

P.P.E. ROBOT SOLAIRE



(Bus I2C)



Sommaire

- A - Présentation fonctionnelle de l'interface TWI de l'ATMega8535
- B - Driver « TWI I2C Master »
 - B1 Envoi d'un message sur le bus I²C par l'intermédiaire de l'interface
 - B2 Réception d'un message sur le bus I²C par l'intermédiaire de l'interface TWI
 - B3 Liste (partielle) des fonctions du driver « Maître TWI »
 - B4 Fichiers nécessaire au driver « Maître TWI »
- C - Commande des périphériques I2C avec les fonctions du driver « TWI I2C Master »
 - C1 Port d'entrées sorties PCF8574
 - C2 LCD « BATRON »
 - C3 Carte « MD23 »
 - C4 Boussole « HMC6352 »
 - C5 Capteur « I2C »

Annexes

- Annexe 1 - Fonctions du langage « C »
- Annexe 2 - Fonctions de la bibliothèque SSI « SSI.h »
- Annexe 3 - Fichier d'en-tête pour la carte MD23 « MD23.h »
- Annexe 4 - Mesures faites sur le bus I2C

Bibliographie

- Datasheet ATMEGA8535
 - AVR155 : Accessing an I2C LCD Display using the AVR 2-wire Serial Interface
 - AVR315 : Using the TWI module as I2C master
-

A) Présentation fonctionnelle de l'interface TWI de l'ATMega8535

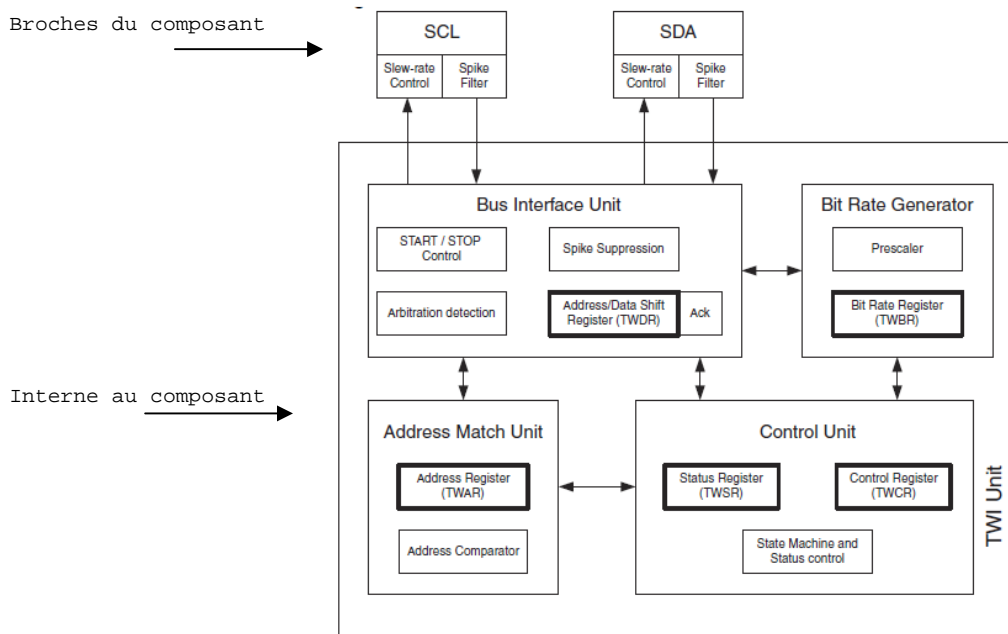
Les microcontrôleurs ATMEL 8 bits possèdent un périphérique dit **Two Wire Interface** « TWI » permettant de gérer une liaison I²C jusqu'à 400kb/s. Cette interface peut être utilisée en mode « **Maître** » ou en mode « **Esclave** ». Dans chacun des modes, elle peut se comporter comme un **émetteur** ou un **récepteur**.



Terminologie

Terme	Description
Maître	Le circuit qui initialise, termine la transmission et génère le signal d'horloge.
Esclave	Le circuit adressé par le maître.
Emetteur	Le circuit plaçant les données sur le bus.
Récepteur	Le circuit lisant les données sur le bus.

Organisation de l'interface TWI (Doc ATMEL AVR315)



Rôle des unités fonctionnelles

Bus Interface Unit

Contrôle la communication (génération ou détection de START, REPEATED START et STOP).

Bit rate Generator

Règle la période du signal SCL.

Address Match Unit

Utilisée uniquement en mode "esclave".

Control Unit

Gère le fonctionnement de l'ensemble.

Que ce soit en mode « Maître » ou « Esclave », l'utilisation de cette interface nécessite un **driver**. Le paragraphe suivant décrit le driver « **Maître** » utilisé par notre application.

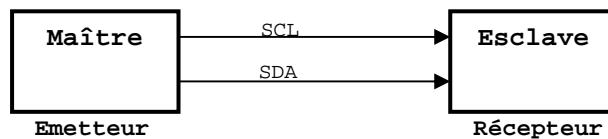
B) Driver « TWI I²C Master »

La note d'application **AVR315** d'ATMEL décrit un « driver maitre I²C » destiné au cross compilateur IAR Embedded Workbench. Ce driver a été adapté pour le cross compilateur CodeVision AVR.

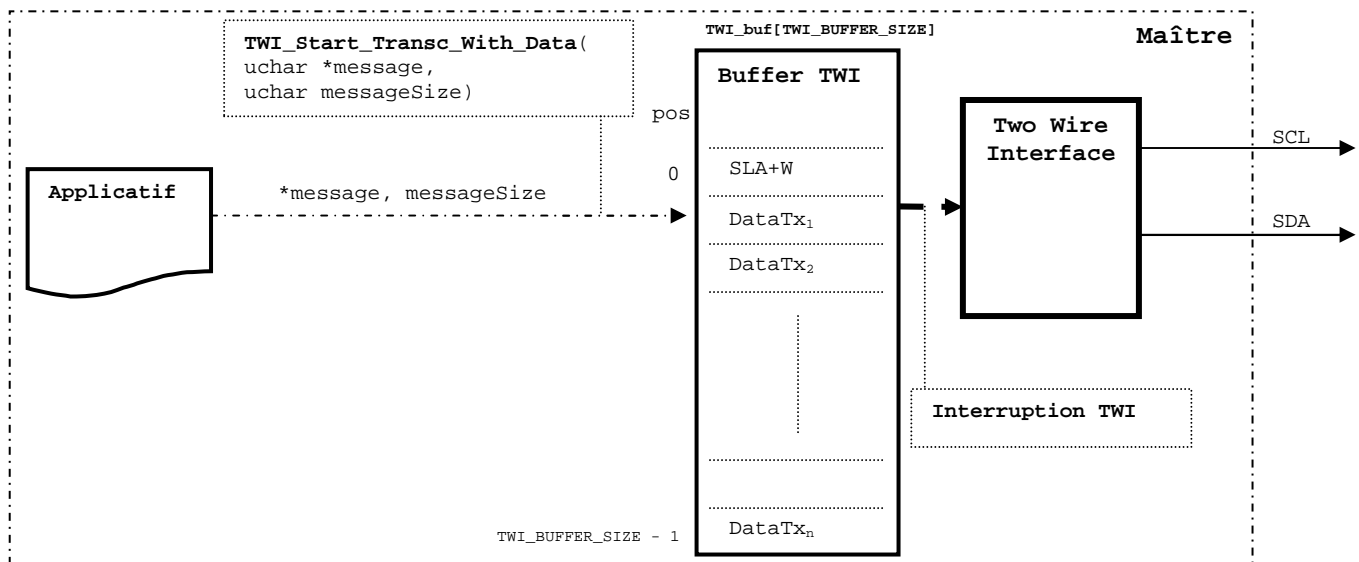
A l'aide de fonctions, ce driver gère un **buffer** (tableaux de caractères) servant d'interface entre le périphérique TWI et le programme applicatif. La suite de ce paragraphe décrit les différents modes d'utilisation de ces fonctions.

B1) Transmission d'un message à un esclave

Sens de la communication : Le maître transmet des données à l'esclave



Gestion de la communication



Hardware

TWI_buf : Buffer TWI déclaré dans le fichier TWI_Master.h (zone de la mémoire SRAM réservée pour placer les octets envoyés à l'esclave ou reçus de l'esclave)

TWI_BUFFER_SIZE : Taille du buffer TWI_buf. Déclaré dans le fichier TWI_Master.h (initialisé à 4 par défaut). A redéfinir dans le programme applicatif, avant l'appel de la bibliothèque, en fonction du besoin.

Software

TWI_Start_Transc_With_Data : Fonction du driver TWI permettant d'envoyer ou de recevoir des informations par l'intermédiaire du bus I²C.

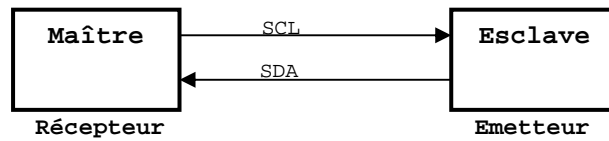
message : Tableau contenant le message, composé de l'adresse de l'esclave + le bit de lecture/écriture positionné en écriture et les données à transmettre à l'esclave {**SLA+W**, **DataTx₁**, ..., **DataTx_n**}. Ce tableau est copié dans le buffer TWI par la fonction précédente.

messageSize : taille du tableau « message »

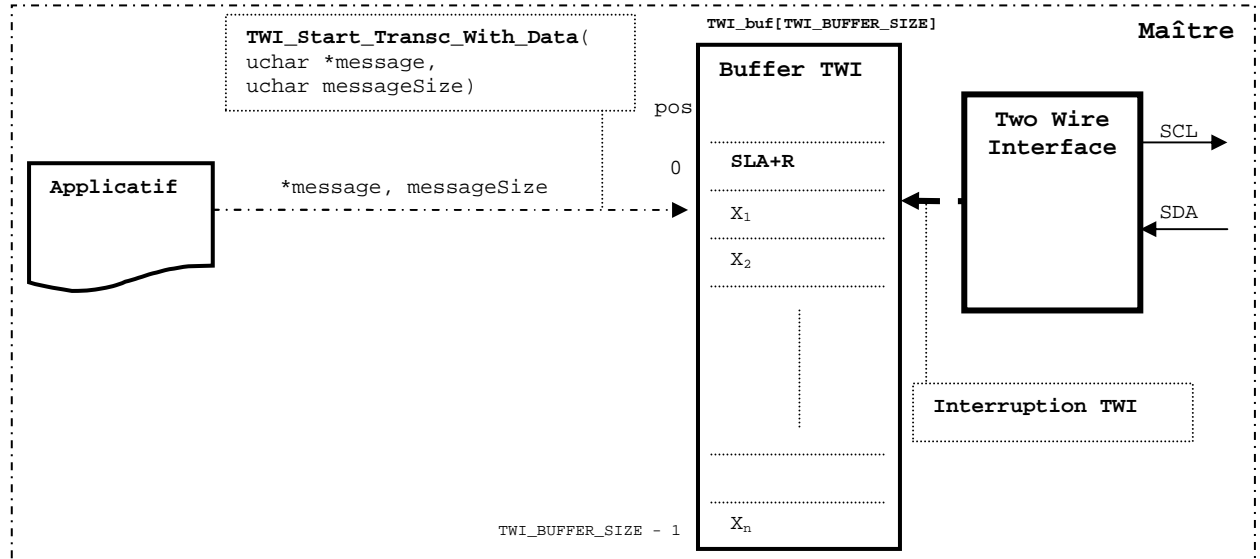
Interruption TWI : Cette fonction du driver TWI se charge de placer les valeurs sur le bus en respectant le protocole I²C.

B2) Réception d'un message placé sur le bus par un esclave

Sens de la communication: Le maître reçoit les données transmises par l'esclave.



Gestion de la communication



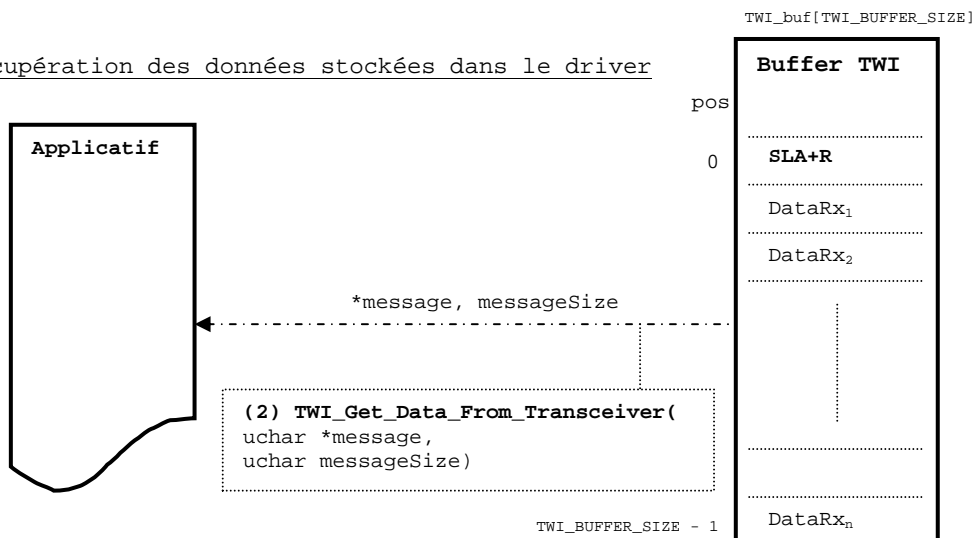
message : Tableau contenant l'adresse de l'esclave + le bit de lecture écriture positionné en lecture et la « place » nécessaire au stockage du message {SLA+R, X₁, ..., X_n}. X = indifférent.

messageSize : taille du tableau « message »

Interruption TWI : Cette fonction du driver TWI se charge de « récupérer » les valeurs placées par l'esclave sur le bus en synchronisme avec l'horloge générée par le maître.

Remarque : Le nombre de données à lire dépend de l'esclave interrogé !

Récupération des données stockées dans le driver



A la fin de la communication, le buffer TWI contient les données transmises par l'esclave. L'utilisation de la fonction TWI_Get_Data_From_Transceiver permet de recopier le contenu du Buffer dans le tableau « message ».

message : Tableau, image du buffer, contenant l'adresse de l'esclave + le bit de lecture écriture positionné en lecture et les données transmises par l'esclave de {SLA+R, DataRx₁, ..., DataRx_n}

messageSize : taille du « message ».

Attention : les informations reçues se situent à partir de la position 1 du buffer.

B3) Liste (partielle) des fonctions du driver « Maître TWI »

Nom	Description
TWI_Master_Initialise()	Appeler cette fonction pour initialiser l'interface TWI et la placer en mode « Attente ». Penser à autoriser les interruptions après l'initialisation.
TWI_Start_Transc_With_Data(uchar *message, uchar messageSize)	<p>Appeler cette fonction pour envoyer un message préparé à un esclave ou lire les données fournies par un esclave.</p> <p>Le premier octet de la table « message », située en SRAM, doit contenir l'adresse de l'esclave plus le bit de lecture/écriture (SLA+R ou SLA+W). Les octets suivants contiennent les valeurs à placer dans le buffer TWI_Buff si la fonction est utilisée en écriture ou un nombre suffisant d'emplacements si elle est utilisée en lecture.</p> <p>En lecture, les informations reçues sur le bus I²C ne sont pas stockées dans la table « message » mais dans le buffer TWI_Buff (voir schéma de principe). Il faut utiliser la fonction TWI_Get_Data_From_Transceiver pour transférer les valeurs reçues du buffer TWI_Buff vers la table « message ».</p> <p>« messageSize » contient le nombre total d'octets à transmettre ou à recevoir.</p>
TWI_Get_Data_From_Transceiver (uchar *message, uchar messageSize)	<p>Appeler cette fonction pour récupérer les données contenues dans le buffer TWI_Buff. Au préalable, il faut appeler TWI_Start_Transc_With_Data pour effectuer une requête en lecture vers l'esclave concerné puis exécuter la présente fonction pour récupérer les données reçues par le buffer.</p> <p>Le nombre de données définit par « messageSize » est placé dans le tableau « message » situé en SRAM.</p>

B4) Fichiers nécessaire au driver « Maître TWI »

La bibliothèque des fonctions **TWI_Master_CVAVR.lib** doit être placée dans le répertoire CVAVR\LIB.

Le fichier d'en-tête **TWI_Master_CVAVR.h** doit être placé dans le répertoire CVAVR\INC.

C) Commande des périphériques I²C avec les fonctions du driver TWI

Port d'entrée sortie PCF8574

Un port d'E/S PCF8574 peut être lu ou écrit.

Trame d'écriture pour le PCF8574

➤ Organisation

Une trame d'écriture sur le port d'E/S doit contenir deux octets:

- Octet0 : l'adresse du PCF8574 et le bit de lecture/écriture = écriture : **SLA+W**
- Octet1 : la valeur à placer sur le port d'E/S : **Data**

SLA+W	Data
-------	------

➤ Exemple de déclaration d'une trame d'écriture (en C)

```
// Trame d'écriture sur les LED de la carte S_ISI => {SLA+W = 0x40, Data = Led (on=0 or off=1)}.
unsigned char Frame_I2C_LED_Switching[]={0x40,0x00}; // Toutes les LED éclairées
```

➤ Exemple d'utilisation d'une trame d'écriture (en C)

```
// Utilisation de la fonction TWI_Start_Transc_With_Data du driver "Maitre TWI"
TWI_Start_Transc_With_Data(Frame_I2C_LED_Switching, sizeof(Frame_I2C_LED_Switching));
```

sizeof() retourne la taille d'une variable (ici la longueur du tableau)

Trame de lecture pour le PCF8574

➤ Organisation

Une trame de lecture du port d'E/S doit contenir deux octets:

- Octet0 : l'adresse du PCF8574 et le bit de lecture/écriture = lecture : **SLA+R**
- Octet1 : libre (la « place ») pour la valeur à lire : initialisé à NUL=0x00

SLA+R	NUL
-------	-----

➤ Exemple de déclaration d'une trame de lecture (en C)

```
// Trame de lecture des BP {SLA+R=0x43, data=NUL}
unsigned char Frame_I2C_BP_Read[]={0x43,0x00};
```

➤ Exemple d'utilisation d'une trame de lecture (en C)

```
// Utilisation de la fonction TWI_Start_Transc_With_Data du driver "Maitre TWI" pour lire
// la valeur et la placer dans le buffer TWI_buff
TWI_Start_Transc_With_Data(Frame_I2C_BP_Read, sizeof(Frame_I2C_BP_Read));

// Utilisation de la fonction TWI_Get_Data_From_Transceiver pour lire le contenu du buffer
// et le placer dans le tableau Frame_I2C_BP_Read
TWI_Get_Data_From_Transceiver(Frame_I2C_BP_Read, sizeof(Frame_I2C_BP_Read));
```

LCD I2C BATRON (PCF2119x)

Comme tout périphérique I²C, le LCD BATRON est commandé par l'intermédiaire de trames I²C. L'afficheur peut recevoir des trames de texte ou des trames de commande (ex : effacer l'écran). Le contenu de la trame I²C doit être adapté en conséquence.

Trames d'écriture pour l'afficheur I2C BATRON

• Trame de commande

➤ Organisation

Une trame de commande doit contenir trois octets :

- Octet0 : adresse de l'afficheur LCD BATRON et le bit de lecture/écriture = écriture : **SLA+W**
- Octet1 : code d'accès au registre d'instruction = **RegIR**
- Octet2 : instruction à exécuter (voir doc BATRON)

SLA+W	RegIR	inst
-------	-------	------

➤ Exemple de déclaration d'une trame de commande (en C)

```
// Effacer l'écran
unsigned char Frame_I2C_LCDBATRON_CLEAR[]={0x76,0x00,0x01} ;
```

➤ Exemple d'utilisation d'une trame de commande (en C)

```
TWI_Start_Transc_With_Data(Frame_I2C_LCD_BATRON_Clear,sizeof(Frame_I2C_LCD_BATRON_Clear));
```

• Trame de texte

➤ Organisation

Une trame de texte contient un nombre variable d'octet :

- Octet0 : adresse de l'afficheur LCD BATRON et le bit de lecture/écriture = écriture : **SLA+W**
- Octet1 : écriture en DDRAM = **WDDRAM**
- Octet2 à n-1 : caractères à afficher (16 max) (**ATTENTION : Code ASCII + 0x80**)
- Octetn : caractère de fin de chaîne (**NUL**) pour pouvoir utiliser les fonctions de gestion de chaîne (strcat, strlen etc...)

Exemple:

SLA+W	WDDRAM	Caract1	Caract2	...	NUL
-------	--------	---------	---------	-----	-----

➤ Exemple de déclaration d'une trame de texte (en C)

Le nombre de caractères à envoyer pouvant être compris entre 1 et 16 et chaque code caractère (ASCII) devant être transcodé avec l'expression ASCII + 0x80, la trame à afficher est construite par la fonction *Built_Strng_Fram_I2C_LCDBATRON()* de la bibliothèque SSI décrite en annexe 2.

```
// Partie declarative du programme
unsigned char String_for_LCD[17];
unsigned char Frame_I2C_Lcd_Line[19]={0x76,0x40};
```

➤ Exemple d'utilisation d'une trame de texte (en C)

```
// Partie exécutive du programme
// Copie du texte à envoyer dans le tableau String_to_LCD (16 caractères max)
strcpyf(String_to_LCD, "PPE Robsol"); // Voir strcpyf en annexe 1 !
// Le texte précédent est transcodé et ajouté à l'en tête de la trame
Built_Strng_Fram_I2C_LCDBATRON (Frame_I2C_Lcd_Line, String_for_LCD,
strlen(String_for_LCD)); // Voir strlen en annexe 1
TWI_Start_Transc_With_Data(Frame_I2C_Lcd_Line,strlen(Frame_I2C_Lcd_Line));
```


Carte de commande « MD23 » des moteurs à courant continu

Les registres de la carte « MD23 » sont accessibles en lecture et en écriture (voir doc MD23). Les déclarations de ces registres, contenues dans le fichier d'en-tête MD23.h, seront utilisées pour construire les trames.

Trames d'écriture pour la carte « MD23 »

➤ Organisation

Une trame d'écriture contient trois octets:

- **Octet0** : adresse de la carte MD23 + le bit de lecture/écriture = écriture : **SLA+W**
- **Octet1** : numéro du registre auquel on souhaite accéder en écriture : **NumReg**
- **Octet2** : Valeur à placer dans le registre sélectionné : **Data**

SLA+W	NumReg	Data
-------	--------	------

➤ Exemple de déclaration d'une d'écriture (en C)

```
// Mise à zéro des registres des encodeurs
unsigned char Reset_MD23[]={MD23_ADD_BASE_W,Reg_Commande, Reset_Encodeur};
```

➤ Exemple d'utilisation d'une trame d'écriture (en C)

```
TWI_Start_Transc_With_Data(Reset_MD23,sizeof(Reset_MD23));
```

Trames de lecture pour la carte « MD23 »

La lecture d'un ou plusieurs registres de la carte « MD23 » nécessite l'envoi de **deux trames** et la lecture des informations reçues dans le buffer TWI:

- **Trame 1** : Envoi de l'octet d'adresse MD23 + bit RW=écriture (**SLA+W**) suivi du numéro du registre ou commence la lecture (**NumReg**).
- **Trame 2** : Envoi de l'octet d'adresse MD23 + bit RW=lecture.
- Lecture des données placées dans le buffer TWI.

➤ Organisation

- **Trame 1** : première trame à envoyer

SLA+W	NumReg
-------	--------

- **Trame 2**: deuxième trame à envoyer

SLA+R	X	X	X	...	X
-------	---	---	---	-----	---

➤ Exemple de déclaration nécessaire à l'envoi d'une trame de lecture (en C)

```
// Trame 1 : Read_Reg_MD23_W={SLA_W=0xB0, N° registre où commence la lecture}
unsigned char Read_Reg_MD23_W[]={MD23_ADD_BASE_W,0};
// Trame 2 : Table_Registers_MD23 {SLA_R, « place » pour la valeur des 17 registres}
unsigned char Table_Registers_MD23[]={MD23_ADD_BASE_R,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
```

➤ Exemple d'utilisation d'une trame de lecture (en C)

```
// Lecture de l'ensemble des registres de la carte MD23
// Envoi de la trame 1
TWI_Start_Transc_With_Data(Read_Reg_MD23_W, sizeof(Read_Reg_MD23_W));
// Envoi de la trame 2
TWI_Start_Transc_With_Data(Table_Registers_MD23,sizeof(Table_Registers_MD23));
// Transfère du buffer TWI dans le tableau « Table_Registers_MD23 »
TWI_Get_Data_From_Transceiver(Table_Registers_MD23,sizeof(Table_Registers_MD23));
```

Annexe 1 : Fonctions du langage « C »**Fonctions divers accessibles avec <stdlib.h>**

void itoa(int n, char *str)

converts the integer n to characters in string str.

Fonctions de traitement de chaîne accessibles avec <string.h>

Une chaîne de caractères ou tableau de caractères (string en anglais) est composée de caractères alphanumériques. Un caractère spécial, **NUL (0x00)**, indique la fin de la chaîne. Ce caractère est appelé « zéro de fin de chaîne ». La taille d'un tableau de caractères doit être suffisante pour contenir le zéro de fin de chaîne. Ce zéro de fin de chaîne est utilisé par les fonctions de traitement de chaîne.

char *strcat(char *str1, char *str2)

Concatenate the string str2 to the end of the string str1.

char *strcpy(char *dest, char *src)

Copies the string src to the string dest.

char *strcpyf(char *dest, char flash *src)Copies the string src, located in FLASH, to the string dest, located in SRAM.
Returns a pointer to the string dest.

unsigned char strlen(char *str)

for the TINY memory model.

returns the length of the string str (in the range 0..255).

unsigned int strlen(char *str)

for the SMALL memory model.

returns the length of the string str (in the range 0..65535).

Fonctions d'entrée/sortie accessibles avec <stdio.h>

```
int sprintf(char *str, char flash *fmtstr [ , arg1, arg2, ...])
```

Formatted text is placed in the null terminated character string **str**, using **putchar**, according to the format specifiers in the **fmtstr** string.

The format specifier string **fmtstr** is constant and must be located in FLASH memory.

The implementation of **printf** is a reduced version of the standard C function.

This was necessary due to the specific needs of an embedded system and because the full implementation would require a large amount of FLASH memory space.

The function returns the number of outputed characters.

The format specifier string has the following structure:

```
%[flags][width][.precision][l]type_char
```

The optional **flags** characters are:

'-' left-justifies the result, padding on the right with spaces. If it's not present, the result will be right-justified, padded on the left with zeros or spaces;

'+' signed conversion results will always begin with a '+' or '-' sign;

' ' if the value isn't negative, the conversion result will begin with a space. If the value is negative then it will begin with a '-' sign.

The optional **width** specifier sets the minimal width of an output value. If the result of the conversion is wider than the field width, the field will be expanded to accommodate the result, so not to cause field truncation.

The following width specifiers are supported:

n - at least n characters are outputted. If the result has less than n characters, then it's field will be padded with spaces. If the '-' flag is used, the result field will be padded on the right, otherwise it will be padded on the left;

0n - at least n characters are outputted. If the result has less than n characters, it is padded on the left with zeros.

The optional **precision** specifier sets the maximal number of characters or minimal number of integer digits that may be outputted.

For the 'e', 'E' and 'f' conversion type characters the precision specifier sets the number of digits that will be outputted to the right of the decimal point.

The precision specifier always begins with a '.' character in order to separate it from the width specifier.

The following precision specifiers are supported:

none - the precision is set to 1 for the 'i', 'd', 'u', 'x', 'X' conversion type characters. For the 's' and 'p' conversion type characters, the char string will be outputted up to the first null character;

.0 - the precision is set to 1 for the 'i', 'd', 'u', 'x', 'X' type characters;

.n - n characters or n decimal places are outputted.

For the 'i', 'd', 'u', 'x', 'X' conversion type characters, if the value has less than n digits, then it will be padded on the left with zeros. If it has more than n digits then it will not be truncated.

For the 's' and 'p' conversion type characters, no more than n characters from the char string will be outputted.

For the 'e', 'E' and 'f' conversion type characters, n digits will be outputted to the right of the decimal point.

The precision specifier has no effect on the 'c' conversion type character.

The optional **'l'** input size modifier specifies that the function argument must be treated as a long int for the 'i', 'd', 'u', 'x', 'X' conversion type characters.

The **type_char** conversion type character is used to specify the way the function argument will be treated.

The following conversion type characters are supported:

'i' - the function argument is a signed decimal integer;

'd' - the function argument is a signed decimal integer;

'u' - the function argument is an unsigned decimal integer;

'e' - the function argument is a float, that will be outputted using the [-]d.ddd ddd e[-]dd format

'E' - the function argument is a float, that will be outputted using the [-]d.ddd ddd E[-]dd format

'f' - the function argument is a float, that will be outputted using the [-]ddd.ddd ddd format

'x' - the function argument is an unsigned hexadecimal integer, that will be outputted with

lowercase characters;

'X' - the function argument is an unsigned hexadecimal integer, that will be outputted with with

uppercase characters;

'c' - the function argument is a single character;

's' - the function argument is a pointer to a null terminated char string located in RAM;

'p' - the function argument is a pointer to a null terminated char string located in FLASH;

'%' - the '%' character will be outputted.

Annexe 2 : Fonctions de la bibliothèque SSI (SSI.h)

Définition du code des BP de l'IHM I2C destinée à la carte ATMEL S_ISI

```
#define ECHAP_I2C          128
#define ENTR_I2C           64
#define OK_I2C             32
#define SET_I2C            16
#define MOINS_I2C          8
#define FlecheB_I2C        4
#define FlecheH_I2C        2
#define PLUS_I2C           1
```

Liste des fonctions développées pour le PPE Robot solaire

- 1) LCD_I2C_BATRON_gotoxy
- 2) Built_Strng_Fram_I2C_LCDBATRON
- 3) Displ_2Lines_On_I2C_LCDBATRON
- 4) void Display_Arrow_0n_I2C_LCDBATRON

void LCD_I2C_BATRON_gotoxy(unsigned char x, unsigned char y);

Rôle : Positionner le curseur sur l'afficheur LCD BATRON

Commentaires : fonction écrite pour le PPE Robot solaire

Entrées : x=position dans la ligne [0,15], y=n° de la ligne [0,1]

void Built_Strng_Fram_I2C_LCDBATRON(unsigned char FrameI2C, unsigned char Tstring, int Tstringsize);

Rôle : Construire une trame de texte de seize caractères maxi destinée à être affichée sur le LCD I2C BATRON.

Commentaires : Les caractères ASCII, passés dans le tableau "Tstring", sont transcodés au format BATRON (ASCII+0x80) avant d'être placés à la suite des codes SLA+W=0x76 et cde=write to DDRAM = 0x40 dans le tableau "FrameI2C".

Entrées :

Tstring: Tableau de 16 caractères ASCII (terminé par Nul)

Tstringsize: longueur du tableau de caractères "Tstring"

Entrée/sortie

FrameI2C: Tableau de 19 caractères, représentatif de la trame de texte qui sera envoyée à l'afficheur avec la fonction *TWI_Start_Transc_With_Data()*. Ce tableau doit être initialisé avec (SLA+W=0x76) et (cde=write to DDRAM=0x40) lors de sa déclaration.

Exemple de déclaration : unsigned char Frame_I2C_Lcd_Line[19]={0x76,0x40};

A l'entrée dans la fonction, FrameI2C={0x76,0x40,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x}

A la sortie de la fonction, FrameI2C={SLA+W=0x76, cde=write to DDRAM=0x40, Contenu de Tstring= 16 caractères max, Nul }

```
void Displ_2Lines_On_I2C_LCDBATRON(char *Buffer_for_LCD_Line0, char  
*Buffer_for_LCD_Line1);
```

Rôle : Afficher 2 lignes de 16 caractères sur le LCD BATRON

Commentaires : Cette fonction construit une trame de texte au format BATRON (ASCII+0x80) à partir des tableaux de caractères ASCII "Buffer_for_LCD_Line0" et "Buffer_for_LCD_Line1" et la transmet à l'afficheur.

Entrées :

Buffer_for_LCD_Line0, Buffer_for_LCD_Line1 : Tableaux de caractères contenant les deux lignes de texte (au format ASCII) à envoyer à l'afficheur {16 caractères ASCII max, Nul}

Exemple de déclaration : char Buffer_for_LCD_Line0[17], Buffer_for_LCD_Line1[17]

```
void Display_Arrow_On_I2C_LCDBATRON(void);
```

Rôle : Afficher les flèches "flèche haute" et "flèche basse" à droite de l'écran LCD.

```
unsigned char Lire_BP_I2C(void);
```

Rôle : Lit l'état des boutons-poussoirs de l'IHM reliée à la carte ATMEL S_ISI(PCF8574) et renvoie un code sur 8 bits.

Commentaires : Dans un premier temps, la fonction renvoie le code du bouton-poussoir actionné (PLUS_I2C = 0x01, FlecheH=0x02, FlecheB=0x04, MOINSI2C = 0x08, SET_I2C = 0x10, OK_I2C = 0x20, ENTR_I2C=0x40, ECHAP_I2C = 0x80) puis elle renvoie 0xFF si la touche est maintenue ou 0x00 si la touche est relâchée.

Entrées : aucunes

Sortie : code de la touche ou 0x00 ou 0xFF

Annexe 3 : Fichier d'en-tête pour la carte MD23 « MD23.h »

Pour lire un ou plusieurs des registres de la carte MD23

Avec la fonction "TWI_Start_Transc_With_Data()":

(1) Envoyer la trame 1 { MD23_ADD_BASE_W, Octet_Numéro_premier_registre_à_Lire } puis

(2) Envoyer la trame 2 { MD23_ADD_BASE_R, emplacements libres pour les données }

Avec la fonction "TWI_Get_Data_From_Transceiver()":

Transférer le contenu du buffer TWI dans le tableau message

Pour écrire dans un registre

Avec la fonction "TWI_Start_Transc_With_Data()"

Envoyer la trame { MD23_ADD_BASE_W, Octet_Numéro_registre_à_Ecrire, Valeur_à_écrire }

Adresse de la carte (par défaut)

```
#define MD23_ADD_BASE_W      0xB0    // SLA+W
#define MD23_ADD_BASE_R      0xB1    // SLA+R
```

Numéro des registres

```
#define Reg_Speed1           0        // R/W Contrôle la vitesse du moteur 1 (mode 0,1)
                                     // ou la vitesse des deux moteurs (mode 2,3)
#define Reg_Speed2_Turn      1        // R/W Contrôle la vitesse du moteur 2 (mode 0,1)
                                     // ou la direction (mode 2,3)
#define Reg_Enc1a            2        // R 1er octet de l'encodeur 1 (MSB)
                                     // => capture du compteur lors d'une lecture
#define Reg_Enc1b            3        // R 2eme octet de l'encodeur 1
#define Reg_Enc1c            4        // R 3eme octet de l'encodeur 1
#define Reg_Enc1d            5        // R 4eme octet de l'encodeur 1 (LSB)
#define Reg_Enc2a            6        // R 1er octet de l'encodeur 2 (MSB)
                                     // => capture du compteur lors d'une lecture
#define Reg_Enc2b            7        // R 2eme octet de l'encodeur 2
#define Reg_Enc2c            8        // R 3eme octet de l'encodeur 2
#define Reg_Enc2d            9        // R 4eme octet de l'encodeur 2 (LSB)
#define Reg_VBatterie        10       // R Tension de la batterie

#define Reg_Imot1            11       // R Intensité dans le moteur 1
#define Reg_Imot2            12       // R Intensité dans le moteur 2
#define Reg_SoftRev          13       // R Version du logiciel
#define Reg_Acceleration     14       // R/W Réglage de l'accélération
#define Reg_Mode             15       // R/W Mode de fonctionnement (0,1,2,3)
#define Reg_Commande         16       // R/W Registre de commande
```

Valeurs à placer dans le registre de Mode

```
#define Mode0                0        // Dans Speed1 ou Speed2, 0 => Rotation max inverse
                                     // 128 => Arrêt
                                     // 255 => Rotation max directe
#define Mode1                1        // Dans Speed1 ou Speed2, -128 => Rotation max inverse
                                     // 0 => Arrêt
                                     // 127 => Rotation max directe
#define Mode2                2        // Speed1 contrôle la vitesse des deux moteurs, Speed2
                                     // contrôle la direction
                                     // 0 => Rotation max inverse
                                     // Pour les deux reg 128 => Arrêt
                                     // 255 => Rotation max directe
#define Mode3                3        // Speed1 contrôle la vitesse des deux moteurs, Speed2
                                     // contrôle la direction
                                     // -128 => Rotation max inverse
                                     // Pour les deux reg 0 => Arrêt
                                     // 127 => Rotation max directe
```

Valeurs à placer dans le registre de commande

```
#define Reset_Encodeur          32

#define Desactive_Regulation     48  // Régulation de la vitesse des moteurs

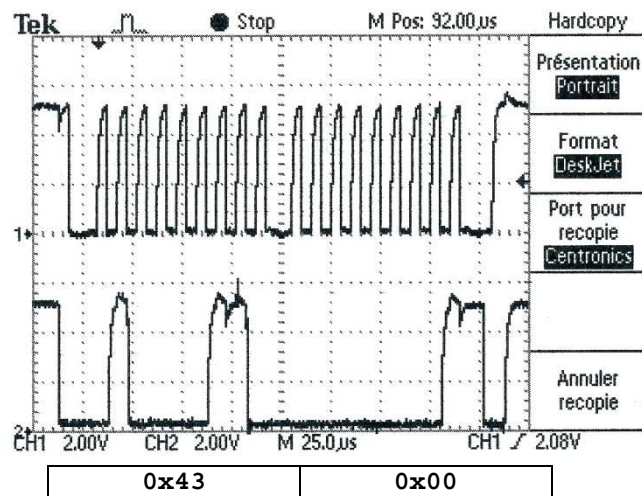
#define Active_Regulation        49

#define Desactive_Timeout_Moteurs 50

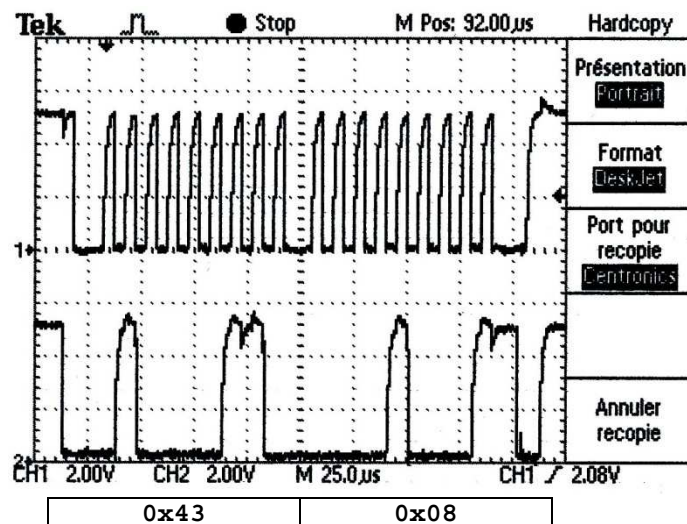
#define Active_Timeout_Moteurs   51 // Arrêt des moteurs si pas de transfert pendant 2s
```

Annexe 4 : Mesures faites sur le bus I2C

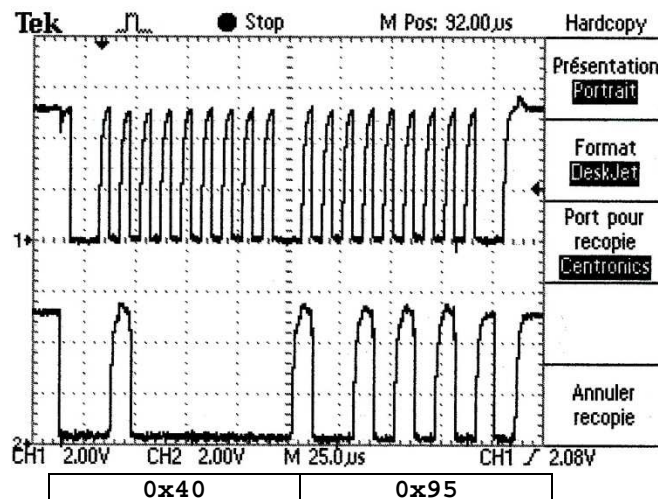
Lecture du clavier : (Aucune touche actionnée)



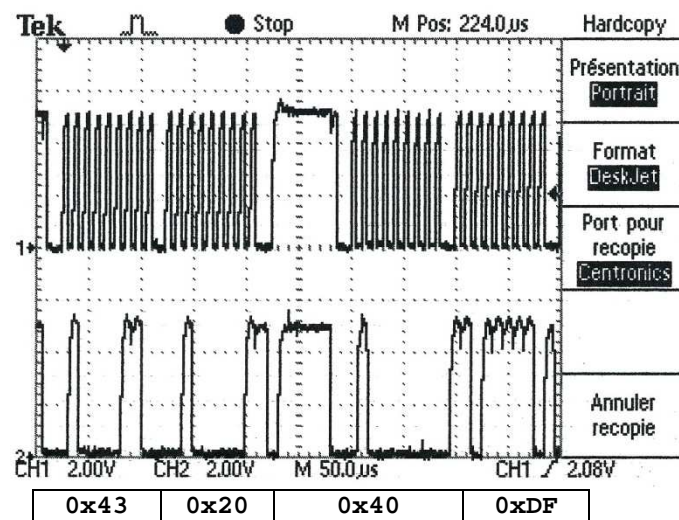
Lecture du clavier : (Touche « - »)



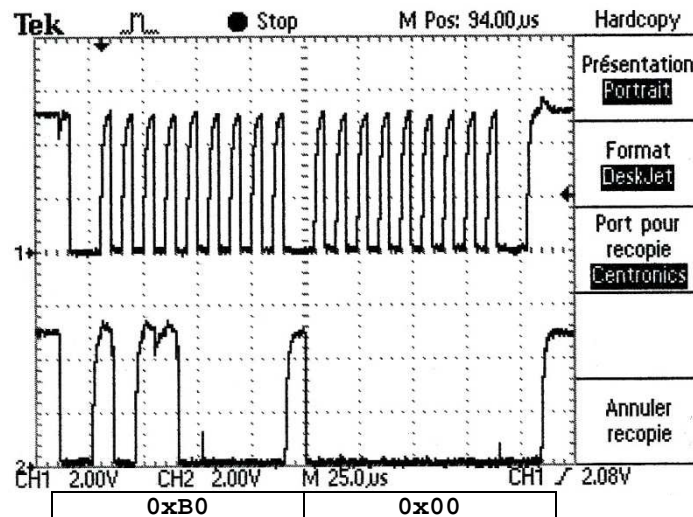
Ecriture sur les LED : (D7, D4, D2, D0 éteintes - D6, D5, D3, D1 éclairées)



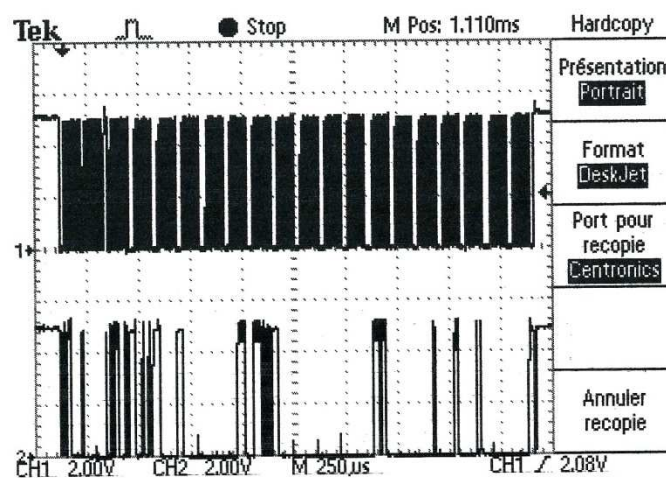
Lecture du clavier et écriture sur les LED : BP « OK » actionné, LED D5 éclairée



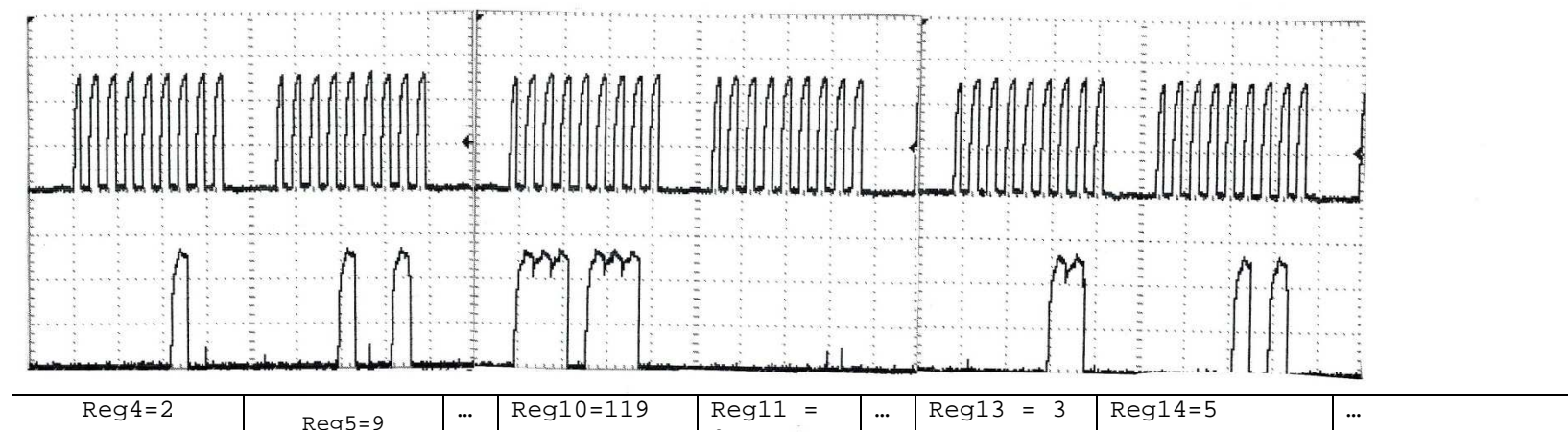
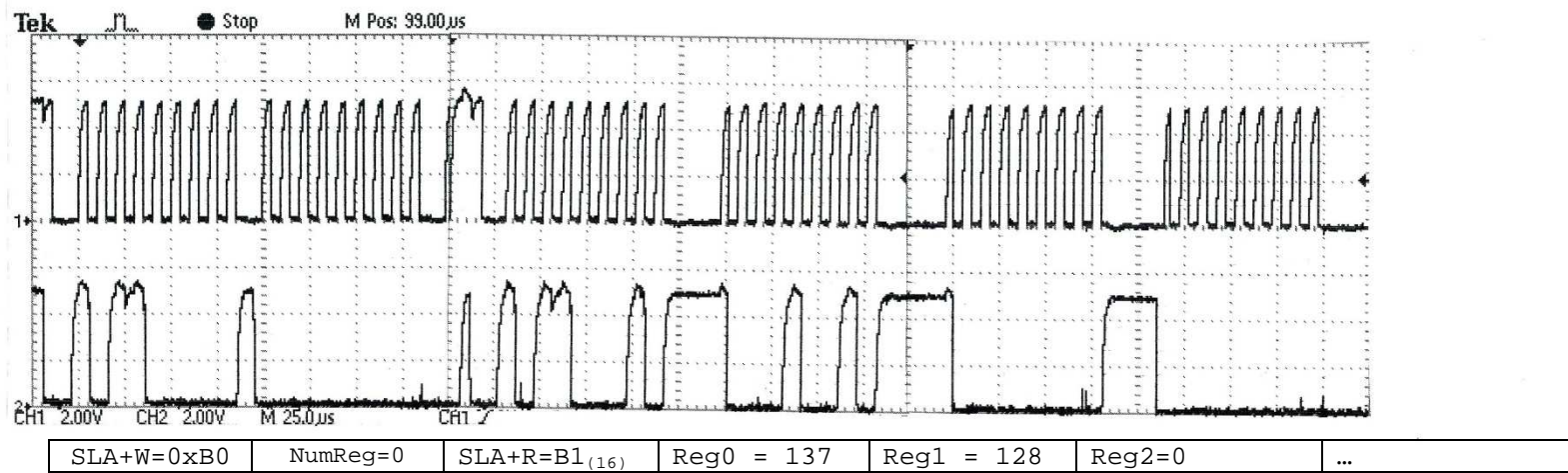
Ecriture à partir du registre 0 de la carte MD23

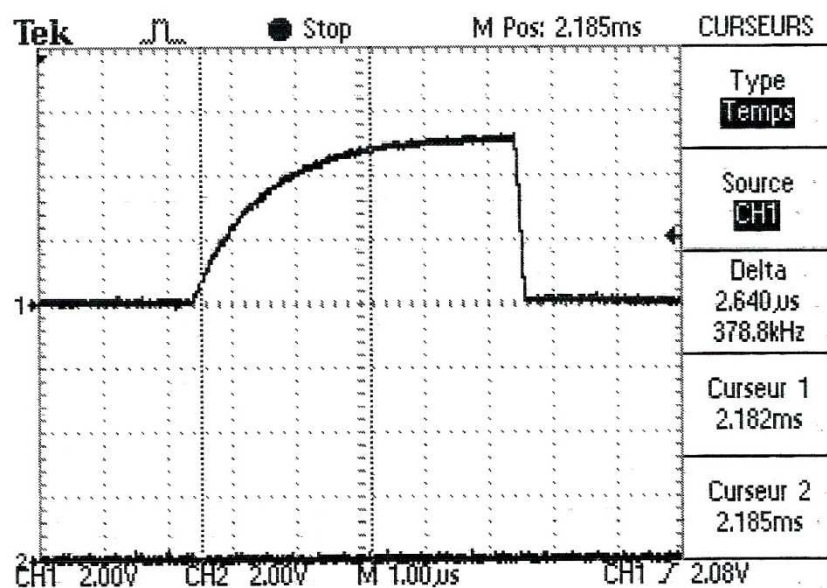


Lecture des dix sept registres de la carte MD23



Lecture de tous les registres de la carte MD23 (zoom sur certains registres)



Influence de la résistance de rappel R_p ($F_{clk} = 100kHz$) $R_p = 10k$ ($t_r = 2,6\mu s$) $R_p = 3,3k$ ($t_r = 0,7\mu s$)