



## PROCESAMIENTO DIGITAL DE SEÑALES

### PRÁCTICAS DE LABORATORIO CON ARDUINO DUE

M.C. GILBERTO SANTILLÁN TOVAR  
DR. DANIEL U. CAMPOS DELGADO

FACULTAD DE CIENCIAS  
UASLP

Enero/2016

## Tabla de contenido

I. INTRODUCCIÓN .....	3
II. CARACTERÍSTICAS GENERALES ARDUINO DUE .....	4
II.A ESPECIFICACIONES:.....	5
II.B CONFIGURACIÓN DE ARDUINO DUE EN WINDOWS.....	5
II.C PROGRAMANDO ARDUINO DUE.....	10
II.D MATERIAL DE APOYO .....	13
III. ESTRUCTURA BÁSICA DE UN PROGRAMA.....	15
III.A FUNCIONES .....	20
III.B INTERRUPCIONES .....	23
III.C ESTRUCTURAS DE CONTROL.....	24
III.D PUERTO SERIE .....	27
III.E LIBRERIAS EN ARDUINO .....	28
IV. PRÁCTICAS EN ARDUINO DUE .....	31
PRÁCTICA # 1: SALIDAS DIGITALES .....	31
PRÁCTICA # 2: EL TEMPORIZADOR.....	33
PRÁCTICA #3: CONFIGURACIÓN DE ENTRADAS DIGITALES.....	35
PRÁCTICA #4: IMPLEMENTACIÓN DE SEMÁFORO .....	37
PRÁCTICA #5: ESTRUCTURA DE CONTROL “FOR” .....	40
PRÁCTICA #6: ENTRADA ANALÓGICA Y PUERTO SERIAL .....	42
PRÁCTICA #7: MODULACIÓN DE ANCHO DE PULSO (PWM) .....	44
PRÁCTICA #8: DISPLAY DE CRISTAL LÍQUIDO.....	46
PRÁCTICA #9: MEDICIÓN DE TEMPERATURA .....	48
PRÁCTICA #10: CONTADOR DE PULSOS.....	50
PRÁCTICA #11: MANEJO DE INTERRUPCIONES.....	52
PRÁCTICA #12: GENERADOR DE SEÑAL DIENTE DE SIERRA.....	54
PRÁCTICA #13: LAZO ADC – DAC (TRASLAPE).....	56
PRÁCTICA #14: GENERADOR DE SEÑAL SENOIDAL .....	59
PRÁCTICA #15: TEMPORIZADOR/CONTADOR PARA EL MUESTREO PERIÓDICO DE UNA SEÑAL ANALÓGICA .....	63
PRÁCTICA #16: FILTRO FIR.....	66
PRÁCTICA #17: FILTRO IIR .....	69
V. REFERENCIAS .....	72
VI. BIBLIOGRAFÍA .....	73

## I. INTRODUCCIÓN

La plataforma Arduino está compuesta por hardware y software, la cual está basada en un microcontrolador con entradas y salidas, tanto analógicas como digitales. Esta plataforma tiene la característica de ser un sistema abierto, lo que significa que su diseño como su distribución son libres, es decir se puede utilizar sin haber adquirido licencia alguna; así también están disponibles los archivos de diseño (CAD) permitiendo al usuario adaptarlo a necesidades específicas. Otra ventaja de Arduino es que es compatible con Windows, Mac OS y Linux, que a diferencia del entorno de otros microcontroladores están limitados a Windows. Debido a que existen diversas librerías, Arduino puede personalizarse con nuevas funcionalidades, por lo que esta plataforma facilita el desarrollo de aplicaciones en distintas áreas de la electrónica, tales como: procesamiento digital de señales, electrónica de potencia, automatización y control, entre otras. Actualmente Arduino, ha comenzado a tomar relevancia a nivel mundial, no solo por ser una plataforma abierta, si no porque también está orientado a usuarios no programadores, ya que utiliza el lenguaje “*Processing*” (el cual ha sido creado para la enseñanza de la programación en un contexto visual) en conjunto con “*Wiring*” (plataforma de Hardware multipropósito con un ambiente para no programadores).

En este manual se describen las propiedades de la plataforma Arduino DUE, donde en secciones posteriores se enlistan sus características principales, instalación del software y hardware. En este contexto, el manual busca ser solo un documento de apoyo para introducir a los estudiantes de la materia de Procesamiento Digital de Señales a la plataforma Arduino, y en general solo se asume que los usuarios tienen conocimientos básicos de programación y de electrónica digital. La primera parte del documento les dará una visión general del Arduino DUE y enseguida se describen 17 prácticas básicas, que incluyen una muestra del código en software que puede ser implementado, y quedando a su criterio y del instructor el modificarlo para lograr el objetivo de la práctica. En la parte final de las prácticas se abordan conceptos como conversión A/D y D/A, traslape, y filtros FIR e IIR que son fundamentales en un curso de procesamiento digital. Con estas bases los alumnos ya podrán desarrollar un proyecto final del curso que involucre una aplicación en su área de especialización (Ing. Electrónica, Biomédica o Telecomunicaciones).

## II. CARACTERÍSTICAS GENERALES ARDUINO DUE

Arduino DUE es la primer placa electrónica basada en un microcontrolador de 32 bits, con la cual mejora las capacidades de las versiones antecesoras tales como el modelo UNO y el modelo Leonardo, las cuales trabajan a 8 bits (<http://www.arduino.cc/>). Gracias al microcontrolador de la compañía **Atmel** (ATSM3X8E), se pueden disponer de entradas y salidas analógicas con resolución de 12 bits; la tasa de muestreo con la que trabaja Arduino DUE es de hasta 1000 ksps (kilomuestras por segundo). En comparación con Arduino UNO que trabaja a una tasa de muestreo de 15 ksps.

Arduino DUE enumera los pines o terminales del 0 – 53, los cuales pueden ser utilizados como entradas o salidas digitales. Todas las entradas/salidas trabajan a 3.3V. Cada pin puede suministrar (soportar) una corriente de 3mA – 15 mA dependiendo del PIN, o recibir de 6 mA – 9 mA, dependiendo del PIN. Estos pines también poseen una resistencia de *pull-down* desactivada por defecto de 100 KΩ. Además, algunos de estos pines tienen funciones específicas.

- Pines 2 al 13 → Salidas PWM de 8 bits de resolución.
- Entradas analógicas: pines de A0 a A11 → Arduino DUE integra 12 entradas analógicas, cada una de las cuales tiene una resolución de 12 bits (4096 valores diferentes). Por defecto, la resolución de la lectura está establecida a 10 bits para que sea compatible con las aplicaciones diseñadas para otras placas Arduino. Es posible cambiar esta resolución ADC mediante la función ***analogReadResolution*** (que en secciones posteriores se define). Las entradas analógicas de Arduino DUE, miden desde el voltaje de referencia de 0 Volts hasta un valor máximo de 3.3 Volts.
- DAC0 y DAC1 → Estos pines proporcionan una salida analógica con una resolución de hasta 12 bits (4096 niveles), aunque con la función ***analogWriteResolution*** (bits), podemos modificar su resolución.
- RX0 – TX0 → 4 Canales de comunicación serial.
- Interfaz I<sup>2</sup>C ( SDA, SCL)
- AREF → Referencia externa para voltaje de entradas analógicas.

## II.A ESPECIFICACIONES:

- Microcontrolador ATSM3X8E.
- Velocidad del reloj 84 MHz.
- Voltaje de operación **3.3 Volts**.
- Voltaje de entrada (jack DC) 7 – 12 Volts.
- Nucleo de 32 bits.
- 54 Entradas/Salidas digitales.
- 12 Entradas analógicas.
- 12 Salidas tipo PWM.
- Corriente maxima en E/S 130mA.
- 2 Salidas analógicas (DAC).
- 4 Puertos UART.
- 512 KB de Memoria flash para código.
- 2 Puertos micro-USB.



Arduino DUE tiene integrados dos botones “*erase*” y “*reset*”. Con el botón “*erase*” es posible borrar la memoria Flash del microcontrolador, basta con mantener presionado este botón durante algunos segundos para eliminar el programa cargado en el microcontrolador y el botón “*reset*” permite reiniciar un programa que se ejecuta en la plataforma de Arduino. También posee dos puertos micro-USB (“*Programming*” y “*Native*”), de los cuales el puerto “*Programming*” generalmente se utiliza para programación y comunicación, y el puerto “*Native*” actuará como Host USB, permitiendo conectar periféricos externos USB, tales como mouse, teclado o “*Shields*”, que agregan nuevas funciones a Arduino DUE.

## II.B CONFIGURACIÓN DE ARDUINO DUE EN WINDOWS

A continuación se muestra la instalación paso a paso del manejador o *driver* y software de Arduino DUE, bajo el entorno de Windows en su versión 7 sp1. Para conectar Arduino DUE a la computadora se necesita de un cable micro-USB, el cual alimentará a la placa y permitirá su programación. La programación de Arduino DUE puede realizarse a través del

puerto “*Programming*” o del puerto “*Native*” (ver Figura 1.1), por recomendación del fabricante la programación se realizará por medio del puerto “*Programming*”.

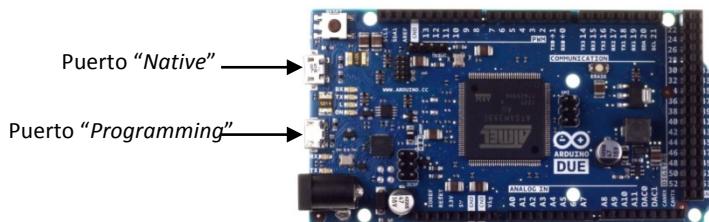
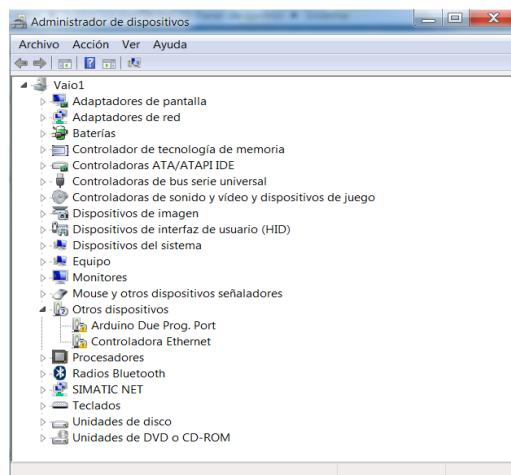


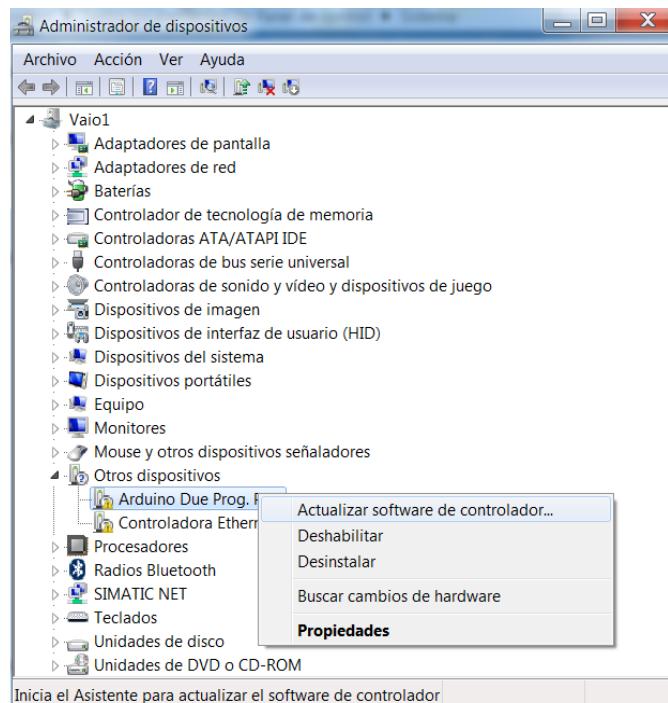
Figura 1.1. Puertos Micro-USB Arduino DUE.

El primer paso para establecer el enlace con Arduino DUE a través de la computadora, es descargar el software desde la página oficial de Arduino (proporcionada en la sección de Material de Apoyo), elegir la descarga de Arduino en la versión 1.5.2 para Windows (o el sistema operativo que se desee utilizar) y ubicar la ruta donde se guardarán los archivos de instalación. Una vez finalizada la descarga de los archivos, se realiza la conexión entre Arduino DUE y la computadora a través del cable micro – USB, y enseguida aparecerá el asistente de Windows para agregar nuevo Hardware detectado, éste asistente no se utilizará, por lo que se debe cancelar esta opción.

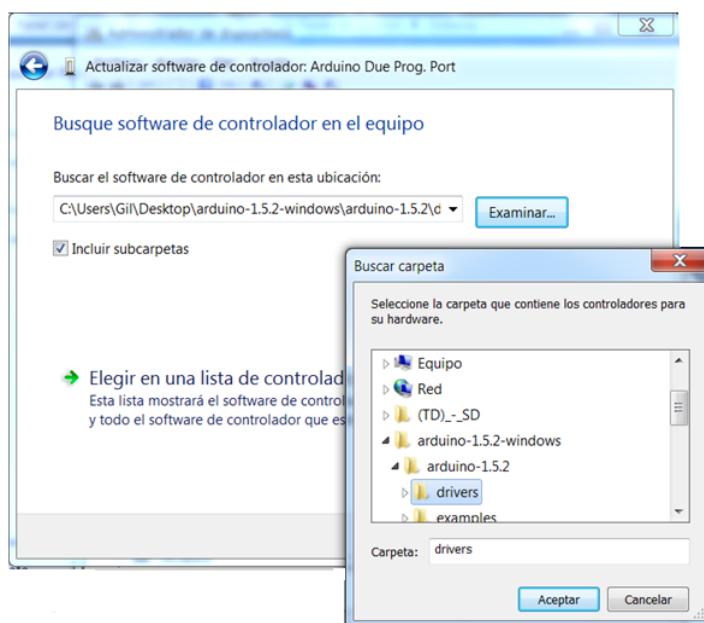
La instalación del controlador de Arduino DUE se realizará de manera manual, esto con la finalidad de identificar el puerto asignado por la computadora a la plataforma de Arduino, ya que posteriormente en el software de programación se especificará el mismo puerto para poder tener el enlace: Computadora – Arduino, y poder escribir el programa en la plataforma. Para dar de alta el *driver* de Arduino de manera manual, se accesa a: “*Administrador de Dispositivos*”.



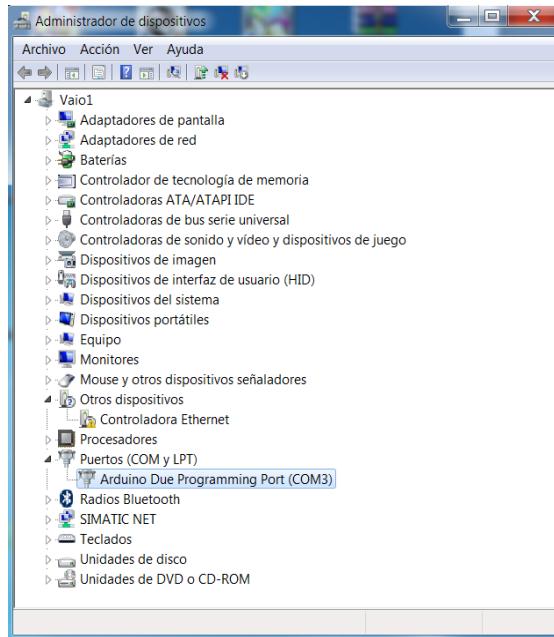
Enseguida se debe seleccionar “*Arduino Due Prog. Port*” y con “click” derecho del “mouse” seleccionar la opción “*Actualizar software de controlador*”.



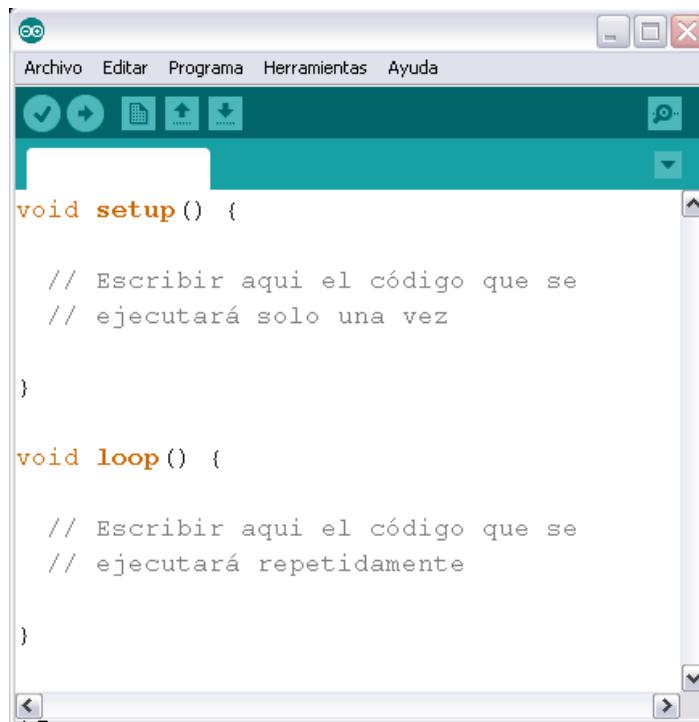
Elegir la opción de “*Instalar desde una ubicación específica*”, seleccionar la ubicación donde se guardaron los archivos de instalación de Arduino 1.5.2, seleccionar la carpeta con el nombre de “*drivers*” y continuar con la instalación del controlador.



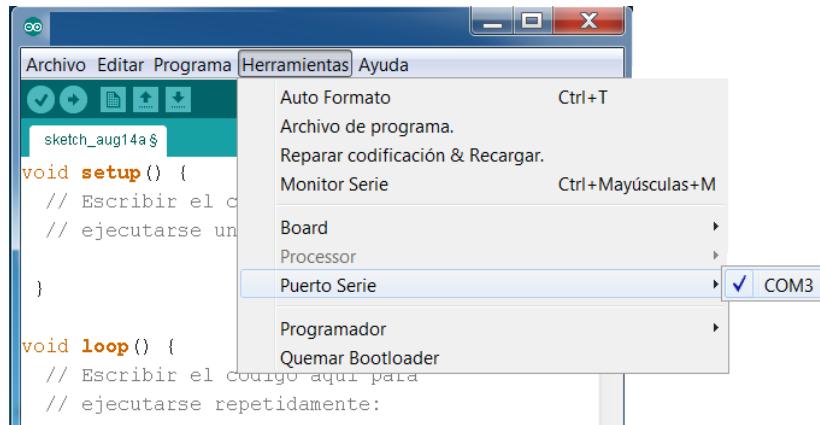
Una vez concluida la instalación del controlador, se muestra la información actualizada del Hardware, así como el puerto al que está conectado.



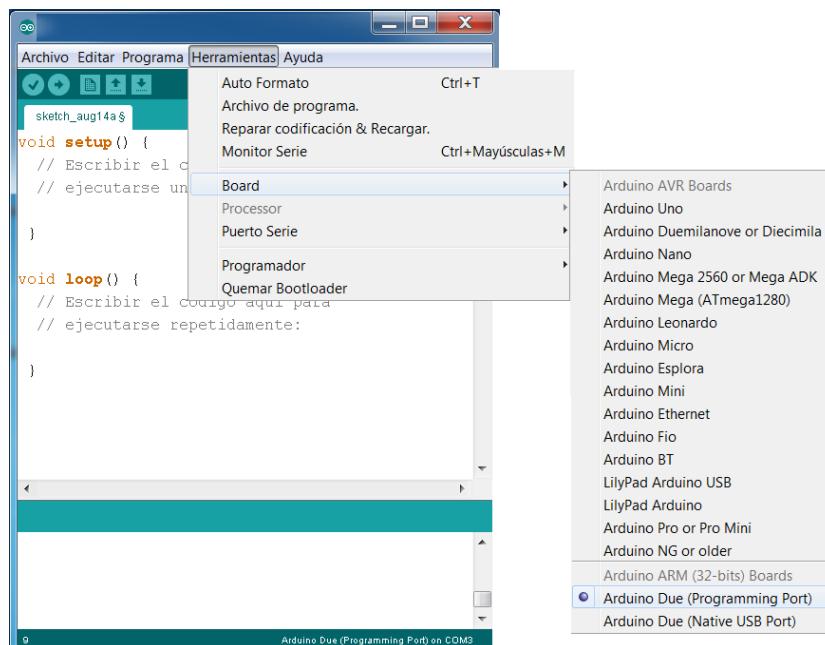
Para la configuración del software de Arduino, se ejecuta el ícono de aplicación “*Arduino.exe*”, el cual está incluido en la carpeta de archivos descargados. Al ejecutarse la aplicación de Arduino se muestra la siguiente ventana:



En el menú de Herramientas → Puerto Serie, especificar el puerto al cual está conectado Arduino.



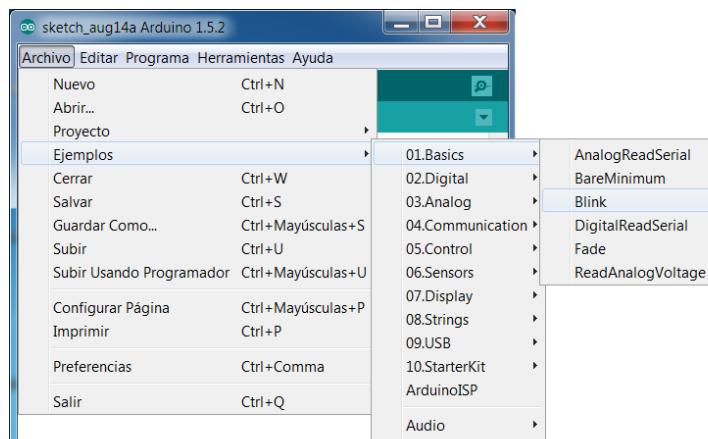
Especificar el modelo de la plataforma Arduino en el menú de Herramientas → Board, el cual corresponde a Arduino DUE y el puerto micro-USB mediante el cual se programará (“*Arduino DUE Programming Port*”).



Con los pasos anteriores se ha configurado el Hardware y Software de Arduino DUE. La configuración es muy similar para algunas otras versiones de Windows, así como en sistemas operativos Mac OS y Linux.

## II.C PROGRAMANDO ARDUINO DUE

Para ilustrar la programación de Arduino DUE, se hará uso de los ejemplos que están precargados en el software. Primero se selecciona el ejemplo “*Blink*”, el cual consiste en el encendido/apagado de un led durante un tiempo de 1 segundo.



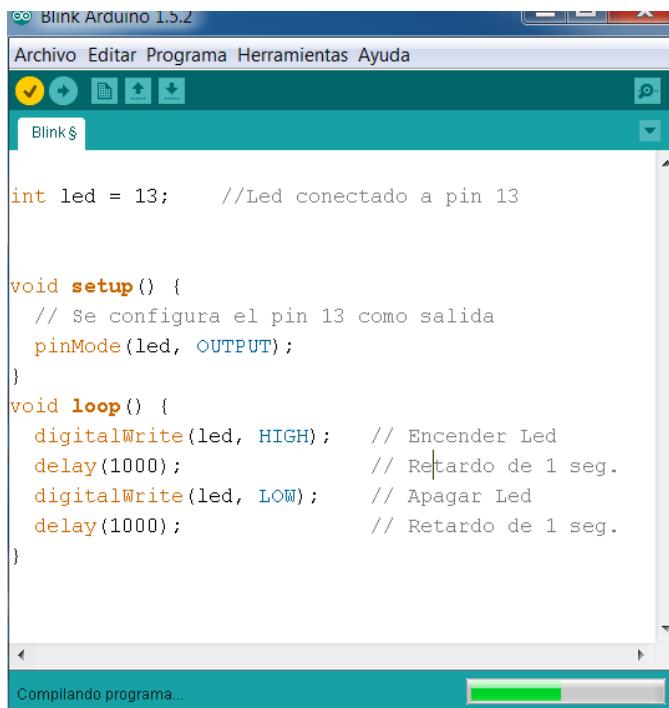
Una vez seleccionado el ejemplo, la ventana del software de Arduino tendrá un esquema como el siguiente:

```
int led = 13;      //Led conectado a pin 13

void setup() {
    // Se configura el pin 13 como salida
    pinMode(led, OUTPUT);
}

void loop() {
    digitalWrite(led, HIGH);    // Encender Led
    delay(1000);              // Retardo de 1 seg.
    digitalWrite(led, LOW);    // Apagar Led
    delay(1000);              // Retardo de 1 seg.
}
```

Finalmente se compila el programa en busca de errores en la programación, haciendo “click” en el ícono: .



The screenshot shows the Arduino IDE interface with the title bar "Blink Arduino 1.5.2". The menu bar includes "Archivo", "Editar", "Programa", "Herramientas", and "Ayuda". Below the menu is a toolbar with icons for file operations. The main window displays the "Blink \$" tab containing the following code:

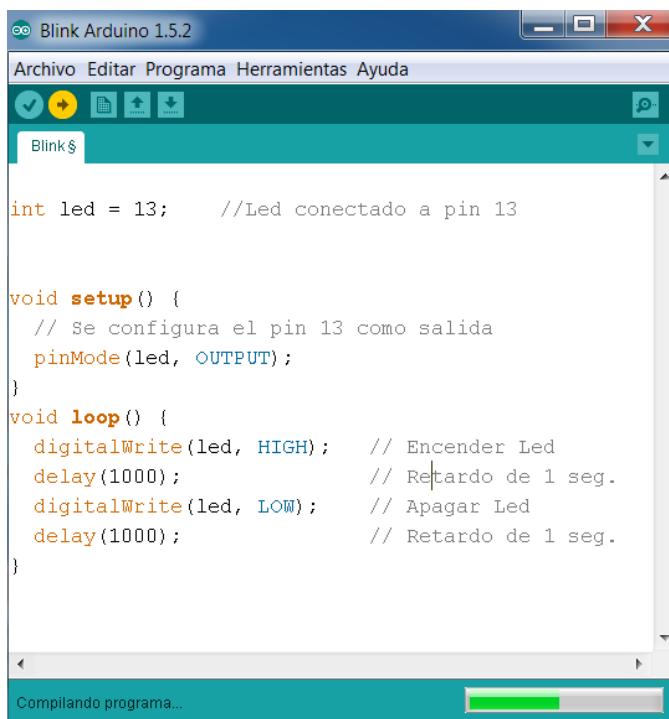
```
int led = 13;      //Led conectado a pin 13

void setup() {
    // Se configura el pin 13 como salida
    pinMode(led, OUTPUT);
}

void loop() {
    digitalWrite(led, HIGH);    // Encender Led
    delay(1000);              // Retardo de 1 seg.
    digitalWrite(led, LOW);    // Apagar Led
    delay(1000);              // Retardo de 1 seg.
}
```

At the bottom of the code editor, a status bar says "Compilando programa..." with a progress bar.

Posteriormente se cargará el programa en la placa de Arduino haciendo “click” en el ícono: .



The screenshot shows the Arduino IDE interface with the title bar "Blink Arduino 1.5.2". The menu bar includes "Archivo", "Editar", "Programa", "Herramientas", and "Ayuda". Below the menu is a toolbar with icons for file operations. The main window displays the "Blink \$" tab containing the same code as the previous screenshot:

```
int led = 13;      //Led conectado a pin 13

void setup() {
    // Se configura el pin 13 como salida
    pinMode(led, OUTPUT);
}

void loop() {
    digitalWrite(led, HIGH);    // Encender Led
    delay(1000);              // Retardo de 1 seg.
    digitalWrite(led, LOW);    // Apagar Led
    delay(1000);              // Retardo de 1 seg.
}
```

At the bottom of the code editor, a status bar says "Compilando programa..." with a progress bar. The upload icon in the toolbar is highlighted in yellow, indicating it is active.

Si se ha cargado el programa correctamente a la plataforma de Arduino DUE, enseguida se mostrará el mensaje “*Subido*”, con lo que concluye la programación de Arduino DUE.



The screenshot shows the Arduino IDE interface. The code editor window contains the following sketch:

```
int led = 13;      //Led conectado a pin 13

void setup() {
    // Se configura el pin 13 como salida
    pinMode(led, OUTPUT);
}

void loop() {
    digitalWrite(led, HIGH);    // Encender Led
    delay(1000);              // Retardo de 1 seg.
    digitalWrite(led, LOW);    // Apagar Led
    delay(1000);              // Retardo de 1 seg.
}
```

The Serial Monitor window at the bottom displays the message "Subido" (highlighted with a green circle), followed by the output of the upload process: "Verify successful", "Set boot flash true", and "CPU reset.".

En las siguientes imágenes, se muestra cuando el led está activado (ver Figura 1.2) y posteriormente cambia de estado, como se observa en la Figura 1.3.



Figura 1.2. Led activado, programa “*Blink*”.



Figura 1.3. Led desactivado después de 1 segundo.

La fuente de alimentación (proporcionada por la computadora) se provee a través del puerto micro-USB “*Programming*”, aunque también se puede realizar mediante el puerto “*Native*”.

## II.D MATERIAL DE APOYO

A continuación se listan las direcciones electrónicas de donde se pueden descargar de manera gratuita los diversos programas y manuales necesarios para la programación de Arduino DUE, así como referencias del lenguaje de programación y características del microcontrolador utilizado en el kit Arduino DUE.

- Página Oficial de Arduino. [Fecha de consulta: Agosto 2014]  
<http://www.arduino.cc/>
- Descarga de Driver y Software. [Fecha de consulta: Agosto 2014]  
[www.arduino.cc/downloads](http://www.arduino.cc/downloads)
- Características principales Arduino DUE. [Fecha de consulta: Agosto 2014]  
<http://arduino.cc/en/Main/arduinoBoardDue>
- Diagrama esquemático Arduino DUE (pcb). [Fecha de consulta: Agosto 2014]  
<http://arduino.cc/en/uploads/Main/arduino-Due-schematic.pdf>
- Página Oficial de Programación “*Processing*”. [Fecha de consulta: Agosto 2014]  
<http://processing.org/>

- Página Oficial plataforma “*Wiring*”. [Fecha de consulta: Agosto 2014]  
<http://www.wiring.org.co/>
- Hoja de datos de microcontrolador AT91SAM3X8E. [Fecha de consulta: Agosto 2014]  
<http://www.atmel.com/Images/doc11057.pdf>

### III. ESTRUCTURA BÁSICA DE UN PROGRAMA

Los programas se implementan haciendo uso del entorno de programación de Arduino, el cual está basado en el lenguaje de programación C/C++. Un programa en Arduino, se le conoce como “*sketch*” y se divide en tres partes fundamentales: *estructura*, *valores* (variables y constantes), y *funciones*. La estructura básica de programación de Arduino es bastante simple, divide la ejecución en dos partes: “*setup*” y “*loop*”. La sección “*setup()*” constituye la preparación o inicialización del programa y “*loop()*” denota la ejecución.

De esta manera, la parte de programa que se encuentra dentro de la función “*setup()*”, es lo primero que se ejecuta y solamente se realiza una vez. En esta sección se pueden definir constantes y variables, también se pueden configurar los pines como entradas/salidas o especificar el tipo según sea analógico/digital. Las instrucciones dentro de la función “*loop()*” se ejecutarán continuamente ó hasta que ocurra un “*reset*”.

Ejemplo de “*sketch*”

```
void setup() {                                // Se ejecuta solo una vez.  
    int pin = 13;                            // Declara variable de tipo entero  
    pinMode(pin, OUTPUT);                    // Establece 'pin' como salida.  
}  
                                         // Final de la función 'setup()'  
void loop() {                                // Se ejecuta continuamente  
    digitalWrite(pin, HIGH);                  // Activa 'pin'  
    delay(1000);                            // Pausa un segundo  
    digitalWrite(pin, LOW);                  // Desactiva 'pin'  
    delay(1000);  
}
```

#### Sintaxis

Como se observa en el ejemplo anterior, al concluir cada declaración, se finaliza con un punto y coma “;”. Para realizar algún comentario dentro del programa y en una línea se antepondrá “//” al comentario, si se desea realizar un bloque de comentarios es decir en varias líneas, se utiliza la siguiente sintaxis “/\*... \*/”. Mientras tanto los símbolos “{” y “}”, indican inicio y fin respectivamente de una función.

## Variables

Una variable es una forma de asignar o almacenar un valor para un uso posterior por el programa, estos datos pueden venir de un sensor o alguna constante, los cuales permiten realizar un cálculo o realizar una acción determinada.

### Declaración de Variables

Declarar una variable, significa asignar un nombre, definir el tipo de dato y opcionalmente asignar un valor. Las variables no necesariamente tienen que ser inicializadas con un valor cuando son declaradas, pero frecuentemente es útil para asegurar el valor asignado.

#### Ejemplo:

```
int variableEntrada1;           // Se define la variable de tipo entero.  
int variableEntrada2 = 0;        // Se define la variable de tipo entero y con un valor de cero.
```

#### *Tipos de variables*

Para asignar el tipo de variable, se debe conocer el tipo de valor que se va a representar dentro del entorno de programación, para así especificar su tipo correcto. Los tipos de variable y características utilizadas en Arduino DUE se listan a continuación.

**char:** Variable del tipo carácter. El carácter va entre comillas y utiliza la equivalencia con el código ASCII, por lo que al definir una variable como 65 del tipo char, será equivalente a asignar la letra A.

#### Ejemplo:

```
char v = 65;                  // Asigna a v la letra A.  
char v = 'A';
```

**byte:** Tipo de variable que almacena números de 0 – 255 en 8 bits. El valor se expresa en binario y el bit menos significativo es el que encuentra más a la derecha.

#### Ejemplo:

```
byte b = B1001;                // asigna a b el numero 9.
```

**int:** Variable de tipo entero. Éste es el tipo principal de variable para almacenar números enteros, que almacena valores de 2 bytes. Por lo que produce un rango entre -32,768 y 32,767.

Ejemplo:

```
int ledpin = 13;           // 13 es asignado a la variable ledpin
```

**unsigned int:** Variable de tipo entero sin signo. Este tipo de variable solo considera los valores positivos asignados a una variable, los cuales se almacenan en 2 bytes teniendo un rango útil de 0 a 65,535 valores.

**long:** Variable entera extendida. Las variables de tipo “long” son de tamaño extendido para almacenamiento de números enteros (4 bytes), donde su rango es desde -2,147,483,648 a 2,147,483,647.

**unsigned long:** Variable entera extendida sin signo. Variable extendida que solamente considera valores enteros positivos, teniendo un rango desde 0 a 4,294,967,295.

**float:** Variable flotante. Tipo de variable para los números en punto flotante (números decimales), ocupa 4 bytes, por lo que las variables tipo “*float*” tienen el valor máximo 3.4028235E+38, y como mínimo pueden alcanzar el -3.4028235E+38 con una precisión de 6 dígitos decimales.

**double:** Variable flotante extendida. Tipo de variable en punto flotante con doble precisión. La implementación “*double*” en Arduino es exactamente la misma que “*float*”.

Cuando una variable sobrepasa su valor máximo permitido, ocurre el efecto llamado “*roll over*”, el cual consiste en pasar al valor del otro extremo del rango permitido para la variable.

Ejemplo

```
int x;  
x = 32767;           // El valor de x será el otro extremo permitido  
x = x+1;            // es decir -32,768.
```

**array:** Es una colección de variables que son accedidas mediante un valor índice, las cuales son utilizadas principalmente en arreglos de elementos representados por un vector o una matriz.

**boolean:** Una variable de asignada de tipo booleano, solamente puede tomar dos valores y se especifican como “*TRUE*” o “*FALSE*”.

**#define:** Es un componente utilizado frecuentemente por el programador, lo que permite dar un nombre a un valor constante antes de que se compile el programa.

Ejemplo:

```
#define nombredeconstante valor          // reemplaza en cualquier parte del programa  
                                         // nombredeconstante por "valor".
```

### Variable local vs Variable Global

El sitio en el que la variable es declarada determina el ámbito o alcance de la misma. Una variable global es aquella que puede ser empleada en cualquier función del programa. Estas variables deben ser declaradas al inicio del programa (antes de la función “*setup()*”). Mientras tanto, una variable local es solamente visible en la función donde es declarada, por ejemplo dentro de un ciclo *for*.

Ejemplo:

```
int v;                                // 'v' es visible en todo el programa (variable global)  
void setup() {  
}  
void loop() {  
    float f;                            // 'f' es visible únicamente en la función loop() (variable local)  
    for (int i=0; i<20; i++){           // 'i' es visible solo en el ciclo for (variable local)  
        delay(500);  
    }  
}
```

### Operadores

El uso de operadores es similar al utilizado en el lenguaje de programación C y C++, se clasifican en aritméticos, booleanos, compuestos y de comparación.

#### *Operadores aritméticos*

Se utilizan empleando variables, valores constantes o elementos de un arreglo.

OPERADOR	ACCIÓN REALIZADA
=	Operador de asignación.
+	Operador de suma.

-	Operador de resta.
*	Operador de multiplicación.
/	Operador de división.

### *Operadores Compuestos*

OPERADOR	ACCION REALIZADA
x++	Incrementa a x en 1 y devuelve el valor antiguo de x.
++x	Incrementa a x en 1 y devuelve el nuevo valor de x.
x--	Decrementa en 1 a x y devuelve el valor antiguo de x.
--x	Decrementa a x en 1 y devuelve el nuevo valor de x.

### *Operadores de Comparación*

OPERADOR	ACCION REALIZADA
==	Igual.
!=	Diferente.
<	Menor que.
>	Mayor que
<=	Menor o igual que.
>=	Mayor o igual que.

### *Operadores Booleanos*

OPERADOR	ACCION REALIZADA
&&	AND Lógico
	OR Lógico
!	NOT Lógico
&	Operador de bits AND
	Operador de bits OR
~	Operador de bits NOT

## Constantes

En el lenguaje de Arduino, existen constantes predefinidas, las cuales se utilizan para asignar un valor a una variable o configurar un pin o terminal como entrada/salida. El uso de estas constantes, nos permiten leer un programa con mayor facilidad e identificar la acción realizada en la instrucción. Existen dos constantes para representar si algo es cierto o falso en Arduino (TRUE y FALSE). “*FALSE*” equivale a definir como 0 y “*TRUE*” se define en la mayoría de las veces como 1, aunque en una definición más amplia cualquier entero que no es cero, es “*TRUE*”.

Existen otras constantes que definen el nivel lógico de los pines, nivel alto (*HIGH*) y nivel bajo (*LOW*), cuando se lee o escribe en un pin digital solamente se pueden obtener o asignar dos valores: “*HIGH*” y “*LOW*”. También existen constantes que definen la función de los pines digitales, se utiliza “*INPUT*” para asignar un pin como entrada y “*OUTPUT*” para asignarlo como salida utilizando la función “*pinmode*”.

## III.A FUNCIONES

Una función en Arduino realiza una tarea en específico y puede retornar un valor; cuando una función es invocada, pasa el control o secuencia de ejecución a esa función y una vez que concluyó su tarea el control vuelve a la línea donde fue llamada. También se les asignan parámetros, los cuales pueden ser modificados por la propia función. Enseguida se describen algunas funciones básicas de Arduino, así como su sintaxis y ejemplos de uso.

### *Función: pinMode(pin, mode)*

Función que configura un pin o terminal para comportarse como entrada (*mode=INPUT*) o salida (*mode=OUTPUT*) digital. Esta función por lo general se declara dentro de la función “*setup()*” y puede tener asignado o no una variable.

#### Ejemplo:

```
void setup() {  
    int led =13; // A variable led se le asigna el pin 13.  
    pinMode (led, OUTPUT); // Configura el pin 13 como salida digital.  
}
```

Otra manera de configurar un pin de Arduino DUE como Salida/Entrada digital, es escribiendo directamente el numero de pin a ser utilizado, con ésta función Arduino reconoce que se está configurando un pin y no como un valor asignado a la función.

Ejemplo:

```
void setup() {  
pinMode (13, INPUT); } // Configura el pin 13 como entrada digital.
```

Los pines de Arduino configurados como “**INPUT**” con la función *pinMode()* se dice que se encuentran en un estado de alta impedancia, lo cual lo hace útil para leer un sensor. Los pines configurados como “**OUTPUT**” con la función *pinMode()* se dice que están en estado de baja impedancia. Esto implica que pueden proporcionar una cantidad sustancial de corriente a otros circuitos.

**Función:** *digitalRead(pin)*

Esta función lee el valor de un pin digital y devuelve un valor “*HIGH*” o “*LOW*”, donde el pin puede ser especificado con una variable o una constante.

Ejemplo:

```
void setup() {  
v = digitalRead(11); // La variable v será igual al valor leído en el pin 11.  
}
```

Si el pin especificado en la función no tiene conexión física, la función *digitalRead()* puede leer un “1” lógico ó “0” lógico, y podría cambiar aleatoriamente.

**Función:** *digitalWrite(pin, value)*

Introduce un “1” lógico (*HIGH*) o “0” lógico (*LOW*) en el pin digital especificado. De nuevo, el pin puede ser asignado con una variable o una constante.

Ejemplo:

```
void setup(){  
int pin = 8;  
digitalWrite(pin, HIGH); } // Asigna a pin 8 “1” lógico.
```

**Función:** *analogRead(pin)*

Lee el valor analógico de un pin con una resolución de 10 bits predeterminada. Esta función solo funciona en los pines analógicos (A0 – A11). El valor resultante es un entero de 0 a 1023. Los pines analógicos, a diferencia de los digitales **NO** necesitan declararse previamente como entrada o salida.

**Ejemplo:**

```
Valor = analogRead(A1); // Lee el valor de la entrada A1 y lo asigna a la variable Valor.
```

**Función:** *analogWrite(pin, valor)*

Escribe un valor de 0 – 255 (resolución de 8 bits) en el pin especificado. Se pueden utilizar los pines del 2 – 13 como salidas. Al utilizar esta función no es necesario utilizar la función pinMode para establecer al pin como salida.

**Ejemplo:**

```
Int pin1 = 10; // Asignación de variable a pin 10.  
Valor = analogWrite(pin1, 127); // En variable “pin1” escribe un valor de 1.65V.
```

El valor máximo es 255, el cual significa que se tendrá la salida de voltaje más alta disponible (3.3V), aunque si se asigna un valor superior a éste, el pin seguirá teniendo su salida máxima.

**Función:** *analogReference(type)*

Configura el valor del voltaje de referencia utilizado para la entrada analógica, es decir el valor que se utiliza como la parte superior del rango de entrada. Para Arduino DUE, la referencia analógica por default es de 3.3 Volts, aunque se puede elegir de la siguiente manera el tipo:

INTERNAL1V1: La referencia será de 1.1 Volts.

INTERNAL2V56: La referencia se establece en 2.56 Volts.

EXTERNAL: La referencia dependerá del voltaje aplicado al pin AREF (0 – 3.3 Volts).

#### **Función:** *analogReadResolution(bits de resolución)*

Establece la resolución con la que se lee una entrada analógica. Su valor predeterminado es de 10 bits (valores entre 0 – 1023). Arduino DUE soporta una resolución máxima de 12 bits, por lo que si se establece un valor de 16, la función *analogReadResolution* dará un número de 16 bits, donde los primeros 12 bits contienen la lectura verdadera y los últimos se rellenan con ceros.

#### **Función:** *analogWriteResolution(bits de resolución)*

Establece la resolución de escritura, su valor predeterminado es de 8 bits (valores entre 0 – 255), aunque esta resolución se puede modificar hasta 12 bits (0 – 4095 valores). Arduino DUE posee 12 pines que por defecto tienen una resolución de 8 bits (PWM), aunque pueden ser cambiadas a 12 bits de resolución. También contiene dos pines DAC (convertidor digital a analógico), con resolución predeterminada de 12 bits.

#### **Función:** *delay(valor en ms).*

Realiza una pausa en el programa según el tiempo especificado en unidades de ms. La función “delay” tiene las desventajas de que mientras se está ejecutando no se pueden leer entradas, realizar cálculos matemáticos o modificar los pines. Sin embargo continúan funcionando las interrupciones, los valores PWM (*analogWrite*) y los estados lógicos de los pines se mantienen.

#### **Función:** *millis()*.

Devuelve la cantidad de milisegundos que lleva la tarjeta Arduino DUE ejecutando el programa actual, puede contar hasta un tiempo equivalente a 50 días, una vez completado, comienza nuevamente.

## **III.B INTERRUPCIONES**

El manejo de interrupciones en Arduino DUE permite ejecutar eventos de manera asíncrona, es decir que el código principal reaccione a eventos externos sin necesidad de estarlos monitoreando continuamente.

### ***Funcion: attachInterrupt(pin, función, modo)***

Esta función nos permite el manejo de interrupciones en un programa, si utilizamos esta función, definiremos los siguientes parámetros:

*pin*: especifica el numero de entrada de interrupción, para Arduino DUE se puede utilizar cualquier pin (0 – 53).

*función*: Se define el nombre de la función a ser llamada cuando ocurre la interrupción.

*modo*: Define la transición del pin para activar la interrupción. Los modos en que se activa la interrupción son los siguientes:

- ✓ LOW: Activa cuando el pin está en nivel bajo.
- ✓ HIGH: Activa cuando el pin está en nivel alto
- ✓ CHANGE: Activa cuando hay un cambio de estado en el pin.
- ✓ RISING: Se activa cuando hay un cambio de nivel bajo a alto.
- ✓ FALLING: Activa la interrupción cuando detecta un cambio de nivel alto a bajo.

Existen restricciones al utilizar las interrupciones, por ejemplo si necesitamos generar un retardo dentro de la función llamada en la interrupción, no se podrá utilizar la función *delay()* y el valor devuelto por la función *millis()* no se incrementará. Además si se tiene la comunicación serial activa, los datos serie recibidos en el transcurso de esta interrupción pueden perderse.

## **III.C ESTRUCTURAS DE CONTROL**

Enseguida se describen 7 estructuras de control en la programación de Arduino, así como su sintaxis y algunos ejemplos básicos.

### ***Condicional: if***

Se utiliza regularmente con operadores de comparación o booleanos para realizar cierta acción si las condiciones se cumplen.

#### **Ejemplo:**

```
If (x > 50){                                // ¿ x mayor que 50?  
    digitalWrite(pin1, HIGH);                  // Realiza las acciones.  
}
```

Para el condicional “if”, si se cumplen las condiciones booleanas entre paréntesis, enseguida se ejecutan las acciones que se encuentran entre las llaves {...}, de lo contrario salta el programa a ejecutar la siguiente instrucción.

### ***Condicional: if... else***

A diferencia del condicional “if”, éste permite mayor control sobre una estructura pues permite ejecutar una acción si la condición se cumple o no. En el condicional “if – else”, se puede tener un mayor control sobre el flujo del código del programa, por lo que permite agrupar diferentes condiciones.

#### **Ejemplo:**

```
if ( x == 500 ){                                // x igual a 500
    digitalWrite(13,HIGH);                      // Activa salida digital 13
}
if ( x < 500 ){                                // x menor que 500
    digitalWrite(12,HIGH);                      // Activa salida digital 12
}
else                                              // De lo contrario
{
    XXXXX                                         // Realiza determinada acción
}
```

Cada condición se verifica y cuando se encuentra una condición verdadera, su bloque correspondiente del código se ejecuta y el programa se salta a la línea siguiente. Si ninguna de las condiciones es verdadera, el bloque “else” se ejecuta por defecto.

### ***Ciclo: for***

Este ciclo se utiliza para repetir un bloque de instrucciones entre las llaves {...}, un contador de incremento se utiliza para tener el control de cuantas veces se ha ejecutado el ciclo, así como su conclusión.

#### **Sintaxis:**

```
for (inicio; condición; incremento) {
    XXXXX                                         // Realiza las instrucciones XXXX
}
```

La declaración de inicio solo se evalúa una vez. Cada vez que se va a repetir el bloque, se verifica la condición, si es verdadera se ejecutan las instrucciones dentro de la función y se evalúa la expresión de incremento. El ciclo se ejecutará hasta que la condición se convierta en falsa, por lo que el ciclo terminará

Ejemplo:

```
for (int i=0; i <= 255; i++) {           // Verifica i = 0
    analogWrite(PWMpin, i);             // Escribe valor de i en PWMpin
    delay(10);                         // Ejecuta un retraso de 10 ms
}
```

**Ciclo: while**

El lazo se repite de forma continua e indefinidamente, hasta que la expresión del tipo entero dentro de los paréntesis ( ) se convierte en falsa, de lo contrario el ciclo *while* nunca terminará.

Sintaxis:

```
while(expresion) {
    // Código ejecutado mientras “expresión”
    // es verdadera.
}
```

**Ciclo: do... while**

Este comando trabaja de la misma manera que el lazo “*while*”, con la excepción de que la condición se comprueba al final del lazo, por lo que este lazo se ejecuta “siempre” al menos una vez.

Sintaxis:

```
do {acción}
while (operación booleana);
```

**Sentencia: switch / case**

Esta sentencia la podemos especificar como un comparador múltiple, se recomienda cuando se tienen tres o más alternativas, además nos permite controlar el flujo del código de forma mas eficiente. En particular, compara el valor de una variable a los valores especificados en las sentencias “*case*”, cuando se encuentra una sentencia cuyo valor coincide con el de la

variable, se ejecuta el código en esa sentencia, lo que permite establecer en cada caso un bloque de instrucciones activo.

### Sintaxis:

```
switch (var) {  
    case 1:  
        // Realizar las acciones cuando var = 1  
        break; // Instrucción para salir de la sentencia  
    case 2:  
        // Realizar las acciones cuando var = 2  
        break;  
    default:  
        // Si no cumple alguna condición anterior realiza  
        // las siguientes acciones.  
}
```

La sentencia “*switch*” funciona de la siguiente manera, se evalúa “*var*” entre paréntesis que puede ser del tipo: int, char o byte, dependiendo del valor se ejecutará el código correspondiente al caso, si no coincide con ningún caso se ejecutará el “*default*”.

### ***Instrucción: break***

Esta instrucción es usada para salir de los lazos *do*, *for*, o *while*, pasando por alto la condición normal del lazo, también es usado para salir de una estructura de control *switch*.

## **III.D PUERTO SERIE**

Los *puertos* de entrada/salida permiten la comunicación con elementos externos a la tarjeta. Arduino DUE posee una interfaz para realizar la comunicación de datos digitales (puerto serial), con lo que es posible intercambiar datos con la computadora a partir de las siguientes funciones:

### *Serial.begin(rate)*

Esta función permite abrir un puerto serie y especificar la velocidad de transmisión. La velocidad típica para la comunicación con el ordenador es de 9600 baudios, aunque se pueden soportar otras velocidades.

### *Serial.print(data)*

Este comando imprime los datos al puerto serie como texto ASCII, y tiene varias formas. Los números son impresos mediante un juego de caracteres ASCII para cada dígito. Los valores de tipo "float" son impresos 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 tal cual. Por ejemplo:

- `Serial.print(78)` imprime "78"
- `Serial.print(1.23456)` imprime "1.23"
- `Serial.print(byte(78))` imprime "N" (cuyo código ASCII es 78)
- `Serial.print('N')` imprime "N"
- `Serial.print("Hello world.")` imprime "Hello world."

### *Serial.println(data)*

La instrucción imprime datos al puerto serie, seguido por un retorno de línea automático. Este comando tiene la misma forma que "`Serial.print()`" pero este último sin el salto de línea al final. Este comando puede emplearse para realizar la depuración de programas. Para ello puede mandarse mensajes de depuración y valores de variables por el puerto serie. Posteriormente, desde el entorno de programación de Arduino, activando el "*Serial Monitor*" se puede observar el contenido del puerto serie, y por lo tanto, los mensajes de depuración. Para observar correctamente el contenido del puerto serie se debe tener en cuenta que el "*Serial Monitor*" y el puerto serie han de estar configurados a la misma velocidad.

## III.E LIBRERIAS EN ARDUINO

Las librerías en Arduino son archivos escritos en C o C++, que permiten ampliar la funcionalidad de los programas. Para hacer uso de una librería en un "*sketch*" (el IDE de Arduino), basta con hacer "*click*" sobre "*Programa - Import Library*" en el menú, escoger la librería correspondiente y se agregará a nuestro programa. Como las librerías se cargan en el Arduino junto con el programa, se consume más espacio de memoria, si un programa ya no necesita una biblioteca, sólo se tiene que borrar el "**#include**" de las declaraciones de la parte superior del código. Esto hará que el IDE de Arduino no enlace la librería con el

programa y disminuya la cantidad de espacio utilizado en la tarjeta del Arduino. Las librerías estándar que ofrece Arduino son las siguientes:

#### *Serial*

Lectura y escritura por el puerto serie.

#### *EEPROM*

Lectura y escritura en el almacenamiento permanente. Funciones: “*read()*” y “*write()*”

#### *Ethernet*

Conexión a Internet mediante “*Arduino Ethernet Shield*”, el cual puede funcionar como un servidor que acepta peticiones remotas o como cliente. Éste permite hasta cuatro conexiones simultáneas.

#### *LiquidCrystal*

Control del LCD’s con chipset Hitachi HD44780 o compatibles. La biblioteca soporta los modos de 4 y 8 bits.

#### *Servo*

Control de servomotores. A partir de la versión 0017 de Arduino, la biblioteca soporta hasta 12 motores en la mayoría de placas Arduino y 48 en la Arduino Mega. Mediante la función “*attach(número de pin)*” añadimos un servo, y mediante ”*write*” se puede indicar el movimiento en grados que se busca tenga el motor (habitualmente de 0° a 180°).

#### *Stepper*

Control de motores a pasos unipolares o bipolares, y que utiliza las funciones:

- *Stepper(pasos, pin1, pin2)*
- *Stepper(pasos, pin1, pin2, pin3, pin4)*
- *setSpeed(rpm)*
- *step(pasos)*

La función *Stepper* permite iniciar el motor, indicando los pasos que tiene y los pines a los cuales está conectado. Se indica la velocidad a la que queramos que gire en revoluciones

por minuto con la función *setSpeed(rpm)* y se indican los pasos que queremos que avance con la función *step(pasos)*.

#### *Wire*

Envío y recepción de datos sobre una red de dispositivos o sensores mediante *Two Wire Interface (TWI/I<sup>2</sup>C)*.

Además las bibliotecas *Matrix* y *Sprite* de “*Wiring*” son totalmente compatibles con Arduino y sirven para el manejo de matrices de leds.

## IV. PRÁCTICAS EN ARDUINO DUE

Enseguida se describe una serie de 17 prácticas de iniciación en Arduino, con lo que se espera el alumno adquiera los conocimientos básicos para el diseño y elaboración de sus propios proyectos y aplicaciones. En cada una de las prácticas siguientes, se utilizará Arduino DUE, con una alimentación de energía por medio del puerto micro-USB (internamente Arduino posee reguladores de voltaje para trabajar a 3.3 Volts) y un “*protoboard*” para realizar las conexiones con los distintos componentes electrónicos utilizados.

### PRÁCTICA # 1: SALIDAS DIGITALES.

**Objetivo:** Configurar pines de Arduino DUE como salidas digitales.

#### Material

- 3 diodos emisor de luz (led's).
- 3 resistencias de  $220\ \Omega$ .

#### Actividades

- Realizar las conexiones mostradas en las Figuras 2.1 y 2.2.
- Desarrollar un programa, para activar 3 salidas digitales, las cuales se indicaran que están activas energizando los Led's.

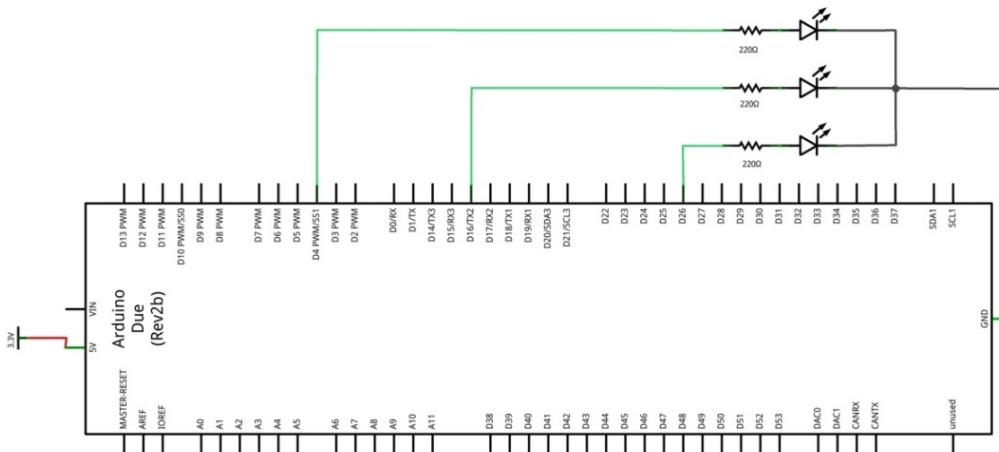


Figura 2.1. Diagrama de conexiones para la práctica #1.

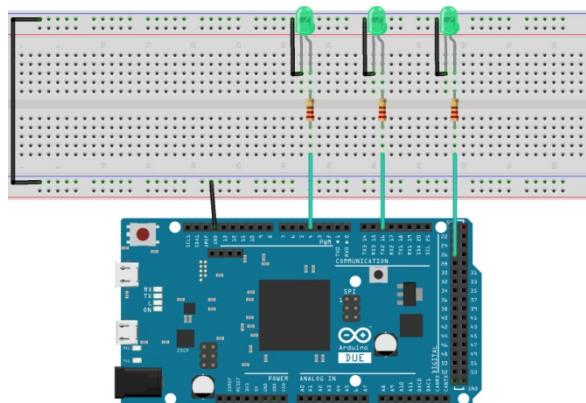


Figura 2.2. Diagrama Esquemático de la práctica #1.

## Descripción del programa

El programa consiste en activar tres salidas digitales de Arduino DUE, para esto, se definen 3 pines como salidas digitales (pines 4, 16 y 26). Para el desarrollo de esta práctica, se definen los pines mencionados anteriormente, aunque se pueden utilizar cualquiera de los pines con etiquetas 0 – 53.

## Programa

```
/*PROGRAMA PARA ACTIVAR PINES 6, 16 Y 26 COMO SALIDAS DIGITALES*/  
int led1 = 6;  
int led2 = 16; // Se definen nombre de las constantes.  
int led3 = 26;  
  
// Configuración de los pines.  
void setup() {  
    pinMode(led1, OUTPUT); // Se declaran los pines como salida.  
    pinMode(led2, OUTPUT);  
    pinMode(led3, OUTPUT);  
}  
void loop() { // Lazo que se ejecutará continuamente.  
    digitalWrite(led1, HIGH); // Escribe "1" lógico a cada uno de los led's.  
    digitalWrite(led2, HIGH);  
    digitalWrite(led3, HIGH);  
}
```

## PRÁCTICA # 2: EL TEMPORIZADOR

**Objetivo:** Implementar la función “*delay*” como temporizador con retardo a la conexión/desconexión.

### Material.

- 3 diodos emisor de luz (led's).
- 3 resistencias de 220 Ω.

### Actividades

- Utilizando el diagrama de conexiones de la práctica #1 (ver Figuras 2.1 y 2.2), realizar una secuencia en la cual los led's se activen/desactiven consecutivamente cada 500 ms.

### Descripción del programa

Teniendo como base el “*sketch*” anterior, donde se declaran algunos pines como salida digital, se le agregará la función “*delay*”, la cual nos permite agregar tiempos muertos.

### Programa

```
/*PROGRAMA QUE ACTIVA CADA 500ms LOS PINES 6, 16 Y 26*/  
int led1 = 6;  
int led2 = 16; // Se definen nombre de las constantes  
int led3 = 26; // Configuracion de los pines.  
  
void setup() {  
    pinMode(led1, OUTPUT); // Se declaran los pines como salida.  
    pinMode(led2, OUTPUT);  
    pinMode(led3, OUTPUT);  
    digitalWrite(led1, LOW); // Iniciar en '0' lógico los pines  
    digitalWrite(led2, LOW);  
    digitalWrite(led3, LOW); } // Lazo que se ejecutará continuamente.  
  
void loop() {  
    digitalWrite(led1, HIGH); // Escribe "1" lógico a led1.  
    delay(500); // Funcion que genera tiempo muerto de 500ms  
    digitalWrite(led1, LOW); // Escribe '0' lógico a led1.  
    delay(500);  
    digitalWrite(led2, HIGH);  
    delay(500);  
    digitalWrite(led2, LOW);
```

```
delay(500);  
digitalWrite(led3, HIGH);  
delay(500);  
digitalWrite(led3,LOW);  
delay(500); }
```

## PRÁCTICA #3: CONFIGURACIÓN DE ENTRADAS DIGITALES.

**Objetivo:** Configurar un pin o terminal de la tarjeta como entrada digital de Arduino DUE utilizando la función “*digitalRead()*”.

### Material

- 1 diodo emisor de luz (led).
- 1 resistencia de  $220\ \Omega$ .
- 1 *push-button* normalmente abierto.
- 1 resistencia de  $10\ k\Omega$ .

### Actividades

- Implementar el diagrama de conexiones de las Figuras 2.3 y 2.4.
- Desarrollar un “*sketch*” para leer una entrada digital. Cuando la entrada sea un ‘1’ lógico, encender el led, de lo contrario mandar un ‘0’ lógico al led.

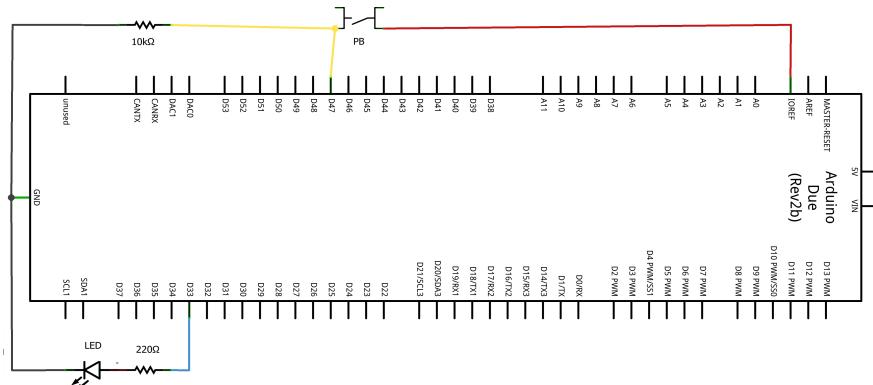


Figura 2.3. Diagrama de conexiones para la práctica #3.

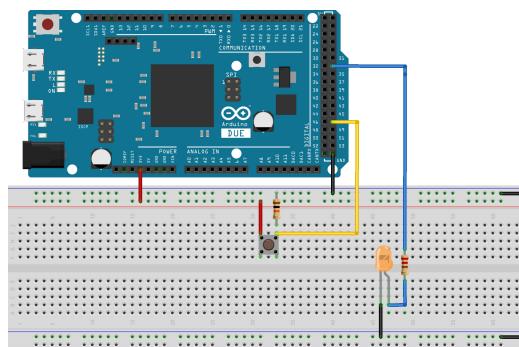


Figura 2.4. Diagrama esquemático para práctica #3.

## Descripción del programa

El siguiente programa lee una entrada digital (pin 47), y dependiendo de su estado lógico activará/desactivará un indicador (led) conectado al pin “33” definido como salida digital.

### Programa

```
/* PROGRAMA QUE LEE UNA ENTRADA DIGITAL */  
  
int pb = 47;                                // Define pines de entrada/salida  
int led = 33;  
int estado = 0;                                // Variable auxiliar  
void setup() {  
    pinMode(led, OUTPUT);                      // Configuración de pines  
    pinMode(pb, INPUT);  
}  
void loop(){  
    estado = digitalRead(pb);                  // Variable estado guarda el valor lógico de “pb”  
    if (estado == HIGH) {  
        digitalWrite(led, HIGH);                // Verifica estado lógico de variable de entrada.  
    }  
    else {  
        digitalWrite(led, LOW);                // Si no se cumple condición desactiva led  
    }  
}
```

## PRÁCTICA #4: IMPLEMENTACIÓN DE SEMÁFORO

**Objetivo:** Implementar un semáforo para el cruce de dos avenidas aplicando los conocimientos adquiridos en prácticas anteriores. Además agregar un botón con el cual ambos semáforos activen la luz ámbar temporalmente.

## Material

- 2 led's (Verdes).
  - 2 led's (Amarillo).
  - 2 led's (Rojo).
  - 6 resistencias 220 Ω.
  - 1 *push-button* normalmente abierto.
  - 1 resistencia de 10 kΩ.

## Actividades

- Implementar el diagrama mostrado en las Figuras 2.5 y 2.6.
  - Implementar un programa, el cual simule el control vehicular (semáforo) para el cruce de dos avenidas. Considerando que en ambos semáforos la señal de alto/siga, tiene un periodo de activación/desactivación de 5 segundos, y la luz preventiva oscila a 2 Hz durante dos periodos. Al activar el *push-button* ambas luces preventivas deben activarse por 100 ms y luego apagarse por 100 ms, y enseguida se retoma la operación normal del semáforo.

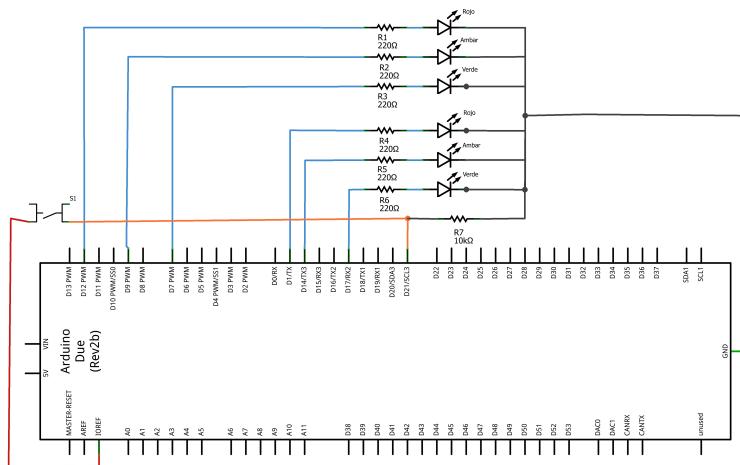


Figura 2.5. Diagrama de conexiones para implementación de la práctica #4.

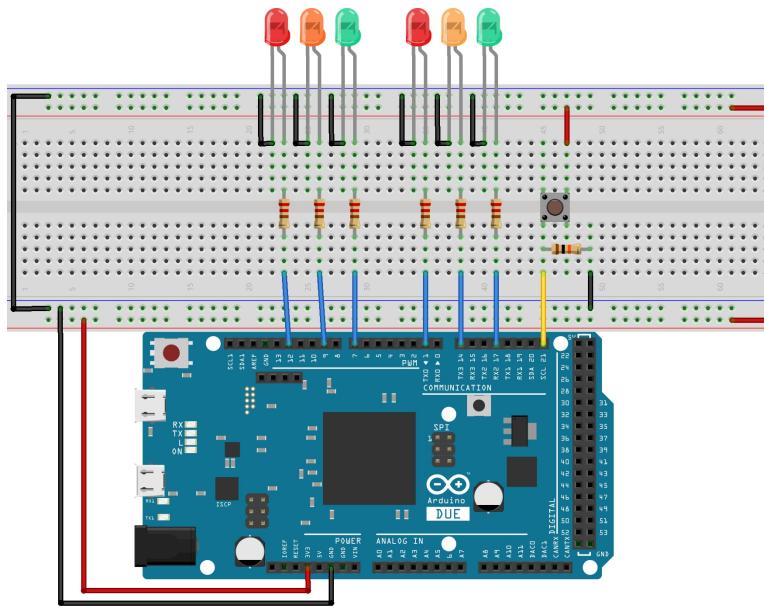


Figura 2.6.- Diagrama esquemático para la práctica #4.

## Programa

```
/* PROGRAMA QUE IMPLEMENTA UN SEMÁFORO PARA 2 AVENIDAS */  
  
int v1 = 12; // Se definen nombre de las constantes.  
int a1 = 9;  
int r1 = 7;  
int v2 = 1;  
int a2 = 14;  
int r2 = 17;  
int pb = 21;  
int b;  
  
//Configuracion de los pines.  
void setup() {  
    pinMode(v1,OUTPUT); // Se declaran los pines como salida.  
    pinMode(a1,OUTPUT);  
    pinMode(r1,OUTPUT);  
    pinMode(v2,OUTPUT); // Se declaran los pines como salida.  
    pinMode(a2,OUTPUT);  
    pinMode(r2,OUTPUT);  
    pinMode(pb,INPUT); // pb se define como entrada.  
    digitalWrite(v1, LOW);  
    digitalWrite(a1, LOW);  
    digitalWrite(r1, LOW);  
    digitalWrite(v2, LOW);  
    digitalWrite(a2, LOW);  
    digitalWrite(r2, LOW);  
    digitalWrite(pb, LOW);  
    // Borrar pines utilizados
```

```
}

void loop(){
    b = digitalRead(pb);                                // Verifica el estado de pb.
    if (b == LOW) {
        digitalWrite(v1,HIGH);
        digitalWrite(r2,HIGH);
        delay(5000);
        digitalWrite(v1,LOW);
        digitalWrite(a1,HIGH);
        delay(500);
        digitalWrite(a1,LOW);
        delay(500);
        digitalWrite(a1,HIGH);
        delay(500);
        digitalWrite(a1,LOW);
        digitalWrite(r2,LOW);
        digitalWrite(r1, HIGH);
        digitalWrite(v2,HIGH);
        delay(5000);
        digitalWrite(v2,LOW);
        digitalWrite(a2,HIGH);
        delay(500);
        digitalWrite(a2,LOW);
        delay(500);
        digitalWrite(a2,HIGH);
        delay(500);
        digitalWrite(a2,LOW);
        digitalWrite(r2,HIGH);
        digitalWrite(r1, LOW);
    }
    else {
        digitalWrite(r2, LOW);
        digitalWrite(a1,HIGH);
        digitalWrite(a2,HIGH);
        delay(100);
        digitalWrite(a1,LOW);
        digitalWrite(a2,LOW);
        delay(100);
    }
}
```

## PRÁCTICA #5: ESTRUCTURA DE CONTROL “FOR”

**Objetivo:** Utilizar la sentencia “*for*” para definir salidas, así como activar salidas secuencialmente.

### Material:

- 6 led's (cualquier color)
- 6 resistencias de 220 Ω

### Actividades:

- Implementar el diagrama mostrado en las Figuras 2.7 y 2.8.
- Realizar un programa para activar/desactivar secuencialmente 6 led's utilizando un ciclo “*for*”. Los pines utilizados como salidas digitales para esta secuencia serán los enumerados del 2 al 7.

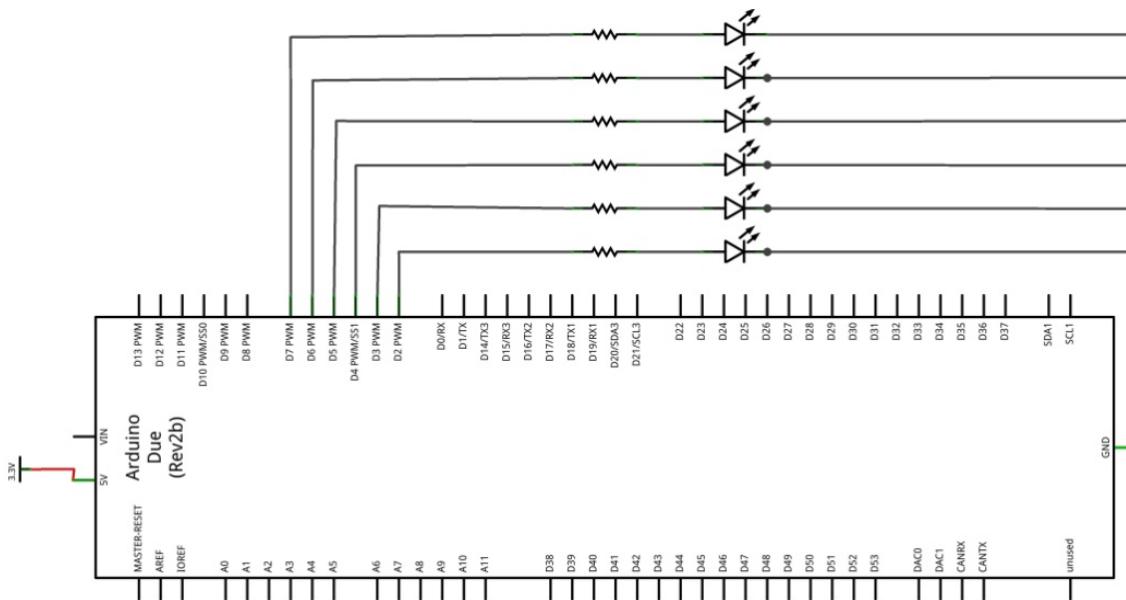


Figura 2.7. Diagrama de conexiones para práctica 5.

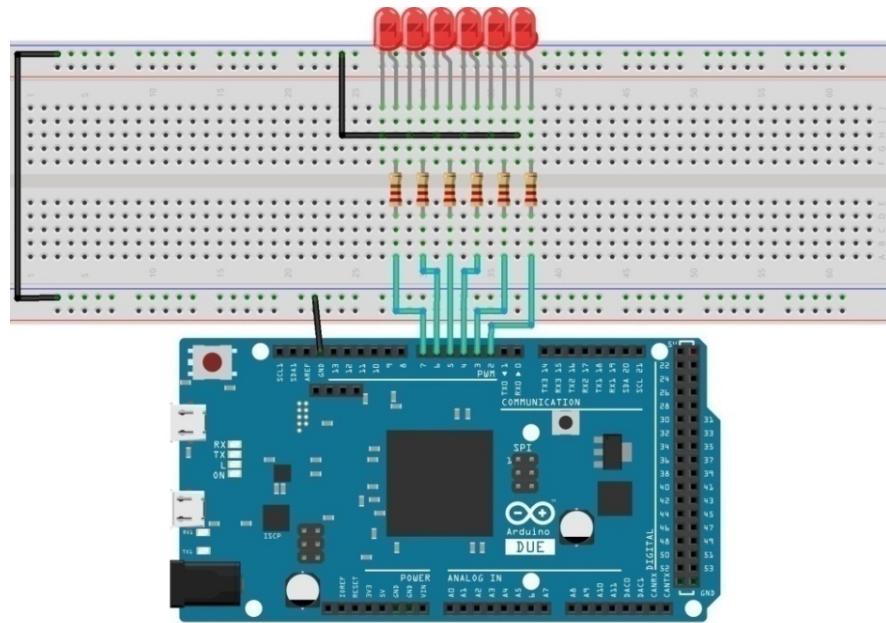


Figura 2.8. Diagrama esquemático practica #5.

## Descripción del programa

El “*sketch*” realiza el encendido de 6 led’s secuencialmente cada 100 ms, una vez que llega al final regresa la secuencia con la misma frecuencia y se iniciará nuevamente el programa.

## Programa

```
/* PROGRAMA QUE IMPLEMENTA UNA SECUENCIA DE TIEMPO EN 6 LED'S*/
void setup() {
    for (int pin = 2; pin < 8; pin++) { // Declaración de salidas.
        pinMode(pin, OUTPUT);
    }
}

void loop() {
    for (int pin = 2; pin < 8; pin++) { // Activa led.
        digitalWrite(pin, HIGH);
        delay(100);
        digitalWrite(pin, LOW); // Desactiva led.
    }
    for (int pin = 8; pin >= 2; pin--) {
        digitalWrite(pin, HIGH);
        delay(100);
        digitalWrite(pin, LOW); }
}
```

## PRÁCTICA #6: ENTRADA ANALÓGICA Y PUERTO SERIAL

**OBJETIVO:** Identificar en Arduino DUE las entradas analógicas, así como configurar el puerto serial de Arduino para su visualización.

### Material

- Potenciómetro de  $1k\Omega$  (tipo: *through hole* o *pcb*)

### Actividades

- Implementar el circuito mostrado en las Figuras 2.9 y 2.10.
- Desarrollar un programa en el que se configure un pin como entrada analógica, la cual estará variando su nivel de voltaje ( $0 - 3.3V$ ) a través de un potenciómetro.
- Para visualizar los datos en el puerto serie, ir al menú Herramientas y seleccionar monitor serie.

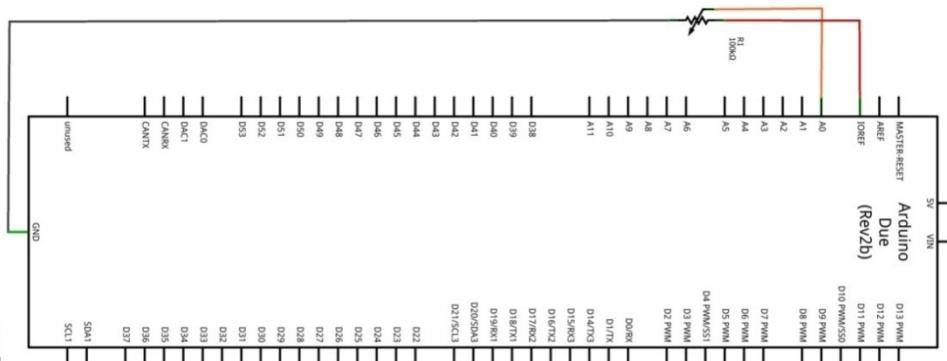


Figura 2.9. Diagrama de conexiones para la práctica 6.

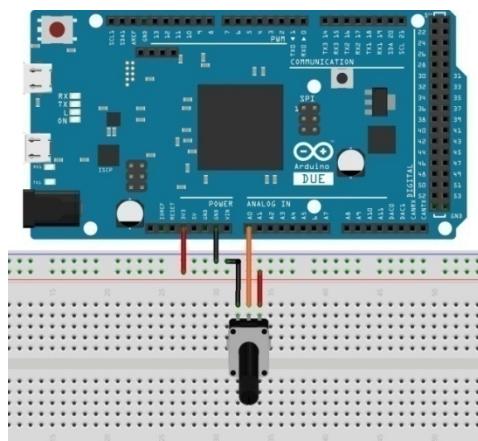


Figura 2.10. Esquemático para práctica #6.

## Descripción del programa

En el siguiente programa se visualiza la entrada analógica (A0) a través del monitor serie de Arduino DUE, la cual está variando en un rango de 0 a 3.3 Volts a través de un potenciómetro.

## Programa

```
/* PROGRAMA QUE LEE UNA ENTRADA ANALÓGICA*/
void setup() {
    Serial.begin(9600);                      // Configura el Puerto serie.
}
void loop() {
    int sensorValue = analogRead(A0);          // Lee entrada A0
    float voltage = sensorValue * (3.3 / 1024.0); // Convierte a valor 0 – 3.3v
    Serial.println(voltage);                  // Envía datos a Puerto serie
}
```

## PRÁCTICA #7: MODULACIÓN DE ANCHO DE PULSO (PWM)

**OBJETIVO:** Variar la velocidad de un motor de CD de 5 Volts a través de una salida PWM de Arduino DUE.

### Material

- 1 potenciómetro de  $1k\Omega$  (tipo: *through hole* o *pcb*).
- 1 transistor NPN 2N3904
- 1 diodo de conmutación rápida 1N4001
- 1 motor de CD de 5Volts.

### Actividades

- Realizar las conexiones mostradas en las Figuras 2.11 y 2.12.
- Implementar un programa en el cual a partir del nivel de entrada analógica (0 a 3.3V) se pueda variar la velocidad de un motor de CD de 5 Volts.

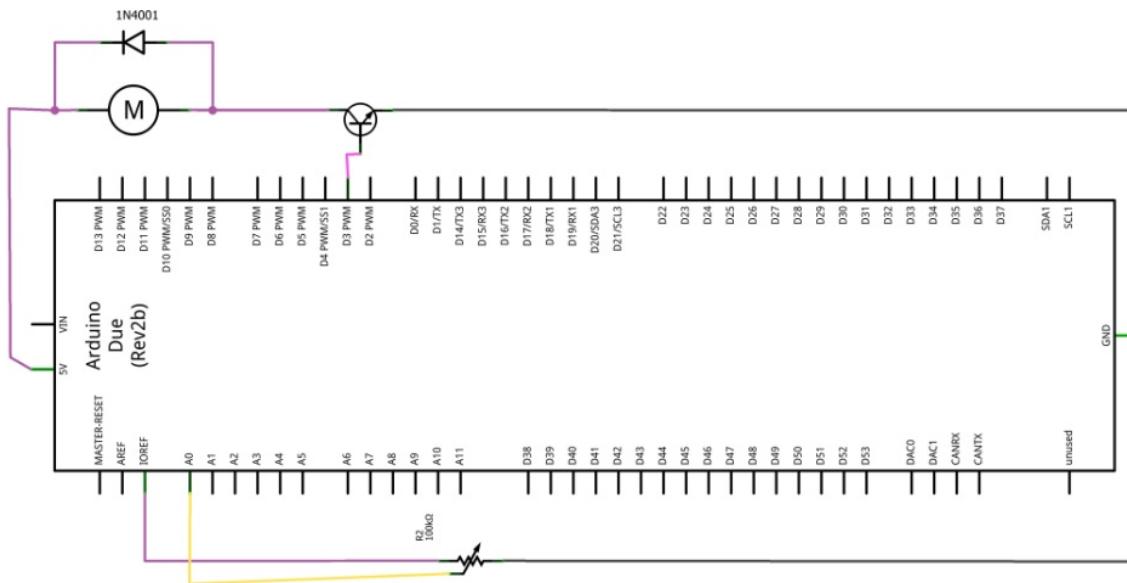


Figura 2.11. Diagrama de conexiones para la práctica 7.

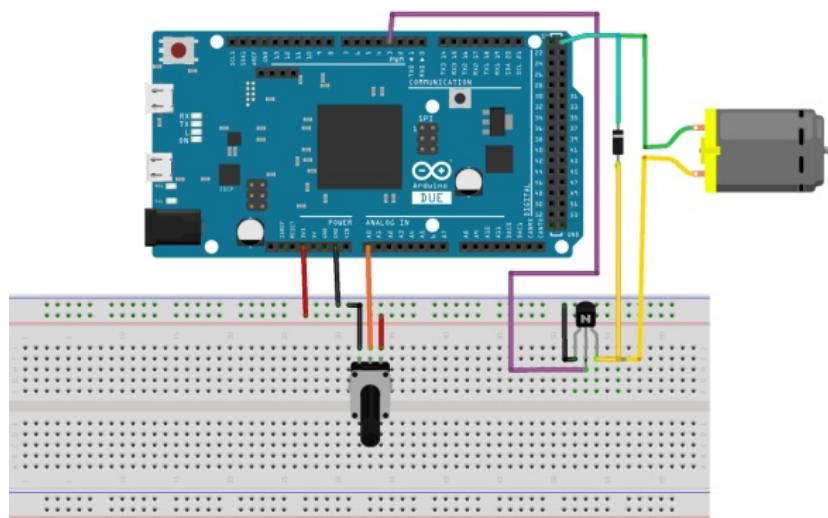


Figura 2.12. Esquemático para práctica #7.

## Descripción del programa

El programa lee una entrada analógica en el pin A0, la cual está variando su nivel de voltaje a través de un potenciómetro en un rango de 0 a 3.3 Volts. Posteriormente se envía a un pin del tipo PWM (pin 3) de Arduino DUE, para esta ser aplicada a la base de un transistor que nos permitirá utilizar la señal para la variación de velocidad de un motor de CD de 5 volts.

## Programa

```
/* PROGRAMA QUE REGULA LA VELOCIDAD DE UN MOTOR CD*/
int In = A0; // Entrada A0
int valor = 0; // Valor comienza en 0.
int s = 3; // Pin de salida.
void setup() {
    pinMode(s, OUTPUT); // variable "s" como salida
}
void loop() {
    valor = analogRead(In); // Lee A0 (resolución de 10 bits.
    analogWrite(s,valor/4); // Escribe en 's' resolución de 8 bits.
}
```

## PRÁCTICA #8: DISPLAY DE CRISTAL LÍQUIDO

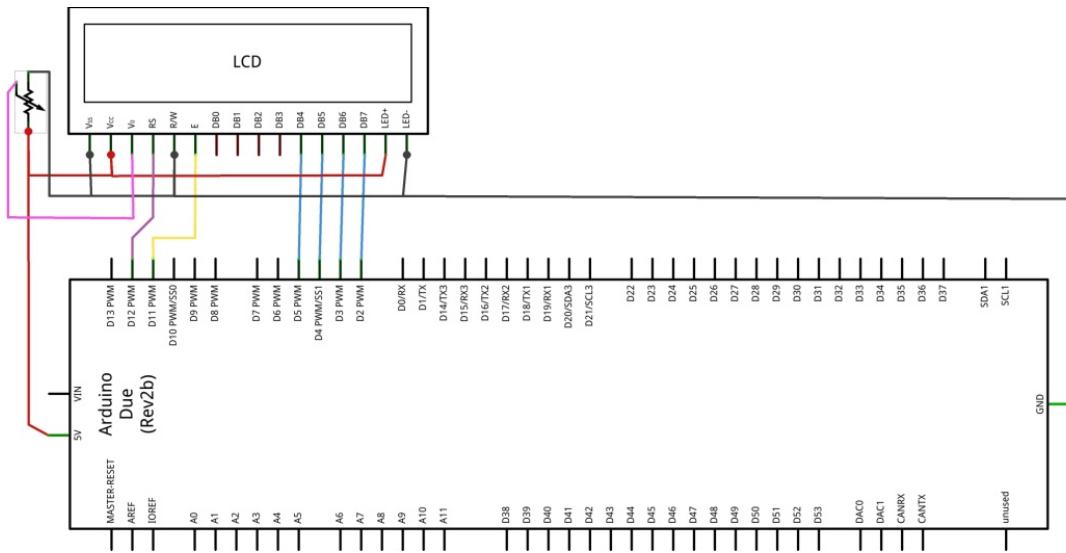
**Objetivo:** Utilizar la librería “*LiquidCrystal.h*” de Arduino para el manejo de LCD’s con chip Hitachi HD44780

### Material

- 1 potenciómetro de  $1k\Omega$  (tipo: *through hole* o *pcb*).
- 1 display de cristal liquido de 16x2 (chip Hitachi HD44780).

### Actividades

- Implementar el diagrama de conexiones de las Figuras 2.13 y 2.14.
- Implementar un programa utilizando la librería “*LiquidCrystal.h*” para configurar el “display” donde se muestre un mensaje de texto.



**Nota:** El potenciómetro regula el nivel de contraste en la pantalla LCD.

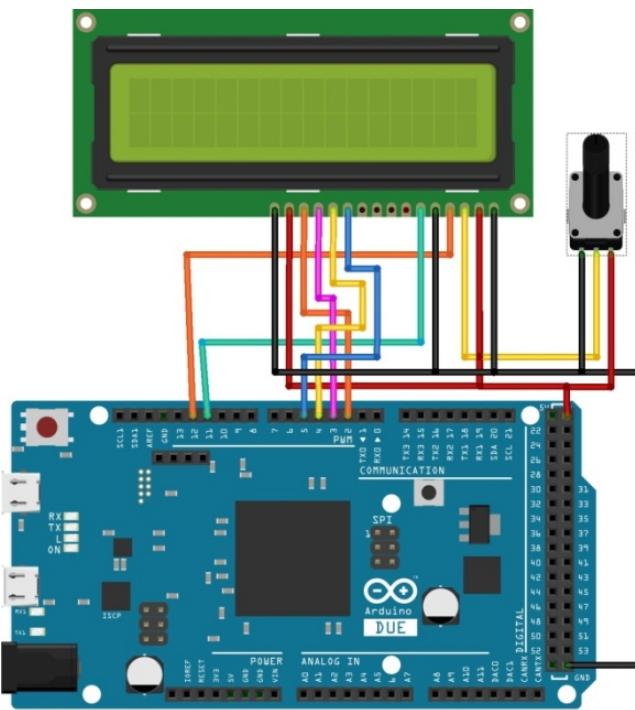


Figura 2.14. Esquemático para práctica #8.

## Descripción del programa

Utilizando la librería de “*LiquidCrystal.h*” (compatible con chips Hitachi HD44780), se configura el display de 16x2, para mostrar un mensaje de texto.

## Programa

```
/* PROGRAMA QUE DESPLIEGA UN MENSAJE DE TEXTO EN UN LCD*/
#include <LiquidCrystal.h> // Incluir librería para manejo del LCD.

LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // Configuración de los pines.

void setup() {
    lcd.begin(16, 2); // Especifica numero de columnas y renglones de lcd.
    lcd.print("Fac. Ciencias!"); // Imprime en lcd "Fac. Ciencias".
}

void loop() {
    lcd.setCursor(0, 1); // Segunda línea del display.
    lcd.print(millis()/1000); // Imprime numero de segundos.
}
```

## PRÁCTICA #9: MEDICIÓN DE TEMPERATURA

**Objetivo:** Realizar la medición de temperatura a través de una entrada analógica y mostrarla en un LCD.

### Material

- 1 sensor de temperatura lm35.
- 1 potenciómetro de  $1k\Omega$  (tipo: *through hole* o *pcb*).
- 1 display LCD 16x2 (chip Hitachi HD44780).

### Actividades

- Implementar el diagrama de conexiones de las Figuras 2.15 y 2.16.
- Implementar un programa utilizando la librería “LiquidCrystal” para configurar el display donde muestre la temperatura obtenida a través del sensor lm35.
- Acondicionar la señal del sensor lm 35 en el programa para desplegar la temperatura en °C.

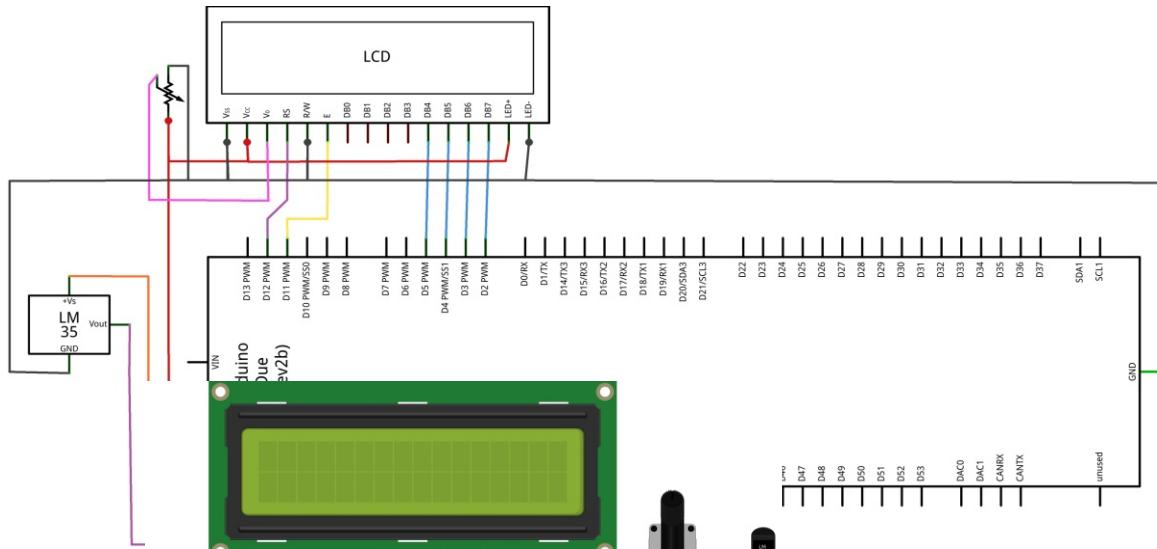


Figura 2.15.  
para la práctica 9.

Diagrama de conexiones

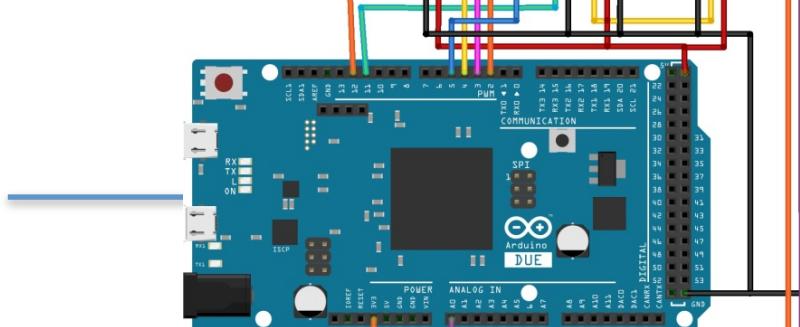


Figura 2.16. Esquemático para práctica #9.

**Nota:** El potenciómetro regula el nivel de contraste en la pantalla LCD.

### Descripción del programa

El programa consiste en leer una entrada analógica proveniente del sensor de temperatura lm35. La señal recibida se acondiciona para realizar la conversión a °C y mostrarla en el display LCD (ver anexos para el “*datasheet*” del sensor lm35).

### Programa

```
/* PROGRAMA QUE LEE LA TEMPERATURA Y LA MUESTRA EN UN LCD*/
#include <LiquidCrystal.h>                                // Cargar librería de lcd.
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);                      // Configuración del lcd
void setup() {
    lcd.begin(16, 2);                                       // Define tamaño del lcd
    lcd.print("Temperatura");                                // Imprime "Temperatura"
}
void loop() {
    analogReadResolution(12);                                // Resolución ADC y DAC
    analogWriteResolution(12);
    int sensorValue = analogRead(A0);                        // Lee A0
    float voltaje = sensorValue * (3.3/4095.0);           // Convierte a voltaje
    lcd.setCursor(0, 1);                                     // Imprime en segunda línea
    lcd.print(voltaje*100);                                  // Convierte a temperatura
}
```

## PRÁCTICA #10: CONTADOR DE PULSOS

**OBJETIVO:** Visualizar a través del monitor serie de Arduino DUE el número de pulsos que ingresan a través de una entrada digital para activar una salida, cuando el número de pulsos sobrepase a una cantidad establecida por el alumno.

### Material

- 1 *push-button* normalmente abierto.
- 1 resistencia de 10 kΩ.
- 1 led.
- 1 resistencia de 220 Ω.

### Actividades

- Implementar el diagrama de conexiones mostrado en las Figuras 2.17 y 2.18.
- Desarrollar un programa para contar el número de pulsos realizados a través del “*push-button*” así como realizar acciones de acuerdo a cierta cantidad de pulsos realizados.

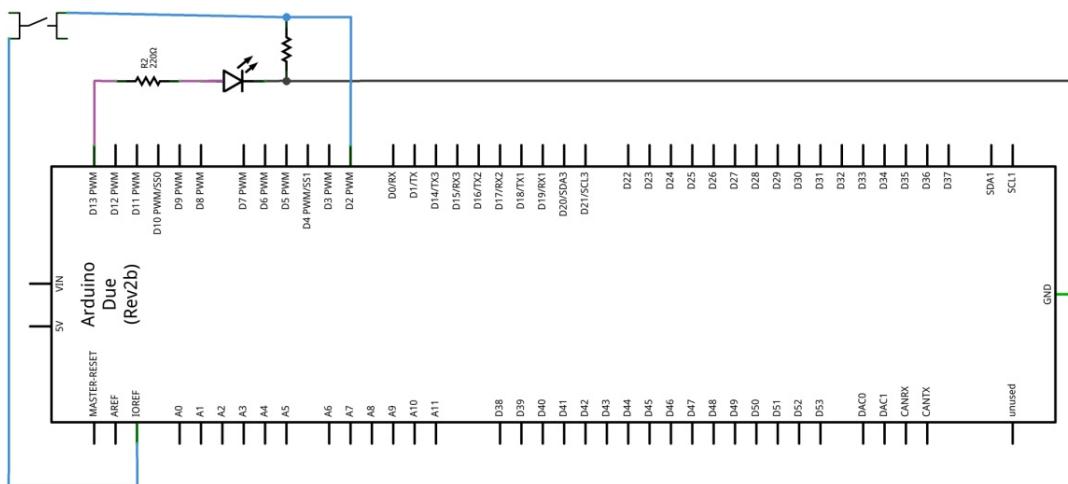


Figura 2.17. Diagrama de conexiones para la práctica 10.

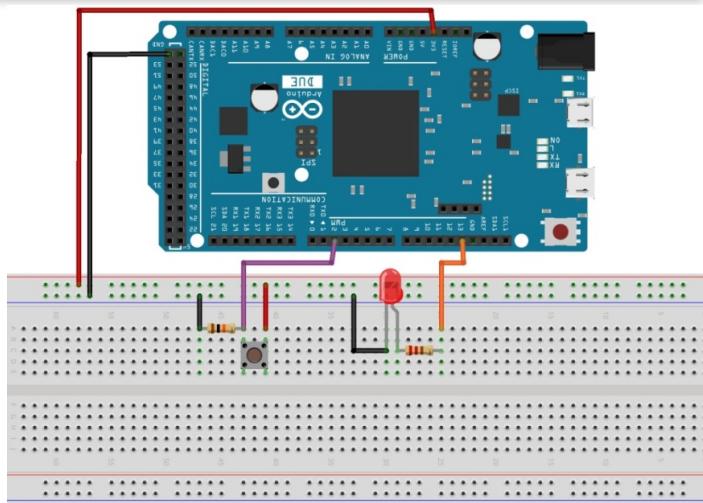


Figura 2.18. Diagrama esquemático práctica #10.

## Descripción del programa

El programa lee una entrada digital (“*push-button*”), la cual se va incrementando y guardando en una variable, de acuerdo a la cantidad de pulsos realizados, y enciende el led después de 5 pulsos y lo apaga después de 8.

## Programa

```
/* PROGRAMA QUE LEE LA CANTIDAD DE PULSOS INGRESADA*/
int cont = 0;                                // Guarda conteo de pulsos.
void setup() {
    Serial.begin(9600);                      // Inicia comunicación serial
    pinMode(2,INPUT);                        // Configura el pin 2 como una entrada, pulsador
    pinMode(13,OUTPUT);                      // Configura el pin 13 como una salida →led
    digitalWrite(13, LOW);                   // Asegurar un led apagado al iniciar
}
void loop()
{
    if ( digitalRead(2) == HIGH ){           // Verifica si “push-button” ha sido presionado
        delay(50);                          // Elimina rebotes
        if ( digitalRead(2) == LOW ){         // Si ocurrió un cambio incrementa “cont”
            cont++;
            Serial.println(cont);          // Envía “cont” al puerto serie
            delay (100); }                // Retardo
    }
    if (cont==5){                           // Si el valor del contador es 5
        digitalWrite(13,HIGH);}            // Activa el led
    if (cont==8){                           // Si el valor del contador es 8
        digitalWrite(13,LOW);}             // Desactiva el led
}
```

## PRÁCTICA #11: MANEJO DE INTERRUPCIONES

**Objetivo:** Configurar un pin de entrada digital para activar una función de interrupción en un “sketch” de Arduino.

### Material

- 1 *push-button* normalmente abierto.
- 1 resistencia de  $10\text{ k}\Omega$ .
- 2 led's.
- 2 resistencias de  $220\ \Omega$ .

### Actividades

- Implementar el diagrama de conexiones mostrado en las Figuras 2.19 y 2.20.
- Desarrollar un programa donde se defina una función que será ejecutada como interrupción al cambiar de estado una entrada digital.

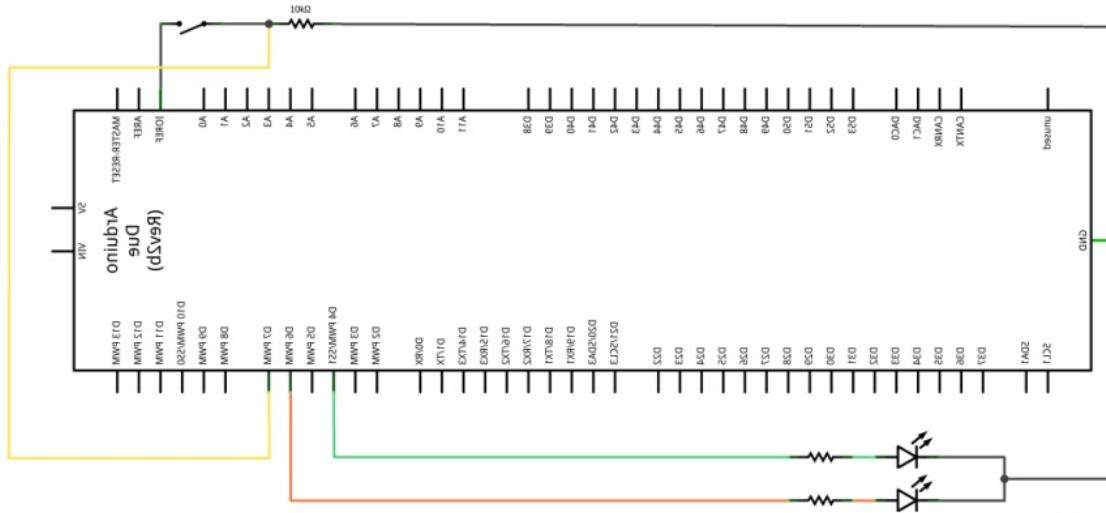


Figura 2.19. Diagrama de conexiones para la práctica #11.

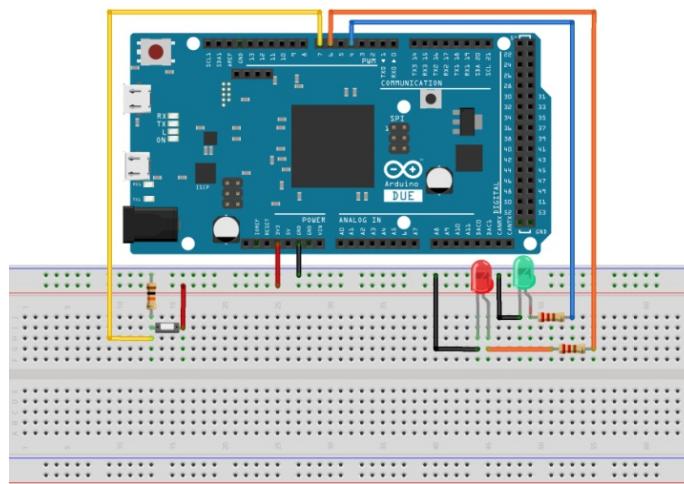


Figura 2.20. Diagrama esquemático práctica #11.

### Descripción del programa

La función principal del programa realiza una secuencia de encendido/apagado cada 5 segundos de un led (led1), cuando ocurre una interrupción generada por la entrada digital, se ejecuta un ciclo “for” que activa otro led (led2) durante 3 segundos, una vez concluida la función de interrupción, regresa a la ejecución de la función principal.

### Programa

```
/* PROGRAMA QUE DESCRIBE EL USO BÁSICO DE INTERRUPCIONES*/
int pb = 7;                                // Declaración de Variables
int led1 = 4;
int led2= 6;
void setup () {
    pinMode (pb, INPUT);                  // Configuración Entradas/Salidas
    pinMode (led1, OUTPUT );
    pinMode (led2, OUTPUT );
    digitalWrite(led2,LOW);                // Led2 comienza desactivado
    attachInterrupt(pb, interrupcion, CHANGE); // Definición de interrupción

void loop () {                                // Función principal
    digitalWrite(led1,HIGH);
    delay (5000);
    digitalWrite(led1, LOW);
    delay (5000);}

void interrupcion(){                         // Se ejecuta cuando existe un cambio de estado en "pb"
    for (int i=0; i <= 500000; i++) {
        digitalWrite(led2, HIGH); }           // Ciclo para generar 3 segundos
    digitalWrite(led2, LOW); }                // Desactiva led2 y retorna a ejecución del programa.
```

## PRÁCTICA #12: GENERADOR DE SEÑAL DIENTE DE SIERRA

**Objetivo:** Desarrollar un programa en el cual se genere una señal diente de sierra.

### Material:

- 1 osciloscopio con entrada analógica.

### Actividades:

- Implementar el programa en Arduino DUE que incrementa unitariamente una variable dentro de un rango 0 a 4096 (12 bits), y al alcanzar el valor máximo se inicializa.
- Visualizar en el osciloscopio la señal en la salida DAC1 de Arduino DUE (ver Figura 2.21).

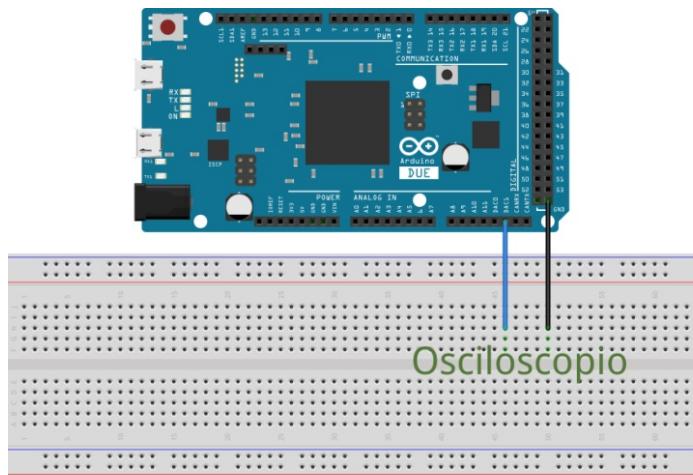


Figura 2.21. Diagrama esquemático practica #12.

### Descripción del programa

Se comienza por definir la resolución del DAC a 12 bits, lo cual permite tener hasta 4096 valores, y enseguida se inicializa la variable de control con “0”, para posteriormente ir incrementando unitariamente la variable “*valor*” e ir enviando los datos al convertidor Digital – Analógico. Para elevar la frecuencia de la señal, se podría definir un paso mayor en cada incremento de la variable, o de forma equivalente un número de incrementos menor en el intervalo 0 a 4096.

## Programa

```
/*PROGRAMA QUE GENERA UNA SEÑAL ANALÓGICA DIENTE DE SIERRA*/
// Definición de Variables
int numincrementos=4096; // Resolución de 12 bits
int valor=0;
int incremento=4096/numincrementos;
int valormaximo=incremento*numincrementos;

void setup() {
    analogWriteResolution(12); // Define resolución del DAC
}
void loop() {
    analogWrite(DAC1, valor); // Escribir al DAC la variable valor
    valor+=incremento; // Incrementa valor en 1
    valor=(valor>valormaximo?0:valor); // Compara si ha llegado a valor máximo
}
```

## Resultados

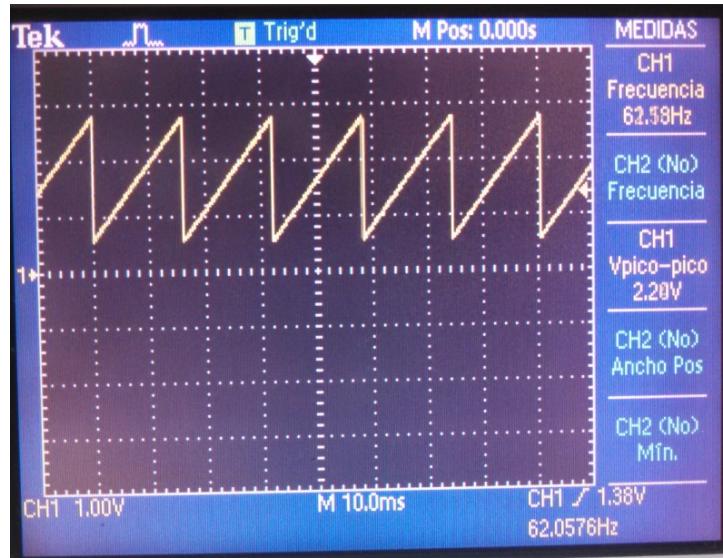


Figura 2.22. Señal obtenida a frecuencia de 60 Hz

## PRÁCTICA #13: LAZO ADC – DAC (TRASLAPE)

**Objetivo:** Configurar la resolución de los convertidores Analógico – Digital y Digital – Analógico, y observar el fenómeno de traslape (*aliasing*) al aumentar la frecuencia de la señal de entrada por encima de la tasa de muestreo de Nyquist.

### Material:

- 1 osciloscopio con entrada analógica.
- 1 generador de funciones
- 1 Resistencia de 330 Ω.
- 1 Diodo Zener 1N4372A (3 Volts a ½ Watt)

### Actividades:

- Implementar el circuito mostrado en las Figuras 2.23 y 2.24 para limitar la amplitud de la señal de entrada, así como para proteger la entrada analógica de Arduino DUE.
- Implementar el programa sugerido en Arduino DUE.
- Configurar el generador de funciones con una señal positiva a una frecuencia de 1KHz, con una forma de onda senoidal y amplitud no mayor a 3.3 Volts pico a pico (amplitud entre 0 y 3.3 V) para que la señal sea aplicada a la entrada A5.
- Visualizar en el osciloscopio la señal en la salida (DAC1).
- Aumentar la frecuencia de 1 KHz a 20 KHz, y modificar también forma de la señal de entrada (triangular y onda cuadrada). ¿Cuál es el efecto en la señal de salida?
- Modificar la resolución del ADC – DAC.

### Notas:

- La frecuencia de muestreo aproximada del Arduino DUE con la configuración anterior es aproximadamente 22.5 KHz. Por lo que la frecuencia de Nyquist es de alrededor de 11.25 KHz, y a partir de esta frecuencia de la señal de entrada se tendrá traslape y la salida tendrá una frecuencia diferente.
- La amplitud de la señal muestreada estará entre 1/6 y 5/6 de la amplitud de la señal original (dato del fabricante del microcontrolador). Por ejemplo si se elige amplitud

pico a pico de 3.3 Volts, la señal de salida tendrá una disminución en amplitud de un 20%.

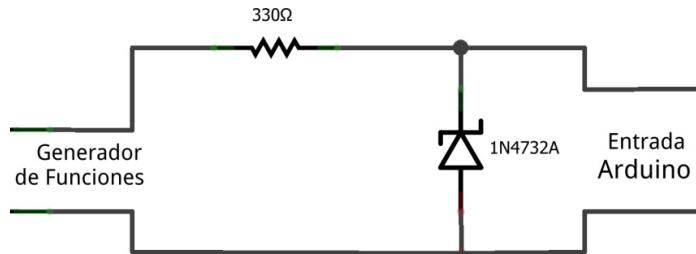


Figura 2.23. Circuito limitador de amplitud.

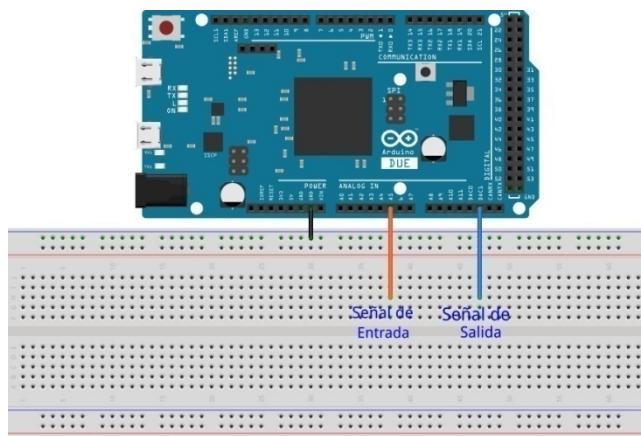


Figura 2.24. Diagrama esquemático practica #13.

## Descripción del programa

Se lee una entrada analógica (A5) con resolución de 12 bits, la cual es enviada directamente al convertidor Digital Analógico (DAC1).

## Programa

```
/* PROGRAMA QUE IMPLEMENTA UN ENLACE DIRECTO ADC-DAC PARA OBSERVAR EL
TRASLAPE*/
void setup() {
analogReadResolution(12); // Configuración de resolución
analogWriteResolution(12);
}
void loop() {
float valor = analogRead(A5); // Lee valor analógico en A5
float voltaje= valor * (3.3 / 4095); // Acondiciona la amplitud de la señal
analogWrite(DAC1, valor ); // Envía datos al DAC1
}
```

## Resultados

Una señal senoidal de entrada se visualiza en el canal 2 (color azul) y las salidas a 1KHz y 20KHz se muestran en el canal 1, donde se observa claramente el traslape a la frecuencia de 20 KHz, lo que resulta en una frecuencia de salida de 2.5 KHz (22.5 KHz – 20 KHz).

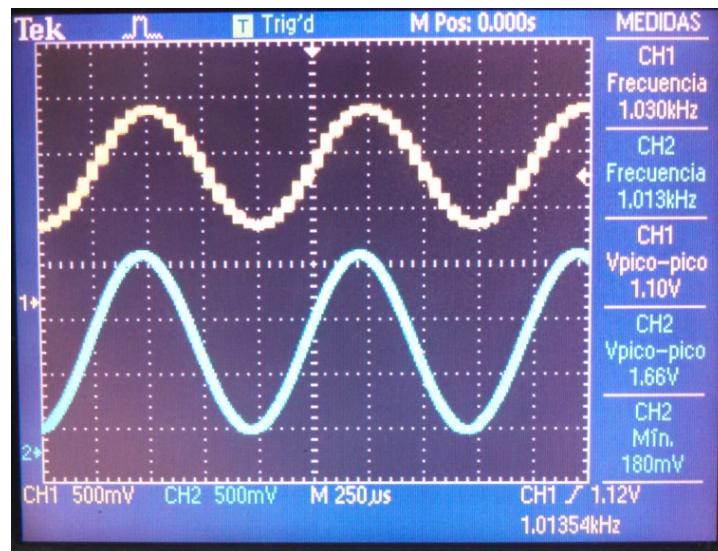


Figura 2.25.Señal de salida del DAC1(CH1) a 1KHz.

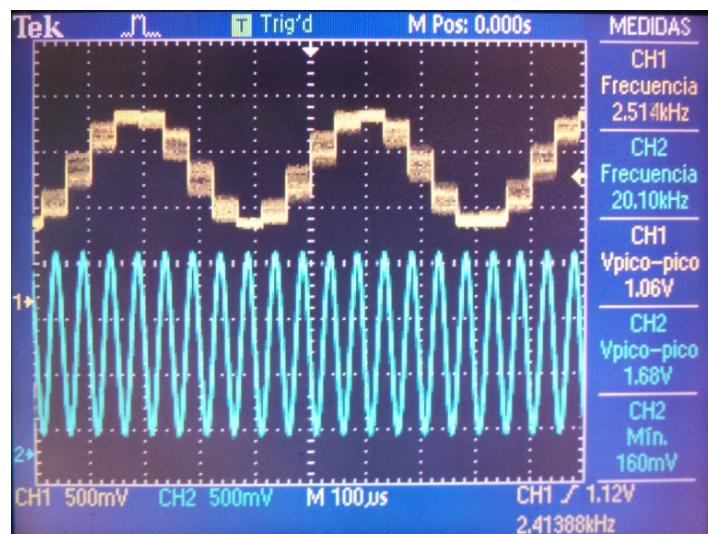


Figura 2.26.Señal de salida del DAC1(CH1) a 20 KHz.

## PRÁCTICA #14: GENERADOR DE SEÑAL SENOIDAL

**Objetivo:** Desarrollar un programa en el cual se genere una señal senoidal analógica a partir de un registro de muestras de la misma y del empleo del DAC, para observar el concepto de resolución en una conversión D/A.

### Introducción.

Para generar una señal senoidal en Arduino DUE, se comienza por definir la cantidad de muestras que se toman de la señal senoidal. Para ésta práctica, se consideran 51 muestras. Para obtener los 51 coeficientes en que se divide la señal, se puede utilizar algún software para simplificar el cálculo numérico. Utilizando MATLAB, se define el siguiente procedimiento: primero se construye un vector de 51 muestras de la señal senoidal que solo tenga valores positivos en el rango 0 a 3.3, y con una frecuencia de 1 Hz:

```
>> n = 0:50; % Cantidad de muestras.  
>> v = 1.15*sin(2*pi*1*n/50)+1.15 % Señal positiva
```

v =

1.1500	1.2941	1.4360	1.5733	1.7040	1.8260	1.9372	2.0361
2.1210	2.1906	2.2437	2.2796	2.2977	2.2977	2.2796	2.2437
2.1906	2.1210	2.0361	1.9372	1.8260	1.7040	1.5733	1.4360
1.2941	1.1500	1.0059	0.8640	0.7267	0.5960	0.4740	0.3628
0.2639	0.1790	0.1094	0.0563	0.0204	0.0023	0.0023	0.0204
0.0563	0.1094	0.1790	0.2639	0.3628	0.4740	0.5960	0.7267
0.8640	1.0059	1.1500					

Enseguida, los coeficientes obtenidos, representan las 51 muestras, las cuales se interpretan de la siguiente manera:

- El valor de referencia para la conversión del DAC por “default” se encuentra en 3.3 Volts, considerando que la amplitud después de la conversión se ubica en un rango de 1/6 a 5/6 de la cota máxima, se obtendrá una amplitud máxima de la señal de 2.3 Volts.
- La resolución se establece de la siguiente manera:

$$\text{Resolución} = \frac{2.3}{2^{12}-1} = 5.616 \times 10^{-4} \text{ volts / paso}$$

Por lo que el equivalente digital se calcula como:

$$\text{Valor Digital} = \frac{\text{Coeficiente}}{\text{Resolución}}$$

Por ejemplo para los dos primeros coeficientes, se calculan y se redondea al entero más cercano:

$$\text{Valor Digital} = \frac{1.15}{5.61610^{-4}} = 2,047.7 \approx 2048$$

$$\text{Valor Digital} = \frac{1.2941}{5.61610^{-4}} = 2,304.3 \approx 2304$$

### Material:

- 1 osciloscopio con entrada analógica.

### Actividades:

- Implementar el programa propuesto en Arduino.
- Visualizar en el osciloscopio la señal en la salida DAC1 de Arduino DUE (ver Figura 2.27).

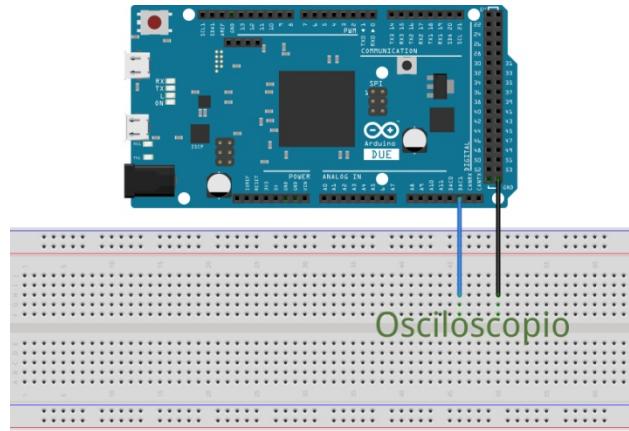


Figura 2.27. Diagrama esquemático práctica #14.

### Descripción del programa

Se comienza por definir los valores digitales de las 51 muestras de la señal senoidal del tipo “array”. Estos valores son enviados al DAC1 mediante un ciclo “for”, hasta completar los 51 valores y nuevamente comenzar el envío de los datos al convertidor D/A.

## Programa

```
/* PROGRAMA QUE GENERA UNA SEÑAL SENOIDAL CON 51 MUESTRAS*/  
int seno[ ] = {2048,2304,2557,2802,3035,3252,3450,3626,  
    3777,3901,3996,4060,4092,4092,4060,3996,  
    3901,3777,3626,3450,3252,3035,2802,2557,  
    2305,2048,1791,1539,1294,1061,844,646,  
    470,319,195,100,36,4,4,36,100,195,319,  
    470,646,844,1061,1294,1539,1791,2048};  
  
void setup() {  
    analogWriteResolution(12);  
    // Configura resolución de 12 bits  
}  
  
void loop() {  
    for(int i = 0; i<50;i++){  
        if(seno[i]>4095){  
            seno[i]=4095;  
        }  
        analogWrite(DAC1, seno[i]);  
        delayMicroseconds(14);  
        // Envio de datos a DAC1  
        // Ajuste de Frecuencia  
    }  
}
```

**Nota:** Si se varia el valor de la función *delayMicroseconds()* se ajusta la frecuencia de salida de la señal.

## Resultados

En la Figura 2.28 se puede observar una muestra de la señal obtenida a la salida del DAC1: señal senoidal positiva, frecuencia de 1 KHz, amplitud pico-pico de 2.3 Volts, Valor máximo de 2.84 Volts (5/6 de 3.3 Volts) y valor mínimo de 520 mV (1/6 de 3.3 Volts).

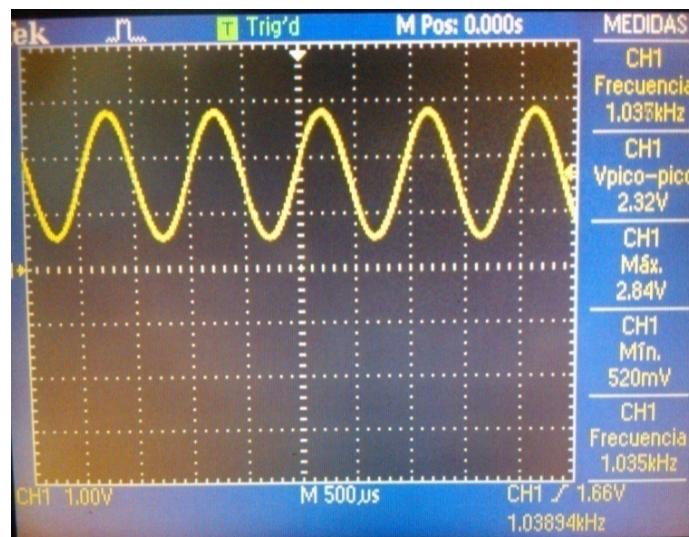


Figura 2.28. Señal obtenida a frecuencia de 1 KHz

## TEMPORIZADOR/CONTADOR

Arduino DUE tiene integrados tres temporizadores de propósito general de 32 bits, los cuales incrementan un contador según el tiempo transcurrido sin detener el flujo del programa, con lo que permite realizar distintas tareas como: Medición de frecuencia, conteo de eventos, medición de intervalos, generador de pulsos, tiempos de retardo y generación de PWM.

El funcionamiento del temporizador o contador en modo normal se incrementa con cada pulso de reloj del microcontrolador hasta llegar a su valor máximo (32 bits) y se reinicia de nuevo desde 0. En el momento que el contador vuelve a 0, se activa una bandera de "*overflow*", donde se puede modificar el tiempo del temporizador o conteo, así como el divisor de frecuencia de reloj que está alimentado al temporizador o "*prescaler*". Los "*prescaler*" disponibles son: 1, 8, 32, 64, 128, 256 y 1024. Se puede configurar una interrupción cada que ocurre un "*overflow*" del temporizador llamando a una función de interrupción por el microcontrolador, por lo que cambia el flujo de programa para atender la interrupción.

El lenguaje Arduino no dispone de funciones propias para configurar los temporizadores. Los registros internos del microcontrolador para configurar son: TCCR2A, TCCR2B, TNT2, OCR2A, OCR2B, TIMSK2, TIFR2, ASSR y GTCCR. En el registro TCCT2B se disponen los bits CS22,CS21 y CS20 (bit 2,1 y 0). Dichos bits son los encargados de configurar el "*prescaler*" del "*timer*" (ver especificaciones del microcontrolador).

Existen librerías adicionales para el manejo de los temporizadores de Arduino DUE, las cuales se pueden descargar de manera gratuita (enlace disponible en la bibliografía). La librería utilizada en las prácticas siguientes es "*Duetimer*", en la cual existen funciones en las que se configuran cada uno de los registros mencionados anteriormente.

## PRÁCTICA #15: TEMPORIZADOR/CONTADOR PARA EL MUESTREO PERIÓDICO DE UNA SEÑAL ANALÓGICA

**Objetivo:** Programar el temporizador para adquirir muestras de una señal analógica cada 5 ms por el ADC y enseguida enviarlas al DAC, considerando una señal senoidal o una señal de electrocardiografía (ECG) de un simulador de paciente.

### Material:

- 1 Osciloscopio.
- 1 Generador de funciones / Simulador ECG “*Teach Patient Cardio*”
- 1 Resistencia de 330 Ω.
- 1 Resistencia de 220 Ω.
- 1 led.
- 1 Diodo Zener 1N4372A (3 Volts a ½ Watt)
- Librería “*DueTimer*” (enlace en bibliografía)

### Actividades:

- Implementar el circuito mostrado en la Figura 2.23 para limitar la amplitud de la señal de entrada, así como para la protección de la tarjeta Arduino DUE. Además conectar un led al pin 13 según la Figura 2.1
- Implementar el programa sugerido en Arduino.
- Configurar el generador de funciones para proveer una señal positiva a una frecuencia de 1.2 Hz, con una forma de onda senoidal y amplitud no mayor a 3.3 Volts pico a pico para que la señal sea aplicada a la entrada analógica A5. Si se cuenta con el simulador ECG “*Teach Patient Cardio*”, configurarlo en modo ECG, 72 LPM, amplitud de 4 mV y sin ruido.
- Visualizar en el osciloscopio la señal en la salida (DAC1).
- Modificar el tiempo de adquisición de datos en el temporizador de Arduino DUE.

### Notas:

- Una vez descargada la librería para el temporizador, copiarla a la carpeta de instalación del programa Arduino en: C:\...\arduino-1.5.2\libraries.

- La amplitud de la señal muestreada estará entre 1/6 y 5/6 de la amplitud de la señal original (dato del fabricante del microcontrolador).

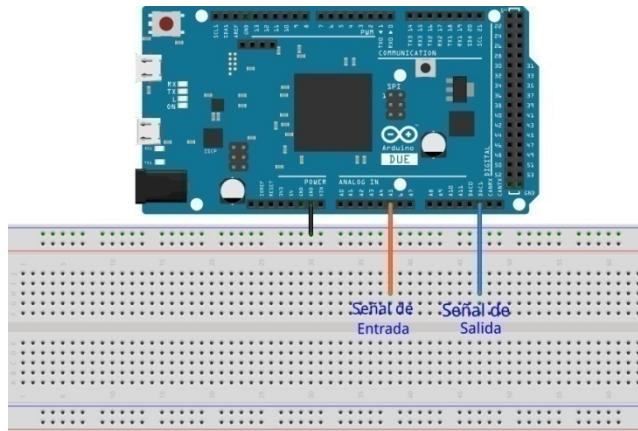


Figura 2.29. Diagrama esquemático practica #15.

## Descripción del programa

Se lee una entrada analógica (A5) con resolución de 12 bits cada 5 ms, para después enviarla al convertidor Digital Analógico (DAC1) (ver Figura 2.29). El led en el pin 13 siempre debe estar en nivel alto, y solo toma un nivel bajo cuando se ejecuta la interrupción.

## Programa

```
/*PROGRAMA QUE MUESTREA DE FORMA PERIÓDICA UNA SEÑAL ANALÓGICA POR MEDIO DE UN TEMPORIZADOR*/
#include <DueTimer.h> // Libreria para timer
float valor; // Definición de variables
int Led = 13;

void setup(){
    analogReadResolution(12);
    analogWriteResolution(12);
    pinMode(Led, OUTPUT); // configura salida
    Timer1.attachInterrupt(timer); // llamada de la función cuando hay "overflow"
    Timer1.start(5000); // Llama a la interrupción cada 5 ms
}

void loop(){
    digitalWrite(Led,HIGH); // Ejecución de tareas cuando no está la interrupción
}
```

```
void timer( ){                                // Función de interrupción
    valor=analogRead(A5);                  // Lee entrada analógica 5
    analogWrite(DAC1,valor)                // Envía datos al convertidor D-A
    digitalWrite(Led,LOW);
}
```

## Resultados

En la Figura 2.30, la señal de entrada considerando el simulador de ECG se visualiza en el canal 2 (color azul) y la salida en el canal 1.

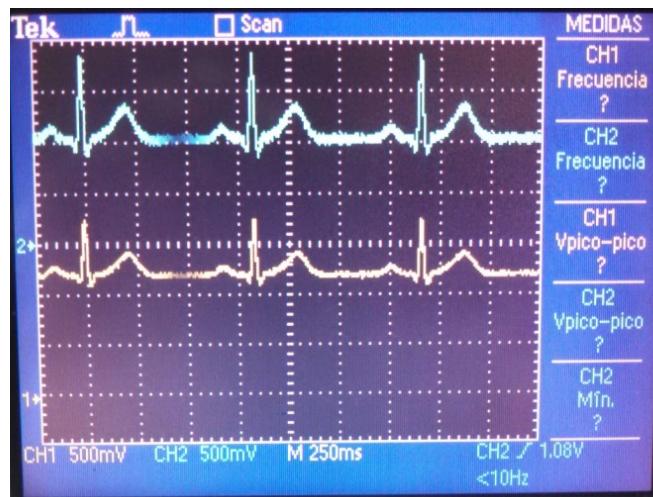


Figura 2.30.Señal de salida del DAC1(CH1) 72 LPM.

## PRÁCTICA #16: FILTRO FIR

**Objetivo:** Implementar un filtro digital FIR que promedie una ventana corrediza de 10 muestras de una entrada discreta  $x[n]$  capturada por el ADC en un instante  $n$ :

$$y[n] = \frac{1}{10} \sum_{i=0}^9 x[n-i] = \frac{1}{10} \{x[n] + x[n-1] + \dots + x[n-9]\} \quad (1.1)$$

y la salida  $y[n]$  enviarla al DAC.

### Material:

- 1 Osciloscopio.
- 1 Generador de funciones
- 1 Resistencia de  $330\ \Omega$ .
- 1 Diodo Zener 1N4372A (3 Volts a  $\frac{1}{2}$  Watt)

### Actividades:

- Implementar el circuito mostrado en la Figura 2.23 para limitar la amplitud de la señal de entrada, así como protección de la entrada analógica de Arduino DUE.
- Implementar el programa sugerido en Arduino con un periodo de muestreo de 0.5 ms (2 KHz). Configurar el generador de funciones para tener una señal positiva a una frecuencia de 200 Hz, con una forma de onda senoidal y amplitud no mayor a 3.3 Volts pico a pico (rango de 0 a 3.3 Volts) para que la señal sea aplicada a la entrada A5. Observar que el filtro FIR tiene ganancia unitaria a frecuencia cero y un comportamiento pasa-bajos.
- Visualizar en el osciloscopio la señal en la salida (DAC1).
- Enseguida incluir ruido aleatorio a la señal senoidal de entrada para observar a detalle el efecto del filtro FIR.

### Notas:

- La amplitud de la señal muestreada estará entre 1/6 y 5/6 de la amplitud de la señal original (dato del fabricante del microcontrolador).

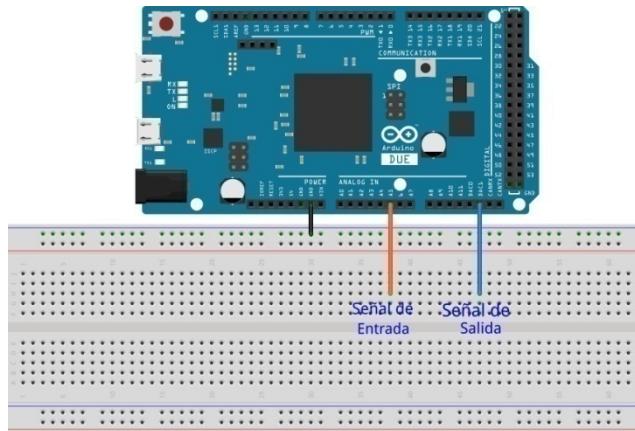


Figura 2.31. Diagrama esquemático practica #16.

## Descripción del programa

Se lee una entrada analógica (A5) con resolución de 12 bits cada 0.5 ms para después aplicarle el promediador de 10 muestras (pasa-bajos), y enviar su salida al convertidor Digital - Analógico (DAC1) (ver Figura 2.31).

## Programa

```
/*PROGRAMA PARA IMPLEMENTAR UN FILTRO FIR DE VENTANA CORREDIZA*/
#include <DueTimer.h> // Librería para timer
float y,prom,suma,m0,m1,m2, // Definición de variables
      m3,m4,m5,m6,m7,m8,m9;

void setup(){
  analogReadResolution(12); // Resolución de convertidores
  analogWriteResolution(12);
  Timer1.attachInterrupt(timer); // llamada de la función cuando hay "overflow"
  Timer1.start(500); // Llama a la interrupción cada 0.5 ms
  suma=0; // Condiciones iniciales en cero.
  m1=m2=m3=m4=m5=m6=m7=m8=m9=0;
}

void loop() // Ninguna tarea se ejecuta fuera de la interrupción
{

void timer(){ // Función de interrupción
  m0=analogRead(A5); // Lectura de x[n]
```

```
suma=(m0+m1+m2+m3+m4+m5+m6+m7+m8+m9);      // Sumatoria de las 10 muestras
prom=0.1*suma;                                     // Promedio
analogWrite(DAC1,prom);                           // Envía datos al DAC
m1=m0;                                            // x[n-1]=x[n]
m2=m1;                                            // x[n-2]=x[n-1]
m3=m2;                                            // x[n-3]=x[n-2]
m4=m3;                                            // x[n-4]=x[n-3]
m5=m4;                                            // x[n-5]=x[n-4]
m6=m5;                                            // x[n-6]=x[n-5]
m7=m6;                                            // x[n-7]=x[n-6]
m8=m7;                                            // x[n-8]=x[n-7]
m9=m8;                                            // x[n-9]=x[n-8]
}
```

## Resultados

En la Figura 2.32, la señal de entrada se visualiza en el canal 2 (color azul) y la salida en el canal 1, sin considerar ruido en la señal senoidal. A partir de esta figura se observa la atenuación de la señal de entrada, debida al comportamiento pasa-bajos del filtro FIR.

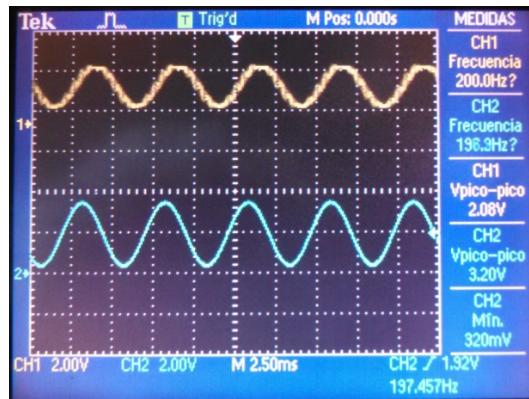


Figura 2.32. Señal de salida del DAC1(CH1).

## PRÁCTICA #17: FILTRO IIR

**Objetivo:** Implementar el filtro digital pasa-bajos IIR de la siguiente ecuación:

$$y[n] = \frac{1}{2}y[n-1] + \frac{1}{4}x[n] + \frac{1}{4}x[n-1] \quad (1.2)$$

donde  $x[n]$  representa la señal discreta de entrada y  $y[n]$  la salida para un instante de tiempo  $n$ .

### Material:

- 1 Osciloscopio.
- 1 Generador de funciones
- 1 Resistencia de 330 Ω.
- 1 Diodo Zener 1N4372A (3 Volts a ½ Watt)

### Actividades:

- Implementar el circuito mostrado en la Figura 2.23 para limitar la amplitud de la señal de entrada, así como protección de la entrada analógica de Arduino DUE.
- Implementar el programa en Arduino con un periodo de muestreo de 0.5 ms (2 KHz). Configurar el generador de funciones una señal positiva a una frecuencia de 1KHz, con una forma de onda senoidal y amplitud no mayor a 3.3 Volts pico a pico para que la señal sea aplicada a la entrada A5.
- Visualizar en el osciloscopio la señal en la salida (DAC1). Observar que el filtro IIR tiene ganancia nula a la frecuencia de Nyquist, es decir para 1 KHz; mientras que su ganancia a frecuencia 0 Hz es unitaria.
- Enseguida incluir ruido en la señal senoidal de entrada para observar a detalle el efecto del filtro IIR.

### Notas:

- La amplitud de la señal muestreada estará entre 1/6 y 5/6 de la amplitud de la señal original (dato del fabricante del microcontrolador).

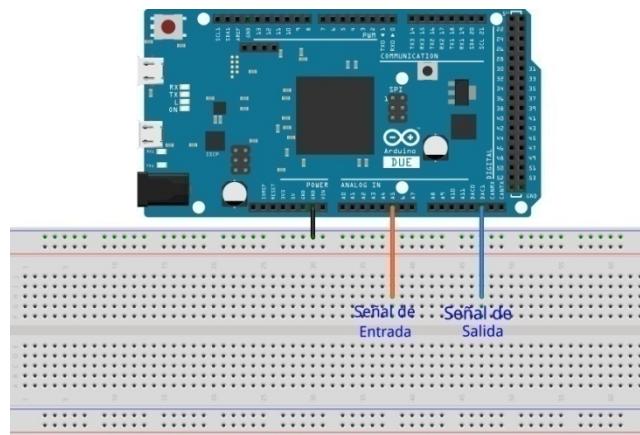


Figura 2.33. Diagrama esquemático practica #17.

## Descripción del programa

Se lee una entrada analógica (A5) con resolución de 12 bits cada 0.5 ms para después aplicarle el filtro pasabajos IIR, y enviar su salida al convertidor Digital - Analógico (DAC1) (ver Figura 2.33).

## Programa

```
/*PROGRAMA QUE IMPLEMENTA UN FILTRO IIR DE 1ER ORDEN*/
#include<DueTimer.h>           // Definición de variables
float x,y;                      // Entrada y salida actual
float xa,ya;                     // Entrada y salida anterior

void setup(){
    analogReadResolution(12);     // Resolución de convertidor
    analogWriteResolution(12);
    xa=0;                         // Condiciones iniciales
    ya=0;                         // Entrada y salida anterior
    x=analogRead(A5);             // Salida inicial
    y=0.25*x;
    analogWrite(DAC1, y);

    Timer1.attachInterrupt(timer); // llamada de la función cuando hay "overflow"
    Timer1.start(500);            // Interrupción cada 0.5ms
}

void loop(){ }

void timer(){
    xa=x;                         // Actualización de entrada y salida anterior
    ya=y;
```

```
x=analogRead(A5);  
y=(0.5*ya+0.25*x+0.25*xa); // Ecuación en diferencias del filtro IIR  
analogWrite(DAC1,y); // Envío de datos al DAC  
}
```

## Resultados

En las Figuras 2.34 y 2.35, la señal de entrada se visualiza en el canal 2 (color azul) y la salida en el canal 1 para las frecuencias de 100 Hz y 1 KHz. En estas figuras se observa la atenuación para la frecuencia de 100 Hz, y la ganancia nula a 1 KHz, como era previsto.

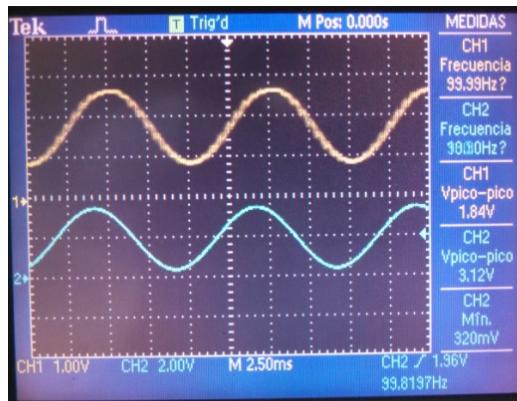


Figura 2.34.Señal de salida del DAC1(CH1) a 100Hz.

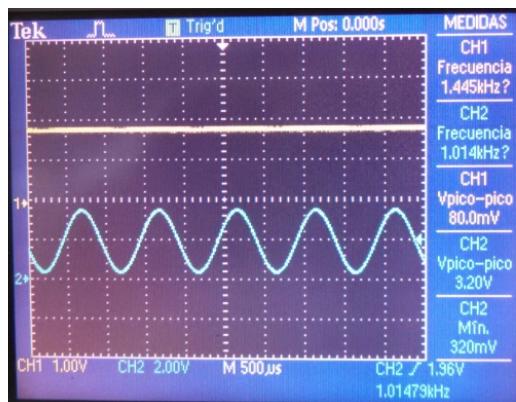
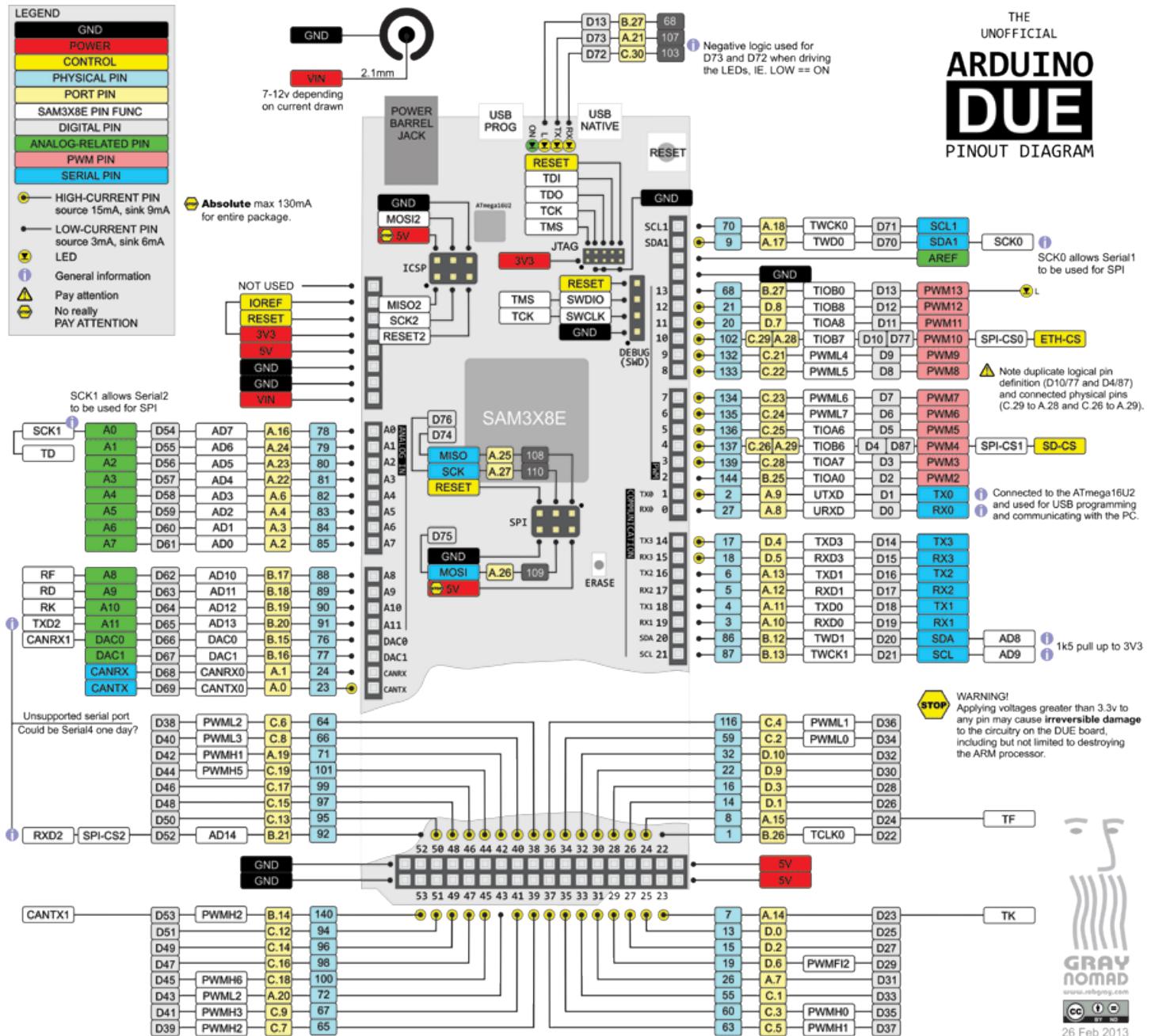


Figura 2.35.Señal de salida del DAC1(CH1) a 1KHz.

## V. REFERENCIAS

## “PINOUT” ARDUINO DUE



## VI. BIBLIOGRAFÍA

- [1] J. Purdum Jack. *Beginning C for Arduino*. 1<sup>a</sup> edición 2012. Editorial Apress
- [2] Torrente Artero Óscar. *Arduino Curso Práctico de Formación*. 1<sup>a</sup> Edicion 2013. Editorial RC Libros. Madrid, España.
- [3] Lajara Vizcaíno José Rafael/Pelegrí Sebastiá José. *Sistemas Integrados con Arduino*. 1<sup>a</sup> edición 2014. Editorial Marcombo. Barcelona, España.
- [4] Librería “Duetimer”. [Fecha de consulta: Agosto 2014]  
<https://github.com/ivanseidel/DueTimer>
- [5] Sitio oficial de “he instruments” Teach Patient Cardio. [Fecha de consulta: Agosto 2014]  
<http://www.heinstruments.com/>
- [6] Sensor de temperatura lm35. [Fecha de consulta: Agosto 2014]  
<http://www.ti.com/lit/ds/symlink/lm35.pdf>
- [7] Transistor 2n3904. [Fecha de consulta: Agosto 2014]  
<http://pdf.datasheetcatalog.net/datasheet/fairchild/2N3904.pdf>
- [8] LCD 16X2 crb16205. [Fecha de consulta: Agosto 2014]  
<http://www.dfrobot.com/image/data/fit0127/datasheet.pdf>