Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Unidad control programable vía Ethernet

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

Juan Andrés Segreda Johanning

Cartago, noviembre de 2011

INSTITUTO TECNOLOGICO DE COSTA RICA

ESCUELA DE INGENIERIA ELECTRONICA

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

Ing. Carlos Badilla Corrales

Profesor lector

Ing. Miguel Hernández Rivera

Profesor lector

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

Lunes, 28 de noviembre de 2011. Cartago, Costa Rica

Declaratoria de autenticidad

Yo, Juan Andrés Segreda Johanning, inscrito como ciudadano costarricense, con la cédula de identidad número 1-1321-0586, declaro que el presente proyecto ha sido realizado por mi persona, y que en los casos en que se han tomado bibliografías o ideas se ha respetado las referencias utilizadas, mediante cita bibliográficas.

Juan Andrés Segreda Johanning

Cédula 1-1321-0586

Carné 200508503

Contenido

Índice de elementos

Índice de códigos

Índice de tablas

Índice de ecuaciones

Índice de ilustraciones

Agradecimiento

Agradezco profundamente a un sinnúmero de personas y profesores que han cooperado de alguna forma a que mi persona haya logrado alcanzar esta importante meta académica. Sin embargo, entre todas ellas destaco a mi padre, Ricardo, y a mi hermano Daniel, quienes han sido un apoyo fundamental durante todos los años de estudio.

Además, también extiendo mi agradecimiento al Ing. Eduardo Interiano, por permitirme realizar el proyecto con él y por su paciencia ante mis múltiples consultas durante el desarrollo de este proyecto. En esta misma línea, agradezco a todos aquellos ingenieros que realizaron parte del marco de este proyecto por tomar de su tiempo libre para explicarme sus avances y un especial reconocimiento a Ronald Soto, por sus valiosos aportes al proyecto y por ser quien dio forma a los circuitos impresos.

Resumen

El proyecto “UCPnet Kontrolo” es un sistema empotrado que tiene como objetivo principal servir de controlador programado para adaptarse a diferentes plantas físicas. Para tal fin, al sistema se le integraron diferentes tipos de entradas y salidas que el usuario puede programar para recibir las señales de los transductores y modificar las variables de estado de los actuadores.

El sistema se conecta con el ordenador mediante el estándar Ethernet, utilizando *sockets* de la familia BSD. Esto hace posible que la unidad “UCPnet Kontrolo” pueda ser manejada desde cualquier sistema operativo que sea compatible con “Telnet”. Sin embargo, el esquema del código de programación se mantuvo con el desarrollado para el laboratorio TeleLab del la Escuela de Ingeniería Electrónica, con el fin de poderse integrar fácilmente con la Interface de desarrollo gráfico del laboratorio.

La unidad controladora “UCPnet Kontrolo” incorpora una gama de módulos virtuales necesarios para desarrollar casi cualquier problema de teoría de control, como por ejemplo diferentes tipos de generadores asimétricos, módulos de operaciones aritméticas, tres versiones de controladores PID, filtros digitales y limitadores de señal, entre otros. Además, también incorpora un sistema de monitoreo en tiempo real que permite al usuario seleccionar y observar los valores en cualquiera de las entradas de los bloques del flujo que este ha programado.

Aparte de los canales analógicos que incorpora el sistema y de las señales de modulación por ancho de pulso, el usuario puede añadir diferentes tipos de periféricos mediante el protocolo Konekto, que fue descrito e implementado en este proyecto con el objetivo de entablar comunicaciones con diferentes dispositivos. De esta forma, esta unidad se vuelve expandible a nuevas necesidades como por ejemplo el control a través del procesamiento de imágenes.

Para el sistema, se desarrollaron circuitos acondicionadores de señal utilizados en los módulos de entradas y salidas analógicas, con lo cual se pueden manejar señales en distintos rangos de tensiones, lo que permite mejores resoluciones en los rangos de tensiones bajas y aumenta el número de señales que el sistema pueda leer y generar.

El sistema utiliza como núcleo la placa de desarrollo para PIC32 de la empresa Microchip, llamada “Ethernet StarterKIT Pic32”. Y se diseñaron los componentes de los módulos de entrada y salida, así como la placa base donde se conectan estos con la placa de desarrollo.

Palabras clave:

Unidad de control, Generador, Osciloscopio, Ethernet, RTP, PIC32, MIPS, Sistemas embebidos, controladores PID.

Abstract

The "UCPnet Kontrolo" project is an embedded system that has as main objective to serve as controller programmed to adapt to different physical plants. For this purpose, the system will integrate different types of inputs and outputs that the user can program to receive signals from the transducers and modify the state variables of the actuators.

The system connects to the computer using the Ethernet standard, using sockets from the BSD family. This allows the unit "UCPnet Kontrolo" to be operated from any “Telnet” compatible operating system. However, the programming code scheme remained developed for the TeleLab laboratory of the Electronic Engineering School, in order to be able to easily integrate with the Lab’s graphic development interface.

The control unit "UCPnet Kontrolo" incorporates a range of virtual modules required to develop almost any problem of control theory, such as different types of asymmetric generators, arithmetic modules, three versions of PID controllers, digital filters, and signal limiters among others. In addition, incorporates a real-time monitoring that allows users to select and see the values in any of the entries of the flow block.

Apart of the analog channels that the system incorporates, the signals and width modulation pulse, the user can incorporate different types of peripherals via Konekta protocol; protocol was described and implemented in this project with the aim of file communications with different devices. This way, the unit becomes expandable to new needs such as over image processing control.

In order to accomplish success for the system, the development of signal conditioning circuits used in the input modules and analog outputs was necessary, which can handle signals at different tension ranges, allowing better resolutions in low tension ranges and increasing the number of signals the system can read and create.
The system uses the development plaque “Ethernet Starterkit PIC32" from the company Microchip as a main core. The components of the input and output modules and the base plate where everything is connected with the development board were designed brand new.

Keywords

Control unit, generator, oscilloscope, Ethernet, RTP, PIC32, MIPS, Embedded Systems, PID.

Capítulo 1: Introducción

La carrera de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica contempla en su plan de estudios el curso “Laboratorio de Control”, el cual ha sido impartido principalmente por el Ing. Eduardo Interino (diplomado en Control de la Universidad Técnica Hamburg-Harburg –TUHH–, Alemania), quien ha utilizado como metodología de aprendizaje, en grupos de 3 o 4 personas, el diseño e implementación de un controlador para una planta física, el cual involucra diseñar el controlador (objetivo del curso) y la creación del circuito donde se aplicará, e incluso, en raras ocasiones, la misma planta.

En el 2008, se inicia el proyecto TeleLab, un laboratorio que tiene como idea poder manipular las plantas de la escuela de forma remota, mediante el establecimiento del diagrama de bloques o la función de transferencia del control. Esta idea nace inspirada en TeleLab Unir en Siena, Italia, y de programas como Simulink y LabView.

En principio, se compraron tarjetas de adquisición y se desarrollaron aplicaciones con LabView, de la marca National Instruments, con el cual se desarrollaron varios proyectos guiados; no obstante, el costo de estas tarjetas superaba los mil dólares haciendo que la expansión del proyecto tuviera que dar otro giro.

En el 2008, se desarrolló una primera unidad de control que se programa vía Ethernet por medio del envío de tramas hexadecimales, las cuales iban conformando el controlador programado. A este sistema se le conoció como BPC y estuvo a cargo del ingeniero Daniel Castro.

En febrero del 2010 el ingeniero Óscar Caravaca desarrolla un entorno de desarrollo integrado (IDE, por sus siglas en inglés) diseñado como un *applet* de java, el cual se situaría en un servidor de la Escuela y serviría de intermediario entre el estudiante y la Unidad BPC, de modo tal que el aprendiz solo tuviese que unir y dimensionar componentes de forma gráfica y no mediante el código numérico que interpretaba el BPC. Además, en esta misma etapa se describe un esquema de programación y se le asignan mnemónicos a las diferentes instrucciones del BPC, con lo que se obtuvo un lenguaje intermedio.

Para el segundo semestre del 2010, se intentó hacer una nueva unidad de control, pero esta vez sobre una plataforma de desarrollo conocida como Hawkboard (OMAPL138). Sin embargo, se dificultó la labor debido a que esta plataforma no estaba diseñada para trabajar en tiempo real, si no que su procesamiento de DSP (Procesamiento de señales digitales) lo hacía a partir de bloques de datos previamente gravados y, por consiguiente, no servían para la controlabilidad de una planta.

Se rescata del diseño experimentado en Hawkboard, las bibliotecas matemáticas que se desarrollaron para este y que prometen una optimización en el tratamiento de las señales que transcurrirían por un controlador, con el fin de poder incorporarla en un futuro diseño del BPC.

En la etapa del proyecto que describe este documento, se diseña una unidad controladora, totalmente nueva. Esta unidad interpreta el lenguaje intermedio del Ing. Óscar Caravaca, recibiéndolo en un solo archivo. Además, esta unidad mejora la velocidad de transmisión por Ethernet, lo que permite realizar las monitorizaciones en tiempo real. También se agregan funciones al sistema como el desplazamiento en el tiempo y se modifican las formas de configurar de los parámetros para los módulos de "función de transferencia" y "controladores PID" aceptando sus constantes en el tiempo continuo.

Se desarrolla e implementa el protocolo "Konekto", el cual permite aumentar el número de periféricos que se puedan conectar al sistema y lo hace más versátil.

En la parte de *hardware* del sistema, se utiliza como fuente de alimentación, las tensiones brindadas por el puerto USB, el cual se ha convertido en la forma más común para alimentar dispositivos de baja tensión, por lo cual existen miles de adaptadores. Además, se escogió diferentes conectores para los distintos tipos de señales, como por ejemplo el uno de conectores BNC para las señales analógicas.

Asimismo, se diseñaron circuitos de acondicionamiento de señales dinámicos que se calibran de forma mixta, tanto por software como por la ubicación de los potenciómetros digitales.

En la Figura 1, se muestra un esquema de la idea de las unidades controladoras, con el fin de facilitar el entendimiento de la razón de ser de estas.

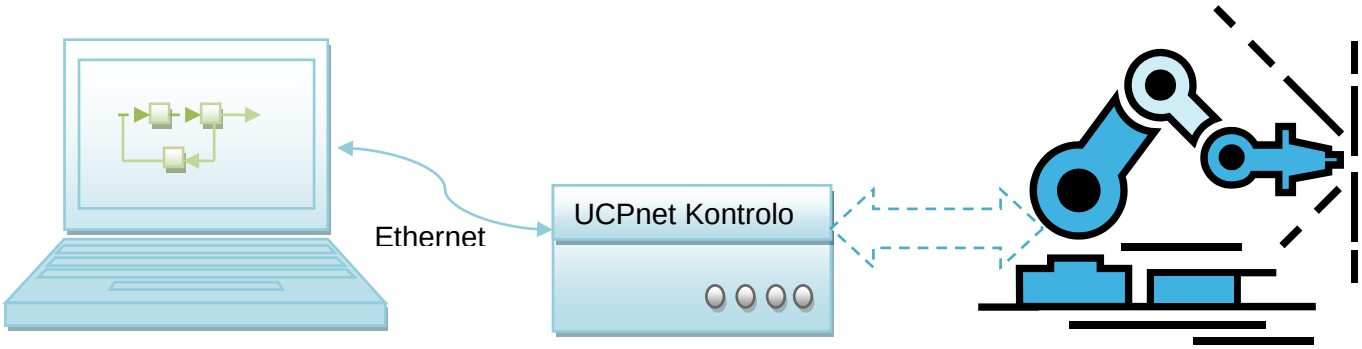


Figura 1: Esquema básico de la idea de la unidad de control

Capítulo 2: Meta y objetivos

2.1 Meta

Dotar al Laboratorio de Control Automático de herramientas que permitan a los estudiantes aprovechar mejor el curso y, a su vez, proteger los recursos que este tiene.

2.2 Objetivo general

Facilitar al estudiantado del curso de Laboratorio de Control Automático una herramienta que permita dedicarse solamente al diseño del controlador.

2.3 Objetivos específicos

1. Diseñar una Unidad de Control capaz de ser programada para controlar diferentes plantas del Laboratorio de Control.
2. Mantener la compatibilidad con la Interface de desarrollo (IDE) programada para el Laboratorio en semestres anteriores.
3. Establecer un protocolo de comunicación para la programación y configuración del dispositivo vía Ethernet sobre TCP/IP.

Capítulo 3: Marco teórico

3.1 Sistemas de control

Un sistema de control es un tipo de sistema que se caracteriza por la presencia de una serie de elementos que permiten influir en el funcionamiento del sistema. La finalidad de un sistema de control es conseguir, mediante la manipulación de las variables de control, un dominio sobre las variables de salida, de modo que estas alcancen unos valores prefijados.[18]

Idealmente, los sistemas de control tienen los siguientes requisitos:

- Deben ser robustos ante las perturbaciones y errores de modelado para que garanticen la estabilidad del sistema.
- Debe de evitar comportamientos bruscos e irreales, siendo eficiente según un criterio preestablecido.
- Operan en tiempo real.

Los sistemas de control se componen básicamente de los siguientes elementos:

- *Transductores/Sensores:* Dispositivos que permiten convertir las variables físicas en señales que pueden ser interpretadas por el controlador, normalmente señales eléctricas.
- *Controlador:* Es el encargado de procesar las señales emitidas por los transductores y calcular los nuevos valores para las variables de control.
- *Actuador:* Este es el mecanismo que ejecuta la acción que permite obtener los valores físicos de las variables de control, calculados por el controlador.

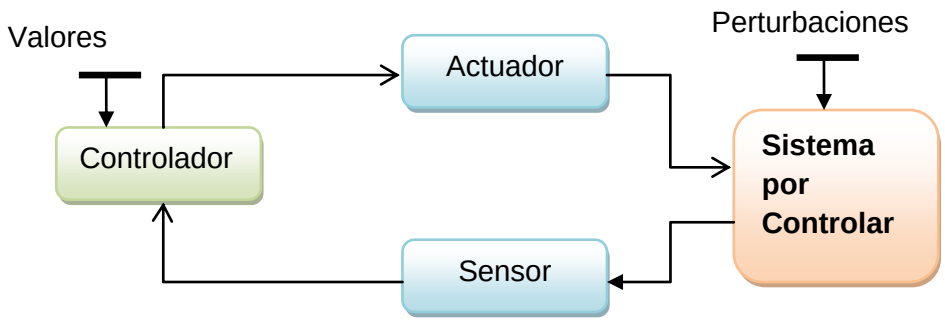


Figura 2: Esquema básico de un sistema de control.

3.2 Sistemas embebidos

Un sistema embebido, también llamado sistema empotrado, es un dispositivo de computación que se diseña para realizar tareas específicas a diferencia de un sistema de uso general, en donde el sistema intenta cubrir las necesidades de muchos tipos de tareas en un solo dispositivo.

Precisamente por la dedicación a unas cuantas tareas, los sistemas empotrados tienden a valores monetarios menores que los sistemas de uso general, además de que sus funciones dedicadas se optimizan, por lo que trabajara mejor que un sistema de uso general para realizar esa función.

Por lo general, estos sistemas dedicados, son dispositivos que necesitan que sus funciones se realicen en tiempo real y, por consiguiente, no se necesita que el sistema pierda tiempo en otras funciones que no sean necesarias para la realización de su misión.

Los sistemas embebidos poseen como núcleo central un microprocesador, un microcontrolador o un DSP, al que habitualmente se le conoce como la unidad central de procesamiento o CPU, por sus siglas en inglés. Además, incorporan una serie de interfaces de comunicación para distintos periféricos. Entre ellos, normalmente se encuentran los puertos USB, Ethernet, I^2C, SPI, GSM, CAM, etc.[9]

3.2.1 El microcontrolador PIC32

El PIC32 es el nombre de la familia más avanzada de microcontroladores, que incorpora un núcleo con un procesador MIPS® M4K, que tiene capacidad de operar en una frecuencia máxima de 80 MHz.

Este microcontrolador posee casi todos los puertos de las familias de microcontroladores anteriores. Cuenta con 8 canales de acceso directo a memoria (DMA), 10/ 100 Ethernet Mac, con 2 canales exclusivos de DMA con Interface MII/RMII, 5 canales de PWM, 16 relojes, 16 canales de entrada analógica (ADC), un reloj y calendario en tiempo real (RTCC), hasta 5 puertos de I^2C , hasta 4 SP1, 16 puertos paralelos, 6 canales para transmisión serial UART. [11][12]

3.2.1.1 *Procesador MIPS® M4K®*

Este procesador es plenamente compatible con el juego de instrucciones para MIPS32, arquitectura Hardvard, que sigue la filosofía RISC y que posee un *pipeline* de 5 etapas, además incorpora las terminales del EJTAG (Enhaced JTAG), una modificación del estándar IEEE 1149.1, que permite realizar pruebas en tiempo real sobre el integrado[14]. El esquema de bloques de este procesador se muestra en la Figura 3.

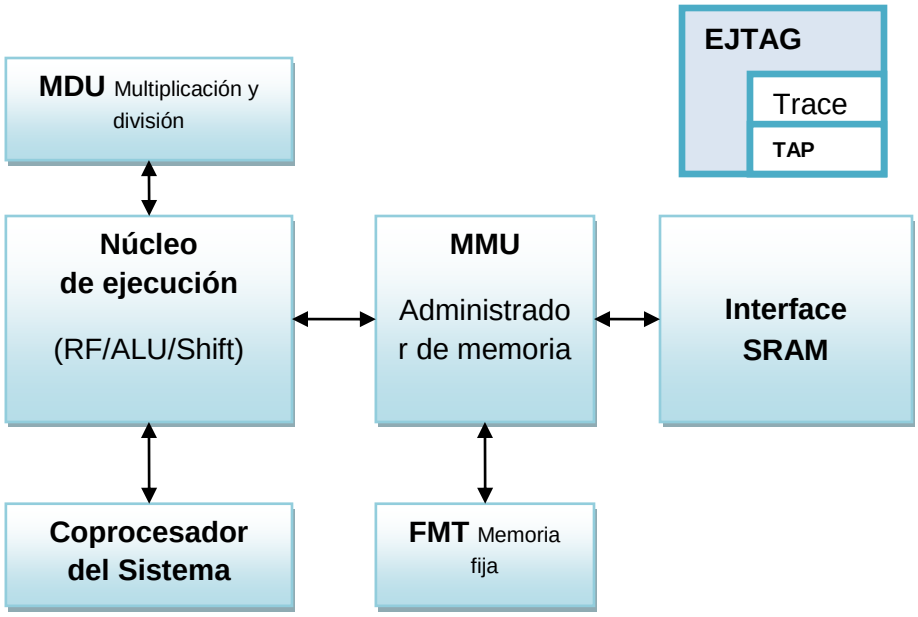


Figura 3: Diagrama de bloques del MIPS(R) M4K(R)

Y en la Figura 4 se muestra las 5 etapas del *pipeline* de un procesador MIPS32

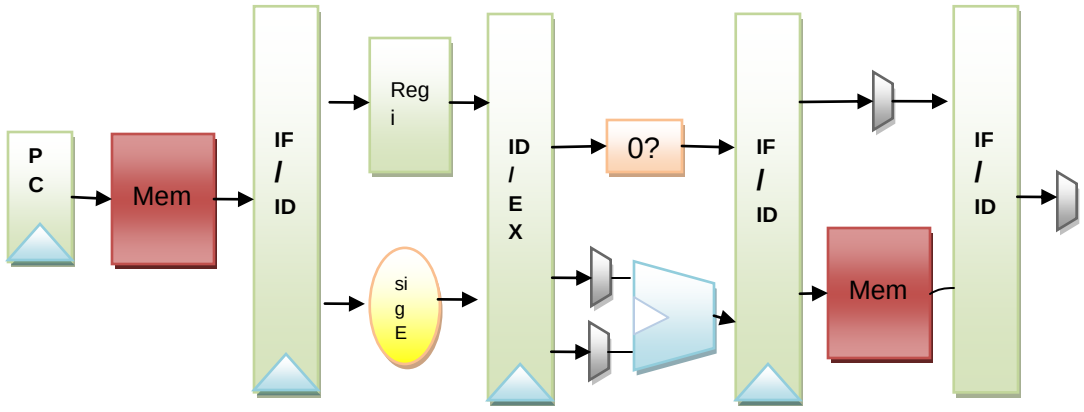


Figura 4: Etapas del *pipeline* de un MIPS32

Microchip Inc.® creó una placa de desarrollo, donde integran el PIC32 junto con una serie de componentes que permiten al desarrollador iniciar directamente con el código. Esta placa posee, además del PIC32, un PIC18 que funciona como dispositivo programador y depurador del PIC32, funciones que se realizan mediante un puerto USB con conector MINI B. También incluye un conector de USB Micro AB y otro A, para los proyectos que necesiten USB, así como una terminal de Ethernet RJ-45 para proyectos en red y un conector Hirose de 120 pines, que le permite a esta placa incorporarse a otros diseños como la Expansion Board (también de Microchip) y de esta manera tener acceso a todos las señales que brinda el microcontrolador.

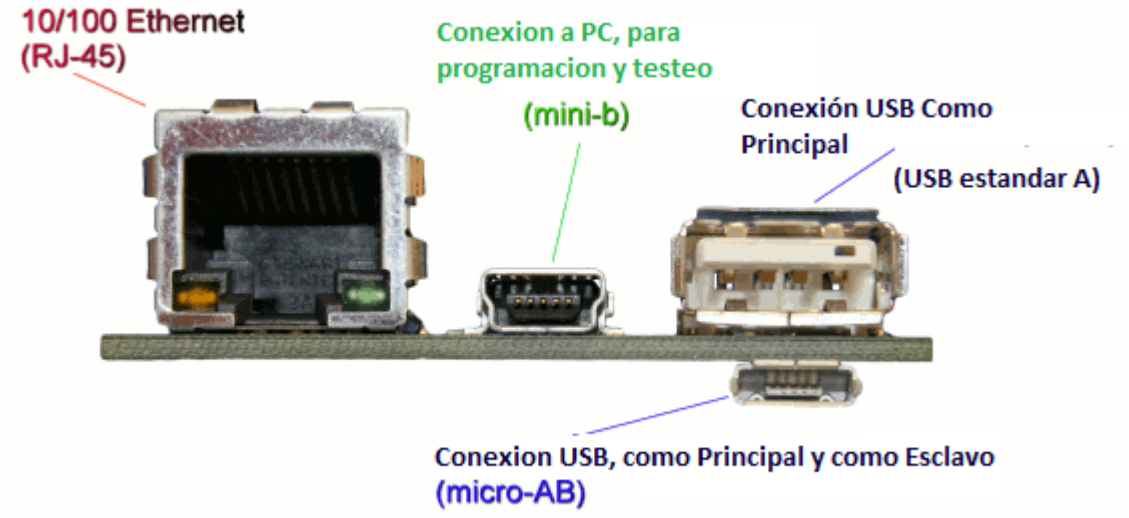


Figura 5: Panel frontal del Ethernet Starter Kit[12]

3.3 Sistemas operativos en tiempo real (RTOS)

Los sistemas operativos en tiempo real (RTOS, por sus siglas en inglés) tienen como meta administrar aplicaciones que necesitan ser ejecutadas casi de forma instantánea. Dada esta exigencia, según la cual el sistema debe gastar la menor cantidad de tiempo, se hace necesario que el sistema sea determinista y, de esta forma, optimizar los tiempos de procesamiento de las interrupciones. (Por ejemplo, deshabilitar ciertas interrupciones por un tiempo, si se sabe que no se ejecutaran en dicho lapso). [20]

Por lo general, este tipo de sistemas operativos son de categoría multitarea. Para poder ejecutar varias tareas al mismo tiempo, de manera tal que sus ejecuciones parezcan instantáneas y simultáneas, se utilizan interrupciones por intervalos de tiempo preestablecidos, de modo que la información que iba en un proceso se almacena, así como la línea de código que ejecutaba. Se empieza a ejecutar otra aplicación y cuando transcurre el mismo lapso de tiempo ocurrirá lo mismo con este proceso (usa el modo apropiativo, o pre-emptive, y no por lotes, o modo batch).[20]

Existen tres tipos en que el sistema apropiativo puede realizar la multitarea:

- Cooperativa: Los procesos deciden cuando quieren ceder el control del sistema, por lo que los tiempos no son regulares.
- Preferente: El sistema operativo es el que reparte los tiempos de ejecución de cada tarea y decide en qué momento la pone en receso para realizar otra operación. Es decir, el sistema operativo tiene el absoluto control de la distribución del tiempo de ejecución de las tareas.

- Real: Este tipo de multitareas se lleva a cabo en sistemas con varios núcleos de procesador, distribuyendo las tareas entre los procesadores. Por consiguiente, las tareas se ejecutan en el mismo tiempo realmente y no mediante una simulación. No obstante, este tipo se combina con cualquiera de los otros dos tipos, para los casos en que haya más tareas que núcleos de procesador.

En el caso de que el sistema sea preferente, existen varias formas en que se distribuyen los lapsos de tiempo que le corresponda a cada proceso, lo cual se hace mediante la planificación de prioridades. Entre estas estrategias de planificación, quizás la más fácil de implementar es la planificación circular Round-Robin, el cual consiste en distribuir de manera idéntica los tiempos de ejecución de las tareas y procesos que posean la prioridad activa más alta en ese momento y, por consiguiente, aquellas tareas que tengan una prioridad más baja no se ejecutarán. Además, dentro de los círculos de ejecución en un mismo nivel, la tarea que ingresó de primero será la que se ejecute primero y la que ingresó de última se ejecutará de última (FIFO, *First In First In Out*).[20]

En la Figura 6, se muestra un esquema de tiempos para cuatro aplicaciones, tres de ellas en la máxima prioridad activa y otra en una prioridad inferior. Después de un tiempo, la tarea que se encontraba en mínima prioridad, cambia su valor de prioridad a un nivel de mayor importancia que las otras tres.

Figura 6: Esquema de distribución de tiempos en una planificación Round-Robin

3.3.1 FreeRTOS

FreeRTOS es un sistema operativo en tiempo real con portaciones a múltiples arquitecturas bajo licencia GPL, por lo que se ha empleado en bastantes diseños de sistemas embebidos.

Este sistema operativo está diseñado para ser lo más pequeño posible, de tal modo que su núcleo está constituido por tan solo tres archivos de C, por lo que es un

núcleo microkernel. Además, existe la documentación necesaria para la realización del sistema UCPnet y ejemplos de uso y programación en toda la Web, razones por las que se eligió para ser el sistema operativo de la Unidad Controladora de Procesos.[4]

A pesar de ser un sistema operativo en tiempo real, se puede realizar la planificación de tareas de modo cooperativo y no solamente preferente como sería de esperar. Otra característica importante es que permite la planificación de corrutinas, que son tareas bastante livianas puesto que comparten parte de su código y datos entre ellas. [4]

3.4 Circuitos acondicionadores de señal (AFE CAS)

Un circuito acondicionador de señal es un circuito analógico que convierte su entrada en otro nivel de tensión o corriente para poder ser interpretado por otro circuito. Normalmente, esta conversión genera una señal lineal y típicamente se utiliza para poder representar los bajos valores de tensión que brindan los sensores en valores más altos; sin embargo, también existen ocasiones en que se desean acoplar dispositivos con diferentes rangos, por lo que el circuito acondicionador de señal puede tener función de atenuador como de amplificador.[8]

En el caso de que se requiera una señal con relación lineal a la salida, se procede a generar una ecuación de la línea recta, donde el eje dependiente es el eje de entrada y el independiente es el eje de salida. De este modo, se obtienen dos valores, la pendiente y el valor de la intersección con el eje Y. Esto se muestra en el siguiente grupo de ecuaciones.

$$m = \frac{Vsalida_{max} - Vsalida_{min}}{Ventrada_{max} - Ventrada_{min}}$$

$$b = Vsalida_{max} - m * Ventrada_{min}$$

$$Vsalida = m * Ventrada + b$$

Ecuación 1: Grupo de ecuaciones para un sistema de acondicionamiento de señales.

En la Figura 7, se muestra la función de uso del acondicionador de señal, el punto anaranjado correspondiente al valor máximo en ambos gráficos, mientras que el punto rojo indica el mínimo valor. Ambos puntos se encuentran en el mismo valor del eje dependiente de los dos gráficos.

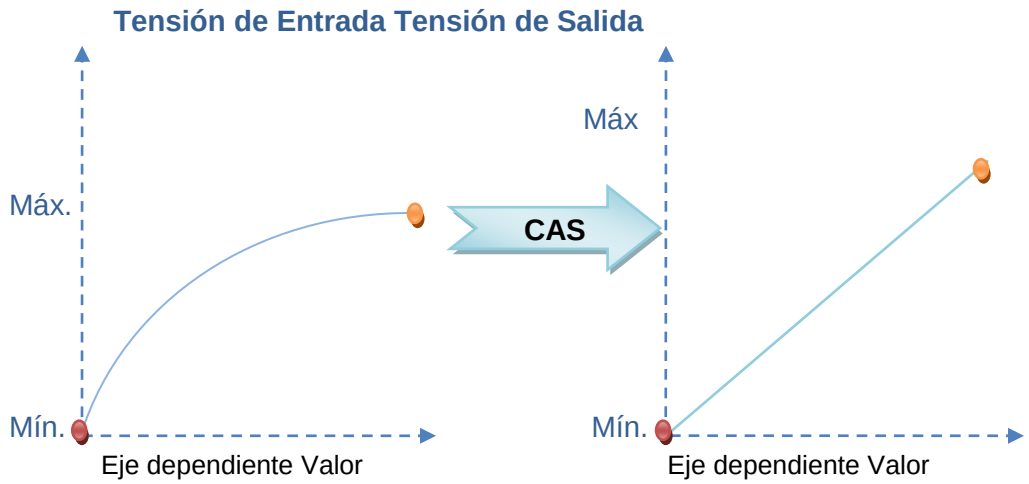


Figura 7: Ejemplo de uso de un acondicionador de señal.

3.5 Protocolos de comunicación sobre Ethernet

Los protocolos de comunicación son reglas que garantizan que dos sistemas se puedan comunicar y entenderse entre sí. De esta forma, el protocolo debe indicar el lenguaje con el que se comunican (la representación de datos), los métodos para detectar errores, la autentificación y la identificación de sus participantes, entre otros aspectos.

Los protocolos de red pueden darse en cualquiera de las capas del modelo TCP /IP, creado en la década de 1970 por la DARPA (agencia del Departamento de Defensa de los Estados Unidos) y es, junto con OSI, una arquitectura de protocolos determinante y básica en el desarrollo de los estándares de comunicación. Es la arquitectura más adoptada para la interconexión de sistemas. Al contrario de lo que ocurre con OSI, el modelo TCP/IP es *software*, es decir, es un modelo para ser implementado en cualquier tipo de red. Facilita el intercambio de información independientemente de la tecnología y el tipo de subredes por atravesar, proporcionando una comunicación transparente por medio de sistemas heterogéneos. Por todo esto, TCP/IP no define una capa física ni de enlace. Este protocolo define solamente tres capas que funcionarán en los niveles superiores de las capas físicas y de enlace para hacerlo así un modelo independiente del *hardware* en el que se implemente. Este modelo se muestra en la Figura 8.[13][19]

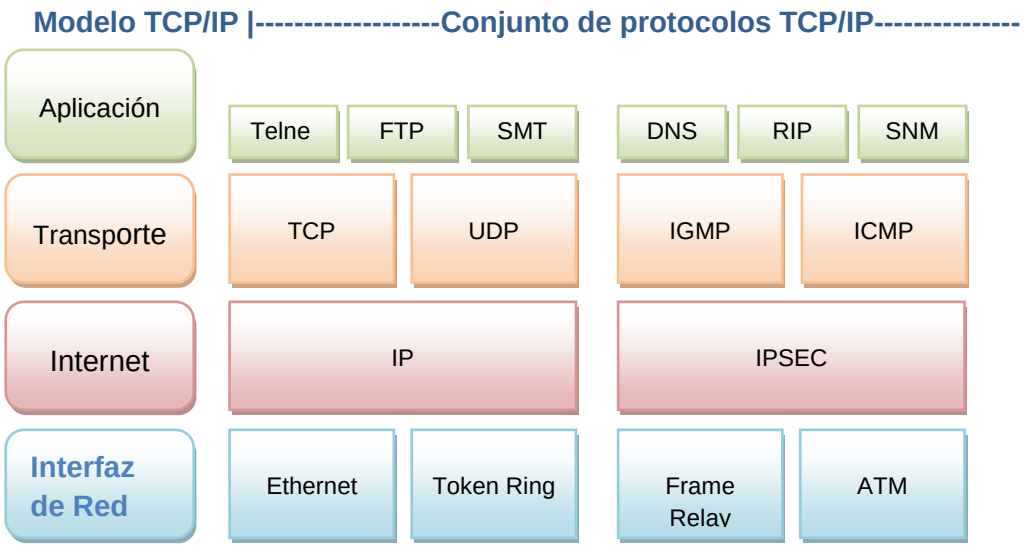


Figura 8: Modelo TCP/IP y conjunto de protocolos TCP/IP

En la Tabla 1 se describe cada una de las capas del modelo TCP/IP y se muestran algunos ejemplos de protocolos que se encuentran en cada una de las capas.

Tabla 1: Descripción y ejemplos de los niveles del modelo TCP/IP [13]

Capa	Descripción	Protocolos
Aplicación	Define los protocolos de aplicación TCP/IP y cómo se conectan los programas de *host* a los servicios del nivel de transporte para utilizar la red.	HTTP, Telnet, FTP, TFTP, SNMP, DNS, SMTP, X Windows y otros protocolos de aplicación
Transporte	Permite administrar las sesiones de comunicación entre equipos *host*. Define el nivel de servicio y el estado de la conexión utilizada al transportar datos.	TCP, UDP, RTP
Internet	Empaqueta los datos en datagramas IP, que contienen información de las direcciones de origen y destino utilizada para reenviar los datagramas entre *hosts* y a través de redes. Realiza el enrutamiento de los datagramas IP.	IP, ICMP, ARP, RARP
Interfaz de red	Especifica información detallada de cómo se envían físicamente los datos a través de la red, que incluye cómo se realiza la señalización eléctrica de los bits mediante los dispositivos de *hardware* que conectan directamente con un medio de red, como un cable coaxial, un cable de fibra óptica o un cable de cobre de par trenzado.	Ethernet, Token Ring, FDDI, X.25, Frame Relay, RS-232, v.35

3.5.1 TCP/IP

Los de protocolos sobre TCP/IP descrito en la estandarización RFC1122 es quizás la familia de protocolos de red más utilizada. Son protocolos que se encuentran sobre la capa de aplicación, teniendo en base en capa de transporte al protocolo TCP y en nivel de red al protocolo IP.

Los protocolos de aplicación sobre TCP (siglas en inglés de Protocolo con Control de Transmisión) se caracterizan por garantizar que la transmisión enviada fue exitosamente recibida por el escucha; esto es que todos los datos fueron recibidos en el mismo orden que fueron enviados.

Las conexiones dentro de este modelo de protocolo se hacen de extremo a extremo (*Host to Client*), lo cual se conoce como circuitos virtuales. Esto permite que un computador tenga más de una conexión TCP abierto para diferentes protocolos sobre la capa de aplicación. El ordenador, para conectarse a un circuito, lo hace mediante un número de puerto diferente para cada aplicación.[19]

Para establecer una conexión, se utiliza una negociación de tres pasos (*Three-way-handshake*). Por lo general, uno de los terminales (servidor) abre un puerto TCP, y queda a la espera de nuevas conexiones. Cuando otra termina (cliente) establece una apertura, envía un paquete de sincronización “SYN” a la primera terminal y, en este momento, inicia la negociación. El servidor revisa si el puerto está escuchando y, de ser así, envía un acuse de sincronización “SYN/ACK” y el cliente responderá con un paquete de acuse “ACK”. En la se muestra la Figuera 9 negociación de tres pasos.

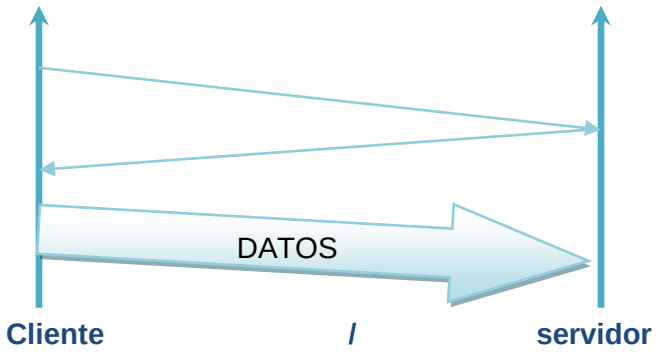


Figura 9: Negociación en tres pasos del protocolo TCP

Es importante tener en cuenta que este tipo de protocolos realiza la desconexión de manera independiente (servidor/cliente); por consiguiente, algún error en alguna de sus partes podría dejar la conexión “medio abierta”. Esta desconexión se realiza en cuatro pasos, dos por cada parte de la conexión, donde un paso es la instrucción de fin

y el otro es la señal de sincronización. En la Figura 10, se muestra el cierre de conexión que presenta el estándar.

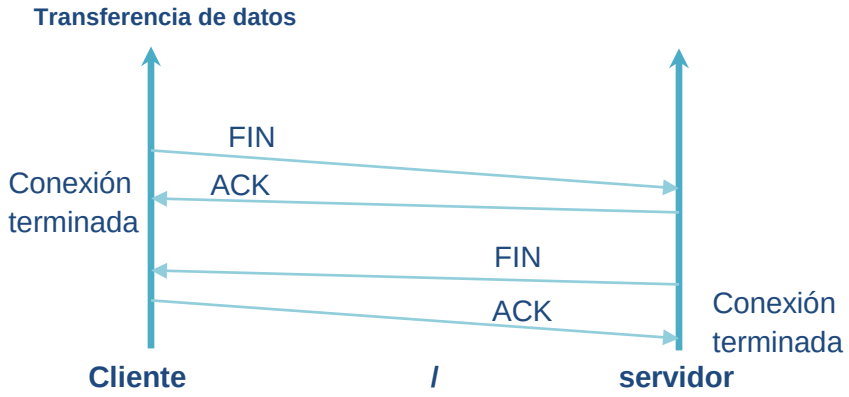


Figura 10: Fin de conexión TCP

3.5.2 UDP

UDP –descrito en la definición RFC 768– es un protocolo de la capa de transporte en el modelo OSI utilizado para el trasiego de información donde no es importante que llegue el 100% de los paquetes ni el orden en que se reciban. Este intercambio de información lo hace mediante paquetes comúnmente llamados datagramas.

Las conexiones entre dos protocolos de aplicación basados en el protocolo UDP no requieren, a diferencia del protocolo TCP/IP, del establecimiento de una conexión previa. Esto gracias a que en la cabecera del datagrama se incorpora toda la información necesaria para el direccionamiento al que debe ser suministrada esta información.

Debido a que este protocolo no necesita una conexión previa ni debe estar reenviando paquetes de información que se hayan corrompido, es muy utilizado en protocolos de audio y video. Esto porque en estas aplicaciones la pérdida de un bajo porcentaje de datos será irrelevante para mostrar la información total deseada. Sin embargo, sí es fundamental que se pierda el menor tiempo posible con el fin de mostrar la información de manera instantánea.

3.5.3 RTP

RTP es un protocolo del nivel de transporte sobre UDP, utilizado para la transferencia de datagramas en tiempo real, como en el caso de voz sobre IP (VoIP) o en videoconferencias.

El protocolo se define en la normativa RFC 3550 y posee la estructura que se puede observar en la Tabla 2.

Tabla 2: Estructura del encabezado de RTP [21]

Byte 0				**Byte 1**		**Byte 2**	**Byte 3**
V	R	X	CC	M	PT	Número de Secuencia	
Estampado de tiempo							
Sincronización de la fuente							
Contenido en la fuente							
Extensión del encabezado (de ser necesario)							
Datos							

Donde V (1 bit) es la versión del protocolo, actualmente 2. R (1 bit) es de relleno bit, X (1) indica si se usará la extensión de encabezado. CC (4 bits) número de identificadores para el contenido de la fuente.

Esquema de programa BPC

La Escuela de Ingeniería en Electrónica del Instituto Tecnológico de Costa Rica hace varios años desarrolló un sistema similar al UCPnet, creación del ingeniero Daniel Castro, egresado de esta escuela. En la primera etapa, el sistema se programaba mediante tramas que contenían código hexadecimal, las cuales representaban una línea de instrucción. La unidad enviaba la misma instrucción al computador como modo de comprobación de que esta trama había llegado exitosamente. En la Figura 11, se muestra como se realiza la transferencia entre el ordenador y el sistema BPC.

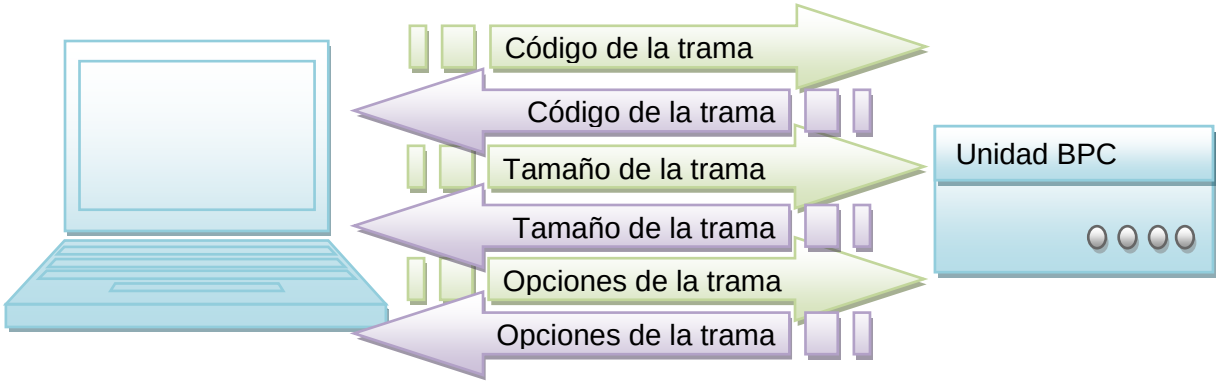


Figura 11: Transferencia de un comando al sistema BPC [7]

Como segunda etapa, se le encargó al ingeniero Óscar Caravaca elaborar una interface de desarrollo gráfico que permitiera al estudiante desarrollar su modelo de control sin entrar en la complejidad de programar la unidad BPC.

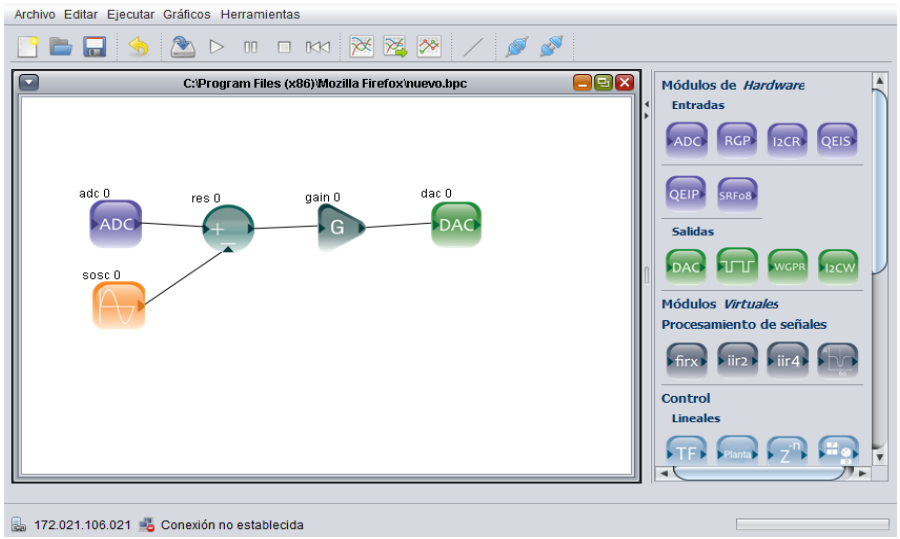


Figura 12: Entorno de desarrollo para las unidades de control [5]

Además del entorno gráfico, el ingeniero Caravaca desarrolló un lenguaje intermedio que permitía leer legiblemente las sentencias de configuración del BPC. A este lenguaje le definió una estructura como se muestra en el Código 1, la cual es la usada por el sistema UCPnet Kontrolo.

```
MOD //Sección donde se declaran los módulos virtuales.
begin
tipoVirtual1, 1; //Se declara 1 modulo del tipo virtual 1
tipoVirtual2, 1; //Se declara 1 modulo del tipo Virtual 2
end
CONFIG //Sección donde se parametrizan todos los módulos que lo requieran
begin
TipoVirtual1 0, param1,param2,paramn; //configura los parámetros de la instancia 0 de los módulos virtuales tipo 1
TipoVirtual2 0, param1,param2,paramn; //configura los parámetros de la instancia 0 de los módulos virtuales tipo 1
end
NETLIST //Sección donde unen las entradas y salidas
begin
tipoVirtual1, 0, 1, tipoVirtual2, 0, 1; //la salida 1 del primer viturtual1 se conecta a la entrada 1 del primer virtual2
```

Código 1: Estructura del intérprete intermedio del BCP

Capítulo 4: Diseño de etapas de la unidad de control

4.1 Tarjeta madre

La placa base de este sistema embebido está diseñada de manera modular, de tal forma que, si se dañara uno de los módulos de las señales de entrada y salida analógica, solo se tendría que cambiar este dispositivo y no todo el sistema. Estos módulos se conectan mediante el uso de pines convencionales, lo que facilita la reparación del sistema.

Se utilizaron diferentes tipos de conectores para los distintos tipos de periféricos, con el fin de obtener la rápida diferenciación de puerto por parte de los usuarios. Además, se utilizaron conectores que posean una única forma de conexión de todas sus señales y de tal modo evitar que el usuario conecte un periférico con una polaridad diferente, o con las señales erradas.

El diagrama de esta placa se muestra en la Figura 13, seguidamente en la Tabla 3 se indican los nombres de sus partes.

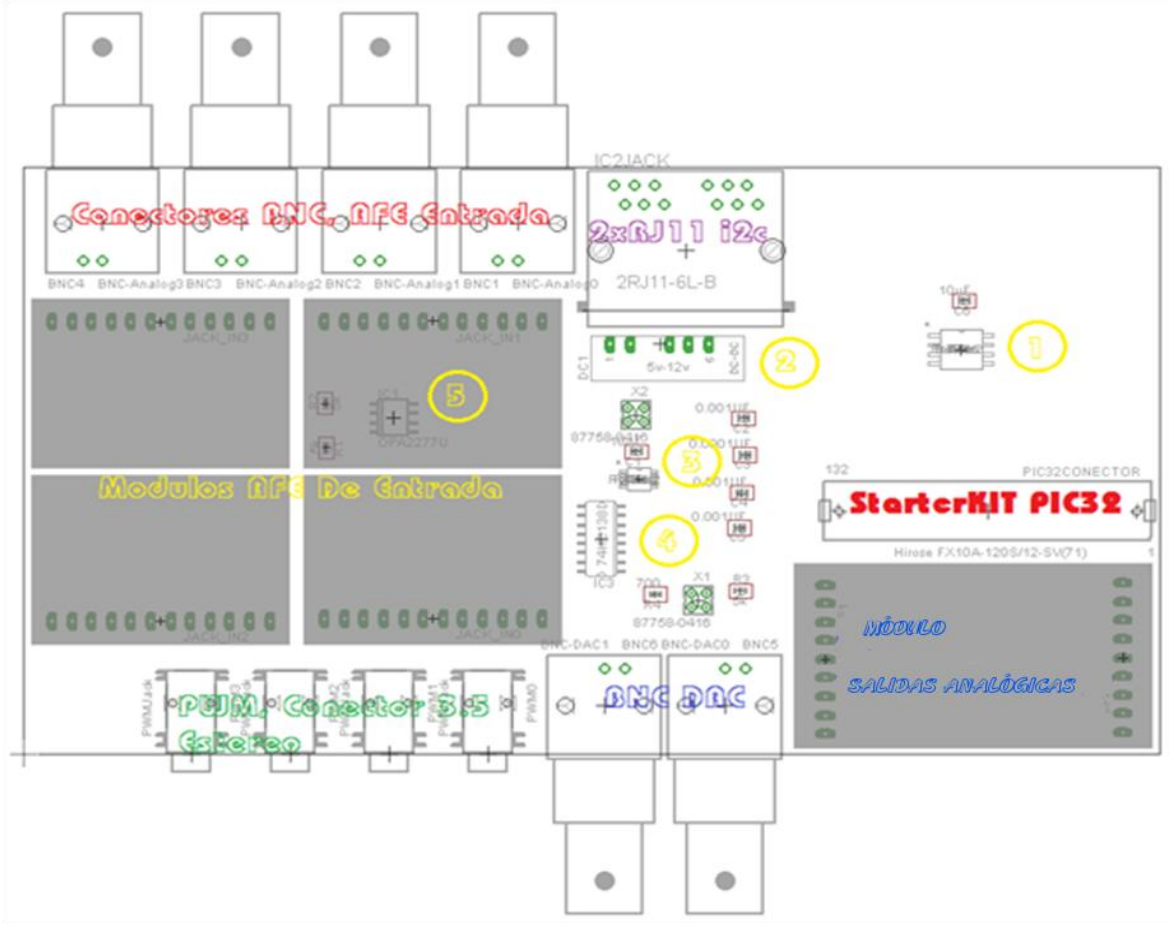


Figura 13: Placa madre del Sistema UCPnet Kontrolo V 0.1

Tabla 3: Componentes esenciales en la placa base del UCPnet Kontrolo

Indicación	Componente	Función
1	24AA1025	EEPROM para salvar datos y constantes
2	RKZ-0512	Convertidor DC-DC de 5V a 12V
3	MCP4728	DAC 4 canales como offset de AFE entrada
4	74HC138	Decodificador para selección de módulo
5	OPA2777	Inversor de 5V con el fin de obtener -5v

4.2 Protocolo Konekto

Este es un protocolo de comunicación creado para el presente proyecto, con el objetivo de que periféricos y transconductores interactúen con la unidad de control universal, y de este modo, expandir las aplicaciones a las que se puede integrar este sistema.

El protocolo Konekto utiliza como base al protocolo I^2C. Se hizo esta elección por el sistema serial que posee y la habilidad de conectar varios dispositivos sobre el mismo bus de datos. Sin embargo, se ha extendido el protocolo, tanto a nivel físico como a nivel de programación.

A nivel físico, se especifico el uso del conector RJ45 (conector habitual de líneas telefónicas) y se usaron las seis señales que incorpora esta clavija. La señal relacionada a cada señal se muestra en la Figura 14:

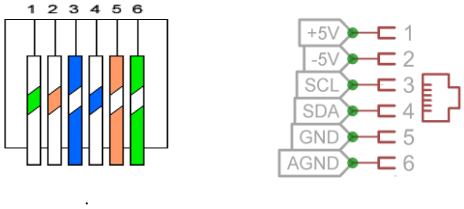


Figura 14: Señales de conexión del protocolo Konekto

Como se observa, el protocolo, además de poseer las dos señales necesarias del bus de datos I^2C, también incluye las señales a tierra del sistema y las señales de alimentación de ±5V.

La manera en que se realiza la comunicación, se interpretan los datos y se realizan las funciones de rutina es mediante mecanismos de programación que siguen el flujo de datos de la Figura 15.

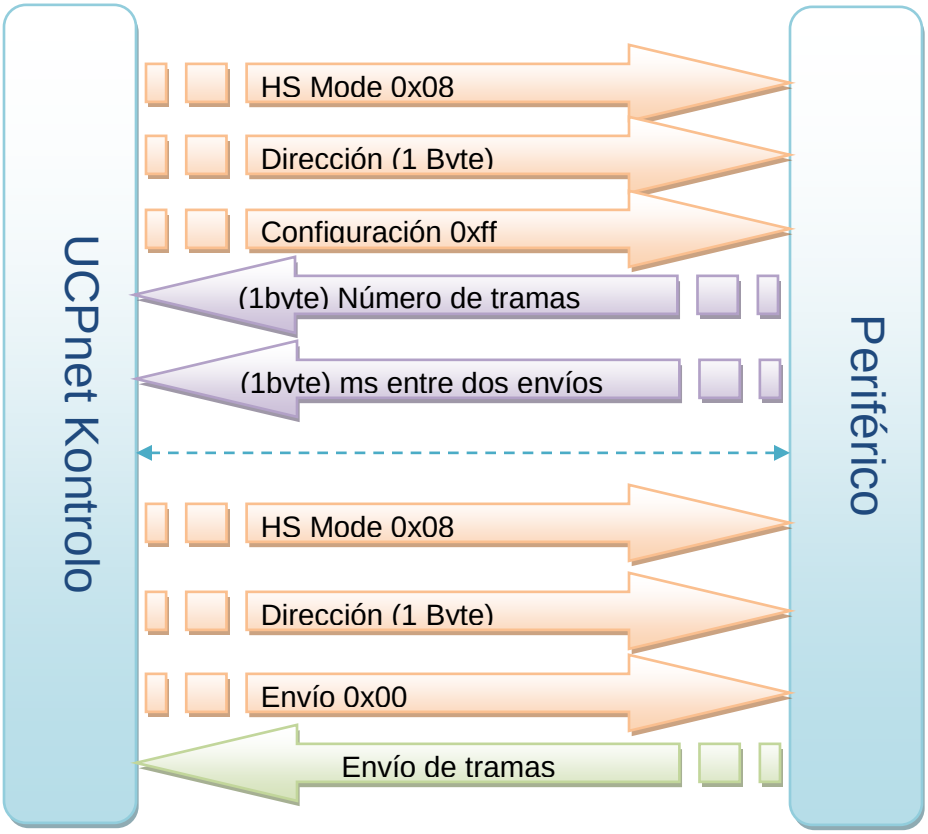


Figura 15: Flujo de datos de la comunicación de Konekto

Los requerimientos mínimos que debe tener el dispositivo para ser compatible con el protocolo se enumeran en la siguiente lista:

- Las transferencia de datos se debe de realizar a una velocidad mínima de 3.4Mbits/s (HS MODE).
- La dirección tiene que estar en formato de 8 bits.
- El periférico no deberá tener ninguna de las siguientes direcciones: 0x00, 0x0C, 0xBX, 0xFA.
- El tiempo de repetición entre el envío de una serie de tramas y otra debe ser un valor entero entre 1 y 255 milisegundos.
- El número de tramas por enviar debe estar comprendido en un valor entero entre 1 y 255.
- Las tramas de datos deben estar en el formato de punto flotante definido en la norma IEEE 754.

4.3 Etapa de energía

El sistema utiliza como fuente de energía principal, la señal de alimentación de 5 voltios suministrada por la conexión USB mini b del "StartKit". Este último posee un regulador de tensión que establece un nivel de referencia de 3.3 voltios, nivel de tensión necesario para el correcto funcionamiento del microcontrolador y máxima tensión de entrada en los canales analógicos de este.

Además de las señales de +5V y +3.3V, se necesitan tres tensiones adicionales, -5V para aquellos circuitos con requerimientos bipolares, y ±12 para la amplificación de las señales de salida analógicas. Estas tres referencias se generarán mediante un conversor DC-DC que eleva la tensión de 5V a ±12, y se utilizan reguladores de tensión para obtener las tensiones de ±5V.

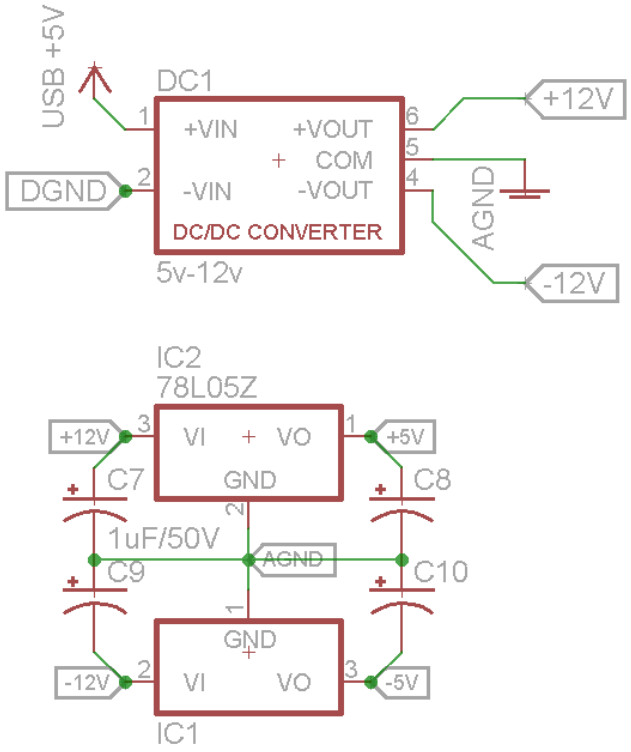


Figura 16: Generación de tensiones de ±5V y ±12V

4.4 Entradas analógicas

Los módulos de entrada analógica se diseñaron para recibir señales en modo diferencial. Esto porque a pesar del blindado del cable coaxial que se utiliza en la transmisión de esta señal, puede existir una señal de ruido que podría afectar las mediciones de las señales cuando estas se encuentren en el rango de los milivoltios (mV). Por esta razón, se utiliza un circuito acondicionador de señal diferencial Este sistema acondiciona señales diferenciales y, por consiguiente, se utiliza un amplificador de instrumentación. El esquema se presenta en la Figura 19.

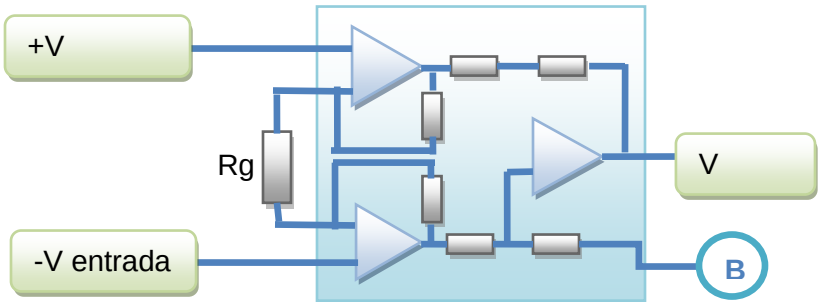


Figura 17: Acondicionador de señal diferencial

La ganancia del amplificador de instrumentación es la pendiente “m” del grupo de ecuaciones 1 y típicamente se describe mediante la Ecuación 2.

$$G = 1 + \frac{R_{interna}}{Rg}$$

Ecuación 2: Ganancia típica de un amplificador de instrumentación

Además, se implemento un filtro pasa bajas para impedir el solapamiento que se leería en la señal muestreada a través del ADC. Se escogió un filtro Bessel con estructura Sallen-key de primer orden, que fue utilizada en el BPC del Ing. Molina que obedece a la configuración que se muestra en la Figura 18 y sus componentes pasivos se calculan mediante el grupo de ecuaciones de Ecuación 3.

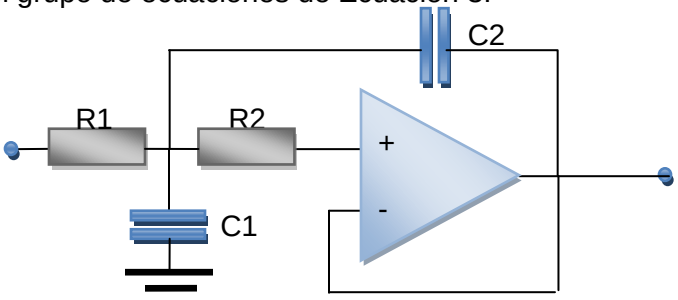


Figura 18: Filtro activo pasa-bajas con estructura Sallen-Key de primer orden [10]

$$f_0 = \frac{1}{2\pi\sqrt{R_1 R_2 C_1 C_2}}$$

$$Q = \frac{\sqrt{R_1 R_2 C_1 C_2}}{C_2(R_1 + R_2)}$$

Ecuación 3: Grupo de ecuaciones para determinar valores de un filtro Sallen-Key de primer orden.

La Figura 19 muestra el esquema utilizado para los módulos de entrada.

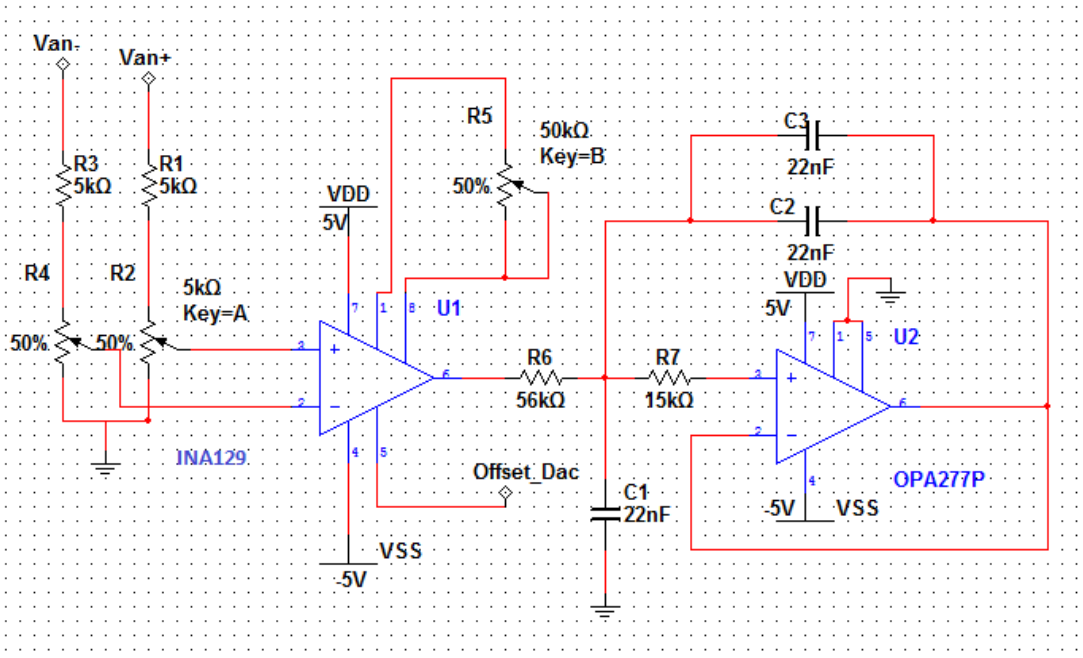


Figura 19: Esquema del módulo de entradas analógicas

Como se observa en la Figura 19, el amplificador de instrumentación utilizado es el INA129, el cual tiene un factor de rechazo de modo común de 120db y su ecuación de amplificación se muestra a continuación.[17]

$$\boldsymbol{G = 1 + \frac{50k\Omega}{Rg}}$$

Ecuación 4: Ganancia del amplificador de instrumentación INA128

Como se observa en la Ecuación 6, el amplificador de instrumentación no puede realizar operaciones de atenuación, función que se hace necesaria para las señales en rango mayor a 3.3 voltios (tensión máxima del eje dependiente del CAS), por lo que se recurre a un divisor de tensión con una de sus resistencias controladas digitalmente para cada una de las entradas.

Para controlar estas resistencias digitalmente, se utilizaron los potenciómetros digitales de la serie AD528X, que poseen 256 pasos seleccionados mediante comunicación I^2C. Para las resistencias variables del divisor, se utilizó un único circuito integrado que posee dos potenciómetros de esta serie (AD5282) con una resistencia máxima de 200kΩ, mientras que para seleccionar el valor de la resistencia de ganancia R_g, se seleccionó el circuito integrado de esta serie que contiene solo un potenciómetro (AD5280) con una resistencia máxima de 50kΩ.

Estos potenciómetros han sido conectados de forma tal que su resistencia está dada por la Ecuación 5. [3]

$$R_{WA}(D) = R_{interna}\left(\frac{256 - D}{256}\right)$$

Ecuación 5: Resistencia del potenciómetro entre A y W

Utilizando la Ecuación 5 para determinar R_g en la Ecuación 4, obtenemos una ganancia diferencial controlada digitalmente con la relación que se muestra en la Ecuación 6.

$$G = 1 + \frac{256}{256 - D}$$

Ecuación 6: Ganancia del amplificador de instrumentación controlada digitalmente

El nivel de CC se genera mediante del convertidor digital/analógico que se encuentra en el esquema de la placa base de la Figura 13.

Como solo se puede tener 4 direcciones para los dispositivos de la familia AD825X y se tienen 9 integrados de esta serie, se recurrió a dejar la patilla AD0 del IC como señal de selección del módulo y la patilla AD1 se utiliza en el módulo de entrada para seleccionar si los datos enviados son para el AD5282 (la patilla está conectada a tierra) o para el AD5280 (la patilla está conectada a VDD).

Los rangos para los cuales se puede configurar estos módulos se muestran en la Tabla 4.

Tabla 4: Rangos de medición de los módulos de entrada (rangos definidos para el BPC)

Opción	Rango bipolar	Rango unipolar
1	±10V	0V-10V
2	±5V	0V-5V
3	±2V	0V-2V
4	±1V	0V-1V
5	±500mV	0V-500mV
6	±200mV	0v-200mV
7	±100mV	0V-100mV
8	±50mV	0V-50mV

4.5 Salidas analógicas

La etapa de salida utiliza acondicionadores de señal modificados como el que se muestra en la Figura 20.

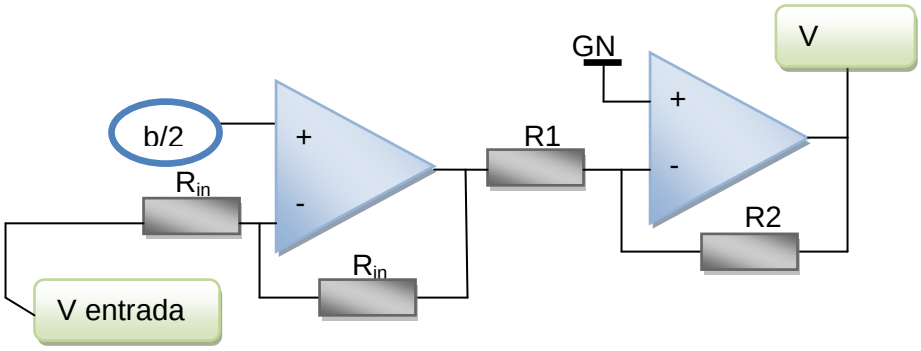


Figura 20: Acondicionador de señal básico modificado.

En la Figura 21, se muestra el esquema de implementación del acondicionador de la Figura 20.

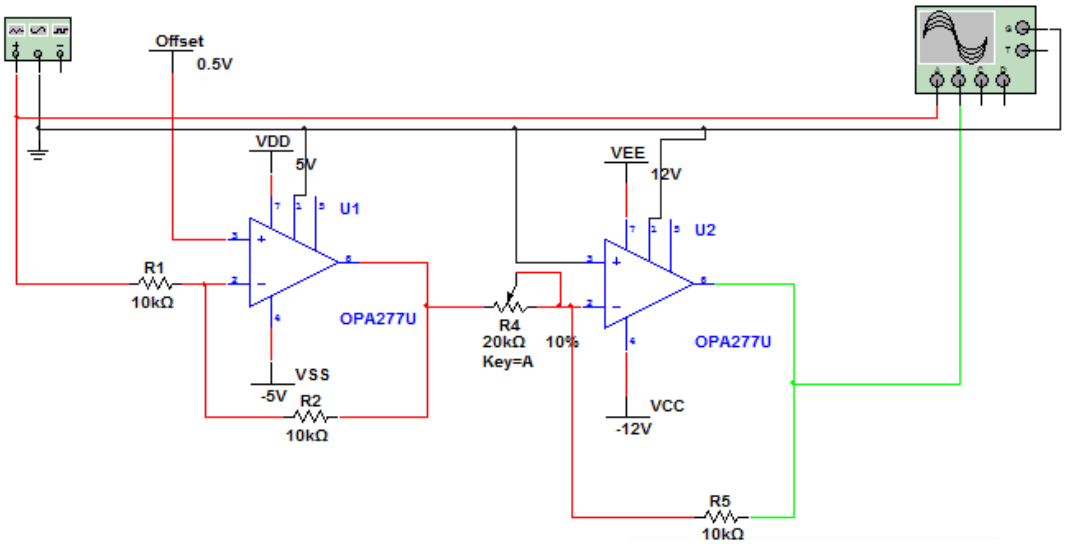


Figura 21: Esquema del módulo de salidas analógicas

Ambas salidas analógicas se encuentran en un único modulo, que contiene como corazón de este al convertidor digital análogo MCP4728 de microchip. Y como se observa en el Anexo 2, las dos primeras señales de salida de este integrado son las señales generadas por el flujo de control programado, mientras que las dos últimas son el *offset* que se utilizará en el acondicionador de señal.

En el primer amplificador se suma el inverso de señal de entrada con el doble de la señal de *offset*, con lo cual, si no hay *offset* se obtiene una señal en el rango de 0V a -2.048V o de -1.024 a 1.024V si la señal de *offset* es de 512mV. Mientras, el segundo amplificador se encarga de escalar e invertir esta señal para que se encuentre entre los valores que se muestran en la Tabla 5.

Tabla 5. Rango de salidas analógicas del BPC y el UCP

Opción	Rango bipolar	Rango unipolar
1	±10V	0V-10V
2	±5V	0V-5V
3	±2V	0V-2V
4	±1V	0V-1V

Capítulo 5: Distribución de tareas y funciones

5.1 Implementación del RTOS

Para el desarrollo del proyecto, solo se usaron unas cuantas instrucciones del sistema operativo, las cuales explicaremos a continuación.[4]

xTaskCreate: Con esta función se crean los espacios en memoria y se agenda las tareas del sistema.

```
portBASE_TYPE xTaskCreate(
pdTASK_CODE pvTaskCode,
const portCHAR * const pcName,
unsigned portSHORT usStackDepth,
void *pvParameters,
unsigned portBASE_TYPE uxPriority,
xTaskHandle *pvCreatedTask
);
```

Explicación de los parámetros

- *pvTaskCode:* Puntero al código de la tarea. Esta tarea debe de contener un ciclo infinito.
- *pcName*: Nombre de la tarea, utilizado para hacer pruebas y comprobaciones (*debugger*).
- *usStackDeph:* Valor que contiene el tamaño de pila que se le asignará a dicha tarea.
- *pvParameters:* Parámetros que se le pueden pasar a la función. Estos parámetros carecen de tipo y, por consiguiente, lo que se pasa es la dirección de almacenaje de dicho parámetro.
- *uxPriority:* Valor de la prioridad que tendrá la tarea.
- *pvCreateTask:* Puntero que recibirá la dirección de la tarea creada, con el fin de poder manipularla futuramente.

vTaskStartScheduler: Esta función es la encargada de iniciar el núcleo del sistema operativo en tiempo real. No posee ningún parámetro.

vTaskPrioritySet: Esta función se encarga de cambiar el nivel de prioridad en tiempo de ejecución.

```
void vTaskPrioritySet(
        xTaskHandle pxTask,
        unsigned portBASE_TYPE ,
       uxNewPriority
       );
```

Explicación de los parámetros

- *pxTask:* Puntero que contiene el manejador de la tarea.

- *uxNewPriority:* Nuevo valor del nivel de prioridad.

vTaskDelete: Usando este llamado se elimina una tarea en específico. A esta función se le pasa como parámetro el puntero del manejador de la tarea que se desea eliminar.

taskENTER_CRITICAL() y taskEXIT_CRITICAL(): Indica que el código que se implemente entre estos dos llamados se ejecutará de forma crítica e ininterrumpidamente, dando absoluto control de los recursos a este segmento del código. No se ejecutarán interrupciones ni se ejecutará el planificador mientras se ejecute un código contenido entre estos.

Para cargar el sistema operativo FreeRTOS, basta con implementar un código como el Código 2

```
/* Inclusiones Estándares */
#include <stdio.h> // Declaración de funciones prototipo E/S

/* Inclusiones del FreeRTOS */
#include "FreeRTOS.h"
#include "task.h"

void tarea1 (void* pvParameters){
        while(1){}
}
void tarea2 (void* pvParameters){
        while(1){}
}

int main (void){
 xTaskCreate( Tarea1, "T1",512, NULL,2, NULL );
 xTaskCreate( Tarea2, "T2",512, NULL,2, NULL );
 vTaskStartScheduler();
}
```

Código 2: Implementación mínima de un sistema con FreeRTOS.

5.2 Función principal

En cualquier programa basado en C/C++, existe una función principal que será la que se ejecute al iniciar el programa; en nuestro caso, esta función se ejecuta cada vez que se reinicia el sistema y es la encargada de configurar el sistema operativo del sistema embebido, así como de ciertas funciones y características que requieren de configuración una sola vez.

Entre las funciones más importantes que se realizan en este procedimiento, se encuentra la configuración del segundo temporizador de que dispone el microcontrolador. Este se configura para que genere una interrupción cada 1ms, tiempo mínimo que tendrá el sistema para el muestreo de señales. Para lograr esta temporización, se utilizan las instrucciones del Código 3.

```
OpenTimer2(T2_ON | T2_PS_1_16, 0x00001388); //Configura el temporizador
ConfigIntTimer2(T2_INT_ON | T2_INT_PRIOR_1); //Se enciende la interrupción con la máxima
```

Código 3: Configuración del temporizador 2.

Para calcular el valor de 0x00001388 presente en el código anterior, se despeja PR2 de la Ecuación 7:

$$T_{Deseado} = \frac{1}{PBCLK} * PREESCALADO * PR2$$

Ecuación 7: Ecuación empleada para el cálculo de PR2.

Otra función importante de este método es la configuración del sistema operativo y su puesta en marcha del mismo, esto se hace mediante el Código 4.

```
xTaskCreate( servidorRED_Configuracion, "SW_CONFIG",1024,
NULL,Prioridad_Configuracion,NULL);
vTaskStartScheduler(); // Inicia el listado ejecutor de tareas
```

Código 4: Programación de tareas en FreeRTOS

Como se observa, en este procedimiento se ejecuta el servidor en red y se le pone una prioridad “Prioridad _Configuración” previamente definida con un valor de 3. Esta tarea se ejecutará desde este momento hasta que el dispositivo se desconecte de la fuente de energía.

Es también en el “*main*” donde habilitamos las entradas analógicas; estas se configuran para entregar el resultado en un número entero entre 0 y 1.023, con referencias internas de VDD a VSS (0 a 3.3V). Esto se explica mejor en el Código 5.

```
CloseADC10();

// Definir Parametros para el ADC
// EncenderModulo| formato INT | trigger auto | Habilitar el AutoMuestreo
#define PARAM1 ADC_MODULE_ON | ADC_FORMAT_INTG | ADC_CLK_AUTO |
ADC_AUTO_SAMPLING_ON

// ADC ref externa | Sin corrector de OFFSET| Scaneado | Tomar 4 muestras | Usar doble BUF | Solo usar el
MUX A(SOLO EXISTEN 2 )
#define PARAM2 ADC_VREF_AVDD_AVSS | ADC_OFFSET_CAL_DISABLE | ADC_SCAN_ON |
ADC_SAMPLES_PER_INT_4 | ADC_ALT_BUF_ON | ADC_ALT_INPUT_OFF

// use el reloj interno | Tickets de Rastreo por vez
#define PARAM3 ADC_CONV_CLK_INTERNAL_RC | ADC_SAMPLE_TIME_15

// Comfigurar An0-An3 como entradas analogicas
#define PARAMe4 ENABLE_AN0_ANA | ENABLE_AN1_ANA| ENABLE_AN2_ANA| ENABLE_AN3_ANA

// Muestrear de An0 a An3
#define PARAM5 SKIP_SCAN_AN4 |SKIP_SCAN_AN5 |SKIP_SCAN_AN6 |SKIP_SCAN_AN7 | \
SKIP_SCAN_AN8 |SKIP_SCAN_AN9 |SKIP_SCAN_AN10 |SKIP_SCAN_AN11 | \
SKIP_SCAN_AN12 |SKIP_SCAN_AN13 |SKIP_SCAN_AN14 |SKIP_SCAN_AN15

// Usae el mux 1 con referencia negativa

SetChanADC10( ADC_CH0_NEG_SAMPLEA_NVREF );

// Abrir el ADX the ADC
OpenADC10( PARAM1, PARAM2, PARAM3, PARAMe4, PARAM5 );
```

Código 5: Fragmento de código para la configuración de los módulos ADC en el PIC32.

5.3 Tarea de comunicación

Esta es la tarea encargada de toda la comunicación Ethernet del dispositivo; tanto la conexión TCP/IP para recibir instrucciones de configuración y programación, como el protocoló RTP utilizado para el envío de los valores de señales del sistema modelado por el estudiante, para ser presentada en el computador.

La conexión TCP/IP del sistema recibe tres posibles instrucciones, y dependiendo de estas, el sistema entra en una u otra máquina de estados de Moore. Estas palabras son: CALIBRACION, PROGRAMACION, MONITOR. En cambio, la conexión RCP se realiza solo si está activada la bandera de monitoreo. Sobre estas dos conexiones, se detallará más en los siguientes subapartados.

El Código 6 contiene las funciones necesarias para inicializar el "socket" TCP/IP.

```
TCPIPSetDefaultAddr (DEFAULT_IP_ADDR, DEFAULT_IP_MASK,
DEFAULT_IP_GATEWAY, DEFAULT_MAC_ADDR);
```

Código 6: Instrucción para la configuración del Ethernet.

5.3.1 Programación del flujo de datos

El sistema entra en esta etapa cuando recibe la instrucción "PROGRAMA". En este momento, el sistema limpia cualquier configuración de flujo anterior y prosigue a enviarle al ordenador la comprobación "Bonvenon" y se da un avance al siguiente estado de la máquina de Moore.

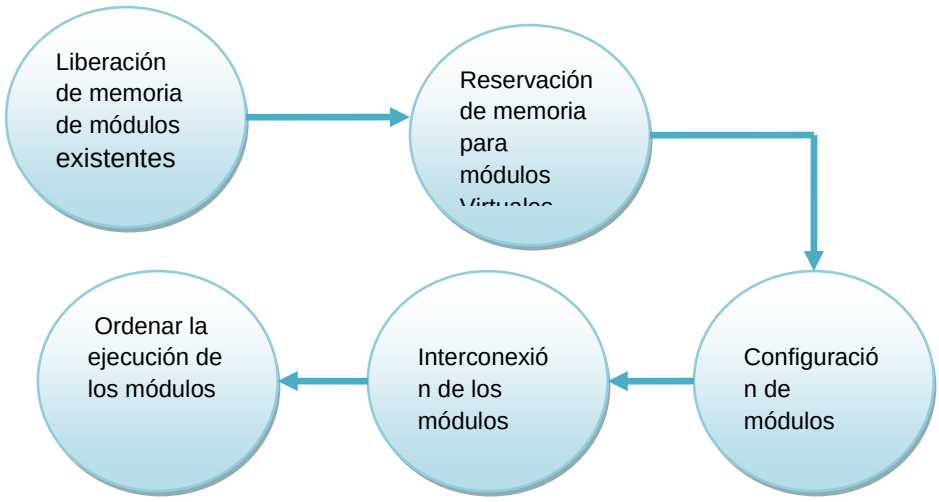


Figura 22: Diagrama de la programación del flujo de datos.

5.3.1.1 *Reservación de espacio de módulos virtuales*

La reservación del espacio en memoria de los módulos virtuales se hace en la segunda etapa del flujo de datos, tal y como se muestra en la Figura 22, mediante un código que sigue la estructura del Código 7.

```
if(!strcmp("tipo," ,comando)){
        tipo = (P_TIPO*) malloc(entero_1*sizeof(P_TIPO));
        eTIPO = TRUE; //Bandera que indica que se hay memoria reservada con
este tipo
`
```

Código 7: Estructura para la reservación de memoria de módulos virtuales.

5.3.1.2 *Configuración de módulos*

En esta etapa se asignan las constantes, los tiempos y otros valores que podrían ser necesarios para el correcto funcionamiento del módulo. Esto se hace siguiendo una estructura similar a la del siguiente código; sin embargo, dependiendo de las necesidades del tipo de módulo, puede variar en el número y tipo de valores con respecto a los mostrados.

```
if(!strcmp("TIPO" ,comando)){
 sscanf(linea,"%s %u, %u, %f;",comando,&entero_1,&entero_2,&flotante_1);
 tipo[entero_1].valor1 = entero_2;
 tipo[entero_1].amplitud = flotante_1;
}
```

Código 8. Estructura para la configuración de los módulos.

5.3.1.3 *Conexión de módulos*

Esta es la última etapa de la máquina de Moore para la programación del flujo de datos. Aquí los módulos se conectan unos con otros, formando un modelo de sistema.

Para lograr este sistema, se le asigna a los punteros a memorias de las entradas, la dirección de memoria de la salida, utilizando un puntero de intercambio, como se muestra en el Código 9.

```
//La Salida de un modulo tipo1
if(!strcmp("tipo1," ,receptor[q].Salida.tipo)){
        puntero = &tipo1[receptor[q].Salida.instancia].Salida;
}// Se conecta a la entrada1 de un modulo tipo2
if(!strcmp("tipo2," ,receptor[q].Entrada.tipo)){
        tipo2[receptor[q].Entrada.instancia].Entrada1 = puntero;
}
```

Código 9: Estructura para la conexión de módulos.

5.3.2 Calibración del sistema

Esta etapa se inicia cuando se envía la palabra CALIBRACION a la unidad de control. Esta última envía al ordenador instrucciones que el usuario deberá realizar mecánicamente, como lo es conectar, en las entradas indicadas, señales con tensiones constantes de alta fidelidad en las terminales indicadas en la instrucción.

La Figura 23 muestra un fragmento de lo que sería el sistema de calibración para una de las terminales de entrada y para una de las terminales de salida, con el fin de poder explicarle al lector el algoritmo realizado por la unidad controladora para manejar las etapas de entrada y salida con la mayor exactitud posible, y así obtener datos confiables. Las demás etapas del sistema de calibración siguen una estructura similar, variando únicamente los valores que aparecen en el diagrama, por los valores que correspondan a la Tabla 6.

Tabla 6: Valores iniciales usados para la etapa de calibración

Calibración (V)	TensiónCD (V)	TensiónCD $_b$	Rg (kΩ)	Rg$_b$	G (ul)	Rdiv (kΩ)	Rdiv$_b$
Cero +2	0	0	21.88	144	3.29	200	0
Cero +1	0	0	8.98	210	6.57	200	0
Cero +500m	0	0	4.10	235	13.20	200	0
Cero +200m	0	0	1.56	248	33	200	0
Cero +100m	0	0	0.78	252	65	200	0
Cero +50m	0	0	0.39	254	129	200	0
Cero +5	0	0	21.88	144	3.29	50	192
Cero +10	0	0	21.88	144	3.29	25	224
Techo +10	0	0	21.88	144	3.29	25	224
Techo ±10	1	2048	21.88	144	3.29	10.16	243
Piso ±10	1	2048	21.88	144	3.29	10.16	243
Techo +5	0	0	21.88	144	3.29	50	192
Techo ±5	1	2048	21.88	144	3.29	25	224
Piso ±5	1	2048	21.88	144	3.29	50	224
Techo +2	0	0	21.88	144	3.29	200	0
Techo ±2	1	2048	21.88	144	3.29	50	192
Piso ±2	1	2048	21.88	144	3.29	50	192
Techo +1	0	0	8.98	210	6.57	200	0
Techo ±1	1	2048	21.88	144	3.29	200	0
Piso ±1	1	2048	21.88	144	3.29	200	0
Techo +500m	0	0	4.10	235	13.20	200	0
Techo ±500m	1	2048	8.98	210	6.57	200	0
Piso ±500m	1	2048	8.98	210	6.57	200	0
Techo +200m	0	0	1.56	248	33	200	0
Techo ±200m	1	2048	3.24	172	16.4	200	0
Piso ±200m	1	2048	3.24	172	16.4	200	0
Techo +100m	0	0	0.78	252	65	200	0
Techo ±100m	1	2048	1.56	248	33	200	0
Piso ±100m	1	2048	1.56	248	33	200	0
Techo +50m	0	0	0.39	254	129	200	0
Techo ±50m	1	2048	0.78	252	65	200	0
Piso ±50m	1	2048	0.78	252	65	200	0

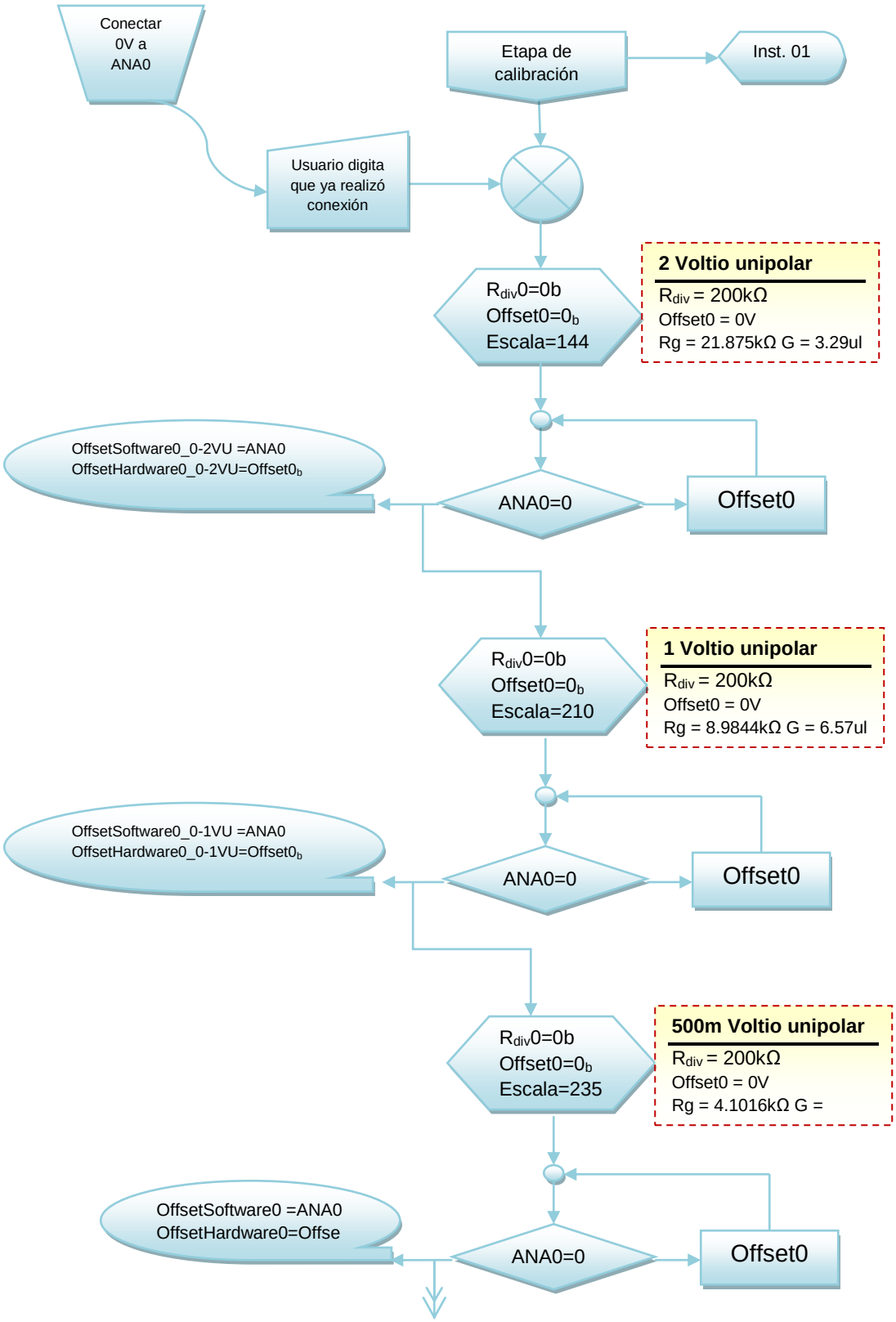


Figura 23: Extracto del diagrama de flujos del sistema de calibración para un único canal.

5.3.3 Osciloscopio y monitorización de señales

La monitorización de señal se hace mediante una comunicación RTP, lo cual permite que los datos lleguen en orden a pesar de que se permite la pérdida de datagramas. Estos datagramas son datos seleccionados mediante una estructura como el Código 10.

```
MONITOR
begin
config 2, 1;
tipo1, 0;
tipo2,0;
end
```

Código 10: Fragmento para monitorizar señales (Implementado hasta UCP).

Como se ve en el Código 10, el fragmento de código que puede ser agregado en el archivo de programación o como escrito aparte, sigue la misma sintaxis que los demás fragmentos del archivo de programación BPC; estando definido entre un *begin* y un *end*. Además, también se observa la instrucción "*config*" seguido de dos parámetros, el primero indica el número de elementos que se observaran y el segundo el tiempo en enteros de milisegundos que pasará entre una y otra transmisión.

Este sistema solo puede observar las salidas internas de los módulos y no sus entradas, por lo que no puede monitorear las salidas de los periféricos pero sí sus entradas monitoreando la salida del módulo al que está conectado el periférico.

La Figura 24 muestra un ejemplo de la señal observada mediante el código de este apartado; sin embargo, cabe destacar que el programa que intercepta y grafica esta señal es un programa de prueba no vinculante para el desarrollo del sistema.

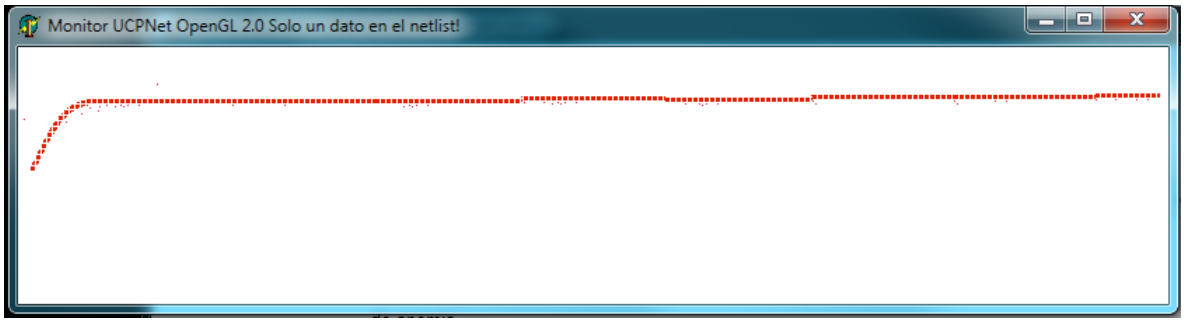


Figura 24: Señal de salida de un sistema programado en el UCPnet, graficada en el computador.

5.4 Tarea del flujo de control

Una vez activada la bandera de flujo, el sistema ejecuta esta tarea con prioridad máxima y de forma crítica, por lo que el flujo de datos se logra resolver en menos de 100uS. Terminado de realizar un ciclo del flujo de control, la tarea devuelve el control al sistema operativo, el cual se queda ejecutando la tarea de comunicación. En la Figura 25, se muestra esta división de tiempo.

Figura 25: División de tiempo cuando existe un flujo de control que ejecutar.

5.5 Módulos de operación matemática

Estos son módulos virtuales que no tienen ningún valor de configuración ni de temporización, y se ejecutan en todos los ciclos del flujo de control, dependiendo única y exclusivamente de sus dos entradas, con las cuales realizará la operación aritmética indicada. Para el caso de la multiplicación, hay que pasar el contenido de la entrada a una memoria local de punto flotante, puesto que no se puede multiplicar el contenido de forma directa.

5.6 Módulos de generación

El sistema es capaz de generar diferentes tipos de señal, tanto señales de ruido y aleatorias como señales sinusoidales, rectangulares y triangulares. En este apartado, se mostrarán los parámetros de configuración que permiten configurar los diferentes generadores.

5.6.1 Señales sinusoidales

Para la generación de las señales sinusoidales, se solicitan dos tipos de parámetros, el primero es la frecuencia de la señal, y el segundo la amplitud. Si se necesitara realizar una señal de este tipo con nivel de CD, se deberá de agregar dicho nivel mediante programación BPC. En la Figura 26, se muestra una señal sinusoidal generada con el Código 11.

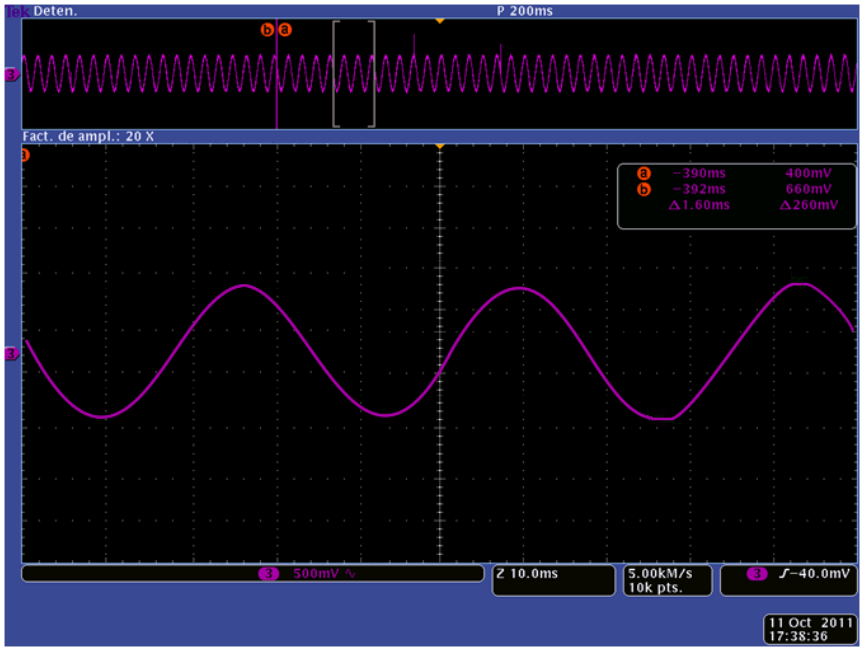


Figura 26: Imagen sinusoidal generada en el UCPnet.

```
MOD //sección de crear módulos
begin
sosc, 1;
end
CONFIG //sección de configurar módulos
begin
RANGEDAC 0, 2;
SOSC 0, 30, 4;
end
NETLIST //sección donde se especifica las conexiones entre los módulos
begin
sosc, 0, 1, dac, 0, 1;
end
```

Código 11: Señal sinusoidal 30hz 4Vpp en BCP.

5.6.2 Señales rectangulares

Las señales rectangulares que se pueden generar en la unidad controladora son tanto simétricas como asimétricas. El usuario puede determinar la duración de todos los lapsos de la señal, así como la tensión positiva y negativa del sistema. Para ilustrar esto, se muestra la Figura 27, donde se podrá observar los parámetros de configuración que corresponderán a cada parte de la señal.

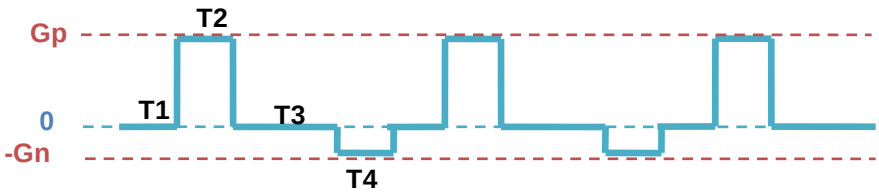


Figura 27: Señal rectangular asimétrica/simétrica.

La Figura 28 muestra una señal asimétrica creada mediante la utilización del sistema al ser programado con el Código 12.

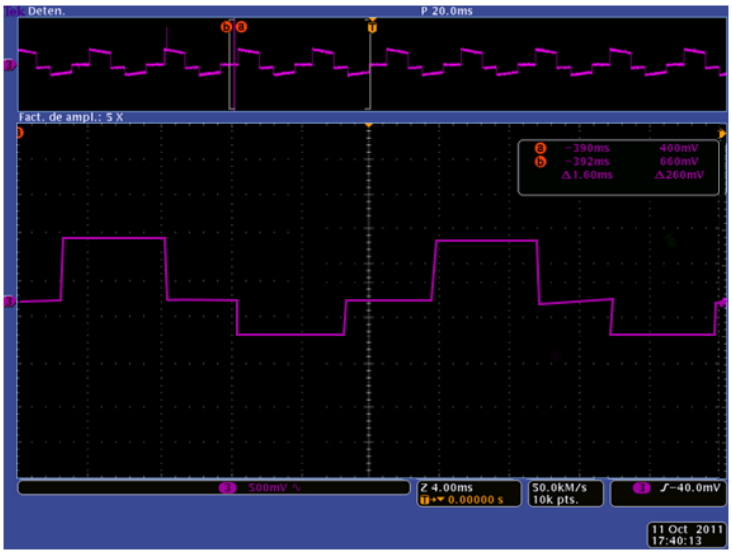


Figura 28: Señal cuadrada asimétrica generada por el UCPnet.

```
MOD //sección de crear módulos
begin
rosc, 1;
end
CONFIG //sección de configurar módulos
begin
RANGEDAC 0, 2;
ROSC 0, 5, 6, 4, 4, 2;
end
NETLIST //sección donde se especifica las conexiones entre los módulos
begin
rosc, 0, 1, dac, 0, 1;
end
```

Código 12: Señal cuadrada asimétrica en BPC.

5.6.3 Señales triangulares

La unidad desarrollada posee la capacidad de generar señales triangulares, estableciendo previamente diferentes parámetros que permitan al usuario crear una amplia gama de señales de este tipo. Para entender estos parámetros y cómo modifican la señal triangular, se utiliza el esquema de la Figura 29.

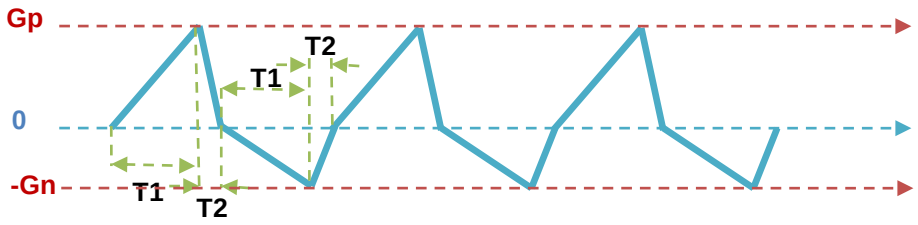


Figura 29: Parámetros de la señal triangular.

Como ejemplo de esta señal, tenemos la generada en la Figura 31 mediante el código BPC que también se muestra a continuación.

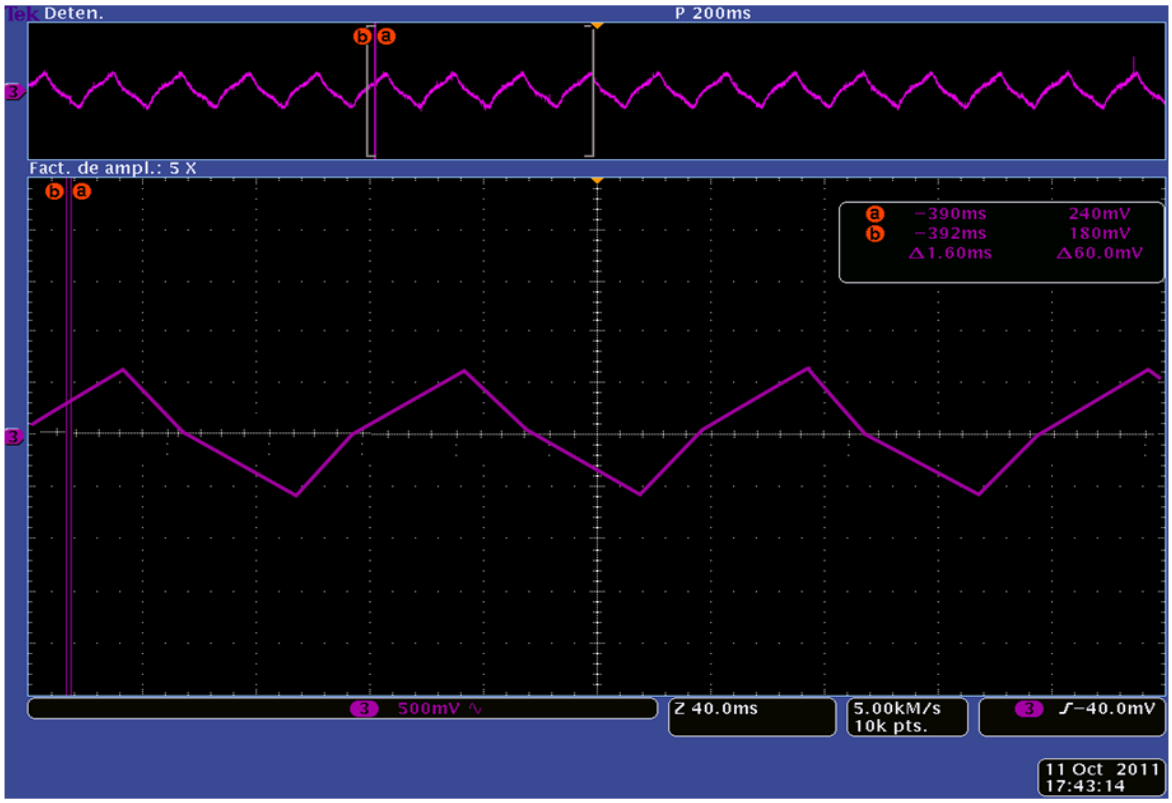


Figura 31: Señal triangular asimétrica generada por el UCPnet.

```
MOD //sección de crear módulos
begin
rosc, 1;
end
CONFIG //sección de configurar módulos
begin
RANGEDAC 0, 2;
ROSC 0, 5, 6, 4, 4, 2;
end
NETLIST //sección donde se especifica las conexiones entre los módulos
begin
rosc, 0, 1, dac, 0, 1;
end
```

Código 13 BCP para generar una señal triangular asimétrica.

5.6.4 Señales de ruido

En todo sistema de control existe presencia de ruido, por lo que esta unidad incluye un generador de ruido blanco, con el fin de poder modelar los sistemas de manera más precisa.

La Figura 32 se generó mediante el código que se muestra debajo de esta.

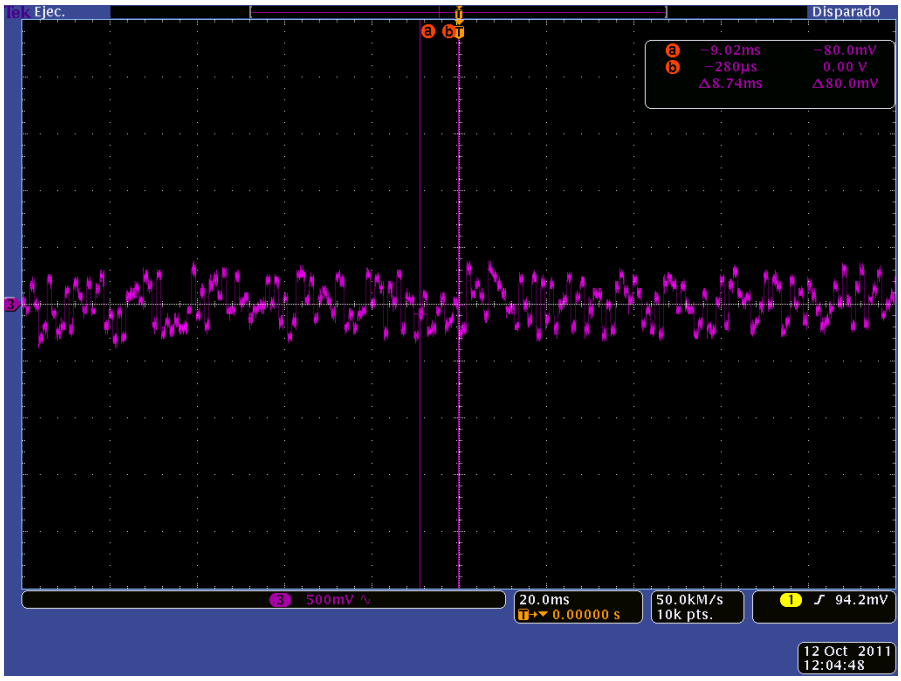


Figura 32: Señal de ruido generada por la unidad de control.

```
MOD //sección de crear módulos
begin
rosc, 1;
end
CONFIG //sección de configurar módulos
begin
RANGEDAC 0, 2;
ROSC 0, 5, 6, 4, 4, 2;
end
NETLIST //sección donde se especifica las conexiones entre los módulos
begin
rosc, 0, 1, dac, 0, 1;
end
```

Código 14: Programa para la generación de una señal de ruido.

5.6.5 Señales de ancho aleatorio

Estos módulos virtuales generan señales con diferentes anchos de manera aleatoria, como se muestran en la Figura 33.

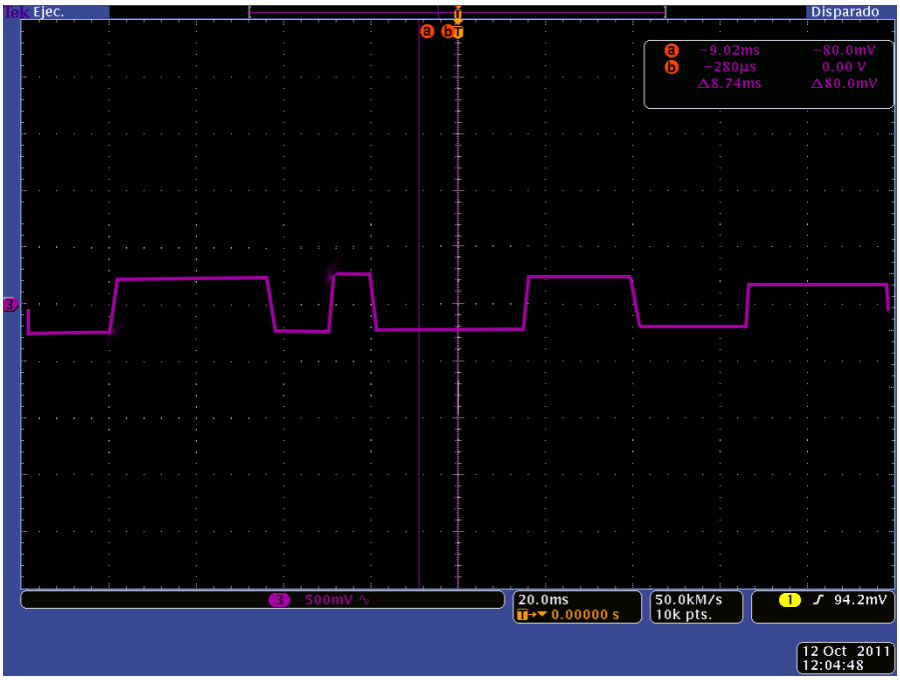


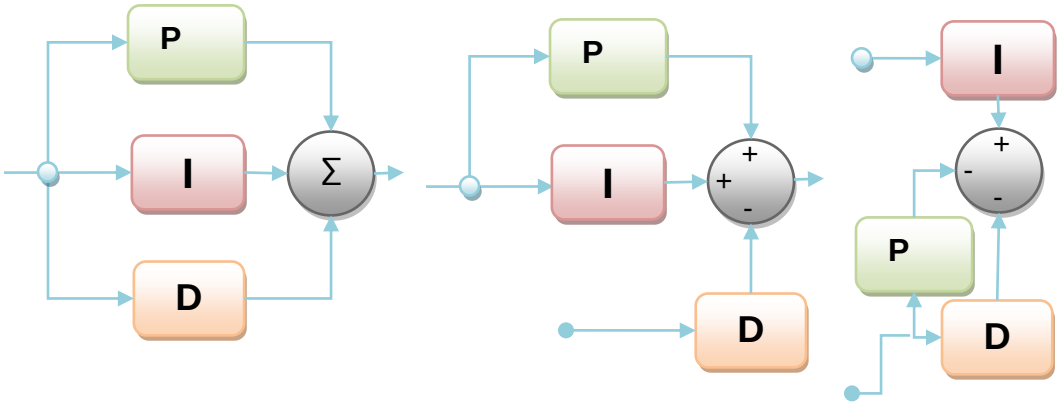
Figura 33: Señal con ancho temporal aleatorio generada en el BPC

5.7 Módulo de controles lineales/no lineales

5.7.1 Controladores PID

La unidad cuenta con tres tipos de controladores PID previamente programados. Uno de estos, el PID convencional, solamente posee una entrada; mientras, los otros dos tipos de controladores poseen dos punteros de entradas, con los cuales realiza la función del control.

En la Figura 14, se muestran las estructuras de estos controladores. Seguidamente, se brinda una breve explicación del código que ejecuta estos módulos, teniendo en cuenta que el cálculo de las constates discretas a partir de las constantes continuas se realizan en la etapa de configuración del módulo, con el fin de reducir el número de operaciones que el sistema debe realizar en una función crítica, como lo es el flujo de control.[16]



PID Convencional PI_D I_PD

Figura 34: Estructuras de PID predefinidas en la unidad controladora.

```
if(pid[Netlist[qq].Salida.instancia].tipo==0){ //Tipo PID
// Cp*X[n]
pid[Netlist[qq].Salida.instancia].Salida = pid[Netlist[qq].Salida.instancia].Cp*usell;
//+ Ci*X[n-1] + Y[n-1]
pid[Netlist[qq].Salida.instancia].Salida =
pid[Netlist[qq].Salida.instancia].Salida+pid[Netlist[qq].Salida.instancia].Ci*pid[Netlist[qq].Sal
ida.instancia].X_n1+pid[Netlist[qq].Salida.instancia].Y_n1;
// + Cd(X[n]-X[n-1]) - FacD*Y[n-1]
pid[Netlist[qq].Salida.instancia].Salida =
pid[Netlist[qq].Salida.instancia].Salida+pid[Netlist[qq].Salida.instancia].Cd*(usell-
pid[Netlist[qq].Salida.instancia].X_n1)-
pid[Netlist[qq].Salida.instancia].FacD*pid[Netlist[qq].Salida.instancia].Y_n1;

pid[Netlist[qq].Salida.instancia].Y_n1 = pid[Netlist[qq].Salida.instancia].Salida;

        pid[Netlist[qq].Salida.instancia].X_n1 = usell;
```

Código 15: Fragmento de código del PID convencional.

La Figura 24 muestra la señal de salida de un bloque de PID convencional generado en la unidad y mostrada vía Ethernet.

5.7.2 Función de transferencia

El usuario puede programar funciones de transferencia que contengan dos polos y dos ceros o menos, así como colocar estos módulos virtuales en cascada para obtener mayores órdenes.

Existen dos formas en que el usuario puede configurar el módulo, ya sea escogiendo el valor de los polos y ceros, o estableciendo el valor de las constantes del polinomio factorizado. La Ecuación 8 y la Ecuación 9 muestran estos dos tipos.

$$h(s) = \frac{(s + z_1)(s + z_2)}{(s + p_1)(s + p_2)}$$

Ecuación 8: Función de transferencia de segundo orden descrita en forma de polos y ceros.

$$h(s) = \frac{b_0 s^2 + b_1 s + b_2}{s^2 + a_1 s + a_0}$$

Ecuación 9: Función de transferencia de segundo orden descrita en forma factorizada.

Como se observa en las Ecuación 8 y Ecuación 9, el usuario debe ingresar los parámetros en el dominio del tiempo. Sin embargo, debido a que el sistema trabaja digitalmente, estas ecuaciones son trasladadas al dominio discreto en la etapa de configuración de los módulos. Por esta razón, los parámetros que son ingresados en configuración de polos y ceros son llevados a su equivalente factorizado, para así aplicar la Ecuación 10 en la etapa del flujo de control.

$$y[n] = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] - a_1 y[n-1] - a_2 y[n-2]$$

Ecuación 10 Ecuación de diferencias para un polinomio de orden 2.

El diagrama de la Figura 35 muestra el sistema que se programó en la unidad de control para lograr la gráfica de la Figura 24, seguido del código que lo representa.

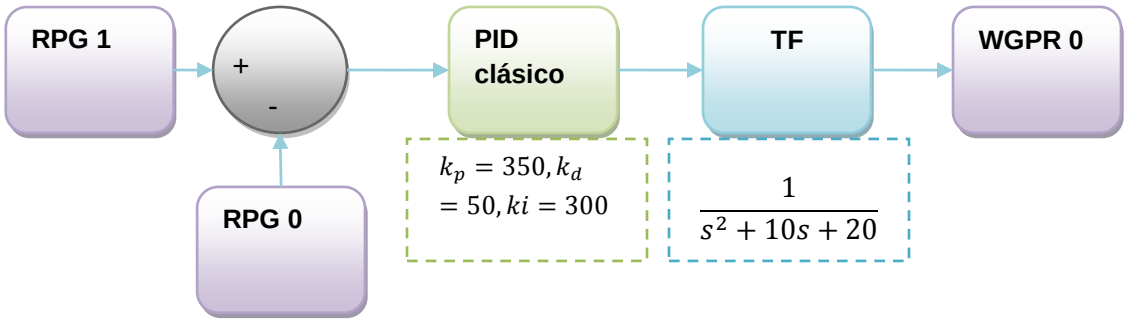


Figura 35: Diagrama de un sistema con PID y una función de transferencia.

```
MOD //Suma de señal seno y cuadrada
begin
tf, 1;
pid, 1;
sum, 1;
end
CONFIG //sección de configurar módulos
begin
TF 0, 1, 1, k_f, 10, 20, 0, 0, 1; //TF Num_Bloque, T, Tipo, a1, a2, b0, b1, b2r;
PID 0, 1, 1, 350, 300, 50, 1000, 0; //PID Num_Bloque, Numero_Tipo,T, KP, KI, KD, FacT, SAT_Integrador;
RGP 1, 1;
end
NETLIST //sección donde se especifica las conexiones entre los módulos
begin
rgp, 1, 1, sum, 0, 1;
rgp, 0, 1, sum, 0, 2;
sum, 0, 1, pid, 0, 1;
pid, 0, 1, tf, 0, 1;
tf, 0, 1, wgpr, 0, 1;
end
MONITOR //sección donde se especifica las conexiones entre los módulos
begin
tf, 1;
end
```

Código 16: Programa para simular una planta junto a su controlador y enviar los datos vía Ethernet

5.7.3 Módulo limitador

Este módulo limita la señal a dos valores establecidos en el programa BPC, creando una saturación en el valor de dichos límites. Es ideal para proteger las salidas y evitar acumulación de errores. La Figura 36 fue creada mediante el código que aparece en este mismo apartado y muestra cómo se vería una señal sinusoidal limitada.

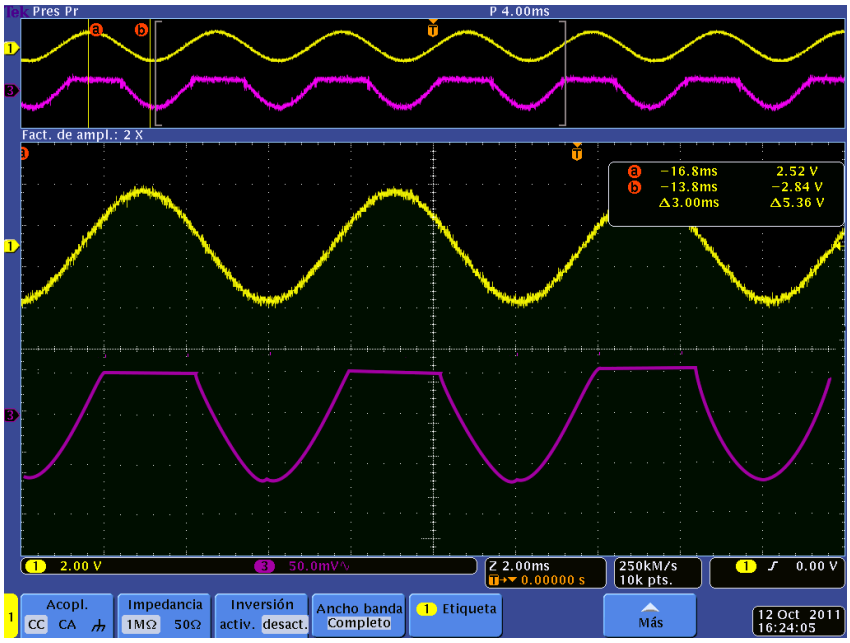


Figura 36: Señal sinusoidal limitada (saturada).

En la Figura 36, como en todas las imágenes de pantalla del osciloscopio donde no se indique lo contrario, la señal identificada con el color amarillo representa la señal capturada en un canal analógico mientras que la señal fucsia representa la señal de salida del sistema.

5.7.4 Función de retraso

Esta es una función que se ha incorporado hasta esta unidad (no venía en el BPC ni en el lenguaje del intermedio), con la cual es posible retrasar el valor de una señal hasta un tiempo máximo de 256 periodos. Este retraso obedece a la Ecuación 11.

$$y[n] = z^{-n}$$

Ecuación 11: Retraso en tiempo discreto.

En la Figura 37 se muestra una señal tomada por medio de la primera entrada analógica, la cual es desplazada 50 ms, como se describe el Código 17.

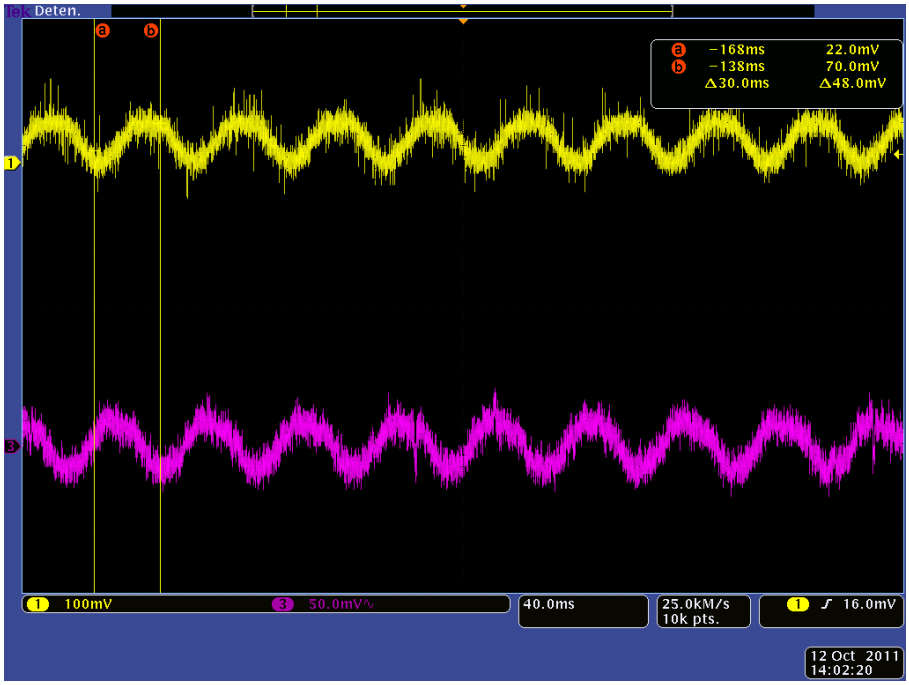


Figura 37: Desplazamiento temporal de una señal sinusoidal.

```
MOD //sección de crear módulos
begin
delay, 1;
end
CONFIG //sección de configurar módulos
begin
RANGEADC 0, 2;
RANGEDAC 0, 2;
DELAY 0, 30, 1; //Retraso de 30 unidades de 1 ms
end
NETLIST //sección donde se especifica las conexiones entre los módulos
Begin
adc, 0, 1, delay, 0, 1;
delay, 0, 1, dac, 0, 1;
end
```

Código 17: Programa BCP que genera un desplazamiento temporal de 50ms de la señal de entrada

5.8 Filtros digitales

5.8.1 Filtro FIR

Este filtro está implementado para recibir 4 coeficientes, los cuales se indican en la etapa "*config*" mediante la instrucción "*FIR nuemeroInstancia, b0, b1, b2, b3;".* Los coeficientes son procesados siguiendo la estructura de la Figura 38.

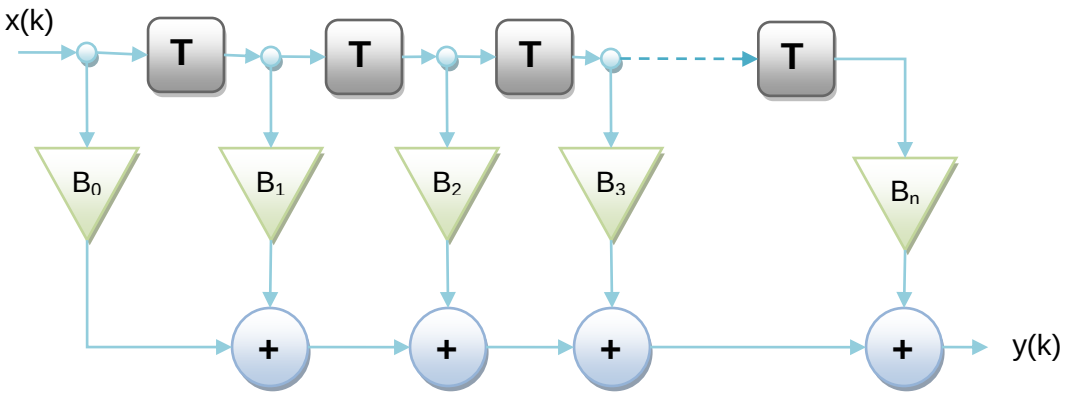


Figura 38: Estructura de un filtro FIR.

La estructura presentada en la Figura 38 corresponde al modelo matemático mostrado en la Ecuación 12 de orden N. [2][15]

$$y_n = \sum_{k=0}^{N-1} B_k x(n-k)$$

Ecuación 12: Ecuación de diferencias de un filtro de coeficientes finitos.

Y esta se ha implementado mediante el Código 18.

```
if(!strcmp("fir," ,Netlist[qq].Salida.tipo)){
usell = 0;
fir[Netlist[qq].Salida.instancia].Xn[0] = *fir[Netlist[qq].Salida.instancia].Entrada1;
for(contador=0;contador<sizeof(FIR_Bk);contador++){
          usell = usell + FIR_Bk[contador]*fir[Netlist[qq].Salida.instancia].Xn[contador];
          if(Contador<sizeof(FIR_Bk)-1)
          fir[Netlist[qq].Salida.instancia].Xn[contador+1]=fir[Netlist[qq].Salida.instancia].Xn[conta
dor];
          }
fir[Netlist[qq].Salida.instancia].Salida = usell;
}
```

Código 18: Implementación en el firmware de un filtro FIR

5.8.2 Filtro IIR

Los filtros IIR, al ser excitados por una señal de impulso, producirán una salida infinita de términos diferentes de cero, por lo cual el sistema no volverá jamás al estado de reposo. Este tipo de filtro depende tanto de la entrada actual como de las entradas anteriores, así como de las salidas pasadas. El filtro se puede representar mediante la Ecuación 13, función de transferencia, y representada digitalmente mediante la ecuación de diferencias que se muestra a continuación. [2]

$$H(z) = \frac{\sum_{k=0}^{M} B_k z^{-k}}{1 + \sum_{k-1}^{N} A_{kz^{-k}}}$$

$$y[n] = \sum_{k=0}^{M} B_k x[n-k] - \sum_{k=1}^{N} A_k\, y[n-k]$$

Ecuación 13: Función de transferencia y ecuación de diferencias del filtro IIR.

Se implementó un algoritmo selector para reducir el número de operaciones recursivas a una sola, teniendo como número de recursiones el orden del filtro que es dado por el valor mayor entre *M Y N* de la ecuación anterior. Esta implementación se muestra en el Código 19.

```
if(!strcmp("iir," ,Netlist[qq].Salida.tipo)){
        usell = 0;
        iir[Netlist[qq].Salida.instancia].Xn[0] = *iir[Netlist[qq].Salida.instancia].Entrada1;
for(Contador=0;Contador<max(sizeof(IIR_Bk),sizeof(IIR_Ak));Contador++)
        {
                if(Contador<sizeof(IIR_Bk)){
                 usell = usell + IIR_Bk[Contador]*iir[Netlist[qq].Salida.instancia].Xn[Contador];
                 if(Contador<sizeof(IIR_Bk)-1)
        iir[Netlist[qq].Salida.instancia].Xn[Contador+1]=iir[Netlist[qq].Salida.instancia].Xn[Cont
ador];   }if(Contador<sizeof(IIR_Ak)){
 usell = usell- IIR_Ak[Contador]*iir[Netlist[qq].Salida.instancia].Yn[Contador];
        if(Contador<sizeof(IIR_Ak)-1)
        iir[Netlist[qq].Salida.instancia].Xn[Contador+1]=iir[Netlist[qq].Salida.instancia].Xn[Cont
ador];
        }
         iir[Netlist[qq].Salida.instancia].Salida = usell;
iir[Netlist[qq].Salida.instancia].Yn[0] = usell;
        }
}
```

Código 19: Implementación del filtro IIR

5.9 Módulos de entradas

5.9.1 Convertidor análogo/digital

Estas tareas se realizan de primero, sin importar en qué posición se encuentren los módulos en el *netlist*, ya que el tiempo de dilatación entre una lectura del sistema y su salida debe de ser el mínimo posible.

Los microcontroladores de la familia PIC32MX poseen 16 canales que pueden ser muestreados mediante la multipexión de dos registros temporales; sin embargo, para lograr leer más de un canal, solo se debe realizar las mediciones a través de un registro temporal, esto se establece en el Código 5 del apartado “Entradas analógicas”. Una vez configuradas correctamente las entradas analógicas, se prosigue a hacer las lecturas mediante desplazamientos en el registro temporal.

Los datos almacenados en la memoria del sistema se encuentran en el rango de 0 a 1.023, por lo que se hace necesario transformar estas lecturas a sus respectivos valores de tensión y así posibilitar que sean utilizados en el flujo normal de la ejecución del control programado. Esto se muestra en el Código 20, perteneciente a la función ObtenerADC.

```
//operacion de obtencion de los adc
void ObtenerADC(){
unsigned int Canal0,Canal1,Canal2,Canal3,OffSet;
 OffSet = 8 * ((~ReadActiveBufferADC10() &0x01));
 Canal0 = ReadADC10(OffSet);
 Canal1 = ReadADC10(OffSet + 1);
 Canal2 = ReadADC10(OffSet + 2);
 Canal3 = ReadADC10(OffSet + 3);
        //Realiza la conversion del canal a tensión solo si este esta
activo(mx+b)
        if (eADC[0]==TRUE)adc[0].Salida = Canal0*adc[0].m+adc[0].b;
        if (eADC[1]==TRUE)adc[1].Salida = Canal1*adc[1].m+adc[1].b;
        if (eADC[2]==TRUE)adc[2].Salida = Canal2*adc[2].m+adc[2].b;
        if (eADC[3]==TRUE)adc[3].Salida = Canal3*adc[3].m+adc[3].b;
}
```

Código 20: Procedimiento para obtener los datos del ADC.

En la Figura 28 y en el Código 20 se muestra la utilización de este módulo.

5.10 Módulos de salidas

5.10.1 Convertidor digital/análogo

Las salidas analógicas se procesan mediante un componente externo, como se ha indicado en el apartado de “

Salidas analógicas”. Este componente consiste en un DAC de 12 bits. Los bits se transfieren a este mediante comunicación I^2C en bloques de 8 bits (1 byte), por lo que es necesario separar los 4 bits más significativos en un bloque y 8 bits menos significativos en otro. Además, en el primer bloque, los 4 bits más significativos del dato que quedaban libres se ponen a un valor de 0 lógico, que indica que se estará transfiriendo datos en el modo rápido.

En la Figura 39, se muestra la distribución de los datos utilizada para las salidas analógicas.

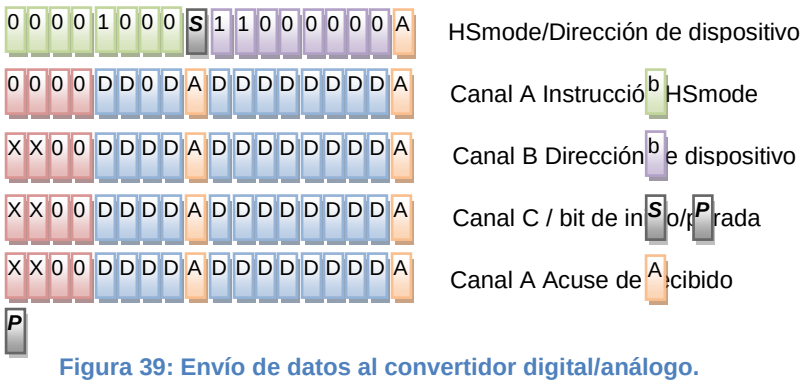


Figura 39: Envío de datos al convertidor digital/análogo.

Esta distribución de bits para obtener las salidas analógicas se hace mediante el fragmento que se ve en el Código 21.

```
void PonDac(){
        float dacA= *dac[0].Entrada1; // DacA, toma el valor del dato a enviar al convertidor
 int NormalizdoCHA = (int) rint( dacA*dac[0].m); //se redondea el dato como INT y se escala
        NormalizdoCHA=NormalizdoCHA+ dac[0].Doffset; //en caso de que sea existan NEG
se eleva
        IC2Dato[2] = (short) (NormalizdoCHA>>8); //Los 4 MSB de Normalizado se insertan al
I2CDato[2]
        IC2Dato[3] = (short) (NormalizdoCHA); //los 8Bits LSB de Normalizado se insertan al
I2CDato[3]
        IC2Dato[4] = (short) dac[0].offsetHi; //La parte alta de Offset se inserta a I2CDato[4]
        IC2Dato[5] = (short) dac[0].offsetLow; // La parte baja de Offset se inserta a I2CDato[4]
        StartI2C2();
        IdleI2C2();
int indice;
for (indice=1;indice<4;indice++){
         MasterWriteI2C2(IC2Dato[indice]); //Se envían los I2CDatos al MCP
        if( I2C2STATbits.ACKSTAT )IdleI2C2();
}
```

Código 21: Ejemplo de la función PonDac (Solo canal A).

5.10.2 Modulador PWM

Otro tipo de señal de salida que posee la unidad controladora es la modulación en ancho de pulso, llamado PWM, por sus siglas en inglés. Este tipo de modulación representa la magnitud de la tensión mediante el porcentaje del ciclo de trabajo de una señal cuadrada, y el signo de dicha tensión mediante una señal adicional que indica con un 1 lógico que la polaridad de dicha tensión es negativa.

A diferencia del PWM del BPC, donde el sistema estaba normalizado, la unidad controladora diseñada requiere que el usuario establezca la máxima amplitud de tensión que será equivalente al 100% del ancho de pulso. Este valor límite será el divisor del cociente del ciclo de trabajo, y la señal será el dividendo. Esto se ve en el fragmento que aparece en el Código 22.

```
if(ePWM[0]){
 usell = *pwm[0].Entrada1;
        SetDCOC1PWM(abs(PR3*usell/pwm[0].rango));
                if(usell< 0 )mPORTESetBits(BIT_1);
                else    mPORTEClearBits(BIT_1);
}
```

Código 22: Ejemplo de modulación por ancho de pulso.

El resultado de esta modulación se observa en la Figura 40, que se consigue cuando el sistema se programa con un código BPC similar al fragmento de Código 23.

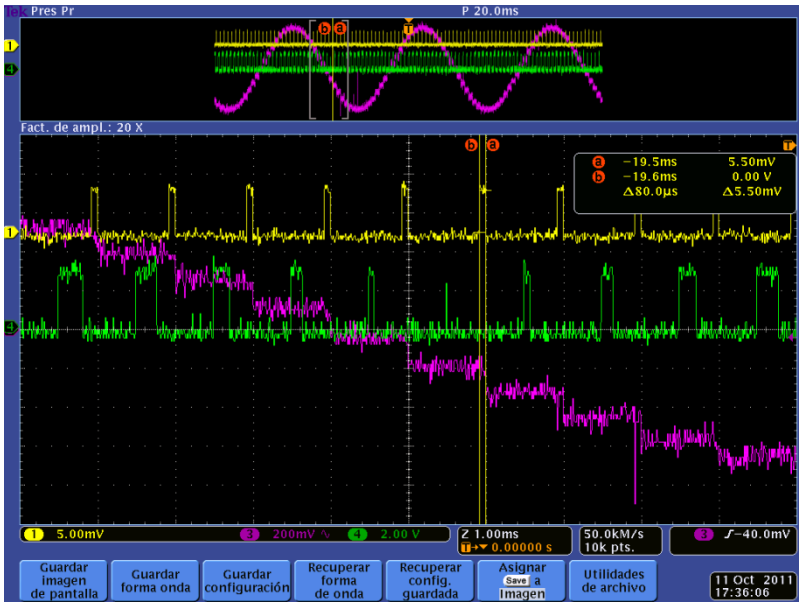


Figura 40: Ejemplo de modulación PWM de una señal sinusoidal.

Donde la señal fucsia representa la señal portadora; la verde, la señal modulada, y la amarilla, el tiempo que duró el sistema en generar la señal, para este caso 80µs.

EL código BPC que generó la señal de la Figura 40 es el Código 23.

```
MOD //sección de crear módulos
begin
sosc, 1;
end
CONFIG //sección de configurar módulos
begin
PWM 0, 1, 10;
SOSC 0, 30, 4;
end
NETLIST //sección donde se especifica las conexiones entre los módulos
begin
sosc, 0, 1, pwm, 0, 1;
sosc, 0, 1, dac, 0, 1;
end
```

Código 23: Programa BCP que modula una señal sinusoidal.

5.10.3 Escritura/lectura a un registro dedicado

El sistema UCPnet Kontrolo es capaz de guardar un valor en un registro temporal para que pueda ser leído en momento futuro. Para esto, se implementaron dos instrucciones ya existentes en el código del BPC: "*wrpg*" –que escribe un valor en el registro– y *"rgp"* –que lee el contenido del registro–. Siguiendo la convención

establecida, según la cual las entradas de los módulos son punteros que apuntan a un valor localizado en la memoria de una salida, se desarrollaron dos módulos: el primero simplemente es un módulo de lectura; por consiguiente, se implementó mediante un arreglo de flotantes; y el segundo es el módulo de escritura, el cual es un arreglo de una estructura que posee el puntero a la entrada. De esta forma, solamente se procesará el módulo de escritura, y no el de lectura.

Este tipo de módulos es utilizado esencialmente para realizar retroalimentación de señales, ya que el módulo de lectura se ejecuta con las unidades generadoras; mientras que el de lectura, con los módulos terminales; lo que implicando que el dato estará disponible para la lectura en el siguiente ciclo de trabajo.

Capítulo 6: Análisis de resultado

Gracias a la utilización de la tarjeta de desarrollo para Pic32, se reduce el número de componentes necesarios para la realización del sistema UCPnet Kontrolo; por ejemplo, no se necesitó integrar en la placa base el conector para Ethernet. Tampoco se necesitó diseñar la fuente principal de energía, como se observa en la Figura 13 y Figura 16, ni los componentes básicos para que funcione el PIC32.

Por otro lado, la utilización de la tarjeta de desarrollo también contribuyó a la rápida programación del PIC, ya que dentro de la misma placa se incluye un sistema para programarlo y depurarlo.

A través del informe se mantuvo el esquema mostrado en la Código 1, lo cual facilitará la adaptación de la interface de entorno de desarrollo al sistema actual. Además, la mayoría de las sentencias se mantuvieron iguales, solamente se modificaron los argumentos de los filtros y las funciones de transferencia, que en este sistema reciben los parámetros definidos en tiempo continuo y se le agregó, además, la opción de poder limitar la señal integral en los controladores PID no convencionales, como se observa en el Código 15.

La utilización del esquema presentado en la Figura 17 permite a los módulos de entrada reducir posibles ruidos que se encuentren en la señal, situación que se vuelve crítica cuando las señales son del rango de los milivoltios.

El esquema de la Figura 20 acondiciona la señal de manera poco convencional, esto debido a la naturaleza de la señal que entra a este acondicionador, la cual siempre va a estar acotada por los mismos valores de tensión (0 y 2.024V), que entrega el convertidor digital/analógico. Por esta razón, en el primer operacional se agrega un valor de tensión cercano a 1.012V para las salidas que requieran un rango bipolar, y uno similar a 0 para cuando la salida está programada para tener un rango unipolar. El segundo operacional es el que se encarga, entonces, de escalar la señal para cumplir con el rango establecido.

Para poder trabajar de forma dinámica en los rangos establecidos en la Tabla 4 y en la Tabla 5, se utilizaron potenciómetros digitales. En los módulos de entrada, estos potenciómetros se utilizan tanto para lograr atenuar la señal en los casos que el rango sea superior a los 3.3V (tensión máxima soportada por el ADC) como para establecer la ganancia que se indica en la Ecuación 6. Mientras, en el módulo de salida sustituye a la resistencia R1 de la Figura 20 y no a la resistencia R2 –como es típicamente utilizado en los amplificadores inversores dinámicos–, debido a que la diferencia de tensión máxima entre terminales de R1 es de 5V mientras que en R2 es hasta de 24V, tensión que no soporta el potenciómetro.

Como se muestra en la Figura 16, el convertidor de DC-DC permitió alimentar el segundo amplificador de los canales del módulo de salida, y de esta forma se pudo amplificar la señal hasta el rango bipolar de 10V. También alimentó a los reguladores de tensión de ±5V, que junto con la referencia de 3.3V del “StarterKit” suministran la energía al resto de los componentes del sistema.

Gracias a la utilización del protocolo TCP/IP para programar la unidad con código BPC, no fue necesario implementar ninguna otra manera de comprobación de que la información llego inequívoca.

El uso del protocolo RTP permitió transmitir altas cantidades de datos, provenientes de las señales observadas por el sistema de monitorización, sin que estas se traslapen; esto gracias a la estampa de tiempo que se le anexa a la secuencia de datos binarios en formato de punto flotante, de modo que el computador receptor de dicha información puede hacer orden de esta y desplegarla como es debido.

A pesar de que el sistema cumple con la condición que indica que el resultado del proceso debe estar a la salida del controlador en al menos la décima parte del tiempo de muestreo, como se ve en la Figura 40, según la cual el tiempo de ejecución es de 80µs (menor de 100 µs, décima parte de 1 ms), se podría reducir este tiempo si se implementaran los flujos en algún formato diferente del punto flotante. Ya que este es demasiado lento en el cálculo y La constelación de números en que se puede llegar a representar es excesivamente superior a la cantidad de datos que se requiere para poder representar las señales, si se toma en cuenta que el valor máximo de paso que se puede obtener a la salida del convertidor análogo digital es de .4941mV.

Haber implementado los cálculos de constantes en la sección de configuración como se muestran en los apartados de Módulo de controles lineales/no lineales, eliminó la necesidad de calcularlos dentro de la tarea de flujo donde es crucial que cada módulo se ejecute de la forma más rápida posible.

Haber implementado el protocolo Konekto a través de la capa física del protocolo I2C, permite extender el número de periféricos que se pueden conectar a la unidad controladora de manera sencilla, esto gracias al sistema de direccionamiento que incorpora este protocolo. Por otro lado, gracias a la implementación de Konekto se pueden desarrollar y adaptar diferentes tipos de periféricos, como por ejemplo sistemas que utilicen una cámara como transductor.

Capítulo 7: Conclusiones y recomendaciones

7.1 Conclusiones

a. Utilizar el StartKit del Pic32 disminuye el número de módulos que se deben crear. Asimismo, el uso de este kit facilita el desarrollo de la etapa programada.

b. El Sistema UCPnet Kontrolo V1.1 facilita el diseño de circuitos de control, ya que posee diferentes herramientas de laboratorio como los generadores de señales y osciloscopios.

c. El Sistema UCPnet Kontrolo V1.1 es compatible en estructura con el lenguaje intermedio del BPC.

d. La utilización de un protocolo de comunicación basado sobre TCP/IP permite asegurar que el programa BPC llegue a la unidad controladora sin errores.

e. La utilización de un protocolo basado en RCP permitió transferir los datos monitoreados en computador en tiempo real y sin errores de traslapamiento temporal en los datos.

f. El Sistema UCPnet Kontrolo V1.1 permite crear y manipular señales que se encuentran en el intervalo de tensión que va desde los 50 mV hasta los 10 V.

g. Utilizar convertidores DC-DC permite que el sistema se pueda alimentar con una señal de USB; este es el adaptador energético más convencional en la actualidad.

h. La utilización de punto flotante reduce el error entre datos, sin embargo, aumenta el tiempo de ejecución de las operaciones.
i. El protocolo Konekto permite extender las interfaces para diferentes variables físicas con las que podría trabajar el sistema.

7.2 Recomendaciones

a. El circuito impreso debería ser creado por maquinaria especializada, con el fin de disminuir su tamaño y facilitar el correcto acople de los módulos funcionales.
b. Diseñar nuevas interfaces que se puedan acoplar mediante el protocolo Konekto y de esta forma obtener un laboratorio más sofisticado.
c. Modificar el sistema de punto flotante por un sistema que mapee toda la constelación de rangos utilizables, con el fin de obtener latencias menores en los procedimientos y, por consiguiente, bajar el tiempo de muestreo, como por ejemplo el formato Q32.
d. Todos los parámetros de nuevos módulos que permanecen constantes durante el flujo de ejecución deben ser calculados a lo sumo en la etapa de programación, para así evitar la dilatación del sistema con operaciones recurrentes.
e. Los registros generales deberían ser implementados de forma dinámica y no de forma estática, con el objetivo de permitir el mejor aprovechamiento del espacio de memoria.

Bibliografía

[1] Aja Fernández, Santiago. Teorema del muestreo [en línea] // Laboratorio de Procesado de Imagen. 5 de setiembre de 2011. *http://www.lpi.tel.uva.es/~santi/slweb/muestreo.pdf.*

[2] Alvarado Moya, José Pablo. *Procesamiento digital de señales* [libro]. Cartago: GNU, 2011.

[3] Analog Devices AD5280/AD5282 // Hoja técnica. Norwood : Analog Device, 2011.

[4] Barry, Richard. *The FreeRTOS reference manual* [libro]. Estados Unidos: NET, 2009.

[5] Caravaca Mora, Óscar Mauricio. *Diseño de un entorno de desarrollo integrado para una unidad controladora de procesos* [informe]. [s.l.]: IE-ITCR, 2010.

[6] Carnegie Mellon. Control Tutuorials for Matlab [en línea] // PID Tutorial. 13 de octubre de 2011. *http://www.engin.umich.edu/group/ctm/PID/PID.html.*

[7] Castro Molina, Daniel. *Unidad Controladora de Procesos* [informe]. - Cartago : IE-ITCR, 2008.

[8] Coughlin, Robert F. *Amplificadores operacionales y circuitos integrados lineales* [libro]. [s.l.]: Prentice-Hall Hispanoamérica. ISBN 968-880-284-0.

[9] Galeano, Gustavo. *Programación de Sistemas Embebidos en C* [libro]. Madrid, España : Alfaomega, 2009.

[10] Microchip Inc. (r) Filtros Digitales/Analogicos [en línea] // Filtro Antialiasing. 29 de setiembre de 2011. *http://ww1.microchip.com/downloads/en/AppNotes/00699b.pdf.*

[11] Microchip. MCP4728 // Hoja de técnica. N.A : Microchip, 2009.

[12] Microship Inc. (r) Microship StarterKits [en línea]. 9 de setiembre de 2011. *http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en545713.*

[13] Microsoft. El modelo TCP/IP [en línea]. TechNet. 19 de noviembre de 2011.

[14] MIPS Technologies. MIPS32(R) M4K(R) Processor core // Hoja tecnica. [s.l.]: MIPS, 2008.

[15] Proakis J.G *et ál*. *Digital Signal Processing: principles, algorithms and applications* [libro]. Englewood Cliffs, UK: Prentice Hall, 1998.

[16]Reinoso G. Óscar *et ál.* *Control de Sistemas Discretos* [libro]. Madrid: Shaum, 2004. Vol. 1.

[17] Texas Instruments. INA128/INA129 // Hoja técnica. Texas: TI, 2010.

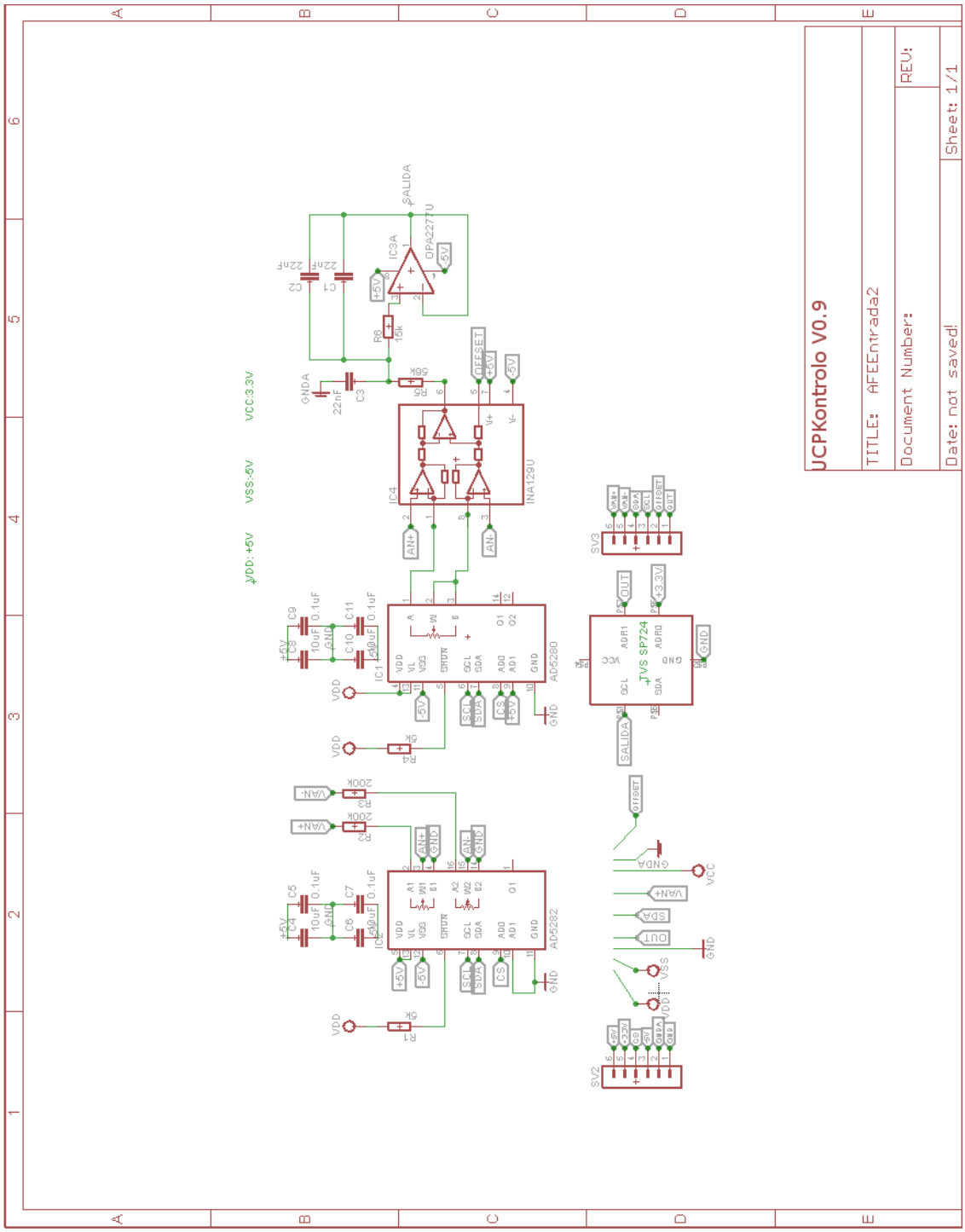
[18] UCP. Unidad de Control [en línea]. 19 de noviembre de 2011. *http://upcommons.upc.edu/pfc/bitstream/2099.1/3330/5/34059-5.pdf.*

[19] UGR. El modelo TCP/IP [en línea]. 19 de noviembre de *2011. http://ceres.ugr.es/~alumnos/redrs232/tcpip.htm.*

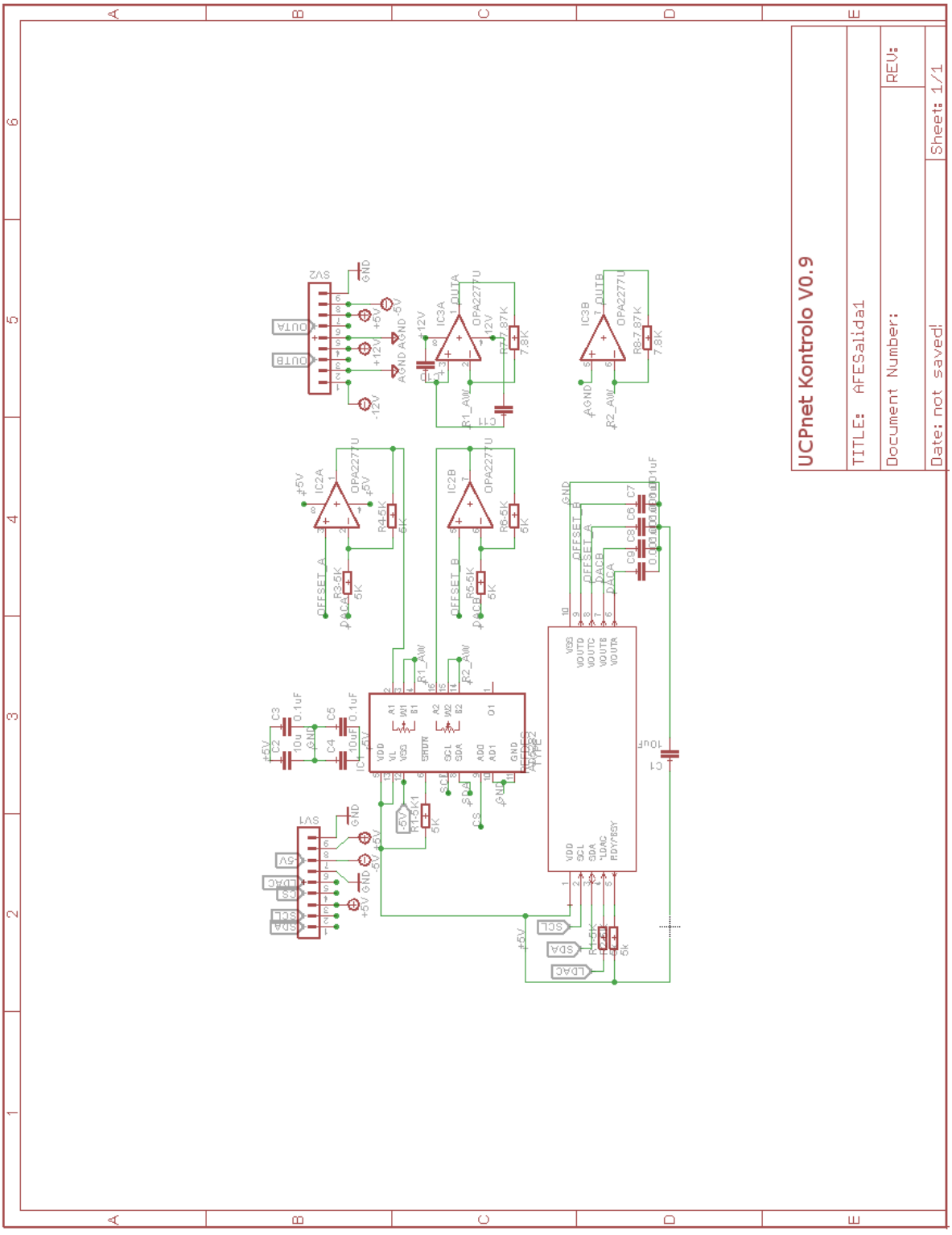
[20] Uniovi.es [en línea]. 10 de setiembre de 2011. *http://www.isa.uniovi.es/docencia/TiempoReal/Recursos/temas/sotr.pdf.*

[21] Wikipedia. RTP Prorocol [en línea]. 13 de setiembre de 2011. *http://es.wikipedia.org/wiki/Real-time_Transport_Protocol.*

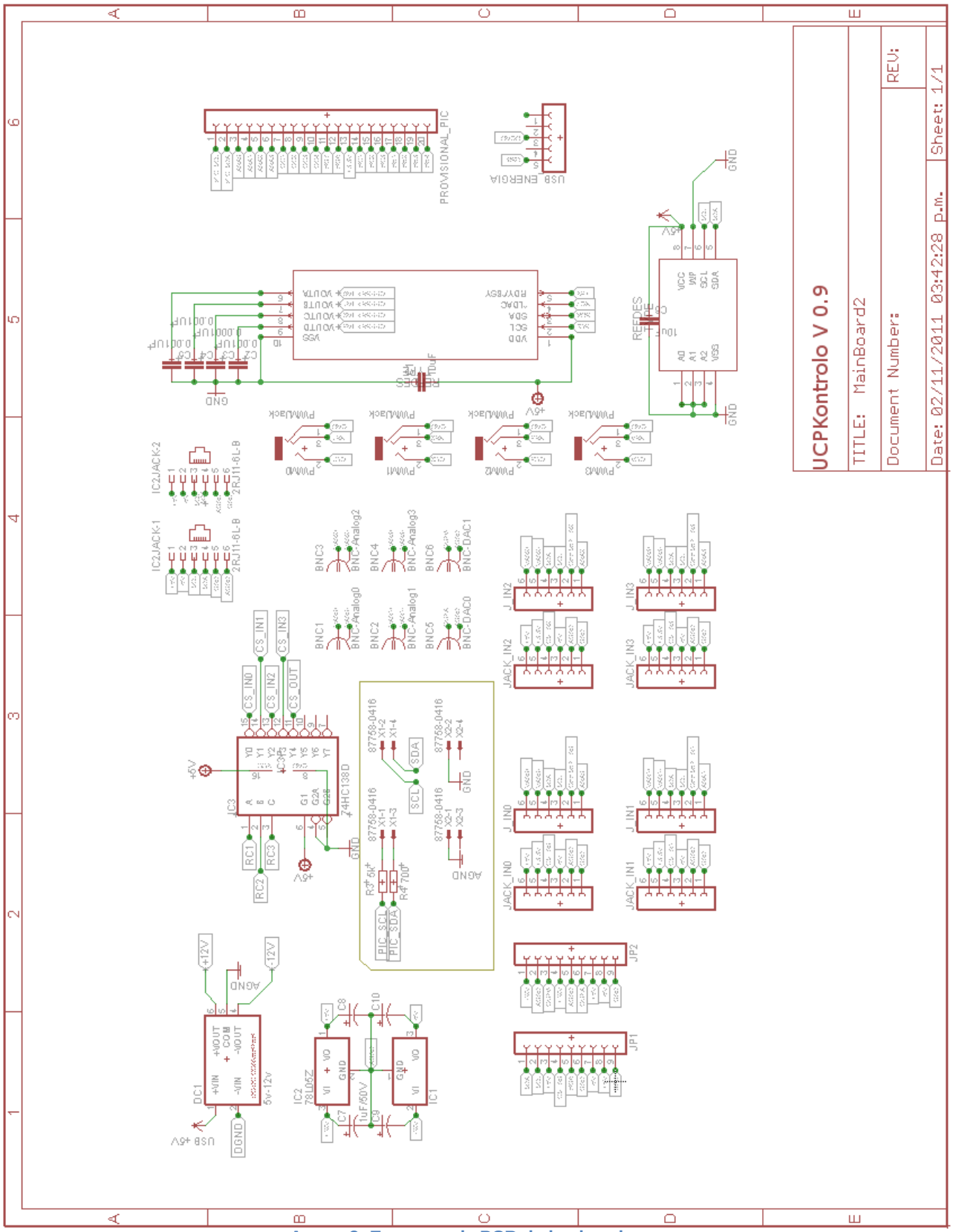
Anexos



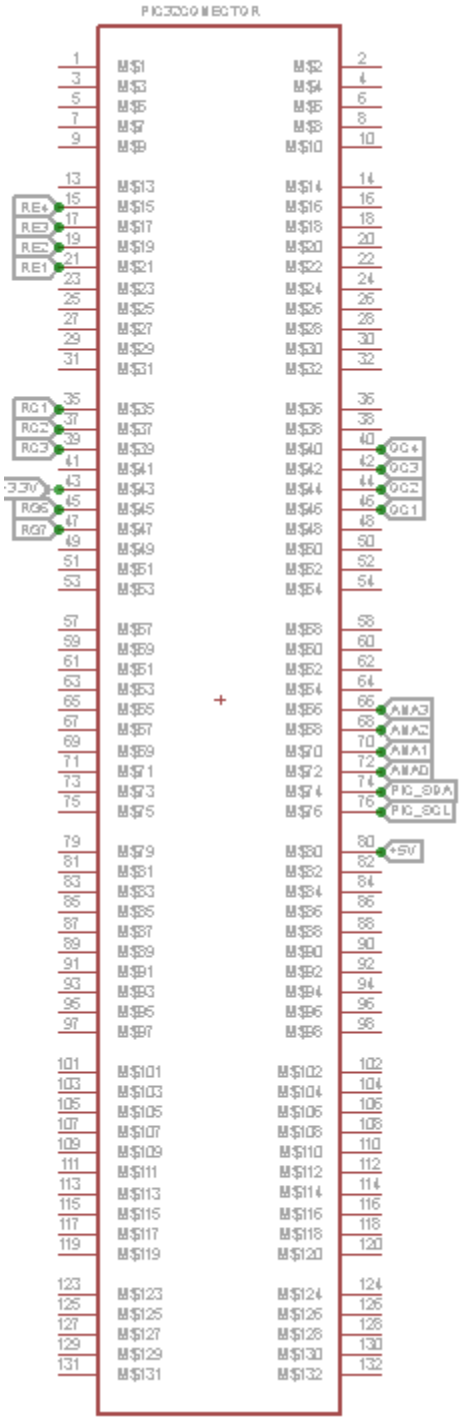
Anexo 1 Esquema del PCB para los módulos de entrada.



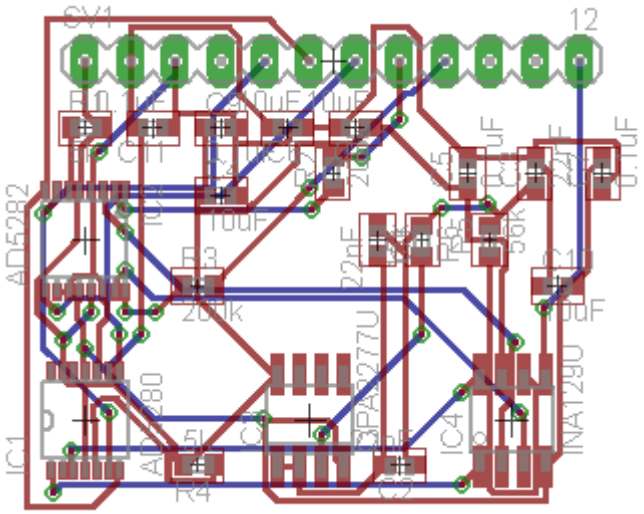
Anexo 2 Esquema de PCB para el módulo de salidas.



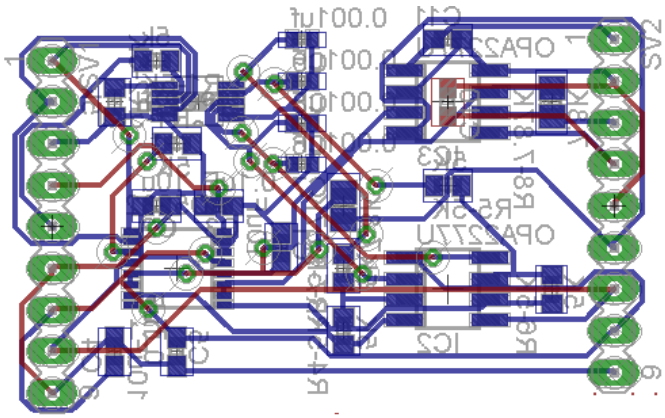
Anexo 3: Esquema de PCB de la placa base.



Anexo 4: Conexión de señales al StarterKit PIC32.



Anexo: 5 PCB para módulos de entrada (Escala 3:1).



Anexo 6: PCB para módulo de salida (Escala 2:1).