Laboratorio 1: Puertos de Entrada y Salida

1. Datos de la Practica

Carrera	INGENIERÍA ELECTRÓNICA		
Semestre		Grupo	
Tipo de Práctica	☐ Laboratorio ☐ Simulación	Fecha	
Asignatura	Curso de Arduino		
Unidad Temática			
Nº Alumnos por práctica	2	Nº Alumnos por reporte	2
Nombre del Profesor			
Nombre(s) de Alumno(s)	1. 2.		
Tiempo estimado		Vo. Bo. Profesor	
Comentarios			

2. Objetivos

- Conocer el funcionamiento de las tarjetas Arduino y su entorno de programación.
- Manejar los pines de una tarjeta Arduino como entrada/salida.

3. Componentes a Utilizar

Por cada práctica y por cada puesto de laboratorio, los materiales a utilizar son:

Cantidad	Descripción
1	Computadora
1	Arduino MEGA
5	LEDs
5	Resistencia de 330 ohm
1	Resistencia de 10K
1	Pulsador

4. Introducción

Arduino

Arduino es una plataforma de desarrollo basada en una placa electrónica de hardware libre que incorpora un microcontrolador ATMEL re-programable y una serie de pines hembra, los que permiten establecer conexiones entre el microcontrolador y los diferentes sensores y actuadores de una manera muy sencilla, creando así diversos proyectos electrónicos de una manera fácil y flexible.

El software de Arduino consiste de dos elementos: un entorno de desarrollo IDE (basado en el entorno de processing y en la estructura del lenguaje de programación Wiring), y en el cargador de arranque (bootloader, por su traducción al inglés) que es ejecutado de forma automática dentro del microcontrolador en cuanto este se enciende.

Las placas Arduino son diversas, pero todas se programan mediante un computador, usando comunicación serie y presentan algunas características en común como son:

- Velocidad en Mhz.
- Tamaño de memoria RAM, FLASH y EEPROM.
- Cantidad de pines de entrada/salida.
- Numero de pines analógicos.
- Cuantos puertos UART, I2C, SPI tiene.
- Tamaño de la tarjeta de evaluación.
- Bits del procesador.
- Voltaje del procesador.

Para la ejecución de estas guías de laboratorio bien pueden implementar cualquier tarjeta de Arduino, siempre y cuando tengan presente las características antes mencionadas. Para efectos demostrativos se ha usado la tarjeta Arduino MEGA R3 con un ATmega2560 que posee 54 pines de I/O, 16 entradas analógicas, 32KB de memoria Flash, 2KB de memoria RAM, 16MHz de frecuencia de operación y opera a un voltaje de 5V. Esta tarjeta se compara con otras similares en la siguiente tabla.

Tipo de Arduino	Procesador	# I/O	# entradas ADC	FLASH/RAM KB	Mhz	Voltaje (VDC)
Arduino UNO R3	ATmega328p	14	6	32 / 2	16	5
Arduino MEGA R3	ATmega2560	54	16	256 / 8	16	5
Arduino NANO	ATmega328p	14	6	32 / 2	16	5/3.3
Arduino Leonardo	ATmega32u4	14	6	28 / 2.5	16	5
Arduino DUE	AT91SAM3X8E	54	12	512 / 96	84	3.3

Tabla 1 Características de los modelos de Tarjetas de Arduino.

Programación del Arduino

La Estructura de un Programa en Arduino se divide en dos partes:

```
void setup (){
Estamentos;
}
```

En donde setup() es la parte encargada de recoger la configuración. (Entradas y Salidas)

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

loop() es la que contienen el programa que se ejecutará cíclicamente (de ahí el termino loop–bucle).

La función setup() se invoca una sola vez cuando el programa empieza. Se utiliza para inicializar los

modos de trabajo de los pins, o el puerto serie. Después de llamar a setup(), la función loop() hace precisamente lo que sugiere su nombre, se ejecuta de forma cíclica, lo que posibilita que el programa este respondiendo continuamente ante los eventos que se produzcan en la placa. Ambas funciones son necesarias para que el programa trabaje.

Las llaves **"()"** sirven para definir el principio y el final de un bloque de instrucciones. Se utilizan para los bloques de programación setup(), loop(), if.., etc.
El punto y coma **";"** se utiliza para separar instrucciones en el lenguaje de programación de Arduino. También se utiliza para separar elementos en una instrucción de tipo "bucle for".

Los bloques de comentarios, o comentarios multi-línea son áreas de texto ignorados por el programa que se utilizan para las descripciones del código o comentarios que ayudan a comprender el programa. Comienzan con **/ *** y terminan con ***** **/** y pueden abarcar varias líneas. Una línea de comentario empieza con **//** y terminan con la siguiente línea de código. Al igual que los comentarios de bloque, los de línea son ignoradas por el programa y no ocupan espacio en la memoria.

Tipos de Datos según Arduino

Tipos de Datos	Memoria que ocupa	Rango de valores
boolean	1 byte	0 o 1 (True o False)
byte/ unsigned char	1 byte	0 – 255
char	1 byte	-128 – 127
int	2 bytes	-32768 – 32767
word / unsigned int	2 bytes	0 – 65535
long	4 bytes	-2147483648 – 2147483647
unsigned long	4 bytes	0 - 4294967295
float / double	4 bytes	-3,4028235E+38 - 3,4028235E+38
string	1 byte + x	Array de caracteres
array	1 byte + x	Colección de variables

Tabla 2 Tipos de Datos según Arduino.

Entre los más importantes y usados tenemos:

- El tipo **boolean** que indica verdadero o falso.
- El tipo **int** para almacenar números enteros.
- El tipo **float** para números con decimales, no enteros
- El tipo **string** que es una secuencia y agrupación de datos de tipo char. Se utiliza para almacenar cadenas de texto.
- y los **array** que es una colección de datos de un mismo tipo.

Operadores en la Programación de Arduino

Operadores booleanos

- && ('y' lógico)
- || ('o' lógico)
- ! (negación lógica)

Operadores aritméticos

- = (operador de asignación)
- + (suma)
- - (resta)
- * (multiplicación)
- / (división)
- % (módulo)

Operadores de comparación

- == (igual que)
- != (distinto que)
- < (menor que)
- > (mayor que)
- <= (menor o igual que)
- >= (mayor o igual que)

Operadores de acceso a punteros

- * operador de indirección
- & acceso a memoria

Operadores compuestos

- ++ (incremento)
- -- (decremento)
- += (suma compuesta)
- -= (resta compuesta)
- *= (multiplicación compuesta)
- /= (división compuesta)
- &= ('y' a nivel de bits compuesto)
- |= ('o' a nivel de bits compuesto)

Operadores a nivel de bits

- & ('y' a nivel de bits)
- | ('o' a nivel de bits)
- ^ (xor a nivel de bits)
- ~ (not a nivel de bits)
- << (desplazamiento de bits a la izquierda)
- >> (desplazamiento de bits a la derecha)

Variables y Constantes

Una variable es una manera de nombrar y almacenar un valor numérico para su uso posterior por el programa. Como su nombre indica, las variables son números que se pueden variar continuamente en contra de lo que ocurre con las constantes cuyo valor nunca cambia. Una variable debe ser declarada y opcionalmente, asignarle un valor.

Las variables pueden ser declaradas al inicio de la función Setup ().

Ejemplo:
```
int Led = 10;
void setup ()
{ Estamentos;
}
```

En Arduino las constantes se clasifican en grupos:

Cierto/falso (true/false): Estas son constantes booleanas que definen los niveles HIGH (alto) y LOW (bajo) cuando estos se refieren al estado de las salidas digitales. FALSE se asocia con 0 (cero), mientras que TRUE se asocia con 1, pero TRUE también puede ser cualquier otra cosa excepto cero. Por lo tanto, en sentido booleano, -1, 2 y -200 son todos también se define como TRUE. (esto es importante tenerlo en cuanta).

Ejemplo:
```
if (b == TRUE);
{
ejecutar las instrucciones;
}
```

high/low: Estas constantes definen los niveles de salida altos o bajos y se utilizan para la lectura o la escritura digital para las patillas. ALTO se define como en la lógica de nivel 1, ON, ó 5 voltios, mientras que BAJO es lógica nivel 0, OFF, o 0 voltios.

Ejemplo: *digitalWrite(13, HIGH); // activa la salida 13 con un nivel alto (5v.)*

input/output: Estas constantes son utilizadas para definir, al comienzo del programa, el modo de funcionamiento de los pines mediante la instrucción *pinMode* de tal manera que el pin puede ser una entrada INPUT o una salida OUTPUT.

Ejemplo: *pinMode(13, OUTPUT); // designamos que el PIN 13 es una salida*

Funciones de Entrada y Salidas Digitales

pinMode(pin, modo): Es usada para configurar un pin dado para comportarse como entrada o salida. Se utiliza en la sección Setup. Los Modos pueden ser: **INPUT** (Entrada) y **OUTPUT** (Salida). Todos los pines de Arduino son de entrada por lo que utilizando la función pinMode(), es entendido que es entrada.

Ejemplo: *pinMode (12,OUTPUT)*

digitalWrite(pin, valor): Envía al ´pin´ definido previamente como OUTPUT el valor HIGH o LOW (poniendo en 1 o 0 la salida). El pin se puede especificar ya sea como una variable o como una constante (0-13).

Ejemplo: *digitalWrite(pin, HIGH); // deposita en el 'pin' un valor HIGH (alto o 1)*

digitalRead(pin): Lee el valor de un pin (definido como digital) dando un resultado HIGH (alto) o LOW (bajo). El pin se puede especificar ya sea como una variable o una constante (0-13).

Ejemplo: *valor = digitalRead(Pin); // hace que 'valor sea igual al estado leído en ´Pin´*

Funciones de Control de Tiempo

delay(ms): Detiene la ejecución del programa la cantidad de tiempo en ms que se indica en la propia instrucción. De tal manera que 1000 equivale a 1seg.

Ejemplo: *delay(1000); // espera 1 segundo*

millis(): Devuelve el número de milisegundos transcurrido desde el inicio del programa en Arduino hasta el momento actual. Normalmente será un valor grande (dependiendo del tiempo que este en marcha la aplicación después de cargada o después de la última vez que se pulsó el botón “reset” de la tarjeta).

Ejemplo: *valor = millis(); // valor recoge el número de milisegundos*

Proteus

La amplia gama de la librería de dispositivos, además de otras herramientas virtuales, del simulador de circuitos Proteus hace de uso esencial para monitorear el funcionamiento de un microcontrolador.

Los modos a usar más frecuentes serán:

- **Modo Componentes**: Selecciona los componentes a utilizar. Ver **Figura.**

Modo Componentes

- **Modo Terminal**: Selecciona Alimentación (Power), Tierra (Ground) y terminales de entrada y salida (Input y Output, respectivamente). Por default, los componentes en Proteus funcionan omitiendo los pines de alimentación y tierra. Ver **Figura.**

Modo de Terminal

- **Modo Generador de Señales**: Permite añadir señales DC, Pulso configurable, flanco (Edge) de subida o caída, señal de reloj, patrones de señales digitales, etc. Ver **Figura.**

Modo Generador de Señales

- **Instrumentos**: Dispone de los equipos de medición más utilizados (Osciloscopio, Voltímetro, Amperímetro, etc.) y herramientas propias de Proteus (I2C Debugger, SPI Debugger, etc.). Ver **Figura.**

Modo de Instrumentos

5. Prácticas de Laboratorio

Practica #1 Led Blink

Código de Arduino

En esta primer practica se hará el parpadeo de un led (Blink) a través de un pin de la tarjeta Arduino. El código para esta práctica es el siguiente:

```
//Practica para apagar y encender un LED
int ledPin = 23;//Esta es la variable que guardara el numero del pin

void setup() {
  pinMode(ledPin, OUTPUT);//Define como salida el numero del pin
}

void loop() {
  digitalWrite(ledPin, HIGH);  //Enciende el led en el nivel High (5V)
  delay(1000);                 // Espera un segundo
  digitalWrite(ledPin, LOW);   // Enciende el led en el nivel Low (0V)
  delay(1000);                 // Espera un segundo
}
```

Simulación en Proteus

Para la simulación de esta primera práctica (Ver **Fig. 1**), haremos uso de la librería de Arduino para Proteus, que previamente se habrá dado, el uso de un Logic Probe para representar un led y el archivo generado por el IDE con extensión .hex.

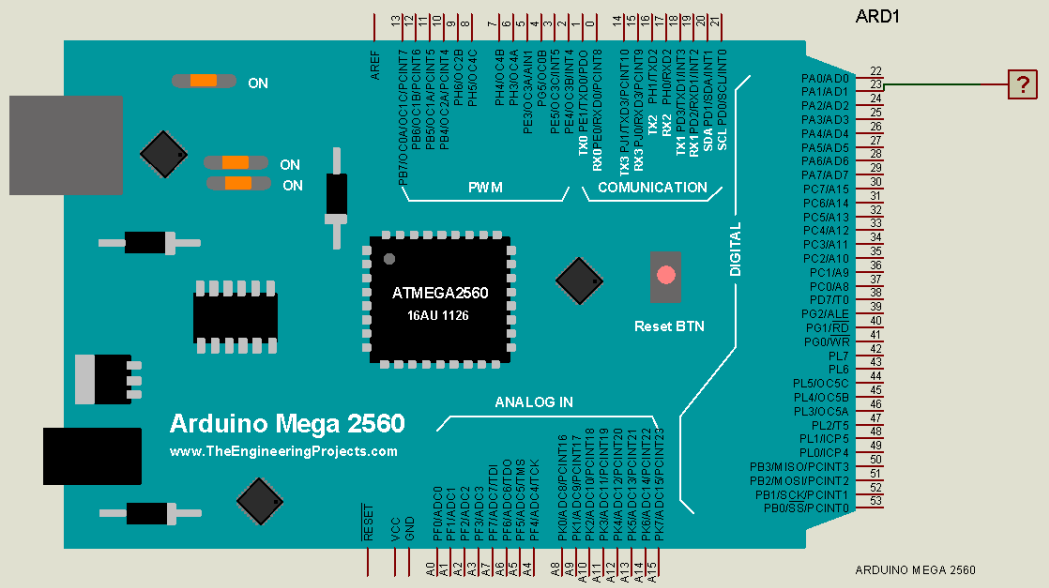
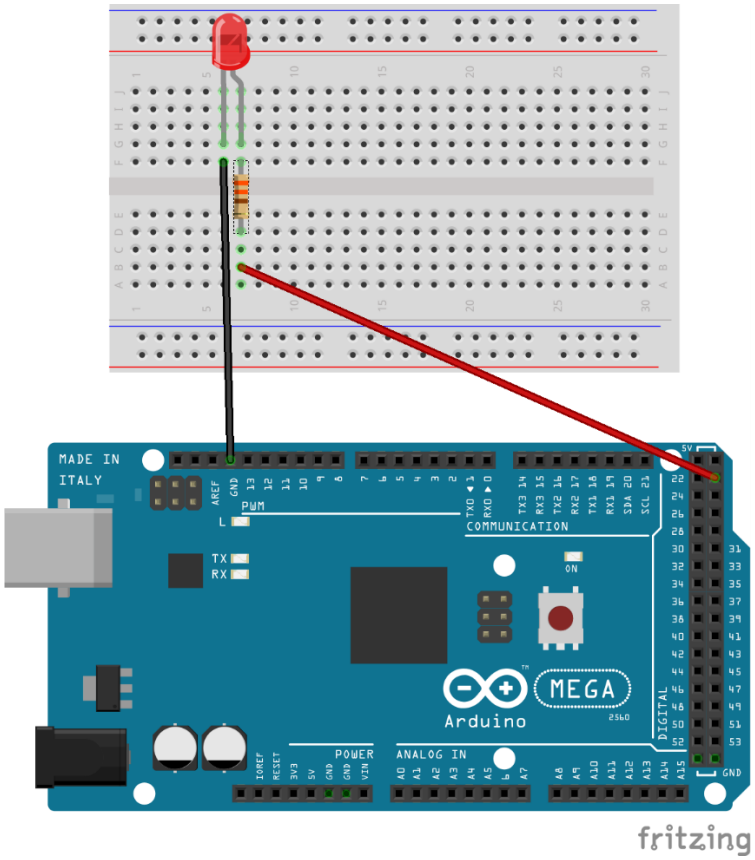


Fig. 1 Simulacion de Led Blink

El archivo compilado será cargado en la tarjeta modelo de Arduino del Simulador y después se correrá la simulación para su respectiva ejecución. El resultado será el parpadeo del led en el pin 23 por un segundo tanto en el estado Alto como en el Bajo.

Montaje del Circuito

El circuito previamente simulado en Proteus, tendrá que ser montado a como se muestra en la Figura, teniendo en cuenta dos cosas: El uso adecuado de la tabla de nodos y la identificación de Ánodo y Cátodo del led. El resultado será el mismo que la simulación.



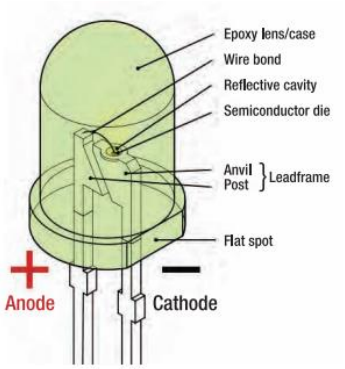


Fig. 2 Diagrama de Montaje Led Blink

Actividad

Realice cambios en los tiempos en la instrucción delay() tanto para el estado alto como para el estado en bajo y anote lo ocurrido. ¿Se podrá utilizar esta misma instrucción para retardos mayores de horas? Si no es así, indique como se haría esta modificación.

Practica #2 Led Blink de varios pines

Código de Arduino

En esta segunda práctica se mostrará como encender varios leds a la misma vez usando distintos pines de la placa de Arduino.

```
//Practica para apagar y encender varios LEDs
int pulsador = 2;
int led[] = {3, 4, 5, 6, 7};
int numLeds = 5;
int tiempoRetardo = 400;
int pinSalida;

void setup() {
  // put your setup code here, to run once:
  pinMode(pulsador, INPUT);
  for (pinSalida = 0; pinSalida < numLeds; pinSalida++)
  {
    pinMode(led[pinSalida], OUTPUT);
  }
}

void loop() {
  // put your main code here, to run repeatedly:
  boolean estadoPulsador = digitalRead(pulsador);
  if (estadoPulsador)
  {
    for (pinSalida = 0; pinSalida < numLeds; pinSalida++)
    {
      digitalWrite(led[pinSalida], HIGH);
    }
    delay(tiempoRetardo);

    for (pinSalida = 0; pinSalida < numLeds; pinSalida++)
    {
      digitalWrite(led[pinSalida], LOW);
    }
    delay(tiempoRetardo);
  }
}
```

Montaje del Circuito

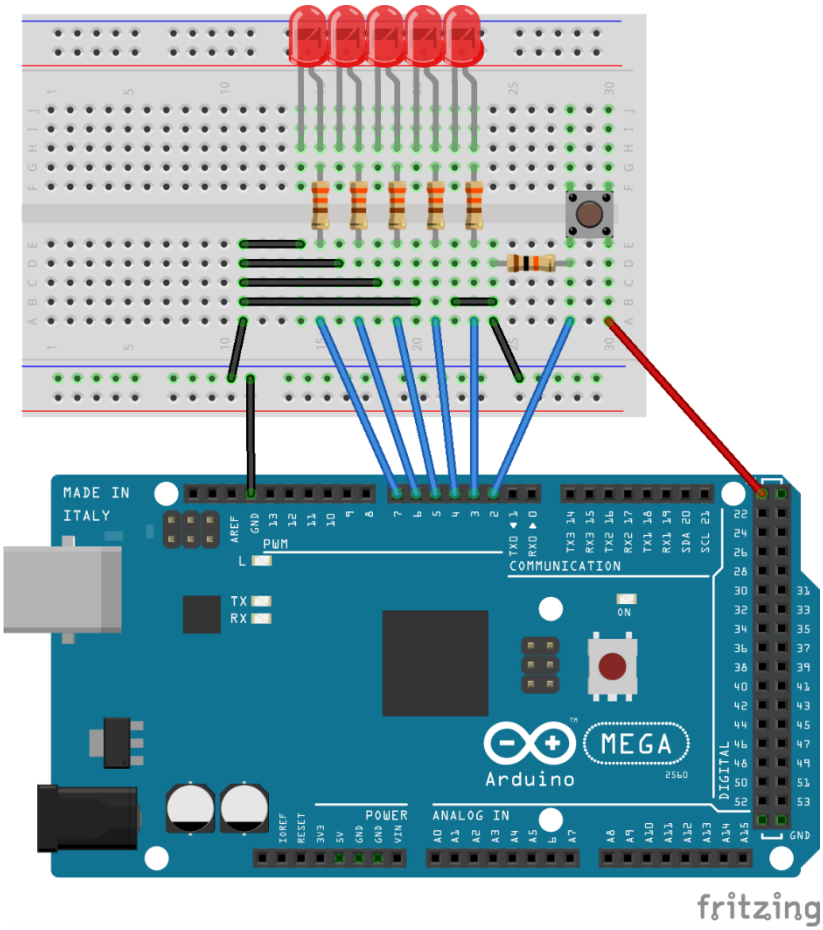


Fig. 3 Diagrama de Montaje Led Blink de varios pines.

Actividad

Del código anterior que nuevas instrucciones de programación observó. Diga cómo estas nuevas instrucciones se utilizaron para encender los leds.

6. Actividades Propuestas

Realizar la Simulación en Proteus de un semáforo Peatonal, de tal forma que el semáforo vehicular siempre este verde mientras que el semáforo peatonal está en rojo. El botón del semáforo peatonal hará ese cambio de luces hasta que pueda ser apretado y haber comprobado que han pasado más de 5 segundos después del último cambio de luces. Los tiempos de encendido de los leds, lo determinaran según la importancia del color en él.