CAPÍTULO 9

Entrada/Salida: ejercicios prácticos con Arduino Due

Índice

9.1. El entorno Arduino

Arduino de Ivrea (955–1015) fue Rey de Italia entre 1002 y 1014. Massimo Banzi y un grupo de docentes del Interaction Design Institute en Ivrea, en Italia, desarrollaron una plataforma de hardware libre basada en un microcontrolador y un entorno de desarrollo diseñados para facilitar la realización de proyectos de electrónica. Banzi y su grupo se reunían habitualmente en el Bar del Rey Arduino, en la localidad de Ivrea, de ahí el nombre del sistema.

Arduino está compuesto por una plataforma de hardware libre y un entorno de desarrollo. A grandes rasgos, esto significa que el diseño está

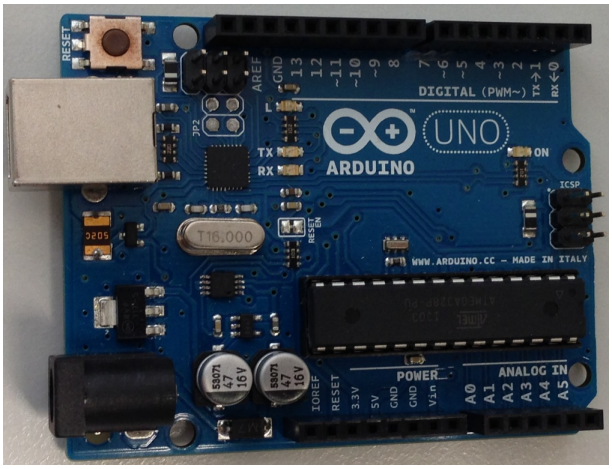

Figura 9.1: Tarjeta Arduino Uno

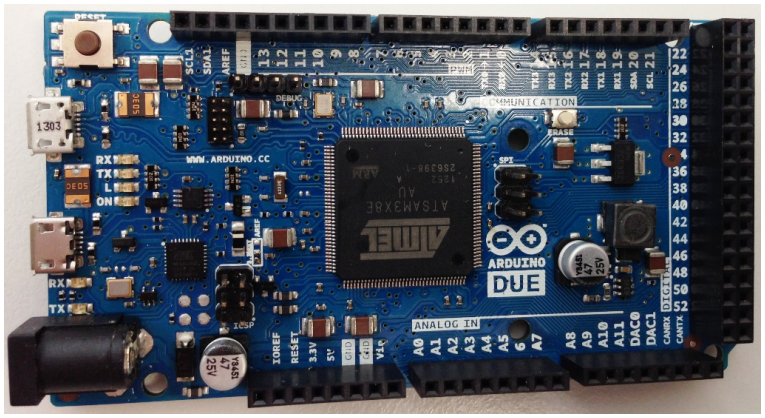

Figura 9.2: Tarjeta Arduino Due

a disposición de quien lo quiera emplear y modificar, dentro de unos límites de beneficio económico y siempre publicando las modificaciones introducidas.

Existen diferentes versiones de la arquitectura Arduino que emplean diversos microcontroladores respetando las dimensiones físicas de los conectores de ampliación y la ubicación de las señales en los mismos. El entorno de desarrollo introduce una capa de abstracción mediante la cual un conjunto de comandos y funciones puede ser empleado en cualquier plataforma Arduino.

En nuestro caso, usaremos la versión *Arduino Due* —véase la Figura 9.2— que, respecto de la tarjeta Arduino Uno original —mostrada

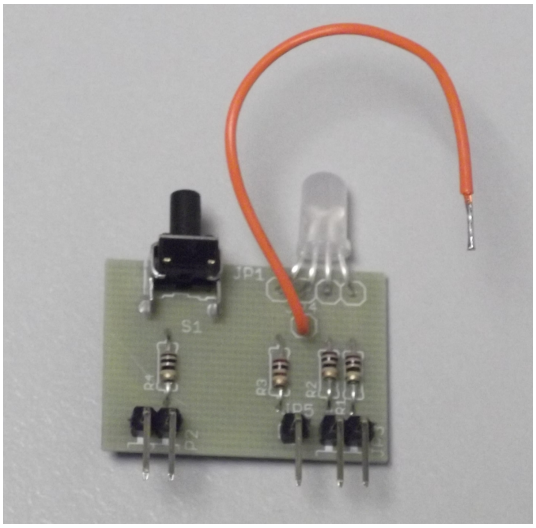

Figura 9.3: Tarjeta de E/S de prácticas de laboratorio

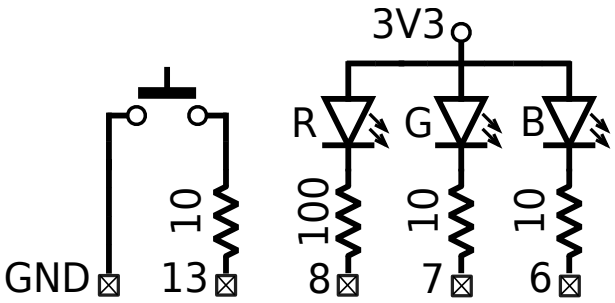

Figura 9.4: Esquema de la tarjeta de E/S de prácticas de laboratorio

en la Figura 9.1—, entre otras diferencias, presenta un microprocesador ATSAM3X8E, más potente que el ATmega328 de aquélla y con mayor número de entradas/salidas.

En la primera parte de las prácticas se ha utilizado el conjunto de instrucciones *Thumb* correspondiente a la versión Cortex-M0 de la arquitectura ARM. El microcontrolador ATMSAM3X8E de la tarjeta AruinoDue implementa la versión Cortex-M3 que utiliza un conjunto de instrucciones mayor, el *Thumb II*. Aunque todas las instrucciones *Thumb* están incluidas en *Thumb II* existe una diferencia crítica en el lenguaje ensamblador, que se señala aquí porque puede dar lugar a errores.

Las instrucciones aritméticas y lógicas del conjunto *Thumb II* permiten no modificar los indicadores de estado —*flags*— mediante cierto bit en su formato. Esta circunstancia se expresa en lenguaje ensambla-

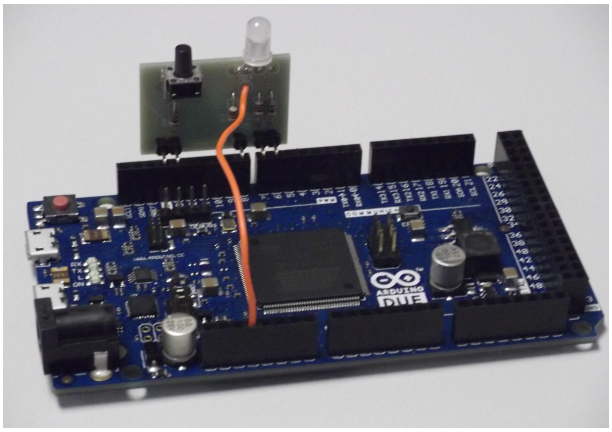

Figura 9.5: Tarjeta de E/S insertada en la Arduino Due

dor añadiendo una `s` en el código de instrucción cuando sí se vayan a modificar, de manera que se tiene:

```
and     r0, r1, r3   @ No modifica los flags
ands    r0, r1, r3   @ Sí los modifica
```

Si bien esta característica añade potencia al conjunto de instrucciones, es fácil confundirse cuando se está acostumbrado a programar con instrucciones *Thumb*, dado que todas afectan a los flags en general.

En el documento `CortexM3 Instruction Set` —disponible en el Aula Virtual de la asignatura EI1004/MT1004, sección *Teoría*, apartado *Tema 2 La Arquitectura ARM*, enlace *Cortex^{TM}-M3 Instruction Set*—, se describe el conjunto de instrucciones *Thumb II* completamente. Dado que es más potente, se recomienda consultarlo para conocer todas las posibilidades de este conjunto.

Para el desarrollo de las prácticas, se ha confeccionado una tarjeta específica —mostrada en la Figura 9.3— que contiene un pulsador y un LED RGB conectados como muestra la Figura 9.4 que se emplearán como dispositivos de E/S. La Figura 9.5 muestra la tarjeta instalada sobre la Arduino Due, donde puede apreciarse que, además de insertar la tarjeta correctamente, hay que conectar un cable de alimentación al pin de la Arduino Due rotulado con el texto `3.3V`. Como se puede ver en la Figura 9.4, el LED RGB es del tipo ánodo común, por lo que será necesario escribir un `0` en cada salida conectada a un LED para encenderlo y un `1` para mantenerlo apagado. Igualmente se puede ver cómo el pulsador se conecta a un pin y a masa. Se infiere, pues, que será necesario activar el `pull-up` de la salida 13 para leer y que en el estado no pulsado se obtendrá un `1` en el pin. Al pulsarlo, lógicamente,

cambiará a `0`.

El entorno de programación de Arduino —véase la Figura 9.6— se presenta como un editor de código en el cual podemos escribir nuestro programa, guardarlo, compilarlo y subirlo al dispositivo Arduino que tengamos conectado.

Figura 9.6: Entorno de programación Arduino

La estructura de un programa típico de Arduino consta de dos funciones:

- «`void setup()`»: Contiene el conjunto de acciones que se realizarán al inicio de nuestro programa. Aquí habitualmente configuraremos las entradas/salidas que vayamos emplear y daremos valores iniciales a las variables.
- «`void loop()`»: Contiene el código que se ejecutará indefinidamente.

Es necesario especificar al entorno el modelo de sistema Arduino que tenemos conectado para que se usen las bibliotecas adecuadas y se asignen correctamente las señales de E/S, se acceda correctamente a los registros del procesador, etcétera. Para ello emplearemos la entrada

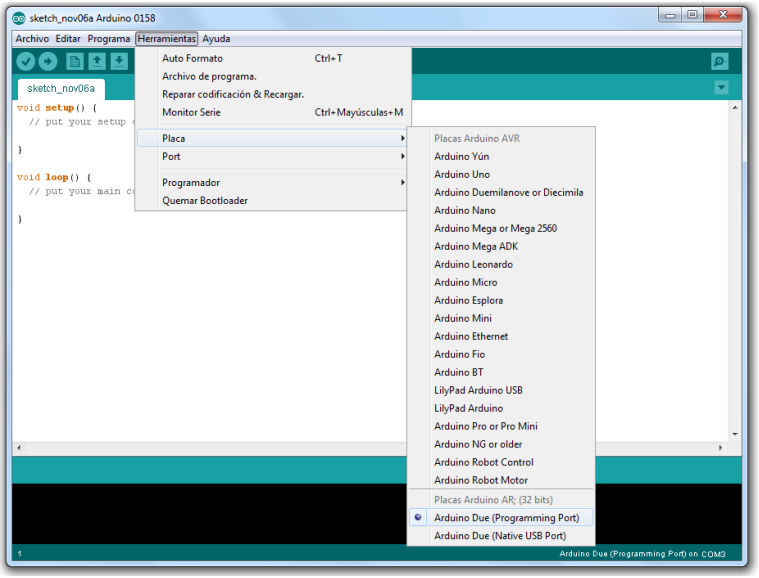

Figura 9.7: Selección del sistema Arduino a emplear en entorno Windows

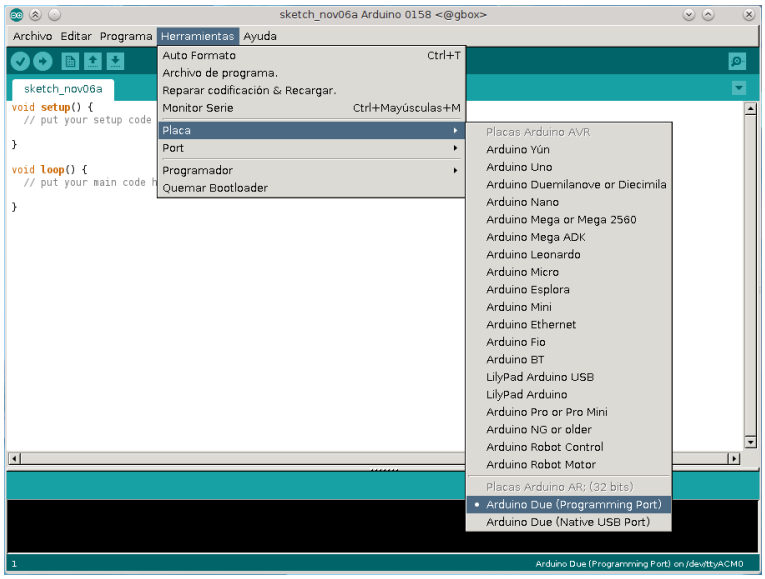

Figura 9.8: Selección del sistema Arduino a emplear en entorno Linux

«`Placa`» dentro del menú «`Herramientas`». En nuestro caso seleccionaremos la opción «`Arduino Due (Programming Port)`» —véase la Figura 9.7 para el entorno Windows y la Figura 9.8 para el entorno Linux—. Los

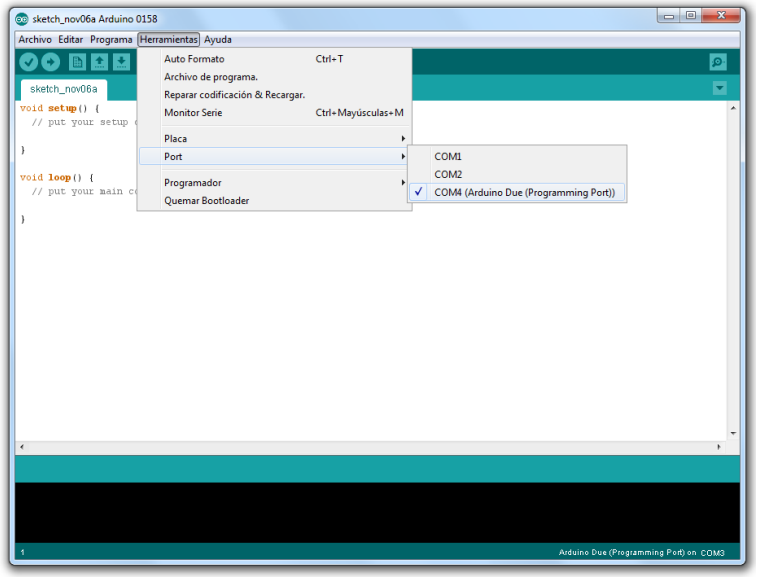

Figura 9.9: Selección del puerto de comunicaciones en entorno Windows

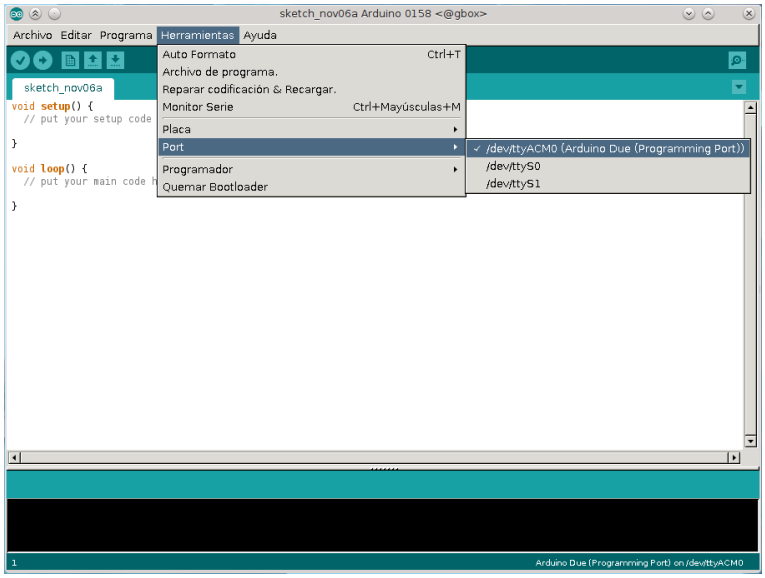

Figura 9.10: Selección del puerto de comunicaciones en entorno Linux

dos puertos USB de la tarjeta Arduino Due están identificados en la cara de soldaduras de la misma.

De la misma forma, hay que indicar el puerto serie de comunica-

ciones en que está conectado el sistema Arduino, lo cual especificaremos mediante la entrada «`Port`» dentro del menú «`Herramientas`», como muestran la Figura 9.9 para el entorno Windows y la Figura 9.10 para el entorno Linux. Como opciones se nos ofrecerán los puertos serie —`/dev/ttyx` o `COMx` en Linux o Windows respectivamente— de que disponga el sistema. Tanto en el entorno Windows como en el entorno Linux aparece como opción un puerto —que es donde se encontrará conectada nuestra tarjeta Arduino Due— junto a cuyo nombre aparecerá el texto *(Arduino Due (Programming Port))* y es, por tanto, el que debemos seleccionar.

9.2. Creación de proyectos

El entorno Arduino posee una estructura denominada *proyecto* —*sketch* en la bibliografía Arduino— que contiene los archivos correspondientes a cada programa. Cuando se inicia el entorno, se nos muestra un proyecto vacío con un nombre del tipo «`sketch_mmmdda`» —donde «`mmm`» es la abreviatura del nombre del mes actual y «`dd`» es el día del mes— que podemos usar como base para desarrollar un nuevo programa. De la misma forma, mediante la entrada «`Abrir...`» dentro del menú «`Archivo`» podemos abrir un proyecto existente.

Los archivos que componen un proyecto se guardan en una carpeta cuyo nombre coincide con el del proyecto. Generalmente un proyecto consta de un solo archivo de extensión «`.ino`» —con el mismo nombre que la carpeta— que contiene el código en lenguaje «`C / C++`» del programa principal.

Un programa puede, sin embargo, constar de más de un archivo. En tal caso, para añadir archivos al proyecto emplearemos el botón del entorno marcado con un recuadro en la esquina superior derecha en la Figura 9.6, que desplegará un menú del cual elegiremos la opción `Nueva Pestaña`. Al hacerlo, en la parte inferior del entorno aparecerá una barra en la que se nos solicitará el nombre de la nueva pestaña —y por tanto del archivo en que se guardará su contenido— donde deberemos especificar tanto el nombre como la extensión de dicho archivo y crear tanto la pestaña como el archivo pinchando en el botón `Ok`.

La versión del entorno que se va a usar en las prácticas ha sido modificada por el profesorado de la asignatura EI1004 para permitir el uso de archivos fuente en ensamblador. Las instrucciones para instalar esta versión se encuentran en el Aula Virtual de dicha asignatura, sección *Laboratorio*, apartado *Material adicional prácticas con Arduino*, enlace *Guías de instalación de Arduino (versión UJI)*.

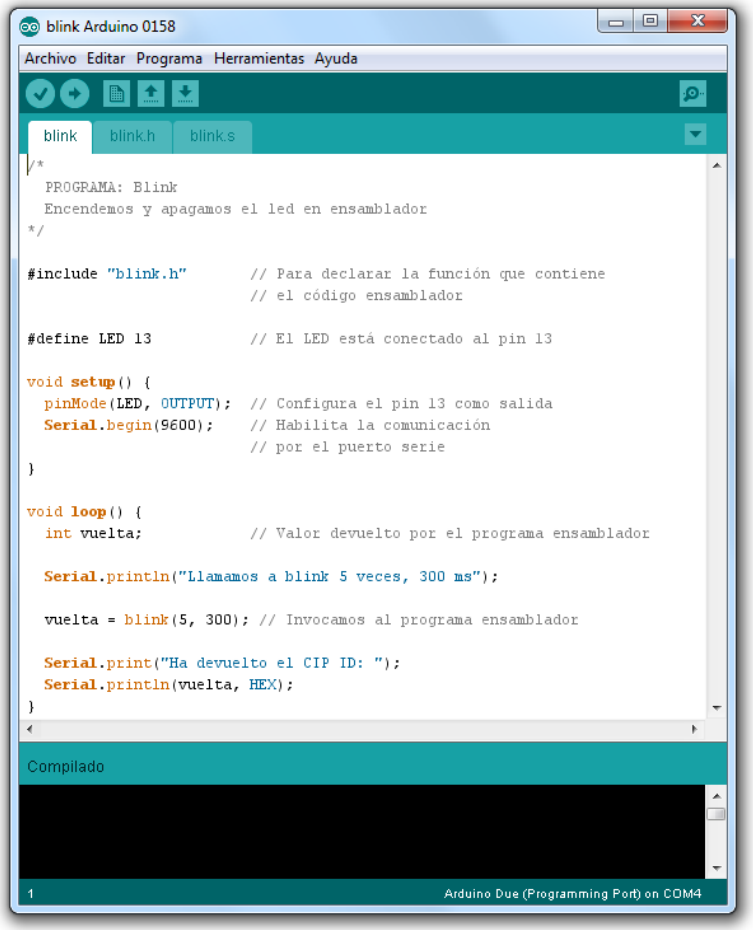

Figura 9.11: Entorno Arduino con el programa «blink» cargado

9.2.1. Ejemplo

Como ejemplo básico vamos a mostrar un programa que hace parpadear el LED incorporado en la tarjeta Arduino. De momento no es necesario que la tarjeta esté conectada al computador, con lo que esta fase se puede llevar a cabo sin necesidad de poseer una tarjeta Arduino. El código es el mostrado en la Figura 9.11:

Una vez introducido el código como texto, podemos comprobar que es correcto —lo cual implica *compilarlo*— mediante el botón de la parte izquierda de la barra de botones —marcado con un círculo en la Figura 9.6 y que hace aparecer el texto `Verificar` (tanto en Linux como en Windows) cuando situamos el cursor sobre él—. Tras el proceso de compilación se mostrarán los posibles errores detectados indicando el número de línea en que se encuentra cada uno de ellos o, si no hay errores,

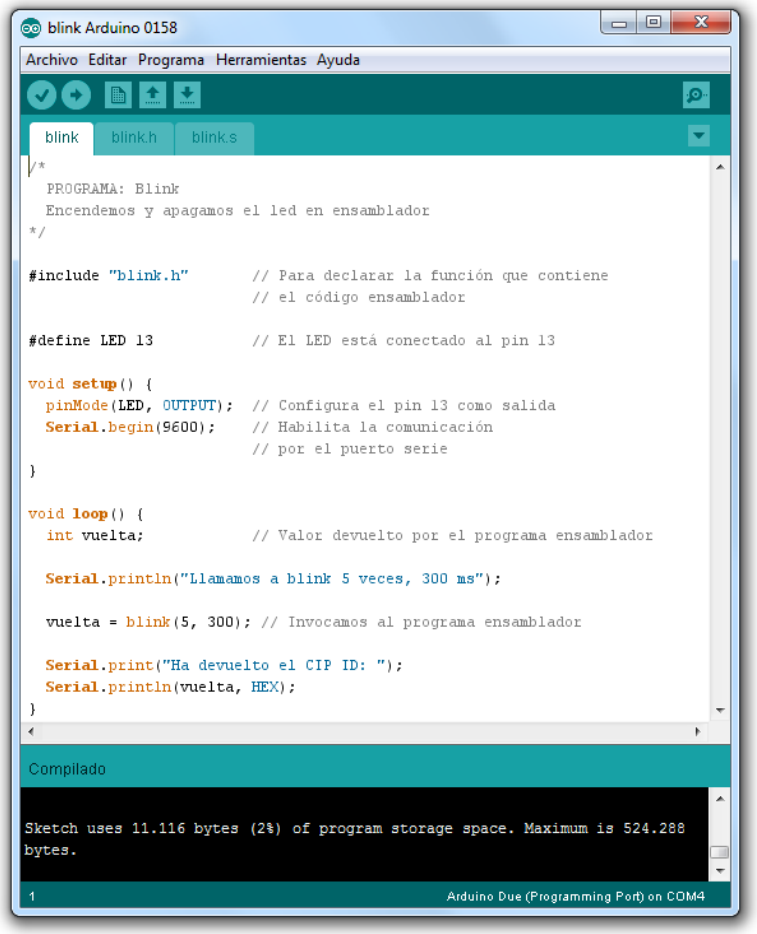

Figura 9.12: Resultado de la compilación del programa «blink»

la cantidad de memoria ocupada por el código generado y el máximo de que dispone la tarjeta Arduino seleccionada actualmente, como puede apreciarse en la Figura 9.12.

Mediante el segundo botón de la barra de botones —marcado con un cuadrado en la Figura 9.6 y que hace aparecer el texto `Subir` (en Windows) o `Cargar` (en Linux) cuando situamos el cursor sobre él— se desencadena el mismo proceso pero, si en la compilación no se han producido errores, el código generado es enviado a la tarjeta Arduino —que ahora sí debe estar conectada al computador— y ejecutado de inmediato. De hecho, la programación se verifica comunicando el ordenador con la tarjeta y forzando a que se ejecute un programa especial llamado *bootloader*. Este programa lee del puerto USB-serie las instrucciones a cargar en la ROM. Una vez terminada la comunicación, el ordenador

fuerza un RESET y el microcontrolador comienza a ejecutar el programa descargado.

Siguiendo el procedimiento descrito se puede programar el microcontrolador empleando el lenguaje de alto nivel `C / C++` y las funciones específicas de Arduino. Nuestro objetivo, sin embargo, es programar el microcontrolador empleando directamente su lenguaje ensamblador y para conseguirlo vamos a introducir unas ligeras modificaciones en el código.

En primer lugar, escribiremos nuestro programa ensamblador en un archivo con la extensión `.s` para identificarlo como código ensamblador. En el caso del programa `blink` que hace parpadear el LED de Arduino, el código ensamblador correspondiente será:

blink.s

```
# blink.s - Parpadeo en ensamblador
# Acceso al controlador del PIO B

.syntax unified
.cpu cortex-m3
.text
.align  2
.thumb
.thumb_func
.extern delay                   @ No es necesario
.global blink                   @ Función externa
.type blink, %function

.equ PIOB,      0x400E1000      @ Dir. base del puerto B
.equ SODR,      0x030           @ OFFSET Set Output Data Reg
.equ CODR,      0x034           @ OFFSET Clear Output Data Reg
.equ CHIPID,    0x400E0940      @ Registro CHIP ID
.equ LEDMSK,    0x08000000      @ El LED está en el pin 27

/* int blink(int times, int delay)
     r0 = times. Número de veces que parpadea
     r1 = delay. Retardo del parpadeo
     Devuelve el CHIP_ID, porque sí
     Los parámetros se pasan en r0-r3
     El valor devuelto en r0 ó r0-r1 si ocupa 8 bytes
     Cualquier función puede modificar r0-r3
     El resto se han de preservar */

blink:
   push         {r4-r7, lr}     @ Vamos a usar de r4 a r7
                                @ porque llamamos a delay
   mov          r4, r0          @ r4 contiene el número de veces
   mov          r5, r1          @ r5 contiene el retardo a pasar a delay
```

```
34    ldr          r6, =PIOB       @ Dirección base del Controlador PIO B
35    ldr          r7, =LEDMSK     @ Máscara con el bit 27 a 1 (pin del LED)
36 principio:
37    str          r7, [r6, #SODR] @ Encendemos el LED escribiendo en SET
38    mov          r0, r5          @ Preparamos el parámetro de delay en r0
39    bl           delay           @ Invocamos a la función delay
40    str          r7, [r6, #CODR] @ Apagamos el LED escribiendo en CLEAR
41    mov          r0, r5          @ Volvemos a llamar a delay como antes
42    bl           delay
43    subs         r4, r4, #1      @ Decrementamos el número de veces
44    bne          principio       @ y si no es cero seguimos. ¡Ojo a la s!
45    ldr          r6, =CHIPID     @ Leemos CHIPID_CIDR
46    ldr          r0, [r6]        @ y devolvemos el valor en r0
47    pop          {r4-r7, pc}     @ ret con pop al pc.
48 .end
```

Para poder ejecutar este código desde el entorno de programación de Arduino es necesario indicar durante el proceso de compilación que se utilizan funciones en otro módulo, y que siguen el convenio de llamada de funciones de lenguaje `C`, ligeramente distinto del de `C++`. Para ello emplearemos un fichero de cabecera con extensión `.h` que llamaremos `blink.h` y cuyo contenido será:

blink.h ✍

```
1 // Declaración de las funciones externas
2
3 extern "C" {
4   int blink(int times, int del);
5   }
```

Este código define una función llamada `blink` que acepta dos argumentos. El primer argumento indica el número de veces que se desea que el LED parpadee y el segundo argumento el periodo de tiempo en milisegundos que el LED permanecerá encendido y apagado en cada ciclo de parpadeo, es decir, el ciclo completo tendrá una duración del doble de milisegundos que el valor de este argumento.

Finalmente, el programa principal se encarga de definir los valores iniciales de las variables y de invocar el código en ensamblador que efectivamente hará parpadear el LED.

blink.ino ✍

```
1 /*
2   PROGRAMA: Blink
3   Encendemos y apagamos el led en ensamblador
4 */
5
6 #include "blink.h"       // Para declarar la función que
```

```
7                          // contiene el código ensamblador
8
9  #define LED 13          // El LED está conectado al pin 13
10
11 void setup() {
12   pinMode(LED, OUTPUT);  // Configura el pin 13 como salida
13   Serial.begin(9600);    // Habilita la comunicación por el puerto serie
14   int vuelta;            // Valor devuelto por el programa ensamblador
15 }
16
17 void loop() {
18
19   Serial.println("Llamamos␣a␣blink␣5␣veces,␣300␣ms");
20
21   vuelta = blink(5, 300); // Invocamos el programa ensamblador
22
23   Serial.print("Ha␣devuelto␣el␣CIP␣ID:␣");
24   Serial.println(vuelta, HEX);
25 }
```

En este programa podemos, además, señalar que se ha hecho uso de la comunicación serie incorporada en la plataforma Arduino para obtener mensajes durante la ejecución del programa. Para ello hay que activar esta funcionalidad dentro de «`setup()`» mediante la llamada a la función «`Serial.begin(9600)`», donde el argumento indica la velocidad de comunicación en baudios. Posteriormente, ya dentro de la función «`loop`», se pueden enviar mensajes a través del puerto serie —asociado al USB— empleando las funciones «`Serial.print`» —muestra el texto que se le pasa como argumento y permite seguir escribiendo en la misma línea— y «`Serial.println`» —muestra el texto y pasa a la línea siguiente—. El argumento de estas funciones puede ser una cadena de caracteres entre comillas —que se mostrará textualmente— o una variable, en cuyo caso se mostrará su valor. En la página *www.arduino.cc* —o desde el propio entorno— se puede acceder a la referencia para obtener información sobre las funciones de Arduino. Para visualizar los mensajes recibidos hay que iniciar el *Monitor Serie*, lo cual se consigue pinchando sobre el botón del extremo derecho de la barra de botones —que contiene el icono de una lupa y que hace aparecer el texto `Monitor Serie` (en Windows) o `Monitor Serial` (en Linux) cuando situamos el cursor sobre él—. Hay que tener en cuenta que al iniciar el *Monitor Serie* se envía a la tarjeta Arduino una señal de *RESET*.

Identificación de las entradas/salidas

El estándar Arduino otorga a cada entrada/salida un número de identificación que es independiente del modelo de tarjeta Arduino em-

pleada. Así pues, la salida número 13 está conectada a un diodo LED incorporado en la tarjeta Arduino y es la salida que usa el programa mostrado. En nuestro caso, el diodo LED RGB de la tarjeta de prácticas está conectado a los pines de E/S números 6 —azul—, 7 —verde— y 8 —rojo—, mientras que el pin 13 está conectado al pulsador, como se muestra en el Cuadro 8.5.

9.3. Problemas del capítulo

9.3.1. Introducción a la E/S

EJERCICIOS

▶ **9.1** Conecta a la tarjeta Arduino la tarjeta de prácticas de forma que los tres pines bajo el LED se correspondan con los pines `6`, `7` y `8` y los otros dos pines estén conectados al pin `13` y al pin `GND` que hay junto a él. Recuerda conectar el cable al pin `3.3V` de la tarjeta Arduino. Inicia el entorno Arduino y abre el proyecto `blink` mediante la opción `Archivo - Ejemplos - 01.Basics - Blink` del menú. Compílalo y súbelo a la tarjeta. Comprueba que el LED incorporado en la Arduino Due parpadea —de color amarillo, situado aproximadamente entre los dos conectores USB de la misma e identificado con la letra `L`—.

Sustituye en `blink.c` las tres apariciones del número 13 (como argumento de la función «`pinMode`» y de las dos llamadas a la función «`digitalWrite`») por el número 6. Compila y sube a la tarjeta el nuevo programa. ¿Cómo ha cambiado el comportamiento del programa?

▶ **9.2** Modifica el programa `blink.c` para que haga parpadear el LED de color rojo.

▶ **9.3** Descarga del Aula Virtual, sección *Laboratorio*, apartado *Programas para Arduino*, los archivos del programa `blink_asm`, compílalo y ejecútalo. Comprueba que el LED de la tarjeta Arduino Due parpadea. Recordemos que el microcontrolador ATSAM3X8E, incorporado en la tarjeta Arduino Due que estamos usando, posee varios `PIOs` (*Parallel Input Output*) con varios pines de E/S cada uno de ellos, de forma que los pines `6`, `7` y `8` de la tarjeta Arduino están físicamente conectados a los pines `24`, `23` y `22` del `PIO C` del ATSAM3X8E respectivamente, como se muestra en el Cuadro 8.5.

Consulta el Cuadro 8.1 para determinar la dirección del registro base del `PIO C` y realiza las modificaciones necesarias en `blink_asm.c` y en `blink.s` para que haga parpadear el LED de color rojo. Ten en cuenta que, mientras que el LED incorporado en la Arduino Due se enciende escribiendo un `1` y se apaga escribiendo un `0` en el puerto correspondiente, cada componente del LED RGB de la tarjeta de prácticas se encien-

de escribiendo un `0` y se apaga escribiendo un `1` en su puerto de E/S. Comenta qué modificaciones has tenido que introducir en el programa.

▶ **9.4** Tal como vimos al estudiar las funciones, hay dos formas de pasar parámetros: por valor y por referencia. En el ejemplo propuesto se muestra la técnica de paso de parámetros por valor a un programa en ensamblador —a través de los registros `r0` y `r1`—. Alguna vez, sin embargo, será necesario poder acceder desde el programa en ensamblador a una variable del programa principal. Debemos establecer un mecanismo para poder transferir información entre el programa en `C` y el código ensamblador. Para ello declaramos un vector en `C` y declaramos su nombre como `.extern` en el programa ensamblador para acceder a él usando su propio nombre como etiqueta.

Descarga del Aula Virtual el programa «`blink_cadena`» y observa los siguientes cambios respecto del programa «`blink_asm`»:

- Se ha modificado la declaración de la función `blink` en el fichero «`blink.h`» para que no acepte parámetros pero siga devolviendo un resultado. Para ello se han eliminado las declaraciones de los dos parámetros: «`int time`» e «`int del`». También se han eliminado los parámetros entre paréntesis en la invocación a la función `blink` en el ficheo «`blink_asm.ino`».

- Se ha declarado una cadena de caracteres al principio del programa en `C`, con el contenido «`mensaje`» donde es importante que el último elemento de la cadena sea el carácter `0`.

- Se ha declarado el nombre de la cadena en el programa ensamblador como «`.extern`» para que dicho nombre pueda utilizarse como una etiqueta dentro del program ensamblador.

- Al eliminar los paŕametros de la función `blink`, en el programa en ensamblador hemos asignado al retardo una cantidad fija —#300— y el número de veces que parpadea el LED será el número de caracteres de que consta la cadena. Para ello se ha confeccionado un bucle que recorre la cadena y realiza un parpadeo por cada carácter de la misma hasta encontrar el `0` del final.

Completa el programa ensamblador para que realice la función descrita. Compílalo y súbelo a la tarjeta Ardunio Due. Modifica la longitud de la cadena para comprobar que realmente el programa hace lo que se espera.

▶ **9.5** Modifica el programa ensamblador para que devuelva el número de caracteres de la cadena simplemente copiándolo en el registro `r0`.

▶ **9.6** Modifica el programa en `C` para que muestre en pantalla el número de caracteres de la cadena. Recuerda que las funciones que muestran información en pantalla son «`Serial.print`» y «`Serial.println`».

▶ **9.7** La técnica de compartición de variables se puede emplear también para devolver información desde el programa en ensamblador al programa invocador. En lenguaje `C` se puede reservar espacio para un vector de enteros llamado «`vector`», de forma equivalente a como haríamos con «`.space m`» en ensamblador, de la siguiente forma:

```
«int vector[n];»
```

Hay que tener en cuenta, sin embargo, que el parámetro «`m`» indica número de bytes a reservar, mientras que cada elemento del vector ocupa 4 bytes —1 word— en memoria. Así pues, para realizar la reserva en memoria de un vector de «`n`» elementos, en ensamblador debemos usar $m = 4 * n$.

Así mismo, desde el programa en `C` podemos acceder al elemento i-ésimo del vector mediante «`vector[i]`», pudiendo «`i`» tomar valores entre «`0`» y «`n-1`».

Considerando lo expuesto y empleándolo de forma adecuada, descarga el programa «`blink_vect`» del Aula Virtual y complétalo para que devuelva los contenidos de los registros `r0` al `r7` en el vector llamado «`retorno`» y los muestre en pantalla empleando «`Serial.print`» y «`Serial.println`». Ten en cuenta que estas funciones pueden mostrar valores numéricos en hexadecimal si se les añade el modificador «`HEX`», como por ejemplo en «`Serial.println(n, HEX);`»

▶ **9.8** Consulta el Cuadro 8.8 y, a partir de su contenido, completa el programa «`leefecha`» del Aula Virtual para que acceda a los registros `RTC_CALR` y `RTC_TIMR` y lea la configuración de fecha y hora actuales del ATSAM3X8E. Comparte esa información con el programa principal mediante los vectores `fecha` y `hora` y muestra en pantalla la fecha y hora actuales usando adecuadamente las funciones «`Serial.print`» y «`Serial.println`». ¿Cuál es la fecha con que se configura el RTC por defecto?

▶ **9.9** En el campo `DAY` del registro `RTC_CALR` se almacena el día de la semana dejando la codificación al criterio del usuario. Atendiendo al contenido de este campo cuando se inicializa el sistema, ¿qué codificación se emplea por defecto para el día de la semana?

9.3.2. E/S por consulta de estado

EJERCICIOS

▶ **9.10** Sabiendo que el pulsador incorporado en la tarjeta de prácticas de laboratorio está conectado a un pin del PIOB, ¿qué registro deberíamos leer para detectar que se ha presionado el pulsador?

▶ **9.11** El pin al que está conectado el pulsador es el correspondiente al bit 27 del PIOB. ¿Qué máscara tenemos que aplicar al valor leído del registro para determinar la posición del pulsador?

▶ **9.12** De acuerdo con el esquema de conexión del pulsador de la tarjeta de prácticas mostrado en la Figura 9.4 y si la resistencia de pull-up de la entrada donde está conectado el pulsador está activada, ¿qué valor leído del bit 27 del registro de E/S del PIOB nos indicará que el pulsador está presionado?

▶ **9.13** Descarga el programa «`pulsa`» del Aula Virtual y complétalo para que, mediante consulta de estado, espere la pulsación del pulsador de la tarjeta de prácticas para regresar devolviendo el `CHIPID` del procesador.

▶ **9.14** Modifica el programa anterior para que regrese cuando se suelte el pulsador tras haberlo pulsado en lugar de cuando se pulse.

▶ **9.15** Completa el programa «`cambia`», disponible en el Aula Virtual, para que con cada pulsación del pulsador encienda cíclicamente el LED RGB con cada uno de sus tres colores básicos.

▶ **9.16** Para poder modificar los contenidos de los registros de fecha —`RTC_CALR`— y hora —`RTC_TIMR`— es preciso detener su actualización escribiendo un `1` en los bits `UPDCAL` y `UPDTIM` del registro de control `RTC_CR` y esperando la confirmación de que se ha detenido la actualización en el bit `ACKUPD` del registro de estado `RTC_SR`. Puedes consultar los detalles de este procedimiento en el apartado *Actualización de la fecha y hora actuales*. Completa el programa «`cambiahora`» para configurar el RTC en el modo de 12 horas con la fecha y hora actuales. Haz que el programa compruebe los valores de los bits `NVCAL` y `NVTIM` para confirmar que la configuración ha sido correcta.

▶ **9.17** Partiendo del programa «`alblink`» disponible en el Aula Virtual, configura una alarma que se active dentro de 8 segundos. Completa el código proporcionado para que consulte el estado del bit de alarma y espere a que ésta se produzca para regresar al programa en `C`, el cual mostrará el instante en que se detectó la activación de la alarma. Ten en cuenta que tienes que copiar la función `cambiahora` confeccionada en el apartado anterior en la zona indicada del código fuente proporcionado.

▶ **9.18** Modifica el periodo del parpadeo para que sea de 6 segundos (3.000 ms encendido y 3.000 ms apagado). ¿Coincide el instante de detección de la alarma con el configurado?¿A qué crees que es debido?

9.3.3. E/S por interrupciones

EJERCICIOS

▶ **9.19** En el programa «`PIOint`», disponible en el Aula Virtual, completa la función «`PIOsetupInt`» para que configure adecuadamente el modo de interrupción seleccionado por los parámetros que se le suministran. Una vez completada, comprueba que funciona correctamente compilando y subiendo el programa a la tarjeta Arduino Due.

▶ **9.20** Modifica el código anterior para ir probando todas las posibles combinaciones de los parámetros «`mode`» y «`polarity`». Completa la siguiente tabla, comentando en cada caso cuál es el comportamiento del programa.

Mode	Polarity	Efecto
FLANCO	BAJO	
	ALTO	
NIVEL	BAJO	
	ALTO	
CAMBIO	BAJO	
	ALTO	

▶ **9.21** Completa en el programa «`RTCint`», disponible en el Aula Virtual, los valores de las etiquetas «`HOY`» y «`AHORA`» para configurar el RTC con la fecha y hora indicadas en los comentarios de las líneas de código que asignan valor a dichas etiquetas.

▶ **9.22** Completa asimismo los valores de las etiquetas «`ALR_FECHA`» y «`ALR_HORA`» para configurar la alarma del RTC de forma que se active a la fecha y hora indicadas en los comentarios de las líneas de código que asignan valor a dichas etiquetas.

▶ **9.23** Completa la función «`RTCsetAlarm`» de forma que configure adecuadamente la fecha y hora de activación de la alarma contenidas en las etiquetas «`ALR_FECHA`» y «`ALR_HORA`» y que active la generación de interrupciones de alarma por parte del `RTC`.

▶ **9.24** Completa en el programa anterior la función «`RTC_Handler`» para que atienda la interrupción de alarma del RTC realizando correctamente las acciones que se describen en los comentarios del código proporcionado. Una vez completada la función, compila y sube el programa a la tarjeta Arduino Due. Explica qué acciones realiza.

▶ **9.25** Prueba diferentes valores de la etiqueta «`RETARDO`», comprueba su efecto en el comportamiento del programa y compáralo con el obtenido con el programa «`alblink`» de la sesión anterior. Comenta las diferencias observadas y relaciónalas con los métodos de consulta de estado e interrupciones.

9.3.4. Pulse Width Modulation (PWM), Direct Memory Access (DMA) y Universal Serial Bus (USB)

EJERCICIOS

▶ **9.26** Consulta en la ayuda de Arduino el funcionamiento de la función «`analogWrite()`» y, al final de la misma, el enlace «`Tutorial:PWM`». ¿Cómo se consigue variar la intensidad luminosa del LED mediante la técnica PWM

▶ **9.27** Descarga el programa «`pwm`» del Aula Virtual, compílalo y súbelo a la tarjeta Arduino Due.

▶ **9.28** Observa que con cada pulsación cambia la intensidad del LED rojo mientras se indica el valor del ciclo de trabajo (*Duty Cycle*) que provoca la intensidad observada. Explica qué relación hay entre el valor del registro `PWM Channel Duty Cycle Register` y la intensidad del LED.

▶ **9.29** Desplaza la tarjeta Arduino Due con suavidad describiendo un arco de unos 60 centímetros a una velocidad moderada y observa el patrón que se visualiza para los diferentes valores del contenido del registro `PWM Channel Duty Cycle Register`. Explica cómo estos patrones obedecen al principio de funcionamiento del PWM visto en el tutorial de Arduino.

▶ **9.30** Cambia el valor actual de la etiqueta «`CLK`» (`0x0000 0680`) por `0x0000 0180`. Repite el experimento anterior y comenta las diferencias apreciadas. ¿Qué cres que ha cambiado?

▶ **9.31** Descarga el programa «`EjemploPWM`» del Aula Virtual, compílalo y súbelo a la tarjeta Arduino Due. ¿Cómo se consiguen los diferentes colores en el LED RGB?

▶ **9.32** Prueba diferentes valores de «`FACTRED`», «`FACTGRN`» y «`FACTBLU`». ¿Qué efecto tienen estos parámetros en el comportamiento del programa?

▶ **9.33** Descarga el programa «`testDMA`» del Aula Virtual, compílalo y súbelo a la tarjeta Arduino Due. Observa en el funcionamiento del programa que el contador de iteraciones se incrementa al mismo tiempo que el DMA realiza las transferencias de datos.

▶ **9.34** Busca en el código del programa el grupo de líneas encerrado en el comentario «`TAMAÑO DE LOS BLOQUES A TRANSFERIR`» y prueba diferentes valores de los parámetros «`SIZE0`», «`SIZE1`», «`SIZE2`» y «`SIZE3`». Ten en cuenta que deben tener valores comprendidos entre 100 y 2048. Explica cómo cambia el tiempo de ejecución en función de dichos valores.

▶ **9.35** ¿Para qué valores de los parámetros se obtiene el tiempo máximo de realización de las transferencias?¿De qué tiempo se trata?¿Qué valor ha alcanzado el contador de iteraciones para dicho tiempo máximo?

▶ **9.36** Descarga el programa «`kbmouse`» del Aula Virtual, compílalo y súbelo a la tarjeta Arduino Due. Abre un editor de textos. Desconecta el cable USB del puerto de programación de la tarjeta Arduino Due y conéctalo al puerto de comunicaciones. Observa el funcionamiento del programa.

▶ **9.37** Consulta la ayuda de Arduino y modifica el programa para que simule un doble click en el instante en que se presiona el pulsador de la tarjeta Arduino Due. Presiona el pulsador cuando el puntero se encuentre sobre un icono del escritorio y comprueba que se inicia la aplicación correspondiente.

▶ **9.38** Modifica el programa para que, en lugar de una elipse, el puntero del ratón describa un rectángulo.

Bibliografía

[Adv95] Advanced RISC Machines Ltd (ARM) (1995). *ARM 7TDMI Data Sheet*.
URL http://www.ndsretro.com/download/ARM7TDMI.pdf

[Atm11] Atmel Corporation (2011). *ATmega 128: 8-bit Atmel Microcontroller with 128 Kbytes in-System Programmable Flash*.
URL http://www.atmel.com/Images/doc2467.pdf

[Atm12] Atmel Corporation (2012). *AT91SAM ARM-based Flash MCU datasheet*.
URL http://www.atmel.com/Images/doc11057.pdf

[Bar14] S. Barrachina Mir, G. León Navarro y J. V. Martí Avilés (2014). *Conceptos elementales de computadores*.
URL http://lorca.act.uji.es/docs/conceptos_elementales_de_computadores.pdf

[Cle14] A. Clements (2014). *Computer Organization and Architecture. Themes and Variations. International edition.* Editorial Cengage Learning. ISBN 978-1-111-98708-4.

[Shi13] S. Shiva (2013). *Computer Organization, Design, and Architecture, Fifth Edition*. Taylor & Francis. ISBN 9781466585546.
URL http://books.google.es/books?id=m5KlAgAAQBAJ