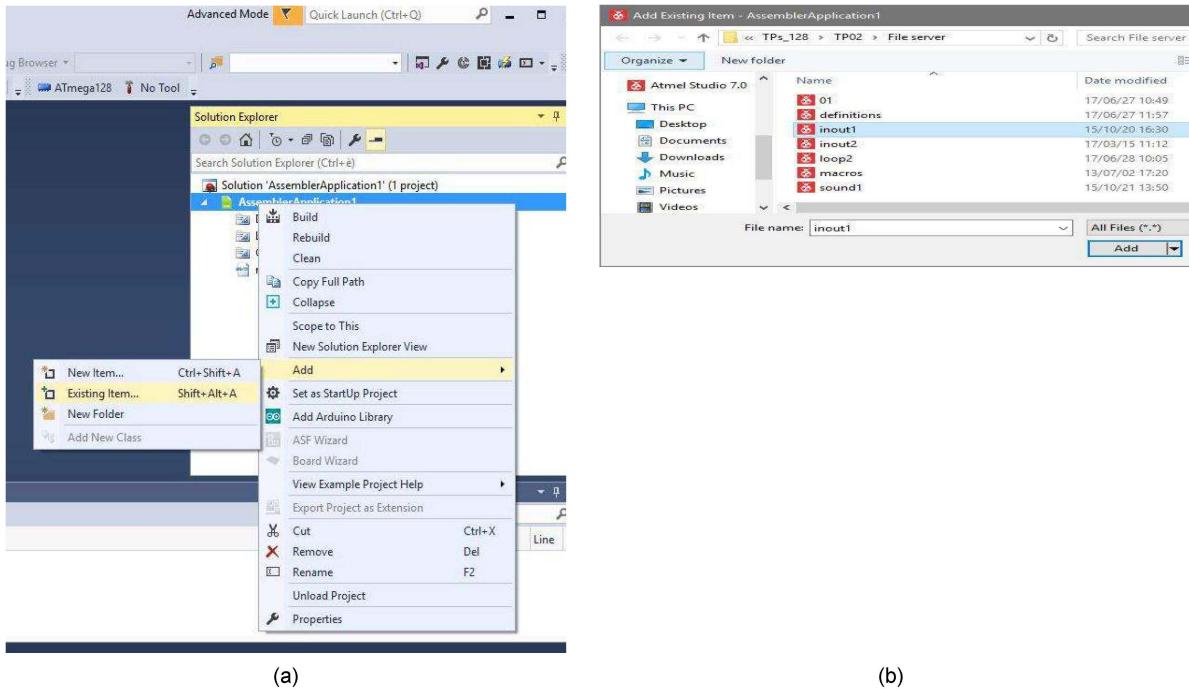


Figure 2.1: Ajout d'un fichier existant à un projet. (a) Selection depuis le Solution Explorer, et (b) choix du fichier.

---

```

; file inout1.asm
.include "m128def.inc"

reset:
    ldi r16,0xff; configure portB as output
    out DDRB,r16
    ldi r16,0x00; configure portD as input
    out DDRD,r16
loop:
    in r16,PINB; read input buttons
    out PORTB,r16; output result to LEDs
    rjmp loop

```

---

Figure 2.2: inout1.asm.

Ces registres ont les fonctions suivantes:

- PORTB: est logé à l'adresse **0x18**; que contient PORTB si le port est configuré en sortie ?  
**Il contient les donné qui sont "afficher" sur les pins de ce port si celle-ci sont en sortie.**
- DDRB: est logé à l'adresse **0x17**; quelle information contient DDRB ?  
**il définit si les pin du port B sont en sortie ou en entrée**  
(0= **entree**, 1= **sortie**, 1 désigne **que le port est en sortie**);
- PINB: est à l'adresse **0x16**; que contient PINB si le port est configuré en entrée ?  
**Il contient les donné qui sont "lu" sur les pins de ce port**.

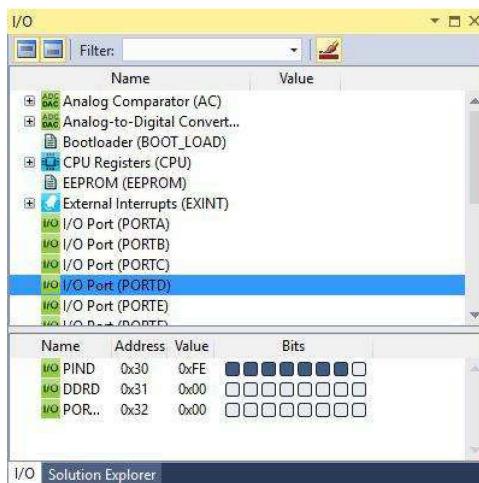
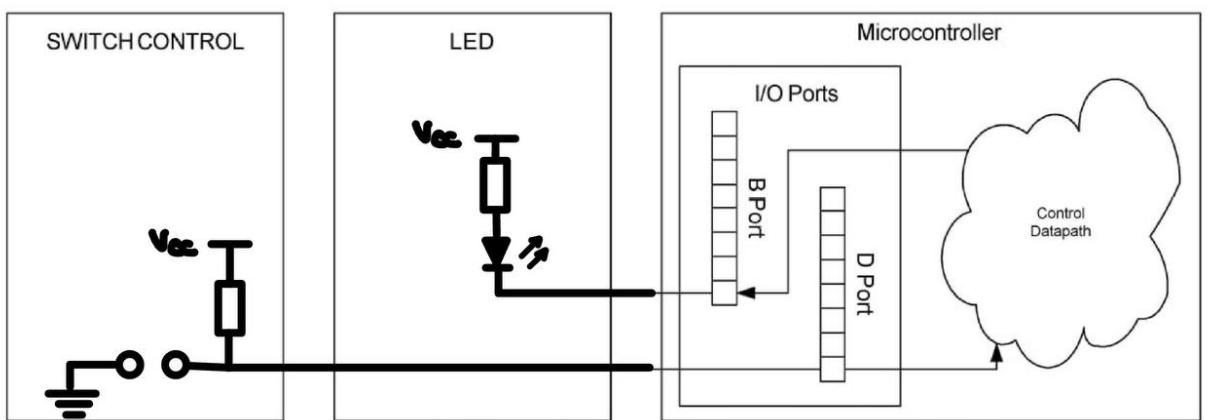


Figure 2.3: Fenêtre de status des entrées/sorties.

DDRB signifie **data direction register** du port **B**.

Les symboles DDRB et DDRD sont nommés  **constantes symboliques** et sont définies dans le fichier **m128def.inc**. Ce fichier peut être accédé depuis la fenêtre Solution Explorer; dans le projet courant, par exemple AssemblerApplication1 se trouve un sous-répertoire nommé Dependencies qui liste tous les fichiers dont le fichier courant dépend. A ce stade inout1.asm ne dépend que d'un seul autre fichier. Ouvrez ce fichier et étudiez ce qui s'y trouve. Par exemple, trouvez-y les valeurs des adresses de DDRB=**0x17** et DDRD=**0x11**.

La différence observée entre l'adresse le DDRB lue dans la fenêtre des I/Os et dans le fichier de configuration résulte de la technique de “memory mapping” qui définit les registres de I/Os comme équivalents à des adresses normales de la mémoire SRAM. En conséquence, les trois registres de chaque port sont accessibles par des instructions aux adresses 0x00 à 0x3F; ils sont également accessibles comme une adresse de la mémoire SRAM située 0x20 plus haut. Complétez ci-dessous le schéma simplifié du circuit comprenant un interrupteur, les ports du microcontrôleurs et une LED.



En conséquence, presser sur un interrupteur sans rebond (activer) conduit à programmer un niveau logique **0** dans le registre PIND; de plus, un niveau logique ‘1’ présent dans le registre PORTB conduit à **eteindre** la LED correspondante.

Modifiez le programme de manière à ce que toutes les LEDs soient allumées lorsque l'interrupteur n'est pas activé, et que la LED dont on presse le bouton s'éteigne. Quelle est votre solution ?

**en remplaçant la boucle par :**

```
loop:  
    in r16, PIND  
    com r16  
    out PORTB, r16  
    rjmp loop
```

Vous pouvez maintenant tester le programme inout1.asm fonctionnant sur la carte STK300.

Le fichier qui peut être exécuté par le système cible est généré au format .hex (Intel Hex - extended). Ce fichier est généré lors de l'assemblage du projet par Build→Build Solution. Le fichier .hex est stocké dans le répertoire tp02/AssemblerApplication1/AssemblerApplication1/Debug sous le nom AssemblerApplication1.hex.

## 2.2 TÉLÉCHARGER UN PROGRAMME VERS LE SYSTÈME CIBLE, DANS LA MÉMOIRE-PROGRAMME DU MCU

Connectez la carte STK-300 à l'ordinateur par un port USB. A cette fin, effectuez les opérations suivantes:

- Branchez le programmeur ISP sur un port USB du PC.
- Reliez le programmeur à la carte STK-300 au moyen du câble plat gris.
- Connectez le bloc d'alimentation 9V au réseau 220V.
- Enclenchez la carte au moyen du petit interrupteur noir situé au fond à gauche. La LED ON doit être allumée.

Le software qui télécharge le fichier .hex, et contrôle le programmeur AVRISP-U peut être déclenché depuis AtmelStudio, si le plugin a été installé, par la commande Tools→AVRISP-U. L'avantage de cette méthode est que AVRISP-U connaît le chemin vers le fichier .hex. AVRISP-U peut aussi être activé comme un programme normal, et il sera alors nécessaire de lui indiquer le chemin vers le fichier .hex qui devra être téléchargé. Effectuez les étapes suivantes.

- Lancez le programme AVRISP-U (version 2.4.1.13). Ce logiciel a pour tâche de télécharger le fichier .hex généré par l'assembleur vers le système cible. Ce fichier sera inscrit dans la mémoire-programme flash, non-volatile et interne du microcontrôleur. La fenêtre principale du programmeur est présentée en Figure 2.4. On y observe les adresses de programmation sur la colonne de gauche, le code au format hexadécimal sur la colonne du milieu, et le code au format ASCII sur la colonne de droite. Choisissez le microcontrôleur dans la fenêtre Device située au sommet.
- Configurez le programmeur par les deux étapes suivantes. Le menu Device→Autoprogram Options est présenté à la Figure 2.5.a; configurez le programmeur comme présenté: activez Reload Files (rechargera automatiquement le fichier .hex désigné), Erase Device, Program Flash Memory (effacer puis programmer, c'est-à-dire télécharger le .hex dans la mémoire), Run (exécution du programme). Puis, cliquez sur le bouton Setup situé au sommet droit de la fenêtre principale afin de configurer les paramètres de la transmission. Le menu présent en Figure 2.5.b apparaît. Ces paramètres dépendent du PC utilisé, et peuvent différer. La programmation échouera en l'absence d'un paramétrage correct. Les deux paramètres suivants doivent être paramétrés: Programmer ISP Speed (doit être plutôt lent avec un microcontrôleur opérant à 4MHz); USB Latency: 30. Des valeurs différentes seront communiquées si elles doivent être adaptées à de nouveaux ordinateurs dans les salles de TP.
- Il est également nécessaire de configurer la source d'horloge du MCU. Dans l'onglet Fuses & Lock Bits, on sélectionne la partie Fuses en cliquant sur un onglet situé sur la droite. Les sources d'horloges possibles apparaissent dans un menu nommé Clock Sources. On choisira l'option 0011: Calibrated Internal RC Oscillator 4.0 MHz (Figure 2.6). La programmation est effectuée par Device→Program→Fuses.

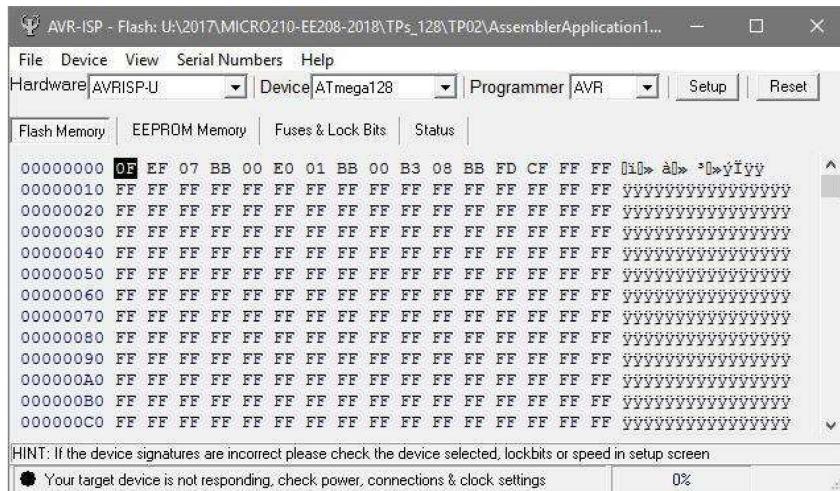


Figure 2.4: Fenêtre “Flash Memory” de AVRISP-U.

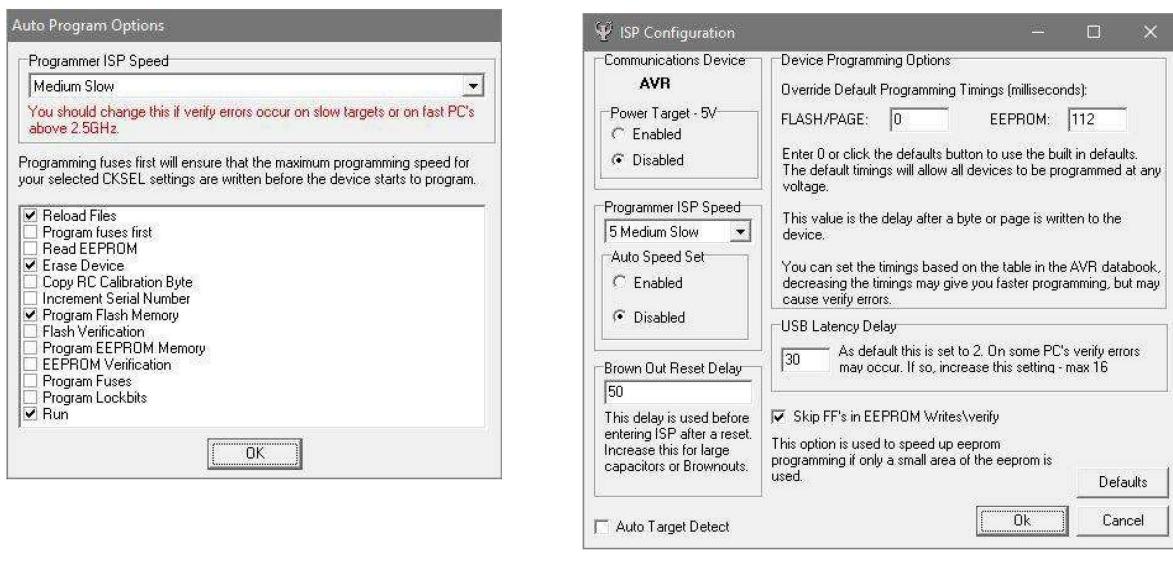


Figure 2.5: Configuration du programmeur.

- Lors de la première utilisation d'un nouveau programme, il faut indiquer le fichier .hex qui doit être téléchargé: File→Load→Flash, puis recherchez le fichier .hex généré par AtmelStudio. Cette étape n'est pas nécessaire si AVRISP-U a été lancé depuis un menu de AtmelStudio.
- Le déclenchement de la programmation se fait par: Device→Autoprogram ou F5. Pendant la programmation, la LED verte du programmeur ISP clignote, et la LED rouge nommée ISP sur la carte STK-300 (deuxième depuis la gauche du pavé de dix LEDs) est allumée pendant la durée de la programmation.

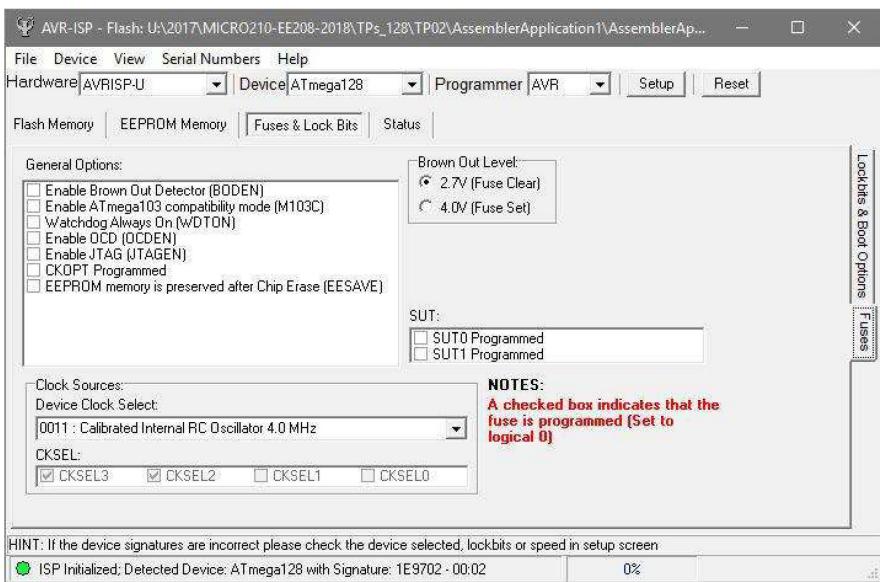


Figure 2.6: Fenêtre de configuration des Fuses, et de l'horloge principale.

## 2.3 MACROS

Introduisez au début du programme inout1.asm une macro OUTI (output immediate) qui place une constante (= valeur immédiate) dans un registre d'entrée/sortie. La définition d'une macro commence par la directive **.macro** et se termine par la directive **.endmacro**.

Complétez le code de la macro en Figure 2.7.

---

```
.macro OUTI
    ldi r16, @1
    out @0, r16
.endmacro
```

---

Figure 2.7: Macro OUTI.

## 2.4 DÉTECTION DE FLANC

Chargez le programme inout2.asm en Figure 2.8. Travaillez avec le simulateur - ne téléchargez pas !

- Que signifie l'instruction sbic ? **sauve l'instruction suivante si le bit est 0**
- Que signifie l'instruction sbis ? **sauve l'instruction suivante si le bit est 1**

Désactivez les deux lignes indiquées ci-dessous (mettez-les en commentaire):

```
;sbic    PIND, 0
;rjmp    PC-1
```

---

```

; file      inout2.asm
; purpose led in/out

reset:
    ldi r16,0xFF ; configure portB as output
    out DDRB,r16
    ldi r16,0x00 ; configure portD as input
    out DDRD,r16
    clr r16

loop:
    sbic PIND,0
    rjmp PC-1
    sbis PIND,0
    rjmp PC-1

    dec r16      ; decrement counter
    out PORTB,r16 ; output result to LEDs
    rjmp loop

```

---

Figure 2.8: inout2.asm.

Assemblez et exécutez. Observez les ports d'entrée/sortie B et D (Debug→Window→I/O). Agissez sur le port d'entrée/sortie D. Que se passe-t'il ?

<b>tant que le bit 0 du registre PIND est à 0, il attend. Si à 1, décrémente r16 et copie r16 dans portB</b>
<b>donc on voit le résultat sur les leds</b>

Reactivez ces deux lignes et désactivez les deux lignes ci-dessous (mettez les en commentaire):

```

;sbis      PIND,0
;rjmp      PC-1

```

Assemblez et exécutez, en observant, et en agissant sur les ports d'entrée/sortie. Que se passe-t'il ?

<b>tant que le bit 0 du registre PIND est à 1, attend. Si à 0, décrémente r16 et copie r16 dans portB</b>
<b>donc on voit le résultat sur les leds</b>

La boucle dans ce cas dure **6** cycles. Le registre r16 est décrémenté toutes les **1.5** microsecondes. Le MSB du groupe de LEDs est allumé pendant **768** microsecondes et éteint pendant **768** microsecondes.

Donnez le détail de votre calcul ci-dessous:

<b>128*6 uS = 768 uS</b>
<b>car on doit attendre que le compteur soit décrémenté avant de recommencer</b>

Cette LED clignote donc à une fréquence de **1302** Hz. Pour cette raison, il n'est pas utile de travailler avec la carte dans cet exercice, car on ne peut pas distinguer l'état des LEDs.

S'il était possible d'observer les états consécutifs des LEDs, qu'observerait-on ? Les LEDs allumées décrivent un compteur qui est incrémenté ou décrémenté **incrémenté**.

Il paraît donc utile de pouvoir disposer d'un mécanisme permettant d'imposer une latence à l'exécution de certaines instructions choisies.

## 2.5 BOUCLE D'ATTENTE ET COMPTEUR

Chargez le programme 01.asm en Figure 2.9. Travaillez avec le simulateur - ne téléchargez pas !

---

```
; file 01.asm target ATmega128L-4MHz-STK300
; purpose delay loop

reset:
    ldi r16,0xFF
    out DDRB,r16; portB = output
loop:
    dec r16          ; decrement low-byte counter
    nop
    nop
    nop          ; adjust to 1.5us loop time
    brne loop      ; loop back (internal loop)
    dec r17          ; decrement high-byte counter
    brne loop      ; loop back (external loop)
    dec r18          ; increment (!) LED counter
    out PORTB,r18    ; output register to portB
    rjmp loop
```

---

Figure 2.9: 01.asm.

Faites une simulation pas à pas, et aidez-vous de la fonction Stop Watch située dans la fenêtre Processor Status pour effectuer des mesures. Le chronomètre est mis à zéro par Reset Stop watch (SBD sur Stop Watch). De plus, il est nécessaire de configurer la fréquence du MCU à 4 MHz qui par défaut serait de 1 MHz. Cela ce fait en éditant le champ situé en regard de la propriété “Frequency.”

La durée de la boucle interne est de **6** cycle(s) ou **1.5** microseconde(s). L'instruction **nop** qui signifie **no operation** et qui a la fonction suivante **ne rien faire pendant un cycle** est utilisée pour ajuster la durée de boucle.

L'instruction **brne** signifie **branch in not equal**. Elle dure **2** cycle(s) s'il y a un branchement, et **1** cycle(s) s'il n'y a pas de branchement. Le branchement est fait si le fanion **z** est égal à **0**.

Placez un point d'arrêt (Debug→Toggle Breakpoint, F9) sur la deuxième instruction “brne loop”. Il est alors possible d'exécuter rapidement le programme avec la fonction Debug→Continue (F5). Le simulateur exécute toutes les instructions séquentiellement, et s'interrompt à la ligne qui contient le point d'arrêt.

La durée de la deuxième boucle est de **384,5** microseconde(s), sauf lors de la toute première exécution car les registres des compteurs ne sont pas initialisés dans la boucle (r16 à 0xff lors du premier passage, puis à 0x00).

Ce résultat se compose de :

- **255** x **1.5** microseconde(s) correspondant à 255 passages de la boucle interne avec branchement,
- **1** x **1.25** microseconde(s) correspondant à la boucle interne sans branchement (passage 256)
- **0.75** microseconde(s) de surplus pour la boucle externe.

Effacez tous les points d'arrêts, puis mettez-en un à l'instruction “dec r18.” Quelle est la période de décrémentation ? **98.43** milliseconde(s). Il est nécessaire d'attendre un petit moment, parce qu'il faut un certain temps au simulateur pour exécuter/simuler tous ces pas. Le bit0 clignote donc à une fréquence de **5.0797** Hz. Téléchargez et vérifiez !

## 2.6 BOUCLE D'ATTENTE PARAMÉTRÉE

Ainsi, il est possible de définir très précisément le temps nécessaire à l'exécution d'une boucle d'attente. Il est possible de s'aider de l'assembleur afin de faire le calcul de constante, permettant de créer des boucles d'attentes paramétrées.

Chargez le programme `loop2.asm` défini en Figure 2.10. Assemblez, mais ne téléchargez pas.

---

```
; file      loop2.asm    target ATmega128L-4MHz-STK300
; purpose  parametrized delay loop

.equ clock= 4000000 ; clock speed 4MHz
.def w     = r16        ; r16 is used in macro

; --- macro definition ---
; wait k (k=3...768) cycles in increments of 3 cycles
.macro  SWAIT_C       ; k
    ldi w,low(@(0)/3)
a:    dec w
    brne a
.endmacro

.macro  SWAIT_US      ; k
; wait a maximum of 768/clock(MHz) microseconds
    ldi w,low((clock/1000*@0)/3000)
a:    dec w
    brne a
.endmacro

; --- main program ---
reset:
    ldi r16,0xFF
    out DDRB,r16        ; portB = output
loop:
    nop
    SWAIT_C 100        ; wait 100 cycles
    inc r0
    out PORTB,r0
    SWAIT_US 15        ; wait 15 us
    inc r0
    out PORTB,r0
    rjmp loop
```

---

Figure 2.10: `loop2.asm`.

Quels sont les rôles des deux macros définies dans ce programme ?

- la macro `SWAIT_C` réalise une boucle d'attente égale à **N cycle**  
**(mais programmable via l'argument de la macro et le résultat est modulo 3 cycle donc pour 100 cycle, 99 seront passé)**
- la macro `SWAIT_US` réalise une boucle d'attente égale à **N microseconde**  
**(mais programmable via l'argument de la macro)**.

Ajoutez les lignes suivante dans la partie principale du programme; effectuez une simulation en mode pas-à-pas (F11) et mesurez les temps d'attente obtenus en termes de cycles:

```
SWAIT_C 10      ;délai de [9] cycles;
SWAIT_C 20      ;délai de [18] cycles;
SWAIT_C 30      ;délai de [30] cycles;
SWAIT_C 100     ;délai de [99] cycles;
SWAIT_C 1200    ;délai de [432] cycles.
```

On observe que les valeurs sont arrondies, puisque la résolution de la boucle d'attente est de [3] cycles. La dernière valeur n'est pas proche de 1200. Que s'est-il passé ? La macro SWAIT\_C ne fonctionne seulement qu'avec des arguments jusqu'à une limite de [3\*256=768].

Il y a dépassement. Observez le dépassement dans le code désassemblé.

Dans le code précédent, changez SWAIT\_C par SWAIT\_US, simulez à nouveau et mesurez les temps d'attente obtenus:

```
SWAIT_US 10      ;délai de [9.75] µsec;
SWAIT_US 20      ;délai de [19.5] µsec;
SWAIT_US 60      ;délai de [60] µsec;
SWAIT_US 100     ;délai de [99.75] µsec;
SWAIT_US 1200    ;délai de [48] µsec.
```

La résolution temporelle est de [0.75] microsecondes. La dernière instruction génère donc le problème grave de [overflow].

## 2.7 GÉNÉRATION DE SON

Les délais sont utilisés afin de générer des sons qui soient audibles. Il est possible de générer une forme d'onde carrée en chargeant une pin d'un port programmé en sortie et connecté à un haut-parleur piézo-électrique avec des '0' et des '1' en alternance. La fréquence audible est déterminée par la période totale.

---

```
; file speaker1.asm target ATmega128L-4MHz-STK300
; purpose fixed frequency sound through piezo
; module: M2, output port: PORTE
.include "macros.asm"
.include "definitions.asm"

reset:
LDSP    RAMEND          ; load stack pointer SP
OUTI    DDRB,$ff          ; make LEDs output
sbi     DDRE,SPEAKER     ; make pin SPEAKER an output

main:
sbi     PORTE,SPEAKER
WAIT_US [1136]           ; insert delay here

cbi     PORTE,SPEAKER
WAIT_US [1136]           ; insert delay here
rjmp   main
```

---

*Figure 2.11: speaker1.asm.*

Téléchargez le programme `speaker1.asm` donné en Figure 2.11, après avoir ajusté le délai et la fréquence associée afin de générer un la 440 Hz. La fréquence générée doit être soigneusement calibrée. En effet, les fréquences audibles sont situées dans une gamme de 20 Hz à 16 kHz. Le microcontrôleur fonctionne à une fréquence de **4** MHz. Afin de produire la note la 440 Hz, il nécessite de compléter le bit de contrôle du haut-parleur au moyen des instructions `sbi` et `cbi` chaque **1.136** msec. Chacune de ces instructions est exécutée en deux cycles, de même pour l'instruction `rjmp`. Il est ainsi possible de calculer le délai associé à la fréquence produite.

Notez les dépendances nouvelles du fichier `speaker1.asm` qui sont les fichiers `macros.asm` et `definitions.asm`, et qui apparaissent après la directive `.include`. Il est nécessaire de placer ces fichiers dans le répertoire du projet: `\tp02\AssemblerApplication1\AssemblerApplication1`. A cet endroit se trouvent également recopier les fichiers `.asm` inclus dans le projet. Finalement, le fichier `m128def.inc` est aussi une dépendance par défaut; il n'est pas nécessaire de le répéter. Après avoir assemblé (F7), les trois fichiers de dépendances apparaissent dans le Solution Explorer sous Dependencies.

Vérifiez la fréquence obtenue au moyen de l'oscilloscope; la touche "measure" permet d'afficher la fréquence, qui s'affiche aussi automatiquement au fond de l'écran. La sonde de l'oscilloscope est placée sur la pin du header du PORTE qui porte la constante symbolique `SPEAKER`; vérifiez dans le fichier `definitions.asm` de quelle pin il sagit **0x02** puis placez (accrochez) la sonde de l'oscilloscope sur cette pin. La terre de la sonde (pince crocodile) doit être connectée à la terre de la carte (anneau métallique situé au fond à droite de la carte STK300).

Le module M2 comprenant un haut-parleur piezo-électrique peut être placé sur le PORTE pour réalisez l'application et entendre le son; cette information de connection se trouve également dans l'en-tête du fichier.

Dans le programme `speaker1.asm`, le son est produit en permanence. Modifiez ce programme en `speaker1b.asm` en Figure 2.12, afin de ne produire un son que si le bouton-poussoir numéro 1 est activé.

---

```

; file      speaker1b.asm    target ATmega128L-4MHz-STK300
; purpose   fixed frequency sound through piezo
; module:  M2, output port: PORTE
.include "macros.asm"
.include "definitions.asm"

reset:
    LDSP      RAMEND           ; load stack pointer SP
    OUTI      DDRB,$ff          ; make LEDs output
    sbi       DDRE,SPEAKER     ; make pin SPEAKER an output

main:
    outi DDRB,$ff              ; insert additional instruction here
    sbic PIND,0                ; insert additional instruction here
    rjmp main                  ; insert additional instruction here
    sbi      PORTE,SPEAKER
    WAIT_US  1136              ; same as in speaker1.asm
    cbi      PORTE,SPEAKER
    WAIT_US  1136              ; same as in speaker1.asm
    rjmp    main

```

---

Figure 2.12: `speaker1b.asm`.

Assemblez, téléchargez et vérifiez.

## 2.8 CONCEPTS AVANCÉS (FACULTATIF)

Les concepts plus avancés suivants sont appliqués au traitement monophonique du son dans le programme sound1.asm en Figure 2.13, et seront étudiés dans la suite et les TPs futurs:

- utilisation d'une fonction;
- inclusion de fichiers (modularité);
- utilisation d'un pointeur (z);
- utilisation d'une table look-up (table de transcorrespondance).

Assemblez et téléchargez le programme sound1.asm. Le fichier dépendant sound.asm doit être placé dans le répertoire du projet.

Les deux arguments de la fonction sound sont les registres:

- qui représente ,
- qui représente .

Quelle fonction réalise le programme sound1.asm ?

le programme joue une note d'une durée et fréquence réglable en touchant le n ième bouton
(le programme compte combien de fois le registrer du port des bouton doit etre shifté donc
quelle bouton et calcul ensuite la durée a partir de ça

---

```
; file      sound1.asm    target ATmega128L-4MHz-STK300
; purpose variable freq. input sound through piezo
; module: M2, output port: PORTE
.include "macros.asm"
.include "definitions.asm"

reset:
LDSP      RAMEND          ; load stack pointer SP
OUTI      DDRB,$ff         ; make LEDs output
sbi       DDRE,SPEAKER    ; make pin SPEAKER an output
rjmp     main

.include "sound.asm"        ; include sound routine

main:
in      w,PIND           ; read buttons
out     PORTB,w           ; write result to LEDs

ldi    a0,-1              ; preload a0 with -1
clc
lsr    w                  ; carry=0
inc    a0                  ; shift right, LSB-> carry
inc    a0                  ; increment counter
brcs   PC-2

clr    a1                  ; clear high byte
ldi    zl, low(2*tbl)     ; load table base into z
ldi    zh,high(2*tbl)
add   zl,a0               ; add offset to table base
adc   zh,a1               ; add high byte
lpm    r0                 ; load program memory, r0 <- (z)

mov   a0,r0               ; load oscillation period
ldi   b0,20                ; load duration (20*2.5ms = 50ms)
rcall sound
rjmp  main

tbl:
.db  do,re,mi,fa,so,la,si,do2,0
```

---

*Figure 2.13: sound1.asm.*

