

Módulo 2

Proyecto CIAA: Primeros pasos en Ubuntu / Linux

Autores: Joaquín Rodríguez, Juan Pablo Vecchio

Tutor: Ing. Marcelo Pistarelli

Supervisor: Ing. José Ignacio Sosa

Asesor: Ing. Gustavo Muro

Índice

1	Introducción.....	3
2	Trabajando en Linux.....	3
2.1	Definición	3
2.2	Sección de análisis	5
2.3	Añadir un directorio a la variable PATH.....	7
2.3.1	Solamente para la sesión activa	7
2.3.2	De manera permanente	8
3	Instalación y configuración del entorno IDE.....	9
3.1	Compilador ARM-GCC.....	9
3.1.1	Agregar la carpeta del compilador de ARM al PATH de Ubuntu.....	13
3.2	PHP	14
3.3	OpenOCD	15
3.4	IDE - Eclipse.....	23
3.5	Plug-In GNU-ARM-OpenOCD para Eclipse	26
3.6	Clonando el repositorio del proyecto CIAA en nuestra PC.....	31
3.6.1	Instalar GIT.....	31
4	Configuración del entorno ECLIPSE-IDE	36
4.1	Elección del Workspace.....	36
4.2	Primeros Pasos: El Proyecto “Blinking”	36
4.3	Indexación de archivos de cabecera.....	38
4.4	Configuración del Makefile	41
4.5	Debug en placa EDU-CIAA y Entorno IDE.....	43
4.5.1	Configuración del entorno CIAA-IDE	43
4.5.2	Compilación del proyecto.....	45
4.5.3	Depuración sobre la placa: configuración de OpenOCD para Debug.....	46

1 Introducción

En el presente módulo, básicamente vamos a realizar una adaptación del *Módulo 1: Utilización de la EDU-CIAA* para el sistema operativo, de licencia libre Ubuntu-Linux. La organización estará hecha de la siguiente manera: Primero, se explicarán brevemente una serie de temas, propios del sistema operativo, como ser la variable PATH, o el modo superusuario (SU), entre otros, luego el manejo de consola y la instalación a través de la misma de distintos paquetes que son necesarios para el funcionamiento de debugger y del entorno IDE. Cabe aclarar, que aquí no hay forma de descargar un entorno desarrollado particularmente para el proyecto, sino usaremos directamente el entorno IDE Eclipse, versión Kepler 4.3.

Al igual que en el módulo anterior, toda la información aquí detallada parte de la página del proyecto mismo, así que cualquier inconveniente, u omisión que se pudiera hacer en este documento, y se quiera ampliar, puede recurrirse a la página www.proyecto-ciaa.com.ar.

Por último, procederemos a la carga del mismo proyecto de ejemplo que se hizo para el sistema operativo Windows (Blinking).

2 Trabajando en Linux

2.1 Definición

Comencemos primero con el término genérico de variable.

Una variable es *un contenedor de memoria* que almacena un dato que podrá variar durante la ejecución de un programa.

Una variable tiene un nombre y un contenido. No entraremos en el detalle de la programación para explicar el tipo de variable y su contenido.

Por el instante nos contentaremos con saber que la variable de la que hablamos se llama **PATH** y que su contenido es una cadena que contiene rutas de directorios que, en Ubuntu, están separadas por el símbolo dos puntos (:). Si quisiéramos ver el contenido de una variable, abrimos un terminal de Linux, y escribimos:

echo \$variable

donde 'variable' es el nombre de la variable de la cuál queremos saber su contenido. Entonces, en el caso de la variable PATH debe escribir en el terminal:

echo \$PATH

Un detalle importante a tener en cuenta es que Linux diferencia las mayúsculas y las minúsculas, por lo que las sentencias ***echo \$PATH*** y ***echo \$path*** nos mostrarán sucesivamente, el contenido de dos variables distintas (en el caso de '***path***', si no está declarada, la ventana del terminal nos arrojará un error).

Para una presentación más agradable puede utilizar el comando

echo \$PATH | tr : \\n

Ejecutando este última línea, veremos una lista de carpetas como se aprecia en la Figura 1.



```
joaquin@ubuntu: ~  
joaquin@ubuntu:~$ echo $PATH | tr : \\n  
/usr/local/sbin  
/usr/local/bin  
/usr/sbin  
/usr/bin  
/sbin  
/bin  
/usr/games  
/usr/local/games  
joaquin@ubuntu:~$
```

← Contenido de la Variable PATH

Figura 1: Terminal de Linux. Contenido de la Variable PATH

2.2 Sección de análisis

Hemos visto que la variable PATH contiene una lista de directorios separados por dos puntos (:). Estos son los directorios en los que el shell (es decir, el *intérprete* de las líneas de comandos que introducimos en el Terminal) busca el comando que uno escribe desde el teclado. La búsqueda se hace en el orden en que están los directorios en la variable PATH.

Ejemplo:

Si escribimos en el terminal:

echo \$PATH

y obtenemos lo siguiente:

/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games:/home/carlos-vialfa/bin

Cuando se escribe un comando, el shell buscará primero en */usr/local/bin*, luego en */usr/bin*, luego en */usr/bin/X11*, luego en */usr/games* y finalmente en */home/carlos-vialfa/bin*. Desde el momento en que el shell encuentra el comando, detiene la búsqueda y ejecuta el comando encontrado.

Para movernos entre directorios, utilizamos el comando *cd*. Así tenemos los siguientes casos:

Comando	Consecuencia (luego de escribirlos y presionar la tecla enter)
<i>cd /</i>	Nos posiciona en el directorio Raíz del sistema, es decir, en el File System.
<i>cd 'DIRECTORIO'</i>	A partir de la posición en la que estamos, nos ubica en el DIRECTORIO indicado (en el caso de que exista). Si DIRECTORIO no existe, nos arroja error. Por ejemplo, si estamos en <i>/bin/cat</i> , y ejecutamos <i>cd etc</i> , luego nos encontraremos en <i>/bin/cat/etc</i> .
<i>cd ..</i>	Nos posiciona en el directorio anterior, respecto al que estamos cuando ejecutamos el comando. Por ejemplo, si estamos en <i>/bin/cat/etc</i> , luego de ejecutado este comando, estaremos ubicados en <i>/bin/cat</i>
<i>ls</i>	Muestra todos los archivos en la ubicación en donde estamos parados

Podemos ejecutar un comando utilizando:

- Su nombre
- La ruta absoluta
- La ruta relativa (utilizando "." o "..", en general para los programas o scripts que no se encuentran en PATH).

. es el directorio actual

.. es el directorio padre

Ejemplo:

Si estamos en la ruta */bin/cat/etc/*, podemos ejecutar el comando *passwd* haciendo alguna de las siguientes instrucciones:

- Por ruta absoluta: */bin/cat/etc/passwd*
- Por ruta relativa: *./passwd*

Al escribir un comando podemos encontrarnos con varias situaciones:

1. El comando es único y se encuentra en uno de los directorios de la variable PATH.
2. El comando no es único y se encuentra en varios directorios de la variable PATH.
3. El comando no se encuentra en ningún directorio de la variable PATH.

En el primer caso no hay mucho problema. Si hay un error al momento de la ejecución probablemente sea a causa de la sintaxis.

Solución: consultar el manual del comando

En el segundo caso las cosas tampoco son tan complicadas. Supongamos que disponemos de un comando que se encuentra en */usr/bin* y que hemos instalado desde las fuentes una versión más reciente del comando (instalamos una versión más reciente del programa) cuyo ejecutable se encuentra en */usr/local/bin*. El comando se llama *'prog'*.

Si llamamos al comando por su nombre ¿qué pasará?

El shell mira el PATH comenzando por el primer directorio encontrado.

En nuestro caso encuentra el comando */usr/local/bin*, entonces la línea completa que se ejecuta será */usr/local/bin/prog*.

En cambio si deseamos de todas maneras ejecutar el comando *prog* que se encuentra en */usr/bin* entonces tendremos que utilizar la ruta absoluta */usr/bin/prog*.

El tercer caso comprende 2 situaciones:

- el comando existe pero no se encuentra en ningún directorio de nuestro PATH
- el comando no existe

En los 2 casos el mensaje de error será *'command not found'*, pero la interpretación no será la misma. En la primera circunstancia, la forma de ejecutarlo será o bien, escribiendo la ruta absoluta, o agregando el directorio de nuestro comando a la variable PATH, como se explicará en la siguiente sección.

Otro detalle a tener en cuenta en Linux es que no cualquier usuario puede cambiar cualquier carpeta y/o archivo. Por ejemplo, un usuario *User1* puede cambiar las carpetas propias de su cuenta, no puede tocar las de otro usuario, ni tampoco aquellos que pertenecen al sistema. Solamente los usuarios autorizados pueden realizar cambios en esas carpetas y archivos. De la misma manera, hay distintos comandos que sólo podrá ejecutar un usuario tipo ROOT. Si intentáramos modificar algún archivo para el cuál no tuviéramos permiso, aparecerá en el terminal una leyenda en el terminal que dice “Permiso Denegado”, y el comando no se ejecutará. Si sabemos *exactamente* lo que estamos haciendo con dicho comando, y las consecuencias que éste acarrea, para poder ejecutar dichos comandos hay que anteponer la palabra *sudo* antes del comando. Ello le asignará permisos de superusuario al comando, y podrá llevarse a cabo.

Antes de ejecutarse, nos pedirá nuestra contraseña de usuario (parecerá que el cursor no se mueve, pero igualmente, se está escribiendo, y al finalizar, presionar la tecla ENTER).

IMPORTANTE: Todo los archivos y carpetas que creamos estando como ROOT, pertenecerán al usuario ROOT, por lo que, como usuarios tipo USER, no podremos modificarlos, sólo leerlos.

2.3 Añadir un directorio a la variable PATH

2.3.1 Solamente para la sesión activa

Si desea añadir por ejemplo `/home/user/prog` a la variable PATH, en la ventana del terminal debemos escribir:

```
export PATH=$PATH:/home/user/prog
```

Luego de ejecutar este comando, nuestro directorio se incorporará a la variable PATH al final de todo. En cambio, si escribimos en el terminal la siguiente sentencia:

```
export PATH=/home/user/prog:$PATH
```

el directorio deseado se incorporará a la variable PATH al principio de todo. Prestar atención a la pequeña diferencia: el comando en esencia, no cambió, solamente cambió lo que se encuentra a la derecha del signo igual. En el primer caso, \$PATH (que significa el contenido de toda la variable PATH), se encuentra a la izquierda, y luego le sigue el directorio que deseamos agregar. En el otro, nuestro directorio es el que está a la izquierda, y el resto del contenido previo, se encuentra a la derecha. En la Figura 2, se muestra el terminal de Linux, ejecutando el primer caso, para otro ejemplo, y luego consultando el valor grabado de la variable PATH.

```
@ubuntu: ~
joaquin@ubuntu:~$ export PATH=$PATH:/home/joaquin/Escritorio (1)
joaquin@ubuntu:~$ echo $PATH | tr : \\n
/usr/local/sbin
/usr/local/bin
/usr/sbin
/usr/bin
/sbin
/bin
/usr/games
/usr/local/games
/home/joaquin/Escritorio
joaquin@ubuntu:~$
```

Figura 2: Agregando un directorio al final de la Variable PATH

Ahora puede utilizar su programa escribiendo simplemente su nombre, pero estos cambios serán válidos mientras la sesión del usuario esté abierta. Cuando se cierre, estos cambios desaparecerán.

2.3.2 De manera permanente

Si deseamos configurar PATH de forma permanente, debe editar el archivo de configuración de su shell de conexión. Como por lo general el shell BASH es el más utilizado, debe editar su archivo */home/user/.bashrc*.

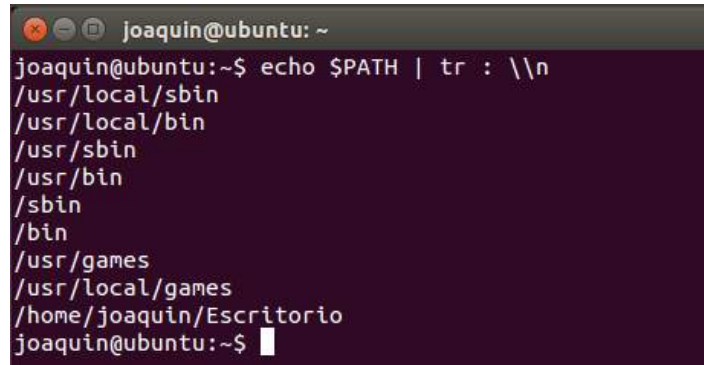
La manera más fácil es utilizar la línea de comandos, y será el método que explicaremos aquí. Con nuestro terminal de Linux abierto, debemos escribir el siguiente comando:

```
echo 'export PATH=$PATH:/home/user/prog' >> /home/user/.bashrc
```

Así, todas las veces que consultemos el valor de PATH, ésta contendrá el directorio */home/user/prog*. Esta operación puede ser ejecutada por el usuario *user* dado que se trata de su entorno. Si consultamos inmediatamente luego de escribir el archivo *bashrc*, puede que no nos aparezca el directorio que acabamos de agregar. Para ello, debemos cerrar el terminal y la sesión de Linux, y luego volver a entrar para que los cambios tengan efectos visibles. En las Figura 3 y 4 se muestran, respectivamente, la línea de comandos luego de ejecutada la instrucción, y seguidamente, la consulta del valor almacenado en PATH mediante el comando *echo \$PATH*.

```
joaquin@ubuntu: ~
joaquin@ubuntu:~$ echo 'export PATH=$PATH:/home/joaquin/Escritorio' >> /home/joaquin/.bashrc
joaquin@ubuntu:~$
```

Figura 3: Línea de comandos para agregar permanentemente un directorio a PATH



```
joaquin@ubuntu: ~  
joaquin@ubuntu:~$ echo $PATH | tr : \\n  
/usr/local/sbin  
/usr/local/bin  
/usr/sbin  
/usr/bin  
/sbin  
/bin  
/usr/games  
/usr/local/games  
/home/joaquin/Escritorio  
joaquin@ubuntu:~$
```

Figura 4: Consulta del valor de PATH, luego de re-abierta la sesión del usuario

3 Instalación y configuración del entorno IDE

3.1 Compilador ARM-GCC

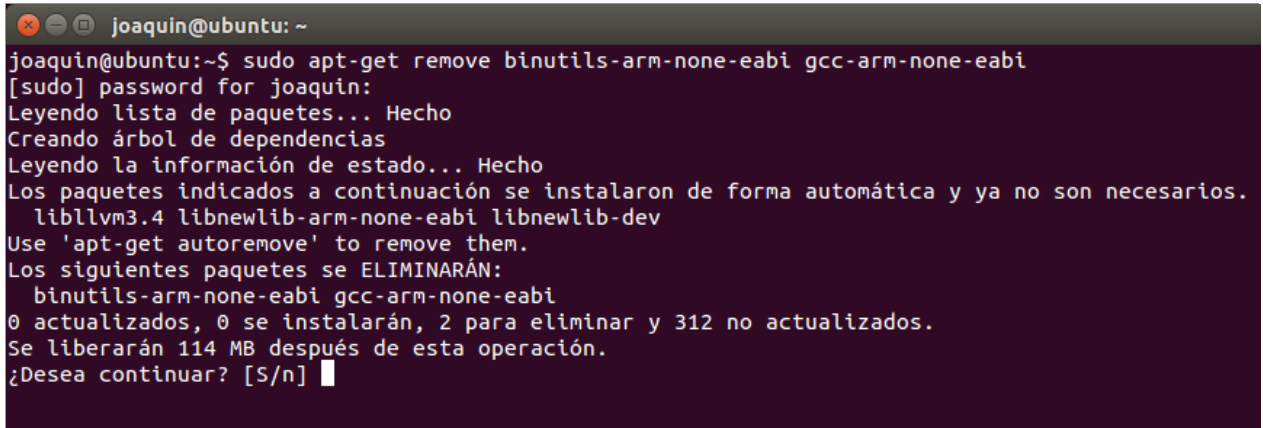
Para compilar el código para el procesador ARM (por ejemplo para la CIAA-NXP, procesador ARM será el LPC4337 del fabricante NXP), debe instalarse el compilador [arm-none-eabi-gcc](https://launchpad.net/gcc-arm-embedded), lo cual se puede hacer por terminal, o accediendo al sitio <https://launchpad.net/gcc-arm-embedded>. Por la misma razón que el proyecto CIAA, estas herramientas están en continua modificación, ya sea para corregir errores, o para ir adaptándola para las distintas versiones que van surgiendo del Sistema Operativo. Es por ello, que en algunos casos, dependiendo las versiones de Ubuntu con la que contemos, los pasos pueden diferir. Es por esto principalmente, que se recomienda dirigirse a la página antes mencionada, y fijarse los pasos a seguir en función del sistema operativo particular en el que se está trabajando. Aquí lo haremos para la versión de Ubuntu 14, que es una de las últimas que liberó.

La instalación consta de un paquete en 3 partes, pero por los problemas mencionados arriba, hay un directorio que forma parte inherente de Ubuntu, y que el instalador corriente desea sobrescribir, y al intentarlo, ocurre un error por falta de permisos. Por lo tanto, debemos realizar un paso previo. Vamos al terminal de Linux, y escribimos la siguiente línea:

sudo apt-get remove binutils-arm-none-eabi gcc-arm-none-eabi

De esta manera, se eliminarán los paquetes anteriores, que casualmente, tienen el mismo nombre que el que queremos instalar. Ello no producirá ningún problema sobre nuestro Ubuntu. La ventana que comienza a ejecutarse se muestra en la Figura 5, la cual, luego de habernos pedido la clave para SuperUsuario, nos solicita confirmación de que aceptamos que se libere cierta

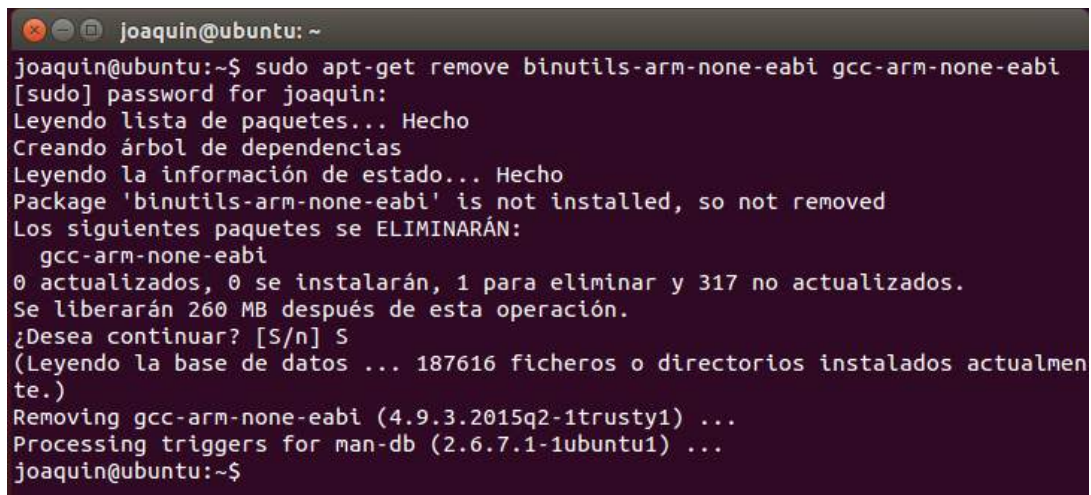
cantidad de espacio en el disco. Escribimos la letra 'S' si estamos de acuerdo, y apretando ENTER procederemos a la desinstalación del programa que nos causa conflicto.



```
joaquin@ubuntu: ~
joaquin@ubuntu:~$ sudo apt-get remove binutils-arm-none-eabi gcc-arm-none-eabi
[sudo] password for joaquin:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Los paquetes indicados a continuación se instalaron de forma automática y ya no son necesarios.
 libllvm3.4 libnewlib-arm-none-eabi libnewlib-dev
Use 'apt-get autoremove' to remove them.
Los siguientes paquetes se ELIMINARÁN:
  binutils-arm-none-eabi gcc-arm-none-eabi
0 actualizados, 0 se instalarán, 2 para eliminar y 312 no actualizados.
Se liberarán 114 MB después de esta operación.
¿Desea continuar? [S/n]
```

Figura 5: Terminal cuando queremos desinstalar el paquete de Ubuntu

La ventana de la desinstalación finalizada se muestra en la Figura 6.



```
joaquin@ubuntu: ~
joaquin@ubuntu:~$ sudo apt-get remove binutils-arm-none-eabi gcc-arm-none-eabi
[sudo] password for joaquin:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Package 'binutils-arm-none-eabi' is not installed, so not removed
Los siguientes paquetes se ELIMINARÁN:
  gcc-arm-none-eabi
0 actualizados, 0 se instalarán, 1 para eliminar y 317 no actualizados.
Se liberarán 260 MB después de esta operación.
¿Desea continuar? [S/n] S
(Leyendo la base de datos ... 187616 ficheros o directorios instalados actualmen
te.)
Removing gcc-arm-none-eabi (4.9.3.2015q2-1trusty1) ...
Processing triggers for man-db (2.6.7.1-1ubuntu1) ...
joaquin@ubuntu:~$
```

Figura 6: Ventana de finalización de la instalación

Ahora, para empezar con la instalación, continuamos sobre un terminal de Linux, y escribimos la primera sentencia:

sudo add-apt-repository ppa:terry.guo/gcc-arm-embedded

Luego de apretar ENTER, nos aparecerá un grupo de comentarios relativo al compilador, y por último, nos pedirá que apretemos ENTER si deseamos continuar, tal y como se muestra en la Figura 7.

```
joaquin@ubuntu: ~  
"sudo apt-get install gcc-arm-none-eabi"  
  
To remove installed toolchain, just input "sudo apt-get remove gcc-arm-none-eabi".  
To update the toolchain, just repeat above step2 and step3.  
  
Questions should be asked at https://answers.launchpad.net/gcc-arm-embedded  
  
Bug can be filed at https://bugs.launchpad.net/gcc-arm-embedded/+filebug. It is highly encouraged to ask question first  
before filing a bug.  
  
!!! If you are using Ubuntu 14.04 and later, please be careful because there are packages with same name but produced b  
y Debian and inherited by Ubuntu. Simply follow the above 3 steps, you may end up with gcc-arm-none-eabi from Ubuntu. S  
o to install gcc-arm-none-eabi from ARM, steps are:  
1). sudo apt-get remove binutils-arm-none-eabi gcc-arm-none-eabi  
2). sudo add-apt-repository ppa:terry.guo/gcc-arm-embedded  
3). sudo apt-get update  
4). sudo apt-get install gcc-arm-none-eabi=4.9.3.2015q1-0trusty13  
   or  
   sudo apt-get install gcc-arm-none-eabi=4.9.3.2015q1-0utopic14  
  
Meanwhile we are working with Debian to consolidate and unify this toolchain.  
Más información: https://launchpad.net/~terry.guo/archive/ubuntu/gcc-arm-embedded  
Pulse [Intro] para continuar o ctrl-c para cancelar
```

Figura 7: Primer Paso para la instalación del compilador GCC de ARM

Si continuamos con la instalación, descargará un par de paquetes, y al finalizar, el terminal se verá como se aprecia en la Figura 8.

```
joaquin@ubuntu: ~  
  
!!! If you are using Ubuntu 14.04 and later, please be careful because there are packages with same name but produced  
y Debian and inherited by Ubuntu. Simply follow the above 3 steps, you may end up with gcc-arm-none-eabi from Ubuntu.  
o to install gcc-arm-none-eabi from ARM, steps are:  
1). sudo apt-get remove binutils-arm-none-eabi gcc-arm-none-eabi  
2). sudo add-apt-repository ppa:terry.guo/gcc-arm-embedded  
3). sudo apt-get update  
4). sudo apt-get install gcc-arm-none-eabi=4.9.3.2015q1-0trusty13  
   or  
   sudo apt-get install gcc-arm-none-eabi=4.9.3.2015q1-0utopic14  
  
Meanwhile we are working with Debian to consolidate and unify this toolchain.  
Más información: https://launchpad.net/~terry.guo/archive/ubuntu/gcc-arm-embedded  
Pulse [Intro] para continuar o ctrl-c para cancelar  
  
gpg: anillo «/tmp/tmpvfm87_p/secring.gpg» creado  
gpg: anillo «/tmp/tmpvfm87_p/pubring.gpg» creado  
gpg: solicitando clave A3421AFB de hkp servidor keyserver.ubuntu.com  
gpg: /tmp/tmpvfm87_p/trustdb.gpg: se ha creado base de datos de confianza  
gpg: clave A3421AFB: clave pública "Launchpad PPA for Terry Guo" importada  
gpg: Cantidad total procesada: 1  
gpg:          importadas: 1 (RSA: 1)  
OK  
joaquin@ubuntu:~$
```

Figura 8: Fin de la primera etapa de instalación del compilador GCC de ARM

La segunda sentencia es una actualización de los componentes a su última versión, lo cuál se realiza con la siguiente sentencia:

sudo apt-get update

El comienzo de la actualización se ve en la Figura 9.

```
joaquin@ubuntu:~$ sudo apt-get update
Ign http://us.archive.ubuntu.com trusty InRelease
Ign http://security.ubuntu.com trusty-security InRelease
Ign http://extras.ubuntu.com trusty InRelease
Ign http://ppa.launchpad.net trusty InRelease
Ign http://us.archive.ubuntu.com trusty-updates InRelease
Obj http://extras.ubuntu.com trusty Release.gpg
Des:1 http://security.ubuntu.com trusty-security Release.gpg [933 B]
Ign http://us.archive.ubuntu.com trusty-backports InRelease
Obj http://ppa.launchpad.net trusty Release.gpg
Obj http://us.archive.ubuntu.com trusty Release.gpg
Obj http://extras.ubuntu.com trusty Release
Des:2 http://security.ubuntu.com trusty-security Release [63,5 kB]
Obj http://ppa.launchpad.net trusty Release
Des:3 http://us.archive.ubuntu.com trusty-updates Release.gpg [933 B]
Obj http://extras.ubuntu.com trusty/main Sources
Obj http://us.archive.ubuntu.com trusty-backports Release.gpg
Obj http://ppa.launchpad.net trusty/main amd64 Packages
Obj http://us.archive.ubuntu.com trusty Release
Obj http://extras.ubuntu.com trusty/main amd64 Packages
Obj http://ppa.launchpad.net trusty/main i386 Packages
Des:4 http://us.archive.ubuntu.com trusty-updates Release [63,5 kB]
Obj http://extras.ubuntu.com trusty/main i386 Packages
```

Figura 9: Segundo paso, actualización de los componentes descargados en la primer etapa

Si todo salió bien, nos encontraremos con una ventana parecida a la Figura 12.

```
joaquin@ubuntu:~$ sudo apt-get update
Obj http://us.archive.ubuntu.com trusty-backports/multiverse Sources
Obj http://us.archive.ubuntu.com trusty-backports/main amd64 Packages
Obj http://us.archive.ubuntu.com trusty-backports/restricted amd64 Packages
Obj http://us.archive.ubuntu.com trusty-backports/universe amd64 Packages
Obj http://us.archive.ubuntu.com trusty-backports/multiverse amd64 Packages
Obj http://us.archive.ubuntu.com trusty-backports/main i386 Packages
Obj http://us.archive.ubuntu.com trusty-backports/restricted i386 Packages
Obj http://us.archive.ubuntu.com trusty-backports/universe i386 Packages
Obj http://us.archive.ubuntu.com trusty-backports/multiverse i386 Packages
Obj http://us.archive.ubuntu.com trusty-backports/main Translation-en
Obj http://us.archive.ubuntu.com trusty-backports/multiverse Translation-en
Obj http://us.archive.ubuntu.com trusty-backports/restricted Translation-en
Obj http://us.archive.ubuntu.com trusty-backports/universe Translation-en
Ign http://us.archive.ubuntu.com trusty/main Translation-es_ES
Ign http://us.archive.ubuntu.com trusty/main Translation-en_US
Ign http://us.archive.ubuntu.com trusty/multiverse Translation-es_ES
Ign http://us.archive.ubuntu.com trusty/multiverse Translation-en_US
Ign http://us.archive.ubuntu.com trusty/restricted Translation-es_ES
Ign http://us.archive.ubuntu.com trusty/restricted Translation-en_US
Ign http://us.archive.ubuntu.com trusty/universe Translation-es_ES
Ign http://us.archive.ubuntu.com trusty/universe Translation-en_US
Descargados 3.214 kB en 47seg. (68,0 kB/s)
Leyendo lista de paquetes... Hecho
joaquin@ubuntu:~$
```

Figura 10: Fin de la actualización

Por último, la sentencia que instalará definitivamente el compilador es la siguiente:

sudo apt-get install gcc-arm-none-eabi

Y la ventana correspondiente a la ejecución de esta instrucción es la Figura 11.

```
joaquin@ubuntu: ~
joaquin@ubuntu:~$ sudo apt-get install gcc-arm-none-eabi
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Los paquetes indicados a continuación se instalaron de forma automática y ya no son necesarios.
  libllvm3.4 libnewlib-arm-none-eabi libnewlib-dev
Use 'apt-get autoremove' to remove them.
Se instalarán los siguientes paquetes NUEVOS:
 gcc-arm-none-eabi
0 actualizados, 1 se instalarán, 0 para eliminar y 312 no actualizados.
Se necesita descargar 0 B/30,2 MB de archivos.
Se utilizarán 260 MB de espacio de disco adicional después de esta operación.
Seleccionando el paquete gcc-arm-none-eabi previamente no seleccionado.
(Leyendo la base de datos ... 177336 ficheros o directorios instalados actualmente.)
Preparing to unpack ../gcc-arm-none-eabi_4.9.3.2015q2-1trusty1_amd64.deb ...
Unpacking gcc-arm-none-eabi (4.9.3.2015q2-1trusty1) ...
Processing triggers for man-db (2.6.7.1-1ubuntu1) ...
Configurando gcc-arm-none-eabi (4.9.3.2015q2-1trusty1) ...
joaquin@ubuntu:~$
```

Figura 11: Fin de la instalación del compilador GCC-ARM

3.1.1 Agregar la carpeta del compilador de ARM al PATH de Ubuntu

Para facilitar el uso del compilador de ARM, se recomienda incluirlo en la variable PATH del Ubuntu. Esta carpeta es `/usr/arm-none-eabi/bin`, y de lo explicado en la “*Añadir un directorio a la variable PATH*”, tenemos que la instrucción a ejecutar en el terminal es:

`echo 'export PATH=$PATH:/usr/arm-none-eabi/bin' >> /home/user/.bashrc`

donde ‘*user*’ deberá ser reemplazado por el nombre de usuario activo en el que se desee registrar este directorio. No olvidemos que luego de ejecutar este comando, para que tenga efecto, debemos cerrar y volver a abrir nuestra sesión en Ubuntu. En la Figura 12 se puede ver que luego de ejecutar esta instrucción, y reiniciar la sesión, la variable fue actualizada.

```
joaquin@ubuntu: ~
joaquin@ubuntu:~$ echo $PATH | tr : \\n
/usr/local/sbin
/usr/local/bin
/usr/sbin
/usr/bin
/sbin
/bin
/usr/games
/usr/local/games
/home/joaquin/Escritorio
/home/joaquin/Escritorio
/usr/arm-none-eabi/bin
joaquin@ubuntu:~$
```

Figura 12: Valor actualizado de la variable PATH

3.2 PHP

Como mencionamos en el Módulo 1, para poder generar el sistema operativo OSEK es necesario que el compilador pueda interpretar código en PHP. Para ello, necesitamos instalar la siguiente herramienta desde el Terminal de Linux:

```
sudo apt-get install php5-cli
```

Y si la instalación finalizó con éxito, veremos una ventana como la que se muestra en la Figura 13.

```
joaquin@ubuntu: ~
Seleccionando el paquete php5-json previamente no seleccionado.
(Leyendo la base de datos ... 180772 ficheros o directorios instalados actualmente.)
Preparing to unpack ../php5-json_1.3.2-2build1_amd64.deb ...
Unpacking php5-json (1.3.2-2build1) ...
Seleccionando el paquete php5-common previamente no seleccionado.
Preparing to unpack ../php5-common_5.5.9+dfsg-1ubuntu4.11_amd64.deb ...
Unpacking php5-common (5.5.9+dfsg-1ubuntu4.11) ...
Seleccionando el paquete php5-cli previamente no seleccionado.
Preparing to unpack ../php5-cli_5.5.9+dfsg-1ubuntu4.11_amd64.deb ...
Unpacking php5-cli (5.5.9+dfsg-1ubuntu4.11) ...
Seleccionando el paquete php5-readline previamente no seleccionado.
Preparing to unpack ../php5-readline_5.5.9+dfsg-1ubuntu4.11_amd64.deb ...
Unpacking php5-readline (5.5.9+dfsg-1ubuntu4.11) ...
Processing triggers for man-db (2.6.7.1-1ubuntu1) ...
Configurando php5-json (1.3.2-2build1) ...
php5_invoke: Enable module json for cli SAPI
Configurando php5-common (5.5.9+dfsg-1ubuntu4.11) ...

Creating config file /etc/php5/mods-available/pdo.ini with new version
php5_invoke: Enable module pdo for cli SAPI

Creating config file /etc/php5/mods-available/opcache.ini with new version
php5_invoke: Enable module opcache for cli SAPI
Configurando php5-cli (5.5.9+dfsg-1ubuntu4.11) ...
update-alternatives: utilizando /usr/bin/php5 para proveer /usr/bin/php (php) en modo automático

Creating config file /etc/php5/cli/php.ini with new version
php5_invoke opcache: already enabled for cli SAPI
php5_invoke json: already enabled for cli SAPI
php5_invoke pdo: already enabled for cli SAPI
Configurando php5-readline (5.5.9+dfsg-1ubuntu4.11) ...

Creating config file /etc/php5/mods-available/readline.ini with new version
php5_invoke: Enable module readline for cli SAPI
joaquin@ubuntu:~$
```

Figura 13: Terminal para la instalación del complemento PHP5

3.3 OpenOCD

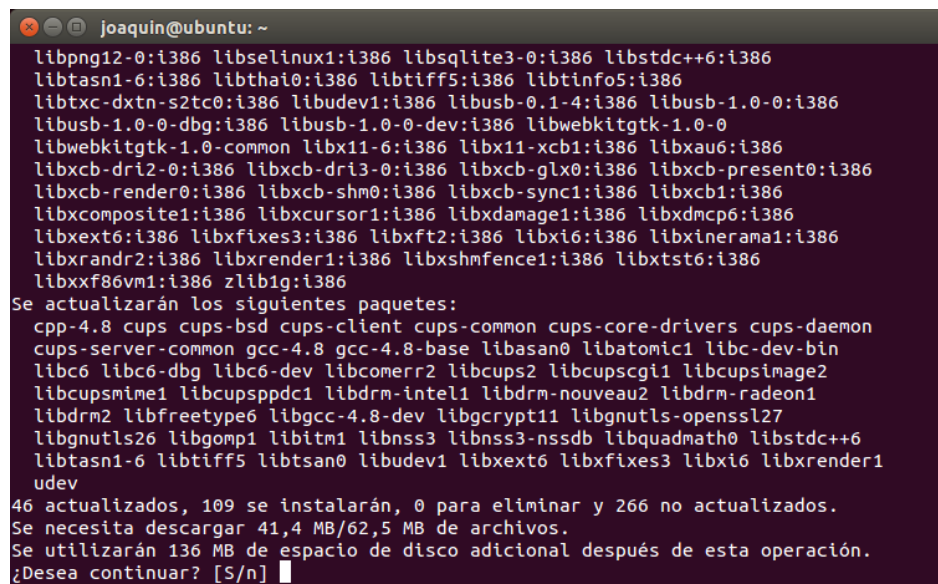
El hardware de la CIAA viene provisto con el chip FT2232H, que se encarga de hacer un puente entre la interface JTAG del microcontrolador y el USB que conecta a la PC en el puerto USB dedicado al debug. La herramienta de código abierto OpenOCD (On-Chip Debugger) es la encargada de manejar el chip FT2232H por el USB y, a la vez, de todo lo referido al JTAG. Luego el debugger (GDB) utilizado en nuestro software IDE-Eclipse puede hacer su tarea simplemente conectándose al puerto 3333 (TCP) que el OpenOCD tiene en escucha esperando la conexión.

Para utilizar OpenOCD en Linux debemos compilarlo, configurándolo para que funcione con el chip FT2232. Para todos estos pasos, es recomendable tener la placa conectada a la PC.

Si estamos en un sistema operativo de 64 bits (ej: Ubuntu 64-bits), es necesario instalar primero las siguientes librerías (para 32 bits saltar este paso)

```
sudo apt-get install libgtk2.0-0:i386 libxtst6:i386 libpangox-1.0-0:i386 libpangoxft-1.0-0:i386 libidn11:i386 libglu1-mesa:i386 libncurses5:i386 libudev1:i386 libusb-1.0:i386 libusb-0.1:i386 gtk2-engines-murrine:i386 libnss3-1d:i386 libwebkitgtk-1.0-0
```

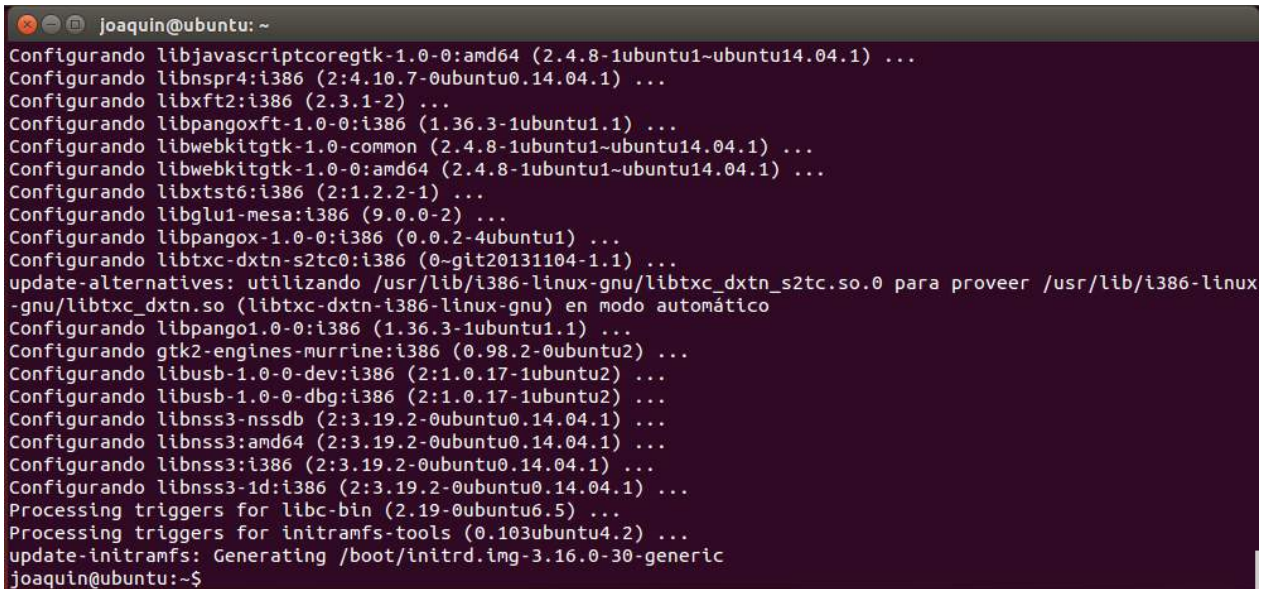
Como en varias instalaciones anteriores, nos pedirá la contraseña de SU, y luego una confirmación por la cantidad de espacio que el nuevo programa ocupará. Esto se ve en la Figura 14. Escribimos 'S' y apretamos ENTER.



```
joaquin@ubuntu: ~  
libpng12-0:i386 libselinux1:i386 libsqlite3-0:i386 libstdc++6:i386  
libtasn1-6:i386 libthai0:i386 libtiff5:i386 libtinfo5:i386  
libtxc-dxtn-s2tc0:i386 libudev1:i386 libusb-0.1-4:i386 libusb-1.0-0:i386  
libusb-1.0-0-dbg:i386 libusb-1.0-0-dev:i386 libwebkitgtk-1.0-0  
libwebkitgtk-1.0-common libx11-6:i386 libx11-xcb1:i386 libxau6:i386  
libxcb-dri2-0:i386 libxcb-dri3-0:i386 libxcb-glx0:i386 libxcb-present0:i386  
libxcb-render0:i386 libxcb-shm0:i386 libxcb-sync1:i386 libxcb1:i386  
libxcomposite1:i386 libxcursor1:i386 libxdamage1:i386 libxdmcp6:i386  
libxext6:i386 libxfixes3:i386 libxft2:i386 libxi6:i386 libxinerama1:i386  
libxrandr2:i386 libxrender1:i386 libxshmfence1:i386 libxtst6:i386  
libxxf86vm1:i386 zlib1g:i386  
Se actualizarán los siguientes paquetes:  
cpp-4.8 cups cups-bsd cups-client cups-common cups-core-drivers cups-daemon  
cups-server-common gcc-4.8 gcc-4.8-base libasan0 libatomic1 libc-dev-bin  
libc6 libc6-dbg libc6-dev libcomerr2 libcups2 libcups2-gi1 libcupsimage2  
libcupsmime1 libcupsppdc1 libdrm-intel1 libdrm-nouveau2 libdrm-radeon1  
libdrm2 libfreetype6 libgcc-4.8-dev libgcrpt11 libgnutls-openssl27  
libgnutls26 libgomp1 libitm1 libnss3 libnss3-nssdb libquadmath0 libstdc++6  
libtasn1-6 libtiff5 libtsan0 libudev1 libxext6 libxfixes3 libxi6 libxrender1  
udev  
46 actualizados, 109 se instalarán, 0 para eliminar y 266 no actualizados.  
Se necesita descargar 41,4 MB/62,5 MB de archivos.  
Se utilizarán 136 MB de espacio de disco adicional después de esta operación.  
¿Desea continuar? [S/n]
```

Figura 14: Instalación del paquete para Ubuntu x64 (no es necesario si tenemos Ubuntu x32)

El fin de la instalación se puede ver en la Figura 15.



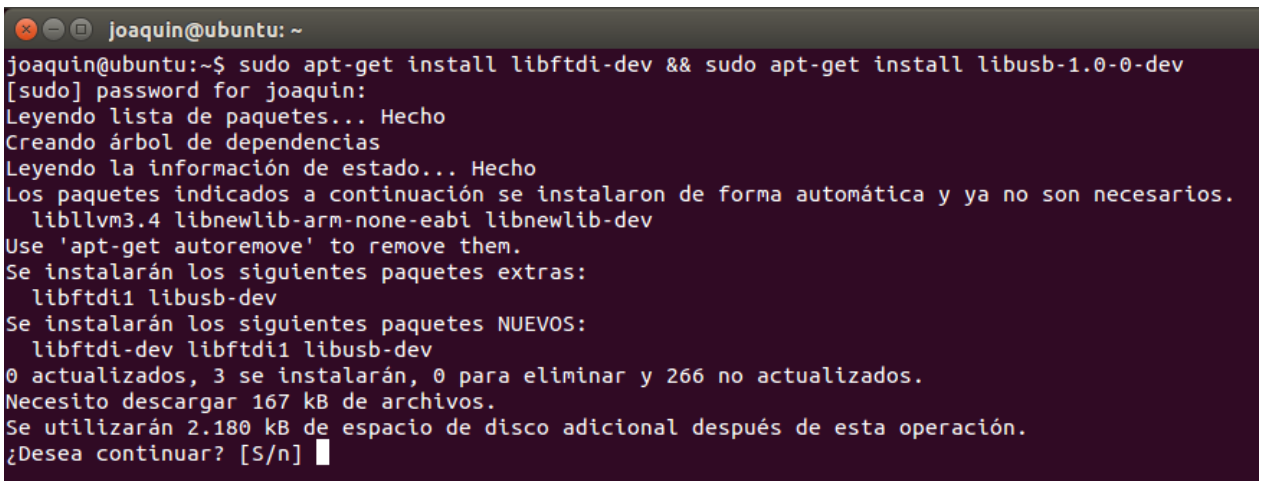
```
joaquin@ubuntu: ~  
Configurando libjavascriptcoregtk-1.0-0:amd64 (2.4.8-1ubuntu1~ubuntu14.04.1) ...  
Configurando libnspr4:i386 (2:4.10.7-0ubuntu0.14.04.1) ...  
Configurando libxft2:i386 (2.3.1-2) ...  
Configurando libpangoxft-1.0-0:i386 (1.36.3-1ubuntu1.1) ...  
Configurando libwebkitgtk-1.0-common (2.4.8-1ubuntu1~ubuntu14.04.1) ...  
Configurando libwebkitgtk-1.0-0:amd64 (2.4.8-1ubuntu1~ubuntu14.04.1) ...  
Configurando libxtst6:i386 (2:1.2.2-1) ...  
Configurando libglu1-mesa:i386 (9.0.0-2) ...  
Configurando libpango-1.0-0:i386 (0.0.2-4ubuntu1) ...  
Configurando libtxc-dxtn-s2tc0:i386 (0-git20131104-1.1) ...  
update-alternatives: utilizando /usr/lib/i386-linux-gnu/libtxc_dxtn_s2tc.so.0 para proveer /usr/lib/i386-linux  
-gnu/libtxc_dxtn.so (libtxc-dxtn-i386-linux-gnu) en modo automático  
Configurando libpango1.0-0:i386 (1.36.3-1ubuntu1.1) ...  
Configurando gtk2-engines-murrine:i386 (0.98.2-0ubuntu2) ...  
Configurando libusb-1.0-0-dev:i386 (2:1.0.17-1ubuntu2) ...  
Configurando libusb-1.0-0-dbg:i386 (2:1.0.17-1ubuntu2) ...  
Configurando libnss3-nssdb (2:3.19.2-0ubuntu0.14.04.1) ...  
Configurando libnss3:amd64 (2:3.19.2-0ubuntu0.14.04.1) ...  
Configurando libnss3:i386 (2:3.19.2-0ubuntu0.14.04.1) ...  
Configurando libnss3-1d:i386 (2:3.19.2-0ubuntu0.14.04.1) ...  
Processing triggers for libc-bin (2.19-0ubuntu6.5) ...  
Processing triggers for initramfs-tools (0.103ubuntu4.2) ...  
update-initramfs: Generating /boot/initrd.img-3.16.0-30-generic  
joaquin@ubuntu:~$
```

Figura 15: Fin de la instalación del paquete para Ubuntu x64

Ahora lo que debemos hacer es instalar el driver necesario para el chip FT2232 y el paquete libusb:

sudo apt-get install libftdi-dev && sudo apt-get install libusb-1.0-0-dev

Nuevamente, nos pedirá nuestra contraseña de usuario, y una confirmación para la instalación, como se ve en la Figura 16. Escribimos ‘S’ y apretamos la tecla ENTER.



```
joaquin@ubuntu: ~  
joaquin@ubuntu:~$ sudo apt-get install libftdi-dev && sudo apt-get install libusb-1.0-0-dev  
[sudo] password for joaquin:  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias  
Leyendo la información de estado... Hecho  
Los paquetes indicados a continuación se instalaron de forma automática y ya no son necesarios.  
 libllvm3.4 libnewlib-arm-none-eabi libnewlib-dev  
Use 'apt-get autoremove' to remove them.  
Se instalarán los siguientes paquetes extras:  
 libftdi1 libusb-dev  
Se instalarán los siguientes paquetes NUEVOS:  
 libftdi-dev libftdi1 libusb-dev  
0 actualizados, 3 se instalarán, 0 para eliminar y 266 no actualizados.  
Necesito descargar 167 kB de archivos.  
Se utilizarán 2.180 kB de espacio de disco adicional después de esta operación.  
¿Desea continuar? [S/n] S
```

Figura 16: Instalación de libusb y del driver para el FT2232. Primera parte

Como utilizamos el operador '&&', nos pedirá confirmación para el segundo operando. Nuevamente, confirmamos con 'S' y apretamos ENTER, como se ve en la Figura 17.

```
joaquin@ubuntu: ~
Seleccionando el paquete libftdi-dev previamente no seleccionado.
Preparing to unpack .../libftdi-dev_0.20-1ubuntu1_amd64.deb ...
Unpacking libftdi-dev (0.20-1ubuntu1) ...
Processing triggers for man-db (2.6.7.1-1ubuntu1) ...
Processing triggers for doc-base (0.10.5) ...
Procesando 2 archivos doc-base añadidos...
Configurando libftdi1:amd64 (0.20-1ubuntu1) ...
Configurando libusb-dev (2:0.1.12-23.3ubuntu1) ...
Configurando libftdi-dev (0.20-1ubuntu1) ...
Processing triggers for libc-bin (2.19-0ubuntu6.5) ...
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Los paquetes indicados a continuación se instalaron de forma automática y ya no son necesarios.
  libllvm3.4 libnewlib-arm-none-eabi libnewlib-dev
Use 'apt-get autoremove' to remove them.
Se instalarán los siguientes paquetes extras:
  libusb-1.0-doc
Se instalarán los siguientes paquetes NUEVOS:
  libusb-1.0-0-dev libusb-1.0-doc
0 actualizados, 2 se instalarán, 0 para eliminar y 266 no actualizados.
Necesito descargar 169 kB de archivos.
Se utilizarán 1.488 kB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] s
```

Figura 17: Instalación de libusb y del driver para el FT2232. Segunda parte

Cuando finalice todo, veremos una ventana de Terminal, parecido a la de la Figura 18.

```
joaquin@ubuntu: ~
Use 'apt-get autoremove' to remove them.
Se instalarán los siguientes paquetes extras:
  libusb-1.0-doc
Se instalarán los siguientes paquetes NUEVOS:
  libusb-1.0-0-dev libusb-1.0-doc
0 actualizados, 2 se instalarán, 0 para eliminar y 266 no actualizados.
Necesito descargar 169 kB de archivos.
Se utilizarán 1.488 kB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] s
Des:1 http://us.archive.ubuntu.com/ubuntu/ trusty/main libusb-1.0-0-dev amd64 2:1.0.17-1ubuntu2 [54,7 kB]
Des:2 http://us.archive.ubuntu.com/ubuntu/ trusty/main libusb-1.0-doc all 2:1.0.17-1ubuntu2 [115 kB]
Descargados 169 kB en 2seg. (74,1 kB/s)
Seleccionando el paquete libusb-1.0-0-dev:amd64 previamente no seleccionado.
(Leyendo la base de datos ... 180576 ficheros o directorios instalados actualmente.)
Preparing to unpack .../libusb-1.0-0-dev_2%3a1.0.17-1ubuntu2_amd64.deb ...
Unpacking libusb-1.0-0-dev:amd64 (2:1.0.17-1ubuntu2) ...
Seleccionando el paquete libusb-1.0-doc previamente no seleccionado.
Preparing to unpack .../libusb-1.0-doc_2%3a1.0.17-1ubuntu2_all.deb ...
Unpacking libusb-1.0-doc (2:1.0.17-1ubuntu2) ...
Processing triggers for doc-base (0.10.5) ...
Procesando 1 archivo doc-base añadido...
Configurando libusb-1.0-0-dev:amd64 (2:1.0.17-1ubuntu2) ...
Configurando libusb-1.0-doc (2:1.0.17-1ubuntu2) ...
joaquin@ubuntu:~$
```

Figura 18: Fin de la instalación de libusb y del driver para el FT2232

Otro paquete que necesitamos instalar es el siguiente:

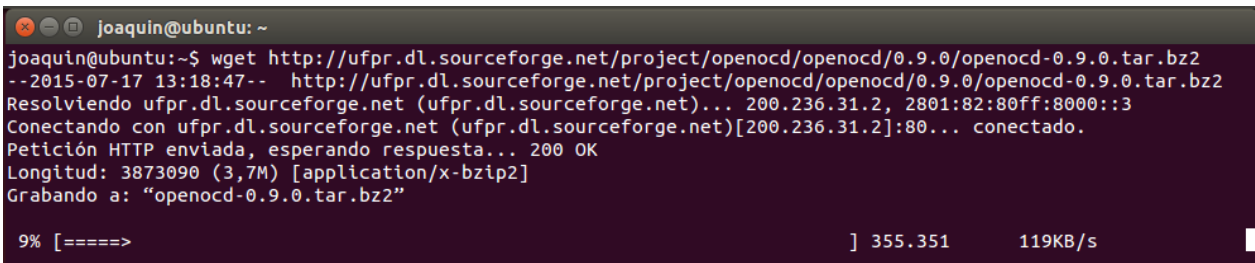
sudo apt-get install pkg-config

Como antes, lo escribimos en el terminal, y al terminar, presionamos ENTER y comienza la instalación.

Para proceder a la descarga del OpenOCD versión 0.9.0, utilizamos el comando **'wget'**, y cuyos comandos se detallan a continuación:

1) ***wget http://ufpr.dl.sourceforge.net/project/openocd/openocd/0.9.0/openocd-0.9.0.tar.bz2***

con lo que empezará una descarga a través del terminal, tal como se ve en la Figura 19.



```
joaquin@ubuntu: ~
joaquin@ubuntu:~$ wget http://ufpr.dl.sourceforge.net/project/openocd/openocd/0.9.0/openocd-0.9.0.tar.bz2
--2015-07-17 13:18:47-- http://ufpr.dl.sourceforge.net/project/openocd/openocd/0.9.0/openocd-0.9.0.tar.bz2
Resolviendo ufpr.dl.sourceforge.net (ufpr.dl.sourceforge.net)... 200.236.31.2, 2801:82:80ff:8000::3
Conectando con ufpr.dl.sourceforge.net (ufpr.dl.sourceforge.net)[200.236.31.2]:80... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 3873090 (3,7M) [application/x-bzip2]
Grabando a: "openocd-0.9.0.tar.bz2"

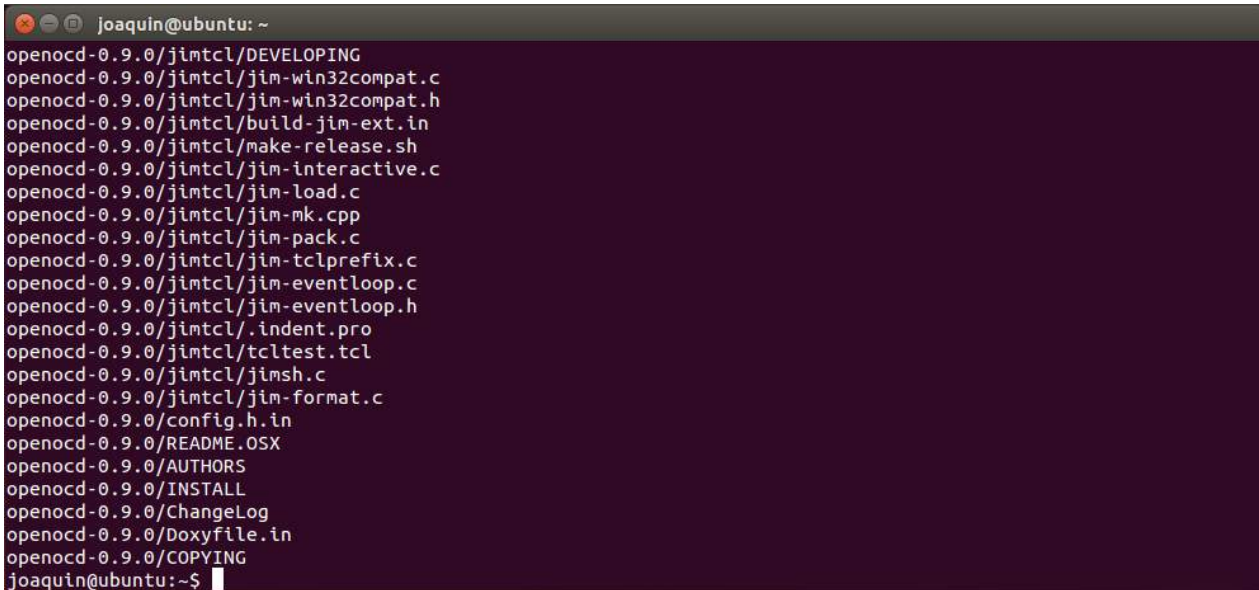
9% [=====>] 355.351 119KB/s
```

Figura 19: Descarga del OpenOCD utilizando wget

y una vez que finalice la descarga, ejecutamos el siguiente:

2) ***tar -xvzf openocd-0.9.0.tar.bz2***

Con lo cual, comenzará a descomprimirse el archivo BZ2. Cuando finalice, si todo salió bien, veremos el terminal como el que se aprecia en la Figura 20.



```
joaquin@ubuntu: ~
openocd-0.9.0/jimtcl/DEVELOPING
openocd-0.9.0/jimtcl/jim-win32compat.c
openocd-0.9.0/jimtcl/jim-win32compat.h
openocd-0.9.0/jimtcl/build-jim-ext.in
openocd-0.9.0/jimtcl/make-release.sh
openocd-0.9.0/jimtcl/jim-interactive.c
openocd-0.9.0/jimtcl/jim-load.c
openocd-0.9.0/jimtcl/jim-mk.cpp
openocd-0.9.0/jimtcl/jim-pack.c
openocd-0.9.0/jimtcl/jim-tclprefix.c
openocd-0.9.0/jimtcl/jim-eventloop.c
openocd-0.9.0/jimtcl/jim-eventloop.h
openocd-0.9.0/jimtcl/.indent.pro
openocd-0.9.0/jimtcl/tcltest.tcl
openocd-0.9.0/jimtcl/jimsh.c
openocd-0.9.0/jimtcl/jim-format.c
openocd-0.9.0/config.h.in
openocd-0.9.0/README.OSX
openocd-0.9.0/AUTHORS
openocd-0.9.0/INSTALL
openocd-0.9.0/ChangeLog
openocd-0.9.0/Doxyfile.in
openocd-0.9.0/COPYING
joaquin@ubuntu:~$
```

Figura 20: Descompresión del paquete del OpenOCD

Finalmente, compilamos OpenOCD para que funcione con nuestro chip FT2232 con estos cuatro pasos:

cd openocd-0.9.0

./configure --enable-ftdi

make

sudo make install

En las Figura 21, 22 y 23 se muestran las correspondientes capturas del terminal de Linux al finalizar los últimos 3 comandos arriba mencionados.

```
joaquin@ubuntu: ~/openocd-0.9.0
jim-config.h is unchanged
jimautoconf.h is unchanged
Created Makefile from Makefile.in
Created build-jim-ext from build-jim-ext.in

OpenOCD configuration summary
-----
MPSSE mode of FTDI based devices          yes
ST-Link JTAG Programmer                   yes (auto)
TI ICDI JTAG Programmer                   yes (auto)
Keil ULINK JTAG Programmer                yes (auto)
Altera USB-Blaster II Compatible          yes (auto)
Versaloon-Link JTAG Programmer            yes (auto)
Segger J-Link JTAG Programmer             yes (auto)
OSBDM (JTAG only) Programmer             yes (auto)
eStick/opendous JTAG Programmer           yes (auto)
Andes JTAG Programmer                     yes (auto)
USBProg JTAG Programmer                   yes (auto)
Raisonance RLink JTAG Programmer          yes (auto)
Olimex ARM-JTAG-EW Programmer             yes (auto)
CMSIS-DAP Compliant Debugger              no
joaquin@ubuntu:~/openocd-0.9.0$
```

Figura 21: Captura del Terminal cuando finaliza la instrucción configure

```
joaquin@ubuntu: ~/openocd-0.9.0
libtool: link: ( cd ".libs" && rm -f "libopenocd.la" && ln -s "../libopenocd.la" "libopenocd.la" )
depbase='echo main.o | sed 's|^[^/]*$|.deps/&|s|.o$||'`; \
gcc -std=gnu99 -DHAVE_CONFIG_H -I. -I.. -I../src -I../src/helper -DPKGDATA_DIR=\"/usr/local/s
hare/openocd\" -DBINDIR=\"/usr/local/bin\" -I../jimtcl -I../jimtcl -g -O2 -Wall -Wstrict-prototypes -Wformat
-security -Wshadow -Wextra -Wno-unused-parameter -Wbad-function-cast -Wcast-align -Wredundant-decls -Werror -M
T main.o -MD -MP -MF $depbase.Tpo -c -o main.o main.c && \
mv -f $depbase.Tpo $depbase.Po
/bin/bash ../libtool --tag=CC --mode=link gcc -std=gnu99 -g -O2 -Wall -Wstrict-prototypes -Wformat-securit
y -Wshadow -Wextra -Wno-unused-parameter -Wbad-function-cast -Wcast-align -Wredundant-decls -Werror -o openo
cd main.o libopenocd.la ../jimtcl/libjim.a -lm -ldl
libtool: link: gcc -std=gnu99 -g -O2 -Wall -Wstrict-prototypes -Wformat-security -Wshadow -Wextra -Wno-unused-
parameter -Wbad-function-cast -Wcast-align -Wredundant-decls -Werror -o openocd main.o ../libs/libopenocd.a -
lusb -lusb-1.0 ../jimtcl/libjim.a -lm -ldl
make[4]: se sale del directorio «/home/joaquin/openocd-0.9.0/src»
make[3]: se sale del directorio «/home/joaquin/openocd-0.9.0/src»
make[2]: se sale del directorio «/home/joaquin/openocd-0.9.0/src»
Making all in doc
make[2]: se ingresa al directorio «/home/joaquin/openocd-0.9.0/doc»
make[2]: No se hace nada para «all».
make[2]: se sale del directorio «/home/joaquin/openocd-0.9.0/doc»
make[2]: se ingresa al directorio «/home/joaquin/openocd-0.9.0»
make[2]: se sale del directorio «/home/joaquin/openocd-0.9.0»
make[1]: se sale del directorio «/home/joaquin/openocd-0.9.0»
joaquin@ubuntu:~/openocd-0.9.0$
```

Figura 22: Captura del Terminal cuando finaliza la instrucción make

```
joaquin@ubuntu: ~/openocd-0.9.0
/bin/mkdir -p '/usr/local/share/man/man1'
/usr/bin/install -c -m 644 openocd.1 '/usr/local/share/man/man1'
make[2]: se sale del directorio «/home/joaquin/openocd-0.9.0/doc»
make[1]: se sale del directorio «/home/joaquin/openocd-0.9.0/doc»
make[1]: se ingresa al directorio «/home/joaquin/openocd-0.9.0»
make[2]: se ingresa al directorio «/home/joaquin/openocd-0.9.0»
make[2]: No se hace nada para «install-exec-am».
/bin/mkdir -p '/usr/local/share/openocd'
/bin/mkdir -p '/usr/local/share/openocd/contrib'
/usr/bin/install -c -m 644 contrib/99-openocd.rules '/usr/local/share/openocd/contrib'
/bin/mkdir -p '/usr/local/share/openocd/contrib/libdcc'
/usr/bin/install -c -m 644 contrib/libdcc/dcc_stdio.c contrib/libdcc/dcc_stdio.h contrib/libdcc/example.c co
ntrib/libdcc/README '/usr/local/share/openocd/contrib/libdcc'
make install-data-hook
make[3]: se ingresa al directorio «/home/joaquin/openocd-0.9.0»
for i in $(find ./tcl -name '*.cfg' -o -name '*.tcl' -o -name '*.txt' | sed -e 's,^./tcl,,'); do \
    j="/usr/local/share/openocd/scripts/$i" && \
    mkdir -p "$(dirname $j)" && \
    /usr/bin/install -c -m 644 ./tcl/$i $j; \
done
make[3]: se sale del directorio «/home/joaquin/openocd-0.9.0»
make[2]: se sale del directorio «/home/joaquin/openocd-0.9.0»
make[1]: se sale del directorio «/home/joaquin/openocd-0.9.0»
joaquin@ubuntu:~/openocd-0.9.0$
```

Figura 23: Captura del Terminal cuando finaliza la instrucción make install

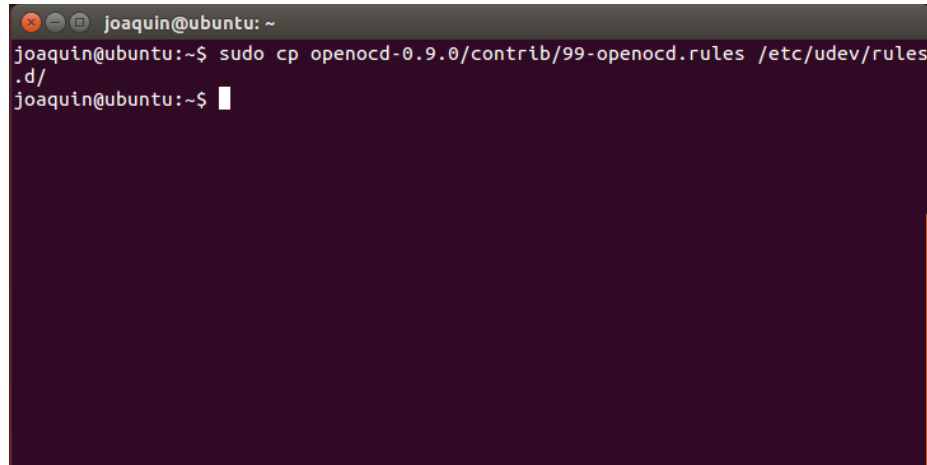
Lo último que debemos hacer es copiar los permisos de OpenOCD en las reglas de *udev* de Linux. Para ello, desde el terminal, ejecutamos el siguiente comando:

sudo cp /home/user/openocd-0.9.0/contrib/99-openocd.rules /etc/udev/rules.d/

donde la dirección */home/user/* corresponde al path donde descomprimimos el paquete descargado del OpenOCD. Si lo descargamos en otro lugar, debemos reemplazar esta cadena por la que corresponda. En este caso, *user* deberá reemplazarse por el nombre de usuario activo.

El punto anterior es importante, dado que si no lo realizamos, el OpenOCD, al ejecutarse desde Eclipse, nos dará un error desde consola, diciendo que no encuentra el dispositivo al cuál deseamos conectarnos.

La respuesta de la ventana de comandos se puede ver en la Figura 24.



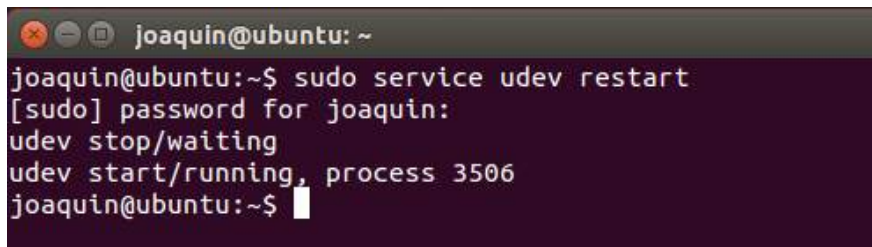
```
joaquin@ubuntu: ~  
joaquin@ubuntu:~$ sudo cp openocd-0.9.0/contrib/99-openocd.rules /etc/udev/rules  
.d/  
joaquin@ubuntu:~$
```

Figura 24: Ventana de comandos luego de ejecutar copiar los permisos del OpenOCD

Lo último que debemos hacer es reiniciar los servicios udev, para dejarlos funcionales. Ello no hacemos copiando el siguiente comando en la consola :

sudo service udev restart

y la consola correspondiente que nos debería aparecer se muestra en la Figura 25.



```
joaquin@ubuntu: ~  
joaquin@ubuntu:~$ sudo service udev restart  
[sudo] password for joaquin:  
udev stop/waiting  
udev start/running, process 3506  
joaquin@ubuntu:~$
```

Figura 25: Consola luego de reiniciar los servicios udev

Con esto, ya tenemos el complemento OpenOCD listo para funcionar.

Un comentario adicional que podemos hacer es que el chip FT2232H posee 2 canales de comunicación independientes (A y B) pero que ambos salen por el mismo USB, es decir, la PC verá 2 dispositivos distintos (en realidad uno compuesto). Uno será el que conecta al JTAG manejado por OpenOCD como fue mencionado, y el otro se verá como un puerto serie por USB. Este puerto debería encontrarse en *'/dev/ttyUSB0'*, y puede servir para principalmente para debug.

3.4 IDE - Eclipse

El IDE utilizado para codificar el Firmware en 'C' es Eclipse CDT (C/C++). La versión correspondiente puede descargarse de acá: [CDT 8.3.0 for Eclipse Kepler SR2](#) (se han encontrado problemas con la versión más reciente de Eclipse 'Luna-R' usado con el plug-in para GNU-ARM-OpenOCD, por lo tanto a la fecha no es recomendable su uso). Si se sigue el link marcado arriba, se redireccionará a la página de Eclipse, donde se debe elegir la versión correspondiente al sistema operativo utilizado. Al día de la fecha, la página se ve como se muestra en la Figura 26, y en la parte superior derecha, se pueden ver los links de descarga del software-IDE Eclipse Kepler.

La primer herramienta que debemos instalar es el Oracle Java JDK. Para ello tenemos que seguir los siguientes pasos:

1) para desinstalar versiones anteriores de Open JDK, ejecutamos el siguiente comando:

```
sudo apt-get purge openjdk*
```

2) Ejecutar el siguiente comando

```
sudo add-apt-repository ppa:webupd8team/java
```

y luego actualizar:

```
sudo apt-get update
```

3) Por último, para instalar la versión 8 de Java, escribimos la siguiente línea:

```
sudo apt-get install oracle-java8-installer
```

Para más detalles sobre esta instalación, se puede consultar el siguiente [link](#).

Una vez hecho esto, podemos proceder a la instalación de Eclipse-Kepler.

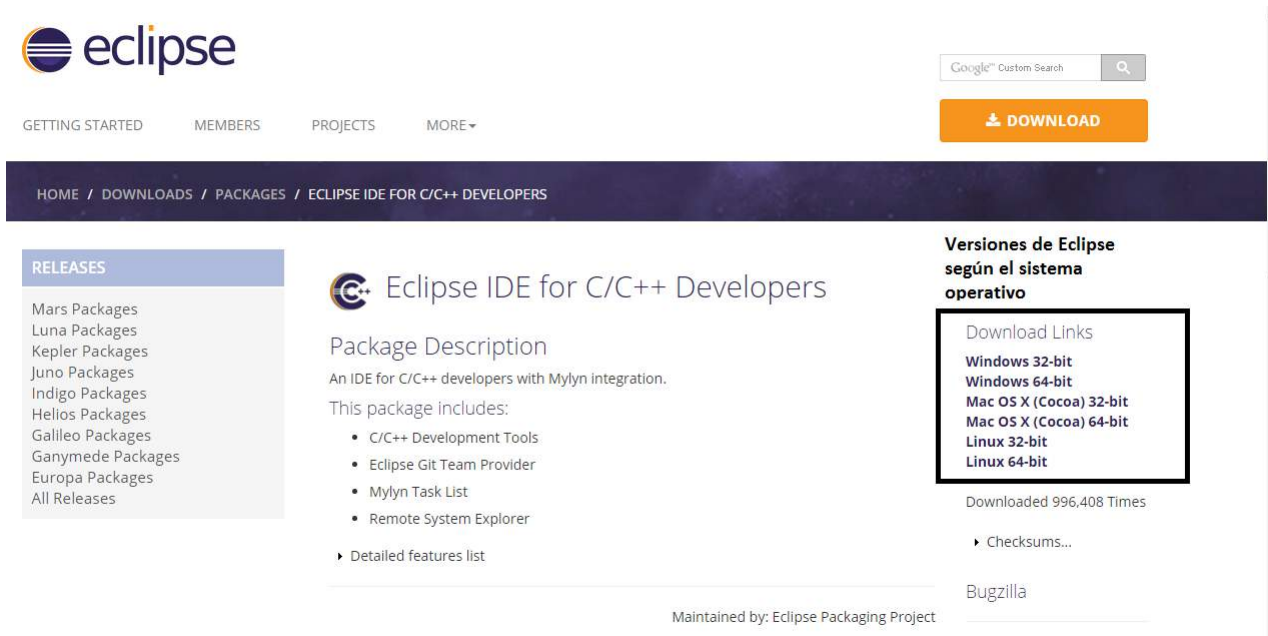


Figura 26: Página oficial del software IDE Eclipse Kepler

Una vez descargado el archivo correspondiente, lo ubicamos en alguna carpeta conocida, y desde el terminal, y de la manera que se explica en la **Sección de análisis**, nos posicionamos sobre esa ubicación. Si por ejemplo, lo descargamos en el Escritorio, entonces para ubicarnos en esa carpeta tenemos que teclear:

`cd /home/user/Escritorio`

donde *user* representa el usuario actual. Para este caso, si queremos ver los elementos que hay en el Escritorio desde el terminal, escribimos *ls*, y veremos algo parecido a lo que se muestra en la Figura 27.

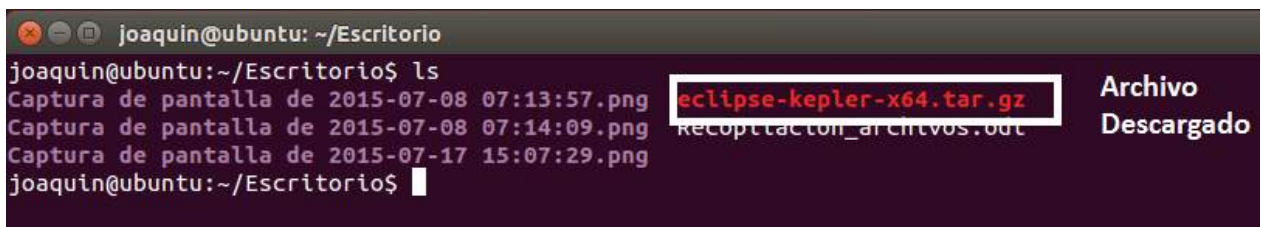


Figura 27: Archivos ubicados en el Escritorio luego de la descarga del Eclipse

Este programa no necesita instalarse, solamente necesita que estén instaladas un par de herramientas, por lo que solamente con descomprimirlo es suficiente para poder usarlo. Para el caso particular del fichero **eclipse-kepler-x64.tar.gz**, vamos a descomprimirlo en la carpeta **/opt/** de la siguiente manera:


```
cd /opt/ && sudo tar -zxvf home/user/Escritorio/eclipse-kepler-x64.tar.gz
```

donde *user* lo reemplazamos por el nombre del usuario activo.

Ahora, si abrimos el programa, lo primero que nos preguntará (al igual que en Windows) es el lugar en donde queremos guardar nuestra información sobre espacio de trabajo (Workspace). En la Figura 28 podemos ver dicha ventana, y en nuestro caso particular, vamos a dejar el valor por defecto.

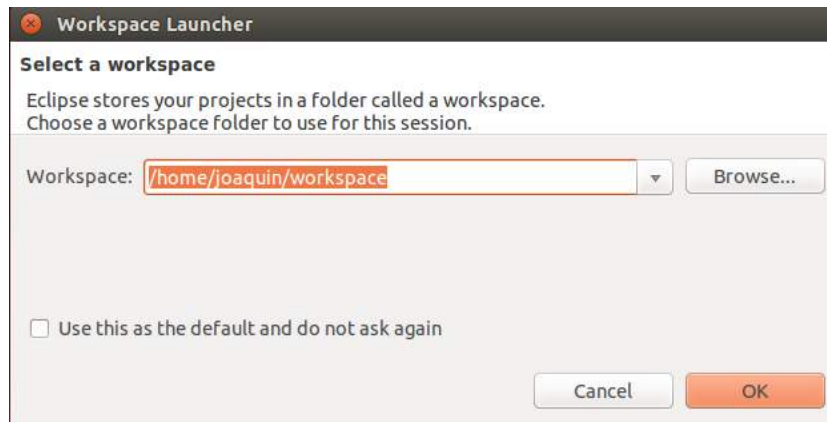


Figura 28: Ventana inicial de Eclipse. Ubicación del Workspace

Luego de apretar en el botón OK, nos aparecerá un entorno similar al de la Figura 29.



Figura 29: Entorno IDE Eclipse-Kepler

3.5 Plug-In GNU-ARM-OpenOCD para Eclipse

Una de las primeras cosas que debemos hacer con nuestro entorno IDE es agregarle los Plug-in para poder debuggear con nuestra placa utilizando el OpenOCD. Para ello, abrimos el Eclipse, y vamos al menú *Help* → *Install New Software...* Nos aparecerá una ventana como la de la Figura 30.

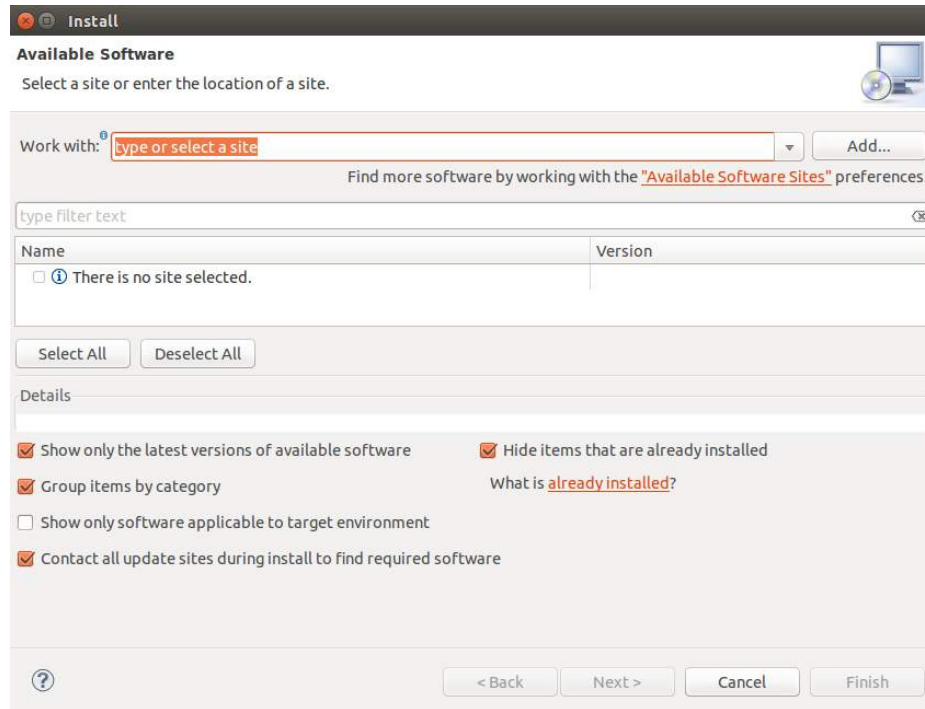


Figura 30: Ventana de instalación de nuevos Plug-In

En la ventana *Work With:* escribimos el siguiente sitio web:

<http://gnuarmeclipse.sourceforge.net/updates>

Y si apretamos la tecla ENTER, comenzará a buscar los distintos Plug-in compatibles con el entorno. Una vez finalizada la búsqueda, nos aparecerá una lista de todos los complementos encontrados, compatibles con la plataforma Eclipse que estemos utilizando, tal como se muestra en la Figura 31.

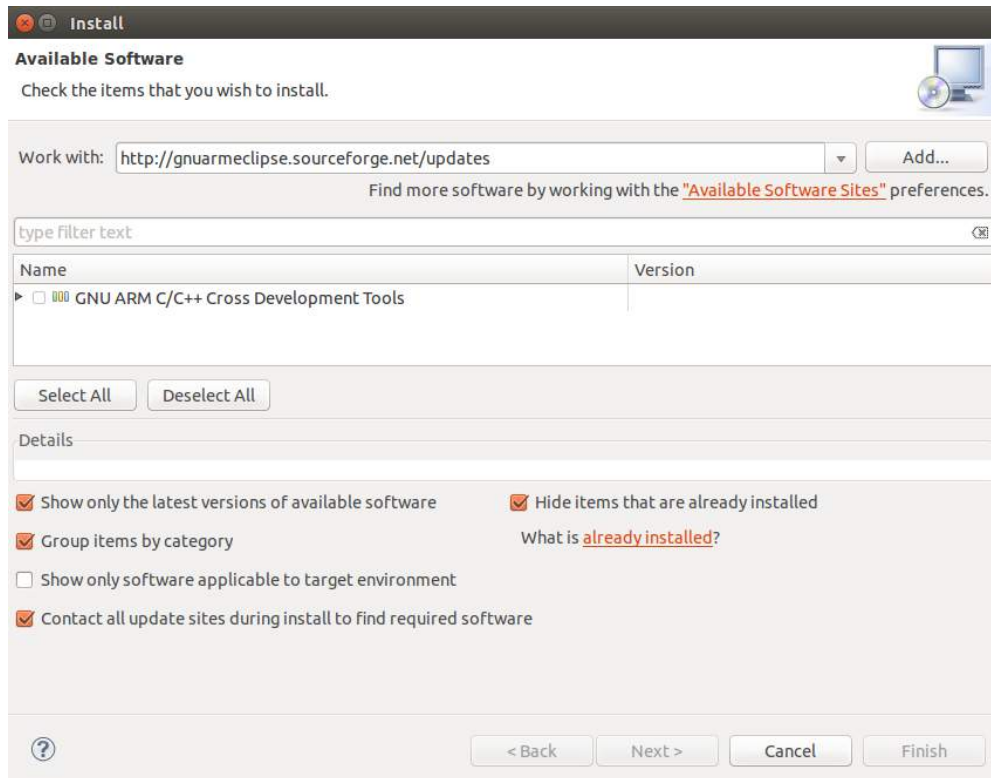


Figura 31: Búsqueda de Plug-ins para Eclipse

Si todo salió bien, en la lista de complementos deberíamos encontrar aquella que se llama '***GNU ARM C/C++ Cross Development Tools***'. Si no es así, puede que aparezca una ventana de error, indicando que no se encontró ningún complemento. Esto se puede deber a alguna de las siguientes razones:

1. Escribimos mal la dirección en donde el IDE debe buscar los plug-ins (posiblemente, debido a algún espacio en la dirección URL), por lo cual se recomienda revisar cuidadosamente lo que escribimos en la línea ***Work With***.
2. Dado que estamos descargando información desde un servidor, que además se actualiza frecuentemente, puede ocurrir que cuando deseemos descargar el complemento, el servidor está caído, o en el peor de los casos, se haya mudado de dominio, o ya no exista más. En ese caso, se puede verificar qué es lo que ocurre, abriendo algún explorador de internet (Chrome, Firefox, etc), copiar el URL arriba mencionado, y ver si en la página se encuentra algún comentario sobre alguna modificación, o estado del servidor.

Luego, si tildamos el rectángulo ubicado a la izquierda de la lista '***GNU ARM C/C++ Cross Development Tools***', estaremos seleccionando todos los componentes incluidos en ella. Para ver el detalle de cuáles son dichos complementos, hacemos click sobre el Triángulo ubicado debajo de ***Name***. Para verificar que estamos instalando el complemento que necesitamos, que es el '***GNU ARM C/C++ OpenOCD Debugging***'. Tenemos que verificar que dicho plug-in esté

tildado, como se puede ver en la Figura 32. Una vez encontrado y tildado, hacemos click en el botón *Next*, ubicado en la parte inferior derecha de la ventana.

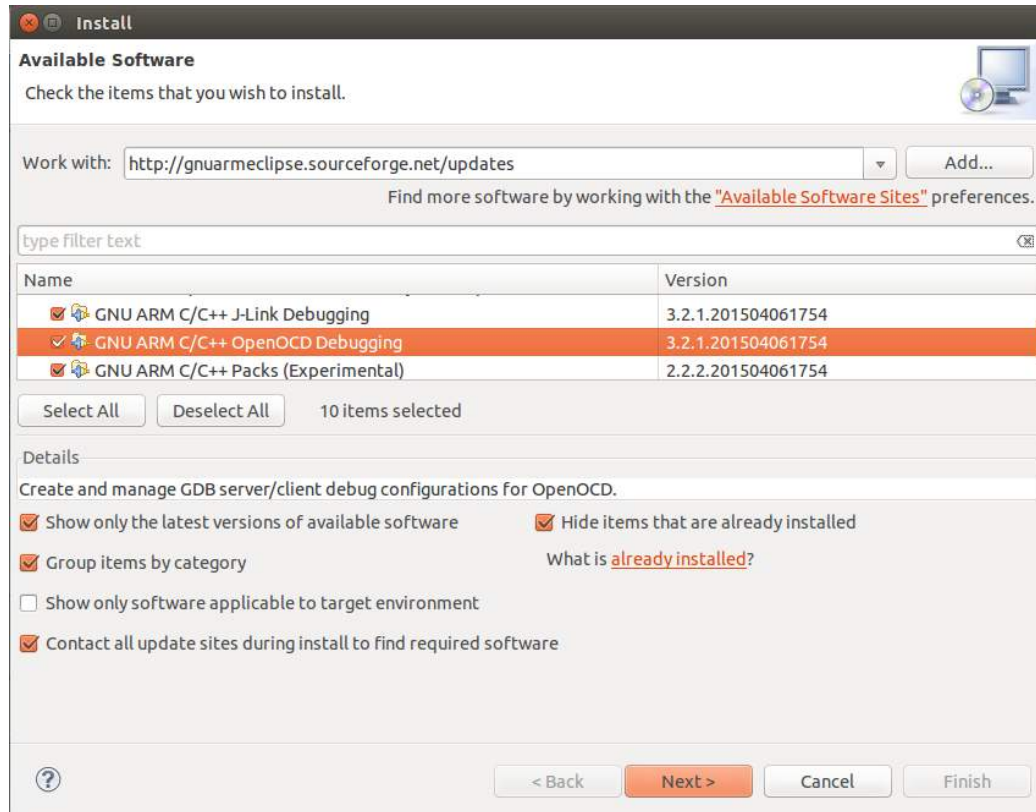


Figura 32: Búsqueda del complemento GNU ARM C/C++ OpenOCD Debugging

Luego, nos aparecerá una lista con el detalle de todos los complementos de Eclipse que se van a instalar (son los mismos que seleccionamos en la ventana anterior). Ésta se puede ver en la Figura 33. Directamente, si estamos de acuerdo, hacemos click en el botón *Next*.

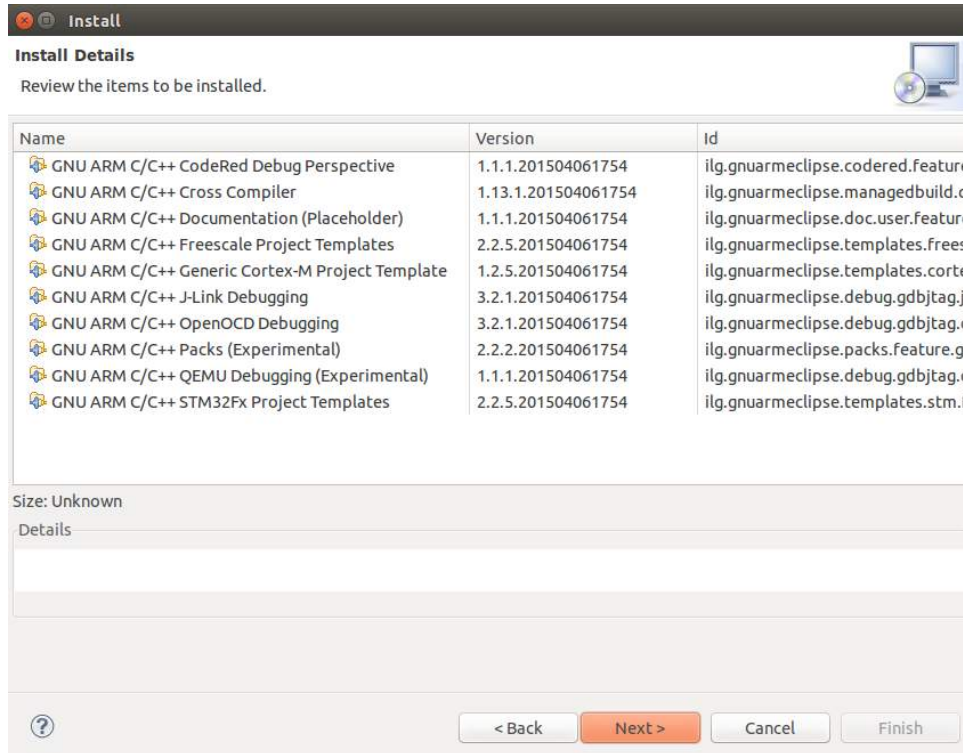


Figura 33: Detalle de los Plug-ins que se van a instalar

En la última ventana, nos mostrará el Acuerdo de Revisión (Review Licenses) de los complementos. Si estamos de acuerdo con todo ello, tildamos la opción '***I accept the terms of the license agreement***', y por último al botón ***Next***. Si no aprobamos el acuerdo de licencia, no podremos continuar con la instalación. La ventana a la que nos referimos, se muestra en la Figura 34.

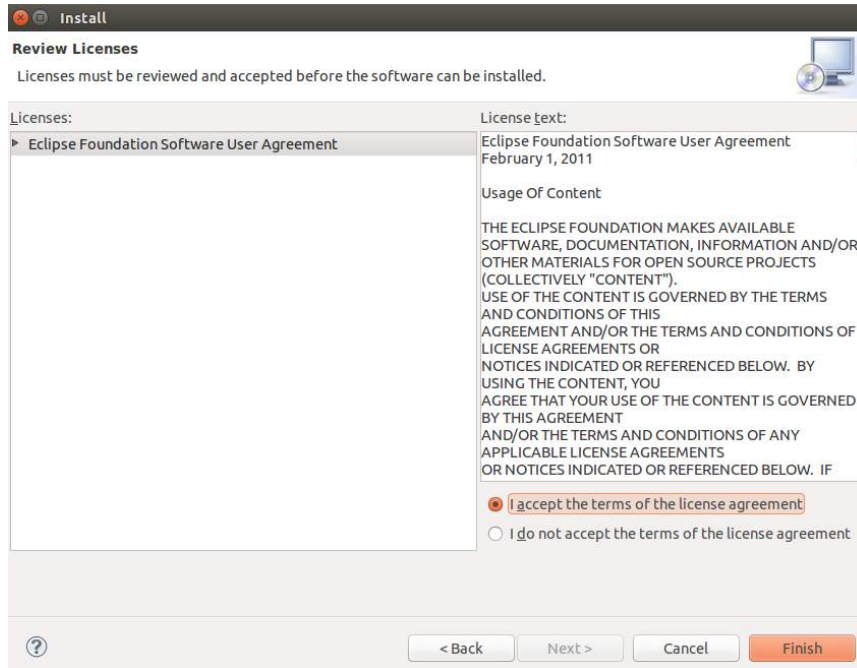


Figura 34: Acuerdo de licencias de los plug-ins a instalar

Al oprimir el botón Finish, comenzará la instalación, por lo que nos aparecerá una ventana como la que se ve en la Figura 35. Aquí, debemos esperar a que finalice.

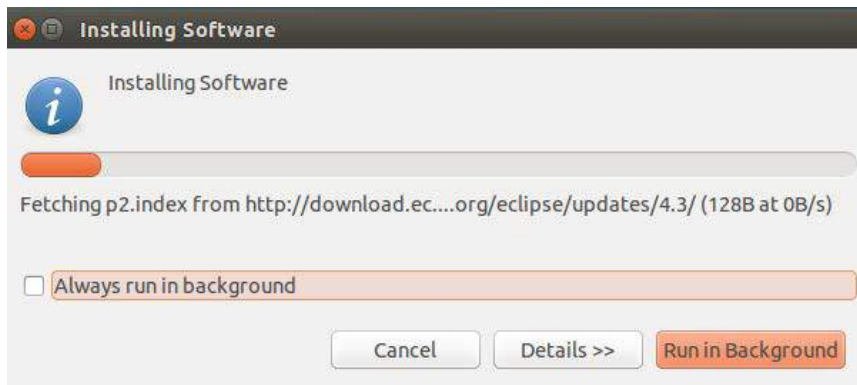


Figura 35: Progreso de la instalación de los plug-ins

Puede ocurrir que alguno de los complementos no tenga los certificados de verificación de identidad de software que Linux/ Ubuntu solicita para corroborar que el software no sea Mal Intencionado, en ese caso, nos aparecerá una advertencia como la que se ve en la Figura 36.

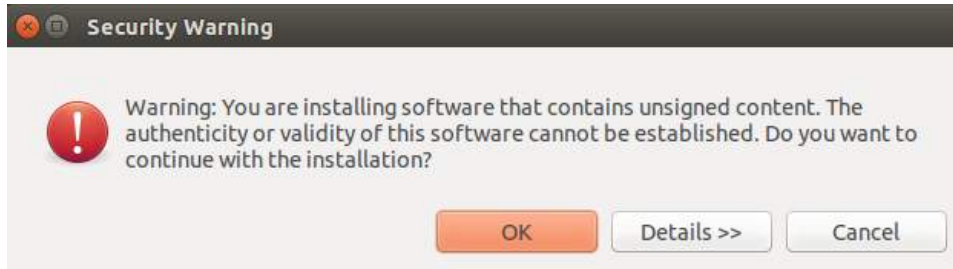


Figura 36: Advertencia de seguridad de Linux

Como nosotros sabemos que el software que estamos instalando no es un virus, hacemos click en **Ok** y de esa forma, puede continuar la instalación.

Al finalizar, aparecerá una ventana que pide Reiniciar Eclipse para que los cambios tengan efecto, como se muestra en la Figura 37. Hacemos click en **Yes** y con todo esto, ya tenemos nuestro entorno IDE en condiciones de ser utilizado para el desarrollo de software para el proyecto CIAA en Ubuntu.

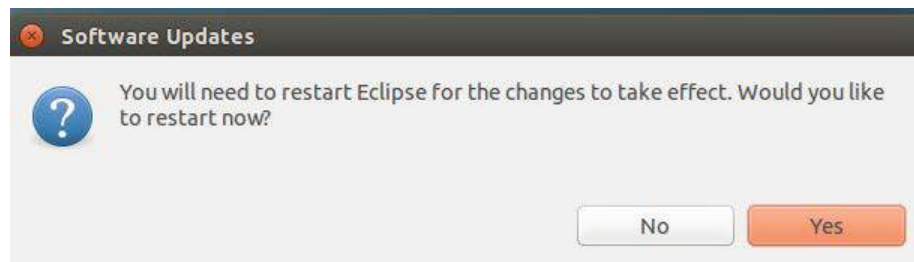


Figura 37: Actualización de Software. Petición para reiniciar Eclipse

3.6 Clonando el repositorio del proyecto CIAA en nuestra PC

3.6.1 Instalar GIT

Para poder trabajar con GitHub en Ubuntu, tenemos que instalar el programa correspondiente, tal como se detalla en los siguientes pasos:

1) Descarga con GIT Shell (consola desde Linux)

Si no tenemos instalado el GIT en nuestra PC, debemos ir al Terminal de Linux, y escribir la siguiente sentencia:

sudo apt-get install git

Como en la mayoría de las instalaciones que realizamos a lo largo de este tutorial, nos preguntará la contraseña para conectarse como superusuario, y luego una confirmación de instalación, como se ve en la Figura 38.

```
joaquin@ubuntu: ~
joaquin@ubuntu:~$ sudo apt-get install git
[sudo] password for joaquin:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Los paquetes indicados a continuación se instalaron de forma automática y ya no son necesarios.
 libllvm3.4 libnewlib-arm-none-eabi libnewlib-dev
Use 'apt-get autoremove' to remove them.
Se instalarán los siguientes paquetes extras:
 git-man liberror-perl
Paquetes sugeridos:
 git-daemon-run git-daemon-sysvinit git-doc git-el git-email git-gui gitk
 gitweb git-arch git-bzr git-cvs git-mediawiki git-svn
Se instalarán los siguientes paquetes NUEVOS:
 git git-man liberror-perl
0 actualizados, 3 se instalarán, 0 para eliminar y 265 no actualizados.
Se necesita descargar 3.325 kB/3.346 kB de archivos.
Se utilizarán 21,6 MB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n]
```

Figura 38: Instalación del GIT Shell. Confirmación de instalación

Si finaliza correctamente, la ventana final de esta instalación se verá como la Figura 39.

```
joaquin@ubuntu: ~
Se necesita descargar 3.325 kB/3.346 kB de archivos.
Se utilizarán 21,6 MB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] s
Des:1 http://us.archive.ubuntu.com/ubuntu/ trusty-updates/main git-man all 1:1.9.1-1ubuntu0.1 [698 kB]
Des:2 http://us.archive.ubuntu.com/ubuntu/ trusty-updates/main git amd64 1:1.9.1-1ubuntu0.1 [2.627 kB]
Descargados 3.325 kB en 27seg. (122 kB/s)
Seleccionando el paquete liberror-perl previamente no seleccionado.
(Leyendo la base de datos ... 187276 ficheros o directorios instalados actualmente.)
Preparing to unpack ../liberror-perl_0.17-1.1_all.deb ...
Unpacking liberror-perl (0.17-1.1) ...
Seleccionando el paquete git-man previamente no seleccionado.
Preparing to unpack ../git-man_1%3a1.9.1-1ubuntu0.1_all.deb ...
Unpacking git-man (1:1.9.1-1ubuntu0.1) ...
Seleccionando el paquete git previamente no seleccionado.
Preparing to unpack ../git_1%3a1.9.1-1ubuntu0.1_amd64.deb ...
Unpacking git (1:1.9.1-1ubuntu0.1) ...
Processing triggers for man-db (2.6.7.1-1ubuntu1) ...
Configurando liberror-perl (0.17-1.1) ...
Configurando git-man (1:1.9.1-1ubuntu0.1) ...
Configurando git (1:1.9.1-1ubuntu0.1) ...
joaquin@ubuntu:~$
```

Figura 39: Instalación del GIT Shell. Fin de la instalación

2) Clonar el repositorio:

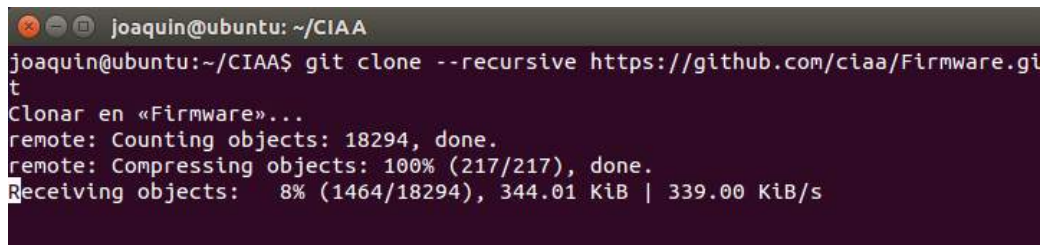
Vamos a utilizar un directorio ubicado en nuestra carpeta de usuario. A modo de ejemplo, lo llamaremos *User*. Entonces, en el terminal escribimos los siguientes comandos:

Comando	Consecuencia
cd \$HOME	Dirige la ubicación actual al interior de la carpeta del usuario
mkdir CIAA	Crea el directorio CIAA en el directorio del usuario <i>User</i>
cd CIAA	Ingresa al directorio CIAA

Ahora, si nos quedamos sobre el directorio CIAA, el comando que sigue copiará el repositorio GitHub del proyecto CIAA en nuestra PC, en la carpeta que acabamos de crear:

git clone --recursive https://github.com/ciaa/Firmware.git

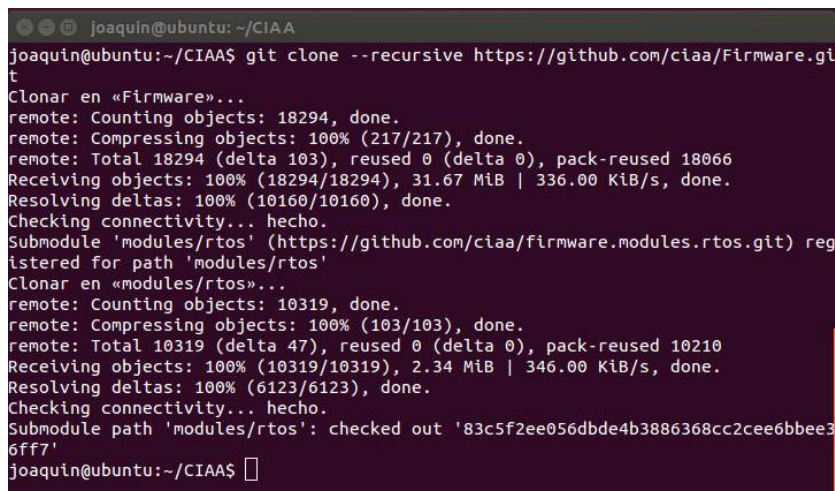
y el repositorio comenzará a descargarse, como muestra en la Figura 40.



```
joaquin@ubuntu: ~/CIAA
joaquin@ubuntu:~/CIAA$ git clone --recursive https://github.com/ciaa/Firmware.git
Clonar en «Firmware»...
remote: Counting objects: 18294, done.
remote: Compressing objects: 100% (217/217), done.
Receiving objects: 8% (1464/18294), 344.01 KiB | 339.00 KiB/s
```

Figura 40: Comienzo de la descarga del repositorio del proyecto CIAA

Cuando finalice, veremos la imagen de la consola parecida a la que se presenta en la Figura 41.



```
joaquin@ubuntu: ~/CIAA
joaquin@ubuntu:~/CIAA$ git clone --recursive https://github.com/ciaa/Firmware.git
Clonar en «Firmware»...
remote: Counting objects: 18294, done.
remote: Compressing objects: 100% (217/217), done.
remote: Total 18294 (delta 103), reused 0 (delta 0), pack-reused 18066
Receiving objects: 100% (18294/18294), 31.67 MiB | 336.00 KiB/s, done.
Resolving deltas: 100% (10160/10160), done.
Checking connectivity... hecho.
Submodule 'modules/rtos' (https://github.com/ciaa/firmware.modules.rtos.git) registered for path 'modules/rtos'
Clonar en «modules/rtos»...
remote: Counting objects: 10319, done.
remote: Compressing objects: 100% (103/103), done.
remote: Total 10319 (delta 47), reused 0 (delta 0), pack-reused 10210
Receiving objects: 100% (10319/10319), 2.34 MiB | 346.00 KiB/s, done.
Resolving deltas: 100% (6123/6123), done.
Checking connectivity... hecho.
Submodule path 'modules/rtos': checked out '83c5f2ee056dbde4b3886368cc2cee6bbe36ff7'
joaquin@ubuntu:~/CIAA$
```

Figura 41: Fin de la descarga del repositorio del proyecto CIAA

3) Creando el archivo MakeFile.mine

Para facilitar la tarea del usuario, el proyecto cuenta con dos Makefile: uno que es el general, y otro que es destinado al usuario. Este último tiene por nombre *Makefile.mine*, y no está incluido en el repositorio clonado. Si intentamos compilar el proyecto sin este archivo, el Makefile general tomará por defecto el *Makefile.config*. Se recomienda fuertemente, no modificar este archivo, y toda modificación que se haga, sea sobre el *Makefile.mine*.

Para crearlo, lo único que debe hacerse es ir a la carpeta firmware desde el explorador de Ubuntu (Nautilus), y buscar el archivo *Makefile.config*. Ahí mismo se hace una copia de éste, y se le cambia el nombre por *Makefile.mine*. Estos pasos se grafican en la Figura 42.

IMPORTANTE: En todo lo anterior se han colocado las capturas de la ventana de comandos para mostrar las últimas acciones que realiza cada comando, y compararlas con aquellas que realiza el lector. En el caso de que algo falle, ya sea por falta de permisos del usuario para realizar un comando, o que la página especificada no exista, o que no estén instalados los complementos previos necesarios para el programa que se instala, la consola emitirá un *mensaje de error* indicando *exactamente* el problema y *cómo solucionarlo*. Es por ello que se le recomienda al lector *leer detenidamente las últimas líneas emitida por la consola* para estar seguro de que lo que se está haciendo está funcionando.



Figura 42: Creación del archivo de usuario Makefile.mine

4 Configuración del entorno ECLIPSE-IDE

4.1 Elección del Workspace

Al iniciar el entorno IDE, éste solicita que seleccione una carpeta de trabajo, como se muestra en la Figura 43:

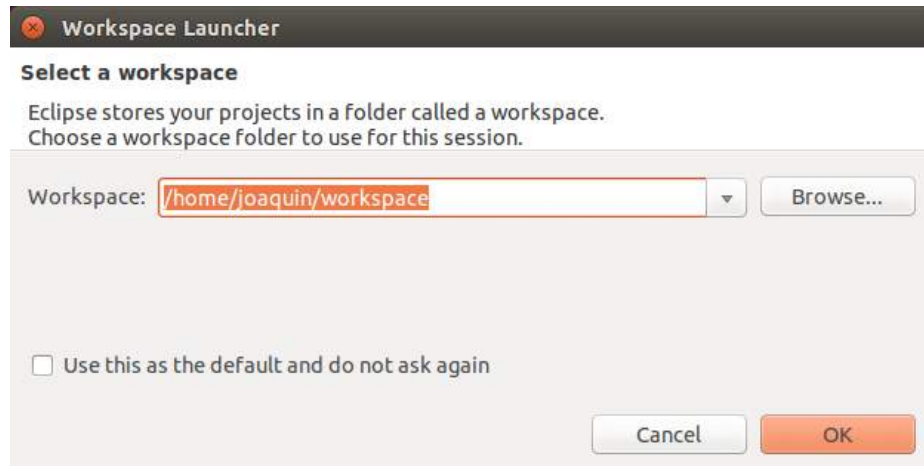


Figura 43: Ventana inicial para ubicar el espacio de trabajo (Workspace)

Todo lo relativo al proyecto en que se esté trabajando (configuraciones, variables globales del entorno y demás) se guarda en esta carpeta. Si siempre se va a utilizar la misma ubicación, se puede tildar la opción “*Use this as the default and do not ask again*”. En cualquier otro caso, cada vez que se abra el entorno, se preguntará por esta ubicación. En particular puede elegirse dentro del directorio de instalación del CIAA-IDE sin ningún problema.

4.2 Primeros Pasos: El Proyecto “Blinking”

Para el usuario sin experiencia con esta plataforma de desarrollo, se aconseja abrir el proyecto del Firmware, para luego compilar y ejecutar un proyecto de ejemplo, llamado *Blinking*. Para ello debemos ir al menú '*File* → *New* → *Makefile Project with Existing Code*'.

Después de haber seleccionado para crear un nuevo proyecto, aparecerá una ventana (Figura 44) en donde se debe indicar el proyecto que se va a cargar: en este caso elegiremos la carpeta Firmware que, si usamos los directorios por defecto, se encuentra en la ubicación:

/home/USER/CIAA/Firmware

Donde *USER* debe reemplazarse por el nombre de usuario activo.

En la ventana *Toolchain for Indexer Settings* seleccionaremos la opción *<none>*, lo cual dejará las opciones por defecto, configuradas en el Makefile.

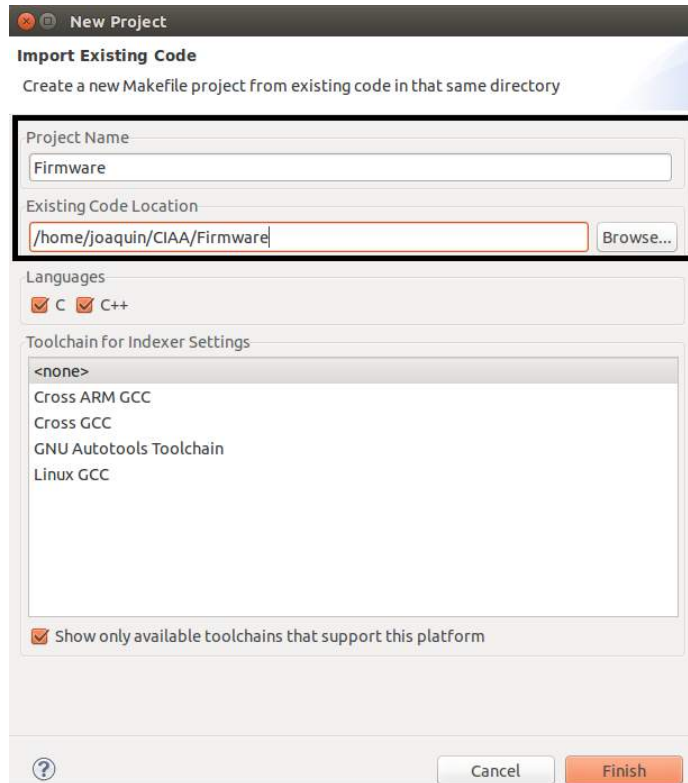


Figura 44: Creación de un nuevo proyecto usando un código existente

Una vez creado el proyecto, cerramos la pestaña *Welcome* con la que inicia Eclipse, y nos encontraremos con un entorno como el que se presenta en la Figura 45.

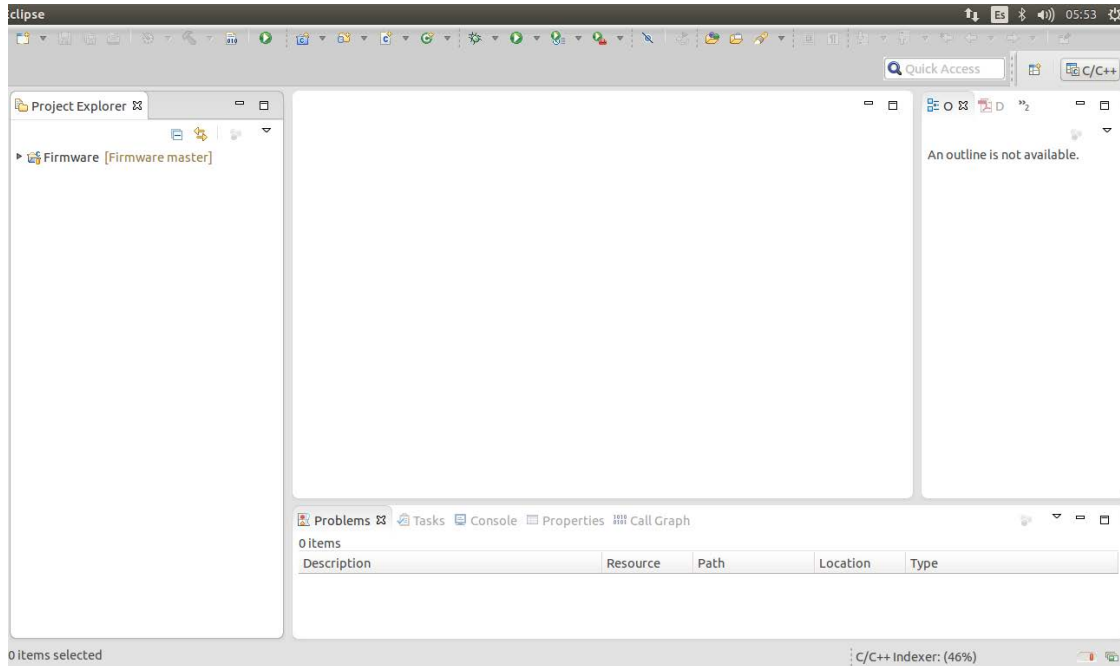


Figura 45: Entorno IDE-CIAA con el proyecto “Blinking” cargado

4.3 Indexación de archivos de cabecera

Para la compilación del proyecto, es necesario que el software IDE sepa dónde encontrar las cabeceras (*Includes*) del estándar *POSIX*.

POSIX (Portable Operating System Interface) es un conjunto de interfaces estándar para sistemas operativos basadas en UNIX. Esta estandarización fue necesaria para que los distintos fabricantes de computadoras o desarrolladores de programas pudieran desarrollar sus aplicaciones independientemente de la plataforma en la cual iba a correr.

Para poder indexar esas definiciones se deben agregar los archivos de cabecera del *GCC (GNU Compiler Collection)*. Esto se efectúa sobre la pestaña *Path and Symbols→Includes*, en una ventana similar a la que muestra la Figura 46.

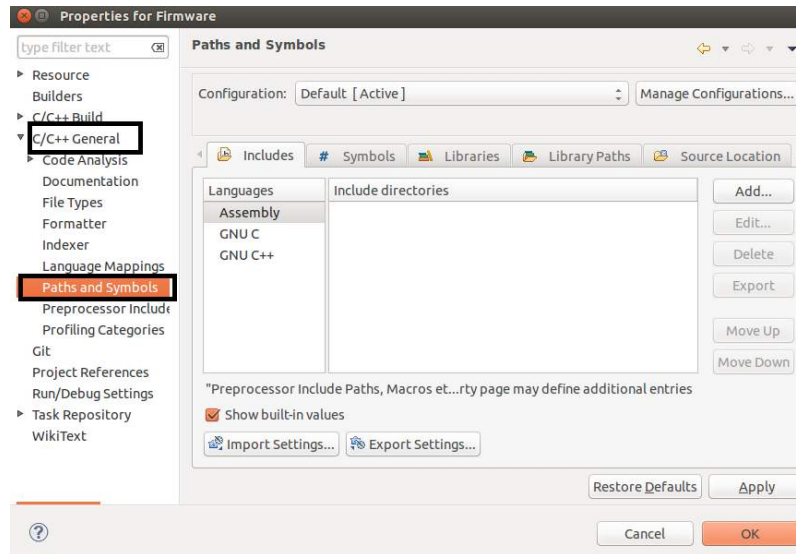


Figura 46: Indexación de las definiciones POSIX

En esta ventana seleccionar '*GNU C*' o '*GNU C++*' según las POSIX que queramos agregar, y luego presionar el botón '*Add*'. Emergerá la ventana mostrada en la Figura 47, en la cual se debe hacer click en el botón '*File System...*' y luego buscar la carpeta correspondiente a agregar.

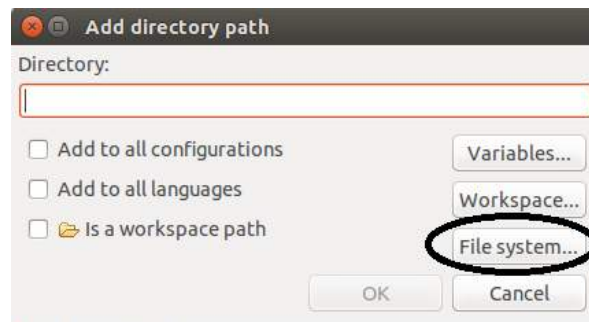


Figura 47: Indexar POSIX

Los *Includes* que deben configurarse (si se siguieron los pasos de la instalación del GCC de ARM), según el lenguaje (*Language*) elegido, son los siguientes:

Language = 'GNU C'

- */usr/arm-none-eabi/include*

Language = 'GNU C++'

- */usr/arm-none-eabi/include*

Finalizada la selección, los **Includes** quedarán de forma similar a la mostrada en la Figura 48.

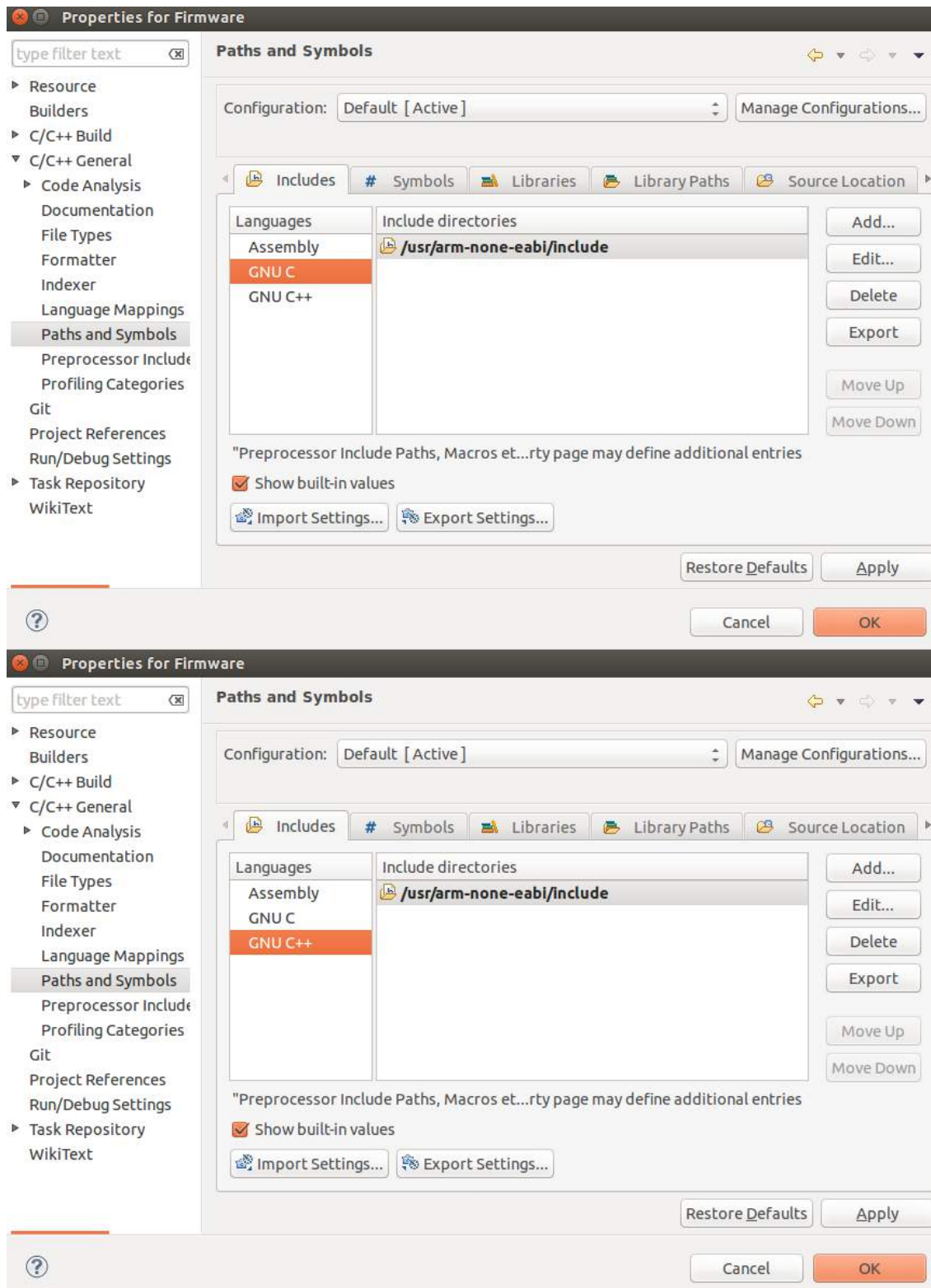


Figura 48: Fin de la configuración de Includes

4.4 Configuración del Makefile

Los archivos *Makefile* son archivos de texto escritos con una sintaxis predeterminedada. Junto con la utilidad '*Make*', permiten construir el software desde sus archivos-fuente, en el sentido de organizar el código, su compilación y enlace (*link*) correcto.

El *Proyecto CIAA* tiene su propio Makefile, por lo que se debe indicarle al IDE cómo manejarse con él: de lo contrario generaría un Makefile automáticamente, lo cual nos traería muchos dolores de cabeza.

Cada vez que hacemos un comando *clean*, estamos borrando los archivos objeto generados previamente, pero antes de poder volver a compilar, necesitamos que esté previamente procesado el código PHP correspondiente al sistema operativo *RTOS-OSEK* (ver nota siguiente). Éste procesado se hace con un comando llamado *generate*. Para no tener que hacer cada función por separado, lo que se hace es pasarle al MakeFile el comando '*clean_generate*', que realiza ambas operaciones, en forma consecutiva y automática. Si se trabaja sin RTOS-OSEK sólo hace falta usar el comando '*clean*'.

La primera vez que se compila el proyecto, es necesario hacer un *Clean Project*. Esto ejecutará el comando *Clean_generate* del make, creando todos los archivos necesarios para la compilación con el RTOS.

NOTA

OSEK(alemán: *Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen*) (inglés: *Open Systems and their Interfaces for the Electronics in Motor Vehicles*) es un gremio de estandarización que ha producido especificaciones para sistemas operativos embebidos, un stack de comunicación, y un protocolo de control de red para un sistema embebido automotriz, además de otras especificaciones relacionadas. OSEK (“*Sistemas abiertos y sus interfaces para la electrónica en automóviles*”) se diseñó para proveer una arquitectura de software abierta estándar para varias unidades de control electrónicas (ECU=Electronic Control Unit) incorporadas en vehículos, para facilitar la integración y portabilidad de software de diferentes proveedores, ahorrando costos y tiempos de desarrollo. Por otra parte, es un sistema operativo muy seguro, dado que todas sus funciones son determinísticas: al momento de iniciar el SO, cada tarea ya tiene asignado su espacio de memoria, evitando la necesidad de contar con instrucciones tipo '*malloc*' para la asignación dinámica de memoria. Así, ante una situación de riesgo (p.ej. al producirse un choque), no se producen retardos en la ejecución de las tareas.

Para realizar esta configuración, ubíquese en la ventana de propiedades del proyecto *Firmware*, y seleccione la rama *C/C++ Build*: se verá una pantalla similar a la que muestra la Figura 49.

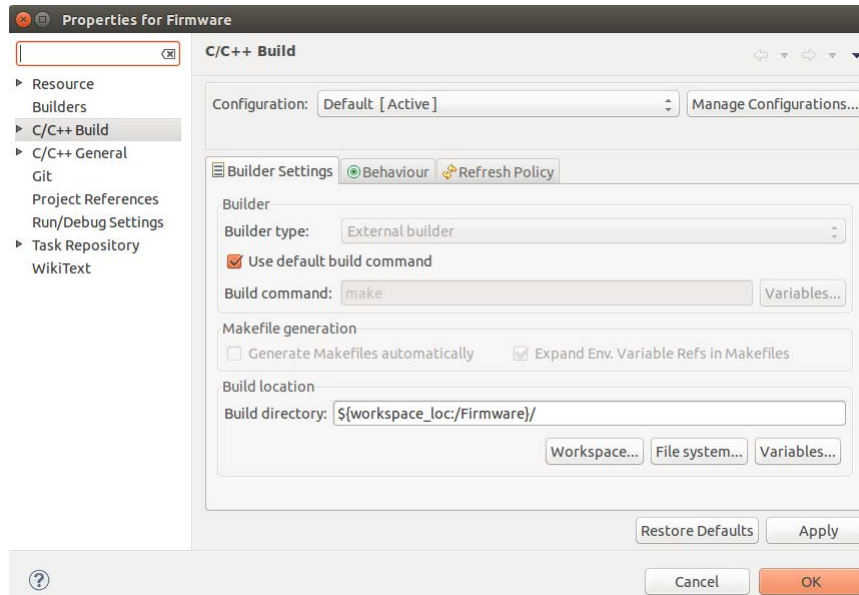


Figura 49: Configuración del Makefile

Dentro de la rama '*C/C++ Build*', configure la pestaña '*Behaviour*' como muestra la Figura 50. Las configuraciones importantes son las siguientes:

- tilde la opción '*Stop on first build error*' y destilde '*Enable parallel build*'
- destilde el casillero '*Build on resource save*' y tilde '*Build (Incremental Build)*' y '*Clean*'. En el campo *Clean*, escriba: *clean_generate*
- borre el contenido del campo *Build* y déjelo en blanco

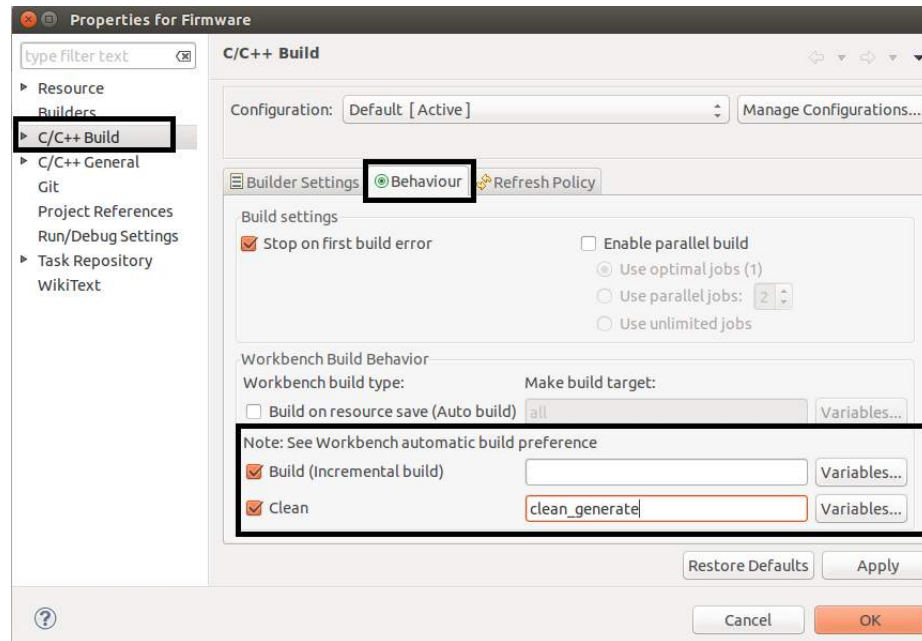


Figura 50: Configuración de comportamiento

Luego de hacer esto, damos presionamos en *Ok*, y luego vamos sobre el proyecto Firmware, hacemos clic derecho, y buscamos la opción *Clean Project*. Por último, damos clic en *Build Project*.

4.5 Debug en placa EDU-CIAA y Entorno IDE

4.5.1 Configuración del entorno CIAA-IDE

Si se cuenta con una placa EDU-CIAA y se quiere depurar sobre el hardware, lo primero que hay que hacer es una limpieza mediante '*Clean Project*'. Esto es necesario porque en todos los *builds* que se hicieron para el Win Debug se crearon archivos que pueden entorpecer la compilación del Debug sobre la placa, pues los mismos estaban pensados para el CPU de la PC, y no para el μC que forma parte de la EDU-CIAA.

A continuación necesitamos que el compilador reconozca que se quiere compilar sobre la placa EDU-CIAA. Ello implica que, en función de la placa disponible, se incluya el código acorde al μC utilizado. Dado que el código cuenta con sentencias *if* de pre-procesado, que dependen del hardware disponible, es necesario indicarle con qué versión de la plataforma se cuenta. Para ello, vamos a modificar manualmente el archivo MakeFile.mine, que es una rama del MakeFile, más reducido, que tiene los datos mínimos y necesarios para compilar. Para hacerlo, desde el software IDE, al mismo nivel que el proyecto Firmware (Figura 51), vamos a encontrar el MakeFile.mine que debemos modificar.

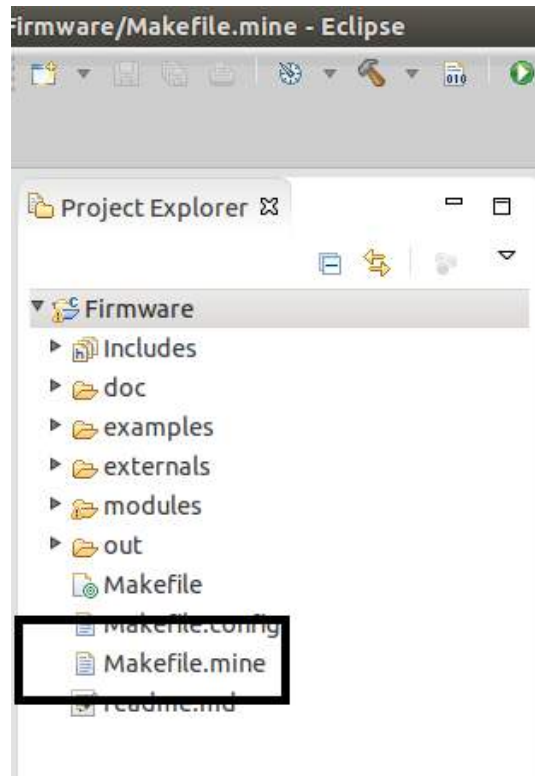


Figura 51: Project Explorer - MakeFile.mine

Le hacemos doble clic y nos va a aparecer a nuestra derecha, un archivo de texto. Si bajamos un poco, vamos a encontrar el código que se muestra en la Figura 52.

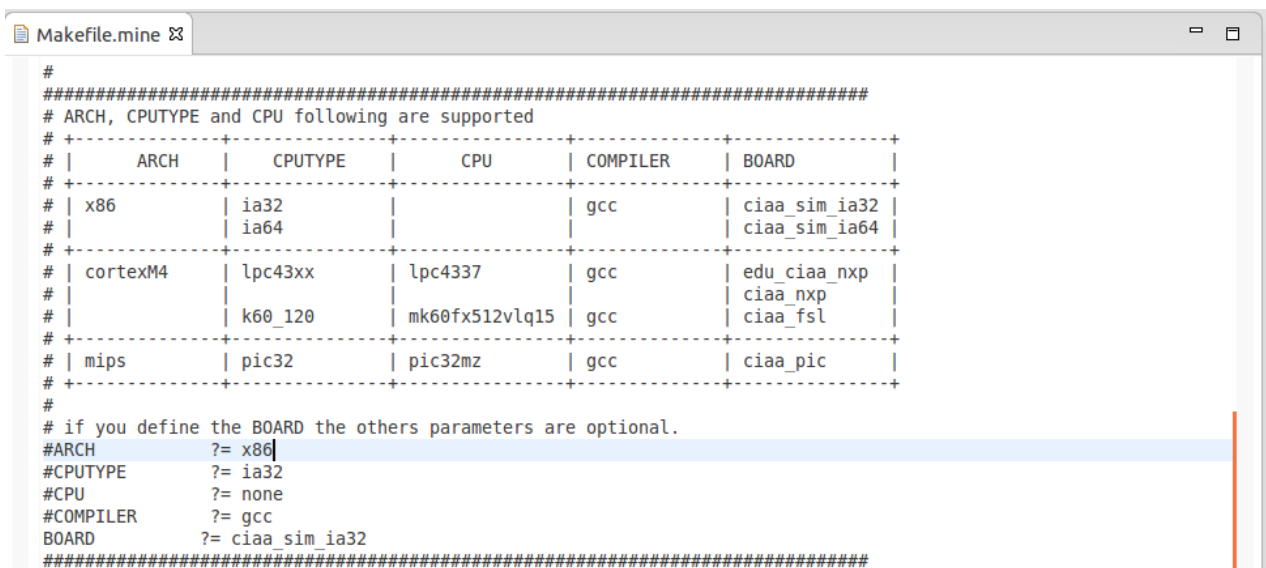


Figura 52: Contenido del archivo MakeFile.mine

En él, podemos ver un cuadro donde, dependiendo de la plataforma en la que corramos el Firmware, debemos cambiar el valor de una variable que está debajo de él (la llamada BOARD. las otras no tiene efecto pues aparecen en forma de comentario, debido al símbolo # colocado al principio de cada línea). Como nosotros contamos con una EDU-CIAA, con micro NXP, entonces corresponde el siguiente valor de la variable:

BOARD ?= edu_ciaa_nxp

Y el archivo modificado nos quedará como se muestra en la Figura 53. Después de ello, tenemos que guardar los cambios.

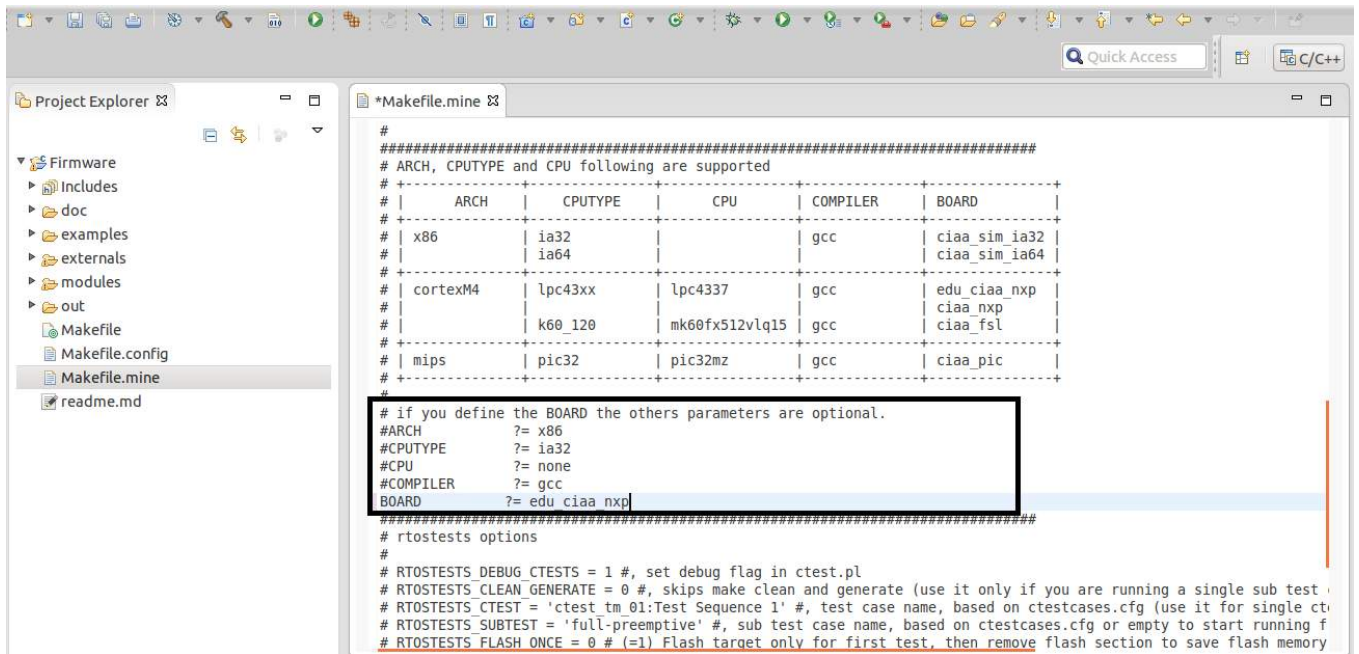


Figura 53: MakeFile.mine luego de la configuración

4.5.2 Compilación del proyecto

Una vez configurado el MakeFile, se debe limpiar (**Clean Project**) y luego compilar el proyecto (**Build Project**). Como se puede ver en la Figura 54, la extensión del archivo generado ya no es **.exe** sino **.axf**. Esta extensión es propia de la arquitectura ARM. Si esto no ocurre, vuelva al paso anterior, y verifique que no existen errores de tipeo en el nombre o en el valor de las variables que se modificaron en los MakeFile.

Para saber si se produjo una correcta compilación, vea la consola del IDE, ubicada en la parte inferior de la pantalla. En caso de que se hayan seguido todos los pasos y no se pueda compilar, hacer un **Clean** primero y luego un **Build**.

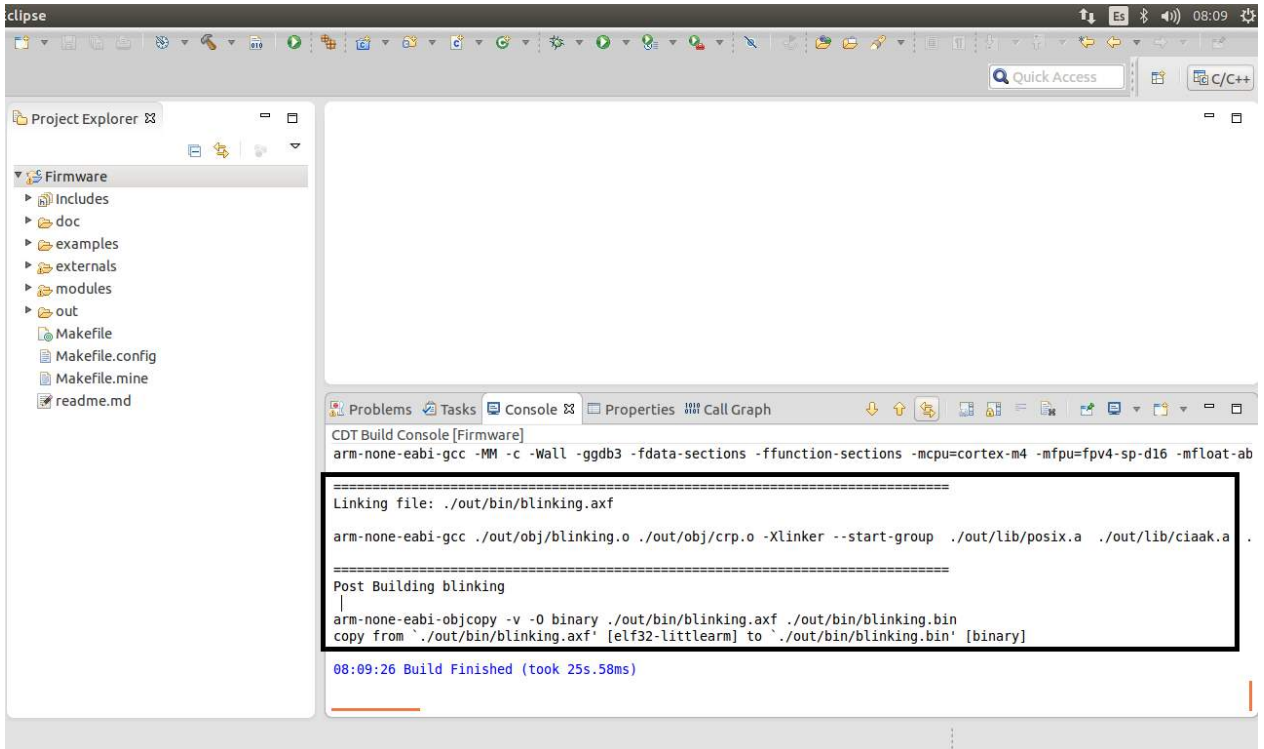


Figura 54: Entorno del Software-IDE luego de la primera compilación

4.5.3 Depuración sobre la placa: configuración de OpenOCD para Debug

Antes de poder depurar sobre la placa, necesitamos configurar el OpenOCD, que descargamos durante la instalación de Plug-ins. Para hacerlo, vamos al menú '*Run → Debug Configurations...*' se debe crear un módulo nuevo de '*Debug configuration*' del tipo '*GDB OpenOCD Debugging*'. Para ello, hacemos doble click sobre la rama que dice '*GDB OpenOCD Debugging*', a la izquierda de la ventana. A continuación, coloque los valores que se muestran en las Figura 55 Y 56, y que se detallan a continuación:

- Pestaña Main:
 1. Name: *Firmware OpenOCD*
 2. Project: *Firmware*
 3. C/C++ Application: */home/USER/CIAA/Firmware/out/bin/blinkng.axf*
 4. Tildar *Disable auto build*
 5. Build configuration: *Default*
- Pestaña Debugger:
 1. Destildamos la opción *start OpenOCD locally*
 2. GDB Client Setup

- a) Executable: *arm-none-eabi-gdb*
- b) Other options y Commands: No lo tocamos
- c) Destildar *'Force thread list update on suspend'*

Si se cuenta con una Placa EDU-CIAA “virgen”, cuya flash nunca ha sido programada, o si por algún motivo se ha borrado completamente la flash del microcontrolador, es posible que al intentar iniciar una sesión de Debug, el CIAA-IDE muestre un error. Para poder iniciar la sesión de debug, recomendamos seguir las instrucciones de la sección *Primeros pasos con el Hardware de la CIAA*, accesible en la página oficial del proyecto.

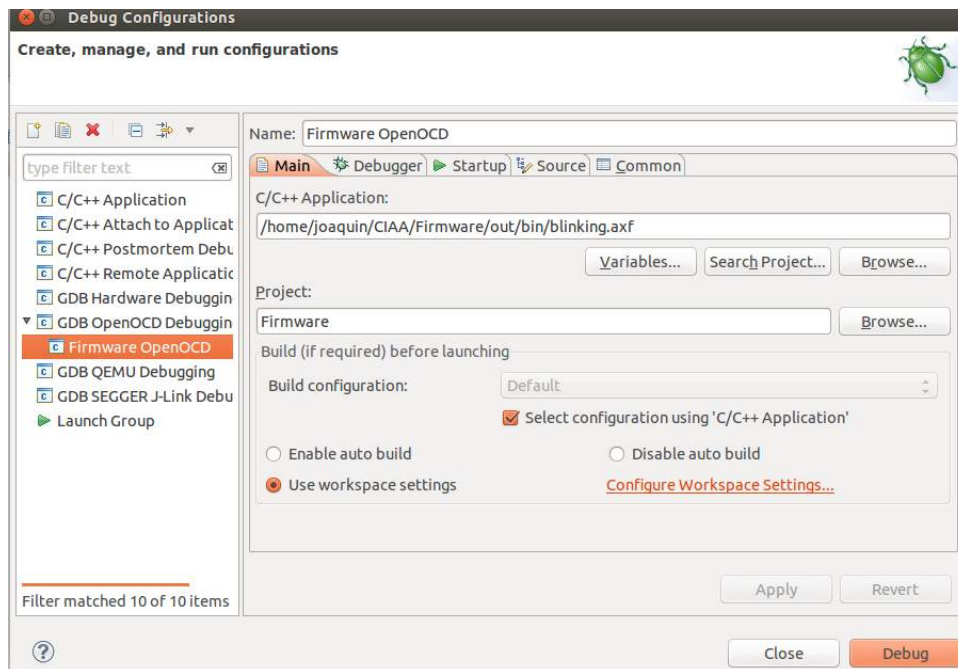


Figura 55: Configuración para poder hacer Debug sobre la placa: pestaña Main

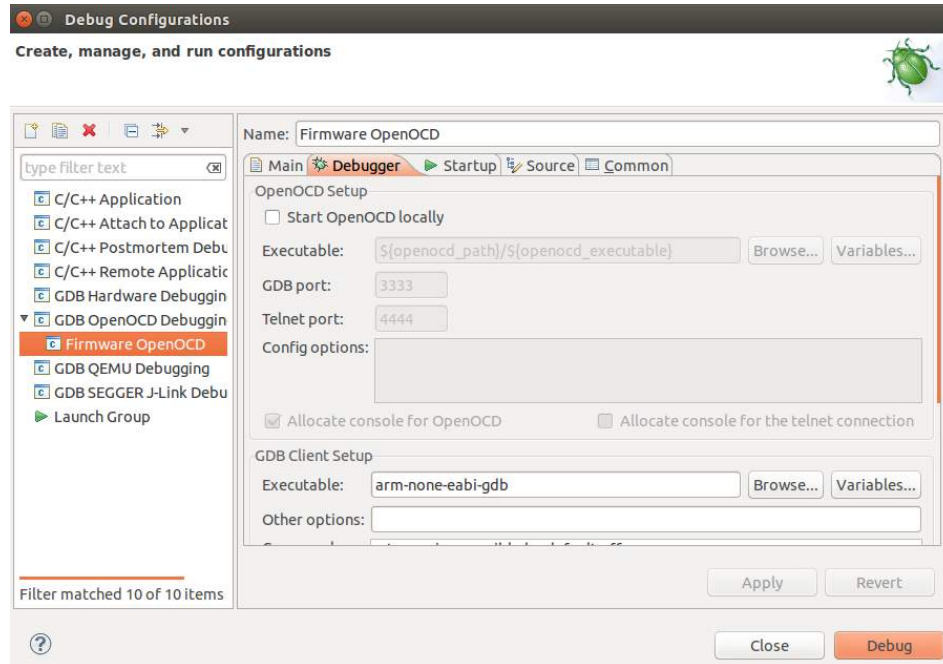


Figura 56: Configuración para poder hacer Debug sobre la placa: pestaña Debugger

Una vez finalizada esta configuración, hacemos click en el botón Apply, y dejamos abierta la ventana de *Debug Configuration*.

Para hacer Debug, es necesario que se abra un puerto a través del OpenOCD. Hay muchas formas de hacerlo. En este tutorial, lo haremos usando la consola de Ubuntu.

Abrimos la consola. Nos aparecerá una ventana como la que se muestra en la Figura 57.

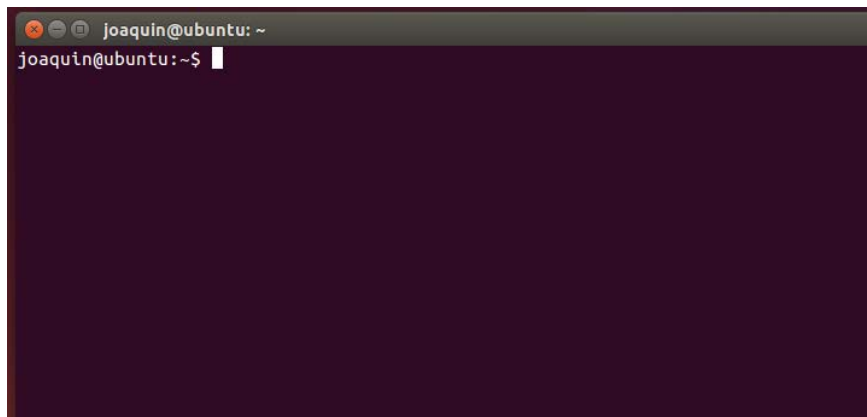


Figura 57: Consola de Ubuntu

Entonces, escribimos los siguientes comandos:

<i>Comando</i>	<i>Descripción</i>
cd \$HOME/CIAA/Firmware	Nos posiciona en la carpeta Firmware (si usamos los Path por defecto)
make openocd	Comienza a correr el servicio del OpenOCD

Si se cuenta con una Placa EDU-CIAA “virgen”, cuya flash nunca ha sido programada, o si por algún motivo se ha borrado completamente la flash del microcontrolador, es posible que al intentar iniciar una sesión de Debug, el CIAA-IDE muestre un error. Para poder iniciar la sesión de debug, recomendamos seguir las instrucciones de la sección **Primeros pasos con el Hardware de la CIAA**, accesible en la página oficial del proyecto.

Luego de iniciado el servicio del OpenOCD, dejamos la consola abierta, volvemos al entorno IDE, y damos clic al botón **Debug** de la ventana que dejamos abierta previamente. El entorno se reconfigurará para mostrar las ventanas más importantes para la depuración. En ese caso, puede ocurrir que nos aparezca una advertencia como la que se ve en la Figura 58. En ella, se comenta que va a cambiar la perspectiva del entorno a una que sea más adecuada para hacer Debug. Si no queremos que nos vuelva a preguntar, tildamos la opción ‘**remember my decision**’ y luego, clic en **Yes**.

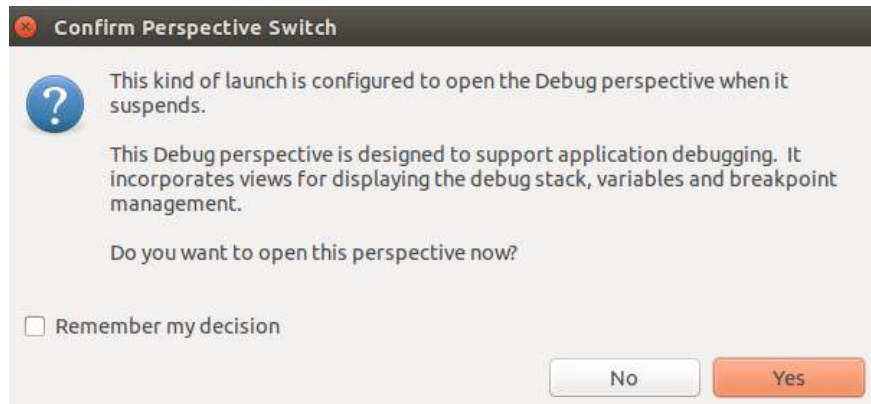


Figura 58: Advertencia de cambio de perspectiva en el proceso de Debug