Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Unidad Controladora de Procesos para el diseño, análisis, simulación e implementación de sistemas de control automático a través de redes TCP/IP

Informe de Proyecto de Graduación para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura

Ejecutor:

Daniel Castro Molina

Cartago, 27 de junio de 2008

Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Proyecto de Graduación

Tribunal Evaluador

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal

Dr. Pablo Alvarado Moya
Profesor Lector

Ing. Gabriela Ortiz León
Profesora Lectora

Ing. Eduardo Interiano Salguero
Profesor Asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica

Cartago, 25 de Junio del 2008

TEC
Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica

Declaratoria de Autenticidad

Declaro que el presente Proyecto ha sido realizado por mi persona, utilizando y aplicando literatura referente al tema, así como la información que haya suministrado la institución para la que se realizó el proyecto, y aplicando e introduciendo conocimientos propios. En los casos en que he utilizado bibliografía, he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad por el contenido de este Proyecto.

Cartago, 25 de Junio del 2008

Daniel Castro Molina

Cédula: 1 – 1140 – 0860

Índice general

Índice de figuras

Índice de tablas

Agradecimiento

Primero deseo darle gracias a Dios por acompañarme siempre en este largo camino e iluminarme con su sabiduría y persistencia.

A mis padres Jorge Castro Umaña y Ruth Molina Loaiza por darme el don de la vida, por enseñarme a ser paciente y brindarme apoyo durante toda mi carrera.

A mi profesor asesor Ing. Eduardo Interiano Salguero por su motivación al trabajo, por su muy acertada asesoría y al gran interés que mostró durante el desarrollo de este proyecto.

Agradezco también a don Gerardo Herrera Herrera y a Ana Catalina Villalobos Gonzáles del Laboratorio de Electrónica y Circuitos Impresos del ICE por la valiosa colaboración.

Resumen

El laboratorio de Control Automático de la escuela de Ingeniería en Electrónica del Instituto Tecnológico de Costa Rica está dedicado a la investigación y experimentación en sistemas de control automático.

Los sistemas de control automático tienen un sin fin de aplicaciones. Es común encontrar éstos en muchos artefactos electrónicos desde los de uso cotidiano como electrodomésticos hasta sistemas aeroespaciales. Uno de los objetivos del laboratorio es el de introducir a los estudiantes al estudio y aplicación del control automático; sin embargo, el laboratorio no cuenta con herramientas de desarrollo que motive y facilite tal propósito.

Mediante la aplicación de la electrónica digital y sistemas operativos en tiempo real se elaboró un sistema de entrenamiento que aprovecha las tecnologías de comunicación actuales como la internet. Este sistema permite estimular, muestrear variables, crear estructuras de control y todo lo necesario, ajustado a la metodolgía de enseñanza del laboratorio para aplicar de manera práctica los conceptos del control automático.

El sistema es abierto y sirve como modelo de desarrollo, disponible para actualizaciones y recomendaciones.

Abstract

The Electronic Automatic Control Lab of the Electronic Engineering School at Costa Rica Institute of Technology is dedicated to the research in automatic control systems and processes.

The automatic control systems have an endless number of applications and it is very common to find them in many electronic devices from the simplest appliances to the most complex military and aerospace systems; therefore, the main Lab's objetive is to introduce students to the study of automatic control applications. Unfortunately, the Lab does not have development tools that encourage and facilitate such tasks.

This project applies digital electronics and real-time operating systems for the creation of a training system that leverages technologies of communication like the Internet. This system allows: stimulation, variable sample and creation of control structures.

The system remains open as a development model, which is available for upgrades and recommendations.

Capítulo 1

Introducción

El presente trabajo se realizó como parte del proyecto TeleLAB del laboratorio de control automático de la escuela de Ingeniería Electrónica del ITCR. El proyecto TeleLAB nace hace 3 años con el propósito de crear un laboratorio "en línea" para la experimentación en sistemas de control automático.

El laboratorio en línea TeleLAB, es un proyecto que tiene como objetivo de darle al usuario la posibilidad de realizar experimentos de control automático a través de internet. Esto abre una amplia gama de beneficios como: un mayor control de horario de uso de los dispositivos, se podría definir cuentas de usuario para uso exclusivo de los estudiantes, permitiría a los profesores hacer demostraciones en clase y además, daría una excelente imagen a nivel internacional sobre los proyectos que se desarrollan en la Escuela de Electrónica.

Los estudiantes han elaborado diseños, propuestas, ideas y han construido una gran variedad de plantas con el fin de utilizarlos para el proyecto TeleLAB. Desafortunadamente, no se han logrado avances considerables que permitan el funcionamiento completo del TeleLAB.

1.1. Problemática del proyecto TeleLAB

El problema reside principalmente en la falta de recurso humano, organización e inversión económica; efecto de ello, es la carencia de un sistema de desarrollo que utilice el proyecto TeleLAB como modelo y que permita su completa operación. Este sistema de desarrollo debe suplir las necesidades técnicas que permitan aplicar los conceptos de la teoría de control automático en los experimentos que se realicen.

El proyecto inició con la compra de tarjetas de adquisición y desarrollo de aplicaciones con LabView, ambos de la marca National Instruments, con esto se desarrollaron varios experimentos en donde el estudiante puede obtener datos de motores eléctricos como: tensión, corriente, velocidad, posición del eje, etc. Estos datos se pueden analizar matemáticamente con *software* especializado como Matlab u Octave; posteriormente, el sistema facilita la creación y verificación de controladores de posición, velocidad o según sea la aplicación.

National Instruments brinda soluciones tanto de *hardware* y *software* para la aplicación computacional en el área de control automático e industria; sin embargo, el desarrollo de aplicaciones en LabView necesita de programadores capacitados y con experiencia en sistemas de control automático, además, las tarjetas de adquisición de datos son bastante costosas (mínimo $1000). A futuro, el TeleLAB sería un proyecto costoso tanto en inversión y mantenimiento.

Como solución económica, se estimuló que cada grupo de estudiantes desarrollen su propio sistema electrónico (analógico o digital) como proyecto final del curso. Dependiendo del éxito de este proyecto se motivaría al grupo de estudiantes para el establecimiento del proyecto como parte del TeleLAB.

Aunque muchos proyectos fueron exitosos, ninguno se ha podido añadir al TeleLAB porque no cumplen los requerimientos técnicos para funcionar como experimento, tales como: capacidad del sistema para integrarse a una red TCP/IP, facilidad de modificación de los parámetros del control, ampliación de funciones (*hardware* o *software*),

documentación, etc.

1.2. Unidad Controladora de Procesos

Se proyecta crear una unidad que permita la adquisición de datos. Es necesario que incluya conectividad *Ethernet* para que sea independiente de una PC de manera local.

Su operatividad debe soportar los controladores y las topologías de retroalimentación comunes vistas en la asignatura de Control Automático y contar con diferentes interfaces de entrada/salida.

La interfaz con el usuario debe ser una página web en la que se permita, modelar la planta y verificar el funcionamiento del sistema en tiempo real.

Para verificar el funcionamiento de la unidad diseñada y construida para este proyecto, se pondrá a prueba en un *Ball & Beam*; además, la página web debe ser funcional y abierta a modificaciones.

La figura 1.1 representa una arquitectura básica del TeleLAB.

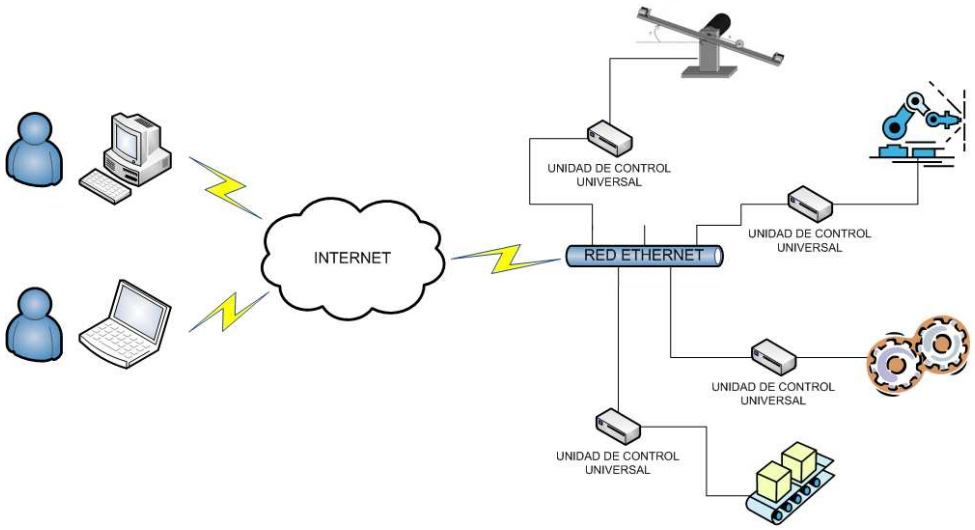


Figura 1.1: Esquema básico del TeleLAB.

En secciones anteriores se señaló que el Laboratorio TeleLAB no se encuentra en operación por problemas organizativos y económicos, pero en especial, porque no existe una base guía sobre la cual se puedan iniciar proyectos y asignación de investigaciones complementarias para mejoras y modificaciones.

El laboratorio TeleLAB, tiene la meta fundamental de permitir conectividad remota al laboratorio para que el usuario pueda diseñar controladores y verificar su funcionamiento en plantas reales, ver figura 1.2.

Figura 1.2: Diagrama de bloques del TeleLAB.

Hay que tener en cuenta que cada planta del laboratorio es diferente una de la otra; estas pueden ser: las características eléctricas, comportamiento físico, perturbaciones externas, etc; es por ello que la solución propuesta debe cumplir los siguientes requisitos:

- *Escalable:* Se refiere a que le sistema debe tener la posibilidad de expansión, ya sea por *hardware* o *software*.
- *Abierto:* En cuanto a *hardware* y *software* para que se puedan hacer modificaciones y mejoras.
- *Eficiente:* Debe hacer uso de solo aquel *hardware* necesario que requiera la planta.
- *Flexible:* El sistema debe adaptarse fácilmente o con modificaciones externas, a cualquier sensor, actuador que se requiera y a toda planta que se desarrolle en el laboratorio.
- *Disponibilidad:* Todas las herramientas de desarrollo y los componentes electrónicos deben ser de fácil adquisición.

La figura 1.3 muestra un esquema de *software* de la unidad controladora y sus componentes externos que permitirá realizar los experimentos.

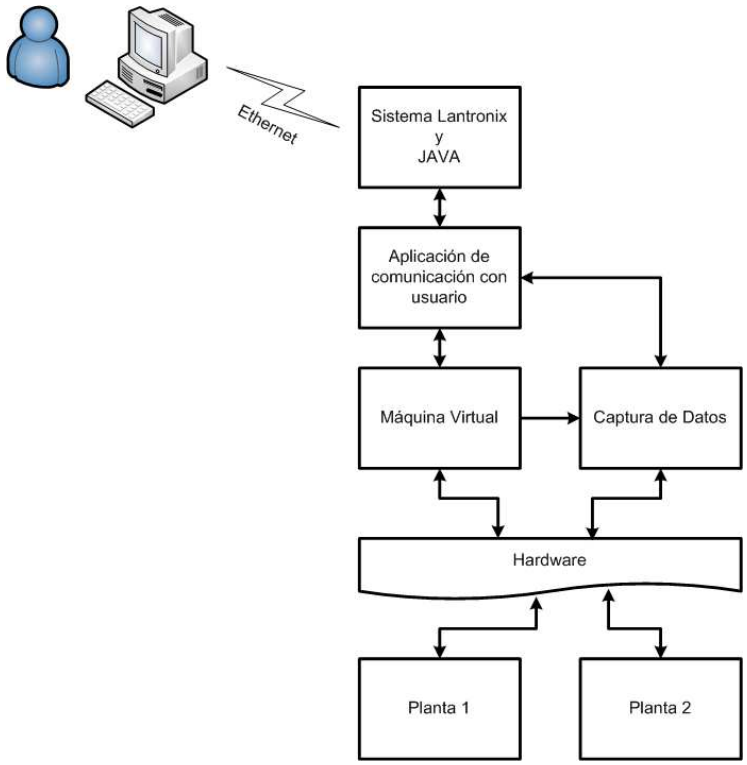


Figura 1.3: Esquema básico de *software* de la unidad controladora.

Se pretende utilizar un RTOS que optimize el uso de la CPU y simplifique la aplicación en varias tareas. El usuario se conecta mediante una interfaz gráfica creada con JAVA.

Esta interfaz de usuario se comunica con la unidad mediante una tarea de comunicaciones. Ésta interpretará y ejecutará las instrucciones de usuario. Deberá existir una tarea (máquina virtual) que se encargan de utilizar el *hardware* de la unidad y las funciones de control para estimular o controlar las plantas. La tarea de captura permitirá almacenar datos y resultados de las funciones para que posteriormente el usuario los analice en su computador.

1.3. Estructura de éste trabajo

En el Capítulo 2, se establecen los objetivos que sirvieron de guía durante este proyecto.

En el Capítulo 3, se presentan conceptos introductorios sobre sistemas operativos en tiempo real (RTOS). El uso de un RTOS en este proyecto simplificó el desarrollo de *software* y dió un manejo eficiente de los recursos de *hardware*.

En el Capítulo 4, se explican las consideraciones y parámetros de diseño que se utilizaron como bases de investigación y desarrollo.

En el Capítulo 5, se muestra el diseño de la unidad controladora de procesos como un conjunto de capas. Se explica el significado y el aporte de cada capa.

En el Capítulo 6, se explica la aplicación de Máquina Virtual.

En el Capítulo 7, se detalla los resultados de la utilización de la unidad controladora de procesos para el modelado y control de un *Ball & Beam*. Además, se muestra la aplicación *web* que se utiliza para experimentar con la planta y se detallan las limitaciones del diseño de la unidad controladora de procesos.

Finalmente, en el Capítulo 8, se encuentran las conclusiones y recomendaciones de este proyecto.

Capítulo 2

Meta y objetivos

2.1. Meta

Permitir que el proyecto en línea TeleLAB avance construyendo un sistema de adquisición de datos económica, flexible y escalable a nivel de *software* y *hardware* para el Laboratorio de Control Automático del ITCR, minimizando así la dependencia de herramientas comercialmente costosas dedicadas al mismo fin.

2.2. Objetivo general

Crear una unidad electrónica que permita el diseño, análisis e implementación de sistemas de control automático; que sea escalable, abierto, eficiente, flexible, que utilice internet como medio de comunicación y que permita acoplamiento de nuevas plantas ofreciéndose como una opción útil para la expansión del proyecto TeleLAB del laboratorio de control automático del ITCR.

2.3. Objetivos específicos

1. Crear una unidad electrónica que estimule y adquiera datos de la planta; que contenga funciones de control automático y diferentes interfaces de entrada/salida analógicas o digitales.

2. Elaborar una página contenida en un servidor web que asista al usuario a: modelar la planta, diseñar controles y aplicarlos a la unidad electrónica para que se pueda verificar la estabilidad o inestabilidad de la planta en tiempo real.

3. Comprobar el funcionamiento de la unidad electrónica en un *Ball & Beam*.

Capítulo 3

Marco teórico

En este capítulo se desarrolla de manera general el siguiente tema:

Sistemas operativos en tiempo real (RTOS): Aplicación y uso de sistemas operativos en sistemas embebidos. Conceptos básicos.

3.1. Sistemas operativos de tiempo real (RTOS)

Un sistema operativo de tiempo real, es un sistema operativo especialmente diseñado para funciones en tiempo real ó aplicaciones en el que el retardo de respuesta del sistema sea lo mínimo posible. Una exigencia es que el sistema operativo sea deteminista; es decir, debe decidir sobre cuales tareas ejecutar de acuerdo a los eventos que operan a su alrededor.

El concepto de RTOS contrasta de la programación orientada a un solo ciclo. Los programas de un solo ciclo se definen como sistemas *foreground/background*[14].

3.1.1. Sistemas unicíclicos

La programación orientada a un solo ciclo es muy común cuando se desarrollan aplicaciones de baja complejidad. Este tipo de aplicaciones ejecutan diferentes funciones

o rutinas de manera secuencial. En [14] se define a esta sección como *background* o nivel de tarea. Cualquier evento asíncrono es manejado dentro de una interrupción (ISR) conocido también como *foreground* ó nivel de interrupción.

La figura 3.1 muestra un esquema de los sistemas *foreground/background*.

El problema de los sistemas *foreground/background* está en que las operaciones criticas son manejadas por las ISR. Esto produce una tendencia de que si hay muchos eventos asíncronos, serán las ISR las que tomen la mayoría del control de la CPU dejando de lado el nivel de tarea. Este tipo de sistema no es deteminista porque el nivel de tarea siempre se ejecuta una vez que termine la ISR.

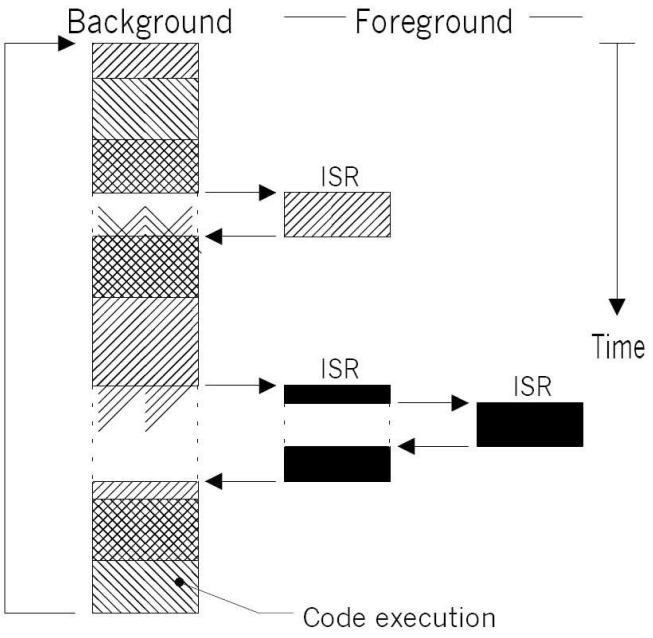


Figura 3.1: Esquema de los sistemas *foreground/background* [14].

El uso de este esquema no es adecuado para un sistema de control. El retraso del cálculo de una operación con tiempo exigente, tal como un filtro o un controlador digital, se puede haber afectado por ejecución de código no prioritario incrementando el tiempo de cálculo; por ejemplo, una ISR por fin de transmisión de datos de algún

bus de comunicaciones interrumpe el cálculo de un filtro digital.

Durante el cálculo de una muestra, el sistema digital deja prácticamente a lazo abierto la planta. Si el cálculo de una muestra es retrasado continuamente, el efecto puede ser inestabilidad o comportamientos no esperados en el funcionamiento del control.

3.1.2. Conceptos y partes de un RTOS

A diferencia del sistema unicíclico, un RTOS administra eficientemente el uso de la CPU; para ello, el usuario debe desarrollar su aplicación como un conjunto de tareas. Cada tarea actúa de acuerdo a eventos del *hardware* o del *software*; el RTOS se encarga de verificar y asignar la CPU a determinada tarea.

¿Qué sucede ahora con las ISR? El código de las ISR solo usarán funciones para indicarle al RTOS que una tarea debe ser ejecutada, es decir, el nivel de interrupción tendrá un código reducido y por lo tanto se ejecutará rápidamente. Esto reduce considerablemente el tiempo de ejecución de la ISR dando prioridad al nivel de tarea. El RTOS, en este caso, determinará el momento adecuado para ejecutar las rutinas que necesita el evento asíncrono. La importancia de la tarea la especifica el programador asignado una prioridad única a la tarea. Es importante que el programador diseñe y clasifique cada tarea. Una buena clasificación de las tareas hacen que un RTOS sea verdaderamente determinista.

En este proyecto, las tareas se pueden clasificar en:

1. **Operaciones críticas:** Como la ejecución de un controlador, filtro o cualquier operación que deba controlar una planta y tenga tiempo de ejecución exigente. Estas operaciones deben tener la prioridad más alta, pues deben ejecutarse inmediatamente.

2. **Operaciones intermedias:** Son todas aquellas funciones en que no son necesa-

rios tiempos de ejecución exigentes o en que su momento, no son de importancia para el usuario. Estas pueden ser rutinas de verificación de *hardware*, liberación de memoria, actualización de información con otros elementos del sistema, etc.

3. **Operaciones de usuario:** Son aquellas en donde el microcontrolador utiliza de sus recursos para brindar algún tipo de información al usuario o viceversa. En este caso, tales operaciones puede ser pospuestas cuando se deba ejecutar una operación crítica o intermedia. Al nivel de usuario, por lo general, se le asigna la prioridad mas baja porque el tiempo de percepción de un ser humano es lento comparado con los tiempos de ejecución de un microcontrolador.

Para comprender el funcionamiento de un RTOS es necesario entender varios conceptos.

Secciones críticas: Es una sección de código que no admite interrupciones. Las razones puede ser:

- El código calcula un resultado y este debe ejecutarse lo más rápido posible.
- El código no es reentrante y no pude haber cambio de contexto.

Por lo general, se deshabilitan todas las interrupciones del sistema antes de ejecutar la sección crítica y se habilitan nuevamente una vez terminada la sección crítica. Cualquier RTOS debe incluir rutinas que permitan al programador establecer secciones críticas. En un sistema de control, la sección crítica puede ser cuando el microprocesador debe tomar la muestra y calcular la salida hacia la planta.

Contexto de la tarea: Se refiere a los resultados intermedios o información que tenía la tarea en los registros de la CPU antes que el sistema operativo detuviera la tarea. Un problema de los RTOS es que el cambio de contexto produce carga de trabajo a la CPU.

Multitasking: Es el proceso que permite que la CPU conmute y ejecute varias tareas o aplicaciones.

Tarea: Una tarea es un programa o rutina. El diseño de una aplicación utilizando un sistema operativo consiste en dividir la aplicación en tareas según como el programador clasifique los eventos o acciones. Cada tarea tiene una prioridad y una sección propia de memoria llamada pila. Esta pila es necesaria para guardar el contexto de la tarea cuando el núcleo conmuta la CPU para ejecutar otra tarea.

Recursos: Un recurso puede ser un dispositivo de E/S, una estructura, un arreglo de datos o simplemente una variable. Estos son usados por las tareas.

Recursos compartidos: Es un recurso que es utilizando por más de una tarea.

Núcleo: Administra y permite la comunicación entre tareas. Una desventaja es que el núcleo requiere de la CPU y de la memoria para realizar sus procesos internos. [14] establece que en un sistema bien diseñado, un núcleo puede llegar a utilizar entre un 2 y 5 % de la CPU.

Administrador de tareas: Determina cuál es la siguiente tarea a ejecutar. La decisión del administrador de tareas se basa en el determinismo programado por el usuario. Este determinismo se especifica asignando una prioridad a cada tarea. La tarea depende de factores externos o internos que hacen que ésta tenga estados, estos estados puede ser (figura 3.2):

1. *En ejecución:* Es cuando la tarea está utilizando la CPU.
2. *Lista:* Es cuando una tarea está esperando que el núcleo le ceda la CPU ya que otra tarea de mayor prioridad está siendo ejecutada.
3. *Esperando:* Es cuando una tarea está esperando por algún evento del *hardware*, tal como una acción de algún dispositivo E/S, un evento por *software*

ó esperar por algún resultado de otra tarea. Cuando la tarea reconoce el evento pasa del estado de espera a lista.

4. *Inactiva:* Una tarea inactiva es una tarea que reside en memoria pero no participa del proceso de conmutación del núcleo.

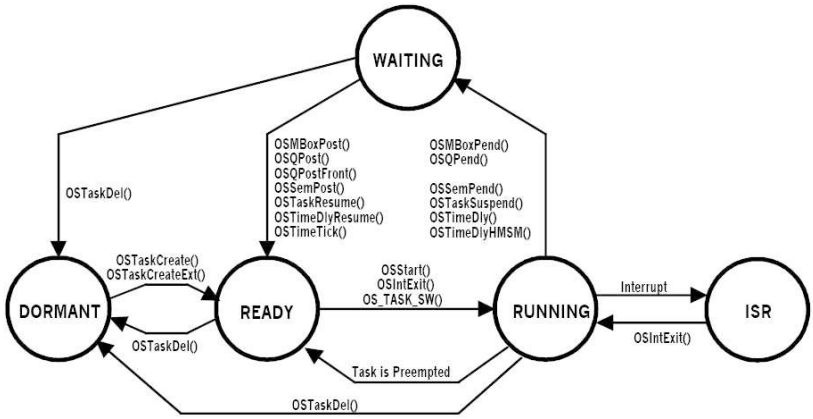


Figura 3.2: Estados de las tareas [14].

Núcleo cooperativo: Un núcleo cooperativo cede el control de la CPU por ciertos momentos de tiempo a cada tarea. La figura 3.3 muestra un esquema de cómo se conmutan las tareas en un núcleo cooperativo.

El programador asigna un tiempo de expiración que representa el tiempo máximo de ejecución de la tarea. Si la tarea no termina antes de el tiempo límite, el núcleo avisa al administrador de tareas para que se guarde el contexto de la tarea y se ejecute otra tarea.

Un núcleo cooperativo no es de tiempo real, pues la tarea de mayor prioridad se ejecuta si y solo si la tarea de menor prioridad termina su código o el tiempo límite vence. Esto produce un retardo en la ejecución de la tarea de mayor prioridad.

Núcleo apropiativo: El objetivo de un núcleo apropiativo es ceder el control de la

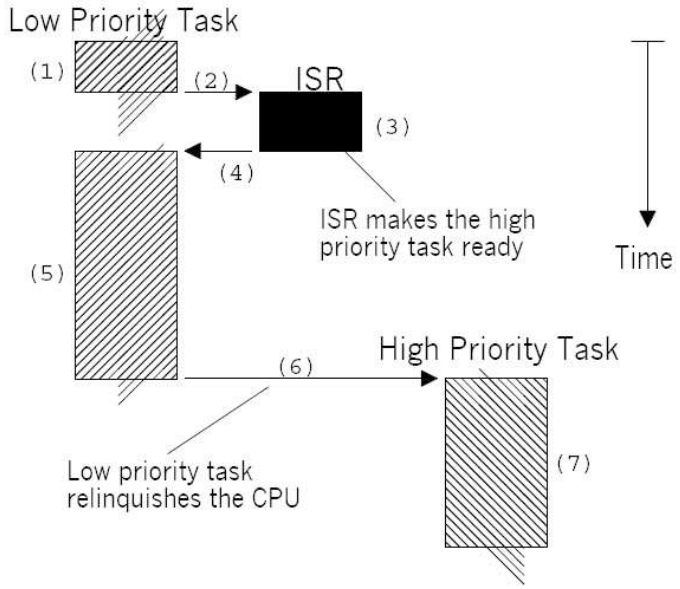


Figura 3.3: Núcleo cooperativo [14].

CPU a aquella tarea de mayor prioridad que este lista para ejecutarse. Este tipo de núcleo es de tiempo real. La figura 3.4 muestra un esquema de como se conmutan las tareas en un núcleo apropiativo.

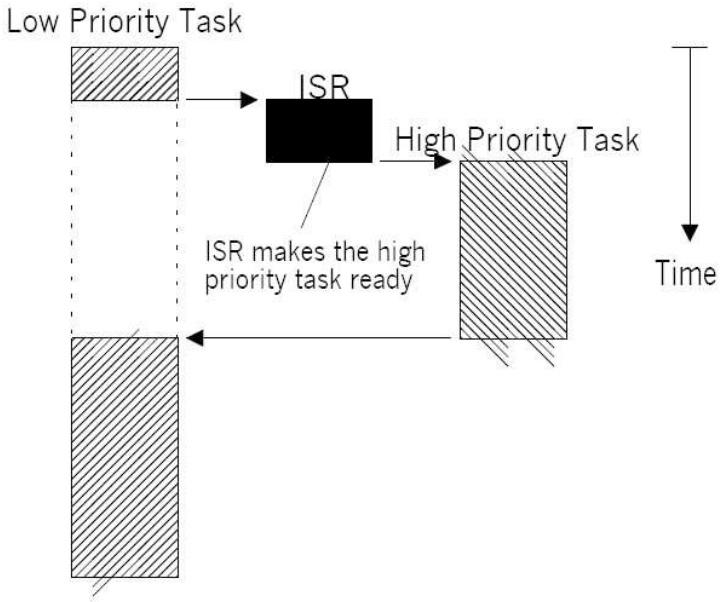


Figura 3.4: Núcleo apropiativo [14].

Capítulo 4

Consideraciones previas sobre el diseño de la unidad

En el mercado existen múltiples productos que pueden solucionar los problemas del TeleLAB; tales como, tarjetas de adquisición de datos y *software* de aplicación desarrollado en LabView de National Instruments. Otra alternativa, es utilizar un PLC o un controlador de procesos. Para tales ejemplos existe una gran diversidad de marcas y modelos; sin embargo, estos dispositivos son costosos.

En este proyecto se trataron aspectos generales que funcionen como base del proyecto TeleLAB y no aspectos específicos de una aplicación. Entre los aspectos fundamentales de las características del sistema se concluyó:

1. **Respecto al *hardware* del sistema:** Diseñar y construir el sistema con las siguientes funciones básicas:

   a) Entradas:

   - Entradas analógicas, rango máximo de ±10 V.
   - Entrada de QEI (*Quadrature Encoder Interface*) para medición digital de velocidad y posición angular de motores.

- Referencias o constantes numéricas dinámicas, ingresadas por el puerto de comunicaciones.

b) Salidas:

- Salidas PWM, tanto para control de motores CD como CA.
- Salidas analógicas, rango máximo de ±10 V y baja potencia.

c) Buses de datos para expansiones de E/S:

- Bus I^2C.
- Bus SPI.

d) Almacenamiento de datos y configuración:

- Memoria para guardar configuraciones y datos.

e) Comunicaciones:

- Puerto serie tipo RS232 y posible mejora a USB.
- Puerto *Ethernet* para comunicaciones TCP/IP.

2. **Respecto al *software* del sistema:** La unidad debe contar con el *software* necesario para ejecutar operaciones de control automático y tratamiento de señales:

- Funciones de filtros IIR y FIR.
- Generación de señales de prueba.
- Funciones de transferencia.
- Selección de tiempo de muestreo.
- Realimentación de estado (4 estados máximo).
- Funciones PID y variantes.
- Operaciones escalares y vectoriales.

- Selección, configuración y calibración dinámica del *hardware*.

Debe existir una capa de *drivers* o controladores de *hardware* que facilite la utilización de los recursos. En tiempo de compilación, todos los periféricos deben ser configurados dentro de un único archivo. Además, debe tener la posibilidad de cambiar esta configuración mediante instrucciones de alto nivel en tiempo de ejecución.

Como última característica, la unidad debe tener programada una máquina virtual para la interpretación de aplicaciones. El propósito de la máquina virtual es el de permitir la interpretación de código de mayor nivel dedicado a la interconexión y ejecución de módulos virtuales. Cada módulo virtual es dinámico y está dedicado a ejecutar una función única, por ejemplo, un módulo encargado de tomar muestras analógicas, o un módulo que ejecuta la función de un filtro digital.

Cada módulo contiene ciertos parámetros que permiten cambiar su modo de funcionamiento; por ejemplo, el filtro digital tiene parámetros de: orden, frecuencia de corte, tipo de filtro, etc. La principal ventaja es que el funcionamiento no está creado estáticamente para una determinada aplicación; sino que el usuario puede crear sus propias estructuras de control y hasta puede simular el comportamiento de su planta utilizando señales de entrada y salida reales, además, las modificaciones del *firmware* son necesarias solo cuando se desea agregar más funciones o módulos, dar mantenimiento al sistema o agregar algún tipo de *hardware* no considerado en este proyecto.

3. **Respecto a las herramientas de desarrollo y *software*:** Utilizar en lo posible *software* libre o con libre acceso institucional, así mismo, que cada herramienta esté respaldada con documentación, manuales, etc.

4.1. Síntesis de la solución

El sistema desarrollado es un sistema embebido con interfaces de entrada/salida, procesamiento de señales, algoritmos e interfaces de comunicación.

Además de controlar una planta o proceso, el sistema debe responder a peticiones de los periféricos, detectar errores y obedecer a las órdenes del usuario, es decir, se deben realizar múltiples tareas en un solo sistema de procesamiento; en resumen, se destaca lo siguiente:

- Respuesta real en el tiempo: Atención casi inmediata a los eventos.
- Sistema multitarea: Ejecución de varias tareas.

La utilización de un RTOS simplifica el diseño de una aplicación ya que el programador puede dividir la aplicación de acuerdo a los eventos o funciones en pequeñas tareas que el RTOS puede administrar. Es por ello, que el proceso de diseño de ingeniería comenzó con la selección de un sistema operativo en tiempo real para sistemas embebidos y no con el diseño de un algoritmo general para el control de la unidad, que en sí, haría del sistema un programa unicíclico (sección 3.1.1).

4.1.1. FreeRTOS vs MicroC/OS-II

Existen diversos tipos de RTOS. Los hay de uso general, especializados en aplicaciones de procesamiento digital de señales, aviación, industria, medicina, etc. Los desarrolladores comerciales más populares de RTOS son: AVIX, ThreadX, AVA, Micrium y Salvo. Estos RTOS tienen la ventaja de un diseño propio y con respaldo de una empresa; pero, requieren de una licencia para su uso. Cada uno tiene un amplio soporte para microcontroladores de todo tipo (ARM, MIPS, 8051, etc).

También, existen los RTOS de uso libre tales como: µCLinux y FreeRTOS.

µCLinux (`http://www.uclinux.org`), se basa en el núcleo Linux pero tiene como desventaja que solo es operable en microcontroladores con arquitecturas de 32bits tipo ARM y otros.

FreeRTOS (`http://www.freertos.org`), tiene soporte para una amplia gama de microcontroladores, es abierto y se puede utilizar sin restricciones, tiene la desventaja de que la documentación y manuales son solo para usuarios experimentados en la materia de RTOS.

Otro RTOS comercial de uso libre pero con limitaciones (la versión completa está disponible libremente con fines académicos e investigación), es el MicroC/OS-II o µC/OS-II, su documentación es amplia[14], sencilla y con soporte directo del desarrollador Micrium (`http://www.micrium.com`).

Se tomó la decisión de comparar entre FreeRTOS y MicroC/OS-II debido a que son los únicos sistemas operativos de licencia libre y que soportan arquitecturas más simples y económicas como los microcontroladores de Microchip. La tabla 4.1 muestra información de las características más sobresalientes de estos RTOS.

- **Política del núcleo:** En secciones anteriores se comentó que el núcleo apropiativo es determinista; este determinismo lo asigna el programador mediante prioridades a las tareas. En esta aplicación, un núcleo apropiativo tiene la ventaja de que las tareas con tiempo de respuesta exigentes (control automático y procesamiento digital) se les asigna con una prioridad alta, mientras que las tareas de menor exigencia de tiempo (interfaz con el usuario, comunicaciones) se asignan con una prioridad baja. FreeRTOS ofrece los dos tipos de núcleo, es decir, se permite que varias tareas puedan tener la misma prioridad; esta característica, provoca que el núcleo apropiativo se comporte al mismo tiempo como un núcleo cooperativo (solo en aquellas tareas con igual prioridad). En estos casos, el administrador de tareas opera mediante *round robin time slicing*, que consiste en asignar (programador) un tiempo de expiración a cada tarea de igual prioridad,

Tabla 4.1: Comparación entre FreeRTOS y MicroC/OS-II.

Característica	FreeRTOS	MicroC/OS-II
Política del núcleo	Cooperativo y Apropiativo mediante declaraciones explícitas tipo *yield* (ceder la CPU)	Apropiativo
Round robin time slicing	Soportado, permite que dos o más tareas tengan la misma prioridad	No soportado
Prioridades dinámicas	Soportado	Soportado
Resolución de *Priority Inversions*	Solo cambiando las prioridades de las tareas	Soportado
Banderas de eventos	No soportado	Soportado
Colas de mensajes	Soportado	Soportado
Semáforos	Soportado	Soportado
Administración de tiempo	Opciones básicas como contador de *ticks*	Soporte más avanzado como retardos en unidades de tiempo
Utilidades de tarea	Soporte básico	Soporte más avanzado con utilidades de estadística y registro
Administración de memoria	Soportado	Soportado

esto evita que una tarea se mantenga por mucho tiempo la CPU. MicroC/OS-II solo ofrece el núcleo apropiativo limitando las opciones y herramientas al programador.

- **Prioridades dinámicas:** Esta característica de los RTOS permite que la prioridad de una tarea se pueda cambiar desde otra tarea o dentro de una interrupción.

- **Resolución de *priority inversions*:** Este es un método de corrección que ofrecen los RTOS a un problema que ocurre frecuentemente entre el administrador de tareas y los recursos compartidos. Los *priority inversions* suceden cuando una tarea de menor prioridad toma un recurso compartido; si instantes después una tarea de mayor prioridad intenta tomár el mismo recurso, esta tarea de mayor prioridad debe "esperar" a que la tarea de menor prioridad libere el recurso. La tarea de mayor prioridad queda virtualmente con una prioridad menor que la tarea de menor prioridad; el problema se empeora cuando existe una tarea de prioridad intermedia que provoca que la tarea de menor prioridad se retrase en liberar el recurso compartido; en este caso, puede decirse que el RTOS no está actuando de manera determinista. La resolución de *priority inversions* consiste en utilizar semáforos de exclusión mutua (*mutexes*), los *mutex* son un tipo de semáforo especial que mantiene registro de la tarea que está utilizando el recurso compartido.

  Si una tarea de mayor prioridad solicita el mismo recurso (mediante una petición *mutex*), el administrador de tareas leerá el registro de *mutexes* y dará preferencia de asignación de la CPU a la tarea que mantiene el recurso compartido; este mecanismo permite que la tarea con el recurso compartido finalice rápidamente.

- **Banderas de eventos:** Esta es una herramienta de los RTOS que permiten sincronizar tareas mediante eventos especiales, tales como la finalización de un procedimiento, una operación matemática, o cambio de estado de una tarea.

- **Colas de mensajes:** Las colas de mensajes es un mecanismo que permite compartir información entre tareas.

- **Semáforos:** Los semáforos son funciones del RTOS que permiten reservar recursos compartidos tales como: registros, periféricos E/S, procedimientos, etc. Los semáforos evitan que la información o el funcionamiento de una tarea sea modificada por otra tarea.

- **Administración de tiempo:** Esta funciones permiten crear retardos, temporizaciones de las tareas. Las utilidades de tiempo son importantes cuando se desea esperar por algún dato, o por la finalización de una ejecución de un periférico, etc. FreeRTOS ofrece estas utilidades en unidades de *ticks* del RTOS, los *ticks* son las cuentas de reloj del RTOS en un determinado periodo, el programador especifica la cantidad de *ticks* por segundo que debe tener el RTOS; ejemplo, 1000 *ticks* por segundo significa que la base de tiempo del RTOS es 1ms, en este caso, si se genera un retardo de 3 *ticks*, este retardo corresponde a 3ms. MicroC/OS-II ofrece el mismo mecanismo de *ticks*, sin embargo, también se puede utilizar un método más amigable para el programador en unidades de milisegundos, segundos, minutos y horas.

- **Utilidades de tarea:** Las utilidades básicas de tarea permiten generar listas de tareas activas, verificar el estado y prioridad de una tarea, entre otras. FreeRTOS solo da soporte a las mencionadas. MicroC/OS-II tiene las utilidades básicas y además utilidades de estadística que permiten verificar el total de memoria libre en la pila y el porcentaje de uso de la CPU.

- **Administración de memoria dinámica:** Los RTOS brindan funciones alternativas de administración y uso de memoria dinámica.

Se seleccionó MicroC/OS-II como el RTOS para este proyecto. El principal motivo está en que MicroC/OS-II presenta una documentación apta para usuarios principiantes

con ejemplos sobre cada función del RTOS, soporte por correo electrónico, notas de aplicación y proyectos en C de ejemplo.

4.1.2. Selección del microcontrolador

La selección correcta se enfoca en el tipo de aplicación, soporte y precio. Se eligieron los microcontroladores de Microchip (PIC); aunque Microchip no tiene los microcontroladores más potentes de la actualidad (en cuanto a velocidad, tamaño del bus de datos, tipo de arquitectura, etc), Microchip ofrece todo el *software* de desarrollo (compilador C o ASM, bibliotecas, el ambiente de programación (IDE), códigos de ejemplo) de manera gratuita.

Entre todas las familias de Microchip la que más resalta en aplicaciones de mayor nivel es la serie dsPIC33F. Su principal característica es que esta nueva serie incorpora un CPU de 16 bits de arquitectura Harvard modificada y procesamiento de hasta 40MIPS.

El dsPIC33FJ256MC710 presenta las siguientes características:

- 40 MIPS con arquitectura Harvard modificada.
- Ruta de datos de 16 bits, e instrucciones de 24 bits.
- Direccionamiento lineal de memoria de programa de 4M instrucciones.
- Direccionamiento lineal de memoria de datos de 64 Kbytes.
- Dieciséis registros de propósito general de 16 bits.
- Operaciones multiplicación de 16×16 fracción/enteros.
- Multiplicación Acumulación para funciones de procesamiento digital de señales (PDS) con *dual data fetch* (se refiere a una característica especial de los procesadores digitales de señales que permiten al procesador leer o escribir dos secciones de memoria simultáneamente).

- 8 canales *hardware* para DMA con *buffer* de 2kbytes.
- Todas las entradas toleran 5V.
- Comunicaciones SPI (2), I^2C (2), UART (2).
- Control de motores con PWM (8 canales).
- Módulo QEI (*Quadrature Encoder Interface*).
- 2 módulos ADC, 10-bit, 1.1 Msps o 12-bit, 500 Ksps con 2, 4 or 8 muestras simultáneas.

Además, el compilador C30 de Microchip incorpora una biblioteca básica de funciones matemáticas y de PDS optimizadas en punto fijo.

4.1.3. Herramientas de desarrollo

Se utilizaron las siguientes placas de evaluación:

- Placa de desarrollo Microchip Explorer 16 (`http://www.microchip.com`).
- Programador & Debugger Microchip MPLAB ICD2.
- Placa de evaluación Lantronix XPortAR (`http://www.lantronix.com`).

El *software* de desarrollo:

- Microchip MPLAB v8.0 con el plug–in Microchip C30 (`http://www.microchip.com`)..
- NetBeans v6.0 (`http://www.netbeans.org`).
- JAVA SE (`http://java.sun.com/javase/`).

Capítulo 5

Diseño de capas de la unidad controladora

El diseño de un sistema embebido comienza con el diseño de una estructura jerárquica de la aplicación a desarrollar y del *hardware* necesario. El modelo que se utilizó se muestra en la figura 5.1.

Figura 5.1: Capas del sistema embebido.

La capa de *hardware* es la capa física. Para la configuración y control se debe programar directamente los registros de microcontrolador según [17]; estas características

hacen necesario de una capa de *software* que simplifique el uso del *hardware* en funciones o rutinas menos complejas, esta capa es llamada "*drivers*" ó controladores.

La capa RTOS está ubicada parcialmente con la capa *drivers* y la capa *hardware* porque los RTOS tienen servicios que pueden hacer que la capa de drivers funcione eficientemente. Tales servicios son los semáforos, las banderas de eventos, etc; por lo tanto, es responsabilidad del programador que adjunte tales servicios a la capa *drivers* permitiéndole al RTOS la administración completa del *hardware*.

En la capa de aplicación se considera todo lo relacionado con el establecimiento de la sesión con el usuario, esto involucra un correcto manejo de la información y ejecución de instrucciones para la máquina virtual; además esta capa abarca todos los algoritmos y rutinas de control automático y de procesamiento digital de señales.

Todo el *firmware* de la unidad se creó con Microchip C30. Éste es un lenguaje de programación para dsPIC que cumple con el estándar ANSI C. El lenguaje es gratuito (sin niveles de optimización) junto con sus bibliotecas de procesamiento digital, control y matemática.

5.1. Aspectos de la capa física ó *hardware*

Un esquema del *hardware* de toda la unidad se muestra en la figura 5.2. Como unidad central de procesamiento se utilizó el dsPIC33FJ256MC710 previamente descrito. El sistema cuenta con:

- 3 buses de datos: SPI, I2C y uno paralelo de 16 bits (desarrollo parcial) para expansión de *hardware*.
- Puerto de comunicación UART para operación y configuración local de la unidad.
- Puerto de comunicación Ethernet mediante Lantronix XPortAR para operación y configuración remota de la unidad.

- 4 entradas y 2 salidas analógicas de rangos fijos seleccionables.
- Control de motores mediante QEI y PWM.
- Puerto RJ11 para programación del dsPIC mediante MPLAB ICD2.

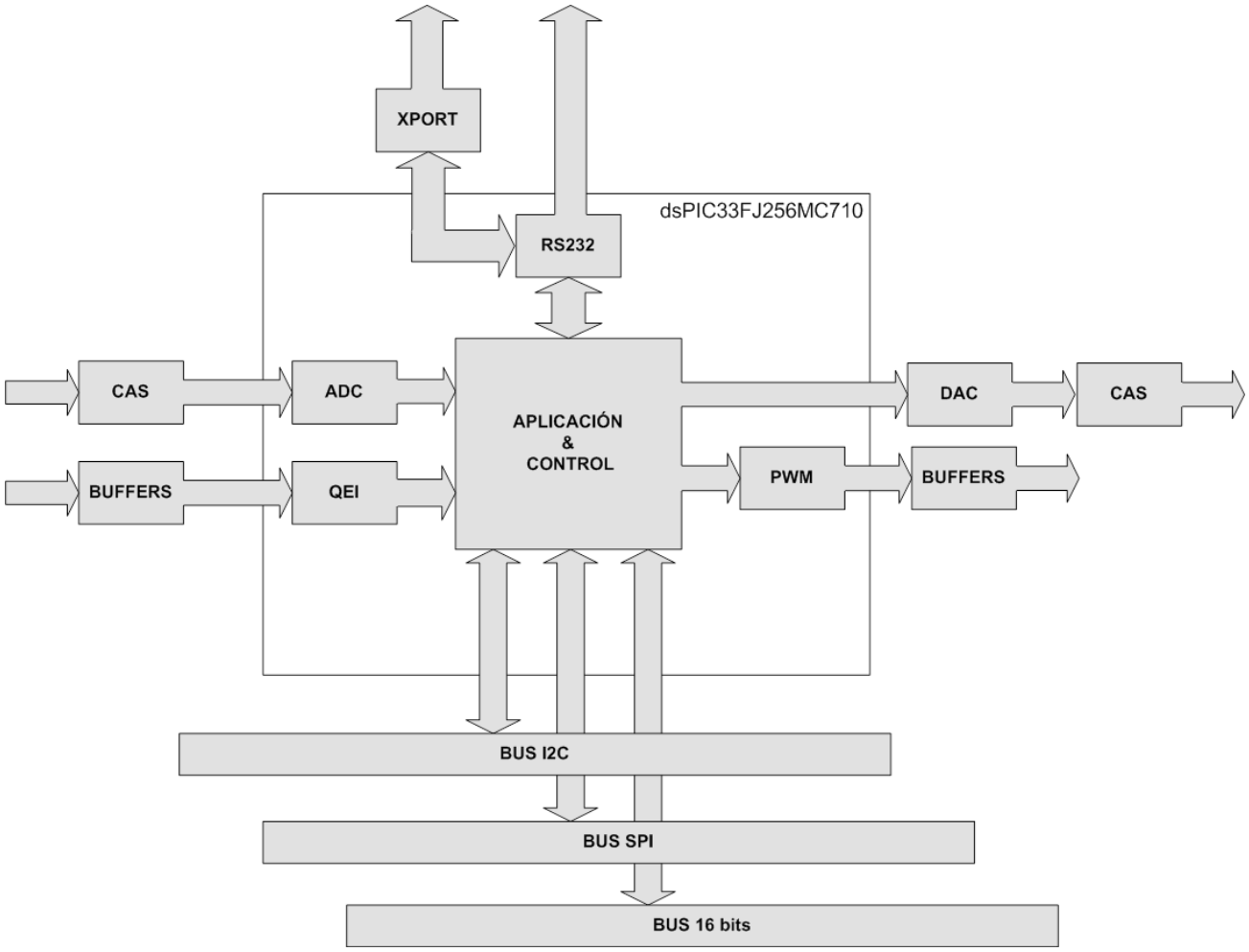


Figura 5.2: Estructura del *hardware* de la unidad.

El apéndice C muestra los diagramas electrónicos de la capa de *hardware*.

5.1.1. Bus de datos SPI

El módulo SPI (interfaz serie para periféricos) es un bus de comunicaciones serie sincrónico que se utiliza para comunicaciones entre circuitos integrados a corta distancia. Estos circuitos integrados son por lo general memorias (RAM), puentes a USB, ADC externos, microcontroladores, sensores digitales, etc. El bus SPI tiene como ventajas: velocidad de comunicación de hasta 20Mbps, no requiere de bits de control dentro de la trama y es posible trasmitir y recibir tramas al mismo tiempo. Como desventaja, se debe asignar una línea de habilitación de dispositivo (CS) por cada dispositivo conectado al bus, esto limita el diseño físico del circuito; además, siempre debe existir un dispositivo maestro (ej: un microcontrolador o un controlador de bus).

La unidad controladora cuenta con 3 pines de bus y 8 pines de selección de dispositivos:

- **SCK**: Salida del reloj de sincronización.
- **SDO**: Salida de datos serie.
- **SDI**: Entrada de datos serie.
- **SSx**: Hasta 8 salidas para selección de dispositivos.

La figura 5.3 muestra un esquema del bus de datos SPI.

5.1.2. Bus de datos I^2C

El módulo I^2C (bus de circuitos inter–integrados) es un bus de datos serie de 2 líneas que permite la conexión de hasta 127 dispositivos en modo simple y hasta 1023 dispositivos en modo extendido. La velocidad de comunicación puede ser de hasta 400Kbps en versiones antiguas; pero, se puede lograr 1Mbps en nuevas versiones limitando el número de dispositivos conectados al bus. La principal ventaja es que solo

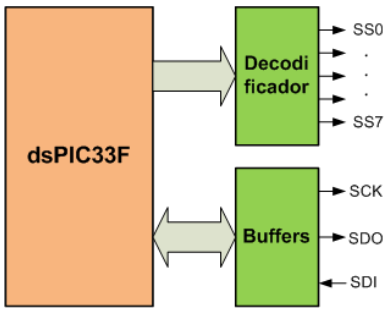


Figura 5.3: Esquema del módulo SPI en la unidad.

se usan 2 líneas de datos sin importar la cantidad de dispositivos conectados; además, se pueden establecer conexiones Maestro – Maestro; sin embargo, estas características acarrean un protocolo de comunicación más complejo, lento y más vulnerable a errores por colisión de datos. Este protocolo es popular en sistemas de pocos pines y en donde la velocidad de transmisión de datos no es importante, ejemplos: memorias (EEPROM), potenciómetros digitales, amplificadores de ganancia programable (PGA), PLL fraccionario, etc.

Los pines del módulo son:

- **SCL**: Salida del reloj de sincronización.
- **SDA**: Pin bidireccional de datos serie.

La figura 5.4 muestra un esquema del bus de datos I^2C.

5.1.3. Bus de datos de 16 bits

De manera opcional se diseñó un bus paralelo de 16 bits similar a la interfaz que presentan los microprocesadores 8086.

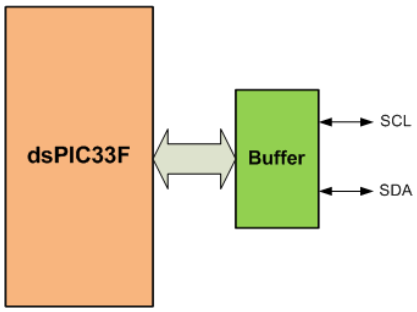


Figura 5.4: Esquema del módulo I^2C en la unidad.

5.1.4. Puerto de comunicación UART

Se habilitó un puerto UART para comunicación local; se usa un cable cruzado RS232 para conexión con la PC. La figura 5.5 muestra un esquema de la conexión.

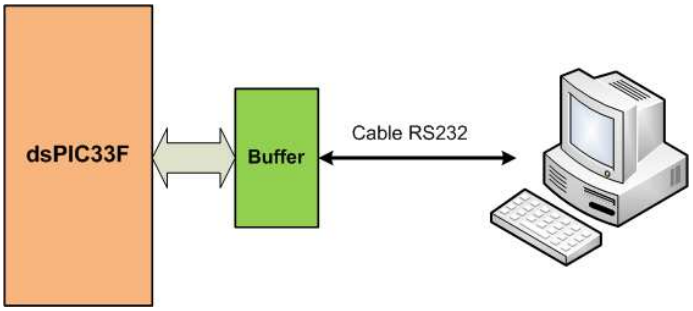


Figura 5.5: Esquema del puerto UART.

5.1.5. Puerto de comunicación *Ethernet*

Para la conectividad Ethernet se utiliza un sistema XPortAR de Lantronix (figura 5.6); este sistema convierte de manera transparente la comunicación serie RS232 a una comunicación *Ethernet*, además integra un servidor *web* que es compatible con aplicaciones JAVA Applets e incorpora un conjunto de métodos para establecer sesiones remotas y funciones: `send, receive, length`, etc.

La figura 5.7 muestra un esquema de la conexión de este dispositivo y la unidad controladora.

Figura 5.6: Sistema Xport AR de Lantronix [15].

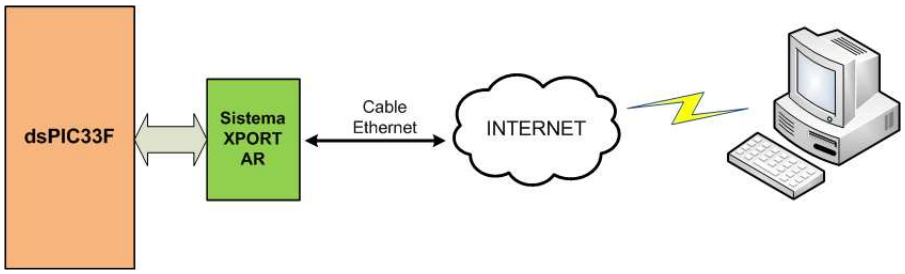


Figura 5.7: Esquema del puerto Ethernet.

5.1.6. Entradas Analógicas

La unidad cuenta con 4 entradas analógicas. Cada entrada analógica puede medir señales de $\pm 50mV$ hasta $\pm 10V$ con una resolución de 12bits, el tiempo de conversión es de $3\mu s$ a $5\mu s$ por entrada, el rango de medición se puede ajustar por *software* y se creó una aplicación de calibración de las entradas. Cada entrada es diferencial y se puede conectar a cualquier tipo de sensor analógico.

El microcontrolador permite un rango máximo de 0 a +3.3V en cada entrada analógica, por lo que se tuvo que crear un circuito acondicionador de señal ajustable para permitir los rangos que muestra la tabla 5.2. Debido al circuito acondicionador, cada entrada tiene una impedancia de $400k\Omega$.

Tabla 5.1: Rangos de medición de cada entrada analógica.

Opción	Rango Bipolar	Rango Unipolar
1	$\pm 10V$	0V – 10V
2	$\pm 5V$	0V – 5V
3	$\pm 2V$	0V – 2V
4	$\pm 1V$	0V – 1V
5	$\pm 500mV$	0V – 500mV
6	$\pm 200mV$	0V – 200mV
7	$\pm 100mV$	0V – 100mV
8	$\pm 50mV$	0V – 50mV

La figura 5.8 muestra un esquema del *hardware* de acondicionamiento, filtro y protección conectados a la unidad controladora.

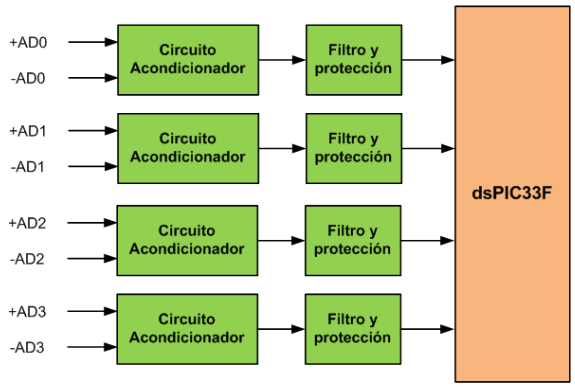


Figura 5.8: Esquema de las entradas analógicas.

5.1.7. Salidas Analógicas

El sistema cuenta con 2 salidas analógicas, cada salida analógica se calibra de la misma manera que las entradas y se pueden seleccionar rangos de operación, los rangos

se muestran en la tabla 5.2.

La figura 5.9 muestra un esquema de las salidas analógicas.

Tabla 5.2: Rangos de cada salida analógica.

Opción	Rango bipolar	Rango unipolar
1	$\pm 10V$	0V – 10V
2	$\pm 5V$	0V – 5V
3	$\pm 2V$	0V – 2V
4	$\pm 1V$	0V – 1V

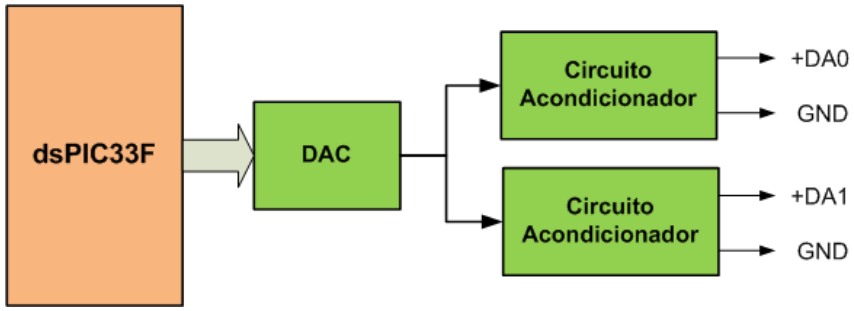


Figura 5.9: Esquema de las salidas analógicas.

5.1.8. Interfaz codificador en cuadratura (QEI)

La interfaz QEI es un circuito especial para medición de posición y velocidad de motores. Este sistema, es el más utilizado a nivel industrial por ser económico y preciso.

Existen varios tipos de decodificadores. La unidad es compatible con codificadores rotatorios incrementales. Este tipo de codificadores tienen por lo general 2 salidas (A y B) que permiten medir velocidad. Si se quiere medir posición se debe utilizar un codificador con 3 salidas (A, B e Index).

La unidad incluye 3 entradas (A, B e Index) con lo cual se puede medir posición y velocidad (figura 5.10).

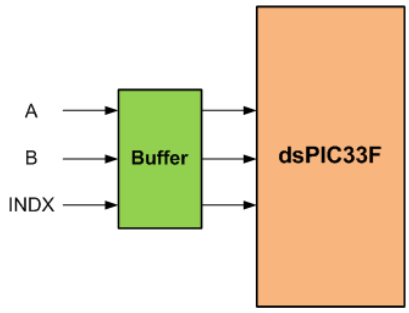


Figura 5.10: Esquema de la QEI.

La velocidad máxima de medición está limitada por el tiempo de muestreo de la unidad (1ms). Con esta restricción se pueden medir velocidades de hasta 30000 rpm; la resolución de medición de posición depende del codificador siempre y cuando el periodo de los pulsos no supere al máximo permitido por el microcontrolador ($300\mu s$).

5.1.9. Salidas de control PWM:

La modulación por ancho de pulsos es una técnica de conversión D/A muy utilizada por su bajo costo de implementación. Por lo general, se utiliza para regulación de velocidad motores CD y motores de inducción trifásicos.

La unidad cuenta con 4 módulos PWM, cada módulo se compone de tres señales:

- **PWMxH**: Salida de modulación por ancho de pulso H.
- **PWMxL**: Salida de modulación por ancho de pulso L.
- **DIR**: Salida de dirección.

Las salidas PWMxH y PWMxL son iguales; sin embargo, la señal de PWMxH puede ser complementaria a la señal de PWMxL; además, se puede seleccionar la función de

cada salida como: PWM, DIR u ON/OFF. Los 4 módulos operaran con un único temporizador. Esto limita el sistema a una frecuencia única para los 4 módulos.

Además se cuenta con 2 entradas FLTx. Estas entradas se pueden utilizar como paro de emergencia del PWM. Ambas señales actúan sobre los cuatro módulos. Las condiciones pueden ser exceso de velocidad, sobrecarga del motor, etc.

La figura 5.11 muestra un esquema de las salidas PWM.

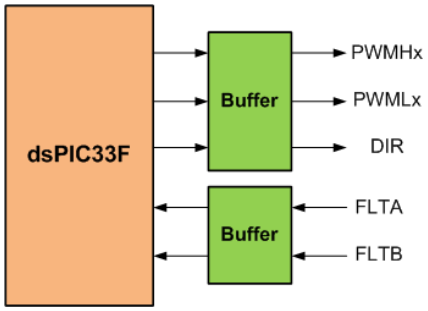


Figura 5.11: Esquema de PWM.

5.2. Aspectos de la capa de controladores

A nivel de sistemas embebidos, es común que el programador seleccione los modos de operación de cada periférico en tiempo de compilación del *firmware*. Para no limitar las características del proyecto, se sugirió crear un método alternativo que permita cambiar la configuración del *hardware* sin necesidad de hacer recompilaciones del *firmware*.

La capa de controladores consiste de:

- **Archivo de configuración:** Este archivo mantiene los parámetros de una configuración funcional, también llamada configuración mínima. Esta configuración es 100 % operativa y se utiliza como configuración de arranque del sistema.
- **Instrucciones de configuración:** Estas instrucciones permiten modificar la

configuración mínima sin necesidad de recompilación del *firmware*. Los parámetros modificados se pueden guardar en memoria y se ejecutan después de inicializada la configuración mínima.

- **Funciones de configuración:** Estas funciones configuran el *hardware* según las opciones del usuario.
- **Funciones de operación:** Este conjunto realiza las distintas operaciones sobre el *hardware* de acuerdo a su función.

La estructura de la capa de controladores se muestra en la figura 5.12; el archivo IO_CFG.H se utiliza como archivo único de configuración mínima y selección de parámetros propios del microcontrolador, los archivos subsecuentes contienen las funciones disponibles para la utilización de cada periférico. Cualquier cambio en el archivo IO_CFG.H solo es considerado en el tiempo de compilación del programa.

Esta configuración mínima solo es necesaria para que el microcontrolador pueda arrancar e inicializar todos los dispositivos y periféricos. A la vez, esta es una configuración conocida de *hardware* funcional, 100 % operativo y se puede utilizar como configuración final del sistema.

Si el usuario necesita otra configuración, ésta se puede realizar mediante instrucciones especiales enviadas por RS–232. La nueva configuración se puede guardar en memoria EEPROM y el microcontrolador la cargará posteriormente [5]. Para ello cada parámetro de configuración está asociado con un registro de instrucción llamado `IOConfig`.

Si la nueva configuración no resulta correcta, se puede omitir dejando presionado el botón ACC [5] durante el arranque. El microcontrolador ignorará la configuración en EEPROM y cargará únicamente la configuración mínima. La secuencia de inicio del sistema se muestra en la figura 5.15.

Las opciones de configuración son propias del microcontrolador dsPIC33FJ256MC710,

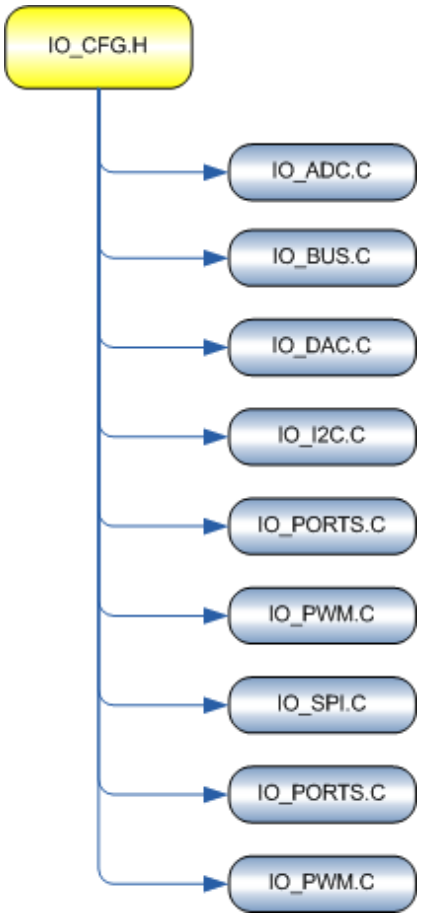


Figura 5.12: Estructura C de la capa drivers.

el manual de referencia [17] de la familia dsPIC33F detalla cada módulo y se recomienda considerarlo antes de realizar cualquier modificación.

5.2.1. Integración del RTOS a la capa de controladores

El RTOS incluye diversas funciones que facilita la administración del *hardware* y optimiza el uso de la CPU. Estas funciones permiten dar prioridad a una tarea cuando se inicie o termine una o varias acciones; un caso muy utilizado, es ceder el procesador a otra tarea cuando se espera mientras algún determinado periférico responda o ejecute

una acción; por ejemplo, la tarea 1 necesita transmitir un *byte* a la PC para continuar, una aplicación sin RTOS normalmente llamaría la función de enviar el *byte*, esperar a que se complete la transmisión y luego continuar con el resto del código; con un RTOS se puede aprovechar ese tiempo de espera para procesar otra tarea mientras la tarea 1 espera el envío del *byte*. El RTOS tiene la función de banderas de eventos (*event flags*), esto se usa en conjunto con las interrupciones de fin de algún proceso en el microcontrolador, tales interrupciones pueden ser:

- ADxIF Fin de conversion ADC.
- UxTXIF Fin de transmisión UART.
- DMAxIF Fin de transferencia de datos por DMA.

El proceso consiste en ceder el procesador después de la orden de inicio de envío, transferencia o conversión en el periférico. El RTOS suspende la tarea 1 y conmuta a la tarea de mayor prioridad (tarea 2).

Cuando la interrupción de fin de proceso se active, ésta levanta la bandera indicándole al RTOS que el proceso concluyó. En este momento el RTOS verifica si la tarea 1 es de mayor prioridad que la tarea 2. Si lo es, el RTOS conmuta a la tarea 1, sino, se marca la tarea 1 como tarea activa y en espera de la CPU.

La figura 5.13 muestra un diagrama de flujo del proceso.

Solamente se integró este mecanismo a la lectura multiple del ADC, envío por UART, todas las operaciones DMA y lectura/escritura en EEPROM. Estos son los periféricos más lentos. Los procesos como el de transmisión/recepción SPI o la lectura simple del ADC, son procesos muy rápidos, donde los tiempos de ejecución son de unos pocos microsegundos y por lo tanto no es conveniente que el RTOS realice algún cambio de contexto. El RTOS debe ejecutar código para realizar la conmutación de tareas y este procedimiento puede llegar a tardar más que la misma función E/S [13].

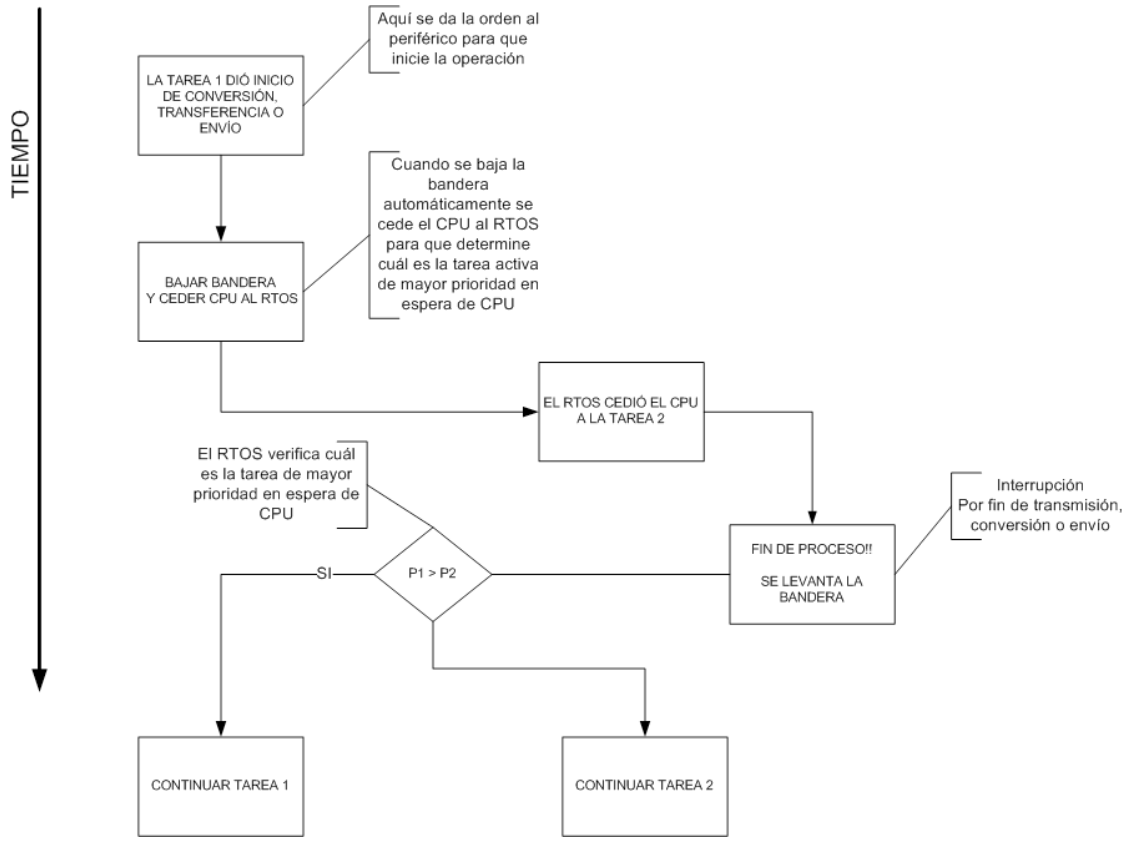


Figura 5.13: Secuencia de ejecución de las banderas de eventos.

5.3. Aspectos de la capa RTOS

Se utilizó la versión 2.85 del *kernel* MicroC/OS–II de Micrium. Este se obtiene gratuitamente del sitio web: `http://www.micrium.com`. La documentación detalla sobre la instalación y uso del kernel MicroC/OS–II se debe consultar a [14] y [5] para la aplicación en este proyecto.

5.4. Aspectos de la capa de aplicación

La figura 5.14 muestra el detalle de las subcapas de funciones y tareas de la capa de aplicación.

Figura 5.14: Subcapas de la capa de aplicación.

La unidad controladora consiste de 6 aplicaciones principales:

- Aplicación de inicio de sistema y diagnóstico.
- Aplicación de comunicaciones.
- Aplicación de máquina virtual.
- Aplicación de captura de datos.
- Aplicación de calibración.
- Compilador e intérprete del archivo *netlist*.

Cada aplicación consiste de un conjunto de tareas y rutinas especiales.

5.4.1. Aplicación de inicio de sistema.

Esta aplicación es la encargada de iniciar el sistema, tal como se resumió en la sección 5.2, el sistema inicia la secuencia de arranque como se muestra en la figura 5.15.

La aplicación de inicio se ejecuta cada vez que se enciende la unidad o después de un reinicio. El primer paso es estabilizar el reloj principal; se utiliza el reloj interno del

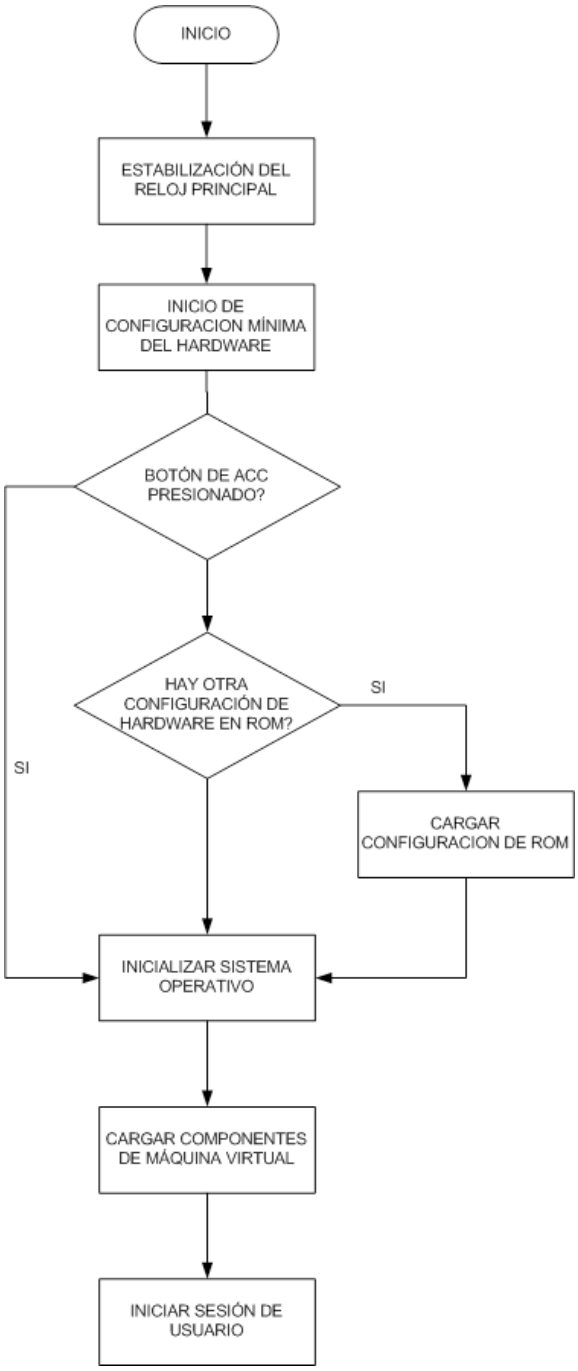


Figura 5.15: Secuencia de inicio del sistema.

microcontrolador el cual utiliza un PLL como multiplicador de frecuencia. En [17] se pueden encontrar más detalles del oscilador.

Después del inicio del PLL, la unidad configura los periféricos de acuerdo con el archivo IO_CFG.H, esta configuración es totalmente operativa y garantiza el funcionamiento completo de los periféricos que en el archivo se establecen. Además, ésta configuración es necesaria para el funcionamiento de los dispositivos externos al microcontrolador.

Si el usuario lo desea, se puede cambiar la configuración de la unidad ya sea cambiando el archivo IO_CFG.H ó enviando una nueva configuración al sistema mediante instrucciones al puerto serie; [5] explica los pasos para cambiar la configuración de los periféricos.

Una vez que la unidad ha leído una configuración de *hardware*, este configura todos los parámetros relacionados con el RTOS (semáforos, banderas, asignación de memoria de las tareas, etc) y la máquina virtual. La configuración de la máquina virtual inicia todos los elementos necesarios para poder establecer 2 sesiones de usuario, cada sesión es independiente una de la otra; sin embargo, los recursos de *hardware* son compartidos. Aunque el RTOS administra el uso del *hardware*, el RTOS no está programado para bloquear el acceso de una sesión a un recurso determinado y que a la vez esté siendo utilizado por otra sesión.

La aplicación de inicio permanece operativa con el fin de diagnosticar errores tales como: desbordamientos de memoria, acceso a espacios de memoria inválidos, desbordamientos del *stack*, errores matemáticos, errores de DMA y falla en el oscilador principal.

5.4.2. Aplicación de comunicaciones.

La aplicación de comunicaciones interpreta y ejecuta todas las instrucciones que el usuario le envía.

La comunicación se establece en un enlace de tres vías. La figura 5.16 muestra un

diagrama de flujo que representa como funciona la aplicación de comunicaciones.

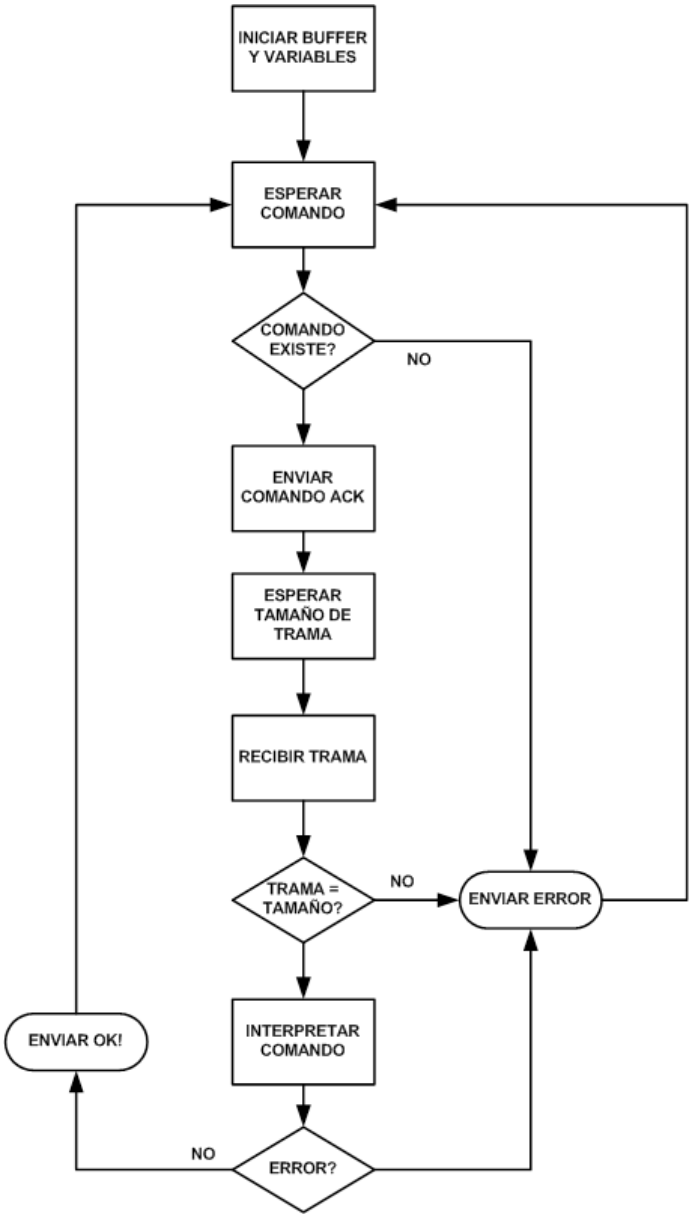


Figura 5.16: Algoritmo de la tarea de comunicación.

La aplicación lee las instrucciones de usuario y ejecuta las acciones requeridas, estas acciones pueden ser: configurar o crear un módulo virtual, ver el estado un una variable,

iniciar otras aplicaciones, enlazar módulos virtuales, etc.

El archivo COMMAND.C conserva todas las rutinas para interpretar las instrucciones. En [5] se detalla la tabla de instrucciones y el formato del protocolo de comunicaciones establecido.

5.4.3. Compilador e intérprete del archivo *netlist*

Esta es una función que se encarga de verificar, interpretar y decodificar el archivo *netlist*; además, crea o destruye en la memoria dinámica todos los módulos virtuales.

El archivo *netlist* es un archivo con todos los enlaces de conexión de los módulos. Cuando el usuario envía el archivo *netlist* primero se verifica que el archivo sea consistente y que no contenga errores de sintaxis. Si el archivo es correcto, se procede a compilar el archivo en un arreglo de punteros a funciones y módulos que posteriormente la tarea de máquina virtual utilizará para ejecutar la estructura de control creada por el usuario.

5.4.4. Aplicación de máquina virtual.

La máquina virtual se encarga de ejecutar todos los bloques virtuales que el usuario haya seleccionado y configurado. Existen 2 máquinas virtuales, una para cada sesión.

Cada máquina virtual se compone de una tarea. Estas tienen la mayor prioridad y ejecutan cada uno de los módulos virtuales enlazados.

Cada máquina lee un archivo propio y temporal que contiene la secuencia de ejecución de los módulos (generado por la función de *netlist*); este archivo se guarda en memoria dinámica. El núcleo de la máquina consiste de un temporizador y un intérprete de punteros a funciones. El capitulo 6 detalla su funcionamiento.

5.4.5. Aplicación de captura y envío de datos.

La aplicación de captura consiste de una tarea y se encarga de almacenar en memoria los resultados de los módulos y registros que el usuario desea procesar en la PC con cualquier *software* de análisis matemático; tales como: MATLAB u Octave. Esta aplicación es útil para tomar datos y posteriormente modelar una planta.

La tarea necesita de tres parámetros para su ejecución:

- Tiempo de submuestreo, en milisegundos.
- Total de tiempo a capturar, en segundos.
- Cadena de los tipos y números de módulos que se desea capturar.

Configurados todos los parámetros, la tarea se ejecuta una vez que la aplicación de máquina virtual esté en funcionamiento. Esta ejecución puede ser de manera sincronizada con la máquina, es decir, cuando la máquina inicia así lo hará la tarea de captura, o, de manera asíncrona, la tarea de captura se ejecuta mediante una instrucción de inicio dada por el usuario tiempo después de que la máquina inició.

Cuando la captura está completa, se envía un mensaje de LISTO, y el usuario puede descargar el archivo para procesarlo en su computadora.

La tarea utiliza una pila de 10 Kbytes máximo de memoria para almacenar los datos; esto limita el tiempo de captura de acuerdo a la cantidad de variables y el tiempo de submuestreo:

$$T_c = \frac{10240}{2 \times n} \times t_{ss} \tag{5.1}$$

Donde:

T_c: Tiempo total de captura.

n: Número de variables.

t_{ss}: Tiempo de submuestreo.

La tarea guarda las variables en la pila y luego las envía. Esto se realiza de esta manera para evitar desbordamientos en la pila de transmisión (128bytes) del puerto de comunicaciones.

Si por alguna razón se excede el uso de memoria, la tarea de comunicación ajusta los parámetros para obtener el máximo tiempo de captura.

Esta tarea también permite enviar una o varias salidas de los módulos virtuales. La tarea busca los resultados de los módulos que el usuario desea observar; la tarea copia los datos y los convierte al formato ASCII. El RTOS determina el momento más adecuado para enviar la trama de datos resultante. Esta función es muy útil para verificar el comportamiento de la estructura de control en el momento de ejecución.

5.4.6. Aplicación de calibración.

Esta aplicación se utiliza para calibrar todos los rangos de las entradas y salidas analógicas, la unidad mantiene en memoria todos los valores de ajuste de los potenciómetros digitales.

Cuando sea necesario calibrar la unidad, el usuario debe contar con señales de referencia y seguir las instrucciones que se detallan en [5].

5.4.7. Aplicación de red multimaestro I^2C

Se desarrolló parcialmente un protocolo de comunicación para expandir las funciones de la unidad. El diseño del protocolo se basa en el protocolo MODBUS para redes industriales.

El propósito del protocolo consiste en utilizar las características del bus I^2C para comunicarse con otros microcontroladores u otras unidades y expandir las funciones de *hardware* de la unidad. No se completó su desarrollo debido a que el microcontrolador

dsPIC33F presenta un error de *hardware* en el bit de detección de colisiones de datos. El detalle del error se explica en [19].

Capítulo 6

Máquina Virtual

La máquina virtual pertenece a la capa de aplicación. Con esta máquina virtual se pueden crear estructuras de control, crear plantas virtuales, etc, utilizando los recursos de *hardware* disponibles.

Cada máquina se compone de un conjunto de módulos con funciones, éstas son:

- Funciones de transferencia de orden 2 por bloque.
- Bloques de realimentación de estado, máximo 4 variables.
- Controladores PID, PI_D y I_PD
- Filtros digitales IIR y FIR.
- Operaciones de suma, resta, multiplicación y ganancia.
- Osciladores senoidales, rectagulares, triangulares, diente de sierra, aleatorios.
- Funciones de *hardware* (ADC, DAC, PWM, etc).

Cada bloque se crea dinámicamente. Estos bloques se pueden conectar virtualmente y crear una estructura de control o simulación. El usuario solo debe indicar cuáles módulos conectar mediante un archivo "*netlist*" y la máquina virtual determinará automáticamente el orden correcto de ejecución de cada módulo.

6.1. Estructura de la máquina virtual

La máquina virtual es una estructura de datos como se muestra a continuación en código ANSI C:

```
typedef struct vm {
     INT8U id;
     INT16U * App_Exe;
     INT16U * App_PC;
     INT16S GP_Reg[ GP_STACK_SIZE ];
     INT16S * GP_Reg_Ptr_In[ GP_STACK_SIZE ];
     INT8U SUM_cont;
     INT8U RES_cont;
     INT8U GAIN_cont;
     INT8U MUL_cont;
     ...
     void * ptr_structs[ VM_TOTAL_STRUCTS ][2];
     void (*VM_Start)( void );
     void (*VM_Stop)( void );
     void (*VM_Pause)( void );
} VM;
```

Detalle de los parámetros:

- `id`: El identificador ID se utiliza para distinguir cual máquina es la que se utiliza, este identificador sirve a la vez para determinar por cuál puerto de comunicaciones se interactúa con el usuario.

- `App_Exe`: Este es un puntero que guarda el archivo de aplicación, este archivo es la secuencia de orden de ejecución de los módulos.

- `App_PC`: Es un puntero auxiliar le indica a la máquina cuál es la siguiente instrucción o módulo a ejecutar.
- `GP_Reg`: Es un arreglo para guardar constantes o almacenar temporalmente valores calculados por lo módulos.
- `GP_Reg_Ptr_In`: Este es un arreglo de punteros a resultados que se desean guardar en GP_Reg, este arreglo es necesario cuando se desea copiar un resultado de un módulo a un registro GP.
- Registros `_cont`: Estos son los contadores de estructuras residentes en memoria dinámica.
- `ptr_structs`: Este es un arreglo de punteros al inicio de las estructuras, los módulos se organizan en subestructuras ordenadas mediante listas enlazadas.
- `VM_Start`: Es el puntero a la función que inicia la máquina.
- `VM_Stop`: Es el puntero a la función que detiene la máquina.
- `VM_Pause`: Es el puntero a la función que pausa la máquina.

6.2. Organización de las estructuras de módulos

Cada módulo virtual está asociado a una estructura de datos. Cada estructura de datos es un espacio en memoria dinámica que mantiene los parámetros del módulo; por ejemplo, para un módulo de filtro digital IIR, se debe mantener un arreglo de constantes, registros de retardo (*delays*), puntero al valor de entrada, registro de salida, parámetros de configuración como: frecuencia de corte y tipo de filtro. Además, si existen varios módulos del mismo tipo, se debe mantener el puntero al módulo anterior y siguiente.

Cada tipo de módulo tiene asignado un valor de tipo (*opcode*); por ejemplo, el ADC tiene asignado el valor `0x01h`, los módulos tipo filtro IIR tiene asignado el valor `0x45h`. Este valor de tipo es necesario para poder identificar cada función disponible en la unidad.

La máquina virtual conoce la ubicación de cada módulo mediante el arreglo de punteros `ptr_structs`. Éste se organiza como lo muestra la figura 6.1.

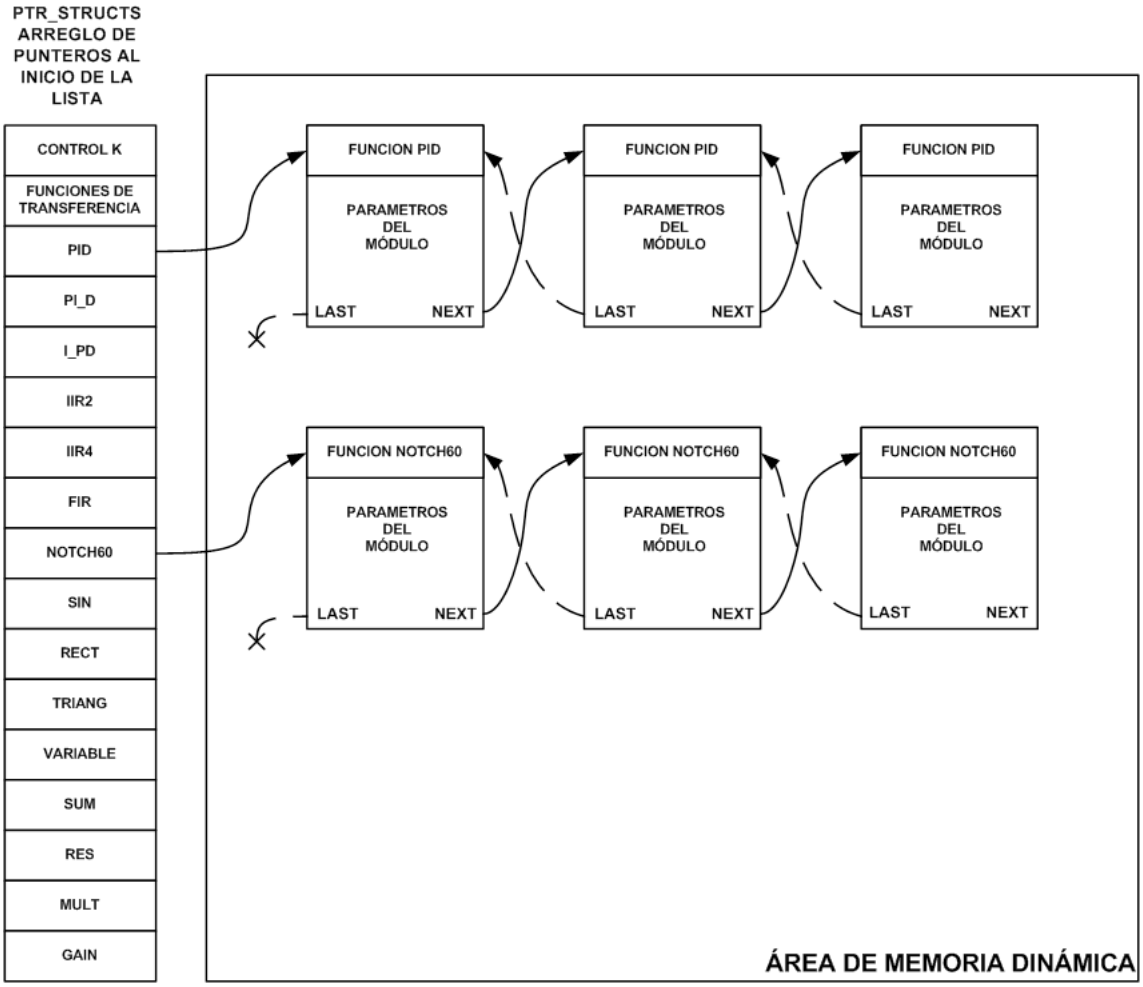


Figura 6.1: Esquema de organización de los módulos virtuales.

Cuando el usuario crea un módulo, éste se enlaza al arreglo `ptr_structs`. Al módulo

se le asigna un número de módulo según la cantidad de saltos que debe realizar la máquina para consultar la estructura, los módulos directamente conectados al arreglo `ptr_structs` tienen el número 0 y así los siguientes con el número 1, 2, 3, etc. El arreglo `ptr_structs` solo considera los tipos de bloques dinámicos; todos los bloques son dinámicos, excepto los registros GP y las funciones de *hardware* como el ADC, PWM, DAC, etc.

Las entradas y salidas de los módulos son numeradas, un módulo con entrada y salida simple tienen la numeración `0x01h` y `0x01h` respectivamente. Los módulos de dos entradas y una salida se numeran `0x01h`, `0x02h` y `0x01h`; en caso de existir módulos con una entrada y dos salidas estas se numeran `0x01h` y `0x01h`,`0x02h`, así sucesivamente.

El archivo SYS_DEF.H define todos los valores tipos de cada función y el archivo SYS_BLOCKS.H muestra todas las estructuras existentes y detalla los parámetros de cada subestructura.

6.3. Organización de enlaces de *NETLIST*

El archivo *NETLIST* es un archivo generado por el usuario en donde indica la conexión de los módulos uno a uno.

La tabla 6.1 muestra un esquema del archivo *netlist*. El archivo se divide en una columna origen y una columna destino, la columna origen se subdivide en el valor de tipo de módulo, el número de módulo y el número de salida; la columna destino detalla el valor de tipo de módulo, el número de módulo y el número de entrada.

Tabla 6.1: Esquema de organización del archivo *netlist*.

Origen			Destino		
Tipo	Número	Salida	Tipo	Número	Entrada

Un módulo se define como una función y un enlace como una unión de módulos. Por ejemplo, la estructura de la figura 6.2 consiste de 6 enlaces y 6 módulos, note que los bloques SUM_0 y PI_D tienen dos entradas.

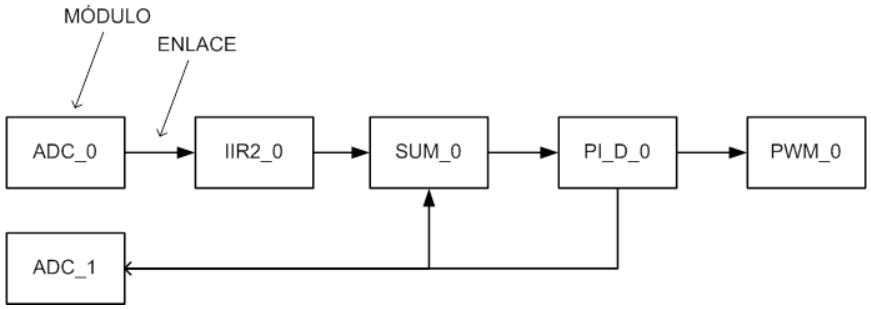


Figura 6.2: Estructura con control con PI_D.

La tabla 1 explica la función de los módulos del ejemplo (el apéndice B muestra la tabla completa de los módulos). El orden en que se coloquen los enlaces no es importante, ya que la unidad utiliza un algoritmo de ordenamiento topológico para determinar la mejor secuencia de ejecución. El usuario solo debe preocuparse por crear un archivo *netlist* con todos los enlaces posibles.

El archivo *netlist* se construye utilizando los valores de tipo de cada módulo, los números de módulo y los números de entrada/salida. La tabla 1 muestra los valores de tipos de los módulos del ejemplo (los valores tipo de todos los módulos se describen en el archivo SYS_DEF.H).

La función IIR2 tiene `0x45h` como valor de tipo; según el ejemplo, solo existe un módulo, por lo tanto, el número de módulo será `0x00h`. También, este módulo solo tiene una entrada y una salida, por lo que ambos utilizan el número `0x01h`; la distinción de entrada o salida se especifica tal como se acomode en el archivo *netlist*.

Entonces, para referirse a la entrada 1 del bloque IIR2 número 0, la secuencia sería: `0x45h, 0x00h, 0x01h`, de la misma forma para referirse a la salida.

La tabla 6.3 muestra el archivo resultante del ejemplo.

Tabla 6.2: Valores de tipo de algunos módulos y mnemónico utilizado.

Valor de tipo	Mnemónico	Función
0x01h	ADC	Tomar muestra analógica
0x49h	SUM	Sumar
0x43h	PI_D	Control PI_D
0x45h	IIR2	Filtro IIR de orden 2
0xC1h	PWM	Salida PWM

Tabla 6.3: Archivo *netlist* de ejemplo.

Origen			Destino		
Valores de tipo	Número	Salida	Valores de tipo	Número	Entrada
0x01h	0x00h	0x01h	0x45h	0x00h	0x01h
0x01h	0x01h	0x01h	0x49h	0x00h	0x02h
0x49h	0x00h	0x01h	0x43h	0x00h	0x01h
0x01h	0x01h	0x01h	0x43h	0x00h	0x02h
0x45h	0x00h	0x01h	0x49h	0x00h	0x01h
0x43h	0x00h	0x01h	0xC1h	0x00h	0x01h

Observe que el orden del archivo no corresponde al orden de ejecución de cada módulo; sin embargo, es importante que el archivo *netlist* contenga los enlaces uno a uno de todos los módulos.

El usuario puede bajar su archivo *netlist* utilizando la instrucción 0x16h (ver [5]).

Esta representación, es la misma cuando se desea leer el resultado de un módulo a través de la aplicación de captura de datos. Supongase que se desea leer el resultado del bloque PI_D; según el apéndice B, el bloque PI_D tiene `0x43h` como valor tipo, como solo existe un módulo, la secuencia para la lectura del resultado del bloque PI_D debe

ser: `0x43h, 0x00h, 0x01h`. La aplicación de captura de datos buscará el registro de resultado del PI_D y lo enviará al usuario o lo almacenará en la pila de captura según sea la opción.

6.4. Funciones de enlace y verificación

Cuando el usuario da la orden de envío de archivo *netlist* la aplicación de comunicaciones lo recibe y llama la función de verificación `App_Verify_Netlist_File`. Luego, se comprueba que el archivo sea consistente y se revisa que todos los módulos existan en memoria dinámica. En caso de error, el archivo *netlist* no se compilará y la ejecución de la máquina virtual no será posible.

Si el archivo es consistente, la unidad utilizará la función `App_Assign_Nets_Out_Ptrs` y enlazará cada módulo destino con su correspondiente módulo fuente, esto es posible porque cada módulo tiene un registro propio donde almacena el resultado de su operación y un puntero al registro resultado del bloque fuente. La figura 6.3 muestra un ejemplo de enlace de bloques.

6.5. Algoritmo de ordenamiento topológico

El algoritmo de ordenamiento topológico se aplica a estructuras de control en donde los módulos que la componen se deben ejecutar en un orden secuencial; en este caso, el algoritmo resuelve el orden correcto de la secuencia ejecutando primero los módulos sin dependencias (entradas) hasta llegar a los módulos raíz (salidas).

Antes de utilizar el algoritmo de ordenamiento topológico, la unidad asignará un código único a cada módulo. Es posible que el orden de ejecución no sea único y pueden existir varias rutas, el algoritmo consiste en recorrer aleatoriamente todo el árbol *netlist*.

El primer paso consiste en buscar un módulo raíz, los módulos raíz se identifican con el valor tipo más alto, ejemplo son los módulos PWM, DAC y GPW (escritura a

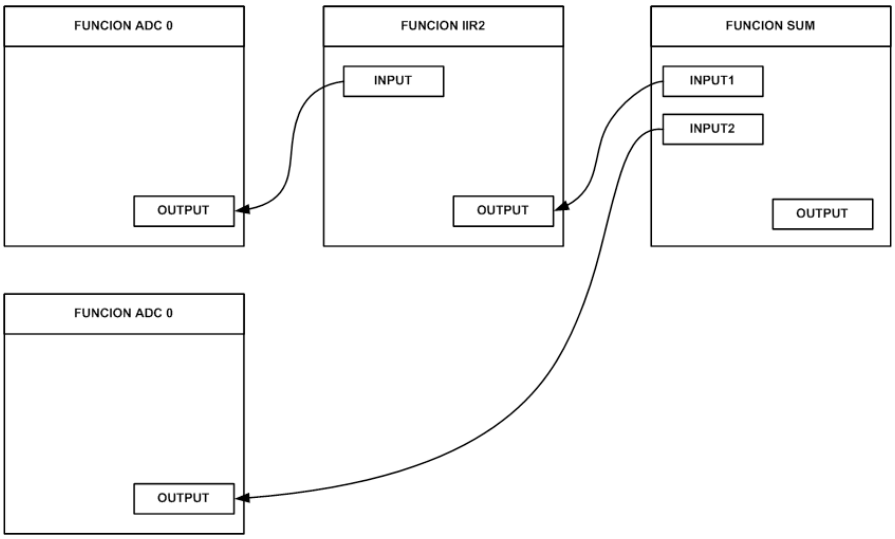


Figura 6.3: Enlace de bloques.

registro de propósito general); si el algoritmo no encuentra ninguno de estos módulos se produce un error y la unidad no está lista para ejecutar una estructura. Si el algoritmo encuentra un módulo raíz, el algoritmo recorre toda la estructura en busca de módulos no dependientes o sin fuente, ejemplos iniciales son los módulos ADC y GPR (lectura en registro de propósito general). Cuando se encuentra un módulo no dependiente, éste se marca y se coloca en una pila temporal; luego, se elimina el módulo de la columna fuente del archivo *netlist*. Éste último paso, produce que módulos dependendientes del módulo eliminado sean independientes.

El algoritmo recorre nuevamente la estructura en busca de nuevos módulos independientes; si encuentra uno, se procede a colocarlos en la pila y se repite el proceso. Esta secuencia se ejecuta varias veces hasta dejar el módulo raíz como módulo no dependiente. Por último, el algoritmo vuelve a revisar el *netlist* en busca de nuevos módulos raíz. Obsérvese que la existencia de varios módulos raíz no afecta el algoritmo, y siempre se ejecutarán primero todas las dependencias hasta llegar a los módulos raíz.

Cuando el algoritmo finaliza, la unidad crea un archivo de instrucciones para la

ejecución de la máquina virtual según el orden establecido en la pila. Cada una de estas instrucciones, se compone de un código de operación y un operando. El formato de la instrucción se muestra en la tabla 6.4.

Tabla 6.4: Formato de las instrucciones para la máquina virtual.

8 bits	16 bits
Valor de tipo (Código de operación)	Puntero a la estructura (Dirección de memoria)

El código de operación es igual al valor de tipo del módulo. Éste código de operación se utiliza para indicarle a la máquina cuál función debe ejecutar. Para módulos dinámicos, el operando es un puntero a la estructura del módulo que contiene la información y parámetros. Para módulos de *hardware*, el operando es el número de módulo. En total, se deben tener tantos códigos de operación como módulos virtuales de la estructura de control. La lista de códigos de operación se guarda en una sección de memoria apuntada por `App_Exe`. La tabla 6.5 muestra el archivo generado por el algoritmo de ordenamiento topológico.

Tabla 6.5: Formato de las instrucciones para la máquina virtual.

Módulo	Valor de tipo	Puntero a la estructura
ADC_0	0x01h	0x0000h
ADC_1	0x01h	0x0001h
IIR2_0	0x45h	0x3C42h
SUM_0	0x49h	0x3974h
PI_D_0	0x43h	0x36A8h
PWM_0	0xC1h	0x0000h

6.6. Ejecución de la máquina virtual

La máquina virtual se puede clasificar como una máquina Eckert-Mauchly [26]. Éste tipo de máquina realiza secuencialmente los siguientes pasos:

1. La máquina inicia reestableciendo un contador a cero o con una posición de memoria inicial.

2. Se lee la instrucción en la memoria de programa apuntada por el contador y se guarda en un registro temporal.

3. La máquina decodifica la instrucción y coordina todos los procesos relacionados con la instrucción.

4. Se ejecuta la instrucción.

5. Se aumenta el contador.

6. Se vuelve al paso 2 para interpretar la siguiente instrucción.

La máquina virtual se encarga de interpretar y ejecutar todos los códigos de operación apuntados por `App_Exe`. Cuando la máquina inicia, toma el valor de `App_Exe` y lo copia al puntero auxiliar `App_PC`; se lee el primer el código de operación y se busca la función en el arreglo `App_Func_Ptrs[ ]`, éste es un arreglo de punteros a funciones en donde el valor tipo obtenido del código de operación corresponde al valor índice del arreglo. La máquina llama la función correspondiente y utiliza el operando para leer la información de subestructura del módulo.

Una vez finalizada la función, se incrementa `App_PC` para ejecutar el siguiente módulo. La figura 6.4 muestra un esquema de cómo se ordena el archivo `App_Exe`; la figura 6.5 muestra el arreglo de punteros a funciones `App_Func_Ptrs[ ]`.

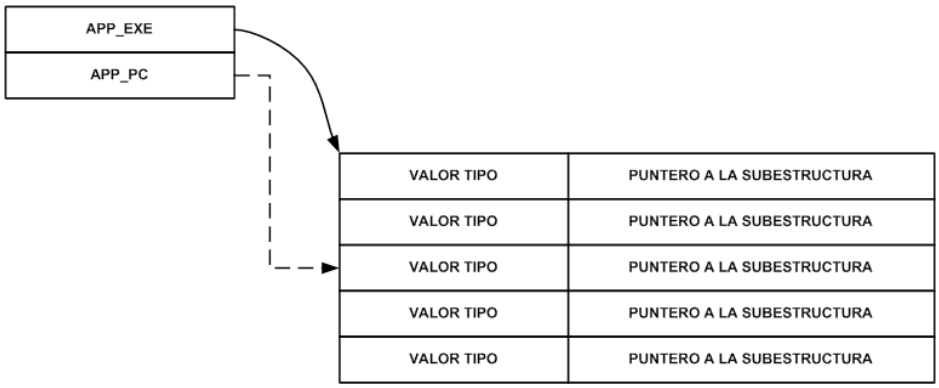


Figura 6.4: Punteros a los códigos de operación.

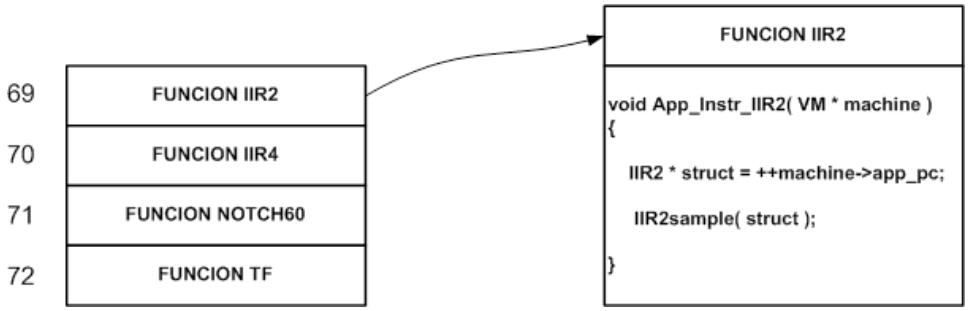


Figura 6.5: Punteros a las funciones.

6.7. Aritmética y normalización

La arquitectura de la familia dsPIC33F es de 16 bits y está optimizada para operaciones DSP; aunque el microcontrolador puede ejecutar operaciones en punto flotante de 32 bits, éstas se realizan por *software* produciendo tiempos de cálculo de $10\mu s$ a $20\mu s$ para multiplicaciones; sin embargo, el microcontrolador puede ejecutar operaciones en punto fijo de 16 bits ($100ns$ para multiplicaciones); además, el compilador MPLAB C30 incluye una biblioteca sobre funciones DSP y operaciones vectoriales [18].

La aritmética utilizada por esta biblioteca se conoce como formato fraccional 1.15 ó Q15 que representa números desde -1.0 hasta 0.999969821422 con una resolución de:

$$R = 2^{-n} = 2^{-15} = 3{,}05176 \times 10^{-5} \tag{6.1}$$

La aritmética en punto fijo es muy sensible a desbordamientos y la representación Q15 no es útil, pues si alguna señal o cálculo intermedio presenta algún sobreimpulso el cálculo sería erróneo, para evitar esto se seleccionó la representación Q2.14.

$$R = 2^{-n} = 2^{-14} = 6{,}103516 \times 10^{-5} \tag{6.2}$$

Esto permite representar números desde -2 hasta 1.99993896484. Con lo que se deja un margen de seguridad de aproximadamente una unidad para sobreimpulsos u operaciones fuera de rango. La representación Q2.14 es la configuración por defecto, sin embargo se puede cambiar a Q1.15, Q3.13, Q4.12 ó Q5.11. Esto, se hace en tiempo de compilación, pues se deben reconstruir las rutinas de la unidad que realizan operaciones matemáticas.

Se estableció que los módulos de *hardware* solo pueden representar valores dentro del rango de -1.0 a 1.0. Los módulos de entrada no devuelven valores fuera del rango propuesto. Los módulos de salida saturan automáticamente a 1.0 ó -1.0 cualquier valor mayor a 1.0 ó menor a -1.0.

Aunque el tiempo de ejecución de las operaciones en punto fijo es mucho menor que las operaciones en punto flotante, el método propuesto tiene la desventaja de que el usuario debe de considerar dentro de sus diseños la normalización; por ejemplo, si una entrada analógica está configurada para operar entre $\pm 10V$, la unidad representará el valor de $10V$ como 1.0 y el valor de $-10V$ como -1.0, el factor de normalización de entrada en este caso es de 0.1. De igual manera, una salida analógica ajustada en $\pm 5V$ tendrá un factor de normalización de salida de 5. El factor de normalización de entrada multiplicado por el factor de normalización de salida es igual a la ganancia inherente dentro de la estructura del sistema virtual. Para este ejemplo, el factor de normalización sería de 0.5.

6.8. Módulos y parámetros

Esta sección presenta las funciones de procesamiento digital, control y operaciones que se pueden realizar en la unidad. Cada módulo está compuesto de un conjunto de parámetros que caracterizan su funcionamiento.

Todos los valores numéricos de cada módulo deben estar normalizados y no deben ser superiores a la aritmética que utilice la unidad.

6.8.1. Módulo Filtro IIR2 y IIR4

Con este módulo se pueden crear filtros digitales de orden 2 y orden 4, las opciones de configuración son:

- Tipo de filtro:
    1. Butterworth Paso Bajos
    2. Butterworth Paso Altos
    3. Butterworth Pasa Banda
    4. Butterworth Rechaza Banda
    5. Bessel Paso Bajos
    6. Bessel Paso Altos
- Frecuencia de corte o central
- Ancho de banda.

La unidad emplea una algoritmo que calcula los coeficientes para el filtro. El algoritmo se basa de la conversión de filtros analógicos a digitales mediante la transformación bilineal; además, utiliza las funciones normalizadas (analógicas) para cada tipo y orden de filtro.

La máquina virtual emplea el tiempo de muestreo base (1ms) como tiempo de muestreo para los filtros, este tiempo de muestreo es constante y no se puede cambiar.

Para protección anti-alias se crearon filtros pasabajas en 138Hz tipo Bessel en las entradas analógicas.

6.8.2. Módulo NOTCH60

Este es un módulo con la función de filtro muesca a 60Hz. La función se implementó con la técnica de introducción de ceros conjugados en el círculo unitario [22].

Como opción de filtro se puede cambiar el radio de ubicación de los polos conjugados. La configuración por defecto es de 0.9.

Para verificar el funcionamiento del filtro se seleccionó una señal rectangular de 20Hz. Este tipo de señal periódica se compone de armónicas impares a 20Hz, 60Hz, 100Hz, etc. En la figura 6.6 se observa la señal rectangular de prueba (celeste), en la parte superior a esta se capturó la transformada rápida de Fourier (FFT) de la señal rectangular. Ésta representa las primeras 10 armónicas de mayor potencia. La segunda armónica corresponde a la armónica en 60Hz aproximadamente. Se utiliza el valor por defecto de la ubicación de los polos (0.9), la señal de color amarillo es la salida resultante del filtro, nuevamente se utilizó la FFT verificar el espectro de la señal de salida (color rojo). La atenuación de la armónica en 60Hz es de aproximadamente 23,5dB.

Se realizó el mismo experimento con la misma señal utilizando $r = 0{,}85$, la figura 6.7 muestra el resultado. La atenuación es de 17,5dB.

6.8.3. Módulo FIR

Este módulo implementa las funciones de filtro tipo FIR. No se desarrolló ningún algoritmo de cálculo de coeficientes, sino, se tiene un arreglo de coeficientes para una serie de frecuencias de corte precalculadas, a 100, 50, 20 y 10Hz.

La característica deseable de este tipo de filtro es la fase lineal y la estabilidad; sin

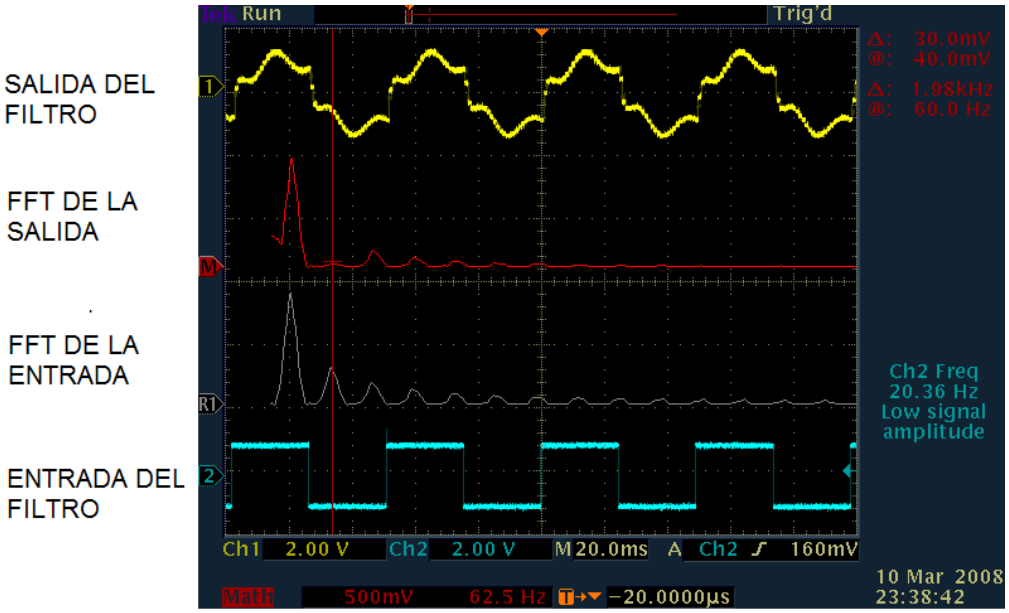


Figura 6.6: Utilización del filtro NOTCH60 con r = 0.9.

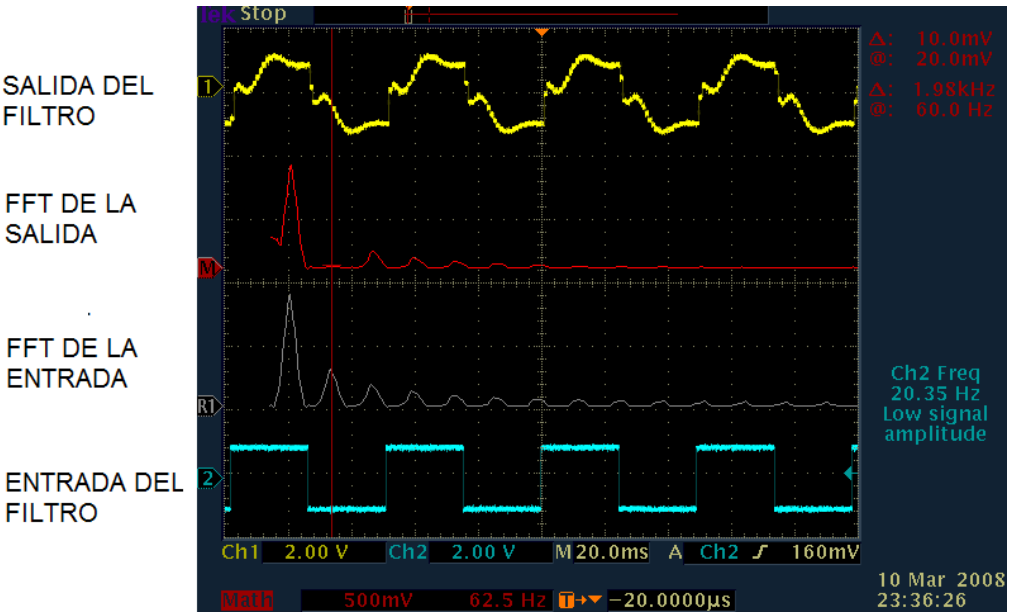


Figura 6.7: Utilización del filtro NOTCH60 con r = 0.85.

embargo, la desventaja es que necesitan de un mayor orden para cumplir las mismas características que un filtro IIR. Esto acarrea una mayor utilización de la memoria RAM ($128 bytes$ contra $20 bytes$ que utiliza un filtro IIR de orden 2) y un aumento del retardo de la señal en la salida del filtro. La figura 6.8 muestra la respuesta en el tiempo de un filtro FIR ante un escalón, el filtro FIR cumple con las siguientes características:

- FIR pasobajos de 32 etapas
- Ventana: Kaiser
- Frecuencia de muestreo: 1kHz
- Banda de paso: 100Hz
- Banda de rechazo (*stop band*) : 150Hz

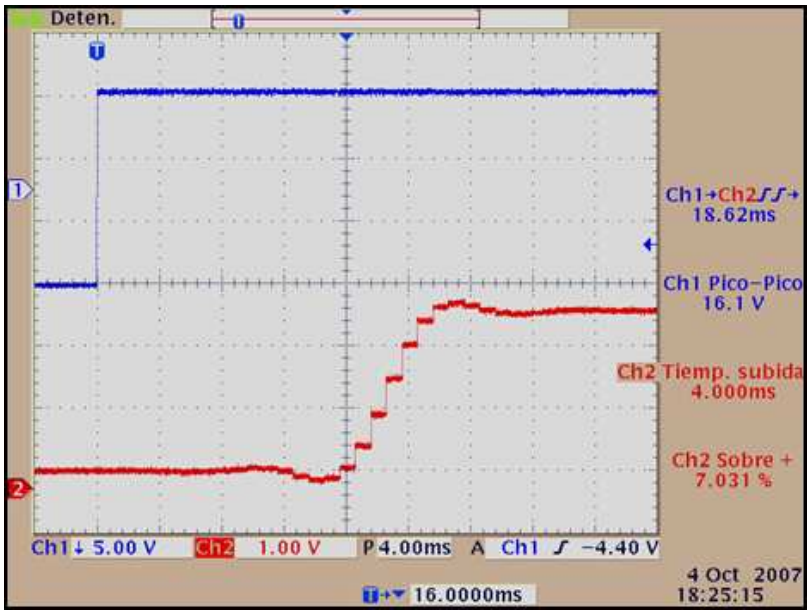


Figura 6.8: Respuesta ante un escalón de un filtro FIR de 32 etapas.

De la figura 6.8 se observa un retardo de 24ms.

6.8.4. Módulo TF

El usuario puede crear sus propios filtros y compensadores mediante las funciones TF (funciones de transferencia), cada módulo TF se puede configurar con un tiempo de submuestreo y los parámetros de la función como polos, ceros y ganancia:

$$H(z) = G\frac{(z - z_1)(z - z_2)}{(z - p_1)(z - p_2)} \tag{6.3}$$

o en formato de coeficientes b y a:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{6.4}$$

Cada bloque TF es de orden 2 (funciones de orden 1 son posibles haciendo p_2 y z_2 iguales a cero).

6.8.5. Módulo PID, PI_D e I_PD

Estos bloques implementan las funciones de control PID paralela, PI_D y I_PD tal como se muestra en las figuras 6.9, 6.10 y 6.11:

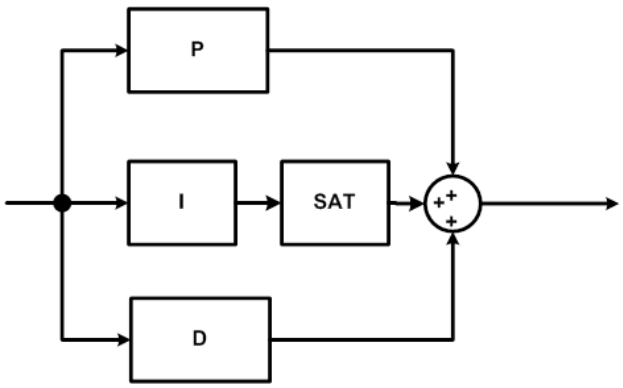


Figura 6.9: Función PID paralela.

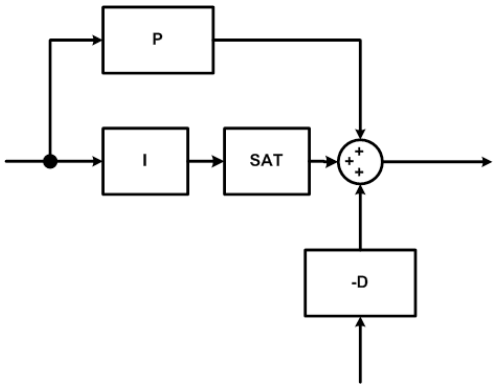


Figura 6.10: Función PI_D.

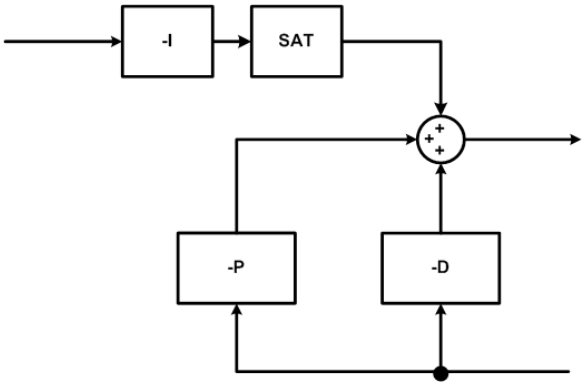


Figura 6.11: Función I_PD.

El usuario especifica las constantes P, I y D, en el formato númerico configurado (Q14 por defecto) y el tiempo de submuestreo en milisegundos.

6.8.6. Módulo SUM, RES, MUL, GAIN

Estas son funciones de operación: suma, resta, multiplicación y ganancia. Para el funcionamiento de cada módulo no es necesario ningún parámetro excepto el tipo de módulo GAIN que requiere el valor de ganancia.

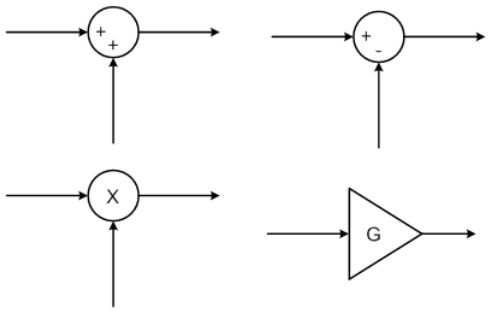


Figura 6.12: Módulos de operadores matemáticos.

6.8.7. Módulo realimentación de estado K

Esta función realiza una multiplicación vectorial de una matriz de ganancia de realimentación de estado de tamaño $1 \times n$, donde n tiene tamaño máximo 4.

El usuario debe especificar el valor n y todas las constantes de ganancia que requiera. Obsérvese que n también es la cantidad de entradas, por lo que deben estar señaladas dentro del archivo *netlist*.

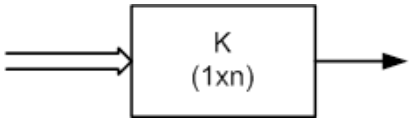


Figura 6.13: Función de realimentación de estado.

6.8.8. Módulo oscilador senoidal

Esta función permite generar ondas tipo senoidal, la máxima frecuencia es de 200Hz. Como parámetros del módulo se deben especificar la frecuencia de oscilación y la amplitud de la señal.

6.8.9. Módulo oscilador con ancho de pulso constante

Esta función genera señales pulsantes y rectangulares ($t_1 = 0$ y $t_3 = 0$), los parámetros se especifican en tiempos de duración en milisegundos, tal como lo muestra la figura 6.14

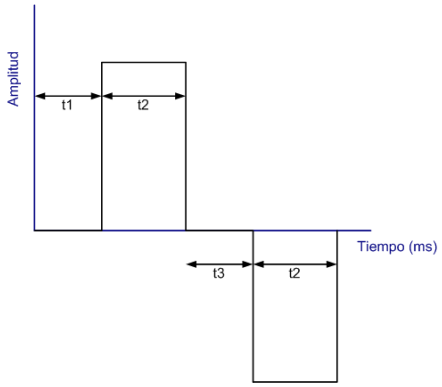


Figura 6.14: Función de oscilador de pulsos constantes.

El ancho de tiempo del pulso positivo y negativo son iguales. El usuario debe especificar, t1, t2, t3, la amplitud positiva y la amplitud negativa.

6.8.10. Módulo oscilador con ancho de pulso variable

Esta función es similar al oscilador con ancho de pulso constante, en este módulo se debe configurar el tiempo mínimo y máximo del ancho de los pulsos; además, se debe configurar la amplitud positiva y la amplitud negativa. La función generará de manera aleatoria los anchos de los pulsos negativos y positivos.

6.8.11. Módulo oscilador triangular

Esta función genera señales triangulares y dientes de sierra($t_1 = 0$ ó $t_2 = 0$). Al igual que los módulos osciladores de pulsos los parámetros se especifican en tiempos, como lo muestra la figura 6.16

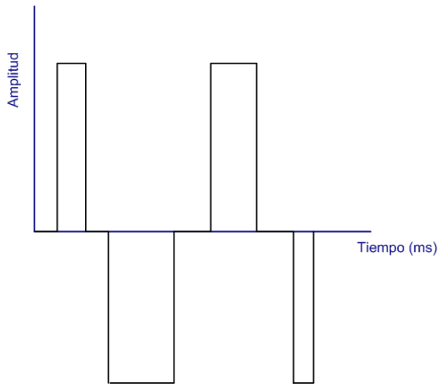


Figura 6.15: Función de oscilador de pulsos variables.

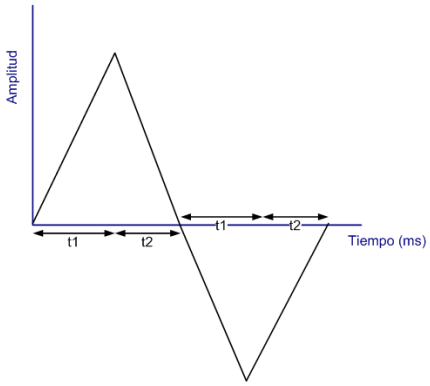


Figura 6.16: Función de triangular.

Se deben especificar, t1, t2, la amplitud positiva y la amplitud negativa.

6.8.12. Módulo oscilador aleatorio

Esta función genera una señal ruidosa. El parámetro de configuración es la amplitud absoluta de la señal de ruido.

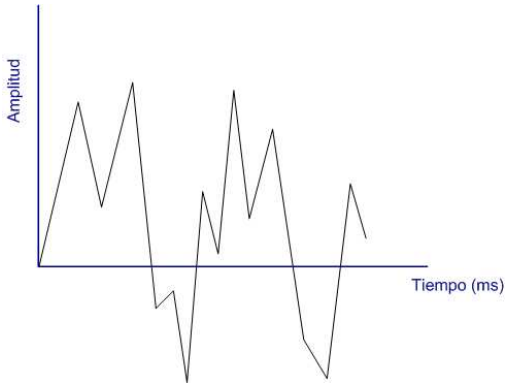


Figura 6.17: Función de oscilador aleatorio.

Capítulo 7

Resultados y demostraciones

7.1. Estabilización de la planta *Ball & Beam*

Un *Ball & Beam* es un sistema que se compone de una barra, un motor y una bola. La figura 7.1 muestra un diagrama simplificado.

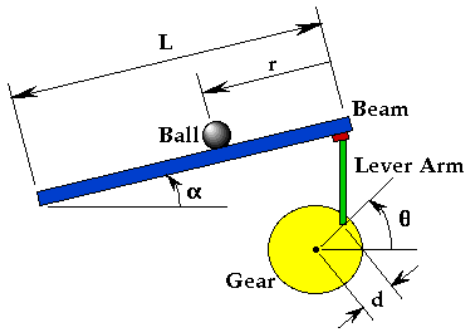


Figura 7.1: Diagrama simplificado de un *Ball & Beam*[25].

La bola se coloca encima de la barra y esta se mueve de un lado o hacia el otro de acuerdo al ángulo de inclinación de la barra. Éste comportamiento caracteriza al *Ball & Beam* como un sistema inestable a lazo abierto.

Ésta sección muestra cómo se logró estabilizar al *Ball & Beam* utilizando las funcio-

nes de captura de datos, control PID y el programa de análisis matemático MATLAB.

7.1.1. Diagrama de la planta

La figura 7.2 muestra una fotografía del *Ball & Beam*. La planta utiliza un potenciómetro como sensor de inclinación de la barra. Este mide un rango de inclinación de aproximadamente $\pm 5°$. Los ángulos negativos se obtienen cuando la barra se inclina hacia el lado izquierdo tal y como se muestra en la figura 7.3.

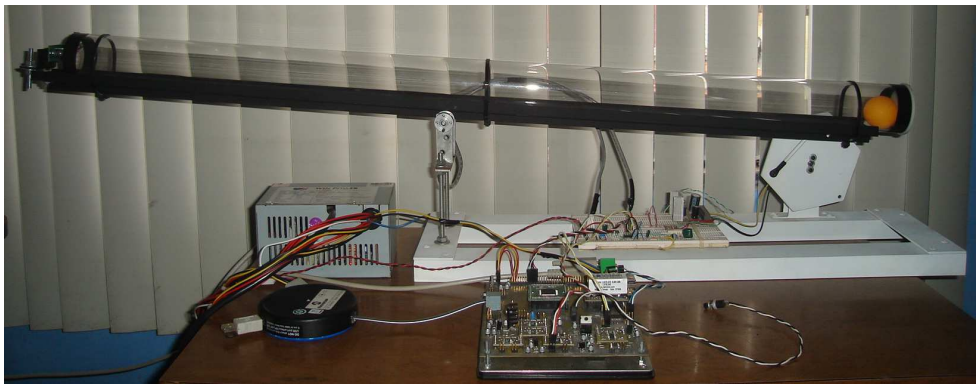


Figura 7.2: *Ball & Beam* utilizado para verificar el funcionamiento de la unidad controladora.

Se utilizó un sensor de distancia por ultrasonidos SRF08 para medir la posición de la pelota. Si la barra se inclina con un ángulo negativo, la pelota se desplazará hacia el lado izquierdo de la barra; en este caso, se obtendrá una medición negativa de la pelota.

Diagrama de bloques

El diagrama de bloques de un *Ball & Beam* se estructura cómo se muestra en la figura 7.4

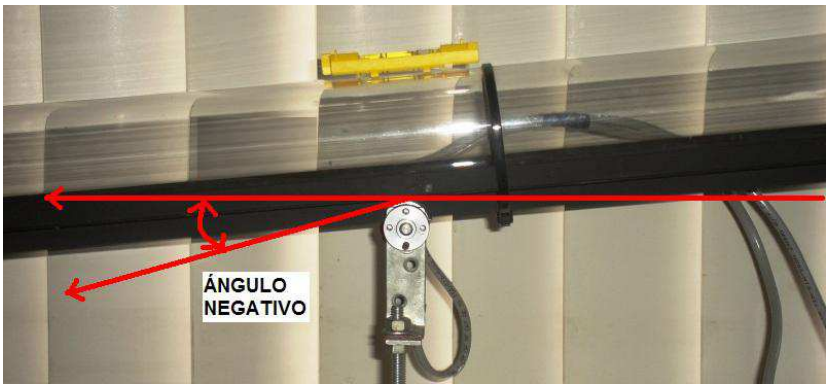


Figura 7.3: Posición de la barra cuando se obtiene una medición negativa en el ángulo de inclinación.

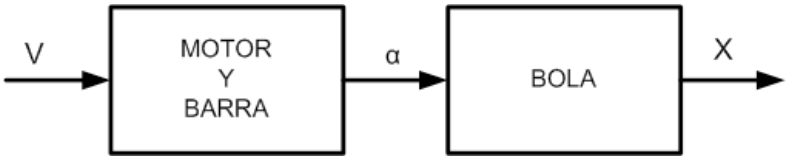


Figura 7.4: Estructura del *Ball & Beam* a lazo abierto.

Es por medio del motor, que se varía el ángulo α de la barra (figura 7.1). Según este ángulo de inclinación, la bola se tendrá una aceleración dependiente de α .

Para estabilizar la bola, se debe realimentar el sistema como se muestra en la figura 7.5.

El objetivo del control del motor (servo) es estabilizar la posición angular de la barra. El control de la bola permite cambiar la inclinación de la barra para poder estabilizar la bola en el centro de la misma.

7.1.2. Modelo matemático del sistema MOTOR – BARRA

Para modelar el motor, se utilizaron las señales de estímulo y la funciones de captura de la unidad controladora.

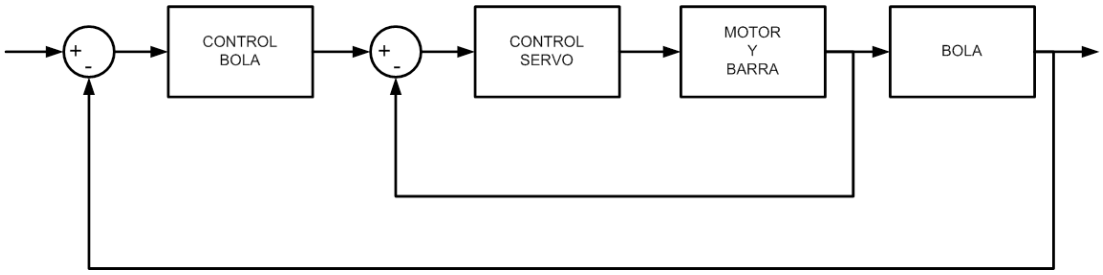


Figura 7.5: Estructura del *Ball & Beam* a lazo cerrado.

El motor no se puede modelar a lazo abierto debido a la configuración mecánica de la planta. La figura 7.6 muestra los dos rangos de giro del motor para que la variación del ángulo de la barra sea lineal.

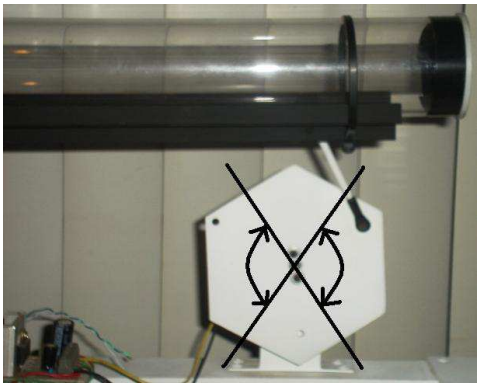


Figura 7.6: Límites de operación del movimiento angular del motor.

La figura 7.7 muestra cómo varía el ángulo α de la barra respecto del movimiento angular θ del motor cuando se opera a lazo abierto.

Se debe asegurar que la barra se mantenga dentro del margen de $\pm 5°$. Si no se elige una señal de estímulo adecuada, se presenta la posibilidad de que la barra se ubique fuera de este margen; obteniéndose así, un modelo erróneo.

Otra forma de estimular el motor, es con el lazo cerrado. La figura 7.8 muestra el esquema.

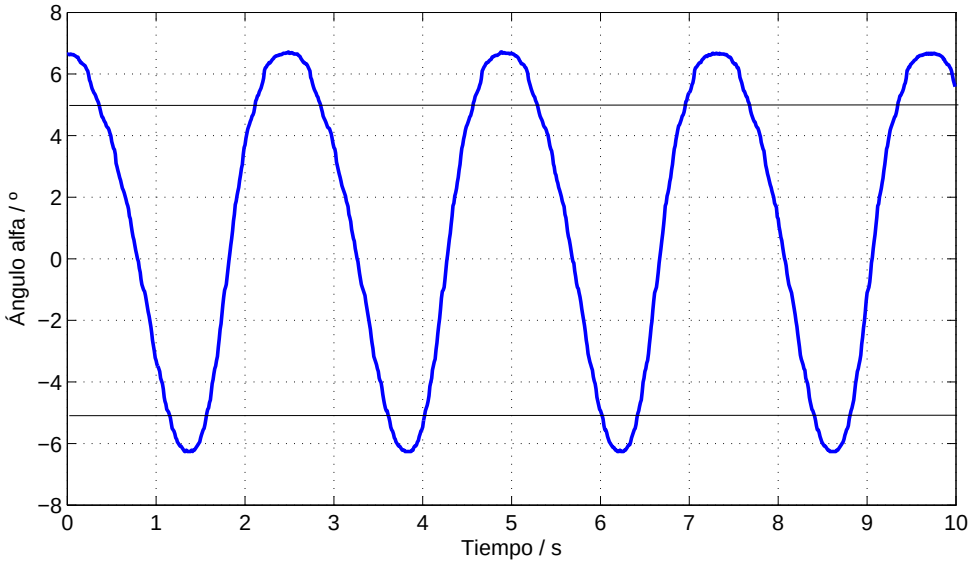


Figura 7.7: Respuesta del movimiento angular de la barra respecto a la rotación del motor.

De esta manera se logra estabilizar el sistema MOTOR – BARRA y se evita que la barra se ubique fuera del margen lineal. En este caso, la amplitud de la señal de estimulo serán el margen de operación del movimiento angular de la barra.

Proceso de captura de datos

La estructura de control utilizada en la unidad se muestra en la figura 7.9.

El ADC2 mide la posición de la barra. Se utilizó una señal de pulsos de ancho variable (VOSC) con una amplitud de $\pm 5°$ como señal de estímulo. El ancho de los pulsos es aleatorio con un mínimo de 0.2 segundos y un máximo de 1.5 segundos. Se capturó a una tasa de muestreo de 10ms durante 10 segundos obteniéndose un total de 1000 muestras.

Se utilizó la herramienta IDENT de MATLAB para modelar el sistema. La tabla 7.1 muestra los resultados de los métodos de estimación con mejor ajuste.

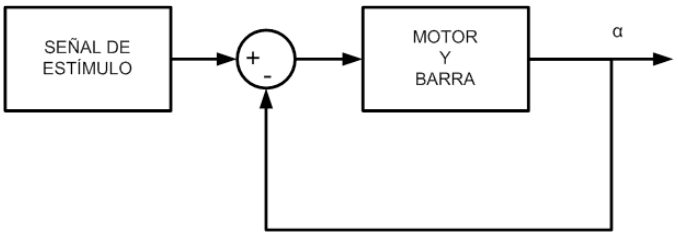


Figura 7.8: Diagrama del motor a lazo cerrado.

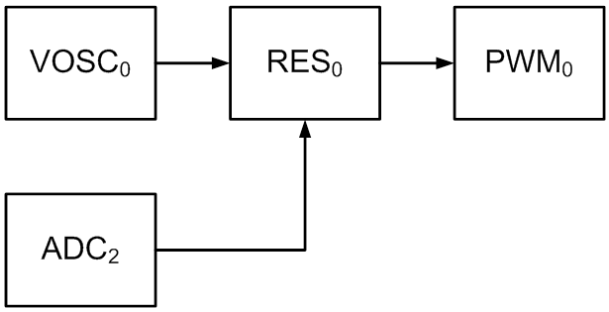


Figura 7.9: Estructura de control para estimular el sistema MOTOR – BARRA.

El modelo con mejor ajuste fue ARMAX de orden 2 con un 86,73 %. La figura 7.10 muestra una comparación del resultado de la estimación matemática y la respuesta real del sistema MOTOR – BARRA a lazo cerrado.

El modelo obtenido es:

$$G(z) = \frac{-0{,}0070971(z - 1{,}713)}{z^2 - 1{,}873z + 0{,}8781} \tag{7.1}$$

Pero éste es el modelo del sistema a lazo cerrado. Para obtener el modelo del sistema MOTOR – BARRA se debe despejar $H(z)$ de la siguiente ecuación.

$$G(z) = \frac{H(z)}{1 + H(z)} \tag{7.2}$$

El modelo matemático del sistema MOTOR – BARRA es entonces,

Tabla 7.1: Porcentaje de ajuste de los distintos métodos de modelado.

Método	Orden	Ajuste / %
ARMAX	2	86.73
BJ	2	86.23
SS	2	86.44
ARX	2	85.59

$$H(z) = \frac{-0{,}0070971(z - 1{,}713)}{(z-1)(z-0{,}8664)} \tag{7.3}$$

7.1.3. Modelo matemático del sistema BOLA

En este sistema, la aceleración de la bola depende directamente de la aceleración de la barra. La figura 7.11 muestra la fuerza resultante que provoca el rodamiento de la bola.

Matemáticamente se tiene que:

$$\sum F = mg \operatorname{sen} \alpha - f \tag{7.4}$$

$$ma_{CM} = mg \operatorname{sen} \alpha - f \tag{7.5}$$

La fuerza de fricción produce un momento de torción sobre la bola igual a

$$\tau = fR \tag{7.6}$$

$$fR = I_{CM} a \tag{7.7}$$

Puesto que $I_{CM} = \frac{2}{5} mR^2$ y $a = a_{CM}/R$, se obtiene

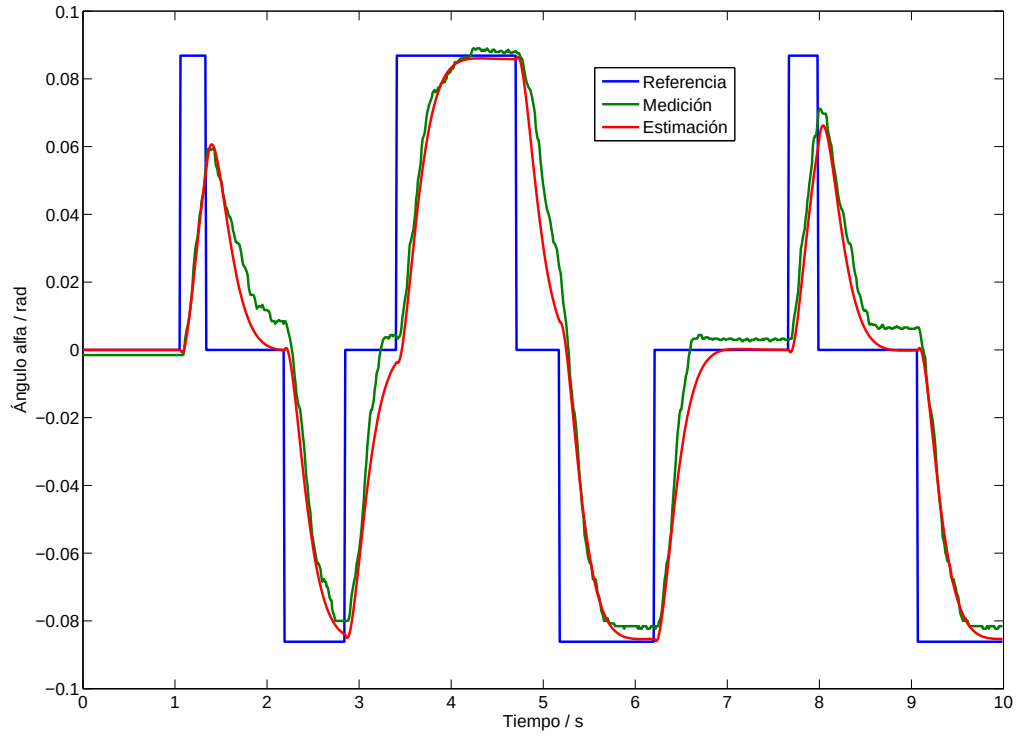


Figura 7.10: Comparación del modelo matemático y la respuesta real del sistema MOTOR – BARRA a lazo cerrado ante la señal de estímulo.

$$f = \frac{2}{5} m a_{CM} \tag{7.8}$$

Ahora

$$m a_{CM} + \frac{2}{5} m a_{CM} = mg \operatorname{sen} \alpha \tag{7.9}$$

$$\frac{7}{5} a_{CM} = g \operatorname{sen} \alpha \tag{7.10}$$

$$a_{CM} = \frac{5}{7} g \operatorname{sen} \alpha \tag{7.11}$$

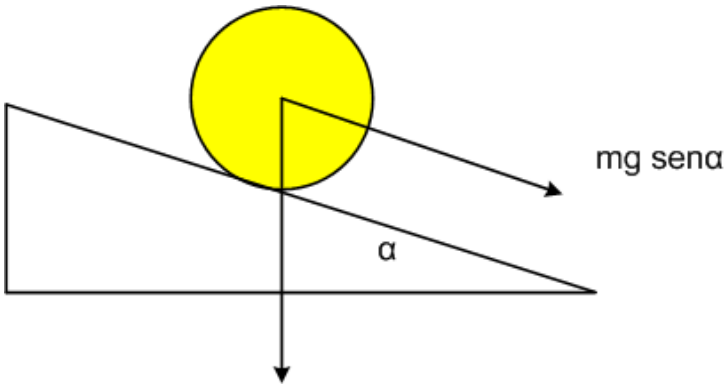


Figura 7.11: Esquema de la bola sobre una rampa inclinada a un ángulo α.

Aproximando $\operatorname{sen}\alpha \approx \alpha$

$$a_{CM} = \frac{5}{7} g\alpha \tag{7.12}$$

$$\tag{7.13}$$

se reemplaza $g = 9{,}8$, $a_{CM} = \frac{d^2}{dt} y(t)$ y $\alpha = x(t)$,

$$\frac{d^2}{dt} y(t) = 7x(t) \tag{7.14}$$

$$\tag{7.15}$$

y expresando en el domino de Laplace

$$Y(s) = \frac{7}{s^2} X(s) \tag{7.16}$$

$$\tag{7.17}$$

por lo que la función de transferencia es

$$H(s) = \frac{7}{s^2} \tag{7.18}$$

Transformado al dominio discreto mediante la aproximación de Tustin, se obtiene,

$$H(z) = \frac{0{,}00035(z+1)}{(z-1)^2} \tag{7.19}$$

con $ts = 10ms$

Modelado experimental

La estructura de control para el proceso de captura de datos se muestra en la figura 7.12.

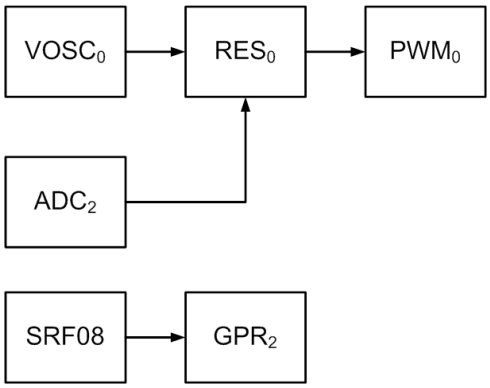


Figura 7.12: Estructura de control para la captura de datos del sistema BOLA.

La señal de entrada del sistema BOLA, es la salida del SISTEMA MOTOR – BARRA (figura 7.5). Esta señal se mide utilizando el ADC2. La distancia de la pelota al centro, es la salida del sistema BOLA. Esta se mide con el sensor de distancia SRF08.

La figura 7.13 muestra los datos capturados con una tasa de muestreo de 10ms durante un periodo de 6 segundos. En la figura se observa la referencia del ángulo de inclinación α de la barra y la inclinación medida por el ADC2. También, se observa la posición de la pelota.

La tabla 7.2 muestra una comparativa del porcentaje de mejor ajuste de los métodos de estimación utilizados. El método seleccionado fue SS de orden 2.

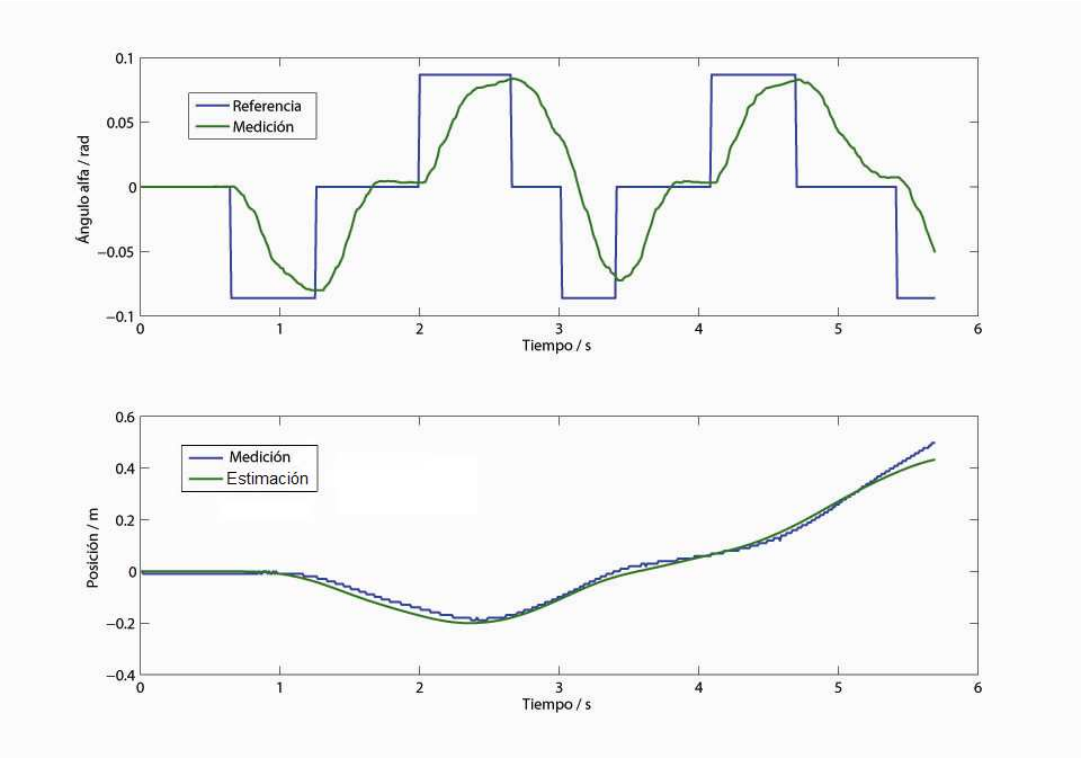


Figura 7.13: Resultados de la captura de datos utilizando la estructura de control de la figura 7.12.

El resultado de la estimación por el método SS con 2 estados es,

$$H(z) = \frac{0{,}0067054(z - 0{,}9363)}{z^2 - 2z + 1} \tag{7.20}$$

con $ts = 10ms$

La figura 7.14 muestra una comparativa del modelo teórico y el modelo experimental con SS. La señal de prueba utilizada fue la posición real de la barra capturada en el experimento.

De la figura se observa que el modelo obtenido con SS tiene mejor ajuste que el modelo teórico.

Tabla 7.2: Porcentaje de ajuste de los distintos métodos de modelado.

Método	Orden	Ajuste / %
ARMAX	2	76.76
BJ	2	76.76
SS	2	89.2
ARX	2	17.7

7.1.4. Compensador PID para el sistema MOTOR – BARRA

En secciones anteriores se cerró el lazo para el sistema MOTOR–BARRA y se comprobó su estabilidad. Para ello se utilizó un compensador de ganancia unitaria.

De la figura 7.10 se observa que el tiempo de estabilización es de aproximadamente 0.8 segundos y con un error de aproximadamente de 7.5 %.

Se utilizó la función de compensador PID de la unidad, para el control del sistema MOTOR – BARRA. La elección de este tipo de compensador, se debe por la facilidad de ajuste de los parámetros (solo tres: P, I y D).

Primero, se debe calcular una aproximación de los parámetros P, I y D utilizando el modelo del sistema MOTOR – BARRA. Para ello, se usó la herramienta SISOTOOL de MATLAB. El objetivo fué tratar de ajustar el tiempo de estabilización lo más bajo posible sin sobredimensionar los valores de salida del PWM (12V máximo ó 1.0 normalizado).

Con la herramienta SISOTOOL, se calculó un compensador que mejora el tiempo de estabilización a 0.5 segundos, con un error de 0 % y con sobreimpulso de 1.54 %. Las figuras 7.15 y 7.16 muestran la ubicación de los polos y ceros y la respuesta ante un escalón de 5° ó $0{,}0873 rad$.

El compensador obtenido es,

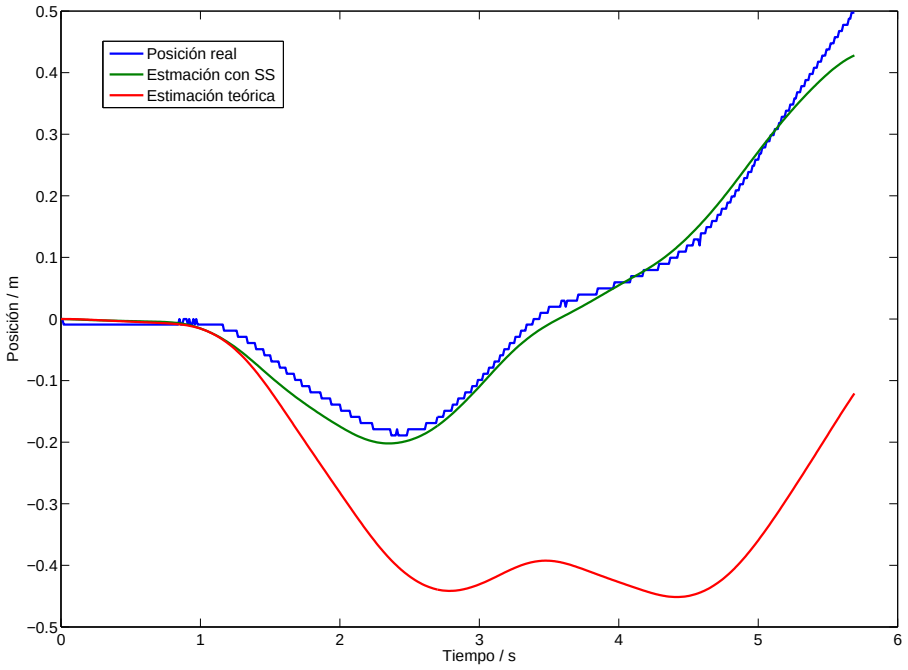


Figura 7.14: Comparación del modelo teórico y el modelo obtenido con SS.

$$C(z) = 1{,}5\frac{z - 0{,}2}{z} \tag{7.21}$$

La representación discreta de un compensador PD es,

$$PD(z) = K_P + K_D(1 - z^{-1}) \tag{7.22}$$

$$PD(z) = (K_P + K_D)\frac{z - \frac{K_D}{K_D + K_P}}{z} \tag{7.23}$$

igualando a $C(z)$ se deduce que,

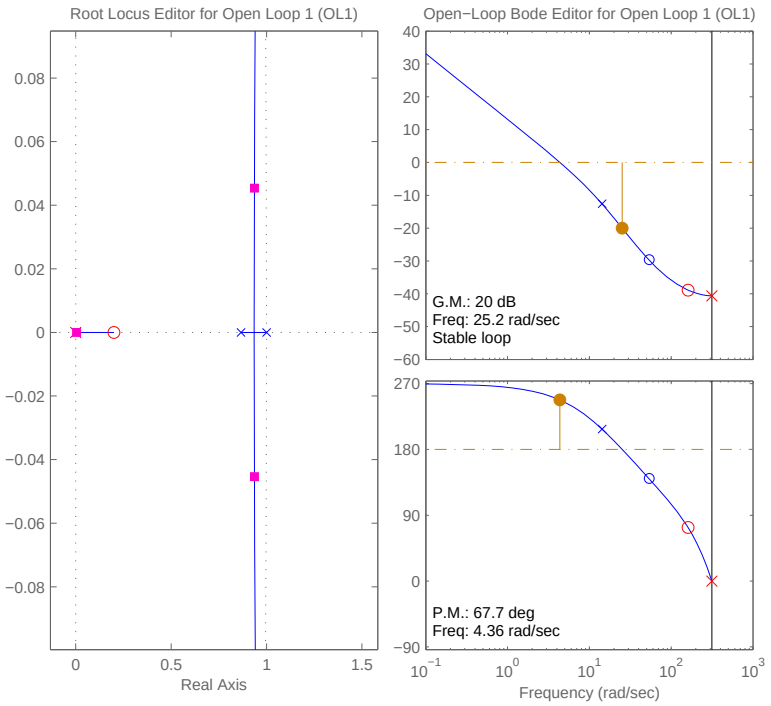


Figura 7.15: Ubicación de polos y ceros del sistema MOTOR – BARRA a lazo cerrado con compensador PD.

$$K_P + K_D = 1{,}5 \tag{7.24}$$

$$\frac{K_D}{K_D + K_P} = 0{,}2 \tag{7.25}$$

Obteniéndose así,

$$K_P = 1{,}2 \tag{7.26}$$

$$K_D = 0{,}3 \tag{7.27}$$

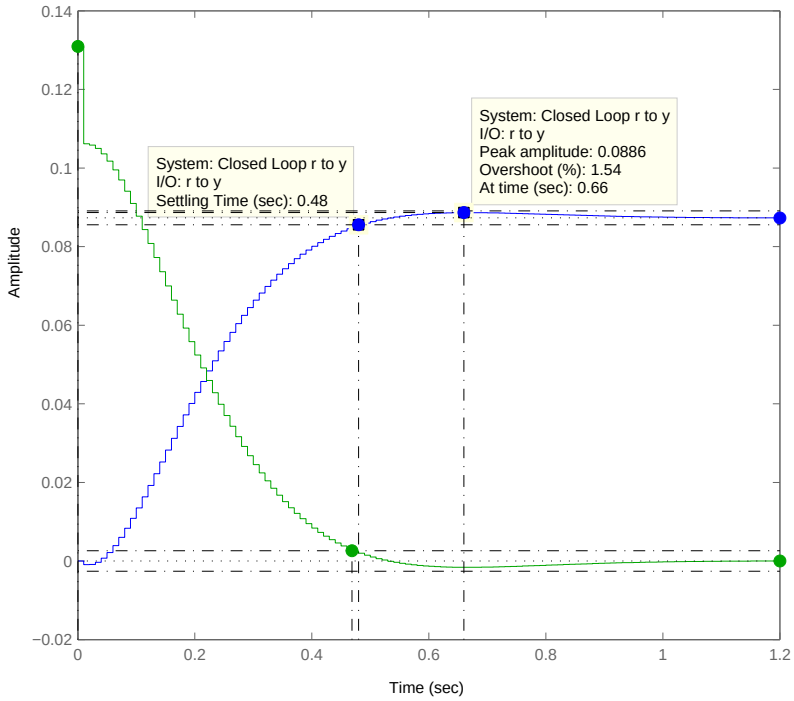


Figura 7.16: Respuesta del sistema MOTOR–BARRA ante un escalón de 0,0873rad.

Prueba del compensador PD para el sistema MOTOR – BARRA

El siguiente paso, fué bajar el compensador a la unidad controladora, y realizar una captura de datos para verificar el tiempo de estabilización, sobre impulso y error. Se utilizó la estructura de control que se muestra en la figura 7.17.

La figura 7.18 muestra la respuesta del sistema MOTOR – BARRA ante un escalón de 0,087rad. Se obtuvo un tiempo de estabilización de 0.5 segundos.

Sin embargo, se obtuvo un sobre impulso de 0,5° y un error de 0,2° cuando la referencia se ubica en 0°.

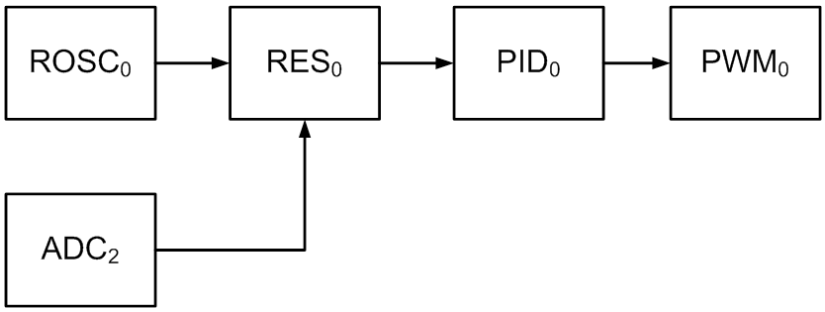


Figura 7.17: Estructura de control para el sistema MOTOR – BARRA.

7.1.5. Compensador PID para el sistema BOLA

Para completar la estabilización de la planta, fué necesario un segundo compensador. Éste compensador, se encarga de controlar la posición de la pelota variando el ángulo de inclinación de la barra.

Para simplificar el diseño, se hizo la suposición de que el sistema MOTOR – BARRA no existe y que el compensador del sistema BOLA puede cambiar directamente la posición de la barra.

El controlador obtenido no dará los resultados esperados; sin embargo, éste permitirá aproximar los valores de K_P y K_D. Posteriormente, se ajustarán los valores K_P y K_D empíricamente hasta conseguir la estabilización ó mejorar el tiempo de estabilización de la planta.

Se utilizó SISOTOOL como herramienta de diseño. Se cambió el tiempo de muestreo del modelo del sistema BOLA a 10 veces el tiempo de muestreo del sistema MOTOR – BARRA. De esta manera, se atenúan las vibraciones en la barra que se puedan producir por cambios rápidos en la entrada del compensador del sistema MOTOR – BARRA.

El nuevo modelo del sistema BOLA es,

$$H(z) = \frac{0{,}086275(z - 0{,}5049)}{(z-1)^2} \tag{7.28}$$

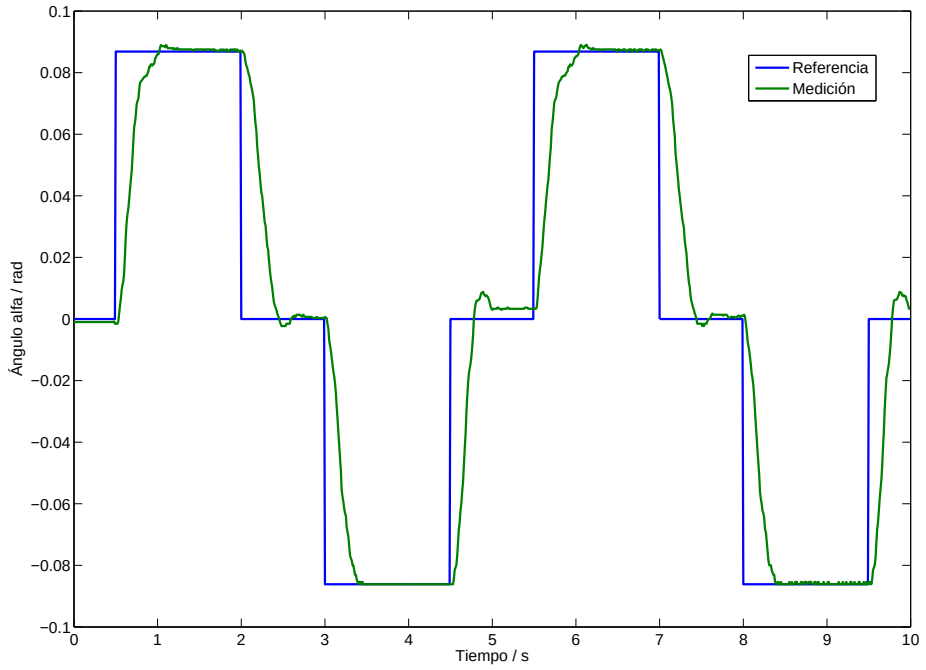


Figura 7.18: Respuesta real del sistema MOTOR – BARRA ante escalones de 0,0873rad.

con $ts = 100ms$.

Con SISOTOOL, se ajustó un compensador que estabiliza teóricamente la planta en 10s con un error de 0 %. La figura 7.19 muestra el diagrama de polos y ceros del sistema realimentado. La figura 7.20 muestra la respuesta de la planta ante un impulso.

El compensador obtenido es,

$$C(z) = 2\frac{(z - 0{,}75)}{z} \tag{7.29}$$

donde los valores de K_P y K_D son,

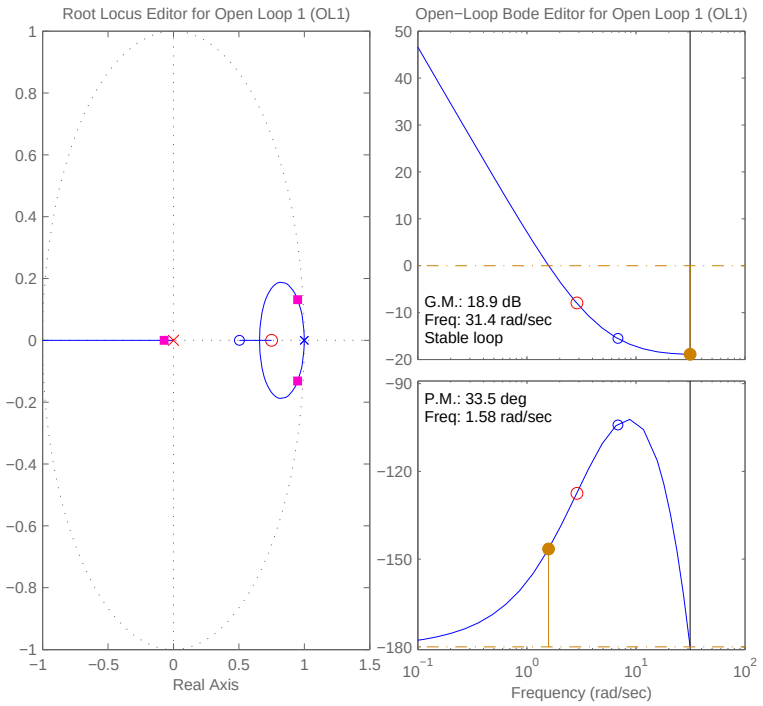


Figura 7.19: Ubicación de polos y ceros del sistema BOLA a lazo cerrado con compensador PD.

$$K_P = 0{,}5 \tag{7.30}$$

$$K_D = 1{,}5 \tag{7.31}$$

Se configuró la estructura de control que muestra la figura 7.26; se utilizaron saturadores en las salidas de los compensadores para evitar que el motor dé giros completos (máxima inclinación de la barra: $\pm 5°$, máximo ciclo de trabajo del PWM: $40\,\%$) .

La figura 7.21 muestra la posición de la bola ante perturbaciones. Las flechas indican el punto donde se produjo un impulso a la pelota. El sistema tarda aproximadamente 18 segundos en estabilizar la planta con un error de hasta 5cm en la posición de la pelota. Como se mencionó anteriormente, el cálculo de los valores K_P y K_D fueron una

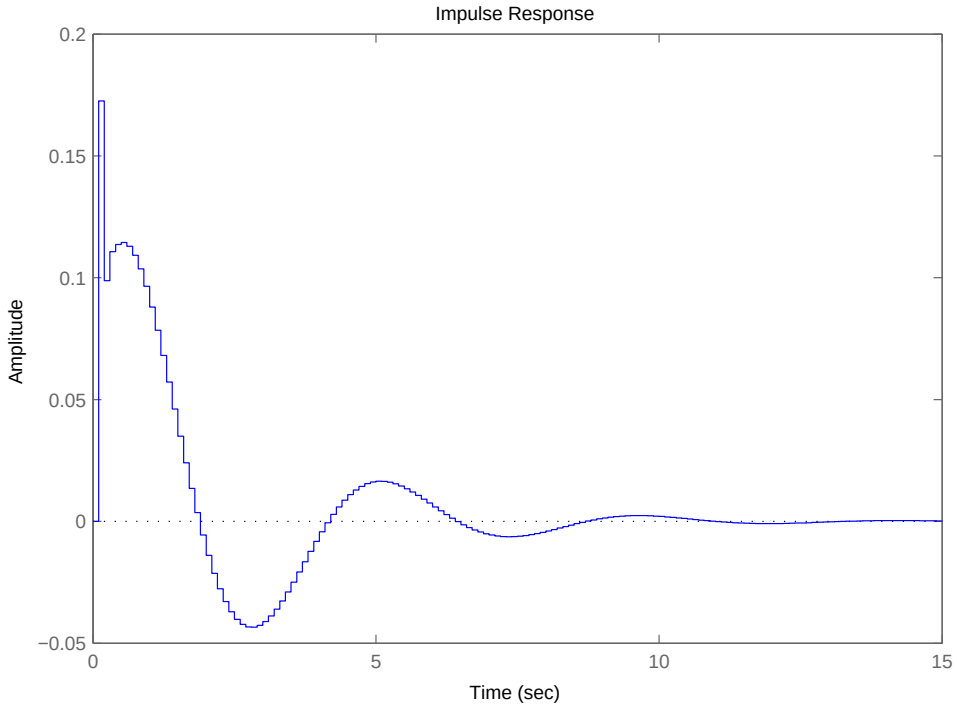


Figura 7.20: Respuesta del sistema MOTOR–BARRA ante un impulso.

aproximación que solo permitieron estabilizar la planta.

Posteriormente, se ajustaron los valores de K_P y K_D con la intención de mejorar el tiempo de estabilización de la planta. De manera empírica, se ajustó $K_P = 0{,}7$ y $K_D = 1{,}9$, la respuesta obtenida se muestra en la figura 7.22.

De la figura 7.22 se observa que el aumento de K_D mejoró el tiempo de estabilización a 14s. También, el aumento de K_P disminuyó el error de 5cm a 2cm.

Ésta es la ventaja que ofrece el compensador PID. Se puede aproximar los valores de P, I y D mediante métodos matemáticos. Luego, se ajusta de manera empírica para mejorar la respuesta del sistema.

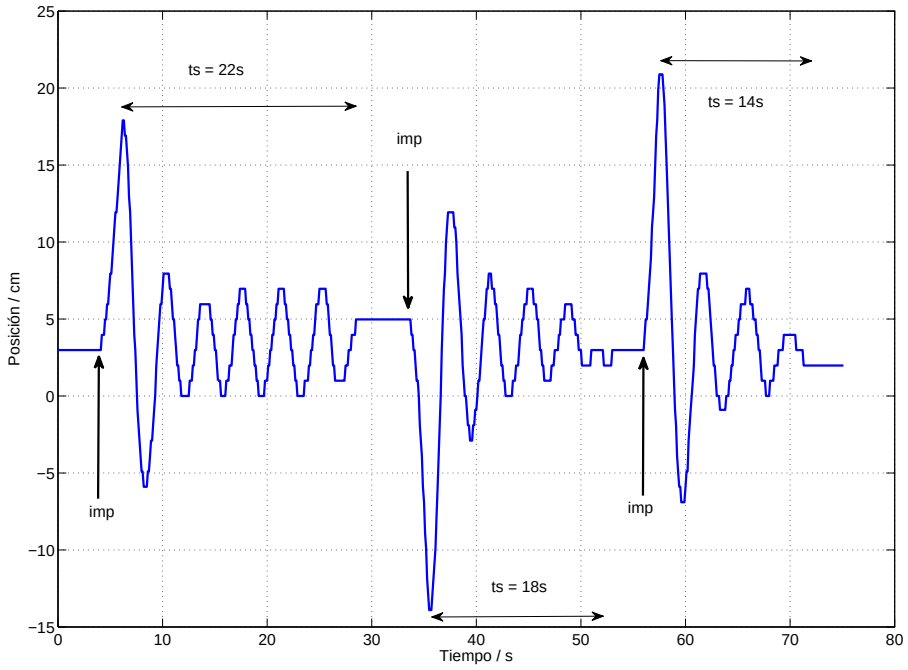


Figura 7.21: Respuesta de la planta ante perturbaciones utilizando los valores de $K_P =$ 0,5 y $K_D = 1{,}5$ en el compensador PD del sistema BOLA.

7.2. Pruebas de tiempo de cálculo y utilización de la CPU

Esta sección detalla pruebas de ejecución de los módulos virtuales de la unidad. Se midieron los tiempos de cálculo y porcentaje de utilización de la CPU.

7.2.1. Sin aplicación activa

Estas mediciones corresponden cuando la unidad está en estado de reposo, no se está ejecutando ninguna máquina virtual ni procesando ningún dato de entrada o salida.

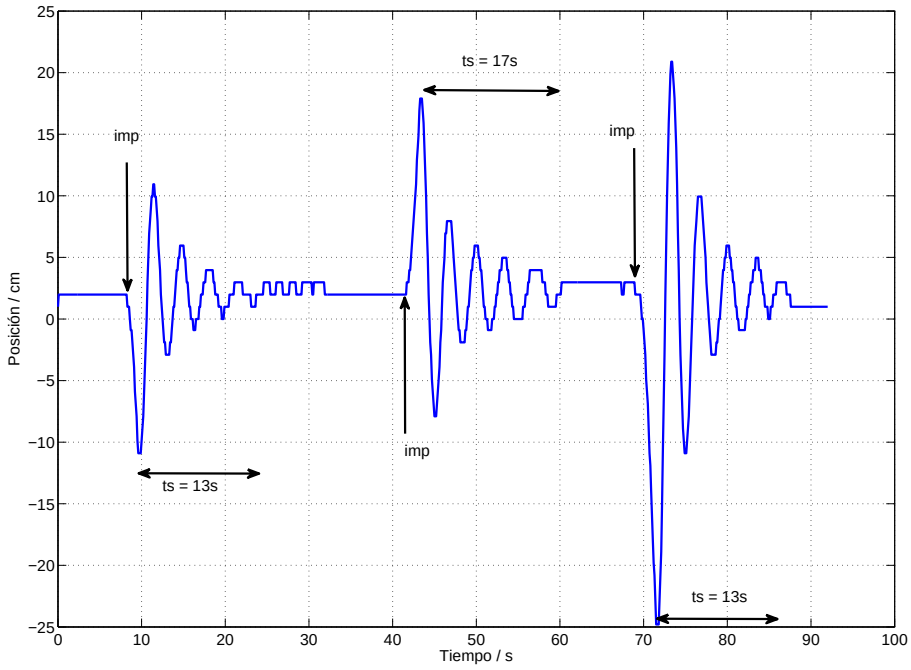


Figura 7.22: Respuesta de la planta ante perturbaciones utilizando los valores de $K_P =$ 0,7 y $K_D = 1{,}9$ en el compensador PD del sistema BOLA.

Tareas activas:

- Inicio.
- Comunicaciones 1 y 2.

Tareas en estado de espera:

- Máquina virtual 0 y 1.
- Envío 0 y 1.
- Captura.

- Ajuste y calibración.

Utilización de la CPU: 2 %

En estado de espera, la unidad utiliza el 2 % del CPU. Esto significa que el RTOS representa una baja carga de trabajo y es aceptable para aplicaciones en tiempo real [14].

7.2.2. Ejecución de aplicación *web Ball & Beam*

Se utilizó el programa de desarrollo NetBeans IDE para crear una apliación *web*. El usuario puede accesar a la página *web* y cargar la aplicación que permite experimentar con el *Ball & Beam*.

La figura 7.23 muestra la pantalla de inicio. Esta indica si la sesion de usuario con la unidad se estableció de manera satisfactoria.

Figura 7.23: Presentación

La figura 7.24 y 7.25 muestran las pantallas que permiten al usuario crear los compensadores y verificar el funcionamiento de los mismos.

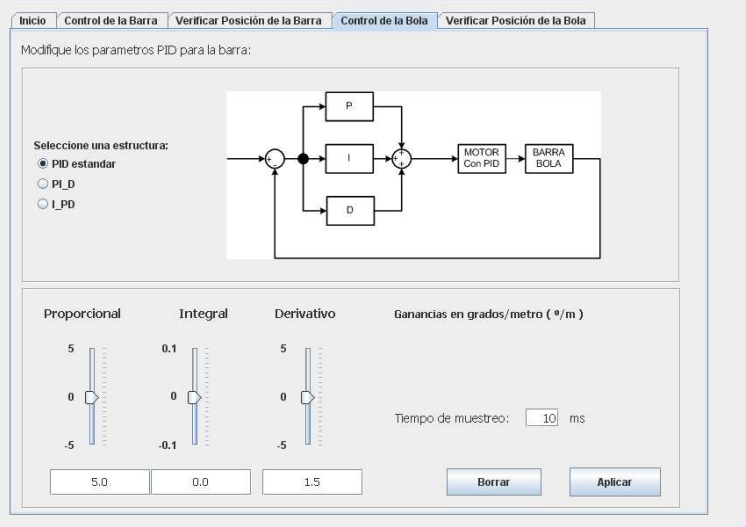


Figura 7.24: Etapa de configuración del controlador

La aplicación *web* se almacena en el sistema XPORT.

Esta aplicación utiliza una estructura de control con 2 compensadores PID, un filtro IIR de orden 2, un sensor de distancia externo con comunicación por el bus I^2C, el ADC2 y elementos de cálculo como restadores y saturadores. Además, se utiliza la tarea de captura para verificar el funcionamiento en tiempo real de la planta.

Tiempos de ejecución cuando se está controlando la planta

Estructura: La figura 7.26 muestra la estructura utilizada para la tarea máquina virtual 0.

Tareas activas:

- Inicio.
- Comunicaciones 1 y 2.
- Máquina virtual 0.

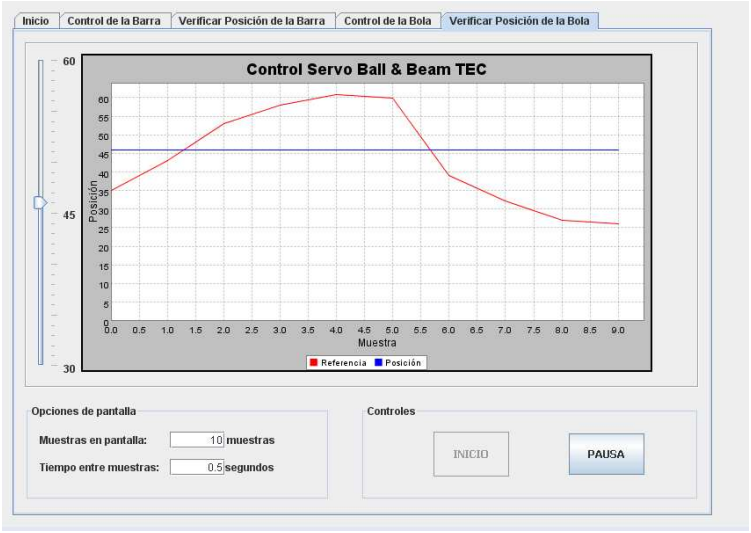


Figura 7.25: Etapa de verificación del controlador

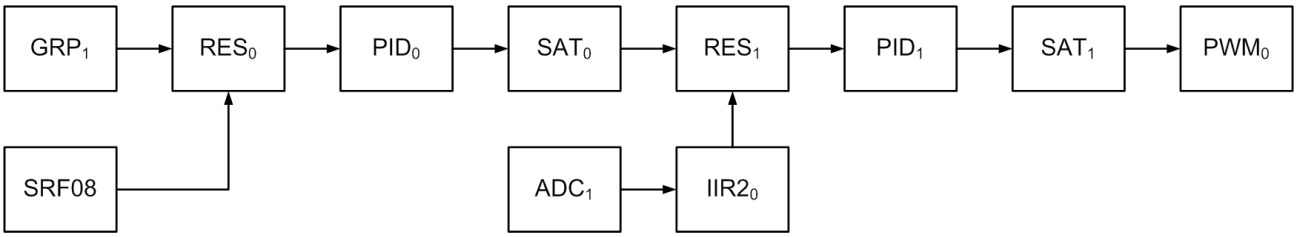


Figura 7.26: Estructura de control

- Envío 0.

Tareas en estado de espera:

- Máquina virtual 1.
- Captura.
- Envío 1.
- Ajuste y calibración.

Tiempo de cálculo de máquina virtual 36: μs

Total de módulos ejecutados por ciclo: **10**

- ADC: 11,1μs
- IIR2: 2,375μs
- SRF08: 1,175μs
- RES0: 0,85μs
- RES1: 0,85μs
- PID0: 2,275μs
- PID1: 2,275μs
- SAT0: 1,25μs
- SAT1: 0,95μs
- PWM0: 7,05μs

Utilización de total de la CPU: **15.91 %** con la tarea de envío 0 ejecutándose cada 0.5 segundos.

El tiempo de calculo representa el 3,6 % del elemento discreto con menor tiempo de muestreo (filtro IIR2 con $ts = 1ms$). Se concluye que la unidad controladora de procesos puede controlar una planta a una tasa de muestreo de 1ms de manera confiable.

7.3. Circuto impreso y limitaciones de diseño

La figura 7.27 muestra una fotografía de la unidad controladora de procesos.

Figura 7.27: Montaje del circuito impreso, etapa final.

Esta consiste de una placa madre fabricada en el Laboratorio de Electrónica y Circuitos Impresos (LECI) del ICE.

La ventaja que ofrece el LECI, es que se fabrican placas con hueco metalizado y pistas a doble cara. Además, se utiliza un torno CNC que hace los huecos de manera automatizada; sin embargo, el LECI define un estándar sobre las dimensiones que debe

tener el circuito impreso para poder ser fabricado. Los parámetros solicitados por el LECI para la fabricación del circuito impreso fueron:

- Tamaño mínimo de pista: 0,4mm.
- Tamaño mínimo de hueco (*vias*): 0,8mm.
- Tamaño mínimo entre pistas(*clearance*): 0,6mm.
- Tamaño mínimo de la dona: doble que el hueco ó 1.6mm.

Estas características, permiten crear circuitos impresos que utilizan circuitos integrados con encapsulado SOIC.

La unidad utiliza potenciómetros AD5263 de Analog Devices. El encapsulado de éstos potenciómetros es el TSSOP donde el espacio entre pistas máximo es de 0.35mm [1].

Como éste tamaño no puede ser fabricado en el LECI, se tuvo que fabricar subplacas para este tipo de encapsulado. Éstas se hicieron en el Laboratorio de Circuitos Impresos de la Escuela de Ingeniería en Electrónica del ITCR.

7.3.1. Limitaciones del diseño

Durante la depuración del circuito impreso y el *hardware*, se encontraron los siguientes errores:

1. **Dimesionamiento del área de disipación de calor en los reguladores de tensión:** La unidad utiliza varios reguladores de tensión para alimentar correctamente los circuitos integrados. En el diseño de la placa original todos los reguladores utilizan la fuente principal ($\pm 12V$) como entrada, esto hace necesario un correcto dimensionamiento del área de cobre para disipación de calor; sin embargo, en la placa de desarrollo no se creó ningún área de disipación de calor.

El problema se dió en el momento de exportar a PROTEL el diseño creado en EAGLE; ya que PROTEL eliminó todas las áreas de cobre para este propósito y la corrección no se pudo realizar.

2. **Utilización de componentes digitales con tecnología TTL:** Algunos componentes son de tecnología TTL y producen un consumo de corriente por arriba de los 200mA.

3. **Resolución de los potenciómetros digitales:** Los potenciómetros digitales se utilizan para crear referencias y ajustar ganancias en las entradas analógicas, se detectó un problema de resolución ($\pm 0{,}08V$) que no permite ajustar de manera precisa las referencias de 0V y 1.65V. Esto se debe a que los potenciómetros utilizan señal de referencia de $\pm 5V$; pudiéndose utilizar referencias cercanas a 1.65V para un ajuste adecuado (aproximadamente $\pm 7mV$).

Capítulo 8

Conclusiones y recomendaciones

8.1. Conclusiones

1. La utilización de un RTOS simplificó el trabajo, produjo un *firmware* eficiente y dio un aspecto elegante de la aplicación en general.

2. La unidad se programó con las siguientes funciones de control automático y DSP: filtros IIR, filtros FIR, funciones de transferencia, realimentación de estado y controladores PID.

3. Durante la depuración de la placa de desarrollo se encontraron limitaciones de *hardware* señalados en la sección 7.3.

4. La programación de las estructuras de control, estímulo y captura se realizan directamente desde las aplicaciones en JAVA o mediante instrucciones enviadas por el puerto serie.

5. La utilización de punto fijo como aritmética de la unidad mejora la velocidad de procesamiento de las funciones de matemáticas; como se mencionó, este método presenta la desventaja de errores por saturación o desbordamientos. Para minimizar tal efecto se utilizó la normalización de las operaciones de matemáticas.

6. Se puso a prueba la unidad, realizando un experimento con el *Ball & Beam*. El experimento consistió en crear aplicaciones de estímulo y captura con la unidad controladora. Se utilizó la herramienta IDENT del programa de análisis matemático MATLAB para procesar los datos obtenidos de la unidad y así estimar modelos de la planta. Posteriormente, se uso la herramienta SISOTOOL para calcular compensadores y luego bajarlos a la unidad para verificar el funcionamiento de los mismos. Se logró estabilizar la planta utilizando dos compensadores PD.

7. La estructura de control para el *Ball & Beam* tiene un tiempo de cálculo de $36\mu s$. Esto significa un 3,6 % de 1ms. Esto demuestra que la unidad puede controlar una planta de manera confiable a una tasa de muestreo de 1ms.

8.2. Recomendaciones

1. Durante el desarrollo del proyecto no se presentó o encontró ningún error de programación de la versión 2.85 de MicroC/OS-II para microcontroladores dsPIC; sin embargo, se recomienda revisar el sitio web (`http://www.micrium.com`) para actualizaciones y nuevas funcionalidades.

2. El *hardware* tiene la capacidad de más aplicaciones y funciones. Con esto se abre un camino al desarrollo de nuevos proyectos.

3. Se recomienda un rediseño de la placa con tecnología CMOS y si es posible fabricar el impreso en una empresa especializada

4. Se recomienda crear una aplicación tipo Simulink o LabVIEW en donde se integre todas las funciones de la unidad y de manera gráfica crear aplicaciones temporales

5. Microchip está introduciendo algoritmos para DSP y otros, con aritmética en punto fijo pero en formato Q16.15, esto es, 1 bit de signo, 15 bits de parte entera

y 15 bits de parte fraccionaria. Se recomienda consultar posibles actualizaciones y/o mejorar las funciones con este nuevo formato.

6. Se diseñó parcialmente un protocolo de comunicación multimaestro utilizando el bus I^2C. Éste protocolo se basó en el protocolo MODBUS. La programación y depuración del protocolo no se completó debido a un error de diseño del *hardware* del microcontrolador. Es posible que Microchip lance nuevas versiones del microcontrolador, en este caso, se recomienda verificar actualizaciones y continuar el desarrollo. Si no se encuentran actualizaciones, se recomienda cambiar la capa física del protocolo a RS485 ó CAN.

Bibliografía

[1] Analog Devices. *AD5263 datasheet*. [en línea]. [15 de Junio, 2008]. URL http://www.analog.com/UploadedFiles/Data_Sheets/AD5263.pdf

[2] Applied Industrial Control Solutions. *Digital Filter Design Writing Difference Equations For Digital Filters* [en línea].[22 de Setiembre, 2007]. URL http://www.apicsllc.com/apics/Sr_3/Sr_3.htm.

[3] Beis, Uwe *Converting Analog into Digital (IIR) Filters* [en línea]. [10 de Enero, 2008]. URL http://www.beis.de/Elektronik/Filter/AnaDigFilt/AnaDigFilt.html

[4] Brey, Barry B. *Los Microprocesadores Intel, Arquitectura, programación e intergaz de los procesadores 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro y Pentium II*. Quinta Edición. Prentice Hall of India, 2001.

[5] Castro Molina, Daniel. *Manual de usuario de la unidad controladora*. [10 de Mayo, 2008].

[6] Cisco Networking Academy. *Principios de Networking: Modelos de Networking* [en línea]. Junio, 2005, [5 de Noviembre, 2007]. URL http://cisco.netacad.net.

[7] Dahmke, Mark. *Microcomputer Operating Systems*. Byte Books, 1982.

[8] Deitel, P.J.; Deitel H.M. *JAVA, How to Program*. Seventh Edition. Prentice Hall, 2007.

[9] Facultad de Ciencias de la Computación *Grafos, Estructruras de Datos y Algoritmos* [en línea]. Febrero, 2008, [20 de Febrero, 2008]. URL http://www.cs.buap.mx/ titab/files/grafos1.pdf.

[10] Hernández Perales, Jorge Arturo; *Teorema de los paréntesis y sus aplicaciones.* [en línea]. [11 de Junio, 2008]. URL http://mictlan.utm.mx/temario/graf_teorpar.htm.

[11] Kernighan Brian W; Ritchie Dennis M. *El Lenguaje de Programación C.* Segunda Edición. Prentice Hall, 1991.

[12] Katsuhiko, Ogata. *Sistemas de Control en Tiempo Discreto.* Segunda Edición. Prentice Hall Hispanoamericana, S.A, 1996.

[13] Labrosse, Jean J. *Embedded Systems Building Blocks.* Second Edition. Lawrence, KS: CMPBooks, 2002.

[14] Labrosse, Jean J. *MicroC/OS-II The Real-Time Kernel.* Segunda Edición. Lawrence, KS: CMPBooks, 2002.

[15] Lantronix Inc. *XPortAR product description.* [en línea]. [24 de Junio, 2008]. URL http://www.lantronix.com/device-networking/embedded-device-servers/xport-ar.html

[16] Microchip Technology Inc. *MPLAB ICD2 In–Circuit Debugger User's Guide.* Microchip Technology Inc., 2005.

[17] Microchip Technology Inc. *dsPIC33F Family Reference Manual.* [en línea]. [14 de Junio, 2008]. URL http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2573.

[18] Microchip Technology Inc. *dsPIC® Language Tools Libraries*. [en línea]. [1 de Agosto, 2007]. URL http://ww1.microchip.com/downloads/en/DeviceDoc/51456b.pdf.

[19] Microchip Technology Inc. *dsPIC33FJXXXMCX06/X08/X10 Rev. A2/A3 Silicon Errata*. [en línea]. [1 de Agosto, 2007]. http://ww1.microchip.com/downloads/en/DeviceDoc/80307D.pdf.

[20] Microchip Technology Inc. *Explorer 16 Development Board User's Guide*. [en línea]. [1 de Agosto, 2007]. http://ww1.microchip.com/downloads/en/DeviceDoc/Explorer %2016 %20User %20Guide %2051589a.pdf

[21] Rorabaugh, C. Britton. *Digital Filter Designer's Handbook*. McGraw Hill, 1993.

[22] Proakis, John G; Manolakis Dimitris G. *Digital Signal Processing, Principles, Algorithms and Applications*. Fourth Edition. Prentice Hall of India, 2006.

[23] Universidad Central de Chile. *Sistemas operativos de tiempo real* [en línea]. [5 de Noviembre, 2007]. URL http://eiec.ucentral.cl/ftp/material/apuntes/iec51/Sistemas.

[24] Tocci, Ronald J. *Sistemas Digitales, Principios y Aplicaciones*. Sexta Edición. Pearson Educación, 1996.

[25] University of Michigan. *Control Tutorials for MATLAB*. [en línea]. [1 de Agosto, 2007]. URL http://www.engin.umich.edu/group/ctm/examples/ball/ball.html

[26] Wikimedia Foundation, Inc. *Arquitectura Eckert-Mauchly* [en línea]. Octubre, 2007, [11 de Junio, 2008]. http://es.wikipedia.org/wiki/Arquitectura_Eckert-Mauchly

[27] Wikimedia Foundation, Inc. *Sistema integrado* [en línea]. Octubre, 2007, [3 de Noviembre, 2007]. URL http://es.wikipedia.org/wiki/Sistema_integrado.

[28] Wikimedia Foundation, Inc. *Sistemas operativos de tiempo real* [en línea]. Noviembre, 2007, [2 de Noviembre, 2007]. URL http://es.wikipedia.org/wiki/Sistemas_operativos_de_tiempo_real.

Apéndice A. Glosario

CPU: *Centra Processing Unit* ó Unidad Central de Procesos. Una CPU es un procesador de instrucciones que se utiliza en un computador. Este procesa datos de una programa.

DSP: *Digital Signal Processing* ó Procesamiento Digital de Señales. Se refiere al campo de la ingeniería que se encarga del estudio de las señales mediante procedimientos computacionales. Por lo general, estas señales son audio o video; sin embargo, el concepto de señal abarca todo aquello que contiene datos o información. También DSP se conoce como un procesador digital de señales; en este caso, se refiere a un procesador que tiene una arquitectura especialmente diseñada para optimizar operaciones matemáticas aplicadas al análisis de señales.

EEPROM: Es una ROM que se puede borrar eléctricamente.

***Firmware*:** Se conoce como todo aquel *software* o conjunto de instrucciones que utiliza un sistema embebido para su operación. Por lo general, este se graba en una ROM.

Microcontrolador: Es una pequeña computadora. Éste incorpora: memoria, una CPU y periféricos (ADC, puertos serie, salidas y entradas digitales, etc). Todo el sistema está encapsulado en un solo circuito integrado.

Planta: Una planta es un sistema o proceso físico. Este puede ser: mecánico, térmico,

luminoso, magnético, químico, etc. Su característica principal, es que requiere de un control ya sea por su inestabilidad ante perturbaciones o por necesidad de una modificación en su comportamiento estable.

PLL: *Phase Locked Loop* ó Lazo de Seguimiento de Fase. Es un dispositivo electrónico utilizado como multiplicador de frecuencia. Se utiliza en microcontroladores para aumentar la frecuencia del reloj principal en base a una referencia generada por un oscilador de cristal.

ROM: Memoria de solo lectura. Es un tipo de memoria que solo es posible leer y no escribir.

RTOS: *Real Time Operative System* ó Sistema Operativo en Tiempo Real. Es un sistema operativo optimizado para trabajar en tiempo real. Su mayor aplicación está dentro de sistemas embebidos donde se requiere una respuesta rápida a los eventos.

TCP/IP: *Transmission Control Protocol / Internet Protocol* ó Protocolo de Control de Transmisión / Protocolo de Internet. Estos protocolos se utilizan para las comunicaciones entre computadoras; por ejemplo, Internet.

Sistema embebido: Conocido también como sistema integrado. Es un sistema eléctrico, electrónico o informático dedicado a una aplicación especifica.

Apéndice B. Descripción de los módulos de la máquina virtual

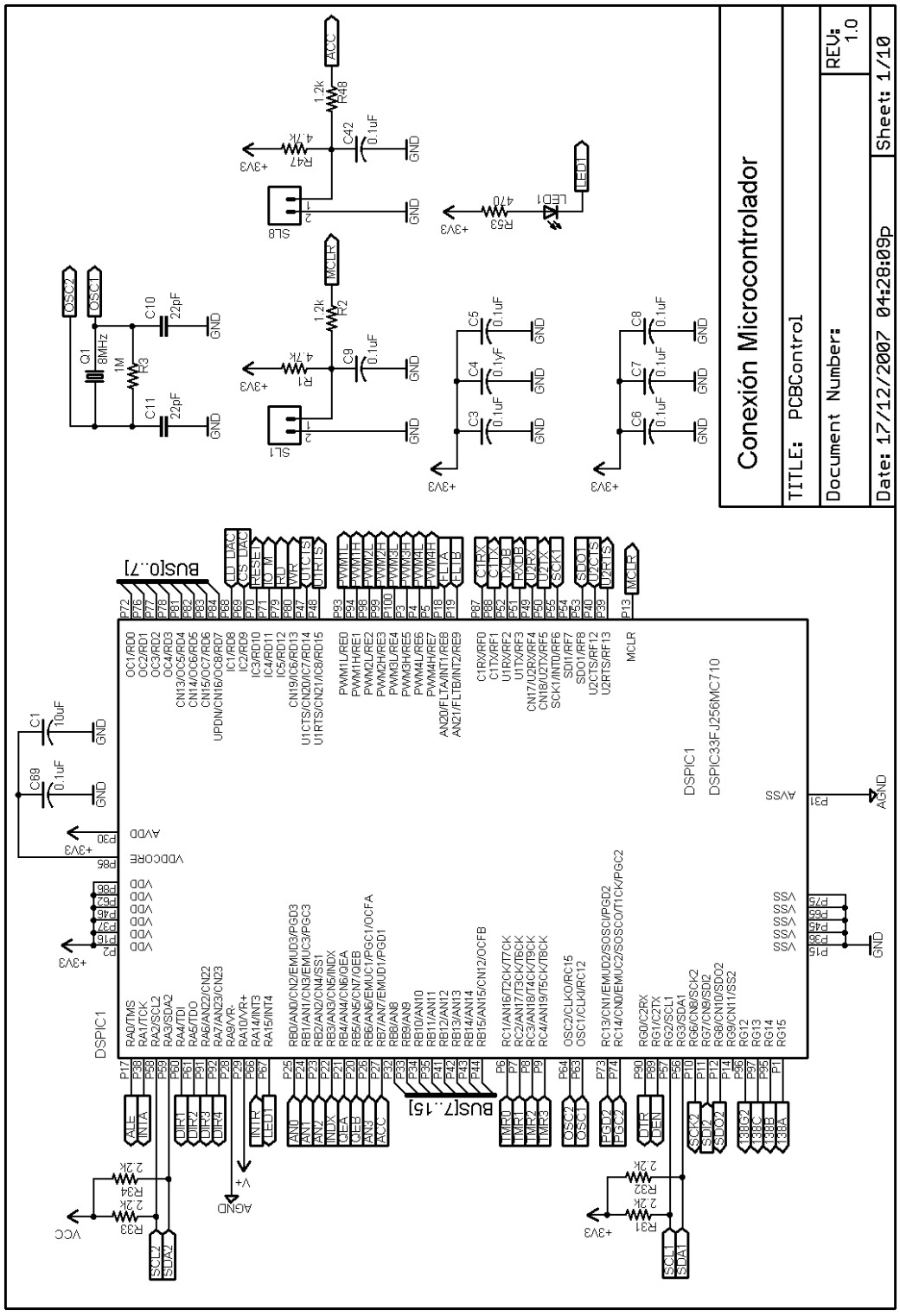
Tabla 1: Valores de tipo de los módulos de la máquina virtual, mnemónico utilizado y función.

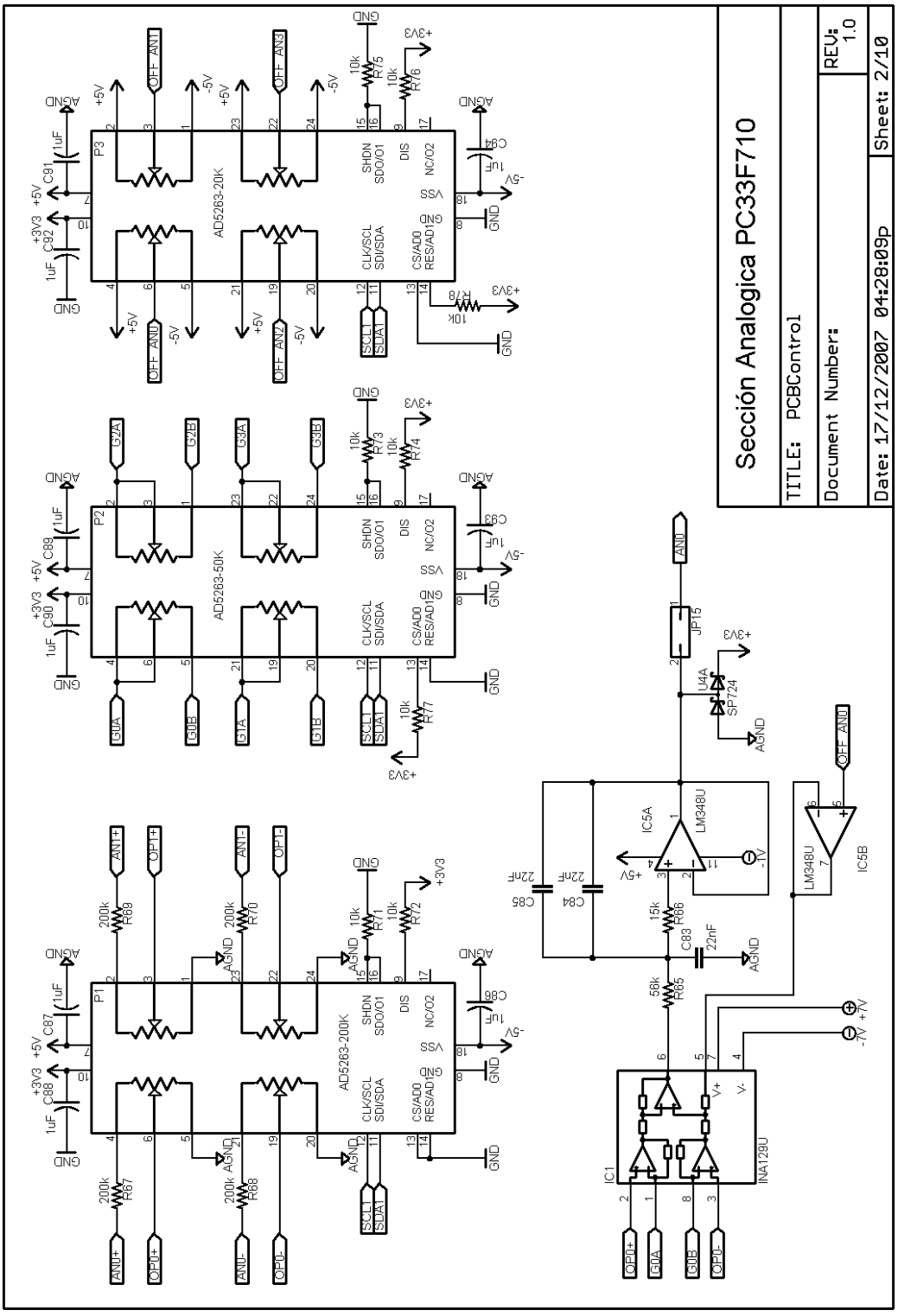
Valor de tipo	Mnemónico	Función
0x01h	ADC	Tomar muestra analógica
0x02h	GPR	Lectura de registro de propósito general
0x04h	QEIS	Lectura de QEI (velocidad)
0x05h	QEIP	Lectrua de QEI (posición)
0x08h	ADC	Lectura de sensor de distancia SRF08
0x40h	K	Bloque para realimentación de estado
0x41h	FIRX	Filtro digital de respuesta de impulso finita
0x42h	PID	Control PID
0x43h	PI_D	Control PI_D
0x44h	I_PD	Control I_PD

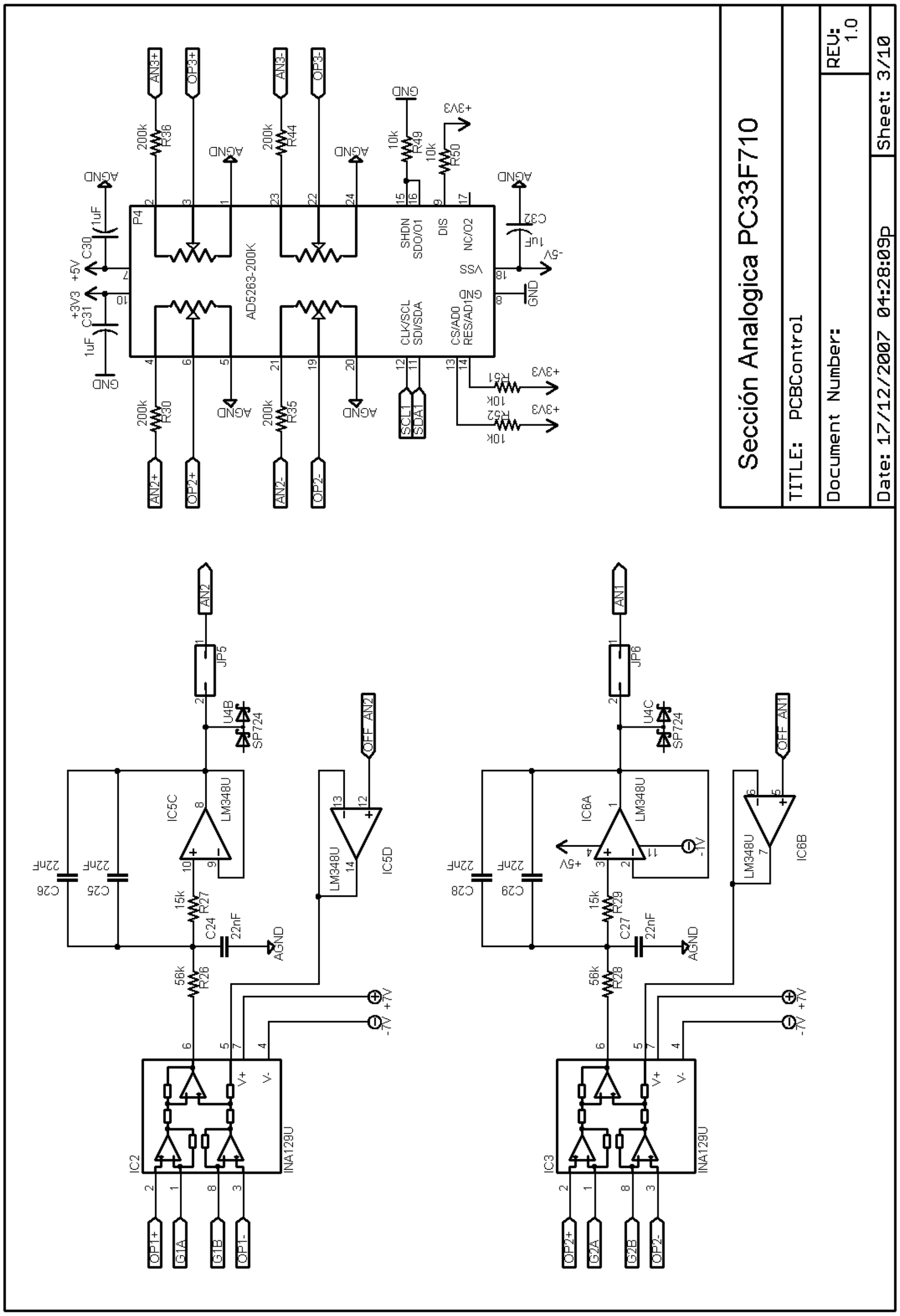
Tabla 2: Valores de tipo de los módulos de la máquina virtual, mnemónico utilizado y función, continuación de tabla 1.

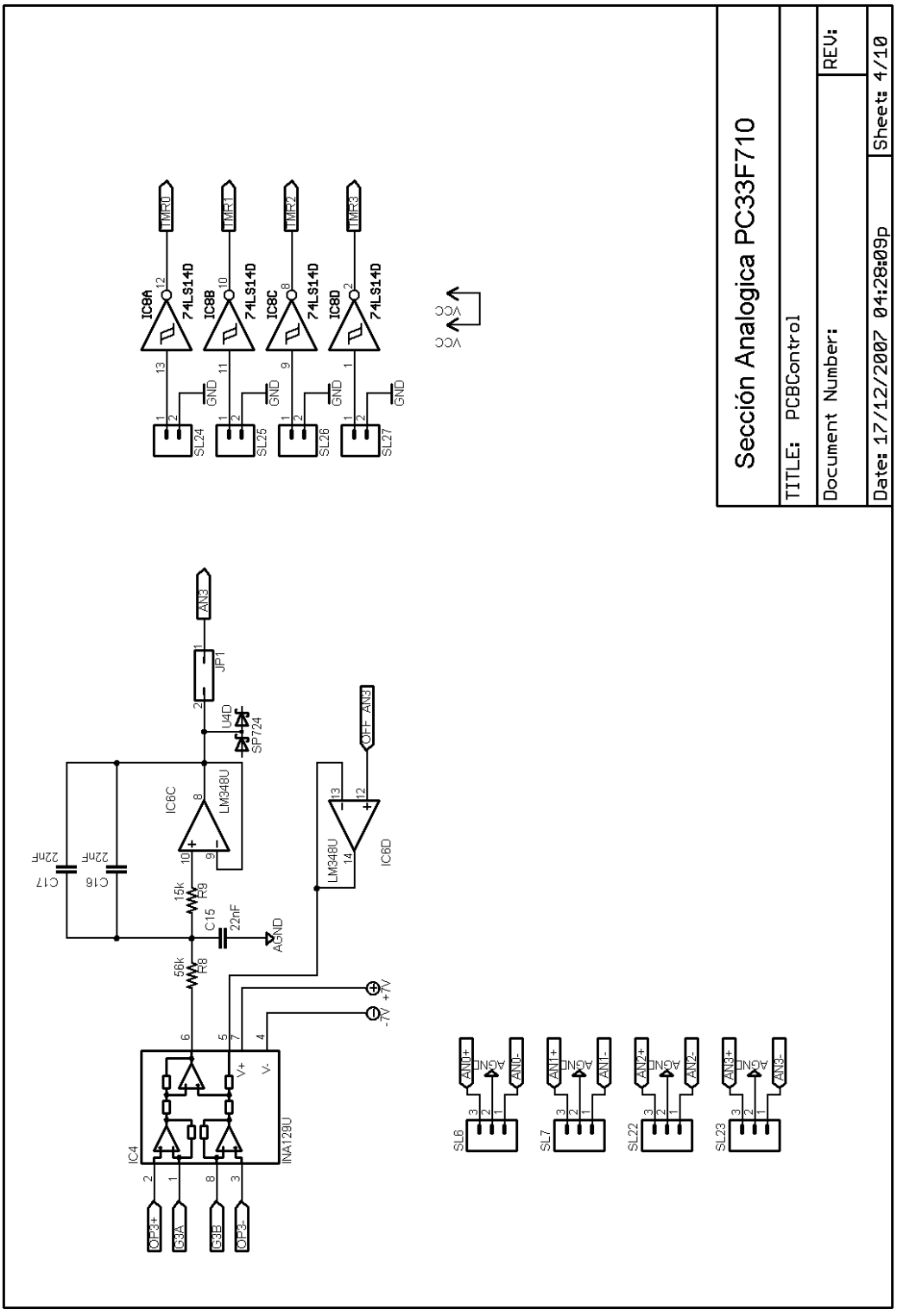
Valor de tipo	Mnemónico	Función
0x45h	IIR2	Filtro digital de respuesta de impulso infinita (orden 2)
0x46h	IIR4	Filtro digital de respuesta de impulso infinita (orden 4)
0x47h	NOTCH60	Filtro de ranura en 60Hz
0x48h	TF	Bloque para funciones de transferencia
0x49h	SUM	Bloque sumador
0x4Ah	SAT	Bloque saturador
0x4Bh	SOSC	Oscilador senoidal
0x4Ch	ROSC	Oscilador de pulsos de ancho constante
0x4Dh	TOSC	Oscilador triangular y diente de sierra
0x4Eh	VOSC	Oscilador de pulsos de ancho variable
0x4Fh	RES	Bloque restador
0x50h	GAIN	Bloque de ganancia
0x51h	MUL	Bloque multiplicador
0x52h	NOISE	Generador de ruido
0x53h	DELAY	Bloque de retardo
0xC0h	DAC	Salida analógica
0xC1h	PWM	Salida PWM
0xC2h	GPW	Escritura en registro de propósito general

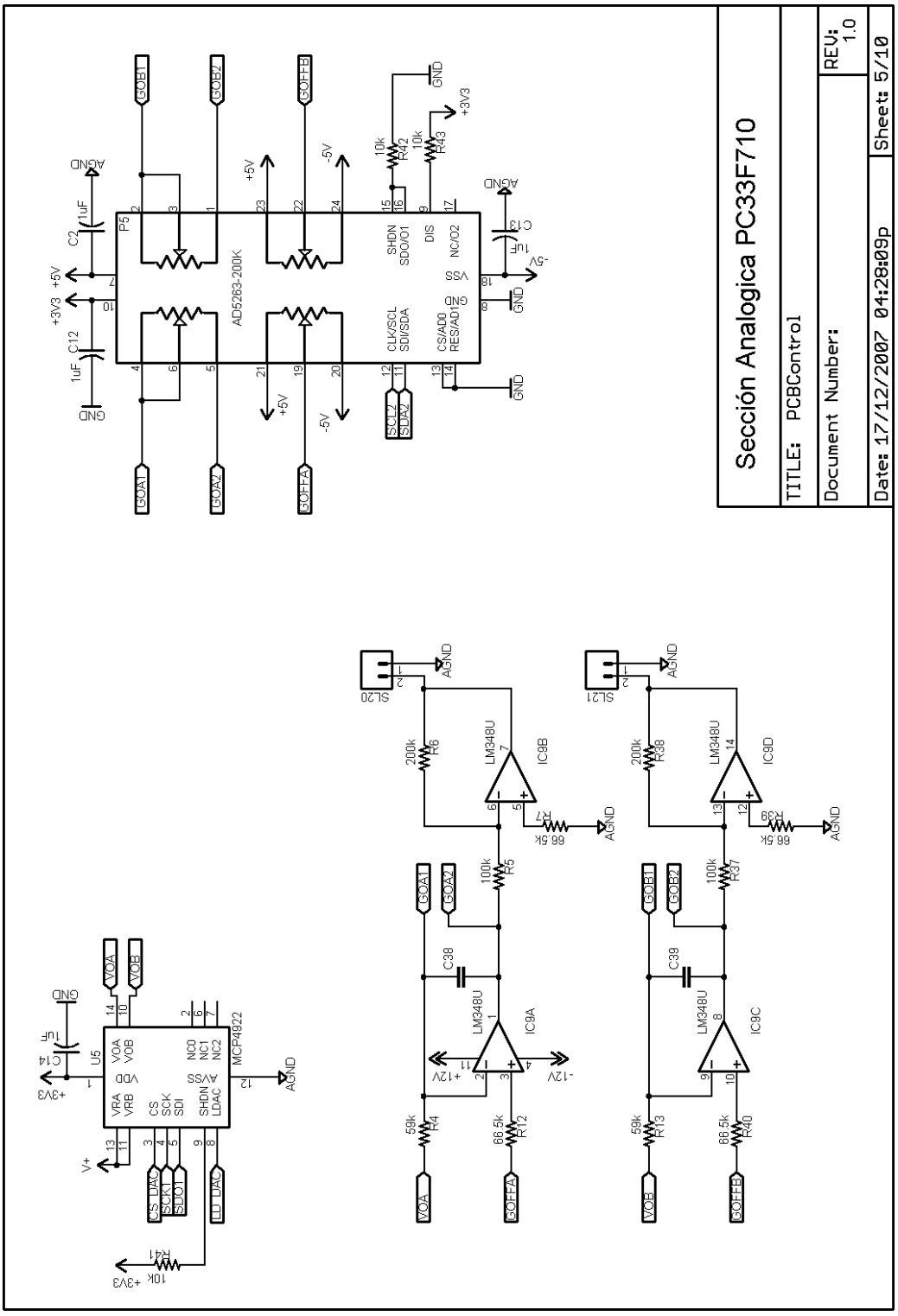
Apéndice C. Diagramas electrónicos

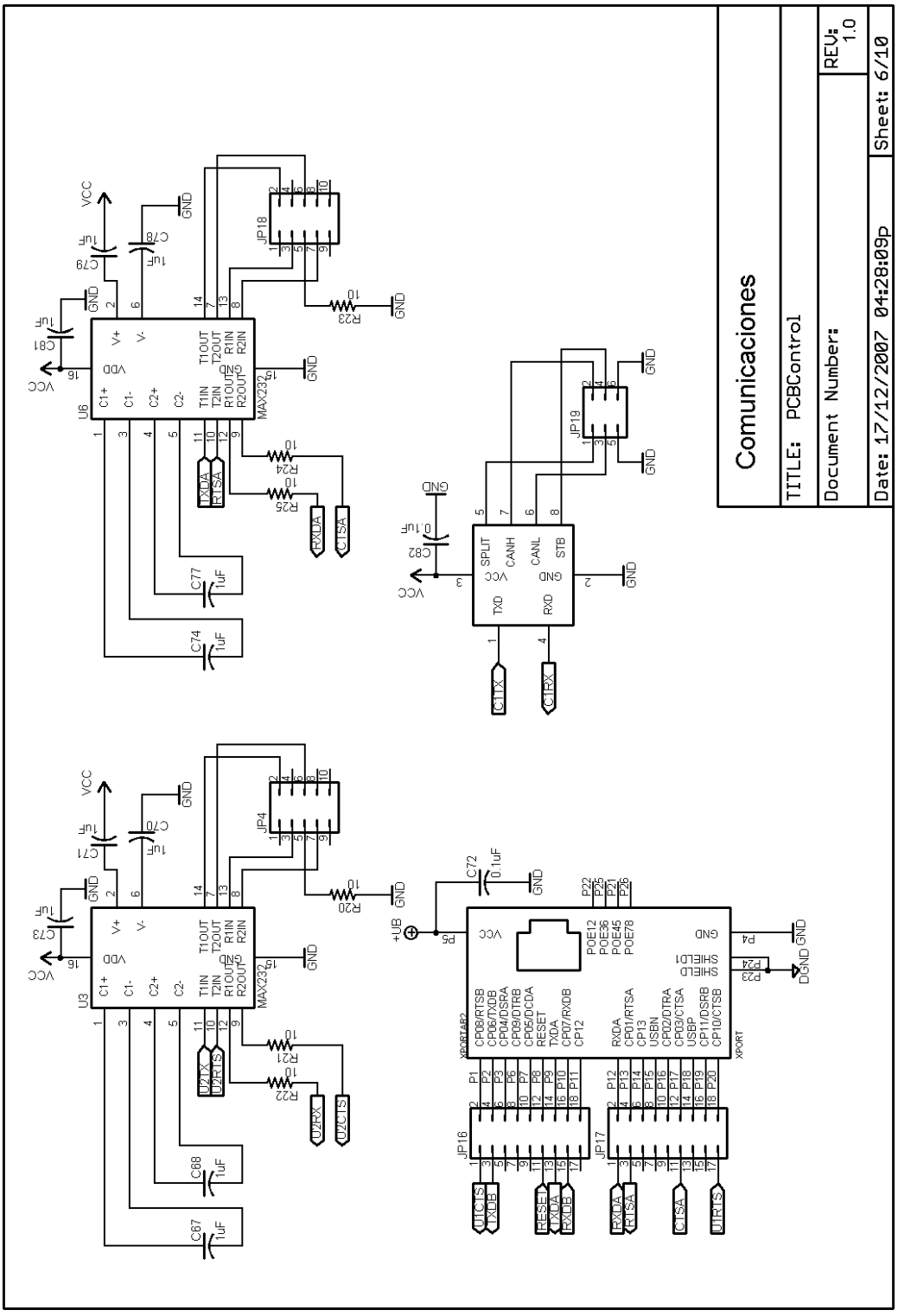

Conexión Microcontrolador
TITLE: PCBControl
Document Number:
REV: 1.0
Date: 17/12/2007 04:28:09p
Sheet: 1/10
DSPIC1
DSPIC33FJ256MC710
BUS[0..7]
BUS[7..15]
MCLR
AVSS
AGND
VDDCORE
AVDD
VCC
+3V3
GND
OSC2
OSC1
SCL2
SDA2
SCL1
SDA1
ACC
LED1

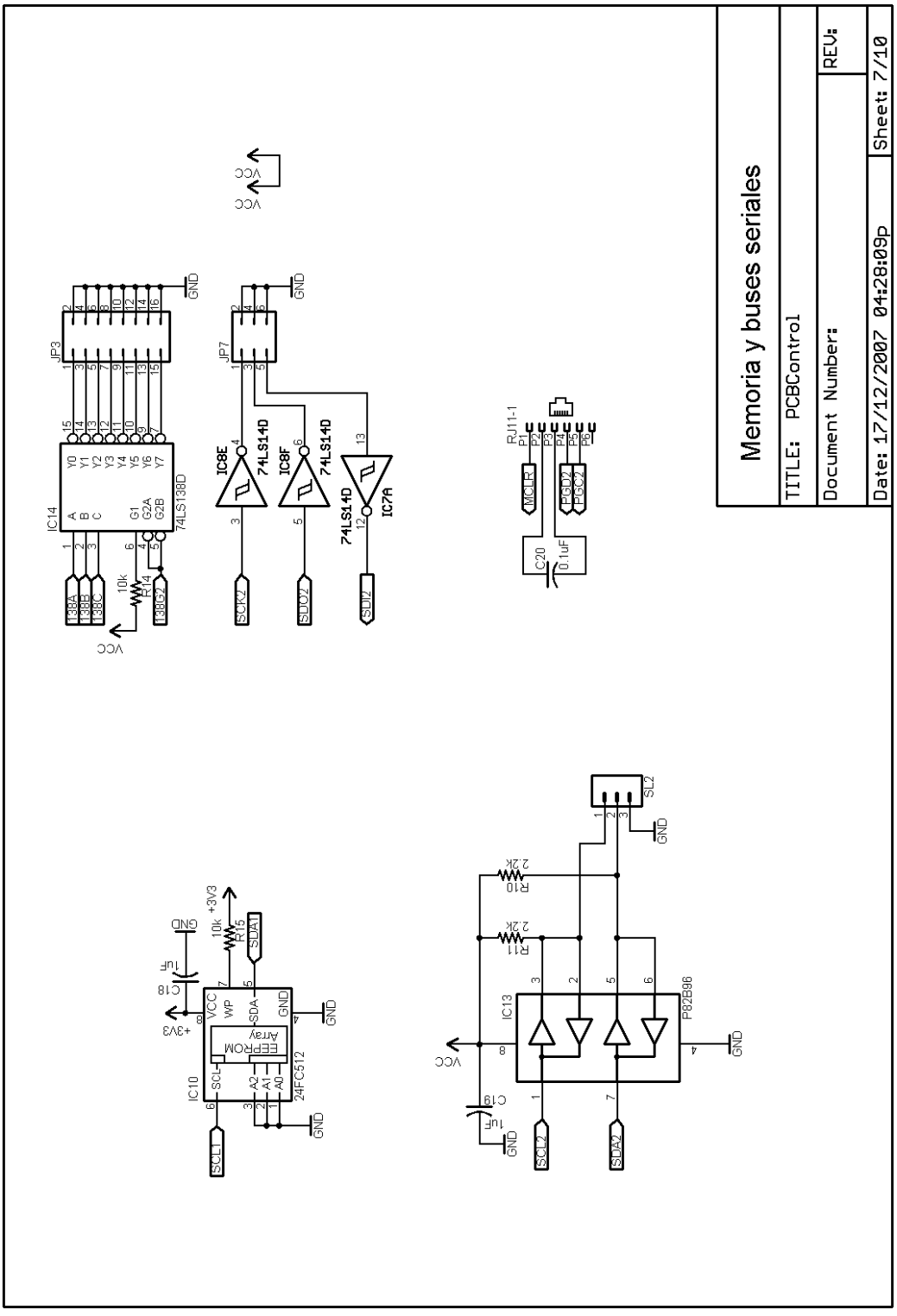

Sección Analogica PC33F710
TITLE: PCBControl
Document Number:
REV: 1.0
Date: 17/12/2007 04:28:09p
Sheet: 2/10
AD5263-200K
AD5263-50K
INA128U
LM348U
SP724
JP15
SCL1
SDA1
AGND
GND
+3V3
+5V
-5V
+7V
-7V

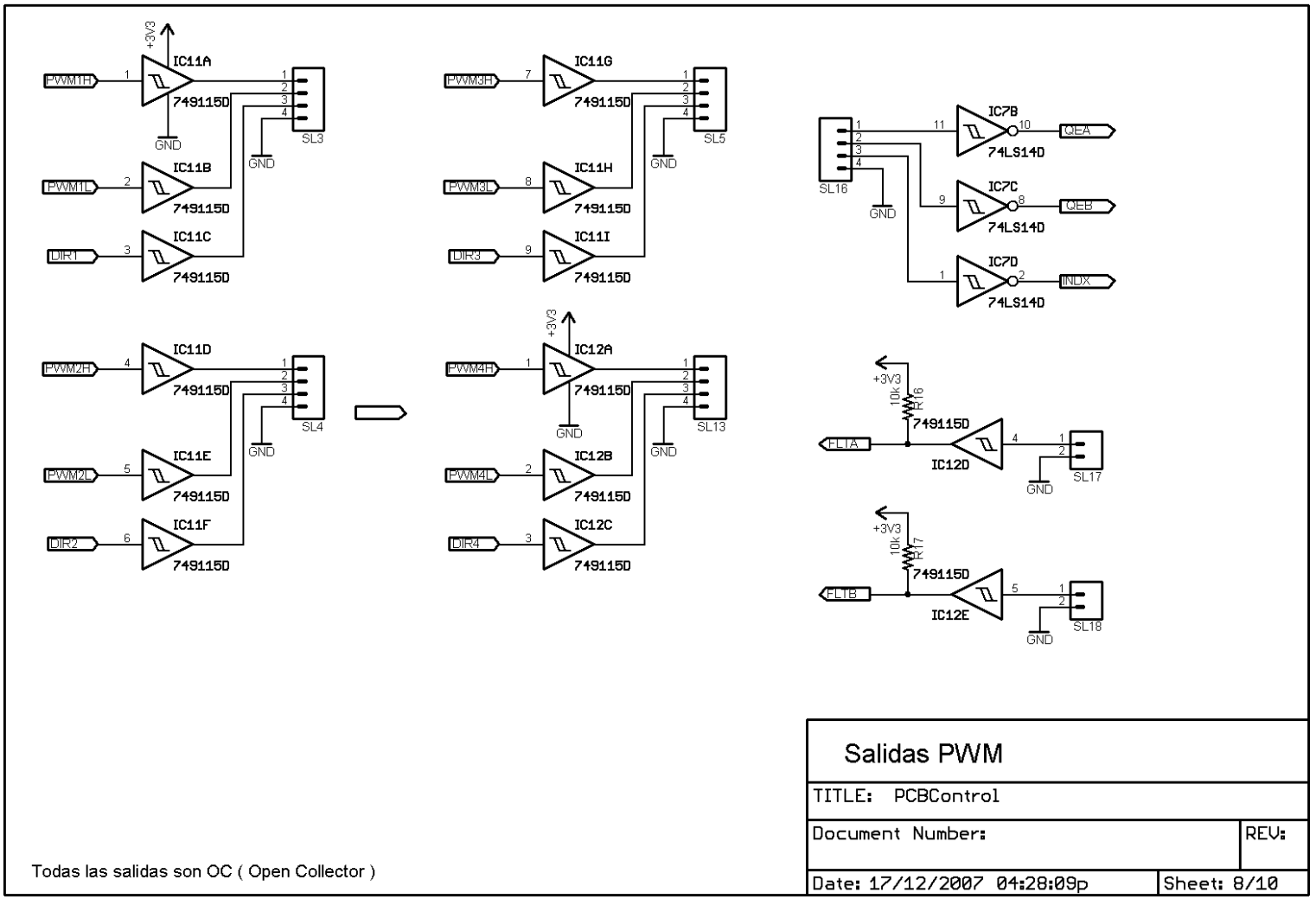

Sección Analogica PC33F710
TITLE: PCBControl
Document Number:
REV: 1.0
Date: 17/12/2007 04:28:09p
Sheet: 3/10

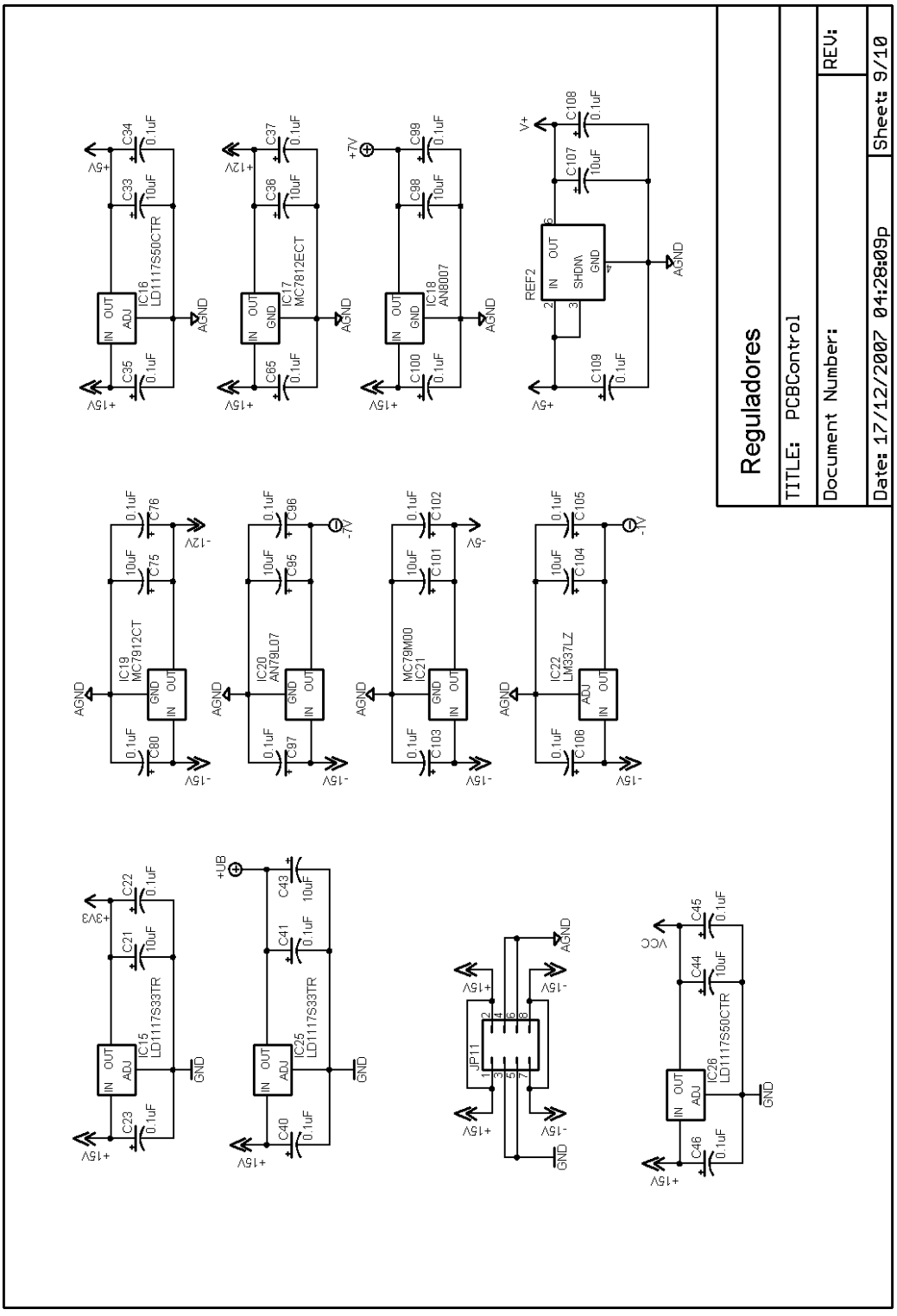

Sección Analogica PC33F710
TITLE: PCBControl
Document Number:
REV:
Date: 17/12/2007 04:28:09p
Sheet: 4/10
IMR0
IMR1
IMR2
IMR3
IC8A
IC8B
IC8C
IC8D
74LS14D
GND
SL24
SL25
SL26
SL27
VCC
AN3
JP1
U4D
SP724
IC6C
LM348U
IC6D
OUT_AN3
22nF
C17
C16
15k
R9
C15
AGND
56k
R8
+V
-V
IC4
INA129U
OP3+
G3A
G3B
OP3-
SL6
SL7
SL22
SL23
AN0+
AN0-
AN1+
AN1-
AN2+
AN2-
AN3+
AN3-
AGND

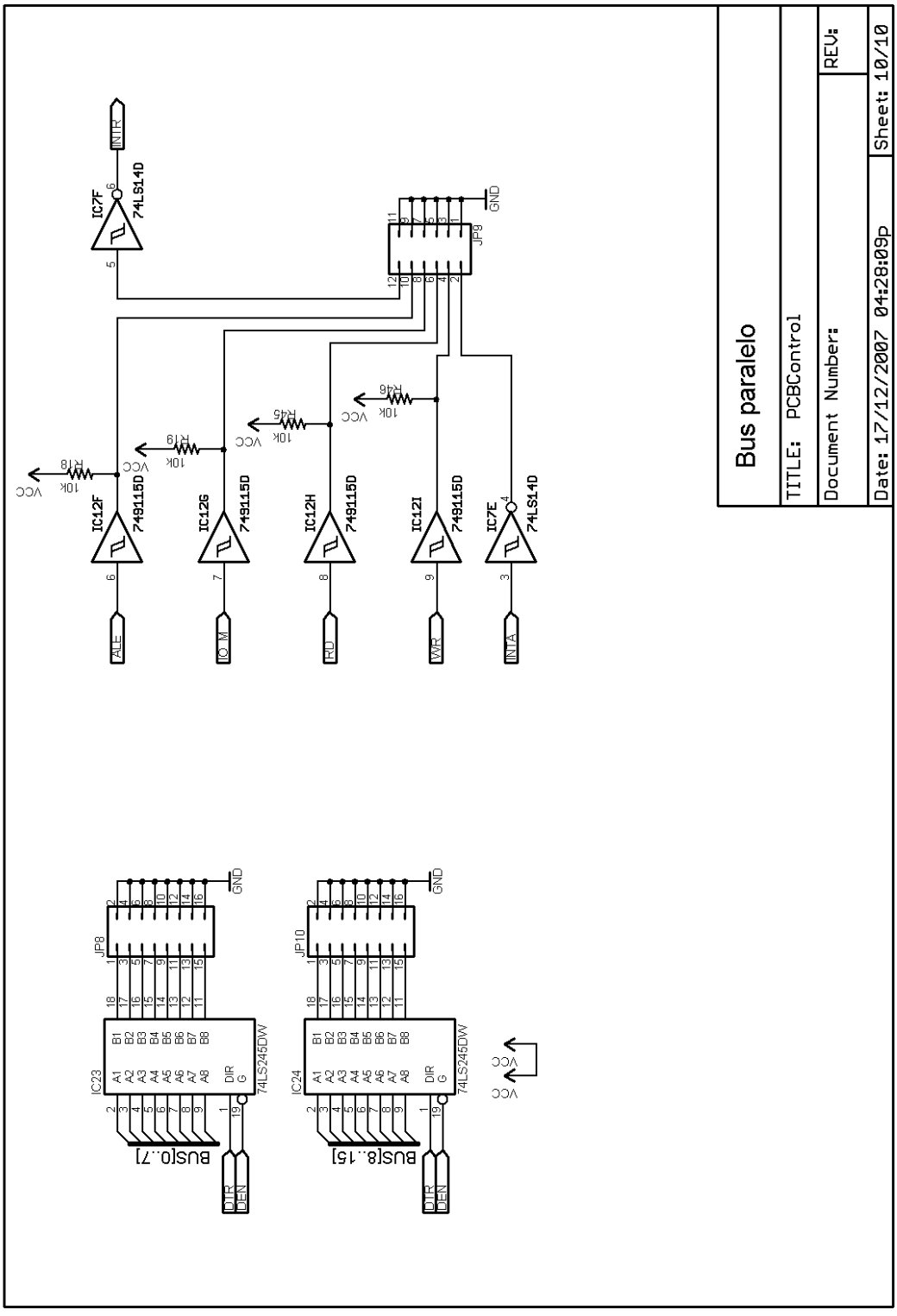

Sección Analogica PC33F710
TITLE: PCBControl
Document Number:
REV: 1.0
Date: 17/12/2007 04:28:09p
Sheet: 5/10
AD5263-200K
MCP4922
LM348U
GOA1
GOA2
GOB1
GOB2
COFFA
COFFB
VOA
VOB
SCL2
SDA2
AGND
GND
+3V3
+5V
-5V
+12V
-12V


Comunicaciones
TITLE: PCBControl
Document Number:
REV: 1.0
Date: 17/12/2007 04:28:09p
Sheet: 6/10
U6
U3
MAX232
JP18
JP19
JP4
JP16
JP17
C77
C74
C68
C67
C72
C82
R23
R24
R25
R20
R21
R22
RXDA
CTSA
CTX
CTRX
U2TX
U2RX
U2CTS
U2RTS
VCC
GND
+UB
DGND


Memoria y buses seriales
TITLE: PCBControl
Document Number:
REV:
Date: 17/12/2007 04:28:09p
Sheet: 7/10
JP3
JP7
IC14
74LS138D
IC8E
IC8F
IC7A
74LS14D
RJ11-1
MCLR
PGD2
PGC2
C20
0.1uF
IC10
24FC512
IC13
P82B96
SCL1
SDA1
SCL2
SDA2
SCK2
SDO2
SDI2
SL2
VCC
GND
+3V3
10k


+3V3
PWM1H
IC11A
74S115D
GND
SL3
PWM1L
IC11B
DIR1
IC11C
PWM2H
IC11D
SL4
PWM2L
IC11E
DIR2
IC11F
PWM3H
IC11G
SL5
PWM3L
IC11H
DIR3
IC11I
PWM4H
IC12A
SL13
PWM4L
IC12B
DIR4
IC12C
SL16
IC7B
74LS14D
QEA
IC7C
QEB
IC7D
INDX
10k
R16
FLTA
IC12D
SL17
R17
FLTB
IC12E
SL18
Salidas PWM
TITLE: PCBControl
Document Number:
REV:
Date: 17/12/2007 04:28:09p
Sheet: 8/10
Todas las salidas son OC (Open Collector)


Reguladores
TITLE: PCBControl
Document Number:
REV:
Date: 17/12/2007 04:28:09p
Sheet: 9/10
IC16 LD1117S50CTR
IC17 MC7812ECT
IC18 AN8007
REF2
IC19 MC7812CT
IC20 AN79L07
MC79M00 IC21
IC22 LM337LZ
IC15 LD1117S33TR
IC25 LD1117S33TR
JP11
IC26 LD1117S50CTR
AGND
GND
VCC


Bus paralelo
TITLE: PCBControl
Document Number:
REV:
Date: 17/12/2007 04:28:09p
Sheet: 10/10
INTR
IC7F
74LS14D
JP9
GND
IC12F
74S115D
IC12G
74S115D
IC12H
74S115D
IC12I
74S115D
IC7E
74LS14D
R48
R49
R50
R51
10K
VCC
ALE
IO_M
RD
WR
INTA
JP8
JP10
IC23
IC24
74LS245DW
DIR
G
BUS[0..7]
BUS[8..15]
DTR
DEN