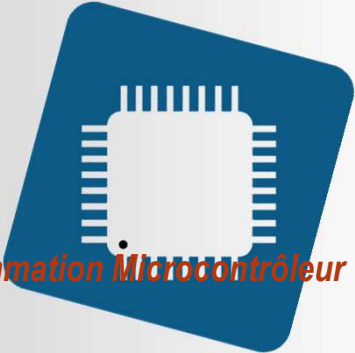


UNIVERSITÉ
La Rochelle



Programmation Microcontrôleur

Cours #03 :
Framework, environnements de développement,
debug
Entrées/Sorties de base
Bernard Besserer

1

UNIVERSITÉ
La Rochelle

Le Framework Arduino

- *Wiring is an open-source programming framework for microcontrollers.*
- Initialement pour ATMEL AVR (platine Arduino), mais :
 - Portage pour microprocesseur Texas Instruments MSP430 (projet Energia)
 - Portage pour ESP8266 et ESP32
- Le framework utilise un C++ épuré de beaucoup de ses fonctionnalités ; les outils sont des versions adaptés et modifiés de GCC, pilotés par un IDE (*Integrated Development Environment*) spécifique

2

IDE Arduino

- L'IDE permet
 - Edition
 - Gestion des plateformes
 - Gestion des bibliothèques
 - Accès à des exemples
 - MAJ automatiques
- Community-driven !
- Compilation et téléversement, mais pas de Debug !

Seule une liaison série peut rester active (terminal)

3

Développement avec les outils GCC en ligne de commande

- Compile
- Link
- Convert .elf to .hex
- Download to Device
- .elf: Executable and Linking Format, segment and section information

4

Le Framework Arduino

- Un programme se nomme "sketch" (.ino)
- Le main() est masqué, un sketch expose deux fonctions :
 - void setup(): appelé initialement, ne s'exécute qu'une fois.
 - void loop(): boucle sans fin (équivalent à while(1) { })
 - Interrompu par un reset, power-down ou bootloader

```

1 void setup() {
2
3 }
4
5 void loop() {
6
7   //Do first...
8   //Do this next...
9   //Do this too...
10
11 }
  
```

C'est la structure générale d'un programme sur microcontrôleur (+ déclaration des variables et constantes)

5

Debug avec l'IDE Arduino

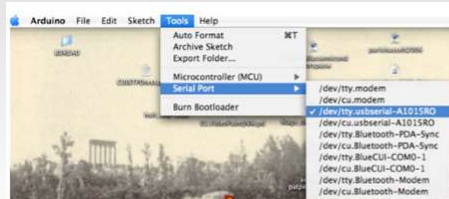
- Ni point d'arrêt, ni « pas à pas », ni inspection des variables... Il n'est même pas possible d'interrompre un programme !

Well, you cannot actually issue a HALT command because there is no such thing for Arduino. Once entered, the void loop() function runs endlessly.

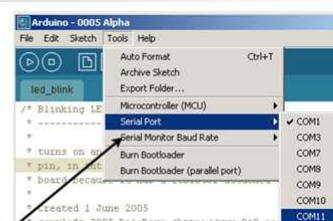
- DEBUG :
Méthode
« ancestrale »
des printf !

```

Serial.begin(9600);
Serial.print("item x=");
Serial.print(x);
Serial.print(", item y=");
Serial.println(y);
  
```



Sur MacOSX

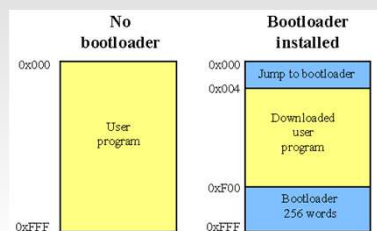


Sur Windows

6

Transférer le programme vers la cible

- Compilation croisée. Il faut une liaison entre plateforme de développement et cible
- **Solution 0** : Le programme est dans une mémoire amovible !
 - Ex : carte mémoire SD du raspberry pi
 - Mais, la mémoire Flash est intégrée dans le microcontrôleur : En général, liaison série (USB) filaire (ce qui « mobilise » 2 broches au minimum). Sur les microcontrôleurs IoT : liaison sans fil
- **Solution 1** : développer un bootloader (programme d'amorçage) qui écoute sur un port série (ou ethernet ?) et qui permet de transférer le code dans la mémoire Flash
- **Solution 1bis** : Mettre un interpréteur sur le microcontrôleur
- **Solution 1ter** : Mettre un OS embarqué

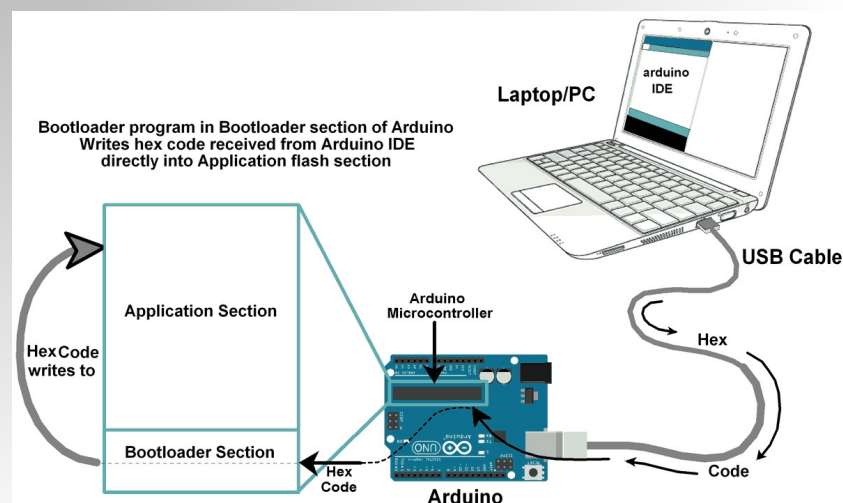


Un interpréteur ou un OS prend aussi beaucoup de place en mémoire

Tout comme les fonctions mises à disposition par un framework !

7

Solution Bootloader ARDUINO



8

Solution Bootloader

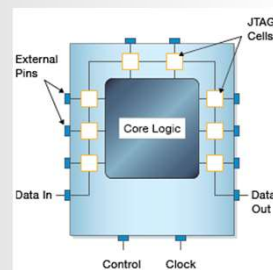
- Comment télécharger et comment communiquer (plus précisément dans la phase de développement -> DEBUG)
- Il faut consacrer au minimum 2 broches à la communication !
 - 1^{er} solution : développer un bootloader qui écoute sur un port serie (ou ethernet ?) et qui permet de transférer le code dans la Flash
 - Il faut livrer le microcontrôleur déjà programmé
 - Solution Arduino
 - 2eme solution : Avoir une norme respectée par tous les constructeurs permettant de dialoguer à tout moment avec le microcontrôleur

9

JTAG, or IEEE Standard 1149.1

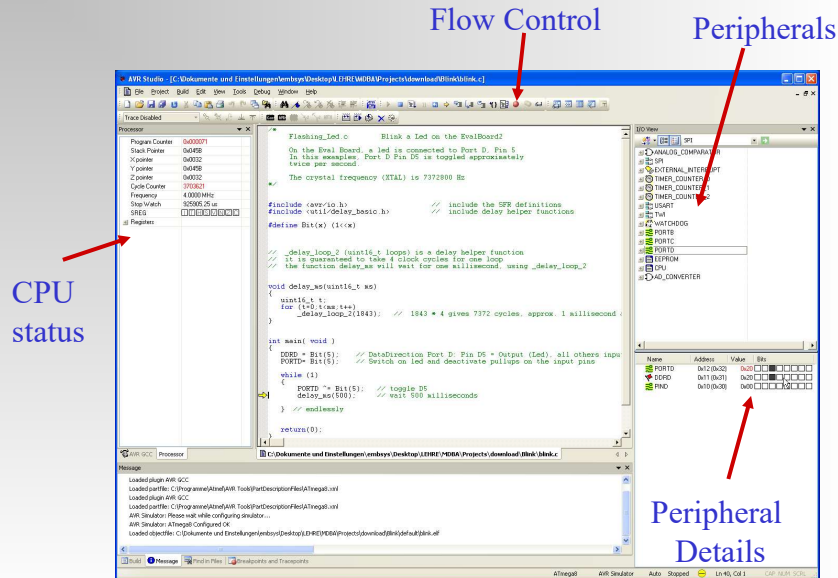


- Permet la programmation (ISP)
- Permet d'établir un dialogue avec le microcontrôleur et de Controller celui-ci :
 - Points d'arrêt
 - Arrêt de l'horloge, pas à pas
 - R/W des registres internes,
 - R/W des mémoires internes et externes
- Permet le *boundary scan*
 - Chaque broche d'entrée-sortie n'est pas connectée directement au core mais à travers une « cellule JTAG »
 - Lecture des niveaux electriques même si packaging BGA
- Pendant l'exécution normale d'un programme (hors mode DEBUG), les cellules JTAG sont non actives et « transparentes »
- L'utilisation des fonctionnalités JTAG nécessite 4 signaux (Bus série synchrone).



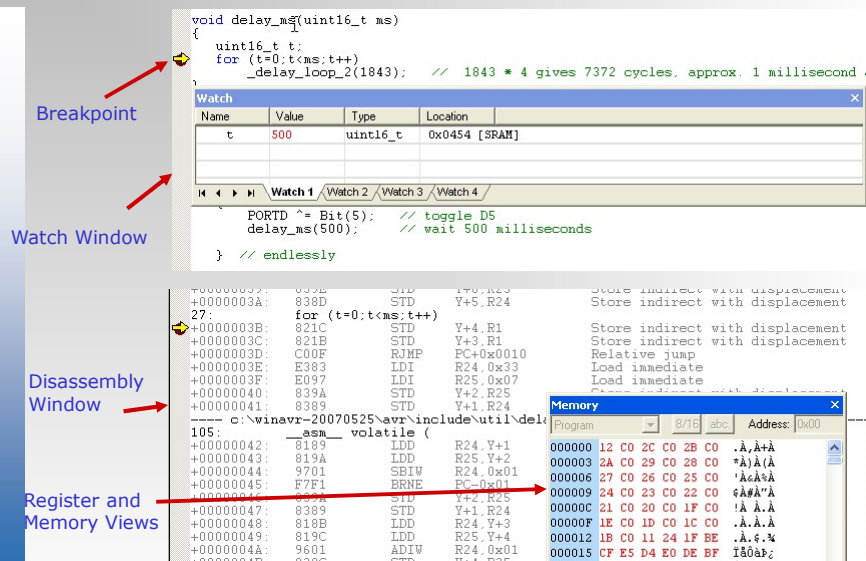
10

Si liaison JTAG -> IDE modernes




11

AVR Studio... Debug




12



Autre ajouts du framework

- Nombreuses fonctions spécifiques microcontrôleur (I/O centric) : pour garantir une portabilité, les accès directs aux registres sont masqués.
- On sélectionne une plateforme -> accès aux fonctions intrinsèques pour cette plateforme
- Il manque des fonctionnalités (intégration GitHub par exemple)
- Visual Studio Code + PlatformIO :
PlatformIO is a cross-platform, cross-architecture, multiple framework, professional tool for embedded systems engineers and for software developers who write applications for embedded products

13



Fonctions élémentaires du framework Arduino

- Entrées-sorties numériques
 - `pinMode(broche, état)` (configuration des broches)
 - `digitalWrite(broche, état)` (écrire un état sur une broche)
 - `digitalRead(broche)` (lire un état sur une broche)
 - `unsigned long pulseIn(broche, état)` (lire / compter impulsion(s) sur une broche num.)
- Entrées analogiques
 - `int analogRead(broche)` (lire la valeur d'une broche ana.)
 - `analogWrite(broche, valeur)` (PWM : écrire une valeur analogique sur les broches 9, 10 ou 11)
- Gestion du temps
 - `unsigned long millis()` (temps de fonctionnement)
 - `delay(ms)` (attente, en millisecondes)
 - `delayMicroseconds(us)` (attente, en microsecondes)
- Liaison Série : Objet Serial
 - `Serial.begin()`, `Serial.print()`, `Serial.println()`, `Serial.read()`

14

Exemple de programme basique

- Faire clignoter une LED (blink)
- Sans framework : le point d'entrée du programme c'est la fonction main()
- Deux parties : configuration (setup) et boucle d'exécution

```
#include <msp430x14x.h>

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
    P2DIR |= 0x02; // Set P2.1 to output direction
    for (;;)
    {
        unsigned int i;
        P2OUT ^= 0x02; // Toggle P2.1 using exclusive-OR
        i = 50000; // Delay
        do
            i--;
        while (i != 0);
    }
}
```

Les
initialisations
ne se font
qu'une fois !!!!

15

Code avec AVR studio (sans framework Arduino)

```
#include <avr/io.h> // Include device specific definitions (SFRs,...)
#include <util/delay_basic.h>

#define Bit(x) (1<<x) // Macro definition, shift left operation

void delay_ms(uint16_t ms) // Function definition, call by value
{
    uint16_t t;
    for (t=0;t<ms;t++)
        _delay_loop_2(1843);
}

int main( void )
{
    DDRD = Bit(5); // SFRs for Port D configuration // DataDirection Port D: Pin D5 = Output (Led)
    PORTD= Bit(5); // Switch on led and deactivate pullups on the inputs

    while (1)
    {
        PORTD ^= Bit(5); // Bit manipulation Port D (XOR) // toggle D5
        delay_ms(500); // wait 500 milliseconds
    }
    return(0);
}
```

16

Exemple de programme basique

■ Avec le framework Arduino

```
const int ledPin = 13; // LED connected to digital pin 13
#define ledPin 13      // autre ecriture
                      // ou utilisation de LED_BUILTIN

void setup() {
  pinMode(ledPin, OUTPUT); // sets the digital pin 13 as
}

void loop() {
  int state;
  state = digitalRead(ledPin); // read the LED state
  digitalWrite(ledPin, !state); // toggle LED
  delay(500);                  // wait
}
```

17

Arduino C is Derived from C++

Clignotement LED pour broche 13

■ avr-libc

```
#include <avr/io.h>
#include <util/delay.h>

int main(void) {
  while (1) {
    PORTB = 0x20;
    _delay_ms(1000);
    PORTB = 0x00;
    _delay_ms(1000);
  }
  return 1;
}
```

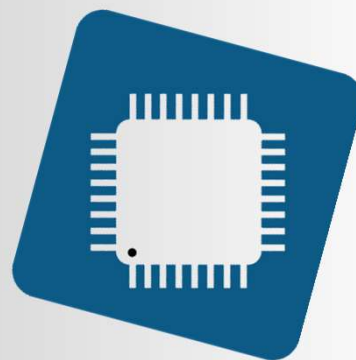
■ Arduino C

```
void setup( )
{
  pinMode(13, OUTPUT);
}

void loop( ) {
  digitalWrite(13,HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

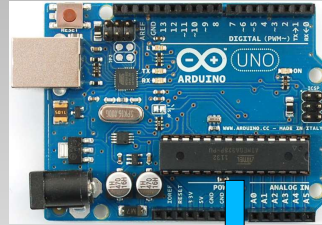
18

- **SPIFFS** (Serial Peripheral Interface Flash File System) est un système de fichiers léger adapté (entre autre) aux micro-contrôleurs disposant d'une mémoire flash SPI tel que l'ESP32 et ESP8266.
 - Attention, la mémoire flash est limitée à 10000 cycles d'écritures.
 - Actuellement, SPIFFS ne gère pas d'arborescence (flat file structure)
 - Il faut réserver de la mémoire Flash dédié à SPIFFS (en général, de 0,5 MB à 4MB)

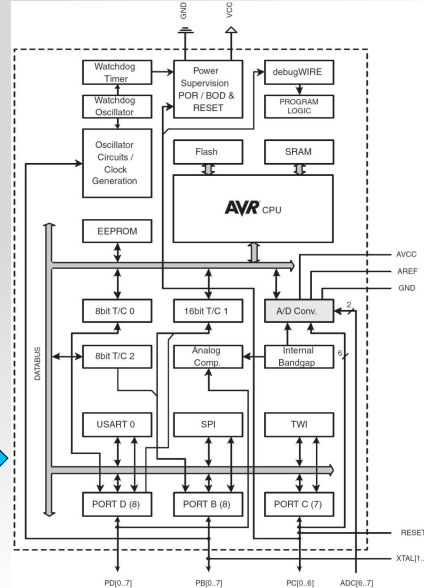


Les ENTREES / SORTIES NUMERIQUES

Architecture de la platine d'experimentation



(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)
(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)
(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)
(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)
(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)
(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)
VCC	7	22	GND
GND	8	21	AREF
(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC
(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)
(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)
(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)
(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2)
(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)




21

Brochage du microcontrôleur ATmega328

Pin name	Pin number
(PCINT14/RESET) PC6	1
(PCINT16/RXD) PD0	2
(PCINT17/TXD) PD1	3
(PCINT18/INT0) PD2	4
(PCINT19/OC2B/INT1) PD3	5
(PCINT20/XCK/T0) PD4	6
VCC	7
GND	8
(PCINT6/XTAL1/TOSC1) PB6	9
(PCINT7/XTAL2/TOSC2) PB7	10
(PCINT21/OC0B/T1) PD5	11
(PCINT22/OC0A/AIN0) PD6	12
(PCINT23/AIN1) PD7	13
(PCINT0/CLKO/ICP1) PB0	14
	15
	16
	17
	18
	19
	20
	21
	22
	23
	24
	25
	26
	27
	28

22




Entrées/sorties numeriques (digital I/O ou GPIO)

- Entrées sorties "de base"
- En general regroupés par paquets de 8 pour former un **Port**
- Ex. **PORT B**
 - Pins PB0 – PB7

- Attention, pas forcément ordonné sur la puce ou la platine
 - En general bidirectionnels et avec un multiplexage des fonctionnalités

PB6	9	20	AVC
PB7	10	19	PB5
PD5	11	18	PB4
PD6	12	17	PB3
PD7	13	16	PB2
PB0	14	15	PB1

23



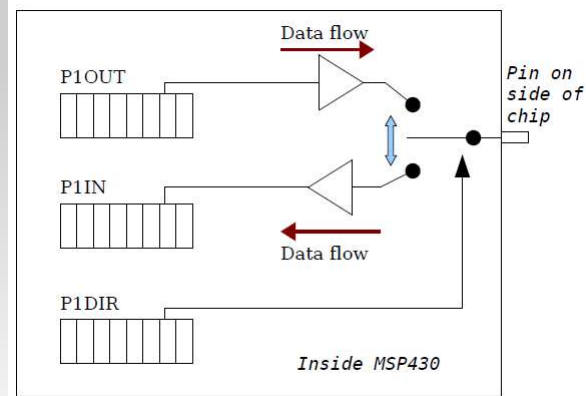
Terminologie

- Une broche / Pin / GPIO / PortX.Y est en entrée (input)
 - Quand le microcontrôleur doit récupérer des informations en provenance du monde extérieur
 - Pour saisir des informations issues de capteurs
- Une broche / Pin / GPIO / PortX.Y est en sortie (output)
 - Quand le microcontrôleur doit modifier une valeur qui sera perçu par quelque chose d'extérieur au microcontrôleur.
 - En général, ce sont des actionneurs / afficheurs.
- A la mise sous tension, tous les GPIO sont en mode « entrée ».

24

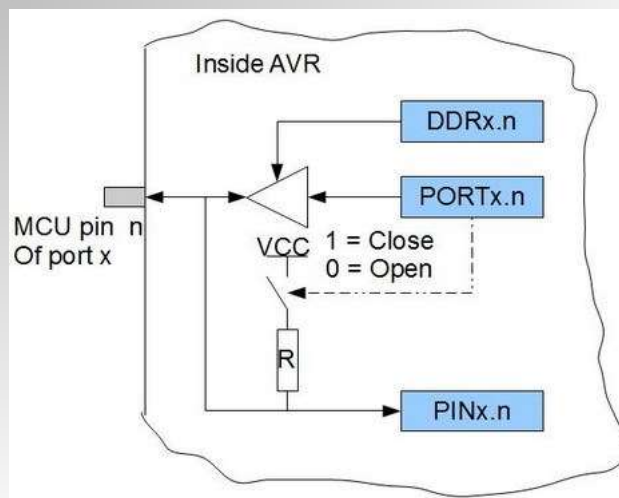
Mode GPIO

- Par défaut (mise sous tension), chaque broche est :
 - en mode GPIO : Entrée / Sortie numérique
 - En mode Entrée



On power up the GPIOs are configured as high impedance CMOS floating **inputs**.

25



26

Details interne des ports AVR

- Donc un registre spécifique pour choisir entre la fonctionnalité de base (GPIO) ou une autre fonction

Port Pin	Alternate Functions
PB0	AIN0 (Analog Comparator positive input)
PB1	AIN1 (Analog Comparator negative input)
PB3	OC1 (Timer/Counter1 Output Compare Match output)
PB5	MOSI (Data input line for memory downloading)
PB6	MISO (Data output line for memory uploading)
PB7	SCK (Serial clock input)

Port Pin	Alternate Function
PD0	RXD (Receive data input for the UART)
PD1	TXD (Transmit data output for the UART)
PD2	INT0 (External interrupt 0 input)
PD3	INT1 (External interrupt 1 input)
PD4	TO (Timer/Counter0 external input)
PD5	T1 (Timer/Counter1 external input)
PD6	ICP (Timer/Counter1 Input Capture pin)

27

Multiplexage des fonctionnalités

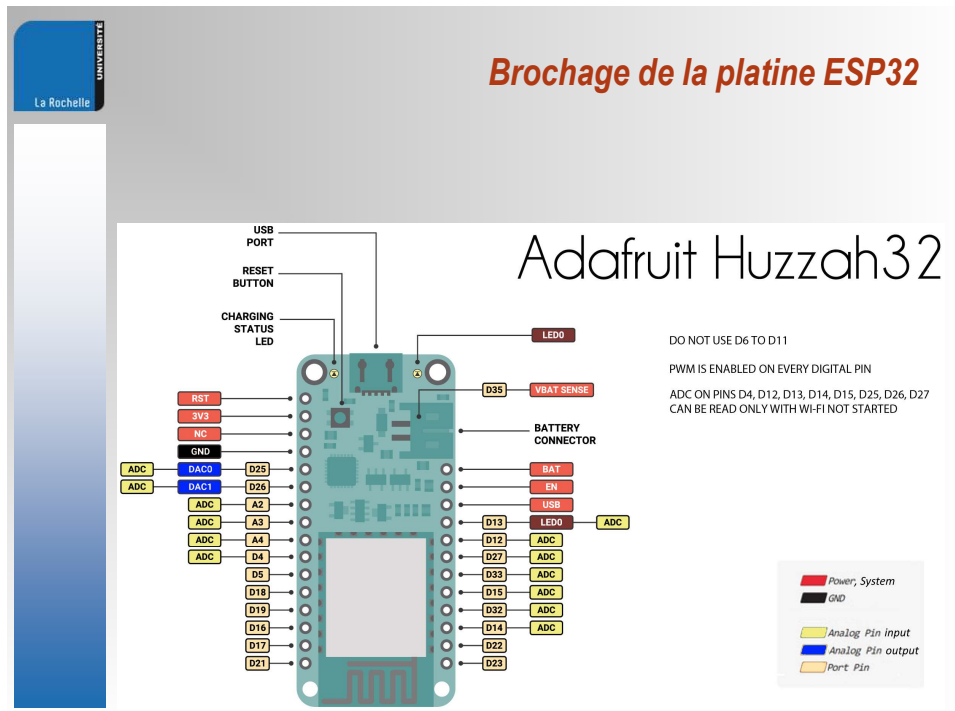
Name	No.	Type	Function
CHIP_PU	9	I	High: On; enables the chip. Low: Off; resets the chip. Note: Do not leave the CHIP_PU pin floating.
VDET_1	10	I	GPIO34, ADC1_CH6, RTC_GPIO4
VDET_2	11	I	GPIO35, ADC1_CH7, RTC_GPIO5
32K_XP	12	IO	GPIO32, 32K_XP (32.768 kHz crystal oscillator input), ADC1_CH4, TOUCH9, RTC_GPIO9
32K_XN	13	IO	GPIO33, 32K_XN (32.768 kHz crystal oscillator output), ADC1_CH5, TOUCH8, RTC_GPIO8
GPIO25	14	IO	GPIO25, DAC_1, ADC2_CH8, RTC_GPIO6, EMAC_RXD0
GPIO26	15	IO	GPIO26, DAC_2, ADC2_CH9, RTC_GPIO7, EMAC_RXD1
GPIO27	16	IO	GPIO27, ADC2_CH7, TOUCH7, RTC_GPIO17, EMAC_RX_DV
MTMS	17	IO	GPIO14, ADC2_CH6, TOUCH6, RTC_GPIO16, MTMS, HSPICLK, HS2_CLK, SD_CLK, EMAC_TXD2
MTDI	18	IO	GPIO12, ADC2_CH5, TOUCH5, RTC_GPIO15, MTDI, HSPICLK, HS2_DATA2, SD_DATA2, EMAC_TXD3
VDDSP3_RTC	19	P	Input power supply for RTC IO (2.3 V ~ 3.6 V)
MTCK	20	IO	GPIO13, ADC2_CH4, TOUCH4, RTC_GPIO14, MTCK, HSPICLK, HS2_DATA3, SD_DATA3, EMAC_RX_ER
MTDO	21	IO	GPIO15, ADC2_CH3, TOUCH3, RTC_GPIO13, MTDO, HSPICLK, HS2_CMD, SD_CMD, EMAC_RXD3

etc...

A moins de réaliser des grandes séries, l'ESP32 n'est pas acquis sous forme ce chip, mais de module (avec l'antenne WiFi) ou de circuit d'évaluation ou de prototypage



28



29

La Rochelle UNIVERSITÉ

Broche utilisée en sortie

- Fonctionnalité ON/OFF
- La broche commute entre VCC et 0V. Courant faible (LED, buzzer, ...)
- Allumer une LED connecté sur la broche 0 (PD0)
 - Configurer le port (ne pas oublier !)
`pinMode(____, ____);`
 - Allumer la LED
`digitalWrite(PIN_LED,HIGH);`
 - Eteindre la LED
`digitalWrite(PIN_LED,LOW);`
- Ne pas oublier la résistance

Arduino pin 0 (PD0)

150R

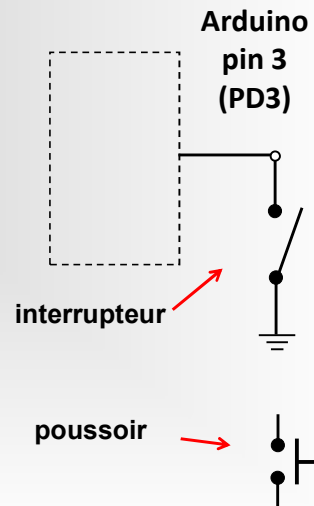
Le montage est de type « active high »

30

Broche utilisée en entrée

■ Branchement d'un interrupteur ou bouton poussoir sur la broche 3 (PD3)

- Configurer la direction `pinMode(____, ____);`
- Quel est la tension lorsque l'interrupteur est fermé ?
0V
- Quel est la tension lorsque l'interrupteur est ouvert ?
Indeterminé !

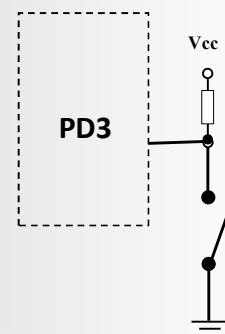


31

Ajout d'une résistance de tirage (pullup)

■ Branchement d'un interrupteur ou bouton poussoir sur la broche 3 (PD3)

- Une résistance permet de fixer le potentiel... si aucun courant (ou presque) ne la traverse.
- La resistance de tirage ne doit pas :
 - être trop faible = court-circuit
 - être trop importante = chute de tension importante même si un faible courant la traverse ($U=RI$)
 - Ordre de grandeur : une dizaine à un centaine de KOhms

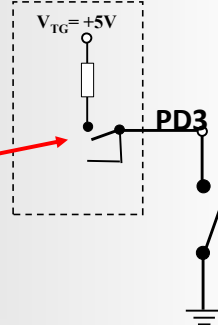


32

Résistance de tirage interne

- Plus pratique, les microcontrôleurs de génération récente peuvent avoir des résistances de tirage intégrés.
 - Il faut penser à les activer !
- Dans le framework Arduino :
`Pinmode(3, INPUT_PULLUP);`
 = Mise en service la resistance de tirage intégrée
- ESP32 : Fonctionnalité disponible sur chaque GPIO, PULLUP seulement (pas PULLDOWN)
- Aucun effet si GPIO en sortie

ATmega328



33

Exemple 1

- Configurer les broches 3, 5 et 7 de l'Arduino (PD3, PD5 et PD7) comme des sorties :

Avec le framework

```
pinMode(3, OUTPUT);
pinMode(5, OUTPUT);
pinMode(7, OUTPUT);
```

Eventuellement :

```
pinMode(PIN_D3, OUTPUT);
pinMode(PIN_D5, OUTPUT);
pinMode(PIN_D7, OUTPUT);
```

Si PIN_DX define dans les fichiers .h

Plus proche du microcontrôleur

```
DDRD = 0b10101000;
```

or

```
DDRD = 0xA8;
```

or

```
DDRD |= 1<<PD7 | 1<<PD5 | 1<<PD3;
```

34

Entrées : actionneurs du type "tout ou rien"

- Interrupteurs a levier, boutons poussoirs, fin de course
- ILS, capteurs hall
- Claviers... de tout types
 - « Décodage » confié à une électronique extérieure
- Joysticks, manettes de jeu
- Souris - trackballs



35

Avec le framework Arduino/Wiring

```
pinMode(pin, mode)
    pin: the number of the pin whose mode you wish
    to set
    mode: INPUT, OUTPUT, or INPUT_PULLUP (R=20K)
```


```
digitalWrite(pin, value)
    pin: the pin number
    value: HIGH or LOW
```

```
digitalRead(pin)
    pin: the number of the digital pin you want to
    read (int)
    Returns HIGH or LOW
```

Defining built-ins: LED_BUILTIN

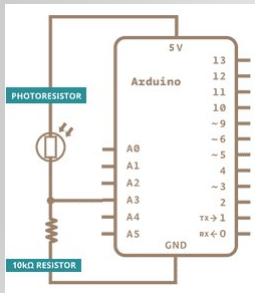
The constant LED_BUILTIN is the number of the pin to which the on-board LED is connected.

36



■ **analogRead()**

Entrée Analogique




```

int analogPin = 3; // potentiometer wiper (middle terminal)
                    // connected to analog pin 3
                    // outside leads to ground and +5V
int val = 0; // variable to store the value read

void setup() {
  Serial.begin(9600); // setup serial
}

void loop() {
  val = analogRead(analogPin); // read the input pin
  Serial.println(val); // debug value
  delay(1000);
}
```

37



■ Potentiomètre

■ Température

- Thermocouple (non linéaire!)
- Circuit spécialisés (linéarisation de la réponse $V = f(T)$)







■ Capteurs photoélectriques

■ Capteur PIR (détection de mouvement)

■ Capteurs de pression, hygrométrie

■ Microphones

Entrées : capteurs analogiques

38

E/S Analogiques

Syntax

```
analogRead(pin)
```

Parameters

pin: the number of the analog input pin to read from (0 to 5 on most boards, 0 to 7 on the Mini and Nano)

Returns

int (0 to 1023) -> ADC sur 10 bits

int (0 to 4095) -> ADC sur 12 bits

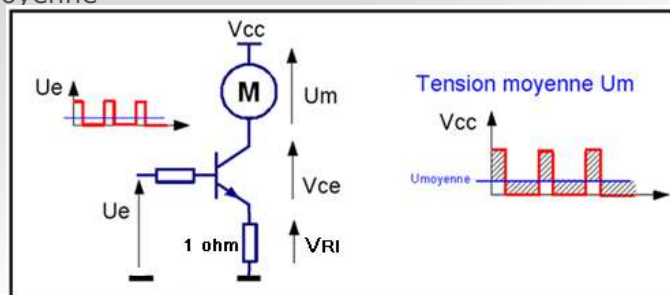
Dépend du microcontrôleur (Wiring porté sur différentes plateformes) et du câblage de la platine d'évaluation

Et analogWrite ?

39

Commande en modulation de largeur d'impulsion PWM

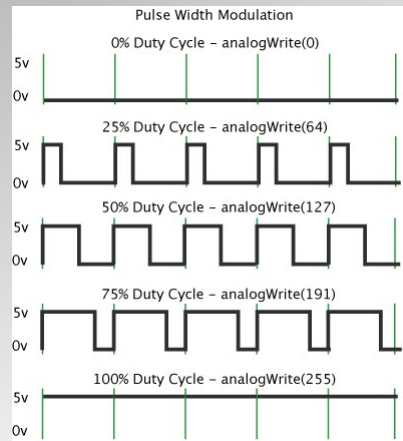
- Comment varier une grandeur électrique ?
 - Commande dite "linéaire" avec un transistor de puissance : rendement faible, 50% de l'énergie en chaleur dans le transistor
 - Commande "hachée" ou "pulsée" : commande tout ou rien dont la durée ON et OFF varie. L'équipement voit la moyenne



Modulation par largeur d'impulsion

40

PWM



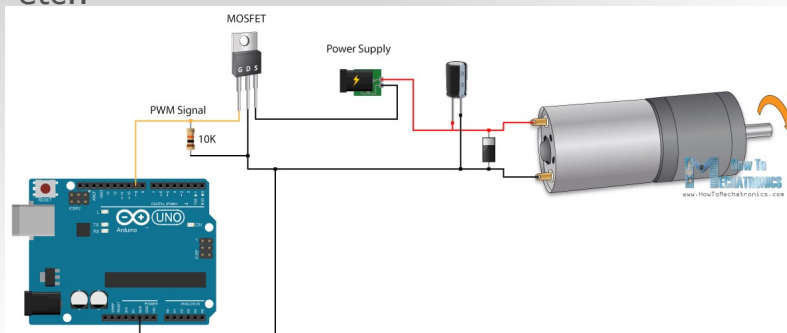
- On peut faire du PWM avec n'importe quel GPIO (c'est le CPU qui fait le travail)
- Bibliothèque de commande PWM spécifique dans le portage de Wiring pour l'ESP32

PWM Hardware = Analog Write

41

PWM

- La commande en PWM se retrouve dans divers domaines
 - Amplification audio
 - Modulation de la puissance de LED
 - Variation de vitesse d'un moteur électrique DC, etc..



42

analogWrite()

`analogWrite(pin, value)`

Parameters

`pin`: the pin to write to.

`value`: the duty cycle: between 0 (always off) and 255 (always on).

Returns Nothing

The frequency of the PWM signal on most pins is approximately 490 Hz.

On the Uno and similar boards, pins 5 and 6 have a frequency of approximately 980 Hz
ESP32 : 2 hardware DAC channels

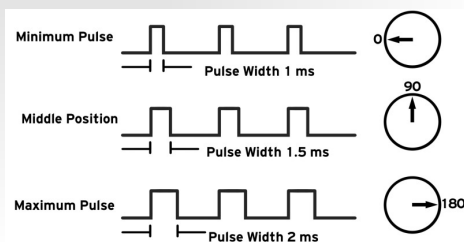
Sorties – Servomoteur et moteurs pas à pas



- Servo-moteurs
 - Economique (jouets/modelisme), à partir de 5€ pièce
 - Une seule ligne de commande, on envoie une impulsion, la durée de l'impulsion détermine un ANGLE
 - Pas de tour complet
- Moteurs pas à pas
 - Contrôle précis d'une POSITION
 - Rotation continue possible
 - Délicats à piloter, il faut respecter une séquences
 - Des composants électroniques externes facilitent leur utilisation

Servomoteur

- On peut commander la position angulaire, de 0° à 180° (ou selon les modèles 270°) avec précision (mais pas de tour complet)
- Facile à connecter
- La commande (improprement nommée PWM...) est une suite d'impulsions dont la durée contrôle l'angle



45

Servomoteur

- Il existe des bibliothèques spécifiques pour les servomoteurs
- Attention : la bibliothèque « classique » des platines ARDUINO ne marche pas avec l'ESP32**
- Plusieurs de dispo spécifique ESP32 sur Github.
- En général, classe Servo avec plusieurs méthodes, par exemple :
 - `int attach(pin)` : Turn a pin into a servo driver. pins 2,4,12-19,21-23,25-27,32-33 are recommended
 - `void write ()` : Sets the servo angle in degrees;
 - `void writeMicroseconds()` : Sets the servo pulse width in microseconds.
 - `void detach()` : Stops the attached servo, frees the attached pin, and frees its channel for reuse.

46