# SISTEMA OPERATIVO DE TIEMPO REAL PARA CONTROL DE PROPÓSITO GENERAL, BASADO EN UN MOTHER BOARD DE PC CON MICROPROCESADOR INTEL PENTIUM Y ARRANQUE DESDE USB

Ing. Carlos Lizardo Corzo Ruíz


**Escuela de Ingenierías**

**Eléctrica, Electrónica**

**y de Telecomunicaciones**

Universidad Industrial de Santander

Facultad de Ingenierías Físico Mecánicas

Escuela de Ingeniería Eléctrica, Electrónica y de Telecomunicaciones

Bucaramanga

2014

# SISTEMA OPERATIVO DE TIEMPO REAL PARA CONTROL DE PROPÓSITO GENERAL, BASADO EN UN MOTHER BOARD DE PC CON MICROPROCESADOR INTEL PENTIUM Y ARRANQUE DESDE USB

Ing. Carlos Lizardo Corzo Ruíz

Trabajo de investigación presentado como requerimiento parcial para optar al título de:

Magíster en Ingeniería Electrónica

Director:

MsC. Jorge Hernando Ramón Suárez


**Escuela de Ingenierías**

**Eléctrica, Electrónica**

**y de Telecomunicaciones**

Universidad Industrial de Santander
Facultad de Ingenierías Físico Mecánicas
Escuela de Ingeniería Eléctrica, Electrónica y de Telecomunicaciones

Maestría en Ingeniería electrónica

Bucaramanga

2014

## AGRADECIMIENTOS

Mi agradecimiento especial al director del trabajo final de grado, Jorge H. Ramón por su colaboración y acompañamiento en este proceso.

A los doctores Rodolfo Villamizar y Daniel Sierra por su apoyo y motivación para finalizar este trabajo.

## DEDICATORIA

A mí amada esposa RuthMira y a mi amado hijo Carlos Mario, por su paciencia, sacrificio y comprensión, durante este largo periodo de formación.

También a mi madre quien al lado del todopoderoso siempre me ha acompañado y a mi padre Arnulfo quien siempre estuvo pendiente de este logro.

**TABLA DE CONTENIDO**

## LISTA DE FIGURAS

## LISTA DE TABLAS

## LISTA DE ANEXOS

## RESUMEN

**TÍTULO:** SISTEMA OPERATIVO DE TIEMPO REAL PARA CONTROL DE PROPÓSITO GENERAL, BASADO EN UN MOTHER BOARD DE PC CON MICROPROCESADOR INTEL PENTIUM Y ARRANQUE DESDE USB[1].

**AUTOR:** CARLOS LIZARDO CORZO RUÍZ[2].

PALABRAS CLAVE: Sistema Operativo, Tiempo Real, RTOS, PID, RS485, Control.

Se presenta la implementación del control remoto mediante interfaz RS-485 y sistema operativo de tiempo real uC/OS-II sobre una plataforma PC de bajo costo para un robot de 6 grados de libertad. Igualmente la implementación de una interfaz de usuario mediante interfaz RS-232 usando herramientas de software libre (*open source)* tanto para la gestión de datos como para el manejo gráfico de los mismos.
Entre los logros a destacar puede mencionarse la disponibilidad de todos los componentes hardware que lo integran en el mercado local; lo que permite su implementación completa con bajo costo. Desde el punto de vista de los programas utilizados, el costo de los mismos es nulo puesto que son del tipo Software libre.
De igual forma el trabajo realizado permitió establecer la metodología para el desarrollo de aplicaciones de sistemas de control usando hardware de propósito general y sistema operativo de tiempo real, con el que se consiguieron tiempos de respuesta por tarea del orden de 62.5 uS, los cuales permanecieron constantes a pesar de la sobrecarga de tareas por incremento del volumen de datos. Del mismo modo se implementaron compensadores PID operando tanto en el sistema operativo uC/OS-II (modo remoto) como en los controladores locales basados en microcontroladores PIC (modo local), los cuales fueron implementados con la posibilidad de programar todas sus constantes así como las diferentes variables implicadas en los mismos como velocidad, aceleración, retardo y tiempo de servo. También se dispuso la transmisión de datos a la terminal de usuario con el fin de monitorear en tiempo real los datos de posición y velocidad, útiles en la sintonización de los controladores.

[1] Trabajo de investigación.
[2] Facultad de Ingenierías Físico Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. Grupo CPS. Director MsC. Jorge Hernando Ramón Suárez.

## ABSTRACT

**TITLE:** REAL TIME OPERATING SYSTEM FOR GENERAL PURPOSE CONTROL BASED ON A PC MOTHER BOARD WITH INTEL PENTIUM MICROPROCESSOR AND BOOT FROM USB[3].

**AUTHOR:** LIZARDO CORZO CARLOS RUIZ[4].

**KEYWORDS:** Operating System, Real Time, RTOS, PID, RS485, Control.

Presents the implementation of RS-485 interface and real-time operating system uC/ OS-II on a low cost PC platform for robot with 6 degrees of freedom. Also the implementation of a user interface via RS-232 using free software tools (open source) for data and graphical management. The achievements can be mentioned to highlight the availability of all hardware components that comprise the local market; allowing full implementation with low cost. From the point of view of the software used, the cost of them is zero since it is the free software type.

The work allowed us to establish the methodology for the development of applications of control systems using general purpose hardware and real-time operating system. The response times were achieved by task in the order of 62.5 uS, which remained constant despite task overload by increasing the volume of data. PID compesators implemented both in uC / OS-II Operating system (remote mode) and the local controllers, PIC microcontroller-based (local mode), which were implemented with the ability to schedule all their constant and the different variables involved in them as speed, acceleration, and time delay servo. Data transmission was also available to the user terminal in order to monitor in real time the position and velocity data useful in the tuning of the controllers.


[3] Research Work.

[4] Physical Faculty of Mechanical Engineering. School of Electrical, Electronics, and Telecommunications. Group CPS. Director: MsC Jorge Hernando Ramón Suárez.

# INTRODUCCIÓN

El Computador personal es usado para labores como: Automatizar las tareas en la oficina, navegar en la WEB, gestionar redes sociales y en algunos casos para controlar y gestionar procesos automáticos en la industria. El amplio uso de esta plataforma ha disminuido el costo de ella al punto de estar al alcance de cualquier individuo. Además, el volumen de compra y la proliferación de fabricantes ha generado el desarrollo de nuevas características que proveen mayores prestaciones. Por ejemplo: gran velocidad de procesamiento, mejor resolución en gráficos, mayores anchos de banda y menores tiempos de acceso a bases de datos. En cuanto a los programas utilizados en las actividades mencionadas se involucran sistemas operativos que van desde monousuario monotarea, hasta multiusuario multitarea en los cuales el tiempo tomado por los procesos involucrados es de relevancia relativa y por tanto no determinístico.

Para las aplicaciones industriales como controlar un robot industrial, se requieren tiempos de procesamiento y respuesta muy cortos comparables con las constantes eléctricas de los motores que lo integran con el objeto de obtener resultados positivos. En esta dirección se orienta la investigación realizada que tiene como objeto principal implementar un sistema operativo de tiempo real que pueda gestionar exitosamente el posicionamiento de un robot de 6 grados de libertad.

Ya que el interés del trabajo de investigación propuesto pretende proveer una plataforma útil en el control de procesos, se presentan tópicos conducentes a la gestión en tiempo real de estos. Por lo anterior se incluye lo referente a la arquitectura del computador personal, inicialización y verificación del sistema (POST), Arranque (BOOT), asignación de memoria para el sistema operativo y aplicaciones; así como el manejo de interrupciones e interconexión de puertos de entrada y salida. De igual forma se muestran aspectos relevantes al núcleo del sistema operativo FREEDOS (usado como puente entre usuario y el sistema base de entrada y salida BIOS) que posee

características similares al MS-DOS y que es usado como punto de arranque para el núcleo de sistema operativo de tiempo real uC/OS-II sobre el cual se ejecutan las tareas de control del robot objeto. Finalmente se indican apartes de los microcontroladores utilizados como dispositivos de adquisición y actuación sobre los componentes del manipulador usado para la validación de los compensadores PID necesarios para posicionarlo; igualmente, se incluye la realización de una red de datos serial RS-232/485 con protocolo propietario gestionada mediante la técnica TOKEN PASS con dirección manejada por paridad marca.

# 1. ANTECEDENTES

La arquitectura INTEL x86 a partir de la versión MMX incluyó instrucciones exclusivas para el tratamiento de números en punto fijo y flotante en su coprocesador matemático. Posteriormente con el advenimiento de los procesadores PENTIUM y la migración hacia la arquitectura HARVARD (RISC) así como la implementación de la técnica SIMD (Single input múltiple data) para la gestión de datos, se logró una plataforma que mantuvo la compatibilidad con la gran mayoría de los programa preexistentes en el mercado y se mejoró el desempeño en aplicaciones de tratamiento digital de señales requerido en el manejo del audio y video presentes en las aplicaciones multimedia. Esta evolución implicó el desarrollo de sistemas operativos que sacaran provecho de estas nuevas características de la CPU **[3].**

Los computadores personales se basan en procesadores de propósito general (GPP) como los PENTIUM y sucesores de múltiples núcleos i3, i5 e i7; que permiten la ejecución de diferentes sistemas operativos ya sean multiusuario y/o multitareas orientados a prestar servicio a labores de humanos así como sistemas operativos orientados a dar servicio a máquinas que controlan procesos.

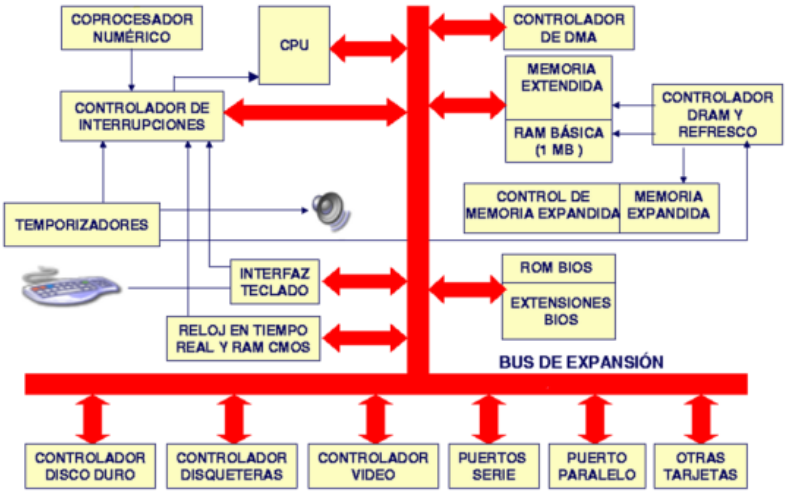
## 1.1 ARQUITECTURA BÁSICA DEL PC CON INTEL PENTIUM

El microprocesador Intel PENTIUM fue el primero en implementar un bus de datos de 64 bits, diseñado para sistemas operativos como DOS, WINDOWS, LINUX, etc. Puede direccionar hasta 4 Giga Bytes de memoria física y hasta 64 Tera Bytes de memoria virtual. Provee registros de traslación de direcciones, hardware para multitarea y mecanismos de protección para uso en sistemas operativos de este tipo. De igual forma incluye características de depuración y prueba de instrucciones, registros para la implementación de puntos de parada en la ejecución de programas y en el acceso a

datos. La compatibilidad hacia atrás con sus predecesores (8088 , 8086, 80286, 80386 y 80486) es garantizada permitiendo acceder a toda la base de programas desarrollados para estos procesadores **[10]**.

En su versión básica (Intel 8086) un PC requiere para su operación al menos de 5 unidades: Base de tiempo (Reloj), Unidad Central de proceso (CPU), memoria de programa, memoria de datos y puertos de entrada/salida interconectadas como se muestra en la figura 1.

Figura 1. Arquitectura básica de un INTEL-PC.




### 1.1.1 Base de Tiempo (Reloj)

La temporización en el PC con microprocesador 8086, está implementada alrededor del chip temporizador INTEL 8254 el cual consta de 3 contadores / temporizadores de 16 bits. Uno de estos temporizadores es utilizado para generar una interrupción con frecuencia 18,2 Hz que corresponde al TICK del sistema operativo (Tiempo de asignación del procesador a cada tarea y que se modifica en el manejo por RTOS). De

igual forma posee un temporizador usado para el refresco de la memoria RAM dinámica y un tercero para realizar temporizaciones de propósito general.

### 1.1.2 Unidad Central de Proceso (CPU)

La unidad central de proceso en un PC (Intel-8086) está conformada por una unidad de ejecución y una unidad de gestión de instrucciones. La unidad de ejecución contiene 8 registros de 32 bits de propósito general los cuales son usados para cálculo de direcciones y operaciones de datos. Del mismo modo está dotado de un *barrel shifter* usado para acelerar las operaciones de suma, rotación de bits, multiplicación y división realizando estas operaciones en un ciclo de máquina. La unidad de instrucciones decodifica los códigos de operación y almacena estos en la cola de instrucciones decodificadas para uso inmediato por parte de la unidad de ejecución.

La unidad de manejo de memoria (MMU) está conformada por una unidad de segmentación y una unidad de paginación. La segmentación permite el manejo del espacio de memoria lógico proporcionando un componente de direccionamiento extra que facilita relocalizar tanto programas como datos. El mecanismo de paginación es transparente a la segmentación y permite manejar las direcciones físicas del espacio de memoria.

Finalmente la unidad de interfaz del canal (BIU) es la encargada de leer las instrucciones y de leer o escribir datos entre el microprocesador y la memoria. Para facilitar el diseño de sistemas hardware de alto rendimiento, el 80386DX provee interfaces de buses, paralelismo de direcciones, tamaño dinámico del bus de datos y habilitación directa de Bytes para cada uno los datos en el bus **[8]**.

### 1.1.3 Memoria de Datos (RAM)

La memoria física de los procesadores 8086, 80286DX, 80386DX y 80486DX difiere en el ancho de la configuración en bits. Para el caso del 8086 es de 16 bits igual que el

ancho del procesador 80286. Para los procesadores 80386DX y 80486DX es de 32 bits. Para el caso de la programación, no hay diferencia en el ancho de la memoria ya que la memoria lógica siempre es de 8 Bits de ancho. Desde el punto de vista del programador se habla de modelo de memoria lógica que es el modelo usado por éste para programar el sistema. En la figura 2 se indica el mapa de memoria lógica de los procesadores 8086, 80286DX, 80386DX y 80486DX. Cuando estos direccionan una palabra de 16 bits en la memoria, se acceden 2 Bytes consecutivos. Si se apunta a una palabra doble de 32 bits, el acceso se realizará a 4 Bytes consecutivos.

Figura 2. Mapa de memoria lógica de los INTEL 80X86DX.

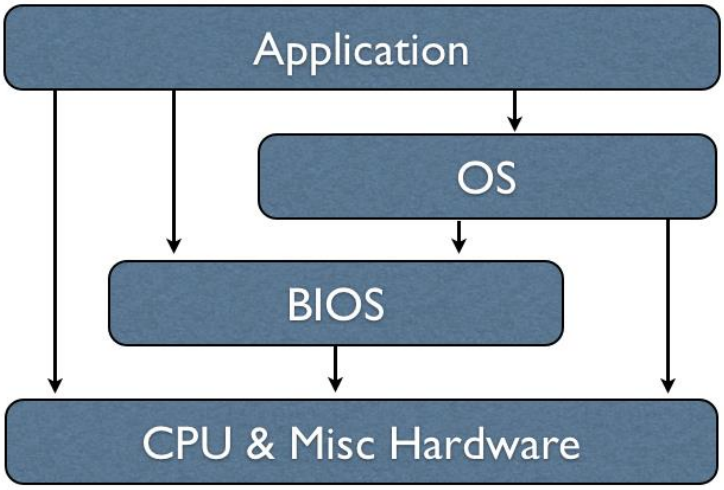
| MAPA DE MEMORIA INTEL 80386DX | | |
|---|---|---|
| INICIO | FIN | SEGMENTO |
| 000000 | 0003FF | Vectores de Interrupción |
| 000400 | 0004FF | Área del BIOS |
| 000500 | 0005FF | Área de ROM BASIC |
| 000600 | 07FFFF | RAM DOS (512K) |
| 080000 | 09FFFF | Expansión RAM (AT) |
| 0A0000 | 0C3FFF | Buffer de Video |
| 0C4000 | 0C7FFF | Expansión ROM |
| 0C8000 | 0CCFFF | Disco Duro |
| 0CD000 | 0D0000 | RAM de Usuario |
| 0D0000 | 0DFFFF | Expansión RAM (LIM) |
| 0E0000 | 0EFFFF | Expansión ROM |
| 0F0000 | 0FFFFF | ROM |
| 100000 | FFFFFF | Expansión RAM (AT) |

### 1.1.4 Memoria de Programa (BIOS)

La memoria de programa básica en un PC es la ROM-BIOS (Sistema básico de entrada salida en ROM) y se encuentra entre los recursos de hardware y el sistema operativo como se indica en la figura 3. Los programas residentes en esta sección de la memoria

de programa están organizados en servicios que son usados por el sistema operativo mediante llamados usando un mecanismo de interrupciones hardware ó software para realizar las tareas fundamentales del sistema, tales como leer y escribir bytes individuales de datos en la pantalla, o en el disco duro. Los servicios de lenguaje de programación y los servicios del sistema operativo se construyen, a menudo, a partir de estas funciones básicas y se mejoran para hacer más eficaces determinados procesos particulares **[18]**.

Figura 3. Ubicación del sistema básico de entrada – salida en la plataforma PC




### 1.1.5 Servicio a Interrupciones

Todos los servicios de la ROM-BIOS se invocan mediante interrupciones las cuales están definidas por direcciones preestablecidas en el modelo de memoria lógica del PC como se presenta en la figura 4. Las direcciones de los vectores de interrupción se ubican en las posiciones bajas de la memoria de datos y apuntan a rutinas de servicio almacenadas

en ROM. Este diseño hace posible que cualquier programa demande un servicio sin conocer la localización de memoria específica de la rutina de servicios de la ROM-BIOS. Permite también que los servicios sean trasladados, expandidos o adaptados, sin afectar a los programas que utilizan dichos servicios.

Figura 4. Interrupciones de la ROM-BIOS.

| INTERRUPCIONES SOFTWARE | | | |
|---|---|---|---|
| INICIO | FIN | INT | DESCRIPCIÓN |
| 00000 | 00003 | 0 | División por Cero |
| 00004 | 00007 | 1 | Depuración |
| 00008 | 0000B | 2 | Int no enmascarable |
| 0000C | 0000F | 3 | Punto de parada |
| 00010 | 00013 | 4 | Rebosamiento Mat |
| 00014 | 00017 | 5 | Video via BIOS |
| 00018 | 0001B | 6 | Reservado |
| 0001C | 0001F | 7 | Reservado |
| 00020 | 00023 | 8 | Temporizador |
| 00024 | 00027 | 9 | Teclado |
| 00028 | 0002B | A | Reservado |
| 0002C | 0002F | B | Comunicaciones |
| 00030 | 00033 | C | Comunicaciones |
| 00034 | 00037 | D | Impresora |
| 00038 | 0003B | E | Unidad de Disquete |
| 0003C | 0003F | F | Control de impresora |
| 00040 | 00043 | 10 | Servicios BIOS - Video |
| 00044 | 00047 | 11 | Chequeo de Equipos BIOS |
| 00048 | 0004B | 12 | Tamaño de RAM - BIOS |
| 0004C | 0004F | 13 | Servicios de disco BIOS |
| 00050 | 00053 | 14 | BIOS - RS232 |
| 00054 | 00057 | 15 | BIOS - Casete |
| 00058 | 0005B | 16 | BIOS - Teclado |
| 0005C | 0005F | 17 | BIOS - Impresora |
| 00060 | 00063 | 18 | ROM BASIC |
| 00064 | 00067 | 19 | Arranque BIOS |
| 00068 | 0006B | 1A | Tiempo BIOS |
| 0006C | 0006F | 1B | Ctr Break BIOS |
| 00070 | 00073 | 1C | Tick del Timer |
| 00074 | 00077 | 1D | Inicio de Video |
| 00078 | 0007B | 1E | Tabla de punteros disco |
| 0007C | 0007F | 1F | Tabla de punteros VIDEO |
| 00080 | 00083 | 20 | Terminación programa DOS |
| 00084 | 00087 | 21 | Llamada de función BIOS |
| 00088 | 0008B | 22 | Fin de programa DOS |
| 0008C | 0008F | 23 | Ctr Break Teclado |
| 00090 | 00093 | 24 | Error DOS |
| 00094 | 00097 | 25 | Lectura disco DOS |
| 00098 | 0009B | 26 | Escritura disco DOS |
| 0009C | 0009F | 27 | TSR DOS |
| 000A0 | 000FF | 28-3F | Reservado |

### 1.1.6 Puertos de Entrada - Salida

En los Intel-PC el intercambio de información con el exterior usando dispositivos de comunicación serial, convertidores análogos/digitales, es realizado mediante dos métodos:

#### 1.1.6.1 Entrada/Salida Aislada.

Este método transfiere datos entre el acumulador del microprocesador y los puertos de Entrada / Salida cuyas direcciones están aisladas de la memoria del sistema en un espacio exclusivo de direcciones de E/S. A este espacio de direcciones puede accederse únicamente con las instrucciones IN, INS, OUT y OUTS.

#### 1.1.6.2 Entrada/Salida Mapeada

En este método de acceso a los puertos se utiliza cualquier instrucción que transfiera datos, entre el microprocesador y la memoria. En este caso los puertos de entrada / salida se mapean en una zona de la memoria, reduciendo así este espacio en la disponibilidad de la memoria de datos. En el PC los puertos están mapeados entre las direcciones 0000H y 03FFH. El mapa correspondiente al direccionamiento de puertos de entrada salida en el PC se muestra en la figura 5.

## 1.2 MODOS DE DIRECCIONAMIENTO

La arquitectura del INTEL-PENTIUM permite la operación del sistema en 2 modos: Direccionamiento en Modo Real y en Modo Protegido. Los microprocesadores 8086 pueden operar únicamente en el modo real, mientras que los microprocesadores 80386DX, 80486DX y PENTIUM pueden operar en modo real y en modo protegido **[4]**.

Figura 5. Mapa de direcciones Entrada-Salida en un PC.

| DIRECCION E/S | DISPOSITIVO |
|---|---|
| (000-00F) A (081-09F) | DMA |
| (010-01F) A (0A0-0A1) | Control INT |
| (040-043) | Timer Sistema |
| (060) | Teclado |
| (061) | Parlante |
| (070-071) | RAM CMOS |
| (0F0-0FF) | Coprocesador |
| (130-14F) | Adaptador SCSI |
| (170-177) | Disco Duro 2 |
| (1F0-1F7) | Disco Duro 1 |
| (200-207) | Joystick |
| (220-22F) | Sonido multimedia |
| (294-297) | Bus PCI |
| (278-27F) | Puerto Paralelo LPT2 |
| (2E8-2EF) | COM4 |
| (2F8-2FF) | COM2 |
| (376-376) | IDE PCI |
| (378-37F) | Puerto Paralelo LPT1 |
| (3E8-3EF) | COM3 |
| (3F0-3F7) | Floppy |
| (3F6-3F6) | IDE primario |
| (3F8-3FF) | COM1 |
| (E000-E01F) | host USB |
| (E800-E87F) | Fast Ethernet |
| (F000-F00F) | Controlador IDE |

### 1.2.1 Direccionamiento en Modo Real

El modo de direccionamiento en modo real permite acceder sólo al primer Mbyte del espacio de memoria, aunque sea un microprocesador PENTIUM. Este es el modo de operación por defecto después de reiniciar el sistema o después de aplicar la energía. Se caracteriza por el uso de direcciones segmentadas de 20 bits, las cuales son conformadas por un registro de segmento de 16 bits (Que permite acceder a 64 KBytes de memoria) más un desplazamiento dentro del segmento establecido por un registro apuntador. Esta estrategia de direccionamiento favorece el cambio de lugar de los

programas en el sistema de memoria; al ser reubicable, un programa puede ser grabado en cualquier zona de la memoria y ejecutado sin cambio. Igual ocurre con los datos y la pila de una tarea en particular lo cual es muy deseable en un sistema de propósito general **[4]**.

### 1.2.2 Direccionamiento en Modo Protegido

El modo de direccionamiento protegido permite acceder a los programas y a los datos que se encuentran arriba del primer Mbyte de memoria. En este modo un registro de segmento contiene ahora un **selector** que selecciona un **descriptor**. Este presenta la ubicación, longitud y derechos de acceso del segmento de memoria.
Los selectores corresponden a los registros de segmento del microprocesador. Pueden direccionar uno de los 8192 descriptores posibles existentes en la tabla de descriptores, la cual se encuentra en la memoria de datos del sistema. Los descriptores contienen la ubicación, longitud y derechos de acceso de un segmento de memoria. Hay dos tablas de descriptores: Una la tabla de descriptores globales que contiene segmentos que se aplican a todos los programas. La segunda tabla corresponde a la de descriptores locales la cual se aplica a programas exclusivos.

## 1.3 INTERRUPCIONES HARDWARE

Los microprocesadores INTEL x86 disponen de los pines INTR, NMI y INTA que en los computadores personales gestionan dos controladores de interrupciones INTEL 8259A interconectados para ampliar el número a 16. Para el caso de las interrupciones software, se dispone de las instrucciones BOUND, INT0, INT3 e INT que pueden ser usadas por el programador para implementar servicios conducentes a realizar la interfaz de programas y dispositivos a modo de controladores de los mismos.

### 1.3.1 Vectores de interrupción hardware

Un vector de interrupción corresponde a una dirección segmentada de 4 Bytes dentro del esquema de memoria del PC. En esta dirección se encuentra alojado el programa de tratamiento de interrupción (ISR) correspondiente a la interrupción decodificada por el hardware o requerida por el usuario mediante programación. En el modo real, ocupan el primer Kbyte de memoria con lo cual pueden establecerse 256 vectores de interrupción diferentes los cuales se indican en la tabla 1.

Tabla 1. Vectores de Interrupción hardware en el PC (Modo Real).

| IRQ | Número de interrupción | Descripción |
|---|---|---|
| 0 | 8 | Temporizador del sistema (18.2 veces/segundo) |
| 1 | 9 | Teclado |
| 2 | 0Ah | Controlador de interrupciones programable |
| 3 | 0Bh | COM2 (puerto serial 2) |
| 4 | 0Ch | COM1 (puerto serial 1) |
| 5 | 0Dh | LPT2 (puerto paralelo 2) |
| 6 | 0Eh | Controlador de disco flexible |
| 7 | 0Fh | LPT1 (puerto paralelo 1) |
| 8 | 70h | Reloj CMOS en tiempo real |
| 9 | 71h | (Se redirige hacia INT 0Ah) |
| 10 | 72h | (Disponible) tarjeta de sonido |
| 11 | 73h | (Disponible) tarjeta SCSI |
| 12 | 74h | Ratón PS/2 |
| 13 | 75h | Coprocesador matemático |
| 14 | 76h | Controlador de disco duro |
| 15 | 77h | (Disponible) |

### 1.3.2 Interrupciones Hardware

El pin de interrupción no enmascarable (NMI) es una entrada del procesador disparada por flanco que requiere la atención del procesador cuando cambia de estado bajo a alto. Es la interrupción de mayor jerarquía y suele ser implementada para señalizar la ocurrencia de errores de paridad tanto en el traslado de información con la memoria como con los periféricos; también se utiliza para señalizar fallos de energía a fin de proveer un lapso de tiempo para salvar la información gestionada en el momento. La entrada de petición de interrupción (INTR) debe ser puesta en nivel alto y mantenerse en este nivel hasta cuando el procesador la reconozca mediante señalización en el pin de reconocimiento de interrupción (INTA). Un evento externo activa la interrupción mediante la entrada INTR y es desactivada dentro del procedimiento de servicio de la interrupción. La detección de nuevos eventos en esta entrada es deshabilitada automáticamente cuando el microprocesador reconoce la interrupción y es habilitada con la instrucción IRET al finalizar el procedimiento de tratamiento de la interrupción. Las interrupciones de hardware están direccionadas en los primeros 32 vectores, que han sido reservados por INTEL para éste propósito y para expansión futura **[9]**.

### 1.3.3 Interrupciones Software

En el modo real las instrucciones BOUND, INT0, INT3 e INT n buscan y cargan un vector de la tabla de vectores y luego llaman el procedimiento guardado en la dirección apuntada por él.

La instrucción BOUND compara el valor de un registro con límites almacenados en la memoria. Si el valor en el registro se encuentra fuera de los límites en memoria se genera la interrupción 5. La instrucción INT0 genera la interrupción 4 cuando el bit de sobre flujo del registro de banderas del microprocesador es puesto a 1.

En el caso de la instrucción INT3, esta es utilizada para señalizar puntos de ruptura en un programa (BREAK POINTS) cuando se está depurando un programa.

Por último la instrucción en la INT n, ya que n puede tomar valores entre 0 y 255 hay 256 diferentes instrucciones disponibles para el programador. Siempre que se ejecuta una instrucción INT n se realizan automáticamente las siguientes acciones:
Salvar el registro de banderas en la pila.
Desactivar los bits de bandera T (Bit de trampa usado para el modo de depuración), y el bit I (Habilitador de interrupciones).
Salvar el registro de segmento de código en la pila.
Recuperar el segmento de código del vector.
Salvar el apuntador de instrucciones (Contador de programa) en la pila.
Recuperar el nuevo valor de contador de programa correspondiente al vector.
Saltar a la dirección apuntada por el segmento de código y contador de programa del vector.
Las interrupciones por software se utilizan para llamar los procedimientos del sistema, los cuales son usados por las aplicaciones como por el sistema operativo.
La instrucción IRET se usa al finalizar procedimientos de servicio de interrupción bien sean hardware o software. Cuando se ejecuta esta instrucción se realizan automáticamente las siguientes acciones:
Recuperar desde la pila el registro apuntador de instrucciones.
Recuperar desde la pila el registro de segmento de código.
Recuperar desde la pila el registro de banderas.

### 1.3.4 Cambio de los servicios de Interrupción

Los procedimientos correspondientes a las interrupciones hardware en el PC son cargados por el sistema básico de entrada / salida (BIOS) cuando se enciende el sistema y antes de realizar las rutinas de verificación inicial del PC (POST). Los procedimientos correspondientes a las interrupciones software son cargados por el sistema operativo que se desea ejecutar en el PC por parte del usuario.

Se suelen cambiar los procedimientos de servicio de interrupción para procesarlos de manera conveniente de acuerdo con la aplicación que se esté ejecutando. De igual forma se pueden direccionar los vectores de interrupción para apuntar a procedimientos especiales requeridos por el usuario mediante aplicaciones del tipo residentes en memoria después de terminar (TSR).

## 1.4 POWER ON SELF TEST

El primer trabajo a realizar por los programas después de encender el PC corresponde a la puesta en marcha del sistema y supervisión del mismo. Entre las tareas a realizar en este proceso se encuentran: la comprobación del estado de la BIOS, la confiabilidad de la memoria de datos principal, la disponibilidad del controlador del video y la inicialización de los chips de soporte para los demás periféricos existentes. De igual forma corresponde fijar la tabla de los vectores de interrupción y establecer el equipo opcional conectado al computador y, si está conectado un controlador de disco, terminan cargando el sistema operativo situado en ese disco.

## 1.5 ARRANQUE DEL SISTEMA (BOOTSTRAP)

En el proceso de inicialización una rutina establece los valores por defecto de los vectores de interrupción. Estos valores apuntan hacia las rutinas dedicadas al tratamiento de interrupciones estándar, que se encuentran localizadas en el interior de la ROM-BIOS, o apuntan hacia rutinas que no hacen nada, y que los programas suplirán posteriormente. Otra rutina de inicialización determina qué dispositivos están conectados al computador, y entonces sitúa un registro de ellos en localizaciones estándar de la parte baja de la memoria.

La parte final del procedimiento de puesta en marcha, después de los test del POST, el proceso de inicialización y la incorporación de las extensiones de la ROM , se denomina cargador BOOTSTRAP, y es una rutina corta que se emplea para cargar un programa

almacenado en el disco duro generalmente. En esencia, el cargador BOOTSTRAP intenta leer un registro, llamado registro de arranque, que está localizado en un disco, y si es satisfactorio el proceso, pasa el control del computador al programa que está incluido en aquel registro. Este programa tiene la tarea de cargar el resto del programa del disco. Si el cargador BOOTSTRAP no puede leer el registro cargador del disco, buscará en todos los dispositivos de almacenamiento que estén configurados para operar como arrancadores del sistema. Tan pronto como ocurre uno de estos dos procesos, el procedimiento de puesta en marcha del sistema termina, y los otros programas se encuentran listos para funcionar.

## 1.6 SISTEMA OPERATIVO FREEDOS

El sistema operativo para PC FREEDOS inició como un experimento cuando se escribían en lenguaje "C" controladores para dispositivos operando sobre el sistema operativo MS-DOS versión 3.1. Inicialmente se desarrollaron controladores que gestionaban datos en bloque o carácter, posteriormente se escribieron funciones para la gestión de la pila del procesador y conmutación de contexto utilizando un nuevo API creado como un pequeño sistema operativo llamado XDOS. Como mejora sustancial se agregó un cargador intermedio de programa (IPL) para configurar el entorno operativo antes de cargar el sistema operativo propiamente dicho. Adicionalmente se implementaron llamadas reentrantes de funciones para su uso en aplicaciones de tiempo real. Por último se agregaron los archivos básicos para la carga desde unidades de disco: Boot, io.sys, msdos.sys y command.com así como los programas utilitarios para conformar el sistema operativo FREEDOS **[25]**.

## 1.7 ARRANQUE DE FREEDOS

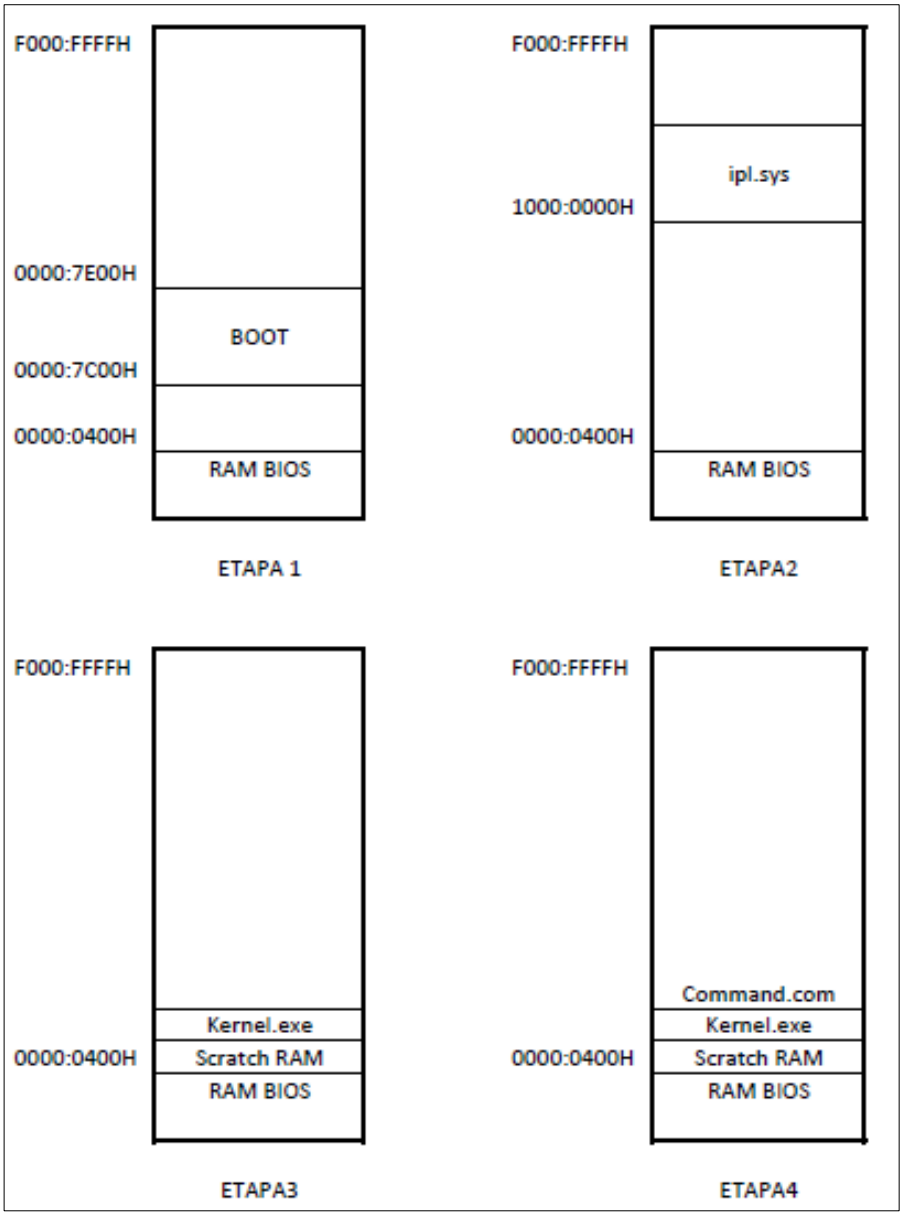
Al encender el PC y después de ejecutar el POST, el BIOS espera cargar el arranque del sistema operativo desde un medio de almacenamiento como un disquete, un disco duro

o un dispositivo USB si este es habilitado por el fabricante del PC; en la dirección segmentada de memoria 0000:7C00 H. Esta dirección es un estándar definido por IBM desde la creación del primer PC basado en el procesador 8088, la cual se mantiene para conservar la compatibilidad entre los sistemas PC basados en los procesadores INTEL x86. Después de cargar el sector de arranque (BOOT) el BIOS realiza una llamada a la dirección donde se encuentra cargado el BOOT SECTOR; este programa entonces verifica la existencia de un programa de carga intermedia IPL (io.sys e ibmdos.sys en MS-DOS). El IPL carga el núcleo (KERNEL) del sistema operativo responsable de las llamadas del sistema y manejo de los recursos en memoria y transfiere el control a éste (command.com en MS-DOS).

De la descripción anterior se pueden establecer 4 fases en la inicialización del sistema operativo en el PC. El proceso de carga del sistema operativo FreeDOS se describe en cuatro fases como se describe a continuación y se muestra en la figura 6:

- FASE 1. Carga el sector de arranque en la dirección 0000:7C00H, es realizada por el BIOS del PC.
- FASE 2. Apunta a la dirección 0000:7C00H y Carga el IPL (*initial program loader*).
- FASE 3. Carga el núcleo del sistema (Kernel.exe) en memoria y cede el control del microprocesador a éste.
- FASE 4. Carga el Shell de usuario (Command.com) y cede el control del PC a este programa. Después de esta fase se ofrece al usuario el prompt del sistema con lo que se muestra la disponibilidad para la carga de programas de aplicación, tales como el sistema de gestión y control del robot PUMA sobre el sistema operativo de tiempo real uC/OS-II.

Figura 6. Etapas de carga de FREEDOS.



## 1.8 ARQUITECTURA DEL NÚCLEO FREEDOS

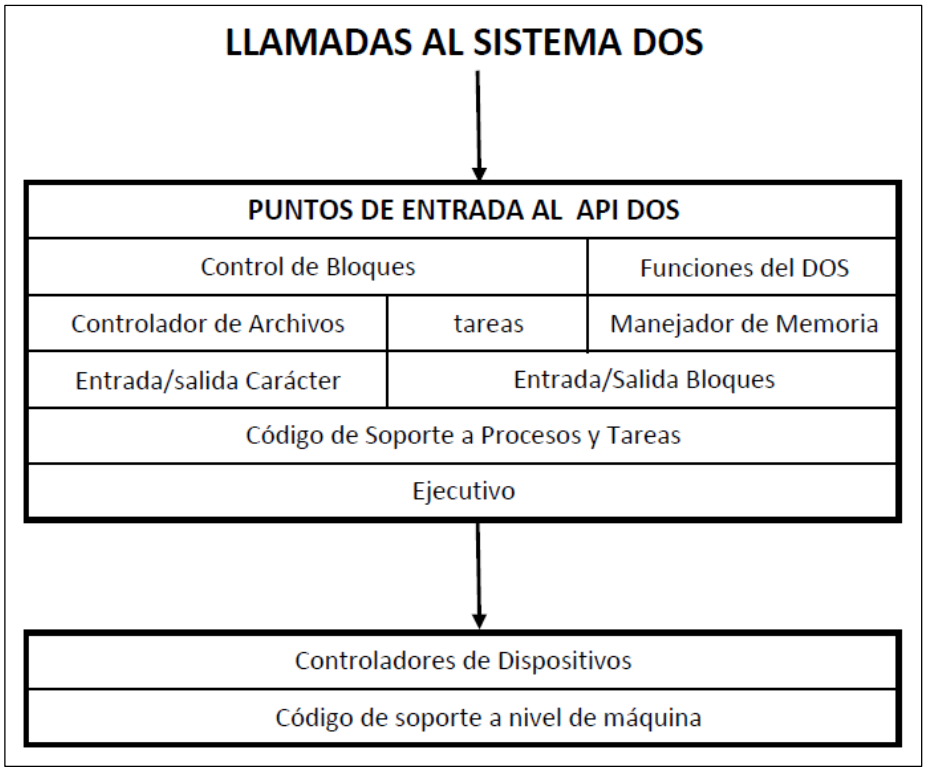
La arquitectura del núcleo de FREEDOS toma una configuración en capas como la mayoría de los sistemas operativos comerciales, la única desviación de los modelos comerciales se encuentra en la separación entre la capa superior y las capas inferiores. Para cumplir con la compatibilidad de los sistemas PC que han sido orientados a la operación con DOS es necesario crear 2 áreas de interfaz. La primera área de interfaz es el API compatible con el sistema operativo DOS el cual incluye las interrupciones 20H, 21H, 22H, 23H, 24H, 27H, 28H y 2FH. La otra área es la de controladores de dispositivos, que generalmente proporciona control directo de la máquina en lo referente a la manipulación de la pila y manejo de interrupciones.

## 1.9 INTERFAZ DE PROGRAMACIÓN DE APLICACIONES FREEDOS

El API (APLICATION PROGRAMMING INTERFACE) del FREEDOS intercepta las llamadas al sistema mediante las interrupciones software (DOS-COMPATIBLE SYSTEM CALLS) y las convierte en una llamada tipo "CALL" en lenguaje "C" (DOS API ENTRY POINTS). La función en C salva el contexto actual y genera una llamada al servicio apropiado (TASK AND PROCESS SUPORT CODE) quien ejecuta el controlador indicado desde el núcleo del sistema operativo (DEVICE DRIVERS). Los componentes de las áreas mencionadas se muestran en la figura 7. Cabe anotar que todas las llamadas a funciones que realiza el FREEDOS son gestionadas por las rutinas del sistema básico de entrada salida BIOS, para garantizar la compatibilidad tanto con plataformas antiguas como las plataformas actuales y las que aparezcan en el futuro próximo.

Por otra parte debe tenerse en cuenta que la herramienta de compilación que se use para el desarrollo de aplicaciones basadas en FREEDOS debe poder generar un RUNTIME compatible con el modo REAL de la arquitectura x86 **[25]**.

Figura 7. Arquitectura del núcleo del FREEDOS.




## 1.10 INTERRUPCIONES DE FREEDOS

El API de FREEDOS se caracteriza por disponer de múltiples funciones mediante una única instrucción INT. Como ejemplo se puede mencionar la INT 21H. Para seleccionar la función se carga en el registro AX del procesador el número de la función correspondiente y en otros registros de la CPU se cargan otras variables aplicables a la llamada del sistema particular o se cargan apuntadores al área de memoria pertinente. FREEDOS ejecuta entonces la función llamada y retorna el resultado de la misma en registros o posiciones de memoria apuntadas por registros. En la tabla 2 se muestran las interrupciones de FREEDOS.

Tabla 2. Interrupciones para llamadas del sistema FREEDOS.

| SISTEMA DE LLAMADO DE INT DEL FREEDOS | |
|---|---|
| **INTERRUPCIÓN** | **FUNCIÓN** |
| 20H | Terminación de Programa |
| 21H | Llamada del sistema |
| 22H | Dirección de finalización |
| 23H | Dirección de ruptura |
| 24H | Error crítico |
| 25H | Lectura absoluta en disco |
| 26H | Escritura absoluta en disco |
| 27H | Terminar cualquier programa residente |
| 28H | Tarea de Soporte |
| 29H | Consola rápida de I/O |
| 2AH | Manejador de RED |
| 2BH | No definido para FREEDOS |
| 2CH | No definido para FREEDOS |
| 2DH | No definido para FREEDOS |
| 2EH | Recarga de command.com |
| 2FH | Interrupción multiplexada |
| 30H | Saltos FAR para entrada DOS |

## 1.11 SISTEMAS OPERATIVOS DE TIEMPO REAL

### 1.11.1 Conceptos básicos

Los sistemas operativos de tiempo real se caracterizan por realizar de manera determinística todas las tareas que se le confieren con restricciones precisas en cuanto al retardo o latencia de las mismas **[2]**.

Existen 2 tipos de RTOS (REAL TIME OPERATING SYSTEMS): los llamados SOFT que se caracterizan por darle mayor relevancia a la realización correcta del proceso, tolerando un poco la imprecisión en el manejo del tiempo; y los llamados HARD, que enfatizan en el manejo preciso y determinístico del tiempo así como la realización correcta de las tareas. Según la organización de las tareas podemos decir que los hay colaborativos

conducidos por eventos FOREGROUND/BACKGROUND y, de programación del tiempo de tarea mediante la conmutación de la CPU; MULTITASKING el cual de acuerdo con el manejo de prioridad pueden ser con manejo de preferencia por rango (PREEMPTIVE) o sin preferencia de rango (NON_PREEMPTIVE) **[13]**.

### 1.11.2 Sistemas FOREGROUND / BACKGROUND.

En este tipo de sistemas operativos el nivel de interrupciones (FOREGROUND); es implementado mediante rutinas de servicio que realizan las tareas críticas del sistema disparadas generalmente por eventos asincrónicos debido a señales externas. El nivel de tarea (BACKGROUND) generalmente consiste en un ciclo infinito que llama funciones para realizar las tareas deseadas. El tiempo de respuesta en el nivel de la tarea depende de qué tanto tiempo tome en ejecutarse el nivel de interrupción el cual típicamente no es constante y en múltiples pasadas de éste código el tiempo puede ser no determinístico.

### 1.11.3 Sistemas MULTITASKING.

Los sistemas multitarea corresponderían a un sistema FOREGROUND/BACKGROUND con múltiples BACKGROUND, y se caracterizan por la programación y conmutación secuencial de la CPU entre las diferentes tareas permitiendo la ejecución de cada una de ellas en un tiempo previamente establecido. La multitarea maximiza el uso de la CPU y facilita la implementación modular de las aplicaciones. En esta estrategia, programas complejos son divididos en secciones modulares que pueden ser sincronizados y mantenidos apropiadamente. MicroC/OS-II es un ejemplo de sistema operativo de tiempo real multitarea.

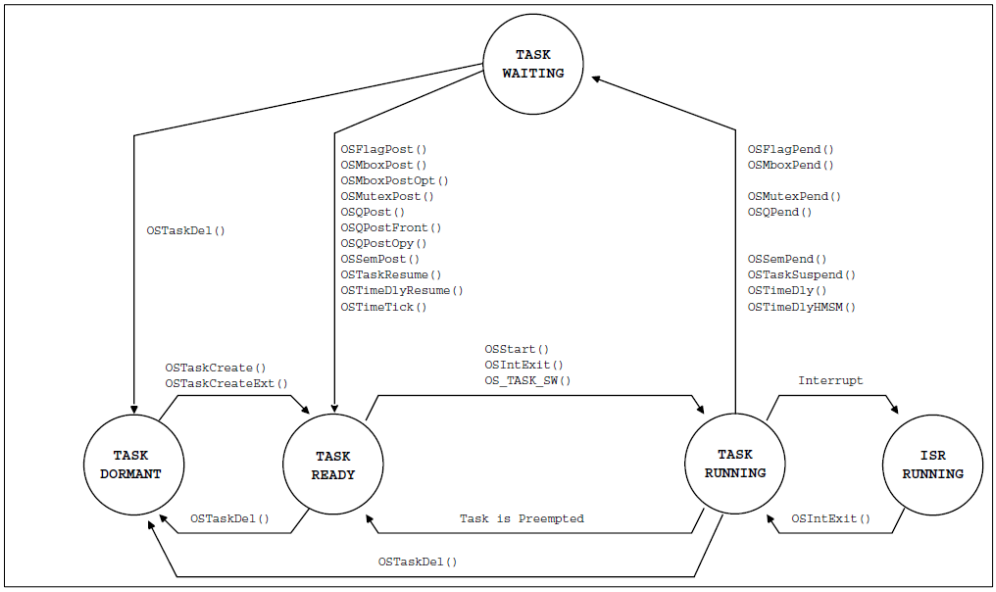
### 1.11.4 Tarea (TASK).

La tarea es el componente básico de los sistemas multitarea. También se conoce como hilo y puede ser definido como un programa que cree tener la CPU sólo para él. En el diseño de aplicaciones de tiempo real el trabajo se divide en múltiples tareas que se responsabilizan de parte de la solución únicamente, y a cada tarea le es asignada una prioridad y un grupo de registros de la CPU para su uso. Las tareas pueden asumir 5 diferentes estados en un sistema operativo de tiempo real: Latente (Dormant), lista para ejecutarse (Ready), ejecutándose (Running), esperando un evento (Waiting) o interrumpida (ISR interrupted). En la figura 8 se muestran estos estados y las funciones correspondientes en el uCOS-II **[13]**.

### 1.11.5 El Kernel.

Es la parte de un sistema multitarea responsable de manejar las tareas comunicándolas y asignándoles tiempo de CPU. Involucra un programa llamado despachador o ejecutivo del sistema, que establece el turno de ejecución de cada tarea basado en un esquema de prioridad la cual es establecida en función de la importancia de la misma. En un kernel basado en la prioridad de las tareas el control de la CPU es siempre otorgado a la tarea de mayor prioridad que esté lista para ejecutarse. El tiempo de CPU en el que se ejecuta la tarea de mayor prioridad depende del tipo de manejo de la prioridad: Sin manejo de preferencia por rango (NON-PREEMPTIVE) o con preferencia de rango (PREEMPTIVE).

Figura 8. Estado de las tareas.

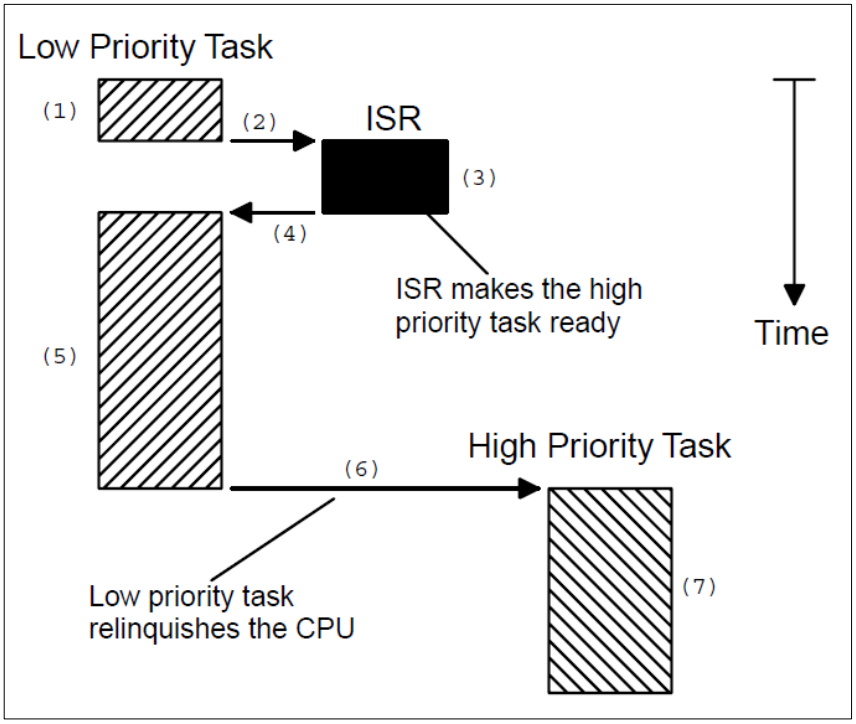



### 1.11.6 Kernel sin manejo de preferencia por rango.

Las tareas solicitan de forma explícita al despachador que requieren el uso de la CPU de modo que para generar la ilusión de tiempo real el proceso descrito debe ser realizado periódicamente. La programación no preferente es llamada comúnmente multitarea cooperativa ya que todas las tareas cooperan con cada una de las otras para compartir el tiempo de CPU. Los eventos asincrónicos son atendidos por la rutina de tratamiento de interrupciones y esta a su vez puede cambiar la prioridad de una tarea lista para correr llevándola al nivel más alto. Esta tarea tomará el control sólo cuando la tarea interrumpida renuncie al control de la CPU; es decir, en un sistema sin manejo preferente de prioridad cada tarea corre hasta cuando ésta voluntariamente renuncia al control de la CPU. Dado el caso que una rutina de servicio de interrupción cambie el estado de una tarea con mayor prioridad a lista para correr, ésta tendrá el control

cuando la tarea interrumpida se complete o decida renunciar al control de la CPU. Este proceso se muestra en la figura 9.

Figura 9. Kernel sin manejo de preferente.

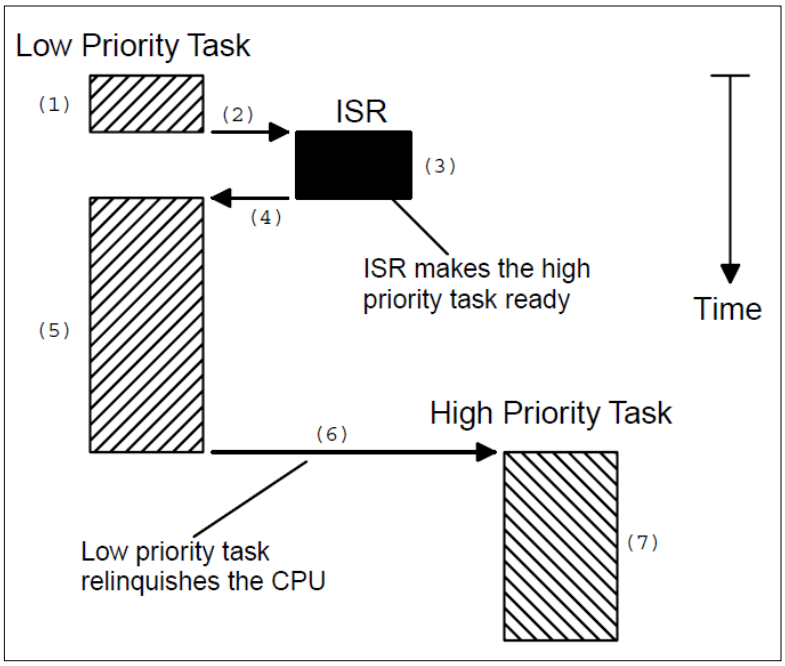



### 1.11.7 Kernel con manejo de preferencia por rango.

En este modelo, siempre la tarea lista para correr de mayor prioridad obtendrá el control de la CPU. Cuando una tarea o una interrupción pone en estado de lista para correr a otra tarea de mayor prioridad, la tarea que se está ejecutando es suspendida por la condición de preferencia y el control de la CPU es entregado a la tarea de mayor prioridad. En el caso en el que una interrupción cambie el estado de una tarea de mayor

prioridad a lista para correr, la tarea interrumpida será suspendida y el control de la CPU se le entregará a la tarea con mayor prioridad al finalizar la interrupción. Cuando la tarea de mayor prioridad finalice o renuncie al control de la CPU éste será entregado a la tarea suspendida por acción de la interrupción. Este proceso se muestra en la figura 10.

Figura 10. Kernel con manejo de preferencia.




### 1.11.8 Programación de tareas Round-Robin.

Cuando dos o más tareas tienen la misma prioridad, para resolver la situación el *kernel* permite correr un tiempo predeterminado a una tarea y luego el mismo tiempo a la otra. Este tiempo se conoce como *QUANTUM*. A este proceso se le conoce como programación *ROUND-ROBIN* o desplazamiento de ventana de tiempo.

# 2. SISTEMA OPERATIVO DE TIEMPO REAL uC/OS-II

## 2.1 CARACTERÍSTICAS

El sistema operativo de tiempo real uCOS-II está totalmente escrito en ANSI-C y cumple con la certificación DO-178B para ser usado en aviónica, la certificación SIL3/SIL4 IEC para sistemas de transporte y manejo de energía nuclear, 99% de los requerimientos de la industria de motores de EEUU y cumple con el estándar MISRA-C: 1998 de la asociación de confiabilidad del software en lo referente a la generación de código C. Provee mecanismos para la programación y sincronización entre procesos con un manejo de tiempo determinístico. Entre sus características relevantes presenta las siguientes:

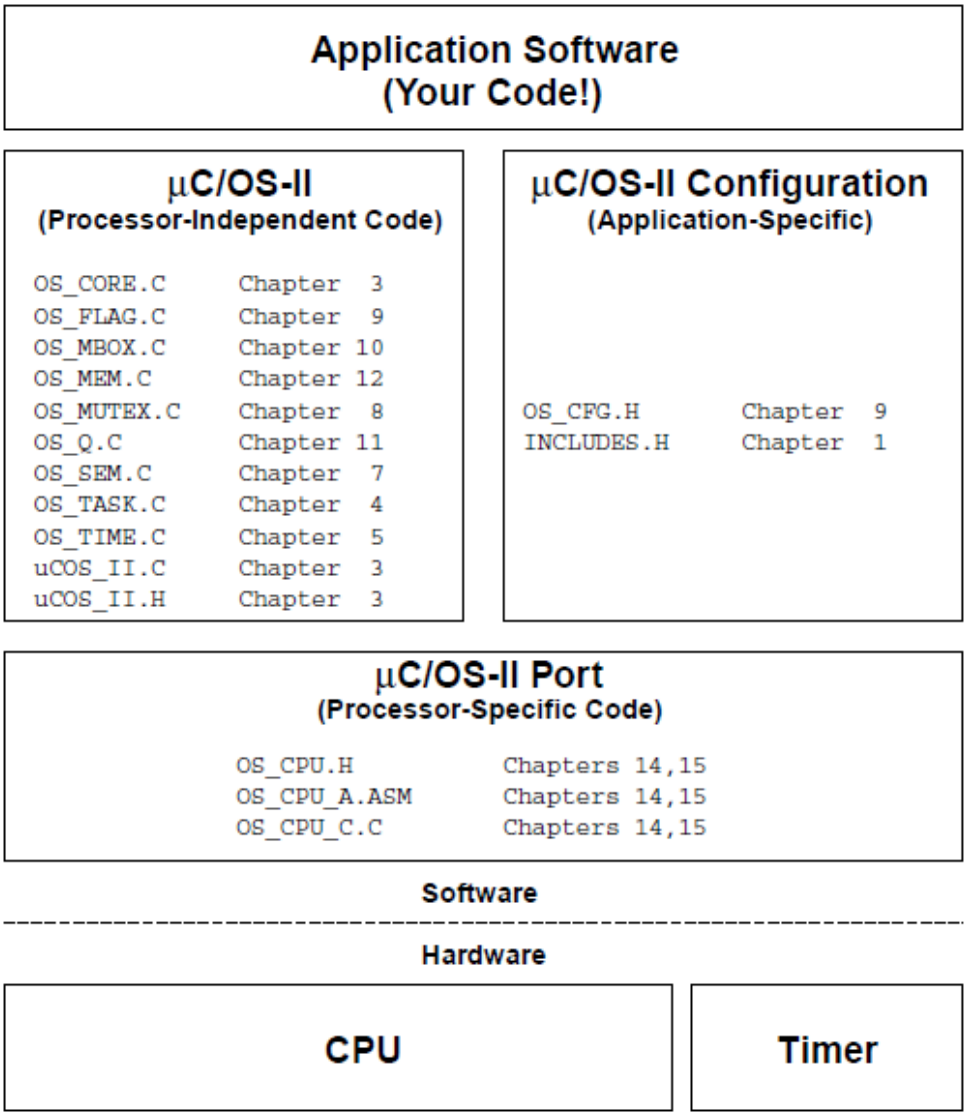
- ✓ Kernel multitarea con manejo de preferencia en tiempo real.
- ✓ Semáforos con exclusión mutua y protocolo para prevenir la inversión de prioridad en las tareas.
- ✓ Expiración de tiempos en llamadas pendientes para evitar deadlocks.
- ✓ Manejo máximo de 254 tareas por aplicación (Prioridad única por tarea) y número ilimitado de objetos del kernel.
- ✓ Altamente escalable (6KBytes – 24KBytes) de espacio de código y 1Kbyte de espacio de datos.

## 2.2 ESTRUCTURA DE ARCHIVOS

La estructura de archivos de uC/-OS-II está organizada en cuatro capas que le permiten independencia del código con respecto al hardware utilizado, siendo fácilmente

migrable hacia diferentes plataformas. Como se muestra en la figura 11, las tres capas inferiores corresponden al código del kernel y corresponden a las llamadas: Capa de código independiente del procesador, capa de código específico del procesador y capa de código específica del procesador.

Figura 11. Estructura de archivos del uC/OS-II



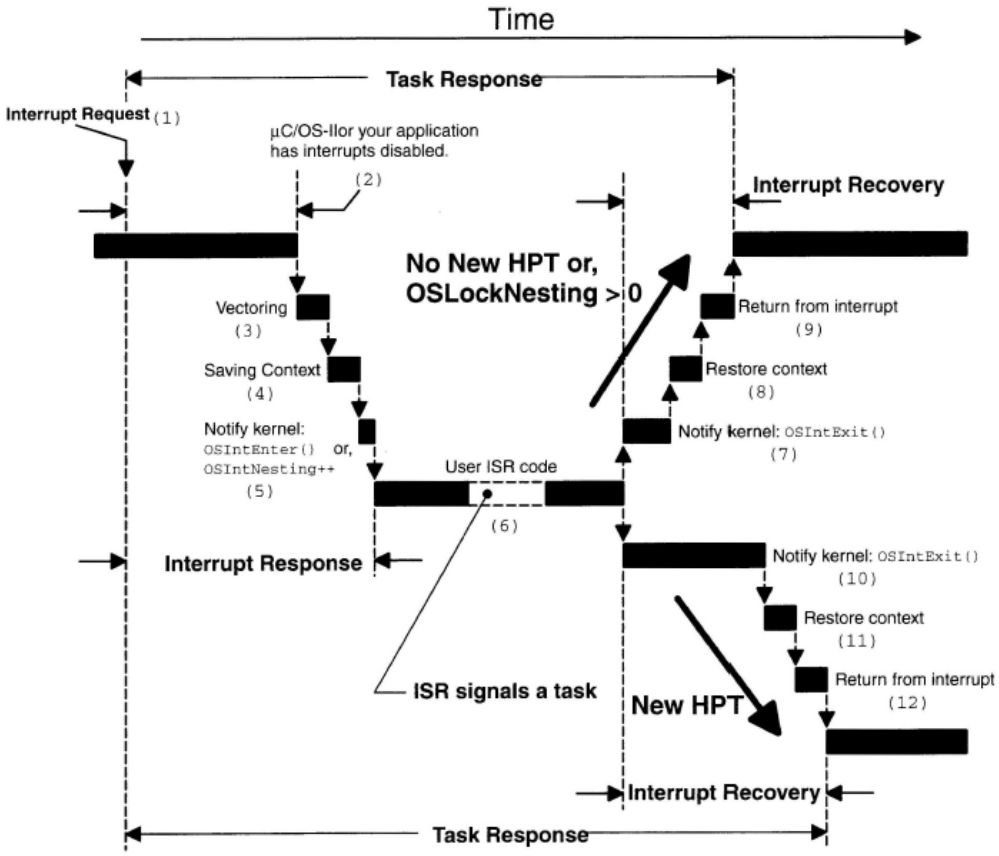
La capa de código específico del procesador contiene funciones para interactuar con la plataforma de hardware en la cual es instalado el sistema operativo. Estás funciones son particulares de esa plataforma y son necesarias para la operación de la capa superior, conmutación de contextos de procesos y habilitación/deshabilitación de interrupciones. La capa de código independiente del procesador corresponde a la programación de los mecanismos de alto nivel para el manejo de memoria y gestión de semáforos, colas, tareas, etc. La capa de código específico de la aplicación corresponde a los métodos de configuración para el software desarrollado a nivel de usuario.

## 2.3 SECCIONES CRÍTICAS

Los sistemas operativos de tiempo real requieren desactivar las interrupciones para acceder a las secciones de código crítica (aquellas que dan servicio a múltiples aplicaciones como son los buffers circulares, dispositivos de adquisición de datos, puertos seriales, etc.) con el fin de salvaguardar la integridad de los datos para evitar el colapso de las aplicaciones. El tiempo de desactivación de las interrupciones es uno de los parámetros de gran relevancia para el caso de tiempo real debido a que afecta los tiempos de respuesta a eventos asíncronos llegando incluso a la pérdida de los mismos cuando este tiempo es muy extenso.

Para el caso de la arquitectura x86, el sistema operativo ofrece 3 métodos para gestionar esta situación, siendo el uso de la instrucción de volcado de registros a la pila mediante el manejo del apuntador de espacio de trabajo provisto por Intel para éste propósito el cual deshabilita automáticamente al iniciar el volcado de los registros a la pila, y los habilita en igual forma al terminar la descarga. En la figura 12 se evidencia el proceso de gestión y servicio de interrupciones.

Figura 12. Gestión y servicio de interrupciones




## 2.4 TIEMPO DE CPU POR TAREA (Tick clock)

El tiempo que cada tarea dispone de la CPU para adelantar ó terminar sus procesos es conocido como tick del sistema. Este tiempo actúa a modo de metrónomo conmutando cada tarea para ser atendida por la CPU según un orden de prioridad dinámico ó estático (Preestablecido); puede asociarse a un proceso de multiplexado en cuyo tiempo una tarea cree tener todos los recursos de hardware disponibles, para sí. Para el caso de la arquitectura x86 éste tiempo es establecido por el sistema básico de

entrada salida BIOS y es configurado típicamente en 56 mS siendo una frecuencia aproximada de 18 Hz. En un sistema operativo de tiempo real esta tasa debe generarse entre 10 y 100 veces la frecuencia mencionada, para plataformas operando con procesadores de 32 bits del tipo 80386DX funcionando a 33 MHz. Con procesadores de mayor desempeño como los actuales este valor puede ser superado ampliamente con la restricción impuesta por los tiempos de respuesta de los periféricos involucrados, los cuales son mucho más largos que los presentados por el procesador.

## 2.5 MANEJO DE TAREAS

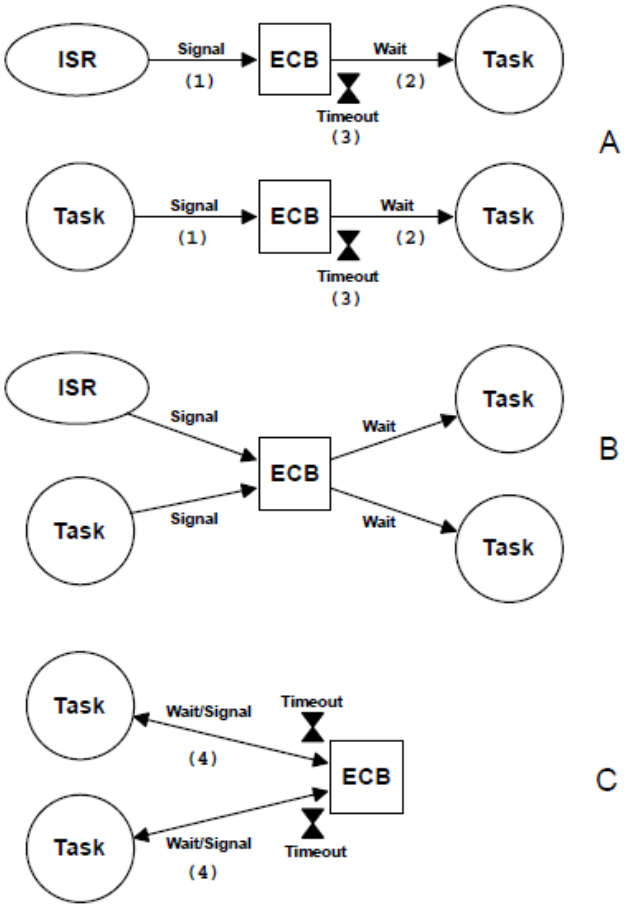
Una tarea es en un sistema operativo de tiempo real una función de lazo infinito o una función que se elimina a ella misma cuando este lazo es ejecutado (Deja de ser infinito). Una función en C es vista como un proceso que retorna con un parámetro al llamador de la misma. En el caso de una tarea; esta función no retorna y puede ser puesta en diferentes estados desde la perspectiva de la CPU: Corriendo (Tiene el control de la CPU actualmente), esperando, durmiendo, lista para correr o una interrupción se ha generado. Las tareas pueden ser creadas, borradas, modificadas en su orden de prioridad de ejecución, suspendida, reactivada y retardada; para lo cual el sistema operativo provee funciones previamente programadas listas para ser utilizadas por el usuario.

## 2.6 BLOQUE DE CONTROL DE EVENTOS

Corresponde a un objeto del sistema operativo de tiempo real que le permite al kernel del mismo implementar la interacción de tareas entre sí, así como la interacción entre interrupciones y tareas como se muestra en la figura 13, donde la señalización es considerada un evento para efectos de la interacción de las entidades mencionadas.

Un bloque de control de eventos es utilizado como base para la implementación de mecanismos de alto nivel para la gestión de datos entre tareas tales como manejo de semáforos, semáforos de exclusión mutua, cajas de mensajes y colas de mensajes.

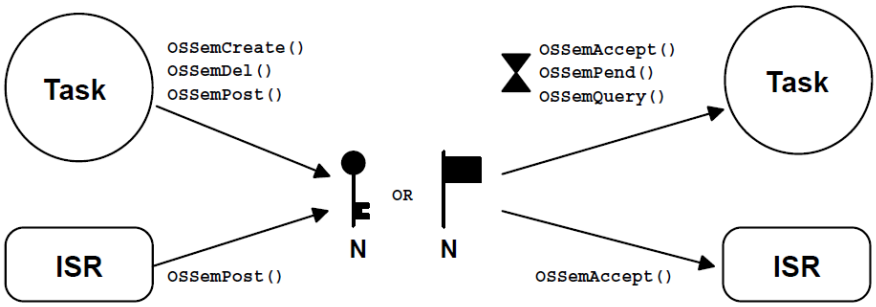
Figura 13. Bloque de control de eventos



### 2.6.1 Semáforos

Puede definirse como un mecanismo de sincronización del intercambio de datos entre varias tareas o entre una interrupción y una tarea. Generalmente está compuesto por un vector de 16 bits o más que actúa como apuntador a una lista de tareas que espera por este recurso en un orden preestablecido. uC/OS-II provee servicios para: Crear, aceptar, señalizar, liberar e indagar por un semáforo. Es de anotar que los semáforos pueden programarse para expirar mediante un parámetro que es pasado al servicio en el llamado del mismo. La figura 14 muestra la representación gráfica de la operación de este mecanismo de manejo de datos.

Figura 14. Relación entre tareas, interrupciones y semáforos.



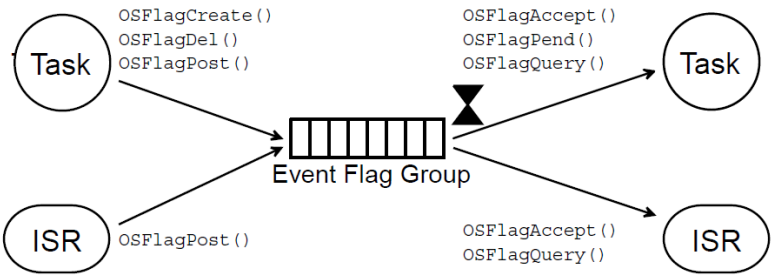

### 2.6.2 Semáforos de Exclusión Mutua

Los semáforos de exclusión mutua son usados por las tareas para ganar acceso exclusivo a los recursos del sistema. Suelen ser utilizados para reducir el problema de inversión de prioridad que se presenta cuando un recurso al que ha ganado acceso una tarea de baja prioridad es requerido por una tarea de mayor prioridad.

### 2.6.3 Banderas de Eventos

Este es un mecanismo provisto por uC/OS-II para mantener el estado actual de un grupo de eventos señalizados por un bit cada uno el cual puede tomar los valores cero y uno (binario), asociados a una tarea que espera por la ocurrencia de una combinación preestablecida de estos bits. Están constituidos por 2 elementos: El grupo de bits y una lista de tareas que esperan por la ocurrencia de una combinación de ellos. El sistema operativo provee servicios para: Crear, aceptar, señalizar, liberar e indagar por estos bits. En la figura 15 se representa la operación de este mecanismo.

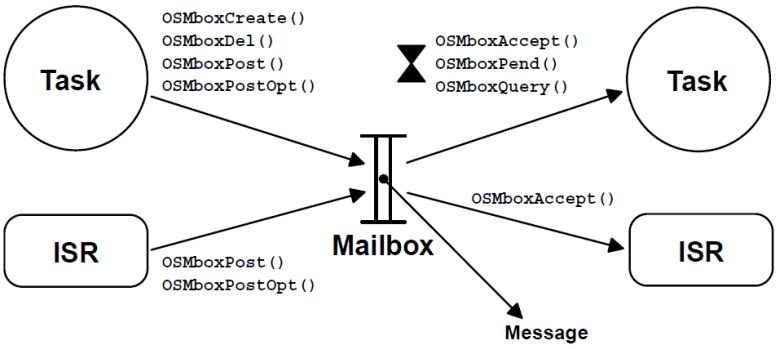
Figura 15. Manejo de banderas de eventos




### 2.6.4 Cajas de mensaje Mailbox

Un Mailbox es un objeto del sistema operativo de tiempo real que le permite a una tarea o a una interrupción; enviar a otra tarea, un apuntador al inicio de una estructura de datos donde se encuentra el mensaje con los datos que se desean pasar. Es útil para transferir datos entre buffer de periféricos tales como video, e interfaces de comunicación entre otros. El sistema uC/OS-II provee servicios para: Crear, aceptar, señalizar, liberar e indagar un Mailbox como se representa en la figura 16.

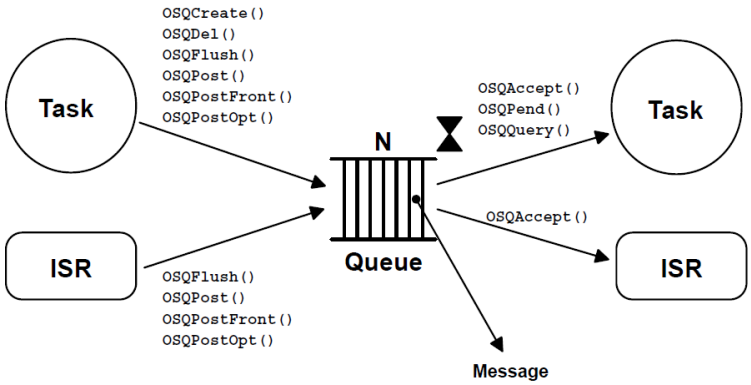
Figura 16. Relación entre tareas, interrupciones y un Mailbox.




### 2.6.5 Colas de Mensajes

Una cola de mensajes es un objeto Mailbox múltipe que provee el sistema operativo para gestionar mensajes encolados cuando una tarea o ISR escribe más rápido que la tarea que lee los mensajes. El sistema uC/OS-II provee para este objeto los mismos servicios que para los Mailbox como se presenta en la figura 17.

Figura 17. Relación entre tareas, interrupciones y una cola Mailbox.



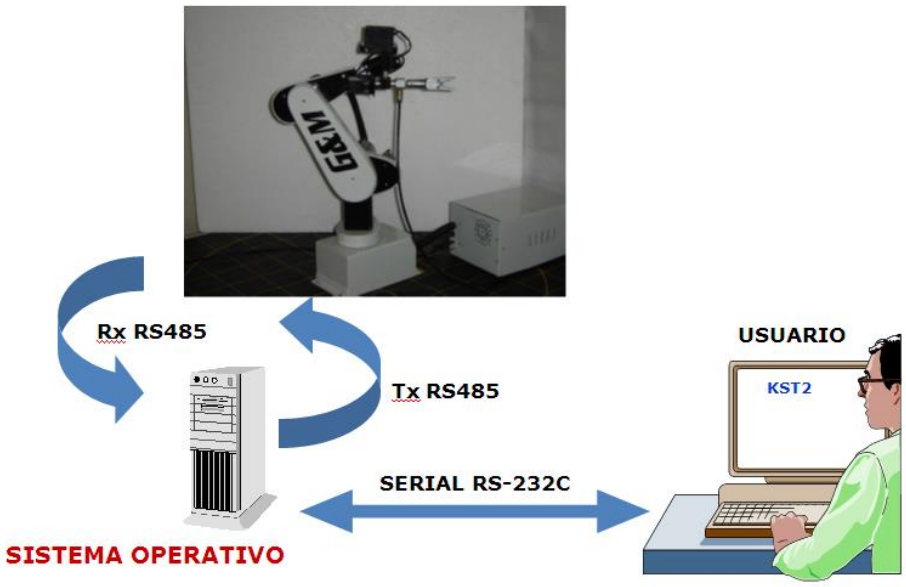
# 3. DESARROLLO DEL TRABAJO DE INVESTIGACIÓN

Realizada la ubicación de la temática en el marco de referencia pertinente al hardware y software, se presentan los componentes y herramientas usadas para el desarrollo del trabajo de investigación así como los métodos y procedimientos implementados.

## 3.1 SOLUCIÓN IMPLEMENTADA

Al pretender desarrollar un sistema Hardware y Software de propósito general que opere en tiempo real y específicamente, para efectos de validación un sistema de control de lazo cerrado aplicado a un robot de seis grados de libertad; se optó por la implementación de dos plataformas Hardware, una basada en un mother board de PC con arquitectura x86 encargada de ejecutar la aplicación de control de posición del robot PUMA, y las comunicaciones entre éste y la terminal remota del usuario sobre el sistema operativo de tiempo real uC/OS-II. La otra plataforma hardware basada en un microcontrolador PIC18F876, encargada de adquirir la información de posición entregada por tres potenciómetros lineales acoplados a las articulaciones que emulan la cintura, hombro y codo del brazo robot; de igual forma encargada de gestionar la posición de los motores mediante los valores entregados por el control central a través de los PWM de estas plataformas. En cada una de estas plataformas se implementó una aplicación de control; la del PC para operar el sistema en el modo remoto, requiriendo periódicamente la posición de los motores para realizar el compensador PID y entregar los valores PWM a cada articulación. La del PIC para operar como sistema de adquisición – actuación cuando se opera en el modo remoto y como controlador embebido cuando se opera en el modo local. Una interfaz de conversión RS232-RS485 para conectar en el PC y un convertidor IRDA-RS232 para conectar el PC remoto del usuario. En la figura 18 se presenta una idea general del sistema. Cada uno de los componentes que conforman la solución se presentan en los siguientes numerales.

Figura 18. Diagrama de la solución implementada.




## 3.2 EL ROBOT PUMA

Como herramienta para la validación del desarrollo sobre el sistema operativo de tiempo se usó un robot tipo PUMA de seis grados de libertad proporcionado por la escuela de ingenierías Eléctrica, Electrónica y Telecomunicaciones de la UIS el cual fue desarrollado en un proyecto anterior **[6]**, el cual presenta las siguientes características:

### 3.2.1 Características mecánicas

Robot de tipo angular con seis grados de libertad constituido por cinco partes mecánicas:

- **Base:** Soporta toda la estructura del brazo, aloja los sistemas de control electrónico y en ella se encuentra la primera articulación que emula el movimiento de la cintura del manipulador y que está construida alrededor de un motor de que opera a 12VDC y consume sin carga de 130 mA con una rotación de 25 RPM logrando un torque de 400 onzas/pulgada. Está asociada a un sistema de transmisión mecánica que permite un movimiento angular de 220°, a la cual se acopla un sensor de posición angular (potenciómetro lineal) para realimentar la información de esta variable al controlador digital de posición. (La configuración y características enunciadas aplican también para las articulaciones del hombro y codo).
- **Brazo:** Soporta y desplaza mecánicamente el antebrazo, muñeca y pinza mediante transmisión tipo piñón corona la cual está unida mediante correa dentada. En la corona mencionada se acopla un sensor de posición angular que permite realimentar un desplazamiento de hasta 220°. Corresponde a la segunda articulación del manipulador y emula los movimientos del hombro.
- **Antebrazo:** Soporta mecánicamente a la muñeca y la pinza mediante una transmisión similar a la implementada en el brazo. Corresponde a la tercera articulación del manipulador y con desplazamiento máximo de 220°. Emula los movimientos del codo.
- **Muñeca:** Implementada mediante tres ejes unidos de manera secuencial, (cada eje se acopla al subsiguiente) controlados cada uno por un servomotor tipo RC que proporcionan movimientos circulares con el fin de orientar al efector final o pinza.
- **Pinza:** Consta de un *gripper* de accionamiento automático fabricado por la firma SAS *automation* con una presión de gestión límite de 87 psi. Las partes o secciones descritas se pueden apreciar en la figura 19.

Figura 19. Robot PUMA de 6 grados de libertad.

## 3.3 DESARROLLO DE HARDWARE

Los componentes y sistemas electrónicos utilizados para la implementación de la interconexión de los sistemas operativos propuestos se dividieron en tres categorías:

1. Tarjeta *Mother Board* con procesador Intel PENTIUM IV, memoria RAM de 1GByte, controlador de video embebido , sin disco duro con puertos USB y puerto serial RS-2332 e Irda.
2. Tarjeta propietaria desarrollada alrededor de la plataforma MICROCHIP PIC16F876; que incluye un microcontrolador de 8 bits, sistema puente H para gestión de potencia a los motores e interfaz de comunicaciones RS-485 para interconectarse mediante red diferencial con protocolo propietario conducido mediante la técnica *token pass*, al sistema PC quien actúa como coordinador de los nodos correspondientes a cada uno de los motores del robot.
3. Convertidor de protocolo IRDA – RS232 **[16]**, para realizar la interfaz de usuario a un computador remoto mediante conexión serial. Se describen a continuación las características de estos tres componentes de la solución.

### 3.3.1 Plataforma PC

La Plataforma PC usada para la implementación de los algoritmos de control y comunicaciones a ser realizados sobre el sistema operativo del tiempo real para gestionar y controlar un robot tipo PUMA de seis grados de libertad fue un *Mother-board* ASROCK P4i65G revisión 6/A 1.01 **[1]**, que se muestra en el esquemático de la figura 20 y el hardware real en la figura 21. Las características sobresalientes de esta plataforma son:

Procesador Intel Pentium IV de 2.4 GHz.

Memoria RAM de 256 MBytes.

6 Puertos USB 2.0.

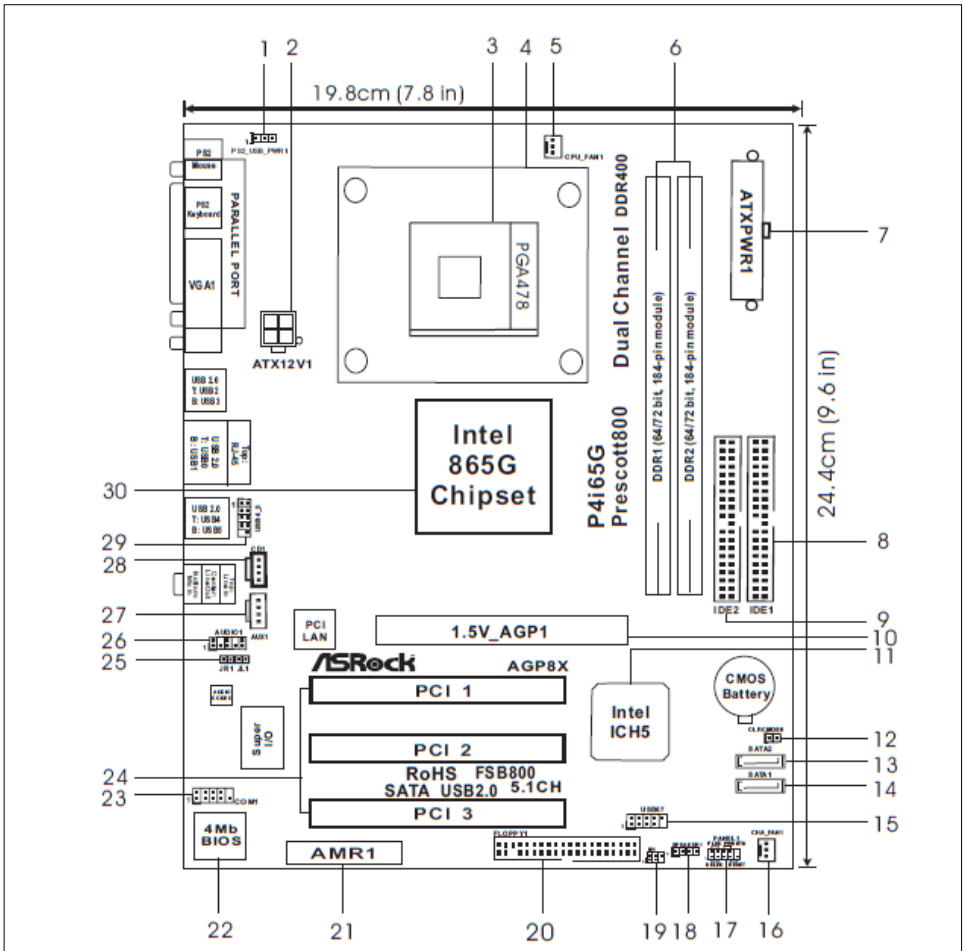
2 Puertos seriales (1 RS-232, 1 IRDA) y 1 puerto paralelo.

1 Salida de video VGA.

2 Interfaces IDE de alta velocidad.

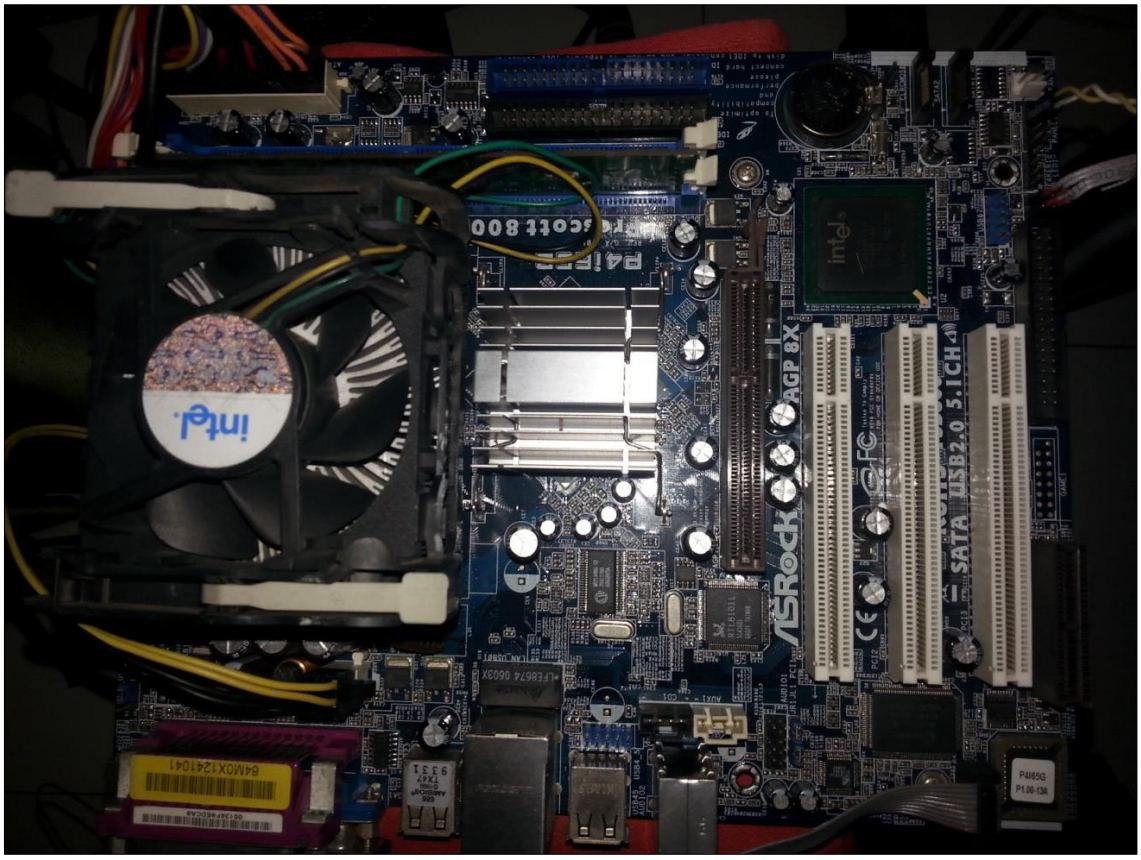
1 Interfaz para unidad de discos flexibles.

Figura 20. Esquemático con ubicación física de componentes en board Intel-PC



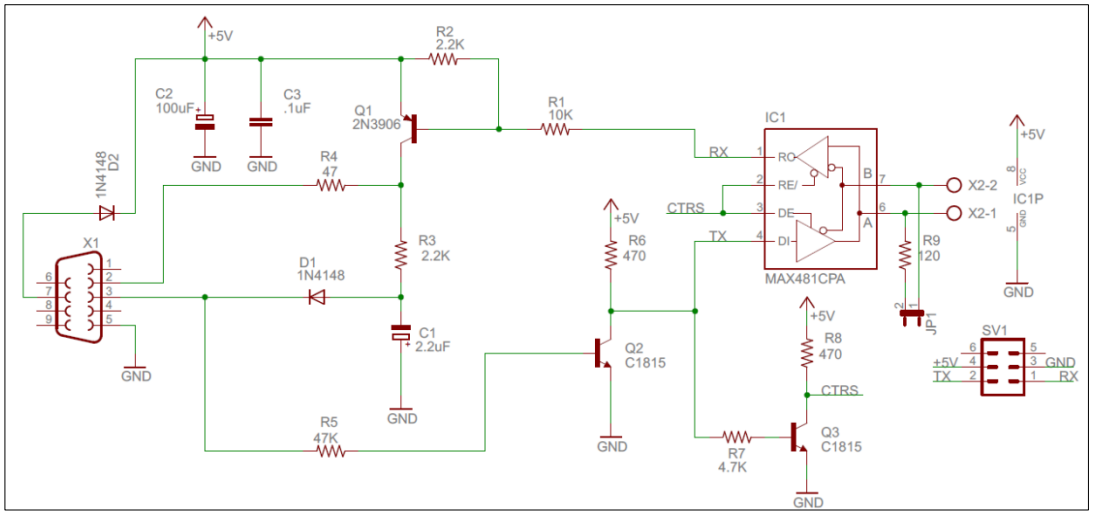

| | | | |
|---|---|---|---|
| 1 | PS2_USB_PWR1 Jumper | 16 | Chassis Fan Connector (CHA_FAN1) |
| 2 | ATX 12V Connector (ATX12V1) | 17 | System Panel Header (PANEL1) |
| 3 | P4-478 CPU Socket | 18 | Chassis Speaker Header (SPEAKER 1) |
| 4 | CPU Heatsink Retention Module | 19 | Infrared Module Header (IR1) |
| 5 | CPU Fan Connector (CPU_FAN1) | 20 | Floppy Connector (FLOPPY1) |
| 6 | 184-pin DDR DIMM Slots (DDR1- 2) | 21 | AMR Slot (AMR1) |
| 7 | ATX Power Connector (ATXPWR1) | 22 | BIOS FWH Chip |
| 8 | Primary IDE Connector (IDE1, Blue) | 23 | Serial Port Connector (COM1) |
| 9 | Secondary IDE Connector (IDE2, Black) | 24 | PCI Slots (PCI1- 3) |
| 10 | AGP Slot (1.5V_AGP1) | 25 | JR1 / JL1 Jumpers |
| 11 | South Bridge Controller | 26 | Front Panel Audio Header (AUDIO1) |
| 12 | Clear CMOS Jumper (CLRCMOS0) | 27 | Internal Audio Connector: AUX1 (White) |
| 13 | Secondary Serial ATA Connector (SATA2) | 28 | Internal Audio Connector: CD1 (Black) |
| 14 | Primary Serial ATA Connector (SATA1) | 29 | Shared USB 2.0 Header (USB4_5, Blue) |
| 15 | USB 2.0 Header (USB67, Blue) | 30 | North Bridge Controller |

Figura 21. Plataforma utilizada para la implementación en PC.



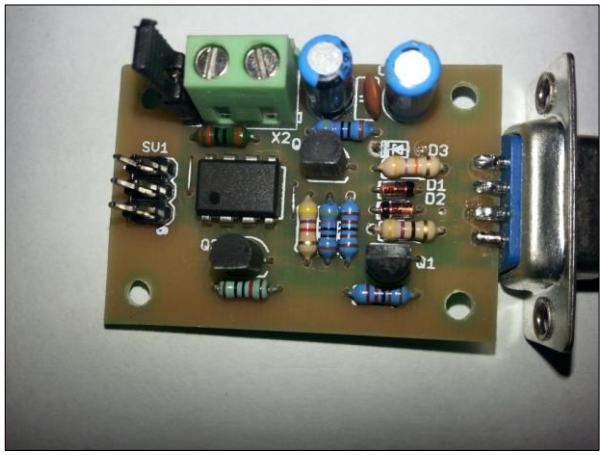

Ya que la interfaz de comunicaciones necesaria para comunicarse con el sistema de control del robot PUMA es serial RS485, Se requirió implementar un convertidor RS232 a RS485 a fin de tener acceso a los controladores de cada motor en el robot. Para habilitar este puerto fue necesario construir una interfaz para adaptar los niveles de voltaje RS-232 a RS485 mediante el circuito presentado en la figura 22 el cual se basa en el microcontrolador PIC12F1822 **[15]**, así como la programación de las funciones pertinentes en los módulos de recepción y transmisión serial para evitar la colisión de datos en esta red *half duplex*.

Figura 22. Esquemático convertidor RS232-RS485.




En la figura 23 se muestra el circuito diseñado, completamente terminado.

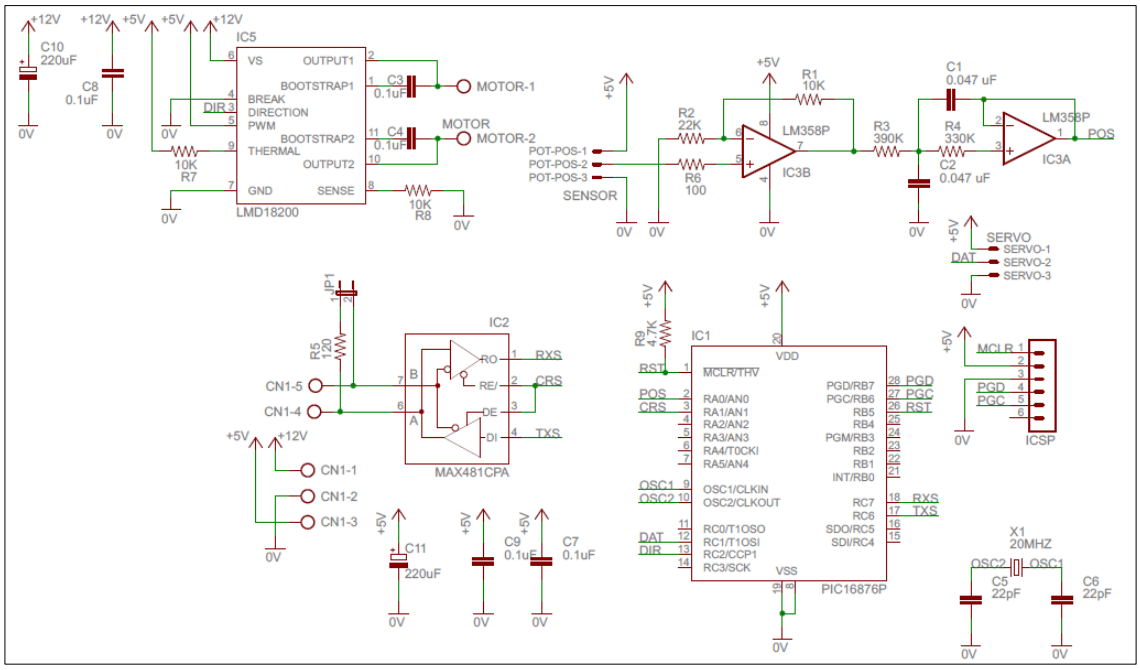
Figura 23. Circuito convertidor RS232 – RS485 terminado.



### 3.3.2 Plataforma propietaria ComPID – RS485

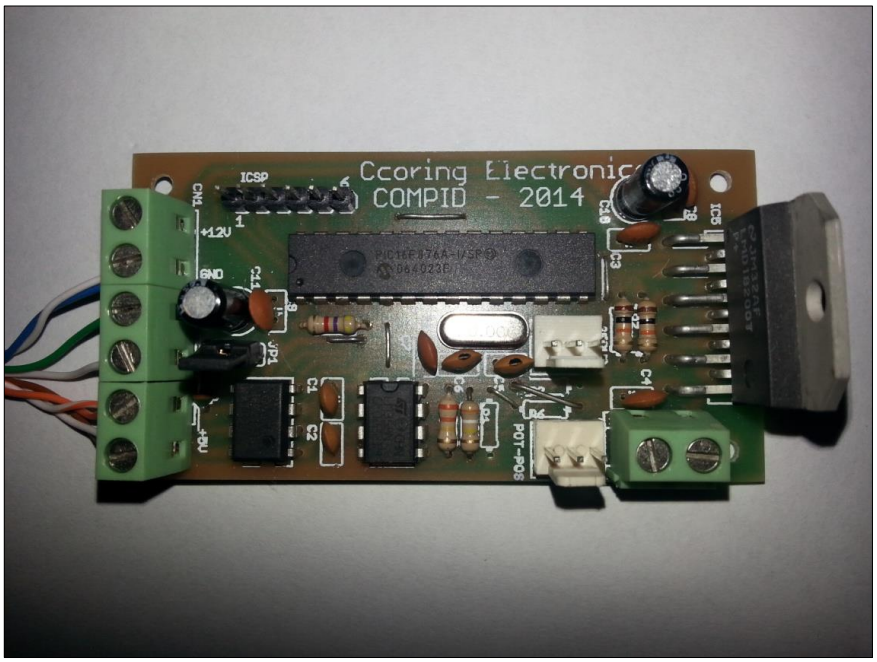
Desarrollada alrededor del microcontrolador PIC16F876 **[16]**, de la firma MICROCHIP, cuenta con un convertidor análogo digital de 10 bits para la captura de la posición angular medida mediante un potenciómetro lineal acoplado mecánicamente a la articulación a controlar en el robot PUMA. Está provisto de un puente H en la configuración bipolar lo que permite frenado dinámico del motor conectado a este dispositivo y que corresponde al actuador de la articulación. De igual forma cuenta con una interfaz RS485 para comunicarse serialmente con el coordinador de la red el cual puede ser un PC con sistema operativo de tiempo real. El esquemático de la plataforma implementada se muestra en la figura 24.

Figura 24. Esquemático plataforma ComPID.



El circuito electrónico montado en PCB y finalizado se muestra en la figura 25. Este circuito es replicado en la solución para cada los motores correspondientes a las articulaciones, cadera, hombro y brazo del robot PUMA. Del mismo modo en cada placa se tiene la gestión de cada uno de los servomotores correspondientes a las articulaciones de la muñeca del mismo robot (Uno por cada tarjeta).

Figura 25. Montaje final ComPID.




### 3.3.3 Convertidor de Protocolo IRDA –RS232

La interfaz de usuario propuesta corresponde a un PC remoto interconectada serialmente al sistema PC de tiempo real. Ya que la tarjeta PC usada para la solución sólo dispone de un puerto serial RS232 fue necesaria la implementación de un

convertidor IRDA serial TTL a RS232 para lo cual se desarrolló el convertidor presentado en el esquemático de la figura 26 así como su montaje final en la figura 27.

Figura 26. Esquemático convertidor Irda-RS232.

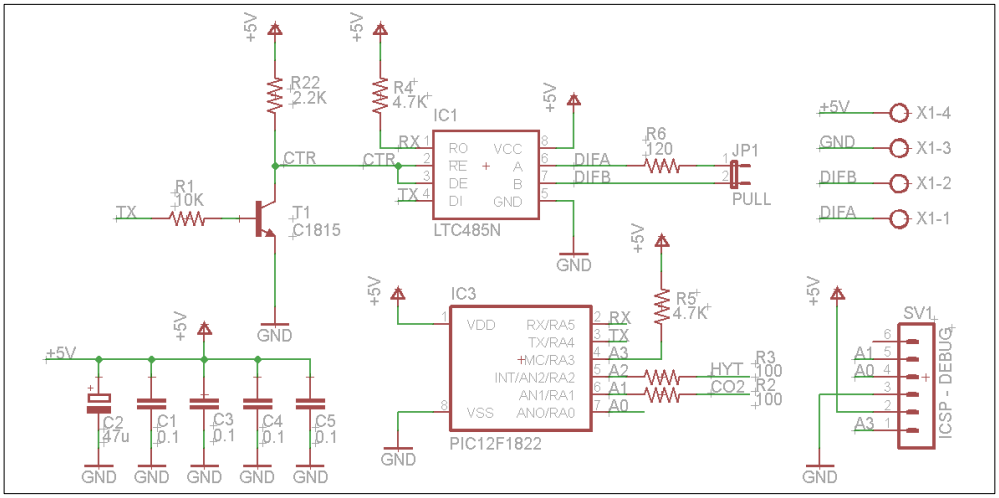
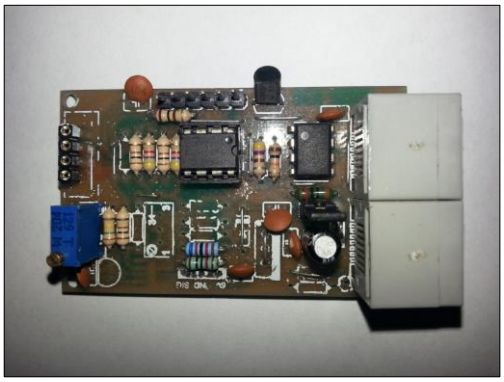



Figura 27. Convertidor IRDA-RS232 finalizado.

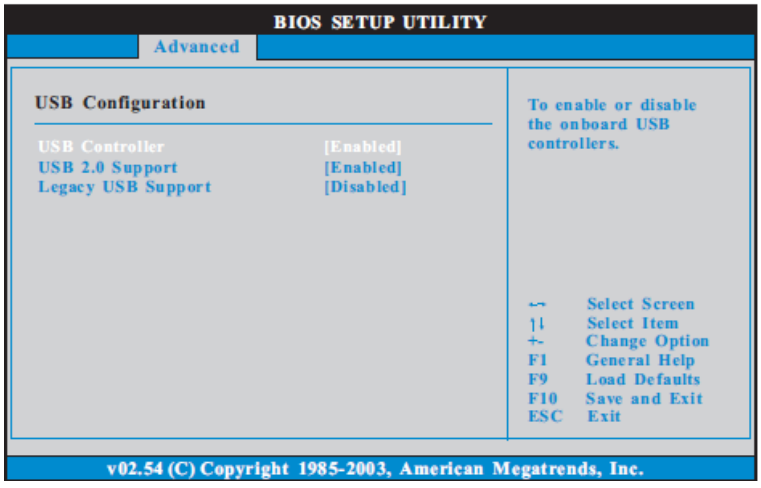


## 3.4 HERRAMIENTAS DE SOFTWARE

### 3.4.1 Configuración del BIOS

Una de las características relevantes de la operación de esta plataforma de propósito general es el arranque del sistema desde un *PENDRIVE USB*. Para lograr lo anterior se requirió que la plataforma usada permitiera el reconocimiento de dispositivos desmontables desde el sistema básico de entrada-salida (BIOS) y que el dispositivo desmontable fuera formateado para operar como una partición de arranque del sistema. Para habilitar esta característica se configuró en la utilidad SETUP dispuesta en el BIOS por el fabricante de la plataforma PC el modo heredado USB (*LEGACY*) el cual ofrece la funcionalidad de arranque tanto en el modo consola como en modo gráfico del sistema. De igual forma se indicó que el dispositivo desmontable es uno que contiene la partición y los archivos requeridos para la operación de inicio del sistema. Esta característica es mostrada en la figura 28.

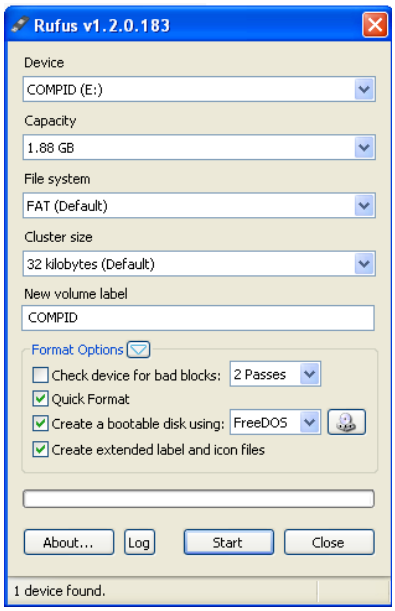
Figura 28. Configuración de dispositivo USB en el modo heredado.



### 3.4.2 Sistema operativo FreeDOS

El sistema operativo básico para la disponibilidad de compatibilidad con la arquitectura X86 utilizado fue FreeDOS **[5]**. Para formatear el *PENDRIVE-USB* con este sistema operativo y configurar el arranque del mismo se utilizó la herramienta **Rufus [20]**; que es una iniciativa *OPENSOURCE* para realizar el formato de gran parte de los dispositivos de almacenamiento existentes en la actualidad. La figura 29 muestra la presentación de la herramienta en el proceso de formateo y carga de los archivos de inicio del sistema básico.

Figura 29. Herramienta de formateo de Dispositivos USB




Como el sistema operativo FreeDOS es de 16 bits, el tipo de partición que debe ser formateada es del tipo FAT. Se debe disponer de una carpeta con una imagen del

sistema operativo para ser cargada en el medio después de realizado el formato y la partición máxima es de 2GBytes.

En la actualidad los *PENDRIVE-USB* existentes en el mercado son de mínimo 4GBytes, lo cual los hace incompatibles con la solución propuesta. Por lo anterior mediante un adaptador USB-MicroSD y una memoria MicroSD de 2 GBytes se implementó el medio de almacenamiento requerido para la implantación del sistema básico FreeDOS, como se muestra en la figura 30.

Figura 30. Adaptador USB-MicroSD

### 3.4.3 Estación de desarrollo

Como herramienta de desarrollo se utilizó un PC portátil con procesador Intel Core2 Duo, memoria RAM de 2 GBytes y sistema operativo Windows XP SP3; ya que éste sistema operativo permite la generación de archivos ejecutables con *RUNTIME* para plataformas de 16 bits **[23]**, compatibles con aplicaciones que operan en el modo real de la arquitectura x86 manejado como un hilo del sistema operativo **[24]**. Como

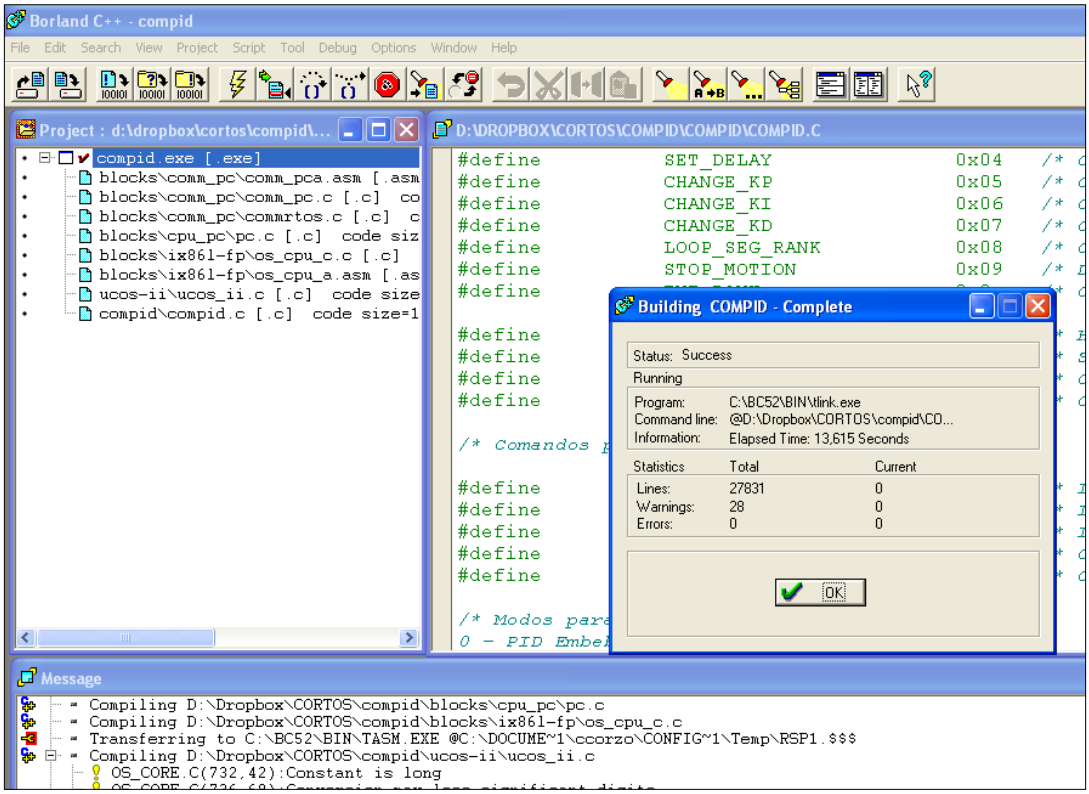
desventaja de esta plataforma se encontró que la latencia de conmutación entre tareas para la adquisición de datos por puertos serial es muy grande debido a la prioridad dada por el sistema a la interfaz gráfica y las herramientas multimedia, lo cual limitó fuertemente la función de depuración en esta plataforma. Para subsanar esta dificultad, inicialmente se utilizó la función de depuración remota disponible en la herramienta de compilación conectando un *Motherboard* independiente por puerto serial la cual resultó no operativa. Se continuó el desarrollo implementando la plataforma final y trasladando la versión del sistema cada vez que se requirió un cambio.

### 3.4.4 Herramientas de desarrollo y programación

#### 3.4.4.1 Borland C++ 4.5

Para realizar la compilación del sistema operativo de tiempo real básico con las aplicaciones específicas para el control del robot de seis grados de libertad y las interfaces de comunicaciones tanto con la red de motores como con la interfaz del usuario se escogió el entorno integrado IDE-Borland C versión 4.5, con el compilador C++ de la misma versión **[22]**. Esta herramienta produce un código ejecutable bastante reducido apto incluso para ser instalado en plataformas microcontroladas o FPGAs y es la recomendada por el desarrollador del sistema operativo básico de tiempo real. En la figura 31 se muestra el proyecto configurado para la solución de control de motores del robot PUMA, el cual integra tanto el sistema operativo de tiempo real básico como los algoritmos de los compensadores PID para los motores de las articulaciones cadera, hombro y codo; también se incluyen la gestión de los puertos RS485 con la técnica TOKEN PASS y la gestión del puerto RS232 para enviar datos que permitan graficar el comportamiento de los motores en tiempo real.

Figura 31. Proyecto ComPID sobre IDE Borland C++ 4.5

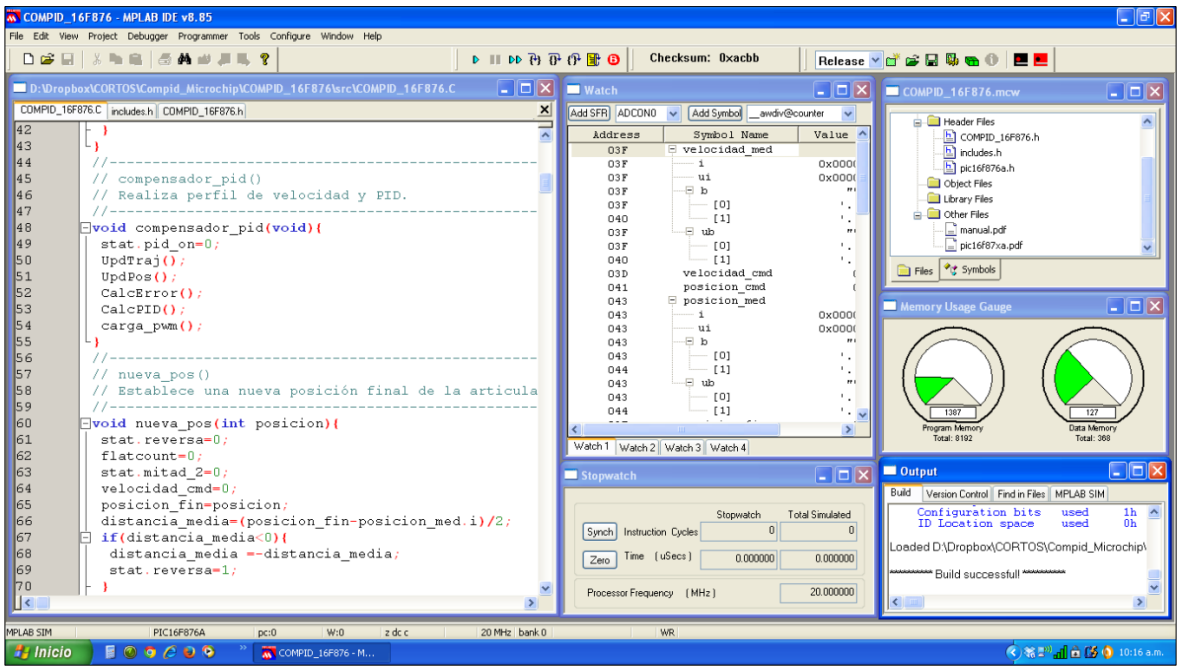



### 3.4.4.2 MPLAB 8.5

El entorno integrado de desarrollo (IDE) MPLAB es una herramienta provista por la firma MICROCHIP INC **[17]**, para el desarrollo de aplicaciones basadas en sus plataformas de microcontroladores de 8, 16 y 32 bits. Incluye todas las herramientas necesarias para esta labor como: Editor de texto con identificación de código en *assembly* o programas C, compiladores C para cada una de las plataformas mencionadas así como la posibilidad de integrar *plugins* para compiladores de otras casas fabricantes como HITECH, CCS, MIKRO-C, entre otros. De igual forma provee las herramientas para el enlazado de objetos binarios, creación de librerías y bibliotecas así como

trasladadores de código binario a las plataformas hardware más conocidas como programadores. Provee de igual forma la posibilidad de formatear proyectos completos mediante el uso de un simulador que implementa la mayoría de características de todas las plataformas hardware comercializadas por MICROCHIP. Tanto los controladores PID de los motores de las articulaciones cintura, hombro y codo como las comunicaciones de estos con el sistema operativo fueron creadas usando este entorno integrado de desarrollo bajo el nombre de ComPID, como se muestra en la figura 32.

Figura 32. ComPID sobre plataforma PIC




### 3.4.4.3 Hitech C

Para el desarrollo de aplicaciones sobre plataformas de ocho bits, Microchip recomienda el uso de un compilador ANSI-C que es totalmente compatible con el IDE

MPLAB. Por esta razón la solución hardware se creó usando este compilador que permitió simular la gran mayoría de los periféricos utilizados para la aplicación. Entre las características sobresalientes de este compilador puede resaltarse la optimización en aspectos como memoria y velocidad del código, lográndose tiempos de respuesta del orden de 50 uS. Cabe aclarar que la programación de la aplicación fue realizada en un porcentaje superior al 90% en ANSI-C lo cual lo hace bastante transportable si se quiere implementar en otra plataforma hardware **[7]**.

## 3.5 DESARROLLOS DE SOFTWARE SOBRE EL PC

Basados en el sistema operativo de tiempo real uC/OS-II el cual está disponible para ser ejecutado sobre una plataforma de arquitectura x86, se implementaron diferentes funciones y programas conducentes a gestionar las comunicaciones de una red propietaria basada en la interfaz RS-485 mediante un protocolo propio usando la técnica *token-pass,* realizar los algoritmos de compensación PID para cada uno de los motores de las articulaciones cintura, hombro y codo. De igual forma se implementó la gestión de información con la interfaz de usuario a la cual se transmiten los datos de posición y esfuerzo de control para ser graficados en una terminal gráfica remota mediante una interfaz IRDA-RS232. Usando este mismo canal de comunicaciones se reciben los comandos enviados por el usuario pertinentes constantes del compensador, velocidad, aceleración, posición deseada y la carga de hasta doce segmentos de movimiento por motor.

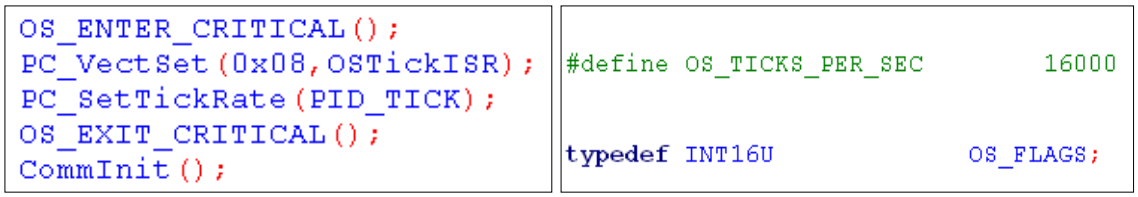
### 3.5.1 ComPID sobre PC con uC/OS-II

El software desarrollado sobre el sistema operativo uC/OS-II consta de 4 módulos que implementan la funcionalidad descrita anteriormente y que se describe en detalle a continuación previa disposición de temporizadores y vectores de interrupción.

### 3.5.2 Gestión de interrupciones desde uC/OS-II

Las interrupciones son el eje del manejo del tiempo de tarea en cualquier sistema operativo, máxime en uno de tiempo real. Por esta razón los vectores de interrupción correspondientes a los temporizadores de la plataforma x86 así como los correspondientes a los puertos seriales involucrados (COM1 y COM2) son redireccionados a rutinas de tratamiento de interrupciones propias que permitan el control total de la plataforma. Para ello se instalan las direcciones de las nuevas ISR salvando las preexistentes de modo que se pueda regresar cuando se desee al PROMPT del sistema operativo FreeDOS. De igual forma se reprograma el temporizador que provee el tiempo de tarea (TICK de tarea) asignando las constantes para tal fin, como se muestra en la figura 33 en donde se muestra que la frecuencia de TICK será de 16KHz es decir; 62,5 uS como tiempo de TICK de tarea.

Figura 33. Asignación de TICK de tarea y nuevos vectores de interrupción.



```
OS_ENTER_CRITICAL();
PC_VectSet(0x08,OSTickISR);
PC_SetTickRate(PID_TICK);
OS_EXIT_CRITICAL();
CommInit();
```

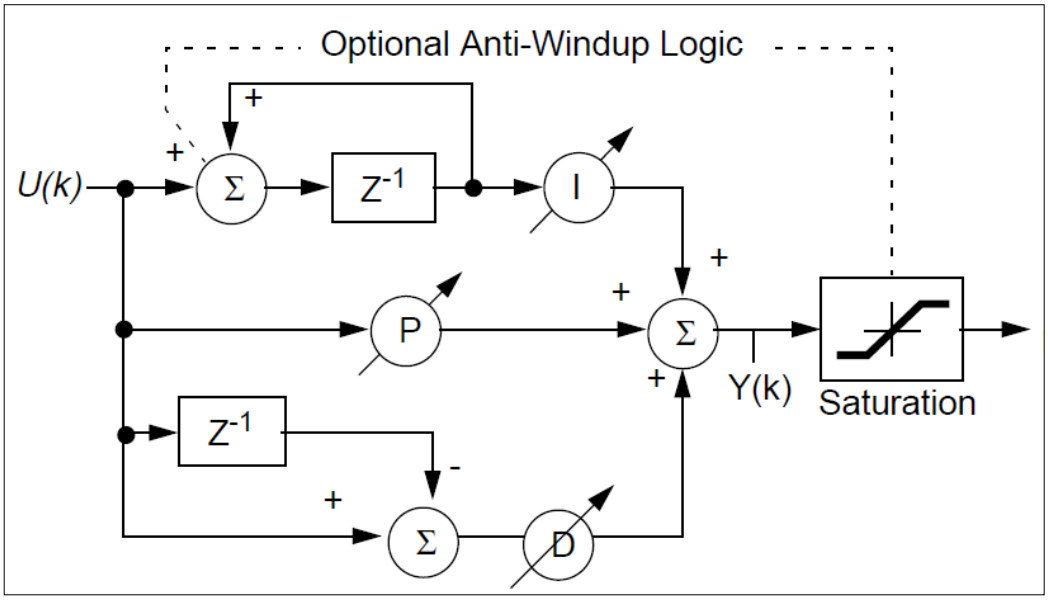
```
#define OS_TICKS_PER_SEC        16000

typedef INT16U              OS_FLAGS;
```

### 3.5.3 Controlador PID sobre uC/OS-II

El compensador PID implementado se muestra en la figura 34 donde u(k) corresponde al error de posición y Y(k) corresponde a la salida del compensador que se aplicará a la planta mediante un puente H Debe anotarse que $Z^{-1}$ corresponde a un retardo de una muestra (1 Tick) **[11]**.

Figura 34. Modelo del PID implementado




El código correspondiente a la implementación del PID en el sistema uc/OS-II se muestra en la figura 35 donde debe anotarse que la estrategia utilizada para gestionar el algoritmo para las tres articulaciones, corresponde a la creación de una estructura de datos correspondiente a todos los parámetros necesarios para el control accedidos mediante punteros a cada elemento que es requerido. De igual forma se muestra la implementación del proceso de Anti-Windup conducente a evitar la saturación del controlador integral del sistema, así como la restricción del esfuerzo de control para hacerlo compatible con el PWM disponible en cada controlador hardware.

Figura 35. Error de posición y PID

```
// Calcula el Error de Posición. //

pcm->ErrorPos=pcm->PosicionCmd-pcm->PosicionMed;

// Calcula PID.  //

pcm->PIDPos.l=pcm->ErrorPos*pcm->Kp;
if(!pcm->CtrState.saturado)pcm->ErrorInt+=pcm->ErrorPos;
if(pcm->Ki)pcm->PIDPos.l+=pcm->ErrorInt*pcm->Ki;
if(pcm->Kd)pcm->PIDPos.l +=(pcm->ErrorPos-pcm->ErrorDif)*pcm->Kd;
pcm->ErrorDif=pcm->ErrorPos;
pcm->CtrState.saturado=0;
if(pcm->PIDPos.l>500){
 pcm->PIDPos.l=500;
 pcm->CtrState.saturado=1;
}
else if(pcm->PIDPos.l<-500){
 pcm->PIDPos.l=-500;
 pcm->CtrState.saturado=1;
}
pcm->PIDPos.l+=512;
pcm->PWM_ctr.ui=pcm->PIDPos.ui[0];
return pcm->PWM_ctr.ui;
}
```

Con el fin de generar arranques y paradas suaves de cada una de las articulaciones gestionadas se implementó un perfil de aceleración trapezoidal que responde a la ecuación Vk = Vk-1+A; donde Vk corresponde a la velocidad comandada actual, Vk-1 velocidad comandada anterior y A aceleración. De igual forma la posición de la trayectoria es gestionada internamente mediante el uso de la ecuación Pk = (Pk-1)+ Vk-1 +A/2; donde Pk corresponde a la posición comandada actual y Pk-1 corresponde a la posición comandada anterior. En la figura 36 se muestra parte del código implementado **[21]**.

Figura 36. Perfil de trayectoria

```
// Actualiza Trayectoria.  //

if(pcm->CtrState.motor_on){
// if(pcm->CtrState.motor on&&!pcm->CtrState.saturado){
 if(!pcm->CtrState.mitad_2){
  if(pcm->VelocidadCmd<pcm->VelocidadLim)pcm->VelocidadCmd+=pcm->Aceleracion;
  else pcm->Flatcount++;
  pcm->DistanciaMed-=pcm->VelocidadCmd;
  if(pcm->DistanciaMed<0)pcm->CtrState.mitad_2=1;
 }
 else{
  if(pcm->Flatcount) pcm->Flatcount--;
  else{
   if(pcm->VelocidadCmd){
    pcm->VelocidadCmd-=pcm->Aceleracion;
    if(pcm->VelocidadCmd<0)pcm->VelocidadCmd=0;
   }
   else{
    pcm->PosicionCmd=pcm->PosicionFin;
    if(pcm->Dtime)pcm->Dtime--;
    else{
     pcm->CtrState.motor_on = 0;
    }
   }
  }
 }
 if(pcm->CtrState.reversa)pcm->PosicionCmd-=pcm->VelocidadCmd;
```

### 3.5.4 Comunicación serial RS-485 y TOKEN PASS sobre uC/OS-II

La técnica de paso por testigo TOKEN PASS se utiliza para sincronizar múltiples usuarios de una red alambrada única en la que participan múltiples receptores simultáneos y múltiples transmisores de los cuales sólo transmite el que le ha sido asignado el TOKEN por parte del coordinador de la red. La red RS-485 implementada es gestionada en la modalidad HALF-DUPLEX. Todos los receptores permanecen en modo recepción y el coordinador en modo transmisión. Para la gestión de la red el coordinador coloca en ella una dirección y un comando con o sin datos dependiendo del comando, después de lo cual pasará al modo recepción para esperar por un tiempo la respuesta del receptor direccionado. Todos los receptores evalúan esta dirección y al

que corresponda responderá configurándose en el modo transmisor y enviando los datos de acuerdo con la petición recibida. Un tiempo de espera es programado, al expirar éste sin recibir respuesta el coordinador cambiará su modo a transmisor nuevamente enviando una nueva dirección correspondiente a otro receptor de la red, señalizando la ocurrencia de un error. Este proceso se repetirá hasta encuestar a todos los integrantes de la red y se reiniciará con el primero de ellos nuevamente. El sistema operativo de tiempo real al ser determinístico creará ventanas de tiempo igualmente espaciadas con el fin de mantener constante el tiempo de muestreo de los receptores involucrados. Ya que los datos transmitidos en la red son binarios existe la posibilidad que en la respuesta generada por un receptor aparezca la dirección de otro receptor; haciendo que este último también responda colisionando la red. Para evitar lo anterior se implementó el manejo de paridad marca para cuando el coordinador envía direcciones (8 bits de datos paridad en alto) y paridad baja para el envío de datos (8 bits de datos un bit de paridad baja), como por ejemplo los mensajes de respuesta de un receptor direccionado. Esta configuración también se conoce como maestro esclavo. En la figura 37 se muestra parte del código correspondiente a la gestión TOKEN PASS implementada.

### 3.5.5 Comunicación serial RS-232 sobre uC/OS-II

Para gestionar el sistema de control del robot PUMA basado en el sistema operativo de tiempo real uC/OS-II por parte del usuario, se propuso realizar la interfaz mediante un puerto serial del tipo RS232 adicional que permitiera conectar remotamente un PC operando en un sistema operativo gráfico comercial como Windows 7.

Figura 37. TOKEN PASS para puerto Serial.

```
data=data;
for(i=1;i<=NUM_MAX_MOTORS;i++){
 PIDMotor[(i-1)].PWM_ctr.ui=512;
 PIDMotor[(i-1)].Kp=40;
}
for(;;){
 for(i=1;i<=NUM_MAX_MOTORS;i++){
  sprintf(trama,"%c%c%c",i,CargaComando[GET_POSITION],GET_POSITION);
  CommAdden(COMM_RS485,1);
  OSTimeDly(tdelay);
  OSMboxPost(Tx485Mbox,(void *)&trama);
  ptrama_i=(INT8U *)OSMboxPend(Ack485Mbox,0,&err);
  ptrama=ptrama_i;
  if(*ptrama != NUL){
   switch(*(ptrama+3) & 0x48){
```

Ya que en la actualidad los *Mother Boards* comerciales sólo se consiguen con un puerto serial de este tipo (en este proyecto utilizado para gestionar la red propietaria de controladores) y un puerto IRDA (Interfaz para dispositivos infrarrojos), fue necesario implementar un convertidor IRDA –RS232 para conectar el PC mencionado. La interfaz IRDA es una versión comprimida de la interfaz RS232 en cuanto a las señales eléctricas pero idéntica desde el punto de vista del protocolo, por esta razón para el BIOS y para el sistema operativo, la configuración y uso del puerto IRDA se realiza pensando en un puerto RS232. Por lo anterior se configuró en el BIOS del sistema la interfaz IRDA como puerto serial COM2 y se implementaron las tareas correspondientes a la trasmisión de datos (error de posición, posición medida, posición comandada, y posición deseada de los tres motores que se gestionaron mediante compensadores PID, como datos de salida. Para el caso de la recepción se implementó la tarea correspondiente con el fin de recibir los datos entregados por el usuario en la terminal remota correspondientes a comandos y datos para modificar las constantes de cada PID, velocidad máxima,

aceleración y segmentos a ejecutar. Esta información puede encontrarse en el anexo 1. En la figura 38 se presenta parte del código implementado para la gestión serial con el PC remoto.

Figura 38. Recepción y transmisión serial con RS232

```
void PIDComtx232(void *data){
 INT8U err,l;
 INT8U *ptrama;

 data = data;
 CommClearFIFOs(COMM_RS232);
 for(;;){
  ptrama=(INT8U *)OSMboxPend(Tx232Mbox,0,&err);
  l=(*ptrama-1);
  ptrama++;
  do{
   CommPutChar(COMM_RS232,*ptrama++,0);
   l--;
  }while(l && err==COMM_NO_ERR);
  while(!CommTxComplete(COMM_RS232));
  OSTimeDly(tdelay);
 }
}
```

### 3.5.6 Gestión de interfaz de usuario sobre uC/OS-II

La interfaz de usuario está orientada en esta aplicación a la transmisión de los datos de posición, velocidad y control de los motores que operan las articulaciones de la cintura, el hombro y el codo del robot PUMA. Estos datos se reciben de cada uno de los controladores en formato binario, para aprovechar al máximo el ancho de banda del canal de comunicaciones. Otros datos son generados por el compensador PID en el sistema operativo de tiempo real con la velocidad requerida para mantener el tiempo de muestreo de los motores; constante. Con el fin de monitorear el comportamiento del compensador por ejemplo para fines de sintonización, los datos deben ser transmitidos

al PC remoto para ser graficados y así realizar los ajustes correspondientes a las constantes del compensador. Para mantener la compatibilidad de estos datos con varias herramientas, los valores correspondientes a las variables enunciadas son convertidos a cadenas de caracteres separados por espacio para ser transmitidos por el puerto serial donde se conecta el PC, lo que incrementa el volumen de datos por unidad de tiempo ya que las cadenas generadas ocupan mayor número de bytes; esto representa un inconveniente que fue resuelto implementando *buffers* circulares de tamaño apropiado para garantizar la integridad de todas las tramas correspondientes a cada muestra recibida de y enviada a cada controlador PID. En la figura 39 se muestra el código que realiza esta función.

Figura 39. Formateo de datos en ASCII para PC remoto.

```
switch(*(ptrama+3) & 0x48){
 case 0x00: //Local(PIC): Reenvia trama en ASCII al PC para RealTerm y KST2. 57 caracteres, máximo (5 mS).
  l.ui=sprintf(cadena1,"%1d %4d %1d %1d %4d %4d %4d %4d %4d %4d %4d %4d %4d\r\n",
  *(ptrama),*(ptrama+1),*(ptrama+2),*(ptrama+3),(*(ptrama+4)*256+*(ptrama+5)),(*(ptrama+6)*256+*(ptrama+7)),
  (*(ptrama+8)*256+*(ptrama+9)),(*(ptrama+10)*256+*(ptrama+11)),(*(ptrama+12)*256+*(ptrama+13)),
  (*(ptrama+14)*256+*(ptrama+15)),(*(ptrama+16)*256+*(ptrama+17)),(*(ptrama+18)*256+*(ptrama+19)),
  (*(ptrama+20)*256+*(ptrama+21)));
  OSMboxPost(Tx232Mbox,ptrama);
  OSSemPend(Rx232Sem,0,&err);
 break;
 case 0x08: //Remoto(ComPID): Envia estado y posición medida (HEX).
  ptr_str=&PIDMotor[(i-1)];
  PIDControl(ptr_str,ptrama);
  sprintf(trama,"%c%c%c%c%c",i,CargaComando[LOAD_PWM],LOAD_PWM,
  PIDMotor[(i-1)].PWM_ctr.ub[0],PIDMotor[(i-1)].PWM_ctr.ub[1]);

  if(i==1){
   sprintf(cadena1,"%c",0);
  }
  sprintf(cadena2," %d",(*(ptrama+4)*256+*(ptrama+5)));
  strcat(cadena1,cadena2);
  if(i==NUM_MAX_MOTORS){
   strcat(cadena1,"\r\n");
   l.ui=strlen(cadena1);
   cadena1[0]=l.ub[0];
   OSMboxPost(Tx232Mbox,(void *)&cadena1);
  }
```

## 3.6 DESARROLLO DE SOFTWARE SOBRE PIC16F876

La implementación de los controladores locales basados en la plataforma PIC16F876 se realizó usando el lenguaje de programación C mayoritariamente con funciones cubiertas por el estándar ANSI, evitando recurrir a las funciones implementadas por el compilador C de HITECH que fue la herramienta de compilación utilizada. La solución consta de dos secciones de código las cuales giran en torno a la adquisición de la posición de cada articulación mediante el procesado análogo-digital así como la gestión del posicionamiento de un motor DC mediante la generación de una señal digital-análoga basada en la variación del ancho de pulso de un canal PWM gestionado por el compensador PID en el sistema operativo de tiempo real. De otra parte la gestión de comunicaciones mediante puerto serial operado por la USART disponible en la plataforma. Estos dos módulos son presentados a continuación. La solución microcontrolador tiene implementados cuatro modos de operación:

1. Operación LOCAL sin transmisión de datos para depuración.
2. Operación LOCAL con transmisión ASCII de datos para depuración.
3. Operación REMOTA con transmisión de posición medida y recepción de PWM en formato binario.
4. Operación REMOTA con transmisión de posición medida, comandada y deseada, error de posición, velocidad medida, comandada y deseada, error de posición, velocidad máxima, aceleración, Ki, Kp y Kd. Los valores mencionados son transmitidos en binario.

Cabe anotar que en el modo LOCAL no se utiliza un sistema operativo adicional de modo que la operación es autónoma. En el modo REMOTO se requiere el uso de la gestión externa por parte del sistema operativo de tiempo real tanto para las comunicaciones como para la adquisición.

En ambos casos se requiere un mecanismo de gestión y sincronización de la red RS485. Seguidamente se describe la implementación de los dos módulos mencionados.

### 3.6.1 Controlador PID sobre microcontrolador PIC

Como se mencionó anteriormente el modo LOCAL realiza y gestiona autónomamente la posición de la articulación a su cargo, recibiendo los requerimientos de desplazamiento y constantes de tiempo del compensador PID. En el modo remoto el sistema operativo de tiempo real externo solicita en cada periodo de muestreo, la posición actual de la articulación, la cual es gestionada mediante el convertidor análogo digital del microcontrolador. Esta condición es tratada por el sistema operativo del microcontrolador el cual es del tipo BACKGROUND – FOREGROUND sin manejo de preferencia por rango ni gestión de inversión de prioridad. La gestión de los periféricos es tratada a nivel de interrupciones. El código correspondiente a la adquisición A/D se muestra en la figura 40.

Figura 40. Adquisición Análogo-Digital

```
void int_ad_10(void){
  PIR1bits.ADIF=0;
  posicion_med.ub[1]=ADRESH;posicion_med.ub[0]=ADRESL;
  if(!bitflags.brake_mot&&!bitflags.modo_op)stat.pid_on=1;
  stat.pid_on=1;
}
```

La implementación del compensador PID en la plataforma PIC requirió el desarrollo de funciones para el cálculo del error de posición, generación de trayectoria y compensador PID, como se indica en la figura 41.

Figura 41. Funciones del compensador PID

```
//-----------------------------------------------
// CalcError()
// Calcula el error de posición
//-----------------------------------------------
void CalcError(void){
 error_pos=posicion_cmd-posicion_med.i;
}
//-----------------------------------------------
// CalcPID()
// Ejecuta el compensador PID y establece el nu
//-----------------------------------------------
void CalcPID(void){
 pid_pos.l=error_pos*kp;
 if(!stat.saturado)error_int += error_pos;
 if(ki)pid_pos.l += error_int*ki;
 if(kd)pid_pos.l += (error_pos-error_dif)*kd;
 error_dif=error_pos;
 stat.saturado = 0;
 if(pid_pos.l > 500){
  pid_pos.l = 500;
  stat.saturado = 1;
 }
 else if(pid_pos.l < -500){
  pid_pos.l = -500;
  stat.saturado = 1;
 }
 pid_pos.l += 512;
 PWM_MOT.i=pid_pos.i[0];
}
```

### 3.6.2 Comunicación serial RS485 sobre microcontrolador PIC

El sistema PIC se configura en el arranque en el modo de recepción con la identificación de dirección mediante el mecanismo de bit de paridad marca, así como los parámetros de comunicación: velocidad 115200 bps, 8 bits de datos, 1 bit de paridad y un bit de parada sin control de flujo. Al recibir una dirección a través de la interfaz RS485 y después de validar que corresponde a la propia, se cambia la paridad a modo espacio para de esta forma recibir el resto de datos que conforman un mensaje (El protocolo es

descrito en el anexo 1). Cuando se completa el mensaje y ya que USART del dispositivo es gestionado por interrupciones, se procede a habilitar el envío de los datos según el modo de operación configurado, los cuales residen en un buffer de transmisión en el cual la rutinas de adquisición y compensador PID cargan los datos de posición, velocidad y demás en cada periodo de muestreo del lazo de control. Este tratamiento es mostrado en la figura 42.

Figura 42. Interrupción de la recepción serial

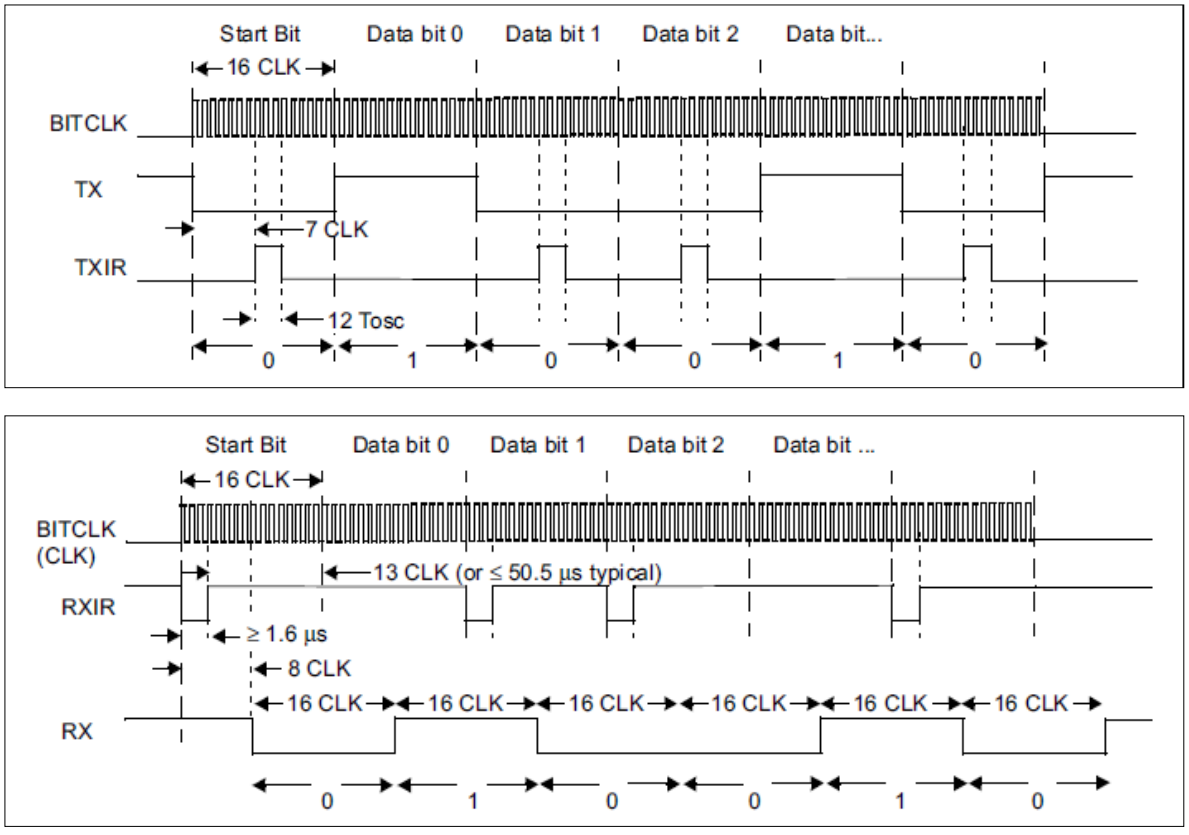
```
//-------------------------------------------------------------
// int_rxUSART()
// Gestiona la recepción de caracteres desde el servidor.
//-------------------------------------------------------------
void int_rxUSART (void){

  if(RCSTAbits.FERR || RCSTAbits.OERR) error_rs232();
 else{
   rx_byte=RCREG;
   TMR0=0;
   INTCONbits.TMR0IF=0;
   if(RCSTAbits.ADDEN){
   if((rx_byte==ID1)||(rx_byte==ID2)||(rx_byte==ID3)){
//   if(rx_byte==ID_MOTOR){
     RCSTAbits.ADDEN=0;
     serial.id_motor=1;
     ptr_rx=&peri_ID;
     bytes_rx=3;
    }
    else error_rs232();
   }
   if(serial.id_motor){
    if((ptr_rx==&rx_long))bytes_rx=rx_byte-1;
    *ptr_rx=rx_byte;
    ptr_rx++;
    bytes_rx--;
    if(!bytes_rx){
     serial.id_motor=0;
     RCSTAbits.ADDEN=1;
     switch(rx_cmd){
      case GET_POSITION:
```

### 3.6.3 Convertidor IRDA RS232 sobre PIC12F88

Al requerirse un puerto serial RS232 adicional al disponible en la mayoría de plataformas PC existentes en la actualidad en el mercado, este fue implementado basándose en la existencia de un puerto IRDA e implementando un convertidor IRDA-RS232 mediante el uso de una plataforma PIC12F88 de 8 bits de bajo costo. La implementación busca realizar las conversiones de señal mostradas en la figura 43 en la que aparecen las señales tanto de recepción y transmisión para la interfaz IRDA como para la interfaz RS232 **[14]**.

Figura 43. Señales IRDA-RS232 para Tx y Rx



Para realizar la conversión propuesta se hizo uso de la generación de interrupciones por cambio de señal en un puerto (Tx-IRDA a Tx-RS232) y la misma estrategia para la conversión de la recepción (Rx-RS232 a Rx-IRDA). El código correspondiente a estas funciones es mostrado en la figura 44.

Figura 44. Código conversión IRDA – RS232.

```
void int_ext(void){
  INTCONbits.INTF=0;
  if(!status.rx_start_irda){
    TEST_PIN=0;
    TMR0=RX_START_IRDA;
    TMR0IF=0;
    status.rx_start_irda=1;
    conta_bits_rx=RX_NUM_BITS;
  }
  status.rx_irda_0=1;
}
void int_rax(void){
  INTCONbits.IOCIF=0;
  if(IOCAFbits.IOCAF5){
    IOCAFbits.IOCAF5=0;
    if(!status.tx_start_irda){
    TMR0=TX_BIT_IRDA;
    TMR0IF=0;
    TX_PIN_IRDA=0;
    NOP();
    NOP();
    TX_PIN_IRDA=1;
//    if(!status.tx_start_irda){
     status.tx_start_irda=1;
     conta_bits_tx=(TX_NUM_BITS);
//    }
    status.tx_irda_0=1;
    conta_bits_tx--;
    }
  }
}
```

# 4. INTERFAZ DE USUARIO

## 4.1 GESTIÓN DE DATOS SERIAL CON DOCKLIGHT

Los datos generados tanto por la plataforma PIC como por la plataforma PC con respecto a la compensación PID de los motores correspondientes a las articulaciones cintura, hombro y codo son gestionados en formato binario en la red de controladores para mantener los tiempos de respuesta acordes a la frecuencia de muestreo de los motores controlados. Para el caso del monitoreo de esos datos y de la gestión de los controladores por parte del usuario, la aplicación desarrollada sobre el sistema operativo de tiempo real uC/OS-II formatea estos en formato ASCII a fin de poder manipularlos con cualquier programa que tenga capacidad de operar sobre texto. En la industria del software existen varias herramientas para la adquisición, monitoreo y gestión de los datos recibidos y transmitidos mediante las interfaces RS232 existente en los PCs. Se utilizó el programa DOCKLIGHT **[19]**, para cumplir con el cometido mencionado. Se presenta un resumen de las características sobresalientes de esta herramienta así como ejemplos de la captura y gestión de datos para el monitoreo del sistema desarrollado.

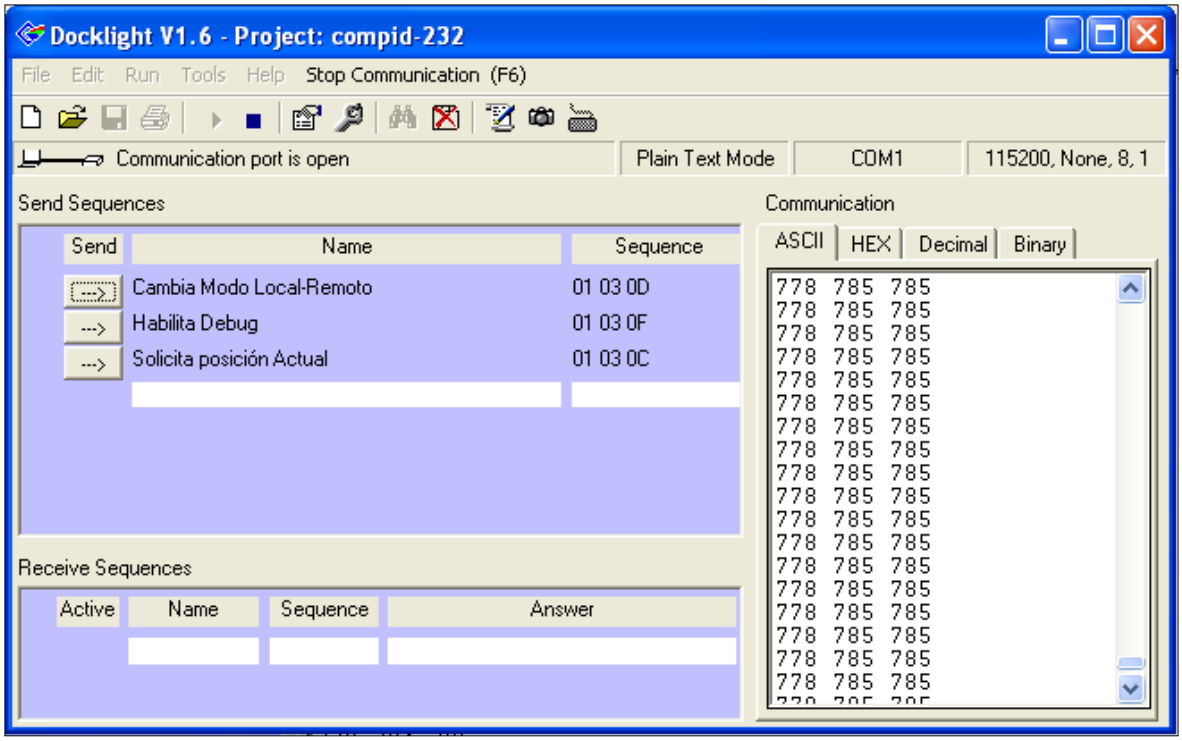
### 4.1.1 Características del programa DOCKLIGHT

Este programa es una herramienta de verificación, análisis y simulación para protocolos de comunicación serial RS232, RS485 y otros, que permite verificar la comunicación entre dos dispositivos seriales o probar las comunicaciones de un dispositivo único. Entre las características relevantes se mencionan:

1. Los datos recibidos a través de un puerto serial RS232 pueden ser grabados en un archivo de texto a alta velocidad.
2. Permite el envío de secuencias previamente establecidas por el usuario, como comandos asíncronos o como respuesta automáticas a secuencias recibidas.
3. Visualiza los datos recibidos en formatos ASCII, Hexadecimal y Decimal.
4. Habilita la manipulación manual de los pines de control de flujo por parte del usuario.

Esta herramienta fue usada como programa de captura de datos para ser presentados en un programa graficador genérico y como consola de gestión para el envío de comandos al sistema operativo de tiempo real. En la figura 45 se presenta un ejemplo de captura de datos y configuración de comandos.

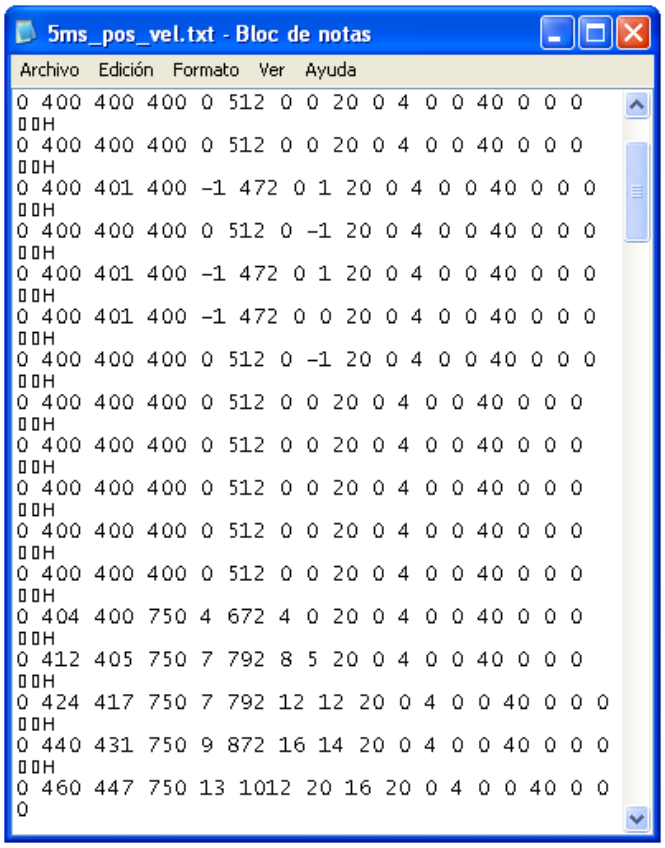
Figura 45. Captura de datos y comandos sobre RS232.



## 4.2 VISUALIZACIÓN DE DATOS MEDIANTE KST2

KST2 es una herramienta de visualización y generación de gráficos *OPENSOURCE* que permite capturar datos en tiempo real desde un archivo de texto con formato genérico, generados desde Matlab o con formatos personalizados, gracias a su estrategia de componentes embebidos y a la posibilidad de programación mediante macros. Con KST2 **[12],** se pueden visualizar los datos capturados en un archivo plano recibidos a través de un puerto serial. Un ejemplo del archivo de datos usado como insumo en la generación de gráficas se muestra en la figura 46.

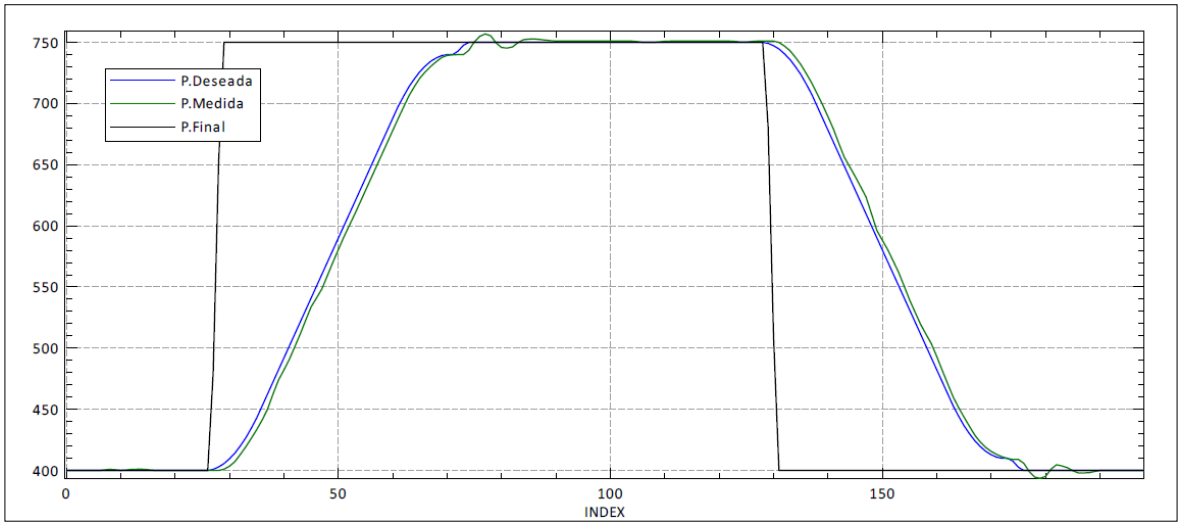
Figura 46. Archivo de texto con datos capturados desde RS232.



### 4.2.1 Presentación de datos en tiempo real con KST2

Los datos capturados en un archivo de texto como el mostrado en la figura 46 son graficados mediante la herramienta KST2 en tiempo real como se muestra en la figura 47.

Figura 47. Gráfica de posición de un motor en KST2



# 5. RESULTADOS

Luego de la implementación del hardware y software descrito, así como el uso de las diferentes herramientas; tanto en el nivel del sistema operativo de tiempo real como en el de usuario del sistema se presentan los resultados obtenidos en referencia a la puesta en marcha y validación de los objetivos propuestos.

## 5.1 Arranque USB

El uso de arranque desde un dispositivo USB se realizó satisfactoriamente obteniéndose 100% de éxito en las pruebas realizadas para la comprobación de esa actividad que involucró 120 arranques desde apagado.

Al instalar el sistema operativo de tiempo real en la USB y ejecutar pruebas con el mismo usando los recursos disponibles en uC/OS-II sin utilizar puertos seriales RS-232 o Irda, el resultado fue igual al anterior.

Caso contrario sucedió cuando se implementó el uso de los puertos descritos ya que el porcentaje de éxito en el arranque se redujo al 25% (30 arranque exitosos) por el mismo número de 120. Lo anterior es atribuible a que se presume una UART del tipo 16550 instalada en la *mother board* pero realmente se tiene un chip ASIC que emula el comportamiento de este sistema, que opera con frecuencias de trabajo diferentes (Hasta 48 Mhz, según hoja de datos).

## 5.2 Interfaz RS-232

Con el fin de establecer la conexión del controlador de los motores con el PC y debido a las pequeñas constantes de tiempo (Frecuencia de muestreo) no fue posible obtener resultados satisfactorios mediante el uso de un solo controlador para los 3 motores y los 3 servomotores disponibles en el robot PUMA. Por lo anterior se decidió implementar un controlador por cada Motor/Servomotor e interconectarlos por una red serial tipo RS-485 *half dúplex*, gestionada por el sistema operativo de tiempo real.

Lo anterior requirió la implementación de un protocolo propietario con un mecanismo de sincronización, tal que permitiera evitar la colisión de datos debida a la respuesta simultánea de dos o más controladores. Se usó la transmisión del noveno bit (Bit de paridad) a modo de señalización de paquetes correspondientes a direcciones (Paridad Marca) o datos (Parida Espacio).

Ya que los *mother board* usados sólo disponían de un solo puerto serial RS-232 y un puerto infrarrojo IrDA, se hizo necesaria la implementación de un convertidor de señal IrDA-RS485 y viceversa, la cual operó satisfactoriamente en la labor de conversión así como en el tiempo de respuesta funcionando a la velocidad máxima de 115.200 bps.

## 5.3 Controlador de motor embebido

La implementación de este sistema requirió la reparación completa del robot PUMA en sus tres articulaciones principales: Cintura, hombro y codo. Se implementaron plantillas de sujeción de los potenciómetros lineales de realimentación de la posición y se cambiaron los 3 potenciómetros. Los instalados fueron de 2.5K, con desplazamiento máximo de 340 grados. (Cabe anotar que las articulaciones del robot, tienen una restricción mecánica de 220 grados cada una).

La funcionalidad implementada en este sistema provee los siguientes comandos y características:

1. Cambiar la Posición.
2. Cambiar la velocidad máxima.
3. Cambiar la Aceleración.
4. Cambiar las constantes Kp, Ki y Kd.
5. Cambiar la frecuencia de muestreo.

Estos cambios pueden realizarse de manera independiente o de forma grupal (Para todos los controladores conectados a la red, los cuales pueden ser hasta 32).

6. El mantenimiento de cada articulación en una posición dada después de realizado un movimiento se logra mediante la técnica de frenado dinámico ya que los PWM implementados son bipolares. Los valores más pequeños de PWM útiles fueron de 18% debido al tiempo muerto de los motores en conjunto con las reducciones mecánicas existentes en cada articulación.
7. La base tiempo más pequeña corresponde a 512 uS y todos los tiempos expresados se realizan en estas unidades. El sistema se probó con un periodo de muestreo de 1,5 ms con resultados satisfactorios sobre los 3 motores de las articulaciones principales.
8. Se implementó un modo de depuración en el que se transmiten todas las variables de interés para la depuración y sintonización del sistema. Por el volumen de datos transmitidos fue necesario disminuir la rata de barrido de la red de controladores a fin de evitar saturación de los buffers de recepción del sistema operativo de tiempo real.
9. El sistema involucra un perfil de aceleración para la realización de cada movimiento de la articulación, con lo cual se pudo evidenciar desplazamientos suaves de esta.

10. No se realizó sintonización de los controladores, pero se ajustó en cada uno de ellos Kp, con buenos resultados en estabilidad, repetitividad y controlabilidad.

## 5.4 Algoritmo compensador PID

El algoritmo compensador incluye la gestión y control de saturación tanto del controlador integral como del PWM. El código se escribió en su totalidad en lenguaje C-ANSI sin recurrir a las librerías del compilador, facilitando la migración del código hacia otras plataformas de hardware, optimizado para espacio (*footprint* de menos de 2 KBytes con uso de RAM inferior a 0,25 KBytes.). Incluye perfil de aceleración trapezoidal de trayectoria para desplazamientos largos y perfil triangular para desplazamientos cortos.

## 5.5 Aplicativo sobre sistema operativo de tiempo real

Se usaron los servicios ofrecidos por el sistema operativo de tiempo real uC/OS-II para implementar la aplicación de control de comunicaciones de la red de controladores y las comunicaciones con el PC del usuario. De igual forma se usaron para implementar un controlador PID para cada motor como mecanismo de validación en la generación de tiempos determinísticos referentes al tiempo de muestreo. Entre las características a resaltar se mencionan:

1. Funciones en Assembly para la gestión de las interrupciones generadas por las UART correspondientes a los puertos seriales.
2. Buffers Circulares de tamaño variable para el almacenamiento de datos recibidos y a transmitir por cada puerto serial.
3. Transmisión de datos entre tareas mediante *Mailboxs* y colas de datos.
4. Sincronización de tareas de la red de controladores mediante semáforos con tiempo de expiración programable.

5. Configuración de parámetros desde la línea de comandos a fin de modificar características como: tick del sistema operativo el cual se probó desde 62,5 uS con regulares resultados; hasta 80 mS con excelentes resultados a partir de 3,5 mS.
6. Gestión desde terminal del usuario de las características descritas anteriormente para el control de los motores, en tiempo real mediante el uso de las aplicaciones REALTERM y DOCKLIGTH. Así como para el monitoreo del sistema con opción de gráficas mediante el programa KST2.

# 6. CONCLUSIONES Y TRABAJO FUTURO

La posibilidad de portar un sistema operativo de tiempo real y aplicativos sobre éste en USB; provee gran versatilidad cuando se pretende usar estas herramientas como plataforma didáctica. De igual forma como plataforma para implementación comercial ofrece gran rentabilidad frente a sistemas de desarrollo especializadas, por cuanto permite el uso del PC para aplicaciones de control en tiempo real.

A pesar de la disponibilidad de procesadores con alta velocidad de procesamiento el ancho de banda para la transmisión/recepción en los puertos seriales; es restringido debido a los tiempos de retardo introducidos por las UART y por las limitaciones propias de la misma interfaz.

El uso de un sistema operativo de tiempo real en aplicaciones de control es viable pero demanda un estudio previo cuidadoso en cuanto al manejo de prioridades y escogencia de servicios para la gestión de datos, ya que de no realizarse este proceso es bastante difícil lograr las metas de tiempo de tarea y la sincronización de las mismas.

La interfaz RS-485 implementada para la intercomunicación de los controladores con el PC que gestiona la red, ofrece mayor robustez frente a las perturbaciones electromagnéticas encontradas en los sectores de aplicación de la robótica como lo es el industrial por ser una interfaz de tipo diferencial. Lo anterior aunado a la estrategia *Token Pass* con identificación de direcciones garantiza la eliminación de colisión de datos en la red.

## 6.1 Trabajo Futuro

Se recomienda implementar la interconexión de los controladores de los motores usando una interfaz Ethernet que ofrece anchos de banda entre 10 MBit/S, y 1GBit/S comparado con los 115.200 bps de la interfaz. El fabricante del sistema operativo uC/OS-II ofrece un *Stack* de servicios completo para la gestión de datos en esta intefaz y los *board* PC comerciales por lo general disponen del hardware necesario para este fin.

En las plataformas PC se dispone de gran cantidad de memoria RAM instalada a bajo costo, lo que permite implementaciones de aplicaciones de gran tamaño gestionadas en tiempo de ejecución por los servicios del sistema uC/OS-II de forma dinámica. Se sugiere aprovechar esta condición para implementar sistemas de emulación y depuración remota en tiempo real.

Para el caso de la gestión de Robots se recomienda implementar la gestión de trayectorias en el PC con el sistema operativo de tiempo real y delegar el control de posición/Velocidad al hardware del controlador con el compensador embebido.

De continuarse con la conexión serial y como consecuencia de la aparición de plataformas microcontrolador con altas capacidades de velocidad y memoria, se recomienda centralizar en una sola plataforma microcontrolador el control de las seis articulaciones del robot con lo cual la red propietaria podría escalarse a la gestión y control de varios manipuladores.

Migrar el sistema operativo de tiempo real a una alternativa que use la arquitectura x86 en el modo protegido, para sacar provecho del ancho del bus de datos de 64 bits disponible en los *mother boards* actualmente. Del mismo modo explotar la posibilidad de asignar tareas específicas a procesadores únicos en plataformas multi-procesador como Intel Core i7.

Para acceder a las herramientas de desarrollo y depuración de última generación para sistemas operativos de tiempo real, se sugieren como alternativa las implementaciones sobre el sistema operativo Linux el cual saca provecho de la arquitectura en toda su extensión. Para el caso de aplicaciones de tiempo real se recomienda usar RTAI; que es un sistema operativo de tiempo real que puede operar con una distribución Linux convencional como tarea de menor prioridad pero con acceso a todos los recursos gráficos, de bases de datos, controladores de hardware, multimedia y conectividad, disponibles actualmente en versiones como Ubuntu, Fedora, RedHat, etc.

## BIBLIOGRAFÍA


[1] Asrock P4i65G. User Manual Versión 1.3. November 2006.

[2] B. B. Shao, *Embedded Real-Time Operating System. UCOS-II*. Beijing: Beijing University of Aeronautics and Astronautics Press, 2003, pp. 60-100.

[3] Brey b. Barry. Los microprocesadores Intel. Arquitectura, Programación e Interfaces. Editorial Prentice Hall ISBN 968-880-481-9. México 1994.

[4] Christopher L. Morgan, Mitchell Waite. "Introducción al microprocesador 8086/8088". Editorial Mc Graw Hill, 1984.

[5] FreeDos Project. Disponible en http://www.freedos.org/. Revisado en Julio 14 de 2014.

[6] Gorky Alfonso; Pérez Juan G;Suárez Marco F. PROTOTIPO DE UN ROBOT TIPO PUMA DE SEIS GRADOS DE LIBERTAD. Junio 2005. Trabajo de Grado. Universidad Industrial de Santander.

[7] HI-TECH C compiler for PIC10/12/16 MCUs (PRO). Disponible en http://www.microchip.com/Developmenttools/ProductDetails.aspx?PartNO=SW500010. Consultado en Julio 14 de 2014.

[8] IBM.Technical Reference Personal Computer AT. Boca Ratón. 1984.

[9] Intel Pentium 4 Processor Optimization Reference Manual Revisado 14/07/14 Disponible http://developer.intel.com/design/pentium4/manuals/248966.htm.

[10] Intel® 64 and IA-32 Architectures. Optimization Reference Manual. May 2017.

[11] John G. Proakis, Dimitris G. Manolakis. "Tratamiento digital de señales". Editorial Prentice may, 1997.

[12] KST2 Visualize your Data. Disponible en http://kst-plot.kde.org/. Consultado en Julio 14 de 2014.

[13] Labrosse Jean j. MicroC/OS-II. The Real-Time Kernel. CMPBooks. Second Edition. ISBN: 1-57820-103-9.Berkeley 2002.

[14] Microchip MCP2120 Infrared Encoder/Decoder DataSheet. DS21618A. 2001.

[15] Microchip Technology Inc. PIC12F/LF1822/PIC16F/LF1823 Data Sheet. DS41413B. 2010.

[16] Microchip Technology Inc. PIC1687XA Data Sheet. DS39582B. 2003.

[17] MPLAB. Integrated Development Environment (IDE). Disponible en: http://www.microchip.com/pagehandler/en-us/family/mplabx/. Consultado en Julio 14 de 2014.

[18] Norton Peter. GUIA DEL PROGRAMADOR PARA EL IBM PC. Red Editorial Iberoamericana S.A. Bogotá. 1987.

[19] RS232 Terminal / RS232 Monitor - Version 2.0. Disponible en: http://www.docklight.de/download_en.htm. Consultado en Julio 14 de 2014.

[20] RUFUS. Cree Unidades USB Arrancables Fácilmente. Disponible en http://rufus.akeo.ie/. Consultado en Julio 14 de 2014.

[21] Stephen Bowling. AN696- PIC18CXXX/PIC16CXXX DC Servomotor Aplication. DS00696A. 2000.

[22] Schildt Herbert. "Borland C ++". Manual de Referencia. ISBN: 84-481-1141-9. Editorial Mc Graw Hill, 1997.

[23] Tanenbaum Andrew S. SISTEMAS OPERATIVOS. DISEÑO E IMPLEMENTACIÓN. Editorial Prentice Hall.ISBN: 0-13-637406-9. México 1998.

[24] Tanenbaum Andrew S. Sistemas Operativos Distribuídos. Editorial Prentice Hall.ISBN: 0-13-219908-4. México 1996.

[25] Villani Pat. FreeDOS Kernel. An MS-DOS Emulator for Platform Independence & Embedded Systems Development. R&D Books. Emeryville – California. 1996.

# ANEXOS

## ANEXO A. MANUAL DEL PROTOCOLO DE COMUNICACIONES.

Parámetros de la Comunicación para COMPID

Velocidad: 115200 baudios/seg.

Bits: 8

Paridad: 1 para dirección, 0 para datos

Bits de parada: 1

Formato de trama ID LNG CMD FLAG* LSB MSB

ID Identificador de dispositivo conectado.

LNG Número de datos de la trama.

CMD Comando ordenado.

FLAG* Banderas del controlador. (Sólo con el comando de requerir posición del motor -0x0C-)

LSB Byte de menor peso del dato.

MSB Byte de mayor peso del dato.

## IDENTIFICADORES DE DISPOSITIVOS CONECTADOS A LA RED

GLOBAL_ID 0x00 Todos los dispositivos conectados

COMPID_ID 0x0e CPU COMPID_RTOS

PC_ID 0x0f PC de interfaz del usuario

X_AXIS_ID 0x01 Controlador eje X

Y_AXIS_ID 0x02 Controlador eje Y

Z_AXIS_ID 0x03 Controlador eje Z

COMANDOS PARA MANEJO DE PERFIL DE MOVIMIENTO DEL SERVOMOTOR

SET_DISTANCE 0x01 Carga/recibe la distancia relativa para el segmento de perfil

Ejemplo Cmd: 12 01 3E A4 Distancia a recorrer es 0xA43E

Rta: No tiene.

SET_VELOCITY 0x02 Carga/recibe la velocidad límite del segmento

Cmd: 12 02 3E A4 Velocidad máxima es 0xA43E

Rta: No tiene.

SET_ACCELERATION 0x03 Carga/recibe la aceleración del segmento

Cmd: 12 03 3E A4 Aceleración 0xA43E

Rta: No tiene.

SET_DELAY 0x04 Carga/recibe el retardo entre segmentos

Cmd: 12 04 3E A4 Retardo entre segmentos es 0xA43E

Rta: No tiene.

CHANGE_KP 0x05 Carga/recibe KP del PID para el segmento

Cmd: 12 05 3E A4 KP para ese perfil es 0xA43E

Rta: No tiene.

CHANGE_KI 0x06 Carga/recibe KI del PID para el segmento

Cmd: 12 06 3E A4 KI para ese perfil es 0xA43E

Rta: No tiene.

CHANGE_KD 0x07 Carga/recibe KD del PID para el segmento

Cmd: 12 07 3E A4 KD para ese perfil es 0xA43E

Rta: No tiene.

LOOP_SEG_RANK 0x08 Carga/recibe rango de segmentos para ejecutarse cíclicamente.

Cmd: 12 08 02 05 Primer segmento es 0x02, último segmento es 0x05.

Rta: No tiene.

STOP_MOTION 0x09 Detiene la ejecución del perfil

Cmd: 10 09 Sin datos

Rta: No tiene.

EXE_RANK 0x0a Carga/recibe rango de segmentos a ejecutarse una vez

Cmd: 12 0a 02 05 Primer segmento es 0x02, último segmento es 0x05.

Rta: No tiene.

PWM_ENABLED 0x0b Habilita/Inhabilita el controlador del PWM

Cmd: 10 0b Sin datos

Rta: No tiene.

GET_POSITION 0x0c Solicita/envía la posición actual del motor y banderas

Cmd: 12 0c Sin datos.

Rta: 13 0c 02 05 01 La posición del motor es 0x0502 y las banderas son 0x01

CHANGE_MODE 0x0d Cambia el modo de operación: local ó remoto

Cmd: 10 0d Sin datos

Rta: No tiene.

LOAD_PWM 0x0e Carga/recibe el valor del PWM

Cmd: 12 0b 3E A4 El PWM a cargar es 0xA43E

Rta: No tiene.

DEBUG_MODE 0x0f Cambia a modo DEBUG

Cmd: 10 0f Sin datos

Rta: 13 0c 02 05 01 La posición del motor es 0x0502 y las banderas son 0x01

**COMANDOS PARA EL PC DESDE COMPID**

FINISH_MOTION 0x0f Indica que el motor ha llegado a la posición final

Cmd: f0 0f Sin datos

Rta: No tiene.

SYS_READY 0x10 Indica que el RTOS y los motores están listos para operar

Cmd: F0 10 Sin datos

Rta: e0 13 Sin datos

RTOS_READY 0x11 Indica que el RTOS está listo para una nueva tarea

Cmd: f0 11 Sin datos

**DECLARACIONES PARA LOS INDICADORES DE UN CONTROLADOR DE MOTOR**

Motion bit0 Se ha finalizado el movimiento

Phase bit1 La fase del movimiento es (0=inicial/1=final)

Back bit2 El movimiento es negativo

Saturated bit3 El compensador está saturado (PWM máximo ó mínimo)

BreakMotion bit4 El motor está dinámicamente frenado

**DECLARACIONES PARA LAS BANDERAS DEL REGISTRO DE ESTADO DEL MOTOR**

Final0 bit0 El motor se encuentra en el origen
Final1 bit1 El motor se encuentra en final del eje
StopMtr bit2 El usuario utilizó la parada de emergencia del motor
StopAll bit3 El usuario utilizó la parada general de emergencia
Mode bit4 El modo de operación del motor: 0=local, 1=remoto
OkPos bit5 El motor alcanzó la posición final ordenada
HandBusy bit6 La pinza conectada a los motores esta en movimiento

OPERACIÓN

El controlador del motor inicia en modo REMOTO al encenderse. ComPID espera hasta cuando los motores se encuentran en posición de casa (HOME). Al cargarse ComPID, solicita la posición de cada motor y el estado del controlador. Esta condición se mantiene hasta cuando se reciba que los motores han sido ubicados en la posición HOME.

ComPID indica al PC que está listo para recibir un mensaje haciendo el pin DTR=1 (Verdadero). Verifica que el PC esté listo enviando el comando 0xF0 0x10 que requiere una respuesta del mismo mediante el pin DTR=1. El PC envía el mensaje 0xE0 13 confirmando que está listo.

El programador remoto después de establecer comunicación con el RTOS se configura por defecto como:

Modo Local – Ejecutar Perfiles. (Los comandos son transmitidos al controlador uC pasando por el RTOS)

En este modo ComPID-VB solicita los perfiles que se desean ejecutar (1-12), estos pueden ser editados antes de ser enviados. Al pulsar iniciar se enviarán los perfiles definidos actualmente.

Antes de pulsar iniciar se pueden editar los perfiles.

Al pulsar DETENER se ordena parar los movimientos en todos los motores. (Ej: 0x10 0x09) para el caso del motor # 1.

Los valores se transmiten: primero LSB y luego MSB enteros con signo

Para el caso de las constantes del control, cuando se modifican y se pulsa establecer, se envía el valor de la constante (La misma para los 3 motores) inmediatamente.

De no modificarse ninguna de las constantes, al darle iniciar se transmiten los valores de los perfiles indicados para los 3 motores. Transmitiendo parámetro por parámetro de cada perfil y así con cada uno de los perfiles indicados para ejecutar. (Distancia, Aceleración, Velocidad y Retardo). Acá no se transmiten las constantes del control.

Modo Local – Manual. En lo concerniente a las constantes del control, aplica igual que para el modo Local – Ejecutar perfiles. En este modo no se permite la edición ni la ejecución de perfiles. Los parámetros del modo manual pueden ser editados antes de pulsar iniciar.

Al pulsar iniciar se transmiten al controlador: La distancia relativa del movimiento, Aceleración, velocidad, retardo, KP, KI, KD, PWM máximo y PWM mínimo-.

Modo Remoto – Ejecutar Perfiles. En este modo no se transmiten las constantes de control al establecerlas manualmente (Funcionamiento correcto?). Los perfiles pueden ser editados antes de pulsar iniciar.

Al pulsar iniciar envía cada perfil (La distancia relativa del movimiento, Aceleración, velocidad, retardo, KP, KI, KD, PWM máximo y PWM mínimo). Envía el comando ejecutar perfil (01-12) entre cada uno de los perfiles. Finalmente se envía OK para cada motor (Ej: 0x10 0x0D)

Modo Remoto – Manual. No es permitido editar perfiles. Pueden editarse los valores Distancia, Aceleración y Velocidad. También pueden editarse las constantes del control sin ser enviadas.

Al pulsar iniciar se transmiten los valores del modo manual (Distancia, aceleración, velocidad y retardo). También son enviadas las constantes del control, PWM máximo y PWM mínimo. Al final es transmitido OK para cada motor.

Debug=0, Modo=0: Transmite Status, Posición medida, I del motor; cada vez que recibe 0X0C. El formato transmitido es KST2.
Debug=0, Modo=1: Transmite Status, posición medida, I del motor cada vez que recibe el comando 0X0C. El formato transmitido es HEX.
Debug=1, Modo=0: Transmite todos los datos cada vez que se genera un tick en el PIC. El formato transmitido es KST2.
Debug=1, Modo=1: Transmite todos los datos cada vez que recibe 0X0C. El formato transmitido es HEX.

**ANEXO B. DATASHEET LM75176.**

TEXAS INSTRUMENTS



# SNx5176B Differential Bus Transceivers

Check for Samples: SN65176B, SN75176B

## FEATURES

- Bidirectional Transceivers
- Meet or Exceed the Requirements of ANSI Standards TIA/EIA-422-B and TIA/EIA-485-A and ITU Recommendations V.11 and X.27
- Designed for Multipoint Transmission on Long Bus Lines in Noisy Environments
- 3-State Driver and Receiver Outputs
- Individual Driver and Receiver Enables
- Wide Positive and Negative Input/Output Bus Voltage Ranges
- ± 60-mA Max Driver Output Capability
- Thermal Shutdown Protection
- Driver Positive and Negative Current Limiting
- 12-kΩ Min Receiver Input Impedance
- ± 200-mV Receiver Input Sensitivity
- 50-mV Typ Receiver Input Hysteresis
- Operate From Single 5-V Supply

## DESCRIPTION

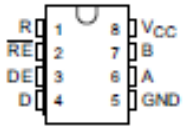
The SN65176B and SN75176B differential bus transceivers are integrated circuits designed for bidirectional data communication on multipoint bus transmission lines. They are designed for balanced transmission lines and meet ANSI Standards TIA/EIA-422-B and TIA/EIA-485-A and ITU Recommendations V.11 and X.27.

The SN65176B and SN75176B devices combine a 3-state differential line driver and a differential input line receiver, both of which operate from a single 5-V power supply. The driver and receiver have active-high and active-low enables, respectively, that can be connected together externally to function as a direction control. The driver differential outputs and the receiver differential inputs are connected internally to form differential input/output (I/O) bus ports that are designed to offer minimum loading to the bus when the driver is disabled or $V_{CC}$ = 0. These ports feature wide positive and negative common-mode voltage ranges, making the device suitable for party-line applications.

The driver is designed for up to 60 mA of sink or source current. The driver features positive and negative current limiting and thermal shutdown for protection from line-fault conditions. Thermal shutdown is designed to occur at a junction temperature of approximately 150°C. The receiver features a minimum input impedance of 12 kΩ, an input sensitivity of ±200 mV, and a typical input hysteresis of 50 mV.

The SN65176B and SN75176B devices can be used in transmission-line applications employing the SN75172 and SN75174 quadruple differential line drivers and SN75173 and SN75175 quadruple differential line receivers.

SN65176B D OR P PACKAGE
SN75176B D, P, OR PS PACKAGE
(TOP VIEW)



**ANEXO C. DATASHEET LM358.**



# LM158/LM258/LM358/LM2904 Low Power Dual Operational Amplifiers

Check for Samples: LM158-N, LM258-N, LM2904-N, LM358-N

## FEATURES

- Available in 8-Bump DSBGA Chip-Sized Package, (See AN-1112 (SNVA009))
- Internally Frequency Compensated for Unity Gain
- Large DC Voltage Gain: 100 dB
- Wide Bandwidth (Unity Gain): 1 MHz (Temperature Compensated)
- Wide Power Supply Range:
  - Single Supply: 3V to 32V
  - Or Dual Supplies: ±1.5V to ±16V
- Very Low Supply Current Drain (500 µA)—Essentially Independent of Supply Voltage
- Low Input Offset Voltage: 2 mV
- Input Common-Mode Voltage Range Includes Ground
- Differential Input Voltage Range Equal to the Power Supply Voltage
- Large Output Voltage Swing

## UNIQUE CHARACTERISTICS

- In the Linear Mode the Input Common-Mode Voltage Range Includes Ground and the Output Voltage Can Also Swing to Ground, even though Operated from Only a Single Power Supply Voltage.
- The Unity Gain Cross Frequency is Temperature Compensated.
- The Input Bias Current is also Temperature Compensated.

## ADVANTAGES

- Two Internally Compensated Op Amps
- Eliminates Need for Dual Supplies
- Allows Direct Sensing Near GND and $V_{OUT}$ Also Goes to GND
- Compatible with All Forms of Logic
- Power Drain Suitable for Battery Operation

## DESCRIPTION

The LM158 series consists of two independent, high gain, internally frequency compensated operational amplifiers which were designed specifically to operate from a single power supply over a wide range of voltages. Operation from split power supplies is also possible and the low power supply current drain is independent of the magnitude of the power supply voltage.

Application areas include transducer amplifiers, dc gain blocks and all the conventional op amp circuits which now can be more easily implemented in single power supply systems. For example, the LM158 series can be directly operated off of the standard +5V power supply voltage which is used in digital systems and will easily provide the required interface electronics without requiring the additional ±15V power supplies.

The LM358 and LM2904 are available in a chip sized package (8-Bump DSBGA) using TI's DSBGA package technology.