Programación de Arduino con las herramientas AVR-GCC Toolchain

Tabla de contenidos

1. Introducción

En esta guía se explica cómo realizar la programación de la plataforma de hardware libre Arduino en lenguaje C para microcontroladores AVR mediante el uso de la cadena de herramientas AVR-GCC, es decir, sin utilizar el Entorno de Desarrollo Integrado de Arduino (Arduino IDE).

Arduino IDE es un entorno de desarrollo codificado en Java que permite programar tarjetas Arduino mediante un lenguaje de programación basado en Wiring. Este lenguaje no es más que una capa de software encima del lenguaje C, incluyendo librerías que permiten al usuario un acceso rápido a los diversos dispositivos internos del microcontrolador (puerto UART, puerto SPI, entre otros) así como un manejo sencillo de dispositivos externos (pantallas LCD, servomotores, entre otros).

Sin embargo, existen personas que prefieren programar microcontroladores utilizando directamente lenguaje C. Esto se debe a que mediante la programación en C se tiene un mejor control sobre lo que se ejecuta en el microcontrolador (no hay código intermedio oculto, lo que programas es lo que se ejecuta), además de un acesso directo a las interrupciones de hardware disponibles (esto depende del microcontrolador).

Para esto, se disponen de un conjunto de herramientas libres que permiten la compilación de código en C en archivos .hex que posteriormente sean subidos

directamente al microcontrolador; en este caso, el microcontrolador que posea la tarjeta arduino con que se desea programar.

2. Herramientas a utilizar

Las herramientas que se utilizan en esta guía son las siguientes:

- **Code::Blocks:** un entorno de desarrollo integrado que permite crear proyectos en C para microcontroladores AVR. También permite la creación de archivos .hex para luego ser enviados al microcontrolador.
- **Avrdude:** es una herramienta de línea de comandos que permite tanto leer como escribir el contenido de las memorias de los microcontroladores, a través de una interfaz de hardware (conocido como programador). Es nuestro caso, la tarjeta arduino en sí funciona como programador.
- **Gcc-avr:** es el compilador libre, creado por la fundación GNU, para la compilación de código en C compatible con los dispositivos AVR de Atmel.
- **Avr-libc:** son las librerías del lenguaje C para su utilización en la programación de microcontroladores AVR.
- **Binutils-avr:** este paquete provee las utilidades de bajo nivel necesarias para que el proceso de compilación del código en C se realice de manera correcta.

Todas estas herramientas se encuentran disponibles para su instalación en los repositorios de Ubuntu, por lo que pueden ser instaladas fácilmente ubicándolas e instalándolas con el manejador de paquetes Synaptic.

3. Programación de Arduino en lenguaje C en Code::Blocks

Una vez instaladas las herramientas, se puede comenzar a crear el programa en lenguaje C.

Para esto, se ejecuta el Code::Blocks.

Luego se ingresa al menú Settings->Compiler and Debugger.

En este menú se procede a cambiar el Compiler por **GNU AVR GCC Compiler**.

Luego se ubica la pestaña Toolchain Executables y se confirma que la información mostrada esté de la siguiente forma (cuadro 1):

C compiler:	avr-gcc
C++ compiler:	avr-g++
Linker for dynamic libs:	avr-g++
Linker for static libs:	avr-ar
Debugger:	avr-gdb
Resource compiler:	
Make program:	make

Cuadro 1. Configuraciones de Toolchain Executables

Si las configuraciones son las correctas, se selecciona Debugger Settings en el panel ubicado a la izquierda y se desmarca la casilla “Auto-build Project to ensure up-to-date”.

Ahora se puede proceder a crear el proyecto.

Se va al menú File->New->Project (figura 1).

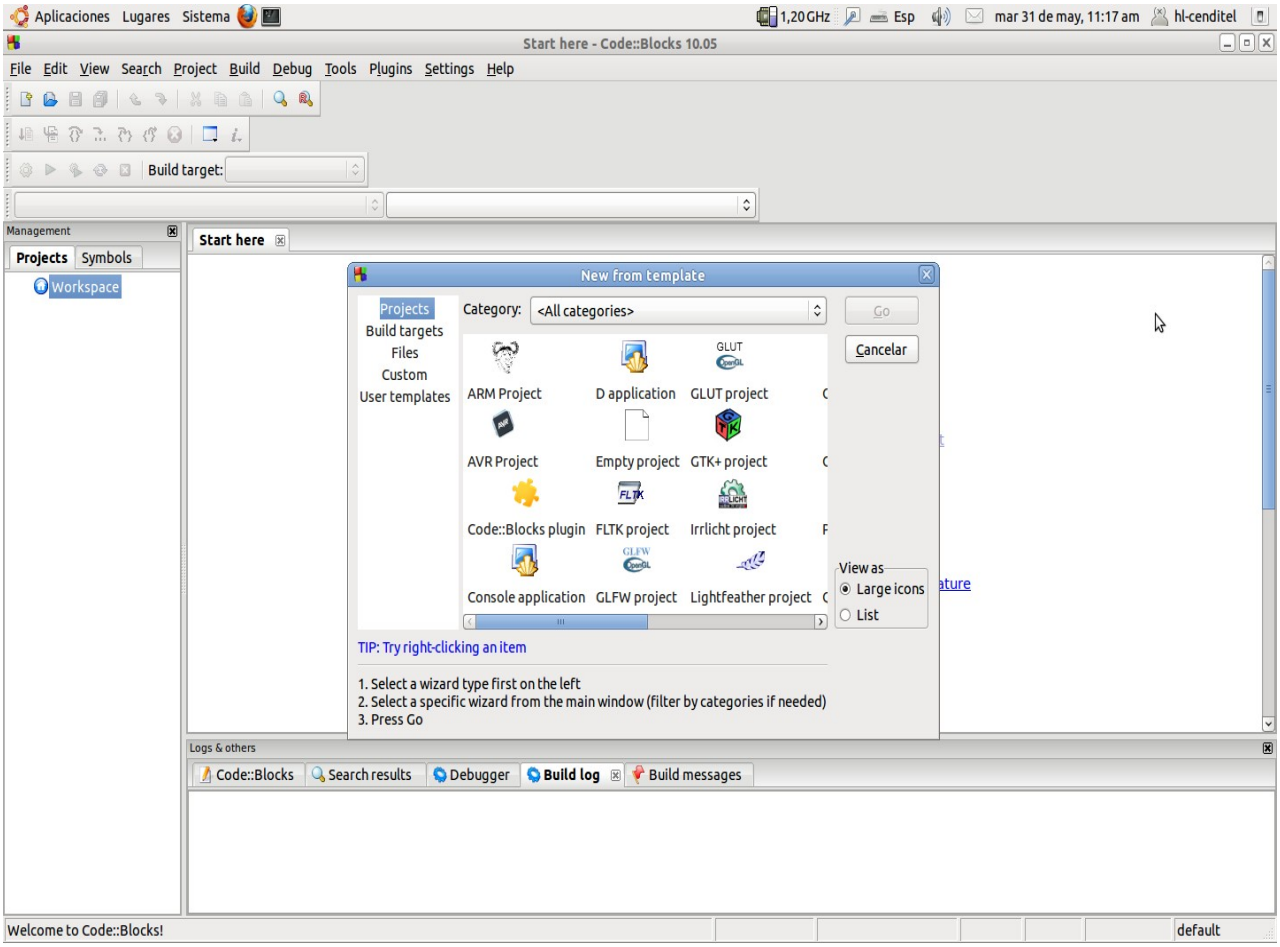

Figura 1. Creación de proyecto AVR

Se selecciona AVR Project y se llena la información referente al proyecto (figura 2).

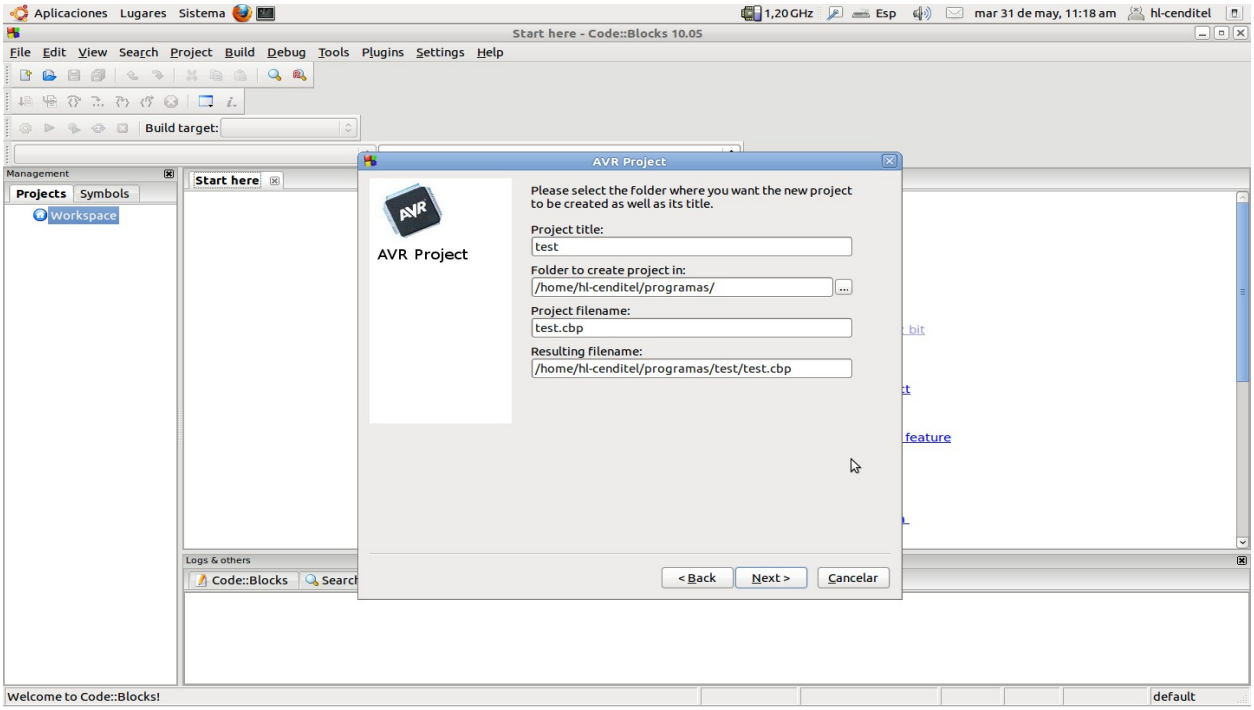

Figura 2. Información del proyecto

En la pantalla siguiente se debe verificar que el compilador seleccionado sea GNU AVR GCC Compiler, y a la vez verificar que las opciones “Create Debug” y “Create Release” estén marcadas (figura 3).

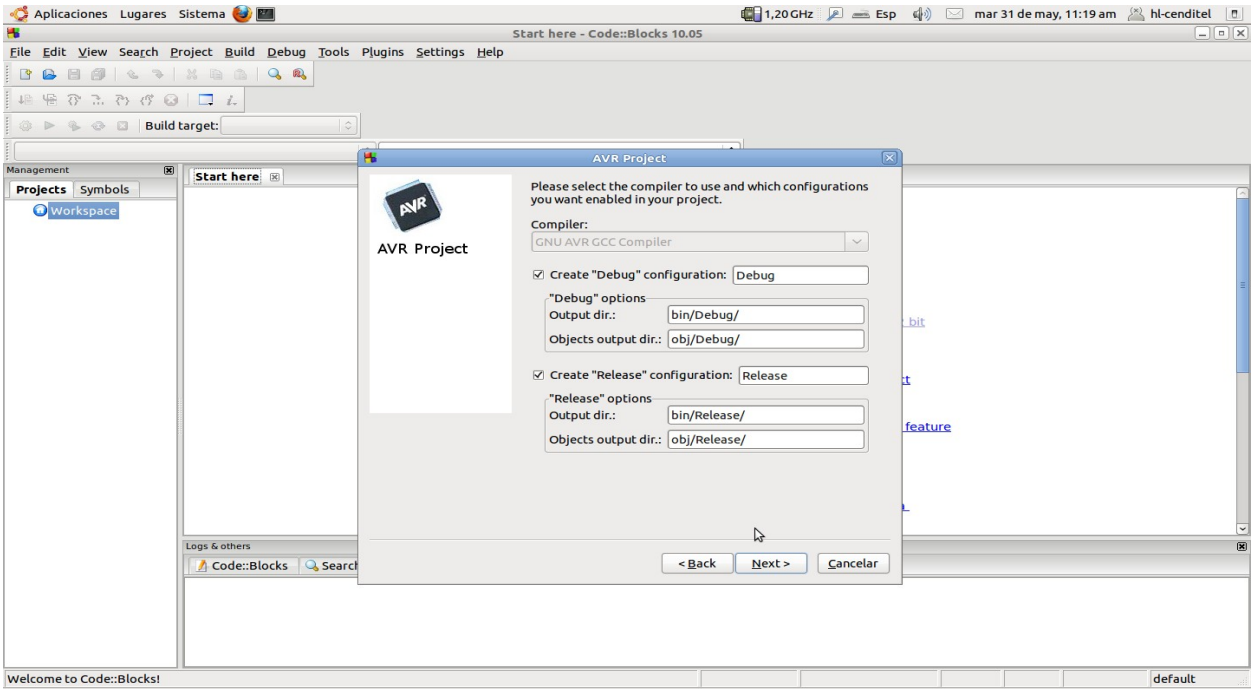

Figura 3. Selección del compilador y opciones

En la próxima pantalla se selecciona de la lista el microcontrolador que posea la tarjeta arduino que se va a programar. En este ejemplo se toma la tarjeta Arduino UNO como tarjeta objetivo, así que se selecciona el microcontrolador ATMega328p (figura 4). También se marca la opción “Run avr-size after build” ubicada al final de la ventana.

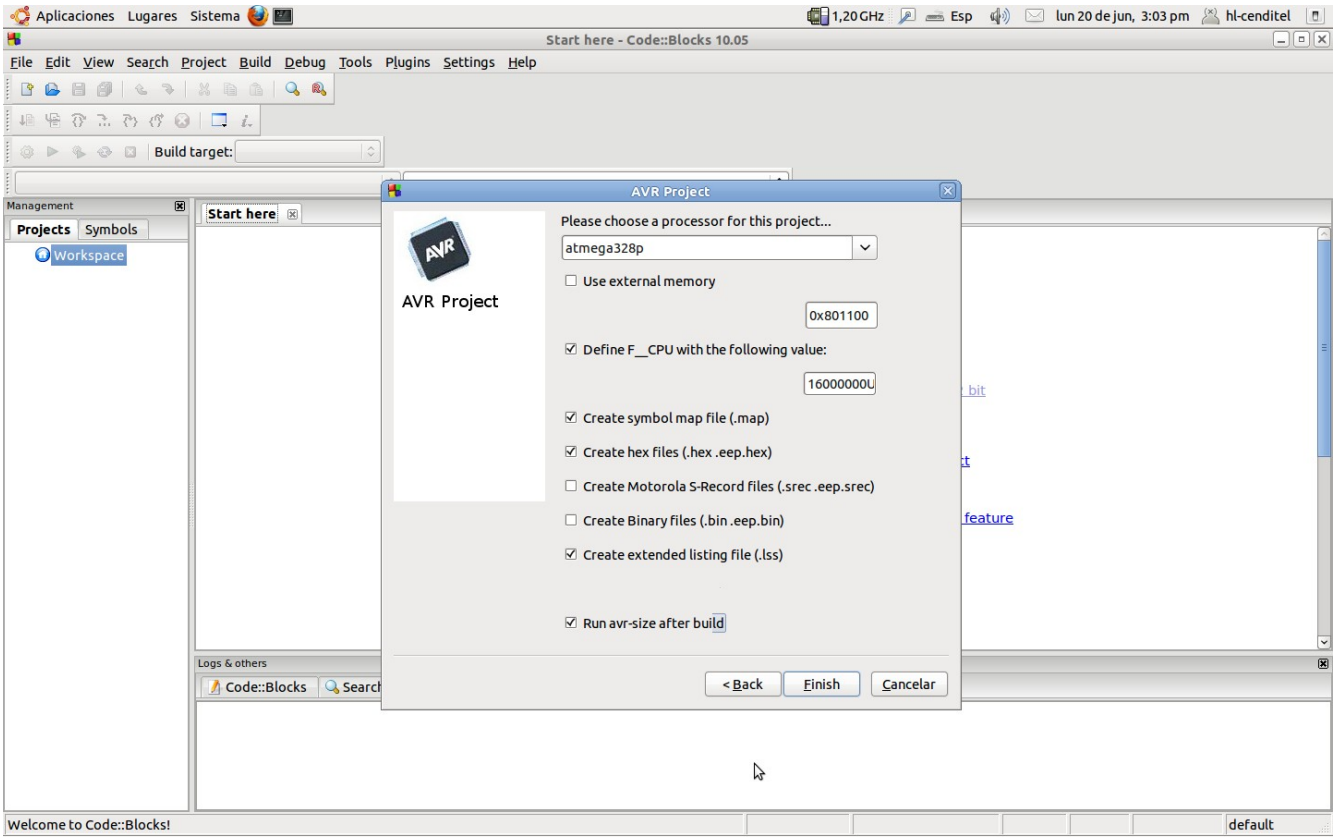

Figura 4. Selección del microcontrolador

Se puede notar que la frecuencia de trabajo predeterminada es de 16 Mhz, por lo que si se va a trabajar con un reloj externo a una frecuencia diferente se debe modificar este valor con el valor deseado, es decir, si se desea trabajar con un cristal a 8 Mhz, sustituimos 16000000UL por 8000000UL.

Esto es solamente necesario si se va a utilizar **un cristal externo a la tarjeta arduino**.

Con esto se culminan las configuraciones del proyecto, por lo que se puede escribir el código en lenguaje C en el archivo main.c que aparece en el árbol de

proyecto para su posterior compilación y generación del fichero hex.

Sin embargo, hay que destacar que, ya que no se está trabajando con las librerías del lenguaje de programación de Arduino, los pines en la tarjeta no corresponde con los nombres de los puertos del microcontrolador, así que se deben ubicar cuál es la correspondencia correcta entre éstos.

En la página oficial de Arduino (http://www.arduino.cc) se pueden ubicar diagramas con la información necesaria de los pines (figura 5). A continuación se presenta el mapping de los microcontroladores Atmega168 y Atmega328p, los cuales, según la página oficial, poseen un pin mapping prácticamente idéntico, y luego el pin mapping de los Atmega8:

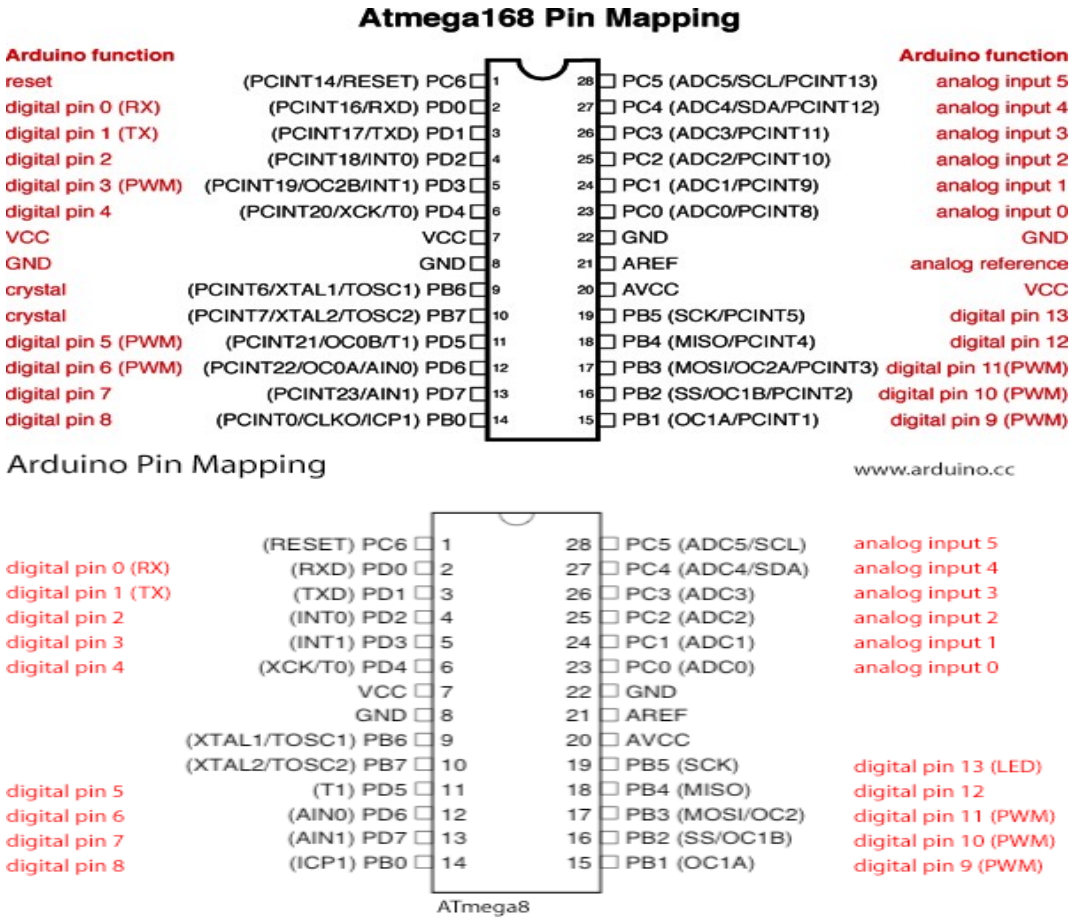

Figura 5. Pin mapping de los microcontroladores ATMega168/328p y ATMega8

Para comprobar que la programación de la tarjeta arduino se realiza de forma correcta, se arma un circuito sencillo y un programa de prueba. Para el

circuito, simplemente se conecta un led al pin 13 de la tarjeta arduino, y para el programa, se escribe el siguiente código en el main.c del proyecto creado en Code::Blocks:

```
#include <avr/io.h>
#include <util/delay.h>
/* Las librerias utilizadas en este codigo */

int main (void)
{
  DDRB = 0xFF; /* Puerto B como salida */

  while (1)
    {
     PORTB = 0xFF; /* Puerto B en alto, es decir, enciende led */
      _delay_ms(3000); /* Espera durante 3 segundos */
      PORTB = 0x00; /* Puerto B en bajo, es decir, apaga led */
      _delay_ms(1500); /* Espera durante 1,5 segundos */
    }
  return 1;
}
```

Una vez escrito, se procede a compilar el programa. Se puede notar que en el Code::Blocks la opción “Build Target” se encuentra seleccionada en “Debug” de manera predeterminada. Esta opción se utiliza mientras se comprueban y corrigen errores en el código, pero cuando el código se encuentre totalmente depurado, es necesario cambiar esta opción a “Release” para que se genere el fichero hex.

Al hacer clic en el botón de compilar con la opción “Build Target” en “Release”, se generará un fichero con el formato *nombre_del_proyecto.*elf.hex. Este archivo estará ubicado en la ruta donde fue creado el proyecto, dentro de la cual se encuentra una carpeta Bin, y dentro las carpetas Debug y Release; dentro de ésta última se puede ubicar el fichero .hex. Por ejemplo, si el nombre del proyecto es blink, ubicado en una carpeta “programas” la ruta sería:

`/programas/blink/bin/Release/blink.elf.hex`

Ya con el fichero .hex creado, debemos pasarlo al microcontrolador en la tarjeta arduino. Para esto se utiliza el programa de línea de comandos Avrdude.

Para hacer esto, es necesario conocer ciertas cosas:

1. El nombre del microcontrolador (para los atmega168 y atmega328p son m168 y m328p respectivamente).
2. El puerto al que está conectado mediante USB la tarjeta Arduino que se va a programar.
3. El programador hardware que se utiliza.
4. El nombre del archivo .hex que se va a subir al microcontrolador.

Respecto al microcontrolador de la tarjera arduino, las tarjetas Duemilanove y el nuevo modelo UNO poseen microcontroladores ATMega328p.

El puerto cambia de acuerdo a la tarjeta. Las primeras versiones de Arduino traen un convertidor USB-Serial del tipo FTDI, y Ubuntu suele reconocerlos como ttyUSB0. Por otra parte, las nuevas tarjetas Arduino vienen con un chip Atmel para la comunicación por USB y Ubuntu los reconoce como ttyACM0.

En cuanto al programador, Avrdude acepta el comando `arduino` como programador, el cual se basa en el programador stk500 versión 1.

Por último, el nombre del archivo depende del nombre del proyecto que se le haya asignado.

Una vez se conozcan estos datos, se abre una terminal y se ubica en la ruta donde está el fichero que a subir.

Una vez allí, se puede ingresar al Avrdude. Es necesario ser usuario root para realizar esta tarea, por lo que si su cuenta no posee permisos de superusuario, se agrega el comando `sudo` antes de las directivas para el Avrdude.

Para las primeras tarjetas Arduino (Duemilanove por ejemplo), las cuales vienen con el chip FTDI para la comunicación por USB, el comando para Avrdude toma la forma:

```
-p microcontrolador -P /dev/ttyUSB0 -c arduino -b 57600 -F -u -U flash:w:NombreDelArchivo.elf.hex
```

Se puede notar que se especifica un valor de 57600 para el baud rate o velocidad de transmisión.

Para las nuevas versiones de Arduino (UNO por ejemplo), las cuales vienen con el nuevo chip Atmel para la comunicación por USB, el comando toma la siguiente forma:

```
-p microcontrolador -P /dev/ttyACM0 -c arduino -F -u -U flash:w:NombreDelArchivo.elf.hex
```

En este caso, no es necesario especificar la velocidad de transmisión a utilizar.

Examinando un poco la estructura de estos comandos:

- -p se refiere al microcontrolador que se va a programar.
- -P se refiere al puerto donde está conectado el programador hardware, es decir, la tarjeta arduino por USB.
- -c se refiere al programador hardware utilizado, en este caso el mismo arduino.
- -b se refiere a la velocidad de transmisión a utilizar.

- -F deshabilita el chequeo de la firma del dispositivo.
- -u deshabilita el chequeo del Safemode de los bits Fuse.
- -U es el comando que indica la operación que debe realizar sobre la memoria en el microcontrolador. Este comando tiene una sintaxis de la siguiente forma: *tipo_de_memoria:operacion:nombre_del_archivo.*

Para conocer más acerca de los comandos del Avrdude y los parámetros admitidos por éstos, basta con leer la información de ayuda de esta herramienta, a la cual se accede con el siguiente comando:

```
man avrdude
```

Por ejemplo, para programar una Tarjeta Arduino UNO, la cual posee un microcontrolador ATMega328p, conectada al puerto ttyACM0 por USB, con el fichero blink.elf.hex, el comando sería:

```
sudo avrdude -p m328p -P /dev/ttyACM0 -c arduino -F -u -U flash:w:blink.elf.hex
```

Note que se agrega la directiva sudo al inicio del comando. Si ha salido todo bien, es decir, sin errores por parte del Avrdude, entonces se puede comprobar que la trasmisión del archivo .hex fue completada correctamente mediante la observación del funcionamiento del programa en el circuito de ejemplo, si el led conectado al pin 13 se enciende por 3 segundos y luego se apagaa por 1,5 segundos, de manera indefinida.

4. Utilizando las librerías de Arduino en Code::Blocks

Mientras que programar directamente en lenguaje C brinda mayor control de los dispositivos, utilizar las librerías del Arduino facilita notablemente la programación de las tarjetas, en especial para personas principiantes en el área,

debido a las librerías preempaquetadas que permiten un control sencillo de las funcionalidades de la tarjeta arduino.

Para utilizar estas librerías se puede usar el entorno de desarrollo integrado de Arduino o bien vincular dichas librerías al entorno en Code::Blocks. En esta guía se examina esta última opción.

Cabe destacar que las librerías del Arduino poseen una licencia GPL, por lo que es permitido su uso fuera del entorno de desarrollo del Arduino creado por Atmel.

Para utilizar las librerías de Arduino, se necesita disponer de la librería estática y los headers correspondientes a la librería.

Las librerías estáticas, o librerías-objetos, son colecciones de ficheros-objetos agrupados en un fichero, usualmente de extensión .a o .lib. Esta librería es creada para cada tipo de dispositivo, es decir, para cada modelo de microcontrolador se crea un librería estática específica. Por lo tanto, en primer lugar, se debe tomar en cuenta para cuál microcontrolador se va a programar, esto debido a que es necesario compilar la librería estática para ese dispositivo. En esta guía se toma como microcontrolador objetivo el ATMega328p, como en el ejemplo anterior.

La manera más rápida de conseguir la librería estática (core.a) es a través del mismo Arduino IDE. No es necesario tener la tarjeta Arduino conectada para realizar este proceso, sólo hace falta cerciorarse que la placa seleccionada en el IDE sea una de las tarjetas que posea el microcontrolador que se desear utilizar. Este procedimiento se debe realizar debido a que cada chip posee características y funcionalidades diferentes.

En el Arduino IDE, se selecciona Tools->Board->Arduino Duemilanove or Nano w/ Atmega328.

Ahora, es necesario compilar un programa (o sketch) para que se genere la librería estática. Para esto se puede utilizar cualquiera de los ejemplos del Arduino IDE. Se va al menú File->Examples->Digital->Blink y luego se procede a compilar este sketch.

Ahora se ubica a la ruta /tmp/buildXXXXXXXXXXXXXXXXXXX.tmp donde las XXXXXXXXXXXXXXXXXXXXXX es un número autogenerado para trazar la ruta del proyecto en cuestión. Dentro de esta carpeta se encuentra la librería estática core.a la cual se copia y guarda en una nueva carpeta en el lugar que se desee.

Para esto, se debe crear una carpeta con el nombre que deseemos para almacenar los archivos que debemos recopilar. Una vez creada, se abre una terminal, se ubica en el directorio donde está ubicado el fichero core.a deseado y se ejecuta el siguiente comando (suponiendo que se creó una carpeta `ArduinoLibraries` en el directorio `home`:

```
cp core.a /home/ArduinoLibraries
```

Hay que recordar que si se va a programar otro chip, es necesario volver a realizar este proceso, es decir, seleccionar una placa con el chip que se desea y recompilar el código para generar la librería compilada del microcontrolador en cuestión.

Los headers se pueden conseguir en la carpeta del Arduino, en la siguiente ruta:

```
arduino/Hardware/arduino/cores/arduino/
```

Dentro de esta carpeta, se seleccionan todos los archivos con la extensión .h y se copian y guardan en la misma carpeta donde anteriormente se

almacenó la librería core.a.

Ahora se procede a crear el proyecto en Code::Blocks de la misma manera que en el ejemplo anterior, pero antes de empezar a escribir el código en el main.c se debe indicar al entorno de desarrollo donde buscar las librerías para que puedan ser utilizadas en el proyecto. Esto se hace definiendo en el IDE el linker con la ruta donde se ubicaron las librería Arduino y así pueda encontrar y ejecutar las llamadas a funciones.

En el proyecto, se va al menú Project->Build Options y luego se hace clic en la pestaña Linker Settings. Aquí se presiona el botón Add y se indica la ruta de la carpeta creada con las librerías y se ubica el archivo **core.a** que es el que necesitamos vincular al Linker. Si al hacer esto, aparece un mensaje preguntando si se quiere mantener esta ruta relativa, seleccionamos la opción No. El lugar donde se tengan almacenados estos ficheros no es importante, siempre y cuando le indiquemos la ruta completa en este Linker.

Ahora es necesario agregar los headers al proyecto. Para esto se va al menú Project->Add Files. Se ubica la carpeta donde están los headers, se seleccionan todos y se hace clic en Aceptar. Ya se puede proceder a escribir el código en el fichero main.c del proyecto. Cabe recordar que se debe codificar en el lenguaje de Arduino y no en C puro.

Como ejemplo se muestra el siguiente código cuya función es encender un led conectado al pin 9 de la tarjeta arduino durante 1 segundo y luego apagarlo por 2 segundos.

```
void __cxa_pure_virtual(void) {};
/* linea para evitar errores del linker */

#include "../ArduinoLibrary/WProgram.h"
/* inclusion del header principal indicando la ruta relativa a la carpeta del proyecto */
```

```
int main(){
    init(); // Siempre se debe colocar al inicio del main
    pinMode(9,OUTPUT); // pin 9 como salida

      while(1){
    digitalWrite(9,HIGH); // pin 9 en alto, led encendido
    delay(1000); // esperar 1 segundo
    digitalWrite(9,LOW); // pin 9 en bajo, led apagado
    delay(2000); // esperar 2 segundos
    }
}
```

Cuando se trabaja con el Arduino IDE, se divide el código en dos partes: `void setup()` y `void loop()`. En este ejemplo, en `int main()` se tiene lo que normalmente se colocaría en `void setup()` mientras que lo que usualmente se codifica en el `void loop()` se hace con un `while(1)`.

Este `while(1)` se va a ejecutar repetidamente, exactamente como el `loop()`, así que no se debe colocar `return` dentro de este lazo.

La función `init()` también es de vital importancia colocarla antes de hacer uso de cualquier de las funciones de la librería de Arduino. Esta función se encarga de inicializar todas las funciones relacionadas con temporizadores, como por ejemplo la función `delay()`.

Continuando con el proyecto, se procede a compilar el código de la misma forma que se hizo en el proyecto anterior. Hay que recordar que para que se genere el fichero .hex es necesario compilar con la opción “Release” seleccionada.

Ya creado el fichero .hex se repite el procedimiento para subir el código a través de una terminal con Avrdude. En este ejemplo el proyecto fue nombrado `arduino`, y se programa una tarjeta Arduino UNO, por lo que el comando utilizado

es el siguiente:

```
avrdude -p m328p -P /dev/ttyACM0 -c arduino -F -u -U flash:w:arduino.elf.hex
```

Se espera a que Avrdude indique que la transferencia se realizó con éxito y se puede comprobar la programación correcta de la tarjeta al ver el funcionamiento del programa encendiendo y apagando el led en el pin 9.