ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

GRADO EN INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

Título del proyecto:

SISTEMA DE MEDIDA DE VIBRACIONES EN ASCENSORES UTILIZANDO SENSORES DE TIPO MEMS

Óscar Cirauqui Díaz

Tutor: Jesús M. Corres Sanz

Pamplona, 10 de octubre de 2014

RESUMEN

En este proyecto se ha llevado a cabo el diseño e implementación de un sistema de medición y análisis de las vibraciones, basado en sensores de tipo MEMS. Este sistema será utilizado por la empresa PERMAGSA con el fin de analizar las vibraciones producidas por sus motores de imanes permanentes utilizados en ascensores. Para realizar este proyecto se ha partido del sistema actualmente en uso en dicha empresa (ADISUSBZ), a partir del cual, se han estudiado diferentes opciones para la mejora del mismo. Tras el estudio de diferentes alternativas se ha optado por el desarrollo de un nuevo sistema basado en el mismo sensor.

Para desarrollar este sistema se ha diseñado y fabricado una nueva tarjeta de circuito impreso basada en un microcontrolador PIC32 de Microchip, mediante la cual se muestrean los datos del sensor utilizando el protocolo SPI y se procesan para su posterior visualización de dos formas distintas:

1. Conversión de los datos a una señal analógica mediante un convertidor digital-analógico permitiendo su visualización con un osciloscopio.
2. Transmisión de los datos al PC mediante el protocolo RS-232 y graficarlos en una aplicación desarrollada en MATLAB.

Una vez realizado todo el diseño y verificado su funcionamiento en la Universidad con motores, se ha procedido a la implementación del sistema en la empresa, realizando ensayos con los motores fabricados por la misma.

AGRADECIMIENTOS

Quisiera agradecer a mi familia todo el apoyo que me han brindado a lo largo de la carrera y, en especial, durante el desarrollo de este proyecto. Espero poder devolverles algún día parte de lo que me han dado. Muchas gracias. No me quiero olvidar tampoco de mis compañeros, que han hecho que el paso por la universidad sea mucho más ameno. También me gustaría dar gracias al personal de la universidad, en especial, a los técnicos de laboratorio del departamento de electrónica, por facilitarme todo lo necesario para que este proyecto llegase a buen puerto.

Por último, me gustaría dar las gracias a las dos personas que han hecho posible la realización de este proyecto: en primer lugar, Jesús Corres, mi tutor, por su inestimable ayuda con los problemas que han ido apareciendo y, en segundo lugar, a Asier Leiza por haberme dado la oportunidad de realizar este proyecto.

INDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE IMÁGENES

ÍNDICE DE DIAGRAMAS

ÍNDICE DE ECUACIONES

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

1. MEMORIA

ÍNDICE DE LA MEMORIA

1.1. INTRODUCCIÓN

El presente proyecto nace ante una necesidad de la empresa Permanent Magnets, S.A., conocida en el mercado como PERMAGSA, la cual recurre a la Universidad Pública de Navarra para lograr sus objetivos.

PERMAGSA es una empresa dedicada a la fabricación de motores síncronos de imanes permanentes para la industria del ascensor con presencia internacional en el sector. Esta empresa afincada en Alsasua (Navarra) cuenta con un amplia área de I+D provista de varias torres de ensayo.

Los motores sincrónicos de imanes permanentes se caracterizan por tener una alta eficiencia, tamaño reducido, alta dinámica de par y amplio rango de velocidad. Estos motores accionados por convertidores de frecuencia se utilizan en la industria en aplicaciones que requieren variación de velocidad con par constante y alta eficiencia. Asimismo, su uso se está incrementando en proyectos en los que se requieren un par suave, bajos niveles de ruido y bajos niveles de vibraciones, como es el caso de los ascensores.

En los motores de imanes permanentes se pueden producir vibraciones indeseadas debido a diversas causas como pueden ser: las desalineaciones entre el rotor y el estator, defectos en los rodamientos, desgaste en los rodamientos debidos al uso. La detección de estas vibraciones permite detectar las causas de estos problemas, tanto en el diseño y fabricación de los motores, como durante la vida útil del mismo una vez ensamblado en el conjunto del ascensor. La detección preventiva de estos defectos supone un aumento en la fiabilidad y seguridad del ascensor.

La medida de las vibraciones se realiza mediante sensores de aceleración, acelerómetros. Tradicionalmente los acelerómetros más utilizados en la industria eran de tipo piezoeléctrico, basados en la compresión de un retículo cristalino piezoeléctrico que produce una carga eléctrica proporcional a la fuerza aplicada. Actualmente existen equipos comerciales que permiten la medida y análisis de vibraciones basadas en acelerómetros piezoeléctricos, aunque a un coste muy elevado. Este hecho hace que se esté incrementando el uso de sensores tipo MEMS.

Imagen 1- Acelerómetro piezoeléctrico

Los dispositivos MEMS(Sistemas Microelectromecánicos) son circuitos integrados con características tridimensionales e incluso piezas móviles cuyo tamaño varía desde un micrómetro hasta un milímetro. El uso de dispositivos MEMS permite obtener en un único chip de tamaño muy reducido la salida del sensor en formato digital. Estos sensores tienen prestaciones similares a los de tipo piezoeléctrico a un precio muy inferior a estos.

Los acelerómetros son dispositivos que miden la aceleración, es decir, la tasa de variación de la velocidad de un objeto. Esto se mide en m/s^2 o en fuerzas G(g). Los acelerómetros pueden detectar las fuerzas de la aceleración, ya sea estática o dinámica. Las fuerzas estáticas incluyen la gravedad, mientras que las fuerzas dinámicas pueden incluir vibraciones y movimiento.

Los acelerómetros tipo MEMS contienen placas capacitivas en su interior. Algunas de estas placas están fijas, mientras que otras están unidas a resortes minúsculos que se mueven en relación el uno al otro, variando la capacidad entre ellos. Es a partir de estas variaciones de capacidad como se determina la aceleración.

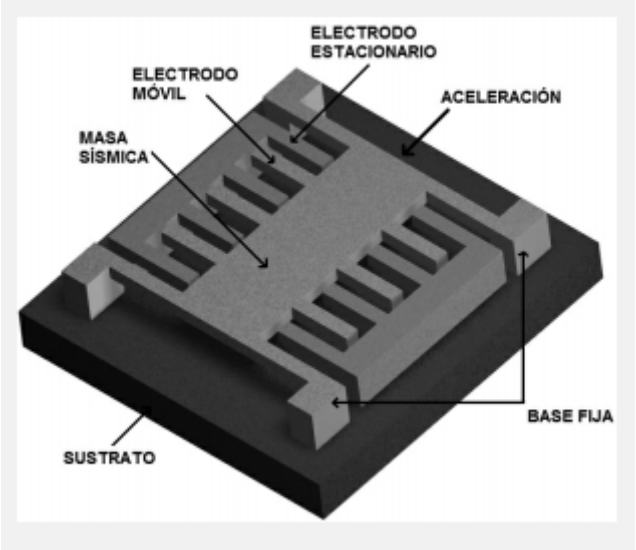


Imagen 2- Acelerómetro MEMS

Ante la manifiesta necesidad de la empresa PERMAGSA de implantar un sistema que permitiera la medición de las vibraciones producidas por sus motores, ésta se puso en contacto con la Universidad Pública de Navarra, dando como resultado el sistema detallado en el apartado *4 (*antecedentes). Sin embargo, tal como se explica en dicho apartado este sistema no es del todo satisfactorio, por lo que se pretende obtener una mejor solución a partir del mismo.

1.2. OBJETO

Este proyecto tiene por objeto la implementación de un sistema de medición y análisis de vibraciones basado en sensores de tipo MEMS. El sistema será utilizado por la empresa *PERMAGSA* para su implementación con motores de imanes permanentes utilizados en ascensores, permitiendo el análisis de las vibraciones producidas por dichos motores.

1.3. ALCANCE

Se encuentra dentro del alcance del proyecto el diseño del hardware, firmware y software del sistema, atendiendo a las especificaciones detalladas en los requisitos de diseño del **apartado 1.6**, así como la fabricación del prototipo y la puesta en marcha dentro de la empresa.

1.4. ANTECEDENTES

1.4.1. SISTEMA DE PARTIDA

El sistema actualmente en uso por la empresa *PERMAGSA*, que sirve a su vez, como punto de partida para este proyecto, está fabricado por la empresa *Analog Devices.* Este sistema se compone de un sensor de tipo MEMS y de una tarjeta de evaluación de la misma empresa. El sensor en concreto es el *ADIS 16003* y la tarjeta de evaluación es la *ADISUSBZ*. Así mismo, el fabricante proporciona un software para Windows (AD*iS16003_Eval_Rev_4)* que grafica los datos obtenidos por el sensor.

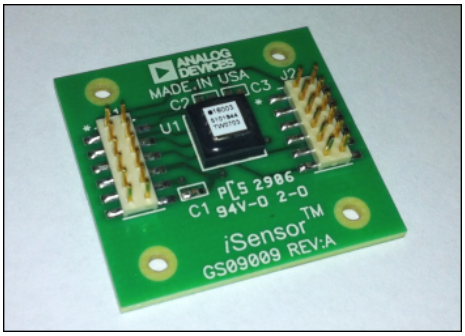


Imagen 3- Acelerómetro *ADIS16003*

Imagen 4-Tarjeta de evaluación *ADISUSBZ*

Esta tarjeta se conecta mediante USB a un PC con Windows, y mediante un software desarrollado para esta plataforma permite ver los datos en tiempo real obtenidos por el acelerómetro y graficarlos. Las opciones de configuración que permite este software son:

- Selección del eje del acelerómetro
- Nº de muestras
- Graficar los datos de forma continua
- Cálculo FFT
- Guardar datos en un fichero de texto
- Lectura de temperatura del sensor

Sin embargo, a pesar de contar con estas opciones, el software no permite ampliar el gráfico obtenido ni ajustar la escala.

El aspecto del software mencionado se puede ver en las siguientes imágenes:

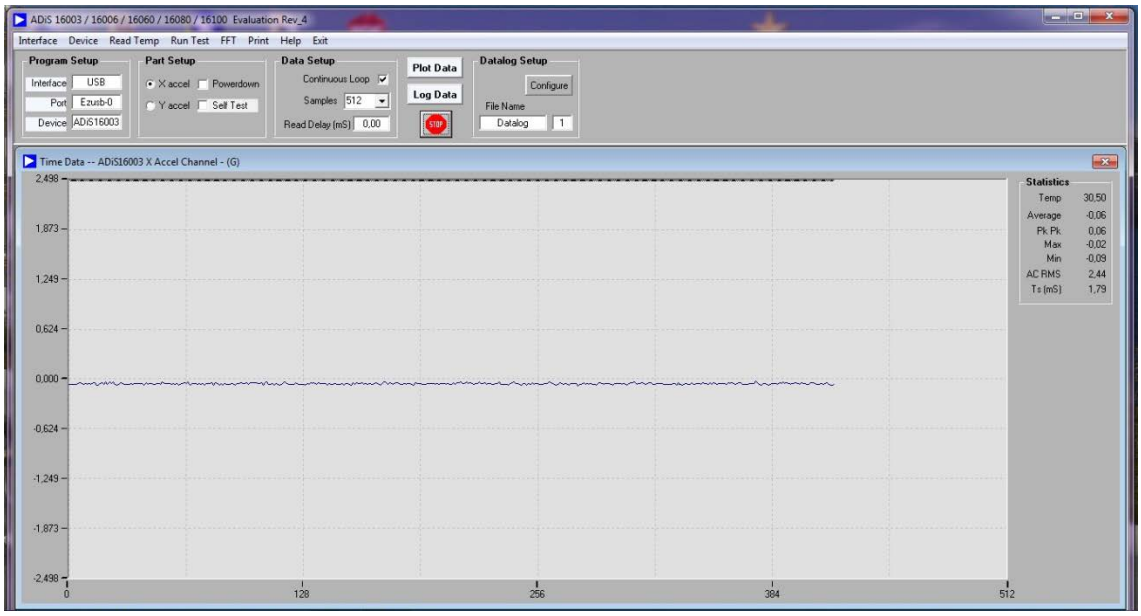


Imagen 5- Medida aceleración 0 *ADISUSBZ*

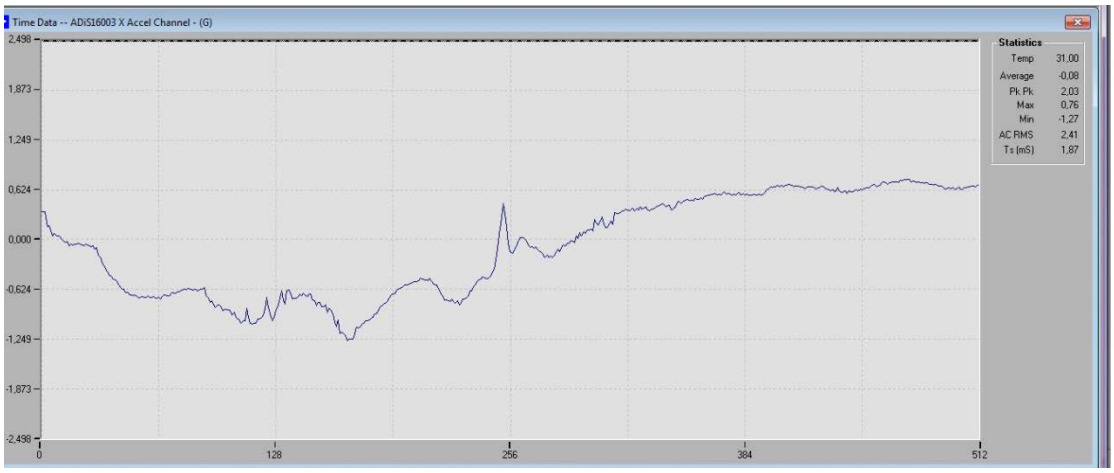


Imagen 6- Medida aceleración Eje X *ADISUSBZ*

Como se puede observar en la imagen 5, hay un pequeño ruido con el sensor en reposo, ya que la señal no es del todo continua.

De esta forma, tras analizar este sistema se concluye que tiene algunas limitaciones:

- El software proporcionado por el fabricante es algo limitado y no permite una óptima evaluación de los datos adquiridos por el sensor.
- Existen problemas de ruido en la señal obtenida.

Partiendo del sistema actual se estudiarán las posibles alternativas para conseguir el objetivo deseado.

1.4.2. ESTUDIO DE ACELERÓMETROS ALTERNATIVOS

Se ha procedido a un estudio de los distintos acelerómetros en el mercado con el fin de mejorar las prestaciones del *ADIS16003*.

Como se ha explicado en la introducción, hay principalmente dos tipos de acelerómetros: piezoeléctricos y de tipo MEMS. El primer grupo se ha descartado por su elevado coste y la dificultad para comprarlos, ya que no se encuentran en las tiendas habituales de componentes electrónicos, a pesar de que ofrecen un ancho de banda amplio.

De esta forma se ha procedido al estudio de los acelerómetros tipos MEMS. Hay varios fabricantes de este tipo de dispositivos como son *Freescale* o *Kionix*. Sin embargo, uno de los fabricantes con mayor catálogo en este tipo de acelerómetros es *Analog Devices,* que es el fabricante del sistema de partida y también el elegido para este nuevo sistema. En la siguiente tabla se pueden ver las opciones principales del catálogo de este fabricante:

Nombre	Rango de medida(g)	Ancho de banda(kHz)	Sensibilidad(mg/LSB)
ADIS16003	+/- 1.7	2.25	1.22
ADIS16227	+/- 70	9	2.384
ADIS16228	+/- 18	5	0.6104
ADIS16223	+/- 70	14	4.768

Tabla 1- Acelerómetros de Analog Devices

Para la aplicación deseada uno de los factores más importantes es la sensibilidad. Mirando la tabla, la sensibilidad del sistema actual es de 1.22, por lo que como mínimo el nuevo sistema deberá tener una sensibilidad igual a esta. Esto deja como única alternativa el *ADIS16228*. Analizando las hojas de características de cada uno de los acelerómetros se puede comparar otro de los factores más importante: el ruido; y que constituye uno de los grandes problemas del sistema actual.

ADIS 16003 → densidad de ruido de $110\mu g/\sqrt{Hz}$

ADIS 16228 → densidad de ruido de $248\mu g/\sqrt{Hz}$

Como se puede ver el *ADIS16228* tiene más ruido que el sensor actual. Por esta razón, se ha optado por usar el mismo sensor, en vez de comprar uno nuevo.

1.5. DEFINICIONES Y ABREVIATURAS

Esquemático: diagrama de conexiones entre componentes de un circuito impreso.

Debugger: Depurador. Programa utilizado para detectar y corregir errores en aplicaciones.

Firmware: software (programas, estructuras de datos) de bajo nivel, fijo y relativamente pequeño que controla la parte hardware. Está muy condicionado al hardware para el que está destinado.

Full-dúplex: los sistemas full dúplex proveen comunicación en las dos direcciones, y simultáneamente.

Half-dúplex: los sistemas half dúplex proveen comunicación en las dos direcciones, pero no simultáneamente.

g: unidad de aceleración equivalente a la aceleración terrestre media.

GND: tensión de referencia.

I^2C, I2C o IIC: *Inter Integrated Circuits*, bus serie multi maestro single-ended inventado por *Philips* y usado para comunicar periféricos de baja velocidad a un sistema embebido.

LCD: *Liquid Crystal Display*. Pantalla de cristal líquido.

LED: *Light Emitting Diode*. Diodo emisor de luz.

PC: *Personal Computer*. Ordenador personal.

PCB: *Printed Circuit Board*, placa de circuito impreso; tecnología de fabricación de dispositivos electrónicos que consiste en una placa de material aislante sobre la cual se trazan pistas de material conductor para la interconexión de componentes electrónicos.

Microcontrolador:

PIC: familia de microcontroladores basados en la arquitectura Harvard, fabricados por *Microchip Technology*.

Pad: zona de un PCB en la cual se suelda un componente.

SCI: *Serial Communications Interface.* Comunicación serie asíncrona utilizada entre microprocesadores y periféricos. Consta de dos señales TxD y RxD.

Single-ended: método de transmisión de señales eléctricas en cables: uno porta el voltaje variable que representa la señal, y el otro se conecta a la tensión de referencia.

SMD: *Surface Mount Device*, dispositivos de montaje superficial

SMC: Surface Mount Component, componente de montaje superficial.

SMT: Surface *Mountage Technology*, tecnología de montaje superficial mediante la cual los componentes electrónicos no requieren de agujeros en la placa base para su montaje.

SPI: *Serial Peripheral Interface*, estándar de comunicación serie creado por *Motorola,* que opera en full-dúplex.

Through-hole: tecnología con la que se realiza un orificio desde la capa superior a la inferior de una PCB, atravesando las intermedias.

RS-232: estándar para comunicaciones serie single-ended y con señales de control, ampliamente usado en puertos serie de ordenadores.

UART: *Universal Asynchronous Receiver/Transmitter*, transmisor/receptor asíncrono universal: transforma un flujo de datos paralelo en una comunicación serie.

USB: *Universal Serial Bus,* bus serie universal, especificación para comunicaciones entre un controlador host y dispositivos externos.

Vcc: Tensión de alimentación.

Zener: diodo preparado para trabajar en inversa con una tensión límite (tensión Zener). Normalmente utilizado como regulador de tensión.

Filtro: se refiere a filtro eléctrico o electrónico; es un elemento que discrimina una determinada frecuencia o gama de frecuencias de una señal eléctrica que pasa a través de él, pudiendo modificar tanto su amplitud como su fase.

LC: referido a la tipología de esquema electrónico formado por una bobina y un condensador.

DAC: Digital to Analog Converter, convertidor digital-analógico.

FFT: Fast Fourier Trasnform, transformada rápida de fourier.

1.6. ANÁLISIS DE SOLUCIONES

En primer lugar, se ha intentado solucionar uno de los dos problemas anteriormente mencionados: el ruido. Para ello existe una solución proporcionada en la hoja de características del sensor. Esta solución consiste en colocar dos condensadores que filtren la señal del sensor. El problema que presenta esta opción es que provoca una reducción del ancho de banda de la señal en función del valor de los mismos, tal y como muestra la siguiente tabla dada por el fabricante:

Bandwidth (Hz)	Capacitor (µF)
1	4.7
10	0.47
50	0.10
100	0.047
200	0.022
400	0.01
2250	0

Tabla 2- Relación entre la capacidad y ancho de banda del *ADIS16003*

Este inconveniente hace que esta solución no sea muy eficaz, ya que reduciría considerablemente las prestaciones del sistema, además de no solucionar el problema del software.

Se han planteado distintas opciones para alcanzar el objetivo deseado:

1. Desarrollar una nueva aplicación para Windows que obtenga los datos del sensor a través del puerto USB y los muestre de forma gráfica, utilizando la tarjeta ADISUBZ.
2. Obtener el firmware del microcontrolador que usa la tarjeta de evaluación, modificarlo y desarrollar la misma aplicación que en el apartado 1.

3. Usando el software *LabVIEW* crear un programa que permita la lectura y visualización de los datos del sensor.
4. Diseñar una nueva tarjeta de circuito impreso con un nuevo microcontrolador que permita visualizar los datos del sensor directamente en un osciloscopio y en el PC mediante una comunicación serie.

1.6.1. APLICACIÓN EN WINDOWS USANDO LA TARJETA ADISUSBZ

La tarjeta de evaluación utiliza un microcontrolador de *Cypress*, en concreto el modelo es el *CY7C68013A*. Este fabricante distribuye de forma gratuita un entorno de desarrollo de aplicaciones para *Windows* basadas en *Microsoft .NET Framework* y *Visual Studio*.

Esta suite USB incluye tanto un driver genérico (CyUSB.sys) como una librería para su uso con Visual Studio (CyUSB.dll). Para poder usar este entorno es necesario asociar el dispositivo con el driver de *Cypress*. Una vez asociado es posible crear una aplicación de *Windows Forms* que utilizando las funciones de la librería presente en la suite de desarrollo, se comunique con el dispositivo.

Tras realizar los pasos anteriores se ha procedido a desarrollar una pequeña aplicación en C#, partiendo de ejemplos proporcionados por Cypress, que permita detectar la tarjeta de *Analog* y leer los datos transmitidos por la misma. Tras ejecutar la aplicación, se ha visto la imposibilidad de recibir datos. Esto se debe a que para hacerlo es necesario conocer el protocolo de comunicación con la tarjeta.

Para intentar descifrar este protocolo se ha recurrido a un sniffer USB. Este tipo de software lo que hace es monitorizar los datos transmitidos a través de los distintos puertos USB del PC. El programa utilizado para este fin ha sido *USBlyzer*. En la siguiente captura se puede ver cómo se han obtenido estos datos:

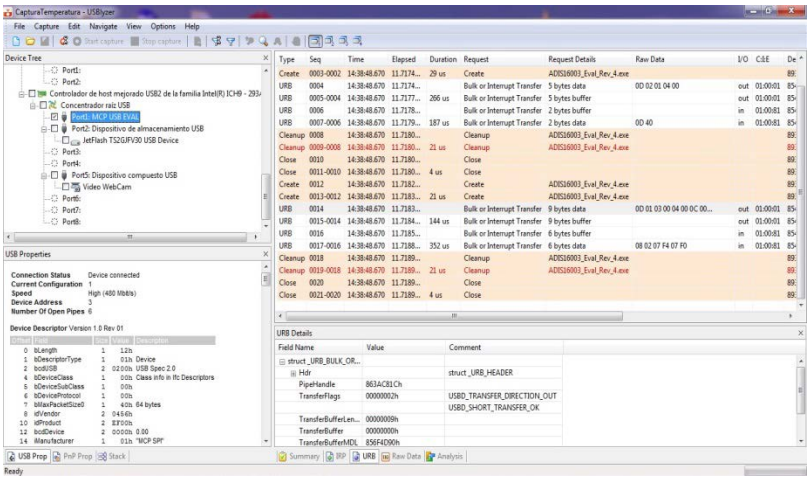


Imagen 7- Captura de datos mediante USBlyzer

Tras ver con el sniffer los datos que la aplicación *ADiS16003_Eval_Rev_4* envía a la tarjeta, se ha procedido a imitar este comportamiento con el programa creado. Sin embargo, no se ha conseguido obtener ninguna respuesta. Por lo tanto, esto indica que el protocolo de comunicación es más complicado con lo que se hace difícil su comprensión únicamente con la información obtenida mediante el sniffer. Por esta razón se descarta esta primera opción y se procederá a evaluar la opción 2.

1.6.2. APLICACIÓN A PARTIR DEL FIRMWARE DEL MICROCONTROLADOR PRESENTE EN LA TARJETA ADISUSBZ

Se ha intentado obtener el firmware de la tarjeta de evaluación *ADISUSBZ* para que partiendo de este firmware se pueda establecer la comunicación a través del puerto USB que permita leer y escribir datos en el sensor. Es necesario obtener este firmware para conocer la forma en la que el fabricante se comunica con el sensor. Sin embargo, el fabricante de la tarjeta (Analog Devices) no distribuye el firmware de la misma.

Como no se tiene acceso al firmware, la única opción restante que permitiría utilizar la tarjeta de evaluación sería la de desarrollar un nuevo firmware desde cero. Por otro lado, el desarrollar todo el firmware y luego crear la aplicación sería muy costoso, requiriendo demasiado tiempo. Por lo que no queda más remedio que descartar esta opción.

1.6.3. DESARROLLO DE UNA APLICACIÓN EN LABVIEW

LabVIEW (acrónimo de Laboratory Virtual Instrumentation Engineering Workbench) es una plataforma y entorno de desarrollo, creado por *National Instruments*, para diseñar sistemas con un lenguaje de programación visual gráfico.

Analog Devices distribuye una librería para la tarjeta *ADISUSBZ*. Esta librería, *MCP_iSensor.dll* dispone de varias funciones para leer y transmitir datos. Es posible llamar a esta librería desde *LabVIEW* y con ella realizar la lectura y la escritura de datos entre el ordenador y la tarjeta.

La siguiente imagen muestra el programa desarrollado en LabVIEW:

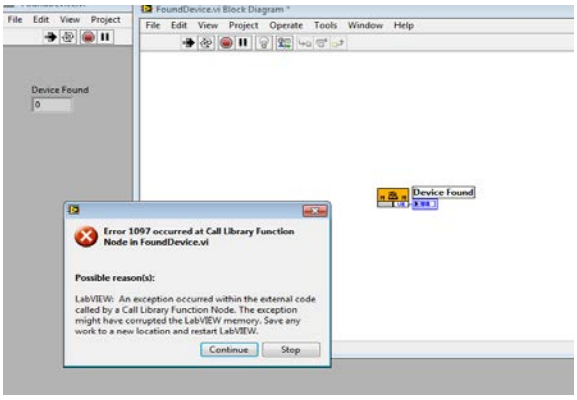


Imagen 8- Aplicación desarrollada en *LabVIEW*

Como se puede observar en la imagen al intentar llamar a la librería aparece un error. Este error parece ser debido a algún problema inherente a la misma. Se ha buscado el motivo del error pero no se ha encontrado ninguna solución. Este error se produce porque alguna función de la librería no está correctamente definida y hay alguna variable que al ser llamada por el programa, provoca dicho error en la memoria de *LabVIEW*. Este hecho conduce al descarte de esta opción, por lo que la única alternativa que resta es la opción número cuatro.

1.7. SOLUCIÓN ADOPTADA

Tras haber descartado el resto de las opciones, se procede a implementar la opción 4 que, como ya se ha dicho, consiste en el desarrollo de una nueva tarjeta, así como el desarrollo del firmware y software necesario para la aplicación.

En primer lugar, se van a establecer una serie de requisitos generales para el diseño de la tarjeta. Estos requisitos se detallan a continuación:

- La tarjeta deberá ser flexible de manera que permita implementar futuras modificaciones y mejoras.
- El tamaño de la tarjeta deberá ser lo más reducido posible.
- La alimentación de la tarjeta se realizará mediante un conector USB.
- Debe incluir indicadores luminosos que permitan reconocer de forma ágil el estado de la misma.
- Debe incluir pulsadores que permita elegir entre distintos modos de funcionamiento.
- Se debe incluir la opción de depuración en la misma placa.
- Deberá incluir la opción de comunicación con un PC mediante dos protocolos: USB y RS-232.
- Visualización mediante un osciloscopio.
- Opción de una pantalla para visualización de datos y mensajes.
- El microcontrolador será de la marca *Microchip* (familiarización con esta marca).
- Los componentes utilizados deberán cumplir los siguientes requisitos:
    - Buena relación calidad/precio sin perjuicio de la funcionalidad.
    - Bajo consumo.
    - Se evitarán los componentes *through-hole*, prefiriendo en su lugar componentes de montaje superficial o SMD que ocupan un menor espacio.

Una vez establecidos los requisitos que debe cumplir la tarjeta, se ha procedido al diseño de la misma.

1.8. DESARROLLO DE LA SOLUCIÓN ADOPTADA

El primer paso para el diseño de la tarjeta es la definición del esquema de la misma y la selección de los componentes. Para ello, es necesario analizar el acelerómetro, ya que sus características condicionan el resto de elementos de la tarjeta. Una vez que se identifique el número y el tipo de todas las conexiones se procederá a la elección de un microcontrolador que se adecue a estas necesidades.

Como se puede observar en las hojas de características del sensor seleccionado (ADIS 16003), el protocolo de comunicación con el mismo es SPI, por lo que habrá que tener esto en cuenta a la hora del diseño. A continuación se muestra un esquema general de la tarjeta:

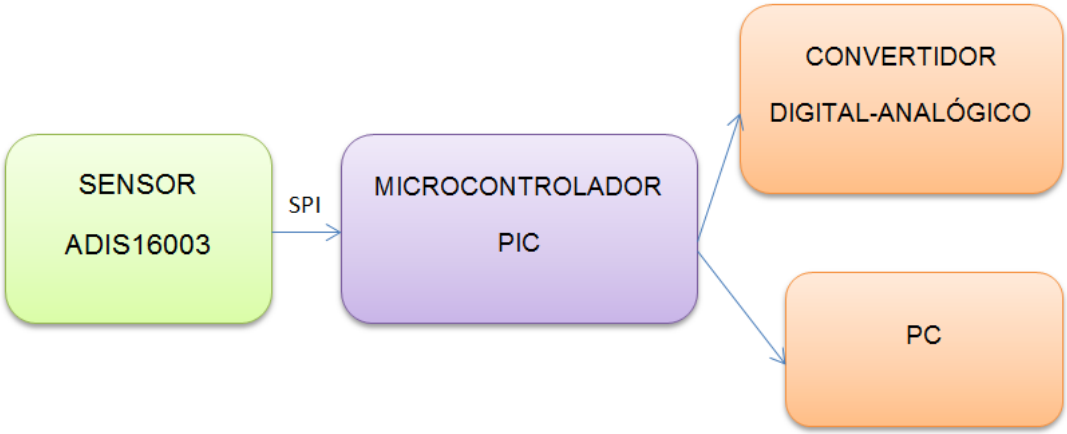


Diagrama 1- Esquema general de la tarjeta

Tanto el diseño de los esquemáticos, como el de la PCB, se han realizado con el software DesignSpark. La razón de esta elección se debe a que este es un software gratuito distribuido por la empresa RS con el que ya se está familiarizado y, además, dispone de un importante número de librerías de componentes descargables.

1.8.1. CONVERTIDOR DIGITAL-ANALÓGICO

Según la definición de Wikipedia: “Un conversor digital analógico es un dispositivo para convertir señales digitales con datos binarios en señales de corriente o de tensión analógica”. Las principales características a tener en cuenta a la hora de elegir un convertidor son las siguientes:

- Número de bits(resolución)
- Código de entrada de los datos (BCD, binario…)
- Protocolo usado en los datos de entrada (serie, paralelo…)
- Velocidad de transmisión de los datos de entrada (bit rate)
- Niveles de tensión admitidos
- Número de canales de salida

- Tipo de señal de salida (tensión o corriente)
- Polaridad (unipolar, bipolar)
- Tensión de referencia (interna o externa)

Una vez conocido esto se fijarán algunas de estas características en relación a las necesidades del proyecto:

Número de bits

Se debe seleccionar el convertidor que se necesita para tener la máxima resolución a la hora de mostrar los datos obtenidos del acelerómetro. Para ello, previamente hay que conocer el número de bits que envía el sensor. En la siguiente imagen obtenida de las hojas de características del *ADIS16003* se puede observar cómo es el formato de salida de los datos de la aceleración:

ACCELEROMETER DATA FORMAT

The accelerometer data comes out in a 12-bit, offset-binary format. See Table 6 for examples of this data format.

Table 6. Acceleration Data Format Examples

Acceleration (*g*)	Decimal	Hex	Binary
+1.7	3442	0xD72	1101 0111 0010
+2/+820	2050	0x802	1000 0000 0010
+1/+820	2049	0x801	1000 0000 0001
0	2048	0x800	1000 0000 0000
−1/+820	2047	0x7FF	0111 1111 1111
−2/+820	2046	0x7FE	0111 1111 1110
−1.7	654	0x28E	0010 1000 1110

Imagen 9- Formato datos de salida ADIS16003

Como se observa en la imagen, el número de bits que se obtienen del acelerómetro son 12. Por lo tanto, para conseguir una resolución óptima y no perder información habrá que elegir un convertidor que tenga al menos el mismo número de bits, es decir, un convertidor de 12 bits.

Código de entrada de los datos

Tal como se ve en la **imagen 9**, los datos obtenidos del acelerómetro están en código binario, por lo que se necesitará un convertidor que admita datos en este mismo formato.

Protocolo usado en los datos de entrada y velocidad de transmisión

Es preferible que el protocolo de lectura de los datos sea serie porque esto simplificará considerablemente la complejidad del circuito, reduciendo el número de pistas de la tarjeta. Principalmente, hay dos tipos de protocolos series usados en convertidores y en la mayoría de componentes electrónicos: SPI e I2C.

Las principales diferencias entre estos dos protocolos se pueden observar en la siguiente tabla:

Protocolo	Nº de conexiones	Velocidad de transmisión	Beneficios
I2C	2	10kb-3.4Mb	-Menos conexiones -Posibilidad de usar direciones de esclavos
SPI	3 o 4	1MHz-100MHz	-Palabras mayores de 8 bits -Manejo de esclavos

Tabla 3- Diferencias entre SPI y I2C

Como se ve en la **tabla 3**, el protocolo SPI tiene la ventaja de que se pueden enviar palabras de más de 8 bits y la de permitir una velocidad de transmisión superior a la del protocolo I2C. Por otro lado, tiene la desventaja de que usa más hilos de transmisión. Debido a su mayor velocidad de transmisión y a una mayor simplicidad a la hora de implementarlo, el SPI se usa para comunicaciones donde no hay una gran cantidad de dispositivos. Cuando se desea comunicar una gran cantidad de componentes es preferible el I2C por su capacidad para direccionar distintos dispositivos.

Tras haber analizado los pros y contras se concluye que es preferible usar el protocolo SPI por su mayor velocidad de transmisión y su mayor sencillez.

Niveles de tensión admitidos

En este aspecto, no se presentan grandes problemas ya que el sensor tiene un rango de tensiones de salida comprendido entre 0.4V y Vcc-0.5V, tal y como se extrae de las hojas de características del mismo.

DIGITAL OUTPUT				
Output High Voltage (V_{OH})	I_{SOURCE} = 200 µA, V_{CC} = 3.0 V to 5.25 V	V_{CC} – 0.5		V
Output Low Voltage (V_{OL})	I_{SINK} = 200 µA		0.4	V

Imagen 10- Rango tensiones de salida ADIS16003

Número de canales de salida

Se fija este número en 2 canales de salida, debido a que el acelerómetro es de dos ejes. De esta forma, se pueden mostrar en el osciloscopio los dos canales al mismo tiempo.

Tipo se señal de salida

Como lo que interesa es ver la salida del convertidor en un osciloscopio, la salida del convertidor deberá ser de tensión.

Polaridad

En este apartado las opciones disponibles son unipolar y bipolar. Para la medida de vibraciones lo que interesa ver con el osciloscopio es la forma de onda de la señal, no siendo importante el nivel de tensión. Por esta razón, es suficiente con un convertidor unipolar.

Tensión de referencia

La tensión de referencia influye en dos parámetros de un convertidor:

- Vsalida

$$Vsalida = \frac{Digital_{IN}}{2^N} \times Vref \times Ganancia$$

N: número de bits

Ecuación 1- Vsalida DAC

- Resolución

$$Resolución = \frac{Vref}{2^N}$$

Ecuación 2- Resolución DAC

Las opciones en este aspecto es usar convertidores con una tensión de referencia interna o usar una tensión de referencia externa. La segunda opción tiene el inconveniente de tener que usar un circuito externo, ya que se debe usar una tensión muy estable como referencia. Debido a esto es preferible utilizar un convertidor con referencia de tensión interna.

Normalmente se usa una tensión de referencia externa cuando se necesita que la tensión obtenida se ajuste a unos niveles concretos. Sin embargo, este no es el caso; porque como se ha explicado antes no es algo importante en este proyecto.

Conocidas las características requeridas por el DAC se procede a evaluar las distintas opciones en el mercado.

Una opción es comprar un microcontrolador que incluya un DAC de 12 bits. Sin embargo, al mirar en la gama de microcontroladores PIC no hay muchos que lo incluyan, y los que lo tienen no cumplen algún otro requisito. Es por esto por lo que se ha decidido comprar un DAC por separado.

Con las características fijadas anteriormente se ha mirado los DAC de la empresa *Microchip*, ya que se va a usar un microcontrolador del mismo fabricante. Filtrando las opciones en su página web se obtiene el siguiente resultado:

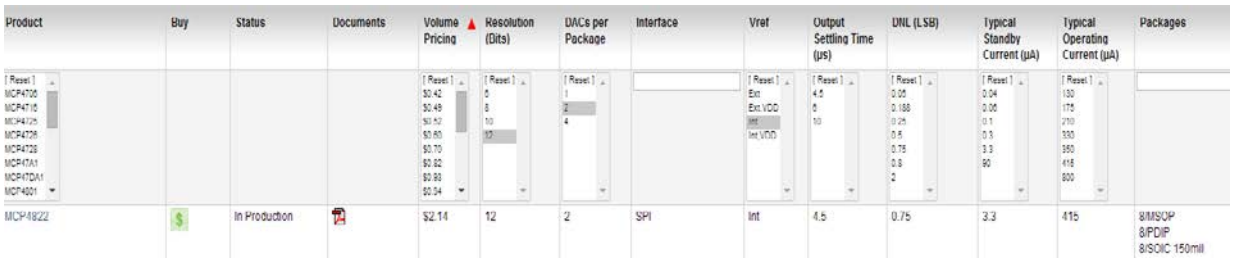


Imagen 11- Selección DAC Microchip

El modelo seleccionado es el MCP4822, que se encuentra en distintos encapsulados de montaje superficial. De sus hojas de características se pueden extraer las características del mismo.

Features

- 12-Bit Resolution
- ±0.2 LSb DNL (typ.)
- ±2 LSb INL (typ.)
- Single or Dual Channel
- Rail-to-Rail Output
- SPI™ Interface with 20 MHz Clock Support
- Simultaneous Latching of the Dual DACs with $\overline{\text{LDAC}}$ pin
- Fast Settling Time of 4.5 µs
- Selectable Unity or 2x Gain Output
- 2.048V Internal Band Gap Voltage Reference
- 50 ppm/°C V_{REF} Temperature Coefficient
- 2.7V to 5.5V Single-Supply Operation
- Extended Temperature Range: -40°C to +125°C

Imagen 12- Características DAC MCP4822

1.8.2. DISEÑO DEL ESQUEMÁTICO

Tras seleccionar el convertidor que se va a utilizar se procede al desarrollo del esquemático. El esquema general del circuito queda ahora de la siguiente forma:

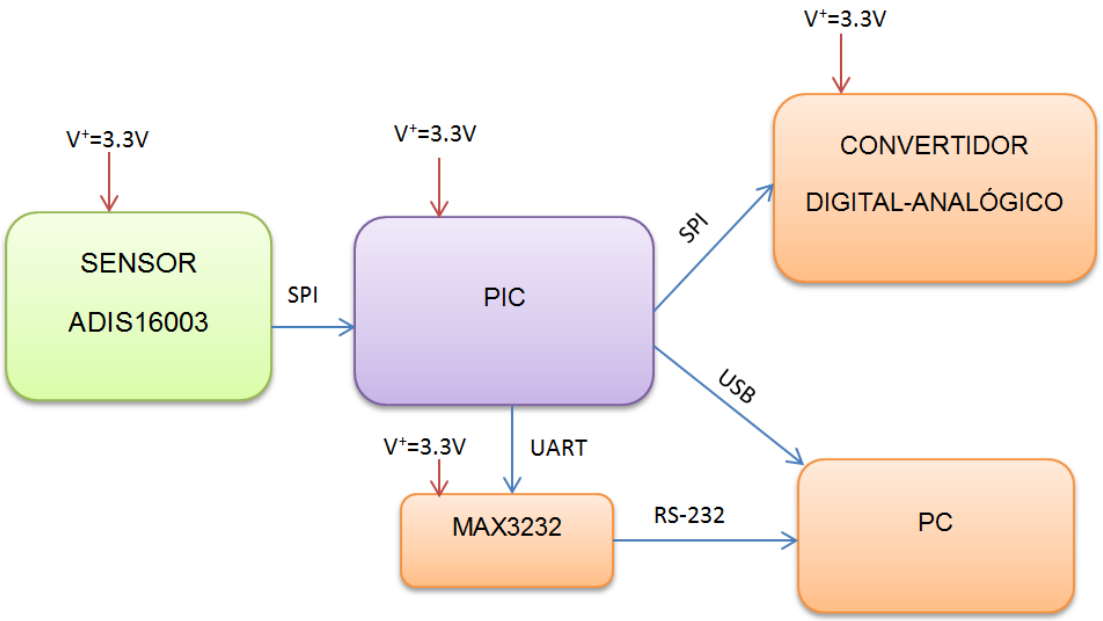


Diagrama 2- Esquema General 2

En la imagen se pueden ver los protocolos de comunicación entre los distintos componentes, así como la tensión a la que se alimentan. Estas dos características son las que van a definir las conexiones del circuito. Asimismo, se puede apreciar cómo se ha introducido un nuevo bloque que no estaba en el **Esquema General 1**.Este bloque se denomina MAX3232. Se trata de un componente cuya función es la de adaptar los niveles de tensión entre el PIC y el PC para la comunicación serie mediante el protocolo RS-232. Esto se explica más detalladamente en el apartado de **comunicación RS-232.**

Para la conexión del acelerómetro y del DAC se ha decidido utilizar el mismo bus SPI. De esta forma, se debe conectar el ChipSelect de cada uno de ellos para poder seleccionar cuál de los dos componentes se usará en cada momento. En el **Anexo I** se explica el funcionamiento de este protocolo.

Una vez definido completamente el esquema general del circuito se procede a definir por separado las conexiones de cada uno de los bloques que componen el mismo. El procedimiento para realizar esto se estructura de la siguiente forma:

1. Análisis de las hojas de características de cada uno de los componentes
2. Definición de las conexiones entre cada uno de los bloques y el PIC
3. Asignación de los pines del PIC

1.8.2.1. ALIMENTACIÓN

Como se observa en la **imagen 13,** todos los componentes se alimentan con el mismo nivel de tensión, 3.3V. Se ha optado por alimentar el circuito mediante USB, algo que se ha extendido mucho hoy en día, y que permite reducir el número de conexiones de la tarjeta en caso de usar también la comunicación USB con el PC. De esta forma, lo único necesario para alimentar la tarjeta es un sencillo circuito basado en un regulador de tensión de 3.3V. Para conseguir esto existen diferentes soluciones.

La opción más barata y sencilla es un regulador por diodo Zener, en el cual el diodo trabaja en inversa y absorbe el exceso de corriente, manteniendo un nivel constante e igual a su tensión Zener entre cátodo y ánodo. Además del zener, esta opción incluye una resistencia en serie que limita la corriente para evitar un mal funcionamiento.

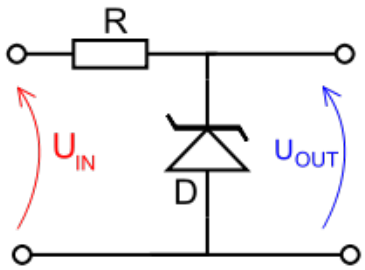


Imagen 13- Regulador Zener básico

Sin embargo, esta no suele ser una muy buena opción ya que el consumo del propio regulador es muy alto, disipando calor y bajando enormemente su rendimiento. Esto hace que en la mayoría de los diseños electrónicos se utilicen reguladores de tensión integrados, que proporcionan tensiones y corrientes de muy amplio rango con un consumo mínimo. Esta es la opción escogida para el diseño, como se puede ver en el **plano nº 1**.

Además de conseguir la tensión deseada, es importante que esta sea lo más “limpia” posible, es decir, que no tenga perturbaciones. Para lograr esto lo que se ha hecho es implementar dos filtros que eliminen las componentes de alta frecuencia para que la tensión sea lo más continua posible.

1.8.2.1.1. REGULADOR DE TENSIÓN

El regulador escogido es el ADP3338. Este componente, del fabricante *Analog Devices,* tiene las características de proporcionar una tensión de 3.3V y hasta 1 A de corriente en un encapsulado muy pequeño, además de admitir hasta 8V de tensión de entrada. Su encapsulado es un SOT-223 (encapsulado de montaje superficial de pequeñas dimensiones) con lo que se adapta muy bien a lo requerido.

FEATURES

High accuracy over line and load: ±0.8% @ 25°C,
±1.4% over temperature
Ultralow dropout voltage: 190 mV (typ) @ 1 A
Requires only C_O = 1.0 µF for stability
anyCAP is stable with any type of capacitor (including MLCC)
Current and thermal limiting
Low noise
2.7 V to 8 V supply range
–40°C to +85°C ambient temperature range
SOT-223 package

Imagen 14- Características ADP3338

Los reguladores de este tipo tienen tres patillas: una de entrada, otra de salida y la de masa. Además, es necesario añadir unos condensadores entre los pines de entrada, salida y masa. En la siguiente imagen se ve la descripción de los pines:

Table 3. Pin Function Descriptions

Pin No.	Mnemonic	Description
1	GND	Ground Pin.
2	OUT	Regulator Output. Bypass to ground with a 1 µF or larger capacitor.
3	IN	Regulator Input. Bypass to ground with a 1 µF or larger capacitor.

Imagen 15- Conexiones ADP3338

1.8.2.1.2. FILTRO DE MODO COMÚN

Es un filtro EMI (Filtro de Interferencias Electromagnéticas) para las señales que vienen de la alimentación. Este filtro consiste en dos bobinas en serie en la línea de alimentación: una de ellas se conecta a la línea de masa y otra en la de Vcc. Existen filtros comerciales de este tipo. Se ha escogido un filtro del fabricante *Bourns*. En las hojas de características del mismo se puede ver cómo se comporta este filtro:

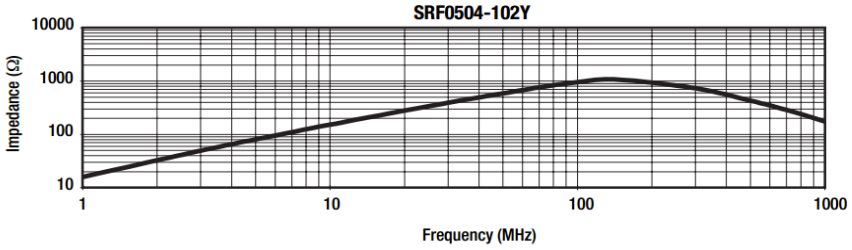


Imagen 16- Comportamiento Filtro Modo Común

El filtro tiene una impedancia muy pequeña a bajas frecuencias. Mientras que a frecuencias altas (a partir de 1 MHz), tal y cómo se ve en la imagen, su impedancia aumenta considerablemente, con lo que no deja pasar las componentes de alta frecuencia de la señal de entrada.

1.8.2.1.3. FILTRO PASO BAJO

Este filtro permite filtrar las componentes de alta frecuencia a la salida del regulador de tensión. La tipología elegida para este filtro es LC, ya que de esta forma se consigue una disipación de potencia muy baja. La finalidad de este filtro es conseguir que la tensión de alimentación del DAC sea lo más "limpia" posible. Su frecuencia de corte viene dada por:

$$f = \frac{1}{2\pi\sqrt{LC}}$$

Ecuación 3- Filtro Paso Bajo LC

Fijando: C=6800uF y L=50mH se obtiene una frecuencia de corte de 8.63Hz.

Como se puede ver en el esquema del **plano nº 1** se han definido dos conexiones distintas de masa y de Vcc.

GND ⟶ señal de masa entrada conector USB

GNDA ⟶ señal de masa a la salida del filtro de modo común

+3.3V ⟶ Vcc para todos los componentes

VccA ⟶ Vcc para el DAC

1.8.2.1.4. CONECTOR USB

El conector USB elegido es el mini USB-A. Este conector ofrece las mismas características que el conector USB común pero con un tamaño más reducido. En la siguiente imagen se puede ver la disposición de los pines del mismo:

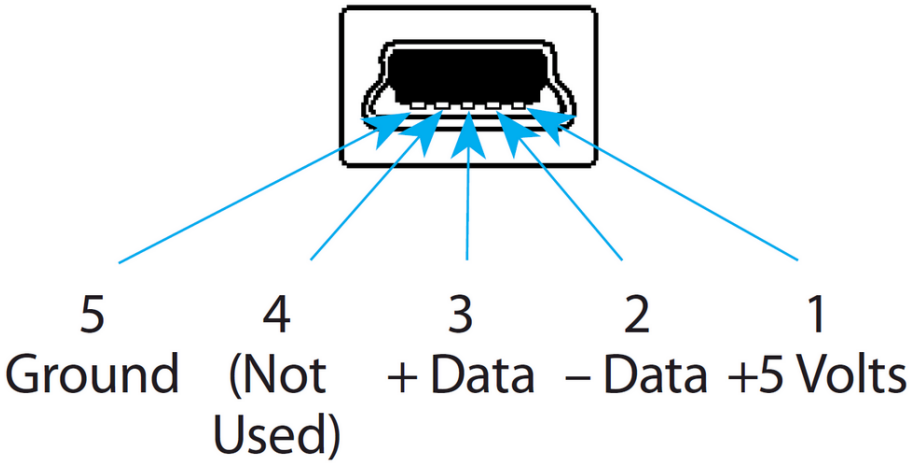


Imagen 17- Conector USB mini A

De esta forma, se definen las siguientes conexiones para el conector de entrada USB:

Pin 1 ⟶ Vcc 5 voltios ⟶ +5V

Pin 2 ⟶ Datos ⟶ D-

Pin 3 ⟶ Datos ⟶ D+

Pin 5 ⟶ Masa ⟶ GND

1.8.2.1.5. INDICADOR DE ALIMENTACIÓN

Se ha optado por la inclusión de un diodo LED que indique si la tarjeta está o no alimentada, algo que es muy habitual en la mayoría de las tarjetas. Para ello, basta con conectar un diodo LED a la salida del regulador e incluir una resistencia en serie que limite la corriente que circula por el mismo. Su valor se debe calcular de forma que la corriente máxima que circule por el diodo no supere la máxima corriente directa estipulada por el fabricante del mismo. De esta forma, la resistencia Rs viene dada por:

$$Rs = \frac{Vdd - Vf}{If}$$

Ecuación 4- Resistencia serie LED

Vdd: tensión de alimentación

Vf: tensión directa(dada por el fabricante)

If: corriente directa(dada por el fabricante)

Este componente no requiere ninguna característica especial, siendo la mayoría de los componentes válidos para el presente diseño, así que el componente elegido es del fabricante OSRAM.

1.8.2.2. ADIS16003

A la hora de conectar el acelerómetro al motor se han definido dos posibilidades:

- Conectar el sensor en la tarjeta junto con el resto de componentes.
- Conectar el sensor de forma individual en una pequeña tarjeta que irá conectada a la tarjeta principal mediante un cable.

En el sistema de evaluación de partida, toda la tarjeta iba colocada encima del motor. Sin embargo, se ha decidido que probablemente sea más conveniente colocar una masa lo más pequeña posible para medir las vibraciones del motor, es decir, colocar el sensor encima del motor y el resto que vaya en una tarjeta aparte.

Para conectar el acelerómetro de forma externa, se ha incluido en el diseño de la tarjeta el mismo conector usado en la tarjeta *ADISUBZ*. Este conector aparece en el **plano nº 3** como *CONECTOR ADIS16003*.

Para realizar las conexiones del acelerómetro se recurre a sus hojas de características:

Table 5. Pin Function Descriptions

Pin No.	Mnemonic	Description
1	$\overline{TCS}$	Temperature Chip Select. Active low logic input. This input frames the serial data transfer for the temperature sensor output.
2	DOUT	Data Out, Logic Output. The conversion of the ADIS16003 is provided on this output as a serial data stream. The bits are clocked out on the falling edge of the SCLK input.
3	DIN	Data In, Logic Input. Data to be written into the control register of the ADIS16003 is provided on this input and is clocked into the register on the rising edge of SCLK.
4	COM	Common. Reference point for all circuitry on the ADIS16003.
5, 7	NC	No Connect.
6	ST	Self-Test Input. Active high logic input. Simulates a nominal 0.75 *g* test input for diagnostic purposes.
8	YFILT	Y-Channel Filter Node. Used in conjunction with an optional external capacitor to band limit the ac signal from the accelerometer.
9	XFILT	X-Channel Filter Node. Used in conjunction with an optional external capacitor to band limit the ac signal from the accelerometer.
10	$\overline{CS}$	Chip Select. Active low logic input. This input provides the dual function of initiating the accelerometer conversions on the ADIS16003 and frames the serial data transfer for the accelerometer output.
11	V_{CC}	Power Supply Input. The V_{CC} range for the ADIS16003 is from 3.0 V to 5.25 V.
12	SCLK	Serial Clock, Logic Input. SCLK provides the serial clock for accessing data from the part and writing serial data to the control register. This clock input is also used as the clock source for the conversion process of the ADIS16003.

Imagen 18- Pines ADIS16003

A partir de la información de la tabla anterior se definen las siguientes conexiones, tal y como se puede observar en el **plano nº 3**:

Pin 1 ⟶ ChipSelect temperatura ⟶ TCS

Pin 2 ⟶ Salida Datos ADIS16003 ⟶ SDI2

Pin 3 ⟶ Entrada Datos ADIS16003 ⟶ SDO2

Pin 4 ⟶ Masa ⟶ GNDA

Pin 6 ⟶ Self Test ⟶ ST

Pin 8 ⟶ Filtro Canal Y

Pin 9 ⟶ Filtro Canal X

Pin 10 ⟶ ChipSelect aceleración ⟶ CSA

Pin 11 ⟶ Vcc ⟶ 3.3V

Pin 12 ⟶ Serial Clock ⟶ SCK2

Cabe mencionar, que en el pin 6 se ha añadido un interruptor con dos resistencias que limitan la corriente, con el propósito de que esto permita diagnosticar el correcto funcionamiento del acelerómetro. La señal del interruptor está conectada al pin 6, *ST*.

Además de esto, se han añadido dos condensadores de valor indefinido en los pines 8 y 9 que permitan en un momento dado limitar el ancho de banda del sensor.

1.8.2.3. CONVERTIDOR DIGITAL-ANALÓGICO

En las hojas de características del MCP4822 se puede ver la funcionalidad de sus pines:

TABLE 3-1: PIN FUNCTION TABLE

MCP4821 Pin No.	MCP4822 Pin No.	Symbol	Function
1	1	V_{DD}	Positive Power Supply Input (2.7V to 5.5V)
2	2	$\overline{CS}$	Chip Select Input
3	3	SCK	Serial Clock Input
4	4	SDI	Serial Data Input
5	5	$\overline{LDAC}$	Synchronization input used to transfer DAC settings from serial latches to output latches
6	—	$\overline{SHDN}$	Hardware Shutdown Input
—	6	V_{OUTB}	DAC_B Output
7	7	AV_{SS}	Analog Ground
8	8	V_{OUTA}	DAC_A Output

Imagen 19- Pines MCP4822

Tras analizar esto, se definen las siguientes conexiones (véase plano nº3):

Pin 1 ⟶ Vdd ⟶ VccA

Pin 2 ⟶ ChipSelect ⟶ CSDAC

Pin 3 ⟶ Serial Clock Input ⟶ SCK2

Pin 4 ⟶ Serial Data Input ⟶ SDO2

Pin 5 ⟶ Sincronización para actualizar salida DAC ⟶ LDAC

Pin 6 ⟶ Salida Canal B

Pin 7 ⟶ Masa analógica ⟶ GNDA

Pin 8 ⟶ Salida Canal A

En los pines 6 y 8 se han colocado unos condensadores opcionales para dar la posibilidad de estabilizar la tensión a la salida del DAC en caso de ser necesario.

1.8.2.4. COMUNICACIÓN RS-232

Este bloque del esquema se basa en la adaptación de los niveles de tensión del PIC (de 0 a 3.3V) a los del PC. Para ello se deben conocer los niveles de tensión que se usan en la interfaz RS-232. Estos niveles de tensión se muestran en la siguiente tabla:

Tensión	Señal	Nivel Lógico	Control
+3 a +15	Espacio	0	On
-3 a –15	Marca	1	Off

Imagen 20- Niveles Tensión RS-232

Como se ha dicho antes, para conseguir adaptar estos niveles existen componentes comerciales que realizan dicha función. El componente empleado es el *MAX3232CD* de *Texas Instruments.*

En las propias hojas de características del componente, se puede encontrar la forma en la que se debe conectar el mismo, así como el valor de los condensadores a usar en función de la tensión de alimentación:

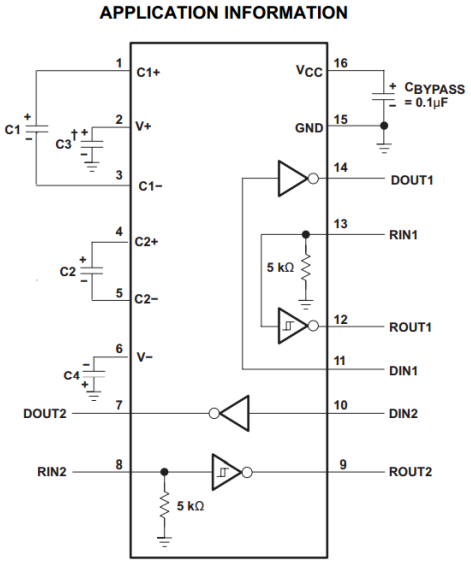


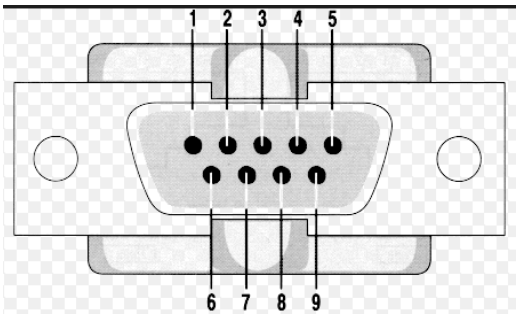
Imagen 21- Aplicación MAX3232CD

V_{CC} vs CAPACITOR VALUES

V_{CC}	C1	C2, C3, C4
3.3 V ± 0.3 V	0.1 µF	0.1 µF
5 V ± 0.5 V	0.047 µF	0.33 µF
3 V to 5.5 V	0.1 µF	0.47 µF

Imagen 22- Valor de C para MAX3232CD

De esta forma, se realizan las conexiones del mismo siguiendo este esquema. El conector usado es el DB-9, que es el conector habitual en PC para conexiones serie.



Número de clavija	Nombre
1	CD: Detector de transmisión
2	RXD: Recibir datos
3	TXD: Transmitir datos
4	DTR: Terminal de datos lista
5	GND: Señal de tierra
6	DSR: Ajuste de datos listo
7	RTS: Permiso para transmitir
8	CTS: Listo para enviar
9	RI: Indicador de llamada
	Protección

Imagen 23- Conexiones DB-9

En este proyecto sólo se usan los pines 2 y 3 que son para enviar y recibir, así como el pin 5, que es el de masa. El resto de pines sirven simplemente para control de flujo, pero no son necesarios. En el **plano nº 4** se puede ver el esquema completo de este bloque.

Las conexiones de este bloque con el PIC quedan definidas de la siguiente forma:

Pin 11 ⟶ Entrada Datos Serie ⟶ TX

Pin 12 ⟶ Salida Datos Serie ⟶ RX

1.8.2.5. PANTALLA LCD

Como ya se ha comentado con anterioridad, la conexión de la pantalla se realiza mediante I2C. De esta forma, mirando las hojas de características se tiene el siguiente esquema de conexiones:

Pin No.	Symbol	External Connection	Function Description
1	RST	MPU	Active LOW Reset
2	SCL	MPU	Serial Clock (requires pull-up resistor)
3	SDA	MPU	Serial Data (requires pull-up resistor)
4	VSS	Power Supply	GND
5	VDD	Power Supply	+3.3V
6	VOUT	CAP	Voltage booster circuit – connect to 1uF cap to VSS or VDD
7	C1+	CAP	Connect to 1uF cap to PIN8
8	C1-	CAP	Connect to 1uF cap to PIN7

Imagen 24- Pines pantalla LCD

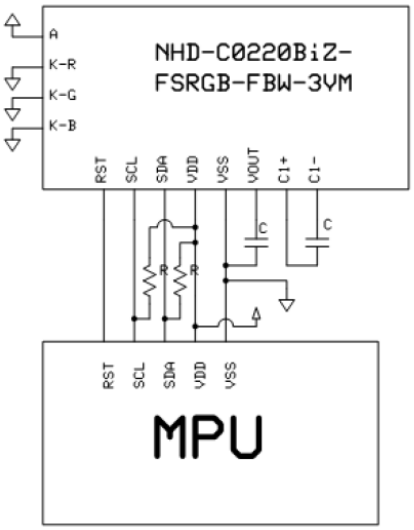


Imagen 25- Esquema conexiones pantalla LCD

Este es el esquema que se ha implementado en el diseño; como se observa en el **plano nº 5**.

Las conexiones entre el PIC y este bloque son:

Pin 2 ⟶ Señal de reloj ⟶ SCL

Pin 3 ⟶ Datos serie ⟶ SDA

1.8.3. MICROCONTROLADOR PIC

Un microcontrolador es “un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria. Está compuesto por varios bloques funcionales, los cuales cumplen una tarea específica. Un microcontrolador incluye en su interior las tres principales unidades funcionales de una computadora: unidad central de procesamiento, memoria y periféricos de entrada/salida”. (Definición de Wikipedia).

Microchip tiene tres familias principales de microcontroladores: 8, 16 y 32 bits. La diferencia entre una familia y otra reside en el tamaño de palabra que el microcontrolador puede manejar a la hora de realizar operaciones. Un micro de 8 bits sólo puede realizar operaciones con palabras de 8 bits, mientras que uno de 16 puede operar con palabras de 16 bits y uno de 32 bits con palabras de 32 bits. En cuanto a precio, no existen diferencias significativas de precio entre las diferentes familias.

Debido a las mayores opciones de módulos para trabajar con periféricos y a la gran potencia de cálculo se ha escogido un microcontrolador de la familia PIC32. El hecho de usar un microcontrolador de esta gama, aunque en principio no es necesaria tanta capacidad de cálculo para la aplicación que se desea implementar, se debe también a que da muchas opciones a la tarjeta de cara a futuras modificaciones y mejoras de la misma. Esto ocurre porque, además de la potencia de cálculo, los PIC32 también ofrecen una amplia capacidad de configuración interna en lo que se refiere a módulos de comunicaciones, así como muchas opciones a la hora de reconfigurar los pines.

Como ya se han definido las conexiones con el microcontrolador en el diseño del esquemático, ya han quedado determinados los requisitos que debe tener. Un resumen de las conexiones definidas se muestra en la siguiente tabla:

Bloque	Nombre conexión	Protocolo	Tipo conexión
Conexión USB/RS-232	D-/RX	USB/UART	Salida digital
	D+/TX	USB/UART	Entrada digital
Acelerómetro ADIS 16003	TCS		Salida digital
	CSA		Salida digital
	SCK2	SPI	Salida digital
	SDI2	SPI	Entrada digital
	SDO2	SPI	Salida digital
CONVERTIDOR DIGITAL-ANALÓGICO	CSDAC		Salida digital
	LDAC		Salida digital
	SCK2	SPI	Salida digital
	SDO2	SPI	Salida digital
PANTALLA LCD	SCL	I2C	Salida digital
	SDA	I2C	Salida/entrada digital

Tabla 4- Conexiones Esquema

Como se puede ver en la tabla, se necesitan 10 salidas digitales, 2 entradas digitales y una que funciona tanto de entrada como de salida, ya que se usa en el protocolo I2C.

1.8.3.1. ELECCIÓN DEL MICROCONTROLADOR

El microcontrolador seleccionado deberá cumplir las siguientes especificaciones:

- 1 o 2 módulos SPI
- 1 módulo UART (implementación de protocolo RS232)
- 1 módulo USB
- 13 salidas/entradas digitales
- Montaje superficial

Tras realizar una búsqueda con los criterios especificados anteriormente en la página de Microchip, el microcontrolador escogido es el PIC32MX220F032B. Sus características se muestran a continuación:

Parameter Name	Value
Family	PIC32MX2xx
Max Speed MHz	50
Program Memory Size (KB)	32
RAM (KB)	8
Auxiliary Flash (KB)	3
Temperature Range (C)	-40 to 105
Operating Voltage Range (V)	2.3 to 3.6
DMA Channels	4
SPI™	2
I²C™ Compatible	2
CODEC Interface (I2S,AC97)	Yes
Peripheral Pin Select (PPS) Crossbar	Yes
CTMU	Yes
USB	Full Speed
USB (Channels, Speed, Compliance	1,Full Speed,USB 2.0
A/D channels	9
Max A/D Resolution	10
Max A/D Sample Rate (KSPS)	1100
Input Capture	5
Output Compare/Std. PWM	5
16-bit Digital Timers	5
Parallel Port	PMP
Comparators	3
Internal Oscillator	8 MHz, 32 kHz
I/O Pins	19
Pin Count	28

Imagen 26- Características PIC32MX220F032B

El siguiente paso es asignar los pines del mismo. Para realizar esto se ha de recurrir a las hojas de características. En ellas se encuentra la siguiente imagen donde se muestra la relación de todos sus pines:

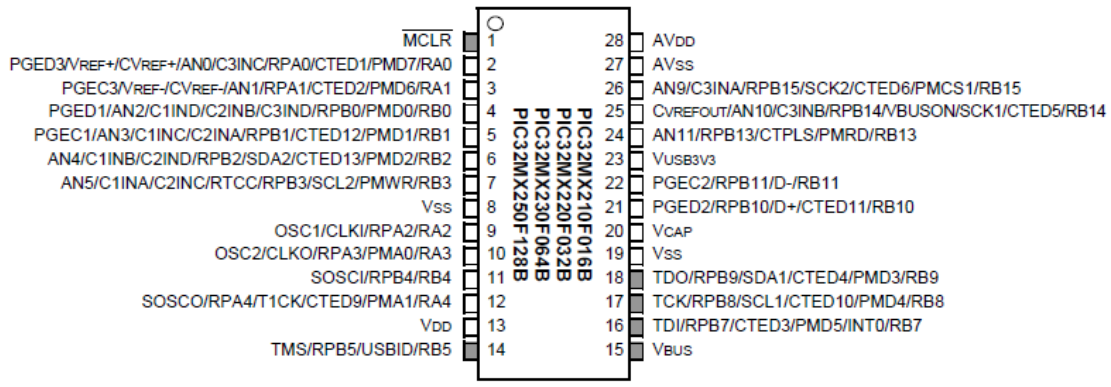


Imagen 27- Relación Pines PIC32MX220F032B

Como se puede ver en la imagen algunos de los pines tienen una función fija, mientras que otros se pueden configurar para funcionar con distintos módulos del microcontrolador. Por lo tanto, habrá que identificar primero los pines con una función fija para después configurar el resto de forma que se adapten a lo que se requiere para el diseño y que se refleja en la **tabla 4**.

1.8.3.2. PROGRAMACIÓN-DEPURACIÓN

Lo primero es definir los pines necesarios para la programación del PIC. El PIC usado tiene tres formas de programación distintas:

- Run-Time Serial-Programming(RTSP)
- EJTAG Programming
- In-Circuit Serial Programming(ICSP)

Lo que interesa es poder programar el PIC mediante el programador disponible en la Universidad, de forma que no haya que comprar otro programador. Este programador es el MPLAB ICD3, ya que el ICD2 no sirve para este PIC. Por lo tanto, hay que recurrir a las especificaciones del ICD3 para ver cómo se programa.

Mirando en sus características se observa que el método que usa el ICD3 es el ICSP. Este es el método que se usa principalmente en los microcontroladores de Microchip y permite la programación y depuración de los mismos sin necesidad de removerlos de su circuito de aplicación. Esto es precisamente lo que se requiere en este diseño.

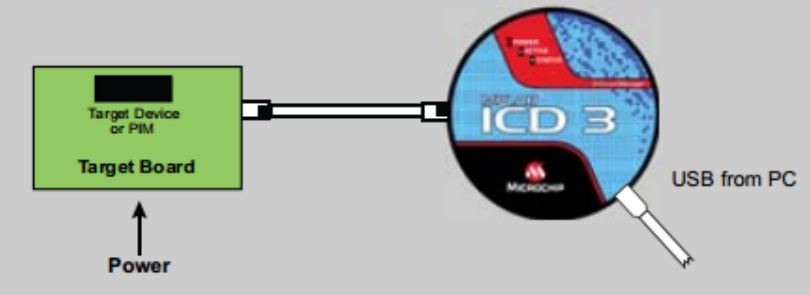


Imagen 28- Programador MPLAB ICD3

En la imagen anterior se ve el programador a usar y las conexiones que tiene:

- Conexión USB al PC
- Conexión al circuito de aplicación mediante conector RJ-11

Las conexiones de conector RJ-11 son:

Modular Connector Pin	Microcontroller Pin
1	$\overline{\text{MCLR}}/V_{PP}$
2	V_{DD}
3	Ground
4	PGD (ICSPDAT)
5	PGC (ICSPCLK)
6	PGM (LVP)

Imagen 29- Conexiones RJ-11

Esto indica que se deben conectar el pin 1 del PIC (MCLR) uno de los pares nombrados como PGEDx y PGECx en la **imagen 27**, por la cercanía al pin 1 se han escogido los pines 2 y 3, PGED3 y PGEC3 respectivamente.

Para la conexión del MCLR (Master Clear) se ha seguido el esquema proporcionado en las hojas de características del PIC.

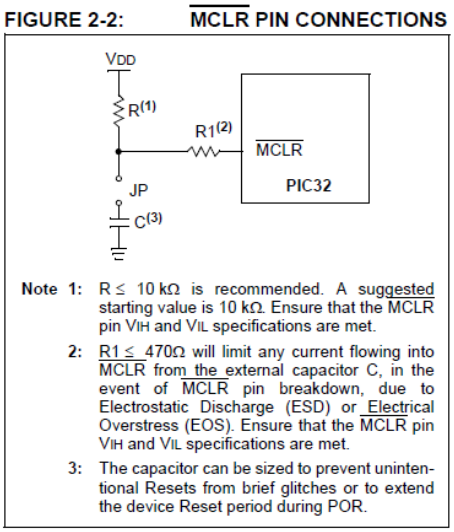


Imagen 30- Conexión MCLR

Los valores elegidos de R y R1 son de 10kΩ y 470Ω respectivamente, que cumplen con la recomendación de la nota de la imagen anterior. En el caso del Condensador se ha escogido un valor de 0.1uF que es el valor utilizado en la tarjeta de evaluación *PIC32 USB Digital Audio Accesory*, aunque el valor que se ponga no es muy importante, pues su función es la de prevenir reinicios no deseados.

En cuanto al resto de conexiones, se han tenido en cuenta una serie de precauciones presentes en las hojas de características del ICD3. Estas precauciones son:

- No usar condensadores de una capacidad superior a 100uF en Vdd
- No usar condensadores en MCLR
- No usar resistencias de pull-ups en PGC/PGD
- No multiplexar los pines PGC/PGD
- No usar condensadores en PGC/PGD
- No usar diodos en PGC/PGD

La inclusión del condensador en el pin MCLR se contradice con la recomendación anterior. Sin embargo, como se ha comentado anteriormente en la tarjeta de evaluación si se incluye este componente. Por esta razón, se ha decidido incluirlo ya que en caso de no ser necesario siempre se puede no conectar y cortocircuitar sus terminales (en la práctica el funcionamiento es correcto con el condensador).

También es necesario, como en la mayoría de componentes electrónicos, la inclusión de unos condensadores externos que aseguren el correcto funcionamiento del PIC. El propio fabricante incluye un esquema que indica cómo hacer esto:

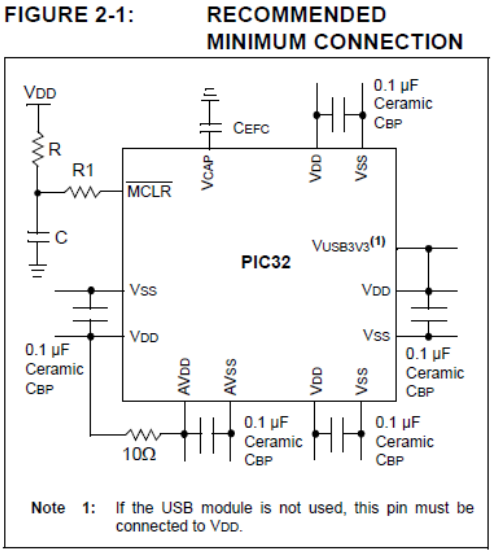


Imagen 31- Conexión Mínima Recomendada PIC32

En el plano **nº 5** se puede ver cómo se han implementado estas conexiones.

1.8.3.3. OSCILADOR

Un oscilador es un circuito que es capaz de convertir la corriente continua en una corriente que varía de forma periódica con el tiempo (definición de Wikipedia). Estas oscilaciones pueden ser senoidales, cuadradas, triangulares…

Un oscilador electrónico es fundamentalmente un amplificador cuya señal de entrada se toma de su propia salida a través de un circuito de realimentación. Se puede considerar que está compuesto por:

- Un circuito cuyo desfase depende de la frecuencia
- Un elemento amplificador
- Un circuito de realimentación

Todo microcontrolador o microprocesador requiere de un oscilador que le indique a qué velocidad debe trabajar. Algunos microcontroladores incluyen internamente un circuito oscilador, mientras que otros necesariamente necesitan de un oscilador externo.

En la página de microchip la sección 6 de *PIC32 Family Reference Manual: Oscillators* se detallan las distintas opciones en cuanto a la fuente de reloj de los PIC32. En concreto, existen cuatro fuentes de reloj:

- Oscilador Primario ($P_{OSC)}$ en los pines OSC1 y OSC2
- Oscilador Secundario ($S_{OSC)}$ en los pines SOSCI y SOSCO
- Oscilador RC Rápido Interno (FRC)
- Oscilador RC Interno de Bajo Consumo (LPRC)

Todas estas fuentes de reloj disponen de lazos de enganche de fase (PLLs), que son divisores y multiplicadores de entrada y salida programables que permiten escalar la

frecuencia de entrada del reloj para adaptarse a la aplicación. Esta fuente de reloj se puede cambiar mediante software durante la ejecución del programa.

A parte de las fuentes de reloj, cabe destacar que los PIC32 trabajan con tres relojes principales:

- Reloj del sistema(SYSCLK), usado por la CPU y algunos periféricos
- Reloj del bus de periféricos(PBCLK), usado por la mayoría de periféricos
- Reloj USB(USBCLK), usado por el módulo USB

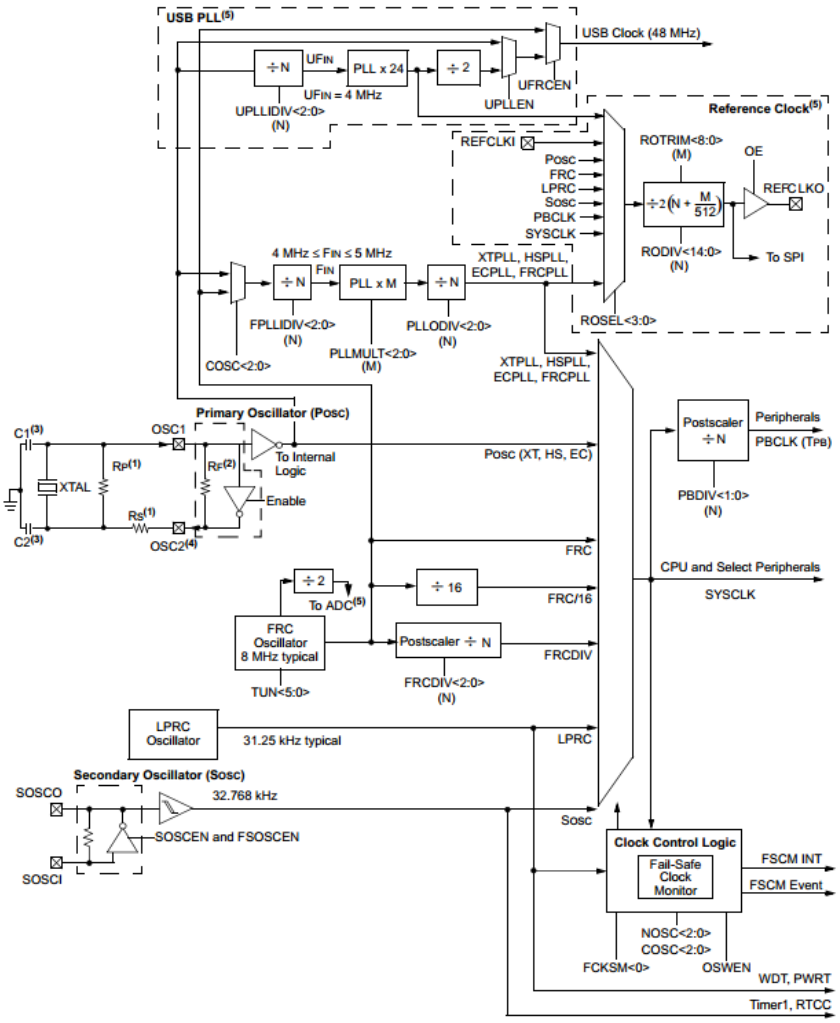


Imagen 32- Diagrama de bloques Sistema del Oscilador

Una vez conocido esto, se puede elegir la configuración a usar. Analizando el diagrama de bloques del sistema de oscilador del PIC se observa como con el oscilador interno de 8MHz es suficiente para obtener máxima velocidad tanto para el SYSCLK como para el PBCLK. Sin embargo, el FRC no permite alcanzar velocidades altas en el módulo USB, ya que para usar el USB PLL es necesario que este proceda de una fuente de reloj externa. Esto se puede ver con mayor claridad en el diagrama de la interfaz USB de las hojas de características del PIC.

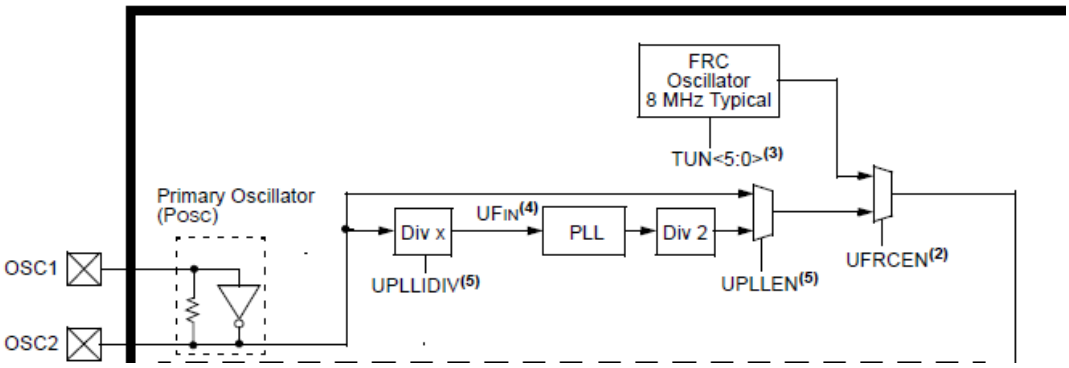


Imagen 33- Diagrama Oscilador módulo USB

Por esta razón, se ha decidido incluir un oscilador externo en el diseño. A la hora de hacer esto también hay dos posibilidades:

- Utilizar un cristal o resonador cerámico conectados a OSC1 y OSC2
- Utilizar un oscilador externo conectado a OSC1

La ventaja principal del cristal es que es más barato que el oscilador. Su desventaja reside en el hecho de que utiliza dos pines en vez de uno, como es el caso del oscilador. Como no sobran muchos pines en el PIC se ha optado por la segunda opción, ya que además para un número pequeño de unidades no supone una gran diferencia de precio. De las hojas de características del micro se extrae la forma de conectarlo.

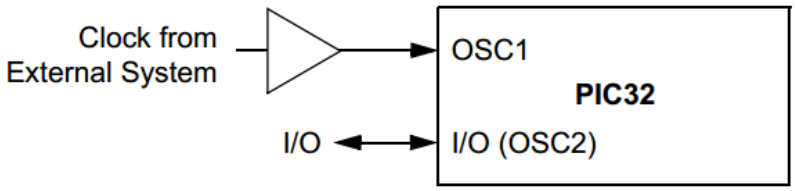


Imagen 34- Conexión Oscilador Externo

El reloj escogido es un oscilador de tipo MEMS del fabricante *ABRACON* de 12MHz que tiene un tamaño reducido, bajo consumo de corriente y gran estabilidad. El rango de su tensión de alimentación es de 1.7V a 3.6V con lo que se ajusta al resto de componentes. Se ha seleccionado un oscilador de 12MHz porque hay que usar necesariamente el PLL, ya que la frecuencia de entrada máxima al multiplicador del PLL es de 5MHz, como se ve en la **imagen 32**. El usar un oscilador de mayor frecuencia aumenta el precio del componente y no aporta nada.

Al igual que el resto de osciladores comerciales de este tipo, el oscilador escogido tiene 4 pines.

No	Pin Terminal
1	Standby
2	GND
3	Output
4	VDD

Imagen 35- Terminales Oscilador ABRACON

El esquema final se muestra en el plano **nº 5**.

1.8.3.4. CONFIGURACIÓN PPS

El Peripheral Pin Select (PPS) es una opción presente en muchos de los dispositivos de la familia PIC que permite la configuración y el emplazamiento de los periféricos en un amplio rango de pines de entrada/salida. Esta configuración se programa mediante software, incluso durante la ejecución del programa. Esta característica provee a los diseñadores de una gran cantidad de soluciones a la hora de satisfacer las necesidades de diseño.

Después de haber asignado los pines correspondientes al depurador y al oscilador, la asignación del resto de pines se ha realizado teniendo en cuenta el siguiente criterio: las conexiones a un mismo componente deben estar lo más próximas posibles de forma que se crucen el menor número posible de conexiones. De esta forma se reducirán los cruzamientos de pistas en la tarjeta simplificando la misma.

Utilizando como guía la **tabla 4** se asigna un pin a cada una de las funciones que se han establecido en el diseño del esquemático.

Los pines que pueden realizar funciones distintas están nombrados en la **imagen 27** (Relación de pines) con el prefijo RP seguido del nombre del puerto y el número del pin. Para saber las funciones que puede realizar cada pin hay que recurrir a las hojas de características del microcontrolador, concretamente al apartado de “Peripheral Pin Select”. Este apartado se divide en dos, dependiendo de si se quiere configurar un pin de entrada o uno de salida.

1.8.3.4.1. CONFIGURACIÓN PINES DE ENTRADA

En el caso de los pines de entrada nos encontramos con la siguiente tabla:

Peripheral Pin	[pin name]R SFR	[pin name]R bits	[pin name]R Value to RPn Pin Selection
INT4	INT4R	INT4R<3:0>	0000 = RPA0 0001 = RPB3 0010 = RPB4 0011 = RPB15 0100 = RPB7 0101 = RPC7 0110 = RPC0 0111 = RPC5 1000 = Reserved . . . 1111 = Reserved
T2CK	T2CKR	T2CKR<3:0>	
IC4	IC4R	IC4R<3:0>	
$\overline{SS1}$	SS1R	SS1R<3:0>	
REFCLKI	REFCLKIR	REFCLKIR<3:0>	
INT3	INT3R	INT3R<3:0>	0000 = RPA1 0001 = RPB5 0010 = RPB1 0011 = RPB11 0100 = RPB8 0101 = RPA8 0110 = RPC8 0111 = RPA9 1000 = Reserved . . . 1111 = Reserved
T3CK	T3CKR	T3CKR<3:0>	
IC3	IC3R	IC3R<3:0>	
$\overline{U1CTS}$	U1CTSR	U1CTSR<3:0>	
U2RX	U2RXR	U2RXR<3:0>	
SDI1	SDI1R	SDI1R<3:0>	
INT2	INT2R	INT2R<3:0>	0000 = RPA2 0001 = RPB6 0010 = RPA4 0011 = RPB13 0100 = RPB2 0101 = RPC6 0110 = RPC1 0111 = RPC3 1000 = Reserved . . . 1111 = Reserved
T4CK	T4CKR	T4CKR<3:0>	
IC1	IC1R	IC1R<3:0>	
IC5	IC5R	IC5R<3:0>	
U1RX	U1RXR	U1RXR<3:0>	
$\overline{U2CTS}$	U2CTSR	U2CTSR<3:0>	
SDI2	SDI2R	SDI2R<3:0>	
OCFB	OCFBR	OCFBR<3:0>	
INT1	INT1R	INT1R<3:0>	0000 = RPA3 0001 = RPB14 0010 = RPB0 0011 = RPB10 0100 = RPB9 0101 = RPC9 0110 = RPC2 0111 = RPC4 1000 = Reserved . . . 1111 = Reserved
T5CK	T5CKR	T5CKR<3:0>	
IC2	IC2R	IC2R<3:0>	
$\overline{SS2}$	SS2R	SS2R<3:0>	
OCFA	OCFAR	OCFAR<3:0>	

Imagen 36- Selección Pines de Entrada

En ella se puede ver, en la columna de la derecha, el nombre de los pines y, en la columna de la izquierda, el nombre de las distintas funciones que pueden desempeñar cada uno de esos pines. Un ejemplo es el pin 24 cuyo nombre es *RPB13* al que se le ha asignado la función de *SDI2* (pin de recepción de datos del módulo SPI2).

1.8.3.4.2. CONFIGURACIÓN PINES DE SALIDA

En este caso la tabla que aparece es la siguiente:

RPn Port Pin	RPnR SFR	RPnR bits	RPnR Value to Peripheral Selection
RPA0	RPA0R	RPA0R<3:0>	0000 = No Connect 0001 = U1TX 0010 = U2RTS 0011 = SS1 0100 = Reserved 0101 = OC1 0110 = Reserved 0111 = C2OUT 1000 = Reserved . . . 1111 = Reserved
RPB3	RPB3R	RPB3R<3:0>	
RPB4	RPB4R	RPB4R<3:0>	
RPB15	RPB15R	RPB15R<3:0>	
RPB7	RPB7R	RPB7R<3:0>	
RPC7	RPC7R	RPC7R<3:0>	
RPC0	RPC0R	RPC0R<3:0>	
RPC5	RPC5R	RPC5R<3:0>	
RPA1	RPA1R	RPA1R<3:0>	0000 = No Connect 0001 = Reserved 0010 = Reserved 0011 = SDO1 0100 = SDO2 0101 = OC2 0110 = Reserved 0111 = C3OUT . . . 1111 = Reserved
RPB5	RPB5R	RPB5R<3:0>	
RPB1	RPB1R	RPB1R<3:0>	
RPB11	RPB11R	RPB11R<3:0>	
RPB8	RPB8R	RPB8R<3:0>	
RPA8	RPA8R	RPA8R<3:0>	
RPC8	RPC8R	RPC8R<3:0>	
RPA9	RPA9R	RPA9R<3:0>	
RPA2	RPA2R	RPA2R<3:0>	0000 = No Connect 0001 = Reserved 0010 = Reserved 0011 = SDO1 0100 = SDO2 0101 = OC4 0110 = OC5 0111 = REFCLKO 1000 = Reserved . . . 1111 = Reserved
RPB6	RPB6R	RPB6R<3:0>	
RPA4	RPA4R	RPA4R<3:0>	
RPB13	RPB13R	RPB13R<3:0>	
RPB2	RPB2R	RPB2R<3:0>	
RPC6	RPC6R	RPC6R<3:0>	
RPC1	RPC1R	RPC1R<3:0>	
RPC3	RPC3R	RPC3R<3:0>	
RPA3	RPA3R	RPA3R<3:0>	0000 = No Connect 0001 = U1RTS 0010 = U2TX 0011 = Reserved 0100 = SS2 0101 = OC3 0110 = Reserved 0111 = C1OUT 1000 = Reserved . . . 1111 = Reserved
RPB14	RPB14R	RPB14R<3:0>	
RPB0	RPB0R	RPB0R<3:0>	
RPB10	RPB10R	RPB10R<3:0>	
RPB9	RPB9R	RPB9R<3:0>	
RPC9	RPC9R	RPC9R<3:0>	
RPC2	RPC2R	RPC2R<3:0>	
RPC4	RPC4R	RPC4R<3:0>	

Imagen 37- Selección Pines de Salida

En este caso el nombre de cada pin se encuentra en la columna de la izquierda, mientras que en la columna de la derecha está la función que cumple.

1.8.4. DISEÑO DEL PCB

Tras haber definido completamente el diseño del esquemático se procede al diseño de la tarjeta de circuito impreso. Esto permitirá implementar el circuito de forma consistente y haciéndolo fácilmente reproducible. Además de la tarjeta diseñada en el esquemático anterior se ha realizado otro PCB que reproduce la tarjeta de *Analog Devices* y que sirve para colocar en ella el acelerómetro *ADIS16003*. Por la simplicidad de esta tarjeta, no se ha realizado el esquemático de la misma, sino que se ha realizado directamente el PCB.

El PCB se ha realizado a dos caras principalmente por dos razones:

- La tarjeta no incluye muchos componentes por lo que el resultado final a dos caras tiene un tamaño satisfactorio
- De esta forma, la fabricación de la tarjeta se puede llevar a cabo en la Universidad

Los siguiente apartados se refieren al diseño del PCB de la tarjeta de interfaz cuyo esquemático se ha diseñado anteriormente. La tarjeta para la colocación del sensor se explica en el apartado titulado *TARJETA ADIS*.

1.8.4.1. REGLAS DE DISEÑO

Al igual que cualquier otro programa de diseño de circuitos impresos, en Design Spark es necesario definir una serie de reglas antes de iniciar el trazado del PCB. Estas reglas hacen referencia a la anchura de pistas y al espaciado entre diferentes elementos. Hay que tener en cuenta una serie de limitaciones que impone la fresadora con la que se va a fabricar el PCB:

- Ancho de pista mínimo: 0.3mm
- Separación entre pistas o pads: 0.3mm
- Diámetro mínimo de taladro: 0.6mm
- Pared mínima de la corona: 0.3mm
- Separación mínima entre pistas y el borde de la tarjeta: 0.3mm

Una vez conocido esto, se decide el ancho de pista a usar para el diseño. Se sabe que la resistencia de un conductor de cobre, es decir de una pista de la tarjeta, viene dada por:

$$R = \rho \frac{L}{A}$$

Ecuación 5- Resistencia de un conductor

$$\rho = resistividad\ cobre = 1.724 \times 10^{-6}\ \Omega\, cm$$

$$L = longitud\ conductor\ (cm)$$

$$A = sección\ transversal\ del\ conductor\ (cm^2)$$

De esta forma, el hecho de aumentar el ancho de pista provoca una disminución de la resistencia de la misma. En este diseño, la corriente que circula por las pistas es muy baja con lo que no es necesaria una gran anchura de pista. Sin embargo, se han elegido anchuras de pistas sobredimensionadas para evitar cualquier problema.

De esta forma, se han establecido las siguientes anchuras de pistas:

- Alimentación mínimo: 0.8mm
- Alimentación nominal: 0.8mm
- Señal mínimo: 0.3mm
- Señal nominal: 0.6mm

Para realizar esta elección se ha utilizado la experiencia previa en la fabricación de una tarjeta en otras asignaturas durante la carrera. En la siguiente imagen se puede observar una relación entre la anchura del conductor y la corriente que se ha tenido en cuenta al realizar la elección (apuntes asignatura *Diseño de Tarjetas Electrónicas*):

Conductor width (in)	Current (Amps)			
	½ oz	1 oz	2 oz	3oz
0.005	0.13	0.50	0.70	1.00
0.010	0.50	0.80	1.40	1.90
0.020	0.70	1.40	2.20	3.00
0.030	1.00	1.90	3.00	4.00
0.050	1.50	2.50	4.00	5.50
0.070	2.00	3.50	5.00	7.00
0.100	2.50	4.00	7.00	9.00
0.150	3.50	5.50	9.00	13.00
0.200	4.00	6.00	11.00	14.00

Imagen 38- Relación anchura de pista/corriente

Además de la anchura de pista, la siguiente imagen muestra las reglas de espaciado establecidas entre los diferentes elementos del diseño:

	Tracks	Pads	Vias	Shapes	Text
Tracks	0.5	0.3	0.3	0.5	0.3
Pads	0.3	0.3	0.3	0.3	0.3
Vias	0.3	0.3	0.3	0.3	0.3
Shapes	0.5	0.3	0.3	0.5	0.3
Text	0.3	0.3	0.3	0.3	0.3
Board	0.3	0.3	0.3	0.3	0.3

Imagen 39- Reglas espaciado *Design Spark*

Cabe destacar la separación entre pistas que se ha aumentado a 0.5mm con el objetivo de:

- Evitar errores en la fabricación
- Minimizar el acoplamiento capacitivo por la cercanía de los conductores

- Facilitar el proceso de colocación y soldadura de los componentes

1.8.4.2. UTILIZACIÓN DE PLANOS

En el caso de la tarjeta con el PIC se han incluido en el diseño del PCB dos planos: uno de masa y otro de +3.3V. Esto es un recurso habitual en el diseño de tarjetas electrónicas con el que se pretende minimizar las perturbaciones de las señales por conexiones comunes con otros elementos.

Cara superior ⟶ plano de masa

Cara inferior ⟶ plano de +3.3V

El plano de masa se ha separado en dos. Uno para la parte del convertidor y otro para el resto de los componentes. Esto se hace porque la salida del convertidor es una señal analógica. Para evitar perturbaciones de esta señal analógica y el resto del circuito se ha creado un plano de masa para esta parte de la tarjeta que se une con el plano de masa del resto de la tarjeta en un único punto.

1.8.4.3. EMPLAZAMIENTO DE LOS COMPONENTES

Se ha decidido emplazar los componentes en una única cara, la cara superior de la tarjeta. De esta manera, se dejará libre la cara inferior para poder atornillar la tarjeta a la caja donde irá colocada la misma. Además, de esta forma quedan accesibles desde la parte de arriba tanto los pulsadores, como los indicadores luminosos y la pantalla. Se ha procurado colocar los componentes por sectores tal como se ha explicado en el diseño del esquemático para, de esta forma, minimizar la longitud de las pistas y los cruzamientos. Otros criterios seguidos en el emplazamiento de los componentes son:

- Colocación del PIC en la parte central para facilitar la conexión con el resto de componentes
- Colocación de indicadores en la parte superior
    - Pantalla LCD ocupando toda la parte superior de la tarjeta
    - LEDS junto a la pantalla para rápida visualización
- Colocación de pulsador e interruptor parte superior izquierda
- Colocación de los conectores en el borde de la tarjeta para fácil acceso
- Colocación condensadores de cada integrado próxima a los mismos
- Colocación del oscilador próximo al PIC
- Minimizar longitud pistas alta frecuencia: SPI e I2C

1.8.4.4. TARJETA ADIS

Esta es la tarjeta que se ha fabricado para poder soldar el sensor en ella y colocarla encima del motor. Para realizarla, se ha tomado como referencia el siguiente esquema de la tarjeta de *Analog Devices* que tiene el mismo fin:

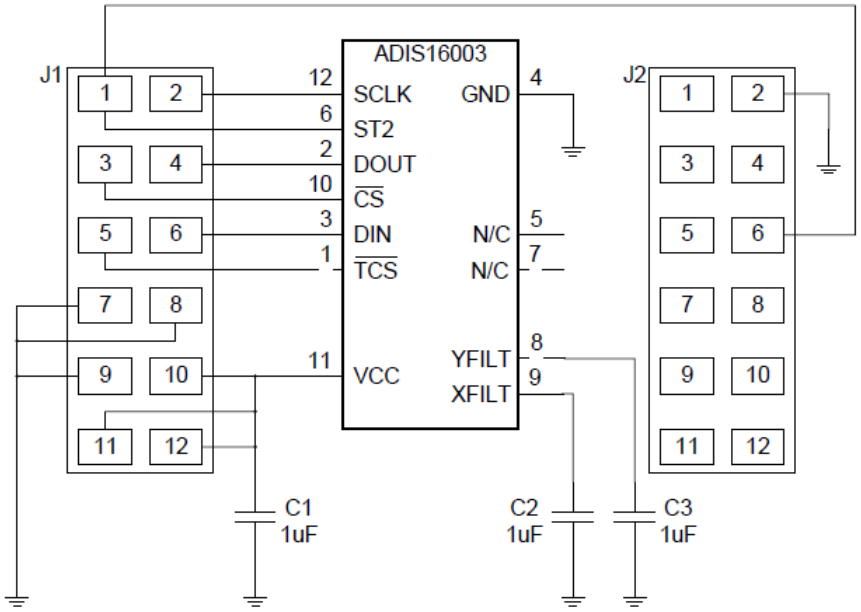


Imagen 40- Esquema tarjeta ADIS16003/PCB

Se ha eliminado el conector J2, ya que es el conector J1 el que sirve para la conexión del sensor. Los condensadores C2 y C3 son los que se utilizan para el control del ancho de banda del sensor.

Como esta tarjeta sólo tiene un único componente, que es el sensor, no se ha considerado necesaria la utilización de planos en el diseño del PCB. Así mismo, sólo se ha utilizado una anchura de pista para toda la tarjeta, en concreto 0.6mm.

1.8.4.5. RESULTADO FINAL

Los planos 8,9 y 10 muestran la situación de los componentes, el trazado de pistas y las vistas 3D de la tarjeta de la interfaz. Mientras que en los planos 11 y 12 se pueden ver el trazado de pistas y las vistas 3D de la tarjeta del sensor.

Hay que destacar que las vistas 3D se han generado con el *Design Spark* y simplemente nos dan una ligera idea del aspecto de la tarjeta, ya que la mayoría de los componentes no tienen definido correctamente su aspecto 3D.

Las siguientes imágenes muestran el PCB de las dos tarjetas fabricadas:

Imagen 41- Vista superior tarjeta_PIC

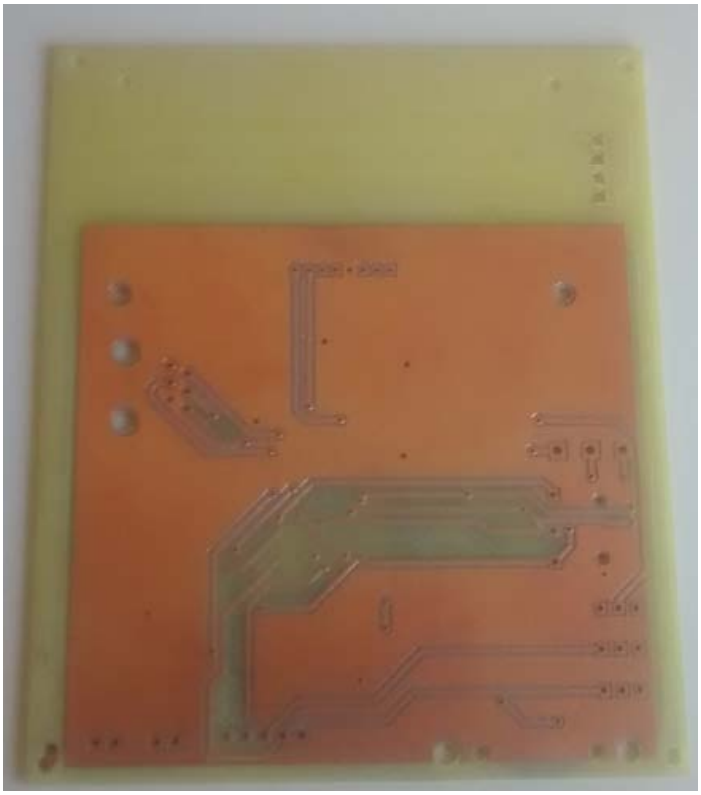


Imagen 42- Vista inferior tarjeta_PIC

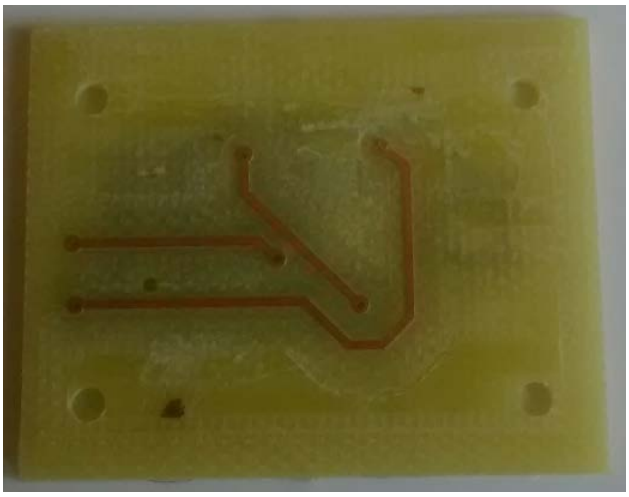


Imagen 43- Vista superior tarjeta_ADIS

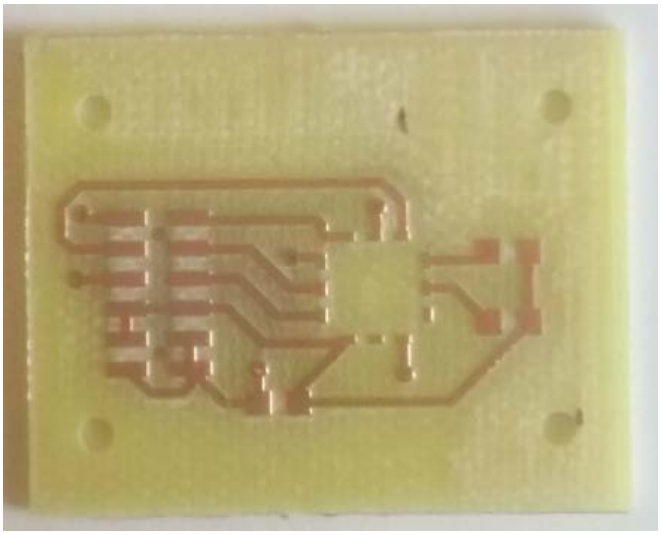


Imagen 44- Vista inferior tarjeta_ADIS

1.8.5. FIRMWARE PIC

Para poder llevar a cabo la aplicación deseada es necesario desarrollar el código para el microcontrolador. El lenguaje de programación utilizado para ello es C. Esto se debe a que es el lenguaje más extendido para este fin. Para desarrollar el código se ha utilizado el entorno de desarrollo integrado *MPLAB IDE* y el compilador empleado es el *MPLAB PIC C32 C Compiler*, también de *Microchip*. Todas estas herramientas se pueden descargar de forma gratuita desde la página web de *Microchip.*

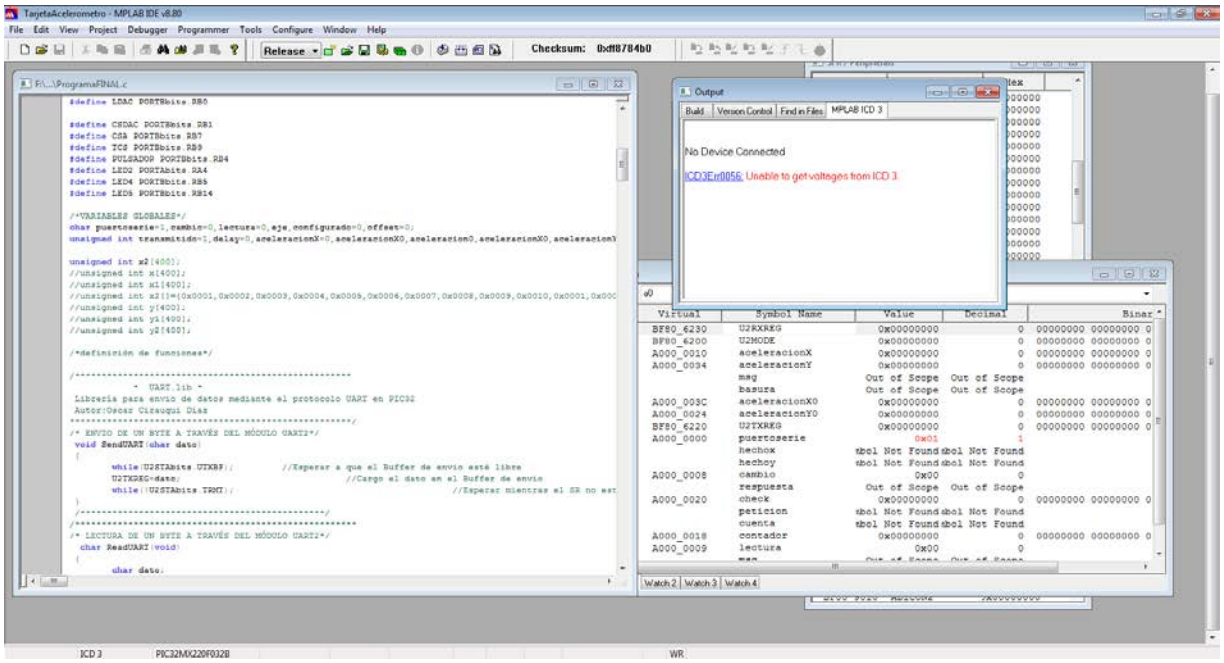


Imagen 45- Entorno de desarrollo MPLAB IDE

Antes de proceder al desarrollo del programa conviene tener claro que es lo que debe hacer éste.

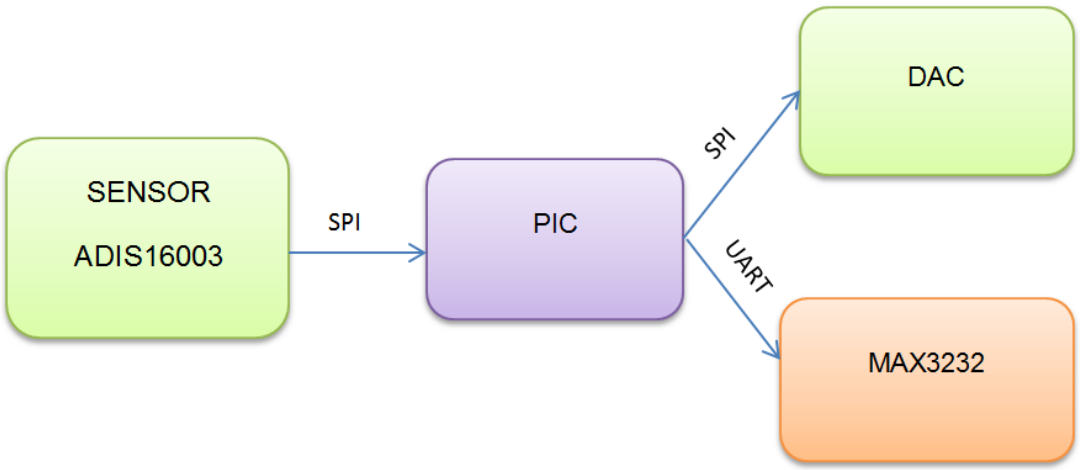


Diagrama 3- Esquema comunicaciones PIC

En la imagen anterior se puede observar un esquema de las comunicaciones del PIC. Las acciones que debe realizar el programa son:

1. Leer datos del sensor (SPI)
2. Procesar los datos adecuadamente
3. Enviar los datos a uno de estos dispositivos:
    3.1. Al DAC (SPI)
    3.2. Al MAX3232 (UART)

Una vez que se tiene esto claro, se procede al desarrollo de cada una de las partes del programa.

Cabe mencionar que existe una librería de Microchip para los PIC32 (plib.h) que incluye muchas definiciones y macros para el uso de los periféricos del PIC. Esta librería ha sido empleada para algunas partes del código como es el caso de las interrupciones, aunque ha sido descartada para los módulos SPI y UART porque no se ha conseguido un funcionamiento óptimo.

Es sabido que los microcontroladores incluyen diferentes fuentes de interrupción que permiten una programación más eficiente. Mediante las interrupciones es posible que el programa siga haciendo cosas sin necesidad de estar comprobando si ha ocurrido un evento determinado, ya que mediante la interrupción se avisa automáticamente cuando éste ha tenido lugar. En este programa no se ha estimado oportuno el uso de interrupciones para las transmisiones de datos porque no es necesario que el programa realice otras acciones.

1.8.5.1. BITS DE CONFIGURACIÓN DEL PIC

Los PIC32 tienen una serie de bits de configuración que deben ser configurados antes de iniciar nuestro programa. Se pueden encontrar en las hojas de características del PIC en el apartado *Special Features* Estos bits permiten configurar cosas muy importantes como son:

- Configuración del reloj
- Protocolo de programación y pines
- Habilitar la depuración
- Protección de código
- Configuración del WatchDog
- Protección de sectores de la memoria Flash

Algunos de estas configuraciones, como es el caso del reloj, se pueden modificar a posteriori (durante la ejecución del programa). Sin embargo, esto no será necesario en este caso.

Para poder establecer las configuraciones deseadas se debe emplear la siguiente instrucción (véase anexo con el programa para mayor claridad):

#pragma config nombre registro=configuración deseada

Los registros que se han configurado son:

- DEVCFG0
    - ICESEL: selección de PGEC3/PGED3
    - JTAGEN: JTAG deshabilitado
    - DEBUG: depurador habilitado
- DEVCFG1
    - FWDTEN: WatchDog Timer deshabilitado
    - FPBDIV: divisor del PBCLK=8

  - FNOSC: selección del FRC con PLL
  - FSOSCEN: deshabilitar oscilador secundario
  - POSCMOD: deshabilitar modo de reloj externo
  - FCKSM: habilitado el cambio de reloj
- DEVCFG2
  - FPLLODIV: PLL divisor de salida=2
  - FPLLMUL: multiplicador PLL=24
  - FPLLIDIV: divisor de entrada del PLL=2

Repasando la configuración anterior se deduce que el reloj se ha configurado en el modo *FRC*, es decir, se utiliza el reloj interno. Esto se ha hecho así porque el módulo *USB* no se ha implementado finalmente. Como el *FRC* es de 8MHz, tal y como se ha explicado con anterioridad, hay que dividirlo para que a la entrada del *PLL* sea de 4 o 5 MHz. Esto se consigue con *FPLLIDIV*. De forma que F_{IN}=8/2=4.

Observando el diagrama de bloques del oscilador (imagen 33) se tiene que:

$$SYSCLK = \frac{F_{IN} * FPLLMUL}{FPLLIDIV}$$

Ecuación 6- Frecuencia de SYSCLK

$$PBCLK = \frac{SYSCLK}{FPBDIV}$$

Ecuación 7- Frecuencia de PBCLK

Con lo que queda una frecuencia de *SYSCLK*=48MHz y de *PBCLK*=6MHz.

1.8.5.2. CONFIGURACIÓN DE PINES

Para configurar la función de los pines del PIC para que coincida con lo establecido en el esquemático (plano nº 5) se deben llevar a cabo dos acciones al inicio del programa, tal y como se explica en el apartado *I/O PORTS* de las hojas de características del PIC:

- Configuración de pines como entrada/salida en el registro TRIS
- Configuración del PPS

En el primer caso, simplemente, hay que escribir el valor deseado en el registro TRIS del puerto correspondiente de forma que un 0 indica salida y un 1 corresponde a entrada. En el programa también se ha escrito un cero en los registros ANSEL de cada puerto. Este registro indica los pines que se usan de forma analógica, en este caso ninguno.

Las siguientes tablas ilustran de forma clara la relación de pines:

PORT A	RA0	RA1	RA2	RA3	RA4
E/S			1	0	0
Nombre	PGED3	PGEC3	OSC1	RST_LCD	LED2

Tabla 5- Pines PORTA

PORT B	RB0	RB1	RB2	RB3	RB4	RB5	RB6	RB7	RB8
E/S	0	0			1	0	0	0	0
Nombre	LDAC	CSDAC	SDA2	SCL2	INT4	LED4		CSA	SDO2

Tabla 6- Pines PORTB_1

PORTB	RB9	RB10	RB11	RB12	RB13	RB14	RB15
E/S	0	0	1	0	1	0	0
Nombre	TCS	U2TX	U2RX		SDI2	LED5/V_{BUS}	SCK2

Tabla 7- Pines PORTB_2

La configuración del PLL requiere de más pasos. Primero es necesario desbloquear la modificación del mismo, ya que por defecto está bloqueada. Esto se hace escribiendo la siguiente secuencia en el registro OSCCON:

1. Escribir 0x46 sobre OSCCON<7:0>
2. Escribir 0x57 sobre OSCCON<7:0>
3. Borrar IOLOCK para desbloquear

Una vez hecho esto, ya es posible configurar el PPS como se desee. Para ello se recurre a la **imagen 36 o 37**, (configuración de los pines) dependiendo de si se trata de una entrada o una salida y se sigue el siguiente proceso:

1. Buscar en la tabla el nombre del pin de los periféricos que se desea configurar.
2. Asignar al registro asociado, (aparece como SFR en la tabla) el valor deseado.

1.8.5.3. LECTURA DE DATOS DEL ACELERÓMETRO

Como se ha mencionado previamente, el acelerómetro utiliza la interfaz SPI. Se necesita saber las características que permite la comunicación SPI de éste para configurar correctamente este protocolo en el PIC. Para conocerlas, se recurren a las hojas de características del *ADIS16003*, de las que se extrae la siguiente información:

- Formato datos de aceleración: 12 bits
- Pines interfaz SPI: 5 pines que ya se han definido en el esquemático
- 10KHz>F_{SCLK}<2MHz
- Modo SPI: CKP=1,CKE=0

La siguiente imagen ayuda a comprender mejor la comunicación:

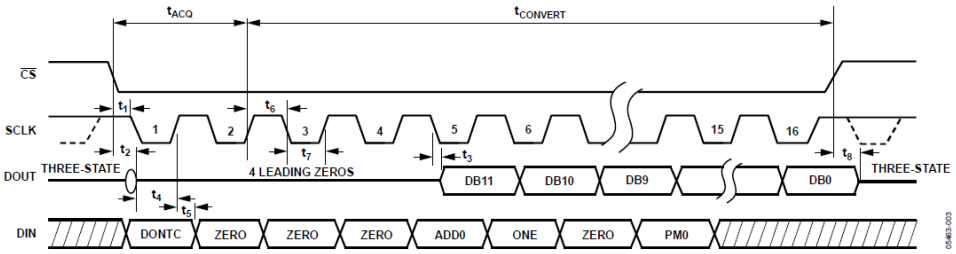


Imagen 46- Diagrama comunicación SPI ADIS16003

Una vez conocido esto, se procede a estudiar el módulo SPI del PIC. En primer lugar, se configura el módulo de forma que sea compatible con los requisitos del acelerómetro. En las hojas de características del PIC se explica que el módulo SPI tiene cuatro registros, a saber:

- SPIxCON: Registro de Control del SPI
- SPIxCON2: Registro de Control 2 del SPI
- SPIxSTAT: Registro de Estado del SPI
- SPIxBUF: Registro de búfer del SPI
- SPIxBRG: Registro que establece la velocidad en baudios del SPI

Por lo que para la configuración del módulo hay que fijarse en los registros SPI2CON, SPI2CON2 y SPI2BRG. Los bits a configurar en estos registros son:

- MSSEN: Selección de Esclavo deshabilitada
- MCLKSEL: PBCLK usado por el generador de baudios
- ENHBUF: Deshabilitar búfer ampliado
- DISSDO: pin SDO2 controlado por el módulo
- MODE: Modo de 16 bits
- SMP: dato de entrada muestreado a la mitad del tiempo de salida
- CKE=0
- CKP=1
- MSTEN: Modo Maestro
- DISSDI: SDI2 controlado por el módulo SPI

Para calcular el valor a cargar en el registro SPIxBRG se tiene la siguiente ecuación:

$$F_{SCK} = \frac{F_{PB}}{2 \cdot (SPIxBRG + 1)}$$

Ecuación 8- Frecuencia del reloj SPI

Dado esto, se ha configurado este registro con el valor de 1, con lo que F_{SCK}=1.5MHz.

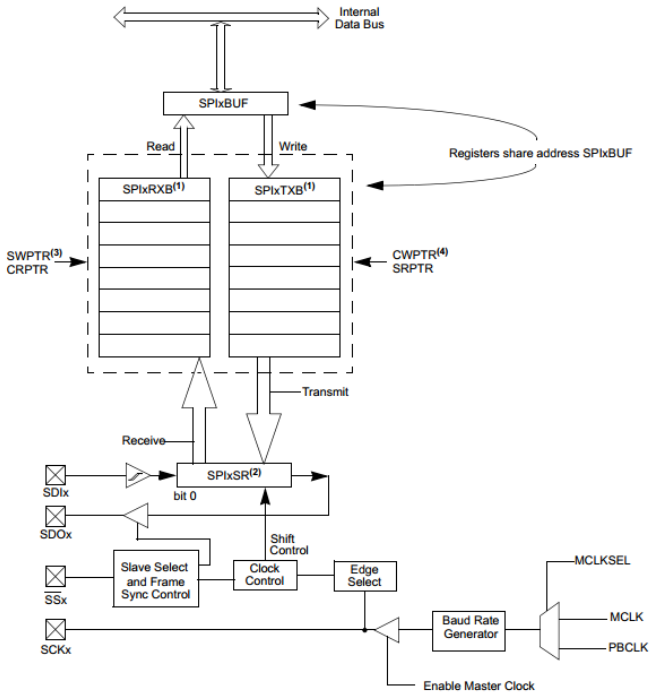


Imagen 47- Diagrama de bloques del módulo SPI

Como se ve en la imagen, la comunicación SPI se basa en el registro SPIxSR que es un registro de desplazamiento. Esto tiene dos implicaciones:

- Cada vez que entra un dato en el registro tiene que salir un dato.
- Cada vez que un dato sale del registro tiene que entrar un dato.

El programador no tiene acceso al SPIxSR, sino al SPIxBUF, el cual viene a funcionar como el primero. Esto se ve mejor en esta otra imagen:

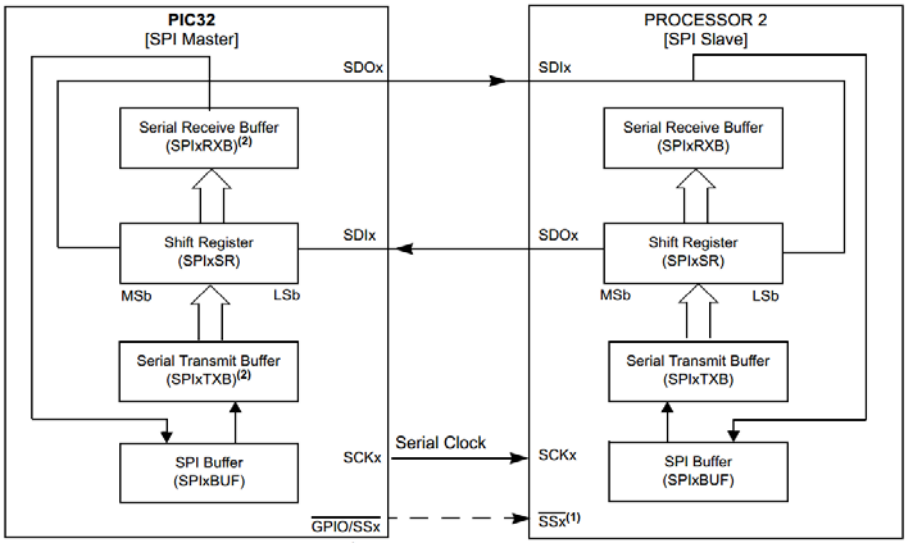


Imagen 48- Diagrama de conexión SPI

Por lo tanto, leer y escribir datos en el SPIxBUF se hace de la misma forma. La única diferencia reside si el dato a leer/escribir es importante o no. De esta forma se ha creado la siguiente función que sirve tanto para leer como para enviar datos a través del módulo SPI.

```
unsigned int SPItransfer(unsigned int dato)
{
    SPI2BUF=dato;
    while(SPI2STATbits.SPITBF);
    return(SPI2BUF);
}
```

A esta función siempre hay que darle un dato y siempre devuelve un dato.

Una vez configurado correctamente el módulo SPI se procede a realizar la programación de la comunicación. En la **imagen 46** se puede observar cómo hay que enviar un dato de configuración al acelerómetro. En las hojas de características del sensor se ve el significado de estos bits.

Bit	Mnemonic	Comments
7	DONTC	Don't care. Can be 1 or 0.
6 to 4	ZERO	These bits should be held low.
3	ADD0	This address bit selects the x-axis or y-axis outputs. A 0 selects the x-axis; a 1 selects the y-axis.
2	ONE	This bit should be held high.
1	ZERO	This bit should be held low.
0	PM0	This bit selects the operation mode for the accelerometer; set to 0 for normal operation and 1 for power-down mode.

Imagen 49- Registro de Control ADIS16003

De la imagen se deduce que habrá que enviar dos datos distintos al acelerómetro en función de cuál de los dos ejes de aceleración se desea leer:

- Eje X=0x0400
- Eje Y=0x0C00

Además de esto es necesario esperar a que finalice la comunicación después de usar la función *SPItransfer*. Esto se realiza mediante la siguiente instrucción:

```
while(SPI2STATbits.SPIBUSY);
```

1.8.5.4. ENVIO DE DATOS AL DAC

Como se ha visto anteriormente, el convertidor analógico-digital también utiliza el protocolo SPI. Al igual que en el caso del acelerómetro hay que definir los requisitos del mismo en lo referente a la comunicación SPI.

- Datos de entrada: 12 bits
- Pines interfaz SPI: 4 pines que ya se han definido en el esquemático
- F_{SCLK}<20MHz
- Modo SPI: CKP=1,CKE=0 o CKP=0,CKE=1

Tras ver esto, se ha escogido la misma configuración que en el caso del sensor, ya que es compatible con ésta.

La función a usar para la transmisión SPI es la misma que en el apartado anterior, pero en este caso sólo se envían datos al DAC por lo que el dato que se obtiene al enviar no interesará. Al igual que en el caso del sensor, es necesario enviar un dato de configuración al inicio del dato de aceleración. En las hojas de características del *MCP4822* se puede ver como es el registro de control sobre el que se escribe este dato:

Upper Half:

W-x	W-x	W-x	W-0	W-x	W-x	W-x	W-x
$\overline{A}$/B	—	$\overline{GA}$	$\overline{SHDN}$	D11	D10	D9	D8
bit 15							bit 8

bit 15 **$\overline{A}$/B:** DAC_A or DAC_B Select bit
1 = Write to DAC_B
0 = Write to DAC_A

bit 14 — Don't Care

bit 13 **$\overline{GA}$:** Output Gain Select bit
1 = 1x (V_{OUT} = V_{REF} * D/4096)
0 = 2x (V_{OUT} = 2 * V_{REF} * D/4096)

bit 12 **$\overline{SHDN}$:** Output Power-down Control bit
1 = Output Power-down Control bit
0 = Output buffer disabled, Output is high-impedance

bit 11-0 **D11:D0:** DAC Data bits
12-bit number "D" which sets the output value. Contains a value between 0 and 4095.

Imagen 50- Registro de Control MCP4822

Se establece la ganancia en 2 para obtener una tensión de salida lo más alta posible. Estando el rango de tensión de salida entre 0V y 4.096V.Esto viene dado por la ecuación

$$V_{OUT} = \frac{2.048V \cdot G \cdot D_N}{2^n}$$

Ecuación 9- V_{OUT} MCP4822

G: ganancia

D_N: dato digital de entrada

n: resolución=12

De esta manera el único dato a cambiar es el bit 15 dependiendo de si se quiere escribir en el canal A o B, siendo el dato a enviar:

- Aceleracion eje X=0x1AAA
- Aceleracion eje Y=0x9AAA

Las tres A corresponden a los 12 bits de la aceleración leída del sensor.

1.8.5.5. ENVIO DE DATOS AL MAX3232

Para enviar los datos correctamente al MAX3232 y poder implementar el protocolo serie de forma adecuada, en primer lugar, se deben configurar las distintas opciones del módulo UART. Los registros que controlan el funcionamiento de este módulo son:

- UxMODE: Registro del Modo de UARTx
- UxSTA: Registro de Estado y de Control del UART
- UxTXREG: Registro de Transmisión del UARTx
- UxRXREG: Registro de Recepción del UARTx
- UxBRG: Generador de Baudios del UARTx

De esta forma, la configuración establecida para el funcionamiento del UART2 es:

U2MODE

ON: Habilitado módulo UART2

BRGH: Habilitar modo de alta velocidad-4x

U2STA:

URXEN: Habilitado el receptor

UTXEN: Habilitado el transmisor

U2BRG=12

El cálculo del valor de U2BRG para el modo de alta velocidad se halla mediante la ecuación siguiente:

$$Baud\ Rate = \frac{F_{PB}}{4 \cdot (UxBRG + 1)}$$

$$UxBRG = \frac{F_{PB}}{4 \cdot Baud\ Rate} - 1$$

Ecuación 10- UART BaudRate con BRGH=1

La velocidad obtenida con este valor es de 115200 baudios. Esta velocidad es un estándar para las comunicaciones a través del puerto serie. Cabe mencionar que de la ecuación anterior el valor que se obtiene para 115200 es 12,02. Sin embargo, en el registro no se puede establecer este valor, sino que hay que poner 12. Esto hace que haya un error en la velocidad de 0.16%. En este caso es pequeño y el funcionamiento es bueno, pero es importante que este error no sea elevado, ya que de ser así se ocasionarían errores en la lectura de los datos por parte del receptor.

Como se explica en el **anexo I**, el protocolo UART es asíncrono y también funciona con un registro de desplazamiento, por lo que la forma de funcionar es similar al SPI. Sin embargo, en este caso se trabaja con dos registros distintos: U2TXREG para enviar y U2RXREG para recibir. Por este motivo, se han creado dos funciones distintas para enviar y recibir datos:

- Envío de datos a través del UART2

```
void SendUART(char dato)
{
    while(U2STAbits.UTXBF); //Esperar a que el Buffer de envio esté libre
    U2TXREG=dato;                    //Cargo el dato en el Buffer de envio
    while(!U2STAbits.TRMT);          //Esperar mientras el SR no esté vacío y haya operaciones pendientes en el módulo UART
}
```

A esta función se le da el dato de 8 bits a enviar.

- Lectura de datos del UART2

```
char ReadUART(void)
{
    char dato;
    dato=U2RXREG;                    //Leo el dato del buffer
    while(!U2STAbits.TRMT);          //Esperar mientras el SR no esté vacío y haya operaciones pendientes
    while(!U2STAbits.RIDLE);         //Espero a que finalice la recepción
    return dato;
}
```

La función devuelve el dato de 8 bits recibido.

1.8.5.6. INTERRUPCIONES

En esta aplicación se requiere el uso de dos interrupciones. Una de ellas para el pulsador y otra para un temporizador. Es necesaria la interrupción del pulsador para poder cambiar el modo de funcionamiento en el programa en cualquier momento,

mientras que el temporizador se necesita para muestrear el sensor a una frecuencia concreta. Antes de proceder a configurar cada una de ellas, existen una serie de generalidades sobre los PIC32 que se necesitan conocer.

La arquitectura del PIC32 puede manejar hasta 64 vectores de interrupción distintos y puede soportar hasta 96 fuentes de interrupción distintas. Cada fuente de interrupción puede tener su propia rutina de interrupción (ISR) en un vector individual, o varias fuentes de interrupción pueden compartir la misma ISR dentro de un solo vector de interrupción.

Cada fuente de interrupción tiene 7 bits de control agrupados en distintos SFRs. Estos bits se describen a continuación:

- **Bit de Habilitación de Interrupción** (terminado en IE):
  0 ⟶ Fuente de interrupción deshabilitada
  1 ⟶ Fuente de interrupción habilitada
  Todas las fuentes de interrupción están deshabilitadas al reset.
- **Bit de bandera de interrupción** (terminado en IF): se activa automáticamente cuando se dispara un evento de interrupción. Se debe poner a 0 por código.
- **Bits de nivel de prioridad grupal** (terminado en IP): son tres bits que determinan la prioridad de la interrupción. Cuando ocurren dos disparos de interrupción al mismo tiempo, se ejecutará primero la fuente de interrupción con mayor prioridad.
- **Bits de nivel de subprioridad** (terminado en IS): son dos bits usados para especificar cuatro niveles de subprioridad distintos dentro de un mismo grupo.

Además de estos dos niveles de prioridad, en el caso de que dos fuentes de interrupción se disparen al mismo tiempo, se ejecutará primero la que tenga la mayor prioridad. En el **anexo 2.3** se pueden ver todas las fuentes de interrupción con sus bits asociados y el orden de prioridad natural, que se corresponde con el orden de la tabla.

El PIC32 permite usar un sólo vector para el manejo de varias fuentes de interrupción, o vectores individuales para cada una de las fuentes. Para este proyecto se ha decidido usar un solo vector, debido a que únicamente se tienen dos fuentes de interrupción. Este modo se llama *Single Vector*. Para su activación se ha hecho uso de la siguiente instrucción de la librería *plib.h*:

```
INTEnableSystemSingleVectoredInt();
```

La rutina de atención a la interrupción en este caso es:

```
void __ISR( 0, ipl1) InterruptHandler (void)
{
  //aquí va el código de la rutina de interrupción
}
```

El argumento *(0, ipl1)* de este código se refiere a que se usará el vector 0 con prioridad 1. Dentro de la rutina anterior hay que comprobar qué bandera de interrupción es la que se ha activado.

1.8.5.6.1. PULSADOR: INT4

En este caso, el pulsador es una interrupción externa que se ha configurado en el PPS como INT4. La interrupción se activa cuando se produce una transición del estado del pin de 0 a 1 o de 1 a 0.Cabe mencionar, que esta interrupción es prioritaria respecto al temporizador. Lo primero para configurar la interrupción es establecer los 7 bits explicados anteriormente asociados a esta interrupción:

- **INT4IP**: prioridad de la interrupción 1 (debe coincidir con el valor de la rutina de atención a la interrupción)
- **INT4IS**: subprioridad de la interrupción de 3 (nivel más alto de prioridad)
- **INT4IF**: se pone a 0 inicialmente
- **INT4IE**: habilitar interrupción

1.8.5.6.2. TIMER 1

La familia PIC32 tiene dos tipos de temporizadores o timers:

- **Tipo A**
  - Temporizador/contador de 16 bits síncrono/asíncrono
  - Puede funcionar durante el modo sleep de la CPU
  - Predivisores seleccionables por software: 1:1,1:8,1:64,1:256.
- **Tipo B**
  - Temporizador/contador de 32 bits síncrono
  - Capacidad de disparador de eventos
  - Predivisores seleccionables por software: 1:1,1:2,1:4,1:8,1:16,1:32,1:64,1:256.

Para este programa cualquiera de los dos tipos son válidos. Se ha elegido un temporizador del tipo A, en concreto, el timer 1. El primer paso es configurar la interrupción:

- **T1IP**: prioridad de la interrupción 1 (debe coincidir con el valor de la rutina de atención a la interrupción)
- **T1IS**: subprioridad de la interrupción de 2 (menor que en el caso anterior)
- **T1IF**: se pone a 0 inicialmente
- **T1IE**: habilitar interrupción

Los registros que determinan el comportamiento del mismo son:

- **T1CON**: Registro de Control Timer de Tipo A
- **TMR1**: registro del Timer
- **PR1**: Registro de Periodo

En el registro T1CON simplemente hay que habilitar el bit ON.

En el registro TMR1 se guarda el valor actual de la cuenta del timer. Este registro se incrementa cada PBCLK/N ciclos de reloj, siendo N el valor del predivisor. Cuando el valor de TMR1 coincide con el valor establecido en PR1, TMR1 se pone a 0 y comienza un nuevo ciclo. El valor de PR1 para el tiempo deseado viene dado por:

$$PR1 = \frac{Tiempo}{T_{PBCLK} \times PSF}$$

Ecuación 11- Cálculo TIMER1

Tiempo: tiempo deseado en segundos

PSF: valor del Prescaler

El valor es variable en el programa dependiendo del periodo de muestreo elegido.

1.8.5.7. DIAGRAMA DE FLUJO

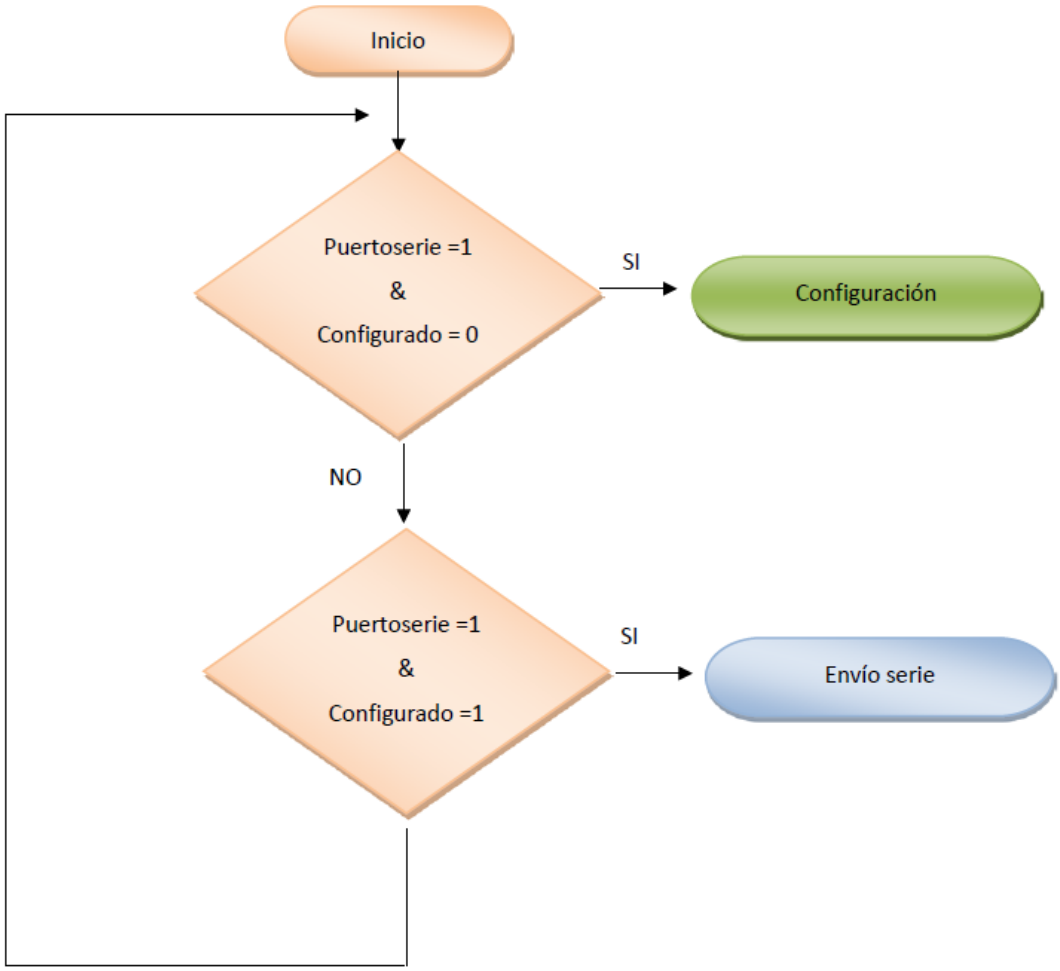


Diagrama 4- Programa PIC. Inicio

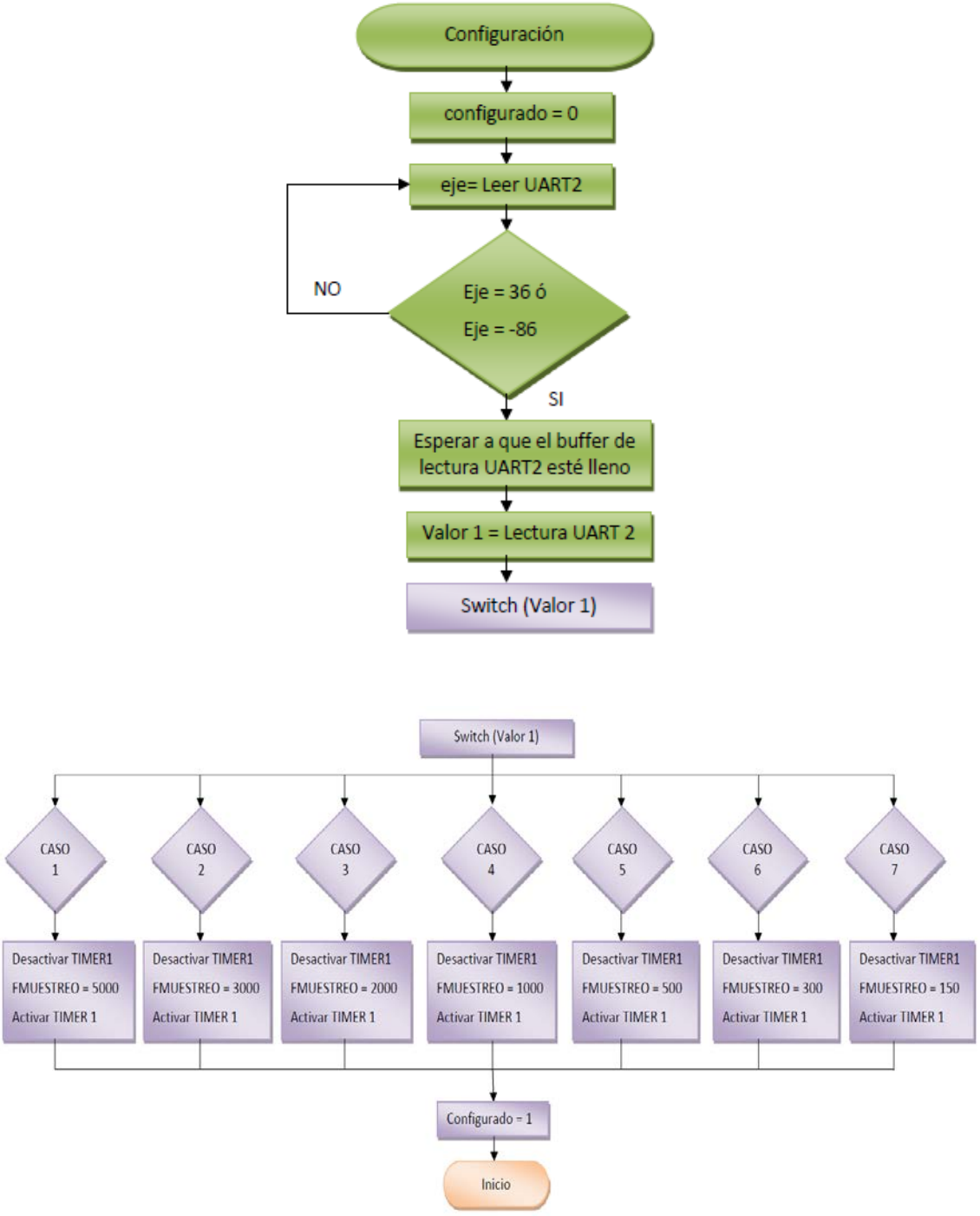


Diagrama 5- Programa PIC. Configuración

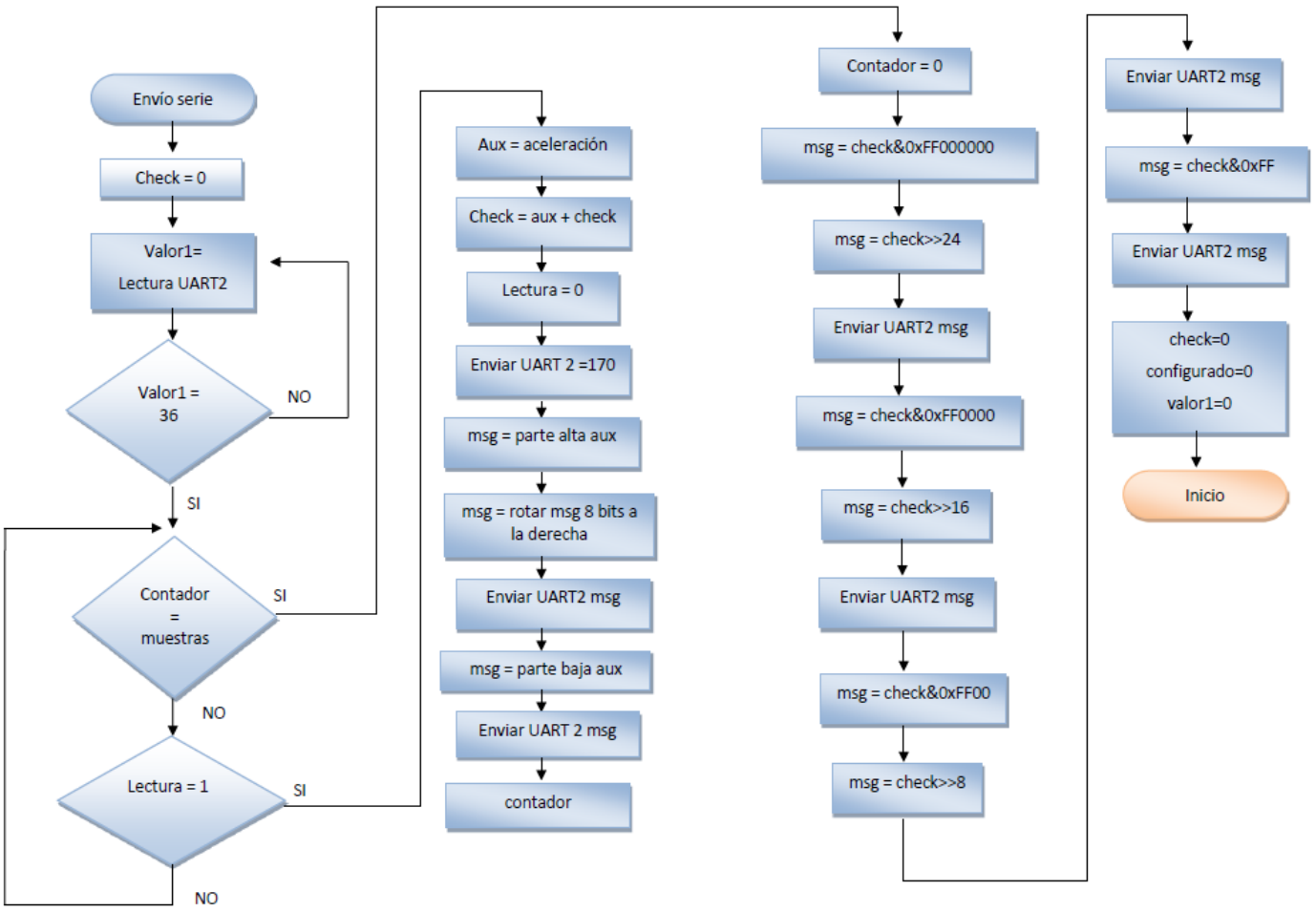


Diagrama 6- Programa PIC. EnvioSerie

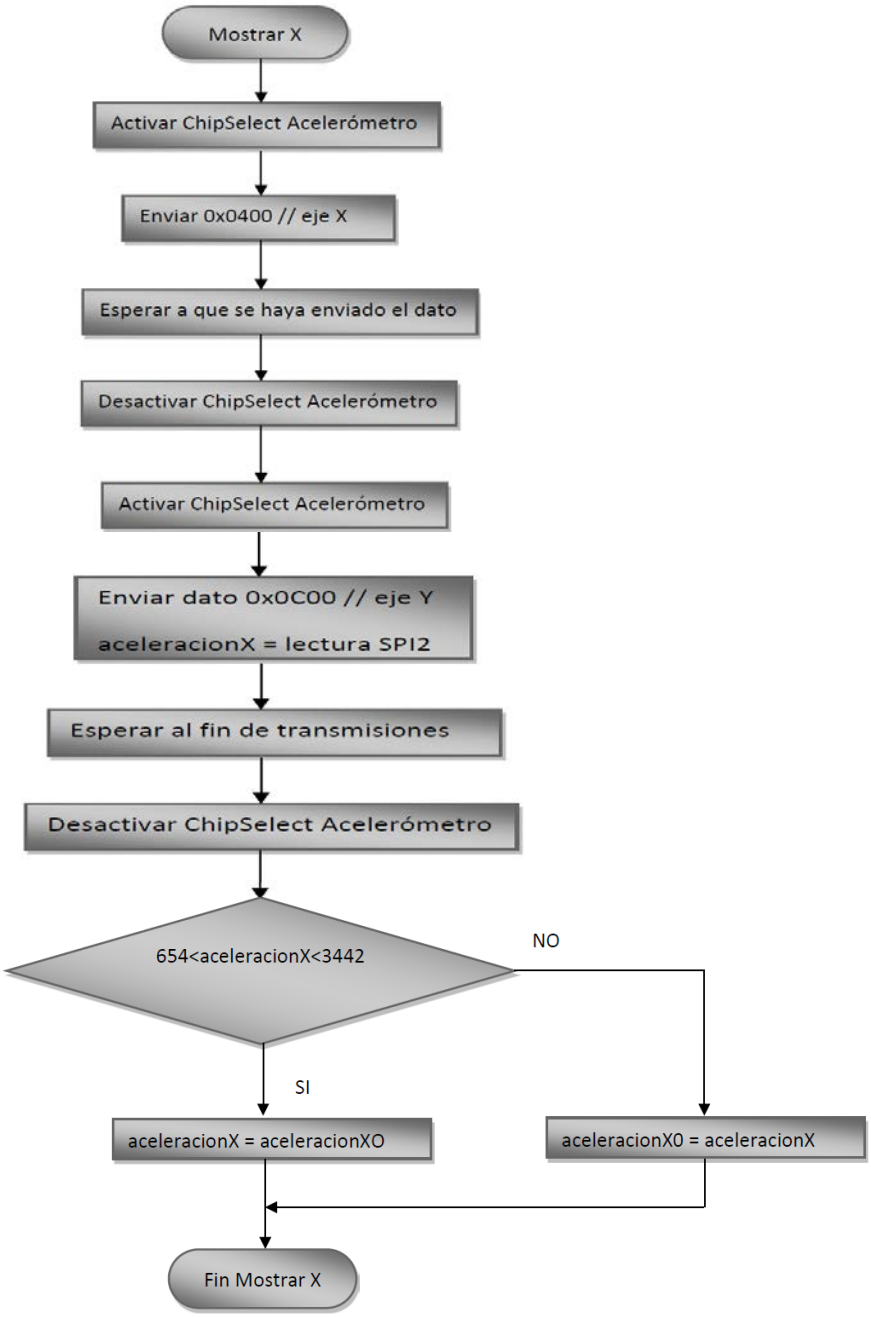


Diagrama 7- Programa PIC. MostrarX

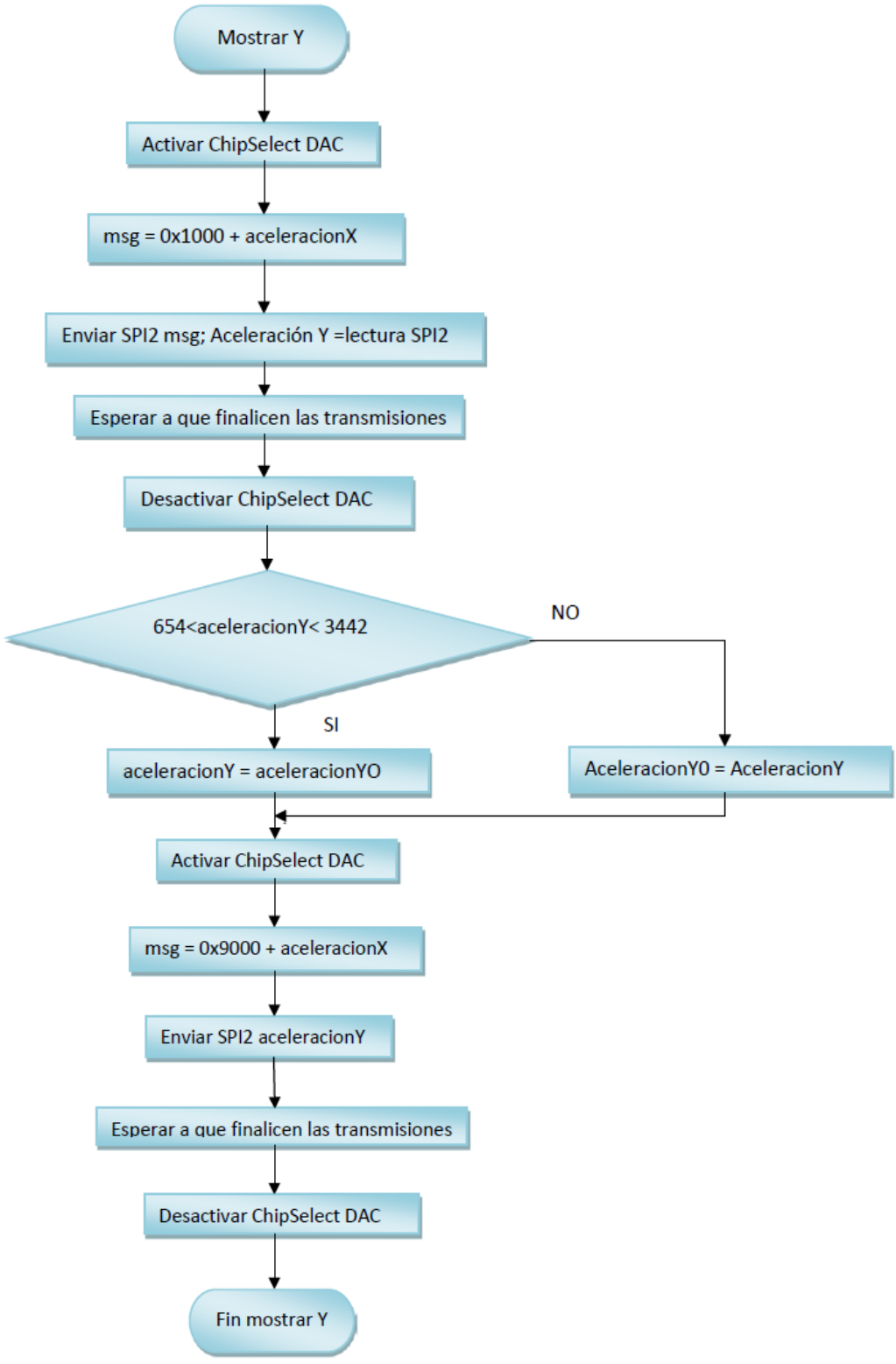


Diagrama 8- Programa PIC. MostrarY

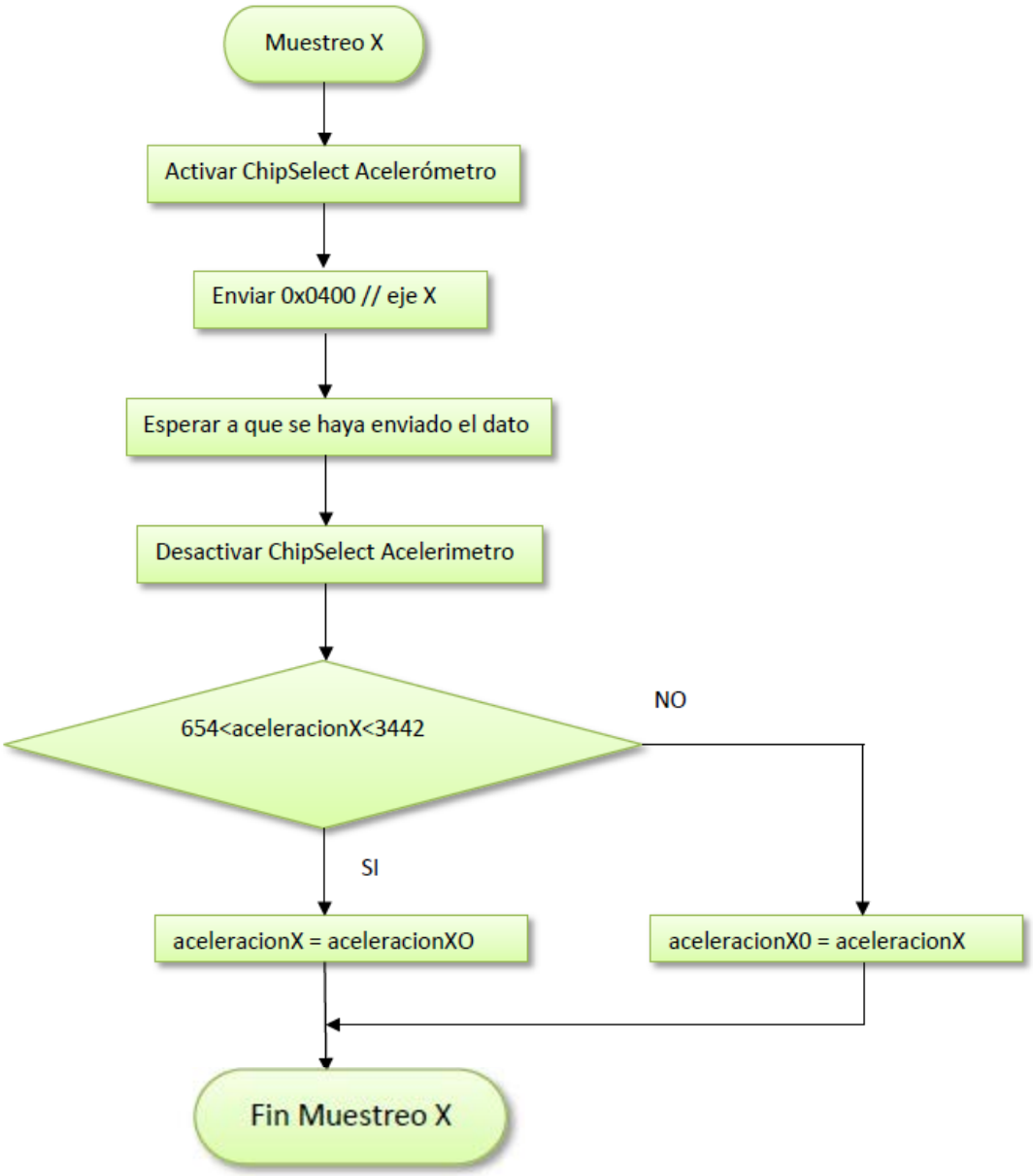


Diagrama 9- Programa PIC. MuestreoX

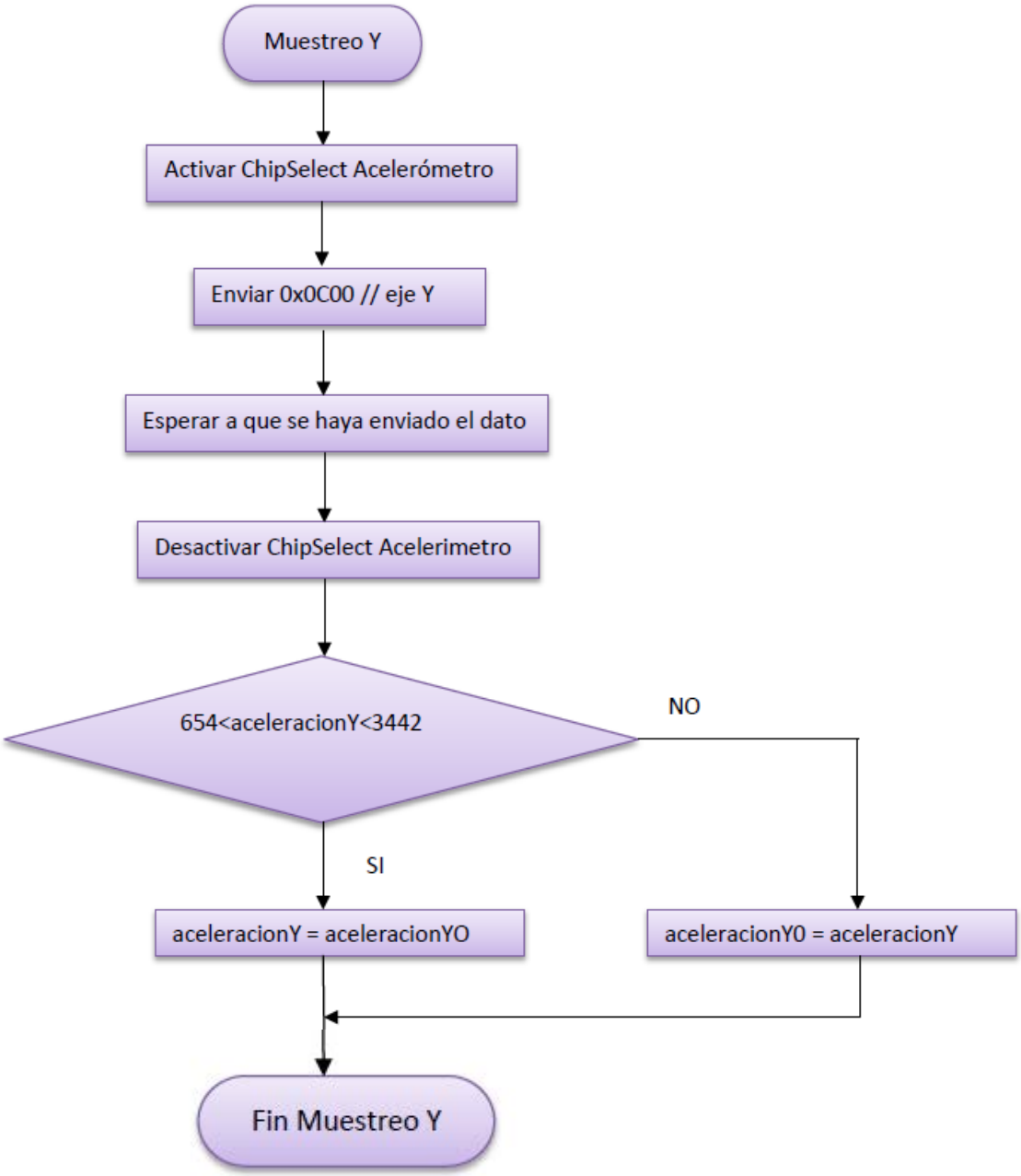


Diagrama 10- Programa PIC. MuestreoY

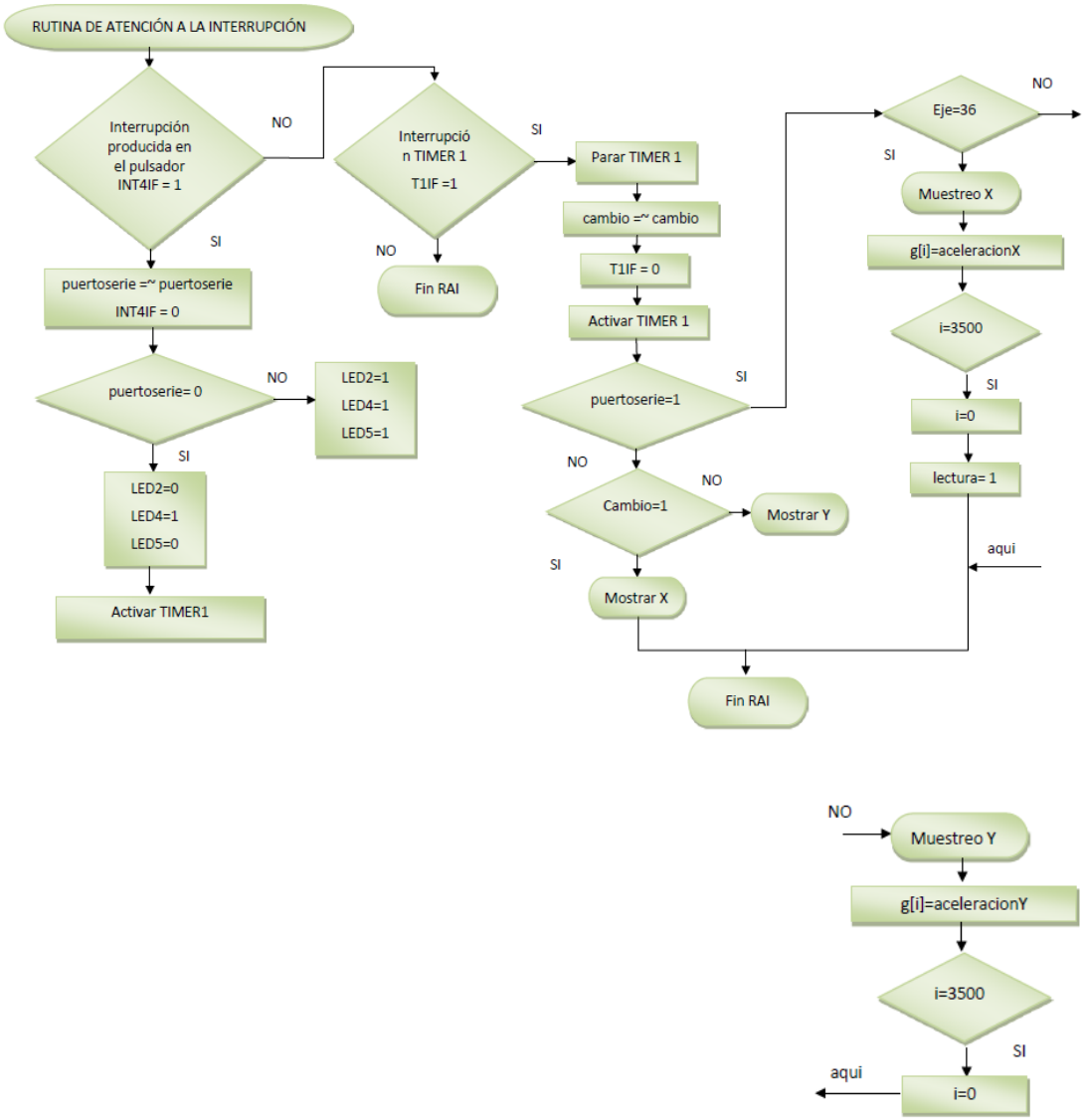


Diagrama 11- Programa PIC. RAI

1.8.6. PROGRAMA MATLAB

MATLAB (abreviatura de *MATrix LABoratory*, "laboratorio de matrices") es una herramienta de software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M) y servicio de especie. Está disponible para las plataformas Unix, Windows, Mac OS X y GNU/Linux.

Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware. El paquete MATLAB dispone de dos herramientas adicionales que expanden sus prestaciones, a saber, Simulink (plataforma de simulación multidominio) y GUIDE (editor de interfaces de usuario - GUI). Además, se pueden ampliar las capacidades de MATLAB con las *cajas de herramientas* (*toolboxes*); y las de Simulink con los *paquetes de bloques* (*blocksets*).

Estas características han llevado a la elección de MATLAB para la creación de un programa de interfaz de usuario que permita leer los datos de aceleración enviados por el puerto serie y graficarlos en cualquier PC.

1.8.6.1. PUERTO SERIE EN MATLAB

Para utilizar el puerto serie en MATLAB existe una clase creada que facilita mucho el uso del mismo. Para poder usar esta clase simplemente hay que crear un objeto de la misma con la siguiente instrucción:

```
obj = serial('port')
```

Una vez creado el objeto, se deben definir las siguientes propiedades para configurar correctamente la comunicación:

```
obj.BaudRate=115200;  %configura la velocidad de transmisión

set(port,'StopBits',1);  % se configura bit de parada a uno
set(port,'DataBits',8);  % se configura que el dato es de 8 bits, debe estar entre 5 y 8
set(port,'Parity','none');  % se configura sin bit de paridad
set(PS,'InputBufferSize',90000);  % se elige el tamaño del buffer de lectura
fopen(PS);  % abrir comunicación
```

Para leer los datos, se emplea la instrucción

```
fread(port,nº datos a leer)
```

Es importante que la lectura de los datos se realice en una instrucción, de forma que los datos sean leídos directamente en un vector. De otra forma, esta lectura de demorará considerablemente.

Tras acabar la comunicación, es necesario cerrar el puerto serie para poder usarlo posteriormente con otra aplicación. Esto se lleva a cabo mediante la siguiente instrucción:

```
fclose(port);
```

Las instrucciones anteriores son las básicas para la configuración del puerto serie. Sin embargo, para la aplicación deseada es necesario que los datos del puerto serie se lean de forma continua. Para ello, MATLAB dispone de BytesAvailableFcnMode. Esta opción, genera un evento después de que un número específico de bytes están disponibles en el búfer de entrada o después de un terminador.
En este proyecto, se ha configurado para que el evento de BytesAvailableFcnMode se genere cada vez que se reciba en el búfer de entrada el número de bytes correspondiente al número de muestras elegidas en la aplicación. Con esto se consigue que una vez elegida todas las opciones en la aplicación, se reciban los datos del puerto serie de forma continua. Esto se configura con las instrucciones siguientes:

```
set(port,'BytesAvailableFcnCount',(handles.muestras)*3);%elección número de bytes que activan el evento
set(port,'BytesAvailableFcnMode','byte');%evento generado por cuando lleguen un determinado número de bytes
port.BytesAvailableFcn = {@mycallback,handles};%nombre del evento generado
```

Una vez hecho esto, la subrutina que se ejecutará cada vez que se genera el evento es:

```
function mycallback(hObject,eventdata,handles) %código a ejecutar cada vez que se genere el evento
```

Otra cosa que cabe mencionar es la forma de abrir el puerto serie en la GUI. Como se puede seleccionar un puerto en ella, para abrir el puerto correctamente se usa la siguiente instrucción:

```
try
   fopen(port);
   set(handles.EstadoPuerto, 'string','conectado');
catch e
   set(handles.EstadoPuerto, 'string', 'desconectado');
   errordlg(e.message);
end
```

Esta instrucción intenta abrir el puerto seleccionado y, en caso de no ser posible, muestra un mensaje de error.

1.8.6.2. MATLAB GUI

Para crear interfaces gráficas en Matlab hay que ejecutar el comando *guide*, con lo que aparecerá la siguiente ventana:

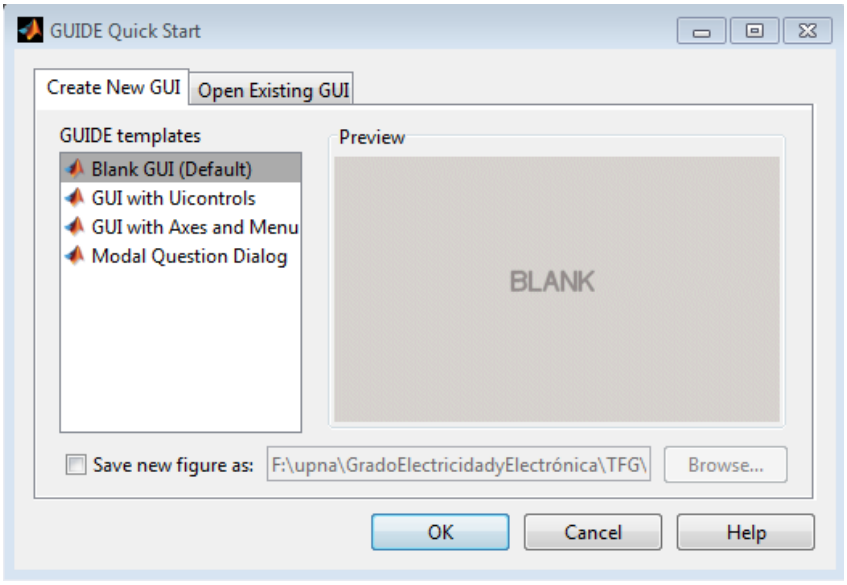


Imagen 51- Creación de GUI en MATLAB

Una vez aquí se selecciona la opción de ***Blank GUI***. Tras esto aparecerán dos ficheros: uno con extensión *.m* y otro con extensión *.fig.* El primero es el ejecutable donde se encuentra el código del programa asociado a la GUI, mientras que el segundo es la ventana gráfica del programa creado, la cual se puede editar añadiendo controles como botones, gráficos etc.

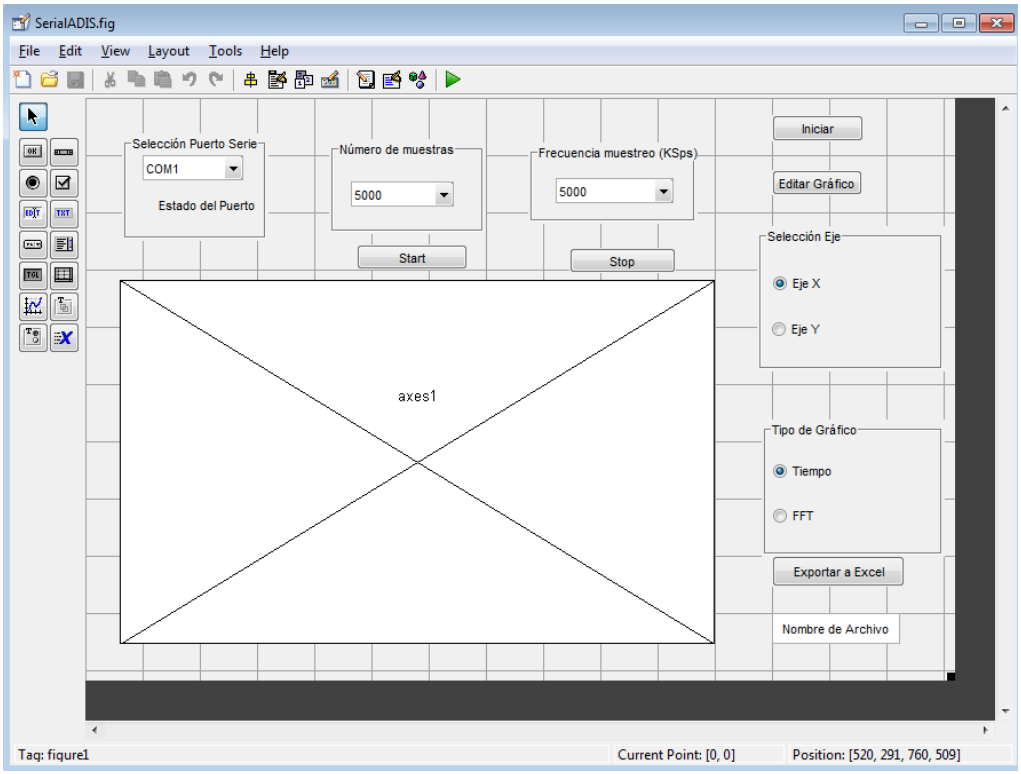


Imagen 52- Edición de .fig MATLAB

Por supuesto, los dos archivos están interrelacionados. Al hacer clic derecho en cualquiera de los elementos gráficos aparecen las distintas funciones asociadas a él.

Al seleccionar una de estas opciones el programa te traslada directamente al fichero *.m* asociado al diseño.

Todos los valores de las propiedades de los elementos (color, valor, posición, string…) y los valores de las variables transitorias del programa se almacenan en una estructura, los cuales son accedidos mediante un único y mismo puntero para todos estos. El puntero a los datos de la aplicación es *handles.* Para salvar los datos de la aplicación se usa la sentencia *guidata*.

Guidata es la función que guarda las variables y propiedades de los elementos en la estructura de datos de la aplicación, por lo tanto, como regla general, en cada subrutina se debe escribir en la última línea lo siguiente:

guidata(hObject,handles)

Esta sentencia garantiza que cualquier cambio o asignación de propiedades o variables quede almacenado.

1.8.6.3. COMPROBACIÓN DE ERRORES

Para asegurar que los datos obtenidos en la comunicación serie sean los correctos, se ha decidido implementar un sistema de comprobación de errores. Este sistema se basa en un *checksum* y funciona de la siguiente manera:

- El PIC suma los datos de la aceleración que se envían y envía la suma (checksum) al final de todos los datos como un entero de 32 bits.
- El PC lee todos los datos y también el checksum que va al final de la trama.
- El PC procesa los datos recibidos para obtener la aceleración y suma todas las aceleraciones.
- El PC comprueba que la suma coincida con el checksum.
    - Si coincide los datos son correctos y se dibuja el gráfico
    - Si no coincide aparece un mensaje de error y se descartan los datos

Este sistema tiene una pequeña desventaja con respecto a otros sistemas de comprobación de errores, que es el hecho de que si los datos son erróneos, estos se pierden. Sin embargo, la realidad es que muy pocas veces se encuentran errores en la transmisión con lo que este método resulta ser muy adecuado para esta aplicación, ya que es rápido y fiable.

1.8.6.4. EXPORTAR DATOS A EXCEL

Matlab permite exportar los datos del espacio de trabajo del programa a una hoja de cálculo de Excel. Para ello, existe la siguiente función

xlswrite(filename,A,sheet)

donde:
filename=nombre fichero

A=matriz con los datos

Sheet=hoja del fichero a la que se quieren exportar los datos

Para tener los datos correctamente colocados en la hoja de cálculo como se quiere, hay que conseguir que la matriz que contiene los datos esté formada por dos columnas: la primera columna indica el número de muestra, la segunda columna corresponde a los datos de la aceleración. El número de filas de la matriz corresponde al número de muestras. Para hacerlo, se han usado algunas de las instrucciones que MATLAB tiene para trabajar con matrices:

```
d=linspace(1,handles.muestras,handles.muestras);%crear vector con el número de muestra
M=[d;aceleracionG];%crea una matriz de dimensión 2x(nºmuestras)con el vector d y la aceleracion
Mt=M'; %calcula la traspuesta de M
```

De esta forma, se obtiene una hoja de cálculo como la que se muestra en la siguiente imagen:

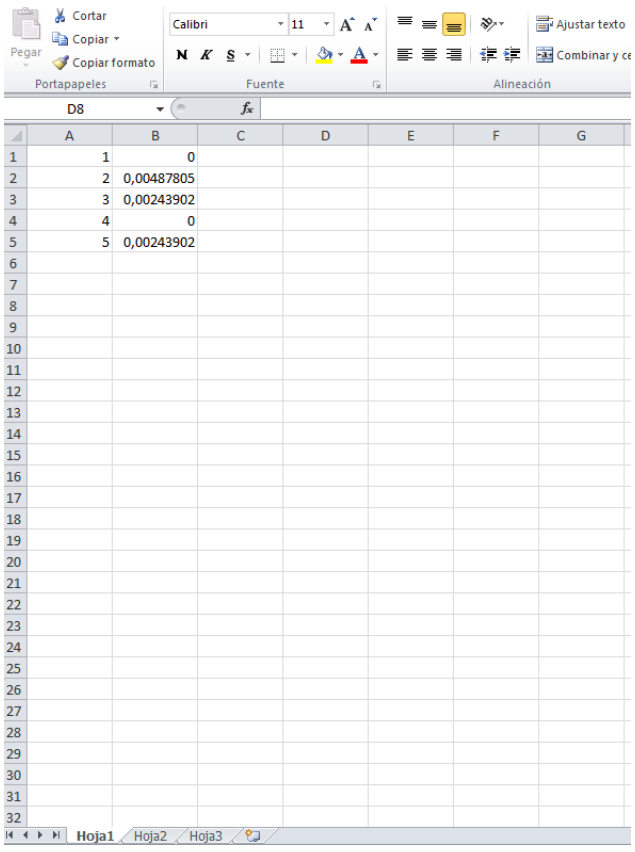


Imagen 53- Datos aceleración en Excel

1.8.6.5. FFT EN MATLAB

En matemáticas, la trasnformada Discreta de Fourier, designada con frecuencia por la abreviatura DFT (Discrete Fourier Trasnform), es una operación ampliamente empleada en el tratamiento de señales y en campos afines para analizar las frecuencias presentes en una señal muestreada. Para el cálculo de la DFT, el método más utilizado es el algoritmo FFT.

Dada una secuencia de N números complejos $X = x_0, x_1, \dots, x_{N-1}$, esta se transforma en una secuencia de N números complejos $F = f_0, f_1, \dots, f_{N-1}$, las cuales son muestras del espectro de la señal analógica, según la fórmula:

$$f_k = \sum_{n=0}^{N-1} x_n e^{-jk\frac{2\pi}{N}n}; \qquad k = 0, \cdots, N-1$$

Ecuación 12- Transformada Discreta de Fourier

Siendo:

- e base de los logaritmos naturales.
- j la unidad imaginaria ($j^2 = -1$)
- N es el número de muestras
- $f_k \in F$

MATLAB permite el cálculo de la FFT para una secuencia de muestras de señal contenidas en un vector. Para ello basta con usar la instrucción:

```
F=fft(X);
```

Sin embargo, es necesario realizar algunos pasos más para obtener una señal de la forma deseada.

Se sabe que el espectro de una señal discreta y periódica es también discreto y periódico. La siguiente imagen ilustra mejor un periodo de dicho espectro:

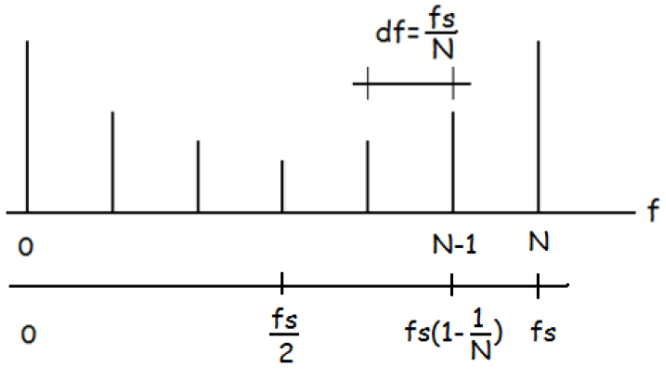


Imagen 54- Periodo de un espectro

De donde se obtienen las siguientes relaciones:

- $f_s = 1/\tau_s$ frecuencia de muestreo
- d_f *resolución frecuencial*
- $f_s/2$ *frecuencia máxima con significado en el espectro*

La **imagen 54** muestra solamente una parte del espectro, el cual tiene la forma que se muestra en la imagen siguiente:

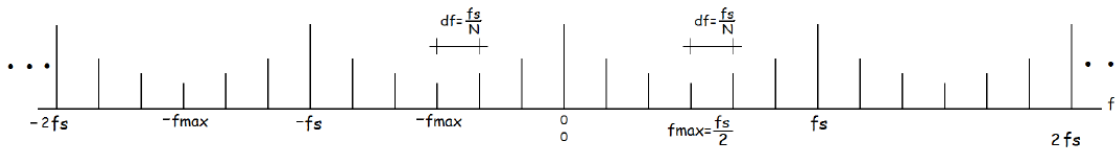


Imagen 55- Forma espectro

Al hacer la FFT en MATLAB lo que se obtiene es todo este espectro. Para obtener un espectro con la forma del que se muestra en la **imagen 56,** existe la siguiente función en MATLAB:

```
F=fftshift(fft(X));
```

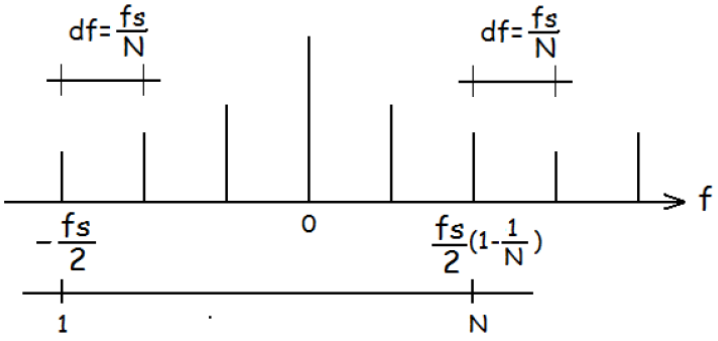


Imagen 56- Espectro tras hacer fftshift

Si se ejecutan los comandos descritos anteriormente y se grafica el vector F, lo que se obtiene poco tiene que ver con la **imagen 56**. Esto se debe a que es necesario graficar un vector de frecuencias (eje X de la imagen) junto con la magnitud de lo obtenido en la FFT (eje X). Este vector se genera según la siguiente ecuación:

$$fHz \quad = \quad -\frac{f_s}{2} : df : \frac{f_s}{2}\left(1 - \frac{1}{N}\right)$$

Ecuación 13- Vector Frecuencias FFT

Siendo:

- f_s $frecuencia\ de\ muestreo$
- $d_f = \frac{f_s}{N}$ $resolución\ frecuencial$
- N $número\ de\ muestras$

Para obtener la verdadera magnitud de la amplitud al realizar la FFT también será necesario multiplicar ésta por un factor de ajuste de $\frac{1}{N}$. En el **anexo 3** se puede ver el código completo.

1.8.6.6. CREACIÓN DE UN EJECUTABLE

MATLAB permite crear un ejecutable de una aplicación creada en él, el cual puede ser ejecutado en cualquier PC sin la necesidad de tener instalado el programa en el mismo y, por tanto, sin comprar la licencia pertinente.

Para crear un ejecutable para Windows, los pasos a seguir se detallan a continuación:

1. Introducir el comando ***deploytool*** en la ventana de comandos de MATLAB

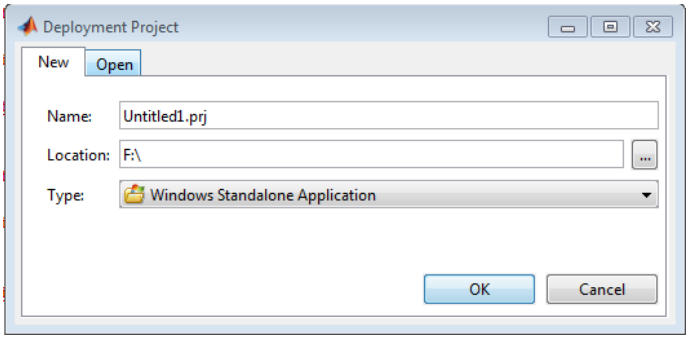


Imagen 57- deployment project MATLAB

Seleccionar el nombre del proyecto y la carpeta donde se guardará. El tipo se deja como aparece en la imagen. Pulsar OK. Aparecerá una ventana como la siguiente en el entorno de trabajo de MATLAB.

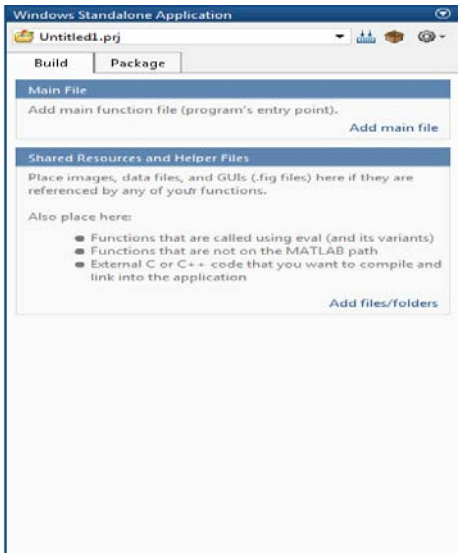


Imagen 58- deploytool MATLAB

2. En la ventana que aparece en el entorno de trabajo de MATLAB, seleccionar la opción ***Add main file*** y elegir el fichero **.m** de la aplicación. Pulsar ***Add files/folders*** y seleccionar el resto de ficheros de la GUI, como el **.fig**.

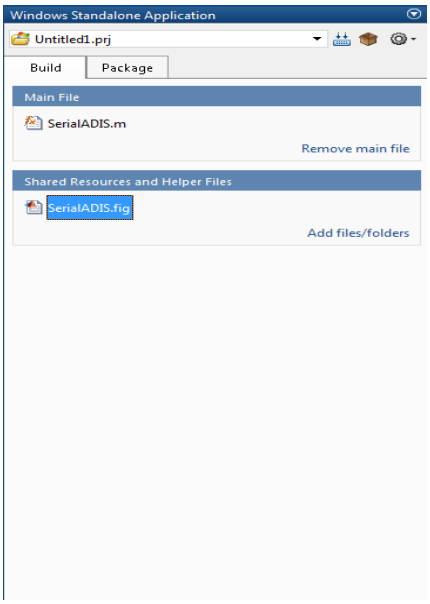


Imagen 59- Build MATLAB

3. Hacer clic en la pestaña ***Package*** pulsar en ***Add MCR y en OK*** cuando se abra una ventana

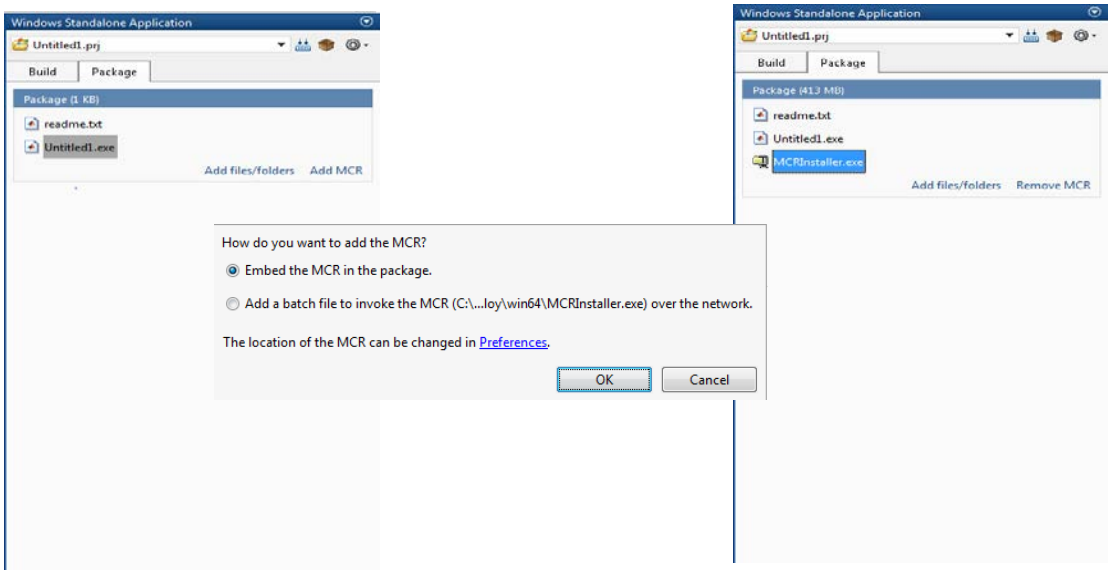


Imagen 60- Selección Ficheros Build MATLAB

4. Pulsar el botón ***Build*** y esperar a que termine el proceso

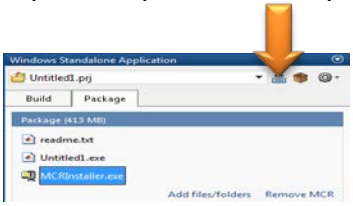


Imagen 61- Botón Build MATLAB

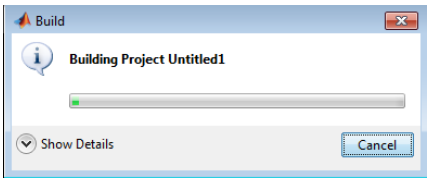


Imagen 62- Building Project MATLAB

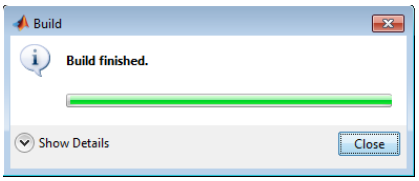


Imagen 63- Build Finished MATLAB

5. Pulsar el botón ***Package*** para empaquetar la aplicación creada y elegir la carpeta donde se desea guardar y pulsar ***OK.***

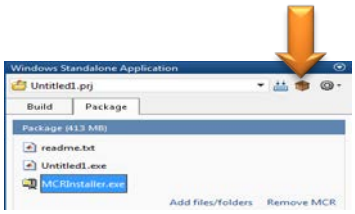


Imagen 64- Botón de empaquetado MATLAB

Esperar a que finalice el proceso.

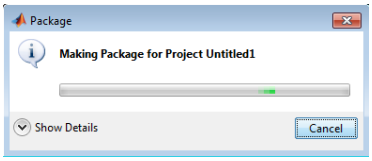


Imagen 65- Proceso de empaquetado MATLAB

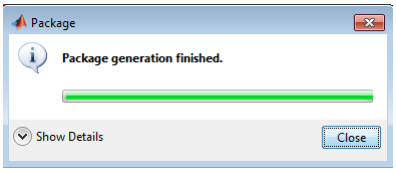


Imagen 66- Final de empaquetamiento MATLAB

Una vez hecho esto ya se habrá creado en la carpeta especificada el fichero **.exe** que es el ejecutable de la aplicación.

1.8.6.7. DIAGRAMA DE FLUJO

Este diagrama de flujo refleja las funciones más importantes del programa en MATLAB. Sólo se incluyen las funciones del botón *Iniciar*, *Detener* y la subrutina llamada mycallback , que es la que se ejecuta cuando se genera el evento del puerto serie explicado anteriormente. El resto de funciones sirven para evaluar los controles restantes de la aplicación. El programa al completo se puede ver en el anexo 3.

Diagrama 12- Iniciar_Callback

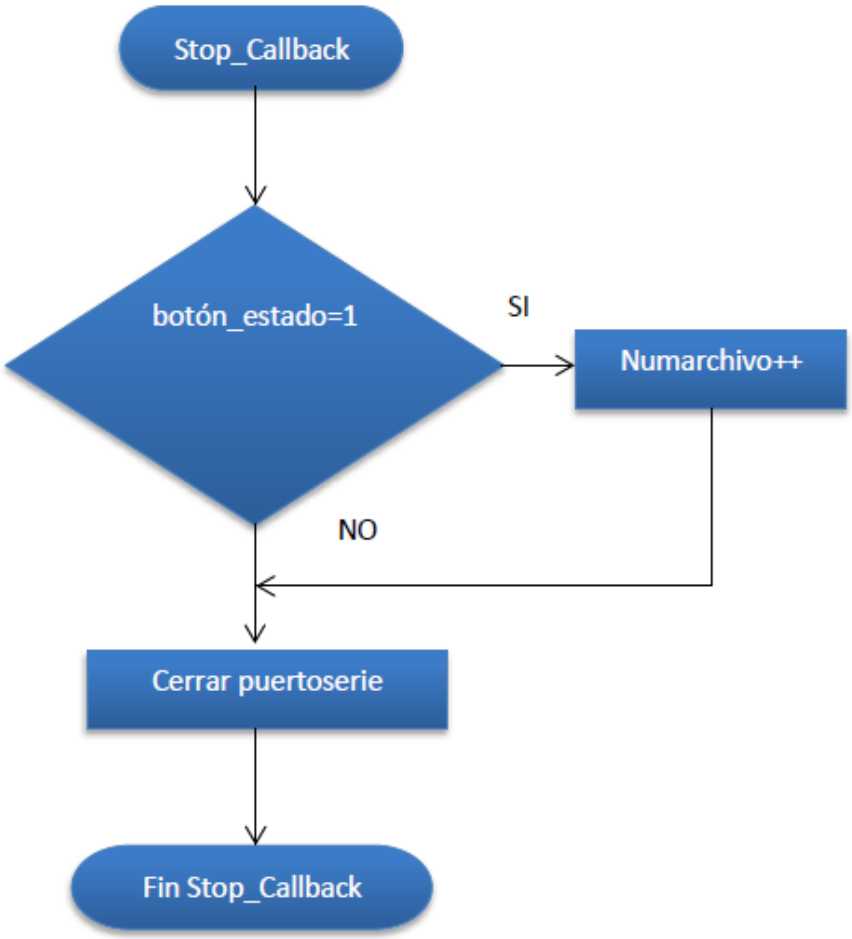
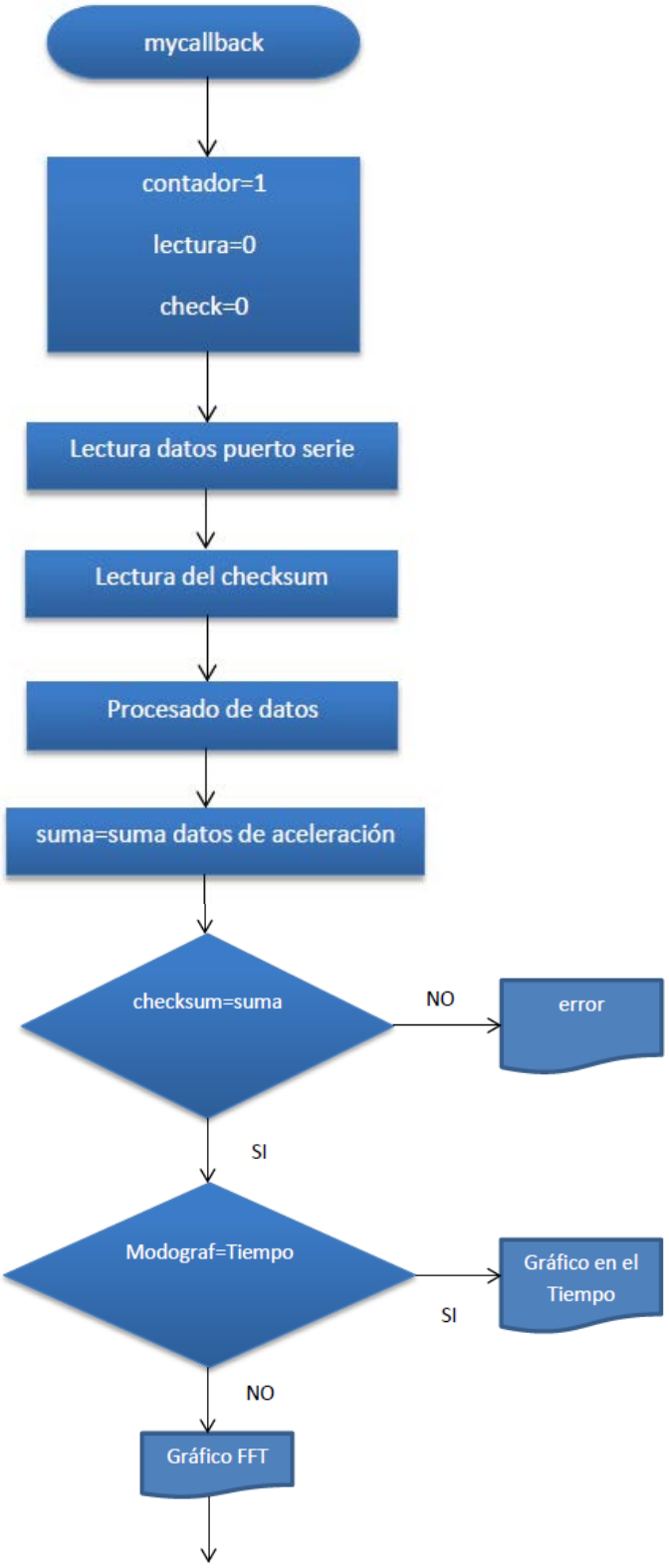


Diagrama 13- Stop_Callback


mycallback
contador=1
lectura=0
check=0
Lectura datos puerto serie
Lectura del checksum
Procesado de datos
suma=suma datos de aceleración
checksum=suma
NO
error
SI
Modograf=Tiempo
Gráfico en el Tiempo
SI
NO
Gráfico FFT

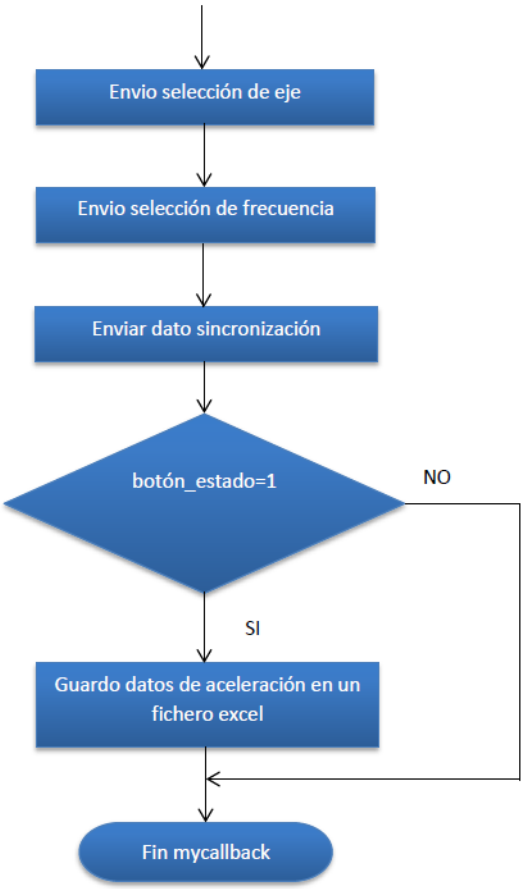


Diagrama 14- mycallback

1.9. BIBLIOGRAFÍA

- Apuntes de Diseño de Tarjetas Electrónicas del año 2013 de la Universidad Pública de Navarra
- Apuntes de Microcontroladores del año 2013 de la Universidad Pública de Navarra
- Páginas web
  - http://www.cypress.com/?id=193
  - http://es.wikipedia.org/wiki/Wikipedia:Portada
  - http://picfernalia.blogspot.com.es/2013/04/comunicaciones-serie-spi.html
  - http://www.dimensionengineering.com/info/accelerometers
  - http://smdelectronicayalgomas.blogspot.com.es/2010/11/entender-especificaciones-clave-del.html#.VDmUuKh_spt
  - http://www.cypress.com/?rID=34870
  - http://lavag.org/topic/11655-data-from-adisusbz/
  - http://lavag.org/topic/11655-data-from-adisusbz/
  - https://ez.analog.com
  - http://www.microchip.com
  - http://picfernalia.blogspot.com.es/2012/06/comunicaciones-puerto-serie-uart.html
  - http://www.mathworks.com/
  - http://5hertz.com/tutoriales/?p=228
  - http://www.mikroe.com/

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

2. MANUAL DE USUARIO

La aplicación *SerialADIS* permite tomar los datos del acelerómetro *ADIS16003* en cualquier PC con Windows. Este documento explica los pasos necesarios para el correcto funcionamiento del sistema. En él se explican los pasos necesarios para la instalación de la aplicación, el correcto conexionado del sistema y el uso de la interfaz de la aplicación.

INSTALACIÓN

Abrir la carpeta ***SerialADIS***. En ella aparecerán los archivos que se pueden ver en la imagen:

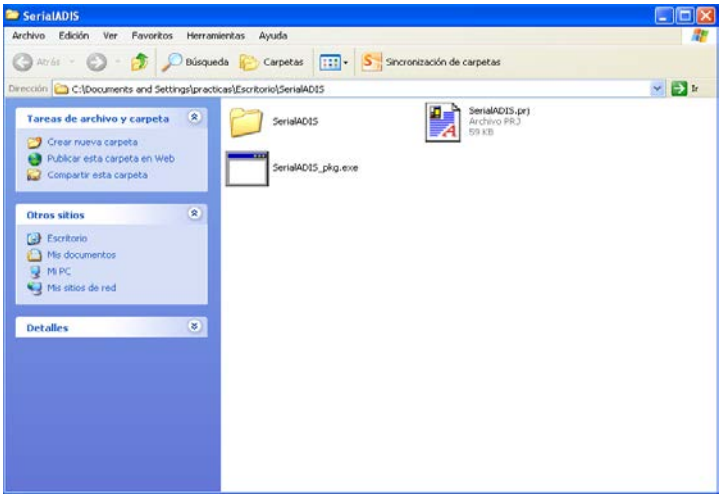


Imagen 67- Archivos carpeta SerialADIS

A continuación, ejecutar (haciendo doble clic sobre él) el archivo ***SerialADIS_pkg.exe***. De esta forma, se extraen los ficheros de instalación. Aparecerá la siguiente ventana:

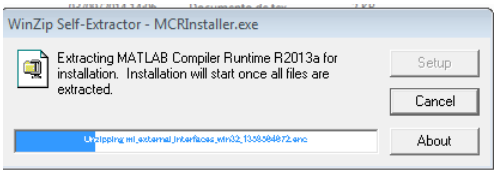


Imagen 68- Ejecutando MCRInstaller

Una vez se haya completado este proceso aparece la siguiente ventana:

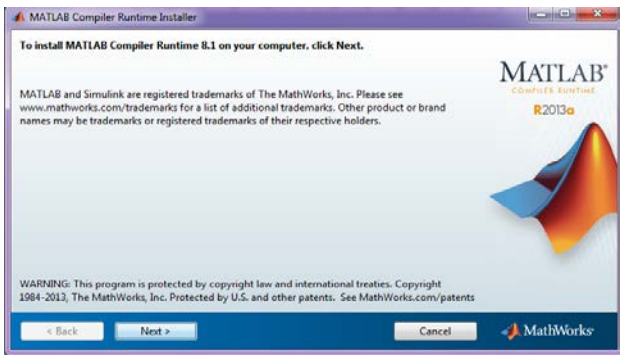


Imagen 69- Instalación paso 1

Hacer clic en *Next*

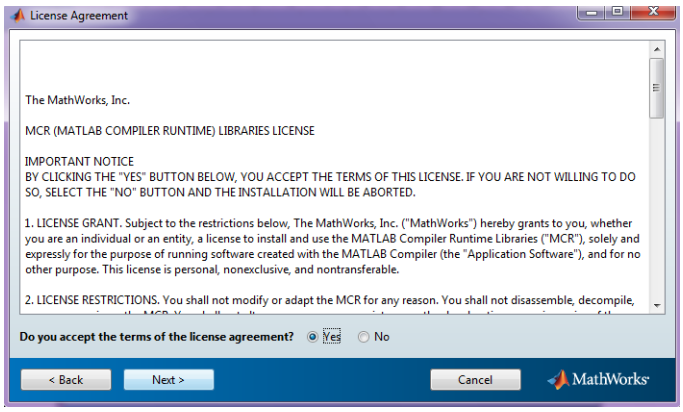


Imagen 70- Instalación paso 2

marcar la casilla *Yes* y clickar *Next*

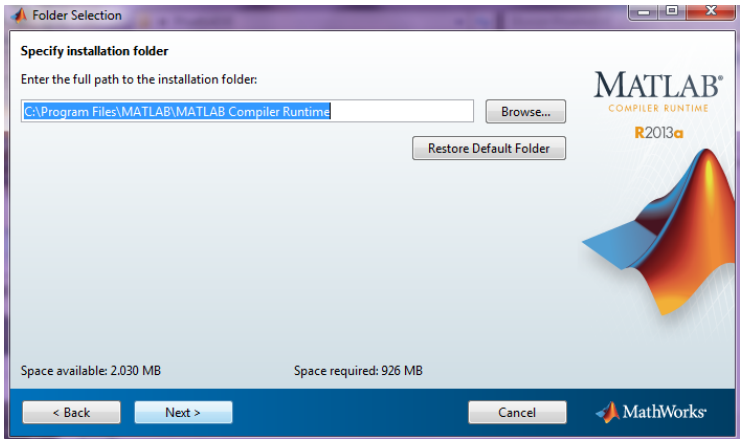


Imagen 71- Instalación paso 3

elegir la carpeta de instalación y hacer clic en *Next*.

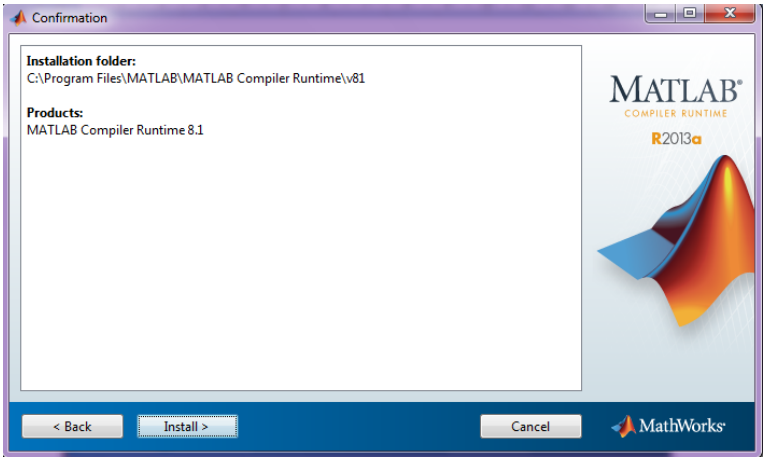


Imagen 72- Instalación paso 4

Hacer clic en ***Install***, lo que dará comienzo al proceso de instalación.

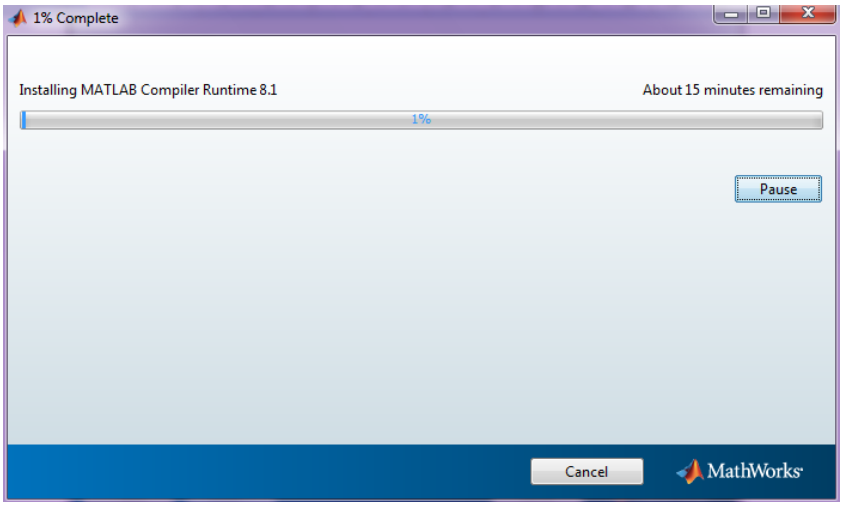


Imagen 73- Instalación 5

Esperar hasta que se complete el proceso y aparezca la ventana siguiente:

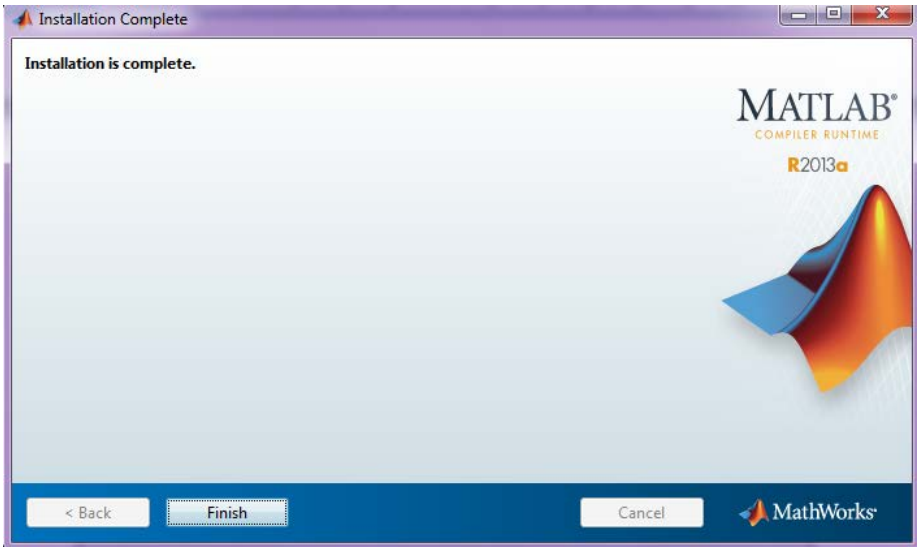


Imagen 74- Instalación 6

Clic en ***Finish***. La instalación se ha completado correctamente. Ya se puede disfrutar de la aplicación ***SerialADIS***.

CONEXIÓN DE DISPOSITIVOS

Para poner en marcha el sistema hay que realizar tres conexiones previas:

1. Comprobar la correcta conexión entre la tarjeta del sensor y la tarjeta principal

Imagen 75- Conexión Acelerómetro

2. Conectar el cable USB al PC

Imagen 76- Conexión USB

3. Conectar el cable serie al PC

Imagen 77- Conexión serie

Nota: este cable debe ser recto y no cruzado.

Una vez realizadas estas conexiones, el sistema de medida de vibraciones ya está listo para funcionar. Se abre la carpeta ***SerialADIS*** donde habrá aparecido un nuevo archivo llamado ***SerialADIS.exe*** que es el ejecutable de nuestra aplicación.

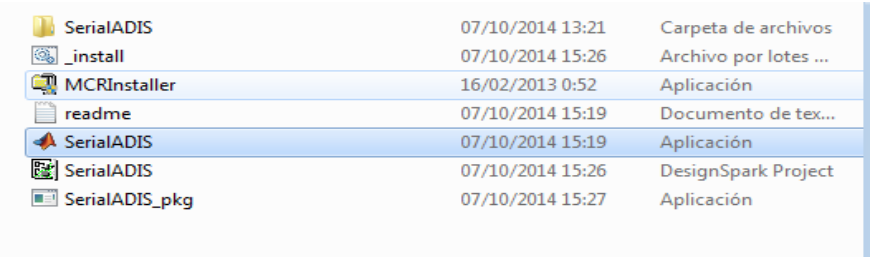


Imagen 78- Conexión serie

Ejecutar el archivo ***SerialADIS.exe*** haciendo doble clic sobre él. Aparecerá la siguiente ventana:

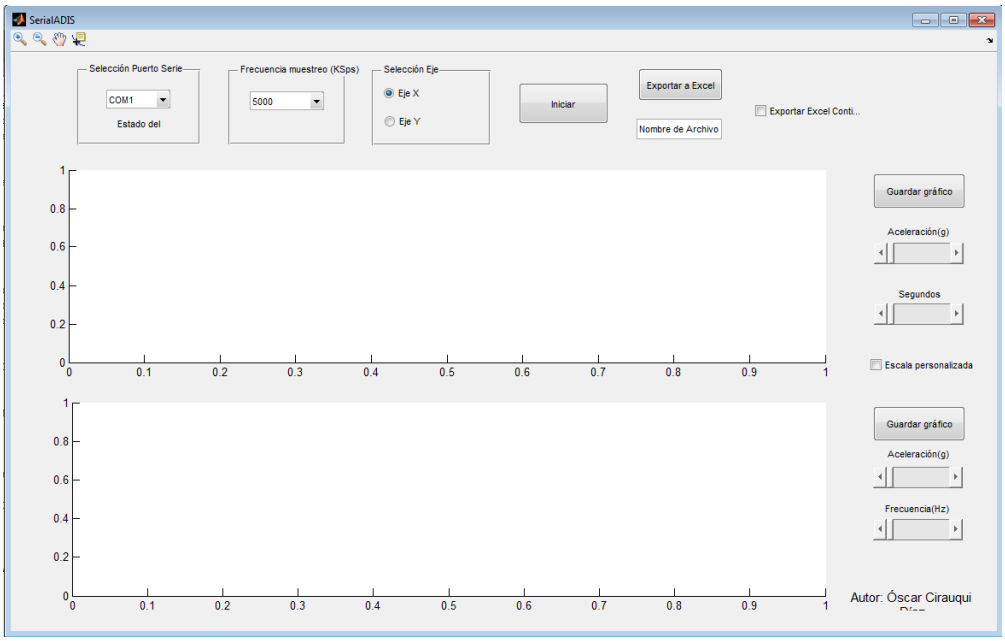


Imagen 79- Interfaz de usuario SerialADIS

Esta ventana es la interfaz de la aplicación. Una vez que se ha llegado aquí ya se puede proceder a la toma de datos.

USO DE LA INTERFAZ DE USUARIO SerialADIS

En primer lugar se debe seleccionar el puerto serie al cuál se ha conectado la tarjeta. (Usualmente COM1). Para ello se utiliza el menú desplegable ***Selección Puerto Serie***.

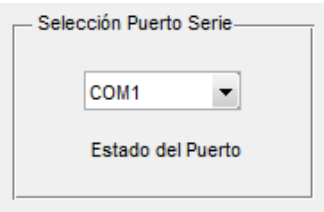


Imagen 80- Selección Puerto Serie

Tras haber seleccionado esto, se seleccionan las distintas opciones de adquisición de datos que nos permite la aplicación:

- Frecuencia de muestreo

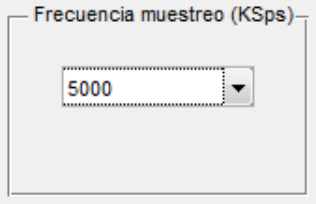


Imagen 81- Selección Frecuencia de muestreo

Permite seleccionar la frecuencia a la cual se desean muestrear los datos. Normalmente, interesará muestrear a 5000KSps para trabajar con todo el ancho de banda del sensor. Sin embargo, si se desea adquirir señales de frecuencias muy bajas, convendrá elegir una frecuencia de muestreo menor.
En todo caso, hay que tener en cuenta que siempre que la frecuencia de muestreo debe ser al menos el doble de la señal que se quiere adquirir.

- Selección del Eje

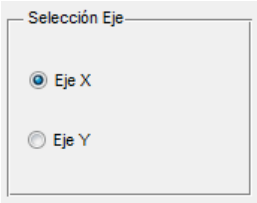


Imagen 82- Selección de eje

Esta opción permite seleccionar entre los dos ejes del acelerómetro.

Una vez elegidas estas opciones se procede a la toma de datos. Para ello, se pulsa el botón ***Iniciar***.

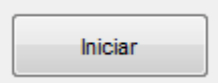


Imagen 84- Botón Iniciar

Transcurridos unos pocos segundos deberá aparecer una señal en el área del gráfico, tal y como se muestra en la imagen siguiente:

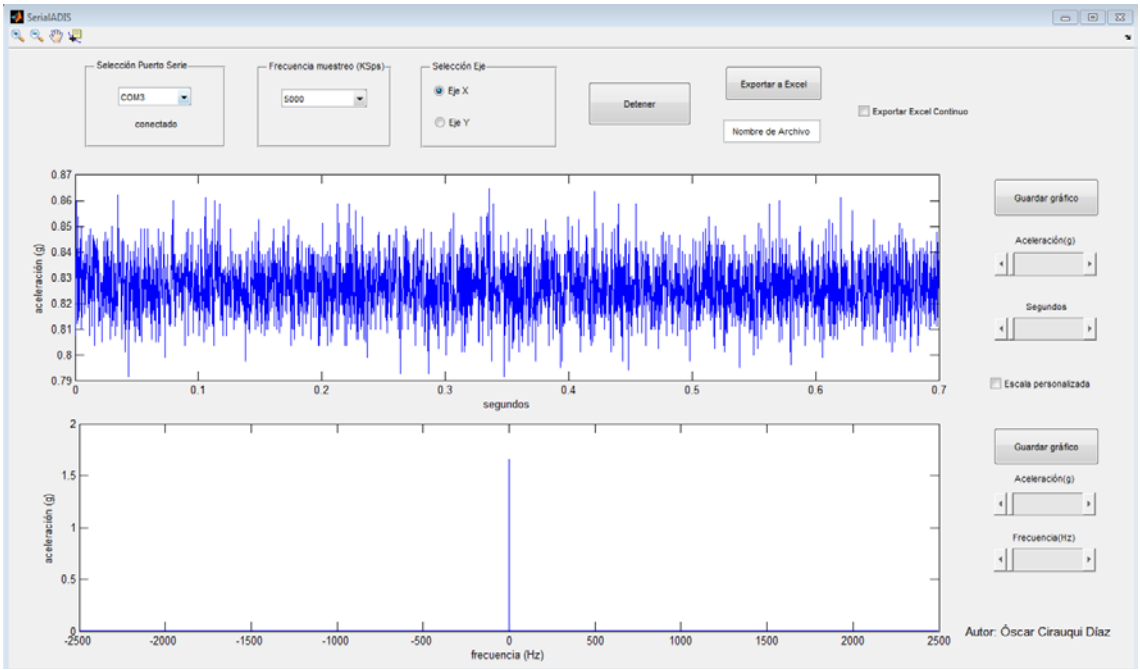


Imagen 85- Toma de datos

Después de haber obtenido los datos, existen varias opciones que permitirán trabajar con ellos:

- Guardar Gráfico

Tras haber pulsado el botón ***Detener***, al pulsar este botón aparece una nueva ventana con el gráfico actual. Esta ventana contiene distintas opciones para poder editar el gráfico como modificar los ejes, ampliar distintas áreas del gráfico, muestra de variables estadísticas etc. Además de esto también permite guardar el gráfico actual en distintos formatos de imagen.

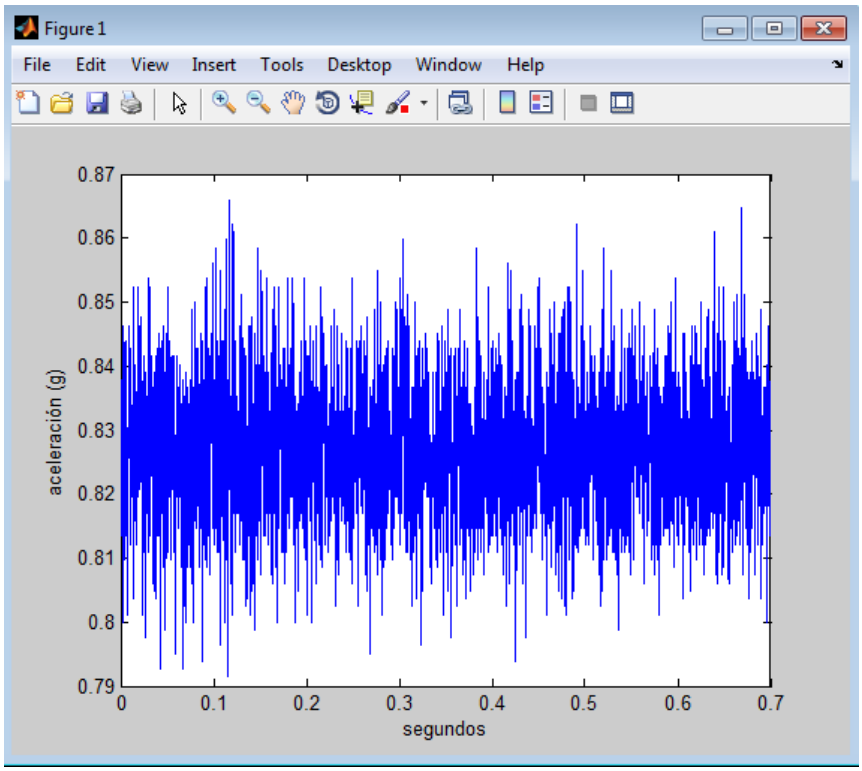


Imagen 86- Editar gráfico

- Exportar Excel Continuo

Si se marca esta casilla se crea un Excel con los datos de la sesión actual. Se creará un fichero en la carpeta *SerialADIS* con los datos de aceleración de la sesión. El fichero queda nombrado como *Vibracion_1.xls*. Dentro de este fichero se crean distintas hojas, en cada una de las cuales aparecen 35000 datos. De esta forma, se cubre la duración de un viaje en ascensor. Si con esta casilla marcada se pulsa el botón *Detener* y luego se vuelve a pulsar *Iniciar* lo que ocurre es que se crea una nueva hoja de Excel (*Vibracion_2.xls*) con los nuevos datos.

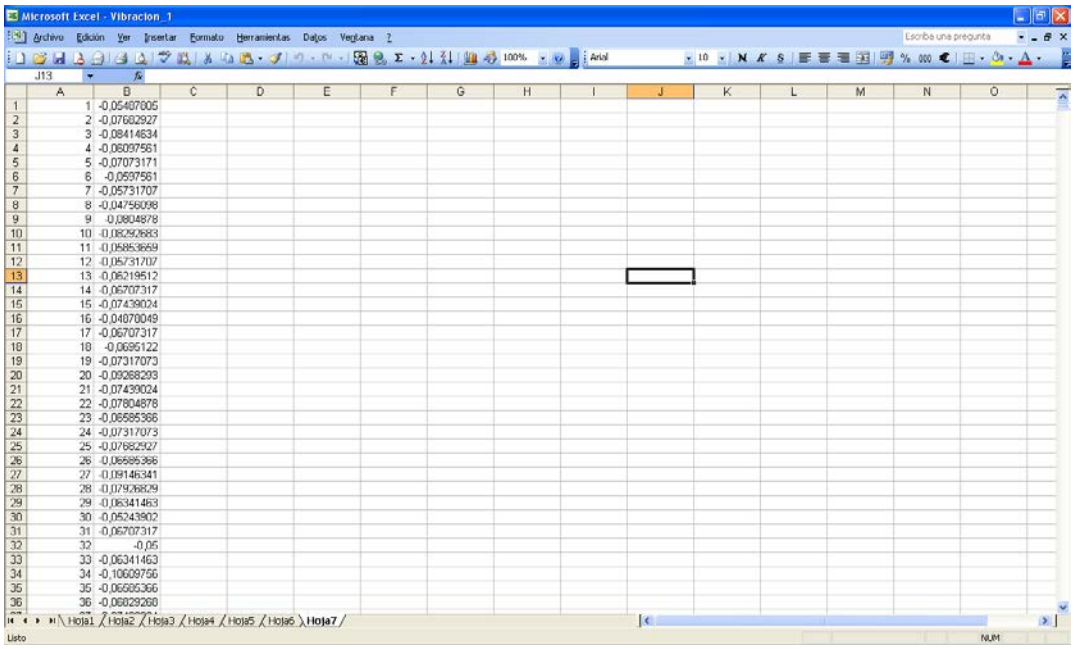


Imagen 87- Captura Excel

- Botón Exportar a Excel

Este botón permite la opción de exportar a Excel los datos del gráfico actual con el nombre que se desee. Para hacerlo se debe introducir el nombre en el recuadro situado bajo el botón y a continuación pulsar el botón. La hoja de cálculo creada aparecerá en la carpeta *SerialADIS.*

- Botón Detener

Este botón permite detener la captura de datos de la aplicación, de forma que se quedará en pantalla la última captura realizada.

- Personalización de escalas

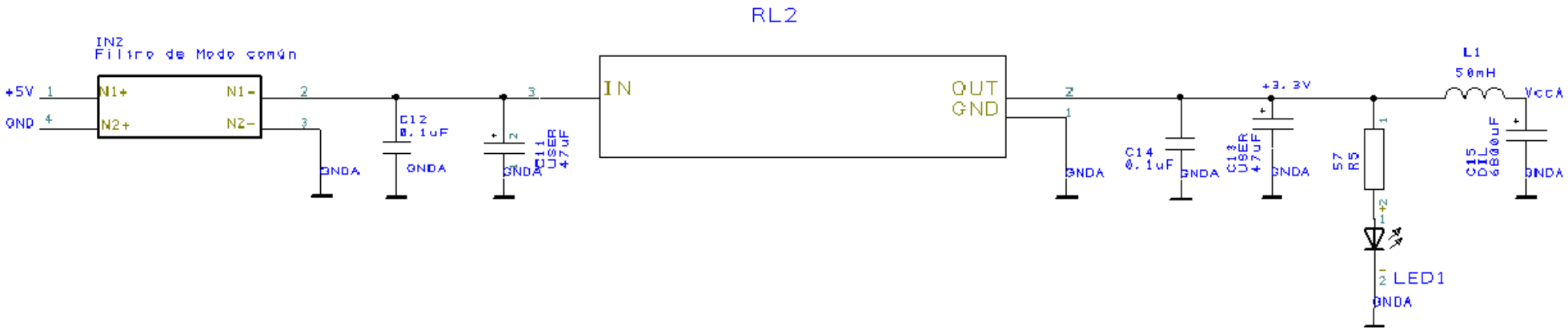
Al marcar la casilla ***Escala personalizada*** es posible ampliar la escala del gráfico de ambos ejes desplazando los sliders situados junto a cada uno de los gráficos. Si esta casilla se encuentra desmarcada la escala se ajusta de forma automática.

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

3. PLANOS

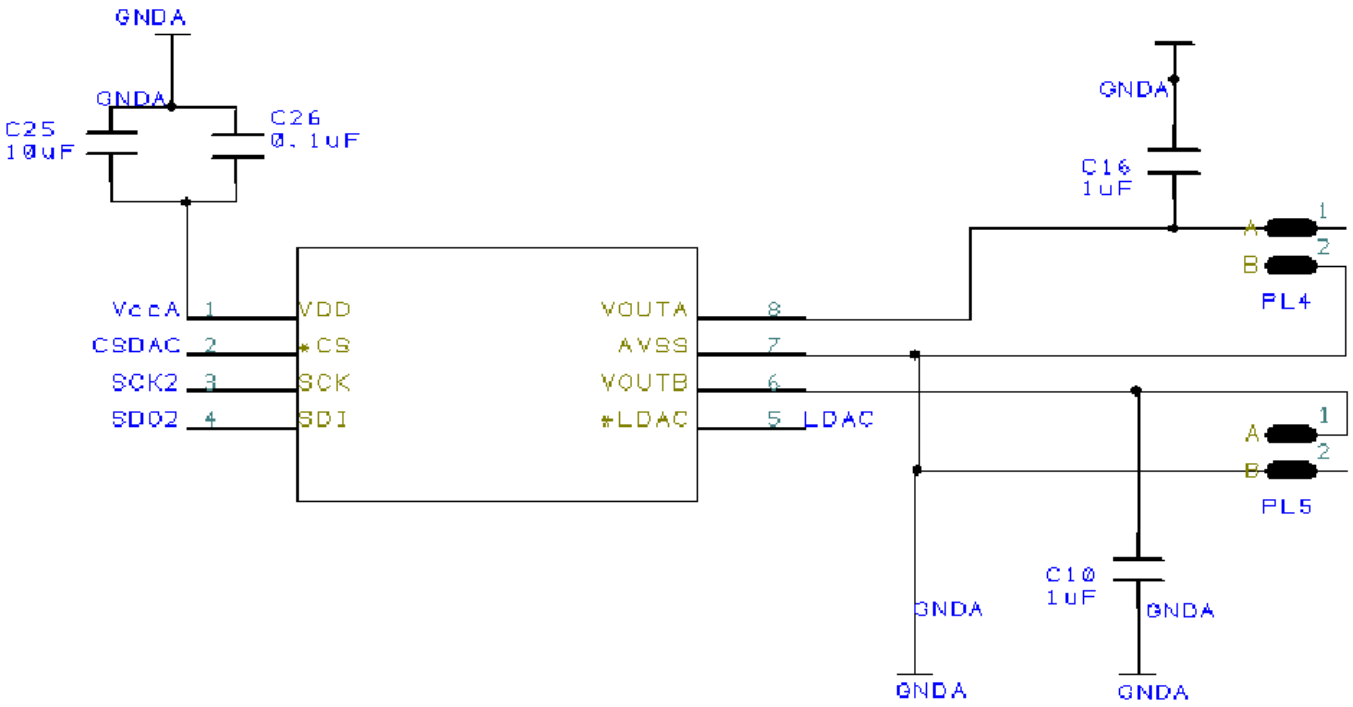
INDICE DE PLANOS

Alimentación



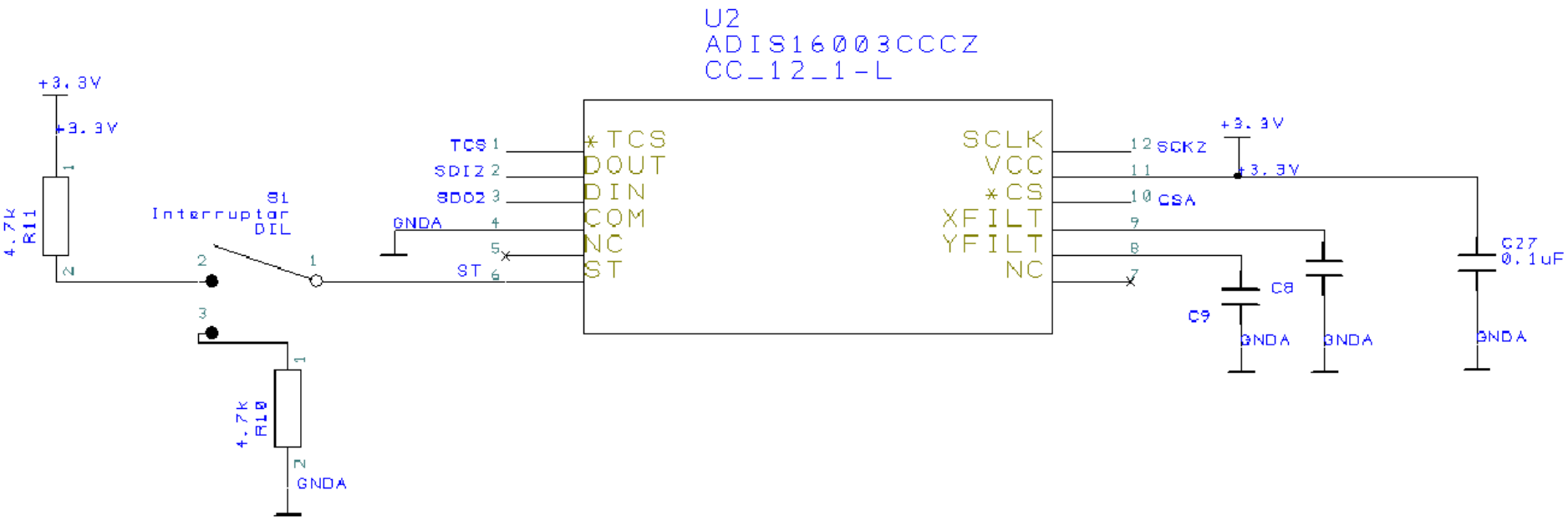
UNIVERSIDAD PÚBLICA DE NAVARRA *NAFARROAKO UNIVERSITATE PUBLIKOA*	DEPARTAMENTO: ING.ELECTRÓNICA	
PROYECTO: MEDIDA DE VIBRACIONES BASADO EN MEMS	REALIZADO: Óscar Cirauqui Díaz	
	FIRMA:	
PLANO: ALIMENTACIÓN	FECHA: 1/09/2014	ESCALA: / Nº 1

CONVERTIDOR DIGITAL-ANALÓGICO



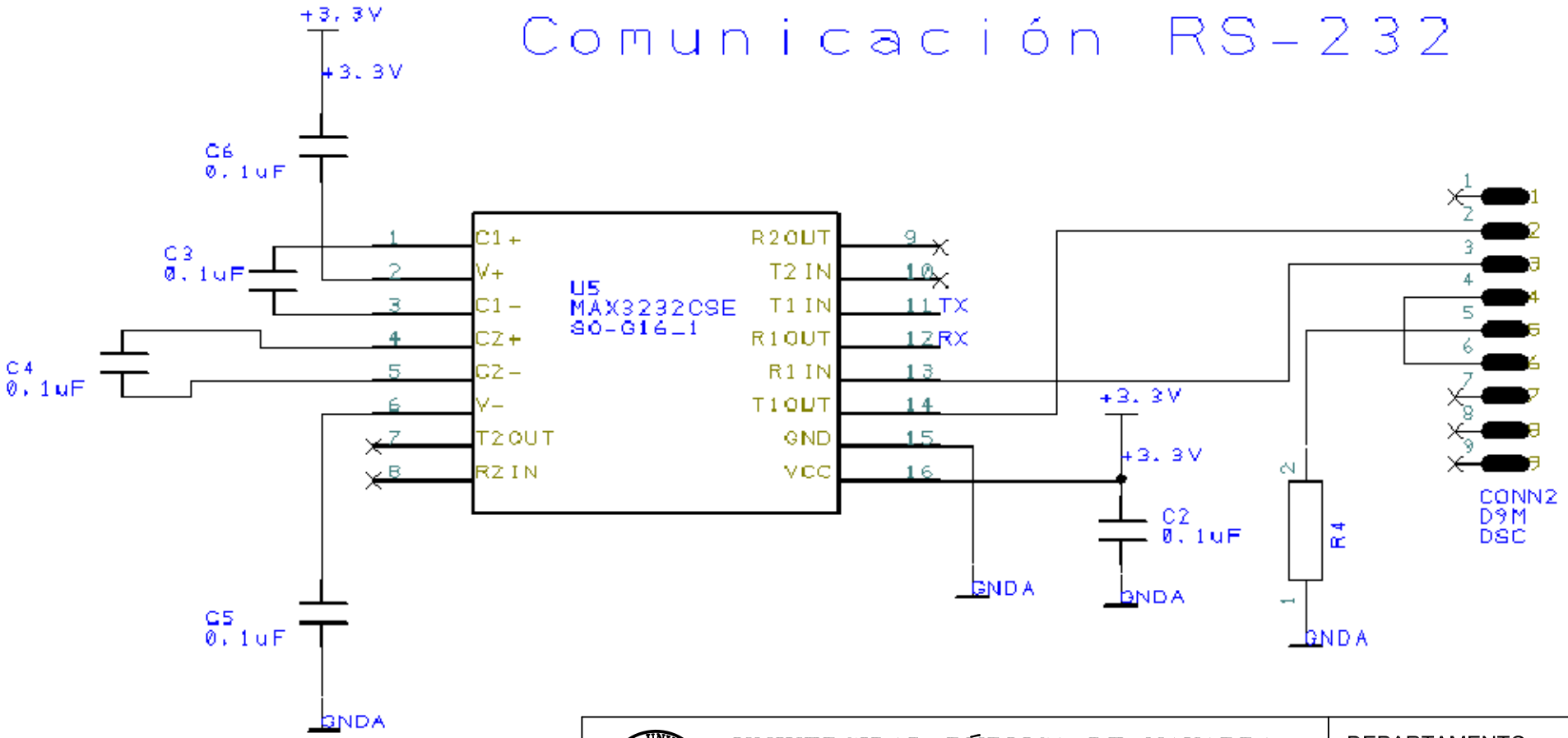
UNIVERSIDAD PÚBLICA DE NAVARRA *NAFARROAKO UNIVERSITATE PUBLIKOA*			DEPARTAMENTO: ING.ELECTRÓNICA
PROYECTO: MEDIDA DE VIBRACIONES BASADO EN MEMS			REALIZADO: Óscar Cirauqui Díaz
			FIRMA:
PLANO: CONVERTIDOR DIGITAL-ANALÓGICO	FECHA: 1/09/2014	ESCALA:	Nº 2

Acelerómetro

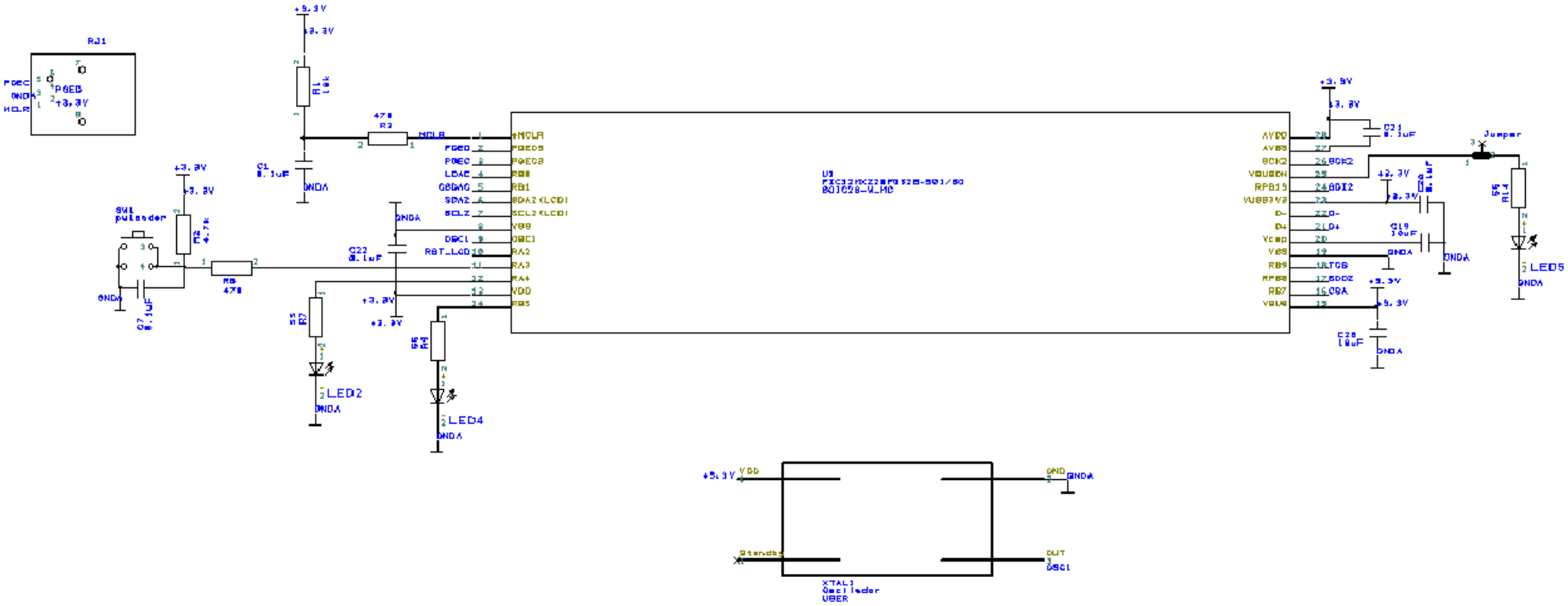


UNIVERSIDAD PÚBLICA DE NAVARRA *NAFARROAKO UNIVERSITATE PUBLIKOA*			DEPARTAMENTO: ING.ELECTRÓNICA
PROYECTO: MEDIDA DE VIBRACIONES BASADO EN MEMS			REALIZADO: Óscar Cirauqui Díaz
			FIRMA:
PLANO: ACELERÓMETRO ADIS16003	FECHA: 1/09/2014	ESCALA:	Nº 3

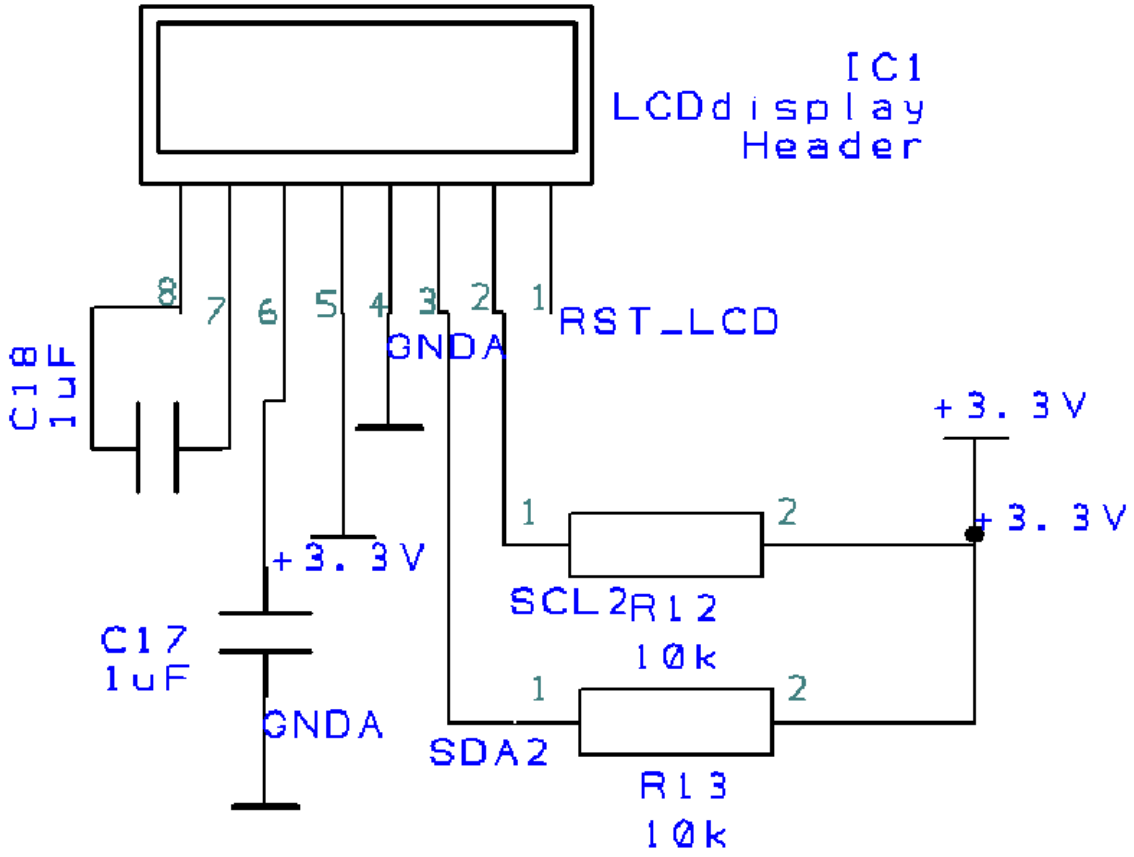
Comunicación RS-232



UNIVERSIDAD PÚBLICA DE NAVARRA *NAFARROAKO UNIVERSITATE PUBLIKOA*	DEPARTAMENTO: ING.ELECTRÓNICA	
PROYECTO: MEDIDA DE VIBRACIONES BASADO EN MEMS	REALIZADO: Óscar Cirauqui Díaz	
	FIRMA:	
PLANO: COMUNICACIÓN RS-232	FECHA: 1/09/2014	ESCALA: / Nº 4

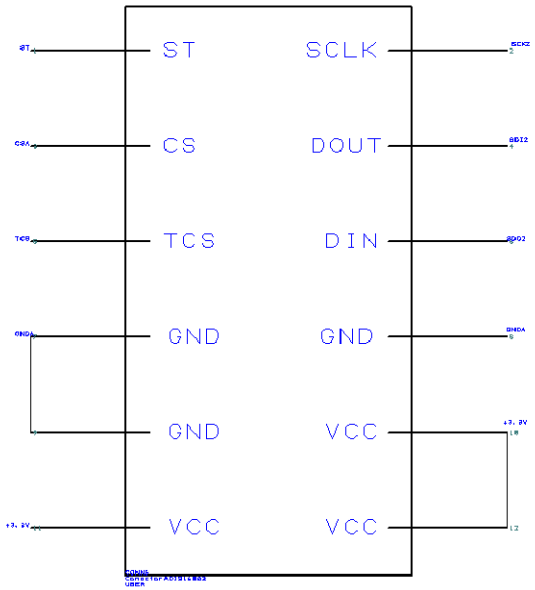


UNIVERSIDAD PÚBLICA DE NAVARRA *NAFARROAKO UNIVERSITATE PUBLIKOA*	DEPARTAMENTO: ING.ELECTRÓNICA		
PROYECTO: MEDIDA DE VIBRACIONES BASADO EN MEMS	REALIZADO: Óscar Cirauqui Díaz		
	FIRMA:		
PLANO: MICROCONTROLADOR PIC	FECHA: 1/09/2014	ESCALA:	Nº 5



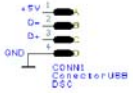
UNIVERSIDAD PÚBLICA DE NAVARRA *NAFARROAKO UNIVERSITATE PUBLIKOA*	DEPARTAMENTO: ING.ELECTRÓNICA		
PROYECTO: MEDIDA DE VIBRACIONES BASADO EN MEMS	REALIZADO: Óscar Cirauqui Díaz		
	FIRMA:		
PLANO: PANTALLA LCD	FECHA: 1/09/2014	ESCALA:	Nº 6

CONECTOR ADIS16003

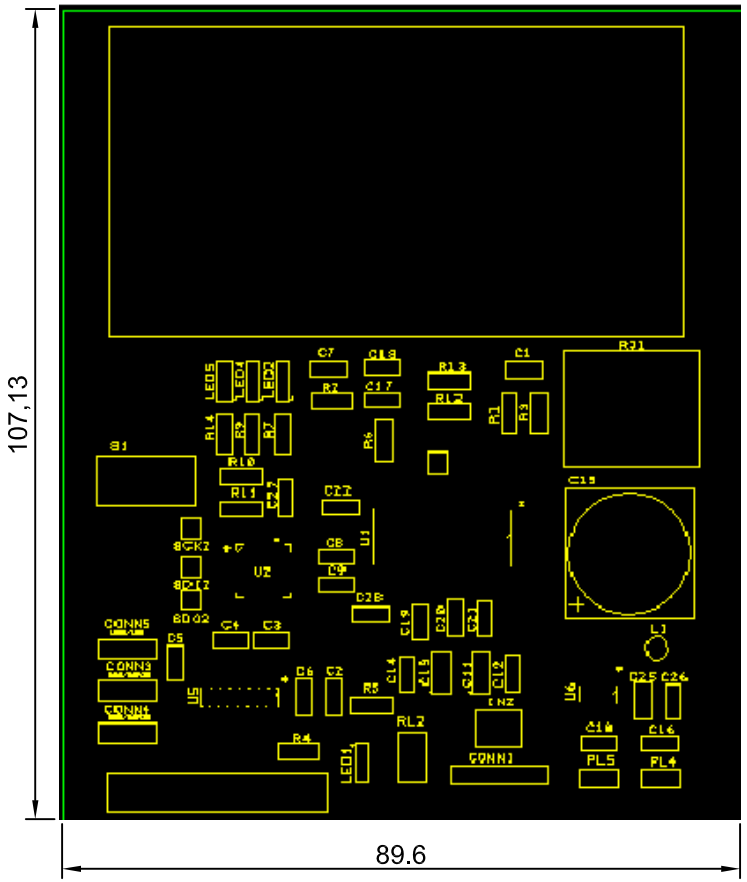


Jumper seleccion USB/RS-232

Conexión USB

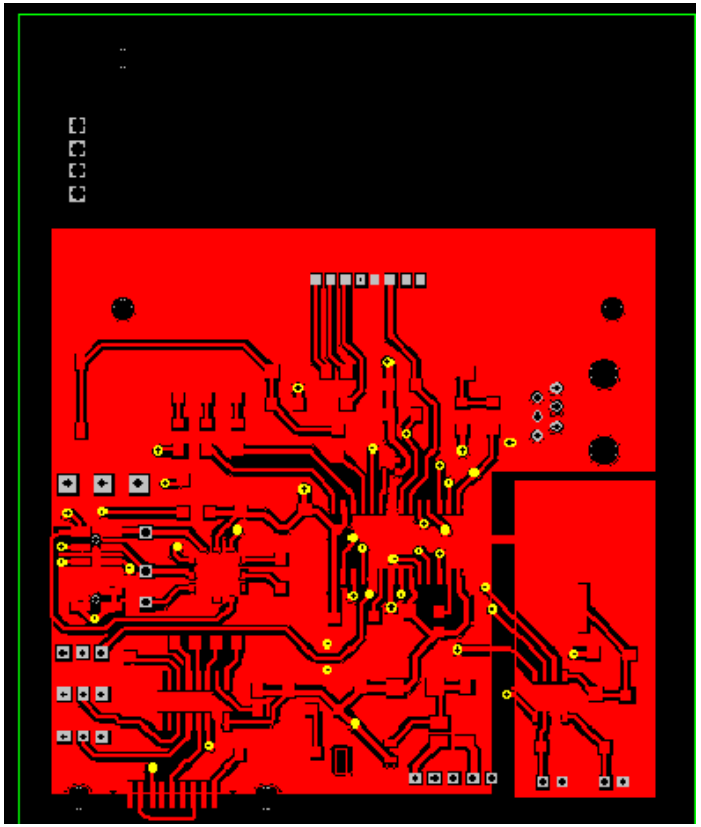


UNIVERSIDAD PÚBLICA DE NAVARRA *NAFARROAKO UNIVERSITATE PUBLIKOA*			DEPARTAMENTO: ING.ELECTRÓNICA
PROYECTO: MEDIDA DE VIBRACIONES BASADO EN MEMS			REALIZADO: Óscar Cirauqui Díaz
			FIRMA:
PLANO: CONEXIONES	FECHA: 1/09/2014	ESCALA:	Nº 7

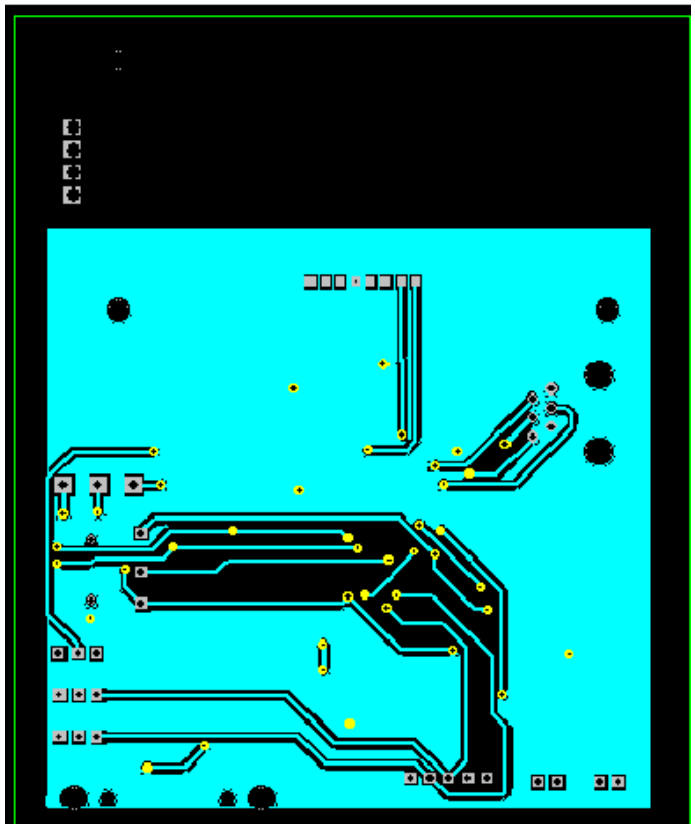


UNIVERSIDAD PÚBLICA DE NAVARRA *NAFARROAKO UNIVERSITATE PUBLIKOA*			DEPARTAMENTO: ING.ELECTRÓNICA
PROYECTO: MEDIDA DE VIBRACIONES BASADO EN MEMS			REALIZADO: Óscar Cirauqui Díaz
			FIRMA:
PLANO: DISTRIBUCIÓN COMPONENTES TARJETA_PIC PCB	FECHA: 1/09/2014	ESCALA: 1/1	Nº 8

CARA SUPERIOR

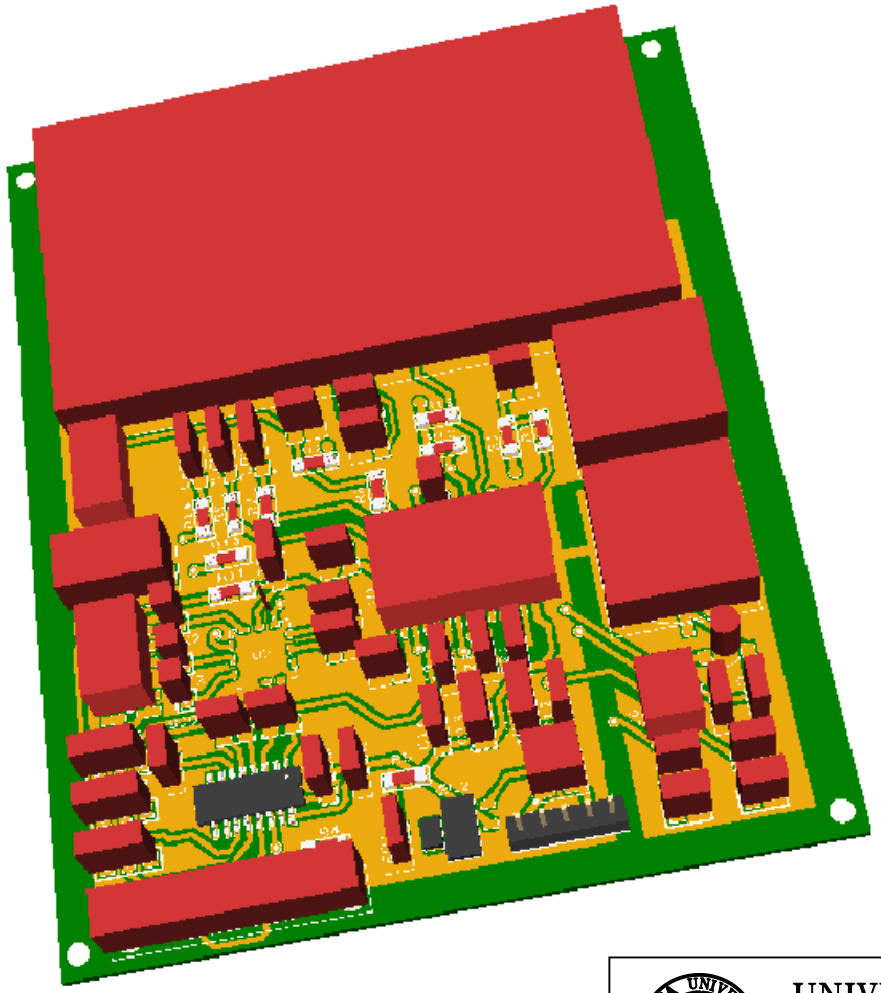


CARA INFERIOR

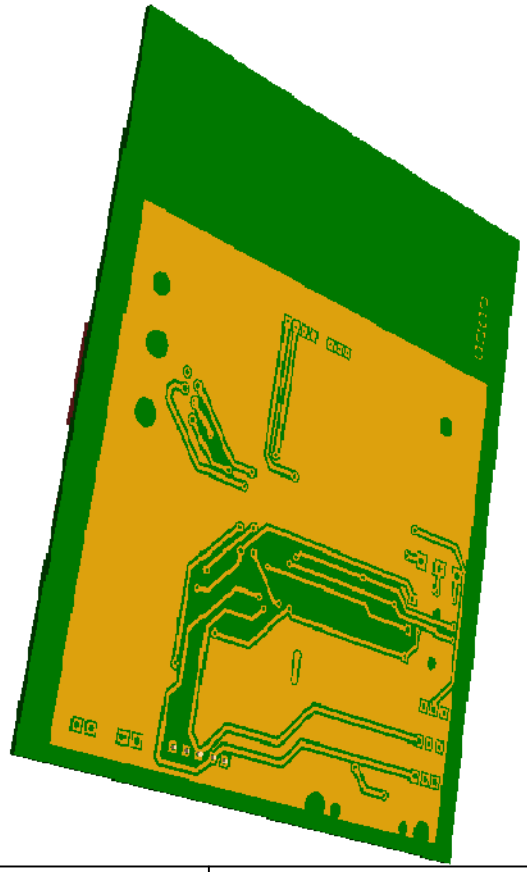


UNIVERSIDAD PÚBLICA DE NAVARRA *NAFARROAKO UNIVERSITATE PUBLIKOA*	DEPARTAMENTO: ING.ELECTRÓNICA		
PROYECTO: MEDIDA DE VIBRACIONES BASADO EN MEMS	REALIZADO: Óscar Cirauqui Díaz		
	FIRMA:		
PLANO: TRAZADO DE PISTAS TARJETA_PIC PCB	FECHA: 1/09/2014	ESCALA: 1/1	Nº 9

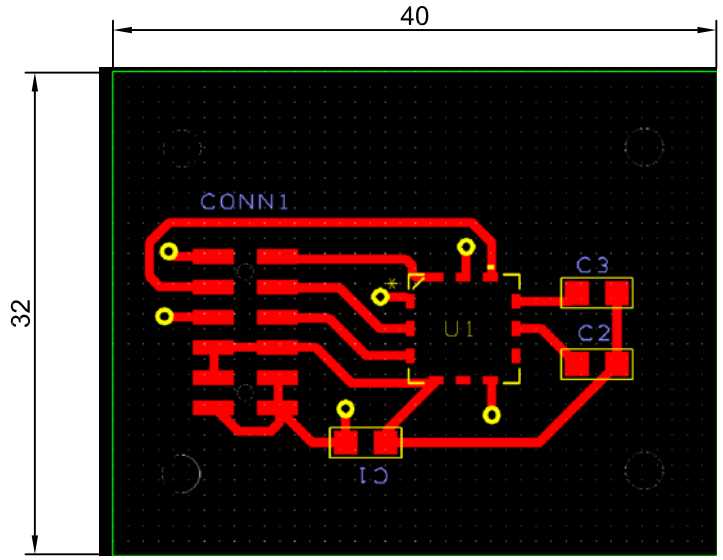
CARA SUPERIOR



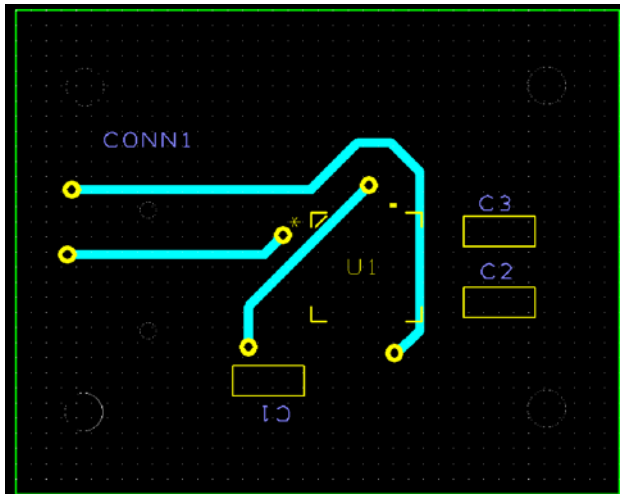
CARA INFERIOR



UNIVERSIDAD PÚBLICA DE NAVARRA *NAFARROAKO UNIVERSITATE PUBLIKOA*			DEPARTAMENTO: ING.ELECTRÓNICA
PROYECTO: MEDIDA DE VIBRACIONES BASADO EN MEMS			REALIZADO: Óscar Cirauqui Díaz
			FIRMA:
PLANO: VISTAS 3D TARJETA_PIC PCB	FECHA: 1/09/2014	ESCALA: 1/1	Nº 10

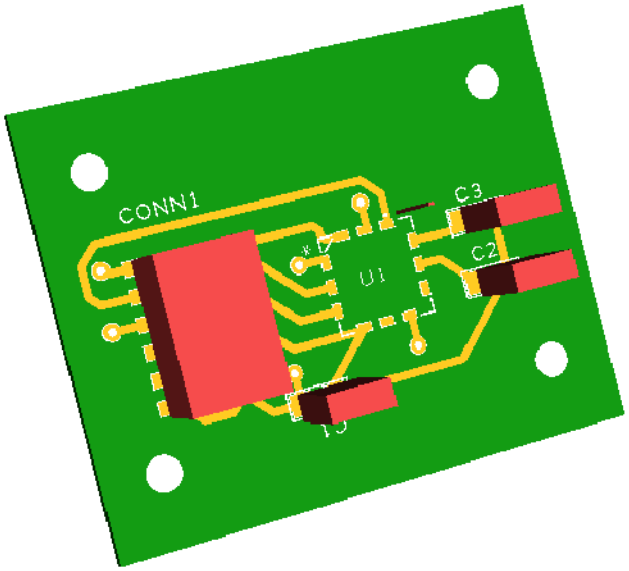


CARA SUPERIOR

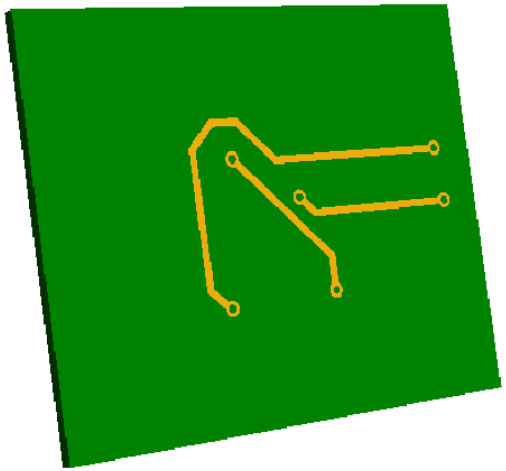


CARA INFERIOR

UNIVERSIDAD PÚBLICA DE NAVARRA *NAFARROAKO UNIVERSITATE PUBLIKOA*			DEPARTAMENTO: ING.ELECTRÓNICA
PROYECTO: MEDIDA DE VIBRACIONES BASADO EN MEMS			REALIZADO: Óscar Cirauqui Díaz
			FIRMA:
PLANO: TRAZADO DE PISTAS TARJETA_ADIS PCB	FECHA: 1/09/2014	ESCALA: 2/1	Nº 11

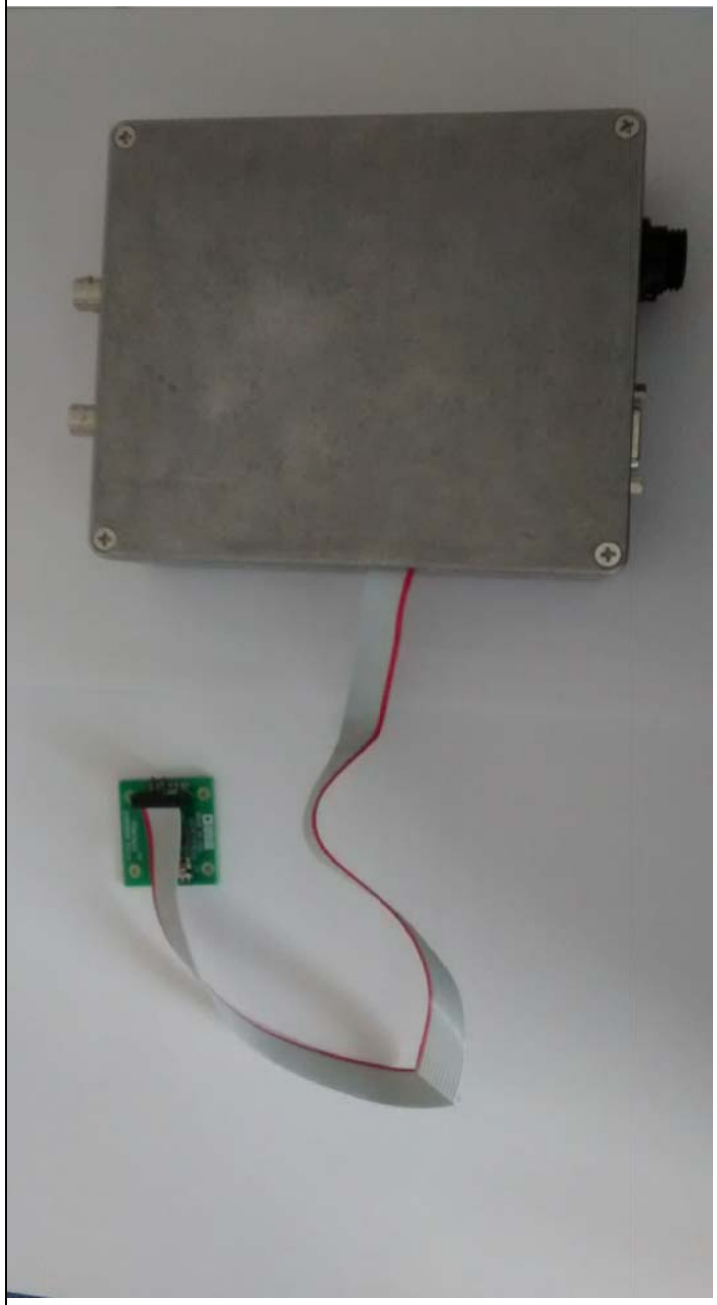


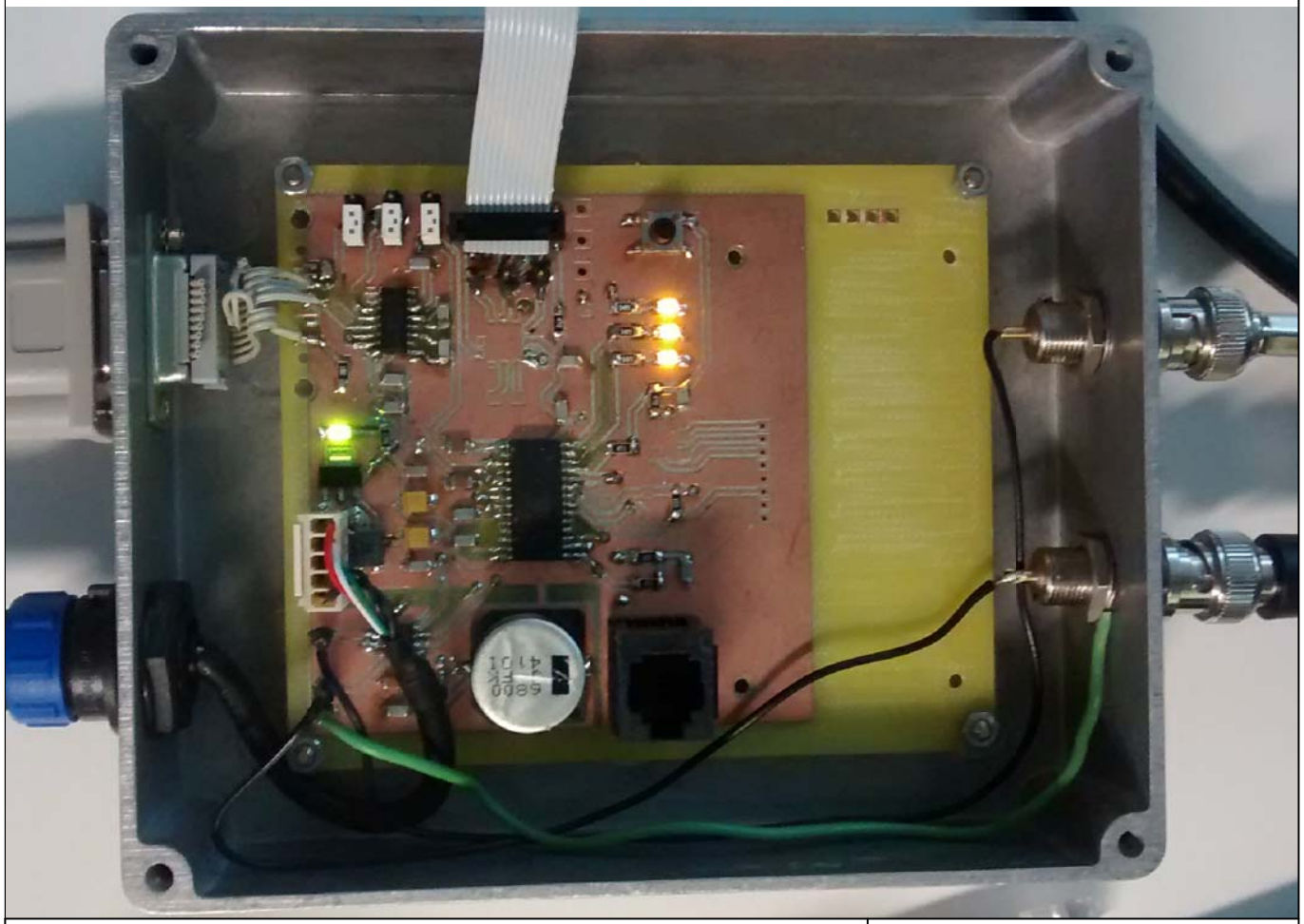
CARA SUPERIOR



CARA INFERIOR

UNIVERSIDAD PÚBLICA DE NAVARRA *NAFARROAKO UNIVERSITATE PUBLIKOA*	DEPARTAMENTO: ING.ELECTRÓNICA		
PROYECTO: MEDIDA DE VIBRACIONES BASADO EN MEMS	REALIZADO: Óscar Cirauqui Díaz		
	FIRMA:		
PLANO: VISTAS 3D TARJETA_ADIS PCB	FECHA: 1/09/2014	ESCALA: 2/1	Nº 12





UNIVERSIDAD PÚBLICA DE NAVARRA *NAFARROAKO UNIVERSITATE PUBLIKOA*	DEPARTAMENTO: ING.ELECTRÓNICA		
PROYECTO: MEDIDA DE VIBRACIONES BASADO EN MEMS	REALIZADO: Óscar Cirauqui Díaz		
	FIRMA:		
PLANO: MONTAJE FINAL	FECHA: 12/10/2014	ESCALA:	Nº 13

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

4. PRESUPUESTO

El presente presupuesto incluye la completa fabricación de una unidad de cada una de las dos tarjetas descritas en la memoria: la tarjetaPIC y la tarjetaADIS. No se incluye el precio del trazado de pistas de la tarjeta, ya que este se ha realizado con el equipo de la universidad. Sí que se ha estimado el precio de mano de obra de montaje y diseño.

Se ha incluido en rojo el precio de la pantalla, aunque este precio no ha sido en cuenta al realizar la suma total del presupuesto, ya que en el diseño final la pantalla se ha descartado.

MATERIALES				
CIRCUITOS INTEGRADOS				
Componente	**Nombre**	**Cantidad**	**Precio unitario(€)**	**Total(€)**
Acelerómetro	ADIS16003	1	26,78	26,78
Microcontrolador	PIC32MX220F032B-50I/SO	1	2,61	2,61
DAC	DAC_MCP4822-E/MS	1	2,53	2,53
Regulador 3,3V	ADP3338AKCZ-3.3RL7	1	2,22	2,22
Adaptador tensión RS-232	MAX3232CD	1	1,87	1,87
	SUBTOTAL			36,01
GENERADORES DE RELOJ				
Componente	**Nombre**	**Cantidad**	**Precio unitario(€)**	**Total(€)**
Oscilador	ASDMB-12.000MHZ-XY-T	1	2,33	2,33
	SUBTOTAL			2,33
INDICADORES LUMINOSOS				
Componente	**Nombre**	**Cantidad**	**Precio unitario(€)**	**Total(€)**
LED Verde	LG N971-KN-1	1	0,056	0,056
LED Amarillo	LY N971-HL-1	3	0,046	0,138
Pantalla LCD	NHD-C0220BiZ-FS(RGB)-FBW-3VM	1	9,6	9,6
	SUBTOTAL			0,194
CONDENSADORES				
Componente	**Nombre**	**Cantidad**	**Precio unitario(€)**	**Total(€)**
0,1uF cerámico	GRM31C5C1E104JA01L	14	0,161	2,254
1uF cerámico	GRM31CR71H105KA61L	4	0,114	0,456
10uF cerámico	GRM31CR71A106KA01L	3	0,186	0,558
47uF electrolítico	T491B476K010AT	2	0,651	1,302
6800uF electrolítico	EEV-FK0J682M	1	1,56	1,56
	SUBTOTAL			6,13
RESISTENCIAS				
Componente	**Nombre**	**Cantidad**	**Precio unitario(€)**	**Total(€)**
Resistencia 57Ω	ERJ-P08F57R6V	1	0,376	0,376
Resistencia 55Ω	ERJ-S08F54R9V	3	0,156	0,468
Resistencias 470Ω	ERJ-P08J471V	2	0,192	0,384
Resistencias 4,7kΩ	ERJ-P08J472V	3	0,077	0,231
Resistencias 10kΩ	ERJ-P08J103V	3	0,176	0,528
Resistencia 10Ω	ERJ-P08J100V	1	0,176	0,176

		SUBTOTAL		2,163
		BOBINAS		
Componente	Nombre	Cantidad	Precio unitario(€)	Total(€)
Bobina 50mH	ELT-3KN028C	1	1,53	1,53
Filtro Modo Común	SRF0504-102Y	1	0,96	0,96
		SUBTOTAL		2,49
		CONECTORES		
Componente	Nombre	Cantidad	Precio unitario(€)	Total(€)
RJ11	615006138521	1	0,81	0,81
Conector USB Panel	PX0443	1	6,87	6,87
Conector DB-9 SMD	K202XHT-E9P-NJ	1	2,06	2,06
Conector Osciloscopio BNC	R141603000W	2	8	16
Conector Acelerómetro	A3A-12PA-2SV(71	2	1,72	3,44
		SUBTOTAL		29,18
		CONMUTADORES		
Componente	Nombre	Cantidad	Precio unitario(€)	Total(€)
Interruptor	1108M2S4V3QE2	1	4,94	4,94
Pulsador	5-1437565-0	1	0,315	0,315
		SUBTOTAL		5,255
		OTROS		
Componente	Nombre	Cantidad	Precio unitario(€)	Total(€)
Jumper	63429-202LF	3	0,078	0,234
Tira de Pines	929834-02-36-RK	1	0,608	0,608
Caja	1590XX	1	9,38	9,38
Placa cobre 457,2x304,8x1,6		1	28,08	28,08
		SUBTOTAL		38,302
		TOTAL MATERIAL		122,054

	DISEÑO Y MANO DE OBRA		
	DISEÑO Y MANO DE OBRA		
Concepto	Horas	Precio unitario(€)	Total(€)
Diseño	100	50	5000
Montaje	10	20	200
	SUBTOTAL		5200
	TOTAL DISEÑO Y MANO DE OBRA		5200

RESUMEN PRESUPUESTO	
TOTAL DISEÑO Y MANO DE OBRA	5200
TOTAL MATERIAL	122,054
TOTAL	5322,1

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

5. ANEXOS

INDICE DE ANEXOS

5.1. ANEXO 1: PROTOCOLOS DE COMUNICACIÓN EMPLEADOS

En la realización del proyecto se han tenido que usar diferentes tecnologías de comunicación, como son el protocolo SPI, SCI e I2C. A continuación se realiza una pequeña descripción de ellos.

3.1.1. SPI

El Bus SPI(Serial Peripheral Interface) es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos. El bus de interfaz de periféricos serie o bus SPI es un estándar para controlar casi cualquier electrónica digital que acepte un flujo de bits serie regulado por un reloj.

Incluye una línea de reloj, dato entrante, dato saliente y un pin de chip select, que conecta o desconecta la operación del dispositivo con el que uno desea comunicarse. De esta forma, este estándar permite multiplexar las líneas de reloj.

La ventaja de un bus serie es que minimiza el número de conductores y el tamaño del circuito integrado. Esto reduce el coste de fabricar, probar y montar la electrónica.

El protocolo SPI requiere de 4 señales:

- SCLK (Clock): Es el pulso que marca la sincronización. Con cada pulso de este reloj, se lee o envía un bit.
- MOSI (Master Output Slave Input) : Salida de datos del Maestro y entrada de datos del Esclavo.
- MISO (Master Input Slave Output): Salida de datos del Esclavo y entrada del Maestro.
- SS/Select: Para seleccionar un Esclavo, o para que el Maestro le diga al Esclavo que se active.

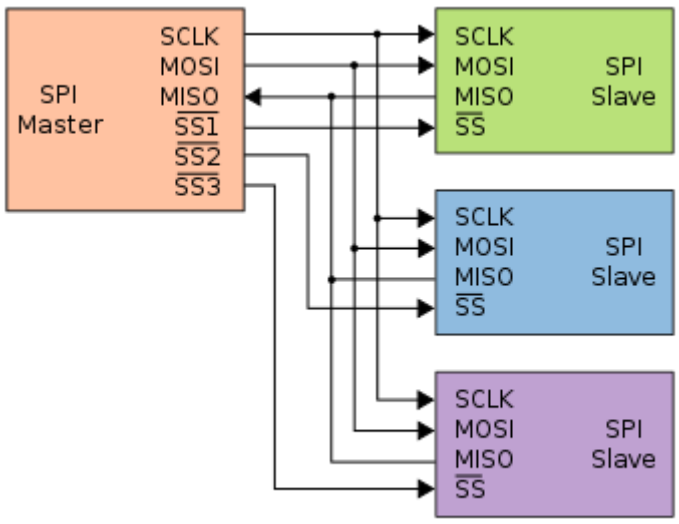


Imagen 1-Conexión Bus SPI

El SPI es un protocolo síncrono. Esto quiere decir que con cada pulso de reloj el Maestro envía un bit. Para que la transmisión se lleve a cabo de forma adecuada, la señal SS del esclavo correspondiente debe estar a cero, manteniendo a 1 las del resto de los esclavos.

La transmisión mediante SPI se puede realizar en cuatro modos diferentes dependiendo de la configuración de dos parámetros. Estos parámetros son:

- CPOL (Clock Polarity): Es la polaridad del reloj. Si es 0 indica que el reloj está bajo mientras el puerto está inactivo. Por el contrario, si es 1 el reloj estará a nivel alto.
- CPHA (Clock Phase): Indica la fase de los datos de salida con respecto al reloj. Si CPHA es 1 significa que el centro del bit de salida corresponde a las segundas transiciones del reloj, mientras que si CPHA es 0 el centro del bit está alineado con la primera transición del reloj.

De esta forma, las diferentes combinaciones en la configuración de estos parámetros constituyen los cuatro modos de transmisión mediante SPI:

1. Con el flanco de subida sin retraso ⟶ CPOL=1 CPHA=1
2. Con el flanco de subida con retraso ⟶ CPOL=1 CPHA=0
3. Con el flanco de bajada sin retraso ⟶ CPOL=0 CPHA=1
4. Con el flanco de bajada con retraso ⟶ CPOL=0 CPHA=0

De las hojas de características de *Microchip* se extrae la siguiente imagen que ilustra estos modos de funcionamiento:

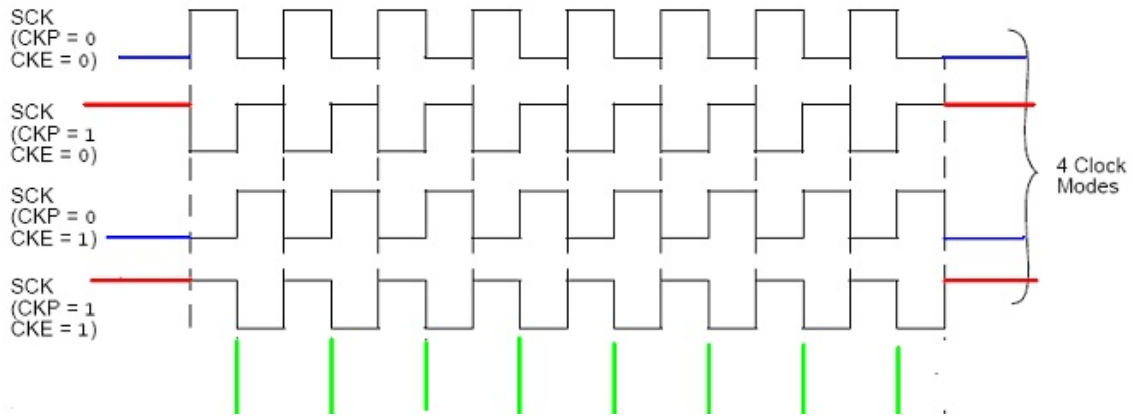


Imagen 2-Modos SPI

En la nomenclatura de Microchip CPOL=CKP y (1-CPHA)=CKE.

3.1.2. SERIAL COMMUNICATION INTERFACE

Un puerto serie o puerto serial es una interfaz de comunicaciones de datos digitales, frecuentemente utilizado por computadoras y periféricos, donde la información es transmitida bit a bit enviando un solo bit a la vez, en contraste con el puerto paralelo que envía varios bits simultáneamente.

3.1.2.1. Introducción

En informática, un puerto serie es una interfaz física de comunicación en serie a través de la cual se transfiere información mandando o recibiendo un bit. A lo largo de la historia de los ordenadores, la transferencia de datos a través del puerto serie ha sido generalizada. Se ha usado o sigue usándose para conectar dispositivos como módems. Los ratones, teclados, y otros periféricos también se conectaban de esta forma.

Mientras que otras interfaces como Ethernet, FireWire, y USB mandaban datos como un flujo en serie, el término "puerto serie" normalmente identifica el hardware más o menos conforme al estándar RS-232, diseñado para interactuar con un módem o con un dispositivo de comunicación similar.

Actualmente en la mayoría de los periféricos serie, la interfaz USB ha reemplazado al puerto serie puesto que es más rápida.Sin embargo, los puertos serie todavía se encuentran en sistemas de automatización industrial y algunos productos industriales y de consumo.

Los dispositivos de redes, como los enrutadores y conmutadores, a menudo tienen puertos series para modificar su configuración. Los puertos serie se usan frecuentemente en estas áreas porque son sencillos, baratos y permiten la interoperabilidad entre dispositivos. La desventaja es que la configuración de las conexiones serie requiere, en la mayoría de los casos, un conocimiento avanzado por parte del usuario y el uso de comandos complejos si la implementación no es adecuada.

3.1.2.2. Puerto serie asíncrono

A través de este puerto se establece usando un protocolo de transmisión asíncrono. En este caso, se envía en primer lugar una señal inicial anterior al primer bit de cada byte, carácter o palabra codificada. Una vez enviado el código correspondiente, se envía inmediatamente una señal de stop después de cada palabra codificada.

La señal de inicio (start) sirve para preparar al mecanismo receptor, la llegada y registro de un símbolo, mientras que la señal de stop sirve para predisponer al mecanismo de recepción para que tome un descanso y se prepare para la recepción del nuevo símbolo.

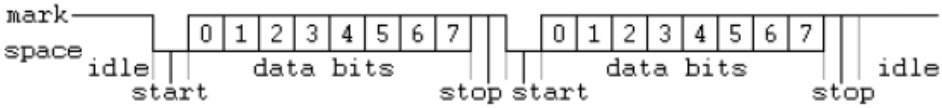


Imagen 3- Trama serie

El puerto serie RS-232 (también conocido como COM) es del tipo asíncrono, utiliza cableado simple desde 3 hilos hasta 25.

El RS-232 original tenía un conector tipo DB-25, sin embargo la mayoría de dichos pines no se utilizaban, por lo que IBM estandarizó con su gama IBM Personal System/2 el uso del conector DB-9 de manera mayoritaria en computadoras.

3.1.3. UART

UART son las siglas de “Universal Asynchronous Receiver-Transmitter”. Éste controla los puertos y dispositivos serie.

El controlador UART es el componente clave del subsistema de comunicaciones series en una computadora. El UART toma bytes de datos y transmite los bits individuales de forma secuencial. En el destino, un segundo UART reensambla los bits en bytes completos. La transmisión serie de la información digital a través de un cable único u otros medios es mucho más efectiva en cuanto a costo que la transmisión en paralelo a través de multiples cables. Se utiliza una UART para convertir la información transmitida entre su forma secuencial y paralela en cada terminal de enlace. Cada UART contiene un registro de desplazamiento que es el método fundamental de conversión entre las formas serie y paralelo.

5.2. ANEXO 2: PROGRAMAS

5.2.1. PROGRAMA PIC

```
/* File:   main.c  TarjetaAcelerómetro
 * Author: Oscar
 *
 * Created on 28 de junio de 2014, 18:41
 */
#include<plib.h>
//#include "spi.h"
/*   DEVCFG0                                                */
/*                                                          */
/*ICESEL=PGEC3/PGED3 pair is used for debugging             */
/*DEBUG=debugger enabled                                    */
/*JTAG DESHABILITADO                                        */
#pragma config ICESEL=ICS_PGx3,DEBUG=ON,JTAGEN=OFF
/*   DEVCFG1                                                */
/*                                                          */
/*FWDTEN = OFF WatchDog Timer disable                       */
/*FPBDIV=Peripheral Busc Clock Divisor=8                    */
/*POSCMOD= External Clock mode selected =HSPLL              */
/*FSOSCEN= Disable Secondary Oscillator                     */
/*FNOSC= FRC with PLL                                       */
#pragma config FWDTEN
=OFF,FPBDIV=DIV_8,FNOSC=FRCPLL,FSOSCEN=OFF,POSCMOD=OFF,FCKSM=CSECME
/*   DEVCFG2                                                */
/*                                                          */
/*FPLLODIV= PLL Output Divisor=2                            */
/*FPLLMUL= 24X                                              */
/*FPLLIDIV=  8/2=4Mhz                                       */
/*FRECUENCIA SYSCLK= (4*24)/2=48Mhz                         */
#pragma config FPLLODIV=DIV_2,FPLLMUL=MUL_24,FPLLIDIV=DIV_2
//SYSCLK=48MHz
//PBCLK=48/8=6MHz
```

```
/*DEFINICION DE BITS IMPORTANTES*/
#define LDAC PORTBbits.RB0
#define CSDAC PORTBbits.RB1
#define CSA PORTBbits.RB7
#define TCS PORTBbits.RB9
#define PULSADOR PORTBbits.RB4
#define LED2 PORTAbits.RA4
#define LED4 PORTBbits.RB5
#define LED5 PORTBbits.RB14
/*VARIABLES GLOBALES*/
char puertoserie=1,cambio=0,lectura=0,eje,configurado=0,offset=0,cuenta=0;
unsigned int
transmitido=1,delay=0,aceleracionX=0,aceleracionX0,aceleracion0,aceleracionX0,aceleracionY0,aceleracionY,aceleracion=0,contador=0,i,aux,veces=0;
unsigned int check;
unsigned short int muestras=3500;
unsigned short int g[3500];
/*definición de funciones*/
void DelayMs(WORD delay)     //función delay para retardo
 {
    unsigned int int_status;
    while( delay-- )
    {
        int_status = INTDisableInterrupts();
        OpenCoreTimer(48000000 / 2000);
        INTRestoreInterrupts(int_status);
        mCTClearIntFlag();
        while( !mCTGetIntFlag() );
    }
    mCTClearIntFlag();
 }
/*****************************************************
         *  UART.lib *
 Librería para envio de datos mediante el protocolo UART en PIC32
 Autor:Oscar Cirauqui Diaz
```

```
**************************************************/

/* ENVIO DE UN BYTE A TRAVÉS DEL MÓDULO UART2*/

 void SendUART(char dato)

 {

                while(U2STAbits.UTXBF);                 //Esperar a que el Buffer de envio esté libre

                U2TXREG=dato;                                                   //Cargo el dato en el
Buffer de envio

                while(!U2STAbits.TRMT);
                //Esperar mientras el SR no esté vacío y haya operaciones pendientes en el módulo UART


 }

 /*********************************************/

/*****************************************************

/* LECTURA DE UN BYTE A TRAVÉS DEL MÓDULO UART2*/

  char ReadUART(void)

 {

                char dato;

                dato=U2RXREG;                                           //Leo el dato del buffer

                while(!U2STAbits.TRMT);                 //Esperar mientras el SR no esté vacío y haya
operaciones pendientes

                while(!U2STAbits.RIDLE);

                return dato;

 }

/*****************************************************

        *  SPItransfer *

 Function: SPItransfer

 Return:

 Arguments:

 Description: envio datos a través de SPI

**************************************************/

unsigned int SPItransfer(unsigned int dato)

{

        SPI2BUF=dato;

        while(SPI2STATbits.SPITBF);

        return(SPI2BUF);

}
```

```
void main(void) {
   /*          CONFIGURACIÓN DE LOS PINES                    */
   /*CONFIGURACIÓN PORTA                                      */
 TRISA=0x04;
ANSELA=0;
   /*CONFIGURACIÓN PORTB                                      */
TRISB=0x2814;
ANSELB=0;

   /*CONFIGURACIÓN DE PPS                                     */
   /*INPUT PIN SELECTION                                     */
  OSCCON=0x46;                                  //DESBLOQUEAR MODIFICACIÓN DEL
REGISTRO DEL PPS
  OSCCON=0x57;
  CFGCONbits.IOLOCK=0;                                  //DESBLOQUEAR MODIFICACIÓN DEL
REGISTRO DEL PPS
  INT4R=0b010;                                  //RB4=INT4  PULSADOR
  U2RXR=0b011;                                  //U2RX=RB11
  SDI2R=0b011;                                  //RB13=SDI2
   /*OUTPUT PIN SELECTION                                    */
  RPB8R=0b100;                                  //SDO2=RB8
 OSCCON=0x8273380;
RPB10R=0b010;                                   //U2TX=RB10

INTEnableSystemSingleVectoredInt();
//Selección de la Prioridad de las Interrupciones
IPC4bits.INT4IP=1;
IPC4bits.INT4IS=3;
//Limpiar flag interrupción y activar interrupción
IFS0bits.INT4IF=0;
IEC0bits.INT4IE=1;                  //Habilitar interrupción de INT4 (pulsador)
int rData;
IEC0CLR=0x03800000;// disable all interrupts
IEC1bits.SPI2RXIE=0;                        //Deshabilitar interrupción de SDI2
IEC1bits.SPI2TXIE=0;                        //Deshabilitar interrupción de SDO2
```

```
SPI2CON = 0; // Stops and resets the SPI2.
SPI2CON2=0;
rData=SPI2BUF;// clears the receive buffer
IFS0CLR=0x03800000;
IPC5CLR=0x1f000000;
IPC5SET=0x0d000000;
IEC0SET=0x03800000;
IFS1=0xFFE3FFFF;                    //Limpiar banderas de interrupción
SPI2BRG=1;
SPI2STATCLR=0x40;
SPI2CON=0x0000A660;
CSA=1;
CSDAC=1;
LDAC=0;
TCS=1;
LED2=1;
LED4=1;
LED5=1;
/*************************************************************/
/*                    INICIALIZACIÓN DEL MODULO UART2                    */
U2BRG=12;                                   //Selección de  BaudRate=115200
 U2STA=0;                                   //Limpiar estado
U2STA=0x1400;
U2MODE=0x8008;                              //Habilitar el módulo de UART2
/*************************************************************/
//        /*Inicialización TIMER1 para delay*/
T1CON=0x000;                        //activar el Timer y limpiar el registro de control
                                    //Prescaler=1
TMR1=0x0;                                   //Limpiar el registro del timer
  PR1=655;                          // Establecer PR1 en 283  para frecuencia de 10kHz
IPC1bits.T1IP=1;                    //Prioridad de interrupción de 1
IPC1bits.T1IS=2;                    //Subprioridad de interrupción de 2 (menos que el pulsador)
//Limpiar flag interrupción y activar interrupción
IFS0bits.T1IF=0;
```

```
IEC0bits.T1IE=1;
/*****************************************************************/
/*DEFINICIÓN VARIABLES MAIN*/
char eje0;
unsigned short int msg=0;
unsigned char respuesta=0,valor1,valor2;
/*****************************************************************/
while(1)
 {
if(puertoserie && configurado==0)   //activado envio por puerto serie pero sin configurar las opciones
{
        check=0;
        configurado=0;
        eje0=0;
        U2STAbits.URXDA=1;
        /*DATO SELECCIÓN EJE ACELERÓMETRO*/
        while(1)
        {
                eje0=ReadUART();
                if(!puertoserie) break;
                if(eje==36 || eje==-86) break;
        }
        if(eje!=eje0) g[3500]=0;
        eje=eje0;
        /*DATO CONFIGURACION FRECUENCIA DE MUESTREO*/
        while(!U2STAbits.URXDA);  //Esperar a que haya dato en el buffer de lectura
        valor1=ReadUART();
        switch (valor1)
        {
                case 1:
                        T1CONbits.ON=0;
                        PR1=589 ;
//                      T1CONbits.ON=1;
                        break;
```

```
                case 2:
                        T1CONbits.ON=0;
                        PR1=982;
                        T1CONbits.ON=1;
                        break;
                case 3:
                        T1CONbits.ON=0;
                        PR1=1474;
                        T1CONbits.ON=1;
                        break;
                case 4:
                        T1CONbits.ON=0;
                        PR1=2948;
                        T1CONbits.ON=1;
                        break;
                case 5:
                        T1CONbits.ON=0;
                        PR1=5896;
                        T1CONbits.ON=1;
                        break;
                case 6:
                        T1CONbits.ON=0;
                        PR1=9826;
                        T1CONbits.ON=1;
                        break;
                case 7:
                        T1CONbits.ON=0;
                        PR1=19654;
                        T1CONbits.ON=1;
                        break;
        }
        configurado=1;
}
else if(puertoserie && configurado==1 )
```

```
{
	check=0;
	while(valor1!=36)
	{
		valor1=ReadUART();
		if(!puertoserie) break;
	}
	T1CONbits.ON=1;
	while(contador<3500)
	{
		if(lectura==1)
		{
			aux=g[contador];
			//SendUART(cuenta);
			check=aux+check;	//dato comprobación errores, suma de todas las aceleraciones
			lectura=0;		//semáforo que muestra si se ha muestreado la aceleración
			/*ENVIO DATO DE ACELERACIÓN*/
			SendUART(170);
			msg=aux&0xFF00;	//Me quedo con la parte alta de aceleraciónX(4 primeros bits)
			msg=msg>>8;		//Desplazo 8 bits hacia la derecha
			SendUART(msg);		//Envio datos a través de la UART
			msg=aux&0xFF;	//Me quedo con la parte baja de aceleraciónX(12 bits menos
significativos)
			SendUART(msg);
			contador++;
		}
	}
	contador=0;
	/*ENVIO SUMA DE LAS ACELERACIONES PARA COMPROBAR ERRORES*/
	msg=check&0xFF000000;				//Envio 1º dato parte alta check
	msg=check>>24;
	SendUART(msg);
	msg=check&0xFF0000;				//Envio 2º dato parte alta check
	msg=check>>16;
```

```
        SendUART(msg);
        msg=check&0xFF;                         //Envio 1º dato parte baja check
        msg=check>>8;
        SendUART(msg);
        msg=check&0xFF;                         //Envio 2º dato parte baja check
        SendUART(msg);
        check=0;
        valor1=0;
        configurado=0;
}
}
}
void __ISR( 0, ipl1) InterruptHandler( void)
{
        if(INTGetFlag(INT_INT4)!=0)             //Si la interrupción se ha producido por INT4
        {
                DelayMs(200);
                puertoserie=~puertoserie;
                puertoserie=puertoserie&0x01;
                INTClearFlag(INT_INT4);                 //Borro bandera interrupción
                if(!puertoserie)                        //Activo el modo del osciloscopio
                {
                        PR1=589 ;                       //Activo el TIMER1
                        T1CONbits.ON=1;
                        LED2=0;
                        LED4=1;
                        LED5=0;
                }
                else                                    //Activo el modo del puertoserie
                {
                        LED2=1;
                        LED4=1;
                        LED5=1;
                }
```

```
}
 else if(IFS0bits.T1IF)                    //Si la interrupcioón se produce por el TIMER1
{
        T1CONbits.ON=0;                    //Desactivo el TIMER
        cambio=~cambio;
        cambio=cambio&0x01;
        IFS0bits.T1IF=0;                   //Borra bandera interrupción
        T1CONbits.ON=1;                    //Activo TIMER1
        if(puertoserie)                    //Si el modo puertoserie está activado
        {
                if(eje==36)                //Eje X seleccionado
                {
                        muestreoX();       //Leo aceleración eje X
                        g[i]=aceleracionX; //guardo los datos en el array hasta que se llene
                                i++;
                                if(i==3500) i=0;
                }
                else    if(eje==-86)               //Eje Y seleccionado
                {
                        muestreoY();
                        g[i]=aceleracionY; //Leo aceleración eje Y
                        i++;               //guardo los datos en el array hasta que se llene
                        if(i==3500) i=0;
                }
                lectura=1;                 //pongo semáforo a 1
        }
        else                               //Si no activo el modo del osciloscopio
        {
                if(cambio)
                {
                        mostrarX();        //mostrar aceleración eje X
                }
                else if(!cambio)
                {
```

```
                                mostrarY();                     //mostrar aceleración eje Y
                        }
                }
        }
}
void  muestreoX(void)
 {
        unsigned int msg=0,basura;
        /*******************************************************/
        /*LECTURA DATOS EJE X ACELEROMETRO*/
        CSA=0;                          //Selecciono el CS del Acelerómetro
        aceleracionX=SPItransfer(0x0400);
        while(SPI2STATbits.SPIBUSY); //Esperar a que no haya transacciones en el módulo SPI
        CSA=1;
        if(aceleracionX<=654 || aceleracionX>=3442)      aceleracionX=aceleracionX0;     //Compruebo
que la aceleración esté dentro del rango correcto, si no vuelvo a enviar el dato anterior
        aceleracionX0=aceleracionX;
}
void muestreoY(void)
{
        unsigned int msg=0,basura;
        ///*ENVIO DATOS EJE CANAL A DAC FUNCIONANDO PERFECTO*/
        CSA=0;                                  //Selecciono el CS del Acelerómetro
        aceleracionY=SPItransfer(0x0C00);
        while(SPI2STATbits.SPIBUSY);      //Esperar a que no haya transacciones en el módulo SPI
        CSA=1;
        if(aceleracionY<=654 || aceleracionY>=3442)      aceleracionY=aceleracionY0;
        //Compruebo que la aceleración esté dentro del rango correcto, si no vuelvo a enviar el dato
anterior
        aceleracionY0=aceleracionY;
}
void  mostrarX(void)
 {
        unsigned int msg=0,basura;
        CSA=0;                          //Selecciono el CS del Acelerómetro
```

```
	basura=SPItransfer(0x0400);		//Envio dato para leer aceleracion eje X
	while(SPI2STATbits.SPIBUSY);	//Esperar a que no haya transacciones
	CSA=1;					//Deselecciono el CS del Acelerómetro
	/*********************************************************/
	/*LECTURA DATOS EJE X ACELEROMETRO FUNCIONANDO PERFECTO*/
	CSA=0;				//Selecciono el CS del Acelerómetro
	aceleracionX=SPItransfer(0x0C00);
	while(SPI2STATbits.SPIBUSY);		//Esperar a que no haya transacciones en el módulo SPI
	CSA=1;
	if(aceleracionX<=654 || aceleracionX>=3442)	aceleracionX=aceleracionX0;	//Compruebo que la aceleración esté dentro del rango correcto, si no vuelvo a enviar el dato anterior
	aceleracionX0=aceleracionX;
}
void mostrarY(void)
{
	unsigned int msg=0,basura;
	///*ENVIO DATOS EJE CANAL A DAC FUNCIONANDO PERFECTO*/
	CSDAC=0;			//Selecciono el CS del DAC
	msg=0x1000+aceleracionX;	//Preparo dato para enviar al DAC canal A
	aceleracionY=SPItransfer(msg);	//envio al DAC aceleracionX
	msg=aceleracionY&0xF00;		//Me quedo con la parte alta de aceleraciónY(4 primeros bits)
	while(SPI2STATbits.SPIBUSY);	//Esperar a que no haya transacciones
	CSDAC=1;				//Deselecciono el CS del DAC
	if(aceleracionY<=654 || aceleracionY>=3442)	aceleracionY=aceleracionY0;	//Compruebo que la aceleración esté dentro del rango correcto, si no vuelvo a enviar el dato anterior
	/*ENVIO DATOS EJE CANAL B DAC FUNCIONANDO PERFECTO*/
	CSDAC=0;			//Selecciono el CS del DAC
	msg=0x9000+aceleracionY;	//Preparo dato a enviar al canal B del DAC
	basura=SPItransfer(msg);	//envio al DAC aceleracionY
	while(!U2STAbits.TRMT);		//Esperar mientras el SR no esté vacío y haya operaciones pendientes en el módulo UART
	while(SPI2STATbits.SPIBUSY);	//Esperar a que no haya transacciones
	CSDAC=1;				//Deselecciono el CS del DAC
	aceleracionY0=aceleracionY;
}
```

5.2.2. PROGRAMA MATLAB

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

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

% Last Modified by GUIDE v2.5 19-Oct-2014 15:20:01

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

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


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

```
function SerialADIS_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to SerialADIS (see VARARGIN)

% Choose default command line output for SerialADIS
global escalay escalax Pescala parada escalaxFFT escalayFFT;

handles.modograf='Tiempo';
handles.eje='X';
handles.PS=serial('COM1');
handles.configurado=0;
handles.muestras=3500;
handles.fm=5000;
handles.frecuencia=1;
handles.boton_estado=0;
parada=0;
escalay=2.498;
escalax=3500;
escalaxFFT=2250;
escalayFFT=1;
Pescala=0;
handles.EscalaX=0.701;
global numarchivo Nombre hoja;
numarchivo=1;
Nombre='Vibracion_';
hoja=1;

% Update handles structure
handles.output = hObject;
guidata(hObject, handles);

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


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

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




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

```
% Hints: contents = cellstr(get(hObject,'String')) returns PuertoSerie
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
PuertoSerie

port = get(hObject, 'Value');
switch port
    case 1
        PS=serial('COM1');
    case 2
        PS=serial('COM2');
    case 3
        PS=serial('COM3');
    case 4
        PS=serial('COM4');
    case 5
        PS=serial('COM5');
    case 6
        PS=serial('COM6');
    case 7
        PS=serial('COM7');
    case 8
        PS=serial('COM8');
    case 9
        PS=serial('COM9');
    case 10
        PS=serial('COM10');
    case 11
        PS=serial('COM11');
    case 12
        PS=serial('COM12');
    case 13
        PS=serial('COM13');
    case 14
        PS=serial('COM14');
    case 15
        PS=serial('COM15');
end
handles.PS=PS;
guidata(hObject,handles);

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

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




% --- Executes on selection change in Fmuestreo.
```

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

% Hints: contents = cellstr(get(hObject,'String')) returns Fmuestreo
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
Fmuestreo

valor=get(hObject,'Value');     %obtener valor introducido
switch valor
    case 1
        fm=5000;
    case 2
        fm=3000;
    case 3
        fm=2000;
    case 4
        fm=1000;
    case 5
        fm=500;
    case 6
        fm=300;
    case 7
        fm=150;
end
handles.fm=fm;
handles.frecuencia=valor;            %guardo puntero a frecuencia
guidata(hObject,handles);

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

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


% --- Executes when selected object is changed in uipanel4.
function uipanel4_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to the selected object in uipanel4
% eventdata  structure with the following fields (see UIBUTTONGROUP)
%   EventName: string 'SelectionChanged' (read only)
%   OldValue: handle of the previously selected object or empty if
none was selected
%   NewValue: handle of the currently selected object
% handles    structure with handles and user data (see GUIDATA)

if (hObject==handles.x)
    %fwrite(handles.PS,41);  %envio un 41 para elegir eje X
    %msgbox('Eje X');
```

```
    handles.eje='X';
else
    %fwrite(handles.PS,42);  %envio un 42 para elegir eje Y
    %msgbox('Eje Y');
    handles.eje='Y';
end
guidata(hObject,handles);


% --- Executes when selected object is changed in uipanel5.
function uipanel5_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to the selected object in uipanel5
% eventdata  structure with the following fields (see UIBUTTONGROUP)
%   EventName: string 'SelectionChanged' (read only)
%   OldValue: handle of the previously selected object or empty if
none was selected
%   NewValue: handle of the currently selected object
% handles    structure with handles and user data (see GUIDATA)

if (hObject==handles.Tiempo)
    %fwrite(handles.PS,41);  %envio un 41 para elegir eje X
    %msgbox('Tiempo');
    handles.modograf='Tiempo';
else
    %fwrite(handles.PS,42);  %envio un 42 para elegir eje Y
    %msgbox('FFT');
    handles.modograf='FFT';
end
guidata(hObject,handles);


% --- Executes on button press in Iniciar.
function Iniciar_Callback(hObject, eventdata, handles)
% hObject    handle to Iniciar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of Iniciar
global X veces v t parada numarchivo;
 X=[];
 veces=0;

 pulsador=get(hObject,'String');
%Configurar y abrir puerto
if  strcmp(pulsador,'Iniciar')
        set(handles.Iniciar,'String','Detener');
        v=0;
        parada=0;
        handles.PS.BaudRate=115200; % se configura la velocidad a
115200 Baudios
        set(handles.PS,'StopBits',1); % se configura bit de parada a
uno
        set(handles.PS,'DataBits',8); % se configura que el dato es de
8 bits, debe estar entre 5 y 8
        set(handles.PS,'Parity','none'); % se configura sin paridad
        set(handles.PS,'InputBufferSize',80000);
        set(handles.PS,'Timeout',10);
        set(handles.PS,'BytesAvailableFcnCount',(handles.muestras)*3);
        set(handles.PS,'BytesAvailableFcnMode','byte');
        handles.PS.ErrorFcn = {@errorcallback,handles};
```

```
        %set(handles.PS,'BytesAvailableFcn',@lecturacallback);
        handles.PS.BytesAvailableFcn = {@mycallback,handles};
        % handles.PS.BytesAvailableFcnCount =handles.muestras*3;
        % handles.PS.BytesAvailableFcnMode = 'byte';
        % handles.PS.BytesAvailableFcn = @lecturacallback;
        try
            fopen(handles.PS);
            set(handles.EstadoPuerto, 'string','conectado');
        catch e
            set(handles.EstadoPuerto, 'string', 'desconectado');
            set(handles.Iniciar,'string','Iniciar');
            errordlg(e.message);

        end
        t= timer(...
            'StartFcn',         @user_timer_start, ...              %
start function
            'TimerFcn',         {@user_timer_update, handles}, ...  %
timer function, has to specific the handle to the GUI,
            'StopFcn',          @user_timer_stop, ...               %
stop function
            'ErrorFcn',         @user_timer_err, ...                %
error function
            'ExecutionMode',    'fixedRate', ...                    %
            'Period',           400, ...                           %
updates every xx seconds
            'TasksToExecute',   inf, ...
            'BusyMode',         'drop');

        % fwrite(handles.PS,170);%envio dato para activar modo serie

        % while dato~=170           %esperar a recibir el mismo dato
        %     dato=fread(handles.PS);
        % end

        flushinput(handles.PS);  %vaciar buffer
        if strcmp(handles.eje,'X')
               fwrite(handles.PS,36);
        else
               fwrite(handles.PS,170);
        end
        fwrite(handles.PS,handles.frecuencia);   %envio seleccion
frecuencia muestreo




        fwrite(handles.PS,36);%envio dato para iniciar la comunicación



        start(t);
elseif strcmp(pulsador,'Detener')
        set(handles.Iniciar,'string','Iniciar');
        if v==0
            delete(t);
            stop(t);
            clear timer;
```

```
            v=1;
        end
        parada=1;
        %delete(handles.t);

        %     hoja=eval('hoja+1');
        fclose(handles.PS);
        set(handles.EstadoPuerto, 'string', 'desconectado');
         if (handles.boton_estado)==1
             numarchivo=eval('numarchivo+1');
         end
end

guidata(hObject,handles);

function user_timer_update(hObject,eventdata,handles)
global v t;

    if v==0
        if handles.PS.BytesAvailable==0
             stop(t);
            delete(t);
            clear timer;
            v=1;
            fclose(handles.PS);
            errordlg('No se ha podido establecer la comunicación. Por
favor, compruebe las conexiones');
        end
    end



% --- Executes on button press in ExportExcel.
function ExportExcel_Callback(hObject, eventdata, handles)
% hObject    handle to ExportExcel (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global G nombrein;
if nombrein==1
    d=linspace(1,handles.muestras,handles.muestras);
    M=[d;G];
    Mt=M';
    ext='.xls';
    a=strcat(handles.Nombre,ext);
    xlswrite(a,Mt);
    nombrein=0;
else
    msgbox('Introduzca un nombre de archivo');
end



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

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

```
%        str2double(get(hObject,'String')) returns contents of
NombreExcel as a double
global nombrein;
nombrein=1;
Nombre=get(hObject,'String');
handles.Nombre=Nombre;
guidata(hObject,handles);

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

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




% --- Executes on button press in Editar1.
function Editar1_Callback(hObject, eventdata, handles)
% hObject    handle to Editar1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global G escalay Pescala parada;
if parada==1
    figure;
    plot([0:handles.muestras-1]./handles.fm,G);
    xlabel('segundos');
    ylabel('aceleración (g)');
    if(Pescala==1)
         xlim('manual');
         xlim([0 handles.EscalaX]);
         ylim('manual');
         ylim([-escalay escalay]);
    end
elseif parada==0
    msgbox('Debe pulsar "Detener" previamente ');
end
guidata(hObject,handles);

% --- Executes on button press in Editar2.
function Editar2_Callback(hObject, eventdata, handles)
% hObject    handle to Editar2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global G escalaxFFT escalayFFT Pescala parada;
if parada==1
    figure;
     fs=handles.fm;
```

```
     N=handles.muestras;
     df=fs/N;
     fHz=-(fs/2):df:(fs/2)*((1-(1/N)));
     F=fftshift(fft(G)/N);
     plot(fHz,2*abs(F));
    xlabel('frecuencia(Hz)');
    ylabel('aceleración(g)');
    if(Pescala==1)
        xlim('manual');
        xlim([-escalaxFFT escalaxFFT]);
        ylim('manual');
        ylim([0 escalayFFT]);
    end
elseif parada==0
    msgbox('Debe pulsar "Detener" previamente ');
end
guidata(hObject,handles);




function mycallback(hObject,eventdata,handles)
global t t1 Pescala escalaxFFT escalayFFT v offset;
if v==0
    stop(t);
    delete(t);
    clear timer;
    v=1;
end
%     delete(handles.t);
global G hoja escalay escalax;
global numarchivo Nombre;
global X veces celdaini celdafin rango;

offset=0;
media=0;
celdaini=0;
celdafin=35000;
contador=1;
    lectura=0;
    check=0;
    aceleracion=[];
    aceleracionG=[];
    matriz=zeros(1,handles.muestras*3);
    matriz = (fread(handles.PS,(handles.muestras*3)))';  %lectura
datos
    handles = guidata(gcf);
%     plot(matriz);
%     axis([0,5000,-1,260]);
%         [check1,3,msg] = fread(obj,size,'precision');
%         msgbox(msg);
        %lectura del checksum'No es posible establecer la
comunicación. Revise las conexione
        check1= fread(handles.PS,1,'uint8');
        check2= fread(handles.PS,1,'uint8');
        check3= fread(handles.PS,1,'uint8');
        check4= fread(handles.PS,1,'uint8');
        check1=check1*2^24;
        check2=check2*2^16;
```

```
        check3=check3*2^8;
        check1=bitor(check1,check2);
        check2=bitor(check3,check4);
        check=check1+check2;
         %procesado de datos
        suma=0;
        contador=1;
        i=1;
        while contador<=(handles.muestras*3)-2
            prueba(i)=matriz(contador);
            lectura1=matriz(contador+1);
            lectura2=matriz(contador+2);
            lectura1=bitshift(lectura1,8);
            lectura=lectura1+lectura2;
            aceleracion(i)=lectura;
            suma=suma+aceleracion(i);
            aceleracionG(i)=(aceleracion(i)-2048)/820;
            contador=contador+3;
            i=i+1;
        end

            %compruebo si suma es igual a check
            if suma==check
                        plot(handles.axes1,[0:handles.muestras-
1]./handles.fm,aceleracionG);
                        xlabel(handles.axes1,'segundos');
                        ylabel(handles.axes1,'aceleración (g)');
                        i=1;
                        if(Pescala==1)
                            while i<=handles.muestras
                                media=media+G(i);
                                i=i+1;
                            end
                            media=media/handles.muestras;
                            if  media>=0.7 && media<=1.3
                                offset=media;
                            elseif media<=-0.7 && media>=-1.3
                                offset=media;
                            elseif media<=-3 && media>=3
                                offset=0;
                            end
                            xlim(handles.axes1,'manual');
                            xlim(handles.axes1,[0 handles.EscalaX]);
                            ylim(handles.axes1,'manual');
                            ylim(handles.axes1,[-escalay+offset
escalay+offset]);
                        end

                            fs=handles.fm;
                            N=handles.muestras;
                            df=fs/N;
                            fHz=-(fs/2):df:(fs/2)*((1-(1/N)));

                            F=fftshift(fft(aceleracionG)/N);
                            plot(handles.axes2,fHz,2*abs(F)); %cambiar
vectores aqui

%                             xlim('manual');
%
```

```
%                               xlim([0 handles.EscalaX]);
%                               ylim('manual');
%                               ylim([-escalay escalay]);
% %                              ylim([-2.498,2.498])
                              xlabel(handles.axes2,'frecuencia (Hz)');
                              ylabel(handles.axes2,'aceleración (g)');
                              if(Pescala==1)
                                  xlim(handles.axes2,'manual');
                                  xlim(handles.axes2,[-escalaxFFT
escalaxFFT]);
                                  ylim(handles.axes2,'manual');
                                  ylim(handles.axes2,[0 escalayFFT]);
                              end
            else
                errordlg('Error lectura datos');
            end

handles.aceleracionG=aceleracionG;

% fwrite(handles.PS,170);%envio dato para activar modo serie
% while(fread(handles.PS)~=170);%esperar a recibir el mismo dato

% while dato~=170           %esperar a recibir el mismo dato
%     dato=fread(handles.PS);
% end


if strcmp(handles.eje,'X')
       fwrite(handles.PS,36);
else
       fwrite(handles.PS,170);
end
fwrite(handles.PS,handles.frecuencia);   %envio seleccion frecuencia
muestreo

flushinput(handles.PS);  %vaciar buffer

fwrite(handles.PS,36);%envio dato para iniciar la comunicación


G=aceleracionG;

if (handles.boton_estado)==1
      X=[X aceleracionG];
      veces=veces+1;
      if veces==10
          d=linspace(1,3500*veces,3500*veces);
          M=[d;X];
          Mt=M';
          ext='.xls';
          str=int2str(numarchivo);
          b=strcat(Nombre,str)
          a=strcat(b,ext);
          xlswrite(a,Mt,hoja);
          hoja=eval('hoja+1');
          veces=0;
          X=[];
      end
```

```
end



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

%cerrar puerto y borrar datos
fclose(handles.PS);
delete(handles.PS);
% Hint: delete(hObject) closes the figure
delete(hObject);


% --- Executes on button press in checkbox1.
function checkbox1_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox1

boton_estado = get(handles.checkbox1,'value');
handles.boton_estado=boton_estado;
guidata(hObject,handles);


function user_timer_start(src, evt)
disp('Timer started!');

function user_timer_stop(src, evt)
disp('Timer stop');

function user_timer_err(src, evt)
disp('Timer error');


% --- Executes on slider movement.
function EscalaY_Callback(hObject, eventdata, handles)
% hObject    handle to EscalaY (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global escalay Pescala G offset;
handles.EscalaY=get(hObject,'Value');
%añadido ahora pruebas papel
i=1;
media=0;
% while i<=handles.muestras
%     media=media+G(i);
%     i=i+1;
% end
```

```
% media=media/handles.muestras;
% if  media<=0.7 && media>=1.3
%     offset=1;
% elseif media<=-0.7 && media>=-1.3
%     offset=-1;
% else
%     offset=0;
% end
handles.EscalaY=2.498-2.498.*handles.EscalaY+0.01;
escalay=handles.EscalaY;
if(Pescala==1)
%       ylim(handles.axes1,'manual');
          ylim(handles.axes1,[-escalay+offset escalay+offset]);
end
guidata(hObject,handles);
%  ylim([-handles.EscalaY handles.EscalaY]);


% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider


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

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
handles.EsccalaY=2.498;


% --- Executes on slider movement.
function EscalaX_Callback(hObject, eventdata, handles)
% hObject    handle to EscalaX (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider
global escalax Pescala;
handles.EscalaX=get(hObject,'Value');
% if strcmp(handles.modograf,'Tiempo')
handles.EscalaX=0.7-0.7.*handles.EscalaX+0.01;
if(Pescala==1)
      xlim(handles.axes1,[0 handles.EscalaX]);
%       ylim(handles.axes1,'manual');
%       ylim(handles.axes1,[-escalay escalay]);
end
escalax=handles.EscalaX;
guidata(hObject,handles);
```

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

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: place code in OpeningFcn to populate axes1


% --- Executes on mouse press over axes background.
function axes1_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global graftiempo
graftiempo=1;   %se pone a uno cuando se pulsa subre el área del
gráfico


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

% Hint: get(hObject,'Value') returns toggle state of
Escalapersonalizada
global Pescala;
Pescala = get(handles.Escalapersonalizada,'value');
guidata(hObject,handles);


% --- Executes on slider movement.
function escalay_FFT_Callback(hObject, eventdata, handles)
% hObject    handle to escalay_FFT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider
```

```
global escalayFFT Pescala;
handles.escalay_FFT=get(hObject,'Value');
handles.escalay_FFT=1-1.*handles.escalay_FFT+0.1;
escalayFFT=handles.escalay_FFT;
if(Pescala==1)
%       ylim(handles.axes1,'manual');
        ylim(handles.axes2,[-escalayFFT escalayFFT]);
end

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

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function escalax_FFT_Callback(hObject, eventdata, handles)
% hObject    handle to escalax_FFT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider
global escalaxFFT Pescala;
handles.escalax_FFT=get(hObject,'Value');
handles.escalax_FFT=2250-2250.*handles.escalax_FFT+0.1;
escalaxFFT=handles.escalax_FFT;
if(Pescala==1)
%       ylim(handles.axes1,'manual');
        xlim(handles.axes2,[-escalaxFFT escalaxFFT]);
end

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

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

5.3. ANEXO 3: ENSAYO EN LA EMPRESA PERMAGSA

Tras haber comprobado el correcto funcionamiento del sistema en el laboratorio se procede a la realización de un ensayo en la empresa PERMAGSA para comprobar el funcionamiento del sistema en el campo de aplicación. Para ello, se ha realizado el ensayo con un ascensor en una de las torres de ensayos de sus instalaciones.

Imagen 1- Torre de ensayo

Para hacer el ensayo se han realizado varios viajes en ascensor colocando el acelerómetro en el suelo del mismo y de forma vertical para captar las vibraciones transmitidas durante el recorrido del ascensor. El motor usado en el ascensor es un nuevo prototipo de motor de la empresa.

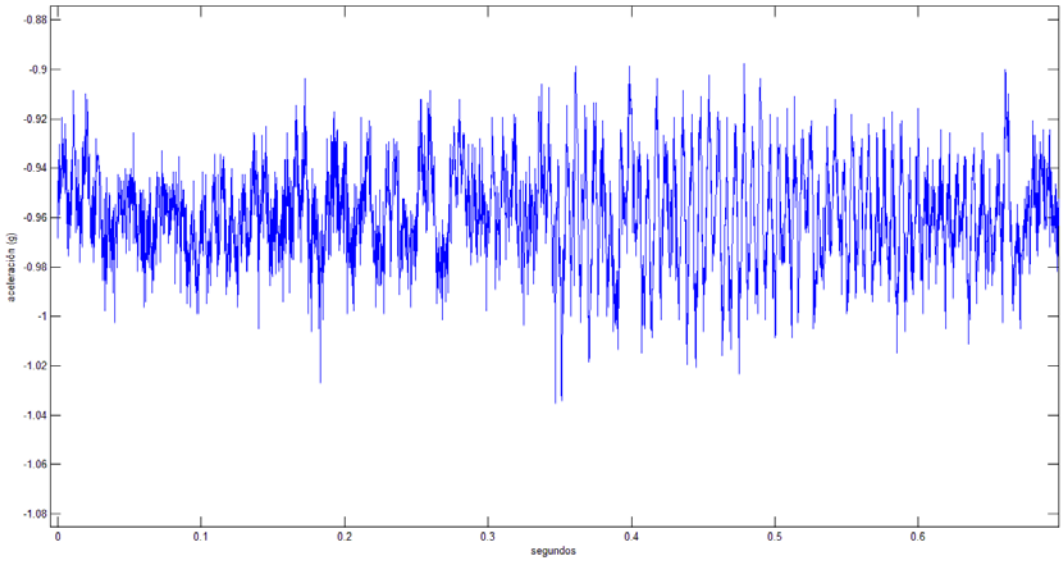


Imagen 2- Medida arranque motor

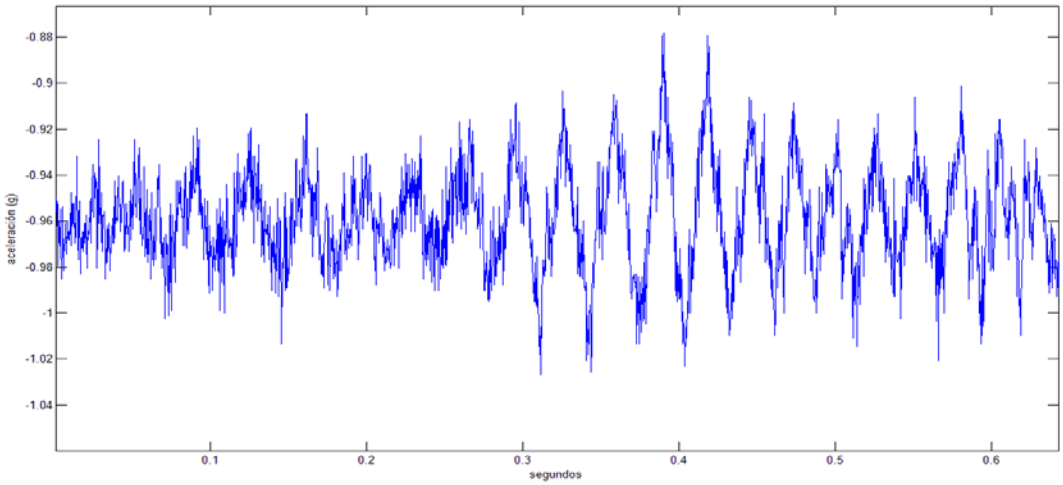


Imagen 3- Medida ascensor subiendo

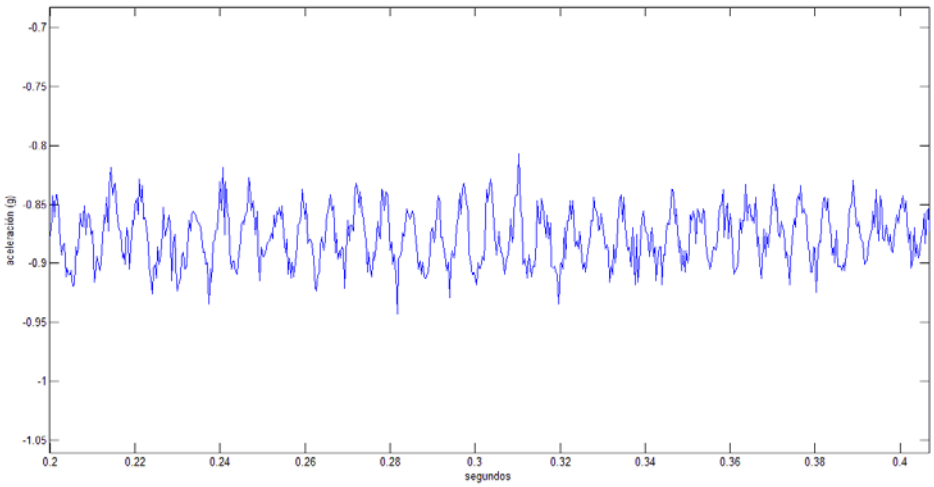


Imagen 4- Medida ascensor bajando

Hay que recalcar que el ascensor del ensayo funcionaba bastante bien con lo que las vibraciones que genera son mínimas. Muy pequeñas para el pasajero del ascensor, por lo que la sensibilidad del acelerómetro es buena ya que se han conseguido captar estas vibraciones muy bien. Se han probado distintas configuraciones en cuanto a la frecuencia de muestreo, aunque se ha visto que la mejor forma de captar la señal es con la frecuencia de muestreo de 5000KSps, ya que al bajarla la forma de la señal que se obtiene es peor.

Comentario de las imágenes:

1. Imagen 2: en el arranque del motor se observan más vibraciones y a una frecuencia mayor
2. Imagen 3: imagen capturada en un punto intermedio del recorrido del ascensor al subir. La vibración tiene una amplitud mayor que en el caso de la tercera imagen ya que al subir el motor está trabajando más.
3. Imagen 4: al bajar el ascensor se ven unas vibraciones más suaves y con una menor amplitud.

Por último, se ha realizado una prueba con un osciloscopio, aunque este método de medida no es tan interesante por su menor versatilidad a la hora de trabajar con los datos obtenidos. El resultado obtenido es el mismo que con la aplicación.

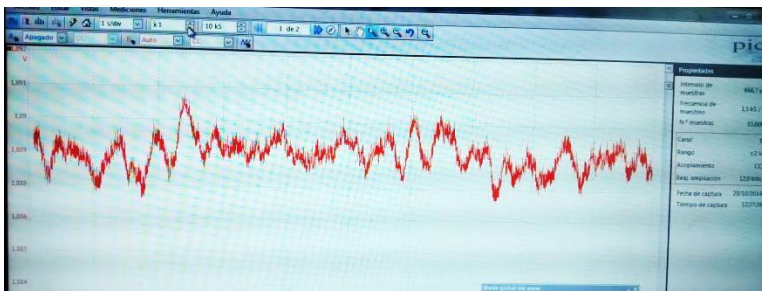


Imagen 5- Medida osciloscopio

5.4. ANEXO 4: HOJAS DE CARACTERÍSTICAS

23.2 STATUS AND CONTROL REGISTERS

Note: Each PIC32 family device variant may have one or more SPI modules. An 'x' used in the names of pins, control/Status bits, and registers denotes the particular module. Refer to the specific device data sheets for more details.

The SPI module consists of the following Special Function Registers (SFRs):

- **SPIxCON: SPI Control Register**
- **SPIxCON2: SPI Control Register 2**
- **SPIxSTAT: SPI Status Register**
- **SPIxBUF: SPI Buffer Register**
- **SPIxBRG: SPI Baud Rate Register**

Table 23-3 summarizes all SPI-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register.

Table 23-3: SPI SFR Summary

Name	Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
SPIxCON(1,2,3)	31:24	FRMEN	FRMSYNC	FRMPOL	MSSEN(4)	FRMSYPW(4)	FRMCNT<2:0>(4)		
	23:16	MCLKSEL(4)	—	—	—	—	—	SPIFE	ENHBUF(4)
	15:8	ON	—	SIDL	DISSDO	MODE32	MODE16	SMP	CKE
	7:0	SSEN	CKP	MSTEN	DISSDI(4)	STXISEL<1:0>(4)		SRXISEL<1:0>(4)	
SPIxCON2(1,2,3,5)	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	SPISGNEXT	—	—	FRMERREN	SPIROVEN	SPITUREN	IGNROV	IGNTUR
	7:0	AUDEN	—	—	—	AUDMONO	—	AUDMOD<1:0>	
SPIxSTAT(1,2,3)	31:24	—	—	—	RXBUFELM<4:0>(4)				
	23:16	—	—	—	TXBUFELM<4:0>(4)				
	15:8	—	—	—	FRMERR(4)	SPIBUSY	—	—	SPITUR
	7:0	SRMT(4)	SPIROV	SPIRBE(4)	—	SPITBE	—	SPITBF(4)	SPIRBF
SPIxBUF	31:24	DATA<31:24>							
	23:16	DATA<23:16>							
	15:8	DATA<15:8>							
	7:0	DATA<7:0>							
SPIxBRG(1,2,3)	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	BRG<12:8>(6)				
	7:0	BRG<7:0>(6)							

Legend: — = unimplemented, read as '0'. Address offset values are shown in hexadecimal.

Note 1: This register has an associated Clear register at an offset of 0x4 bytes. These registers have the same name with CLR appended to the end of the register name (e.g., SPIxCONCLR). Writing a '1' to any bit position in the Clear register will clear valid bits in the associated register. Reads from the Clear register should be ignored.

2: This register has an associated Set register at an offset of 0x8 bytes. These registers have the same name with SET appended to the end of the register name (e.g., SPIxCONSET). Writing a '1' to any bit position in the Set register will set valid bits in the associated register. Reads from the Set register should be ignored.

3: This register has an associated Invert register at an offset of 0xC bytes. These registers have the same name with INV appended to the end of the register name (e.g., SPIxCONINV). Writing a '1' to any bit position in the Invert register will invert valid bits in the associated register. Reads from the Invert register should be ignored.

4: This bit is not available on all devices. Refer to the specific device data sheet for details.

5: This register is not available on all devices. Refer to the specific data sheet for details.

6: Depending on the device, BRG can have up to 13 bits. Refer to the specific device data sheet for details.

Register 23-1: SPIxCON: SPI Control Register

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FRMEN	FRMSYNC	FRMPOL	MSSEN(1,2)	FRMSYPW(1)	FRMCNT<2:0>(1)		
23:16	R/W-0	U-0	U-0	U-0	U-0	U-0	R/W-0	R/W-0
	MCLKSEL	—	—	—	—	—	SPIFE	ENHBUF(1)
15:8	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	ON	—	SIDL	DISSDO	MODE32	MODE16	SMP	CKE
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	SSEN	CKP	MSTEN	DISSDI	STXISEL<1:0>(1,3)		SRXISEL<1:0>(1,3)	

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

bit 31 **FRMEN:** Framed SPI Support bit
1 = Framed SPI support is enabled ($\overline{SSx}$ pin used as FSYNC input/output)
0 = Framed SPI support is disabled

bit 30 **FRMSYNC:** Frame Sync Pulse Direction Control on $\overline{SSx}$ pin bit (Framed SPI mode only)
1 = Frame sync pulse input (Slave mode)
0 = Frame sync pulse output (Master mode)

bit 29 **FRMPOL:** Frame Sync Polarity bit (Framed SPI mode only)
1 = Frame pulse is active-high
0 = Frame pulse is active-low

bit 28 **MSSEN:** Master Mode Slave Select Enable bit(1,2)
1 = Slave select SPI support enabled. The $\overline{SS}$ pin is automatically driven during transmission in Master mode. Polarity is determined by the FRMPOL bit
0 = Slave select SPI support is disabled

bit 27 **FRMSYPW:** Frame Sync Pulse Width bit(1)
1 = Frame sync pulse is one word length wide, as defined by MODE<32,16> bits (SPIxCON<11:10>)
0 = Frame sync pulse is one clock wide

bit 26-24 **FRMCNT<2:0>:** Frame Sync Pulse Counter bits
This bit controls the number of data characters transmitted per pulse(1).
111 = Reserved; do not use
110 = Reserved; do not use
101 = Generate a frame sync pulse on every 32 data characters
100 = Generate a frame sync pulse on every 16 data characters
011 = Generate a frame sync pulse on every 8 data characters
010 = Generate a frame sync pulse on every 4 data characters
001 = Generate a frame sync pulse on every 2 data characters
000 = Generate a frame sync pulse on every data character
This bit is only valid in Framed SPI mode (FRMEN = 1).

bit 23 **MCLKSEL:** Master Clock Select bit(2)
1 = MCLK is used by the Baud Rate Generator
0 = PBCLK is used by the Baud Rate Generator

bit 22-18 **Unimplemented:** Write '0'; ignore read

Note 1: These bits are not available on all devices. Refer to the specific device data sheet for availability.
2: When FRMEN = 1, the MSSEN bit is not used.
3: Valid only when enhanced buffers are enabled (ENHBUF = 1).

Register 23-1: SPIxCON: SPI Control Register (Continued)

bit 17 **SPIFE:** Frame Sync Pulse Edge Select bit (Framed SPI mode only)
1 = Frame synchronization pulse coincides with the first bit clock
0 = Frame synchronization pulse precedes the first bit clock

bit 16 **ENHBUF:** Enhanced Buffer Enable bit(1)
1 = Enhanced Buffer mode is enabled
0 = Enhanced Buffer mode is disabled

bit 15 **ON:** SPI Peripheral On bit
1 = SPI Peripheral is enabled
0 = SPI Peripheral is disabled

When ON = 1, DISSDO and DISSDI are the only other bits that can be modified. When using the 1:1 PBCLK divisor, the user's software should not read or write the peripheral's SFRs in the SYSCLK cycle immediately following the instruction that clears the module's ON bit.

bit 14 **Unimplemented:** Write '0'; ignore read

bit 13 **SIDL:** Stop in Idle Mode bit
1 = Discontinue operation when CPU enters in Idle mode
0 = Continue operation in Idle mode

bit 12 **DISSDO:** Disable SDOx pin bit
1 = SDOx pin is not used by the module (pin is controlled by associated PORT register)
0 = SDOx pin is controlled by the module

bit 11-10 **MODE<32,16>:** 32/16-bit Communication Select bits

When AUDEN = 1:

MODE32	MODE16	Communication
1	1	24-bit Data, 32-bit FIFO, 32-bit Channel/64-bit Frame
1	0	32-bit Data, 32-bit FIFO, 32-bit Channel/64-bit Frame
0	1	16-bit Data, 16-bit FIFO, 32-bit Channel/64-bit Frame
0	0	16-bit Data, 16-bit FIFO, 16-bit Channel/32-bit Frame

When AUDEN = 0:

MODE32	MODE16	Communication
1	x	32-bit
0	1	16-bit
0	0	8-bit

bit 9 **SMP:** SPI Data Input Sample Phase bit
Master mode (MSTEN = 1):
1 = Input data sampled at end of data output time
0 = Input data sampled at middle of data output time

Slave mode (MSTEN = 0):
SMP value is ignored when SPI is used in Slave mode. The module always uses SMP = 0.

bit 8 **CKE:** SPI Clock Edge Select bit
1 = Serial output data changes on transition from active clock state to idle clock state (see CKP bit)
0 = Serial output data changes on transition from idle clock state to active clock state (see CKP bit)

The CKE bit is not used in the Framed SPI mode. The user should program this bit to '0' for the Framed SPI mode (FRMEN = 1).

bit 7 **SSEN:** Slave Select Enable (Slave mode) bit
1 = $\overline{SSx}$ pin used for Slave mode
0 = $\overline{SSx}$ pin not used for Slave mode, pin controlled by port function.

bit 6 **CKP:** Clock Polarity Select bit
1 = Idle state for clock is a high level; active state is a low level
0 = Idle state for clock is a low level; active state is a high level

Note 1: These bits are not available on all devices. Refer to the specific device data sheet for availability.
2: When FRMEN = 1, the MSSEN bit is not used.
3: Valid only when enhanced buffers are enabled (ENHBUF = 1).

Register 23-1: SPIxCON: SPI Control Register (Continued)

bit 5 **MSTEN:** Master Mode Enable bit
1 = Master mode
0 = Slave mode

bit 4 **DISSDI:** Disable SDI bit
1 = SDIx pin is not used by the SPI module (pin is controlled by PORT function)
0 = SDIx pin is controlled by the SPI module

bit 3-2 **STXISEL<1:0>:** SPI Transmit Buffer Empty Interrupt Mode bits(1,3)
11 = SPIxTXIF is set when the buffer is not full (has one or more empty elements)
10 = SPIxTXIF is set when the buffer is empty by one-half or more
01 = SPIxTXIF is set when the buffer is completely empty
00 = SPIxTXIF is set when the last transfer is shifted out of SPISR and transmit operations are complete

bit 1-0 **SRXISEL<1:0>:** SPI Receive Buffer Full Interrupt Mode bits(1,3)
11 = SPIxRXIF is set when the buffer is full
10 = SPIxRXIF is set when the buffer is full by one-half or more
01 = SPIxRXIF is set when the buffer is not empty
00 = SPIxRXIF is set when the last word in the receive buffer is read (i.e., buffer is empty)

Note 1: These bits are not available on all devices. Refer to the specific device data sheet for availability.
2: When FRMEN = 1, the MSSEN bit is not used.
3: Valid only when enhanced buffers are enabled (ENHBUF = 1).

Register 23-2: SPIxCON2: SPI Control Register 2

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
23:16	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1	R/W-0	R/W-0
	SPISGNEXT	—	—	FRMERREN	SPIROVEN	SPITUREN	IGNROV	IGNTUR
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	AUDEN(1,3)	—	—	—	AUDMONO(2)	—	AUDMOD<1:0>(2,4)	

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

bit 31-16 **Unimplemented:** Write '0'; ignore read

bit 15 **SPISGNEXT:** Sign Extend Read Data from the RX FIFO bit
1 = Data from RX FIFO is sign extended
0 = Data from RX FIFO is not sign extened

bit 14-13 **Unimplemented:** Write '0'; ignore read

bit 12 **FRMERREN:** Enable Interrupt Events via FRMERR bit
1 = Frame Error overflow generates error interrupts
0 = Frame Error does not generate error interrupts

bit 11 **SPIROVEN:** Enable Interrupt Events via SPIROV bit
1 = Receive overflow generates error interrupts
0 = Receive overflow does not generate error interrupts

bit 10 **SPITUREN:** Enable Interrupt Events via SPITUR bit
1 = Transmit Underrun generates error interrupts
0 = Transmit Underrun does not generate error interrupts

bit 9 **IGNROV:** Ignore Receive Overflow bit (for Audio Data Transmissions)
1 = A ROV is not a critical error; during ROV data in the FIFO is not overwritten by receive data
0 = A ROV is a critical error which stop SPI operation

bit 8 **IGNTUR:** Ignore Transmit Underrun bit (for Audio Data Transmissions)
1 = A TUR is not a critical error and zeros are transmitted until the SPIxTXB is not empty
0 = A TUR is a critical error which stop SPI operation

bit 7 **AUDEN:** Enable Audio CODEC Support bit(1,3)
1 = Audio protocol enabled
0 = Audio protocol disabled

bit 6-5 **Unimplemented:** Write '0'; ignore read

Note 1: This bit can only be written when the ON bit = 0.

2: This bit can only be written when the ON bit = 0, and is only valid for AUDEN = 1.

3: When Audio mode is enabled (i.e., AUDEN = 1), the following bits in the SPIxCON register are configured by the module internally:
- The direction of the Bit Clock (BCLK) and Left/Right Channel Clock (LRCK) are selected based on the MSTEN bit
- FRMEN = 1, FRMCNT = 1, SMP = 0
- In Slave mode (MSTN = 0, FRMSYNC = 1) and in Master mode MSTN = 1, FRMSYNC = 0

4: In I^2S mode, SPIFE = 0, in Right or Left-Justified mode, SPIFE = 1, except in DSP/PCM mode when FRMSYPW = 0.

5: This feature is not available on all devices. Refer to the specific device data sheet for availability.

Register 23-2: SPIxCON2: SPI Control Register 2 (Continued)

bit 3 **AUDMONO:** Transmit Audio Data Format bit(2)
1 = Audio data is mono (Each data word is transmitted on both left and right channels)
0 = Audio data is stereo

bit 2 **Unimplemented:** Write '0'; ignore read

bit 1-0 **AUDMOD<1:0>:** Audio Protocol Mode bit(2,4)
11 = PCM/DSP mode
10 = Right-Justified mode
01 = Left-Justified mode
00 = I^2S mode(5)

Note 1: This bit can only be written when the ON bit = 0.

2: This bit can only be written when the ON bit = 0, and is only valid for AUDEN = 1.

3: When Audio mode is enabled (i.e., AUDEN = 1), the following bits in the SPIxCON register are configured by the module internally:
- The direction of the Bit Clock (BCLK) and Left/Right Channel Clock (LRCK) are selected based on the MSTEN bit
- FRMEN = 1, FRMCNT = 1, SMP = 0
- In Slave mode (MSTN = 0, FRMSYNC = 1) and in Master mode MSTN = 1, FRMSYNC = 0

4: In I^2S mode, SPIFE = 0, in Right or Left-Justified mode, SPIFE = 1, except in DSP/PCM mode when FRMSYPW = 0.

5: This feature is not available on all devices. Refer to the specific device data sheet for availability.

Register 23-3: SPIxSTAT: SPI Status Register

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	R-0	R-0	R-0	R-0	R-0
	—	—	—	RXBUFELM<4:0>(1)				
23:16	U-0	U-0	U-0	R-0	R-0	R-0	R-0	R-0
	—	—	—	TXBUFELM<4:0>(1)				
15:8	U-0	U-0	U-0	R/C-0, HS	R-0	U-0	U-0	R/C-0,HS
	—	—	—	FRMERR	SPIBUSY	—	—	SPITUR(1)
7:0	R-0	R/C-0,HS	R-0	U-0	R-1	U-0	R-0	R-0
	SRMT(12)	SPIROV	SPIRBE(1)	—	SPITBE	—	SPITBF(1)	SPIRBF

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

bit 31-29 **Unimplemented:** Write '0'; ignore read

bit 28-24 **RXBUFELM<4:0>:** Receive Buffer Element Count bits (valid only when ENHBUF = 1)(1)
Number of receive samples contained in the FIFO.

bit 23-21 **Unimplemented:** Write '0'; ignore read

bit 20-16 **TXBUFELM<4:0>:** Transmit Buffer Element Count bits (valid only when ENHBUF = 1)(1)
Number of transmit samples remaining in the FIFO.

bit 15-13 **Unimplemented:** Write '0'; ignore read

bit 12 **FRMERR:** SPI Frame Error status bit
1 = Frame error detected
0 = No Frame error detected
FRMERR is only valid when FRMEN = 1. This bit is only set by hardware. It can be cleared by writing a zero, preferably with the command SPIxSTATCLR = 1<<12. It can also be cleared by disabling and re-enabling the module using the SPIxCON.ON bit.

bit 11 **SPIBUSY:** SPI Activity Status bit
1 = SPI peripheral is currently busy with some transactions
0 = SPI peripheral is currently idle

bit 10-9 **Unimplemented:** Write '0'; ignore read

bit 8 **SPITUR:** Transmit Under Run bit(1)
1 = Transmit buffer has encountered an underrun condition
0 = Transmit buffer has no underrun condition
This bit is only valid in Framed Sync mode. This bit is only set by hardware. It can be cleared by writing a zero, preferably with the command SPIxSTATCLR = 1<<8. It can also be cleared by disabling and re-enabling the module using the SPIxCON.ON bit.

bit 7 **SRMT:** Shift Register Empty bit (valid only when ENHBUF = 1)(1)
1 = When SPI module shift register is empty
0 = When SPI module shift register is not empty

bit 6 **SPIROV:** Receive Overflow Flag bit
1 = A new data is completely received and discarded. The user software has not read the previous data in the SPIxBUF register.
0 = No overflow has occurred
This bit is only set by hardware. It can be cleared by writing a zero, preferably with the command SPIxSTATCLR = 1<<6. It can also be cleared by disabling and re-enabling the module using the SPIxCON.ON bit.

Note 1: These bits are not available on all devices. Refer to the specific device data sheet for availability.

Register 23-3: SPIxSTAT: SPI Status Register (Continued)

bit 5 **SPIRBE:** RX FIFO Empty bit (valid only when ENHBUF = 1)
1 = RX FIFO is empty (CRPTR = SWPTR)
0 = RX FIFO is not empty (CRPTR < SWPTR)

bit 4 **Unimplemented:** Write '0'; ignore read

bit 3 **SPITBE:** SPI Transmit Buffer Empty Status bit(1)
1 = Transmit buffer, SPIxTXB is empty
0 = Transmit buffer, SPIxTXB is not empty
Automatically set in hardware when SPI transfers data from SPIxTXB to SPIxSR.
Automatically cleared in hardware when SPIxBUF is written to, loading SPIxTXB.

bit 2 **Unimplemented:** Write '0'; ignore read

bit 1 **SPITBF:** SPI Transmit Buffer Full Status bit(1)
1 = Transmit not yet started, SPITXB is full
0 = Transmit buffer is not full
Standard Buffer Mode:
Automatically set in hardware when the core writes to the SPIBUF location, loading SPITXB.
Automatically cleared in hardware when the SPI module transfers data from SPITXB to SPISR.
Enhanced Buffer Mode:
Set when there is no available space in the FIFO.

bit 0 **SPIRBF:** SPI Receive Buffer Full Status bit
1 = Receive buffer, SPIxRXB is full
0 = Receive buffer, SPIxRXB is not full
Standard Buffer Mode:
Automatically set in hardware when the SPI module transfers data from SPIxSR to SPIxRXB.
Automatically cleared in hardware when SPIxBUF is read from, reading SPIxRXB.
Enhanced Buffer Mode:
Set when there is no available space in the FIFO.

Note 1: These bits are not available on all devices. Refer to the specific device data sheet for availability.

Register 23-4: SPIxBUF: SPI Buffer Register

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	DATA<31:24>							
23:16	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	DATA<23:16>							
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	DATA<15:8>							
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	DATA<7:0>							

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

bit 31-0 **DATA**<31:0>**:** SPI Transmit/Receive Buffer register

Serves as a memory-mapped value of Transmit (SPIxTXB) and Receive (SPIxRXB) registers.

When 32-bit Data mode is enabled (MODE<32,16> (SPIxCON<11:10>) = 1x):
All 32 bits (SPIxBUF<31:0>) of this register are used to form a 32-bit character.

When 16-bit Data mode is enabled (MODE<32,16> (SPIxCON<11:10>) = 01):
Only the lower 16 bits (SPIxBUF<15:0>) of this register are used to form the 16-bit character.

When 8-bit Data mode is enabled (MODE<32,16> (SPIxCON<11:10>) = 00):
Only the lower 8 bits (SPIxBUF<7:0>) of this register are used to form the 8-bit character.

Register 23-5: SPIxBRG: SPI Baud Rate Register

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
23:16	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
15:8	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	—	—	—	BRG<12:8>				
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	BRG<7:0>							

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

bit 31-13 **Unimplemented:** Write '0'; ignore read

bit 12-0 **BRG<12:0>:** Baud Rate Divisor bits

21.2 CONTROL REGISTERS

Note: Each PIC32 family device variant may have one or more UART modules. An 'x' used in the names of pins, Control/Status bits and registers denotes the particular module. Refer to the **"UART"** chapter in the specific device data sheet for more details.

Each UART module consists of the following Special Function Registers (SFRs):

- **UxMODE: UARTx Mode Register**

  This register does the following:

  - Enables or disables the UART module
  - Enables or disables the IrDA encoder and decoder
  - Enables or disables the WAKE, ABAUD and Loopback features
  - Enables or disables the $\overline{\text{UxRTS}}$ and $\overline{\text{UxCTS}}$ pins
  - Configures the $\overline{\text{UxRTS}}$ pin for the desired mode of operation
  - Configures the polarity of the UxRX pin
  - Selects the type of baud rate
  - Selects the number of data bits, parity and stop bits

Note: The UxRTS and UxCTS pins are not available on all devices. Refer to the **"Pin Diagrams"** section in the specific device data sheet to determine availability.

- **UxSTA: UARTx Status and Control Register**

  This register does the following:

  - Selects the Transmission Interrupt mode
  - Selects the Receive Interrupt mode
  - Enables or disables the UART transmission
  - Controls the Address Detect mode
  - Indicates various status conditions, such as transmit and receive buffer state, parity error, framing error and overflow error

- **UxTXREG: UARTx Transmit Register**

  This register provides the data to be transmitted.

- **UxRXREG: UARTx Receive Register**

  This register stores the received data.

- **UxBRG: UARTx Baud Rate Register**

  This register stores the baud rate value of the transmitted or received data.

Each UART module also has associated bits for interrupt control:

Note: Refer to the **"Interrupts Controller"** chapter in the specific device data sheet for availability and descriptions of these bits, and **Section 8. "Interrupts"** (DS61108) for additional information.

- Transmit Interrupt Enable Control bit (UxTXIE)
- Transmit Interrupt Flag Status bit (UXTXIF)
- Receive Interrupt Enable Control bit (UxRXIE)
- Receive Interrupt Flag Status bit (UxRXIF)
- Error Interrupt Enable Control bit (UxEIE)
- Error Interrupt Flag Status bit (UxEIF)
- Interrupt Priority Control bits (UxIP<2:0>)
- Interrupt Subpriority Control bits (UxIS<1:0>)

Table 21-1 summarizes all UART-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register bit.

Table 21-1: UART SFRs Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
UxMODE(1)	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ON	—	SIDL	IREN	RTSMD(2)	—	UEN<1:0>(2)	
	7:0	WAKE	LPBACK	ABAUD	RXINV	BRGH	PDSEL<1:0>		STSEL
UxSTA(1)	31:24	—	—	—	—	—	—	—	ADM_EN
	23:16	ADDR<7:0>							
	15:8	UTXISEL<1:0>		UTXINV	URXEN	UTXBRK	UTXEN	UTXBF	TRMT
	7:0	URXISEL<1:0>		ADDEN	RIDLE	PERR	FERR	OERR	URXDA
UxTXREG	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	TX<8>
	7:0	TX<7:0>							
UxRXREG	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	—	—	—	—	—	—	—	RX<8>
	7:0	RX<7:0>							
UxBRG(1)	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	BRG<15:8>							
	7:0	BRG<7:0>							

Note 1: These registers have an associated Clear register at an offset of 0x4, 0x8, and 0xC bytes, respectively. The Clear, Set, and Invert register shave the same name with CLR, SET, or INV appended to the register name (e.g., UxMODECLR). Writing a '1' to any bit position in these registers will clear, set, or invert valid bits in the associated register. Reads from these registers should be ignored.

2: These bits are not available in some UART modules. Refer to the **"UART"** chapter in the specific device data sheet for more information on availability of these bits in different UART modules.

Register 21-1: UxMODE: UARTx Mode Register

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
23:16	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
15:8	R/W-0	U-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
	ON(1)	—	SIDL	IREN	RTSMD(2)	—	UEN<1:0>(2)	
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	WAKE	LPBACK	ABAUD	RXINV	BRGH	PDSEL<1:0>		STSEL

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

bit 31-16 **Unimplemented:** Read as '0'

bit 15 **ON:** UARTx Enable bit(1)
- 1 = UARTx is enabled. UARTx pins are controlled by UARTx as defined by UEN<1:0> and UTXEN control bits
- 0 = UARTx is disabled. All UARTx pins are controlled by corresponding bits in the PORTx, TRISx and LATx registers; UARTx power consumption is minimal

bit 14 **Unimplemented:** Read as '0'

bit 13 **SIDL:** Stop in Idle Mode bit
- 1 = Discontinue operation when device enters Idle mode
- 0 = Continue operation in Idle mode

bit 12 **IREN:** IrDA® Encoder and Decoder Enable bit
- 1 = IrDA is enabled
- 0 = IrDA is disabled

bit 11 **RTSMD:** Mode Selection for $\overline{\text{UxRTS}}$ Pin bit(2)
- 1 = $\overline{\text{UxRTS}}$ pin is in Simplex mode
- 0 = $\overline{\text{UxRTS}}$ pin is in Flow Control mode

bit 10 **Unimplemented:** Read as '0'

bit 9-8 **UEN<1:0>:** UARTx Enable bits(2)
- 11 = UxTX, UxRX and UxBCLK pins are enabled and used; $\overline{\text{UxCTS}}$ pin is controlled by corresponding bits in the PORTx register
- 10 = UxTX, UxRX, $\overline{\text{UxCTS}}$ and $\overline{\text{UxRTS}}$ pins are enabled and used
- 01 = UxTX, UxRX and $\overline{\text{UxRTS}}$ pins are enabled and used; $\overline{\text{UxCTS}}$ pin is controlled by corresponding bits in the PORTx register
- 00 = UxTX and UxRX pins are enabled and used; $\overline{\text{UxCTS}}$ and $\overline{\text{UxRTS}}$/UxBCLK pins are controlled by corresponding bits in the PORTx register

bit 7 **WAKE:** Enable Wake-up on Start bit Detect During Sleep Mode bit
- 1 = Wake-up enabled
- 0 = Wake-up disabled

bit 6 **LPBACK:** UARTx Loopback Mode Select bit
- 1 = Loopback mode is enabled
- 0 = Loopback mode is disabled

Note 1: When using the 1:1 PBCLK divisor, the user software should not read/write the peripheral SFRs in the SYSCLK cycle immediately following the instruction that clears the module's ON bit.

2: These bits are not available in some UART modules. Refer to the **"UART"** chapter in the specific device data sheet for more information on availability of these bits in different UART modules.

Register 21-1: UxMODE: UARTx Mode Register (Continued)

bit 5 **ABAUD:** Auto-Baud Enable bit
1 = Enable baud rate measurement on the next character – requires reception of Sync character (0x55); cleared by hardware upon completion
0 = Baud rate measurement disabled or completed

bit 4 **RXINV:** Receive Polarity Inversion bit
1 = UxRX Idle state is '0'
0 = UxRX Idle state is '1'

bit 3 **BRGH:** High Baud Rate Enable bit
1 = High-Speed mode – 4x baud clock enabled
0 = Standard Speed mode – 16x baud clock enabled

bit 2-1 **PDSEL<1:0>:** Parity and Data Selection bits
11 = 9-bit data, no parity
10 = 8-bit data, odd parity
01 = 8-bit data, even parity
00 = 8-bit data, no parity

bit 0 **STSEL:** Stop Selection bit
1 = 2 Stop bits
0 = 1 Stop bit

Note 1: When using the 1:1 PBCLK divisor, the user software should not read/write the peripheral SFRs in the SYSCLK cycle immediately following the instruction that clears the module's ON bit.

2: These bits are not available in some UART modules. Refer to the **"UART"** chapter in the specific device data sheet for more information on availability of these bits in different UART modules.

Register 21-2: UxSTA: UARTx Status and Control Register

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/W-0
	—	—	—	—	—	—	—	ADM_EN
23:16	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	ADDR<7:0>							
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-1
	UTXISEL<1:0>(1)		UTXINV	URXEN	UTXBRK	UTXEN	UTXBF	TRMT
7:0	R/W-0	R/W-0	R/W-0	R-1	R-0	R-0	R/W-0	R-0
	URXISEL<1:0>(1)		ADDEN	RIDLE	PERR	FERR	OERR	URXDA

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

bit 31-25 **Unimplemented:** Read as '0'

bit 24 **ADM_EN:** Automatic Address Detect Mode Enable bit
- 1 = Automatic Address Detect mode is enabled
- 0 = Automatic Address Detect mode is disabled

bit 23-16 **ADDR<7:0>:** Automatic Address Mask bits

When the ADM_EN bit is '1', this value defines the address character to use for automatic address detection.

bit 15-14 **UTXISEL<1:0>:** TX Interrupt Mode Selection bits(1)

For 4-level deep FIFO UART modules:
- 11 = Reserved, do not use
- 10 = Interrupt is generated when the transmit buffer becomes empty
- 01 = Interrupt is generated when all characters have been transmitted
- 00 = Interrupt is generated when the transmit buffer contains at least one empty space

For 8-level deep FIFO UART modules:
- 11 = Reserved, do not use
- 10 = Interrupt is generated and asserted while the transmit buffer is empty
- 01 = Interrupt is generated and asserted when all characters have been transmitted
- 00 = Interrupt is generated and asserted while the transmit buffer contains at least one empty space

bit 13 **UTXINV:** Transmit Polarity Inversion bit

If IrDA mode is disabled (i.e., IREN (UxMODE<12>) is '0'):
- 1 = UxTX Idle state is '0'
- 0 = UxTX Idle state is '1'

If IrDA mode is enabled (i.e., IREN (UxMODE<12>) is '1'):
- 1 = IrDA encoded UxTX Idle state is '1'
- 0 = IrDA encoded UxTX Idle state is '0'

bit 12 **URXEN:** Receiver Enable bit
- 1 = UARTx receiver is enabled. UxRX pin is controlled by UARTx (if ON = 1)
- 0 = UARTx receiver is disabled. UxRX pin is ignored by the UARTx module. UxRX pin is controlled by port.

bit 11 **UTXBRK:** Transmit Break bit
- 1 = Send Break on next transmission. Start bit followed by twelve '0' bits, followed by Stop bit; cleared by hardware upon completion
- 0 = Break transmission is disabled or completed

Note 1: These bits have different functions based depending on the available UART module. Refer to the **"UART"** chapter in the specific device data sheet for availability and interrupt implementation.

Register 21-2: UxSTA: UARTx Status and Control Register (Continued)

bit 10 **UTXEN:** Transmit Enable bit
1 = UARTx transmitter is enabled. UxTX pin is controlled by UARTx (if ON = 1)
0 = UARTx transmitter is disabled. Any pending transmission is aborted and buffer is reset. UxTX pin is controlled by port.

bit 9 **UTXBF:** Transmit Buffer Full Status bit (read-only)
1 = Transmit buffer is full
0 = Transmit buffer is not full, at least one more character can be written

bit 8 **TRMT:** Transmit Shift Register is Empty bit (read-only)
1 = Transmit shift register is empty and transmit buffer is empty (the last transmission has completed)
0 = Transmit shift register is not empty, a transmission is in progress or queued in the transmit buffer

bit 7-6 **URXISEL<1:0>:** Receive Interrupt Mode Selection bit(1)

For 4-level deep FIFO UART modules:
11 = Interrupt flag bit is set when receive buffer becomes full (i.e., has 4 data characters)
10 = Interrupt flag bit is set when receive buffer becomes 3/4 full (i.e., has 3 data characters)
0x = Interrupt flag bit is set when a character is received

For 8-level deep FIFO UART modules:
11 = Reserved; do not use
10 = Interrupt flag bit is asserted while receive buffer is 3/4 or more full (i.e., has 6 or more data characters)
01 = Interrupt flag bit is asserted while receive buffer is 1/2 or more full (i.e., has 4 or more data characters)
00 = Interrupt flag bit is asserted while receive buffer is not empty (i.e., has at least 1 data character)

bit 5 **ADDEN:** Address Character Detect bit (bit 8 of received data = 1)
1 = Address Detect mode is enabled. If 9-bit mode is not selected, this control bit has no effect.
0 = Address Detect mode is disabled

bit 4 **RIDLE:** Receiver Idle bit (read-only)
1 = Receiver is Idle
0 = Data is being received

bit 3 **PERR:** Parity Error Status bit (read-only)
1 = Parity error has been detected for the current character
0 = Parity error has not been detected

bit 2 **FERR:** Framing Error Status bit (read-only)
1 = Framing error has been detected for the current character
0 = Framing error has not been detected

bit 1 **OERR:** Receive Buffer Overrun Error Status bit.
This bit is set in hardware and can only be cleared (= 0) in software. Clearing a previously set OERR bit resets the receiver buffer and RSR to empty state.
1 = Receive buffer has overflowed
0 = Receive buffer has not overflowed

bit 0 **URXDA:** Receive Buffer Data Available bit (read-only)
1 = Receive buffer has data, at least one more character can be read
0 = Receive buffer is empty

Note 1: These bits have different functions based depending on the available UART module. Refer to the **"UART"** chapter in the specific device data sheet for availability and interrupt implementation.

Register 21-3: UxTXREG: UARTx Transmit Register

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
23:16	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
15:8	U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/W-0
	—	—	—	—	—	—	—	TX<8>
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	TX<7:0>							

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

bit 31-9 **Unimplemented:** Read as '0'

bit 8-0 **TX<8:0>:** Data bits 8-0 of the character to be transmitted

Register 21-4: UxRXREG: UARTx Receive Register

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
23:16	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
15:8	U-0	U-0	U-0	U-0	U-0	U-0	U-0	R-0
	—	—	—	—	—	—	—	RX<8>
7:0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
	RX<7:0>							

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

bit 31-9 **Unimplemented:** Read as '0'

bit 8-0 **RX<8:0>:** Data bits 8-0 of the received character

Register 21-5: UxBRG: UARTx Baud Rate Register

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
23:16	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	BRG<15:8>							
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	BRG<7:0>							

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

bit 31-16 **Unimplemented:** Read as '0'

bit 15-0 **BRG<15:0>:** Baud Rate Divisor bits

7.0 INTERRUPT CONTROLLER

Note 1: This data sheet summarizes the features of the PIC32MX1XX/2XX family of devices. It is not intended to be a comprehensive reference source. To complement the information in this data sheet, refer to **Section 8. "Interrupt Controller"** (DS61108) in the *"PIC32 Family Reference Manual"*, which is available from the Microchip web site (www.microchip.com/PIC32).

2: Some registers and associated bits described in this section may not be available on all devices. Refer to **Section 4.0 "Memory Organization"** in this data sheet for device-specific register and bit information.

PIC32MX1XX/2XX devices generate interrupt requests in response to interrupt events from peripheral modules. The interrupt control module exists externally to the CPU logic and prioritizes the interrupt events before presenting them to the CPU.

The PIC32MX1XX/2XX interrupt module includes the following features:

- Up to 64 interrupt sources
- Up to 44 interrupt vectors
- Single and multi-vector mode operations
- Five external interrupts with edge polarity control
- Interrupt proximity timer
- Seven user-selectable priority levels for each vector
- Four user-selectable subpriority levels within each priority
- Software can generate any interrupt
- User-configurable interrupt vector table location
- User-configurable interrupt vector spacing

Note: On PIC32MX1XX/2XX devices, the dedicated shadow set is not present.

FIGURE 7-1: INTERRUPT CONTROLLER MODULE BLOCK DIAGRAM

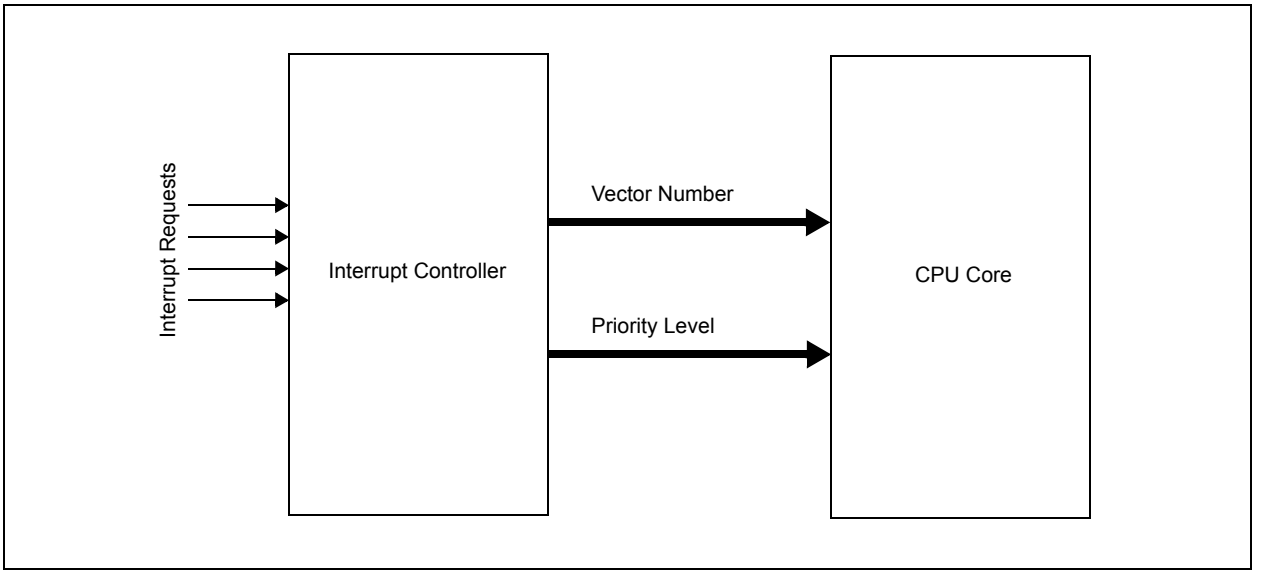


TABLE 7-1: INTERRUPT IRQ, VECTOR AND BIT LOCATION

Interrupt Source(1)	IRQ #	Vector #	Interrupt Bit Location				Persistent Interrupt
			Flag	Enable	Priority	Sub-priority	
Highest Natural Order Priority							
CT – Core Timer Interrupt	0	0	IFS0<0>	IEC0<0>	IPC0<4:2>	IPC0<1:0>	No
CS0 – Core Software Interrupt 0	1	1	IFS0<1>	IEC0<1>	IPC0<12:10>	IPC0<9:8>	No
CS1 – Core Software Interrupt 1	2	2	IFS0<2>	IEC0<2>	IPC0<20:18>	IPC0<17:16>	No
INT0 – External Interrupt	3	3	IFS0<3>	IEC0<3>	IPC0<28:26>	IPC0<25:24>	No
T1 – Timer1	4	4	IFS0<4>	IEC0<4>	IPC1<4:2>	IPC1<1:0>	No
IC1E – Input Capture 1 Error	5	5	IFS0<5>	IEC0<5>	IPC1<12:10>	IPC1<9:8>	Yes
IC1 – Input Capture 1	6	5	IFS0<6>	IEC0<6>	IPC1<12:10>	IPC1<9:8>	Yes
OC1 – Output Compare 1	7	6	IFS0<7>	IEC0<7>	IPC1<20:18>	IPC1<17:16>	No
INT1 – External Interrupt 1	8	7	IFS0<8>	IEC0<8>	IPC1<28:26>	IPC1<25:24>	No
T2 – Timer2	9	8	IFS0<9>	IEC0<9>	IPC2<4:2>	IPC2<1:0>	No
IC2E – Input Capture 2	10	9	IFS0<10>	IEC0<10>	IPC2<12:10>	IPC2<9:8>	Yes
IC2 – Input Capture 2	11	9	IFS0<11>	IEC0<11>	IPC2<12:10>	IPC2<9:8>	Yes
OC2 – Output Compare 2	12	10	IFS0<12>	IEC0<12>	IPC2<20:18>	IPC2<17:16>	No
INT2 – External Interrupt 2	13	11	IFS0<13>	IEC0<13>	IPC2<28:26>	IPC2<25:24>	No
T3 – Timer3	14	12	IFS0<14>	IEC0<14>	IPC3<4:2>	IPC3<1:0>	No
IC3E – Input Capture 3	15	13	IFS0<15>	IEC0<15>	IPC3<12:10>	IPC3<9:8>	Yes
IC3 – Input Capture 3	16	13	IFS0<16>	IEC0<16>	IPC3<12:10>	IPC3<9:8>	Yes
OC3 – Output Compare 3	17	14	IFS0<17>	IEC0<17>	IPC3<20:18>	IPC3<17:16>	No
INT3 – External Interrupt 3	18	15	IFS0<18>	IEC0<18>	IPC3<28:26>	IPC3<25:24>	No
T4 – Timer4	19	16	IFS0<19>	IEC0<19>	IPC4<4:2>	IPC4<1:0>	No
IC4E – Input Capture 4 Error	20	17	IFS0<20>	IEC0<20>	IPC4<12:10>	IPC4<9:8>	Yes
IC4 – Input Capture 4	21	17	IFS0<21>	IEC0<21>	IPC4<12:10>	IPC4<9:8>	Yes
OC4 – Output Compare 4	22	18	IFS0<22>	IEC0<22>	IPC4<20:18>	IPC4<17:16>	No
INT4 – External Interrupt 4	23	19	IFS0<23>	IEC0<23>	IPC4<28:26>	IPC4<25:24>	No
T5 – Timer5	24	20	IFS0<24>	IEC0<24>	IPC5<4:2>	IPC5<1:0>	No
IC5E – Input Capture 5 Error	25	21	IFS0<25>	IEC0<25>	IPC5<12:10>	IPC5<9:8>	Yes
IC5 – Input Capture 5	26	21	IFS0<26>	IEC0<26>	IPC5<12:10>	IPC5<9:8>	Yes
OC5 – Output Compare 5	27	22	IFS0<27>	IEC0<27>	IPC5<20:18>	IPC5<17:16>	No
AD1 – ADC1 Convert done	28	23	IFS0<28>	IEC0<28>	IPC5<28:26>	IPC5<25:24>	Yes
FSCM – Fail-Safe Clock Monitor	29	24	IFS0<29>	IEC0<29>	IPC6<4:2>	IPC6<1:0>	No
RTCC – Real-Time Clock and Calendar	30	25	IFS0<30>	IEC0<30>	IPC6<12:10>	IPC6<9:8>	No
FCE – Flash Control Event	31	26	IFS0<31>	IEC0<31>	IPC6<20:18>	IPC6<17:16>	No
CMP1 – Comparator Interrupt	32	27	IFS1<0>	IEC1<0>	IPC6<28:26>	IPC6<25:24>	No
CMP2 – Comparator Interrupt	33	28	IFS1<1>	IEC1<1>	IPC7<4:2>	IPC7<1:0>	No
CMP3 – Comparator Interrupt	34	29	IFS1<2>	IEC1<2>	IPC7<12:10>	IPC7<9:8>	No
USB – USB Interrupts	35	30	IFS1<3>	IEC1<3>	IPC7<20:18>	IPC7<17:16>	Yes
SPI1E – SPI1 Fault	36	31	IFS1<4>	IEC1<4>	IPC7<28:26>	IPC7<25:24>	Yes
SPI1RX – SPI1 Receive Done	37	31	IFS1<5>	IEC1<5>	IPC7<28:26>	IPC7<25:24>	Yes
SPI1TX – SPI1 Transfer Done	38	31	IFS1<6>	IEC1<6>	IPC7<28:26>	IPC7<25:24>	Yes

Note 1: Not all interrupt sources are available on all devices. See **TABLE 1: "PIC32MX1XX General Purpose Family Features"** and **TABLE 2: "PIC32MX2XX USB Family Features"** for the lists of available peripherals.

TABLE 7-1: INTERRUPT IRQ, VECTOR AND BIT LOCATION (CONTINUED)

Interrupt Source(1)	IRQ #	Vector #	Interrupt Bit Location				Persistent Interrupt
			Flag	Enable	Priority	Sub-priority	
U1E – UART1 Fault	39	32	IFS1<7>	IEC1<7>	IPC8<4:2>	IPC8<1:0>	Yes
U1RX – UART1 Receive Done	40	32	IFS1<8>	IEC1<8>	IPC8<4:2>	IPC8<1:0>	Yes
U1TX – UART1 Transfer Done	41	32	IFS1<9>	IEC1<9>	IPC8<4:2>	IPC8<1:0>	Yes
I2C1B – I2C1 Bus Collision Event	42	33	IFS1<10>	IEC1<10>	IPC8<12:10>	IPC8<9:8>	Yes
I2C1S – I2C1 Slave Event	43	33	IFS1<11>	IEC1<11>	IPC8<12:10>	IPC8<9:8>	Yes
I2C1M – I2C1 Master Event	44	33	IFS1<12>	IEC1<12>	IPC8<12:10>	IPC8<9:8>	Yes
CNA – PORTA Input Change Interrupt	45	34	IFS1<13>	IEC1<13>	IPC8<20:18>	IPC8<17:16>	Yes
CNB – PORTB Input Change Interrupt	46	34	IFS1<14>	IEC1<14>	IPC8<20:18>	IPC8<17:16>	Yes
CNC – PORTC Input Change Interrupt	47	34	IFS1<15>	IEC1<15>	IPC8<20:18>	IPC8<17:16>	Yes
PMP – Parallel Master Port	48	35	IFS1<16>	IEC1<16>	IPC8<28:26>	IPC8<25:24>	Yes
PMPE – Parallel Master Port Error	49	35	IFS1<17>	IEC1<17>	IPC8<28:26>	IPC8<25:24>	Yes
SPI2E – SPI2 Fault	50	36	IFS1<18>	IEC1<18>	IPC9<4:2>	IPC9<1:0>	Yes
SPI2RX – SPI2 Receive Done	51	36	IFS1<19>	IEC1<19>	IPC9<4:2>	IPC9<1:0>	Yes
SPI2TX – SPI2 Transfer Done	52	36	IFS1<20>	IEC1<20>	IPC9<4:2>	IPC9<1:0>	Yes
U2E – UART2 Error	53	37	IFS1<21>	IEC1<21>	IPC9<12:10>	IPC9<9:8>	Yes
U2RX – UART2 Receiver	54	37	IFS1<22>	IEC1<22>	IPC9<12:10>	IPC9<9:8>	Yes
U2TX – UART2 Transmitter	55	37	IFS1<23>	IEC1<23>	IPC9<12:10>	IPC9<9:8>	Yes
I2C2B – I2C2 Bus Collision Event	56	38	IFS1<24>	IEC1<24>	IPC9<20:18>	IPC9<17:16>	Yes
I2C2S – I2C2 Slave Event	57	38	IFS1<25>	IEC1<25>	IPC9<20:18>	IPC9<17:16>	Yes
I2C2M – I2C2 Master Event	58	38	IFS1<26>	IEC1<26>	IPC9<20:18>	IPC9<17:16>	Yes
CTMU – CTMU Event	59	39	IFS1<27>	IEC1<27>	IPC9<28:26>	IPC9<25:24>	Yes
DMA0 – DMA Channel 0	60	40	IFS1<28>	IEC1<28>	IPC10<4:2>	IPC10<1:0>	No
DMA1 – DMA Channel 1	61	41	IFS1<29>	IEC1<29>	IPC10<12:10>	IPC10<9:8>	No
DMA2 – DMA Channel 2	62	42	IFS1<30>	IEC1<30>	IPC10<20:18>	IPC10<17:16>	No
DMA3 – DMA Channel 3	63	43	IFS1<31>	IEC1<31>	IPC10<28:26>	IPC10<25:24>	No
Lowest Natural Order Priority							

Note 1: Not all interrupt sources are available on all devices. See **TABLE 1: "PIC32MX1XX General Purpose Family Features"** and **TABLE 2: "PIC32MX2XX USB Family Features"** for the lists of available peripherals.

REGISTER 7-1: INTCON: INTERRUPT CONTROL REGISTER

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
23:16	U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/W-0
	—	—	—	—	—	—	—	SS0
15:8	U-0	U-0	U-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0
	—	—	—	MVEC	—	TPC<2:0>		
7:0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	—	—	—	INT4EP	INT3EP	INT2EP	INT1EP	INT0EP

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

bit 31-17 **Unimplemented:** Read as '0'

bit 16 **SS0:** Single Vector Shadow Register Set bit
1 = Single vector is presented with a shadow register set
0 = Single vector is not presented with a shadow register set

bit 15-13 **Unimplemented:** Read as '0'

bit 12 **MVEC:** Multi Vector Configuration bit
1 = Interrupt controller configured for multi vectored mode
0 = Interrupt controller configured for single vectored mode

bit 11 **Unimplemented:** Read as '0'

bit 10-8 **TPC<2:0>:** Interrupt Proximity Timer Control bits
111 = Interrupts of group priority 7 or lower start the Interrupt Proximity timer
110 = Interrupts of group priority 6 or lower start the Interrupt Proximity timer
101 = Interrupts of group priority 5 or lower start the Interrupt Proximity timer
100 = Interrupts of group priority 4 or lower start the Interrupt Proximity timer
011 = Interrupts of group priority 3 or lower start the Interrupt Proximity timer
010 = Interrupts of group priority 2 or lower start the Interrupt Proximity timer
001 = Interrupts of group priority 1 start the Interrupt Proximity timer
000 = Disables Interrupt Proximity timer

bit 7-5 **Unimplemented:** Read as '0'

bit 4 **INT4EP:** External Interrupt 4 Edge Polarity Control bit
1 = Rising edge
0 = Falling edge

bit 3 **INT3EP:** External Interrupt 3 Edge Polarity Control bit
1 = Rising edge
0 = Falling edge

bit 2 **INT2EP:** External Interrupt 2 Edge Polarity Control bit
1 = Rising edge
0 = Falling edge

bit 1 **INT1EP:** External Interrupt 1 Edge Polarity Control bit
1 = Rising edge
0 = Falling edge

bit 0 **INT0EP:** External Interrupt 0 Edge Polarity Control bit
1 = Rising edge
0 = Falling edge

REGISTER 7-2: INTSTAT: INTERRUPT STATUS REGISTER

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
23:16	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
15:8	U-0	U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0
	—	—	—	—	—	SRIPL<2:0>(1)		
7:0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	—	—	VEC<5:0>(1)					

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

bit 31-11 **Unimplemented:** Read as '0'

bit 10-8 **SRIPL<2:0>:** Requested Priority Level bits(1)
000-111 = The priority level of the latest interrupt presented to the CPU

bit 7-6 **Unimplemented:** Read as '0'

bit 5-0 **VEC<5:0>:** Interrupt Vector bits(1)
00000-11111 = The interrupt vector that is presented to the CPU

Note 1: This value should only be used when the interrupt controller is configured for Single Vector mode.

REGISTER 7-3: IPTMR: INTERRUPT PROXIMITY TIMER REGISTER

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	IPTMR<31:24>							
23:16	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	IPTMR<23:16>							
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	IPTMR<15:8>							
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	IPTMR<7:0>							

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

bit 31-0 **IPTMR<31:0>:** Interrupt Proximity Timer Reload bits
Used by the Interrupt Proximity Timer as a reload value when the Interrupt Proximity timer is triggered by an interrupt event.

REGISTER 7-4: IFSx: INTERRUPT FLAG STATUS REGISTER

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	IFS31	IFS30	IFS29	IFS28	IFS27	IFS26	IFS25	IFS24
23:16	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	IFS23	IFS22	IFS21	IFS20	IFS19	IFS18	IFS17	IFS16
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	IFS15	IFS14	IFS13	IFS12	IFS11	IFS10	IFS09	IFS08
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	IFS07	IFS06	IFS05	IFS04	IFS03	IFS02	IFS01	IFS00

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

bit 31-0 **IFS31-IFS00:** Interrupt Flag Status bits
1 = Interrupt request has occurred
0 = No interrupt request has occurred

Note: This register represents a generic definition of the IFSx register. Refer to Table 7-1 for the exact bit definitions.

REGISTER 7-5: IECx: INTERRUPT ENABLE CONTROL REGISTER

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	IEC31	IEC30	IEC29	IEC28	IEC27	IEC26	IEC25	IEC24
23:16	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	IEC23	IEC22	IEC21	IEC20	IEC19	IEC18	IEC17	IEC16
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	IEC15	IEC14	IEC13	IEC12	IEC11	IEC10	IEC09	IEC08
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	IEC07	IEC06	IEC05	IEC04	IEC03	IEC02	IEC01	IEC00

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

bit 31-0 **IEC31-IEC00:** Interrupt Enable bits
1 = Interrupt is enabled
0 = Interrupt is disabled

Note: This register represents a generic definition of the IECx register. Refer to Table 7-1 for the exact bit definitions.

REGISTER 7-6: IPCx: INTERRUPT PRIORITY CONTROL REGISTER

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	—	—	—	IP03<2:0>			IS03<1:0>	
23:16	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	—	—	—	IP02<2:0>			IS02<1:0>	
15:8	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	—	—	—	IP01<2:0>			IS01<1:0>	
7:0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	—	—	—	IP00<2:0>			IS00<1:0>	

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

bit 31-29 **Unimplemented:** Read as '0'

bit 28-26 **IP03<2:0>:** Interrupt Priority bits
- 111 = Interrupt priority is 7
- •
- •
- •
- 010 = Interrupt priority is 2
- 001 = Interrupt priority is 1
- 000 = Interrupt is disabled

bit 25-24 **IS03<1:0>:** Interrupt Subpriority bits
- 11 = Interrupt subpriority is 3
- 10 = Interrupt subpriority is 2
- 01 = Interrupt subpriority is 1
- 00 = Interrupt subpiority is 0

bit 23-21 **Unimplemented:** Read as '0'

bit 20-18 **IP02<2:0>:** Interrupt Priority bits
- 111 = Interrupt priority is 7
- •
- •
- •
- 010 = Interrupt priority is 2
- 001 = Interrupt priority is 1
- 000 = Interrupt is disabled

bit 17-16 **IS02<1:0>:** Interrupt Subpriority bits
- 11 = Interrupt subpriority is 3
- 10 = Interrupt subpriority is 2
- 01 = Interrupt subpriority is 1
- 00 = Interrupt subpriority is 0

bit 15-13 **Unimplemented:** Read as '0'

bit 12-10 **IP01<2:0>:** Interrupt Priority bits
- 111 = Interrupt priority is 7
- •
- •
- •
- 010 = Interrupt priority is 2
- 001 = Interrupt priority is 1
- 000 = Interrupt is disabled

Note: This register represents a generic definition of the IPCx register. Refer to Table 7-1 for the exact bit definitions.

REGISTER 7-6: IPCx: INTERRUPT PRIORITY CONTROL REGISTER (CONTINUED)

bit 9-8 **IS01<1:0>:** Interrupt Subpriority bits
11 = Interrupt subpriority is 3
10 = Interrupt subpriority is 2
01 = Interrupt subpriority is 1
00 = Interrupt subpriority is 0

bit 7-5 **Unimplemented:** Read as '0'

bit 4-2 **IP00<2:0>:** Interrupt Priority bits
111 = Interrupt priority is 7
•
•
•
010 = Interrupt priority is 2
001 = Interrupt priority is 1
000 = Interrupt is disabled

bit 1-0 **IS00<1:0>:** Interrupt Subpriority bits
11 = Interrupt subpriority is 3
10 = Interrupt subpriority is 2
01 = Interrupt subpriority is 1
00 = Interrupt subpriority is 0

Note:	This register represents a generic definition of the IPCx register. Refer to Table 7-1 for the exact bit definitions.

14.2 CONTROL REGISTERS

Note: Each PIC32 family device may have one or more timer modules. An 'x' used in the names of pins, control/status bits and registers denotes the particular module. For more information, refer to the specific device data sheet.

Each Timer module is a 16-bit timer/counter that consists of the following Special Function Registers (SFRs), which are summarized in Table 14-2:

- **T1CON: Type A Timer Control Register**
- **TxCON: Type B Timer Control Register**
- **TMRx: Timer Register**
- **PRx: Period Register**

Each Timer module also has the following associated bits for interrupt control:

- TxIE: Interrupt Enable Control bit in IEC0 interrupt register
- TxIF: Interrupt Flag Status bit in IFS0 interrupt register
- TxIP<2:0>: Interrupt Priority Control bits in IPC1, IPC2, IPC3, IPC4, and IPC5 interrupt registers
- TxIS<1:0>: Interrupt Subpriority Control bits in IPC1, IPC2, IPC3, IPC4, and IPC5 interrupt registers

Note: Refer to **Section 8. "Interrupts"** (DS61108) in the *"PIC32 Family Reference Manual"* for more information on these registers.

Table 14-2: Timers SFR Summary

Register Name(1)	Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
T1CON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ON	—	SIDL	TWDIS	TWIP	—	—	—
	7:0	TGATE	—	TCKPS<1:0>		—	TSYNC	TCS	—
TxCON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ON	—	SIDL	—	—	—	—	—
	7:0	TGATE	TCKPS<2:0>(2)			T32(3)	—	TCS	—
TMRx	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	TMRx<15:8>							
	7:0	TMRx<7:0>							
PRx	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	PRx<15:8>							
	7:0	PRx<7:0>							

Note 1: All registers have an associated Clear, Set, and Invert register at an offset of 0x4, 0x8, and 0xC bytes, respectively. These registers have the same name with CLR, SET, or INV appended to the end of the register name (e.g., T1CONCLR). Writing a '1' to any bit position in these registers will clear, set, or invert valid bits in the associated register. Reads from these registers should be ignored.

2: The TCKPS<2:0> bits are available only on even numbered Type B timers. For example, Timer2 and Timer4 in 32-bit Timer mode.

3: The T32 bit is available only on even numbered Type B timers, such as Timer2, Timer4, and so on.

Register 14-1: T1CON: Type A Timer Control Register

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
23:16	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
15:8	R/W-0	U-0	R/W-0	R/W-0	R-0	U-0	U-0	U-0
	ON(1)	—	SIDL	TWDIS	TWIP	—	—	—
7:0	R/W-0	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	U-0
	TGATE	—	TCKPS<1:0>		—	TSYNC	TCS	—

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

bit 31-16 **Unimplemented:** Read as '0'

bit 15 **ON:** Timer On bit(1)
- 1 = Timer is enabled
- 0 = Timer is disabled

bit 14 **Unimplemented:** Read as '0'

bit 13 **SIDL:** Stop in Idle Mode bit
- 1 = Discontinue operation when device enters Idle mode
- 0 = Continue operation when device enters Idle mode

bit 12 **TWDIS:** Asynchronous Timer Write Disable bit
- 1 = Writes to TMR1 are ignored until pending write operation completes
- 0 = Back-to-back writes are enabled (Legacy Asynchronous Timer functionality)

bit 11 **TWIP:** Asynchronous Timer Write in Progress bit

In Asynchronous Timer mode:
- 1 = Asynchronous write to TMR1 register in progress
- 0 = Asynchronous write to TMR1 register complete

In Synchronous Timer mode:
This bit is read as '0'.

bit 10-8 **Unimplemented:** Read as '0'

bit 7 **TGATE:** Timer Gated Time Accumulation Enable bit

When TCS = 1:
This bit is ignored.

When TCS = 0:
- 1 = Gated time accumulation is enabled
- 0 = Gated time accumulation is disabled

bit 6 **Unimplemented:** Read as '0'

bit 5-4 **TCKPS<1:0>:** Timer Input Clock Prescale Select bits
- 11 = 1:256 prescale value
- 10 = 1:64 prescale value
- 01 = 1:8 prescale value
- 00 = 1:1 prescale value

bit 3 **Unimplemented:** Read as '0'

Note 1: When using 1:1 PBCLK divisor, the user's software should not read/write the peripheral SFRs in the SYSCLK cycle immediately following the instruction that clears the module's ON bit.

Register 14-1: T1CON: Type A Timer Control Register (Continued)

bit 2 **TSYNC:** Timer External Clock Input Synchronization Selection bit

When TCS = 1:
1 = External clock input is synchronized
0 = External clock input is not synchronized

When TCS = 0:
This bit is ignored.

bit 1 **TCS:** Timer Clock Source Select bit

1 = External clock from TxCKI pin
0 = Internal peripheral clock

bit 0 **Unimplemented:** Read as '0'

Note 1: When using 1:1 PBCLK divisor, the user's software should not read/write the peripheral SFRs in the SYSCLK cycle immediately following the instruction that clears the module's ON bit.

Register 14-2: TxCON: Type B Timer Control Register

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
23:16	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
15:8	R/W-0	U-0	R/W-0	U-0	U-0	U-0	U-0	U-0
	ON(1)	—	SIDL(2)	—	—	—	—	—
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	U-0
	TGATE	TCKPS<2:0>			T32(3)	—	TCS(4)	—

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

bit 31-16 **Unimplemented:** Read as '0'

bit 15 **ON:** Timer On bit(1)
1 = Module is enabled
0 = Module is disabled

bit 14 **Unimplemented:** Read as '0'

bit 13 **SIDL:** Stop in Idle Mode bit(2)
1 = Discontinue operation when device enters Idle mode
0 = Continue operation when device enters Idle mode

bit 12-8 **Unimplemented:** Read as '0'

bit 7 **TGATE:** Timer Gated Time Accumulation Enable bit
When TCS = 1:
This bit is ignored and is read as '0'.

When TCS = 0:
1 = Gated time accumulation is enabled
0 = Gated time accumulation is disabled

bit 6-4 **TCKPS<2:0>:** Timer Input Clock Prescale Select bits
111 = 1:256 prescale value
110 = 1:64 prescale value
101 = 1:32 prescale value
100 = 1:16 prescale value
011 = 1:8 prescale value
010 = 1:4 prescale value
001 = 1:2 prescale value
000 = 1:1 prescale value

bit 3 **T32:** 32-bit Timer Mode Select bit(3)
1 = TMRx and TMRy form a 32-bit timer
0 = TMRx and TMRy form separate 16-bit timer

bit 2 **Unimplemented:** Read as '0'

bit 1 **TCS:** Timer Clock Source Select bit(4)
1 = External clock from TxCK pin
0 = Internal peripheral clock

bit 0 **Unimplemented:** Read as '0'

Note 1: When using 1:1 PBCLK divisor, the user's software should not read/write the peripheral SFRs in the SYSCLK cycle immediately following the instruction that clears the module's ON bit.

2: While operating in 32-bit mode, the SIDL bit (TxCON<13>) of consecutive odd number timers of the 32-bit timer pair has an affect on the timer operation. All other bits in this register have no affect.

3: The T32 bit is available only on even numbered Type B timers, such as Timer2, Timer4, and so on.

4: The TxCK pin is not available on all timers. Refer to the **"Timers"** chapters in the specific device data sheet for availability.

Register 14-3: TMRx: Timer Register

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
23:16	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	TMR<15:8>							
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	TMR<7:0>							

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

bit 31-16 **Unimplemented:** Read as '0'

bit 15-0 **TMR<15:0>:** Timer Count Register bits

16-bit mode:

These bits represent the complete 16-bit timer count.

32-bit mode (Type B Timer only):

Timer2 and Timer4: These bits represent the least significant half word (16 bits) of the 32-bit timer count.

Timer3 and Timer5: These bits represent the most significant half word (16 bits) of the 32-bit timer count.

Register 14-4: PRx: Period Register

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
23:16	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
15:8	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
	PR<15:8>							
7:0	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
	PR<7:0>							

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

bit 31-16 **Unimplemented:** Read as '0'

bit 15-0 **PR<15:0>:** Period Register bits

16-bit mode:

These bits represent the complete 16-bit period match.

32-bit mode (Type B Timer only):

Timer2 and Timer4: These bits represent the least significant half word (16 bits) of the 32-bit period match.

Timer3 and Timer5: These bits represent the most significant half word (16 bits) of the 32-bit period match.