Mise en place d'un IoT avec Arduino

I. Mise en place de l'environnement de développement Arduino

A. Carte Arduino Genuino 101

Les travaux pratiques se dérouleront sur une carte de développement Arduino Genuino 101.

La carte Genuino 101 est basée sur un microcontrôleur Intel Curie cadencé 32 MHz disposant de deux cœurs, un X86 et un 32 Bits ARC. Cette carte comporte en plus par rapport à la carte Arduino Uno classique un gyroscope, un accéléromètre, un module Bluetooth Low Energy et un microcontrôleur plus performant avec plus d'espace disponible pour le stockage des programmes.

Des connecteurs situés sur les bords extérieurs du circuit imprimé permettent d'enficher une série de modules complémentaires de la même façon que les cartes Arduino/Genuino Uno.

B. Installation de l'environnement Arduino / librairie Intel Curie

Afin de programmer cette carte nous allons installer l'environnement de développement Arduino en mode portable avec la librairie pour les cartes basées sur un module Intel Curie.

Manipulation : installation

1. Installer l'environnement de développement Arduino en mode portable. Sous windows prendre la version de l'IDE compressée .zip dans l'onglet **Ressources** ;
2. Installer la librairie pour les modules Intel Curie.

Remarque : Les développements entre groupes ne doivent pas interférer les uns avec les autres. Pour cette raison l'IDE Arduino doit être rendue portable (elle est propre à chaque utilisateur). Après décompression de votre archive de l'IDE Arduino assurez-vous de l'avoir rendue portable.

II. Mise en place d'une application sur la carte

A. Application simple

Bien choisir la carte Genuino 101 dans la partie Outils → Type de cartes : Intel Curie. Si l'erreur Avr Dude apparait, vérifier le choix de la carte. Bien définir le port de connexion de la carte.

Manipulation : Compilation et exécution du code suivant

```
void setup()
{
	pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
```

```
        static bool state=false;
        digitalWrite(LED_BUILTIN,state);
        state=!state;
        delay(100);
}
```

Dans un programme Arduino il y a plusieurs zones. Il existe la fonction setup() qui permet d'initialiser certains paramètres du programme (entrées/sorties du microcontrôleur) et la fonction loop() où se déroulera le programme.

B. Les entrées-sorties numériques

1. Fonction pinMode(): pinMode(broche, mode)

La fonction pinMode() permet de configurer une broche en sortie ou en entrée (haute impédance). La fonction pinMode() accepte deux paramètres obligatoires : le numéro de broche à configurer en premier paramètre et le mode de fonctionnement de la broche en second paramètre.Le mode de fonctionnement de la broche peut prendre les valeurs suivantes : OUTPUT (sortie), INPUT (entrée) ou INPUT_PULLUP (entrée avec résistance de tirage).

2. Fonction digitalWrite(): digitalWrite(broche, etat)

La fonction digitalWrite() permet de choisir l'état d'une broche. La fonction digitalWrite() accepte deux paramètres obligatoires : le numéro de broche à configurer en premier paramètre et l'état de la broche en second paramètre. L'état de la broche peut prendre les valeurs suivantes : HIGH (état haut) ou LOW (état bas).

3. Fonction digitalRead(): digitalRead(broche)

La fonction digitalRead() permet de "lire" l'état d'une broche. La fonction digitalRead() accepte un paramètre obligatoire : le numéro de la broche à lire. La fonction digitalRead() retourne une valeur pouvant être : HIGH (état haut) ou LOW (état bas), en fonction de l'état physique de la broche.

Exercice : Allumer les DELs à l'aide des boutons-poussoirs

Compléter le code suivant pour qu'en appuyant sur le bouton 3 la DEL D5 s'allume pendant 3 secondes et puis s'éteint et la même chose pour le couple bouton 2 et la DEL D6.

```
// définition des différentes connexions
// GPIO General Purpose Input/Output

const int led_D5 = 13;
const int led_D6 = 12;
const int bouton_S2 = 8;
const int bouton_S3 = 9;
```

```
// fonction d'initialisation de la carte
void setup()
{
	// TODO
}
// fonction principale, elle se répète (s'exécute) à l'infini
void loop()
{
	// TODO
}
```

Solution de l'exercice : Compilation et exécution du code suivant

```
// définition des différentes connexions
// GPIO General Purpose Input/Output
const int led_D5 = 13;
const int led_D6 = 12;
const int bouton_S2 = 8;
const int bouton_S3 = 9;
int etat_S2, etat_S3;

// fonction d'initialisation de la carte
void setup()
{
	// initialisation des broches 12 et 13 comme étant des sorties
	pinMode(led_D5, OUTPUT);
	pinMode(led_D6, OUTPUT);
	// initialisation des broches 8 et 9 comme étant des entrées
	pinMode(bouton_S2, INPUT);
	pinMode(bouton_S3, INPUT);
}

// fonction principale, elle se répète (s'exécute) à l'infini
void loop()
{
	etat_S2 = digitalRead(bouton_S2);
	etat_S3 = digitalRead(bouton_S3);
	if(etat_S2 == LOW)
	{
		digitalWrite(led_D6, HIGH);
		delay(3000);
		digitalWrite(led_D6, LOW);
	}
	if(etat_S3 == LOW)
	{
		digitalWrite(led_D5, HIGH);
		delay(3000);
		digitalWrite(led_D5, LOW);
	}
}
```

Dans cet exemple, nous manipulons des entrées/sorties numériques qui peuvent être ou mesurer un état haut ('1' <=> + 5 ou 3,3 V) ou un état bas ('0' <=> 0 V).

C. Les entrées - "sorties" analogiques

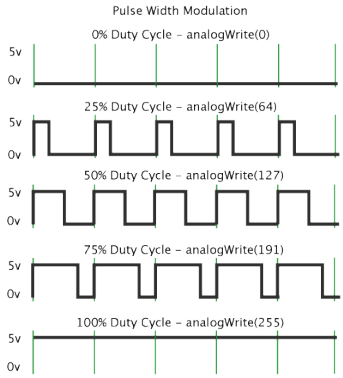
Il est possible de gérer des entrées analogiques (présence de convertisseur analogique numérique). Néanmoins il n'existe pas de sortie analogique à proprement parlé, il faut utiliser les sorties PWM.

1. Fonction analogRead(): int analogRead(broche)

La fonction analogRead() permet de mesurer une tension sur une broche analogique. La fonction analogRead() accepte un paramètre obligatoire : le numéro de broche analogique à lire. La fonction analogRead() retourne un nombre entier (int) compris entre 0 et 1023 car le CAN fonctionne sur 10 bits. Ce nombre correspondant à la tension mesurée, 0 = 0 volt, 1023 = tension alimentation = 5 volts (ou 3v3 suivant les cartes Arduino).

2. Les sorties PWM

Le signal PWM (Pulse Width Modulation) ou MLI (Modulation à largeur d'impulsion) est un signal carré de fréquence fixe mais de rapport cyclique variable. De ce fait la valeur moyenne du signal est réglable et la présence d'un filtre-bas en aval de la sortie permet d'avoir une sortie analogique. Dans l'exemple de la commande d'un moteur, la constante de temps mécanique du système est suffisante pour réaliser ce filtrage, il n'y a donc pas de composants supplémentaires à rajouter. Afin de générer ce signal il faut utiliser la fonction analogWrite.



3. Fonction analogWrite(): analogWrite(broche, valeur)

La fonction analogWrite() permet de générer un signal PWM. La fonction analogWrite accepte deux paramètres. Le premier est le numéro de la broche utilisée pour "écrire" l'impulsion. Cette broche devra être une broche ayant la fonction PWM. Le deuxième la valeur du rapport cyclique (proportion de l'onde carrée qui est au niveau HAUT) : entre 0 (0% HAUT donc toujours au niveau BAS) et 255 (100% HAUT donc toujours au niveau HAUT).

D. Gestion des interruptions

En informatique, une interruption est une suspension temporaire de l'exécution d'un programme informatique par le microcontrôleur afin d'exécuter un programme prioritaire (appelé service d'interruption). Les interruptions sont provoquées par exemple par des changements d'état d'une entrée numérique.

Manipulation : Compilation et exécution du code

```
// définition des différentes connexions
// GPIO General Purpose Input/Output
const int led_D5 = 13;
const int led_D6 = 12;
const int bouton_S2 = 8;
const int interrupt_bouton_S3 = 9;
int etat_S2, etat_S3;

// fonction d'initialisation de la carte
void setup()
{
	// initialisation des broches 12 et 13 comme étant des sorties
	pinMode(led_D5, OUTPUT);
	pinMode(led_D6, OUTPUT);
	// initialisation des broches 8 et 9 comme étant des entrées
	pinMode(bouton_S2, INPUT);
	pinMode(interrupt_bouton_S3, INPUT);
	// la broche 9 va générer une interruption lorsqu'elle sera à l'état BAS
	attachInterrupt(digitalPinToInterrupt(interrupt_bouton_S3), reagir_S3_Bas, LOW);
}
// fonction principale, elle se répète (s'exécute) à l'infini
void loop()
{
	etat_S2 = digitalRead(bouton_S2);
	if(etat_S2 == LOW)
	{
		digitalWrite(led_D6, HIGH);
		delay(3000);
		digitalWrite(led_D6, LOW);
	}
}

void reagir_S3_Bas()
{
	digitalWrite(led_D5, HIGH);
	delay(3000);
	digitalWrite(led_D5, LOW);
}
```

III. Module ESP8266 / Commandes AT

A. Module ESP8266

L'ESP8266 est un module Wifi très bon marché (de l'ordre de 2€). Il permet à l'origine de s'interfacer en série avec d'autres périphériques et donc de faire de l'UART vers Wifi. Son faible coût et ses capacités croissantes (firmwares alternatifs, nouvelles versions hardware) ont font un candidat sérieux pour tous les "objets connectés".

Il existe différentes façons de programmer l'ESP. En effet, différents firmware ont été développé. Une première façon de le programmer (pas abordé dans ses TP) est l'utilisation du firmware Arduino. La communauté Arduino s'est mise à développer un compilateur et des librairies (notamment ESP8266Wifi.h) permettant d'utiliser l'ESP comme un Arduino classique. La compilation du code crée un firmware qui sera installé sur la mémoire flash externe du module.

Le firmware original «AT» est celui d'origine et installé par Expressif (la société ayant développé l'ESP). Cela permet d'utiliser l'ESP comme simple module Wifi à l'aide des commandes AT.

B. Commandes AT

Le module ESP8266 utilise des commandes AT pour les communications avec l'Arduino, voici la liste des commandes principales :

1. Connexion a un point d'accès WIFI

AT	Ne fait rien, à part renvoyer "OK"
AT+GMR	Retourne la version du firmware
AT+RST	Redémarre le module
AT+CWMODE	Permet de choisir le mode Wi-Fi à utiliser : AT+CWMODE=1 : mode station (utiliser ce mode pour se connecter à un réseau Wi-Fi existant) AT+CWMODE=2 : mode AP (Access Point: utiliser ce mode pour créer un nouveau réseau Wi-Fi) AT+CWMODE=3 : les deux
AT+CWLAP	Retourne la liste des réseaux Wi-Fi à portée
AT+CWJAP	Connection à un réseau Wi-Fi. Cette commande prend deux paramètres : le SSID du réseau, et le mot de passe. Par exemple : AT+CWJAP="Livebox-1234","MonMotDePasseWifi"
AT+CIFSR	Une fois le module connecté à un réseau, cette commande permet de récupérer son adresse IP

2. TCP Client

AT+CIPSTART	Initialise une connexion.
AT+CIPMUX	Permet de choisir si on veut pouvoir gérer plusieurs connexions simultanément ou pas : AT+CIPMUX=0 : une seule connexion AT+CIPMUX=1 : plusieurs connexions
AT+CIPSTART	Initialise une connexion. Deux cas, en fonction de ce que vous avez choisi pour AP+CIPMUX : Si CIPMUX=0: AT+CIPSTART prend 3 paramètres: le type (TCP ou UDP), l'IP, et le port. Par exemple : AT+CIPSTART="TCP",192.168.0.5,80 Si CIPMUX=1: AT+CIPSTART a un paramètre supplémentaire qui est le numéro de la connexion à utiliser (entre 0 et 4). On a donc : le numéro de la connexion, le type (TCP ou UDP), l'IP, et le port. Par exemple : AT+CIPSTART=1,"TCP",192.168.0.5,80

AT+CIPSEND	Démarre l'envoie de données sur une connexion. Ici aussi, deux cas, en fonction de ce que vous avez choisi pour AP+CIPMUX: Si CIPMUX=0 : AT+CIPSTART prend un seul paramètre: le nombre de caractères à envoyer. Par exemple : AT+CIPSEND=10 Si CIPMUX=1 : AT+CIPSTART a un paramètre supplémentaire qui est le numéro de la connexion à utiliser (entre 0 et 4). On a donc : le numéro de la connexion, et le nombre de caractères. Par exemple : AT+CIPSEND=1,10 Pour envoyer le message à proprement parler, on écrira sur la liaison série à la suite de cette commande. Par exemple : AT+CIPSEND=18 GET / HTTP/1.0\r\n\r\n
AT+CIPCLOSE	Ferme une connexion. Si CIPMUX=0, ne prend pas de paramètre. Si CIPMUX=1, prend en paramètre le numéro de connexion à fermer. Par exemple : AP+CIPCLOSE=1

3. TCP Serveur

AT+CIPSERVER	Configure en mode serveur TCP. AT+CIPSERVER=mode[,port]. Si mode = 1, cela crée le serveur, si mode = 0, cela le supprime et le port est par défaut à 333 Par exemple : AT+CIPSERVER=1,9999 : configure en mode serveur TCP sur le port 9999, (1 signifie Autoriser)
AT+CIFSR	Vérifie l'adresse IP du module

C. ESP8266 & Arduino

Manipulation : Faisons un premier test. Uploadez le programme suivant, puis ouvrez le moniteur série de l'IDE Arduino en activant les deux CR et NL.

```
void setup() {
	Serial.begin(9600);
	//Configurer les fins de lignes dans la console série
	// Activer les deux CR et NL
	while(!Serial);
	Serial1.begin(9600);
	while(!Serial1);
	pinMode(13,OUTPUT);
	pinMode(12,OUTPUT);
}

void loop() {
	// put your main code here, to run repeatedly:
}

void serialEvent(){
	Serial1.write(Serial.read());
	digitalWrite(12,!digitalRead(12));
}

void serialEvent1(){
```

```
        Serial.write(Serial1.read());
        digitalWrite(13,!digitalRead(13));
}
```

Dans la zone de saisie, entrez la commande suivante : AT
Vous devriez avoir la réponse suivante : OK

Cette commande "AT" permet de vérifier que nous arrivons à communiquer avec le module.

Dans la zone de saisie, entrez la commande suivante : AT+GMR
Vous devriez obtenir une réponse proche de :
AT version:1.3.0.0(Jul 14 2016 18:54:01)
SDK version:2.0.0(656edbf)
compile time:Jul 19 2016 18:44:22

Ce "AT+GMR" renvoie la version du firmware installé sur le module.

Avec ce programme, l'Arduino va servir de relais entre le moniteur série de notre IDE et l'ESP8266 (mode pass through). Tout ce qui est saisi dans le moniteur série sera transmis au module Wi-Fi, et toutes les réponses du module Wi-Fi seront affichées dans le moniteur série. Vous noterez qu'on utilise ici 2 liaisons série : une pour la communication moniteur série - Arduino (Serial), et une autre pour la communication Arduino - ESP8266 (Serial1).