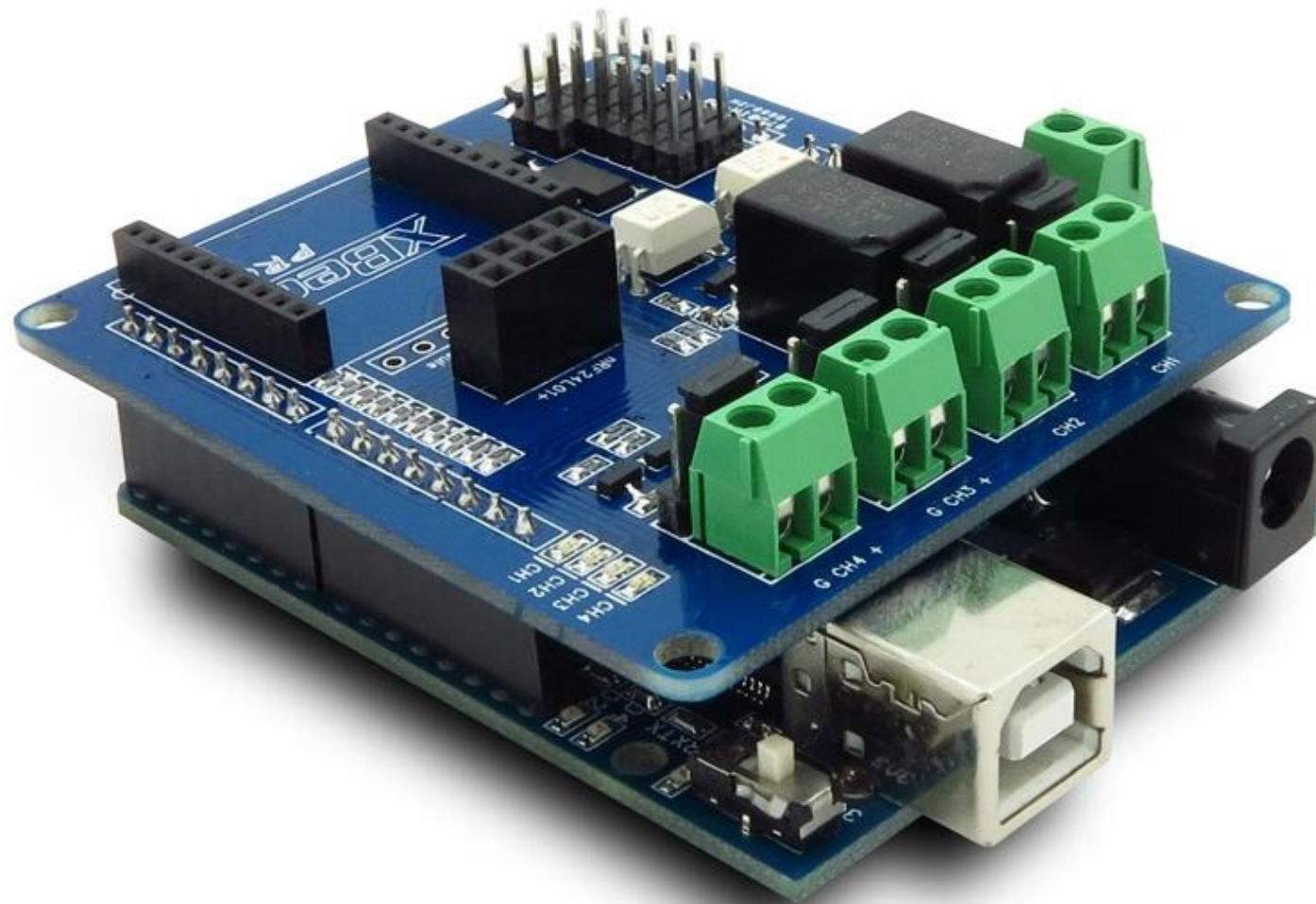




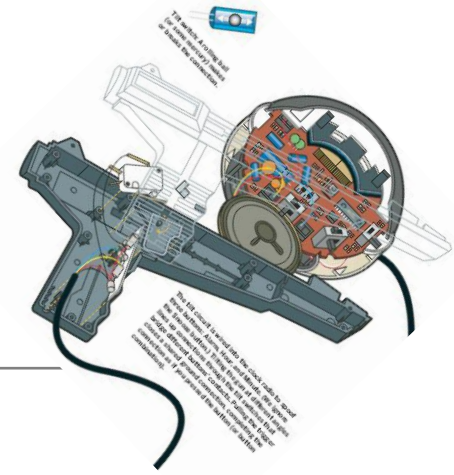
PUERTOS DIGITALES

SISTEMAS EMBEBIDOS



Filosofía Arduino

La filosofía Arduino se fundamenta en la idea de crear prototipos en lugar de solo hacer ideas!



Prototipo

Crear lo antes posible un demo de la idea a desarrollar



Mezclar

Podría utilizar este proyecto para este otro ...



Reusar

Este aparato ya no sirve, pero el ventilador lo podría usar en ...



Explorar

Que pasa si se hago esto? Como se vería si estos dos objetos se unen?



Modificar

El juguete solo dice una palabra, podría agregar mas ?



Colaborar

Juan puede prender leds, tal vez el pueda ayudarme a ...

PARTES DEL PROGRAMA

1. **Barra de Navegación:** Permite al usuario acceder a programas, ejemplos y principalmente seleccionar el Arduino a usar y el puerto de acceso.
2. **Cuadro de trabajo:** Donde programamos al Arduino.
3. **Cuadro de compilación:** Nos indica el proceso de compilación y la ubicación de archivo temporal .hex
4. **Barra de Acceso:** Permite compilar al Arduino y cargar el programa.
5. **Barra de Ubicación:** Observaremos el tipo de Arduino conectado y el puerto de comunicación.



PARTES DEL SKETCH

1. Descripción del programa: Para reutilización de código, es necesario presentar una breve descripción del programa y su funcionalidad.

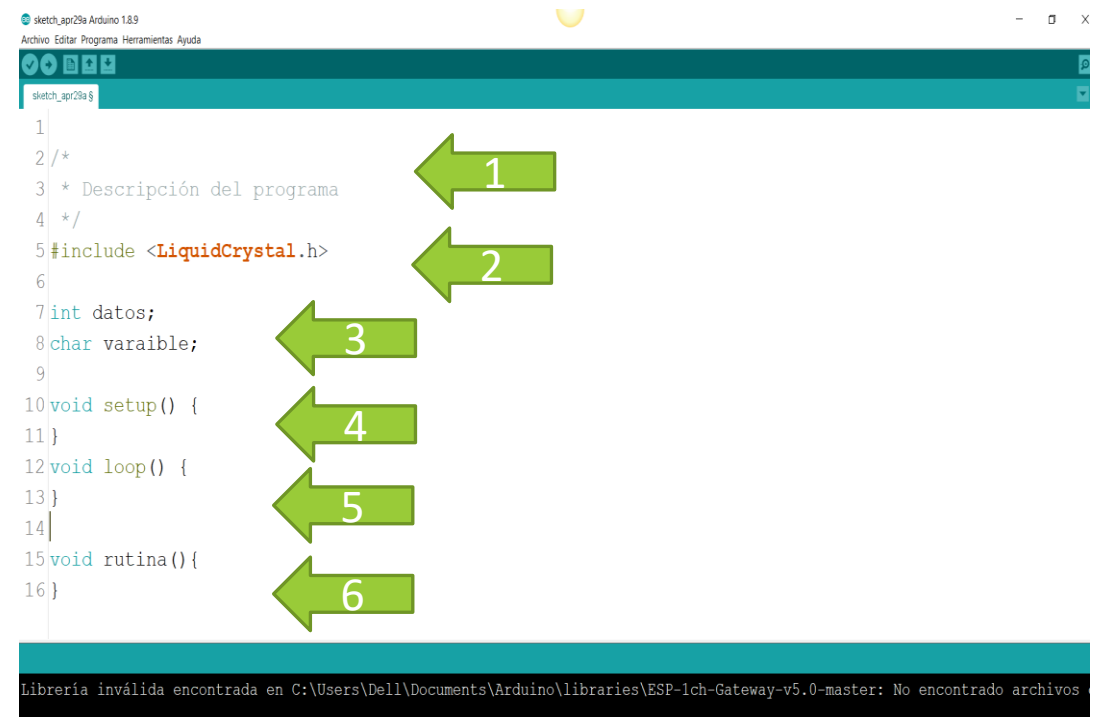
2. Librerías: Son líneas de código ya construidas para el uso de ciertos aspectos de Arduino para una mayor facilidad y se llaman a estos subprogramas como métodos o subrutinas.

3. Variables: Es un espacio de memoria donde se almacenan datos, se forma temporal o fija, dependiendo del tipo de dato se debe elegir la variable a usar, además de ellos es necesario primero declarar la variable con un nombre que la represente, pero si puede tener varios datos contenidos en una misma variable.

4. Void Setup: Es el lugar donde configuramos al Arduino sobre que pines y como vamos a usar, en cada tema de programación observamos la forma de configurar correctamente.

5. Void Loop: Es un espacio de programa que se repetirá infinitas veces hasta que alguna variable cambie de estado o se desconecte de la fuente, es donde irá el programa que dará el accionamiento al Arduino.

6. Void rutina: Rutinas de programación fuera del programa principal



```
1
2 /*
3  * Descripción del programa
4  */
5 #include <LiquidCrystal.h>
6
7 int datos;
8 char variable;
9
10 void setup() {
11 }
12 void loop() {
13 }
14
15 void rutina() {
16 }
```

Librería inválida encontrada en C:\Users\Dell\Documents\Arduino\libraries\ESP-1ch-Gateway-v5.0-master: No encontrado archivos

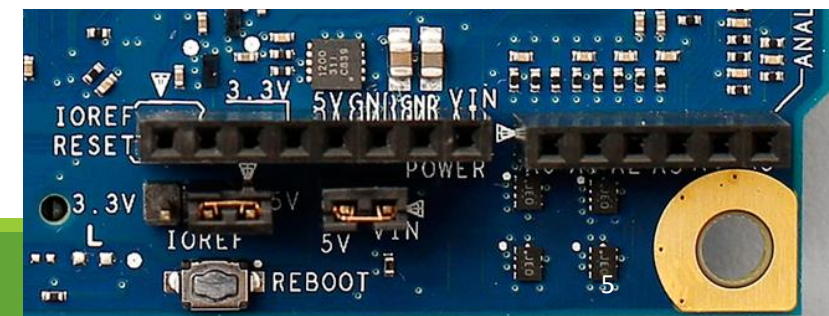


Entrada-Salidas Digitales de Propósito General

Las placas de arduino tienen 14 terminales de salida digital de propósito general. Cada terminal esta representada por un identificador numérico. Este identificador es utilizado para utilizar cada una de las terminales durante la programación del dispositivo.



Gran parte de los circuitos a conectar a Arduino se alimentan con **5 V**, sin embargo dispositivos mas recientes tienen interfaces de **3.3 V**. Arduino puede seleccionar el uso de cualquiera de los dos voltajes por medio del jumper **IOREF**.





Bibliotecas Arduino

Arduino comparte la amplia librería de rutinas de software para el control de diversos periféricos, así como para facilitar el uso de tareas sencillas como la escritura a una terminal de salida digital.

Algunas bibliotecas requieren su importación al sketch, otras son accesibles de manera implícita en cualquier sketch para Galileo.

En esta sección se utilizan las funciones:

- **pinMode()**
- **delay()**
- **digitalWrite()**

Para usar las funciones anteriores no es necesario incluir librería alguna.

pinMode() digitalWrite() delay()

pinMode()

Al hacer una llama a la rutina pinMode se le deja saber a Arduino si una terminal será utilizada como entrada o salida.

Sintaxis:

```
pinMode(pin, MODO)
```

MODO: INPUT o OUTPUT.

digitalWrite()

Una vez que se a especificado que una terminal de Arduino será utilizada como salida, se usa esta función para escribir un valor digital alto o bajo a dicha terminal.

Sintaxis:

```
digitalWrite(pin,  
ESTADO)
```

ESTADO: HIGH o LOW.

delay()

Rutina de retardo universal, detiene el flujo de ejecución del sketch actual.

Sintaxis:

```
delay(ms)
```

ms: numero de milisegundos a esperar. (1000 ms = 1).

Código blink.ino

```
/* blink.ino */  
  
int led = 13;  
  
void setup() {  
    pinMode(led, OUTPUT);  
}  
  
void loop() {  
    digitalWrite(led, HIGH);  
    delay(1000);  
    digitalWrite(led, LOW);  
    delay(1000);  
}
```

Lenguaje C

Asigna a la terminal 13 el rol de salida digital.

Escribe un valor lógico **ALTO** al pin 13.

Retardo de 1 segundo.

MANEJO DE VARIABLES DE NÚMEROS

Las variables son espacios de memoria del micro para almacenar datos temporales y que en el tiempo de ejecución cambia de valor.

int led =8 ;
1 2 3 4

1. Tipo de dato (integer).
2. Nombre a la variable.
3. Asginación de valor a la variable (opcional).
4. Fin de línea de código.

MANEJO DE CONSTANTES

No cambian su valor en tiempo de ejecución. Son útiles para prevenir que no se tenga errores.

`#define`

Ventajas

No ocupa memoria para almacenar las constantes, puesto que son remplazadas por sus valores

Si se usa en proyecto pequeños, como definir un pin, una cadena, etcétera, es adecuado.

Desventajas

No tienen tipo. Por lo que el compilador no podrá avisar si algo está mal

Son globales, por lo que se debe ser sumamente cuidadosos al referirse a ellas.

```
1 #define NUMERO
2
3 void setup() {
4     pinMode(NUMERO, OUTPUT);
5     digitalWrite(NUMERO, HIGH);
6 }
7
```

MANEJO DE CONSTANTES

Const

Ventajas

Obedecen el ámbito de las funciones. Es decir, se puede declarar dos constantes con el mismo nombre en diferentes funciones

También tiene la ventaja de que tiene tipo, por lo que el compilador nos informará los errores que puede que estemos cometiendo.

const también permite declarar arreglos, que #define no.

```
1 const int numero = 1;
2
3 void setup() {
4     pinMode(numero, OUTPUT);
5     digitalWrite(numero, HIGH);
6 }
7
```

PROGRAMACIÓN



CÓDIGO 1: PUERTOS COMO
SALIDA



CÓDIGO 2: MANEJO DE CICLOS
CON PUERTOS ENTRADA SALIDA



Bibliotecas Arduino

Algunas funciones a utilizar para el uso de las entradas digitales son:

- **pinMode()**
- **digitalRead()**

Para usar las funciones anteriores no es necesario incluir librería alguna.

pinMode() digitalRead()

pinMode()

Al hacer una llamada a la rutina pinMode se le deja saber a Galileo si una terminal será utilizada como entrada o salida.

Sintaxis:

```
pinMode(pin, MODO)
```

MODO: INPUT o OUTPUT.

digitalRead()

Después de especificar que una terminal se usará como entrada, se emplea esta función para leer un valor digital alto o bajo presente en dicha terminal.

Sintaxis:

 **pin** -> Número de terminal a leer

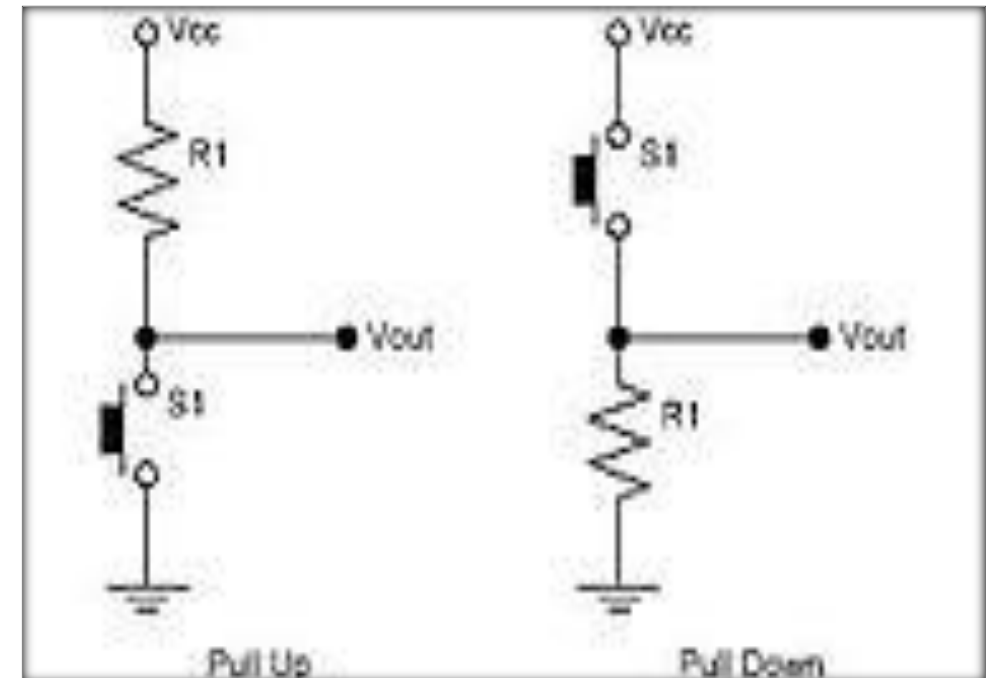
```
digitalRead(pin)
```

pullUP pullDown

Se consideran entradas digitales en el momento que nosotros recibimos 0 lógico o 1 lógico del exterior, sea desde un sensor, un switch, un botón, etc.

0 lógico : 0 voltios y 1,8 voltios

1 lógico: 2,1 voltios y 5 voltios



Código button.ino

```
const int buttonPin = 2;  
const int ledPin = 13;
```

```
int buttonState = 0;
```

```
void setup() {
```

```
    pinMode(ledPin, OUTPUT);  
    pinMode(buttonPin, INPUT);
```

← Configura la terminal 13 como salida y la terminal 2 como entrada.

```
}
```

```
void loop() {
```

```
    buttonState = digitalRead(buttonPin);
```

← Lee el estado de la terminal 2..

```
    if (buttonState == HIGH) {
```

```
        digitalWrite(ledPin, HIGH);
```

← Escribe un estado ALTO en la terminal 13

```
    }
```

```
    else {
```

```
        digitalWrite(ledPin, LOW);
```

← Escribe un estado BAJO en la terminal 13

```
    }
```

```
}
```

PROGAMACIÓN



CÓDIGO 3: PUERTOS COMO
ENTRADA



CÓDIGO 4: USO DE PULSADORES
Y CICLOS DE PROGRAMACIÓN