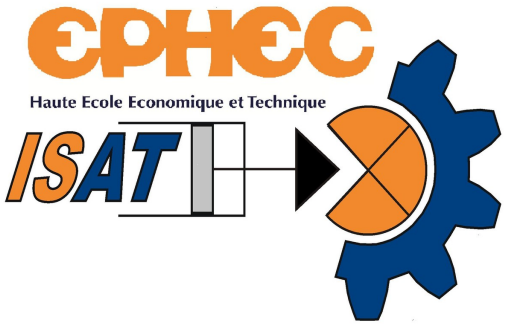


UE A211 : SYSTÈMES EMBARQUÉS I

Professeur : COSTA EMILE

RAPPORT DE LABORATOIRE

Etudiants : KORKUT CANER

TP4 Bis - Gestion des interruptions

Table des matières

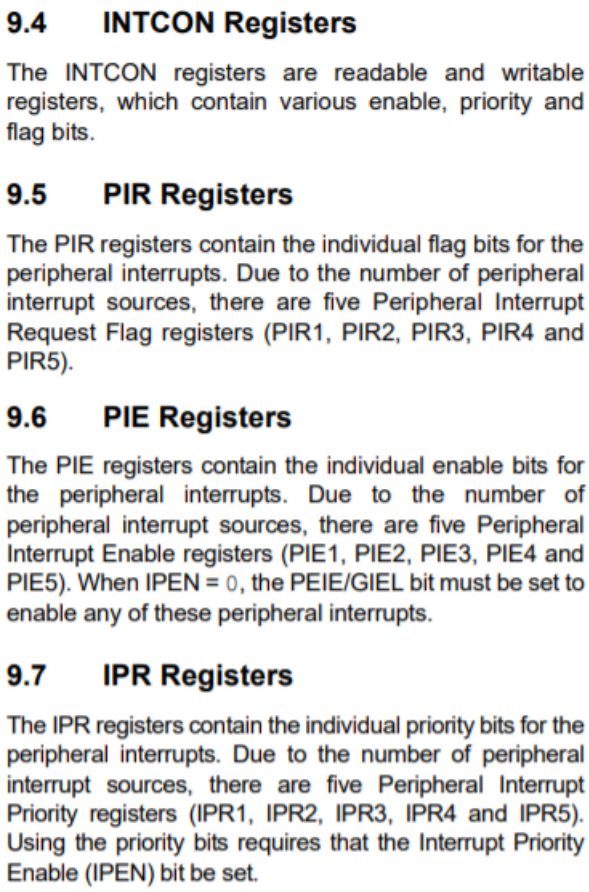
1 Objectif du TP

Cette séance est consacrée à l'utilisation des interruptions dans le cadre de la communication UART. En effet Il a été question d'implémenter la *réception des données* en incluant le mécanisme des interruptions. Pour ce faire, nous sommes repartis du programme effectué au *TP3 - Mise en oeuvre de la communication série (UART)* et avons modifié, entre autres, la partie concernant la réception des données.

2 Introduction

Dans cette section nous reverrons le mécanisme d'interruption dans les détails et aborderons certaines notions comme *vecteur d'interruption*, *sous-routine d'interruption* (ISR) et *requête d'interruption* (IRQ). Pour rappel, il existe de manière générale trois types d'interruption :

- **Logiciel** : généré directement par le programme. Nous avions traité cette problèmatique dans le cas du registre `TIMER0;`
- **Exception** : le processeur peut générer des interruptions s'il n'est pas capable de lire ou d'exécuter une instruction (opcode invalide, division par 0, mémoire protégée, etc).
- **Périphériques** : généré par les périphériques externes connectés au μC. Il s'agit de toute interruption autre que la modification des registres `TMR0`, `INT` ou `PORTB`



9.4 INTCON Registers

The INTCON registers are readable and writable registers, which contain various enable, priority and flag bits.

9.5 PIR Registers

The PIR registers contain the individual flag bits for the peripheral interrupts. Due to the number of peripheral interrupt sources, there are five Peripheral Interrupt Request Flag registers (PIR1, PIR2, PIR3, PIR4 and PIR5).

9.6 PIE Registers

The PIE registers contain the individual enable bits for the peripheral interrupts. Due to the number of peripheral interrupt sources, there are five Peripheral Interrupt Enable registers (PIE1, PIE2, PIE3, PIE4 and PIE5). When IPEN = 0, the PEIE/GIEL bit must be set to enable any of these peripheral interrupts.

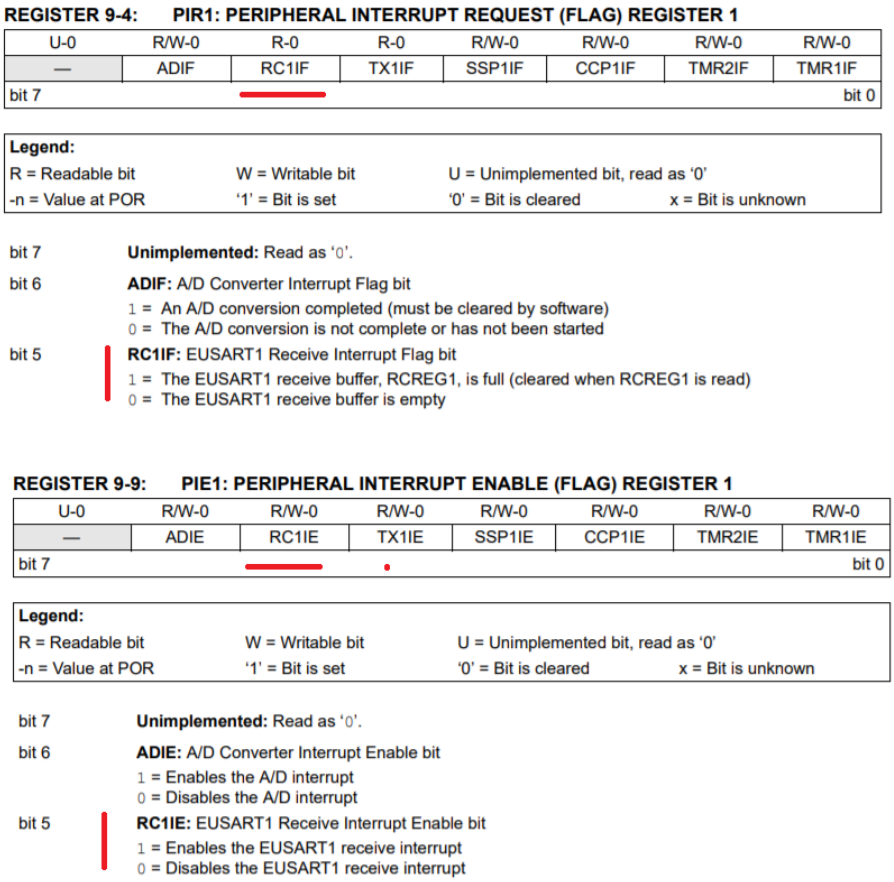
9.7 IPR Registers

The IPR registers contain the individual priority bits for the peripheral interrupts. Due to the number of peripheral interrupt sources, there are five Peripheral Interrupt Priority registers (IPR1, IPR2, IPR3, IPR4 and IPR5). Using the priority bits requires that the Interrupt Priority Enable (IPEN) bit be set.

FIGURE 1 – Extrait de la datasheet du *PIC*18*F*45*K*22, p 108

Dans cette manipulation, il s'agit d'une interruption périphérique liée à l'UART1 [1]. Pour ce faire nous utiliserons les registres `PIR1` [2] et `PIE1` [3]. Comme le montre la figure 1, il s'agit respectivement des registres associés au flag et au bit d'activation d'une interruption périphérique UART. En pratique, nous utiliserons les bits `RC1IE_bit` (ENABLE) et `RC1IF_bit` (FLAG) pour notre bit d'activation et notre flag de contrôle de données lors de la réception de données.

1. Pour rappel le *PIC*18*F*45*K*22 possède 2 interfaces pour la communication UART.
2. PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 1
3. PERIPHERAL INTERRUPT ENABLE (FLAG) REGISTER 1



REGISTER 9-4: PIR1: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 1

U-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
—	ADIF	RC1IF	TX1IF	SSP1IF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **Unimplemented:** Read as '0'.

bit 6 **ADIF:** A/D Converter Interrupt Flag bit
1 = An A/D conversion completed (must be cleared by software)
0 = The A/D conversion is not complete or has not been started

bit 5 **RC1IF:** EUSART1 Receive Interrupt Flag bit
1 = The EUSART1 receive buffer, RCREG1, is full (cleared when RCREG1 is read)
0 = The EUSART1 receive buffer is empty

REGISTER 9-9: PIE1: PERIPHERAL INTERRUPT ENABLE (FLAG) REGISTER 1

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	ADIE	RC1IE	TX1IE	SSP1IE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **Unimplemented:** Read as '0'.

bit 6 **ADIE:** A/D Converter Interrupt Enable bit
1 = Enables the A/D interrupt
0 = Disables the A/D interrupt

bit 5 **RC1IE:** EUSART1 Receive Interrupt Enable bit
1 = Enables the EUSART1 receive interrupt
0 = Disables the EUSART1 receive interrupt

FIGURE 2 – Extrait de la datasheet du *PIC18F45K22*, pp 112 - 118

Nous verrons ces cas en pratique lors de l'analyse du code.

3 Schéma de principe

Le schéma de principe ainsi que la simulation ne diffèrent pas de la séance TP3. Pour rappel, es composants utilisés sont les suivants :

- PIC18F45K22 ;
- 4 Résistances 330Ω ;
- 1 Potentiomètre 1kΩ ;
- 1 LCD 16x2 ;
- 4 Boutons poussoir ;
- 6 Alimentations 5V ;
- 1 Masse ;
- 1 COMPIM (RS232).

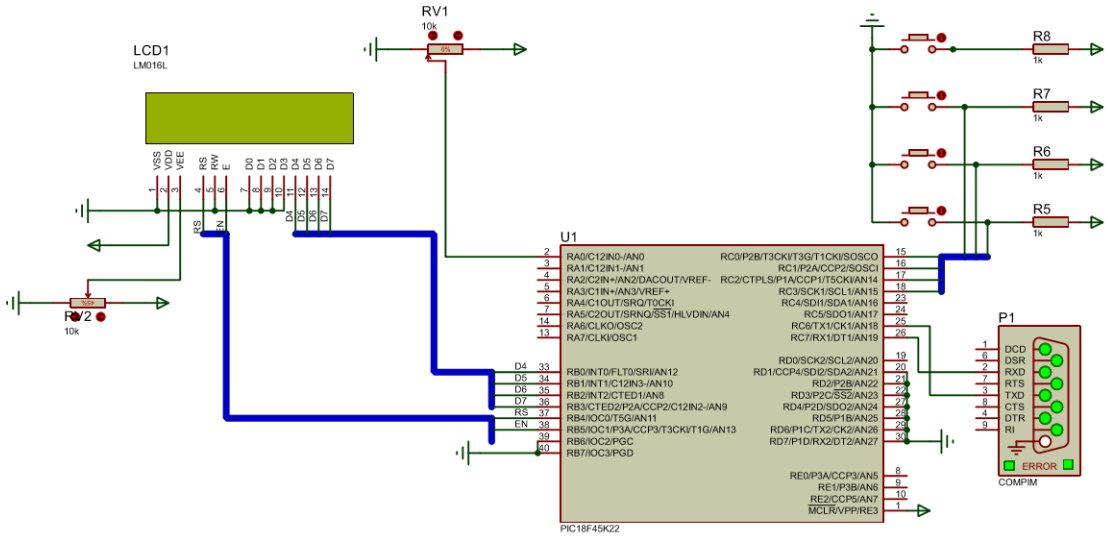


FIGURE 3 – Schéma de principe sur Proteus

4 Algorigramme

Toujours en partant du travail précédent, nous avons restructuré notre algorigramme afin qu'il réponde au programme à coder. Afin de ne pas surcharger le rapport, nous ne présenterons ici que les nouvelles fonctions créées.

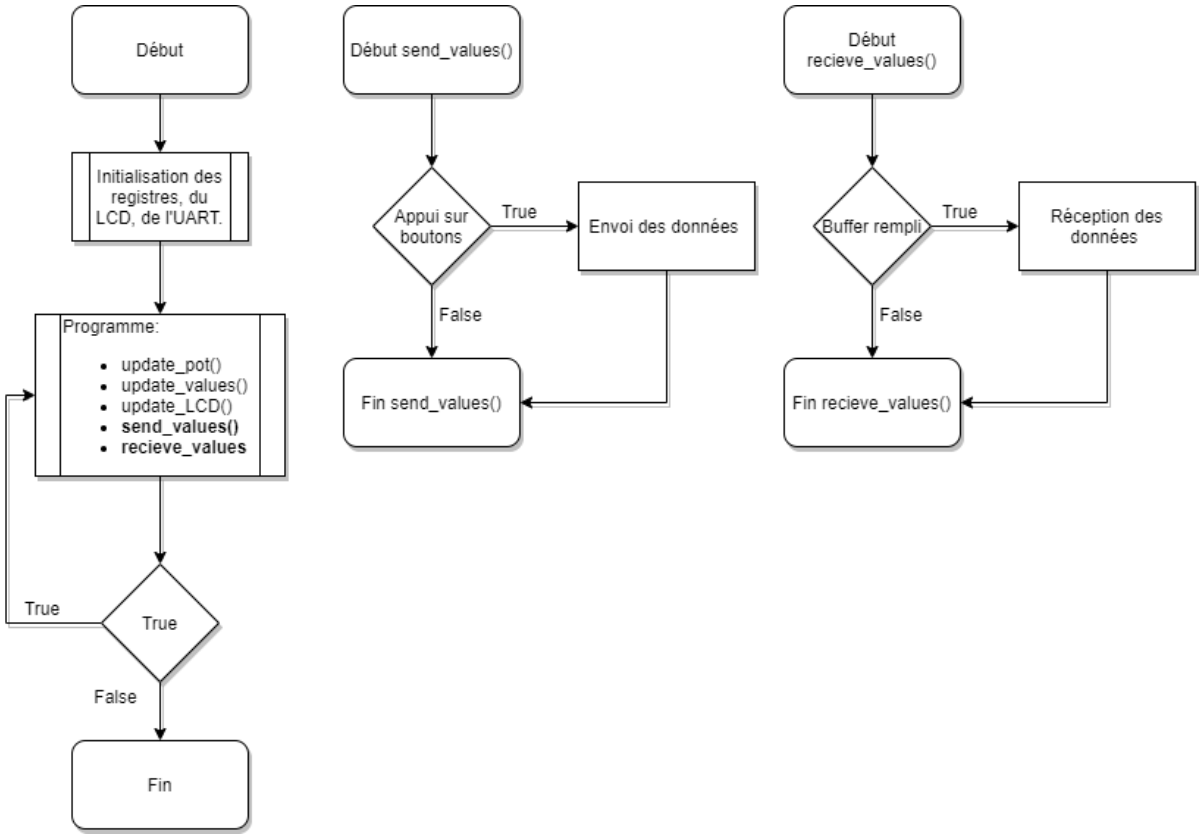


FIGURE 4 – Algorigramme du programme principal

5 Code source

```
#define SOT 2
#define EOT 3
#define LF 10
#define CR 13
#define MAX_CMD_LEN 40

// Connexions du module LCD
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;

sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// Fin des connexions du module LCD

// Declaration des variables globales
int val_1 = 0;
int val_2 = 0;
int index;

bit update_flag;
bit data_recieved_flag;

unsigned int pot_val;
unsigned int old_pot_val;

char content_line_1[16];
char buffer[MAX_CMD_LEN];
char recieved_data;
char command_line[MAX_CMD_LEN];

// Envoi des valeurs
void send_values(){
  if (update_flag){
```

```
42      UART1_Write(SOT);
43      UART1_Write_Text(content_line_1);
44      UART1_Write(EOT);
45      UART1_Write(LF);
46      UART1_Write(CR);
47      update_flag = 0;
48    }
49  }
50
51  // Interrupt
52  void interrupt(){
53     if(RC1IF_bit == 1){
54       recieved_data  = UART_Read();   // Storing read data
55       switch (recieved_data){
56          case '[': // Debut de l'acquisition
57            index = 0;
58            break;
59          case ']': // Fin de l'acquisition
60            command_line[index+1] = '\0';
61            data_recieved_flag = 1;
62            break;
63          default : // Acquisition de la donnee
64            command_line[index++] = recieved_data;
65        }
66     }
67  }
68
69  // Lecture de l'etat des boutons
70  void update_values(){
71      if (Button(&PORTC, 0, 1, 0) && val_1 + 10 <= 255) {
72        val_1 += 10;
73        update_flag = 1;
74        delay_ms(300);
75      }
76
77      if (Button(&PORTC, 1, 1, 0) && val_1 - 10 >= 0) {
78        val_1 -= 10;
79        update_flag = 1;
80        delay_ms(300);
81      }
82
83      if (Button(&PORTC, 2, 1, 0) && val_2 + 10 <= 255) {
84        val_2 += 10;
```

```
85        update_flag = 1;
86        delay_ms(300);
87      }
88
89      if (Button(&PORTC, 3, 1, 0) && val_2 - 10 >= 0) {
90        val_2 -= 10;
91        update_flag = 1;
92        delay_ms(300);
93      }
94  }
95
96  // Lecture de l'etat du potentiometre
97  void update_pot(){
98    old_pot_val = pot_val;
99    pot_val = ADC_Read(0);
100   if (old_pot_val != pot_val){
101     update_flag = 1;
102   }
103 }
104
105 // Met a jour l'ecran du LCD
106 void update_LCD(){
107   if (update_flag){
108     sprintf(content_line_1, "%03u %03u %04u", val_1, val_2, pot_val);
109     Lcd_Out(1, 1, content_line_1);
110     }
111   if (data_recieved_flag){
112     Lcd_Out(2,3, strncpy(buffer,command_line+0,3));
113     Lcd_Out(2,9, strncpy(buffer,command_line+4,3));
114     data_recieved_flag = 0;
115   }
116 }
117
118 // Affichage du message initial sur le Terminal
119 void terminal_init_message(){
120   UART1_Write_Text("Connect  au PIC18F45K22");
121   UART1_Write(LF);
122   UART1_Write(CR);
123   UART1_Write_Text("Bienvenue    mon programme...");
124   UART1_Write(LF);
125   UART1_Write(CR);
126   UART1_Write_Text("Format des donn es    envoyer: [X;Y]");
127   UART1_Write(LF);
```

```
128   UART1_Write(CR);
129   UART1_Write_Text("O  X et Y sont des entiers compris entre 0 et
        999.");
130   UART1_Write(LF);
131   UART1_Write(CR);
132 }
133
134 // Affichage du message initial sur le LCD
135 void LCD_init_message(){
136   Lcd_Cmd(_LCD_CLEAR);
137   Lcd_Cmd(_LCD_CURSOR_OFF);
138   Lcd_Out(1,1,"Initialisation...");
139   Delay_ms(1000);
140   Lcd_Cmd(_LCD_CLEAR);
141   sprintf(content_line_1, "%03u %03u %04u", val_1, val_2, pot_val);
142   Lcd_Out(1, 1, content_line_1 );
143   Lcd_Out(2,1,"A:");
144   Lcd_Out(2,7,"B:");
145 }
146
147 // Fonction d'initialisation
148 void init(){
149   ANSELA  = 0b00000001;
150   ANSELB = 0;
151   ANSELD  = 0;
152   ANSELC = 0;
153
154   update_flag = 0;
155   data_recieved_flag = 0;
156
157   // Desactive les comparateurs
158   C1ON_bit = 0;
159   C2ON_bit = 0;
160
161   // Initialise les entrees et sorties
162   TRISC = 0b10001111;
163   TRISA = 0b00000001;
164
165   /*Initialise les objets lies
166   au bibliotheques utilisees*/
167   ADC_Init();
168
169   UART1_Init(115200);
```

```
170   terminal_init_message();
171
172   Lcd_Init();
173   LCD_init_message();
174
175   // Gestion des interruptions
176   Delay_1sec();
177   RC1IE_bit = 1;  // Active les interruptions de reception sur UART1
178   RC1IF_bit = 0;  // Initialise le flag d'interruption a 0
179   PEIE_bit  = 1;  // Active les interruptions peripheriques
180   GIE_bit   = 1;  // Active les interruptions globales
181 }
182
183 // Fonction principale:
184 void main() {
185       init();
186
187   // Programme:
188   for(;;){
189     update_pot();
190     update_values();
191     update_LCD();
192     send_values();
193   }
194 }
```

6 Analyse du code source

Dans cette section nous nous concentrerons sur la routine d'interruption associée à la réception des données. Avant d'analyser la fonction à proprement dite, voyons quelles sont les nouvelles instructions qui composent la séquence d'initialisation :

```
175 // Gestion des interruptions
176   Delay_1sec();
177   RC1IE_bit = 1;  // Active les interruptions de reception sur UART1
178   RC1IF_bit = 0;  // Initialise le flag d'interruption a 0
179   PEIE_bit  = 1;  // Active les interruptions peripheriques
180   GIE_bit   = 1;  // Active les interruptions globales
```

Comme mentionné dans la figure 2 de l'introduction, `RC1E_bit` permet d'activer les interruptions lors de la réception de données dans le registre associé à l'UART1. `RC1F_bit` est le flag de réception de données. Ce dernier permet à l'exécution du programme de passer à l'adresse du vecteur d'interruption lorsqu'il est activé.

Suite a cette séquence d'initalisation, il n'est plus nécessaire de faire appel à la fonction void `recieve_values()` que nous avions dans notre ancien programme. Cette tâche est dorénavant effectuée par la *sous-routine* d'interruption qui suit :

```
51 // Interrupt
52 void interrupt(){
53    if(RC1IF_bit == 1){
54      recieved_data  = UART_Read();  // Stocke la donnee recue
55      switch (recieved_data){
56         case '[':  // Debut de l'acquisition
57           index = 0;
58           break;
59         case ']':  // Fin de l'acquisition
60           command_line[index+1] = '\0';
61           data_recieved_flag = 1;
62           break;
63         default :  // Acquisition de la donnee
64           command_line[index++] = recieved_data;
65       }
66    }
67 }
```

Après la mise à 1 de `RC1IF_bit` a lieu la désencapsulation des données. Enfin, lorsqu'on rencontre le caractère de fin d'acquisition `]`, on utilise un nouveau flag `data_recieved_flag` afin d'afficher la donnée reçue sur le LCD :

```
106 // Met a jour l'ecran du LCD
107 void update_LCD(){
108   if (update_flag){
109     sprintf(content_line_1, "%03u %03u %04u", val_1, val_2, pot_val);
110     Lcd_Out(1, 1, content_line_1);
111     }
112   if (data_recieved_flag){
113     Lcd_Out(2,3, strncpy(buffer,command_line+0,3));
114     Lcd_Out(2,9, strncpy(buffer,command_line+4,3));
115     data_recieved_flag = 0;
116   }
117 }
```

Le reste du programme a déjà été traité dans le rapport du TP3. Nous constatons tout de même que dans cette dernière fonction, nous levons le flag `data_recieved_flag` mais pas `update_flag` ceci est lié au fait qu'on ne lève un flag que lorsque **toutes les tâches associées à ce flag sont terminées**. Ici, il s'agira de lever ledit flag dans l'appel de fonction `send_values` :

```
39 // Envoi des valeurs
40 void send_values(){
41   if (update_flag){
42     UART1_Write(SOT);
43     UART1_Write_Text(content_line_1);
44     UART1_Write(EOT);
45     UART1_Write(LF);
46     UART1_Write(CR);
47     update_flag = 0;
48   }
49 }
```

7 Conclusion

Lors de cette manipulation, nous avons utilisé pour la première fois les **interruptions périphériques**. Cela nous a non seulement permis de mieux nous familiariser avec le mécanise d'interruptions mais aussi d'être de plus en plus à l'aise avec la lecture de documentations officielles comme la **datasheet**.

Nous constatons également qu'une interruption n'est autre qu'un **dispositif matériel** et demande dès lors un compromis au niveau de l'utilisation des ressources. En effet, bien qu'on ne fasse plus appel au *polling* pour traiter les évênements, nous sollicitons des registres supplémentaires. De nos jours, cela ne pose pas vraiment problème étant donné que nous disposons des technologies permettant d'implémenter des circuits de l'infiniment petit mais on pourrait toujours envisager une situation où cette solution montrera ses limites.

Enfin, alors que cette matière nous semblait floue et incompréhensible au départ, nous constatons maintenant que nous disposons d'une certaine maîtrise sur le sujet. Cela démontre une fois de plus la nécessité de **mettre ses mains dans le cambouis** afin de pousser nos connaissances plus loin.