Comparaison des temps de réaction interruptions / scrutation AT mega328p

Quand on pense à une approche temps réel sur un µC, l'opinion politiquement correcte est d'utiliser les interruptions.
L'interruption n'est en fait qu'un sous-programme, déclenché par une entrée (généralement hardware) lorsque la phase d'horloge atteint la valeur ad hoc (synchronisation d'un événement asynchrone). Ce n'est donc pas strictement immédiat. L'interruption commence par une sauvegarde des registres importants du processeur qui permettent de retrouver le contexte en quittant l'interruption lors de la phase de restitution du contexte. Cette sauvegarde concerne un nombre variable de registres selon le degré de précaution. Ce degré de protection est contrôlable par les options du compilateur. Ces options sont appelées par le Makefile ou plus rarement par la commande directe de compilation par ligne de code. On peut normalement modifier son makefile mais je ne sais pas ce qu'il en est dans le monde Arduino.
Le compilateur avr-gcc distribué gratuitement sous Linux dispose de 4 niveaux d'optimisation de compilation qui agissent sur de nombreux paramètres, dont la frilosité des interruptions, -O0, -O1, -O2, -O3 et -Os (speed).

J'utilise une micro-carte Arduino nano pro (34 x 19 mm) hors de tout environnement logiciel Arduino, juste la carte à 1,3€ avec mes programmes écrits en c. Le programme de test, en annexe, génère deux impulsions visibles sur la LED de la carte, qui sont déclenchées, selon la pin d'entrée utilisée, soit par une interruption, soit par une scrutation (polling).
La carte est équipée d'un quartz (précis) au lieu du classique résonateur céramique (10 à 100 fois moins précis), cela entraîne des options programmées pour le fabricant de la carte : les fuses ne sont pas ceux d'origine, certains sont nécessaires pour le bon fonctionnement du quartz, d'autres pour l'emplacement interne du boot, d'autres pour le comportement en cas de perte de tension, et surtout le bit CKDIV8 qui est habituellement placé en division par 8 systématique de la vitesse du processeur, ici, il est d'origine placé en pleine vitesse (16MHz soit 62,5 ns pour l'instruction la plus courte puisque c'est un processeur RISC)
Le Makefile utilisé ici est créé par des auteurs de la communauté travaillant près d'Atmel : Jörg Wünsch, Peter Fleury, ...
Le code compilé (main.lss) est observé pour le nombre d'instruction assembleur crées (si le makefile est programmé pour). Le code source fait une dizaines de lignes de c, le reste sont des initialisations et des commentaires.

1 Optimisation zéro :

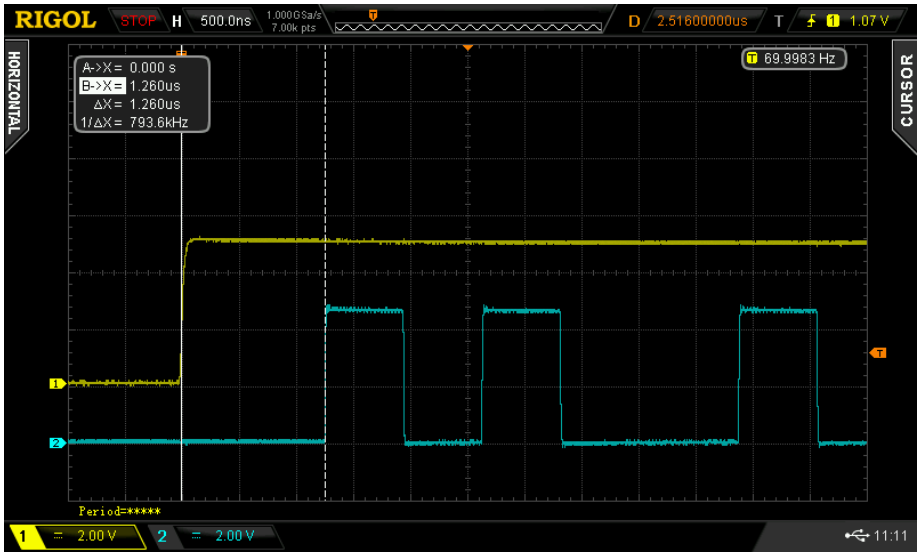
option -O0

1.1 code polling généré opt0 :

90 lignes assembleur

```
// polling check
              if (PIND &(1<<PD3)){
  dc:   89 e2         ldi     r24, 0x29       ; 41
  de:   90 e0         ldi     r25, 0x00       ; 0
  e0:   fc 01         movw    r30, r24
  e2:   80 81         ld      r24, Z
  e4:   88 2f         mov     r24, r24
  e6:   90 e0         ldi     r25, 0x00       ; 0
  e8:   88 70         andi    r24, 0x08       ; 8
  ea:   99 27         eor     r25, r25
  ec:   00 97         sbiw    r24, 0x00       ; 0
  ee:   29 f1         breq    .+74            ; 0x13a <main+0xaa>
                      PORTB |= (1<<PB5);
  f0:   85 e2         ldi     r24, 0x25       ; 37
  f2:   90 e0         ldi     r25, 0x00       ; 0
```

```
 f4:   25 e2          ldi     r18, 0x25      ; 37
 f6:   30 e0          ldi     r19, 0x00      ; 0
 f8:   f9 01          movw    r30, r18
 fa:   20 81          ld      r18, Z
 fc:   20 62          ori     r18, 0x20      ; 32
 fe:   fc 01          movw    r30, r24
100:   20 83          st      Z, r18
                      PORTB &= ~(1<<PB5);
102:   85 e2          ldi     r24, 0x25      ; 37
104:   90 e0          ldi     r25, 0x00      ; 0
106:   25 e2          ldi     r18, 0x25      ; 37
108:   30 e0          ldi     r19, 0x00      ; 0
10a:   f9 01          movw    r30, r18
10c:   20 81          ld      r18, Z
10e:   2f 7d          andi    r18, 0xDF      ; 223
110:   fc 01          movw    r30, r24
112:   20 83          st      Z, r18
                      PORTB |= (1<<PB5);
114:   85 e2          ldi     r24, 0x25      ; 37
116:   90 e0          ldi     r25, 0x00      ; 0
118:   25 e2          ldi     r18, 0x25      ; 37
11a:   30 e0          ldi     r19, 0x00      ; 0
11c:   f9 01          movw    r30, r18
11e:   20 81          ld      r18, Z
120:   20 62          ori     r18, 0x20      ; 32
122:   fc 01          movw    r30, r24
124:   20 83          st      Z, r18
                      PORTB &= ~(1<<PB5);
126:   85 e2          ldi     r24, 0x25      ; 37
128:   90 e0          ldi     r25, 0x00      ; 0
12a:   25 e2          ldi     r18, 0x25      ; 37
12c:   30 e0          ldi     r19, 0x00      ; 0
12e:   f9 01          movw    r30, r18
130:   20 81          ld      r18, Z
132:   2f 7d          andi    r18, 0xDF      ; 223
134:   fc 01          movw    r30, r24
136:   20 83          st      Z, r18
               }
```

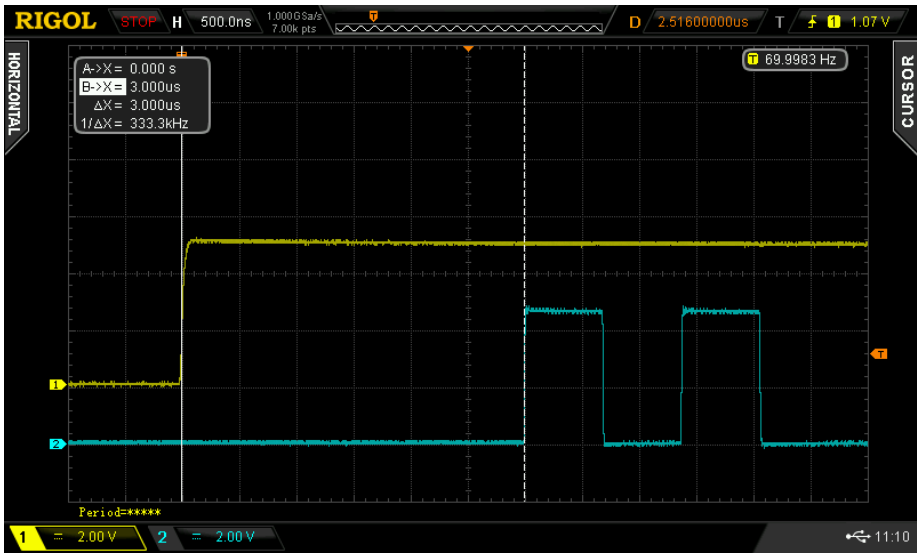


temps de latence 1,2 µs

1.2 code interruption généré opt0

130 lignes assembleur

```
ISR (INT0_vect){
 13c:   1f 92           push    r1
 13e:   0f 92           push    r0
 140:   00 90 5f 00     lds     r0, 0x005F
 144:   0f 92           push    r0
 146:   11 24           eor     r1, r1
 148:   2f 93           push    r18
 14a:   3f 93           push    r19
 14c:   8f 93           push    r24
 14e:   9f 93           push    r25
 150:   ef 93           push    r30
 152:   ff 93           push    r31
 154:   cf 93           push    r28
 156:   df 93           push    r29
 158:   cd b7           in      r28, 0x3d      ; 61
 15a:   de b7           in      r29, 0x3e      ; 62
        PORTB |= (1<<PB5);
 15c:   85 e2           ldi     r24, 0x25      ; 37
 15e:   90 e0           ldi     r25, 0x00      ; 0
 160:   25 e2           ldi     r18, 0x25      ; 37
 162:   30 e0           ldi     r19, 0x00      ; 0
 164:   f9 01           movw    r30, r18
 166:   20 81           ld      r18, Z
 168:   20 62           ori     r18, 0x20      ; 32
 16a:   fc 01           movw    r30, r24
 16c:   20 83           st      Z, r18
        PORTB &= ~(1<<PB5);
 16e:   85 e2           ldi     r24, 0x25      ; 37
 170:   90 e0           ldi     r25, 0x00      ; 0
 172:   25 e2           ldi     r18, 0x25      ; 37
 174:   30 e0           ldi     r19, 0x00      ; 0
 176:   f9 01           movw    r30, r18
 178:   20 81           ld      r18, Z
 17a:   2f 7d           andi    r18, 0xDF      ; 223
 17c:   fc 01           movw    r30, r24
 17e:   20 83           st      Z, r18
        PORTB |= (1<<PB5);
 180:   85 e2           ldi     r24, 0x25      ; 37
 182:   90 e0           ldi     r25, 0x00      ; 0
 184:   25 e2           ldi     r18, 0x25      ; 37
 186:   30 e0           ldi     r19, 0x00      ; 0
 188:   f9 01           movw    r30, r18
 18a:   20 81           ld      r18, Z
 18c:   20 62           ori     r18, 0x20      ; 32
 18e:   fc 01           movw    r30, r24
 190:   20 83           st      Z, r18
        PORTB &= ~(1<<PB5);
 192:   85 e2           ldi     r24, 0x25      ; 37
 194:   90 e0           ldi     r25, 0x00      ; 0
 196:   25 e2           ldi     r18, 0x25      ; 37
 198:   30 e0           ldi     r19, 0x00      ; 0
 19a:   f9 01           movw    r30, r18
 19c:   20 81           ld      r18, Z
 19e:   2f 7d           andi    r18, 0xDF      ; 223
 1a0:   fc 01           movw    r30, r24
 1a2:   20 83           st      Z, r18
}
 1a4:   df 91           pop     r29
 1a6:   cf 91           pop     r28
 1a8:   ff 91           pop     r31
 1aa:   ef 91           pop     r30
 1ac:   9f 91           pop     r25
 1ae:   8f 91           pop     r24
 1b0:   3f 91           pop     r19
 1b2:   2f 91           pop     r18
 1b4:   0f 90           pop     r0
 1b6:   00 92 5f 00     sts     0x005F, r0
 1ba:   0f 90           pop     r0
 1bc:   1f 90           pop     r1
 1be:   18 95           reti
```

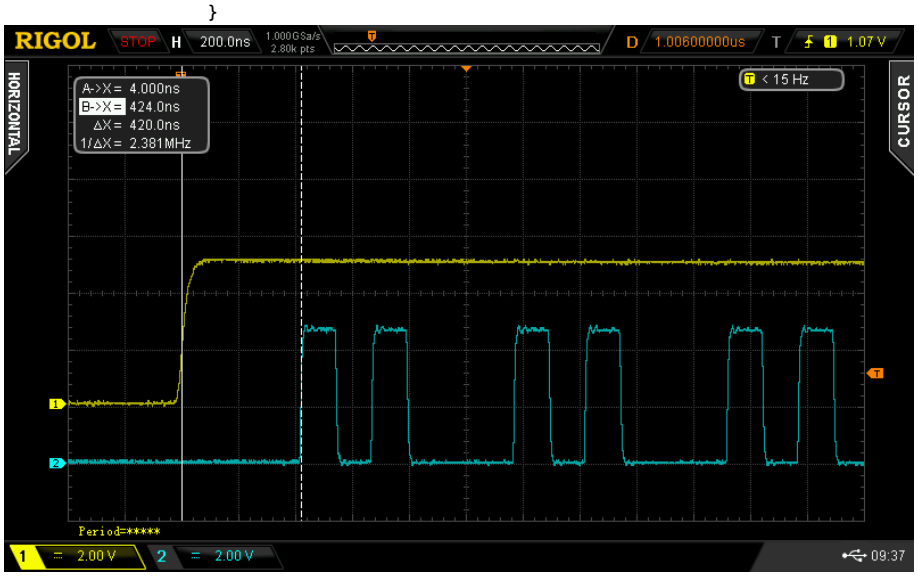
temps de latence 3 µs

2 Optimisation Un

2.1 Code polling généré opt1

10 lignes assembleur, difficile de faire mieux si on l'écrit en assembleur !

```
// polling check
                if (PIND &(1<<PD3)){
  a8:   4b 9b           sbis    0x09, 3 ; 9
  aa:   fe cf           rjmp    .-4             ; 0xa8 <main+0x18>
                        PORTB |= (1<<PB5);
  ac:   2d 9a           sbi     0x05, 5 ; 5
                        PORTB &= ~(1<<PB5);
  ae:   2d 98           cbi     0x05, 5 ; 5
                        PORTB |= (1<<PB5);
  b0:   2d 9a           sbi     0x05, 5 ; 5
                        PORTB &= ~(1<<PB5);
  b2:   2d 98           cbi     0x05, 5 ; 5
000000b6 <__vector_1>:
                }
```

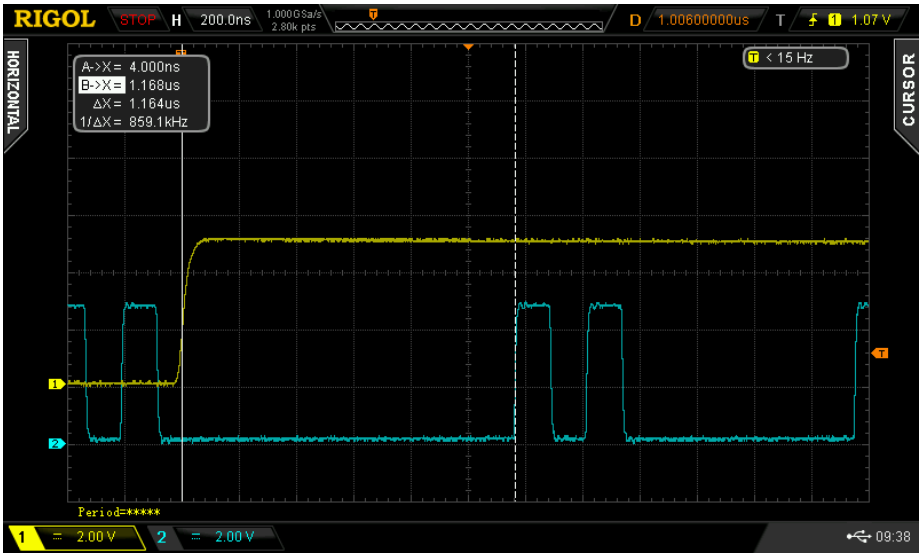


temps de latence 420 ns

2.2 code interruption généré opt1

26 lignes assembleur générées

```
ISR (INT0_vect){
  b6:   1f 92           push    r1
  b8:   0f 92           push    r0
  ba:   0f b6           in      r0, 0x3f       ; 63
  bc:   0f 92           push    r0
  be:   11 24           eor     r1, r1
        PORTB |= (1<<PB5);
  c0:   2d 9a           sbi     0x05, 5 ; 5
        PORTB &= ~(1<<PB5);
  c2:   2d 98           cbi     0x05, 5 ; 5
        PORTB |= (1<<PB5);
  c4:   2d 9a           sbi     0x05, 5 ; 5
        PORTB &= ~(1<<PB5);
  c6:   2d 98           cbi     0x05, 5 ; 5
}
  c8:   0f 90           pop     r0
  ca:   0f be           out     0x3f, r0       ; 63
  cc:   0f 90           pop     r0
  ce:   1f 90           pop     r1
  d0:   18 95           reti
```



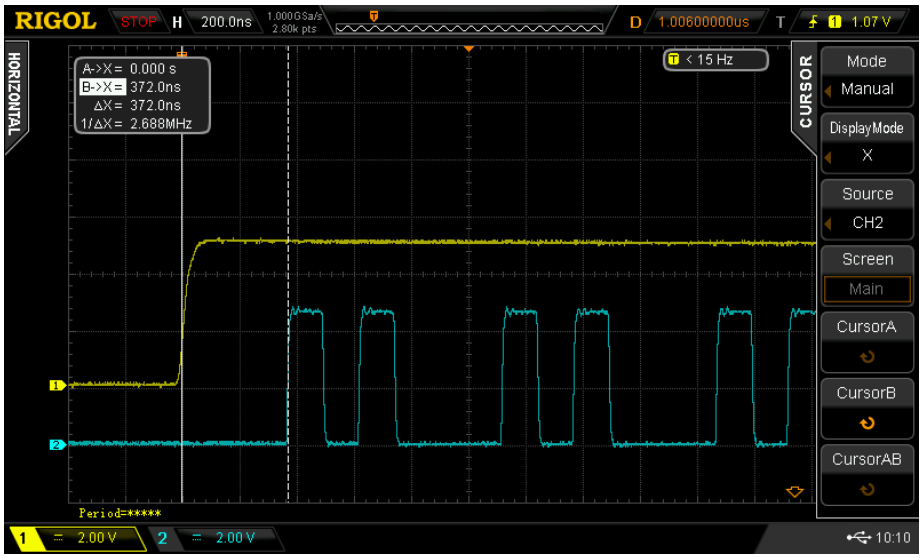
temps de latence 1,2 µs

3 Optimisation Deux

3.1 Code polling généré opt2

10 lignes assembleur générées

```
// polling check
              if (PIND &(1<<PD3)){
  c4:   4b 9b          sbis    0x09, 3 ; 9
  c6:   fe cf          rjmp    .-4             ; 0xc4 <main+0x18>
                       PORTB |= (1<<PB5);
  c8:   2d 9a          sbi     0x05, 5 ; 5
                       PORTB &= ~(1<<PB5);
  ca:   2d 98          cbi     0x05, 5 ; 5
                       PORTB |= (1<<PB5);
  cc:   2d 9a          sbi     0x05, 5 ; 5
                       PORTB &= ~(1<<PB5);
  ce:   2d 98          cbi     0x05, 5 ; 5
```

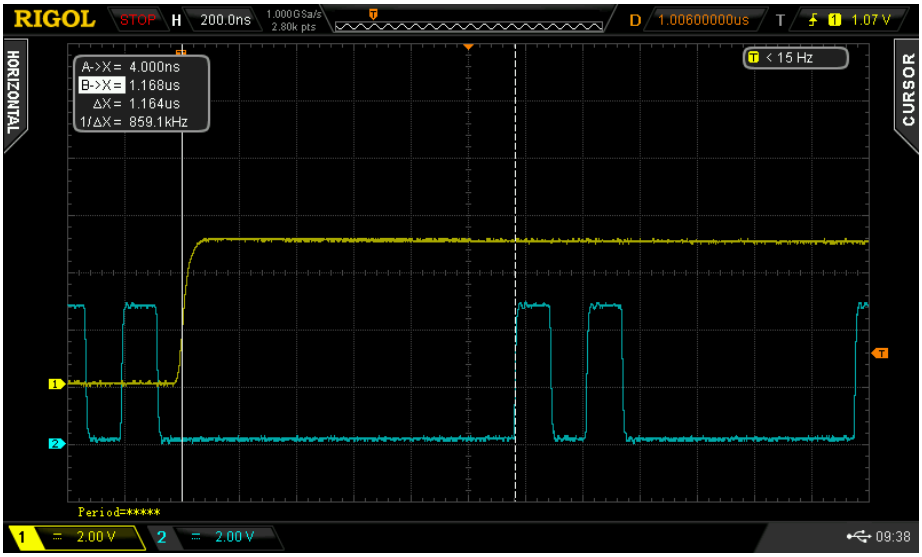


temps de latence 370 ns

3.2 code interruption généré opt2

26 lignes assembleur générées

```
ISR (INT0_vect){
  90:   1f 92           push    r1
  92:   0f 92           push    r0
  94:   0f b6           in      r0, 0x3f       ; 63
  96:   0f 92           push    r0
  98:   11 24           eor     r1, r1
        PORTB |= (1<<PB5);
  9a:   2d 9a           sbi     0x05, 5 ; 5
        PORTB &= ~(1<<PB5);
  9c:   2d 98           cbi     0x05, 5 ; 5
        PORTB |= (1<<PB5);
  9e:   2d 9a           sbi     0x05, 5 ; 5
        PORTB &= ~(1<<PB5);
  a0:   2d 98           cbi     0x05, 5 ; 5
}
  a2:   0f 90           pop     r0
  a4:   0f be           out     0x3f, r0       ; 63
  a6:   0f 90           pop     r0
  a8:   1f 90           pop     r1
  aa:   18 95           reti
```



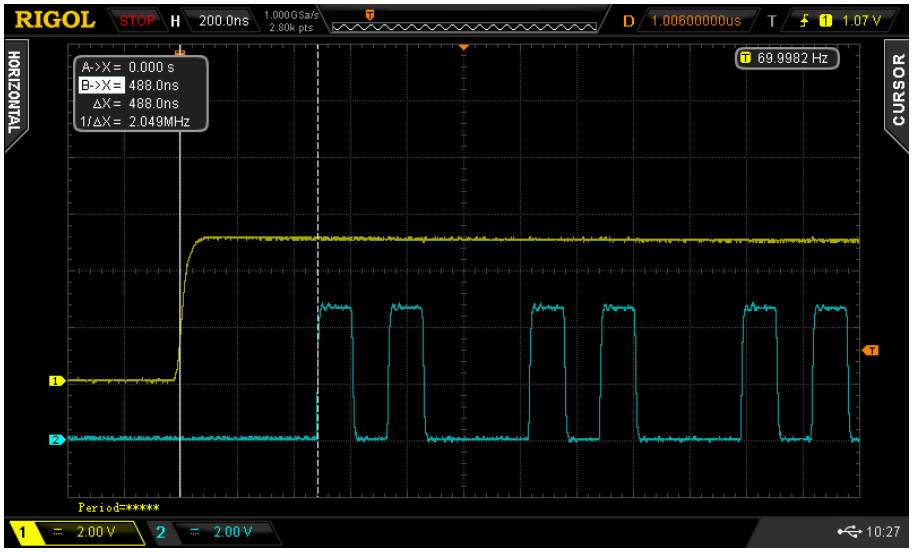
temps de latence 1,2 µs

4 Optimisation Trois

4.1 Code polling généré opt3

10 lignes assembleur générées

```
// polling check
              if (PIND &(1<<PD3)){
  c4:   4b 9b          sbis    0x09, 3 ; 9
  c6:   fe cf          rjmp    .-4             ; 0xc4 <main+0x18>
                       PORTB |= (1<<PB5);
  c8:   2d 9a          sbi     0x05, 5 ; 5
                       PORTB &= ~(1<<PB5);
  ca:   2d 98          cbi     0x05, 5 ; 5
                       PORTB |= (1<<PB5);
  cc:   2d 9a          sbi     0x05, 5 ; 5
                       PORTB &= ~(1<<PB5);
  ce:   2d 98          cbi     0x05, 5 ; 5
```

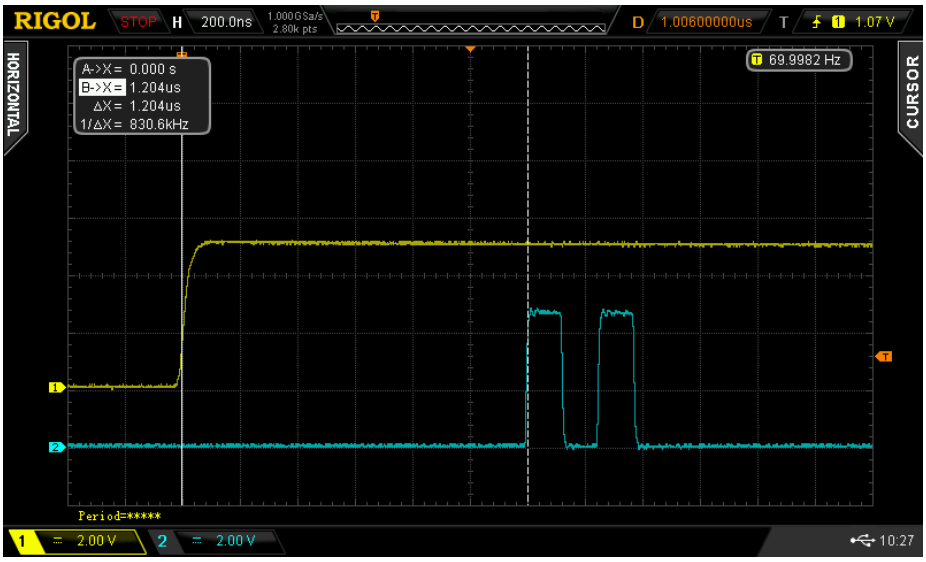


temps de latence 490 ns

4.2 code interruption généré opt3

26 lignes assembleur générées

```
ISR (INT0_vect){
  90:   1f 92           push    r1
  92:   0f 92           push    r0
  94:   0f b6           in      r0, 0x3f       ; 63
  96:   0f 92           push    r0
  98:   11 24           eor     r1, r1
        PORTB |= (1<<PB5);
  9a:   2d 9a           sbi     0x05, 5 ; 5
        PORTB &= ~(1<<PB5);
  9c:   2d 98           cbi     0x05, 5 ; 5
        PORTB |= (1<<PB5);
  9e:   2d 9a           sbi     0x05, 5 ; 5
        PORTB &= ~(1<<PB5);
  a0:   2d 98           cbi     0x05, 5 ; 5
}
  a2:   0f 90           pop     r0
  a4:   0f be           out     0x3f, r0       ; 63
  a6:   0f 90           pop     r0
  a8:   1f 90           pop     r1
  aa:   18 95           reti
```



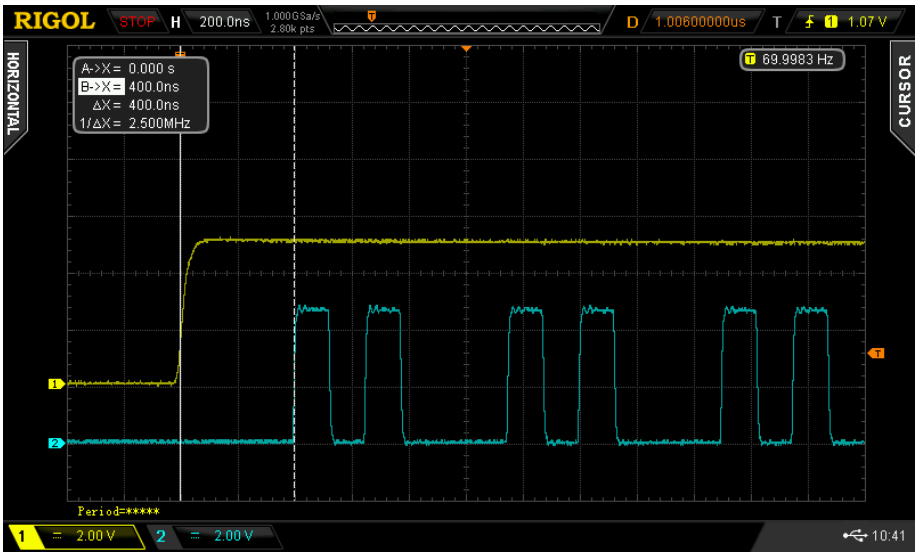
temps de latence 1,2 µs

5 Optimisation s (speed)

5.1 Code polling généré opt s

10 lignes assembleur générées

```
// polling check
              if (PIND &(1<<PD3)){
  c4:   4b 9b          sbis    0x09, 3 ; 9
  c6:   fe cf          rjmp    .-4             ; 0xc4 <main+0x18>
                       PORTB |= (1<<PB5);
  c8:   2d 9a          sbi     0x05, 5 ; 5
                       PORTB &= ~(1<<PB5);
  ca:   2d 98          cbi     0x05, 5 ; 5
                       PORTB |= (1<<PB5);
  cc:   2d 9a          sbi     0x05, 5 ; 5
                       PORTB &= ~(1<<PB5);
  ce:   2d 98          cbi     0x05, 5 ; 5
```

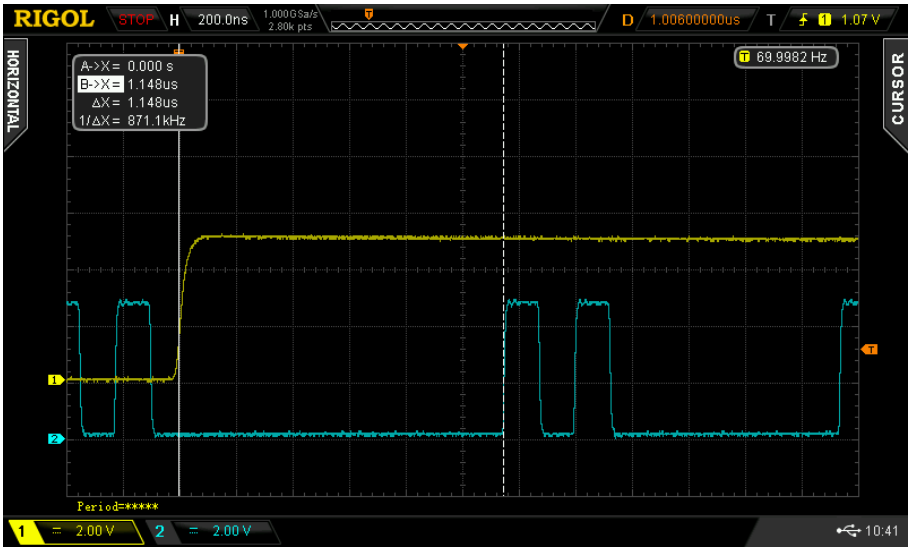


temps de latence 400 ns

5.2 Code interruption généré opt s

26 lignes assembleur générées

```
ISR (INT0_vect){
  90:   1f 92           push    r1
  92:   0f 92           push    r0
  94:   0f b6           in      r0, 0x3f       ; 63
  96:   0f 92           push    r0
  98:   11 24           eor     r1, r1
        PORTB |= (1<<PB5);
  9a:   2d 9a           sbi     0x05, 5 ; 5
        PORTB &= ~(1<<PB5);
  9c:   2d 98           cbi     0x05, 5 ; 5
        PORTB |= (1<<PB5);
  9e:   2d 9a           sbi     0x05, 5 ; 5
        PORTB &= ~(1<<PB5);
  a0:   2d 98           cbi     0x05, 5 ; 5
}
  a2:   0f 90           pop     r0
  a4:   0f be           out     0x3f, r0       ; 63
  a6:   0f 90           pop     r0
  a8:   1f 90           pop     r1
  aa:   18 95           reti
```



temps de latence 1,15 µs

6 Conclusions

Avec une carte arduino nano pro cadencée par un quartz à 16 MHz et configurée par les fuses pour sa vitesse max (62,5 ns par cycle horloge système), et pour le programme de test utilisé

- **le temps de latence par scrutation est de 400 ns pour les optimisations de compilation de 1 à s**
- **le temps de latence par interruptions est de 1,2µs pour les optimisations de compilation de 1 à s**
- **le temps de latence par scrutation est de 1,2 µs pour optimisation de compilation zéro**
- **le temps de latence par interruptions est de 3 µs pour optimisation de compilation zéro**

Ces temps de latence sont multipliés par 8 lorsque le bit CKDIV8 du lfuse est selectionné (cas de la livraison standard Atmel et des cartes avec résonateur céramique)

La scrutation est le moyen le plus réactif de répondre à une sollicitation externe, mais peut demander une machine d'état pour pouvoir faire autre chose ou autre technique potentiellement dévoreuse de ressources. L'incertitude est dans les deux cas d'une demi-période d'horloge système, ici ± 31,25 ns.

L'environnement Arduino, qui s'adresse à des utilisateurs qui n'ont pas forcément de connaissances suffisantes, utilise peut-être des protections maximales au détriment de la vitesse, à confirmer.

Code c

```
/* processor = mega328
 *            ISR tester
 *
 * atmega 328p arduino nano pro
 * crystal clock 16 MHz
 * compiler avr-gcc 4.3.5

 *
 * PB0  D8
 * PB1  D9      OC1A
 * PB2  D10     OC1B
 * PB3  D11     MOSI           ICSP    1
 * PB4  D12     MISO           ICSP    9
 * PB5  D13     SCK            ICSP    7       LED
 *
 * PC0  A0
 * PC1  A1
 * PC2  A2
 * PC3  A3
 * PC4  A4
 * PC5  A5
 * PC6  -       RESET          ICSP    5
 *
 * ADC6 A6
 * ADC7 A7
 *
 * PD0  RXI
 * PD1  TXO
 * PD2  D2      INT0
 * PD3  D3      INT1           polling input
 * PD4  D4                     INT0 flag
 * PD5  D5
 * PD6  D6      OC0A
 * PD7  D7
 *
 * RMZ # 224
 *
 * lfuse FF    hfuse DA       efuse 00 arduino nano pro (delivery fuses)
 *
 * status:
 * keywords: timer1 ISR, INTO ISR, switch, USART
 *
 * Zibuth27 2016/05/29
 */

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define        TEMPS_RETOUR   45000


volatile uint8_t locked =0;


int main (void)       {

// variables

// ports
        DDRB = 0xff;                                    // includes on-board LED
        DDRC = 0x02;
        PORTC |= (1<<PC2);                              // pull-ups
        DDRD = 0x00;                                    // inputs

// timer0

// timer1

// timer2
```

```
//USART
/*      UCSR0B |= (1<<TXEN0);
        UCSR0C |= (1<<UCSZ01)|(1<<UCSZ00);          // 8 bit
        UBRR0H = 0;
        UBRR0L = 103;
*/

// IRQs
        EICRA = 0x03;                               // INT0 rising edge
        EIMSK = 0x01;                               // INT0 IRQ enabled

        sei();                                      // enable interrupts


// main loop
        while (1){
// polling check
                if (PIND &(1<<PD3)){
                        PORTB |= (1<<PB5);
                        PORTB &= ~(1<<PB5);
                        PORTB |= (1<<PB5);
                        PORTB &= ~(1<<PB5);
                }

        }
  return 0;
}
ISR (INT0_vect){
        PORTB |= (1<<PB5);
        PORTB &= ~(1<<PB5);
        PORTB |= (1<<PB5);
        PORTB &= ~(1<<PB5);
}
```

