Universidad del País Vasco
Euskal Herriko Unibertsitatea

ZTF-FCT
Zientzia eta Teknologia Fakultatea
Facultad de Ciencia y Tecnología

Gradu Amaierako Lana / Trabajo Fin de Grado
Ingenieritza Elektronikoko Gradua / Grado en Ingeniería Electrónica

Configuración de una red de sensores inalámbrica para adquisición de datos en tiempo real. Acondicionamiento y procesado de datos mediante FPGAs.

Egilea/Autor:
Adur Ayerza Zubiria
Zuzendaria/Director/a:
Estibaliz Asua Uriarte
Inés del Campo Hagelstrom

Leioa, 2015ko ekainaren 22a /Leioa, 22 de junio de 2015

Índice

1. Introducción y Objetivos.

1.1 Objetivos del trabajo.

El análisis de las vibraciones es utilizado comúnmente en el mantenimiento de los motores. Es por ello que en la industria resulta imprescindible la monitorización de todas las máquinas que estén en funcionamiento; ya que el fallo de uno de los equipos puede causar grandes pérdidas. La adquisición de las vibraciones mediante una red de sensores y el posterior procesado de datos permiten monitorizar en tiempo real el estado de cada uno de los motores [1].

Debido a la importancia del desarrollo de tales sistemas, este trabajo tiene como objetivo diseñar e implementar un sistema para la medición de vibraciones y su posterior procesado para el análisis de los datos en el dominio de la frecuencia. La toma de datos éstos se realizará mediante una red de sensores inalámbricos, haciendo uso de un acelerómetro, y el procesado mediante una FPGA.

La primera tarea, consistirá en entender por qué puede resultar útil el estudio de las vibraciones y en estudiar el proceso que ha de seguirse para tratar los datos. A continuación, se explicará cómo crear una red de sensores inalámbricos, analizando diferentes sensores y métodos para el envío de datos vía antena. A continuación, se hará una introducción a las FPGAs, comentando las mayores ventajas que ofrecen y centrándose en la familia de dispositivos que se utilizará en este proyecto. Finalmente, se explicarán detalladamente todos los pasos que se han seguido para desarrollar el sistema de medición y análisis de las vibraciones.

1.2 Estado del arte.

Desde el desarrollo de los dispositivos digitales, el análisis de datos ha resultado fundamental para el estudio de sistemas en tiempo real. El análisis de las vibraciones permite afinar instrumentos musicales, detectar el mal funcionamiento de motores, estudiar las propagaciones de vibraciones en edificios, etc.

Respecto al procesado de datos, las FPGAs resultan muy útiles para este fin. Estos dispositivos están muy extendidos y son empleados en sectores como el aeroespacial, en medicina, en el mundo militar, procesado de video en tiempo real, etc. La gran ventaja que ofrecen éstos, es el hecho de que son programables a nivel de hardware. Mediante un lenguaje HDL (Hardware Description Language), se describe la arquitectura del sistema y de esta manera se consigue desarrollar un componente diseñado exclusivamente para un fin concreto. Por ejemplo es posible diseñar una aplicación que procese varios canales de forma paralela; siendo esto una tarea muy compleja de implementar en un microprocesador.

En lo que respecta al uso de redes de sensores inalámbricos, si se desea estudiar por ejemplo, la calidad del aire en una ciudad, o una cosecha, se deben utilizar muchos sensores, resultando imprescindible que se encuentren interconectados entre ellos. Además, siendo inalámbricos se evitan problemas de accesibilidad. Plataformas como el *Waspmote* permite este tipo de aplicaciones [5].

1.3 Dispositivos a utilizar.

En este trabajo se utilizará la plataforma *Waspmote* de *Libelium* para la adquisición de vibraciones. Al estar diseñados para emplearlos en una red de sensores, disponen de librerías para la utilización de módulos de comunicación, simplificando en gran medida la tarea de programación.

El procesado de datos se efectuará mediante la FPGA *Spartan 3E* de *Xilinx.* Se disponen de códigos predefinidos que facilitan al usuario implementar su aplicación.

En este trabajo, se utilizara el programa *LabView* como una herramienta de desarrollo para establecer una comunicación entre los dos dispositivos. El objetivo final, sería realizar el envío y recepción de datos mediante un hardware inalámbrico entre *Waspmote* y FPGA, pero esta parte queda fuera del alcance de este proyecto. Por ello, la comunicación se realizará mediante un PC, como se muestra en la imagen 1. Además, este software ofrece la posibilidad de mostrar en pantalla los datos obtenidos mediante una interfaz gráfica atractiva que es de ayuda para verificar el correcto funcionamiento del sistema.

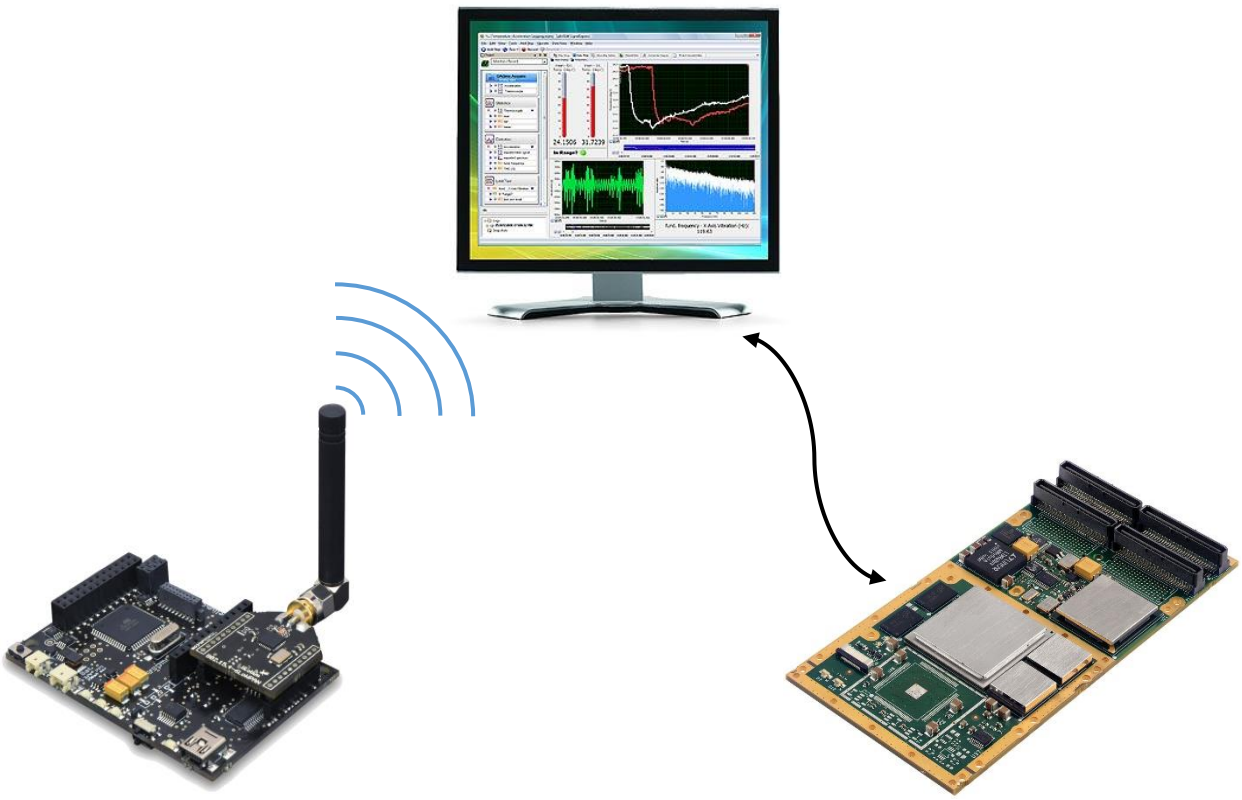

Imagen 1. Esquema general de la aplicación.

2. Vibraciones, acelerómetros y FFT.

2.1 Análisis de las vibraciones

Una vibración es la oscilación repetitiva de un objeto alrededor de una posición de equilibrio, Por ejemplo, al tocar una cuerda de la guitarra, ésta empieza a vibrar acorde a su frecuencia de oscilación, que está determinada por el material de la cuerda y por la tensión a la que está sujeta.

Los objetos vibran a determinadas frecuencias dependiendo de las circunstancias a las que están sometidas, por lo que es posible determinar las condiciones de los objetos en vibración partiendo de sus frecuencias de oscilación. De esta manera, es posible analizar su correcto funcionamiento.

Dependiendo de la situación, las vibraciones pueden ser tanto positivas como negativas.

- **Vibraciones positivas:**

Este es el caso donde se busca que un cuerpo vibre a una frecuencia concreta. El mejor ejemplo es un instrumento musical. Para un músico es fundamental que su instrumento esté perfectamente afinado, pero ¿cómo puede ayudar el análisis de las vibraciones con la afinación?

Los instrumentos musicales poseen cavidades resonantes, lugar donde se producen las ondas sonoras, que a su vez son ondas de presión que viajan por el aire. Entonces, estas ondas hacen vibrar el instrumento. Obteniendo la frecuencia de estas vibraciones, se determina si está afinado o no.

- **Vibraciones negativas:**

Es el caso donde las vibraciones producen efectos no deseados en un equipo. Un ejemplo de la importancia del análisis de las vibraciones se encuentra en la industria. En cualquier fábrica existe algún motor, y los motores al girar, vibran (ver imagen 2). Por tanto, cualquier vibración que este fuera de lo normal, indica que una máquina no está trabajando correctamente.

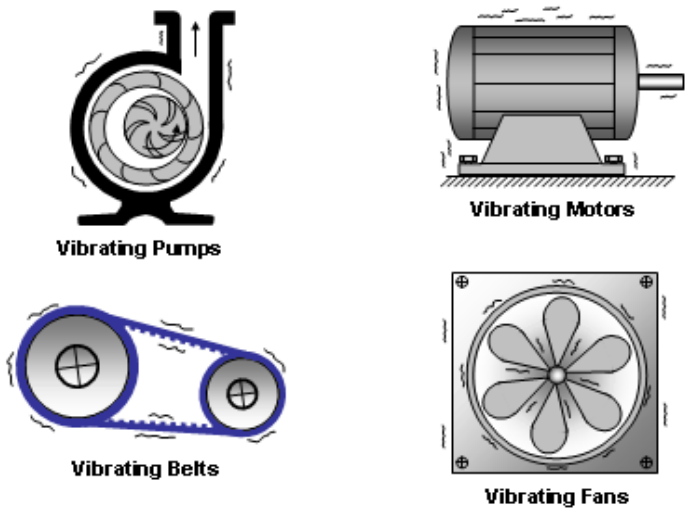

Imagen 2. Vibraciones en motores.

Estas anomalías en las vibraciones puede deberse a diversas causas, pero las generales suelen ser las siguientes.

- **Fuerzas repetitivas:**

Una fuerza que se repite una y otra vez da a lugar a una vibración. Estas pueden ser formadas por un balanceamiento incorrecto del motor, es decir, que el centro de masa no se encuentre en el eje de rotación. Otras causas pueden ser el mal alineamiento entre el motor y el eje o el desgaste de los engranajes o rodamientos.

- **Holgura:**

Si el motor no está bien sujeto, las vibraciones que son tolerables pueden convertirse en excesivas. Estos casos pueden ocurrir cuando los cojinetes están colocados en huecos demasiado grandes, por la corrosión o por grietas en la estructura.

- **Resonancia:**

Al igual que cuando un niño se columpia en el parque, si se le empuja en el momento correcto, el niño cada vez llegará más alto con el columpio. A esto se le llama frecuencia de resonancia y es la frecuencia donde el sistema alcanza el máximo grado de oscilación.

Todo cuerpo o sistema tiene una o varias frecuencias de resonancia, por tanto, los motores también. Entonces, ¿qué ocurre cuando un objeto oscila en la misma frecuencia que la de resonancia? Es este caso el que hay que evitar a toda costa ya que cuando esto ocurre el objeto en cuestión oscila cada vez más y más. Las vibraciones pueden causar daños severos en muy poco tiempo. Hay varios ejemplos donde puentes enteros han colapsado por este efecto.

2.2 Acelerómetros.

Un acelerómetro es un sensor que mide las aceleraciones. El principio básico de este instrumento es muy sencillo, se basa en la segunda ley de Newton.

$$F = ma \tag{1}$$

Fuerza y aceleración están unidas mediante la ecuación 1, por lo que si se posee una masa y se conoce la fuerza que se le está aplicando, se obtiene la aceleración.

Para poder implementar este instrumento en un chip de tamaño reducido, se ideó el acelerómetro capacitivo. Se crean pequeños condensadores basculantes, dos placas conductoras fijas y en medio otra placa conductora que está fijada a un muelle (ver imagen 3). De esta manera, cuando la placa que está fijada al muelle se desplaza por el efecto de una fuerza externa, cambia la capacitancia que forman las tres placas. Midiendo este valor se puede determinar cuánto se ha desplazado la parte móvil y por ello, calcular la aceleración.

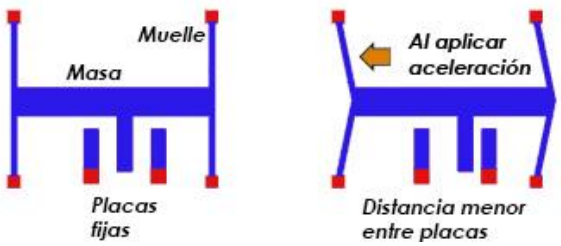

Imagen 3. Funcionamiento del acelerómetro.

Gracias a este diseño, es posible añadir este sensor en aparatos electrónicos del día a día, como es el caso de los teléfonos móviles.

2.2.1 Modo de empleo.

Para medir correctamente las vibraciones se debe asegurar que los acelerómetros han sido colocados correctamente, situándolos en los ejes de las vibraciones a medir. En la imagen 4, se muestra su posición para el caso de un motor. Es recomendable que el contacto entre sensor y el objeto a estudiar se haga mediante unas gomas que estén bien sujetas, para así poder transferir las vibraciones sin ninguna distorsión.

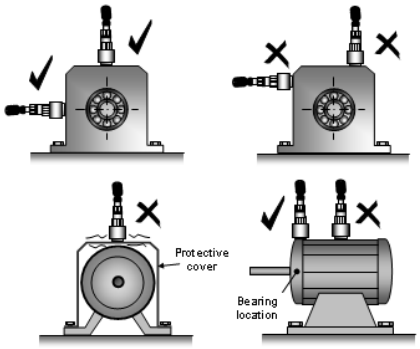

Imagen 4. Cómo colocar los acelerómetros.

2.3 Transformada de Fourier y FFT.

Los datos que se obtienen de los acelerómetros están representados en el dominio del tiempo, pero muchas veces resulta más útil representar los datos en el dominio de la frecuencia. Haciendo uso de la transformada de Fourier, se trasladan funciones del dominio del tiempo al de frecuencia, como se muestra en la imagen 5.

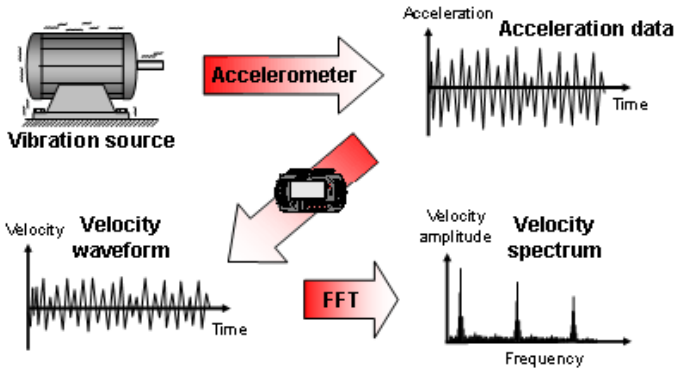

Imagen 5. Procesado de los datos.

2.3.1 Transformada de Fourier y transformada discreta de Fourier.

La transformada de Fourier es una herramienta matemática que permite trasladar las señales del dominio del tiempo, al dominio de la frecuencia. Siendo f(x) una señal en el dominio del tiempo, aplicando la ecuación 2 se obtiene su respectiva función, g(w), en el dominio de la frecuencia.

$$g(w) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-iwx} dx, \;\; f(x) \in \mathbb{C} \tag{2}$$

En el caso de una función periódica, como puede ser el caso de una nota musical, la transformada de Fourier se puede simplificar y reescribirse como una serie, llamada la serie de Fourier.

$$f(x) \approx \sum_{n=-\infty}^{\infty} C_n e^{2\pi i \frac{n}{T} x}, C_n \in \mathbb{C} \tag{3}$$

En la ecuación 3, siendo T el periodo de f(x) y C_n los coeficientes de Fourier, se observa que la función f(x) se puede escribir mediante una suma infinita de exponenciales complejas. Los coeficientes de Fourier determinan el módulo de éstas. Para el cálculo de éstos se utiliza la ecuación 4.

$$C_n = \int_{-T/2}^{T/2} f(x) e^{-2\pi i \frac{n}{T} x} dx \tag{4}$$

Lo que realmente destaca en la transformada de Fourier es que el resultado es el espectro de la frecuencia de una señal. En el caso que se quiera analizar un sonido, al aplicar la serie de Fourier, se obtienen los coeficientes C_n, indicando la amplitud de cada seno que compone ese sonido. De esta manera se puede identificar qué frecuencias y con qué amplitud se compone esa señal.

Esta herramienta es muy útil en electrónica en el ámbito de procesamiento de señales, pero los dispositivos de procesamiento son digitales, por lo que trabajan con señales discretas. En este caso se debe utilizar la transformada discreta de Fourier (DFT). La función de entrada, x_n, debe ser discreta, de duración finita e implica que el segmento que se analiza es un periodo de una señal periódica que se extiende de forma infinita.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn}, \quad \begin{matrix} k = 0, \dots, N-1 \\ x_n \in \mathbb{C} \end{matrix} \tag{5}$$

2.3.2 Funciones de ventana.

Las discontinuidades en los extremos de la señal a analizar, pueden generar distorsión en los resultados obtenidos. Las funciones de ventanas no son más que distintas funciones que al multiplicar con la señal a tratar, suavizan los bordes (ver imagen 6) para reducir este efecto no deseado. Existen varios tipos de ventanas, cada una de ellas con sus respectivas características. Las ventanas más comunes son las de Hanning, Hamming y Blackman.

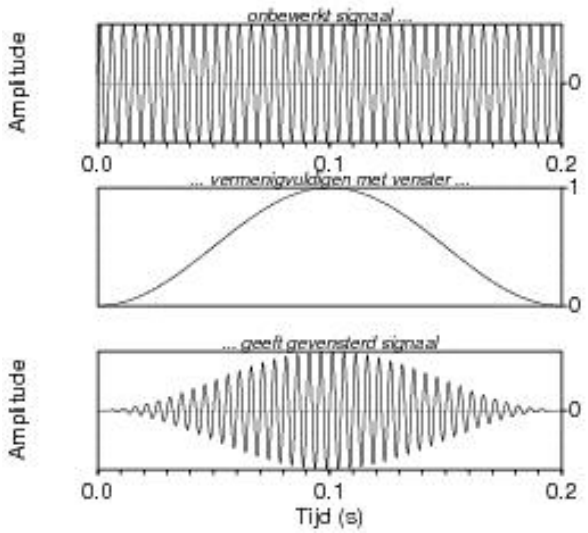

Imagen 6. Funciones de ventana.

2.3.3 Frecuencia de Nyquist.

Otro parámetro a tener en cuenta a la hora de realizar la DFT, es la frecuencia de muestreo, es decir, el número de muestras que se toma por unidad de tiempo de una señal contínua. Para determinarla se tendrá en cuenta la frecuencia de Nyquist (f_N).

El teorema de Nyquist-Shannon determina que para poder replicar con exactitud la forma de una señal, la frecuencia de muestreo debe ser superior al doble de la frecuencia máxima a muestrear (ecuaCIÓN6).

$$f_s > 2f_N \tag{6}$$

Siendo f_s la frecuencia de muestreo y f_N la frecuencia de Nyquist, ésta última determina la frecuencia crítica. Una vez se ha fijado una tasa de muestreo, todos los componentes de mayor frecuencia no serán correctamente muestreados y se dará el efecto 'aliasing', donde una señal de mayor frecuencia se confunde con una de menor frecuencia, obteniendo de esta manera unos resultados incorrectos.

2.3.4 Ejemplo de uso de la transformada de Fourier.

Para entender mejor la Transformada de Fourier, se empleará un ejemplo. En la imagen 7, se observa una señal periódica, f(x), y el objetivo es determinar qué frecuencias componen esa señal y con qué amplitud.

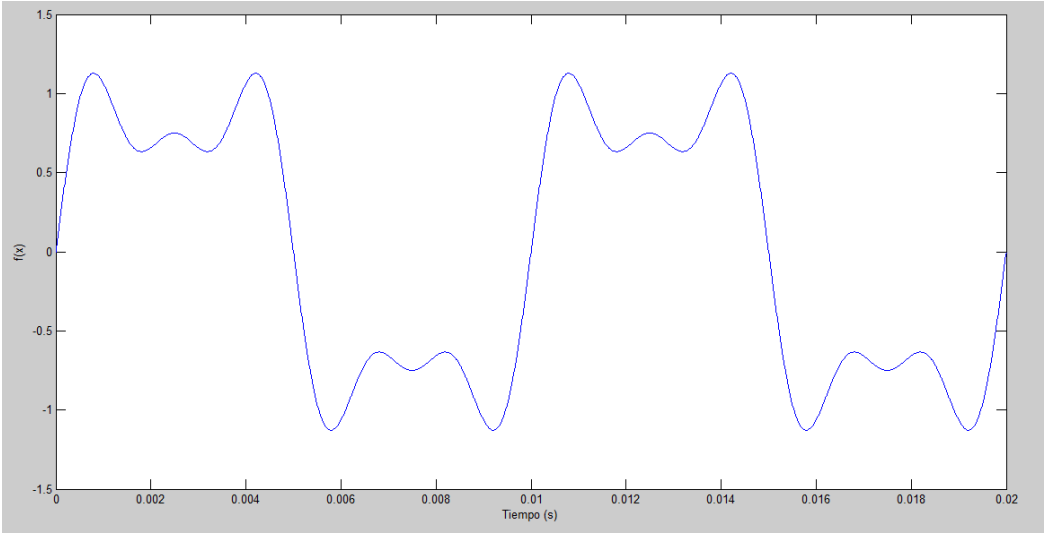

Imagen 7. Función periódica f(x).

Con la representación en el tiempo de la señal, no se puede determinar qué suma de senos da como resultado la señal f(x), pero como se ha visto, se puede utilizar la DFT para ello. En la imagen 8 se muestra el resultado de esta operación.

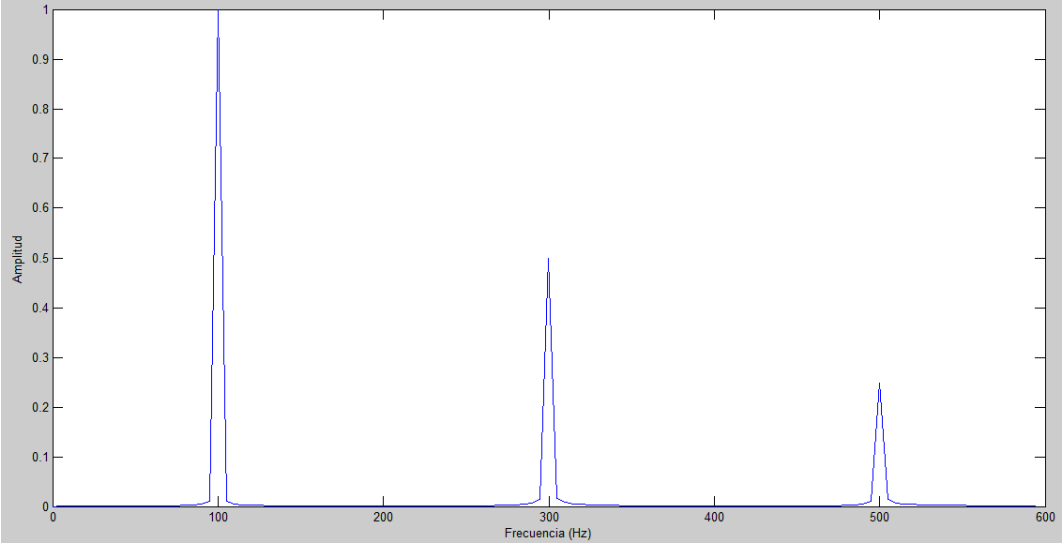

Imagen 8. DFT de la señal f(x).

Como se puede ver, son tres las frecuencias que componen la señal f(x).

$$f_1 = 100Hz, \;\; A_1 = 1$$
$$f_2 = 300Hz, \;\; A_2 = 0.5$$
$$f_3 = 500Hz, \;\; A_3 = 0.25$$

Por lo tanto, la señal f(x) se obtiene como:

$$f(x) = sin(2\pi \cdot 100x) + 0.5sin(2\pi \cdot 300x) + 0.25sin(2\pi \cdot 500x)$$

A la hora de representar los resultados de la transformada, de las N muestras que se obtienen, solo la mitad son de interés. La otra mitad corresponden al complejo conjugado de la primera mitad, y como las señales que se analizarán son reales, la segunda mitad resulta simétrica a la primera.

La resolución depende de dos factores, la frecuencia de muestreo y el número de muestras.

$$\Delta f = \frac{f_s}{N} \tag{7}$$

2.3.5 La transformada rápida de Fourier - FFT.

En cualquier dispositivo electrónico, la velocidad de procesamiento es de suma importancia. Calcular la transformada de Fourier mediante la ecuación 5 no resulta eficiente, por lo que se desarrolló un nuevo algoritmo mucho más eficaz, la transformada rápida de Fourier o más conocida como FFT. Este nuevo algoritmo reduce sustancialmente el tiempo requerido para realizar el cálculo. Aplicando la ecuación de la DFT, hacen falta N^2 operaciones, sin embargo la FFT sólo requiere de N·log(N) operaciones, siendo N el número muestras.

El algoritmo consiste en descomponer la transformada a tratar en otras más simples, y éstas a su vez hasta llegar a transformadas de dos elementos, donde el índice *n* de la ecuación 5 solo puede tomar valores entre 0 y 1.

En 1965, Cooley y Tukey popularizaron este algoritmo que hoy es comúnmente conocido como la FFT. Este algoritmo es mucho más eficiente para un número grande de muestras y el único requerimiento es que el número de muestras sea una potencia de 2.

El algoritmo más famoso para el cálculo de la FFT es algoritmo de diezmado en el tiempo (DIT). Éste se basa en dividir la señal de entrada en dos partes, por un lado los coeficientes pares y por el otro los impares. A continuación se explicará este algoritmo.

Para simplificar las expresiones, se utilizará la siguiente notación.

$$W_N = e^{-\frac{2\pi i}{N}} \tag{8}$$

El primer paso es separar las muestras de la ecuación 5 entre pares e impares.

$$X_k = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} W_N^{(2n)k} + \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} W_N^{(2n+1)k} \tag{9}$$

Expandiendo la ecuación 9.

$$X_k = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} W_N^{(2n)k} + \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} W_N^{(2n)k} W_N^k \tag{10}$$

Se reescribe el coeficiente $W^{(2n)k}$.

$$W_N^{(2n)k} = e^{-\frac{2\pi i}{N}2nk} = e^{-\frac{2\pi i}{N}2nk*\frac{1/2}{1/2}} = e^{-\frac{2\pi i}{N/2}nk} = W_{N/2}^{nk} \tag{11}$$

Se reescribe la ecuación 10.

$$X_k = \underbrace{\sum_{n=0}^{\frac{N}{2}-1} x_{2n} W_{N/2}^{nk}}_{Ec.\ DFT} + \underbrace{\sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} W_{N/2}^{nk}}_{Ec.\ DFT} W_N^k \tag{12}$$

Como se ve en la ecuación 12, se obtienen dos definiciones de la DFT, pero con la mitad de las muestras.

$$X_{k,p} = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} W_{N/2}^{nk}; \qquad X_{k,i} = \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} W_{N/2}^{nk} W_N^k \tag{13}$$

Por lo tanto, se consigue la primera ecuación que permitirá calcular la FFT.

$$X_k = X_{k,p} + X_{k,i} W_N^k, \qquad k = \{0, \dots, \frac{N}{2} - 1\} \tag{14}$$

Para calcular la otra mitad se utilizará la periodicidad de la DFT.

$$X_{k+\frac{N}{2},p} = X_{k,p} \qquad X_{k+\frac{N}{2},i} = X_{k,i} \tag{15}$$

Y se obtiene la segunda ecuación.

$$X_{k+\frac{N}{2}} = X_{k,p} - X_{k,i}W_N^k, \qquad k = \{0, \dots, \frac{N}{2} - 1\} \tag{16}$$

Como resultado, se logran dos ecuaciones que al aplicarlos en una rutina recursiva dan como resultado la DIT FFT de la señal.

$$\begin{cases} X_k = X_{k,p} + X_{k,i}W_N^k \\ X_{k+\frac{N}{2}} = X_{k,p} - X_{k,i}W_N^k \end{cases} \qquad k = \{0, \dots, \frac{N}{2} - 1\} \tag{17}$$

En la imagen 9 se observa cómo se aplica este algoritmo. Partiendo de DFTs de dos puntos y separando las muestras pares de las impares, se consigue realizar la transformada de Fourier pero de una manera más eficiente. Cada bloque DFT, se conoce también como mariposa (butterfly), y en cada una de ellas se realiza una multiplicación y dos sumas complejas.

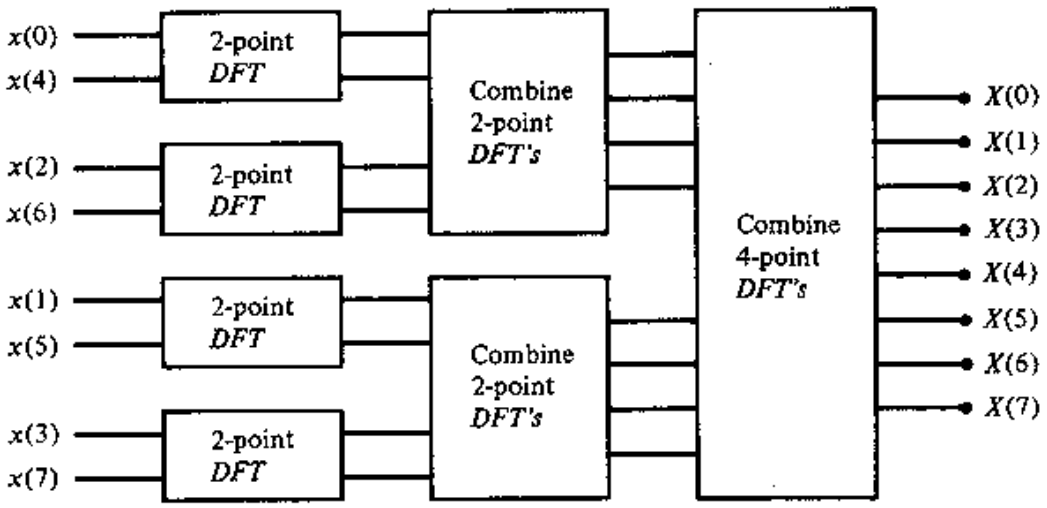

Imagen 9. Proceso del cálculo de la FFT

Una vez se ha comprendido la importancia del análisis de las vibraciones y de cómo realizar el adecuado análisis de éstos, se procederá al estudio de la configuración de una red de sensores inalámbrica.

3. Configuración de una red de sensores inalámbrica.

Uno de los objetivos de este trabajo es crear una red de sensores para medir vibraciones. No obstante, además de estudiar una plataforma en concreto (*Waspmote*) para la creación de esta red, se analizarán diversos sensores y su modo de empleo.

3.1 ¿Qué es un *Waspmote*?

Es una tarjeta de desarrollo programable (ver imagen 10) comercializado por la compañía *Libelium*, diseñado especialmente para crear redes de sensores inalámbricos, ya que ofrecen una gran autonomía. Viene equipado con un microprocesador ATmega1281 que trabaja a una frecuencia de 14,7 MHz. Lo que le convierte en especial para el uso como red de sensores es que posee un consumo muy bajo. Con el adecuado código, podría estar obteniendo y enviado datos durante un año.

Su versatilidad también lo convierte en un producto muy atractivo. La empresa *Libelium* tiene a la venta diferentes tipos de tarjetas de expansión que se le añaden al *Waspmote* para cada tipo de sensores y entornos. Por ejemplo, existen tarjetas específicas para uso agrícola, detección de diferentes gases, radiaciones y etc. Además, éstos vienen diseñados con un zócalo específico para la conexión del módulo de la antena (XBee sockets). Gracias a ésta, crear una red de sensores inalámbricos es una tarea sencilla.

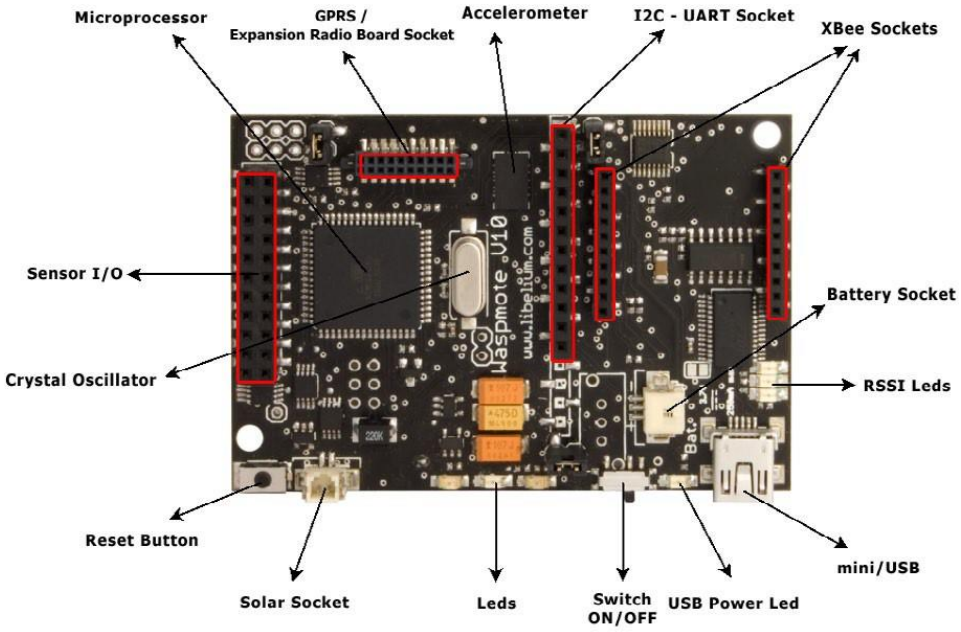

Imagen 10. Waspmote.

3.2 Partes de la red inalámbrica de sensores.

Para crear una red de sensores, además del *Waspmote*, hacen falta otros componentes. El *Waspmote* es la plataforma que permitirá ejecutar la aplicación que se desea desarrollar. Se conectará éste vía USB al PC para programarlo, y una vez comprobado su correcto funcionamiento se desconectará y se comunicará con el PC vía inalámbrica mediante las antenas (*XBee*). Además, para que el PC pueda recibir esta información, se añadirá un dispositivo receptor (*Gateway)* que ira conectado al PC. Se pueden utilizar tantos

Waspmotes como se deseen, comunicándose todos ellos vía inalámbrica con el mismo *Gateway*.

A continuación se describen más detalladamente estos elementos.

- **XBee:**

Los módulos *XBee* (ver imagen 11) son los dispositivos de radio que realizan el envío y la recepción de datos. Existen varios módulos distintos con diferentes protocolos de comunicación.

- **Gateway:**

El *Gateway* (ver imagen 12) es un adaptador que permite conectar los *XBee* con un ordenador mediante el puerto USB. De esta manera es posible recibir los datos en un PC.

Imagen 12. XBee..

Imagen 11. Gateway.

- **Tarjetas de expansión:**

Estas tarjetas ofrecen la posibilidad de añadir fácilmente diferentes tipos de sensores a la red, puesto que no hará falta calibrarlos ni realizar tareas suplementarias. Con una línea de código, se obtendrá el valor de la medida con su correspondiente unidad.

Cada una de estas tarjetas está diseñada para un entorno diferente. Por ejemplo, la tarjeta de agricultura ofrece sensores de humedad, presión atmosférica, humedad de la tierra, crecimiento de los tallos y frutas, etc.; o la tarjeta eventos, que incluye sensores de movimiento, luz, temperatura, vibraciones, fuerza, etc.

El único aspecto a tener en cuenta, es colocar el sensor en su zócalo correspondiente, dado que cada uno está adaptado para un tipo de sensor en concreto.
Esta información está descrita en la hoja de características de cada tarjeta de extensión.

Imagen 13. Tarjeta de expansión de eventos.

3.3 XBee: Protocolos, configuración e implementación.

Los módulos *XBee* permiten crear una comunicación entre los *Waspmotes* y el *Gateway*. En este proyecto, se disponen de módulos "*XBee Series 2*", que permiten dos protocolos de comunicación: 802 y *ZigBee*. Se utilizará el protocolo ZigBee.

3.3.1 Protocolo ZigBee.

Este protocolo es usado en la radiodifusión digital de bajo consumo. Su objetivo son las aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y maximización de vida útil de la batería. Por tanto, es importante utilizar este protocolo para crear redes de sensores. La velocidad de envío de datos es de 250 kbits/s y utiliza la banda libre de los 2.4GHz, también utilizado por los servicios WIFI.

Otra de las grandes ventajas aparte del consumo, es la posibilidad de configuración en red de malla. Mediante esta configuración, podemos tener interconectados hasta 65535 nodos.

3.3.2 Implementación de una red ZigBee.

Previa a la creación de una red *ZigBee,* se debe tener en cuenta que cada módulo *XBee* se puede configurar de tres maneras diferentes.

- Coordinador: Debe existir un coordinador por cada red que se cree. Sus funciones son las de encargarse de controlar la red y los caminos que deben seguir los dispositivos para conectarse entre ellos.

- Router: Permite interconectar los dispositivos que se sitúan por debajo de él en la topología de la red.

- End-Device: Posee la funcionalidad necesaria para comunicarse con su nodo padre (el coordinador o un router), pero no puede transmitir información destinada a otros dispositivos. De esta forma, este tipo de nodo puede estar dormido la mayor parte del tiempo, aumentando la vida media de sus baterías.

En el caso que nos ocupa, el coordinador irá situado en el Gateway y solo se dispondrá de routers, ya que hubo problemas a la hora de configurar los end-devices, y lógicamente, los routers son válidos como nodos finales también. De esta manera, serán los routers quienes envíen la información al coordinador. Éste, al estar conectado en el *Gateway,* enviará la información recibida al PC mediante el puerto USB.

3.3.3 Configuración de los XBee (X-CTU).

Para poder utilizar los módulos de *XBee* en la red de sensores, el primer paso deberá ser su configuración. Para ello, se utilizará el programa X-CTU, un software gratuito que permite configurar los distintos módulos *XBee*.

Para empezar, se colocará el módulo a programar en el *Gateway* y éste conectado al PC mediante USB. Una vez hecho esto, se debe conocer el puerto al que está conectado así como sus parámetros, para que el programa pueda leer correctamente el *XBee*. Como se utilizará el protocolo *ZigBee*, la velocidad de comunicación es de 38.400 baudios, (valor de símbolos por segundo en una comunicación digital). El resto de valores por defecto son los indicados en la imagen 14. Pulsando el botón 'Test/Query' el programa intentará conectarse con el *XBee,* de esta manera se conocerá si se han introducido correctamente los parámetros.

Una vez que el programa haya detectado el *XBee*, se debe conectar a él, mediante la pestaña 'Modem Configuration' y pulsando el botón 'Read'. Si todo es correcto, en la ventana aparecerán varios parámetros que permiten su modificación.

Cuando el programa haya accedido al *XBee*, lo primero a decidir es qué función ocupará el módulo conectado: Coordinator, router o end-device. Al decidir utilizar los módulos como coordinador y router, se deben escoger las opciones 'ZigBee Coordinator API' y 'ZigBee Router API' en el cuadro 'Function Set' como se indica con color morado en la imagen 15.

Una vez seleccionado el tipo de función, comienza la configuración de los parámetros necesarios.

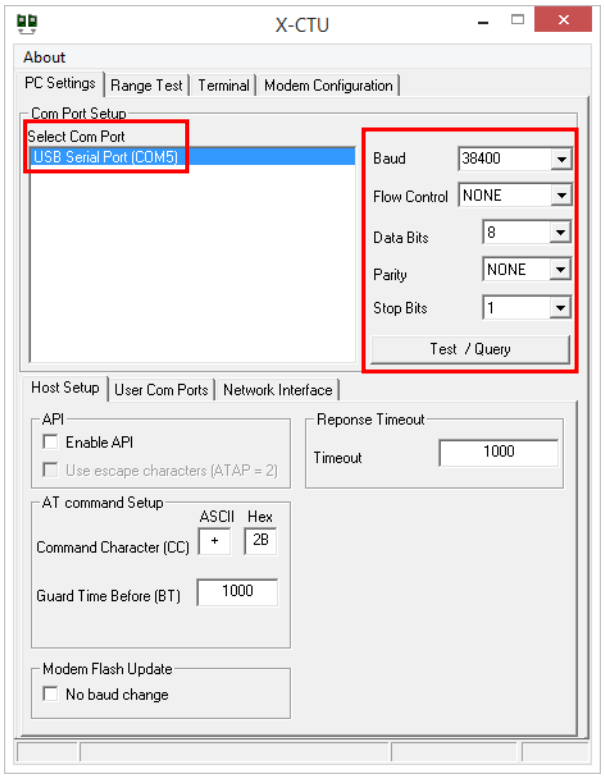

Imagen 14. X-CTU, accediendo a los XBee.

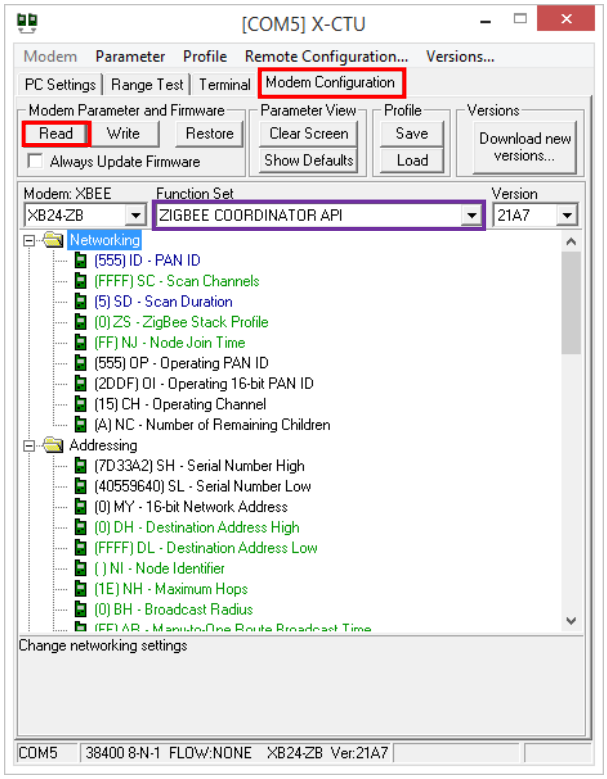

Imagen 15. X-CTU, configurando parámetros.

- **Configuración del coordinador.**

  Seleccionar las siguientes opciones para configurar el coordinador.

  Networking:
  PAN ID: 555 — Número que identifica a la red
  Scan Channels: FFFF — Asignar que canales escaneará. FFFF: Todos
  Scan Duration: 5 — Duración del escaneo
  ZigBee Stack Profile: 0
  Node Join Time: FF — Límite de tiempo para unirse a la red. FF: Sin limites

  Security:
  Encription Enable: 0 — Estado de la encriptación. 0: Desactivado
  Encription Options: 2
  Encription Key: KY
  Network Encription Key: NK

  Serial Interfacing:
  Baud Rate: 5 – 38400 — Valor de los baudios.
  API Enable: 2

  Todos los demás parámetros se dejarán con los valores por defecto.

- **Configuración del router.**

  Seleccionar las siguientes opciones para configurar el router.

Networking:	
PAN ID: 555	Número que identifica a la red
Scan Channels: FFFF	Asignar que canales escaneará. FFFF: Todos
Scan Duration: 5	Duración del escaneo
ZigBee Stack Profile: 0	
Node Join Time: FF	Límite de tiempo para unirse a la red. FF: Sin limites
Channel Verification: 1	Activado (1), el dispositivo verifica si existe un coordinador en ese canal.
Security:	
Encription Enable: 0	Estado de la encriptación. 0: Desactivado
Encription Options: 2	
Encription Key: KY	
Serial Interfacing:	
Baud Rate: 5 - 38400	
API Enable: 2	Valor de los baudios.

  Todos los demás parámetros se dejarán con los valores por defecto.

A la hora de programar los dispositivos, los valores no tienen por qué ser los utilizados en este proyecto. Lo importante es que los valores elegidos sean los mismos en todos los módulos *XBee* que compongan la misma red. Una vez configurados todos los módulos *XBee*, ya están disponibles para utilizarlos y crear la primera aplicación.

3.4 Waspmote IDE.

Se dispone de una herramienta de desarrollo para la programación de los *Waspmotes*. Éste se llama Waspmote-IDE y es gratuito. Mediante éste, se escribirá el código que se desea implementar en el dispositivo.

Una vez abierto el editor, en el panel superior existen una serie de botones y la barra de herramientas que dispone cualquier programa. Desde estos botones se pueden abrir nuevos archivos, guardar, abrir ejemplos, etc. Pero la opción más importante de la barra de herramientas está en la pestaña 'Tools'. Desde aquí se accede a otra pestaña 'Serial Port' (ver imagen 16). En ésta, aparecerán todos los puertos USB que están conectados al PC.

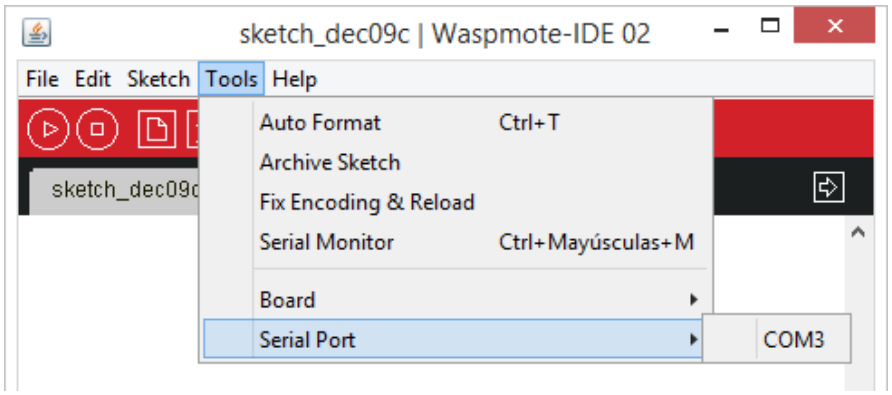

Imagen 16. Entorno de desarrollo Waspmote IDE.

Como el *Waspmote* también estará conectado mediante USB, a la hora de programarlo, se deberá conocer qué valor se le asigna exactamente a cada *Waspmote*. En el caso de elegir un puerto equivocado dará error al intentar descargar el código.

Una vez elegido el puerto se procede a la programación. Los programas constan de dos partes (ver imagen 17): ‘setup’ y ‘loop’.

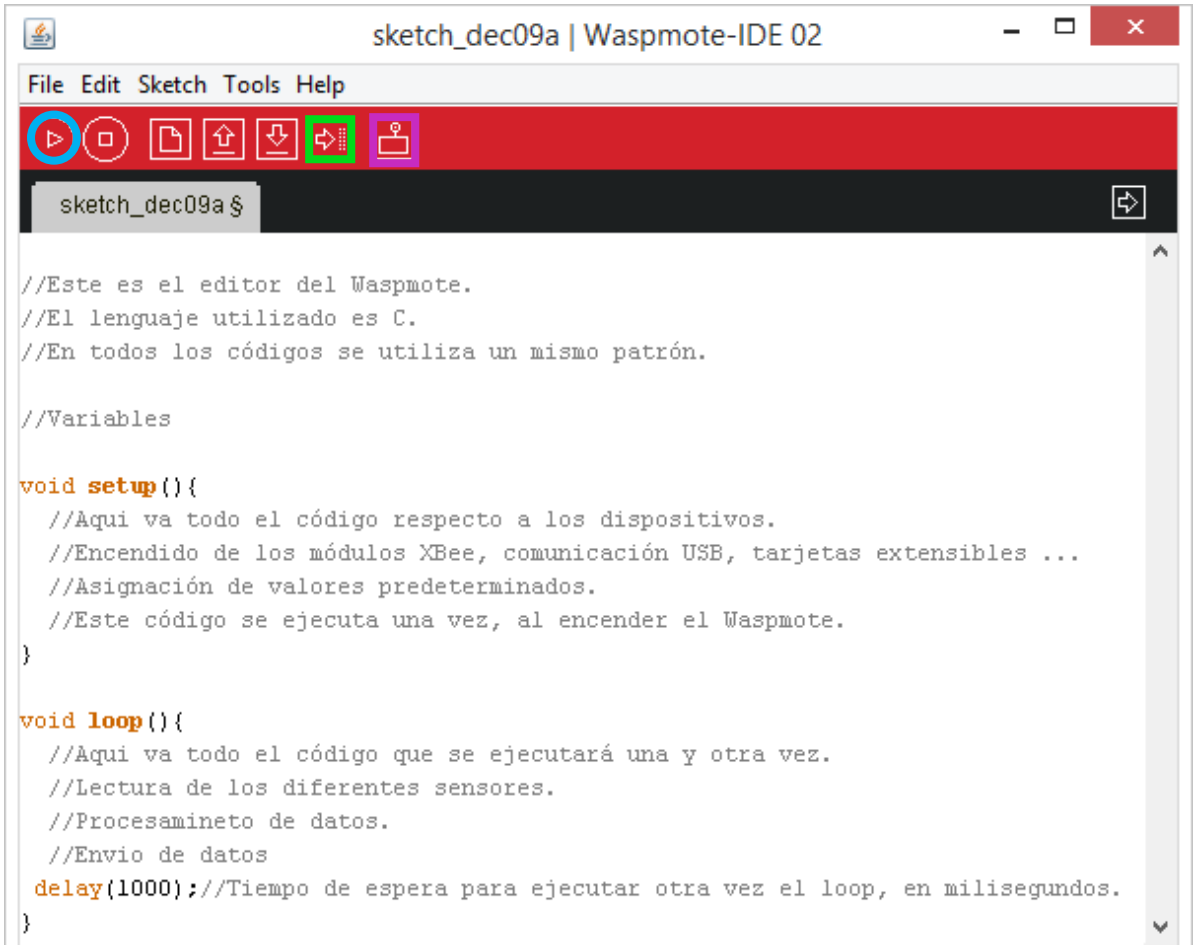

Imagen 17. Plantilla de programación.

- **Setup:** En esta sección se inicializan los diferentes componentes que se van a utilizar, como pueden ser los módulos *XBee*¸ los buses de comunicación USB, tarjetas de expansión, etc. Esta parte del código solo se ejecuta una vez.

- **Loop:** Es la parte donde se escribe la función que se desea realizar, como leer los valores de los sensores, procesar datos, enviar datos, etc. Todo el código que se escriba en el loop se repetirá una y otra vez.

3.4.1 Implementar código.

Una vez escrito el código, debe ser descargado al *Waspmote.* Para ello, se debe conectar al PC mediante un cable USB el dispositivo que se quiera programar. Éste debe estar encendido y no debe tener conectado ningún módulo *XBee*. En caso contrario, el compilador mostrará un error y no se podrá efectuar la descarga. Una vez se ha descargado el código, se podrá desconectar la tarjeta de desarrollo y éste funcionara con autonomía.

Pulsando el botón del círculo azul de la imagen 17, el editor verificará si el código está correctamente escrito. Si no contiene errores, al pulsar el botón del cuadro verde, el programa comenzará a compilarlo para posteriormente descargarlo al *Waspmote.* Si existe algún error o el puerto seleccionado no es el correcto, el editor avisará de ello.

Para comprobar que el código funciona correctamente, pulsando el botón del cuadrado morado, abrirá una ventana, donde muestra por pantalla, todo lo recibido por el *Gateway* o lo enviado por el *Waspmote,* sí está conectado vía USB. Todo dependerá del puerto seleccionado.

En la imagen 18, se puede comprobar que la ventana corresponde al puerto 5. En este caso, el puerto corresponde al *Gateway,* por lo que todo lo mostrado en pantalla, es la información recibida en el ordenador, vía inalámbrica.

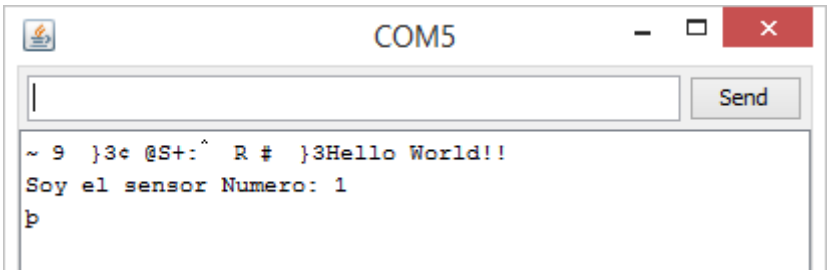

Imagen 18. Información recibida desde el Gateway.

3.5 Sensores analizados.

Para impulsar la versatilidad de los Waspmotes, se encuentran a disposición del usuario varios módulos de comunicación, (*XBee,* GPS, telefonía móvil,..), tarjetas de sensores, (agricultura, radiación,…) y muchos más.

Se han analizado algunos sensores de dos tarjetas de extensión (agricultura y eventos), y los sensores internos del *Waspmote*.

3.5.1 Tarjeta de agricultura.

Con este módulo se han utilizado tres sensores: temperatura, presión y humedad. A continuación, se explicará cómo obtener datos desde cada uno. Para utilizar esta placa, primero se debe inicializarlo en el 'Setup' con el siguiente código:

```
SensorAgr.setBoardMode(SENS_ON);
```

- **Sensor de temperatura.**

El módulo posee dos zócalos de tres pines y en uno de estos se coloca el sensor de temperatura. El sensor que viene con esta placa es el MCP9700A (ver imagen 20), un sensor de temperatura semiconductor.

Especificaciones:	
Rango:	-40ºC ~ 125ºC
Precisión:	±2ºC (0ºC ~ 70ºC)
	±4ºC (-40ºC ~ 125ºC)
Consumo típico:	6µA
Tiempo de respuesta:	1.65s

Código:

El código que debe ir dentro del 'loop'.

```
SensorAgr.setSensorMode(SENS_ON, SENS_AGR_TEMPERATURE);
delay(2000);
temp = SensorAgr.readValue(SENS_AGR_TEMPERATURE);
SensorAgr.setSensorMode(SENS_OFF, SENS_AGR_TEMPERATURE);
```

- **Sensor de humedad.**

El otro zócalo de tres pines sirve para el sensor de humedad. Aquí se conecta el sensor 808H5V5 (ver imagen 21).

Especificaciones:

Rango:	0%RH ~ 100% RH (Humedad Relativa)
Precisión:	±4%RH (25ºC, 30%RH ~ 80%RH)
	±6%RH (0%RH ~ 100%RH)
Consumo típico:	0,38mA
Tiempo de respuesta:	15s

Código:

```
SensorAgr.setSensorMode(SENS_ON, SENS_AGR_HUMIDITY);
delay(15000);
hum = SensorAgr.readValue(SENS_AGR_HUMIDITY);
SensorAgr.setSensorMode(SENS_OFF, SENS_AGR_ HUMIDITY);
```

- **Sensor de presión atmosférica.**

El sensor de presión MPX4115A (ver imagen 22) irá conectado en el zócalo de cinco pines

Especificaciones:

Rango:	15kPa ~ 115kPa
Precisión:	±1.5%V (0ºC ~ 85ºC)
Consumo típico:	7mA
Tiempo de respuesta:	20ms

Código:

```
SensorAgr.setSensorMode(SENS_ON, SENS_AGR_PRESSURE);
delay(20);
hum = SensorAgr.readValue(SENS_AGR_ PRESSURE);
SensorAgr.setSensorMode(SENS_OFF, SENS_AGR_ PRESSURE)
```

Imagen 21. Sensor de temperatura.

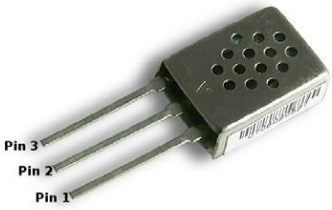

Imagen 20. Sensor de humedad..

Imagen 19. Sensor de presión atmosférica.

3.5.2 Tarjetas de eventos.

Esta placa consta de varios sensores pero solo se explicaran dos de ellos: el sensor de presencia y el de luz. El primer paso como ya se sabe es inicializarlo.

```
SensorEvent.setBoardMode(SENS_ON);
```

También es importante añadir que esta placa posee ocho interruptores manuales (ver imagen 22) para poder activar o desactivar cada zócalo. Por eso, hay que revisar que los interruptores de los zócalos a utilizar estén activados para que los sensores estén alimentados.

- **Sensor de luz.**

Este módulo posee cinco zócalos de dos pines, por lo que se puede conectar el LDR (Light Dependant Resistor, ver imagen 24) en cualquiera de ellos. Se conectará en el pin 1 cuando la intensidad de la luz sea baja y en el pin 3 o 6 cuando la intensidad sea alta.

Especificaciones:	
Resistencia en oscuridad:	20MΩ
Resistencia bajo luz: (10lux)	5 ~ 20kΩ
Rango de espectro:	400nm – 700nm

Código:

```
luz = SensorEvent.readValue(SENS_SOCKET1);
```

- **Sensor de presencia.**

Este sensor de presencia (ver imagen 23) está basado en un sensor pirolítico, es decir, detecta la presencia de un objeto detectando variaciones de calor emitidas en forma de luz infrarroja. Este sensor se debe conectar en el zócalo 7.

Especificaciones:	
Rango de detección:	6 ~ 7m
Rango de espectro:	10µm

Código:

```
alarma = SensorEvent.readValue(SENS_SOCKET7);
```

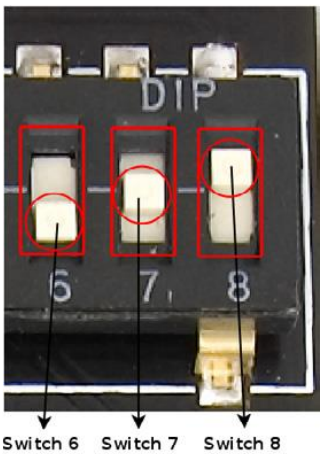

Imagen 22. Interruptores manuales.

Imagen 24. Sensor de luz

Imagen 23. Sensor de presencia.

3.5.3 Sensores integrados en el *Waspmote*.

Los sensores no solo se pueden utilizar con las tarjetas de expansión. El propio *Waspmote* viene equipado con un conector I/O que cuenta con 8 entradas/salidas digitales y 8 entradas analógicas, por lo que podemos conectar hasta 8 sensores en las entradas analógicas, pero teniendo en cuenta que entonces debemos calibrar y adaptar vía software los datos obtenidos. Aparte de esto, el dispositivo viene equipado con un acelerómetro y un sensor de temperatura interno (RTC).

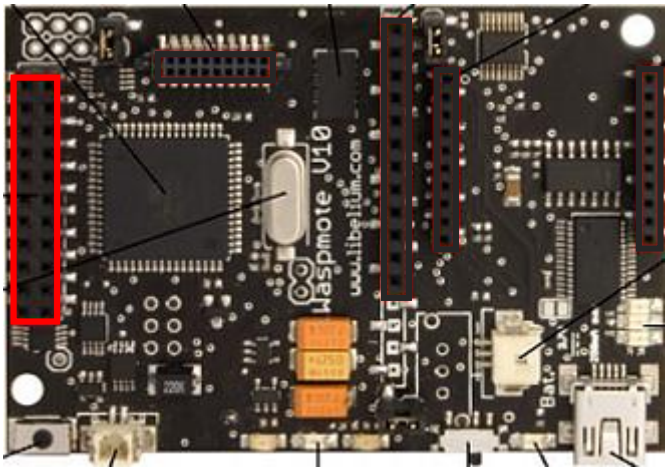

Imagen 25. Conector I/O resaltado en rojo.

- **Acelerómetro.**

La tarjeta Waspmote viene incorporado con un acelerómetro de tres ejes, por lo que se pueden medir los valores de cada eje independientemente. Para inicializarlo se usará el siguiente comando.

```
ACC.ON();
```

Especificaciones:

Resolución:	0,0096m/s^2
Frecuencia de muestreo:	40Hz, 160Hz, 640Hz y 2560Hz

Código:

```
ACC.setSamplingRate(ACC_RATE_2560);
accX = ACC.getX();
accY = ACC.getY();
accZ = ACC.getZ();
```

En las especificaciones se observa que el acelerómetro posee cuatro frecuencias de muestreo distintas, por lo que se puede elegir la más adecuada para la tarea a realizar.

- **Sensor de temperatura (RTC)**

Otro sensor que dispone el *Waspmote* es el RTC. Éste incorpora un sensor de temperatura y un reloj programable. Como siempre, primero se debe inicializarlo.

```
RTC.ON();
RTC.setTime("14:12:15:03:19:14:00");
```

El primer comando inicializa el sensor, mientras que el segundo establece la hora en el momento en el que se ejecuta este comando. El formato es el siguiente:

año:mes:día:día de la semana:hora:minuto:segundo

En el formato del 'día de la semana' un '01' indica que es domingo y un '07' que es sábado.

Código:

```
hora = RTC.getTime();
temp = RTC.getTemperature();
```

Estos son todos los sensores que han sido analizados, que son solo una parte de los muchos disponibles. Como se ha visto con los ejemplos de los códigos, obtener datos mediante uno de estos sensores resulta una tarea muy sencilla. De todos los sensores analizados, debido a la aplicación que se va a desarrollar, se utilizará de aquí en adelante solo el acelerómetro.

3.6 Envío de datos vía inalámbrica.

Por último, los datos recibidos por los sensores hay que enviarlos. Para ello disponemos de dos modos de envío, AT y API.

3.6.1 Modo AT.

Es la manera más simple para enviar los datos. Se establece una comunicación serie entre emisor y receptor donde el primero emite los bits uno tras otro mientras el receptor los recibe. No se establece ninguna comunicación entre las dos estaciones. Por ello es más fácil enviar datos de esta manera pero no se garantiza la integridad de los datos.

Para poder iniciar la transmisión de los datos, primero se debe realizar una configuración. Recordemos que tenemos dos *Xbee* configurados de manera distinta, uno es el coordinador que ira conectado al PC mediante el *Gateway* y el otro es el router, el cual conectaremos al *Waspmote*.

```
void setup(){
        //Iniciar USB
        USB.begin();
        //Iniciar XBee
        xbeeZB.init(ZIGBEE,FREQ2_4G,NORMAL);
        xbeeZB.ON();
        //Escanear Canales
        xbeeZB.setScanningChannels(0xFF, 0xFF);
        while(!brother){
          //Escoger Canal
          xbeeZB.getChannel();
          if(!xbeeZB.error_AT ){
              USB.print("Channel is: ");
              USB.println(xbeeZB.channel,HEX);
          }
          else USB.println("Error getting channel");
}
```

Este código primero inicia el puerto USB y el módulo *XBee*. (Iniciar el USB es opcional, lo hemos utilizado para mostrar mensajes en pantalla y ver que funciona). Después de esto, escaneamos el entorno en busca del canal del coordinador.

Hemos mencionado que la frecuencia que se utiliza es la de 2.4 GHz, pero esta a su vez se divide en 16 canales diferentes, (es por esto por lo que podemos conectar a una misma red WIFI más de un equipo). Cuando se inicia el módulo configurado como coordinador, éste escanea la red en busca de un canal libre y si existe alguno disponible establece su red en ese canal. Por ello, para que el router pueda comunicarse con el coordinador, éste debe escanear todos los canales en su busca. Si lo detecta, se podrá realizar la transmisión de datos con los siguientes comandos.

```
void loop(){
    xbeeZB.send("0013A20040559640","Hello World");
    delay(1000);
}
```

Se puede observar que este modo es muy simple ya que solo necesitamos un solo comando para enviar un dato. En él, introducimos primero la dirección MAC del destinatario, el coordinador en nuestro caso, y después los datos que queremos enviar. En su contra, este método solo nos posibilita para enviar 74 bytes a la vez; si superamos este valor el compilador nos mostrará un error.

3.6.2 Modo API.

A diferencia del modo AT, con este otro modo se envían los datos en forma de paquetes. Cada paquete contiene información adicional aparte de la que queremos enviar como se observa en la imagen 26. Por ejemplo se define el origen y el destino del paquete, por lo que podemos identificar qué dispositivo nos ha enviado la información. Otra clara ventaja es el límite de datos que podemos enviar a la vez. Con este modo, se aumenta la capacidad hasta 1500 bytes. Esto es posible gracias a que se fragmenta el paquete a enviar. Cada fragmento se envía por separado y una vez que se hayan recibido todos, se unen para recuperar el mensaje original.

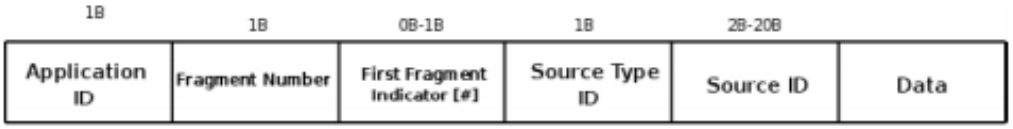

Imagen 26. Paquete API

Además, se asegura la correcta recepción de los datos mediante el envío de ACKs (Acknwoledgement) y por medio de reintentos. El receptor envía un ACK cada vez que recibe un dato para que el emisor sepa que el paquete ha llegado al destino. Si no recibe esta notificación, el emisor lo enviará de nuevo. La integridad de los datos se asegura mediante 'checksum'.

Este es el código que se utilizará para transmitir los datos en modo API.

```
//Definir paquete
packetXBee* paq_sent;
void setup(){
        //Iniciar USB
        USB.begin();
        //Iniciar XBee
        xbeeZB.init(ZIGBEE,FREQ2_4G,NORMAL);
        xbeeZB.ON();
}
void loop(){
    //Establecer parametros para enviar
    paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
    paq_sent->mode=UNICAST;
    paq_sent->MY_known=1;
    paq_sent->packetID=0x52;
    paq_sent->opt=0x00;
    //Establecer dirección de origen
    xbeeZB.setOriginParams(paq_sent, "0013A20040532B3A",MY_TYPE);
    //Establecer dirección de destino y datos a enviar
    xbeeZB.setDestinationParams(paq_sent, "0013A20040559640",data,
                                        MAC_TYPE,DATA_ABSOLUTE);
    //Enviar paquete
    xbeeZB.sendXBee(paq_sent);
    //Borrar paquete para reutilizarlo
    paq_sent=NULL;
    delay(2000);
}
```

La configuración del ‘setup’ en este caso es algo diferente. Al utilizar este método hay varias opciones que debemos elegir.

- **Mode:** ‘Unicast’ para enviar el dato a un solo nodo y ‘Broadcast’ para múltiples nodos.
- **My_known:** Especifica si la dirección de la red es conocida. ‘0’ si es desconocida y ‘1’ si es conocida.
- **PacketID:** Número para identificar los paquetes.
- **Opt:** Opciones para la transmisión. ‘0x00’ para no elegir ninguna opción y ‘0x08’ para activar la transmisión ‘multicast’.

Una vez que hayamos configurado los parámetros de envío, solo queda añadir dos comandos más. El primero, ‘setOriginParms()’, fija la dirección MAC del *XBee* que envía la información. El siguiente, ‘setDestinationParms()’, define la dirección MAC del destino y añadimos los datos que queremos transmitir. Por último, el comando ‘xbeeZB.sendXBee()’, envía el paquete.

4. Tecnología de las FPGA.

Llegados a este punto, se ha comprendido cómo crear una red de sensores para adquirir medidas de diversos entornos gracias al uso de los distintos sensores. En el capítulo 2 se ha explicado que en este proyecto se desea realizar la FFT con los datos recibidos. Esta operación se podría realizar empleando esta misma tarjeta, pero el objetivo de este proyecto, es realizar el procesado de datos en tiempo real, siendo necesario para ello, una gran velocidad de procesado. Por esta razón, se hará uso de las FPGAs, ya que nos permitirá realizar la FFT casi de inmediato.

4.1 ¿Qué es una FPGA?

Las siglas provienen de '*Field Programmable Gate Array*' pero ¿qué es en realidad este dispositivo?

Una FPGA es un dispositivo cuya arquitectura consiste en una matriz de celdas configurables o CLBs (Configurable Logic Block) que se comunican con los bloques de entrada y salida o bloques I/O a través de unos canales de interconexiones horizontales y verticales (ver imagen 27). En estos dispositivos se configuran tanto las interconexiones de las celdas como las propias celdas.

La tecnología actual para su programación, es la SRAM. Se emplean celdas SRAM, constituidas por 5 transistores, para almacenar un bit. Depende de su valor, se establece la conexión entre dos caminos o no, definiendo así tablas de verdad o creando multiplexores, circuitos lógicos etc. La principal característica de estas memorias, es el hecho de que son volátiles, es decir, una vez se retira la alimentación, el dispositivo pierde toda la configuración actual. Por esta razón, es necesario configurar las FPGAs cada vez que se encienden.

En una FPGA puede haber desde cientos de miles hasta millones de elementos lógicos programables o puertas lógicas equivalentes. El usuario, mediante un lenguaje de descripción de hardware (HDL), la arquitectura del circuito que desea. De esta manera se disminuye el tiempo de procesado ya que solo interviene el hardware necesario para cada función. Otra característica importante, es la posibilidad de implementar circuitos en paralelo, una tarea complicada de realizar en un microprocesador. Además, en la mayoría de las FPGAs, se pueden encontrar funciones de alto nivel embebidas en la propia matriz de interconexiones, como sumadores, multiplicadores, bloques RAM, etc (ver imagen 28). De esta manera, si uno necesita hacer uso de memorias RAM, no hace falta que los diseñe, basta con utilizar las memorias integradas de la FPGA.

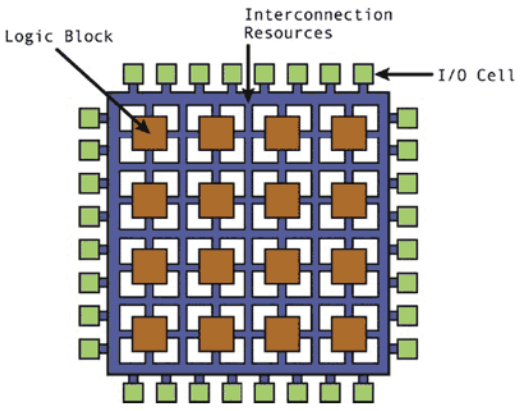

Imagen 27. Arquitectura de una FPGA.

4.2 Familia Spartan 3E.

La familia de FPGAs *Spartan 3E* está compuesta por cinco dispositivos diferentes, ofreciendo desde 100.000 hasta 1.600.000 puertas lógicas equivalentes. La principal característica de esta gama, es el precio comparado con las prestaciones que ofrece, lo cual, lo convierte en un dispositivo ideal para empezar a introducirse en el mundo de las FPGAs. El dispositivo concreto a utilizar, es el XC3S500E.

Este modelo, tiene un número de 1164 CLBs (Configurable Logic Block). Estos bloques programables contienen tablas de verdad, o LUTs (Look Up Table) flexibles, que implementan elementos de lógica y de almacenamiento, permitiendo utilizarlos como Flip-Flops. Además,se pueden emplear para realizar una gran variedad de funciones lógicas o para almacenar información.

Respecto a las entradas y salidas, se cuenta con 232 pines I/O, siendo cada una de ellas bidireccional y soportando una gran variedad de estándares para las señales.

Como se ha comentado, las FPGAs, suelen tener incorporados varios elementos incorporados en la propia matriz. En este caso, se disponen de multiplicadores, bloques RAM y DCMs (Digital Clock Manager).

- **Multiplicadores:** Existen 20 multiplicadores de 18 bits. Gracias a esto, no es necesario es uso de CLBs para implementar éstos elementos.
- **Bloques RAM:** Hasta 360kB de memoria emebedida, distribuidas en bloques de 18kB.
- **DCM:** Proveen soluciones digitales para distribuir, multiplicar, dividir, retrasar y desplazar en fase las señales de reloj. Se cuentan con 4 DCMs en este dispositivo.

En la imagen 28, se muestra la disposición de los elementos empotrados en la arquitectura de la FPGA.

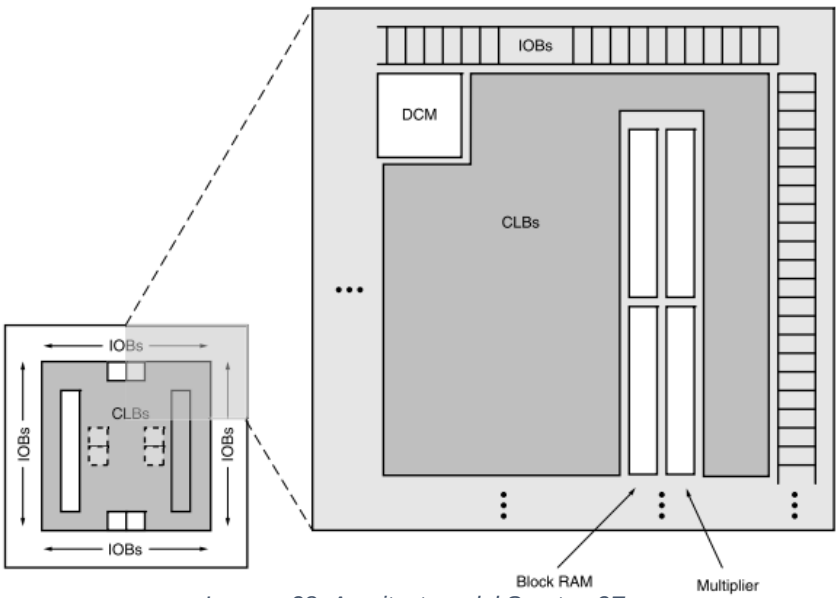

Imagen 28. Arquitectura del Spartan 3E.

4.3 Desarrollo de aplicaciones para FPGAs.

Para el desarrollo de aplicaciones para las FPGAs, se utilizará la herramienta de desarrollo, '*Xilinx ISE Design Tools'.* Se disponen de dos lenguajes para trabajar con estos dispositivos, 'VHDL' y 'Verilog'. Mediante este software se tiene la posibilidad de elegir cualquier dispositivo del fabricante, implementar el código en él para ver los recursos empleados, realizar simulaciones, etc.

Además, tiene la opción de implementar módulos predefinidos. Estos módulos se conocen como IP-COREs (núcleos de propiedad intelectual) y existen varios para casos generales, como el procesado digital de señales (DSP), procesado de imágenes y vídeo, funciones matemáticas, comunicación serial, etc.

La ventaja de utilizar estos núcleos reside en que las funciones que se implementan están muy optimizadas y ahorran el tiempo que se necesita para programar dicha función. Para que ésta se adapte a las características del desarrollador, cada núcleo ofrece la posibilidad de configurar varios parámetros.

En la página web del fabricante, se encuentran las hojas de características de todos los IP-COREs. Estos documentos son muy importantes ya que aquí se explica el funcionamiento de cada núcleo, todos los parámetros involucrados, definición de cada uno de éstos, etc.

5. Aplicación desarrollada.

Una vez se han analizado los dispositivos a utilizar, se procede al desarrollo del sistema para la monitorización de vibraciones. Se trata de medir y enviar las vibraciones al PC, para luego transferirlas a la FPGA. Se realizará la FFT y una vez finalizado el procesado, se mostrarán los resultados en una gráfica.

Este será el esquema que se seguirá a la hora de diseñar los diferentes dispositivos.

1. El *Waspmote* medirá las vibraciones y los enviara vía inalámbrica.
2. El *Gateway* conectado al PC recibirá los datos.
3. Mediante el programa *LabView* se extraerán los datos de las vibraciones y se enviaran con la adecuada sincronización a la FPGA.
4. La FPGA realizará la FFT con los datos recibidos y enviará los resultados al PC.
5. Se mostrarán en pantalla los resultados obtenidos mediante gráficos.

Teniendo en cuenta las frecuencias de muestreo del acelerómetro, se ha decidido enfocar el sistema para el análisis de vibraciones fuertes y de bajas frecuencias. De este modo, se descarta para el uso como afinador de instrumentos, donde las frecuencias superan los kilohercios, pero es adecuado para analizar electrodomésticos que funcionan con motor. En este ejemplo, se ha analizado el motor de una lavadora.

5.1 Análisis de las vibraciones a medir.

La lavadora que se ha usado, tiene cinco velocidades distintas para el centrifugado.

Velocidades de centrifugado

400rpm	600rpm	800rpm	1000rpm	1200rpm
6,67Hz	10Hz	13,33Hz	16,67HZ	20Hz

Teniendo en cuenta estas velocidades, se fijará la frecuencia máxima a medir en 50Hz. Teniendo en cuenta el teorema de Nyquist, la frecuencia de muestreo debe ser de 100Hz.

El siguiente parámetro a fijar es el número de muestras. La resolución de la FFT es inversamente proporcional a éste, por lo que a mayor número de muestras mejor resolución. Se ha escogido, en un principio, un valor de N = 4096 muestras para lograr una resolución que permita diferenciar pequeñas variaciones en la velocidad del motor, obteniendo de esta manera una resolución de:

$$\Delta f = \frac{f_s}{N} = \frac{100}{4096} \rightarrow \Delta f = 0{,}024Hz$$

5.2 Adquisición y envío de los datos.

Para realizar esta tarea, se programará el *Waspmote* para que obtenga el valor de la aceleración de cada eje y para que lo envíe vía inalámbrica cada 10ms. Para ello, siguiendo las indicaciones del tema anterior, se ha creado el siguiente código.

```
//Paquete
packetXBee* paq_sent;
//Variables
char data[72];
int ejeX;
int ejeY;
int ejeZ;
void setup(){
  //Iniciar XBee
  xbeeZB.init(ZIGBEE,FREQ2_4G,NORMAL);
  xbeeZB.ON();
  //Iniciar acelerómetro
  ACC.ON();
  ACC.setSamplingRate(ACC_RATE_160); //Frecuencia de muestreo: 160Hz
  //Paquete
  paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
  paq_sent->mode=UNICAST; //Tipo de envio: Unicast
  paq_sent->MY_known=0; //Establecer Network Address conocido
  paq_sent->packetID=0x52; //Establecer ID aplication level
  paq_sent->opt=0x00; //Opciones no seleccionadas
}
void loop(){
  //Datos acelerómetro
  ejeX = ACC.getX();
  ejeY = ACC.getY();
  ejeZ = ACC.getZ();
  //Agrupar los datos
  sprintf(data,"::%i::%i::%i::\n",ejeX,ejeY,ejeZ);
  //Envio paquete
  xbeeZB.setDestinationParams(paq_sent, "0013A20040559640", data,
                                        MAC_TYPE,DATA_ABSOLUTE);
  xbeeZB.sendXBee(paq_sent);
  delay(10);
}
```

Mediante este código, se obtienen los valores de la aceleración de cada eje y se juntan en una cadena de texto con el siguiente formato: ::ejeX::ejeY::ejeZ::. Finalmente se envía este fragmento al PC.

Se ha decido agrupar los datos de esta manera porque desde el PC se leerán todos los bytes que ha recibido el *Gateway*, incluyendo la información de la cabecera del paquete. De esta manera, se identifican los datos ya que a cada valor le precede la cadena de texto “::”.

5.3 Procesado de datos mediante FPGAs.

5.3.1 IP CORE FFT 7.1

Para añadir un núcleo al proyecto, se debe elegir la opción IP-CORE al añadir una fuente al proyecto.

Al elegirlo, se abrirá una ventana con todos los IP-COREs disponibles ordenados por carpetas. Solo hace falta elegir el que interesa. En este caso, el IP-CORE FFT 7.1.

Después de cargar el núcleo, aparecerá otra ventana donde se especifican los parámetros a implementar.

Estos son todos los parámetros que se pueden configurar.

- **Canales:** Se pueden elegir hasta 12 canales para poder efectuar la FFT en cada uno de ellos de manera paralela. Al tener recursos limitados, se ha elegido un solo canal. Por tanto, se debe efectuar la misma operación con las muestras de cada eje.

- **Número de muestras:** Número de muestras con el que efectuará la FFT.

- **Arquitectura:** Existen cuatro arquitecturas diferentes.

  - Pipelined, Streaming I/O: Esta arquitectura encadena mariposas de dos puntos para ofrecer un procesamiento continuo de los datos. Cada etapa, posee su propia memoria, almacenando los datos de entrada y de salida. Por lo tanto, es posible ir introduciendo nuevos datos mientras se obtienen los resultados previos de la FFT. Solo se puede implementar utilizando un canal.

  - Radix 4, Burst I/O: Una mariposa realimentada de cuatro puntos procesa los datos. Al utilizar una sola etapa, primero se cargan todos los datos en la memoria, después se procesan los datos y finalmente se descargan. Esta arquitectura no ofrece un procesamiento continuo pero si es posible implementar diferentes canales en paralelo. Al utilizar un solo lazo realimentado, el tiempo de procesamiento es mayor pero es más liviano.

  - Radix 2, Burst I/O: Es la misma arquitectura que la anterior solo que con etapas de dos puntos, por tanto, necesita más tiempo para procesar los datos pero utiliza menos recursos.

  - Radix 2-Lite, Burst I/O: Mismo concepto que en la anterior arquitectura pero con recursos reducidos, por lo tanto, mayor tiempo de procesamiento. Es la arquitectura que se ha elegido por ser la que menos recursos utiliza. Más adelante se explicará en detalle el funcionamiento de éste.

- **Actualizar número de muestras:** Marcando esta casilla, se puede cambiar el número de muestras mientras se ejecuta el código. En este caso el número de muestras se mantendrá fijo.

- **Formato de los datos:** El IP-CORE puede manejar dos tipos de datos: coma fija o coma flotante. Se ha escogido la representación de coma fija por utilizar menos recursos.

- **Longitud de los datos de entrada:** Número de bits que se utilizarán para representar los datos de entrada. L_i = 16 bits.

- **Longitud del factor de fase:** Número de bits que se utilizará para representar la fase, es decir, la exponencial compleja de la DFT. P_i = 34 bits.

- **Opciones de escalado:**
  - Sin escalar: Cada vez que los datos atraviesan una etapa, se incrementa la longitud de palabra para evitar la pérdida de precisión. Con esta opción, el IP-CORE no se ocupa de escalar los datos, pero necesita más recursos para manipular datos de mayor longitud de palabra. Para que el resultado sea correcto, al final de la FFT se debe dividir el resultado por el número de muestras.

- Escalado: Cada vez que un dato atraviesa una mariposa, se tiene la opción de desplazarlo hasta 3 bits. De esta manera, se reduce el tamaño de los datos para que no haya un crecimiento incontrolado. La manera para especificar cuantos bits se deben desplazar en cada etapa, se efectúa mediante un array. Cada pareja de bits del array especifica cuantos bits se desplazarán en cada mariposa:
  "010011" -> 1 bit en el primero, 0 bits en el segundo y 3 bits en el último.
- Block Floating Point: El IP-CORE determina el escalado necesario.

- **Modos de redondeo:**
  - Truncamiento: Después de cada lazo se descartan los dígitos menos significativos.
  - Redondeo: Se redondea el resultado a la salida del lazo. Al ser más preciso se ha elegido esta opción.

- **Orden de salida:**
  - Orden inverso: Por naturaleza, la FFT no devuelve los resultados en orden. Por lo tanto, es el usuario quien tiene que ordenarlos.
  - Orden natural: El núcleo utiliza más recursos para ordenar los datos. Por comodidad se usará este modo.

- **Pines opcionales:**
  - CE (Clock Enable): Pin que permite activar o desactivar la entrada de reloj.
  - SCLR (Synchronous Clear): Pin para limpiar los datos.
  - OVFL (Overflow): Pin que indica cuándo ha habido un descarrilamiento de los datos.
- **Opciones de memoria: Datos**
  - Block RAM: Utilizar los bloques RAM para almacenar datos.
  - RAM distribuido: Crear memoria RAM utilizando LUTs, para almacenar datos.
- **Opciones de memoria: Fase**
  - Block RAM: Utilizar los bloques RAM para almacenar datos.
  - RAM distribuido: Crear memoria RAM utilizando LUTs para almacenar datos.
- **Opciones de optimizado:**
  - Lógica CLB: Utilizar LUTs para implementar los multiplicadores complejos.
  - Multiplicador-3: Opción para optimizar los recursos.
  - Multiplicador-4: Opción para optimizar el rendimiento.

Estas últimas opciones son muy interesante ya que cada FPGA tiene un número de bloques RAM y multiplicadores complejos. En nuestro caso, el número de LUTs disponibles es limitado, por lo que no tiene sentido utilizarlos para implementar los bloques RAM o los multiplicadores complejos. Además, éstos tienen un mejor rendimiento que los LUTs.

A priori no se puede saber cuántos recursos ocupará cada arquitectura. Por ello, se han calculado los recursos utilizados, frecuencia máxima de trabajo y ciclos de operación dependiendo de la arquitectura y del número de muestras. Para ello, se han implementado las cuatro arquitecturas, empleando distintos números de muestras en cada una de ellas. Con los resultados, se ha construido la siguiente tabla para determinar qué arquitectura es la más conveniente.

Nº muestras	Recursos (%) (LUTs + RAM + MULT)				Frecuencia máxima (MHz)				Ciclos de operación*			
	R2L	R2	R4	STR	R2L	R2	R4	STR	R2L	R2	R4	STR
1024	12.7	15.7	37.3	51.7	151.6	161.9	136.0	152.1	12453	7364	5559	2159
2048	16.0	19.0	38.7	66.3	139.5	155.7	125.0	164.6	26804	15575	--	--
4096	19.7	22.7	45.0	78.7!	116.9	147.4	116.7	154.0	65655	41277	22739	12475
8192	38.0!	34.7!	57.7!	--	100.4	150.9	105.3	--	--	--	--	24819

Tabla 1. Recursos utilizados en función de la arquitectura y número de muestras.

R2L: Radix 2-Lite, Burst I/O
R2: Radix 2, Burst I/O.
R4: Radix 4, Burst I/O
STR: Pipelined, Streaming I/O

*: Datos obtenidos del datasheet.
!: Algún recurso al límite

La importancia de realizar esta tabla es la de asegurar que se dispondrá de suficientes recursos para hacer un post-procesado de los datos que proporcione la FFT. Estos resultados son números complejos, resultando difícil la interpretación de éstos. Por ello, se calculará el módulo de los datos obtenidos.

Observando la tabla 1, se ha decidido utilizar la arquitectura Radix 2-Lite, Burst I/O, manteniendo el número de muestras en 4096. Con esto, se utilizarán el 19.7% de los recursos, dejando suficientes recursos para calcular después el módulo de los resultados de la FFT.

5.3.2 Radix 2-Lite, Burst I/O.

En el capítulo 2, en la imagen 9, se ha mostrado un esquema del uso de las mariposas para efectuar la FFT. En él, se observan que éstas están concatenadas en serie. En el caso de las arquitecturas Radix 4, Burst I/O, Radix 2, Burst I/O y Radix 2-Lite, Burst I/O, se emplea una sola etapa para el cálculo. Haciendo uso de memorias ROM (Read Only Memory), se guardan los términos calculados para poder utilizarlos después. Una vez se ha terminado el proceso, se descargan todos los datos.

La arquitectura a analizar tiene la característica de emplear una etapa de dos puntos, pero con una sola entrada. De esta manera, se comparte la mariposa, generando una salida en cada ciclo. Esto implica un uso menor de recursos a costa de emplear más tiempo para el cálculo de cada mariposa. En la imagen 29, se muestran con una cruz roja, todos los elementos que han sido eliminados de la arquitectura Radix 2, Burst I/O.

Esta arquitectura permite realizar la FFT con un número de muestras desde 8 hasta 65536. Además, es posible la implementación en paralelo, pudiendo de esta manera realizar el cálculo de hasta 12 canales simultáneamente. En contra, primero se deben cargar todos los datos en la memoria ROM y una vez se han obtenido todos los resultados, se procede a la descarga de los datos.

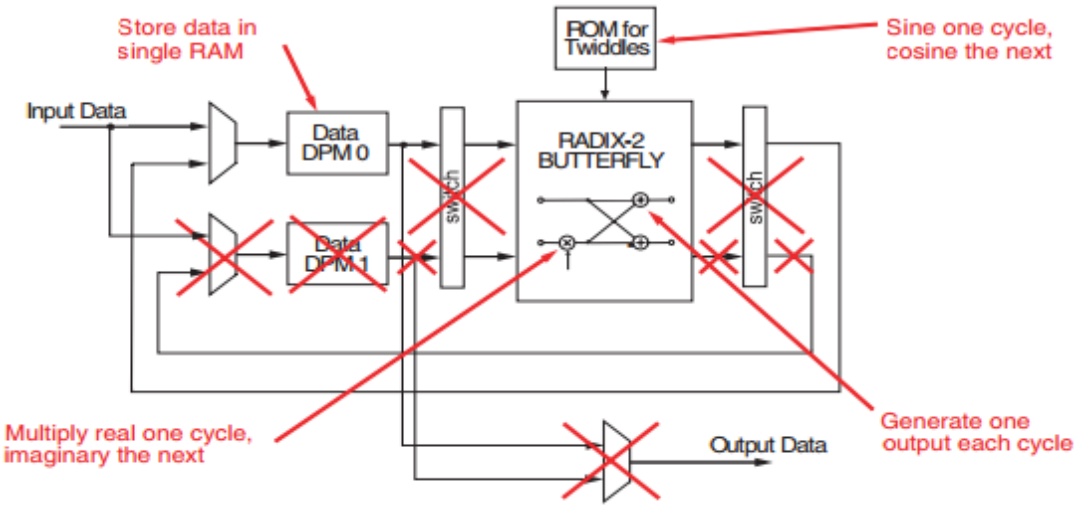

Imagen 29. Arquitectura Radix 2-Lite, Burst I/O vs. Radix 2, Burst I/O.

5.3.3 Arquitectura del diseño implementado en la FPGA.

La FFT da como resultado un número complejo, pero lo que se quiere mostrar es el módulo del resultado, lo que muestra un analizador de espectros. Para ello, se han implementado otros dos IP-CORE, un multiplicador complejo y un núcleo para calcular la raíz cuadrada. Para que el cálculo sea correcto, se debe multiplicar el número complejo, obtenido del resultado de la FFT con su conjugado. Mediante otro IP-CORE que efectúa una resta, restándole a 0 el valor imaginario obtenido en la FFT se obtiene la parte imaginaria conjugada. De este modo se obtendrá el espectro de frecuencias de las vibraciones analizadas.

En la imagen 30 se muestra el esquema RTL (Register Transfer Level) del diseño de la FPGA.

5.3.4 Sincronización de los datos.

Para el correcto funcionamiento del IP-CORE de la FFT, los datos se deben introducir con la debida sincronización. A continuación, se explicará cómo se deben introducir y leer los datos.

Se dispone de una entrada ‘Start’, con la que se decide cuándo comenzará a ejecutarse el núcleo. Una vez activada, a los dos ciclos de reloj, se activará la señal de salida ‘RFD’ (Ready For Data), señalando que se están guardando en la memoria los valores de las aceleraciones, introducidas mediante el bus ‘Xn_re’. El bus ‘Xn_index’ indica en que índice se están guardando los datos. Los datos se deben introducir sincronizadas con el reloj del sistema.

Una vez se han cargado todos los datos, se desactivará la señal RFD y se activará la señal ‘Busy’. Ésta permanecerá en el mismo estado hasta que finalice de procesar los datos. Un ciclo de reloj antes de finalizar, la señal ‘Edone’ indicará que el cálculo de los resultados va a finalizar. Cuando ésto suceda, la señal ‘Done’ será quién muestre el fin de la etapa de procesado. Estas dos señales, permanecen activas durante un ciclo de reloj. Finalmente, la salida DV (Data Valid), indicará que los resultados están disponibles en el bus ‘Módulo’ con su respectivo índice, señalado por ‘Xk_index’.

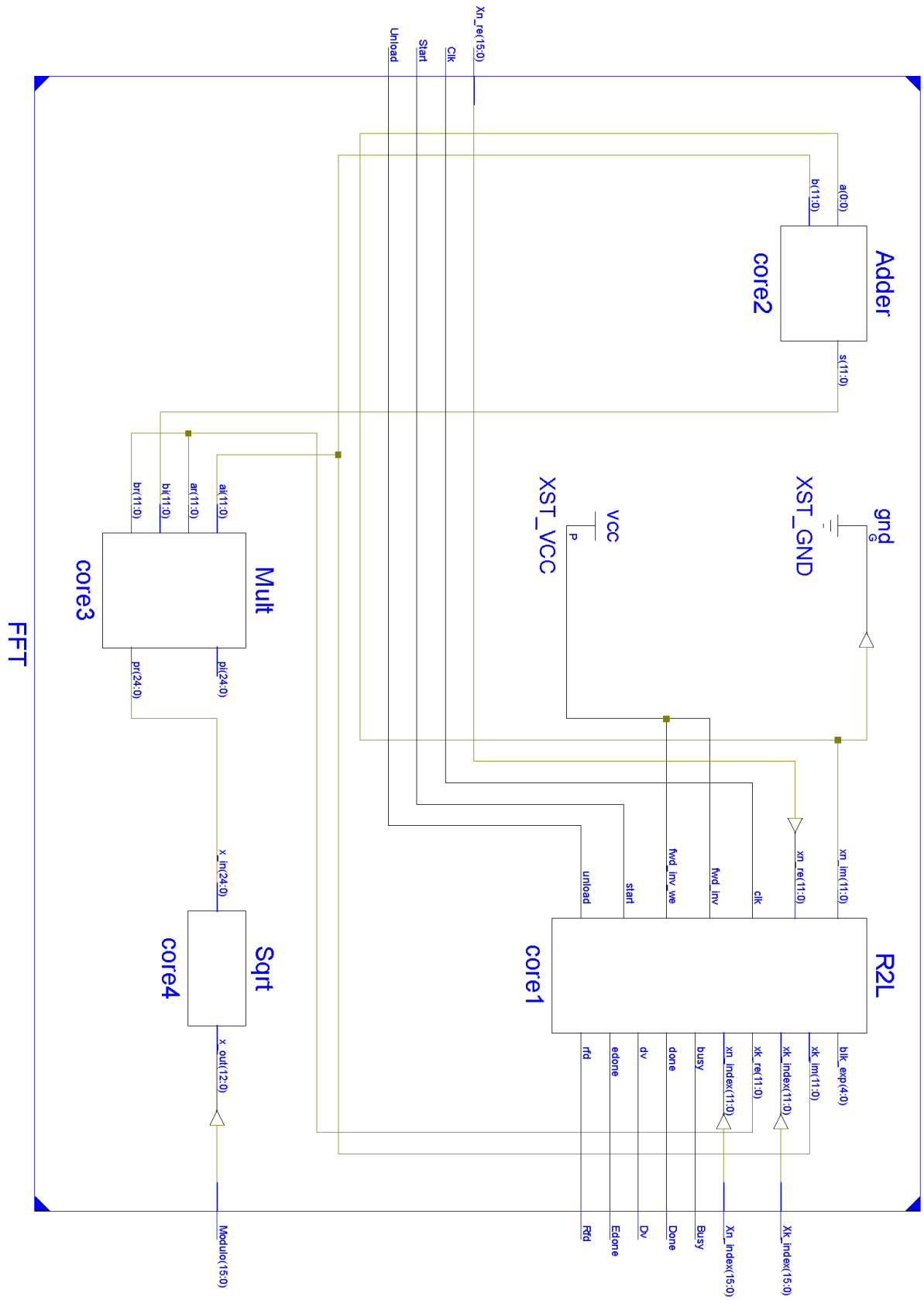

Imagen 30. Esquema RTL del diseño implementado.

5.3.5 Recursos utilizados.

Después de haber implementado el diseño de la imagen 30 en el dispositivo *Spartan 3E,* se ha obtenido el porcentaje empleado de los distintos recursos de la FPGA.

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	1,276	9,312	13%
Number of 4 input LUTs	1,328	9,312	14%
Number of bonded IOBs	57	232	24%
Number of RAMB16s	8	20	40%
Number of BUFGMUXs	1	24	4%
Number of MULT18X18SIOs	6	20	30%

Tabla 2. Recursos utilizados de la FPGA.

- **Number of Slice Flip Flops:** Número de flip flops utilizados.
- **Number of 4 input LUTs:** Número de de LUTs de cuatro entradas utilizados.
- **Number of bonded IOBs:** Número de pines de entrada/salida utilizadas.
- **Number of RAMB16s:** Bloques RAM empleados.
- **Number of BUFGMUXs:** Número utilizado de multiplexores.
- **Number of MULT18X18SIOs:** Número utilizado de multiplicadores empotrados.

Otro dato de interés es la frecuencia máxima de trabajo. El sistema que se ha realizado tiene un retardo de la señal de 7,33ns. Por lo tanto, se obtiene una frecuencia máxima de trabajo de 136,4 MHz. Dividiendo el número de ciclos correspondiente de la tabla 1, por la máxima frecuencia obtenida, la latencia del procesado de los datos es de 0,48 ms.

5.4 Recepción, adaptación y envío de los datos.

Como se ha indicado en primer capítulo de esta memoria, la comunicación entre la tarjeta de sensores y la FPGA se efectuará mediante el uso de un PC. Utilizando el programa *LabView*, se realizará la recepción, adaptación y envío de los datos a la FPGA. Una vez se ha efectuado la FFT, los resultados se mostrarán mediante un gráfico utilizando este mismo software. Como se ha mencionado en la introducción, se hace uso de *LabView* solo como herramienta de desarrollo. En la aplicación final, la comunicación entre estos dos, sería directa.

5.4.1 LabView.

LabView, es un entorno de desarrollo para diseñar sistemas que utiliza un lenguaje de programación visual. Principalmente, se desarrolló como un software para el uso de control de instrumentos, pero hoy en día, se emplea para una multitud de tareas.

Gracias a su lenguaje gráfico, diseñar distintos programas resulta una tarea más sencilla y más intuitiva. Además, ofrece la posibilidad de diseñar un panel frontal donde situar todos los controles que se incluyan (botones, interruptores, entrada de texto,…), además de poder mostrar los resultados mediante gráficos.

A continuación se explicará por partes cómo se ha realizado la comunicación entre los dos dispositivos.

Para empezar a trabajar, lo primero será crear un proyecto. Desde aquí, se podrán añadir dispositivos, VIs (Virtual Instruments), configurar opciones de ejecución, etc.

Una vez hecho esto, se indicará el dispositivo a utilizar en editor de *LabView* (ver imagen 31). Cabe recordar que es necesario tener instalado el driver de *Spartan 3E*. Después, se programarán dos VIs (bloques básicos de construcción para los programas de *LabView*). Se hará uso de dos VIs, uno para el host, donde se programará el bloque para la recepción y el envío de los datos a la FPGA, y el otro para el device, el cual configurará la FPGA para realizar la función que se ha diseñado.

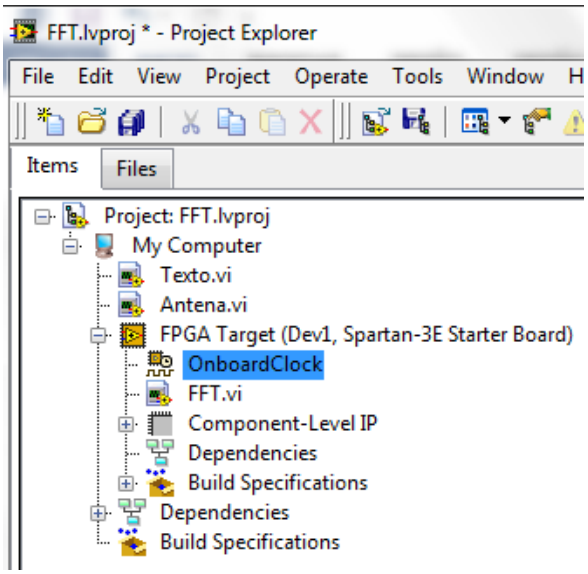

Imagen 31. Proyecto en LabView

5.4.2 Recepción y procesado de los datos.

A la hora de configurar el *Waspmote*, se ha decidido emplear una frecuencia de muestreo de 100Hz, esto es, se enviarán datos cada 10 ms. Teniendo esto en cuenta, se tendrá que leer la información proveniente del puerto USB, puerto donde estará situado el *Gateway*, con la misma frecuencia. Los datos se transferirán a la FPGA, una vez se han obtenido 4096 muestras.

Para ello, primero se ha creado un bucle, que estará constantemente comprobando si se ha recibido información desde el puerto USB. Si la información recibida supera los 10 bytes, se procederá a extraer los datos.

Una vez se reciba la información, se separarán los valores de cada eje. Los datos están separados por un par de doble puntos, por lo que cada paquete tendrá la siguiente forma:

$$\underbrace{asR\beta;ykQvx}_{Cabecera}\ \underbrace{::98::-103::1026::}_{Información}$$

Los datos recibidos de cada eje, se irán guardando en una matriz hasta haber obtenido 4096 muestras. Cuando se haya llegado a este punto, se descargarán los valores del eje indicado a la FPGA.

En la imagen 32, se muestra el diagrama de bloques de este VI. Como se aprecia, toda la programación se realiza uniendo diferentes bloques que desempeñan diferentes funciones.

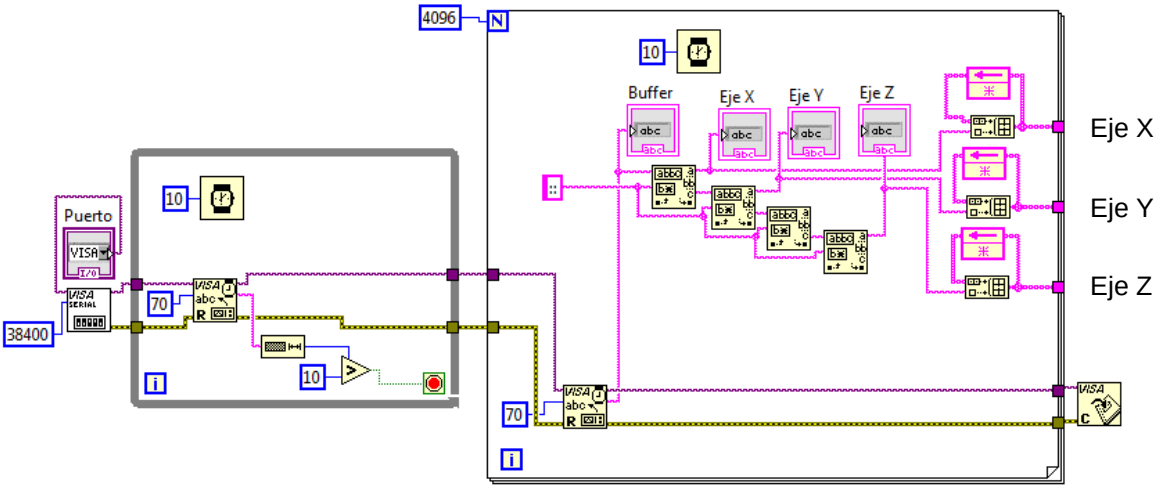

Imagen 32. Host: recepción y adaptación de datos.

5.4.3 Comunicación entre PC y FPGA.

- **Device:** VI situado dentro de la FPGA.

Para poder ejecutar el programa al completo, es necesario crear un componente IP, el cual determina la configuración del hardware de la FPGA. Éste se creará utilizando los archivos .vhd o .ngc que se han generado al crear el código. En este proyecto no se ha podido compilar el programa usando los archivos .vhd, y en vez de éstos, se han utilizado los archivos .ngc.

Una vez se ha añadido el componente IP, se comenzará a programar el VI de la FPGA. Se añadirá un bucle 'while' para que se ejecute continuamente. Dentro de éste, añadiremos un nodo I/O como se muestra en la imagen 33. En este nodo, aparecerán las entradas y salidas del componente IP.

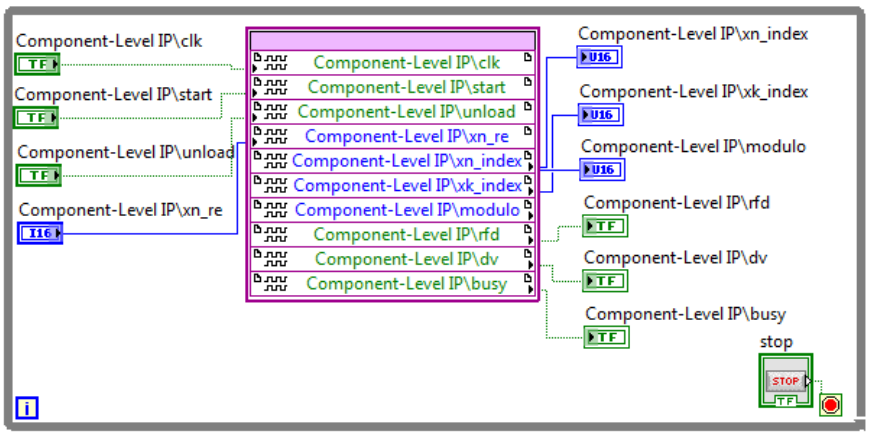

Imagen 33. Device: VI que configurará la FPGA.

Cada una de las salidas se debe conectar a un control, y las salidas a un indicador. De esta manera, se tendrá la opción de introducir los datos desde un VI externo.

- **Host:** VI externo, desde el cual se controlarán las entradas y salidas.

Por último, se programará el VI que introduzca los datos en la FPGA y que reciba los resultados para poder mostrarlos mediante un gráfico.

En la imagen 34, se indica en rojo, los bloques a utilizar para la escritura y lectura de los datos desde la FPGA. En éstos, aparecerán los controles e indicadores que hemos añadido dentro del VI del dispositivo. El bloque indicado en naranja, hace referencia al VI del dispositivo y es por ello que aparecen las entradas y salidas del host, en los bloques de escritura y lectura.

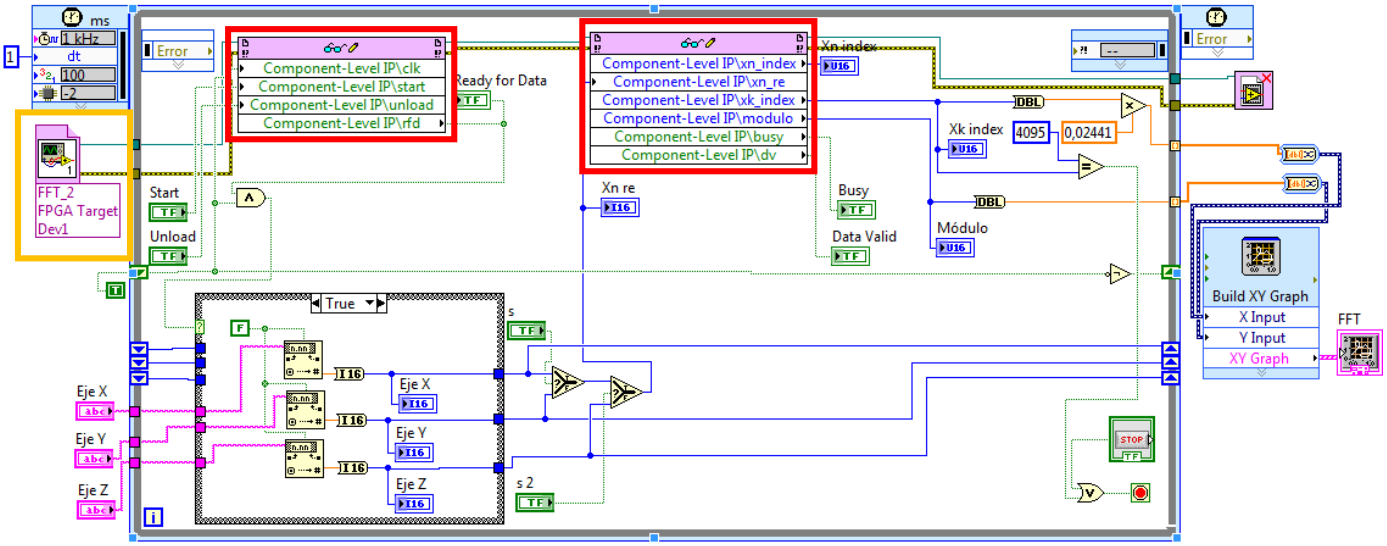

Imagen 34. Host: VI que enviará los datos a la FPGA.

5.4.4 Velocidad de procesado y FIFOs.

Una vez diseñados los VIs, todo el sistema está listo para funcionar, pero existe un problema. La velocidad máxima a la que puede operar la FPGA, en este caso, es de 136,4 MHz. A esta velocidad, el PC no es capaz de leer los datos. Por ello, se deben utilizar los bloques FIFO.

Las siglas FIFO, provienen de First In, First Out, es decir, primero en llegar y primero en salir. Estas estructuras se emplean a la hora de gestionar colas. En este caso, al obtener resultados a una velocidad que no se pueden procesar, los bloques FIFOs se encargarían de almacenar éstos valores y enviarlos al PC, a un ritmo más lento y manteniendo el orden.

En este proyecto se ha dado cuenta de que el *Spartan 3E* no es compatible con el uso de FIFOs. En consecuencia, el sistema tendrá que trabajar a una velocidad mucho más baja. La máxima velocidad que se ha podido implementar manualmente, ha sido de 1 kHz. Este valor queda muy lejos de los 136.4MHz, pero resulta imposible utilizar esta FPGA si se desean visualizar los datos.

5.5 Resultados experimentales.

A pesar de la limitación de la velocidad del reloj, el sistema debe funcionar correctamente. Para comprobar el funcionamiento, se ha puesto en marcha todo el sistema, midiendo y enviando los valores de las aceleraciones desde el *Waspmote* y realizando el procesado en la FPGA.

En la imagen 35 se muestran los resultados obtenidos. Se observa un pico bien definido en los tres ejes a la frecuencia de 20 Hz. Esta frecuencia expresada en minutos, equivale a 1200 rpm, siendo éste la velocidad del motor de la lavadora analizada. Otro detalle que se aprecia, principalmente en el eje Z, son los armónicos que aparecen. Esta lavadora no oscilaba libremente en el momento que se ha realizado esta prueba, sino que parte de sus vibraciones han sido frenadas por estar en contacto con el suelo. De esta manera, la oscilación que en un principio debería de tener una forma sinusoidal, ha adoptado una forma en la que se parece más a una onda cuadrada. En el espectro de una onda cuadrada, aparecen los armónicos impares de la frecuencia fundamental, sin embargo, si la señal no es perfectamente cuadrada, los armónicos se sitúan también en los armónicos pares. Por lo tanto, tal vez sea esta la causa de la aparición de las señales en torno a los 40 Hz. Se confirma, por tanto, el correcto funcionamiento de la aplicación de este proyecto.

Como última comprobación del sistema desarrollado, se ha utilizado el acelerómetro de un teléfono para medir las vibraciones. Se ha descargado una aplicación que permite adquirir datos de aceleraciones en cada eje y guardarlos. Una vez hecho esto, se ha efectuado la FFT con las muestras de cada eje mediante Matlab. Los espectros obtenidos se muestran en la imagen 36, resultados totalmente coherentes con los representados en la imagen anterior.

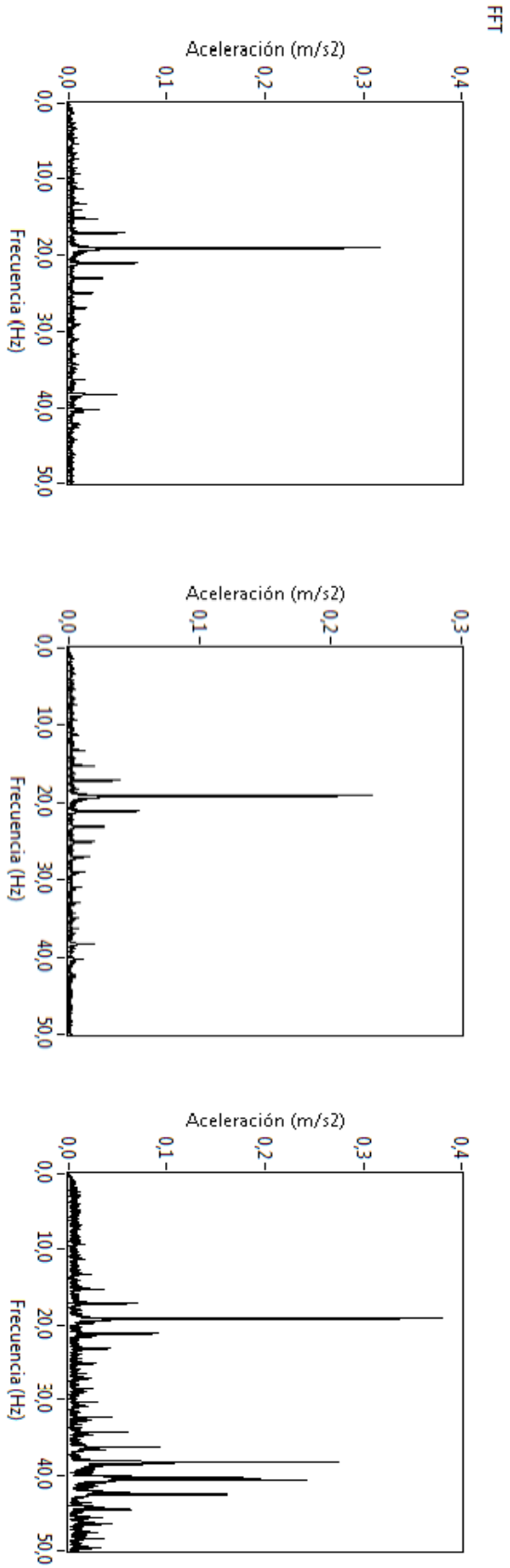

Imagen 35. Resultados obtenidos mediante el sistema desarrollado.

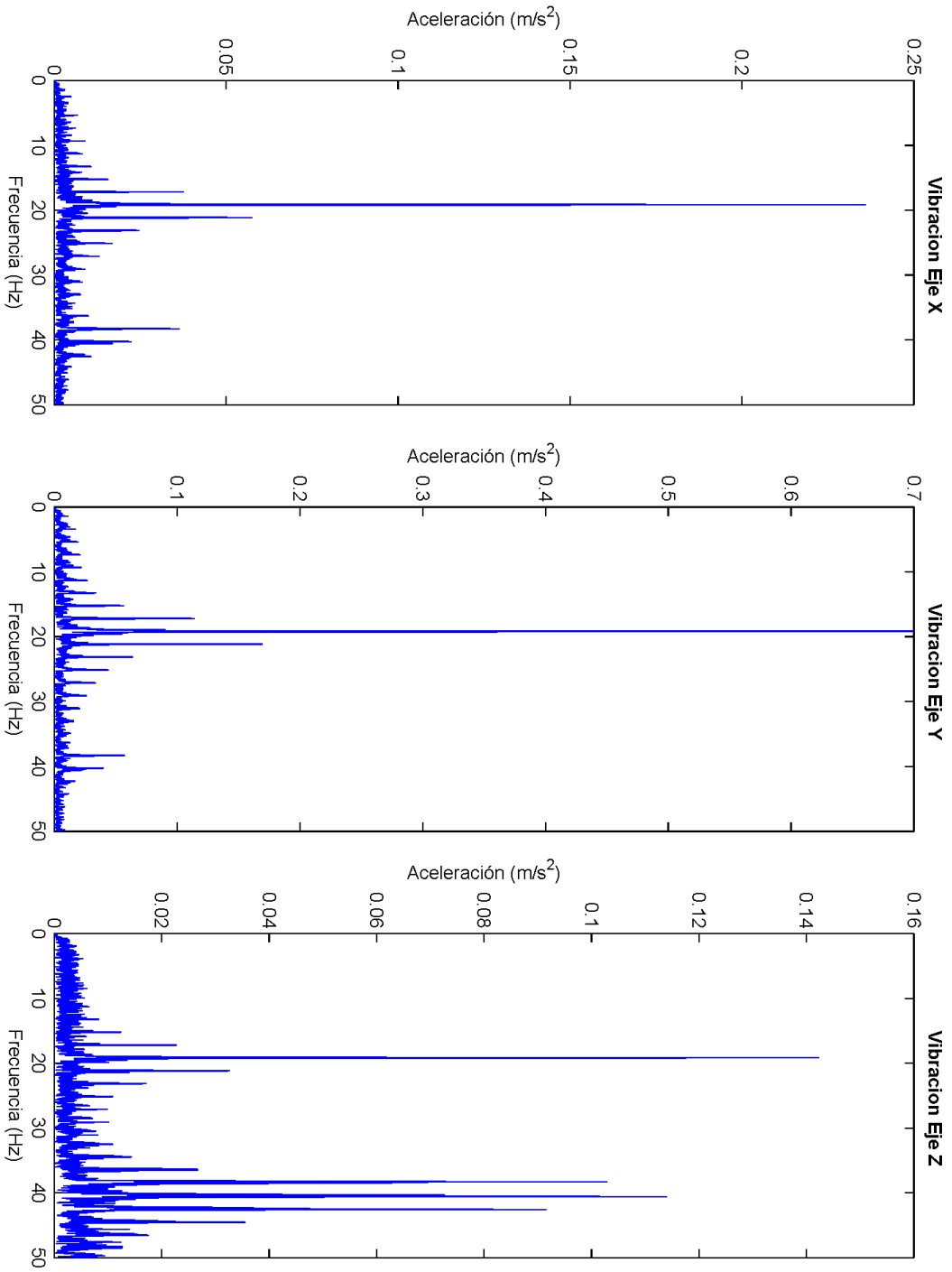

Imagen 36. Resultados obtenidos mediante Matlab.

6. Conclusiones.

Del trabajo realizado en este proyecto se concluye lo siguiente:

Se ha logrado desarrollar un sistema inalámbrico para la medición de las vibraciones con su posterior procesado mediante una FPGA. Aunque se ha hecho uso de un solo nodo para la adquisición de los datos, se han adquirido los conocimientos necesarios para la creación de una red de sensores, y este procedimiento se podría ampliar para una red más compleja.

Respecto al procesado de los datos, ha sido de gran utilidad el uso de los IP-COREs. Mediante estos componentes prediseñados, la tarea de diseño se ha simplificado notablemente, además de proporcionar núcleos muy optimizados en cuanto a recursos y velocidad. Resultaría complejo para un diseñador poco experimentado, desarrollar la misma aplicación con las mismas características, desarrollando él mismo, el código completo.

Analizando los resultados, se confirma el correcto funcionamiento del sistema completo. Sin embargo, quedan varias líneas abiertas:

Se ha indicado al principio de la memoria, el uso de *LabView* como herramienta de desarrollo. Una vez terminado con el diseño, la comunicación debiera ser directa entre la red de sensores y la FPGA.

Por problemas de soporte entre la FPGA *Spartan 3E* y *LabView*, no se ha podido realizar una comunicación en tiempo real. Con el uso de las FIFOs, se podría haber realizado el procesamiento en tiempo real, o al menos, a una velocidad muchísimo mayor. Al no poder utilizar esta opción, la velocidad del reloj ha quedado limitada a 1kHz; lejos del máximo posible: 136 MHz. Esto implica que el sistema desarrollado necesite dos horas aproximadamente para realizar la FFT, mientras que a una velocidad óptima, apenas duraría medio milisegundo aproximadamente.

7. Bibliografía.

[1] *Beginner's Guide to Machine Vibration*, 2006. Commtest Instruments Ltd, New Zealand, 1999.

[2] *Waspmote Acelerometer Programming Guide*, 2010. Libelium Comunicaciones Distribuidas S.L., Spain, 2010.

[3] *Agriculture 2.0 Technical Guide*, 2012. Libelium Comunicaciones Distribuidas S.L., Spain.

[4] *Events Technical Guide*, 2012. Libelium Comunicaciones Distribuidas S.L., Spain, 2012.

[5] *Waspmote ZigBee Networking Guide*, 2012. Libelium Comunicaciones Distribuidas S.L., Spain, 2012.

[6] *Waspmote Utilities Programming Guide*, 2011. Libelium Comunicaciones Distribuidas S.L., Spain, 2011.

[7] *Waspmote RTC Programming Guide*, 2012. Libelium Comunicaciones Distribuidas S.L., Spain, 2012.

[8] *LogiCORE IP Fast Fourier Transform v7.1*, 2011. Xilinx, United States, 2003.

[9] *Spartan 3E FPGA Family* Data Sheet, 2013. Xilinx, United States, 2005.

[10] "Transformada Rápida de Fourier", [Online] Eibar: Escuela Universitaria de Ingeniería Técnica Industrial. Disponible en: http://www.sc.ehu.es/sbweb/energias-renovables/MATLAB/datos/fourier/fourier_1.html

[11] J. Echanove, E. Asua, I. del Campo, "Integration of a Wireless Sensor-Actuator Network and an FPGA for Intelligent Inhabited Environments".

8. Anexos.

8.1 Código VHDL.

```
1   ----------------------------------------------------------------------------------
2   -- Engineer:       Adur Ayerza
3   --
4   -- Create Date:    13:23:31 03/30/2015
5   -- Module Name:    FFT - Behavioral
6   -- Project Name:   Vibraciones
7   -- Target Devices: Spartan 3E Starter Board
8   -- Description: Cálculo de la FFT y su posterior acondicionamiento.
9   ----------------------------------------------------------------------------------
10  library IEEE;
11  use IEEE.STD_LOGIC_1164.ALL;
12  use IEEE.STD_LOGIC_SIGNED.ALL;
13  use IEEE.NUMERIC_STD.ALL;
14  use IEEE.STD_LOGIC_ARITH.ALL;
15  use IEEE.STD_LOGIC_UNSIGNED.ALL;
16
17  entity FFT is
18     port (
19        Clk : in STD_LOGIC;
20        Start : in STD_LOGIC;
21        Unload : in STD_LOGIC;
22
23        Xn_re : in STD_LOGIC_VECTOR (15 downto 0);
24
25        Xn_index : out STD_LOGIC_VECTOR (15 downto 0);
26        Xk_index : out STD_LOGIC_VECTOR (15 downto 0);
27
28        Modulo : out STD_LOGIC_VECTOR (15 downto 0);
29
30        Rfd : out STD_LOGIC;
31        Dv : out STD_LOGIC;
32        Busy : out STD_LOGIC;
33        Edone : out STD_LOGIC;
34        Done : out STD_LOGIC);
35  end FFT;
36
37  architecture Behavioral of FFT is
38     --Señales FFT
39     signal Fwd : STD_LOGIC;
40     signal Fwd_we : STD_LOGIC;
41     signal Xn1_re : STD_LOGIC_VECTOR (11 downto 0);
42     signal Xk1_re : STD_LOGIC_VECTOR (11 downto 0);
43     signal Xk1_im : STD_LOGIC_VECTOR (11 downto 0);
44     signal Xk2_re : STD_LOGIC_VECTOR (11 downto 0);
45     signal Xk2_im : STD_LOGIC_VECTOR (11 downto 0);
46     signal Xk2_im_c : STD_LOGIC_VECTOR (11 downto 0);
47     signal Xn1_index : STD_LOGIC_VECTOR (11 downto 0);
48     signal Xk1_index : STD_LOGIC_VECTOR (11 downto 0);
49     signal Xn_im : STD_LOGIC_VECTOR (11 downto 0);
50     signal ground1 : STD_LOGIC_VECTOR (4 downto 0);
51     --Señales complemento a 2
52     signal Zero : STD_LOGIC_VECTOR (0 downto 0);
53     --Señales multiplicador
54     signal Square : STD_LOGIC_VECTOR (24 downto 0);
55     signal ground2 : STD_LOGIC_VECTOR (24 downto 0);
56     --Señales raíz cuadrada
57     signal result : STD_LOGIC_VECTOR (12 downto 0);
```

```
 58
 59      component R2L
 60         port (
 61            clk : in STD_LOGIC;
 62            start : in STD_LOGIC;
 63            unload : in STD_LOGIC;
 64            fwd_inv : in STD_LOGIC;
 65            fwd_inv_we : in STD_LOGIC;
 66
 67            xn_re : in STD_LOGIC_VECTOR (11 downto 0);
 68            xn_im : in STD_LOGIC_VECTOR (11 downto 0);
 69            xn_index : out STD_LOGIC_VECTOR (11 downto 0);
 70            xk_index : out STD_LOGIC_VECTOR (11 downto 0);
 71            xk_re : out STD_LOGIC_VECTOR (11 downto 0);
 72            xk_im : out STD_LOGIC_VECTOR (11 downto 0);
 73
 74            rfd : out STD_LOGIC;
 75            dv : out STD_LOGIC;
 76            busy : out STD_LOGIC;
 77            edone : out STD_LOGIC;
 78            done : out STD_LOGIC;
 79            --No usados
 80            blk_exp : OUT STD_LOGIC_VECTOR(4 DOWNTO 0));
 81      end component;
 82
 83      component Adder
 84         port (
 85            a : in STD_LOGIC_VECTOR(0 downto 0);
 86            b : in STD_LOGIC_VECTOR(11 downto 0);
 87            s : out STD_LOGIC_VECTOR(11 downto 0));
 88      end component;
 89
 90      component Mult
 91         port (
 92            ar: in STD_LOGIC_VECTOR(11 downto 0);
 93            ai: in STD_LOGIC_VECTOR(11 downto 0);
 94            br: in STD_LOGIC_VECTOR(11 downto 0);
 95            bi: in STD_LOGIC_VECTOR(11 downto 0);
 96            pr: out STD_LOGIC_VECTOR(24 downto 0);
 97            --No usados
 98            pi: out STD_LOGIC_VECTOR(24 downto 0));
 99      end component;
100
101      component Sqrt
102         port (
103            x_in : in STD_LOGIC_VECTOR(24 downto 0);
104            x_out : out STD_LOGIC_VECTOR(12 downto 0));
105      end component;
106
107      begin
108      Fwd <= '1';
109      Fwd_we <= '1';
110      Xn_im <= "000000000000";
111      Xk2_re <= Xk1_re;
112      Xk2_im <= Xk1_im;
113      Zero <= "0";
114
```

```
115      Bucle1:
116      for i in 0 to 11 generate
117         Xn1_re(i) <= Xn_re(i);
118         Xn_index(i) <= Xn1_index(i);
119         Xk_index(i) <= Xk1_index(i);
120      end generate;
121
122      Bucle2:
123      for i in 0 to 12 generate
124         Modulo(i) <= result(i);
125      end generate;
126
127      core1: R2L
128         port map(
129            Clk,
130            Start,
131            Unload,
132            Fwd,
133            Fwd_we,
134
135            Xn1_re,
136            Xn_im,
137            Xn1_index,
138            Xk1_index,
139            Xk1_re,
140            Xk1_im,
141
142            Rfd,
143            Dv,
144            Busy,
145            Edone,
146            Done,
147            ground1);
148
149      core2: Adder
150         port map(
151            Zero,
152            Xk2_im,
153            Xk2_im_c);
154
155      core3: Mult
156         port map(
157            Xk1_re,
158            Xk1_im,
159            Xk2_re,
160            Xk2_im_c,
161            Square,
162            ground2);
163
164      core4: Sqrt
165         port map(
166            Square,
167            result);
168
169   end Behavioral;
170
171
```