

# ESP32

## programmation multi-tâches

Module IOC — MU4IN109

Franck Wajsbürt

IOC - MU4IN109

1

## Problème

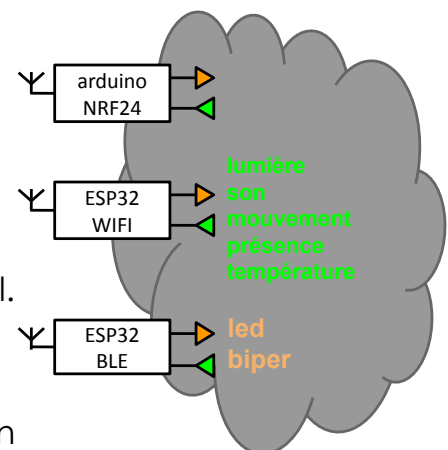
Les modules permettant de faire des mesures ou d'agir sur l'environnement ont peu de mémoire et une faible puissance de calcul, mais ils doivent gérer un grand nombre d'événements asynchrones et potentiellement un grand nombre d'actions périodiques avec les contraintes du temps réel.

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

IOC - MU4IN109

3

## 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>

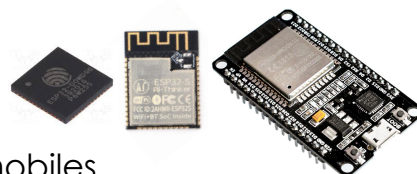


IOC - MU4IN109

4

# Espressif

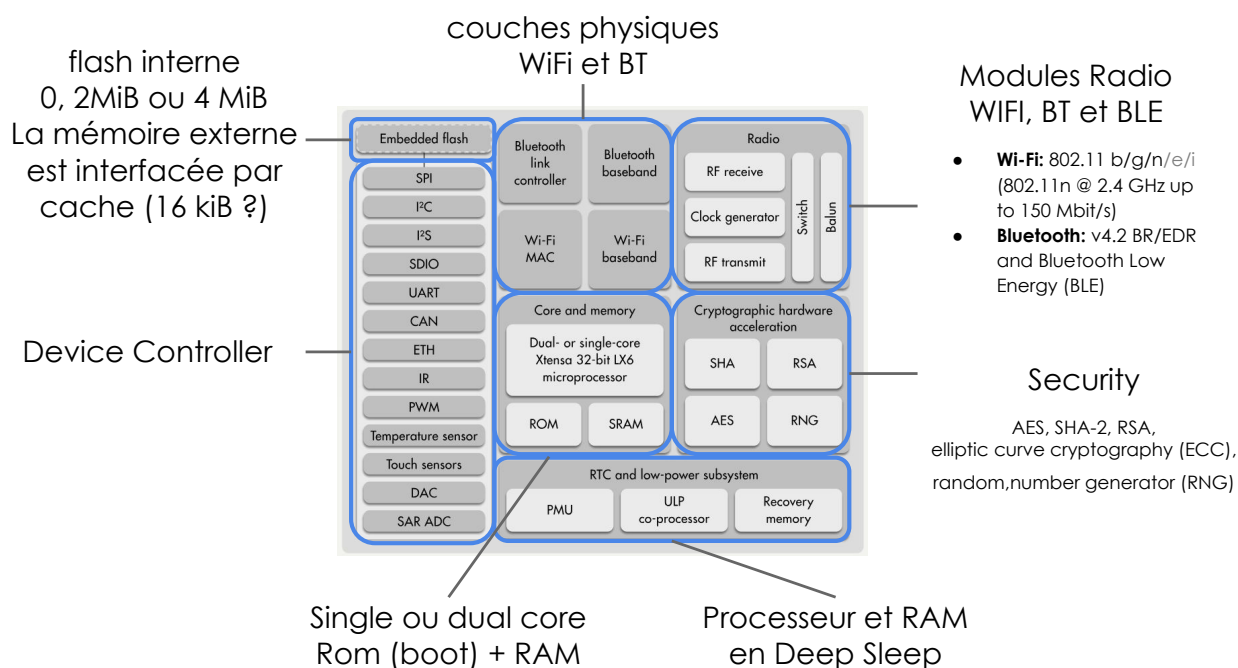
- Espressif Systems société chinoise fabless créée en 2008  
Composants IoT WiFi et Bluetooth, low power, sécurisé et robuste.  
<https://www.espressif.com/en/company/about-us/who-we-are>
- Conception
  - SoC : ESP8266 et ESP32
  - Module : SoC + flash + antenne + devices
  - Board : SoC | Module + devices
  - 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.<https://www.espressif.com/en/company/about-us/milestones>



IOC - MU4IN109

5

## Contenu d'un ESP32

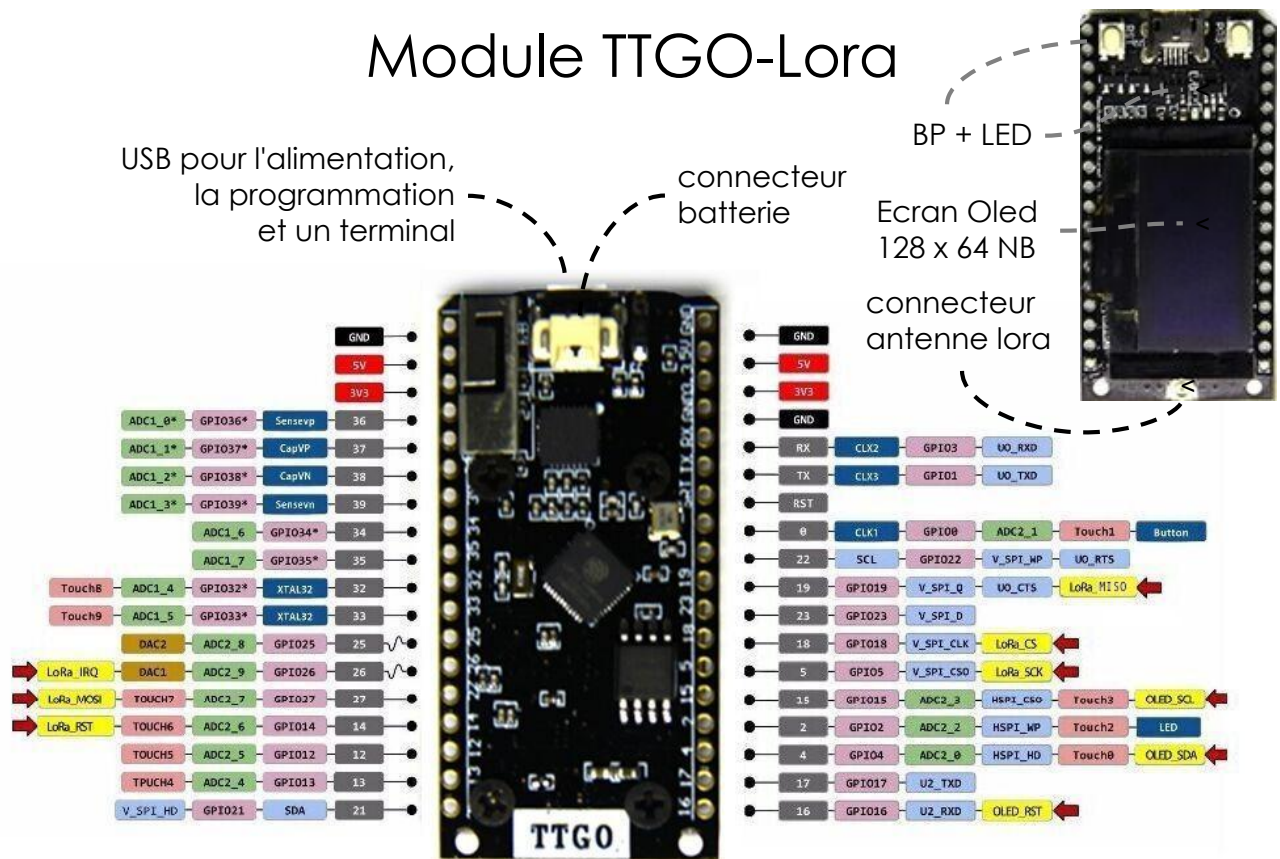


IOC - MU4IN109

<http://esp32.net/>

6

# 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 (en TP 80MHz)
- 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 I<sup>2</sup>S\*, 2x I<sup>2</sup>C, 3x UART, CAN 2.0, IR, Touch Sensor
- Capteur de température
- Cryptographie :
  - AES, SHA-2, RSA, ECC, random number generator (RNG)

\* Integrated Interchip Sound

IOC - MU4IN109

8

# Programmation

L'ESP32 se programme de plusieurs manières :

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

Le chargement (bootloader) peut être :

- **Par liaison série via USB**
- Par Wifi OTA (Over The Air)

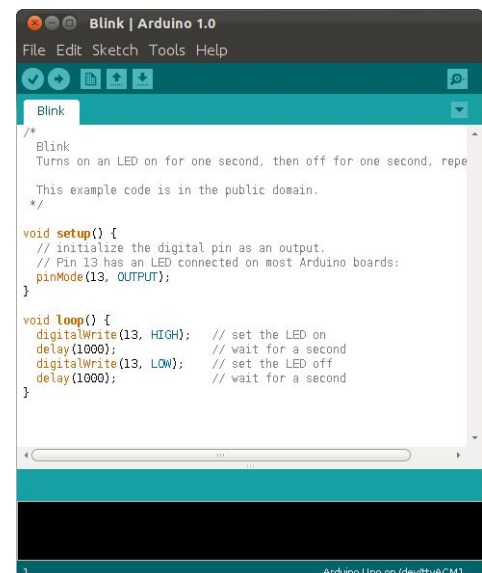
## Environnement de développement logiciel

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.

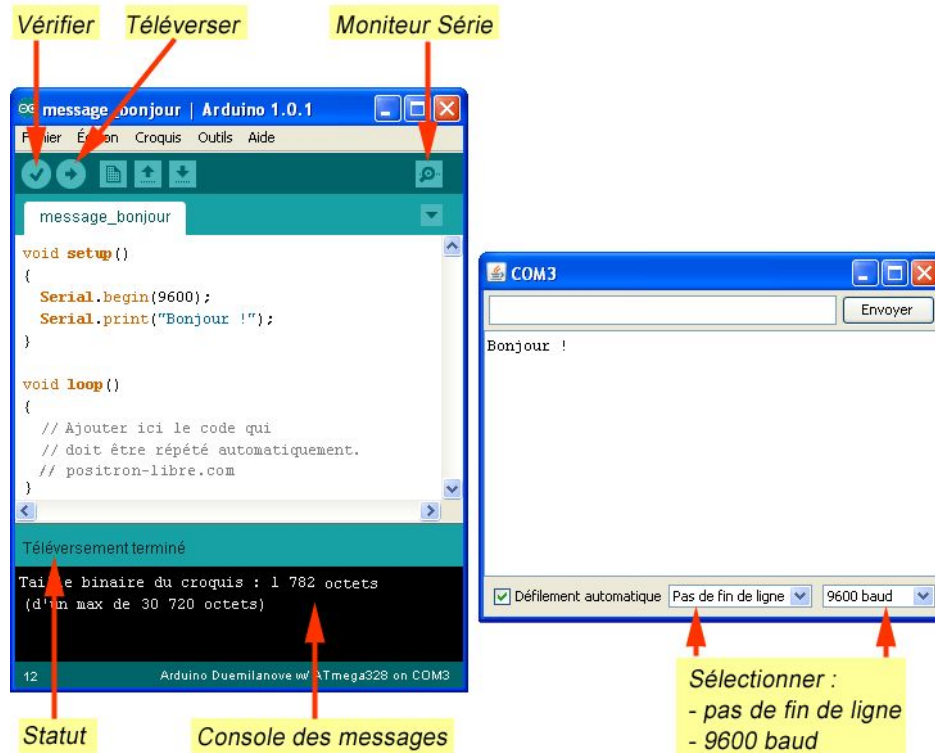
### Langage de programmation

- 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

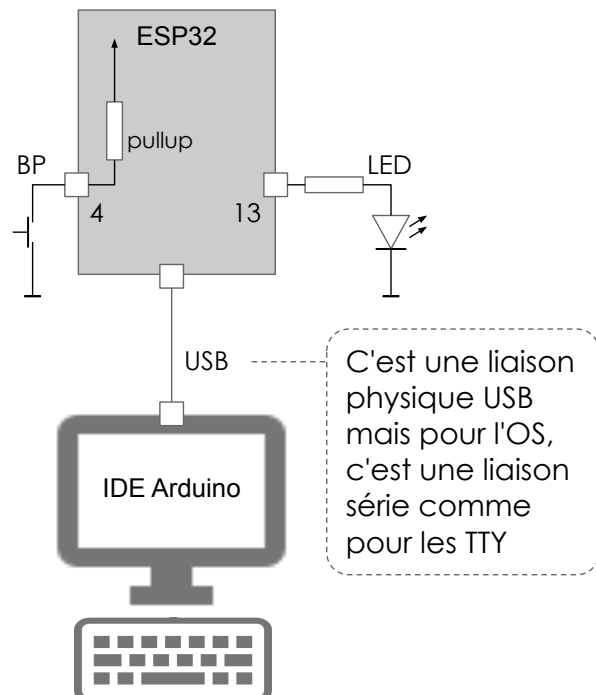
## Exemple de programme

```
// 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

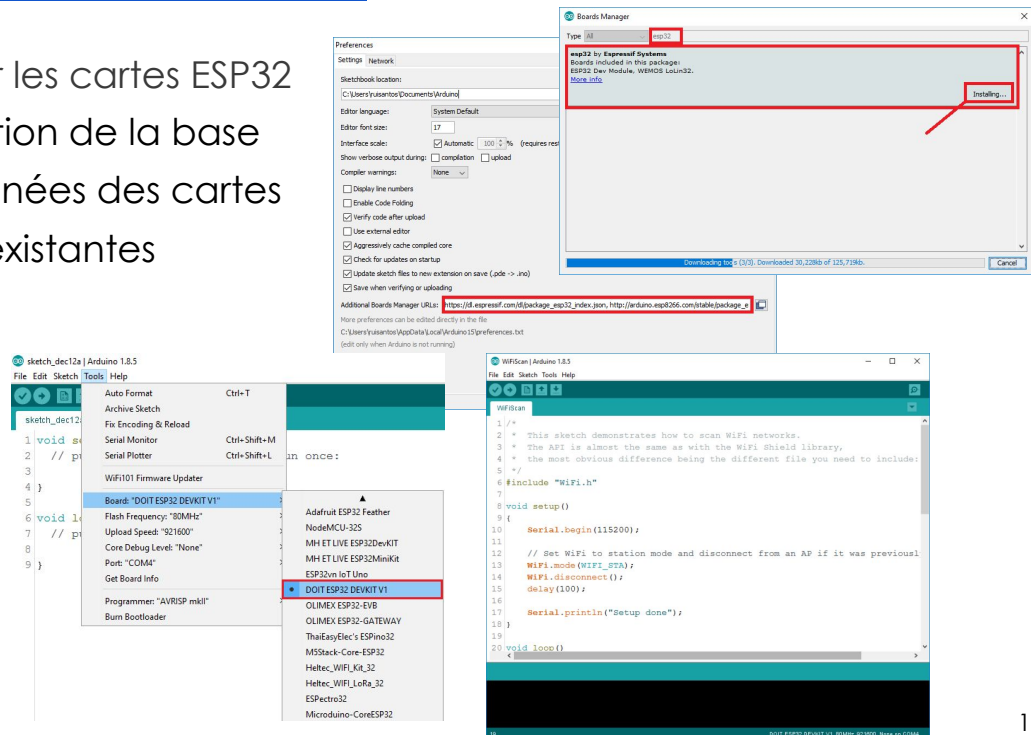


# Installation sur Arduino

<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)
```

### Constants

```
HIGH | LOW
INPUT | OUTPUT
true | false
143 // Decimal number
0173 // Octal number
0b1011111 // Binary
0x7B // Hex number
7U // Force unsigned
10L // Force long
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};
```

### Conversion

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

### Qualifiers

```
static // persists between calls
volatile // use RAM (nice for ISR)
const // make read-only
PROGMEM // 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,bit) bitWrite(x,bit,bit)
bitSet(x,bit) bitClear(x,bit)
bit(bin) //bin: 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(address, 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)
```

	ATmega168	ATmega328	ATmega1280
Flash (2k for bootloader)	16kB	32kB	128kB
SRAM	1kB	2kB	8kB
EEPROM	512B	1kB	4kB

	Duino/Arduino Pro Mini	Mega
# of I/O	14 + 6 analog (Nano has 14+8)	54 + 16 analog
Serial Pins	0 - RX 1 - TX	0 - RX1 1 - TX1 18 - RX2 18 - TX2 17 - RX3 18 - TX3 15 - RX4 14 - TX4
Ext Interrupts	2 - (int 0) 3 - (int 1)	2,3,21,20,19,18 (IRQ0- IRQ6)
PWM pins	2,6 - Timer 0 3,11 - Timer 1 10 - SS 11 - MOSI 12 - MISO 13 - SCK Analog+ SDA Analog- SCL	53 - SS 51 - MOSI 50 - MISO 52 - SCK 20 - SDA 21 - SCL

### ATtiny2313

RESET RST  
RXD0 RXD1  
TXD0 TXD1  
VCC  
GND

### ATmega48/88/168/328 Arduino

D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12 D13 D14 D15 D16 D17 D18 D19 D20 D21 D22 D23 D24 D25 D26 D27 D28 D29 D30 D31 D32 D33 D34 D35 D36 D37 D38 D39 D40 D41 D42 D43 D44 D45 D46 D47 D48 D49 D50 D51 D52 D53 D54 D55 D56 D57 D58 D59 D60 D61 D62 D63 D64 D65 D66 D67 D68 D69 D70 D71 D72 D73 D74 D75 D76 D77 D78 D79 D80 D81 D82 D83 D84 D85 D86 D87 D88 D89 D90 D91 D92 D93 D94 D95 D96 D97 D98 D99 D100 D101 D102 D103 D104 D105 D106 D107 D108 D109 D110 D111 D112 D113 D114 D115 D116 D117 D118 D119 D120 D121 D122 D123 D124 D125 D126 D127 D128 D129 D130 D131 D132 D133 D134 D135 D136 D137 D138 D139 D140 D141 D142 D143 D144 D145 D146 D147 D148 D149 D150 D151 D152 D153 D154 D155 D156 D157 D158 D159 D160 D161 D162 D163 D164 D165 D166 D167 D168 D169 D170 D171 D172 D173 D174 D175 D176 D177 D178 D179 D180 D181 D182 D183 D184 D185 D186 D187 D188 D189 D190 D191 D192 D193 D194 D195 D196 D197 D198 D199 D200 D201 D202 D203 D204 D205 D206 D207 D208 D209 D210 D211 D212 D213 D214 D215 D216 D217 D218 D219 D220 D221 D222 D223 D224 D225 D226 D227 D228 D229 D230 D231 D232 D233 D234 D235 D236 D237 D238 D239 D240 D241 D242 D243 D244 D245 D246 D247 D248 D249 D250 D251 D252 D253 D254 D255 D256 D257 D258 D259 D260 D261 D262 D263 D264 D265 D266 D267 D268 D269 D270 D271 D272 D273 D274 D275 D276 D277 D278 D279 D280 D281 D282 D283 D284 D285 D286 D287 D288 D289 D290 D291 D292 D293 D294 D295 D296 D297 D298 D299 D300 D301 D302 D303 D304 D305 D306 D307 D308 D309 D310 D311 D312 D313 D314 D315 D316 D317 D318 D319 D320 D321 D322 D323 D324 D325 D326 D327 D328 D329 D330 D331 D332 D333 D334 D335 D336 D337 D338 D339 D340 D341 D342 D343 D344 D345 D346 D347 D348 D349 D350 D351 D352 D353 D354 D355 D356 D357 D358 D359 D360 D361 D362 D363 D364 D365 D366 D367 D368 D369 D370 D371 D372 D373 D374 D375 D376 D377 D378 D379 D380 D381 D382 D383 D384 D385 D386 D387 D388 D389 D390 D391 D392 D393 D394 D395 D396 D397 D398 D399 D400 D401 D402 D403 D404 D405 D406 D407 D408 D409 D410 D411 D412 D413 D414 D415 D416 D417 D418 D419 D420 D421 D422 D423 D424 D425 D426 D427 D428 D429 D430 D431 D432 D433 D434 D435 D436 D437 D438 D439 D440 D441 D442 D443 D444 D445 D446 D447 D448 D449 D450 D451 D452 D453 D454 D455 D456 D457 D458 D459 D460 D461 D462 D463 D464 D465 D466 D467 D468 D469 D470 D471 D472 D473 D474 D475 D476 D477 D478 D479 D480 D481 D482 D483 D484 D485 D486 D487 D488 D489 D490 D491 D492 D493 D494 D495 D496 D497 D498 D499 D500 D501 D502 D503 D504 D505 D506 D507 D508 D509 D510 D511 D512 D513 D514 D515 D516 D517 D518 D519 D520 D521 D522 D523 D524 D525 D526 D527 D528 D529 D530 D531 D532 D533 D534 D535 D536 D537 D538 D539 D540 D541 D542 D543 D544 D545 D546 D547 D548 D549 D550 D551 D552 D553 D554 D555 D556 D557 D558 D559 D560 D561 D562 D563 D564 D565 D566 D567 D568 D569 D570 D571 D572 D573 D574 D575 D576 D577 D578 D579 D580 D581 D582 D583 D584 D585 D586 D587 D588 D589 D590 D591 D592 D593 D594 D595 D596 D597 D598 D599 D600 D601 D602 D603 D604 D605 D606 D607 D608 D609 D610 D611 D612 D613 D614 D615 D616 D617 D618 D619 D620 D621 D622 D623 D624 D625 D626 D627 D628 D629 D630 D631 D632 D633 D634 D635 D636 D637 D638 D639 D640 D641 D642 D643 D644 D645 D646 D647 D648 D649 D650 D651 D652 D653 D654 D655 D656 D657 D658 D659 D660 D661 D662 D663 D664 D665 D666 D667 D668 D669 D670 D671 D672 D673 D674 D675 D676 D677 D678 D679 D680 D681 D682 D683 D684 D685 D686 D687 D688 D689 D690 D691 D692 D693 D694 D695 D696 D697 D698 D699 D700 D701 D702 D703 D704 D705 D706 D707 D708 D709 D710 D711 D712 D713 D714 D715 D716 D717 D718 D719 D720 D721 D722 D723 D724 D725 D726 D727 D728 D729 D730 D731 D732 D733 D734 D735 D736 D737 D738 D739 D740 D741 D742 D743 D744 D745 D746 D747 D748 D749 D750 D751 D752 D753 D754 D755 D756 D757 D758 D759 D760 D761 D762 D763 D764 D765 D766 D767 D768 D769 D770 D771 D772 D773 D774 D775 D776 D777 D778 D779 D780 D781 D782 D783 D784 D785 D786 D787 D788 D789 D790 D791 D792 D793 D794 D795 D796 D797 D798 D799 D800 D801 D802 D803 D804 D805 D806 D807 D808 D809 D810 D811 D812 D813 D814 D815 D816 D817 D818 D819 D820 D821 D822 D823 D824 D825 D826 D827 D828 D829 D830 D831 D832 D833 D834 D835 D836 D837 D838 D839 D840 D841 D842 D843 D844 D845 D846 D847 D848 D849 D850 D851 D852 D853 D854 D855 D856 D857 D858 D859 D860 D861 D862 D863 D864 D865 D866 D867 D868 D869 D870 D871 D872 D873 D874 D875 D876 D877 D878 D879 D880 D881 D882 D883 D884 D885 D886 D887 D888 D889 D890 D891 D892 D893 D894 D895 D896 D897 D898 D899 D900 D901 D902 D903 D904 D905 D906 D907 D908 D909 D910 D911 D912 D913 D914 D915 D916 D917 D918 D919 D920 D921 D922 D923 D924 D925 D926 D927 D928 D929 D930 D931 D932 D933 D934 D935 D936 D937 D938 D939 D940 D941 D942 D943 D944 D945 D946 D947 D948 D949 D950 D951 D952 D953 D954 D955 D956 D957 D958 D959 D960 D961 D962 D963 D964 D965 D966 D967 D968 D969 D970 D971 D972 D973 D974 D975 D976 D977 D978 D979 D980 D981 D982 D983 D984 D985 D986 D987 D988 D989 D990 D991 D992 D993 D994 D995 D996 D997 D998 D999

### ATmega48/88/168/328 Arduino

D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12 D13 D14 D15 D16 D17 D18 D19 D20 D21 D22 D23 D24 D25 D26 D27 D28 D29 D30 D31 D32 D33 D34 D35 D36 D37 D38 D39 D40 D41 D42 D43 D44 D45 D46 D47 D48 D49 D50 D51 D52 D53 D54 D55 D56 D57 D58 D59 D60 D61 D62 D63 D64 D65 D66 D67 D68 D69 D70 D71 D72 D73 D74 D75 D76 D77 D78 D79 D80 D81 D82 D83 D84 D85 D86 D87 D88 D89 D90 D91 D92 D93 D94 D95 D96 D97 D98 D99 D100 D101 D102 D103 D104 D105 D106 D107 D108 D109 D110 D111 D112 D113 D114 D115 D116 D117 D118 D119 D120 D121 D122 D123 D124 D125 D126 D127 D128 D129 D130 D131 D132 D133 D134 D135 D136 D137 D138 D139 D140 D141 D142 D143 D144 D145 D146 D147 D148 D149 D150 D151 D152 D153 D154 D155 D156 D157 D158 D159 D160 D161 D162 D163 D164 D165 D166 D167 D168 D169 D170 D171 D172 D173 D174 D175 D176 D177 D178 D179 D180 D181 D182 D183 D184 D185 D186 D187 D188 D189 D190 D191 D192 D193 D194 D195 D196 D197 D198 D199 D200 D201 D202 D203 D204 D205 D206 D207 D208 D209 D210 D211 D212 D213 D214 D215 D216 D217 D218 D219 D220 D221 D222 D223 D224 D225 D226 D227 D228 D229 D230 D231 D232 D233 D234 D235 D236 D237 D238 D239 D240 D241 D242 D243 D244 D245 D246 D247 D248 D249 D250 D251 D252 D253 D254 D255 D256 D257 D258 D259 D260 D261 D262 D263 D264 D265 D266 D267 D268 D269 D270 D271 D272 D273 D274 D275 D276 D277 D278 D279 D280 D281 D282 D283 D284 D285 D286 D287 D288 D289 D290 D291 D292 D293 D294 D295 D296 D297 D298 D299 D300 D301 D302 D303 D304 D305 D306 D307 D308 D309 D310 D311 D312 D313 D314 D315 D316 D317 D318 D319 D320 D321 D322 D323 D324 D325 D326 D327 D328 D329 D330 D331 D332 D333 D334 D335 D336 D337 D338 D339 D340 D341 D342 D343 D344 D345 D346 D347 D348 D349 D350 D351 D352 D353 D354 D355 D356 D357 D358 D359 D360 D361 D362 D363 D364 D365 D366 D367 D368 D369 D370 D371 D372 D373 D374 D375 D376 D377 D378 D379 D380 D381 D382 D383 D384 D385 D386 D387 D388 D389 D390 D391 D392 D393 D394 D395 D396 D397 D398 D399 D400 D401 D402 D403 D404 D405 D406 D407 D408 D409 D410 D411 D412 D413 D414 D415 D416 D417 D418 D419 D420 D421 D422 D423 D424 D425 D426 D427 D428 D429 D430 D431 D432 D433 D434 D435 D436 D437 D438 D439 D440 D441 D442 D443 D444 D445 D446 D447 D448 D449 D450 D451 D452 D453 D454 D455 D456 D457 D458 D459 D460 D461 D462 D463 D464 D465 D466 D467 D468 D469 D470 D471 D472 D473 D474 D475 D476 D477 D478 D479 D480 D481 D482 D483 D484 D485 D486 D487 D488 D489 D490 D491 D492 D493 D494 D495 D496 D497 D498 D499 D500 D501 D502 D503 D504 D505 D506 D507 D508 D509 D510 D511 D512 D513 D514 D515 D516 D517 D518 D519 D520 D521 D522 D523 D524 D525 D526 D527 D528 D529 D530 D531 D532 D533 D534 D535 D536 D537 D538 D539 D540 D541 D542 D543 D544 D545 D546 D547 D548 D549 D550 D551 D552 D553 D554 D555 D556 D557 D558 D559 D560 D561 D562 D563 D564 D565 D566 D567 D568 D569 D570 D571 D572 D573 D574 D575 D576 D577 D578 D579 D580 D581 D582 D583 D584 D585 D586 D587 D588 D589 D590 D591 D592 D593 D594 D595 D596 D597 D598 D599 D600 D601 D602 D603 D604 D605 D606 D607 D608 D609 D610 D611 D612 D613 D614 D615 D616 D617 D618 D619 D620 D621 D622 D623 D624 D625 D626 D627 D628 D629 D630 D631 D632 D633 D634 D635 D636 D637 D638 D639 D640 D641 D642 D643 D644 D645 D646 D647 D648 D649 D650 D651 D652 D653 D654 D655 D656 D657 D658 D659 D660 D661 D662 D663 D664 D665 D666 D667 D668 D669 D670 D671 D672 D673 D674 D675 D676 D677 D678 D679 D680 D681 D682 D683 D684 D685 D686 D687 D688 D689 D690 D691 D692 D693 D694 D695 D696 D697 D698 D699 D700 D701 D702 D703 D704 D705 D706 D707 D708 D709 D710 D711 D712 D713 D714 D715 D716 D717 D718 D719 D720 D721 D722 D723 D724 D725 D726 D727 D728 D729 D730 D731 D732 D733 D734 D735 D736 D737 D738 D739 D740 D741 D742 D743 D744 D745 D746 D747 D748 D749 D750 D751 D752 D753 D754 D755 D756 D757 D758 D759 D760 D761 D762 D763 D764 D765 D766 D767 D768 D769 D770 D771 D772 D773 D774 D775 D776 D777 D778 D779 D780 D781 D782 D783 D784 D785 D786 D787 D788 D789 D790 D791 D792 D793 D794 D795 D796 D797 D798 D799 D800 D801 D802 D803 D804 D805 D806 D807 D808 D809 D810 D811 D812 D813 D814 D815 D816 D817 D818 D819 D820 D821 D822 D823 D824 D825 D826 D827 D828 D829 D830 D831 D832 D833 D834 D835 D836 D837 D838 D839 D840 D841 D842 D843 D844 D845 D846 D847 D848 D849 D850 D851 D852 D853 D854 D855 D856 D857 D858 D859 D860 D861 D862 D863 D864 D865 D866 D867 D868 D869 D870 D871 D872 D873 D874 D875 D876 D877 D878 D879 D880 D881 D882 D883 D884 D885 D886 D887 D888 D889 D890 D891 D892 D893 D894 D895 D896 D897 D898 D899 D900 D901 D902 D903 D904 D905 D906 D907 D908 D909 D910 D911 D912 D913 D914 D915 D916 D917 D918 D919 D920 D921 D922 D923 D924 D925 D926 D927 D928 D929 D930 D931 D932 D933 D934 D935 D936 D937 D938 D939 D940 D941 D942 D943 D944 D945 D946 D947 D948 D949 D950 D951 D952 D953 D954 D955 D956 D957 D958 D959 D960 D961 D962 D963 D964 D965 D966 D967 D968 D969 D970 D971 D972 D973 D974 D975 D976 D977 D978 D979 D980 D981 D982 D983 D984 D985 D986 D987 D988 D989 D990 D991 D992 D993 D994 D995 D996 D997 D998 D999

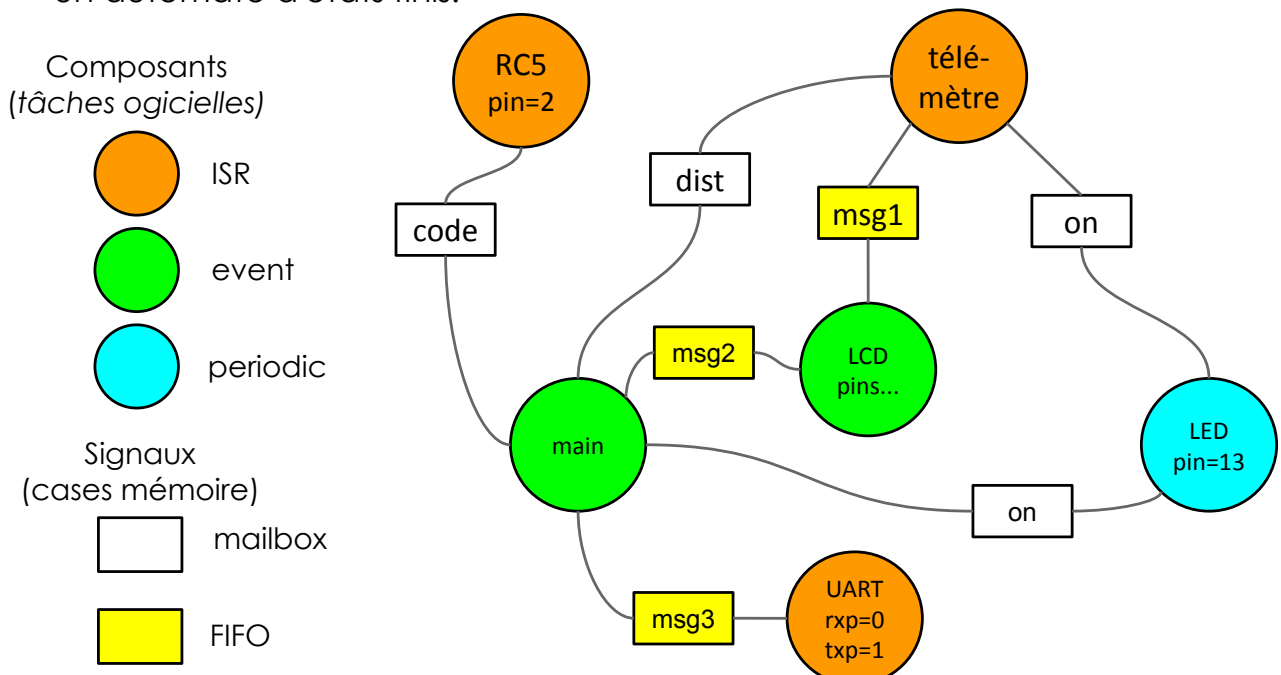
### ATmega48/88/168/328 Arduino

D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12 D13 D14 D15 D16 D17 D18 D19 D20 D21 D22 D23 D24 D25 D26 D27 D28 D29 D30 D31 D32 D33 D34 D35 D36 D37 D38 D39 D40 D41 D42 D43 D44 D45 D46 D47 D48 D49 D50 D51 D52 D53 D54 D55 D56 D57 D58 D59 D60 D61 D62 D63 D64 D65 D66 D67 D68 D69 D70 D71 D72 D73 D74 D75 D76 D77 D78 D79 D80 D81 D82 D83 D84 D85 D86 D87 D88 D89 D90 D91 D92 D93 D94 D95 D96 D97 D98 D99 D100 D101 D102 D103 D104 D105 D106 D107 D108 D109 D110 D111 D112 D113 D114 D115 D116 D117 D118 D119 D120 D121 D122 D123 D124 D125 D126 D127 D128 D129 D130 D131 D132 D133 D134 D135 D136 D137 D138 D139 D140 D141 D142 D143 D144 D145 D146 D147 D148 D149 D150 D151 D152 D153 D154 D155 D156 D157 D158 D159 D160 D161 D162 D163 D164 D165 D166 D167 D168 D169 D170 D171 D172 D173 D174 D175 D176 D177 D178 D179 D180 D181 D182 D183 D184 D185 D186 D187 D188 D189 D190 D191 D192 D193 D194 D195 D196 D197 D198 D199 D200 D201 D202 D203 D204 D205 D206 D207 D208 D209 D210 D211 D212 D213 D214 D215 D216 D217 D218 D219 D220 D221 D222 D223 D224 D225 D226 D227 D228 D229 D230 D231 D232 D233 D234 D235 D236 D237 D238 D239 D240 D241 D242 D243 D244 D245 D246 D247 D248 D249 D250 D251 D252 D253 D254 D255 D256 D257 D258 D259 D260 D261 D262 D263 D264 D265 D266 D267 D268 D269 D270 D271 D272 D273 D274 D275 D276 D277 D278 D279 D280 D281 D282 D283 D284 D285 D286 D287 D288 D289 D290 D291 D292 D293 D294 D295 D296 D297 D298 D299 D300 D301 D302 D303 D304 D305 D306 D307 D308 D309 D310 D311 D312 D313 D314 D315 D316 D317 D318 D319 D320 D321 D322 D323 D324 D325 D326 D327 D328 D329 D330 D331 D332 D333 D334 D335 D336 D337 D338 D339 D340 D341 D342 D343 D344 D345 D346 D347 D348 D349 D350 D351 D352 D353 D354 D355 D356 D357 D358 D359 D360 D361 D362 D363 D364 D365 D366 D367 D368 D369 D370 D371 D372 D373 D374 D375 D376 D377 D378 D379 D380 D381 D382 D383 D384 D385 D386 D387 D388 D389 D390 D391 D392 D393 D394 D395 D396 D397 D398 D399 D400 D401 D402 D403 D404 D405 D406 D407 D408 D409 D410 D411 D412 D41

# Programmation par automates

## Applications

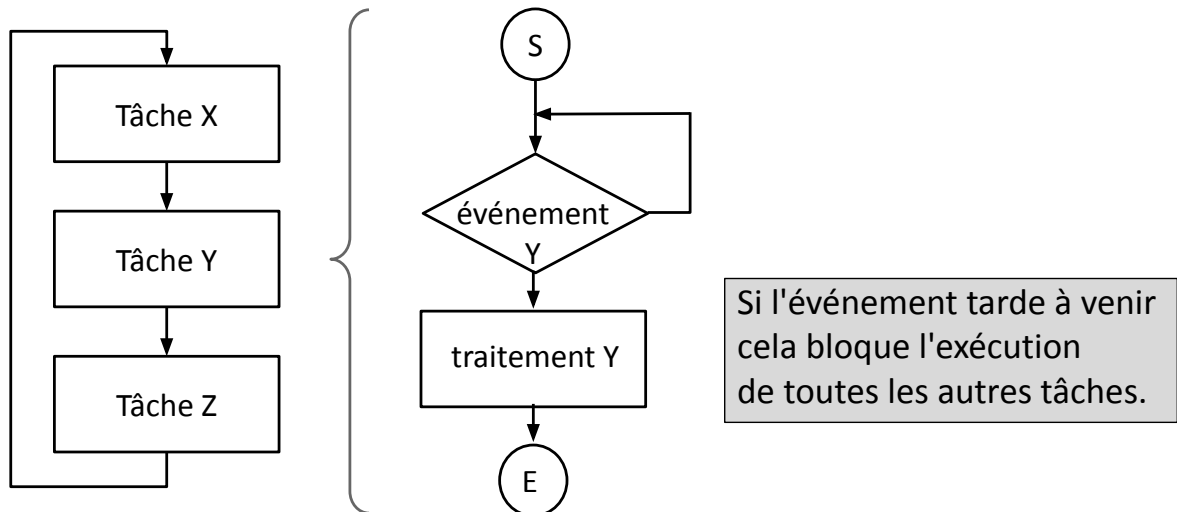
Les applications peuvent être vues comme des composants matériels communiquant par des signaux. Chaque composant étant décrit par un automate d'états finis.





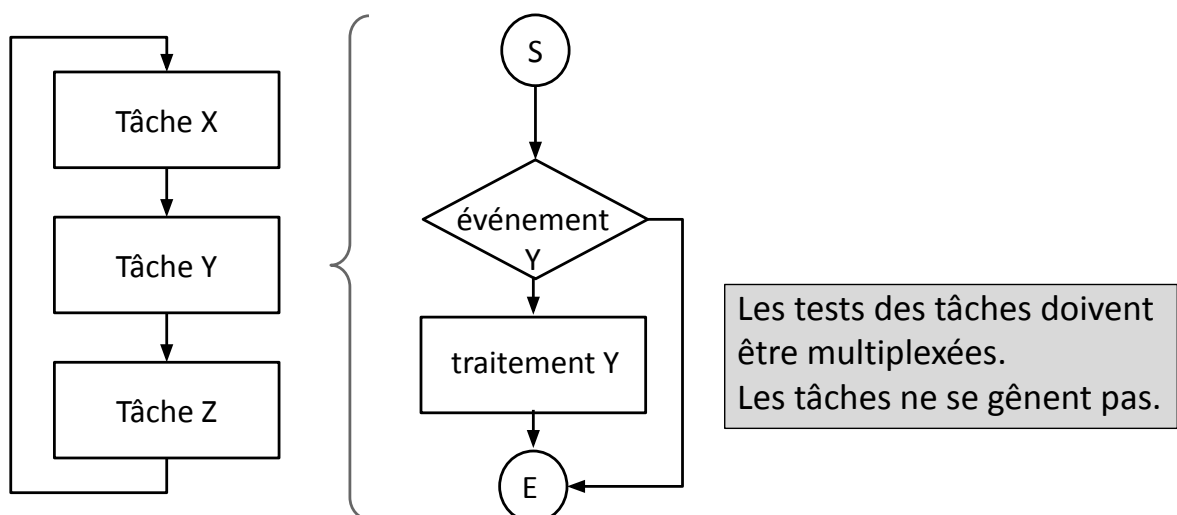
# Problème : exécution de plusieurs tâches

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



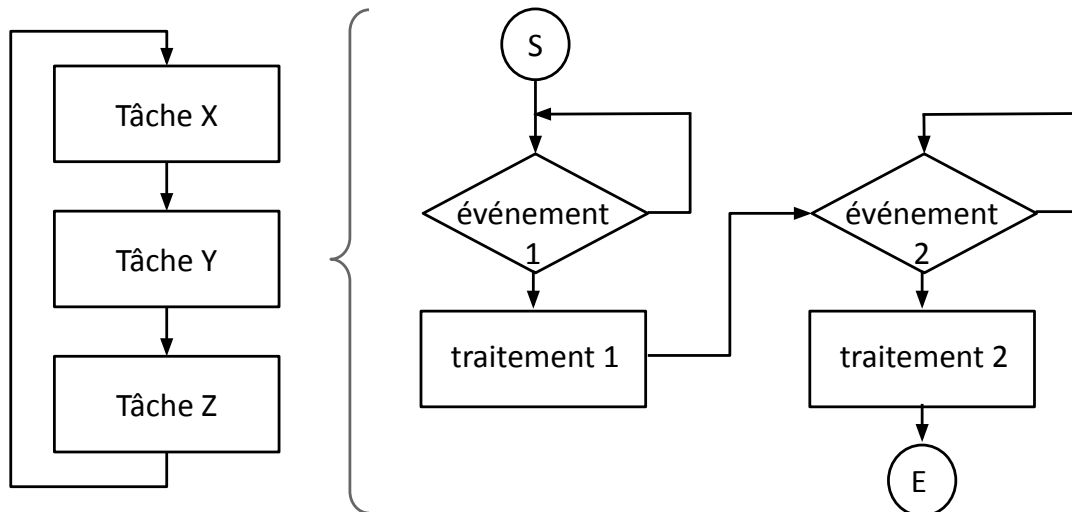
→ si les tâches coopèrent, on attend pas

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



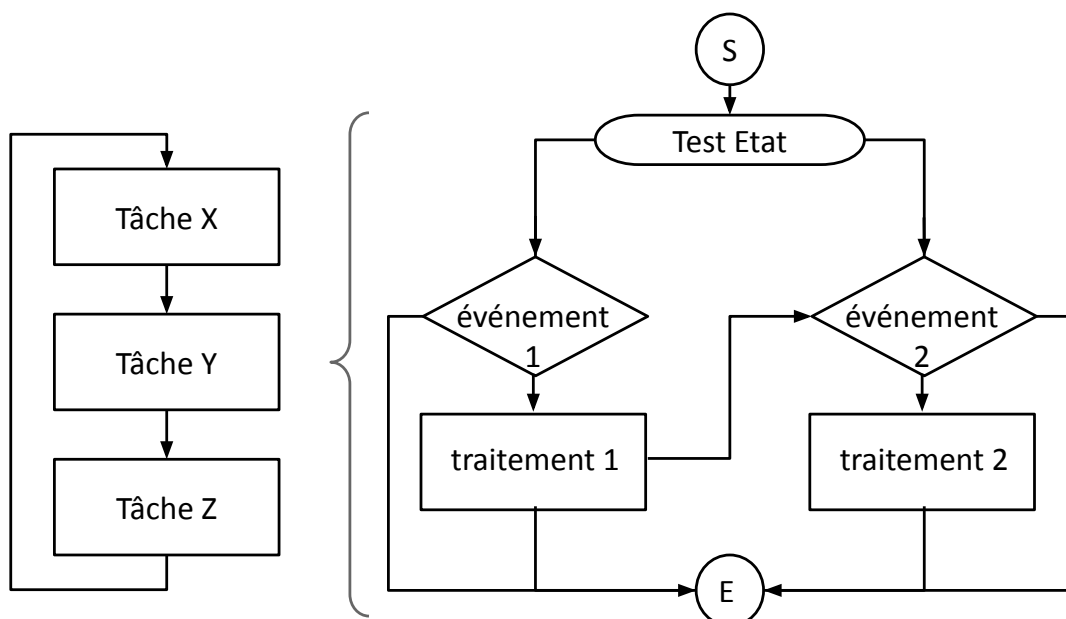
# Une tâche peut avoir plusieurs traitements

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() {
    tache_1(&x, &y, &z);
    tache_2(&x, &t);
    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 une fonction
- Cette fonction 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 `setup()`
  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 setup_tache(ctx_tache_t *ctx_tache, int ETAT) {
    ctx_tache->etat = ETAT;
    ...
}
void loop_tache(ctx_tache_t *ctx_tache, int * in_out, ...) {
    switch (ctx_tache->etat) {
        case ETAT0:
            ...
    }
}
```

```
typedef struct ctx_tache_st {
    int etat;
} ctx_t0, ctx_t1;
int x;
setup() {
    setup_t0(&ctx_t0, ETAT0);
    setup_t1(&ctx_t1, ETAT1);
}
loop() {
    loop_t0(&ctx_t0, &x);
    loop_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)

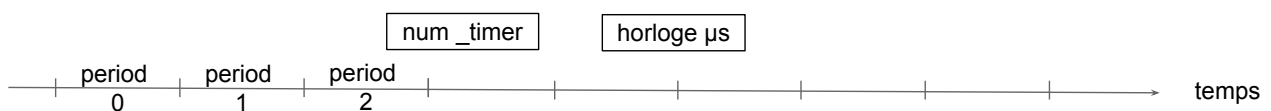
Soit P une période

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 32bits 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 microseconde
// valeur de retour :
// - nombre de périodes écoulées depuis le dernier appel (normalement c'est 1)

#define MAX_WAIT_FOR_TIMER 2
unsigned int waitFor(int timer, unsigned long period){
    static unsigned long waitForTimer[MAX_WAIT_FOR_TIMER];
    unsigned long newTime = micros() / period;           // numéro de la période modulo 2^32
    int delta = newTime - waitForTimer[timer];           // delta entre la période courante et celle enregistrée
    if ( delta < 0 ) delta += 1 + (0xFFFFFFFF / period); // en cas de dépassement du nombre de périodes possibles
    if ( delta ) waitForTimer[timer] = newTime;          // enregistrement du nouveau numéro de période
    return delta;
}
```

# Tâche ISR

- Les tâches invoquées par loop ont un ordonnancement fifo
- Arduino permet facilement de configurer des ISR  
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 = LOW;

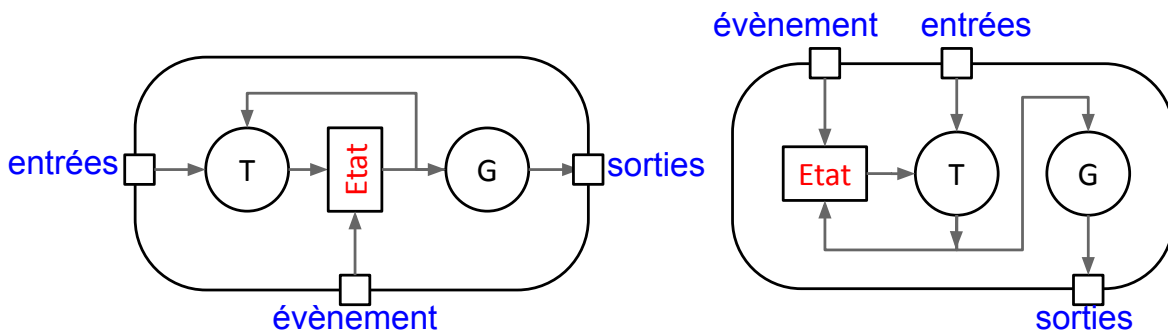
void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
}

void loop() {
  digitalWrite(ledPin, state);
}

void blink() {
  state = !state;
}
```

- ici blink est un type de tâche ISR

## Une tâche est une machine d'états finis

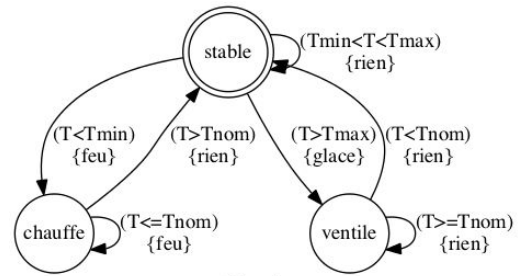


Aller à l'état courant (p.ex. E0)  
**Etat**=E0:  
 Si événement attendu  
 Alors  
   G : sorties ← traitement  
   T : en fonction des entrées  
       **Etat** ← état suivant  
 Fsi  
**Etat**=E1:  
 [...]

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:  
 [...]

# Automate en C

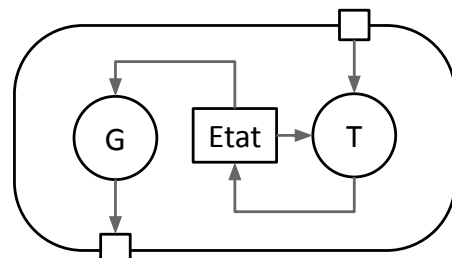
```
enum {STABLE, CHAUFFE, VENTIL} state;
void setup_clim(ctx_clim_t * ctx, int timer, int period, int *T, int pin_V, int pin_C) {
    ctx->state=STABLE; ctx->ventil=OFF; ctx->radiateur=OFF;
    ctx->timer = timer;
    ctx->period = period;
    ctx->T = T; ctx->pin_v = pin_V ; ctx->pin_c = pin_C;
}
void loop_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);
}
```



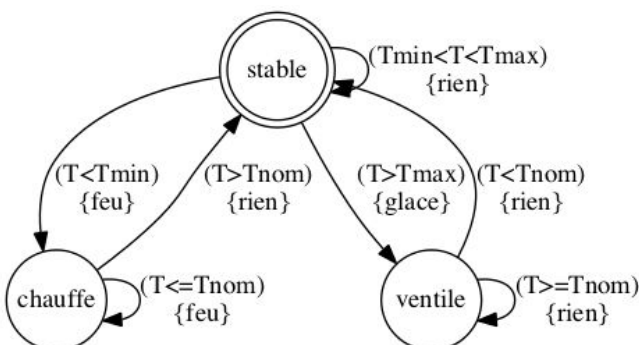
## FSM : Machine à états finis

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

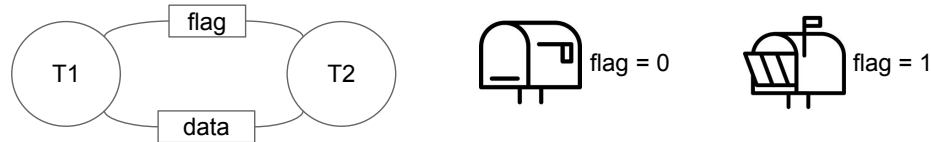
3 températures : Tmin - Tnom - Tmax (p.ex. 18° 19° 20°)  
la température T

ACTIONS

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



# 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

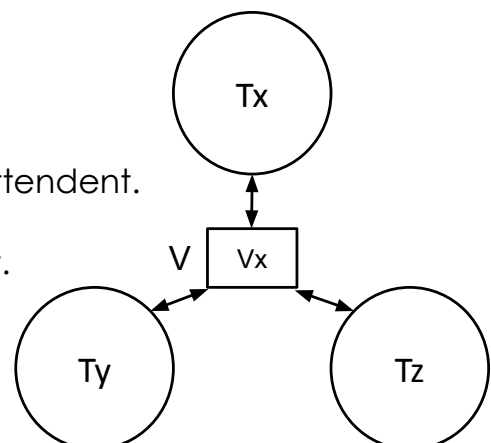
## Set-Reset Flag

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

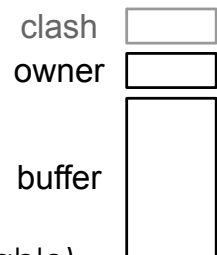
⇒ communication sans perte

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

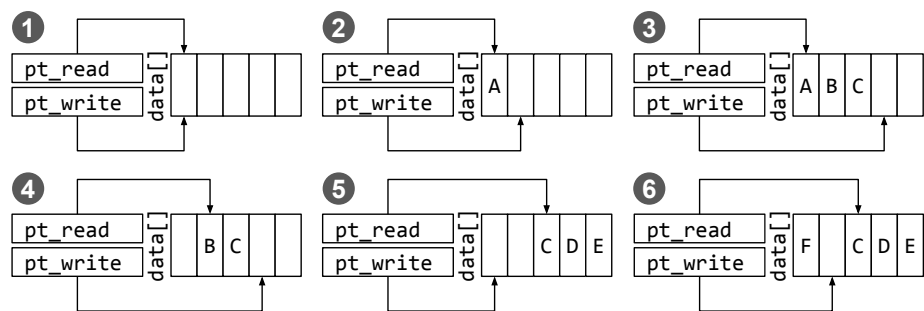
Un dernier mécanisme de communication simple est la FIFO

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;
};
```

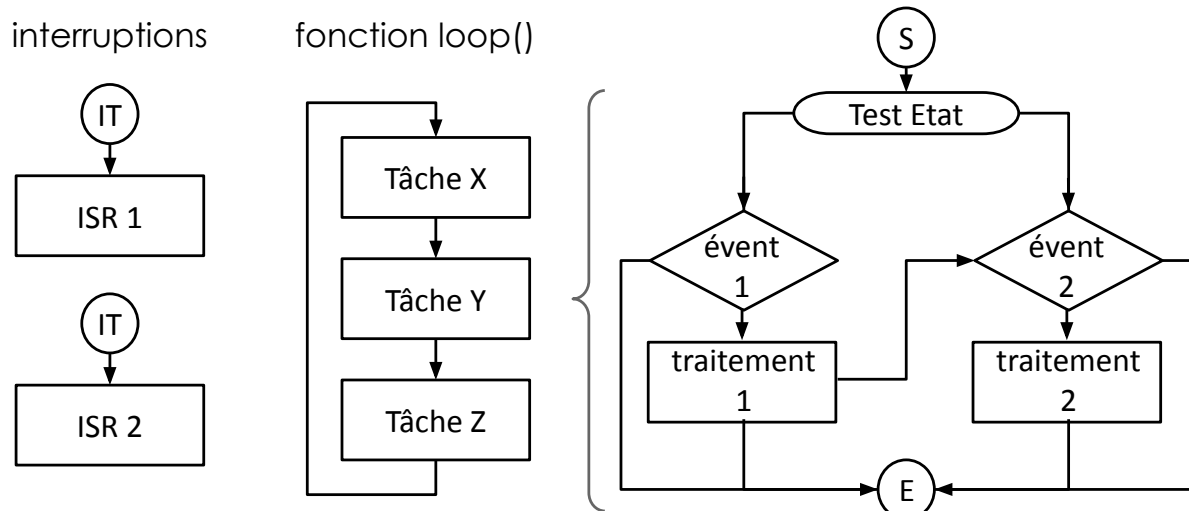
```
static int tty_fifo_push(struct tty_fifo_s *fifo, int c) {
    int pt_write_next = (fifo->pt_write + 1) % sizeof(fifo->data);
    if (pt_write_next != fifo->pt_read) {
        fifo->data[fifo->pt_write] = c;
        fifo->pt_write = pt_write_next;
        return 1;
    }
    return 0;
}
```

```
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 : setup
  - cas normal : loop 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 la led
  - 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
  - ...