Programación en C para Arduino

Preparado por Gabriel Astudillo Muñoz
Escuela de Ingeniería Civil Informática
Universidad de Valparaíso

1 Introducción

Arduino[1] es una plataforma de hardware libre[2], basada en una placa electrónica que tiene un microcontrolador[3] (µC) y un entorno de desarrollo integrado (IDE[4]), diseñada para ser utilizada en proyectos multidisciplinares. Su programación se basa en el lenguaje C/C++. A pesar de que no es un requisito, es conveniente que la persona que quiera programar una placa Arduino, sepa lo básico de programación en los lenguajes mencionados.

Existen variados modelos de placas, las que se pueden revisar en el sitio web de Arduino. Además, en el mercado, hay diversas empresas que fabrican placas totalmente compatibles con Arduino. La Escuela de Ingeniería Civil Informática, dispone de placas Sparkfun[5], que es compatible con el modelo Arduino UNO. La ventaja de éste, es que se entrega en formato de kit de desarrollo, el que tiene todo lo necesario para realizar las tareas en las asignaturas que se ocupe, como por ejemplo, cables, motores, protoboard[6], luces LED, sensores, etc.

El objetivo de este material es que quien lo lea, este capacitado para programar un sistema con Arduino y entender el funcionamiento básico de algunos sensores que dispone dicho microcontrolador.

2 Descripción del hardware

Las placas Sparkfun, que son las que la Escuela dispone, están basadas en el chip ATmega328, que es el mismo µC que el utiliza el modelo Arduino UNO. Tiene 14 pines de entrada/salida (E/S) digitales, de los cuales 6 puedan simular salidas análogas mediante la técnica PWM y 6 entradas análogas. Funciona a una velocidad de 16[MHZ] y tiene 32[KB] de memoria flash y 2[KB] de SRAM. La Figura 1 muestra un diagrama de la placa.

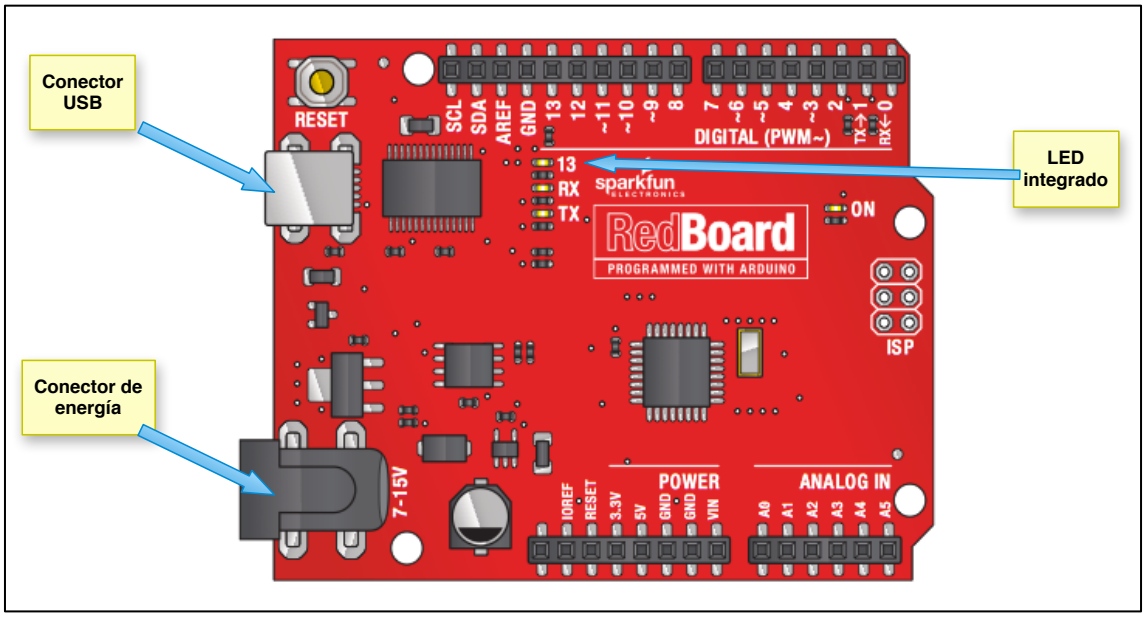


Figura 1

[1] http://www.arduino.cc
[2] http://es.wikipedia.org/wiki/Hardware_libre
[3] http://es.wikipedia.org/wiki/Microcontrolador
[4] http://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado
[5] http://www.sparkfun.com
[6] http://es.wikipedia.org/wiki/Placa_de_pruebas

2.1 Alimentación

La placa se puede energizar a través del conector USB o con una fuente externa entre 7[V] y 15[V]. Normalemente, se puede utilizar una batería de 9[V]. El conector externo, en el último caso, debe ser de 2,1[mm], con centro positivo.

2.2 Entradas y salidas

Cada uno de los 14 pines digitales (numerados del 0 al 13) pueden utilizarse como entradas o como salidas usando las funciones **`pinMode()`**, **`digitalWrite()`** y **`digitalRead()`** (ver Anexos 4.4 y 4.5, pp. 25). Las E/S operan a 5[V]. Cada pin puede proporcionar o recibir una corriente máxima de 40[mA].

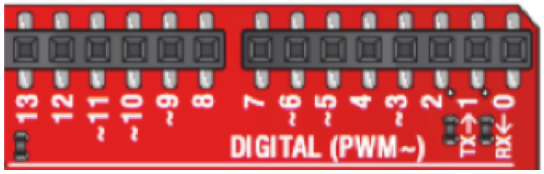


Figura 2

Los pines 3, 5, 6, 9, 10, y 11 proporcionan una salida PWM (modulación por ancho de pulsos) de 8 bits de resolución (valores de 0 a 255) mediante la función **`analogWrite()`**.

El pin digital 13 lleva conectado un LED integrado en la propia placa. Se encenderá cuando dicho pin se configura como salida y adopte un valor **`HIGH`**; con valor **`LOW`** se apaga.

La placa tiene 6 entradas analógicas, y cada una de ellas proporciona una resolución de 10 bits (1024 valores enteros sin signo).

2.3 Comunicaciones

La placa proporciona comunicación serial a través de los pines digitales 0 y 1, utilizados para la recepción (RX) y transmisión (TX) de datos. Un chip integrado en la placa canaliza esta comunicación serie, además, a través del puerto USB. El software de Arduino incluye un monitor de puerto serie, que permite enviar y recibir información textual hacia y desde la placa Arduino. Los leds RX y TX de la placa parpadearán cuando se detecte comunicación transmitida a través de la conexión USB.

3 Preparando el escenario de trabajo

3.1 Entorno de Desarrollo de Software

El entorno de desarrollo integrado (IDE) se puede bajar desde la página de Arduino y en la Figura 3 se puede distingue las partes que lo constituyen.

El área de edición de código es donde se escribirá el código del software que se requiere. Es sencillo visualmente y tiene ciertas características que usted debe descubrir mediante su uso. Normalmente, en la jerga de Arduino, aquí se crea el "*sketch*" (código fuente).

Una sección importante es el área de mensajes, en donde el compilador le informa posibles errores en el código. Además, en esta área, la placa arduino puede enviar información acerca de su estado, según como sea programado.

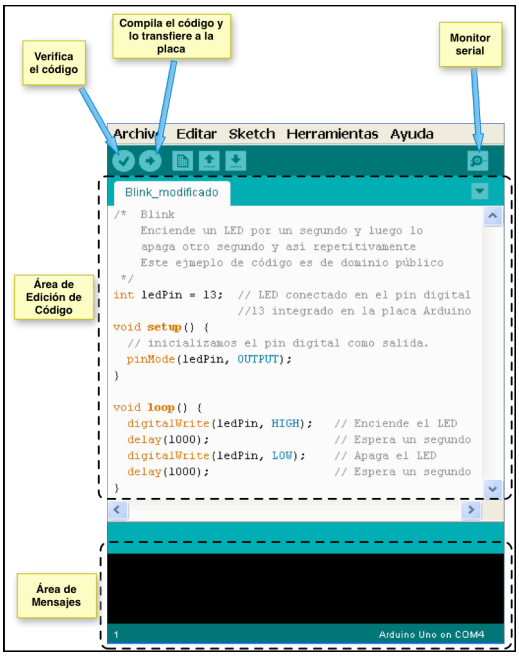


Figura 3

3.2 Conexión de la placa al computador

Cada kit Sparkfun dispone de un cable USB para poder conectar la placa a un computador, tal como se muestra en la Figura 5.

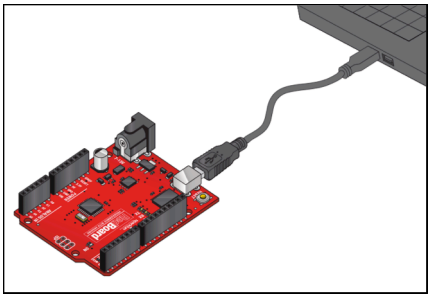


Figura 5

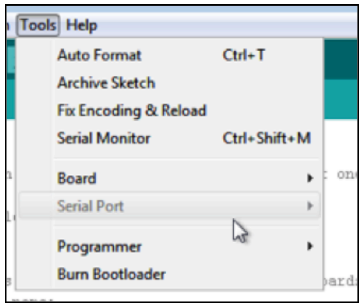


Figura 4

Cuando conecte ambos elementos, el sistema operativo del computador le puede solicitar los drivers para poder acceder a la placa. Si el menú *Herramientas→Puerta Serial* no está habilitado (ver ejemplo en la Figura 4), debe seguir las instrucciones de la página https://learn.sparkfun.com/tutorials/how-to-install-ftdi-drivers.

3.3 Protoboard[7]

Una placa de pruebas (o protoboard) es una placa de uso genérico, que se utiliza para construir prototipos de circuitos electrónicos sin utilizar herramientas para soldar. En el caso de las placas Sparkfun, éstas vienen con una, similar a la mostrada en la Figura 6. El diagrama de conexiones eléctricas se visualiza en la Figura 7.

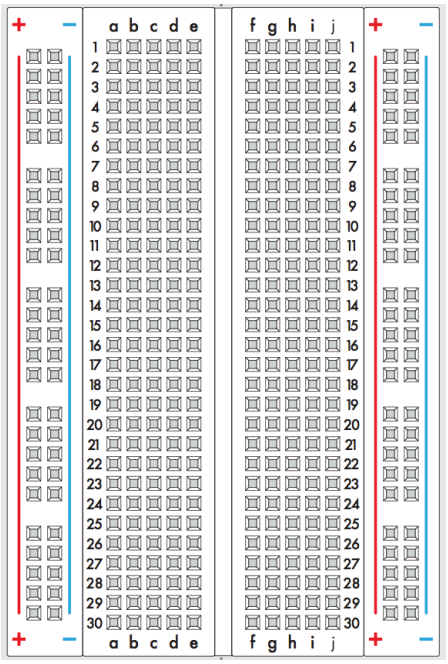


Figura 6

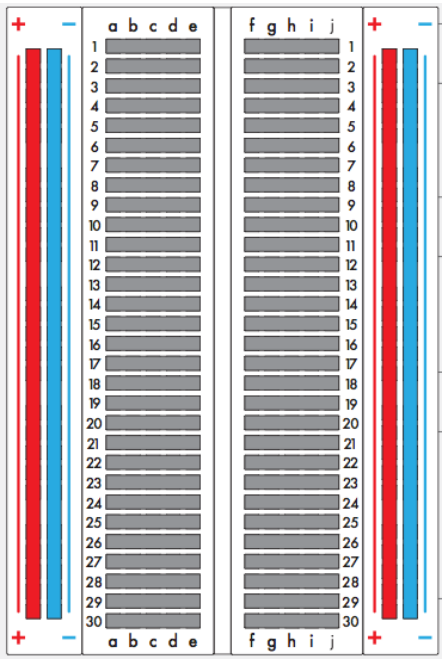


Figura 7

[7] http://es.wikipedia.org/wiki/Placa_de_pruebas

4 Forma de programar Arduino

El código que se debe escribir en el IDE de arduino debe tener cierta estructura, la que se indica en la Figura 8.

Figura 8

En la primera sección (Declaraciones Globales), se deben poner las bibliotecas específicas que se utilizarán (ninguna en el código de ejemplo) y las variables globales que se utilizarán (`pinLed` en este caso). Este bloque se ejecuta una sóla vez y bajo cualquier de los siguientes eventos:

- Encendido de la placa.
- Después de un reset.
- Después de cargar un programa desde el computador.

La función **`setup()`** se ejecuta después de la sección anterior y por una sóla vez. Se utiliza para configurar el hardware que se utilizará. En el ejemplo, se inicializa el pin 10 como salida.

La función **`loop()`**, por otro lado, se ejecuta después de la función anterior, de forma "perpetua", a una tasa de repetición muy cercana a la velocidad de trabajo de la placa, dependiendo de la cantidad de intrucciones que tenga.

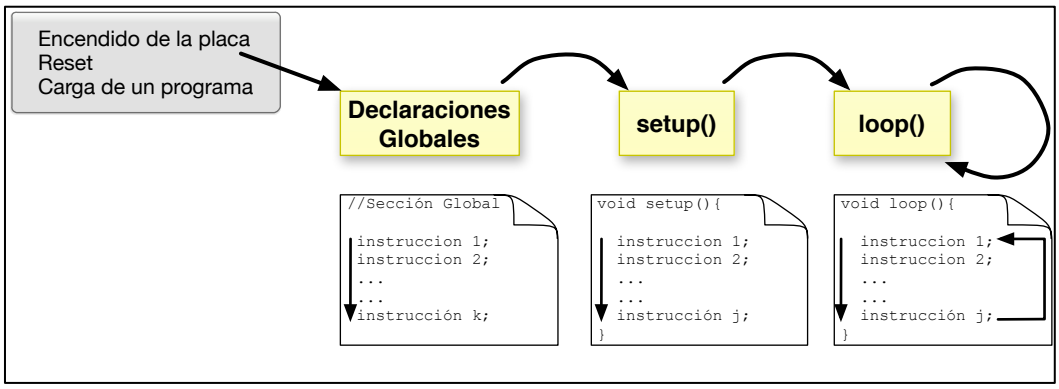


Figura 9

Los detalles del lenguaje de programación para el microcontrolador Arduino, se puede leer en Anexo 4, página 23.

5 Modalidad de trabajo

Arduino se creó para facilitar el prototipaje de ideas que mezclan software con componentes electrónicos, con el fin de crear soluciones que interactuén con el medio ambiente y con otros sistemas. En la Figura 10 se muestra un esquema que representa el trabajo que usted debe realizar para implementar sus ideas.

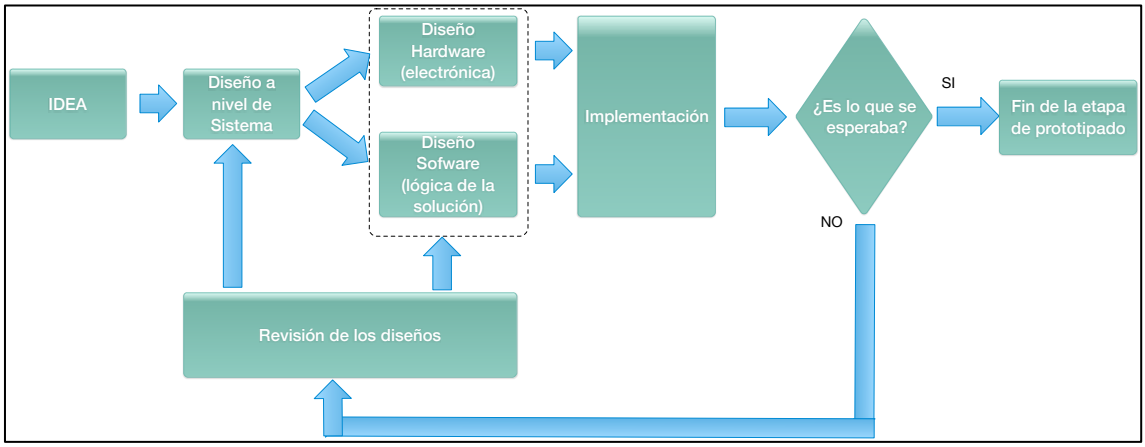


Figura 10

6 Primer prototipo con Arduino

6.1 La idea inicial

A modo de ejemplo, supongamos que usted quiere realizar un sistema, basado con Arduino, que prenda y apague un LED cada 1 segundo.

6.2 Diseño del hardware

En la sección 2.2 se menciona que Arduino dispone de cierta cantidad de salidas digitales, las que pueden estar en un nivel alto devoltaje (5[V]) y un nivel bajo de voltaje (cero), según cómo estén programadas. Luego, el LED debería estar conectado a una de dichas salidas.

El diagrama eléctrico podría ser el de la Figura 11. Bajo condiciones normales, el voltaje del LED es de 2[V] y para que tenga una luminosidad aceptable, su corriente debe ser por lo menos 5[mA]. Cuando el pin 10 está a 5[V], el voltaje de la resistencia es 3[V]. Luego, su corriente es, según la Ley de Ohm, aproximadamente 9[mA], corriente que es suficiente para que el LED se vea encendido.

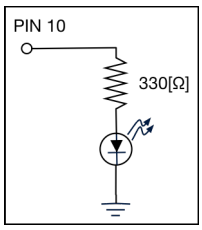


Figura 11

En la Figura 12 y Figura 13 se muestra la equivalencia para el LED y la resistencia utilizada.

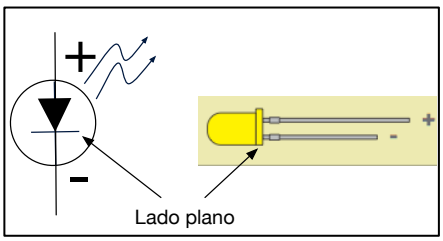


Figura 12

Figura 13

Finalmente, el esquema en la placa de desarrollo es el que se muestra en la Figura 14.

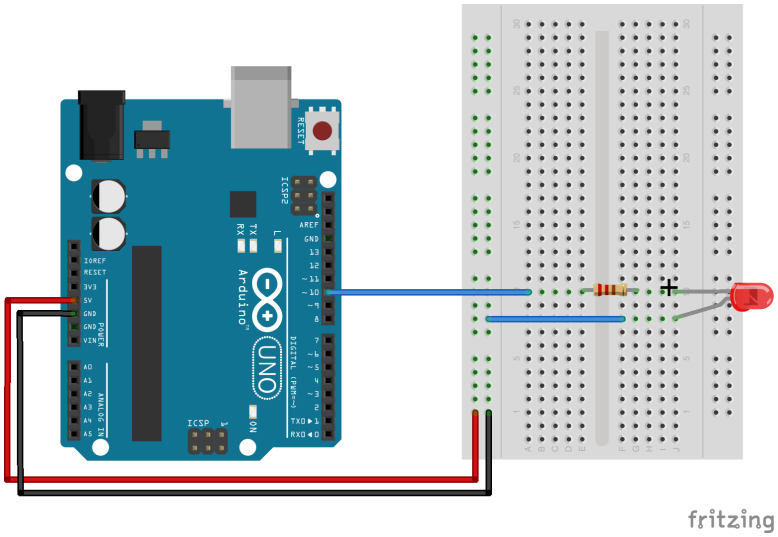


Figura 14 Circuito de la solución al problema planteado

6.3 Diseño del software

Para diseñar el software, primero hay que tomar en cuenta que éste y el hardware deben estar en sintonía para que el sistema funcione como se espera. Esto se traduce en visualizar correctamente el prototipo y entender cómo el software le va a enviar las señales eléctricas al hardware para que éste opere según lo planeado.

Según lo diseñado en la sección anterior, el pin 10 debe tener dos posibles valores: ALTO para que tenga un voltaje suficiente como para prender el LED, y BAJO, para apagarlo. Luego, este pin debe ser *digital* y de *salida* (esto es, debe entregar voltaje). Esto debe ser configurado en la función **`setup()`**, ya que se está preparando el hardware. Lo anterior, a nivel de código, se expresa como:

```
void setup() {
  pinMode(10, OUTPUT);
}
```

Tabla 1 Ejemplo de código

Además, es conveniente siempre dejar comentarios dentro del código. Esto representa que el programador tiene las ideas claras acerca de lo que está haciendo. Además, sirve para entender el código y encontrar posibles mal funcionamientos. Luego, el código de la Tabla 1 se debe reescribir como se muestra en la Tabla 2.

```
void setup() {
  pinMode(10, OUTPUT); //Inicializa el pin 10 como salida
}
```

Tabla 2 Ejemplo de código mejorado con comentarios pertinentes

Luego que se tiene configurado el hardware que se va a utilizar, hay que diseñar la lógica de la solución. Hay diversas formas, pero como recomendación, siempre se debe terminar esta fase con un diagrama donde indique cómo va a funcionar el software. Típicamente, esto es un diagrama de flujo[8], como el mostrado en la Figura 15.

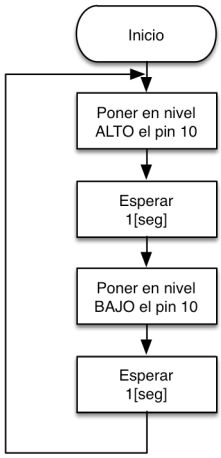


Figura 15

Tal como se ve en el diagrama de flujo, la solución se basa en un ciclo infinito de funcionamiento, el que se acomoda perfectamente con el modo de funcionamiento de la funcion **`loop()`**.

Para escribir un dato digital (Nivel alto o bajo, un Verdadero o Falso), se ocupa la funcion **`digitalWrite()`** y para que el sistema espere durante cierta cantidad de tiempo, se ocupa la función **`delay()`**. El código de la función **`loop()`** se muestra en la Tabla 3.

[8] http://es.wikipedia.org/wiki/Diagrama_de_flujo

```
void loop() {
  digitalWrite(10, HIGH);   // prenda el LED
  delay(1000);              // espere 1[seg]
                            // (1000 milisegundos
  digitalWrite(10, LOW);    // apague el LED
  delay(1000);              // espere 1[seg]
}
```

Tabla 3

Finalmente, en la Tabla 4 se muestra el código de la solución. Se agregó un variable global, `pinLed` y `duracion`, para que el mantenimiento del código a futuro sea menos costoso en tiempo.

```
int pinLed = 10; //Pin asociado al LED
                 //que se va a prender/apagar
int duracion = 1000; //duracion del parpadeo

void setup() {
  pinMode(pinLed, OUTPUT); //Se inicializa el
                           //pin como salida
}

void loop() {
  digitalWrite(pinLed, HIGH);// prenda el LED
  delay(duracion);           // espere 1[seg]
  digitalWrite(pinLed, LOW); // apague el LED
  delay(duracion)            // espere 1[seg]
}
```

Tabla 4 Código de la solución al problema planteado

Ahora sólo falta probarlo. Esto lo puedo hacer usted. Experimente. Vea el resultado. Modifique el código.

7 Ejemplos de circuitos

7.1 Parpadeo controlable de un LED

La idea del sistema es que la velocidad de parpadeo de un LED se controle a través de un potenciómetro[9], tal como se muestra en la Figura 16.

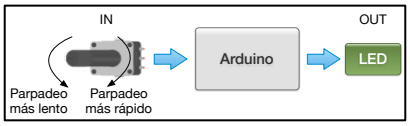


Figura 16

7.1.1 *Antecedentes para la solución*

El potenciómetro es una resistencia variable, que externamente tiene 3 pines de conexión y una perilla de ajuste (en el caso del disponible en el kit Spakfun), tal como se muestra en la Figura 17.

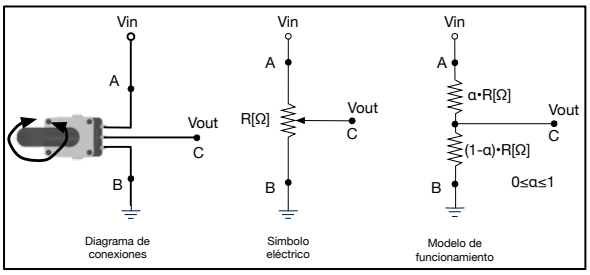


Figura 17

El factor α representa el nivel de rotación que tiene la perilla de ajuste: 0 para 0º y 1 para 180º. Por ejemplo, si α=0.5, entonces la perilla está justo en en mitad (90º). Realizando un divisor de voltaje (ver Anexo 2, página 21), se determina que el voltaje en en punto C es:

$$Vout = (1 - \alpha)Vin, \qquad 0 \leq \alpha \leq 1$$

Luego, a través del potenciómetro, se puede controlar el voltaje Vout, el que puede variar desde 0[V] hasta Vin. Desde el punto de vista de Arduino, si se conecta el punto C a un pin de entrada análogo, se podría leer un número entero en el rango [0,1023].

7.1.2 *Diseño de la solución*

El diseño debería ser similar al presentado en la sección 6 (página 5), salvo que ahora el valor de la variable duracion debe depender del voltaje presente en el pin central del potenciómetro. Para esto, se propone conectar dicho pin al pin análogo A0 y el LED al pin digital 10. El esquema eléctrico de la solución se muestra en Figura 18 y la implementación final en Figura 19.

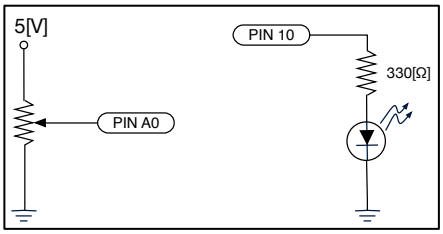


Figura 18

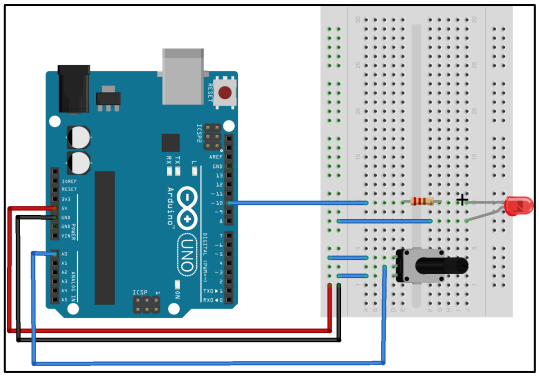


Figura 19

[9] Potenciómetro es un tipo de resistencia variable.

Finalmente, en la Tabla 5 se muestra el código de la solución.

```
int sensorPin = A0;   // El potenciometro esta conectado
                      // al pin analogo A0
int ledPin = 10;      // El LED esta conectado al
                      // pin digital 10
int duracion;
void setup() {
  pinMode(sensorPin, INPUT); //pin como entrada
  pinMode(pinLed, OUTPUT);   //pin como salida
}

void loop() {
  duracion = analogRead(sensorPin);
     //Por tratarse de una entrada analogica
     //duracion esta entre 0 y 1023
  digitalWrite(pinLed, HIGH);// prenda el LED
  delay(duracion);           // espere
  digitalWrite(pinLed, LOW); // apague el LED
  delay(duracion)            // espere
}
```

Tabla 5

Después que realice este ejercicio, detecte las debilidades que tiene la solución y proponga las respectivas mejoras.

7.2 Implementación de una compuerta AND

Este sistema implementa una función lógica Y en base a dos interruptores. La tabla de verdad de esta función es la que se muestra en la Tabla 6. El diseño conceptual, a nivel de sistema, se muestra en la Figura 20.

INa	INb	OUT
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 6

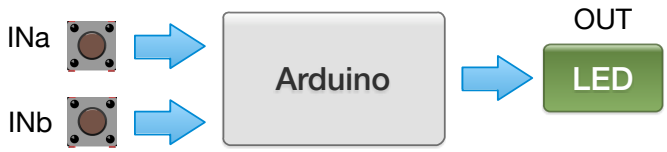


Figura 20

Mientras no se presione un interruptor, éste deberá entregar un valor **`LOW`** a cierto pin de entrada. Por el contrario, si se presiona, deberá entregar un valor **`HIGH`** al mismo pin. Un posible diseño eléctrico de la solución se muestra en la Figura 21. Analice por qué es necesario la resistencia en la parte de las entradas.

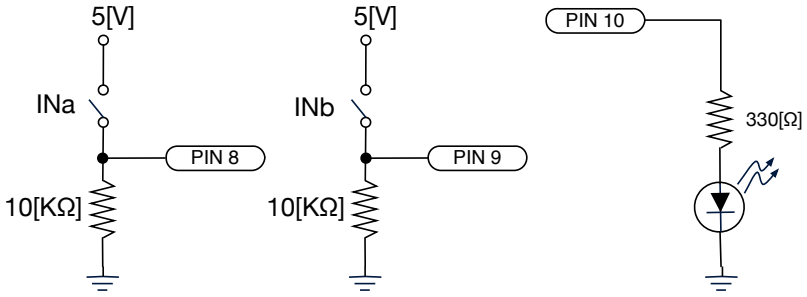


Figura 21

Finalmente, en la Tabla 7 se muestra el código de la solución.

```
int pin_INa = 8;    //
int pin_INb = 9;    //
int pin_LED = 10;   //

boolean INa, INb, OUT;

void setup() {
  pinMode(pin_INa, INPUT); //pin como entrada
  pinMode(pin_INb, INPUT); //pin como entrada
  pinMode(pin_LED, OUTPUT);//pin como salida
}

void loop() {
  INa = digitalRead(pin_INa);
  INb = digitalRead(pin_INb);

  OUT = INa && INb;
  digitalWrite(pin_LED, OUT);// prenda el LED
}
```

Tabla 7

7.3 Prender automáticamente un LED

La idea de este sistema es que un LED se encienda en ausencia de luz ambiental.Si hay luz ambiente, el LED debe permanecer apagado. Para medir la luz ambiental, se utilizar una foto-resistencia. El diseño conceptual, a nivel de sistema, se muestra en la

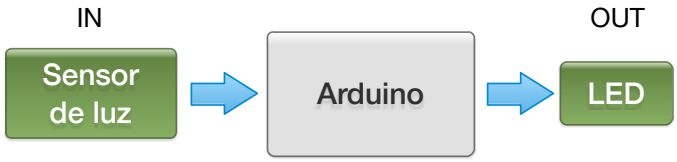


Figura 22

7.3.1 Antecedentes para la solución

Una foto-resistencia es una resistencia cuyo valor en [Ω] depende de la intensidad lumínica. La foto-resistencia que tiene el kit Sparkfun, posee las siguientes características:

(Ausencia de Luz)	(Luz directa)
$R_{sensor} \approx 30[K\Omega]$	$R_{sensor} < 1[K\Omega]$

La configuración que se puede utilizar es:

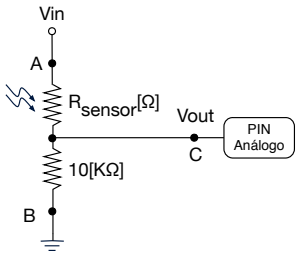


Figura 23

Bajo este esquema, realizando el divisor de voltaje respectivo, se tiene que $\mathbf{V_{out}=1,25[V]}$ cuando no hay luz ambiente. Para fines prácticos, se puede decir que si $\mathbf{V_{out}<1,5[V]}$, indicaría que hay penumbra o ausencia total de luz (hay que probar el sistema)

Desde el punto de vista del softwar, un voltaje de **1,5[V]** en las entradas analógicas representa un valor entero, aproximadamente, **307**.

7.3.2 Diseño de la solución

Para la solución, se propone conectar la foto-resistencia al pin análogo A0 y el LED al pin digital 10. El esquema eléctrico de la solución se muestra en Figura 24 y la implementación final en Figura 25.

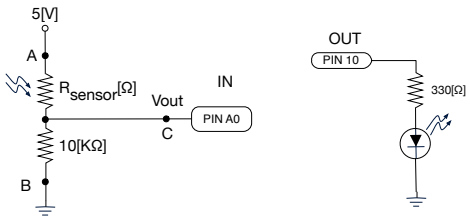


Figura 24

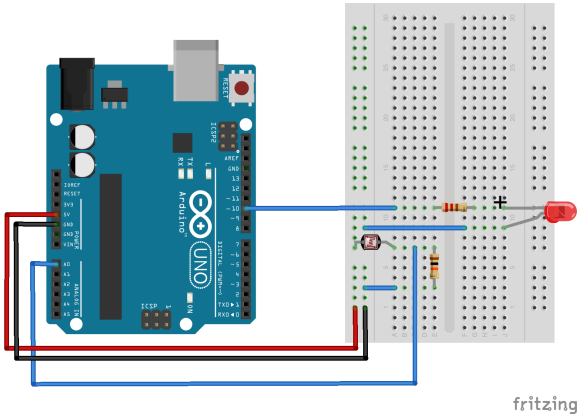


Figura 25

Finalmente, en la Tabla 8 se muestra el código de la solución.

```
int sensorPin = A0;    // El potenciometro esta conectado
                       // al pin analogo A0
int pinLed = 10;       // El LED esta conectado al
                       // pin digital 10
int intensidad;
void setup() {
  pinMode(sensorPin, INPUT); //pin como entrada
  pinMode(pinLed, OUTPUT);   //pin como salida
}

void loop() {
  intensidad = analogRead(sensorPin);
     //Por tratarse de una entrada analogica
     //intensidad esta entre 0 y 1023
  if(intensidad < 307)  //No hay luz ambiente o penumbra
    digitalWrite(pinLed, HIGH);// prenda el LED
  else
    digitalWrite(pinLed, LOW); // apague el LED
}
```

Tabla 8

Después que realice este ejercicio, detecte las debilidades que tiene la solución y proponga las respectivas mejoras.

7.4 Secuenciador de 6 LED

Este ejemplo hace uso de la programación a nivel de registros de Arduino (ver sección 6) y de los operadores al bit (ver sección 4.3)

```
void setup() {
  DDRB = B00111111; //pines 8 al 13 como salidas.
                    //los 2 bits MSB no se toman en cuenta
}

void loop() {
  PORTB=0; //Todos los led apagados
  for(int i=0; i<=5; i++){
    PORTB = (1 << i);
    delay(1000);
  }
}
```

Tabla 9

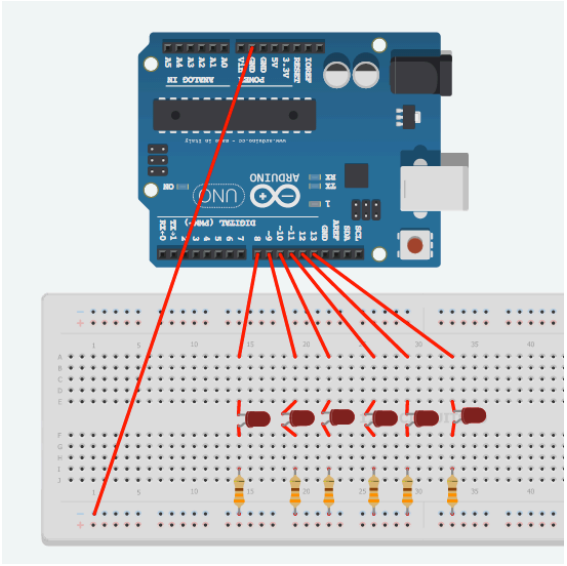


Figura 26

7.5 Mostrar mensajes en un Display LCD de 16x2

El kit Sparkfun viene con un displat LCD de 16x2 (16 columnas por 2 filas). El diagrama eléctrico se muestra en Figura 27.

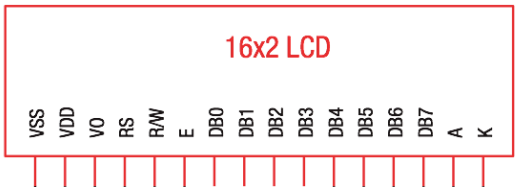


Figura 27

7.5.1 Función `LiquidCrystal()`

Esta función crea una nueva instancia en software de un LCD conectado a la placa.

Sintaxis: `LiquidCrystal name_display(RS, E, DB4, DB5, DB6, DB7);`

Ejemplo: `LiquidCrystal lcd(12, 11, 5, 4, 3, 2);`

El código anterior, representa el diagrama de conexiones de la Figura 28 y Figura 29.

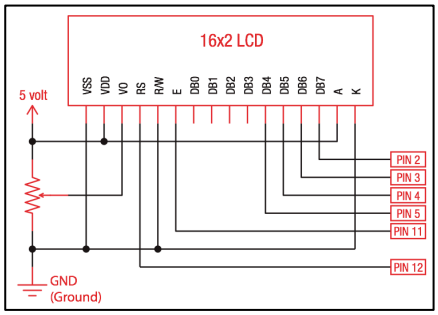


Figura 28

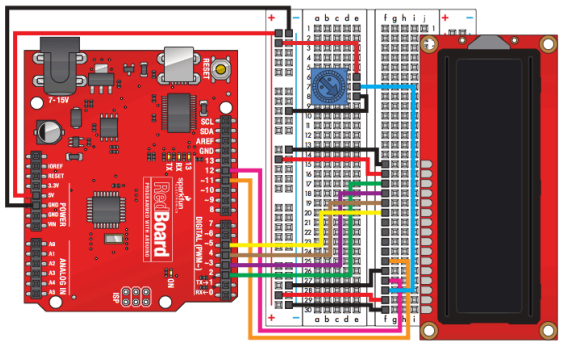


Figura 29

7.5.2 Función `begin(columnas, filas)`

Con la instacia creada, esta función defina el tamaño del display.

Ejemplo: `lcd.begin(16,2);`

7.5.3 Función `print(datos[, base])`

Envía datos a la pantalla.

Ejemplo 1: `lcd.print("hola");`

Ejemplo 2: `lcd.print(analogRead(A3), DEC);`

7.5.4 Función `write(dato)`

Envía un carácter al display

Ejemplo 1: `lcd.write(236);`

Ejemplo 1: `lcd.write(0xEC);`

7.5.5 Función `clear()`

Borra contenido de la pantalla.

Ejemplo: `lcd.clear();`

7.5.6 Función `setCursor(columna, fila)`

Especifica las coordenadas del lugar donde se va a escribir en la pantalla. En el caso de la pantalla que se utiliza, `columna` está en el rango [0,15] y `fila` en [0,1].

Ejemplo: `lcd.setCursor(0,1);`

7.5.7 Función `createChar(num, data)`

Permite crear caracteres definidos por el progamador.

Ejemplo: Para definir la letra ñ, se dibuja una matriz de 8x5 y se rellena adecuadamente:

	bit 4	bit 3	bit 2	bit 1	bit 0	Valor por fila
Byte 0	0	1	1	1	0	0x0E
Byte 1	0	0	0	0	0	0x00
Byte 2	1	0	1	1	0	0x16
Byte 3	1	1	0	0	1	0x19
Byte 4	1	0	0	0	1	0x11
Byte 5	1	0	0	0	1	0x11
Byte 6	1	0	0	0	1	0x11
Byte 7	0	0	0	0	0	0x00

```
#include <LiquidCrystal.h>

byte ntilde[] = {
  B01110, //Tambien se puede asignar 0x0E
  B00000, //0x00
  B10110, //0x16
  B11001, //0x19
  B10001, //0x11
  B10001, //0x11
  B10001, //0x11
  B00000  //0x00
};

void setup(){
  LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
  lcd.createChar(0, ntilde);
}

void loop(){
  lcd.write(0);
}
```

Tabla 10

Para crear caracteres, se puede utilizar la aplicación online que está disponible en http://mikeyancey.com/hamcalc/lcd_characters.php

7.6 Mostrar una página web con el dato de un sensor

Para tener conectividad dentro de una red LAN, es necesario conectar a la placa arduino un shield Ethernet. Para que éste shield funcione en la redboard, es necesario cablear externamente los pines del ICSP a los pines que se especifican

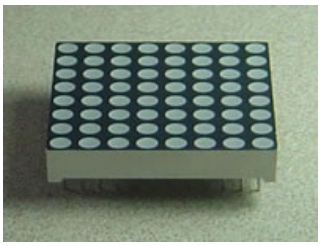
Figura 30

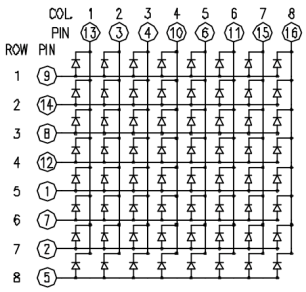
Un ejemplo de servidor web

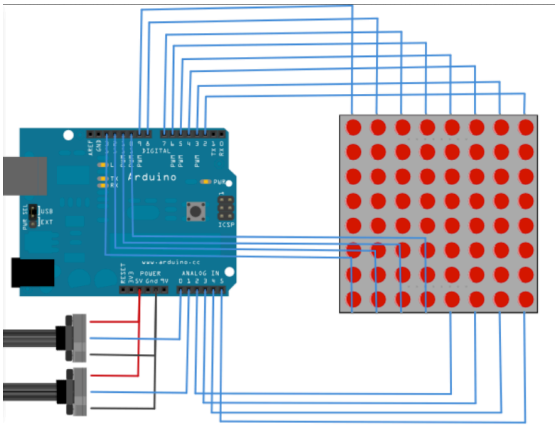
```
#include <SPI.h>
#include <Ethernet.h>

byte mac[]={0x90,0xA2,0xDA,0x0D,0x78,0x80};
IPAddress ip(192,168,1,177);

EthernetServer server(80);

void setup() {
 // Open serial communications and wait for port to open:
  Serial.begin(9600);
   while (!Serial) {
    ; //Needed for Leonardo only
  }

  // start the Ethernet connection and the server:
  Ethernet.begin(mac, ip);
  server.begin();
  Serial.print("Server: ");
  Serial.println(Ethernet.localIP());
}

void loop() {
  // listen for incoming clients
  EthernetClient client = server.available();
  if (client) {
    Serial.println("new client");
    // an http request ends with a blank line
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        Serial.write(c);
        if (c == '\n' && currentLineIsBlank) {
          // send a standard http response header
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println("Connection: close");
          client.println("Refresh: 5
          client.println();
          client.println("<!DOCTYPE HTML>");
          client.println("<html>");
          // output the value of each analog input pin
          for (int analogChannel = 0; analogChannel < 6; analogChannel++) {
            int sensorReading = analogRead(analogChannel);
            client.print("analog input ");
            client.print(analogChannel);
            client.print(" is ");
```

```
            client.print(sensorReading);
            client.println("<br />");
          }
          client.println("</html>");
          break;
        }
        if (c == '\n') {
          // you're starting a new line
          currentLineIsBlank = true;
        }
        else if (c != '\r') {
          // you've gotten a character on the current line
          currentLineIsBlank = false;
        }
      }
    }
    // give the web browser time to receive the data
    delay(1);
    // close the connection:
    client.stop();
    Serial.println("client disonnected");
  }
}
```

7.7 Matrix de Led de 8x8







```
#include <TimerOne.h>
#include <FrequencyTimer2.h>


// 2-dimensional array of row pin numbers:
const int row[8] = {
  2,7,19,5,13,18,12,16 };

// 2-dimensional array of column pin numbers:
const int col[8] = {
  6,11,10,3,17,4,8,9  };

// 2-dimensional array of pixels:
int pixels[8][8];

// cursor position:
int x = 5;
int y = 5;

byte filaActual = 0;

void setup() {
```

```
  for (int thisPin = 0; thisPin < 8; thisPin++) {
    pinMode(col[thisPin], OUTPUT);
    pinMode(row[thisPin], OUTPUT);
    // LEDS  off:
    digitalWrite(col[thisPin], HIGH);
  }


  // initialize the pixel matrix:
  for (int x = 0; x < 8; x++)
    for (int y = 0; y < 8; y++)
      pixels[x][y] = HIGH;

  FrequencyTimer2::disable(); //pin 11 se usa como salida
(Time2 inabilita PWM en esete pin)
  // Set refresh rate (interrupt timeout period)
  FrequencyTimer2::setPeriod(500); //useg
  // Set interrupt routine to be called
  FrequencyTimer2::setOnOverflow(updateDisplay);

  Timer1.initialize(1000); //1000 useg
  Timer1.attachInterrupt(readSensors);
}

void loop() {


}

void updateDisplay() {

    digitalWrite(row[filaActual], HIGH);

    for(int columnaActual = 0; columnaActual < 8; columnaActual++){
      digitalWrite(col[columnaActual], pixels[filaActual][columnaActual]);

      if(pixels[filaActual][columnaActual] == 0)
        digitalWrite(col[columnaActual], HIGH);
    }
    digitalWrite(row[filaActual], LOW);

    filaActual = ++filaActual % 8;
}


void readSensors() {
  // turn off the last position:
  pixels[x][y] = HIGH;
  // read the sensors for X and Y values:
  x = map(analogRead(A0), 0, 1023, 0, 7);
  y = map(analogRead(A1), 0, 1023, 0, 7);
  // set the new pixel position low so that the LED will turn
on
  // in the next screen refresh:
  pixels[x][y] = LOW;

}
```

ANEXOS

1 Ley de Ohm

Ley de Ohm

Esta ley dice que el Voltaje en un conductor eléctrico es directamente proporcional a la corriente que circula a través de ella. La constante de proporcionalidad se llama Resistencia y se mide en [Ω] (ohm).

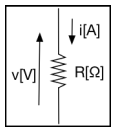


Figura 31

$$v = R \cdot i$$

Por ejemplo, si la resistencia R=10[KΩ] y el voltaje v=3[V], entonces i=0,3[mA]

2 Divisor de voltaje

Un divisor de voltaje consiste en dos resistencias, dispuestas en una configuración coma la mostrada en la Figura 32. El voltaje Vout se determina por la siguiente ecuación:

$$Vout = \underbrace{\frac{R2}{R1 + R2}}_{G} \cdot Vin$$

El factor G es menor que 1. Por lo tanto, se tiene que Vout<Vin.

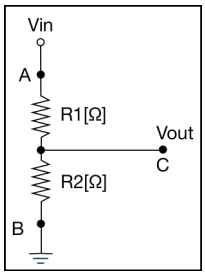


Figura 32

3 Medición de sensores resistivos

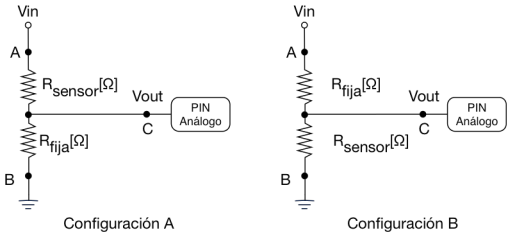


Figura 33

Los sensores que se utilizan con arduino, en general, son resistivos. Esto quiere decir que son resistencias que varían su valor según alguna interacción con el medio ambiente. Por ejemplo, el sensor de luz (fotoresistencia) que tiene el kit Sparkfun es una resistencia, cuyo valor depende de la temperatura ambiente.

Ahora bien, la placa Arduino no puede medir resistencia, pero sí puede medir voltajes, a través de sus entradas análogas. Luego, integrando los conocimientos acerca del divisor de voltaje, es posible disponer de un voltaje que dependa del medio ambiente. Para esto, se debe utilizar una de las configuraciones de la Figura 33. Para efectos prácticos, R_{fija}=10[KΩ] y Vin=5[V].

Ahora, se debe tener claro que, según la configuración utilizada, se obtendrán distintos valores de entrada para la placa, bajo las mismas condiciones ambientales. Esto se ejemplifica a continuación.

3.1 Ejemplo de rango de valores entregados por el sensor.

Se desea utilizar la fotoresistencia para medir la luz ambiente. En ausencia de luz, tiene una resistencia de 1[MΩ] y con luz ambiente (aprox 10[lux]), 50[KΩ]. Encontrar el rango de valores que serán leídos por la placa.

Configuración A)

$50[K\Omega] < R_{sensor} < 1000[K\Omega]$ y $Vout = \frac{10[K\Omega]}{R_{sensor}+10[K\Omega]} \cdot Vin$

Si R_{sensor}=50[KΩ], Vout≈0,9[V]

Si R_{sensor}=1000[KΩ], Vout≈0,05[V]

Configuración B)

$50[K\Omega] < R_{sensor} < 1000[K\Omega]$ y $Vout = \frac{R_{sensor}}{R_{sensor}+10[K\Omega]} \cdot Vin$

Si R_{sensor}=50[KΩ], Vout≈4,2[V]

Si R_{sensor}=1000[KΩ], Vout≈5[V]

Para la configuración A, el valor mínimo es 0,05[V], que Arduino lo interpretará como el número entero 10. El más alto, será 0,9[V], que corresponde a un valor 184 (recordar que si se lee 5[V], Arduino interpreta dicho voltaje como el número entero 1023).

Para la configuración B, el valor mínimo es 4,2[V], que corresponde al número entero 859. El más alto, es 5[V], que equivale al número 1023.

Resumiendo, según la configuración utilizada, se tendrán los siguientes rangos de voltajes y de valores de conversión para Arduino.

Configuración	**Voltaje en la entrada**		**Valor de conversión**	
	(Ausencia de Luz)	**(Luz ambiente)**	**(Ausencia de Luz)**	**(Luz ambiente)**
A	0,05[V]	0,9[V]	10	184
B	5[V]	4,2[V]	1023	859

En conclusión, dependiendo de la configuración del hardware de conexión del sensor, los valores interpretados por Arudino cambiarán. Luego, la lógica del software cambia en la toma de decisiones. Por ejemplo, si se quiere tomar una decisión en torno a si hay luminosidad o no, el código en ambas configuraciones debería ser:

Configuración	**Código para detectar ausencia de luz**

A	`...`
	`nivelLuz = analogRead(A0);`
	`if(nivelLuz < 20) //Hay poca luminosidad`
	`  { //Haga algo }`
	`...`

B	`...`
	`nivelLuz = analogRead(A0);`
	`if(nivelLuz > 1000) //Hay poca luminosidad`
	`  { //Haga algo }`
	`...`

4 Especificación del lenguaje de programación

Arduino se programa en C, pero por tratarse de un μC, la cantidad de intrucciones es menor a las que se pueden utilizar en un computador de escritorio o portátil. Además, se tiene otras funciones que no existen en C estándar. En esta sección, se describen las funciones que son propias del lenguaje C de Arduino y que serán utilizadas por usted en el taller o asignatura que está cursando. El resto de las funciones clásicas de C, como por ejemplo, **`for()`**, **`while()`**, **`if()`**, etc, las puede estudiar en el sitio web de Arduino.

Las palabras reservadas del lenguaje es escriben, en este documento, con una fuente Courier New, en negrita. Por ejemplo **`delay()`** es una palabra reservada. Si una palabra no está en negrita, pero con la fuente mencionada, se refiere a una variable o función que se utiliza en el código. Por ejemplo, `pinLed` se refiere a una variable utilizada en un código.

4.1 Constantes

4.1.1 Niveles Lógicos

`false`: se define como 0 (cero).

`true`: es cualquier número distinto que 0. Por comodidad, se asocia un valor 1 a este nivel.

4.1.2 Niveles de Voltaje

`HIGH`: Representa un voltaje de 5[V] en una salida y un voltaje mayor que 3[V] en una entrada.

`LOW`: Representa un voltaje de 0[V] en una salida y un voltaje menor que 2[V] en una entrada.

4.2 Tipos de datos

Tipo	Descripción	Ejemplo

void	Sólo en las declaraciones de funciones. Indica que no retorna un valor.	`void setup(){`
		`  ...`
		`}`

boolean	Variable que puede tener dos valores: **`true`** o **`false`**. También **`HIGH`** o **`LOW`**.	`boolean ledON = false;` `boolean OFF = LOW;`
char	Almacena número enteros entre [-128, 127]. Ocupa 1[Byte] de memoria y almacena un valor de caracter.	`char dato = 'A';` `char dato = 65; //Es lo mismo`
unsigned char	Lo mismo que char, pero almacena número enteros de 8 bits. Rango entre [0, 255].	`unsigned char dato = 240;`
byte	Lo mismo que unsigned char. Es preferible desde el punto de vista de la coherencia del código.	`byte b = B10010;`
int	Permite almacenar un número entero en el rango [-32768, 32767] . Ocupa 2[Bytes] de memoria.	`int valorInt =23456;`

unsigned int	Permite almacenar un número entero en el rango [0, 2^{16}-1] .
	Ocupa 2[Bytes] de memoria.
		`unsigned int valor = 65432;`
		`unsigned int valor = -1; //valor`
		`//es 65535`

word	Lo mismo que unsigned int	`word valor = 65432;` `word valor = -1; //valor es 65535`
long	Permite almacenar un número entero en el rango [-2^{31}, 2^{31}-1] . Ocupa 4[Bytes] de memoria.	`long valor = 2345678L;`
unsigned long	Permite almacenar un número entero en el rango [0, 2^{32}-1] . Ocupa 4[Bytes] de memoria.	

float	Permite almacenar un número en punto flotanta (fraccionario) en el rango [$-3.4 \cdot 10^{-38}$, $3.4 \cdot 10^{-38}$].
	Ocupa 4[Bytes] de memoria.
		`float valorSensor = 4.7657;`
		`double valorSensor = 4.7657;`
		`//precision doble no existe`
		`//en Arduino. Queda como float`

arrays	Los arreglos es una colección de variables a las que se accede a través de un número índice, que comienza en cero.	`int pinPWM[] = {3, 5, 6, 9, 10, 11};` `char mensaje[] = "El dato llego";`

char array	Es una de las formas de declarar un string[10]. Cada elemento del string se puede acceder a través de su respectivo índice.	`char str1[] = {'h','o','l','a','\0'};`
		`char str2[] = "hola";`
		`//str1 y str2`
		`//son lo mismo`

4.3 Operaciones matemáticas al bit

4.3.1 Operaciones lógicas

	AND	OR	XOR	NOT

Definición	`0 & 0 == 0`
	`0 & 1 == 0`
	`1 & 0 == 0`
	`1 & 1 == 1`
		`0 | 0 == 0`
		`0 | 1 == 1`
		`1 | 0 == 1`
		`1 | 1 == 1`
			`0 ^ 0 == 0`
			`0 ^ 1 == 1`
			`1 ^ 0 == 1`
			`1 ^ 1 == 0`
				`0 == ~1`
				`1 == ~0`

4.3.2 Corrimiento de bits

	<< n	>> n
Definición	`Shift Left`	`Shift Right`

4.3.3 Ejemplos

```
int a =  92;     // 01011100
int b = 101;     // 01100101
int c = a & b;   // c = 01000100      ==> 68
int d = a | b;   // d = 01111101      ==> 125
int e = a ^ b;   // e = 00111001      ==> 57
int f = ~a;      // f = 10100011      ==> -93
int g = a >> 4; // g = 00000101      ==> 5
int h = a << 4; // h = 010111000000 ==> 1472
```

4.3.4 Usos de las operaciones lógicas al bit

Para todos los ejemplos de esta sección, se debe considerar las siguientes variables:

```
int a = 93;     // 01011101
int b;
```

4.3.4.1 Masking

El operador `&` es utilizado para seleccionar un conjunto de bits de un número.

Ejemplo 1: Se necesita accesar a los 3 bits menos significados de la variable **`a`** y almacenarlos en la variable **`b`**, el código sería:

```
b = a & B111;     // 100 ==> 5
```

Ejemplo 2: Se necesita accesar a los bits 2, 3 y 4 de la variable **`a`** y almacenarlos en la variable **`b`**, el código sería:

```
b = a & B11100; // 00011100 ==> 28
b >>= 2;        // 111 ==> 7
```

Ejemplo 3: Se necesita accesar a los bit 4 de la variable **`a`** y almacenarlos en la variable **`b`**, el código sería:

```
b = (a & B00010000) >> 4;
b = (a & (1 << 4)) >> 4; //Es lo mismo
```

4.3.4.2 Seteo y Reseteo de bits

El operador | es utilizado para llevar a 1 determinados bits de un número. Por otro lado, el operador `&` es utilizado para llevar a 0 determinados bits de un número.

Ejemplo 1: Se necesita asegurar que los bits 0, 5 y 6 de **`a`** sean "1". El código sería:

```
b = a | B01100001;     // 01111101 ==> 125
```

[10] La otra forma es a través de un objeto String. La información para este tipo de declaración, está en el sitio de Arduino.

Ejemplo 2: Se necesita asegurar que los bits 0, 5 y 6 de **a** sean "0". El código sería:

```
b = a & B10011110;     // 00011100 ==> 28
```

4.3.4.3 Cambio de bits

El operador ^ es utilizado para cambiar el estado de determinados bits de un número. Esto es, si es 1, lo lleva a 0 y viceversa.

Ejemplo: Se necesita cambiar los bits 0, 5, 6 y 7 de **a**. El código sería:

```
b = a ^ B11100001;     // 10111100 ==> 188
```

4.4 Modo de funcionamiento de los pines de entrada/salida

Los pines, tanto digitales como análogos, se pueden utilizar como entrada o como salida. La configuración se realiza a través de la función **pinMode(**pin, modo**)**, donde pin es el número del pin utilizado y modo puede ser **OUTPUT** o **INPUT**, dependiendo si que quiere que sea salida o entrada, respectivamente. Por ejemplo, el código de la Tabla 11 configura el pin digital 9 y 10 como entrada y salida, respectivamente y el pin análogo A3 como entrada.

```
void setup() {
  pinMode(9, INPUT)
  pinMode(10, OUTPUT);
  pinModo(A3,INPUT);
}
```

Tabla 11

OBSERVACIÓN: Los pines configurados como salidas pueden ser dañados si se ponen en cortocircuito a tierra o a la alimentación de 5[V]. Por esta razón, es conveniente conectar los pines de salida a otros dispositivos con resistencias, para limitar la corriente máxima. Típicamente, se utilizarán resistencias de 330[Ω] ó 10[KΩ], que son las que vienen con el kit utilizado.

4.5 Funciones de Entrada/Salida digitales

Al leer o escribir en un pin digital sólo son posibles dos valores: **HIGH** y **LOW**. Para leer, se utiliza la función **digitalRead(**pin**)**, y para escribir, **digitalWrite(**pin, valor**)**.

4.5.1 Función digitalRead(pin)

En el código de la Tabla 12, dato tendrá el valor **HIGH** si, y sólo si, en el pin 9 hay un voltaje mayor que 3[V].

```
void loop() {
  ...
  dato = digitalRead(9);//Lee el dato que
                        //esta en el pin 9
  ...
}
```

Tabla 12

Por otro lado, dato tendrá el valor **LOW** si, y sólo si, en el pin 9 hay un voltaje menor que 2[V].

4.5.2 Función digitalWrite(pin, valor)

En el código de la Tabla 13 , si dato tiene el valor **HIGH**, el pin 10 tendrá un voltaje de 5[V].

```
void loop() {
  ...
  digitalWrite(10, valor);//Escribe en el pin 10
  ...
}
```

Tabla 13

Por otro lado, si dato tien el valor **LOW**, el pin 10 hay un voltaje de 0[V].

4.6 Funciones de Entrada/Salida análogas

Para leer, se utiliza la función **`analogRead(`**`pin`**`)`**, y para escribir, **`analogWrite(`**`pin, valor`**`)`**.

4.6.1 Función `analogRead(`pin`)`

Lee el valor de voltaje en el pin analógico especificado por la variable `pin` (**`A0`** a **`A5`**) y dependiente del voltaje, devuelve un número entero entre 0 y 1023, siendo éste último el valor que se le asigna a un voltaje de 5[V].

```
void loop() {
  ...
  dato = analogRead(A3);//Lee el voltaje que
                        //esta en el pin A3
  if(dato > 512) {
    //Hay más de 2,5[V]
  }
  ...
}
```

Tabla 14

4.6.2 Función `analogWrite(`pin, valor`)`

Permite generar un voltaje en los pines digitales marcadas como PWM[11] (3, 5, 6, 9, 10 y 11). Un valor 255 genera un voltaje de 5[V] y una valor 0, genera 0[V]. Por ejemplo, el código de la Tabla 15 permite variar continuamente el brillo de un led conectado al pin digital 9.

```
int ledPin = 9;     // LED conectado al pin digital 9
int valor;          // variable que representa el
                    // voltaje del led
void setup() {
  pinMode (ledPin, OUTPUT);  // establece el pin como salida
}
void loop() {
  for(valor=0;valor<=255; valor += 10){
    analogWrite(ledPin, valor);
    delay(100);
  }
  for(valor=255;valor>=0; valor -= 10){
    analogWrite(ledPin, valor);
    delay(100);
  }
}
```

Tabla 15

[11] En realidad, lo que hace esta función es generar un voltaje pseudo-analógico en el *pin digital* especificado, mediante una onda cuadrada constante con el ciclo de trabajo especificado (0 corresponde a siempre “off” y 255 a siempre “on”). Esta técnica se conoce como onde PWM.

4.7 Funciones de comunicación serie

Se utilizan para la comunicación entre la placa Arduino y un computador u otros dispositivos. Las placas Arduino se comunican entre sí por los pines digitales 0 y 1, para la recepción (RX) y transmisión (TX), respectivamente, y con el computador a través de la conexión USB. Por lo tanto, si utiliza estas funciones, no puede usar los pines 0 y 1 para entrada o salida digital.

4.7.1 *Comunicación Arduino - Computador*

Para que el computador pueda visualizar los datos enviados por la placa, se puede utilizar el "Monitor Serial" incorporado en el entorno de Arduino, tal como se muestra en la Figura 34. El monitor serial debe tener seleccionada la misma velocidad utilizada en la función **`Serial.begin()`**.

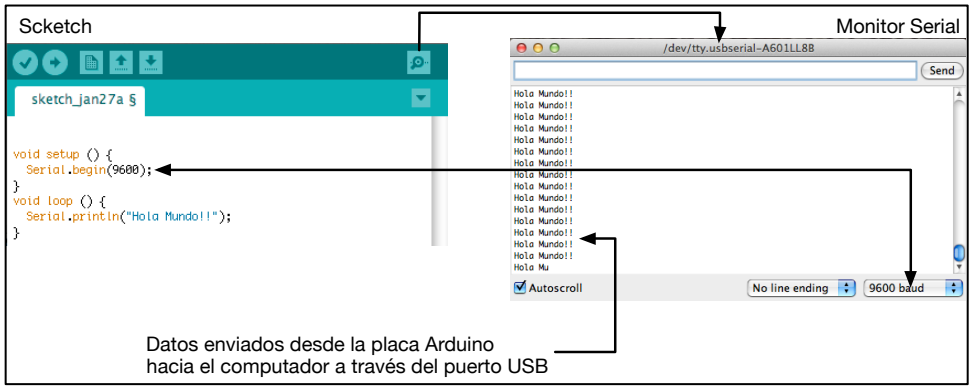


Figura 34

4.7.2 *Función `Serial.begin(velocidad)`*

Establece la velocidad de transmisión de datos en símbolos por segundo (baudios)[12] para la transmisión de datos serie. Por costumbre, se suele utilizar una velocidad de 9600[baudios]. Obviamente, se pueden utilizar velocidades de transmisión numéricamente más grandes[13].

4.7.3 *Función `Serial.available()`*

Devuelve el número de Bytes que están disponibles para **lectura** en el buffer de entrada del puerto serie, el que puede almacenar hasta 64[Bytes].

```
void setup() {
   Serial.begin(9600);
}

void loop() {
  if (Serial.available() > 0) {//Si hay datos disponibles
    //Codigo relacionado con la lectura de datos
    //del puerto serie.
  }
}
```

Tabla 16

4.7.4 *Función `Serial.end()`*

Desactiva la comunicación serie, permitiendo a los pines 0 (RX) y 1 (TX) ser utilizados como entradas o salidas digitales. Para volver a habilitar la comunicación serie, se llama a **`Serial.begin()`**.

4.7.5 *Función `Serial.print(valor[,formato])` y `Serial.println(valor[,formato])`*

Envía datos al puerto serie como texto ASCII. Los datos float son impresos por omisión con dos decimales. La diferencia entre **`Serial.print()`** y **`Serial.println()`** es que la primera no añade retorno de carro ni nueva línea.

[12] Normalemente, el símbolo que se transmite es un bit. En este caso, *baudio* y *bits por segundo* (*bps*) significan lo mismo.
[13] Las velocidades permitidas (en baudios) son: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200.

Si se envía un número entero, el campo formato se puede utilizar para establecer la base numérica del número. El campo formato puede tomar los siguientes valores: `DEC`, `HEX`, `OCT`, `BIN`.

Si se envía un número en punto flotante, el campo formato se puede utilizar para especificar la cantidad dígitos decimales que tiene dicho número.

4.7.6 Función `Serial.write(datos)`

Envía datos binarios al puerto serie. La variable datos puede ser un byte o una serie de bytes. Por ejemplo:

- **`Serial.write(`**`35`**`)`** enviará un byte, con el número 35 codificado en binario.
- **`Serial.write`**`("hola")` enviará una serie de bytes, con el string "hola". Esta función es similar a **`Serial.print()`**.

4.7.7 Función `Serial.read()`

Permite leer los datos que llegan por el puerto serie.

```
int incomingByte = 0;  // dato serial que llega

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

void loop() {
  // Enviar datos solo si se reciben datos
  if (Serial.available() > 0) {
    // leer el byte que llega
    incomingByte = Serial.read();

    // Mostrar el dato recibido
    Serial.print("Dato recibido: ");
    Serial.println(incomingByte, DEC);
  }
}
```

Tabla 17

4.8 Funciones de tiempo más utilizadas

4.8.1 Función `millis()`

Devuelve la cantidad de milisegundos transcurridos desde que la placa Arduino empezó a ejecutar el programa actual. Este número se desbordará (volverá a cero), después de aproximadamente 50 días. El dato devuelto es de tipo `unsigned long` (rango $[0, 2^{32} - 1]$).

Observación: Debido a que el dato que retorna **`millis()`** es de tipo `unsigned long`, por lo que se pueden generar errores si se intenta hacer operaciones matemáticas con otros tipos de datos.

4.8.2 Función `delay(mtiempo)`

Pausa el programa durante el tiempo (en milisegundos) especificado en el parámetro `mtiempo`. El dato dado como parámetro es de tipo `unsigned long`.

Observación: esta función detiene las lecturas de sensores, no se pueden manipular los pines, entre otras cosas.

5 Interrupciones

Una interrupción permite que el flujo normal de un programa se detenga y se ejecute un función específica. Una vez que esta función termina, el control retorna al programa. Al igual que en otros sitemas, Arduino tiene dos clases de interrupciones: por hardware y software.

5.1 Interrupciones por Hardware

En el caso de arduino UNO, tiene dos interupciones, **`INT0`** e **`INT1`** asociadas a los pines 2 y 3, respectivamente. Una interrupción suceder en cualquier momento de la ejecución del programa.

5.1.1 Función attachInterrup(interrupt, ISR, mode)

Los argumentos de esta función son:

`interrupt:` Número de la interrupción {0, 1}

`ISR:` (**I**nterruption **S**ervice **R**outine) Función que se va a ejecutar cuando se produzca la interrupción.

`mode:` Define cuando la interrupción se gatillará.

`LOW`: se gatilla cuando el pin está en un nivel bajo.

`CHANGE`: se gatilla cuando el pin cambia de valor.

`FALLING`: se gatilla cuando el pin pasa de un nivel alto a un nivel bajo.

`RISING`: se gatilla cuando el pin pasa de un nivel bajo a un nivel alto

```
int pin = 13;
volatile int state = LOW;

void setup(){
  pinMode(pin, OUTPUT);
  attachInterrupt(0, blink, CHANGE);
}

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

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

Tabla 18

5.1.2 Función noInterrupts() y interrupts()

En algunas ocaciones, se quiere que ciertas zonas del código no sean interrumpidas por las interrupciones. Esto se logra con estas funciones.

```
int pin = 13;
volatile int state = LOW;

void setup(){
  pinMode(pin, OUTPUT);
  attachInterrupt(0, blink, CHANGE);
}

void loop(){
  digitalWrite(pin, state);
  noInterrupts();
  // Codigo que no debe ser interrumpido
  interrupts();
```

```
  // el resto del codigo
}

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

Tabla 19

5.2 Interrupciones por Software

Las interrupciones por software, en al caso de Arduino, están asociadas a temporizadores. Esta significa que se debe configurar un temporizador por cierta cantidad de tiempo, y cuando éste se cumpla, se gatilla la ejecución de una función previamente definida. Existen varias bibliotecas que permiten el uso de temporizadores. En particular, se verá la `timer1.h`.

Esta biblioteca se debe bajar desde la URL http://code.google.com/p/arduino-timerone/downloads/list. El texto de esta sección se basa en la página web http://playground.arduino.cc/Code/Timer1.

5.2.1 Ejemplo de uso

```
#include "TimerOne.h"

void setup(){
  pinMode(10, OUTPUT);
  Timer1.initialize(500000);// Inicializa el timer1,
                            // y lo configura a 500[ms]
  Timer1.attachInterrupt(callback);// asociada la funcion
                                   // saludo() como servicio de
                                   // de interrupcion
}

void callback(){
  digitalWrite(10, digitalRead(10) ^ 1);
}

void loop()
{
  // codigo...
}
```

5.2.2 Función `initialize(period)`

Especifica cada cuánto microsegundos se ejecutará la función que atienda la interrupción. Por omisión, el períodoes 1[segundo]. Cuando se inicializa el timer, los pines 9 y 10 se deshabilitan para la función `analogWrite()`.

5.2.3 Función `attachInterrupt(function)`

Llama a la función fuction en el intervalo de tiempo especificado por la función anterior.

5.2.4 Función `detachInterrupt()`

Esta función deshabilita la interrupción inicializada.

6 Programación a nivel de registros

La placa Arduino, tiene tres puertos (`B`, `C` y `D`) según se resume en la Tabla 20.

Puerto	Pines
B	Digitales 8 al 13
C	Entradas Análogas
D	Digitales 0 al 7

Tabla 20

Además, cada puerto es controlado por tres registros, los que están definidos como variables a nivel de programa (ver Tabla 21). Cada uno de estos registros es de 8[bits]. En los puertos que tienen menos de 8 pines, como es el caso del puerto B, los bits más significativos, que no están asociados a pines físicos, no se toman en cuenta.

Registro	Descripción
`DDR`	Determina qué pines son entradas o salidas (valor `1` para salida, `0` para entrada)
`PORT`	Determina qué pin está en un nivel alto o bajo.
`PIN`	Entrega el estado de los pines del puerto (si es entrada o salida)

Tabla 21

Por ejemplo, se quiere configurar los puertos digitales 8 al 13 como salidas. Para esto, se debe utilizar el registro `DDRB` y el valor que se le debe asignar es:

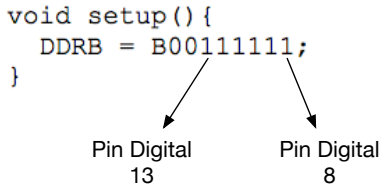


Figura 35

Cabe destacar que el código de la Figura 35, es equivalente a `DDRB=63`.

Si se quiere que los pines del puerto `B` 8, 10 y 13 estén en un nivel alto, se puede escribir el siguiente código:

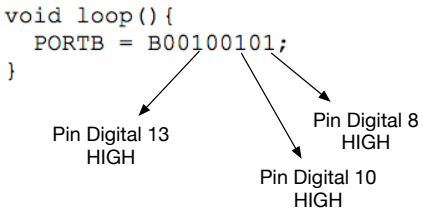


Figura 36

Ver ejemplo de uso de estos registros en la sección 7.4.