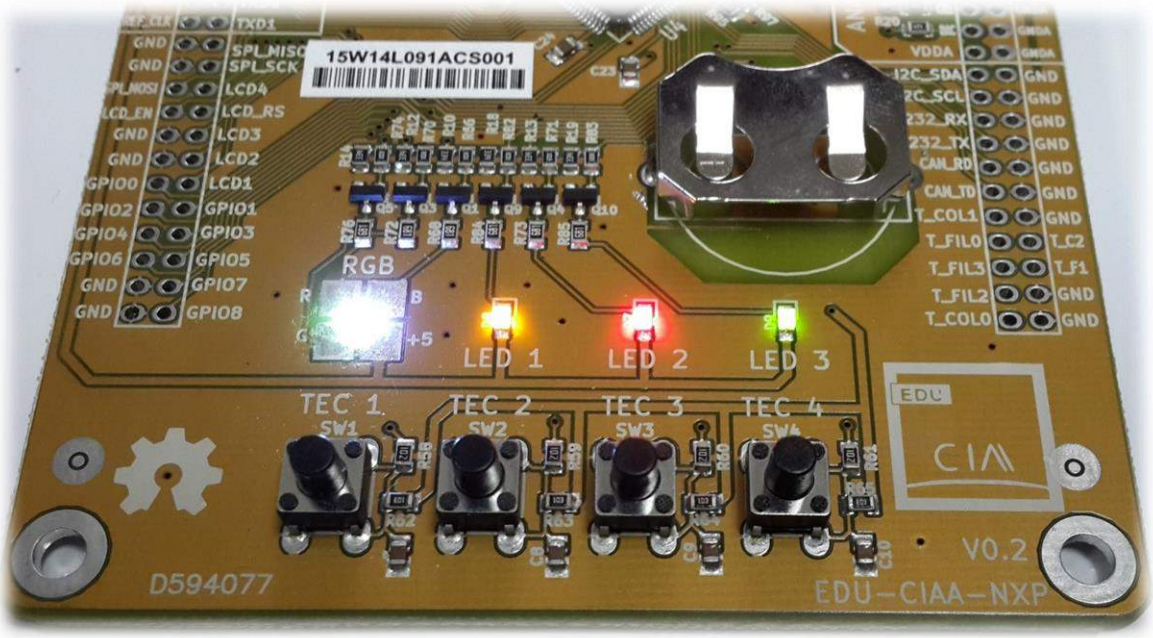


Módulo 3

Creando un nuevo proyecto en la EDU-CIAA.

Ejemplos con los Leds y Switches

Autores: Joaquín Rodríguez, Juan Pablo Vecchio

Tutor: Ing. Marcelo Pistarelli

Supervisor: Ing. José Ignacio Sosa

Asesor: Ing. Gustavo Muro

Índice

1 Introducción

En el presente documento se explicaran los pasos para crear un nuevo proyecto utilizando el Software CIAA IDE en Windows. Se recomienda haber leído "Módulo 1: Proyecto CIAA: Utilizando la EDU-CIAA", en el cual se desarrolla todo lo referente a la instalación del entorno y configuración del mismo. Aquí se hará foco en la creación de un nuevo proyecto, pasando rápidamente por la parte de configuraciones que ya fueron explicadas en el Modulo 1.

Posteriormente, se presentaran algunos ejemplos en los cuales aprenderemos a manejar los seis Leds de la placa junto con los switches, utilizando tareas del sistema operativo OSEK.

2 Creando un nuevo proyecto

Lo primero que debemos hacer es crear una carpeta "projects" en el directorio del Firmware, tal como se muestra en la Figura 1.

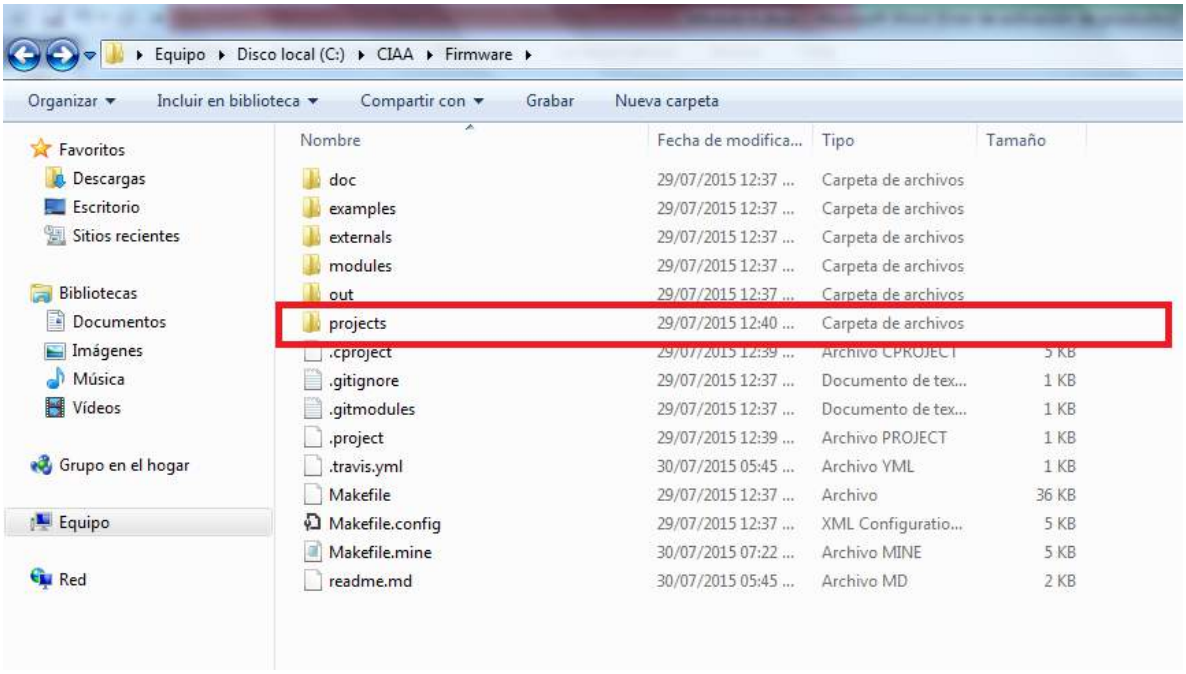


Figura 1: Creamos la carpeta projects.

Dentro de esta carpeta es en donde ubicaremos todos nuestros proyectos. Cada uno dentro de otra carpeta con su correspondiente nombre.

Por ejemplo, crearemos un nuevo proyecto al cual llamaremos "myproject". Para ello debemos crear una carpeta con dicho nombre dentro de ***C:\CIAA\Firmware\projects*** como se puede ver en la Figura 2. Todos los archivos correspondientes a nuestro proyecto deberán ubicarse dentro de la misma.

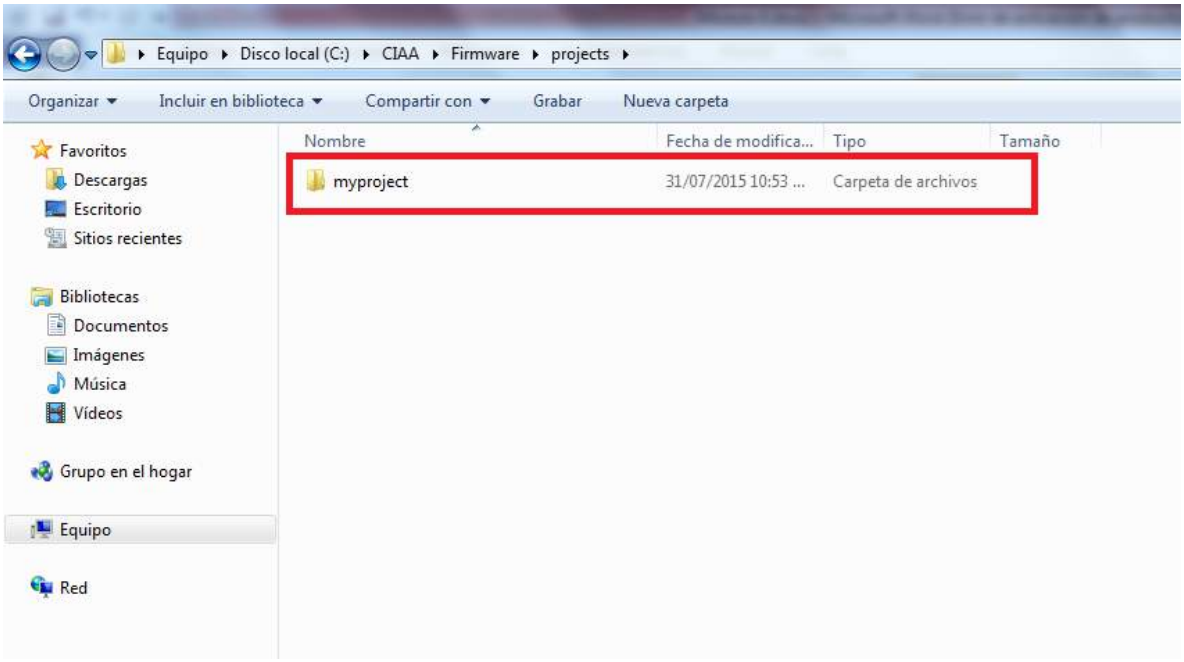


Figura 2: Creamos la carpeta de nuestro nuevo proyecto "myproject".

Para seguir la misma estructura ordenada de archivos que se pretende en la CIAA, crearemos cuatro carpetas para los distintos archivos de nuestro proyecto. Estas serán:

- **src** para los archivos fuentes **.c**
- **inc** para los archivos de cabecera **.h**
- **etc** para otros archivos como el **.oil** donde se define la configuración del sistema operativo por ejemplo.
- **mak** para el Makefile

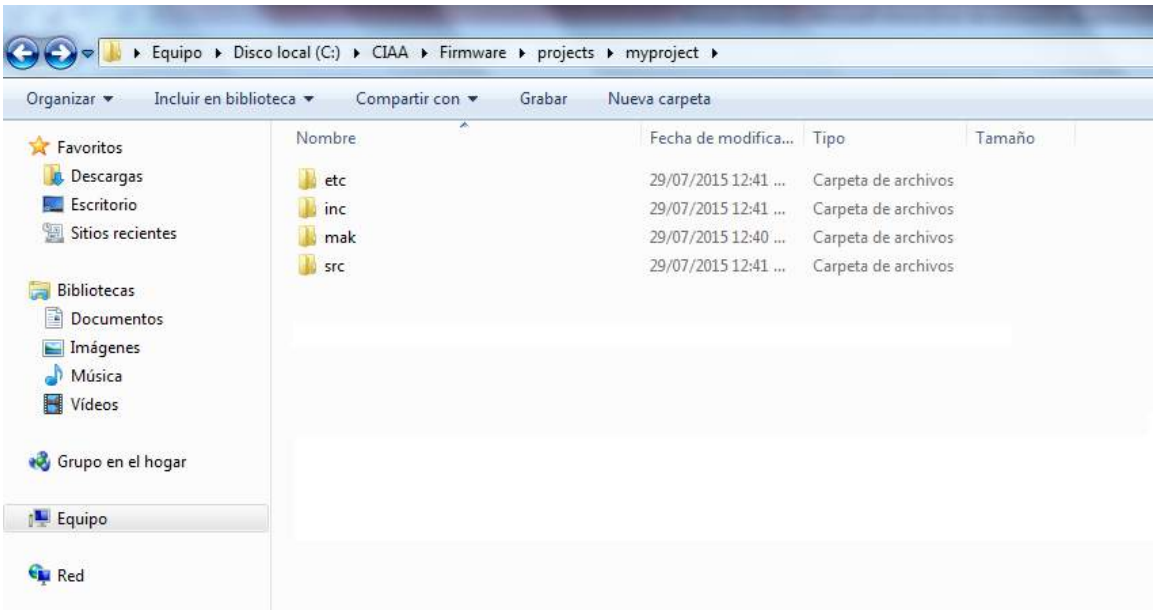


Figura 3: Estructura de carpetas para el nuevo proyecto.

Crearemos nuestro nuevo proyecto copiando el ejemplo blinking de modo de tener la misma estructura de archivos que se requieren. Para ello debemos dirigirnos a la carpeta ***C:\CIAA\Firmware\examples*** y copiamos la carpeta ***blinking*** en nuestra carpeta ***C:\CIAA\Firmware\projects*** como se expone en la Figura 4.

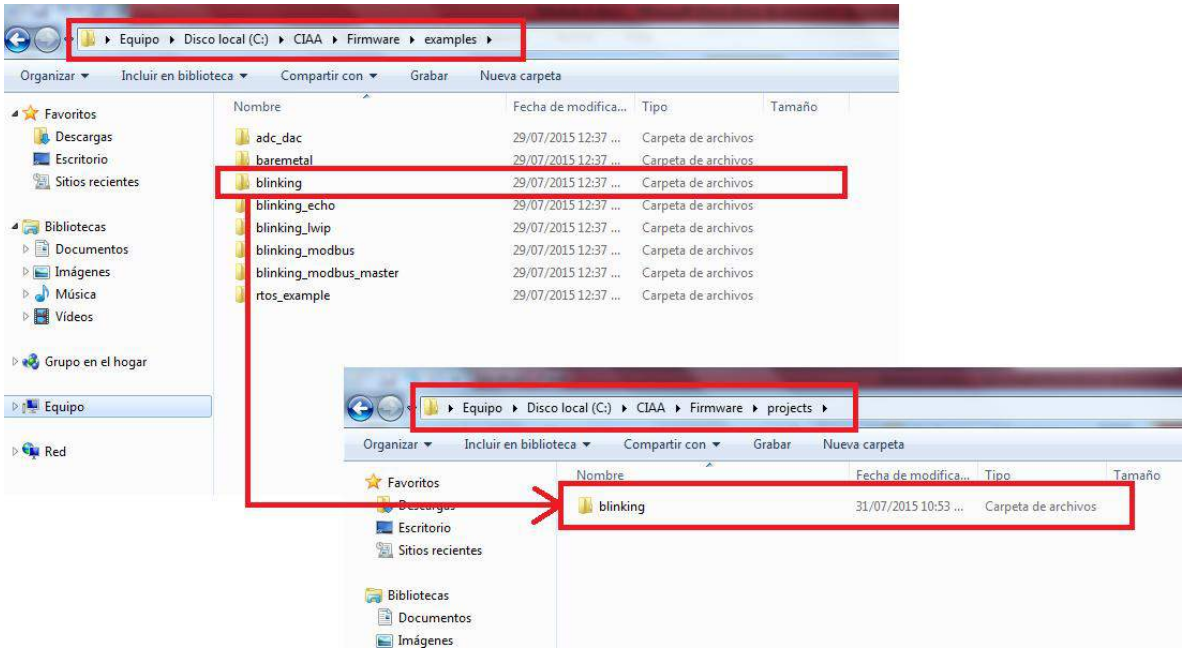


Figura 4: Copiando el ejemplo blinking en nuestro nuevo proyecto.

Ahora le cambiaremos los nombres a nuestro gusto. Como en nuestro caso vamos a llamar "myproject" a nuestro proyecto, le podremos los nombres como sigue:

- Cambiamos el nombre de la carpeta: ***blinking → myproject***
- Cambiamos el nombre de los archivos respectivos ubicados dentro de las carpetas del proyecto:
    - ***blinking.c → myproject.c*** (no es necesario que tenga el mismo nombre que el proyecto)
    - ***blinking.h → myproject.h*** (no es necesario que tenga el mismo nombre que el proyecto)
    - ***blinking.oil → myproject.oil***
- El archivo Makefile no se debe modificar.

Así nos quedará como se observa en las Figura 5Figura 6.

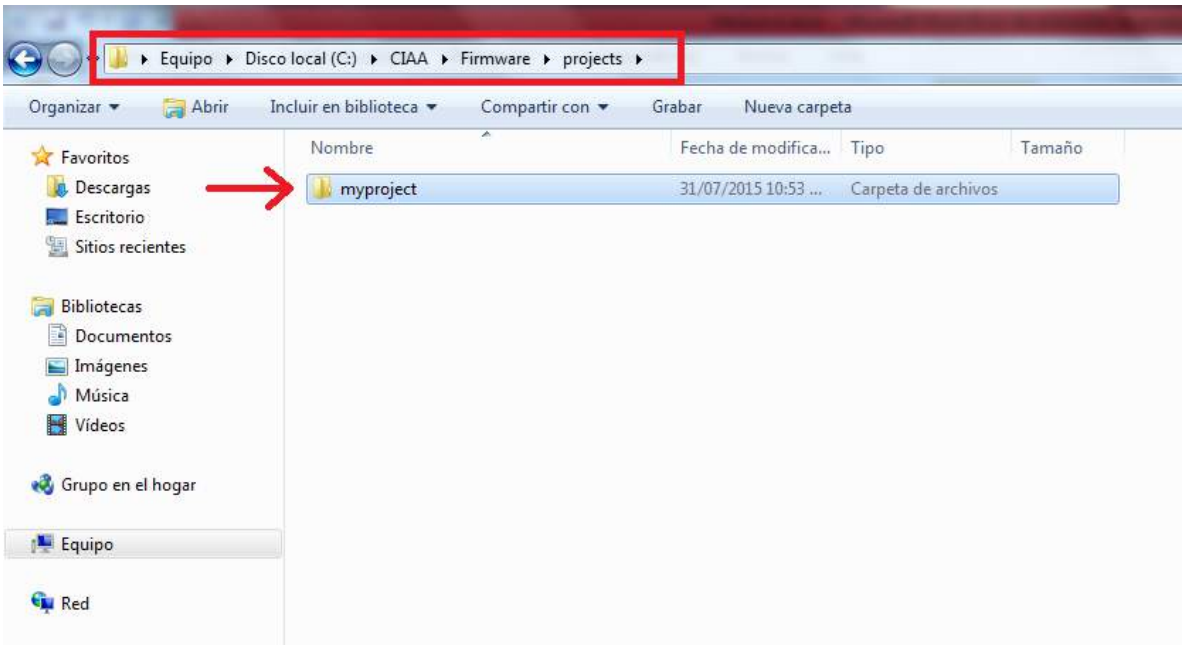


Figura 5: Cambio del nombre de la carpeta blinking copiada.

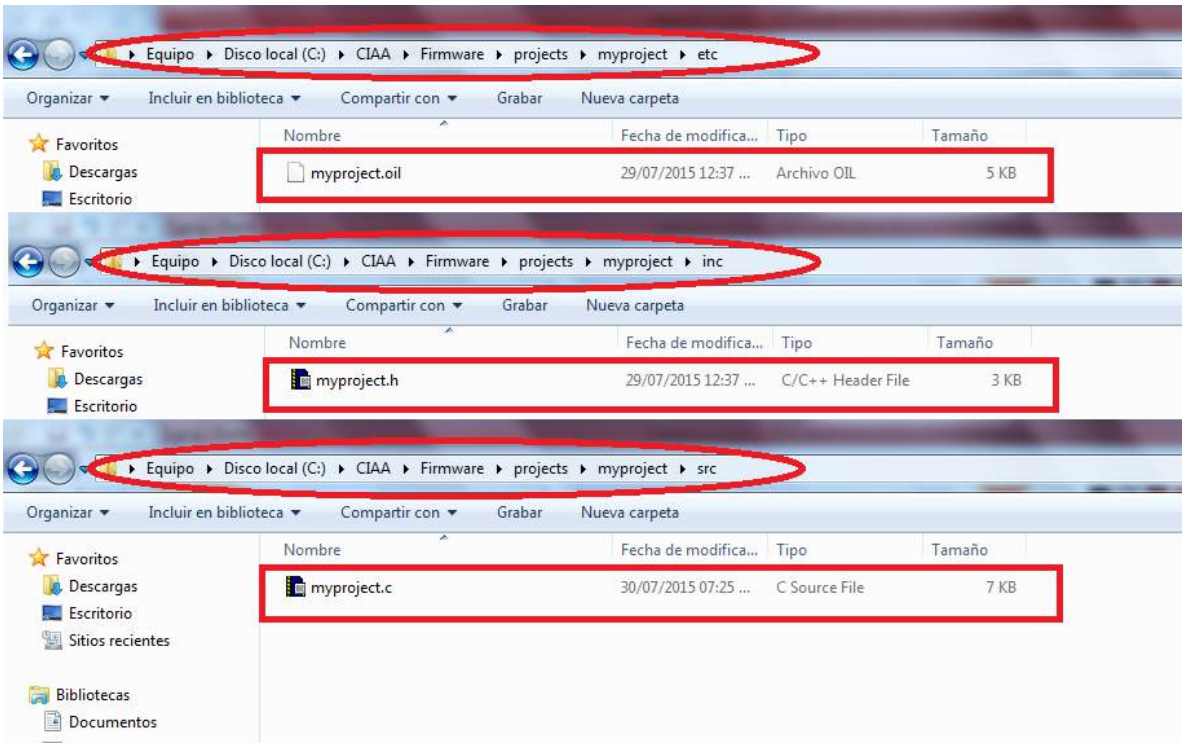


Figura 6: Cambio de los nombres de los archivos.

Ya tenemos nuestro nuevo proyecto prácticamente listo. Ahora es momento de abrirlo con el CIAA IDE y configurar el entorno de manera similar a como se explicó en el Modulo 1, teniendo en cuenta que la ruta y nombre de nuestro proyecto son otras.

2.1 Compilando nuestro proyecto "myproject"

Como primer paso, abriremos el proyecto del Firmware yendo a ***File → New → Makefile Project with Existing Code***, como se muestra en la Figura 7.

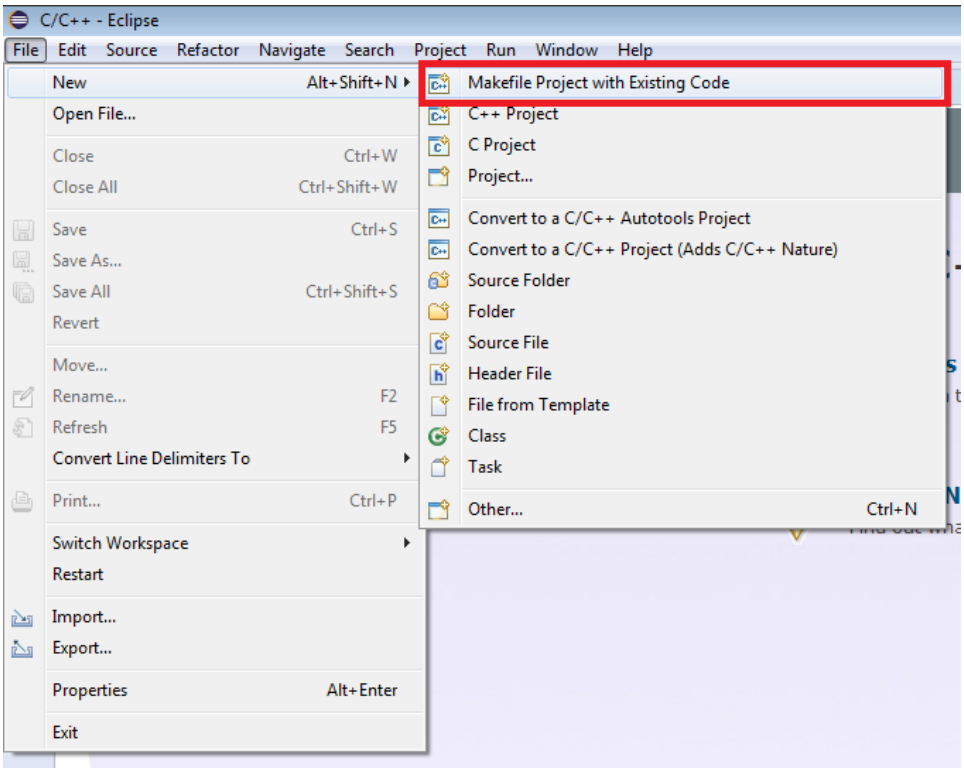


Figura 7: Comienzo de la carga del código existente.

Allí hacemos clic en 'Browse' y seleccionamos la ruta correspondiente al mismo:

C:\CIAA\Firmware

En la ventana ***Toolchain for Indexer Settings*** seleccionaremos la opción ***<none>***, lo cual dejará las opciones por defecto, configuradas en el Makefile, tal como se presenta en la Figura 8.

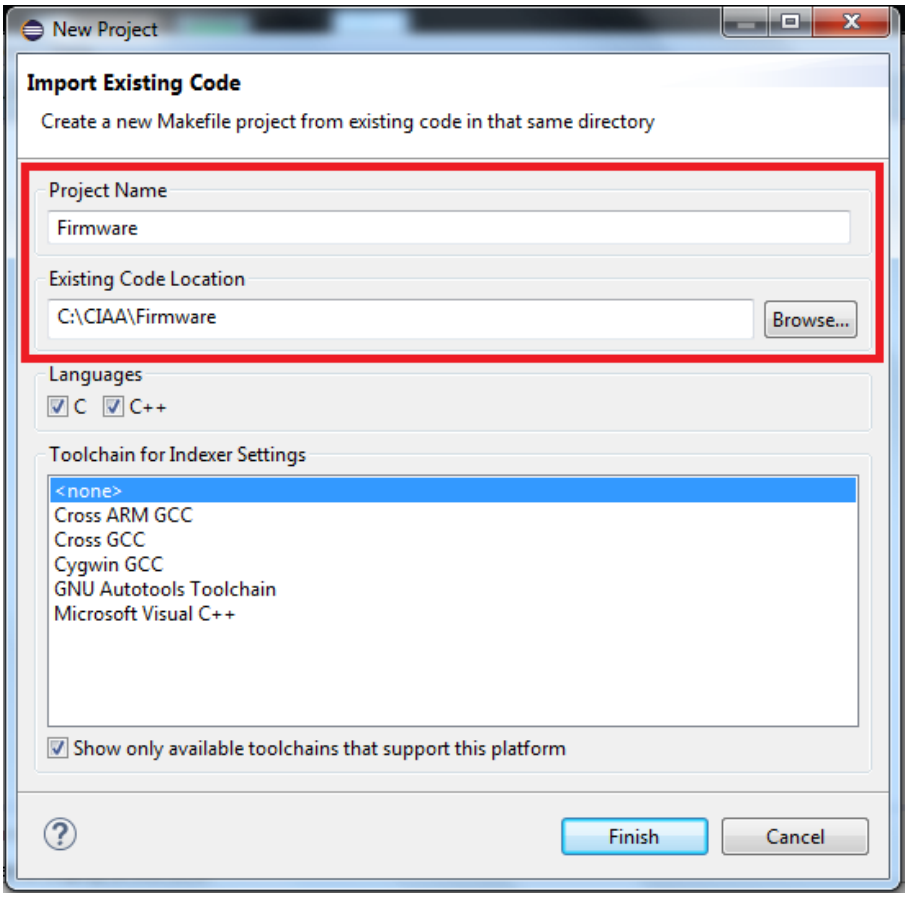


Figura 8: Creación de un nuevo proyecto usando un código existente.

Luego debemos agregar las cabeceras del estándar ***POSIX***. Para poder indexar estas definiciones debemos agregar los archivos ***Includes*** del ***GCC*** (*GNU **C**ompiler **C**ollection*).

Vamos a ***Properties*** → ***C/C++ General*** → ***Path and Symbol*** → ***Includes*** y agregamos:

En Language → 'GNU C'

- <dIDE>***\cygwin\usr\include***. Si se aplican los directorios por defecto, éste quedaría ***C:\CIAA\cygwin\usr\include***
- <dIDE>***\cygwin\lib\gcc\i686-pc-cygwin\4.9.2\include***
  (por defecto: ***C:\CIAA\cygwin\lib\gcc\i686-pc-cygwin\4.9.2\include***)

(dIDE = directorio de instalación del software-IDE: por defecto es ***C:\CIAA***)

En Language → 'GNU C++'

- <dIDE>***cygwin*****usr*****include**
  (por defecto: ***C:\CIAA\cygwin\usr\include***)
- <dIDE>***cygwin*****lib*****gcc*****i686-pc-cygwin*****4.9.2*****include*****c++**
  (por defecto: ***C:\CIAA\cygwin\lib\gcc\i686-pc-cygwin\4.9.2\include\c++***)

(dIDE = directorio de instalación del software-IDE: por defecto es ***C:\CIAA***)

Configuración del Makefile

En ***Properties → C/C++ Build***, dentro de la pestaña ***'Behaviour'***:

- Tildamos la opción ***'Stop on first build error'*** y Destildamos ***'Enable parallel build'***
- Destildamos el casillero ***'Build on resource save'*** y Tildamos ***'Build (Incremental Build)'*** y ***'Clean'.***
- En el campo ***Clean***, escribimos: ***clean_generate***
- Borramos el contenido del campo ***Build*** dejándolo en blanco.

Path Mapping

En ***'Windows → Preferences' → 'C/C++' → 'Debug' → 'Source Lookup Path'*** hacemos clic en ***'Add'***. Allí seleccionamos ***Path Mapping*** y en la ventana que se abre agregamos la nueva fuente (source) haciendo clic en ***'Add',*** con la siguiente configuración:

- Name: ***Firmware***
- Compilation Path: ***\cygdrive\c\CIAA\Firmware***
- Local file system path: ***C:\CIAA\Firmware***

Ya tenemos el entorno configurado correctamente para poder compilar nuestro proyecto. Sin embargo en nuestro caso particular que estamos haciendo una copia exacta del ejemplo blinking debemos asegurarnos de cambiar también los includes al nuevo nombre de nuestro proyecto. Para ello abrimos el archivo fuente ***myproject*** > ***src*** > ***myproject.c*** y corregimos el include del archivo cabecera con el nombre correcto como puede ver en la Figura 9.

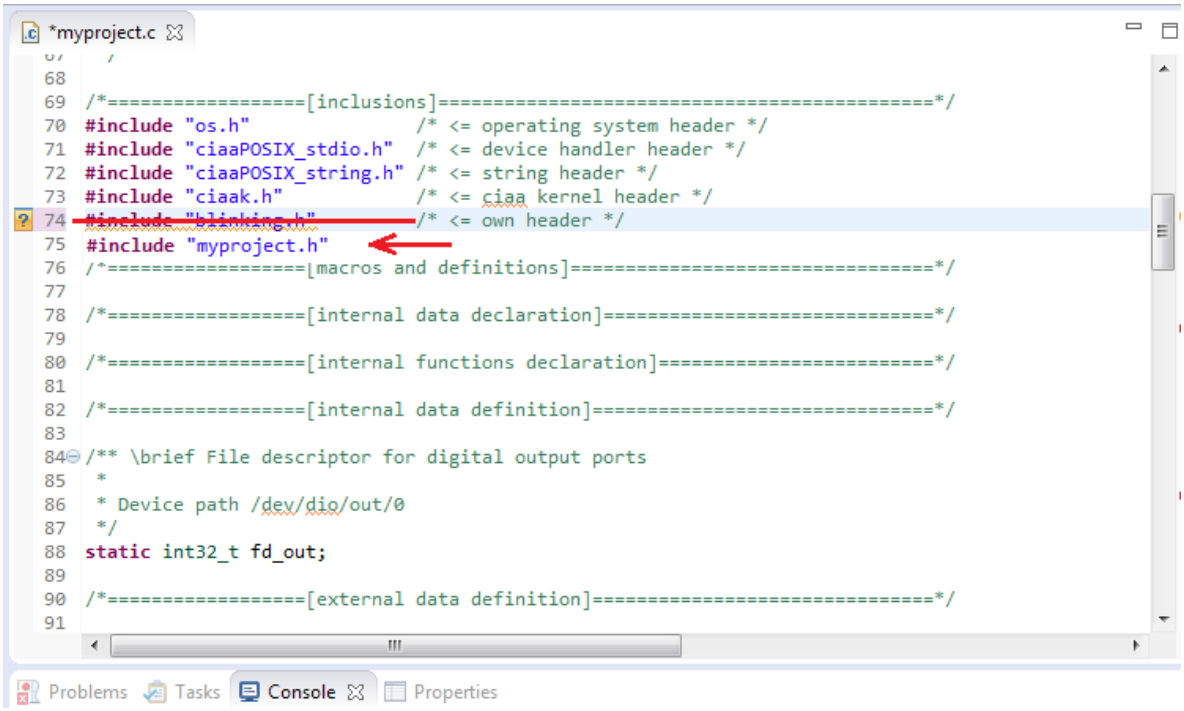


Figura 9: Corrigiendo el nombre del archivo cabecera .h

A continuación se resumen brevemente las configuraciones necesarias para depurar nuestro proyecto utilizando OpenOCD y la EDU-CIAA-NXP. Se recomienda dirigirse al Módulo 1, Sección 6 y 7 donde se profundiza sobre el tema.

Si no lo hemos hecho antes, debemos abrir el archivo ***MakeFile.mine*** que se encuentra al mismo nivel que el proyecto Firmware (Figura 10). Allí debemos indicar que utilizaremos la EDU-CIAA-NXP, asignando el siguiente valor a la variable BOARD:

BOARD ?= edu_ciaa_nxp

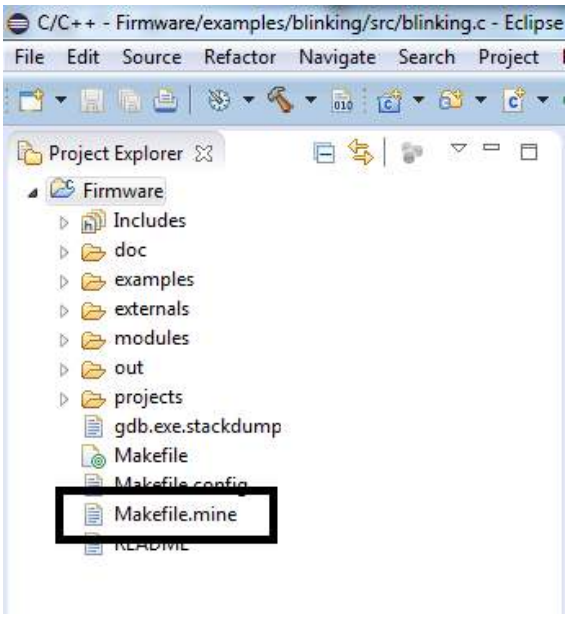


Figura 10: Project Explorer - MakeFile.mine

Luego en ***Run → Debug Configurations…*** se debe crear un módulo nuevo de ***'Debug configuration'*** del tipo ***'GDB OpenOCD Debugging'*** y realizar las siguientes configuraciones:

- Pestaña Main:
    1. Name: ***MyProject OpenOCD***
    2. Project: ***myproject***
    3. C/C++ Application: ***C:\CIAA\Firmware\out\bin\myproject.axf***
    4. Tildar ***Disable auto build***
    5. Build configuration: ***Default***

- Pestaña Debugger:
    1. OpenOCD Setup: Destildar la opción ***Start OpenOCD locally***
    2. GDB Client Setup
        a) Executable: ***C:\CIAA\cygwin\usr\arm-none-eabi\bin\arm-none-eabi-gdb.exe***
        b) Other options y Commands: dejarlo como está
        c) Destildar ***'Force thread list update on suspend'***

Hacemos clic en ***Apply*** para guardar los cambios y cerramos esta ventana.

Por último, debemos abrir un puerto a través del OpenOCD. Para esto, abrimos la consola CygWin que incluye el intalador del CIAA-IDE y allí escribimos los siguientes commandos:

Comando	***Descripción***
cd C:/CIAA/Firmware	Nos posiciona en la carpeta Firmware (si usamos los Path por defecto)
make openocd	Comienza a correr el servicio del OpenOCD

Luego de iniciado el servicio del OpenOCD, dejamos la ventana abierta, volvemos al entorno IDE, y hacemos ***Debug***.

3 Ejemplo utilizando los Leds

En la sección previa se explicó cómo crear un nuevo proyecto haciendo una copia del ejemplo blinking. Comenzaremos entendiendo un poco dicho ejemplo y como realizarle algunas modificaciones como ser: cambio del Led activo y cambio de frecuencia.

La EDU-CIAA posee cuatro Leds:

- LED RGB
- LED1 (Amarillo)
- LED2 (Rojo)
- LED3 (Verde)

Como uno de los mismos es un Led RGB, sería equivalente a tener un total de seis Leds posibles para encender por medio de las salidas digitales DIO.

Dado que en el proyecto de utiliza el estándar POSIX, para acceder a los dispositivos se deberá primero abrir el dispositivo ***DIO*** de salida, obteniendo el fildes (file descriptor

associated with the terminal device) o manejador correspondiente y luego con la función WRITE encender el Led deseado.

Los Leds forman parte de las ***DIO (Digital input-output)*** y su ruta correspondiente es: ***dev/dio/out/0***

Si deseamos encender el LED3, el mismo que se usa en el ejemplo blinking, lo podríamos hacer con las siguientes instrucciones:

```
static int32_t fd_out;
uint8 outputs;

ciaak_start();
fd_out = ciaaPOSIX_open("/dev/dio/out/0", O_RDWR);
outputs = 0x20;
ciaaPOSIX_write(fd_out, &outputs, 1);
```

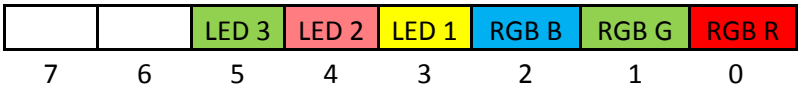
Primero estamos definiendo dos variables, una estática tipo entero de 32 bits para guardar el fildes y un entero sin signo de 8 bits en el cual se pasara qué Led se va a escribir.

Luego se debe llamar a la función ***ciaak_start()*** que se encarga de inicializar el Kernel y los dispositivos de la CIAA, entre ellos las DIO que utilizaremos y posteriormente con la función ***ciaaPOSIX_open*** se obtiene el fildes correspondiente a las salidas digitales de los Leds.

A la función ***ciaaPOSIX_open*** se le pasan dos parámetros. Primero la ruta en formato de archivo del dispositivo (dev/dio/out/0) y luego se indica que se abre como Lectura y Escritura (O_RDWR).

Finalmente para encender el Led se llama a la función ***ciaaPOSIX_write*** a la cual se le pasan tres parámetros. Primero el fildes obtenido previamente, luego un buffer de 8 bits donde se indica el o los Leds que se desean encender/apagar y por último el tamaño del buffer en bytes, que corresponderá a 1.

El registro de escritura de los Leds tiene la siguiente estructura



Como en este caso deseamos encender el LED 3 y mantener el resto apagado, pasamos como buffer un 1 en la posición 5 y el resto ceros, que corresponde a valor en hexadecimal 0x20.

3.1 Ejemplo Blinking

En el ejemplo blinking lo que se hace es encender y apagar el Led verde (LED3) con un periodo de medio segundo, es decir, el mismo permanecerá encendido 250ms y apagado otros 250ms, repitiéndose esta secuencia indefinidamente. Esto se implementa por medio de una tarea con el sistema operativo OSEK. Dicha tarea se activa periódicamente por medio del seteo de una alarma.

Para mejor entendimiento del ejemplo explicaremos cada una de las partes del código de nuestro archivo fuente myproject.c (originalmente blinking.c).

Primero que nada, como se muestra en la Figura 11, se incluye el Copyright, una breve descripción, los autores y su historia de modificaciones. Todo esto es requerido en cualquier archivo nuevo para respetar las reglas establecidas en el Proyecto CIAA.

*myproject.c

```
/* Copyright 2014, Mariano Cerdeiro
/** \brief Blinking_echo example source file
 **
 ** This is a mini example of the CIAA Firmware.
 **
 **/

/** \addtogroup CIAA_Firmware CIAA Firmware
 ** @{ */
/** \addtogroup Examples CIAA Firmware Examples
 ** @{ */
/** \addtogroup Blinking Blinking_echo example source file
 ** @{ */

/*
 * Initials     Name
 * ---------------------------
 * MaCe         Mariano Cerdeiro
 * PR           Pablo Ridolfi
 * JuCe         Juan Cecconi
 * GMuro        Gustavo Muro
 * ErPe         Eric Pernia
 */

/*
 * modification history (new versions first)
 * -----------------------------------------------------------
 * 20150603 v0.0.3   ErPe change uint8 type by uint8_t
 *                        in line 172
 * 20141019 v0.0.2   JuCe add printf in each task,
 *                        remove trailing spaces
 * 20140731 v0.0.1   PR   first functional version
 */

```

Figura 11: Comentario iniciales del archivo blinking.c

A continuación se hacen los includes necesarios, entre ellos se incluyen los archivos de configuración para el sistema operativo, las librerías POSIX, el núcleo o kernel de la CIAA y el archivo cabecera correspondiente a nuestro proyecto. Después se declara una variable estática tipo entero de 32bits que será utilizada para pasar el fildes. Esto se presenta en la Figura 12.

```
68
69 /*==================[inclusions]=============================================*/
70 #include "os.h"               /* <= operating system header */
71 #include "ciaaPOSIX_stdio.h"  /* <= device handler header */
72 #include "ciaaPOSIX_string.h" /* <= string header */
73 #include "ciaak.h"            /* <= ciaa kernel header */
74 #include "myproject.h"         /* <= own header */
75
76 /*==================[macros and definitions]=================================*/
77
78 /*==================[internal data declaration]==============================*/
79
80 /*==================[internal functions declaration]=========================*/
81
82 /*==================[internal data definition]===============================*/
83
84 /** \brief File descriptor for digital output ports
85  *
86  * Device path /dev/dio/out/0
87  */
88 static int32_t fd_out;
89
90 /*==================[external data definition]===============================*/
91
92 /*==================[internal functions definition]==========================*/
93
94 /*==================[external functions definition]==========================*/
```

Figura 12: Includes del ejemplo blinking.

Luego como se aprecia en la Figura 13 se declara el Main en donde únicamente se hace la inicialización del sistema operativo en el modo de aplicación 1.

```
95 /** \brief Main function
96  *
97  * This is the main entry point of the software.
98  *
99  * \returns 0
100 *
101 * \remarks This function never returns. Return value is only to avoid compiler
102 *          warnings or errors.
103 */
104 int main(void)
105 {
106    /* Starts the operating system in the Application Mode 1 */
107    /* This example has only one Application Mode */
108    StartOS(AppMode1);
109
110    /* StartOs shall never returns, but to avoid compiler warnings or errors
111     * 0 is returned */
112    return 0;
113 }
114
```

Figura 13: Main del ejemplo blinking.

Siguiendo, en la Figura 14, se declara la función ErrorHook que se llamará desde el sistema operativo, si una API devuelve un error.

```
115 /** \brief Error Hook function
116  *
117  * This fucntion is called from the os if an os interface (API) returns an
118  * error. Is for debugging proposes. If called this function triggers a
119  * ShutdownOs which ends in a while(1).
120  *
121  * The values:
122  *    OSErrorGetServiceId
123  *    OSErrorGetParam1
124  *    OSErrorGetParam2
125  *    OSErrorGetParam3
126  *    OSErrorGetRet
127  *
128  * will provide you the interface, the input parameters and the returned value.
129  * For more details see the OSEK specification:
130  * http://portal.osek-vdx.org/files/pdf/specs/os223.pdf
131  *
132  */
133 void ErrorHook(void)
134 {
135    ciaaPOSIX_printf("ErrorHook was called\n");
136    ciaaPOSIX_printf("Service: %d, P1: %d, P2: %d, P3: %d, RET: %d\n", OSErrorGetService
137    ShutdownOS(0);
138 }
139
```

Figura 14: Función ErrorHook.

Más abajo, en la Figura 15, se declara la primera tarea ***InitTask.*** Dentro de la misma se inicializa el kernel de la CIAA y los dispositivos con la función ***ciaak_start()***. Se hace una salida por pantalla en caso de ejecutar el programa en una PC y se abre el dispositivo correspondiente a los Leds. Por último se establece una alarma para la próxima tarea.

```
144 TASK(InitTask)
145 {
146    /* init CIAA kernel and devices */
147    ciaak_start();
148
149    /* print message (only on x86) */
150    ciaaPOSIX_printf("Init Task...\n");
151
152    /* open CIAA digital outputs */
153    fd_out = ciaaPOSIX_open("/dev/dio/out/0", O_RDWR);
154
155    /* activate periodic task:
156     *  - for the first time after 350 ticks (350 ms)
157     *  - and then every 250 ticks (250 ms)
158     */
159    SetRelAlarm(ActivatePeriodicTask, 350, 250);
160
161    /* terminate task */
162    TerminateTask();
163 }
164
```

Figura 15: Tarea de inicialización InitTask.

Terminando el archivo, en la Figura 16, se declara la tarea PeriodicTask que será la encargada de encender y apagar los Leds. Con la alarma previamente establecida, esta se llamará cada cierto periodo de tiempo, en nuestro caso cada 250ms.

Aquí aparece la función POSIX de lectura ***ciaaPOSIX_read***. La misma tiene tres argumentos: el fildes del dispositivo, un buffer en el cual se muestra la información leída y por último el tamaño del buffer, en este caso 1 byte.

En esta tarea se lee el estado de las salidas, recibiéndolo en la variable ***outputs***, luego se realiza una XOR con el bit correspondiente al Led 3 y se escribe la salida. Básicamente lo que hace es cambiar el estado del pin correspondiente al Led 3 cada vez que se ejecuta. Por lo tanto la primera vez leerá el valor cero y escribirá el valor 1 en la posición 5 encendiendo el Led 3. En la próxima ejecución de la tarea, 250ms después, se comenzará leyendo un 1 en la posición 5 que haciendo la XOR se escribirá un 0 apagando el Led. De esta forma el mismo titilará.

La función ***ciaaPOSIX_printf()*** mostrará por consola la palabra ‘Blinking’ cada vez que se ejecute la tarea.

```
165 /** \brief Periodic Task
166  *
167  * This task is started automatically every time that the alarm
168  * ActivatePeriodicTask expires.
169  *
170  */
171 TASK(PeriodicTask)
172 {
173    uint8_t outputs;
174
175    /* write blinking message */
176    ciaaPOSIX_printf("Blinking\n");
177
178    /* blink output */
179    ciaaPOSIX_read(fd_out, &outputs, 1);
180    outputs ^= 0x20;
181    ciaaPOSIX_write(fd_out, &outputs, 1);
182
183    /* terminate task */
184    TerminateTask();
185 }
186
187 /** @} doxygen end group definition */
188 /** @} doxygen end group definition */
189 /** @} doxygen end group definition */
190 /*==================[end of file]============================================*/
191
```

Figura 16: Tarea periódica PeriodicTask.

Se recomienda a aquellos que no se encuentren familiarizados con sistemas operativos de tiempo real leer el documento *'**Breve introducción a OSEK-VDX'*** de Mariano Cerdeiro. Se puede encontrar en el sitio oficial de la CIAA haciendo clic aquí.

3.2 Cambio de la frecuencia en el ejemplo Blinking

Si deseamos modificar la frecuencia con la que parpadea el Led, solo debemos modificar el tercer parámetro de *SetRelAlarm* en la primer tarea. Por ejemplo para encenderlo cada 1 segundo (500ms apagado – 500ms encendido) realizamos la siguiente modificación:

SetRelAlarm(ActivatePeriodicTask, 350, **250**);

↓

SetRelAlarm(ActivatePeriodicTask, 350, **500**);

3.3 Cambio del Led a encender en el ejemplo Blinking

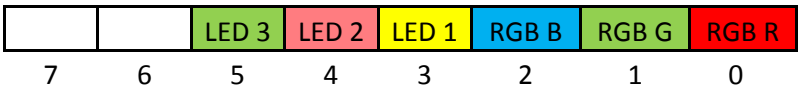
Si queremos que en vez de parpadear el Led 3 (Verde), lo haga por ejemplo el Led Azul del RGB, sólo debemos modificar la posición del 1 que escribimos. El RGB Azul corresponde a la posición 2. En hexadecimal equivale a 0x04. Por lo cual debemos hacer el siguiente cambio:

```
ciaaPOSIX_read(fd_out, &outputs, 1);
outputs ^= 0x20;
ciaaPOSIX_write(fd_out, &outputs, 1);
```

↓

```
ciaaPOSIX_read(fd_out, &outputs, 1);
outputs ^= 0x04;
ciaaPOSIX_write(fd_out, &outputs, 1);
```

De manera similar para cualquiera de los demás Leds según su mapeo:



4 Ejemplo incorporando los Switches

La EDU-CIAA tiene cuatro switches:

- TEC1
- TEC2
- TEC3
- TEC4

Para poder leer sus estados se lo hace de forma similar que con los Leds, teniendo en cuenta que se ubican de la siguiente forma en el registro leído por la función POSIX correspondiente:

Si por ejemplo, deseamos leer el Switch 1 (TEC1) debemos realizar las siguientes instrucciones:

```
static int32_t fd_input;
uint8 inputs;

ciaak_start();
fd_in = ciaaPOSIX_open("/dev/dio/in/0", O_RDONLY);
ciaaPOSIX_read(fd_in, &inputs, 1)
```

Análogamente que con los Leds, primero abrimos el dispositivo y obtenemos el fildes con la función ***ciaaPOSIX_open***. En este caso la ruta de los mismo es: **/dev/dio/in/0** y como son entradas, se los abre para solo lectura (O_RDONLY). Luego con la función ***ciaaPOSIX_read*** obtenemos el estado de los mismos en la variable tipo entero sin signos de 8 bits ***inputs***.

4.1 Secuencia circular activada por un Switch.

A continuación se presentará un ejemplo donde se utilizan los seis Leds y dos de los switches. Consiste en una secuencia circular, la cual se inicia con uno de los switches y se pausa con el otro. La lectura del pulsador se lleva a cabo haciendo polling utilizando una tarea periódica que se ejecuta cada 50ms.

Se parte del ejemplo blinking, del cual se aprovecha la tarea periódica para realizar la secuencia circular y se crea una nueva tarea, también periódica, para "preguntar" por el estado de los pulsadores cada un determinado tiempo y a partir de estos realizar la acción deseada.

En primer lugar definiremos las variables que necesitamos para los fildes, un contador para realizar la secuencia y una variable auxiliar para el estado activada/pausada. Esto corresponde a la Figura 17.

```
82 /*==================[internal data definition]=============================*/
83
84 /** \brief File descriptor for digital output ports
85  * Device path /dev/dio/out/0
86  * File descriptor for digital input ports
87  * Device path /dev/dio/in/0
88  */
89 static int32_t fd_out, fd_in;
90 static uint32_t Periodic_Task_Counter;
91 static uint8 SecuenciaON=0;
92
```

Figura 17: Definición de variables necesarias para el ejemplo.

Posteriormente, como se ve en la Figura 18, se agrega en el archivo myproject.oil la configuraciones de la nueva tarea para los switches y su respectiva alarma.

```
86   ALARM ActivatePeriodicTask {
87      COUNTER = HardwareCounter;
88      ACTION = ACTIVATETASK {
89         TASK = PeriodicTask;
90      }
91   }
92
93   TASK SwitchesTask {
94      PRIORITY = 2;
95      ACTIVATION = 1;
96      STACK = 512;
97      TYPE = EXTENDED;
98      SCHEDULE = NON;
99      RESOURCE = POSIXR;
100     EVENT = POSIXE;
101  }
102
103  ALARM ActivateSwitchesTask {
104     COUNTER = HardwareCounter;
105     ACTION = ACTIVATETASK {
106        TASK = SwitchesTask;
107     }
108  }
109
110  COUNTER HardwareCounter {
111     MAXALLOWEDVALUE = 1000;
112     TICKSPERBASE = 1;
113     MINCYCLE = 1;
114     TYPE = HARDWARE;
```

Figura 18: Agregando una nueva tarea y alarma para los switches en el archivo .oil

Luego, dentro de la tarea ***InitTask*** agregamos la obtención del fildes correspondientes a las DIO de entrada ***fd_in*** e inicializamos nuestro contador ***Periodic_Task_Counter*** como se muestra en la Figura 19. Además seteamos una alarma para la tarea ***SwitchesTask***, que se utilizará para realizar la lectura de los switches cada 50ms.

```
143 /** \brief Initial task
144  *
145  * This task is started automatically in the application mode 1.
146  */
147 TASK(InitTask)
148 {
149    /* init CIAA kernel and devices */
150    ciaak_start();
151
152    /* print message (only on x86) */
153    ciaaPOSIX_printf("Init Task...\n");
154
155    /* open CIAA digital outputs */
156    fd_out = ciaaPOSIX_open("/dev/dio/out/0", O_RDWR);
157    /* open CIAA digital inputs */
158    fd_in = ciaaPOSIX_open("/dev/dio/in/0", O_RDONLY);
159
160    /* activate periodic task:
161     *  - for the first time after 350 ticks (350 ms)
162     *  - and then every 250 ticks (250 ms)
163     */
164    Periodic_Task_Counter = 0;
165    SetRelAlarm(ActivatePeriodicTask, 350, 250);
166
167    /* activate periodic switches task:
168     *  - for the first time after 350 ticks (350 ms)
169     *  - and then every 100 ticks (50 ms)
170     */
171    SetRelAlarm(ActivateSwitchesTask, 350, 50);
172
173    /* terminate task */
174    TerminateTask();
175 }
176
```

Figura 19: Modificando tarea InitTask.

Después, en la tarea ***PeriodicTask*** (Figura 20) es donde realizamos encendido/apagado de los Leds. Si el estado de la variable ***SecuenciaON*** está en uno se realizará la secuencia circular de los Leds. De lo contrario, estando en cero, se frenará. Para la secuencia, primero se lee el valor de las salidas y se hace una XOR con un '1' que se va desplazando a la izquierda cada vez que se vuelve a ejecutar la sentencia. Esto lo que hace es ir cambiando el estado de un bit por vez, es decir un Led por vez. Posteriormente se escribe el nuevo valor encendiendo o apagando ciertos Leds.

```
177 /** \brief Periodic Task
178  *
179  * This task is started automatically every time that the alarm
180  * ActivatePeriodicTask expires.
181  *
182  */
183
184 TASK(PeriodicTask)
185 {
186    uint8 outputs;
187
188    /* Circular sequence */
189    if(SecuenciaON == 1){
190
191        ciaaPOSIX_read(fd_out, &outputs, 1);
192
193        outputs ^= 1<<Periodic_Task_Counter;
194
195        ciaaPOSIX_write(fd_out, &outputs, 1);
196
197        Periodic_Task_Counter++;
198        if(Periodic_Task_Counter>5){
199            Periodic_Task_Counter=0;
200        }
201    }
202
203    /* terminate task */
204    TerminateTask();
205 }
206
```

Figura 20: Modificando tarea PeriodicTask.

Por último, se crea una nueva tarea, ***SwitchesTask*** (Figura 21), en la cual se realiza la lectura de los pulsadores. Se comienza leyendo el estado de los mismos, el cual se guarda en ***inputs.*** Luego haciendo una máscara sobre el primer bit para el Switch 1 y sobre el segundo bit para el Switch 2 cambiamos el estado de la variable ***SecuenciaON*** según cuál de los dos fue presionado. Presionar el Switch 1 pondrá en alto nuestra variable auxiliar. Ejecutándose la secuencia circular, mientras que el Switch 2 hará que vuelva a cero, pausándola.

```
207 /** \brief Switches Task
208  *
209  * This task is started automatically every time that the alarm
210  * ActivateSwitchesTask expires.
211  *
212  */
213
214 TASK(SwitchesTask)
215 {
216    uint8 inputs;
217
218    /* Read Switch 1 and change state of variable pulsador */
219    ciaaPOSIX_read(fd_in, &inputs, 1);
220
221    if((inputs&0x01) == 1){
222       SecuenciaON=1;
223    }
224    if((inputs&0x02) == 2){
225       SecuenciaON=0;
226    }
227
228    /* terminate task */
229    TerminateTask();
230 }
```

Figura 21: Tarea SwitchesTask.

Se adjunta en el apéndice el código fuente correspondiente a este ejemplo.

5 Apéndice

5.1 Código fuente de la secuencia circular de Leds activada por el Switch 1

Archivo myproject.c

```
/* Copyright 2014, Mariano Cerdeiro
 * Copyright 2014, Pablo Ridolfi
 * Copyright 2014, Juan Cecconi
 * Copyright 2014, Gustavo Muro
 *
 * This file is part of CIAA Firmware.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. Neither the name of the copyright holder nor the names of its
 *    contributors may be used to endorse or promote products derived from this
 *    software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 *
 */

/** \brief Blinking_echo example source file
 **
 ** This is a mini example of the CIAA Firmware.
 **
 **/

/** \addtogroup CIAA_Firmware CIAA Firmware
 ** @{ */
```

```
/** \addtogroup Examples CIAA Firmware Examples
 ** @{ */
/** \addtogroup Blinking Blinking_echo example source file
 ** @{ */

/*
 * Initials     Name
 * ---------------------------
 * MaCe         Mariano Cerdeiro
 * PR           Pablo Ridolfi
 * JuCe         Juan Cecconi
 * GMuro        Gustavo Muro
 * ErPe         Eric Pernia
 */

/*
 * modification history (new versions first)
 * -----------------------------------------------------------
 * 20150603 v0.0.3   ErPe change uint8 type by uint8_t
 *                        in line 172
 * 20141019 v0.0.2   JuCe add printf in each task,
 *                        remove trailing spaces
 * 20140731 v0.0.1   PR   first functional version
 */

/*==================[inclusions]=============================================*/
#include "os.h"               /* <= operating system header */
#include "ciaaPOSIX_stdio.h"  /* <= device handler header */
#include "ciaaPOSIX_string.h" /* <= string header */
#include "ciaak.h"            /* <= ciaa kernel header */
#include "myproject.h"         /* <= own header */

/*==================[macros and definitions]=================================*/
#define CantLeds 5

/*==================[internal data declaration]==============================*/

/*==================[internal functions declaration]=========================*/

/*==================[internal data definition]===============================*/

/** \brief File descriptor for digital output ports
 * Device path /dev/dio/out/0
 * File descriptor for digital input ports
 * Device path /dev/dio/in/0
 */
static int32_t fd_out, fd_in;
static uint32_t Periodic_Task_Counter;
static uint8 SecuenciaON=0;

/*==================[external data definition]===============================*/

/*==================[internal functions definition]==========================*/

/*==================[external functions definition]==========================*/
```

```
/** \brief Main function
 *
 * This is the main entry point of the software.
 *
 * \returns 0
 *
 * \remarks This function never returns. Return value is only to avoid compiler
 *          warnings or errors.
 */
int main(void)
{
   /* Starts the operating system in the Application Mode 1 */
   /* This example has only one Application Mode */
   StartOS(AppMode1);

   /* StartOs shall never returns, but to avoid compiler warnings or errors
    * 0 is returned */
   return 0;
}

/** \brief Error Hook function
 *
 * This fucntion is called from the os if an os interface (API) returns an
 * error. Is for debugging proposes. If called this function triggers a
 * ShutdownOs which ends in a while(1).
 *
 * The values:
 *    OSErrorGetServiceId
 *    OSErrorGetParam1
 *    OSErrorGetParam2
 *    OSErrorGetParam3
 *    OSErrorGetRet
 *
 * will provide you the interface, the input parameters and the returned value.
 * For more details see the OSEK specification:
 * http://portal.osek-vdx.org/files/pdf/specs/os223.pdf
 *
 */
void ErrorHook(void)
{
   ciaaPOSIX_printf("ErrorHook was called\n");
   ciaaPOSIX_printf("Service: %d, P1: %d, P2: %d, P3: %d, RET: %d\n",
OSErrorGetServiceId(), OSErrorGetParam1(), OSErrorGetParam2(),
OSErrorGetParam3(), OSErrorGetRet());
   ShutdownOS(0);
}

/** \brief Initial task
 *
 * This task is started automatically in the application mode 1.
 */
TASK(InitTask)
{
   /* init CIAA kernel and devices */
   ciaak_start();
```

```
   /* print message (only on x86) */
   ciaaPOSIX_printf("Init Task...\n");

   /* open CIAA digital outputs */
   fd_out = ciaaPOSIX_open("/dev/dio/out/0", O_RDWR);
   /* open CIAA digital inputs */
   fd_in = ciaaPOSIX_open("/dev/dio/in/0", O_RDONLY);

   /* activate periodic task:
    *  - for the first time after 350 ticks (350 ms)
    *  - and then every 250 ticks (250 ms)
    */
   Periodic_Task_Counter = 0;
   SetRelAlarm(ActivatePeriodicTask, 350, 250);

   /* activate periodic switches task:
    *  - for the first time after 350 ticks (350 ms)
    *  - and then every 100 ticks (50 ms)
    */
   SetRelAlarm(ActivateSwitchesTask, 350, 50);

   /* terminate task */
   TerminateTask();
}

/** \brief Periodic Task
 *
 * This task is started automatically every time that the alarm
 * ActivatePeriodicTask expires.
 *
 */

TASK(PeriodicTask)
{
   uint8 outputs;

   /* Circular sequence */
   if(SecuenciaON == 1){

         ciaaPOSIX_read(fd_out, &outputs, 1);

         outputs ^= 1<<Periodic_Task_Counter;

         ciaaPOSIX_write(fd_out, &outputs, 1);

         Periodic_Task_Counter++;
         if(Periodic_Task_Counter>CantLeds){
               Periodic_Task_Counter=0;
         }
   }

   /* terminate task */
   TerminateTask();
}
```

```
/** \brief Switches Task
 *
 * This task is started automatically every time that the alarm
 * ActivateSwitchesTask expires.
 *
 */

TASK(SwitchesTask)
{
   uint8 inputs;

   /* Read Switch 1 and change state of variable pulsador */
   ciaaPOSIX_read(fd_in, &inputs, 1);

   if((inputs&0x01) == 1){
      SecuenciaON=1;
   }
   if((inputs&0x02) == 2){
      SecuenciaON=0;
   }

   /* terminate task */
   TerminateTask();
}

/** @} doxygen end group definition */
/** @} doxygen end group definition */
/** @} doxygen end group definition */
/*==================[end of file]============================================*/
```

Agregado en el archivo myproject.oil

```
   TASK SwitchesTask {
      PRIORITY = 2;
      ACTIVATION = 1;
      STACK = 512;
      TYPE = EXTENDED;
      SCHEDULE = NON;
      RESOURCE = POSIXR;
      EVENT = POSIXE;
   }

   ALARM ActivateSwitchesTask {
      COUNTER = HardwareCounter;
      ACTION = ACTIVATETASK {
         TASK = SwitchesTask;
      }
   }
```