Institute of Electrical and Microengineering
Biomedical and neuromorphic microelectronic systems

Microcontrôleurs 2023, MT-BA4, Microcontrôleurs et systèmes numériques 2023, EL-BA4. TP07-2023-v5.5.fm
v5.5 A. Schmid 2022, May 23

MICROCONTRÔLEURS
MICROCONTRÔLEURS ET SYSTÈMES NUMÉRIQUES
TRAVAIL PRATIQUE NO 7

MT GROUPE A	MT Groupe B		EL

No du Groupe	Premier Etudiant	Second Etudiant	Evaluation	Visa Correcteur
093	Nathann Morand	Felipe Ramirez		

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 AtmelStudio. Téléchargez-le sur la carte, après avoir programmé les fusibles de fréquence à 4 Mhz, rigoureusement comme présenté en Figure 2.6 (TP02).

- Le bit AS0 est mis à 1, donc la source d'horloge du timer provient de l'horloge à quartz avec une fréquence de 32'768 Hz.
- Le prescaler est à 2. La période d'interruption (temps entre deux interruptions consécutives) est donnée par la formule T= 256*8/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, et MEASURE→CH2–Period. Vous ne devez pas déconnecter le cable plat, ni sur le PortD ni PortB pendant la mesure au moyen de la sonde, car ces cable réalisent la connection des boutons-poussoirs et des LEDs sur les ports de l'AVR; il faut donc placer la sonde à la base de la pin, et laisser le connecteur au dessus.

La période du signal observé est de 125 ms sur PB1 et 192 ms sur PB7.

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

```
ldi  r16, 0b00000101
out  TCCR0, r16
```

Effectuez le changement, téléchargez le programme. La période d'interruption est 1000 ms. Il peut être utile d'utiliser la mesure au moyen des curseurs (CURSOR); dans ce cas, la source du trigger doit être choisie sur le canal observé (CHANNEL 1 ou CHANNEL 2) car les signaux observent entre eux un décalage temporel qui rend leur observation simultanée difficile.

7.2 LE PRESCALER

Téléchargez le programme timer0_prescaler.asm en Figure 7.1. Placez le module M2 sur le PORTE. Placez une sonde de l'oscilloscope sur PD7 afin de mesurer les périodes des "timer0 overflows." Entrez les valeurs 000 jusqu'à 111 au moyen des boutons-poussoirs, effectuez les mesures et remplissez les cases en Table 7.1.

```
; file timer0_prescaler.asm   target ATmega128L-4MHz-STK300
; purpose timer 0 overflow
; module: M3, output port: PORTE
.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=",FBIN,TCCR0+0x20,0

     WAIT_MS   100
     rjmp      main
```

Figure 7.1: timer0_prescaler.asm

Etudiez le code afin de comprendre comment les boutons-poussoirs doivent être utilisés.

Afin de simplifier la mesure, utiliser le bouton de l'oscilloscope "RUN/STOP" qui figera la trace; puis, il est possible de mesurer au moyen de la fonction "CURSOR," par exemple.

TCCR	prescaler/fonction	période (*)
000	stopped	infini
001	1	128 us
011	32	4096 us
110	256	32.76 ms
111	1024	131 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 (il ne s'agit pas de la période de l'horloge principale).

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	64.1 Hz	15.6 ms
timer1	PB3	0.476 Hz	2.099 s
timer2	PB5	976.6 Hz	1.024 ms
programme principal	PB7	4.96 Hz	201.6 ms

Table 7.2: Etude du prescaler.

Travaillez maintenant avec le simulateur AtmelStudio. Placez un breakpoint sur la première instruction de l'interruption numéro 3 (rjmpoverflow2). Simulez sans arrêt jusqu'à ce breakpoint par Run to Cursor, CTRL-F10). Utilisez le chronomètre et indiquez le temps passé entre les interruptions du timer 2: 512 µs.

Chargez et assemblez le ficher timer_ov-2.asm. Placez le curseur sur la première ligne de code du main; il s'agit du premier d'une série de nop, puis effectuant un Run to Cursor (CTRL-F10). Ceci fait, continuez la simulation en pas-à-pas (F11) et ouvrez et observez la fenêtre des I/Os, et spécifiquement le timer lié à l'interruption numéro 3.

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 déclenchée lorsque ce registre passe par la valeur 0xFF. Combien de cycles peut-on compter dans l'intervalle entre deux interruptions du timer 2 ? 2048 cycles. Vous pouvez garder F11 pressé afin d'avancer rapidement dans la simulation des nop, et afin d'observer l'incrémentation du compteur. Effectuez cette même manipulation plusieurs fois. Pourquoi l'interruption est-elle parfois retardée d'un cycle ?

car il arrive que l'unstruction arrive au moment ou on execute un instruction qui dur plus de 1 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 TCNTxx 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 (délais) 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, x est 0 ou 1).

```
.set      clock0 = 32768              ; in Hz
.set      clock1 = 4000               ; 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 = timeout0*clock0/(1000*prescaler0)
.set      timer1 = timeout1*clock1/(1000*prescaler1)
```

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 = 11 msec; timout0 réel synthétisé = 10.98 ms;
- timeout0 désiré = 3 msec; 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 son émis en agissant sur la valeur stockée dans le registre interne OCR2 qui stocke la valeur de comparaison du timer.

```
; file      pulsout4.asm   target ATmega128L-4MHz-STK300
; purpose generation of rectangular signal using timer2
; module: M5, output port: PORTE
.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
      PRINTFLCD
.db   "CS2=",FHEX,a+1," OCR2=",FHEX,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_CYCb0,10,250
      rjmp  main
decremb:
      DEC_CYCb0,10,250
      rjmp  main
increma:
      INC_CYCa1,2,5
      rjmp  main
decrema:
      DEC_CYCa1,2,5
      rjmp  main
```

Figure 7.4: pulsout4.asm.

Appuyer sur les boutons PD2 et PD3 sert à faire varier la fréquence du timer 2 donc la gamme de son joué en agissant sur la valeur stockée dans CS 2 du reg TCCR2 que correspond à préscaler du timer 2.

7.6 CONTRÔLE DU MOTEUR PAS-À-PAS

PREPARATION

PREPARATION

PREPARATION

Un moteur pas-à-pas de faible puissance peut être contrôlé directement par les ports de sortie d'un micro-contrôleur. Le moteur pas à pas X27 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 donné en Figure 7.5 avec les indications manquantes suivantes:

- 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.

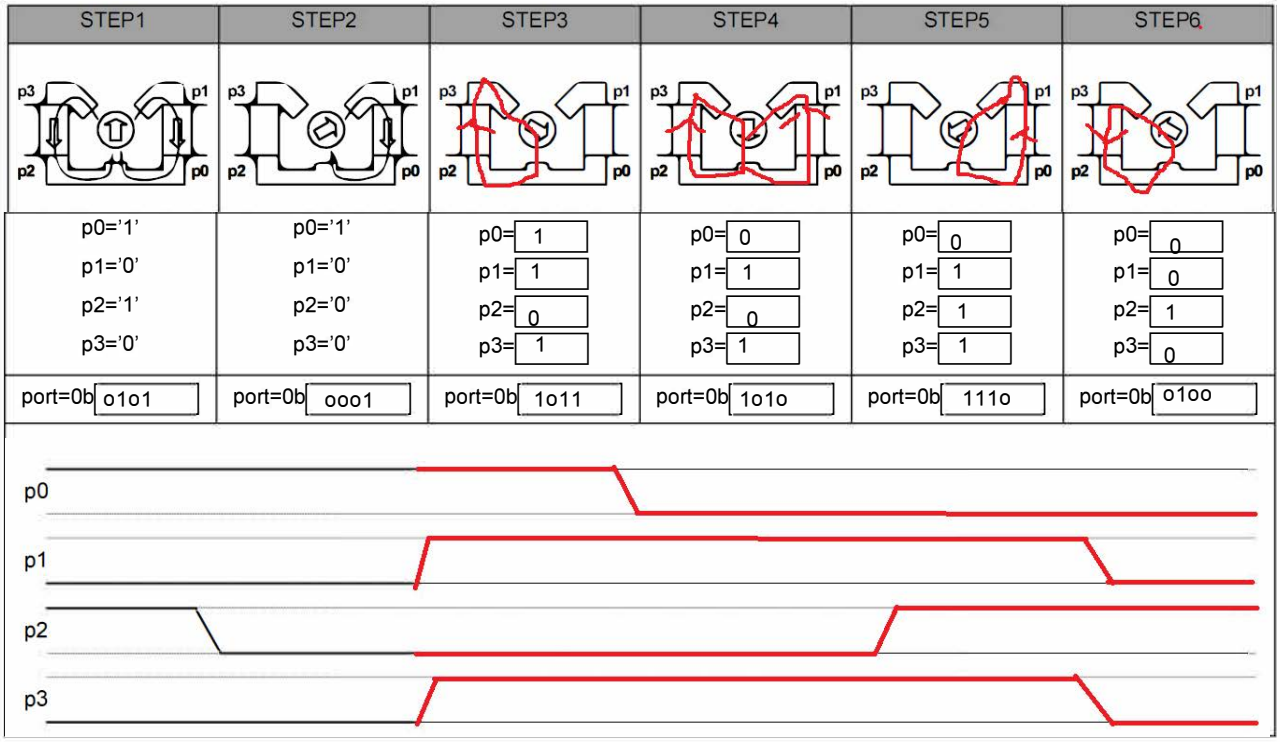

Figure 7.5: Fonctionnement du moteur pas-à-pas.

Téléchargez le programme motor.asm donné en Figure 7.6. Complétez les constantes servant à contrôler l’avance du rotor. Placez le module M1 sur le PORTA.

```
; file    motor.asm   target ATmega128L-4MHz-STK300
; purpose stepper motor control
; module: M1, output port: PORTA
.include "macros.asm"
.include "definitions.asm"

.equ t1       = 1000                ; waiting period in micro-seconds
.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                  ; wait period
.endmacro

reset: LDSP   RAMEND                ; load stack pointer SP
     OUTI     DDRA,0x0f             ; make motor port output
loop:
     MOTOR    0b 0101               ; output motor patterns COMPLETE HERE
     MOTOR    0b 0001
     MOTOR    0b 1011
     MOTOR    0b 1010
     MOTOR    0b 1110
     MOTOR    0b 0100
     rjmp     loop

wait: WAIT_US t1                    ; wait routine
     ret
```

Figure 7.6: motor.asm.

Que peut-on contrôler au moyen de t1 ? nombre de rotation par minute. Diminuez la valeur de t1, quelle est la limite de décrochage du moteur ? 500 ms. 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 moteurs. Le pointeur z pointe dans cette table et est incrémenté ou décrémenté 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   target ATmega128L-4MHz-STK300
; purpose stepper motor control using timer2 interrupt and LUT
; module: M1, output port: PORTA
.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 ?
       lpm                                   ; r0 <- lookup(z)
       out    port_mot, r0                   ; output pattern to stepping motor
       out    PORTC, r0                      ; to connect oscilloscope probe
       inc    zl                             ; increment table pointer
       cpi    zl,2*tbl_mot
       br lo  PC+2                           ; are we at end of table?
       ldi    zl,2*tbl_mot+6                 ; 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, 0b1011, 0b1010, 0b1110, 0b0100


; === 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,2*tbl_mot                   ; point to table entry

       OUTI   TCCR2,3                        ; CS2=3 CK/64
       OUTI   TIMSK,(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.