MSP430F1232

Bus I2C en mode esclave
Couche liaison sans coupleur spécifique

Principe retenu : utilisation intensive des interruptions
Le programme proposé traite les trames I2C d'écriture et de lecture à longueur variable, mais sans adresse secondaire.

1. Principe

Les signaux SDA sont connectés au µC comme suit :

- SCL sur P1.0 (bit 0 du port 1)
- SDA sur P2.0

D'autres connexions sont possibles. On recommande toutefois d'utiliser 2 ports différents pour disposer de 2 vecteurs d'interruption spécifiques.
Les changements d'état de SDA et SCL provoque des demandes d'interruption. Le traitement dépend de la phase en cours du protocole I2C.

2. Sources

2.1 Constantes et variables

```
/*------------------------------------------------------------------------------
  Nom           :  Gestion_I2C_MSP1132.c
  Description   :  Gestion I2C "esclave" sur MSP430F1132
  Author        :  29/01/2007 - CREMMEL Marcel
  History       :  29/01/2007 - Première version
------------------------------------------------------------------------------*/
#include <msp430x12x2.h>

/*------------------------------------------------------------------------------
                                    Définition de type
------------------------------------------------------------------------------*/
typedef enum
{
  Idle_I2C    = 0,
  Start       = 1,
  Receive_Adr = 2,
  Send        = 3,
  Receive     = 4,
  Deb_Ack_RX  = 5,
  Fin_Ack_RX  = 6,
  Deb_Ack_TX  = 7,
  Fin_Ack_TX  = 8,
  Deb_NAck    = 9,
  Fin_NAck    = 10,
  Deb_RAck_TX = 11
}tI2C_State; // Type sequence I2C en cours

typedef enum
{
  Idle        = 0, // Etat de veille
  I2C_InUse   = 1, // Controleur connecté
}tMode_Fct; // Type de mode de fonctionnement
```

```
/*-------------------------------------------------------------------------------
                          Constantes non mémorisées
-------------------------------------------------------------------------------*/

/*------------------
       Câblage
--------------------*/
// Signaux de test
#define TEST1_Out P3DIR|= BIT0 // TEST1 sur P2.0
#define Set_TEST1 P3OUT|= BIT0 // TEST1 sur P2.0
#define Clr_TEST1 P3OUT&=~BIT0 // TEST1 sur P2.0
#define TEST2_Out P3DIR|= BIT1 // TEST2 sur P3.1
#define Set_TEST2 P3OUT|= BIT1 // TEST2 sur P3.1
#define Clr_TEST2 P3OUT&=~BIT1 // TEST2 sur P3.1
#define Inv_TEST2 P3OUT^= BIT1 // TEST2 sur P3.1
#define TEST3_Out P3DIR|= BIT2 // TEST3 sur P2.2
#define Set_TEST3 P3OUT|= BIT2 // TEST3 sur P2.2
#define Clr_TEST3 P3OUT&=~BIT2 // TEST3 sur P2.2
#define TEST4_Out P3DIR|= BIT3 // TEST4 sur P2.3
#define Set_TEST4 P3OUT|= BIT3 // TEST4 sur P2.3
#define Clr_TEST4 P3OUT&=~BIT3 // TEST4 sur P2.3
#define TEST5_Out P3DIR|= BIT4 // TEST5 sur P2.4
#define Set_TEST5 P3OUT|= BIT4 // TEST5 sur P2.4
#define Clr_TEST5 P3OUT&=~BIT4 // TEST5 sur P2.4

// I2C
#define Init_SDA    P2OUT&=~BIT0; P2DIR&=~BIT0 // SDA=P2.0="1" en "drain ouvert"
#define Set_SDA     P2DIR&=~BIT0 // Mise à "1" SDA par pull-up externe
#define Clr_SDA     P2DIR|= BIT0 // Mise à "0" SDA en mode "output"
#define SDA_Int_En  P2IE |= BIT0 // Validation interruption SDA sur P2.0
#define SDA_Int_Inh P2IE &=~BIT0 // Inhibition interruption SDA sur P2.0
#define SDA_Int_LH  P2IES&=~BIT0 // Interruption SDA sur flanc montant
#define SDA_Int_HL  P2IES|= BIT0 // Interruption SDA sur flanc descendant
#define Clr_SDA_Flg P2IFG&=~BIT0 // Raz indicateur interruption SDA
#define Test_SDA    P2IN & BIT0  // Test état SDA

#define Init_SCL    P1OUT&=~BIT0; P1DIR&=~BIT0 // SCL=P1.0="1" en "drain ouvert"
#define Set_SCL     P1DIR&=~BIT0 // Mise à "1" SCL par pull-up externe
#define Clr_SCL     P1DIR|= BIT0 // Mise à "0" SCL en mode "output"
#define SCL_Int_En  P1IE |= BIT0 // Validation interruption SCL sur P1.0
#define SCL_Int_Inh P1IE &=~BIT0 // Inhibition interruption SCL sur P1.0
#define SCL_Int_LH  P1IES&=~BIT0 // Interruption SCL sur flanc montant
#define SCL_Int_HL  P1IES|= BIT0 // Interruption SCL sur flanc descendant
#define Clr_SCL_Flg P1IFG&=~BIT0 // Raz indicateur interruption SCL
#define Test_SCL    P1IN & BIT0  // Test état SDA

/*------------------
         I2C
--------------------*/
#define Adr_DATASCONT 0x50 // Adresse I2C pour affecter la structure DATASCONT
                           // (bit 0 = "0" : écriture)
#define Adr_PARAMCONT 0x51 // Adresse I2C pour renvoyer la structure PARAMCONT
                           // (bit 1 = "1" : lecture)
```

```
/*-------------------------------------------------------------------------------
                                    Variables globales
-------------------------------------------------------------------------------*/
/*-----------------
   Indicateurs
-------------------*/
struct
{
 unsigned New_DATASCONT :1;  // Comme son nom l'indique
 unsigned unused        :15;
}Flags;

/*-------------------------
   Mode de fonctionnement
---------------------------*/
tMode_Fct Mode_Fct; // Modes de fonctionnement du tableau de bord

/*-------------------------
       Liaison I2C
---------------------------*/
tI2C_State I2C_State; // Code de la phase I2C actuelle
unsigned int Nb_Datas_I2C;  // Nombre d'octets de datas à transmettre
unsigned int Cpt_octets_I2C;// Compteur d'octets transmis sur le bus I2C
unsigned int Cpt_bits_I2C;  // Compteur de bits transmis sur le bus I2C
unsigned char I2C_Byte;     // Octet en cours de réception ou de transmission
char *pDatas_I2C;     // Utilisé comme pointeur vers les données I2C

/*-------------------------
    Paramètres et données
       du controleur
---------------------------*/
struct
{
 char PAYS;    // Code du pays
 char PROFIL;  // Code du profil d'assistance
 char NB_POLE; // Nb de pôles du moteur BLDC
 char SINUS;   // Type de commande (sinus ou trapèze)
 char EQUIP;   // Equipement du controleur
} PARAMCONT;
#define Lng_PARAMCONT 5; // Structure PARAMCONT = 5 octets

struct
{
 unsigned long NB_HALL; // Nb de chgts d'états des capteurs depuis la
                        // 1° mise en service
 unsigned long PERIOH_H;// Durée entre 2 chgts d'état des capteurs Hall
 struct
 {
  unsigned int VBAT; // Tension batterie. Quantum=80mV
  unsigned int QBAT; // Quantité d'électricité consommée. Résolution : 20mAH
 } ETATBAT; //
 char PROFIL; // Profil actuel d'assistance
 char CLEF_OK;// Présence de la clef codée
} DATASCONT;
#define Lng_DATASCONT 14; // Structure DATASCONT = 14 octets
```

2.2 Programme principal (pour les tests)

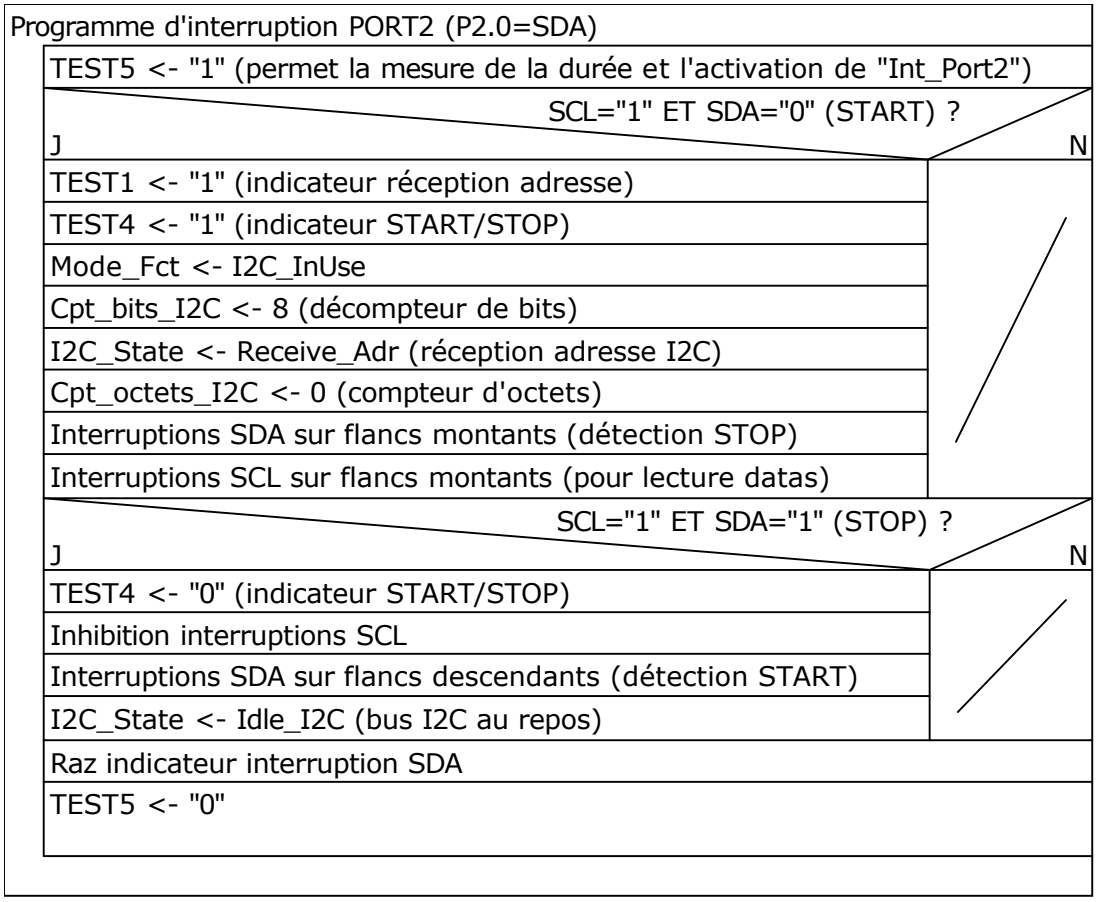
```
void main(void)
{
 WDTCTL=WDTPW+WDTHOLD;       // Arrêt du chien de garde
 SetDCOClock(7,7); // DCO=7, RSEL=7 : FDCO = 4,9 MHz typique
 // Initialisations ports E/S
 TEST1_Out; // TEST1 en sortie
 TEST2_Out; // TEST2 en sortie
 TEST3_Out; // TEST3 en sortie
 TEST4_Out; // TEST4 en sortie
 TEST5_Out; // TEST5 en sortie
 Clr_TEST1;
 Clr_TEST2;
 Clr_TEST3;
 Clr_TEST4;
 Clr_TEST5;
 // I2C
 Init_SDA;    // SDA = "1" par pull-up externe
 Init_SCL;    // SDA = "1" par pull-up externe
 SCL_Int_Inh; // Inhibition interruption SCL
 Clr_SDA_Flg; // Raz indicateur interruption SDA
 Clr_SCL_Flg; // Raz indicateur interruption SCL
 SDA_Int_HL;  // Interruption SDA sur flanc descendant (séquence START)
 SDA_Int_En;  // Validation interruption SDA sur P2.0
 I2C_State=Idle_I2C; // Aucune séquence I2C en cours
 // Affectation des variables
 PARAMCONT.PAYS=0x55;
 PARAMCONT.PROFIL=2;
 PARAMCONT.NB_POLE=3;
 PARAMCONT.SINUS=4;
 PARAMCONT.EQUIP=5;
 // Validation interruptions ds CPU
 _EINT();

/*--------------------
   Boucle principale
 ---------------------
do
  {
   //Inv_TEST2; // Inverser TEST2
   switch (Mode_Fct)
   {
    case Idle :
    {
     __low_power_mode_3(); // Tout à l'arrêt sauf ACLK
     break;
    }
    case I2C_InUse :
    {
     break;
    }
    default : break;
   }
  }
 while (1);
}
```

2.3 Programme d'interruption provoqué par SDA

2.3.1 Source

```
/*-----------------------
   Interruption Port P2
 -----------------------
 Provoquée :
  - au flanc descendant pour détecter la séquence START
  - au flanc montant pour détecter la séquence STOP
*/
 #pragma vector=PORT2_VECTOR
 __interrupt void Int_Port2()
 {
  Set_TEST5;   // Pour mesurer durée programme d'interruption
  if ((Test_SCL) && (!Test_SDA)) // Flanc descendant sur SDA et SCL="1" : START
   {
    Set_TEST1;             // TEST1="1" au START
    Set_TEST4;             // TEST4="1" au START
    Mode_Fct=I2C_InUse;    // Pour ????
    Cpt_bits_I2C=8;        // Compteur bits = 8
    I2C_State=Receive_Adr; // Début de la séquence de réception de l'adresse
    Cpt_octets_I2C=0;      // Raz compteur d'octets
    SDA_Int_LH;    // Interruption SDA sur flanc montant (condition STOP)
    SCL_Int_LH;    // Interruption SCL sur flanc montant
    Clr_SCL_Flg;   // Raz indicateur interruption SCL
    SCL_Int_En;    // Validation interruption SCL sur P1.0
   }
  if ((Test_SCL) && (Test_SDA)) // Flanc montant sur SDA et SCL="1" : STOP
   {
    Clr_TEST4;   // TEST4="0" au STOP
    SCL_Int_Inh; // Inhibition interruption SCL
    SDA_Int_HL;  // Interruption SDA sur flanc descendant (condition START)
    I2C_State=Idle_I2C; // Aucune séquence I2C en cours
   }
  Clr_SDA_Flg; // Raz indicateur interruption
  Clr_TEST5;   // Pour mesurer durée programme d'interruption
  __low_power_mode_off_on_exit();
 }
```

2.3.2 Algorithme



3. Programme d'interruption provoqué par SCL

3.1 Source

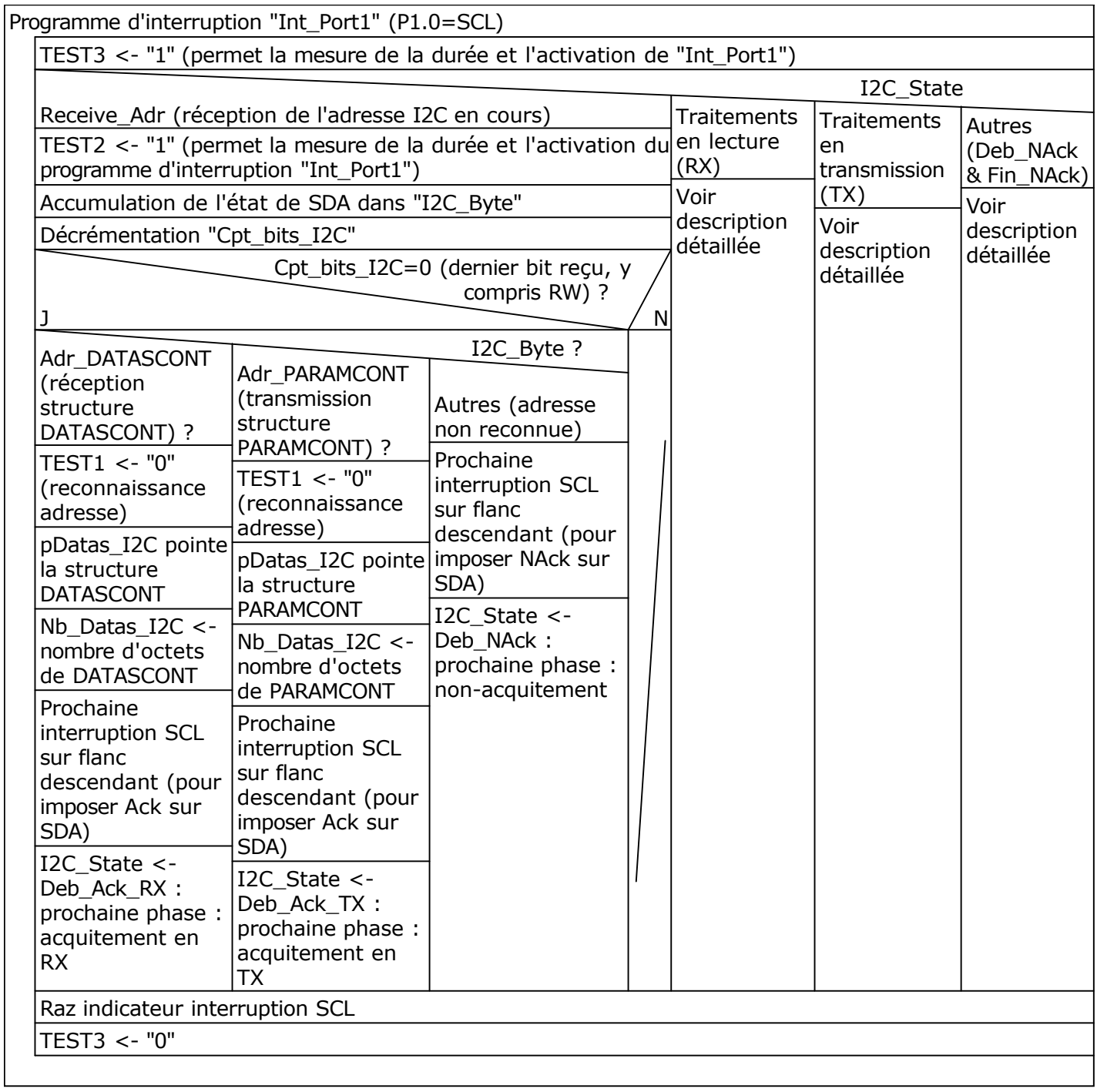
```
/*-----------------------
   Interruption Port P1
 -----------------------
 Activée au flanc montant de SCL dans toutes les phases I2C hors Idle et Stop */
 #pragma vector=PORT1_VECTOR
 __interrupt void Int_Port1()
 {
  Set_TEST3;   // Pour mesurer durée programme d'interruption
  switch(I2C_State)
  {
   case Receive_Adr : // Séquence de réception de l'adresse en cours ?
   {// Oui : accumulation des 8 bits et test adresse I2C
    Set_TEST2;
    if (Test_SDA) I2C_Byte=(I2C_Byte << 1)|1;
             else I2C_Byte<<=1;
    Cpt_bits_I2C--;
    if (!Cpt_bits_I2C)
    {// Oui : identification de l'adresse et affectation des paramètres
     if (I2C_Byte==Adr_DATASCONT) // Réception de la structure DATASCONT ?
     {
      Clr_TEST1; // TEST1="0" à la reconnaissance d'une adresse I2C
      pDatas_I2C=(char *)&DATASCONT; // Variable de destination
      Nb_Datas_I2C=Lng_DATASCONT;    // Nombre d'octets attendus
      SCL_Int_HL; // Prochaine interruption SCL sur flanc descendant
      I2C_State = Deb_Ack_RX;
     }
     else if (I2C_Byte==Adr_PARAMCONT) // Transm. de la structure PARAMCONT?
     {
      Clr_TEST1; // TEST1="0" à la reconnaissance d'une adresse I2C
      pDatas_I2C=(char *)&PARAMCONT; // Variable "source"
      Nb_Datas_I2C=Lng_PARAMCONT;    // Nombre d'octets à transmettre
      SCL_Int_HL; // Prochaine interruption SCL sur flanc descendant
      I2C_State=Deb_Ack_TX;
     }
     else // Adresse non reconnue ?
     {
      SCL_Int_HL;         // Prochaine interruption SCL sur flanc descendant
      I2C_State=Deb_NAck; // Send NAck par pull-up
     }
    }
    break;
   }
   case Deb_Ack_RX : // Début acquitement en réception d'octets en cours ?
   {// Oui : SDA forcé à "0" juste aprés le flanc descendant de SCL
    Clr_TEST2;
    Clr_SDA;              // Acquitement : SDA forcé à "0"
    I2C_State=Fin_Ack_RX; // Fin acquitement en réception en cours
    break;
   }
```

```
  case Fin_Ack_RX : // Fin acquitement en réception d'octets en cours ?
  {// Oui : préparation réception d'un octet ou fin de transfert
   Set_SDA;        // SDA = entrée juste aprés le flanc descendant de SCL
   SCL_Int_LH;     // Prochaine interruption SCL sur flanc montant
   if (Cpt_octets_I2C==Nb_Datas_I2C) // Dernier octet lu ?
    {// Oui : bus I2C au repos
     I2C_State=Idle_I2C;    // Aucune séquence I2C en cours
     Flags.New_DATASCONT=1; // Nouvelles données à afficher
    }
   else
    {// Non : préparation à la réception de l'octet suivant
     Cpt_bits_I2C=8;       // Bit counter = 8, RX data
     I2C_State=Receive;    // Début de la séquence de réception d'un octet
    }
   break;
  }
  case Deb_NAck : // Début séquence de non-acquitement de l'adresse I2C ?
  {// Oui : aucun traitement, on attend juste le prochain flanc descendant
   Clr_TEST2;
   I2C_State=Fin_NAck; // Fin de non-acquitement en cours
   break;
  }
  case Fin_NAck : // Fin de non-acquitement en cours ?
  {// Oui : bus I2C placé au repos
   I2C_State=Idle_I2C; // Aucune séquence I2C en cours
   break;
  }
  case Receive : // Réception d'un octet "data" en cours ?
  {// Oui : - Accumulation bit suivant
   //       - Si dernier bit :
   //           - rangement de l'octet reçu
   //           - préparation de la transmission du bit d'acquitement
   Set_TEST2;
   if (Test_SDA) I2C_Byte=(I2C_Byte << 1)|1;
            else I2C_Byte<<=1;
   Cpt_bits_I2C--;
   if (!Cpt_bits_I2C) // Dernier bit d'un octet ?
   {
    *pDatas_I2C++=I2C_Byte;  // Ranger l'octet reçu
    Cpt_octets_I2C++;        // Incrémentation du compteur d'octets
    SCL_Int_HL; // Prochaine interruption SCL sur flanc descendant
    I2C_State=Deb_Ack_RX;    // Début transmission Ack
   }
   break;
  }
  case Deb_Ack_TX : // Début acquitement en transmission d'octets en cours ?
  {// Oui :
   Clr_TEST2;
   Clr_SDA;              // Acquitement : SDA forcé à "0"
   I2C_State=Fin_Ack_TX; // Fin acquitement en transmission en cours
   break;
  }
  case Fin_Ack_TX : // Fin acquitement en transmission d'octets en cours ?
  {// Oui :
   Set_TEST2;
   I2C_Byte=*pDatas_I2C++;// Octet suivant à transmettre
   Cpt_octets_I2C++;      // Incrémentation du compteur d'octets transmis
   if (I2C_Byte & 0x80) Set_SDA; // Affectation SDA
                   else Clr_SDA;
   I2C_Byte<<=1;   // Préparation bit suivant
   Cpt_bits_I2C=7; // Il reste 7 bits à transmettre
   I2C_State=Send; // Début d'une transmission
   break;
  }
```

```
  case Deb_RAck_TX : // Début lecture Ack en transmission d'octets en cours ?
  {// Oui : lecture de l'acquitement du Master après flanc montant SCL
   Clr_TEST2;
   if ((Test_SDA)||(Cpt_octets_I2C==Nb_Datas_I2C))
    I2C_State=Idle_I2C;// Arrêt séquence I2C si NAck du Master ou dernier octet
   else
   {// Si Ack du Master : on attend le flanc descendant de SCL
    SCL_Int_HL;  // Interruption SCL sur flanc descendant
    I2C_State=Fin_Ack_TX;
   }
   break;
  }
  case Send : // Transmission d'un octet en cours ?
  {// Oui:- si dernier bit: préparation à la lecture de l'acquitement du Master
   //     - affectation SDA avec bit suivant juste après flanc desc. sur SCL
   if (!Cpt_bits_I2C)
   {
    SCL_Int_LH;  // Interruption SCL sur flanc montant pour lecture SDA
    Set_SDA;     // SDA à "1" par pull-up pour lecture SDA
    I2C_State=Deb_RAck_TX;
   }
   else
   {
    if (I2C_Byte & 0x80) Set_SDA; // Affectation SDA
                    else Clr_SDA;
    I2C_Byte<<=1;   // Préparation bit suivant
    Cpt_bits_I2C--; // Décomptage bits
   }
   break;
  }
  default : break;
 }
 Clr_SCL_Flg; // Raz indicateur interruption
 Clr_TEST3;   // Pour mesurer durée programme d'interruption
 __low_power_mode_off_on_exit();
}
```

3.2 Algorithmes

3.2.1 Programme d'interruption complet



3.2.2 Traitements en réception (RX)

Programme d'interruption "Int_Port1" (P1.0=SCL) - Traitements en réception (RX)

Deb_Ack_RX	Fin_Ack_RX		I2C_State
	(fin acquitement sur flanc descendant SCL)		Receive (réception d'un bit sur flanc montant SCL)
Deb_Ack_RX (début acquitement sur flanc descendant SCL)	SDA <- "1" (SDA en entrée pour réception datas)		TEST2 <- "1" (début de réception d'un octet de data)
TEST2 <- "0" (fin de réception adresse)	Prochaine interruption SCL sur flanc montant		Accumulation de l'état de SDA dans "I2C_Byte"
SDA <- "0" (forcer acquitement)	Cpt_octets_I2C=Nb_Datas_I2C (dernier octet) ?		Décrémenter "Cpt_bits_I2C"
	J	N	Cpt_bits_I2C=0 (dernier bit reçu) ?
I2C_State <- Fin_Ack_RX (prochaine interruption : fin acquitement)	I2C_State <- Idle_I2C (bus I2C au repos)	Cpt_bits_I2C <- 8 (prochain octet : réception de 8 bits)	J / N
	New_DATASCONT <- "1" : nouvelles données à afficher	I2C_State <- Receive : prochaine interruption : réception du 1° bit de data	Affectation octet pointé par "pDatas_I2C" par "Byte_I2C"
			Incrémentation pointeur "pDatas_I2C" et compteur "Cpt_octets_I2C"
			Prochaine interruption SCL sur flanc descendant
			I2C_State <- Deb_Ack_RX (prochaine interruption : début acquitement)

3.2.3 Traitements en transmission (TX)

Programme d'interruption "Int_Port1" (P1.0=SCL) - Traitements en émission (TX)

I2C_State					
Deb_Ack_TX (début acquitement sur flanc descendant SCL)	Fin_Ack_TX (fin acquitement sur flanc descendant SCL)	Send (fin de la transmission d'un bit sur flanc descendant de SCL)		Deb_RAck_TX (début lecture SDA au flanc montant de SCL)	
		Cpt_bits_I2C=0 (dernier bit) ?		TEST2 <- "0" : fin de transmission d'un octet	
		J	N	SDA="1" ou Cpt_octets_I2C=Nb_Datas_I2C ?	
				J	N
TEST2 <- "0" (fin de réception adresse)	TEST2 <- "1" : indicateur de transmission d'octet	Interruptions SCL sur flancs montants (pour lecture SDA)	SDA <- état du bit d'indice 7 de "Byte_I2C"	I2C_State <- Idle_I2C : début libération bus I2C	Interruptions SCL sur flancs descendants (pour débuter transmission octet suivant)
SDA <- "0" (forcer acquitement)	I2C_Byte <- Octet pointé par pDatas_I2C (octet suivant à transmettre)	SDA <- "1" (pour libérer la ligne)	Décaler "Byte_I2C" d'un cran à gauche		I2C_State <- Fin_Ack_TX (prochaine interruption : fin lecture acquitement)
I2C_State <- Fin_Ack_TX (prochaine interruption : fin acquitement)	Incrémenter pointeur "pDatas_I2C"	I2C_State <- Deb_RAck_TX (début lecture acquitement du Master)	Décrémenter "Cpt_bits_I2C"		
	Incrémenter Cpt_octet_I2C				
	SDA <- état du bit d'indice 7 de "Byte_I2C"				
	Décaler "Byte_I2C" d'un cran à gauche				
	Cpt_bits_I2C <- 7 : il reste 7 bits à transmettre				
	I2C_State <- Send : début transmission d'un octet				

3.2.4 Autres traitements

Programme d'interruption "Int_Port1" (P1.0=SCL) - Non-acquitement de l'adresse I2C

I2C_State	
Deb_NAck (début du non-acquitement sur flanc descendant SCL)	Fin_NAck (fin non-acquitement sur flanc descendant SCL)
TEST2 <- "0" (fin de réception adresse)	I2C_State <- Idle : bus I2C au repos
I2C_State <- Fin_NAck (prochaine interruption : fin non-acquitement)	

4. Relevés

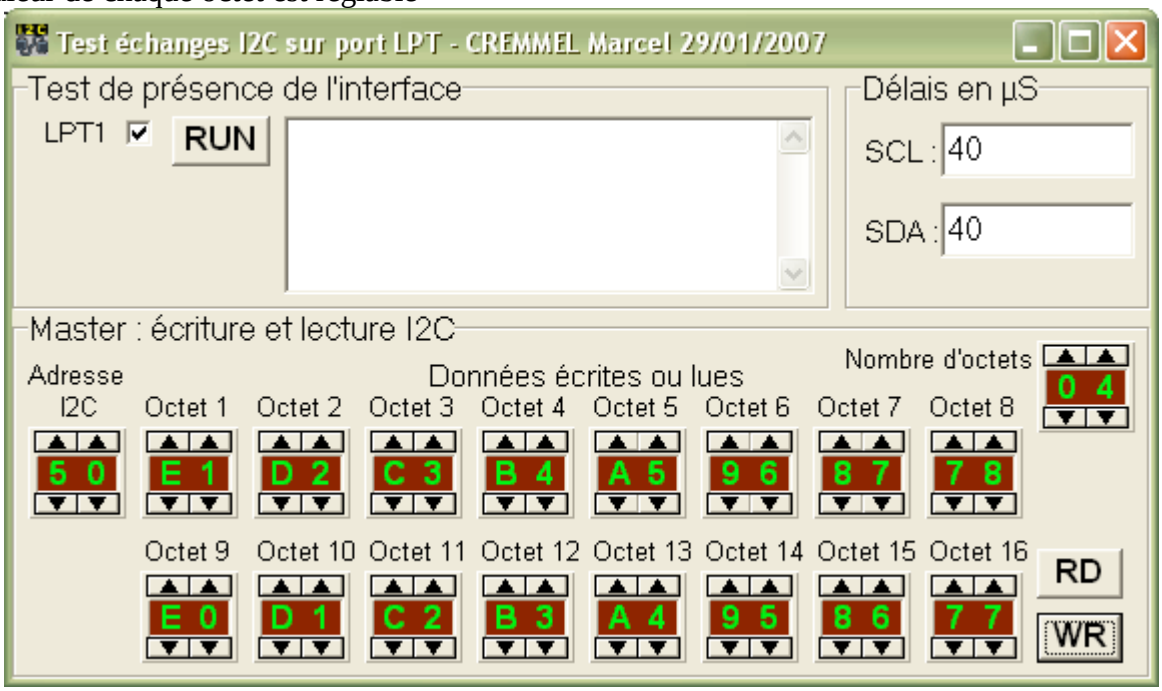
Un certain nombre de signaux de test sont produits par les programmes d'interruption pour identifier les phases et faciliter la validation.
Les relevés sur RIGOL DS1102CD comportent 7 signaux repérés D0 à D6 :

- D0 = SDA
- D1 = SCL
- D2 = TEST1 : mis à "1" à la condition START, mis à "0" à la reconnaissance de l'adresse I2C
- D3 = TEST2 : à "1" pendant la réception ou la transmission d'un octet
- D4 = TEST3 : à "1" pendant l'exécution du programme d'interruption "Int_Port1" (SCL)
- D5 = TEST4 : à "1" entre les conditions START et STOP
- D6 = TEST5 : à "1" pendant l'exécution du programme d'interruption "Int_Port2" (SDA)

4.1 Programme de test sur PC : "Test_I2C.exe"

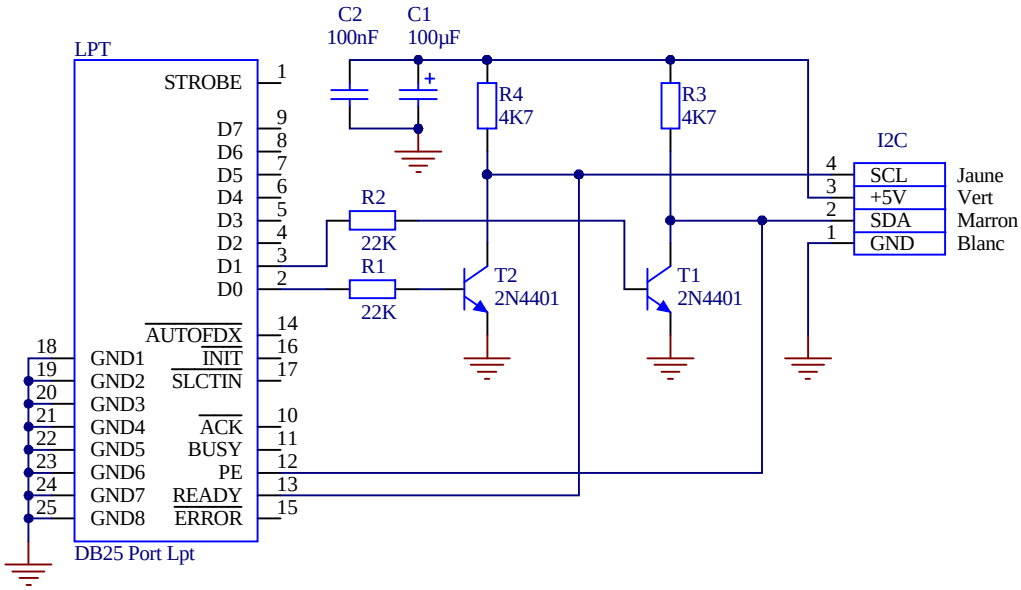
Ce programme transforme le PC en "Master I2C" pour écrire ou lire des octets dans un "esclave" :

- L'adresse I2C est réglable à volonté,
- Le nombre d'octets peut être ajusté entre 1 et 16
- La valeur de chaque octet est réglable



- Des délais réglables peuvent être insérés après chaque changement d'état de SCL ou SDA.

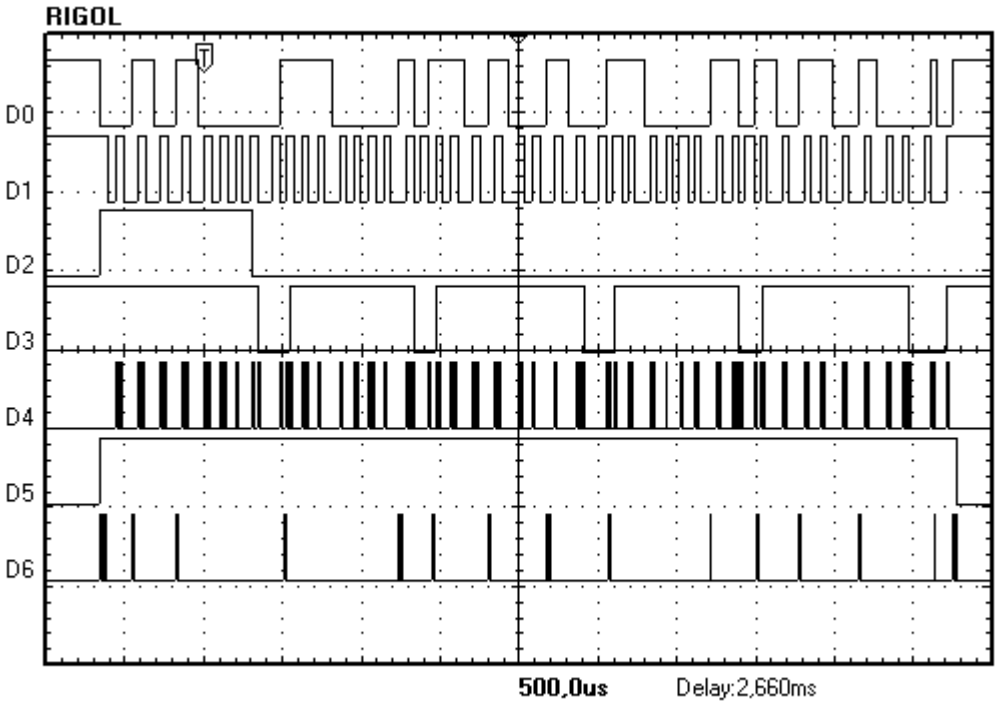
Schéma de l'interface :



4.2 Ecriture

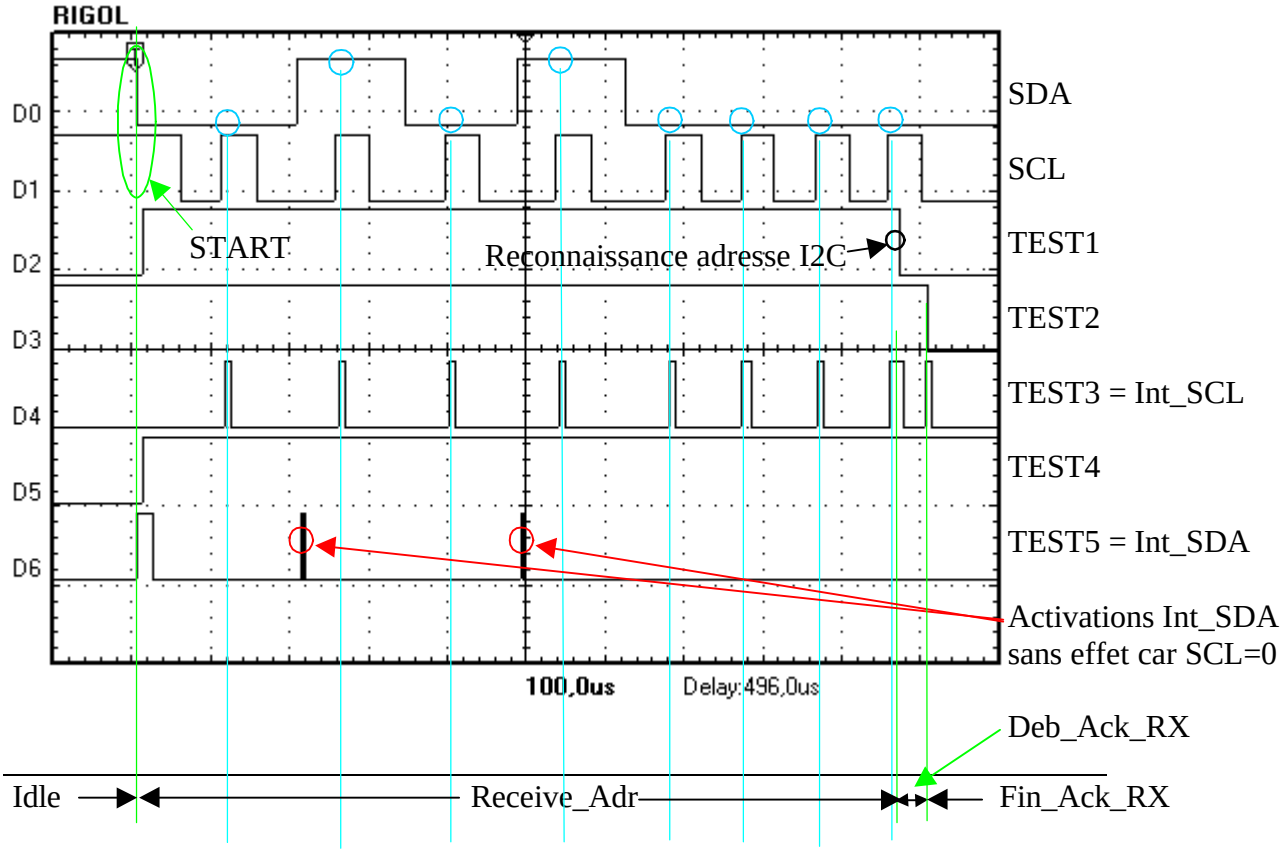
Les relevés sont ceux d'une écriture de 4 octets à l'adresse I2C (50)h.

Trame complète



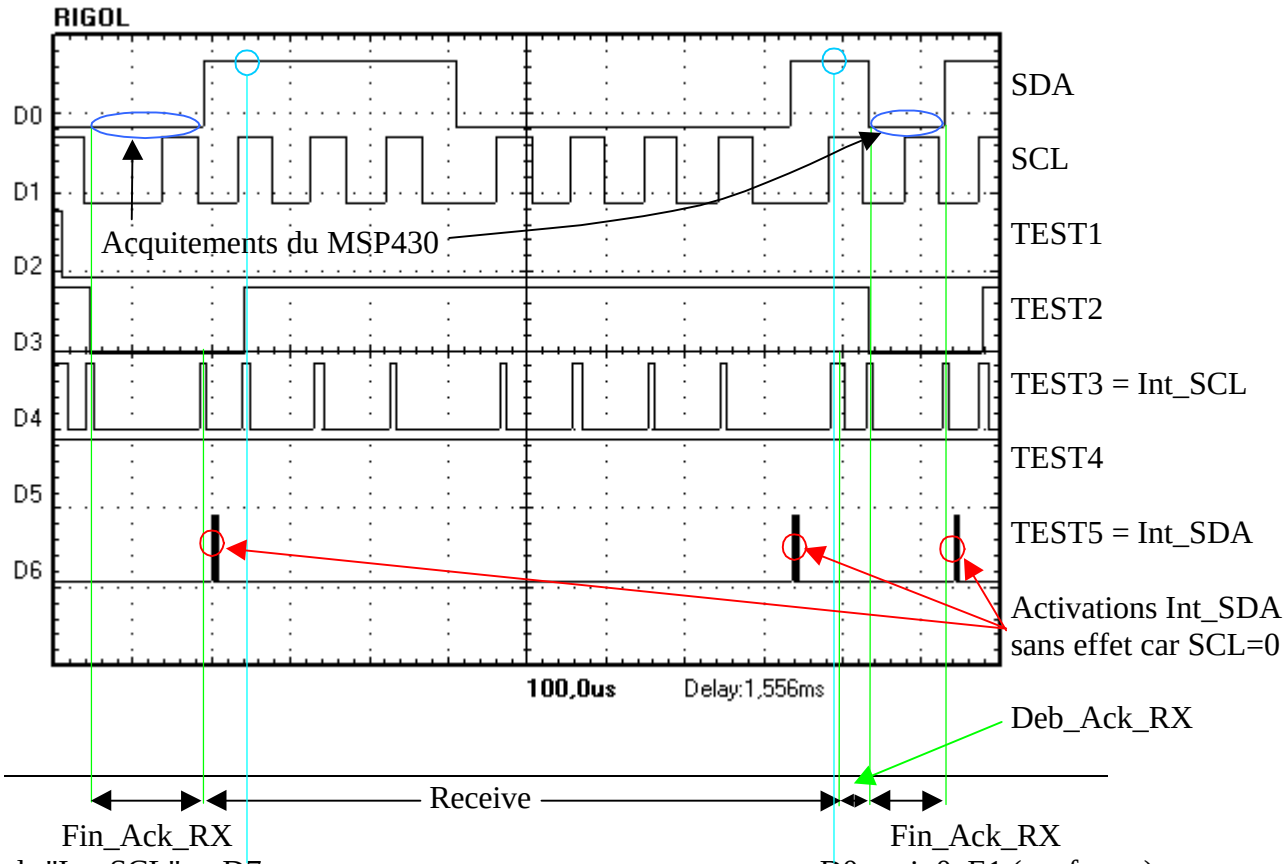
La fonction détection "glitch" de l'oscilloscope est activée pour observer les impulsions très courtes.
Durée du transfert =5,42mS environ (délai de 40µS dans le logiciel PC).

START + Adresse I2C



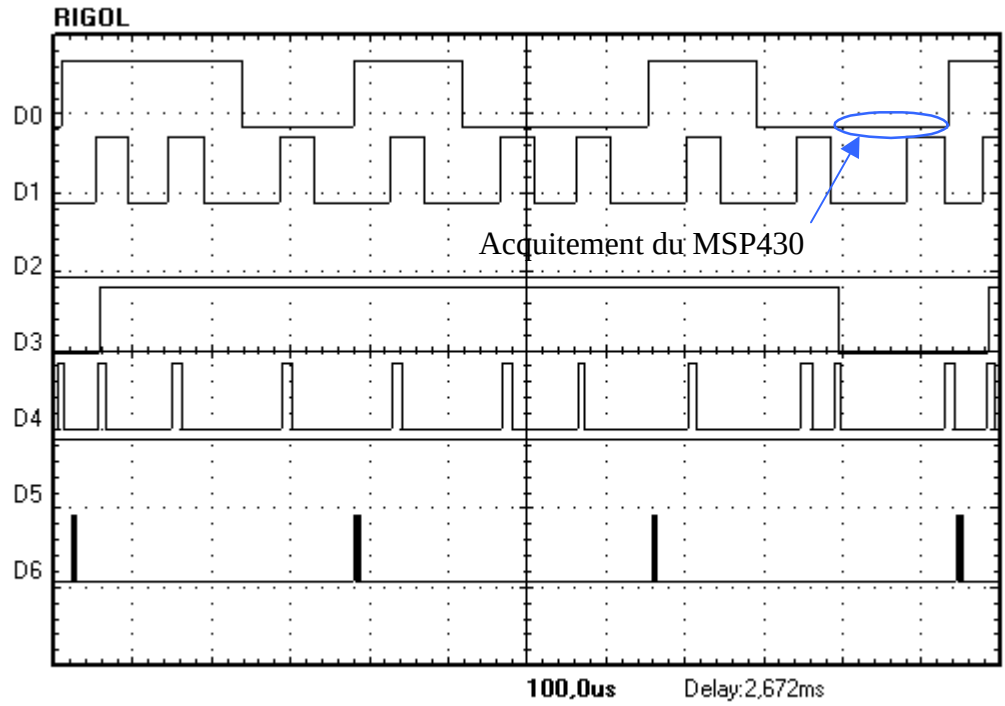
Lecture bit ds "Int_SCL" : A6 A5 A4 A3 A2 A1 A0 R/W soit 0x50 (conforme)

Acquitement adresse et premier octet



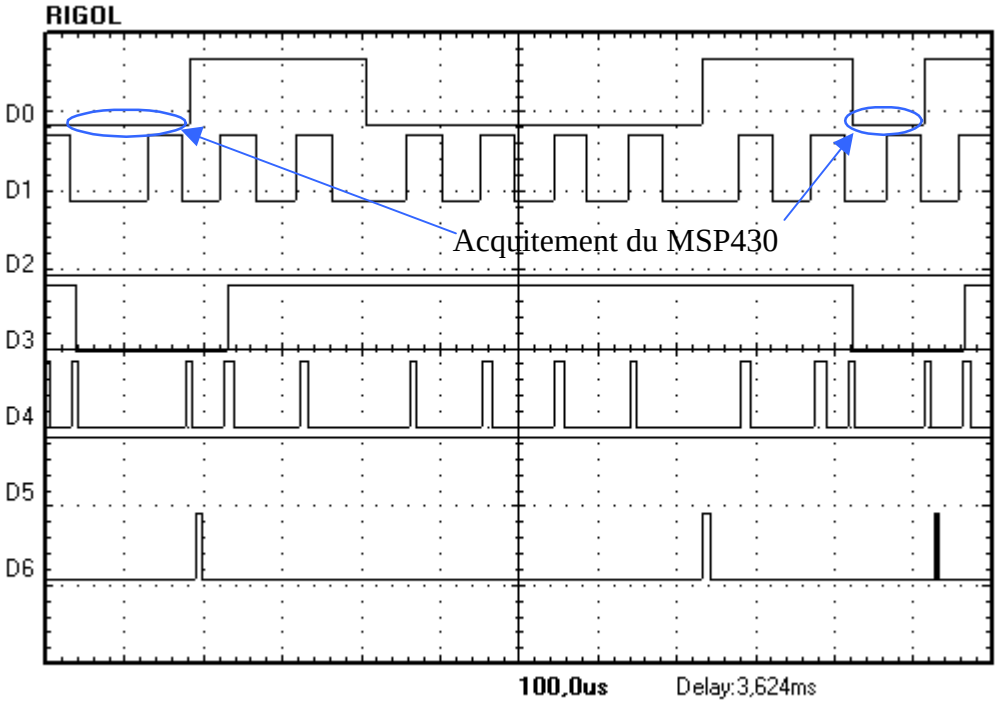
Lecture bit ds "Int_SCL" : D7 ---------- D0 , soit 0xE1 (conforme)

Octet 2



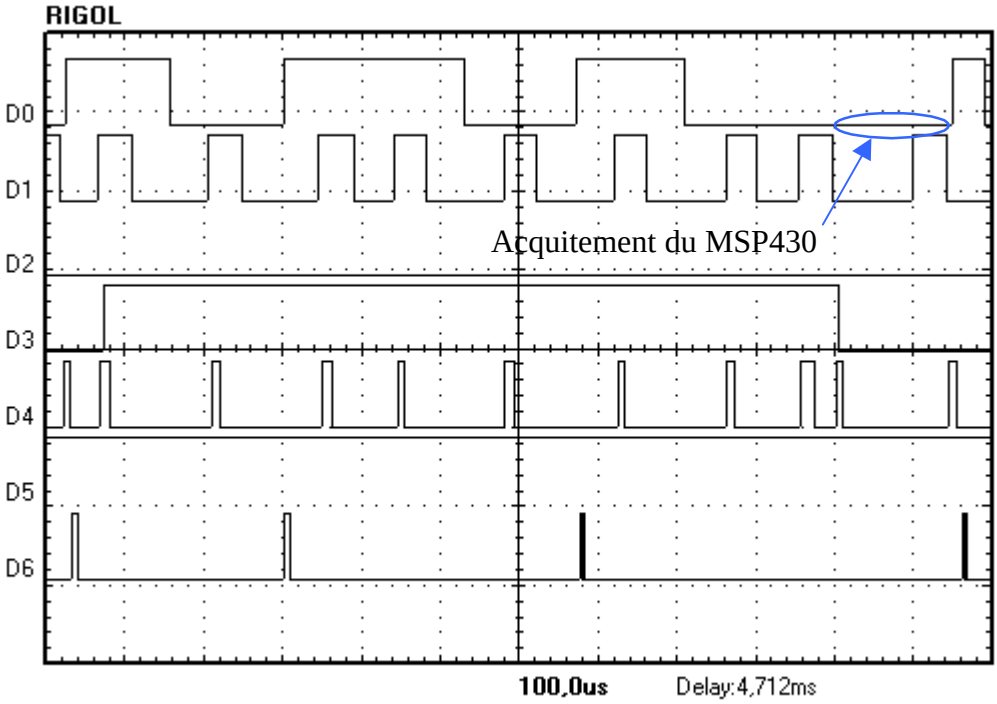
Octet 2 = 0b11010010 = 0xD2 (conforme)

Octet 3



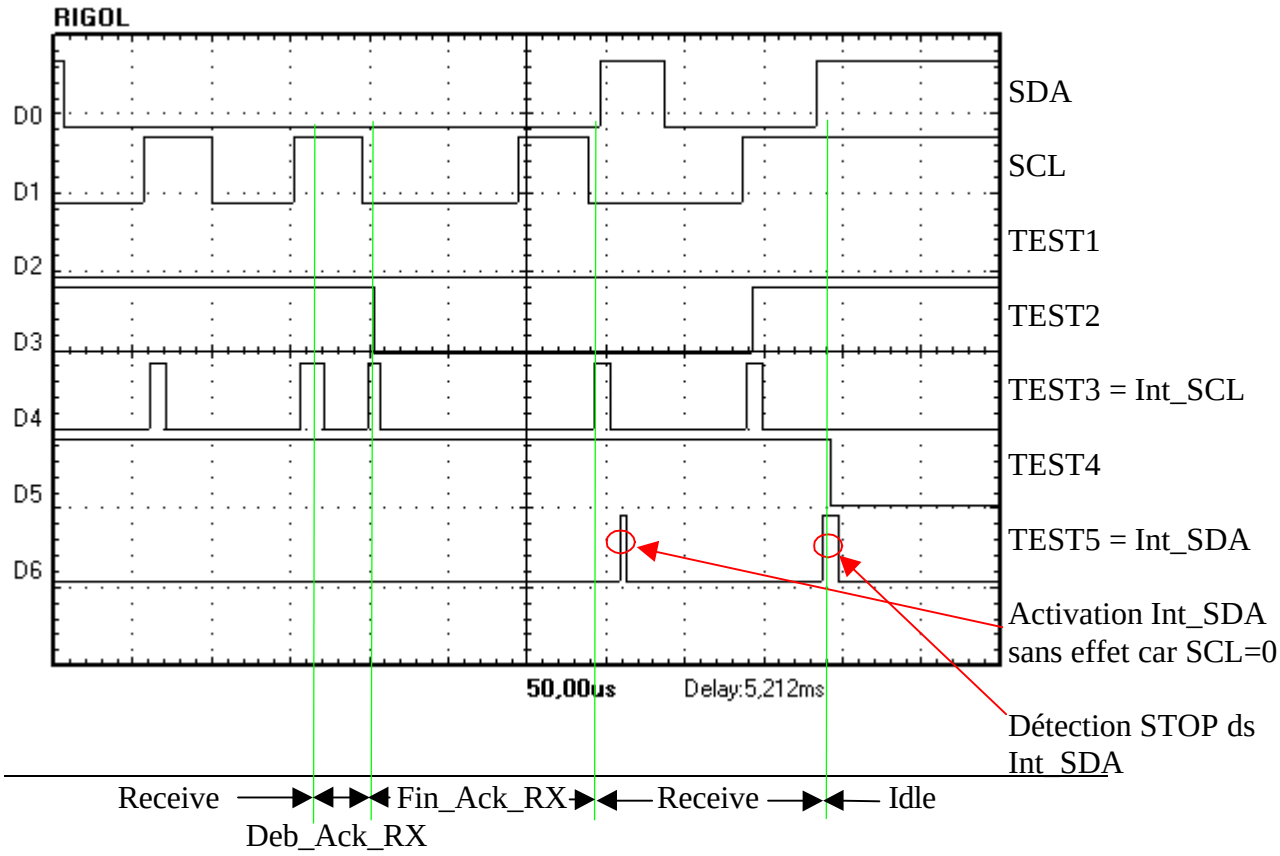
Octet 3 = 0b11000011 = 0xC3 (conforme)

Octet 4



Octet 4 = 0b10110100 = 0xB4 (conforme)

Stop


RIGOL
D0
D1
D2
D3
D4
D5
D6
SDA
SCL
TEST1
TEST2
TEST3 = Int_SCL
TEST4
TEST5 = Int_SDA
Activation Int_SDA
sans effet car SCL=0
50,00µs
Delay:5,212ms
Détection STOP ds
Int_SDA
Receive
Deb_Ack_RX
Fin_Ack_RX
Receive
Idle

4.3 Lecture

Les chronogrammes ont été relevés avec une nouvelle version du programme (05/02/07). Celui-ci place le microcontroleur en mode LPM3 avec une horloge DCO de 80kHz environ en l'absence de toute activité (pour minimiser la consommation).
Le CPU est "réveillé" au premier flanc descendant de SDA pour détecter la condition START.
On constate le délai d'activation du programme d'interruption "Int_SDA" illustré par la mise à "1" de TEST5 (600µS environ entre START et mise à "1" de TEST5). Cette lenteur est due à la faible fréquence F_{DCO} de 80kHz.
Le programme de test sur PC (Test_I2C.exe) laisse le temps au MSP430 de réagir en insérant un délai de 1mS entre les premiers flancs descendants de SDA et SCL comme on le vérifie sur le chronogramme.
L'horloge DCO est immédiatement basculée sur 4,9MHz environ après la détection de la condition START.
Le reste du traitement peut alors se faire à la vitesse nominale.
L'horloge DCO est reprogrammée à 80kHz à la détection de la condition STOP ce qui se constate par le prolongement conséquent de la durée du programme d'interruption "Int_SDA".

Nouvelle version de la boucle sans fin de la fonction "main" :

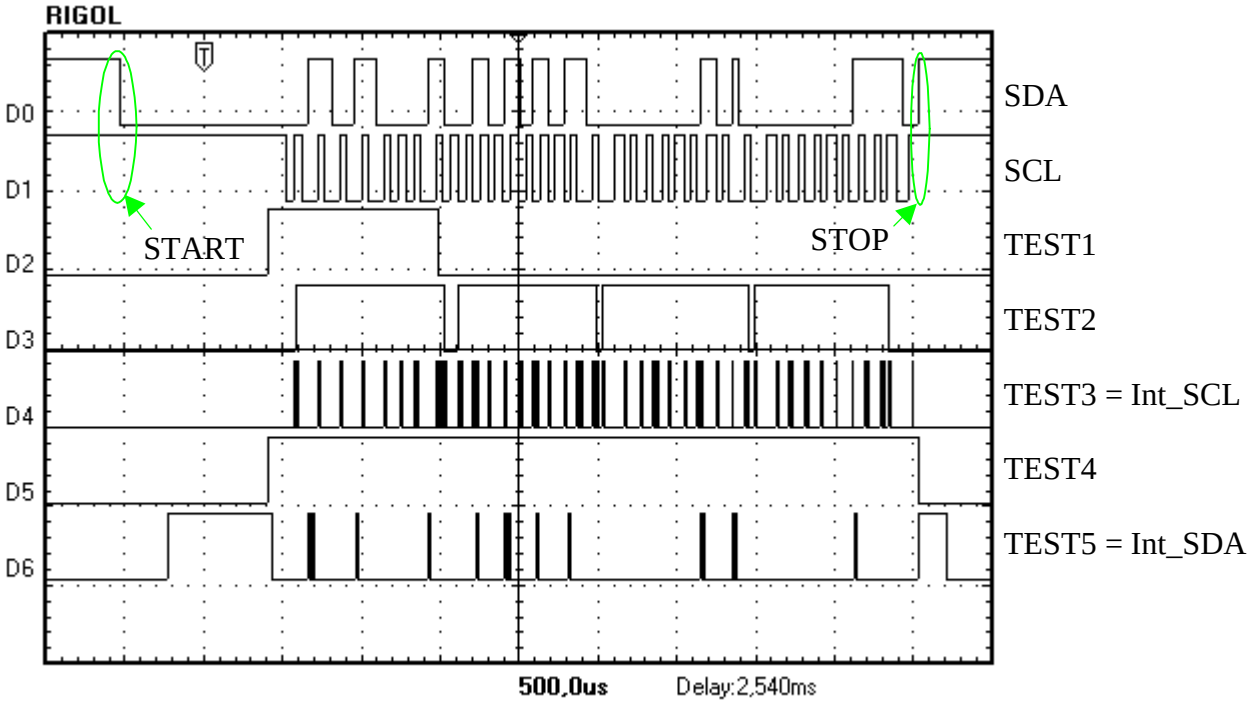
```
/*--------------------
   Boucle principale
 ---------------------
 Contrôle du mode de fonctionnement :
  - Controleur déconnecté et pas d'action sur le clavier :
    Low power mode 3 (LPM3) pour maintenir ACLK (horloge temps réel)
  - Controleur déconnecté et action en cours sur le clavier :
    "Active mode" et horloge CPU basse
  - Controleur connecté
    "Active mode" et horloge CPU haute */
 do
  {
   //Inv_TEST2; // Inverser TEST2
   if (I2C_State==Idle_I2C)
   {
    __low_power_mode_3(); // Tout à l'arrêt sauf ACLK
   }
  }
 while (1);
}
```

Nouvelle version du programme d'interruption "Int_Port2" :

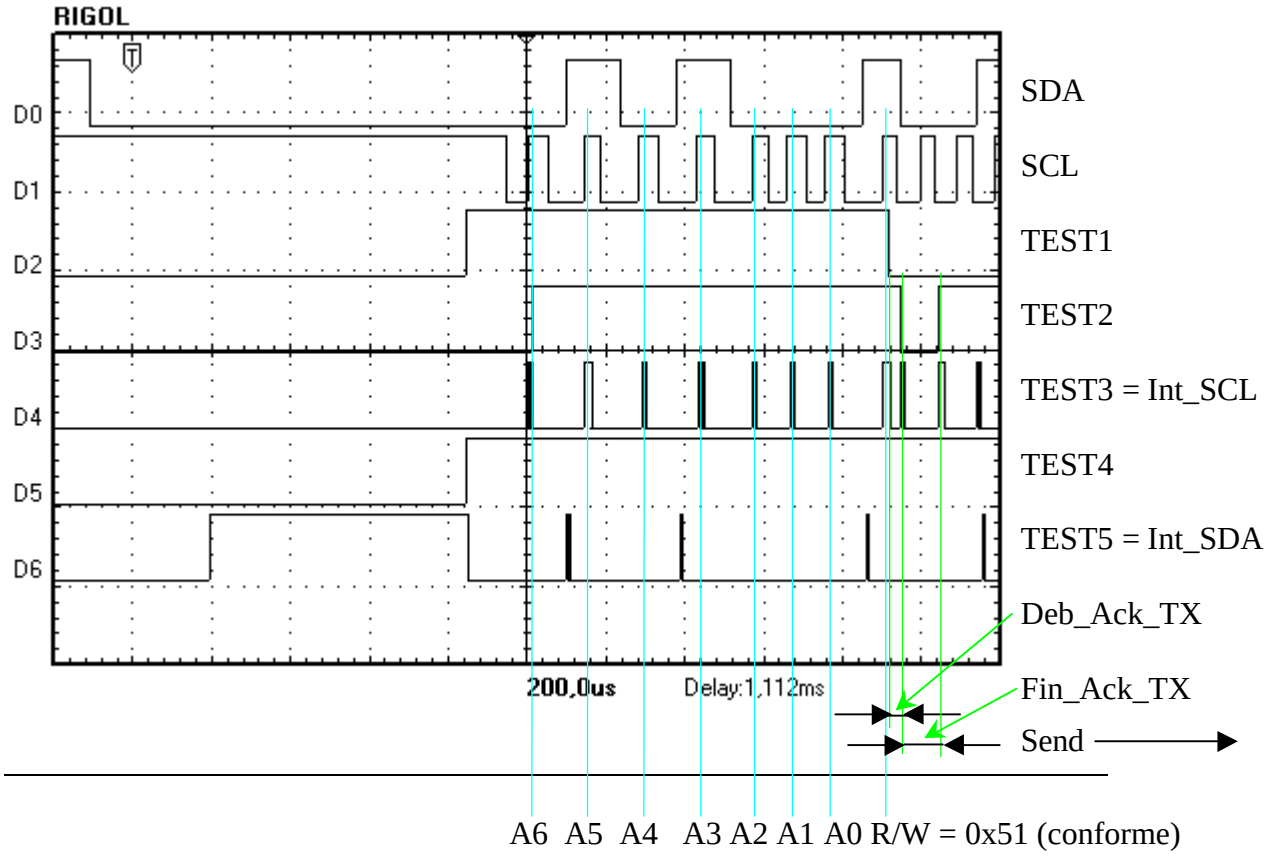
```
/*-----------------------
   Interruption Port P2
 -----------------------
 Provoquée :
  - au flanc descendant pour détecter la séquence START
  - au flanc montant pour détecter la séquence STOP
*/
 #pragma vector=PORT2_VECTOR
 __interrupt void Int_Port2()
 {
  Set_TEST5;   // Pour mesurer durée programme d'interruption
  if (Test_SCL)
   if (Test_SDA) // Flanc montant sur SDA et SCL="1" : STOP
   {
    Clr_TEST4;   // TEST4="0" au STOP
    SCL_Int_Inh; // Inhibition interruption SCL
    SDA_Int_HL;  // Interruption SDA sur flanc descendant (condition START)
    I2C_State=Idle_I2C; // Aucune séquence I2C en cours
    SetDCOClock(0,0); // DCO=0, RSEL= : FDCO = 80 kHz typique
   }
   else          // Flanc descendant sur SDA et SCL="1" : START
   {
    SetDCOClock(7,7);      // DCO=7, RSEL=7 : FDCO = 4,9 MHz typique
    Set_TEST1;             // TEST1="1" au START
    Set_TEST4;             // TEST4="1" au START
    Cpt_bits_I2C=8;        // Compteur bits = 8
    I2C_State=Receive_Adr; // Début de la séquence de réception de l'adresse
    Cpt_octets_I2C=0;      // Raz compteur d'octets
    SDA_Int_LH;  // Interruption SDA sur flanc montant (condition STOP)
    SCL_Int_LH;  // Interruption SCL sur flanc montant
    Clr_SCL_Flg; // Raz indicateur interruption SCL
    SCL_Int_En;  // Validation interruption SCL sur P1.0
   }
  Clr_SDA_Flg; // Raz indicateur interruption
  Clr_TEST5;   // Pour mesurer durée programme d'interruption
  __low_power_mode_off_on_exit();
 }
```

Les relevés sont ceux d'une lecture de 3 octets à l'adresse I2C (51)h.

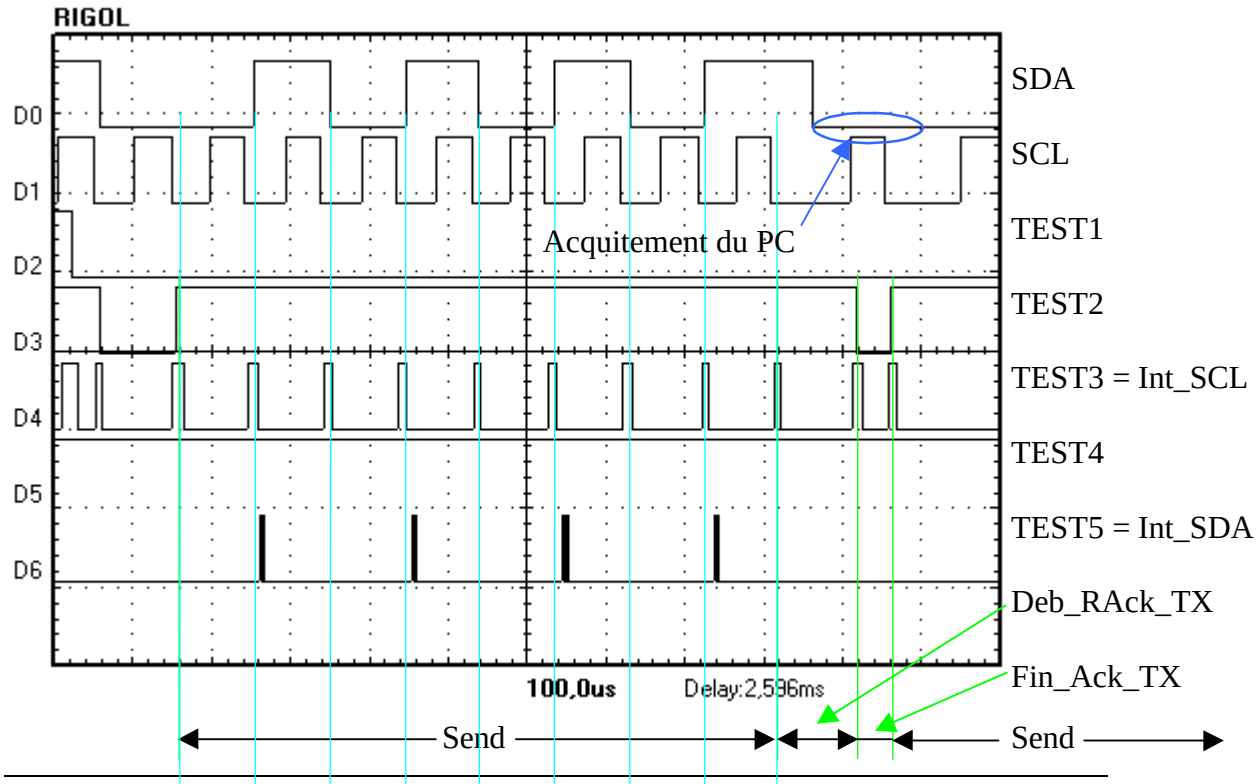
Trame complète



START + Adresse I2C

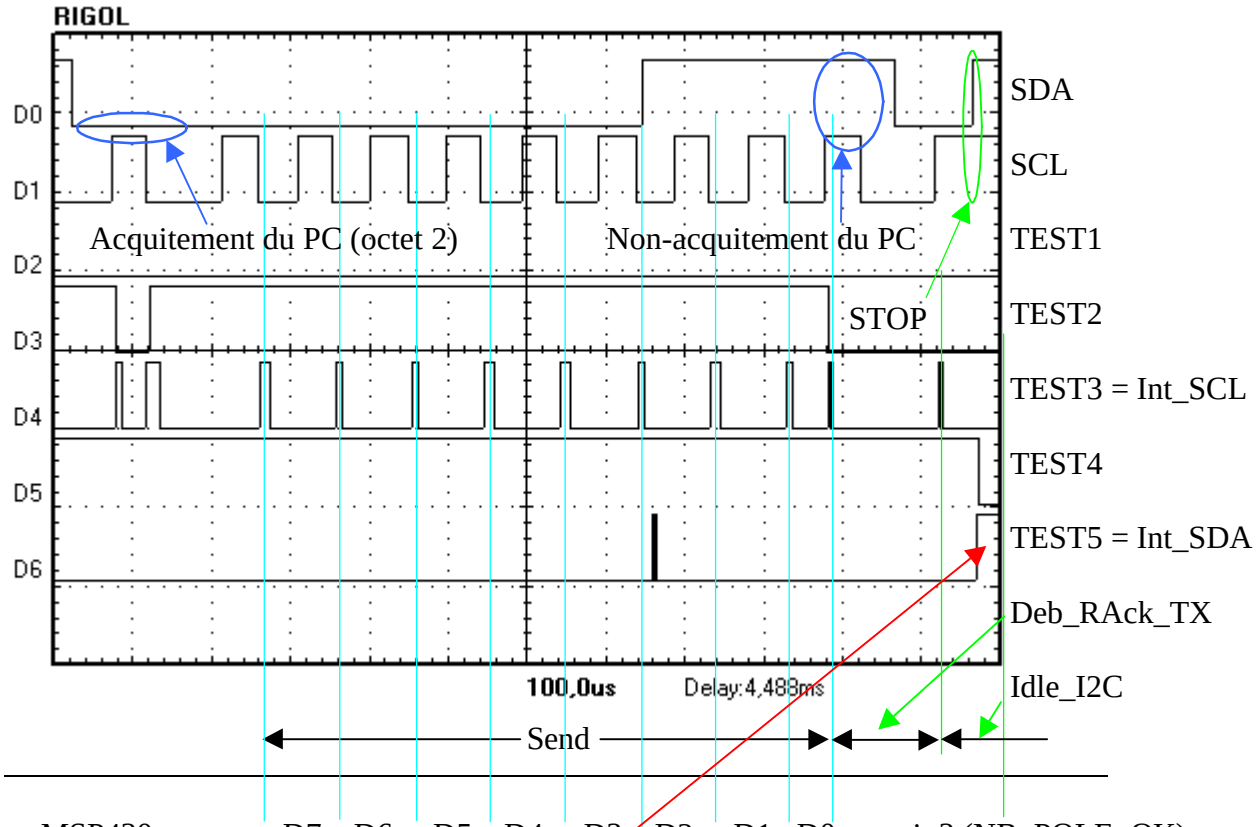


Octet 1



Transmission MSP430 : D7 D6 D5 D4 D3 D2 D1 D0 : soit 0x55 (code PAYS conforme)

Octet 3 (dernier)



Transmission MSP430 : D7 D6 D5 D4 D3 D2 D1 D0 : soit 3 (NB_POLE OK)

Note : TEST5 revient à "0" environ 150µS plus tard (fin de "Int_SDA" avec F_{DCO} = 80kHz).