Escola Tècnica Superior d'Enginyeries
Industrial i Aeronàutica de Terrassa
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Titulación:

INGENIERÍA EN AUTOMÁTICA Y ELECTRÓNICA INDUSTRIAL

Alumno:

ALBERTO IZQUIERDO PERÁLVAREZ

Título del PFC:

DISEÑO DE UN DISPOSITIVO BASADO EN MICROPROCESADOR PARA LA IDENTIFICACIÓN DE SISTEMAS DINÁMICOS LINEALES

Director del PFC:

ALBERT MASIP ALVAREZ

Convocatoria de entrega del proyecto:

SEPTIEMBRE DEL 2013

Contenido del documento:

MEMORIA

1. Índice de contenido

2. Índice de figuras

3. Índice de tablas

4. Abreviaciones

RW – Escritura/Lectura.

RO – Solo Lectura.

Fs – Frecuencia de muestreo.

Ts – Tiempo de muestreo.

CE – Conformidad Europea.

ADC – Convertidor Analógico-Digital.

DAC – Convertidor Digital-Analógico.

ROM – Memoria de solo lectura.

COM – Puerto de comunicaciones.

DFT – Transformada discreta de Fourier.

SVD – Descomposición de valores singulares.

TVS – Supresor de voltaje transitorio.

PWM – Modulación por ancho de pulsos.

Alpha – Fase inicial de un producto y sin el proceso de verificación aplicado.

5. Agradecimientos

Agradezco a mi padre, a mi madre y a mi hermana, que a pesar de la distancia que nos separa, todos los días han estado a mi lado.

A mi familia, la cual ha hecho más llevadero este proyecto.

A mis amigos, que han sabido cuando estar a mi lado y cuando aguantar todas mis conversaciones.

A mi tutor por haber llevado este proyecto.

6. Introducción y alcance del proyecto

6.1 Objetivos

El objetivo de este proyecto es implementar un dispositivo basado en un microprocesador que permita la identificación de sistemas. Se implementará desde el dispositivo, hasta el Firmware del dispositivo y el Software que permitirá la identificación.

6.2 Antecedentes y justificaciones

En la industria actual deben controlarse muchos procesos. En la mayoría de ocasiones es necesario saber cómo se comportará el sistema antes de poder realizar el control de una manera óptima.

Para poder saber el comportamiento del sistema y realizar controles más avanzados se debe identificar el sistema y de esta manera poder extraer un modelo matemático.

Actualmente existen pocos dispositivos que permitan una identificación sin el uso de herramientas de Software avanzadas y de pago como Matlab.

En este proyecto se ha diseñando un dispositivo programado en idiomas abiertos y estándar cómo C ANSI y C++, pudiendo ser un sistema totalmente modulable y escalable a muchas aplicaciones. Este dispositivo es capaz de extraer el modelo para su posterior uso en control.

Además, este sistema es fácilmente escalable a un dispositivo que pueda estar encastado en la línea de control, debido a que se ha programado en C++ y se ha implementado la arquitectura. Por tanto se pueden realizar controles e identificaciones avanzadas.

6.3 Alcance y estructura del documento

Tal y como se verá en los capítulos posteriores, este dispositivo es capaz de implementar una identificación.

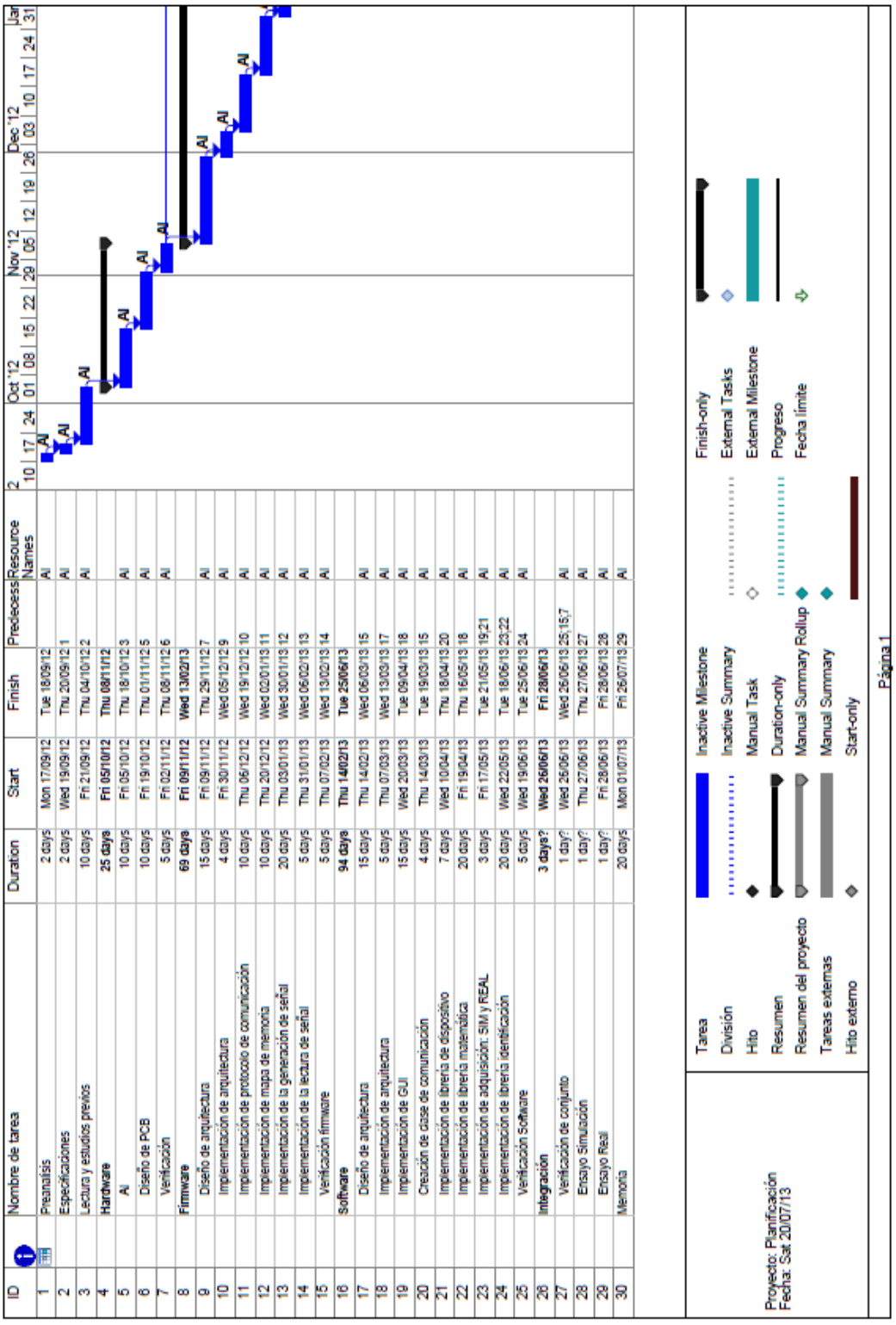
En los capítulos 7 y 8 se realiza una introducción a los modelos y a los tipos de identificaciones que se pueden realizar.

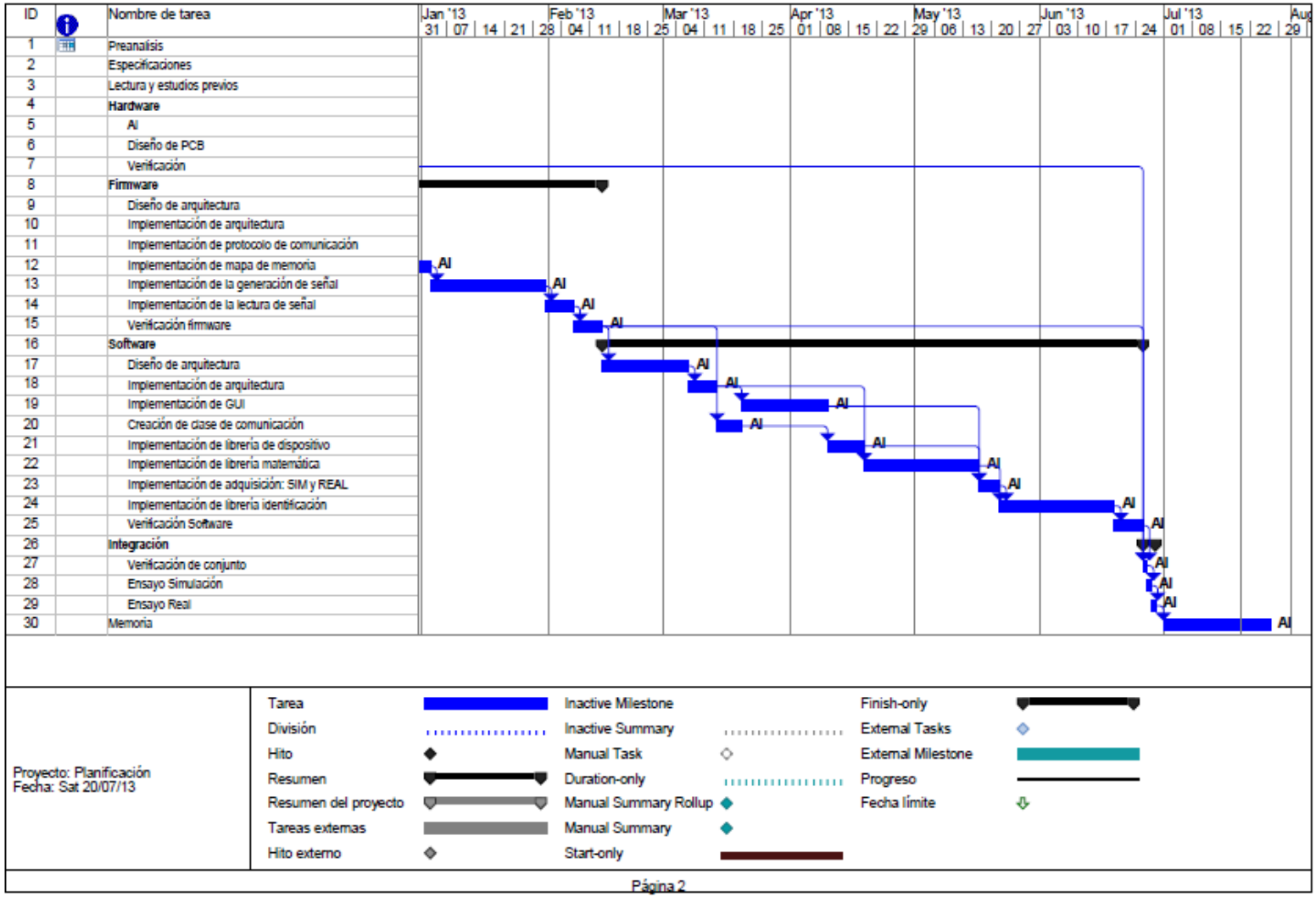
En segundo lugar, en el capítulo 9 se explican las funciones matemáticas que se utilizarán durante el proceso de identificación.

A lo largo de los capítulos 10, 11, 12 y 13 se explica cada parte desarrollada en el proyecto, es decir, desde el Software hasta el Hardware.

Por último se podrá ver el ensayo realizado y las conclusiones.

7. Planificación




ID
Nombre de tarea
Jan '13
Feb '13
Mar '13
Apr '13
May '13
Jun '13
Jul '13
1 Preanalisis
2 Especificaciones
3 Lectura y estudios previos
4 Hardware
5 AI
6 Diseño de PCB
7 Verificación
8 Firmware
9 Diseño de arquitectura
10 Implementación de arquitectura
11 Implementación de protocolo de comunicación
12 Implementación de mapa de memoria
13 Implementación de la generación de señal
14 Implementación de la lectura de señal
15 Verificación firmware
16 Software
17 Diseño de arquitectura
18 Implementación de arquitectura
19 Implementación de GUI
20 Creación de clase de comunicación
21 Implementación de librería de dispositivo
22 Implementación de librería matemática
23 Implementación de adquisición: SIM y REAL
24 Implementación de librería identificación
25 Verificación Software
26 Integración
27 Verificación de conjunto
28 Ensayo Simulación
29 Ensayo Real
30 Memoria
AI
Proyecto: Planificación
Fecha: Sat 20/07/13
Tarea
División
Hito
Resumen
Resumen del proyecto
Tareas externas
Hito externo
Inactive Milestone
Inactive Summary
Manual Task
Duration-only
Manual Summary Rollup
Manual Summary
Start-only
Finish-only
External Tasks
External Milestone
Progreso
Fecha límite
Página 2

8. Modelado de sistemas

El modelado de sistemas consiste en realizar una representación matemática aproximada de elementos físicos.

Se pueden realizar diferentes tipos de modelados:

- Modelos físicos: Maquetas.
- Modelos matemáticos: Ecuaciones.
- Modelos verbales: Sistemas expertos.
- Modelos mentales: Opinión de personas.

El objetivo de realizar modelos de sistemas es poder representar sistemas, poder generar soluciones, explorar relaciones o controlar sistemas.

En este proyecto se pretende conseguir la implementación de un modelo matemático para usos en control de sistemas.

Los modelos matemáticos serán un conjunto de ecuaciones que relacionarán las variables de interés del proceso y representarán con la máxima exactitud su comportamiento. Habrá que tener en cuenta que siempre serán aproximaciones y que por siguiente se deberá llegar a un compromiso: facilidad del modelo & exactitud.

El modelo que se va a buscar en este proyecto va a ser un modelo discreto, es decir, un modelo donde el tiempo no es continuo, sino que existe una discretización del tiempo. En este caso, el tiempo se debe discretizar por el hecho que el modelado se realizará en un entorno digital, de manera que números serán finitos.

En este proyecto se obtendrá el modelo matemático a partir del proceso *Identificación q*ue consiste en interactuar con el elemento a modelar y así poder hallar la ecuación matemática del modelo mediante la observación de la respuesta.

9. Identificación de modelos dinámicos

La identificación de modelos dinámicos consiste en la modelización de un sistema a partir de la interacción con éste.
Para realizar una identificación se debe excitar el sistema de una determinada manera y obtener su respuesta. Así se puede obtener el modelo matemático mediante la comparación de la señal de excitación y de la de respuesta

El procedimiento para la identificación de sistemas debe tener en cuenta:

- Planificación del experimento, es decir, elección de señal de excitación.
- Selección de la estructura del modelo, es decir, elección del orden o la complejidad del modelo.
- Estimación de los parámetros del modelo.
- Validación del modelo.

Se pueden realizar dos tipos de identificaciones:

- Identificaciones temporales.
- Identificaciones frecuenciales.

9.1 Identificación temporal

En este tipo de identificación, el análisis del modelo se realiza en el ámbito temporal.

9.1.1 Señales de excitación

La elección de la señal de excitación es una de las partes más importantes en una identificación temporal, debido a que la calidad del modelo dependerá en gran parte de la señal de excitación.
Los siguientes parámetros se deberán tener en cuenta:

- Voltaje máximo y mínimo de las señales.
- Tiempo de la señal.
- Corriente de la señal de excitación.

El tipo de señales que son apropiadas para una identificación temporal son:

- Señal Step:

Esta señal es una función que tiene un valor inicial y, en un determinado instante, cambia de valor:

$$u(t) = \begin{cases} V_{min}, & t < 0 \\ V_{max}, & t \geq 0 \end{cases}$$ **Ecu. 1**

La Figura 1 muestra la forma de una señal step.

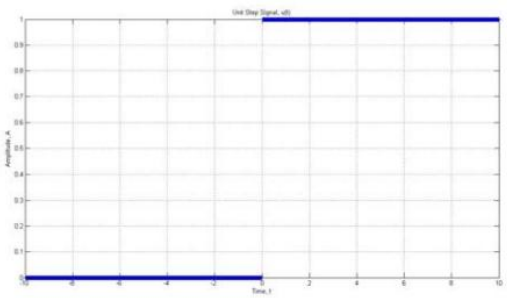


Figura 1: Step signal

- Señal Ramp.

Esta señal incrementa su amplitud proporcionalmente al tiempo. La definición matemática es:

$$u(t) = \begin{cases} V_{min}, & t < 0 \\ V_{max} \cdot k \cdot t, & t \geq 0 \end{cases}$$ **Ecu. 2**

La Figura 2 muestra la forma de una señal ramp.

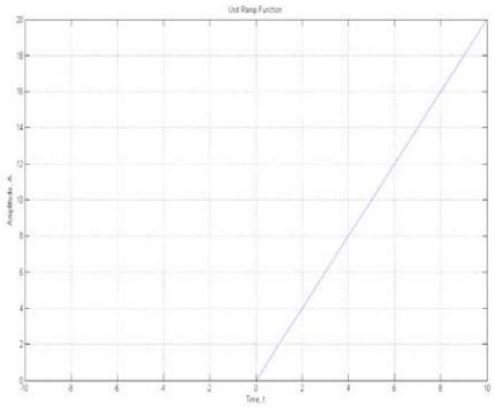


Figura 2: Ramp signal

9.1.2 Análisis de los datos

Una vez se ha seleccionado el tipo de señal y además se ha realizado el ensayo, se habrán obtenido todos los datos del ensayo.

Si se han utilizado señales de tipo temporal, el tipo de identificación puede ser:

- Modelos lineales paramétricos.
- Modelos del proceso.
- Modelos no lineales.
- Modelos correlativos.

9.2 Identificación frecuencial

En este tipo de identificación, el análisis del modelo se realiza en el ámbito frecuencial.

9.2.1 Señales de excitación

La elección de la señal de excitación es una de las partes más importantes en una identificación frecuencial, ya que la calidad del modelo dependerá en gran parte de ella.
Los siguientes parámetros se deberán tener en cuenta:

- Voltaje máximo y mínimo de las señales.
- Tiempo de la señal.
- Corriente de la señal de excitación.
- Frecuencia mínima y máxima de la señal.

El tipo de señales que son apropiadas para una identificación frecuencial son:

- Señal Multisine:

Es una señal compuesta con una suma de sinusoidales.

- Señal aleatoria Uniform:

Es un tipo de señal dónde sus valores conforman una distribución uniforme.

9.2.2 Análisis de los datos

Una vez se ha seleccionado el tipo de señal y además se ha realizado el ensayo, se dispone de todos los datos del ensayo.

Si se han utilizado señales de tipo frecuencial, el tipo de identificación puede ser:

- Modelos lineales paramétricos.
- Modelos espectrales.

9.3 Validación

La validación del modelo es una parte muy importante para poder determinar si se ha estimado correctamente.

Por tanto, para validar un modelo se deben cumplir estas condiciones:

- Similitud entre las respuestas temporales y frecuenciales.
- Lejanía entre los ceros y los polos y sobre-parametrización del modelo.
- No correlación entre la señal de error y la entrada.

En muchas validaciones se utilizan parámetros o algoritmos que analizan la adecuación del modelo:

- Factor de calidad:

$$Q = 100\left(1 - \frac{|y - \hat{y}|}{|y - mean(y)|}\right) \qquad \textbf{Ecu. 3}$$

10. Funciones matemáticas para la identificación

Para poder implementar los algoritmos de identificación de este dispositivo, se han necesitado algoritmos matemáticos adicionales.

Los algoritmos implementados son:

- DFT (*Discrete Fourier Transform*).
- SVD (*Singular Value Decomposition*).
- Matriz pseudoinversa.

10.1 DFT

La transformada discreta de Fourier convierte en una lista finita de muestras temporales a una lista finita de sinusoidales complejas ordenadas por su frecuencia.

Es por esto que esta transformada convierte una función temporal al dominio frecuencial.

El algoritmo a implementar es el siguiente:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N} \qquad \textbf{Ecu. 4}$$

El código que implementa esta función es el siguiente:

```
void DFT (bool inverse, double u32size, double *pu32VectorIn, double
*dVectorOut )
{

	std::complex<double> output_data[FSAMPLING];

	UINT32 u32Index = 0;
	std::complex<double> cTemp = (0,0);

	double dTemp = 0;
	double dW[FSAMPLING];
	double cosx = 0;
	double sinx = 0;
```

```
	UINT32 u32j = 0;
	UINT32 u32n = 0;
	ProgressBarCounter = 0;
	ProgressBarCounterMax = u32size;

	for (u32j = 0; u32j < (FSAMPLING); u32j++)
	{
		ProgressBarCounter++;
		dW[u32j] = 2*PI*(u32j + 1)*0.1;

		for (u32n = 0; u32n < u32size; u32n++)
		{
			cosx =
cos((double)dW[u32j]*(double)u32n/(double)FSAMPLING);
			sinx = -
1*sin((double)dW[u32j]*(double)u32n/(double)FSAMPLING);

			cTemp.real(cosx);
			cTemp.imag(sinx);

			output_data[u32j] = output_data[u32j] +
(double)pu32VectorIn[u32n]*(2/(double)u32size)*cTemp;
			dVectorOut[u32j] = abs (output_data[u32j]);
		}

	}
}
```

10.2 SVD

La descomposición en valores singulares de una matriz trata de factorizar una matriz real o compleja. En consecuencia, es muy útil en aplicaciones dónde se requiera la mejorar de la velocidad de cálculo.

El SVD consiste en hallar la matriz que cumpla:

$$A = U\Sigma V^T$$ **Ecu. 5**

Donde:

A = Matriz a extraer los SVD.

U = Matriz unitaria.

∑ = Matriz diagonal.

V = Matriz unitaria.

Para poder implementar de forma automática mediante código C++ se ha utilizado la factorización QR.

10.3 Matriz pseudoinversa

Para el cálculo de la identificación es necesaria la utilización de matrices y por tanto se deberá operar con matrices inversas.

Hay que tener en cuenta que como las matrices pueden no ser cuadradas se deberá utilizar el algoritmo de Moore–Penrose.

11. Arquitectura del sistema de identificación

En este proyecto se ha implementado un sistema completo de identificación de modelos dinámicos. En la Figura 3 se puede apreciar la implementación del sistema.

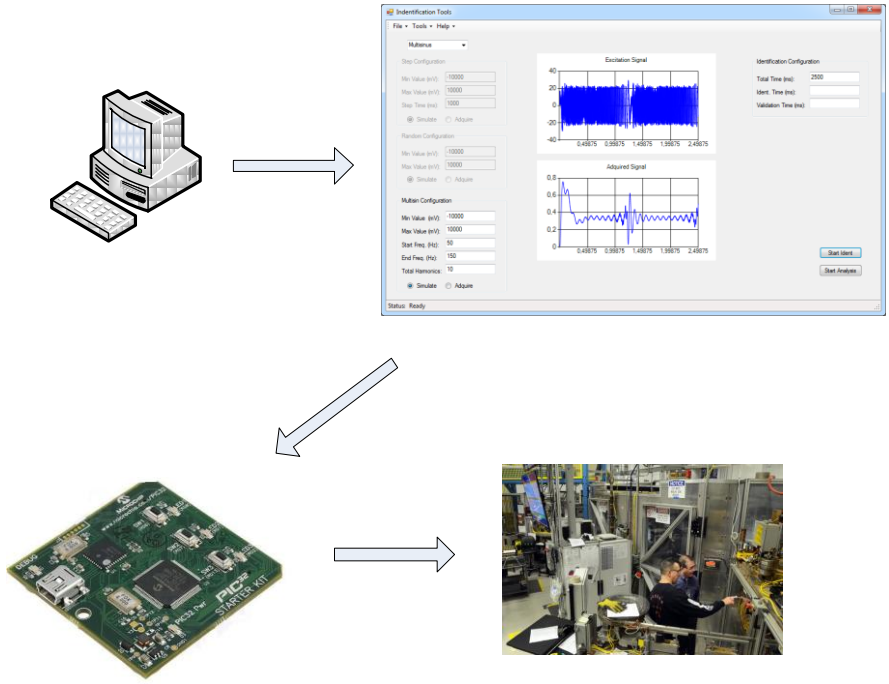


Figura 3: Arquitectura del sistema

Para poder realizar una identificación dinámica (véase Apartado 7) el sistema debe constar de:

- Hardware + Firmware capaz de generar las señales de excitación, realizar una lectura de las señales de la planta a identificar y una alta capacidad de almacenamiento de datos capturados.
- Software capaz de interpretar los datos del ensayo y extraer el modelo.

11.1 Hardware (HW)

El Hardware es la parte tangible del sistema, es decir todos los elementos electrónicos, eléctricos, cables, etc.

Para que el HW sea capaz de poder generar las señales debe tener las siguientes características:

- Alimentación a 220V
- Comunicación USB.
- Generación de señal de excitación -10 V a 10 V con corriente máx. 3 A y una resolución de 2048 cuentas (9,75 mV/c).
- Lectura de señal de respuesta de -10 V a 10V con una resolución de 1024 c (19,53125 mV/c)

En la Figura 4 se puede apreciar la arquitectura Hardware.

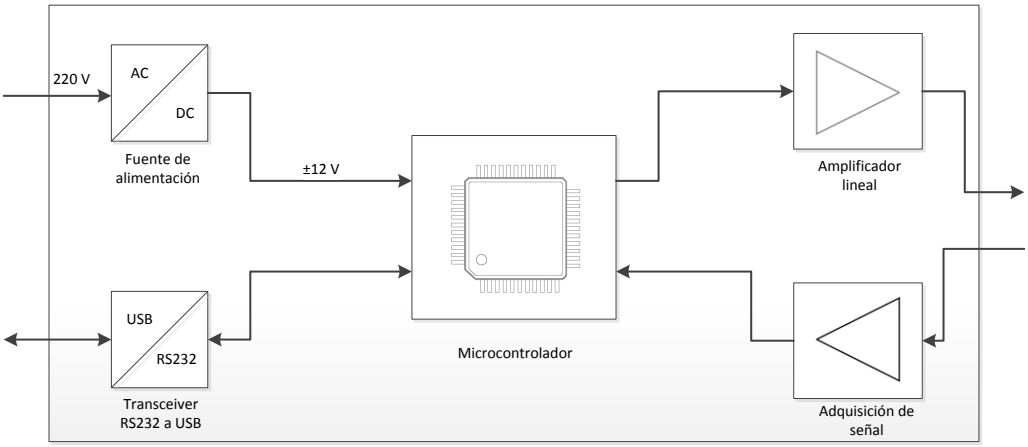


Figura 4: Arquitectura Hardware

11.2 Firmware (FW)

El firmware es el programa que necesita el Hardware para desarrollar su normal operación. Es el encargado de gestionar las comunicaciones, generar la señal de excitación y la lectura de la señal de respuesta en el sistema implementado.

Para que el FW sea capaz de satisfacer las funcionalidades anteriores debe cumplir las siguientes características:

- Implementar un mapa de memoria para poder configurar el dispositivo y almacenar 20.000 posiciones de memoria para la señal de excitación y la de respuesta.
- Implementar un protocolo de comunicaciones que permita una comunicación robusta.
- Implementar funciones para la generación de señales en tiempo real (Frecuencia de muestreo → Fs = 800 Hz).
- Implementar la generación de señales:
    - Step.
    - Uniform.
    - Multisine Schroeder.
- Implementar lectura de señales (Frecuencia de muestreo → Fs = 800 Hz).

En la Figura 5 se puede apreciar la arquitectura Firmware.

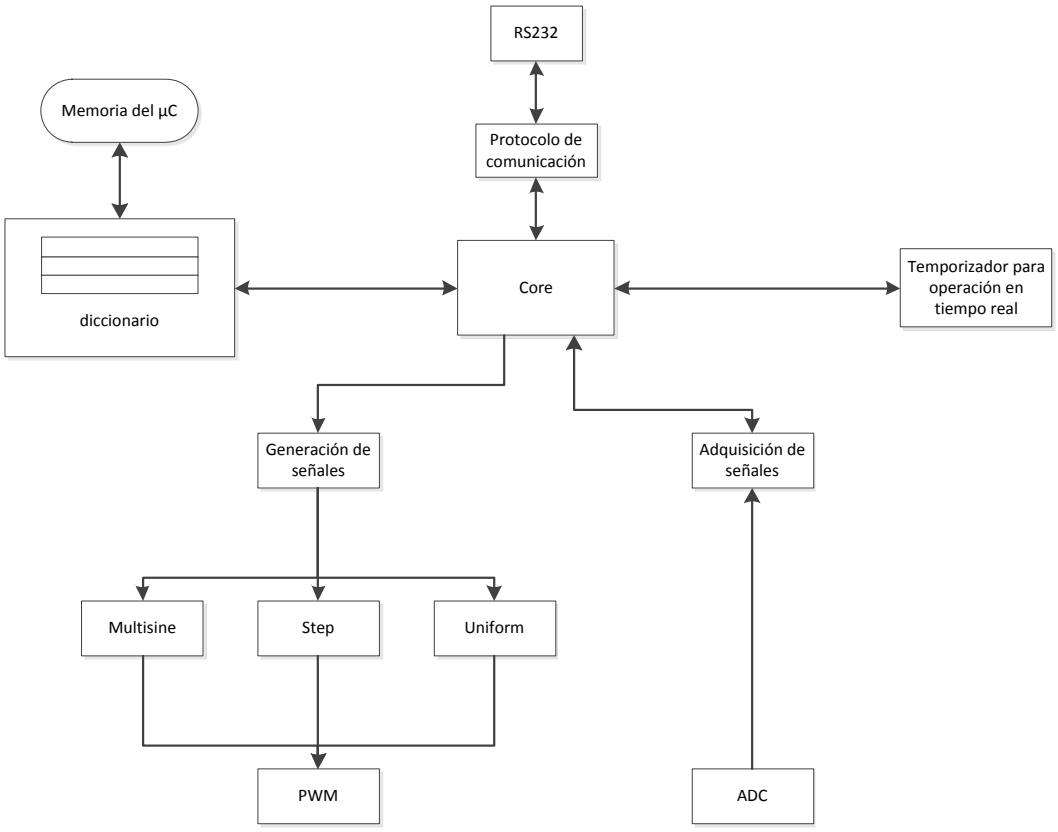


Figura 5: Arquitectura Firmware

11.3 Software (SW)

Según el estándar IEEE 729 *El software es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados, que forman parte de las operaciones de un sistema de computación.*

Por tanto sistema actual el software es el encargado de gestionar el dispositivo, ejecutar el proceso de identificación, y analizar los datos para extraer el modelo.

Para que el software pueda realizar todas sus funciones, debe cumplir las siguientes especificaciones:

- Implementar protocolo de comunicaciones para poder realizar comunicarse con el dispositivo.
- Configuración del dispositivo para la ejecución y lectura del la identificación.
- Visualización de las señales de excitación y de respuesta.
- Visualización de las señales procesadas (Temporal y frecuencial).
- Pre-procesado de la señal (Eliminación de media o tendencia lineal).
- Estimación temporal del modelo.
- Estimación frecuencial del modelo.
- Validación del modelo.
- Visualización de las señales del modelo.
- Visualización de los parámetros del modelo.

En la Figura 6 se puede apreciar la arquitectura del Software.

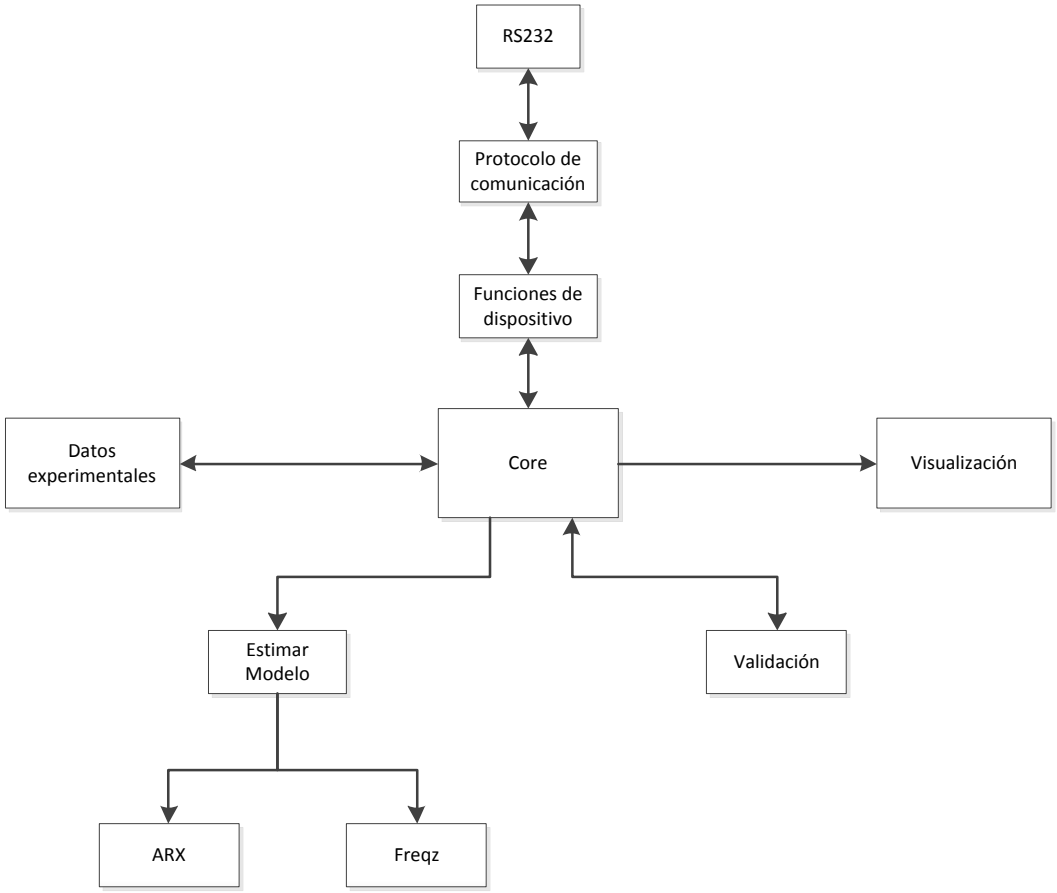


Figura 6: Arquitectura Software

12. Hardware del sistema

En este apartado se detallará cómo se ha implementado el Hardware en este proyecto. La explicación quedará detallada por módulo. Véase Figura 4.

12.1 Fuente de alimentación

Se ha escogido una fuente de alimentación que sea capaz de administrar los ±12 V @1,5 A. Véase Tabla 1.

Código fabricante	Código Farnell	Código RS
TXL 035-1212D	1242618	466-6862

Tabla 1: Referencia fuente alimentación

Como se detallará en el apartado *Tratamiento de la señal de excitación*, el módulo que genera la señal de excitación tiene mayor capacidad de corriente que de salida. Así, si se requiere aumentar la corriente máxima del sistema, se podrá escoger una fuente alternativa que sea capaz de dar hasta 3 A.

Se ha escogido una fuente con menor capacidad de corriente para poder reducir costes de producción. No obstante, si una aplicación requiere la máxima corriente del HW se podría escoger la fuente de la Tabla 2.

Código fabricante	Código Farnell	Código RS
LPQ252-C	1171526	192-1245

Tabla 2: Referencia fuente alimentación recomendada.

12.2 Comunicaciones

Como especificación del proyecto, se ha establecido que la comunicación tiene que ser de tipo USB. Con el propósito de que el producto sea versátil y se pueda utilizar en cualquier PC.

Para la implementación del USB se han planteado 3 opciones:

- Integrar la comunicación USB en el µC.
- Implementar un diseño hardware con un transceiver USB-RS232.
- Incorporar un módulo "*plug & play*" USB-RS232.

12.2.1 Comunicación USB en el µC

Esta tipología se basa en agregar un hardware específico en el µC con controlador de USB integrado que sea independiente a la CPU, de este modo el mismo µC es quien gestiona el protocolo y no se requiere ninguna electrónica adicional para la comunicación USB[1].

El µC escogido incorpora el hardware para poder implementar el protocolo USB a través de FW. Véase apartado *Microcontrolador (µC)*.

- Ventajas:
    - No requiere electrónica adicional.
    - El coste de producción es bajo.
    - Velocidad de la comunicación alta.
- Desventajas:
    - El tiempo de desarrollo firmware es alto.

Para mayor información sobre la implementación del USB ver apartado *Bibliografía.*

[1] El µC no incorpora las protecciones adecuadas para pasar la norma CE.

12.2.2 Comunicación USB con transceiver

Esta tipología se basa en integrar un módulo RS232-USB con todos sus periféricos. Véase Figura 7.

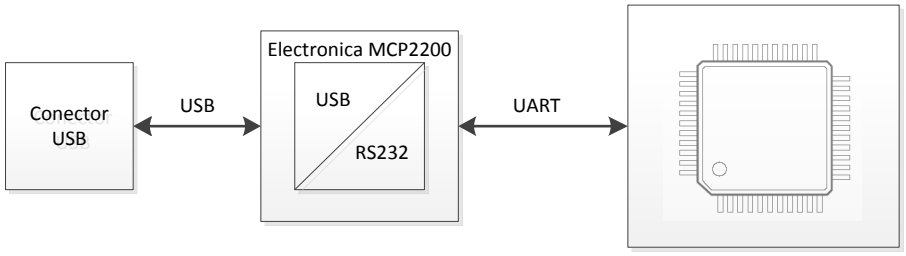


Figura 7: Comunicación USB con transceiver

Como se puede apreciar en la Figura 7, la tipología anterior requiere que el µC disponga de comunicación RS232 para que después el transceiver implemente un *bridge*[2] USB-RS232.

Un transceiver que puede satisfacer las especificaciones puede ser el que se aprecia en la Tabla 3.

Código fabricante	Código Farnell	Código RS
MCP2200-I/SO	1781148	698-8993

Tabla 3: Referencia transceiver RS232

En el documento de planos se puede ver el circuito de una posible implementación.

- Ventajas:
  - Coste de producción medio.
  - Tiempo de desarrollo medio.
- Desventajas:
  - Requiere hardware adicional.
  - Requiere configuración software del transceiver.
  - Requiere instalación del driver.

[2] Dispositivo que interconecta más de un protocolo de red (En este caso RS232 con USB)

12.2.3 Comunicación USB con módulo

Esta tipología se basa en insertar un módulo USB-RS232 con conector USB directamente en la salida UART del µC.

Los módulos USB-RS232 “plug&play” incorporan el hardware de protección, el protocolo y todos los drivers de instalación al PC. Por tanto el usuario tiene que conectar el dispositivo al ordenador y el PC se auto-configura para su uso.

La estructura de esta tipología es similar a la anterior. Véase Figura 8.

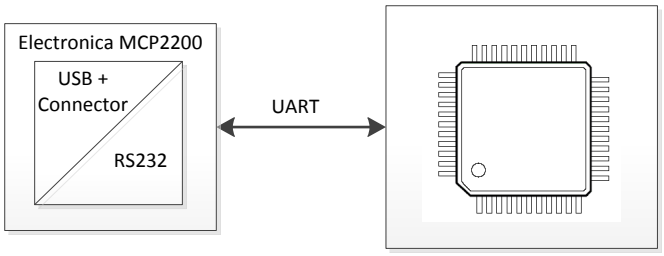


Figura 8: Comunicación USB con módulo

Un transceiver que puede satisfacer las especificaciones puede ser el que se aprecia en la Tabla 4.

Código fabricante	Código Farnell	Código RS
MM232R	1146037	406-562

Tabla 4: Referencia módulo USB-RS232

- Ventajas:
  - La implementación es muy rápida.
  - No requiere instalación de driver en el ordenador.
  - No requiere configuración del módulo.
- Desventajas:
  - El coste de producción es alto.

12.2.4 Elección para el dispositivo

Finalmente se ha escogido el módulo *“plug&play”* para este dispositivo. Esta elección se debe a que el uso del modulo “plug&play” minimiza tanto el tiempo de desarrollo en la fase de prototipo como la posibilidad de fallo.

En un futuro se deberá tender a la solución integrada, es decir, a la solución donde el hardware esté integrado en el µC. De este modo se deberá desarrollar un firmware específico que implemente el USB y un hardware que disponga las protecciones adecuadas que establece las normativas CE.

La solución actual se implementara con el módulo de la Figura 9.

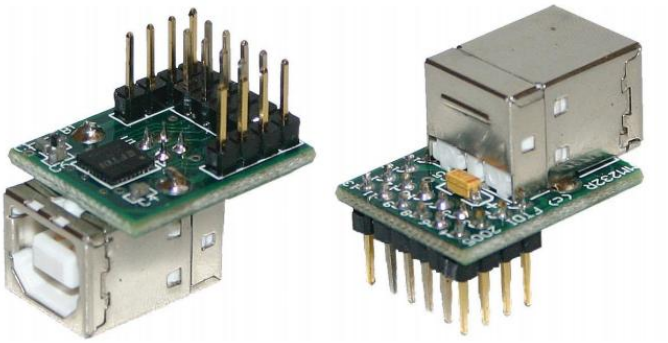


Figura 9: Módulo USB-RS232

El esquema del circuito se encuentra en el apartado Planos.

12.3 Microcontrolador (µC)

Para una correcta elección del microcontrolador se deben tener en cuenta los siguientes aspectos:

- Comunicaciones.
- Tipo de CPU.
- Velocidad de procesado.
- Memoria de programa.
- Memoria RAM.
- Finalidad de la aplicación.
- Proyección de futuro.
- Coste del producto.

12.3.1 Comunicaciones

Para que este dispositivo sea fácilmente ampliable en un futuro, se establecen los siguientes requisitos de comunicaciones:

- 1 x Comunicación UART:

Para comunicación con módulo USB-RS232.

- 1 x Comunicación USB integrada en µC:

Para comunicación directa con USB.

- 1 x Ethernet MAC:

Para futuras aplicaciones con Ethernet.

- 1 x CAN:

Para futuras aplicaciones industriales dónde se pueda implementar CANOpen.

- SPI:

Para comunicaciones con memoria externa.

12.3.2 Tipo de CPU

Para que se puedan realizar todos los cálculos en tiempo real de la generación de la señal, se necesita un µC que sea capaz de optimizar las operaciones a 16 bit. Es necesario que las operaciones sean de 16 bit para poder implementar las funciones de generación de la señal con una resolución mínima de 16 bit.

12.3.3 Velocidad de procesado

La generación en tiempo real de señal requiere que el cálculo se realice rápidamente. De esta manera se podrá mantener la señal en el momento adecuado. Además el µC simultáneamente debe poder gestionar memoria, así

como el protocolo de comunicación, etc. (para mayor información véase apartado *Firmware*). Se podría tener una estimación de la frecuencia mínima del sistema con la Ecu. 6 si tenemos en cuenta que el generación de señales no debe sobrepasar el 40% de la carga de la CPU.

$$F_{min} = \frac{1}{T_s} \cdot \frac{Cicles\ Signal\ Generation}{Max\ CPU\ usage} = \quad \textbf{Ecu. 6}$$

$$= \frac{1}{800\mu s} \cdot \frac{2000}{0.4} = \mathbf{62,5\ MHz}$$

12.3.4 Proyección de futuro

Para poder escoger adecuadamente el µC hay que tener la aplicación y la vida útil del producto a realizar.

Actualmente este prototipo implementa la identificación de procesos, pero realizando los mínimos cambios se podrían mejorar las comunicaciones, se podían controlar los procesos una vez se hayan identificado, etc.

Razón por la cual se escogerá un µC el cual no vaya limitado en recursos y así sea fácilmente ampliable para su uso industrial.

12.3.5 Elección para el dispositivo

Se ha escogido un µC de la familia PIC32 de Microchip el cuál reúne todas las características anteriores con un coste muy bajo. Véase Tabla 5.

Código fabricante	Código Farnell	Código RS
PIC32MX795F512H-80I/PT	1778989	687-9344

Tabla 5: Referencia µC

12.4 Adaptación de la señal

Como se ha visto en el apartado *Hardware*, las señales deben generarse y/o leerse a una tensión de ± 10 V. El µC, por lo contrario, tiene un rango de tensiones de 0…3 V, así que debemos hacer un tratamiento de la señal para poder adaptar los niveles.

Además, en ambas señales se realiza un filtrado para reducir el ruido leído o generado.

12.4.1 Tratamiento de la señal adquirida

El circuito que se ha implementado se puede ver en el apartado *Planos*. Este módulo es el encargado de poder adaptar la señal adquirida a la señal que se le inyecta al µC. Es por ello que se ponen una serie de protecciones, filtros y adaptaciones de señal. Véase Figura 10.

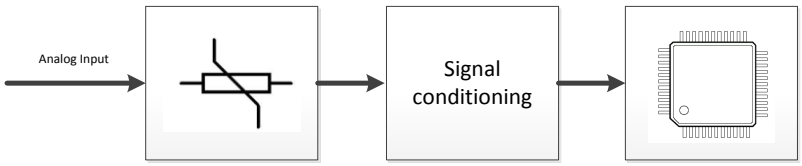


Figura 10: Adaptación de señal adquirida.

La adaptación de esta señal consiste en ajustar una señal adquirida de ±5V a una señal de 0 ~ 3.3V.

También se establece un seguidor de tensión y un TVS para proteger al µC de la señal adquirida. Para mayor información sobre el circuito véase el apartado *Planos*.

12.4.2 Tratamiento de la señal de excitación

El circuito que se ha implementado se puede ver en el apartado *Planos*. Este módulo es el encargado de poder adaptar la señal de excitación generada por el µC. Es por ello que se ponen una serie de protecciones, filtros y adaptaciones de señal. Véase Figura 11.

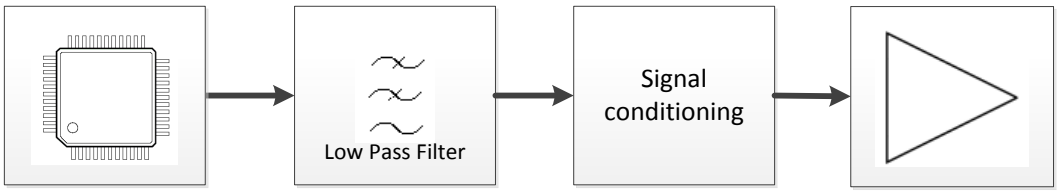


Figura 11: Adaptación de la señal de excitación

La adaptación de esta señal consiste en ajustar una señal PWM generada por el µC de 0 ~ 3.3V a una señal de excitación no PWM de ±5V.

Para mayor información sobre el circuito véase el apartado *Planos*.

13. Firmware del sistema

El firmware del sistema es el encargado de implementar la generación de señales, lectura de señales y almacenaje de la información.

Para poder hacer que el firmware ejecute una señal u otra y leer las señales capturadas, se ha implementado un protocolo de comunicaciones. Así el software puede comunicarse de manera cómoda.

El firmware se ha programado bajo la plataforma Microchip con el µC PIC32MX795F512H. La estructura de este controlador se puede ver en la Figura 12.

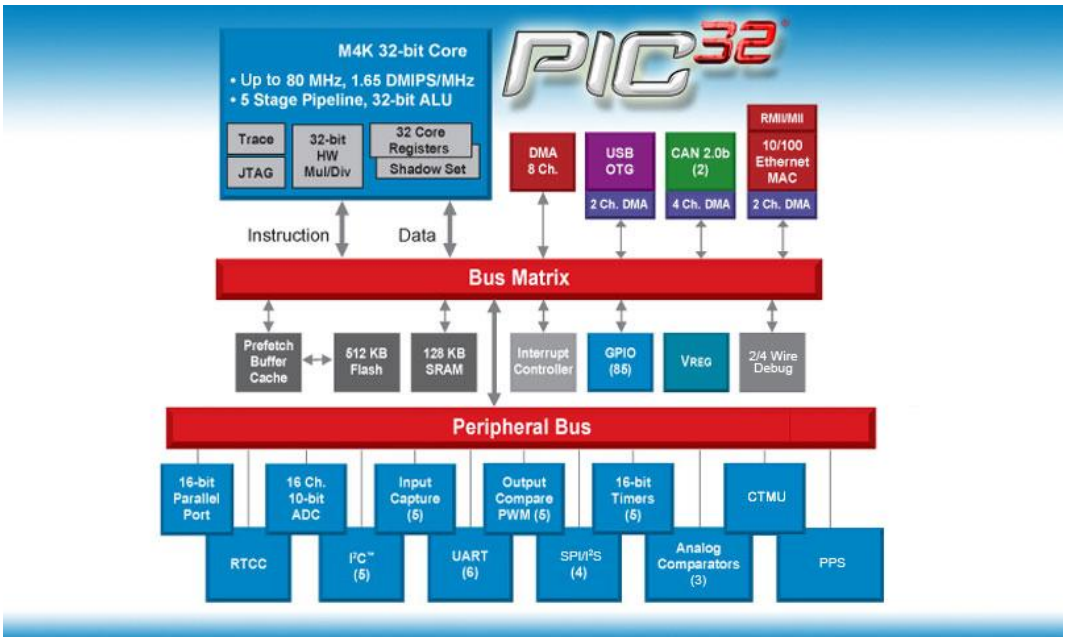


Figura 12: Arquitectura PIC32

13.1 Protocolo de comunicaciones

El protocolo de comunicaciones que se ha implementado utiliza el interfaz RS232.

Este interfaz está diseñado para cortas distancias, 15 m aproximadamente, y además para bajas velocidades. Este interfaz puede trabajar tanto síncronamente como asíncronamente.

13.1.1 Configuración de la interfaz

Para realizar una correcta configuración de la interfaz de comunicación, el software de comunicación se debe ajustar a los parámetros que se pueden ver en la Tabla 6.

Baudrate	115200 bps
Data bits	8 bits
Parity	None
Stop bits	1 bit
Flux control	None

Tabla 6: Configuración del RS232

Un ejemplo de software con el que se podría comunicar este dispositivo seria cualquier programa que fuese capaz de abrir un COM, por ej. *Hyperterminal, TeraTerm, RealTerm*, etc. En el caso de este proyecto, esta configuración ya la implementa el Software desarrollado.

13.1.2 Formato de la trama

Una vez se ha configurado el interfaz de comunicación se debe seguir una trama correcta para poderse comunicar de manera satisfactoria.
La trama implementada utiliza códigos ASCII[3]. El formato de la trama que se ha desarrollado es parecido a la trama CANopen[4], permitiendo así, que en un futuro sea fácilmente escalable el dispositivo a comunicaciones CANopen.

- **Identificador de nodo:** En este prototipo se debe indicar el nodo 0x20, pero la estructura de programación permite que se puedan poner varios ejes en una misma interfaz RS232.

- **Tipo de función a realizar (FCT):** Este campo puede ser de escritura o de lectura. Cuando el FCT contiene el carácter 'W', indica que el comando contiene un proceso de escritura. Por otro lado, si el FCT contiene el carácter 'R', indica que el comando contiene un proceso de lectura.

- **Número de registro:** Este campo indica el registro al que se quiere acceder para leer o escribir.

[3] Para mayor información véase http://www.asciitable.com/
[4] Para mayor información véase http://www.can-cia.org/

- **Valor del registro:** Este campo sólo es necesario en el caso que se lleve a cabo una proceso de escritura y se indicará el valor que se desee escribir.

La Tabla 7 muestra la composición de la trama:

ID	Espacio	FCT	Espacio	Registro	Espacio	Valor	CR	LF
0x20	‘_‘	‘R’/’W’	‘_‘	0x6040	‘_‘	1	/r	/n

Tabla 7: Trama protocolo de comunicaciones implementado

Ejemplo de lectura:

- Enviado desde el host: 0x20 R 0x6040
- Respuesta del dispositivo: 0x20 W 0x6040 0

Ejemplo de escritura:

- Enviado desde el host: 0x20 W 0x6040 1

13.2 Registros de configuración (Mapa de memoria)

Como se ha visto en el apartado *Protocolo de comunicaciones*, el dispositivo se ha clasificado en registros para que la comunicación se pueda realizar de manera satisfactoria.

Estos registros se inicializan al arrancar el Firmware y se guardan en la memoria RAM. El usuario, mediante el protocolo los puede modificar y el Firmware en cada bucle de ejecución los actualiza para su correcto funcionamiento.

Existen diferentes tipos de registros:

- Registros de control de sistema.
- Registros de configuración.
- Registros de lectura.

13.2.1 Registros de control de sistema

Estos registros son los encargados de gestionar el estado del sistema:

ControlWord:

Índice	Nombre	Tipo de datos	Acceso	Valor defecto
0x6040	Control Word	UINT32	RW	0

Modificando el valor del registro, el dispositivo podrá desempeñar una función u otra. y por tanto modificando el valor de éste, el dispositivo podrá desempeñar una función u otra.

Los estados del dispositivo son los siguientes:

- *Operating Mode:*

El sistema espera nuevo parámetro.

- *Stop Mode:*

El sistema está parado.

- *Open Mode* (uso futuro)*:*

El sistema está en modo open, es decir, se puede establecer un valor en la señal de excitación y leer un valor en la señal de respuesta.

- *Ident Mode*:

El sistema está en modo identificación, es decir, el sistema está listo para realizar una identificación del modelo.

- *Control Mode* (uso futuro)*:*

El sistema está en modo control en lazo cerrado, es decir, el sistema puede realizar un control de la planta con los parámetros de identificación encontrados.

- *Monitoring Mode* (uso futuro)*:*

El sistema realiza una lectura de la señal de respuesta a partir de que se entra en este modo.

La Tabla 8 mostrada a continuación muestra los estados que el registro *ControlWord* puede tomar:

Valor	Descripción del estado
0	*Operating Mode*
1	*Stop Mode*
2	*Open Mode*
3	*Ident Mode*
4	*Control Mode*
5	*Monitoring Mode*

Tabla 8: Estados del *ControlWord*

Hay que tener en cuenta que para que el dispositivo realice un cambio de estado el *ControlWord* tiene que tener una transición de estado. Es por ello que cada vez que se quiera reiniciar la operación de un estado se deberá pasar por el estado *Operating Mode*.

- Ejemplo: Ejecutar 2 veces *Ident Mode*

1. Escribir → 0x20 W 0x6040 0
2. Escribir → 0x20 W 0x6040 3
3. Esperar a que el proceso de identificación finalice
4. Escribir → 0x20 W 0x6040 0
5. Escribir → 0x20 W 0x6040 3

StatusWord:

Índice	Nombre	Tipo de datos	Acceso	Valor defecto
0x6041	Status Word	UINT32	RO	0

Este registro es el encargado de mostrar el estado actual del dispositivo.

Los estados que el *StatusWord* muestra son:

- *Identification Started*:

El *StatusWord* muestra que el sistema está ejecutando un proceso de identificación.

- *Identification Finished*:

El *StatusWord* muestra que el sistema ha finalizado un proceso de identificación.

La Tabla 9 mostrada a continuación muestra los estados que el registro *StatusWord* puede tomar:

Valor	Descripción del estado
0	*Default*
1	*Identification Started*
2	*Identification Finished*

Tabla 9: Estados del *StatusWord*

El uso del *StatusWord* es muy útil para el Software, ya que en todo momento se puede saber el estado del dispositivo.

- Ejemplo: Ejecutar 2 veces *Ident Mode* teniendo en cuenta el estado del *StatusWord.*

1. Escribir → 0x20 W 0x6040 0
2. Escribir → 0x20 W 0x6040 3
3. Leer → 0x20 R 0x6041
4. Repetir punto 3 hasta que el valor leído sea 2.
5. Escribir → 0x20 W 0x6040 0
6. Escribir → 0x20 W 0x6040 3
7. Leer → 0x20 R 0x6041
8. Repetir punto 3 hasta que el valor leído sea 2.

13.2.2 Registros de configuración

Estos registros son los encargados de configurar el tipo de identificación que se va a desempeñar.

IdentType:

Índice	Nombre	Tipo de datos	Acceso	Valor defecto
0x12050	Ident Type	UINT8	RW	0

Este registro es el encargado de configurar el tipo de señal que se va a generar para realizar la identificación. El tipo de señales que se pueden generar son:

- Multisin:

Esta señal es una composición de sinusoidales, ideal para realizar identificación frecuencial. Para mayor información sobre la generación de esta señal véase el apartado *Multisine Schroeder*.

- Step:

Esta señal es de tipo escalón, ideal para realizar identificación temporal. Para mayor información sobre la generación de esta señal véase el apartado *Step*.

- Uniform:

Esta señal es de tipo aleatoria (con una distribución uniforme. Para mayor información sobre la generación de esta señal véase el apartado *Uniform*.

La Tabla 10 mostrada a continuación muestra los estados que el registro *IdentType* puede tomar:

Valor	Descripción del estado
0	Multisine
1	Step
2	Uniform

Tabla 10: Estados del *IdentType*

TotalTime:

Índice	Nombre	Tipo de datos	Acceso	Valor defecto
0x22050	Total Time	UINT32	RW	5000

Este registro es el encargado de establecer el tiempo de toda la identificación. Está expresado en **ms** y su valor por defecto es 5000 ms.

MSINStartFrecuency:

Índice	Nombre	Tipo de datos	Acceso	Valor defecto
0x12051	MSIN Start Frecuency	UINT16	RW	50

Este registro es el encargado de definir la frecuencia mínima en la señal Multisine.

Para evitar el uso de decimales en la implementación del Firmware, este registro se ha escalado y por tanto las unidades no son Hz.

La unidad es múltiple de 100 mHz. La Tabla 11 muestra el escalado de unidades.

Valor SI	Valor en registro
0 Hz	0
100 mHz	1
10 Hz	100

Tabla 11: Escalado de unidades

MSINEndFrecuency:

Índice	Nombre	Tipo de datos	Acceso	Valor defecto
0x22051	MSIN End Frecuency	UINT16	RW	150

Este registro es el encargado de definir la frecuencia máxima en la señal Multisine.

Para evitar el uso de decimales en la implementación del Firmware, este registro se ha escalado y por tanto las unidades no son Hz.

La unidad es múltiple de 100 mHz. La Tabla 12 muestra el escalado de unidades.

Valor SI	Valor en registro
0 Hz	0
100 mHz	1
10 Hz	100

Tabla 12: Escalado de unidades

MSINNumHarmonics:

Índice	Nombre	Tipo de datos	Acceso	Valor defecto
0x32051	MSIN Num Harmonics	UINT8	RW	2

Este registro es el encargado de definir los harmónicos totales que tendrá la señal multisine.

Idealmente, cuanto mayor es el número de harmónicos en una señal multisine mayor es la calidad de la señal. Sin embargo, hay que tener en cuenta que la discretización mínima viene determinada por la Fs[5] y por tanto no se podrá establecer un número de harmónicos que suponga una superación en la frecuencia de muestreo. Véase Ecu. 7.

$$\frac{F_{max} - F_{min}}{NumHarmonics} > F_s$$ **Ecu. 7**

MMinValue:

Índice	Nombre	Tipo de datos	Acceso	Valor defecto
0x42051	MSIN Min Value	UINT16	RW	0

Este registro es el encargado de definir el valor mínimo de tensión que tendrá la señal.

Para evitar el uso de decimales en la implementación del Firmware, este registro se ha escalado a unidades del ADC[6]. Este escalado se puede ver según la Tabla 13.

Valor Registro	Valor en señal
0	-10 V
32767	0 V
65535	10 V

Tabla 13: Escalado de unidades

MMaxValue:

Índice	Nombre	Tipo de datos	Acceso	Valor defecto
0x52051	MSIN Max Value	UINT16	RW	65535

[5] Véase apartado *Abreviaciones.*
[6] Véase apartado *Abreviaciones.*

Este registro es el encargado de definir el valor máximo de tensión que tendrá la señal.

Para evitar el uso de decimales en la implementación del Firmware, este registro se ha escalado a unidades del ADC. Este escalado se puede ver según la Tabla 14.

Valor Registro	Valor en señal
0	-10 V
32767	0 V
65535	10 V

Tabla 14: Escalado de unidades

StepTime:

Índice	Nombre	Tipo de datos	Acceso	Valor defecto
0x62051	Step Time	UINT32	RW	1000

Este registro es el encargado de establecer el tiempo cuando la señal de step cambia su valor de estado lógico 0 a 1. Está expresado en **ms** y su valor por defecto es 1000 ms.

13.2.3 Registros de lectura

Estos registros son los encargados de gestionar la lectura de las señales del ensayo.

IndexInVector:

Índice	Nombre	Tipo de datos	Acceso	Valor defecto
0x42050	Index In Vector	UINT16	RW	0

Este registro se utiliza para poder leer los datos del vector in, es decir, de la señal de respuesta del sistema.

En este registro se debe indicar la posición del vector que se va a leer.

DataInVector:

Índice	Nombre	Tipo de datos	Acceso	Valor defecto
0x52050	Data In Vector	UINT16	RO	0

Este registro se utiliza para poder leer los datos del vector in, es decir, de la señal de respuesta del sistema.

Ejemplo de lectura de 2 datos de la señal: respuesta del sistema:

1. Escribir → 0x20 W 0x42050 0
2. Leer → 0x20 R 0x52050 (Se obtiene valor 0)
3. Escribir → 0x20 W 0x42050 1
4. Leer → 0x20 R 0x52050 (Se obtiene valor 1)

IndexOutVector:

Índice	Nombre	Tipo de datos	Acceso	Valor defecto
0x62050	Index Out Vector	UINT16	RW	0

Este registro se utiliza para poder leer los datos del vector out, es decir, de la señal de excitación del sistema.

En este registro se debe indicar la posición del vector que se va a leer.

DataOutVector:

Índice	Nombre	Tipo de datos	Acceso	Valor defecto
0x72050	Data Out Vector	UINT16	RO	0

Este registro se utiliza para poder leer los datos del vector out, es decir, de la señal de respuesta del sistema.

Ejemplo de lectura de 2 datos de la señal: excitación del sistema:

1. Escribir → 0x20 W 0x42050 0
2. Leer → 0x20 R 0x52050 (Se obtiene valor 0)
3. Escribir → 0x20 W 0x42050 1
4. Leer → 0x20 R 0x52050 (Se obtiene valor 1)

13.3 Generación de las señales de excitación

La generación de señales es una de las partes más importantes de Firmware considerando que la calidad de la identificación depende de la calidad de la señal.

Para realizar una generación de la señal en tiempo real se ha implementado un firmware donde la función prioritaria es la generación de la señal en cuestión, por tanto, procesos como las comunicaciones o la actualización de parámetros quedan en segundo plano.

La señal generada se debería realizar mediante un DAC. Pero se ha utilizado una señal PWM modulada mediante la señal que se requiera de excitación. Después se aplica un filtrado para eliminar la generación PWM.

Esta función va ligada a un temporizador configurado a la Fs, entonces cada tiempo (Ts), se ejecuta la rutina de generación de señal.
En la rutina de generación de señal se determina qué señal se debe generar y se ejecuta una u otra según la configuración. Además, se deben pasar parámetros como el tiempo del sistema, etc.

Los encabezados de las rutinas de generación de señal son los siguientes:

*u16PWM = **MultisineSchroeder**(u16MSINStartFrecuency, u16MSINEndFrecuency, u8MSINNumHarmonics, TOGGLES_PER_SEC_1, u16MSINMinValue, u16MSINMaxValue, u64SystemSecCount);*

*u16PWM = **GenerateUniform**(u16MSINMinValue, u16MSINMaxValue);*

*u16PWM = **GenerateStep** (u16MSINMinValue, u16MSINMaxValue, u32StepTime, u64SystemSecCount);*

Como se puede ver en las funciones anteriores, el funcionamiento de la generación de señales requiere que se establezca el tiempo como parámetro.

Una vez la función tiene el tiempo actual, puede calcular el valor de la señal.

13.3.1 Step

La señal de step se implementa mediante la comparación del tiempo actual con el tiempo escalón. Si el tiempo es menor al escalón, la señal es un ‘0’ lógico, sin embargo, si el tiempo actual es mayor al tiempo escalón, entonces la señal es un ‘1’ lógico. Véase Figura 13.

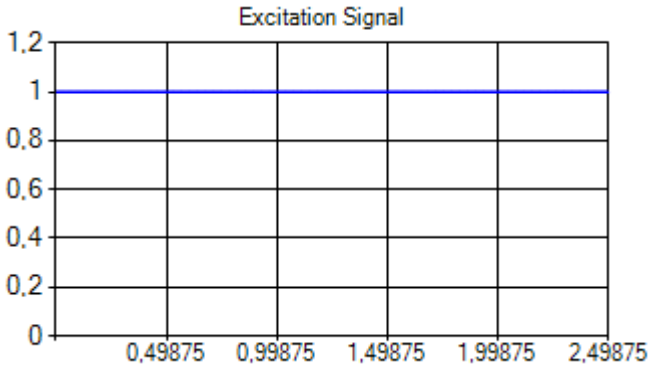


Figura 13: Señal Step

El siguiente código implementa esta función

```
T_UINT16 GenerateStep (T_UINT16 u16MinValue, T_UINT16 u16MaxValue, T_UINT32
u32StepTime, T_UINT64 u64SystemTime)
{
   T_UINT16 u16StepValue;
   if ( u32StepTime < u64SystemTime)
   {
      u16StepValue = u16MaxValue;
   }
   else
   {
      u16StepValue = u16MinValue;
   }
   return u16StepValue;
}
```

Esta comprobación se hace en cada iteración, es decir, en cada intervalo del tiempo de muestreo.

13.3.2 Uniform

La señal Uniform se genera mediante la generación de números aleatorios de siguiendo una distribución uniforme. Véase Figura 14.

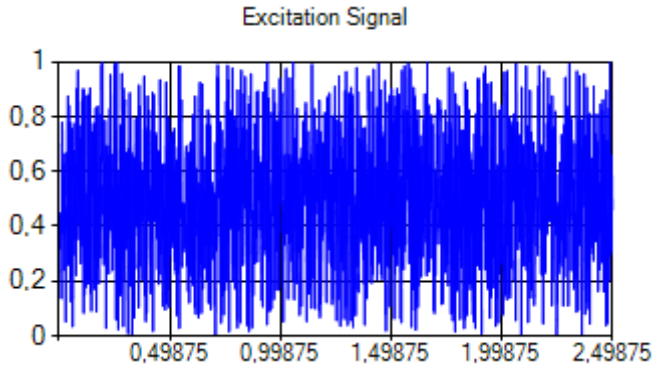


Figura 14: Señal Uniform

La implementación de una señal uniforme está dividida en dos partes:

- Generación de número aleatorio.
- Generación de distribución uniforme a partir de número aleatorio.

13.3.2.1 Generación de números aleatorios

Para la generación de números aleatorios se ha utilizado una librería que Microchip que es capaz de generarlos a partir de una semilla.

El funcionamiento es el siguiente:

- A partir de un valor inicial o semilla se aplica un algoritmo como el de la siguiente ecuación y se genera una serie de números aleatorios:

$$X_{k+1} = (g \cdot X_k) mod(n)$$ **Ecu. 8**

13.3.2.2 Generación de distribución uniforme

Una vez se ha obtenido la lista de valores aleatorios se debe ajustar a una distribución uniforme utilizando el teorema de centro-límite.

Este teorema indica que si S_n es la suma de n variables aleatorias independientes, entonces la función de distribución de S_n se aproxima a una distribución uniforme. Véase Figura 15.

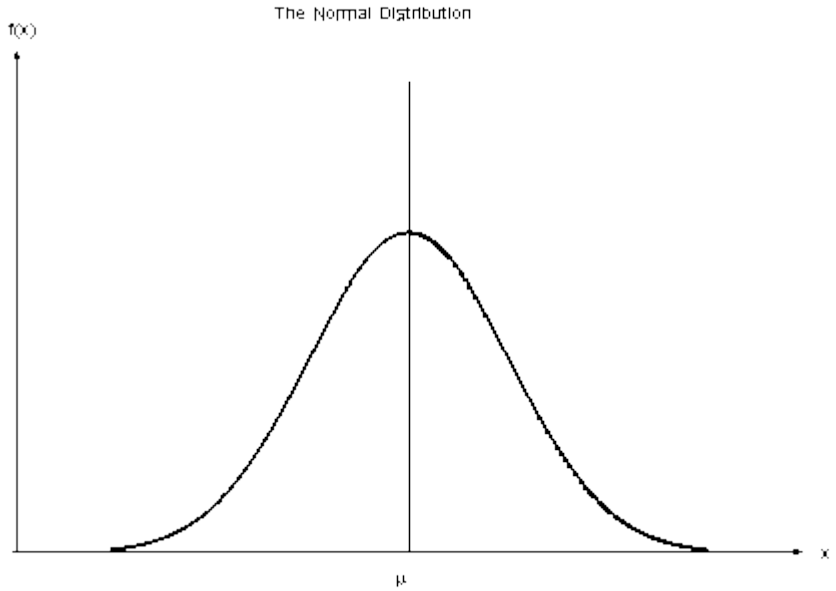


Figura 15: Distribución uniforme

El algoritmo que implementa una el teorema del centro-limite es el siguiente:

```
T_UINT16 GenerateUniform (T_UINT16 MinValue, T_UINT16 MaxValue)
{
    //static T_UBYTE u8IndexRSAMP = 0;
    static T_UINT16 u16RandomSignal [RANDOMSAMPLES];

    T_INT64 i64X = 0;
    T_UINT16 u16Uniform = 0;
    T_UBYTE u8Index1 = 0;

    for (u8Index1 = 0; u8Index1 < RANDOMSAMPLES; u8Index1++)
    {
        i64X += rand();
    }

    i64X -= (T_INT64)(32767/2)*(T_INT64)RANDOMSAMPLES;
    i64X *= sqrt(12.0/RANDOMSAMPLES);
    u16Uniform = (T_UINT16)(32767 + i64X/2);


    return (T_UINT16)(((T_UINT32)u16Uniform*(T_UINT32)MaxValue -
(T_UINT32)u16Uniform*(T_UINT32)MinValue)/32767 + MinValue);
}
```

Como se puede apreciar en el algoritmo anterior, la primera parte genera el número aleatorio con distribución uniforme y la segunda parte escala el valor

para que esté comprendido entre el Máximo y el Mínimo establecido en la configuración por el usuario.

13.3.3 Multisine Schroeder

Una señal multisine es una suma de sinusoidales con frecuencias y fases diferentes.

A diferencia de una multisine convencional, una multisine con Schroeder es una multisine donde las fases de las sinusoidales han sido calculadas para obtener una la amplitud mínima pico-pico. La siguiente ecuación muestra la función temporal de una multisinus:

$$u_m(t) = \sum_{k=1}^{N} A_m \cos\left(2\pi\omega_k t + \varphi_k\right) \qquad \textbf{Ecu. 9}$$

Donde:

N = Número total de harmónicos

$$\varphi_k \triangleq \frac{-k(k-1)\pi}{N}$$

$$\omega_k \triangleq \omega_{min} + \frac{(k-1)(\omega_{max} - \omega_{min})}{N}$$

En la Figura 16 se puede observar una señal multisine.

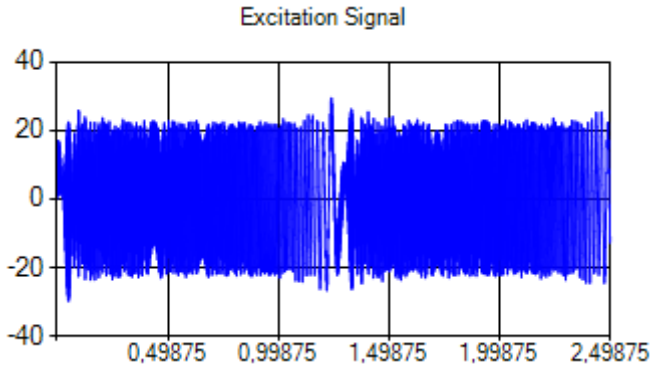


Figura 16: Señal Multisine

Para implementar la generación de esta señal a nivel Firmware se ha subdividido en 3 capas:

- Generación de función: $y = cos(x)$.
- Generación de onda: $y(t) = A \cdot cos(2\pi ft + \varphi)$
- Generación de multisinus.

13.3.3.1 Generación de función Cos(x)

La implementación en el Firmware de la función trigonométrica cosinus se puede realizar de diversos modos:

- Utilizando la función *cosl* de la librería *Math*[7]*:*

Esta opción consiste en establecer una llamada en el código a la función de la librería.
Es un método que requiere un tiempo bajo de implementación. Sin embargo, esta función requiere de mucho tiempo de procesado y por tanto no se recomienda su uso para generación.

- Utilizando una tabla de valores:

Esta opción consiste en tener una tabla estática con los valores del resultado de la función, es decir, una tabla donde la variable 'x' defina la posición de la tabla y en la celda 'x' de la tabla se pueda hallar el valor *cos(x)*.
Es un método que requiere un gran espacio en la memoria ROM. Por otro lado no necesita tiempo de procesado, ya que la única instrucción es un acceso a memoria ROM.

A pesar de la desventaja de necesitar un gran espacio de memoria, esta opción es la preferida para generar señales en tiempo real y es la que se ha implementado en este proyecto.

- Utilizando una tabla de valores + interpolación lineal:

Esta opción consiste en tener una tabla estática con pocos valores y el resto de valores extraerlos a partir de una interpolación lineal[8].

[7] Véase el siguiente enlace para mayor información http://ww1.microchip.com/downloads/en/DeviceDoc/51685E.pdf
[8] Véase el siguiente enlace para mayor información http://www.eng.fsu.edu/~dommelen/courses/eml3100/aids/intpol/

Es un método que no requiere tanto espacio en memoria como el caso anterior. En cambio requiere mayor tiempo de procesado y además el error de la función es mayor.
El código que implementa la función anterior es el siguiente:

```
T_UINT16 FCosine(T_UINT32 phase, T_UINT16 MinValue, T_UINT16
MaxValue)
{
    T_UINT16 Value;
    T_UINT64 Temp;

       if (phase < 3600)
       {
           Value = CosineValues[(T_UINT16)(phase)];
       }
       else
       {
           Value = CosineValues[((T_UINT16)(phase)%3600)];
       }

       Temp = (Value*MaxValue - Value*MinValue)/65535 + MinValue;

       return Temp;
}
```

Como se puede observar en el código anterior, la variable *CosineValues* es la tabla estática dónde se rescata el valor. Una vez se ha rescatado el valor de la tabla, se escala al valor máximo y mínimo configurado.

13.3.3.2 Generación de onda simple

La implementación de la onda simple se realiza siguiendo la siguiente ecuación:

$$y(t) = A \cdot cos(2\pi ft + \varphi)$$ **Ecu. 10**

El código que implementa esta función es el siguiente:

```
T_UINT16 CosineWaveform(T_UINT16 frecuency, T_UINT64 time, T_UINT16
phase, T_UINT16 SampleFrecuency,
                        T_UINT16 MinValue, T_UINT16 MaxValue)
{
    T_UINT64 Totalphase = 0;
```

```
    Totalphase = (360*frecuency*time)/SampleFrecuency + phase;
    if (Totalphase >= 3600)
    {
        Totalphase = Totalphase%3600;
    }

    return FCosine((T_UINT32)(Totalphase), MinValue, MaxValue);
}
```

Como se puede observar en el código anterior, esta función calcula la onda del coseno en función del tiempo.

13.3.3.3 Generación de multisine

La generación de la señal multisine se basa en la suma de sinusoidales. La implementación se realiza de la siguiente manera:

```
T_UINT16 MultisineSchroeder(T_UINT16 StartFrecuency, T_UINT16
EndFrecuency, T_UBYTE NumberHarmonics,
                            T_UINT16 SampleFrecuency, T_UINT16
MinValue, T_UINT16 MaxValue, T_UINT64 time)
{
    T_UINT64    TotalValue          = 0;
    T_UBYTE     HarmIter            = 0;
    T_UINT16    phase               = 0;
    T_UINT16    Temp                = 0;

    if (NumberHarmonics < 2)
    {
        NumberHarmonics = 2;
    }

    for (HarmIter = 0; HarmIter < NumberHarmonics; HarmIter++)
    {
        Temp = 1800*(HarmIter)/NumberHarmonics;
        if (Temp < 3600)
        {
            phase = Temp;
        }
        else
        {
            phase = Temp%3600;
        }

        //phase = 0;
        TotalValue = CosineWaveform((HarmIter)*(EndFrecuency -
StartFrecuency)/(NumberHarmonics - 1) + StartFrecuency,
                                    time, phase, SampleFrecuency,
MinValue, MaxValue)/NumberHarmonics + TotalValue;

    }
```

13.4 Adquisición de señales

La adquisición de la señal de respuesta se debe realizar de manera síncrona, por consiguiente, cuando se genera la señal, se ejecuta la rutina de lectura de señal.

Para realizar una lectura de señal se ha utilizado la librería de control de periféricos de microchip. La adquisición se realiza mediante un convertidor ADC de 10 bits que incorpora el µC internamente.

La lectura se realiza mediante la siguiente instrucción:

```
u16InVector[pointer] = ReadADC10(u16offset);
```

14. Software del sistema

El software del sistema es el encargado de gestionar todas las funciones del dispositivo. También implementa la identificación de la planta para poder extraer el modelo.

El software se ha desarrollado con el idioma de programación C++ bajo la plataforma Microsoft Visual Studio 2010.
Se ha elegido utilizar C++ para la implementación de la aplicación por el hecho que es un idioma estándar, modulable, escalable e integrable en aplicaciones más grandes.

Para el diseño de la parte visual se han utilizado las librerías .NET Framework, por tanto si un PC quiere utilizar el software desarrollado necesita los siguientes requerimientos:

- Instalar las librerías .Net Framework 4.0[9]

- Instalar el Microsoft Visual C++ 2010 Redistributable Package[10].

Para poder implementar el software de la aplicación se ha definido la arquitectura de la Figura 6.
En esta arquitectura se definen varios módulos que se irán detallando en los apartados posteriores.

[9] Descargar en http://www.microsoft.com/en-us/download/details.aspx?id=17718

[10] Descargar en http://www.microsoft.com/en-us/download/details.aspx?id=5555

14.1 Protocolo de comunicaciones

Para poder realizar la comunicación con el dispositivo de manera satisfactoria, se ha creado un protocolo de comunicación (véase apartado *Protocolo de comunicaciones).*

En el Software se debe crear una librería que permita:

- Abrir un canal COM.
- Detectar si hay dispositivo.
- Rutinas de escritura.
- Rutinas de lectura.

Para poder implementar el protocolo de comunicaciones a nivel Software se ha creado una clase[11] RS232 con el protocolo incorporado.

Esta clase se la ha llamado *Serial* e incluye las siguientes funciones:

- Initialize class:

Esta función abre el puerto COM del ordenador con la configuración seleccionado como parámetro.

- Write232:

Esta función escribe un carácter en la línea COM.

- Read232:

Esta función lee un carácter en la línea COM.

- WriteRS232:

Esta función utiliza internamente la función *Write232* para escribir todos los caracteres con el protocolo implementado. Esta función necesita los parámetros: identificador del dispositivo a comunicarse, registro a escribir y dato.

- ReadRS232:

Esta función utiliza internamente la función *Read232* para leer todos los caracteres con el protocolo implementado. Esta función necesita los parámetros: identificador del dispositivo a comunicarse y registro. Como parámetro de salida se obtiene el valor del registro a leer.

[11] Construcción que se utiliza como un modelo (o plantilla) para crear objetos de ese tipo.

- CloseRS232

Esta función cierra el puerto para dejar el COM libre y así otro software podrá acceder al dispositivo.

14.2 Ejecución del ensayo en el dispositivo

Para favorecer a la programación se ha creado una librería que es capaz de implementar todas las funciones que se pueden realizar desde el dispositivo.
Para el intercambio de información se han utilizado punteros[12] de C++ optimizando la rapidez de intercambio de información.

Las funciones que puede realizar esta librería son las siguientes:

- Execute Multisine Identification
- Execute Random Identification
- Execute Uniform Identification
- Read Identification Parameters

El proceso de ejecución del ensayo siempre consiste en lanzar 2 procesos:

En primer lugar se ejecuta la Identificación XXX, pudiendo ser la señal de excitación XXX: Multisine, Random o Uniform.

- Se ejecuta el proceso de lectura y se almacena los datos en dos vectores, vector señal excitación y vector señal de respuesta.

Para seleccionar la identificación se debe ejecutar la aplicación desarrollada y después en el *comboBox*[13] de *Excitation Signal* debe seleccionarse el tipo de identificación a desempeñar. Véase Figura 17.

[12] En programación C++, el puntero es la variable que contiene la dirección de memoria de otra variable.
[13] Elemento que permite al usuario escribir sobre este o seleccionar una opción de una lista existente de opciones

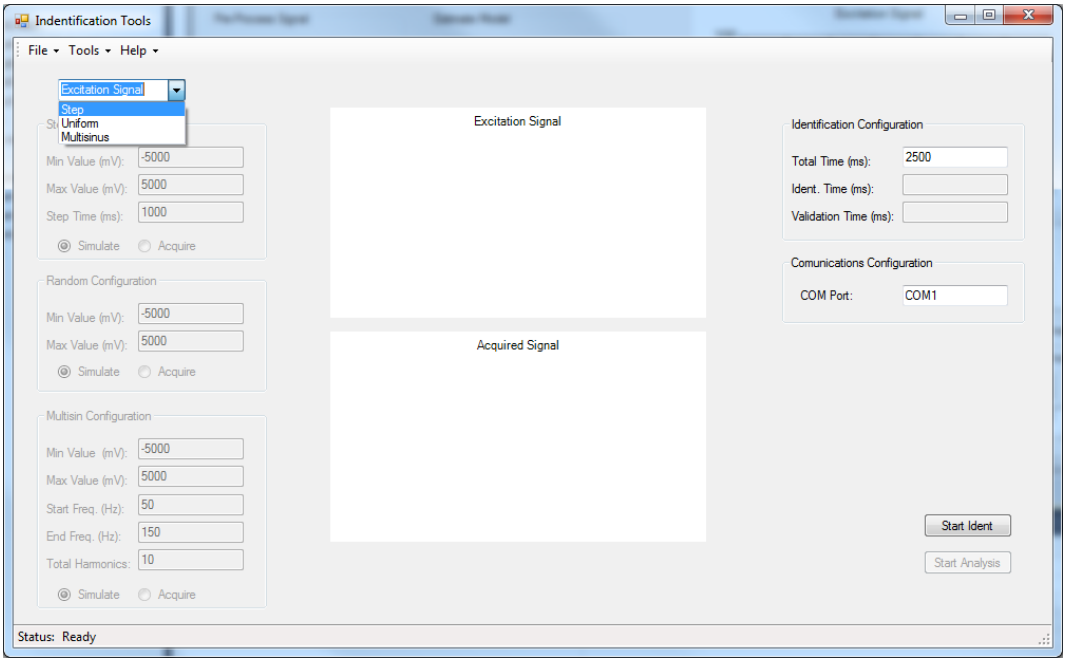


Figura 17: Selección del tipo de señal a ejecutar

Para ejecutar la identificación se tiene que pulsar sobre el botón *Start Ident*. Véase Figura 18.

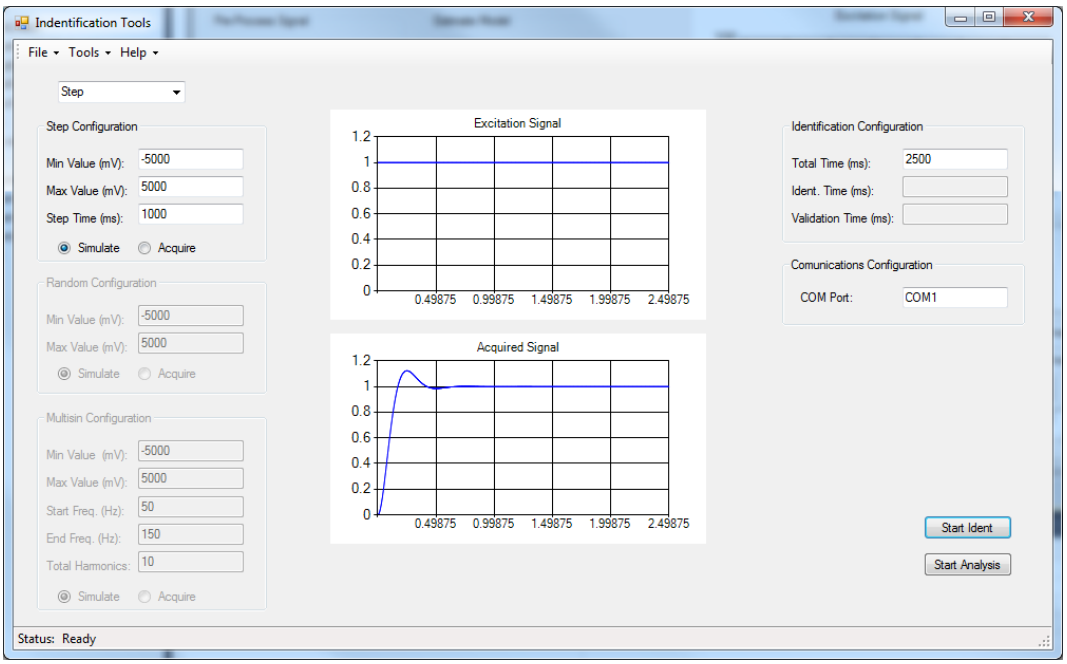


Figura 18: Ejecutar identificación

14.2.1 Execute Multisine Identification

Esta rutina ejecuta una identificación con una señal de excitación de tipo multisine, esta función es bloqueante[14]. Para poder utilizar esta rutina se necesitan los siguientes parámetros:

- TimeOut: Este parámetro define el tiempo máximo de bloqueo de la rutina. Se expresa en milisegundos.
  Establece un 1000 de valor. Si la identificación dura más de 1 s la rutina parará la identificación y dará error.
- TotalTime: Establece el tiempo total de identificación, este parámetro se escribe al dispositivo. Véase apartado *Registros de configuración (Mapa de memoria)*.
- StartFreq: Se escribe al dispositivo el parámetro Frec. Inicial. Véase apartado *Registros de configuración (Mapa de memoria)*.
- EndFreq: Se escribe al dispositivo el parámetro Frec. Final. Véase apartado *Registros de configuración (Mapa de memoria)*.
- NumHarmonics: Se escribe al dispositivo el parámetro número de hármonicos. Véase apartado *Registros de configuración (Mapa de memoria)*.
- MinValue: Se escribe al dispositivo el parámetro valor mínimo. Véase apartado *Registros de configuración (Mapa de memoria)*.
- MaxValue: Se escribe al dispositivo el parámetro valor máximo. Véase apartado *Registros de configuración (Mapa de memoria)*.

La función se ha implementado de la siguiente manera:

- Abrir canal de comunicación COM.
- Se escribe y se comprueba mediante una lectura el registro de identificación para multisin.
- Se escribe y se comprueba mediante una lectura el registro de tiempo total.
- Se escribe y se comprueba mediante una lectura el registro de frecuencia de start y end.
- Se escribe y se comprueba mediante una lectura el registro de número de harmónicos.
- Se escribe y se comprueba mediante una lectura el registro de mín. y máx. valor.

[14] Función que no finaliza hasta que no ha acabado la tarea.

- Se escribe y se comprueba mediante una lectura el registro ControlWord el estado modo de operación.
- Se escribe y se comprueba mediante una lectura el registro ControlWord el estado modo identificación.
- Se lee el registro StatusWord hasta que indique la identificación haya sido ejecutada, nótese que el bucle no es infinito, sino que como máximo estará el tiempo indicado en *TimeOut*.
- Se lee el registro StatusWord hasta que indique la identificación haya sido finalizada, nótese que el bucle no es infinito, sino que como máximo estará el tiempo indicado en *TimeOut*.
- Se cierra el puerto COM.

14.2.2 Execute Random Identification

Esta rutina ejecuta una identificación con una señal de excitación de tipo random. Esta función es bloqueante[15]. Para poder utilizar esta rutina se necesitan los siguientes parámetros:

- TimeOut: Este parámetro define el tiempo máximo de bloqueo de la rutina. Se expresa en milisegundos.
  Estableciendo un 1000 de valor, si la identificación dura más de 1 s la rutina parará la identificación y dará error.
- TotalTime: Establece el tiempo total de identificación, este parámetro se escribe al dispositivo. Véase apartado *Registros de configuración (Mapa de memoria)*.
- MinValue: Se escribe al dispositivo el parámetro valor mínimo. Véase apartado *Registros de configuración (Mapa de memoria)*.
- MaxValue: Se escribe al dispositivo el parámetro valor máximo. Véase apartado *Registros de configuración (Mapa de memoria)*.

La función se ha implementado de la siguiente manera:

- Abrir canal de comunicación COM.
- Se escribe y se comprueba mediante una lectura el registro de identificación para random.
- Se escribe y se comprueba mediante una lectura el registro de tiempo total.
- Se escribe y se comprueba mediante una lectura el registro de mín. y máx. valor.

[15] Función que no finaliza hasta que no ha acabado la tarea.

- Se escribe y se comprueba mediante una lectura el registro ControlWord el estado modo de operación.
- Se escribe y se comprueba mediante una lectura el registro ControlWord el estado modo identificación.
- Se lee el registro StatusWord hasta que indique la identificación haya sido ejecutada, nótese que el bucle no es infinito, sino que como máximo estará el tiempo indicado en *TimeOut*.
- Se lee el registro StatusWord hasta que indique la identificación haya sido finalizada, nótese que el bucle no es infinito, sino que como máximo estará el tiempo indicado en *TimeOut*.
- Se cierra el puerto COM.

14.2.3 Execute Step Identification

Esta rutina ejecuta una identificación con una señal de excitación de tipo step. Esta función es bloqueante[16]. Para poder utilizar esta rutina se necesitan los siguientes parámetros:

- TimeOut: Este parámetro define el tiempo máximo de bloqueo de la rutina. Se expresa en milisegundos.
  Estableciendo un 1000 de valor, si la identificación dura más de 1 s la rutina parará la identificación y dará error.
- TotalTime: Establece el tiempo total de identificación, este parámetro se escribe al dispositivo. Véase apartado *Registros de configuración (Mapa de memoria)*.
- MinValue: Se escribe al dispositivo el parámetro valor mínimo. Véase apartado *Registros de configuración (Mapa de memoria)*.
- MaxValue: Se escribe al dispositivo el parámetro valor máximo. Véase apartado *Registros de configuración (Mapa de memoria)*.
- StepTime: Se escribe al dispositivo el parámetro StepTime. Véase apartado *Registros de configuración (Mapa de memoria)*.

La función se ha implementado de la siguiente manera:

- Abrir canal de comunicación COM.
- Se escribe y se comprueba mediante una lectura el registro de identificación para random.

[16] Función que no finaliza hasta que no ha acabado la tarea.

- Se escribe y se comprueba mediante una lectura el registro de tiempo total.
- Se escribe y se comprueba mediante una lectura el registro de step time.
- Se escribe y se comprueba mediante una lectura el registro ControlWord el estado modo de operación.
- Se escribe y se comprueba mediante una lectura el registro ControlWord el estado modo identificación.
- Se lee el registro StatusWord hasta que indique la identificación haya sido ejecutada, nótese que el bucle no es infinito, sino que como máximo estará el tiempo indicado en *TimeOut*.
- Se lee el registro StatusWord hasta que indique la identificación haya sido finalizada, nótese que el bucle no es infinito, sino que como máximo estará el tiempo indicado en *TimeOut*.
- Se cierra el puerto COM.

14.2.4 Execute Read Parameters

Esta rutina ejecuta una lectura de las señales de identificación obtenidas a partir del ensayo realizado. Esta rutina es bloqueante. Para poder utilizar esta rutina se necesitan los siguientes parámetros:

- TotalIndex: Se indica el número de iteraciones totales a realizar.
- IndexRegister: Se escribe al dispositivo el parámetro Index Register. Véase apartado *Registros de configuración (Mapa de memoria)*.
- DataRegister: Se escribe al dispositivo el parámetro Data Register. Véase apartado *Registros de configuración (Mapa de memoria)*.
- PointerVector: Puntero del vector dónde se almacenarán los datos leídos.
- La función se ha implementado de la siguiente manera:

La función se ha implementado de la siguiente manera:

- Abrir canal de comunicación COM.
- Comprobar que las comunicaciones son correctas.
- Escribir en el registro IndexRegister el número de iteración
- Leer el registro DataRegister
- Guardar el valor en el PointerVector.
- Repetir el proceso hasta finalizar el bucle.
- Cerrar el canal de comunicación COM.

14.3 Pre-procesamiento de las señales

Una vez se han obtenido todos los datos de la identificación se puede hacer un tratamiento de la señal para así favorecer la correcta identificación.

Los dos algoritmos implementados son:

- Eliminación de media aritmética.
- Eliminación de tendencia lineal.

Para poder ejecutar el análisis se debe apretar sobre el botón *StartAnalysis. Véase* Figura 19*.*

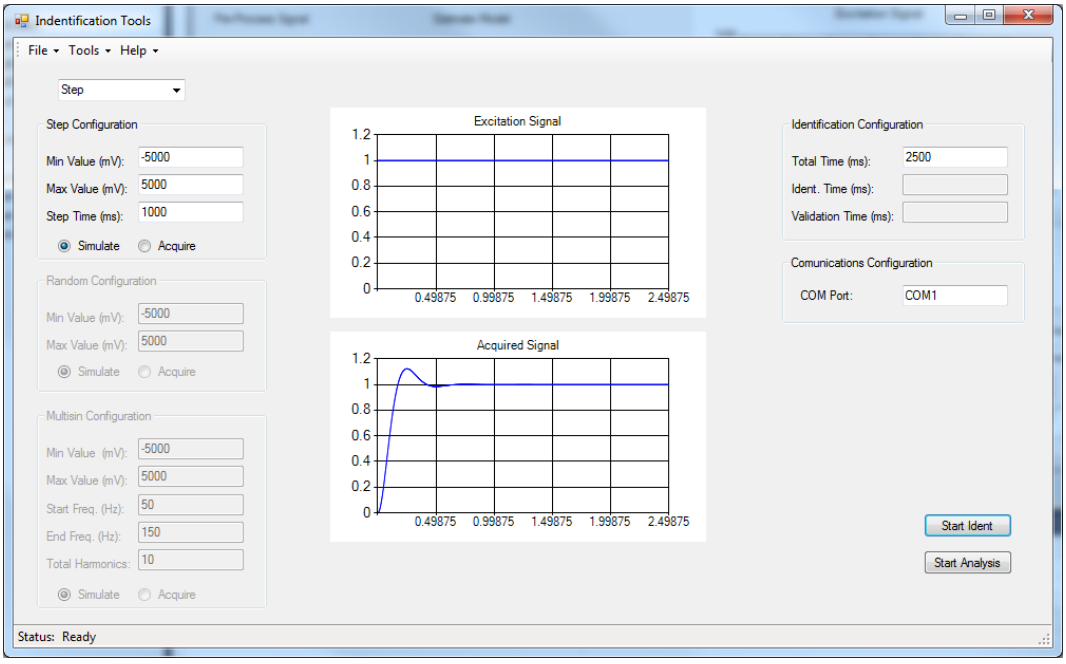


Figura 19: Empezar el análisis de las señales adquiridas

Para seleccionar el tipo de pre-procesamiento se debe seleccionar la señal a pre-analizar y el tipo de procesamiento. Véase Figura 20.

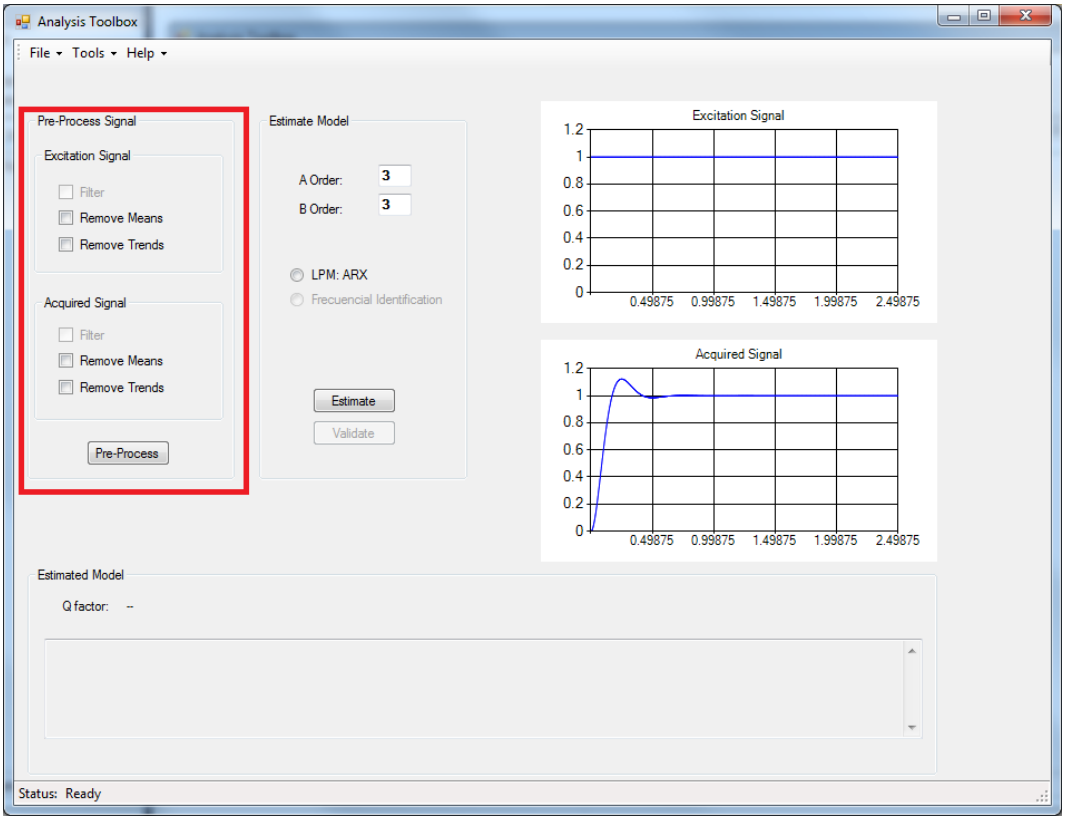


Figura 20: Seleccionar tipo de pre-procesado.

14.3.1 Eliminación de media aritmética

Por definición, la media aritmética es la suma de todos los elementos de un vector divididos por su tamaño, por tanto:

$$\bar{x} = \frac{1}{N}\sum_{k=1}^{N} x_k$$ **Ecu. 11**

$$\bar{x} = \frac{1}{N}(x_1 + \cdots + x_N)$$ **Ecu. 12**

La implementación en software es la siguiente:

- Obtener el vector al que se va a quitar la media.
- Sumar todos los elementos del vector.
- Dividir el resultado por el tamaño del vector.
- Recorrer todo el vector restando la media extraída.

$$x_k = (x_k - \bar{x}) \quad \textbf{Ecu. 13}$$

El código que implementa este algoritmo es el siguiente:

```
//Calculate Mean Signal
double dAdqMean = 0;
for (u16Index = 0; u16Index < (u32TotalTime*FSAMPLING/1000);
u16Index++)
{
	dAdqMean += dPrePAdqSignal [u16Index];

dAdqMean /= (u32TotalTime*FSAMPLING/1000);

//Remove Mean Signal
for (u16Index = 0; u16Index < (u32TotalTime*FSAMPLING/1000);
u16Index++)
{
	dPrePAdqSignal [u16Index] -= dAdqMean;
}
```

14.3.1 Eliminación de la tendencia lineal

La eliminación de la línea de tendencia consiste en hallar la línea de regresión de la señal y restársela. Como consecuencia se requiere implementar el cálculo de la línea de regresión.

Una línea de regresión se define como la recta que minimiza el error cuadrático medio. La ecuación que define como encontrar la recta es la siguiente:

$$y = \beta x + \alpha \quad \textbf{Ecu. 14}$$

Donde:

$$\beta = \frac{\sum_{k=1}^{N}(x_k - \bar{x})(y_k - \bar{y})}{\sum_{k=1}^{N}(x_k - \bar{x})^2} \quad \textbf{Ecu. 15}$$

$$\propto = \bar{y} - \beta\bar{x} \quad \textbf{Ecu. 16}$$

La implementación de esta función se ha realizado mediante el siguiente código:

```
double dAdqMeanY = 0;
 double dAdqMeanX = 0;
 //Get Lineal Regresion
 //Calculate Mean
 for (u16Index = 0; u16Index < (u32TotalTime*FSAMPLING/1000);
u16Index++)
 {
       dAdqMeanY += dPrePAdqSignal [u16Index];
       dAdqMeanX += u16Index;
 }
 dAdqMeanY /= (u32TotalTime*FSAMPLING/1000);
 dAdqMeanX /= (u32TotalTime*FSAMPLING/1000);

 double dnumAdq = 0;
 double ddenAdq = 0;
 for (u16Index = 0; u16Index < (u32TotalTime*FSAMPLING/1000);
u16Index++)
{
        dnumAdq += (dPrePAdqSignal [u16Index] -
dAdqMeanY)*(u16Index - dAdqMeanX);
        ddenAdq += (u16Index - dAdqMeanX)*((u16Index - dAdqMeanX));
 }
 //Remove trend
 double dBetaAdq = 0;
 double dAlphaAdq = 0;
 dBetaAdq = dnumAdq/ddenAdq;
 dAlphaAdq = dAdqMeanY - dBetaAdq * dAdqMeanX;

 for (u16Index = 0; u16Index < (u32TotalTime*FSAMPLING/1000);
u16Index++)
 {
        dPrePAdqSignal [u16Index] -= dAlphaAdq + dBetaAdq
*u16Index;
 }
```

14.4 Selección del tipo de identificación

Una vez se ha realizado el pre-procesamiento de las señales, se realiza la identificación.

Se han implementado dos tipos de identificación:

- Identificación ARX: Análisis temporales y frecuenciales.
- Identificación Frecuencial: Análisis frecuencial.

El Software determina qué tipo de identificación se puede realizar según la señal de excitación seleccionada, es decir, si se selecciona:

- Señal Step: Identificación con algoritmo ARX.
- Señal Uniform: Identificación con algoritmo ARX o frecuencial.
- Señal Multisine: Identificación con algoritmo ARX o frecuencial.

14.4.1 Orden del sistema

El orden del sistema define la ecuación que se utiliza para modelar la planta:

$$G(z) = \frac{Y(z)}{U(z)} = \frac{b_1 z^{-1} + b_2 z^{-2} + \cdots + b_n z^{-n}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_n z^{-n}} = \frac{B(z)}{A(z)} \qquad \textbf{Ecu. 17}$$

En el software que se ha desarrollado se debe especificar el orden del polinomio B(z) y A(z). Cuanto mayor sea el orden del sistema, mejor será la identificación. Aunque hay que tener en cuenta que cuanto de mayor orden sea el modelo, se hace más difícil de implementar el controlador.

Existen algoritmos que son capaces de encontrar el orden óptimo del sistema como el operador y matriz de Hankel.

Para indicar el orden el Software se debe insertar el orden del polinomio A y del polinomio B. Véase Figura 21.

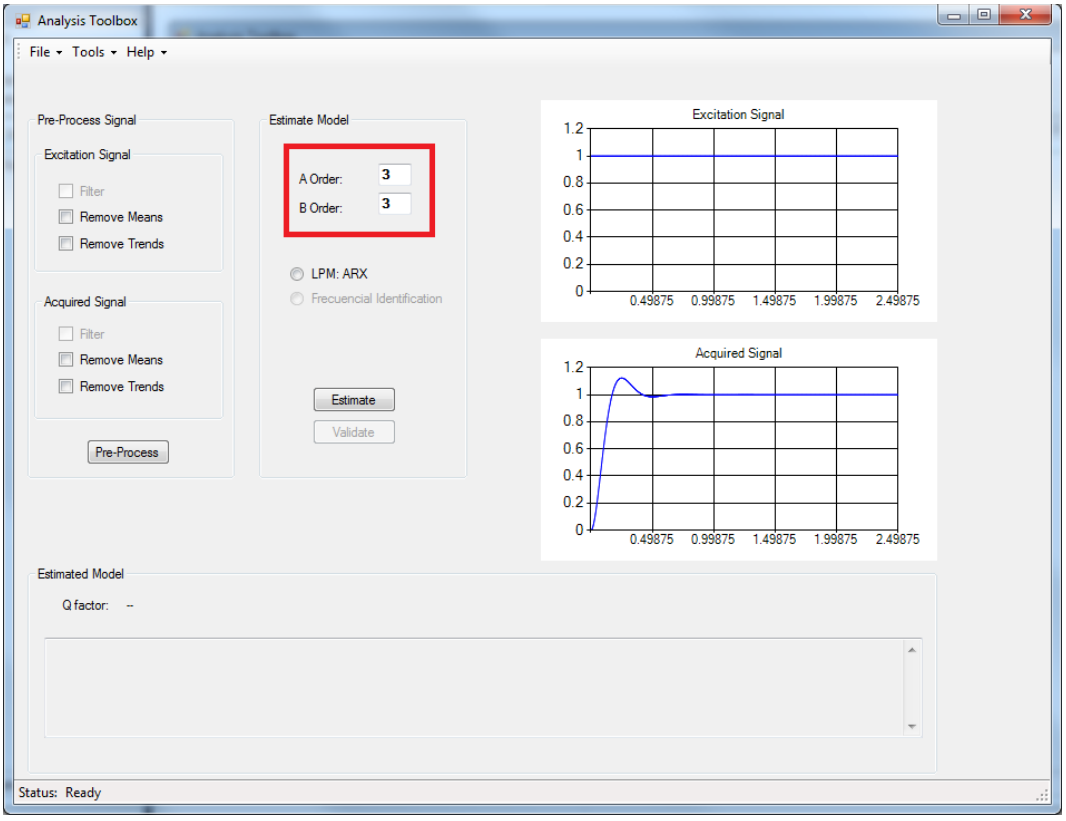


Figura 21: Orden del sistema

14.4.2 ARX (AutoRegresive model)

El algoritmo ARX o *AutoRegresive model* implementa una identificación que se basa en la minimización del error predicho. Esta identificación es de forma paramétrica, por tanto el modelo obtenido es más semejante a la planta cuanto más datos hallan en la identificación.

Este algoritmo utiliza el siguiente criterio para minimizar el error:

$$\min(V) = \min\left(\frac{1}{N}\sum_{t=1}^{N} e(t)^2\right) \qquad \textbf{Ecu. 18}$$

Un sistema se puede definir a través de la siguiente ecuación:

$$\mathrm{y(t)} = \varphi^{T}(\mathrm{t})\theta + \mathrm{v(t)} \qquad \textbf{Ecu. 19}$$

$$\varphi^{T}(\mathrm{t}) = [-\mathrm{y(t-1)} - \mathrm{y(t-2)} - \cdots \\ -\mathrm{y(t-na)}\ \mathrm{u(t-1)}\ \mathrm{u(t-2)}\ \mathrm{u(t-nb)} \qquad \textbf{Ecu. 20}$$

$$\theta = [a_1\ a_2\ a_{na}\ b_1\ b_2\ b_{na}] \qquad \textbf{Ecu. 21}$$

Siendo:

Y(t) = Respuesta del sistema.

$\varphi^{T}(\mathrm{t}) =$ Datos experimentales.

na = Orden del polinomio A(z)

nb = Orden del polinomio B(z)

$\theta =$ Constantes del modelo.

Si expresamos las ecuaciones de forma matricial, la respuesta del sistema sería la siguiente:

$$\mathrm{Y} = \Phi\Theta + \upsilon \qquad \textbf{Ecu. 22}$$

$$\mathrm{Y} = [y(t_1)\ y(t_2) \ldots\ y(t_N)] \qquad \textbf{Ecu. 23}$$

$$\Phi^{\mathrm{T}} = [\varphi(t_2)\ \varphi(t_2) \ldots\ \varphi(t_N)] \qquad \textbf{Ecu. 24}$$

$$\Phi^{\mathrm{T}} = \begin{bmatrix} -y(t_1-1) & & -y(t_1-na) & u(t_1-1) & & u(t_1-nb) \\ -y(t_2-1) & \dots & -y(t_2-na) & u(t_2-1) & \cdots & u(t_2-nb) \\ \vdots & & \vdots & \vdots & & \vdots \\ -y(t_N-1) & \dots & -y(t_N-na) & u(t_N-1) & \dots & u(t_N-nb) \end{bmatrix} \quad \textbf{Ecu. 25}$$

Si se aplica el algoritmo LTS[17] a la ecuación anterior para estimar las constantes del modelo se determina que:

$$\hat{\theta} = [\Phi^{\mathrm{T}}\,\Phi]^{-1}\Phi^{\mathrm{T}}\mathrm{Y} \quad \textbf{Ecu. 26}$$

Como se puede apreciar mediante las ecuaciones anteriores, este método no es recursivo, por tanto no se compone de bucles infinitos[18]. Por tanto los pasos a seguir para la programación son:

- Crear matriz Y.
- Crear matriz Φ.
- Realizar la operación $\hat{\theta} = [\Phi^{\mathrm{T}}\,\Phi]^{-1}\Phi^{\mathrm{T}}\mathrm{Y}$

El código que implementa el cálculo de los parámetros estimados es el siguiente:

[17] Least Squares Method: Algoritmo para determinar los parámetros que minimizan el error cuadrático.
[18] Según la norma IEEE 729 no se deben realizar iteraciones en programación con algoritmos que puedan implicar bucles infinitos.

```
//Variables
UINT32 u32MaxOrder = (UINT32)(max_Math((double)na, (double)nb));

//Create Yarx
double** dYarx = createMatrix(u32TotalLength - u32MaxOrder, 1);
for (int k = 0; k < (u32TotalLength - u32MaxOrder); k++)
{
	dYarx[k][0] = d_y[k + u32MaxOrder];
}

//Create Xarx
double** dXarx = createMatrix(u32TotalLength - u32MaxOrder, (na +
nb));
for (int k = 0; k < (u32TotalLength - u32MaxOrder); k++)
{
	for (int IdxTh = 0; IdxTh < (na + nb); IdxTh++)
	{
		if ( IdxTh < na )
		{
			dXarx[k][IdxTh] = -1*d_y[k - IdxTh +
u32MaxOrder - 1];
		}
		else
		{
			dXarx[k][IdxTh] = d_u[k - IdxTh + u32MaxOrder +
na - 1];
		}
	}
}

//(Xarx'*Xarx)
double** dXarxT = createMatrix((na + nb), u32TotalLength -
u32MaxOrder);
dXarxT = MatrixTranspose(dXarx,u32TotalLength - u32MaxOrder, (na +
nb));
```

```
double** dXarxT_x_dXarx = createMatrix((na + nb), (na + nb));
dXarxT_x_dXarx = MatrixMultiplication(dXarxT, dXarx, (na + nb),
u32TotalLength - u32MaxOrder,u32TotalLength - u32MaxOrder, (na +
nb));

//pinv(Xarx'*Xarx)
double** Inv_dXarxT_x_dXarx = createMatrix((na + nb), (na + nb));
Inv_dXarxT_x_dXarx = MatrixPseudoInverse(dXarxT_x_dXarx, (na + nb),
(na + nb));

//pinv(Xarx'*Xarx)*Xarx'
double** dAux = createMatrix((na + nb),  u32TotalLength -
u32MaxOrder);
dAux = MatrixMultiplication(Inv_dXarxT_x_dXarx, dXarxT, (na + nb),
(na + nb), (na + nb), u32TotalLength - u32MaxOrder);

//pinv(Xarx'*Xarx)*Xarx'*Yarx
double** dTheta = createMatrix((na + nb), 1);
dTheta = MatrixMultiplication(dAux, dYarx, (na + nb),
u32TotalLength - u32MaxOrder, u32TotalLength - u32MaxOrder,1);

DeleteMatrix (dYarx, u32TotalLength - u32MaxOrder);
DeleteMatrix (dXarx, u32TotalLength - u32MaxOrder);
DeleteMatrix (dXarxT, (na + nb));
DeleteMatrix (dXarxT_x_dXarx, (na + nb));
DeleteMatrix (Inv_dXarxT_x_dXarx, (na + nb));
DeleteMatrix (dAux, (na + nb));

return dTheta;
```

14.4.3 Identificación frecuencial

La identificación frecuencial se basa en transformar las señales temporales en señales frecuenciales y después estimar los parámetros del modelo. La identificación frecuencial se podrá dividir en dos partes:

- Cálculo de señal temporal a frecuencial.
- Estimación de coeficientes del modelo.

El modelo resultante de la identificación frecuencial es mejor cuantas más frecuencias contenga la señal de excitación.

14.4.3.1 Cálculo de la señal frecuencial

Para el cálculo de la señal frecuencial se utiliza el algoritmo DFT (véase apartado *DFT* para mayor información). Los pasos son:

- Calcular la DFT de la señal de excitación.
- Calcular la DFT de la señal de respuesta.

- Realizar la división frecuencial entre señal de respuesta y señal de excitación.

El código que implementa esta función es el siguiente:

```
//TFreqResult FreqResult;
TFreqResult *FreqResult;
FreqResult = new TFreqResult [n];

double *dBVectorOutRe;
dBVectorOutRe = new double [n];
double *dBVectorOutIm;
dBVectorOutIm = new double [n];
double *dAVectorOutRe;
dAVectorOutRe = new double [n];
double *dAVectorOutIm;
dAVectorOutIm = new double [n];

DFT_Complex (0, n, dB, dBVectorOutRe, dBVectorOutIm);
DFT_Complex (0, n, dA, dAVectorOutRe, dAVectorOutIm);

std::complex <double> *dBVectorOut;
dBVectorOut = new std::complex <double> [n];
std::complex <double> *dAVectorOut;
dAVectorOut = new std::complex <double> [n];
std::complex <double> *dH;
dH = new std::complex <double> [n];

double HrealNote[2000];
double HimagNote[2000];
double WNote[2000];

for (int i = 0; i < n; i++)
{
	dBVectorOut[i].real(dBVectorOutRe[i]);
	dBVectorOut[i].imag(dBVectorOutIm[i]);
	dAVectorOut[i].real(dAVectorOutRe[i]);
	dAVectorOut[i].imag(dAVectorOutIm[i]);

	dH[i] = dBVectorOut[i] / dAVectorOut [i];

	FreqResult[i].HReal = dH[i].real();
	FreqResult[i].HImg = dH[i].imag();
}

for (int i = 0; i < n; i++)
{
	FreqResult[i].w = i*PI/n;
}

delete [] dBVectorOut;
delete [] dAVectorOut;
delete [] dH;
delete [] dBVectorOutRe;
delete [] dBVectorOutIm;
delete [] dAVectorOutRe;
delete [] dAVectorOutIm;
```

14.4.3.2 Cálculo de coeficientes del modelo

Una vez se ha obtenido la respuesta frecuencial del sistema se debe proceder a la estimación de los parámetros del modelo.

El algoritmo consiste en minimizar el método de la ecuación de error en el dominio de la frecuencia. Dado el espectro $H(e^{j\omega_k})$ con unas frecuencias espaciadas $\omega_k = 2\pi k/N$.

$$\widehat{H}(z) \triangleq \frac{\hat{B}(z)}{\hat{A}(z)} \triangleq \frac{\sum_{k=0}^{nb} b_k z^{-k}}{\sum_{k=0}^{na} a_k z^{-k}} \qquad \textbf{Ecu. 27}$$

Dónde:

nb = Orden de polinomio B.

na = Orden de polinomio A.

Si normalizamos $a_0 = 1$:

$$J_E^2 = \sum_{k=0}^{N-1} \left|\hat{A}(e^{j\omega_k})Y(e^{j\omega_k}) - \hat{B}(e^{j\omega_k})U(e^{j\omega_k})\right|^2 \qquad \textbf{Ecu. 28}$$

Dónde:

$Y(e^{j\omega_k}) =$ Respuesta frecuencial del sistema.

$U(e^{j\omega_k}) =$ Señal de excitación frecuencial.

N = Número total de muestras.

Si resolvemos:

$$\underline{R}_{uu}\,(n) \triangleq DFT^{-1}\left|U\left(e^{j\omega_k}\right)\right|^2 \qquad \textbf{Ecu. 29}$$

$$\underline{R}_{yy}\,(n) \triangleq DFT^{-1}\left|Y\left(e^{j\omega_k}\right)\right|^2 \qquad \textbf{Ecu. 30}$$

$$\underline{R}_{yu}\,(n) \triangleq DFT^{-1}\left(Y\left(e^{j\omega_k}\right)\overline{U(e^{j\omega_k})}\right) \qquad \textbf{Ecu. 31}$$

$$n = 0,1,\dots,N-1 \qquad \textbf{Ecu. 32}$$

Si aplicamos las matrices de Toeplitz[19]:

$$Ruu \triangleq T\left(\underline{R}_{uu}[\,0{:}\,nb]\right) \qquad \textbf{Ecu. 33}$$

$$R_{yy} \triangleq T\left(\underline{R}_{yy}[\,0{:}\,na-1]\right) \qquad \textbf{Ecu. 34}$$

$$R_{yu} \triangleq T\left(\underline{R}_{yu}[-1{:}\,nb-1], \underline{R}_{yu}^{T}[-1{:}\,-na]\right) \qquad \textbf{Ecu. 35}$$

$$R_{uy} \triangleq R_{yu}^{T} \qquad \textbf{Ecu. 36}$$

Donde la solución es:

$$\hat{\theta}^{*} = \begin{bmatrix}\underline{\hat{B}}^{*} \\ \underline{\hat{A}}^{*}\end{bmatrix} = \begin{bmatrix}R_{uu} & R_{uy} \\ R_{yu} & R_{yy}\end{bmatrix}^{-1}\begin{bmatrix}\underline{R}_{yu}[0{:}\,nb] \\ \underline{R}_{yy}[1{:}\,na]\end{bmatrix} \qquad \textbf{Ecu. 37}$$

[19] Para mayor información véase http://ee.stanford.edu/~gray/toeplitz.pdf

$$\underline{\hat{B}}^* \triangleq \begin{bmatrix} \hat{b}_0^* \\ \vdots \\ \hat{b}_{nb}^* \end{bmatrix} \qquad \textbf{Ecu. 38}$$

$$\underline{\hat{A}}^* \triangleq \begin{bmatrix} \hat{a}_0^* \\ \vdots \\ \hat{a}_{nb}^* \end{bmatrix} \qquad \textbf{Ecu. 39}$$

El código que implementa en pseudocódigo es el siguiente:

```
  s = sqrt(-1)*F;
  if plane == 'z'
    if max(F)>pi || min(F)<0
      disp('hey, you frequency is outside the range 0 to pi, making
my own')
      F = linspace(0,pi,length(H));
      s = sqrt(-1)*F;
    end
    s = exp(-s);
  end;

  for k=1:nF
    Zk = (s(k).^(0:n)).';
    Hk = H(k);
    aHks = Hk*conj(Hk);
    Rk = (W(k)*Zk)*Zk';
    rRk = real(Rk);
    Ruu = Ruu+rRk(1:mB,1:mB);
    Ryy = Ryy+aHks*rRk(2:mA,2:mA);
    Ryu = Ryu+real(Hk*Rk(2:mA,1:mB));
    Pu = Pu+W(k)*real(conj(Hk)*Zk(1:mB));
    Py = Py+(W(k)*aHks)*real(Zk(2:mA));
    aHks_v(k) = aHks;
  end;

  R = [Ruu,-Ryu';-Ryu,Ryy];
  P = [Pu;-Py];
  Theta = R\P;

  B = Theta(1:mB)';
  A = [1 Theta(mB+1:mB+nA)'];
```

14.4.4 Resultados de la identificación

Una vez se han realizados las identificaciones, los resultados se muestran en forma de coeficientes en la pantalla de la Figura 22.

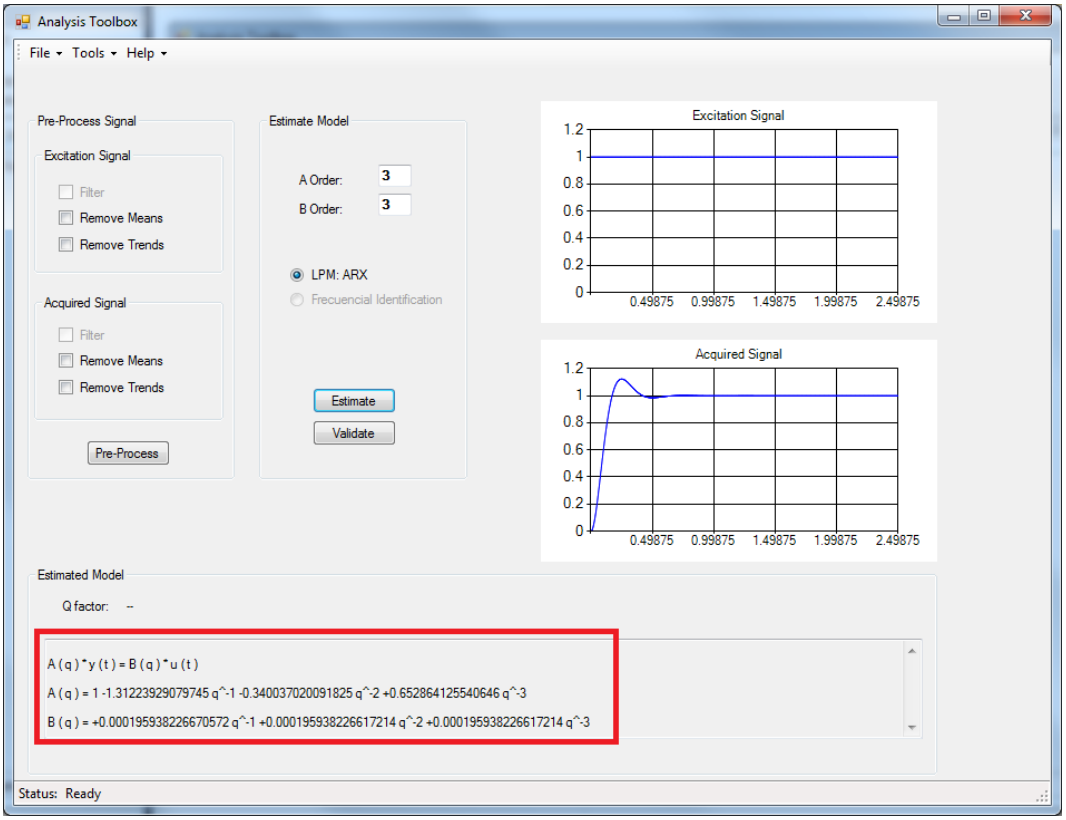


Figura 22: Modelo estimado

14.5 Verificación de la identificación

Para comprobar si la identificación se ha realizado de manera correcta, el programa incorpora una función que verifica el modelo obtenido. Se debe apretar en el botón *Validate (véase* Figura 23*).*

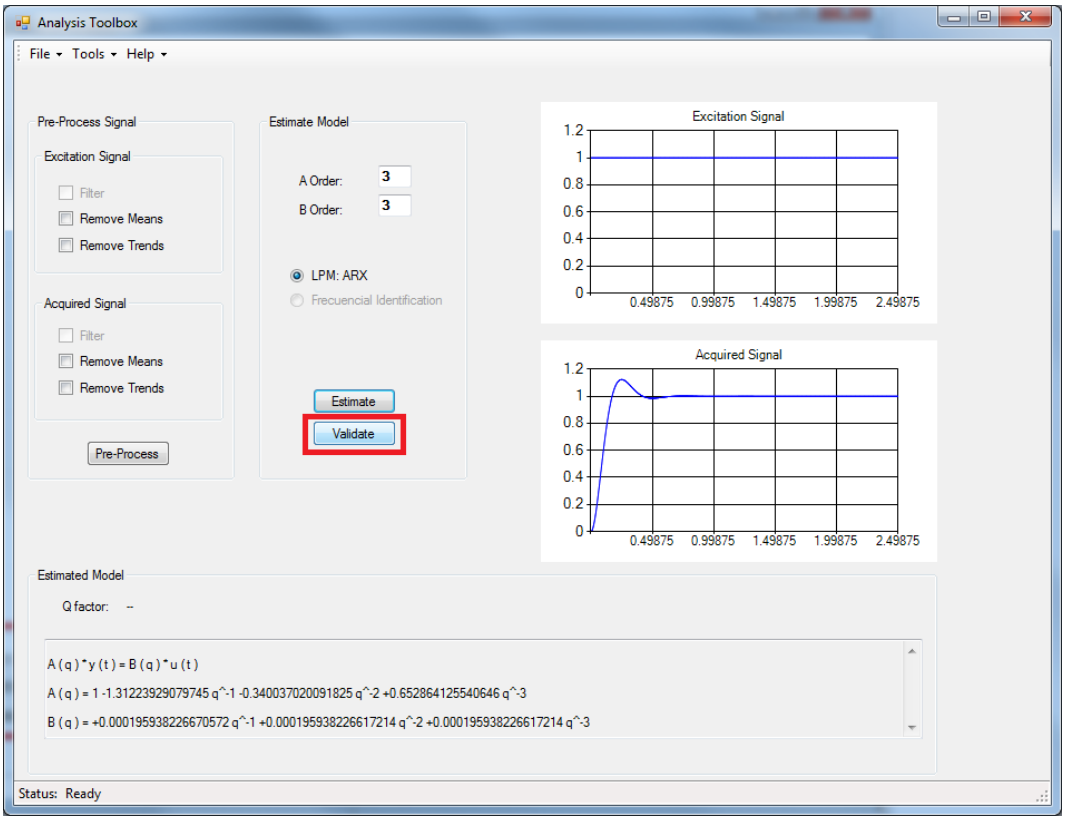


Figura 23: Validar el modelo

Los pasos que se realizan para la validación del modelo son los siguientes:

- Ejecución de un nuevo ensayo.
- Captura de parámetros del nuevo ensayo.
- Visualización de ensayo y modelo.
- Cálculo del Q factor.

14.5.1 Verificación visual

La verificación visual se realiza mediante la visualización en un gráfico de *Adquired Signal*. La señal capturada se muestra en color azul, sin embargo la señal estimada se muestra en color verde. Véase Figura 24.

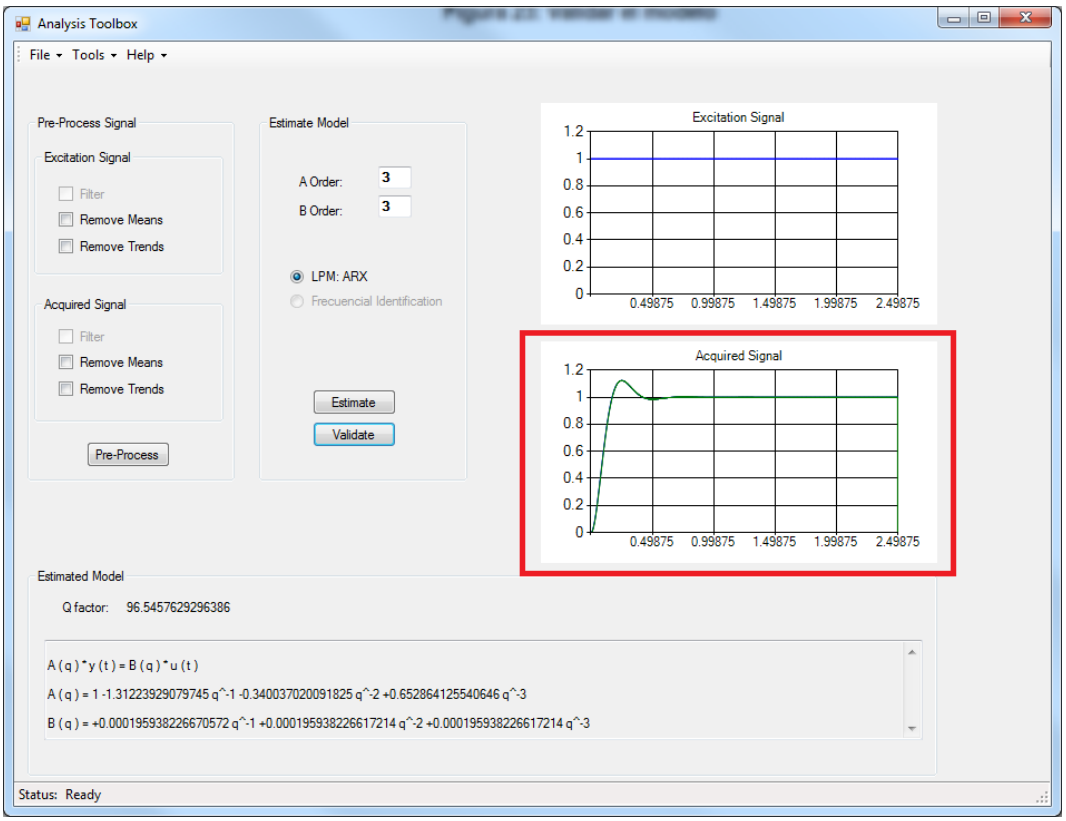


Figura 24: Validación visual del modelo.

Para poder estimar la respuesta del modelo a partir de los coeficientes del modelo estimado se implementa de la siguiente manera:

```
double** dYest = createMatrix(u32size,1);

for (int i=(int)(max_Math(na,nb)); i< (u32size); i++)
{
	double dB = 0;
	for (int k1 = 0; k1 < max(na,nb); k1++)
	{
		if (k1 == 0)
		{
			dB += d_Theta[na][0];
		}
		else
		{
			dB += d_Theta[na + k1][0] *
dPrePExitationSignal [i - (k1 - na)] ;
		}
	}
```

```
        double dA = 0;
        for (int k2 = 0; k2 < na; k2++)
        {
                dA += d_Theta[k2][0] * dYest [i - k2 - 1][0] ;
        }

        dYest[i][0] = dB - dA;
}

return dYest;
```

14.5.2 Cálculo del factor Q

Para poder evaluar el modelo calculado se utiliza una evaluación por factor de calidad NRMSD[20].

$$Q = 100\left(1 - \frac{\sqrt{\frac{\sum_{t=1}^{n}(y_t - \hat{y}_t)^2}{n}}}{|\max(y,\hat{y}) - \min(y,\hat{y})|}\right)$$ **Ecu. 40**

El código que implementa esta función es el siguiente:

```
        //NRMSD Algorithm
        for (UINT32 i = 0; i < u32size; i++)
        {
                dRMSD += (dYreal[i] - dYest[i][0]) * ((dYreal[i] -
                dYest[i][0]));
                if (dYest[i][0] > dXmax) dXmax = dYest[i][0];

                if (dYreal[i] > dXmax) dXmax = dYreal[i];

                if (dYest[i][0] < dXmin) dXmin = dYest[i][0];

                if (dYreal[i] < dXmin)dXmin = dYreal[i];

        }
        dRMSD /= u32size;
        dRMSD = std::sqrt(dRMSD);

        dNRMSD = dRMSD/ (dXmax - dXmin);

        //return dfit;
        return (100 - dNRMSD*100);
```

[20] *Normalized Root Mean Square Deviation*

15. Ensayo de sistema

Para verificar que el dispositivo funciona adecuadamente se deben realizar dos tipos de ensayos:

- Ensayo en simulación.
- Ensayo real.

15.1 Ensayo simulación

En la simulación se ha generado un modelo a partir de Matlab y se han introducido los parámetros en el Software desarrollado.

En la simulación se ha realizado 3 pasos:

- Generación de una señal adquirida y de excitación.
- Análisis de las señales.
- Verificación de las señales.

Esta simulación se realiza con el fin de poder evaluar los algoritmos desarrollados y lo puede realizar el usuario sin necesidad de tener el dispositivo. Para poder escoger esta opción se debe seleccionar en la señal de excitación el botón *Simulation. Véase* Figura 25.

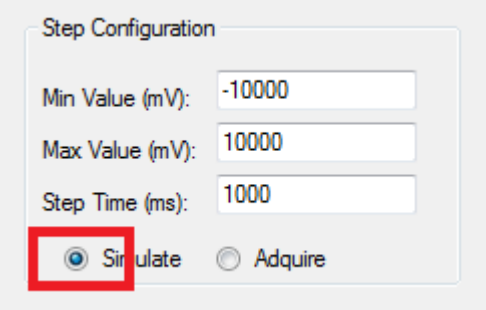


Figura 25: Simulación de algoritmos de identificación

Si realizamos la identificación mediante simulación puede apreciarse el siguiente ensayo capturado:

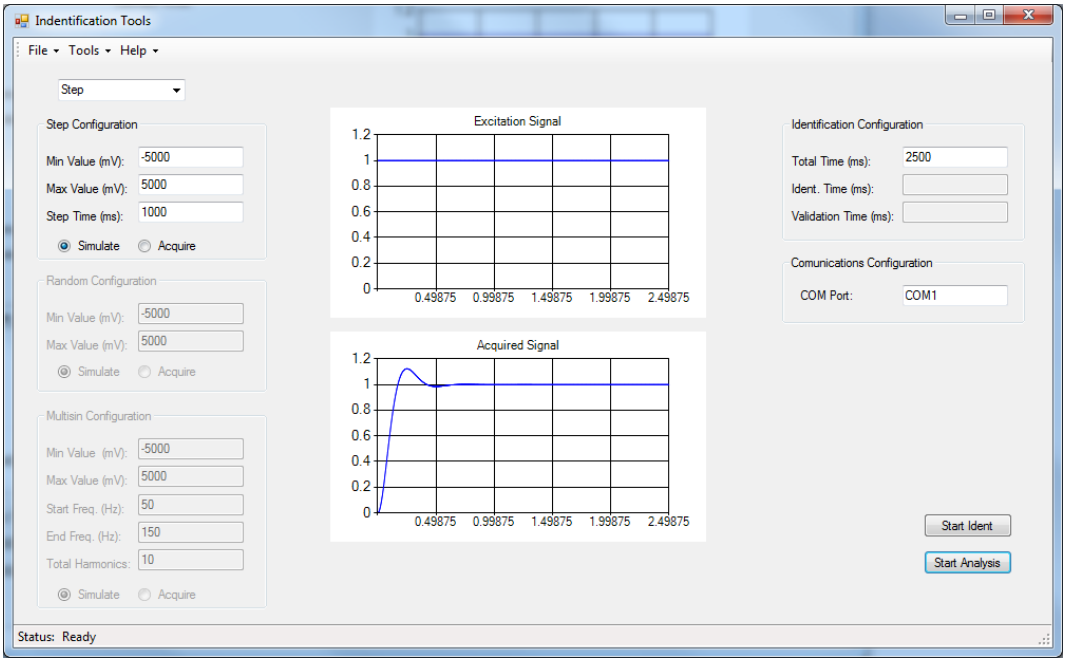


Figura 26: Lectura de la identificación

Los resultados del ensayo son los siguientes:

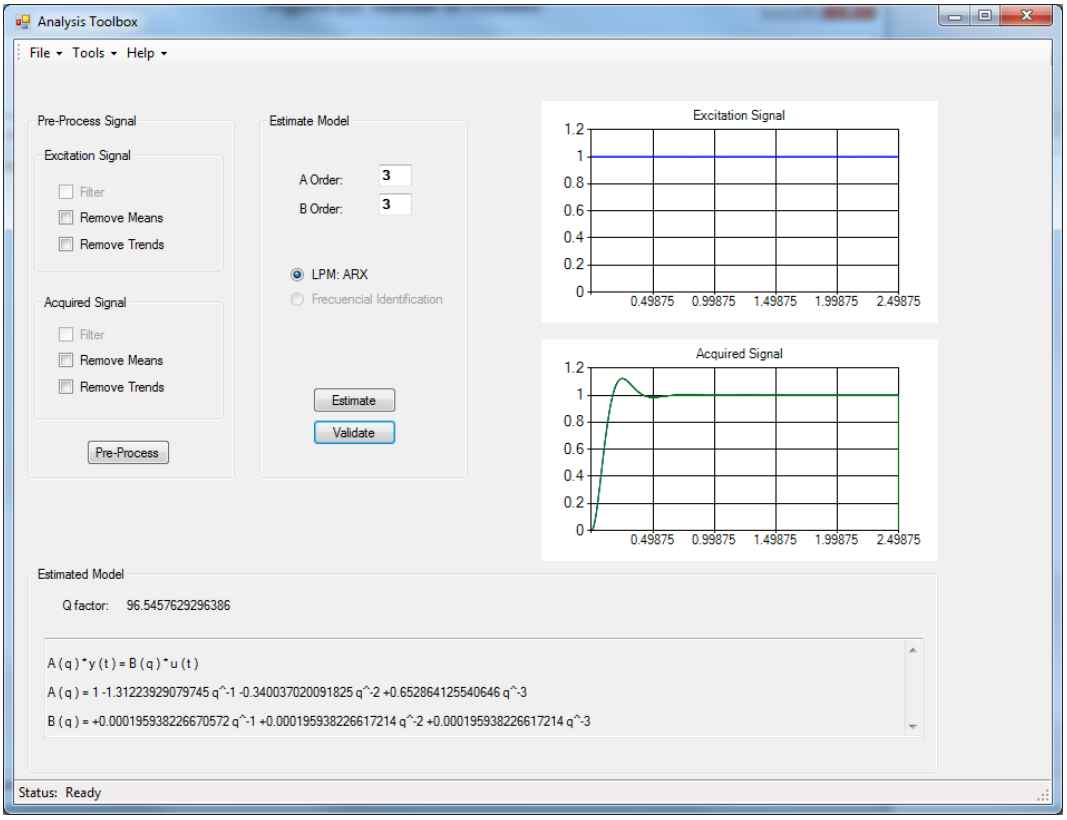


Figura 27: Resultado del ensayo

15.1 Ensayo real

Para poder ensayar el dispositivo desarrollado se ha implementado un montaje con un condensador y una resistencia (circuito RC). De esta manera se podrá ver la respuesta en voltaje frente a un escalón. El circuito utilizado es el que se puede ver en la Figura 28.

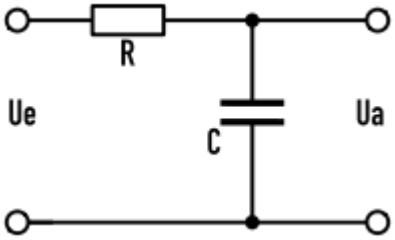


Figura 28: Montaje para ensayo

Se ha modelado un circuito que tenga un tiempo de respuesta de 1s, permitiendo al dispositivo capturar la carga del condensador.
Este modelo se ha diseñado de la siguiente manera:

Si C = 10 µC y tiempo de respuesta $\tau = 1\,s$, entonces R = 100 kΩ. Véase la siguiente ecuación.

$$\tau = R \cdot C \qquad \textbf{Ecu. 41}$$

Para poder escoger esta opción se debe seleccionar en la señal de excitación el botón *Adquire (véase* Figura 29*).* Nótese que para realizar este ensayo se debe tener el dispositivo conectado.

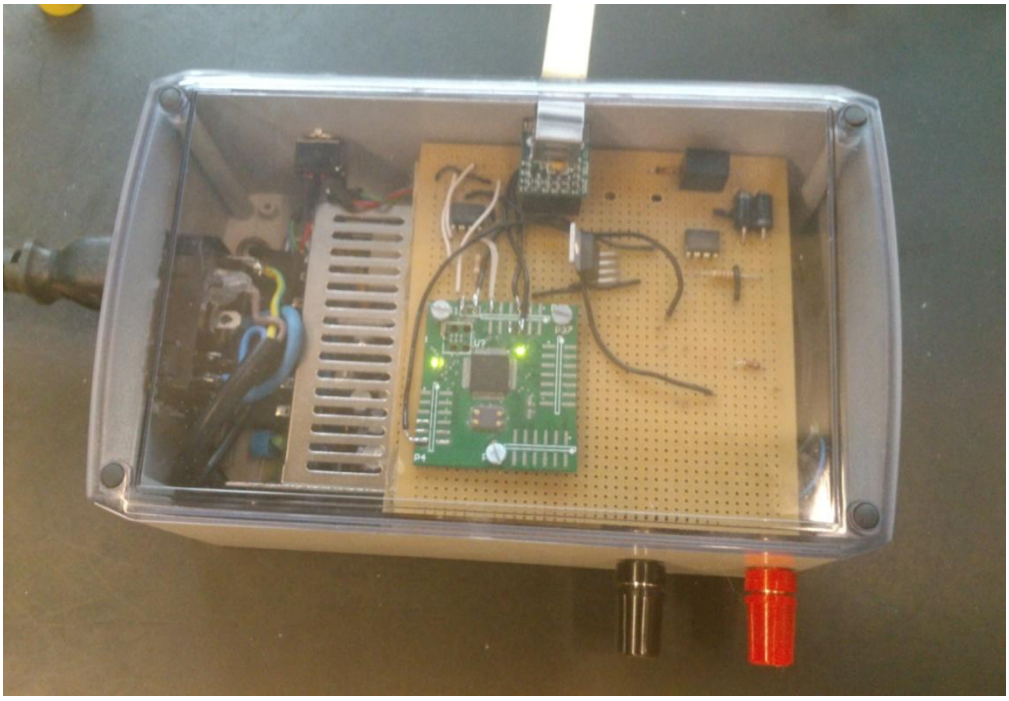


Figura 29: Montaje del ensayo

La captura de datos que ha realizado el ensayo es la siguiente:

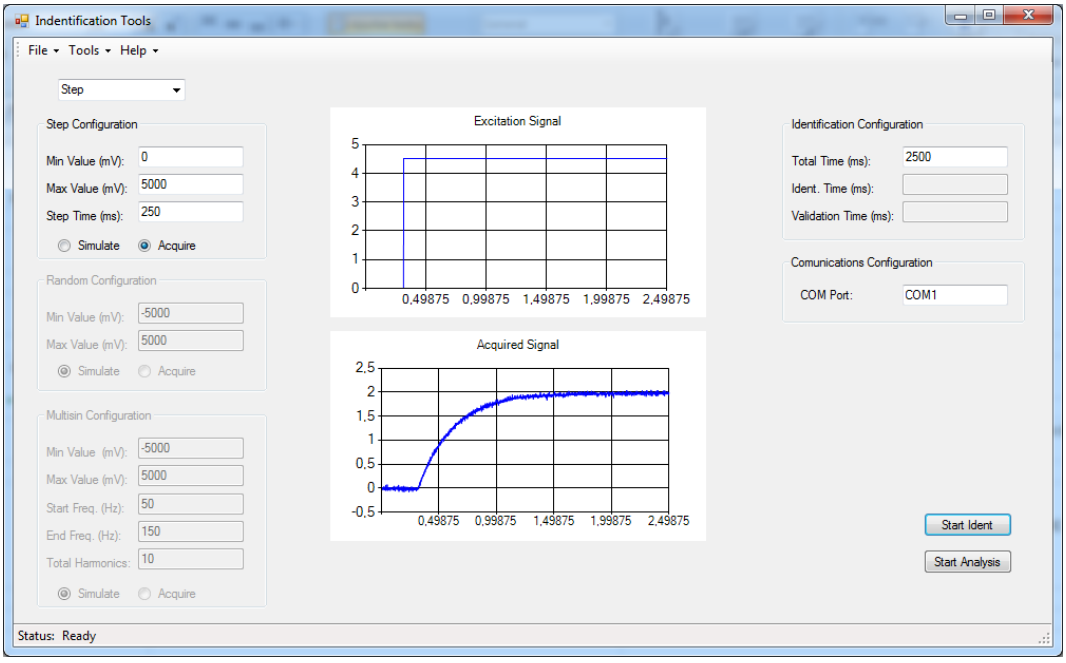


Figura 30: Lectura de la identificación.

Los resultados del ensayo son los siguientes:

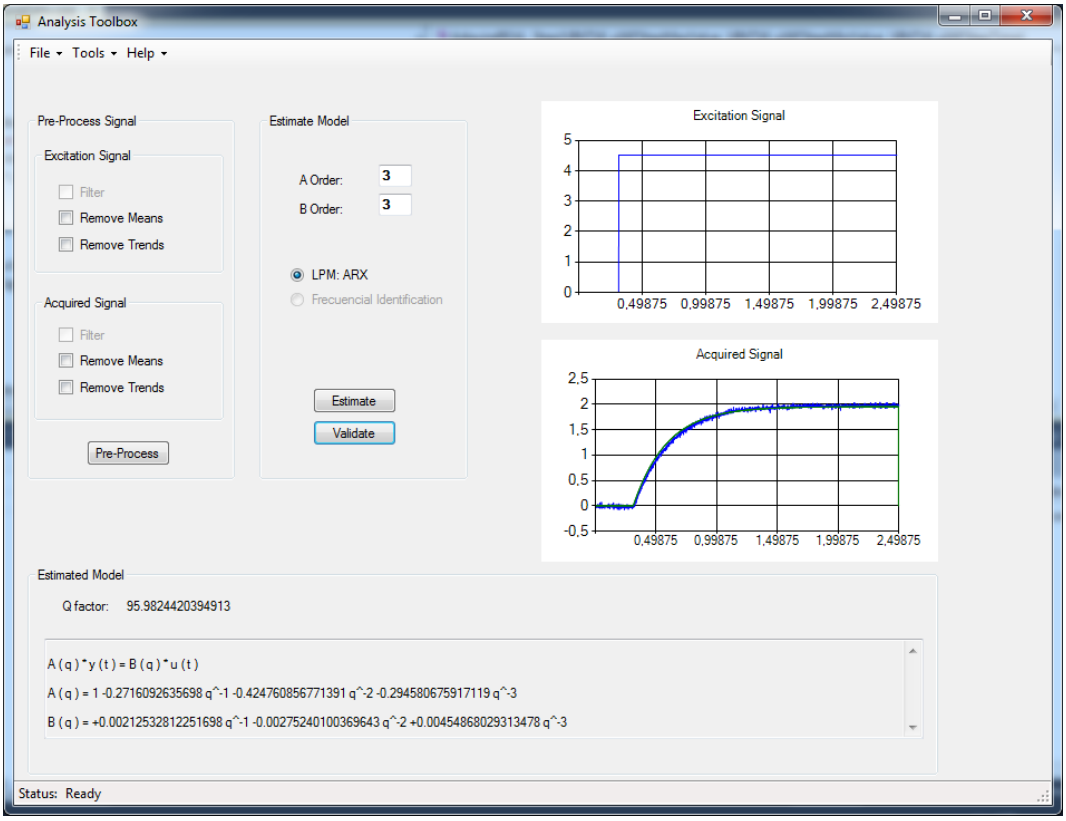


Figura 31: Resultados de la identificación

16. Presupuesto

16.1 Costes de ingeniería

	Horas	€/hora	Total
Diseño	670	60	40560
Planos	10	45	450
Memoria	60	30	1800

Total: 42810€

16.2 Costes indirectos

Los gastos indirectos corresponden a los desplazamientos o dietas y se elige un 5% normalmente. También a los gastos que puede haber en material de oficina, fotocopias, etc. Se suele elegir un 2 %, es decir un total de un 7 %.

Total: 2997€

16.1 Costes de materiales

Module	Description	Manufacturing ID	Codigo Proveedor	Min. Cantidad	Cantidad Unitaria	Precio Unitario	Batch (1)	Cantidad montada	Cantidad total	Total
Analog Output	OP AMP, 8-60V, 3A, 1MHZ, 10V/US, 7TO220	OPA548T	1564885	1	1	13,180	25	25	25	329,500
	CAP, CERAMIC, 10UF, 16V, Y5V, 1206	MCCA000564	1759445RL	10	2	0,074	25	50	50	3,700
	TAPA, CERÁMICA, 0,1UF, 16V, X7R, 0603	C0603C104K4RACTU	1414610RL	10	2	0,012	25	50	50	0,600
	MLCC, 0603, 10.000PF, 100 V, X7R, 10%	CC0603KRX7R0BB103	1284119RL	1	4	0,020	25	100	100	2,000
	IC, SM, TENSIÓN REF, 3,3 V	ZRC330F03TA	1132713RL	1	1	0,770	25	25	25	19,250
	THIN FILM CHIP RESISTOR 20 kohm	RR0816P-203-B-T5	1653264RL	1	2	0,114	25	50	50	5,700
	RESISTOR, 0603, 27K0, 1%	CRCW060327K0FKEA	1652864	1	1	0,070	25	25	25	1,750
	RESISTOR, 0603, 0.17W, 54K9, 0.1%, 25PPM	RP73PF1J54K9BTDF	2116863	5	1	0,240	25	25	25	6,000
	THICK FILM CHIP RESISTOR 10 kohm	CRCW060310K0FKTA	1652827	1	4	0,101	25	100	100	10,100
	TERMINAL BLOCK, PCB, 2.54MM, 2WAY	1725656	3041359	1	1	1,000	25	25	25	25,000
	LM358N - AMPLIFICADOR OPERACIONAL, BAJA POTENCIA DUAL, DIP8, 358	LM358N	9755900	1	1	0,200	25	25	25	5,000
Analog Input	DIODO, TVS, SOD-523	PESD5V0S1BB	8737703RL	5	1	0,122	25	25	25	3,050
	THIN FILM CHIP RESISTOR 20 kohm	RR0816P-203-B-T5	1653264RL	1	1	0,114	25	25	25	2,850

	THICK FILM CHIP RESISTOR 10 kohm	CRCW060310K0FKTA	1652827	1	1	0,101	25	25	25	2,525
	RESISTOR, 26R7, 0805 0.1% 25PPM 0.1W	MCTC0525B267JT5E	1575455RL	5	1	0,059	25	25	25	1,475
	RESISTOR, 0603, 1% 50PPM 3K3	RT0603FRE073K3L	1500623	5	1	0,037	25	25	25	0,925
	LM358N - AMPLIFICADOR OPERACIONAL, BAJA POTENCIA DUAL, DIP8, 358	LM358N	9755900	1	1	0,200	25	25	25	5,000
Power Supply	CONVERTIDOR CC/CC, 2W, O/P INDIVIDUAL	IL1203S	2084319	1	1	6,62	25	25	25	165,500
	DIODE,RECTIFIER,3A,600V, DO-201	1N5406	1467459	1	2	0,039	25	50	50	1,950
	TERMINAL BLOCK, PCB, 2.54MM, 3WAY	1725669	3041360	1	1	1,52	25	25	25	38,000
	DIODO, TVS, SOD-523 12V	PESD12VS1UB	8737371RL	5	2	0,101	25	50	50	5,050
	Fuente alim entrada universal,+/-12V 35W	TXL 035-1212D	466-6862	1	1	44,8	25	25	25	1120,00
Board	LABOR CARD, FR2, RM 2, 50 mm	RE100-HP	1837349	1	1	5,92	25	25	25	148,000
USB-232	Eval Kit module USB-UART FT232RL MM232R	MM232R	406-562	1	1	17,48	25	25	25	437,000
	LED, SMD, GREEN, 15MCD, 572NM	HSMG-C190	8554609	5	3	0,19	25	75	75	14,250
	RESISTOR, POWER, 0603, 1%, 10R0	CRCW060310R0FKEAHP	1738878	25	1	0,098	25	25	25	2,450
Core	RESISTOR, THIN FILM	RR0816P-202-D	1612043	1	2	0,077	25	50	50	3,850

	RESISTOR, POWER, 0603, 1%, 10K0	CRCW060310K0FKEAHP	1738918	25	1	0,098	25	25	25	2,450
	RT0603FRE07470RL - RESISTOR, 0603, 1% 50PPM 470R	RT0603FRE07470RL	1500635	5	1	0,072	25	25	25	1,800
	CPF0603F75RC1 - RESISTOR, 0603, 75R 1% 50PPM	CPF0603F75RC1	1527343	5	5	0,042	25	125	125	5,250
	PIC32MX775F256H-80I/PT - MCU, 32BIT, 256K FLASH, USB, 64TQFP	PIC32MX775F256H-80I/PT	1778978	1	1	10,12	25	25	25	253,000
	12SMX (B) 30/5020/16 8.0 MHZ - CRYSTAL, 12SMX (B), 8.0MHZ, SMD	12SMX (B) 30/5020/16 8.0 MHZ	1276690	1	1	2,28	25	25	25	57,000
	CAP MLCC, 18PF, 50V, C0G/NP0, 5%, 0603	06035A180JAT2A	1612182	1	2	0,112	25	50	50	5,600
	CAP MLCC, 1000PF, 50V, C0G/NP0, 5%, 0603	06035A102JAT2A	1650827	10	5	0,43	25	125	130	55,900
	CAP MLCC, 0.1UF, 50V, X7R, 10%, 0603	C0603C104K5RAC-TU / 06035C104KAT2A	1692286 / 1301804RL	5	4	0,102	25	100	100	10,200
	CAPACITOR MLCC, 10UF, 10V, 10%, 1206	LMK316BJ106KL-T / C1206C106K8PACTU	1650933 / 9227890	10	1	0,16	25	25	30	4,800
	IC, EEPROM, 256K, I2C, 2.5V, SOIC8	24FC256-I/SN	1579572	1	1	1,51	25	25	25	37,750

Batch	Precio / Disp.	Precio Total
1	123,54	123,54
10	111,945	1119,45
25	111,769	2794,225
50	111,769	5588,45

16.2 Costes de realización del montaje y verificación de producción

Batch	Horas/ dispositivo	€/hora	Precio Total
1	2	15	30
10	2	15	300
25	2	15	750
50	2	15	1500

16.3 Costes totales y conclusiones

Se puede ver que la cantidad más elevada es la del coste de ingeniería, pero como la aplicación es industrial, se estimará que habrá un batch productivo de 25 dispositivos, por tanto el coste de ingeniería se dividirá.

Es por ello que si no hiciéramos la tirada de 25 productos el coste total sería:

$$\text{Coste Dispositivo} = \frac{\text{C. Ingenieria} + \text{C. Indirectos} + \text{C. Materiales} + \text{C. Montaje}}{25}$$

$$\text{Coste Dispositivo} = \frac{42810€ + 2997€ + 2794€ + 750€}{25}$$

Total: 1974€ Precio sin IVA

Si se aplica un margen de beneficio de un 30%:

Total: 2820€ Precio sin IVA

17. Viabilidad económica

A continuación se presentan los criterios que se han tenido en cuenta para realizar los estudios de viabilidad económica de cada propuesta:

- **Inversión:** Se ha asumido que por el hecho de utilizar una identificación automática en vez de un ensayo manual se ahorra un 30% en el tiempo ingeniero.
- **Tasa de actualización:** Se ha tenido en cuenta un 4%.
- **TIR:** La tasa de rentabilidad de todas las inversiones se ha calculado a 10 años.
- **Coste de precio ingeniero:** Se ha estimado que el incremento anual del precio ingeniero es de un 3%.
- **Coste de mantenimiento:** Se estiman que son los mismos que actualmente.
- **Vida útil:** Se han tenido en cuenta inversiones en el año del final de la vida útil de los productos.

$$\text{Inversión} = \text{Precio producto} * \text{Total Producto}$$

$$\text{Inversión} = 2820€ * 25 = 70.500€$$

$$\text{Total Ahorro } (€) = \text{Ahorro} * \text{UsosDispositivo} * \text{TotalProducto} * \text{CosteIngeniero}$$

$$\text{Total Ahorro } (€) = 0.2 * 50 * 25 * 60 = 15000$$

	Año	Año	Año	Año	Año	Año
	0	1	2	3	4	5
TIR	-70.500 €	15000	15000	15000	15000	15000
VAN	-70.500 €	14855,7692	14284,3935	13734,9937	13206,7248	12698,7738
	Año	Año	Año	Año	Año	
	6	7	8	9	10	
TIR	15000	15000	15000	15000	15000	
VAN	12210,3594	11740,7302	11289,1637	10854,9651	10437,4664	

TIR	17%
VAN	54.813 €

Como tiene un alto TIR se puede apreciar que la inversión de este producto es viable.

18. Conclusiones

Como se ha podido apreciar a lo largo de esta memoria, en este proyecto se ha estudiado en profundidad el funcionamiento de un sistema completo de identificación, una solución que permitirá en un futuro integrar la identificación con el control de procesos industriales.

Los resultados que se han obtenido demuestran que esta arquitectura es funcional, flexible y modular, dotando a este dispositivo de una gran capacidad de ensamblaje en sistemas industriales. Esta modularidad es gracias a de que todos los programas implementados han sido escritos en un idioma estándar y por tanto este sistema no está ligado al Hardware desarrollado, pudiendo aumentar así las capacidades de la identificación y control fácilmente.

Se ha desarrollado a nivel experimental un dispositivo completo para poder ver la viabilidad técnica. Se ha observado que el Hardware es capaz de de generar las señales adecuadas con un error bajo y que por tanto el sistema implementado es idóneo para realizar identificaciones, sin embargo, podemos ver que el µC utilizado (PIC32) tiene bajas prestaciones, y por tanto no podemos almacenar más de 25 segundos de ensayo, limitando su uso en muchas aplicaciones.
Este dispositivo está en fase alpha, por tanto, no es capaz de pasar el certificado CE, ni comercializarse en un tiempo breve, sin embargo, demuestra que la arquitectura escogida es idónea para realizar ensayos de identificación en ámbitos industriales.
Además, la librería matemática desarrollada que permite la identificación frecuencial y paramétrica en línea ha funcionado correctamente en la identificación.

La implementación de este dispositivo ha instigado un contacto físico y real con el mundo profesional. Ha sido una carta de presentación a recursos y personas que se dedican a este sector, generando una afluencia de dudas que se han ido solventando poco a poco. Personalmente la realización de este proyecto me ha ayudado a afrontar a objetivos nuevos, complementándome en aspectos de rigor, conocimiento de marcar, etc.

19. Posibles mejoras

Actualmente este dispositivo es capaz de realizar una identificación paramétrica y frecuencial con diversas señales de excitación. La arquitectura de este dispositivo está pensada para poderse ampliar de diversas formas en el futuro. Sistema multinodo, es decir, que el protocolo de comunicaciones soporte diferentes nodos en una misma red y por tanto podrá permitir identificar/controlar diferentes procesos simultáneamente. Sistema con diferentes algoritmos de identificación, es decir, al haber desarrollado la librería matemática, es fácil implementar un algoritmo nuevo de identificación. Sistema con señal de excitación especificada por el usuario, es decir, actualmente se pueden elegir 3 tipos de señales de excitación, con esta opción el usuario podría elegir la señal que desee implementar. Sistema con capacidad de control, es decir, el sistema una vez haya realizado la identificación podría ajustar los parámetros para realizar el control.

Finalmente se puede concluir que con el desarrollo implementado se ha podido demostrar la viabilidad del proyecto y se han abierto varios caminos de investigación para determinar la dirección que podría tomar el producto. En mi opinión, para poder industrializar este producto, se debería implementar un sistema multinodo, con identificación y control en el propio dispositivo, dónde se pudiese escoger el tipo de control.

20. Bibliografía

[1] Kim Otten and Bud Caldwell (Microchip). AN1143 Generic Client Driver for a USB Embedded Host. 2009. Disponible en: <http://ww1.microchip.com/downloads/en/AppNotes/01143a.pdf>.

[2] Amardeep Gupta (Microchip). AN1247 Communication Device Class (CDC) Host. 2009. Disponible en: <http://ww1.microchip.com/downloads/en/AppNotes/01247a.pdf>

[3] Peter Hall. Central limit theorem for integrated square error of multivariate nonparametric density estimators. Australia, 1984m vol. 14, núm. 1, p 1 – 16.

[4] Sheldon Ross. A first Course in probability. 5º ed. August 17, 1997. ISBN 978-0137463145.

[5] Matt Donadio. Generate white Gaussian Noise. Disponible en: <http://www.dspguru.com/dsp/howtos/how-to-generate-white-gaussian-noise>

[6] Jesse B. Hoagg, Seth L. Lacy, Vit Babuska. Sequential Multisine Excitation Signals for System Identification of Large Space Structures. September 6, 2006.

[7] Okko H. Bosgra, Huibert Kwakernaak, Gjerrit Meinsma. Design Methods for Control Systems. Disponible en: <http://innocenti.dsea.unipi.it/kwak.pdf>

[8] Microchip. 32-bit Language Tools Libraries. Disponible en: <http://ww1.microchip.com/downloads/en/DeviceDoc/51685E.pdf>

[9] Leon van Dommelen. Linear Interpolation. Disponible en: <http://www.eng.fsu.edu/~dommelen/courses/eml3100/aids/intpol/>

[10] Douglas C. Montgomery, Elizabeth A. Peck and G. Geoffrey Vining. Introduction to Linear Regression Analysis. 5º ed. April 2012. ISBN 978-0470542811.

[11] Ljung, L (1999). System Identification: Theory for the User, 2º ed. Prentice Hall. ISBN 0-13-656695-2.

[12] Johansson, R (1993): System Modeling and Identification. Prentice-Hall, New Jersey.

[13] Levi, E.C., “Complex-Curve Fitting”, IRE Trans. On Automatic Control, Vol. AC-4 (1959), pp. 37-44.

[14] Dennis, J.E., Jr., and R.B. Schnable, Numerical Methods for Unconstrained Optimization and Nonlinear Equations, Prentice-Hall, 1983.

[15] Smith, Julius O. Introduction to Digital Filters with Audio Applications, W3K Publishing, http://books.w3k.org/, 2007, ISBN 978-0-9745607-1-7.

[16] Nicholas J. Higham. QR factorization with complete pivoting and accurate computation of SVD. April 200, vol 309, núm 1-3, p 153-174.

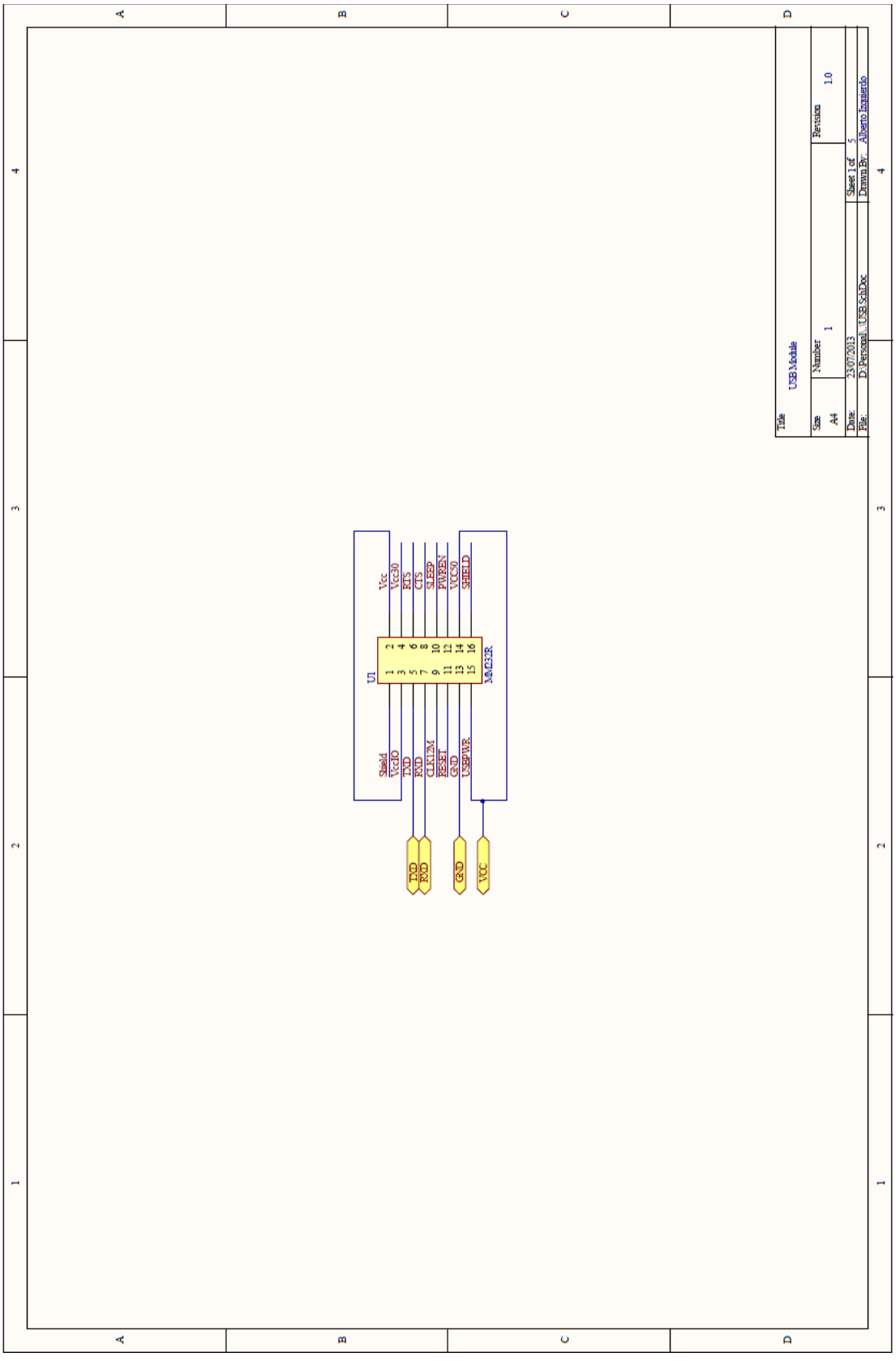
[17] *SIAM J. Sci. and Stat. Comput.*, The modified truncated SVD Method for Regularization in General Form, 13(5), 1142–1150. (9 pages)

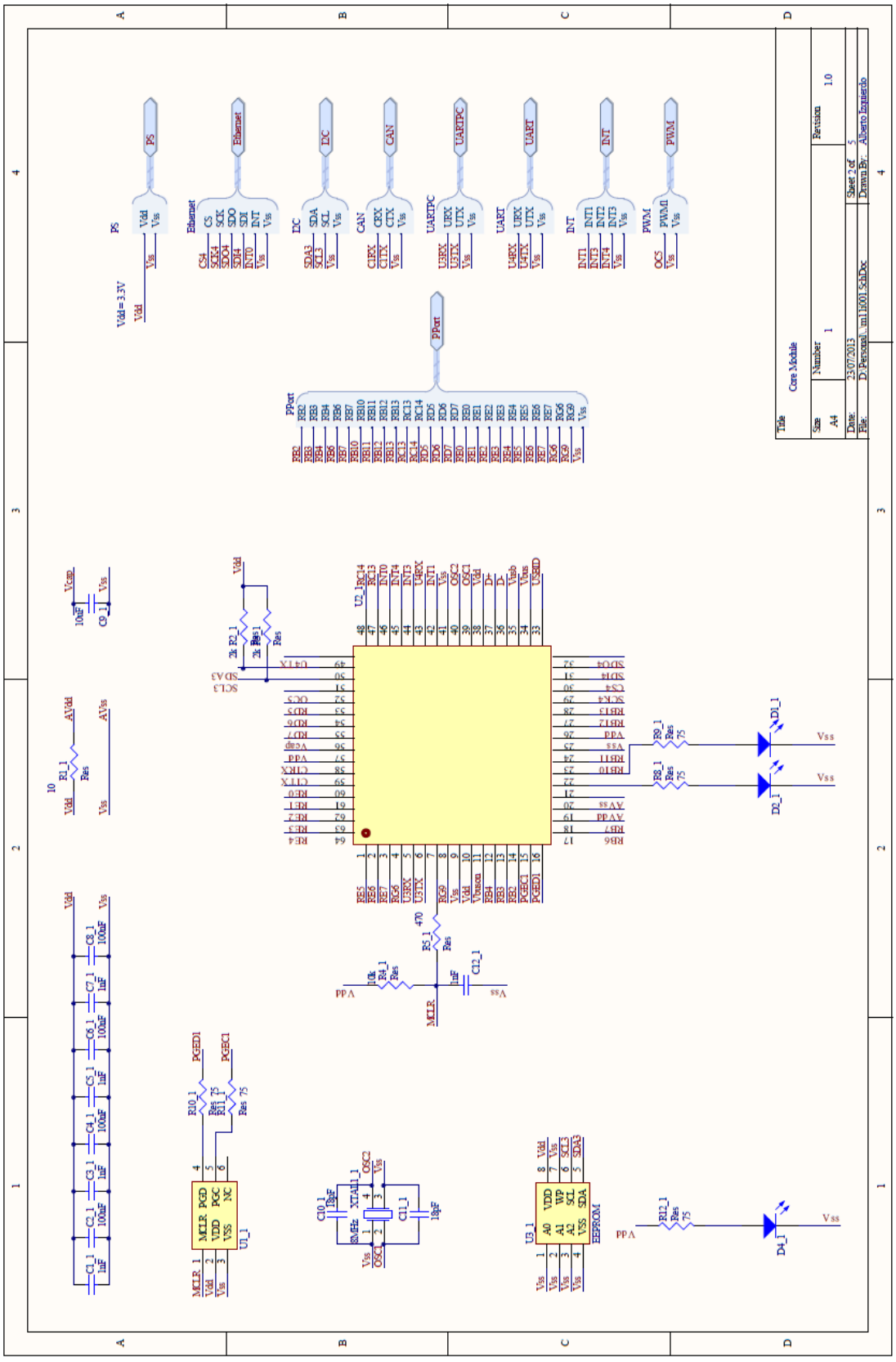
[18] Emmanuel A. Gonzalez, Student Member, IEEE, and Martin Christian G. Leonor. An Introduction to Signals and Sequences. Sept. 2006, vol 1, num 3.

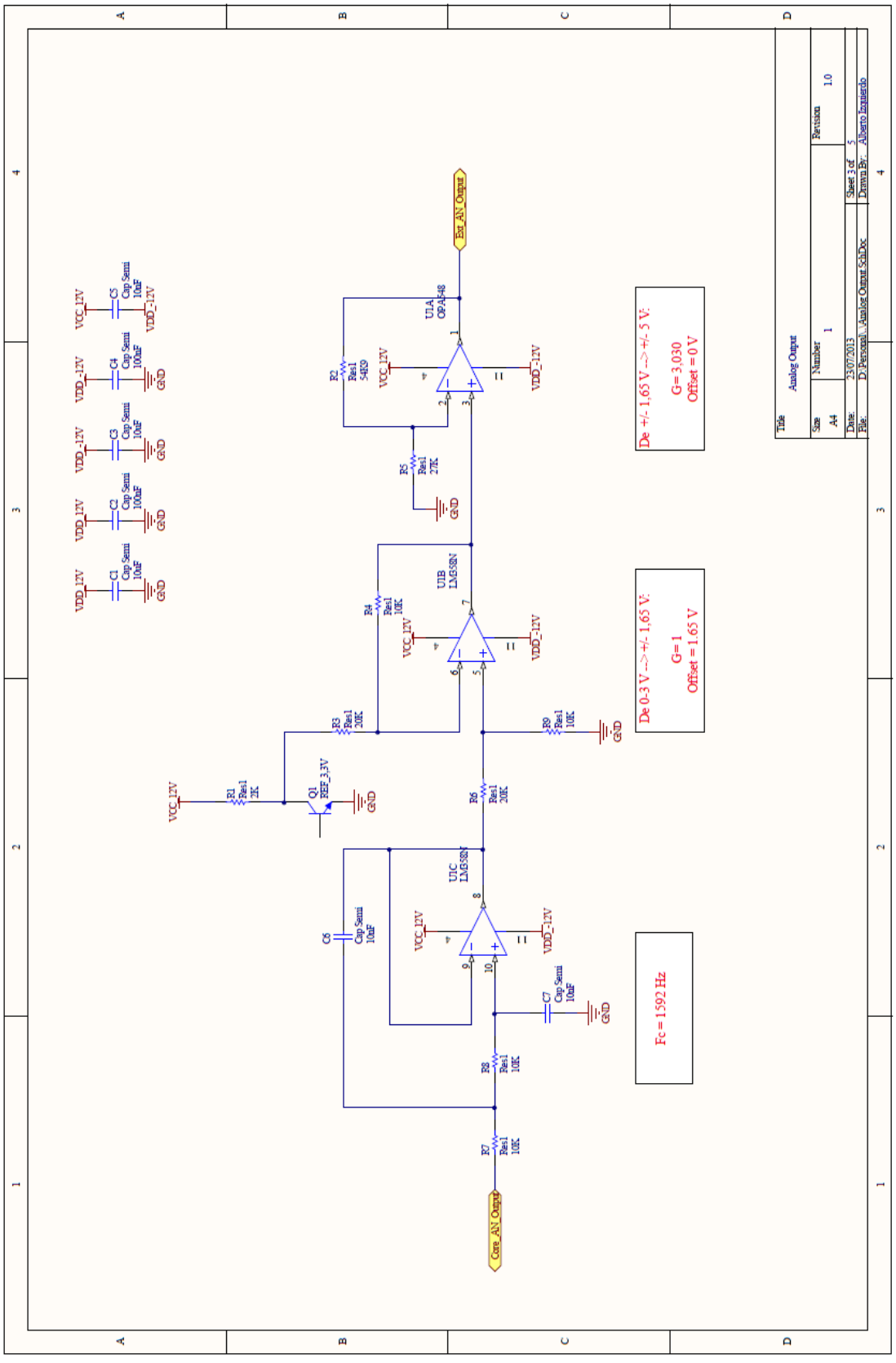
[19] Eric Vallejo R. Identificación paramétrica de sistemas dinámicos. 1997

[20] Ljung, Lennart, Torkel Glad. Modeling of dymanic systems. Ed 1, New Jersey, May 5, 1994. ISBN 0-13-597097-0

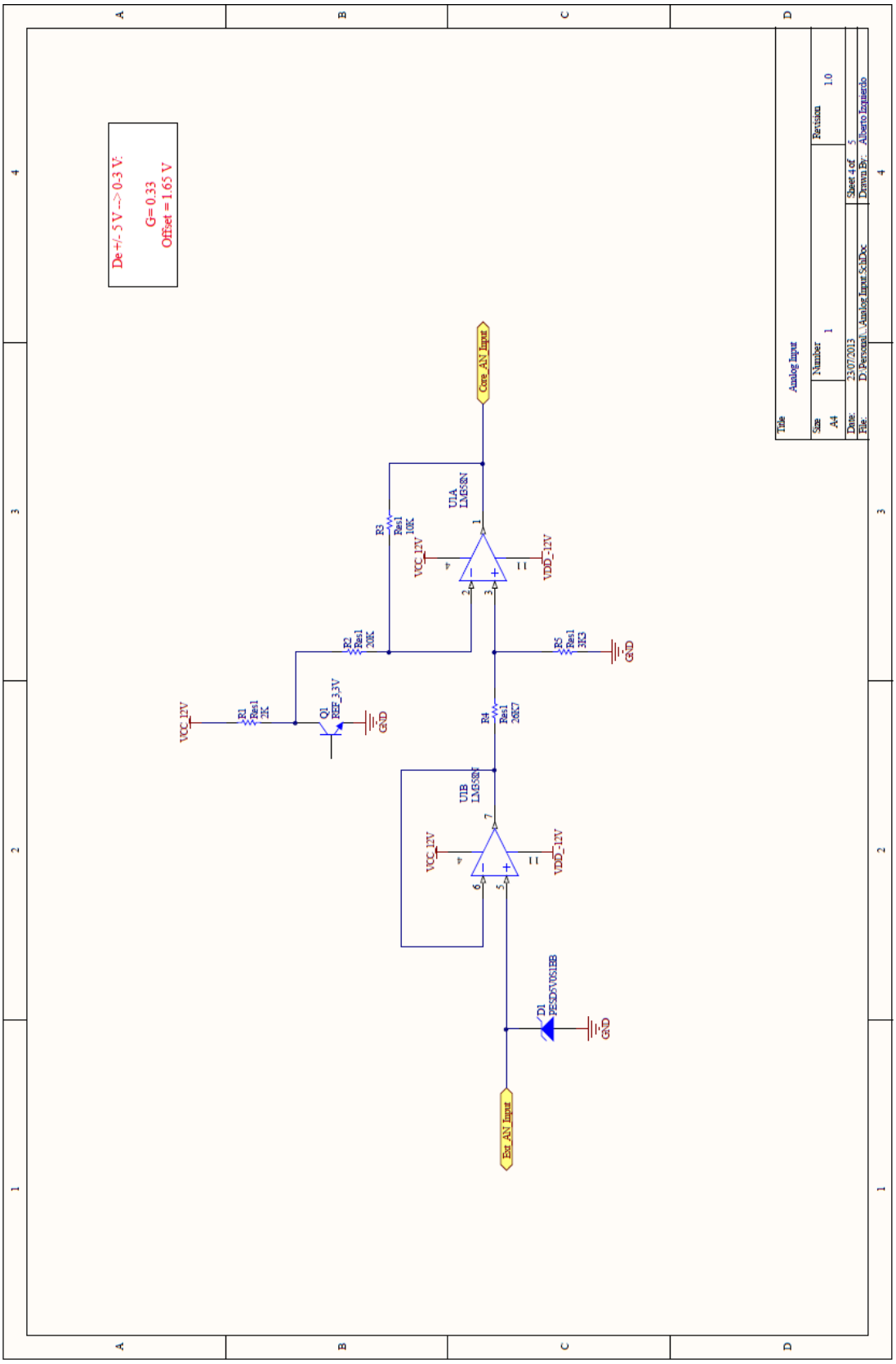
21. Planos

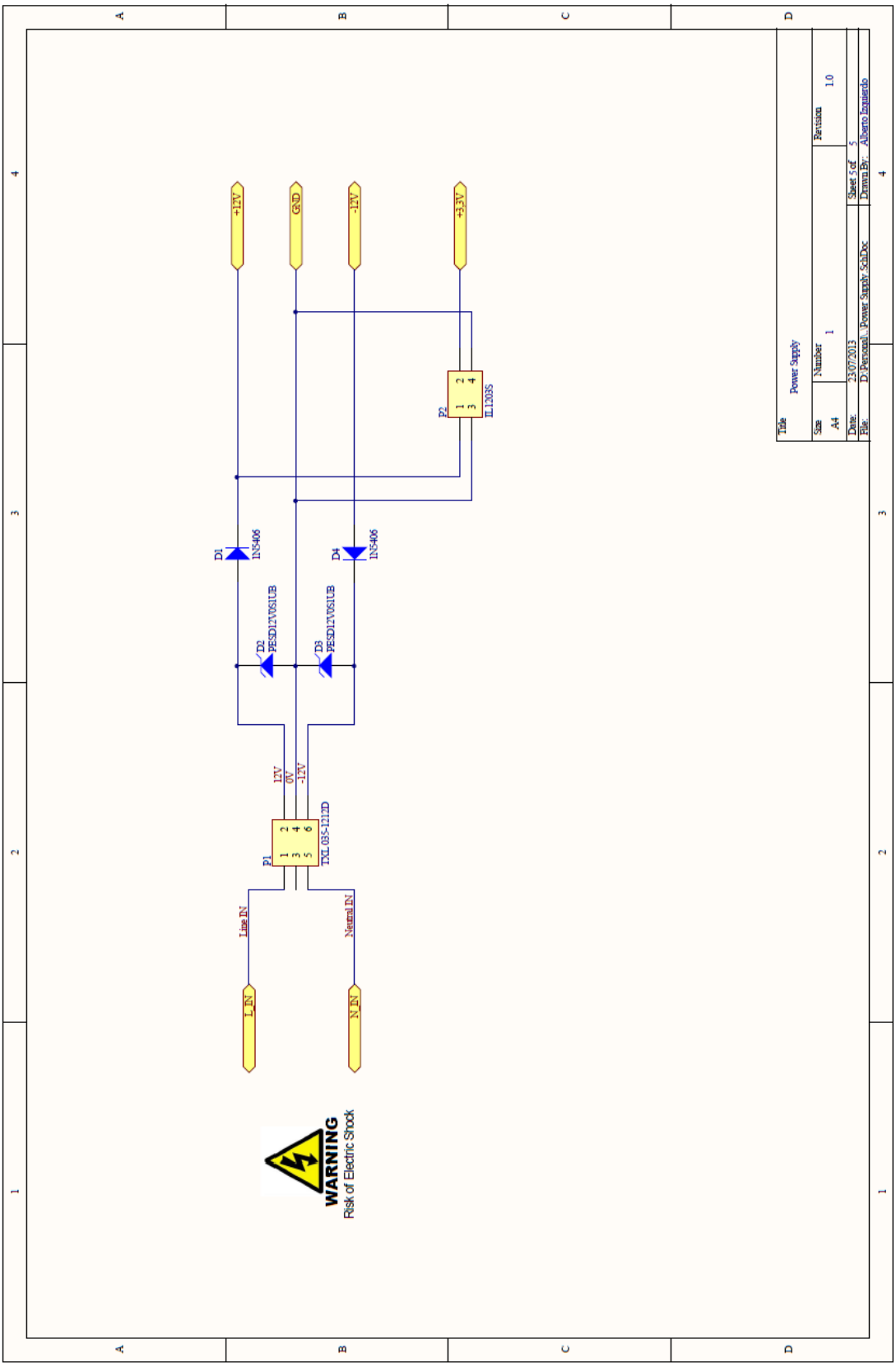

A
B
C
D
1
2
3
4
Title
USB Module
Size
A4
Number
1
Revision
1.0
Date:
23/07/2013
Sheet 1 of
5
File:
D:\Personal\..\USB.SchDoc
Drawn By:
Alberto Izquierdo
U1
Vcc
Vcc30
RTS
CTS
SLEEP
PWREN
VCC50
SHIELD
Shield
Vcc30
TXD
RXD
CLK12M
RESET
GND
USBPWR
MAC32R
TXD
RXD
GND
VCC




Ext_AN Output
U1A OPA348
R2 Res1 54K9
R5 Res1 27K
U1B LM358N
R4 Res1 10K
R3 Res1 20K
R9 Res1 10K
R1 Res1 2K
Q1 PBSF_3.3V
R6 Res1 20K
U1C LM358N
C6 Cap Semi 10uF
C7 Cap Semi 10uF
R8 Res1 10K
R7 Res1 10K
Core_AN Output
C5 Cap Semi 10uF
C4 Cap Semi 100uF
C3 Cap Semi 10uF
C2 Cap Semi 100uF
C1 Cap Semi 10uF
VCC_12V
VDD_-12V
GND
De +/- 1.65 V --> +/- 5 V:
G= 3.030
Offset = 0 V
De 0-3 V --> +/- 1.65 V:
G= 1
Offset = 1.65 V
Fc = 1592 Hz
Title
Analog Output
Size
A4
Number
1
Revision
1.0
Date:
23/07/2013
Sheet 3 of 5
File:
D:\Personal\Analog Output.SchDoc
Drawn By:
Alberto Izquierdo

+-


De+/- 5 V --> 0-3 V:
G= 0.33
Offset = 1.65 V
Core_AN Input
U1A
LM358N
R3
Res1
10K
VCC_12V
VDD_-12V
R2
Res1
20K
R5
Res1
3K3
GND
R1
Res1
2K
Q1
REF_3.3V
R4
Res1
26K7
U1B
LM358N
D1
PESD5V0S1BB
Ext_AN Input
Title
Analog Input
Size
A4
Number
1
Revision
1.0
Date:
23/07/2013
Sheet 4 of 5
File:
D:\Personal\Analog Input Sch.SchDoc
Drawn By:
Alberto Ingusardo


+12V
GND
-12V
+3.3V
P2
IL1203S
D1
1N5406
D4
1N5406
D2
PESD12VS1UB
D3
PESD12VS1UB
12V
0V
-12V
P1
TXL035-1212D
Line IN
Neutral IN
L_IN
N_IN
WARNING
Risk of Electric Shock
Title
Power Supply
Size
A4
Number
1
Revision
1.0
Date:
23/07/2013
File:
D:\Personal\...\Power Supply.SchDoc
Sheet 5 of 5
Drawn By:
Alberto Ramiendo