LABORATOIRE DE SYSTEMES MICROELECTRONIQUES

EPFL STI – IMM – LSM
ELD
Station n° 11
CH-1015 Lausanne

Téléphone : +4121 693 6955
Fax : +4121 693 6959
E-mail : lsm@epfl.ch
Site web : lsm.epfl.ch

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Microcontrôleurs 2012, MT-BA4: TP03-2012-v3.5.fm

v1.0	R. Holzer	I2S	2000-2003
v2.4.1	A. Schmid	LSM	Janvier 2006
v3.5	A. Schmid	LSM	Octobre 2011

MICROCONTRÔLEURS
TRAVAIL PRATIQUE NO 3

GROUPE A	Mercredi 07.03.2012, 08:00-10:00	GROUPE B	Lundi 12.03.2012, 11:00-13:00

No du Groupe	Premier Etudiant	Second Etudiant	Evaluation	Visa Correcteur
ME 8	Demierre Timothée	Bereau Jean-Bernard	1	

3. OPÉRATIONS BOOLÉENNES, OPÉRATIONS SUR LES BITS, ET UTILISATION DE L'OSCILLOSCOPE

Ce travail pratique voit l'étude des opérations Booléennes, ainsi que de diverses opérations sur des bits particuliers. Les instructions, et méthodes étudiées sont fondamentales au développement de programmes complexes.

L'oscilloscope est mis en oeuvre dans une deuxième partie, démontrant le type de sigaux générés aux ports du microcontrôleur, ainsi que la méthode de développement/deverminage basée sur l'analyse des signaux temporels.

PREPARATION PREPARATION PREPARATION

3.1 LES FANIONS

Le registre SREG signifiant en anglais [status register] contient des bits qui sont mis à '1' ou '0' en fonction du résultat des opérations arithmétiques. Trois des fanions les plus utilisés sont:

- C qui est l'abréviation de [carry] et qui signale [le dépassement de capacité lors des additions et soustractions non signées],
- Z qui est l'abréviation de [zero] et qui signale [que le résultat d'une opération est egal à 0],
- N qui est l'abréviation de [negative] et qui signale [que le resultat d'une opération est negatif].
-

Ecrivez le programme qui vous permet de simuler les opérations données en Table 3.1. Exécutez en pas-à-pas, reportez vos résultats, et soyez sûrs d'avoir bien compris les vraies raisons du comportement du microcontrôleur.

Opération	Résultat	H	S	V	N	Z	C	Instruction à utiliser	Validité du résultat
0xec+0x41	0x2D	0	0	0	0	0	1	adc	faux : overflow C=1 ⇒ carry généré !
0x41-0xec	0x55	1	0	0	0	0	1	sbc	faux : dépassement de capacité C=1

Table 3.1: Opérations arithmétiques.

Opération	Résultat	H	S	V	N	Z	C	Instruction à utiliser	Validité du résultat
0xfa-0xfe	0xFC	1	1	0	1	0	1	sbc	faux Vrai
0x37-0xa4	0x93	0	0	1	1	0	1	sbc	faux

Table 3.1: Opérations arithmétiques.

3.2 OPÉRATIONS BOOLÉENNES

3.2.1 OPÉRATIONS SUR DES OPÉRATEURS 8-BIT

L'assembleur AVR connaît trois fonctions Booléennes ayant deux opérandes. Elles s'appellent and, or, et eor (exclusive or). Complétez en Table 3.2 la table de vérité pour ces trois fonctions.

a	1	1	0	0
b	1	0	1	0
a and b	1	0	0	0
a or b	1	1	1	0
a eor b	0	1	1	0

Table 3.2: Table de vérité des trois fonctions.

Ces fonctions sont souvent appliquées afin d'exécuter la fonction Booléenne en parallèle sur les 8-bit d'un registre en [1] cycle(s). Simulez le programme donné Figure 3.1 et reportez les résultats observés en Table 3.3.

```
reset:    ldi  r16, 0b1100
          ldi  r17, 0b1100
          ldi  r18, 0b1100
          ldi  r19, 0b1010

          and  r16, r19
          or   r17, r19
          eor  r18, r19
```

Figure 3.1: Test des fonction Booléennes.

Registre	Hexadécimal	Décimal	Binaire
r16	0x08	8	0b1000
r17	0x0E	14	0b1110
r18	0x06	6	0b0110

Table 3.3: Résultat d'exécution du programme donné en Figure 3.1.

Une autre application courantes des fonctions Booléennes sur deux opérandes 8-bit consiste à masquer un certain nombre de bits. La fonction du masque est généralement de:

- forcer certains bits choisis à '1', tout en ne modifiant pas les autres, ou
- forcer certains bits choisis à '0', tout en ne modifiant pas les autres.

Téléchargez le programme donné en Figure 3.2. Etudiez-le, puis effectuez les manipulations qui vous permettent de répondre aux questions suivantes:

```
.include "m103def.inc"
reset:  ldi     r16,0xff
        out     DDRB,r16  ; make portB an output

loop:   in      r16, PIND
        ori     r16, 0b00000010
        andi    r16, 0b11110111
        out     PORTB,r16
        rjmp    loop
```

Figure 3.2: Test de fonctions de masquage.

- qu'observez-vous sur les LEDs ? Quelles sont les LEDs toujours/jamais/parfois et dans quelles conditions actives (active=allumée) ?

La LED 3 est toujours active
La LED 1 est toujours éteinte
Les autres LED s'allument quand on appuie sur le bouton correspondant

- quel type de masque est réalisé par l'instruction ori ? il force certains bits à 1
- quel type de masque est réalisé par l'instruction andi ? il force certains bits à 0

Il y a plusieurs façons de créer des masques. Typiquement, l'opérateur de décalage "<<" peut être utilisé afin de placer un bit isolé à sa position désirée. Donnez quatre façons permettant de fixer les bits 3 et 7 à '1':

```
ori     r16, ( 1 << 6 + 1 ) + ( 1 << 2 * 3 )
ori     r16, 0b 1000 1000
ori     r16, 0x 88
ori     r16, 128+ 8
```

De même, indiquez quatre manières de mettre à '0' les bits 2 et 4. Aidez-vous de l'opérateur de négation (~) si nécessaire:

```
andi    r16, ~ ( ( 1 << 2 ) + ( 1 << 4 ) )
andi    r16,0b 11101011
andi    r16,~0x 0x16
andi    r16, 0xff-0x 10 -0x 04
```

En vous aidant de ce qui a été étudié précédamment, soit l'utilisation d'opérateurs Booléens et de masques pour modifier des parties spécifiques d'un byte, complétez la macro INVB donnée en Figure 3.3 qui a pour fonction d'inverser un bit choisi dans un mot. Téléchargez le code et vérifiez votre programme.

PREPARATION

```
.include "m103def.inc"

; inverse a bit (INVB reg,bit)
.macro   INVB
     ldi     r16, 1 << @1          ; create mask
     eor     @0, r16               ; execute masking operation
.endmacro

reset:ldi    r16,0xff
     out     DDRB,r16              ; make portB an output

loop:in      r0, PIND
     INVB    r0,1
     INVB    r0,3
     out     PORTB,r0
     rjmp    loop
```

Figure 3.3: Macro INVB.

3.2.2 OPÉRATIONS BOOLÉENNES SUR DES BITS SINGULIERS

Nous avons étudié comment effectuer des opérations Booléennes sur des mots de 8-bits; cette méthode n'est pas applicable à des opérations Booléennes sur des bits singuliers, choisis aléatoirement dans un ou plusieurs registres, comme suggéré en Figure 3.4.

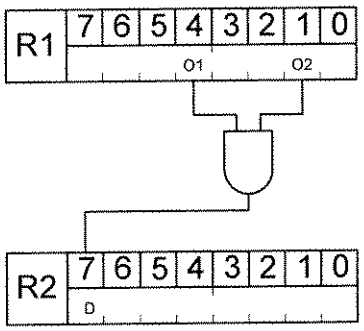


Figure 3.4: Opération Booléenne sur des bits singuliers.

Comme le suggère la Figure 3.4, les opérations sur des bits singuliers nécessitent de pouvoir effectuer des transferts de bits. Pour cela, les instructions de transfert vers et du bit T sont utilisées:

- l'instruction bst reg,b qui [écrit la T le bit b du registre reg],
- l'instruction bld reg,b qui [écrit au bit b du registre reg le bit T].

Définissez en Figure 3.5 une macro MOVB qui copie un bit choisi d'un registre vers un bit choisi d'un autre registre.

```
.macro MOVB
;reg0,b0 <- reg1,b1
    bt      @2, @3
    bld     @0, @1
    .endmacro
```

Figure 3.5: Macro MOVB.

Complétez le programme donné en Figure 3.6 afin q'il effectue une copie de la valeur de LED2 sur LED6.

```
.include "m103def.inc"
.include "macros.asm"


reset:ldi     r16,0xff
      out     DDRB,r16          ; make portB an output

loop:in       r0,PIND
      MOVB    0, 6, 0, 2
      out     PORTB,r0
      rjmp    loop
```

Figure 3.6: Copie de bit sur le portB.

Considérons maintenant les opérations Booléennes sur des bits singuliers. Une solution efficace est proposée dans la macro ANDB. Assemblez le code donné en Figure 3.7 et etudiez-le.

```
.include "macros.asm"

      ANDB    r2,7, r1,1, r1,4
```

Figure 3.7: Etude de la macro ANDB.

Reportez en Figure 3.8 le code désassemblé généré; indiquez à la place du commentaire à quoi servent les quatre instructions ?

```
+00000000: 9468    SET
+00000001: FE14    SBRS    R1,4    ; Saute la prochaine instruction si le bit 4 est à 1
+00000002: 94E8    CLT             ; Met T à 0
+00000003: FE11    SBRS    R1,1
+00000004: 94E8    CLT
+00000005: F827    BLD     R2,7    ; Charge la valeur de T dans le bit 7 de r2
```

Figure 3.8: Code désassemblé généré pour la macro ANDB.

Ainsi il est possible d'émuler la fonction ANDB au moyen d'une suite d'instructions qui font appel:

- au bit T utilisé pour stocker la valeur de chaque bit,
- à l'instruction SET qui permet de forcer à '1' le bit T, et à l'instruction CLT qui permet de forcer à '0' le bit T,
- aux instruction sbrs ou sbrc qui sautent une instruction suivant la valeur d'un bit de contrôle,

- à l'instruction bld.

Compléter le diagramme de flux en Figure 3.9 décrivant les opérations exécutées par la macro ANDB.

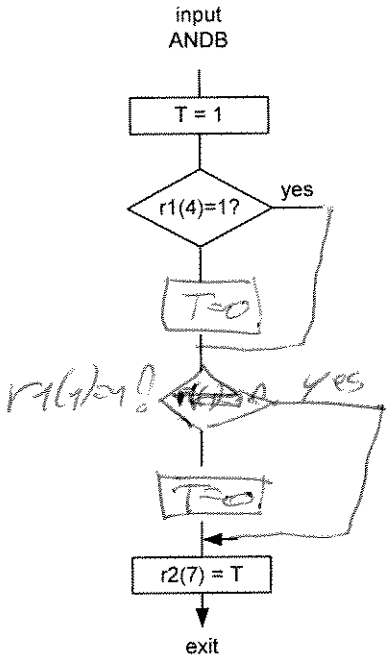


Figure 3.9: Diagramme de flux de la macro ANDB.

Sur le même principe, complétez en Figure 3.10 la macro NORB réalisant la fonction NOR sur des bits singuliers.Simulez afin de vérifier le comportement correct.

```
; macro definition
.macro    NORB
; r0,b0 <- r1,b1 NOR r2,b2

    set
    sbrc                @4, @5
    clt
    sbrc                @2, @3
    clt
    bld                 @0, @7
.endmacro;
```

Figure 3.10: Macro NORB.

3.3 UTILISATION DE L'OSCILLOSCOPE

L'oscilloscope est un outil indispensable au développement et à la maintenance de systèmes électroniques sur carte PCB (printed circuit board). Des points de mesure peuvent être contrôlés au niveau de leur timings afin de garantir la syncronisation des différents éléments. La vérification de valeur logiques sur une large

plage temporelle doit être réalisée au moyen d'un analyseur logique. L'oscilloscope permet cependant de contrôler de relativement petits paquets, ce qui est souvent suffisant à un déverminage efficace.

3.3.1 GÉNÉRATION DE PULSES

Le programme pulsout1b.asm donné en Figure 3.11 génère une impulsion répétée sur une ligne. Indiquez combien de cycles, et donc combien de temps nécessite l'exécution des instructions.

```
; file    pulsout1b.asm
.include "m103def.inc"       ; include definitions

reset:
     ldi  r16,0xff           ; load immediate value into register
     out  DDRE,r16           ; output register to i/o Data Direction
main:
     sbi  PORTE,7            ; set bit 1 in i/o port E- 2 cycles 500 ns
     cbi  PORTE,7            ; clear bit 1 in i/o port E- 2 cycles 500 ns
     nop                     ; No OPeration (do nothing- 2 cycles 500 ns
     nop
     rjmp main               ; jump back to main- 2 cycles 500 ns
```

Figure 3.11: pulsout1.asm.

Ainsi, l'exécution de la boucle nécessite [8] cycles, soit [1000] ns.

Téléchargez le programme sur le système cible. Connectez la sonde de l'oscilloscope à la broche PE7 et observez le signal généré. Le clip de masse doit être connecté à la broche GND. Configurez l'oscilloscope comme indiqué sur la Figure 3.12, soit:

- CHANNEL1, 5V et 500ns;
- trigger sur CH1, flanc montant à 2.5V;
- au moyen du bouton HORIZONTAL POSITION, modifiez la position du déclenchement du trigger jusque sur la gauche;
- mettre la sonde de l'oscilloscope sur "10X."

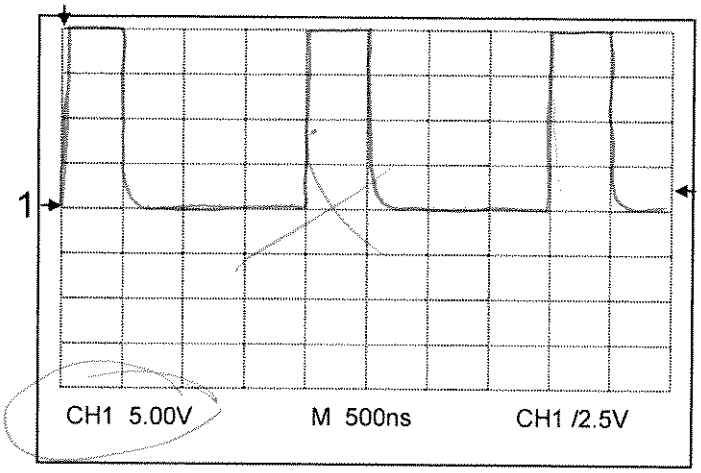


Figure 3.12: Observation de l'exécution de pulsout1b.asm.

Reportez sur la Figure 3.12 le signal obtenu, indiquez l'exécution des instructions correspondantes, et répondez aux questions suivantes (utilisez la fonction MEASURE afin de simplifer les mesures):

- Quelle est la durée de l'impulsion générée ? 500 ns.
- Quelle est la période du signal ? 2000 ns.
- Quel en est le rapport cyclique ? 0,25
- Quelle est la fréquence du signal ? 500 kHz.

3.3.2 TEMPS DE MONTÉE ET TEMPS DE DESCENTE

Les temps de montée et de descente d'un signal ne sont pas instantanés. L'oscilloscope permet de les visualiser et mesurer, afin de garantir une parfaite synchronisation entre les différents modules composant une carte.

Les temps de montée et de descente sont définis comme le temps mis par le signal pour passer de 10% à 90% (respectivement 90% à 10%) de son amplitude.

Utilisez programme pulsout1b.asm. Configurez l'oscilloscope comme indiqué en Figure 3.13 et Figure 3.14, puis reportez-y respectivement le flanc montant et le flanc descendant observés. Centrez la transistion, indiquez les niveaux 10% et 90% et mesurez les temps de montée et de descente. Aidez-vous des curseurs pour faciliter la mesure; il sont enclenchés par la fonction CURSOR; il faut choisir entre Time et Voltage puis utiliser les boutons VERTICAL→POSITION pour les positionner.

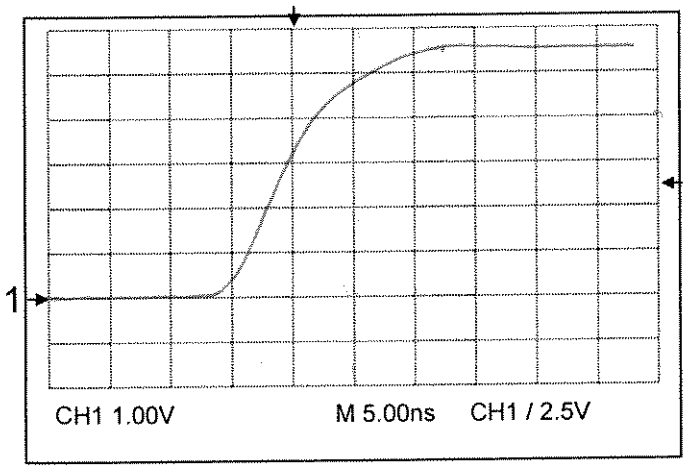


Figure 3.13: Flanc montant, temps de montée=environ ~~50 ns~~ ~~25 ns~~ 5-10 ns

Ecrivez un programme qui génère une impulsion positive d'une durée de 0.5 microseconde, un intervalle de 0.5 microseconde, une deuxième impulsion de 1 microseconde, puis un intervalle de 2.0 microsecondes en utilisant les instructions sbi, cbi, nop.

Quelle est la plus petite impulsion que l'on puisse générer, et pourquoi ?

0,5 µs, car cbi prend 2 clock !

Quelle est la résolution temporelle, et pourquoi ?

0,25 µs, car le nop, lui, ne prend qu'une clock

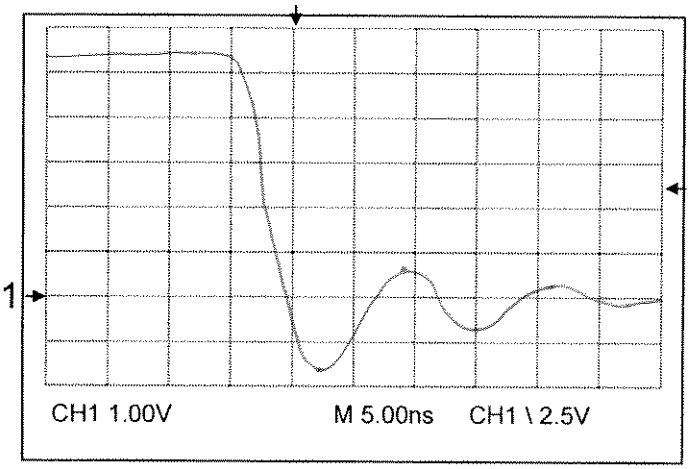


Figure 3.14: Flanc descendant, temps de descente=environ 25 ns (avec les oscillations)

LABORATOIRE DE SYSTEMES MICROELECTRONIQUES

EPFL STI – IMM – LSM
ELD
Station n° 11
CH-1015 Lausanne

Téléphone : +4121 693 6955
Fax : +4121 693 6959
E-mail : lsm@epfl.ch
Site web : lsm.epfl.ch

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Microcontrôleurs 2012, MT-BA4: TP04-2012-v3.5.fm

v1.0	R. Holzer	I2S	2000-2003
v2.4.1	A. Schmid	LSM	Janvier 2006
v3.5	A. Schmid	LSM	Octobre 2011

MICROCONTRÔLEURS
TRAVAIL PRATIQUE NO 4

GROUPE A	Mercredi 21.03.2012, 08:00-10:00	GROUPE B	Lundi 19.03.2012, 11:00-13:00

No du Groupe	Premier Etudiant	Second Etudiant	Evaluation	Visa Correcteur
ME 08	Demierre Timothee	Bideaux Yann-Bernard	7	S. Allemand

PREPARATION PREPARATION PREPARATION

4. UTILISATION DES SOUS-ROUTINES, INTRODUCTION À L'AFFICHAGE LCD

Ce travail pratique voit dans une première partie l'étude des sous-routines, et leur comparaison avec les macros. Le fonctionnement de la pile est également abordé.

Dans une deuxième partie, une introduction au fonctionnement du module LCD est présentée. L'accès à un contrôleur de LCD standard, ainsi que les opérations de base sont abordés.

4.1 LA PILE

La pile (stack) est une zone mémoire utilisée pour la sauvegarde de variables temporaires et des adresses de retour lors des appels de sous-routines. Le SP signifiant [stack pointer] est un pointeur sur une zone en mémoire SRAM. Le ATmega103 possède deux registres spéciaux SPL et SPH qui stockent cette valeur.

- L'instruction push r0 place [la valeur stockée dans r0 à l'endroit pointé par SP] puis, le SP est [decrementé].
- L'instruction pop r0 [incremente] le SP, puis place [la valeur pointée par SP dans r0.].

Simulez en pas-à-pas le code donné en Figure 4.1.

```
.include "m103def.inc"
     ldi  r16,0xff
     out  SPL,r16
loop:inc  r16
     push r16
     rjmp loop
```

Figure 4.1: Traitement avec la pile (1).

Le SP pointe à l'adresse [0x FF]. Dans la boucle, r16 est [incremente], puis cette valeur est placée [sur la pile].

L'instruction push dure [2] cycle(s).

Les instructions push et pop sont toujours utilisées en paire. L'instruction push est utilisée pour la sauvegarde temporaire du contenu de registres qui doivent être mis à disposition d'une macro ou routine par exemple. L'instruction pop restitue la valeur stockée.

Complétez, et simulez en pas-à-pas le code donné en Figure 4.2.

```
.include "m103def.inc"

     ldi  r16,0xff
     out  SPL,r16
     ldi  r16,0x0a
     ldi  r17,0x0b

loop:push r16        ; save r17, r16
     push r17

     clr  r17        ; do other stuff with r16,r17
     clr  r16

     pop  r17        ; restore r16, r17
     pop  r16
     rjmp loop
```

Figure 4.2: Traitement avec la pile (2).

Quelle valeur est restituée par l'instruction pop dans le cas où l'instruction push a été précédemment utilisée plusieurs fois ? [l'instruction pop retourne la dernière valeur mise dans la pile]. Ce principe de mémoire s'appelle LIFO, signifiant [last in first out], en opposition avec [FIFO] signifiant [first in first out].

Il convient toutefois de travailler avec la pile et les instructions push et pop de façon très rigoureuse. Voici trois cas dans lesquels des erreurs on été commises et qui conduisent à différent comportements erronnés. Identifiez les erreurs, leurs sources et conséquences. Pour vous aider dans cette tâche, affichez les fenêtres "processor," "registers" et "memory" à l'adresse de la pile et simulez en pas-à-pas.

- Simulez le code donné en Figure 4.3, puis répondez aux questions.
  - après l'exécution du pop, r16 ne récupère pas la valeur attribuée dans le reset, quelle valeur récupère-t'il à la place ? [r17];
  - la pile est-elle une zone mémoire protégée ? [non];
  - quel comportement erronné a ainsi été la source de l'erreur ? [r16 est stocké à 0x0fff. Or lorsqu'on écrase la commande st x,r17, la valeur de r17 est aussi stockée à 0x0fff car x est initialisé à cette valeur d'adresse. r16 est donc écrasé par r17 sur la pile]

```
; file    failure01.asm
.include "m103def.inc"
.include "macros.asm"
.include "definitions.asm"

reset:
     LDSP RAMEND    ;set up stack pointer (SP)

     ldi  r16, 0x11
     ldi  r17, 0xaa

main:
     ldi  xl, 0xff
     ldi  xh, 0x0f

     push r16

     st   x, r17

     pop  r16
```

Figure 4.3: Erreur type No. 1, failure01.asm.

- Simulez le code donné en Figure 4.4, puis répondez aux questions.

```
; file    failure02.asm
.include "m103def.inc"
.include "macros.asm"
.include "definitions.asm"


;=============================================
; this stands for a big macro, over which a
; programer may have lost control
.macro DISTRACT  ;void
     ;several instructions ...
     push r17
     push r18

     ;several instructions ...
     pop  r18

     ;several instructions ...
.endmacro

reset:
     LDSP RAMEND    ;set up stack pointer (SP)

     ldi  r16, 0x11
     ldi  r17, 0xaa
     ldi  r18, 0x55

main:
     push r16

     DISTRACT

     pop  r16
```

Figure 4.4: Erreur type No. 2, failure02.asm.

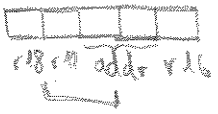
- après l'exécution du pop, r16 ne récupère pas la valeur attribuée dans le reset, quelle valeur récupère-t'il à la place ? [r17] ;
- quel comportement erronné a ainsi été la source de l'erreur ? [dans DISTRACT on push 2 valeurs mais seulement une est popée Quand on veut pope r16 c'est donc r17 qui vient]

- Simulez le code donné en Figure 4.5, puis répondez aux questions. Affichez le code désassemblé et suivez en pas-à-pas.

```
; file     failure03.asm
.include "m103def.inc"
.include "macros.asm"
.include "definitions.asm"

reset:
     LDSP RAMEND    ;set up stack pointer (SP)

     ldi  r16, 0x11
     ldi  r17, 0xaa
     ldi  r18, 0x55

main:
     push r16

     rcall distract

     pop  r16

;=============================================
; this stands for a big subroutine, where a
; programer may have lost control
distract:
     ;several instructions ...
     push r17
     push r18

     ;several instructions ...
     pop  r18

     ;several instructions ...
     ret
```

Figure 4.5: Erreur type No. 3, failure03.asm.

- quelle est l'adresse à laquelle le PC saute à l'exécution de l'instruction ret ? 0x00
- qu'est-ce que cela signifie du point de vue de l'exécution ? on redemarre le programme
- d'ou cette adresse a-t'elle été copiée vers le PC ? de la pile
- comment ces deux octets sont-ils parvenu (par "qui" ont-ils été placés) à cet endroit ?
  rcall place l'adresse du PC dans la pile pour pouvoir reprendre où il en était à la fin de la sous-routine.
- quel comportement erronné a ainsi été la source de l'erreur ? on push 2 fois et on pop qu'une seule fois → on prend la mauvaise adresse de retour



4.2 COMPARAISON ENTRE MACRO ET SOUS-ROUTINE

4.2.1 SOUS-ROUTINE

Une sous-routine est un bout de code qui commence par une étiquette, et qui se termine par l'instruction ret. Complétez la sous-routine mul5 donnée en Figure 4.6, qui multiplie l'argument r16 par cinq, sachant que l'instruction de multiplication n'est pas disponnible sur ATmega103.

```
; ============================================
; this subroutine multiplies a register by 5
; in: r16
; out:r16
; mod:r17
mul5:mov    r17, r16
     lsl    r16
     lsl    r16
     add    r16, r17
     ret
```

Figure 4.6: Sous-routine mul5.

A l'exécution de l'appel à une sous-routine, l'adresse de retour est pile sur la pile, et l'adresse de la première instruction à exécuter dans la sous-routine est placée dans le PC. Le SP est décrémenté afin de pointer vers la prochaine place libre. Simulez le code donné en Figure 4.7 basé sur la sous-routine mul5 développée précédemment. La sous-routine mul5 est appelée trois fois. Indiquez les éléments suivants: l'adresse courante du PC, l'adresse de retour courante de la sous-routine, l'argument courant d'entrée, et l'argument courant de sortie.

```
.include "m103def.inc"
.include "macros.asm"
reset:LDSP  RAMEND

     ldi  r16,3
     rcallmul5 ; PC=0x[      ],ret-addr=0x[06  ],in=0x[03  ],out=0x[0F  ]
     rcallmul5 ; PC=0x[      ],ret-addr=0x[07  ],in=0x[0F  ],out=0x[4B  ]

     ldi  r16,14
     rcallmul5 ; PC=0x[      ],ret-addr=0x[09  ],in=0x[0E  ],out=0x[46  ]
     rjmp reset

; ============================================
; this sub-routine multiplies a register by 5
; in r16
; outr16
; modr17
mul5:
     ...
     ret
```

Figure 4.7: Utilisation d'appels aux sous-routines.

A quel endroit/adresse(s) peut on observer l'adresse de retour de sous-routine ? [dans la pile]

PREPARATION

4.2.2 MACRO

Ecrivez en Figure 4.8 la macro MUL5 qui multiplie un registre par cinq, en appliquant la même structure et les mêmes instructions que dans le cas de la sous-routine.

```
.macro MUL5; reg
        mov r17, @0
        ldi @0
        ldi @0
        add @0, r17
.endmacro
```

Figure 4.8: Macro MUL5.

La macro MUL5 peut être appliquée à tous les registres, sauf r17. Dans ce cas ce ne serait pas une multiplication par cinq qui serait effectuée, mais par [8].

4.2.3 COMPARAISON

Réécrivez le programme donné en Figure 4.7 en remplaçant les appels aux sous-routines par des invocations des macros.

Assemblez, puis comparez les codes désassemblés obtenus.

Comment se manifeste l'appel à une sous-routine dans le code désassemblé ?

[le programme fait appel à rcall qui gère le PC afin de sauter au bon endroit de la sous-routine.]

Comment se manifeste l'invocation d'une macro dans le code désassemblé ?

[le compilateur remplace le code de la macro dans le programme à chaque fois qu'elle est appelée]

La macro MUL5 regxx est remplacée par [4] instruction(s) qui durent [4] cycles(s) ou [1] us. L'appel de fonction mul5 introduit [3] cycles supplémentaires ([3] cycles pour l'instruction rcall et [4] cycles pour l'instruction ret).

Placez un point d'arrêt sur l'instruction rjmp reset. Comparez les temps d'exécution.

Le code faisant appel à des sous-routines nécessite [20] us.

Le code faisant appel à des macros nécessite [4,5] us.

4.3 INTRODUCTION AU LCD

4.3.1 INITIALISATION DU LCD

PREPARATION

Le LCD est contrôlé par deux registres: IR signifiant [instruction register], et DR signifiant [data register]. Ces deux registres sont adressés comme la mémoire externe. Le registre IR est situé à l'adresse 0x[8000] et le registre DR à l'adresse 0x[c000].

Pour accéder à la mémoire externe, il faut activer deux bits dans le registre MCUCR, signifiant [mcu general control Register]. Complétez et commentez en Figure 4.9 la suite de trois instructions nécessaires à activer la mémoire externe.

```
in   r16, MCUCR  ; sauvegarde MCUCR dans r16
sbr  r16, (1<<SRE)+(1<<SRW) ;set des bits 7 et 6
out  MCUCR, r16  ; place r16 dans MCUCR (avec les nouveaux bits mis)
```

Figure 4.9: Instructions d'activation de la mémoire externe.

Chargez le programme lcd1.asm donné en Figure 4.10. Il fait une initialisation de LCD.

```
; file  lcd1.asm
.include "m103def.inc"

.equ LCD_IR= 0x8000      ; address LCD instruction reg
.equ LCD_DR= 0xc000      ; address LCD data register

.macroLD_IR
a:   lds  r16,LCD_IR     ; read the SRAM (LCD IR) into r16
     sbrc r16,7          ; check the busy flag (bit7)
     rjmp a              ; jump back if busy flag set
     ldi  r16,@0         ; load value into r16
     sts  LCD_IR,r16; store value to SRAM (LCD IR)
     .endmacro

reset:
     in   r16,MCUCR; enable ext. SRAM access
     sbr  r16,(1<<SRE)+(1<<SRW)
     out  MCUCR,r16
main:
     LD_IR  0b00000001  ; clear display
     LD_IR  0b00000010  ; return home
     LD_IR  0b00000110  ; entry mode set
     LD_IR  0b00001111  ; display on/off control
loop:
     rjmp loop          ; infinite loop
```

Figure 4.10: lcd1.asm.

Ajoutez une ligne de code qui initialise le LCD pour deux lignes.

LD_IR 0b00111000

Ajoutez une ligne qui initialise le LCD pour 1 ligne mais utilise des grands caractères de 5x10 pixels

LD_IR 0b00110100

4.3.2 DÉPLACEMENT DE L'AFFICHAGE ET DU CURSEUR (FACULTATIF)

Téléchargez le programme lcd2.asm donné en Figure 4.11.

Comment faut-il modifier lcd2.asm afin que l'affichage apparaisse et défile sur la deuxième ligne (sans tenir compte des curseurs) ?

```
; file  lcd2.asm
.include "m103def.inc"        ;include AVR port/bit definitions
.include "macros.asm"         ;include macro definitions
.include "definitions.asm"    ;include register/constant definitions

reset:
     LDSP      RAMEND         ;set up stack pointer (SP)
     OUTI      DDRB,0xff      ;configure portB to output
     rcall     LCD_init       ;initialize LCD
     rcall     LCD_blink_on   ;turn blinking on
intro:
     ldi       a0,'A'         ;write the character 'A'
     rcall     lcd_putc
     ldi       a0,'V'         ;write the character 'V'
     rcall     lcd_putc
     ldi       a0,'R'         ;write the character 'R'
     rcall     lcd_putc
main:
     WAIT_MS 100
     CP0  PIND,0,LCD_home     ;CP0: Call if Port=0
     CP0  PIND,1,LCD_clear
     CP0  PIND,2,LCD_display_right
     CP0  PIND,3,LCD_display_left
     CP0  PIND,4,LCD_cursor_right
     CP0  PIND,5,LCD_cursor_left
     JP0  PIND,6,intro        ;JP0: Jump if Port=0
     rjmp main

; === include ===
.include "lcd.asm"
```

Figure 4.11: lcd2.asm.

Complétez en Figure 4.12 les sous-routines qui permettent de déplacer verticalement le curseur. La méthode à appliquer consiste à lire l'adresse courante du curseur, puis ajouter l'offset en agissant sur le bit qui indique la ligne 1 ou 2.

Associez les nouvelles sous-routines aux boutons PD2, PD3, et PD7. Complétez le code dans lcd2-2.asm et testez.

```
LCD_cursor_up:
     lds        w, LCD_IR       ; read AC (address counter)
     andi       w, 0x40         ; clear bit for line 1
     ori        w, 0x80         ; set bit (table 8-4 lecture notes, last line)
     sts        LCD_IR,w
     ret

LCD_cursor_down:
     lds        w, LCD_IR       ; read AC (address counter)
     ori        w, 0x40         ; set bit for line 2
     ori        w, 0x80         ; set bit
     sts        LCD_IR,w
     ret

LCD_cursor_toggle:
     lds        w, LCD_IR       ; read AC (address counter)
     subi       w,              ; add offset
     ori        w,              ; set bit
     sts        LCD_IR,w
     ret
```

Figure 4.12: Routines de déplacement vertical du curseur.

LABORATOIRE DE SYSTEMES MICROELECTRONIQUES

EPFL STI – IMM – LSM
ELD
Station n° 11
CH-1015 Lausanne

Téléphone : +4121 693 6955
Fax : +4121 693 6959
E-mail : lsm@epfl.ch
Site web : lsm.epfl.ch

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Microcontrôleurs 2012, MT-BA4: TP05-2012-v3.5.fm

v1.0	R. Holzer	I2S	2000-2003
v2.4.1	A. Schmid	LSM	Janvier 2006
v3.5	A. Schmid	LSM	Octobre 2011

MICROCONTRÔLEURS
TRAVAIL PRATIQUE NO 5

GROUPE A	Mercredi 28.03.2012, 08:00-10:00	GROUPE B	Lundi 26.03.2012, 11:00-13:00

No du Groupe	Premier Etudiant	Second Etudiant	Evaluation	Visa Correcteur
ME 08	Barteaux Jean-Bernard	Dorriere Timothée	5.4	

PREPARATION PREPARATION PREPARATION

5. OPÉRATIONS AVANCÉES AVEC L'AFFICHAGE LCD

Ce travail pratique voit l'étude de l'affichage sur le module LCD de chaînes de caractères, et de chaînes de caractères formattées, avec conversions. L'utilisation du LCD en mode pseudo-graphique est présenté en deuxième partie.

5.1 AFFICHACHE LCD ET CHAÎNES DE CARACTÈRES

5.1.1 AFFICHAGE D'UNE CHAÎNE DE CARACTÈRES

L'affichage de chaînes de caractères constantes et une fonction standard. La chaîne constante peut être placée en mémoire programme au moyen de la directive .db signifiant [définie boite]. Cette directive place le code ASCII dans la mémoire-programme. Pour accéder à cette constante on place une étiquette avant la chaîne pour donner un nom symbolique à son adresse. La chaîne est terminée par la valeur 0 pour indiquer sa fin. Par exemple :

```
string1 :
.db  "this is a string",0
```

Complétez la sous-routine LCD_putstring donné en Figure 5.1 qui affiche une chaîne constante au LCD.

A quelle adresse de la mémoire-programme se trouve le 'wo' de "world" ? [0x052].

```
; file  puts.asm                    ; display an ASCII string
.include "m103def.inc"
.include "macros.asm"
.include "definitions.asm"

reset:
     LDSP      RAMEND
     rcall     LCD_init
     rjmp      main
.include "lcd.asm"

str0:
.db  "hello world",0

main:
     ldi       r16,str0
     ldi       zl, low(2*str0)     ; load pointer to string
     ldi       zh,high(2*str0)
     rcall     LCD_putstring       ; display string
     rjmp      PC                  ; infinite loop

LCD_putstring:
; in z
     lpm                           ; load program memory into r0
     tst       r0                  ; test for end of string
     breq      donc
     mov       a0,r0               ; load argument
     rcall     LCD_putc
     adiw      zl,1                ; increase pointer address
     rjmp      LCD_putstring       ; restart until end of string
done:ret
```

Figure 5.1: puts.asm.

5.1.2 AFFICHAGE D'UN CHIFFRE AU FORMAT HEXADÉCIMAL

Complétez putx.asm donné en Figure 5.2 une fonction qui affiche une valeur hexadécimale. La valeur lue au format binaire sur les interrupteurs est affichée sur les LEDs, et est affichée en hexadécimal sur l'affichage LCD.

Les huit bits copiés des boutons-poussoirs sont affichés sous forme de deux caractères ASCII représentant chacun un nibble (demi-byte = 4 bits). Pour transformer un nibble en code ASCII, une table (lookup table) contenant les caractères possibles du code hexadécimal est utilisée. Les étapes à suivre sont les suivantes:

- Traitement des bits de poids fort (higher nibble):
  - extraction de la valeur du higher nibble par masquage;
  - attribution de l'offset obtenu au pointeur z;
  - chargement du caractère pointé par z dans la table située en mémoire programme;
  - affichage au LCD.
- Traitement du lower nibble de façon identique.

```
;file  putx.asm                    ; display a hex value
.include "m103def.inc"
.include "macros.asm"
.include "definitions.asm"

reset:
     LDSP      RAMEND              ; load stack pointer
     OUTI      DDRB,0xff           ; make portB output
     rcall     LCD_init            ; initialize LCD
     rjmp      main
.include "lcd.asm"

main:rcall     LCD_home            ; place cursor to home position
     in        a0,PIND             ; read switches
     out       PORTB,a0            ; write to LED
     com       a0                  ; invert a0
     rcall     LCD_putx            ; display in hex on LCD
     WAIT_MS   100
     rjmp      main

hextb:
.db  "0123456789abcdef"

LCD_putx:
; in  a0
     push      a0                            ; save
     swap      a0                            ; display high nibble first
     andi      a0,0x0f                       ; mask higher nibble
     mov       zl,a0                         ; load low byte of z
     clr       zh                            ; clear high byte of z
     subi      zl, low(-2*hextb)             ; add offset to table base
     sbci      zh,high(-2*hextb)
     lpm                                     ; look up ASCII code
     mov       a0,r0
     rcall     LCD_putc                      ; put character to LCD
     pop       a0
     andi      a0,0x0f
     mov       zl,a0                         ; load offset in low byte
     clr       zh                            ; clear high byte
     subi      zl,low(-2*hextb)              ; add offset to table base
     sbci      zh,high(-2*hextb)
     lpm                                     ; look up ASCII code
     mov       a0,r0
     rcall     LCD_putc                      ; put character to LCD
     ret
```

Figure 5.2: putx.asm.

5.1.3 AFFICHAGE D'UN CHIFFRE AU FORMAT BINAIRE

Complétez putb.asm donné en Figure 5.3 une routine qui affiche au format binaire la valeur lue en binaire sur les interrupteurs. La méthode proposée consiste à effectuer une boucle parcourue huit fois dans laquelle les opérations suivantes sont effectuées:

- décalage à gauche de la valeur lue sur les boutons-poussoirs;
- vérification de la valeur du carry; si ce dernier est actif, alors il faut afficher un '1' à la position courante du LCD sinon un zéro par défaut;
- affichage LCD.

```
; file  putb.asm                    ; display a binary value
.include "m103def.inc"
.include "macros.asm"
.include "definitions.asm"

reset:
     LDSP      RAMEND               ; load stack pointer
     OUTI      DDRB,0xff            ; make portB output
     rcall     LCD_init             ; initialize LCD
     rjmp      main
.include "lcd.asm"

main:rcall     LCD_clear            ; place cursor to home position
     in        a0,PIND              ; read switches
     out       PORTB,a0             ; write to LED
     com       a0                   ; invert a0
     rcall     LCD_putb             ; display in binary on LCD
     WAIT_MS   100
     rjmp      main

LCD_putb:
; in  a0
     mov       a1,a0                ; move argument to different register
     ldi       a2,8                 ; load counter
loop:ldi       a0,'0'               ; load '0'
     lsl       a1                   ; shift bit into carry
     brcc      PC+2
     ldi       a0,'1'               ; load '1' into register to be displayed
     rcall     LCD_putc             ; put character to LCD
     dec       a2                   ; decrement counter
     brne      loop
     ret
```

Figure 5.3: putb.asm.

5.1.4 AFFICHAGE DE CHAÎNES FORMATTÉES

Utilisez la librairie printf.asm pour afficher au LCD la valeur lue sur les boutons-poussoirs dans les différents formats demandés. Aidez-vous du code useprintf.asm dans lequel vous remplacez la ligne commentée par la ligne de code d'impression formattée adéquate.

- Affichage de la valeur lue sur les boutons-poussoirs en décimal signé.
  - quel est la ligne de code ? .db "b=", DEC:SIGN, a, 0.
- Affichage de la valeur lue en binaire non-signé sur les boutons-poussoirs dans le format fractionaire suivant:
  - higher nibble: partie entière;
  - lower nibble: partie fractionaire.
  - Quel est la plage possible ? 0 – 15,9375 par pas de 0,0625;
  - quel est la ligne de code ? .db "a=", HEX, a, 4, $14, 0.

5.2 LCD EN MODE PSEUDO-GRAPHIQUE (FACULTATIF)

Il est possible d'utiliser le LCD en mode pseudo-graphique, c'est-à-dire d'effectuer l'affichage d'un pixel particulier dans la matrice de 16x2 caractères. De la place mémoire dans le générateur de caractères a été réservée à cet effet à hauteur de 64bytes (CGRAM), soit huit caractères.

Le programme animation1.asm donné en Figure 5.4 est basé sur animation0.asm donné dans les notes de cours. Il permet l'affichage d'un effet de scrolling sur un caractère en forme de flèche vers le haut, ou vers le bas si un bouton poussoir est activé.

Etudiez la première partie de ce programme (sous-routine LCD_drCGRAMupw), et complétez ci-dessous:

- L'adresse du caractère reprogrammé dans la CGRAM est [ ].
- Adaptez sur la Figure 5.5 le schéma de principe donné dans les notes de cours, présentant les positions des pointeurs, et offset au programme dans le cas ou la flèche défile du bas vers le haut.
- Que contiennent r18, r22, et r24 ? [ ].
- Sur quel bouton-poussoir faut-il presser afin de changer le sens de défilement de la flèche ? [ ].

Complétez la sous-routine LCD_drCGRAMdnw afin qu'elle réalise l'affichage de la flèche défilant vers le bas, tête vers le bas, sans changer la table contenant la définition de la flèche. Pour cela vous devez reconsidérer les positions de la mémoire pointées, ainsi que les valeurs de l'offset. Evaluez soigneusement dans quel sens doit évoluer le pointeur z afin que la flèche défile vers le bas, ainsi que dans quel sens il faut parcourrir la table arrow0 afin que la tête de la flèche soit orientée vers le bas. Remplissez la Figure 5.6 afin de vous aider dans cette tâche.

```
; file animation1.asm                    ;display a character scrolling animation
.include "m103def.inc"
.include "macros.asm"
.include "definitions.asm"

reset:
      LDSP  RAMEND
      rcall LCD_init
      rjmp  main
.include "lcd.asm"
.include "printf.asm"

str0:
.db   0x03,0
arrow0:
.db
0b00000100,0b00001110,0b00011111,0b00000100,0b00000100,0b00000100,0b00000100,0b00000000

main:
      ldi   r22,8

   prog0:
      in    r25,PIND
      com   r25
      tst   r25
      breq  _dnm

      rcall       LCD_home
      PRINTF      LCD
.db   "  Arrowhead down ",0
      rcall       LCD_drCGRAMdnw            ;load animation arrowhead downwards
      rjmp  _gon

  _dnm:rcallLCD_home
      PRINTFLCD
.db   "  Arrowhead up  ",0,0
      rcall       LCD_drCGRAMupw            ;load animation arrowhead upwards


 _gon:ldi   r16,str0                        ;load text, including animated character
      ldi   z1, low(2*str0)                 ;load pointer to string
      ldi   zh,high(2*str0)
      rcall LCD_putstring                   ;display string
      WAIT_MS200                            ;wait
      dec   r22                             ;decrement offset
      _BRNE prog0                           ;animated sequence steps not completed
      rjmp  main                            ;infinite loop

LCD_putstring:
; in  z
      lpm                                   ;load program memory into r0
      tst   r0                              ;test for end
      breq  done
      mov   a0,r0                           ;load argument
      rcall LCD_putc
      adiw  z1,1
      rjmp  LCD_putstring
done: ret
```

Figure 5.4: animation1.asm.

```
LCD_drCGRAMupw:
      lds   u, LCD_IR                  ;read IR to check busy flag  (bit7)
      JBl   u,7,LCD_drCGRAMupw         ;Jump if Bit=1 (still busy)
      ldi   r16, 0b01011000            ;2MSBs:write into CGRAM(instruction),
                                       ;6LSBs:address in CGRAM and in charact.
      sts   LCD_IR, r16                ;store w in IR
      ldi   zl,low(2*arrow0)+8
      ldi   zh,high(2*arrow0)
      mov   r23,zl                     ;upper address limit
      dec   r23
      mov   r24,r23                    ;store upper limit of character in memory
      ldi   r18,8                      ;load size of caracter in table arrow0
      sub   zl,r22                     ;subtract current value of moving offset

   loop01:
      lds   u, LCD_IR                  ;read IR to check busy flag  (bit7)
      JBl   u,7,loop01                 ;Jump if Bit=1 (still busy)
      lpm                              ;load from z into r0
      mov   r16,r0
      adiw  zl,1
      mov   r23,r24                    ;garantee z remains in character memory
      sub   r23,zl                     ;zone, if not then restart at the begining
      brge  _reg1                      ;subtract current value of moving offset
      subi  zl,8

  _reg1:stsLCD_DR, r16                 ;load definition of one charecter line
      dec   r18
      brne  loop01
      rcall LCD_home                   ;leaving CGRAM
      ret

LCD_drCGRAMdnw:
      lds   u, LCD_IR
      JBl   u,7,LCD_drCGRAMdnw
      ldi   r16, [redacted]
      sts   [redacted]
      ldi   zl,low(2*arrow0)+8
      ldi   zh,high(2*arrow0)
      mov   r23,zl
      [redacted]r23,9
      mov   r24,r23
      ldi   r18,8
      sub   zl,r22

   loop02:
      lds   u, LCD_IR
      JBl   u,7,loop02
      lpm
      mov   r16,r0
      [redacted]zl
      mov   r23,r24
      sub   r23,[redacted]
      [redacted]_reg2
      [redacted]zl,8

  _reg2:stsLCD_DR, r16
      dec   r18
      brne  [redacted]
      rcall LCD_home
      ret
```

Figure 5.4: animation1.asm.

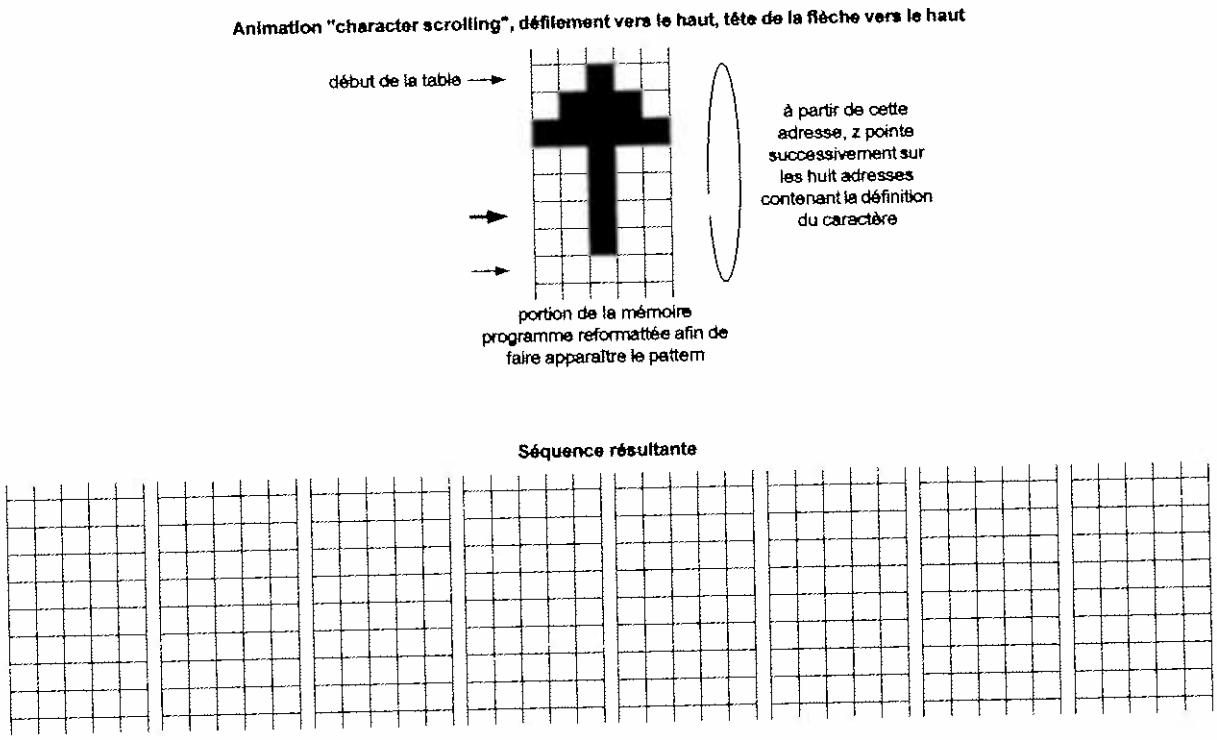


Figure 5.5: Animation générée par LCD_drCGRAMupw.

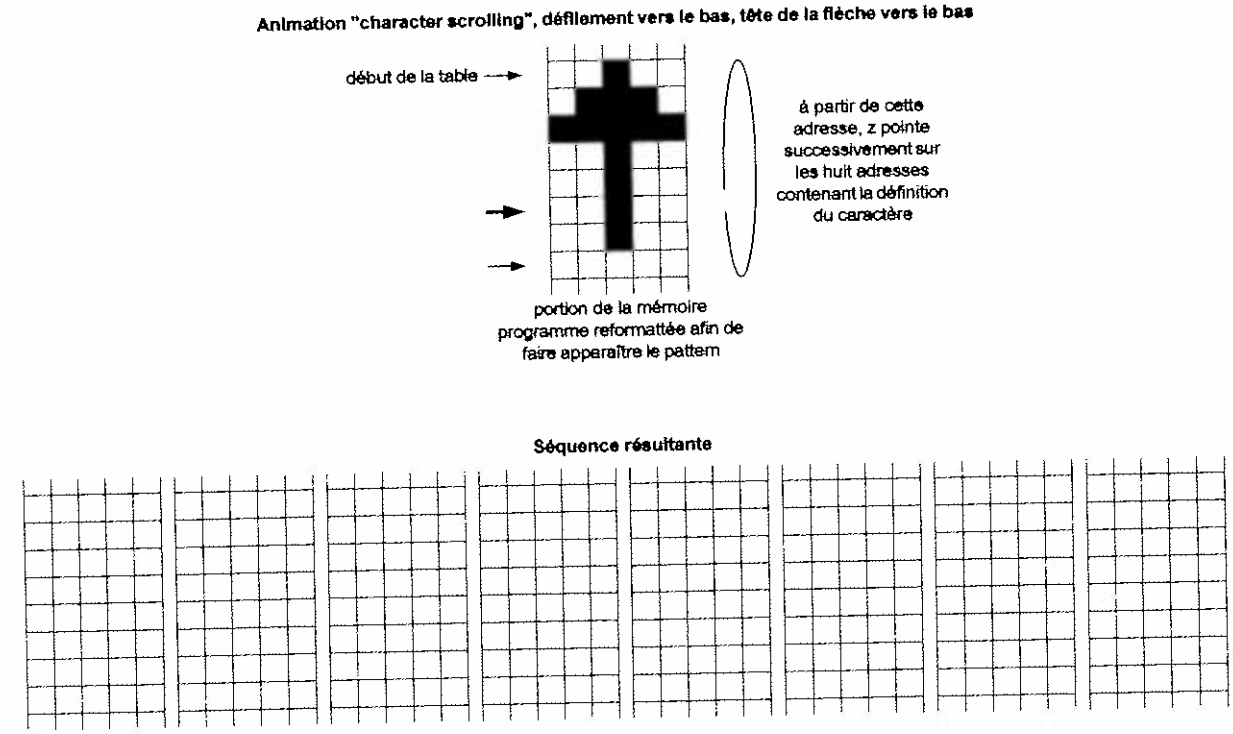


Figure 5.6: Animation générée par LCD_drCGRAMdnw.

LABORATOIRE DE SYSTEMES MICROELECTRONIQUES

EPFL STI – IMM – LSM
ELD
Station n° 11
CH-1015 Lausanne

Téléphone : +4121 693 6955
Fax : +4121 693 6959
E-mail : lsm@epfl.ch
Site web : lsm.epfl.ch

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Microcontrôleurs 2012, MT-BA4: TP06-2012-v3.5.fm

v1.0	R. Holzer	I2S	2000-2003
v2.4.1	A. Schmid	LSM	Janvier 2006
v3.5	A. Schmid	LSM	Octobre 2011

56 → 1
64

MICROCONTRÔLEURS
TRAVAIL PRATIQUE NO 6

GROUPE A	Mercredi 04.04.2012, 08:00-10:00	GROUPE B	Lundi 02.04.2012, 11:00-13:00

No du Groupe	Premier Etudiant	Second Etudiant	Evaluation	Visa Correcteur
ME 8	Demierre Timothee	Barleaux Jean-Lionel		

6. INTERRUPTIONS

Les interruptions sont le mécanisme dont dispose un microcontrôleur lui permettant de réagir rapidement à des événements externes ou internes. Le but de ce travail pratique est d'acquérir la connaissance sur les différents interruptions disponible sur ATmega103, leur programmation, et leur déroulement.

6.1 LE VECTEUR D'INTERRUPTION, PRIORITÉ DES INTERRUPTIONS

Les quatre lignes d'interruptions externes $\overline{INT0}$...$\overline{INT3}$ (barre car [niveau bas]) se trouvent sur les pins [0] à [3] du port [D]. Les interruptions INT4...INT7 se trouvent sur les pins [4] à [7] du port [E]. A ces dernières est associé un registre de contrôle nommé [EICR] qui permet d'en programmer la sensibilité.

Deux étapes sont nécessaires à l'autorisation d'une interruption:

- le bit correspondant à l'instruction doit être activé dans le registre nommé EIMSK, signifiant [External Interrupt Mask Register], et qui se trouve à l'adresse [$ 39];
- les interruptions doivent être globalement autorisées en mettant le [I] dans SREG à 1 avec l'instruction [SEI] signifiant [Global Interrupt Enable].

Si les interruptions INT0...INT3 sont autorisées, un niveau logique '0' sur une de ces lignes force le microcontrôleur à suspendre le programme principal courant et à sauter à une routine spéciale d'asservissement de l'interruption (en anglais ISR signifiant [Interrupt Service Routine]).

Le microcontrôleur répond à une interruption de façon autonome par l'exécution des tâches suivantes, chronologiquement:

1.) [l'exécution de l'instruction courante est stoppée. Non justement pas !!!]
2.) [les interruptions sont désactivées. Le PC est sauvé sur la pile et le]
3.) [programme saute au vecteur d'interruption ✓ jmp à l'instruction routine ✓]
4.) [service de la routine ✓]
5.) [lorsque rti est détecté, les interruptions sont activées et le PC est retrouvé de la pile. ✓]

PREPARATION PREPARATION PREPARATION

Chargez le fichier int0.asm en Figure 6.1 dans Studio, compilez-le et téléchargez-le.

```
; file    int0.asm  ; using INT0..INT3
.include "m103def.inc"       ; include definition file
.include "macros.asm"        ; include macros definitions
.include "definitions.asm"   ; include register definitions

; === interrupt table ===
.org 0
     jmp  reset

.org INT0addr
     jmp  ext_int0

.org INT1addr
     jmp  ext_int1

.org INT2addr
     jmp  ext_int2

.org INT3addr
     jmp  ext_int3

; === interrupt service routines
ext_int0:
     cbi  PORTB,0   ; turn on LED 0
     reti
ext_int1:
     cbi  PORTB,1   ; turn on LED 1
     reti
ext_int2:
     cbi  PORTB,2   ; turn on LED 2
     reti
ext_int3:
     cbi  PORTB,3   ; turn on LED 3
     reti

; === initialization (reset) ====
reset:
     LDSP RAMEND              ; load stack pointer SP
     OUTI DDRB, 0xFF          ; portB = output
     OUTI EIMSK,0b00001111    ; enable INT0..INT3;
     sei                      ; set global interrupt

; === main program ===
main:
     WAIT_US 10000            ; wait 10 msec
     dec  r18                 ; decrement counter
     out  PORTB,r18           ; output counter value to LED
     rjmp main
```

Figure 6.1: int0.asm

Sans appuyer sur les boutons PD0...PD3 le programme exécute uniquement le partie principale avec l'étiquette [main]. Lorsqu'un des interrupteurs est activé avec persistance, le programme principal est constamment interrompu et doit exécuter la routine d'interruption correspondante de façon répétée.

Que se passe-t'il si plusieurs interruptions sont actives simultanément; y-a-t'il une forme de priorité?

Testez-le au moyen de la carte STK300 !

Réponse: [ceux ! Les interruptions avec un chiffre plus faible sont prioritaires sur les autres ! (INT0 > INT1)]

6.2 TABLE DES VECTEURS D'INTERRUPTIONS

A quoi sert la ligne .org INTXaddr ?

[Lorsque le microcontrôleur détecte une interruption, le PC saute à cette adresse pour ensuite pouvoir sauter à la routine de service.]

Mettez les instructions de saut des interruptions (INT3, INT5, et UART_RXC) à la bonne adresse dans la table des vecteurs d'interruption, au début du programme en Figure 6.2. Ecrivez les adresses en hexadécimal.

```
; === interrupt table ===
.org [0x000]
     jmp  reset

.org [0x006]
     jmp  ext_int2

.org [0x010]
     jmp  ext_int7

.org [0x014]
     jmp  UART_RXC
```

Figure 6.2: Portion de la table d'interruption du ATmega103.

Dans quel document peut-on trouver cet information ?

[mémo. de l]

Ces adresses correspondent'elles à une implémentation matérielle ou logicielle ?

[Implémentation matérielle]

6.3 AUTORISATION DES INTERRUPTIONS

Ecrivez le bout de code qui autorise les interruptions INT0, INT2, INT5, INT6 à l'exclusion des autres.

```
ldi     r16,0b[01100101]
out     [EIMSK],r16
```

Ecrivez le bout de code qui autorise les interruptions INT0, INT1, INT5, INT7 sans modifier l'état des autres.

```
in      r16,[EIMSK]
ori     r16,[0b10100011]
out     EIMSK,[r16]
```

Ecrivez le bout de code qui bloque les interruptions INT1, INT3 sans modifier l'état des autres.

```
in      r16,EIMSK
andi    r16,0xF5
out     EIMSK,r16
```

6.4 CONDITIONS DES INTERRUPTIONS

Ecrivez le bout de code qui autorise les interruptions INT5, INT6, INT7, et qui configure INT5 pour un flanc montant, INT6 pour un flanc descendant et INT7 pour un niveau 'low'.

```
ldi     r16,0b11100000
out     EIMSK,r16       ; external interrupt mask
ldi     r16,0b00010100
out     EICR,r16        ; external interrupt control register
```

Ecrivez le bout de code qui autorise les interruptions INT4, INT5, et qui configure INT4 pour un flanc montant, INT5 pour un flanc descendant et qui laisse les autres interruptions déjà configurées dans leur état.

```
in      r16,EIMSK
ori     r16,0b00110000
out     EIMSK,r16       ; set external interrupt mask
in      r16, EICR
andi    r16,0b11111011  ; set the 0s
ori     r16,0b00001011  ; set the 1s
out     EICR,r16; external interrupt control register
```

6.5 SIMULATION DES INTERRUPTIONS EXTERNES

Il est possible de simuler les interruptions externes et en observer les effets sur le déroulement du programme. Chargez le programme int0b.asm en Figure 6.3 et simulez-le (ne le téléchargez pas !). Vérifiez les points suivants, en simulation pas à pas:

1.) Vérifiez que le programme effectue une seule instruction du programme principal, puis saute dans la routine d'interruption. Quelle est la raison de ce comportement ? Quelle(s) interruption(s) est(sont) active(s) ?

Il saute toujours dans l'interruption car 0. Toutes les interruptions sont activées (niveau haut, mais il y a est prioritaire

2.) Mettez le bit I dans SREG au niveau logique '0' afin de bloquer toutes les interruptions. Que constatez-vous ?

le programme reste tout le temps dans le programme principal

3.) Remettez le bit I au niveau logique '1'. Mettez le bit de contrôle INT0 dans EIMSK au niveau logique '0' et vérifiez que l'interruption 0 est bloquée. Quelle interruption est alors servie ? Int 1

4.) Mettez PD0 à PD3 successivement au niveau logique '1' et observez le changement de comportement dans le déroulement du programme.

5.) L'interruption INT4 est déclenchée par une transition du niveau logique '1' à '0' sur la ligne PE4. Vérifiez ce comportement en simulant en mode "Auto Step" et changeant les valeurs sur PE4.

6.) L'interruption INT5 est déclenchée par une transition du niveau logique '0' à '1' sur la ligne PE5. Vérifiez ce comportement de même.

```
; file int0b.asm    ; using INT0..INT3
.include "m103def.inc"  ; include definition file
.include "macros.asm"   ; include macros definitions

; === interrupt table ===
     jmp  reset
     jmp  int_0; PIND0..3
     jmp  int_1
     jmp  int_2
     jmp  int_3
     jmp  int_4; PINE4..7
     jmp  int_5
     jmp  int_6
     jmp  int_7

; === interrupt service routines
int_0:inc r0
     reti
int_1:reti
int_2:reti
int_3:reti
int_4:inc r1
     reti
int_5:reti
int_6:reti
int_7:reti

; === initialization (reset) ====
reset:LDSPRAMEND; load SP
     OUTI EIMSK,0b00111111 ; INT0..4
     OUTI EICR, 0b00001110 ; 00=low, 10=fall, 11=rise
     sei  ; set global interrupt

; === main program ===
main:inc  r16
     nop
     rjmp main
```

Figure 6.3: int0b.asm.

6.6 OBERVATION DES DIAGRAMES DES TEMPS (TIMINGS) À L'OSCILLOSCOPE

Il est possible d'observer les différents délais associés à l'exécution d'une interruption, ainsi que le déroulement de l'exécution au moyen de l'oscilloscope. Le programme int1.asm donné en Figure 6.4 permet cette observation car il adresse un port en sortie dans le programme principal, et un autre port en sortie dans la routine d'asservissement de l'interruption. Il suffit alors d'observer l'état des ports, et d'associer les timings obtenus aux instructions exécutées. C'est le but de l'exercice proposé.

```
; fileint1.asm            ; using INT0..INT3
.include "m103def.inc"    ; include definition file
.include "macros.asm"     ; include macros definitions

; === interrupt table ===
.org 0
     jmp  reset

.org INT3addr
     jmp  ext_int3

; === interrupt service routines
ext_int3:
     sbi  PORTA,5              ; pulse on PINA5
     cbi  PORTA,5
     reti

; === initialization (reset) ====
reset:
     LDSP RAMEND               ; load stack pointer SP
     OUTI DDRA, 0b00100010     ; PA1,PA5 = output
     OUTI EIMSK,0b00001000     ; enable INT3;
     sei                       ; set global interrupt

; === main program ===
main:
     sbi  PORTA,1              ; pulse on PINA1
     cbi  PORTA,1
     rjmp main
```

Figure 6.4: int1.asm.

6.6.1 PRÉPARATION DU MONTAGE

1.) Connectez la sonde 1 de l'oscilloscope à la pin PA1. Dans quel document trouvez-vous le plan de connection ?

Stauer kit (Orcad Guide Schéma (polycop)

2.) Connectez la sonde 2 de l'oscilloscope à la pin PA5.
3.) Placez le diviseur sur chaque sonde sur la position 10X.
4.) Configurez l'oscilloscope avec les valeurs données au bas de la copie d'écran en et , soit [5.00V/div, 500ns/div] et [5.00V/div, 1us/div] respectivement.
5.) Réglez la position verticale (VERTICAL POSITION) des la trace comme indiqué sur les Figure 6.5 et Figure 6.6. Ainsi, le niveau de la masse est donné par les positions respectives des flèches pour chaque canal: 1→, 2→.
6.) Configurez les paramètres d'aquisition des canaux dans les menus CH1 et CH2 comme suit:
   - sélectionnez Coupling DC;
   - sélectionnez BW (Bandwith) Limit OFF;
   - sélectionnez Volts/Div Coarse;
   - sélectionnez Probe 10X;
   - sélectionnez Invert OFF.
7.) Configurez les paramètres du trigger dans le menu TRIGGER comme suit:

- sélectionnez Edge (trigger sur des flancs);
- sélectionnez Rising (trigger sur un flanc montant);
- sélectionnez Source CH1 (trigger sur le canal 1, pin PA1 contrôlée par le programme principal);
- sélectionnez Mode Normal et placez le niveau du trigger TRIGGER LEVEL sur 2.5V (au milieu de la gamme, donné par la flèche ←);
- sélectionnez Coupling DC.

6.6.2 MANIPULATIONS

Téléchargez int1.asm dans le microcontrôleur. Observez les signaux générés en l'absence d'interruption.

Dessinez sur la Figure 6.5 les deux traces que vous observez à l'oscilloscope. Associez aux traces visibles les instructions suivantes du programme principal (main):

- sbi PORTA,1
- cbi PORTA,1
- rjmp main

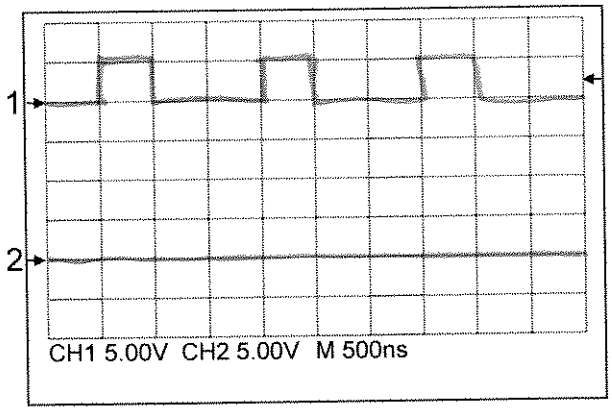


Figure 6.5: Traces visibles à l'oscilloscope: sans interruption active.

Appuyez sur le bouton PD3 avec persistance afin d'activer INT3. L'exécution du programme alterne alors entre le programme principal, et la routine d'asservissement de l'interruption. Considérant le code software qui contrôle la génération de signaux, la trace 1, correspondant à la pin PA1 est contrôlée par le programme principal alors que la trace 2, correspondant à la pin PA5 est contrôlée par la routine d'asservissement.

Modifiez la base de temps à 1.0us, provoquez l'interruption et maintenez-la, puis au moyen de la mollette HORZONTAL POSITION, déplacez la trace 1 de façon à centrer le pulse obtenu comme indiqué en Figure 6.6. Reportez la trace 2. Associez les instructions suivantes au diagramme temporel obtenu, et indiquez à chaque fois le nombre de cycles, en vérifiant bien d'avoir le total exact de cycles nécessaires correspondant à la trace visible:

- sbi PORTA,1
- saut au vecteur d'interruption
- jmp ext_int3
- sbi PORTA,5
- cbi PORTA,5
- reti
- cbi PORTA,1

instructions ?

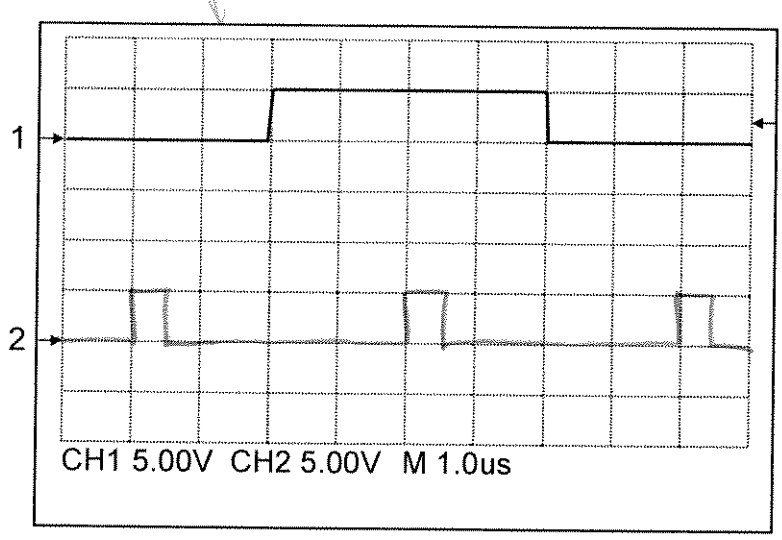


Figure 6.6: Traces visibles à l'oscilloscope: avec interruption active.

Dans le cas où il n'y a pas d'interruption, le pulse sur la trace 1 dure [2] cycle(s); alors que dans le cas où il y a une interruption (maintenue), le pulse sur la trace 1 dure [16] cycles. Sachant que l'instruction cbi prend deux cycles, cela signifie-t'il qu'une interruption a été acceptée à chacun de ces deux cycles ? Commentez !

Non ! Il faut terminer une interruption pour pouvoir en prendre une autre !

Oui mais tu ne réponds pas à la question.

LABORATOIRE DE SYSTEMES MICROELECTRONIQUES

EPFL STI – IMM – LSM
ELD
Station n° 11
CH-1015 Lausanne

Téléphone : +4121 693 6955
Fax : +4121 693 6959
E-mail : lsm@epfl.ch
Site web : lsm.epfl.ch

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Microcontrôleurs 2012, MT-BA4: TP07-2012-v3.5.fm

v1.0	R. Holzer	I2S	2000-2003
v2.4.1	A. Schmid	LSM	Janvier 2006
v3.5	A. Schmid	LSM	Octobre 2011

MICROCONTRÔLEURS
TRAVAIL PRATIQUE NO 7

GROUPE A	Mercredi 18.04.2012, 08:00-10:00	GROUPE B	Lundi 16.04.2012, 11:00-13:00

No du Groupe	Premier Etudiant	Second Etudiant	Evaluation	Visa Correcteur
ME 8	Domingos Ginolhac	Rebaux Jean-Bernard	1	S. Allemand

PREPARATION

PREPARATION

7. TIMERS ET COMPTEURS

Les timers liés à leurs compteurs internes permettent au microcontrôleur de se référencer par rapport à une base de temps externe, et ainsi d'ajuster avec grande précision le déclenchement d'opérations par un mécanisme d'interruptions causées par l'overflow des timers. Ce travail pratique aborde différents aspects des timers et propose un exemple d'utilisation pour le contrôle d'un moteur pas-à-pas.

7.1 INTERRUPTION PAR UN TIMER OVERFLOW

Chargez le programme tim0_ov-1.asm dans Studio. Téléchargez-le sur la carte.

- Le bit AS0 est mis à 1, donc l'entrée pour le timer provient [du quartz horloger ✓] avec une fréquence de [32'768] Hz.
- Le prescaler est à 2. La période d'interruption est donné par la formule T= [256 · 2 / f] = [0,0625] sec.

Connectez la sonde 1 de l'oscilloscope à la ligne PB1 et la sonde 2 à la ligne PB7 et étudiez les signaux au moyen de la fonction MEASURE→CH1–Period, MEASURE→CH2–Period. La période du signal observé est de [125] ms sur PB1 et [500] ms sur PB7. Le nombre de "timer overflows" par période est de [2].

Complétez les instructions ci-dessous nécessaires à configurer le prédiviseur du timer à CK/128.

```
ldi  r16, 0b[00000101]
out  [TCCR0], r16
```

Effectuez le changement, téléchargez le programme. La période d'interruption est [2000] ms.

7.2 LE PRESCALER

Téléchargez le programme timer0_prescaler.asm en Figure 7.1. Placez le module avec le haut-parleur bleu sur le PORTE. Placez une sonde de l'oscilloscope sur PD7, pour mesurer les périodes des "timer0 overflows." Entrez les valeurs 000 à 111 avec les boutons poussoirs et remplissez les cases en Table 7.1.

```
; file    timer0_prescaler.asm    ; timer 0 overflow
.include "m103def.inc"            ; include definition file
.include "macros.asm"             ; include macro definitions
.include "definitions.asm"

; === interrupt vector table ===
.org 0
     rjmp reset                   ; reset

.org OVF0addr                     ; timer0 overflow interrupt
     rjmp ovf0

; === interrupt service routines ====
.org 0x30
ovf0:INVP      PORTE,SPEAKER      ; make a sound
     INVP      PORTD,7            ; oscilloscope probe
     reti

.include "lcd.asm"
.include "printf.asm"

; === reset ===
reset:
     LDSP      RAMEND             ; load stack pointer (SP)
     OUTI      DDRB,0xff          ; LEDs = output
     sbi       DDRE,SPEAKER       ; speaker = output
     sbi       DDRD,7             ; oscilloscope probe = output
     rcall     LCD_init           ; initialize LCD

     OUTI TIMSK,1<<TOIE0          ; Timer0 Overflow Interrupt Enable
     sei                          ; set global interrupt

; === main program ===
main:
     in        r20,PIND           ; read buttons
     out       PORTB,r20          ; write LEDs
     com       r20                ; invert register
     out       TCCR0,r20          ; write Timer Control Reg

     rcall     LCD_clear          ; clear LCD
     PRINTF    LCD                ; display formatted string
.db  "TCCR0=",BIN,TCCR0+0x20,0

     WAIT_MS   100
     rjmp      main
```

Figure 7.1: timer0_prescaler.asm

TCCR	prescaler/fonction	période (*)
000	0	0
001	1	128 µs
101	1024	16.4 ms
110	256	32.8 ms
111	1024	131.1 ms

Table 7.1: Etude du prescaler.

(*) La période reportée ici est la période observée sur PD7, elle est égale au double de la période du timer car la valeur dans PD7 est inversée à chaque timer overflow.

7.3 MULTIPLES INTERRUPTIONS EN PARALLÈLE

Téléchargez le programme timer_ov-1.asm. Observez les trois signaux générés par les interruptions ainsi que le signal créé par le programme principal au moyen de l'oscilloscope. Mesurez la fréquence et la période avec la fonction MEASURE→CH1–Period/Freq, et reportez les résultat sur la Table 7.2.

Source	Pin	Fréquence	période (*)
timer0	PB1	61 kHz	16.64 µs
timer1	PB3	0.496 Hz	2.1
timer2	PB5	976 Hz	1.024 ms
programme principal	PB7	4.953 kHz	201.9 ms

Table 7.2: Etude du prescaler.

Travaillez maintenant avec le simulateur Studio. Placez un breakpoint sur la première instruction de l'interruption numéro 3 (rjmp overflow2). Simulez sans arrêt jusqu'à ce breakpoint par Go/Run (F5). Utilisez le chronomètre et indiquez le temps passé entre les interruptions du timer 2: 512.75 µs.

Effectuez un "toggle source/mixed mode ou CTRL-F11," puis continuez la simulation en "auto-step par Debug→auto-step." Vous observez la simulation de l'exécution du code assemblé, à vitesse relativement lente.

Observez la fenêtre des I/Os par view→new I/O view. Le registre TCNT2 signifiant Timer/Counter 2 et situé à l'adresse 0x24(0x44-SRAM) est incrémenté à chaque coup d'horloge. Une interruption est déclanchée lorsque ce registre passe par la valeur 0xFF.

Combien de cycles peut-on compter dans l'intervalle entre deux interrutions du timer 2 ? 1024 cycles.

Effectuez cette même manipulation plusieurs fois. Pourquoi l'interruption est-elle parfois retardée d'un cycle ?

Parce que pour sauver dans une interruption il faut attendre que l'instruction dans mul soit finie et elles ne font pas toute le même nombre de cycle.

7.4 INTERRUPTIONS À DES INTERVALLES DÉTERMINÉS

Il est possible de forcer les timers à générer des overflow interrupts à des intervalles précis et non déterminés par la division de l'horloge effectuée par le prescaler. Pour cela il est nécessaire de recharger une valeur choisie dans le registre compteur du timer nommé [TCNT] à chaque overflow interrupt afin de garantir le nombre de coups d'horloge restant avant le prochain overflow interrupt.

Téléchargez le programme timer_ov1.asm. Vérifiez les timeouts au moyen de l'oscilloscope.

Modifiez le programme afin d'observer des timeouts à 11msec (timer0), et 3ms (timer1). Pour cela, remplacez les constantes symboliques données en Figure 7.2

```
.set    timer0 = 100
.set    timer1 = 2000
```

Figure 7.2: timer_ov1.asm, constantes symboliques à remplacer.

par des expressions qui calculent automatiquement la bonne valeur pour les timers, en Figure 7.3 (pensez aussi à modifier la programmation du prescaler dans la section reset: en accord avec la valeur donnée à la constante symbolique prescalerx ci-dessous).

```
.set    clock0 = [32768]          ; in Hz
.set    clock1 = [6000]           ; in kHz
.set    prescaler0 = [8]          ; 1..1024
.set    prescaler1 = [1]
.set    timeout0 = [11]           ; in msec
.set    timeout1 = [3000]         ; in usec
    ;dans les deux cas suivants, il faut placer une formule
.set    timer0 = [clock0/prescaler0 * timeout0/1000]
.set    timer1 = [clock1/prescaler1 * timeout1/1000]
```

clock/pres . timer = 1/timeout

timer = clock . timeout / pres

Figure 7.3: timer_ov1.asm, code à insérer afin de générer les timeouts désirés.

L'ordre des termes pour le calcul automatique des timerx est important, parce que l'assembleur résout les expressions mathématiques en utilisant des entiers 4-byte. Ainsi la limite supérieure est 2^{32}=4'294'967'296. L'ordre des divisions/multiplications est important car les résultats intermédiaires ne doivent pas dépasser cette limite; de plus, lors de divisions (entiers) la partie fractionnaire est perdue.

Suivant la solutions que vous avez choisie, quelle erreur obtenez-vous sur le timeout ?

- timeout0 désiré = 11msec; timout0 réel synthétisé = [10,98] ms;
- timeout0 désiré = 3msec; timout0 réel synthétisé = [3] ms.

Vérifiez les timings à l'aide de l'oscilloscope.

7.5 GÉNÉRATION DE SIGNAL RECTANGULAIRE AVEC FRÉQUENCE VARIABLE

Télécharger le programme pulsout4.asm donné en Figure 7.4.

Observez les signaux à l'oscilloscope.

Appuyer sur les boutons PD0 et PD1 sert à faire varier [la fréquence du haut-parleur] en agissant sur la valeur stockée dans le registre interne [OCR1] qui stocke [la valeur de l'output compare register qui est comparé au timer].

```
; file      pulsout4.asm
; generation of rectangular signal using timer2
.include "m103def.inc"              ; include definitions
.include "macros.asm"               ; include macro definitions
.include "definitions.asm"

; === interrupt vector table ===
      rjmp        reset
.org        OC2addr
      rjmp        oc2

; === interrupt routines ===
oc2:        INVP        PORTE,SPEAKER     ; make a sound
            reti

; === initialization ===
reset:
      LDSP        RAMEND                  ; load the stack pointer
      OUTI        DDRB,0xff               ; make portB all output
      sbi         DDRE,SPEAKER            ; make speaker an output
      OUTI        TCCR2,0b00011001; CS2=001 (CK), COM=01 (toggle) CTC=1 (clear)
      rcall       LCD_init

      ldi         b0,10                   ; preset OCR2
      ldi         a1,4                    ; preset TCCR2

      OUTI        TIMSK,1<<OCIE2
      sei
      rjmp        main
.include "lcd.asm"
.include "printf.asm"

main: in    r0, PIND                ; copy buttons to LED
      out   PORTB,r0

      out   OCR2,b0                 ; set output compare register

      in    w,TCCR2
      andi  w,0b11111000
      add   w,a1
      out   TCCR2,w

      rcall       LCD_clear         ; set cursor to home position
      PRINTF      LCD
.db   "CS2=",HEX,a+1," OCR2=",HEX,b,0
      WAIT_MS100        ; wait 100msec

loop: JP0   PIND,0,incremb          ; jump if pin=0, check the buttons
      JP0   PIND,1,decremb
      JP0   PIND,2,increma
      JP0   PIND,3,decrema
      rjmp  loop                    ; jump back
incremb:
      INC_CYC     b0,10,250
      rjmp        main
decremb:
      DEC_CYC     b0,10,250
      rjmp        main
increma:
      INC_CYC     a1,2,5
      rjmp        main
decrema:
      DEC_CYC     a1,2,5
      rjmp        main
```

Figure 7.4: pulsout4.asm.

Appuyer sur les boutons PD2 et PD3 sert à faire varier la fréquence du non produit par le haut-parleur en agissant sur la valeur stockée dans TCCR1 que correspond à la valeur du prescaler.

7.6 CONTRÔLE DU MOTEUR PAS-À-PAS

Un moteur pas-à-pas de faible puissance peut être contrôlé directement par les ports de sortie d'un microcontrôleur. Le moteur pas à pas SWITEC comprend deux bobines et trois pôles. Le rotor peut être placé dans six positions différentes en fonction de la direction du champ magnétique dans chacune des deux bobines (up, down, zero).

Complétez le schéma donnée en Figure 7.5 avec les indications manquantes suivante:

- la direction du champ magnétique dans les bobine nécessaire à faire tourner le rotor dans le sens direct CW (clockwise), ainsi que les lignes de champ;
- les valeurs de tension aux bornes px des bobines ('0'≡0V et '1'≡5V);
- les valeurs à écrire dans le port de sortie du microcontrôleur;
- le diagramme des temps pour les signaux de commande du moteur.

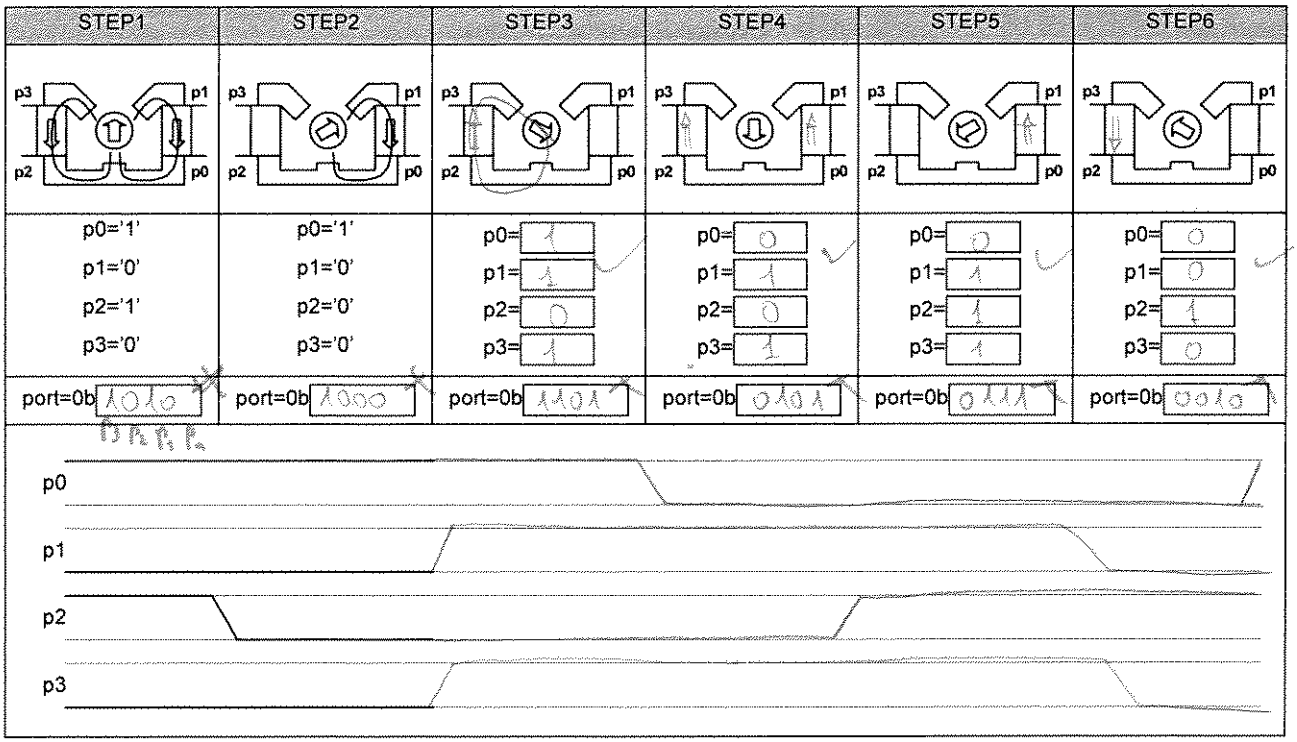


Figure 7.5: Fonctionnement du moteur pas-à-pas.

Téléchargez le programme motor.asm donné en Figure 7.6. Complétez les constantes servant à contrôler l'avance du rotor.

```
; file      motor.asm
; by        Raphael Holzer
; date      18.4.2001
;(modified: Alexandre Schmid 05.11.2003)

; program to control the SWITEC stepping motor

.include "m103def.inc"
.include "macros.asm"
.include "definitions.asm"

.equ      t1   = 1000          ; waiting period in microseconds
.equ      port_mot= PORTA      ; port to which motor is connected

.macro    MOTOR
     ldi        w,@0
     out        port_mot,w     ; output motor pin pattern
     rcall      wait           ; waiting period
.endmacro

reset:    LDSP       RAMEND    ; load stack pointer SP
          OUTI       DDRA,0x0f ; make motor port output
loop:
     MOTOR      0b1010         ; output motor patterns. COMPLETE HERE
     MOTOR      0b1000
     MOTOR      0b1001
     MOTOR      0b0001
     MOTOR      0b0011
     MOTOR      0b0010
     rjmp       loop

wait:     WAIT_US   t1          ; waiting routine
          ret
```

Figure 7.6: motor.asm.

Que peut-on contrôler au moyen de t1 ? la vitesse angulaire. Diminuez la valeur de t1, quelle est la limite de décrochage du moteur ? ~400. Essayez de faire tourner le moteur à contresens.

7.7 USAGE D'UN TIMER POUR LE CONTRÔLE DU MOTEUR PAS-À-PAS

L'usage des timers permet d'assurer la génération des signaux nécessaires au contrôle d'un moteur pas-à-pas de façon parfaitement régulière.

Téléchargez le programme motor2.asm. Il utilise le timer2 pour générer des interruptions régulières. Les nouvelles valeurs de commande de contrôle sont écrites dans le port moteur par la routine d'interruption.

Une look-up table sert à stocker la séquence de constantes à envoyer aux moteur. Le pointeur z pointe dans cette table et est [ ] ou [ ] à chaque exécution de la routine. A l'extrémité de la table le pointeur z doit être rebouclé vers le début de la table. Le bouton PD0 permet de changer la direction du moteur. Ceci se fait en parcourant la table lookup dans le sens inverse. Complétez le programme motor2.asm donné en Figure 7.7.

```
; file        motor2.asm
; by          Raphael Holzer
; date        10.5.2001
;(modified: Alexandre Schmid 05.11.2003)

; program to control the SWITEC stepping motor
; using the timer2 interrupt and a look-up table

.include "m103def.inc"
.include "macros.asm"
.include "definitions.asm"
.equ        port_mot = PORTA

; === interrupt table ===
.org   0
       rjmp         reset

.org        OVF2addr            ; timer overflow 2 interrupt vector
       rjmp         ovf2

; === interrupt routines ===
ovf2:
       JP0    PIND,0,reverse              ; forward/reverse ?
       [ ]                                ; r0 <- lookup(z)
       [ ]    [ ]                         ; output pattern to stepping motor
       out    PORTC, r0                   ; to connect oscilloscope probe
       [ ]    [ ]                         ; increment table pointer
       cpi    zl,[ ]
       br[ ]  PC+2                        ; are we at end of table?
       ldi    zl,[ ]                      ; reset to begin of table
       reti
reverse:
       lpm                                ; r0 <- lookup(z)
       out    port_mot, r0                ; output pattern to stepping motor
       out    PORTC, r0                   ; to connect oscilloscope probe
       dec    zl                          ; decrement table pointer
       cpi    zl,2*tbl_mot
       brsh   PC+2                        ; are we at begin of table?
       ldi    zl,2*tbl_mot+5              ; reset to end of table
       reti

; === lookup table ===
tbl_mot:
.db    0b0101, 0b0001, [ ], [ ], [ ], [ ]

; === initialization ===
reset: LDSP         RAMEND                ; initialize stack pointer SP
       OUTI   DDRA, 0x0f                  ; make motor lines output
       OUTI   DDRB, 0xff                  ; make portB (LEDs) output

       clr    zh                          ; Z high-byte is always zero
       ldi    zl,[ ]                      ; point to table entry

       OUTI   [ ],3     ; CS2=3 CK/64
       OUTI   [ ],(1<<TOIE2); timer 2 overflow enable
       sei                ; set global interrupt

; === main program ===
main:  in     a0, PIND
       out    PORTB, a0
       rjmp   main
```

Figure 7.7: motor2.asm.

LABORATOIRE DE SYSTEMES MICROELECTRONIQUES

EPFL STI – IMM – LSM
ELD
Station n° 11
CH-1015 Lausanne

Téléphone : +4121 693 6955
Fax : +4121 693 6959
E-mail : lsm@epfl.ch
Site web : lsm.epfl.ch

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Microcontrôleurs 2012, MT-BA4: TP08-2012-v3.5.fm

v1.0	R. Holzer	I2S	2000-2003
v2.4.1	A. Schmid	LSM	Janvier 2006
v3.5	A. Schmid	LSM	Octobre 2011

MICROCONTRÔLEURS
TRAVAIL PRATIQUE NO 8

GROUPE A	Mercredi 25.04.2012, 08:00-10:00	GROUPE B	Lundi 23.04.2012, 11:00-13:00

No du Groupe	Premier Etudiant	Second Etudiant	Evaluation	Visa Correcteur
MT 8	Damiano Ticaldo	Boileau Jean Bernard	1	T.B. ✓

8. INTERFACE UART (RS232) ET CHAINES DE CARACTÈRES

Les communications séries constituent une importante catégorie parmi les méthodes de transmission généralement appliquées par les microcontrôleurs. Ce travail pratique voit l'étude du protocole RS232 par l'établissement d'une transmission entre le PC et la carte STK-300 sur l'exemple de transmission de caractères ASCII. Dans une deuxième partie, des opérations sur les chaînes de caractères sont étudiées.

8.1 LE PROTOCOLE RS232

Un programme terminal est nécessaire à établire un communication avec le PC par l'interface série. Le programme Hyperterminal livré avec le système d'exploitation Windows peut être utilisé à cet effet. Il se lance par le navigateur Windows Program→Accessories→Communications→Hyperterminal. Les configurations du port COM1 sont données en Figure 8.1.

Figure 8.1: Configuration du port COM1.

- Bits par seconde: 19200,
- bits de données: 8,
- parité: aucune,

- bits d'arrêt: 1,
- contrôle de flux: aucun,
- paramètres→configuration ASCII: reproduction locale des caractères active (local echo on).

Le câble sériel doit être branché sur le port COM1 du PC, et l'autre extrémité dans la carte STK-300.

Observez, et reproduisez sur la Figure 8.2 les signaux émis lorsque vous affectez une entrée par l'hyperterminal de votre PC vers la carte utilisant le protocole RS232. Placez la probe de l'oscilloscope sur la pin PE0 (Rx) (jumper au millieu de la carte). Configurez l'oscilloscope de la façon suivante:

- CH1/CH2: DC, BW limit off, Coarse, 10X, Invert off, 5V/DIV;
- TRIGGER: Edge, Falling, CH1, Normal, DC, 2.5V;
- HORIZONTAL: Main, Level, 100us/DIV.

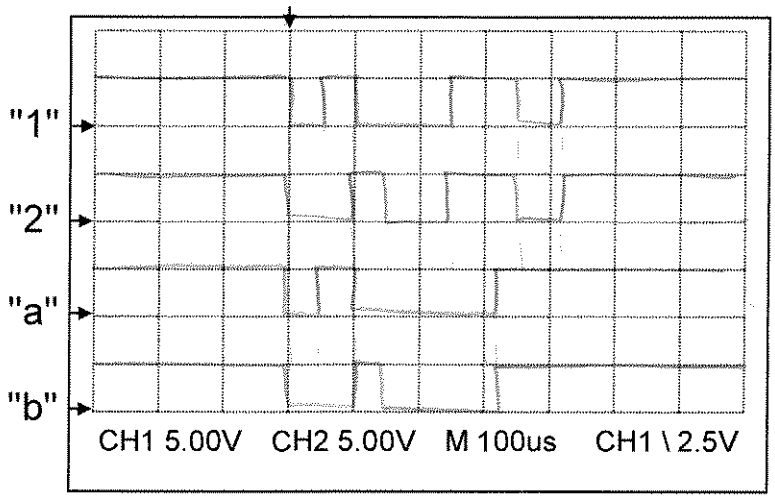


Figure 8.2: Transmission RS232.

Sur le terminal, tapez les caractère 1 puis 2, a, et finalement b. Reportez les quatre signaux observés. Indiquez clairement les start bit (S), stop bit (P), et les huit data bits. Donnez la suite des valeurs logiques obtenues en Figure 8.3.

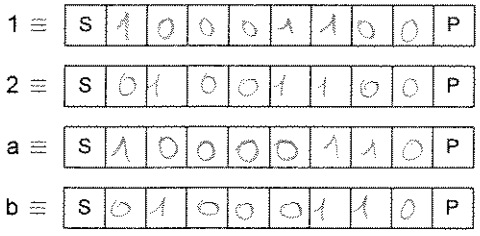


Figure 8.3: Niveaux logiques lors de la transmission.

Les bits sont transmis avec le bit de poids faible en tête. Inversez l'ordre des bits mesurés, et donnez le code ASCII en binaire et en hexadécimal en Figure 8.4.

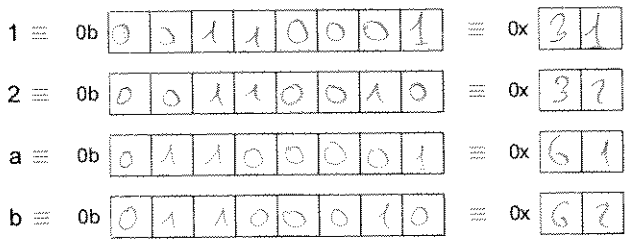


Figure 8.4: Codes hexadécimaux transmis.

8.2 LECTURE/ÉCRITURE

Chargez le programme uart1.asm. Ce programme lit une valeur et la retransmet vers le PC.

Décrivez en Figure 8.5 les signaux observés en plaçant le probe relié au canal 1 sur PE0 (Rx, receive from PC), et le probe relié au canal 2 sur PE1 (Tx, transmit to PC), lorsque vous appuyez sur la touche Z (shift-z).

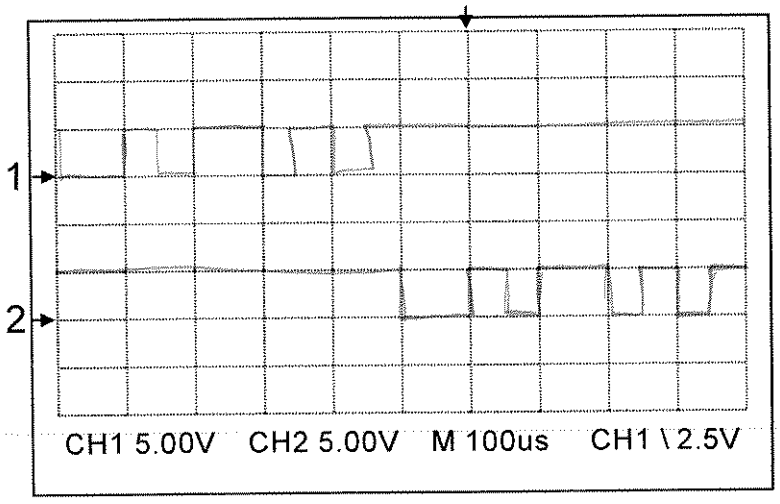


Figure 8.5: Signaux observés lors de la transmission par uart1.asm.

Les routines getc et putc servent à effectuer les opérations de shift-in et shift-out nécessaires aux transferts entre la ligne série et le registre d'entrée/sortie contenant la donnée sur 8-bit. Cette opération aurait pu être réalisée au moyen de compteurs initialisés à huit. Un autre méthode est cependant appliquée, et un autre test est réalisé:

- Quelle condition indique dans putc que tous les huit bits ont été sortis (shift-out) ?
  les 8 bits ont été transmis lorsque la retenue vaut 0
- Quelle condition indique dans getc que huit bits sont entrés (shift-in) ?
  les 8 bits ont été reçus si la retenue vaut 1

Modifiez le programme principal (main) pour que toutes les 100ms une lettre de l'alphabet soit envoyée vers le terminal; complétez le code donné en Figure 8.6.

Pourquoi a-t'on besoin d'utiliser un registre intermédiaire r23, au lieu de travailler directement avec le registre c (=r20) ?

faute modifier 1.6

```
main:
     ldi  r23, 0x61 ; initialize char to 'a'
next:
     mov       r25, r23
     rcall     putc      ; put character c
     WAIT_MS   100       ; wait 100 ms
     inc       r23       ; increment c
     cpi       r23, 'z'  ;compare c with 'z'
     brne      next      ; branch to next if not equal
     rjmp      main
```

Figure 8.6: main de uart1-alphabet.asm

On constate que le caractère z n'est pas affiché. Comment peut-on changer la condition de comparaison afin que la lettre z soit aussi affichée ?

```
cpi  r23, 0x7B          ; compare c with {.
```

8.3 UTILISATION DU MODULE UART

PREPARATION

Chargez le programme uart2.asm. Ce programme utilise le module UART de l'AVR avec ses registres à décalage qui s'occupent de l'envoi et de la réception des caractères.

Quel est le Baud rate maximum (acceptable) avec un quartz à 4MHz ? Dans quel document trouve-t'on cette information ? 19200 Bauds. datasheet Atmega 16 p 68.

Est-ce que le registre UDR signifiant UART data register est identique pour la transmission et la réception ?

Non : le registre UDR est 2 registres reçoivent physiquement mais ayant la même adresse dans l'espace

8.4 OPÉRATIONS AVEC DES CHAÎNES DE CARACTÈRES

Chargez le programme string1.asm (ne téléchargez pas). Simulez ce programme est observez comment sont utilisés les pointeurs (x, y, z) pour passer les constantes de texte comme arguments à des fonctions. Une chaîne de caractères est terminée par un 0 NUL, dont le code ASCII est 0x00.

Que fait la fonction strldi ?

strldi x,z charge la chaîne de caractère z dans x

Les deux registre pointeurs x et z sont les argument de la fonction strldi. Quelle est leur fonction ?

- z pointe la chaîne à copier program memory
- x pointe vers l'endroit où copier la chaîne SRAM

Quelle est l'adresse de la chaîne constante s1 (exprimé en bytes) ? 0x60. C'est la valeur écrite en x pour pointer sur le début de la chaîne constante.

Mettez un breakpoint sur chacune des lignes suivantes:

```
CXZ  strldi,s1,2*c1     ; load buffers with string constants
CXZ  strldi,s2,2*c2
CXZ  strldi,s3,2*c3
```

Simulez en continu jusqu'au breakpoint; dès le breakpoint atteint, simulez en pas à pas et observez les modifications sur les pointeurs et la SRAM.

8.5 EXTRACTION DE SOUS-CHAÎNES

En utilisant comme point de départ la fonction strcpy à modifier, écrivez les trois routines décrites en Table 8.7. Il faut à chaque fois ajouter un offset au pointeur, décrémenter un compteur et tester.

fonction	paramètres	explication	
str_left	x, y, a	copie les premiers a caractères de y vers x	x←left(y,a)
str_right	x, y, a	copie les derniers a caractères de y vers x	x←right(y,a)
str_mid	x, y, a, b	copie b caractères à partir de la position a de y vers x	x←mid(y,a,b)

Table 8.7: Paramètres des fonctions à écrire.

- Pour la fonction str_left il faut décompter le nombre de caractères dans a0 jusqu'à zéro, ou jusqu'à ce que la fin de la chaîne soit atteinte, au cas où elle serait plus courte que a0. Complétez le code en Figure 8.8.

```
str_left:
    ld      w,y+
    dec     a0
    breq    PC+4
    st      x+,w
    tst     w
    brne    str_left
    ret
```

Figure 8.8: str_left.

- Pour la fonction str_right il faut d'abord trouver la fin de la chaîne (y), et ensuite soustraire a0 au pointeur y (2-byte).Complétez le code en Figure 8.9.

```
str_right:
    ld      w,y+        ; find the end of string (y)
    tst     w
    brne    PC-2
    sub     yl,a0       ; decrement y by a
    brcc    PC+2        ; check for carry
    dec     yh          ; adjust in case of carry

    ld      w,y+
    st      x+,w
    tst     w
    brne    PC-3
    ret
```

Figure 8.9: str_right.

- Pour la fonction str_mid il faut combiner les deux méthodes précédentes: d'abord trouver la fin de la chaîne (y), et ensuite soustraire a0 du pointeur y (2-byte), et copier b0 caractères vers (x). Complétez le code en Figure 8.10.

```
str_mid:
        ld      w,y+    ; find the end of string (y)
        tst     w
        bne     PC-2
        sub     y,a     ; decrement y by a
        bcc     PC+2    ; check for carry
        dec     y       ; adjust in case of carry

        ld      w,y+
        dec     a
        breq    PC+4
        st      x+,w
        tst     w
        brne    PC-5
        ret
```

Figure 8.10: str_mid.

Utiliser le programme string2.asm donné en Figure 8.11 et complétez-le afin de tester vos réponses.

```
; file        string2.asm
; lab extraction de sous-chaines

.include "m103def.inc"                          ; include AVR port/bit definitions
.include "macros.asm"                           ; include macro definitions
.include "definitions.asm"                      ; include register/constant definitions

; === interrupt table ===
.org   0
       jmp    reset

.org   0x30
.include "string.asm"                           ; include string manipulation routines
.include "uart.asm"                             ; include UART routines
.include "printf.asm"                           ; include formatted printing routines

reset:
       LDSP          RAMEND                     ; Load Stack Pointer (SP)
       rcall         UART_init
       rjmp          main

; === string constants in program memory ===
c1:    .db    "hello world. ",0
c2:    .db    "how are you today?",0
c3:    .db    0

; === string buffers in SRAM ===
.dseg
s1:    .byte         32
s2:    .byte         32
s3:    .byte         32
.cseg

main:
       CXZ    strldi,s1,2*c1                    ; load string constants into buffer SRAM
       CXZ    strldi,s2,2*c2
       CXZ    strldi,s3,2*c3

       ldi    a0,                               ; COMPLETE HERE
       ldi    b0,                               ; COMPLETE HERE

       CXY    str_left,s1,s2                    ; test str_left routine; comment others
       ;CXY   str_right,s1,s2                   ; uncomment to test
       ;CXY   str_mid,s1,s2                     ; uncomment to test

       rjmp main

str_left:
       ld     w,y+                              ; COMPLETE HERE (3 lines)

       tst    w
       brne   str_left
       ret

str_right:                                       ; COMPLETE HERE (5 lines)
       dec    yh                                ; adjust in case of carry

       ld     w,y+                              ; COMPLETE HERE (2 lines)

       brne   PC-3
       ret

str_mid:                                         ; COMPLETE HERE (6 lines)

       ld     w,y+                              ; COMPLETE HERE (2 lines)

       st     x+,w
       tst    w                                 ; COMPLETE HERE (1 line)

       ret
```

Figure 8.11: string2.asm.

LABORATOIRE DE SYSTEMES MICROELECTRONIQUES

EPFL STI – IMM – LSM
ELD
Station n° 11
CH-1015 Lausanne

Téléphone : +4121 693 6955
Fax : +4121 693 6959
E-mail : lsm@epfl.ch
Site web : lsm.epfl.ch

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

93/95

Microcontrôleurs 2012, MT-BA4: TP09-2012-v3.5.fm

v1.0	R. Holzer	I2S	2000-2003
v2.4.1	A. Schmid	LSM	Janvier 2006
v3.5	A. Schmid	LSM	Octobre 2011

MICROCONTRÔLEURS
TRAVAIL PRATIQUE NO 9

GROUPE A	Mercredi 02.05.2012, 08:00-10:00	GROUPE B	Lundi 07.05.2012, 11:00-13:00

No du Groupe	Premier Etudiant	Second Etudiant	Evaluation	Visa Correcteur
JE8	Berteaux Jean-Bernard	Perriere Timothée	1	B. Rossier

9. INTERFACE I²C AVEC EEPROM

Ce travail pratique voit l'étude du protocole I²C sur un exemple de communication avec une EEPROM.

9.1 LE STANDARD I²C

Le standard I²C est un mode de communication s[erie] s[ynchrone], c'est-à-dire qu'une ligne transmet l'horloge. Ce standard utilise les deux lignes:

- SDA pour [Serial Data/Adress Input/Output], et
- SCL pour [Serial Clock].

Plusieurs modules périphériques peuvent être connectés simultanément sur ces lignes. Pour garantir ceci, leur étage d'accès aux lignes est constitué de circuit en c[ollecteur] o[uvert].

Afin que les lignes ne restent pas en haute impédance lorsqu'aucun module n'y accède, SDA et SCL sont connectés à [Vdd] par des [résistance] le pull-up.

Le long des lignes on trouve un seul module m[aitre] ainsi que un ou plusieurs modules es[clave].

A l'état de repos, les lignes SDA et SCL sont au niveau logique [1].

Décrivez sur la les conditions de début et de fin de transfert d'un octet, ainsi que leur diagramme des temps simplifié.

Condition de start:

Un flanc descendant de SDA lorsque SCL est à 1

Condition de stop:

Un flanc montant de SDA lorsque SCL est à 1

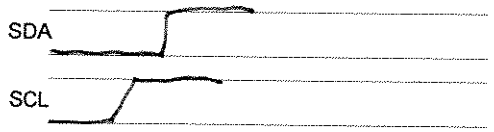


Figure 9.1: Conditions de start et stop pour la transmission d'un octet par protocole I2C.

PREPARATION PREPARATION PREPARATION

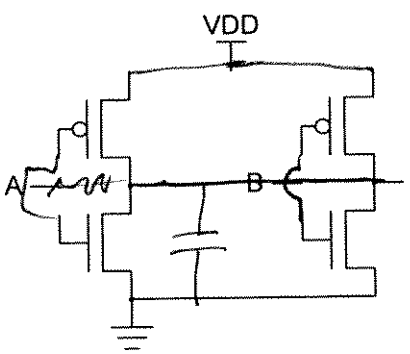
13/13

Donc, ce qui différentie les conditions de start et stop de la transmission des données au niveau du protocole c'est que pour start on passe de Vdd à 0 et l'inverse pour Stop

start/stop : SCL est à 1, transmission : SCL est à 0

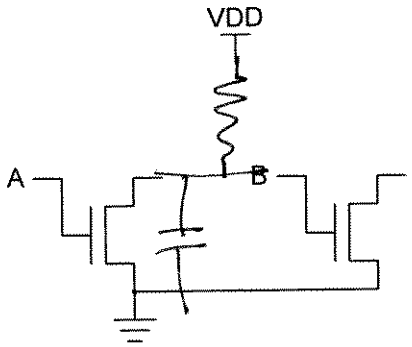
Deux différents types d'étages de sortie sont décrit en Figure 9.2. Complétez sur la figure les connections manquantes; connectez les sortie ensemble; ajoutez la résistance de pull-up dans le cas du circuit collecteur ouvert. Indiquez l'état de la ligne pour le cas des différentes combinaisons de A et B par '1', '0', court-circuit ou haute impédance.

Etage de sortie CMOS



A	B	état de la ligne
0	0	Vdd (1) ✓
0	1	court-circuit ✓
1	0	court-circuit ✓
1	1	0 ✓

Etage de sortie en collecteur ouvert (drain ouvert)



A	B	état de la ligne
0	0	Vdd (1) ✓
0	1	0 ✓
1	0	0 ✓
1	1	0 ✓

Figure 9.2: Comparaison étage de sortie CMOS avec collecteur ouvert.

Téléchargez le programme i2cx_1.asm donné en Figure 9.3 qui teste les quatre macros SCL0, SDA0, SCL1, et SDA1. Placez le module I2C sur le port B. Connectez la sonde 1 de l'oscilloscope avec SCL, et la sonde 2 avec SDA.

```
; file    i2cx_1.asm
.include "m103def.inc"
.include "macros.asm"

; testing pull-up, pull-down

reset:    rjmp      main
.include "i2cx.asm"

main:SCL0           ; serial clock = 0
     SDA0           ; serial data  = 0
     SCL1           ; serial clock = 1
     SDA1           ; serial data  = 1
     rjmp      main
```

Figure 9.3: i2cx_1.asm.

10/11

Reportez en Figure 9.4 les signaux observés, et indiquez les instructions correspondant aux différentes phases.

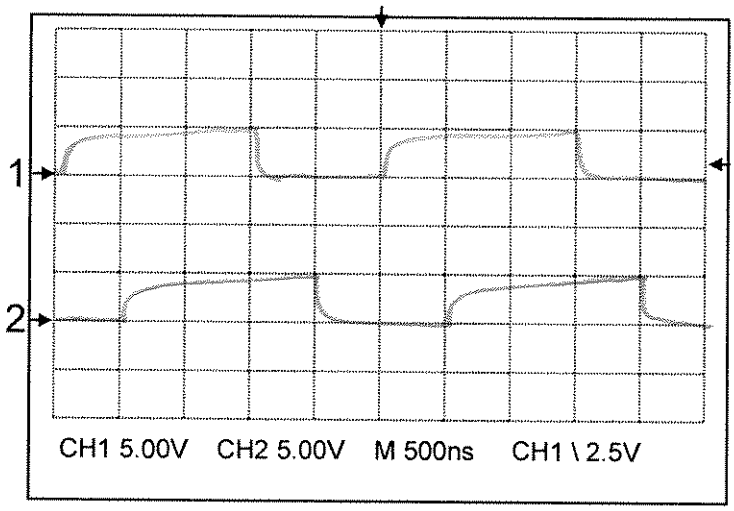


Figure 9.4: Signaux observés lors de l'exécution de i2cx_1.ams.

Pourquoi les flancs sont-ils asymétriques ? Pour comprendre, simulez en observant les I/Os (New I/O view) avec le port B déployé.

Parce qu'il y a une capacité! Quand on passe de 0 à 1, la capacité se charge avec une constante de temps 1/RC. Par contre quand elle se décharge elle le fait par contre quand on passe de 1 à 0, elle se décharge presque instantanément. (résistance 10 MΩ à l'état passant très faible).

Ajoutez la ligne suivante, et téléchargez à nouveau:

```
reset:    OUTI      PORTB,0xff
          rjmp      main
```

Expliquez pourquoi les lignes ne transmettent plus rien.

On applique que des 1 au port B ce qui se traduit par des 0 en I^2C (111)
La ligne est en permanence tirée à 1 soit par la valeur due à PORTB = 0xff, soit par la résistance de pull-up

Ainsi, pour simuler une sortie en collecteur ouvert qui n'existe pas sur le microcontrôleur, il est possible d'utiliser une combinaison des valeurs de PORTx et DDRx, dans notre cas PORTB et DDRB. Complétez cette combinaison en Table 9.1.

sortie désirée	état de la ligne (pull-up, -down, high-Z)	DDRx	port en entrée/ sortie	PORTx
0	pull-down	1	Sortie	0
1	high-Z	0	entrée	-

Table 9.1: Conditions de simulation de circuit à collecteur ouvert.

La routine i2c_init initialise la direction (DDRx) et la valeur de port (PORTx) de la bonne manière. Elle doit être appelée avant toute utilisation du microcontrôleur en émulation de communication par I^2C.

PREPARATION

12/12

Au début du programme la constante symbolique SDA_port=PORTB est définie. Dans le programme on accède au registre SDA_port-1. Quel registre est alors adressé ? DDRB ✓.

9.2 EMISSION D'UN BYTE SUR UN BUS I²C

Téléchargez le programme i2cx_3.asm donné en Figure 9.5. Ce programme teste la fonction i2c_start qui produit la condition de start et ensuite transmet les 8 bits de l'octet situé dans le registre a0 et finalement reçoit l'acknowledge.

```
; file    i2cx_3.asm
.include "m103def.inc"
.include "macros.asm"
.include "definitions.asm"

; testing i2c_start

reset:LDSP    RAMEND          ; load stack pointer SP
      rcall   i2c_init        ; initialize DDRx and PORTx
      sbi     DDRB,0x07       ; activate LED PB7
      rjmp    main

.include "i2cx.asm"

main:ldi      a0,0x5a         ; load the parameter
      rcall   i2c_start       ; send byte
      WAIT_MS 1

      bld     r0,7
      tst     r0
      brne    PC+2
      sbi     PORTB,0x07
      rjmp    main
```

Figure 9.5: i2cx_3.asm.

L'envoi d'un seul bit est fait par la macro I2C_BIT_OUT reproduite en Figure 9.6. Un bit est produit sur 10 cycles soit 2,5 ✓ microsecondes.

```
.macro        I2C_BIT_OUT              ;bit
      sbi     SCL_port-1,SCL_pin       ; SCL low (output, port=0)
      in      w,SDA_port-1
      bst     a0,@0
      bld     w,SDA_pin
      out     SDA_port-1,w             ; transfer bit_x to SDA
      cbi     SCL_port-1,SCL_pin       ; release SCL (input, hi Z)
      rjmp    PC+1                     ; wait 2 cyles
.endmacro
```

Figure 9.6: Macro I2C_BIT_OUT.

Etudiez le déroulement de la macro, et reportez en Figure 9.7 les signaux attendus avec leurs timings correctes, et les instructions correspondantes. Hypothèse: a0 = 0xff.

2/2

Cycle #	1	2	3	4	5	6	7	8	9	10
SDA										
SCL										
Instruction	sbi	sbi	in	bst	bld	out	cbi	cbi	rjmp	rjmp

Figure 9.7: Timings de la macro I2C_BIT_OUT.

Exécutez le programme i2cx_3.asm sur le système cible, et reportez sur la Figure 9.8 les signaux observés à l'oscilloscope. Indiquez sur le diagramme les trois phases du protocole visibles, conformément à la Figure 12-5 du cours, pour la transmission de la donnée 0x5a = 0b01011010 ✓

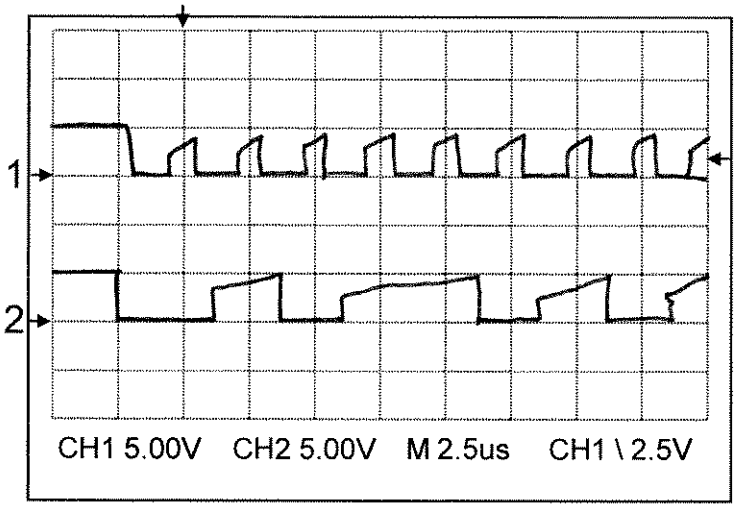


Figure 9.8: Exécution avec 0x55 comme donnée transmise.

Quelle est l'information affichée par la LED PB7, et que signifie le fait qu'elle soit allumée, est-ce normal ?

La led 7 s'éteint lorsque le maître reçoit une confirmation de réception de la part d'un esclave. Comme il n'y a aucun périphérique de branché, personne n'enverra de confirmation et la led restera allumée. ✓

2/2

9.3 ADRESSAGE D'UN MODULE

Le protocole I²C permet l'adressage de cent vingt-sept esclaves différents. L'adresse d'identification de l'EEPROM est 0b1010xxx auquel s'ajoutent 3 bits supplémentaires:

- la première partie de l'adresse est fixe et est composée de quatre bits; par qui et de quelle façon est-elle déterminée ? c'est le constructeur qui la fixe. Où peut-on trouver la valeur de cette adresse ? dans la data-sheet M24C64 table #2;
- la deuxième partie de l'adresse n'est pas fixe et est composée de trois bits; par qui et de quelle façon est-elle déterminée ? elle est définie par nous sur la puce et indique l'adresse d'un circuit EEPROM;
- ainsi, il est possible de connecter 8 EEPROMs M24C64 sur un même bus I2C.

Le premier byte envoyé par le maître est l'adresse du module I²C avec lequel il désire communiquer. L'unique module présent sur le bus qui reconnaît son adresse répond par un acknowledge, pratiquement tire la ligne vers le bas.

Téléchargez le programme i2cx_4.asm donné en Figure 9.9.

```
; file    i2cx_4.asm
.include "m103def.inc"
.include "macros.asm"
.include "definitions.asm"

; testing i2c EEPROM device address

reset:LDSP    RAMEND                    ; load stack pointer SP
      rcall   i2c_init                  ; initialize DDRx and PORTx
      rjmp    main

.include "i2cx.asm"

main:
      CA      i2c_start, 0b10100000
      rcall   i2c_stop
      WAIT_US 10

      ;CA     i2c_start, 0b10110000
      ;rcall  i2c_stop
      WAIT_MS 100
      rjmp    main
```

Figure 9.9: i2cx_4.asm.

Exécutez le programme sur le système cible, et reportez sur la Figure 9.10 les signaux observés à l'oscilloscope. Indiquez sur le diagramme les trois phases du protocole visibles, conformément à la Figure 12-5 du cours, pour la transmission de la donnée 0xa0 = 0b10100000.

L'EEPROM a-t'elle été adressée et a-t'elle répondu ? Oui

L'EEPROM a ses 3 bits d'adresse (programmables) à 0, l'adresse est correcte => ack = Ok

7/7

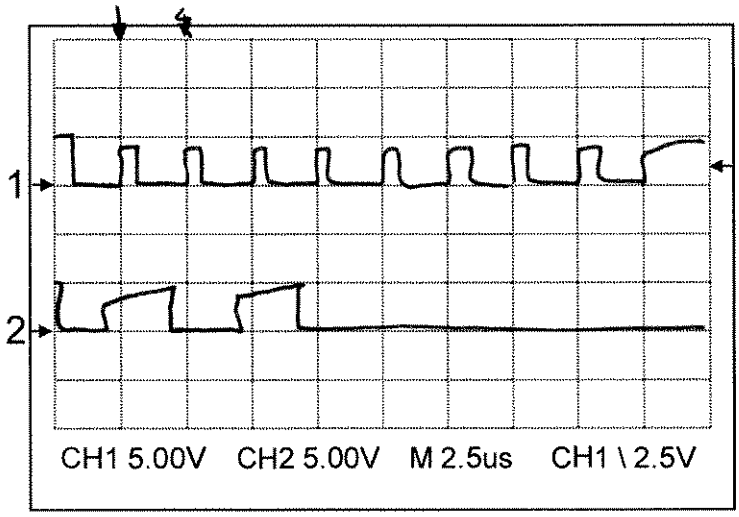


Figure 9.10: Exécution avec 0xa0 comme adresse transmise.

Commentez les lignes d'adressage du haut, et enlevez les signes de commentaire sur les lignes d'adressage du bas. Répétez l'expérience.

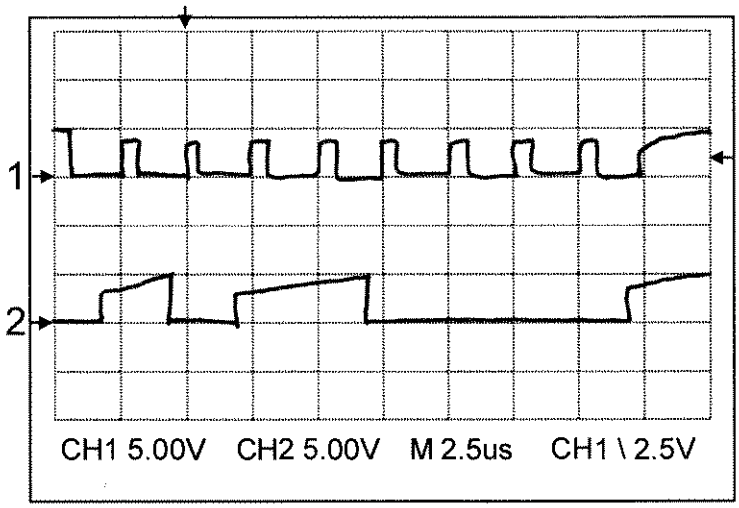


Figure 9.11: Exécution avec 0xb0 comme adresse transmise.

L'EEPROM a-t'elle été adressée et a-t'elle répondu ? Non l'adresse est fausse.
Il n'y a pas eu plus d'acq de la part de l'EEPROM

9.4 ANALYSE DE LECTURE/ÉCRITURE DANS L'EEPROM

Téléchargez le programme objet i2c_eeprom1.hex qui permet de communiquer avec l'EEPROM M24C64 par protocole I^2C. L'EEPROM comprend 64kb de mémoire, soit 8 kB.

Exécutez le programme sur le système cible, et observez les signaux SCL et SDA au moyen de l'oscilloscope, utilisé à cette occasion comme remplaçant d'un analyseur logique. Configurez l'oscilloscope avec les valeurs données en Figure 9.12.

2/2

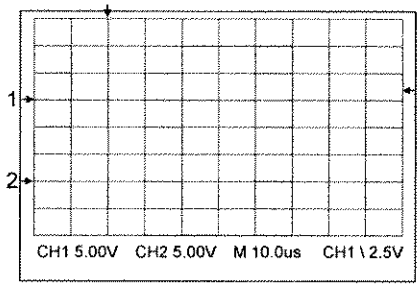


Figure 9.12: Configuration de l'oscilloscope.

Analysez les traces observées en vous aidant impérativement des datasheets de l'EEPROM M24C64. Identifiez les conditions start/stop, repérez les paquets de huit bits, les acknowledge. Utilisez le bouton Horizontal→Position ainsi que la base de temps de l'oscilloscope afin d'observer la totalité de la communication. Reportez en Figure 9.13 les différentes phases du protocole identifiées, les valeurs transmises, ainsi que le nom du module qui parle à ce moment (master/slave) et le type de communication (read/write). Aidez-vous des curseurs, et si nécessaire, configurez le trigger en mode single shot.

PHASE #	PHASE1	PHASE2	PHASE3	PHASE4	PHASE5	PHASE6	PHASE7	PHASE8	PHASE9
DATA / PHASE NAME	start	1010000 0	ack	00000000	ack	00000000	ack	10100001	ack
M/S TALKING	master	master	slave	master	slave	master	slave	master	slave
Read/Write Mode	-	write	-	write	-	write	-	write	-

PHASE #	PHASE10	PHASE11	PHASE12	PHASE13	PHASE14	PHASE15	PHASE16
DATA / PHASE NAME	01010101	ack	00110011	ack	00001111	no ack	stop
M/S TALKING	master slave	slave master	master slave	slave master	master slave	slave master	master
Read/Write Mode	write read	-	write read	-	write read	-	-

Figure 9.13: Analyse de lecture/écriture par I2C avec l'EEPROM M24C64.

Sur la base de votre analyse, quelle est l'opération effectuée par ce programme ?

Conversation entre le master et le slave. Master envoie d'abord à slave puis ensuite les rôles s'inversent.

X

1) effectue la lecture de trois adresses consécutives. Pour cela il est nécessaire que l'EEPROM ait été programmé auparavant

15/16