Institut de technologie

Élève : Bielen Pierre

Stage professionnel.

Journal de stage.

Maître de stage : Monsieur Nicolas Martin.
Professeur référent : Monsieur André Dupont.
Lieu de stage :
Centre spatial de Liège
Avenue du Pré-Aily, 4031 Angleur(Belgique)

Table des matières

Cahier des charges.

1ere étape :

- Écrire des routines qui permettent les échanges bidirectionnels par port USB avec la carte MSP430. Donc aussi bien en entrée qu'en sortie. Le but est de pouvoir envoyer des consignes (chiffres, nombres ou texte) vers le MSP430 pour lui faire exécuter des commandes (de position de moteur par exemple). Idem dans l'autre sens, pouvoir afficher à l'écran du PC des messages (chiffres, nombre ou texte) venant du MSP430

- Ensuite, si le MSP430 est en mode data logger et qu'il envoie des données (chiffres ou nombres), pouvoir les afficher à l'écran du PC sous forme graphique (courbes avec l'heure par exemple en abscisse, si nécessaire). Plus généralement, pouvoir afficher sous forme graphique à l'écran du PC une série de données stockées dans un fichier. De même, il faut pouvoir sauver les données reçues dans un fichier. Et pouvoir relire le fichier pour l'afficher à l'écran

2e étape :

Utilisation du module de mesure de température et d'humidité

- Pouvoir afficher la température (humidité optionnelle) mesurée par le capteur, l'afficher à l'écran du PC et pouvoir stocker la température toutes les 10 secondes (ou une autre valeur, plus longue)

- Afficher la température en continu sur l'affichage 7 segments

- Afficher la température en continu sur l'affichage LCD

3e étape :

Utilisation des LEDs

- Faire clignoter la (ou les) LED à une fréquence entre 1 Hz et 1 kHz

- Pouvoir régler le rapport cyclique, de 0 à 100 %

- Afficher la fréquence de clignotement et le rapport cyclique

- Cela suppose évidemment que l'on peut choisir au clavier ou avec la souris (ou le PAD) les valeurs désirées

4e étape :

Utilisation du moteur pas à pas

- Raccorder le driver de moteur pas à pas à la carte MSP430 et le moteur à son driver

- Il faut le faire tourner à une vitesse variant de 1 t/min à 1 t/sec

- Prévoir l'affichage des paramètres de fonctionnement

- Prévoir la possibilité de choisir au clavier ou avec la souris (ou le PAD) les valeurs désirées

5e étape :

- Prévoir un fichier script pour programmer un scénario qui s'exécuterait automatiquement

6e étape :

- Tester le fichier script avec tous les devices exhaustivement
- Rédiger le mémoire

Matériel et logiciels.

Semaine du 3 février 2021 au 5 février 2021.

Mercredi, jeudi, vendredi.

A faire :

- Configuration de l’horloge.
- Routine de communication bidirectionnel via le protocole « uart ».

Travail effectué :

- Création des librairies suivante
  - CLOCK.
  - UART.
  - TOOLS.
  - LCD.
- Teste de l’envoi de données.
  - Numérique.
  - Texte.

Le protocole d’envoie de données.

- Envoi de données numérique.
  - Send a char at 255 (begin sending).
  - Convert value to string.
  - Send string.
  - Send a char at 255 (end of sending).
- Envoie de chaîne de catères.
  - Send a char at 254 (begin sending).
  - Send string.
  - Send a char at 255 (end of sending).

Archive du projet (zip à téléchargé depuis « github »).

STAGE 1 A

Semaine du 8 février 2021 au 12 février 2021.

Lundi, mardi, mercredi.

A faire :

- Réalisation du logiciel PC.
  - En mode réception.
    - Le logiciel doit pouvoir être placé en mode réception et en utilisation non bloquante.
    - Le logiciel doit pouvoir déterminé le type de donnée et les stocker en mémoire.

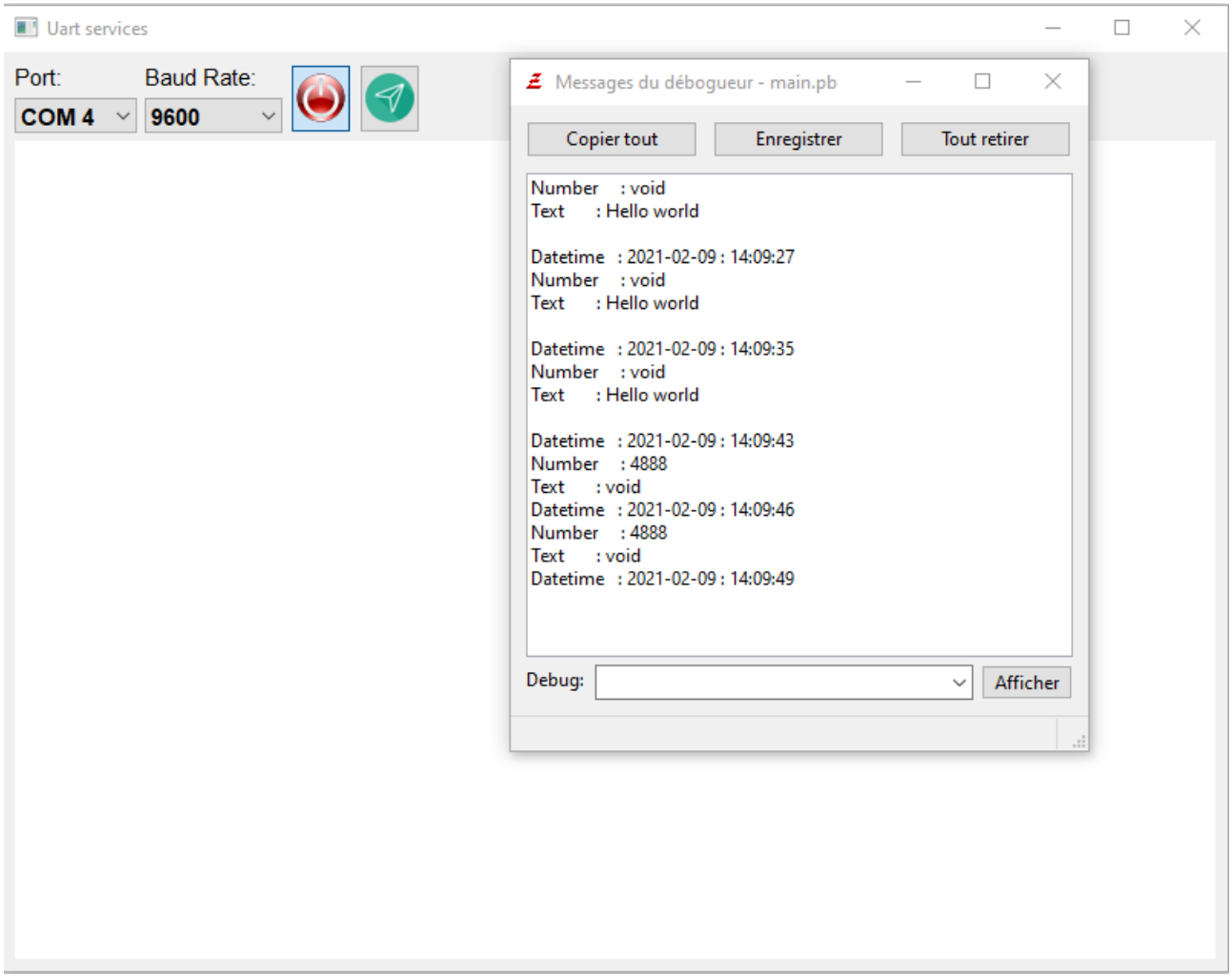
Protocole d’envoi de données (MCU & PC) :

- Number.
  - For say we begin the sending we send a char => 255
  - Send the number converter into a string.
  - For say send is finished we send a char =>255
- String.
  - For say we begin the sending we send a char => 254
  - Send the string.
  - For say send is finished we send a char =>255

Protocole de réception (MCU & PC).

- Number.
  - If we reveive a char at 255 we listen the port in number mode.
  - We push the char received into a buffer.
  - If we reveive a char at 255 we manage a speciphic treatment for the number value.
- String.
  - If we reveive a char at 254 we listen the port in string mode.
  - We push the char received into a buffer.
  - If we reveive a char at 255 we manage a speciphic treatment for the string value.

Test (MCU data send PC data receive) :



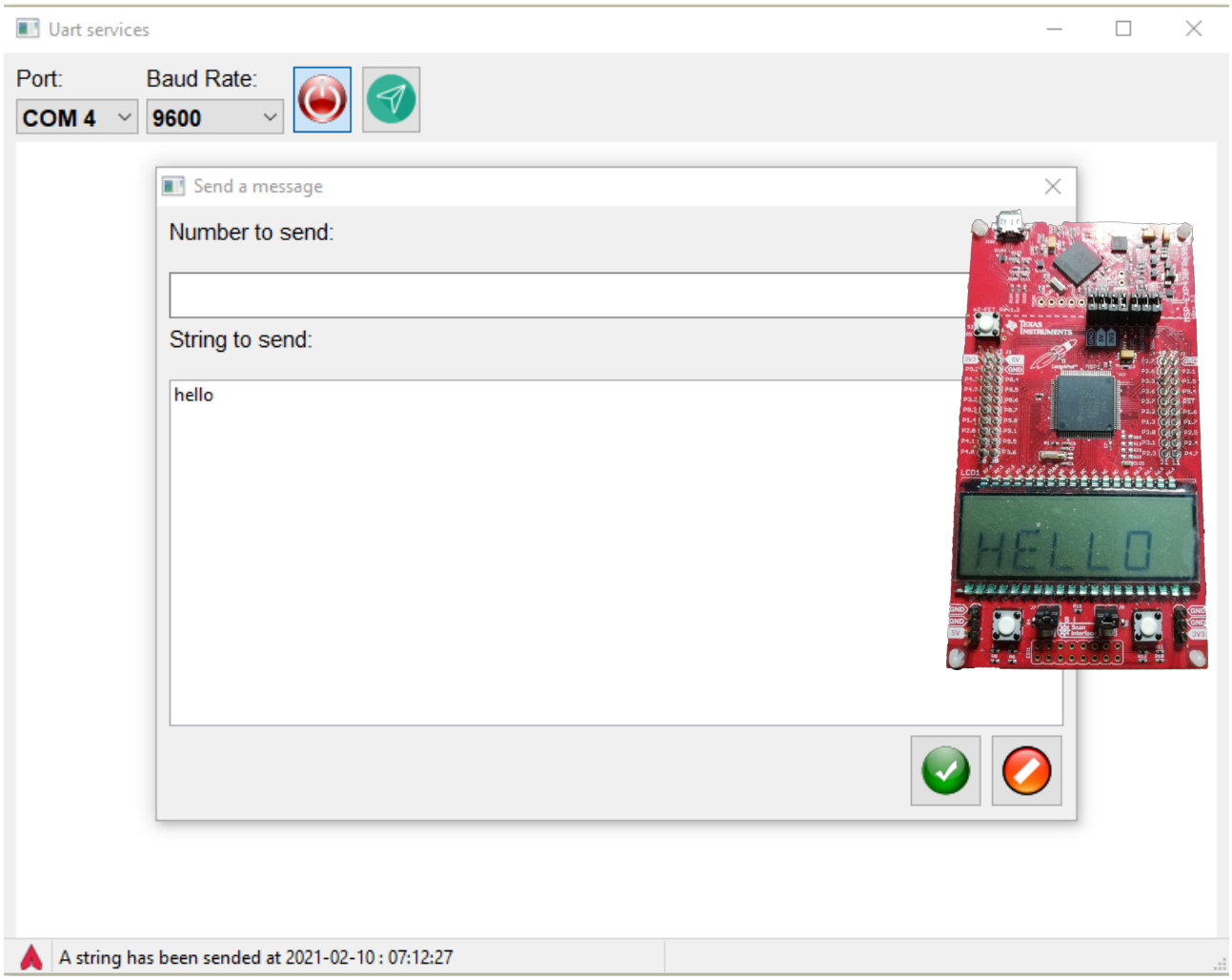
Archive du projet MCU (Étape 1 version 2) :

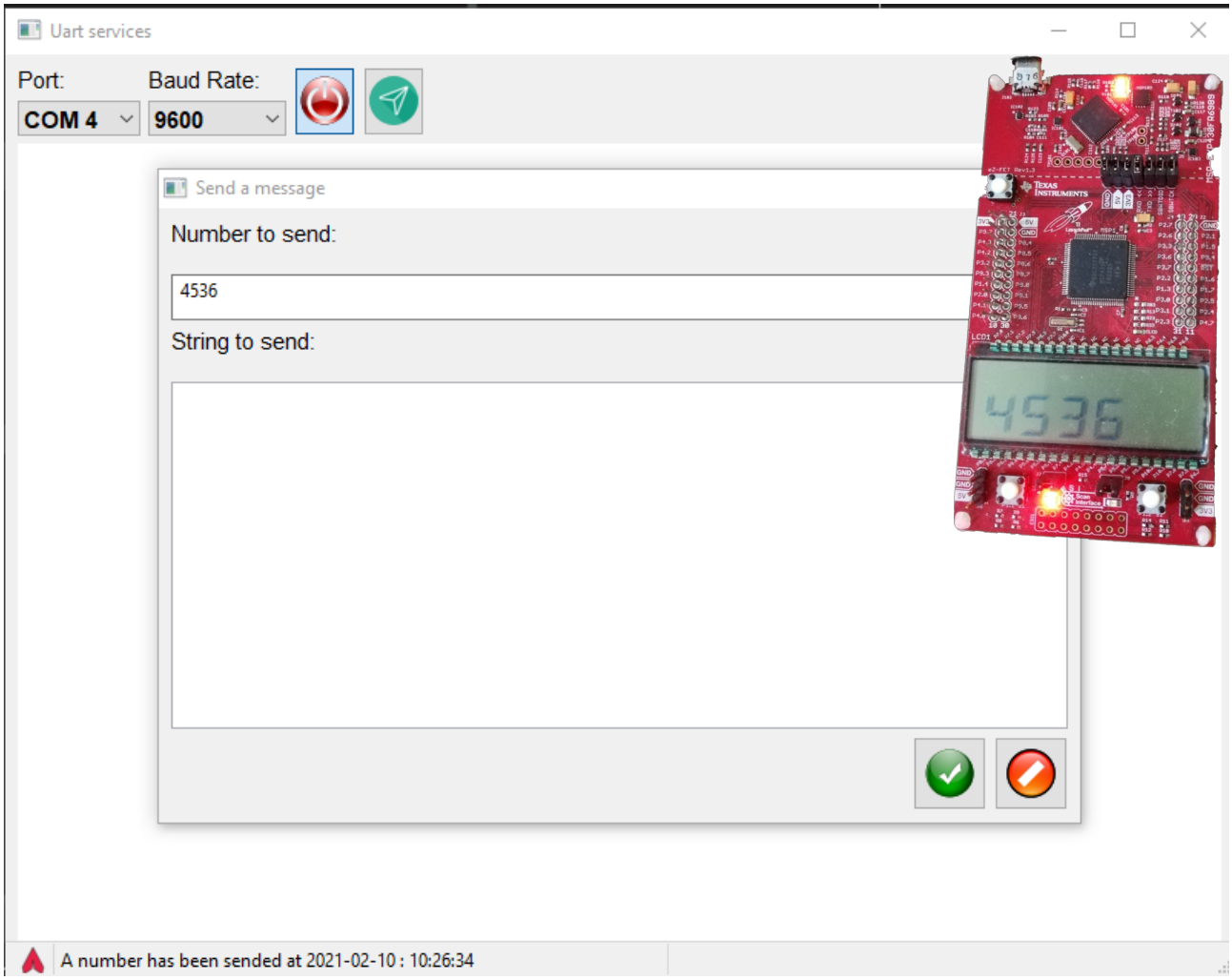
Etape 1 version 2.

Archive du projet PC (Étape 1 A) :

Etape 1 A (Purebasic Ap)

Test (MCU data receive PC data send) :




Uart services
Port:
COM 4
Baud Rate:
9600
Send a message
Number to send:
4536
String to send:
4536
A number has been sended at 2021-02-10 : 10:26:34

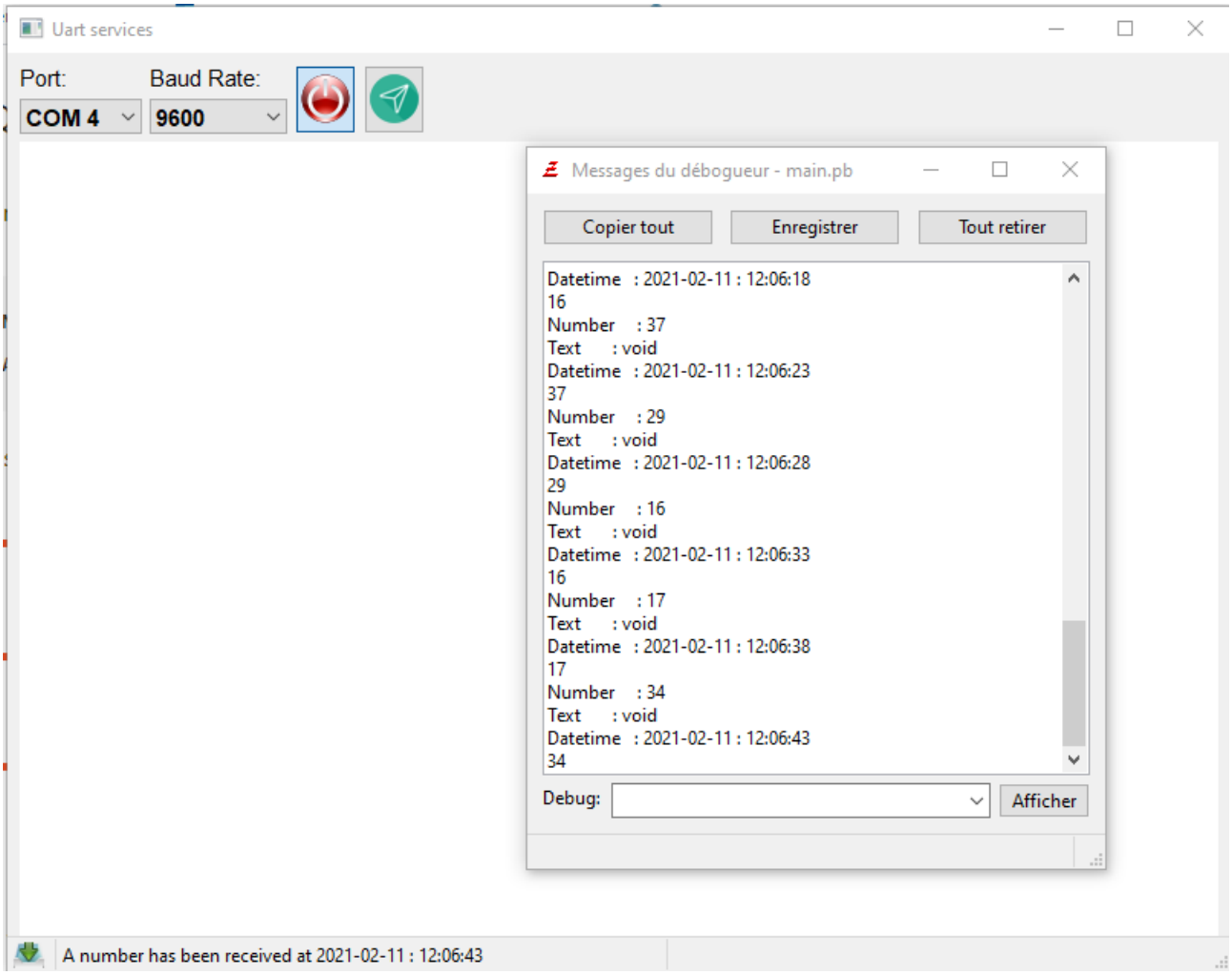
Consommation de ressource :

Jeudi :

A faire :

- Faire de sorte que le MCU envoi des nombre à des intervalles de temps régulier.
- Programme PC traité l’entrée de données.

Teste PC :


Uart services
Port:
Baud Rate:
COM 4
9600
Messages du débogueur - main.pb
Copier tout
Enregistrer
Tout retirer
Datetime : 2021-02-11 : 12:06:18
16
Number : 37
Text : void
Datetime : 2021-02-11 : 12:06:23
37
Number : 29
Text : void
Datetime : 2021-02-11 : 12:06:28
29
Number : 16
Text : void
Datetime : 2021-02-11 : 12:06:33
16
Number : 17
Text : void
Datetime : 2021-02-11 : 12:06:38
17
Number : 34
Text : void
Datetime : 2021-02-11 : 12:06:43
34
Debug:
Afficher
A number has been received at 2021-02-11 : 12:06:43

Archive du code MCU :

Data logger sender

Archive du code PC :

Data logger receiver V1

Configuration de l’horloge pour le « MSP430FR6989 ».

Étapes à suivre :

1. Autoriser « FRAM » à fonctionner à une vitesse supérieure de 8MHZ.
2. Déverrouiller les registres $CSTL_X$.
3. Régler le « DCO » à 16MHZ.
4. Définir les différents quartz pour les différentes horloges « ACLK ; SMLK ; MCLK ».
5. Régler les diviseurs de fréquences à 1.
6. Verrouiller le registre $CSTL_X$.

Unlock « FRAM »

```
// Configure one FRAM waitstate as required by the device datasheet for
MCLK
   // operation beyond 8MHz _before_ configuring the clock system.
   FRCTL0 = FRCTLPW | NWAITS_1;
```

Unlock CTL_X.

Figure 3-5. CTL0 Register

15	14	13	12	11	10	9	8
KEY							
R/W-96h							
7	6	5	4	3	2	1	0
RESERVED							
R-0h							

Table 3-4. CTL0 Register Field Descriptions

Bit	Field	Type	Reset	Description
15-8	KEY	R/W	96h	CSKEY password. Must always be written with A5h; a PUC is generated if any other value is written. Always reads as 96h. After the correct password is written, all CS registers are available for writing. A5h (W) = 0xA5
7-0	RESERVED	R	0h	Reserved. Always reads as 0.

```
//-- Unlock CS registers
CSCTL0_H = CSKEY >> 8;
```

Set clock at 16MHZ.

Figure 3-6. CTL1 Register

15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED	DCORSEL	RESERVED		DCOFSEL			RESERVED
R-0h	R/W-0h	R-0h		R/W-6h			R-0h

Table 3-5. CTL1 Register Field Descriptions

Bit	Field	Type	Reset	Description
15-7	RESERVED	R	0h	Reserved. Always reads as 0.
6	DCORSEL	R/W	0h	DCO range select. For high speed devices, this bit can be written by the user. For low speed devices, it is always reset. See description of DCOFSEL bit for details.
5-4	RESERVED	R	0h	Reserved. Always reads as 0.
3-1	DCOFSEL	R/W	6h	DCO frequency select. Selects frequency settings for the DCO. Values shown below are approximate. Please refer to the device specific datasheet. 0h (R/W) = If DCORSEL = 0: 1 MHz; If DCORSEL = 1: 1 MHz 1h (R/W) = If DCORSEL = 0: 2.67 MHz; If DCORSEL = 1: 5.33 MHz [illegible] 3h (R/W) = If DCORSEL = 0: 4 MHz; If DCORSEL = 1: 8 MHz 4h (R/W) = If DCORSEL = 0: 5.33 MHz; If DCORSEL = 1: 16 MHz 5h (R/W) = If DCORSEL = 0: 7 MHz; If DCORSEL = 1: 21 MHz 6h (R/W) = If DCORSEL = 0: 8 MHz; If DCORSEL = 1: 24 MHz 7h (R/W) = If DCORSEL = 0: Reserved. Defaults to 8. It is not recommended to use this setting; If DCORSEL = 1: Reserved. Defaults to 24. It is not recommended to use this setting
0	RESERVED	R	0h	Reserved. Always reads as 0.

Define clock « ACLK ; SMCLK ; MCLK »

Figure 3-7. CTL2 Register

15	14	13	12	11	10	9	8
RESERVED					SELA		
R-0h					R/W-0h		
7	6	5	4	3	2	1	0
RESERVED	SELS			RESERVED	SELM		
R-0h	R/W-3h			R-0h	R/W-3h		

Table 3-6. CTL2 Register Field Descriptions

Bit	Field	Type	Reset	Description
15-11	RESERVED	R	0h	Reserved. Always reads as 0.
10-8	SELA	R/W	0h	Selects the ACLK source 0h (R/W) = LFXTCLK : LFXTCLK when LFXT available, otherwise VLOCLK. 1h (R/W) = VLOCLK : VLOCLK 2h (R/W) = LFMODCLK : LFMODCLK
7	RESERVED	R	0h	Reserved. Always reads as 0.
6-4	SELS	R/W	3h	Selects the SMCLK source 0h (R/W) = LFXTCLK : LFXTCLK when LFXT available, otherwise VLOCLK. 1h (R/W) = VLOCLK : VLOCLK 2h (R/W) = LFMODCLK : LFMODCLK 3h (R/W) = DCOCLK : DCOCLK 4h (R/W) = MODCLK : MODCLK 5h (R/W) = HFXTCLK : HFXTCLK when HFXT available, otherwise DCOCLK.
3	RESERVED	R	0h	Reserved. Always reads as 0.
2-0	SELM	R/W	3h	Selects the MCLK source 0h (R/W) = LFXTCLK : LFXTCLK when LFXT available, otherwise VLOCLK 1h (R/W) = VLOCLK : VLOCLK 2h (R/W) = LFMODCLK : LFMODCLK 3h (R/W) = DCOCLK : DCOCLK 4h (R/W) = MODCLK : MODCLK 5h (R/W) = HFXTCLK : HFXTCLK when HFXT available, otherwise DCOCLK

```
//-- Set ACLCK use VLOVK & SMCLK & MCLK use  DCO
CSCTL2 = SELA__VLOCLK | SELS__DCOCLK | SELM__DCOCLK;
```

Define clock divider.

Figure 3-8. CTL3 Register

15	14	13	12	11	10	9	8
RESERVED					DIVA		
R-0h					R/W-0h		
7	6	5	4	3	2	1	0
RESERVED	DIVS			RESERVED	DIVM		
R-0h	R/W-3h			R-0h	R/W-3h		

Table 3-7. CTL3 Register Field Descriptions

Bit	Field	Type	Reset	Description
15-11	RESERVED	R	0h	Reserved. Always reads as 0.
10-8	DIVA	R/W	0h	ACLK source divider. Divides the frequency of the ACLK clock source. 0h (R/W) = 1 : /1 1h (R/W) = 2 : /2 2h (R/W) = 4 : /4 3h (R/W) = 8 : /8 4h (R/W) = 16 : /16 5h (R/W) = 32 : /32
7	RESERVED	R	0h	Reserved. Always reads as 0.
6-4	DIVS	R/W	3h	SMCLK source divider. Divides the frequency of the SMCLK clock source. 0h (R/W) = 1 : /1 1h (R/W) = 2 : /2 2h (R/W) = 4 : /4 3h (R/W) = 8 : /8 4h (R/W) = 16 : /16 5h (R/W) = 32 : /32
3	RESERVED	R	0h	Reserved. Always reads as 0.
2-0	DIVM	R/W	3h	MCLK source divider. Divides the frequency of the MCLK clock source. 0h (R/W) = 1 : /1 1h (R/W) = 2 : /2 2h (R/W) = 4 : /4 3h (R/W) = 8 : /8 4h (R/W) = 16 : /16 5h (R/W) = 32 : /32

```
//-- Set all dividers
  CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1
;
```

Unlock CTL$_x$

```
//-- Lock CS registers
  CSCTL0_H = 0;
```

Code du fichier Header.

```
/*
 * LIBRARY NAME : CLOCK
 * AUTHOR       : Bielen Pierre
 * PROCESS      : manage the clock device frequency
 * CMU          : MSP430FR6989
 */

#ifndef LIB_CLOCK_H_
#define LIB_CLOCK_H_

#include <msp430.h>

/*
 * Function name    : CLOCK_initTo16MHz
 * Parameter        : void
 * Returned Value   : void
 * Process          : initiation of clock at 16MHZ
 * Note             ;
 */
void CLOCK_initTo16MHz(void);



#endif /* LIB_CLOCK_H_ */
```

Code du fichier C.

```
/*
 * LIBRARY NAME : CLOCK
 * AUTHOR       : Bielen Pierre
 * PROCESS      : manage the clock device frequency
 * CMU          : MSP430FR6989
 */

#include "CLOCK.h"

void CLOCK_initTo16MHz(void)
{
  // Configure one FRAM waitstate as required by the device datasheet for MCLK
  // operation beyond 8MHz _before_ configuring the clock system.
  FRCTL0 = FRCTLPW | NWAITS_1;
  //-- For LFXT
  PJSEL0 = BIT4 | BIT5;
  // Clock System Setup
```

```
  //-- Unlock CS registers
  CSCTL0_H = CSKEY >> 8;
  //-- Set DCO to 16MHz
  CSCTL1 = DCOFSEL_4 | DCORSEL;
  //-- Set ACLCK use VLOVK & SMCLK & MCLK use  DCO
  CSCTL2 = SELA__LFXTCLK | SELS__DCOCLK | SELM__DCOCLK;
  //-- Set all dividers
  CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;
  //-- Enable LFXT
  CSCTL4 &= ~LFXTOFF;
  do
  {
    //-- Clear LFXT fault flag
    CSCTL5 &= ~LFXTOFFG;
    //-- Test oscillator fault flag
    SFRIFG1 &= ~OFIFG;
  }while (SFRIFG1 & OFIFG);
  //-- Lock CS registers
  CSCTL0_H = 0;

}
```

Configuration de l'« UART ».

Étapes à suivre.

1. Configuration de la « clock ».
2. Calcul du « baud rate ».
3. Choix du compensateur d'erreur

Configuration de la « clock ».

Figure 30-12. UCAxCTLW0 Register

15	14	13	12	11	10	9	8
UCPEN	UCPAR	UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
UCSSELx		UCRXEIE	UCBRKIE	UCDORM	UCTXADDR	UCTXBRK	UCSWRST
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

7-6	UCSSELx	RW	0h	eUSCI_A clock source select. These bits select the BRCLK source clock. 00b = UCLK 01b = ACLK 10b = SMCLK 11b = SMCLK

Calcul du « baud rate ».

$$UCAXBR0 = \frac{Frequency}{Baud\,rate} \quad \text{par exemple} \quad \frac{32768}{9600} = 3{,}413$$

```
//---- 32768/9600
 UCA1BR0 = 3;
UCA1BR1 = 0;
```

Choix du compensateur d'erreur.

On utilise le tableau ci-dessous pour modifier la valeur de **UCAXMCTLW**.

Table 30-5. Recommended Settings for Typical Crystals and Baud Rates[1] (continued)

BRCLK	Baud Rate	UCOS16	UCBRx	UCBRFx	UCBRSx[2]	TX Error[2] (%)		RX Error[2] (%)	
						neg	pos	neg	pos
8388608	230400	1	2	4	0x92	-1.62	1.37	-3.56	2.06
8388608	460800	0	18	-	0x11	-2	3.37	-5.31	5.55
12000000	9600	1	78	2	0x0	0	0	0	0.04
12000000	19200	1	39	1	0x0	0	0	0	0.16
12000000	38400	1	19	8	0x65	-0.16	0.16	-0.4	0.24
12000000	57600	1	13	0	0x25	-0.16	0.32	-0.48	0.48
12000000	115200	1	6	8	0x20	-0.48	0.64	-1.04	1.04
12000000	230400	1	3	4	0x2	-0.8	0.96	-1.84	1.84
12000000	460800	1	1	10	0x0	0	1.76	0	3.44
16000000	9600	1	104	2	0xD6	-0.04	0.02	-0.09	0.03
16000000	19200	1	52	1	0x49	-0.08	0.04	-0.1	0.14
16000000	38400	1	26	0	0xB6	-0.08	0.16	-0.28	0.2
16000000	57600	1	17	5	0xDD	-0.16	0.2	-0.3	0.38
16000000	115200	1	8	10	0xF7	-0.32	0.32	-1	0.36
16000000	230400	1	4	5	0x55	-0.8	0.64	-1.12	1.76
16000000	460800	1	2	2	0xBB	-1.44	1.28	-3.92	1.68
16777216	9600	1	109	3	0xB5	-0.03	0.02	-0.05	0.06
16777216	19200	1	54	9	0xEE	-0.06	0.06	-0.11	0.13
16777216	38400	1	27	4	0xFB	-0.11	0.1	-0.33	0
16777216	57600	1	18	3	0x44	-0.16	0.15	-0.2	0.45
16777216	115200	1	9	1	0xB5	-0.31	0.31	-0.53	0.78
16777216	230400	1	4	8	0xEE	-0.75	0.74	-2	0.87
16777216	460800	1	2	4	0x92	-1.62	1.37	-3.56	2.06
20000000	9600	1	130	3	0x25	-0.02	0.03	0	0.07
20000000	19200	1	65	1	0xD6	-0.06	0.03	-0.1	0.1
20000000	38400	1	32	8	0xEE	-0.1	0.13	-0.27	0.14
20000000	57600	1	21	11	0x22	-0.16	0.13	-0.16	0.38
20000000	115200	1	10	13	0xAD	-0.29	0.26	-0.46	0.66
20000000	230400	1	5	6	0xEE	-0.67	0.51	-1.71	0.62
20000000	460800	1	2	11	0x92	-1.38	0.99	-1.84	2.8
8000000	115200	1	4	5	0x55	-0.8	0.64	-1.12	1.76
8000000	230400	1	2	2	0xBB	-1.44	1.28	-3.92	1.68
8000000	460800	0	17	-	0x4A	-2.72	2.56	-3.76	7.28
8388608	9600	1	54	9	0xEE	-0.06	0.06	-0.11	0.13
8388608	19200	1	27	4	0xFB	-0.11	0.1	-0.33	0
8388608	38400	1	13	10	0x55	-0.21	0.21	-0.55	0.33
8388608	57600	1	9	1	0xB5	-0.31	0.31	-0.53	0.78
8388608	115200	1	4	8	0xEE	-0.75	0.74	-2	0.87

Code du fichier header.

```
/*
 * LIBRARY NAME : UART
 * AUTHOR       : Bielen Pierre
 * PROCESS      : manage the serial connection through UART protocol
 * CMU          : MSP430FR6989
 */

#ifndef LIB_UART_H_
#define LIB_UART_H_

#define UART_SMCLK_11500     0
#define UART_SMCLK_9600      1

#include <msp430.h>
#include "TOOLS.h"

/*
 * Function name    : UART_init
 * Parameter        : int baud_rate
 *                    you can chose between this value
 *                    SMCLK_9600
 *                    SMCLK_11500
 * Returned Value   : void
 * Process          : initiation of the UART protocol
 * Note             ;
 */
void UART_init(unsigned int baudRate);

/*
 * Function name    : UART_sendString
 * Parameter        : char* string
 *                    the message to send
 *                    WARNING : please finish your message with \n
 *                    Example : UART_sendString("my message\n")
 * Returned Value   : void
 * Process          : send a message through the uart
 * Note             ;
 */
void UART_sendString(char* string);

/*
 * Function name    : UART_sendByte
 * Parameter        : unsigned char byte
 *                    the byte to send
 * Returned Value   : void
 * Process          : send a byte through the uart
```

```
* Note             ;
*/
void UART_sendByte(unsigned char byte );

/*
* Function name    : UART_sendInteger
* Parameter        : int number
*                    the number to send
* Returned Value   : void
* Process          : send a integer through the uart
* Note             ;
*/
void UART_sendInteger(int number);
#endif /* LIB_UART_H_ */
```

Code du fichier C.

```
/*
* LIBRARY NAME : UART
* AUTHOR       : Bielen Pierre
* PROCESS      : manage the serial connection through UART protocol
* CMU          : MSP430FR6989
*/

#include "UART.h"

/*
*  Global variables (only for this file)
*/
char* messageToSend;
unsigned char wait = 0;
unsigned char rxData;
/*
*  public functions
*/
void UART_init(unsigned int baudRate)
{
  //-- set UART ports
  //---- eUSCI_A1 UART
  P3SEL0 |= BIT4 + BIT5;
  //---- eUSCI_A1 UART
  P3SEL1 &= ~(BIT4 + BIT5);
  // Configure USCI_A0 for UART mode
  //-- Put eUSCI in reset
  UCA1CTLW0 = UCSWRST;
  switch(baudRate)
  {
```

```
    case UART_SMCLK_11500:
      //-- CLK = SMCLK
      UCA1CTLW0 |= UCSSEL__SMCLK;
      //-- Baud Rate Setting
      //---- Use Table 30-5 in Family User Guide
      UCA1BR0 = 8;
      UCA1BR1 = 0;
      //---- 0xF700 is UCBRSx = 0xF7
      UCA1MCTLW |= UCOS16 | UCBRF_10 | 0xF700;
      break;
    case UART_SMCLK_9600:
      //-- CLK = SMCLK
      UCA1CTLW0 |= UCSSEL__SMCLK;
      //-- Baud Rate Setting
      //---- Use Table 30-5 in Family User Guide
      UCA1BR0 = 104;
      UCA1BR1 = 0;
      //---- 0xD600 is UCBRSx = 0xD6
      UCA1MCTLW |= UCOS16 | UCBRF_2 | 0xD600;
      break;
  }

  //-- Initialize eUSCI
  UCA1CTLW0 &= ~UCSWRST;
  //-- Enable USCI_A0 RX interrupt
  UCA1IE |= UCRXIE;
}

void UART_sendString(char* string)
{
  //-- wait if uart is busy to send a message
  while(wait == 1);
  //-- copy the message to send
  messageToSend = string;
  //-- turn on TX complete
  UCA1IE |= UCTXCPTIE;
  //-- clears TX complete flag
  UCA1IFG &= ~UCTXCPTIFG;
  //-- put char through UART A1
  unsigned char c = *messageToSend;
  if (c != 0)
  {
    UCA1TXBUF = c;
    *(messageToSend)++;
    wait = 1;
  }
}
```

```
void UART_sendByte(unsigned char byte )
{
  //-- wait if uart is busy to send a message
  while(wait == 1);
  UCA1TXBUF = byte;
}

void UART_sendInteger(int number)
{
  char buffer[33];
  char* string;
  string = TOOLS_itoa(number, buffer, 10);
  UART_sendString(string);
}

/*
 * Interrupt service routines
 */
__attribute__((interrupt(USCI_A1_VECTOR)))
void USCI_A1_ISR(void)
{
  //-- read the next character from message to send
  unsigned char c = *messageToSend;
  //-- look for entry data
  rxData = UCA1RXBUF;
  //-- if the message to send is not empty or totally send
  if(c != 0)
  {
    //-- push the character into the uart buffer
    UCA1TXBUF = c;
    //-- go to next character
    *(messageToSend)++;
  }
  else
  {
    *messageToSend = 0;
    wait = 0;
    UCA1IE &= ~UCTXCPTIE;
  }
  //-- clear TX complete flag
  UCA1IFG &= ~UCTXCPTIFG;
}
```

Création d’une librairie Tools.

Cette librairie pourrat contenir plusieurs routines utilitaires. Actuelement elle comprend une rourine de convertion d’un nombre entier en chaîne de caractères.

Code du fichier Header.

```
/*
 * LIBRARY NAME : TOOLS
 * AUTHOR       : Bielen Pierre
 * PROCESS      : contains general purposes
 * CMU          : MSP430FR6989
 */

#ifndef LIB_TOOLS_H_
#define LIB_TOOLS_H_

#include <msp430.h>

/*
 * Function name    : TOOLS_itoa
 * Parameter        : int value
 *                    value to convert
 * Parameter        : char* result
 *                    string to result
 * Parameter        : int base
 *                    base to convert
 * Returned Value   : char*
 *                    result of conversion
 * Process          : convert a int to a string
 * Note             ;
 */

char* TOOLS_itoa(int value, char* buffer, int base);

#endif /* LIB_TOOLS_H_ */
```

Code du fichier C.

```c
/*
 * LIBRARY NAME : TOOLS
 * AUTHOR       : Bielen Pierre
 * PROCESS      : contains general purposes
 * CMU          : MSP430FR6989
 */

#include "TOOLS.h"

// inline function to swap two numbers
inline void swap(char *x, char *y) {
  char t = *x; *x = *y; *y = t;
}

// function to reverse buffer[i..j]
char* reverse(char *buffer, int i, int j)
{
  while (i < j)
    swap(&buffer[i++], &buffer[j--]);

  return buffer;
}

char* TOOLS_itoa(int value, char* buffer, int base)
{
  // invalid input
  if (base < 2 || base > 32)
    return buffer;

  // consider absolute value of number
  int n = abs(value);

  int i = 0;
  while (n)
  {
    int r = n % base;

    if (r >= 10)
      buffer[i++] = 65 + (r - 10);
    else
      buffer[i++] = 48 + r;

    n = n / base;
  }

  // if number is 0
```

```
  if (i == 0)
    buffer[i++] = '0';

  // If base is 10 and value is negative, the resulting string
  // is preceded with a minus sign (-)
  // With any other base, value is always considered unsigned
  if (value < 0 && base == 10)
    buffer[i++] = '-';

  buffer[i] = '\0'; // null terminate string

  // reverse the string and return it
  return reverse(buffer, 0, i - 1);

}
```

Le capteur de température et d’humidité « DHT11 ».

Spécifications techniques.

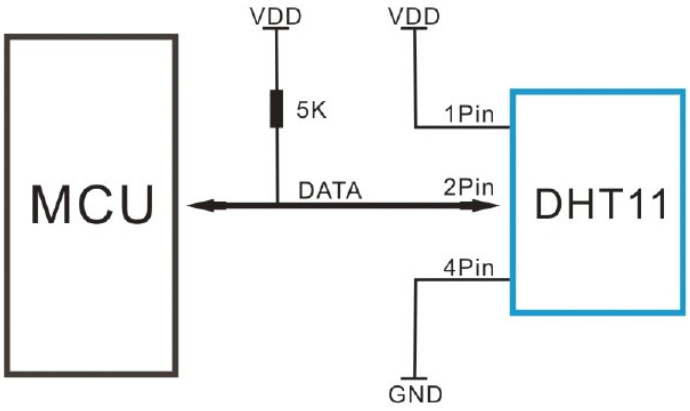
Parameters	Conditions	Minimum	Typical	Maximum
Humidity				
Resolution		1%RH	1%RH	1%RH
			8 Bit	
Repeatability			±1%RH	
Accuracy	25℃		±4%RH	
	0-50℃			±5%RH
Interchangeability	Fully Interchangeable			
Measurement Range	0℃	30%RH		90%RH
	25℃	20%RH		90%RH
	50℃	20%RH		80%RH
Response Time (Seconds)	1/e(63%)25℃, 1m/s Air	6 S	10 S	15 S
Hysteresis			±1%RH	
Long-Term Stability	Typical		±1%RH/year	
Temperature				
Resolution		1℃	1℃	1℃
		8 Bit	8 Bit	8 Bit
Repeatability			±1℃	
Accuracy		±1℃		±2℃
Measurement Range		0℃		50℃
Response Time (Seconds)	1/e(63%)	6 S		30 S

Connexion.

Ce capteur possède 4 broches de connections.

1. Broche d’alimentation entre (3,5V et 5,5V) en courant continu.
2. Broche de transfert de données en série.
3. Non utilisée.
4. Broche devant être liée à la masse.

Branchement typique.



Trame de données.

La trame de données comporte sur 40 bits.

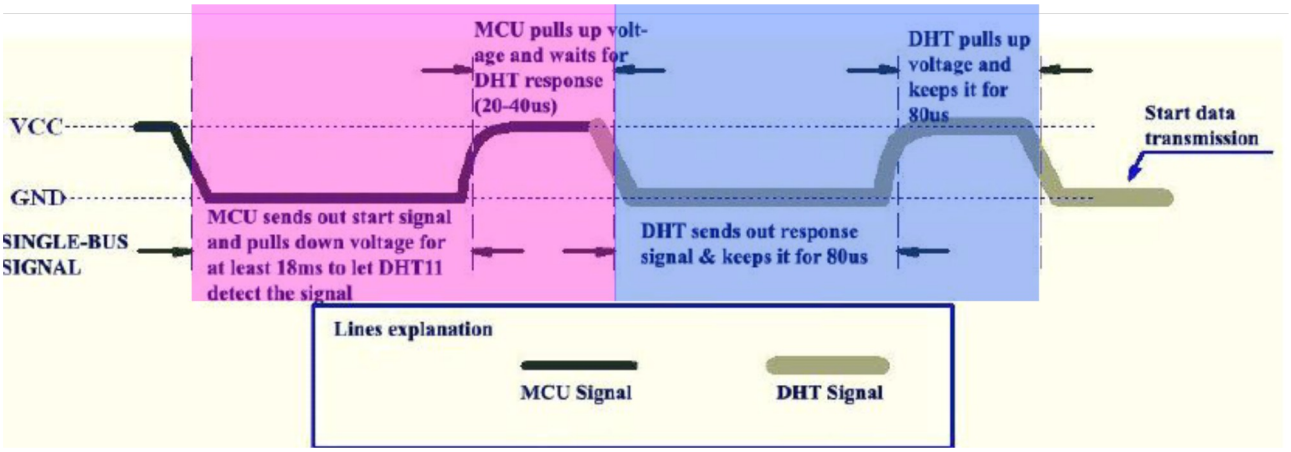
Nombre de bits.	Nom.	Description.
8	Humidity (integer)	La valeur **entière** du taux d'humidité.
8	Humidity (fractional)	La valeur **décimal** du taux d'humidité.
8	Temperature (integer)	La valeur **entière** de la température.
8	Temperature (fractional)	La valeur **décimal** de la température.
8	Checksum	Valeur de **vérification** devant être **égal** à la somme des **4 valeurs** précédentes.

Protocole de communication.

Le « DHT11 » utilise un seul port de communication bidirectionnel.

Synchronisation entre le MCU et le DHT11.

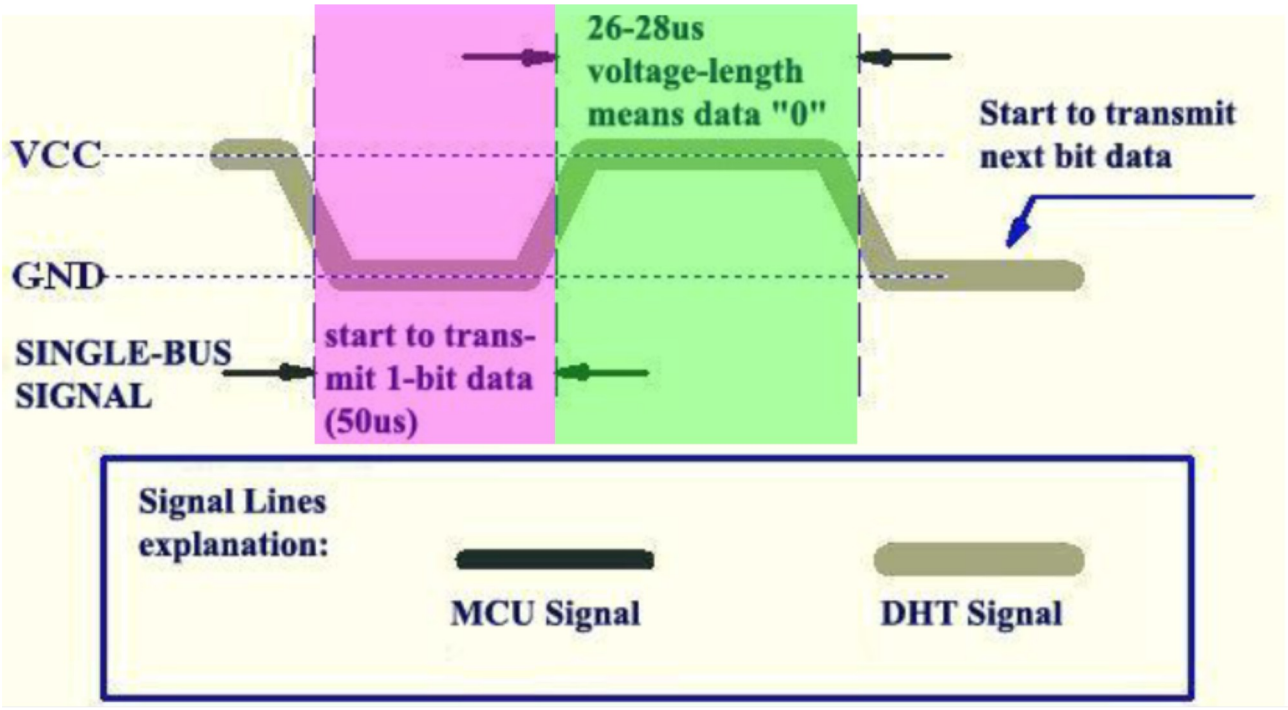
Étape.	Remarque	Durée	De	Vers	État du port
1	After power-up (unsteady state)	1s	MCU	DHT11	LOW
2	Set to OUTPUT	18ms	MCU	DHT11	LOW
3	Set to INPUT Pull up voltage and wait for DHT response.	20 – 40 µs	MCU	DHT11	HIGH
4	DHT sent out response	80µs	DHT11	MCU	LOW
5	DHT pull up voltage	80µs	DHT11	MCU	HIGHT



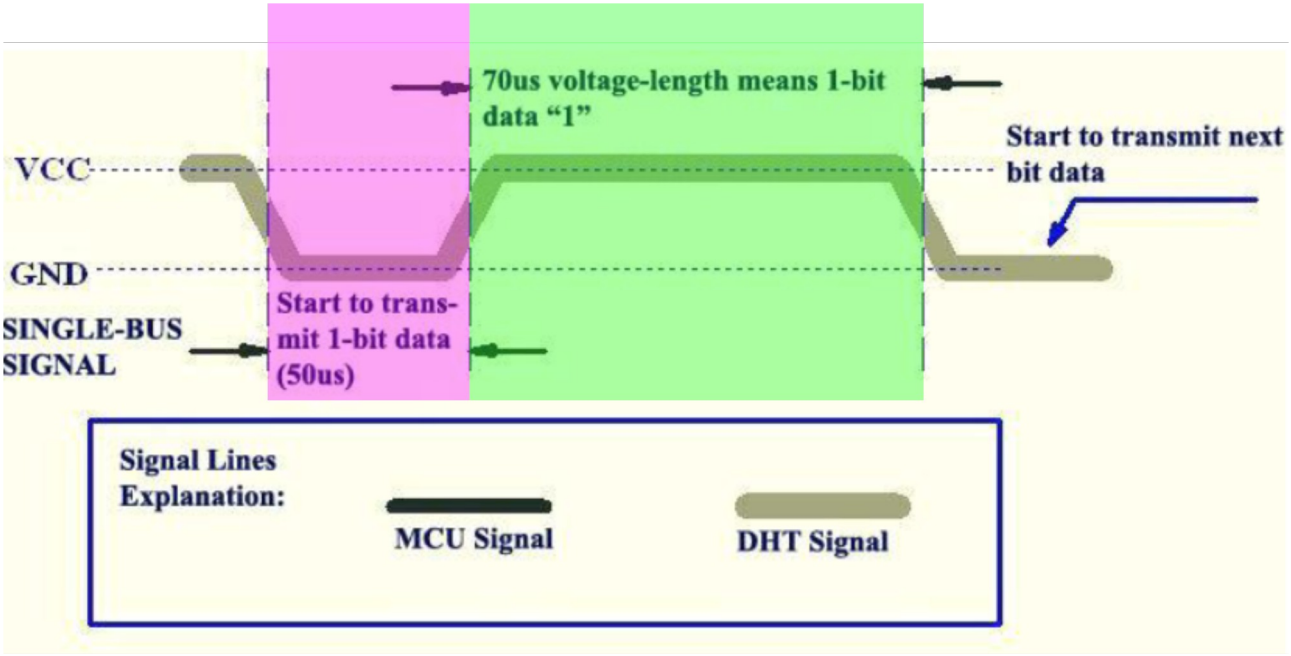
Réception de données depuis le DHT11.

Pour les 40 bits attendus.

Durée à l'état HAUT.	Durée à l'état BAS.	Résultat
50µs	26~28µs	0
50µs	70µs	1



BIT = 0



BIT = 1

MCU directives.

Commande A	État	Commande B	État	Résultat
PxDIR	**0**			Port en entrée.
PxDIR	**1**			Port en sortie
PxSEL	**1**	PxSEL2	**0**	Use as timer & clock