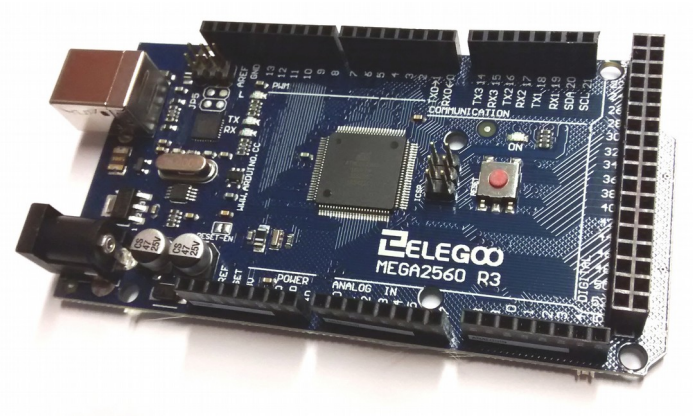
Práctica 5. E/S con Arduino



Introducción

Arduino es una empresa de software y hardware abiertos (bajo licencias LGPL y GPL), y también un proyecto y comunidad internacionales, que diseñan y fabrican placas de desarrollo de hardware con sensores y actuadores para interactuar con objetos del mundo real. Arduino se enfoca en acercar y facilitar el uso de la electrónica y programación de sistemas embebidos en proyectos multidisciplinares. Los diseños de las placas Arduino usan diversos microcontroladores y microprocesadores, generalmente microcontroladores Atmel AVR.

Los **AVR** son una familia de microcontroladores RISC del fabricante Atmel, hoy día propiedad de Microchip Technology. Dentro de la arquitectura de los AVR hay varias familias:

- ATxmega: procesadores muy potentes con 16 a 384 KB de memoria flash programable, capacidad de DMA, eventos, criptografía y un amplio conjunto de periféricos.
- ATmega: microcontroladores AVR grandes con 4 a 256 KB de memoria flash programable, repertorio de instrucciones extendido (multiplicación y direccionamiento de programas mayores) y un amplio conjunto de periféricos.
- ATtiny: pequeños microcontroladores AVR con 0,5 a 8 KB de memoria flash programable, y un limitado conjunto de periféricos.
- Modelos especiales, por ejemplo, para el control de los cargadores de baterías, pantallas LCD, control de los motores o iluminación.

Dado que el hardware es libre, existen placas y equipos de hardware compatibles con la tecnología Arduino de diversos fabricantes. En esta práctica usaremos un equipo muy completo del fabricante chino **Elegoo**: el equipo "*The Most Complete Starter Kit Mega 2560 Project*" [7], basado en el microcontrolador ATmega2560 [1].

También es posible simular los circuitos antes de crearlos físicamente usando **simuladores** como *Autodesk Tinkercad circuits* [5]. Aunque en este guión no cubrimos este aspecto, el estudiante es libre de probar los diseños en un simulador.

Objetivos

- Conocer el entorno de desarrollo de Arduino.
- Aprender a escribir programas sencillos de E/S para entorno Arduino que interactúen con hardware real.
- Montar pequeños circuitos de E/S analógica y digital.

IDE Arduino: entorno de desarrollo para interactuar y programar con placas Arduino

En la web de Arduino:

https://www.arduino.cc/ → LEARN (https://www.arduino.cc/en/Guide/HomePage)

encontramos dos alternativas para interactuar y programar las placas Arduino:

- Alternativa 1. Code online on the Arduino Web Editor (Programar en línea en el editor web de Arduino)
- Alternativa 2. Install the Arduino Desktop IDE (Instalar el IDE de escritorio de Arduino)
  - En la ETSIIT tenemos instalado el IDE en la última distribución de Ubuntu. Basta entrar en Ubuntu 18.04, y buscar Arduino entre los programas, o bien escribir “arduino” en el terminal.
  - En nuestro ordenador, seguiremos los pasos de instalación, eligiendo primero el sistema operativo entre:
    - Windows
    - Mac OS X
    - Linux
    - Portable IDE (Windows and Linux)

Trabajaremos en principio con la alternativa 2. Si el estudiante lo desea, puede opcionalmente optar por la alternativa 1 en su propio ordenador.

Describimos aquí la instalación del Ide de escritorio de Arduino (alternativa 2) en Linux 64 bits: https://www.arduino.cc/en/Guide/Linux

Paso 1. Descargamos la última versión (Arduino 1.8.7 o posterior) desde la página de descarga https://www.arduino.cc/en/Main/Software, seleccionando Linux 64 bits:

https://www.arduino.cc/download_handler.php?f=/arduino-1.8.7-linux64.tar.xz

Pulsaremos en “Just download” o en “Contribute & Download”

Pulsaremos en “Guardar archivo”. El archivo arduino-1.8.7-linux64.tar.xz se guardará en nuestra carpeta de Descargas. Podemos mover el archivo a nuestro home:

```
mv ~/Descargas/arduino-1.8.7-linux64.tar.xz ~
```

Paso 2. En nuestro home podemos hacer doble clic en el archivo **arduino-1.8.7-linux64.tar.xz**

o bien desde el terminal

```
cd
tar -xvf arduino-1.8.7-linux64.tar.xz
```

Comprobamos que se ha creado el directorio arduino-1.8.7

```
ls -l arduino-1.8.7
```

Paso 3. Ya podemos instalar el IDE:

```
cd arduino-1.8.7
./install.sh
```

Paso 4. Puede ocurrir que cuando carguemos un programa (conocido como sketch en Arduino), tras seleccionar nuestra placa y puerto serie, aparezca un error "*Error opening serial port...*", en cuyo caso debemos configurar los permisos del puerto serie. Para ello, escribimos la siguiente orden en el terminal:

```
ls -l /dev/ttyACM*
```

Si aparece el mensaje

```
ls: no se puede acceder a '/dev/ttyACM*': No existe el archivo o el directorio
```

probablemente no tengamos conectada la tarjeta Arduino al puerto USB. Es necesario conectar la tarjeta Arduino, y obtendremos:

```
crw-rw---- 1 root dialout 166, 0 nov 25 15:48 /dev/ttyACM0
```

El "0" al final de ACM puede ser un número diferente, o se pueden devolver múltiples entradas. El dato que necesitamos es "dialout" (el propietario del grupo del archivo). Ahora solo necesitamos agregar nuestro usuario al grupo:

```
sudo usermod -a -G dialout <nombre de usuario>
```

donde <nombre de usuario> es nuestro nombre de usuario de Linux. La orden *usermod* modifica los archivos de la cuenta del usuario especificado para reflejar los cambios que se especifiquen. En este caso **-a** indica que se agregue al usuario a los grupos mencionados por la opción **-G** sin eliminarlo de otros grupos.

Podemos comprobar a qué grupos pertenecemos ejecutando:

```
groups <nombre de usuario>
```

Tras añadir nuestro usuario al grupo dialout, deberemos cerrar sesión e iniciar sesión de nuevo para que este cambio surta efecto. Podemos comprobar el significado de cada grupo viendo el archivo **/usr/share/doc/base-passwd/users-and-groups.html** desde el navegador, escribiendo la siguiente ruta en la barra de direcciones del navegador:

file:///usr/share/doc/base-passwd/users-and-groups.html

Como podemos leer, pertenecer al grupo **dialout** permite el acceso a los puertos serie.

Después de este procedimiento, deberíamos poder cargar los programas en nuestra placa desde el IDE de Arduino.

Kit de desarrollo Elegoo

En esta práctica usaremos el kit “The Most Complete Starter Kit Mega 2560 Project” de la empresa Elegoo. El manual de este kit está disponible en SWAD, en el archivo

Elegoo The Most Complete Starter Kit for MEGA V1.0.2018.11.14.zip

y también podemos descargar la última versión desde la web de Elegoo:

https://www.elegoo.com/download/

pulsando sobre el kit “Elegoo Mega 2560 The Most Complete Starter Kit”.

https://www.elegoo.com/tutorial/Elegoo%20The%20Most%20Complete%20Starter%20Kit%20for%20MEGA%20V1.0.2018.11.22.zip

Guardamos el archivo zip en nuestra ubicación preferida, y lo descomprimimos. Se habrá creado la siguiente carpeta:

‘Elegoo The Most Complete Starter Kit for MEGA V1.0.2018.11.22’

Dentro de ella encontraremos los siguientes archivos:

```
$ ls -l ‘Elegoo The Most Complete Starter Kit for MEGA V1.0.2018.11.22’
drwxrwxr-x 3 usuario usuario    4096 nov 22 16:45  Datasheet
drwxrwxr-x 4 usuario usuario    4096 nov 22 16:45  Deutsch
drwxrwxr-x 5 usuario usuario    4096 nov 22 16:44  English
drwxrwxr-x 4 usuario usuario    4096 nov 13 09:37  Espanol
drwxrwxr-x 4 usuario usuario    4096 nov 13 09:37  Français
drwxrwxr-x 4 usuario usuario    4096 nov 13 09:37  Italiana
drwxrwxr-x 4 usuario usuario    4096 nov 13 09:37  Pºσºπºπº▄º┌º█
-rw-rw-r-- 1 usuario usuario    1094 nov  2  2017 'Read Me First.txt'
-rw-rw-r-- 1 usuario usuario 1359196 ago 29  2016 'UNO R3, MEGA, NANO DRIVER FAQ.pdf'
drwxrwxr-x 4 usuario usuario    4096 nov 13 09:37  ╚┌▒┘ ┌Z
```

Figura 1. Kit de desarrollo de Elegoo Mega 2560 The Most Complete Starter Kit.

Veremos preferiblemente el contenido de la carpeta English, pues la traducción a Español no es buena.

```
$ cd 'Elegoo The Most Complete Starter Kit for MEGA V1.0.2018.11.22'/
$ cd English/
$ ls -l
drwxrwxr-x 32 usuario usuario     4096 nov 13 09:37  code
drwxrwxr-x  3 usuario usuario     4096 nov 13 09:37  Datasheet
-rw-rw-r--  1 usuario usuario   898651 abr 13  2018 'ELegoo UNO R3 , MEGA, DRIVER FAQ.pdf.pdf'
-rw-rw-r--  1 usuario usuario   591191 ago  1 18:50 'INSTALL ARDUINO ON LINUX.pdf'
drwxrwxr-x  2 usuario usuario     4096 nov 13 09:37  Libraries
-rw-rw-r--  1 usuario usuario     1202 nov 20  2017 'Read Me First.txt'
-rw-rw-r--  1 usuario usuario 22489583 nov 22 16:44 'The Most Complete Starter Kit for MEGA
V1.0.18.11.22.pdf'
```

En el archivo **`'The Most Complete Starter Kit for MEGA V1.0.18.11.22.pdf'`** encontraremos 34 lecciones (de la 0 a la 33) para usar todos los componentes del kit. La lección 0 nos enseña a instalar el IDE de Arduino, la lección 1 nos muestra cómo instalar bibliotecas adicionales para Arduino y cómo manejar el Monitor Serie de Arduino. El resto de lecciones nos muestran distintos programas de ejemplo para manejar todos los componentes incluidos en el kit.

El principal componente del ki es la placa Elegoo Mega2560 R3, que contiene un microcontrolador Atmel ATmega2560 [1] (Figura 2).

Figura 2. Detalle de la placa Elegoo Mega2560 R3 con el microcontrolador ATmega2560 [1].

Configuración de nuestra placa y conexión

Conectaremos la placa Elegoo a través del cable USB a un puerto USB de nuestro ordenador, y a continuación abriremos el IDE de Arduino. Cuando abrimos el IDE de Arduino por primera vez, tenemos que configurar qué tipo de placa tenemos y a qué puerto está conectada. Para configurar nuestra placa, vamos al menú ***Herramientas > Placa*** y seleccionamos ***Arduino/Genuino Mega or Mega 2560***. Para configurar el puerto serie, vamos al menú ***Herramientas > Puerto*** y seleccionamos ***Puertos serie: /dev/ttyACM0***. El IDE de Arduino mostrará nuestra configuración actual de placa en la parte inferior derecha de la ventana (Figura 4).

Primer programa: hacer parpadear el led de la placa

En la lección 2 construiremos nuestro primer programa, que consistirá en hacer parpadear intermitentemente el led integrado en la placa Elegoo.

Componentes necesarios:

- *(1) x Elegoo Mega 2560 R3*

La placa Elegoo MEGA 2560 R3 tiene filas de conectores en sus lados que se utilizan para conectarse a varios dispositivos electrónicos y a placas “shield”' que amplían su capacidad. También tiene un led integrado que podemos controlar desde nuestros programas. A menudo se lo denomina led 'L', ya que así está etiquetado en la placa (Figura 3).

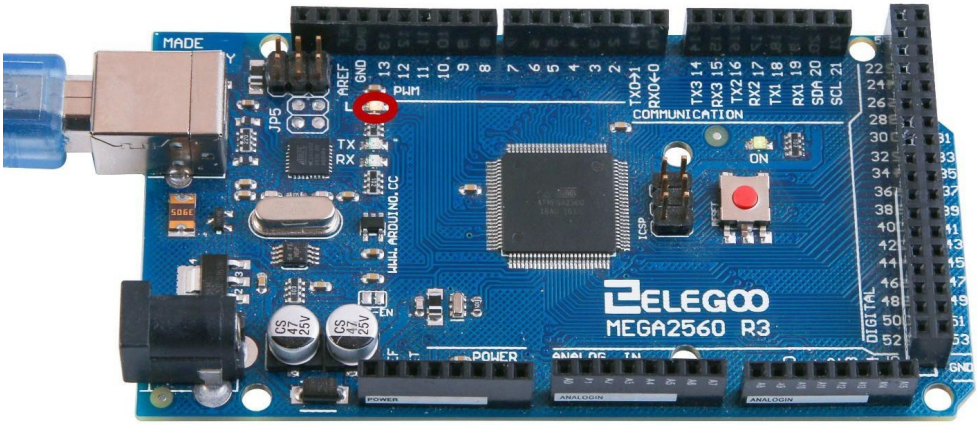


Figura 3. Situación del led L en la placa Elegoo Mega2560 R3 (marcado en rojo en la imagen). [7]

Es posible que el led L de la placa MEGA 2560 R3 ya parpadee cuando lo conectemos al USB. Esto se debe a que las placas generalmente se distribuyen con el programa 'Blink' preinstalado. En este primer programa, reprogramaremos la placa MEGA 2560 R3 con nuestro propio programa Blink y luego cambiaremos la velocidad a la que parpadea.

En lenguaje de Arduino

Abriremos el IDE de Arduino, que incluye una gran colección de programas de ejemplo que podemos cargar y usar. Entre ellos se incluye un programa de ejemplo para hacer que el led L parpadee. Cargaremos el programa 'Blink' que encontraremos en el menú ***Archivo*** > ***Ejemplos*** > ***01.Basics*** > ***Blink***.

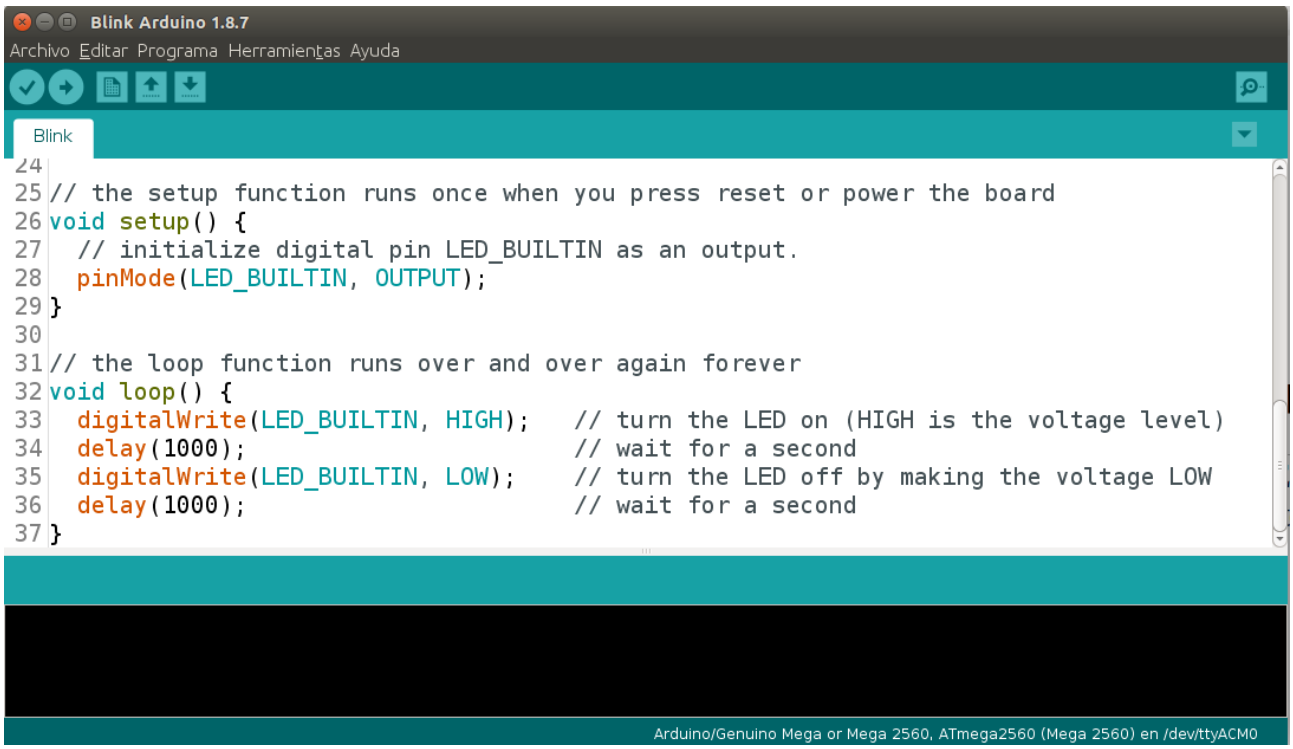


Figura 4. Programa Blink en el IDE de Arduino.

Los programas de ejemplo incluidos con el IDE de Arduino son de solo lectura. Como vamos a cambiar este programa, lo primero que tenemos que hacer es guardar nuestra propia copia. En el menú Archivo en el IDE de Arduino, seleccionamos ***Guardar Como...*** y guardamos el programa con el nombre **`my_blink`** o similar en nuestra carpeta de programas. Si más adelante queremos volver a encontrarlo, podemos abrirlo utilizando la opción de menú ***Archivo*** > ***Proyecto*** o ***Archivo*** > ***Abrir reciente***.

Para copiar el programa en la placa, pulsaremos en el botón ***Subir Usando Programador***. Es el segundo botón de la izquierda en la barra de herramientas, y representa una flecha apuntando hacia la derecha (Figura 4). Si observamos el área de estado del IDE, veremos una barra de progreso y una serie de mensajes. Al principio, dirá ***Compilando programa…*** Después el estado cambiará a ***Subiendo…*** Los led de la placa parpadearán a medida que se transfiere el programa. Por último aparecerá ***Subido***.

En la consola aparecerá un mensaje que nos dice que el programa (sketch en la nomenclatura de Arduino) está utilizando 1460 bytes bytes de los 253 952 bytes disponibles.

Una gran parte del programa se compone de comentarios. Todo lo que haya entre /* y */ es un comentario de bloque; se usa por ejemplo para explicar el propósito del programa. Los comentarios de una sola línea comienzan con // y todo lo que haya hasta el final de esa línea se considera un comentario.

Lo primero que encontramos es la función **`void setup()`**. Como dice el comentario, esta se ejecuta cuando se pulsa el botón de *reset* (botón rojo a la derecha de la Figura 3). También se ejecuta cada vez que la placa se reinicia por cualquier razón, como ocurre cuando se alimenta la placa tras estar desconectada, o después de que se suba un programa.

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}
```

Cada programa de Arduino debe tener una función **`setup`**. En este caso, solo contiene una llamada a pinMode, que, como dice el comentario, indica a la placa Arduino que vamos a utilizar el pin **`LED_BUILTIN`** como salida.

También es obligatorio que un programa tenga una función **`loop`**. A diferencia de la función **`setup`**, que solo se ejecuta una vez después de un reinicio, la función **`loop`** comenzará a ejecutarse de nuevo una vez que haya terminado de ejecutarse.

```
// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

Dentro de la función **`loop`**, primero se enciende el pin LED_BUILTIN, luego hay una espera de 1000 milisegundos (1 segundo), luego se apaga el led y de nuevo se hace otra pausa de 1 segundo.

Vamos a hacer que nuestro led parpadee más rápido. La clave es cambiar el parámetro de la función **`delay`**. Este período de demora es en milisegundos, por lo que si desea que el led parpadee dos

veces más rápido, tendremos que cambiar el valor de 1000 a 500. Subimos el programa nuevamente y veremos que el led comienza a parpadear más rápidamente.

Podemos consultar los detalles del lenguaje y de las funciones en el manual de fundamentos https://www.arduino.cc/en/Tutorial/Foundations [3] y en el manual de referencia del lenguaje: https://www.arduino.cc/reference/en/ [4].

En ensamblador

Vamos a ver el mismo ejemplo del led parpadeante, pero en esta ocasión programado en ensamblador usando ensamblador en línea [9].

```
void my_delay(uint16_t ms);

#define CPU_FREQUENCY 16000000UL
uint16_t delay_count = CPU_FREQUENCY / 4000;

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  // pinMode(LED_BUILTIN, OUTPUT);
  asm volatile (
    "sbi %0, %1 \n"
    : : "I" (_SFR_IO_ADDR(DDRB)), "I" (DDB7)
  );
}

// the loop function runs over and over again forever
void loop() {
  // digitalWrite(LED_BUILTIN, HIGH);// turn the LED on (HIGH is the voltage level)
  // digitalWrite(13         , HIGH);// turn the LED on (HIGH is the voltage level)
  asm volatile (
    "sbi %0, %1 \n"
    : : "I" (_SFR_IO_ADDR(PORTB)), "I" (PORTB7)
  );

  my_delay(10);   // wait for 10 ms

  // digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
  // digitalWrite(13         , LOW); // turn the LED off by making the voltage LOW
  asm volatile (
    "cbi %0, %1 \n"
    : : "I" (_SFR_IO_ADDR(PORTB)),"I" (PORTB7)
  );

  my_delay(990);  // wait for 990 ms
}

void my_delay(uint16_t ms) {
  uint16_t cnt;
  asm volatile (
                      "\n"
    "L_dl1%=:"        "\n\t"
      "mov %A0, %A2"  "\n\t"
      "mov %B0, %B2"  "\n"

    "L_dl2%=:"        "\n\t"
      "sbiw %A0, 1"   "\n\t"
      "brne L_dl2%="  "\n\t"

      "sbiw %1,  1"   "\n\t"
      "brne L_dl1%="  "\n\t"

    : "=&w" (cnt)
    : "r" (ms), "r" (delay_count)
  );
}
```

La función

```
pinMode(LED_BUILTIN, OUTPUT);
```

se sustituye por:

```
  asm volatile (
    "sbi %0, %1 \n"
    : : "I" (_SFR_IO_ADDR(DDRB)), "I" (DDB7)
  );
```

Por un lado, en el archivo …**/arduino-1.8.7/hardware/arduino/avr/variants/mega/pins_arduino.h** encontramos la definición de la constante **LED_BUILTIN**:

```
#define LED_BUILTIN 13
```

y en el archivo de cabecera .../arduino1.8.7/hardware/arduino/avr/cores/arduino/Arduino.h encontramos la definición de las constantes **OUTPUT** y **HIGH**:

```
#define OUTPUT 0x1
#define HIGH 0x1
```

Para acceder al pin 13 de la placa, tenemos que ver a qué pin del microcontrolador ATMega2560 está conectado. En la Figura 5 podemos ver que está conectado al bit 7 del puerto B [2].

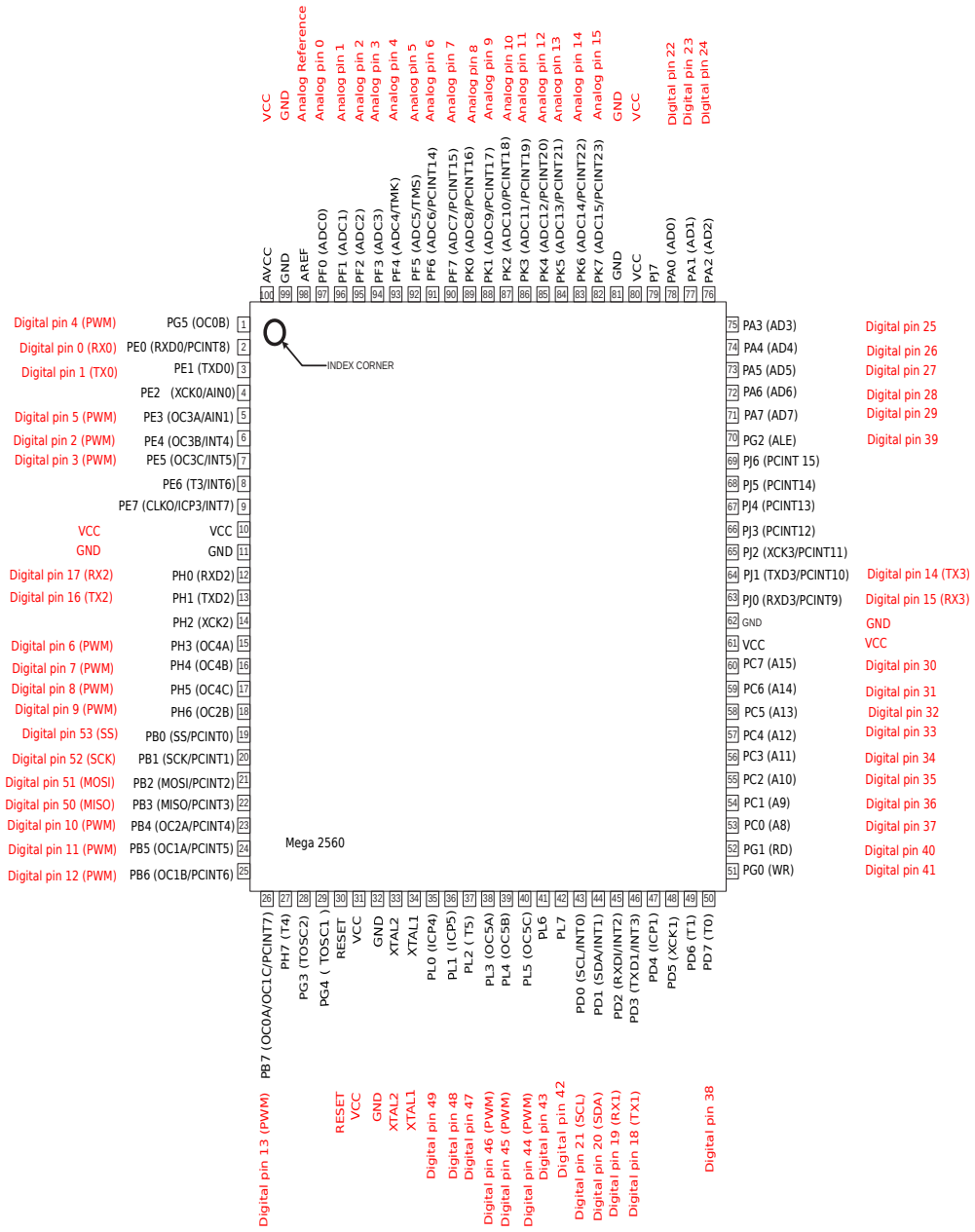


Figura 5. Diagrama de conexión de pines entre la placa (rojo) y el microcontrolador AtMega2560 (negro) [2].

La instrucción máquina **SBI P,b** (*Set Bit in I/O Register*) pone a 1 el bit **b** del registro **P** [1]. Cada uno de los 8 bits del registro *Port B Data Direction Register* (DDRB) indica si el bit correspondiente del puerto B está configurado como entrada (0) o salida (1). En este caso queremos indicar que el bit 7 del registro B tiene que estar configurado como salida. Para ello ponemos a 1 el bit 7 (DDB7) del registro **DDRB**. La macro **_SFR_IO_ADDR**, donde SFR significa *Special Function Register*, calcula la dirección de E/S del registro **DDRB** necesaria para la instrucción SBI, que es 0x04, a partir de su dirección mapeada en memoria, que es 0x24. Efectivamente, en ciertos archivos de cabecera **…/arduino-1.8.7/hardware/tools/avr/avr/include/avr/*.h** podemos encontrar las siguientes definiciones de constantes y macros:

```
#define DDRB _SFR_IO8(0x04)
#define PORTB _SFR_IO8(0x05)
#define __SFR_OFFSET 0x20
#define _SFR_IO8(io_add) ((io_addr) + __SFR_OFFSET)
#define _SFR_IO_ADDR(sfr) ((sfr) - __SFR_OFFSET)
```

de las cuales obtenemos que:

DDRB = **0x04+0x20** = **0x24**
PORTB = **0x05+0x20** = **0x25**
SFR_IO_ADDR(DDRB) = **0x24-0x20** = **0x04**
SFR_IO_ADDR(PORTB) = **0x25-0x20** = **0x05**

La misma explicación se aplica para

```
digitalWrite(LED_BUILTIN, HIGH);
```

que se sustituye por:

```
  asm volatile (
    "sbi %0, %1 \n"
    : : "I" (_SFR_IO_ADDR(PORTB)), "I" (PORTB7)
  );
```

Por su parte, para poner a nivel bajo el led:

```
digitalWrite(LED_BUILTIN, LOW);
```

hay que usar la instrucción máquina **CBI P,b** (*Clear Bit in I/O Register*), que pone a 0 el bit **b** del registro **P** [1]:

```
  asm volatile (
    "cbi %0, %1 \n"
    : : "I" (_SFR_IO_ADDR(PORTB)),"I" (PORTB7)
  );
```

En todos los casos anteriores se aplica la restricción de entrada **"I"**, que indica una constante positiva de 6 bits (0 a 63) [9, 11, 12].

Veamos la función **my_delay**:

```
#define CPU_FREQUENCY 16000000UL
uint16_t delay_count = CPU_FREQUENCY / 4000;

void my_delay(uint16_t ms) {
  uint16_t cnt;
  asm volatile (
                      "\n"
    "L_dl1%=:"        "\n\t"
      "mov %A0, %A2"  "\n\t"
```

```
      "mov %B0, %B2"  "\n"

    "L_dl2%=:"        "\n\t"
      "sbiw %A0, 1"   "\n\t"
      "brne L_dl2%="  "\n\t"

      "sbiw %1,  1"   "\n\t"
      "brne L_dl1%="  "\n\t"

    : "=&w" (cnt)
    : "r" (ms), "r" (delay_count)
  );
}
```

Esta función contiene dos bucles anidados. El bucle exterior itera tantas veces como indique la variable **`ms`**. El interior itera tantas veces como indique la variable **`delay_count`**. **`delay_count`** está calculada como la frecuencia de reloj *f* dividida por 4000, es decir, *f* / 4000. Cada iteración del bucle interior tarda 4 ciclos de reloj, ya que la instrucción **`sbiw`** tarda 2 ciclos y la instrucción **`brne`** otros 2 ciclos. Cada iteración del bucle exterior supone por tanto *f*/4000 iteraciones * 4 ciclos/iteración = 4*f*/4000 ciclos = *f*/1000 ciclos. Como cada ciclo tarda 1/*f* segundos, cada iteración del bucle exterior tardará (*f*/1000)/*f* segundos = 1 milisegundo. La función supone por tanto una espera de **`ms`** milisegundos.

El procesador ATmega2560 tiene 32 registros de propósito general de 8 bits. Ocho de los 32 registros pueden usarse como registros punteros de 16 bits: (r25:r24) y los registros X (r27:r26), Y (r29:r28) y Z (r31:r30) [1].

La restricción **`"r"`** significa cualquier registro, de r0 a r31, es decir, que tanto **`ms`** como **`delay_count`** se copian en registros de 16 bits. La restricción **`"=&w"`** significa cualquier pareja de registros especiales (r24, r26, r28, r30), y el **`&`** especifica que esa pareja de registros debe usarse sólo para salida y no debe usarse para entrada.

Si un operando no casa con un único registro, el compilador automáticamente asignará suficientes registros para almacenar el operando completo. En el código ensamblador usaremos %A0 para referirnos al byte bajo del primer operando y %A1 para el byte bajo del segundo operando, el siguiente byte del primer operando será %B0, el siguiente byte del segundo operando será %B1, y así sucesivamente. Por tanto, el código:

```
      "mov %A0, %A2"  "\n\t"
      "mov %B0, %B2"  "\n"
```

copia el byte bajo del operando 2 (**`delay_count`**) en el byte bajo del operando 0 (**`cnt`**), y luego el byte alto del operando 2 (**`delay_count`**) en el byte alto del operando 0 (**`cnt`**).

La instrucción **`SBIW Rdl,K`** (*Subtract Immediate from Word*) resta la constante **`K`** del registro de 16 bits **`Rdh:Rdl`**, es decir, hace Rdh:Rdl ← Rdh:Rdl – K.

La instrucción **`BRNE k`** (*Branch if Not Equal*) salta si no igual/no cero. El patrón **`%=`** en las etiquetas significa que se reemplazará dicho patrón por un número único en cada sentencia en ensamblador.

Escriba y pruebe esta versión del programa de parpadeo del led escrita en ensamblador.

Segundo programa: zumbador pasivo

Montaremos el circuito del la Lección 7 (zumbador pasivo, páginas 68 a 71) del tutorial del kit Elegoo [7].

Componentes necesarios:

- *(1) x Elegoo Mega 2560 R3*

- *(1) x zumbador pasivo*
- *(2) x cables hembra-macho*

El código se encuentra en la carpeta **`'Elegoo The Most Complete Starter Kit for MEGA V1.0.2018.11.22'/English/code/'Lesson 7 Passive Buzzer/passive_buzzer/'`**. Lo abriremos con ***Archivo*** > ***Abrir***. Si el estudiante lo desea, puede cambiar la melodía.

```
//www.elegoo.com
//2016.12.08

#include "pitches.h"

// notes in the melody:
int melody[] = {
  NOTE_C5, NOTE_D5, NOTE_E5, NOTE_F5, NOTE_G5, NOTE_A5, NOTE_B5, NOTE_C6};
int duration = 500;  // 500 miliseconds

void setup() {

}

void loop() {
  for (int thisNote = 0; thisNote < 8; thisNote++) {
    // pin8 output the voice, every scale is 0.5 sencond
    tone(8, melody[thisNote], duration);

    // Output the voice after several minutes
    delay(1000);
  }

  // restart after two seconds
  delay(2000);
}
```

La entrega de esta parte consistirá en un PDF con el código final, un dibujo esquemático y al menos una fotografía del circuito montado, y un breve vídeo del programa funcionando.

Tercer programa: theremín de luz

Un theremín es un instrumento musical electrónico que se controla sin necesidad de contacto físico del intérprete con el instrumento. Su nombre deriva del nombre de su inventor ruso Léon Theremin, que lo desarrolló en 1920. El instrumento está formado por dos antenas metálicas que detectan la posición relativa de las manos del intérprete respecto a las antenas para controlar la frecuencia con una mano y la amplitud (volumen) con la otra. Las señales eléctricas del theremin se amplifican y se envían a un altavoz.

En este tercer programa vamos a construir un theremín muy básico basado en un sensor de luz (en lugar de antenas) y un zumbador pasivo. Para ello nos inspiraremos en los ejemplos de las lecciones 7 (páginas 68 a 71) y 26 (célula fotoeléctrica, páginas 175 a 179) . Montaremos el circuito de la lección 7 y la parte de la célula fotoeléctrica del circuito de la lección 26, incluyendo la resistencia que acompaña a la célula fotoeléctrica, pero sin incluir en principio los ledes.

Componentes necesarios:

- *(1) x Elegoo Mega 2560 R3*
- *(1) x zumbador pasivo*
- *(2) x cables hembra-macho*

- *(1) x placa de prototipado de 830 contactos*
- *(8) x ledes (opcionales)*
- *(8) x resistencias de 220 Ω (opcionales)*
- *(1) x resistencia de 1 kΩ*
- *(1) x 74hc595 IC (opcional)*
- *(1) x foto resistencia (célula fotoeléctrica)*
- *(16) x cables macho-macho (algunos opcionales)*

El código del theremín de luz se encuentra en ***Archivo*** > ***Ejemplos*** > ***10.StarterKit_BasicKit*** > ***p06_LightTheremin***.

```
/*
  Arduino Starter Kit example
  Project 6 - Light Theremin

  This sketch is written to accompany Project 6 in the Arduino Starter Kit

  Parts required:
  - photoresistor
  - 10 kilohm resistor
  - piezo

  created 13 Sep 2012
  by Scott Fitzgerald

  http://www.arduino.cc/starterKit

  This example code is part of the public domain.
*/

// variable to hold sensor value
int sensorValue;
// variable to calibrate low value
int sensorLow = 1023;
// variable to calibrate high value
int sensorHigh = 0;
// LED pin
const int ledPin = 13;

void setup() {
  // Make the LED pin an output and turn it on
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);

  // calibrate for the first five seconds after program runs
  while (millis() < 5000) {
    // record the maximum sensor value
    sensorValue = analogRead(A0);
    if (sensorValue > sensorHigh) {
      sensorHigh = sensorValue;
    }
    // record the minimum sensor value
    if (sensorValue < sensorLow) {
      sensorLow = sensorValue;
    }
  }
  // turn the LED off, signaling the end of the calibration period
  digitalWrite(ledPin, LOW);
}

void loop() {
  //read the input from A0 and store it in a variable
```

```
  sensorValue = analogRead(A0);

  // map the sensor values to a wide range of pitches
  int pitch = map(sensorValue, sensorLow, sensorHigh, 50, 4000);

  // play the tone for 20 ms on pin 8
  tone(8, pitch, 20);

  // wait for a moment
  delay(10);
}
```

Para obtener ayuda sobre las funciones, podemos mirar la referencia del lenguaje [4].

Opcionalmente, si dentro de la sesión de prácticas sobra tiempo, sugerimos como mejora montar la parte de los ledes del circuito de la lección 26, y la parte del programa que los controla. El código de la lección 26 se encuentra en la carpeta **`'Elegoo The Most Complete Starter Kit for MEGA V1.0.2018.11.22'/English/code/'Lesson 26 Photocell/Photocell/'`** y podemos abrirlo con ***Archivo*** > ***Abrir***.

La entrega de esta parte consistirá en un PDF con el código final, un dibujo esquemático y al menos una fotografía del circuito montado, y un breve vídeo con el programa funcionando.

Segunda sesión de prácticas

En caso de que el estudiante quisiera montar la parte de los ledes del theremin de luz y no le hubiera dado tiempo en la primera sesión, puede hacerlo opcionalmente en esta sesión, o bien puede montar otro de los programas de ejemplo del kit Elegoo [7]. También es posible inspirarnos en otras fuentes de programas para Arduino, por ejemplo [8].

La entrega de esta parte consistirá en un PDF con el código final, un dibujo esquemático y al menos una fotografía del circuito montado, y un breve vídeo con el programa funcionando.

Referencias

1. Atmel, “Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V datasheet”. http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_Summary.pdf (breve) http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf (completa)
2. Arduino, “ATmega2560-Arduino Pin Mapping” https://www.arduino.cc/en/Hacking/PinMapping2560
3. Arduino, “Foundations” https://www.arduino.cc/en/Tutorial/Foundations
4. Arduino, “Languaje Reference”. https://www.arduino.cc/reference/en/
5. Autodesk Tinkercad. “Tinkercad circuits” https://www.tinkercad.com/circuits
6. Elegoo, “Download”. https://www.elegoo.com/download/
7. Elegoo, “The Most Complete Starter Kit Tutorial For Mega 2560, V1.0.18.11.22” https://www.elegoo.com/tutorial/Elegoo%20The%20Most%20Complete%20Starter%20Kit%20for%20MEGA%20V1.0.2018.11.22.zip
8. Grupo Sabika. “Ejercicios de Arduino resueltos”

https://aptandalucia.files.wordpress.com/2014/05/ejercicios_de_arduino_resueltos.pdf
9. Jim Eli, “Arduino Inline Assembly Tutorial”
https://ucexperiment.wordpress.com/2016/03/04/arduino-inline-assembly-tutorial-1/
10. Jim Eli, “GCC Inline Assembler Cookbook”
https://ucexperiment.wordpress.com/2013/06/03/gcc-inline-assembler-cookbook/
11. NonGNU, “AVR LibC”
https://www.nongnu.org/avr-libc/
http://savannah.nongnu.org/download/avr-libc/avr-libc-user-manual-2.0.0.pdf.bz2
12. NonGNU, “GCC-AVR Inline Assembler Cookbook”
https://www.nongnu.org/avr-libc/user-manual/inline_asm.html
https://www.microchip.com/webdoc/AVRLibcReferenceManual/inline_asm.html