



Conception des objets connectés

Dr. Eng. Chiheb Ameer ABID

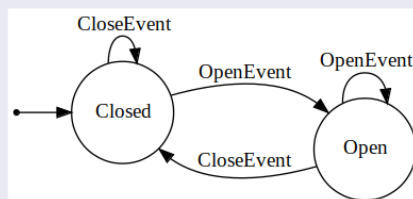
[in /in/chiheb-ameur-abid](#)
[✉ chiheb.abid@gmail.com](mailto:chiheb.abid@gmail.com)

Les machines à états

Les machines à états

↳ Un automate fini ou machine à états finis (finite state machine) est un modèle mathématique pour la modélisation de comportement de systèmes numériques

- ☛ Son rôle, est de décrire le fonctionnement d'une machine (ou d'un objet) ayant un comportement **séquentiel**
- ☛ Modèle graphique
- ☛ Utilisé dans de nombreux domaines : conception de programmes informatiques, protocoles de communication, contrôle des processus, analyse linguistique, etc.



Plan

- 1 Programmation avec les machines à états
- 2 Fonctions de communication
- 3 Communication sur un bus I2C (Inter Integrated Circuit)

Les machines à états

Les machines à états

↳ Dans un système séquentiel, le comportement d'une machine (ou d'un objet) dépend non seulement des événements qu'il reçoit mais aussi de ce qui s'est passé avant ces événements.

- ☛ Par exemple, on ne peut pas faire descendre un ascenseur s'il est déjà en bas
- ☛ Les machines ayant un fonctionnement séquentiel passent par un nombre de situations (états) limitées et clairement identifiées. Nous disons alors que ce sont des machines à états **finis**.

Les machines à états

Les machines à états

- Une machine à états est construite de quatre éléments de base
 - Des états
 - Des événements
 - Des transitions
 - Des actions

États

- Un état est une situation stable qui possède une certaine durée pendant laquelle un objet exécute une activité ou attend un événement
- Un état représenté par un cercle dans lequel le nom de l'état est inscrit



Les machines à états

Transitions

- Une transition définit la réponse d'un objet à l'arrivée d'un événement. Elle indique qu'un objet qui se trouve dans un état peut « transiter » vers un autre état en exécutant éventuellement certaines activités
- Une transition est représentée par une flèche allant de l'état de départ vers l'état d'arrivée

Évènements

- Un événement est un fait qui déclenche le changement d'état, qui fait donc passer un objet d'un état à un autre état
- Un événement se produit à un instant précis et est dépourvu de durée. Quand un événement est reçu, une transition peut être déclenchée et faire basculer l'objet dans un nouvel état.
- Un événement est représenté par une expression inscrite sur une transition

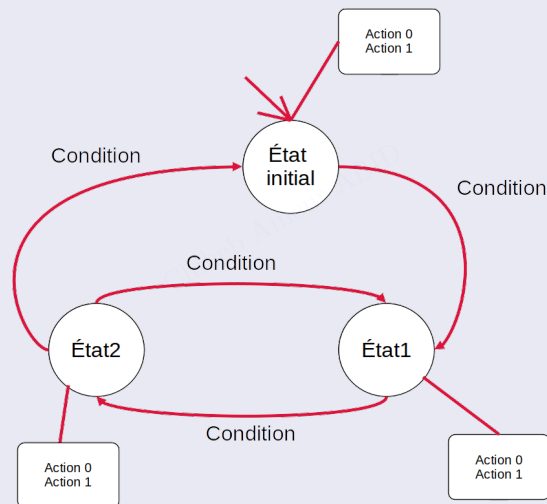
Actions

- Une action consiste à envoyer un signal, à faire appel à une méthode, à affecter une valeur à un attribut, etc.



Les machines à états

Représentation graphique d'une machine à états



Traduction d'une machine à états en C++

- Définir un type énumératif

```
enum class State {etatInitial, etat1, etat2, ...};
```
- Définir une variable globale pour mémoriser l'état courant

```
State etat {State::etatInitial};
```
- L'implémentation de la machine états se fait par un `switch` en créant un `case` pour chaque état

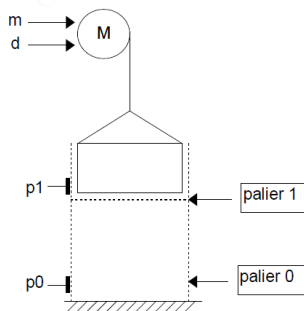
```
1 switch(etat) {
2   case State::etatInitial: // Actions de etatInitial
3     if (cond) { // Condition de transition
4       // Actions juste avant la transition
5       etat=etatSuivant;
6     }
7     break;
8   case State::etat0: ...
9 }
```



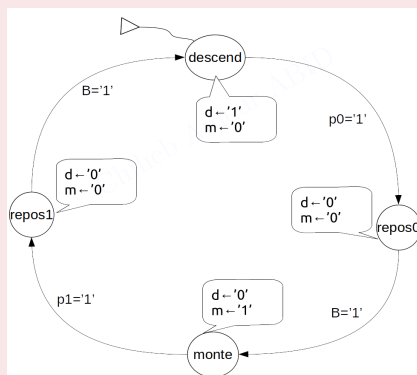
Application

On se propose de réaliser la commande d'une monte-charge. Le moteur est commandé par deux signaux *m* et *d* pour montée et descente. Cette commande élabore ces commandes à partir de ses entrées *B*, *p1* et *p0* suivant le cahier des charges suivant :

- À l'initialisation, la monte-charge doit descendre au palier 0
- Quand la cabine est entre deux paliers, la dernière commande (*m* ou *d*) est maintenue
- Quand la cabine est à un palier, si *B* est inactif elle s'arrête, si *B* est actif, elle change de palier
- L'entrée *p0* (respectivement *p1*) indique, lorsqu'elle est active, que la monte charge est au palier 0 (respectivement palier 1).



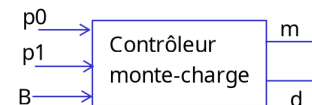
Correction (1/3)



Les machines à états

Application

- Donner la machine à états qui décrit la commande de monte-charge
- Donner le programme C/C++ implémentant la commande en adoptant les connexions suivantes : *m* <-> GPIO14, *d* <-> GPIO15, *B* <-> GPIO17, *p0* <-> GPIO18 et *p1* <-> GPIO27



Correction (2/3)

```

1 #include <iostream>
2 #include <unistd.h>
3 #include <pigpio.h>
4
5 constexpr uint8_t m {14};
6 constexpr uint8_t d {15};
7 constexpr uint8_t B {17};
8 constexpr uint8_t p0 {18};
9 constexpr uint8_t p1 {27};
10 enum class State {descend, repos0, monte, repos1};
11
12 State etat;
13 auto main() -> int {
14     if (gpioInitialise() < 0) {
15         std::cout << "Echec d'initialisation...\n";
16         exit(-1);
17     }
18
19     etat = State::descend;
20     gpioSetMode(m, PI_OUTPUT); gpioSetMode(d, PI_OUTPUT);
21     gpioSetMode(p0, PI_INPUT);
22     gpioSetMode(p1, PI_INPUT);
23     gpioSetMode(B, PI_INPUT);
    
```

Correction (2/3)

Correction (3/3)

```

1  while (true) {
2      switch (etat) {
3          case State::descend:
4              gpioWrite(m,0);
5              gpioWrite(d,1);
6              if (gpioRead(p0)) etat = State::repos0;
7              break;
8          case State::repos0:
9              gpioWrite(m,0);
10             gpioWrite(d,0);
11             if (gpioRead(B)) etat = State::monte;
12             break;
13          case State::monte:
14             gpioWrite(m,1);
15             gpioWrite(d,0);
16             if (gpioRead(p1)) etat = State::repos1;
17             break;
18          case State::repos1:
19             gpioWrite(m,0);
20             gpioWrite(d,0);
21             if (gpioRead(B)) etat = State::descend;
22             break;
23         }
24     }
25     gpioTerminate();
26     return 0;
27 }

```



Bus

Plan

- 1 Programmation avec les machines à états
- 2 Fonctions de communication
- 3 Communication sur un bus I2C (Inter Integrated Circuit)



Bus

Définition d'un bus

- Lignes de données : liaison qui assure le transfert des informations entre un élément et un autre
- Lignes d'adresses : liaison qui permet la sélection des informations à traiter dans un emplacement mémoire.
- Lignes de commandes : liaison pour assurer la synchronisation des flux d'informations sur les bus de données et d'adresses : signal horloge, demandes d'interruption, demandes d'accord (ACK), arbitrage des échanges, le contrôle des échanges (read/write, type de transfert, types des données, etc.)



Types de bus

- Bus série (i2c, uart, spi, usb, etc.) : permettent de relier entre un certain nombre de circuits en transmettant les données bit par bit
 - ❌ Lenteur (remède : augmenter la fréquence de transmission)
 - ✅ Résistant aux perturbations
 - ✅ Économique (moins de fils)
 - ✅ Adapté aux systèmes embarqués (moins d'espace)
- Bus parallèle (AGP, PCI, etc.) : est un ensemble de conducteurs électriques parallèles
 - À chaque cycle de temps, chaque conducteur transmet un bit
 - Les tailles les plus courantes (en bits) sont : 8, 16, 32, 64 ou plus
 - ❌ Couteux et encombrant
 - ❌ Moins résistant aux perturbations
 - ✅ Rapidité



Communication

Principe de communication

↳ Les communications se font par signaux

- ▣ **via un port** : point de connexion (registre I/O mappé en mémoire)
- ▣ **sur un bus** : contient plusieurs câbles et se définit par le protocole (règles) de communication.
- ▣ **grâce à un contrôleur** : les données échangées entre un périphérique et le processeur transitent par l'interface (contrôleur) associé à ce périphérique. L'interface possède de la mémoire tampon pour stocker les données échangées. Le contrôleur stocke aussi les informations nécessaires à la gestion de la communication

Communication

Caractéristiques

↳ Simplex/Duplex

- ▣ Simplex : les données circulent dans un seul sens : émetteur vers récepteur (exemple : souris->ordinateur, ordinateur -> imprimante)
- ▣ Half-duplex : les données circulent dans les 2 sens mais pas simultanément : la bande passante est utilisée en intégralité (aussi appelé alternat ou semi-duplex). Exemple : talkie/walkies, êtres humains (on ne coupe pas la parole).
- ▣ Full-duplex : les données circulent de manière bidirectionnelle et simultanément : la bande passante est divisée par 2 pour chaque sens (duplex intégral).

Communication

Caractéristiques

- ↳ Rapidité : taux de transfert
- ↳ Il existe 2 unités pour qualifier la rapidité des échanges
 - ▣ Bauds : nombre de bits de données transmis par seconde
 - ▣ Bits/sec : nombres de bits (quelconques) transmis par seconde

Communication

Caractéristiques

- ↳ Synchrone/asynchrone
- ↳ Pour une liaison série, un seul fil transporte l'information
 - ▣ Problème de synchronisation entre émetteur et récepteur (distinguer et reconnaître les séquences de bits utiles)
- ① Communication synchrone : l'émetteur et le récepteur sont cadencés à la même fréquence d'horloge
 - ▣ Un signal horloge commun
- ② Communication asynchrone : la synchronisation est imposée par le protocole
 - ▣ La transmission se fait caractère par caractère

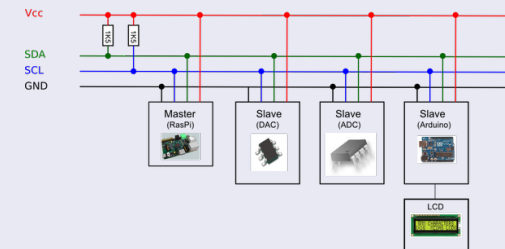
Plan

- 1 Programmation avec les machines à états
- 2 Fonctions de communication
- 3 Communication sur un bus I2C (Inter Integrated Circuit)

Le bus I2C

Le bus I2C

- I2C est un bus permettant la communication synchrone en half-duplex
- Le protocole de communication permet de mettre en communication un composant maître (généralement le microprocesseur) et plusieurs périphériques esclaves
- Plusieurs maîtres peuvent partager le même bus, et un même composant peut passer du statut d'esclave à celui de maître ou inversement.
- La communication n'a lieu qu'entre un seul maître et un seul esclave.
- Chaque esclave possède un identifiant unique (adresse). Cette adresse est fournie par l'esclave.



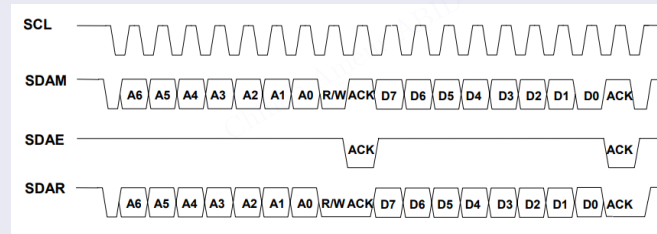
Communication sur I2C

Communication sur I2C

Principe de communication sur I2C

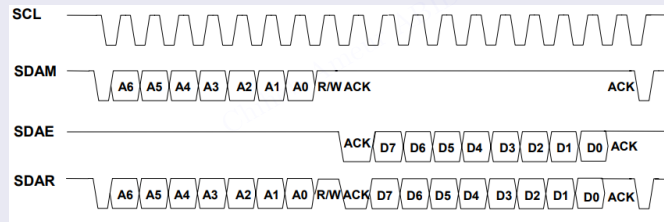
- 1 Le Maître envoie sur le bus l'adresse du composant avec qui il souhaite communiquer,
- 2 Chacun des esclaves ayant une adresse fixe, l'esclave qui se reconnaît répond à son tour par un signal de confirmation,
- 3 Puis le Maître continue la procédure de communication : écriture/lecture
- 4 Les transactions devraient être confirmées par un ACK (acquiescement)

Envoi d'une donnée



Communication sur I2C

Lecture d'une donnée



Le bus I2C

Communication I2C avec pigpio

- `int i2cOpen(unsigned i2cBus, unsigned i2cAddr, unsigned i2cFlags)` : initialise la communication sur le bus I2C pour un périphérique
 - ☞ `i2cBus` : ≥ 0 ; spécifie le bus i2c à utiliser
 - ☞ `i2cAddr` : 0-0x7F; spécifie l'adresse du périphérique sur le bus i2c
 - ☞ `i2cFlags` : doit être à 0 (pas de drapeaux définis)
 - ☞ Retourne un handle ≥ 0 pour le périphérique dont l'adresse est spécifiée si OK, sinon `PI_BAD_I2C_BUS`, `PI_BAD_I2C_ADDR`, `PI_BAD_FLAGS`, `PI_NO_HANDLE` ou `PI_I2C_OPEN_FAILED`
- `int i2cClose(unsigned handle)` : termine la communication sur le bus i2c avec le périphérique ayant le handle spécifié
 - ☞ `handle` : le handle ≥ 0 renvoyé par `i2cOpen()`

Le bus I2C

Le bus I2C

Communication I2C avec pigpio

➤ Les fonctions d'envoi

- `int i2cWriteQuick(unsigned handle, unsigned bit)` : envoyer un seul bit
- `int i2cWriteByte(unsigned handle, unsigned bVal)` : envoyer un seul octet
- `int i2cWriteDevice(unsigned handle, char *buf, unsigned count)` : envoyer jusqu'à 32 octets
- `int i2cWriteByteData(unsigned handle, unsigned i2cReg, unsigned bVal)` : écrire un seul octet dans un registre
- `int i2cWriteWordData(unsigned handle, unsigned i2cReg, unsigned wVal)` : écrire un mot (16 bits) dans un registre
- `int i2cWriteI2CBlockData(unsigned handle, unsigned i2cReg, char *buf, unsigned count)` : écrire jusqu'à 32 octets dans un registre
 - ☞ `handle` : le handle ≥ 0 renvoyé par `i2cOpen()`
 - ☞ Renvoie 0 si OK, sinon `PI_BAD_HANDLE`, `PI_BAD_PARAM` ou `PI_I2C_WRITE_FAILED`

Communication I2C avec pigpio

➤ Les fonctions de réception

- `int i2cReadByte(unsigned handle)` : lire un seul octet
- `int i2cReadDevice(unsigned handle, char *buf, unsigned count)` : lire jusqu'à 32 octets
- `int i2cReadByteData(unsigned handle, unsigned i2cReg)` : lire un seul octet depuis un registre
- `int i2cReadWordData(unsigned handle, unsigned i2cReg)` : lire un mot (16 bits) depuis un registre
- `int i2cReadI2CBlockData(unsigned handle, unsigned i2cReg, char *buf, unsigned count)` : lire jusqu'à 32 octets depuis un registre
 - ☞ `handle` : le handle ≥ 0 renvoyé par `i2cOpen()`
 - ☞ Renvoie ≥ 0 si OK, sinon `PI_BAD_HANDLE`, `PI_BAD_PARAM` ou `PI_I2C_READ_FAILED`

Merci pour votre attention



Questions?

