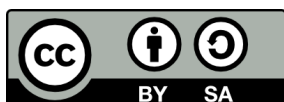


Instalación de GNU MCU Eclipse IDE for C/C++ para las plataformas CIAA-NXP y EDU-CIAA-NXP en Ubuntu 16.04 (64-bits) o Ubuntu 18.04

Licencia



“Instalación de GNU MCU Eclipse IDE for C/C++ para las plataformas CIAA-NXP y EDU-CIAA-NXP en Ubuntu 16.04 (64-bits) o Ubuntu 18.04” por Patricio Bos y Eric Pernia, se distribuye bajo una [Licencia Creative Commons Atribución-CompartirIgual 4.0 Internacional](https://creativecommons.org/licenses/by-sa/4.0/).

Esta guía se basa en guías anteriores, como por ejemplo:

[Paso a paso: CIAA-IDE en Ubuntu con una EDU-CIAA-NXP](#)

Requisitos

- Una EDU-CIAA-NXP o CIAA-NXP.
- Ubuntu 16.04 o 18.04 (64-bits).
- Al menos 1GB de espacio libre en la carpeta home del usuario (/home/\$USER, o \$HOME).

Notas:

- Para todos los comandos a ejecutar se debe abrir una terminal (puede usar Ctrl+Alt+t).
- Los comandos a ejecutar en la terminal se marcan con fondo gris, si están separados por un renglón en blanco, ejecutar de a uno (para poder chequear errores).

Clonar el repositorio de firmware

```
cd $HOME/CIAA
```

```
git clone http://github.com/epernia/cese-edu-ciaa-template/
```

Descargar de GNU MCU Eclipse para linux

1) Descargar e instalar GNU MCU Eclipse:

```
cd ~/CIAA

wget
https://github.com/gnu-mcu-eclipse/org.eclipse.epp.packages/releases/download/v4.5.1-20190101-2018-12/20190101-2023-gnumcueclipse-4.5.1-2018-12-R-linux.gtk.x86_64.tar.gz
```

Nota: el segundo comando incluye desde wget hasta tar.gz
Eclipse requiere tener una VM de java ver:
<https://docs.datastax.com/en/install/6.0/install/installJdkDeb.html>

2) Descomprimir los archivos

```
tar xzvf 20190101-2023-gnumcueclipse-4.5.1-2018-12-R-linux.gtk.x86_64.tar.gz
```

Nota: el comando incluye desde tar hasta tar.gz

3) Crear un script para ejecutar el IDE

```
nano start-ide
```

Se abre un editor de texto donde se debe copiar el siguiente texto:

```
#!/bin/sh

export
PATH=$PATH:$HOME/CIAA/gcc-arm-none-eabi-7-2018-q2-update/bin:$HOME/CIAA/openocd-0.10.0/src
./eclipse/eclipse -data $HOME/CIAA/workspace/ &
```

Guardar los cambios y salir del editor de texto con:

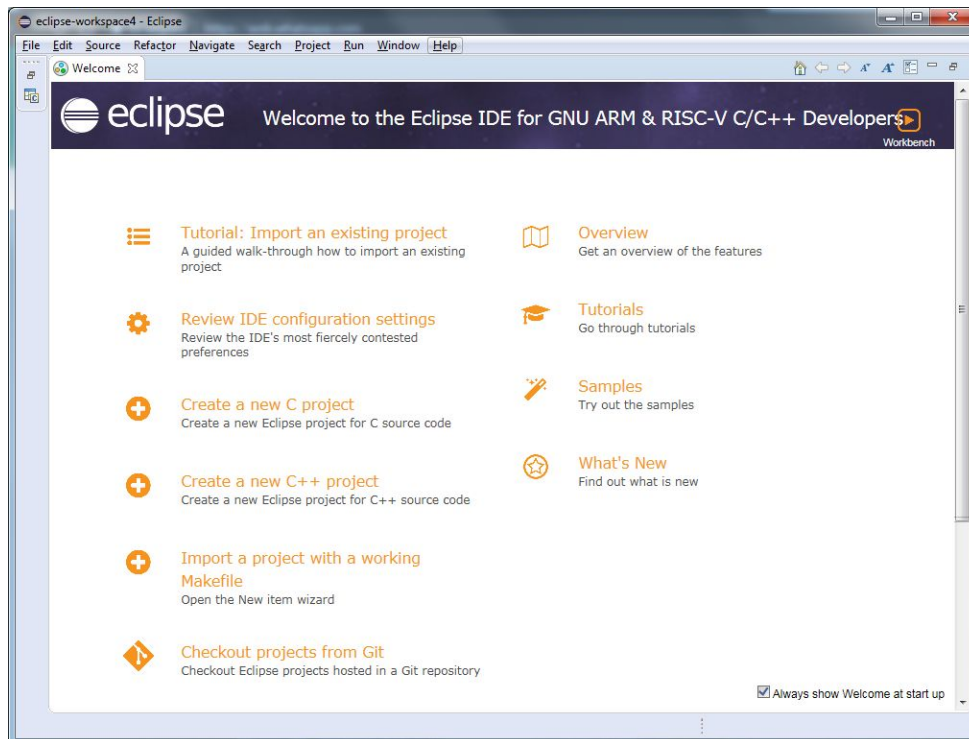
ctrl+o, enter, ctrl+x

4) Dar permisos de ejecución al script

```
chmod +x start-ide
```

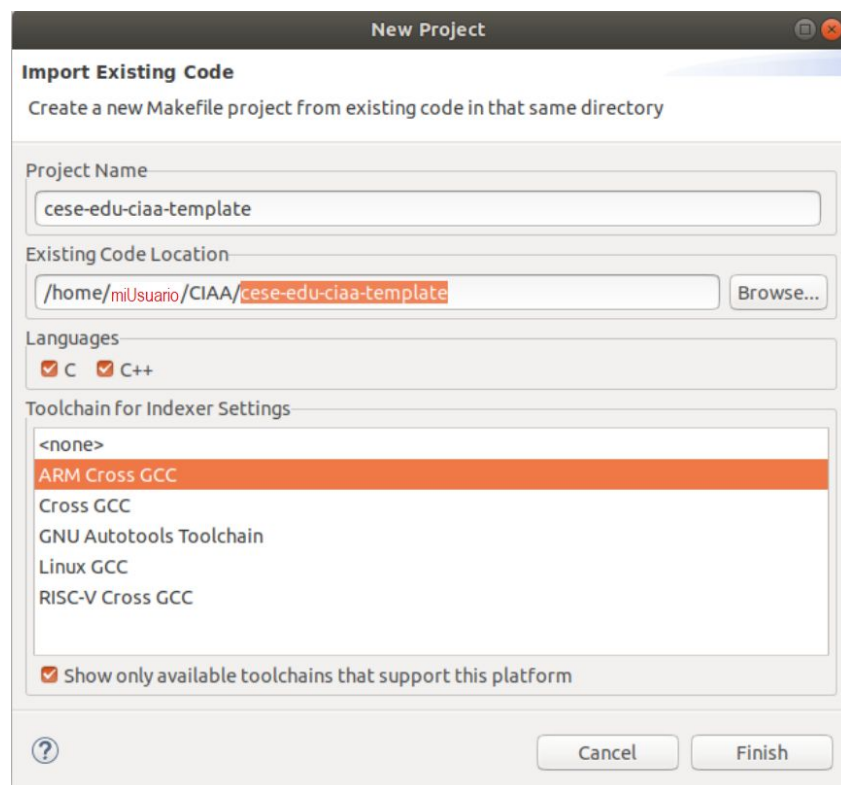
5) Ejecutar el script para iniciar el IDE

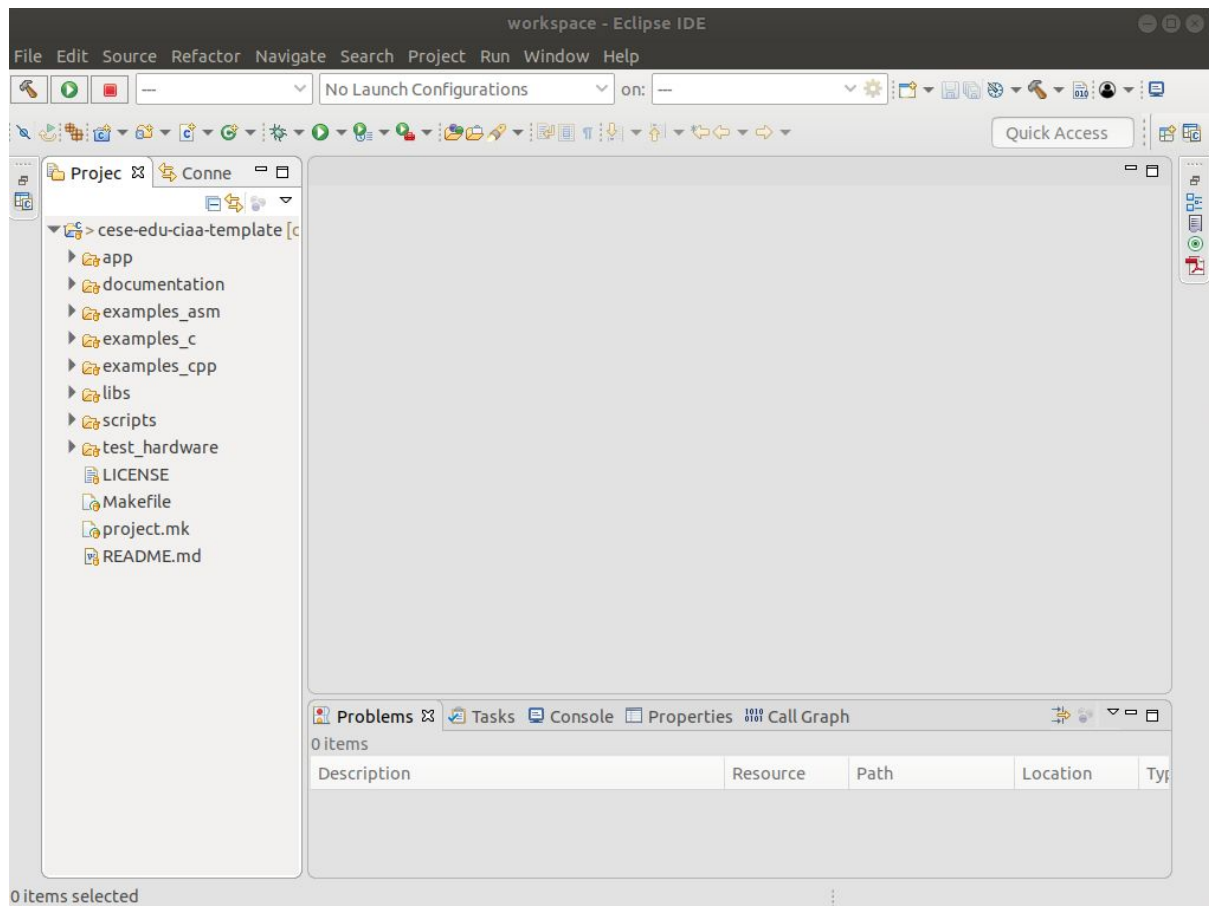
```
cd $HOME/CIAA
./start-ide
```



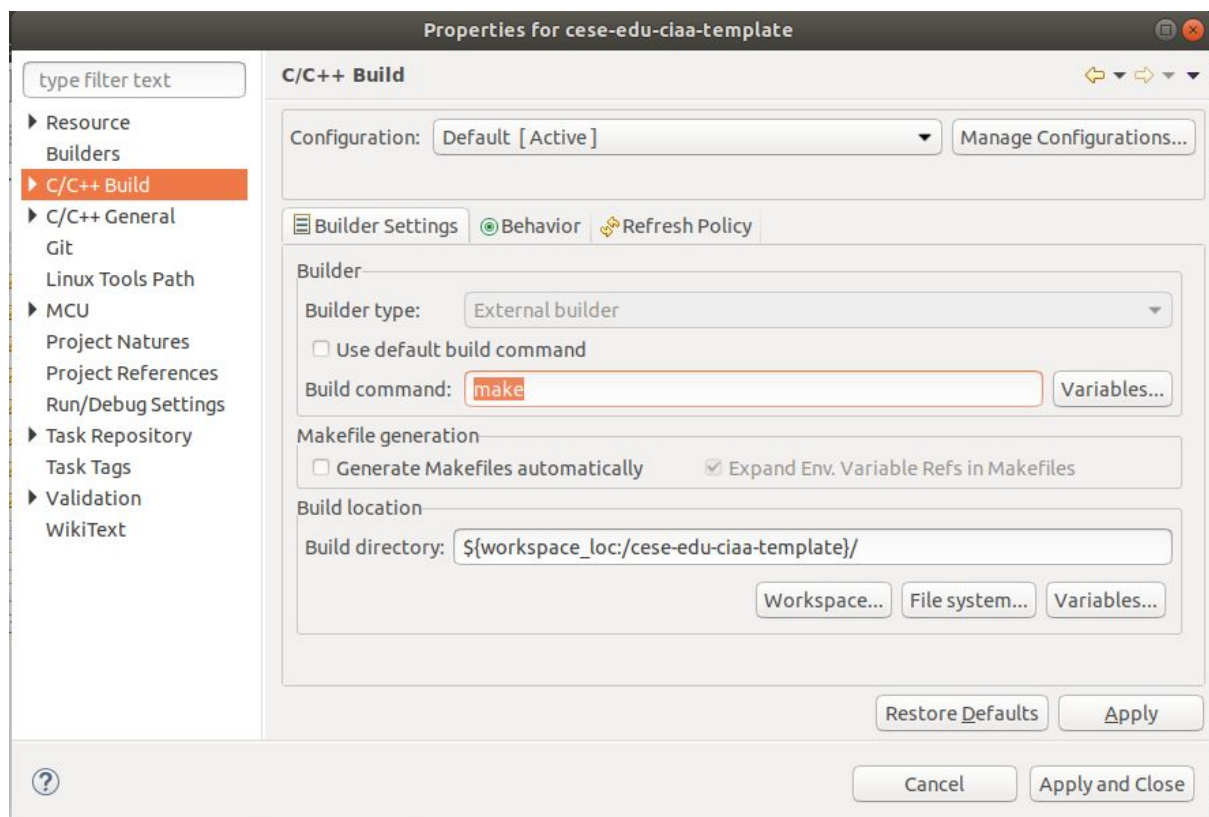
Importar el firmware a Eclipse

Menú File -> New -> Makefile Project with Existing Code





Configurar el comando de compilación: Menú Project -> Properties



Compilar un ejemplo

1) El archivo “program.mk” que se encuentra a nivel raíz en el explorador de archivos de eclipse permite elegir el programa a compilar (si no existe, crearlo).

Este archivo de texto es un archivo makefile que es incluido dentro del makefile principal (archivo Makefile ubicado también en la raíz) y se utiliza para especificar las variables `PROGRAM_PATH` y `PROGRAM_NAME`. `PROGRAM_PATH` indica la ruta (path) de la carpeta que contiene el programa, mientras que `PROGRAM_NAME` especifica el nombre del programa dentro de dicha carpeta.

Note que el símbolo `#` indica un comentario dentro de los archivos de makefile con lo cual todo lo escrito precedido por `#` no será tenido en cuenta.

Si no se altera el contenido de este archivo por defecto apunta a compilar el programa “app” que se encuentra en la carpeta “examples/c”:

```
PROGRAM_PATH = examples/c
PROGRAM_NAME = app
```

2) Presione Ctrl + B para compilar (o el icono del martillo).

3) Puede cambiar estas variables para apuntar a otro programa a compilar, por ejemplo:

```
PROGRAM_PATH = examples/c/sapi/gpio
PROGRAM_NAME = blinky
```

4) Presione nuevamente Ctrl + B para compilar este otro programa (o el icono del martillo).

Note que al compilar cualquier programa se crea una carpeta out dentro de la carpeta que incluye al programa con el resultado de la compilación.

Crear un nuevo programa

En esta estructura de firmware un programa es simplemente una carpeta que en su interior requiere de al menos:

- Una carpeta **src** que contenga un archivo con extensión “.c”, por ejemplo, **archivo.c** el cual contenga la función **main()** (función principal y punto de partida de la aplicación) declarada en el mismo.

- Un archivo que debe nombrarse **config.mk** con configuraciones de compilación y bibliotecas a utilizar.

De esta manera un programa típico quedará con esta estructura:



Ejemplo de **archivo.c**

```
#include "sapi.h"
int main( void )
{
    boardInit();
    while(1){
        gpioToggle(LED);
        delay(200);
    }
}
```

Ejemplo de **config.mk**

```
# Compile options

VERBOSE=n
OPT=g
USE_NANO=y
SEMIHOST=n
USE_FPU=y

# Libraries

USE_LPCOPEN=y
USE_SAPI=y
```

En caso de requerir archivos headers (por ejemplo, **archivo.h**) se puede agregar al programa una carpeta **inc** e incluirlos en ella.

Ejemplo de **archivo.h**

```
#ifndef __ARCHIVO_H_
#define __ARCHIVO_H_
    // Your public declarations...
#endif
```

Una estructura más completa de un programa sería entonces:



Si la carpeta **mis_programas** se sitúa en la raíz de la carpeta **cese-edu-ciaa-template**, entonces las rutas para compilarlo que se deben indicar en el archivo **program.mk** serán:

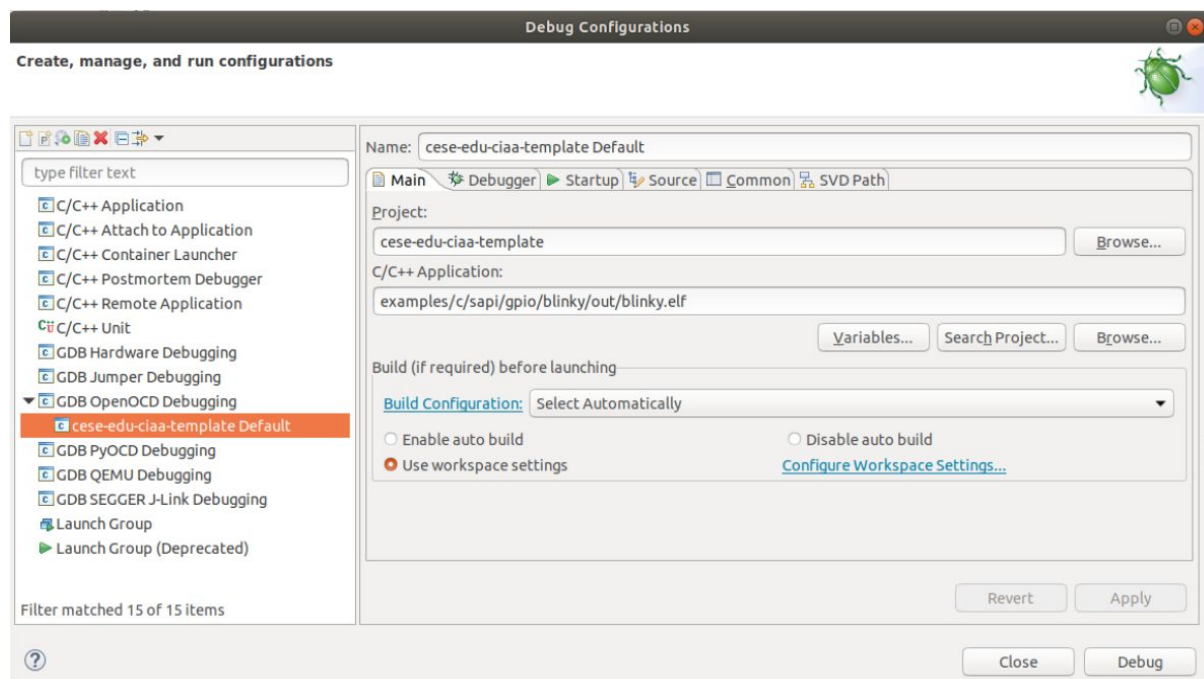
```
PROGRAM_PATH = mis_programas
PROGRAM_NAME = programa1
```

Esta forma de organización es una buena práctica de programación y permite contar con una carpeta **mis_programas** con el **nombre de cada materia** y dentro de la misma los programas que se realizarán en las prácticas.

Configurar el debugger dentro de eclipse

Menú Run -> Debug Configuration.

Doble clic en GDB OpenOCD Debugging



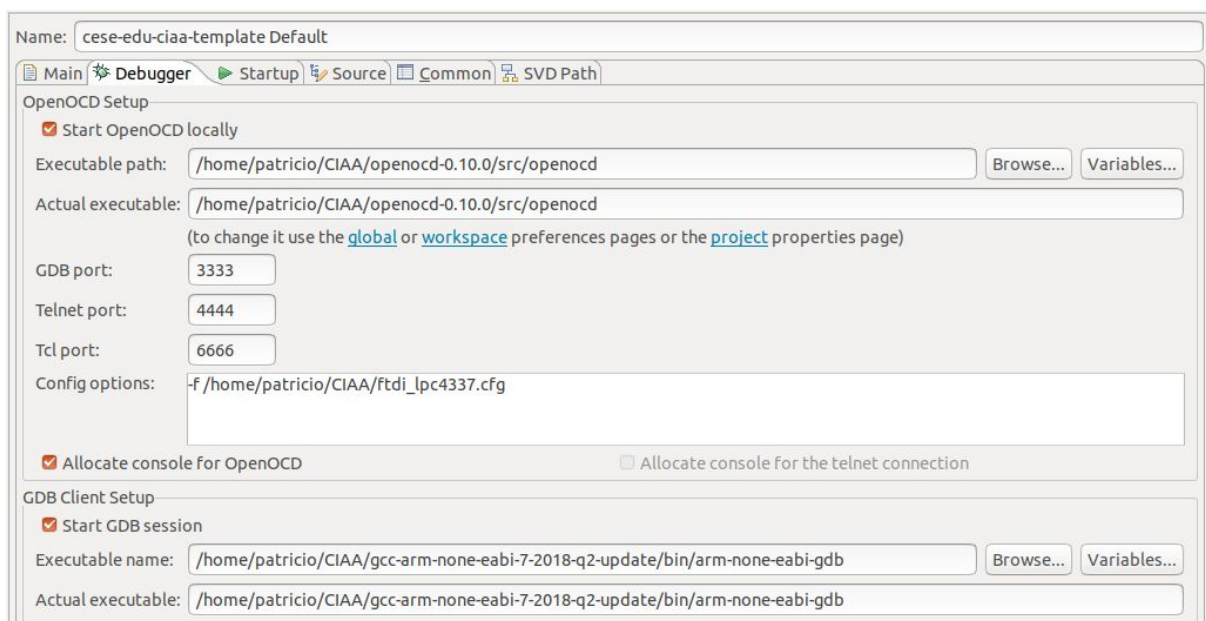
Verificar en la pestaña “Main” que:

- En “Project” figure “cese-edu-ciaa-template”.
- En “C/C++ Application” figure la ruta relativa hasta el binario. Para este ejemplo sería “examples/c/sapi/gpio/blinky/out/blinky.elf” (notar que se puede buscar el archivo mediante el “Browse”)

En la pestaña “Debugger”:

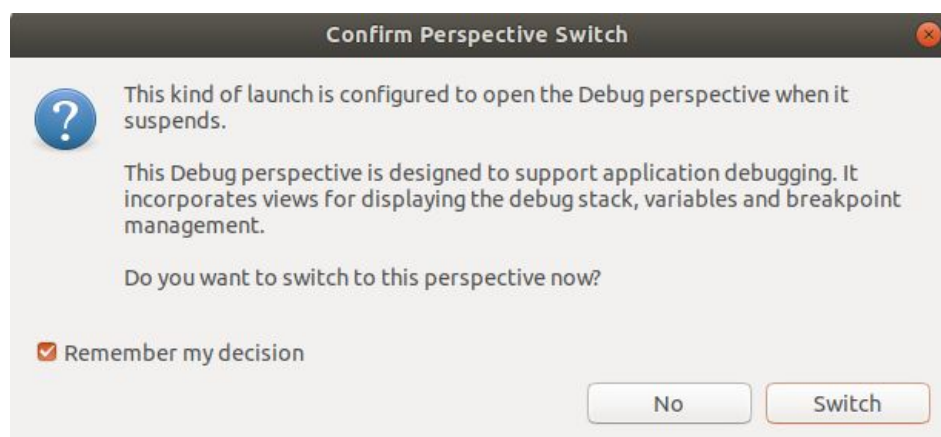
- En OpenOCD Setup:
 - Executable path -> Browse...
Buscar el ejecutable “openocd” dentro de la carpeta “/CIAA/openocd-0.10.0/src”.
Como ejemplo, debería quedar:
/home/**miUsuario**/CIAA/openocd-0.10.0/src/openocd
 - Config options: “-f /home/**miUsuario**/CIAA/ftdi_lpc4337.cfg”
- GDB Client Setup
 - Executable name -> Browse...
Buscar el ejecutable “arm-none-eabi-gdb” dentro de la carpeta “/gcc-arm-none-eabi-7-2018-q2-update/bin/”.
Como ejemplo, debería quedar:
“/home/**miUsuario**/CIAA/gcc-arm-none-eabi-7-2018-q2-update/bin/arm-none-eabi-gdb”

Nota: en rojo deben reemplazar su nombre de usuario.



Luego, con la CIAA-NXP o la EDU-CIAA-NXP conectada a la PC a través del puerto DBG o DEBUG, respectivamente, presionar “Apply” y “Debug” para probar la configuración del debugger.

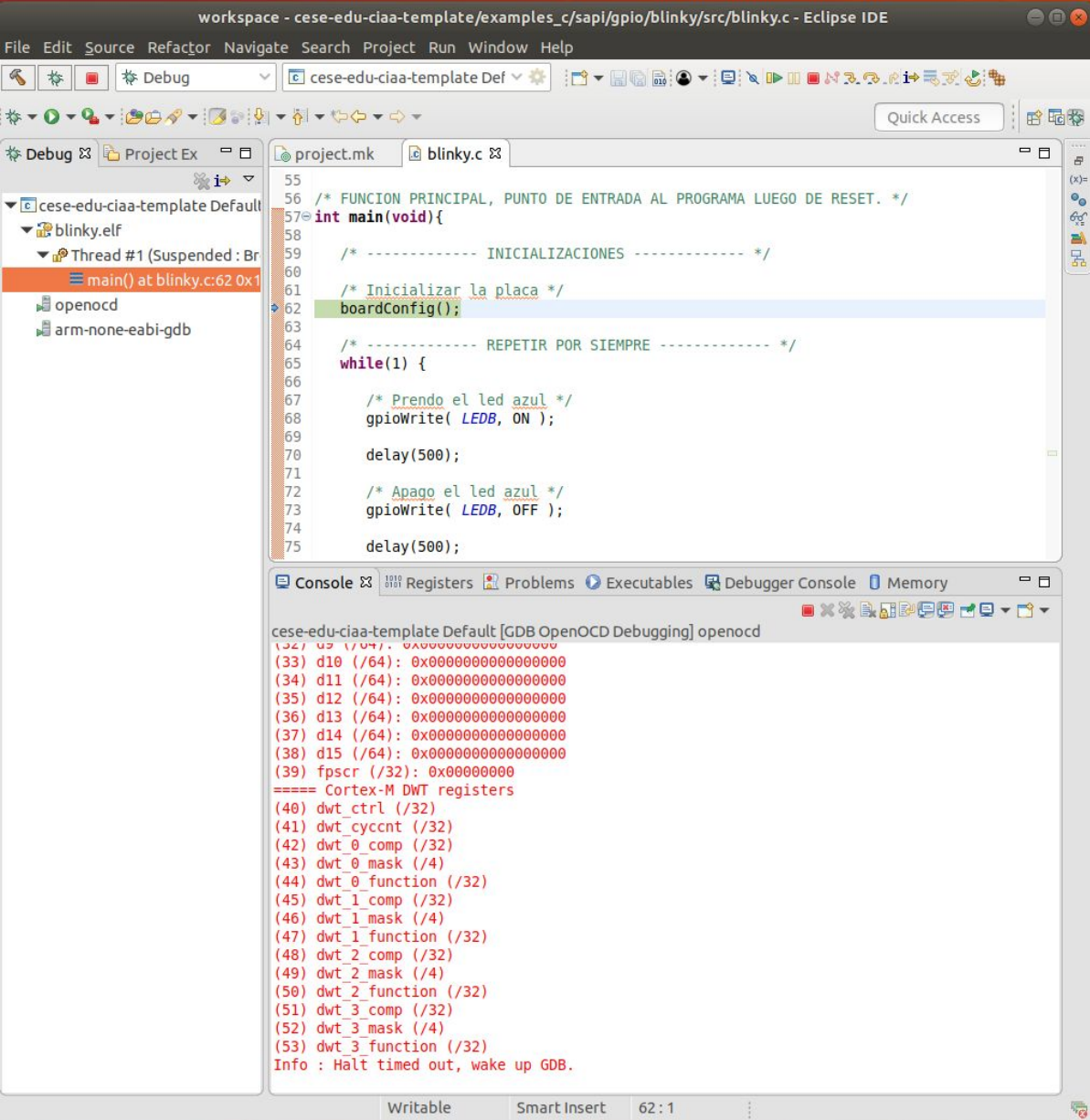
La primera vez que iniciamos una sesión de debug debería aparecer una ventana pidiendo confirmación para realizar un cambio de “perspectiva” que es simplemente un nuevo arreglo del IDE para mostrar botones y controles más afines a la depuración. Tildar “remember my decision” y presionar “switch”.



Si todo fue configurado correctamente, debería iniciarse una sesión de debug del programa de ejemplo “blink.c”

A continuación se muestra una figura de qué debería verse en la ventana de eclipse en la perspectiva “debug”. Notar que arriba a la derecha, al costado de cuadro quick access se encuentran dos botones para cambiar entre las perspectivas de programación y debug.

Para finalizar la sesión de debug, en la barra de herramientas se encuentran dos botones de “stop” con un ícono cuadrado rojo que funcionan indistintamente a estos fines. Alternativamente se puede ir al menú Run -> Terminate para obtener el mismo resultado.



workspace - cese-edu-ciaa-template/examples_c/sapi/gpio/blink/src/blink.c - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Debug cese-edu-ciaa-template Def

Quick Access

Debug Project Ex

cese-edu-ciaa-template Default

blinky.elf

Thread #1 (Suspended: Breakpoint)

main() at blinky.c:62 0x1

openocd

arm-none-eabi-gdb

```
55
56 /* FUNCION PRINCIPAL, PUNTO DE ENTRADA AL PROGRAMA LUEGO DE RESET. */
57 int main(void){
58
59     /* ----- INICIALIZACIONES ----- */
60
61     /* Inicializar la placa */
62     boardConfig();
63
64     /* ----- REPETIR POR SIEMPRE ----- */
65     while(1) {
66
67         /* Prendo el led azul */
68         gpioWrite( LEDB, ON );
69
70         delay(500);
71
72         /* Apago el led azul */
73         gpioWrite( LEDB, OFF );
74
75         delay(500);
76     }
77 }
```

Console Registers Problems Executables Debugger Console Memory

cese-edu-ciaa-template Default [GDB OpenOCD Debugging] openocd

```
(32) d9 (/64): 0x0000000000000000
(33) d10 (/64): 0x0000000000000000
(34) d11 (/64): 0x0000000000000000
(35) d12 (/64): 0x0000000000000000
(36) d13 (/64): 0x0000000000000000
(37) d14 (/64): 0x0000000000000000
(38) d15 (/64): 0x0000000000000000
(39) fpscr (/32): 0x00000000
===== Cortex-M DWT registers
(40) dwt_ctrl (/32)
(41) dwt_cycnt (/32)
(42) dwt_0_comp (/32)
(43) dwt_0_mask (/4)
(44) dwt_0_function (/32)
(45) dwt_1_comp (/32)
(46) dwt_1_mask (/4)
(47) dwt_1_function (/32)
(48) dwt_2_comp (/32)
(49) dwt_2_mask (/4)
(50) dwt_2_function (/32)
(51) dwt_3_comp (/32)
(52) dwt_3_mask (/4)
(53) dwt_3_function (/32)
Info : Halt timed out, wake up GDB.
```

Writable Smart Insert 62:1