UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Departamento de Ingeniería Eléctrica

PROYECTO FIN DE CARRERA

INGENIERÍA INDUSTRIAL

Sistema de adquisición de datos mediante Waspmote

AUTOR: Joaquín Cruz Isabel

DIRECTOR: Guillermo Robles Muñoz

Leganés, 10 de Abril de 2012

Gracias a mi familia y amigos por su apoyo incondicional.

Dedicado con especial cariño a mi abuelo,
José Antonio Isabel Gómez.

Índice general

Lista de figuras **3**

1. Introducción **4**

2. Objetivos **6**

3. Señal modelo **7**
3.1. Síntesis de la señal modelo 7
3.2. Acondicionamiento de la señal 9
3.2.1. Circuito detector de envolvente 10
3.2.2. Simulación en Multisim 12
3.2.3. Circuito real 14

4. Adquisición de datos, Waspmote **16**
4.1. Introducción 16
4.1.1. Características Generales: 16
4.2. Código de la aplicación en Waspmote 19
4.2.1. Diagrama del código 19
4.2.2. Configuración de registros 21
4.2.3. AnalogReadDif 23
4.2.4. EstablishContact 25
4.2.5. *Trigger* y número de muestras 26

5. Aplicación desarrollada en Matlab **28**
5.1. Introducción 28
5.2. Detector de envolvente software 28
5.3. Comunicacion USB 29
5.4. Adquisición de datos 30
5.5. Aplicación de usuario 31
5.6. Diagrama de bloques 33

6. Ampliaciones y mejoras **39**
6.1. Almacenamiento en tarjeta SD 39
6.1.1. Herramientas de ayuda en la programación 39
6.1.2. Inicialización y finalización 40
6.1.3. Información complementaria 41

6.1.4. Directorios 42
6.1.5. Operaciones con archivos 44
6.2. XBee 802.15.4 47

7. Conclusiones **49**

Índice de figuras

1.1. Diagrama de bloques del proyecto. 5

3.1. Señal con 5000 muestras. 8
3.2. Señal recortada con 1000 muestras. 8
3.3. Señal introducida. 10
3.4. Circuito detector de envolvente. 11
3.5. Montaje en laboratorio 11
3.6. Circuito detector de envolvente simulado. 13
3.7. Resultado de la simulación. 13
3.8. Resultado final del filtro de envolvente. 15

4.1. Diagrama de programación Waspmote. 20
4.2. Detalle del registro ADMUX. 21
4.3. Tabla de tensiones de referencia. 22
4.4. Lectura diferencial en diferentes canales. 22
4.5. Salidas y entradas disponibles en Waspmote. 23
4.6. Detalle del registro ADCSRA. 23
4.7. Lectura de datos de los registros ADCL y ADCH 24

5.1. Detector de envolvente software [9]. 29
5.2. Panel de elementos. 31
5.3. Icono de acceso al archivo .m 32
5.4. Diagrama de bloques de la aplicación de usuario. 33
5.5. Aplicación en el estado inicial. 35
5.6. Aplicación dibujando la señal original. 36
5.7. Aplicación dibujando la señal original junto con el detector de envolvente software. 37

6.1. Módulo XBEE 802.15.4 47

7.1. Señal modelo. 51

Capítulo 1

Introducción

En este proyecto se pretende diseñar un sistema de adquisición de señales procedentes de sensores de una manera sencilla y eficaz atendiendo a criterios económicos. Este sistema debe ser capaz de acondicionar esa señal para adquirirla y enviarla a un ordenador para ser analizada.

Para facilitar la tarea de desarrollo del proyecto se ha optado por realizar una toma de datos de una señal modelo sintetizada de una señal real con un tamaño máximo de 1000 puntos. Esta señal se introduce en un generador de ondas TEKTRONIX AFG3252 del laboratorio de Alta Tensión del Departamento de Ingeniería Eléctrica de la Universidad Carlos III de Madrid con la ayuda de una memoria USB. De esta manera, se tiene una señal modelo que se puede modificar en amplitud y frecuencia. Los datos de la señal modelo se pueden encontrar representados en el Anexo 1. La principal ventaja que se obtiene es que se pueden realizar estudios basados en la repetibilidad de la señal. Teniendo esto en cuenta, podremos averiguar si nuestro dispositivo es coherente en la toma de datos. Dado que se tiene una señal perfectamente definida, es posible comparar la señal que se ha enviado desde el generador de ondas con la que se recibe finalmente en Matlab, esto permite realizar un estudio empírico de la fiabilidad que se tiene en la toma de datos.

Una vez que la señal de entrada ha sido definida, el objetivo será adquirirla mediante un dispositivo hardware que permita la transmisión de los datos a un ordenador. Si bien se puede considerar que una mayor frecuencia de muestreo conlleva la obtención de mayor cantidad de información sobre la señal, este hecho contrasta con el aumento de precio que supone usar dispositivos con una capacidad de muestreo más elevada. Para la aplicación de este sistema de adquisición no es necesario tomar todos los puntos de la señal, basta con conocer el contorno que tiene. Siendo esto así, se establece que la información que nos resulta útil es la envolvente de la señal, dado que contiene información sobre su duración y los máximos valores alcanzados.

Para cumplir el objetivo marcado se han llevado a cabo tres bloques claramente diferenciables que serán explicados en capítulos posteriores a éste; bloque detector de envolvente, bloque del dispositivo hardware de adquisición (Waspmote) y bloque de la aplicación de usuario.

A continuación, ver Figura 1.1, mostramos un diagrama que resume el proyecto reduciéndolo a varios bloques básicos.

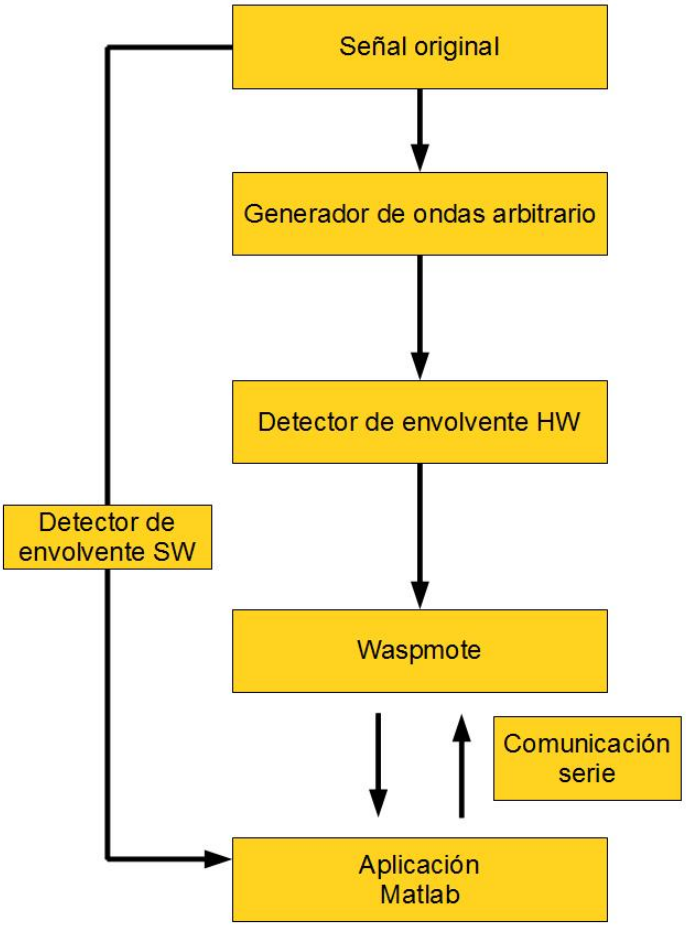

Figura 1.1: Diagrama de bloques del proyecto.

Capítulo 2

Objetivos

Vamos a realizar un listado de los objetivos que nos hemos planteado en este proyecto.

- Poder recoger medidas de una señal modelo con las siguientes restricciones:
  - Presenta unas oscilaciones de aproximadamente 5 μs de periodo.
  - Contaremos con un microprocesador de bajo coste.
- Hacer el diseño del sistema de adquisición basado en un dispositivo con las siguientes características:
  - Dispositivo de carácter modular para realizar posibles ampliaciones futuras.
  - Velocidad de muestreo suficiente para nuestros objetivos.
  - Repetibilidad en la toma de medidas de las señales.
  - Uso de un dispositivo fácilmente programable, preferiblemente en C.
  - Posibilidad de establecer una red de dispositivos que se comuniquen entre ellos.
  - Posibilidad de procesar los datos con Matlab.
  - Pequeño tamaño del dispositivo.
  - Memoria suficiente para guardar diferentes muestras si se desease.
  - Dispositivo portable.
  - Bajo consumo, alta autonomía.
- Creación de un código cuyo tiempo de adquisición sea estable para que la toma de datos sea fiable.
- Garantizar la repetibilidad en la toma de medidas de las señales.

Capítulo 3

Señal modelo

3.1. Síntesis de la señal modelo

Con el objetivo de conseguir una señal modelo que poder usar para realizar estudios posteriores hemos tenido que realizar varias acciones. En primer lugar, a través de un osciloscopio TEKTRONIX MDO4054-3 capaz de muestrear señales de alta frecuencia, obtenemos una señal de 5000 muestras procedente de la lectura que está realizando un sensor en el laboratorio.

Esos datos se introducen en Matlab y hay que acondicionarlos antes de introducirlos en el generador de ondas TEKTRONIX AFG3252 del laboratorio. Este generador de ondas sólo admite señales con un número máximo de 1000 muestras, por lo que se cogerá solo la parte de la señal que contiene información válida. De la señal de la Figura 3.1, se escoge 1 ms en la zona comprendida entre 950 ms y 1050, lo que corresponde a 1000 puntos de los 5000 totales, y se representa en la Figura 3.2.

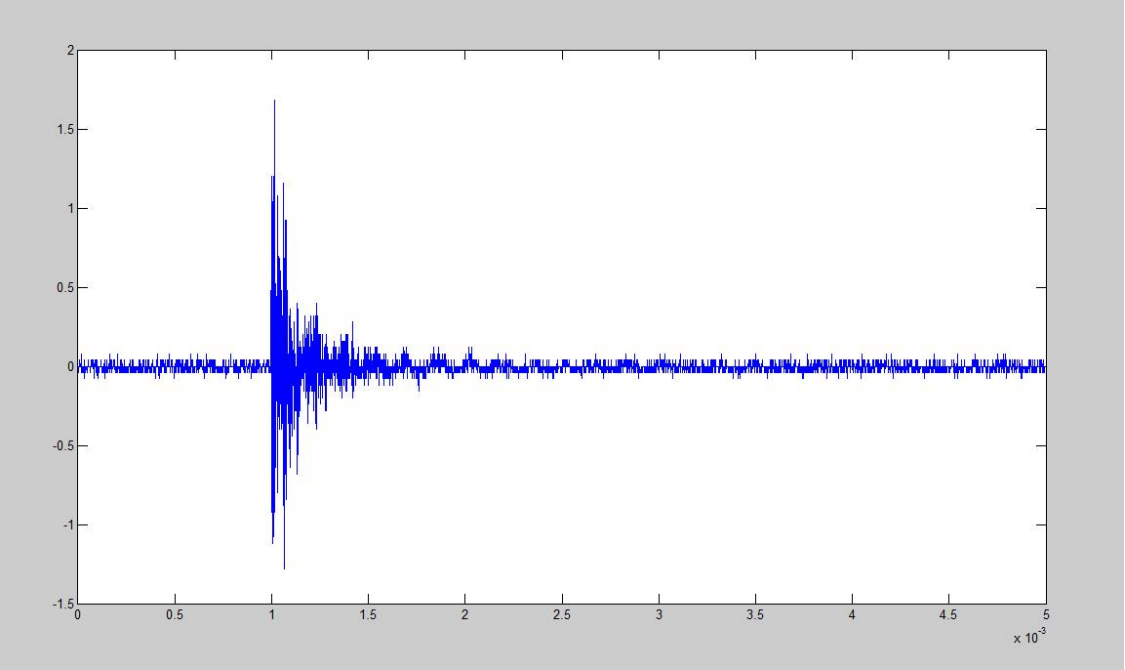

Figura 3.1: Señal con 5000 muestras.

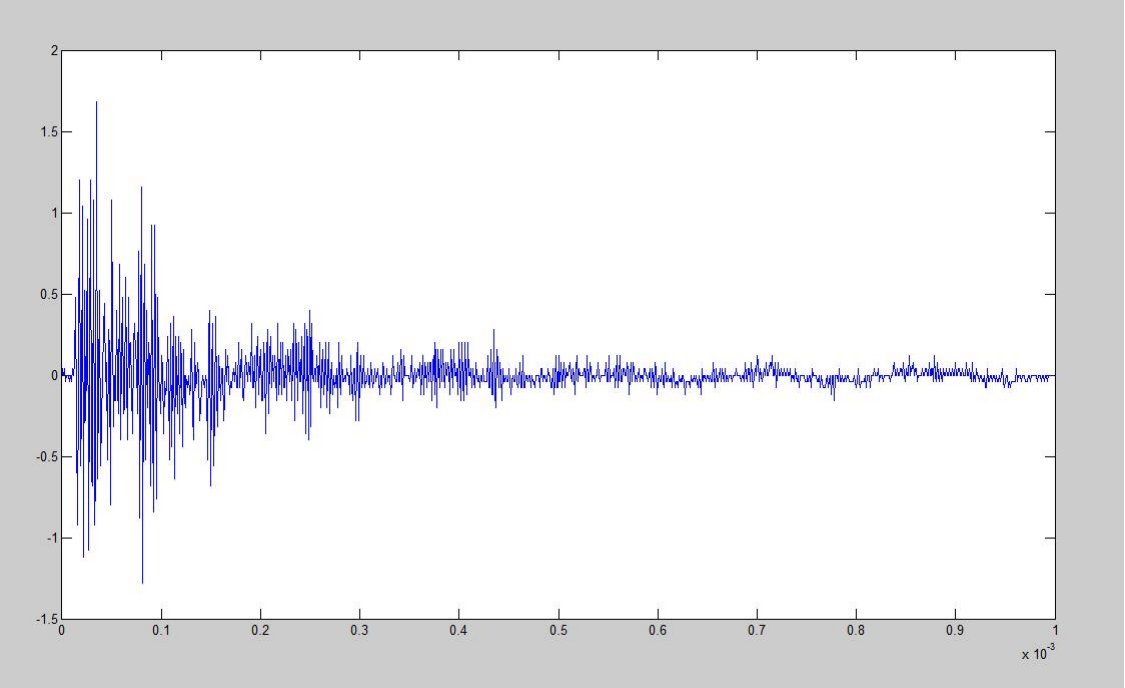

Figura 3.2: Señal recortada con 1000 muestras.

Esta última señal se guarda en una memoria USB en formato ASCII y se introduce en el generador de ondas. Para introducirla se debe seleccionar en el generador la función de generación de ondas arbitrarias y seleccionar el archivo donde se encuentra la matriz con

los datos en dos columnas, una con los datos temporales y otra con los de tensión. Una vez ha sido introducida, podemos elegir entre las opciones: *continuous*, *modulation*, *sweep* o *burst*. En nuestro caso seleccionaremos una continuidad de la señal tipo *burst*. Este modo de reproducción de la señal permite repetir un patrón un número determinado de veces cada cierto tiempo. Tanto la amplitud de la señal como el ancho de los pulsos es seleccionable, gracias a toda esta variedad de opciones se podrá configurar una onda que sera nuestro objeto de estudio. En nuestro caso se reproducirá una señal o un patrón con un intervalo de 10 ms entre ellos. En la Figura 3.2 se observa que la señal tiene una duración completa de 1 ms, siendo hasta 0,3 ms el periodo de mayor actividad. Se caracteriza por unas rápidas oscilaciones que suelen estar entorno a los 5 μs, con un tiempo de subida máximo de 2,16 V/μs y un tiempo de bajada máximo de 1,12 V/μs.

Si bien se pierden las muestras fuera del intervalo que se ha escogido se descartan para el estudio porque no son relevantes. Cabe indicar que esta señal tan solo es un modelo de un conjunto de señales con características similares pero no idénticas, es por ello que siempre debemos tener presente las pequeñas variaciones que pudiesen darse entre diferentes señales.

Para realizar variaciones en la amplitud de la señal y en la frecuencia de la misma hay que seleccionar *amplitude* y *frequency*, respectivamente, en el generador de ondas arbitrario. Curiosamente se observa que hay una variación de la posición del '0' de la señal si modificamos la amplitud, para corregirla se deberá hacer uso del ajuste de *offset*, subiendo o bajando la señal. Estas modificaciones de amplitud y frecuencia, son especialmente útiles para verificar el comportamiento que tendría el sistema ante señales que fuesen ligeramente diferentes.

3.2. Acondicionamiento de la señal

Como ya se ha definido, debemos ser capaces de tomar datos de una señal muy oscilante. La información que se necesita de la señal se limita a los niveles máximos de tensión que se alcanzan y la duración de estos niveles del pulso, con ello conseguimos la parametrización de la señal. En principio hay dos caminos que se pueden seguir para lograr este objetivo, uno es mediante la adquisición con un hardware sofisticado y el posterior tratamiento software de la señal utilizando un programa como Matlab que ayude a extraer los dos parámetros significativos: duración y valores máximos. El otro camino consiste en la introducción de un elemento hardware, un detector de envolvente que modifique la señal para poder ser adquirida con un hardware más sencillo y económico. Como ya se explicará en el capítulo siguiente, la velocidad del microprocesador y de muestreo están muy relacionadas con el precio del mismo, esto se traduce en que debemos usar la segunda opción, el uso de un dispositivo hardware que no supone una variación apreciable del precio del conjunto. Una vez se ha conectado todo el circuito la señal de entrada del mismo se representa de esta manera en el osciloscopio, ver Figura 3.3.

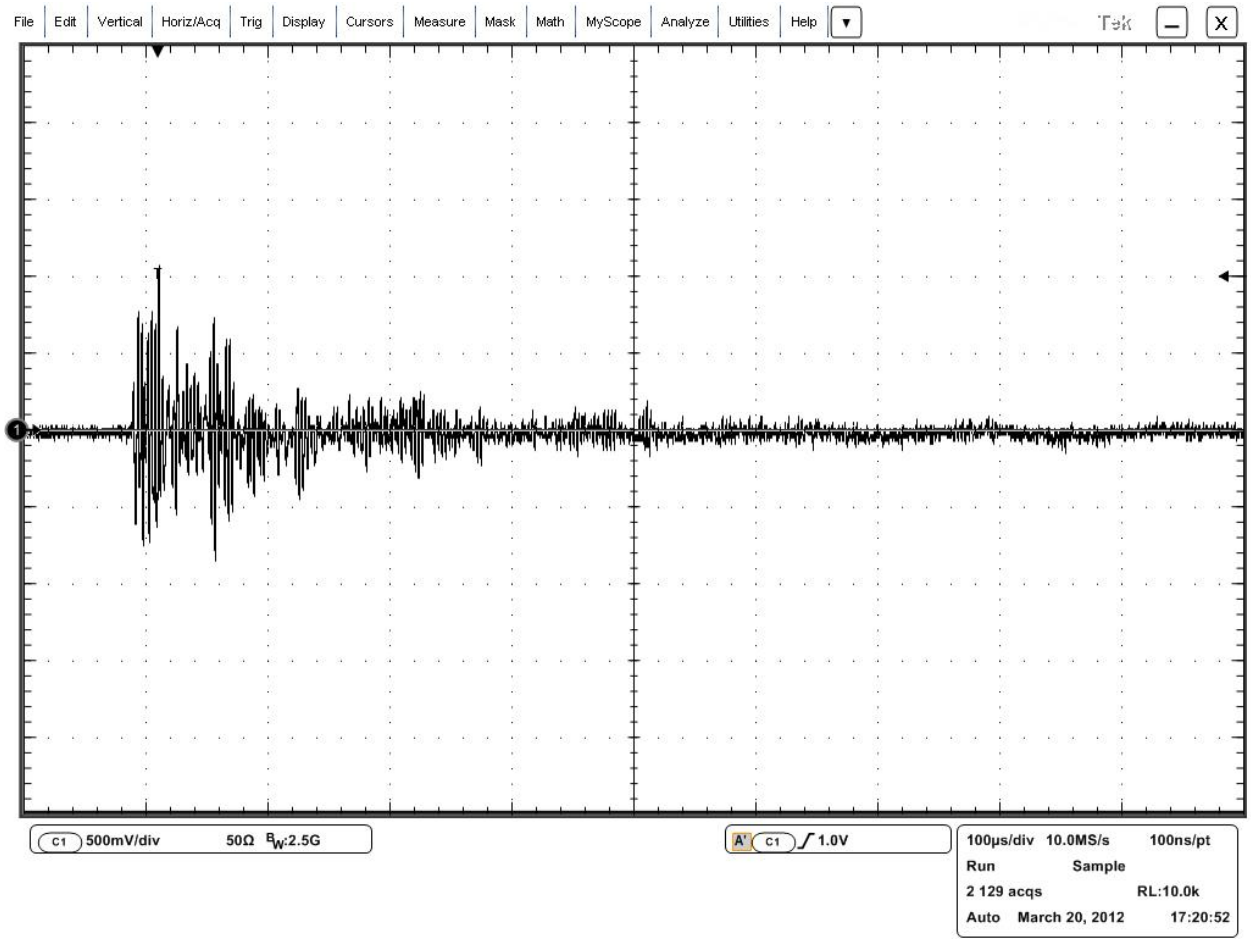

Figura 3.3: Señal introducida.

Como pudimos ver en el anterior capítulo, es una señal de 1 ms de duración, con oscilaciones de aproximadamente 5 μs. La variabilidad de amplitudes y periodos entre oscilaciones va a dificultar la elección de los valores del circuito. Es por ello, que se realizon unos cálculos aproximados de dichos valores, para más tarde simularlos y probarlos en un circuito real.

3.2.1. Circuito detector de envolvente

El diseño del circuito detector de envolvente se ha realizado con el esquema típico que consta de un diodo y un grupo RC en paralelo, ver Figura 3.4. El funcionamiento del circuito es claro, el diodo permite que circule la intensidad cuando es positiva, cargando el condensador y haciendo que se descargue al ritmo que marque la constante de tiempo a través de la resistencia en la parte negativa. El montaje global queda representado en la Figura 3.5.

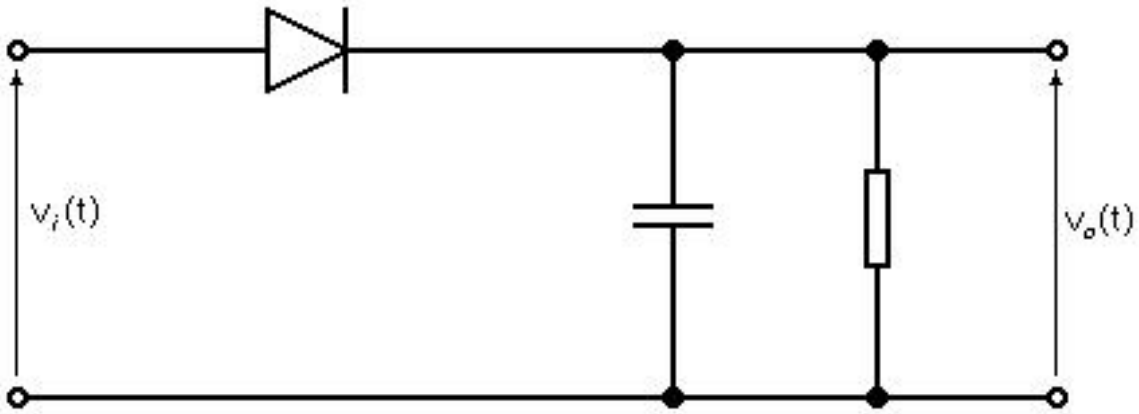

Figura 3.4: Circuito detector de envolvente.

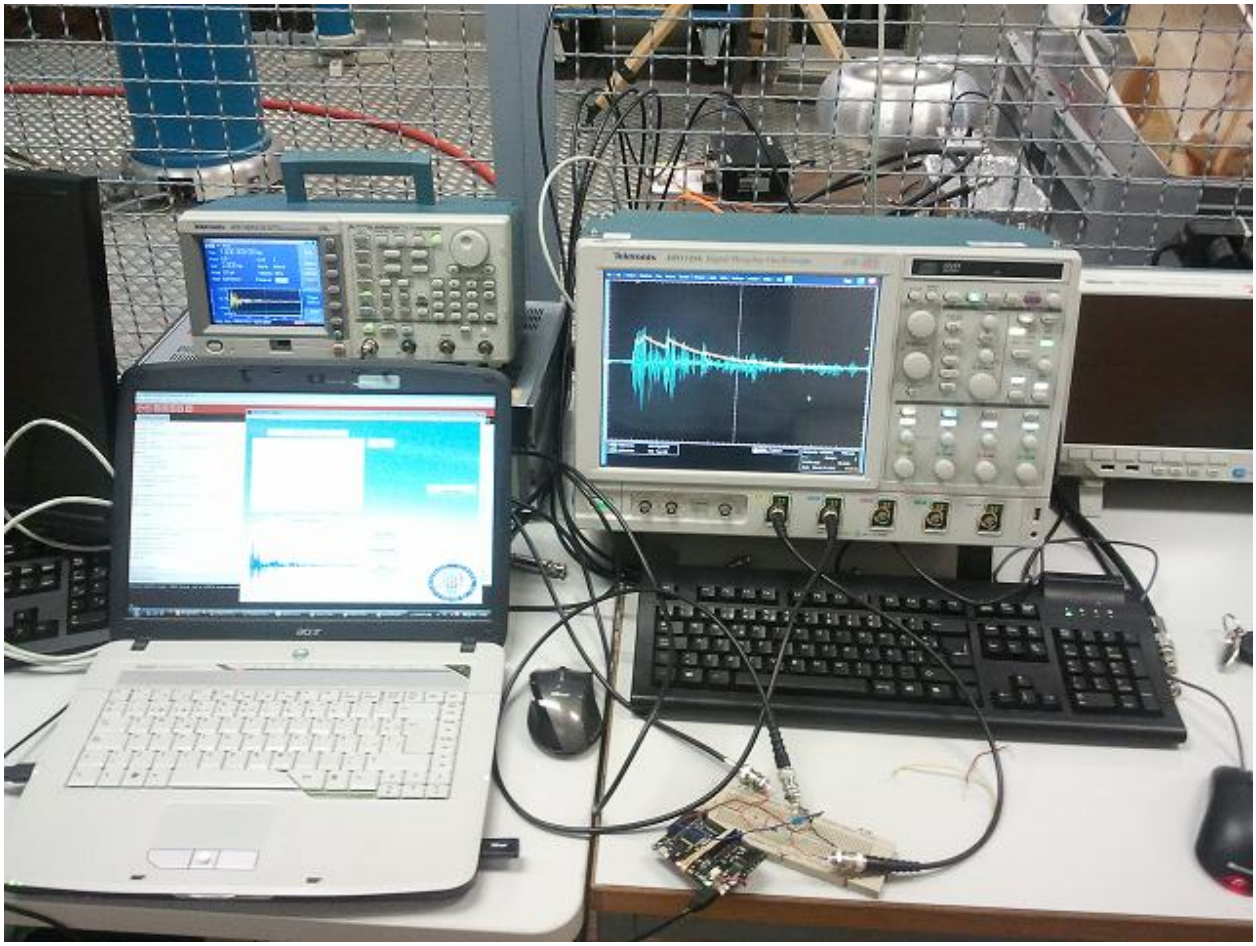

Figura 3.5: Montaje en laboratorio

La constante de tiempo τ se define como el tiempo necesario para que la carga del condensador alcance el 63,2 % del valor final, $\tau = \mathrm{RC}$, por lo tanto $1/\mathrm{f_{min}} > \mathrm{RC} \cdot 2\pi > 1/\mathrm{f_{max}}$ [5], [6]. Si consideramos que el periodo máximo de la señal es de 1 ms y el más pequeño es de 5 μs; $7,9 \cdot 10^{-7} < \mathrm{RC} < 1,6 \cdot 10^{-4}$.

Las señales que se están utilizando son pulsos con tiempos de subida muy rápidos y con oscilaciones de distintas frecuencias. Esto hace que la elección de la constante de tiempo no sea una tarea sencilla. Se han escogido dos extremos de periodos que podrían encontrarse en la señal, $\mathrm{T_{max}} = 1$ ms que es la duración total de la señal adquirida y que es muy conservador y $\mathrm{T_{min}} = 5$ μs que es la duración de la frecuencia más importante del pulso. La constante de tiempo debe estar entre $7,9 \cdot 10^{-7}$ y $1,6 \cdot 10^{-4}$ como se ha indicado anteriormente. Además, hay que tener en cuenta que la señal es tan solo un modelo y las señales que se reciban serán ligeramente diferentes a la introducida. Para solventar estos problemas se ha recurrido a una simulación previa que será útil para la elección de los componentes más adecuados.

3.2.2. Simulación en Multisim

Para poder elegir los valores realizamos un circuito en el programa de simulación eléctrica NI Multisim y vamos probando con diferentes valores de resistencias y condensadores. Si bien es cierto que hay una gran cantidad de combinaciones que pueden ser válidas, no todos los valores están presentes en el laboratorio o no son valores comerciales. El circuito definitivo se obtuvo como resultado de estas limitaciones, ver Figura 3.6.

NI Multisim incluye una fuente de tensión capaz de reproducir una señal desde un archivo de datos, esta función se llama *PWL voltage*. Tan solo tenemos que seleccionar el archivo y reproducirá esa misma señal en el circuito. Como se puede observar en la Figura 3.6, hemos realizado un circuito similar al indicado en el apartado anterior. La resistencia elegida es de 12 kΩ y el condensador de 6,8 nF, por lo tanto la constante de tiempo RC será aproximadamente $8 \cdot 10^{-5}$. Esta constante de tiempo se encuentra bastante cercana al valor límite superior hallado anteriormente, esto es debido a que nos interesa una reacción lenta del circuito detector de envolvente, sin llegar a disminuir mucho el nivel de amplitud de la señal.

El resultado de la simulación puede verse en la Figura 3.7. El comportamiento, si bien es idealizado, se acerca a lo que necesitamos obtener para introducir dicha señal en el sistema de adquisición de datos. Analizando la representación más detalladamente se puede ver la reacción del circuito ante los cambios de la señal. Ante la llegada inicial de un escalón de casi 1 V el circuito detector de envolvente reacciona con un rápido ascenso del nivel de tensión, posteriormente los picos de tensión se mantienen de un modo bastante estable hasta que alrededor de los 110 μs aparecen los picos más altos de la señal modelo, aproximadamente 1,5 V, que hacen de nuevo ascender el nivel del circuito RC. Posteriormente comienza un lento descenso del nivel de tensión hasta que finalmente sobre los 500 μs la señal queda extinta.

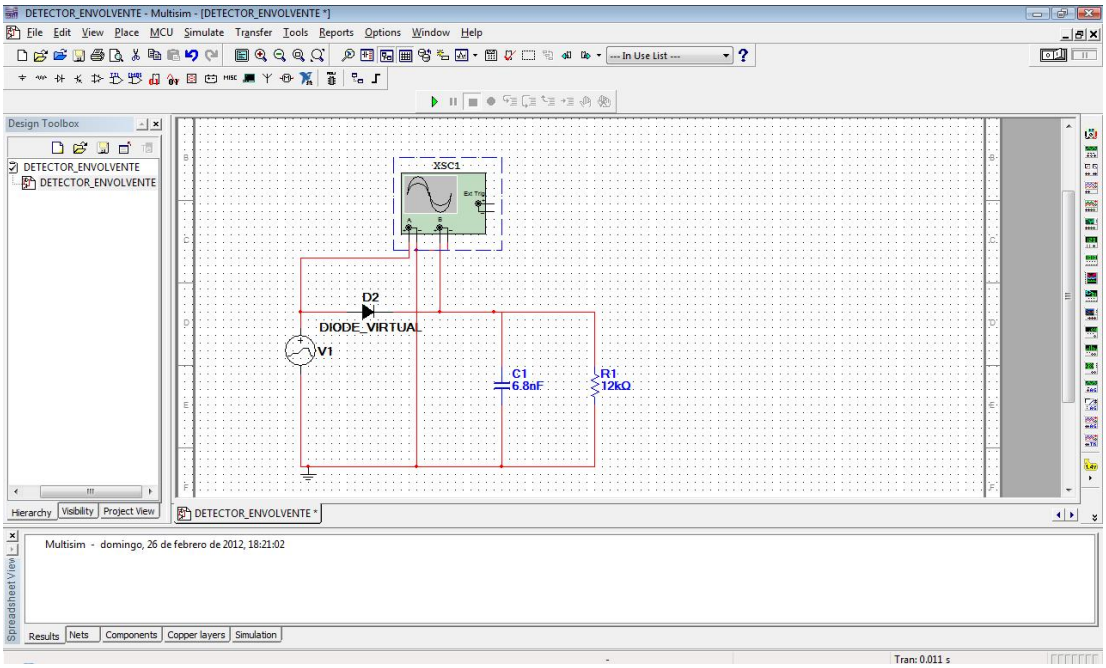

Figura 3.6: Circuito detector de envolvente simulado.

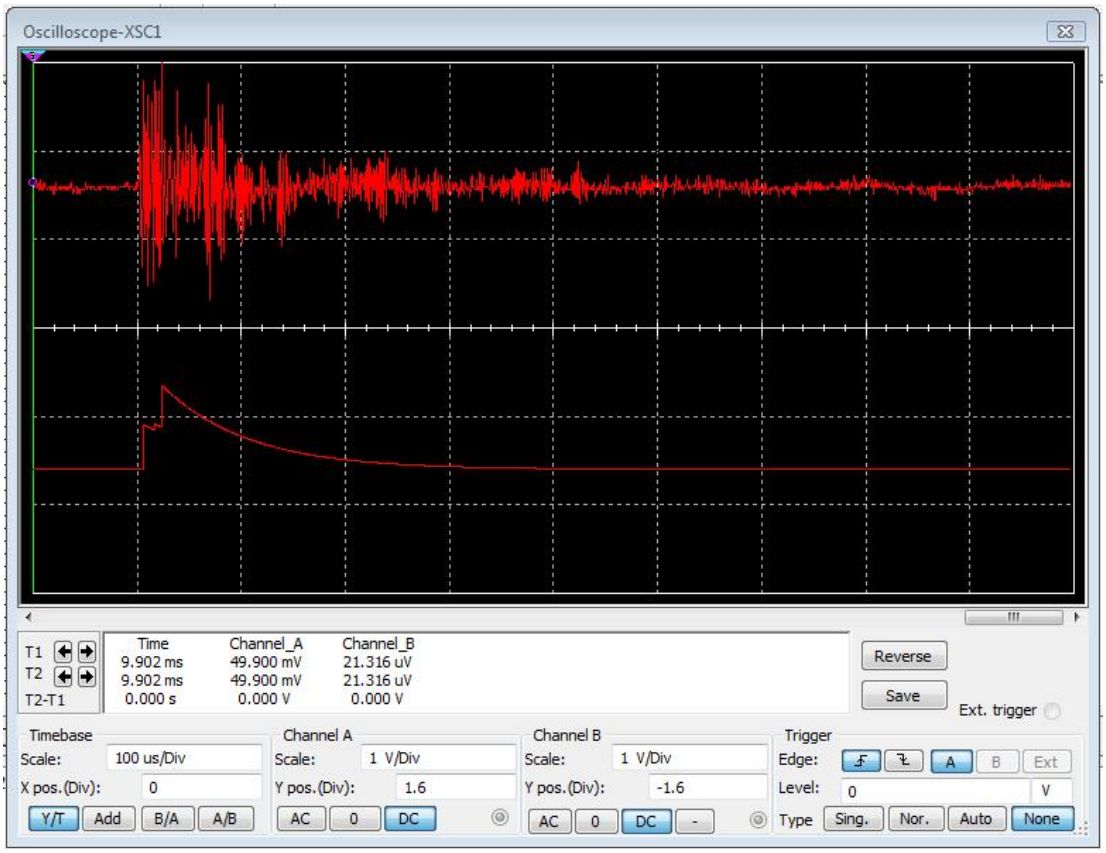

Figura 3.7: Resultado de la simulación.

3.2.3. Circuito real

Los valores seleccionados para la resistencia y el condensador son los indicados anteriormente. La elección del diodo del circuito debe tener en cuenta el nivel de *slew rate* que es capaz de soportar, así como el rango de frecuencias y niveles de tensión soportados. En nuestro caso se ha elegido el modelo de diodo 1N5818TR [7], se trata de un tipo de diodo especial, es un diodo Schottky [12], caracterizado por soportar frecuencias más altas que un diodo común. Debido a que la señal tenía oscilaciones de aproximadamente 200 kHz, los diodos comunes dejaban pasar parte de la señal negativa haciendo que el circuito no tuviese el comportamiento esperado, el diodo 1N5818TR tiene un *slew rate* de 10000 V/μs [7] mientras que los tiempos de subida y bajada máximos de nuestra señal modelo se encuentran en torno a 2 V/μs, valor claramente por debajo del límite ofrecido por el diodo. Se puede esperar que el comportamiento sea similar al representado en NI Multisim en el apartado anterior.

En cuanto al condensador y a la resistencia, únicamente tendremos la precaución de no utilizar condensadores electrolíticos, los cuales sólo funcionan correctamente cuando se trata de frecuencias bajas.

La señal de salida del circuito detector de envolvente queda representada junto a la señal de entrada en la Figura 3.8. En esta imagen se puede apreciar como el comportamiento es prácticamente idéntico al simulado anteriormente, quedando representada la envolvente de la señal, si bien es cierto que el circuito real tiene una caída de tensión de aproximadamente 0,2 ó 0,3 V, que corresponde con la caída de tensión típica del diodo. Esto se observa con claridad en el punto más algo de ambas señales, la original alcanza un valor máximo superior de aproximadamente 1 V mientras que el detector de envolvente tan solo llega a un valor máximo superior de 0,8 V.

Se concluye que se han elimitado las altas frecuencias de la señal, obteniendose como resultado la envolvente de la misma, esto facilitará la toma de datos del microprocesador, ya que al tener niveles más estables de tensión permitirá cargar el condensador que posee el circuito *sample and hold* del conversor analógico digital [13] y tomar medidas de manera correcta.

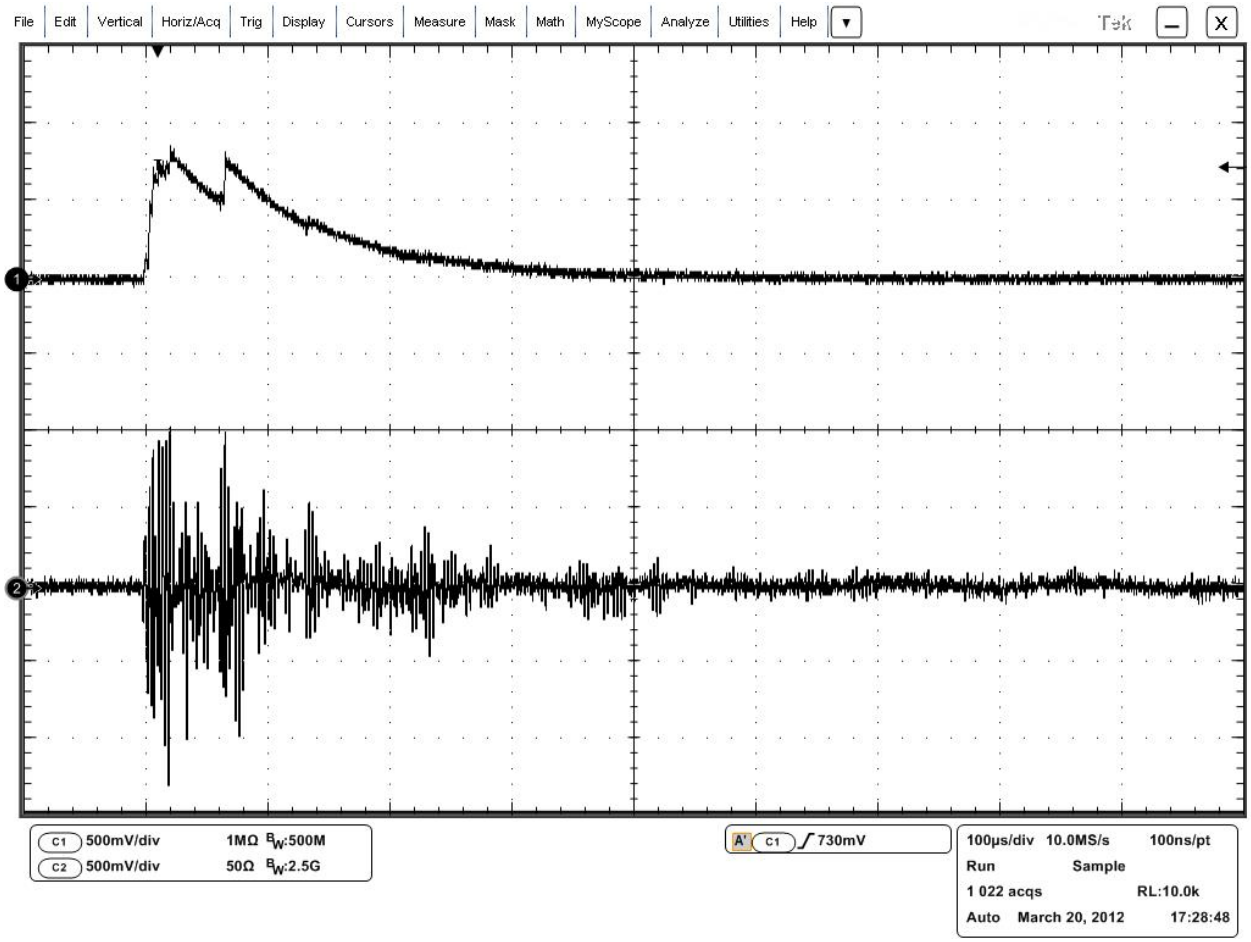

Figura 3.8: Resultado final del filtro de envolvente.

Capítulo 4

Adquisición de datos, Waspmote

4.1. Introducción

El mercado de los microprocesadores y tarjetas de adquisición es muy amplio, se puede encontrar una gran variedad de productos, precios y características. El modelo que se usará en este proyecto es Waspmote de la empresa española Libelium, se trata de una tarjeta que cuenta con un microprocesador Atmel ATmega1281 [1], esta orientada tanto al uso doméstico como industrial, su principal ventaja reside en que se trata de un sistema abierto, lo cual facilita su difusión, abarata su coste y repercute en que haya una gran información sobre él en internet. A continuación, se indican las características generales del producto [1]. Se puede hacer especial hincapié en el bajo consumo que tiene. En condiciones normales es de sólo 9 mA, pero se puede reducir a 0,7 uA en modo hibernación en el que está a la espera de recibir una señal que lo active para realizar las medidas y luego volver a ese mismo estado. De esta manera, puede funcionar en durante meses con la carga de una batería.

4.1.1. Características Generales:

- Microcontrolador: ATmega1281
- Frecuencia: 8 MHz
- SRAM: 8 KB
- EEPROM: 4 KB
- FLASH: 128 KB
- SD Card: 2 GB
- Peso: 20 gr
- Dimensiones: 73.5 x 51 x 13 mm
- Rango de Temperatura: [-20ºC, +65ºC]
- Reloj: RTC (32 kHz)

- Consumo:

  ON: 9 mA

  Sleep: 62 μA

  Deep Sleep: 62 μA

  Hibernate: 0,7 μA

- Funcionamiento sin recarga: 1año

- Entradas/Salidas:

  7 Analógicas (I), 8 Digitales (I/O), 1 PWM

  2 UART, 1 I2C, 1USB

- Características Eléctricas:

  Tensión de batería: 3,3 V - 4,2 V

  Carga USB: 5 V - 100mA

  Carga placa solar: 6 - 12 V - 280 mA

  Tensión batería auxiliar: 3 V

  Sensores integrados en la placa:

  Temperatura (+/-): -40ºC , +85ºC. Precisión: 0,25ºC

  Acelerómetro: $\pm$2g(1024LSB/g)/ $\pm$ 6g(340LSB/g)

  40 Hz/160 Hz/640 Hz/2560 Hz

El objetivo principal es realizar un sistema que, cumpliendo con las especificaciones previstas, tenga un coste lo más reducido posible. En el mercado se disponen de múltiples opciones, pero si se quiere tener un coste reducido estas se reducen básicamente a dos, ArduinoUno [14] y Waspmote. Estos dos dispositivos se basan en una interfaz de programación que fue desarrollada por Arduino inicialmente, creando un dispositivo de bajo coste *open-hardware* de libre distribución, dando a luz un producto muy económico y con gran volumen de mercado. En la actualidad, y siguiento con la filosofía de libre distribución y bajo coste se encuentra la tarjeta ArduinoUno y la tarjeta Waspmote. ArduinoUno cuenta con un microprocesador ATmega328P [15] en lugar del ATmega1281 de Waspmote, la principal ventaja del segundo sobre el primero es que existe la posibilidad de realizar una lectura diferencial entre dos canales. Para ello, el microprocesador ATmega1281 divide el rango de tensiones que puede tomar entre dos y reparte la mitad de *bits* de resolución entre los valores positivos y negativos. Esta posibilidad no existe en el microprocesador de ArduinoUno, el ATmega328P. La lectura diferencial de la señal es una característica requerida dado que si se diese el caso de recibir un pico negativo de tensión y el diodo no funcionase correctamente, bien por que la señal tiene un *slew rate* que supera al diodo o bien por algún defecto, se quiere tener constancia de dicho evento.

Si comparamos la memoria disponible en ambos dispositivos, ArduinoUno cuenta con:

- SRAM: 2 KB
- EEPROM: 1 KB
- FLASH: 32 KB

Estas cifras son claramente inferiores a las citadas anteriormente de Waspmote, además hay que tener en cuenta que la tarjeta Waspmote incorpora la posibilidad de incluir una tarjeta microSD de hasta 2 GB directamente, mientas que si se quiere realizar esa ampliación en ArduinoUno hay que comprar un modulo especial. De igual manera ocurre con las comunicaciones, si bien la mayoría de modulos de comunicación se pueden usar en ambos dispositivos, ArduinoUno requiere un modulo especial de interconexión con la tarjeta, no ocurre así en Waspmote, donde se pueden pinchar los modulos encima de la tarjeta directamente. Por todo ello, se seleccionó Waspmote, el dispositivo de Libelium, que desde un punto de vista personal, no deja de ser una versión mejorada de la creación de libre distribución de Arduino.

Como puede observarse no se ha atendido solo a criterios económicos, ArduinoUno es ligeramente más barato, sino que se atendió a la posibilidad de poder realizar diversas modificaciones con respecto al proyecto principal, como es la conexión inalámbrica o el uso de la tarjeta microSD para salvar los datos tomados.

Las ventajas más destacables a la hora de valorar Waspmote son:

Ventajas

- Bajo precio.
- Una arquitectura modular, que nos permite añadir las siguientes funcionalidades:

  Aumento de la memoria de almacenamiento mediante tarjeta microSD.

  Comunicación Bluetooth.

  Comunicación Wi-Fi.

  Comunicación GPRS.

  GPS.
- Programación en lenguaje derivado del lenguaje C, mediante una interfaz Java.
- Gran cantidad de información en la red, foro propio.
- Código de libre distribución
- Portable
- Poco consumo, alta duración de batería
- Comparte muchas instrucciones y gran parte de las características con un producto muy extendido, el Arduino.

Cabe indicar que no suponen un sobre coste importante los diferentes módulos incluidos, ésta es una de las razones por las cuales elegimos este dispositivo, ya que en otras tarjetas de adquisición también se podía contar con esas características pero el precio aumentaba de manera considerable.

4.2. Código de la aplicación en Waspmote

Como ha quedado reflejado en los objetivos, se debe capturar la envolvente de la señal indicada por medio de un muestreo con Waspmote. Para poder realizar esta tarea de la manera más óptima posible, en este proyecto:

- Se han modificado algunos registros del microprocesador ATmega1281 para conseguir la máxima resolución en tensión disminuyendo el rango máximo de medida.
- Se ha creado una función de comunicación para establecer contacto con Matlab.
- Se ha creado una función específica para realizar una lectura diferencial de la tensión.
- Se ha analizado y depurado el código con el ánimo de lograr la mayor velocidad de muestreo real, analizando los ciclos de reloj que tardaban en implementarse diferentes instrucciones y sacando del código principal operaciones secundarias.

Se puede encontrar el código completo en el Anexo 2. La explicación de las partes más señalables del código se encuentra en los apartados siguientes.

4.2.1. Diagrama del código

El diagrama de flujo del programa sobre Waspmote, Figura 4.1, representa un breve resumen explicativo de lo que realiza nuestro código. No se han incluido instrucciones concretas sino las líneas generales que sigue el programa definitivo. Cualquier programa en Waspmote debe constar con dos partes fundamentales, una primera función *setup*, en la que se incluyen términos como la inicialización de las comunicaciones serie, vía puerto USB. Por otro lado, está la función *loop*, función que se repetirá sucesivamente. Es necesario tener en cuenta que, a no ser que se reinicie el programa, no se volverá a pasar por la función *setup*, por lo que no pueden incluirse instrucciones que se deban realizar en cada ciclo.

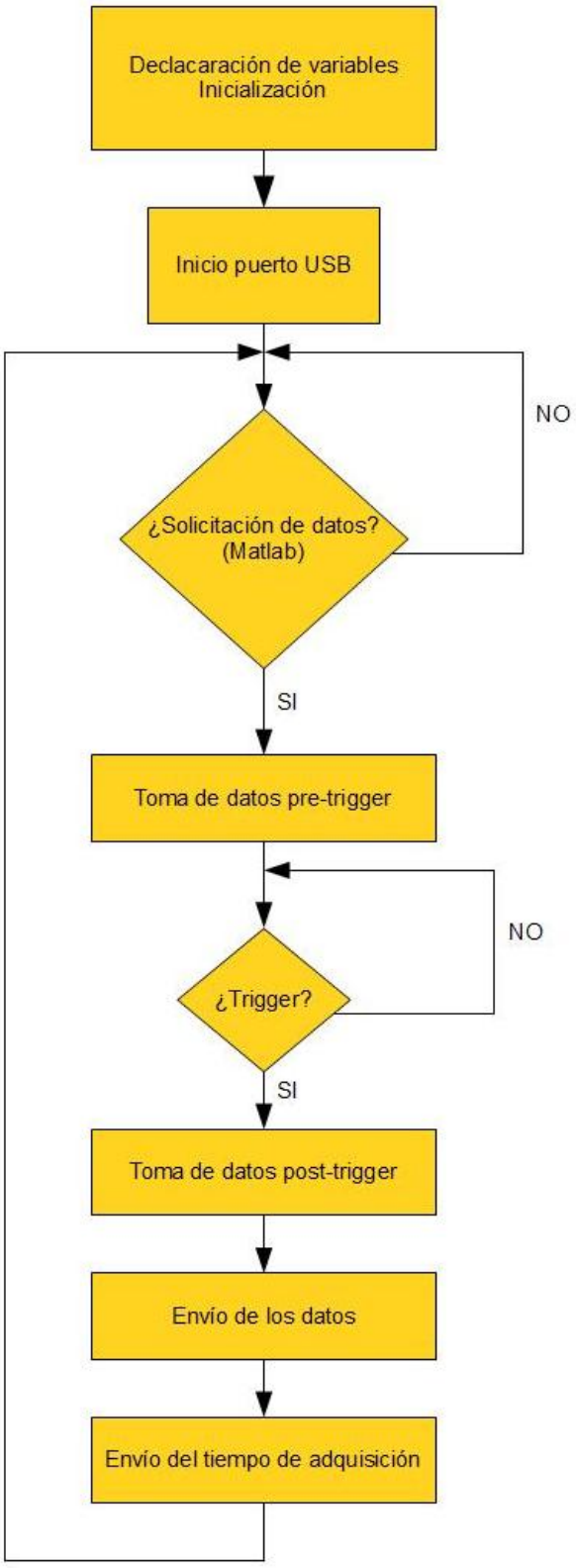

Figura 4.1: Diagrama de programación Waspmote.

Se puede ver cómo tras inicializar las variables y activar las comunicaciones serie, el programa queda a la espera de una solicitud de Matlab. Esta solicitud llegará por el puerto USB a través de la función `EstablishContact` cuyo funcionamiento más detallado se explicará más adelante. Una vez se recibe la solicitud de toma de datos enviada por el usuario a través de Matlab, se procede a la toma de datos. Para realizarla se esperará un *trigger* por nivel de tensión de 500 mV, escogido empíricamente, que indica que ha localizado el paso de una señal. Cuando se haya recibido, se tomarán todos los datos y posteriormente se procederá a enviar al ordenador el vector de muestras y el tiempo en milisegundos que se tardó en tomarlas. Ahí serán recogidos y tratados por Maltab. A continuación se explica con más detalle el proceso de configuración y adquisición.

4.2.2. Configuración de registros

La configuración original de nuestro sistema de adquisición sólo contempla señales de carácter positivo de 0 a 3,3 V, por lo que se debe cambiar la configuración para conseguir tomar medidas en modo diferencial. Para ello se debe configurar determinados *bits* del registro ADMUX, ver Figura 4.2. [3]

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 4.2: Detalle del registro ADMUX.

Vamos a detallar para qué sirve cada uno de los *bits* indicados.

- *Bit* 7 y 6 - Gracias a estos dos bits se pueden configurar diferentes tensiones de referencia. En nuestro caso, si se selecciona una entrada diferencial de tensión se perderá resolución, pasaremos de tener 10 *bits* disponibles para 3,3 V a tener solamente la mitad para la parte positiva de la señal. De esta manera tenemos que la resolución original era $3,3/1024 = 3{,}22$ mV/LSB, con la configuración del modo diferencial tendríamos $3,3/512 = 6{,}44$ mV/LSB y con la disminución del nivel máximo de tensión tenemos finalmente $2,56/512 = 5$ mV/LSB. Hemos preferido disminuir la tensión máxima con el propósito de recuperar parte de esa resolución perdida en el conversor analógico digital. De esta manera la tensión no deberá superar los 2,56 V, tensión que por las características de nuestra señal, no es previsible que sea alcanzada. La siguiente disminución de tensión de referencia corresponde a 1,1 V pero esta no es compatible con una configuración de lectura de señal diferencial. Según la tabla de la Figura 4.3 los dos *bits* deben ser puestos a 1.

REFS1	REFS0	Voltage Reference Selection[1]
0	0	AREF, Internal V_{REF} turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Internal 1.1V Voltage Reference with external capacitor at AREF pin
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

Figura 4.3: Tabla de tensiones de referencia.

- *Bit* 5 - Nuestro sistema trabaja con 8 *bits*, pero el conversor analógico digital consta de 10 *bits*. Esto implica que la conversión sea guardada en dos registros diferentes. Con este *bit* se puede seleccionar una justificación a la derecha o a la izquierda de dichos registros de conversión.
- *Bit* 4 a 0 - Nos permite seleccionar una de entre las múltiples siguientes posibilidades:

MUX5:0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
001000[1]	N/A	ADC0	ADC0	10×
001001[1]		ADC1	ADC0	10×
001010[1]		ADC0	ADC0	200×
001011[1]		ADC1	ADC0	200×
001100[1]		ADC2	ADC2	10×
001101[1]		ADC3	ADC2	10×
001110[1]		ADC2	ADC2	200×
001111[1]		ADC3	ADC2	200×
010000		ADC0	ADC1	1×
010001		ADC1	ADC1	1×
010010		ADC2	ADC1	1×
010011		ADC3	ADC1	1×
010100		ADC4	ADC1	1×
010101		ADC5	ADC1	1×
010110		ADC6	ADC1	1×
010111		ADC7	ADC1	1×
011000		ADC0	ADC2	1×
011001		ADC1	ADC2	1×
011010	N/A	ADC2	ADC2	1×
011011		ADC3	ADC2	1×
011100		ADC4	ADC2	1×
011101		ADC5	ADC2	1×

Figura 4.4: Lectura diferencial en diferentes canales.

Se puede ver que la tabla 4.4 hace referencia a MUX5:0 cuando en el registro ADMUX de la Figura 4.2 solo aparecen cuatro *bits* MUX, esto se debe a que el quinto *bit* se encuentra en el registro ADCSRB. El quinto *bit* sirve para realizar configuraciones más alla de los 8 canales analógicos que se tienen en el dispositivo seleccionado para este proyecto, por lo tanto no se podrá hacer uso de esos direccionamientos, el *bit* MUX 5 será 0.

4.2.3. AnalogReadDif

Esta función ha sido creada en el proyecto especialmente para leer datos en modo diferencial. El microprocesador ATmega1281 contempla la posibilidad de leer la tensión en modo diferencial pero no esta activado por defecto en la placa Waspmote. Para realizar la función de lectura en modo diferencial se ha usado como base una función existente similar, la función `AnalogRead(pin _ number)`. Funciona indicando el *pin* entre paréntesis, luego se conectará el terminal positivo de la señal en dicho *pin*, mientras que el terminal negativo será conectado a GND.

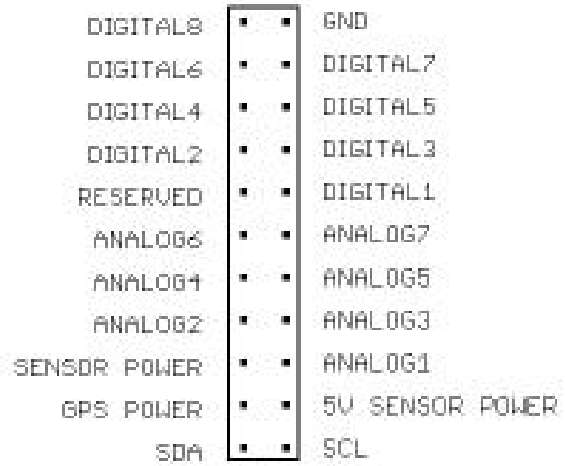

Figura 4.5: Salidas y entradas disponibles en Waspmote.

Para poder realizar esta función hemos tenido de nuevo que internarnos en los registros y realizar una lectura paso a paso del dato. En primer lugar, hay que modificar con cada toma de datos el *bit* ADSC, *bit* 6 dentro del registro ADCSRA, ver Figura 4.6.

Bit	7	6	5	4	3	2	1	0	
(0x7A)	**ADEN**	**ADSC**	**ADATE**	**ADIF**	**ADIE**	**ADPS2**	**ADPS1**	**ADPS0**	**ADCSRA**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 4.6: Detalle del registro ADCSRA.

Si se pone el *bit* ADSC a 1 con la funcion `sbi(ADCSRA,ADSC)`, el conversor comienza a realizar la conversión, poniéndolo automáticamente de nuevo a 0 cuando termina. El resto de *bits* del registro ADCSRA no se han modificado de manera manual.

Seguidamente, se espera en el bucle *while*, línea 7 del código que puede verse al final de este apartado, a que se reciba un 0 en el *bit* ADSC, es decir, a que termine la conversión. Una vez ha finalizado la conversión tenemos dos registros de 8 *bits* donde el conversor analógico digital escribe el dato que se ha tomado, ADCL y ADCH. Para poder guardarlo como un entero de 16 *bits* hay que realizar una operación de desplazamiento y una operación lógica OR, se puede ver más claro en el siguiente ejemplo donde se enfatiza en las operaciones realizadas en las líneas 8, 9 y 10 del código.

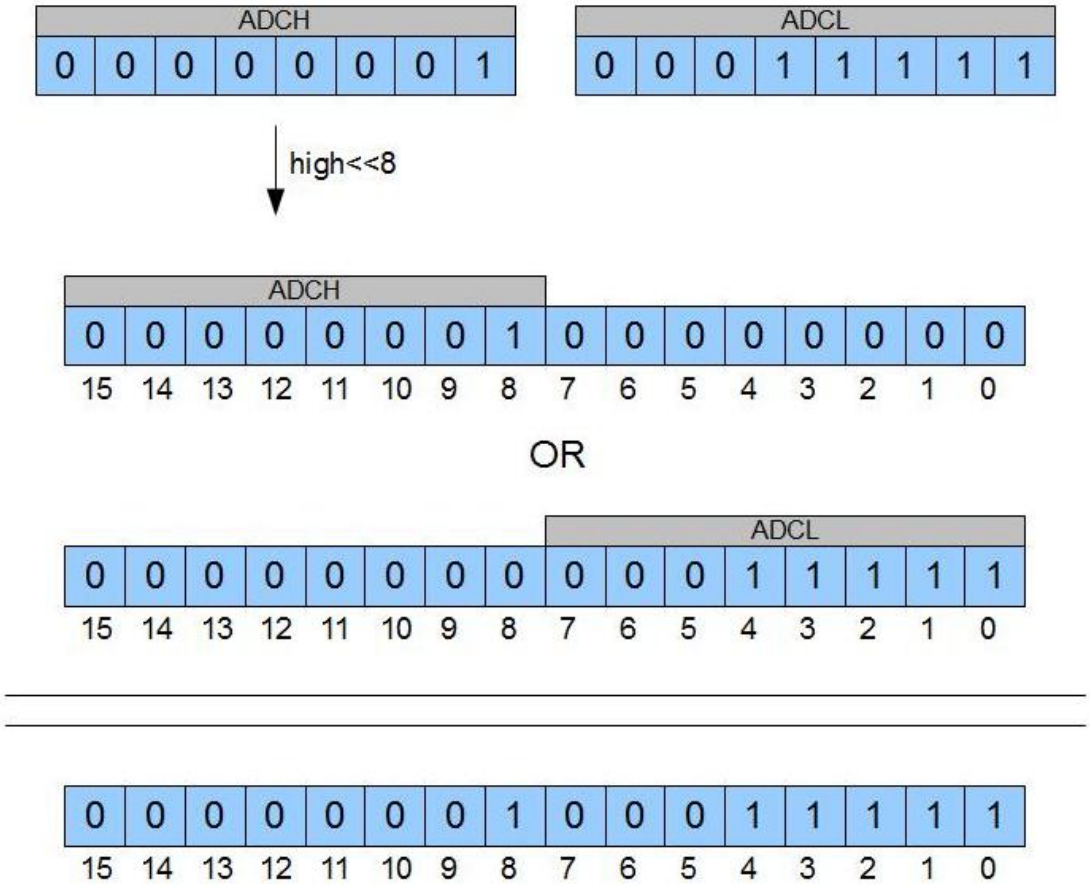

Figura 4.7: Lectura de datos de los registros ADCL y ADCH

Dado que la justificación por defecto del registro ADCH es a la derecha, en la Figura 4.7 se observa como con el operador << se desplazan los *bits* las posiciones que deseemos, en nuestro caso, se desplazará el registro ADCH 8 posiciones a la izquierda. Una vez lo hemos desplazado, se realiza una operación lógica `OR` entre ambas partes, recordemos que la operación `OR` equivale a la suma de ambas posiciones binarias. De este modo, en la variable final de tipo entero de 16 *bits* se tiene el resultado, el nombre de dicha variable entera es *result*.

La función creada es la siguiente.

```
1)  int AnalogReadDif()
2)
3)    {
4)   int result;
5)   uint8_t low, high;
6)   sbi(ADCSRA, ADSC);
7)   while (bit_is_set(ADCSRA, ADSC));
8)   low = ADCL;
9)   high = ADCH;
10)   result=(high << 8) | low;
11)   return result;
12)   }
```

Al tener los datos ya guardados en un vector se tienen dos opciones, guardarlos en una tarjeta de memoria microSD o enviarlos a través del puerto USB a un ordenador, para poder realizar esta segunda opción se ha creado la función del siguiente apartado.

4.2.4. EstablishContact

Todo código de Waspmote debe incluir un bucle *loop*, se trata de un bucle infinito que repite las instrucciones que hay en su interior hasta el reinicio de la tarjeta. La función `EstablishContact` es llamada al comienzo de cada uno de esos bucles. Ha sido creada con el fin de establecer comunicación con Matlab y al mismo tiempo crear un punto de espera para no estar tomando datos continuamente sin que el usuario, a través de Matlab, lo haya ordenado. La ejecución del código quedará retenida entre las líneas 4 y 8 del código que se indica a continuación. Mientras se ejecute esa parte, se estará enviando el código ASCII *backspace*, la finalidad de este código es que Matlab detecte que Waspmote está a la espera de ordenes suyas. Se ha incluido un pequeño *delay* de 300 ms con la finalidad de no saturar las comunicaciones. La condición impuesta para que el código continúe su ejecución es que `USB.available` sea distinto de 0, dicha instrucción indica si hay algún dato en el *buffer* pendiente de ser leído, si es así, muestra un 1, de lo contrario muestra un 0. De este modo, solo cuando Matlab envíe a Waspmote alguna orden, éste saldrá del bucle. Se puede concluir que estas líneas de código se encargan de enviar un código de comunicación y permanecen a la espera de una respuesta por parte de Matlab. Cuando se recibe esta respuesta, se borra el contenido del *buffer* de la UART mediante la instrucción `USB.flush`, de esta manera `USB.available` vuelve a ser 0.

El detalle del código de esta función es el siguiente.

```
1)  void establishContact()
2)   {
3)
4)   while (USB.available() <= 0)
5)   {
6)   USB.println(8);
7)   delay(300);
8)   }
9)
10)   USB.flush();
11)   }
```

4.2.5. *Trigger* y número de muestras

Para poder comprender la elección tomada al respecto es fundamental comprender algunas características de la señal y del propio Waspmote.

El reloj que incorpora Waspmote es de 8 MHz, no obstante, el conversor ADC no recibe esa frecuencia, sino que se realiza un preescalado previo que por defecto es de 64. Siendo esto así, 8000000/64 = 125000 Hz. Por su lado, el conversor tarda 13 ciclos de reloj en realizar una conversión completa, de esta manera la frecuencia final de muestreo quedaría 125000/13 = 9615 Hz.

Esta frecuencia resulta baja para intentar registrar la señal oscilante inicial, pero es suficiente para la señal que procede del detector de envolvente. La señal que procede del detector ocupa aproximadamente 1 ms, siendo esto así, podríamos tomar 9 ó 10 muestras dentro de ese periodo de tiempo. Cabe destacar que para poder obtener este valor ideal de muestreo, apenas se deben gastar ciclos en el resto del código, es por ello que se han hecho diferentes pruebas con varios códigos para hallar el que más rápidamente respondía. De estas comparaciones se sacaron varias conclusiones, la primera de ellas es que realizar una comparación simple entre números enteros apenas consumía ciclos de reloj, contrastando con la gran cantidad de ciclos de reloj que consumían las instrucciones que implicaban medidas y comparaciones temporales. Por lo tanto, para establecer un límite en la recogida de muestras se puede usar dos métodos:

- Un límite temporal, se guardan el máximo número de muestras posible en un tiempo T indicado. Alto consumo de ciclos de reloj.
- Un límite numérico, se guarda un número indicado de muestras N, sin límite de tiempo. Bajo consumo de ciclos de reloj.

Se ha elegido la segunda opción por lo comentado anteriormente, implica un menor gasto de ciclos de reloj. En segundo lugar, se hizo patente la imposibilidad de transmitir por USB cada dato tomado nada más ser adquirido, el gasto de tiempo era excesivo y lo más

problemático es que se trataba de una cantidad de tiempo aleatoria, por lo que se optó por separar la adquisición de la transmisión de datos.

El procedimiento que se ha llevado a cabo para la captura de muestras está basado en el usado en muchos sistemas de adquisición de datos, el vector de datos constará de unos datos previos al *trigger*, denominador *pre-trigger* y unos datos posteriores al *trigger*, denominados *post-trigger*. Los datos de *pre-trigger* corresponden en nuestro caso a la toma de dos datos de manera continua mientras se espera la llegada del *trigger*, éste se dispara al alcanzar un umbral de tensión de 500 mV. La elección del número de muestras del *pre-trigger* se debe a que siendo la frecuencia máximam de muestreo 10 kHz, se toman 10 muestras en 1 ms o lo que es lo mismo, 2 muestras en 0,2 ms, si observamos la señal que hemos introducido, con 0,2 ms antes del nivel de 500 mV está garantizado el tomar la señal desde su inicio. Una vez ha ocurrido este evento, comienzan a tomarse los datos de *post-trigger* que en nuestro caso se trata de los 18 restantes hasta completar las 20 muestras totales entre *pre-trigger* y *post-trigger*. Como ya se ha dicho en capítulos anteriores, es necesario saber los niveles máximos alcanzados por la señal y también el tiempo de duración de estos niveles, esta es la razón de que se hayan incluido los datos *pre-trigger*, para poder recuperar la señal completa.

El número de muestras se ha elegido teniendo en cuenta las características de funcionamiento de nuestra tarjeta, la instrucción de medida de tiempo más pequeña que posee Waspmote es `millis()` la cual tiene una resolución del orden del milisegundo. A priori se podría disminuir el número de muestras tomadas a 10, ya que la señal solo tiene una duración de 1 ms y la frecuencia de muestreo es 10 kHz, pero en experimentos empíricos se ha demostrado que no funcionaba correctamente. La razón es que en muchos casos el tiempo de duración de la etapa de muestreo era 0 ms, dato que en realidad no se correspondía con la realidad sino que era fruto del truncamiento de la variable devuelta por `millis`. Por lo tanto, para que esto no sucediese se selecciono un número de datos que se corresponde con un intervalo de tiempo entre 1 y 2 ms, teniendo en cuenta el truncamiento que realiza el microprocesador.

El uso del método de lanzamiento de *trigger* por umbral de tensión se ha considerado el más adecuado por las características de nuestra señal. Si miramos más allá de la señal modelo, el intervalo entre señales que se puede dar en campo puede variar mucho. Nosotros hemos simulado un intervalo de 10 ms entre señales utilizando el generador de ondas en modo *burst*. Este periodo de tiempo será indefinido, aunque acotado, en una situación de toma de datos real. Todo ello justifica el uso de dicho método para marcar el inicio de la toma de datos, evitando el muestreo de zonas temporales en las que no exista señal alguna.

Capítulo 5

Aplicación desarrollada en Matlab

5.1. Introducción

Para la recepción, tratamiento y reproducción de los datos se ha elegido el programa Matlab. Hay múltiples programas con los que se podría haber desarrollado la misma aplicación, sin ir más lejos encontramos Labview, capaz de realizar una programación de objetos mucho más intuitiva y realizar un filtrado más sencillo de la señal.

La elección de Matlab fue por dos motivos, en primer lugar, se priorizó la capacidad de cálculo frente a la comodidad que pudiese otorgar Matlab Guide para la creación de una aplicación. Matlab Guide es un entorno de programación que ofrece Matlab para poder realizar y ejecutar programas de simulación a medida de forma simple. Tiene las características básicas de todos los programas visuales como Visual Basic o Visual C++ [8], pero no es tan sencillo realizar aplicaciones como lo es con otros programas como Labview. En segundo lugar, Mathworks, líder en desarrollo de software de cálculo matemático para ingenieros y científicos, además de creador de Matlab, ofrece múltiples detectores de envolvente ya desarrollados y listos para probarse.

5.2. Detector de envolvente software

El detector de envolvente software es una utilidad que nos ha servido para poder comparar el detector hardware con una envolvente que se ha obtenido mediante programación y que se puede considerar ideal.

Para realizar el detector de envolvente software nos hemos basado en uno que ya estaba desarrollado en Mathworks [9]. Este programa, idealmente realiza la conversión de la Figura 5.1 según anuncia su autor.

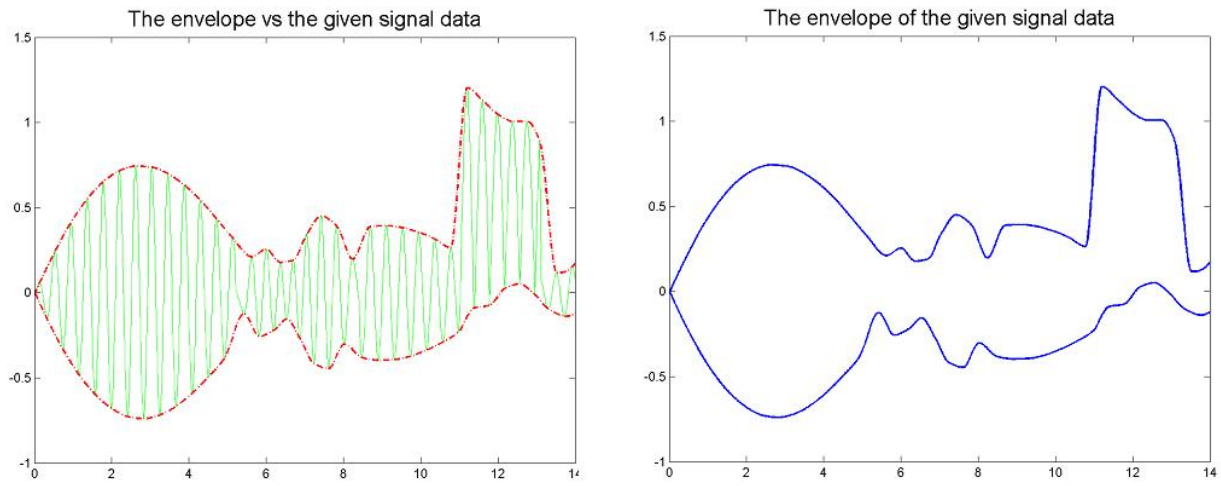

Figura 5.1: Detector de envolvente software [9].

El código es sencillo y está correctamente comentado por su autor, sencillamente se crea una función `envelope` que recibe dos vectores, X e Y y el modo de interpolación a utilizar. Con estos datos, halla los valores extremos superiores e inferiores y realiza una interpolación para cada listado de valores, por un lado los positivos y por otro los negativos. Se puede encontrar este código en el Anexo 4.

5.3. Comunicacion USB

Para la realización de una comunicación serie a través de Matlab es fundamental el uso de la función `serial`. Está función se usa para indicar el puerto mediante el cual se establecerá la comunicación, la velocidad y en definitiva todos los parámetros relativos a la transmisión de datos. La función devuelve un objeto que representará este puerto, de esta manera nuestro código será el siguiente.

```
s1 = serial('COM4','Baudrate',38400);
set(s1, 'terminator', 'LF');
set(s1,'Timeout',100);
```

En la última parte del código establecemos el periodo de espera, `Timeout`, que tiene que transcurrir para que el programa considere que no recibe datos y la transmisión es errónea, en este caso 100 segundos.

Una vez tenemos el código del puerto, hacemos uso de las funciones `fopen(s1)` y `fclose(s1)` para abrir o cerrar el puerto. Es extremadamente importante cerrar el puerto antes de que finalice el programa y debemos asegurarnos de que así sea, aunque haya un error durante la ejecución. Para esta tarea hay una función en Matlab que se denomina `try` y funciona de la siguiente manera.

```
TRY
statement, ..., statement,
CATCH ME
statement, ..., statement
END
```

La parte que se encuentra entre `try` y `catch` será ejecutada de una manera corriente, la diferencia está en que si ocurriese algún error durante la ejecución de esa parte de código no finalizaría súbitamente la ejecución del programa sino que previamente se ejecutaría el contenido situado entre `catch` y `end`. Es precisamente ahí donde se debe situar una instrucción para cerrar el puerto serie y garantizar que siempre quede cerrado, así se ha hecho en el código de la aplicación.

Para realizar las comunicaciones Matlab, el código implementado en Waspmote envía el código numérico 8, es decir, *backspace* en código ASCII y espera que le devuelvan el mismo resultado, cuando así sea, capturará en un vector el resto de los datos.

5.4. Adquisición de datos

Una vez capturados los datos en el vector como se vio en el apartado anterior, se transforma a su equivalente en tensión. En la página 288 del manual del microprocesador [3], se puede encontrar la siguiente fórmula de equivalencia para lecturas diferenciales.

$$\mathrm{ADC} = \frac{(\mathrm{V_{pos}} - \mathrm{V_{neg}})512}{\mathrm{V_{ref}}}$$

El voltaje de referencia, como ya se dijo en el capítulo dedicado a la programación de Waspmote será 2,56 V, ganando algo de resolución respecto a los 3,3 V de fondo de escala que hay por defecto. Siendo así, tenemos que la tensión medida en función del dato numérico recibido.

$$\mathrm{V} = \mathrm{ADC[unid]} \cdot 5[\frac{\mathrm{mV}}{\mathrm{unid}}]$$

Cada incremento numérico del conversor A/D de Waspmote implica 5 mV de voltaje.

Una vez adquiridos los datos de voltaje se toma el último dato que ha llegado, ese dato indica el tiempo en milisegundos que se tardó en tomar la muestra. Este número, por la cantidad de datos tomados y la velocidad de muestreo del microprocesador se puede concluir que estará entre 1 y 2 ms. Esta es la máxima precisión alcanzada por Waspmote, por lo tanto, dividiremos el tiempo entre el número de muestras para averiguar el espacio temporal entre muestras. Una vez se tienen ambos vectores, se dibuja en el cuadro de ejes que elijamos, para ello se debe incluir en el código en el primer termino de la función `plot` el código `handles.eje` donde *eje* representa el cuadro de ejes que se desee, para más detalles ver el Anexo 3.

5.5. Aplicación de usuario

La aplicación de usuario se puede arrancar con un archivo *.exe* y permite realizar todas las tareas desde esa pantalla. Para realizar programas con Matlab Guide lo primero que hay que hacer es acceder al propio Guide, de esta manera si vamos a la *command windows* de Matlab y escribimos:

>> *guide*

Nos aparecerá una pantalla donde nos permite la opción de cargar un archivo anterior o comenzar uno nuevo sugiriéndonos varias posibilidades, nosotros elegiremos la más sencilla que consta de un espacio en blanco, las otras tres son plantillas que fueron creadas con el objetivo de hacer más fácil la experiencia del usuario, o como ejemplo. Una vez dentro se puede visualizar algo similar a la Figura 5.2.

En la parte izquierda de la pantalla, se encuentran iconos con las diferentes opciones que se pueden incluir dentro de la ventana que visualizará el usuario; gráficas, botones, mensajes por pantalla, menús desplegables, entre otros.

La metodología a seguir programando con Guide se encuentra entre la programación por objetos y la programación clásica de Matlab, consta de dos archivos uno `.m` (ejecutable) y otro `.fig` la parte gráfica. Las dos partes están unidas a través de las subrutinas `callback`. Para acceder a la parte ejecutable se debe pulsar el icono que aparece señalado en la Figura 5.3.

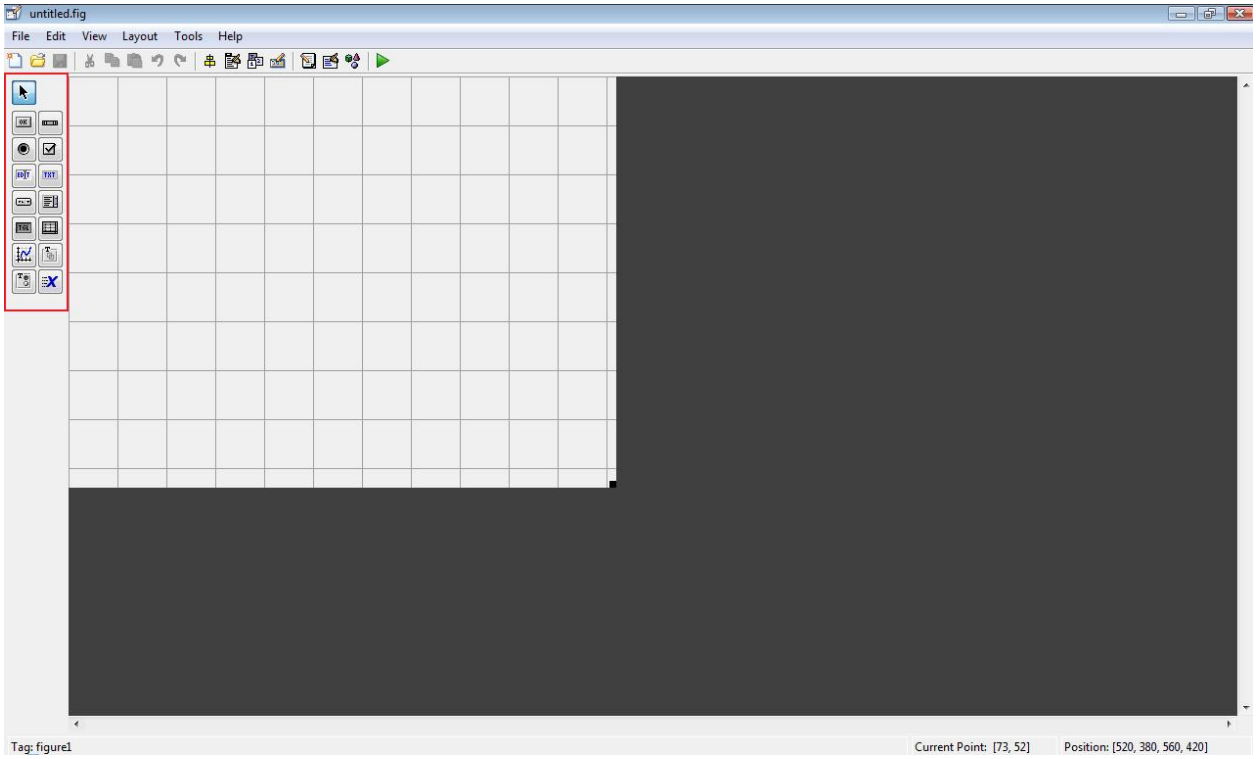

Figura 5.2: Panel de elementos.

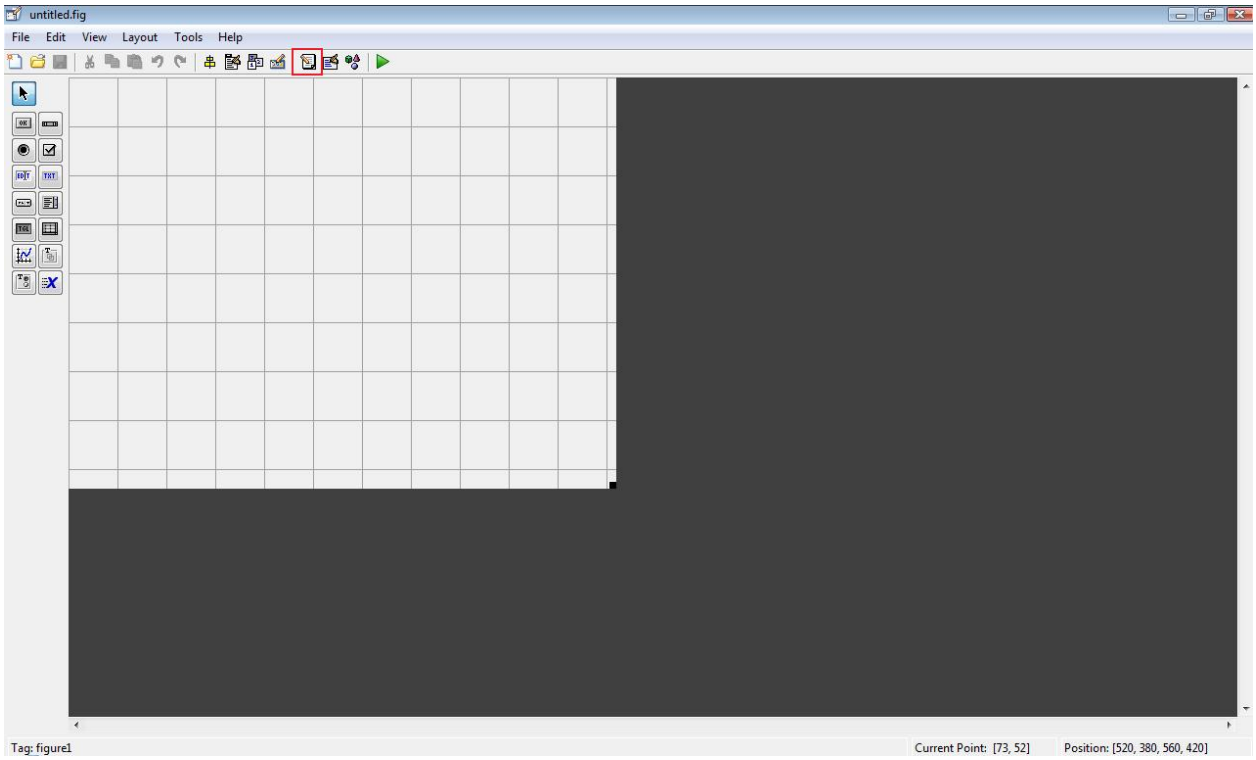

Figura 5.3: Icono de acceso al archivo .m .

Todos los valores de las propiedades de los elementos; color, valor, posición, *string* y los valores de las variables transitorias del programa se guardan en una determinada estructura y son accedidos mediante un único y mismo puntero para todos estos. El nombre del puntero se asigna en el encabezado del archivo *.m*.

La programación a través de este método resulta especialmente inflexible con la manipulación de datos, esto se debe a que se ha organizado cada elemento visual del programa como una subrutina independiente. La transferencia u obtención de los valores de las propiedades de los elementos se realiza mediante las funciones `get` y `set`. A través de `guidata()` se guardan las variables de la aplicación [10].

Los botones y las funciones usadas para la aplicación de usuario de este proyecto quedaran representadas en el siguiente diagrama, en él se podrá ver de una manera intuitiva qué realiza el programa según los diferentes botones y opciones de las que se dispone.

5.6. Diagrama de bloques

Para entender el funcionamiento de los botones y acciones de la aplicación se ha realizado un diagrama de bloques que resume los botones y acciones asociadas, ver Figura 5.4.

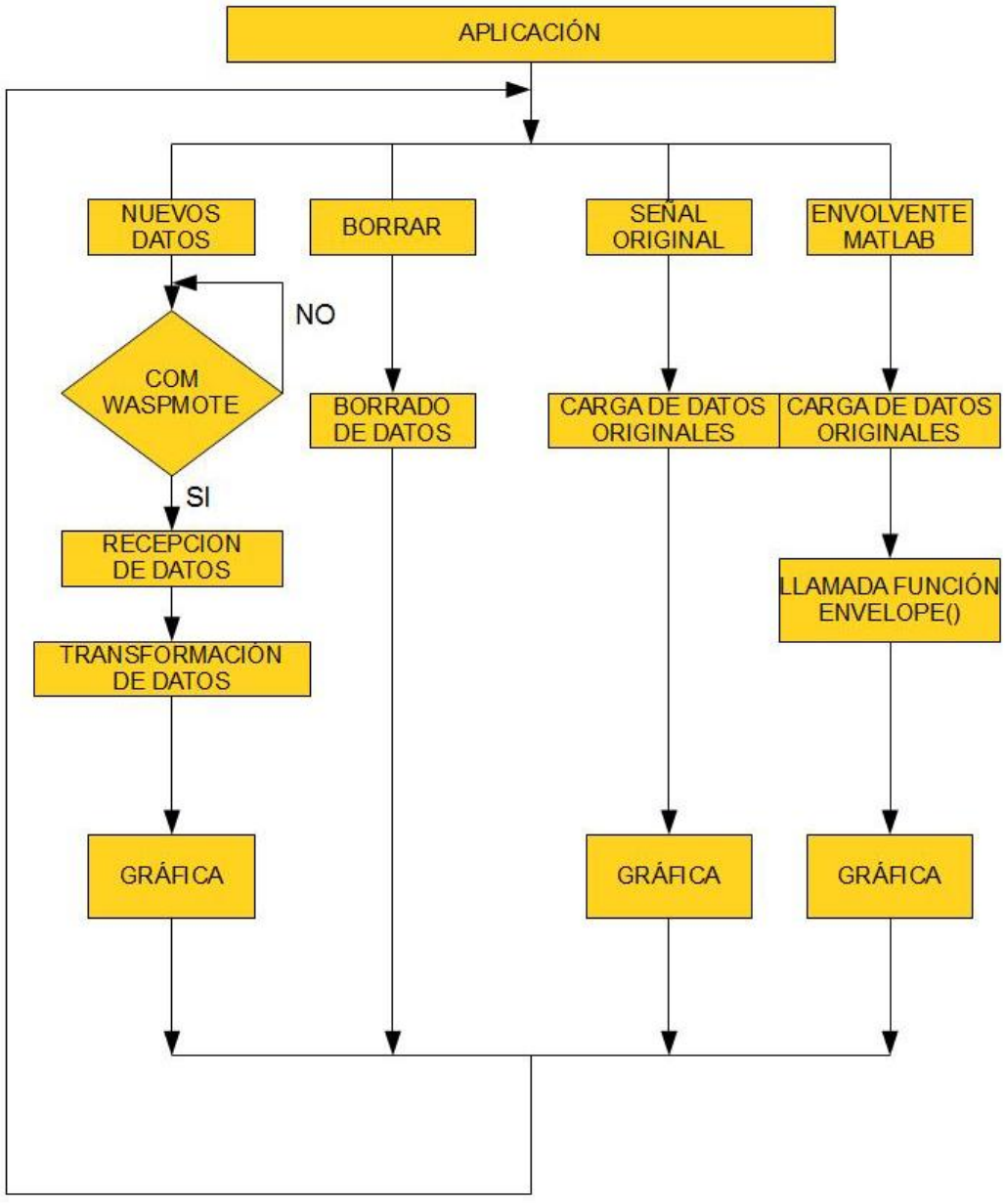

Figura 5.4: Diagrama de bloques de la aplicación de usuario.

Dentro de la aplicación podemos acceder a cuatro acciones principales:

- Adquirir nuevos datos desde Waspmote pulsando el botón 'Nuevos datos', se lanza una orden de petición de datos a Waspmote mediante comunicación serie, se reciben los datos, se procesan haciendo el cambio de unidades numéricas a unidades en mV y se realiza la gráfica en pantalla.
- Se pueden borrar los datos de cada una de las dos pantallas pulsando en 'Borrar'.
- Se puede realizar una gráfica de la señal original pulsando en 'Señal original', toma los datos guardados en la misma carpeta que el programa en los cuales está la señal original de 1000 puntos y la dibuja.
- Por último, si se desea realizar una envolvente software de la señal original, solo hay que pulsar en el botón 'Envolvente Matlab', una vez se ha pulsado el programa llamará a la función `ENVELOPE(X,Y,method)` e introducirá los valores de tiempo y voltaje de dicha señal, además de indicarle que el método de interpolación que se desea usar es *lineal*.

Se puede observar en las siguientes capturas el funcionamiento del programa al realizar diferentes tareas, así como el aspecto que ofrece.

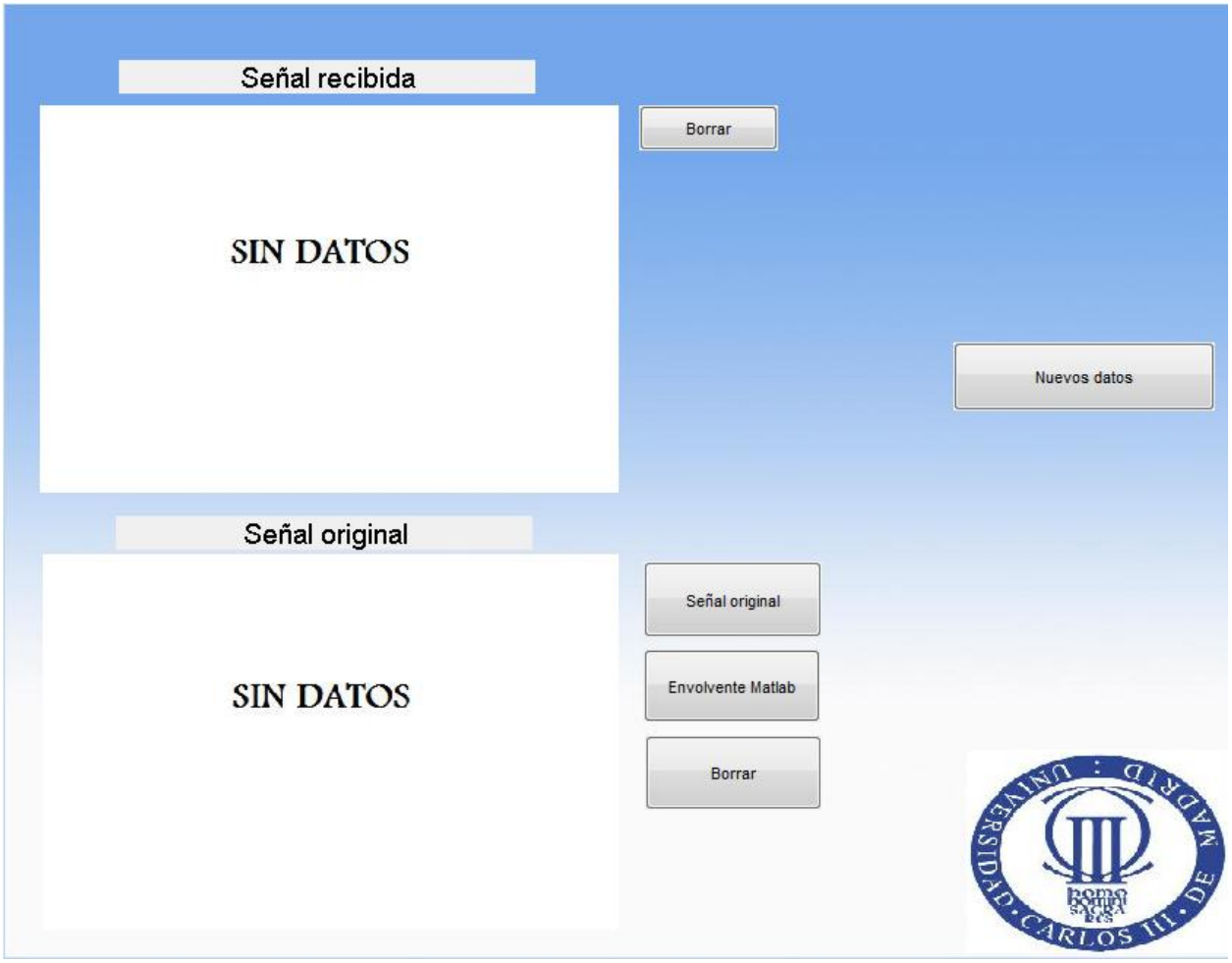

Figura 5.5: Aplicación en el estado inicial.

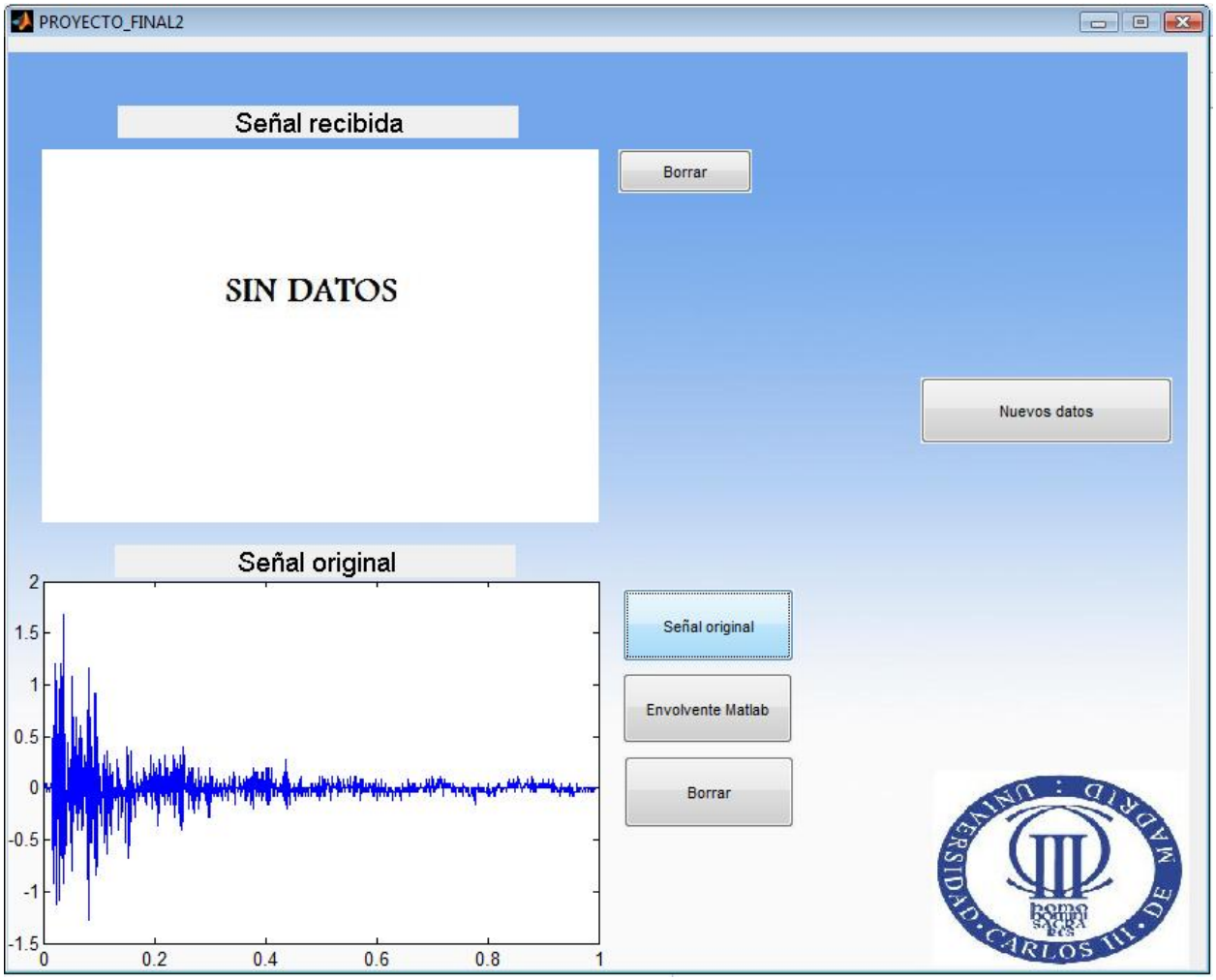

Figura 5.6: Aplicación dibujando la señal original.

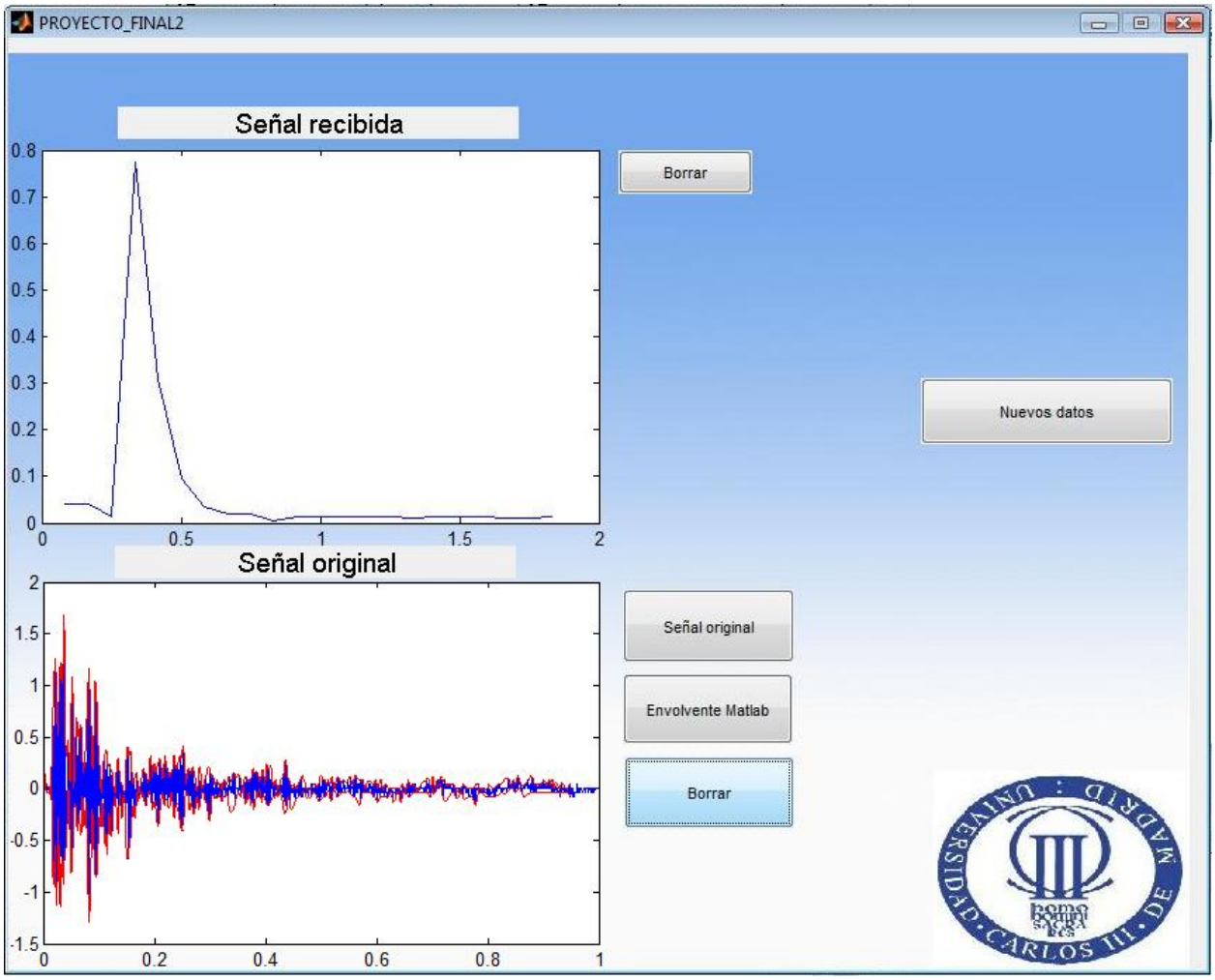

Figura 5.7: Aplicación dibujando la señal original junto con el detector de envolvente software.

Tanto en la Figura 5.5, como en la Figura 5.6, se observan las diferentes ventanas que aparecen en el programa en función de si se acaba de lanzar la aplicación o si hemos pulsado el botón de 'Señal original' respectivamente. En la Figura 5.7 se ven todas las opciones que tiene el programa representadas en pantalla, se va a analizar ambas gráficas.

- 'Señal original': en esta gráfica quedará representado en el eje X el tiempo que se tardó en realizar la adquisición de 20 muestras indicada en apartados anteriores. Este tiempo por lo general es 2 ms. Como se puede ver en la gráfica inferior denominada 'Señal recibida', la escala de dicho eje es diferente, llegando solo hasta 1 ms. Cabe recordar que Waspmote tiene una resolución de tiempo de 1 ms, como ya se explicó anteriormente, esto justifica la elección de 20 muestras e implica que la escala temporal de esta gráfica debe ser 2 ms. Con respecto al eje Y, se corresponde con medidas de voltaje, como ya se explicó, se produce un descenso del nivel de tensión debido a la caída de tensión que produce el circuito detector de envolvente. La duración de la envolvente es de aproximadamente 400 ms, estando el nivel alto desplazado al

centro de la gráfica debido a los datos de *pre-tigger* que se representan al comienzo de la misma.

- 'Señal recibida': se muestra una gráfica con las mismas unidades de medida que la anterior 'Señal original'. En rojo se puede ver la envolvente de la señal, mientras que en azul se representa la señal modelo. Se puede observar como la envolvente realizada por software es mucho más rápida que la hardware, es decir, su constante de tiempo es más pequeña. Al mismo tiempo, el nivel alto de la señal en este caso dura apenas 200 ms, la mitad de la duración de la envolvente hardware, este hecho es premeditado y se debe a que nos interesaba una constante de tiempo grande que no se viese influenciada por las múltiples oscilaciones.

Capítulo 6

Ampliaciones y mejoras

6.1. Almacenamiento en tarjeta SD

En este apartado se ofrece una alternativa que se implemento al inicio del proyecto. Antes de desarrollar las comunicaciones vía puerto serie entre Waspmote y Matlab, se buscó guardar los datos para consultarlos cuando se fuese conveniente. Debido a las limitaciones de Waspmote solo es capaz de guardar en memoria unos 2000 datos tipo entero, este número variará en función de la longitud y complejidad del código que se haya introducido.

Es por ello que acudimos a realizar una ampliación de memoria mediante una tarjeta microSD. Los requisitos principales que debe tener la de memoria usada es ser menos de 2 GB y estar en formato FAT16. La manera más directa de conocer las instrucciones pertinentes para la utilización de una tarjeta microSD en Waspmote las podemos encontrar en el material didáctico del propio dispositivo[4]:

6.1.1. Herramientas de ayuda en la programación

Flag

Para ayudarnos en las labores de programación cabe destacar un 'flag' que se ha creado para obtener información cuando se opere con la tarjeta microSD:

- 0 : nada ha fallado, todo se ha ejecutado correctamente.
- 1 : no hay SD en la ranura.
- 2 : fallo de inicialización.
- 4 : fallo de partición.
- 8 : fallo de archivo de sistema.
- 16 : la raíz del directorio es errónea.
- 32 : Datos demasiado grandes, datos truncados.
- 64 : error al abrir archivo.

- 128 : error al crear archivo.
- 256 : error al crear directorio.
- 512 : error al escribir en archivo.

Mensajes de error

Por otro lado, también tenemos mensajes de error que serán devueltos en forma de cadena de caracteres y que nos proporcionan información sobre lo ocurrido en la operación que hemos llevado a cabo sobre la tarjeta:

- No SD in the slot
- MMC/SD initialization failed
- Opening partition failed
- Opening files system failed
- Opening root dir failed

Hay un *buffer* de 256 *bytes* que limita el tamaño de escritura sobre la SD. Hay que tener en cuenta que el microprocesador trabaja con 8 *bits* y es posible que el límite venga dado por la imposibilidad de direccionar más allá de esos 256 bytes. Ese será el máximo tamaño del paquete de datos que le podremos entregar a la SD para que lo guarde. Este límite debe ser considerado en el desarrollo de las aplicaciones, intentar grabar datos de longitud mayor a la indicada supondrá el truncamiento de los datos, y por lo tanto, el mal funcionamiento del código en conjunto. Resulta obvio que el límite no es solo en la escritura, si no que la lectura esta limitada del mismo modo.

6.1.2. Inicialización y finalización

Pese a que en la guía de programación de Waspmote se indican diferentes instrucciones para hacer uso de la SD, lo cierto es que en el propio foro oficial de Libelium podemos encontrar que recomiendan usar una única instrucción que han creado recientemente para realizar todos los pasos iniciales:

```
{
SD.on();
}
```

Cuando se finalice la escritura es recomendable indicar que el uso de la SD ha finalizado, para lo que se usa la siguiente función:

```
{
SD.close();
}
```

6.1.3. Información complementaria

Presencia

Para comprobar la presencia de la SD en la ranura se puede usar el siguiente código:

```
{
uint8_t present;
present=SD.isSD();
}
```

Si devuelve un '1" es que la SD esta colocada en la correspondiente ranura, mientras que si nos devuelve un '0' significa que no se encuentra colocada, o no está colocada de la manera correcta.

Información sobre la tarjeta

Si realizamos las instrucciones que siguen a continuación obtendremos toda la información acerca de la tarjeta de memoria:

```
{
char* diskInfo;
diskInfo=SD.print_disk_info();
}
```

Un ejemplo del resultado que podemos obtener leyendo *diskInfo* sería.

```
manuf: 0x2
oem: TM
prod: SD01G
rev: 32
serial: 0x9f70db88
date: 2/8
size: 947 MB
free: 991428608/992608256
copy: 0
wr.pr.: 0/0
format: 0
```

Si lo que queremos averiguar es el tamaño total de la tarjeta, usaremos el siguiente código para averiguarlo:

```
{
offset_t diskSiZe;
diskSize=SD.getDiskSize();
}
```

Los resultados son devueltos en *bytes*. Del mismo modo si necesitamos conocer el espacio libre también podemos obtenerlo con la función siguiente.

```
{
offset_t diskFree;
diskFree=SD.getDiskFree();
}
```

Y de nuevo el resultado será devuelto en *bytes*.

6.1.4. Directorios

Con el objetivo de una mayor organización de los archivos guardados en la SD, se puede hacer uso de diferentes directorios. A continuación, vamos a explicar todo lo referente a la creación y gestión de directorios.

Cambiar directorio

Si lo que queremos es cambiar el directorio al que estamos apuntando, usaremos el código que sigue a continuación:

```
{
uint8_t cdState;
const char* command="Folder";//Indica el directorio a cambiar
cdState =SD.cd(command);//Cambia el directorio especificado en command.
cdState =SD.cd("Folder");//Cambia al directorio especificado
}
```

Número de directorios

Devuelve el número de archivos o subdirectorios encontrados en el directorio, o cero si no hay archivos o directorios. Si ocurre un error, devuelve un número negativo.

```
{
int8_t numfiles;
numfiles =SD.numFiles();
}
```

Si por el contrario lo que queremos obtener es el listado de archivos usaremos las instrucciones que indicamos a continuación. Trabajaremos con tres parámetros principales.

- Offset: salta un valor *offset* de archivos de la lista
- Scope: incluye un total de *scope* archivos en el buffer
- Info: limita la cantidad de información que es devuelta.

```
Ejemplo de uso.
{
char* listing;
listing=SD.ls(); // hace un listado de todos los archivos del directorio
```

```
listing=SD.ls(1); // Devuelve el nombre del segundo archivo
listing=SD.ls(2,0, NAMES);//Hace un listado desde el segundo archivo
hasta el final
listing=SD.ls(4,2, SIZES);//Lista dos archivos desde la posición 4
incluyendo el tamaño
listing=SD.ls(1,1,ATTRIBUTES);//Muestra los atributos del archivo de
la posición primera
}
```

SD.buffer almacena las cadenas de caracteres del listado del directorio actual.
Un ejemplo de la salida del comando SD.ls() sería.

```
File_XBee.txt 509
ephemeris.txt 0
```

Un ejemplo de la salida del comando SD.ls(1) sería.

```
ephemeris.txt 0
```

Un ejemplo de la salida del comando SD.ls(1,1,ATTRIBUTES) sería.

```
-w ephemeris.txt 0
```

Buscar un directorio

Con la siguientes instrucciones buscamos un subdirectorio de nombre determinado, dentro del directorio actual. Si la función devuelve '1' significa que el directorio existe, si es un '0', existe pero no es un directorio, y si devuelve '-1', no existe.
Ejemplo de uso.

```
{
int8_t isdir;
const char* name=Folder;
isdir=SD.isDir(name);//Realiza una búsqueda de Folder en el
directorio actual
isdir=SD.isDir(Folder);//Realiza una búsqueda de Folder en el
directorio actual
}
```

Crear un directorio

Las siguientes instrucciones nos permiten crear un directorio, si es creado devolverá un '1' y si no ha podido crearlo un '0'.
Ejemplo de uso.

```
{
uint8_t dirCreation;
const char* name=Folder;
dirCreation=SD.mkdir(name);//Crea un directorio en Folder
dirCreation=SD.mkdir(Folder);//Crea un directorio en Folder
}
```

Borrar un directorio

Estas líneas de código borran el directorio y todo lo que contenga en su interior pero si hay algún subdirectorio devolverá un error.

Ejemplo de uso

```
{
const char* name=Folder;
uint8_t delState;
delState=SD.del(name);//Borra el directorio llamado Folder
delState=SD.del(Folder);//borra el directorio llamado Folder
}
```

6.1.5. Operaciones con archivos

Creacion de archivos

Crea un archivo, devuelve un 1 si el archivo ha sido creado y un 0 si ha ocurrido un error.

Ejemplo de uso.

```
{
const char* name=FileText;
uint8_t fileCreation;
fileCreation=SD.create(name);//Crea un archivo llamado FileText
fileCreation=SD.create(FileText);//Crea un archivo llamado FileText
}
```

Borrado de archivos

Si queremos borrar un archivo determinado situado en el directorio actual. Devuelve un '1' si la operación se realizó de manera satisfactoria o un 0 si ocurrió algún error.

Ejemplo de uso.

```
{
const char* name=FileText;
uint8_t fileDelete;
fileDelete =SD.del(name);//Elimina un archivo de nombre FileText
fileDelete =SD.del(FileText);//Elimina un archivo de nombre
FileText previamente creado
}
```

Apertura de archivos

Abre el archivo indicado, si este no estuviese disponible devolvería un '0'.

Ejemplo de uso.

```
{
struct fat_file_struct* fp;
fp=SD.openFile(file); // abre el archivo y le asigna el puntero fp
}
```

Cierre de archivos

Cierra el puntero que apuntaba al archivo indicado. Es recomendable siempre cerrar el archivo, para evitar funcionamientos anómalos del código.

Ejemplo de uso.

```
{
SD.closeFile(fp); // close file de-assigning fp to it
}
```

Búsqueda de archivos

Encuentra el archivo en el directorio actual. Devuelve un '1' si existe y un '0' si existe pero no es una carpeta, si no existe devolverá un -1.

Ejemplo de uso.

```
{
const char* name=FileText;
int8_t fileFound;
fileFound=SD.isFile(name);
fileFound=SD.isFile(FileText);
}
```

Lectura de datos

En las instrucciones siguientes explicaremos como realizar una lectura de datos, cabe recordar el comentario realizado al principio de este capítulo, en él hacíamos referencia a la limitación del *buffer*. Es por ello, que los datos que se escriben o leen están limitados, si se quiere enviar un tamaño mayor, hay que dividirlo en diferentes paquetes de lo contrario los datos serán truncados.

Ejemplo de uso.

```
{
const char* name=FileText;
char* dataRead;
dataRead=SD.cat(name,3,17);//Almacena en buffer 17 caracteres después
de saltar 3 posiciones
dataRead=SD.cat(name,0,100);//Almacena en buffer 100 caracteres desde
el principio
dataRead=SD.catln(name,2,3);//Almacena en buffer 3 lineas después de
saltar las dos primeras
}
```

Escritura de datos

Hay muchas maneras de escribir datos. Una es escribir datos en un fichero indicando la posición donde empezar a escribir dentro del archivo. Otra forma, es escribiendo datos en un archivo al final del mismo. Una tercera forma es escribiendo en un archivo al final del mismo incluyendo un carácter EOL.

Ejemplo de uso.

```
{
const char* name=FileText;
uint8_t writeState;
writeState=SD.writeSD(name,hello,0);//Escribe hello en la posicion 0
writeState=SD.writeSD(FileText,hello,20);//Escribe hello en la
posición 20
writeState=SD.writeSD(FileText,hello,20,3);//Escribe hel en la
posición 20
writeState=SD.append(name,hello);//Escribe hello al final del a
rchivo
writeState=SD.appendln(FileText,hello);//Escribe  hello al final
del archivo con EOL
}
```

Número de líneas

Cuenta el número de líneas que hay en el archivo indicado. En realidad cuenta el número de \n que encuentra en el texto.

Ejemplo de uso.

```
{
const char* name=FileText;
int32_t numberLines;
numberLines=SD.numln(name);//Cuenta el número de líneas del archivo FileText
numberLines=SD.numln(FileText);//Cuenta el número de líneas del archivo FileText
}
```

Obtener el tamaño del archivo

De esta manera podemos saber el tamaño que tiene el archivo que hemos indicado, el valor será devuelto en número de *bytes*, si devuelve -1 es que ha ocurrido un error.

Ejemplo de uso.

```
{
const char* name=FileText;
int32_t sizeFile;
sizeFile=SD.getFileSize(name);//Obtiene el tamaño del archivo FileText
sizeFile=SD.getFileSize(FileText);//Obtiene el tamaño del archivo FileText
}
```

Gracias a las herramientas explicadas en esta sección se podría plantear un código alternativo que guardase de manera continua datos en la SD y que los enviase todos juntos cuando Matlab lo solicitase por ejemplo, o que esos datos fuesen introducidos directamente al ordenador cambiando la SD de Waspmote al ordenador, en definitiva, múltiples posibilidades que permiten movernos dentro de las diferentes funcionalidades que Waspmote ofrece en función de la aplicación que queramos realizar.

6.2. XBee 802.15.4

Se trata de un módulo que se conecta en la parte superior de la tarjeta Waspmote, junto con las antenas pertinentes, para de esta forma poder realizar una conexión mediante el protocolo 802.15.4.

Figura 6.1: Módulo XBEE 802.15.4 .

La siguiente información resume las características más importantes del módulo [2]. Los módulos XBee 802.15.4 cumplen con el estándar IEEE 802.15.4 que define el nivel físico y el nivel de enlace (capa MAC). A las funcionalidades aportadas por el estándar, los módulos XBee añaden ciertas funcionalidades como:

- Descubrimiento de nodos: se añade cierta información a las cabeceras de los paquetes de forma que se pueden descubrir otros nodos dentro de la misma red. Permite enviar

un mensaje de descubrimiento de nodos, de forma que el resto de nodos de la red responden indicando sus datos (Node Identifier, @MAC, @16 *bits*, RSSI).

- Detección de paquetes duplicados: Esta funcionalidad no se establece en el estándar y es añadida por los módulos XBee.

De cara a obtener tramas totalmente compatibles con el estándar IEEE 802.15.4 y de esta forma poder interoperar con otros chipsets han creado el comando XBee.setMacMode(m) mediante el cual podemos seleccionar en cada momento si queremos que los módulos utilicen un formato de cabecera totalmente compatible o por el contrario si queremos que puedan usar las opciones de descubrimiento de nodos y detección de paquetes duplicados.

Se proporciona cifrado mediante el algoritmo AES 128b. Concretamente mediante el tipo AES-CTR. En este caso el campo Frame Counter del frame tiene un ID único y se cifra toda la información contenida en el campo Payload que es el lugar del frame 802.15.4 donde almacenan los datos a enviar. De la forma en la que se han desarrollado las librerías para la programación del módulo activar cifrado es tan sencillo como ejecutar la función de inicialización y darle una clave para usarla en el cifrado.

```
{
xbee802.encryptionMode(1);
xbee802.setLinkKey(key);
}
```

Información extra sobre los sistemas de cifrado en redes sensoriales 802.15.4 y ZigBee puede ser consultada en la sección de Development de la web de Libelium, concretamente en el documento: 'Security in 802.15.4 and ZigBee networks'.

Capítulo 7

Conclusiones

Una vez se ha llegado al final del proyecto, resulta necesario analizar si se han cumplido los objetivos que se fijaron al inicio del mismo. Al comenzar el proyecto se tomaron muestras de la señal procedente de un sensor situado en el Laboratorio de Alta Tensión de la Universidad Carlos III de Madrid. Dichos datos constaban de tiempo y voltaje, con una longitud de 5000 muestras. Con el objetivo de poder comprobar la repetibilidad del microprocesador que se encargaría de la toma de datos, se tomó la determinación de hacer uso del generador de ondas TEKTRONIX AFG3252 del laboratorio, dicho generador solo admite señales con un máximo de 1000 puntos. Se procedió a la introducción de dichos datos en Matlab y a limitar la señal al núcleo central de 1000 puntos. Gracias a ello se pudo cumplir uno de los objetivos marcados, introduciendo los datos de la señal en el generador, se comprobó que la repetibilidad era la deseada.

Para poder adquirir la onda se tomó la decisión de usar un dispositivo denominado Waspmote. Este dispositivo cumple con todas las características que se tenían como objetivo, entre las que destacan: modularidad, bajo consumo, precio reducido y pequeño tamaño. La frecuencia del dispositivo resulta insuficiente para capturar todas las oscilaciones de 5 μs, no obstante la información que se requiere de la señal son los valores máximos que alcanza y el tiempo que la señal permanece en dichos niveles. No se pretende sobredimensionar el dispositivo ya que encareceriamos el coste de manera innecesaria, por lo que se recurre a una solución que se adapta a nuestros requisitos, un detector de envolvente.

Se usará como herramienta de apoyo al diseño del detector de envolvente el programa Multisim, de especial relevancia fue la elección del diodo para que fuese capaz de rectificar las oscilaciones negativas de la señal y seleccionar un valor RC adecuado, cuya resistencia no interfiriese con los diferente dispositivos conectados.

Por último se realizó una aplicación para el usuario en Matlab, si bien esta parte escapa a los objetivos que se habían marcado, nos pareció necesario realizar una interfaz más estética y que facilitase el uso del programa. De esta manera se realiza un proyecto completo, que abarca desde la señal tomada en el sensor hasta la interfaz de usuario.

Se puede considerar que todos los objetivos han sido cumplidos, obteniendo como resultado un conjunto que funciona, un dispositivo que es capaz de tomar los parámetros de duración y amplitud máxima de una señal rápida y que es precisamente lo que se pretendía. Añadiendo la interfaz de usuario y algunas ampliaciones que se han realizado como el uso de la tarjeta SD para guardar datos.

Bibliografía

[1] Documento de características de Waspmote: http://www.libelium.com/documentation/waspmote/waspmote-datasheet_esp.pdf

[2] Documento de características de los diferentes módulos: www.libelium.com/documentation/waspmote/waspmote-technical_guide_esp.pdf

[3] Manual completo ATmega1281: www.atmel.com/dyn/resources/prod_documents/doc2549.pdf

[4] Manual de programación SD Waspmote: www.libelium.com/documentation/waspmote/waspmote-sdcard-programming_guide.pdf

[5] Diseño de moduladores y demoduladores (Universidad de Sevilla) http://www.dte.us.es/personal/pfortet/practica5tbc.pdf

[6] The envelope detector (University of St. Andrews) http://www.st-andrews.ac.uk/~jcgl/Scots_Guide/RadCom/part9/page2.html

[7] Datasheet diodo 1N5818TR http://docs-europe.electrocomponents.com/webdocs/0fbc/0900766b80fbce4f.pdf

[8] Documento/manual Matlab Guide www.ingelec.uns.edu.ar/icd2763/tut.doc

[9] Función software detector de envolvente http://www.mathworks.com/matlabcentral/fileexchange/3142-envelope1-1/all_files

[10] Tutorial de Matlab Guide www.ingelec.uns.edu.ar/icd2763/tut.doc

[11] Tutorial de Matlab Guide http://web.usal.es/~gfdc/docencia/GuiSection.pdf

[12] Robert L. Boylestad y Louis Nashelsky.
Electrónica: Teoría de circuítos y dispositivos electrónicos.
PEARSON EDUCACIÓN, México, 2003.

[13] Eduard Bertran Albert.
Señales y sistemas de tiempo discreto.
Ediciones UPC, 2003.

[14] Arduino Uno http://arduino.cc/en/Main/ArduinoBoardUno

[15] ATmega328 http://www.atmel.com/Images/doc8161.pdf

ANEXO1

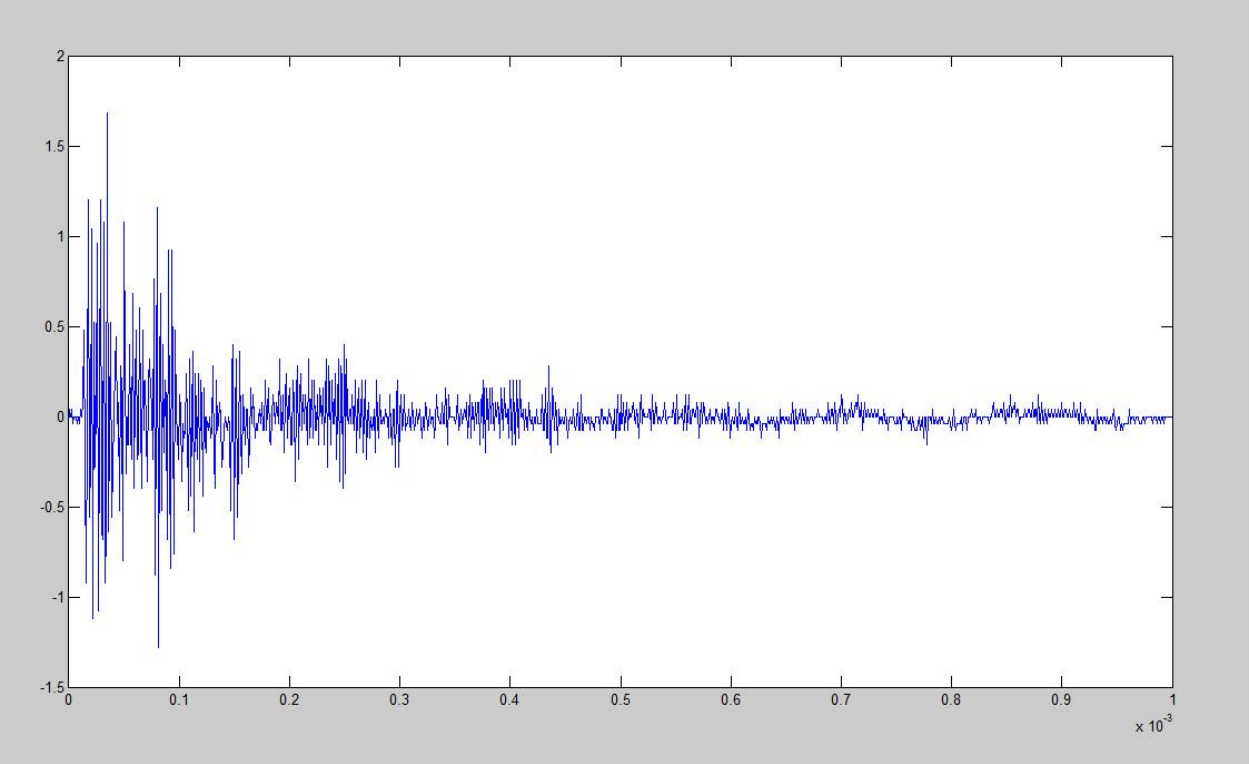

Figura 7.1: Señal modelo.

ANEXO 2

```
#define FASTADC 1
#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif
#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif


int ledPin=13;
int i=0, j[100];
uint8_t low, high;
unsigned long start=0, tmillis=0;
int ENTERO=0, flag=0;

void setup ()
{
//Activamos las comunicacion serie via USB
  USB.begin();
  USB.println('USB port started...');
}


void loop()
{
 //Encendemos el LED de la placa y esperamos que Matlab solicite datos
 digitalWrite(ledPin,HIGH);
 establishContact();
 //Apagamos el LED de la placa al enviarlos MATLAB la solicitud de datos
 digitalWrite(ledPin,LOW);

 //Activamos el flag, señal de que debemos esperar hasta que se active el trigger
 flag=1;

while (flag==1)
```

```
{
 //En cada ciclo que ocurre a la espera de la señal de trigger se comienza
 //a contar el tiempo.
 //Se toman las dos primeras muestras pretrigger indicando con ADMUX
 //los puertos usados.

start=millis();
i=1;
ADMUX = 0xD3;
j[i]=ENTERO;
ENTERO = AnalogReadDif();
i=i+1;
j[i]=ENTERO;

 //Se establece el nivel de trigger, en este caso 100, que corresponde a 500mV

if(ENTERO>100)
{
 //Si el trigger es lanzado, se toman los 18 datos siguientes, se vuelve a poner
 //el flag a 0.

flag=0;
while(i<20)
{
ADMUX = 0xD3;
i=i+1;
ENTERO = AnalogReadDif();
j[i]=ENTERO;
}
 //Guardamos el tiempo que hemos tardado en tomar los datos
 //Este tiempo será la diferencia entre el tiempo actual 'millis'
 //y el tiempo cuando se tomo el primer dato, 'start'

tmillis=(millis()-start);

 //Enviamos los datos por el puerto serie a Matlab

i=0;
while (i<20)
{
i=i+1;
USB.println(j[i]);
}
USB.println(tmillis);
delay(30);
```

```
USB.println(514);
}
}
}
```

```
//Función de lectura de datos en modo diferencial

int AnalogReadDif()
{
  int result;
  sbi(ADCSRA, ADSC);
  while (bit_is_set(ADCSRA, ADSC));
  low = ADCL;
  high = ADCH;
  result=(high << 8) | low;
  return result;
}

//Función de espera y comunicación con Matlab

void establishContact()
{
while (USB.available() <= 0)
{
    USB.println(8);
    delay(300);
}
USB.flush();
}
```

ANEXO 3

```
function varargout = PROYECTO_FINAL2(varargin)
% PROYECTO_FINAL2 M-file for PROYECTO_FINAL2.fig
%      PROYECTO_FINAL2, by itself, creates a new PROYECTO_FINAL2 or raises the
existing
%      singleton*.
%
%      H = PROYECTO_FINAL2 returns the handle to a new PROYECTO_FINAL2 or the handle
to
%      the existing singleton*.
%
%      PROYECTO_FINAL2('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PROYECTO_FINAL2.M with the given input arguments.
%
%      PROYECTO_FINAL2('Property','Value',...) creates a new PROYECTO_FINAL2
or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before PROYECTO_FINAL2_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to PROYECTO_FINAL2_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 PROYECTO_FINAL2

% Last Modified by GUIDE v2.5 01-Mar-2012 19:21:29

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @PROYECTO_FINAL2_OpeningFcn, ...
                   'gui_OutputFcn',  @PROYECTO_FINAL2_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 PROYECTO_FINAL2 is made visible.
function PROYECTO_FINAL2_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 PROYECTO_FINAL2 (see VARARGIN)
%axes(handles.logo)
axes(handles.logo)
handles.imagen=imread('logo1','jpg')
; imagesc(handles.imagen)
axis off


axes(handles.axes3)
handles.imagen=imread('sin_datos','jpg')
; imagesc(handles.imagen)


axes(handles.axes4)
handles.imagen=imread('sin_datos','jpg')
; imagesc(handles.imagen)
axis off


% Choose default command line output for
PROYECTO_FINAL2 handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes PROYECTO_FINAL2 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = PROYECTO_FINAL2_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 button press in nuevos_datos.
function nuevos_datos_Callback(hObject, eventdata, handles)
% hObject    handle to nuevos_datos (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
f=[];
w=0;

s1 = serial('COM4','Baudrate',38400);    % define serial port
set(s1, 'terminator', 'LF');
set(s1,'Timeout',100);
% define the terminator for println
fopen(s1);

try
```

```
while (w~=8)


    display('Collecting data');

    fprintf(s1,'%d\n','8');

    w=fscanf(s1,'%d')

end

i=1;
flag1=0;
tiempo1=0;
contador1=0;

while (flag1==0)


    f(i)=fscanf(s1,'%d')

    if (f(i)==514)
       %Dado que no se pueden obtener valores negastivos por la
       %incorporación del detector de envolvente (debido al diodo) es
       %imposible tener un valor superior a 512, por ello tomamos 514 como
       %código para indicar el fin de la transmisión
       tiempo1=f(i-1);
       contador1=(i-2);
       %No contamos ni el dato del tiempo, ni el código de fin de
       %transmisión
       i=1;
       flag1=1;
    end
    i=i+1;

end


fclose(s1);
if (tiempo1==0)
    tiempo1=1;
end
%Si fuese menor de 1ms aproximariamos a la cifra más cercana que es 1ms
unid_time1=tiempo1/contador1;
j=contador1;
for(i=1:1:j)
    tiempo1(i)=unid_time1*i
end

plot(handles.axes3,tiempo1(1:(i-2)),0.005*f(1:(i-2)));
%Como se puede comprobar en el propio plot se transforman los datos a
%voltaje multiplicandolo por el factor 0.005


catch me
    display('ERROR')
    fclose(s1);
end
```

```
% --- Executes on button press in eliminar2.
function eliminar2_Callback(hObject, eventdata, handles)
% hObject    handle to eliminar2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes3)
handles.imagen=imread('sin_datos','jpg')
; imagesc(handles.imagen)
axis off;


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

load PROYECTOSIGNAL;

plot(PROYECTOSIGNAL(:,1),PROYECTOSIGNAL(:,2));

hold off;

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

% En esta parte del código introducimos la creación de la envolvente por
% software
load PROYECTOSIGNAL;
[up,down] = envelope(PROYECTOSIGNAL(:,1),PROYECTOSIGNAL(:,2),'spline');
hold on;

plot(PROYECTOSIGNAL((1:950),1),up(1:950),'r');
plot(PROYECTOSIGNAL((1:950),1),down(1:950),'r');


hold off;

% --- Executes on button press in eliminar3.
function eliminar3_Callback(hObject, eventdata, handles)
% hObject    handle to eliminar3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes4)
handles.imagen=imread('sin_datos','jpg')
; imagesc(handles.imagen)
axis off;
```

ANEXO 4

```
function [up,down] = envelope(x,y,interpMethod)

%ENVELOPE gets the data of upper and down envelope of the known input (x,y).
%
%   Input parameters:
%    x               the abscissa of the given data
%    y               the ordinate of the given data
%    interpMethod    the interpolation method
%
%   Output parameters:
%    up      the upper envelope, which has the same length as x.
%    down    the down envelope, which has the same length as x.
%
%   See also DIFF INTERP1

%   Designed by: Lei Wang, <WangLeiBox@hotmail.com>, 11-Mar-2003.
%   Last Revision: 21-Mar-2003.
%   Dept. Mechanical & Aerospace Engineering, NC State University.
% $Revision: 1.1 $  $Date: 3/21/2003 10:33 AM $

if length(x) ~= length(y)
    error('Two input data should have the same length.');
end

if (nargin < 2)|(nargin > 3),
 error('Please see help for INPUT DATA.');
elseif (nargin == 2)
    interpMethod = 'linear';
end


% Find the extreme maxim values
% and the corresponding indexes
%---------------------------------------------------
extrMaxValue = y(find(diff(sign(diff(y)))==-2)+1);
extrMaxIndex =   find(diff(sign(diff(y)))==-2)+1;


% Find the extreme minim values
% and the corresponding indexes
%---------------------------------------------------
extrMinValue = y(find(diff(sign(diff(y)))==+2)+1);
extrMinIndex =   find(diff(sign(diff(y)))==+2)+1;


up = extrMaxValue;
up_x = x(extrMaxIndex);

down = extrMinValue;
down_x = x(extrMinIndex);

% Interpolation of the upper/down envelope data
%---------------------------------------------------
up = interp1(up_x,up,x,interpMethod);
down = interp1(down_x,down,x,interpMethod);
```