Universidad de Pinar del Río
Facultad de Ciencias Técnicas
Dpto. de Telecomunicaciones y Electrónica

PROYECTO INTEGRADOR 3er AÑO TELE.

Título: Sistema de Telecontrol de estaciones de bombeo de agua mediante red inalámbrica.

Autores:

- **José Guerra Carmenate**
- **Leonardo Gonzales Reyes**

Pinar del Río. 2017

Contenido

Resumen

Este proyecto da solución a la necesidad de realizar el telecontrol de estaciones de bombeo de agua, a partir del uso del sistema inteligente DIGILENT CEREBOT (PIC 32MX460F512L) asociado al módulo de red inalámbrica ESP8266. Se diseñó e implementó a nivel de maqueta todo el sistema, incluyendo las interfaces de potencia, el sensor de nivel de agua. Se realizó la programación del microntrolador y la aplicación de escritorio para el control del sistema creada en Python.

Situación Problémica

Es una situación generalizada que las estaciones de bombeo de agua de centros estatales se encuentran a una distancia considerable de la edificación central y dado que dichas estaciones son de operación manual, requieren de un operador que o bien permanezca en la estación o que se desplace hacia ella varias veces al día. Esto ocurre dada la no existencia en el país de sistemas de telecontrol capaces de realizar ya sea de forma autónoma o mediante un gobierno remoto dicha tarea.

Problema

Necesidad de un Sistema de monitoreo y accionamiento remoto de estaciones de bombeo.

Objeto

Sistemas de Telecontrol

Objetivo

Diseñar un sistema inteligente interconectado vía WIFI para monitoreo y accionamiento de estaciones de bombeo.

Hipótesis

La implementación de un sistema inteligente con la Placa de Desarrollo *Digilent Cerebot 32MX4* y el *Módulo ESP8266* que permitirá el monitoreo y accionamiento de estaciones de bombeo de forma remota.

Métodos Científicos

Para el desarrollo de la investigación se utilizaron los siguientes métodos científicos:

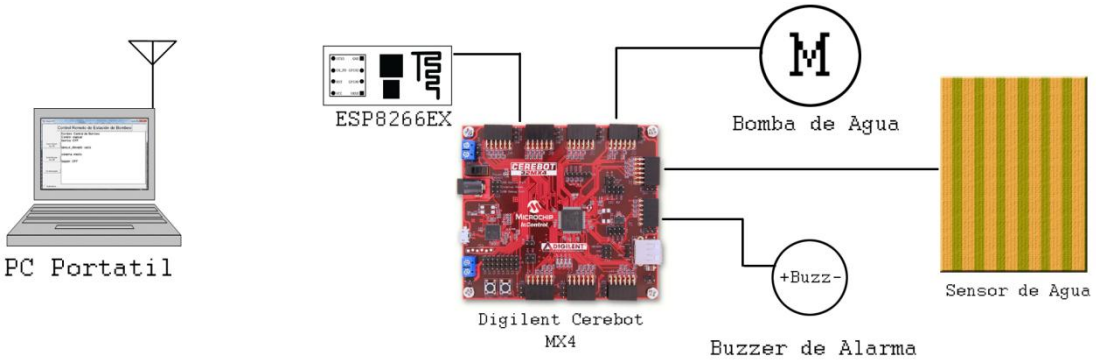
- **Teóricos**
  - **Histórico-Lógico**: Para el estudio teórico de los antecedentes de los sistemas microntroladores PIC de 32 bit.
  - **Inductivo-Deductivo**: Para identificar los principales elementos para el diseño de la aplicación.
- **Empíricos**
  - **Análisis documental**: Para evaluar características, parámetros y requerimientos de las aplicaciones Python y las tecnologías utilizadas
  - **Simulación:** para la comprobación del funcionamiento de la propuesta a desarrollar.

Tareas

- Estudiar Placa de Desarrollo Digilent Cerebot 32MX4
- Estudiar Modulo WIFI ESP8266
- Desarrollar aplicación para la Placa de Desarrollo Digilent Cerebot 32MX4 que gestione pines de Entrada/Salida para monitoreo y accionamiento; y comunicación serie para control del Módulo WIFI ESP8266
- Desarrollar aplicación en entorno Python3 para PC personal o de mesa que presente una Interfaz Gráfica que permita al usuario el monitoreo de parámetros y accionamiento de actuadores utilizando un estructura XML para el intercambio de información.

Desarrollo

1.1 Arquitectura del Sistema



El núcleo del sistema es la Placa de Desarrollo Digilent Cerebot 32MX4 siendo esta la encargada de censar y actuar físicamente en la estación de bombeo. Además actúa como servidor para la aplicación cliente que controla el sistema. Esta placa realiza también la configuración del módulo de red inalámbrica y el intercambio de datos por esta vía. La placa interactúa físicamente con cuatro elementos:

1. Módulo WIFI ESP8266EX
2. Relé de accionado de Bomba de Agua
3. Sensor de presencia de agua
4. Zumbador de Alarma

Módulo WIFI ESP8266EX:

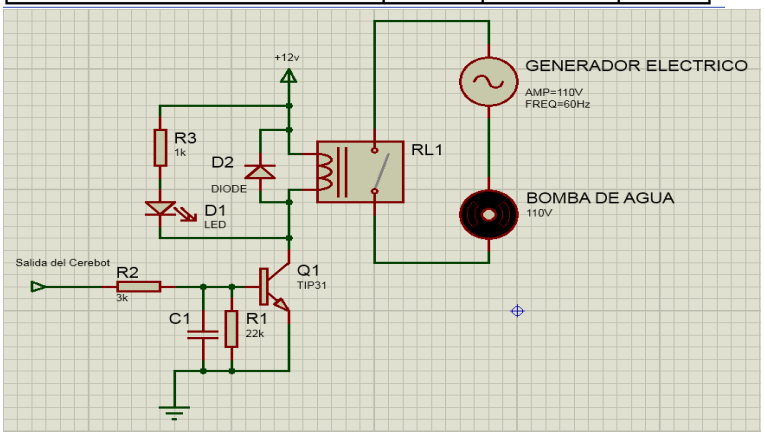
Interactúa con la Digilent Cerebot mediante el UART2 (implementando comunicación Serie RS232). Este módulo está configurado como Punto de Acceso WiFi (AP) por lo tanto permite que cualquier dispositivo capaz de interactuar con redes WiFi se conecte a él. Además, también implementa el protocolo de comunicación TCP/IP de forma autónoma sirviendo como pasarela entre la aplicación cliente que gobierna el sistema alojada en el terminal inalámbrico y el servidor implementado en la Placa de Desarrollo.

Interfaz de potencia para el accionado de Bomba de Agua:

Es el encargado de la activación de la bomba de agua. Su componente principal lo constituye un relé con tensión de bobina de 12V y manejo de corriente por contactos de 50A. El mismo se activa con la presencia de 3.3V procedente de la placa Digilent a la entrada del interfaz, lo que provoca la activación del relé y el encendido de la bomba de agua. Se ha añadido a la entrada de la base un circuito paso bajo para la supresión de las interferencias de 60 Hz y de radiofrecuencia de onda media.

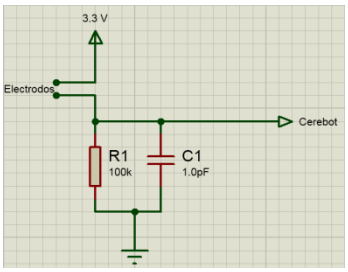
MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Collector − Emitter Voltage TIP31G, TIP32G TIP31AG, TIP32AG TIP31BG, TIP32BG TIP31CG, TIP32CG	V_{CEO}	 40 60 80 100	Vdc
Collector−Base Voltage TIP31G, TIP32G TIP31AG, TIP32AG TIP31BG, TIP32BG TIP31CG, TIP32CG	V_{CB}	 40 60 80 100	Vdc
Emitter−Base Voltage	V_{EB}	5.0	Vdc
Collector Current − Continuous	I_C	3.0	Adc
Collector Current − Peak	I_{CM}	5.0	Adc
Base Current	I_B	1.0	Adc
Total Power Dissipation @ T_C = 25°C Derate above 25°C	P_D	 40 0.32	 W W/°C
Total Power Dissipation @ T_A = 25°C Derate above 25°C	P_D	 2.0 0.016	 W W/°C
Unclamped Inductive Load Energy (Note 1)	E	32	mJ
Operating and Storage Junction Temperature Range	T_J, T_{stg}	−65 to +150	°C



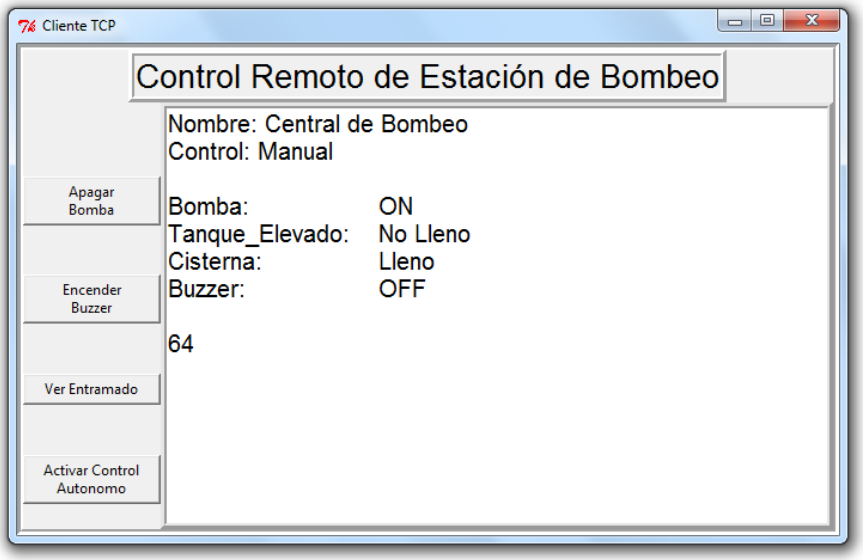
Sensor de presencia de agua:

Es el encargado de censar el estado del tanque (Lleno o No Lleno).



Está compuesto por varios electrodos intercalados de forma tal que cuando estos entren en contacto con el agua se obtenga una resistencia eléctrica que aproxima a los 10K. El resistor R1 funciona como resistencia de Pull-Down, logrando que ante la ausencia de agua el pin de entrada quede conectado a tierra. En caso de presencia de agua tendríamos un divisor resistivo, por lo tanto, en la entrada del Cerebot tendríamos cerca de 3V, suficiente para que este obtenga una lectura de Estado Alto (1 Lógico). El Capacitor C1 se utiliza como filtro pasa-bajo para eliminar posibles ruidos o interferencias provocadas por ondas de radio y de esta forma evitar falsas lecturas.

Aplicación de Gobierno:



Por otro lado tenemos una computadora, ya sea portátil o de escritorio, conectada vía WiFi a la Digilent Cerebot mediante el ESP8266EX. Esta máquina deberá contar con python3 para poder ejecutar la aplicación cliente que permitirá al usuario gobernar y monitorear la estación de agua.

La aplicación de control consta de Tres Botones de control (uno para la bomba, otro para el buzzer y un último para configurar el control de la estación como automático o manual) y además de un área de información donde se puede visualizar el estado actual del sistema. También cuenta con un botón que nos permite ver los datos recibidos por la aplicación "en bruto".

1.2 Placa de Desarrollo Digilent Cerebot 32MX4

La placa Cerebot 32MX4 es una herramienta útil para control embebido y proyectos de robótica tanto para estudiantes como aficionados. Su versátil diseño y su micro-controlador programable permiten acceder a numerosos dispositivos periféricos y programar la placa para múltiples usos. La placa cuenta con numerosos conectores de Entrada/Salida y varias opciones de alimentación. (1).

El Cerebot 32MX4 provee conectores para dispositivos periféricos. Tiene nueve conectores para acoplar módulos periféricos *Digilent Pmod™*. Los módulos periféricos incluyen Puentes H, conversores analógico-digital y digital-analógicos, amplificadores de audio, interruptores, botones, LEDs, así como conversores para facilitar la conexión a RS232, jacks tipo BNC, motores Servo, y más. (1)

Principales características:

- Un micro-controlador PIC32MX460F512L
- Soporte para programación y depuración con el Entorno de Desarrollo Microchip MPLAB
- Nueve Conectores Pmod para placas de Módulos Periféricos (*Digilent Pmod™*)
- Ocho conectores para motores RC hobby Servo
- Soporte para Dispositivos USB 2.0, Host y OTG
- Dos Botones tipo Push
- Cuatro LEDs

- Múltiples opciones para alimentación, incluyendo alimentación USB.
- Protección ESD y protección contra cortocircuito en todos los pines de Entrada/Salida
- EEPROM (IC2) 24LC256
- Conversor Digital-Analógico (IC3) MCP4725

1.2.1 Micro-controlador PIC32MX460F512L

La Serie de 32 bits de micro-controladores PIC32 de Microchip está diseñada para cumplir con los requerimientos de los clientes para lograr mejores características y calidad en sus aplicaciones. (2)
Principales características (3):

- Rango de temperatura de operación de -40ºC a +105ºC
- Rango de voltaje de operación de 2.3V a 3.6V
- 512KB Memoria flash de programa interna
- 32KB de Memoria SRAM interna
- USB 2.0 compliant full-speed On-The-Go (OTG) controller with dedicated DMA channel
- Dos Buses SPI (*Serial Peripheral Interface*)
- Dos Interfaces Series UART
- Dos Interfaces Series I2C
- Cinco temporizadores/Contadores de 16 bits
- Cinco temporizadores de captura de entrada
- Cinco Comparadores/Salida PWM
- Dieciséis entradas analógicas de 10 bits
- Dos comparadores analógicos

1.3 Proceso de Desarrollo de Aplicación para PIC32

MikroC PRO for PIC32 es un IDE especialmente diseñado para programar PIC32 con depurador incluido, que a pesar de no ser compatible con la Digilent Cerebot si resulta un entorno agradable y factible para generar nuestra aplicación con una sintaxis de lenguaje muy similar al lenguaje de programación *C*. Además, este IDE trae consigo un extenso conjunto de bibliotecas para los más variados usos entre los que destacan el trabajo con cadenas de caracteres (strings), comunicación con periféricos y operaciones matemáticas. Mediante este IDE podemos obtener tanto el código en ensamblador (*.asm), como el código ya pre-compilado (*.hex) que se debe 'quemar' hacia el microcontrolador. Luego para grabar este código, ya sea en ensamblador o directamente el hex, utilizamos el IDE MPLB que si es compatible con la Digilent Cerebot.

1.3.1 Manipulación de Puertos como Entrada/Salida Digital

El microcontrolador PIC32MX460F512L cuenta tanto con pines analógicos como digitales, los pines analógicos pueden ser configurados como digitales mediante el registro `AD1PCFG`. Dado que nuestra aplicación no requiere ninguna obtención de datos analógicos, por cuestiones de seguridad se configuran todos los pines como señales digitales:

```
AD1PCFG = 0xFFFF;
```

Con esto nos aseguramos de no intentar escribir o leer por error un valor digital en un pin configurado como analógico.
Para la manipulación de puertos cada pin cuenta con varios registros a continuación se enuncian los utilizados en este trabajo:

- Registros **TRISx**:
  Los registros **TRISx** donde x = {A, B, C,…} son los que nos permiten definir la configuración de los pines como entrada o salida.
  Por ejemplo si deseamos configurar el 3er bit del puerto A como entrada y el 4to bit del puerto B como salida debemos escribir:
  ```
  TRISAbits.TRISA2 = 1;
  TRISBbits.TRISA3 = 0;
  ```
  Importante destacar que al momento de inicio del microcontrolador todos los pines están configurados como entradas.
- Registros **PORTx**:
  Los registros PORTx donde x = {A, B, C,...} son los que nos permiten leer o escribir datos hacia los pines del PIC.
  Por ejemplo si queremos poner un valor en el puerto A del microcontrolador la orden en MikroC sería `PORTA = 0x09;`
- Registros **LATx**:
  Los registros LATx donde x = {A, B, C,...} son los que nos permiten al igual que los registros PORT leer o escribir datos hacia los pines del PIC, pero en caso de lectura estos solo se pueden utilizar para leer los estados de pines configurados como salida pero no para obtener lecturas de pines de entrada.
  Por ejemplo si queremos almacenar un 1 lógico en el 2do bit del puerto A podríamos hacerlo mediante la orden:
  `TRISAbits.TRISA1 = 1;` o `PORTAbits.RA1 = 1;`
  Sin embargo si quisiéramos leer el dato en un pin de entrada solo puede obtenerse mediante el registro PORT.

Existen otros registros que permiten configurar los pines como colector-abierto, configurarlo con resistor pull-up o pull-down, etc. Estos registros son irrelevantes para el trabajo desarrollado y por tanto no se requirió de su uso.
Dado que los puertos del PIC32 en cuestión no coinciden con la distribución de puertos de la placa Digilent Cerebot para ejecutar este trabajo se desarrolló una Biblioteca (*CEREBOT32MX4.h*) compatible con MikroC en la cual se asocian los puertos del Cerebot con los del PIC.
Supongamos que deseamos activar el pin 1 del puerto JA presente en el Cerebot, normalmente sería necesario recurrir al datasheet de la placa para conocer a que pin del microcontrolador está conectado el pin 1 del puerto JA, (que sería el bit 1 del puerto E del microcontrolador). Gracias a la biblioteca desarrollada nos ahorramos este trabajo y simplemente escribimos de forma intuitiva:

```
CEREBOT_PORTJA_1_bit = 1;//Acceso al registro PORT del bit en JA.1
```

Esta librería también tiene definidos el resto de los registros mencionados anteriormente:

- CEREBOT_TRIS:
  Ejemplo: `CEREBOT_TRISJB_4_bit = 0;//configura JB.4 como salida`
- CEREBOT_LAT:
  Ejemplo: `CEREBOT_LATJK_4_bit = 0;//poner a nivel bajo a JK.4`

Además en la biblioteca también hay definiciones para trabajar de forma cómoda con los leds y botones presentes en la placa.

1.3.2 Comunicación Serie (UART) y Biblioteca CyclicBuffer

El microcontrolador en cuestión cuenta con una estructura tipo FIFO en el Receptor de ambos UART, el problema reside en que esta tiene un límite máximo de 8 bytes, los cuales son insuficientes para nuestra aplicación. Por esta razón nos decidimos a implementar la biblioteca CyclicBuffer.

Esta biblioteca contiene una implementación de una estructura tipo FIFO llamada *Buffer* y todo un conjunto de funciones para utilizar. Esta estructura nos permite insertar elementos en ella y luego obtenerlos en el mismo orden en fueron insertados.

Ahora solo se necesita una forma para que los datos recibidos por UART2 (comunicación con módulo WIFI) se almacenen automáticamente en nuestra estructura *Buffer* sin afectar el funcionamiento de nuestra aplicación. Para conseguir esto es necesario crear una variable global tipo *Buffer* y un arreglo de bytes (char) que nuestro Buffer utilizara como almacenamiento*:*

```
Buffer WIFI_input;
char __buff_storage[MAX_BUFF_STORAGE];
```

Luego necesitamos inicializar nuestro Buffer utilizando la función `Buffer_Init` de la siguiente manera:

```
Buffer_Init (&WIFI_in, __buff_storage, MAX_BUFF_STORAGE);
```

Dónde:

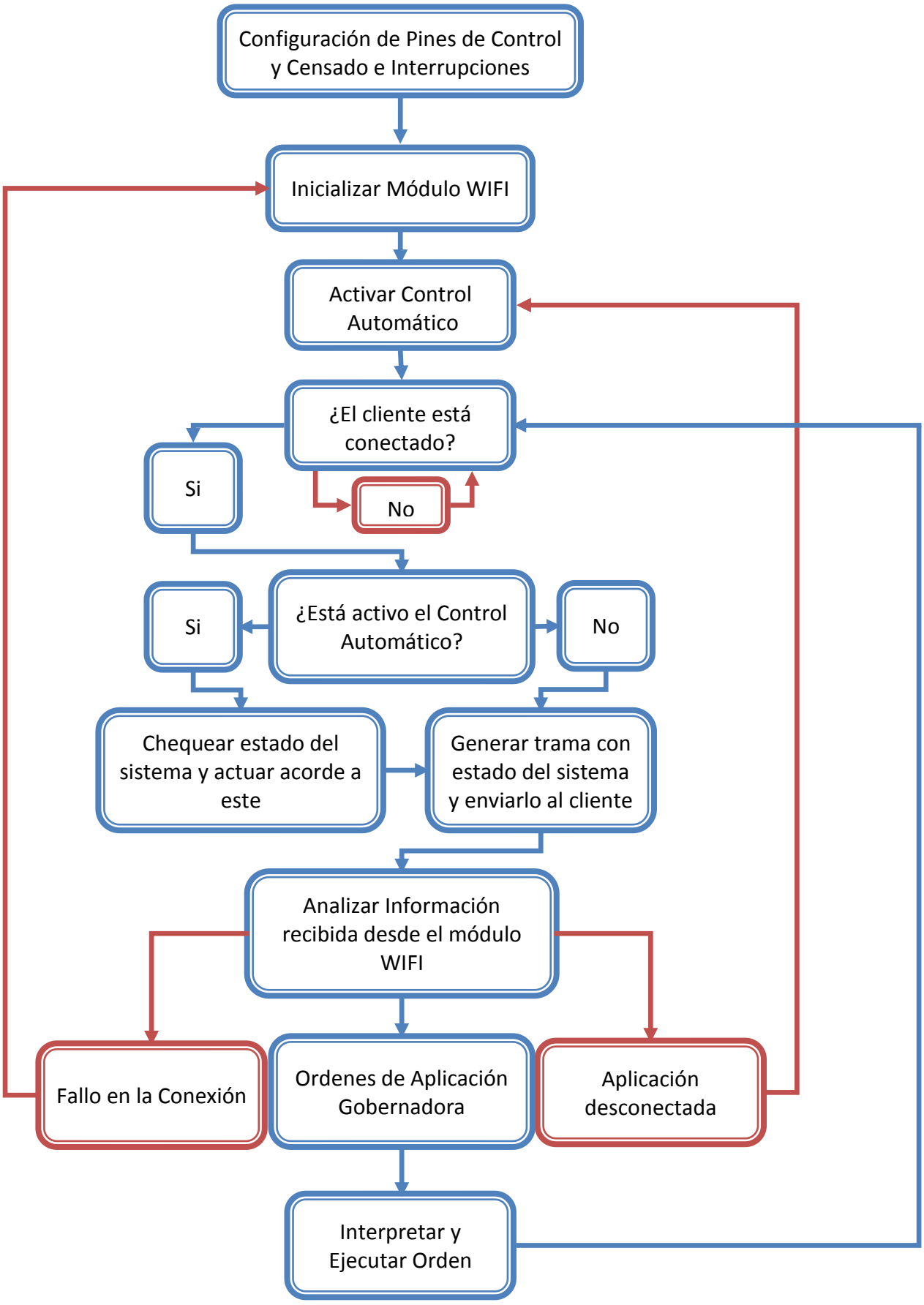
- `WIFI_in` es el *Buffer* a inicializar
- `__buff_storage` es el arreglo utilizado como almacenamiento
- `MAX_BUFF_STORAGE` es la cantidad máxima de elementos que puede almacenar el *Buffer*.

También es necesario configurar y habilitar las interrupciones de UART2 y en la función de interrupción indicarle que el byte leído sea insertado en el buffer:

```
char t = UART2_Read();

Buffer_PushData( &WIFI_in, t );
```

1.3.3 Flujo de Trabajo de la Aplicación Implementada



1.3.3.1 Configuración de Pines de Control y Censado e Interrupciones

En esta etapa se configuran los pines de control de la bomba y de buzzer como pines de salida digital y el pin de sensor de agua como entrada digital. Además se configuran como salida digital los pines correspondientes a los leds presentes en la placa para de esta forma poder utilizarlos como indicadores.

También se configuran los registros necesarios para configurar los puertos de comunicación UART1 y UART2. El UART1 está destinado a la PC, y fue utilizado para depurar el programa y comprobar su correcto funcionamiento. Por otra parte el UART2 es el encargado de comunicarse con el módulo WIFI ESP8266EX. Ambos son configurados a la misma velocidad.

Se activan las interrupciones correspondientes a los puertos antes mencionados y se configura el timer 1 para su uso como alarma (No se activa la interrupción, solo se configura).

1.3.3.2 Inicializar Modulo WIFI

Para inicializar y configurar el módulo WIFI la placa debe enviar un conjunto de comandos con sus respectivos argumentos al módulo vía UART2.

Los comandos y sus argumentos están almacenados en dos arreglos paralelos:

```
const uchar* ATcommand[] = {  //Indices de los comandos AT a ejecutar
  /*CONFIGURACION AT*/
  "AT", //Comprueba el modulo AT en el ESP8266--> ans: OK.
  "AT+RST", //Reinicia el ESP8266 --> ans: OK.
  "ATE", //Configura el Eco de Comandos --> ans: OK.
  /*CONFIGURACION DE WIFI*/
  "AT+CWMODE",//Configura el modo de trabajo del modulo ESP8266 -->
OK.
  "AT+CWSAP",//Configura los parametros de SoftAP. --> OK.
  "AT+CWDHCP",//Configura el DHCP. --> OK.
  /*CONFIGURACION DE TCP/IP*/
  "AT+CIPMUX",//Enable or Disable Multiple Connections  --> ans: OK
  "AT+CIPSERVER",//Deletes/Creates TCP Server   --> OK.
  "AT+CIFSR"//Gets the Local IP Address
};
const char *ATappend[] = { // Parametros de los comandos AT a ejecutar
  "",
  "",
  "0",//0 - Apaga el eco de comandos
  "=2",//Configura el ESP8266 en modo SoftAP.
  "=\"ESTACION\",\"digilentcerebot\",1,2",// configura los parametros
del modo SoftAP.
  "=0,1",//Activa el DHCP para el SoftAP.
  "=1",//Activa conexiones multiples
  "=1,5656",//Crea Servidor TCP en puerto 5656
  ""
};
```

Para enviar estos comandos el Cerebot concatena cada par Comando-parámetros y los envía al módulo WIFI, luego verifica que este le conteste con una afirmación “OK” de lo contrario empezara nuevamente a pasar los comandos desde el primero.

En la Sección Configuración de Modulo WIFI se expone una tabla con los con los comandos, la respuesta esperada y su utilidad.

Además de estos dos arreglos también existe un tercero paralelo a los anteriores:

```
const char *ATnotify[] = { //Notificacion a enviar al PC para cada
Comando
    "Comprobando Modulo AT de ESP8266...\0",
    "Reiniciando Modulo ESP8266...\0",
    "Apagando el Eco de Commandos...\0",
    "Configurando Modulo ESP8266 como SoftAP...\0",
    "Configurando Access Point...\0",
    "Activando DHCP...\0",
    "Activando Conexiones Multiples...\0",
    "Creando Servidor TCP...\0",
    "Obteniendo Direccion IP Local...\0"
};
```

Este arreglo contiene los mensajes a enviar al PC vía UART1 para comprobar el funcionamiento del dispositivo.

1.3.3.3 Activando Control Automático

Esta etapa es muy simple y consta de una única instrucción:

```
Control_estado = 0;
```

Esta instrucción garantiza que el dispositivo siempre se inicie en modo automático (ya que al iniciarse es obvio que la aplicación cliente no está conectada a él).

1.3.3.4 ¿El cliente está conectado?

En esta etapa simplemente esperamos a que el usuario se conecte mediante la aplicación, para esto esperamos a que el módulo WIFI nos envié el string *"0,CONNECT"*. Esto significa que una aplicación estableció comunicación con el modulo.

A pesar de no estar especificado en el diagrama de flujo en esta etapa la aplicación supervisa el estado del sistema para actuar de ser necesario (ya que se encuentra en modo automático).

1.3.3.5 ¿Está activo el Control Automático?

Esta etapa comprueba si el modo de control es automático, en caso positivo supervisa el estado del sistema para actuar de ser necesario.

1.3.3.6 Generar trama con estado del sistema y enviarla al cliente

En esta etapa la aplicación verifica el estado del sistema y elabora un entramado XML que la aplicación de escritorio luego es capaz de decodificar y presentar al usuario en un formato legible. Para genera una trama se llama a la función *GenerarTrama* de la siguiente manera:

```
GenerarTrama(output_state);
```

Esta función almacena en el arreglo que se le pasa como parámetro el entramado con el nuevo estado. Luego la función *WIFI_Send* se encarga de enviar el entramado al módulo ESP8266EX mediante el puerto UART2

```
WIFI_Send(0, output_state, strlen(output_state));
```

1.3.3.7 Analizar Información recibida desde el módulo WIFI

En esta etapa la aplicación analiza los datos recibidos desde el módulo WIFI:

- En caso de un error de conexión (cuando ocurre el modulo envía el string *"CONNECT FAIL"*) se reinicia el Modulo WIFI y se reinicia el ciclo.
- En caso de que el usuario se desconecte (cuando ocurre el modulo envía el string *"0,CLOSED"*) se reinicia el ciclo automáticamente.
- En caso de que se reciba una orden de la aplicación gobernadora esta es procesada y en caso de tener la estructura correcta es ejecutada.

  Cuando un usuario nos envía un conjunto de datos el módulo ESP8266 lo entrega con la siguiente sintaxis:
  +IPD,n1,n2:datos…
  Dónde:
  - n1 es el id del cliente que envía los datos,
  - n2 es la cantidad de bytes enviados
  - datos… son los bytes en si

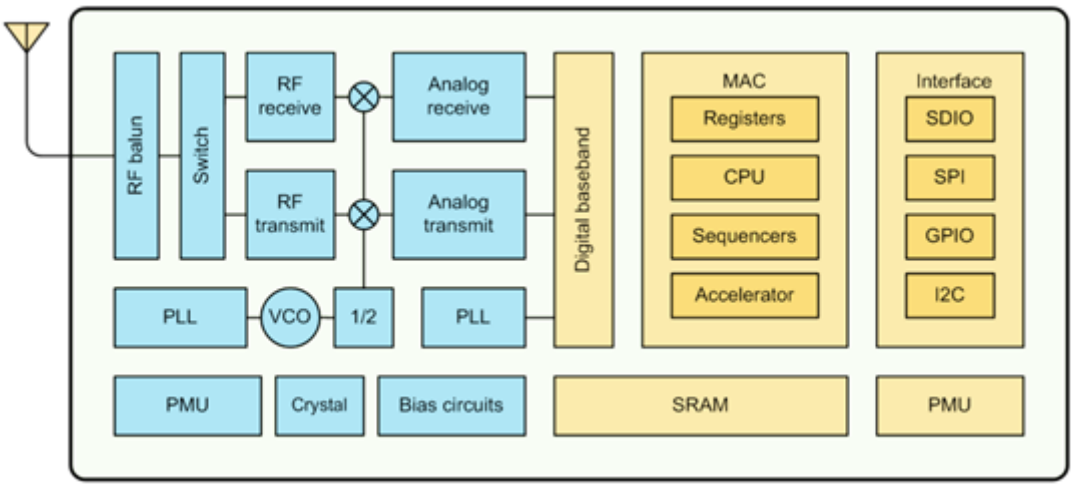
  Entonces para detectar un mensaje buscamos el patron "+IPD," y llamamos a la función `Get_And_Procces_Data`, pasándole como parámetro un puntero a la posición donde se encontró el patrón. Esta función se encarga de obtener los datos y procesarlos. En caso de que los datos recibidos estén corruptos (no cumplan con las especificaciones XML utilizadas) este lo detecta e ignora dichas peticiones.

1.4 Módulo WIFI ESP8266EX

ESP8266EX ofrece una completa y auto contenida solución a la interconexión WIFI; este puede ser utilizado para alojar la aplicación o para funcionar como adaptador WIFI.

Cuando ESP8266EX aloja la aplicación, esta arranca directamente desde una memoria flash externa.

Alternativamente, utilizándolo como Adaptador WIFI, acceso a internet inalámbrico puede ser agregado a cualquier micro-controlador con una simple conexión (Interfaz SPI/SDIO o I2C/UART).



ESP8266EX está entre los mejores circuitos de WIFI integrados en la industria. (4)

Características (4):

- 802.11 b/g/n
- Microcontrolador de 32 bits Integrado
- Conversor Analogico-Digital Integrado
- Protocolo TCP/IP Integrado
- Integrated TR switch, balun, LNA, power amplifier and matching network
- PLL, reguladores, and power management units Integrado
- WiFi 2.4 GHz, con soporte WPA/WPA2
- Soporta varios modos de operación: STA/AP/STA+AP
- Soporte de Smart Link Function para dispositivos Android y iOS
- SDIO 2.0, (H) SPI, UART, I2C, I2S, IR Remote Control, PWM, GPIO
- STBC, 1x1 MIMO, 2x1 MIMO
- A-MPDU & A-MSDU aggregation & 0.4sguard interval
- Deep sleep power <10uA, Power down leakage current < 5uA
- Wake up and transmit packets in < 2ms
- Potencia consumida en Standby es < 1.0mW (DTIM3)
- Potencia de Salida de +20 dBm en modo 802.11b
- Rango de temperatura -40C ~ 125C
- FCC, CE, TELEC, WiFi Alliance, and SRRC certified

Principales Aplicaciones (4):

• Aparatos Domésticos
• Automatización Casera
• Conector de redes
• Control Inalámbrico
• Monitores de Bebe
• Cámaras IP
• Sensor Networks
• Electrónica Móvil
• WiFi Location-aware Devices
• Security ID Tags
• WiFi Position System Beacons

1.4.1 Configuración del módulo WIFI

Para configurar el módulo WIFI ESP8266EX acorde a los requerimientos del sistema la Placa Digilent Cerebot 32MX4 debe enviar los siguientes comandos a dicho modulo:

Comando	Respuesta Esperada	Descripción
AT	OK	Comprueba que el subsistema AT del ESP8266 está funcionando correctamente
AT+RST	OK	Reinicia el Modulo
ATE0	OK	Apaga el Eco de Comandos
AT+CWMODE=2	OK	Configura el modulo en

		modo SoftAP
AT+CWSAP ="ESTACION","digilentcerebot",1,2	OK	Configura el AP con protección WAP, con SSID 'Estacion', y contraseña 'digilentcerebot'
AT+CWDHCP=0,1	OK	Activa el modo DHCP para la distribución de direcciones IP
AT+CIPMUX=1	OK	Activa conexiones Múltiples
AT+CIPSERVER=1,5656	OK	Crea Servidor TCP/IP escuchando en el Puerto 5656

1.5 Formato de intercambio de información estructurada XML.

XML, siglas en inglés de *e**X**tensible **M**arkup **L**anguage* ('lenguaje de marcas extensible'), es un lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible. Proviene del lenguaje SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML) para estructurar documentos grandes. A diferencia de otros lenguajes, XML da soporte a bases de datos, siendo útil cuando varias aplicaciones deben comunicarse entre sí o integrar información.

XML no ha nacido sólo para su aplicación para Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

1.5.1 Ventajas del XML

- Es extensible: Después de diseñado y puesto en producción, es posible extender XML con la adición de nuevas etiquetas, de modo que se pueda continuar utilizando sin complicación alguna.
- El analizador es un componente estándar, no es necesario crear un analizador específico para cada versión de lenguaje XML. Esto posibilita el empleo de cualquiera de los analizadores disponibles. De esta manera se evitan *bugs* y se acelera el desarrollo de aplicaciones.
- Si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarla. Mejora la compatibilidad entre aplicaciones. Podemos comunicar aplicaciones de distintas plataformas, sin que importe el origen de los datos, es decir, podríamos tener una aplicación en Linux con una base de

datos Postgres y comunicarla con otra aplicación en Windows y Base de Datos MS-SQL Server.

- Transformamos datos en información, pues se le añade un significado concreto y los asociamos a un contexto, con lo cual tenemos flexibilidad para estructurar documentos.

1.5.2 Estructura de un documento XML

La tecnología XML busca dar solución al problema de expresar información estructurada de la manera más abstracta y reutilizable posible. Que la información sea estructurada quiere decir que se compone de partes bien definidas, y que esas partes se componen a su vez de otras partes. Entonces se tiene un árbol de trozos de información. Ejemplos son un tema musical, que se compone de compases, que están formados a su vez por notas. Estas partes se llaman *elementos*, y se las señala mediante etiquetas.

Una etiqueta consiste en una marca hecha en el documento, que señala una porción de éste como un elemento. Un pedazo de información con un sentido claro y definido. Las etiquetas tienen la forma `<nombre>`, donde *nombre* es el nombre del elemento que se está señalando.

A continuación se muestra un ejemplo para entender la estructura de un documento XML:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE Edit_Mensaje SYSTEM "Edit_Mensaje.dtd">

<Edit_Mensaje>
   <Mensaje>
      <Remitente>
         <Nombre>Nombre del remitente</Nombre>
         <Mail> Correo del remitente </Mail>
      </Remitente>
      <Destinatario>
         <Nombre>Nombre del destinatario</Nombre>
         <Mail>Correo del destinatario</Mail>
      </Destinatario>
      <Texto>
         <Asunto>
            Este es mi documento con una estructura muy sencilla
            no contiene atributos ni entidades...
         </Asunto>
         <Parrafo>
            Este es mi documento con una estructura muy sencilla
            no contiene atributos ni entidades...
         </Parrafo>
      </Texto>
   </Mensaje>
</Edit_Mensaje>
```

1.5.3 Documentos XML bien formados y control de errores

Los documentos denominados como «bien formados» (del inglés *well formed*) son aquellos que cumplen con todas las definiciones básicas de formato y pueden, por lo tanto, analizarse correctamente por cualquier analizador sintáctico (*parser*) que cumpla con la norma. Se separa esto del concepto de validez que se explica más adelante.

- Los documentos han de seguir una estructura estrictamente jerárquica con lo que respecta a las etiquetas que delimitan sus elementos. Una etiqueta debe estar correctamente incluida en otra, es decir, las etiquetas deben estar correctamente anidadas. Los elementos con contenido deben estar correctamente cerrados.
- Los documentos XML sólo permiten un elemento raíz del que todos los demás sean parte, es decir, solo pueden tener un elemento inicial.
- Los valores atributos en XML siempre deben estar encerrados entre comillas simples o dobles.
- El XML es sensible a mayúsculas y minúsculas. Existe un conjunto de caracteres llamados espacios en blanco (espacios, tabuladores, retornos de carro, saltos de línea) que los procesadores XML tratan de forma diferente en el marcado XML.
- Es necesario asignar nombres a las estructuras, tipos de elementos, entidades, elementos particulares, etc. En XML los nombres tienen alguna característica en común.
- Las construcciones como etiquetas, referencias de entidad y declaraciones se denominan marcas; son partes del documento que el procesador XML espera entender. El resto del documento entre marcas son los datos «entendibles» por las personas.

1.5.4 Partes de un documento XML

Un documento XML está formado por el prólogo y por el cuerpo del documento así como texto de etiquetas que contiene una gran variedad de efectos positivos o negativos en la referencia opcional a la que se refiere el documento, hay que tener mucho cuidado de esa parte de la gramática léxica para que se componga de manera uniforme.

1.5.4.1 Prólogo

Aunque no es obligatorio, los documentos XML pueden empezar con unas líneas que describen la versión XML, el tipo de documento y otras cosas.

El prólogo de un documento XML contiene:

- Una declaración XML. Es la sentencia que declara al documento como un documento XML.
- Una declaración de tipo de documento. Enlaza el documento con su DTD (definición de tipo de documento), o el DTD puede estar incluido en la propia declaración o ambas cosas al mismo tiempo.
- Uno o más comentarios e instrucciones de procesamiento.

EJEMPLO: <?xml version="1.0" encoding="UTF-8"?>

1.5.4.2 Cuerpo

A diferencia del prólogo, el cuerpo no es opcional en un documento XML, el cuerpo debe contener solo un elemento raíz, característica indispensable también para que el documento esté bien formado. Sin embargo es necesaria la adquisición de datos para su buen funcionamiento.

EJEMPLO:

<Edit_Mensaje>
(...)
</Edit_Mensaje>

Elementos

Los elementos XML pueden tener contenido (más elementos, caracteres o ambos), o bien ser elementos vacíos.

Atributos

Los elementos pueden tener atributos, que son una manera de incorporar características o propiedades a los elementos de un documento. Deben ir entre comillas.

Por ejemplo, un elemento «estudiante» puede tener un atributo «Mario» y un atributo «tipo», con valores «come croquetas» y «taleno» respectivamente.

<Estudiante Mario="come croquetas" tipo="taleno">Esto es un día que Mario va paseando…</Estudiante>

1.5.5 Estructura XML utilizada en este proyecto

Para este proyecto se utilizó una estructura XML simple y sin prólogo, pero perfectamente válida ante cualquier parser XML.

Un ejemplo de reporte de estado enviado por la placa Cerebot hacia el cliente sería:

```
<estacion_de_bombeo name="Estación de Bombeo UPR" control="Manual">
<bomba estado="OFF"></bomba>
<tanque_elevado estado="vacio"></tanque_elevado>
<cisterna estado="vacia"></cisterna>
<buzzer estado="OFF"></buzzer>
</estacion_de_bombeo>
```

Este entramado nos comunica que estamos conectados a la **Estación de Bombeo UPR**, que actualmente se encuentra configurada en control **manual**, que la bomba está apagada (**OFF**), que el tanque elevado este **vacío**, la cisterna esta **vacía** y el buzzer está apagado (**OFF**).

Por otra parte los comandos enviados por la aplicación de escritorio a la placa tienen una estructura más simple y concisa para optimizar el tiempo de reconocimiento por parte del microcontrolador.

Un ejemplo seria el siguiente:

```
<bomba estado="ON"></bomba>
```

Esta trama le indica al Cerebot que encienda la bomba.

1.6 Lenguaje de Programación Python3

Python es un lenguaje de programación poderoso y fácil de aprender. Cuenta con estructuras de datos eficientes y de alto nivel y un enfoque simple pero efectivo a la programación orientada a objetos. La elegante sintaxis de Python y su tipado dinámico, junto con su naturaleza interpretada, hacen de éste un lenguaje ideal para

scripting y desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas. (5)
El intérprete de Python y la extensa biblioteca estándar están a libre disposición en forma binaria y de código fuente para las principales plataformas desde el sitio web de Python, http://www.python.org/, y puede distribuirse libremente. El mismo sitio contiene también distribuciones y enlaces de muchos módulos libres de Python de terceros, programas y herramientas, y documentación adicional. (5)
El intérprete de Python puede extenderse fácilmente con nuevas funcionalidades y tipos de datos implementados en C o C++ (u otros lenguajes accesibles desde C). Python también puede usarse como un lenguaje de extensiones para aplicaciones personalizables. (5)

1.6.1 Módulo Tkinter

Tkinter es un binding de la biblioteca gráfica Tcl/Tk para el lenguaje de programación Python, con estos queremos decir que Tk se encuentra disponible para varios lenguajes de programación entre los cuales se encuentra Python con el nombre de Tkinter. Este no es más que una adaptación de esta librería para el lenguaje Python con lo cual usar Tk en otro lenguaje no nos supondrá un inconveniente. (6)

Se considera un estándar para la interfaz gráfica de usuario (GUI) para Python y es el que viene por defecto con la instalación para Microsoft Windows y preinstalado en la mayoría de las distribuciones de GNU/Linux. Con Tkinter podremos conseguir resultados casi tan buenos como con otras librerías gráficas siempre teniendo en cuenta que quizás con otras herramientas podamos realizar trabajos más complejos donde necesitemos una plataforma más robusta, pero como herramienta didáctica e interfaces sencillas nos sobrara, dándonos una perspectiva de lo que se trata el desarrollo de una parte muy importante de una aplicación si deseamos distribuirla. Gracias a Tkinter podemos realizar una interfaz capaz de interactuar con el usuario pidiéndole el ingreso de datos, capturando la pulsación de teclas, movimientos del mouse, entre otras cosas. (6)

Para utilizar el módulo Tkinter de Python lo primero es importarlo:

```
import tkinter as tk
```

Notar que a partir de este momento para acceder a las funciones y objetos de tkinter no es necesario escribir todo el nombre sino solo tk. Ahora tenemos que crear un objeto Tk, que es el equivalente a una ventana convencional, para esto escribimos:

```
root = tk.Tk()
```

Ahora root es una ventana vacía, necesitamos agregarle algún widgets en tkinter tenemos disponibles los siguientes widgets:

- *Label*: Utilizados principalmente para agregar texto a la aplicación.
- *Button*: Son botones y es un medio de comunicación entre el usuario y la interfaz. Es posible especificar una función para que este la ejecute cuando se presionado.
- *Entry*: Es el equivalente al input box de otros lenguajes. Nos proporciona un cuadro en el que el usuario puede agregar texto de una línea
- *Text*: Es similar al *Entry* solo que permite texto de varias líneas

- Existen otras como *Checkbutton*, *Menu*, *Radiobutton*, etc, pero que no fueron utilizadas en este proyecto.

Ahora si queremos agregar algún elemento a nuestra ventana root tenemos que crear este nuevo elemento indicando que pertenece a root y luego necesitamos posicionarlo en la ventana:

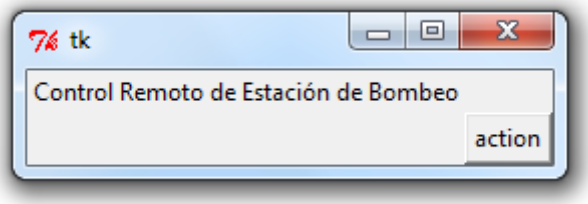
```
titulo = tk.Label( root, text="Control Remoto de Estación de Bombeo")
titulo.grid( row=0,column=0 )
```

La primera línea indica que la etiqueta (Label) 'titulo' está en root y su texto es `"Control Remoto de Estación de Bombeo"`

Luego el método *grid* nos permite posicionar los elementos dentro de root como si esta fuera una cuadricula, entonces el argumento *row* indica la fila y *column* la columna.

```
btn = tk.Button( root, text="OK" )
btn.grid( row=1,column=1 )
```

Con estas nuevas órdenes hemos agregado un botón a nuestra interfaz con el texto action. Ahora solo queda ejecutar `root.mainloop()` este método es el que ejecuta la interfaz gráfica. Esta es la ventana resultante en Windows 7.



Es importante destacar que el aspecto exterior de la ventana puede cambiar ya que este depende del sistema operativo y no del módulo tkinter.

1.6.2 Módulo Socket

El modulo socket de Python nos ofrece una implementación del protocolo de comunicación TCP/IP que nos permitirá de manera sencilla establecer una comunicación mediante dicho protocolo con la Placa de Desarrollo.

Para crear un cliente TCP en Python con unos pocos pasos es más que suficiente.

Primero es necesario importar el modulo:

```
import socket
```

Ahora creamos el socket:

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Los argumentos pasados al constructor indican que el protocolo de comunicación es TCP.

Luego nos conectamos:

```
sock.connect( (ip, port) )
```

Donde **ip** es la Dirección IP donde se encuentra alojado el servidor y **port** es el Puerto en que el servidor está a la escucha.

Para enviar datos utilizamos la función `sock.send(data)` donde **data** es el flujo de bytes a enviar. Para recibir datos utilizamos la función `dat = sock.recv(size)` donde **size** es la máxima cantidad de bytes a recibir, estos bytes serán almacenados en dat. Para cerrar la conexión se utiliza el método `sock.close()`.

1.6.3 Módulo Threading

El módulo threading de python nos ofrece soporte para el trabajo con varios hilos de ejecución.

Es necesario trabajar con varios hilos de ejecución ya que la interfaz gráfica va estar utilizando el hilo principal y por tanto no podremos usarlo para recibir información desde el servidor, entonces necesitamos un hilo que paralelamente al principal obtenga los datos enviados por el servidor y actualice la interfaz gráfica en función de este.

Ahora para crear un hilo de ejecución paralelo al principal primero debemos importar dicho módulo:

```
import threading
```

Una vez importado creamos una clase que herede de la clase *Thread* incluida dentro de threading.

Ejemplo:

```
class my_class( threading.Thread ):

      def __init__( self, master ):

            thread.Thread.__init__( self )

      def run( self ):

            for el in range(1, 10):

                  print('runing…')
```

Como podemos observar es necesario que el método __init__ de nuestra clase llame al __init__ de su clase padre (en este caso *thread.Thread*). También necesitamos definir la función *run* y dentro de esta ingresar el código que deseamos se ejecute en el segundo hilo.

Ahora solo necesitamos crear una instancia de esta clase: `my = my_class()` y ejecutar el método start de la misma `my.start()`

Este método se encarga de ejecutar el código almacenado en run en un hilo diferente e independiente el principal.

1.6.4 Aplicación de Control en Python

Nuestra aplicación está programada en python3 y está compuesta principalmente por tres clases:

- ActualizarEstado:

Esta clase es la encargada de mantener actualizada la interfaz gráfica de usuario. A la hora de inicializar esta clase es necesario pasarle dos argumentos mediante su constructor: el socket por el cual recibirá la información y el tkinter.Frame (interfaz gráfica) que modifica, en nuestro caso es la interfaz principal.

En el método __init__ de esta clase esta implementado un ciclo infinito que recibe información del servidor y modifica la interfaz gráfica de la aplicación. Esto solo se detiene al cerrar la aplicación.

- MainWindow:

  Esta clase hereda de la clase tkinter.Frame y es la encargada de generar la interfaz de usuario. Esto quiere decir que en caso de querer agregar algún widget o realizar alguna modificación a la interfaz es necesario reliazarlo mediante esta clase.

- App

  Esta clase hereda de la clase threading.Thread y es la clase principal de nuestra aplicación.

  En su inicialización esta clase se encarga de:

  - Inicializar su clase padre threading.Thread
  - Crear y conectar el socket con el servidor remoto.
  - Obtener una instancia de la clase MainWindow
  - Configurar los comandos que ejecutara cada botón de dicha instancia

  Además esta clase tiene implementado en el método run la obtención de una instancia de la clase ActualizarEstado a la que pasa como parámetros el socket de conexión y la instancia de MainWindow obtenida anteriormente.

1.6.4.1 Función main

Es la función principal del programa ella se encarga de obtener una instancia de la clase tkinter.Tk llamada **root** y una de la clase App llamada **app**, luego ejecuta el método start de la instancia **app** obtenida para iniciar el refrescamiento de la interfaz gráfica y ejecuta el método mainloop de **root** para crear la ventana.

Conclusiones

Se cumplieron todos los objetivos validando la hipótesis, cumpliendo el objetivo y dando solución al problema.

Se estudiaron las principales características de la placa Cerebot 32MX4 y el micro-controlador PIC32MX460F512L por ser este el eje principal de dicha placa. Fueron estudiados las características del módulo WIFI ESP8266EX fundamentalmente en su uso como adaptador WIFI. Se analizaron las distintas variantes para desarrollar la aplicación para la Cerebot 32MX4 utilizando finalmente el IDE **mikroC PRO for PIC32** de mikroElektronika para programar la aplicación y obtener el archivo ".hex", y el IDE **MPLAB** de Microchip como programador del micro-controlador. Para el desarrollo de la GUI en Python3 se utilizó el módulo *tkinter*, ya que al pertenecer este a los módulos estándar de Python permite que nuestra aplicación sea multiplataforma. Además, se utilizaron los módulos *socket* (para implementar la comunicación TCP/IP) y *threading* (para el trabajo con varios hilos de ejecución).

Bibliografía

1. **Digilent Inc.** Cerebot 32MX4™ Board Reference Manual. *Cerebot 32MX4™ Board Reference Manual.* Agosto 26, 2011.
2. **Microchip Technology Inc.** PIC32MX3XX/4XX Data Sheet. *High-Performance, General Purpose and USB 32-bit Flash Microcontrollers.* 2011.
3. **Microchip Technology Inc.** PIC32 Family Reference Manual. *PIC32 Family Reference Manual.* 2011.
4. —. mikroC PRO for PIC32 Help. 2015.
5. **Espressif Systems IOT Team.** ESP8266EX Datasheet version 4.3. *ESP8266EX Datasheet version 4.3.* 2015.
6. **Espressif Systems.** ESP8266EX Datasheet Version 5.4. 2017.
7. —. ESP8266 AT Instruction Set Version 2.1.0. Copyright © 2017.

ANEXO A: Código de la Aplicación de Control en Python

Fichero cliente_graficov2.py

```
#
# Cliente TCP/IP para control de estacion de bombeo v2
# for Python 3.1.1
#
import tkinter
import tkinter as tk
import tkinter.scrolledtext as scrolledtext
import threading as thread
import socket
import time
IP = '192.168.4.1'
PORT = 5656

class Estado:
      def __init__(self):
            self.bomba = "OFF";
            self.buzzer = "ON";
            self.tanque = "NO Lleno";

def get_attrib_value( label, attrib ): #dada una etiqueta y un
atributo devuelve el valor de este atributo
      """Dada una etiqueta completa y un atributo esta funcion retorna
el valor de este atributo.
            --> En caso de error de sintaxis lo reporta
            Ej1: get_attrib_value( '<pepe come="mucho"/>', 'come')
            retornaria 'mucho'
            Ej2: get_attrib_value( '<pepe come="mucho" comida="pan" >',
'comida')
            retornaria 'pan'
      """
      i = label.find(attrib+"=");
      if i == -1: return "ERROR, no existe atributo ("+attrib+")\n";
      i += len(attrib)+1
      if label[i] != '"': return "ERROR, el valor de '"+attrib+"' no
está en comillas"
      ini = i+1
      end = label.find( '"', ini )
      if end == -1: return "ERROR, falta comilla de cierre"
      return label[ini:end];

def get_label_name( label ):
      """ Dada una etiqueta retorna el nombre o identificador de la
etiqueta
            Ej1: get_label_name( '<rele tipo="chino" estado="ON" >' )
            retornaria 'rele'
      """
      i = label.find('<')
      if i == -1: return "ERROR no existe marca de etiqueta(<)"
      return label[i+1:label.find(' ')]

def decode_state( xml_state ):
      """Esta funcion toma todo el entramado xml generado y enviado
por el sistema
            y lo decodifica en informacion legible para la interfaz de
usuario
      """
      result = ""
```

```
	result +=( "Nombre: " );
	xml_state = xml_state.split('\r\n');
	result += ( get_attrib_value(xml_state[0], 'name') ) + "\n"
	result += ( "Control: " + get_attrib_value(xml_state[0],
'control') )+'\n\n'

	for i in range(1, len(xml_state)-1):
		result += ( get_label_name(xml_state[i])+":\t" );
		if get_label_name(xml_state[i]) != "Tanque_Elevado":
			result += '\t';
		result += ( get_attrib_value(xml_state[i], 'estado')+'\n'
);
	return result

class ActualizarEstado( ):
	""" Esta clase se encarga de recibir el entramado de estado y
teniendo
		en cuenta el nuevo estado actualizar la Interfaz de
Usuario.
	"""
	def __init__(  self, server, mainframe ):
		self.server = server
		self.winframe = mainframe
		self.cont = 0;
		while True:
			print('try')
			try:
				self.new_state_xml = self.server.recv(1024)
				self.cont += 1
			except:
				print('el servidor no envia
nada!!!'.encode('ascii'))
				break;
			print( self.new_state_xml )
			self.dec = decode_state(
self.new_state_xml.decode('ascii') )
			self.tmp = self.dec.split('\n');
			for line in self.tmp:
				if line.find('Bomba') != -1:
					if line.find("ON") != -1:
						self.winframe.switch_bomb['text'] =
"Apagar\nBomba";
					else:
						self.winframe.switch_bomb['text'] =
"Encender\nBomba";

				elif line.find('Buzzer') != -1:
					if line.find("ON") != -1:
						self.winframe.switch_buzzer['text']
= "Apagar\nBuzzer";
					else:
						self.winframe.switch_buzzer['text']
= "Encender\nBuzzer";

				elif line.find('Control') != -1:
					if line.find('Manual') != -1:

	self.winframe.switch_control['text'] = "Activar
Control\nAutonomo"
					elif line.find('Autonomo') != -1:
```

```
	self.winframe.switch_control['text'] = "Activar Control\nManual"
                if self.winframe.debug_active == False:
                        self.winframe.table_of_info['text'] = self.dec
+ '\n' + str(self.cont)
                else:
                        self.winframe.table_of_info['text'] =
self.new_state_xml.decode('ascii') + '\n' + str(self.cont)

                if
self.winframe.switch_control['text'].find("Autonomo") != -1:
                        self.winframe.switch_bomb['state'] = 'normal'
                        self.winframe.switch_buzzer['state'] = 'normal'
                else:
                        self.winframe.switch_bomb['state'] = 'disabled'
                        self.winframe.switch_buzzer['state'] =
'disabled'


class MainWindow(tk.Frame):
	""" Esta clase crea el Frame de la Interfaz de Usuario"""
	def __init__( self ):
		tk.Frame.__init__( self )
		self['relief'] = 'groove'
		self['borderwidth'] = 5
		self.pack();
		#Vars
		self.debug_active = False

		#WindowConfig
		self.master.title( "Cliente TCP" )
		self.ConectionFrame_is_active = False;

		self.titulo = tk.Label( self, text="Control Remoto de
Estación de Bombeo")
		self.titulo['font'] = self.titulo['font'].split()[0] + "
20"
		self.titulo['relief'] = 'ridge'
		self.titulo['borderwidth'] = 5
		self.titulo.grid( sticky=tk.N, columnspan=3 )

		self.switch_bomb = tk.Button( self, text="Switch
Bomb\nOn/Off")
		self.switch_bomb.grid(row = 2, column=0, sticky=tk.W )
		self.switch_bomb['width'] = '15';

		self.switch_buzzer = tk.Button( self, text="Switch
Buzzer\nOn/Off" )
		self.switch_buzzer.grid( row = 3, column=0, sticky=tk.W )
		self.switch_buzzer['width'] = '15';

		self.switch_active_debug = tk.Button( self, text="Ver
Entramado" )
		self.switch_active_debug['width'] = '15'
		self.switch_active_debug.grid( row=4, column=0, sticky=tk.W
)

		self.switch_control = tk.Button( self, text="Activar
Control\nAutonomo" )
		self.switch_control['width'] = '15'
		self.switch_control.grid( row = 5, column=0, sticky=tk.W );
```

```
            self.table_of_info = tk.Label( self )
            self.table_of_info['font'] =
self.table_of_info['font'].split()[0] + ' 15'
            self.table_of_info['bg'] = 'white'
            self.table_of_info['borderwidth'] = 5
            self.table_of_info['height'] = 15
            self.table_of_info['state'] = 'normal'
            self.table_of_info['justify'] = 'left'
            self.table_of_info['width']  = 50
            self.table_of_info['relief'] = 'ridge'
            self.table_of_info['anchor'] = 'nw'
            self.table_of_info.grid( sticky=tk.W, row = 1, column=1,
rowspan=5 );


class App( thread.Thread ):
      def __init__( self, master ):
            thread.Thread.__init__( self )
            self.client = socket.socket();
            print( "ip: " +IP + " port: " + str(PORT) )
            self.client.connect( (IP, PORT) )
            self.current_state = Estado();
            self.mainFrame = MainWindow()
            self.mainFrame.pack()
            self.mainFrame.switch_bomb['command']=self.sw_bomb_func
            self.mainFrame.switch_buzzer['command']=self.sw_buzzer_func

      self.mainFrame.switch_control['command']=self.sw_control_func

      self.mainFrame.switch_active_debug['command']=self.sw_debug_acti
ve_func

      def run( self ):
            ActualizarEstado( self.client, self.mainFrame )

      def send_data(self, sms):
            #self.client2 = socket.socket();
            #self.client2.connect( (IP,PORT) );
            self.client.send( bytes(sms.encode('ascii')) )
            #self.client2.close()

      def sw_bomb_func(self):
            if self.mainFrame.switch_bomb['text'].find( 'Encender' ) !=
-1:
                  self.send_data( '<Bomba estado="ON"/>' )
            else:
                  self.send_data( '<Bomba estado="OFF"/>' )
            print( 'sw_bomb' )
      def sw_buzzer_func(self):
            if self.mainFrame.switch_buzzer['text'].find( 'Encender' )
!= -1:
                  self.send_data( '<Buzzer estado="ON"/>' )
            else:
                  self.send_data('<Buzzer estado="OFF"/>')
            print( 'sw_buzzer' )
      def sw_control_func(self):
            if self.mainFrame.switch_control['text'].find( 'Autonomo' )
!= -1:
                  self.send_data( '<Control estado="Autonomo"/>' )
            else:
```

```
                self.send_data( '<Control estado="Manual"/>' )
            print( 'sw_control' )

      def sw_debug_active_func(self):
            if self.mainFrame.switch_active_debug['text'].find(
'Entramado' ) != -1:
                self.mainFrame.debug_active = True;
                self.mainFrame.switch_active_debug['text'] = 'Ver
Informe'
            else:
                self.mainFrame.debug_active = False;
                self.mainFrame.switch_active_debug['text'] = 'Ver
Entramado'

def main():
      root = tk.Tk()

      app = App( root )
      app.start()
      root.mainloop()
      app.client.close()



if __name__ == '__main__':
      main()
```

ANEXO B: Código de Programación del Microcontrolador

Fichero Control_Inalambrico_Estacion_de_Bombeo.c:

```
/************************************************
 * Universidad de Pinar del Rio
 * Facultad de Ciencias Técnicas
 * Carrera de Telecomunicaciones y Electrónica
 ************************************************
 * Proyecto: Proyecto Final de 3er Año
 * Nombre  : Control Remoto de Estación de Bombeo
 * Autores :
 *     - Jose Guerra Carmenate
 *     - Leonardo Gonzales Reyes
 * Placa de Desarrollo:
 *     - Digilent Cerebot 32MX4 @ 80 MHz
 ************************************************/
/**Bibliotecas**/
#include "CEREBOT32MX4.h"
#include "CyclicBuffer.h"
#include <stdbool.h>

typedef unsigned char uchar;
typedef unsigned int uint;
typedef unsigned short ushort;

/***** Modulo WIFI Begin *****/
/*Constantes*/
const uchar* ATcommand[] = {  //Indices de los comandos AT a ejecutar
  /*CONFIGURACION AT*/
  "AT", //Comprueba el modulo AT en el ESP8266--> ans: OK.
  "AT+RST", //Reinicia el ESP8266 --> ans: OK.
  "ATE", //Configura el Eco de Comandos --> ans: OK.
  /*CONFIGURACION DE WIFI*/
  "AT+CWMODE",//Configura el modo de trabajo del modulo ESP8266 -->
OK.
  "AT+CWSAP",//Configura los parametros de SoftAP. --> OK.
  "AT+CWDHCP",//Configura el DHCP. --> OK.
  /*CONFIGURACION DE TCP/IP*/
  "AT+CIPMUX",//Enable or Disable Multiple Connections  --> ans: OK
  "AT+CIPSERVER",//Deletes/Creates TCP Server   --> OK.
  "AT+CIFSR"//Gets the Local IP Address
};
const char *ATappend[] = { // Parametros de los comandos AT a ejecutar
  "",
  "",
  "0",//0 - Apaga el eco de comandos
  "=2",//Configura el ESP8266 en modo SoftAP.
  "=\"ESTACION\",\"digilentcerebot\",1,2",// configura los parametros
del modo SoftAP.
  "=0,1",//Activa el DHCP para el SoftAP.
  "=1",//Activa conexiones multiples
  "=1,5656",//Crea Servidor TCP en puerto 5656
  ""
};
const char *ATnotify[] = { //Notificacion a enviar al PC para cada
Comando
   "Comprobando Modulo AT de ESP8266...\0",
   "Reiniciando Modulo ESP8266...\0",
   "Apagando el Eco de Commandos...\0",
   "Configurando Modulo ESP8266 como SoftAP...\0",
```

```
   "Configurando Access Point...\0",
   "Activando DHCP...\0",
   "Activando Conexiones Multiples...\0",
   "Creando Servidor TCP...\0",
   "Obteniendo Direccion IP Local...\0"
};
/*Funciones*/
int WIFI_Init();
void WIFI_Wait_Init();
bool WIFI_Send( unsigned short user, uchar *message, uint size );
/*****Modulo WIFI End*****/

/*****Buffer and Stack Begin*****/
//Buffer Vars
const int MAX_BUFF_STORAGE = 512;
Buffer WIFI_input;
uchar __buff_storage[MAX_BUFF_STORAGE+10];
/*****Buffer End*****/

/*****Metodos Generales Begin*****/
/*SETUP*/
void Setup();//Inicializa el uC
void Timer1_Config();//Configura el Timer1 para la alarma
bool Get_And_Procces_Data( const char *dat );
/*Send to PC*/
void Send_To_PC( const uchar *s );//Envia un string via UART1
void uint_to_str( unsigned int n, char *out );
/*****Metodos Generales End*****/

/*****Variables Globales del Sistema Begin*****/
#define Tanque_estado CEREBOT_PORTJE_7_bit   //( 0-No Lleno, 1-Lleno )
#define Buzzer_estado CEREBOT_LATJE_8_bit   //( 0-OFF, 1-ON )
#define Bomba_estado CEREBOT_LATJE_9_bit    //( 0-OFF, 1-ON )
#define Cisterna_estado         1            //( 0-Vacio, 1-No Vacio )
--->NO USADO.
ushort Control_estado=1;                     //( 0-manual, 1-
automatico )
char output_state[600];//aqui se almacena la trama que se envia al
cliente

char input[600];       //aqui se almacenan los datos recividos desde
el modulo
                        //wifi al sacarlos del buffer
bool user_connected = 0;//indica si la aplicacion de control esta
conectada
const uchar *NO_LLENO = "No Lleno",
            *LLENO = "Lleno";
const uchar *ROOT_LABEL = "Estacion_de_bombeo",
            *TANQUE_LABEL = "Tanque_Elevado",
            *CISTERNA_LABEL = "Cisterna",
            *BOMBA_LABEL = "Bomba",
            *BUZZER_LABEL = "Buzzer",
            *CONTROL_LABEL = "Control";
const uchar *Nombre_Estacion = "Central de Bombeo";
/*****Variables Globales del Sistema End*****/

uchar* getStringDeEstado( ushort num_de_estado ){
        if( num_de_estado == 0 ) return (uchar*)NO_LLENO;
        if( num_de_estado == 1 ) return (uchar*)LLENO;
        return "";
}
```

```
/*****Analisis de Trama Begin*****/
//Analiza si t es una trama valida (1-valida, 0-invalida)
//int Check( const uchar *t );

//Crea una trama con el estado actual del sistema
void GenerarTrama( uchar *trama );

/* Crea etiqueta de apertura en 'to' con nombre 'name' y dos
attributo( attrib1 , attrib2 ) con sus
 * respectivos valores( value1 , value2 ). Retorna la cantidad de
caracteres agregados.
 * Si attrib(1|2) = "" este no se agrega a la etiqueta.
 */
int getOpenLabel( uchar *to, uchar *name, uchar *attrib1, uchar
*value1 , uchar *attrib2 , uchar *value2  );

//Crea etiqueta de Cierre en to con nombre name. Retorna la cantidad
de caracteres agregados.
int getCloseLabel( uchar *to, uchar *name );

/**Utilizamos los valores hash para comparar si dos strings son
iguales**/
const uint hash_P = 53;//Numero Primo utilizado como Base en el Hash
uint get_Hash( uchar *s, int len ); //Devuelve un hash del string s
/*****Analisis de Trama End*****/


/*****Interrupciones Begin**/
/*Interrupciones UART2*/
void uart2_interrupt() iv IVT_UART_2 ilevel 7 ics ICS_SRS { ///MODULO
WIFI
  if( IFS1bits.U2RXIF  ){
    uchar t = UART2_Read();
    Buffer_PushData( &WIFI_input, t );
    IFS1bits.U2RXIF = 0;
  }
  if( IFS1bits.U2TXIF ){
    IEC1bits.U2TXIE = 0;        // Disable UART2 TX ISR
    IFS1bits.U2TXIF = 0;
  }
}
/*Interrupciones UART1*/
void uart1_interrupt() iv IVT_UART_1 ilevel 6 ics ICS_AUTO {//
Terminal Serie en PC
  if( IFS0bits.U1RXIF ){
    //UART1_Read();
    UART2_Write( UART1_Read() );
    IFS0bits.U1RXIF = 0;
  }
  if( IFS0bits.U1TXIF ){
      IEC0bits.U1TXIE = 0;        // Disable UART1 TX ISR
      IFS0bits.U1TXIF = 0;
  }
}
/*Interrupcion Timer1*/
void timer1_interrupt() iv IVT_TIMER_1 ilevel 5 ics ICS_SOFT {
  Buzzer_estado = ~Buzzer_estado;
  T1IF_bit = 0; // Clear T1IF.
}
```

```
/*****Interrupciones End*****/

void main(){
  char *ptr; //puntero para trabajo
  char num[10];
  delay_ms(200);
  user_connected = 0;
  Setup();
  delay_ms(1000);
  WIFI_Wait_Init();

  //Timer1_Enable();
  Delay_ms(1000);
  while(1){
    while( !user_connected ){
      CEREBOT_LED2_LAT = ~CEREBOT_LED2_LAT;
      if( Buffer_Data_Ready( &WIFI_input ) ){
        Buffer_GetAllData( &WIFI_input, input );
        Send_To_PC("no_conected input: ");
        Send_To_PC(input);
        if( strstr( input, "0,CONNECT" ) != 0 ){
          user_connected = 1;
          Control_estado = 0;
        }
      }
     if( Control_estado == 1 ){ //Si esta en automatico
        if( Tanque_estado == 1 ){ //y el tanque esta lleno
        Bomba_estado = 0;                                   //apaga bomba
        T1CONbits.ON = 1;                                   //enciende
alarma
        }
        else{                                               //sino esta
lleno
          T1CONbits.ON = 0;                                 //apaga alarma
          Buzzer_estado = 0;
        }
      }
      delay_ms(400);
    }
     GenerarTrama(output_state);
    WIFI_Send(0, output_state, strlen(output_state));
    //Buffer_GetAllData( &WIFI_input, input );
    delay_ms(300);

    delay_ms(400);
    CEREBOT_LED3_LAT = ~CEREBOT_LED3_LAT;
    if( Buffer_Data_Ready( &WIFI_input ) ){
      Buffer_GetAllData( &WIFI_input, input );
      /*Send_To_PC("input("); int_to_str((uint)strlen(input), num);
      Send_To_PC( num );
      Send_To_PC( "): " );
      Send_To_PC(input);   */

      ptr = strstr(input, "CONNECT FAIL");
      if( ptr ){
        user_connected = 0;
        delay_ms( 500 );
        WIFI_Wait_Init();
        continue;
      }
      ptr = strstr( input, "0,CLOSED" );
```

```
      if( ptr ){
        user_connected = 0;
        Control_estado = 1;
        continue;
      }
      ptr = strstr(input, "+IPD,");
      if( ptr ){ //Datos provenientes del cliente
          Get_And_Procces_Data( ptr );
      }
      if( Control_estado == 1 ){ //Si esta en automatico
        if( Tanque_estado == 1 ){ //y el tanque esta lleno
        Bomba_estado = 0;                                  //apaga bomba
        T1CONbits.ON = 1;                                  //enciende
alarma
        }
        else{                                              //sino esta
lleno
          T1CONbits.ON = 0;                                //apaga alarma
          Buzzer_estado = 0;
        }
      }

    }

  }
}//End Main

/*****IMPLEMENTACIONES*****/
void Timer1_Config(){
//Configuracion de Timer1
  TMR1 = 0;         //inicializa en cero el valor del timer
  PR1 = 65535;      //fija el  periodo del timer

  T1IP0_bit = 1;    //fija la prioridad
  T1IP1_bit = 0;    //del timer1
  T1IP2_bit = 1;    //a 5 de 7

  TCKPS0_bit = 1;   //Set Timer Input Clock
  TCKPS1_bit = 1;   //Prescale value to 1:256

  T1IE_bit = 1;     //habilita las interrupciones por timer1
}

void Setup(){
  AD1PCFG = 0xFFFF; // config. pines analogicos como digitales
  //JTAGEN_bit = 0 ;  // Deshabilita el JTAGEN (debuger)
  /**Config. pines de leds onboard y apagando los leds**/
  CEREBOT_LED1_TRIS = 0;
  CEREBOT_LED1_LAT = 0;
  CEREBOT_LED2_TRIS = 0;
  CEREBOT_LED2_LAT = 0;
  CEREBOT_LED3_TRIS = 0;
  CEREBOT_LED3_LAT = 0;
  CEREBOT_LED4_TRIS = 0;
  CEREBOT_LED4_LAT = 0;
  /**configurando los pines de control**/
  CEREBOT_TRISJE_7_bit = 1;//config. como entrada el pin del Sensor de
Precencia de Agua
  CEREBOT_TRISJE_8_bit = 0;//config. como salida el pin de control del
Buzzer
  Buzzer_estado = 0;      //Apagando el Buzzer
```

```
  CEREBOT_TRISJE_9_bit = 0;//config. como salida el pin de control de
la Bomba de Agua
  Bomba_estado = 0;       //Apagando la Bomba


  DisableInterrupts();                // Tell CPU to stop paying
attention to interrupts

  //Configuracion de UART1
  UART1_Init(115200);
  INTCONbits.MVEC = 1;                // Multi Vector interrupts
  U1STAbits.URXISEL = 0;              // 0x = Interrupt flag bit is
set when a character is received
  U1STAbits.UTXISEL = 1;              // 01 = Interrupt flag bit is
set when all characters have been transmitted
  IPC6bits.U1IP = 6;                  // Set UART1 priority 6 of 7
  IPC6bits.U1IS = 0;                  // Set UART1 sub priority to 0
  IFS0bits.U1RXIF = 0;                // Clear UART1 RX interrupt flag
  IFS0bits.U1TXIF = 0;                // Clear UART1 TX interrupt flag
  IEC0bits.U1RXIE = 1;                // Enable UART1 RX ISR

//Configuracion de UART2
  UART2_Init(115200);
  INTCONbits.MVEC = 1;                // Multi Vector interrupts
  U2STAbits.URXISEL = 0;              // 0x = Interrupt flag bit is
set when a character is received
  U2STAbits.UTXISEL = 1;              // 01 = Interrupt flag bit is
set when all characters have been transmitted
  IPC8bits.U2IP = 7;                  // Set UART2 priority 7 of 7
  IPC8bits.U2IS = 0;                  // Set UART2 sub priority to 0
  IFS1bits.U2RXIF = 0;                // Clear UART2 RX interrupt flag
  IFS1bits.U2TXIF = 0;                // Clear UART2 TX interrupt flag
  IEC1bits.U2RXIE = 1;                // Enable UART2 RX ISR

  Timer1_Config();

  EnableInterrupts();

  //Configuracion de Buffer de UART2 ( Comunicacion con Modulo Wifi)
  Buffer_Init( &WIFI_input, __buff_storage, MAX_BUFF_STORAGE );
}

void Send_To_PC( const uchar *s ){
  int i = 0;
  while( s[i] != 0 )
    UART1_Write( s[i++] );
}

int WIFI_Init(){
  uchar tmp[50];
  int i;
  Send_To_PC( "Inicializando WIFI:\r\n" );
  delay_ms(1000);
  for( i = 0; i < 9; i++ ){
    CEREBOT_LATJK_1_bit = ~CEREBOT_LATJK_1_bit;
    strcpy( tmp, (uchar*)ATcommand[ i ] );// tmp = ATcommand[ i ];
    strcat( tmp, (uchar*)ATappend[i] );   // tmp = tmp +  ATappend[i];
    UART2_Write_Text( tmp );
    UART2_Write_Text( "\r\n" );
    delay_ms(500);
    Send_To_PC(ATnotify[i]);
```

```
    delay_ms(500);
    Buffer_GetAllData( &WIFI_input, input );

    if( strstr( input, "ERROR" ) || strlen(input) == 0 ){ //Compruebo
que el modulo responda sin ERRORES
        Send_To_PC( "ERROR!!!\r\n" );
        return 0;
    }
    else{
         Send_To_PC( "OK!!!\r\n" );
    }
    //Send_To_PC( sms );
    //delay_ms(1000);
  }
  return 1;
}

void GenerarTrama( uchar *t ){
        int sz = 0;//tamaño de la trama
        uchar *aux;
        sz += getOpenLabel( t, ROOT_LABEL, "name", Nombre_Estacion,
"control", ((Control_estado==0)?"Manual":"Autonomo") );
        t[sz++] = '\r';t[sz++] = '\n';

        aux = (Bomba_estado==0)?"OFF":"ON";
        sz += getOpenLabel( t+sz, (uchar*)BOMBA_LABEL,"estado",aux,
"","" );
        sz += getCloseLabel( t+sz, (uchar*)BOMBA_LABEL );
        t[sz++] = '\r';t[sz++] = '\n';

        aux = getStringDeEstado( (bool)Tanque_estado );
        sz += getOpenLabel( t+sz, (uchar*)TANQUE_LABEL, "estado",
aux,"","" );
        sz += getCloseLabel( t+sz, (uchar*)TANQUE_LABEL );
        t[sz++] = '\r';t[sz++] = '\n';

        aux = getStringDeEstado( Cisterna_estado );
        sz += getOpenLabel( t+sz, (uchar*)CISTERNA_LABEL, "estado",
aux,"","" );
        sz += getCloseLabel( t+sz, (uchar*)CISTERNA_LABEL );
        t[sz++] = '\r';t[sz++] = '\n';

        aux = (Buzzer_estado==0)?"OFF":"ON";
        sz += getOpenLabel( t+sz, (uchar*)BUZZER_LABEL, "estado", aux,
"","" );
        sz += getCloseLabel( t+sz, (uchar*)BUZZER_LABEL );
        t[sz++] = '\r';t[sz++] = '\n';

        sz += getCloseLabel( t+sz, ROOT_LABEL );
}

int getOpenLabel( uchar *to, uchar *name, uchar *attrib1, uchar
*value1 , uchar *attrib2 , uchar *value2 ){
        int sz;
        sz = 1+strlen(name);
        strcpy( to, "<" );
        strcpy( to+1, name );
        if( strlen(attrib1) > 0 ){
                strcpy( to+sz, " " ); sz++;
                strcpy( to+sz, attrib1 ); sz += strlen(attrib1);
                strcpy( to+sz, "=\"" ); sz += 2;
```

```
                strcpy( to+sz, value1 ); sz += strlen(value1);
                strcpy( to+sz, "\"" ); sz ++;

        }
        if( strlen(attrib2) > 0){
                strcpy( to+sz, " " ); sz++;
                strcpy( to+sz, attrib2 ); sz += strlen(attrib2);
                strcpy( to+sz, "=\"" ); sz += 2;
                strcpy( to+sz, value2 ); sz += strlen(value2);
                strcpy( to+sz, "\"" ); sz ++;
        }
        strcpy( to+sz, ">" ); sz++;
        return sz;
}

int getCloseLabel( uchar *to, uchar *name ){
        int sz = 2+strlen(name);
        strcpy( to, "</" );
        strcpy( to+2, name );
        strcpy( to+sz, ">" ); sz++;
        return sz;
}

uint getHash( char *s, int len ){
        uint hash = 1, i;
        for( i = 0; s[i] != 0; i++ ){
                hash = hash + hash_P*s[i];
        }
        return hash;
}

void Write_to_Wifi( const uchar *s ){
  int i = 0;
  while( s[i] != 0 )
    UART2_Write( s[i++] );
}

void WIFI_Wait_Init(){
  while( !WIFI_Init() );
  Buzzer_estado = 1;
  delay_ms(200);
  Buzzer_estado = 0;
}

void uint_to_str( unsigned int n, char *out ){
     int len;
     if(n == 0){
          out[0] = '0';
          out[1] = '\0';
          return;
     }
     len = log10(n)+1;
     out[len] = '\0';
     while( n ){
       len--;
       out[len] = '0' + (n%10);
       n /= 10;
     }

}
bool WIFI_Send( unsigned short user, const uchar *message, unsigned
```

```
int size ){
  char aux[10];
  CEREBOT_LED4_LAT = 0;
  uint_to_str( size, aux );
  Write_to_Wifi( "AT+CIPSEND=0," );
  Write_to_Wifi( aux );
  Write_to_Wifi( "\r\n" );


  delay_ms(200);
  Write_to_Wifi(message);
  delay_ms(200);
  CEREBOT_LED4_LAT = 1;
  return true;
}

char p_data[50];  //arreglo temporal para procesar datos
char p_label[20]; //almacena la etiqueta del comando recibido
char p_estado[15];//almacena el valor del atributo estado del comando
bool Get_And_Procces_Data( const char *dat ){
  int len, p_data_id = 0, p_label_id = 0, p_estado_id = 0;
  //Get Data
  dat += 7; //me posiciono en el principio de la cant de datos ej:
+IPD,0,120: ....
            //
^
  for( p_data_id =0; *dat != ':'; p_data_id++, dat++ )  //guardo el
numero en p_data
    p_data[p_data_id] = *dat;
  p_data[p_data_id] = '\0';

  len = atoi(p_data);
  dat++;
  for( p_data_id = 0; p_data_id < len; p_data_id++ ){
        p_data[p_data_id] = *dat;
        dat++;
  }
  p_data[p_data_id] = '\0';
  Send_To_PC( "Get Data:" );
  Send_To_PC( p_data );
  Send_To_PC( "\r\n" );

  //Process Data
  p_data_id =0;
  while( isspace(p_data[p_data_id]) ) //salto espacios en blanco
    p_data_id++;

  if( p_data[ p_data_id ] != '<' ){//ERROR falta llave(<) de apertura
      Send_To_PC( "Get And Process ERROR 1\r\n" );
      return 0;
  }
  p_data_id++;
  //get label
  while( !isspace( p_data[ p_data_id ] ) ){
    p_label[p_label_id++] = p_data[p_data_id];
    p_data_id++;
  }
  p_label[p_label_id] = '\0';

  //get attrib estado
  while( isspace(p_data[p_data_id]) ) //salto espacios en blanco
```

```
    p_data_id++;

  while( p_data[ p_data_id ] != '=' ){ //guardo el attributo
    p_estado[p_estado_id++] = p_data[p_data_id];
    p_data_id++;
  }
  p_estado[p_estado_id] = '\0';
  if( strcmp( p_estado, "estado" ) != 0 ){//ERROR atributo incorrecto
      Send_To_PC( "Get And Process ERROR 2\r\n" );
      return 0;
  }
  p_data_id++;
  if( p_data[ p_data_id ] != '"' ){//ERROR falta comilla
      Send_To_PC( "Get And Process ERROR 3\r\n" );
      return 0;
  }
  p_data_id++;
  p_estado_id = 0;
  while( p_data[p_data_id] != '"' ){
    p_estado[p_estado_id++] = p_data[p_data_id];
    p_data_id++;
  }
  p_estado[p_estado_id] = '\0';
  //Send_to_PC("label:");
  //Send_To_PC(p_label);
  //Send_To_PC("\r\n");

  //Send_To_PC("value: ");
  //Send_To_PC( p_estado );
  //Send_To_PC( "\r\n" );
  p_data_id++;
  while( isspace(p_data[p_data_id]) ) //salto espacios en blanco
    p_data_id++;

  if( p_data[p_data_id] != '/' || p_data[p_data_id+1] != '>' ){//ERROR
falta cierre
      Send_To_PC( "Get And Process ERROR 4\r\n" );
  }
  if( strcmp( p_label, BOMBA_LABEL ) == 0 ){
      //Send_To_PC("Es Bomba\r\n");
      if( strcmp( p_estado, "ON" ) == 0 ){
         // Send_To_PC( "estoy encendiendo!!!\r\n" );
          Bomba_estado = 1;
      }
      else if( strcmp( p_estado, "OFF" ) == 0 )
          Bomba_estado = 0;
      else{//ERROR estado invalido
          Send_To_PC( "Get And Process ERROR 5\r\n" );
      }
  }
  else if( strcmp( p_label, BUZZER_LABEL ) == 0 ){
      if( strcmp( p_estado, "ON" ) == 0 )
          Buzzer_estado = 1;
      else if( strcmp( p_estado, "OFF" ) == 0 )
          Buzzer_estado = 0;
      else{//ERROR estado invalido
          Send_To_PC( "Get And Process ERROR 6\r\n" );
      }
  }
  else if( strcmp( p_label, CONTROL_LABEL ) == 0 ){
      if( strcmp( p_estado, "Autonomo" ) == 0 )
```

```
            Control_estado = 1;
        else if( strcmp( p_estado, "Manual" ) == 0 )
            Control_estado = 0;
        else{//ERROR estado invalido
            Send_To_PC( "Get And Process ERROR 7\r\n" );
        }

    }
    else{//ERROR label incorrecto
      Send_To_PC( "Get And Process ERROR 8\r\n" );
    }
    return 1;
}
```

Bibliotecas Implementadas

CEREBOT32MX4

Fichero CEREBOT32MX4.h

```
/*
 * File          : CEREBOT32MX4.h
 * Project       : CEREBOT32MX4 Adapter
 * Revision History:
            2017/6/26:
                  - Added Macros for LEDs-onBoard and SERVO-Pins
control
            2017/6/2:
                  - Added constants
            2017/5/20:
                  - initial release

 * Authors       : Jose Guerra Carmenate
                   Leonardo González Reyes
 * Description   :
 *                      Esta biblioteca contiene definiciones y
herramientas para
 *                      facilitar el trabajo con la placa de desarrollo
 *                      Diligent Cerebot32MX4 (con PIC32MX460F512L)
 */
#ifndef __Lib_CEREBOT32MX4
#define __Lib_CEREBOT32MX4
 //Some useful constants
const char Cerebot_INPUT = 1,
              Cerebot_OUTPUT= 0,
             Cerebot_LOW = 0,
               Cerebot_HIGH = 1;

/** PORT_JA REGISTERS MAPS **/
//PORT->JA
#define CEREBOT_PORTJA_1_bit RE0_bit
#define CEREBOT_PORTJA_2_bit RE1_bit
#define CEREBOT_PORTJA_3_bit RE2_bit
#define CEREBOT_PORTJA_4_bit RE3_bit
#define CEREBOT_PORTJA_7_bit RE4_bit
#define CEREBOT_PORTJA_8_bit RE5_bit
#define CEREBOT_PORTJA_9_bit RE6_bit
#define CEREBOT_PORTJA_10_bit RE7_bit
//TRIS->JA
```

```
#define CEREBOT_TRISJA_1_bit TRISE0_bit
#define CEREBOT_TRISJA_2_bit TRISE1_bit
#define CEREBOT_TRISJA_3_bit TRISE2_bit
#define CEREBOT TRISJA 4 bit TRISE3 bit
#define CEREBOT_TRISJA_7_bit TRISE4_bit
#define CEREBOT_TRISJA_8_bit TRISE5_bit
#define CEREBOT_TRISJA_9_bit TRISE6_bit
#define CEREBOT_TRISJA_10_bit TRISE7_bit
//LAT->JA
#define CEREBOT_LATJA_1_bit LATE0_bit
#define CEREBOT_LATJA_2_bit LATE1_bit
#define CEREBOT_LATJA_3_bit LATE2_bit
#define CEREBOT_LATJA_4_bit LATE3_bit
#define CEREBOT_LATJA_7_bit LATE4_bit
#define CEREBOT_LATJA_8_bit LATE5_bit
#define CEREBOT_LATJA_9_bit LATE6_bit
#define CEREBOT_LATJA_10_bit LATE7_bit
//ODC->JA
#define CEREBOT_ODCJA_1_bit ODCE0_bit
#define CEREBOT_ODCJA_2_bit ODCE1_bit
#define CEREBOT_ODCJA_3_bit ODCE2_bit
#define CEREBOT_ODCJA_4_bit ODCE3_bit
#define CEREBOT_ODCJA_7_bit ODCE4_bit
#define CEREBOT_ODCJA_8_bit ODCE5_bit
#define CEREBOT_ODCJA_9_bit ODCE6_bit
#define CEREBOT_ODCJA_10_bit ODCE7_bit


/** PORT_JB REGISTERS MAPS **/
//PORT->JB
#define CEREBOT_PORTJB_1_bit RG9_bit
#define CEREBOT_PORTJB_2_bit RG8_bit
#define CEREBOT_PORTJB_3_bit RG7_bit
#define CEREBOT_PORTJB_4_bit RG6_bit
#define CEREBOT_PORTJB_7_bit RB15_bit
#define CEREBOT_PORTJB_8_bit RD5_bit
#define CEREBOT_PORTJB_9_bit RD4_bit
#define CEREBOT_PORTJB_10_bit RB14_bit
//TRIS->JB
#define CEREBOT_TRISJB_1_bit TRISG9_bit
#define CEREBOT_TRISJB_2_bit TRISG8_bit
#define CEREBOT_TRISJB_3_bit TRISG7_bit
#define CEREBOT_TRISJB_4_bit TRISG6_bit
#define CEREBOT_TRISJB_7_bit TRISB15_bit  //AN15
#define CEREBOT_TRISJB_8_bit TRISD5_bit
#define CEREBOT_TRISJB_9_bit TRISD4_bit
#define CEREBOT_TRISJB_10_bit TRISB14_bit//AN14
//LAT->JB
#define CEREBOT_LATJB_1_bit LATG9_bit
#define CEREBOT_LATJB_2_bit LATG8_bit
#define CEREBOT_LATJB_3_bit LATG7_bit
#define CEREBOT_LATJB_4_bit LATG6_bit
#define CEREBOT_LATJB_7_bit LATB15_bit
#define CEREBOT_LATJB_8_bit LATD5_bit
#define CEREBOT_LATJB_9_bit LATD4_bit
#define CEREBOT_LATJB_10_bit LATB14_bit
//ODC->JB
#define CEREBOT_ODCJB_1_bit ODCG9_bit
#define CEREBOT_ODCJB_2_bit ODCG8_bit
#define CEREBOT_ODCJB_3_bit ODCG7_bit
#define CEREBOT_ODCJB_4_bit ODCG6_bit
```

```
#define CEREBOT_ODCJB_7_bit ODCB15_bit
#define CEREBOT_ODCJB_8_bit ODCD5_bit
#define CEREBOT_ODCJB_9_bit ODCD4_bit
#define CEREBOT_ODCJB_10_bit ODCB14_bit

/** PORT_JC REGISTERS MAPS **/
/**
 * NOTE:
  JC-01 --> Compartido con servo S1
  JC-02 --> Compartido con servo S2
  JC-03 --> Compartido con servo S3
  JC-04 --> Compartido con servo S4
  JC-07 --> Compartido con servo S5
  JC-08 --> Compartido con servo S6
  JC-09 --> Compartido con servo S7
  JC-10 --> Compartido con servo S8
  */
//PORT->JC
#define CEREBOT_PORTJC_1_bit RG12_bit
#define CEREBOT_PORTJC_2_bit RG13_bit
#define CEREBOT_PORTJC_3_bit RG14_bit
#define CEREBOT_PORTJC_4_bit RG15_bit
#define CEREBOT_PORTJC_7_bit RG0_bit
#define CEREBOT_PORTJC_8_bit RG1_bit
#define CEREBOT_PORTJC_9_bit RF0_bit
#define CEREBOT_PORTJC_10_bit RF1_bit
//TRIS->JC
#define CEREBOT_TRISJC_1_bit TRISG12_bit
#define CEREBOT_TRISJC_2_bit TRISG13_bit
#define CEREBOT_TRISJC_3_bit TRISG14_bit
#define CEREBOT_TRISJC_4_bit TRISG15_bit
#define CEREBOT_TRISJC_7_bit TRISG0_bit
#define CEREBOT_TRISJC_8_bit TRISG1_bit
#define CEREBOT_TRISJC_9_bit TRISF0_bit
#define CEREBOT_TRISJC_10_bit TRISF1_bit
//LAT->JC
#define CEREBOT_LATJC_1_bit LATG12_bit
#define CEREBOT_LATJC_2_bit LATG13_bit
#define CEREBOT_LATJC_3_bit LATG14_bit
#define CEREBOT_LATJC_4_bit LATG15_bit
#define CEREBOT_LATJC_7_bit LATG0_bit
#define CEREBOT_LATJC_8_bit LATG1_bit
#define CEREBOT_LATJC_9_bit LATF0_bit
#define CEREBOT_LATJC_10_bit LATF1_bit
//ODC->JC
#define CEREBOT_ODCJC_1_bit ODCG12_bit
#define CEREBOT_ODCJC_2_bit ODCG13_bit
#define CEREBOT_ODCJC_3_bit ODCG14_bit
#define CEREBOT_ODCJC_4_bit ODCG15_bit
#define CEREBOT_ODCJC_7_bit ODCG0_bit
#define CEREBOT_ODCJC_8_bit ODCG1_bit
#define CEREBOT_ODCJC_9_bit ODCF0_bit
#define CEREBOT_ODCJC_10_bit ODCF1_bit

/** PORT_JD REGISTERS MAPS **/
//PORT->JD
#define CEREBOT_PORTJD_1_bit RD7_bit
#define CEREBOT_PORTJD_2_bit RD1_bit
#define CEREBOT_PORTJD_3_bit RD9_bit     //Shared with SPI Port1
Connector,J1
#define CEREBOT_PORTJD_4_bit RC1_bit
```

```
#define CEREBOT_PORTJD_7_bit RD6_bit
#define CEREBOT_PORTJD_8_bit RD2_bit
#define CEREBOT_PORTJD_9_bit RD10_bit     //Shared with SPI Port1
Connector,J1
#define CEREBOT_PORTJD_10_bit RC2_bit
//TRIS->JD
#define CEREBOT_TRISJD_1_bit TRISD7_bit
#define CEREBOT_TRISJD_2_bit TRISD1_bit
#define CEREBOT_TRISJD_3_bit TRISD9_bit
#define CEREBOT_TRISJD_4_bit TRISC1_bit
#define CEREBOT_TRISJD_7_bit TRISD6_bit
#define CEREBOT_TRISJD_8_bit TRISD2_bit
#define CEREBOT_TRISJD_9_bit TRISD10_bit
#define CEREBOT_TRISJD_10_bit TRISC2_bit
//LAT->JD
#define CEREBOT_LATJD_1_bit LATD7_bit
#define CEREBOT_LATJD_2_bit LATD1_bit
#define CEREBOT_LATJD_3_bit LATD9_bit
#define CEREBOT_LATJD_4_bit LATC1_bit
#define CEREBOT_LATJD_7_bit LATD6_bit
#define CEREBOT_LATJD_8_bit LATD2_bit
#define CEREBOT_LATJD_9_bit LATD10_bit
#define CEREBOT_LATJD_10_bit LATC2_bit
//ODC->JD
#define CEREBOT_ODCJD_1_bit ODCD7_bit
#define CEREBOT_ODCJD_2_bit ODCD1_bit
#define CEREBOT_ODCJD_3_bit ODCD9_bit
#define CEREBOT_ODCJD_4_bit ODCC1_bit
#define CEREBOT_ODCJD_7_bit ODCD6_bit
#define CEREBOT_ODCJD_8_bit ODCD2_bit
#define CEREBOT_ODCJD_9_bit ODCD10_bit
#define CEREBOT_ODCJD_10_bit ODCC2_bit

/** PORT_JE REGISTERS MAPS **/
//PORT->JE
#define CEREBOT_PORTJE_1_bit RD14_bit
#define CEREBOT_PORTJE_2_bit RD15_bit
#define CEREBOT_PORTJE_3_bit RF2_bit
#define CEREBOT_PORTJE_4_bit RF8_bit
#define CEREBOT_PORTJE_7_bit RD13_bit
#define CEREBOT_PORTJE_8_bit RD3_bit
#define CEREBOT_PORTJE_9_bit RD11_bit
#define CEREBOT_PORTJE_10_bit RC3_bit
//TRIS->JE
#define CEREBOT_TRISJE_1_bit TRISD14_bit
#define CEREBOT_TRISJE_2_bit TRISD15_bit
#define CEREBOT_TRISJE_3_bit TRISF2_bit
#define CEREBOT_TRISJE_4_bit TRISF8_bit
#define CEREBOT_TRISJE_7_bit TRISD13_bit
#define CEREBOT_TRISJE_8_bit TRISD3_bit
#define CEREBOT_TRISJE_9_bit TRISD11_bit
#define CEREBOT_TRISJE_10_bit TRISC3_bit
//LAT->JE
#define CEREBOT_LATJE_1_bit LATD14_bit
#define CEREBOT_LATJE_2_bit LATD15_bit
#define CEREBOT_LATJE_3_bit LATF2_bit
#define CEREBOT_LATJE_4_bit LATF8_bit
#define CEREBOT_LATJE_7_bit LATD13_bit
#define CEREBOT_LATJE_8_bit LATD3_bit
#define CEREBOT_LATJE_9_bit LATD11_bit
#define CEREBOT_LATJE_10_bit LATC3_bit
```

```
//ODC->JE
#define CEREBOT_ODCJE_1_bit ODCD14_bit
#define CEREBOT_ODCJE_2_bit ODCD15_bit
#define CEREBOT ODCJE 3 bit ODCF2 bit
#define CEREBOT_ODCJE_4_bit ODCF8_bit
#define CEREBOT_ODCJE_7_bit ODCD13_bit
#define CEREBOT_ODCJE_8_bit ODCD3_bit
#define CEREBOT_ODCJE_9_bit ODCD11_bit
#define CEREBOT_ODCJE_10_bit ODCC3_bit

/** PORT_JF REGISTERS MAPS **/
//PORT->JF
#define CEREBOT_PORTJF_1_bit RA2_bit     //Shared with I2C daisy chain
#2,J6
#define CEREBOT_PORTJF_2_bit RA3_bit     //Shared with I2C daisy chain
#2,J6
#define CEREBOT_PORTJF_3_bit RA6_bit     //Shared with BTN1
#define CEREBOT_PORTJF_4_bit RA7_bit     //Shared with BTN2
//TRIS->JF
#define CEREBOT_TRISJF_1_bit TRISA2_bit
#define CEREBOT_TRISJF_2_bit TRISA3_bit
#define CEREBOT_TRISJF_3_bit TRISA6_bit
#define CEREBOT_TRISJF_4_bit TRISA7_bit
//LAT->JF
#define CEREBOT_LATJF_1_bit LATA2_bit
#define CEREBOT_LATJF_2_bit LATA3_bit
#define CEREBOT_LATJF_3_bit LATA6_bit
#define CEREBOT_LATJF_4_bit LATA7_bit
//ODC->JF
#define CEREBOT_ODCJF_1_bit ODCA2_bit
#define CEREBOT_ODCJF_2_bit ODCA3_bit
#define CEREBOT_ODCJF_3_bit ODCA6_bit
#define CEREBOT_ODCJF_4_bit ODCA7_bit

/** PORT_JH REGISTERS MAPS **/
//PORT->JH
#define CEREBOT_PORTJH_1_bit RF12_bit
#define CEREBOT_PORTJH_2_bit RF13_bit
#define CEREBOT_PORTJH_3_bit RF4_bit
#define CEREBOT_PORTJH_4_bit RF5_bit
#define CEREBOT_PORTJH_7_bit RE8_bit
#define CEREBOT_PORTJH_8_bit RD0_bit     //Shared with SPI Port1
Connector,J1
#define CEREBOT_PORTJH_9_bit RD8_bit
#define CEREBOT_PORTJH_10_bit RE9_bit    //Shared with USB OC_SENSE
via JP5
//TRIS->JH
#define CEREBOT_TRISJH_1_bit TRISF12_bit
#define CEREBOT_TRISJH_2_bit TRISF13_bit
#define CEREBOT_TRISJH_3_bit TRISF4_bit
#define CEREBOT_TRISJH_4_bit TRISF5_bit
#define CEREBOT_TRISJH_7_bit TRISE8_bit
#define CEREBOT_TRISJH_8_bit TRISD0_bit
#define CEREBOT_TRISJH_9_bit TRISD8_bit
#define CEREBOT_TRISJH_10_bit TRISE9_bit
//LAT->JH
#define CEREBOT_LATJH_1_bit LATF12_bit
#define CEREBOT_LATJH_2_bit LATF13_bit
#define CEREBOT_LATJH_3_bit LATF4_bit
#define CEREBOT_LATJH_4_bit LATF5_bit
#define CEREBOT_LATJH_7_bit LATE8_bit
```

```
#define CEREBOT_LATJH_8_bit LATD0_bit
#define CEREBOT_LATJH_9_bit LATD8_bit
#define CEREBOT_LATJH_10_bit LATE9_bit
//ODC->JH
#define CEREBOT_ODCJH_1_bit ODCF12_bit
#define CEREBOT_ODCJH_2_bit ODCF13_bit
#define CEREBOT_ODCJH_3_bit ODCF4_bit
#define CEREBOT_ODCJH_4_bit ODCF5_bit
#define CEREBOT_ODCJH_7_bit ODCE8_bit
#define CEREBOT_ODCJH_8_bit ODCD0_bit
#define CEREBOT_ODCJH_9_bit ODCD8_bit
#define CEREBOT_ODCJH_10_bit ODCE9_bit

/** PORT_JJ REGISTERS MAPS **/
//PORT->JJ
#define CEREBOT_PORTJJ_1_bit RB0_bit      //AN0
#define CEREBOT_PORTJJ_2_bit RB1_bit      //AN1
#define CEREBOT_PORTJJ_3_bit RB2_bit      //AN2
#define CEREBOT_PORTJJ_4_bit RB3_bit      //AN3
#define CEREBOT_PORTJJ_7_bit RB4_bit      //AN4
#define CEREBOT_PORTJJ_8_bit RB5_bit      //AN5//Selected by J16
#define CEREBOT_PORTJJ_9_bit RB8_bit      //AN8
#define CEREBOT_PORTJJ_10_bit RB9_bit     //AN9
//TRIS->JJ
#define CEREBOT_TRISJJ_1_bit TRISB0_bit
#define CEREBOT_TRISJJ_2_bit TRISB1_bit
#define CEREBOT_TRISJJ_3_bit TRISB2_bit
#define CEREBOT_TRISJJ_4_bit TRISB3_bit
#define CEREBOT_TRISJJ_7_bit TRISB4_bit
#define CEREBOT_TRISJJ_8_bit TRISB5_bit
#define CEREBOT_TRISJJ_9_bit TRISB8_bit
#define CEREBOT_TRISJJ_10_bit TRISB9_bit
//LAT->JJ
#define CEREBOT_LATJJ_1_bit LATB0_bit
#define CEREBOT_LATJJ_2_bit LATB1_bit
#define CEREBOT_LATJJ_3_bit LATB2_bit
#define CEREBOT_LATJJ_4_bit LATB3_bit
#define CEREBOT_LATJJ_7_bit LATB4_bit
#define CEREBOT_LATJJ_8_bit LATB5_bit
#define CEREBOT_LATJJ_9_bit LATB8_bit
#define CEREBOT_LATJJ_10_bit LATB9_bit
//ODC->JJ
#define CEREBOT_ODCJJ_1_bit ODCB0_bit
#define CEREBOT_ODCJJ_2_bit ODCB1_bit
#define CEREBOT_ODCJJ_3_bit ODCB2_bit
#define CEREBOT_ODCJJ_4_bit ODCB3_bit
#define CEREBOT_ODCJJ_7_bit ODCB4_bit
#define CEREBOT_ODCJJ_8_bit ODCB5_bit
#define CEREBOT_ODCJJ_9_bit ODCB8_bit
#define CEREBOT_ODCJJ_10_bit ODCB9_bit

/** PORT_JK REGISTERS MAPS **/
//PORT->JK
#define CEREBOT_PORTJK_1_bit RB10_bit     //AN10//Shared with LED1
#define CEREBOT_PORTJK_2_bit RB11_bit     //AN11//Shared with LED2
#define CEREBOT_PORTJK_3_bit RB12_bit     //AN12//Shared with LED3
#define CEREBOT_PORTJK_4_bit RB13_bit     //AN13//Shared with LED4
#define CEREBOT_PORTJK_7_bit RA9_bit
#define CEREBOT_PORTJK_8_bit RA10_bit
#define CEREBOT_PORTJK_9_bit RD12_bit
#define CEREBOT_PORTJK_10_bit RC4_bit
```

```
//TRIS->JK
#define CEREBOT_TRISJK_1_bit TRISB10_bit
#define CEREBOT_TRISJK_2_bit TRISB11_bit
#define CEREBOT TRISJK 3 bit TRISB12 bit
#define CEREBOT_TRISJK_4_bit TRISB13_bit
#define CEREBOT_TRISJK_7_bit TRISA9_bit
#define CEREBOT_TRISJK_8_bit TRISA10_bit
#define CEREBOT_TRISJK_9_bit TRISD12_bit
#define CEREBOT_TRISJK_10_bit TRISC4_bit
//LAT->JK
#define CEREBOT_LATJK_1_bit LATB10_bit
#define CEREBOT_LATJK_2_bit LATB11_bit
#define CEREBOT_LATJK_3_bit LATB12_bit
#define CEREBOT_LATJK_4_bit LATB13_bit
#define CEREBOT_LATJK_7_bit LATA9_bit
#define CEREBOT_LATJK_8_bit LATA10_bit
#define CEREBOT_LATJK_9_bit LATD12_bit
#define CEREBOT_LATJK_10_bit LATC4_bit
//ODC->JK
#define CEREBOT_ODCJK_1_bit ODCB10_bit
#define CEREBOT_ODCJK_2_bit ODCB11_bit
#define CEREBOT_ODCJK_3_bit ODCB12_bit
#define CEREBOT_ODCJK_4_bit ODCB13_bit
#define CEREBOT_ODCJK_7_bit ODCA9_bit
#define CEREBOT_ODCJK_8_bit ODCA10_bit
#define CEREBOT_ODCJK_9_bit ODCD12_bit
#define CEREBOT_ODCJK_10_bit ODCC4_bit


//////////////HARDWARE CONSTANTS\\\\\\\\\\\\\\\\
//---------->BUTTONS<---------------\\
//Button 1
#define CEREBOT_BTN1_TRIS CEREBOT_TRISJF_3_bit
#define CEREBOT_BTN1_PORT CEREBOT_PORTJF_3_bit
#define CEREBOT_BTN1_LAT  CEREBOT_LATJF_3_bit
#define CEREBOT_BTN1_ODC  CEREBOT_ODCJF_3_bit
//Button 2
#define CEREBOT_BTN2_TRIS CEREBOT_TRISJF_4_bit
#define CEREBOT_BTN2_PORT CEREBOT_PORTJF_4_bit
#define CEREBOT_BTN2_LAT  CEREBOT_LATJF_4_bit
#define CEREBOT_BTN2_ODC  CEREBOT_ODCJF_4_bit
//----------->LEDS<---------\\
//LED 1
#define CEREBOT_LED1_TRIS CEREBOT_TRISJK_1_bit
#define CEREBOT_LED1_PORT CEREBOT_PORTJK_1_bit
#define CEREBOT_LED1_LAT  CEREBOT_LATJK_1_bit
#define CEREBOT_LED1_ODC  CEREBOT_ODCJK_1_bit
//LED 2
#define CEREBOT_LED2_TRIS CEREBOT_TRISJK_2_bit
#define CEREBOT_LED2_PORT CEREBOT_PORTJK_2_bit
#define CEREBOT_LED2_LAT  CEREBOT_LATJK_2_bit
#define CEREBOT_LED2_ODC  CEREBOT_ODCJK_2_bit
//LED 3
#define CEREBOT_LED3_TRIS CEREBOT_TRISJK_3_bit
#define CEREBOT_LED3_PORT CEREBOT_PORTJK_3_bit
#define CEREBOT_LED3_LAT  CEREBOT_LATJK_3_bit
#define CEREBOT_LED3_ODC  CEREBOT_ODCJK_3_bit
//LED 4
#define CEREBOT_LED4_TRIS CEREBOT_TRISJK_4_bit
#define CEREBOT_LED4_PORT CEREBOT_PORTJK_4_bit
#define CEREBOT_LED4_LAT  CEREBOT_LATJK_4_bit
```

```
#define CEREBOT_LED4_ODC  CEREBOT_ODCJK_4_bit

//----------------->SERVOS<------------------\\
//SERVO 1
#define CEREBOT_SERVO1_TRIS CEREBOT_TRISJC_1_bit
#define CEREBOT_SERVO1_PORT CEREBOT_PORTJC_1_bit
#define CEREBOT_SERVO1_LAT  CEREBOT_LATJC_1_bit
#define CEREBOT_SERVO1_ODC  CEREBOT_ODCJC_1_bit
//SERVO 2
#define CEREBOT_SERVO2_TRIS CEREBOT_TRISJC_2_bit
#define CEREBOT_SERVO2_PORT CEREBOT_PORTJC_2_bit
#define CEREBOT_SERVO2_LAT  CEREBOT_LATJC_2_bit
#define CEREBOT_SERVO2_ODC  CEREBOT_ODCJC_2_bit
//SERVO 3
#define CEREBOT_SERVO3_TRIS CEREBOT_TRISJC_3_bit
#define CEREBOT_SERVO3_PORT CEREBOT_PORTJC_3_bit
#define CEREBOT_SERVO3_LAT  CEREBOT_LATJC_3_bit
#define CEREBOT_SERVO3_ODC  CEREBOT_ODCJC_3_bit
//SERVO 4
#define CEREBOT_SERVO4_TRIS CEREBOT_TRISJC_4_bit
#define CEREBOT_SERVO4_PORT CEREBOT_PORTJC_4_bit
#define CEREBOT_SERVO4_LAT  CEREBOT_LATJC_4_bit
#define CEREBOT_SERVO4_ODC  CEREBOT_ODCJC_4_bit
//SERVO 5
#define CEREBOT_SERVO5_TRIS CEREBOT_TRISJC_7_bit
#define CEREBOT_SERVO5_PORT CEREBOT_PORTJC_7_bit
#define CEREBOT_SERVO5_LAT  CEREBOT_LATJC_7_bit
#define CEREBOT_SERVO5_ODC  CEREBOT_ODCJC_7_bit
//SERVO 6
#define CEREBOT_SERVO6_TRIS CEREBOT_TRISJC_8_bit
#define CEREBOT_SERVO6_PORT CEREBOT_PORTJC_8_bit
#define CEREBOT_SERVO6_LAT  CEREBOT_LATJC_8_bit
#define CEREBOT_SERVO6_ODC  CEREBOT_ODCJC_8_bit
//SERVO 7
#define CEREBOT_SERVO7_TRIS CEREBOT_TRISJC_9_bit
#define CEREBOT_SERVO7_PORT CEREBOT_PORTJC_9_bit
#define CEREBOT_SERVO7_LAT  CEREBOT_LATJC_9_bit
#define CEREBOT_SERVO7_ODC  CEREBOT_ODCJC_9_bit
//SERVO 8
#define CEREBOT_SERVO8_TRIS CEREBOT_TRISJC_10_bit
#define CEREBOT_SERVO8_PORT CEREBOT_PORTJC_10_bit
#define CEREBOT_SERVO8_LAT  CEREBOT_LATJC_10_bit
#define CEREBOT_SERVO8_ODC  CEREBOT_ODCJC_10_bit

#endif
```

CyclicBuffer

Fichero *CyclicBuffer.h*

```
/*
 * File         : CyclicBuffer.c
 * Project      : Cyclic Buffer Library
 * Revision History:
 *                 2017/06/15:
 *                         - initial release
 *
 * Author       : José Guerra Carmenate
 *
 * Description  : This library provides you a FIFO-Buffer struct and a
comfortable set of routines
 *        to work with this Buffers
 * Routines:
 *                        - Buffer_Init
 *                        - Buffer_Data_Ready
 *                        - Buffer_GetData
 *                        - Buffer_GetAllData
 *                        - Buffer_PushData
 *                        - __Buffer_Next_index
 */
#ifndef __LibCyclicBuffer
#define __LibCyclicBuffer
 /*********************
  *   Buffer Struct   *
  *********************/
typedef struct CyclicBuffer{
        unsigned int max_size, // Maximun Capacity
                front,           // index of data on the top of the
buffer
                back,           // index of the last data in the
buffer
                data_count;// quantity of data in the buffer
        unsigned char *buff;
} Buffer;

 /*************************************************************
  * Prototype:
  *                void Buffer_Init( Buffer *buffer, unsigned char
*array, int max_size )
  * Description:
  *                                    Initilaize the Buffer 'buffer'
  * Parameters:
  *     - buffer: Buffer to be initialized
  *                - array : Global Array used for the buffer
  * Returns:
  *         nothing
  * Example:
  *                // Initialize the 'Buff1' Buffer
  *                Buffer_Init( &Buff1, *buff_arr, 256 );
  *************************************************************/
void Buffer_Init( Buffer *buffer, unsigned char *array, unsigned int
max_size );

 /*************************************************************
  * Prototype:
  *                int Buffer_Data_Ready( Buffer *buff )
  * Description:
```

```
  *                                   get the quantity of data on
'buff'.
  * Parameters:
  *     - buff: buffer
  * Returns:
  *         the quantity of data on 'buff'.
  * Example:
  *                 // Get the quantity of data ready to be read from
Buff1
  *                 int sz = Buffer_Data_Ready( &Buff1 );
  ************************************************************/
int Buffer_Data_Ready( Buffer *buffer );

 /************************************************************
  * Prototype:
  *                 unsigned char Buffer_GetData( Buffer *buff )
  * Description:
  *                                   get the data on the top of
'buff' and erase it.
  * Parameters:
  *     - buff: buffer
  * Returns:
  *         the data on the top of 'buff'
  * Example:
  *                 // get the data on the top of Buff1
  *                 unsigned char sz = Buffer_GetData( &Buff1 );
  ************************************************************/
unsigned char Buffer_GetData( Buffer *buffer );

 /************************************************************
  * Prototype:
  *                 int Buffer_GetAllData( Buffer *buff, unsigned char
*data )
  * Description:
  *                                   put all the data on the 'buff'
in the 'data' array and erase it.
  * Parameters:
  *     - buff: buffer
  *         - data:
  * Returns:
  *         the quantity of bytes stored on 'data'.
  * Example:
  *                 // get all the data on Buff1 and put it in 'data'
array
  *                 int sz = Buffer_GetAllData( &Buff1, data );
  ************************************************************/
int Buffer_GetAllData( Buffer *buffer, unsigned char *dat );

 /************************************************************
  * Prototype:
  *                 char Buffer_PushData( Buffer *buff, unsigned char
dat )
  * Description:
  *                                   put dat on the 'buff'.
  * Parameters:
  *     - buff: buffer
  *         - dat : byte to be pushed into the buffer
  * Returns:
  *         1 - byte pushed successfully
  *                         0 - Overflow error
  * Example:
```

```
  *                //put the data 0x85 on the buffer Buff1
  *                int ok = Buffer_PushData( &Buff1, 0x85 );
  ************************************************************/
int Buffer_PushData( Buffer *buffer, unsigned char dat );

 /************************************************************
  * Prototype:
  *                void __Buffer_PopFront( Buffer *buff )
  * Description:
  *                                    "Erace" the next element on the
Buffer
  * Parameters:
  *     - buff: buffer
  * Returns:
  *         - Nothing
  * NOTA:
  *                - This routine is only for internal use
  * Example:
  *                                __Buffer_PopFront( &Buff1 );
  ************************************************************/
void __Buffer_PopFront( Buffer *buffer );


#endif
```

Fichero CyclicBuffer.c:

```
/*
 * File         : CyclicBuffer.c
 * Project      : Cyclic Buffer Library
 * Revision History:
 *                  2017/06/15:
 *                          - initial release
 *
 * Author       : José Guerra Carmenate
 *
 * Description  : This library provides you a FIFO-Buffer struct and a
comfortable set of routines
 *        to work with this Buffers
 * Routines:
 *                         - Buffer_Init
 *                         - Buffer_Data_Ready
 *                         - Buffer_GetData
 *                         - Buffer_GetAllData
 *                         - Buffer_PushData
 *                         - __Buffer_Next_index
 */
#include "CyclicBuffer.h"
void Buffer_Init( Buffer *buffer, unsigned char *array, unsigned int
max_size ){
        (*buffer).buff = array;
        (*buffer).max_size = max_size;
        (*buffer).front = (*buffer).back = (*buffer).data_count = 0;
}

int Buffer_Data_Ready( Buffer *buffer ){
        return (*buffer).data_count;
}

unsigned char Buffer_GetData( Buffer *buffer ){
        unsigned char res;
```

```
        if( (*buffer).data_count == 0 ) return 0;
        res = (*buffer).buff[ (*Buffer).front ];
        __Buffer_PopFront( buffer );
        return res;
}

int Buffer_GetAllData( Buffer *buffer, unsigned char *out ){
        int cnt = 0;
        while( Buffer_Data_Ready( buffer ) ){
                out[cnt++] = (*buffer).buff[ (*buffer).front ];
                __Buffer_PopFront( buffer );
        }
        out[cnt] = '\0';
        (*buffer).data_count = 0;
        return cnt;
}

int Buffer_PushData( Buffer *buffer, unsigned char dat ){
        if( (*buffer).data_count == (*buffer).max_size+1 )
                return 0;
        (*buffer).buff[ (*buffer).back ] = dat;
        (*buffer).back++;
        (*buffer).data_count++;
        if( (*buffer).back == (*buffer).max_size )
                (*buffer).back = 0;
        return ( (*buffer).front != (*buffer).back);
}

void __Buffer_PopFront( Buffer *buffer ){
        (*buffer).front++;
        (*buffer).data_count--;
        if( (*buffer).front == (*buffer).max_size )
                (*buffer).front = 0;
}
```