Universidad Zaragoza
1542

Proyecto Fin de Carrera

Ingeniería de telecomunicación

Estudio, diseño, construcción y prueba de un medidor digital de potencia para fibras ópticas de plástico

Autor:

José Lisbona Nicolás

Director:

Javier Mateo Gascón

Escuela de ingeniería y Arquitectura, Universidad de Zaragoza

2016

En este documento expongo el proceso de construcción y prueba de un medidor de potencia para fibras ópticas de plástico basado en una arquitectura digital, realizado en el laboratorio de comunicaciones ópticas de la Escuela de Ingeniería y Arquitectura de la Universidad de Zaragoza.

Resumen

En el trabajo con fibras ópticas de plástico (POF) es de particular interés la disponibilidad de un dispositivo de medida de potencia óptica que permita determinar la atenuación que introduce un elemento pasivo, como puede ser un tramo de fibra o atenuador variable, o bien, caracterizar la potencia emitida por un emisor óptico.

Se realizó un proyecto fin de carrera con un enfoque puramente analógico, lo que impide dotarlo de flexibilidad y facilidad de uso. El presente proyecto hallará una solución digital, constando de mucha mayor flexibilidad sobre el proceso de medida y permitiendo el tratamiento posterior de los datos, a través de un interfaz USB.

El proyecto comprenderá las fases de estudio, diseño, construcción y caracterización de un medidor de potencia óptica digital.

En la fase de estudio se detallarán los antecedentes al proyecto y el propósito del mismo. Se pretende analizar las prestaciones del medidor analógico, introduciendo la arquitectura general del sistema, y el método de medida empleado. Se dará una breve explicación de cómo abordar sus desventajas desde una perspectiva digital, buscando un enfoque eficiente a la vez que dotando al usuario de control sobre el método de medida.

En la fase de diseño se especificarán los bloques correspondientes al sistema, junto con su correspondiente arquitectura digital justificada, incluyendo la elección del microcontrolador más adecuado y los cálculos necesarios. Se trabajará partiendo de un PCB del microcontrolador elegido con la circuiteria básica de alimentación y control. Esto flexibilizará enormemente el desarrollo, facilitando la construcción rápida de los prototipos.

En la fase de construcción se explicará la implementación de los bloques anteriores en una placa de circuito impreso, junto con las posibles adaptaciones necesarias para la interconexión de los bloques, y las conexiones externas tanto de alimentación como de datos con el PCB del microcontrolador.

En la fase de caracterización se detallará el funcionamiento del medidor, con especial énfasis en la interfaz USB y el análisis de los datos, obtenidos con un protocolo de comunicación propio implementado en el microcontrolador y en dos programas de PC, uno de bajo nivel (control completo del medidor), y otro de alto nivel (calibración y obtención de medidas).

Agradecimientos

A todos los que confiaron y siguen confiando en mí, porque me dieron las fuerzas cuando realmente las necesitaba.

A mis compañeros y profesores de salsa, por permitirme conocer un mundo alegre y desenfadado.

A mis profesores del Instituto, porque gracias a ellos empecé la aventura.

A mis profesores en Santander, porque me abristeis el mundo en un año irrepetible.

A D.José Luis Villaroel, Da.Natalia Ayuso y D.Javier Nogueras, por estar ahí para cualquier consulta, en todo momento.

A José Luis Yécora, por facilitarme tanto la vida en el laboratorio.

A D.Javier Mateo, porque desde la primera clase me contagió su pasión por trabajar, y sentí que el proyecto no podía ser con otra persona o de otra manera. No hubiera podido ser posible sin ti.

A mi madre, mi tía Carmen y mi tío Salva, porque han estado conmigo día a día, en los momentos más dulces y en los más duros, y merecen todo mi respeto y cariño, hasta el fin de mis días.

Índice

Índice de figuras

Memoria

1. Estudio

1.1. Introducción

Desde el nacimiento de las telecomunicaciones gracias al telégrafo de Samuel Morse, el interés por aumentar el espectro frecuencial para la transmisión de lotes de información ha ido en aumento. Si bien los cables de cobre han sido el método más utilizado hasta la segunda mitad del siglo XX, la eclosión de las comunicaciones ópticas ha permitido dar el salto hacia una porción mas amplia del espectro, esto es, el ancho de banda óptico, tal y como podemos observar en la siguiente figura:

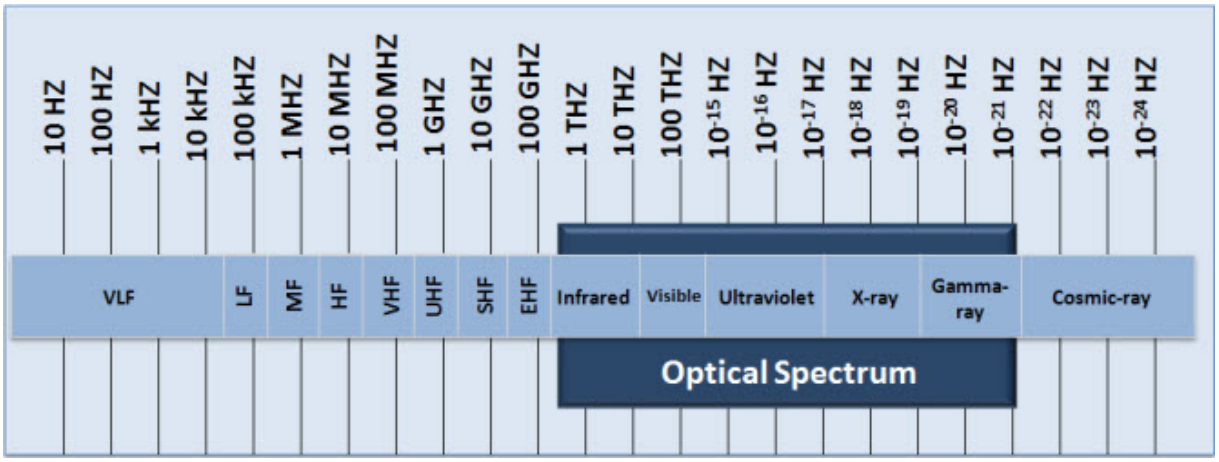


Figura 1: Espectro para comunicaciones ópticas. Fuente:Nasa.gov

La transmisión de datos por fibra óptica trae consigo una serie de ventajas respecto a la transmisión eléctrica, tales como:

- Mayor capacidad de transmisión, varios órdenes de magnitud mayor que con par de cobre.
- Menor atenuación, lo cual conlleva mayores distancias sin repetidores (hasta cientos de Km).
- Inmunidad frente a las interferencias externas.
- Mayor seguridad, ya que es muy difícil interceptar la información que viaja a través de una fibra.

Sin embargo, como inconvenientes tenemos:

- Mayor coste de la infraestructura necesaria, lo cual lo hace inviable para redes de corta distancia.
- Necesidad de equipamiento especializado de alto coste para su instalación.

Simultáneamente a los avances en fibras ópticas de vidrio, que fueron las primeras usadas en comunicaciones, se han producido distintos avances en fibras ópticas de plástico. Las primeras de este último tipo datan de 1960, con núcleo de PMMA(polimetacrilato).

Si bien las aplicaciones con fibras ópticas de vidrio han sido numerosas debido a sus características de baja atenuación y gran ancho de banda de transmisión, el uso de fibras ópticas de plástico con aplicaciones comerciales no ha sido tan acusado, y ha empezado en las últimas dós décadas, debido a las mejoras en atenuación y ancho de banda que las sitúa como un medio de transmisión con muy bajo coste, inmunidad frente a interferencias y gran flexibilidad, en contraste con el alto coste de los sistemas con fibra de vidrio y las interferencias inherentes a los sistemas eléctricos. Por ello, el campo de los sensores y las redes PON son sólo algunos de los usos de este tipo de fibras, que pretenden superar las limitaciones que tienen en redes de corta distancia las redes de fibra óptica de vidrio.

Tanto es así, que han sido varios los proyectos de la Universidad de Zaragoza versando distintas aplicaciones, incluyendo sensores, de las fibras ópticas de plástico. Las características de estas fibras, hoy en día, nada tienen que ver con las primeras cuyas atenuaciones hacían inviable la comunicación más allá de unos metros, con atenuaciones no menores a los cientos de dB/km. Distintos tipos de fibras de plástico podemos encontrar hoy en día, siendo un campo en plena evolución.

Por un lado están las fibras de salto de índice o SIPOF, cuyo ancho de banda es de 12.5 MHz/Km y atenuación de 180 dB/Km, basadas en un núcleo de PMMA. Además podemos encontrar fibras de índice gradual o GIPOF, cuyo potencial es de transmisión a 3 Gb/s sobre 100 m de cable, con 16 dB/Km a una longitud de onda de 650 m. Distintos materiales experimentales mejoran estas prestaciones, el futuro de las fibras ópticas de plástico está por llegar, asi como la especificación de un estándar IEEE que estandarice las comunicaciones a corta distancia y con gran ancho de banda mediante estas fibras.

1.2. Antecedentes

En el contexto de las aplicaciones de las fibras ópticas de plástico, se desarrolló hace 10 años un proyecto en la Universidad de Zaragoza para contruir un medidor de potencia, un intrumento poco accesible en el momento para este tipo de fibras.

Si bien el proyecto fue exitoso y el comportamiento del medidor era bastante adecuado, el manejo del mismo y un análisis profundo dejaban entrever que podía ser mejorado en distintos aspectos.

Uno de ellos es la automatización del proceso en todas sus etapas. El ajuste manual de la amplificación y de la coma decimal hacían algo farragosa la obtención de los datos, y el establecimiento de curvas experimentales debía hacerse también de manera manual, a base de anotar observaciones.

Otro de ellos es la mejora en la tecnología que se ha producido en este tiempo, que nos permite disponer de mejores transmisores y detectores para fibras ópticas de plástico, y también de disponer de encapsulados para el resto de tareas de bajo coste. De esta manera no sólo el proceso puede ser más cómodo sino que además más preciso y más barato.

Y un último interés de revisar el proyecto es la investigación. Mediante el rediseño del mismo podemos conocer más detalles de cómo influyen los distintos parámetros de los que disponemos, permitiendo obtener un diseño más abierto, a la par que funcional.

1.3. Propuesta

Y es por todo lo anterior que en este proyecto se trata de revisar las etapas analógicas del anterior proyecto, y sustituirlas por bloques digitales, añadiendo versatilidad al conjunto.

Son controlables digitalmente todos los parámetros del sistema que se han considerado de interés. En el caso de la emisión se pueden alternar periodos de luz con periodos de no luz, lo cual se denominará emisión **chopeada**. Tanto la frecuencia como la potencia de luz emitida se pueden controlar. En el caso de la recepción se puede controlar la ganancia de las distintas etapas en cascada que la componen. Además, los datos pueden ser procesados con un filtrado IIR, y ser entregados periódicamente, o bajo demanda.

Tanto el control como la respuesta han sido diseñadas siguiendo un protocolo serie estándar, del que se proporcionan los comandos disponibles. A modo de demostración de las posibilidades, se han hecho dos programas en el lenguaje de desarrollo .NET de PC que los utilizan, y uno en lenguaje Matlab.

2. Diseño

2.1. Arquitectura general del sistema

El diseño del sistema ha sido modular, facilitando así la puesta en marcha y las modificaciones cuando han sido necesarias.

A grandes rasgos, los bloques que compondrán nuestro sistema son los siguientes:

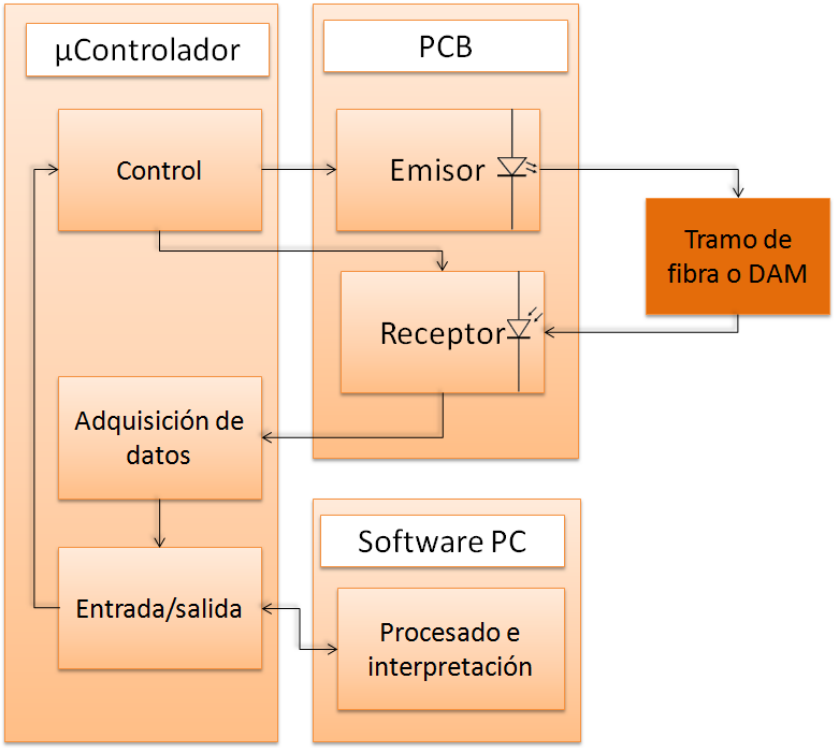


Figura 2: Diagrama de bloques del sistema

Diseñaremos un emisor, que, controlado por el uControlador, proporcionará la potencia de referencia para la medida de la atenuación.

Tras circular la luz por el tramo de fibra o por el Dispositivo A Medir (DAM), tendremos un receptor que hará la conversión a señal eléctrica analógica, y la entregará al uControlador en su rango de conversión analógico-digital.

Una vez adquirida la muestra, el uControlador posibilitará el envío del dato a un software de PC externo, asimismo dando control al usuario sobre el proceso para el análisis posterior de los datos.

Ese análisis junto con el correspondiente calibrado serán funciones del software de PC. Los programas realizados se han clasificado en 'bajo nivel'y 'alto nivel'. El 'bajo nivel' corresponde a permitir el control completo del uControlador, para la obtención de medidas, envío de comandos y recepción de datos, mientras que el 'alto nivel' permite la calibración para la obtención de medidas indirectas (esto es, en unidades de medida diferentes a las del conversor A/D del uControlador). De esta manera se ofrece la posibilidad de cambiar todos los parámetros de control del uControlador manualmente, o tener una idea de la aplicación del medidor a alto nivel.

Los datos se muestrean con una frecuencia constante en el microcontrolador, y son en-

tregados por el puerto serie de una de dos maneras:

- Al recibir un número fijo de datos (Por ejemplo, cada mil muestras).
- Al recibir un comando de petición de dato. (Y los datos que ocurren entre medias pueden ser descartados)

Además de esto, se puede establecer un filtro IIR, de manera que se van acumulando los datos para ganar precisión, a costa de añadir algo de procesado en el microcontrolador. La convergencia de dicho filtro fue probada en Matlab, y corresponde a la siguiente ecuación:

$$y_n = \frac{63y_{n-1} + 64x}{64} \tag{1}$$

En cuanto a los datos recibidos, en lo concerniente a los valores de luz recibida, se definen los siguientes valores:

Techo Es el valor recibido al emitir luz. Está codificado en los 10 bits del conversor A/D(si el filtro IIR no está activado), o valor salida del filtro IIR de 16 bits (si el filtro IIR está activado).

Suelo Valor recibido en ausencia de luz emitida.

Resta Valor diferencia entre el valor techo y el valor suelo, es el valor que realmente nos informa de la potencia de luz recibida.

En resumen, los datos que principalmente deberá gestionar el software de PC corresponden al muestreo de la luz recibida, alternando semiperiodos de emisión y de no emisión de luz, y que se denominará emisión **chopeada**. Se verá en más detalle en la sección 2.4.

2.2. El microcontrolador

El mercado nos ofrece múltiples alternativas para la elección de un microcontrolador adecuado a nuestros requerimientos. Necesitamos un dispositivo que tenga un número aceptable de puertos de entrada y salida, que tenga una velocidad suficiente como para poder adquirir las muestras y procesarlas, junto con el control del emisor, y que, a ser posible, sea sencillo de programar y tenga un bajo consumo.

A su vez, podemos decantarnos por microcontroladores sueltos o placas de desarrollo que ya los incluyen, junto con una conexión con el PC que sirve tanto para programar como, en algunos casos, para establecer una comunicación puerto serie.

Por la rapidez en cuanto a tiempo de desarrollo, y la disponibilidad en el mismo circuito de circuitos de alimentación regulados y la interfaz USB, nos decantamos por una placa de desarrollo sencilla, el Arduino Uno.

Figura 3: Arduino Uno

Una de las principales ventajas de optar por esta plataforma es que además de contar con las funciones que necesitamos, es un microcontrolador ampliamente soportado por una comunidad de software abierto, es decir, que hay mucha documentación tanto para la arquitectura y la programacion, como a la hora de realizar todo tipo de conexiones con periféricos externos.

Las características del microcontrolador que vamos a utilizar son las siguientes, para una descripción más completa, consúltese el anexo 2.

- Dos salidas digitales de alta resolución (16bits) para el control de la potencia emitida y el offset del receptor.
- Una salida digital de baja resolución (8 bits) para el control del periodo de la señal chopeada a transmitir.
- La posibilidad de sincronizarnos por hardware con la señal transmitida, de manera que la adquisición del dato recibido sea en el instante adecuado.
- Comunicación serie estándar (SPI) para establecer los parámetros de los amplificadores de recepción (PGAs).
- Conversión AD de 10 bits para la recepción de los datos.

Una vez conocido el microcontrolador a utilizar, se detallarán las distintas conexiones necesarias con el PCB.

2.3. Conexionado general microcontrolador - PCB

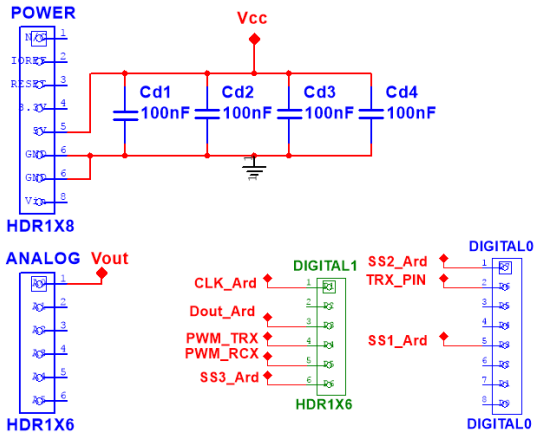


Figura 4: Conexión entre Arduino y el PCB

He aquí las conexiones necesarias entre la placa de Arduino y nuestra placa. Por una parte tenemos la alimentación de los componentes a 5 V (Vcc), que está llevada a tierra con condensadores de un valor intermedio, para filtrar ruido y posibles picos de tensión de alta frecuencia procedentes del procesador o del resto de componentes digitales.

Por otra parte el emisor requerirá de dos salidas digitales, una PWM para controlar la intensidad emitida (PWM_TRX), y otra lógica para conmutar el emisor (TRX_PIN).

El receptor requerirá de las salidas del puerto SPI, que será compartido para todos los chips de recepción, de manera que habra dos pines comunes (pines CLK_Ard y Dout_Ard) y un pin adicional de habilitación para cada dispositivo, que es el que determinará cuál de ellos recibe los datos en cada momento (pines SSx_Ard). También Se incluye una salida PWM para controlar el offset del receptor (pin PWM_RCX)

Finalmente, necesitamos capturar el dato de salida, para lo cual usamos una de las entradas con conversión analógico-digital (pin Vout)

Partiendo de estas conexiones, veamos en detalle las etapas de emisión y recepción diseñadas en el PCB.

2.4. Arquitectura del emisor

Los requisitos mínimos para el emisor óptico son:

- Robustez, inherente a tratarse del diseño de un equipo de medida. La potencia emitida debe fluctuar lo menos posible.
- Sencillez, es decir mínimo número de componentes, manteniendo un coste bajo.
- Control de la intensidad emitida.
- Posibilidad de emisión mediante chopping (Emisión de una onda cuadrada de ciclo variable alternando períodos de emisión y no emisión de luz).

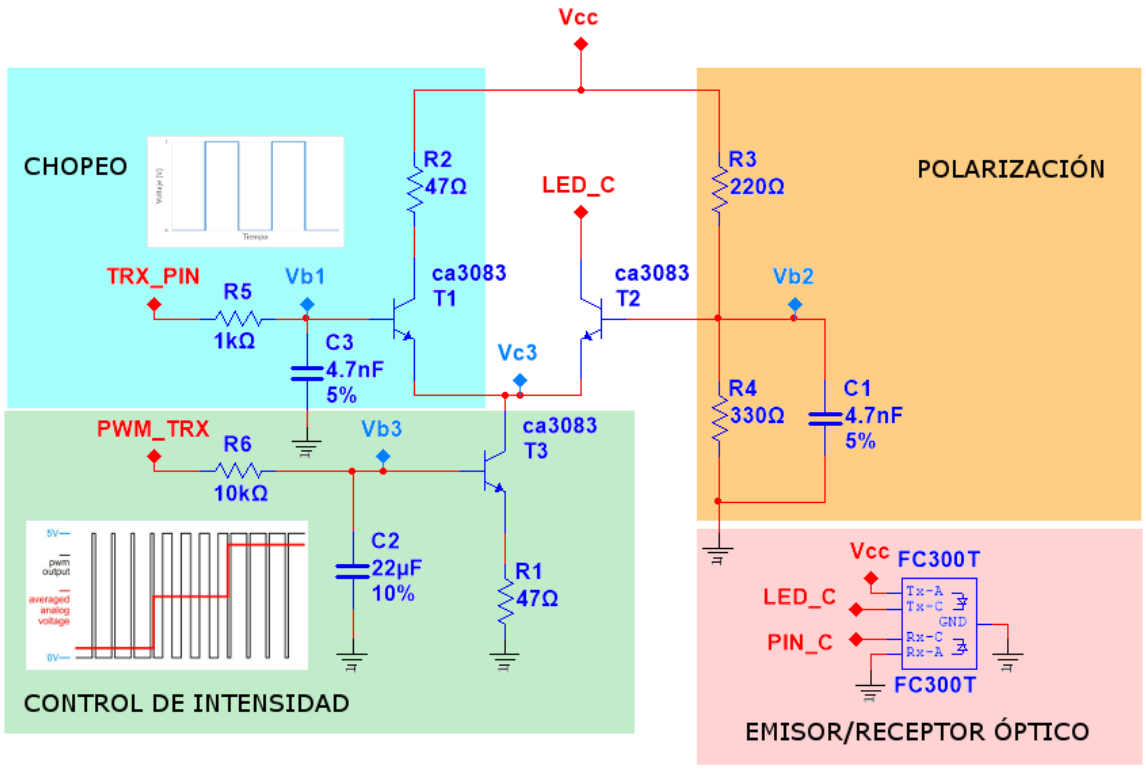


Figura 5: Esquema del emisor

La arquitectura de emisión más básica consiste en un inversor con un único transistor bipolar que suministra la corriente que necesita el led (LED_C). La emisión chopeada es posible con una onda cuadrada a la entrada, así como el control de la intensidad, pero las variaciones térmicas de la intensidad pueden ser considerables. Es por ello que se añade una rama inferior ($T3$) que controla la intensidad. Esto conlleva dos cambios en la parte superior, un divisor de resistencias para asegurar la polarización del fotodiodo, asegurando 3V en la base de T2 (Vb2), y una rama simétrica que consuma la intensidad en los semiperíodos de no emisión.

El funcionamiento de las ramas simétricas es el siguiente: en primer lugar, partimos de que la rama R3-R4 fuerza 3V en Vb2. Si se corta el transistor T1 (Forzando TRX_PIN=0V)

el paso de toda la intensidad proporcionada por por la rama inferior (T3) es a través del transistor T2, que debido a que tiene 3V en la base está en zona activa y por tanto la intensidad circulará por el LED, sin embargo, si TRX_PIN=5V, T1 se satura, lo cual impone en Vc3 un voltaje mayor a 3V, cortando T2 y por tanto el fotodiodo no emite luz.

Sin embargo es necesario el suministro de la corriente por parte de la rama inferior. En ella, podemos controlar la intensidad suministrada a los superiores mediante la resistencia R_1. Para ello, a mayor voltaje en la base, mayor intensidad de colector en $T3$, siempre que superemos el umbral minimo de $V_{BE} = 0{,}7V$, y esto es posible variando el voltaje PWM_TRX. Un valor importante a tener en cuenta es el valor de la intensidad maxima que circulará por el LED, ya que de sobrepasar el valor máximo establecido por el fabricante (100 mA) es posible que el fotodiodo se destruya. Este valor es (PWM_TRX(max)-V_{be3})/R_1, de manera que, con los valores implementados (PWM_TRX(max)=5V, $R_1 = 47\Omega$, el valor teórico máximo es de 91.5 mA. No obstante, luego veremos, en las curvas de caracterización del emisor, los márgenes reales y la zona de trabajo.

Para detalles de la implementación de los PWM, consúltese la sección 2.2.2

Por último, los condensadores añadidos tienen la función de filtrar, especialmente C2, que debe filtrar la señal PWM para convertirla en un voltaje analógico. El orden de magnitud de R3 y R4 en las decenas de Ω permite que el voltaje en el fotodiodo sea estable, pero sin comprometer el consumo del conjunto.

2.5. Interfaz emisor-uControlador

Partiendo de la figura 6 del apartado anterior continúo describiendo los elementos necesarios.

Los dos elementos a controlar son las tensiones de base de T1 y T3.

Para la tensión de base de $T1$ tenemos que asegurar que la intensidad demandada por el transistor no supera los 40 mA máximos por pin que soporta Arduino, para lo cual tenemos una resistencia en la base, $R5$. Con 1K la intensidad máxima de salida serán 5mA. En cuanto a la forma de la onda, el uControlador puede generar sin problemas una onda cuadrada de ciclo variable, que obtendremos a través de la salida OC0A del timer 0 en el modo CTC.

Para Vb3, sin embargo, necesitamos suministrar un voltaje lo más constante posible para mantener la intensidad en el LED constante. Desafortunadamente, el Atmel 328p no dispone de conversor D/A, y, por tanto, no disponemos de una salida analógica para controlar directamente la base. No obstante sí disponemos de salidas PWM, que podemos filtrar para generar un voltaje constante. Es ésta una práctica habitual, y tan sólo necesitamos dimensionar adecuadamente la frecuencia del PWM generado, de manera que sea sencilla de filtrar con un filtro de orden uno, y, a la vez, que tengamos el mayor número de pasos de PWM posibles, lo que determinará la resolución de salida de este conversor A/D.

Siendo f=16MHz el tiempo de conteo más rápido disponible (y por tanto el que nos dará la f_{PWM} mayor), debemos elegir la resolución que queremos, de manera que tengamos una precisión de mV y una frecuencia no menor a 10KHz para poder filtrar con componentes discretos de valores comunes (un C no electrolítico, por ejemplo). Fijando directamente f=10KHz, tenemos que un período de conteo serán 1600 ciclos del reloj del timer. Si establecemos un PWM automatico con doble conteo, de manera que la fase no cambie bruscamente al cambiar el valor PWM, el periodo a fijar serán 800 ciclos para el voltaje maximo, y de 1 ciclo para el mínimo. El PWM automático tiene la ventaja de dejar libre de recursos al microprocesador, salvo cuando queramos cambiar el valor del propio PWM.

El paso que tenemos con estos valores es de $V = \frac{5}{800}V = \frac{50}{8}mV = 6,25mV$, siendo menor el efectivo ya que la resistencia de filtrado paso bajo se quedará con parte del voltaje generado como efecto de limitar la corriente de salida del transistor U1C con la resistencia R6. Tener más precisión implicaria una frecuencia de PWM menor, y una frecuencia mayor implicaría menor precisión, de ahí que se ha llegado a un buen compromiso entre ambos.

Para el filtrado del PWM, hay que asegurar que el rizado es mínimo, ya que cualquier oscilación en la base del transistor debido a la naturaleza digital del PWM será una oscilación en la luz emitida. Con un valor de condensador de 22uF, y el valor estándar de R6 mencionado de $10K\Omega$, la frecuencia de corte a 3dB es menor a 1Hz, con lo cual el rizado de frecuencias 10KHz o superiores (que son las que componen la señal PWM) es despreciable.

2.6. Arquitectura del receptor

El receptor consta de un preamplificador de poca ganancia y bajo ruido, y de tres etapas amplificadoras programables.

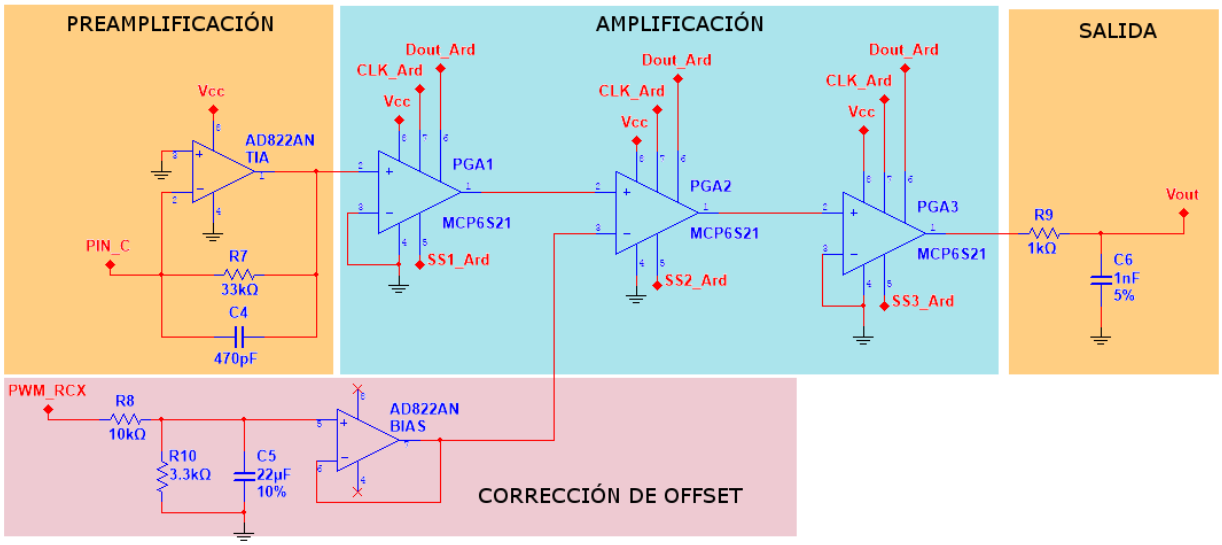


Figura 6: Etapas del receptor analogico

El preamplificador es indispensable en el diseño de este tipo de sistemas, tanto por la necesaria primera conversión Intensidad-Voltaje, como por la necesidad de una etapa de bajo ruido para recuperar señales muy degradadas, ya que tan importante es tener un diodo receptor sensible como tener una etapa eléctrica que sea capaz de atender dicha sensibilidad. El $AD822AN$ se trata de un amplificador operacional con función rail to rail unipolar, de manera que alimentándolo con 5V, como al resto de los componentes, nos entrega una señal dentro de ese rango. Este requisito, junto con característica de bajo ruido y offset de intensidad nos lo ofrece el AD822. La ganancia teórica de esta etapa es $V_{out} = R_7 I$, de manera que $R7$ debe ser lo más grande posible, teniendo en cuenta que no sature la salida del operacional al recibir la cantidad máxima de luz y que el ruido que introduzca no sea excesivo, ya que el ruido térmico crece con el valor de R.
El condensador tiene como fin un primer filtrado paso bajo del ruido entrante, de ahí su valor pequeño.

Para las etapas amplificadoras posteriores disponemos de un amplificador programable, el $MCP6S21$, pues es sencillo de manejar mediante el estándar SPI, de bajo ruido y característica cercana al rail-to-rail unipolar.
Dado que una sola etapa nos proporciona una insuficiente ganancia de 32, hemos de poner varios amplificadores en cascada, el segundo de los cuales tuvo que ser modificado para tener un offset controlable, ya que si no con ganancias altas los valores chopeados sin emisión (suelos) eran amplificados debido al offset de las anteriores etapas. Para ese menester, disponemos de un segundo operacional en el $AD822AN$, que configuramos como seguidor de tensión, y atenuamos mediante R10, ya que el offset que queremos introducir es pequeño. De esta manera, sin modificar la configuración del PWM anterior, podemos aumentar la precisión de 6.25 mV, a costa de tener una señal máxima también menor. El valor de R8 y R10 está pensado para disminuir aproximadamente en un factor 4 la tensión de entrada.

Una vez amplificada la señal, a la salida de $R9$, es necesario además el acondicionamiento de los datos para enviarlos a través del puerto serie, de manera que $R9$ y $C6$ hacen la funcion de filtrado antialiasing, y de paso un último filtrado paso bajo. La frecuencia de adquisición de los datos es controlable por software.

2.7. Interfaz receptor-uControlador

Para poder controlar el receptor se ha desarrollado un sistema de comandos, de manera que el envío de los datos sea automático o bajo demanda, buscando siempre que el programa en el microcontrolador sea simple, y el procesado de la información sea en el PC. Se detalla a continuación el control de las distintas etapas.

De entre los amplificadores de ganancia programable (PGAs) disponibles en el mercado, los tenemos con distintas maneras de control, véase a traves de resistencias externas, de voltajes externos, y los hay controlables digitalmente, principalmente por puertos SPI o I2C. El interfaz SPI es el elegido por ser sencillo de manejar,y no requiere de un hardware complicado, lo cual implica dispositivos de bajo coste. Los $MCP6S21$ en este aspecto cuentan con buena documentación y con notas de aplicación en el campo de los sensores que nos facilitan familiarizarnos con ellos. En las notas de características tenemos detallado el funcionamiento del SPI del PGA, que desarrollamos en una rutina. En conjunto podemos amplificar en un rango discreto de 1 a 32768 (32^3), puesto que tenemos tres etapas en cascada, cada una de ellas con ganancia de hasta 32.

El control de varios PGAs con el mismo puerto SPI conlleva compartir todas las entradas SPI salvo la entrada de SS. Una vez hecho, hay que rediseñar el software y repartir la ganancia de forma adecuada. De acuerdo a los principios de minimización del ruido en etapas en cascada, la mayor ganancia debe distribuirse siempre hacia la primera etapa.

Una descripción ampliada de los puertos SPI tanto del Atmel 328p como del MCP6S21 puede obtenerse en los anexos 2 y 3.4.

Tras la amplificación se requiere un filtrado antialiasing para el conversor AD. Dicho filtrado es el más simple, un filtro paso bajo de orden uno, con frecuencia de corte menor a 62.5KHz ya que ésta será la frecuencia de Nyquist, es decir la mitad de la frecuencia máxima del conversor, que es 125KHz cumpliendo con la recomendación de Atmel (rango 50KHz - 200KHz para resolución máxima de 10 bits). Con los valores de resistencia y condensador propuestos la frecuencia de corte es de 15 KHz, lo cual daría una atenuación algo mayor a 27 dB a la frecuencia de Nyquist (atenuación de 6dB por octava).

La configuración del conversor AD es para conversión simple, que iniciaremos unos instantes antes del cambio de valor del transmisor, que es cuando el valor ha tenido más tiempo para estabilizarse.

2.8. Modificaciones al diseño original

2.8.1. Ganancia receptora

En primera instancia la decisión de diseño fue tener dos etapas en cascada, pero la ganancia resultó insuficiente en los primeros prototipos a la hora de buscar la sensibilidad del sistema. Una vez ajustado el software para dos PGAs, nada limitaba el empleo de tres, como se hizo en un segundo rediseño del PCB.

Sin embargo, una vez hecho esto, resultó que el offset de tensión del primer amplificador se confundía tanto con las señales muy atenuadas que ambos son amplificados por igual. Para ello, se añadió un circuito de compensación para el offset, que veremos a continuación ($BIAS$).

2.8.2. Compensación del offset del receptor

El circuito de compensación de offset consiste en un PWM filtrado, equivalente al del transmisor, pero con un seguidor de tensión a continuación, que realiza la adaptación de impedancias entre la salida y la entrada de offset del segundo PGA, disponiendo asi del rango de salida completo 0-5V. El valor inicial de offset es nulo, y mediante el programa de Matlab puede variarse, en el caso de necesidad de las ganancias más altas, que es cuando se hace patente la necesidad de este circuito. La variación de éste voltaje con la precisión de 6.25 mV hacía inviable el análisis del comportamiento, de ahí que se añadiese $R10$ para forzar valores menores y mayor precisión por igual.

Sin embargo, es necesario añadir que hay un cierto offset tanto en la salida como en la entrada de los PGAs, de manera que no se ha conseguido el resultado esperado de poder controlar el offset de una manera simple mediante la señal PWM de recepción.

3. Implementación

3.1. Software en el Arduino

El software incluido en el Arduino tiene como objetivo adquirir muestras de la luz recibida (a través de un pin de recepción analógico/digital) y de responder a los comandos serie que le llegan para controlar el proceso.

Es éste un programa de microcontrolador. Como en la mayoría de microcontroladores sencillos, el programa consta de dos partes, una inicial en la que se configura todo, y otra que se ejecuta en bucle.

Como elementos que configurar tenemos, en la parte de emisión:

- Potencia PWM para la emisión (en el diagrama 5 es PWM_TRX).
- Frecuencia para los pulsos de luz emitidos, y configuración del chopeo (en el diagrama 5 es TRX_PIN).

A su vez, para la recepción:

- Configuración de los pines para el control de los amplificadores de recepción programables digitalmente o PGAs. (Ver parte de amplificación en el diagrama 6
- Configuración del offset para el segundo amplificador (en el diagrama 6 es PWM_RCX).

Y, además del control del hardware tenemos los siguientes añadidos:

- Activación o desactivación de filtro IIR al capturar las muestras.
- Configuración del puerto serie.
- Gestión de la entrega de datos por el puerto serie, entregando los datos capturados de manera constante cada N muestras, o esperando a que los datos sean pedidos desde el puerto serie (es decir, desde el programa de PC).

Todo lo anterior se configura en la primera de las dos partes comentadas del programa, la de arranque, con los siguientes valores por defecto:

- Potencia de emisión ->230 (valor PWM) (Consultar el anexo 1 para más detalles)
- Periodo del pulso de emisión 822 Hz.
- Offset para la recepción Voffset=0V)
- Ganancias de los tres amplificadores programables de valor 1.
- Modo chopeado para la emisión, es decir, se emite luz un semiperiodo, y se deja de emitir en el siguiente.
- Los datos se entregan cuando los pide el software de PC.
- El filtrado IIR está desactivado.

Esta parte inicial se cargará siempre que arranque el sistema. Una vez termina, empieza el algoritmo que se ejecuta en bucle, que ejecuta las siguientes tareas:

- Procesar una muestra de recepción y enviarla por puerto serie, si corresponde.
- Comprobar si hay comandos externos a los que responder por puerto serie.

Considérese que la emisión es chopeada, es decir, que se emite una señal cuadrada. A la hora de hacer la conversión analógico digital, se elige un instante fijo antes de que la señal cuadrada cambie para iniciar la conversión, y se gestionan de manera independiente al bucle principal, mediante interrupciones. Éste es un pequeño diagrama que permite aclarar la adquisición de datos.

Para que funcione este esquema, hay que elegir el periodo de la señal cuadrada de manera adecuada, y configurar a bajo nivel el chip Atmel que forma parte de Arduino. Además, para que el bucle principal y la captura de datos esté sincronizada, se realizaron medidas de cuánto cuesta la ejecución de iteraciones del bucle principal, de manera que puedan ejecutarse entre captura y captura de dato sin problemas.

En resumidas cuentas, lo que se hace en esta parte en bucle es procesar un dato que ha sido capturado, ver si hay que responder por puerto serie a algún comando, y dormir el microprocesador para esperar a la siguiente captura de dato.

3.1.1. Protocolo de comunicación con Arduino

El formato del puerto serie para los comandos que utiliza este software de Arduino es:

Velocidad: 115200 bps, 8N1

Formato de los comandos El siguiente cuadro muestra el formato de los comandos que permiten controlar el comportamiento del Arduino.

Comando	Efecto	Valores posibles	Respuesta
#P0XXX	Valor PWM emisor	0 - 800 0 - 5V	ACK P/XXX
#Q0XXX	Valor PWM receptor	0 - 800 0 - 5V	ACK Q/XXX
#T00XX	Ganancia PGA1	1/4/5/8/10/16/32	ACK G1/XX
#U00XX	Ganancia PGA1	1/4/5/8/10/16/32	ACK G2/XX
#V00XX	Ganancia PGA3	1/4/5/8/10/16/32	ACK G3/XX
#C000X	Modo chopeado on\off	0 off; !=0 on	OK C ON\OFF
#D0XXX	Periodo del pulso, múltiplo de 32 us	0 - 255 125KHz -122 Hz	ACK D/XXX
#M0000	Peticion de dato	0 (devolver dato)	T(dato)S(dato)
#MXXXX	Intervalo de envío de dato (cada M muestras)	1 - 999	ACK M/XXXX
#RXXXX	Reset	—	ACK RESET
#FXXXX	Activar filtrado IIR	0 **OFF** otro caso **ON**	ACK F/XXXX
#WXXXX	Info valores actuales	—	INFO C/D/M/P/Q/- T/U/V/F
Otros	Comando desconocido	—	ERR/

Cuadro 1: Protocolo serie de comunicaciones con Arduino

Descripción del funcionamiento del protocolo

Al empezar a funcionar Arduino lo hace con los siguientes valores por defecto, que corresponden a lo comentado anteriormente:

- P0230 (Potencia de emision aproximadamente -6dBm)
- Q0000 (Voffset=0V)
- T0001
- U0001
- V0001 (Ganancia para los tres PGAS G1=G2=G3=1)
- C0001 (chop on)
- D0118 (822 Hz)

- M0000 (Los datos se sacan bajo demanda)
- F0000 (Filtrado IIR desactivado)

Todos los comandos son de 6bytes, comenzando por el carácter '#' (carácter ASCII 23), y tras la recepción de un comando se reinician los valores temporales de tal manera que no se entreguen por el puerto serie valores incorrectos debidos a promediados previos no finalizados. Para facilitar la recepción por software, el carácter de finalización de todas las respuestas es el retorno de carro o CR (carácter ASCII 13)

Es importante reseñar que los datos se pueden pedir de dos maneras, o bien pidiéndole al equipo que nos lo envíe con una cierta cadencia (cada N muestras), lo cual se haria con el comando #M000N, o pidiéndoselo al equipo en cualquier momento mandándole un #M0000 (en cuyo caso se manda el dato y se dejan de enviar datos de forma automática si asi estaba configurado).

Formato de las muestras convertidas

La trama empieza con el carácter 'T' seguido de un valor numérico, que representa el valor con luz o 'techo', seguido del carácter 'S' que representa el valor en ausencia de luz. En modo no chopeado ambos valores son de muestras con luz. El caracter fin de linea '\n' (ASCII 13), finaliza la trama.
Formato de datos: 'T %dS %d \n'
Por ejemplo, 'T610S3\n' indica que el valor con luz es 610 y el valor sin luz es 3.

3.2. Software en el PC

Si bien se ha detallado el protocolo serie de manera que se pueda interactuar con la placa desde cualquier programa o dispositivo, para facilitar el manejo de la placa se ofrece el software 'Arduino POF Manager '. El objetivo de esta sección es presentar la funcionalidad del software junto con una explicación de cómo debemos de utilizarlo correctamente para efectuar las medidas oportunas.

3.2.1. Primeros pasos en los programas

En primer lugar, tras la conexión de la placa a un puerto usb del PC, debemos seleccionar el puerto COM al que está conectada. Si hemos arrancado el programa antes de conectar la placa, podemos averiguarlo dándole al boton refrescar antes y después de haberlo conectado, ya que el puerto aparecerá listado sólo la segunda vez.

NOTA: la primera vez que se conecta a un ordenador requiere que se instalen los drivers para el control serie de Arduino. Esta instalación es automática en Windows 7 y posteriores, para versiones de Windows anteriores u otros sistemas operativos puede que sea necesaria la descarga e instalación de los drivers desde www.arduino.cc

3.2.2. Programa de Matlab

Los primeros pasos con el sistema se dieron en la plataforma Matlab, por la familiaridad adquirida con el lenguaje.

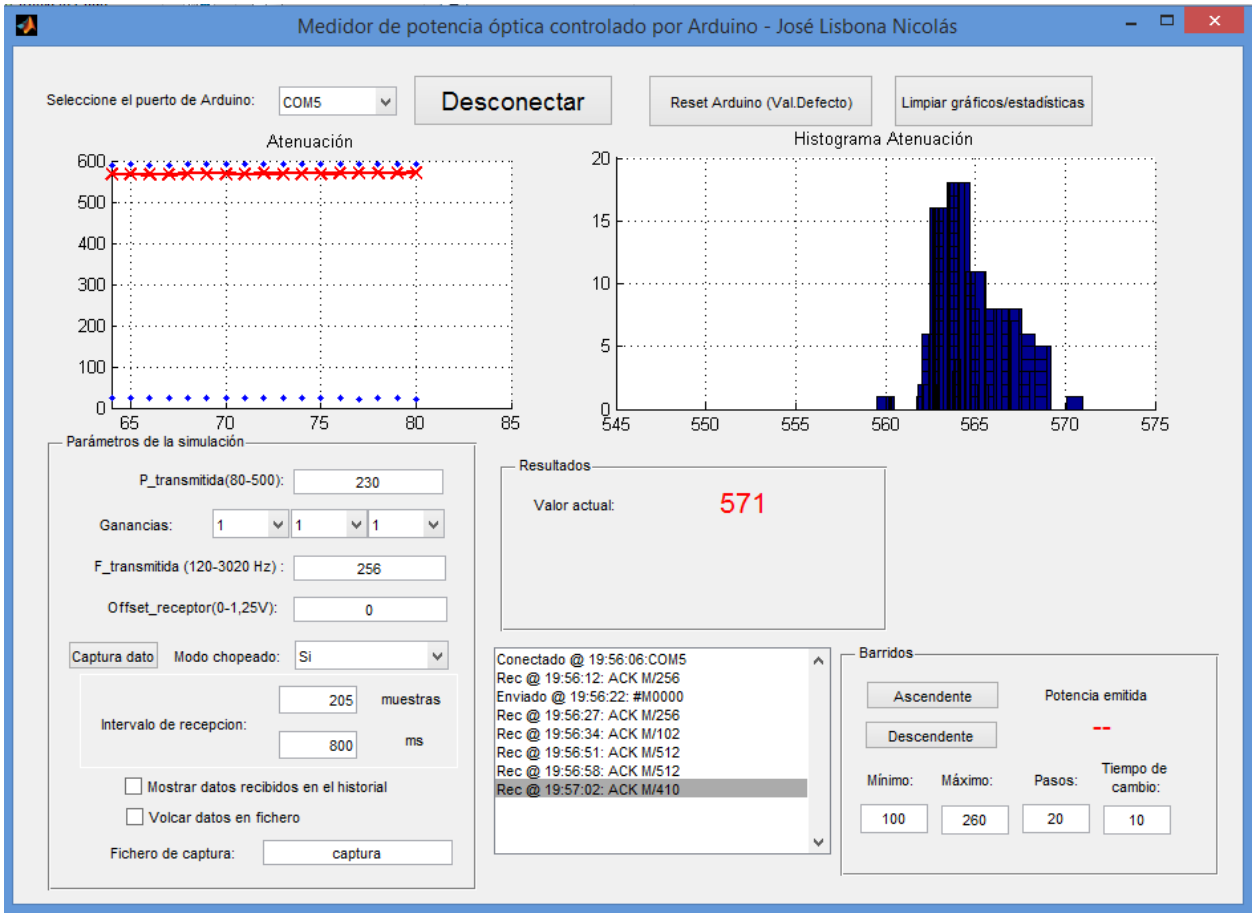


Figura 7: Captura de pantalla del programa de Matlab

Para el interfaz gráfico se utilizó GUIDE, una utilidad interna de Matlab, acompañada de un fichero .m que contiene el código de los distintos eventos que pueden ocurrir (pulsaciones de botones, disponibilidad de datos en el puerto serie... etc). La idea del interfaz era hacerlo similar al disponible en uno de los medidores de potencia comerciales disponibles de Thorlabs, que además del valor medido muestra un histograma con los valores para poder ver las derivas de los valores de potencia en el tiempo. Además se le añaden los comandos de control de Arduino de manera que se ejecuten por debajo del interfaz gráfico.

La principal desventaja, además de la poca portabilidad del programa, es el no muy fiable control del puerto serie que dispone Matlab, que obliga, por ejemplo, a reiniciarlo para poder detectarlo en caso de desconectar el cable.

No obstante, se puede utilizar y se adjunta el código en uno de los anexos.

Para la captura de datos se puede proceder de la siguiente manera:

- Tras abrir el programa, seleccionar el puerto COM de la lista y pulsar Conectar.
- Dado que por defecto Arduino no envia automáticamente datos, y el programa no los pide, debemos pulsar el boton captura dato para recibir un valor, o escribir un número

de muestras o de ms en el cuadro inferior de intervalo de recepción, de manera que se vayan recibiendo datos automáticamente.

Como características adicionales se incorpora en este programa un barrido de potencia emitida, de manera que se puede comprobar la coherencia del sistema, y está la posibilidad de exportar los datos recibidos a un fichero.

En la figura se puede observar una captura. En el gráfico de la izquierda (Atenuación) se marcan con puntos azules los valores recibidos con luz y sin luz. En rojo se marca la resta de ambos valores, que es el valor que realmente interesa, y el que se acaba mostrando numéricamente tanto en la sección resultados como en el histograma de atenuación, donde podemos ver la frecuencia relativa de cada valor en el histórico de valores.

3.2.3. Programa de bajo nivel

Este es uno de los dos programas escritos en .NET, en este caso sirve para controlar los parámetros de la misma manera que se hacía en MATLAB.

El funcionamiento del mismo es el siguiente: una vez seleccionado el puerto, pulsamos a conectar, lo cual permite que el programa abra el puerto serie y esté a la espera de enviar/recibir comandos. En un primer arranque, con el Arduino arrancado con el programa por defecto (es decir, tras ejecutar el programa por primera vez tras conectar el cable) tendremos la siguiente interfaz:

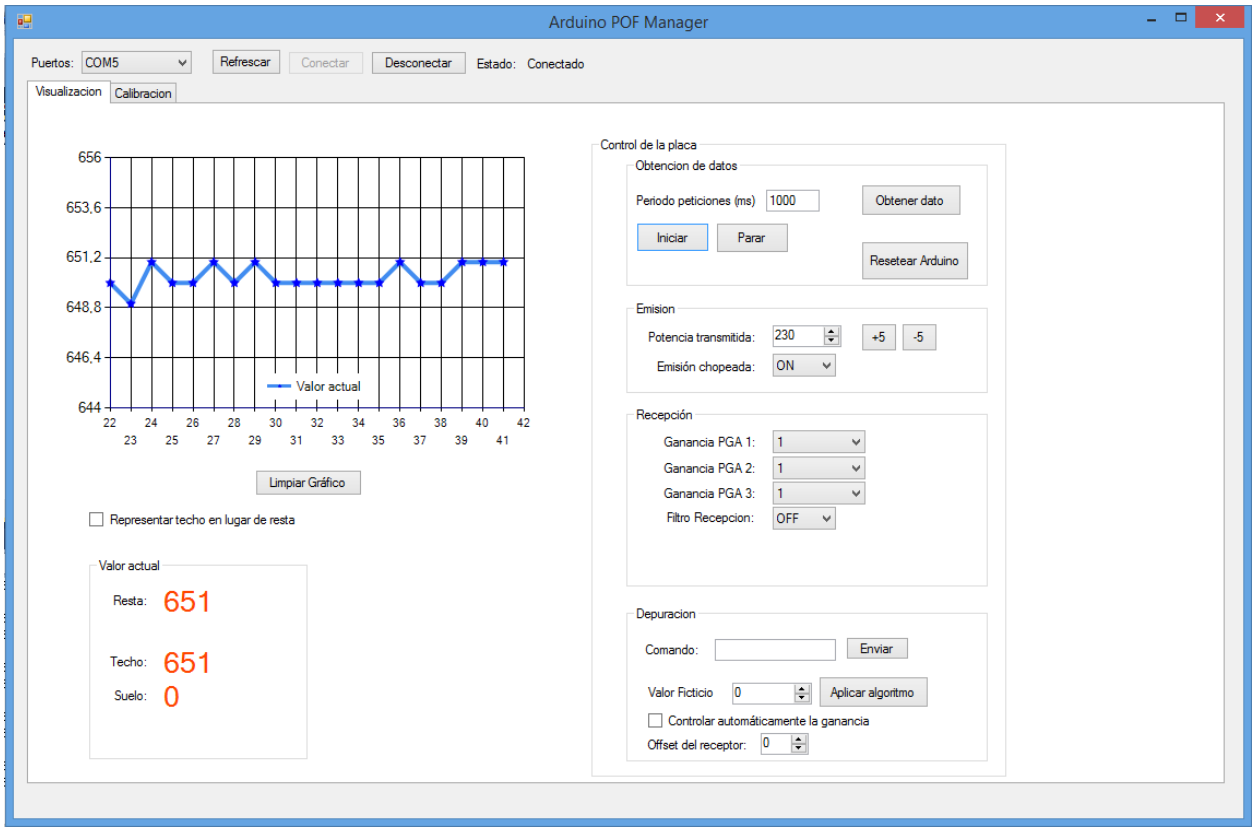


Figura 8: Captura de pantalla del programa de bajo nivel

Tenemos dos secciones, la izquierda para visualizar los datos, y la derecha para el control de la placa.

En la parte de visualización, tendremos un gráfico, que se mostrará al recibir los primeros datos desde el Arduino, además de los valores resto, techo y suelo. Se comentará más adelante en detalle qué significan.

A su vez, en la parte de control de la placa tenemos distintas subsecciones.

En la parte de obtención de datos podemos Iniciar o parar la petición automática de muestras, con el intervalo de petición que se quiera introducir, teniendo un valor por defecto de 1 segundo. También podemos obtener los datos de manera manual pulsando el boton obtener dato, o devolver al Arduino a la configuración del programa por defecto.

Bajo el subpanel de emisión podemos controlar si la emisión la hacemos chopeada (esto es, mandar pulsos de luz o un haz continuo) y con qué potencia, según los valores PWM posibles (de 80 a 800).

Por otra parte, en la pestaña de recepción podemos controlar las ganancias de los amplificadores de recepción y si debe usarse un filtrado IIR desde el propio Arduino para la obtención de los datos.

Finalmente, hay una subsección de depuración que no estaria en una versión final, pero que se adjunta para mostrar el envio de los comandos serie detallados en la anterior seccion.

El proceso de captura de datos es bastante sencillo. Tras conectarse al puerto serie tal y como se decía en la parte de primeros pasos, pulsamos iniciar, de manera que vayamos recibiendo datos cada segundo. Así, veremos una sucesión de valores resta y suelo en el gráfico, asi como el último valor recibido de tipo resta, techo y suelo. La explicación del significado de estos valores se dispone en el apartado 2.1

Aclarado este punto, pasamos al programa de alto nivel.

3.2.4. Programa de alto nivel

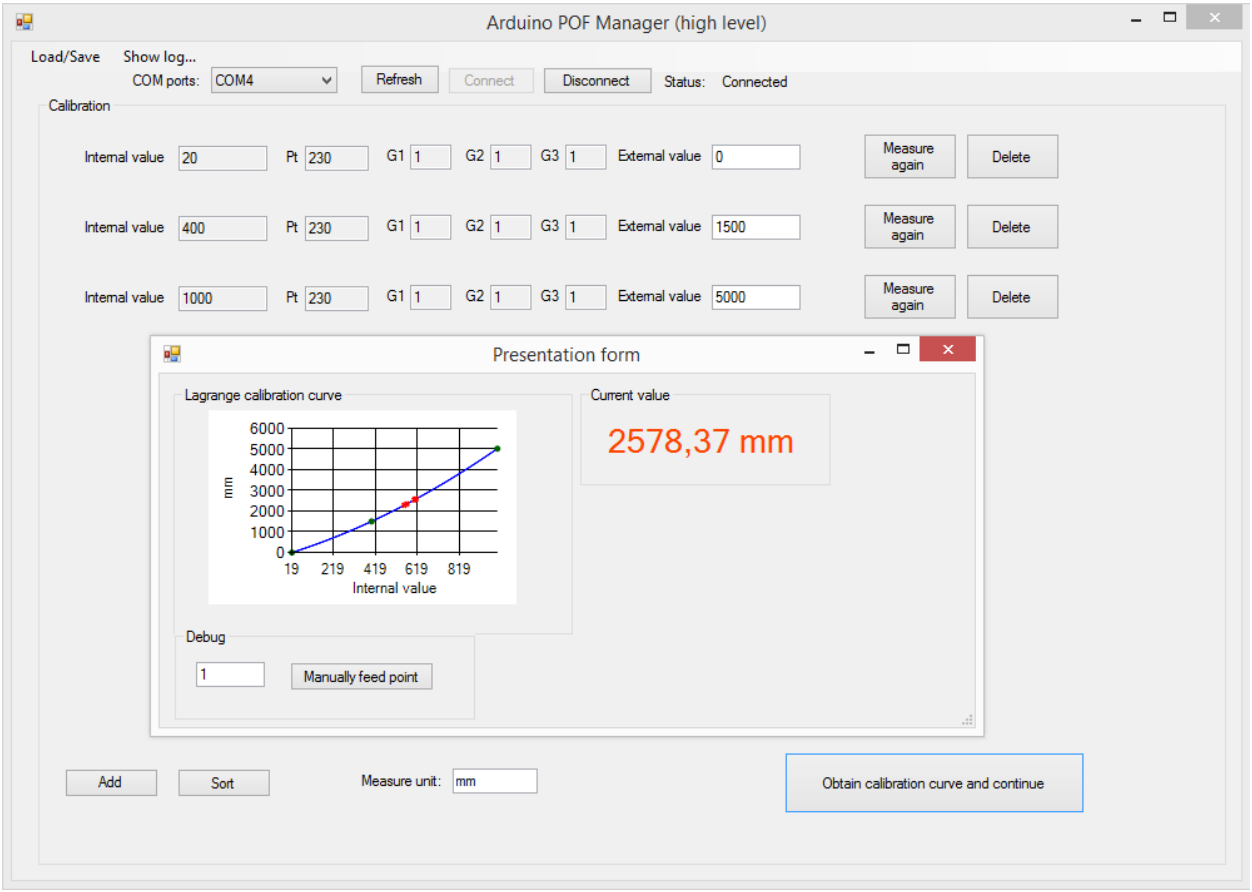


Figura 9: Captura del programa de alto nivel

En esta segunda parte lo que se hace es calibrar el dispositivo para mostrar los datos según la magnitud que nos interese.

Es decir, se hace una transformación de unidades, de la interna del conversor, a la externa que nosotros introduzcamos, de manera que, por ejemplo, podría asociarse un valor de luz a una longitud de fibra en cm, a la concentración de una disolución de sal en la que la fibra estuviese sumergida... es decir de un caso físico que plantee una variación de la potencia de luz recibida.

Para iniciar el proceso de calibración debemos de conectarnos al puerto serie. Una vez hecho esto tenemos dos posibilidades: añadir puntos de calibración manualmente (botón **Add** inferior), o cargarlos de un fichero (Menú **Load** superior).

En caso de añadir los puntos manualmente, se pulsa el botón **Measure again** para tomar la medida del punto, para el cual podemos establecer los valores de potencia y ganancia que deseemos, una vez hecho le damos a start, e irán apareciendo los valores que se leen del Arduino. Podemos darle a **Save current value** en cualquier momento para guardarlo.

Tanto si se ha añadido manualmente como si se ha cargado de archivo, tendremos que introducir la magnitud de la unidad de medida externa, que podemos introducir en el cuadro de texto **Measure unit** de la parte inferior.

Una vez tengamos los puntos requeridos, debemos pulsar **Obtain calibration curve and continue**, para que se muestre la curva de calibración mediante el método de Lagrange.

En el gráfico, los puntos verdes son los que sirven de base para la curva de Lagrange (en negro), y los puntos rojos son los datos que son leídos en tiempo real y trasformados a la unidad externa.

Una vez vistas las posibilidades, hay que señalar que la principal carencia de este programa de alto nivel en su versión actual es la no posibilidad de cambiar la ganancia al medir. Idealmente al obtener un punto de calibración debería de gestionarse internamente el control de ganancia para aumentarla en caso de que no llegase suficiente luz. Este cambio conllevaria además complicar la curva de Lagrange, ya que habría que tener en cuenta puntos con distinta ganancia.

3.3. Diseño de PCB

Tras las pruebas en la placa de prototipado, se procedió a la fabricación de la siguiente placa de circuito impreso. Los límites de la misma vienen determinados por la distribución de los pines del Arduino, y la ubicación del FC300T en la misma cara que la alimentación USB del Arduino.

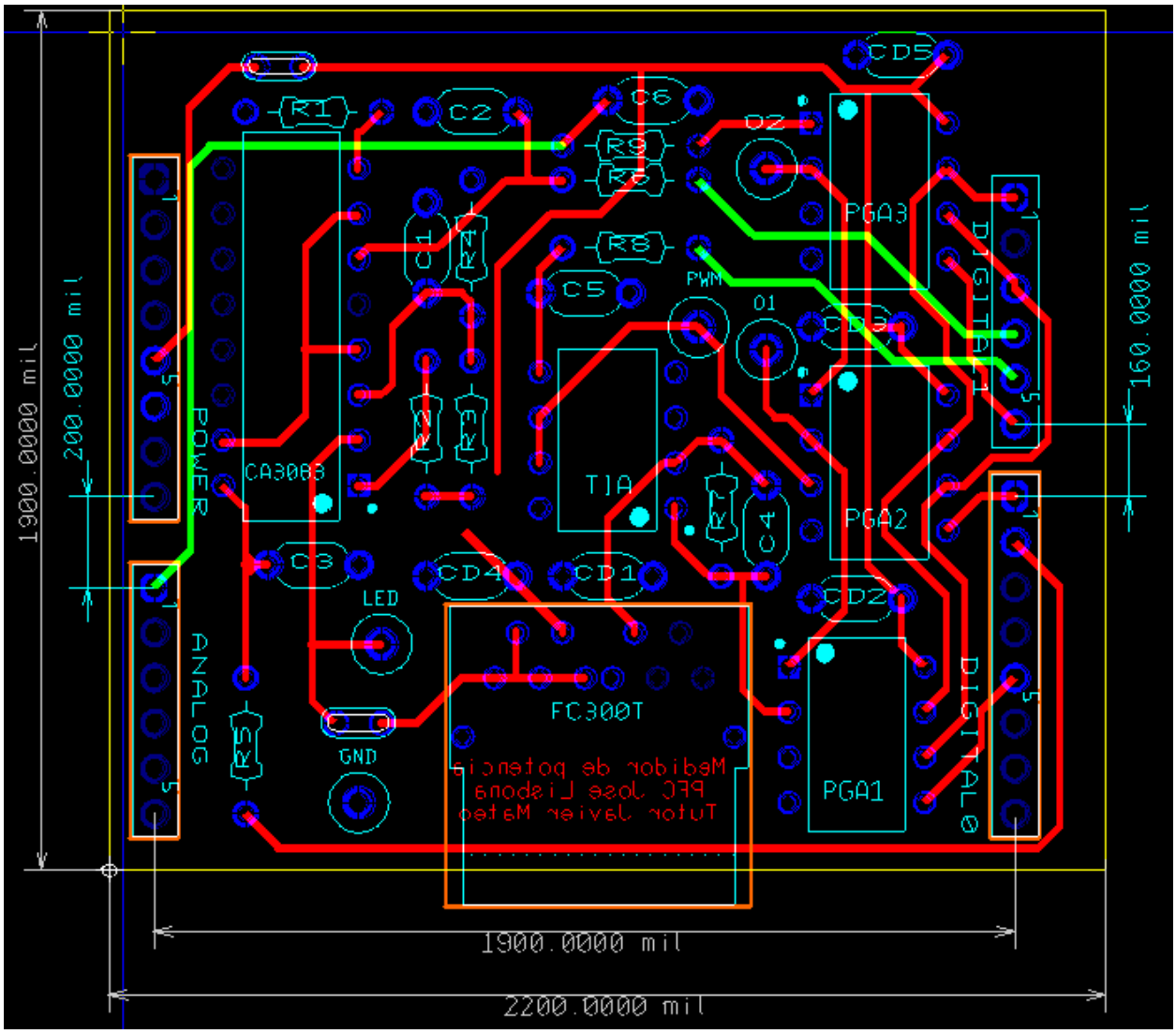


Figura 10: Placa de circuito impreso

Una vez construido, montado, y funcionando, éste es el aspecto de la placa montada sobre el Arduino:

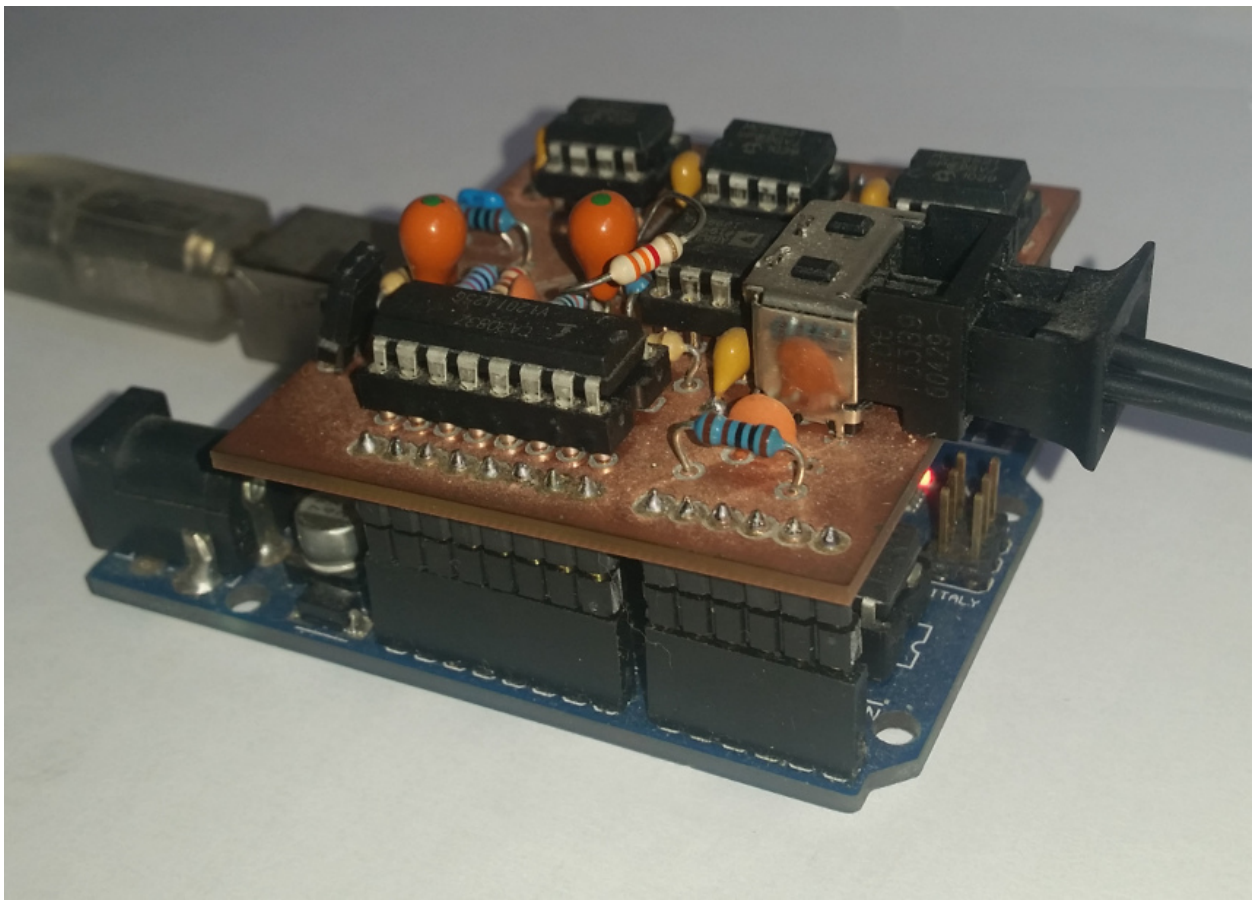


Figura 11: Foto del sistema completo

4. Posibles mejoras

En este apartado voy a relatar posibles mejoras que, a mi juicio, podrían hacerse, en el caso de futuros proyectos fin de carrera con temática similar.

En primer lugar, un cambio de sistema receptor. En lugar de tener un PC alimentando el sistema y obtener los datos localmente, podría acomplarse al UART un módulo bluetooth y la alimentación ser bajo batería. Dado que la potencia transmitida es un parámetro de control y que el mayor consumo viene dado por la rama de resistencias de polarización del diodo, podrían cambiarse éstas resistencias a costa de perder algo de estabilidad en el voltaje. De esta manera, sin mayor trabajo que el de programar una aplicación para un teléfono, podrían obtenerse resultados similares.

Otra opción similar sería un sistema desatendido. Con un procesador más especializado en tratamiento de señales y el control de una pantalla podrían presentarse diferentes datos correspondientes a la simulación.

Asimismo, la resolución del ADC en sistemas donde sea importante la precisión tanto como la verosimilitud puede quedarse bastante pequeña, con sólo 10 bits. El uso de un ADC externo convenientemente aislado de ruido podría ser de utilidad, habiendo modelos con conversión de hasta 24 bits.

Hubiese sido interesante disponer de más tiempo para poder realizar una aplicación más grande y que abarcara junta tanto la calibración como el control automático de la placa, así como el diseño de algún tipo de experimento con fibras de plástico que permitiera mostrar la versatilidad de este prototipo frente a los anteriores.

5. Conclusiones

A lo largo de estas páginas he intentado condensar el trabajo realizado durante meses, parte en el laboratorio de comunicaciones ópticas, y otra buena parte en mi ordenador personal.

El mayor interés de este proyecto ha sido aunar distintas ramas de conocimiento, desde los conceptos generales de lo que se pretendía hacer hasta el diseño de los bloques que componen el sistema, requiriendo conocimientos de óptica, electrónica y programación a distintos niveles.

Así pues, con distintas dificultades se ha llegado a disponer de un sistema funcional, que responde a las pretensiones iniciales, y que me ha permitido tener una perspectiva clara de los problemas que van surgiendo en todo diseño de prototipos. Para mi ha sido lo más enriquecedor, poder dedicar horas de esfuerzo y ver un resultado final fruto de cada línea de código, de cada pista del PCB, de elegir el valor de cada componente, de la elección de un integrado y no otro.

Este proyecto abre la vía a otros similares basados en microcontroladores, donde no es necesaria una gran cantidad de conocimientos previos, sino tener claro el concepto de lo que se quiere construir, hacer un estudio de lo que hay en el mercado en ese momento, y desarrollar una placa personalizada si es necesario añadir funcionalidad al microcontrolador como en este caso. Gran cantidad de ideas surgen en el camino y el resultado es un prototipo adecuado a las necesidades iniciales, de bajo consumo y de fácil integración a un producto final.

Referencias

[1] Ramón Pallás Areny. *Sensores y acondicionadores de señal*. Marcombo, 1994.

[2] Atmel Corporation. *ATmega 48A/PA/88A/PA/168A/PA/328/P datasheet*. Atmel, 2012.

[3] Diego Orlando Barragán Guerrero. *Manual de Interfaz Gráfica de Usuario en Matlab*. Web, 2008.

[4] Mario Lou. Medidor potencia óptica basado en fop. Master's thesis, Universidad de Zaragoza, 2012.

[5] playground.arduino.cc. *ArduinoSleepCode*. playground.arduino.cc/Learning/ArduinoSleepCode, 2012.

[6] Wikibooks. *Source Code Listings*. en.wikibooks.org/wiki/LaTeX/Source_Code_Listings, 2012.

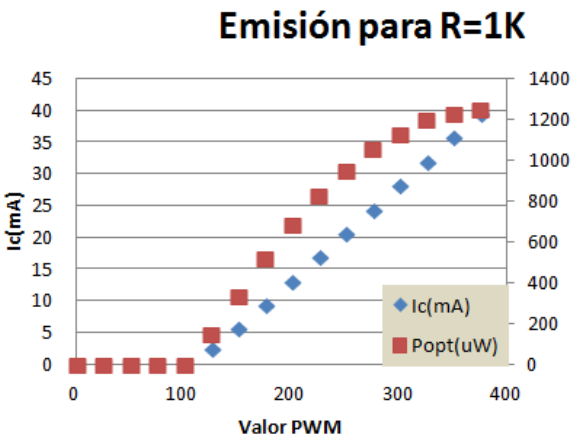
Anexo

1. Elección de resistencia para emisión

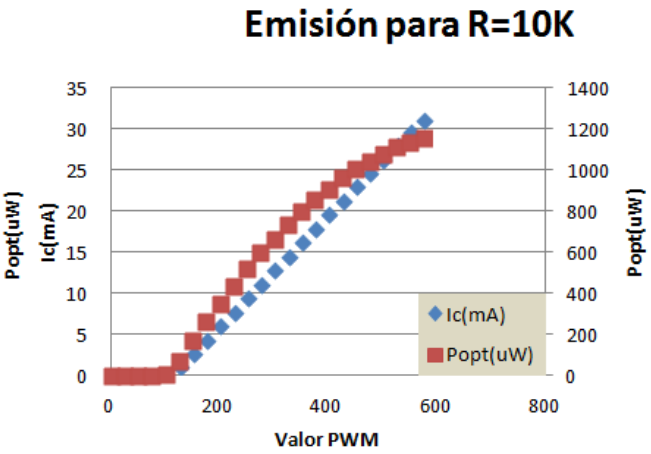
La resistencia R_6 del esquema de emisor (página 19 de la memoria) dictamina la intensidad máxima que va a poder circular por la rama de T3, y, por tanto, la potencia óptica que sera emitida para un valor concreto del control PWM de transmisión PWM_TRX. Esto es así porque la propia resistencia de salida del pin de PWM forma un divisor con la resistencia R_6. Así pues, es necesario dimensionar dicha resistencia adecuadamente para garantizar un rango de valores de intensidad emitida adecuados para el dispositivo emisor FC300T.

Con la placa de prototipado, se hizo un barrido con el PWM y se midió la potencia recibida con un medidor de potencia óptica comercial con un látigo de fibra de plástico corto (de aproximadamente 1m).

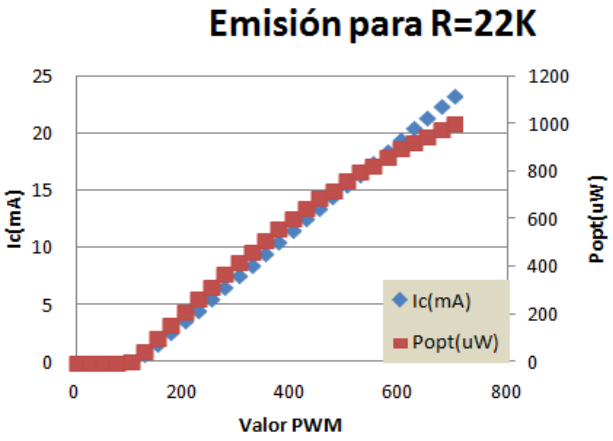
En primer lugar, se midió la influencia de variar la resistencia. Una mayor resistencia limita más la intensidad y filtra mejor, pero un valor demasiado alto no permitiría llegar a los 10 mA que son los recomendados en condiciones normales para el FC300T.



(a) PWM con resistencia de 1K



(b) PWM con resistencia de 10K



(c) PWM con resistencia de 22K

A la vista de estos resultados, se eligió la resistencia de 10K, por dar un buen compromiso entre buena precisión de la intensidad y posibilidad de obtener tanto valores altos como bajos de intensidad. Así también puede entenderse el valor elegido de PWM para tener una intensidad de colector de 10 mA, que es aproximadamente de 230.

2. Arquitectura de Arduino

Arduino es, en realidad, una plataforma de desarrollo abierta, basada en procesadores de 8 bits del fabricante Atmel.

Es interesante la defición de la plataforma según los creadores:

> *"Arduino es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. Se creó para artistas, diseñadores, aficionados y cualquiera interesado en crear entornos u objetos interactivos."*

Por tanto, es ideal para empezar con microcontroladores, pero, dadas sus características de precio, disponibilidad y comunidad, también es adecuada para profundizar y realizar diseños de complejidad media-baja.

En este proyecto, no me he limitado a usar las herramientas básicas y dejar que las librerías, muy útiles por otra parte, hagan el trabajo por mí, sino que he profundizado con detalle en la arquitectura para elaborar el software de manera óptima, controlando el hardware subyacente a él.

El modelo de Arduino elegido ha sido el Arduino Uno R3, cuyo procesador es el ATMEGA328P.

2.1. La placa de desarrollo

El esquema de la placa de desarrollo se adjunta en la página siguiente.

La arquitectura de la placa consta, básicamente, de un microcontrolador ATMEGA328P con todo el pinout de entrada salida fácilmente accesible.

Además, previo a éste tenemos un coprocesador ATMEGA16U2, que implementa la interfaz UART a USB para poder programar el ATMEGA328P.

Junto a éste, hay varios circuitos cuya función es regular el voltaje que llega, seleccionando como fuente de alimentación el USB siempre que la entrada por la alimentación externa no sea superior a los 7V.

No hay implementación JTAG, de manera que no se puede depurar el código intrucción a instrucción. Sin embargo, sí tenemos todo lo necesario para hacer funcionar el micro, un cristal de 16MHz como referencia externa, un switch de reset y cuatro leds, uno para indicar que la placa está alimentada, dos para indicar transmisión/recepción por UART, y otro conectado al pin 9 de libre configuración.

2.2. El ATMEGA328P

En este anexo comentaré los aspectos más destacables del ATMEGA328P, el microcontrolador que ha sido usado para programar este proyecto, así como las características de los periféricos que vamos a usar.

Generalidades El ATMEGA328P es un Microcontrolador de 8-bit con las siguientes características:

- Arquitectura RISC de bajo consumo.
  - Set de 131 instrucciones, la mayoría de ejecución en un sólo ciclo.
  - 32 Registros de propósito general.
  - Multiplicador de 2 ciclos integrado.
  - 5 modos de bajo consumo.
- Segmentación de memoria
  - 32 KB de memoria Flash para programa.
  - 1KB EEPROM
  - 2KB SRAM
- Periféricos internos
  - 3 timers con modos de comparación e interrupciones programables, uno de ellos de 16 bits.
  - Módulo serie USART programable.
  - Puerto serie SPI.
  - Conversor A/D de 10bit y 6 canales.
  - Watchdog programable con oscilador interno.
- Entrada/salida
  - 32 pines, 23 de ellos de entrada salida de propósito general.
  - 1KB EEPROM
  - 2KB SRAM
- Consumos medidos a 1MHz,1.8V,25ºC
  - Modo activo: 0.2 mA
  - Modo apagado: 0.1 uA
  - Modo ahorro: 0.75 uA

Ésta es la configuración de pines:

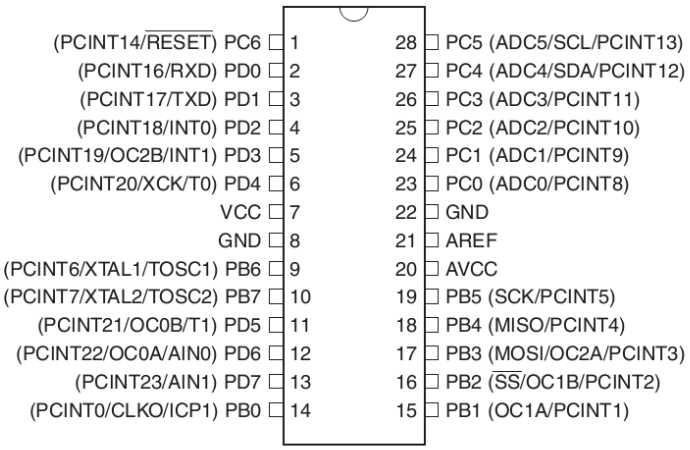


Figura 13: Pinout del ATMEGA328P

Y éstos son los bloques que componen la CPU:

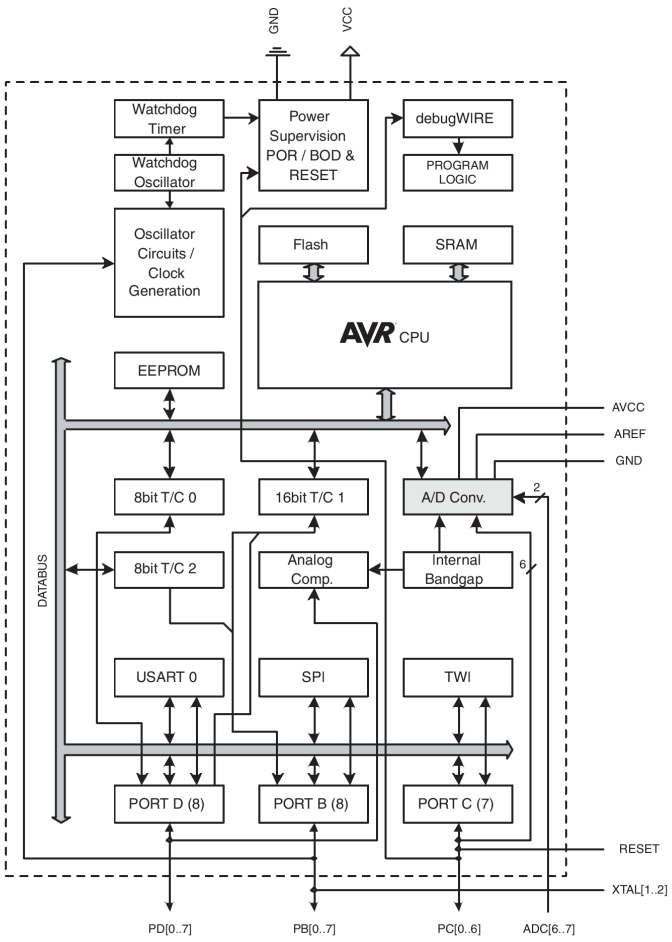


Figura 14: Bloques del ATMEGA328P

2.2.1. Periféricos

A continuación citaré los periféricos internos que voy a utilizar y configurar para este proyecto:

- Timers 1 y 2
- Puerto SPI
- Conversor A/D
- Interfaz UART

Timers En este epígrafe comentaré los modos de funcionamiento de los timers, que implican,a su vez, el comportamiento de las salidas PWM.

Empezaré por el Timer 1, ya que es el de 16 bits y tiene mayor complejidad. Las características de este contador según Atmel son las siguiente:

- Diseño de 16 bits (permite PWM de 16 bits)
- Dos unidades independientes para Output Compare, con doble Buffer.
- Una unidad de Input Capture
- Borrado del Timer tras comparación (conteo automático)
- PWM de correción de fase libre de glitches.
- PWM de período variable.
- Cuatro fuentes independientes de interrupciones (TOV1,OCF1A,OCF1B,ICF)

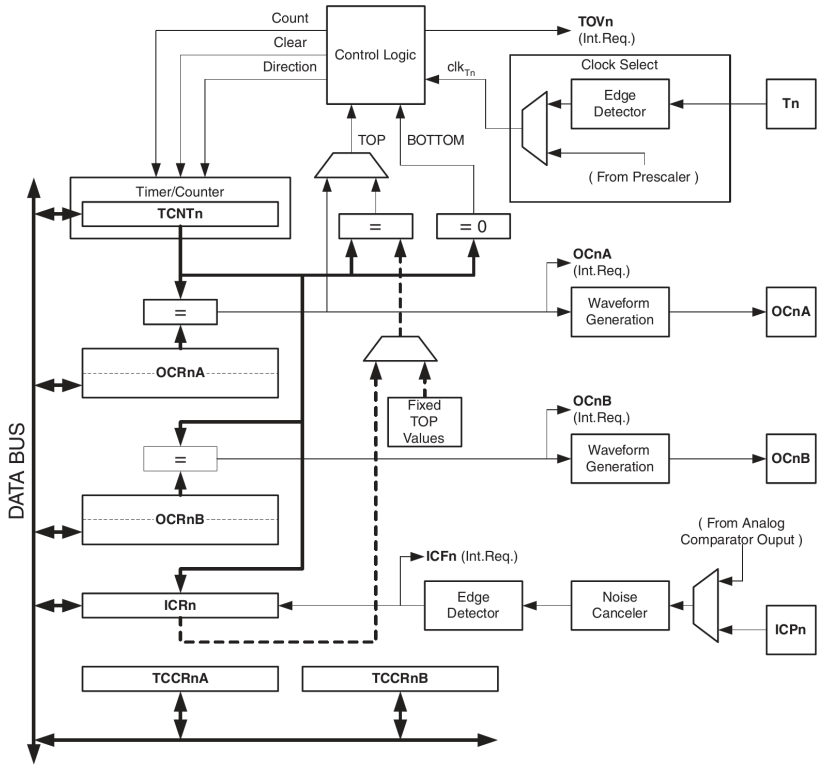


Figura 15: Diagrama del Timer1

Para su configuracion y uso adecuado, dispone de los siguientes registros:

- TCCR1A Timer/Counter Control Register 1A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - Los bits COM1A/COM1B determinan cómo se comporta el output compare para los pines a los que están conectados los canales A y B del timer. Dependen del modo PWM que esté activado, así que hablaré de ellos a tratar el tipo de PWM que utilizaré.
  - Los bits WGM14..WGM10 especifican el modo de trabajo PWM. Hay 14 modos de trabajo disponibles. Se detallará más adelante en la sección modos de trabajo PWM.

- TCCR1B Timer/Counter Control Register 1B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  Los primeros bits se refieren al Input Capture, el cual no utilizaré en este proyecto. Sin embargo los bits CS2..0 son vitales, determinan la fuente de reloj y el divisor a utilizar.

- TCCR1C Timer/Counter Control Register 1C

Bit	7	6	5	4	3	2	1	0	
(0x82)	FOC1A	FOC1B	–	–	–	–	–	–	TCCR1C
Read/Write	R/W	R/W	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

  FOC1A y FOC1B permiten simular que se ha llegado al valor del Output Compare en los canales A y B, forzando la salida al valor correspondiente, en función de COM1A y COM1B. No los utilizaré.

- TCNT1 Timer/Counter 1

Bit	7	6	5	4	3	2	1	0	
(0x85)	TCNT1[15:8]								TCNT1H
(0x84)	TCNT1[7:0]								TCNT1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  Este registro contiene el valor actual del reloj. Debe ser puesto a cero con el reloj parado para evitar glitches.

- OCR1A,OCR1B Output Compare Register 1A/1B

Bit	7	6	5	4	3	2	1	0	
(0x8B)	OCR1B[15:8]								OCR1BH
(0x8A)	OCR1B[7:0]								OCR1BL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Este registro contiene el valor a comparar. Cuando TCNT1 es igual al valor de este registro, se ejecuta la acción correspondiente al modo de funcionamiento.

- ICR1 Input Capture Register 1

Bit	7	6	5	4	3	2	1	0	
(0x87)	ICR1[15:8]								ICR1H
(0x86)	ICR1[7:0]								ICR1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Este registro contiene el valor de TCNT1 para el Input Capture. No se utilizará.

- TIMSK1 Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

En los bits individuales de este registro podemos habilitar o deshabilitar las interrupciones por parte del Input Capture, los dos canales del Output Compare y la interrupción por haber llegado al valor de la cima del contador TOV.

- TIFR1 Timer/Counter1 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Para los mismos casos que anteriormente, al producirse una interrupción se activa el flag, que habrá que desactivar manualmente en la rutina de interrupción si el modo de trabajo no la desactiva automáticamente.

2.2.2. Modos de trabajo PWM

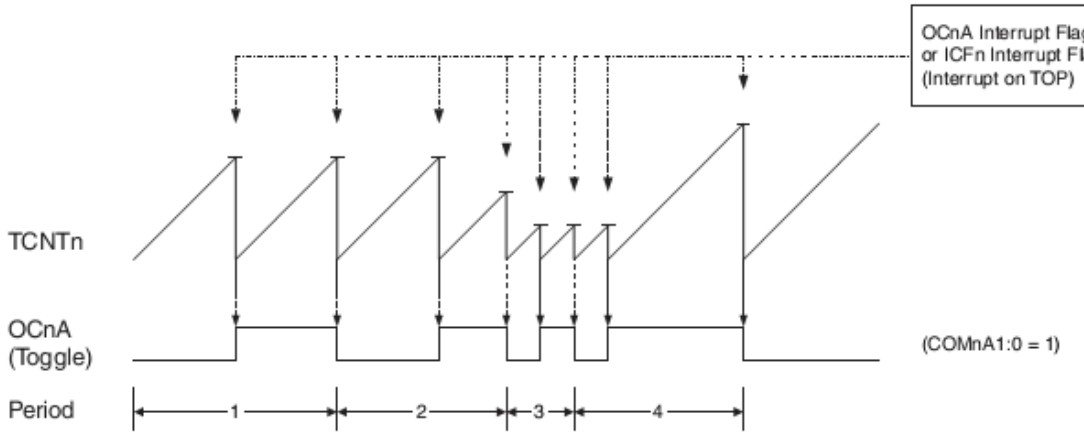
La elección de uno de estos modos determina el valor de los bits WGM13..0 y COM1x1:0, determinando los primeros el modo de funcionamiento y los segundos el comportamiento de la salida en esos modos.

Normal Mode

El modo más simple. El contador siempre se incrementa, y cuenta desde 0 hasta 0xFFFF. Cada vez que se alcanza la cima, se activa el Flag de interrupción TOV1, pudiendo alargar la resolución del timer por software.

Clear Timer on Compare Match

El contador siepmre se incrementa, pero cuenta desde 0 hasta el valor del registro OCR1A O ICR1, activando entonces el flag OCR1A o ICF. Se puede generar una onda cuadrada simé-



trica de frecuencia focna.

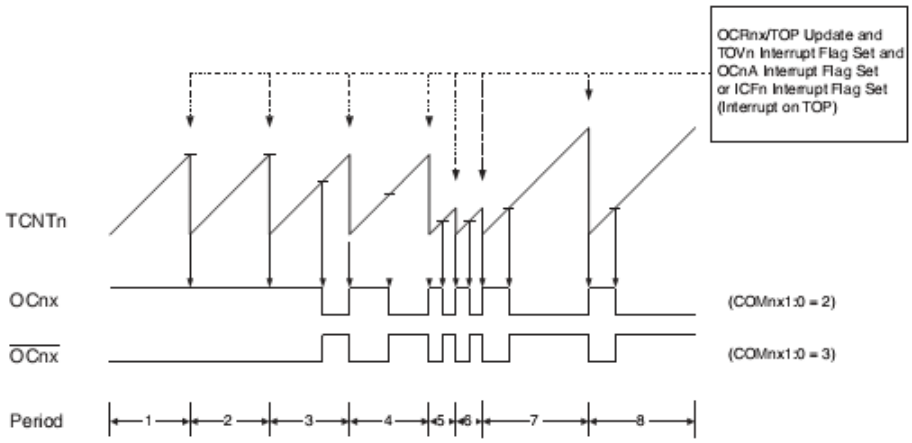
Fast PWM Mode

Permite la generación de PWM de frecuencias altas, debido a que las operaciones se realizan sobre la rampa de subida (mientras el contador se incrementa).

El Timer cuenta desde 0 hasta la cima, que es o el valor marcado por ICR1 o OCR1A, o un valor fijo (PWM de 8,9 o 10 bits). Si se establece el valor manualmente, se tiene un compromiso entre resolución de PWM y la frecuencia de la señal generada.

La salida OCnx cambia al llegar el contador al valor OCR1A. Se activan los flags de las interrupciones correspondientes al llegar el contador a los valores cima,OCnA o ICFn (en función del registro elegido para marcar el valor cima).

A continuación un ejemplo con OCR1A o ICR1A marcando el valor cima, siendo las marcas horizontales cuando el contador llega al valor OCR1A. El valor de la salida es invertido o no dependiendo de los bits COM.

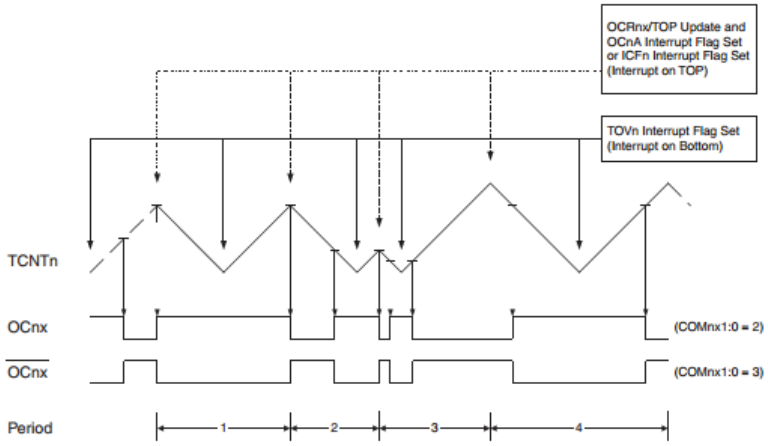


Phase Correct PWM Mode

Éste es el modo más complejo, y que permite mayores opciones de configuración.

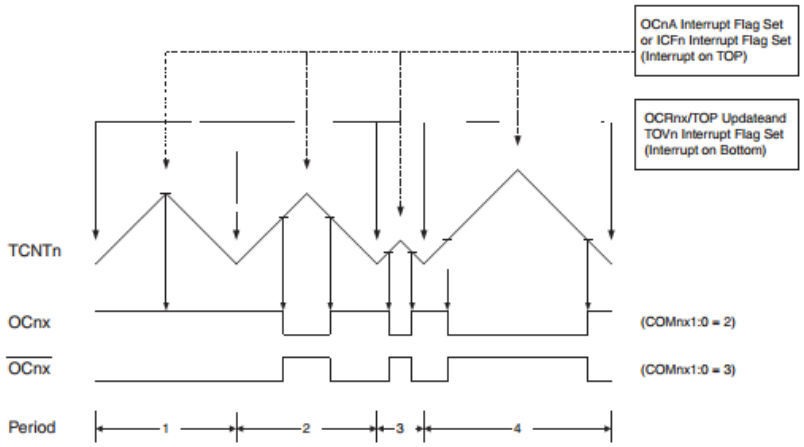
El contador cuenta desde 0 hasta la cima, que es o el valor marcado por ICR1 o OCR1A, o un valor fijo (PWM de 8,9 o 10 bits), y luego vuelve a contar hasta 0. Esta doble operación, tanto conteo hacia arriba como hacia abajo, es útil para control de motores.

El comportamiento es similar al del Fast PWM, pero con actualización del Output compare tanto en la rampa de subida como en la de bajada. En este caso las interrupciones posibles, además de las anteriores, se producen también en las mitades de los ciclos de PWM, por tanto, se ponen a 1 los flags TOV, siendo la cima el momento en el que cambiar el registro OCnX para el Output Compare, y el 0 el momento en el que cambiar el valor cima.



Phase and Frequency Correct PWM Mode

El funcionamiento es similar al del PWM con fase corregida, pero el valor de la cima y el del registro OCR1x se actualizan al llegar al valor mínimo. De esta manera se mantiene la simetría de las ondas, aunque perdiendo algo de posibilidad de configuración (una vez por ciclo en lugar de dos).



2.2.3. SPI

El SPI, o Interfaz Serie con Periféricos, es un módulo del ATMEGA328P que permite transferencia de datos síncrona entre la CPU y cualquier periférico compatible con este interfaz, incluyendo otros ATMEGA328P.

En nuestro caso, nos interesa la configuración del SPI siendo el ATMEGA328P el Master y nuestro periférico (que será el MCP6S21, pero podría ser cualquier otro) como esclavo.

Éste es el diagrama del SPI:

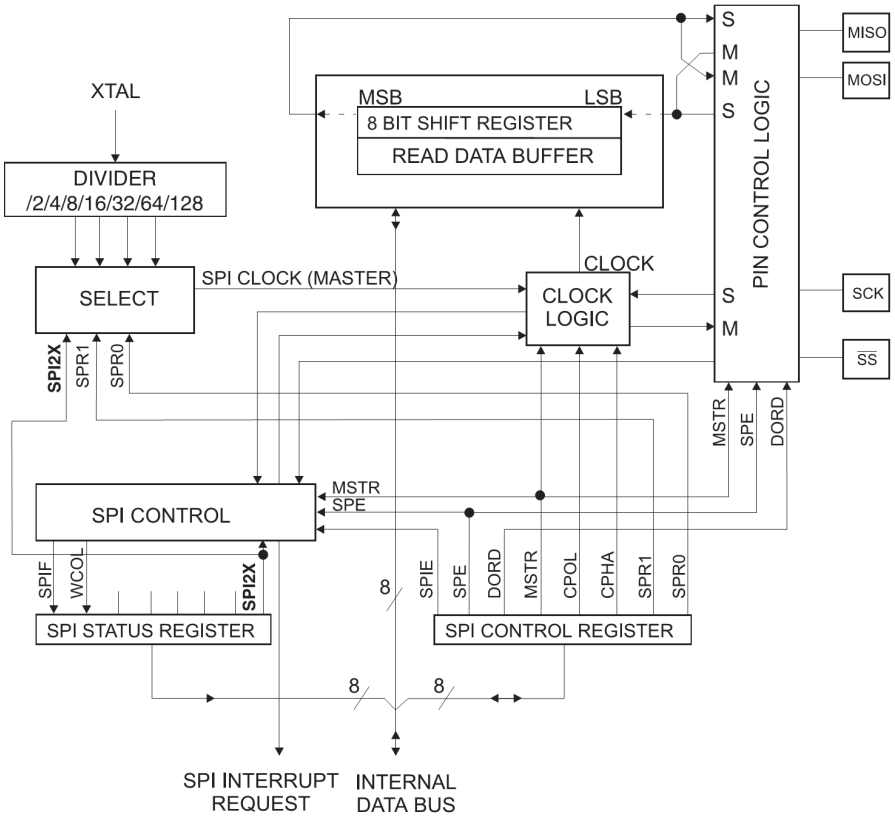


Figura 16: Diagrama del SPI

Donde tenemos que con el exterior los pines utilizados son:

MISO (Master Input Slave Output) Entrada del maestro, salida del esclavo. No la utilizaremos porque no necesitamos recibir datos de los periféricos.

MOSI (Master Output Slave Input) Salida del maestro, entrada del esclavo. Por este pin enviamos los datos al periférico.

SCK (Serial Clock) Reloj para la sincronización de los datos.

SS (Slave Select) Este pin se utiliza para controlar el inicio y el final de las transferencias.

La interconexión maestro-esclavo queda simplificada en esta figura:

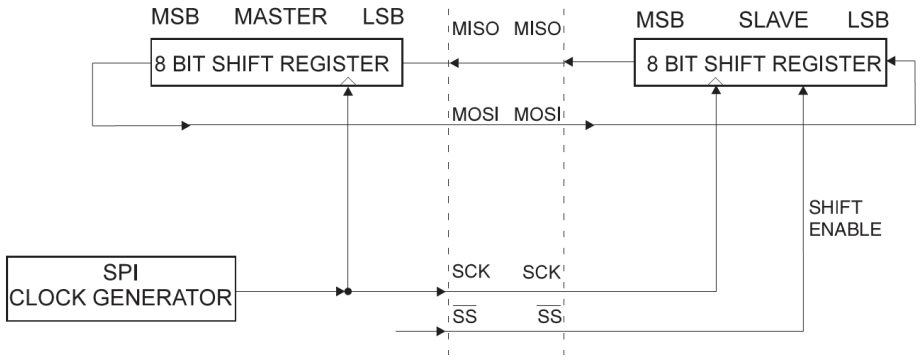


Figura 17: Interconexión del SPI en una configuración maestro-esclavo

Como puede verse, los registros son de 8 bits tanto para transmisión como para recepción, en el caso de transferencias de 16 bits se requieren dos escrituras en el registro de transmisión, o dos lecturas en el de recepción.

Además, hay varios modos de configuración del reloj respecto a la transmisión y recepción de los datos:

SPI Mode	Conditions	Leading Edge	Trailing eDge
0	CPOL=0, CPHA=0	Sample (Rising)	Setup (Falling)
1	CPOL=0, CPHA=1	Setup (Rising)	Sample (Falling)
2	CPOL=1, CPHA=0	Sample (Falling)	Setup (Rising)
3	CPOL=1, CPHA=1	Setup (Falling)	Sample (Rising)

Figura 18: Modos de configuración del reloj SPI

Debemos conocer qué modos son compatibles con nuestro periférico y acordes a nuestras necesidades. En nuestro caso la configuración es en modo 0, con lo cual el cronograma es el siguiente:

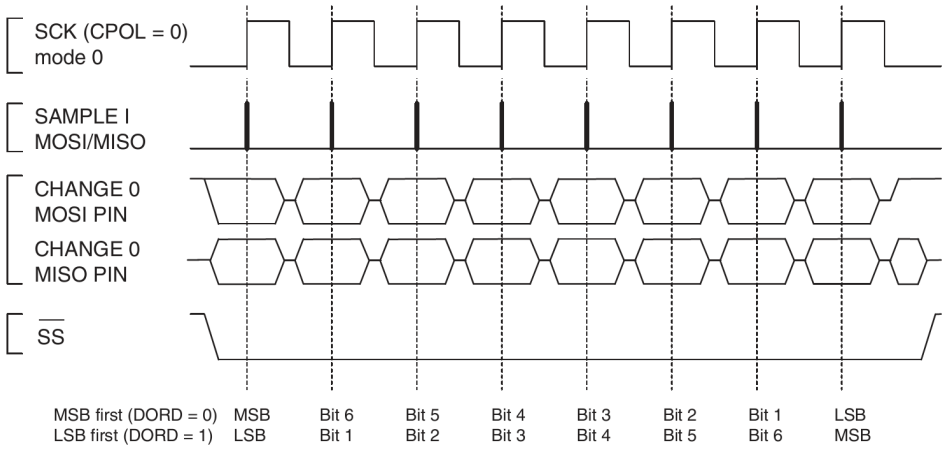


Figura 19: Cronograma para configuración en modo 0

Asimismo, es importante establecer el orden de los bits, que en nuestro caso es el bit más significativo primero, pues así lo requiere nuestro periférico.

A continuación, los registros de configuración del puerto serie en los que establecer estas opciones:

SPCR (SPI Control Register)

Bit	7	6	5	4	3	2	1	0	
0x2C (0x4C)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

SPIE (SPI Interrupt Enable) Sirve para activar las interrupciones del puerto serie.

SPE (SPI Enable) Para activar el puerto Serie. Debe estar a 1 para poder usar el SPI.

DORD (Data Order) 1 El bit mas significativo de la palabra a transmitir es el primero. 0 El bit menos significativo primero.

MSTR Se debe poner a uno para que que el ATMEGA328P haga de maestro.

CPOL,CPHA Establecen el modo de configuración, para modo 0, CPOL y CPHA son 0, lo cual implica que la transmisión se inicia con el flanco de bajada, y la recepción con el flanco de subida, siendo el reposo en nivel bajo.

SPR1 y SPR0 (SPI Clock Rate) Para seleccionar el divisor del reloj para SCK, desde division por 2 (8 MHz) a 128 (125 KHz). Esto permite adecuar la velocidad del bus a la del periférico.

SPSR (SPI Status Register)

Bit	7	6	5	4	3	2	1	0	
0x2D (0x4D)	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

SPIF (SPI Interrupt Flag) Este bit se activa al terminar una transferencia. Si están activadas las interrupciones del puerto serie se genera una interrupción y se pone este bit a cero.

WCOL (Write COLlition Flag) Se activa si el registro de datos SPDR se escribe durante una transferencia de datos. Para reiniciar las transferencias hay que leer este registro con WCOL=1 y acceder de nuevo al registro de datos SPDR.

SPI2X (Double SPI Speed Bit) Si se activa la frecuencia de SCK se dobla en modo maestro, de manera que la velocidad efectiva de SCK se establece junto con los bits SPR del registro de control.

SPDR (SPI Data Register)

Bit	7	6	5	4	3	2	1	0	
0x2E (0x4E)	MSB							LSB	SPDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	X	X	X	X	X	X	X	X	Undefined

Este registro actúa como registro intermedio tanto para las lecturas como las escrituras por el puerto serie. Escribir en él inicia la transmisión de los datos, leer de él permite el acceso al registro interno de recepción de datos.

2.2.4. Conversor A/D

Las características generales del conversor A/D, que funciona por aproximaciones sucesivas (SAR) son:

- Resolución de 10 bits.
- No linearidad de 0.5 LSB, con precisión de 2 LSB.
- Tiempo de conversión entre 13 y 260 us.
- 6 entradas multiplexadas, más una con sensor de temperatura.
- Ajuste opcional de los bits más significativos del resultado.
- Rango de la entrada entre 0 y Vcc.
- Referencia seleccionable de 1.1V, o externa.
- Modo de conversión simple o ejecución libre.
- Interrupción tras conversión completa.
- Modo de reposo para el Cancelador de Ruido.

En la siguiente página puede verse el diagrama del conversor.

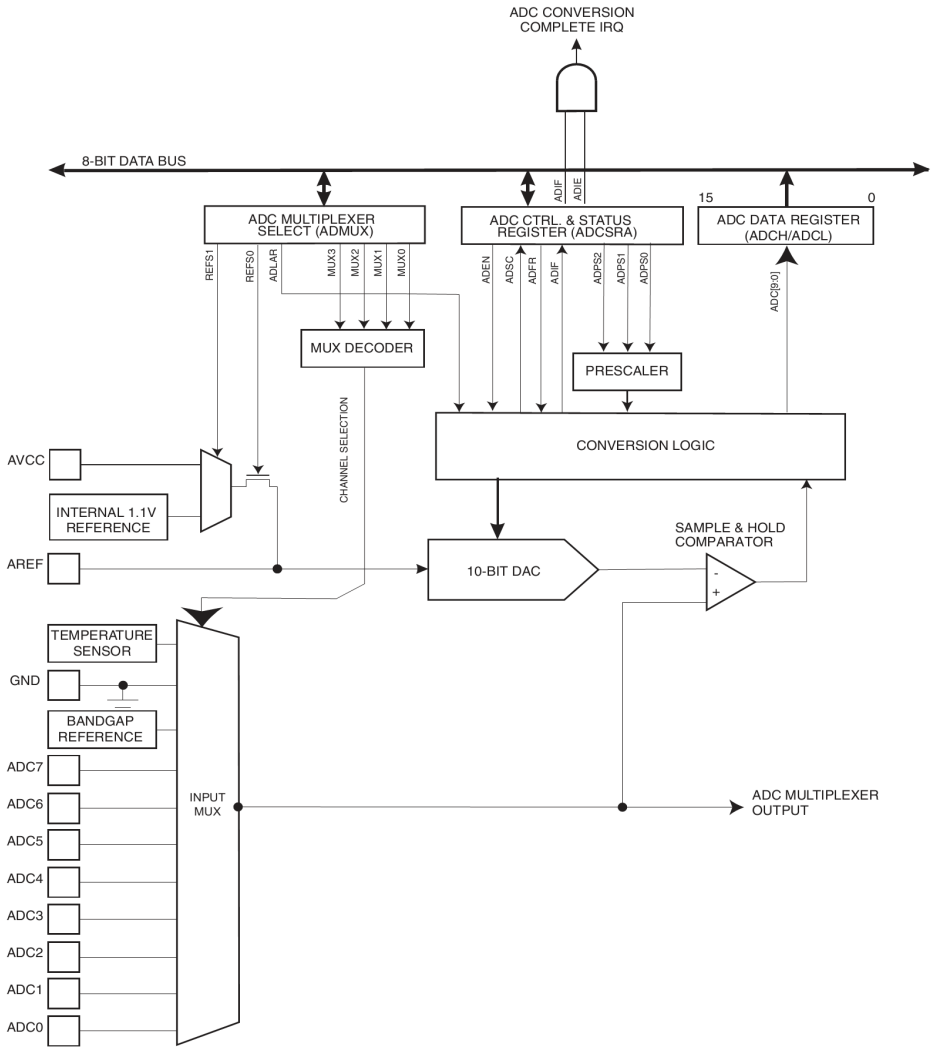


Figura 20: Diagrama del Conversor A/D

El funcionamiento del conversor es el siguiente:

- Se debe desactivar el bit PRADC que mantiene el ADC apagado por defecto para ahorrar potencia.
- El resultado del conversor estará entre GND y el voltaje en el pin AREF (puede tener varios orígenes) menos 1 LSB.
- El canal de entrada se elige escribiendo en los bits MUX en el registro ADMUX.
- Se activa el bit ADEN para activar el conversor.
- Una vez escrito un 0 y un 1 en el bit ADCSC, la conversión se inicia y se nos entregará en los registros ADCH y ADCL, con una interrupción además en el caso de que se hayan activado, y en todo caso ADCSC se queda en valor 0 una vez acabada la conversión.
- Una vez leído ADCH, se libera el acceso para el conversor y se puede iniciar una nueva.

Modo de auto inicio de conversión

Si bien se pueden iniciar las conversiones manualmente con ADSC, también pueden iniciarse automáticamente desde distintas fuentes, establecidas en los bits ADTS (ver tabla). La conversión se inicia cuando la fuente elegida tiene un flanco de subida, y si hay varios mientras se efectúa la conversión, éstos se ignoran.

Un caso peculiar es el de utilizar como fuente ADIF, el flag de interrupción, que hace que el conversor inicie constantemente conversiones.

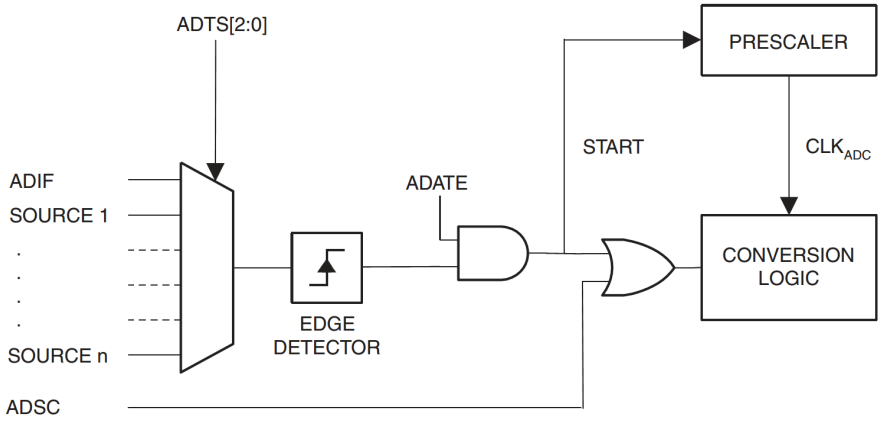


Figura 21: Función de Autoinicio de conversión

Pre-escalado y velocidad del conversor

Por defecto, para la máxima resolución de 10 bits, se requiere una velocidad de reloj de entre 50kHz y 200kHz, por tanto, se necesita una unidad para reducir el reloj del sistema a esta velocidad.

Esta unidad se activa con ADEN activo, y su tiempo de conteo depende de los bits ADPS. Una conversión estándar, con el ADC activado, lleva 13 ciclos de reloj del ADC, mientras que la primera conversión tras activar ADEN conllva 25 ciclos.

Se debe tener la precaución de que, para que la conversión sea correcta, el voltaje haya tenido un tiepmo para estabilizarse, y que haya pasado el tiempo mínimo de hold para la entrada analógica, que es de 1.5 ciclos (13.5 ciclos la primera conversión).

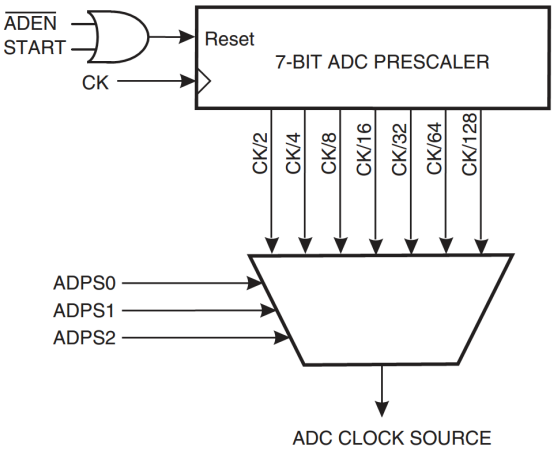


Figura 22: Diagrama del bloque de pre-escalado

A continuación, los tiempos para la conversión simple, que es la que utilizaremos. Por un lado tenemos los tiempos de la primera conversión, y por otro el resto. Para asegurarnos un retraso constante y, por tanto, un comportamiento uniforme desde la primera medida, el modo de funcionamiento será de medida única, con disparo de interrupción cuando la conversión sea completa.

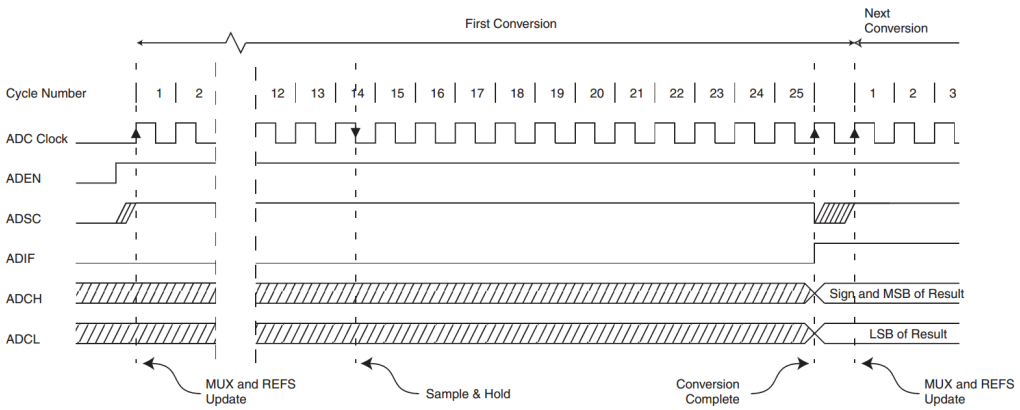


Figura 23: Cronograma para la primera conversión

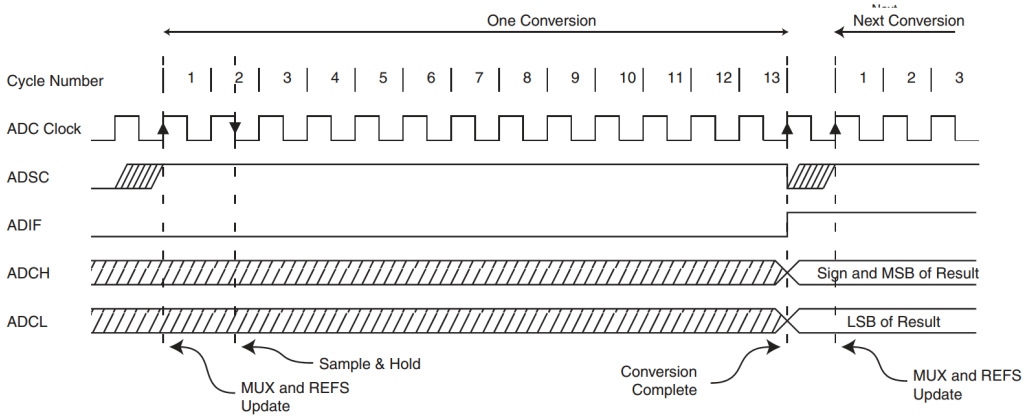


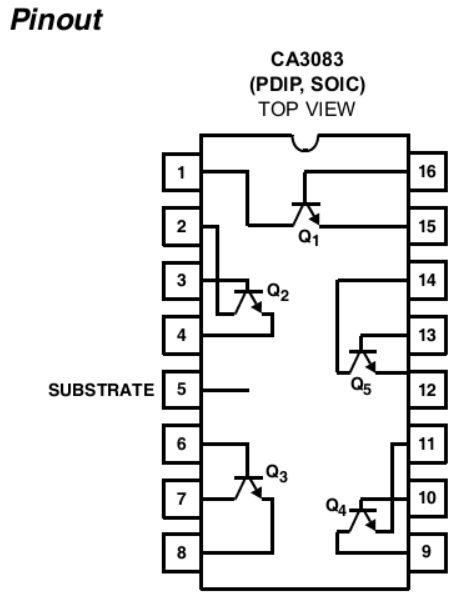
Figura 24: Cronograma para el resto de conversiones

Veamos todos los registros en detalle.

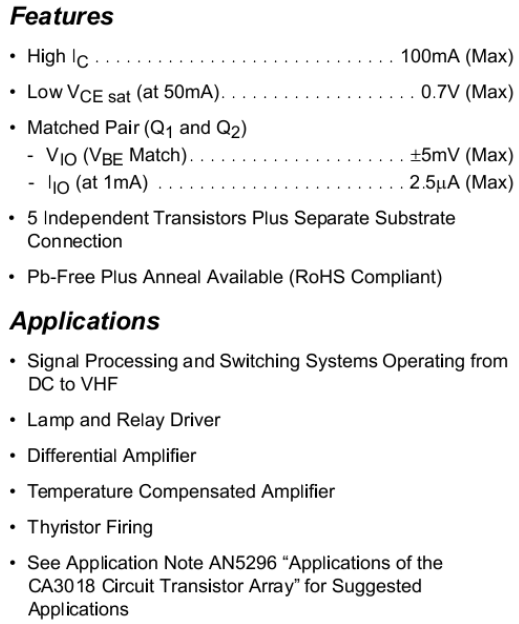
3. Dispositivos externos

3.1. Array de transistores C3083

El C3083 es un encapsulado que contiene cinco transistores NPN cuya característica mas importante es la de proporcionar hasta 100 mA de corriente de colector. Dado que ese también es el límite maximo de corriente que soportara nuestro LED emisor, como veremos en sus hojas de caraterísticas, y la arquitectura de nuestro emisor comprende 3 transistores NPN, ésta es una elección adecuada y ademas de bajo coste. A continuación un resumen de las especificaciones, el Pinout y sus características completas.



(a) Configuración de los pines del C3083



(b) Resumen de especificaciones del C3083

Absolute Maximum Ratings

The following ratings apply for each transistor in the device:
Collector-to-Emitter Voltage, V_{CEO} 15V
Collector-to-Base Voltage, V_{CBO} 20V
Collector-to-Substrate Voltage, V_{CIO} (Note 1) 20V
Emitter-to-Base Voltage, V_{EBO} 5V
Collector Current (I_C) 100mA
Base Current (I_B) 20mA

Thermal Information

Thermal Resistance (Typical, Note 2)	θ_{JA} (°C/W)	θ_{JC} (°C/W)
PDIP Package	135	N/A
SOIC Package	200	N/A

Maximum Power Dissipation (Any One Transistor) 500mW
Maximum Junction Temperature (Plastic Package) . . . . . 150°C
Maximum Storage Temperature Range . . . . . -65°C to 150°C
Maximum Lead Temperature (Soldering 10s) . . . . . 300°C
(SOIC - Lead Tips Only)

Operating Conditions

Temperature Range . . . . . -55°C to 125°C

CAUTION: Stresses above those listed in "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress only rating and operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

NOTES:

1. The collector of each transistor of the CA3083 is isolated from the substrate by an integral diode. The substrate must be connected to a voltage which is more negative than any collector voltage in order to maintain isolation between transistors and provide normal transistor action. To avoid undesired coupling between transistors, the substrate Terminal (5) should be maintained at either DC or signal (AC) ground. A suitable bypass capacitor can be used to establish a signal ground.
2. θ_{JA} is measured with the component mounted on an evaluation PC board in free air.

Electrical Specifications For Equipment Design, T_A = 25°C

PARAMETER	SYMBOL	TEST CONDITIONS		MIN	TYP	MAX	UNITS
FOR EACH TRANSISTOR							
Collector-to-Base Breakdown Voltage	$V_{(BR)CBO}$	I_C = 100μA, I_E = 0		20	60	-	V
Collector-to-Emitter Breakdown Voltage	$V_{(BR)CEO}$	I_C = 1mA, I_B = 0		15	24	-	V
Collector-to-Substrate Breakdown Voltage	$V_{(BR)CIO}$	I_{CI} = 100μA, I_B = 0, I_E = 0		20	60	-	V
Emitter-to-Base Breakdown Voltage	$V_{(BR)EBO}$	I_E = 500μA, I_C = 0		5	6.9	-	V
Collector-Cutoff-Current	I_{CEO}	V_{CE} = 10V, I_B = 0		-	-	10	μA
Collector-Cutoff-Current	I_{CBO}	V_{CB} = 10V, I_E = 0		-	-	1	μA
DC Forward-Current Transfer Ratio (Note 3) (Figure 1)	h_{FE}	V_{CE} = 3V	I_C = 10mA	40	76	-	
			I_C = 50mA	40	75	-	
Base-to-Emitter Voltage (Figure 2)	V_{BE}	V_{CE} = 3V, I_C = 10mA		0.65	0.74	0.85	V
Collector-to-Emitter Saturation Voltage (Figures 3, 4)	$V_{CE\ SAT}$	I_C = 50mA, I_B = 5mA		-	0.40	0.70	V
Gain Bandwidth Product	f_T	V_{CE} = 3V, I_C = 10mA		-	450	-	MHz
FOR TRANSISTORS Q_1 AND Q_2 (As a Differential Amplifier)							
Absolute Input Offset Voltage (Figure 6)	$\lvert V_{IO}\rvert$	V_{CE} = 3V, I_C = 1mA		-	1.2	5	mV
Absolute Input Offset Current (Figure 7)	$\lvert I_{IO}\rvert$	V_{CE} = 3V, I_C = 1mA		-	0.7	2.5	μA

NOTE:

3. Actual forcing current is via the emitter for this test.

Figura 25: Especificaciones Eléctricas del C3083

3.2. Transceptor óptico FC300T

SPECIFICATIONS

Table 1
TRANSMITTER ABSOLUTE MAXIMUM RATINGS

These are the absolute maximum ratings at or beyond which the FOT can be expected to be damaged.

Parameter	Symbol	Minimum	Maximum	Unit
Storage Temperature	T_{stg}	-40	+100	°C
Operating Temperature	T_{op}	-20	+85	°C
Soldering Temperature[1]	T_{sld}		+260	°C
Continuous Forward Current	I_f		100	mA
Reverse Voltage	V_R		5	V

Notes:
1. 260°C, 5s 3 times, at least 2.2 mm away from lead root.

Table 2
TRANSMITTER OPTICAL AND ELECTRICAL CHARACTERISTICS

Unless otherwise stated, T_A= +25°C.

Parameter	Symbol	Minimum	Typical	Maximum	Unit	Test Condition
Peak Wavelength	λ	640	660	670	nm	10 mA, -20 to 85°C
Spectral FWHM	Δλ	15		30	nm	10 mA, -20 to 85°C
Peak Wavelength Temperature Coefficient	dλ/dT		-0.13		nm/°C	
Optical Power[1]	P_{OP}	-10		0	dBm	10 mA, -20 to 85°C
Forward Voltage	V_f	1.7		2.3	V	10 mA, -20 to 85°C
Cutoff Frequency 10 mA Bias	f_c		80		MHz	-3dB Optical Power
Cutoff Frequency 20 mA Bias	f_c		100		MHz	-3dB Optical Power
Capacitance	C_o		5		pFV	$_f$=0V, f=1MHz

Notes:
1. Optical power coupled into 1 mm diameter plastic fiber.

SPECIFICATIONS (Continued)

Table 3
RECEIVER ABSOLUTE MAXIMUM RATINGS

These are the absolute maximum ratings at or beyond which the FOT can be expected to be damaged.

Parameter	Symbol	Minimum	Maximum	Unit
Storage Temperature	T_{stg}	-40	+100	°C
Operating Temperature	T_{op}	-20	+85	°C
Electrical Power Dissipation	P_{tot}		100	mW
Reverse Voltage	V_R		25	V
Soldering Temperature[1]	T_{sld}		+260	°C

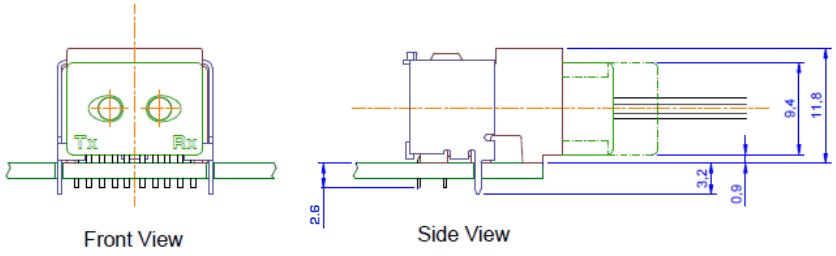
Notes:
1. 260°C, 5s 3 times, at least 2.2 mm away from lead root.

Table 4
RECEIVER OPTICAL AND ELECTRICAL CHARACTERISTICS

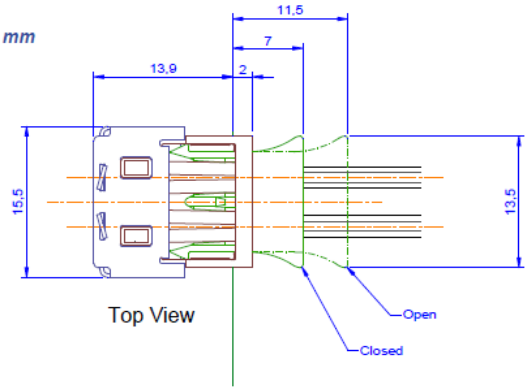
Unless otherwise stated, T_A= +25°C.

Parameter	Symbol	Minimum	Typical	Maximum	Unit	Test Condition
Responsivity, λ=660 nm	R		0.3		A/W	
Dark Current	I_R		0.2	1.0	nA	10 mA, -20 to 85°C
Rise Time (20% to 80%)	t_r		1		ns	
Fall Time (80% to 20%)	t_f		1		ns	
Dark Noise Density			10		fA/√Hz	
Capacitance	C_o		3		pF	Bias=8v, f=1MHz

MECHANICAL DATA



Mechanical Tolerances: +/- 0.2 mm



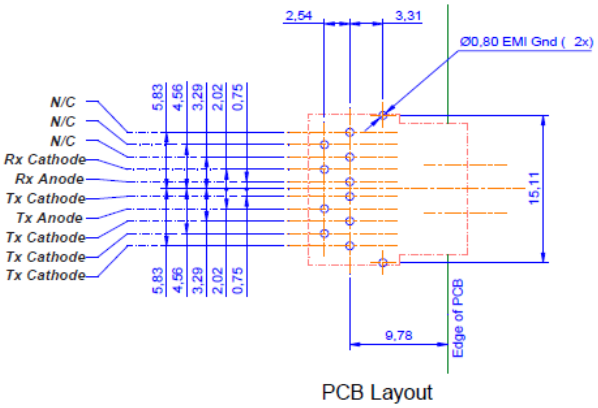


FIGURE 5
PCB layout (top view, looking down) for the Firecomms transceiver FC300T

Firecomms

3.3. Amplificador bajo ruido AD822

ANALOG DEVICES

Single-Supply, Rail-to-Rail Low Power FET-Input Op Amp

AD822

FEATURES

True single-supply operation
- **Output swings rail-to-rail**
- **Input voltage range extends below ground**
- **Single-supply capability from 5 V to 30 V**
- **Dual-supply capability from ±2.5 V to ±15 V**

High load drive
- **Capacitive load drive of 350 pF, G = +1**
- **Minimum output current of 15 mA**

Excellent ac performance for low power
- **800 µA maximum quiescent current per amplifier**
- **Unity-gain bandwidth: 1.8 MHz**
- **Slew rate of 3 V/µs**

Good dc performance
- **800 µV maximum input offset voltage**
- **2 µV/°C typical offset voltage drift**
- **25 pA maximum input bias current**

Low noise
- **13 nV/√Hz @ 10 kHz**
- **No phase inversion**

APPLICATIONS

Battery-powered precision instrumentation
Photodiode preamps
Active filters
12-bit to 14-bit data acquisition systems
Medical instrumentation
Low power references and regulators

CONNECTION DIAGRAM

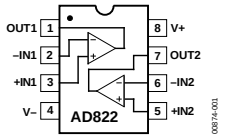


Figure 1. 8-Lead PDIP (N Suffix); 8-Lead MSOP (RM Suffix); and 8-Lead SOIC_N (R Suffix)

GENERAL DESCRIPTION

The AD822 is a dual precision, low power FET input op amp that can operate from a single supply of 5 V to 30 V or dual supplies of ±2.5 V to ±15 V. It has true single-supply capability with an input voltage range extending below the negative rail, allowing the AD822 to accommodate input signals below ground in the single-supply mode. Output voltage swing extends to within 10 mV of each rail, providing the maximum output dynamic range.

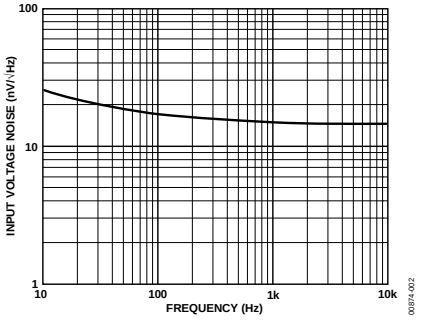


Figure 2. Input Voltage Noise vs. Frequency

Offset voltage of 800 µV maximum, offset voltage drift of 2 µV/°C, input bias currents below 25 pA, and low input voltage noise provide dc precision with source impedances up to a gigaohm. The 1.8 MHz unity-gain bandwidth, –93 dB THD at 10 kHz, and 3 V/µs slew rate are provided with a low supply current of 800 µA per amplifier.

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700 www.analog.com
Fax: 781.461.3113

The AD822 drives up to 350 pF of direct capacitive load as a follower and provides a minimum output current of 15 mA. This allows the amplifier to handle a wide range of load conditions. Its combination of ac and dc performance, plus the outstanding load drive capability, results in an exceptionally versatile amplifier for the single-supply user.

The AD822 is available in two performance grades. The A grade and B grade are rated over the industrial temperature range of −40°C to +85°C.

The AD822 is offered in three varieties of 8-lead packages: PDIP, MSOP, and SOIC_N.

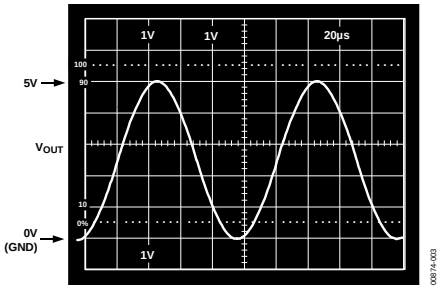


Figure 3. Gain-of-2 Amplifier; V_S = 5 V, 0 V, V_{IN} = 2.5 V Sine Centered at 1.25 V, R_L = 100 Ω

SPECIFICATIONS

V_S = 0 V, 5 V @ T_A = 25°C, V_{CM} = 0 V, V_{OUT} = 0.2 V, unless otherwise noted.

Table 1.

		A Grade			B Grade			
Parameter	**Conditions**	**Min**	**Typ**	**Max**	**Min**	**Typ**	**Max**	**Unit**
DC PERFORMANCE								
Initial Offset			0.1	0.8		0.1	0.4	mV
Maximum Offset Over Temperature			0.5	1.2		0.5	0.9	mV
Offset Drift			2			2		µV/°C
Input Bias Current	V_{CM} = 0 V to 4 V		2	25		2	10	pA
At T_{MAX}			0.5	5		0.5	2.5	nA
Input Offset Current			2	20		2	10	pA
At T_{MAX}			0.5			0.5		nA
Open-Loop Gain	V_{OUT} = 0.2 V to 4 V							
	R_L = 100 kΩ	500	1000		500	1000		V/mV
T_{MIN} to T_{MAX}		400			400			V/mV
	R_L = 10 kΩ	80	150		80	150		V/mV
T_{MIN} to T_{MAX}		80			80			V/mV
	R_L = 1 kΩ	15	30		15	30		V/mV
T_{MIN} to T_{MAX}		10			10			V/mV
NOISE/HARMONIC PERFORMANCE								
Input Voltage Noise								
f = 0.1 Hz to 10 Hz			2			2		µV p-p
f = 10 Hz			25			25		nV/√Hz
f = 100 Hz			21			21		nV/√Hz
f = 1 kHz			16			16		nV/√Hz
f = 10 kHz			13			13		nV/√Hz
Input Current Noise								
f = 0.1 Hz to 10 Hz			18			18		fA p-p
f = 1 kHz			0.8			0.8		fA/√Hz
Harmonic Distortion	R_L = 10 kΩ to 2.5 V							
f = 10 kHz	V_{OUT} = 0.25 V to 4.75 V		−93			−93		dB
DYNAMIC PERFORMANCE								
Unity-Gain Frequency			1.8			1.8		MHz
Full Power Response	V_{OUT} p-p = 4.5 V		210			210		kHz
Slew Rate			3			3		V/µs
Settling Time								
To 0.1%	V_{OUT} = 0.2 V to 4.5 V		1.4			1.4		µs
To 0.01%	V_{OUT} = 0.2 V to 4.5 V		1.8			1.8		µs
MATCHING CHARACTERISTICS								
Initial Offset				1.0			0.5	mV
Maximum Offset Over Temperature				1.6			1.3	mV
Offset Drift			3			3		µV/°C
Input Bias Current				20			10	pA
Crosstalk @ f = 1 kHz	R_L = 5 kΩ		−130			−130		dB
Crosstalk @ f = 100 kHz	R_L = 5 kΩ		−93			−93		dB
INPUT CHARACTERISTICS								
Input Voltage Range[1], T_{MIN} to T_{MAX}		−0.2		+4	−0.2		+4	V
Common-Mode Rejection Ratio (CMRR)	V_{CM} = 0 V to 2 V	66	80		69	80		dB
T_{MIN} to T_{MAX}	V_{CM} = 0 V to 2 V	66			66			dB

Parameter	Conditions	A Grade Min	A Grade Typ	A Grade Max	B Grade Min	B Grade Typ	B Grade Max	Unit
Input Impedance								
Differential			10^{13}||0.5			10^{13}||0.5		Ω||pF
Common Mode			10^{13}||2.8			10^{13}||2.8		Ω||pF
OUTPUT CHARACTERISTICS								
Output Saturation Voltage[2]								
$V_{OL} - V_{EE}$	I_{SINK} = 20 µA		5	7		5	7	mV
T_{MIN} to T_{MAX}				10			10	mV
$V_{CC} - V_{OH}$	I_{SOURCE} = 20 µA		10	14		10	14	mV
T_{MIN} to T_{MAX}				20			20	mV
$V_{OL} - V_{EE}$	I_{SINK} = 2 mA		40	55		40	55	mV
T_{MIN} to T_{MAX}				80			80	mV
$V_{CC} - V_{OH}$	I_{SOURCE} = 2 mA		80	110		80	110	mV
T_{MIN} to T_{MAX}				160			160	mV
$V_{OL} - V_{EE}$	I_{SINK} = 15 mA		300	500		300	500	mV
T_{MIN} to T_{MAX}				1000			1000	mV
$V_{CC} - V_{OH}$	I_{SOURCE} = 15 mA		800	1500		800	1500	mV
T_{MIN} to T_{MAX}				1900			1900	mV
Operating Output Current		15			15			mA
T_{MIN} to T_{MAX}		12			12			mA
Capacitive Load Drive			350			350		pF
POWER SUPPLY								
Quiescent Current, T_{MIN} to T_{MAX}			1.24	1.6		1.24	1.6	mA
Power Supply Rejection	V+ = 5 V to 15 V	66	80		70	80		dB
T_{MIN} to T_{MAX}		66			70			dB

[1] This is a functional specification. Amplifier bandwidth decreases when the input common-mode voltage is driven in the range (V+ − 1 V) to V+. Common-mode error voltage is typically less than 5 mV with the common-mode voltage set at 1 V below the positive supply.
[2] $V_{OL} - V_{EE}$ is defined as the difference between the lowest possible output voltage (V_{OL}) and the negative voltage supply rail (V_{EE}). $V_{CC} - V_{OH}$ is defined as the difference between the highest possible output voltage (V_{OH}) and the positive supply voltage (V_{CC}).

3.4. Amplificador de ganancia programable MCP6S21

MICROCHIP

MCP6S21/2/6/8

Single-Ended, Rail-to-Rail I/O, Low Gain PGA

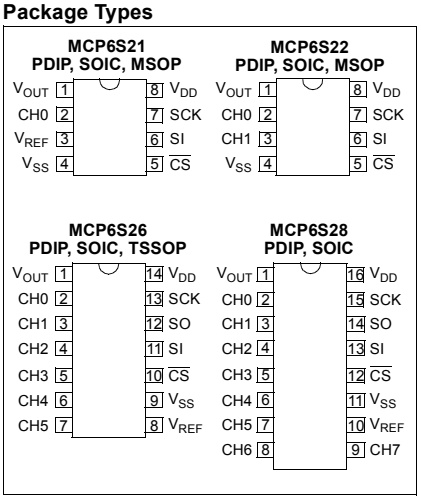
Features

- Multiplexed Inputs: 1, 2, 6 or 8 channels
- 8 Gain Selections:
  - +1, +2, +4, +5, +8, +10, +16 or +32 V/V
- Serial Peripheral Interface (SPI™)
- Rail-to-Rail Input and Output
- Low Gain Error: ±1% (max)
- Low Offset: ±275 µV (max)
- High Bandwidth: 2 to 12 MHz (typ)
- Low Noise: 10 nV/√Hz @ 10 kHz (typ)
- Low Supply Current: 1.0 mA (typ)
- Single Supply: 2.5V to 5.5V

Typical Applications

- A/D Converter Driver
- Multiplexed Analog Applications
- Data Acquisition
- Industrial Instrumentation
- Test Equipment
- Medical Instrumentation

Package Types

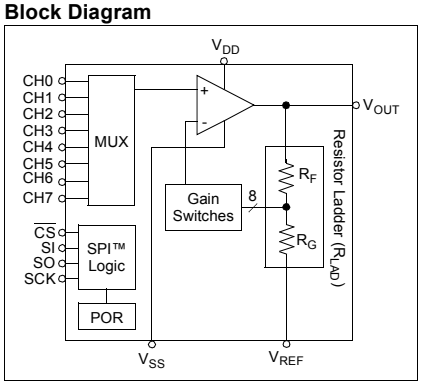


Description

The Microchip Technology Inc. MCP6S21/2/6/8 are analog Programmable Gain Amplifiers (PGA). They can be configured for gains from +1 V/V to +32 V/V and the input multiplexer can select one of up to eight channels through an SPI port. The serial interface can also put the PGA into shutdown to conserve power. These PGAs are optimized for high speed, low offset voltage and single-supply operation with rail-to-rail input and output capability. These specifications support single supply applications needing flexible performance or multiple inputs.

The one channel MCP6S21 and the two channel MCP6S22 are available in 8-pin PDIP, SOIC and MSOP packages. The six channel MCP6S26 is available in 14-pin PDIP, SOIC and TSSOP packages. The eight channel MCP6S28 is available in 16-pin PDIP and SOIC packages. All parts are fully specified from -40°C to +85°C.

Block Diagram



MCP6S21/2/6/8

1.0 ELECTRICAL CHARACTERISTICS

Absolute Maximum Ratings †

V_{DD} - V_{SS} 7.0V
All inputs and outputs..... V_{SS} - 0.3V to V_{DD} +0.3V
Difference Input voltage ..... $|V_{DD} - V_{SS}|$
Output Short Circuit Current.....continuous
Current at Input Pin .....±2 mA
Current at Output and Supply Pins ..... ±30 mA
Storage temperature .....-65°C to +150°C
Junction temperature .....+150°C
ESD protection on all pins (HBM;MM)..... ≥ 2 kV; 200V

† Notice: Stresses above those listed under "Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operation listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

PIN FUNCTION TABLE

Name	Function
V_{OUT}	Analog Output
CH0-CH7	Analog Inputs
V_{SS}	Negative Power Supply
V_{DD}	Positive Power Supply
SCK	SPI Clock Input
SI	SPI Serial Data Input
SO	SPI Serial Data Output
$\overline{CS}$	SPI Chip Select
V_{REF}	External Reference Pin

DC CHARACTERISTICS

Electrical Specifications: Unless otherwise indicated, T_A = +25°C, V_{DD} = +2.5V to +5.5V, V_{SS} = GND, V_{REF} = V_{SS}, G = +1 V/V, Input = CH0 = (0.3V)/G, CH1 to CH7 = 0.3V, R_L = 10 kΩ to V_{DD}/2, SI and SCK are tied low and $\overline{CS}$ is tied high.

Parameters		Sym	Min	Typ	Max	Units	Conditions
Amplifier Input							
Input Offset Voltage		V_{OS}	-275	—	+275	µV	G = +1, V_{DD} = 4.0V
Input Offset Voltage Drift		$\Delta V_{OS}/\Delta T_A$	—	±4	—	µV/°C	T_A = -40 to +85°C
Power Supply Rejection Ratio		PSRR	70	85	—	dB	G = +1 **(Note 1)**
Input Bias Current		I_B	—	±1	—	pA	CHx = V_{DD}/2
Input Bias Current over Temperature		I_B	—	—	250	pA	T_A = -40 to +85°C, CHx = V_{DD}/2
Input Impedance		Z_{IN}	—	10^{13}‖15	—	Ω‖pF	
Input Voltage Range		V_{IVR}	V_{SS}–0.3	—	V_{DD}+0.3	V	
Amplifier Gain							
Nominal Gains		G	—	1 to 32	—	V/V	+1, +2, +4, +5, +8, +10, +16 or +32
DC Gain Error	G = +1	g_E	-0.1	—	+0.1	%	V_{OUT} ≈ 0.3V to V_{DD} – 0.3V
	G ≥ +2	g_E	-1.0	—	+1.0	%	V_{OUT} ≈ 0.3V to V_{DD} – 0.3V
DC Gain Drift	G = +1	$\Delta G/\Delta T_A$	—	±0.0002	—	%/°C	T_A = -40 to +85°C
	G ≥ +2	$\Delta G/\Delta T_A$	—	±0.0004	—	%/°C	T_A = -40 to +85°C
Internal Resistance		R_{LAD}	3.4	4.9	6.4	kΩ	**(Note 1)**
Internal Resistance over Temperature		$\Delta R_{LAD}/\Delta T_A$	—	+0.028	—	%/°C	**(Note 1)** T_A = -40 to +85°C
Amplifier Output							
DC Output Non-linearity	G = +1	V_{ONL}	—	±0.003	—	% of FSR	V_{OUT} = 0.3V to V_{DD} – 0.3V, V_{DD} = 5.0V
	G ≥ +2	V_{ONL}	—	±0.001	—	% of FSR	V_{OUT} = 0.3V to V_{DD} – 0.3V, V_{DD} = 5.0V
Maximum Output Voltage Swing		V_{OH}, V_{OL}	V_{SS}+20	—	V_{DD}-100	mV	G ≥ +2; 0.5V output overdrive
			V_{SS}+60	—	V_{DD}-60		G ≥ +2; 0.5V output overdrive, V_{REF} = V_{DD}/2
Short-Circuit Current		$I_{O(SC)}$	—	±30	—	mA	

Note 1: R_{LAD} (R_F + R_G in Figure 4-1) connects V_{REF}, V_{OUT} and the inverting input of the internal amplifier. The MCP6S22 has V_{REF} tied internally to V_{SS}, so V_{SS} is coupled to the internal amplifier and the PSRR spec describes PSRR+ only. We recommend the MCP6S22's V_{SS} pin be tied directly to ground to avoid noise problems.
2: I_Q includes current in R_{LAD} (typically 60 µA at V_{OUT} = 0.3V). Both I_Q and I_{Q_SHDN} exclude digital switching currents.
3: The output goes Hi-Z and the registers reset to their defaults; see Section 5.4, “Power-On Reset”.

DC CHARACTERISTICS (CONTINUED)

Electrical Specifications: Unless otherwise indicated, T_A = +25°C, V_{DD} = +2.5V to +5.5V, V_{SS} = GND, V_{REF} = V_{SS}, G = +1 V/V, Input = CH0 = (0.3V)/G, CH1 to CH7 = 0.3V, R_L = 10 kΩ to V_{DD}/2, SI and SCK are tied low and $\overline{CS}$ is tied high.

Parameters	Sym	Min	Typ	Max	Units	Conditions
Power Supply						
Supply Voltage	V_{DD}	2.5	—	5.5	V	
Quiescent Current	I_Q	0.5	1.0	1.35	mA	I_O = 0 **(Note 2)**
Quiescent Current, Shutdown mode	I_{Q_SHDN}	—	0.5	1.0	µA	I_O = 0 **(Note 2)**
Power-On Reset						
POR Trip Voltage	V_{POR}	1.2	1.7	2.2	V	**(Note 3)**
POR Trip Voltage Drift	$\Delta V_{POR}/\Delta T$	—	-3.0	—	mV/°C	T_A = -40°C to+85°C

Note 1: R_{LAD} (R_F + R_G in Figure 4-1) connects V_{REF}, V_{OUT} and the inverting input of the internal amplifier. The MCP6S22 has V_{REF} tied internally to V_{SS}, so V_{SS} is coupled to the internal amplifier and the PSRR spec describes PSRR+ only. We recommend the MCP6S22's V_{SS} pin be tied directly to ground to avoid noise problems.

2: I_Q includes current in R_{LAD} (typically 60 µA at V_{OUT} = 0.3V). Both I_Q and I_{Q_SHDN} exclude digital switching currents.

3: The output goes Hi-Z and the registers reset to their defaults; see Section 5.4, "Power-On Reset".

AC CHARACTERISTICS

Electrical Specifications: Unless otherwise indicated, T_A = +25°C, V_{DD} = +2.5V to +5.5V, V_{SS} = GND, V_{REF} = V_{SS}, G = +1 V/V, Input = CH0 =(0.3V)/G, CH1 to CH7=0.3V, R_L = 10 kΩ to V_{DD}/2, C_L = 60 pF, SI and SCK are tied low, and $\overline{CS}$ is tied high.

Parameters	Sym	Min	Typ	Max	Units	Conditions
Frequency Response						
-3 dB Bandwidth	BW	—	2 to 12	—	MHz	All gains; V_{OUT} < 100 mV_{P-P} **(Note 1)**
Gain Peaking	GPK	—	0	—	dB	All gains; V_{OUT} < 100 mV_{P-P}
Total Harmonic Distortion plus Noise						
f = 1 kHz, G = +1 V/V	THD+N	—	0.0015	—	%	V_{OUT} = 1.5V ± 1.0V_{PK}, V_{DD} = 5.0V, BW = 22 kHz
f = 1 kHz, G = +4 V/V	THD+N	—	0.0058	—	%	V_{OUT} = 1.5V ± 1.0V_{PK}, V_{DD} = 5.0V, BW = 22 kHz
f = 1 kHz, G = +16 V/V	THD+N	—	0.023	—	%	V_{OUT} = 1.5V ± 1.0V_{PK}, V_{DD} = 5.0V, BW = 22 kHz
f = 20 kHz, G = +1 V/V	THD+N	—	0.0035	—	%	V_{OUT} = 1.5V ± 1.0V_{PK}, V_{DD} = 5.0V, BW = 80 kHz
f = 20 kHz, G = +4 V/V	THD+N	—	0.0093	—	%	V_{OUT} = 1.5V ± 1.0V_{PK}, V_{DD} = 5.0V, BW = 80 kHz
f = 20 kHz, G = +16 V/V	THD+N	—	0.036	—	%	V_{OUT} = 1.5V ± 1.0V_{PK}, V_{DD} = 5.0V, BW = 80 kHz
Step Response						
Slew Rate	SR	—	4.0	—	V/µs	G = 1, 2
		—	11	—	V/µs	G = 4, 5, 8, 10
		—	22	—	V/µs	G = 16, 32
Noise						
Input Noise Voltage	E_{ni}	—	3.2	—	μV_{P-P}	f = 0.1 Hz to 10 kHz **(Note 2)**
		—	26	—		f = 0.1 Hz to 200 kHz **(Note 2)**
Input Noise Voltage Density	e_{ni}	—	10	—	nV/√Hz	f = 10 kHz **(Note 2)**
Input Noise Current Density	i_{ni}	—	4	—	fA/√Hz	f = 10 kHz

Note 1: See Table 4-1 for a list of typical numbers.

2: E_{ni} and e_{ni} include ladder resistance noise. See Figure 2-33 for e_{ni} vs. G data.

MCP6S21/2/6/8

DIGITAL CHARACTERISTICS

Electrical Specifications: Unless otherwise indicated, T_A = +25°C, V_{DD} = +2.5V to +5.5V, V_{SS} = GND, V_{REF} = V_{SS}, G = +1 V/V, Input = CH0 = (0.3V)/G, CH1 to CH7 = 0.3V, R_L = 10 kΩ to V_{DD}/2, C_L = 60 pF, SI and SCK are tied low, and $\overline{CS}$ is tied high.

Parameters	Sym	Min	Typ	Max	Units	Conditions
SPI Inputs ($\overline{CS}$, SI, SCK)						
Logic Threshold, Low	V_{IL}	0	—	$0.3V_{DD}$	V	
Input Leakage Current	I_{IL}	-1.0	—	+1.0	µA	
Logic Threshold, High	V_{IH}	$0.7V_{DD}$	—	V_{DD}	V	
Amplifier Output Leakage Current	—	-1.0	—	+1.0	µA	In Shutdown mode
SPI Output (SO, for MCP6S26 and MCP6S28)						
Logic Threshold, Low	V_{OL}	V_{SS}	—	V_{SS}+0.4	V	I_{OL} = 2.1 mA, V_{DD} = 5V
Logic Threshold, High	V_{OH}	V_{DD}-0.5	—	V_{DD}	V	I_{OH} = -400 µA
SPI Timing						
Pin Capacitance	C_{PIN}	—	10	—	pF	All digital I/O pins
Input Rise/Fall Times ($\overline{CS}$, SI, SCK)	t_{RFI}	—	—	2	µs	**Note 1**
Output Rise/Fall Times (SO)	t_{RFO}	—	5	—	ns	MCP6S26 and MCP6S28
$\overline{CS}$ high time	t_{CSH}	40	—	—	ns	
SCK edge to $\overline{CS}$ fall setup time	t_{CS0}	10	—	—	ns	SCK edge when $\overline{CS}$ is high
$\overline{CS}$ fall to first SCK edge setup time	t_{CSSC}	40	—	—	ns	
SCK Frequency	f_{SCK}	—	—	10	MHz	V_{DD} = 5V **(Note 2)**
SCK high time	t_{HI}	40	—	—	ns	
SCK low time	t_{LO}	40	—	—	ns	
SCK last edge to $\overline{CS}$ rise setup time	t_{SCCS}	30	—	—	ns	
$\overline{CS}$ rise to SCK edge setup time	t_{CS1}	100	—	—	ns	SCK edge when $\overline{CS}$ is high
SI set-up time	t_{SU}	40	—	—	ns	
SI hold time	t_{HD}	10	—	—	ns	
SCK to SO valid propagation delay	t_{DO}	—	—	80	ns	MCP6S26 and MCP6S28
$\overline{CS}$ rise to SO forced to zero	t_{SOZ}	—	—	80	ns	MCP6S26 and MCP6S28
Channel and Gain Select Timing						
Channel Select Time	t_{CH}	—	1.5	—	µs	CHx = 0.6V, CHy =0.3V, G = 1, CHx to CHy select $\overline{CS}$ = $0.7V_{DD}$ to V_{OUT} 90% point
Gain Select Time	t_G	—	1	—	µs	CHx = 0.3V, G = 5 to G = 1 select, $\overline{CS}$ = $0.7V_{DD}$ to V_{OUT} 90% point
Shutdown Mode Timing						
Out of Shutdown mode ($\overline{CS}$ goes high) to Amplifier Output Turn-on Time	t_{ON}	—	3.5	10	µs	$\overline{CS}$ = $0.7V_{DD}$ to V_{OUT} 90% point
Into Shutdown mode ($\overline{CS}$ goes high) to Amplifier Output High-Z Turn-off Time	t_{OFF}	—	1.5	—	µs	$\overline{CS}$ = $0.7V_{DD}$ to V_{OUT} 90% point
POR Timing						
Power-On Reset power-up time	t_{RPU}	—	30	—	µs	V_{DD} = V_{POR} - 0.1V to V_{POR} + 0.1V, 50% V_{DD} to 90% V_{OUT} point
Power-On Reset power-down time	t_{RPD}	—	10	—	µs	V_{DD} = V_{POR} + 0.1V to V_{POR} - 0.1V, 50% V_{DD} to 90% V_{OUT} point

Note 1: Not tested in production. Set by design and characterization.

2: When using the device in the daisy chain configuration, maximum clock frequency is determined by a combination of propagation delay time ($t_{DO} \leq 80$ ns), data input setup time ($t_{SU} \geq 40$ ns), SCK high time ($t_{HI} \geq 40$ ns), and SCK rise and fall times of 5 ns. Maximum f_{SCK} is, therefore, ≈ 5.8 MHz.

TEMPERATURE CHARACTERISTICS

Electrical Specifications: Unless otherwise indicated, V_{DD} = +2.5V to +5.5V, V_{SS} = GND.

Parameters	Sym	Min	Typ	Max	Units	Conditions
Temperature Ranges						
Specified Temperature Range	T_A	-40	—	+85	°C	
Operating Temperature Range	T_A	-40	—	+125	°C	**(Note Note:)**
Storage Temperature Range	T_A	-65	—	+150	°C	
Thermal Package Resistances						
Thermal Resistance, 8L-PDIP	θ_{JA}	—	85	—	°C/W	
Thermal Resistance, 8L-SOIC	θ_{JA}	—	163	—	°C/W	
Thermal Resistance, 8L-MSOP	θ_{JA}	—	206	—	°C/W	
Thermal Resistance, 14L-PDIP	θ_{JA}	—	70	—	°C/W	
Thermal Resistance, 14L-SOIC	θ_{JA}	—	120	—	°C/W	
Thermal Resistance, 14L-TSSOP	θ_{JA}	—	100	—	°C/W	
Thermal Resistance, 16L-PDIP	θ_{JA}	—	70	—	°C/W	
Thermal Resistance, 16L-SOIC	θ_{JA}	—	90	—	°C/W	

Note 1: The MCP6S21/2/6/8 family of PGAs operates over this extended temperature range, but with reduced performance. Operation in this range must not cause T_J to exceed the Maximum Junction Temperature (150°C).

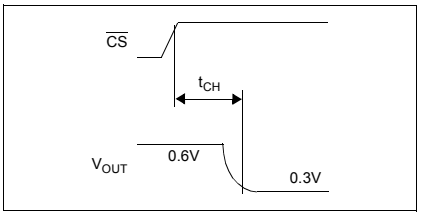


FIGURE 1-1: *Channel Select Timing Diagram.*

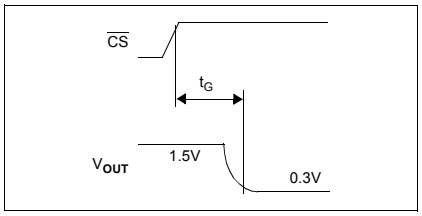


FIGURE 1-3: *Gain Select Timing Diagram.*

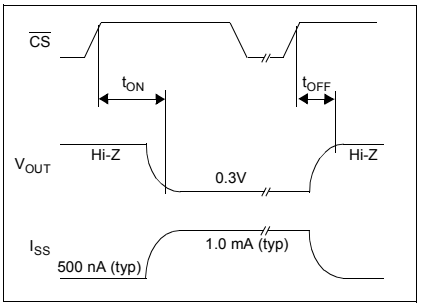


FIGURE 1-2: *PGA Shutdown timing diagram (must enter correct commands before $\overline{CS}$ goes high).*

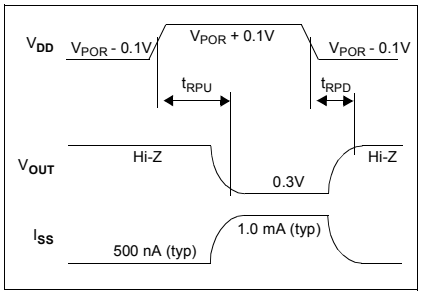


FIGURE 1-4: *POR power-up and power-down timing diagram.*

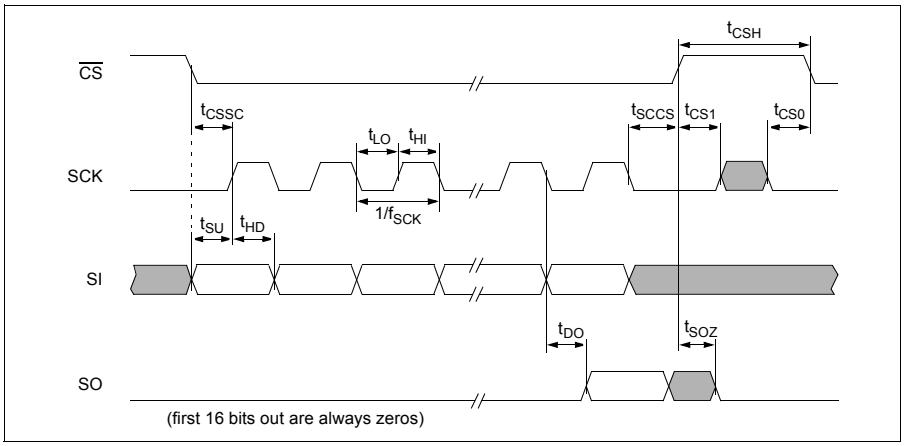


FIGURE 1-5: *Detailed SPI Serial Interface Timing, SPI 0,0 mode.*

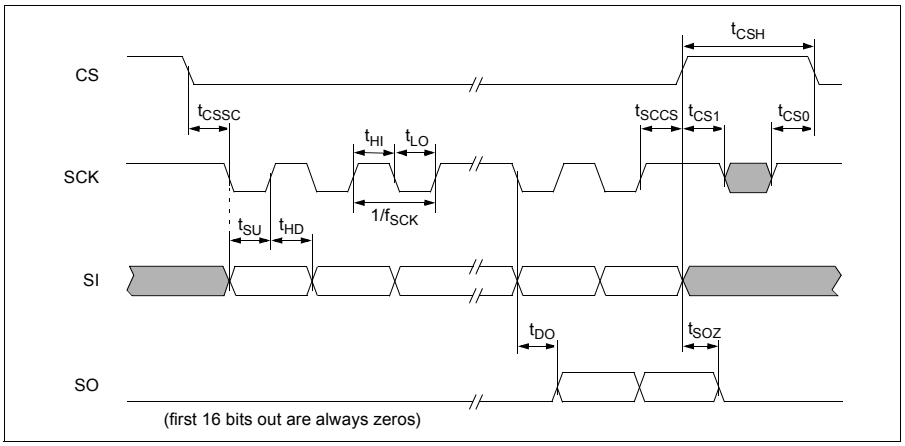


FIGURE 1-6: *Detailed SPI Serial Interface Timing, SPI 1,1 mode.*

1.1 DC Output Voltage Specs / Model

1.1.1 IDEAL MODEL

The ideal PGA output voltage (V_{OUT}) is:

EQUATION

$$V_{O\_ideal} = GV_{IN} \qquad V_{REF} = V_{SS} = 0V$$

where: G is the nominal gain

(see Figure 1-7). This equation holds when there are no gain or offset errors and when the V_{REF} pin is tied to a low impedance source (<< 0.1Ω) at ground potential (V_{SS} = 0V).

1.1.2 LINEAR MODEL

The PGA's linear region of operation, including offset and gain errors, is modeled by the line V_{O_linear}, shown in Figure 1-7.

EQUATION

$$V_{O\_linear} = G(1+g_E)(V_{IN} - 0.3V + V_{OS}) + 0.3V$$
$$V_{REF} = V_{SS} = 0V$$

The endpoints of this line are at V_{O_ideal} = 0.3V and V_{DD}-0.3V. The gain and offset specifications referred to in the electrical specifications are related to Figure 1-7, as follows:

EQUATION

$$g_E = 100\% \frac{V_2 - V_1}{G(V_{DD} - 0.6V)}$$

$$V_{OS} = \frac{V_1}{G(1+g_E)} \qquad G = +1$$

$$\Delta G / \Delta T_A = \frac{\Delta g_E}{\Delta T_A}$$

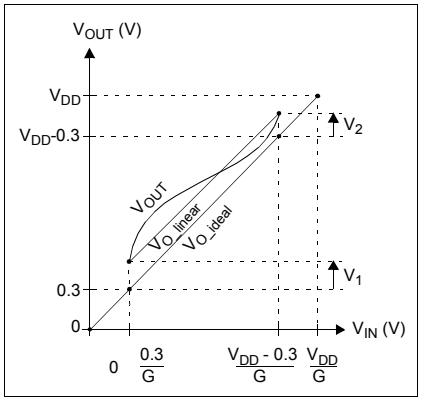


FIGURE 1-7: *Output Voltage Model with the standard condition V_{REF} = V_{SS} = 0V.*

1.1.3 OUTPUT NON-LINEARITY

Figure 1-8 shows the Integral Non-Linearity (INL) of the output voltage.

EQUATION

$$INL = V_{OUT} - V_{O\_linear}$$

The output non-linearity specification in the electrical specifications is related to Figure 1-8 by:

EQUATION

$$V_{ONL} = \frac{max\{V_4, V_3\}}{V_{DD} - 0.6V}$$

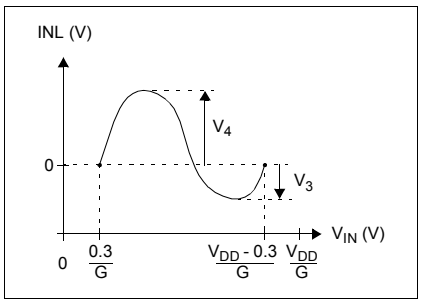


FIGURE 1-8: *Output Voltage INL with the standard condition V_{REF} = V_{SS} = 0V.*

1.1.4 DIFFERENT V_{REF} CONDITIONS

Some of the plots in Section 2.0, “Typical Performance Curves”, have the conditions $V_{REF} = V_{DD}/2$ or $V_{REF} = V_{DD}$. The equations and figures above are easily modified for these conditions. The ideal V_{OUT} becomes:

EQUATION

$$V_{O\_ideal} = V_{REF} + G(V_{IN} - V_{REF})$$

$$V_{DD} \geq V_{REF} > V_{SS} = 0V$$

The complete linear model is:

EQUATION

$$V_{O\_linear} = G(1 + g_E)(V_{IN} - V_{IN\_L} + V_{OS}) + 0.3V$$

where the new V_{IN} endpoints are:

EQUATION

$$V_{IN\_L} = \frac{0.3V - V_{REF}}{G + V_{REF}}$$

$$V_{IN\_R} = \frac{V_{DD} - 0.3V - V_{REF}}{G + V_{REF}}$$

The equations for extracting the specifications do not change.

3.0 PIN DESCRIPTIONS

The descriptions of the pins are listed in Table 3-1.

TABLE 3-1: PIN FUNCTION TABLE

MCP6S21	MCP6S22	MCP6S26	MCP6S28	Symbol	Description
1	1	1	1	V_{OUT}	Analog Output
2	2	2	2	CH0	Analog Input
—	3	3	3	CH1	Analog Input
—	—	4	4	CH2	Analog Input
—	—	5	5	CH3	Analog Input
—	—	6	6	CH4	Analog Input
—	—	7	7	CH5	Analog Input
—	—	—	8	CH6	Analog Input
—	—	—	9	CH7	Analog Input
3	—	8	10	V_{REF}	External Reference Pin
4	4	9	11	V_{SS}	Negative Power Supply
5	5	10	12	$\overline{CS}$	SPI Chip Select
6	6	11	13	SI	SPI Serial Data Input
—	—	12	14	SO	SPI Serial Data Output
7	7	13	15	SCK	SPI Clock Input
8	8	14	16	V_{DD}	Positive Power Supply

3.1 Analog Output

The output pin (V_{OUT}) is a low-impedance voltage source. The selected gain (G), selected input (CH0-CH7) and voltage at V_{REF} determine its value.

3.2 Analog Inputs (CH0 thru CH7)

The inputs CH0 through CH7 connect to the signal sources. They are high-impedance CMOS inputs with low bias currents. The internal MUX selects which one is amplified to the output.

3.3 External Reference Voltage (V_{REF})

The V_{REF} pin should be at a voltage between V_{SS} and V_{DD} (the MCP6S22 has V_{REF} tied internally to V_{SS}). The voltage at this pin shifts the output voltage.

3.4 Power Supply (V_{SS} and V_{DD})

The positive power supply pin (V_{DD}) is 2.5V to 5.5V higher than the negative power supply pin (V_{SS}). For normal operation, the other pins are between V_{SS} and V_{DD}.

Typically, these parts are used in a single (positive) supply configuration. In this case, V_{SS} is connected to ground and V_{DD} is connected to the supply. V_{DD} will need a local bypass capacitor (0.1 µF) at the V_{DD} pin. It can share a bulk capacitor with nearby analog parts (typically 2.2 µF to 10 µF within 4 inches (100 mm) of the V_{DD} pin.

3.5 Digital Inputs

The SPI interface inputs are: Chip Select ($\overline{CS}$), Serial Input (SI) and Serial Clock (SCK). These are Schmitt-triggered, CMOS logic inputs.

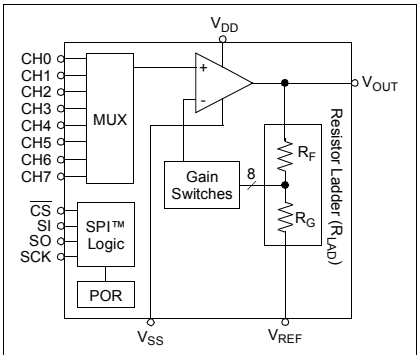
3.6 Digital Output

The MCP6S26 and MCP6S28 devices have a SPI interface serial output (SO) pin. This is a CMOS push-pull output and does not ever go High-Z. Once the device is deselected ($\overline{CS}$ goes high), SO is forced low. This feature supports daisy chaining, as explained in Section 5.3, “Daisy Chain Configuration”.

MCP6S21/2/6/8

4.0 ANALOG FUNCTIONS

The MCP6S21/2/6/8 family of Programmable Gain Amplifiers (PGA) are based on simple analog building blocks (see Figure 4-1). Each of these blocks will be explained in more detail in the following sub-sections.



MCP6S21–One input (CH0), no SO pin

MCP6S22–Two inputs (CH0, CH1), V_{REF} tied internally to V_{SS}, no SO pin

MCP6S26–Six inputs (CH0 to CH5)

MCP6S28–Eight inputs (CH0 to CH7)

FIGURE 4-1: *PGA Block Diagram.*

4.1 Input MUX

The MCP6S21 has one input, the MCP6S22 and MCP6S25 have two inputs, the MCP6S26 has six inputs and the MCP6S28 has eight inputs (see Figure 4-1).

For the lowest input current, float unused inputs. Tying these pins to a voltage near the used channels also works well. For simplicity, they can be tied to V_{SS} or V_{DD}, but the input current may increase.

The one channel MCP6S21 has the lowest input bias current, while the eight channel MCP6S28 has the highest. There is about a 2:1 ratio in I_B between these parts.

4.2 Internal Op Amp

The internal op amp provides the right combination of bandwidth, accuracy and flexibility.

4.2.1 COMPENSATION CAPACITORS

The internal op amp has three compensation capacitors connected to a switching network. They are selected to give good small signal bandwidth at high gains, and good slew rate (full power bandwidth) at low gains. The change in bandwidth as gain changes is between 2 MHz and 12 MHz. Refer to Table 4-1 for more information.

TABLE 4-1: GAIN VS. INTERNAL COMPENSATION CAPACITOR

Gain (V/V)	Internal Compensation Capacitor	Typical GBWP (MHz)	Typical SR (V/µs)	Typical FPBW (MHz)	Typical BW (MHz)
1	Large	12	4.0	0.30	12
2	Large	12	4.0	0.30	6
4	Medium	20	11	0.70	10
5	Medium	20	11	0.70	7
8	Medium	20	11	0.70	2.4
10	Medium	20	11	0.70	2.0
16	Small	64	22	1.6	5
32	Small	64	22	1.6	2.0

Note 1: FPBW is the Full Power Bandwidth. These numbers are based on V_{DD} = 5.0V.
2: No changes in DC performance (e.g., V_{OS}) accompany a change in compensation capacitor.
3: BW is the closed-loop, small signal -3 dB bandwidth.

4.2.2 RAIL-TO-RAIL INPUT

The input stage of the internal op amp uses two differential input stages in parallel; one operates at low V_{IN} (input voltage), while the other operates at high V_{IN}. With this topology, the internal inputs can operate to 0.3V past either supply rail. The input offset voltage is measured at both $V_{IN} = V_{SS}$ - 0.3V and V_{DD} + 0.3V to ensure proper operation.

The transition between the two input stages occurs when $V_{IN} \approx V_{DD}$ - 1.5V. For the best distortion and gain linearity, avoid this region of operation.

4.2.3 RAIL-TO-RAIL OUTPUT

The Maximum Output Voltage Swing is the maximum swing possible under a particular output load. According to the specification table, the output can reach within 60 mV of either supply rail when R_L = 10 kΩ and $V_{REF} = V_{DD}/2$. See Figure 2-21 for typical performance under other conditions.

4.2.4 INPUT VOLTAGE AND PHASE REVERSAL

The amplifier family is designed with CMOS input devices. It is designed to not exhibit phase inversion when the input pins exceed the supply voltages. Figure 2-34 shows an input voltage exceeding both supplies with no resulting phase inversion.

The maximum voltage that can be applied to the input pins (CHX) is V_{SS} - 0.3V to V_{DD} + 0.3V. Voltages on the inputs that exceed this absolute maximum rating can cause excessive current to flow in or out of the input pins. Current beyond ±2 mA can cause possible reliability problems. Applications that exceed this rating must be externally limited with an input resistor, as shown in Figure 4-2.

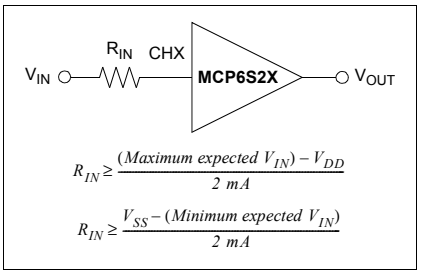


FIGURE 4-2: *R_{IN} limits the current flow into an input pin.*

4.3 Resistor Ladder

The resistor ladder shown in Figure 4-1 ($R_{LAD} = R_F + R_G$) sets the gain. Placing the gain switches in series with the inverting input reduces the parasitic capacitance, distortion and gain mismatch.

R_{LAD} is an additional load on the output of the PGA and causes additional current draw from the supplies.

In Shutdown mode, R_{LAD} is still attached to the OUT and V_{REF} pins. Thus, these pins and the internal amplifier's inverting input are all connected through R_{LAD} and the output is not high-Z (unlike the external op amp).

While R_{LAD} contributes to the output noise, its effect is small. Refer to Figure 2-12.

4.4 Shutdown Mode

These PGAs use a software shutdown command. When the SPI interface sends a shutdown command, the internal op amp is shut down and its output placed in a high-Z state.

The resistive ladder is always connected between V_{REF} and V_{OUT}, even in shutdown. This means that the output resistance will be on the order of 5 kΩ and there will be a path for output signals to appear at the input.

The Power-on Reset (POR) circuitry will temporarily place the part in shutdown when activated. See Section 5.4, "Power-On Reset", for details.

5.0 DIGITAL FUNCTIONS

The MCP6S21/2/6/8 PGAs use a standard SPI compatible serial interface to receive instructions from a controller. This interface is configured to allow daisy chaining with other SPI devices. There is an internal POR (Power On Reset) that resets the registers under low power conditions.

5.1 SPI Timing

Chip Select ($\overline{CS}$) toggles low to initiate communication with these devices. The first byte of each SI word (two bytes long) is the instruction byte, which goes into the Instruction Register. The Instruction Register points the second byte to its destination. In a typical application, $\overline{CS}$ is raised after one word (16 bits) to implement the desired changes. Section 5.3, “Registers”, covers applications using multiple 16-bit words. SO goes low after $\overline{CS}$ goes high; it has a push-pull output that does not go into a high-Z state.

The MCP6S21/2/6/8 devices operate in SPI Modes 0,0 and 1,1. In 0,0 mode, the clock idles in the low state (Figure 5-1) and, in 1,1 mode, the clock idles in the high state (Figure 5-2). In both modes, SI data is loaded into the PGA on the rising edge of SCK and SO data is clocked out on the falling edge of SCK. In 0,0 mode, the falling edge of $\overline{CS}$ also acts as the first falling edge of SCK (see Figure 5-1). There must be multiples of 16 clocks (SCK) while $\overline{CS}$ is low or commands will abort (see Section 5.3, “Registers”).

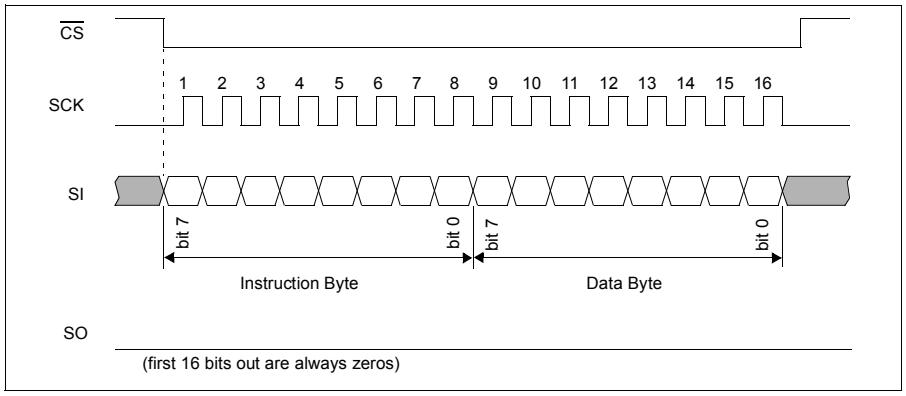


FIGURE 5-1: *Serial bus sequence for the PGA; SPI 0,0 mode (see Figure 1-5).*

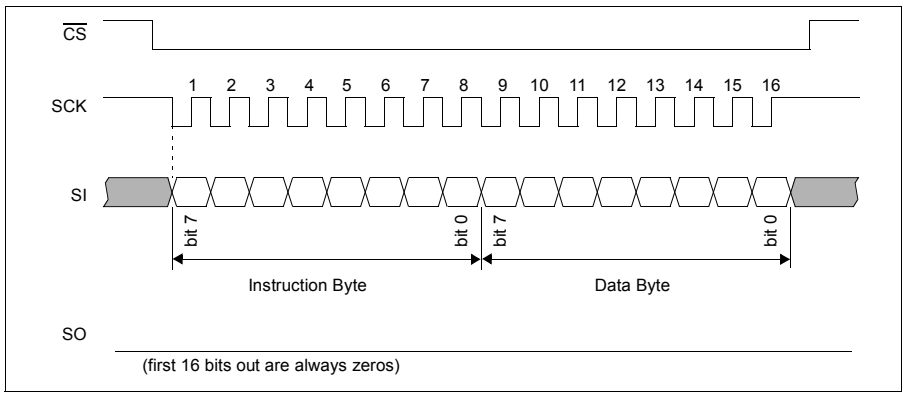


FIGURE 5-2: *Serial bus sequence for the PGA; SPI 1,1 mode (see Figure 1-6).*

5.2 Registers

The analog functions are programmed through the SPI interface using 16-bit words (see Figure 5-1 and Figure 5-2). This data is sent to two of three 8-bit registers: Instruction Register (Register 5-1), Gain Register (Register 5-2) and Channel Register (Register 5-3). The power-up defaults for these three registers are:

- Instruction Register: 000x xxx0
- Gain Register: xxxx x000
- Channel Register: xxxx x000

Thus, these devices are initially programmed with the Instruction Register set for NOP (no operation), a gain of +1 V/V and CH0 as the input channel.

5.2.1 INSTRUCTION REGISTER

The Instruction Register has 3 command bits and 1 indirect address bit; see Register 5-1. The command bits include a NOP (000) to support daisy chaining (see Section 5.3, "Registers"); the other NOP commands shown should not be used (they are reserved for future use). The device is brought out of Shutdown mode when a valid command, other than NOP or Shutdown, is sent and $\overline{CS}$ is raised.

REGISTER 5-1: INSTRUCTION REGISTER

W-0	W-0	W-0	U-x	U-x	U-x	U-x	W-0
M2	M1	M0	—	—	—	—	A0
bit 7							bit 0

bit 7-5 **M2-M0: Command Bits**
000 = NOP (Default) **(Note 1)**
001 = PGA enters Shutdown Mode as soon as a full 16-bit word is sent and $\overline{CS}$ is raised. **(Notes 1 and 2)**
010 = Write to register.
011 = NOP (reserved for future use) **(Note 1)**
1XX = NOP (reserved for future use) **(Note 1)**

bit 4-1 **Unimplemented:** Read as '0' (reserved for future use)

bit 0 **A0: Indirect Address Bit**
1 = Addresses the Channel Register
0 = Addresses the Gain Register (Default)

Note 1: All other bits in the 16-bit word (including A0) are "don't cares".

2: The device exits Shutdown mode when a valid command (other than NOP or Shutdown) is sent and $\overline{CS}$ is raised; that valid command will be executed. Shutdown does not toggle.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

5.2.2 SETTING THE GAIN

The amplifier can be programmed to produce binary and decimal gain settings between +1 V/V and +32 V/V. Register 5-2 shows the details. At the same time, different compensation capacitors are selected to optimize the bandwidth vs. slew rate trade-off (see Table 4-1).

REGISTER 5-2: GAIN REGISTER

U-x	U-x	U-x	U-x	U-x	W-0	W-0	W-0
—	—	—	—	—	G2	G1	G0
bit 7							bit 0

bit 7-3 **Unimplemented:** Read as '0' (reserved for future use)

bit 2-0 **G2-G0: Gain Select Bits**

000 = Gain of +1 (Default)
001 = Gain of +2
010 = Gain of +4
011 = Gain of +5
100 = Gain of +8
101 = Gain of +10
110 = Gain of +16
111 = Gain of +32

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

5.2.3 CHANGING THE CHANNEL

If the instruction register is programmed to address the channel register, the multiplexed inputs of the MCP6S22, MCP6S26 and MCP6S28 can be changed per Register 5-3.

REGISTER 5-3: CHANNEL REGISTER

U-x	U-x	U-x	U-x	U-x	W-0	W-0	W-0
—	—	—	—	—	C2	C1	C0
bit 7							bit 0

bit 7-3 **Unimplemented:** Read as '0' (reserved for future use)

bit 2-0 **C2-C0: Channel Select Bits**

	MCP6S21	MCP6S22	MCP6S26	MCP6S28
000	= CH0 (Default)	CH0 (Default)	CH0 (Default)	CH0 (Default)
001	= CH0	CH1	CH1	CH1
001	= CH0	CH0	CH2	CH2
011	= CH0	CH1	CH3	CH3
100	= CH0	CH0	CH4	CH4
101	= CH0	CH1	CH5	CH5
110	= CH0	CH0	CH0	CH6
111	= CH0	CH1	CH0	CH7

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

5.2.4 SHUTDOWN COMMAND

The software Shutdown command allows the user to put the amplifier into a low power mode (see Register 5-1). In this shutdown mode, most pins are high impedance (Section 4.4, “Shutdown Mode”, and Section 5.1, “SPI Timing”, cover the exceptions at pins V_{REF}, V_{OUT} and SO).

Once the PGA has entered shutdown mode, it will remain in this mode until either a valid command is sent to the device (other than NOP or Shutdown), or the device is powered down and back up again. The internal registers maintain their values while in shutdown.

Once brought out of shutdown mode, the part comes back to its previous state (see Section 5.4 for exceptions to this rule). This makes it possible to bring the device out of shutdown mode using one command; send a command to select the current channel (or gain) and the device will exit shutdown with the same state that existed before shutdown.

5.3 Daisy Chain Configuration

Multiple devices can be connected in a daisy chain configuration by connecting the SO pin from one device to the SI pin on the next device and using common SCK and $\overline{CS}$ lines (Figure 5-3). This approach reduces PCB layout complexity.

The example in Figure 5-3 shows a daisy chain configuration with two devices, although any number of devices can be configured this way. The MCP6S21 and MCP6S22 can only be used at the far end of the daisy chain because they do not have a serial data out (SO) pin. As shown in Figure 5-4 and Figure 5-5, both SI and SO data are sent in 16-bit (2 byte) words. These devices abort any command that is not a multiple of 16 bits.

When using the daisy chain configuration, the maximum clock speed possible is reduced to ≈ 5.8 MHz because of the SO pin’s propagation delay (see Electrical Specifications).

The internal SPI shift register is automatically loaded with zeros whenever $\overline{CS}$ goes high (a command is executed). Thus, the first 16-bits out of the SO pin once $\overline{CS}$ line goes low are always zeros. This means that the first command loaded into the next device in the daisy chain is a NOP. This feature makes it possible to send shorter command and data byte strings when the farthest devices do not need to change. For example, if there were three devices on the chain and only the middle device needed changing, only 32 bytes of data need to be transmitted (for the first and middle devices), and the last device on the chain would receive a NOP when the $\overline{CS}$ pin is raised to execute the command.

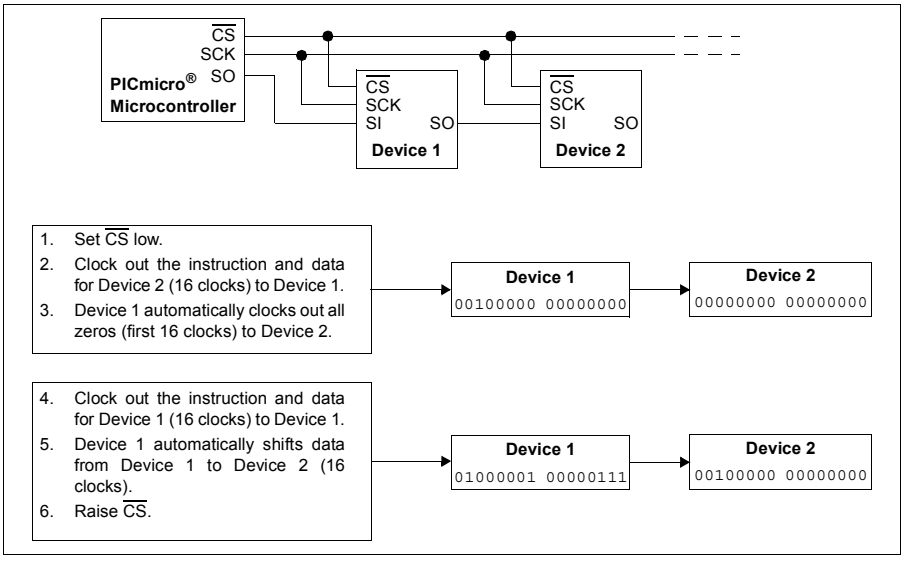


FIGURE 5-3: *Daisy Chain Configuration.*

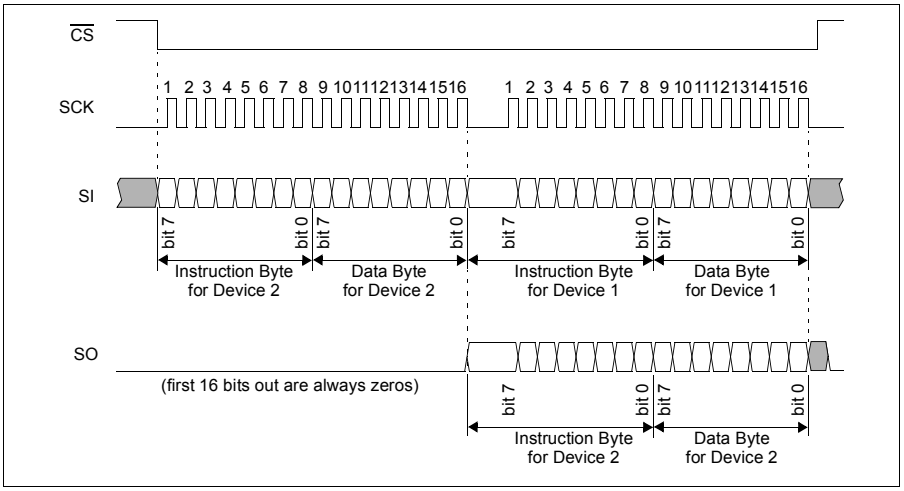


FIGURE 5-4: *Serial bus sequence for daisy-chain configuration; SPI 0,0 mode.*

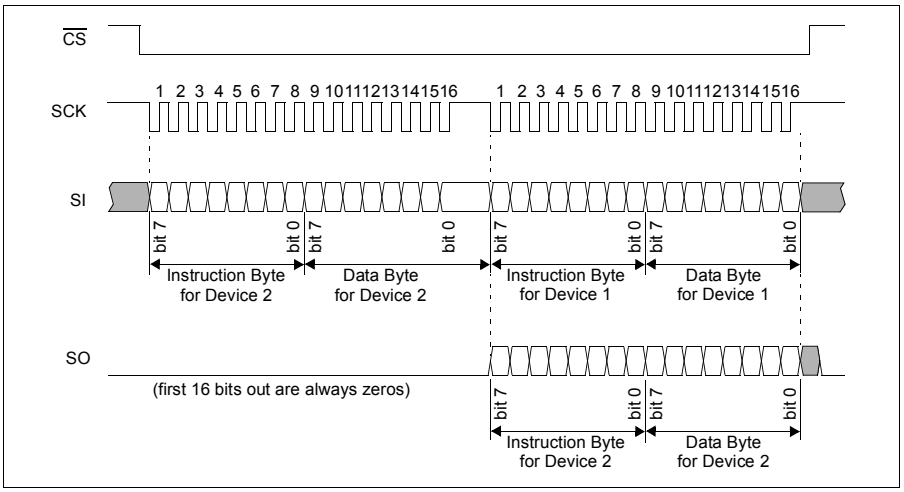


FIGURE 5-5: *Serial bus sequence for daisy-chain configuration; SPI 1,1 mode.*

4. Código de Arduino

```
/*
    ================================================================================


Program:        POF manager v18
Author:         Jose Lisbona
Date:           11/11/2015

v18
Corregido el string INFO devuelto

v17ñ
Aadida opcion de estado de filtro a la info //TODO
Cambiado el reloj del prescaler del ADC a 128, para cumplir con la precision
    de 10 bits (estaba a 1MHz y debe estar a 125 KHz) ADCSRA LSB-> F en lugar
    de E

  -Filtro ON/OFF

Comando
   #W0000
Respuesta
   INFO chopear/periodo_cuad/intervaloPuertoSerie/pwm1_value/pwm2_value/
       g1_value/g2_value/g3_value/filtro

v16ñ
Aadida opcion para eliminar filtro
#F0000 lo desactiva
#F0001 lo activa
Los comandos para el puerto serie solo se tramitan despues de haber procesado
    un suelo

mejorado el reset, ya no se intercambian suelos y techos



v15
Modificadas las respuestas para no confundirse con comandos.
Para devolver dato
  T(VALOR)S(VALOR)
Para responder comando
I-(Comando)
Y se devuelve siempre la configuracion completa
  -Chop ON/OFF                              1/0 -> variable chopear
  -Valor D (periodo TRX)                        -> variable periodo_cuad
  -Valor M ( 0 - dato bajo demanda, otro valor (numero de muestras tras las
      que enviar dato))  -> variable intervaloPuertoSerie
  -Valor P (potencia transmitida)                -> pwm1_value
  -Valor Q(offset en receptor)                    -> pwm2_value
  -Valor G1                                       -> g1_value (creado)
  -Valor G2
  -Valor G3

Comando
   #W0000
Respuesta
```

```
    INFO chopear/periodo_cuad/intervaloPuertoSerie/pwm1_value/pwm2_value/
        g1_value/g2_value/g3_value

  Ejemplo:
    INFO 1/118/0/230/0/1/1/1



para ello tener un vector con las var

TODO ñAadir posibilidad de establecer todos los parametros juntos
  SET ...



v14
La lectura de datos Serie es constante, de manera que si se reciben todos los
    valores juntos
se apliquen a la vez, posibilitando  la configuracion en una unica cadena
    larga


v13
Los envios por puerto serie se realizan bajo demanda, por defecto desactivados
    .
Se aplica el filtro por separado para techos y suelos, enviando el dato de
    nuevo como TxxxxSyyyy
Desactivado el promediado
Revisado comando M,
#M0000 devuelve el dato actual
#MXXXX devuelve el dato tras XXXX iteraciones de panera periodica

v12
Anyadido filtro IIR, para activarlo, enviar el comando #M0000, el resto de los
     valores
activan el promediado

v11
Salida de muestras constante, cada 64 valores recibidos, independientemente
    del muestreo

v10
Los promedios ahora salen directamente como DXXXX

v9
La salida de los datos es a frecuencia constante, tras 4096 datos
#MXXXX puede ser 1,2,4,8,16,32,64,128

v8ñ
Aadidos comandos AX,BX,CX para controlar la ganancia de cada PGA
    individualmente

v7
Cambios respecto a v6
Se reinicia el promediado al recibir un comando

v6
Cambios respecto a v5
-ñAadido tiempo de espera al inicio para estabilizar el emisor (5 segundos)
```

```
Cambios respecto a v4
-Cambiados SS1 y SS2 a los pines 9 y 10
-ñRediseo de las tareas para reducir el tiempo de computo (promedio separado)
-ñAadidos modo de reposo tras ejecucion de bucle principal, que finaliza al
    llegar un comando o una interrupcion
-Desconexion de perifericos no utilizados, como el Timer2 y el comparador
    analogico
-Correccion de bugs como la no sincronizacion correcta de techos y suelos tras
     recibir comandos
-ñAadida tarea "flush" para evitar el envio de muestras incorrectas
-El pin de prueba mide el tiempo de reposo
-Revision general del codigo, ñaadiendo mas comentarios
-ñAadir comando M, que determina cuanto se muestrea 1/2/4/8/16/32/64

Bugs
-Techos y Suelos se intercambian al recibir comandos o resetear


Cambios respecto a v1

-Eliminado codigo innecesario
-Cambio de Timer2 a Timer0 (output compare por hardware)
-Reestructuracion de codigo para aligerar interrupciones
-Cambios en las ganancias de los PGA para que sean multiplos de 2

Config Arduino

 El timer 0 se configura para interrupciones marcadas por OC0A(8bits)*31 us


 El timer 1 se configura para PWM a 10Khz (mediante output compare)

 pin 13 SCK
 pin 11 MOSI
 pin 10 Pwm base T3
 pin 9 Pwm recepcion
 pin 8 SS3 (SS PGA3)
 pin 7 SS2 (SS PGA2)
 pin 6 Onda cuadrada trx
 pin 3 SS1 (SS PGA1)


 pin A0 entrada analogica

 Pinout para MCP6S21

 Con Arduino

 PIN 7 SCK
 PIN 6 MOSI (SI)-
 PIN 5 SS (not CS)



 Otros
 PIN 8 VDD +5V
 PIN 4 VSS GND
 PIN 3 VREF GND
 PIN 2 CH0 Vin
 PIN 1 VOUT
```

```
    ==========================================================================
    */
#include <avr/interrupt.h>
#include <avr/io.h>
#include <avr/power.h>
#include <avr/sleep.h>
// Constantes
//pines
#define SCK_pin 13
#define MOSI_pin 11
#define SS1_pin 3
#define SS2_pin 7
#define SS3_pin 8
#define trx_pin 6
#define pwm_trx_pin 10
#define pwm_rcx_pin 9
//valores del programa



//valores iniciales
//#define T_INICIAL 118 //64 Hz (1 seg)  118*4*33 us
#define T_INICIAL 200 //118 //256 Hz (1 seg)  118*33 us
#define P_INICIAL 230 // Algo por debajo de 10mA, para no saturar el receptor
#define Q_INICIAL 0 //Offset nulo
#define CHOP_INICIAL 1 //Chopeado ON
#define G1_INICIAL 1 //LSB=0 (ganancia x1)
#define G2_INICIAL 1
#define G3_INICIAL 1



#define BAUDRATE 115200

#define INIT_ADD_CONVERSION_VALUE 0x1F
#define END_ADD_CONVERSION_VALUE 0x0F

//Definicion de flags y variables asociadas
boolean data_flag; //Data valid

boolean cambiarValorChop=false;

unsigned char techo_flag;  //marcar si he convertido un techo o un suelo

//definicion de otras variables
unsigned char periodo_cuad;

//Variables SPI
volatile unsigned char intr_byte=0x40;
volatile unsigned char data_byte=0x0; //Ganancia x1
// Variables trx y rcx
unsigned char chopear; //chopear

//Funciones de configuracion
void config_pines();
void inic_variables();
void config_timer0(); //ñSeal transmision y sincronizacion receptor
```

```
void config_PWM();  //PWM_trx y PWM_trx
void config_AD();  //conversor AD
void config_SPI(); //PGAS
//Tareas principales
void calcularDato();  //Comprobar eventos tras conversion AD y respuesta
void comprobarPuertoSerie(); //Busqueda y parseo de comandos serie
void dormir(); //Poner en modo reposo la CPU
//Funciones de envio de datos
void generar_PWM();  //Control de la ñseal PWM
void transferirPGA(unsigned char quePGA, unsigned char dato1,unsigned char
    dato0); //Control PGA1
//Funciones auxiliares
void pars_gan(unsigned int gan); //determinar valores a escribir en PGAs en
    funcion del entero recibido
void desplazamiento(unsigned int *); //desplazamiento variable segun promedio
void inic_val_promedio(int n);//Configuracion del promedio en funcion del
    entero recibido
void config_chop(void); //configurar el timer0 en modo chopeado o no
void reset_valores(void); //reiniciar los datos
void reinicio_total(void); //reinicio y vuelta a config inicial
void datoAGanancia(int val); //Obtener el valor del caracter que representa la
     ganancia
//Interrupcion que indica el momento de convertir
ISR(TIMER0_COMPB_vect) {
  ADCSRA=0XCF; //Activar conversor
};
//Interrupcion que indica el momento de conversion terminada
unsigned char p_alta;
unsigned char p_baja;
unsigned int techo;
unsigned int suelo;
//unsigned int *puntero;

unsigned int convertido; //Comunicacion con bucle principal
ISR(ADC_vect) {
  //Aviso de que el dato esta listo
  data_flag=true;
  p_baja=ADCL; //ADCL primero para bloquear ADCH
  p_alta=ADCH;

   if (!techo_flag){
               suelo=(p_alta << 8) | p_baja;
               techo_flag=true;
       //Para procesar el caso de reset en el momento adecuado
   }else {
           techo_flag=false;

            techo=(p_alta << 8) | p_baja;
            //Prueba para ver si no se cambia solo el valor de suelo y t

   }


  ADCSRA=END_ADD_CONVERSION_VALUE;//Desactivar ADC

       if (cambiarValorChop&&!techo_flag)
      {
        techo_flag=true;
         config_chop();
```

```
            cambiarValorChop=false;
        }
};

unsigned int intervaloPuertoSerie;
//Este codigo se ejecuta al arrancar o resetear el programa
void setup() {

  noInterrupts();
  power_timer2_disable();
  power_twi_disable();
  inic_variables();
  config_pines();
  config_PWM();
  config_SPI();
  config_AD();
  config_timer0();
  interrupts();
  Serial.begin(BAUDRATE);
//  Serial.println("Inicio captura");

}

//Este codigo se ejecuta constantemente de principio a fin
void loop() {
  calcularDato();
  comprobarPuertoSerie();
  dormir();
}

unsigned int prom;
unsigned int num_muestras;


unsigned long y_techo,y_techo_1,y_suelo,y_suelo_1;
unsigned long tmp;

boolean filtro_flag;

boolean serial_flag=false;
void calcularDato(){
 if (data_flag) {
    num_muestras++;
    data_flag=false;
    if (techo_flag){

        if (filtro_flag)
        {
        //Aplicar el filtro
      tmp=y_techo_1*63+(techo<<6);
     y_techo=tmp>>6;
     y_techo_1=y_techo;
         }
         else
         {
         //No aplicar el filtro
         y_techo=techo;
         }
```

```
} else {
                if (filtro_flag)
                {
                        tmp=y_suelo_1*63+(suelo<<6);
                         y_suelo=tmp>>6;
                         y_suelo_1=y_suelo;
                }
        else
        {
                        y_suelo=suelo;
        }
                 if ((num_muestras==(intervaloPuertoSerie))&&(serial_flag)){
                Serial.print('T');
                Serial.print(y_techo);
                Serial.print('S');
                Serial.println(y_suelo);
                  num_muestras=0;
        }
}

 }
}
//Iniciar las variables la pirmera vez
void inic_variables(){
chopear=CHOP_INICIAL; //por defecto true

data_flag=false;
y_techo_1=512;
y_suelo_1=512;
techo_flag=true;
serial_flag=false;
num_muestras=1;
filtro_flag=false;
periodo_cuad=T_INICIAL; //frecuencia 2 KHz (mayor tiempo)
intervaloPuertoSerie=0;
}
//Iniciar las variables el resto de veces
//No cambio el techo_flag, porque no debo variar las interrupciones hardware.
void reset_variables()
{


data_flag=false;
y_techo_1=512;
y_suelo_1=512;
serial_flag=false;
num_muestras=1;
filtro_flag=false;
periodo_cuad=T_INICIAL; //frecuencia 2 KHz (mayor tiempo)
intervaloPuertoSerie=0;
chopear=CHOP_INICIAL; //por defecto true
cambiarValorChop=true;
//potencia por defecto
config_PWM();
config_SPI();
//TODO: comporobar que TODOS los valores son los adecuados
//config_chop();
}
```

```
/*
    Configurar el Timer0 en modo CTC con interrupcion en OC0B para marcar el
        instante de conversion
 */
 unsigned char tim_aux;
void config_timer0(){
  TCCR0B&=0xF8; //Parar timer
  TIFR0|=0x07;//Borrar interrupciones pendientes TIFR0
  TCNT0  = 0x00;          //Reset Timer Count
  TIFR0  = 0x00;         //Timer2 INT Flag Reg: Clear Timer Overflow Flag
  TIMSK0 = 0x04;         //Timer2 INT Reg: Output Compare B Interrupt Enable
  if (chopear) TCCR0A = 0x42;        //Timer2 Control Reg A: Modo CTC} Toggle
      on OCA
  else TCCR0A = 0x82;    //Timer2 Control Reg A: Modo CTC} clear on OCA
  //TCCR0B = 0x05;         //Timer2 Control Reg B: Timer Prescaler set to 256
      (16 us)
  TCCR0B = 0x04;
  OCR0A=periodo_cuad;    // Establecer la cima
  tim_aux=periodo_cuad>>2;
  OCR0B=periodo_cuad-tim_aux; // Establecer el valor del OC
}
/*
    Configuracion TIMER1 DE 16 bits, con conteo hasta 800
 */
 unsigned int pwm1_value;
 unsigned int pwm2_value;
void config_PWM(){
  TCCR1A = 0x00; // sets timer control bits to PWM Phase and Frequency Correct
      mode
  TCCR1B = 0x11; // sets timer control bits to Prescaler N = 1
  ICR1 = 800; // 10 Khz (subida + bajada)
  pwm1_value=P_INICIAL;
  pwm2_value=Q_INICIAL;
  generar_PWM();
}
void config_pines(){
    //trx
  pinMode(trx_pin,OUTPUT);
  //pwm
  pinMode(pwm_trx_pin,OUTPUT);
  pinMode(pwm_rcx_pin,OUTPUT);
  //spi
  pinMode(SCK_pin,OUTPUT);
  pinMode(MOSI_pin,OUTPUT);
  pinMode(SS1_pin,OUTPUT);
  pinMode(SS2_pin,OUTPUT);
  pinMode(SS3_pin,OUTPUT);
  //conversor AD
  pinMode(A0,INPUT);
}
/*
    Configuracion AD, sin iniciar el modulo
 */
void config_AD(){
  ADMUX=0x40; //REF AVcc, ajuste dcha, canal 0
  DIDR0=0x3E; //Todas las entradas salvo 0 deshabilitadas (ahorro consumo)
  ADCSRA=END_ADD_CONVERSION_VALUE; //Desactivado, prescaler maxima frec
}
```

```
/*
    Configuracion SPI modo0 y ganancias x1
 */
 unsigned char g1_value;
 unsigned char g2_value;
 unsigned char g3_value;
void config_SPI(void){

g1_value=G1_INICIAL;
g2_value=G2_INICIAL;
g3_value=G3_INICIAL;
  digitalWrite(SS1_pin,HIGH); // PGAS no seleccionados
  digitalWrite(SS2_pin,HIGH); //
  digitalWrite(SS3_pin,HIGH); //
  SPCR=0x53;
  /*
  SPIE 0 SPI Interrupts not enabled
   SPE 1  SPI Port Enabled
   DORD 0 MSB First
   MSTR 1 Master
   CPOL,CPHA 0 SPI Mode 0
   SPR1,SPR0 1 SCK=Fosc/128
   */
  SPSR&=0xFE;
  byte clr;//Por si hubiese alguna transf. pendiente
  clr=SPSR;
  clr=SPDR;
  //Poner a los PGAs en ganancia x1

  datoAGanancia(g1_value);
  transferirPGA(1,intr_byte,data_byte);
  datoAGanancia(g2_value);
  transferirPGA(2,intr_byte,data_byte);
  datoAGanancia(g3_value);
  transferirPGA(3,intr_byte,data_byte);
}

/*
    Rutina de envio de datos a los PWM de 16 bits
 */
void generar_PWM(){
  analogWrite(pwm_trx_pin,pwm1_value);
  analogWrite(pwm_rcx_pin,pwm2_value);
}
/*
    Rutina de envio de datos a los PGAs mediante SPI
 */
void transferirPGA(unsigned char quePGA,unsigned char dato1,unsigned char
    dato0){
  //Activar pin SS
  switch (quePGA){
  case 1: digitalWrite(SS1_pin,LOW); //inicio transmision
          break;
  case 2: digitalWrite(SS2_pin,LOW); //inicio transmision
          break;
  case 3: digitalWrite(SS2_pin,LOW); //inicio transmision
          break;
```

```
  default:break;
  }
  //Transmitir primer byte
  SPDR=dato1;
  while(((SPSR>>6)&&0x1)!=0x1){
  };
  //Transmitir segudno byte
  SPDR=dato0;
  while(((SPSR>>6)&&0x1)!=0x1){
  };
  //Desactivar pin SS
  switch (quePGA){
  case 1: digitalWrite(SS1_pin,HIGH); //fin transmision
          break;
  case 2: digitalWrite(SS2_pin,HIGH); //fin transmision
          break;
  case 3: digitalWrite(SS2_pin,HIGH); //fin transmision
          break;
  default: break;
  }
}

/*
Parseo de comandos de 4 bytes en el formato
 #P0400  PWM1 0400  0-800
 #Q0410  PWM2 0410  0-800
 #A00XX  PGA1 XX     1/4/5/8/10/16/32
 #B00XX  PGA2 XX     1/4/5/8/10/16/32
 #C00XX  PGA3 XX     1/4/5/8/10/16/32
 #C0000  NO CHOP    0 NO CHOP !=0 CHOP
 #D0000  FREC.CHOP  125KHZ/VALOR (17-->1.2KHz)
 #F000X  Filtrado 0 desactivar/ otro valor activar
 #M00XX  Intervalo Pto serie 0/enviar valor XXXX/ enviar un valor cada XXXX
    calculos
 #RXXXX  Reset
*/
int val;
unsigned char queChar;
char aux[4];

void comprobarPuertoSerie(){

  while(Serial.available()>5){ //Recepcion de 6 bytes
    queChar=Serial.read(); //Buscar token #
    while (queChar!=0x23){
      queChar=Serial.read();
        }
   // Serial.println("Leido #");
  //Captura del token
    switch (queChar=Serial.read()){
    case -1:
      break; //ERROR
    case 67: //ACCION PARA "C"
      Serial.print("ACK C/");
      Serial.readBytes(aux,4);
      chopear=atoi(aux);
      if (chopear) Serial.println("1");
        else Serial.println("0");
      config_chop();
```

```
      break;
      case 68: //ACCION PARA "D"
570         Serial.readBytes(aux,4);
      periodo_cuad=atoi(aux);
      Serial.print("ACK D/");
      Serial.println(periodo_cuad);
      config_timer0();
      //reset_valores();
      break;
          case 70: //ACCION PARA "F"
          Serial.readBytes(aux,4);
      val=atoi(aux);
580   if (val!=0)
      filtro_flag=true;
      else filtro_flag=false;
      Serial.print("ACK F/");
      Serial.println(aux);
          break;
    case 77: //ACCION PARA "M"
      Serial.readBytes(aux,4);
      intervaloPuertoSerie=atoi(aux)<<1;
      num_muestras=0;
590    if (intervaloPuertoSerie==0) { //Enviar dato suelto, desactivar envio de
         dato constante
                      //  Serial.println("Dato bajo demanda:");
                        Serial.print('T');
                        Serial.print(y_techo);
                        Serial.print('S');
                        Serial.println(y_suelo);
                        serial_flag=false;

                                                }
                  else{
600                   Serial.print("ACK M/");
                      Serial.println(intervaloPuertoSerie);
                      serial_flag=true;
                    }
      break;
    case 80: //ACCION PARA "P"
      Serial.print("ACK P/");
      Serial.readBytes(aux,4);
      pwm1_value=atoi(aux);
      Serial.println(pwm1_value);
610   analogWrite(pwm_trx_pin,pwm1_value);
      break;
    case 81: //ACCION PARA "Q"
      Serial.print("ACK Q/");
      Serial.readBytes(aux,4);
      pwm2_value=atoi(aux);
      Serial.println(pwm2_value);
      analogWrite(pwm_rcx_pin,pwm2_value);
      break;
    case 82: //ACCION PARA "R"
620   Serial.println("ACK RESET");
      Serial.readBytes(aux,4);
      reinicio_total();
      break;

      case 84: //ACCION PARA "G1" con val
```

```
      Serial.readBytes(aux,4);
      Serial.print("ACK G1/");
      g1_value=atoi(aux);
      val=g1_value;
      Serial.println(val);
      datoAGanancia(val);
      transferirPGA(1,intr_byte,data_byte);
      break;
        case 85: //ACCION PARA "G2" con val
      Serial.readBytes(aux,4);
      Serial.print("ACK G2/");
      g2_value=atoi(aux);
      val=g2_value;
      Serial.println(val);
      datoAGanancia(val);
      transferirPGA(2,intr_byte,data_byte);
      break;
      case 86: //ACCION PARA "G1" con val
      Serial.readBytes(aux,4);
      Serial.print("ACK G3/");
      g3_value=atoi(aux);
      val=g3_value;
      Serial.println(val);
      datoAGanancia(val);
      transferirPGA(3,intr_byte,data_byte);
      break;
          case 87: //ACCION para devolver la informacion de la config actual
//Formato: INFO chopear/periodo_cuad/intervaloPuertoSerie/pwm1_value/
    pwm2_value/g1_value/g2_value/g3_value
          Serial.print("INFO ");//+(unsigned int)chopear);
          Serial.print(chopear);
          Serial.print("/");
          Serial.print(periodo_cuad);
          Serial.print("/");
          Serial.print(intervaloPuertoSerie);
          Serial.print("/");
          Serial.print(pwm1_value);
          Serial.print("/");
          Serial.print(pwm2_value);
          Serial.print("/");
          Serial.print(g1_value);
          Serial.print("/");
          Serial.print(g2_value);
          Serial.print("/");
          Serial.print(g3_value);
          //TODO: filtro ON/OFF
          Serial.print("/");
          Serial.println(filtro_flag?"1":"0");
          break;
     default: //Si no sabemos que comando es
      Serial.readBytes(aux,4);
      Serial.print("ERR/");
      Serial.print(queChar);
      Serial.println(aux);
      break;
    }
  }
}
```

```
unsigned int timb_val;
void config_chop(void){
  timb_val=TCCR0B; //Guardar configuracion
  TCCR0B&=0xF8; //Parar timer
  TIFR0|=0x07;//Borrar interrupciones pendientes TIFR0
  TCNT0  = 0x00;           //Reset Timer Count
  if (chopear) TCCR0A = 0x42;         //Timer2 Control Reg A: Modo CTC} Toggle on OCA
  else TCCR0A = 0x82;    //Timer2 Control Reg A: Modo CTC} clear on OCA

  //ADCSRA=0X1F;//Desactivar ADC, limpiar interrupcion pendiente
  data_flag=false;
  convertido=0;

  prom=0;
  //techo_flag=true;
  num_muestras=0;

  TCCR0B=timb_val; //volver a iniciar timer
}

void datoAGanancia(int val){
       switch(val){
       case 1: default: data_byte=0x0;
                break;
       case 2: data_byte=0x1;
                break;
       case 4: data_byte=0x2;
                break;
       case 5: data_byte=0x3;
                break;
       case 8: data_byte=0x4;
                break;
       case 10: data_byte=0x5;
                break;
       case 16: data_byte=0x6;
                break;
       case 32: data_byte=0x7;
                break;
             }

}

void reinicio_total(void){
  reset_variables();
}
void dormir(void){
  set_sleep_mode(SLEEP_MODE_IDLE);    // sleep mode is set here
  sleep_enable();  //Activar bit seguridad
  sleep_mode();     //Modo resposo activado
  sleep_disable();  //tras reinicio desactivamos bit seguridad

}
```

5. Código de Matlab

Pese a que en la versión final los programas se hicieron en .NET, es perfectamente posible controlar el Arduino con cualquier software que pueda manejar un puerto serie. Aquí se adjunta un ejemplo con Matlab, que no llegó a ser la versión final porque no me parecía lo suficientemente "portable" respecto a la alternativa de c#.

El programa de Matlab se divide en dos ficheros que se adjuntan en la version digital, uno de ellos es un .fig, el interfaz gráfico de GUIDE, y el otro es el .m que es el que se ejecuta y realiza las tareas. El código de este último se añade a continuación.

```
function varargout = Medidor(varargin)
% Medidor MATLAB code for Medidor.fig
%      Medidor, by itself, creates a new Medidor or raises the existing
%      singleton*.
%
%      H = Medidor returns the handle to a new Medidor or the handle to
%      the existing singleton*.
%
%      Medidor('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in Medidor.M with the given input arguments.
%
%      Medidor('Property','Value',...) creates a new Medidor or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Medidor_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Medidor_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Medidor

% Last Modified by GUIDE v2.5 09-Jun-2013 18:23:54

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Medidor_OpeningFcn, ...
                   'gui_OutputFcn',  @Medidor_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT



%--- Executes just before Medidor is made visible.
```

```
function Medidor_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Medidor (see VARARGIN)
%mi codigo inicio
serialPorts = instrhwinfo('serial');
nPorts = length(serialPorts.SerialPorts);
set(handles.menu_puertos, 'String', ...
    [{'Elige un puerto'} ; serialPorts.SerialPorts ]);
set(handles.menu_puertos, 'Value', nPorts+1);

set(handles.caja_historial,'Value',0);
set(handles.caja_historial,'String','');

%Estimacion inicial
set(handles.grafico1,'Visible','on');
set(handles.grafico1,'UserData',0);

hold(handles.grafico1,'on');
grid(handles.grafico1,'on');
title(handles.grafico1, 'óAtenuacin');
hold(handles.grafico3,'on');
grid(handles.grafico3,'on');
title(handles.grafico3, 'Histograma óAtenuacin');

handles.vSize=15;
handles.datos=zeros(handles.vSize,2);
handles.aten=zeros(handles.vSize,1);
handles.tmpSize=1;
handles.tmpInd=1;
handles.tmpData=zeros(handles.tmpSize,2);
handles.fichInd=1;
handles.fichSize=1001;
handles.histData=zeros(handles.fichSize,1);
handles.output = hObject;

handles.volcadoFichero=false;
handles.barriendo=false;
handles.waitData=true;
% Choose default command line output for Medidor
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Medidor wait for user response (see UIRESUME)
% uiwait(handles.figure1);


%--- Outputs from this function are returned to the command line.
function varargout = Medidor_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

```
%--- Executes during object creation, after setting all properties.
function menu_puertos_CreateFcn(hObject, eventdata, handles)
% hObject    handle to menu_puertos (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function menu_puertos_Callback(hObject, eventdata, handles)

%--- Executes on button press in boton_conecta.
function boton_conecta_Callback(hObject, eventdata, handles)
% hObject    handle to boton_conecta (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if strcmp(get(hObject,'String'),'Conectar')
    com=controladorSerie(0,'',handles);
    escribeTextoCaja(['Conectado @ ' datestr(now,'HH:MM:SS') ':' com],handles)
        ;
    set(hObject,'String','Desconectar');
  % salida=controladorSerie(2,'',handles); %get String inicial
  else com=controladorSerie(1,'',handles);
     set(hObject,'String','Conectar');
     escribeTextoCaja(['Desconectado @ ' datestr(now,'HH:MM:SS') ':' com],
         handles);
end;

guidata(hObject, handles);


function received_data(hObject,eventdata,figure)
handles=guidata(figure);
s1=controladorSerie(2,'',handles);
S= strtrim(s1);
if strcmp(S(1),'T')
  % Descomentar para tener echo de los datos recibidos en el monitor
 if get(handles.mostrar_datos_check,'Value')
    escribeTextoCaja(['Dato @ ' datestr(now,'HH:MM:SS') ': ' S],handles);
 end
A = sscanf(S,'%*c %d %*c %d');
if A(1)<A(2)
    tmp=A(1);
    A(1)=A(2);
    A(2)=tmp;
end
handles.tmpData(handles.tmpInd,1)=A(1);
handles.tmpData(handles.tmpInd,2)=A(2);
set(handles.caja_res,'String',A(1)-A(2));
%[y1,y2]=estimaRecepcion((A(1)-A(2)),figure);
%set(handles.caja_res_uw,'String',[num2str(y1) ' uW']);
%set(handles.caja_res_dbm,'String',[num2str(y2) ' dBm']);

%escribeTextoCaja(['Escrito tmp @ ' datestr(now,'HH:MM:SS') ': ' handles.
```

```
    tmpInd],handles);
if (handles.tmpInd==handles.tmpSize)
    %handles.tmpInd
    %handles.tmpSize
    handles.tmpInd=1;
     %Abrir y guardar fichero
    if handles.volcadoFichero
170             if (handles.fichInd==1)
                handles.fidName=strcat(get(handles.edit_captura,'String'),datestr(
                    now,'_dd_HH-MM'),'.txt');
                handles.fid=fopen(handles.fidName,'w');
                if handles.fid~=-1
                    escribeTextoCaja(['Inicio fichero @ ' datestr(now,'HH:MM:SS')
                        ': ' handles.fidName],handles);
                else
                    warning(errFich,'Error al crear fichero');
                    escribeTextoCaja(['Problema con fichero @ ' datestr(now,'HH:MM
                        :SS') ': ' handles.fidName],handles);
                end
                end
180     end
    %Representacion ágrfica

    if (handles.fichInd<=handles.vSize)
        %Adquisicion datos

        handles.datos(handles.fichInd,1)=handles.tmpData(1);
        handles.datos(handles.fichInd,2)=handles.tmpData(2);
        handles.aten(handles.fichInd)=(handles.datos(handles.fichInd,1)-
            handles.datos(handles.fichInd,2));
        if handles.volcadoFichero
190         fprintf(handles.fid,'%d\n',handles.aten(handles.fichInd));
        end
        handles.histData(handles.fichInd)=handles.aten(handles.fichInd);
        %Grafico
        hold(handles.grafico1,'on');
        grid(handles.grafico1,'on');
        xlim(handles.grafico1,[1 handles.vSize+5]);



200     plot(1:handles.fichInd,handles.datos(1:handles.fichInd,1),'b.','Parent
            ',handles.grafico1,'LineWidth',0.1,'MarkerSize',6);
        plot(1:handles.fichInd,handles.datos(1:handles.fichInd,2),'b.','Parent
            ',handles.grafico1,'LineWidth',0.1,'MarkerSize',6);
        plot(1:handles.fichInd,handles.aten(1:handles.fichInd),'r-x','Parent',
            handles.grafico1,'LineWidth',2,'MarkerSize',10);



        %hold(handles.grafico1,'off');
        hist(handles.histData(1:handles.fichInd),20,'Parent',handles.grafico3)
            ;
        handles.fichInd=handles.fichInd+1;

210 else    %Adquisicion datos
        tmp=zeros(handles.vSize,3);
        tmp(1:handles.vSize-1,1)=handles.datos(2:handles.vSize,1);
        tmp(1:handles.vSize-1,2)=handles.datos(2:handles.vSize,2);
```

```
        tmp(1:handles.vSize-1,3)=handles.aten(2:handles.vSize);
        handles.datos(handles.vSize,1)=mean(handles.tmpData(:,1));
        handles.datos(handles.vSize,2)=mean(handles.tmpData(:,2));
        handles.aten(handles.vSize)=handles.datos(handles.vSize,1)-handles.
            datos(handles.vSize,2);
        if handles.volcadoFichero
        fprintf(handles.fid,'%d\n',handles.aten(handles.vSize));
        end
        handles.histData(handles.fichInd)=handles.aten(handles.vSize);

        handles.datos(1:handles.vSize-1,:)=tmp(1:handles.vSize-1,1:2);
        handles.aten(1:handles.vSize-1)=tmp(1:handles.vSize-1,3);
        %Grfico

             hold(handles.grafico1,'on');
             grid(handles.grafico1,'on');
         xlim(handles.grafico1,[handles.fichInd-handles.vSize-1 handles.fichInd
             +5]);

         handles.pl1=plot((handles.fichInd-handles.vSize+1):handles.fichInd,
             handles.datos(1:handles.vSize,1),'b.','Parent',handles.grafico1,'
             LineWidth',0.1,'MarkerSize',6);
         handles.pl2=plot((handles.fichInd-handles.vSize+1):handles.fichInd,
             handles.datos(1:handles.vSize,2),'b.','Parent',handles.grafico1,'
             LineWidth',0.1,'MarkerSize',6);
         handles.pl3=plot((handles.fichInd-handles.vSize+1):handles.fichInd,
             handles.aten(1:handles.vSize),'r-x','Parent',handles.grafico1,'
             LineWidth',2,'MarkerSize',10);
         hist(handles.histData(2:handles.fichInd),10,'Parent',handles.grafico3)
             ;

         hold(handles.grafico1,'off');

         handles.fichInd=handles.fichInd+1;
end
if (handles.fichInd==handles.fichSize)
    handles.fichInd=1; %reset contador
    if handles.volcadoFichero
    fclose(handles.fid);
    end
    %handles.datos=zeros(handles.vSize,2);
    %handles.aten=zeros(handles.vSize,1);
    cla(handles.grafico1);
    cla(handles.grafico3);

    escribeTextoCaja(['Escrito fichero @ ' datestr(now,'HH:MM:SS') ': '
        handles.fidName],handles);
end
else     handles.tmpInd=handles.tmpInd+1;
end




else escribeTextoCaja(['Rec @ ' datestr(now,'HH:MM:SS') ': ' S],handles);
end
if (handles.waitData==true)
    handles.waitData=false;
end
```

```
guidata(figure,handles);


%END USER CODE



function caja_historial_Callback(hObject, eventdata, handles)
%hObject    handle to caja_historial (see GCBO)
%eventdata  reserved - to be defined in a future version of MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'String') returns contents of caja_historial as text
%        str2double(get(hObject,'String')) returns contents of caja_historial
    as a double


%--- Executes during object creation, after setting all properties.
function caja_historial_CreateFcn(hObject, eventdata, handles)
%hObject    handle to caja_historial (see GCBO)
%eventdata  reserved - to be defined in a future version of MATLAB
%handles    empty - handles not created until after all CreateFcns called

%Hint: edit controls usually have a white background on Windows.
%      See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


%--- Executes during object creation, after setting all properties.
function edit_p_CreateFcn(hObject, eventdata, handles)
%hObject    handle to edit_p (see GCBO)
%eventdata  reserved - to be defined in a future version of MATLAB
%handles    empty - handles not created until after all CreateFcns called
set(hObject,'String','230');
%Aadido para el control del valor
set(hObject, 'UserData','230');
%Hint: edit controls usually have a white background on Windows.
%      See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



%--- Executes on button press in ayuda_p.
function ayuda_p_Callback(hObject, eventdata, handles)
%hObject    handle to ayuda_p (see GCBO)
%eventdata  reserved - to be defined in a future version of MATLAB
%handles    structure with handles and user data (see GUIDATA)
set(handles.ayuda_edit, 'String','');
%--- Executes on button press in boton_g.
function boton_g_Callback(hObject, eventdata, handles)
%hObject    handle to boton_g (see GCBO)
%eventdata  reserved - to be defined in a future version of MATLAB
%handles    structure with handles and user data (see GUIDATA)
```

```
320
%--- Executes on button press in ayuda_g.
function ayuda_g_Callback(hObject, eventdata, handles)
% hObject    handle to ayuda_g (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.ayuda_edit, 'String','');


330
%--- Executes during object creation, after setting all properties.
function edit_f_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_f (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
set(hObject,'String','256');
%Aadido para el control del valor
set(hObject, 'UserData','256');

340 % Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


%--- Executes on button press in enviar_f.
function enviar_f_Callback(hObject, eventdata, handles)
% hObject    handle to enviar_f (see GCBO)
350 % eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



%--- Executes on button press in ayuda_f.
function ayuda_f_Callback(hObject, eventdata, handles)
% hObject    handle to ayuda_f (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
360 set(handles.ayuda_edit, 'String','');



%--- Executes during object creation, after setting all properties.
function edit_o_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_o (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
set(hObject,'String','0');
370 %Aadido para el control del valor
set(hObject, 'UserData','0');
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end


%--- Executes on button press in enviar_o.
function enviar_o_Callback(hObject, eventdata, handles)
% hObject    handle to enviar_o (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


%--- Executes on button press in ayuda_o.
function ayuda_o_Callback(hObject, eventdata, handles)
% hObject    handle to ayuda_o (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.ayuda_edit, 'String','');


%--- Executes on button press in enviar_c.
function enviar_c_Callback(hObject, eventdata, handles)
% hObject    handle to enviar_c (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


%--- Executes on button press in ayuda_c.
function ayuda_c_Callback(hObject, eventdata, handles)
% hObject    handle to ayuda_c (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.ayuda_edit, 'String','');



%--- Executes during object creation, after setting all properties.
function menu_g1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to menu_g1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
set(hObject,'String',[{'1'};{'2'};{'4'};{'5'};{'8'};{'10'};{'16'};{'32'}]);
set(hObject,'Value',1);
%BUSCAME
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end



%--- Executes during object creation, after setting all properties.
function menu_chop_CreateFcn(hObject, eventdata, handles)
% hObject    handle to menu_chop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
set(hObject,'String',[{'Si'};{'No'}]);
set(hObject,'Value',1);
```

```
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

%--- Executes on button press in boton_grafico.
function boton_grafico_Callback(hObject, eventdata, handles)
% hObject    handle to boton_grafico (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if (strcmp(get(handles.grafico1,'Visible'),'on'))
                set(handles.grafico1,'Visible','off');
else set(handles.grafico1,'Visible','on');
end;

%--- Executes on button press in boton_refresco.
function boton_refresco_Callback(hObject, eventdata, handles)
% hObject    handle to boton_refresco (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
serialPorts = instrhwinfo('serial');
nPorts = length(serialPorts.SerialPorts);
set(handles.menu_puertos, 'String', ...
    [{'Elige un puerto'} ; serialPorts.SerialPorts ]);
set(handles.menu_puertos, 'Value', 1);

%--- Executes on button press in boton_estimar.
function boton_estimar_Callback(hObject, eventdata, handles)
% hObject    handle to boton_estimar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



%--- Executes on button press in boton_reset_ard.
function boton_reset_ard_Callback(hObject, eventdata, handles)
% hObject    handle to boton_reset_ard (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
TxText = '#R0000';
controladorSerie(3,TxText,handles);
escribeTextoCaja(['Enviado @ ' datestr(now,'HH:MM:SS') ': RESET'],handles);
%Valores por defecto en las cajas
set(handles.edit_p,'String','230');
set(handles.edit_f,'String','64');
set(handles.edit_o,'String','0');
set(handles.menu_g1,'Value',1);
set(handles.menu_g2,'Value',1);
set(handles.menu_g3,'Value',1);
set(handles.menu_chop,'Value',1);

    cla(handles.grafico1,'reset');
    cla(handles.grafico3,'reset');
handles.fichInd=1;
handles.tmpInd=1;
```

```
%Aadir flush fichero
guidata(hObject, handles);

%--- Executes on button press in boton_historial.
function boton_historial_Callback(hObject, eventdata, handles)
% hObject    handle to boton_historial (see GCBO)
% eventdata  reserved - dto be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.caja_historial,'Value',0);
set(handles.caja_historial,'String','');
cla(handles.grafico1,'reset');
cla(handles.grafico3,'reset');
handles.fichInd=1;
handles.tmpInd=1;
%Aadir flush fichero

guidata(hObject, handles);



function escribeTextoCaja(arg,handles)
valor=get(handles.caja_historial, 'Value');
set(handles.caja_historial, 'Value', valor+1);
currList = get(handles.caja_historial, 'String');
newList =[currList; {arg}];
set(handles.caja_historial, 'String',newList);




function out=controladorSerie(arg,cadena,handles)
 persistent s puerto_com
 switch arg
 case 0
     serList = get(handles.menu_puertos,'String');
     serPortn =get(handles.menu_puertos,'Value');
     puerto_com = serList{serPortn};
     delete(instrfind({'Port'},{puerto_com}));
     s = serial(puerto_com);
     s.BaudRate=115200;
     s.TimeOut=10;
     s.BytesAvailableFcnMode='terminator';
     s.BytesAvailableFcn=@(src, event)received_data(src, event, handles.
         figure1);
     s.Terminator=13;
     fopen(s);
     out=puerto_com;
 case 1
     fclose(s);
     delete(s);
     out=puerto_com;
 case 2
    out=fscanf(s);
 case 3
    fprintf(s, cadena);
 case 4
     out=s.BytesAvailable();
 end
%guidata(s, handles);
```

```
550

%--- Executes on button press in enviar_m.
function enviar_m_Callback(hObject, eventdata, handles)
% hObject    handle to enviar_m (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% salida=controladorSerie(2,'',handles)
% while ~strcmp(salida(1:3),'Rec') %Esperar a cadena de recibido OK
560 % salida=controladorSerie(2,'',handles);
% escribeTextoCaja(['Recibido @ ' datestr(now,'HH:MM:SS') ': ' salida],handles
    );
%end


%--- Executes on button press in ayuda_m.
function ayuda_m_Callback(hObject, eventdata, handles)
% hObject    handle to ayuda_m (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
570 set(handles.ayuda_edit, 'String','');




%--- Executes during object creation, after setting all properties.
function edit_intervalo_serie_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_intervalo_serie (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
580 set(hObject,'String','0');
%Aadido para el control del valor
set(hObject, 'UserData','0');
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


590 %--- Executes on button press in mostrar_historial.
function mostrar_historial_Callback(hObject, eventdata, handles)
% hObject    handle to mostrar_historial (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if (strcmp(get(handles.caja_historial,'Visible'),'on'))
                set(handles.caja_historial,'Visible','off');
else set(handles.caja_historial,'Visible','on');
end;

600


%--- Executes during object creation, after setting all properties.
function edit_captura_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_captura (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



%--- Executes on selection change in menu_g1.
function menu_g1_Callback(hObject, eventdata, handles)
% hObject    handle to menu_g1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns menu_g1 contents as
    cell array
%        contents{get(hObject,'Value')} returns selected item from menu_g1
switch get(handles.menu_g1, 'Value')
    case 1
        aux='0001';
    case 2
        aux='0002';
    case 3
        aux='0004';
    case 4
        aux='0005';
    case 5
        aux='0008';
    case 6
        aux='0010';
    case 7
        aux='0016';
    case 8
        aux='0032';
    otherwise
        warning('Debug g');
end
TxText =strcat('#T', aux);
controladorSerie(3,TxText,handles);
escribeTextoCaja(['Enviado @ ' datestr(now,'HH:MM:SS') ': ' TxText],handles);


function  edit_p_Callback(hObject, eventdata, handles)
% hObject    handle to edit_p (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_p as text
%        str2double(get(hObject,'String')) returns contents of edit_p as a
    double

aux=get(handles.edit_p, 'String');
%TODO: poner el valor anterior, guardado cuando es correcto en userData...

num=str2double(aux);
%Si no es un únmero restaurar el valor anterior MOD
if isnan(str2double(aux))
```

```
    set(handles.edit_p, 'String',get(handles.edit_p,'UserData'));
    errordlg('ingrese un valor entre 80 y 500','error','modal')
    return
end
if (num<80)
   set(handles.edit_p, 'String','0080');
    errordlg('El valor ímnimo es 80','error','modal')
    return
else if (num>500)
    set(handles.edit_p, 'String','0500');
    errordlg('El valor ámximo es 500','error','modal')
    return
    end
end

%Guardar el valor que se va a enviar MOD
set(handles.edit_p, 'UserData',num2str(num));

if length(aux)==2 aux=strcat('00',aux);
else if length(aux)==3 aux=strcat('0',aux);
    end
end
TxText =strcat('#P', aux);
if (str2double(aux)<40 || str2double(aux)>500) warning('Valor de P no ávlido, ingrese un valor entre 70 y 500')
else
controladorSerie(3,TxText,handles);
escribeTextoCaja(['Enviado @ ' datestr(now,'HH:MM:SS') ': ' TxText],handles);
%Actualizar estimacion
x=str2double(get(handles.edit_p,'String'));
set(handles.est1, 'String',estimaEmision(x));
end


function edit_f_Callback(hObject, eventdata, handles)
%hObject    handle to edit_f (see GCBO)
%eventdata  reserved - to be defined in a future version of MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'String') returns contents of edit_f as text
%       str2double(get(hObject,'String')) returns contents of edit_f as a double
aux=get(handles.edit_f, 'String');
pwmnum=round(1/(4*33*str2double(aux)*10^-6));
aux=num2str(pwmnum);
resul=round(1/(4*33*pwmnum*10^-6));
%num=round(31000*num);
%set(handles.edit_f, 'String',num);
%set(handles.est2, 'String',num);
if length(aux)==2 aux=strcat('00',aux);
else if length(aux)==3 aux=strcat('0',aux);
    end
end
if (pwmnum<10 || pwmnum>255)
    set(handles.edit_f, 'String','');
else
set(handles.edit_f, 'String',num2str(resul));
TxText =strcat('#D', aux);
controladorSerie(3,TxText,handles);
```

```
escribeTextoCaja(['Enviado @ ' datestr(now,'HH:MM:SS') ': ' TxText],handles);

%resul=1./(x.*31.*10.^(-6));
%set(handles.est2, 'String',num2str(resul));
%set(handles.edit_f, 'String',num2str(resul));
end
tmp_val2=estimaSegundos(handles.figure1);
set(handles.menu_act,'String',[{[num2str(tmp_val2) ' seg']};{[num2str(3.*
    tmp_val2) ' seg']};{[num2str(12.*tmp_val2) ' seg']}]);
guidata(hObject,handles);


function edit_o_Callback(hObject, eventdata, handles)
% hObject    handle to edit_o (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_o as text
%        str2double(get(hObject,'String')) returns contents of edit_o as a
    double
aux=get(handles.edit_o, 'String');
%CAMBIADO
cad=round(str2double(aux)*800/1.25);
aux=num2str(cad);
set(handles.edit_o,'String',num2str(cad*1.25/800));
if length(aux)==1 aux=strcat('000',aux);
else
if length(aux)==2 aux=strcat('00',aux);
else if length(aux)==3 aux=strcat('0',aux);
    end
end
end
TxText =strcat('#Q', aux);
if (str2double(aux)<0 ||str2double(aux)>800) warning('Valor de offset no
    válido, ingrese un valor entre 0 y 800')
else
controladorSerie(3,TxText,handles);
escribeTextoCaja(['Enviado @ ' datestr(now,'HH:MM:SS') ': ' TxText],handles);
x=str2double(get(handles.edit_o,'String'));
resul=1.25.*x./800;
end

%--- Executes on selection change in menu_chop.
function menu_chop_Callback(hObject, eventdata, handles)
% hObject    handle to menu_chop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns menu_chop contents
    as cell array
%        contents{get(hObject,'Value')} returns selected item from menu_chop
if get(handles.menu_chop,'Value')==1
    aux='0001';
else aux='0000';
end
TxText =strcat('#C', aux);
controladorSerie(3,TxText,handles);
escribeTextoCaja(['Enviado @ ' datestr(now,'HH:MM:SS') ': ' TxText],handles);
```

```
%--- Executes on selection change in edit_intervalo_serie.
function edit_intervalo_serie_Callback(hObject, eventdata, handles)
% hObject    handle to edit_intervalo_serie (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%TODO: TRANSFORMAR A 4 DIGITOS
aux=get(handles.edit_intervalo_serie, 'String');
val= str2double(aux);
frec=str2double(get(handles.edit_f,'String'));
%Actualizo edit_intervalo_ms
set(handles.edit_intervalo_ms,'String',num2str(round(1000*val/frec(1))));


if length(aux)==1 aux=strcat('000',aux);
else
if length(aux)==2 aux=strcat('00',aux);
else if length(aux)==3 aux=strcat('0',aux);
    end
end
end

if (val==0)
    set(handles.bCapturaDato,'Visible','on');
else set(handles.bCapturaDato,'Visible','off');
end
TxText =strcat('#M', aux);
controladorSerie(3, TxText,handles);
escribeTextoCaja(['Enviado @ ' datestr(now,'HH:MM:SS') ': ' TxText],handles);
%Cambiar valor sincronismo
%tmp_val=estimaSegundos(handles.figure1);
%set(handles.menu_act,'String',[{[num2str(tmp_val) ' seg']};{[num2str(3*
    tmp_val) ' seg']};{[num2str(12*tmp_val) ' seg']}]);
guidata(hObject,handles);
% Hints: contents = cellstr(get(hObject,'String')) returns
    edit_intervalo_serie contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
    edit_intervalo_serie


function edit_captura_Callback(hObject, eventdata, handles)
% hObject    handle to edit_captura (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_captura as text
%        str2double(get(hObject,'String')) returns contents of edit_captura as
     a double



function edit10_Callback(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit10 as text
%        str2double(get(hObject,'String')) returns contents of edit10 as a
    double
```

```
830
%--- Executes during object creation, after setting all properties.
function edit10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
840     set(hObject,'BackgroundColor','white');
end


%--- Executes on button press in boton_limite.
function boton_limite_Callback(hObject, eventdata, handles)
% hObject    handle to boton_limite (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

850
%--- Executes on button press in limite_ayuda.
function limite_ayuda_Callback(hObject, eventdata, handles)
% hObject    handle to limite_ayuda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.ayuda_edit, 'String','');




860
% hObject    handle to bprueba (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

870 %START USER CODE
% Necessary to provide this function to prevent timer callback
% from causing an error after GUI code stops executing.
% Before exiting, if the timer is running, stop it.

%END USER CODE

% Hint: delete(hObject) closes the figure
delete(hObject);

880
%--- Executes on selection change in menu_act.
function menu_act_Callback(hObject, eventdata, handles)
% hObject    handle to menu_act (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = cellstr(get(hObject,'String')) returns menu_act contents
```

```
    as cell array
%       contents{get(hObject,'Value')} returns selected item from menu_act
switch get(hObject,'Value')
    case 1
        handles.tmpSize=1;
        handles.tmpInd=1;
        handles.tmpData=zeros(handles.tmpSize,2);
        handles.fichInd=1;
    case 2
        handles.tmpSize=3;
        handles.tmpInd=1;
        handles.tmpData=zeros(handles.tmpSize,2);
        handles.fichInd=1;
    case 3
        handles.tmpSize=12;
        handles.tmpInd=1;
        handles.tmpData=zeros(handles.tmpSize,2);
        handles.fichInd=1;
end
guidata(hObject,handles);
%--- Executes during object creation, after setting all properties.
function menu_act_CreateFcn(hObject, eventdata, handles)
% hObject    handle to menu_act (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
set(hObject,'String',[{[num2str(round(4096*1/819)) ' seg']};{[num2str(round
    (3.*4096*1/819)) ' seg']};{[num2str(round(12.*4096*1/819)) ' seg']}]);
set(hObject,'Value',1);
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function y=estimaEmision(x)
y=-4e-8.*x.^5 + 5e-5.*x.^4 - 0.0189.*x.^3 + 3.9113.*x.^2 - 359.58.*x + 11755;
y=round(1000.*log10(y./1e6))./100;


%Funcin que estimaba el valor de potencia un tanto "de la patilla", de
%manera que se elimina.
% function [y1,y2]=estimaRecepcion(x,figure)
% misHandles=guidata(figure);
% selected1=get(misHandles.menu_g1,'Value');
% vector_g1=str2double(get(misHandles.menu_g1,'String'));
% selected2=get(misHandles.menu_g2,'Value');
% vector_g2=str2double(get(misHandles.menu_g2,'String'));
% selected3=get(misHandles.menu_g3,'Value');
% vector_g3=str2double(get(misHandles.menu_g3,'String'));
% tmp=x/(vector_g1(selected1).*vector_g2(selected2).*vector_g3(selected3));
% y1=6.1322.*tmp - 13.012;
% y2=round(1000.*log10(y1./1e6))./100;
% y1=round(y1);

function y=estimaSegundos(figure)
misHandles=guidata(figure);
muestras=64;
```

```
frec=str2double(get(misHandles.edit_f,'String'));
y=round(10*(1./frec).*muestras)./10;


%--- Executes on selection change in menu_g2.
function menu_g2_Callback(hObject, eventdata, handles)
% hObject    handle to menu_g2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
switch get(handles.menu_g2, 'Value')
    case 1
        aux='0001';
    case 2
        aux='0002';
    case 3
        aux='0004';
    case 4
        aux='0005';
    case 5
        aux='0008';
    case 6
        aux='0010';
    case 7
        aux='0016';
    case 8
        aux='0032';
    otherwise
        warning('Debug g');
end
TxText =strcat('#U', aux);
controladorSerie(3,TxText,handles);
escribeTextoCaja(['Enviado @ ' datestr(now,'HH:MM:SS') ': ' TxText],handles);
% Hints: contents = cellstr(get(hObject,'String')) returns menu_g2 contents as
    cell array
%        contents{get(hObject,'Value')} returns selected item from menu_g2


%--- Executes during object creation, after setting all properties.
function menu_g2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to menu_g2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
set(hObject,'String',{'1';'2';'4';'5';'8';'10';'16';'32'});
set(hObject,'Value',1);
%BUSCAME
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



%--- Executes on selection change in menu_g3.
function menu_g3_Callback(hObject, eventdata, handles)
% hObject    handle to menu_g3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
switch get(handles.menu_g3, 'Value')
    case 1
        aux='0001';
    case 2
        aux='0002';
    case 3
        aux='0004';
    case 4
        aux='0005';
    case 5
        aux='0008';
    case 6
        aux='0010';
    case 7
        aux='0016';
    case 8
        aux='0032';
    otherwise
        warning('Debug g');
end
TxText =strcat('#V', aux);
controladorSerie(3,TxText,handles);
escribeTextoCaja(['Enviado @ ' datestr(now,'HH:MM:SS') ': ' TxText],handles);
% Hints: contents = cellstr(get(hObject,'String')) returns menu_g3 contents as
    cell array
%        contents{get(hObject,'Value')} returns selected item from menu_g3


%--- Executes during object creation, after setting all properties.
function menu_g3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to menu_g3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
set(hObject,'String',[{'1'};{'2'};{'4'};{'5'};{'8'};{'10'};{'16'};{'32'}]);
set(hObject,'Value',1);
%BUSCAME
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


%--- Executes on button press in B_ascendente.
function B_ascendente_Callback(hObject, eventdata, handles)
% hObject    handle to B_ascendente (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

t2 = timer('StartDelay', 0,'Period', str2num(get(handles.fTiempoPaso,'String')
   ),'ExecutionMode', 'FixedRate');
t2.TimerFcn = {@callback_ascendente_fcn, handles.figure1};
t2.StopFcn = @(x,y)delete(t2);
t2.UserData=100;
start(t2);

%--- Executes on button press in B_descendente.
function  [handles]=B_descendente_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to B_descendente (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

t1 = timer('StartDelay', 0,'Period', str2num(get(handles.fTiempoPaso,'String'
    )),'ExecutionMode', 'FixedRate');
t1.TimerFcn = {@callback_descendente_fcn, handles.figure1};
t1.StopFcn = @(x,y)delete(t1);
t1.UserData=str2num(get(handles.fMax,'String'));
start(t1);




function callback_ascendente_fcn(obj,event,figura)
  misHandles=guidata(figura);
 set(misHandles.edit_p,'String',num2str(obj.UserData));
 guidata(figura,misHandles);
  edit_p_Callback(misHandles.edit_p,0,misHandles);
set(misHandles.emitido,'String',num2str(obj.UserData));
guidata(figura,misHandles);
obj.UserData=obj.UserData+str2num(get(misHandles.fPaso,'String'));
if obj.UserData==str2num(get(misHandles.fMax,'String'))
    stop(obj);
    destroy(obj);
end


function callback_descendente_fcn(obj,event,figura)
  misHandles=guidata(figura);
 set(misHandles.edit_p,'String',num2str(obj.UserData));
 guidata(figura,misHandles);
  edit_p_Callback(misHandles.edit_p,0,misHandles);
set(misHandles.emitido,'String',num2str(obj.UserData));
guidata(figura,misHandles);
obj.UserData=obj.UserData-str2num(get(misHandles.fPaso,'String'));
if obj.UserData==str2num(get(misHandles.fMin,'String'))
    stop(obj);
    destroy(obj);
end




%--- Executes on button press in mostrar_datos_check.
function mostrar_datos_check_Callback(hObject, eventdata, handles)
% hObject    handle to mostrar_datos_check (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mostrar_datos_check


%--- Executes on button press in volcarFichero.
function volcarFichero_Callback(hObject, eventdata, handles)
% hObject    handle to volcarFichero (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
handles.volcadoFichero=get(hObject,'Value');
% Hint: get(hObject,'Value') returns toggle state of volcarFichero


%--- Executes on button press in bCapturaDato.
function bCapturaDato_Callback(hObject, eventdata, handles)
% hObject    handle to bCapturaDato (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
TxText ='#M0000';
controladorSerie(3, TxText,handles);
escribeTextoCaja(['Enviado @ ' datestr(now,'HH:MM:SS') ': ' TxText],handles);


%--- Executes during object creation, after setting all properties.
function bCapturaDato_CreateFcn(hObject, eventdata, handles)
% hObject    handle to bCapturaDato (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns calleds
set(hObject,'Visible','on');



function edit_intervalo_ms_Callback(hObject, eventdata, handles)
% hObject    handle to edit_intervalo_ms (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Mandar valor calculado en muestras
frec=str2double(get(handles.edit_f,'String'));
num_ms=str2double(get(handles.edit_intervalo_ms,'String'));
num_muestras=num_ms(1)*frec(1)/1000;
num_muestras_str=num2str(round(num_muestras));
%Actualizar el campo de arriba
set(handles.edit_intervalo_serie,'String',num_muestras_str);


%Transformar a String
TxText ='#M';
if length(num_muestras_str)==1 num_muestras_str=strcat('000',num_muestras_str)
    ;
else
if length(num_muestras_str)==2 num_muestras_str=strcat('00',num_muestras_str);
else if length(num_muestras_str)==3 num_muestras_str=strcat('0',
    num_muestras_str);
     end
end
end

TxText=strcat(TxText,num_muestras_str);
controladorSerie(3, TxText,handles);









% Hints: get(hObject,'String') returns contents of edit_intervalo_ms as text
```

```
%        str2double(get(hObject,'String')) returns contents of
    edit_intervalo_ms as a double


%--- Executes during object creation, after setting all properties.
function edit_intervalo_ms_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_intervalo_ms (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
%Aadido para el control del valor
set(hObject, 'String','0');
%Aadido para el control del valor
set(hObject, 'UserData','0');
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function fMin_Callback(hObject, eventdata, handles)
% hObject    handle to fMin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of fMin as text
%        str2double(get(hObject,'String')) returns contents of fMin as a
    double


%--- Executes during object creation, after setting all properties.
function fMin_CreateFcn(hObject, eventdata, handles)
% hObject    handle to fMin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
set(hObject,'String','100');
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function fMax_Callback(hObject, eventdata, handles)
% hObject    handle to fMax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of fMax as text
%        str2double(get(hObject,'String')) returns contents of fMax as a
    double


%--- Executes during object creation, after setting all properties.
function fMax_CreateFcn(hObject, eventdata, handles)
```

```
    %hObject    handle to fMax (see GCBO)
    %eventdata  reserved - to be defined in a future version of MATLAB
    %handles    empty - handles not created until after all CreateFcns called
    set(hObject,'String','260');
    %Hint: edit controls usually have a white background on Windows.
1230 %       See ISPC and COMPUTER.
    if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
       defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end



    function fPaso_Callback(hObject, eventdata, handles)
    %hObject    handle to fPaso (see GCBO)
    %eventdata  reserved - to be defined in a future version of MATLAB
1240 %handles    structure with handles and user data (see GUIDATA)

    %Hints: get(hObject,'String') returns contents of fPaso as text
    %       str2double(get(hObject,'String')) returns contents of fPaso as a
       double


    %--- Executes during object creation, after setting all properties.
    function fPaso_CreateFcn(hObject, eventdata, handles)
    %hObject    handle to fPaso (see GCBO)
    %eventdata  reserved - to be defined in a future version of MATLAB
1250 %handles    empty - handles not created until after all CreateFcns called
    set(hObject,'String','20');
    %Hint: edit controls usually have a white background on Windows.
    %       See ISPC and COMPUTER.
    if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
       defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end



1260 function fTiempoPaso_Callback(hObject, eventdata, handles)
    %hObject    handle to fTiempoPaso (see GCBO)
    %eventdata  reserved - to be defined in a future version of MATLAB
    %handles    structure with handles and user data (see GUIDATA)

    %Hints: get(hObject,'String') returns contents of fTiempoPaso as text
    %       str2double(get(hObject,'String')) returns contents of fTiempoPaso as
       a double


    %--- Executes during object creation, after setting all properties.
1270 function fTiempoPaso_CreateFcn(hObject, eventdata, handles)
    %hObject    handle to fTiempoPaso (see GCBO)
    %eventdata  reserved - to be defined in a future version of MATLAB
    %handles    empty - handles not created until after all CreateFcns called

    %Hint: edit controls usually have a white background on Windows.
    %       See ISPC and COMPUTER.
    set(hObject,'String','10');
    if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
       defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end
```

6. Código de c#

El código de c# se distribuye en múltiples ficheros, son de una extensión considerable y además buena parte del código está relacionado con funciones gráficas, que pueden verse en Microsoft Visual Studio (se utilizó la versión 2010). Es por esto que me remito a los ficheros depositados digitalmente junto con el proyecto para poder revisar el código con detalle.

7. Software utilizado

- **MiKTeX 2.9 64bit**
  Redacción del proyecto.
  http://miktex.org/download
- **Arduino IDE 1.03**
  Elaboración de código para Arduino.
  https://www.arduino.cc/en/Main/Software
- **Microsoft Visual Studio 2010**
  Elaboración de código para PC.
- **National Instruments Design Circuits Suite 13.0**
  Elaboración del prototipo de PCB.
  https://sine.ni.com/nips/cds/view/p/lang/es/nid/204742
- **Microsoft Office 2007**
  Gráficos, diagramas y edición de portadas.
- **GIMP 2.8**
  Edición de gráficos.
  http://www.gimp.org/
- **Screenshot Captor**
  Capturas de pantalla.
  https://www.donationcoder.com/Software/Mouser/screenshotcaptor/