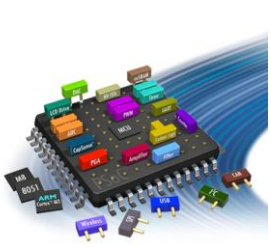
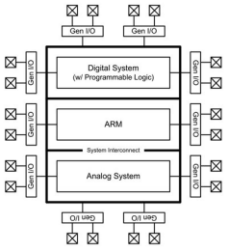
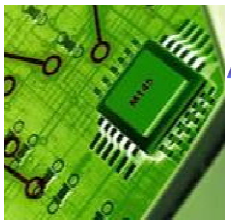
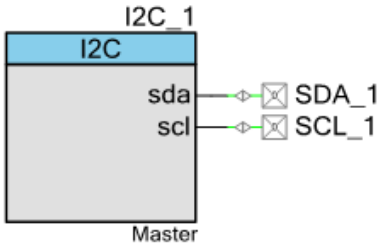






PSoC Creator mise en œuvre du bus I2C Etude du logiciel



Le bus I2C, brevet de la société Philips, permet d'interconnecter des composants électronique entre eux.

I2C signifie : **I**nter **I**ntegrated **C**ircuit

1 Mise en œuvre du bus I2C, avec PSoC Creator principes de bases

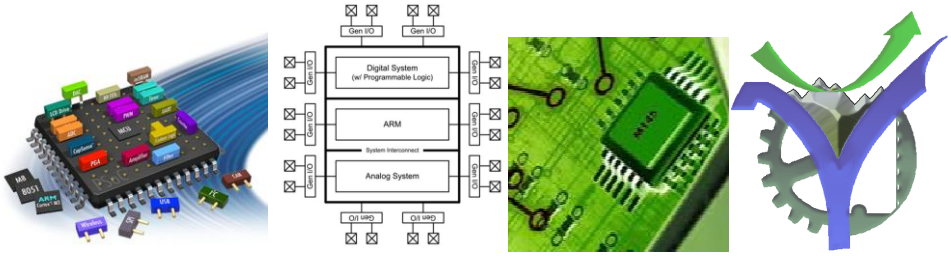
Ce document se propose de guider la mise en œuvre du bus I2C. Ce bus est utilisé pour interfacer un grand nombre de capteurs ou de systèmes électroniques.

Citons par exemple le capteur de température et d'humidité de Honeywell

Les travaux préconisés dans ce document ont pour base de départ le projet :

Test_Toutes_Cartes_I2C_Depart_Eleve

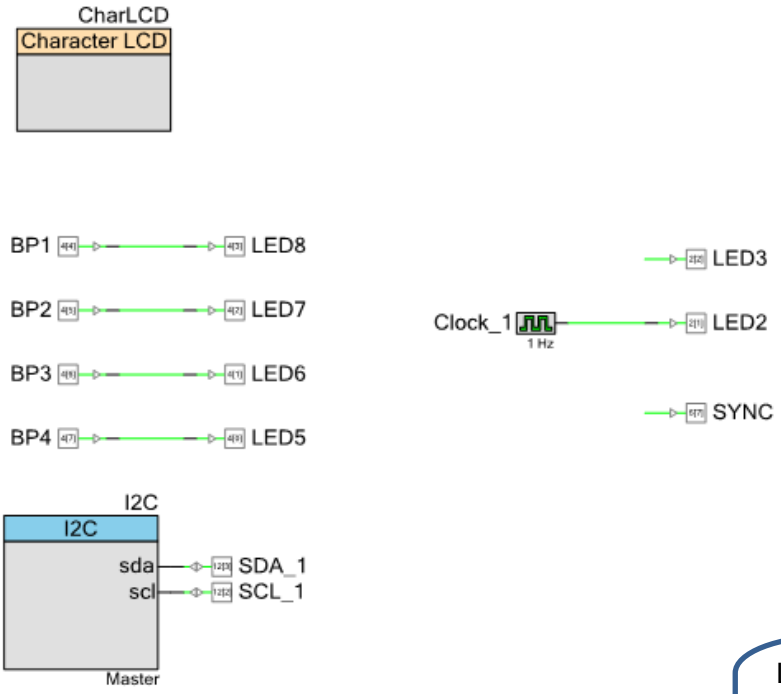
Vous devez recopier le projet dans le répertoire de travail habituel avant de le modifier.



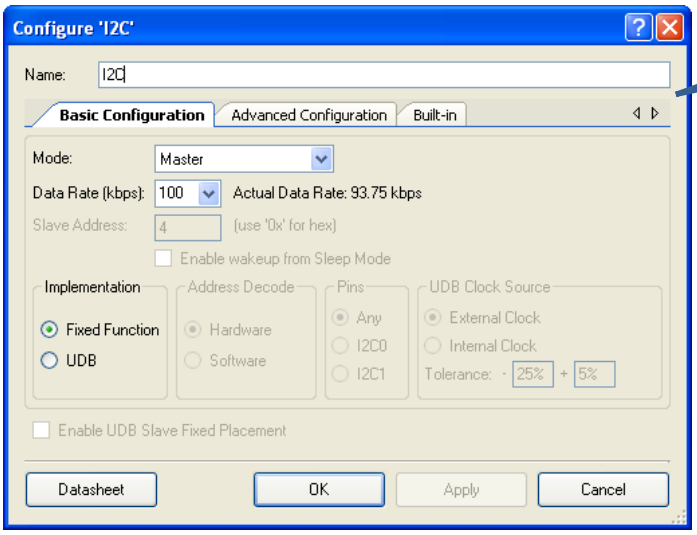
2 Le projet de départ PSoC Creator

☞ Il faut insérer le composant I2C dans le projet et lui assigner les broches pour la platine PSoCVox :

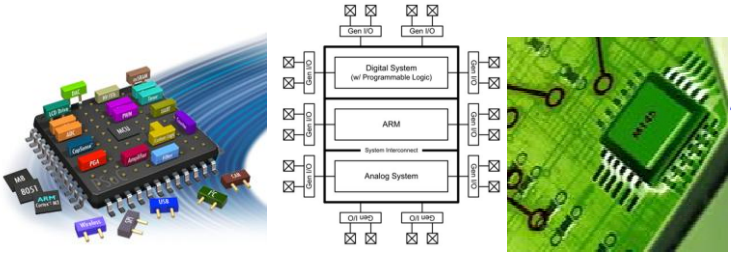
SCL_1	P12[2]
SDA_1	P12[3]



☞ Le composant doit être ensuite configuré

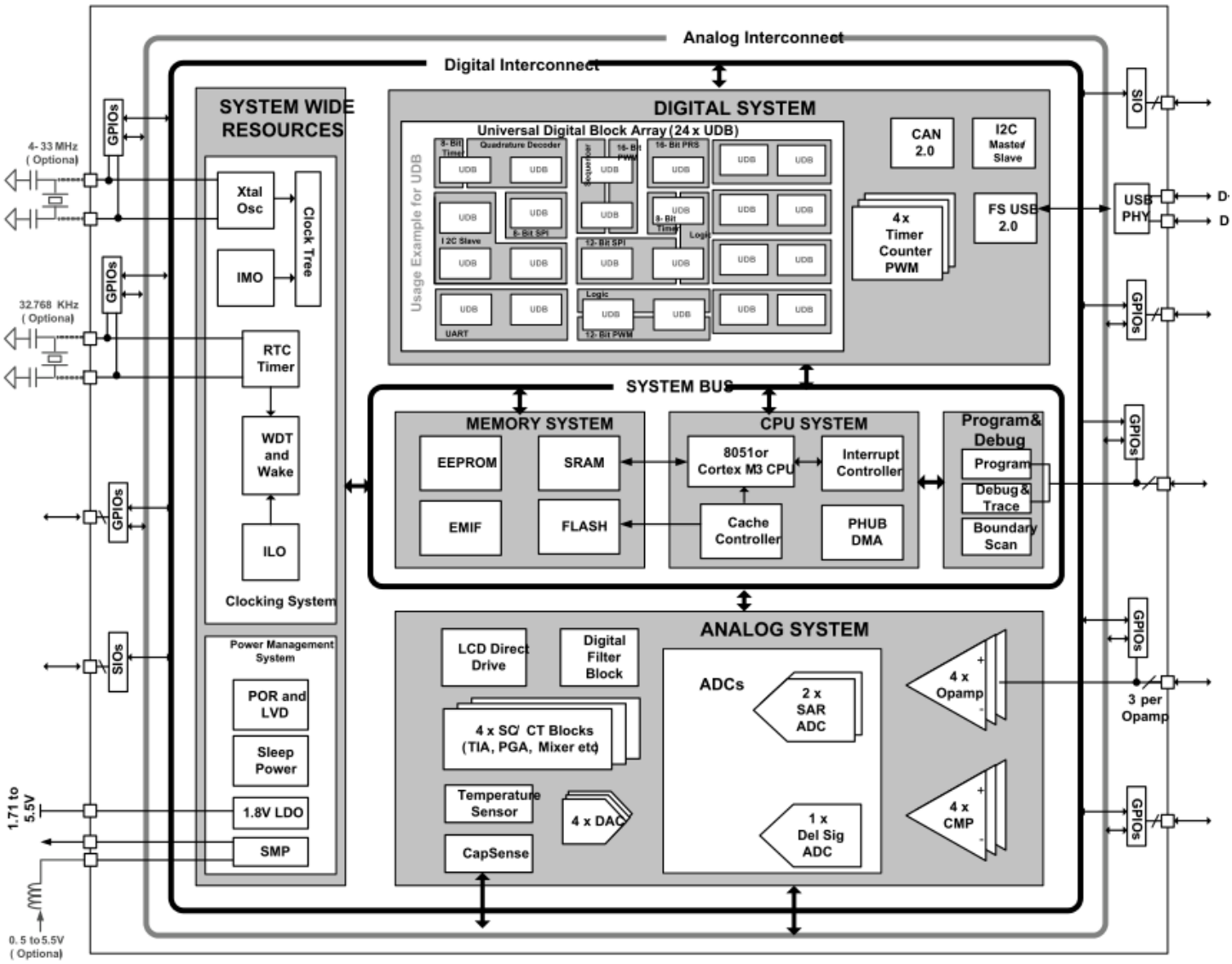


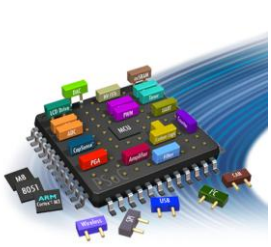
La configuration est très simple dans notre projet il suffit de choisir la fréquence de fonctionnement du Bus ici 100 KHz

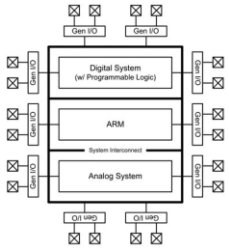


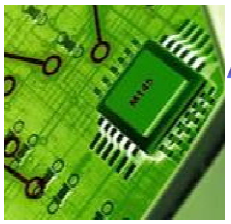
Une vue des différentes parties du PSoC de la famille 5 :

Figure 1-1. Simplified Block Diagram









3 La mise en œuvre logicielle

Les fonctions disponibles de l'API sont dans le datasheet du composant. Nous utilisons essentiellement les deux fonctions ci-dessous :

uint8 I2C_MasterWriteBuf(uint8 slaveAddress, uint8 * wrData, uint8 cnt, uint8 mode)

Description: This function automatically writes an entire buffer of data to a slave device. After the data transfer is initiated by this function, the included ISR manages further data transfer in byte-by-byte mode. Enables the I^2C interrupt.

Parameters: uint8 slaveAddress: Right-justified 7-bit slave address (valid range 0 to 127).

uint8 wrData: Pointer to the buffer of the data to be sent.

uint8 cnt: Number of bytes of the buffer to send.

uint8 mode: Transfer mode defines: (1) Whether a Start or Restart condition is generated at the beginning of the transfer, and (2) Whether the transfer is completed or halted before the Stop condition is generated on the bus.

Transfer mode, mode constants may be ORed together.

Mode Constants	Description
I2C_MODE_COMPLETE_XFER	Perform complete transfer from Start to Stop.
I2C_MODE_REPEAT_START	Send Repeat Start instead of Start.
I2C_MODE_NO_STOP	Execute transfer without a Stop

Return Value: uint8: Error Status. See the I2C_MasterSendStart() function for constants.

Side Effects: None

uint8 I2C_MasterReadBuf(uint8 slaveAddress, uint8 * rdData, uint8 cnt, uint8 mode)

Description: This function automatically reads an entire buffer of data from a slave device. Once this function initiates the data transfer, the included ISR manages further data transfer in byte by byte mode. Enables the I^2C interrupt.

Parameters: uint8 slaveAddress: Right-justified 7-bit slave address (valid range 0 to 127).

uint8 rdData: Pointer to the buffer in which to put the data from the slave.

uint8 cnt: Number of bytes of the buffer to read.

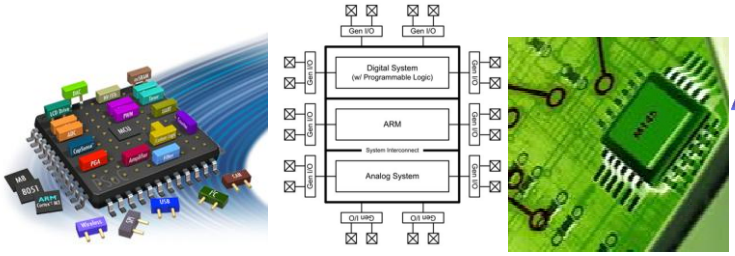
uint8 mode: Transfer mode defines: (1) Whether a Start or Restart condition is generated at the beginning of the transfer and (2) Whether the transfer is completed or halted before the Stop condition is generated on the bus.

Transfer mode, mode constants may be ORed together

Mode Constants	Description
I2C_MODE_COMPLETE_XFER	Perform complete transfer for Start to Stop.
I2C_MODE_REPEAT_START	Send Repeat Start instead of Start.
I2C_MODE_NO_STOP	Execute transfer without a Stop

Return Value: uint8: Error Status. See the I2C_MasterSendStart() function for constants.

Side Effects: None



Pour simplifier la mise en œuvre les fonctions de lecture et écriture les plus courantes ont été réalisées :

Ecriture sur les interfaces parallèles :

```
49  void WriteI2C_PCF8574(uint8 Adresse, uint8 Donnee)
50  {
51      uint8 wbuffer[2];
52      wbuffer[0]=Donnee;
53      I2C_MasterClearStatus();
54      status=I2C_MasterWriteBuf(Adresse,wbuffer,1,I2C_MODE_COMPLETE_XFER);
55      while(Ou == (I2C_MasterStatus() & I2C_MSTAT_WR_CMPLT)){ }
56  }
```

Ecriture sur le convertisseur CNA CAN :

```
58  void WriteI2C_PCF8591(uint8 Adresse, uint8 Configuration, uint8 Donnee)
59  {
60      uint8 wbuffer[2];
61      wbuffer[0]=Configuration;
62      wbuffer[1]=Donnee;
63      I2C_MasterClearStatus();
64      status=I2C_MasterWriteBuf(Adresse,wbuffer,2,I2C_MODE_COMPLETE_XFER);
65      while(Ou == (I2C_MasterStatus() & I2C_MSTAT_WR_CMPLT)){ }
66  }
```

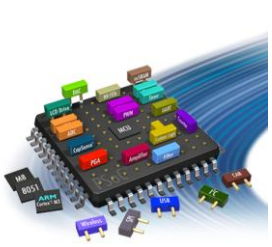
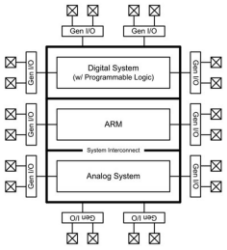
Lecture du convertisseur CNA CAN :

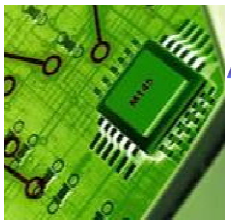
```
68  uint8 ReadI2C_PCF8591(uint8 Adresse)
69  {
70      uint8 rbuffer[2];
71      I2C_MasterReadBuf(Adresse, rbuffer,2, I2C_MODE_COMPLETE_XFER );
72      //wait until Transfer is complete
73      while((I2C_MasterStatus() & I2C_MSTAT_RD_CMPLT )==0);
74      return rbuffer[1];
75  }
```

Détection d'une erreur lors d'un accès au bus

(Réalisé par l'une des fonctions précédentes, permet de détecter la présence d'un esclave ou pas)

```
79  bool Test_Status_I2C(void)
80  {
81      uint8 I2C_Status;
82      // On test le bit global erreur I2C
83      if ((I2C_MasterStatus() & I2C_MSTAT_ERR_XFER)==0) return true; else false;
84  }
```



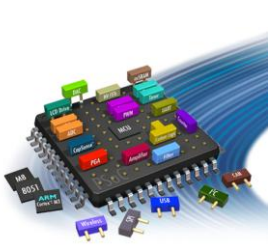
Les déclarations en début de programme :

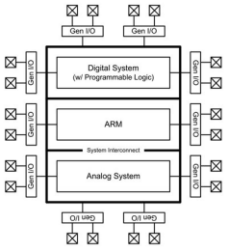
```
13  #include <device.h>
14  #include <stdio.h>
15  #include <stdbool.h>
16
17  // Adressage des cartes esclaves I2C utilisées
18  #define Adresse_LM75     0x4F
19  #define Adresse_PCF8574  0x20
20  #define Adresse_PCF8574A 0x38
21  #define Adresse_PCF8591  0x48
22
23
24  // Déclarations des flages présence cartes slaves
25  bool PCF8574_Present=false;
26  bool PCF8574A_Present=false;
27  bool PCF8591_Present=false;
28  bool LM75_Present=false;
29
30
31  // Configuration du convertisseur PCF8591
32  #define CONF_PCF8591 0b01000000
```

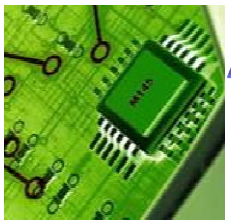
4 Premiers travaux

Les fonctions décrites à la page précédente ne sont pas documentées, pour la première voilà ce qu'il est possible de faire :

```
55  //=========================================================================
56  // WriteI2C_PCF8574
57  //=========================================================================
58  // Fonction qui écrit dans les composants de type PCF8574 ou PCF8574A
59  // Interface paralléle sur le bus I2C
60  // Paramètres :
61  //      Adresse du périphérique sur 7 bits : Adresse uint8
62  //      Data à écrire sur le périphérique  : Donnee  uint8
63  // Retour :
64  //      Aucun
65  //=========================================================================
66  void WriteI2C_PCF8574(uint8 Adresse, uint8 Donnee)
67  {
68      uint8 wbuffer[2];
69      wbuffer[0]=Donnee;
70      I2C_MasterClearStatus();
71      status=I2C_MasterWriteBuf(Adresse,wbuffer,1,I2C_MODE_COMPLETE_XFER);
72      while(0u == (I2C_MasterStatus() & I2C_MSTAT_WR_CMPLT)){ }
73  }
```



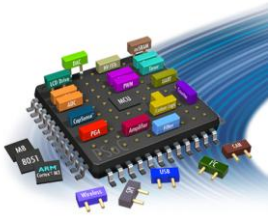


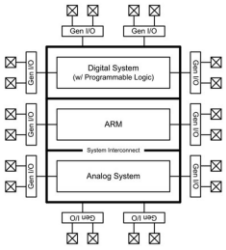


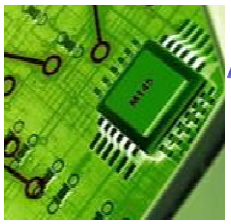
Documenter le code

La documentation est primordiale et ne doit en aucun cas être négligée. La maintenance d'un code est une opération fréquente et doit pouvoir être réalisée par l'auteur du programme, plusieurs mois ou années après son écriture initiale, mais elle doit pouvoir être réalisée par un autre intervenant qui n'aura alors qu'une vision plus restreinte du code en question.

☞ Documenter de la même manière les autres fonctions de la page 5 dans votre projet.





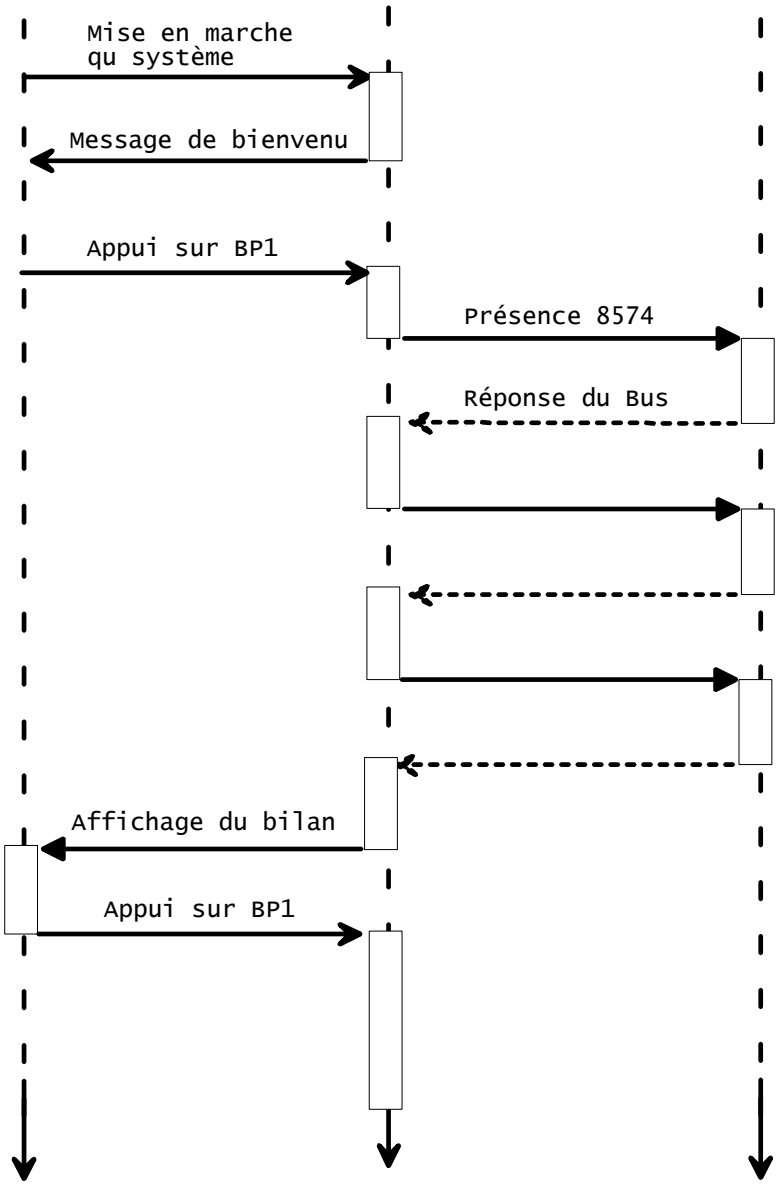


Formation PSoC

Enumération des esclaves présents (Sysml)

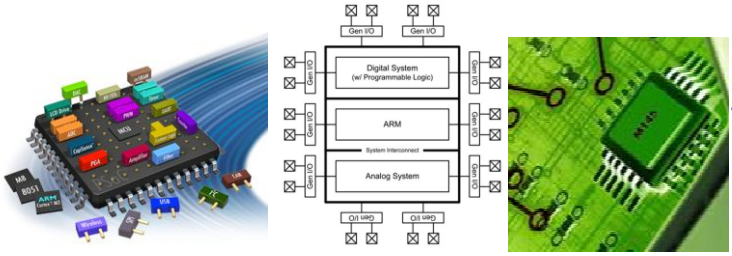
Diagramme de séquence

Enumération des périphériques I2C



☞ Compléter ce diagramme en ajoutant les commentaires des flèches existantes.

☞Indiquer les numéros de lignes de programmes concernées par chaque activité.



Code du programme :

```
126
127     // Affichage du message de bienvenu
128     // Attente d'un appui sur BP1 pour continuer
129     CharLCD_Position(0,0);
130     CharLCD_PrintString("Hello  World");
131     CharLCD_Position(3,0);
132     CharLCD_PrintString("Start => Appui BP1");
133     while (BP1_Read() == REPOS ) {};
134     CharLCD_ClearDisplay();
135     while (BP1_Read() == APPUYE ) {};
136
137     // lecture du statut de la liaison I2C
138     // vérification de la présence du PCF8574
139     WriteI2C_PCF8574(Adresse_PCF8574,~0x00);
140     PCF8574_Present=Test_Status_I2C;
141
142     // lecture du statut de la liaison I2C
143     // vérification de la présence du PCF8574A
144     WriteI2C_PCF8574(Adresse_PCF8574A,~0x00);
145     PCF8574A_Present=Test_Status_I2C;
146
147     // Initialisation du PCF8591
148     // lecture du statut de la liaison I2C
149     WriteI2C_PCF8591(Adresse_PCF8591, CONF_PCF8591, 0);
150     PCF8591_Present=Test_Status_I2C;
151
152     // Affichage du message d'accueil
153     // Carte PCF8574
154     CharLCD_Position(0,0);
155     if (PCF8574_Present)
156         CharLCD_PrintString("PCF8574  => ACK");
157     else
158         CharLCD_PrintString("PCF8574  => NACK");
159
```

```
160     // Carte PCF8574A
161     CharLCD_Position(1,0);
162     if (PCF8574A_Present)
163         CharLCD_PrintString("PCF8574A => ACK");
164     else
165         CharLCD_PrintString("PCF8574A => NACK");
166
167     // Carte PCF8591
168     CharLCD_Position(2,0);
169     if (PCF8591_Present)
170         CharLCD_PrintString("PCF8591  => ACK");
171     else
172         CharLCD_PrintString("PCF8591  => NACK");
173
174     // Appui sur BP1 pour continuer
175     CharLCD_Position(3,0);
176     CharLCD_PrintString("Start => Appui BP1");
177     while (BP1_Read() == REPOS ) {};
178
```