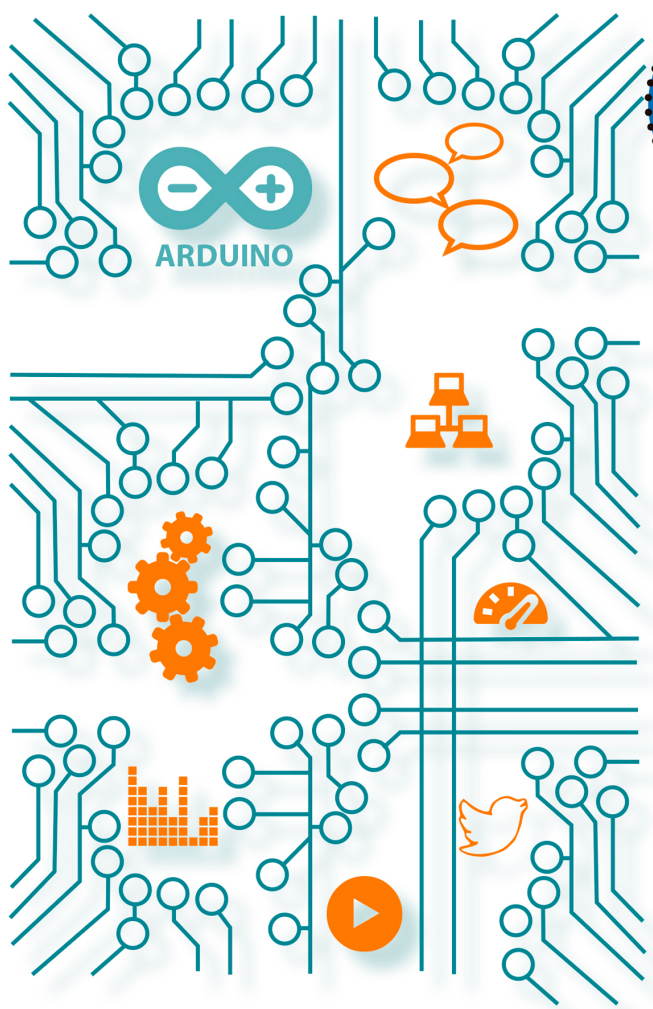
ARDUINO

MONITOREO, CONTROL Y ADQUISICIÓN DE DATOS CON ARDUINO Y VISUAL BASIC .NET

Rubén Oliva Ramos



Incluye Material WEB

Alfaomega

Marcombo

Monitoreo, control y adquisición de datos con Arduino y Visual Basic .NET

Rubén Oliva Ramos

Monitoreo, control y adquisición de datos con Arduino y Visual Basic .NET

Rubén Oliva Ramos

Alfaomega

Marcombo

Monitoreo, control y adquisicón de datos con arduino y Visual Basic .NET
Rubén Oliva Ramos

Primera edición: febrero 2017
ISBN: 978-607-622-757-2

Primera edición: MARCOMBO, S.A. 2018

ISBN: 978-84-267-2567-7
D.L.: B-20651-2017

Impreso en Prodigitalk
Printed in Spain

Acerca del autor

Rubén Oliva Ramos

Especialista en desarrollo de software para Arduino y Raspberry Pi con .NET; así como en plataformas de uso libre, desarrollo de aplicaciones para control y monitoreo de procesos a distancia, aplicaciones móviles para monitoreo y control, Sistemas SCADA, tecnologías Android, iOS, Windows Phone, Sitios web y Servicios web aplicados a las plataformas de desarrollo de Arduino y Raspberry Pi con diferentes Frameworks y aplicaciones para la nube (cloud services).

- Ingeniero en Sistemas Computacionales por el Instituto Tecnológico de León.
- Maestro en Ingeniería de Sistemas Electrónicos y Computacionales por la Universidad de la Salle Bajío en León, Guanajuato.
- Especialista en teleinformática y redes por la Universidad de la Salle Bajío en León, Guanajuato.
- Docente a nivel bachillerato en el Centro de Bachillerato Tecnológico Industrial y de Servicios No. 225 (CBTis 225) en León, Guanajuato, en la especialidad de Mecatrónica y Programación, dando cursos de electrónica, automatización, microcontroladores, robótica y control, desarrollo de aplicaciones con Android, bases de datos y plataformas web para monitoreo y control; lo cual ha llevado a cabo desde el año 2011 hasta la actualidad.
- Desde el 2008 hasta la actualidad ha sido docente en la Universidad de La Salle Bajío a nivel posgrado en la Especialidad en Mecatrónica y en la maestría en Diseño e Ingeniería de Sistemas Mecatrónicos, donde también participó en el diseño curricular y en la actualización del plan de estudios; en esta misma institución, además, imparte clases a nivel licenciatura.
- Participación en proyectos de investigación y de vinculación con empresas a través del sistema conacyt en colaboración con la Universidad de la Salle Bajío.
- Desarrollo de múltiples prototipos de sistemas mecatrónicos y robóticos.
- Fundador de Edu-training, donde actualmente brinda capacitación y consultoría en cursos de formación en las áreas de automatización y electrónica, sistemas de monitoreo a distancia y tecnología aplicada.
- En el 2014 fue coordinador de cursos de educación continua en las áreas de mecatrónica, dispositivos de control y automatización en la Universidad de la Salle Bajío.
- Del 2009 al 2011 llevó a cabo la tarea de director de la Facultad de Ingeniería Mecatrónica en la Universidad de León.

Dedicatoria

A Dios por darme la sabiduría y el ejemplo para seguir y cumplir mis sueños, sin Él no los podría llevar a cabo; gracias por darme todo lo que tengo y soy.
A mi esposa Mayte y a mis hijos Rubén y Darío; gracias por sus vidas, por su comprensión y amor. Los amo mucho.
A mis papás Rubén y Rosalía por su gran ejemplo y dedicación a lo largo de mi vida; muchas gracias por ser mis papás. Los quiero mucho.
A mis hermanos Tomás y Rosalía; gracias por su comprensión y apoyo.

Contenido

Plataforma de contenidos interactivos

Para tener acceso al material de la plataforma de contenidos interactivos del libro: *Monitoreo, control y adquisici·n de datos con Arduino y Visual Basic .NET*, 1a. edición, siga los pasos de la primera página del libro.

NOTA: Se recomienda respaldar los archivos descargados en un soporte físico.

Introducción

Capítulo 1. Bienvenido a Arduino y Visual Basic .NET. A lo largo de este capítulo se hace una introducción al manejo de la plataforma de Arduino, a las instrucciones básicas de programación, a las configuraciones para preparar el sistema y a la comunicación para poder llevar a cabo los proyectos que se presentan en los siguientes capítulos.

Capítulo 2. Aspectos generales de la programación en Visual Basic .NET. En este apartado se explica la programación en Visual Basic .NET requerida para realizar las interfaces gráficas hombre-máquina; además, se realizará un programa con el que se comunicará Arduino con una pantalla hecha en .NET.

Capítulo 3. Estación meteorológica de monitoreo con Arduino y Visual Basic .NET. Bajo la guía de esta sección se conectará un sensor de temperatura y humedad DTH11 a una fotorresistencia, esto con el objetivo de realizar una estación meteorológica y enviar los datos capturados a una pantalla conectada a una PC a través del puerto serial. Además, se enlazará una pantalla LCD 20x4 y el módulo I2C para ejecutar la interfaz de la pantalla con la placa de Arduino, con lo cual se visualizarán constantemente los datos.

Capítulo 4. Detección de presencia inalámbrica con módulos XBEE. A lo largo de este capítulo se explica el uso de los módulos de comunicación XBEE para realizar una alarma inalámbrica integrando las tecnologías de Arduino y Visual Basic .NET; con ello, se buscará conectar un sensor de movimiento PIR y efectuar la conexión de los módulos de comunicación con el software de la interfaz.

Capítulo 5. Control de las luces desde una interfaz HMI. Esta sección abarca el control de dispositivos desde la interfaz, gracias a lo cual se enlazará un módulo de relevador al Arduino que se podrá controlar desde la interfaz HMI; por otro lado, se podrán también controlar dispositivos de potencia, tal como la conexión de una lámpara .

Capítulo 6. Control de un motor de corriente directa. En este capítulo se conectará un motor de corriente directa al Arduino a través de un puente H para controlar el giro y la velocidad; con ello, se realizará la interfaz requerida para el control del dispositivo.

Capítulo 7. Sistema de alarma inalámbrica. Este capítulo explica cómo realizar una alarma a través de comunicación inalámbrica; asimismo se llevará a cabo el diseño de la interfaz web para monitoreo del sistema.

Capítulo 8. Estación de Registro de datos. En este capítulo se integran los conocimientos de los proyectos realizados previamente, aquello se efectúa a través de un registro de datos de forma local. Por otro lado, se desarrollará como proyecto integrador un registro de información (*datalogger*) en una base de datos y se realizará un reporte para consultarlo en Excel.

Capítulo 9. Desarrollo de proyectos del Internet de las cosas basados en el Shield Ethernet de Arduino. En éste se desarrollarán y aplicarán proyectos adaptados al Internet de las cosas para alcanzar la comprensión de las tecnologías enfocadas en determinadas áreas.

Capítulo 10. Prototipo de un Sistema de Control Supervisorio y Adquisición de datos a distancia (SCADA). Apartado en el cual se aplicarán los conceptos de los sistemas de monitoreo y control, desarrollando una aplicación de un sistema SCADA utilizando la tecnología .NET y la plataforma de Arduino.

Capítulo 11. Rastreador móvil por medio de GSM/GPRS y GPS. Gracias a este capítulo se desarrollará un proyecto de tecnologías aplicadas para sistemas web en la nube con el uso de la placa de GSM/GPRS y GPS; igualmente, se podrá elaborar un prototipo de un tracker móvil con tecnología de programación y Arduino con el objetivo de aplicarlo en un ejemplo real y práctico.

Capítulo 12. Robot controlado inalámbricamente. En esta sección se aplicará todo lo aprendido para desarrollar un prototipo de robot controlado inalámbricamente con las tecnologías de Arduino y diferentes aplicaciones de robótica; ello con el objetivo de que se puedan ejecutar posteriormente proyectos mecatrónicos propios.

Capítulo 1

Bienvenido a Arduino y Visual Basic .NET

Objetivos

En este capítulo se asentarán las bases para utilizar la plataforma Arduino, se describirán las instrucciones básicas de programación y se mencionarán las configuraciones tanto para preparar el sistema como para establecer la comunicación, esto con la finalidad de que se sienten los cimientos para realizar los proyectos que se presentan en los siguientes capítulos.

1.1 Introducción

Este capítulo es una introducción a Arduino y a la programación entorno a él necesaria e importante para establecer una comunicación serial con los diferentes dispositivos externos para el control y la adquisición de datos. Dicha comunicación serial será fundamental en los proyectos de los siguientes capítulos. A continuación se explicará cómo se lleva a cabo la comunicación serial entre Visual Basic .NET y la tarjeta Arduino UNO, para enlazar los controles de los formularios y el aspecto gráfico con el hardware.

1.2 Entorno de programación de Arduino IDE

Arduino es un sistema de desarrollo para microcontroladores de la firma ATMEL. Fue desarrollado en Italia y está compuesto por un software editor-compilador (basado en Processing) en donde se escribe un programa en lenguaje C (basado en Wiring), así como un hardware que consiste en un microcontrolador ATMEL, el cual tiene precargado un sistema operativo (Bootstrap) que permite su programación directa *in-circuit* a través de señales seriales de comunicación.

1.2.1 Funciones básicas iniciales

Función Setup. Contiene todas las configuraciones iniciales del programa. Aquí se incluyen las condiciones iniciales para la operación de algunas instrucciones o librerías que se agregarán en el programa.

Función Loop. En ella se colocan todas las instrucciones que van a realizarse en forma repetitiva (*loop* significa lazo o bucle).

Además de estas funciones, se pueden agregar otras creadas por el usuario, aunque las anteriores no pueden omitirse en el programa. La estructura general de una función se describe en el ejemplo 1.1:

Ejemplo 1.1 Estructura general de una función.

```
void setup( )
{
 Instrucciones;
}
void loop( )
{
Instrucciones;
}

void usuario( )
{
Instrucciones;
}
```

Aquí se puede observar que cada función inicia con void, seguida del nombre de la función. En el caso de setup y loop, no es posible cambiar su nombre. En el caso de una función de usuario, es posible colocar cualquier nombre, empleando letras, números, guiones medio y bajo, aunque no acepta signos ni caracteres especiales. Seguido del nombre de la función se colocan los paréntesis (( )). Entre las llaves ({ }) se colocan los comandos o instrucciones; al final de cada una, se debe agregar el punto y coma (;).

Es importante que en cada función e instrucción se respeten las mayúsculas y minúsculas, así como los espacios y símbolos que deben emplearse; de lo contrario, aparecerá un código de error.

1.2.2 Agregar comentarios

Es posible agregar comentarios a nuestro programa. Para ello, se colocan dos diagonales (//) seguidas del texto con el comentario que se desea mostrar. Pueden colocarse en cualquier parte del programa. (Ver ejemplo 1.2.)

Ejemplo 1.2 Agregar comentarios al programa.

```
Void setup( ){
pinMode(buttonPin, INPUT);    // la variable buttonPin es
configurada como entrada.
pinMode(ledPin, OUTPUT);// la variable ledPin es configura-
da como salida.
}
```

También es posible escribir un bloque de texto, colocando diagonal y asterisco (/*) al inicio del bloque y asterisco diagonal (*/) al final del bloque. (Ver ejemplo 1.3.)

Ejemplo 1.3 Escribir bloques de texto.

```
/*
Programa de Ejemplo para activar un led
En intervalos de 0.s segundos Empleando el led integrado
a la tarjeta Arduino Duemilanove (pin 13)
*/
```

1.3 Conceptos básicos de programación

En esta sección se revisarán algunos conceptos básicos de programación.

1.3.1 Declaración de variables y constantes

Una regla básica para emplear variables y constantes es que siempre deben declararse para que puedan utilizarse. También es importante indicar el tipo de variable según su

formato numérico. En función de su capacidad numérica, las variables y constantes pueden ser de tipo:

Byte. Valores numéricos con capacidad de 8 bits (0-255). (Ver ejemplo 1.4.)

Ejemplo 1.4 Declaración de variables tipo byte.

```
// Asigna a la variable tiempo una longitud de byte (0-
255) sin valor inicial.
byte tiempo;
// Asigna a la variable temperatura una longitud de byte
(0-255) iniciando con el valor de 0.
byte temperatura = 0;
```

Int. Entero. Valores numéricos con capacidad de 16 bits con signo (-32768 y 32767). (Ver ejemplo 1.5.)

Ejemplo 1.5 Declaración de variables tipo int.

```
// Asigna a la variable como_se_llame, una longitud de
int, sin valor inicial.
int como_se_llame;

// Asigna a la variable como_se_llame, una longitud de
int, iniciando con el valor de 0.
int como_se_llame  = 1765;
```

Long. Extendido. Valores numéricos enteros con capacidad de 32 bits (-2147483648 a 2147483647). (Ver ejemplo 1.6.)

Ejemplo 1.6 Declaración de variables tipo long.

```
// Asigna a la variable cualquier_nombre, una longitud de
long, sin valor inicial.
long cualquier_nombre;

// Asigna a variable cualquier_nombre, una longitud de
long, iniciando con el valor de 0.
long cualquier_nombre  = 150000;
```

Float. Flotante. Valores numéricos con fracción decimal con capacidad de 32 bits (3.4028235E +38 y 3.4028235E -38). Es importante destacar que los resultados de las operaciones matemáticas sólo muestran dos decimales con redondeo. (Ver ejemplo 1.7.)

Ejemplo 1.7 Declaración de variables tipo float.

```
// Asigna a la variable ponle_un_nombre, una longitud de
int, sin valor inicial.
float ponle_un_nombre;

// Asigna a variable ponle_un_nombre, una longitud de
int, iniciando con el valor de 0.
float ponle_un_nombre  = 3.14;
```

1.3.2 Arreglos con variables

También es posible hacer arreglos con variables. De esta forma, se pueden asignar variables con el mismo nombre pero asignando una posición; o bien, realizar matrices de valores indicando el tipo de dato, el tamaño y asignar valores a una posición específica. (Ver ejemplo 1.8.)

Ejemplo 1.8 Arreglos con variables.

```
/*
En este arreglo, el primer valor se encuentra en la posi-
ción 0 de mi variable, el segundo valor, en la posición
1, el tercer valor, en la posición 2, etcétera.
*/

int mi_arreglo[]  = {10, 50, otra_variable}

En donde:

mi_arreglo[0]  = 10
mi_arreglo[1]  = 50
mi_arreglo[2]  = lo que vale otra variable
```

De igual forma, es posible declarar una matriz de valores indicando el tipo de datos y el tamaño, y, posteriormente, asignar valores a una posición específica. (Ver ejemplo 1.9.)

Ejemplo 1.9 Declarar matrices de valores.

```
int mi_arreglo[5];        // declara un arreglo de ente-
ros de 6 posiciones (0 - 5).
mi_arreglo[3]  = 10;   // asigna el valor 10 a la posi-
ción 4 del arreglo.
```

Recordemos que el valor que se coloca entre corchetes ([ ]) representa el máximo valor de elementos en el arreglo, comenzando desde el cero. El cero cuenta como la posición uno.

Es posible asignar a una variable una determinada terminal de la tarjeta, definida como *entrada* o *salida*, tanto analógica como digital. (Ver ejemplo 1.10.)

Ejemplo 1.10 Asignar una determinada terminal de la tarjeta a una variable.

```
Int ledPin = 13;                                    // asigna
la variable ledPin a la terminal (pin) 13 de la tarjeta
Arduino.
Int Valor_Analógico  =  0;          // asigna a la varia-
ble Valor_Analógico al pin 0 de la tarjeta.
```

1.3.3 Operaciones aritméticas

Arduino puede manejar las cuatro operaciones aritméticas básicas: suma, resta, multiplicación y división. Para efectuar cualquiera de estas operaciones, es posible emplear números directos o variables. Cuando se emplean variables, es importante declarar cada variable con el formato numérico deseado (int, long, float, etc.). Hay que recordar que el único formato que puede realizar operaciones con valores decimales es el float. (Ver ejemplo 1.11.)

Ejemplo 1.11 Operaciones aritméticas.

```
X = y + 5;
Z = x - 30;
P = valor * 50;
R = valor1/valor2;
```

Si se requiere realizar alguna ecuación, se escribe la operación entre paréntesis (( )). (Ver ejemplo 1.12.)

Ejemplo 1.12 Ecuaciones.

```
Temperatura = (valor_Analógico * 500) / 1023
```

1.3.4 Asignaciones compuestas

Las asignaciones compuestas combinan una operación aritmética con una variable asignada. Éstas comúnmente se utilizan en las condiciones de ciclos, tal como se describe más adelante (ver ejemplo 1.13). Estas asignaciones compuestas pueden ser las siguientes:

```
x ++    // igual que x = x + 1, o incrementar x en + 1
x --    // igual que x = x - 1, o decrementar x en -1
x += y    // igual que x = x + y, o incrementar x en +y
x -= y   // igual que x = x - y, o decrementar x en -y
x *= y   // igual que x = x * y, o multiplicar x por y
x /= y   // igual que x = x / y, o dividir x por y
```

Ejemplo 1.13 Asignaciones compuestas.

```
x * = 3 hace que x se convierta en el triple del antiguo
valor x y, por lo tanto, x se reasigna al nuevo valor.
```

1.3.5 Operadores de comparación

Las comparaciones entre variables y constantes se utilizan con frecuencia en las estructuras condicionales if para establecer si una condición es verdadera, como se mostrará más adelante. Los operadores de comparación en Arduino son los siguientes:

```
x == y              // x es igual a y
x != y               // x no es igual a y
x < y            // x es menor que y
x > y           // x es mayor que y
x <= y  // x es menor o igual que y
x >= y  // x es mayor o igual que y
```

1.3.6 Operadores lógicos

Comúnmente, los operadores lógicos son una forma de comparar dos expresiones y dar como respuesta un estado *verdadero* o *falso*, dependiendo del operador lógico utilizado. Existen tres operadores lógicos: AND (&&), OR (||) y NOT (!), los cuales se utilizan generalmente en condicionales de tipo if. A continuación se muestran algunos ejemplos. (Ver ejemplo 1.14.)

Ejemplo 1.14 Ejemplos de operadores lógicos AND, OR y NOT.

Lógica AND:
if (x > 0 && x < s) // condición es verdadera sólo si las dos expresiones son ciertas

Lógica OR:
if (x > 0 || y > 0) // condición es verdadera si cualquiera de las expresiones es cierta

Lógica NOT:
if (!x > 0) // condición es verdadera solo si la expresión (x > 0) es falsa

1.3.7 Constantes

El lenguaje de programación de Arduino contiene valores predeterminados, que son llamados "constantes". Estos valores, realizan funciones específicas que pueden ser empleadas, principalmente, por valores digitales, o bien, por valores numéricos.

En el caso de una constante numérica, se antepone la palabra *const*, seguida de su formato numérico (int, float, byte, etc.). Finalmente, se le asigna un valor. (Ver ejemplo 1.15.)

Ejemplo 1.15 Ejemplos de constantes.

```
const int nombre_variable = 10;    // asigna el valor de
10 a la constante declarada
const float b = 3.14;// asigna el valor de 3.14 a la cons-
tante llamada b
const int ledPin = 13;// asigna el valor de 13 a la cons-
tante ledPin
```

Cuando se emplean valores digitales –también llamados valores booleanos–, se pueden utilizar las siguientes constantes asignadas, que la programación Arduino establece directamente.

TRUE/FALSE

FALSE se define como un valor de cero. TRUE se asocia con un valor de 1; sin embargo, cualquier entero que es no-cero es true, en un sentido booleano. Es decir, -1, 2 y -200 son todos true. Es importante considerar que las constantes TRUE y FALSE se escriben en mayúsculas. (Ver ejemplo 1.16.)

Ejemplo 1.16 Constantes TRUE y FALSE.

```
if (x == TRUE);
{
ejecutar estas instrucciones;
}
```

INPUT/OUTPUT

Estas constantes se emplean al inicio del programa —dentro de la función void_setup–, para definir el sentido de una señal digital, si es de entrada —INPUT—o si es de salida —OUTPUT–. Estas constantes deberán escribirse siempre en mayúsculas. Para que puedan operar, se requiere anteponer la instrucción *pinMode*, seguida del valor de la terminal –pin– que se desea direccionar separado éste por una coma (,) del sentido de la señal; todo esto, entre paréntesis (()). (Ver ejemplo 1.17.)

Ejemplo 1.17 Constantes INPUT y OUTPUT.

```
pinMode (13,OUTPUT);  // asigna el pin 13 del Arduino
como una salida
```

También es posible que el pin sea asignado a través de una variable previamente definida. (Ver ejemplo 1.18.)

Ejemplo 1.18 Pin asignado a través de una variable previamente definida.

```
Int ledPin = 13;                                    //
asigna a la variable ledPin el valor de 13
pinMode(ledPin, OUTPUT);    // asigna el valor de la va-
riable ledPin como una salida
```

HIGH/LOW

Estas constantes también se escriben en mayúscula y establecen el estado lógico directamente de un pin de entrada/salida. HIGH establece que el nivel lógico es de 1; LOW establece que el nivel lógico es de 0. Estas constantes forman parte de las funciones de lectura –digitalRead– o escritura –digitalWrite– de valores digitales en los pines de entrada/salida de Arduino. (Ver ejemplo 1.19.)

Ejemplo1.19 Constantes HIGH y LOW.

```
digitalWrite(ledPin, HIGH); // envía a un estado lógico 1
al pin establecido en la variable
digitalWrite(ledPin, LOW);             // envía a un es-
tado lógico 0 al pin establecido en la variable
```

1.3.8 Estructuras de control

A continuación se revisarán algunas estructuras de control y sus formatos de instrucción.

Condicional if

If es una instrucción que se utiliza para determinar si una condición se ha cumplido; por ejemplo, averiguar si un valor analógico está por encima de un cierto número. Si la condición se cumple, deberá ejecutar una serie de instrucciones que se escriben dentro de llaves ({ }). Si no se cumple, el flujo del programa salta esta condición y no ejecuta las operaciones que están dentro de las llaves.

El formato de instrucción es el siguiente:

```
if (Variable Condición Valor)
{
Instrucciones a ejecutar;
```

En donde:

Variable es un valor o dato que se desea condicionar. Puede estar dado en cualquier formato para valores numéricos.

Condición es cualquiera de los operadores de comparación (==, !=, >, <, =>, =<).

Valor es el dato numérico (valor directo o variable) que se desea comparar. (Ver ejemplo 1.20.)

Ejemplo 1.20 Ejemplos de condicional if.

```
If (temperatura >= 30)      // si la variable temperatura
es mayor o igual a 30:
{
digitalWrite(ledPin, HIGH);    // activa el led conectado
en el pin designado (ledpin).
}


If (valor_actual == valor_establecido)   // si la varia-
ble valor_actual es igual a variable
{       // valor_establecido
Serial.print("Valor Alcanzado");  // envía por el serial
del Arduino el texto.
}       // "Valor Alcanzado".



If (x != 50)  // si la variable x no es igual a s0
{
digitalWrite(10, LOW)                  // apaga el pin
10.
}
```

Condicional if else

If else es una instrucción que se emplea para definir lo que deberá realizarse en el programa en caso de que una determinada condición no se cumpla. Esto equivale a decir: "Si se cumple esta condición, realiza estas operaciones; de lo contrario (else), realiza éstas". (Ver ejemplo 1.21.)

El formato de instrucción es el siguiente:

```
if (variable condición valor)
{
Instrucciones a ejecutar;
```

```
}
else
{
Instrucciones a ejecutar;
}
```

Ejemplo 1.21 Condición if else.

```
if (boton == HIGH) { // si el estado contenido en la
variable botón esta en alto
digitalWrite(13, HIGH);     //entonces, activa el pin 13.
}
else {  // de lo contrario,
digitalWrite(13, LOW);      // apaga el pin 13.
}
```

Ciclo for

Este comando realiza un cierto número de veces un ciclo repetitivo de las operaciones que se encuentran dentro de él. Una vez que termina el ciclo, el programa continúa ejecutando las instrucciones fuera de éste. Si se quiere repetir el ciclo, se puede recomenzar.

El formato de instrucción es el siguiente:

```
for (Inicialización; Condición; Expresión)
{
Instrucciones a ejecutar;
}
```

En donde:

Inicialización es el valor inicial dado a una variable. Éste es el valor de inicio del ciclo.

Condición es el operador de comparación que indica hasta dónde puede llegar el valor de la variable (valor final).

Expresión indica si el conteo de los ciclos dentro del valor inicial y final, se incrementa o decrementa. Aquí se emplea una asignación compuesta (++, --). (Ver ejemplo 1.22.)

Ejemplo 1.22 Ciclo for.

```
for (int x = 0; x < 12; x++)            // x vale desde 0
hasta 12 y se incrementa de 1 en 1.
{
```

```
Serial.println(x);   // escribe al serial el valor que
toma x
delay(500);    // espera 0.s segundos antes de enviar
otro valor
}
```

Es importante observar en el ejemplo 1.22 que la variable x se puede declarar dentro de la instrucción como un valor int (entero), o bien, puede ser previamente declarada al inicio del programa, con cualquier formato numérico.

El ejemplo 1.23 muestra cómo podemos realizar ciclos en los que se incrementen de 2 en 2 los valores declarados en la función.

Ejemplo 1.23 Ciclos en los que se incrementan de 2 en 2 los valores declarados en la función.

```
for (int x = 0; x < 12; x+=2)              // x vale
desde 0 hasta 12 y se incrementa de 2 en 2.
{
Serial.println(x);   // escribe al serial el valor que
toma x.
delay(500);    // espera 0.s segundos antes de enviar
otro valor.
}
```

En el ejemplo 1.24 se observa cómo realizar un ciclo negativo de valores –decremento– de 1 en 1.

Ejemplo 1.24 Ciclo negativo de valores.

```
for (int x = 10; x > -1; x--)              // x vale
desde 10 hasta 0 y se incrementa de 1 en 1.
{
Serial.println(x);  // escribe al serial el valor que
toma x.
delay(s00); // espera 0.s segundos antes de enviar otro
valor.

}
```

Comparador múltiple Switch Case

Esta instrucción relaciona los valores de una variable con diferentes condiciones; las compara y realiza las instrucciones indicadas para cada caso (case). Al finalizar las instrucciones que deberán ejecutarse en cada uno, se coloca una instrucción *break*, necesaria para que el programa pueda seguir su flujo normal. Al final del último case, se coloca el comando

default, para indicar que, si no se cumplió ninguno, el programa continúa en la siguiente línea. (Ver ejemplo 1.25.)

El formato de instrucción es el siguiente:

```
switch (var) {
case etiqueta: instrucciones; break;
case etiqueta: instrucciones;
break;
default:
instrucciones;
}
```

Ejemplo 1.25 Comparador múltiple Switch Case.

```
switch (var) {
case 1:                                    // cuando valor
de var sea igual a 1:
digitalWrite(13, HI  H);
break;
case 2:                                   // cuando valor de
var sea igual a 2:
digitalWrite(13, LOW);
break;
}
```

Ciclo while

Los ciclos while se ejecutan continuamente hasta que la expresión dentro del paréntesis (condición de comparación) deja de cumplirse. Algo debe modificar la variable comprobada; de lo contrario, el ciclo while nunca terminará. Lo que modifique la variable puede estar dentro del código, como una variable que se incrementa, o ser una condición externa; por ejemplo, el valor de un sensor.

El formato de instrucción es el siguiente. (Ver ejemplo 1.26):

```
while(expresión)
{
instrucciones;
}
```

Ejemplo 1.26 Ciclo while.

```
while(var <  200) { // mientras que var sea menor que 200:
Serial.pritnln(var); // envía por el serial, el valor de
var
    var++;     // incrementa en 1 el valor de var
}
```

Ciclo do while

El ciclo do while ejecuta las instrucciones especificadas, mientras que la condición indicada dentro del paréntesis (condición de comparación) se cumpla. Cuando la condición se cumple, se repite el ciclo.

El formato de instrucción es el siguiente. (Ver ejemplo 1.27):

```
do
{
Instrucciones;
}
while (condición);
```

Ejemplo 1.27 Ciclo do while.

```
{
delay(50);   // espera a que los sensores se estabilicen
x =
readSensors(); // comprueba los sensores

} while (x < 100);          //si se cumple la condición se
repite el bucle
```

Instrucción continue

Cuando se escribe esta instrucción dentro de un ciclo (do, for o while), el programa se "salta" éste y se dirige a las siguientes instrucciones fuera del ciclo. (Ver ejemplo 1.28.)

Ejemplo 1.28 Instrucción continue.

```
for (x = 0; x < 2ss; x ++)
{
if (x > 20 && x < 100){ // Si se cumple esta condición:
continue;      // se salta desde el 21 hasta el 99 del
ciclo for
}

analogWrite(10, x);
delay(s0);
}
```

1.4 Tipos de comunicación

Uno de los objetivos de este libro es explicar la comunicación serial con el Arduino, puesto que se trata de una parte fundamental, además de ser una herramienta de suma importancia para llevar a cabo los proyectos que se presentan en los siguientes capítulos. En este punto hablaremos de la comunicación Serial UART y la comunicación serial por software a través de la librería SoftwareSerial.

1.4.1 Comunicación Serial UART

Se utiliza para la comunicación entre la tarjeta Arduino y una PC u otros dispositivos. Todas las placas Arduino tienen al menos un puerto serie (también conocido como un UART o USART). Éste utiliza los pines 0 (RX) y 1 (TX), y se comunica con el ordenador a través de USB. Los pines digitales están reservados para esta función.

Se puede usar el monitor serial incorporado en el entorno de Arduino para comunicarse con una placa Arduino. Para ello, hacer clic en el botón de monitor de serie en la barra de herramientas y seleccionar la misma velocidad de transmisión que se ha programado en las funciones de velocidad de transmisión. (Ver figura 1.1.)

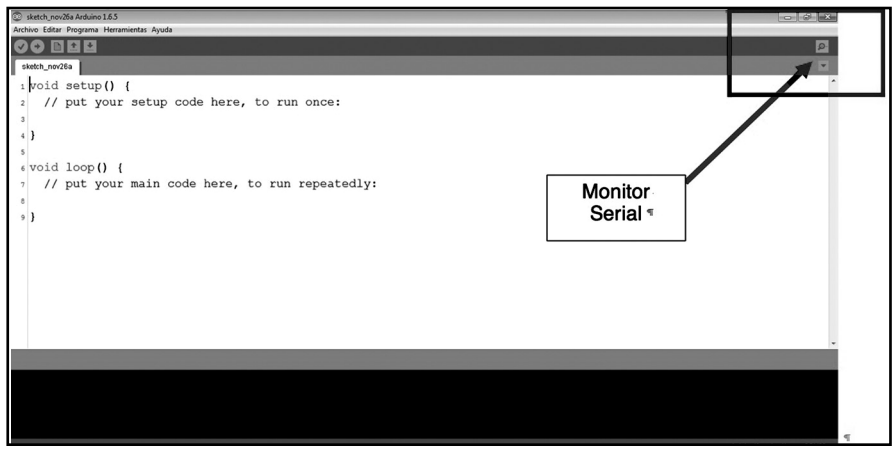


Figura 1.1 Comunicación del monitor serial con la placa Arduino.

Funciones de comunicación serial

Al emplear la comunicación serial, es importante considerar que los pines 0 (Rx) y 1 (Tx) de la placa Arduino se reservan para esta función. También es importante recordar que este puerto se utiliza para la programación del Arduino; sin embargo, mientras se emplea la función serial, no hay interferencia entre ambas operaciones.

Las funciones relacionadas con la comunicación serial son las siguientes:

```
Serial.begin( )
Serial.end( )
Serial.available( )
```

```
Serial.read( )
Serial.flush( )
Serial.print( )
Serial.println( )
Serial.write( )
```

Función Serial.begin( );

En esta función se abre el puerto serial del Arduino y se determina la velocidad de transferencia en bps (bps = bits por segundo) de datos entre dispositivos. Los valores más comunes pueden ser 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 o 115200. Esta función deberá colocarse dentro de la rutina void setup(). (Ver ejemplo 1.29.)

Ejemplo 1.29 Función Serial.begin( );

```
Serial.begin(9600);     // configura la comunicación
serial a 9600 bps.
```

Función Serial.end();

Desactiva la comunicación serial, permitiendo a los pines 0 (Rx) y 1 (Tx) utilizarse como entradas o salidas digitales. Esta función se coloca dentro de cualquier rutina en el lugar o instante en que se desea desactivar la comunicación. Para volver a reactivar la comunicación, es necesario volver a colocar en el lugar o instante deseado la función Serial.begin(bps), y así volver a inicializar la transferencia de datos.

Función Serial.available();

Esta función cuenta y guarda el número de bytes disponibles para que el puerto serie los lea. Esto se refiere a los datos que ya han sido recibidos y están almacenados en el búfer de recepción del puerto, el cual tiene una capacidad de hasta 128 bytes. Más adelante se realizará un ejercicio para ejemplificar esta función.

Función Serial.read();

Lee los datos (byte) que son recibidos por el puerto serial. Esta función requiere de una variable para cargar el valor del dato recibido por la función. Si la variable se declara como int, el valor que recibe la función es el equivalente al valor decimal del valor ASCII enviado. Si la variable se declara como char, se recibe el valor directo del valor ASCII enviado. (Ver ejemplo 1.30.)

Ejemplo 1.30 Función Serial.read();

```
datoRecibido  =  Serial.read();  // guarda el dato leído
por el puerto serial en la variable datoRecibido.
```

Función Serial.flush();

Con esta función se vacía el búfer de entrada de datos del puerto serial.

Función Serial.print();

Imprime –o envía– los datos al puerto serie como texto ASCII. Esta función puede tomar diversas opciones. Los números se imprimen mediante un juego de caracteres ASCII para cada dígito. Los valores de tipo float se imprimen en forma de dígitos ASCII con dos decimales por defecto. Los valores tipo byte se envían como un único carácter. Los caracteres y las cadenas se envían como son. (Ver ejemplo 1.31.)

Ejemplo 1.31 Función Serial.print();

```
Serial.print(15);                    // imprime "15"
Serial.print(3.1416); // imprime "3.1416"
Serial.print(byte(65));    // imprime "A" (cuyo código
ASCII es 65)
Serial.print('A');   // imprime "A"
Serial.print("Hola");      // imprime "Hola"
```

Un segundo parámetro opcional especifica la base (formato a usar). Los valores permitidos son BYTE, BIN (binarios o base 2), OCT (octales o base 8), DEC (decimales o base 10) y HEX (hexadecimales o base 16).

```
Serial.print(65, BYTE);     // imprime "A"
Serial.print(65, BIN);      // imprime "1000001"
Serial.print(65, OCT):      // imprime "101"
Serial.print(65, DEC);      // imprime "65"
Serial.print(65, HEX);      // imprime "41"
```

Para imprimir números enteros o de punto flotante, hay que agregar un valor separado por una coma. Esto especifica el número de dígitos a usar (de izquierda a derecha), empezando en la posición 0. (Ver ejemplo 1.32.)

Ejemplo 1.32 Imprimir números enteros o de punto flotante.

```
Serial.print(3.1416, 0);    // imprime "3"
Serial.print(3.1416, 2);    // imprime "3.14"
Serial.print(3.1416, 4);    // imprime "3.1416"
```

Para agregar un tabulador, hay que agregar un comando "\t" dentro de la función. (Ver ejemplo 1.33.)

Ejemplo 1.33 Agregar un tabulador.Serial.print("Valor A: ");

```
Serial.print("Valor A: "); Serial.print("\t"); // el
siguiente texto se imprime  tabulado.
Serial.print("Valor B: ");
```

Para aplicar un retorno de carro y avance de línea, hay que agregar un comando "\n" dentro de la función. (Ver ejemplo 1.34.)

Ejemplo: 1.34 Aplicar un retorno de carro y avance de línea.

```
Serial.print("Resultado A:");
Serial.print("\n")                          // el siguiente
texto se imprime al inicio de la siguiente línea.
Serial.print("Resultado B:");
```

Función Serial.println();

Imprime o envía los datos al puerto serie como texto ASCII, seguido de un comando de retorno de carro y avance de línea. Para esta función aplican todas las opciones anteriores de Serial.print(). (Ver ejemplo 1.35.)

Ejemplo 1.35 Función Serial.println();

```
Serial.println("Hola");     // imprime "Hola" y envía el
comando de retorno de carro y avance de línea.
```

Función Serial.write();

Esta función es muy similar a Serial.print(), a diferencia de que, si se envían valores numéricos en forma directa, los interpretará como valores ASCII. Se recomienda, entonces, emplear la función Serial.print(). Los comandos "\t" (tabulador) y "\n" (retorno de carro y avance de línea) aplican de igual forma. (Ver ejemplo 1.36.)

Ejemplo 1.36 Función Serial.write();

```
Serial.write(65);     // envía la letra A (ASCII 6s = A).
Serial.write("Hola");   // envía el texto "Hola".
Serial.write("\t");       // aplica tabulador.
Serial.write("\n");     // aplica retorno de carro y
avance de línea.
```

1.4.2 Comunicación Serial por software

El hardware Arduino ha incorporado un soporte para la comunicación en serie entre los pines 0 y 1, que también se conectan a la computadora a través de la conexión USB.

La comunicación serial por software se ha desarrollado para permitir la comunicación en serie en otros pines digitales del Arduino. Es decir, se usa el software para replicar la funcionalidad: es posible tener por software múltiples puertos serie con velocidades de hasta 115200 bps. Un parámetro permite la señalización invertida para dispositivos que requieren dicho protocolo.

1.4.3 Librería SoftwareSerial

La biblioteca cuenta con las siguientes limitantes:

- Si utiliza software de múltiples puertos serie, se reciben los datos uno a la vez.
- No todos los pines funcionan en el Arduino Mega, por lo que solamente se puede utilizar para RX: 10, 11, 12, 13, 14, 15, 50, 51, 52, 53, A8 (62), A9 (63), A10 (64), A11 (65), A12 (66), A13 (67), A14 (68), A15 (69).

1.5 Tarjetas Arduino y los puertos de comunicación

En las siguientes secciones se describirán los tipos de tarjetas Arduino y sus puertos de comunicación.

1.5.1 Arduino UNO

Utiliza los pines digitales 0 (RX) y 1 (TX) para establecer comunicación con los dispositivos.

1.5.2 Arduino MEGA

El Arduino Mega tiene tres puertos seriales adicionales: Serial1, en los pines 19 (RX) y 18 (TX); Serial2, en los pines 17 (RX) y 16 (TX); Serial3, en los pines 15 (RX) y 14 (TX). Para utilizar estos pines para comunicarse con la PC, se requiere de un adaptador adicional de USB a serie, ya que no están conectados al adaptador de los Mega-USB a serie.

Para utilizarlos para comunicarse con un dispositivo serie TTL externo, conectar el pin TX al pin RX del dispositivo; el RX a TX pin del dispositivo, y el suelo del Mega a tierra del dispositivo. Estos pines no se deben conectar directamente a un puerto serie RS232, ya que operan a ±12 V y pueden dañar la placa Arduino.

1.5.3 Arduino Due

El Arduino Due tiene tres puertos seriales TTL de 3.3 V Serial1, en los pines 19 (RX) y 18 (TX); Serial2, en los pines 17 (RX) y 16 (TX); Serial3, en los pines 15 (RX) y 14 (TX). Los pines 0 y 1 también están conectados a los pines correspondientes del USB

-to- Serial TTL chip de ATmega16U2, que está conectado al puerto USB de depuración. Además, hay un puerto USB-SERIAL nativo en el chip SAM3X.

1.5.4 Arduino YUN

Utiliza los pines digitales 0 (RX) y 1 (TX) para establecer comunicación con los dispositivos.

Para fines prácticos, en los proyectos se utilizará la tarjeta Arduino UNO para establecer comunicación serial con los diferentes dispositivos.

1.6 Aspectos de comunicación serial con Visual Basic .NET

Para establecer comunicación con una PC a través de una interfaz HMI (interfaz hombre-máquina), se utilizará el lenguaje de programación de Microsoft Visual Basic .NET. Para llevar a cabo la comunicación, se utilizará la clase de .NET SerialPort, que permitirá realizar una comunicación serial con la tarjeta Arduino, con el fin de leer y controlar lo que se le conecte.

1.6.1 Clase SerialPort

El control SerialPort del entorno de Visual Basic .NET permite establecer una comunicación serial con cualquier dispositivo; en este caso, la computadora. Para ello, se debe arrastrar al formulario. (Ver figuras 1.2 y 1.3.)

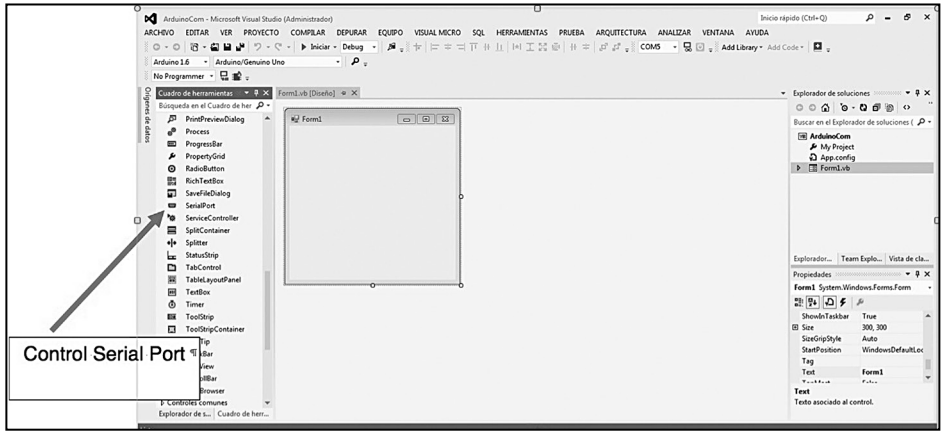


Figura 1.2 Tomar el control SerialPort del menú Cuadro de herramientas.

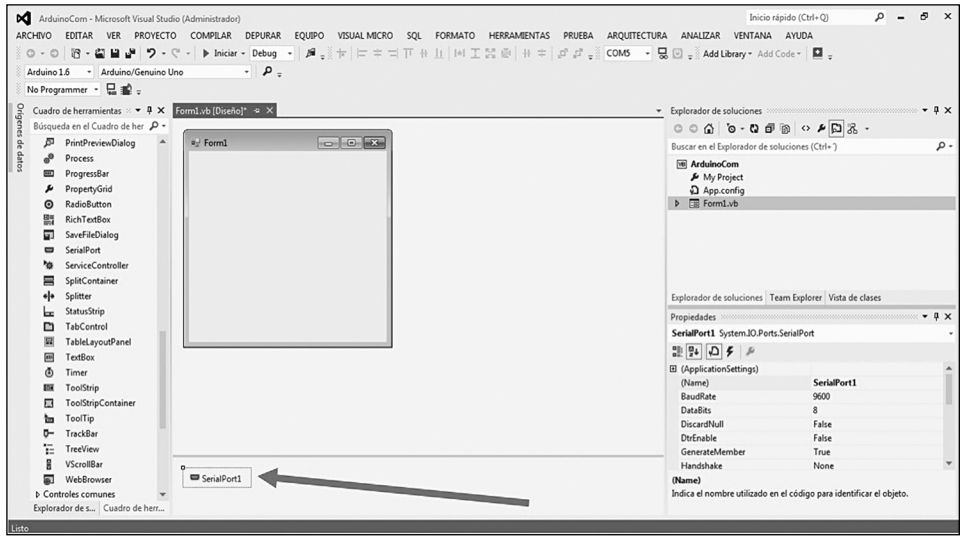


Figura 1.3 Arrastrar el control SerialPort al formulario.

Una vez que se tiene el control serial, se pueden ver sus propiedades, que deben configurarse como se muestra en la figura 1.4.

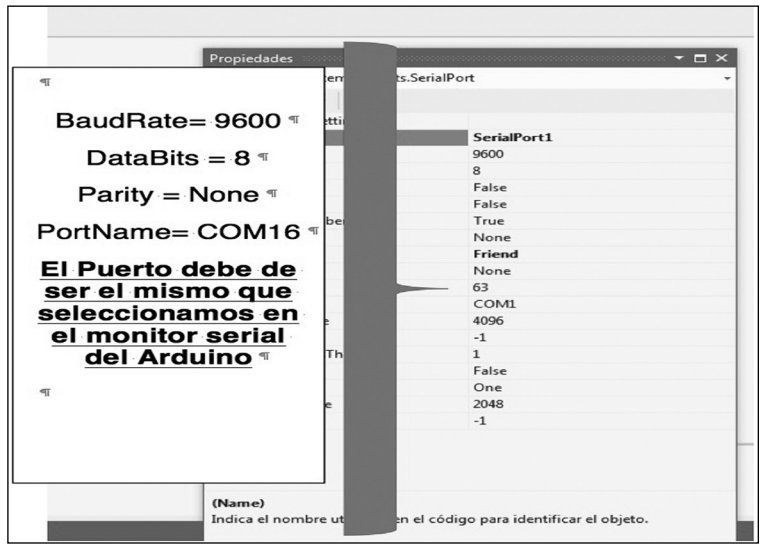


Figura 1.4 Configuración del control SerialPort.