



Facultad de Ciencias  
Físico Matemáticas  
y Naturales



Universidad  
Nacional de  
San Luis

# Ingeniería Electrónica OSD

# TRABAJO FINAL DE

# PROCESADORES II

**Tutorial de manejo de placa EDU CIAA  
con RTOS OSEK**

**Profesores:** Roberto Martín Murdocca.

**Sergio Javier Hernández  
Velázquez.**

**Alumnos:** Suarez Facundo, Ezequiel Duperre  
Jonathan Sáez Ríos.



# Índice

<b>1. Introducción</b>	<b>3</b>
1.1. Origen del proyecto CIAA . . . . .	3
1.2. Descripción de la placa . . . . .	4
<b>2. Instalacion de Software</b>	<b>7</b>
2.1. Conceptos previos . . . . .	7
2.2. Firmware de la EDU CIAA . . . . .	7
2.3. Estructura de Directorios de Firmware de EDU CIAA . . . . .	9
2.4. Iniciación a través de ejemplos . . . . .	10
2.5. Instalación de IDE . . . . .	10
<b>3. Desinstalación</b>	<b>16</b>
<b>4. Primeros Pasos con la EDU-CIAA</b>	<b>16</b>
4.1. Programacion en Baremetal a través de libreria sAPI . . . . .	16
4.2. Implementacion de ejemplo N°1: Salidas Digitales . . . . .	19
4.3. Implementacion de ejemplo N°2: Entradas y Salidas Digitales . . . . .	21
4.4. Implementacion de ejemplo N°3: Entradas Analógicas y Uso de Display 7 Segmentos	22
4.5. Implementacion de ejemplo N°4: Conversor Digital/Analógico (DAC) . . . . .	24
4.6. Implementacion de ejemplo N°5: Uso de UART y Display LCD 16x2 . . . . .	26
4.7. Implementacion de ejemplo N°6: Comunicación Serial . . . . .	28
4.8. Implementacion de ejemplo N°7:Comunicación Serial-Utilización Display LCD 16x2	29
<b>5. Sistemas Operativos en Tiempo Real</b>	<b>30</b>
5.1. Tareas . . . . .	31
5.1.1. Modelo de Tareas Básicas . . . . .	32
5.1.2. Modelo de Tareas Extendidas . . . . .	32
5.2. Scheduler . . . . .	34
5.3. Modos de Aplicación . . . . .	34
5.4. Eventos . . . . .	34
5.5. Clases de Conformidad . . . . .	35
5.6. Configuración y Generación . . . . .	36
5.6.1. Implementacion de ejemplo N°1: Tareas en OSEK . . . . .	37
5.6.2. Implementacion de ejemplo N°2: Múltiples Tareas en OSEK . . . . .	43
5.6.3. Implementacion de ejemplo N°3: Eventos en OSEK . . . . .	44
5.7. Alarmas . . . . .	45
5.7.1. Implementacion de ejemplo N°4: Eventos y Alarmas en OSEK . . . . .	45
5.7.2. Implementacion de ejemplo N°5: Alarmas en OSEK . . . . .	46
5.8. Interrupciones . . . . .	47
5.8.1. Implementacion de ejemplo N°6: Interrupciones en OSEK . . . . .	47
5.8.2. Implementacion de ejemplo N°7: Interrupciones en OSEK . . . . .	49
5.8.3. Implementacion de ejemplo N°8: Interrupciones en OSEK . . . . .	50
<b>6. Implementacion de ejemplo N°9: Inclusión de librería LCD y formación de librerias propias para trabajar</b>	<b>51</b>



## 1. Introducción

En este trabajo se pretende desarrollar un tutorial que permita al usuario poder iniciar sus primeros proyectos usando RTOS y comprender la arquitectura de los procesadores Cortex.

Este estudio aborda en primer lugar, las características principales de la placa y la correcta instalación del software sobre Windows para permitir el desarrollo de códigos sobre ella; además se incluyen posibles problemas que puedan suceder en el proceso junto con sus soluciones.

La siguiente sección comienza introduciendo al usuario en el uso de la placa a través de la biblioteca *sAPI* (realizada por Eric Pernia) ésta nos permite el desarrollo de programas utilizando lenguaje C. La idea es brindar una explicación simple de la arquitectura de los procesadores Cortex y a su vez brindar una base para la siguiente sección del informe.

Luego, se introduce al usuario en la programación a través de el sistema operativo *OSEK OS*, el cual es el método de programación en el que se proyectó cuando se realizó el diseño de la placa. Nuestra meta es fijar las bases conceptuales principales de los sistemas operativos en tiempo real, e introducir al usuario a la programación de códigos simples y alentar al usuario a profundizar sobre este estudio.

### 1.1. Origen del proyecto CIAA

Sobre julio de 2013, la Secretaría de Planeamiento Estratégico Industrial del Ministerio de Industria de la Nación (SPEI) y la Secretaría de Políticas Universitarias del Ministerio de Educación de la Nación (SPU) convocaron a la Asociación Civil para la Investigación, Promoción y Desarrollo de los Sistemas Electrónicos Embebidos (ACSE) y a la Cámara de Industrias Electrónicas, Electromecánicas y Luminotécnicas (CADIEEL) a participar en el "Plan Estratégico Industrial 2020". A partir de dicha convocatoria se inició el desarrollo de la Computadora Industrial Abierta Argentina (CIAA).

El pedido inicial fue que desde el sector académico (ACSE) y desde el sector industrial (CADIEEL) se presenten propuestas para agregar valor en distintas ramas de la economía (maquinaria agrícola, bienes de capital, forestal, textil, alimentos, etc.) a través de la incorporación de sistemas electrónicos en procesos productivos y en productos de fabricación nacional. Debe destacarse que muchas empresas argentinas de diversos sectores productivos no incorporaban electrónica en sus procesos productivos o en sus productos, otras utilizaban sistemas electrónicos obsoletos, muchas utilizaban sistemas importados y sólo unas pocas utilizaban diseños propios basados en tecnologías vigentes y competitivas.

A partir de esta situación, la ACSE y CADIEEL propusieron desarrollar un sistema electrónico abierto de uso general, donde toda su documentación y el material para su fabricación estuviera libremente disponible en internet, con el objetivo de que dicho sistema pueda ser fabricado por la mayoría de las empresas PyMEs nacionales, y realizar modificaciones en base a las necesidades específicas que puedan tener.

Hoy en día la CIAA está disponible en la versión CIAA-NXP y otras seis versiones están en



elaboración: CIAA-ATMEL, CIAA-FSL, CIAA-PIC, CIAA-RX, CIAA-ST, CIAA-TI. Además, se está trabajando en el firmware y en el software, para que la CIAA se pueda programar en lenguaje C utilizando una API especialmente diseñada para ser compatible con los estándares POSIX y que sea portable a diversos sistemas operativos de tiempo real.

Desde la concepción del proyecto, el diseño de la placa se encuentra pensada para soportar las condiciones hostiles de los ambientes industriales los que abundan ruidos, vibraciones, temperaturas extremas, picos de tensión e interferencias electromagnéticas, y además se diseñó de modo tal que pueda ser fabricada en Argentina[1].

## 1.2. Descripción de la placa

La CIAA es una placa electrónica provista de un microcontrolador y puertos de entrada y salida, cuyo diseño se encuentra disponible en Internet. Esta placa fue concebida para ser utilizada para sistemas de control de procesos productivos, agroindustria, automatización, entre otras; es importante destacar que gracias a la posibilidad del acceso a la información de dicha plataforma, cualquier empresa que desee utilizarla para la elaboración de sus productos puede rediseñarla; de modo que esto fomenta el diseño y la fabricación nacional de sistemas electrónicos.



Figura 1: Placa EDU CIAA

La placa EDU CIAA es la versión educativa de ésta, la cual se encuentra diseñada con el propósito de conseguir una plataforma base para el desarrollo de proyectos educativos, en este caso, se busca proporcionar las bases del desarrollo de códigos utilizando RTOS.

El procesador que utiliza es el LPC4337, basado en el ARM Cortex M4, utilizado para aplicaciones en sistemas embebidos, dicho sistema incluyen el procesador Cortex M0. Utiliza una memoria flash de 1Mb, memoria de 264 kB de SRAM, memoria ROM de 64 kB, E2PROM de 16kB, y una memoria OTP de 64 bit.

La frecuencia de trabajo del procesador alcanza los 204 Mhz. Una característica vital del procesador, es que provee soporte para la depuración en JTAG, con posibilidad de incluir hasta 8 breakpoints, y 4 watchpoints.



Dicho procesador brinda una interface que hace posible extender hasta 164 pines de entrada-salida de propósito general (GPIO), provee una interface USB Host/Device 2.0 de alta velocidad con soporte para acceso directo de memoria, una interface UART 550 con soporte DMA , tres USART 550 con soporte para DMA[2].

Como perifericos analógicos, debe destacarse la inclusión de un DAC de 10 bits, con soporte DMA y frecuencia de conversión de 400000 muestras por segundo. Dos ADC's con soporte DMA, y frecuencia de conversión de 400000 muestras por segundo, con un numero máximo de 8 canales sobre cada ADC. Y un ADC de 12 bits de 6 canales con soporte DMA, y frecuencia de conversión que puede alcanzar hasta los  $80,10^6$  muestras por segundo.

El cristal oscilador posee un rango de operación desde 1 Mhz hasta 25 Mhz, este procesador incluye un reloj de tiempo real de baja potencia, el cual utiliza un oscilador de cristal.

La placa provee alimentación de 3,3 V (cuyo rango oscila entre 2,2 V hasta 3,6 V), y es capaz de operar en cuatro modos, los cuales se denominan *sleep*, *deep-sleep*, *power-down*,y *deep power-down*.Es posible restablecer la operación de la placa desde los modos *deep-sleep*, *power-down*, y *deep power-down*, a través de interrupciones externas.

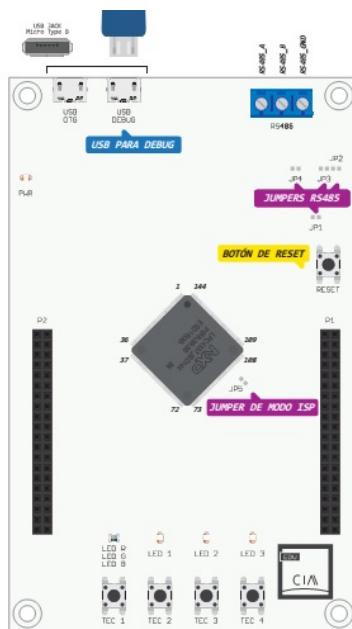


Figura 2: Imagen frontal de placa

Sobre la Figura 3 se proporciona el diagrama en bloques de la placa[3], puede observarse que la placa cuenta con 2 puertos micro-USB (uno para aplicaciones y debugging, otro para alimentación); 4 salidas digitales implementadas con leds RGB, 4 entradas digitales con pulsadores; 1 puerto de comunicaciones RS485 con bornera. La Figura 2 nos muestra una imagen frontal de la placa; nótese la presencia de dos puertos sobre los cuales se ubican los pines correspondientes a la placa, la Figura 4 ilustra el distribución de dichos pines sobre cada puerto.

Sobre el puerto P1, se ubican los siguientes módulos:



- 3 entradas analógicas (ADC0,1,2 y 3)
- 1 salida analógica (DAC0).
- 1 puerto I2C.
- 1 puerto asincrónico full duplex (para RS-232).
- 1 puerto CAN.
- 1 conexión para un teclado 3x4.

Sobre el puerto P1, se ubican los siguientes módulos:

- 1 puerto Ethernet
- 1 puerto SPI
- 1 puerto para Display LCD con 4 bits de datos, Enable y RS.
- pines genéricos de I/O.

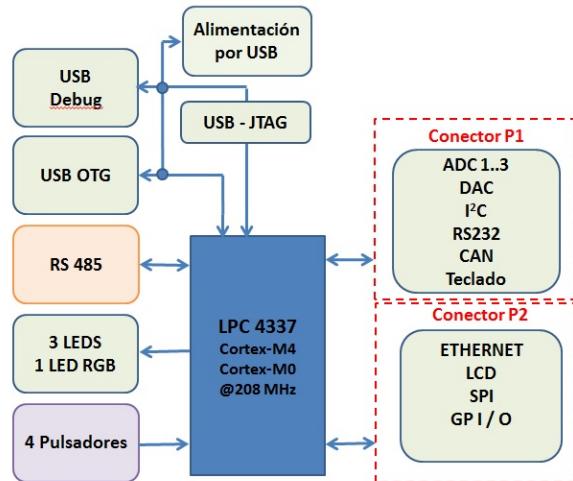


Figura 3: Diagrama en bloques de EDU CIAA basado en LPC4337.

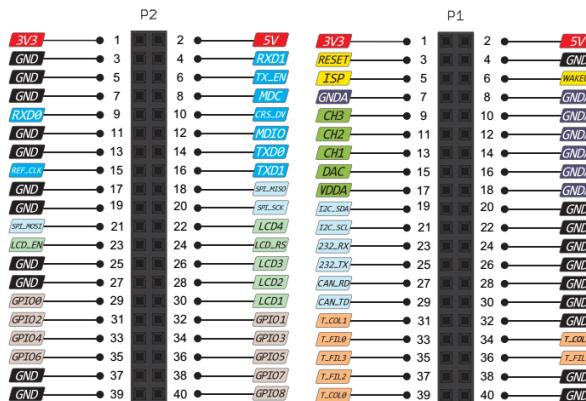


Figura 4: Distribución de pines de la placa



## 2. Instalacion de Software

### 2.1. Conceptos previos

El desarrollo de códigos para sistemas embebidos tiene ciertas semejanzas con el desarrollo de aplicaciones en las PC, en este caso particular se utiliza un compilador llamado *GCC* con soporte para la compilación de proyectos sobre los procesadores basados en la arquitectura ARM. En este caso particular, el compilador utilizado para el procesador de la EDU CIAA (el LPC4337) se lo denomina *arm-none-eabi-gcc*.

Para la ejecución de la depuración de algún programa previamente compilado, el hardware de la CIAA viene provisto con el chip *FT2232H*, que se encarga de hacer un puente entre la interfase JTAG del microcontrolador, y el USB que conecta a la PC en el puerto USB dedicado al debug. Mediante la herramienta de código abierto *OpenOCD (On Chip Debugger)* se controla el chip *FT2232H* por el USB y además todo lo referido al JTAG. Luego la herramienta de depuración *GDB* utilizado en el IDE-Eclipse que se instala, se comunica sobre el puerto 3333 (TCP) que el *OpenOCD* tiene en escucha esperando la conexión[4].

Debe tenerse en cuenta que el chip *FT2232H* posee 2 canales de comunicación independientes (A y B), sin embargo, ambos salen por el mismo USB, de modo que la PC detecta 2 dispositivos distintos (en realidad es uno compuesto). Uno de ellos, se conecta al JTAG manejado por *OpenOCD* como fue mencionado, mientras que el otro se ve como un puerto virtual COM. Este último sirve principalmente para la depuración.

Dado que al funcionar como dos dispositivos distintos, para cada uno de ellos debe realizarse la instalación de un driver adecuado, en principio debe optarse por realizar la instalación de los drivers por defecto del fabricante FTDI para puerto virtual.

### 2.2. Firmware de la EDU CIAA

Considerando que el usuario previamente ha trabajado sobre placas de desarrollo tales como la MCE Debug, etc, y sobre microcontroladores PIC. Es necesario destacar un concepto teórico que brinda la posibilidad de fundamentar el trabajo sobre la placa EDU CIAA. Al trabajar sobre los otros dispositivos, es común la utilización de programas tales como *MPLABX*, o *PICC* a través del compilador *CCS Compiler*; puntualmente; cuando se inicia un nuevo proyecto a través de la herramienta de creación de la misma, es usual configurar este proyecto de manera que el software IDE genera un archivo *makefile* para la compilación del proyecto.

En este caso, el software IDE de la EDU CIAA trabaja de forma ligeramente distinta, el usuario debe modificar un archivo *makefile* (basándose en un archivo proporcionado previamente, denominado *Makefile.config*) para poder efectuar la compilación del archivo y lograr la correcta configuración del programa, sobre la placa EDU CIAA. Dentro de él se establece la configuración para la arquitectura del procesador utilizado.

Cuando se desea realizar el primer proyecto sobre la placa, el usuario debe crear su propio archivo *Makefile.mine*, de manera que éste se encuentra basada en *Makefile.config* brindado previamente



al momento de establecer un nuevo proyecto añadiendo un Firmware que ha sido diseñado para el funcionamiento de la placa.

Debe tenerse en cuenta que la dinámica de trabajo sobre la placa se encuentra pensada para trabajar sobre la plataforma de versionado *Git*; en este caso en particular, el archivo *Makefile.mine* se encuentra diseñado de forma tal que dicho archivo sea ignorado al sincronizar su repositorio local, con su repositorio remoto (ubicado sobre *Github*).

En *Makefile.mine* se pueden editar y configurar los siguientes parámetros[5]:

1. **ARCH** indica la arquitectura del hardware para la cual se desea compilar. Ej: x86, cortexM4.
2. **CPUTYPE** indica el tipo de CPU. Ej: none, ia32, ia64, lpc43xx.
3. **CPU** indica la CPU para la que se desea compilar. Ej: none, lpc4337.
4. **COMPILER** es el compilador a utilizar. Ej: gcc.
5. **BOARD** es la placa sobre la cual se trabajaca (CIAA-NXP, EDU-CIAA-NXP, etc.)
6. **PROJECT** es el Path al proyecto a compilar. Ej: examples\$(*DS*)*blinkingbase*.

Se utiliza la variable *\$(DS)* para indicar el separador de directorios (de manera automática se usa '/' para linux y '\ para windows).

En el mismo Makefile aparecen al comienzo comentarios donde se indican los valores que pueden tomar estos parámetros.

Otro concepto importante que se tiene en cuenta cuando se desarrollan proyectos propios, es que cada proyecto tiene también su propio archivo *Makefile*. El mismo se encuentra bajo el directorio **mak** en el directorio principal del proyecto o ejemplo. En el ejemplo **examples/blinkin**g el makefile del ejemplo se encuentra en **examples/blinkin/mak** y se llama **Makefile**.

Sobre este archivo *Makefile* contiene las siguientes definiciones:

1. **PROJECT** el nombre del proyecto y por ende nombre del ejecutable.
2. **\$(PROJECT)\_PATH** es el directorio del proyecto.
3. **INCLUDE** los paths a indicar al compilador para buscar includes files.
4. **SRC\_FILES** archivos a compilar ya sean archivos c como c++.
5. **OIL\_FILES** configuración del sistema operativo (si es utilizado).

Cada proyecto incluye en su Makefile los módulos a compilar en una variable llamada MODS, por ejemplo:

```
MODS += modules$(DS)posix
modules$(DS)ciaak
modules$(DS)config
```



```
modules$(DS)bsp
modules$(DS)platforms
```

Es recomendable utilizar \$(DS) en vez de / o \ para mantener la compatibilidad entre sistemas operativos (Linux, Windows, MAC OS).

### 2.3. Estructura de Directorios de Firmware de EDU CIAA

La Figura 5 ilustra los contenidos de la estructura de directorios.

```
Firmware/
├── doc
├── examples
│   ├── adc_dac
│   └── blinking
│   /*
│   |   └── blinking_echo      /* ejemplo básico que hace echo en el bus
│   |   └── serial
│   |       ├── blinking_lwip /* ejemplo utilizando lwip (ethernet) */
│   |       ├── blinking_modbus /* ejemplo utilizando modbus RS485 (ascii) */
│   |       └── blinking_modbus_master /* ejemplo utilizando modbus master */
│   └── rtos_example          /* ejemplo de utilización de FreeOSEK */
└── externals
    ├── base                /* archivos básicos necesarios para
    |   └── compiler/linkear */
    |   └── ceedling           /* ceedling utilidad para correr los unit
    test */
    ├── drivers
    |   ├── lcov              /* utilidad para analizar el coverage de los
    unit test */
    |   └── lwip               /* stack tcpip */
    ├── Makefile
    ├── Makefile.config
    └── propio
        └── Makefile.mine      /* Makfile con la configuración del usuario
    */
    └── modules
        ├── base
        |   └── y_linkear
        |       ├── ciaak
        |       ├── drivers
        |       ├── libs
        |       ├── modbus
        |       ├── plc
        |       ├── posix
        |       ├── rtos
        |       └── systests
        |       /*
        |       |   ├── kernel de la ciaa */
        |       |   ├── drivers para posix */
        |       |   ├── algunas librerías genéricas */
        |       |   ├── módulo para manejo de modbus */
        |       |   ├── módulo de PLC (en desarrollo) */
        |       |   ├── posix like library */
        |       |   ├── RTOS de CIAA-Firmware (FreeOSEK) */
        |       |   /* utilidad para correr los tests en HW (en
```

- <http://proyecto-ciaa.com.ar/devwiki/>

Figura 5: Estructura de directorios del Firmware

En el directorio principal luego de hacer un git clone o al bajar una release oficial se pueden encontrar los siguientes Directorios y Archivos[6]:

#### Directorio "externals"(Software y Tools Externos)

Este directorio contiene el Software y Tools externos al CIAA-Firmware, que son necesarios para compilar, testear, etc. el Firmware. Tenga en cuenta que el Software y Tools en esta carpeta no son parte de CIAA-Firmware y pueden contener otras licencias. Sobre el Cuadro 1 se ilustra los contenidos del directorio y su descripción,mientras que el Cuadro 2 contiene todos los archivos generados por el CIAA-Firmware.

ceedling	Tool utilizada para los Unit Tests o Pruebas Unitarias
base	Fuentes, headers y linker scripts necesarios para poder compilar y linkar el código en la plataforma
drivers	Drivers provistos por el proveedor del chip, los cuales son luego adaptados al formato de la CIAA.

Cuadro 1: Utilidades,archivos,y drivers para la placa



bin	Contiene el binario del proyecto, es el archivo que se va a correr en la PC o a cargar en el CIAA-Firmware
gen	Archivos generados de OSEK RTOS
lib	Por cada Módulo el make genera un archivo .a, osea una librería
obj	Todos los archivos fuentes son compilados a object files y almacenados en este directorio

Cuadro 2: Descripción de archivos generados en un proyecto en RTOS.

## 2.4. Iniciación a través de ejemplos

En el Firmware de la placa se pueden varios ejemplos, estos se encuentran en la carpeta **examples** y pueden ser utilizados como base para iniciar cualquier proyecto.

Cualquiera de los ejemplos puede ser copiado y utilizado de base para nuevos proyectos. Por ejemplo con el siguiente comando: `cp -r examples/blinkingprojects/my_project` y adaptando el Makefile.mine indicado: `PROJECT_PATH = projects/my_project`.

## 2.5. Instalación de IDE

El entorno de desarrollo integrado (IDE) posibilita el trabajo en un ambiente ameno, además provee las herramientas necesarias para el desarrollo de aplicaciones en el Firmware de forma automática. La CIAA utiliza una versión modificada de la plataforma de software (IDE) *Eclipse*, denominada *CIAA-Software-IDE*, la cual contiene herramientas de programación tales como editor de texto, compilador, plataforma para depuración, etc. Sobre la página web del proyecto, se provee un instalador llamado *CIAA-IDE-SUITE*, desde donde se puede configurar automáticamente todas las herramientas necesarias para trabajar con la placa. Este instalador solamente es para los usuarios que poseen Windows XP o superior.

El paquete de instalación incluye:

1. **Eclipse**
2. **PHP (Hypertext Pre-processor)** es un lenguaje de programación de uso general de código desde el lado del servidor, originalmente diseñado para el desarrollo de contenido dinámico. En este caso, se utiliza solamente en forma de scripts para poder generar algunos archivos del **Sistema Operativo OSEK**
3. **Cygwin** es una consola que se ejecuta en Windows, de modo de emular la consola de comandos de Linux. Cuenta con todos los comandos, y el compilador GCC, propio del sistema operativo libre.
- Una vez realizada la descarga del instalador, se ejecuta dicha aplicación, la Figura 6 muestra el arranque del instalador, sobre ella, debe seleccionarse *Siguiente*. A continuación deben aceptarse los términos de uso.



Figura 6: Arranque del instalador del software IDE

- Sobre la ventana siguiente deben elegirse cuáles componentes se desea instalar, en el caso que el usuario no posea la placa disponible, no es necesario instalar los drivers, si se adquiere dicha placa en un momento posterior,dado que los drivers se instalan junto con el IDE, los mismos quedarán en la carpeta de destino para su instalación en forma manual; otra forma de instalar los los controladores es ejecutar el instalador del CIAA-IDE Suite y tildar únicamente la opción **drivers** al momento de seleccionar los componentes a instalar.La Figura 7 ilustra lo explicado anteriormente.

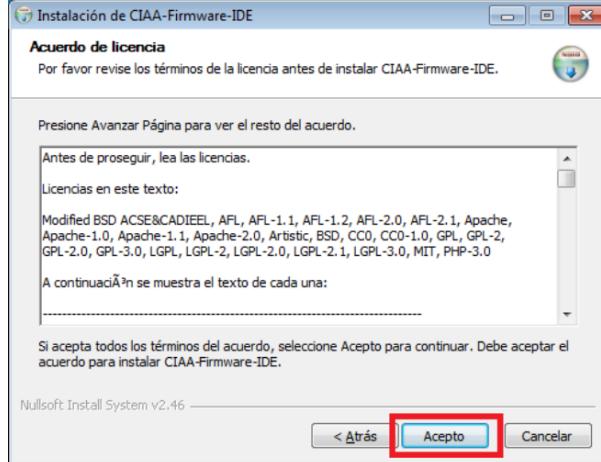


Figura 7: Arranque del instalador del software IDE

- Sobre la Figura 8, se deben seleccionar los componentes a instalar.

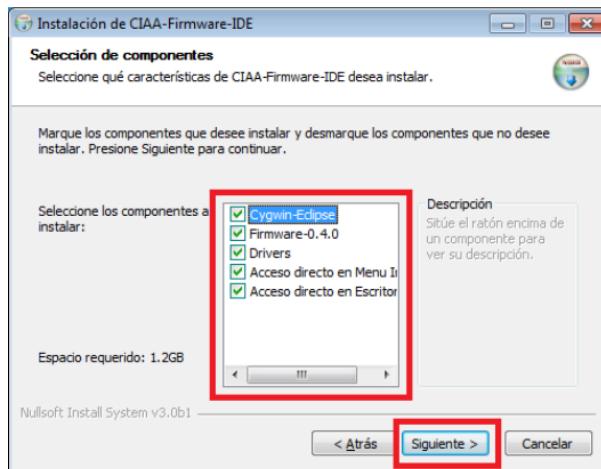


Figura 8: Selección de componentes del instalador

- A continuación debe establecerse la dirección en donde se desea instalar el entorno. La ventana que corresponde a este proceso se ilustra en la Figura 9. En caso de que se desee cambiar dicha dirección, debe tenerse la precaución de no elegir una dirección donde los directorios posean espacios en sus nombres. Es recomendable no cambiar la unidad de instalación, pues en los siguientes pasos del documento se utilizarán direcciones que harán referencia a esta carpeta de instalación, y si se cambia, se deberán cambiar consecuentemente dichas direcciones.

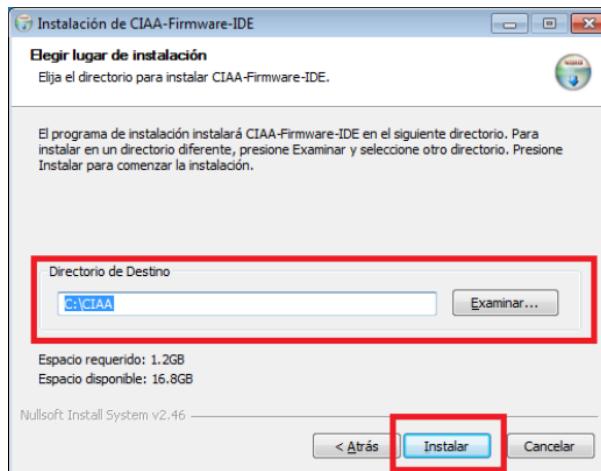


Figura 9: Elección de la ruta de instalación

- Luego de dicha configuración, se inicia automáticamente el proceso de instalación. En un momento aparecerá una ventana emergente, similar a la que se muestra en la Figura 10, en donde el programa pregunta si se dispone del hardware, pues para la instalación del driver es necesario conectar la placa.
- De no ser así, aún puede continuar la instalación haciendo click en 'No'. Por el contrario, si se dispone de la EDU-CIAA, se debe hacer click en 'Yes', y emergirá otra ventana, como se muestra en la Figura 11.

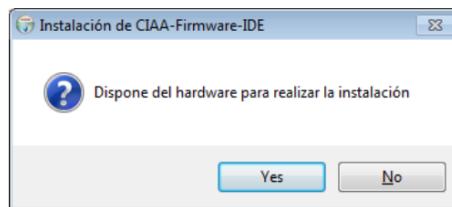


Figura 10: Instalación de los drivers: primera instancia



Figura 11: Instalación de drivers si se dispone del hardware

- Una vez finalizada esta etapa, se procede a la instalación de los drivers por defecto del fabricante FTDI para puerto virtual. Este proceso se ilustra en las figuras 12 ,13,14,15 y 16.

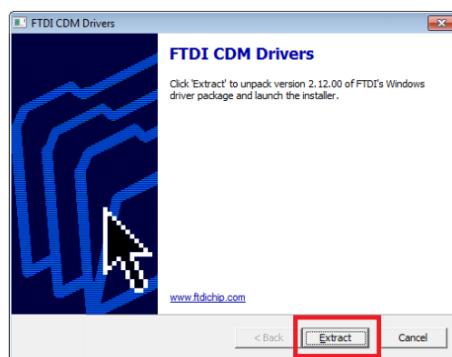


Figura 12: Instalador de drivers FTDI parte 1

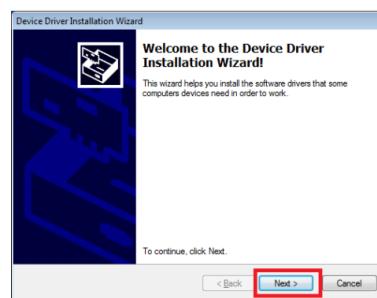


Figura 13: Instalador de drivers FTDI parte 2

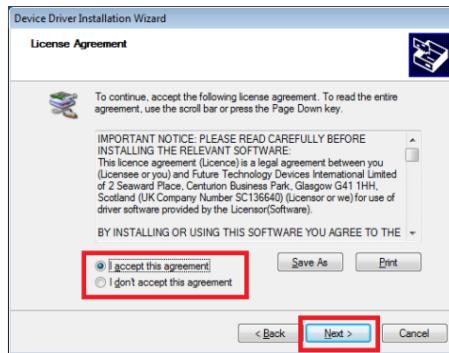


Figura 14: Instalador de drivers FTDI parte 3

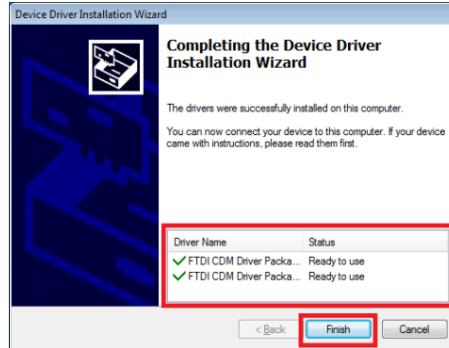


Figura 15: Instalador de drivers FTDI parte 4

- Una de las posibles fallas que pueden surgir a través de este proceso, se presenta al producirse una falla en la comunicación a través del puerto virtual FTDI, que impide la correcta comunicación entre la placa y el entorno IDE. Su corrección debe efectuarse manualmente, fuera del instalador, y es posible la aparición de una ventana de error emergente como la que se muestra en la Figura 16.

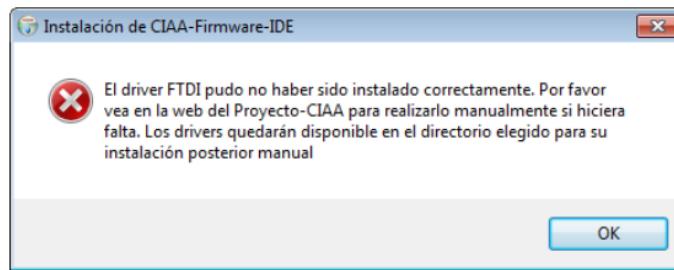


Figura 16: Instalador de drivers FTDI parte 5

- El instalador incluye en la carpeta donde se instaló el software (Por defecto,  $C : /CIAA$ ) un programa que configura el driver del controlador serie, emulado por la placa, para que uno de ellos pueda ser utilizado como interfaz JTAG. Dicho programa se llama *Zadig\_Win\_7\_2\_1\_1.exe*, la Figura 17 muestra una ilustración de la ubicación del programa.



Nombre	Fecha de modifica...	Tipo	Tamaño
cygwin	29/08/2016 13:05	Carpetas de archivos	
Firmware	29/08/2016 21:21	Carpetas de archivos	
IDE4PLC	28/08/2016 0:17	Carpetas de archivos	
Linux	28/08/2016 0:17	Carpetas de archivos	
local-repo	28/08/2016 0:16	Carpetas de archivos	
usbdriver	28/08/2016 0:17	Carpetas de archivos	
driver_winusb_zadig_ft2232h	10/04/2015 16:43	Imagen PNG	25 KB
Setup_Win_7_FTDI_2_12.00	08/06/2015 10:15	Aplicación	2.188 KB
SetUsers	31/03/2015 10:49	Archivo por lotes ...	2 KB
uninstall	29/08/2016 13:26	Aplicación	61 KB
zadig_Win_7_2.1.1	08/06/2015 10:16	Aplicación	5.069 KB

Figura 17: Instalador de drivers FTDI parte 6

- Al abrir la aplicación, se presenta la Figura 18. Antes de proceder, el usuario debe conectar la placa, posteriormente, debe abrir el menú contextual *Options* y presionar sobre *List all devices*

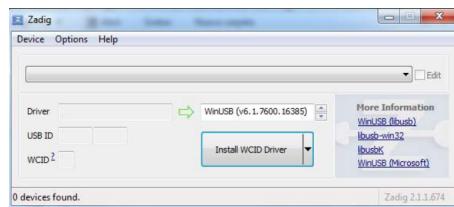


Figura 18: Entorno del software corrector Zadig para Windows

- Aparecerá una lista de dispositivos de comunicación relacionados al USB. Se tiene que buscar aquellos cuyos nombres tengan relación con el puerto serie (puede aparecer Dual RS232-HS, USB Serial Converter, o algo similar).

Por lo general, aparecerán 2 con el mismo nombre, excepto que uno es Interface 0 y el otro Interface 1, como se muestra en la Figura 19 (la lista de drivers que se muestra puede diferir, dependiendo de la computadora que se utilice).

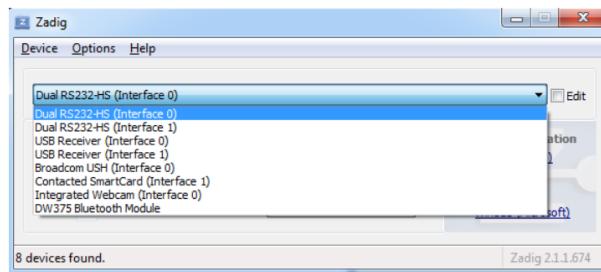


Figura 19: Lista de dispositivos vinculados a USB

- Para configurar el driver:

- 1 seleccionar la **Interfase 0**
- 2 elegir el **“WinUSB v6.1”**
- 2 hacer click en el botón **“Replace Driver”**.

La Figura 20 muestra la ventana ya configurada:

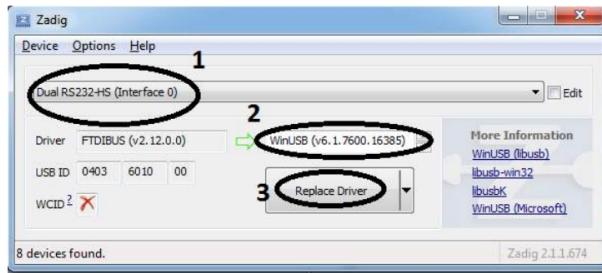


Figura 20: Configuración del Zadig para el reemplazo del driver

### 3. Desinstalación

Si se instaló el Software de CIAA-IDE y luego se desea desinstalarlo, se debe tener especial cuidado en quitar cualquier contenido que se quiera conservar de la carpeta *C : \ CIAA*, o el directorio de instalación elegido. Esto se debe a que el desinstalador del Software CIAA-IDE elimina el directorio y todo su contenido.

## 4. Primeros Pasos con la EDU-CIAA

### 4.1. Programacion en Baremetal a través de libreria sAPI

Debido a que la programación de la placa en Baremetal puede tornarse un tanto engorrosa en principio, es posible utilizar una librería diseñada para la implementación de proyectos en Baremetal con la placa EDU CIAA, dicha librería se la denomina *sAPI*. Esta biblioteca implementa una API simple para la programación de dicha placa, es necesario comentar que una API es una interfaz de programación de aplicaciones, y consta de un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca; para permitir ser utilizada por otro software como una capa de abstracción en la programación (aunque no necesariamente) entre los niveles o capas inferiores y las superiores del software[7].

La motivación para el desarrollo de la biblioteca sAPI surge de la necesidad de manejar los periféricos directamente desde una máquina virtual de Java para el desarrollo de Java sobre la CIAA y corresponde a la parte de bajo nivel de las clases de periféricos en Java que básicamente bindea a funciones escritas en C. Luego se extendió la misma para facilitar el uso de la EDU-CIAA-NXP a personas no expertas en la arquitectura del LPC4337 facilitando el uso de esta plataforma.

La idea es tener periféricos abstractos y lo más genéricos posibles. Que sea bien independiente de la arquitectura y en lo posible que las funciones sean todas del tipo:

- moduloConfig();
- moduloRead();
- moduloWrite();

Los siguientes módulos están incluidos:

- Tipos de datos.



- Mapa de periféricos.
- Plataforma.
- Tick.
- Retardo.
- E/S Digital.
- E/S Analógica.
- Uart.

Es necesario destacar que actualmente se encuentra disponible para las plataformas EDU CIAA NXP (microcontrolador NXP LPC4337) y para la plataforma CIAA NXP (microcontrolador NXP LPC4337). La Figura 21 ilustra como son las distintas capas de software y la correspondiente ubicación de la *sAPI*.

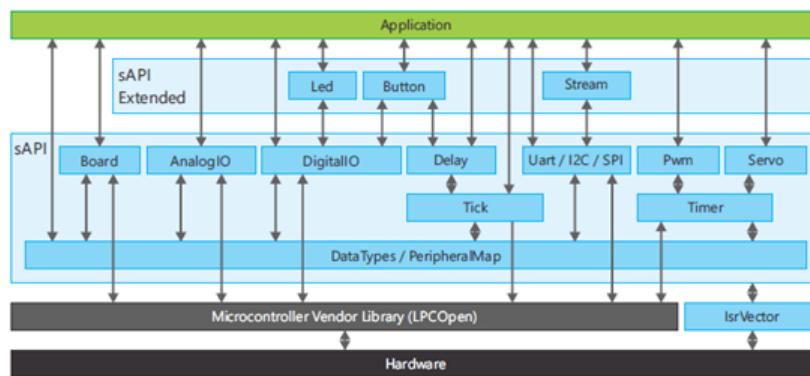


Figura 21: Capas de abstracción de software

#### Breve descripción de los módulos de la biblioteca:

- **sAPI\_Config:** Contiene configuraciones de la biblioteca.
- **sAPI\_DataTypes:** Define las siguientes constantes:
  - Estados lógicos, *TRUE* y *FALSE*.
  - Estados funcionales *ON* y *OFF*
  - Estados eléctricos *HIGH* y *LOW*
  - Tipos de datos: Booleanos (*bool\_t*), enteros sin signo (*uint8\_t*, *uint16\_t*, *uint32\_t*, *uint64\_t*), enteros con signo (*int8\_t*, *int16\_t*, *int32\_t*, *int64\_t*).
- **sAPI\_IsrVector:** Contiene la tabla de vectores de interrupción.
- **sAPI\_Board:** Contiene la función de configuración para inicialización de la plataforma de hardware.



- **sAPI\_PeripheralMap:** Contiene el mapa de periféricos, para la interpretación de éste, el usuario debe utilizar el diagrama esquemático de la placa y el archivo *PeripheralMap.txt* ubicado sobre el directorio *sapi-bm*.
- **sAPI\_DigitalIO:** Utilizada para el manejo de entradas y salidas digitales.

#### Utilización de librería sAPI:

Es posible descargar la versión más reciente de dicha biblioteca, a través del repositorio de la misma, ubicado en la página web (especificar la referencia a la página web del repositorio).

Luego de haber descargado la versión actual de sAPI se procede a copiar la carpeta *sapi\_bm* y pegarla en *C:/CIAA/Firmware/modules*.

A continuación, el usuario debe crear un directorio el cual contiene al proyecto, y sobre el mismo, deben realizarse los directorios que contengan las cabeceras, el *Makefile*, y los archivos fuentes.

Al utilizar la biblioteca *sAPI* para un proyecto en Baremetal, se debe modificar el makefile del mismo dentro de la carpeta mak y cambiar la última línea de: **MODS += externals \$(DS)drivers**, y reemplazarla por: **MODS += modules \$(DS)sapi\_bm**.

Luego, sobre cada archivo fuente en la que se ejecuten instrucciones que contenga esta librería, se debe realizar la inclusión del mismo mediante la instrucción: **# include "sAPI.h"**. Debe aclararse que en la versión 0.3.0 no es necesario incluir el archivo *chip.h* dado que la propia biblioteca lo incluye, además, esta librería maneja el vector de interrupción, por lo tanto, para el usuario que haya trabajado anteriormente con proyectos escritos en instrucciones de *LPCOpen*, debe tenerse en cuenta que ya no se necesita realizar el archivo que contenga las direcciones de las subrutinas para la atención de las interrupciones. El ejemplo "*baremetal*" del Firmware en el que se realiza este tutorial, trae consigo dicho archivo.

Para la compilación y depuración del programa mediante el IDE del Firmware, se debe colocar sobre la carpeta Firmware y mediante click derecho, el usuario debe ubicarse en *Propiedades* → *C/C++ Build* → *Behaviour*. A continuación debe modificar sobre la rama *Clean*, y reemplazar la palabra *Clean\_generate* por la palabra *clean*, como puede apreciarse en la Figura 22.

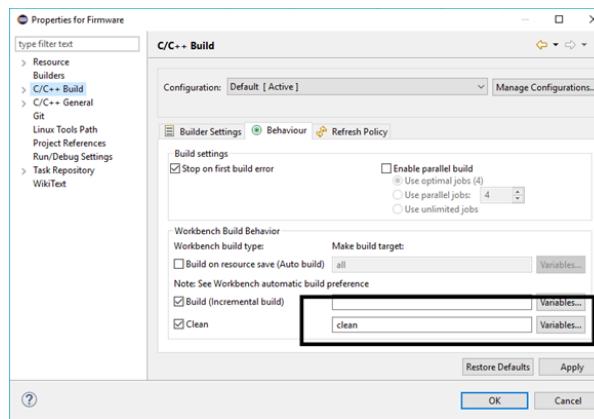


Figura 22: Cambio de “clean\_generate” a “clean”.



## 4.2. Implementacion de ejemplo N°1: Salidas Digitales

El primer ejemplo aborda las salidas digitales, a través de los leds que provee la placa EDU-CIAA. Este programa enciende los leds de la placa en forma secuencial, de izquierda a la derecha. Luego de finalizar esta secuencia, se apagan de derecha a izquierda.

- Sobre el IDE de la placa, debe agregarse un nuevo directorio, realizando click derecho sobre *Firmware*, luego se selecciona la opción *New → Folder*; a continuación se abre una ventana que permite designar el nombre de dicho directorio, en el cual se ubica el proyecto que se implementa. El usuario debe recordar que para compilar dicho proyecto debe establecer la ruta que le corresponde sobre el archivo *Makefile*. Asimismo, deben agregarse los directorios *etc*, *mak*, *inc*, y *src*, sobre el directorio creado recientemente.

Para generar el archivo cabecera, el usuario debe ubicarse sobre la carpeta *inc*, y luego oprimiendo click derecho, debe seleccionar la opción *New → Header File*. A continuación se abre una ventana cuyo aspecto es similar al de la Figura 23.

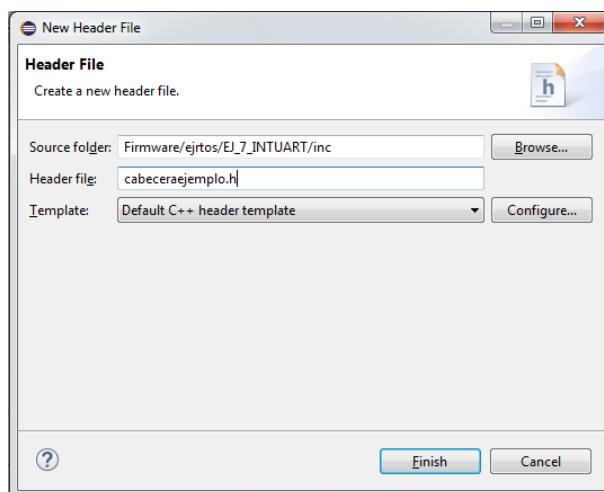


Figura 23: Generación de archivo cabecera para implementación de proyecto.

En ella, se debe establecer el nombre del archivo cabecera. Al terminar la escritura de su nombre, debe escribirse la terminación *.h*, y oprimir el botón *Finish*, para concluir la generación de dicho archivo. En este caso, el archivo cabecera se define como *Ejemplo\_1.h*.

- Para generar el archivo fuente, el usuario debe ubicarse sobre la carpeta *inc*, y luego oprimiendo click derecho, debe seleccionar la opción *New → Source File*. A continuación se abre una ventana cuyo aspecto es similar al de la Figura 24, en ella, se debe establecer el nombre del archivo fuente, el cual en este caso es *Ejemplo\_1.c*. Al terminar la escritura de su nombre, debe escribirse la terminación *.c*, y oprimir el botón *Finish*, para concluir la generación de dicho archivo.
- Ingresar el archivo cabecera de la librería *sAPI* para poder referenciar los módulos a utilizar y referenciar el archivo cabecera del proyecto en caso de que el usuario desee declarar de forma directa alguna función o algún identificador. La Figura 25 ilustra dicho concepto.

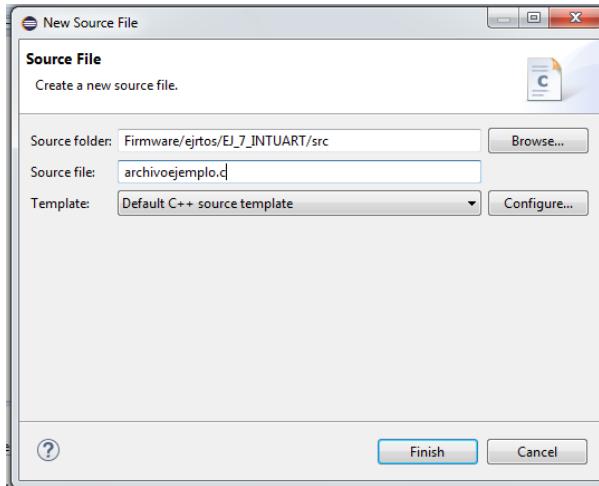


Figura 24: Generación de archivo fuente para implementación de proyecto.

```

5 /*Autores: Suearez Facundo, Saez Jonathan, Duperre Ezquiel. */
6
7 #include "Ejemplo_1.h" // Libreria propia del programa
8 #include "sAPI.h"      //libreria sAPI

```

Figura 25: Generación de archivo fuente para implementación de proyecto.

- Una vez declarada la función principal *main()*, es importante destacar algunas funciones utilizadas:
  - *boardconfig()*: Esta función se encarga de la configuración para inicializar la plataforma del hardware. No se debe especificar ningún parámetro.
  - *tickconfig()*: Configura una interrupción periódica de temporizador cada tickRateMS-value milisegundos para utilizar de base de tiempo del sistema. El primer parámetro es el tiempo en milisegundos en el que se desee interrumpir el periférico Systick y aumente un contador que se usa para el módulo delay o para lo que se requiera.
  - *digitalConfig()*: Inicializa los puertos GPIO de la placa y define sus funciones como entradas y salidas digitales. El primer parámetro debe ser 0, mientras que el segundo debe ser *ENABLE\_DIGITAL\_IO*.
  - *digitalWrite()*: Permite el establecimiento en *ALTO* o en *BAJO* de un pin configurado como salida digital. Sobre el primer parámetro de la función debe establecerse el nombre del PIN, teniendo en cuenta la definición realizada sobre el módulo *sAPI\_PeripheralMap*. El segundo parámetro que debe determinarse es el estado funcional *ON* u *OFF*.
  - *delay()*: Permite implementar una temporización en ms sobre el parámetro de dicha función.
- Para realizar la compilación debe seleccionarse el directorio *Firmware* y posteriormente hacer click sobre el botón *Build*.



### 4.3. Implementacion de ejemplo N°2: Entradas y Salidas Digitales

El siguiente ejemplo tiene como objetivo el manejo de las entradas y salidas digitales, dicho programa enciende los leds instalados sobre la placa, de acuerdo con la cantidad de veces que se oprime la tecla *TEC1*. Sobre dicho ejemplo se implemento una maquina de estado, a través del programa *uModel Factory* para poder establecer un orden sobre los eventos y las acciones que debe ejecutar el procesador. Debe tenerse en cuenta que se implementan multiples archivos fuentes y cabeceras, en particular, el archivo fuente *funciones.c* define las acciones y la evaluación de los eventos y *funciones.h* establece la declaración de dichas funciones y de los estados.

La funcion *digitalRead* se encarga de las entradas digitales. En este caso se pregunta si se ha oprimido la tecla 1, como se puede apreciar esta misma se activa por bajo, por eso se pregunta si la lectura es igual a OFF.

- Realizar la generación de los archivos cabecera y fuente como se establece sobre la sección 4.2.
- A través del programa *uModel Factory*, implementar la siguiente Maquina de Estado Finita:
  - Estados:
    - Estado **INICIAL**: Contexto en el cual, si se oprime la tecla *TEC1*, se enciende un color del led RGB.
    - Estado **ROJO**: Contexto en el cual, si se oprime la tecla *TEC1*, se enciende *LED1*.
    - Estado **ENCENDIDO1**: Contexto en el cual, si se oprime la tecla *TEC1*, se enciende *LED2*.
    - Estado **ENCENDIDO2**: Contexto en el cual, si se oprime la tecla *TEC1*, se enciende *LED3*.
    - Estado **ENCENDIDO3**: Contexto en el cual, si se oprime la tecla *TEC1*, se enciende un color del led RGB.
  - Eventos:
    - Evento **PRESIONO**: Se ha oprimido la tecla *TEC1*.
    - Evento **PRESIONO\_4**: Se ha realizado la primer secuencia apretando la tecla *TEC1*.
    - Evento **PRESIONO\_7**: Se ha realizado la segunda secuencia apretando la tecla *TEC1*.
    - Evento **PRESIONO\_10**: Se ha realizado la tercer y última secuencia apretando *TEC1*.
  - Acciones:
    - Accion **LED\_R**: Encendido de *LEDR,LEDB,LEDG* según corresponda.
    - Accion **P\_LED\_1**: Encendido de *LED1*.
    - Accion **P\_LED\_2**: Encendido de *LED2*.
    - Accion **P\_LED\_3**: Encendido de *LED3*.
    - Accion **BORRO**: Renuevo contador de control.

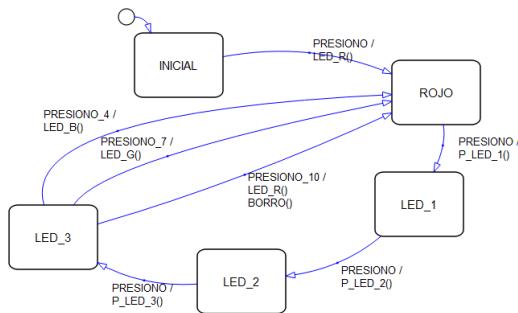


Figura 26: Diagrama de estados de ejemplo N°2

La Figura 26 especifica el diagrama de estados del ejemplo:

- Incluir sobre el archivo fuente, el archivo cabecera *funciones.h*
- Incluir sobre el archivo fuente *funciones.c* el archivo cabecera de la librería *sAPI* y el archivo cabecera del proyecto.

#### 4.4. Implementacion de ejemplo N°3: Entradas Analógicas y Uso de Display 7 Segmentos

En este ejemplo se realizan dos tareas distintas, dependiendo de la tecla pulsada (*TEC1* y *TEC2*). Si se pulsa la tecla *TEC1*, se encienden los Leds en una secuencia determinada. Mientras que si se pulsa la tecla *TEC2*, se realiza una muestra de los dígitos desde 1 hasta 9 a través del display 7 segmentos.

Se implementa una maquina de estados en donde se establecen los prototipos de funciones que corresponden a los eventos y las acciones que acontecen.

Para el manejo del display se utiliza una biblioteca. Para ello, deben generarse dos archivos los cuales pertenecen a la cabecera y a la fuente de dicha biblioteca. Sobre la cabecera deben establecerse las inclusiones de la biblioteca *sAPI* y las declaraciones de las funciones que corresponden con el manejo del dispositivo.

En este tutorial se realiza el manejo de un display de 7 segmentos del tipo cátodo común, de manera que para provocar el encendido de un led perteneciente al arreglo, se debe establecer en alto la salida de el pin conectado a dicho led. La biblioteca que provee este tutorial determina el conexionado de los pines del display, sobre los correspondientes terminales GPIO ubicados en la placa EDU CIAA. La Figura 27 ilustra los terminales de la placa y del display utilizados para la conexión del dispositivo.

##### Uso de Entradas Analógicas:

En este código se hará uso del conversor analógico digital que posee esta placa. La idea es introducir una tensión analógica mediante la entrada CH1 y luego convertirla en un valor digital, para que posteriormente dentro del código, mediante operaciones matemáticas, poder mostrar dicho valor por el display.

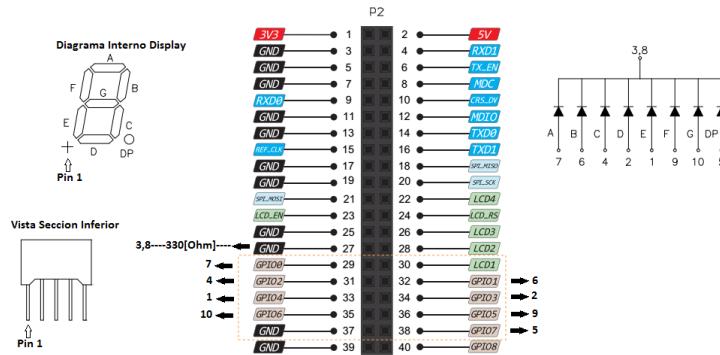


Figura 27: Diagrama de conexión display 7 segmentos.

- Realizar la generación de los archivos cabecera y fuente como se establece sobre la sección 4.2.
- A través del programa *uModel Factory*, implementar la siguiente Maquina de Estado Finita:
  - Estados:
    - Estado **INICIAL**: Estado inicial en el cual, en presencia del evento *NO\_APRETO*, el programa se establece en el mismo estado; mientras que para el caso que suceda evento *APRETO1*, el programa se establece en el estado *ESTADO1*; en el caso de la ocurrencia del evento *APRETO2*, el programa se establece en el estado *ESTADO2*.
    - Estado **ESTADO1**: Sobre dicho estado se realiza la secuencia de luces y la muestra de los dígitos decimales sobre el display.
    - Estado **ESTADO2**: Sobre dicho estado se muestra de un dígito decimal en caso de que se oprima la tecla *TEC2*.

• Eventos:

- Evento **NO\_APRETO**: Identifica si no se ha oprimido la tecla *TEC1* o la tecla *TEC2*.
- Evento **APRETO1**: Identifica si se oprime la tecla *TEC1*.
- Evento **APRETO2**: Identifica si se oprime la tecla *TEC2*.

La Figura 28, ilustra el diagrama de estados de este ejemplo:

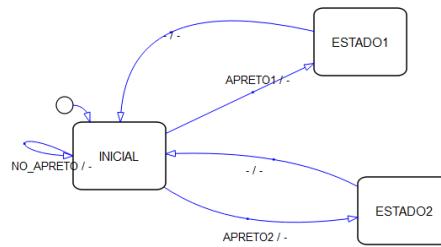


Figura 28: Diagrama de estados de ejemplo N°3.

- Incluir sobre el archivo fuente, el archivo cabecera *funciones.h*



- Incluir sobre el archivo fuente *funciones.c* el archivo cabecera de la librería *sAPI* y el archivo cabecera del proyecto.

#### 4.5. Implementacion de ejemplo N°4: Conversor Digital/Analógico (DAC)

Un convertidor Digital/Analógico (DAC), es un dispositivo electrónico que recibe como dato de entrada un valor digital (en forma de una palabra de "n"bits) y lo transforma a una señal analógica. Cada una de las combinaciones binarias de entrada es convertida en niveles lógicos de tensión de salida. Un DAC transfiere información expresada en forma digital a una forma analógica. Para ubicar la función de este dispositivo conviene recordar que un sistema combina y relaciona diversos subsistemas que trabajan diferentes tipos de información analógica, como son; magnitudes eléctricas, mecánicas, etc.

El siguiente ejemplo muestra al usuario como realizar el uso del módulo ADC de la placa, para realizar la lectura de una entrada analógica. Este programa enciende algunos de los leds de la placa, en función a la magnitud de la tensión leída. Por ejemplo, si la tensión leída es 2[V], entonces se encienden dos leds.

La placa EDU-CIAA posee un DAC de 10 bits de resolución ,por lo tanto,el número de niveles de voltaje de salida analógico que es capaz de generar es de  $2^n = 2^{10} = 1024$ .

Para la implementación del ejemplo, es posible realizar la lectura del terminal medio de un potenciómetro, la Figura 29 ilustra la conexión realizada. Debe tenerse en cuenta que es posible realizar la conexión del potenciómetro a partir de la terminal de 5[V] y la terminal GND que ofrece la placa.

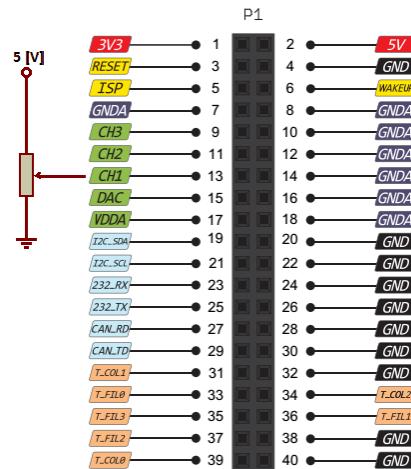


Figura 29: Diagrama de conexión potenciómetro para lectura analógica.

Este ejemplo utiliza maquinas de estado para poder establecer un orden sobre los eventos y las acciones que debe ejecutar el procesador.

La instrucción *analogConfig( ENABLE\_ANALOG\_INPUTS )* se encarga de habilitar las entradas analógicas. Mientras que *analogConfig( ENABLE\_ANALOG\_OUTPUTS )* se encarga de



habilitar las salidas analógicas, con las que trabajara el DAC.

- Realizar la generación de los archivos cabecera y fuente como se establece sobre la sección 4.2.
- A través del programa *uModel Factory*, implementar la siguiente Maquina de Estado Finita:
  - Estados:
    - Estado **UN\_VOLT**: Estado en el cual, la tensión leída es igual a 1[V].
    - Estado **DOS\_VOLT**: Estado en el cual, la tensión leída es igual a 2[V].
    - Estado **TRES\_VOLT**: Estado en el cual, la tensión leída es igual a 3[V].
    - Estado **CUATRO\_VOLT**: Estado en el cual, la tensión leída es igual a 4[V].
    - Estado **INTERMEDIO**: Estado inicial en el cual, la tensión leída no es igual a 1[V],2[V],3[V],4[V],o 5[V].
  - Eventos:
    - Evento **LEYO\_1**: Identifica si la lectura realizada sobre el ADC corresponde a 1[V].
    - Evento **LEYO\_2**: Identifica si la lectura realizada sobre el ADC corresponde a 2[V].
    - Evento **LEYO\_3**: Identifica si la lectura realizada sobre el ADC corresponde a 3[V].
    - Evento **LEYO\_4**: Identifica si la lectura realizada sobre el ADC corresponde a 4[V].
    - Evento **LEYO\_5**: Identifica si la lectura realizada sobre el ADC corresponde a 5[V].
    - Evento **NOES1**: Identifica si la lectura realizada sobre el ADC es distinto a 1[V].
    - Evento **NOES2**: Identifica si la lectura realizada sobre el ADC es distinto a 2[V].
    - Evento **NOES3**: Identifica si la lectura realizada sobre el ADC es distinto a 3[V].
    - Evento **NOES4**: Identifica si la lectura realizada sobre el ADC es distinto a 4[V].
    - Evento **NOES5**: Identifica si la lectura realizada sobre el ADC es distinto a 5[V].
    - Evento **NOESNINGUNO**: Identifica si la lectura realizada sobre el ADC no corresponde a 1[V],2[V],3[V],4[V] y 5[V].

La Figura 30, ilustra el diagrama de estados de este ejemplo:

- Incluir sobre el archivo fuente, el archivo cabecera *funciones.h*
- Incluir sobre el archivo fuente *funciones.c* el archivo cabecera de la librería *sAPI* y el archivo cabecera del proyecto.

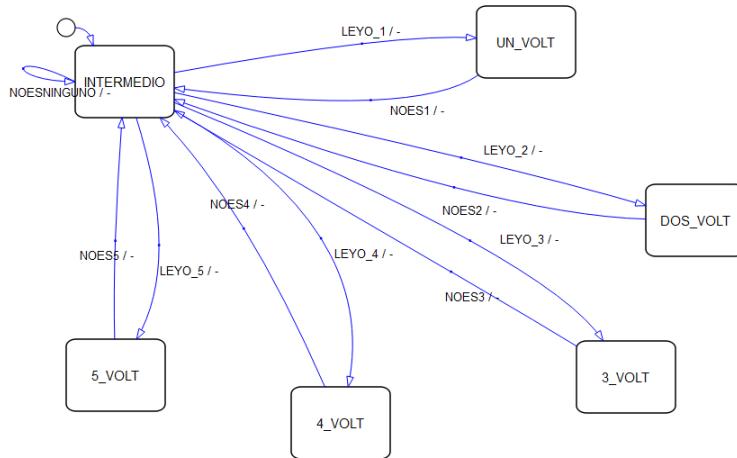


Figura 30: Diagrama de estados de ejemplo N°4.

#### 4.6. Implementacion de ejemplo N°5: Uso de UART y Display LCD 16x2

En este código se utiliza el modulo conversor analógico digital de la placa, para efectuar la lectura de una entrada analógica sobre esta. En el caso que dicha medición sea superior a 2[V], se establece una salida analógica de 2[V] utilizando el módulo conversor digital analógico. De forma similar, si la tensión de entrada resulta mayor a 4[V], se define una salida analógica de 3[V]. Es necesario destacar que la placa ya integra internamente el conversor digital analógico, de modo que no es necesario realizar una interfaz para la ejecución de dicha salida, además, es necesario destacar que la tensión de referencia interna se encuentra establecida, y adquiere un valor igual a 3.3[V]. La Figura 31 ilustra el esquemático que muestra dicho concepto.

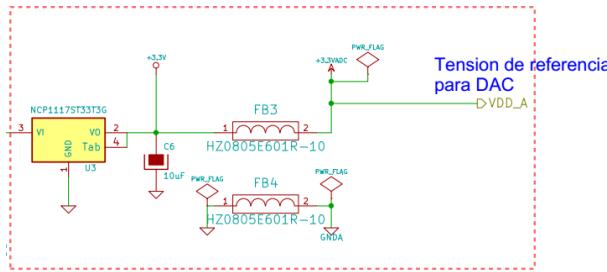


Figura 31: Ilustración de tensión de referencia de DAC.

La instrucción `analogRead()`, permite realizar la lectura de la tensión aplicada sobre el canal 1 del modulo ADC0 de la placa. A partir de la conversión a tensión, el programa aplica la tensión de salida correspondiente.

Este ejemplo también utiliza maquina de estado para establecer un orden sobre los eventos y las acciones que debe ejecutar el procesador.

- Realizar la generación de los archivos cabecera y fuente como se establece sobre la sección 4.2.



- A través del programa *uModel Factory*, implementar la siguiente Maquina de Estado Finita:

- Estados:

- Estado ***LEO\_POTENCIOMETRO***: Estado inicial, en el cual se determina la lectura analógica.
- Estado ***SALIDA\_2***: Estado en el cual se realiza una salida analógica igual a 2[V], en el caso que la lectura realizada sea superior a 2[V].
- Estado ***SALIDA\_3***: Estado en el cual se realiza una salida analógica igual a 3[V], en el caso que la lectura realizada sea superior a 4[V].

- Eventos:

- Evento ***MAYOR\_2***: Identifica si la lectura realizada sobre el ADC es superior a 2[V].
- Evento ***MAYOR\_4***: Identifica si la lectura realizada sobre el ADC es superior a 4[V].
- Evento ***MENOR\_4***: Identifica si la lectura realizada sobre el ADC es inferior a 4[V].
- Evento ***NINGUNO***: Identifica si la lectura realizada sobre el ADC es inferior a 2[V].

- Acciones

- Accion ***APAGAR\_DAC***: Establece una salida de 0 [V] sobre el terminal del DAC.
- Accion ***DAC\_2***: Establece una salida de 2 [V] sobre el terminal del DAC.
- Accion ***DAC\_3***: Establece una salida de 3 [V] sobre el terminal del DAC.

La Figura 32 ilustra el diagrama de estados que corresponde al ejemplo

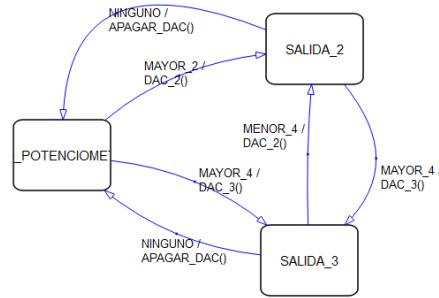


Figura 32: Diagrama de estados de ejemplo N°5

- Incluir sobre el archivo fuente, el archivo cabecera *funciones.h*
- Incluir sobre el archivo fuente *funciones.c* el archivo cabecera de la librería *sAPI* y el archivo cabecera del proyecto.



## 4.7. Implementación de ejemplo N°6: Comunicación Serial

El siguiente ejemplo introduce al usuario en la utilización de comunicación serial, es necesario destacar que para la implementación del ejemplo se utiliza el programa *Octoplus Terminal* para la visualización y simulación del envío y transmisión de datos a través del terminal conectado sobre el puerto USB.

El programa se encuentra disponible de forma gratuita en la dirección web <http://octoplus-terminal.software.informer.com/download/>. La instalación de dicho programa es simple, sin embargo, es necesario destacar que durante dicho proceso, el usuario debe obviar la instalación de los drivers que provee dicho programa.

Si bien ofrece distintas herramientas para la programación de sistemas embebidos, este tutorial se centra en la utilización de dicho programa para el envío y la recepción de datos sobre el terminal perteneciente a la placa. Para ello, a continuación se realizará una serie de pasos para la utilización de *Octoplus* junto con la placa.

Es necesario mencionar que si existe un programa ya cargado previamente, también es posible utilizar *Octoplus* de forma similar.

En este apartado se verá un código base para utilizar la UART del LPC4337 mediante las funciones que incluye la librería sAPI. Las funciones provistas por esta biblioteca son las encargadas de realizar los siguientes pasos básicos para poder utilizar la UART0 [9].

- Habilitar la alimentación del periférico.
- Habilitar el clock al periférico.
- Configurar la velocidad de transmisión.
- Habilitar las FIFO del transmisor y el receptor.
- Configurar la función de los pines para utilizarlo con la UART0.
- Habilitar la interrupción.

El usuario debe seguir los siguientes pasos para implementar este ejemplo:

- Realizar la generación de los archivos cabecera y fuente como se establece sobre la sección 4.2.
- Suponiendo que el usuario ya se encuentran instalados los drivers para la placa, y las herramientas de depuración para la misma, luego de realizar la conexión entre la placa y la computadora a través de *OpenOCD* (sobre la consola *cygwin*), el usuario debe iniciar *Octoplus*.
- A continuación se presenta una pantalla similar a la Figura 33

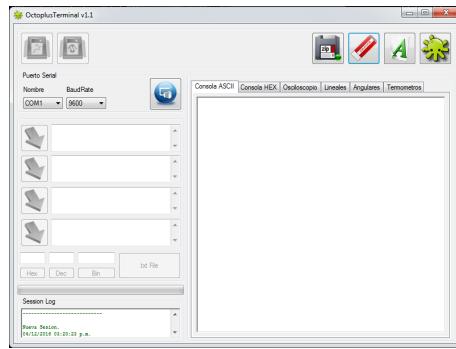


Figura 33: Panel principal de *Octoplus*

- Sobre la sección *Puerto Serial* debe identificarse sobre el menú desplegable *Nombre*, el puerto COM asociado a la placa. La Figura 34 ilustra dicho concepto



Figura 34: Seleccion de puerto COM asociado a la placa

- Debe tenerse en cuenta sobre la instrucción *uartConfig()*, perteneciente a la librería *sAPI*, deben asignarse dos parámetros de entrada, uno de ellos indica el módulo de la UART sobre la cual se desea trabajar, en este caso el parámetro es *UART\_USB*. El segundo parámetro que debe asignarse a dicha instrucción es la velocidad de transmisión en baudios. El valor de la velocidad previamente configurada sobre el programa debe ser configurado sobre el menú desplegable *BaudRate*, ubicado sobre el costado derecho del menú desplegable *Nombre*. Por ejemplo, si la velocidad configurada es igual a 9600 baudios, entonces sobre dicho menú debe seleccionarse 9600, como lo ilustra la Figura 34.
- Al realizar la depuración, el usuario debe seleccionar el botón *Abrir/Cerrar puerto COM*, el cual se encuentra sobre la derecha del menú desplegable *BaudRate*, como lo ilustra la Figura 33.
- Para efectuar transmisiones sobre el puerto COM seleccionado, el usuario debe ingresar el dato sobre el cuadro de texto asociado al mismo y apretar el botón asociado a dicho cuadro.

#### 4.8. Implementacion de ejemplo N°7:Comunicación Serial-Utilización Display LCD 16x2

El siguiente ejemplo utiliza los leds incorporados en la placa, de forma tal que el numero de leds encendidos corresponda al caracter enviado. Por ejemplo, si se envía el carácter '3', entonces se encienden 3 leds.

En este ejemplo también se utiliza un display LCD 16x2. Para el manejo del mismo fue necesario utilizar una librería no perteneciente a *sAPI*, la cual fue diseñada por Matias Ferraro[10].



- Realizar la generación de los archivos cabecera y fuente como se establece sobre la sección 4.2.
- Realizar la inclusión de los siguientes archivos cabecera y fuente para la utilización del Display LCD:

- Archivos Cabecera:

- *font8x8.h.h*
- *font8x16.h*
- *lcd.h*
- *puertos.h*

- Archivos Fuente:

- *lcd.c*
- *puertos.c*

- Sobre la Figura 35 se ilustra el diagrama esquemático para el conexionado del Display sobre la placa.

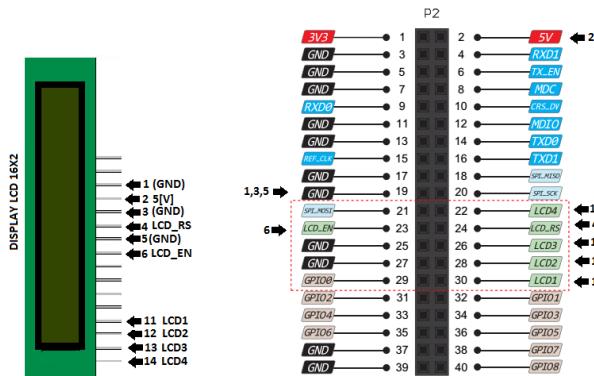


Figura 35: Diagrama esquemático de conexión display LCD 16x2

- Implementar la conexión que se indica sobre la Sección 4.4 para el Display 7 Segmentos.

## 5. Sistemas Operativos en Tiempo Real

Un Sistema Operativo es un conjunto de programas que ayuda al programador de aplicaciones a gestionar recursos de hardware disponible, entre ellos el tiempo del procesador y la memoria [11]. Una parte del OS se encarga de asignar el tiempo de ejecución a todos los programas que tiene cargados en base a un juego de reglas conocido de antemano. A dichos subprogramas se los denomina Tareas.

Con esto se logra aparentar que múltiples programas se están ejecutando simultáneamente, sin embargo solo pueden hacerlo de uno a la vez.

El encargado de realizar esta gestión es un componente del sistema operativo denominado Scheduler o Programador de Tareas. La función de éste es determinar qué tarea debe estar en ejecución a cada momento.



Ante la ocurrencia de ciertos eventos el Scheduler revisa si la tarea en ejecución debe reemplazarse por alguna otra tarea. A este reemplazo se lo denomina cambio de contexto de ejecución.

Debido a que el OS debe guardar el contexto completo de la tarea actual, y reemplazarlo por la tarea reentrant, debe reservarse un bloque de la memoria de datos para cada tarea. Lo cual limita el número de tareas que pueden ejecutarse de forma simultánea. Los cambios de contexto que se realizan cuando se reemplaza una tarea por otra no agregan trabajo al programador, de modo que al retornar a dicha tarea, el programador no observa ningún síntoma de haberla pausado alguna vez.

Un RTOS (sigla de *Real Time Operating System*) realiza la misma función que un OS común, sin embargo agrega herramientas para que los programas de aplicación puedan cumplir compromisos temporales definidos por el programador. De modo que se encuentran diseñados para la administración de varias tareas simultáneas con plazos de tiempo estrictos.

El RTOS ofrece funcionalidad para asegurar que una vez ocurrido un evento, la respuesta ocurra en un tiempo acotado. Es necesario destacar que esto no lo hace por si solo sino que brinda al programador herramientas para implementarlo de forma más simple.

## 5.1. Tareas

El bloque básico de software escrito sobre un RTOS es la Tarea, sobre la mayoría de las RTOS que se ofrecen en el mercado, una Tarea es una simple subrutina[8].

Una Tarea provee un contexto en el cual se ejecuten las funciones. Mientras que el Scheduler organiza la secuencia de ejecución de las tareas[12]. En algún punto del programa se produce el llamado a alguna función que produzca la ejecución de dicha tarea[8].

En el sistema operativo OSEK las Tareas deben ser definidas en la configuración, de manera que no es posible crear Tareas de forma dinámica (a diferencia de los Sistemas Operativos de escritorio)[13]. En el momento de la compilación se genera el código del sistema operativo y se definen la cantidad de Tareas y sus características. Además deben utilizarse macros para las definiciones y las funciones (a dichas funciones se las suele denominar Servicios del Sistema) que ofrece este Sistema Operativo, para ello, debe incluirse sobre el programa, el archivo cabecera *os.h*.

Este sistema operativo ofrece dos conceptos distintos de Tareas[12]:

- Tareas del tipo *BASIC*: Este tipo de Tareas solamente libera al procesador en los siguientes casos:
  - 1 Cuando terminan.
  - 2 Cuando son interrumpidas por acción del Scheduler, al realizar la activación de una Tarea que posea mayor prioridad (característica propia de los Sistemas Operativos del tipo *preemptive*).
  - 3 Ocurre una interrupción que causa que el procesador acuda a una subrutina de atención de interrupción.
- Tareas del tipo *EXTENDED*: Se distinguen de las Tareas extendidas por el hecho de que pueden utilizar una función propia de este OS, denominado *WaitEvent()*. Este tipo de Tareas permite liberar el procesador y ser relevado hacia Tareas de menor prioridad, sin la necesidad de que finalice la Tarea que se encuentra esperando. La ventaja de las Tareas extendidas es que pueden manejar coherentemente una Tarea sin importar que eventos de sincronización se encuentren activos



### 5.1.1. Modelo de Tareas Básicas

El modelo de las Tareas Basicas es similar al de las Tareas Extendidas, excepto que éstas no contienen un estado *Waiting*. A continuación se realiza una breve descripción de los estados definidos sobre este tipo de Tareas.

- **Running:** En este estado, la CPU es asignada a la Tarea, de modo que se ejecutan sus instrucciones. Solamente una sola Tarea (sin importar de que tipo sea) puede estar en este estado, en un instante determinado; mientras las demás Tareas pueden adoptar otros estados.
- **Ready:** Se cumplieron todos los pre-requisitos para que la Tarea adquiera el estado *Running*. El *Scheduler* decide cuál Tarea en estado *Ready* será desplazada al estado *Running*.
- **Suspended:** En este estado, la Tarea se encuentra en estado pasivo y puede ser activada.

La Figura 36 ilustra los conceptos explicados previamente.

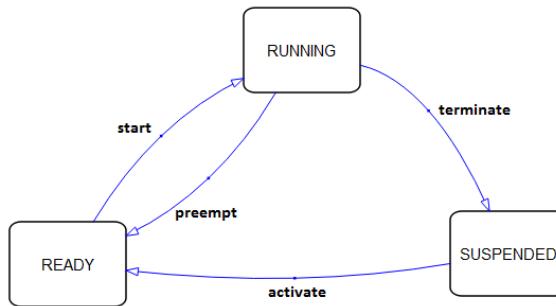


Figura 36: Modelo de las Tareas del tipo *BASIC*

El Cuadro 3 explica los diferentes eventos que provocan las transiciones entre los estados.

Transicion	Estado Actual	Nuevo Estado	Descripcion
activate	suspended	ready	Una nueva tarea se asigna al estado <i>Ready</i> por el OS.
start	ready	running	Una Tarea en estado <i>Ready</i> es seleccionada por el <i>Scheduler</i> para ser ejecutada.
preempt	running	ready	El <i>Scheduler</i> decide empezar otra Tarea. La Tarea que estaba en estado <i>Running</i> se establece en estado <i>Ready</i> .
terminate	running	suspended	La tarea en estado <i>Running</i> se establece en estado <i>Suspended</i> gracias a la acción de algún servicio del sistema.

Cuadro 3: Tabla de transiciones de las Tareas del tipo *Basic*.

### 5.1.2. Modelo de Tareas Extendidas

El modelo de las Tareas Extendidas posee cuatro estados:

- **Running:** En dicho estado, la CPU es asignada a la Tarea, de modo que se ejecuten las instrucciones que contiene la misma. Solamente una Tarea puede encontrarse en dicho estado en un instante determinado, mientras las otras Tareas pueden adoptar los demás estados.
- **Ready:** Se cumplieron todos los pre-requisitos necesarios para la transición al estado *Running*. El *Scheduler* decide cuál de las tareas en estado *Ready* será ejecutada a continuación.



- **Waiting:** La ejecución de una Tarea no puede continuar debido a que debe "esperar" a que un evento ocurra.
- **Suspended:** La Tarea se encuentra en modo pasivo hasta que se active.

La Figura 37 ilustra los conceptos explicados previamente.

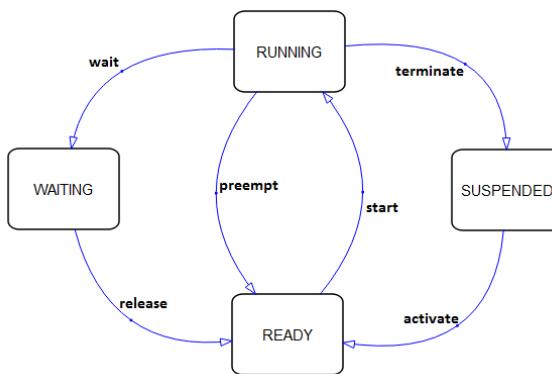


Figura 37: Modelo de las Tareas del tipo *EXTENDED*

El Cuadro 4 explica los diferentes eventos que provocan las transiciones entre los estados.

Transicion	Estado Actual	Nuevo Estado	Descripcion
activate	suspended	ready	Una nueva tarea se asigna al estado <i>Ready</i> por el OS.
start	ready	running	Una Tarea en estado <i>Ready</i> es seleccionada por el <i>Scheduler</i> para ser ejecutada.
wait	running	waiting	La tarea se establece en estado <i>Waiting</i> debido a la accion de un Servicio del Sistema.
release	waiting	ready	Al menos ocurrio un evento que esperaba la Tarea.
preempt	running	ready	El <i>Scheduler</i> decide iniciar otra Tarea. La Tarea que se encontraba en estado <i>Running</i> se establece en estado <i>Ready</i> .
terminate	running	suspended	La Tarea en estado <i>Running</i> se establece al estado <i>Suspended</i> gracias a un Servicio del Sistema.

Cuadro 4: Transiciones de las Tareas del tipo *Extended*.

#### Activación de Tareas:

La activación de una Tarea es realizada mediante el Servicio del Sistema *ActivateTask* o por *ChainTask*[12]. Luego de la activación de una Tarea, ésta se encuentra lista para ejecutarse desde su primer instrucción. En el caso que se produzcan múltiples activaciones de una Tarea del tipo *BASIC*, el Sistema Operativo OSEK almacena estas activaciones paralelas, siempre y cuando se haya definido sobre éste, una clase de conformidad.

#### Finalización de las tareas:

En el Sistema Operativo OSEK, una Tarea solo puede finalizarse por acción propia, mediante el Servicio del Sistema *TerminateTask*. Cabe destacar que el Servicio del Sistema *ChainTask* provee una forma de realizar la activación de una Tarea determinada, justo después de la finalización de la Tarea que se encuentra corriendo.

Esta estrictamente prohibido realizar una Tarea que no finalize con alguno de estos dos Servicios del Sistema, dado que puede ocasionar un comportamiento indefinido.

#### Prioridad de las tareas:

La asignación de prioridades sobre las Tareas se realiza a través de la definición de números enteros desde 0 hasta 255[13]. Mientras mayor es el valor del numero, mayor será la prioridad. Una Tarea que se encuentra corriendo, no puede ser interrumpida por una de menor o igual prioridad.



## 5.2. Scheduler

El algoritmo del Scheduler puede ser visto fundamentalmente como un coordinador de Tareas con las prioridades ya establecidas, en el cual se define un número máximo de activaciones de las mismas. El sistema operativo OSEK define tres políticas de *Scheduling*, las cuales se mencionan a continuación.

### Scheduling Full Preemptive:

Esta política de *Scheduling* implica que una Tarea que se encuentra en el estado *Running* puede ser interrumpida sobre cualquier instrucción, como consecuencia de la ocurrencia de alguna condición o evento; que ya se encuentre definido previamente sobre el OS. *Full Preemptive Scheduling* determinará que una Tarea en el estado *Running* sea establecido en el estado *Ready*, tan pronto como una Tarea de mayor prioridad se establezca en estado *Ready*.

El contexto de la Tarea será almacenado de forma que la Tarea que fue interrumpida pueda continuar sobre la instancia en donde fue interrumpida.

### Scheduling Non Preemptive:

Cuando el cambio de Tareas solo se produce mediante la ejecución de algún Servicio del Sistema (explicitados en los puntos de *Scheduling*).

### Puntos de Scheduling:

- Puntos de *Scheduling* en una tarea *NON-PREEMPTIVE*
  - Al llamar a la interfase *Schedule* cuando retorna E\_OK.
- Puntos de *Scheduling* en una tarea *PREEMPTIVE*
  - Al finalizar un llamado a la interfase *ActivateTask* que retorna E\_OK.
  - Al finalizar un llamado a la interfase *ChainTask* que no retorna.
  - Al finalizar un llamado a la interfase *TerminateTask* que no retorna.
  - Al finalizar un llamado a la interfase *ReleaseResource* que retorna E\_OK.
  - Al finalizar un llamado a la interfase *SetEvent* que no retorna.
  - Al expirar una alarma que activa una tarea.
  - Al terminar la ejecución de una ISR de categoría 2

## 5.3. Modos de Aplicación

Los Modos de Aplicación son designados sobre un Sistema Operativo OSEK, para trabajar sobre diferentes modos de operación. El número mínimo de modos de aplicación que puede soportar este Sistema Operativo es igual a 1. Un ejemplo que puede ilustrarse es la definición de dos modos de operación, uno para la operación normal del programa, y otro para la finalización de la misma.

## 5.4. Eventos

Los eventos son objetos manejados por el OS. Es posible considerarla de forma abstracta como un bit o un flag, que puede ser enmascarado. Dichos eventos pueden ser seteados o restablecidos utilizando Servicios del Sistema. Es importante destacar que solamente las Tareas del tipo *EXTENDED* pueden utilizar esta herramienta para permitir la sincronización de las mismas. El Servicio



del Sistema *WaitEvent* permite esperar al suceso del evento, para poder realizar las instrucciones que contenga dicha Tarea.

Las Tareas del tipo *EXTENDED* típicamente poseen su propio *stack* o pila que registra el suceso de dicho evento. Sin embargo, es muy importante destacar que este tipo de Tareas solo pueden ser activadas una sola vez.

Un evento individual es identificado por el propietario de dicho evento, y por su nombre (denominado Máscara según la especificación). Cuando una Tarea de tipo *EXTENDED* es activada, los eventos asignados a la misma son establecidos a 0 por el OS. El significado de un evento es definido por la aplicación; por ejemplo, puede significar que ha expirado un *Timer*, puede significar que se encuentra disponible un recurso, puede significar la recepción de un mensaje, etc.

Existen varias opciones disponibles para manipular los eventos, tanto para las Tareas que son propietarias de dicho Evento, como de las demás Tareas que no son propietarias del Evento, las cuales no necesariamente deben ser Tareas del tipo *EXTENDED*.

Todas las Tareas e Interrupciones del tipo *ISR2* pueden indicar el suceso de un Evento e informarlo a la Tarea propietaria del evento (del tipo *EXTENDED*) siempre y cuando no se encuentre en estado *SUSPENDED*. Sin embargo, solamente la Tarea propietaria de dicho Evento puede restablecer el flag del Evento.

Debe destacarse que cuando una Tarea tipo *EXTENDED*, se encuentra en estado *Waiting*, se establece en el estado *Ready* si al menos ocurre uno de los Eventos que dicha Tarea estaba esperando. Si una Tarea tipo *EXTENDED* se encuentra ejecutándose y trata de esperar un Evento que ya haya ocurrido, dicha Tarea se mantiene en el estado *Running*.

## 5.5. Clases de Conformidad

Con la finalidad de permitir que OSEK-OS pueda ser utilizado en una gran variedad de sistemas con diferentes capacidades y demandas (por ejemplo memoria, capacidad de procesamiento, etc.) es que OSEK-OS define 4 clases de conformidad (CC).

Las diferentes clases existen para poder comparar entre sistemas, agrupar las interfaces según la clase de conformidad y facilitar la certificación de conformidad. Las clases de conformidad se determinan según los siguientes atributos:

- Múltiple activación de tareas básicas.
- Tipos de tareas aceptadas: básicas y extendidas.
- Cantidad de tareas por prioridad.

De esta forma se definen las siguientes clases de conformidad:

- BCC1: que soporta únicamente tareas básicas con máximo una activación por tarea, y todas las tareas con prioridad diferente.
- BCC2: como BCC1 pero con más de una tarea por prioridad y las tareas soportan más de una activación.
- ECC1: como BCC1 pero con tareas extendidas.
- ECC2: como ECC1 pero con más de una tarea por prioridad y las tareas soportan más de una activación.



El OSEK-OS de la CIAA soporta todas las conformidades. Dependiendo de esta configuración el sistema decidirá si se implementa una clase BCC1 o ECC2. Por lo que para el usuario final del CIAA-Firmware es prácticamente irrelevante conocer las clases. El Firmware decide automáticamente y configura la clase de conformidad correcta según los requerimientos del usuario. La Figura 37 ilustra los conceptos presentados anteriormente.

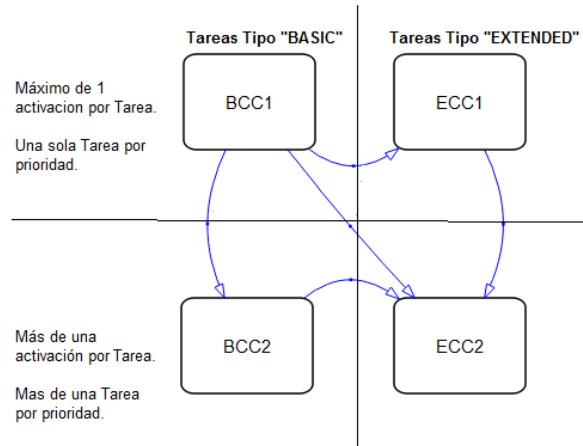


Figura 38: Clases de Conformidad.

## 5.6. Configuración y Generación

Al ser OSEK-OS un sistema estático es necesario configurarlo. La cantidad de tareas, que prioridad tienen las mismas, el tamaño de pila que utilizan, etc. Para ello OSEK-VDX definió otro estándar llamado OSEK Implementation Language comúnmente llamado „OIL”.

OIL es un lenguaje textual con una sintaxis similar al lenguaje C donde se indican las características del sistema operativo como ser: tareas, prioridades, pila, interrupciones etc. El siguiente ejemplo ilustra al usuario sobre la generación de dicho archivo.

Es necesario destacar que para la utilización de *RTOS OSEK* utilizando instrucciones en *LPCOpen versión 2.17*, se debe eliminar la definiciones de *TRUE/FALSE* del tipo *boolean* que realiza *LPCOpen* de forma que no genere conflictos con las definiciones propias del Sistema Operativo[14]. La Figura 39 ilustra una ventana resultante del error en el caso que no se haya realizado la eliminación de dicha definición.

Para solventar dicho error, una alternativa propuesta es la siguiente:

- Sobre el *Firmware* del proyecto, el usuario debe dirigirse hacia:

*Firmware/externals/drivers/cortexM4/lpc43xx/inc/lpc\_types.h*

- Comentar la instrucción *typedef enum FALSE =0, TRUE=!FALSE Bool*. La Figura 40 ilustra dicha instrucción comentada:

Archivos Makefile utilizados sobre cada ejemplo:

Otro aspecto que es necesario destacar es que para la implementación de los siguientes ejemplos se utiliza el archivo *Makefile* proporcionado en los ejemplos del *Firmware* de la *CIAA*.



```
=====
Compiling c file: PROYECTOS_FACU/blinking_mod/src/blinking_mod.c

arm-none-eabi-gcc -c -Wall -ggdb3 -fdata-sections -ffunction-sections -mcpu=cortex-m4 -mfpu=fpv4-sp-d16 -mfloat-abi=softfp -mthumb -DCORE_M4 -I`cygpath -w PROYECTOS_FACU/blinking_mod/include` -c ./modules/rtos/inc/os.h:55:8,
    from _PROYECTOS_FACU/blinking_mod/src/blinking_mod.c:72:
./modules/rtos/inc/Types.h:106:15: error: expected identifier before '(' token
#define FALSE ((boolean) 0)

./externals/drivers/cortexM4/lpc43xx/inc/lpc_types.h:50:15: note: in expansion of macro 'FALSE'
typedef enum {FALSE = 0, TRUE = !FALSE} Bool;
^
In file included from ./externals/drivers/cortexM4/lpc43xx/inc/cmsis.h:32:0,
    from ./externals/drivers/cortexM4/lpc43xx/inc/chip.h:33,
    from _PROYECTOS_FACU/blinking_mod/src/blinking_mod.c:82:
./externals/drivers/cortexM4/lpc43xx/inc/lpc_types.h:205:0: warning: "INLINE" redefined
#define INLINE inline
^
In file included from ./modules/rtos/inc/os.h:55:8,
    from _PROYECTOS_FACU/blinking_mod/src/blinking_mod.c:77:
./modules/rtos/inc/Types.h:97:0: note: this is the location of the previous definition
#define INLINE
^
make: *** [blinking_mod.o] Error 1
Makefile:491: fallo en las instrucciones para el objetivo 'blinking_mod.o'

18:27:08 Build Finished (took 4m:14s.567ms)
```

ERROR QUE ME IDENTIFICA EL CONFLICTO AL INTERPRETAR VARIABLE LOGICA

ERROR QUE ME IDENTIFICA UN CONFLICTO CON UNA DEFINICION PREVIAMENTE HECHA EN Types.h

Figura 39: Mensaje de consola de error por conflicto de definiciones de valores lógicos

```
48  * @brief Boolean Type definition
49  */
50 //typedef enum {FALSE = 0, TRUE = !FALSE} Bool;
51
```

Figura 40: Eliminación de definición de variables lógicas en *LPCOpen*

### 5.6.1. Implementacion de ejemplo N°1: Tareas en OSEK

Este ejemplo utiliza los leds de la placa y un Display 7 Segmentos, para implementar un contador de numeros decimales, a través de una sola Tarea del tipo *BASIC*.

- En primer lugar es necesario generar el archivo OIL para configurar el Sistema Operativo OSEK. Sobre el IDE de la placa, debe agregarse un nuevo directorio, realizando click derecho sobre *Firmware*, luego se selecciona la opción *New → Folder*.
- A continuación se abre una ventana que permite designar el nombre de dicho directorio, en el cual se ubica el proyecto que se implementa. El usuario debe recordar que para compilar dicho proyecto debe establecer la ruta que le corresponde sobre el archivo *Makefile*. Asimismo, deben agregarse los directorios *etc*, *mak*, *inc*, y *src*, sobre el directorio creado recientemente.
- Para generar el archivo cabecera, el usuario debe ubicarse sobre la carpeta *inc*, y luego oprimiendo click derecho, debe seleccionar la opción *New → Header File*. A continuación se abre una ventana cuyo aspecto es similar al de la Figura 41, en ella, se debe establecer el nombre del archivo cabecera. Al terminar la escritura de su nombre, debe escribirse la terminación *.h*, y oprimir el botón *Finish*, para concluir la generacion de dicho archivo.

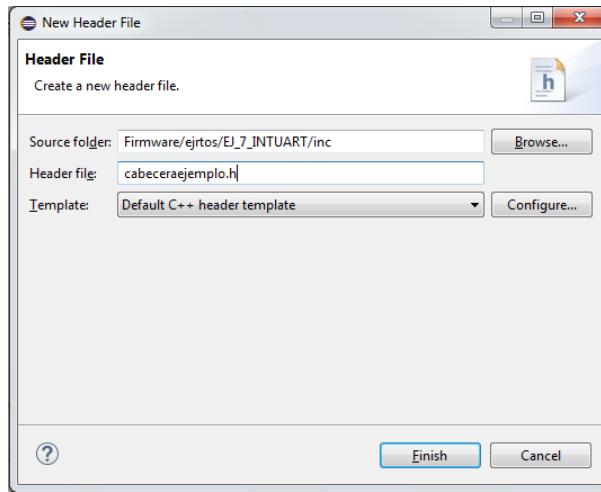


Figura 41: Generación de archivo cabecera para implementación de proyecto.

- Para generar el archivo fuente, el usuario debe ubicarse sobre la carpeta *inc*, y luego oprimiendo click derecho, debe seleccionar la opción *New → Source File*. A continuación se abre una ventana cuyo aspecto es similar al de la Figura 42, en ella, se debe establecer el nombre del archivo cabecera. Al terminar la escritura de su nombre, debe escribirse la terminación *.c*, y oprimir el botón *Finish*, para concluir la generacion de dicho archivo.

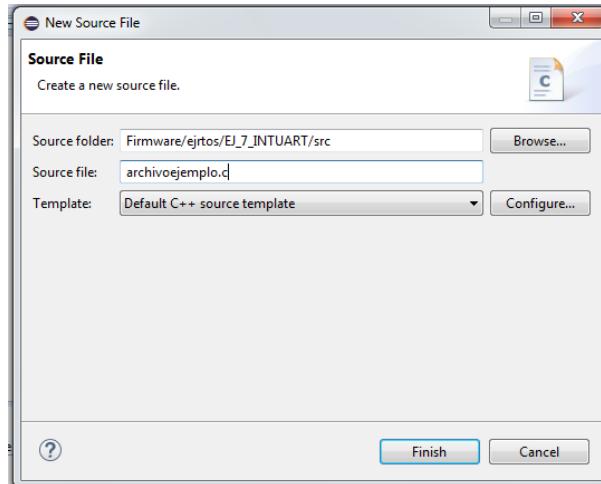


Figura 42: Generación de archivo fuente para implementación de proyecto.

- Debido a que no todos los módulos que integra la librería *sAPI* son compatibles con el Sistema Operativo, debemos especificar sobre el proyecto cuáles son los módulos a utilizar. En primer lugar debemos copiar y pegar aquellos archivos cabeceras de los módulos (compatibles) sobre la carpeta *inc*. A continuación se detalla una lista (respecto la version 0.3.0) de éstos modulos:
  - sAPI\_AnalogIO.h*
  - sAPI\_Board.h*
  - sAPI\_DataTypes.h*



- *sAPI\_DigitalIO.h*
- *sAPI\_Hmc5883l.h*
- *sAPI\_I2c.h*
- *sAPI\_PeripheralMap.h*
- *sAPI\_Rtc.h*
- *sAPI.h*

En particular el módulo *sAPI\_Uart* puede ser compatible en el caso que se borren los Handler de interrupción. Para ello, el usuario puede comentar la sección del código (ilustrado sobre la Figura 43) sobre *sAPI\_Uart.h*.

```

64 /*=====
65 =====*[ISR external functions declaration]=====
66 */
67 /* 0x28 0x000000A0 - Handler for ISR UART0 (IRQ 24) */
68 //void UART0_IRQHandler(void);
69 /* 0x2a 0x000000A8 - Handler for ISR UART2 (IRQ 26) */
70 //void UART2_IRQHandler(void);
71 /* 0x2b 0x000000AC - Handler for ISR UART3 (IRQ 27) */
72 //void UART3_IRQHandler(void);

```

Figura 43: Modificación para compatibilidad de módulo sAPI\_Uart.Parte 1

Sobre el archivo *sAPI\_Uart.c*, debe realizar las modificaciones que se muestran en las Figuras 44 y 45.

```

67     //Enable UART Rx Interrupt
68     Chip_UART_IntEnable(UART_USB_LPC,UART_IER_RBRINT );
69     // Enable UART line status interrupt
70     Chip_UART_IntEnable(UART_USB_LPC,UART_IER_RLSINT );
71     // NVIC_SetPriority(USART2_IRQn, 6);
72     // Enable Interrupt for USART channel
73     // NVIC_EnableIRQ(USART2_IRQn);

```

Figura 44: Modificación para compatibilidad de módulo sAPI\_Uart.Parte 2

```

136 /*=====
137 =====*[ISR external functions definition]=====
138 //__attribute__ ((section(".after_vectors")))
139
140 /* 0x28 0x000000A0 - Handler for ISR UART0 (IRQ 24) */
141 //void UART0_IRQHandler(void){
142 //}
143
144 /* 0x2a 0x000000A8 - Handler for ISR UART2 (IRQ 26) */
145 //void UART2_IRQHandler(void){
146 //}
147
148 /* 0x2b 0x000000AC - Handler for ISR UART3 (IRQ 27) */
149 //void UART3_IRQHandler(void){
150     //if (Chip_UART_ReadLineStatus(UART_232) & UART_LSR_RDR) {
151     //    receivedByte = Chip_UART_ReadByte(UART_232);
152     //}
153 }

```

Figura 45: Modificación para compatibilidad de módulo sAPI\_Uart.Parte 3

- Para finalizar la inclusión de dichos módulos, el usuario debe eliminar las inclusiones realizadas sobre el archivo cabecera *sAPI.h*. La Figura 46 ejemplifica la inclusión de el módulo *sAPI\_DigitalIO.h*.



```

41 #include "sAPI_DataTypes.h"
42 //#include "sAPI_IsrVector.h"
43
44 #include "sAPI_Board.h"
45 #include "sAPI_PeripheralMap.h"
46 #include "sAPI_DigitalIO.h"

```

Figura 46: Ejemplo de inclusión de módulo de entradas y salidas digitales.

Debe destacarse que es necesaria la inclusión de las librerías *sAPIBoard.h*, *sAPIDataTypes.h* y *sAPIPeripheralMap.h*. En primer lugar para la definición de la placa a utilizar, en segundo lugar los tipos de datos definidos por la librería, y por último para la inclusión de las definiciones de los terminales para los periféricos.

- A través del *Bloc de Notas* o a través de cualquier editor de texto, debe almacenarse el archivo de configuración de tipo *OIL* sobre la carpeta *etc* previamente creada sobre el directorio del proyecto, en dicho archivo, debe iniciar la configuración a través de la sintaxis *OSEK OSEK* y sobre los símbolos { y }, se determinarán los restantes aspectos que tendrá el OS. La Figura 47 ilustra lo mencionado anteriormente.

```

OSEK OSEK {
}

```

Figura 47: Inicio de Archivo de Configuración.

- Luego debe definirse mediante la sintaxis *OS*, el nombre del sistema operativo (el cual es arbitrario, dado que se ha especificado en la instrucción anterior que se implementa un sistema operativo *OSEK*), la Figura 48 ilustra el concepto explicado anteriormente.

```

OSEK OSEK{
    OS EJEMPLO1{
    }
}

```

Figura 48: Archivo de Configuración. Comienzo de especificación

- A continuación debe definirse las siguientes configuraciones para el OS del ejemplo:
  - El nivel de chequeo de errores que se pueden producir. Existen dos niveles que pueden determinarse:
    - *STANDARD*: Implica un chequeo mínimo de errores, se encuentra pensado para el empleo en sistemas que deben producirse en masa. La instrucción utilizada es *OS = STANDARD*.



- *EXTENDED*: Implica un chequeo máximo de errores y *debug hooks*. La instrucción utilizada es *OS = EXTENDED*.
- Funciones implementadas por el usuario que el OS llamará en circunstancias específicas, tales como:
  - *Startuphoonk*: es llamada durante la inicialización del sistema operativo, antes de ser completada. En caso de utilizarla se debe escribir la instrucción *STARTUPHOOK = TRUE*; en caso contrario se debe asignar *FALSE*.
  - *Shutdownhook*: es llamada al finalizar el apagado del sistema operativo. En caso de utilizarla se debe escribir la instrucción *SHUTDOWNHOOK = TRUE*; en caso contrario se debe asignar *FALSE*.
  - *Pretaskhook*: es llamada antes de proceder a ejecutar una tarea. En caso de utilizarla se debe escribir la instrucción *PRETASKHOOK = TRUE*; en caso contrario se debe asignar *FALSE*.
  - *Posttaskhook*: es llamada al finalizar la ejecución de una tarea. En caso de utilizarla se debe escribir la instrucción *POSTTASKHOOK = TRUE*; en caso contrario se debe asignar *FALSE*.
  - *Errorhook*: es llamada en caso de que alguna de las interfaces del sistema operativo retorne un valor distinto a *E\_OK*. En caso de utilizarla se debe escribir la instrucción *ERRORHOOK = TRUE*; en caso contrario se debe asignar *FALSE*.
- Uso del *Scheduler* como recurso.
- Uso del mapa de memoria.

La Figura 49 ilustra un OS configurado de forma que el nivel de chequeo de errores sea el mínimo, no se determine el uso del *Scheduler*, y no se utilice una cantidad específica de memoria, por último solamente se define una función en caso de un error producido en el OS.

```
OSEK OSEK{
    OS EJEMPLO1{
        STATUS=STANDARD;
        ERRORHOOK=TRUE;
        PRETASKHOOK=FALSE;
        POSTTASKHOOK=FALSE;
        STARTUPHOOK=FALSE;
        SHUTDOWNHOOK=FALSE;
        USERESSCHEDULER=FALSE;
        MEMMAP=FALSE;
    }
}
```

Figura 49: Ejemplo de especificación de OS sobre archivo OIL

- A continuación se define una Tarea que efectua la configuración inicial de la placa, y de los puertos GPIO utilizados para el Display 7 Segmentos. La instrucción que permite la definición de una Tarea sobre el archivo de configuracion *OIL* es *TASK*, la Figura 50 ilustra



el proceso llevado a cabo para dicho proceso. El usuario debe tener en cuenta que se lleva a cabo este proceso para cada uno de los proyectos implementados sobre este Tutorial.

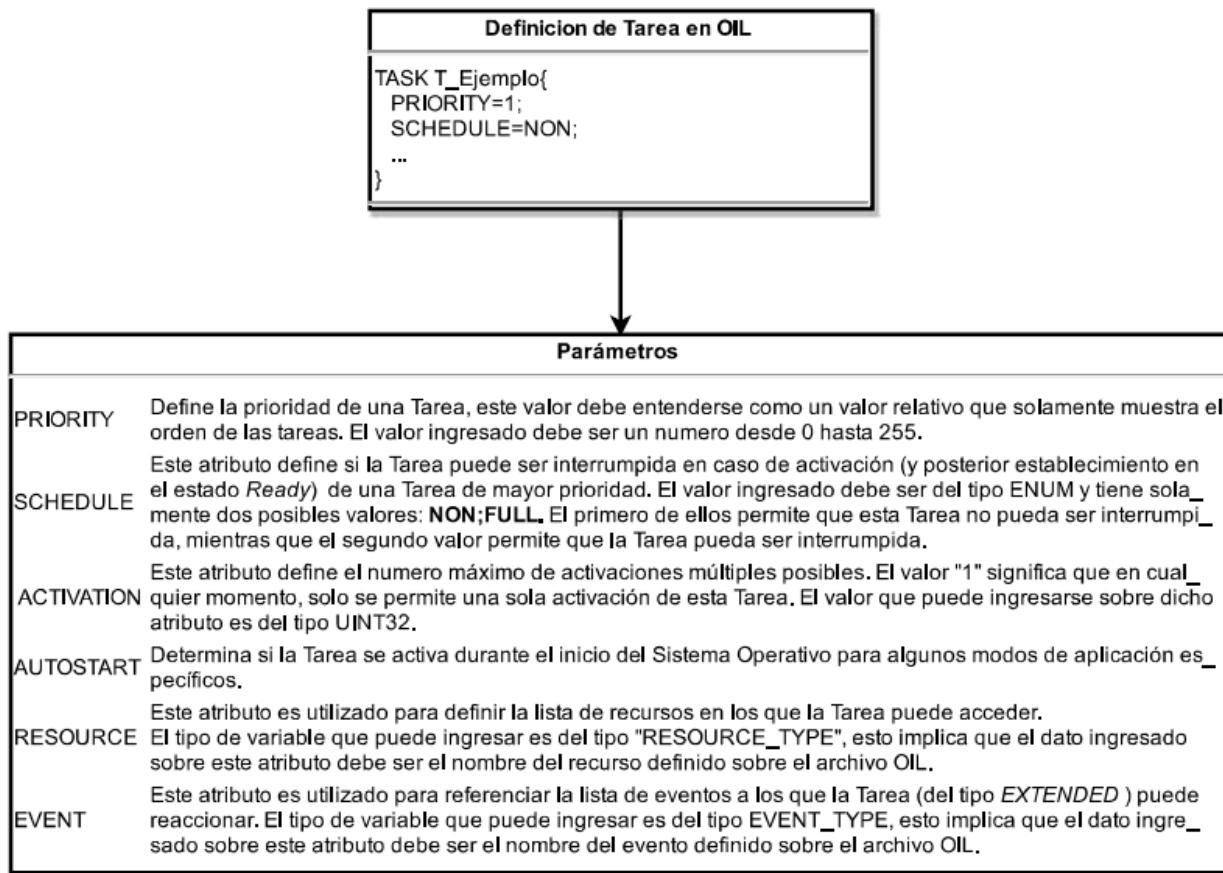


Figura 50: Definición de Tarea sobre archivo de configuración

- En este caso en particular, la tarea que se desea definir en primer lugar, necesita activarse en el momento de inicio del programa. Teniendo en cuenta los atributos que se explican sobre la figura 50 ,se determina que el atributo *AUTOSTART* adquiera el valor *TRUE*. Debe considerarse que no se prevee que dicha Tarea vuelva a ejecutarse durante el desarrollo del programa, por ese motivo, se asigna una prioridad baja y su tipo designado es *BASIC*.
- La siguiente Tarea realiza la muestra de los dígitos sobre los leds incorporados en la placa (forma binaria) y sobre el Display 7 Segmentos. Es importante destacar que esta Tarea se ejecuta constantemente, debido a que no existe otra Tarea de mayor prioridad que se active e interrumpta su ejecución. Para permitir esta ejecución constante es necesario la instrucción propia del SO (también denominada Servicio del Sistema) *ChainTask()* para permitir que esta Tarea se reactive a si misma.La Figura 51 ilustra las definiciones de las dos Tareas descriptas anteriormente.



```

TASK Inicializar{
    PRIORITY=1;
    ACTIVATION=1;
    STACK=256;
    TYPE=BASIC;
    SCHEDULE=NON;
    AUTOSTART=TRUE{
        APPMODE=AppMode1;
    }
    RESOURCE=POSIXR;
    EVENT = POSIXE;
}

TASK DIGITOS{
    PRIORITY=10;
    ACTIVATION=1;
    STACK=256;
    TYPE=BASIC;
    SCHEDULE=NON;
    RESOURCE=POSIXR;
    EVENT = POSIXE;
}

```

Figura 51: Ilustración de definición de Tareas de Ejemplo N°1

### 5.6.2. Implementacion de ejemplo N°2: Múltiples Tareas en OSEK

El siguiente ejemplo pretende demostrar como pueden implementarse varias Tareas del Tipo *BASIC* a través de las instrucciones del Sistema Operativo *TerminateTask()* y *ChainTask()*.

En este caso, se desea implementar un programa que permita encender los leds incorporados en la placa de izquierda a derecha, oprimiendo *TEC1*, *TEC2*, *TEC3*, *TEC4* en el mismo orden.

Se realiza la definición de una Tarea que establece la configuración inicial de la placa, de forma similar al ejemplo anterior, sin embargo, dicha Tarea produce la activación de otra denominada *LEDUNO*, la cual permite el encendido del LED1 (es importante aclarar en esta instancia para evitar posibles problemas al usuario, que para la implementación de otros programas, es recomendable que al definir nombres para las Tareas, éstos sean distintos a definiciones o nombres de funciones de bibliotecas incorporadas ). La tarea *LEDB* se reactiva automáticamente a través de

ESTRUCTURA GENERAL: EJEMPLO 2
<pre> nt main(void)  StartOS(AppMode1); return 0;  TASK (LEDB){     if (digitalRead(TEC1)==ON){         ...         ActivateTask(LED1);         TerminateTask();     }     ChainTask(LEDB);      TASK (LED1){         if (digitalRead(TEC2)==ON){             ...             ActivateTask(LED2);             TerminateTask();         }         ChainTask(LED1);     } } </pre>

Figura 52: Estructura general de archivo fuente de Ejemplo N°2



la instrucción del Sistema Operativo *ChainTask()*, salvo que se oprima la tecla *TEC1*, en cuyo caso el código procede a encender el *LEDB*, y posteriormente activa la Tarea siguiente y finaliza la que se encuentra ejecutando, a través de las instrucciones *ActivateTask()* y *TerminateTask()*. Las siguientes Tareas poseen una estructura similar.

Es importante destacar que las Tareas **LED1,LED2,LED3**, no son interrumpidas en el momento de la activación de la posterior Tarea que le corresponde, debido a que en el archivo de configuración se establece el valor *NON* sobre el atributo *SCHEDULE*(Es posible consultar la Figura 50). La Figura 52 ilustra la estructura de este ejemplo.

### 5.6.3. Implementacion de ejemplo N°3: Eventos en OSEK

Este ejemplo pretende brindar al usuario una introducción al manejo de las Tareas del tipo *EXTENDED* a través de los Eventos. Se pretende encender el *LED1* de la placa en el instante en que el usuario oprime la tecla *TEC1*, a continuación, se desea encender el *LED2* de la placa oprimiendo la tecla *TEC2*.

```
Estructura condicional ➔ while(digitalRead(TEC1)==ON){
    que "detiene" flujo de
    programa mientras TEC1
}
no se oprime.
```

Chaintask(LED1); ➔ Terminación y reactivación de Tarea LED\_1

Figura 53: Transición de las Tareas mediante Eventos en Ejemplo N°3

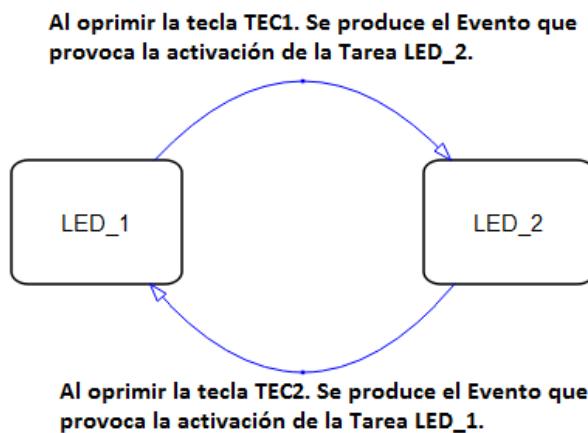


Figura 54: Transición de las Tareas mediante Eventos en Ejemplo N°3

Se realiza la definición de dos Tareas, denominadas *LED\_1* y *LED\_2*. La primera de ellas produce el encendido del *LED1* en caso de que se oprima *TEC1*, a continuación realiza la activación de la Tarea *LED\_2*. En el instante en que se aprete la tecla *TEC2*, se produce el encendido del *LED2*, a continuación realiza la activación de la Tarea *LED\_1*. Para efectuar este ejemplo se utilizó la estructura condicional *while* para detener el flujo de la Tarea *LED\_1* y *LED\_2*, mientras se no se opriman las teclas *TEC1* o *TEC2* según corresponda. La Figura 53 ilustra el concepto explicado anteriormente.



```

TASK(LED_1){
    while(digitalRead(TEC1)==ON){
        ChainTask(LED_1);
    }
    digitalWrite(LED1,ON);
    digitalWrite(LED2,OFF);
    ActivateTask(LED_2); // Activación de Tarea LED_2
    SetEvent(LED_2,tecla2); // Generación de Evento "tecla2" perteneciente a Tarea LED_2
    TerminateTask();
}

```

↓  
Al oprimir TEC1 se permite la continuación del flujo del programa.

Figura 55: Transición de las Tareas mediante Eventos en Ejemplo N°3

La Figura 54 ilustra la transición que se produce entre las Tareas a través de los Eventos. La Figura 55 ilustra la transición entre las Tareas *LED\_1* y *LED\_2*, cuando se oprime alguna de las teclas correspondientes.

## 5.7. Alarmas

Las alarmas son herramientas que proporciona el Sistema Operativo *OSEK* para la ejecución de una acción luego de determinado tiempo [13]. Éstas pueden realizar tres tipos de acciones:

- Activar una tarea.
- Establecer un evento de una tarea.
- Llamar una callback (retrollamada) en C.

Para la implementación de las alarmas, ya sean que se utiliza una ó más alarmas el sistema operativo necesita un contador de hardware. Con un contador es posible configurar la cantidad de alarmas que sean necesarias.

Luego de que dicha alarma expire, solo puede realizar una sola acción, la cual es definida en el archivo de configuración *OIL* a través de la variable *ACTION*. A continuación se detalla los posibles parámetros que pueden ingresarse:

- *ACTIVATETASK* define la activación de una Tarea cuando dicha alarma expire.
- *SETEVENT* Permite establecer un Evento (perteneciente a una Tarea) cuando dicha alarma expire.
- *ALARMCALLBACK* define el nombre de la rutina *callback* cuando la alarma expira.

### 5.7.1. Implementación de ejemplo N°4: Eventos y Alarmas en OSEK

Este ejemplo permite introducir al usuario en la utilización de alarmas para la temporización de acciones. Se pretende elaborar dos secuencias de luces a través de los leds instalados en la placa, la primera de ellas se activa al oprimir la tecla *TEC1*, mientras que la segunda se activa al oprimir *TEC2*.

Sobre el archivo de configuración se definen dos Tareas en particular: *SECUENCIA1* y *SECUENCIA2*. Dichas Tareas se activan y ejecutan sus correspondientes códigos a través del establecimiento del Evento *primersecuencia* o *segsecuencia*.

Durante la ejecución de las Tareas *SECUENCIA1* o *SECUENCIA2* se identifica el Evento ocurrido, a través de las instrucciones *WaitEvent()* y *GetEvent()*. Definiendo como tipo de variable para la máscara de los Eventos de las Tareas, como *Eventos1* y *Eventos2* respectivamente.



En caso de que haya ocurrido el Evento *primersecuencia* o *segsecuencia*, los códigos de las Tareas proceden a activar las Alarmas *Set\_Event.tiempo1* y *Set\_Event.tiempo2* respectivamente, para lograr la temporización deseada en las secuencias. La Figura 56 ilustra los conceptos explicados anteriormente.

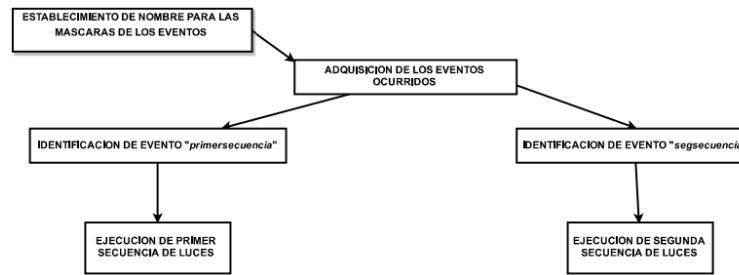


Figura 56: Identificacion de Evento en Ejemplo N°4

### 5.7.2. Implementacion de ejemplo N°5: Alarmas en OSEK

En este ejemplo se utilizan las Alarmas para permitir la activación de una Tarea en forma periódica. En particular, esta Alarma activa una Tarea la cual permite la lectura analógica sobre el canal 1 del *ADC0* de la placa.

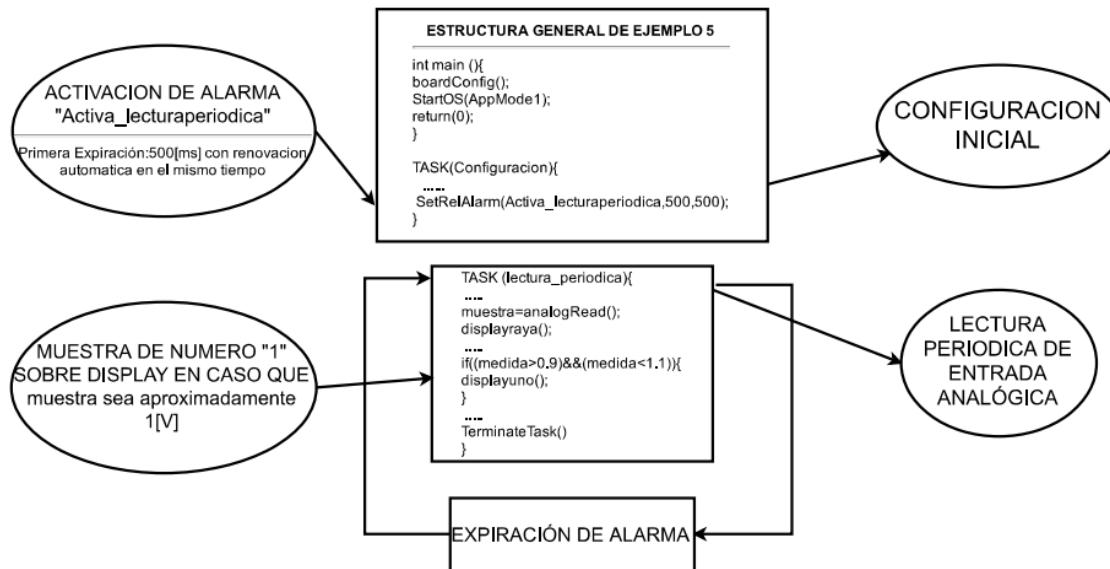


Figura 57: Estructura y funcionamiento de archivo fuente en Ejemplo N°5

Se realiza la definición de la Tarea *lectura\_periodica*, la cuál se activa cada 500[ms], gracias a la previa activación de la Alarma *Activa\_lecturaperiodica* mediante la instrucción *SetRelAlarm()*. En



caso que la lectura sea igual a 1[V], 2[V], 3[V], 4[V] o 5[V]; se efectúa la muestra sobre el Display 7 Segmentos del valor correspondiente. La Figura 57 ilustra la estructura del archivo fuente de este ejemplo.

## 5.8. Interrupciones

Las funciones para realizar la atención de las distintas interrupciones sobre el programa se pueden subdividir en 2 categorías[13]:

- Interrupciones del tipo **ISR1**: Éste tipo de interrupciones son transparentes al Sistema Operativo *OSEK* y por ello no pueden utilizar casi ninguna interfaz del Sistema Operativo.
- Interrupciones del tipo **ISR2**: Éste tipo de interrupciones tienen una mínima intervención del Sistema Operativo, y por ello pueden utilizar algunas interfaces del Sistema Operativo. Puede consultarse como referencia el *Introducción a OSEK-OS: El Sistema Operativo del CIAA-Firmware* sobre la Tabla A1 de la página 51 y 52 para consultar las instrucciones del Sistema Operativo que pueden utilizarse según el contexto.

Sin importar si se trata de una tarea *PREEMPTIVE* o *NON PREEMPTIVE* la misma va a ser interrumpida si se recibe una interrupción. En caso de querer evitar esto el sistema operativo provee al usuario las siguientes interfaces para desactivar las interrupciones:

- *DisableAllInterrupts*
- *EnableAllInterrupts*
- *SuspendAllInterrupts*
- *ResumeAllInterrupts*
- *SuspendOSInterrupts*
- *ResumeOSInterrupts*

### 5.8.1. Implementacion de ejemplo N°6: Interrupciones en OSEK

En este ejemplo se busca ilustrar al usuario a la utilización de Interrupciones en *OSEK*. En particular, este ejemplo implementa una interrupción externa, a través de la utilización de una librería creada por Pablo Ridolfi[15]. Para la inclusión de dicha librería, se debe proceder de forma similar que la explicada en la sección 5.6.1.

```
ISR GPIOINTHandler0 {
    INTERRUPT = GPIO0;
    CATEGORY = 2;
    PRIORITY = 0;
}
```

Figura 58: Definición de Interrupción sobre Ejemplo N°6



Sobre la Tarea *Iniciarizar* se realiza la habilitación de la interrupción a través del GPIO0 (el cual se encuentra configurado sobre el pin perteneciente a la tecla *TEC4* incorporada a la placa) mediante la instrucción (previamente definida sobre el archivo cabecera que pertenece a la librería) *ciaoGpioIntEnable(0, appIrqHandler)*.

Sobre el primer parámetro de dicha instrucción se especifica el canal utilizado. En este caso la librería utiliza el canal 0, el cual indica que se trabaja sobre el Registro de selección de Pines de Interrupción N°0 (denominado registro *PINTSEL 0*).

Sobre el segundo parámetro utilizado se especifica el *Handler* de interrupción utilizado, en este caso la librería la denomina *appIrqHandler*.

Sobre el archivo de configuración del Sistema Operativo se define una Interrupción denominada *GPIOINTHandler0*. La Figura 58 ilustra dicha definición sobre el archivo *OIL* y la interpretación de los parámetros establecidos sobre dicha Interrupción.

Es importante aclarar en esta instancia que es conveniente definir para los nombres de las interrupciones, nombres distintos a las definiciones de interrupciones previamente declaradas sobre el Firmware, dado que se genera conflictos. Es posible ver que sobre el parámetro *CATEGORY* se debe establecer el tipo de Interrupción a implementar, en caso de que la interrupción sea del tipo *ISR1* debe asignarse el valor 1 sobre este atributo, en cambio, si dicha Interrupción es del tipo *ISR2* debe asignarse el valor 2. Sobre el parámetro *INTERRUPT* debe indicarse el nombre de la

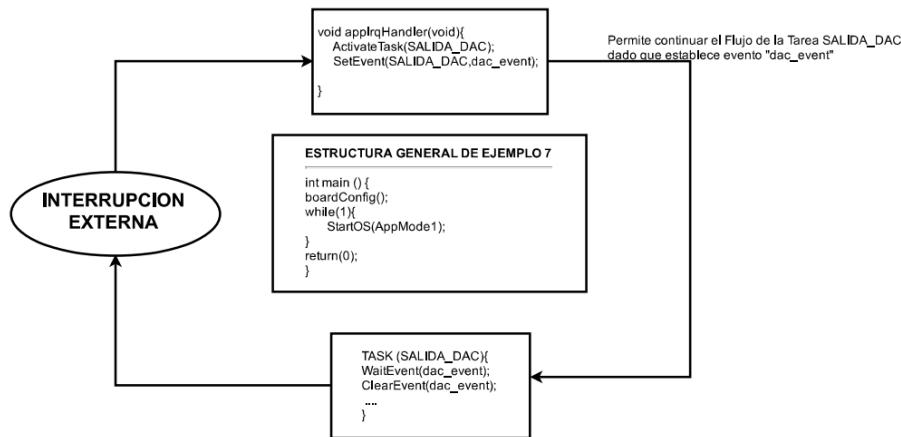


Figura 59: Estructura de archivo fuente en Ejemplo N°6

interrupción. Es importante tener en cuenta que dicho nombre es específico de cada implementación de Sistema Operativo, en este caso en particular, el nombre de la interrupción asociada tiene como nombre *GPIO0*.

Puede observarse los nombres de las interrupciones asociadas en esta implementación de *OSEK*, sobre el archivo *Os\_Internal\_Arch\_Cfg.c.php* ubicado sobre la siguiente ruta:

*Firmware/modules/rtos/gen/src/CortexM4*.

Cuando la interrupción externa se produce el establecimiento de una tensión de salida analógica a través del uso del DAC, la magnitud de la tensión es igual a 1[V], 2[V] o 3.3[V], según se aprete una, dos o tres veces la tecla asignada a la interrupción externa (*TEC4*). La Figura 59 ilustra el funcionamiento y la estructura del archivo fuente del ejemplo.



```
ISR_UART2{
    CATEGORY=2;
    INTERRUPT=UART2;
    PRIORITY=0;
}
```

Figura 60: Definición de Interrupción sobre Ejemplo N°7

### 5.8.2. Implementacion de ejemplo N°7: Interrupciones en OSEK

En este ejemplo se implementa comunicación serial mediante la interrupción sobre el módulo UART del microprocesador. Para la inclusión del módulo *sAPI\_Uart* debe realizarse los procedimientos explicados en la sección 5.6.1.

En particular este ejemplo realiza la recepción a través de una Interrupción producida por la UART, de manera tal que provoque un Evento denominado *palabra*. A continuación la Tarea asociada a dicho Evento identifica si el carácter recibido es igual a "L", en cuyo caso, se establece el envío de la palabra "*Recibido caracter deseado*". Sobre el archivo de configuración se define la interrupción *UART2*, la cual se ilustra sobre la Figura 60.

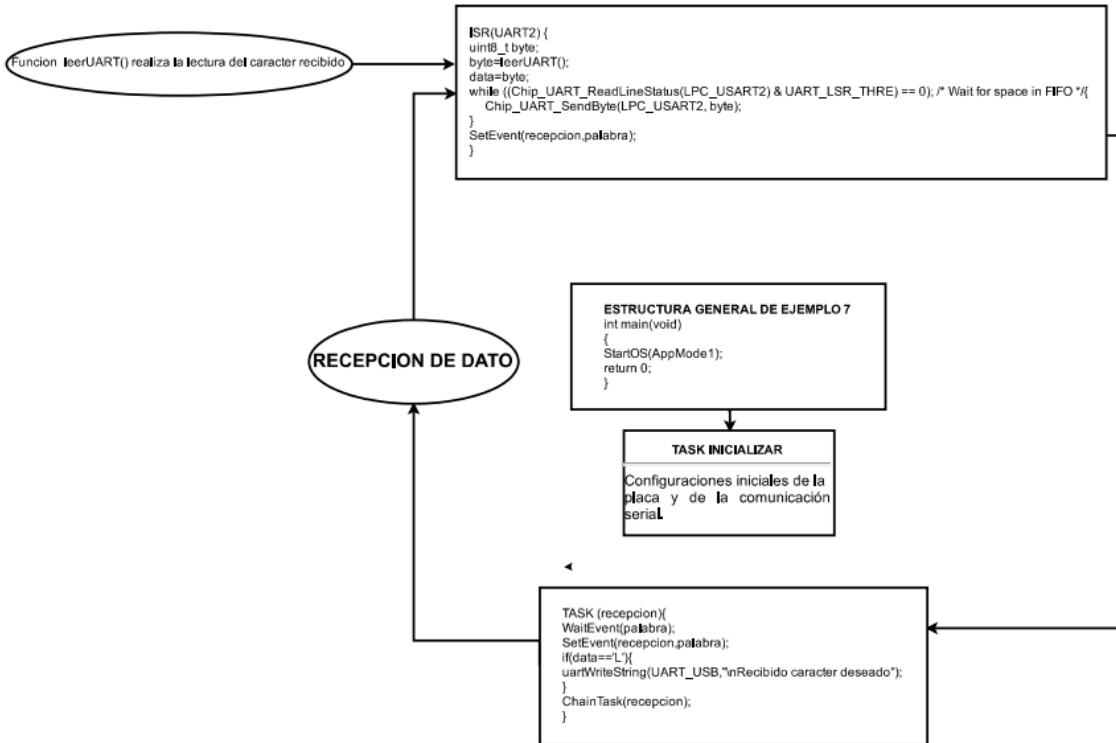


Figura 61: Estructura y funcionamiento de archivo fuente sobre Ejemplo N°7

Es importante destacar que el nombre utilizado en la interrupción puede definirse arbitrariamente, mientras que el nombre de la interrupción establecido sobre el parámetro *INTERRUPT* es



*UART2*, el cuál es el nombre que esta implementación de *OSEK* ha utilizado. La Figura 61 ilustra la estructura y el funcionamiento del ejemplo.

### 5.8.3. Implementacion de ejemplo N°8: Interrupciones en OSEK

El siguiente ejemplo permite mostrar como puede implementarse la interrupción por recepción de datos sobre el módulo UART, de manera tal que el usuario pueda interactuar con el programa, a través del envío de datos sobre el puerto serial con el programa *Octoplus*. Para efectuar la implementación de este ejemplo, el usuario puede recurrir a la sección 4.7 para repasar el modo de utilización de dicho programa.

Sobre el archivo de configuración se efectúa la definición de una Tarea denominada *BLINKING*, en la cuál se realiza la identificación del evento establecido y su posterior acción de manera similar a la que se aplica sobre el ejemplo N°4.

Dicha Tarea realiza un blinkeo del *LED1* de la placa, igual a un numero de veces determinado por un solo carácter recibido sobre el puerto serial. De modo que el *LED1* puede ejecutar el blinkeo entre 1 y 9 veces.

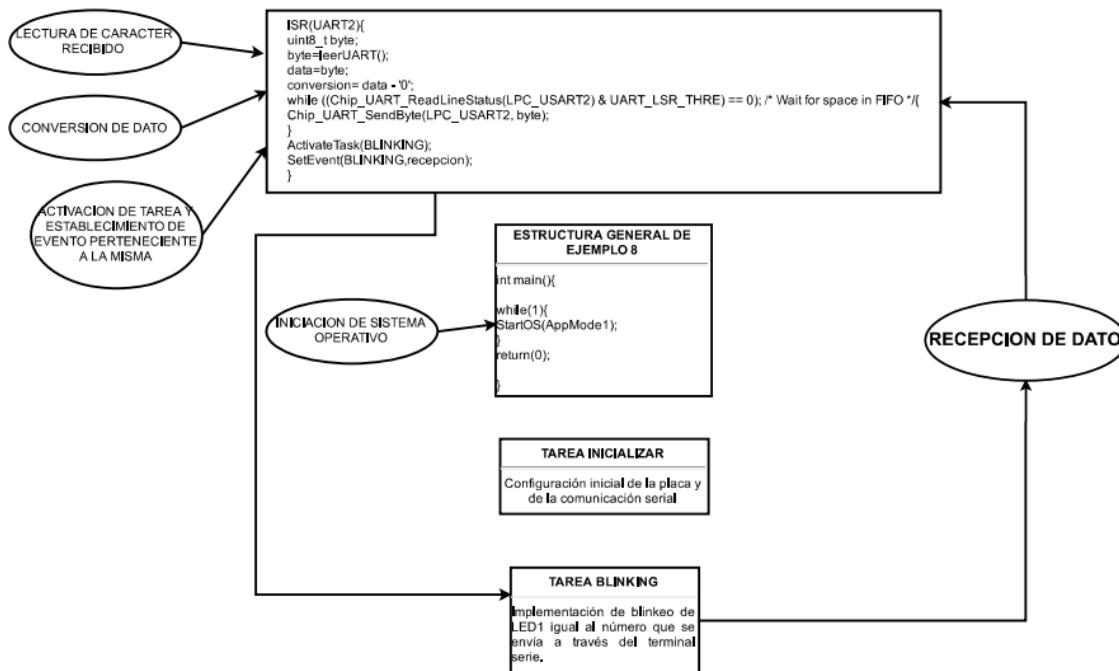


Figura 62: Estructura y funcionamiento de archivo fuente sobre Ejemplo N°8

Para efectuar la recepción del carácter se implementa una interrupción del tipo *ISR2* denominada *UART2* en la cual se efectúa la lectura del carácter recibido a través de la función *leerUART()*. A continuación, se produce el pasaje hacia el número entero asociado a dicha letra recibida. Finalmente se realiza la activación de la Tarea *BLINKING* y el establecimiento del evento *repcion*, el cual esta asociado a dicha Tarea. Es necesario destacar la utilización de la variable global *conversion* para el control de la cantidad de veces que debe realizarse el blinking. La Figura ilustra esquemáticamente la estructura y el comportamiento del archivo fuente del ejemplo.



## 6. Implementación de ejemplo N°9: Inclusión de librería LCD y formación de librerías propias para trabajar

El siguiente ejemplo tiene como objetivo ilustrar al usuario como puede implementarse librerías propias para la agilización de proyectos y proveer una forma de clarificar la visualización del programa.

En este ejemplo se realiza la lectura analógica de una tensión de entrada sobre el canal 1 del puerto ADC0 de la placa. El usuario puede recurrir a la Figura 29 de la sección 4.5 para repasar el diagrama de conexión.

Para comenzar la implementación de una librería propia se procede de forma similar a la que se estableció para la generación de un archivo cabecera y un archivo fuente sobre la sección 5.6.1. En este caso en particular se implementa una librería propia que tiene como propósito configurar la placa, las teclas y los leds incorporados; facilitar el encendido de un solo Led en particular (a través de una sola instrucción creada en la librería); facilitar la conversión de la lectura analógica realizada por el ADC; simplificar la inicialización del Display LCD 16x2 mediante las instrucciones propias de la librería de Matias Ferraro [10]; y generar instrucciones de utilización propia en particular.

Debe tenerse en cuenta que al utilizar instrucciones propias de la librería *sAPI* y la del Display LCD, sobre el archivo cabecera de la biblioteca propia, debe incluirse los archivos cabecera de las librerías listadas anteriormente. La Figura 63 ilustra el concepto explicado.

```
#include "sAPI.h" // Inclusión de librería sAPI
#include "lcd.h" // Inclusión de librería de Display LCD 16x2
```

Figura 63: Inclusión de librerías *sAPI* y librería de Display LCD sobre librería propia

Este ejemplo en particular realiza la lectura analógica e imprime sobre la pantalla del display el resultado de dicha lectura, al oprimir la tecla *TEC1*. La Figura ilustra el funcionamiento de dicho Ejemplo

## Referencias

- [1] [http://proyecto-ciaa.com.ar/devwiki/doku.php?id=proyecto:origen\\_ciaa](http://proyecto-ciaa.com.ar/devwiki/doku.php?id=proyecto:origen_ciaa)  
Fecha: 15/10/16 10:55 am.
- [2] *UM10503 LPC43xx/LPC43Sxx ARM Cortex-M4/M0 multi-core microcontroller* Pags: 12-16.
- [3] <http://proyecto-ciaa.com.ar/devwiki/doku.php?id=desarrollo:edu-ciaa:edu-ciaa-nxp>  
Fecha: 15/10/16 10:55 am.
- [4] [http://proyecto-ciaa.com.ar/devwiki/doku.php?id=desarrollo:firmware:instalacion\\_sw&s\[\]](http://proyecto-ciaa.com.ar/devwiki/doku.php?id=desarrollo:firmware:instalacion_sw&s[])=puente  
Fecha: 15/10/16 11:00 am.
- [5] [http://proyecto-ciaa.com.ar/devwiki/doku.php?id=docu:fw:bm:make&s\[\]](http://proyecto-ciaa.com.ar/devwiki/doku.php?id=docu:fw:bm:make&s[])=makefile  
Fecha: 15/10/16 11:00 am.



[6] [http://proyecto-ciaa.com.ar/devwiki/doku.php?id=docu:fw:bm:estructura&s\[\]=estructura&s\[\]=de&s\[\]=directorios](http://proyecto-ciaa.com.ar/devwiki/doku.php?id=docu:fw:bm:estructura&s[]=estructura&s[]=de&s[]=directorios)  
Fecha: 15/10/16 11:00 am.

[7] *sAPI versión 0.3.0: Introducción*

Documento existente sobre el repositorio ubicado en *Github*.

Dirección web: <https://github.com/epernia/sAPI>

[8] *.An Embedded Software Primer* Primera Edición Autor: David E. Simon Editorial: Pearson Education. Pags: 138-139.

[9] *UM10503 LPC43xx/LPC43Sxx ARM Cortex-M4/M0 multi-core microcontroller* Capítulo 40.

[10] <https://github.com/MatiasFerraro/CIAA-MatiasFerraro>

Fecha: 30/10/16 16:00 pm.

[11] *Introduccion a RTOS* por Alejandro Celery. SASE 2011.  
[http://www.sase.com.ar/2011/files/2010/11/SASE2011-Introduccion\\_RTOS.pdf](http://www.sase.com.ar/2011/files/2010/11/SASE2011-Introduccion_RTOS.pdf)

[12] *OSEK/VDX Operating System Version 2.2.3*

<http://www.irisa.fr/alf/downloads/puaut/TPNXT/images/os223.pdf>

[13] *Introducción a OSEK OS: El Sistema Operativo del CIAA-Firmware*. Primera Edición. Autor: Mariano Cerdeiro

[14] Extraído de pagina web:

<https://groups.google.com/forum/#searchin/ciaa-firmware/boolean%7Csort:relevance/ciaa-firmware/SNLmTVZbIkk/iSPYr-l-EAAJ>

[15] Enlace web de repositorio de librería que implementa interrupción externa sobre GPIO0:

[https://gitlab.com/pridolfi/rtos\\_ii\\_ejercicios/tree/master/gpiont](https://gitlab.com/pridolfi/rtos_ii_ejercicios/tree/master/gpiont)