

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL CÓRDOBA



PROYECTO FINAL DE *Software en Tiempo Real*

Sistema con sensor controlador por una línea serial

Alumno:
Luis Alberto GUANUCO

Docentes:
Ing. Carlos CENTENO
Ing. Luis TOLEDO

6 de marzo de 2014

Índice general

1. Introducción	1
2. Especificaciones del diseño	2
2.1. <i>Hardware</i>	2
2.2. <i>Software</i>	2
3. Implementación del circuito electrónico	3
3.1. Circuito del sensor	3
3.1.1. Oscilador <i>Schmitt-Trigger</i>	3
3.1.2. Funcionamiento básico	5
3.2. Circuito del dsPIC®	8
4. μC/OS-II™ en el dsPIC®	12
4.1. Tareas del RTOS	14
4.2. Módulo DIO	17
4.3. Tarea para el control del sensor	20
4.4. Tarea procesamiento de datos del sensor	21
4.5. Tarea de comunicación serial	22
5. μC/OS-II™ en la PC	24
5.1. Tarea de comunicación serial	27
5.2. Tarea visualización de datos	29
6. Ensayos	31
A. Repositorio del proyecto	34
B. Códigos del proyecto	35

Índice de figuras

2.1. Esquema general de la implementación.	2
3.1. Circuito con <i>Schmitt-Trigger</i>	4
3.2. Curva V_{CC} vs. k	4
3.3. Proceso de conversión de variables en el sistema de censado.	5
3.4. Implementación del circuito electrónico.	6
3.5. Diagramas temporales de los ciclos de Espera y Adquisición.	7
3.6. Esquemático del circuito del microcontrolador, solo los bloques fundamentales.	10
4.1. Estados de las tareas en el $\mu C/OS-II^{\text{TM}}$	13
4.2. Diagrama de flujo del Módulo DIO.	17
4.3. Canales de entradas discretas del Módulo DIO.	18
4.4. Canales de salidas discretas del Módulo DIO.	18
4.5. Funcionamiento de los <i>MailBoxes</i>	22
5.1. Entorno gráfico del <i>software</i> TurboC.	24
5.2. Captura de la pantalla del proyecto en funcionamiento.	27
5.3. Diagrama en bloque del Módulo <i>COMM_PC</i>	28
6.1. Muestra 1.	32
6.2. Muestra 2.	32
6.3. Muestra 3.	33

Índice de cuadros

3.1. Rango de variación del LDR.	5
3.2. Rango de frecuencias de salida en función de la red RC	6
6.1. Captura de datos del sistema de sensor a diferentes niveles de intensidad de luz.	31

Resumen

El presente trabajo se enfocada en la implementación de un sistema de comunicación sobre una red serial donde se encuentran conectados un sensor, un sistema electrónico con un microcontrolador y por último se conecta una computadora personal (PC). Los sistemas electrónicos basados en un procesador poseen un Sistema Operativo en Tiempo Real, RTOS por sus siglas en inglés (*Real-Time Operating Systems*). El sensor presenta una particularidad, y es que tanto la alimentación del transductor como el canal de datos es el mismo. Los parámetros principales en el diseño son definidos por el circuito del sensor.

Sección 1

Introducción

La posibilidad de reducir el costo en cualquier sistema electrónico es uno de los grandes retos con el que se presenta el desarrollador electrónico. Si bien hay un gran avance en los procesos de fabricación de circuitos integrados, existen dos inconvenientes para hacerse de dicha tecnología. El primero es el *acceso* a los nuevos dispositivos electrónicos que se encuentran en el mercado. El segundo inconveniente que se podría enumerar, muy relacionado al primero, es el *costo* que tienen estos nuevos dispositivos. Obviamente que estos inconvenientes planteados son desde un perfil estudiantil. Seguramente en desarrollos industriales los factores de disponibilidad y costo son evaluados en función de las prestaciones que ofrece el dispositivo en cuestión. Pero como nuestro trabajo se encuentra orientado a un ámbito académico se priorizará el diseño que requiera mayor tiempo de investigación y desarrollo para obtener un sistema que cumpla con nuestros requerimientos a un costo reducido y que pueda ser reproducible fácilmente.

El procesamiento de datos se realizará sobre un dispositivo microcontrolador dsPIC[®] fabricado por Microchip Inc.. En este μC se encuentra embebido un RTOS donde se implementan varias tareas para el sistema operativo (OS). Debido a la complejidad del desarrollo, a nivel de *software*, no se presentan problemas en los tiempos de procesamiento del OS. Como se dijo en el Resumen del informe, las principales especificaciones son propias del sistema transductor (sensor).

La información relevada por el sistema embebido es transmitida a una PC, quién presentará la información obtenida del sensor. Siguiendo la línea del uso sistemas operativos en tiempo real, se implementa nuevamente un RTOS. Al igual que en el caso del dsPIC[®], aquí se adaptan la capa más baja del código máquina para compilar el sistema operativo con la arquitectura a utilizar. La mayoría de la información necesaria para esta *adaptación* se encuentra disponible por el desarrollador del RTOS (Micrium Inc.).

Sección 2

Especificaciones del diseño

Las especificaciones del desarrollo se las puede dividir en dos grupos. Por un lado especificaciones de *Hardware*, aquí se encuentra tanto el circuito del sensor como así también la plataforma de procesamiento y los recursos disponibles. Por otro lado se especifica el *Software* que se debe implementar, el sistema operativo en tiempo real $\mu\text{C}/\text{OS-II}^{\text{TM}}$. Ya familiarizado por el desarrollador del proyecto por ser el OS utilizado en la *Cátedra de Software en Tiempo Real* dictado como materia electiva en la Carrera Ingeniería Electrónica de la Universidad Tecnológica Nacional – Facultad Regional Córdoba.

2.1. Hardware

El esquema general del *hardware* se puede ver en la Fig. 2.1. Aquí se especifica el flujo de dato entre los diferentes bloques. Sobre las líneas de comunicación se tiene comentado que información dispone físicamente cada conexión. En la conexión entre el sensor y el dsPIC[®] se no solo se comparte información (datos) sino también se transmite energía para el circuito transductor. Entre el dsPIC[®] y la PC se implementa una comunicación RS-232.

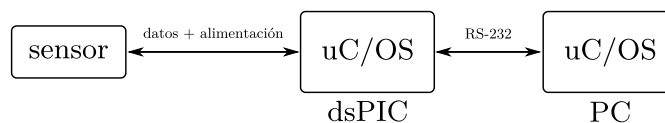


Figura 2.1: Esquema general de la implementación.

2.2. Software

Como se mencionó anteriormente, tanto en el dsPIC[®] como en el PC se implementan el sistema operativo $\mu\text{C}/\text{OS-II}^{\text{TM}}$. Si bien las especificaciones a nivel *hardware* no demanden el uso de un RTOS, ya que los requerimientos en la respuesta del procesamiento no son críticas, se hará uso de estos sistemas operativos como una aplicación de las características mas destacadas con la que cuentan los RTOS ($\mu\text{C}/\text{OS-II}^{\text{TM}}$ en nuestro caso). En las secciones siguientes se describirá con mayor detalle tanto el *hardware* como el *software*.

Sección 3

Implementación del circuito electrónico

A nivel electrónico, el desarrollo se divide en dos diseños. El primero es el circuito de sensor y el segundo es el sistema embebido con el dsPIC® como dispositivo principal.

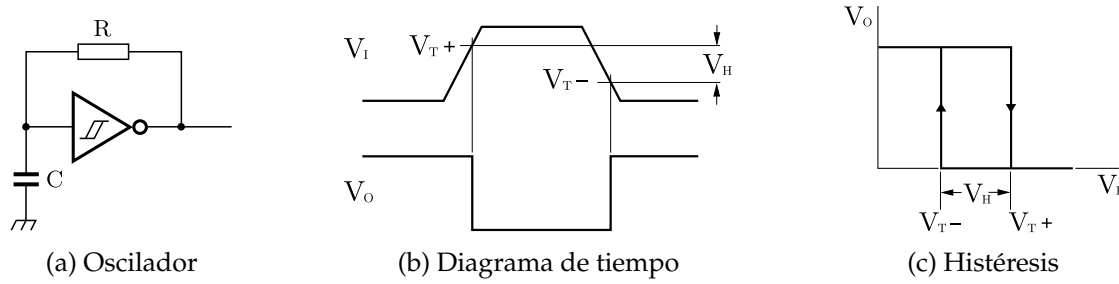
3.1. Circuito del sensor

La base del circuito transductor es un oscilador formado por un inversor digital (IC) y una red RC que determina la frecuencia de oscilación del oscilador. Para mantener más estable la frecuencia del oscilador se utiliza un inversor *Schmitt-Trigger*. Esto último se debe a que el sistema sensor presentará algunas perturbaciones en el nivel de la tensión de alimentación, causa que se abordará más adelante en esta sección. La transducción del fenómeno físico que se quiere adquirir se producirá afectando la red RC, es decir la frecuencia del oscilador. Por la disponibilidad de transductores discretos, se opta por usar un sensor fotoeléctrico ó LDR (por sus siglas en inglés *Light Dependant Resistor*).

3.1.1. Oscilador *Schmitt-Trigger*

El oscilador basado con un inversor es un circuito muy sencillo que demanda el uso de una red formada por un resistor y un capacitor, ilustrado en la Figura 3.1a. El inversor utilizado en este caso es el **MM74HCT14**¹, el cual dispone de ocho inversores con *Schmitt-Trigger*. El concepto sobre de *Schmitt-Trigger* fue inventado por el científico estadounidense *Otto H. Schmitt* en el año 1934. Se lo llama *Trigger* pues la salida del circuito retiene su estado hasta que la entrada presenta un valor suficiente para disparar (en inglés *trigger*) en cambio de estado. En la Figura 3.1b se observa los niveles umbrales de tensión (*threshold*) que presenta este tipo de inversor. El comportamiento dinámico de ambos niveles umbrales se lo puede graficar como un ciclo de histéresis (Figura 3.1c), este comportamiento permite decir el *Schmitt-Trigger* posee *memoria* y puede actuar como un circuito biestable (latch o flip-flop). Aunque su principal uso es el acondicionamiento de señales digitales, permitiendo remover el ruido en la señal.

¹Se puede utilizar cualquier otro integrado con inversores pero debe contar con la tecnología *Schmitt-Trigger*.

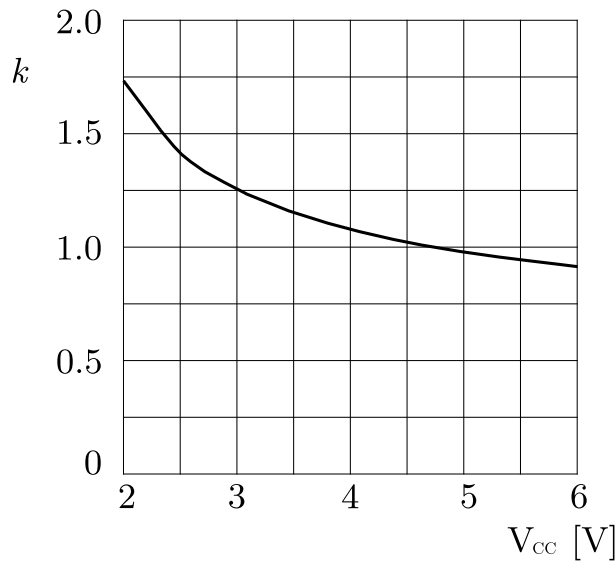
Figura 3.1: Circuito con *Schmitt-Trigger*.

La frecuencia de oscilación del circuito implementado en la Figura 3.1a es determinado por el valor de los componentes pasivos (RC), el nivel de la tensión de alimentación V_{CC} , y los niveles umbrales V_{T+} y V_{T-} . La ecuación (3.1) se obtuvo desde una nota de aplicación del integrado a utilizar.

$$f \approx \frac{1}{RC \ln \left[\frac{V_{T+}(V_{CC}-V_{T-})}{V_{T-}(V_{CC}-V_{T+})} \right]} \quad (3.1)$$

$$f \approx \frac{1}{RCk}$$

La ecuación (3.1) demuestra que puede considerarse como una constante k la operación logarítmica para facilitar la estimación de la frecuencia a la que oscila el circuito con el inversor. Se podría representar el comportamiento del valor constante k como una curva donde se considerará variaciones en el valor de la tensión de alimentación V_{CC} . La Figura 3.2 demuestra que es posible mantener estable la frecuencia aun con variaciones significativas en la tensión de alimentación. Más adelante se especificará como influye esto en nuestro caso, sobre todo por la particularidad de que nuestro sistema será sometido a perturbaciones intencionales en el nivel de alimentación.

Figura 3.2: Curva V_{CC} vs. k .

3.1.2. Funcionamiento básico

En la sección anterior se describe el funcionamiento del oscilador basado en un circuito con un inversor. En este circuito la frecuencia de oscilación, ecuación (3.1), es inversamente proporcional al producto RCk . Aquí vamos a considerar que tanto C como k son constantes, el parámetro que determinará la frecuencia es el valor de R . El resistor R es en nuestro caso el transductor LDR. Este transductor varía su parámetro de resistividad en función de la cantidad de luz que incida sobre él. Un esquema del proceso de conversión de variables se puede ver en la Figura 3.3.

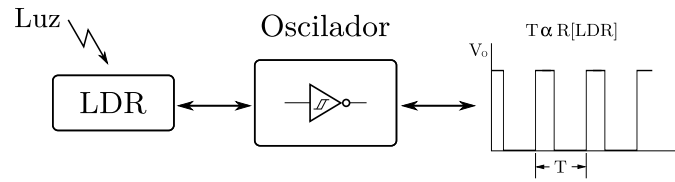


Figura 3.3: Proceso de conversión de variables en el sistema de censado.

En función a lo anteriormente dicho, se debe adquirir y procesar una onda oscilante donde la información se encuentra en el frecuencia que dicha señal posee. Además se especifica que *no se considerarán variaciones bruscas* en la intensidad de la luz recibida por el LDR. Para determinar los rangos máximos y mínimos en la variación de la resistividad del LDR se realizaron algunas mediciones que se pueden ver en la Tabla 3.1.

Condición de Luz	Valor de resistividad
Oscuridad	180 K Ω
Claridad	1 K Ω

Tabla 3.1: Rango de variación del LDR.

Con la información del rango resistivo del LDR se puede definir el rango de frecuencias de la señal que se obtendrá del sensor oscilador. Otra especificación a tener en cuenta es la frecuencia de muestreo que dispone el bloque siguiente al sensor, el microcontrolador dsPIC[®]. Aquí se tiene que el tiempo de muestreo es de 100 μ S.. Según el teorema del muestreo de Nyquist-Shannon, se podría definir un límite en la frecuencia máxima posible a ser adquirida. El enunciado de este teorema dice,

$$F_S > 2F_{MÁX}, \text{ donde } F_{MÁX} \text{ es la frecuencia máxima de la señal.} \quad (3.2)$$

en función a esta definición, se puede encontrar cual será la frecuencia máxima que se detectará correctamente si la frecuencia de muestreo es de 100 μ S. Es decir,

$$\begin{aligned} \text{sí} \quad F_S &> 2F_{MÁX} \\ \text{entonces} \quad F_{MÁX} &< \frac{F_S}{2} \\ F_{MÁX} &< \frac{1}{2T_S} \\ F_{MÁX} &< \frac{1}{2 \cdot 100 \mu S} = 5 \text{ KHz} \end{aligned} \quad (3.3)$$

Una vez obtenida la máxima frecuencia que se puede reconocer por parte del sensor, conjuntamente con los valores de resistividad (Tabla 3.1), se realizan ensayos para

determinar el valor que debe tener el capacitor de la red RC del oscilador². En la Tabla 3.2 se muestran dos configuraciones diferentes para distintos valores de capacitancia (10 nF y 100 nF).

Capacitor	Rango del LDR [Ω]	Frecuencia del osc.
10 nF	1 K Ω	100 KHz
	180 K Ω	555 Hz
100 nF	1 K Ω	10 KHz
	180 K Ω	55 Hz

Tabla 3.2: Rango de frecuencias de salida en función de la red RC .

En función a la Tabla 3.2 se puede decir que el valor de capacidad más apropiado es 100 nF. Pues con este valor se tiene un rango de frecuencia aceptable para la frecuencia de muestreo de 100 μ s.

Una *característica importante* del circuito planteado en este trabajo es la posibilidad de *reutilizar* la línea de comunicación como canal de alimentación. El circuito a utilizar se presenta en la Fig. 3.4. En el esquema de conexión se destacan dos componentes que son responsables de mantener el nivel de tensión de alimentación, el capacitor electrolítico $C1$ y el diodo $D1$. El capacitor electrolítico almacenará carga recibida de la línea serial, direccionada por $D1$. En el momento que la línea baje su nivel de tensión, el capacitor proporcionará la corriente al circuito inversor, obviamente que lo hará por un tiempo reducido ya que comenzará a descargarse. El resistor $R2$ permite aumentar la impedancia de la salida del inversor (TTL).

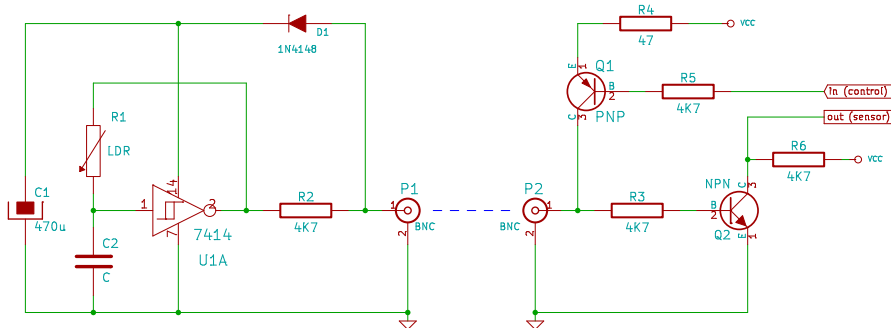


Figura 3.4: Implementación del circuito electrónico.

La línea de comunicación/alimentación conecta el circuito del sensor (del lado izquierdo de la Figura 3.4) con el circuito de control del sistema (a la derecha de la figura). El sistema de control es muy sencillo y cuenta de solo dos secuencias.

Mode de espera, este modo consiste en mantener la línea a un nivel de tensión necesario para alimentar el circuito del sensor (inversor). Para esta situación se debe enviar una señal de control a la base del transistor $Q1$. La señal *in(control)* proviene del dsPIC[®], controlador por el RTOS, que veremos en las siguientes secciones. Si la señal presenta un 0 lógico el transistor PNP se polarizará y la línea quedará conectada a la tensión V_{CC} . La señal de salida *out(sensor)* tendrá el mismo nivel que la línea (V_{CC}) ya que el transistor $Q2$ está polarizado con la señal de la línea.

²Se considera $k = 1$

Modo adquisición, la adquisición de la señal oscilante que genera el circuito inversor es transmitida cuando la línea es liberada por el transistor Q1. En el momento que se presenta un 1 en la señal de control de Q1, éste queda abierto. Por una lado, el *Schmitt-trigger* deja de ser alimentado por la línea pero recibe carga del capacitor C1. Por otro lado la señal generada por el oscilador es transmitida a la línea que será amplificada por el transistor Q2. Si bien la señal del oscilador podría encontrarse debilita por pérdidas en la línea, el circuito de Q2 permite polarizar con niveles bajos de corriente el transistor NPN.

Esta secuencia debe mantener una relación entre el *ciclo de adquisición* y *ciclo de espera*. Por ejemplo, un ciclo de adquisición muy largo provocará una elevada reducción en la tensión de alimentación del inversor que, en este ciclo, es proporcionado por el capacitor C1. En caso contrario, un ciclo de adquisición muy corto podría no registrar el ciclo completo de oscilación del sensor. Una relación que se consideró apropiada es $1/10$, para un ciclo completo de control T el tiempo de adquisición se toma $1/T$ y el tiempo de espera $9/T$. La representación temporal de este proceso se puede ver en la Figura 3.5.

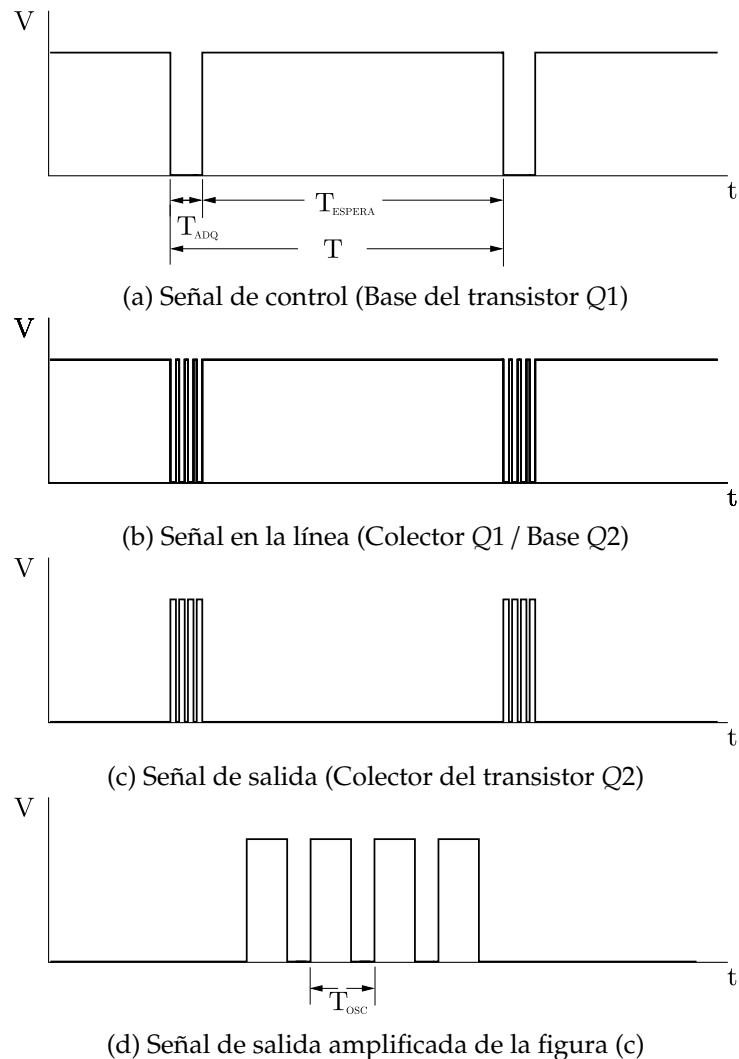


Figura 3.5: Diagramas temporales de los ciclos de Espera y Adquisición.

3.2. Circuito del dsPIC®

Para realizar el proceso de control del sistema de censado se utiliza un microcontrolador del fabricante Microchip, el DSPIC33FJ128GP804-E/PT. Este dispositivo presenta enormes prestaciones en recursos de *hardware*, aquí se presentan algunos de ellos:

Clock Management

- 2 % internal oscillator
- Programmable PLL and oscillator clock sources
- Fail-Safe Clock Monitor (FSCM)
- Independent Watchdog Timer
- Low-power management modes
- Fast wake-up and start-up

Core Performance

- Up to 40 MIPS 16-bit dsPIC33F CPU
- Single-cycle MUL plus hardware divide

Communication Interfaces

- Parallel Master Port (PMP)
- Two UART modules (10 Mbps)
 - Supports LIN 2.0 protocols
 - RS-232, RS-485, and IrDA® support
- Two 4-wire SPI modules (15 Mbps)
- Enhanced CAN (ECAN) module (1 Mbaud) with 2.0B support
- I2C module (100K, 400K and 1Mbaud) with SMBus support
- Data Converter Interface (DCI) module with I2S codec support

System Peripherals

- 16-bit dual channel 100 ksps Audio DAC
- Cyclic Redundancy Check (CRC) module
- Up to five 16-bit and up to two 32-bit Timers/Counters
- Up to four Input Capture (IC) modules
- Up to four Output Compare (OC) modules
- Real-Time Clock and Calendar (RTCC) module

Advanced Analog Features

- 10/12-bit ADC with 1.1Msps/500 ksps rate:
 - Up to 13 ADC input channels and four S&H
 - Flexible/Independent trigger sources

- 150 ns Comparators:
 - Up to two Analog Comparator modules
 - 4-bit DAC with two ranges for Analog Comparators

Input/Output

- Software remappable pin functions
- 5V-tolerant pins
- Selectable open drain and internal pull-ups
- Up to 5 mA overvoltage clamp current/pin
- Multiple external interrupts

Direct Memory Access (DMA)

- 8-channel DMA with no CPU stalls or overhead
- UART, SPI, ADC, ECAN, IC, OC, INT0

Debugger Development Support

- In-circuit and in-application programming
- Two program breakpoints
- Trace and run-time watch

La placa de desarrollo de este microcontrolador es un diseño realizado por estudiantes de nuestra casa de estudios *Andrés Hoc* y *Gonzalo Vassia*³, Figura 3.6. Para la programación de este microcontrolador se utilizó el entorno de desarrollo *MPLAB® IDE 8.8v* y el correspondiente compilador *MPLAB C30 C Compiler 3.32v*.

Como se mencionó anteriormente, quizá la complejidad del planteo del proyecto no requiere tanta capacidad de procesamiento, como así también justifique la utilización de un sistema operativo en tiempo real a fin de lograr adquirir la información censada. Pero debido a que el presente trabajo es una implementación de los RTOS, se prioriza la posibilidad de experimentar el manejo de sistemas operativos en desarrollos electrónicos. Los recursos de hardware a utilizar son:

Puerto de Entradas y Salidas, son necesarios para el control del circuito de censado que se describió en la Sección 3.1.2.

UART, se utiliza el interfaz serial UART sobre RS-232 para comunicarse con la PC quién procesará y mostrará la información censada al usuario.

El modo en que serán utilizados y su configuración es tema del código fuente del *software* del proyecto. La estructura de directorio para la compilación del $\mu\text{C}/\text{OS-II}^{\text{TM}}$ en el dsPIC[®] se puede ver en el siguiente árbol,

```
firmware_dsPIC
├── stdint.h
└── p33FJ128GP804.inc
```

³Los estudiantes realizaron la placa con la finalidad de disponer un diseño sencillo y flexible para múltiples propósitos.

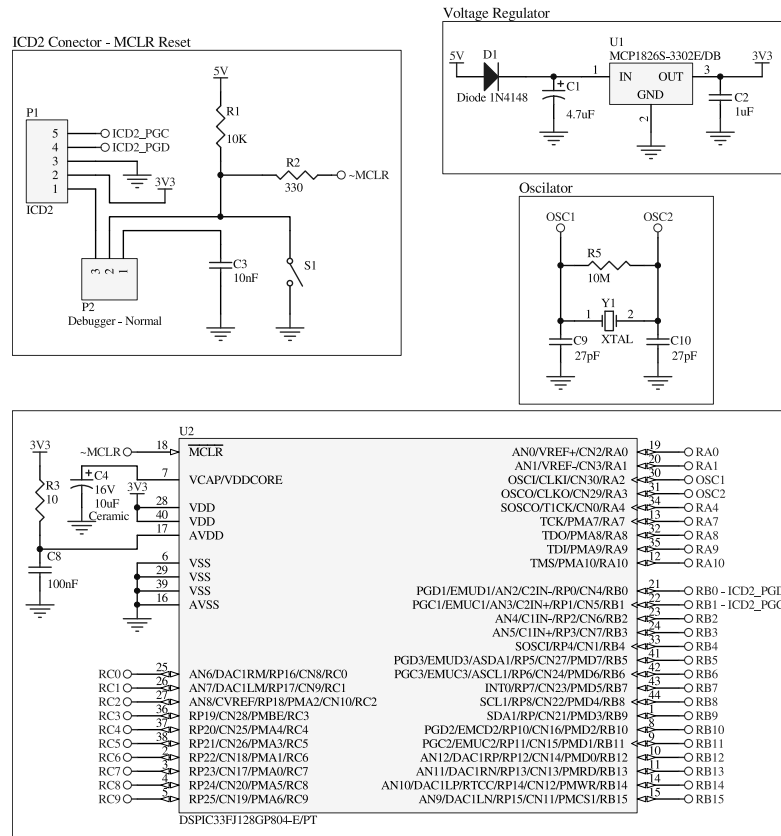


Figura 3.6: Esquemático del circuito del microcontrolador, solo los bloques fundamentales.

```

├─ p33FJ128GP804.h
├─ p33FJ128GP804.gld
├─ os_cfg.h
├─ MeDef.mcw
├─ MeDef.mcp
├─ isr.s
├─ interfaz_cfg.h
├─ interfaz_cfg.c
├─ iniinterfaz.c
├─ includes.h
├─ globals.h
├─ globals.c
├─ GenericTypeDefs.h
├─ dsPIC.delay.h
├─ dsPIC.delay.c
├─ dsPIC_cfg.h
├─ dsPIC_cfg.c
├─ dsPIC.a.s
├─ drivers
├─ └─ UART.h

```

```
├── UART.c
├── DIO
│   └── SOURCE
│       ├── DIO.C
│       └── DIO.H
├── configobj.h
├── configinterfaz.h
├── configinterfaz.c
├── Compiler.h
├── app_hooks.c
├── app_cfg.h
├── app.c
├── RTOS
│   ├── ucos_ii.h
│   ├── os_tmr.c
│   ├── os_time.c
│   ├── os_task.c
│   ├── os_sem.c
│   ├── os_q.c
│   ├── os_mutex.c
│   ├── os_mem.c
│   ├── os_mbox.c
│   ├── os_flag.c
│   ├── os_dbg.c
│   ├── os_cpu_util.a.s
│   ├── os_cpu.h
│   ├── os_cpu.c.c
│   ├── os_cpu.a.s
│   ├── os_core.c
│   ├── lib_str.h
│   ├── lib_str.c
│   ├── lib_mem.h
│   ├── lib_mem.c
│   ├── lib_def.h
│   ├── cpu.h
│   └── cpu.def.h
```


Sección 4

μ C/OS-II™ en el dsPIC®

μ C/OS-II™ es el acrónimo de *Micro-Controller Operating Systems Version 2*. Este es un sistema operativo multitareas en tiempo real *pre-emptive* basado en prioridades en su funcionamiento. El código se encuentra escrito principalmente en lenguaje de programación C. Se encuentra destinado para el uso en sistemas embebidos. Sus características son:

- Es un *kernel* de tiempo real muy pequeño.
- Con un espacio en memoria de aproximadamente unos 20KB para un completo funcionamiento del *kernel*.
- El código fuente se encuentra escrito en su mayoría en ANSI C.
- Gran portabilidad, muy escalable, *preemptive* en tiempo real, determinístico, *kernel* multitareas.
- Puede manejar hasta 64 tareas (con 56 tareas disponibles al usuario).
- Se encuentra disponible para más de 100 microcontroladores y microcontroladores.
- Es sencillo de usar y simple de implementar a la vez que es muy eficiente a la hora de comparar con otros RTOS en relación *precio/performance*.
- Soporta todos los tipos de procesadores desde 8-bit a 64-bit.

En el presente documento no se describirá el funcionamiento del μ C/OS-II™. No solo porque nuestra intención es la implementación del mismo, sino también se dispone de una gran cantidad de documentación sobre estos RTOS. De todas formas se señalará algunas características básicas de como trabaja el μ C/OS-II™.

El *kernel* del μ C/OS-II™ dispone de los recursos de hardware, sobre todo los recursos de procesamiento y el indexado de memoria. Para cada implementación del RTOS se debe adaptar tanto el compilador como los *drivers* o *ports* necesarios para facilitar el acceso a los diferentes periféricos del microcontrolador. Una vez logrado establecer estos requerimientos de *software* simplemente se agrega al μ C/OS-II™ *tareas* para que el RTOS las procese. Una *tarea* es un código de programa el cual, en principio, se puede escribir con la idea de suponer que se dispone de todo el CPU para ella misma. El

proceso de diseño para una aplicación en tiempo real implica la división del trabajo a realizar en tareas que son responsables de una parte del problema. A cada tarea se le asigna una prioridad, registros del CPU, y su propia área de stack para almacenar las variables locales que contiene cuando se está conmutando de tareas en el proceso *multitasking*. Cada tarea se encuentra en un bucle infinito (*loop*) el cual, en el tiempo de ejecución del código en el hardware, puede encontrarse en cualquier de cinco estados posibles:

DORMANT

READY

RUNNING

WAITING

INTERRUPTED

El estado *DORMANT* corresponde a una tarea que reside en memoria pero no ha sido habilitada a ingresar al *multitasking*. Una tarea está *READY* cuando puede ejecutarse pero su prioridad es menor que la tarea que se encuentra actualmente corriendo. Una tarea está en *RUNNING* cuando tiene el control del CPU. Una tarea está en *WAITING* cuando requiere que ocurra un evento (por ejemplo; esperando que a una operación de un periférico se complete). Finalmente, una tarea está en el estado *INTERRUPTED* cuando una interrupción ocurrió y el CPU se encuentra atendiendo este llamado. La Figura 4.1 se muestran los estados anteriormente nombrados y además se puede ver las funciones provistas por el $\mu\text{C}/\text{OS-II}^{\text{TM}}$ para realizar la conmutación entre un estado a otro.

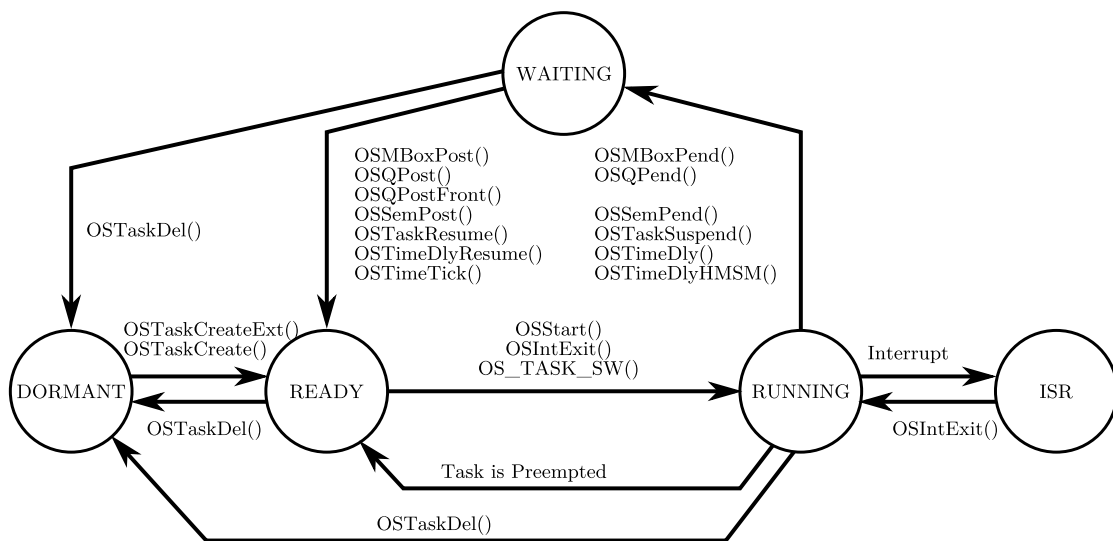


Figura 4.1: Estados de las tareas en el $\mu\text{C}/\text{OS-II}^{\text{TM}}$.

4.1. Tareas del RTOS

Antes de comenzar a escribir el código de las *tareas* se define los procesos requeridos. A continuación se lista las tareas necesarias para el proyecto

- Generación de la señal de control para el sistema del sensor (Sección 3.1.2).
- Procesamiento de los datos recibidos por el sensor.
- Comunicación serial con la PC (envío de datos adquiridos).

Obviamente que se requerirá de otras tareas pero son requerimientos del $\mu\text{C}/\text{OS-II}^{\text{TM}}$ son comunes en cualquier proyecto de este tipo. La función principal del proyecto *main()* contiene el llamado a todas las funciones de inicialización y configuración del microcontrolador, Código 4.1. Luego del llamado a la función *OSInit()* se debe crear todas las tareas del RTOS previo a que el sistema operativo entre en un estado de funcionamiento a través del llamado a *OSStart()*. Como convención se creará una tarea *TareaInicio()* que será donde se lancen la creación de todas las tareas. También se puede ver que se hace el llamado a las funciones *DIOInit()* que inicializa el módulo DIO que será descrito en la Sección 4.2.

Código 4.1: Función principal del proyecto, *main()*

```

CPU_INT16S  main (void)
{
    CPU_INT08U  err;

    BSP_IntDisAll(); /* Disable all interrupts until we are ready to accept
                       them */

    RCON = 0;

    OSInit(); /* Initialize 'uC/OS-II, The Real-Time Kernel' */
    DIOInit();

    OSTaskCreateExt(TareaInicio,
        (void *)0,
        (OS_STK *)&tareaInicioStk[0],
        TAREA_INICIO_PRIO,
        TAREA_INICIO_PRIO,
        (OS_STK *)&tareaInicioStk[TAREA_INICIO_STK_SIZE-1],
        TAREA_INICIO_STK_SIZE,
        (void *)0,
        OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);

    #if OS_TASK_NAME_SIZE > 11
    OSTaskNameSet(TAREA_INICIO_PRIO, (CPU_INT08U *)'Start Task', &err);
    #endif

    OSStart(); /* Start multitasking (i.e. give control to uC/OS-II) */

    return (-1); /* Return an error - This line of code is unreachable */
}

```

La *TareaInicio()* realiza el llamado a la función *AppTaskCreate()* que será la encargada de crear todas las tareas del proyecto. Además en esta primera tarea se crea e inicializa

el *MailBox* que será utilizado para la comunicación interna entre tareas ofrecido por el *kernel*, pero esto se verá más adelante en la Sección 4.4.

Código 4.2: Función de la primer tarea creada, *TareaInicio()*

```
static void TareaInicio (void *p_arg)
{
    CPU_INT08U i;
    CPU_INT08U j;

    (void)p_arg;

    BSP_Init(); /* Initialize BSP functions */

    #if OS_TASK_STAT_EN > 0
    OSStatInit(); /* Determine CPU capacity */
    #endif

    AppTaskCreate(); /* Create additional user tasks */

    /*Se crea un MailBox para indicar cuando hay un nuevo periodo de
    velocidad */
    mBoxPromData= OSMboxCreate((void *) NULL);
    mBoxPromData->OSEventPtr = 0;

    while (DEF_TRUE)
    { /* Task body, always written as an infinite loop. */
        OSTimeDly(100);
    }
}
```

La creación de las tareas se pueden ver en el código 4.3. Esta función solo realiza el llamado a la función *OSCreateExt()* que permite crear tareas y que sea procesada por el $\mu\text{C}/\text{OS-II}^{\text{TM}}$. Los argumentos que se debe proporcionar para cada llamado son

```
INT8U OSTaskCreateExt (void (*task)(void *p_arg),
                        void *p_arg,
                        OS_STK *ptos,
                        INT8U prio,
                        INT16U id,
                        OS_STK *pbos,
                        INT32U stk_size,
                        void *pext,
                        INT16U opt)
```

task: es un puntero a la función de la tarea.

p_arg: es un puntero opcional a una área de datos el cual pueda ser usado para pasar parámetros a la tarea cuando es ejecutado por primera vez.

ptos: es un puntero al extremo superior del *stack*. (Debe revisarse la constate de configuración *OS_STK_GROWTH*).

prio: es la prioridad de la tarea. Un único número debe ser asignado para cada tarea.

id: es el identificador de la tarea (0 ··· 65535).

pbos: es un puntero al extremo inferior del *stack*. (Debe revisarse la constate de configuración *OS_STK_GROWTH*).

stk_size: es el tamaño del *stack* en número de elementos.

pext: es el puntero a una posición de memoria suministrada por el usuario para ser usada como una extensión *TCB (Task Control Block)*.

opt: contiene información adicional (u opcional) sobre el comportamiento de la tarea.

- `OS_TASK_OPT_STK_CHK`, chequeo del *stack* está permitido por la tarea.
- `OS_TASK_OPT_STK_CLR`, limpiar el *stack* cuando la tarea es creada.
- `OS_TASK_OPT_SAVE_FP`, si el CPU tiene registros de punto-flotante, guardarlos durante un intercambio de estado.

En función de los argumentos recibidos, la función *OSCreateExt()* devuelve,

OS_ERR_NONE: si la función ha sido creada correctamente.

OS_PRIO_EXIT: si la prioridad de la tarea ya existe.

OS_ERR_PRIO_INVALID: si la prioridad especificada es mayor que la prioridad más alta permitida.

OS_ERR_TASK_CREATE_ISR: si intenta crear una tarea desde un *ISR*

Código 4.3: Función de la tarea *AppTaskCreate()*

```
static void AppTaskCreate (void)
{
    CPU_INT08U err;

    OSTaskCreateExt(TareaControlSensor,
        (void *)0,
        (OS_STK *)&tareaControlSensor[0],
        TAREA_CONTROLSSENSOR_PRIO,
        TAREA_CONTROLSSENSOR_PRIO,
        (OS_STK *)&tareaControlSensorStk[TAREA_CONTROLSSENSOR_STK_SIZE-1],
        TAREA_CONTROLSSENSOR_STK_SIZE,
        (void *)0,
        OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR | OS_TASK_OPT_SAVE_FP);

    OSTaskCreateExt(TareaPromDatos,
        (void *)0,
        (OS_STK *)&tareaPromDatos[0],
        TAREA_PROMDATOS_PRIO,
        TAREA_PROMDATOS_PRIO,
        (OS_STK *)&tareaPromDatosStk[TAREA_PROMDATOS_STK_SIZE-1],
        TAREA_PROMDATOS_STK_SIZE,
        (void *)0,
        OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR | OS_TASK_OPT_SAVE_FP);

    OSTaskCreateExt(TareaComSerial,
        (void *)0,
        (OS_STK *)&tareaComSerial[0],
        TAREA_COMSERIAL_PRIO,
        TAREA_COMSERIAL_PRIO,
        (OS_STK *)&tareaComSerialStk[TAREA_COMSERIAL_STK_SIZE-1],
        TAREA_COMSERIAL_STK_SIZE,
```

```

(void *)0,
OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR | OS_TASK_OPT_SAVE_FP);
}

```

4.2. Módulo DIO

El módulo *DIO* es un bloque de código que fue desarrollado por el mismo grupo que escribió el $\mu\text{C}/\text{OS-II}^{\text{TM}}$. Este se encuentra adaptado a la estructura de funcionamiento del RTOS. El bloque permite controlar canales de entradas y salidas digitales en forma independiente. En la Figura 4.2 se puede ver un diagrama del módulo completo. El módulo *DIO* consiste en una simple tarea (*DIOTask()*) que se ejecuta a intervalos regulares *DIO_TASK_DLY_TICKS*. *DIOTask()* puede manejar una gran cantidad de canales discretos de Entradas y Salidas (hasta 250 cada una). El módulo *DIO* se inicializa con la llamada a la función *DIOInit()*. Cada *DIO_TASK_DLY_TICKS*, *DIOTask()* llama a las funciones *DIRd()*, *DIUpdate()*, *DOWr()* y *DOUpdate()*. *DITbl[]* es una tabla que contiene configuración e información en tiempo real de cada canal de entrada. Las entradas son leídas y mapeadas a *DITbl[i].DIIn* por el controlador del *hardware* implementado en la función *DIRd()*. *DIRd()* es la función que interacciona con el dispositivo físico (puerto del dsPIC®). Además en la Figura 4.2 se puede ver que se cuentan con varias funciones que permiten abstraerse del *hardware* y disponer de ellos, siendo la tarea del módulo *DIO* la que lidie con los problemas que puedan existir en la lectura y escritura de los puertos físicos de nuestra placa.

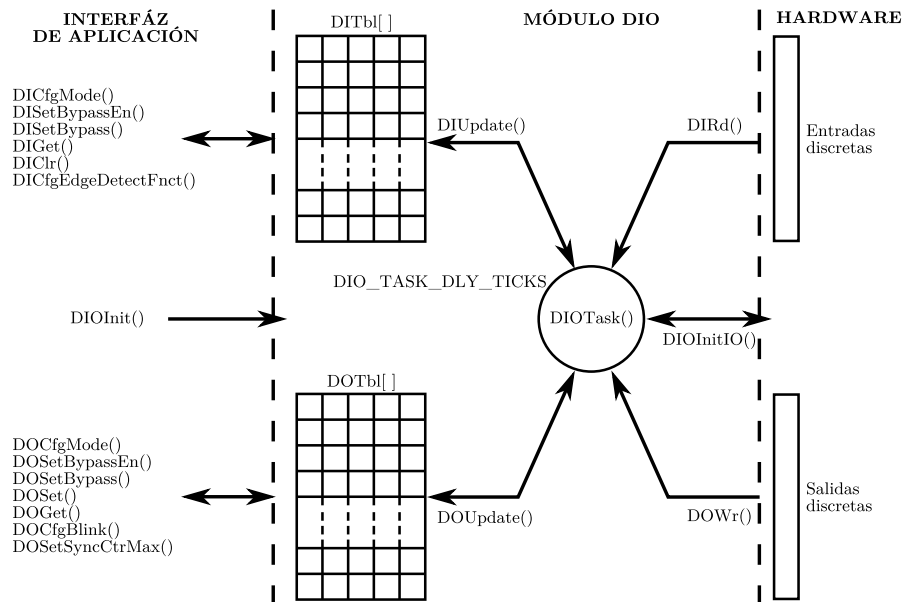


Figura 4.2: Diagrama de flujo del Módulo DIO.

El diagrama en bloque de la Figura 4.3 muestra el funcionamiento del módulo *DIO* para el tratado de las señales de entrada. *.DIIn*, *.DIModeSel*, *.DIBypassEn* y *.DVal* son miembros de la estructura de datos *DIO_DI* que se encuentra definida en el código

fuente del módulo dio (archivo `DIO.H`). `DIUUpdate()` es el responsable de actualizar todas los canales de entradas discretas del módulo.

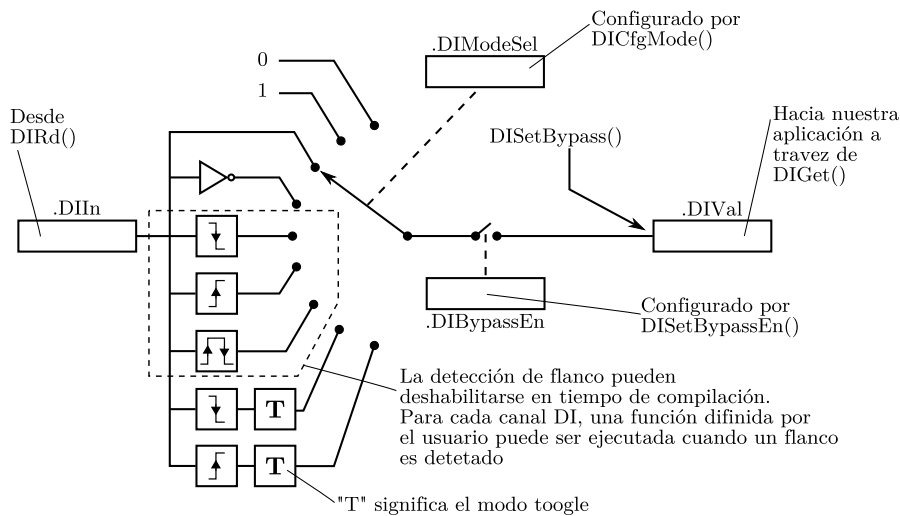


Figura 4.3: Canales de entradas discretas del Módulo DIO.

Al igual que los canales de entrada, Figura 4.2, *DOTbl[]* es una tabla que contiene configuración e información en tiempo real por cada canal de salidas. Las salidas discretas son mapeadas desde *DOTbl[i].DOOut* al puerto físico a través de la función *DOWr()*. La función *DOWr()* es el controlador o *driver* que interacciona con la capa física de nuestro sistema (al igual que *DIRr()*). En la Figura 4.4 se muestra un diagrama de como funciona el módulo DIO para las señales de salida de nuestro bloque. *.DOCtrl*, *.DOBy-passEn*, *.DOBypass*, *.DOBlinkEnSel*, *.DOModeSel*, *.DOInv* y *.DOOut* son miembros de la estructura de datos *DIO_DO* definido en el archivo *DIO.H*. *DOUpdate()* es el responsable de actualizar todos los canales de salidas discretas.

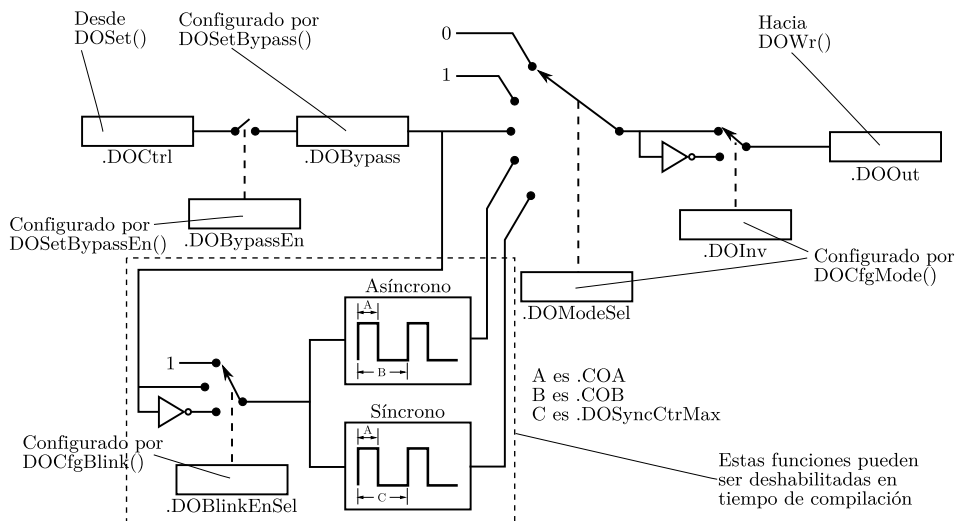


Figura 4.4: Canales de salidas discretas del Módulo DIO.

Como toda tarea que debe atender el RTOS, se debe asignar una prioridad y una latencia en ejecución. Se asigna una de las más altas prioridades al módulo pues será quien

controle el sensor. Las demás tareas planteadas en la Sección 4.1 no son tan demandantes. En el caso de especificar a que frecuencia debe ejecutarse el módulo DIO, se considera el valor más alto, lo que implica que se refrescará las Entradas/Salidas lo más rápido posible. El archivo DIO.H contiene varios `#define` que parametrizan el comportamiento del módulo.

Código 4.4: Definiciones de parámetros de prioridad y latencia del Módulo DIO.

```
#define DIO_TASK_PRIO          1
#define DIO_TASK_DLY_TICKS    1
```

Como se dijo anteriormente, las funciones *DOWr()* y *DIRd()* serán quienes interactúen con el *hardware*. Originalmente el código del módulo DIO estaba escrito para acceder al puerto paralelo de una computadora de escritorio. La configuración de los puertos del microcontrolador se realiza en la función *DIOInitIO()*,

Código 4.5: Función *DIOInitIO()*

```
void DIOInitIO (void)
{
    /* Configuración de los puertos I/O */
    /* SALIDAS */
    TRISBbits.TRISB9 = 0;
    TRISBbits.TRISB8 = 0;
    TRISBbits.TRISB7 = 0;
    TRISBbits.TRISB6 = 0;

    /* ENTRADAS */
    TRISCbits.TRISC0 = 1;
    TRISCbits.TRISC1 = 1;
    TRISCbits.TRISC2 = 1;
    TRISCbits.TRISC3 = 1;

    AD1PCFGLbits.PCFG6 = 1;    //RC0
    AD1PCFGLbits.PCFG7 = 1;    //RC1
    AD1PCFGLbits.PCFG8 = 1;    //RC2
}
```

Una vez configurados los puertos, la función *DIOInitIO()* es llamada desde la función principal. La función *DIRd()* es modificada para asignar los puertos que serán leídos y cargados en la tabla *DITbl[]*.

Código 4.6: Función *DIRd()*

```
void DIRd (void)
{
    DIO_DI *pdi;
    INT8U i;
    INT8U in;
    INT8U msk;

    pdi = &DITbl[0]; /* Point at beginning of discrete inputs */
    msk = 0x01; /* Set mask to extract bit 0 */
    in = 0;
    in |= PORTCbits.RC0; /* Read the physical port (8 bits) */
    in |= (PORTCbits.RC1<<1);
    in |= (PORTCbits.RC2<<2);
    in |= (PORTCbits.RC3<<3);
    for (i = 0; i < 8; i++) /* Map all 8 bits to first 8 DI channels */
```



```

{
    pdi->DIIn    = (BOOLEAN)(in & msk) ? 1 : 0;
    msk          <=<= 1;
    pdi++;
}
}

```

Recordemos que el módulo DIO puede controlar hasta 256 canales de entradas y salidas. En nuestro caso no se requería de más de dos canales, uno de entrada y otro de salida. De todas formas se configuran cuatro canales I/O. Para la función que escribe el puerto de salida del microcontrolador, *DOWr()*, se debe asignar que puertos físicos tomarán los valores de la tabla *DOTbl[]*. Al igual que en el caso de la lectura, aquí se utilizaron solo cuatro canales de los que dispone el módulo DIO.

Código 4.7: Función *DIWr()*

```

void DOWr (void)
{
    DIO_DO    *pdo;
    INT8U     i;
    INT8U     out;
    INT8U     msk;

    pdo = &DOTbl[0]; /* Point at first discrete output channel */
    msk = 0x01; /* First DO will be mapped to bit 0 */
    out = 0x00; /* Local 8 bit port image */
    for (i = 0; i < 8; i++) /* Map first 8 DO to 8 bit port image */
    {
        if (pdo->DOOut == TRUE) {
            out |= msk;
        }
        msk <=<= 1;
        pdo++;
    }
    PORTBbits.RB9 = out & 0x01;
    PORTBbits.RB8 = (out>>1) & 0x01;
    PORTBbits.RB7 = (out>>2) & 0x01;
    PORTBbits.RB6 = (out>>3) & 0x01;
}

```

4.3. Tarea para el control del sensor

En la Sección 3.1.2 se describe el funcionamiento del sistema de censado que se implementa en el correspondiente trabajo. Las señales que controlan el proceso de adquisición de datos a través de la línea de comunicación son generados por una tarea del $\mu\text{C}/\text{OS-II}^{\text{TM}}$. La tarea *TareaControlSensor()* utiliza el módulo DIO para generar la señal de *control*, véase el circuito de la Figura 3.4. El tiempo de *adquisición* y el tiempo de *espera* es controlado por las funciones de tiempo del RTOS,

Código 4.8: Función de la tarea *TareaControlSensor()*

```

static void TareaControlSensor (void)
{
    // Configuración de canales del módulo DIO

```

```

DOCfgMode(0, DO_MODE_DIRECT, 0);
DICfgMode(0, DI_MODE_EDGE_HIGH_GOING);
ValChZero = 0;
CntData = 0;
DIClr(0);

while(1)
{
    DOSet(0, TRUE);
    OSTimeDlyHMSM(0, 0, 0, 100);
    DOSet(0, FALSE);
    OSTimeDlyHMSM(0, 0, 1, 0);

    ValChZero += DIGet(0);
    CntData++;
    DIClr(0);
}
}

```

La salida del microcontrolador que maneja la señal de control del sensor será el primer canal del módulo DIO (canal 0). En esta función se puede apreciar que el tiempo de *adquisición* es 100 microsegundos y el tiempo de *espera* es de 1 segundo, relación especificada en la Sección 3.1.2.

La entrada será el canal 0 y es conectada a la señal de salida del sensor (colector del transistor Q2, Fig. 3.4). Este canal se configura en el modo de detección de flanco ascendente, *DICfgMode(0, DI_MODE_EDGE_HIGH_GOING)*. Se acumula en la variable *ValChZero* la cantidad de flancos recibidos desde la línea de comunicación y a la vez se acumula el número de muestras recibidas, pues se enviará un promedio del periodo calculado. Se aclara que la mayoría de las variables que se utilizan en este proyecto son variables globales. La declaración de estas variables se pueden ver al final del documento donde se proporciona todo el código fuente del proyecto.

4.4. Tarea procesamiento de datos del sensor

La tarea *TareaPromDatos()* realiza un promedio de la cantidad de pulsos recibidos. La tarea procesa la cantidad de flancos recibidos, que son acumulados en la variable *ValChZero*, y los promedia con la cantidad de veces que se realiza la captura, *CntData*. Este proceso se realiza a una frecuencia menor que la tarea *TareaControlSensor()*, cada 4 segundos.

En esta función se implementan los *MailBoxes* para comunicarse con la tarea que envíe los datos por el puerto serie. Los *MailBoxes* son recursos que ofrece el $\mu\text{C}/\text{OS-II}^{\text{TM}}$ para intercambiar mensajes entre tareas a través del *kernel*. Una tarea que desee un mensaje desde un *MailBox* vacío es suspendida y puesta en una lista de espera hasta que el mensaje sea recibido. El *kernel* permite que la tarea espere por el mensaje hasta que haya transcurrido un tiempo determinado (*timeout*). Si el mensaje no es recibido, transcurrido el tiempo de espera, la tarea que requiere del mensaje pasará del estado *Ready* a *Run* retornará un código de error en el retorno del pedido. El mecanismo con el que funciona este servicio de mensajería del sistema operativo se puede ver en al

Figura 4.5. La función *OSMboxPost()* realiza la acción de **poner** el mensaje que en este caso es el valor promedio de los datos adquiridos. El primer argumento de la función *OSMboxPost()*, *mBoxPromData*, es una variable del tipo *OS_EVENT* que es una estructura de datos definida por el $\mu\text{C}/\text{OS-II}^{\text{TM}}$ para el manejo de información interna del sistema operativo.

Código 4.9: Función de la tarea *TareaPromDatos()*

```
static void TareaPromDatos (void)
{
    TRISBbits.TRISB5 = 0;

    while(1)
    {
        if(CntData!=0)
        {
            Prom = (float) ValChZero / (float) CntData;
            ValChZero = 0;
            CntData = 0;
        }
        else
        {
            Prom = 0;
        }

        OSMboxPost(mBoxPromData, &Prom); /* Envío msj. a tarea que transmite
            los datos (promedio de los pulsos recibidos) */

        OSTimeDlyHMSM(0,0,4,0);
    }
}
```

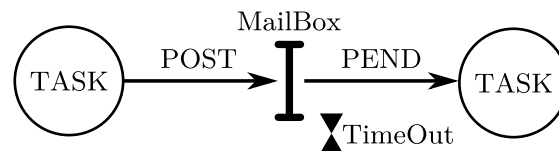


Figura 4.5: Funcionamiento de los *MailBoxes*.

4.5. Tarea de comunicación serial

La últimas de las tareas implementadas en el microcontrolador es la comunicación serial a través de la UART del dsPIC[®]. La configuración de la UART se realiza en la función *InitUART()*. La tarea *TareaComSerial()* se encuentra condicionada a la recepción del mensaje desde la tarea *TareaPromDatos()* a través del *MailBox* *mBoxPromData*. El tiempo de espera por el mensaje se define en *TIMEOUT_PEND_MBOX* y se encuentra en unidades ticks. Una vez que se reciba el mensaje, *OSMboxPend()* proporciona un puntero a la posición de memoria donde se encuentra el mensaje. Y sobre esta información se procesa el dato a ser enviado por el canal serial. La frecuencia con que trabaja esta tarea es de un segundo, pero como se dijo al comienzo, se encuentra condicionada por la espera del *MailBox*.

Código 4.10: Función de la tarea *TareaComSerial()*

```

static void TareaComSerial (void)
{
    uint8_t errorMboxPend;

    asm volatile (‘‘mov #OSCCONL, w1 \n’’
        ‘‘mov #0x46, w2 \n’’
        ‘‘mov #0x57, w3 \n’’
        ‘‘mov.b w2, [w1] \n’’
        ‘‘mov.b w3, [w1] \n’’
        ‘‘bclr OSCCON, #6’’);

    RPINR18bits.U1RXR = 22;
    RPOR11bits.RP23R = 0b00011;

    asm volatile (‘‘mov #OSCCONL, w1 \n’’
        ‘‘mov #0x46, w2 \n’’
        ‘‘mov #0x57, w3 \n’’
        ‘‘mov.b w2, [w1] \n’’
        ‘‘mov.b w3, [w1] \n’’
        ‘‘bset OSCCON, #6’’);

    InitUART1();

    while(1)
    {
        PtrPromFromMBox = OSMboxPend(mBoxPromData, TIMEOUT_PEND_MBOX, &
            errorMboxPend); /* Esperamos a que llegue un nuevo promedio */
        PromFromMbox = *PtrPromFromMBox;
        whole=(int) PromFromMbox ; /* obtains whole part */
        decimal=(int) ((PromFromMbox - (float) whole)*100.0); /* obtains
            decimal part (1 digit) */
        sprintf(array,‘‘:%04d.%02d\n‘‘,whole,decimal); /* converts to string */
        for(iiTaskUart = 0; iiTaskUart<9 ; iiTaskUart++)
        {
            while(U1STAbits.UTXBF != 0);
            U1TXREG = array[iiTaskUart];
        }

        OSTimeDlyHMSM(0,0,1,0);
    }
}

```

Sección 5

μ C/OS-II™ en la PC

Siguiendo la línea del uso de los sistemas operativos en tiempo real, se implementa el sistema μ C/OS-II™ en una arquitectura de PC estándar. Se utilizará un proyecto base publicado por Micrium Inc., creador del μ C/OS-II™, compilado con el *software TurboC*. Las características del proyecto serán,

- Procesar los datos recibidos a través de una comunicación serial con el RTOS que corre en el dsPIC®.
- Manejo de funciones gráficas para la presentación de los datos en la pantalla de la PC.

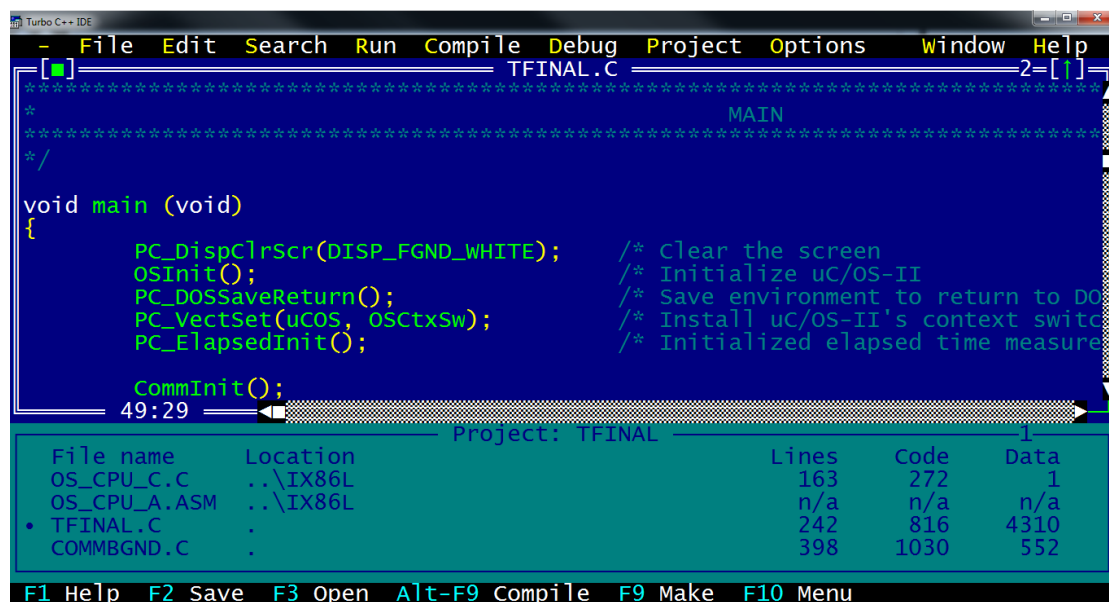


Figura 5.1: Entorno gráfico del *software TurboC*.

La función principal *main()* se puede ver en el Código 5.1. Como en el caso del μ C/OS-II™ en el dsPIC®, se debe realizar la configuración y la creación de las tareas luego del llamado a la función *OSInit()* y antes de que el RTOS comience a correr *OSStart()*.

Código 5.1: Función principal del μ C/OS-II™ sobre la PC (*main()*)

```

void main (void)
{
    PC_DisPClrScr(DISP_FGND_WHITE); /* Clear the screen */

    OSInit(); /* Initialize uC/OS-II */

    PC_DOSSaveReturn(); /* Save environment to return to DOS */
    PC_VectSet(uCOS, OSCtxSw); /* Install uC/OS-II's context switch vector */
    PC_ElapsedInit(); /* Initialized elapsed time measurement */

    CommInit();

    /* Crear los Semaphores (Buzones) */
    TransmisionesListasSem = OSSemCreate(0);

    /* Crear los MailBox (Buzones) */
    DatoRecibidoMbox=OSMboxCreate((void *)0);

    OSTaskCreate(TaskStart, (void *)0, &TaskStartStk[TASK_STK_SIZE-1], 0);

    OSStart(); /* Start multitasking */
}

```

Junto a los archivos del $\mu\text{C}/\text{OS-II}^{\text{TM}}$ se dispone de varios códigos fuentes para hacer uso de varios periféricos. Estos se encuentran en el directorio *BLOCKS*. Se puede listar estos archivos como,

```

SOFTWARE
├── BLOCKS
│   ├── AIO
│   ├── CLK
│   ├── COMM
│   │   └── SOURCE
│   │       ├── COMMRTOS.H
│   │       ├── COMMRTOS.C
│   │       ├── COMM_PC.H
│   │       ├── COMM_PC.C
│   │       ├── COMM_PCA.ASM
│   │       ├── COMMBGND.H
│   │       └── COMMBGND.C
│   ├── DIO
│   ├── KEY_MN
│   ├── LCD
│   ├── LED
│   ├── PC
│   │   └── SOURCE
│   │       ├── PC.C
│   │       └── PC.H
│   ├── SIMPLE
│   └── TMR
├── HPLISTC
├── UCOS-II
└── doc

```

Los bloques que se utilizarán se encuentran resaltados en texto negrita. Pero también se puede ver el módulo DIO utilizado en el dsPIC[®] también está listado en este directorio.

Para el manejo de la pantalla se utilizan funciones del módulo *PC*. Este módulo provee servicios para mostrar caracteres ASCII en una pantalla VGA básica de una computadora. En el modo normal, la pantalla de una PC puede manejar hasta 2000 caracteres organizados en un arreglo de 25 filas por 80 columnas. La memoria de vídeo en la PC y comienza en una dirección absoluta de memoria 0x000B8000 (o usando notación segmentada, B800:0000). Cada caracter a mostrar requiere dos *bytes* para ser visualizado en la pantalla. El primer *byte* (menor posición en memoria) es el caracter que se quiere mostrar, mientras que el segundo *byte* es un atributo que determina la combinación de color *foreground/background* del caracter. El color *foreground* es especificado en los menores 4 *bits* del atributo, mientras que el color *background* en los superiores, del *bits* 4 al 6. Finalmente el más significativo *bit* determina si el caracter titilará (con 1) o no (con 0).

El el Código 5.2 muestra la primer tarea donde se configura y ejecuta varias llamadas a las funciones de pantalla para nuestro proyecto.

Código 5.2: Primera tarea del μ C/OS-II[™]sobre la PC (*TaskStart()*)

```
void TaskStart (void *data)
{
    char    s[80];
    WORD key;

    data = data; /* Prevent compiler warning */

    OS_ENTER_CRITICAL(); /* Install uC/OS-II's clock tick ISR */
    PC_VectSet(0x08, OSTickISR);
    PC_SetTickRate(OS_TICKS_PER_SEC); /* Reprogram tick rate */
    OS_EXIT_CRITICAL();
    OSStatInit();

    OStaskCreate(Task_Times, (void *)0, &TaskTimesStk[TASK_STK_SIZE-1], 7);
    OStaskCreate(Task_Rx, (void *)0, &TaskRxStk[TASK_STK_SIZE-1], 8);

    for (;;)
    {
        PC_DisPStr(16, 1, "Trabajo Final - uC/OS-II, The Real-Time Kernel",
            DISP_FGND_WHITE + DISP_BGND_RED + DISP_BLINK);
        PC_DisPStr(25, 27, "<-PRESS 'ESC' TO QUIT->", DISP_FGND_WHITE +
            DISP_BLINK);
        if (PC_GetKey(&key)== TRUE) /* See if key has been pressed */
        {
            if (key == 0x1B) /* Yes, see if it's the ESCAPE key */
            {
                PC_DOSReturn(); /* Yes, return to DOS */
            }
        }
        OSTimeDlyHMSM(0, 0, 0, 200);
    }
}
```

En la primera tarea se crean las demás tareas del μ C/OS-II[™]. En este caso son, la tarea que realiza la recepción de datos a través del puerto serie y última es una tarea

que realiza la limpieza de la pantalla. Una captura de la pantalla se observa en la Figura 5.2.

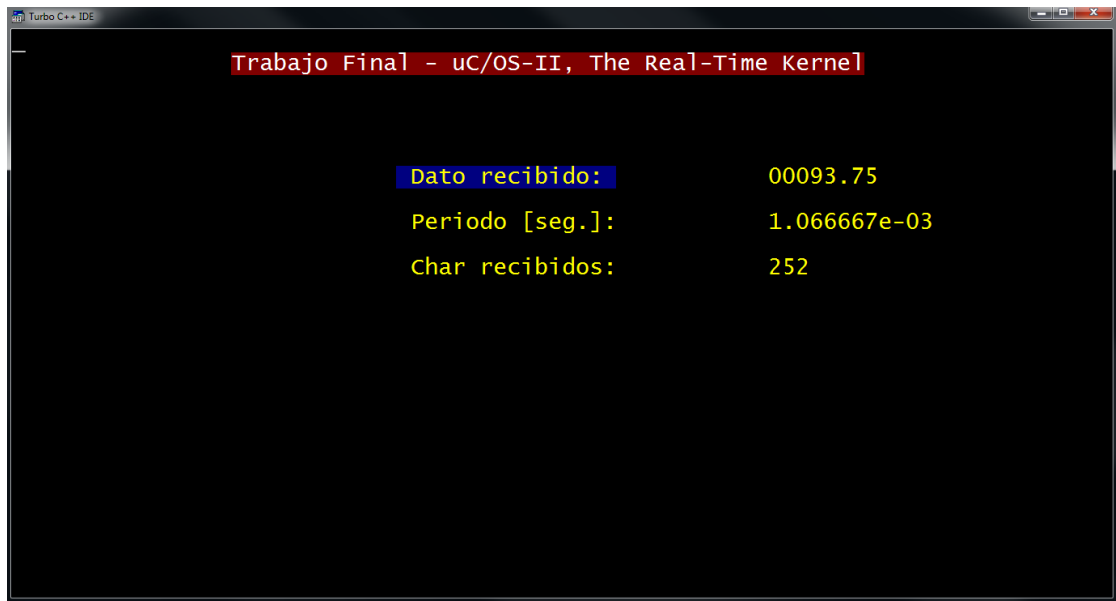


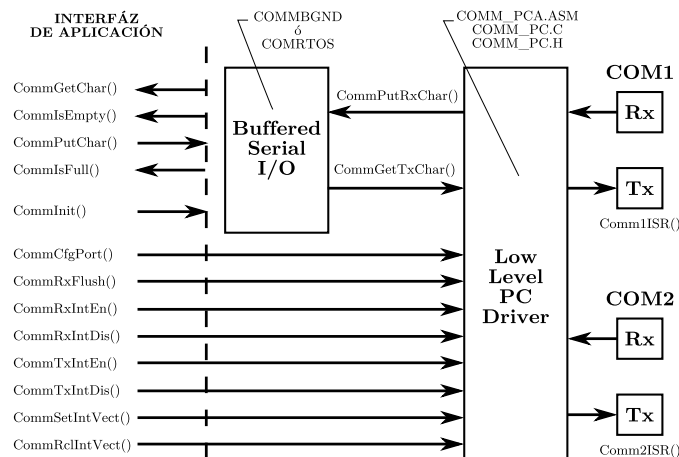
Figura 5.2: Captura de la pantalla del proyecto en funcionamiento.

5.1. Tarea de comunicación serial

Para manejar el interfaz serial de la PC se hace uso de un módulo proporcionado por el autor del $\mu\text{C}/\text{OS-II}^{\text{TM}}$, el *COMM_PC*. Este módulo hace mucho más fácil de usar el puerto serie. El código y sus funcionalidades del *driver* son fácilmente portable a diferentes arquitecturas y entornos de desarrollo. La estructura del módulo cuenta con dos bloques como se muestra en la Figura 5.3. El bloque *Low Level PC Driver* es responsable de interactuar con el *hardware*, es decir, la UART de la PC. Se provee de funciones a nuestras aplicaciones que nos permitan configurar los dos puertos (*COM1* ó *COM2*), habilitar/deshabilitar interrupciones de comunicación, y adquirir/cargar el vector de interrupción del puerto *COM*. Nuestras aplicaciones también interactúan a dos posibles tipos de *buffers* seriales, los módulos de I/O: *COMMBGND* ó *COMMRTOS*. Se puede usar el código del *COMMBGND* en aplicaciones *foreground/background* y el *COMMRTOS* si se está utilizando un *kernel* en tiempo real como el $\mu\text{C}/\text{OS-II}^{\text{TM}}$.

Se podría describir en detalle el funcionamiento del módulo *COMM_PC* pero no es la finalidad del presente informe. La documentación completa sobre este módulo se encuentra descrito en detalle en el libro *Embedded Systems Building Blocks, Second Edition* escrito por el *Jean J. Labrosse*.

La comunicación es unidireccional, por las especificaciones del proyecto, pero para realizar una comunicación bidireccional no ha de ser inconveniente. En el Código 5.3 antes de lanzar el bucle infinito de la tarea, se configura el puerto serie de la PC, *CommCfgPort(COMM2,9600,8,COMM_PARITY_NONE,1)*. Y se lanza las interrupciones del puerto *COMM2*. Ya en tiempo de ejecución, se espera a que el *buffer* del puerto serie se encuentre con datos a ser procesados y son almacenados temporalmente para

Figura 5.3: Diagrama en bloque del Módulo *COMM_PC*.

ser visualizados con los servicios de pantalla de la función *PC_DisPStr()*. Además en esta tarea se utiliza *MailBox*, recursos de comunicación interna entre tareas que ofrece el $\mu\text{C}/\text{OS-II}^{\text{TM}}$. Aquí el *MailBox DatoRecibidoMbox* envía la trama recibida a la tarea de visualización. Esta trama de datos está identificada por un caracter de comienzo ":" (en decimal 58), y uno de final de trama "n" (en decimal 10).

Código 5.3: Tarea de la recepción de datos en la PC (*Task_Rx()*)

```
void Task_Rx (void *data)
{
    INT8U error, errSem, errMbox;
    INT8U i, c;
    INT8U periodo[7], s[8];
    int k=0, kk=0, kdata = 0;
    char sdata[] = '0000.00';
    INT8U FlagInicio = 0;

    data = data;

    CommCfgPort(COMM2, 9600, 8, COMM_PARITY_NONE, 1);
    CommSetIntVect(COMM2);
    CommRxIntEn(COMM2);

    for(;;)
    {
        PC_DisPStr(28, 6, " Dato recibido: ", DISP_FGND_YELLOW + DISP_BGND_BLUE);
        PC_DisPStr(28, 8, " Periodo [seg.]: ", DISP_FGND_YELLOW);
        PC_DisPStr(28, 10, " Char recibidos: ", DISP_FGND_YELLOW);

        if(!CommIsEmpty(COMM2))
        {
            c = CommGetChar(COMM2, &error);
            if(error==COMM_NO_ERR)
            {
                switch(c)
                {
                    case 58:

```

```

        FlagInicio = 1;
        break;
    case 10:
        kdata = 0;
        FlagInicio = 0;
        errMbox = OSMboxPost(DatoRecibidoMbox, (void *)&sdata[0]);
        break;
    default:
        if(FlagInicio == 1)
        {
            k++;
            kdata++;
            sprintf(s, '%d', k);
            PC_DispStr(55, 10, s, DISP_FGND_YELLOW);
            sdata[kdata] = c;
        }
    }
}

OSTimeDlyHMSM(0, 0, 0, 100);
}
}

```

5.2. Tarea visualización de datos

Esta tarea es muy sencilla de interpretar. Aquí se espera a que se haya recibido una trama completa en la tarea *Task_Rx*, pues se espera por el *MailBox*. A la vez que se escriben los datos en la pantalla, se calcula el periodo en unidades de segundos y también es visualizada. .

Código 5.4: Tarea de limpieza de pantalla (*Task_DispData()*)

```

void Task_DispData(void *data)
{
    INT8U    err;
    INT8U    errSem, errMbox;
    char     s[40];
    char *PtrSData;
    float    Fdata, periodo;
    data = data;

    for (;;)
    {

        PtrSData = OSMboxPend(DatoRecibidoMbox, 0, &errMbox);

        PC_DispStr(55, 6, PtrSData, DISP_FGND_YELLOW);

        Fdata = atof(PtrSData);

        if(Fdata == 0.0)
            periodo = 0.0;
        else
            periodo = 0.1 / Fdata;
    }
}

```

```
    sprintf(s, '%e', periodo);
    PC_DispStr(55, 8, s, DISP_FGND_YELLOW);

    //PC_DispClrScr(DISP_FGND_WHITE);
    //clrscr();

    OSTimeDlyHMSM(0, 0, 0, 100);    /* Espera 1 Segundo */
}
}
```

Sección 6

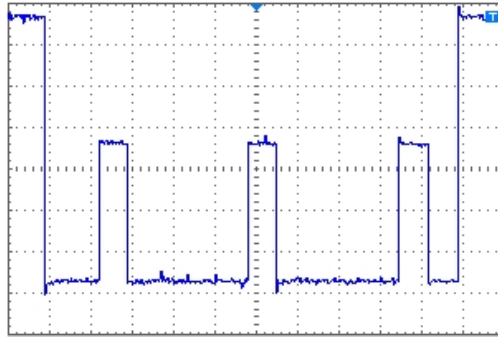
Ensayos

Los ensayos realizados sobre todo el sistema funcionando son principalmente sobre el sistema de comunicación entre el *Sensor* y el bloque de *Control*. La forma de las señales sobre el canal serial en el caso ideal se observan en la Figura 3.5. Para realizar el testeo del sistema se ha utilizado un *Dimmer* que nos permitirá modificar la intensidad de luz que incide sobre el sensor fotoeléctrico. La Tabla 6.1 presenta varias mediciones realizadas a diferentes niveles de atenuación del *Dimmer*. Se contrasta el periodo adquirido por el dsPIC® (segunda columna) y la medición del mismo con un osciloscopio. (tercera columna).

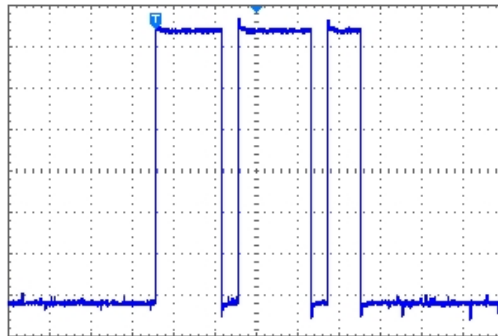
Muestra	Periodo	
	dsPIC®	Medido
Muestra 1	3.5	368 μ seg.
Muestra 2	29.5	2873 μ seg.
Muestra 3	85	8622 μ seg.
Muestra 4	120	12.1 μ seg.

Tabla 6.1: Captura de datos del sistema de sensor a diferentes niveles de intensidad de luz.

Por último se muestran las capturas desde el osciloscopio para las diferentes muestras presentadas en la Tabla 6.1.

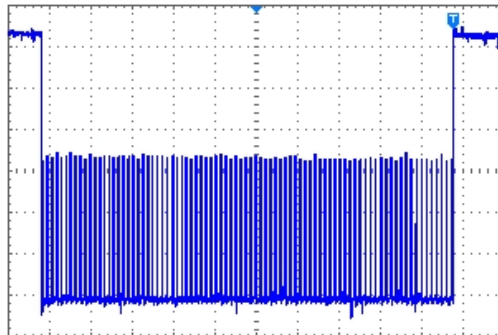


(a) Señales tomadas sobre la línea de comunicación del sensor/control.

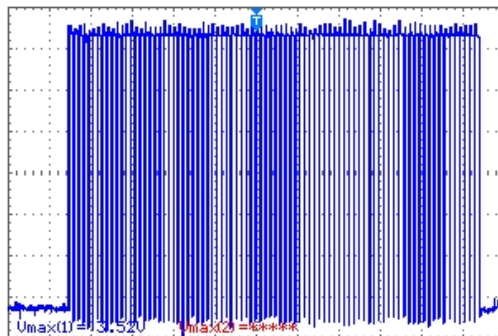


(b) Señales tomadas en la salida del sistema de control (colector Q2).

Figura 6.1: Muestra 1.

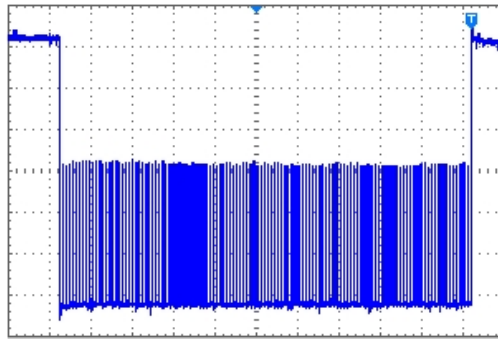


(a) Señales tomadas sobre la línea de comunicación del sensor/control.

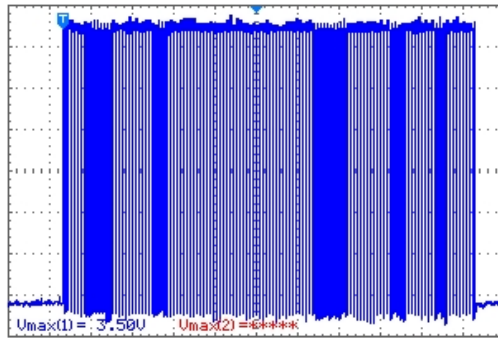


(b) Señales tomadas en la salida del sistema de control (colector Q2).

Figura 6.2: Muestra 2.



(a) Señales tomadas sobre la línea de comunicación del sensor/control.



(b) Señales tomadas en la salida del sistema de control (colector Q2).

Figura 6.3: Muestra 3.

Apéndice A

Repositorio del proyecto

El proyecto se encuentra bajo en control de versiones proporcionado por el *software Subversion*, conocido también con las siglas *SVN*. Este proyecto como su código fuente y documentación se encuentra publicado bajo *Licencias Libre* para su uso y modificación. Para acceder al proyecto, bajo sistemas operativos GNU/Linux & Unix, simplemente se corre el comando,

```
svn checkout http://proyecto5r1.googlecode.com/svn/branches/ProyectoSensorConUCOSII/firmware proyectosensorconucosii -read-only
```

El proyecto es una ramificación de un proyecto ya realizado por los estudiantes *Andrés Hoc* y *Gonzalo Vassia* quienes han decidido publicar su proyecto también en forma libre. Para conocer más sobre el manejo de repositorios *SVN* y como se realizan proyectos a partir de una raíz (*trunk*) acceda a la documentación de la siguiente página web: <http://svnbook.red-bean.com/en/1.6/svn.intro.whatis.html>.

Apéndice B

Códigos del proyecto

Se adjuntan los códigos fuentes de los archivos más importantes del proyecto. De todas formas este puede ser accedido desde el repositorio.

../v0.1/app.c

```
/*
*****
*
*                               Luis Alberto Guanuco
*                               Proyecto Final
*                               Ctedra Software en Tiempo Real
*
*                               UTN - Facultad Regional Cordoba
*
* File : APP.C
*
*****
*/

#include <includes.h>
#include <globals.h>

/*
*****
*                               CONSTANTS
*****
*/

/*
*****
*                               VARIABLES
*****
*/

OS_STK  tareaInicioStk[TAREA_INICIO_STK_SIZE];
OS_STK  tareaLuisTestStk[TAREA_LUISTEST_STK_SIZE];
OS_STK  tareaLuisTestTwoStk[TAREA_LUISTESTTWO_STK_SIZE];
OS_STK  tareaLuisTestThreeStk[TAREA_LUISTESTTHREE_STK_SIZE];
OS_STK  DIOTaskStk[DIO_TASK_STK_SIZE];

uint8_t errorStkChk;

/*
*****
*                               FUNCTION PROTOTYPES
*****
*/

static void TareaInicio(void *p_arg);
static void AppTaskCreate(void);
static void TareaLuisTest(void);
static void TareaLuisTestTwo(void);
static void TareaLuisTestThree(void);

/*
*****
*                               main()
*
* Description : This is the standard entry point for C code.
* Arguments   : none
*****
*/

CPU_INT16S main (void)
```



```

{
    CPU_INT08U err;

    BSP_IntDisAll(); /* Disable all interrupts until we are ready
                       to accept them */

    RCON = 0;

    OSInit(); /* Initialize 'uC/OS-II, The Real-Time Kernel'
               */
    DIOInit();

    OSTaskCreateExt(TareaInicio,
                    (void *)0,
                    (OS_STK *)&tareaInicioStk[0],
                    TAREA_INICIO_PRIO,
                    TAREA_INICIO_PRIO,
                    (OS_STK *)&tareaInicioStk[TAREA_INICIO_STK_SIZE-1],
                    TAREA_INICIO_STK_SIZE,
                    (void *)0,
                    OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);

#ifdef OS_TASK_NAME_SIZE > 1
    OSTaskNameSet(TAREA_INICIO_PRIO, (CPU_INT08U *)"Start Task", &err);
#endif

    OSStart(); /* Start multitasking (i.e. give control to uC
               /OS-II) */

    return (-1); /* Return an error - This line of code
                 is unreachable */
}

/*
*****
*
* STARTUP TASK
*
* Description : This is an example of a startup task. As mentioned in the book's text, you MUST
*               initialize the ticker only once multitasking has started.
*
* Arguments : p_arg is the argument passed to 'TareaInicio()' by 'OSTaskCreate()'.
*
* Notes : 1) The first line of code is used to prevent a compiler warning because 'p_arg' is not
*           used. The compiler should not generate any code for this statement.
*          2) Interrupts are enabled once the task start because the I-bit of the CCR register was
*           set to 0 by 'OSTaskCreate()'.
*****
*/

static void TareaInicio (void *p_arg)
{
    CPU_INT08U i;
    CPU_INT08U j;

    (void)p_arg;

    BSP_Init(); /* Initialize BSP functions
                 */

    #if OS_TASK_STAT_EN > 0
        OSStatInit(); /* Determine CPU capacity
                       */
    #endif

    AppTaskCreate(); /* Create additional user tasks
                     */

    //Se crea un MailBox para indicar cuando hay un nuevo periodo de velocidad
    mBoxPromData= OSMboxCreate((void *) NULL);
    mBoxPromData->OSEventPtr = 0;

    while (DEF_TRUE)
    { /* Task body, always written as an infinite loop.
      //Chequeamos el tamaño de Stack de esta tarea
      //errorStkChk = OSTaskStkChk(TAREA_INICIO_PRIO, &tareaInicioStk);
      OSTimeDly(100);
    }
} //Fin TareaInicio ()

/*
*****
*
* Tarea LuisTest
*
* Description : Tarea para testar el proyecto
* Arguments : p_arg is the argument passed to 'TareaInicio()' by 'OSTaskCreate()'.
* Notes :
*****
*/

static void TareaLuisTest (void)
{
    // Inicializacin del mdulo DIO

```

```

        DOCfgMode(0, DO_MODE_DIRECT, 0);
        DICfgMode(0, DI_MODE_EDGE_HIGH_GOING);
        ValChZero = 0;
        CntData = 0;
        DIClr(0);

while(1)
{
    //Chequeamos el tamaño de Stack de esta tarea
    //errorStkChk = OSTaskStkChk(TAREA_LUISTEST_Prio, &tareaLuisTestStk);
    DOSet(0, TRUE);
    OSTimeDlyHMSM(0, 0, 0, 100);
    DOSet(0, FALSE);
    OSTimeDlyHMSM(0, 0, 1, 0);

    ValChZero += DIGet(0);
    CntData++;
    DIClr(0);

    OSTimeDly(100);
}
} //Fin TareaLuisTest()

/*
*****
*                               Tarea LuisTestTwo
* Description : Tarea para testar el proyecto
* Arguments   : p_arg is the argument passed to 'TareaInicio()' by 'OSTaskCreate()'.
* Notes      :
*****
*/
static void TareaLuisTestTwo (void)
{
    TRISBbits.TRISB5 = 0;

    while(1)
    {
        //Chequeamos el tamaño de Stack de esta tarea
        //errorStkChk = OSTaskStkChk(TAREA_LUISTESTTWO_Prio, &tareaLuisTestTwoStk);
        if(CntData!=0)
        {
            Prom = (float) ValChZero / (float) CntData;
            ValChZero = 0;
            CntData = 0;
        }
        else
            Prom = 0;

        OSMBboxPost(mBoxPromData, &Prom); //Enviamos un mensaje a la funcion que transmite el dato (promedio
        de los pulsos recibidos
        OSTimeDlyHMSM(0, 0, 4, 0);
    }
} //Fin TareaLuisTest()

/*
*****
*                               Tarea LuisTestThree
* Description : Tarea para testar el proyecto
* Arguments   : p_arg is the argument passed to 'TareaInicio()' by 'OSTaskCreate()'.
* Notes      :
*****
*/
static void TareaLuisTestThree (void)
{
    uint8_t errorMboxPend;

    asm volatile ('mov #OSCCONL, w1 \n'
        'mov #0x46, w2 \n'
        'mov #0x57, w3 \n'
        'mov.b w2, [w1] \n'
        'mov.b w3, [w1] \n'
        'bclr OSCCON, #6');

    RPINR18bits.U1RXR = 22;
    RPOR11bits.RP23R = 0b00011;

    asm volatile ('mov #OSCCONL, w1 \n'
        'mov #0x46, w2 \n'
        'mov #0x57, w3 \n'
        'mov.b w2, [w1] \n'
        'mov.b w3, [w1] \n'
        'bset OSCCON, #6');

    InitUART1();

while(1)

```

```

{
    //Chequeamos el tamaño de Stack de esta tarea
    //errorStkChk = OSTaskStkChk(TAREA_LUISTESTTHREE_PRIO, &tareaLuisTestThreeStk);

    PtrPromFromMBox = OSMboxPend(mBoxPromData, TIMEOUT_PEND_MBOX, &errorMboxPend); //Esperamos a que
    //llegue un nuevo promedio
    PromFromMbox = *PtrPromFromMBox;
    whole=(int) PromFromMbox ;
    //obtains whole part
    decimal=(int) ((PromFromMbox - (float) whole)*100.0); //obtains decimal part (1 digit)
    sprintf(array,':%04d.%02d\n',whole,decimal); //converts to string
    for(iiTaskUart = 0; iiTaskUart<9 ; iiTaskUart++)
    {
        while(U1STAbits.UTXBF != 0);
        U1TXREG = array[iiTaskUart];
    }

    OSTimeDlyHMSM(0,0,1,0);
}
} //Fin TareaLuisTest()

```

```

/*
*****
*                               CREATE ADDITIONAL APPLICATION TASKS
*****
*/

```

```

static void AppTaskCreate (void)
{
    CPU_INT08U err;
    OSTaskCreateExt(TareaLuisTest,
        (void *)0,
        (OS_STK *)&tareaLuisTestStk[0],
        TAREA_LUISTEST_PRIO,
        TAREA_LUISTEST_PRIO,
        (OS_STK *)&tareaLuisTestStk[TAREA_LUISTEST_STK_SIZE-1],
        TAREA_LUISTEST_STK_SIZE,
        (void *)0,
        OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR | OS_TASK_OPT_SAVE_FP);

    OSTaskCreateExt(TareaLuisTestTwo,
        (void *)0,
        (OS_STK *)&tareaLuisTestTwoStk[0],
        TAREA_LUISTESTTWO_PRIO,
        TAREA_LUISTESTTWO_PRIO,
        (OS_STK *)&tareaLuisTestTwoStk[TAREA_LUISTESTTWO_STK_SIZE-1],
        TAREA_LUISTESTTWO_STK_SIZE,
        (void *)0,
        OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR | OS_TASK_OPT_SAVE_FP);

    OSTaskCreateExt(TareaLuisTestThree,
        (void *)0,
        (OS_STK *)&tareaLuisTestThreeStk[0],
        TAREA_LUISTESTTHREE_PRIO,
        TAREA_LUISTESTTHREE_PRIO,
        (OS_STK *)&tareaLuisTestThreeStk[TAREA_LUISTESTTHREE_STK_SIZE-1],
        TAREA_LUISTESTTHREE_STK_SIZE,
        (void *)0,
        OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR | OS_TASK_OPT_SAVE_FP);
}

```

../v0.1/app_cfg.h

```

/*
*****
*                               uC/OS-II Application Configuration
*
*                               DO NOT DELETE THIS FILE, IT IS REQUIRED FOR OS_VER > 2.80
*
*                               CHANGE SETTINGS ACCORDINGLY
*
*
* File : app_cfg.h
* By   : Eric Shufro
*****
*/

#ifndef APP_CFG_H
#define APP_CFG_H

/*
*****
*                               INCLUDES
*****
*/

```

```
#include <lib_def.h>

/*
*****
*
*      ADDITIONAL uC/MODULE ENABLES
*
*****
*/

#define LIB_STR_CFG_FP_EN          DEF_DISABLED

/*
*****
*
*      TASK PRIORITIES
*
*****
*/

#define TAREA_INICIO_PRIO          0          /* Lower numbers are of
        higher priority          */
#define TAREA_LUISTEST_PRIO          5
#define TAREA_LUISTESTTWO_PRIO      6
#define TAREA_LUISTESTTHREE_PRIO    7
#define TAREA_LUISTESTFOUR_PRIO     8

/*
*****
*
*      TASK STACK SIZES
*
* Notes :  1) Warming, setting a stack size too small may result in the OS crashing. It the OS crashes
*            within a deep nested function call, the stack size may be to blame. The current maximum
*            stack usage for each task may be checked by using uC/OS-View or the stack checking
*            features of uC/OS-II.
*****
*/

#define TAREA_INICIO_STK_SIZE          126
#define TAREA_LUISTEST_STK_SIZE        253
#define TAREA_LUISTESTTWO_STK_SIZE     253
#define TAREA_LUISTESTTHREE_STK_SIZE   253

#endif
/* End of file
*/
```

../v0.1/dsPIC_a.s

```
;
; *****
;
;      uC/OS-II
;      The Real-Time Kernel
;
;      (c) Copyright 2002, Jean J. Labrosse, Weston, FL
;      All Rights Reserved
;
;
;      dsPIC33FJ Board Support Package
;
;
; File      : bsp_a.s
; By       : Eric Shufro
; *****
;
;
; *****
;
;      CONSTANTS
;
; *****
;
; .equ      __33FJ256GP710, 1          ; Inform the p33FG256GP710 header file that we are using a
;      dsPIC33FJ256GP710
; .equ      __33FJ128GP804, 1          ; Inform the p33FG256GP710 header file that we are using a
;      dsPIC33FJ128GP804
;
; *****
;
;      INCLUDES
;
; *****
;
; .include 'p33FJ256GP710.inc'          ; Include assembly equates for various CPU registers and bit
;      masks
; .include 'p33FJ128GP804.inc'          ; Include assembly equates for various CPU registers and bit
;      masks
; .include 'os_cpu_util_a.s'            ; Include an assembly utility files with macros for saving and
;      restoring the CPU registers
;
; *****
;
;      LINKER SPECIFICS
;
; *****
;
```

```
.text                                ; Locate this file in the text region of the
    build

;
;*****
;                                     GLOBALS
;*****
;

.global __T2Interrupt
.global __T4Interrupt


;
;*****
;                               OS Time Tick ISR Handler
;*****
; Description : This function services the OS Time Tick Interrupt when configured using Timer #2
; Notes       : All user interrupts should be defined as follows.
;*****
__T2Interrupt:
    OS_REGS_SAVE                    ; 1) Save processor registers

    mov     _OSIntNesting, w1
    inc.b   [w1], [w1]              ; 2) Call OSIntEnter() or increment
        OSIntNesting

    dec.b   _OSIntNesting, wreg      ; 3) Check OSIntNesting. if OSIntNesting ==
        1, then save the stack pointer, otherwise jump to T2_Cont

    bra nz, T2_Cont
    mov     _OSTCBCur, w0
    mov     w15, [w0]

T2_Cont:
    call    _OS_Tick_ISR_Handler    ; 4) Call YOUR ISR Handler (May be a C
        function). In this case, the OS Tick ISR Handler

    call    _OSIntExit             ; 5) Call OSIntExit() or decrement 1 from
        OSIntNesting

    OS_REGS_RESTORE                ; 6) Restore registers

    retfie                          ; 7) Return from interrupt

;
;*****
;                               OS Time Tick ISR Handler
;*****
; Description : This function services the OS Time Tick Interrupt when configured using Timer #4
; Notes       : All user interrupts should be defined as follows.
;*****
__T4Interrupt:
    OS_REGS_SAVE                    ; 1) Save processor registers

    mov     _OSIntNesting, w1
    inc.b   [w1], [w1]              ; 2) Call OSIntEnter() or increment
        OSIntNesting

    dec.b   _OSIntNesting, wreg      ; 3) Check OSIntNesting. if OSIntNesting ==
        1, then save the stack pointer, otherwise jump to T2_Cont

    bra nz, T4_Cont
    mov     _OSTCBCur, w0
    mov     w15, [w0]

T4_Cont:
    call    _OS_Tick_ISR_Handler    ; 4) Call YOUR ISR Handler (May be a C
        function). In this case, the OS Tick ISR Handler

    call    _OSIntExit             ; 5) Call OSIntExit() or decrement 1 from
        OSIntNesting

    OS_REGS_RESTORE                ; 6) Restore registers

    retfie                          ; 7) Return from interrupt



./../v0.1/dsPIC_cfg.c

/*
*****
MeDef
Mecedor de Eficiencia
*/
File : dsPIC_cfg.C
```

```

*
*****
*/

#include <includes.h>

/*
*****
*                                     MPLAB CONFIGURATION MACROS
*****
*/

    /*__CONFIG(FWDT, WDTDIS);                                     /* Disable the watchdog timer
    /*__CONFIG(FOSCSEL, OSCPLL);                                   /* Enable the primary XT / HS oscillator
        with PLL
    */

    /*_FOSC( POSCMD_NONE & OSCIOFNC_OFF)
    /*_FOSCSEL( FNOSC_FRCPLL & IESO_OFF);

    _FWDT( FWDTEN_OFF & WDTPRE_PR32 & WDTPOST_PS1 );
    _FOSC( POSCMD_HS & OSCIOFNC_OFF );
    _FOSCSEL( FNOSC_PRIPLL & IESO_OFF );

/*
*****
*                                     CONSTANTS
*****
*/

/*
*****
*                                     VARIABLES
*****
*/

/*
*****
*                                     PROTOTYPES
*****
*/

static void PLL_Init(void);
static void Tmr_TickInit(void);

/*
*****
*                                     BSP INITIALIZATION
*
* Description : This function should be called by your application code before you make use of any of the
*               functions found in this module.
*
* Arguments   : none
*****
*/

void BSP_Init (void)
{
    RCON    &= ~SWDTEN;                                           /* Ensure Watchdog disabled via IDE CONFIG
        bits and SW.
    */

    PLL_Init();
    Tmr_TickInit();
    /*
    */
}

/*
*****
*                                     PLL_Init()
*
* Description : This function configures and enables the PLL with the external oscillator
*               selected as the input clock to the PLL.
*
* Notes       : 1) The PLL output frequency is calculated by  $FIN * (M / (N1 * N2))$ .
*               2)  $FIN$  is the PLL input clock frequency, defined in bsp.h as
*                  CPU_PRIMARY_OSC_FR. This is the same as the external primary
*                  oscillator on the Explorer 16 Evaluation Board.
*               3)  $M$  is the desired PLL multiplier
*               4)  $N1$  is the divider for  $FIN$  before  $FIN$  enters the PLL block (Pre-Divider)
*               5)  $N2$  is the PLL output divider (Post-Divider)
*
* Summary     : The PLL is configured as  $(8MHz) * (40 / (2 * 2)) = 80MHz$ 
*               The processor clock is (1/2) of the PLL output.
*               Performance = 40 MIPS.
*****
*/

static void PLL_Init (void)
{

```

```

//Frecuencia del Cristal = Fin = 10 MHz
//Ajustamos el PLL para tener Fosc = 79.608.000 Hz
//Fcy = Fosc/2 = 39.804.000 Hz
//Tcy = 1/Fcy = 25.123 ns
CLKDIVbits.PLLPRE = 8; //Dividimos en 10 la frecuencia del cristal
DelayTcy(10);
PLLFBDbits.PLLDIV = 158; //PLLFBDbits.PLLDIV = 158; //Multiplicamos por M=160
DelayTcy(10);
CLKDIVbits.PLLPOST = 0x00; //Dividimos por 2
DelayTcy(10);
CLKDIVbits.DOZE = 0x00;

//PLLFBD = 38; //Set the Multiplier (M) to
//CLKDIV = 40 (2 added automatically) //Clear the PLL Pre Divider bits, N1 = N2
//CLKDIV = 2;

}

/*
*****
* BSP_CPU_ClkFrq()
*
* Description : This function determines the CPU clock frequency (Fcy)
* Returns : The CPU frequency in (HZ)
*****
*/

CPU_INT32U BSP_CPU_ClkFrq (void)
{
    CPU_INT08U Clk_Selected;
    CPU_INT08U PLL_n1;
    CPU_INT08U PLL_n2;
    CPU_INT16U PLL_m;
    CPU_INT16U FRC_Div;
    CPU_INT32U CPU_Clk_Frq;

    PLL_m = (PLLFBD & PLLDIV_MASK) + 2; //Get the Multiplier value
    PLL_n1 = (CLKDIV & PLLPRE_MASK) + 2; //Computer the Pre Divider value
    PLL_n2 = ((CLKDIV & PLLPOST_MASK) >> 6); //Get the Post Divider register value
    PLL_n2 = ((PLL_n2 * 2) + 2); //Compute the Post Divider value

    FRC_Div = ((CLKDIV & FRCDIV_MASK) >> 8); //Get the FRC Oscillator Divider register
    FRC_Div = ((1 << FRC_Div) * 2); //Compute the FRC Divider value

    Clk_Selected = (OSCCON & COSC_MASK) >> 12; //Determine which clock source is currently
    selected

    switch (Clk_Selected) {
        case 0: //Fast Oscillator (FRC) Selected
            CPU_Clk_Frq = CPU_FRC_OSC_FRQ; //Return the frequency of the internal fast
            oscillator
            break;

        case 1: //Fast Oscillator (FRC) with PLL Selected
            CPU_Clk_Frq = ((CPU_FRC_OSC_FRQ * PLL_m) / (FRC_Div * PLL_n1 * PLL_n2)); //Compute the PLL output frequency using
            the FRC as FIN
            break;

        case 2: //Primary External Oscillator Selected
            CPU_Clk_Frq = CPU_PRIMARY_OSC_FRQ; //Return the frequency of the primary
            external oscillator
            break;

        case 3: //Primary External Oscillator with PLL
            CPU_Clk_Frq = ((CPU_PRIMARY_OSC_FRQ * PLL_m) / (PLL_n1 * PLL_n2)); //Compute the PLL output frq using the PRI
            EXT OSC as FIN
            break;

        case 4: //Secondary External Low Power Oscillator (
            SOSC) Selected
            break;

        case 5: //Internal Low Power RC Oscillator Selected
            CPU_Clk_Frq = CPU_SECONDARY_OSC_FRQ; //Return the frq of the external secondary
            Low Power OSC
            break;

        case 6:
    }
}

```

```

        CPU_Clk_Frq = 0;                                /* Return 0 for the Reserved clock setting
        break;                                          */

    default:
        CPU_Clk_Frq = 0;                                /* Return 0 if the clock source cannot be
        determined                                     */
        break;
    }

    CPU_Clk_Frq /= 2;                                    /* Divide the final frq by 2, get the actual
    CPU_Frq (Fcy) */

    return (CPU_Clk_Frq);                                /* Return the operating frequency
    */
}

/*
*****
*                                     DISABLE ALL INTERRUPTS
*
* Description : This function disables all interrupts from the interrupt controller.
*
* Arguments   : none
*****
*/

void BSP_IntDisAll (void)
{
    SRbits.IPL = 0b111;
}

/*
*****
*                                     TICKER INITIALIZATION
*
* Description : This function is called to initialize uC/OS-II's tick source (typically a timer generating
*               interrupts every 1 to 100 mS).
*
* Arguments   : none
*
* Note(s)     : 1) The timer operates at a frequency of Fosc / 4
*               2) The timer resets to 0 after period register match interrupt is generated
*****
*/

static void Tmr_TickInit (void)
{
    CPU_INT32U tmr_frq;
    CPU_INT16U cnts;

    //tmr_frq = BSP_CPU_ClkFrq();                        /* Get the CPU Clock Frequency (Hz) (Fcy)
    //tmr_frq = 39804000;                                */
    //cnts = (tmr_frq / OS_TICKS_PER_SEC) - 1;           /* Calaculate the number of
    timer ticks between interrupts */

    #if BSP_OS_TMR_SEL == 2
        T2CON = 0;                                       /* Use Internal Osc (Fcy), 16 bit mode,
        prescaler = 1                                    */
        TMR2 = 0;                                       /* Start counting from 0 and clear the
        prescaler count                                  */
        T2CONbits.TCKPS = 0;                             /*Prescaler divide por 8
        //PR2 = cnts;                                    */
        //PR2 = 2498;                                    /*
        //luisPR2 = 39996;                                //Pres = 8 => 1000.3 ticks por segundo
        PR2 = 3980;                                       //Pres = 1 => 1000.0 ticks por segundo
        IPC1 &= ~T2IP_MASK;                             //luis para 10.000 ticks per sec
        priority bits                                     /* Clear all timer 2 interrupt
        IPC1 |= (TIMER_INT_PRIO << 12);                 */
        IFS0 &= ~T2IF;                                   /* Set timer 2 to operate with an interrupt
        IEC0 |= T2IE;                                     /* Clear the interrupt for timer 2
        T2CON |= TON;                                     /* Enable interrupts for timer 2
        */
        */
        /* Start the timer
    #endif

    #if BSP_OS_TMR_SEL == 4
        T4CON = 0;                                       /* Use Internal Osc (Fcy), 16 bit mode,
        prescaler = 1                                    */
        TMR4 = 0;                                       /* Start counting from 0 and clear the
        prescaler count                                  */
        PR4 = cnts;                                       /* Set the period register
        */
        IPC6 &= ~T4IP_MASK;                             /* Clear all timer 4 interrupt priority bits
        */
        IPC6 |= (TIMER_INT_PRIO << 12);                 /* Set timer 4 to operate with an interrupt
        priority of 4 */
    
```



```

    IFS1    &=  ~T4IF;                /* Clear the interrupt for timer 4
    IEC1    |=  T4IE;                /* Enable interrupts for timer 4
    T4CON    |=  TON;                /* Start the timer
                                   */
#endif
}

/*
*****
*                                     OS TICK INTERRUPT SERVICE ROUTINE
*
* Description : This function handles the timer interrupt that is used to generate TICKs for uC/OS-II.
*****
*/

void OS_Tick_ISR_Handler (void)
{
    #if BSP_OS_TMR_SEL == 2
        IFS0 &= ~T2IF;
    #endif

    #if BSP_OS_TMR_SEL == 4
        IFS1 &= ~T4IF;
    #endif

    OSTimeTick();
}

../v0.1/dsPIC_cfg.h

/*
*****
*                                     Microchip PIC33
*                                     Board Support Package
*
*                                     Micrium
*                                     (c) Copyright 2005, Micrium, Weston, FL
*                                     All Rights Reserved
*
* File : BSP.H
* By : Eric Shufro
*****
*/

/*
*****
*                                     OSCILLATOR FREQUENCIES
*****
*/

#define CPU_PRIMARY_OSC_FRQ      8000000L    /* Primary External Oscillator Frequency
*/
#define CPU_FRC_OSC_FRQ        7370000L    /* Internal Fast Oscillator Frequency
*/
#define CPU_SECONDARY_OSC_FRQ   32768L      /* Secondary External Oscillator
Frequency
*/

/*
*****
*                                     OS TICK TIMER SELECTION
*****
*/

#define BSP_OS_TMR_SEL          2            /* Select a timer for the OS Tick
Interrupt (2 or 4)
*/
#define TIMER_INT_PRIO          4            /* Configure the timer to use interrupt
priority 4
*/
#define PROBE_INT_PRIO          4            /* Configure UART2 Interrupts to use
priority 4
*/

/*
*****
*                                     DATATYPES
*****
*/

typedef void (*PFUNCT)(void);

/*
*****
*                                     MACROS
*****
*/

/*
*****
*                                     CHIP SPECIFIC MACROS
*****

```

```

*****
*/
/* OSCCON register bits
*/

#define XT_HS_EC_PLL_SEL    (CPU_INT16U) (3 << 0)
#define COSC_MASK          (CPU_INT16U) (7 << 12)
#define LOCK               (CPU_INT16U) (1 << 5)
#define OSWEN              (CPU_INT16U) (1 << 0)

/* CLKDIV register bits
*/

#define FRCDIV_MASK        (CPU_INT16U) (7 << 8)
#define PLLPOST_MASK      (CPU_INT16U) (3 << 6)
#define PLLPRE_MASK       (CPU_INT16U) (0x1F << 0)
#define PLLDIV_MASK       (CPU_INT16U) (0xFF << 0)

/* Timer Control register bits
*/

#define TON                (CPU_INT16U) (1 << 15)

/* IPC1 Interrupt Priority register bits
*/
#define T2IP_MASK          (CPU_INT16U) (7 << 12)

/* IPC5 Interrupt Priority register bits
*/
#define T4IP_MASK          (CPU_INT16U) (7 << 12)

/* IPC7 Interrupt Priority register bits
*/
#define U2TXIP_MASK        (CPU_INT16U) (7 << 12)
#define U2RXIP_MASK        (CPU_INT16U) (7 << 8)

/* IEC0 Interrupt Enable register bits
*/
#define T2IE               (CPU_INT16U) (1 << 7)

/* IEC1 Interrupt Enable register bits
*/
#define T4IE               (CPU_INT16U) (1 << 11)
#define U2TXIE             (CPU_INT16U) (1 << 15)
#define U2RXIE             (CPU_INT16U) (1 << 14)

/* IFS0 Interrupt Flag register bits
*/
#define T2IF               (CPU_INT16U) (1 << 7)

/* IFS1 Interrupt Flag register bits
*/
#define T4IF               (CPU_INT16U) (1 << 11)
#define U2TXIF             (CPU_INT16U) (1 << 15)
#define U2RXIF             (CPU_INT16U) (1 << 14)

/* UxMODE register bits
*/
#define UART_EN            (CPU_INT16U) (1 << 15)

/* UxSTA register bits
*/
#define UTXISEL            (CPU_INT16U) (1 << 15)
#define UTXEN              (CPU_INT16U) (1 << 10)
#define TRMT               (CPU_INT16U) (1 << 8)
#define URXDA              (CPU_INT16U) (1 << 0)

/* System Control register bits
*/
#define SWDTEN             (CPU_INT16U) (1 << 5)

/* Interrupt Configuration register 1 bits
*/
#define NSTDIS             (CPU_INT16U) (1 << 15)

/*
*****
*
* FUNCTION PROTOTYPES
*
*****
*/

void BSP_Init(void);
void BSP_IntEn(CPU_INT08U IntCont, CPU_INT08U IntNum, CPU_INT08U IntPol, CPU_INT08U IntAct, PFNCT pfncnt);
void BSP_IntDis(CPU_INT08U IntCont, CPU_INT08U IntNum);
void BSP_IntDisAll(void);

CPU_INT32U BSP_CPU_ClkFrq(void);

/*
*****
*
* LED SERVICES
*
*****
*/

//void LED_Init(void);
//void LED_On(CPU_INT08U led);
//void LED_Off(CPU_INT08U led);
//void LED_Toggle(CPU_INT08U led);

/*
*****
*
* TICK SERVICES
*
*****
*/

void Tmr_TickISR_Handler(void);

```

../v0.1/dsPIC_delay.c

```

// */
void Delay_100usec()
{
    char i;
    for(i=0;i<39;i++)
        Nop();
}

```

```

    }

/*Funcin Delay_x100useg
-----

Descripcin: Esta funcin realiza una demora de, ms o menos, x*100 useg. Es muy imprecisa, jeje
Entrada: x: Cantidad de veces que se repite la demora de 100 useg
Salida: nada
//-----
*/
void Delay_x100useg(int x)
{
    int i;
    while(x!=0)
    {
        for(i=0;i<37;i++)
            Nop();
        x--;
    }
}

```

../v0.1/dsPIC_delay.h

```

#ifndef DELAY_H
#define DELAY_H

//INCLUDES
#include 'p30fxxx.h'
#include 'os_cpu.h'

//PROTOTIPOS DE FUNCIONES

/*Funcin DelayTcy
-----

Descripcin: Esta funcin realiza una demora de 'ciclos' ciclos
Entrada:
    INT16U ciclos: nmero de ciclos de demora
Salida: nada
//-----
*/
void DelayTcy(INT16U ciclos);

/*Funcin Delay_3_6useg
-----

Descripcin: Esta funcin realiza una demora de, ms o menos, 3,6 useg.
Entrada: nada
Salida: nada
//-----
*/
void Delay_3_6useg();

/*Funcin Delay_10useg
-----

Descripcin: Esta funcin realiza una demora de, ms o menos, 10 useg.
Entrada: nada
Salida: nada
//-----
*/
void Delay_10useg();

/*Funcin Delay_100useg
-----

Descripcin: Esta funcin realiza una demora de, ms o menos, 100 useg.
Entrada: nada
Salida: nada
//-----
*/
void Delay_100useg();

/*Funcin Delay_x100useg
-----

Descripcin: Esta funcin realiza una demora de, ms o menos, x*100 useg. Es muy imprecisa, jeje
Entrada: x: Cantidad de veces que se repite la demora de 100 useg
Salida: nada
//-----
*/
void Delay_x100useg(int x);

#endif

```

../v0.1/globals.c

```

#include 'globals.h'

```

```

//VARIABLES GLOBALES
uint8_t err;

float TmpCnt;

OS_EVENT      *mBoxPromData; //Manejador del MailBox que significa que un nuevo periodo ha sido leído y debe
                        almacenarse en el buffer

//DIO variable
uint16_t ValChZero, CntData = 0;
float Prom = 0;
float *PtrPromFromMBox;
float PromFromMbox;
uint16_t whole = 0;
uint16_t decimal = 0;
char array[9];
float TmpCnt;
uint8_t iiTaskUart;

../../v0.1/globals.h

#ifndef GLOBALS_H
#define GLOBALS_H

//INCLUDES

#include <dsPIC_delay.h>
#include <p33FJ128GP804.h>
#include <stdint.h>
#include <cpu.h>
#include <includes.h>

//DEFINES RELATIVOS A LAS BASES DE TIEMPO
#define TOSC 0.000000203021 //0.000000040200
#define TCY ((float) 4 * (float) TOSC)
#define FCY 1000000UL
//define BASE_T0 (TCY * 10000)

//DEFINES GENERALES
//Defines utilizados para hacer demoras
#define NOP5 Nop();Nop();Nop();Nop();Nop();
#define NOP10 NOP5 NOP5
#define NOP15 NOP10 NOP5
#define NOP20 NOP10 NOP10
#define NOP30 NOP20 NOP10
#define NOP40 NOP20 NOP20
#define NOP50 NOP40 NOP10
#define NOP100 NOP50 NOP50

#ifndef NULL
#define NULL 0
#endif

//define Sleep(); _asm SLEEP _endasm
//define Reset(); _asm RESET _endasm

//define PIN_POWER_ON_OFF
//define TRIS_POWER_ON_OFF
#define IE_POWER_ON_OFF INTCON3bits.INT3IE
#define IF_POWER_ON_OFF INTCON3bits.INT3IF
#define IP_POWER_ON_OFF INTCON2bits.INT3IP
#define EDGE_POWER_ON_OFF INTCON2bits.INTEDG3

//VARIABLES GLOBALES

extern OS_EVENT *mBoxPromData; //Manejador del MailBox que significa que un nuevo periodo ha sido leído y debe
                        almacenarse en el buffer

extern struct ConfigdsPIC33 config;

#define TIMEOUT_PEND_MBOX 50000

//DIO variable
extern uint16_t ValChZero, CntData ;
extern float Prom;
extern float *PtrPromFromMBox;
extern float PromFromMbox;
extern uint16_t whole;
extern uint16_t decimal;
extern char array[];
extern float TmpCnt;

extern uint8_t err;
extern uint8_t iiTaskUart;

extern float TmpCnt;

```

```
extern OS_EVENT *mBoxPromData; //Manejador del MailBox que significa que un nuevo periodo ha sido leído y debe
                                almacenarse en el buffer
```

```
#endif //GLOBALS_H
```

../v0.1/includes.h

```
/*
*****
*                                     uC/OS-II
*                               The Real-Time Kernel
*
*                               (c) Copyright 2006, Micrium, Weston, FL
*                               All Rights Reserved
*
*                               MASTER INCLUDE FILE
*****
*/

#ifndef INCLUDES_H
#define INCLUDES_H

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>

#include <p33FJ128GP804.h>

#include <cpu.h>
#include <ucos_ii.h>

#include <lib_def.h>
#include <lib_str.h>
#include <lib_mem.h>

#include <dsPIC_cfg.h>
#include <DIO.H>
#include <UART.h>

#if (uC_PROBE_OS_PLUGIN > 0)
#include <os_probe.h>
#endif

#if (uC_PROBE_COM_MODULE > 0)
#include <probe_com.h>
#endif

#if (PROBE_COM_METHOD_RS232 > 0)
#include <probe_rs232.h>
#endif
#endif

/* End of File */
```

../v0.1/drivers/DIO/SOURCE/DIO.C

```
/*
*****
*                               Embedded Systems Building Blocks
*                               Complete and Ready-to-Use Modules in C
*
*                               Discrete I/O Module
*
*                               (c) Copyright 1999, Jean J. Labrosse, Weston, FL
*                               All Rights Reserved
*
*   Filename   : DIO.C
*   Programmer : Jean J. Labrosse
*****
*/

/*
*****
*                               INCLUDE FILES
*****
*/

#define DIO_GLOBALS
#include <includes.h>
// #include 'DIO.H'
#include <os_cpu.h>
#include <cpu_def.h>
#include <cpu.h>
// #include <teclado.h>
```

```

/*
*****
*
*                                LOCAL VARIABLES
*
*****
*/

extern OS_STK      DIOTaskStk[DIO_TASK_STK_SIZE];

/*
*****
*
*                                LOCAL FUNCTION PROTOTYPES
*
*****
*/

static void      DIIsTrig(DIO_DI *pdi);

static void      DIOTask(void *data);

static void      DIUpdate(void);

static BOOLEAN   DOIsBlinkEn(DIO_DO *pdo);
static void      DOUpdate(void);

/* $PAGE */
/*
*****
*
*                                CONFIGURE DISCRETE INPUT EDGE DETECTION
*
* Description : This function is used to configure the edge detection capability of the discrete input
*               channel.
* Arguments   : n      is the discrete input channel to configure (0..DIO_MAX_DI-1).
*               fnct   is a pointer to a function that will be executed if the desired edge has been
*               detected.
*               arg    is a pointer to arguments that are passed to the function called.
* Returns     : None.
*****
*/

#if DI_EDGE_EN
void DICfgEdgeDetectFnct (INT8U n, void (*fnct)(void *), void *arg)
{
    DIO_DI *pdi;
    #if (CPU_CFG_CRITICAL_METHOD == CPU_CRITICAL_METHOD_STATUS_LOCAL)
        CPU_SR      cpu_sr;
    #endif

    if (n < DIO_MAX_DI) {
        pdi          = &DITbl[n];
        OS_ENTER_CRITICAL();
        pdi->DITrigFnct = fnct;
        pdi->DITrigFnctArg = arg;
        OS_EXIT_CRITICAL();
    }
}

#endif
/* $PAGE */
/*
*****
*
*                                CONFIGURE DISCRETE INPUT MODE
*
* Description : This function is used to configure the mode of a discrete input channel.
* Arguments   : n      is the discrete input channel to configure (0..DIO_MAX_DI-1).
*               mode   is the desired mode and can be:
*               DI_MODE_LOW           input is forced LOW
*               DI_MODE_HIGH          input is forced HIGH
*               DI_MODE_DIRECT         input is based on state of physical sensor (default)
*               DI_MODE_INV           input is based on the complement of physical sensor
*               DI_MODE_EDGE_LOW_GOING a LOW-going transition is detected
*               DI_MODE_EDGE_HIGH_GOING a HIGH-going transition is detected
*               DI_MODE_EDGE_BOTH      both a LOW-going and a HIGH-going transition are detected
*               DI_MODE_TOGGLE_LOW_GOING a LOW-going transition is detected in toggle mode
*               DI_MODE_TOGGLE_HIGH_GOING a HIGH-going transition is detected in toggle mode
* Returns     : None.
* Notes      : Edge detection is only available if the configuration constant DI_EDGE_EN is set to 1.
*****
*/

void DICfgMode (INT8U n, INT8U mode)
{
    #if (CPU_CFG_CRITICAL_METHOD == CPU_CRITICAL_METHOD_STATUS_LOCAL)
        CPU_SR      cpu_sr;
    #endif

    if (n < DIO_MAX_DI) {
        OS_ENTER_CRITICAL();
        DITbl[n].DModeSel = mode;
        OS_EXIT_CRITICAL();
    }
}

/* $PAGE */
/*
*****

```

```

*
*                                     CLEAR A DISCRETE INPUT CHANNEL
*
* Description : This function clears the number of edges detected if the discrete input channel is
*               configured to count edges.
* Arguments   : n       is the discrete input channel (0..DIO_MAX_DI-1) to clear.
* Returns     : none
* ****
*/

#if DI_EDGE_EN
void DIClr (INT8U n)
{
    DIO_DI *pdi;
    #if (CPU_CFG_CRITICAL_METHOD == CPU_CRITICAL_METHOD_STATUS_LOCAL)
        CPU_SR      cpu_sr;
    #endif

    if (n < DIO_MAX_DI) {
        pdi = &DITbl[n];
        OS_ENTER_CRITICAL();
        if (pdi->DIModeSel == DI_MODE_EDGE_LOW_GOING ||          /* See if edge detection mode selected */
            pdi->DIModeSel == DI_MODE_EDGE_HIGH_GOING ||
            pdi->DIModeSel == DI_MODE_EDGE_BOTH) {
            pdi->DIVal = 0;                                       /* Clear the number of edges detected */
        }
        OS_EXIT_CRITICAL();
    }
}
#endif

/*$PAGE*/
/*
*****
*                                     GET THE STATE OF A DISCRETE INPUT CHANNEL
*
* Description : This function is used to get the current state of a discrete input channel. If the input
*               mode is set to one of the edge detection modes, the number of edges detected is returned.
* Arguments   : n       is the discrete input channel (0..DIO_MAX_DI-1).
* Returns     : 0       if the discrete input is negated or, if an edge has not been detected
*               1       if the discrete input is asserted
*               > 0     if edges have been detected
* ****
*/

INT16U DIGet (INT8U n)
{
    INT16U val;
    #if (CPU_CFG_CRITICAL_METHOD == CPU_CRITICAL_METHOD_STATUS_LOCAL)
        CPU_SR      cpu_sr;
    #endif

    if (n < DIO_MAX_DI) {
        OS_ENTER_CRITICAL();
        val = DITbl[n].DIVal;                                     /* Get state of DI channel */
        OS_EXIT_CRITICAL();
        return (val);
    } else {
        return (0);                                              /* Return negated for invalid channel */
    }
}

/*$PAGE*/
/*
*****
*                                     DETECT EDGE ON INPUT
*
* Description : This function is called to detect an edge (low-going, high-going or both) on the selected
*               discrete input.
* Arguments   : pdi     is a pointer to the discrete input data structure.
* Returns     : none
* ****
*/

#if DI_EDGE_EN
static void DIIsTrig (DIO_DI *pdi)
{
    BOOLEAN trig;

    trig = FALSE;
    switch (pdi->DIModeSel) {
        case DI_MODE_EDGE_LOW_GOING:                             /* Negative going edge */
            if (pdi->DIPrev == 1 && pdi->DIIn == 0) {
                trig = TRUE;
            }
            break;

        case DI_MODE_EDGE_HIGH_GOING:                             /* Positive going edge */
            if (pdi->DIPrev == 0 && pdi->DIIn == 1) {
                trig = TRUE;
            }
            break;
    }
}
#endif

```



```

        case DI_MODE_EDGE_BOTH:                                /* Both positive and negative going */
            if ((pdi->DIPrev == 1 && pdi->DIIn == 0) ||
                (pdi->DIPrev == 0 && pdi->DIIn == 1)) {
                trig = TRUE;
            }
            break;
    }
    if (trig == TRUE) {
        if (pdi->DITrigFnct != NULL) {                          /* See if edge detected */
            (*pdi->DITrigFnct)(pdi->DITrigFnctArg);             /* Yes, see used defined a function */
            /* Yes, execute the user function */
        }
        if (pdi->DIVal < 255) {                                  /* Increment number of edges counted */
            pdi->DIVal++;
        }
    }
    pdi->DIPrev = pdi->DIIn;                                    /* Memorize previous input state */
}
#endif

/* $PAGE */
/* $PAGE */
/*
*****
*                               UPDATE DISCRETE IN CHANNELS
*
* Description : This function processes all of the discrete input channels.
* Arguments   : None.
* Returns     : None.
*****
*/

static void DIUpdate (void)
{
    INT8U i;
    DIO_DI *pdi;

    pdi = &DITbl[0];
    for (i = 0; i < DIO_MAX_DI; i++) {
        if (pdi->DIBypassEn == FALSE) {                         /* See if discrete input channel is bypassed */
            switch (pdi->DIModelSel) {                          /* No, process channel */
                case DI_MODE_LOW:                               /* Input is forced low */
                    pdi->DIVal = 0;
                    break;

                case DI_MODE_HIGH:                             /* Input is forced high */
                    pdi->DIVal = 1;
                    break;

                case DI_MODE_DIRECT:                           /* Input is based on state of physical input */
                    pdi->DIVal = (INT8U)pdi->DIIn;              /* Obtain the state of the sensor */
                    break;

                case DI_MODE_INV:                               /* Input is based on the complement state of input */
                    pdi->DIVal = (INT8U)(pdi->DIIn ? 0 : 1);
                    break;
            }
        }
        #if DI_EDGE_EN
        case DI_MODE_EDGE_LOW_GOING:
        case DI_MODE_EDGE_HIGH_GOING:
        case DI_MODE_EDGE_BOTH:
            DIIsTrig(pdi);                                     /* Handle edge triggered mode */
            break;
        #endif
    }
    /* $PAGE */

    case DI_MODE_TOGGLE_LOW_GOING:
        if (pdi->DIPrev == 1 && pdi->DIIn == 0) {
            pdi->DIVal = pdi->DIVal ? 0 : 1;
        }
        pdi->DIPrev = pdi->DIIn;
        break;

    case DI_MODE_TOGGLE_HIGH_GOING:
        if (pdi->DIPrev == 0 && pdi->DIIn == 1) {
            pdi->DIVal = pdi->DIVal ? 0 : 1;
        }
        pdi->DIPrev = pdi->DIIn;
        break;
    }
    }
    pdi++;
}
/* Point to next DIO_DO element */

}

/* $PAGE */
/*
*****
*                               DISCRETE I/O MANAGER INITIALIZATION
*
* Description : This function initializes the discrete I/O manager module.
* Arguments   : None

```

```

* Returns      : None.
*****
*/

void DIOInit (void)
{
    INT8U    err;
    INT8U    i;
    DIO_DI   *pdi;
    DIO_DO   *pdo;

    pdi = &DITbl[0];
    for (i = 0; i < DIO_MAX_DI; i++) {
        pdi->DIVal      = 0;
        pdi->DIBypassEn = FALSE;
        pdi->DIModeSel   = DI_MODE_DIRECT; /* Set the default mode to direct input */
    }
    #if DI_EDGE_EN
        pdi->DITrigFnct   = (void *)0; /* No function to execute when transition detected */
        pdi->DITrigFnctArg = (void *)0;
    #endif
    pdi++;
    pdo = &DOTbl[0];
    for (i = 0; i < DIO_MAX_DO; i++) {
        pdo->DOOut      = 0;
        pdo->DOBypassEn = FALSE;
        pdo->DOModeSel   = DO_MODE_DIRECT; /* Set the default mode to direct output */
        pdo->DOInv       = FALSE;
    }
    #if DO_BLINK_MODE_EN
        pdo->DOBlinkEnSel = DO_BLINK_EN_NORMAL; /* Blinking is enabled by direct user request */
        pdo->DOA          = 1;
        pdo->DOB          = 2;
        pdo->DOBCtr       = 2;
    #endif
    pdo++;
}
#if DO_BLINK_MODE_EN
    DOSyncCtrMax = 100;
#endif
DIOInitIO();
//OSTaskCreate(DIOTask, (void *)0, &DIOTaskStk[DIO_TASK_STK_SIZE], DIO_TASK_PRIO);

OSTaskCreateExt(DIOTask,
                (void *)0,
                (OS_STK *)&DIOTaskStk[0],
                DIO_TASK_PRIO,
                DIO_TASK_PRIO,
                (OS_STK *)&DIOTaskStk[DIO_TASK_STK_SIZE-1],
                DIO_TASK_STK_SIZE,
                (void *)0,
                OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);
}

/*$PAGE*/
/*
*****
DISCRETE I/O MANAGER TASK
*****
* Description : This task is created by DIOInit() and is responsible for updating the discrete inputs and
*               discrete outputs.
*               DIOTask() executes every DIO_TASK_DLY_TICKS.
* Arguments   : None.
* Returns     : None.
*****
*/

static void DIOTask (void *data)
{
    data = data; /* Avoid compiler warning (uC/OS requirement) */
    for (;;) {
        OSTimeDly(DIO_TASK_DLY_TICKS); /* Delay between execution of DIO manager */
        DIRd(); /* Read physical inputs and map to DI channels */
        DIUpdate(); /* Update all DI channels */
        DOUpdate(); /* Update all DO channels */
        DOWr(); /* Map DO channels to physical outputs */
    }
}
/*$PAGE*/

/*
*****
SET THE STATE OF THE BYPASSED SENSOR
*****
* Description : This function is used to set the state of the bypassed sensor. This function is used to
*               simulate the presence of the sensor. This function is only valid if the bypass 'switch'
*               is open.
* Arguments   : n is the discrete input channel (0..DIO_MAX_DI-1).
*               val is the state of the bypassed sensor:
*                   0 indicates a negated sensor
*                   1 indicates an asserted sensor
*                   > 0 indicates the number of edges detected in edge mode

```

```

* Returns      : None.
*****
*/

void DISetBypass (INT8U n, INT16U val)
{
    DIO_DI  *pdi;

    #if (CPU_CFG_CRITICAL_METHOD == CPU_CRITICAL_METHOD_STATUS_LOCAL)
        CPU_SR      cpu_sr;
    #endif
    if (n < DIO_MAX_DI) {
        pdi = &DITbl[n];
        OS_ENTER_CRITICAL();
        if (pdi->DIBypassEn == TRUE) {          /* See if sensor is bypassed */
            pdi->DIVal = val;                    /* Yes, then set the new state of the DI channel */
        }
        OS_EXIT_CRITICAL();
    }
}

/* $PAGE */
/*
*****
SET THE STATE OF THE SENSOR BYPASS SWITCH
*
* Description : This function is used to set the state of the sensor bypass switch. The sensor is
*                bypassed when the 'switch' is open (i.e. DIBypassEn is set to TRUE).
* Arguments   : n    is the discrete input channel (0..DIO_MAX_DI-1).
*                state is the state of the bypass switch:
*                    FALSE disables sensor bypass (i.e. the bypass 'switch' is closed)
*                    TRUE  enables sensor bypass (i.e. the bypass 'switch' is open)
* Returns     : None.
*****
*/

void DISetBypassEn (INT8U n, BOOLEAN state)
{
    #if (CPU_CFG_CRITICAL_METHOD == CPU_CRITICAL_METHOD_STATUS_LOCAL)
        CPU_SR      cpu_sr;
    #endif
    if (n < DIO_MAX_DI) {
        OS_ENTER_CRITICAL();
        DITbl[n].DIBypassEn = state;
        OS_EXIT_CRITICAL();
    }
}

/* $PAGE */
/*
*****
CONFIGURE THE DISCRETE OUTPUT BLINK MODE
*
* Description : This function is used to configure the blink mode of the discrete output channel.
* Arguments   : n    is the discrete output channel (0..DIO_MAX_DO-1).
*                mode is the desired blink mode:
*                    DO_BLINK_EN      Blink is always enabled
*                    DO_BLINK_EN_NORMAL Blink depends on user request's state
*                    DO_BLINK_EN_INV  Blink depends on the complemented user request's state
*                a    is the ON time relative to how often the DIO task executes (1..250)
*                b    is the period (in DO_MODE_BLINK_ASYNC mode) (1..250)
* Returns     : None.
*****
*/

#if DO_BLINK_MODE_EN
void DOcfgBlink (INT8U n, INT8U mode, INT8U a, INT8U b)
{
    DIO_DO  *pdo;

    #if (CPU_CFG_CRITICAL_METHOD == CPU_CRITICAL_METHOD_STATUS_LOCAL)
        CPU_SR      cpu_sr;
    #endif

    if (n < DIO_MAX_DO) {
        pdo = &DOTbl[n];
        OS_ENTER_CRITICAL();
        pdo->DOBlinkEnSel = mode;
        pdo->DOA         = a;
        pdo->DOB         = b;
        pdo->DOBCtr      = 0;
        OS_EXIT_CRITICAL();
    }
}

#endif

/* $PAGE */
/*
*****
CONFIGURE DISCRETE OUTPUT MODE
*
* Description : This function is used to configure the mode of a discrete output channel.

```

```

* Arguments   : n       is the discrete output channel to configure (0..DIO_MAX_DO-1).
*               mode    is the desired mode and can be:
*               DO_MODE_LOW      output is forced LOW
*               DO_MODE_HIGH     output is forced HIGH
*               DO_MODE_DIRECT   output is based on state of DOBypass
*               DO_MODE_BLINK_SYNC output will be blinking synchronously with DOSyncCtr
*               DO_MODE_BLINK_ASYNC output will be blinking based on DOA and DOB
*               inv       indicates whether the output will be inverted:
*               TRUE      forces the output to be inverted
*               FALSE     does not cause any inversion
* Returns     : None.
*****
*/

void DOCfgMode (INT8U n, INT8U mode, BOOLEAN inv)
{
    DIO_DO *pdo;

    #if (CPU_CFG_CRITICAL_METHOD == CPU_CRITICAL_METHOD_STATUS_LOCAL)
        CPU_SR      cpu_sr;
    #endif

    if (n < DIO_MAX_DO) {
        pdo = &DOTbl[n];
        OS_ENTER_CRITICAL();
        pdo->DModeSel = mode;
        pdo->DOInv = inv;
        OS_EXIT_CRITICAL();
    }

    /*$PAGE*/
    /*
    *****
    GET THE STATE OF THE DISCRETE OUTPUT
    *****
    * Description : This function is used to obtain the state of the discrete output.
    * Arguments   : n       is the discrete output channel (0..DIO_MAX_DO-1).
    * Returns     : TRUE    if the output is asserted.
    *               FALSE   if the output is negated.
    *****
    */

    BOOLEAN DOGet (INT8U n)
    {
        BOOLEAN out;
        #if (CPU_CFG_CRITICAL_METHOD == CPU_CRITICAL_METHOD_STATUS_LOCAL)
            CPU_SR      cpu_sr;
        #endif

        if (n < DIO_MAX_DO) {
            OS_ENTER_CRITICAL();
            out = DOTbl[n].DOOut;
            OS_EXIT_CRITICAL();
            return (out);
        } else {
            return (FALSE);
        }
    }

    /*$PAGE*/
    /*
    *****
    SEE IF BLINK IS ENABLED
    *****
    * Description : See if blink mode is enabled.
    * Arguments   : pdo     is a pointer to the discrete output data structure.
    * Returns     : TRUE    if blinking is enabled
    *               FALSE   otherwise
    *****
    */

    #if DO_BLINK_MODE_EN
    static BOOLEAN DOIsBlinkEn (DIO_DO *pdo)
    {
        BOOLEAN en;

        en = FALSE;
        switch (pdo->DOBlinkEnSel) {
            case DO_BLINK_EN: /* Blink is always enabled */
                en = TRUE;
                break;

            case DO_BLINK_EN_NORMAL: /* Blink depends on user request's state */
                en = pdo->DOBypass;
                break;

            case DO_BLINK_EN_INV: /* Blink depends on the complemented user request's state */
                en = pdo->DOBypass ? FALSE : TRUE;
                break;
        }
    }
    #endif

```

```

    return (en);
}
#endif

/* $PAGE */
/*
*****
*
* SET THE STATE OF THE DISCRETE OUTPUT
*
* Description : This function is used to set the state of the discrete output.
* Arguments   : n    is the discrete output channel (0..DIO_MAX_DO-1).
*               state is the desired state of the output:
*                 FALSE indicates a negated output
*                 TRUE  indicates an asserted output
* Returns      : None.
* Notes        : The actual output will be complemented if 'DIInv' is set to TRUE.
*****
*/

void DOSet (INT8U n, BOOLEAN state)
{
    #if (CPU_CFG_CRITICAL_METHOD == CPU_CRITICAL_METHOD_STATUS_LOCAL)
        CPU_SR
    #endif
    if (n < DIO_MAX_DO) {
        OS_ENTER_CRITICAL();
        DOTbl[n].DOCtrl = state;
        OS_EXIT_CRITICAL();
    }
}

/* $PAGE */
/*
*****
*
* SET THE STATE OF THE BYPASSED OUTPUT
*
* Description : This function is used to set the state of the bypassed output. This function is used to
*               override (or bypass) the application software and allow the output to be controlled
*               directly. This function is only valid if the bypass switch is open.
* Arguments   : n    is the discrete output channel (0..DIO_MAX_DO-1).
*               state is the desired state of the output:
*                 FALSE indicates a negated output
*                 TRUE  indicates an asserted output
* Returns      : None.
* Notes        : 1) The actual output will be complemented if 'DIInv' is set to TRUE.
*               2) In blink mode, this allows blinking to be enabled or not.
*****
*/

void DOSetBypass (INT8U n, BOOLEAN state)
{
    DIO_DO *pdo;
    #if (CPU_CFG_CRITICAL_METHOD == CPU_CRITICAL_METHOD_STATUS_LOCAL)
        CPU_SR
    #endif

    if (n < DIO_MAX_DO) {
        pdo = &DOTbl[n];
        OS_ENTER_CRITICAL();
        if (pdo->DOBypassEn == TRUE) {
            pdo->DOBypass = state;
        }
        OS_EXIT_CRITICAL();
    }
}

/* $PAGE */
/*
*****
*
* SET THE STATE OF THE OUTPUT BYPASS
*
* Description : This function is used to set the state of the output bypass switch. The output is
*               bypassed when the 'switch' is open (i.e. DOBypassEn is set to TRUE).
* Arguments   : n    is the discrete output channel (0..DIO_MAX_DO-1).
*               state is the state of the bypass switch:
*                 FALSE disables output bypass (i.e. the switch is closed)
*                 TRUE  enables output bypass (i.e. the switch is open)
* Returns      : None.
*****
*/

void DOSetBypassEn (INT8U n, BOOLEAN state)
{
    #if (CPU_CFG_CRITICAL_METHOD == CPU_CRITICAL_METHOD_STATUS_LOCAL)
        CPU_SR
    #endif
    if (n < DIO_MAX_DO) {
        OS_ENTER_CRITICAL();
        DOTbl[n].DOBypassEn = state;
        OS_EXIT_CRITICAL();
    }
}

```

```

/* $PAGE */

/*
*****
*
* SET THE MAXIMUM VALUE FOR THE SYNCHRONOUS COUNTER
*
* Description : This function is used to set the maximum value taken by the synchronous counter which is
*               used in the synchronous blink mode.
* Arguments   : val is the maximum value for the counter (1..255)
* Returns     : None.
*****
*/

#if DO_BLINK_MODE_EN
void DOSetSyncCtrMax (INT8U val)
{
    #if (CPU_CFG_CRITICAL_METHOD == CPU_CRITICAL_METHOD_STATUS_LOCAL)
        CPU_SR      cpu_sr;
    #endif
    OS_ENTER_CRITICAL();
    DOSyncCtrMax = val;
    DOSyncCtr    = val;
    OS_EXIT_CRITICAL();
}
#endif
/* $PAGE */

/*
*****
* UPDATE DISCRETE OUT CHANNELS
*
* Description : This function is called to process all of the discrete output channels.
* Arguments   : None.
* Returns     : None.
*****
*/

static void DOUpdate (void)
{
    INT8U      i;
    BOOLEAN    out;
    DIO_DO     *pdo;

    pdo = &DOTbl[0];
    for (i = 0; i < DIO_MAX_DO; i++) {
        /* Process all discrete output channels */
        if (pdo->DOBypassEn == FALSE) {
            /* See if DO channel is enabled */
            pdo->DOCtrl;
            /* Obtain control state from application */
        }
        out = FALSE;
        /* Assume that the output will be low unless changed */
        switch (pdo->DOModeSel) {
            case DO_MODE_LOW:
                /* Output will in fact be low */
                break;

            case DO_MODE_HIGH:
                /* Output will be high */
                out = TRUE;
                break;

            case DO_MODE_DIRECT:
                /* Output is based on state of user supplied state */
                out = pdo->DOBypass;
                break;
        }

/* $PAGE */
        #if DO_BLINK_MODE_EN
            case DO_MODE_BLINK_SYNC:
                /* Sync. Blink mode */
                if (DOIsBlinkEn(pdo)) {
                    /* See if Blink is enabled ... */
                    if (pdo->DOA >= DOSyncCtr) {
                        /* ... yes, High when below threshold */
                        out = TRUE;
                    }
                }
                break;

            case DO_MODE_BLINK_ASYNC:
                /* Async. Blink mode */
                if (DOIsBlinkEn(pdo)) {
                    /* See if Blink is enabled ... */
                    if (pdo->DOA >= pdo->DOBCtr) {
                        /* ... yes, High when below threshold */
                        out = TRUE;
                    }
                }
                if (pdo->DOBCtr < pdo->DOB) {
                    /* Update the threshold counter */
                    pdo->DOBCtr++;
                }
                else {
                    pdo->DOBCtr = 0;
                }
                break;
        #endif
    }
    if (pdo->DOInv == TRUE) {
        /* See if output needs to be inverted ... */
        pdo->DOOut = out ? FALSE : TRUE;
        /* ... yes, complement output */
    }
    else {
        pdo->DOOut = out;
        /* ... no, no inversion! */
    }
    pdo++;
    /* Point to next DIO_DO element */
}

```

```

#if DO_BLINK_MODE_EN
    if (DOSyncCtr < DOSyncCtrMax) {                                /* Update the synchronous free running ctr */
        DOSyncCtr++;
    } else {
        DOSyncCtr = 0;
    }
#endif
}
/* $PAGE */
#ifndef CFG_C
/*
*****
*
*                               INITIALIZE PHYSICAL I/Os
*
* Description : This function is by DIOInit() to initialize the physical I/O used by the DIO driver.
* Arguments   : None.
* Returns     : None.
* Notes       : The physical I/O is assumed to be an 82C55 chip initialized as follows:
*               Port A = OUT (Discrete outputs)
*               Port B = IN  (Discrete inputs)
*               Port C = OUT (not used)
*****
*/

void DIOInitIO (void)
{
    //Configuracin de los puertos como
    // SALIDAS
    TRISBbits.TRISB9 = 0;
    TRISBbits.TRISB8 = 0;
    TRISBbits.TRISB7 = 0;
    TRISBbits.TRISB6 = 0;
    // ENTRADAS
    AD1PCFGLbits.PCFG6 = 1; //RC0
    AD1PCFGLbits.PCFG7 = 1; //RC1
    AD1PCFGLbits.PCFG8 = 1; //RC2

    TRISCbits.TRISC0 = 1;
    TRISCbits.TRISC1 = 1;
    TRISCbits.TRISC2 = 1;
    TRISCbits.TRISC3 = 1;
}

/*
*****
*
*                               READ PHYSICAL INPUTS
*
* Description : This function is called to read and map all of the physical inputs used for discrete
*               inputs and map these inputs to their appropriate discrete input data structure.
* Arguments   : None.
* Returns     : None.
*****
*/

void DIRd (void)
{
    DIO_DI *pdi;
    INT8U i;
    INT8U in;
    INT8U msk;

    pdi = &DITbl[0];
    msk = 0x01;                                /* Point at beginning of discrete inputs */
                                              /* Set mask to extract bit 0 */

    in |= PORTCbits.RC0;                        /* Read the physical port (8 bits) */

    in |= (PORTCbits.RC1<<1);
    in |= (PORTCbits.RC2<<2);
    in |= (PORTCbits.RC3<<3);

    for (i = 0; i < 8; i++) {                    /* Map all 8 bits to first 8 DI channels */
        pdi->DIIn = (BOOLEAN)(in & msk) ? 1 : 0;
        msk <<= 1;
        pdi++;
    }
}
/* $PAGE */
/*
*****
*
*                               UPDATE PHYSICAL OUTPUTS
*
* Description : This function is called to map all of the discrete output channels to their appropriate
*               physical destinations.
* Arguments   : None.
* Returns     : None.
*****
*/

void DOWr (void)
{
    DIO_DO *pdo;
    INT8U i;
    INT8U out;

```

```

INT8U    msk;

pdo = &DOTbl[0];          /* Point at first discrete output channel */
msk = 0x01;               /* First DO will be mapped to bit 0 */
out = 0x00;               /* Local 8 bit port image */
for (i = 0; i < 8; i++) { /* Map first 8 DO to 8 bit port image */
    if (pdo->DOOut == TRUE) {
        out |= msk;
    }
    msk <<= 1;
    pdo++;
}

PORTBbits.RB9 = out & 0x01;
PORTBbits.RB8 = (out>>1) & 0x01;
PORTBbits.RB7 = (out>>2) & 0x01;
PORTBbits.RB6 = (out>>3) & 0x01;
}
#endif

```

../v0.1/drivers/DIO/SOURCE/DIO.H

```

/*
*****
*
*      Embedded Systems Building Blocks
*      Complete and Ready-to-Use Modules in C
*
*      Discrete I/O Module
*
*      (c) Copyright 1999, Jean J. Labrosse, Weston, FL
*      All Rights Reserved
*
*      Filename    : DIO.H
*      Programmer  : Jean J. Labrosse
*****
*/

/*
*****
*
*      CONFIGURATION CONSTANTS
*****
*/

#ifndef CFG_H
#define FALSE
#define TRUE !FALSE
#define DIO_TASK_PRIO 1
#define DIO_TASK_DLY_TICKS 1
#define DIO_TASK_STK_SIZE 256//luis126

#define DIO_MAX_DI 8 /* Maximum number of Discrete Input Channels (1..255) */
#define DIO_MAX_DO 8 /* Maximum number of Discrete Output Channels (1..255) */

#define DI_EDGE_EN 1 /* Enable code generation to support edge trig. (when 1) */
#define DO_BLINK_MODE_EN 1 /* Enable code generation to support blink mode (when 1) */

#endif

#ifdef DIO_GLOBALS
#define DIO_EXT
#else
#define DIO_EXT extern
#endif

/*
*****
*
*      DISCRETE INPUT CONSTANTS
*****
*/

#define DI_MODE_SELECTOR_VALUES /*
#define DI_MODE_LOW 0 /* Input is forced low */
#define DI_MODE_HIGH 1 /* Input is forced high */
#define DI_MODE_DIRECT 2 /* Input is based on state of physical input */
#define DI_MODE_INV 3 /* Input is based on the complement of the physical input */
#define DI_MODE_EDGE_LOW_GOING 4 /* Low going edge detection of input */
#define DI_MODE_EDGE_HIGH_GOING 5 /* High going edge detection of input */
#define DI_MODE_EDGE_BOTH 6 /* Both low and high going edge detection of input */
#define DI_MODE_TOGGLE_LOW_GOING 7 /* Low going edge detection of input */
#define DI_MODE_TOGGLE_HIGH_GOING 8 /* High going edge detection of input */

#define DI_EDGE_TRIGGERING_MODE_SELECTOR_VALUES /*
#define DI_EDGE_LOW_GOING 0 /* Negative going edge */
#define DI_EDGE_HIGH_GOING 1 /* Positive going edge */
#define DI_EDGE_BOTH 2 /* Both positive and negative going */

```



```

** DISCRETE OUTPUT CONSTANTS **
*/

/* DO MODE SELECTOR VALUES */
#define DO_MODE_LOW 0 /* Output will be low */
#define DO_MODE_HIGH 1 /* Output will be high */
#define DO_MODE_DIRECT 2 /* Output is based on state of user supplied state */
#define DO_MODE_BLINK_SYNC 3 /* Sync. Blink mode */
#define DO_MODE_BLINK_ASYNC 4 /* Async. Blink mode */

/* DO BLINK MODE ENABLE SELECTOR VALUES */
#define DO_BLINK_EN 0 /* Blink is always enabled */
#define DO_BLINK_EN_NORMAL 1 /* Blink depends on user request's state */
#define DO_BLINK_EN_INV 2 /* Blink depends on the complemented user request's state */

/*
** DATA TYPES **
*/

typedef struct dio_di { /* DISCRETE INPUT CHANNEL DATA STRUCTURE */
    BOOLEAN DIIn; /* Current state of sensor input */
    INT16U DIVal; /* State of discrete input channel (or # of transitions) */
    BOOLEAN DIPrev; /* Previous state of DIIn for edge detection */
    BOOLEAN DIBypassEn; /* Bypass enable switch (Bypass when TRUE) */
    INT8U DIModeSel; /* Discrete input channel mode selector */
} DI_EDGE_EN;

void (*DITrigFnc)(void *); /* Function to execute if edge triggered */
void *DITrigFncArg; /* arguments passed to function when edge detected */

#endif
} DIO_DI;

typedef struct dio_do { /* DISCRETE OUTPUT CHANNEL DATA STRUCTURE */
    BOOLEAN DOOut; /* Current state of discrete output channel */
    BOOLEAN DOCtrl; /* Discrete output control request */
    BOOLEAN DOBypass; /* Discrete output control bypass state */
    BOOLEAN DOBypassEn; /* Bypass enable switch (Bypass when TRUE) */
    INT8U DOModeSel; /* Discrete output channel mode selector */
    INT8U DOBlinkEnSel; /* Blink enable mode selector */
    BOOLEAN DOInv; /* Discrete output inverter selector (Invert when TRUE) */
} DO_EDGE_EN;

void (*DOTrigFnc)(void *); /* Function to execute if edge triggered */
void *DOTrigFncArg; /* arguments passed to function when edge detected */

#endif
} DIO_DO;

/* $PAGE */
/*
** GLOBAL VARIABLES **
*/

DIO_EXT DIO_DI DITbl[DIO_MAX_DI];
DIO_EXT DIO_DO DOTbl[DIO_MAX_DO];

#if DO_BLINK_MODE_EN
DIO_EXT INT8U DOSyncCtr;
DIO_EXT INT8U DOSyncCtrMax;
#endif

/*
** FUNCTION PROTOTYPES **
*/

void DIOInit(void);

void DICfgMode(INT8U n, INT8U mode);
INT16U DIGet(INT8U n);
void DISetBypassEn(INT8U n, BOOLEAN state);
void DISetBypass(INT8U n, INT16U val);

#if DI_EDGE_EN
void DIClr(INT8U n);
void DICfgEdgeDetectFnc(INT8U n, void (*fnc)(void *), void *arg);
#endif

void DOCfgMode(INT8U n, INT8U mode, BOOLEAN inv);
BOOLEAN DOGet(INT8U n);
void DOSet(INT8U n, BOOLEAN state);
void DOSetBypass(INT8U n, BOOLEAN state);
void DOSetBypassEn(INT8U n, BOOLEAN state);

#if DO_BLINK_MODE_EN
void DOCfgBlink(INT8U n, INT8U mode, INT8U a, INT8U b);
void DOSetSyncCtrMax(INT8U val);
#endif

```

```
#endif

/*
*****
*
*                               FUNCTION PROTOTYPES
*                               HARDWARE SPECIFIC
*****
*/

void    DIOInitIO(void);
void    DIRd(void);
void    DOWr(void);
```

../v0.1/drivers/UART.c

```
//#include <includes.h>
#include <p33FJ128GP804.h>
#include 'UART.h'
void InitUART1(void) {

    // configure U1MODE
    U1MODEbits.UARTEN = 0; // Bit15 TX, RX DISABLED, ENABLE at end of func
    //U1MODEbits.notimplemented; // Bit14
    U1MODEbits.USIDL = 0; // Bit13 Continue in Idle
    U1MODEbits.IREN = 0; // Bit12 No IR translation
    U1MODEbits.RTSMD = 0; // Bit11 Simplex Mode
    //U1MODEbits.notimplemented; // Bit10
    U1MODEbits.UEN = 0; // Bits8,9 TX,RX enabled, CTS,RTS not
    U1MODEbits.WAKE = 0; // Bit7 No Wake up (since we don't sleep here)
    U1MODEbits.LPBACK = 0; // Bit6 No Loop Back
    U1MODEbits.ABAUD = 0; // Bit5 No Autobaud (would require sending '55')
    U1MODEbits.URXINV = 0; // Bit4 IdleState = 1 (for dsPIC)
    U1MODEbits.BRGH = 0; // Bit3 16 clocks per bit period
    U1MODEbits.PDSEL = 0; // Bits1,2 8bit, No Parity
    U1MODEbits.STSEL = 0; // Bit0 One Stop Bit

    // U1BRG = (Fcy / (16 * BaudRate)) - 1
    // U1BRG = (39.804.000 / (16 * 9600)) - 1
    // U1BRG = 258.140625 //Round to 258

    U1BRG = 258;

    // Load all values in for U1STA SFR
    U1STAbits.UTXISEL1 = 0; //Bit15 Int when Char is transferred (1/2 config!)
    U1STAbits.UTXINV = 0; //Bit14 N/A, IRDA config
    U1STAbits.UTXISEL0 = 0; //Bit13 Other half of Bit15
    //U1STAbits.notimplemented = 0; //Bit12
    U1STAbits.UTXBRK = 0; //Bit11 Disabled
    U1STAbits.UTXEN = 0; //Bit10 TX pins controlled by periph
    U1STAbits.UTXBF = 0; //Bit9 *Read Only Bit*
    U1STAbits.TRMT = 0; //Bit8 *Read Only bit*
    U1STAbits.URXISEL = 0; //Bits6,7 Int. on character recieved
    U1STAbits.ADDEN = 0; //Bit5 Address Detect Disabled
    U1STAbits.RIDLE = 0; //Bit4 *Read Only Bit*
    U1STAbits.PERR = 0; //Bit3 *Read Only Bit*
    U1STAbits.FERR = 0; //Bit2 *Read Only Bit*
    U1STAbits.OERR = 0; //Bit1 *Read Only Bit*
    U1STAbits.URXDA = 0; //Bit0 *Read Only Bit*

    U1MODEbits.UARTEN = 1; // And turn the peripheral on
    U1STAbits.UTXEN = 1;
    // I think I have the thing working now.
}
```

../v0.1/drivers/UART.h

```
#ifndef UART_H
#define UART_H

void InitUART1(void);

#endif
```

/* End of file

*/

/media/sda2/SOFTWARE/UCOS-II/TFINAL/TFINAL.C

```
#include 'includes.h'

/*
*****
*
*
*****
*/

#define TASK_STK_SIZE 512 /* Size of each task's stacks (# of WORDs) */

#define CFG_H 1

/*
*****
*
*
*****
*/

OS_STK TaskStartStk[TASK_STK_SIZE]; /* Startup task stack */
OS_STK TaskTimesStk[TASK_STK_SIZE]; /* Task Times task stack */
OS_STK TaskRxStk[TASK_STK_SIZE]; /* Task RX task stack */

OS_EVENT *TransmisionesListasSem;
OS_EVENT *DatoRecibidoMbox;

/******Variables globales*****/

/*
*****
*
*
*****
*/

FUNCTION PROTOTYPES

/*
*****
*
*
*****
*/

void TaskStart (void *data); /* Function prototypes of tasks */
void Task_Rx (void *data);
void Task_Times (void *data);

/*
*****
*
*
*****
*/

MAIN

/*
*****
*
*
*****
*/

void main (void)
{
    PC_DispcLrScr(DISF_FGND_WHITE); /* Clear the screen */
    OSInit(); /* Initialize uC/OS-II */
    PC_DOSSaveReturn(); /* Save environment to return to DOS */
    PC_VectSet(uCOS, OSCtxSw); /* Install uC/OS-II's context switch vector */

    CommInit();

    //Crear los Semaphores (Buzones)
    TransmisionesListasSem = OSSemCreate(0);

    //Crear los MailBox (Buzones)
    DatoRecibidoMbox=OSMboxCreate((void *)0);

    OSTaskCreate(TaskStart, (void *)0, &TaskStartStk[TASK_STK_SIZE-1], 0);

    OSStart(); /* Start multitasking */

}

/*
*****
*
*
*****
*/

STARTUP TASK

/*
*****
*
*
*****
*/

void TaskStart (void *data)
{
    char s[80];
    WORD key;

    data = data; /* Prevent compiler warning */

    PC_DispcStr(25, 27, '<-PRESS 'ESC' TO QUIT->', DISF_FGND_WHITE);
    PC_DispcStr(16, 1, 'Trabajo Final - uC/OS-II, The Real-Time Kernel', DISF_FGND_WHITE + DISF_BGND_RED);

    OS_ENTER_CRITICAL(); /* Install uC/OS-II's clock tick ISR */
    PC_VectSet(0x08, OSTickISR);
    PC_SetTickRate(OS_TICKS_PER_SEC); /* Reprogram tick rate */
    OS_EXIT_CRITICAL();
    OSStatInit();

    OSTaskCreate(Task_Times, (void *)0, &TaskTimesStk[TASK_STK_SIZE-1], 7);
    OSTaskCreate(Task_Rx, (void *)0, &TaskRxStk[TASK_STK_SIZE-1], 8);
}
```

```

    for (;;)
    {
        PC_Dispatch(25, 27, '<-PRESS 'ESC' TO QUIT->', DISP_FGND_WHITE);
        PC_Dispatch(16, 1, 'Trabajo Final - uC/OS-II, The Real-Time Kernel', DISP_FGND_WHITE + DISP_BGND_RED);
    }

    if (PC_GetKey(&key) == TRUE) /* See if key has been pressed */
    {
        if (key == 0x1B) /* Yes, see if it's the ESCAPE key */
        {
            PC_DOSReturn(); /* Yes, return to DOS */
        }
    }
    OSTimeDlyHMSM(0, 0, 1, 0); /* Espera 1 segundo */
}

/*
*****
* TASK_RX
*****
*/
void Task_Rx (void *data)
{
    INT8U error, errSem, errMbox;
    INT8U i, c;
    INT8U periodo[7], s[8];
    int k=0, kk=0, kdata = 0;
    char sdata[] = '0000.00';
    INT8U FlagInicio = 0;

    data = data;

    CommCfgPort(COMM2, 9600, 8, COMM_PARITY_NONE, 1);
    CommSetIntVect(COMM2);
    CommRxIntEn(COMM2);

    for(;;)
    {
        PC_Dispatch(28, 6, 'Dato recibido: ', DISP_FGND_YELLOW + DISP_BGND_BLUE);
        PC_Dispatch(28, 8, 'Periodo [seg.]: ', DISP_FGND_YELLOW);
        PC_Dispatch(28, 10, 'Char recibidos: ', DISP_FGND_YELLOW);

        if(!CommIsEmpty(COMM2))
        {
            c = CommGetChar(COMM2, &error);
            if(error == COMM_NO_ERR)
            {
                switch(c)
                {
                    case 58:
                        FlagInicio = 1;
                        break;
                    case 10:
                        kdata = 0;
                        FlagInicio = 0;
                        //errSem = OS_SemPost(TransmisionesListasSem);
                        //printf('\n %s', sdata);
                        errMbox = OS_MboxPost(DatoRecibidoMbox, (void *)&sdata[0]);
                        break;
                    default:
                        if(FlagInicio == 1)
                        {
                            k++;
                            kdata++;
                            sprintf(s, '%d', k);
                            PC_Dispatch(55, 10, s, DISP_FGND_YELLOW);
                            sdata[kdata] = c;
                        }
                }
            }
        }

        OSTimeDlyHMSM(0, 0, 0, 100);
    }
}

/*
*****
* Task_Times
* Medicion de tiempos de ejecucion
*****
*/
void Task_Times(void *data)
{
    INT8U err;
    INT8U errSem, errMbox;
    char s[40];
    char * PtrSData;
    float Fdata, periodo;
    data = data;

```

```

for (;;)
{
    //OSSemPend(TransmisionesListasSem,0,&errSem);

    PtrSData = OSMboxPend(DatoRecibidoMbox,0,&errMbox);

    PC_DisPStr(55, 6, PtrSData, DISP_FGND_YELLOW);

    Fdata = atof(PtrSData);

    if(Fdata == 0.0)
        periodo = 0.0;
    else
        periodo = 0.1 / Fdata;

    sprintf(s, '%e', periodo);
    PC_DisPStr(55, 8, s, DISP_FGND_YELLOW);

    //PC_DisPClrScr(DISP_FGND_WHITE);
    //clrscr();

    OSTimeDlyHMSM(0, 0, 0, 100);    /* Espera 1 Segundo */
}
}

```

/media/sda2/SOFTWARE/UCOS-II/TFINAL/OS_CFG.H

```

/*
*****
*
*                               uC/OS-II
*                               The Real-Time Kernel
*
*       (c) Copyright 1992-1998, Jean J. Labrosse, Plantation, FL
*                               All Rights Reserved
*
*                               Configuration for Intel 80x86 (Large)
*
* File : OS_CFG.H
* By   : Jean J. Labrosse
*****
*/

/*
*****
*                               uC/OS-II CONFIGURATION
*                               *****
*/

#define OS_MAX_EVENTS            30    /* Max. number of event control blocks in your application ... */
/* ... MUST be >= 2 */
#define OS_MAX_MEM_PART        30    /* Max. number of memory partitions ... */
/* ... MUST be >= 2 */
#define OS_MAX_QS               30    /* Max. number of queue control blocks in your application ... */
/* ... MUST be >= 2 */
#define OS_MAX_TASKS            30    /* Max. number of tasks in your application ... */
/* ... MUST be >= 2 */

#define OS_LOWEST_PRIO          20    /* Defines the lowest priority that can be assigned ... */
/* ... MUST NEVER be higher than 63! */

#define OS_TASK_IDLE_STK_SIZE   512    /* Idle task stack size (# of 16-bit wide entries) */

#define OS_TASK_STAT_EN         1    /* Enable (1) or Disable(0) the statistics task */
#define OS_TASK_STAT_STK_SIZE   512    /* Statistics task stack size (# of 16-bit wide entries) */

#define OS_CPU_HOOKS_EN         1    /* uC/OS-II hooks are found in the processor port files */
#define OS_MBOX_EN              1    /* Include code for MAILBOXES */
#define OS_MEM_EN              0    /* Include code for MEMORY MANAGER (fixed sized memory blocks) */
#define OS_Q_EN                 0    /* Include code for QUEUES */
#define OS_SEM_EN              1    /* Include code for SEMAPHORES */
#define OS_TASK_CHANGE_PRIO_EN  1    /* Include code for OSTaskChangePrio() */
#define OS_TASK_CREATE_EN       1    /* Include code for OSTaskCreate() */
#define OS_TASK_CREATE_EXT_EN   1    /* Include code for OSTaskCreateExt() */
#define OS_TASK_DEL_EN          1    /* Include code for OSTaskDel() */
#define OS_TASK_SUSPEND_EN      1    /* Include code for OSTaskSuspend() and OSTaskResume() */

#define OS_TICKS_PER_SEC        200    /* Set the number of ticks in one second */

```