Seminario 2

Configuración e instalación de las herramientas de desarrollo

Introducción

A la hora de desarrollar una aplicación en el ámbito de la computación doméstica, lo normal es utilizar la misma plataforma en la que posteriormente se va a ejecutar, entendiendo por plataforma el conjunto de hardware más software de sistema. Por ejemplo, para desarrollar una aplicación que se ejecute en un procesador con arquitectura IA-32 y sistema operativo *Linux*, se instala la cadena de herramientas de desarrollo de GNU (GNU *toolchain*) en el mismo computador y se procede a su desarrollo.

Sin embargo, esta opción no es nada común en el desarrollo del software que se ejecutará en un sistema empotrado. Los sistemas empotrados se diseñan con unas características tan ajustadas a la función que van a desarrollar, que no tienen por qué tener la potencia o memoria necesarias para construir un programa, o puede que no dispongan de teclado o pantalla. Es más, seguramente no exista un compilador que se pueda ejecutar en el sistema empotrado. Por tanto, la opción más habitual para desarrollar el software empotrado consiste en usar una plataforma de desarrollo cruzado. El desarrollo se realiza en un computador anfitrión (*host*), normalmente un computador personal o una estación de trabajo, y el resultado final será un programa ejecutable en la plataforma de destino (*target*), el sistema empotrado.

Normalmente, la cadena de herramientas de desarrollo cruzado está formada por un compilador que genera ficheros ejecutables en la plataforma remota, una serie de utilidades para el manejo de estos ficheros binarios (ensamblador, enlazador, etc.), y por un depurador para comprobar que el código se ejecuta correctamente en el sistema empotrado. En algunos casos también será necesario el uso de un servidor de depuración, que hace posible la conexión entre el depurador (que se ejecuta en el PC) y el programa a depurar (que se ejecuta en el sistema empotrado). Se denomina *"cadena de herramientas"* (*toolchain*) porque normalmente las herramientas de desarrollo que la componen se usan en cadena. Estas herramientas se deben seleccionar según sea la arquitectura del procesador de nuestro sistema empotrado. Además, también es bastante útil (aunque no es fundamental) el uso de un Entorno de Desarrollo Integrado (IDE) que nos permita acceder a todas estas herramientas de una forma cómoda e intuitiva.

Teniendo todo esto en cuenta, en este seminario se construirá una cadena de herramientas de desarrollo para que genere código para el procesador de la plataforma de prácticas y se configurará el entorno de desarrollo *Eclipse* para hacer uso de dichas herramientas.

Objetivos

Los objetivos que se persiguen con el desarrollo de este seminario son:

- Entender la necesidad del desarrollo cruzado de aplicaciones para un sistema empotrado.
- Saber cómo instalar una cadena de herramientas de desarrollo cruzado.
- Configurar el IDE *Eclipse* para hacer uso de las herramientas.
- Familiarizarse con las herramientas de desarrollo.

Selección de la cadena de herramientas de desarrollo

El primer paso para seleccionar una cadena de herramientas de desarrollo cruzado consiste en identificar el procesador (o procesadores) para los que se va a desarrollar. Como a lo largo de las prácticas vamos a usar como plataforma de destino la placa de desarrollo *Redwire Econotag*[1], este primer paso consiste en determinar el procesador en el que está basada la consola, el ARM7TDMI.

La mayoría de los fabricantes de procesadores tienen acuerdos con compañías que venden cadenas de desarrollo propietarias para sus procesadores. Sin embargo, si optamos por herramientas propietarias tenemos un par de inconvenientes. Por un lado está su alto coste, y por otro la dificultad que plantea el aprendizaje de unas nuevas herramientas de desarrollo.

Lo ideal sería usar siempre la misma cadena de herramientas de desarrollo para diferentes proyectos, de forma que aunque cambie el procesador empotrado, no se pierda tiempo en aprender a utilizar nuevas herramientas. Por tanto, puestos a escoger una cadena de herramientas de desarrollo, nos interesará que sea multi-plataforma. Además de esta característica, si andamos cortos de presupuesto, nos interesará que su precio sea bajo, y por último, también es deseable que la cadena de desarrollo genere código eficiente para la plataforma de destino, ya que normalmente, la memoria y la potencia del procesador de un sistema empotrado están muy ajustados.

La cadena de herramientas de desarrollo de GNU cumple todas estas características, ya que es gratuita, genera un código de gran calidad, y como se dispone de su código fuente, se puede construir para que genere código cruzado para multitud de plataformas, entre las que está la arquitectura ARM[2]. Por tanto, y teniendo en cuenta estas ventajas, usaremos esta cadena de herramientas para el desarrollo de las prácticas.

Componentes de la cadena de desarrollo

La cadena de herramientas de desarrollo que vamos a construir está compuesta por la colección de compiladores *gcc*, las utilidades de manipulación de ficheros binarios *binutils*, la biblioteca de funciones C *newlib*, el depurador *gdb*, y el servidor de depuración *OpenOCD*. Además, también configuraremos el entorno de desarrollo *Eclipse* para que pueda hacer uso de estas herramientas.

Los compiladores *gcc*

Realmente, *gcc* no es un único compilador, sino una colección de compiladores de diferentes lenguajes, como *C*, *C++*, *Java*, *Fortran, Ada,* etc., para diferentes plataformas (*IA-32*, *Alpha, ARM, MIPS, PowerPC, Sparc,* etc.)[3].

La mayoría de los programadores están acostumbrados al uso de algunos compiladores de esta colección (sobre todo los de C, C++ y Java) previamente construidos para generar código para la arquitectura del PC, ya que las distribuciones de *Linux* los incorporan como sus compiladores por defecto. Sin embargo, si en vez de instalar sólo los binarios de algunos de los compiladores de GNU, se instala el código fuente de *gcc*, se pueden construir compiladores de los lenguajes que queramos para multitud de plataformas, simplemente cambiando los parámetros de construcción de gcc.

1 http://www.redwirellc.com
2 http://gcc.gnu.org/install/specific.html
3 *gcc* es el acrónimo de GNU *Compiler Collection*.

Las *binutils*

Son una colección de utilidades para manejar ficheros binarios. Aunque las más importantes son `ld` y `as`, el enlazador y el ensamblador de GNU, hay otras bastante útiles:

- `addr2line` - Convierte las direcciones de los símbolos de un fichero objeto al nombre del fichero fuente y número de línea
- `ar` - Utilidad para crear bibliotecas de funciones
- `gprof` - Muestra información útil sobre la ejecución de un programa para ayudar a su optimización
- `nm` - Genera una lista con los símbolos de un fichero objeto
- `objcopy` - Copia y cambia el formato de ficheros objeto
- `objdump` - Muestra la información de un fichero objeto
- `ranlib` - Genera un índice de los archivos contenidos en una biblioteca de funciones
- `readelf` - Muestra información de cualquier fichero en formato ELF
- `size` - Muestra el tamaño de las secciones de un fichero
- `strings` - Lista las cadenas imprimibles de un fichero
- `strip` - Descarta los símbolos de un fichero

El depurador *gdb*

Permite depurar el código que desarrollemos: consultar la memoria, inspeccionar el valor de las variables y registros, poner puntos de ruptura, etc. La única diferencia con respecto al uso que se hace tradicionalmente de *gdb* es que al hacer desarrollo cruzado, *gdb* se ejecutará en nuestro PC mientras que el programa que estemos depurando se ejecutará en el sistema empotrado. Por tanto, para facilitar la depuración será necesario conectar físicamente el sistema empotrado al PC (vía JTAG), y establecer un protocolo de depuración común (mediante *OpenOCD*).

La biblioteca de funciones C *newlib*

Es una implementación reducida de la biblioteca de funciones estándar de C, como las funciones de tratamiento de cadenas, de ficheros, de memoria, etc., junto con la biblioteca de funciones matemáticas, destinada a su uso en sistemas empotrados. Está mantenida por los desarrolladores de *RedHat* Jeff Johnston y Tom Fitzsimmons y se puede integrar dentro de gcc.

El servidor de depuración *OpenOCD*

Es un demonio capaz de comunicarse con la mayoría de procesadores de ARM que soporta múltiples tipos de interfaces JTAG de diferentes fabricantes. Está concebido para abstraer los detalles de la conexión entre el PC de desarrollo y el sistema empotrado y ofrecer a *gdb* un canal de comunicación estándar (TCP) y un protocolo de depuración único.

Instalación de las herramientas de desarrollo desde los repositorios

En el caso de que se esté utilizando un sistema operativo basado en *Linux*, se pueden obtener las herramientas de desarrollo directamente desde los repositorios. Para distribuciones basadas en *Debian/Ubuntu*, es necesario instalar los siguientes paquetes:

- `binutils-arm-none-eabi`
- `gcc-arm-none-eabi`
- `libnewlib-arm-none-eabi`
- `gdb-arm-none-eabi`
- `openocd`

Para distribuciones de *Linux* basadas en *RedHat/Fedora*, será necesario instalar los siguientes paquetes:

- `arm-none-eabi-binutils-cs`
- `arm-none-eabi-gcc-cs`
- `arm-none-eabi-newlib`
- `arm-none-eabi-gdb`
- `openocd`

Construcción de la cadena de herramientas de desarrollo

Otra posibilidad para tener las herramientas de desarrollo instaladas en un PC es la construcción de las mismas a partir de sus fuentes. El proceso de construcción es lento y complejo, pero es interesante intentarlo, ya que permite obtener las herramientas en aquellas plataformas en las que no existen repositorios. Aunque las herramientas de desarrollo de GNU se pueden construir tanto en un PC bajo *Linux* como bajo *Windows* o *Mac OSX*, dado que en los laboratorios de la Escuela se usarán PC con *Linux*, en este seminario sólo se describirá la instalación bajo *Linux*.

Prerrequisitos

Antes de poder construir las herramientas de desarrollo, es necesario tener instaladas en el computador varias utilidades y bibliotecas. En el caso de que se use *Linux*, los paquetes necesarios varían según la distribución instalada. Para distribuciones basadas en *Debian/Ubuntu*, es necesario instalar los siguientes paquetes:

- Para descargar las fuentes: `wget`
- Para construir la *toolchain*: `build-essential texinfo bison flex libncurses5-dev libgmp3-dev libmpfr-dev libmpc-dev zlib1g-dev`
- Para construir *OpenOCD:* `libtool pkg-config autoconf libusb-1.0-0-dev libftdi-dev`

Para distribuciones de *Linux* basadas en *RedHat/Fedora*, será necesario instalar los siguientes paquetes:

- Para descargar las fuentes: `wget`
- Para construir la *toolchain*: `texinfo bison flex ncurses-devel gmp-devel mpfr-devel libmpc-devel zlib-devel`

- Para construir *OpenOCD*: `libtool pkg-config autoconf texinfo libusb-devel libftdi-devel`

Construcción de las herramientas

Normalmente, la construcción de cualquier programa *open source* se realiza siempre de la misma forma, siguiendo los siguientes pasos:

1. *Descargar las fuentes:* Para ello se puede usar la herramienta `wget`.
2. *Descomprimir las fuentes:* Normalmente hace falta la herramienta `tar` y alguna utilidad de compresión como `zip` o `bzip2`.
3. Configurar el programa: mediante el *script* de configuración `configure` y asignando valores a *flags* se crea un *Makefile* a medida de nuestras necesidades.
4. Construcción: Una vez creado el *Makefile*, normalmente sólo hay que invocar a `make`.
5. Instalación: Una vez construido el programa se instala mediante `make install`.

Un aspecto muy importante a tener en cuenta es que no todas las versiones de cada una de las herramientas funcionan correctamente con otras versiones de otras herramientas. Por lo tanto, y aunque haya versiones más actualizadas de las que se usan en este guión, se recomienda ceñirse a las versiones que se recomiendan aquí, ya que la integración y buen funcionamiento de otras versiones no está garantizada.

Otro aspecto importante es que para construir correctamente algunas herramientas es necesario descomprimir las fuentes en un directorio y construirlas en otro directorio diferente. También es interesante guardar las fuentes comprimidas por si es necesario volver a construir alguna herramienta. Por tanto, se recomienda crear la siguiente estructura de directorios:

```
Raíz de la instalación   → Puede ser $HOME/se/toolchain, o cualquier otro directorio
      |-- downloads      → Subdirectorio para descargar las fuentes comprimidas
      |-- build          → Subdirectorio para la construcción
```

Construcción e instalación de las *binutils*

Dado que las *binutils* contienen el ensamblador y el enlazador para la plataforma remota, es necesario construirlas al principio, antes de todas las demás herramientas, ya que necesitaremos el enlazador y el ensamblador cruzados para construir el resto de las herramientas.

Podemos descargarnos las fuentes de las *binutils* de http://ftp.gnu.org/gnu/binutils. Para nuestra *toolchain* usaremos la versión 2.24. Una vez descargadas (en el directorio `downloads`), las descomprimiremos en el directorio `build`, lo que nos generará el directorio `build/binutils-2.24`.

La construcción de las *binutils* debe hacerse en un directorio diferente al que contiene las fuentes, por lo que crearemos el directorio `build/binutils-build` para ello. Una vez dentro de `build/binutils-build`, invocaremos al *script* de configuración de las *binutils* (`../binutils-2.24/configure`) con los siguientes *flags*:

- `--prefix="DIR"` → siendo "`DIR`" el directorio raíz de la instalación (por ejemplo `$HOME/se/toolchain`) . Si se omite este *flag*, el directorio de instalación por defecto será `/usr/local`.

- `--target=arm-econotag-eabi` → lo que indica al configurador el procesador y la arquitectura para la que queremos construir las herramientas.
- `--enable-interwork --enable-multilib` → Para tener soporte tanto de los modos ARM como *Thumb*.
- `--with-float=soft` → Dado que el procesador de la *Econotag* no dispone de FPU.
- `--disable-werror` → Para evitar que los *warnings* causen error.

Tras la ejecución de este *script* se habrá generado un *Makefile* en `build/binutils-build`, de forma que ya se pueden construir e instalar las *binutils* mediante la orden `make all install`.

Como hemos dicho antes, las *binutils* serán necesarias para la posterior creación del resto de herramientas, por tanto es necesario modificar la variable de entorno `PATH` para añadirle el directorio en el que se han instalado (`$HOME/se/toolchain/bin`), de forma que el sistema pueda encontrarlas cuando sea necesario. Si queremos que la variable `PATH` quede modificada permanentemente, debemos indicarlo en nuestro fichero `.bashrc`. Por otro lado, para poder consultar las páginas del manual de las herramientas de nuestra *toolchain* deberíamos añadir el directorio `$HOME/se/toolchain/man` a la variable de entorno `MANPATH` (también en nuestro `.bashrc`) para que la utilidad `man` pueda encontrarlas.

Construcción del compilador: 1ª etapa

El compilador de C se debe construir en dos etapas. En la primera se construye un compilador mínimo sin soporte de la biblioteca *libC*. Aunque este compilador no es suficiente para construir un programa típico, sí que servirá para compilar la biblioteca *newlib*, que posteriormente se integrará dentro del compilador en la segunda etapa de construcción.

Podemos descargarnos las fuentes de *gcc* de http://ftp.gnu.org/gnu/gcc. Para nuestra *toolchain* usaremos la versión 4.9.1. Una vez descargadas (en el directorio `downloads`), las descomprimiremos en el directorio `build`, lo que nos generará el directorio `build/gcc-4.9.1`.

La construcción de *gcc* también debe hacerse en un directorio diferente al que contiene las fuentes, por lo que crearemos el directorio `build/gcc-build` para ello. Una vez dentro de `build/gcc-build`, invocaremos al *script* de configuración de *gcc* (`../gcc-4.9.1/configure`) con los *flags* que usamos para construir las *binutils*, a los que añadiremos los siguientes:

- `--with-system-zlib` → Para que use la versión de la biblioteca *zlib* de nuestro sistema en vez de la que viene con las fuentes de *gcc*.
- `--enable-languages="c,c++"` → De todos los lenguajes soportados (Ada, C#, …) sólo nos interesan los compiladores de C y C++.
- `--with-newlib` → Usaremos *Newlib* como biblioteca *libC*.
- `--with-headers=$HOME/se/toolchain/build/newlib-2.1.0/newlib/libc/include` → Ubicación de las cabeceras de *Newlib* una vez que descomprimamos las fuentes.

Tras la ejecución de este *script* se habrá generado un *Makefile* en `build/gcc-build`, de forma que para construir e instalar la versión *bootstrap* de gcc invocaremos la orden `make all-gcc install-gcc`.

Construcción de *newlib*

Con el compilador creado en el paso anterior se puede construir la biblioteca de C que se integrará finalmente en el compilador más adelante. De entre todas las opciones disponibles para la biblioteca *libC* hemos escogido *Newlib*, que se puede descargar de ftp://sources.redhat.com/pub/newlib. Para nuestra *toolchain* usaremos la versión 2.1.0, que descargaremos en el directorio `downloads` y descomprimiremos en el directorio `build`, generando el directorio `newlib-2.1.0`.

Al igual que las herramientas anteriores, *Newlib* también debe construirse en un directorio diferente del que alberga sus fuentes, por lo que crearemos el directorio `build/newlib-build`, y una vez dentro de este directorio, configuraremos *Newlib* con las mismas opciones que usamos para construir las *binutils*, añadiendo la opción `–disable-newlib-supplied-syscalls`. Esta opción es muy importante, ya que por defecto, las llamadas a sistema de *Newlib* se apoyan en el monitor de depuración *Angel* que solían llevar las placas de desarrollo de ARM. Como en nuestra plataforma de desarrollo no disponemos de dicho monitor, tendremos que implementar las llamadas a sistema desde cero, por lo que hay que indicarlo explícitamente en el proceso de construcción de la biblioteca. Una vez configurada, podemos construir e instalar la biblioteca mediante la orden `make all install`.

Construcción del compilador: 2ª etapa

Una vez construida la biblioteca estándar de C ya podemos integrarla dentro del compilador para terminar su construcción. Como en la configuración ya dejamos indicado que se usaría Newlib y el directorio que contiene las cabeceras, ahora sólo hay que cambiarse al directorio `build/gcc-build` y terminar la construcción mediante la orden `make all install`.

Construcción e instalación de depurador

Podemos descargarnos las fuentes del depurador *gdb* de http://ftp.gnu.org/gnu/gdb. Para nuestra *toolchain* usaremos la versión 7.8. Una vez descargadas (en el directorio `downloads`), las descomprimiremos en el directorio `build`, lo que nos generará el directorio `build/gdb-7.8`.

Al igual que las herramientas anteriores, la construcción de *gdb* debe hacerse en un directorio diferente al que contiene las fuentes, por lo que crearemos el directorio `build/gdb-build`, y desde dentro de dicho directorio, invocaremos al *script* de configuración del depurador con las mismas opciones que usamos para construir las *binutils*. Una vez obtenido el *Makefile*, ejecutaremos la orden `make all install` para construirlo e instalarlo.

Construcción e instalación de *OpenOCD*

Podemos descargarnos las fuentes de http://sourceforge.net/projects/openocd/files/openocd. Para nuestra *toolchain* usaremos la versión 0.9.0. Una vez descargadas (en el directorio `downloads`), las descomprimiremos en el directorio `build`, lo que nos generará el directorio `build/openocd-0.9.0`. *OpenOCD* sí que se puede construir en el mismo directorio que sus fuentes, por lo que para construirlo nos cambiaremos a `build/openocd-0.9.0` y ejecutaremos su *script* de configuración con la siguientes opciones:

- `--prefix="DIR"` → siendo "`DIR`" el directorio raíz de la instalación (por ejemplo `$HOME/se/toolchain`) . Si se omite este *flag*, el directorio de instalación por defecto será `/usr/local`.
- `--enable-libftdi` → La placa *Econotag* usa un chip FTDI para sacar los puertos serie y el

puerto JTAG del procesador principal a un USB, que es el que finalmente se conectará al PC de desarrollo.

- `--disable-werror` → Para evitar que los *warnings* causen error.

Una vez obtenido el *Makefile*, ejecutaremos la orden `make all install` para construirlo e instalarlo.

Configuración de la placa Econotag en Linux

Es necesario que añadamos una regla para el servicio *udev* de *Linux* que permita que los usuarios que no tienen privilegios de *root* puedan acceder a la *Econotag* a través del puerto USB. Para ello, debemos crear un fichero con el nombre `99-ftdi.rules` en el directorio `/etc/udev/rules.d` con el siguiente contenido:

```
# Regla para permitir a los usuarios acceder a la Econotag
SUBSYSTEMS=="usb", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="6010", MODE:="666"
```

Una vez creada la regla, debemos reiniciar el servicio *udev* para que haga uso de esta nueva regla. Para ello, ejecutaremos la orden `service udev restart`.

Prueba de las herramientas

Antes de seguir es conveniente comprobar que las herramientas se han instalado correctamente. Para ello usaremos la versión del *Hola Mundo* para la *Econotag*, cuyo código está accesible en la carpeta `/fenix/depar/atc/se/pracs/hello` desde los ordenadores del aula de prácticas. Para ello copiaremos el contenido de la carpeta `hello` a nuestro PC, por ejemplo a la carpeta `$HOME/se/pracs/hello`.

El *Makefile* de este ejemplo está preparado para usar la *toolchain* instalada desde los repositorios. Por tanto, asume que las herramientas están instaladas en el directorio `/usr` y que tienen el prefijo `arm-none-eabi`. En el caso de que se haya construido la *toolchain* a partir de las fuentes será necesario cambiar las variables `TOOLS_PATH` y `TOOLS_PREFIX` del *Makefile* por los valores correctos, por ejemplo `TOOLS_PATH` se debería cambiar por `$(HOME)/se/toolchain/bin` (o la ruta correspondiente, si es que se ha instalado la *toolchain* en otro directorio) y `TOOLS_PREFIX` por `arm-econotag-eabi`.

La construcción del programa se realiza con `make`. Si las herramientas están bien instaladas (y el *Makefile* ha sido modificado correctamente), se deberían generar los ficheros `hello.elf` y `hello.bin`, que son, respectivamente, el ELF y la imagen de nuestro *hola mundo* para la *Econotag*.

Para comprobar que *OpenOCD* se ha instalado correctamente, probaremos a subir la imagen del programa a la placa y ejecutarla. Para ello, conectaremos la placa al puerto USB del PC y escribiremos `make run`, lo que causará que se inicie el demonio de *OpenOCD*, al que se le pedirá que suba la imagen a la placa y la ejecute. Una vez terminado este proceso, deberíamos ver el led rojo de la placa parpadeando, lo que indicará que todo ha funcionado correctamente. Para detener la ejecución del programa podemos escribir `make halt` en la consola.

Instalación y configuración de Eclipse

A continuación se describe cómo instalar y configurar adecuadamente el IDE *Eclipse* para el

desarrollo de las prácticas de la asignatura.

Prerrequisitos

Dado que *Eclipse* está desarrollado en *Java*, lo primero es asegurarnos de que tenemos una máquina virtual de *Java* instalada. Para ello podemos escribir la orden:

```
java -version
```

En caso de que no esté instalada, podemos instalarla mediante el paquete `default-jre` para distribuciones basadas en *Debian*, o `java-1.8.0-openjdk` para distribuciones basadas en *Fedora*.

Instalación de *Eclipse*

Una vez que tenemos la máquina virtual de *Java* instalada, el siguiente paso es bajarnos el *Eclipse IDE for C/C++ Developers*[4] de http://www.eclipse.org/downloads y descomprimirlo en el directorio en el que queramos (ej. `/opt`, `/usr/local`, `$HOME/se`, …). Si ya teníamos previamente una instalación de *Eclipse* en nuestro PC, es necesario comprobar que tiene instalado el *pluging* CDT (*C/C++ Development Tooling*). Para ello podemos consultarlo en el menú *Help > Installation Details*. En el caso de que nuestra instalación de *Eclipse* no tenga dicho *pluging*, se puede añadir en el menú *Help > Install New Software …*. Una vez en la ventana de instalación de nuevo software simplemente hay que añadir el repositorio pulsando el botón *Add ...* y especificar la URL del repositorio de la versión adecuada del CDT según la versión de nuestra instalación de *Eclipse*. Para obtener esta información podemos consultar la página *Web* del CDT[5].

Hecho esto, ya podemos lanzar el IDE, y cuando nos pregunte por el directorio que usaremos como *workspace*, seleccionaremos `$HOME/se/pracs`, que es el directorio en el que tenemos la carpeta `hello` con nuestro *Hola Mundo*.

Instalación del soporte de desarrollo cruzado de *Eclipse*

Aunque ya tenemos *Eclipse* preparado para desarrollar en *C/C++*, esta versión no trae por defecto el soporte para depuración remota de aplicaciones en sistemas empotrados. Para instalarlo nos iremos al menú *Window > Preferences* y, dentro de la opción *Install/Update > Available Software Sites*, habilitaremos el repositorio del CDT, que por defecto está desactivado. Una vez habilitado dicho repositorio, nos iremos al menú *Help > Install New Software* y lo seleccionaremos en la lista desplegable *Work with*. Cuando nos aparezcan más abajo los paquetes que incluye el repositorio del CDT, instalaremos el paquete *CDT Optional Features > C/C++ GDB Hardware Debugging*.

Creación de un proyecto

Para crear un proyecto nuevo nos iremos al menú *File > New > C Project*. Llamaremos al proyecto `hello`, para que coincida con el directorio que contiene el código de nuestro *Hola Mundo*. Como tipo de proyecto seleccionaremos *Makefile project > Empty Project* y como *toolchain* seleccionaremos -- *Other Toolchain* --. Pulsando el botón *Finish* se creará un proyecto que contendrá nuestro *Hola Mundo*.

Una vez creado el proyecto, hay que configurarlo. Para ello, pulsaremos en el menú *Project* y

4 En el momento de la redacción de este guión se ha usado la versión *Luna* (4.4.1) de *Eclipse*.
5 https://eclipse.org/cdt/downloads.php

desactivaremos la opción *Build Automatically*. Después, nos iremos al menú *Project > Properties > C/C++ Build > Settings > Binary Parsers* y nos aseguraremos de que sólo está marcado el *GNU Elf Parser*.

Gestión del *Makefile*

Pulsando con el botón derecho sobre el proyecto en el panel *Project Explorer* podremos limpiar el proyecto con la opción *Clean Project* y construirlo con la opción *Build Project*. La salida del *Makefile* se puede observar en la pestaña *Console* del panel inferior del IDE.

Por otro lado, en el panel derecho hay una pestaña llamada *Build Targets* en el que se pueden crear accesos directos a cualquier regla del *Makefile*.

Configuración del depurador

Antes de configurar las opciones de depuración de un proyecto debemos asegurarnos de que está correctamente construido. Para depurar, primero necesitamos abrir la perspectiva *Debug* (menú *Window > Perspective > Open Perspective > Debug*). Después, pasamos a configurar el depurador. Para ello, crearemos una nueva configuración de depuración para nuestro proyecto pinchando en el menú desplegable del icono del insecto y seleccionando la opción *Debug Configurations*. Una vez abierta la ventana de configuración, seleccionaremos una configuración del tipo *GDB Hardware Debugging* y pulsaremos sobre el botón *New*.

Ahora pasamos a completar la configuración de depuración. Fijaremos el nombre de la configuración a *hello*, para que coincida con el nombre del proyecto. En la pestaña *Main* nos aseguraremos de que la configuración de depuración esté asociada al proyecto *hello*, y de seleccionar el fichero ELF que contiene nuestra aplicación mediante el botón *Search Project* del campo *C/C++ Application*.

En la pestaña *Debugger* debemos cambiar el campo *GDB Command* por `arm-none-eabi-gdb` o `arm-econotag-eabi-gdb`, según la *toolchain* que estemos usando, y en el caso de que estemos usando la *toolchain* construida a partir de las fuentes, si no hemos exportado la ruta de la carpeta `bin` de nuestra *toolchain* a la variable de entorno `PATH` en el fichero `.bashrc`, deberemos especificar la ruta completa a nuestro depurador. Por otro lado, debemos indicar que la conexión a la plataforma se hará mediante *OpenOCD*. Para ello, en la configuración de la plataforma remota (*Remote Target*) debemos marcar la opción *Use Remote Target*, indicar que la conexión al dispositivo JTAG se hará mediante el protocolo TCP/IP, que la dirección IP del servidor de depuración es *localhost* (ya que *OpenOCD* corre en nuestro PC), y que el puerto es el 3333, que es el puerto de depuración por defecto de *OpenOCD*.

En la pestaña *Startup* debemos indicarle al *debugger* cómo inicializar el proceso de depuración. En la parte de *Initialization Commands* activaremos la casilla *Reset and Delay* y le asignaremos un valor de 3 segundos y desactivaremos la casilla *Halt*. Adicionalmente indicaremos las siguientes órdenes a *OpenOCD*:

```
monitor soft_reset_halt
set *0x80020010 = 0
```

La primera para hacer un *soft reset* de la placa y la segunda para desactivar el controlador de interrupciones.

En la parte *Load Image and Symbols* indicaremos que se suban a la *Econotag* tanto la imagen de la

aplicación como sus símbolos, y en la parte *Runtime Options* indicaremos que se fije el contador de programa a la dirección `400000`, que es la dirección del mapa de memoria en la que está mapeada la RAM, y que se fije un punto de ruptura en la etiqueta `_start,` que el el punto de entrada de nuestro ejecutable.

Depuración

Para depurar nuestro programa, previamente tendremos que conectar la *Econotag* a un puerto USB del PC y lanzar el demonio de depuración de *OpenOCD* (mediante la regla `openocd` de nuestro *Makefile* en una consola aparte). Posteriormente, y ya en el IDE de *Eclipse,* abriremos la perspectiva de depuración, seleccionaremos nuestra configuración de depuración *hello,* y pulsaremos el botón *Debug*. A partir de aquí, ya podremos inspeccionar registros, variables, ejecutar paso a paso, fijar puntos de ruptura, etc., para supervisar la ejecución de nuestro programa en la *Econotag*.

Desarrollo de las prácticas en la ETSIIT

En las aulas están instaladas tanto la *toolchain* de los repositorios como la construida a partir de las fuentes. Ésta última está instalada en el directorio `/fenix/depar/atc/se/toolchain`. Para poder usarla correctamente es necesario arrancar los PC de prácticas con la versión de 32 bits del sistema operativo *Ubuntu 16.04*, ya que esta es la única imagen que tiene la regla de *udev* para poder usar el chip *FTDI* de las *Econotags.*

Por otra parte, el código del programa *Hola Mundo* para la *Econotag* se encuentra en el directorio `/fenix/depar/atc/se/pracs/hello`.