

ESP32

programmation multi-tâches

Module IOC — MU4IN109

Franck Wajsbürt

IOC - MU4IN109

1

Problème

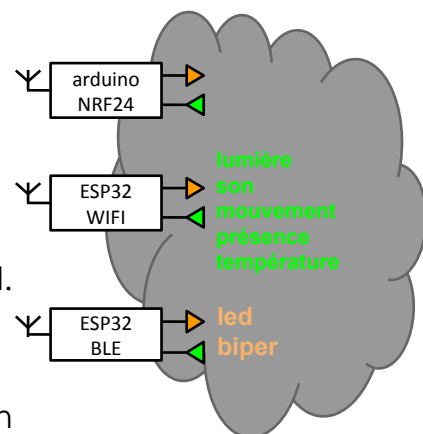
Les modules permettant de faire des mesures ou d'agir sur l'environnement ont peu de mémoire et une faible puissance de calcul, mais ils doivent gérer un grand nombre d'événements asynchrones et potentiellement un grand nombre d'actions périodiques avec les contraintes du temps réel.

Il n'est pas raisonnable*, voire il est souvent impossible*, d'installer un système d'exploitation proposant naturellement la programmation multi-threads. Pour autant, la programmation multi-threads (multi-tâches) est nécessaire pour avoir des programmes évolutifs.

Dans cette séance, nous allons voir :

1. une présentation du SoC ESP32 (micro-contrôleur évolué)
2. une proposition de programmation multi-tâches sans OS

* Pourquoi ?



IOC - MU4IN109

2

ESP 32

Qu'est ce que l'ESP32



C'est un SoC (micro-contrôleur)

- Conçu par [Espressif](https://www.espressif.com/) une société fabless chinoise (Shanghai)
- Fabriqué par TSMC en 40 nm
- Processor Xtensa dual-core 32-bit LX6 (ou single core)
- 160 MHz ou 240 MHz (*les modules des TME sont à 80 Mhz*)
- 512 kiB SRAM + 448 kiB ROM + 0 à 4 MiB Flash
- Fonctionnel entre -40°C et +125°C
- Ultra-basse consommation (5µA deep sleep)
- Intégrant des transceivers WIFI et Bluetooth
- Intégrant des accélérateurs de cryptographie
- Bon marché (qq\$ €)
- Destiné au marché IoT
- Site : <https://en.wikipedia.org/wiki/ESP32> & <http://esp32.net>



Espressif

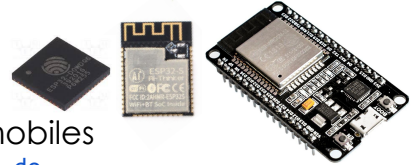
- Espressif Systems société chinoise fabless créée en 2008
Composants IoT WiFi et Bluetooth, low power, sécurisé et robuste.

<https://www.espressif.com/en/company/about-us/who-we-are>

- Conception

- SoC : ESP8266 et ESP32
- Module : SoC + flash + antenne + devices
- Board : SoC | Module + devices
- Logiciel : SDK, projets open-sources, app mobiles

<https://www.espressif.com/en/company/about-us/what-we-do>



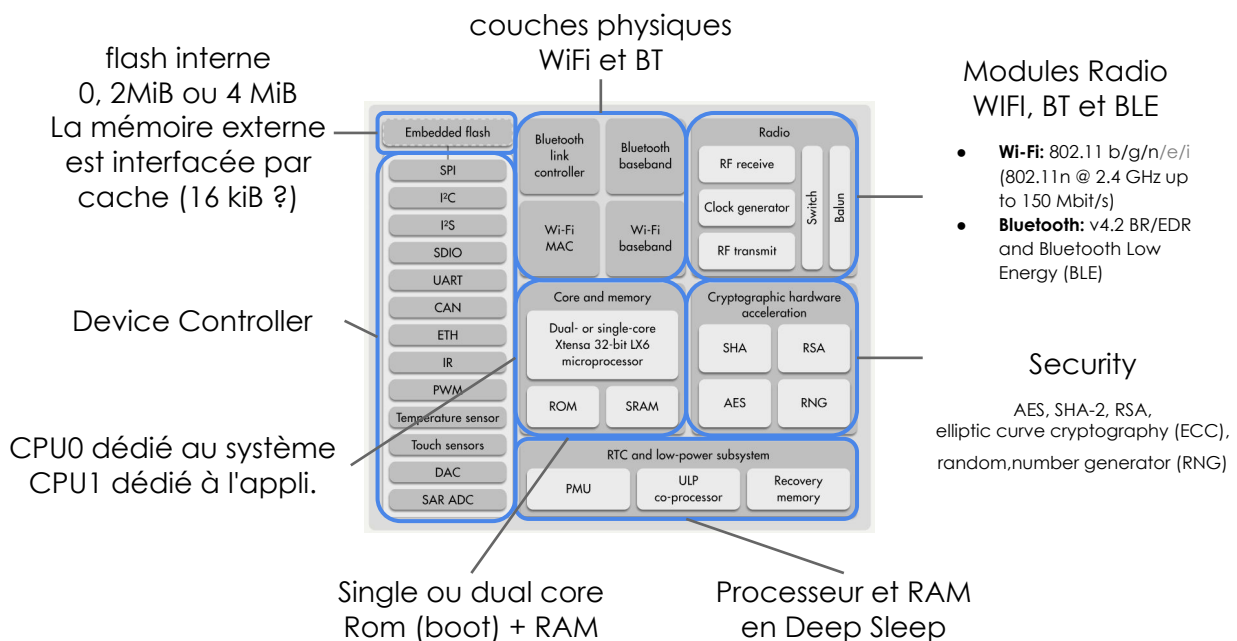
- Dates clés

- 2008 : création de Espressif Systems
- 2013 : premier circuit ESP8089 transceiver WIFI
- 2014 : premier SoC ESP8266EX WiFi
- 2016 : premier SoC ESP32 WiFi + BT
- 2018 : Plus de 100 Millions de chip vendus (en tout)
- 2025 : 200 à 300 Millions par an (famille ESP32).

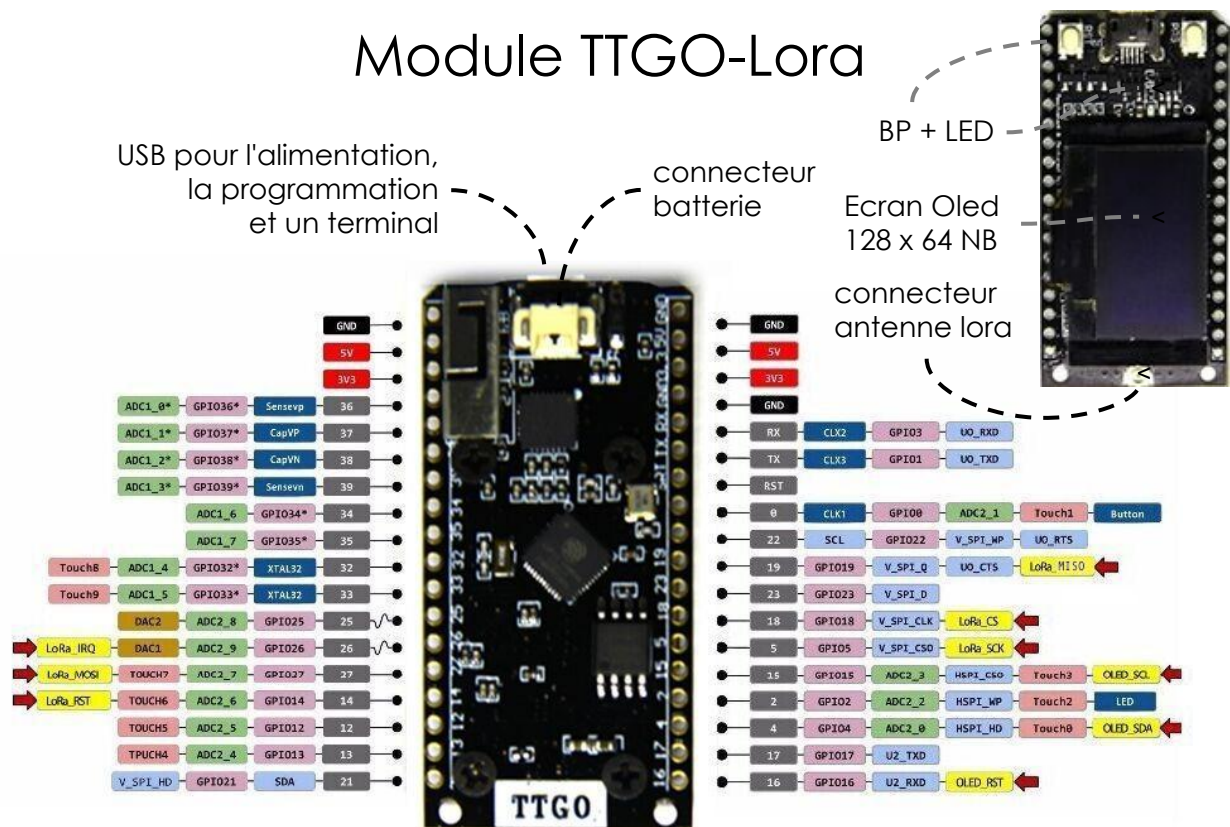
<https://www.espressif.com/en/company/about-us/milestones>



Contenu d'un ESP32



Module TTGO-Lora



<https://primalcortex.wordpress.com/2017/11/24/the-esp32-oled-lora-ttgo-lora32-board-and-connecting-it-to-ttn/>

IOC - MU4IN109

7

Caractéristiques techniques

- Microprocesseur dual core à 240 MHz (80MHz par défaut)
- 16 MiB de mémoire flash
- Connectivité
 - WiFi 802.11 b/g/n conforme à la norme IEEE 802.11 compatible avec les sécurités WPA, WPA/WPA2 et WAPI
 - Bluetooth 4.0 LE et BR/EDR
 - Lora 433MHz (SX1278) (*long-range wide-area network*)
- Entrées/Sorties (48 sur le chip mais moins sur le module)
 - 26x E/S numériques (3.3V)
 - 12x entrées analogiques (SAR - Successive Approximation Register)
 - 4x SPI, 2x I²S*, 2x I²C, 3x UART, CAN 2.0, IR, Touch Sensor
- Capteur de température (précision 5 à 10°...)
- Cryptographie :
 - AES, SHA-2, RSA, ECC, random number generator (RNG)

* Integrated Interchip Sound

IOC - MU4IN109

8

Programmation

L'ESP32 se programme de plusieurs manières :

- FreeRTOS
- IDF Internet Development Framework développé par Espressif
 - C
 - MicroPython
 - **C++ Arduino**

Le chargement du programme (firmware) peut être :

- par liaison série via USB
- par Wi-Fi (OTA, Over The Air)

Le mécanisme OTA repose sur l'utilisation de 2 partitions applicatives :

- P0 : FreeRTOS + IDF + Arduino + application (incluant le code OTA)
- P1 : FreeRTOS + IDF + Arduino + application (incluant le code OTA)

L'application en cours d'exécution (P0 ou P1) peut télécharger et écrire une nouvelle version dans l'autre partition, puis redémarrer dessus.

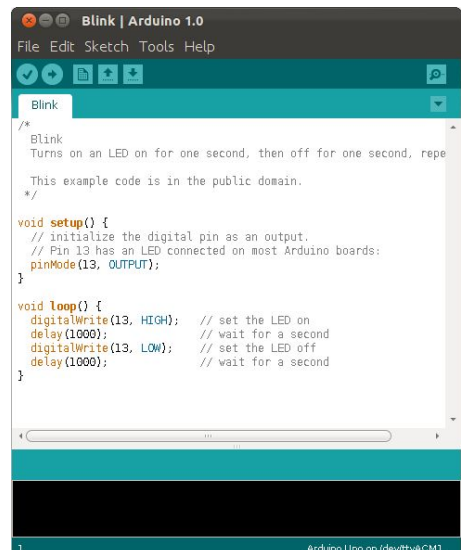
Environnement de développement logiciel

L'environnement de développement (IDE) : <http://arduino.cc/en/main/software>

- écrit en Java (linux, windows, macos)
 - éditeur de code
 - compilateur
 - programmeur
 - terminal de commande
- Il est possible de compiler et de charger les programmes en lignes de commande.

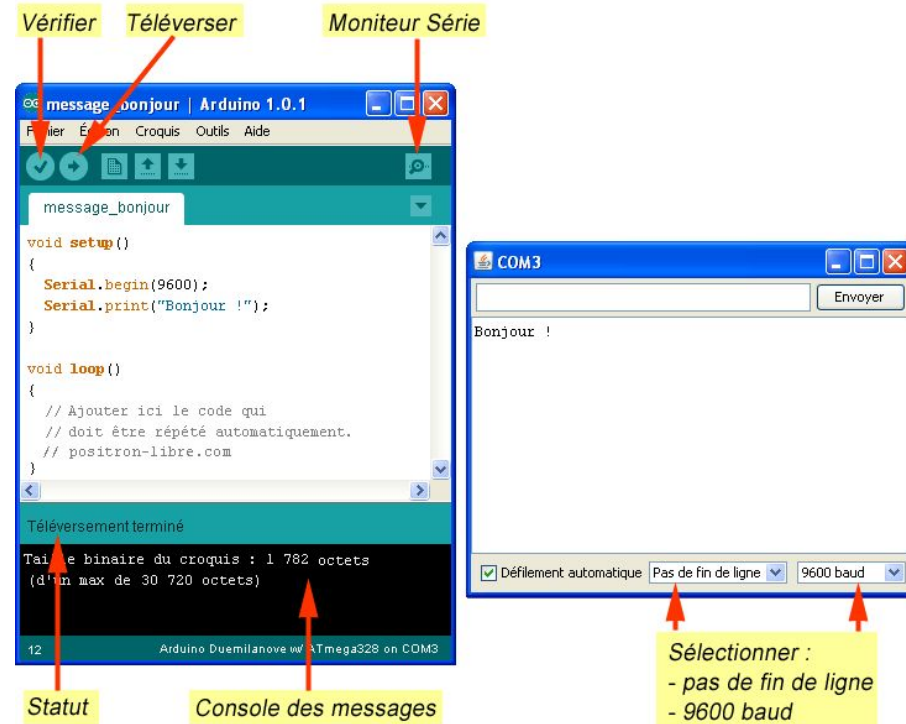
Langage de programmation

- C++, compilé avec avr-g++
- bibliothèque de développement Arduino nommée Wiring pour le contrôle des composants internes du microcontrôleur.
- Un programme Arduino se nomme **sketch**, composé au minimum de 2 fonctions
 - **setup()** exécutée une fois pour initialiser les composants et les variables
 - **loop()** exécutée en boucle jusqu'à l'extinction de la carte



Menus de la fenêtre Arduino

<http://www.positron-libre.com/robotique/robots/boe-shield-bot/notice/premier-programme.php>



IOC - MU4IN109

11

Exemple de programme

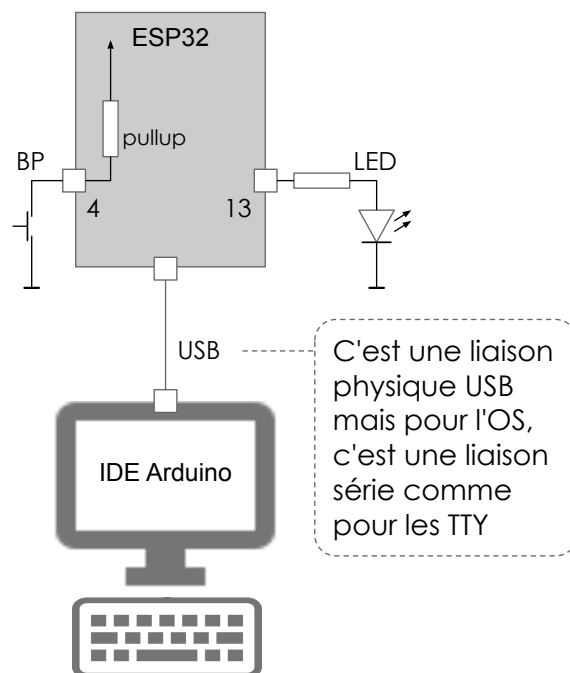
```

// Pas d'include par défaut
// Pas de fonction main()

// IL faut connaître Les pins et L'architecture
const int buttonPin = 4;
const int ledPin = 13;
int buttonState = 0;

// Fonction exécutée une seule fois au reset
void setup() {
  Serial.begin(115200);
  pinMode(buttonPin, INPUT_PULLUP);
  pinMode(ledPin, OUTPUT);
}

// Fonction exécutée en boucle sans arrêt
void loop() {
  buttonState = digitalRead(buttonPin);
  Serial.println(buttonState);
  if (buttonState == HIGH) {
    digitalWrite(ledPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }
}
    
```



IOC - MU4IN109

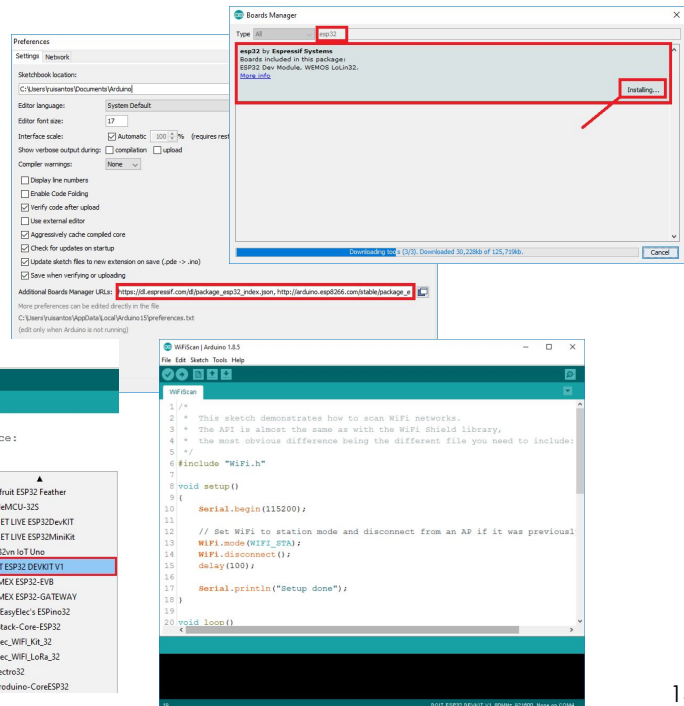
12

Installation sur Arduino

<https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/>

1. Installer les cartes ESP32
installation de la base
de données des cartes
ESP32 existantes

2. Tester



IOC - MU4IN109

13

ARDUINO CHEAT SHEET V.02C

Mostly taken from the extended reference:
<http://arduino.cc/en/Reference/Extended>
Gavin Smith – Robots and Dinosaurs, The Sydney Hackspace

Structure

```
void setup() { void loop() }
```

Control Structures

```
if (x<5) { } else { }  
switch (myvar) {  
  case 1:  
    break;  
  case 2:  
    break;  
  default:  
    break;  
}  
for (int i=0; i<= 255; i++) { }  
while (x<5) { }  
do { } while (x<5);  
continue; // Go to next in do/for/while loop  
return x; // Or "return"; for voids.  
goto // considered harmful :-)
```

Further Syntax

```
// (single line comment)  
/* (multi-line comment) */  
#define DOZEN 12 //Not baker's!  
#include <avr/pgmspace.h>
```

General Operators

= (assignment operator)
+ (addition) - (subtraction)
* (multiplication) / (division)
% (modulo)
== (equal to) != (not equal to)
< (less than) > (greater than)
<= (less than or equal to) >= (greater than or equal to)
&& (and) || (or) ! (not)

Pointer Access

& reference operator
* dereference operator

Bitwise Operators

& (bitwise and) | (bitwise or)
^ (bitwise xor) ~ (bitwise not)
<< (bitshift left) >> (bitshift right)

Compound Operators

++ (increment) -- (decrement)
+= (compound addition)
-= (compound subtraction)
*= (compound multiplication)
/= (compound division)
&= (compound bitwise and)
|= (compound bitwise or)

Conversion

char()	byte()
int()	word()
long()	float()

Constants

HIGH | LOW
INPUT | OUTPUT
true | false
143 // Decimal number
0173 // Octal number
0b1101111 // Binary
0x7B // Hex number
7U // Force unsigned
15UL // Force long unsigned
10.0 // Forces floating point
2.4e5 // 240000

Data Types

void
boolean (0, 1, false, true)
char (e.g. 'a' -128 to 127)
unsigned char (0 to 255)
byte (0 to 255)
int (-32,768 to 32,767)
unsigned int (0 to 65535)
word (0 to 65535)
long (-2,147,483,648 to 2,147,483,647)
unsigned long (0 to 4,294,967,295)
float (-3.4028235E+38 to 3.4028235E+38)
double (currently same as float)
sizeof(myint) // returns 2 bytes

Strings

```
char S1[15];  
char S2[8]={'a','r','d','u','i','n','o'};  
char S3[8]={'a','r','d','u','i','n','o','\0'};  
//Included \0 null termination  
char S4[] = "arduino";  
char S5[8] = "arduino";  
char S6[15] = "arduino";
```

Arrays

```
int myInts[6];  
int myPins[] = {2, 4, 8, 3, 6};  
int mySensVals[6] = {2, 4, -8, 3, 2};
```

Qualifiers

static // persists between calls
volatile // use RAM (nice for ISR)
const // make read-only
PROGRAM // use flash

Digital I/O

```
pinMode(pin, (INPUT,OUTPUT))  
digitalWrite(pin, value)  
int digitalRead(pin)  
//Write High to inputs to use pull-up res
```

Analog I/O

```
analogReference((DEFAULT,INTERNAL,EXTERNAL))  
int analogRead(pin) //Call twice if switching pins from high Z source.  
analogWrite(pin, value) // PWM
```

Advanced I/O

```
tone(pin, freqHz)  
tone(pin, freqHz, duration_ms)  
noTone(pin)  
shiftOut(dataPin, clockPin, [MSBFIRST,LSBFIRST], value)  
unsigned long pulseIn(pin, [HIGH,LOW])
```

Time

```
unsigned long millis() // 50 days overflow.  
unsigned long micros() // 70 min overflow  
delay(ms)  
delayMicroseconds(us)
```

Math

```
min(x,y) max(x,y) abs(x)  
constrain(x, minval, maxval)  
map(val, fromL, fromH, toL, toH)  
pow(base, exponent) sqrt(x)  
sin(rad) cos(rad) tan(rad)
```

Random Numbers

```
randomSeed(seed) // Long or int  
long random(max)  
long random(min, max)
```

Bits and Bytes

```
lowByte() highByte()  
bitRead(x,bin) bitWrite(x,bin,bit)  
bitSet(x,bin) bitClear(x,bin)  
bit(bitin) / bit: 0-LSB 7-MSB
```

External Interrupts

```
attachInterrupt(interrupt, function, [LOW,CHANGE,ISING,FALLING])  
detachInterrupt(interrupt)  
interrupts()  
noInterrupts()
```

Libraries:

Serial.
begin([300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200])
end()
int available()
int read()
flush()
print()
println()
write()
EEPROM (#include <EEPROM.h>)
byte read(intAddr)
write(intAddr,myByte)
Servo (#include <Servo.h>)
attach(pin, [min_uS, max_uS])
write(angle) // 0-180
writeMicroseconds(us) //1000-2000, 1500 is midpoint
read() // 0-180
attached() //Returns boolean
detach()
SoftwareSerial(RxPin,TxPin)
// #include<SoftwareSerial.h>
begin(longSpeed) // up to 9600
char read() // blocks till data
print(myData) or println(myData)
Wire (#include <Wire.h>) // For I2C
begin() // Join as master
begin(addr) // Join as slave @ addr
requestFrom(addr, count)
beginTransmission(addr) // Step 1
send(mybyte) // Step 2
send(char * mystring)
send(byte * data, size)
endTransmission() // Step 3
byte available() // Num of bytes
byte receive() //Return next byte
onReceive(handler)
onRequest(handler)

Wire

```
Wire.begin() // Join as master  
Wire.begin(addr) // Join as slave @ addr  
Wire.requestFrom(addr, count)  
Wire.beginTransmission(addr) // Step 1  
Wire.send(mybyte) // Step 2  
Wire.send(char * mystring)  
Wire.send(byte * data, size)  
Wire.endTransmission() // Step 3  
Wire.available() // Num of bytes  
Wire.receive() //Return next byte  
Wire.onReceive(handler)  
Wire.onRequest(handler)
```

ATmega168

Flash (3x for bootloader)	ATmega168	ATmega328	ATmega1280
SRAM	1kB	32kB	128kB
EEPROM	512B	1kB	4kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Flash (3x for bootloader)	ATmega328P	ATmega1280
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega1280

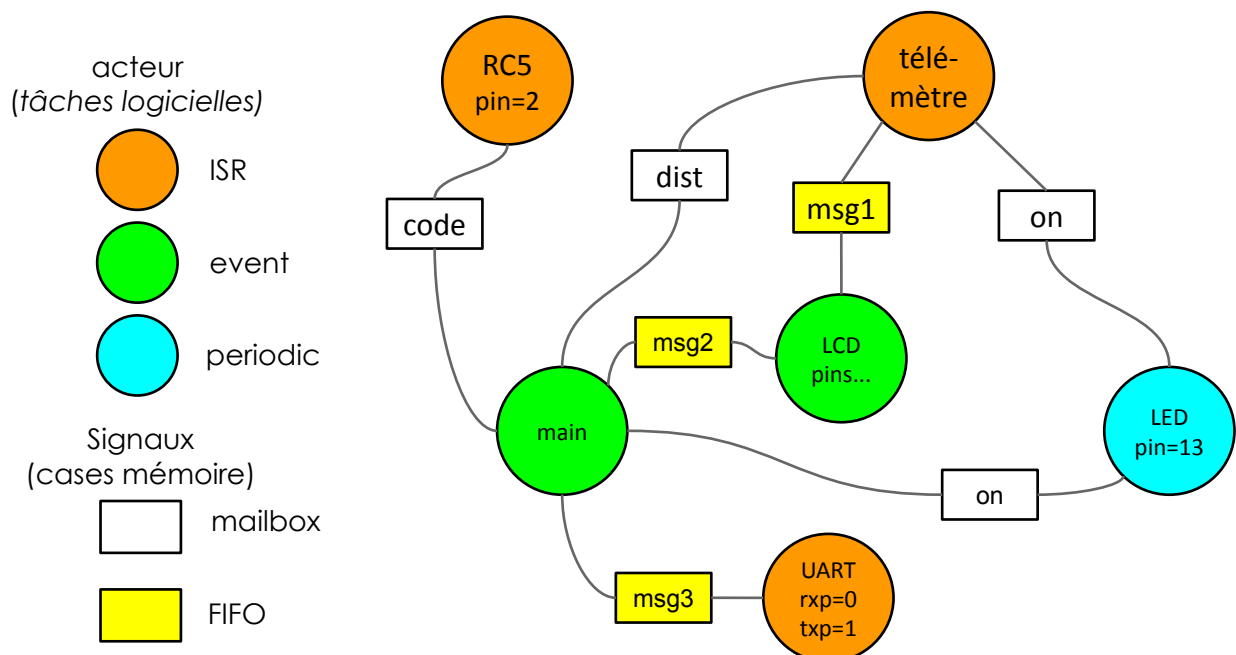
Flash (3x for bootloader)	ATmega1280	ATmega328P
SRAM	1kB	32kB
EEPROM	512B	1kB

ATmega328P

Programmation par automates

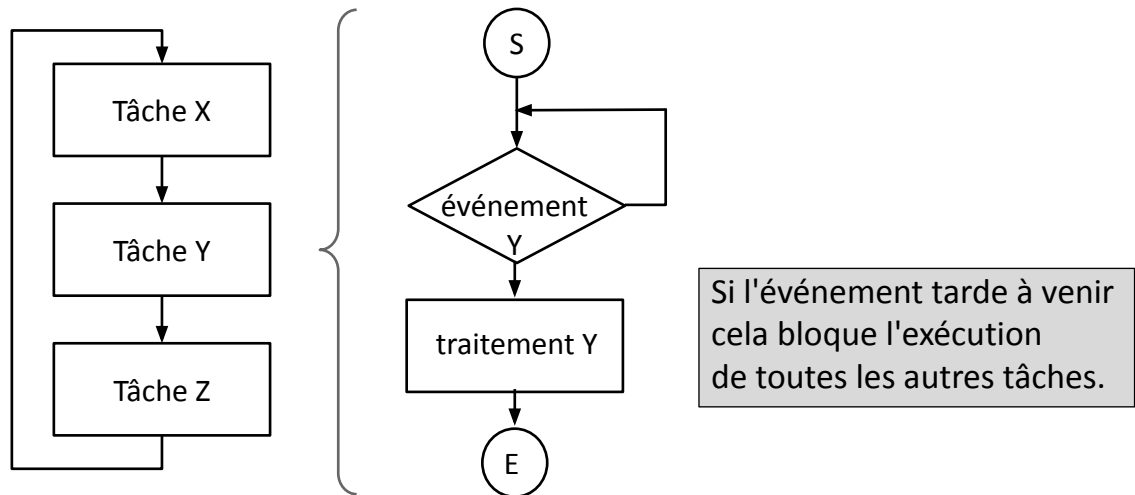
Applications

Les applications peuvent être vues comme des acteurs matériels communiquant par des canaux de communication. Chaque acteur (tâche) étant décrit par un automate d'états finis.



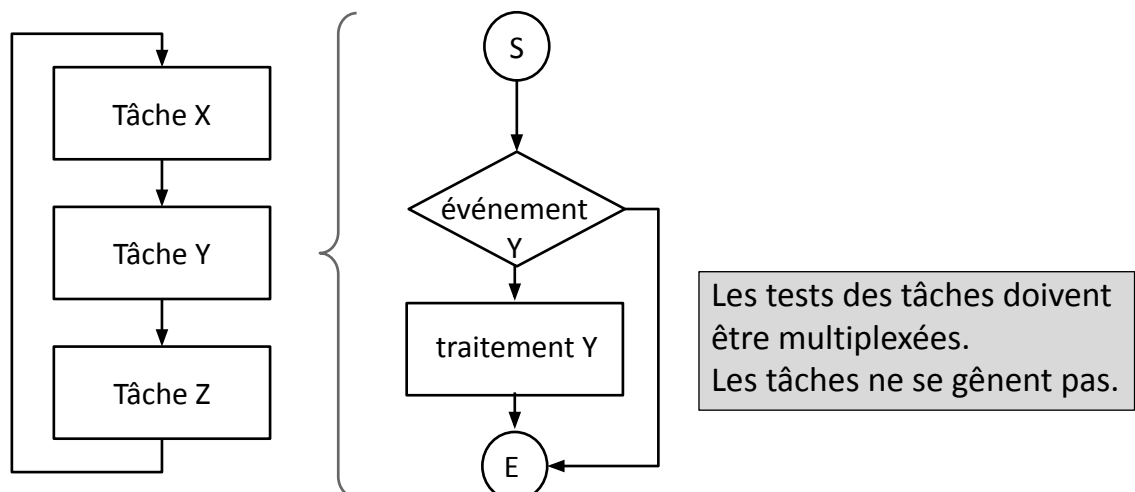
Problème : exécution de plusieurs tâches

Un grand nombre des tâches destinées à un microcontrôleur dépendent d'événements externes périodiques ou non.



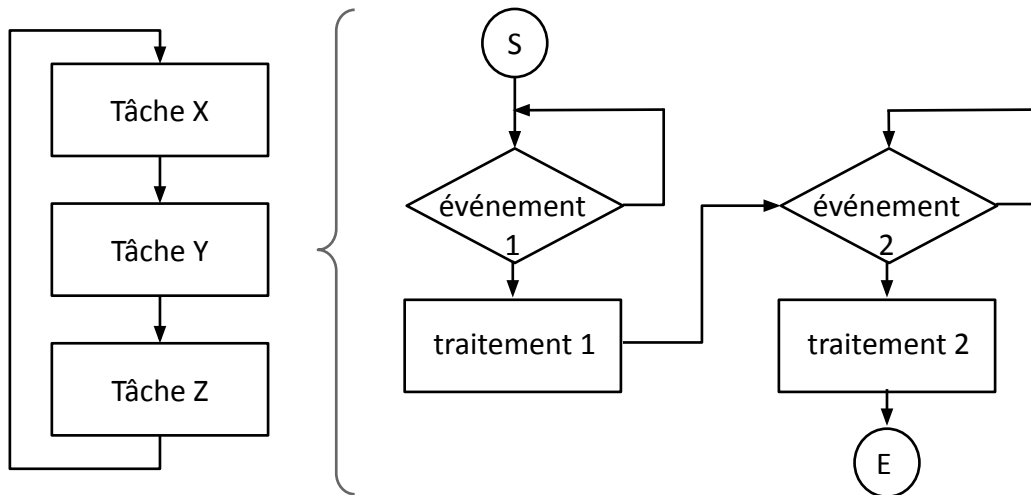
→ si les tâches coopèrent, on attend pas

Si on peut garantir que les traitements sont bornés, alors il n'est pas obligatoire d'attendre l'événement.



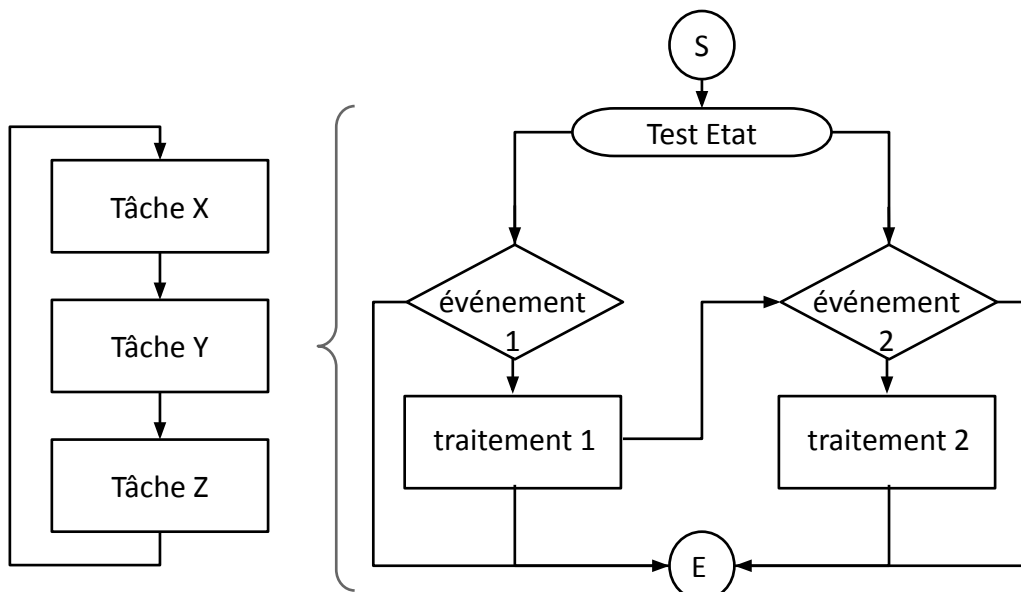
Une tâche peut avoir plusieurs traitements

Quand l'événement 1 arrive, il faut exécuter le traitement 1
Quand l'événement 2 arrive, il faut exécuter le traitement 2



→ les tâches doivent avoir des états

Si on n'attend pas les événements, il faut se souvenir où en était la tâche.



Ordonnancement des tâches

- Une tâche est représentée par une fonction dont les arguments sont des entrées/sorties (adresses de cases mémoire)
`void tache_1 (types entrées_sorties, ...)`
- Une instance d'exécution est l'exécution de la fonction de la tâche, cette instance est bornée dans le temps par construction
- L'ordonnancement consiste à exécuter les fonctions en boucle dans la fonction **loop()**

```
int x, y, z, t;
void loop() {
    step_tache_1(&x, &y, &z);
    step_tache_2(&x, &t);
    step_tache_3(&y, &z);
}
```

- La fonction setup va servir à initialiser un état interne.

Représentation de l'état interne

- Le comportement de la tâche est représenté par 2 fonctions :
init (appelée par la fonction setup) et step (appelée par la fonction loop)
- La fonction step peut avoir des variables locales (temporaires) pour ses calculs
- Une tâche peut avoir un état interne (p.ex. le registre Etat)
qui *pourrait* être représenté dans des variables locales **static**,
mais cela pose deux problèmes :
 1. Il n'est pas possible d'initialiser ce registre d'état dans la fonction init()
 2. Si une tâche a plusieurs exemplaires, il faut autant de fonctions que d'exemplaires.
- Donc on utilise une variable globale comme contexte d'exécution de la tâche

```
typedef struct ctx_tache_st {
    int etat;
    int x, y;
} ctx_tache_t;
ctx_tache_t ctx_tache;
void init_tache(ctx_tache_t *ctx_tache, int ETAT, int x, int y) {
    ctx_tache->etat = ETAT;
    ctx_tache->x = x;
    ctx_tache->y = y;
}
void step_tache(ctx_tache_t *ctx_tache, int * in_out, ...) {
    switch (ctx_tache->etat) {
    case ETAT:
        ...
    }
}
```

```
typedef struct ctx_tache_st {
    int etat, x, y;
} ctx_t0, ctx_t1;
int x;
setup() {
    init_t0(&ctx_t0, ETAT0,1,1);
    init_t1(&ctx_t1, ETAT1,2,2);
}
loop() {
    step_t0(&ctx_t0, &x);
    step_t1(&ctx_t1, &x);
}
```

Tâches périodiques

- Pour beaucoup de tâches, l'événement attendu est le temps. Par ex: on doit exécuter une tâche toutes les 20ms.
- Il y a deux solutions :
 1. Configurer un timer pour envoyer une interruption périodique
 2. Utiliser une horloge qui compte le temps et la consulter souvent

Soit H une horloge qui s'incrémente automatique

Soit T un registre qui servira à enregistrer le temps (date)

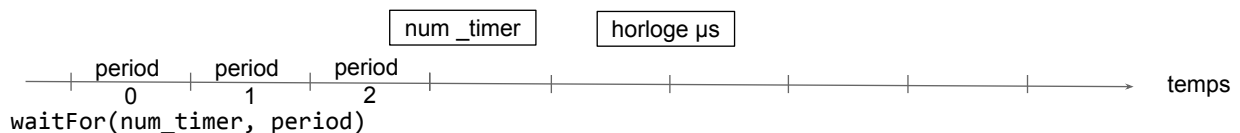
Soit P une période

Initialisation $T \leftarrow H + P$

tâche :

```
si (H > T)          // événement attendu (on attend une date)
    exécution
    T ← T + P      // mise à jour de la date
fsi
```

Fonction waitFor



waitFor(num_timer, period)

rend le nombre de périodes écoulées depuis le dernier appel

micros()

rend la valeur de l'horloge (sur 32 bits donc retour à 0 toutes les 1h11m)

```
// Configuration :
// - MAX_WAIT_FOR_TIMER : nombre maximum de timers utilisés
// arguments :
// - timer : numéro de timer entre 0 et MAX_WAIT_FOR_TIMER-1
// - period : période souhaitée exprimée en microsecondes
// valeur de retour :
// - Retourne le nombre de périodes écoulées (0, 1, ou >1 si retard, non significatif au 1er appel)

#define MAX_WAIT_FOR_TIMER 2
unsigned long waitFor(int timer, unsigned long period)
{
    static unsigned long last_period[MAX_WAIT_FOR_TIMER];
    unsigned long current = micros() / period;           // numéro de période
    unsigned long delta = current - last_period[timer];   // gère le wrap-around
    if (delta) last_period[timer] = current;              // mise à jour si déclenchement
    return delta;
}
```

"Tâche" ISR

- Les tâches invoquées par loop ont un ordonnancement fifo
- Arduino permet facilement de configurer des ISR (Interrupt Service Routine)

source:

<https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/>

```
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state, event;

// parce que c'est une ISR qui va modifier ces variables

void init_blink() {
    pinMode(ledPin, OUTPUT);
    pinMode(interruptPin, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(interruptPin), step_blink_frontend, CHANGE);
    state = LOW;
}

// on ne peut pas passer un contexte ici...

void step_blink_frontend() {
    state = !state; event = 1;
}

// c'est une ISR

void step_blink_backend() {
    if (event) {
        digitalWrite(ledPin, state); event = 0;
    }
}

// ce que l'ISR n'a pas fait

void setup() {
    init_blink();
}

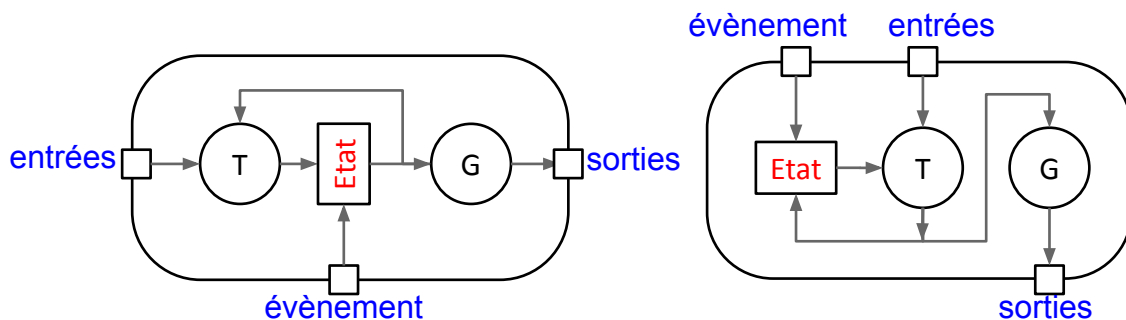
void loop() {
    step_blink_backend();
}
```

- ici blink est un type de "tâche" ISR

IOC - MU4IN109

25

Une tâche est une machine d'états finis



Aller à l'état courant (p.ex. E0)
Etat=E0:
 Si événement attendu
 Alors
 G : sorties ← traitement
 T : en fonction des entrées
 Etat ← état suivant
 Fsi
Etat=E1:
 [...]

Aller à l'état courant (p.ex. E0)
Etat=E0:
 Si événement attendu
 Alors
 en fonction des entrées
 T : Etat ← état suivant
 G : sorties ← traitement
 Fsi
Etat=E1:
 [...]

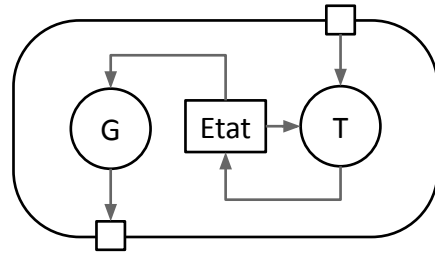
IOC - MU4IN109

26

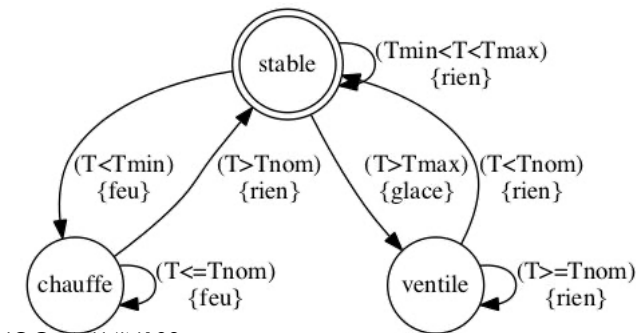
FSM : Machine à états finis

Une FSM (automate) est défini par :

- un nombre d'états finis
- un état initial
- un état courant
- une fonction de transition d'états définissant l'état futur à partir des entrées et de l'état courant
- une fonction de génération définissant les sorties à partir de l'état courant ou futur



On décrit le comportement d'uns FSM avec son graphe de transition d'états



Climatiseur

INPUTS

3 températures : Tmin - Tnom - Tmax (p.ex. 18° 19° 20°)
la température T

ACTIONS

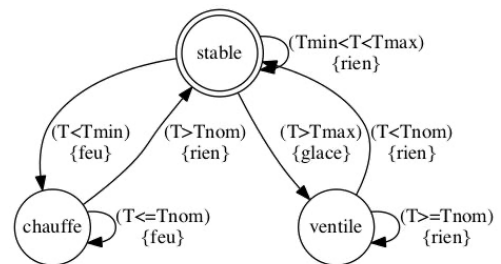
feu : {ventilateur = OFF; radiateur = ON;}
glace : {ventilateur = ON; radiateur = OFF;}
rien : {ventilateur = OFF; radiateur = OFF;}

IOC - MU4IN109

27

Automate en C

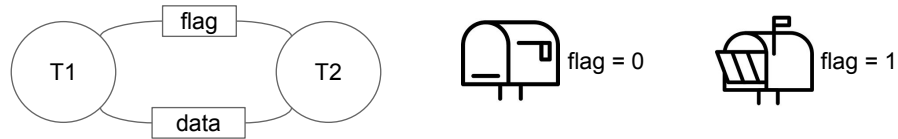
```
enum {STABLE, CHAUFFE, VENTIL} state;
void init_clim(ctx_clim_t * ctx, int timer, int period, int *T, int pin_V, int pin_C) {
    ctx->state=STABLE; ctx->ventil=OFF; ctx->radiateur=OFF;
    ctx->timer = timer;
    ctx->period = period;
    ctx->T = T; ctx->pin_v = pin_V; ctx->pinc = pin_C;
}
void step_clim(ctx_clim_t * ctx) {
    if (!waitfor(ctx->timer, ctx->period) return;
    switch (ctx->state) {
        case STABLE:
            if (ctx->T>Tmax) {ctx->state=VENTIL; ctx->ventil=ON; ctx->radiateur=OFF;}
            else if (ctx->T<Tmin) {ctx->state=CHAUFFE; ctx->ventil=OFF; ctx->radiateur=ON;}
            else {ctx->ventil=OFF; ctx->radiateur=OFF;}
            break;
        case CHAUFFE:
            if (ctx->T>Tnom) {ctx->state=STABLE; ctx->ventil=OFF; ctx->radiateur=OFF;}
            break;
        case VENTIL:
            if (ctx->T<Tnom) {ctx->state=STABLE; ctx->ventil=OFF; ctx->radiateur=OFF;}
            break;
    }
    digitalWrite(ctx->pin_v, ctx->ventil);
    digitalWrite(ctx->pin_c, ctx->radiateur);
}
```



IOC - MU4IN109

28

Communication des tâches : boîte à lettre



- Les tâches communiquent
 - une T1 produit une donnée qu'utilise une tâche T2
 - le plus simple est d'utiliser une variable globale comme boîte à lettre

```
typedef struct boite_st {  
    int flag;  
    int data;  
} boite_t;  
  
boite_t b;  
  
void T1 (... , boite_t *b,...) {  
    if (b->flag == 0) {  
        utiliser b->data;  
        b->data = VALUE;  
        b->flag = 1;  
    }  
}  
  
void T2 (... , boite_t *b,...){  
    if (b->flag == 1) {  
        utiliser b->data;  
        b->data = RETOUR;  
        b->flag = 0;  
    }  
}
```

dans setup ()
b->flag = 0;
b->data = valeur_initiale;

- On peut utiliser la boîte dans les deux sens
- On peut faire une communication avec et sans perte

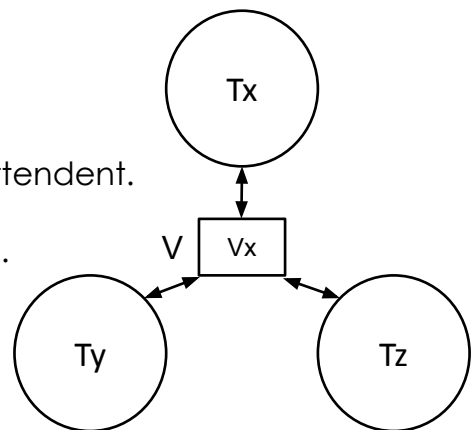
Set-Reset Flag

Pour la synchronisation des tâches on utilise une variable partagée V.

- On associe à chaque tâche T_i une valeur propre V_i .
- Une tâche T_x ne peut changer la valeur de la variable que si elle contient sa valeur propre V_x

A l'initialisation:

- $V \leftarrow V_x$
Tx peut donc travailler, les autres tâches attendent.
- Puis Tx place V_y dans V
Ty peut alors travailler, les autres attendent.
- Puis Ty place V_z dans V
C'est alors Tz qui peut avancer



Buffer + flags (Set-Reset-Flag)

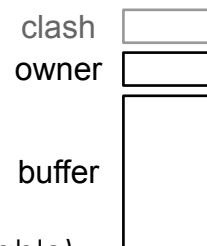
Si deux tâches T1 et T2 s'échangent des données par BAL

- La valeur du drapeau désigne le propriétaire du buffer
 - le buffer appartient à T1
 - T1 peut lire ce que contient le buffer
 - T1 peut remplir le buffer
 - T1 informe T2 que le buffer est plein en mettant 1 dans le drapeau
 - le buffer appartient à T2
 - T2 peut lire ce qu'à écrit T1
 - T2 peut écrire une réponse
 - T2 informe T1 que le buffer est lu (et qu'une réponse est mise) en mettant 0 dans le drapeau

Si T1 s'interdit d'écrire si le drapeau est à 1
 ⇒ communication sans perte

Si T1 écrit même quand le drapeau owner est à 1
 ⇒ communication à perte
 Il faut un drapeau de collision en plus (clash)

Le buffer peut être utilisée dans les deux sens (entente préalable)



FIFO

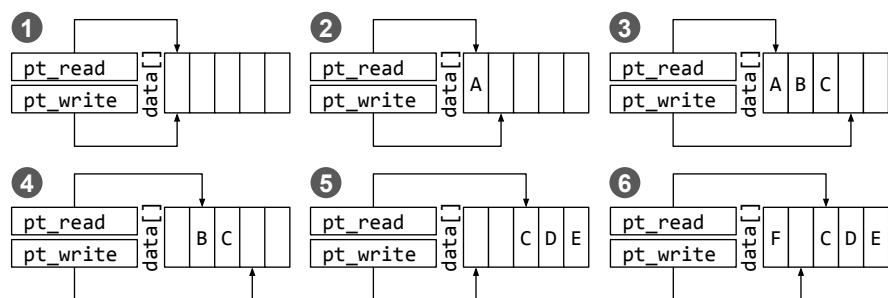
Un dernier mécanisme de communication simple est la FIFO
 (ici on perd une case, mais on peut ne pas la perdre avec un compteur de phase)

Suivant le comportement de **tty_fifo_push()** on peut avoir une FIFO avec ou sans perte

```
struct tty_fifo_s {
    char data[20];
    int pt_read;
    int pt_write;
};
```

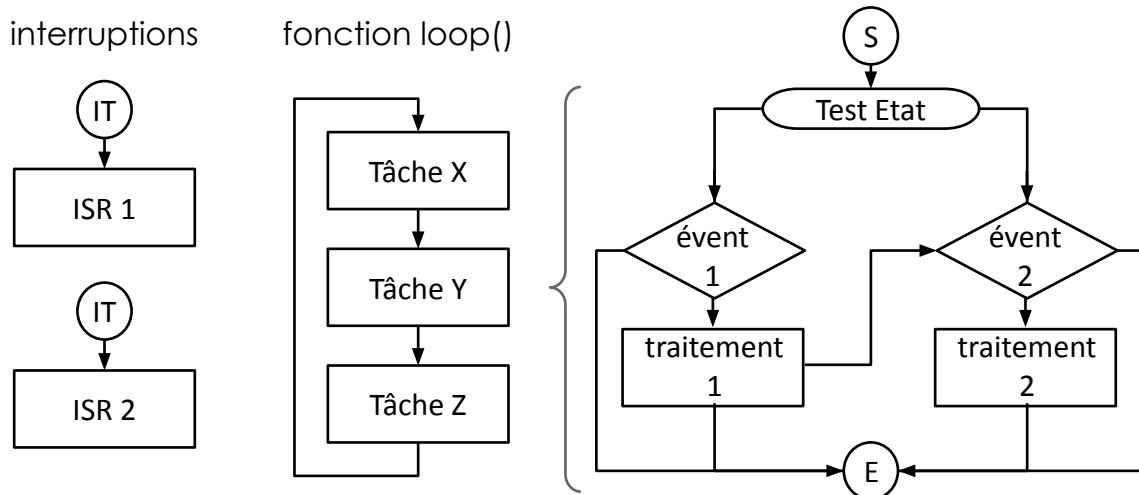
```
static int tty_fifo_push(struct tty_fifo_s *fifo, int c) {
    int pt_write_next = (fifo->pt_write + 1) % sizeof(fifo->data);
    if (pt_write_next != fifo->pt_read) {
        fifo->data[fifo->pt_write] = c;
        fifo->pt_write = pt_write_next;
        return 1;
    }
    return 0;
}
```

```
static int tty_fifo_pull(struct tty_fifo_s *fifo, int *c) {
    if (fifo->pt_read != fifo->pt_write) {
        *c = fifo->data[fifo->pt_read];
        fifo->pt_read = (fifo->pt_read + 1) % sizeof(fifo->data);
        return 1;
    }
    return 0;
}
```



En résumé 1/2

Les tâches sont des fonctions exécutées périodiquement ou après une interruption.
Ces fonctions réalisent un traitement unitaire borné dans le temps.



En résumé 2/2

- Une interface vers les signaux
- Des paramètres de configuration globaux
- Un contexte
 - mémoire d'état interne
 - configuration d'instance
- Des fonctions de comportement
 - initialisation : init
 - cas normal : step ou ISR
 - *exception : cas d'erreur optionnel*
- mailbox
 - un buffer de données
 - un drapeau d'état
 - ready / busy
 - full / empty
- FIFO
 - un tableau de buffer
 - un état
 - un pointeur de lecture
 - un pointeur d'écriture
 - API push - pull

TME

- Programmation de l'ESP32 en multitâches
 - installer l'environnement ESP32
 - faire clignoter une led (et oui... encore)
 - faire clignoter la led en fonction de la lumière reçue
 - afficher l'état du bouton poussoir sur l'écran oled
 - configurer le comportement en fonction de commandes reçu sur le terminal
 - ...