

## MICROCONTRÔLEURS MICROCONTRÔLEURS ET SYSTÈMES NUMÉRIQUES TRAVAIL PRATIQUE NO 9

MT GROUPE A || MT Groupe B || EL

No du Groupe	Premier Etudiant	Second Etudiant	Evaluation	Visa Correcteur
--------------	------------------	-----------------	------------	-----------------

**093** Nathann Morand Felipe Ramirez  
**9.** INTERFACE I<sup>2</sup>C AVEC EEPROM

Ce travail pratique voit l'étude du protocole I<sup>2</sup>C sur un exemple de communication avec une EEPROM.

### 9.1 LE STANDARD I<sup>2</sup>C

Le standard I<sup>2</sup>C est un mode de communication s[érie] s[ynchrone], c'est-à-dire qu'une ligne transmet l'horloge. Ce standard utilise les deux lignes:

- SDA qui signifie [Serial Data/Address input/output], et
- SCL qui signifie [Serial Clock].

Plusieurs modules périphériques peuvent être connectés simultanément sur ces lignes. Pour garantir l'intégrité des données transmises, l'étage d'accès aux lignes d'un module comprend un circuit connecté en c[ollecteur] o[uvert].

Afin que les lignes ne restent pas en haute impédance lorsqu'aucun module n'y accède, chaque ligne SDA et SCL est connectée individuellement à VDD par des [résistance de pull up].

Sur un bus I<sup>2</sup>C, il est nécessaire de configurer un module en tant que maître dont le rôle est de de gérer la communication avec chaque module esclave pour éviter les collision ainsi que un ou plusieurs modules (au minimum un) nommés es[claves].

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

Condition de start:

un flanc descendant sur SDA et SCL à 1

Condition de stop:

flanc montant sur SDA et SCL à 1



Figure 9.1: Conditions de start et stop pour la transmission d'un octet par protocole I<sup>2</sup>C.

Ansi, les conditions de start et stop se diffèrentent de la transmission des données au niveau du timing du protocole par le fait que pour le start c'est le master qui desend SDA avant la clock la ou le stop c'est le device qui parle qui va relacher le SDA une fois la clock valider.

Deux différents types d'étages de sortie sont décrit en Figure 9.2. Complétez sur la figure les connections électriques manquantes; connectez les sorties ensemble en un bus; ajoutez la résistance de pull-up dans le cas du circuit en 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.

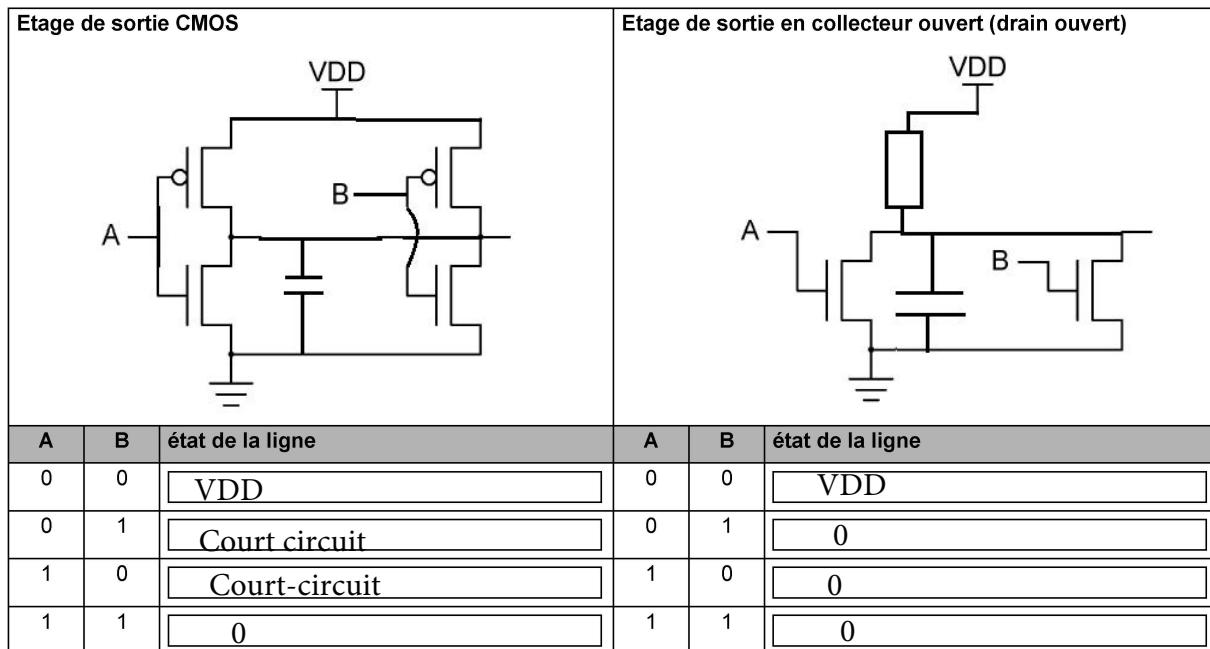


Figure 9.2: Comparaison entre étages de sortie CMOS et à collecteur ouvert.

Téléchargez le programme i2cx\_1.asm donné en Figure 9.4 qui teste les quatre macros SCL0, SDA0, SCL1, et SDA1. Placez le module M5 sur le port B. Vérifiez la connexion correcte des jumpers sur la carte, qui doit être comme indiqué en Figure 9.3. La configuration des jumpers peut être modifiée si l'affectation de SDA et SCL sur les ports du MCU sont aussi modifiés dans le fichier i2cx.asm.



Figure 9.3: Position des jumpers de programmation et des points de mesure sur le module M5.

Connectez la sonde 1 de l'oscilloscope avec SCL (PM8), et la sonde 2 avec SDA (PM9).

Reportez en Figure 9.5 les signaux observés, et indiquez les instructions correspondant aux différentes phases.

---

```

; file      i2cx_1.asm      target ATmega128L-4MHz-STK300
; purpose observe I2C signals using oscilloscope
; module: M5, output port: PORTB
.include "macros.asm"
.include "definitions.asm"

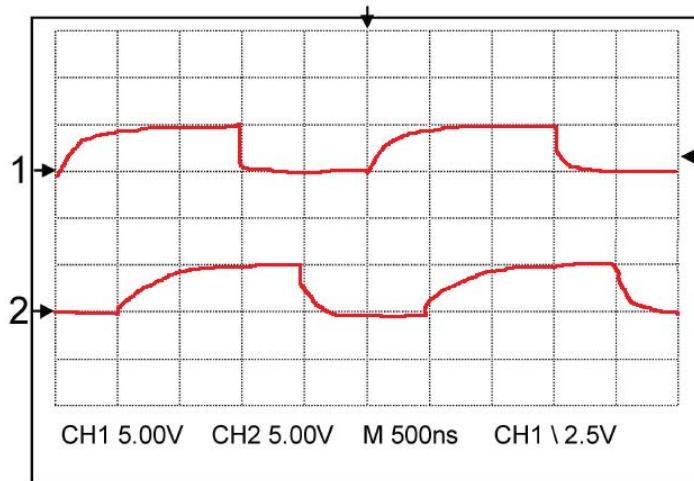
reset: rjmp    main
.include "i2cx.asm"

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

```

---

*Figure 9.4: i2cx\_1.asm.*



*Figure 9.5: Signaux observés lors de l'exécution de i2cx\_1.asm.*

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

On a un condensateur qui se charge en $1/RC$ alors qu'il se décharge via un mosfet qui à une résistance interne très faible donc presque instantanément

Ajoutez la ligne suivante, et téléchargez à nouveau:

```

reset: OUTI      PORTB, 0xff
      rjmp    main

```

Expliquez pourquoi les lignes ne transmettent plus rien.

car les lignes I2C sont au repos à VCC à cause des pull-up donc les mettre à 1 ne modifie pas l'état des lignes

Ainsi, pour émuler (simuler par le matériel) une sortie en collecteur ouvert, 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)	DDR <sub>x</sub>	port en entrée/ sortie	PORT <sub>x</sub>
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 (DDR<sub>x</sub>) et la valeur de port (PORT<sub>x</sub>) de la bonne manière. Elle doit être appelée avant toute utilisation du microcontrôleur en émulation de communication par I<sup>2</sup>C.

Au début du programme la constante symbolique SDA\_port=PORTB est définie. Dans le programme on accède au registre SDA\_port-1 (“SDA\_port moins 1”). Quel registre est alors adressé ? **DDRB**.

## 9.2 EMISSION D’UN BYTE SUR UN BUS I<sup>2</sup>C

Téléchargez le programme i2cx\_3.asm donné en Figure 9.6. 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 (il n’y a pas de condition de stop).

---

```

; file      i2cx_3.asm    target ATmega128L-4MHz-STK300
; purpose   I2C start condition test
; module: M5, output port: PORTB, PORTC
; misc: flexible cable from LEDs to PORTC
.include "macros.asm"
.include "definitions.asm"

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

```

---

Figure 9.6: i2cx\_3.asm.

Connectez le câble plat de sorte à ce que les LEDs soient connectées au PORTC comme présenté en Figure 9.7, car le PORTB est utilisé par le protocole I<sup>2</sup>C.

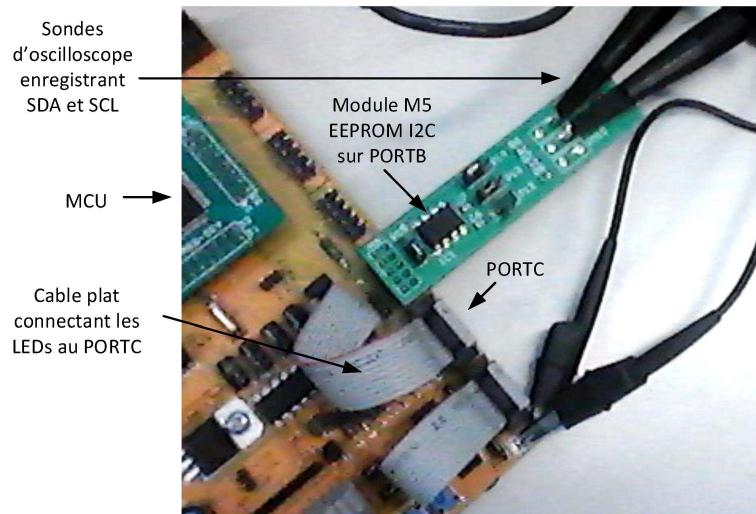


Figure 9.7: Connection des LEDs sur le PORTC, alors que le module M5 occupe le PORTB.

L'envoi d'un seul bit est fait par la macro I2C\_BIT\_OUT reproduite en Figure 9.8. 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 cycles
.endmacro

```

---

Figure 9.8: Macro I2C\_BIT\_OUT.

Etudiez le déroulement de la macro, et reportez en Figure 9.9 les signaux attendus avec leurs timings correctes, et les instructions correspondantes. Hypothèse: a0 = 0xff.

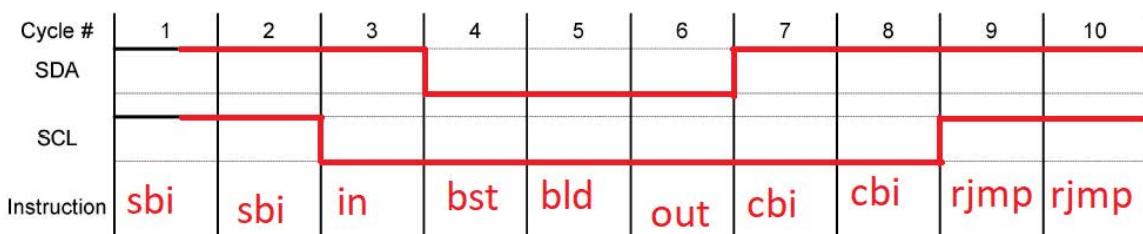


Figure 9.9: Timings de la macro I2C\_BIT\_OUT.

Exécutez le programme i2cx\_3.asm 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 13.10 du cours, pour la transmission de la donnée 0x5a = 0b01011010.

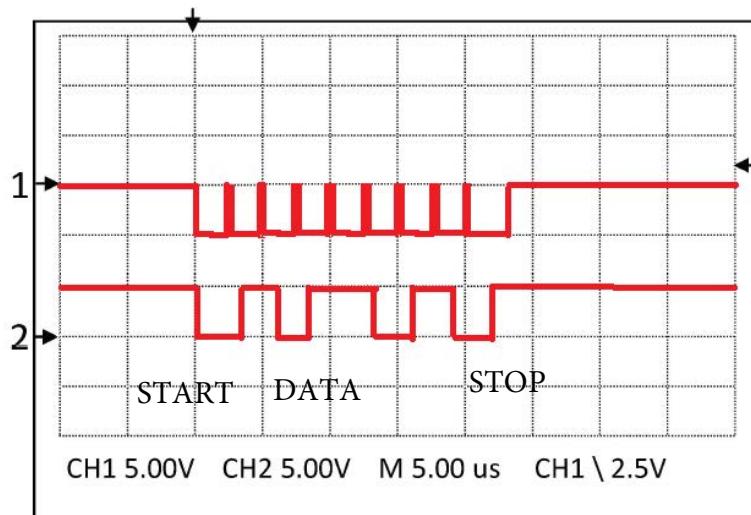


Figure 9.10: Exécution avec 0x5a comme donnée transmise.

Quelle est l'information affichée par la LED PC7 (LEDs sur PORTC), et que signifie le fait qu'elle soit allumée, est-ce normal ?

la led z s'éteint quand on reçois une confirmation de reception de la part d'un des modules esclave. Si on ne connecte pas d'esclave, il ne pourra pas répondre et donc la led reste allumé

### 9.3 ADRESSAGE D'UN MODULE

Le protocole I<sup>2</sup>C permet l'adressage de cent vingt-sept esclaves différents. L'adresse d'identification de l'EPPROM est 0b[1010xxx] 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 fabricant qui choisit et la mets dans le silicium. La même pour toutes les devices de la même série]. Où peut-on trouver la valeur de cette adresse ? [dans le datasheet...];
- 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 ? [on peut la programmer à l'aide de pin];
- ainsi, il est possible d'adresser une nombre maximal de [8] EEPROMs M24C64 sur un même bus I<sup>2</sup>C.

Le premier byte envoyé par le maître est l'adresse du module I<sup>2</sup>C avec lequel il désire communiquer. L'unique module présent sur le bus qui reconnaît son adresse répond par un acknowledge, et pratiquement tire la ligne vers le niveau bas.

Téléchargez le programme i2cx\_4.asm donné en Figure 9.11.

Exécutez le programme sur le système cible, et reportez sur la Figure 9.12 les signaux observés à l'oscilloscope. Indiquez sur le diagramme les trois phases du protocole visibles, conformément à la Figure 13.10 du cours, pour la transmission de la donnée 0xa0 = 0b[10100000].

---

```

; file      i2cx_4.asm      target ATmega128L-4MHz-STK300
; purpose  I2C EEPROM address testing
; module: M5, output port: PORTB
.include "macros.asm"
.include "definitions.asm"

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

.include "i2cx.asm"

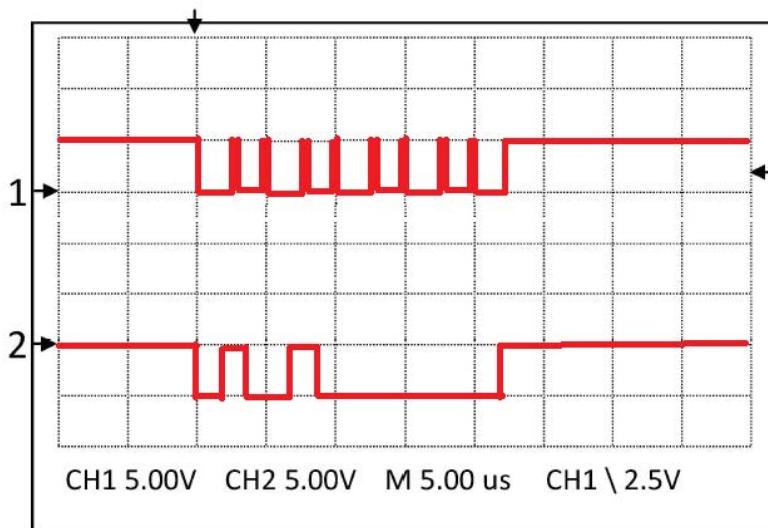
main:
       CA      i2c_start, 0b10100000
       rcall   i2c_stop
       WAIT_US 10

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

```

---

*Figure 9.11: i2cx\_4.asm.*



*Figure 9.12: Exécution avec 0xa0 comme adresse transmise.*

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

Commentez les lignes d'adressage du haut, et enlevez les signes de commentaire sur les lignes d'adressage du bas. Répétez l'expérience et reportez le résultat en Figure 9.13.

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

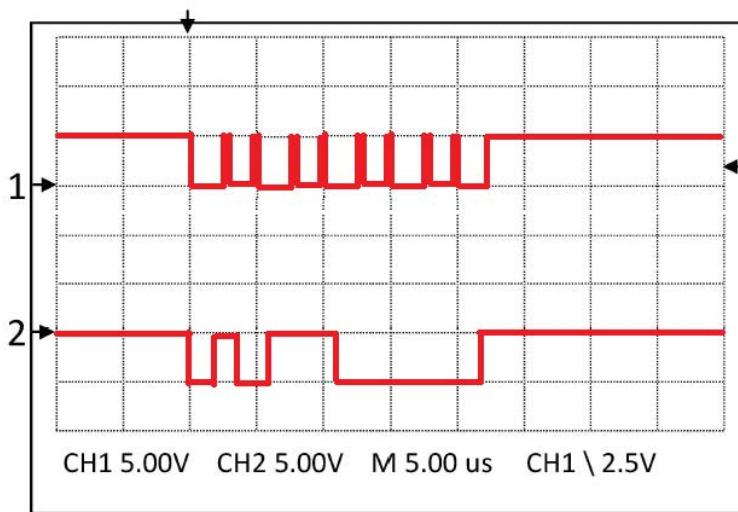


Figure 9.13: Exécution avec 0xb0 comme adresse transmise.

#### 9.4 ANALYSE DE LECTURE/ÉCRITURE DANS L'EEPROM

Téléchargez le programme objet i2c\_eeprom2017.hex qui permet de communiquer avec l'EEPROM M24C64 par protocole I<sup>2</sup>C. L'EEPROM comprend 64kb de mémoire, soit █ 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 9.14.

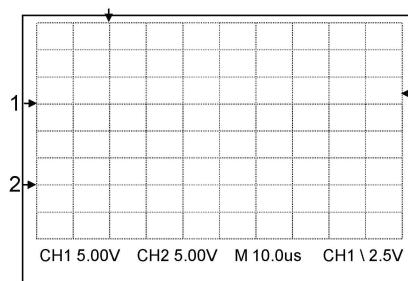


Figure 9.14: Configuration de l'oscilloscope.

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 9.15 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 l'acquisition de trace en single shot.

PHASE #	PHASE1	PHASE2	PHASE3	PHASE4	PHASE5	PHASE6	PHASE7	PHASE8	PHASE9
DATA / PHASE NAME	start	10100000	ack	00000000	ack	00000000	ack	10100001	ack
M/S TALKING	master	slave	master	slave	master	slave	master	slave	
Read/Write Mode	-	write	-	write	-	write	-	write	-
		PHASE10	PHASE11	PHASE12	PHASE13	PHASE14	PHASE15	PHASE16	
DATA / PHASE NAME	01010101	ack	00110011	ack	00001111	no ack	STOP		
M/S TALKING	slave	master	slave	master	slave	master	master		
Read/Write Mode	read	-	read	-	read	-	-		

Figure 9.15: Analyse de lecture/écriture par I2C avec l'EEPROM M24C64.

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

le programme écrits 4 donné dans d'eprom puis lit 3 donné en retour.

