

MICROCONTRÔLEURS MICROCONTRÔLEURS ET SYSTÈMES NUMÉRIQUES TRAVAIL PRATIQUE NO 3

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

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 signaux générés aux ports du microcontrôleur, ainsi que la méthode de développement/déverminage basée sur l'analyse des signaux temporels.

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. Les trois des fanions les plus utilisés sont:

- C qui est l'abréviation de **carry** et qui signale **le dépassement de capacité** **après une additions ou soustraction non signé**,
- Z qui est l'abréviation de **zero** et qui signale **que le résultat d'une opération est égal à zero**,
- N qui est l'abréviation de **négative** et qui signale **que le résultat d'une opération est négatif**.

Ecrivez le programme qui vous permet de simuler les opérations données en Table 3.1 en assumant tous les fanions à zero. 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 (complément à deux, signé)
0xec+0x32	0x1e	<input type="checkbox"/>	X	adc	non				

Table 3.1: Opérations arithmétiques.

Opération	Résultat	H	S	V	N	Z	C	Instruction à utiliser	Validité du résultat (complément à deux, signé)
0x3a-0xec	0x4e	X					X	sbc	non
0xf0-0xfd	0xf3	X	X		X		X	sbc	oui
0x39-0xa7	0x92			X	X		X	sbc	oui

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:

```

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 allumé

la led 1 est toujours éteinte

les autres led s'allume si on appuie sur leurs bouton respectif

- quel type de masque est réalisé par l'instruction ori ? force des bits à 1
- quel type de masque est réalisé par l'instruction andi ? force des bits à zero

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, (<input type="text" value="1"/>	<<	<input type="text" value="7"/>) + (<input type="text" value="1"/>	<<	<input type="text" value="3"/>)
ori	r16, 0b	<input type="text" value="10001000"/>						;	
ori	r16, 0x	<input type="text" value="88"/>						;	
ori	r16, 128+	<input type="text" value="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, [~	((<input type="text" value="1"/>	<<	<input type="text" value="2"/>) + (<input type="text" value="1"/>	<<	<input type="text" value="4"/>))
andi	r16, 0b	<input type="text" value="11101011"/>						;			
andi	r16, ~0x	<input type="text" value="14"/>						;			
andi	r16, 0xff-0x	<input type="text" value="10"/>						-0x	<input type="text" value="4"/>	;	

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.

```

; 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.

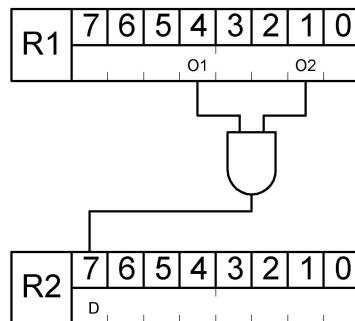


Figure 3.4: Opération Booléenne sur des bits singuliers.

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 copie sur le bit b du registre reg sur le flag T,
- l'instruction bld reg,b qui copie le flag T sur le bit b du registre reg.

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
    bst @2, @3
    bld @0, @1
.endmacro

```

Figure 3.5: Macro MOVB.

Complétez le programme donné en Figure 3.6 afin qu'il effectue une copie de la valeur de LED2 sur LED6 (après acquisition des boutons-poussoirs afin de disposer d'une valeur changeant dynamiquement).

```
.include "macros.asm"

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

loop:in       r0,PIND
    MOVB    r0,6,r0,2
    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 étudiez-le.

```
.include "macros.asm"

ANDB      r2,7, r1,1, r1,4
; newline is mandatory with this assembler
```

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 68.94	set		: met T à 1
00000001 14.fe	sbrs	r1,4	: saute la prochaine instruction si le bit 4 de r1 est à 1
00000002 e8.94	clt		: met T à 0
00000003 11.fe	sbrs	r1,1	
00000004 e8.94	clt		
00000005 27.f8	bld	r2,7	: copy la valeurs 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 stocké la valeur d'un bit (buffer),
- 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 instructions 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.

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, @1
.endmacro;	

Figure 3.9: Diagramme de flux de la macro ANDB.

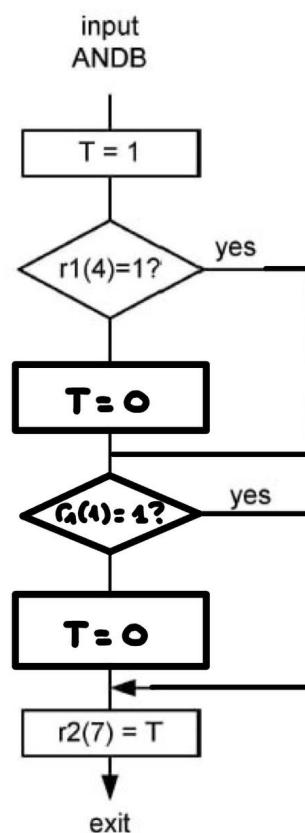


Figure 3.10: Macro NORB.

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 synchronisation 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 target ATmega128L-4MHz-STK300
; purpose port switching for training oscilloscope operation
; module: none, output port: PORTE

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 7 in i/o port E - [2] cycles [500] ns
    cbi PORTE,7            ; clear bit 7 in i/o port E - [2] cycles [500] ns
    nop                   ; No OPeration (do nothing)-[1] cycles [250] ns
    push r0                ; push -[2] cycles [500] ns
    pop r0                 ;pop -[2] cycles [500] ns
    rjmp main              ; jump back to main -[2] cycles [500] ns

```

Figure 3.11: pulsout1.asm.

Ainsi, l'exécution de la boucle main nécessite [11] cycles, soit [2750] 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, 2V et 1.0 μ s;
- trigger sur CH1, flanc montant à 2.0V;
- au moyen du bouton HORIZONTAL POSITION de la trace, modifiez la position du déclenchement du trigger jusque sur la gauche, 2 μ s après le début de la trame;
- mettre la sonde de l'oscilloscope sur “10X.”

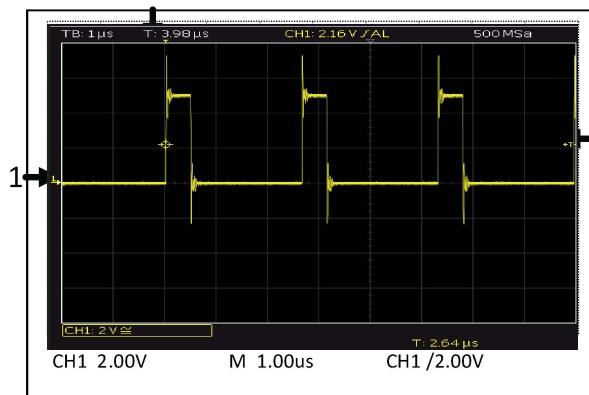


Figure 3.12: Observation de l'exécution de pulsout1b.asm.

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 simplifier les mesures):

- Quelle est la durée de l'impulsion générée ? [500 ns].
- Quelle est la période du signal ? [2640 ns].
- Quel en est le rapport cyclique ? [18.30%].
- Quelle est la fréquence du signal ? [377 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 transition, 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/MEASURE; il faut choisir entre Time et Voltage puis utiliser les boutons TIME DIV→POSITION pour les positionner.

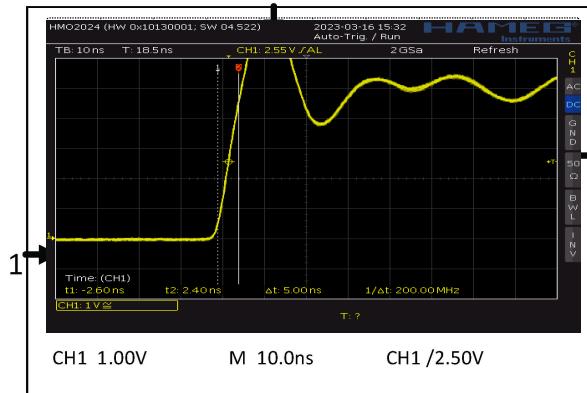


Figure 3.13: Flanc montant, temps de montée=environ 5.0 ns.

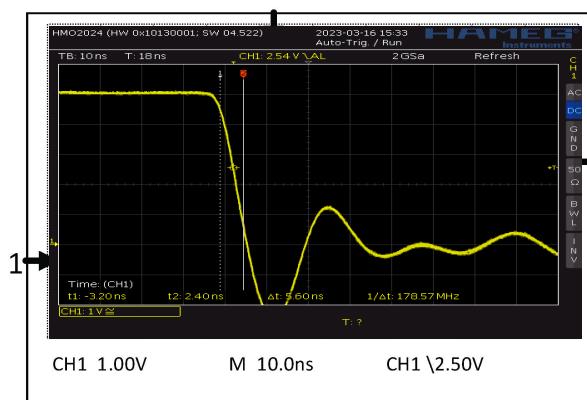


Figure 3.14: Flanc descendant, temps de descente=environ 5.6 ns.

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 microseconde car cbi prend 2 clock

Quelle est la résolution temporelle, et pourquoi ?

0.25 micro seconde car le nop ne prend qu'une clock et on peut jouer avec pour ajuster le timming

