Capítulo 4

Aplicación desarrollada

Se trata de dar una visión de los distintos puntos de los que consta la aplicación o programa desarrollado en el transcurso del trabajo. Para una mejor compresión, se recomienda la lectura del Capítulo 3, en el cual se explican los conceptos básicos de programación en nesC. También puede ser útil comprobar la implementación de determinadas tareas o bloques directamente en el código fuente (véase el documento que acompaña a esta memoria).

La estructura de este extenso capítulo es a grandes rasgos como sigue. Se comienza por el sistema de identificación de los nodos dentro de la red. Después se detallan los módulos de temporización y relojes utilizados en el sensor. Operaciones iniciales (llevadas a cabo cuando arranca el mote). Gestión de entradas/salidas digitales de propósito general. Configuración y funcionamiento del convertidor analógico-digital del microcontrolador. Procesamiento de datos recogidos por los sensores, así como la gestión de la memoria flash externa. Protocolos de comunicación radio (colección, diseminación...), y comunicación serie. Posteriormente, se pasa a describir las funciones y tareas creadas en el código (variables usadas, procedimientos...). Se finaliza con la configuración de la pasarela Tmote Connect

(comunicación a través de TCP/IP) y funcionamiento de la aplicación para la monitorización y control en MATLAB.

4.1 Identificación de los nodos

Al cargar el *firmware* en cada uno de los nodos, se debe dar un identificador de red, accesible desde el código a través de la variable `TOS_NODE_ID`. Para distinguir a los nodos raíz o principales del resto (recogida de datos) se usará el siguiente criterio: son nodos raíz todos aquellos cuyo resto de dividir `TOS_NODE_ID` entre un divisor constante y fijado `ROOT_DIVIDER` sea cero.

Por ejemplo, si `ROOT_DIVIDER` se fija a 20 (por defecto) se considerarán nodos raíz todos aquellos con un identificador $0, 20, 40, 60, 80\ldots$

4.2 Bases de tiempo

Los diversos módulos que componen el microcontrolador del mote funcionan con pulsos de reloj digitales. El esquema con las bases de tiempo se muestra en la Figura 4.1.

- **ACLK**. Reloj auxiliar. Se basa en un oscilador de cristal externo conectado a través de los pines XIN y XOUT. En el caso del Tmote Sky, se trata de un reloj a 32 kHz binarios (2^{15} ciclos en un segundo). Posibilidad de escalar la frecuencia.
- **MCLK**. Reloj principal. Un selector permite elegir qué oscilador se usará para generar la señal: el anterior, uno externo (que no está disponible en Tmote Sky) o el oscilador interno digital. También se puede escalar la frecuencia.

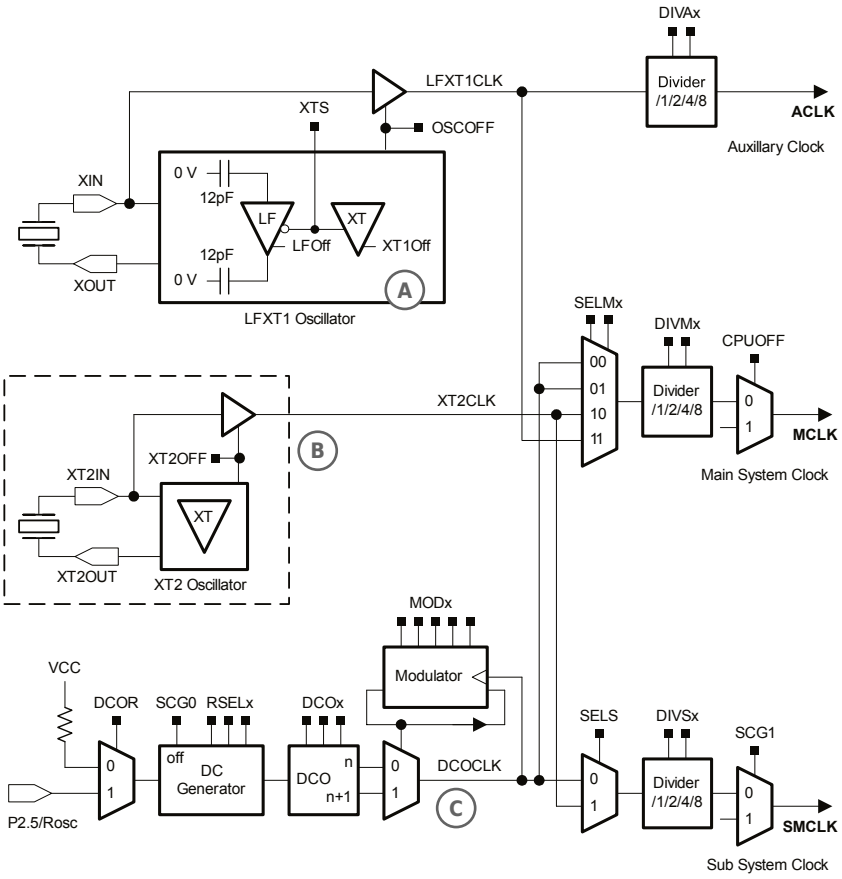


Figura 4.1 Diagrama de bloques de los relojes de la CPU

El bloque oscilador LFXT1 (A) está controlado por un oscilador externo conectado a los pines 8 (XIN) y 9 (XOUT) del microcontrolador. La señal XT2CLK (B) es opcional: en el modelo Tmote Sky no existe oscilador conectado a estos pines 53 (XT2IN) y 52 (XT2OUT). La señal DCOCLK es generada internamente (el pin P2.5 no está conectado) (C).

- **SMCLK**. Reloj del subsistema. Usará el oscilador digital interno DCOCLK. También es posible escalar la frecuencia del mismo. Por defecto, el reloj DCOCLK está configurado para funcionar a $4\,\text{MHz}$ binarios, o lo que es lo mismo, $2^{22} = 4194304$ ciclos por segundo. Posteriormente, se divide entre cuatro, lo que provoca que SMCLK funcione a $1\,\text{MHz}$ binario i.e. $2^{20} = 1048576$ ciclos por segundo[31].

[31] *Se podría pensar en no dividir y por tanto tener SMCLK a* $4\,\text{MHz}$ *binarios y así obtener más resolución. Sin embargo, la mayoría de componentes de periféricos (temporizadores, envío de datos…) están configurados para funcionar a* $1\,\text{MHz}$*, lo que provocaría que no funcionase ninguno.*

4.3 Temporización

Las conversiones analógico-digitales (tanto de tensión-intensidad como de otros sensores auxiliares) requieren de un sistema de temporización lo más exacto posible. El convertidor A-D del microcontrolador puede acceder a las distintas fuentes de reloj existentes (tanto internas como externas). La configuración del mismo detallada se tratará en el punto 4.7.

4.3.1 Watchdog

Se trata de un mecanismo de seguridad para garantizar la continuidad en la ejecución de código. Grosso modo, consiste en un contador de $16\,\text{bit}$, que se va incrementando según una fuente de reloj determinada. Cuando el bit más significativo pasa de 0 a 1 (i.e. valor 32768) se señala una interrupción y se provoca un reinicio forzado del mote.

Dentro del programa se incorporan instrucciones que vuelven a 0 al dicho contador cada cierto tiempo (lógicamente menor que el periodo de desbordamiento del contador), de forma que nunca se llegaría al reinicio.

Si por algún motivo el código queda bloqueado en algún punto, el contador no se reinicia, llega al valor máximo, desborda y el mote vuelve a arrancar. De esta forma volvería a funcionar normalmente. Este sistema debería usarse ante posibles errores eventuales de ejecución, memoria...

Concretamente para el sistema que nos ocupa, al no existir componentes "oficiales" para manejar el *watchdog*, se ha decidido trabajar directamente sobre los registros internos del MSP430F1611 (todos comienzan con las iniciales `WDT`[32]).

Las instrucciones usadas son las siguientes (`WDTCTL` es el registro de control):

```
WDTCTL = WDTPW + WDTHOLD;
WDTCTL = WDTPW + WDTCNTCL + WDTSSEL + 0x00;
```

[32] *WatchDog Timer*.

La primera línea detiene cualquier contador que pudiera haber en marcha. La segunda, resetea el contador, selecciona la fuente de reloj SMCLK (recuérdese que funciona a 32 kHz binarios) y selecciona el divisor para este reloj (en este caso 32768). Esto provocaría (si no se reseteara el contador), un reinicio cada segundo.

Para volver a reiniciar el contador, son necesarias estas instrucciones:

```
WDTCTL = WDTPW + WDTCNTCL;
WDTCTL = WDTPW + WDTHOLD;
WDTCTL = WDTPW + WDTCNTCL + WDTSSEL + 0x00;
```

La primera provoca un reseteo del contador interno. Las dos últimas son iguales que el caso anterior (reinician el sistema).

Si por algún motivo se desea detener el *watchdog*, se ejecuta:

```
WDTCTL = WDTPW + WDTHOLD;
```

Para facilitar la labor de programación de código, se crean unas macros (en archivo de cabecera `Maxwell.h`): `wdt_stop()` para detener el temporizador; `wdt_start()` para arrancarlo; y `wdt_touch()` para resetear el *watchdog*.

4.3.2 TimerMilliC

Componente genérico que proporciona un temporizador con precisión de milisegundos binarios.

INTERFACES PROPORCIONADAS

- `Timer<TMilli>`. Interfaz parametrizada que aporta funcionalidad propia de temporizadores. Algunos de los comandos: `startOneShot`, que inicia un temporizador durante un tiempo determinado una sola vez; `startPeriodic` hace lo propio pero de forma continua; `stop`, detiene un temporizador activo; `getNow` devuelve el valor actual del temporizador, etc. Proporciona un único evento: `fired`, que indica cuando expira un temporizador (bien una sola vez, o de forma periódica).

Este componente se utiliza después del arranque del sensor, para encender los tres LED alternativamente (ver 4.5.1).En los nodos de tipo raíz o base, se utiliza

otro temporizador para resetear (cada menos de un segundo) el *watchdog* del sistema. Para los nodos de recogida de datos, el reseteo del contador se produce en periódicamente cada vez que se realiza una conversión de medidas.

4.4 Arranque del mote

Son operaciones de inicio que tienen lugar cuando el dispositivo acaba de arrancar. El componente usado es `MainC`.

4.4.1 MainC

Maneja las interfaces de secuencia de arranque del sistema TinyOS. Accede al programador interno y los recursos *hardware*.

INTERFACES PROPORCIONADAS

- `Boot`. Notifica, a través de su evento `booted` que el mote ha arrancado todos sus componentes correctamente. Esto no implica que otros servicios (como el uso de la radio) no haya que iniciarlos aparte.

Primero se llama a la macro `welcome()`, que (en caso de que se esté en modo de depuración) muestra por la consola de comandos un mensaje de bienvenida incluyendo la versión del *firmware* instalada.

A continuación se comprueba si el identificador del nodo es correcto: si en el mote se ha cargado una imagen compilada para nodo raíz, su identificador debe ser también de raíz (según el criterio indicado en 4.1). De la misma forma ocurre con los nodos de recogida de datos.

Si existiera algún problema en este paso, el nodo no continuaría la ejecución de programa. De nuevo, si se ha especificado modo de depuración, se mostraría el correspondiente mensaje de error.

Finalmente, se activa el temporizador (interfaz llamada `TimerInit`) para realizar la comprobación de los LED. El temporizador vence periódicamente cada 1s.

4.5 Operaciones iniciales

Estas operaciones se inician con el evento `fired` de la interfaz `TimerInit` del temporizador.

4.5.1 LedsC

Control de los tres LED de propósito general que incluye el mote. Se accede a ellos mediante una numeración. Para el Tmote Sky son `led0` (rojo), `led1` (verde) y `led2` (azul). Tienen valores binarios asociados, siendo `led0` el menos significativo y `led2` el más.

INTERFACES PROPORCIONADAS

- `Leds`. Manipulación de LED. Carece de eventos. Los comandos son `set`, para fijar un valor binario representado en los LED. Por ejemplo, si se da un valor 3, se activarían `led0` y `led1`. Por el contrario, `get` devuelve el estado de los LED. Operaciones individuales para cada uno de ellos (`n` puede ser 0, 1 o 2: `lednOn` (encendido), `lednOff` (apagado), `lednToggle` (cambia el estado de encendido a apagado o viceversa).

* * *

Cada vez que tenga lugar (vence el temporizador) se active un LED distinto, comprobando así que funcionan correctamente. Después, el comportamiento varía dependiendo si se trata de un nodo raíz o un nodo de datos (el temporizador se detiene en ambos casos, pues ya no hará falta más).

- En caso de un nodo raíz, se realiza una llamada a la función `Arrancar`, concretamente iniciar el sistema de radio (parámetro `RADIO`).
- Para un nodo de datos, se llama a la función auxiliar `InicioMote` para inicializar algunas variables del programa, y después se comienza con el montado del bloque de memoria externa flash, según se describe en 4.9. Después, se llama a la función `Arrancar` para iniciar la radio.

Como se observa, en ambos casos se termina por (intentar) arrancar el sistema radio. En caso positivo, se recibe el evento `RadioControl.startDone`, indicando si ha habido o no error alguno. El próximo paso es iniciar el sistema de diseminación de datos, usando la función `Arrancar(DISSEM)`. Si se inicia correctamente, se pasa a iniciar el control de enrutamiento, de nuevo con la función auxiliar `Arrancar(ROUTING)`.

Si no ha habido error hasta ahora, el resto de las operaciones vuelve a distinguir los dos tipos de nodos:

- Para el nodo raíz, se hace `Arrancar(ROOT)` y así se establece ese nodo como base para el protocolo de colección de datos (más información en 4.10). También se inicia la comunicación serie a través del puerto USB del mote, con `Arrancar(SERIAL)`. Este tipo de nodo quedará a la espera de recepción de mensajes a través de la radio (provenientes de la red de nodos) o bien desde el puerto serie (desde un PC por ejemplo). Por ello es necesario arrancar el temporizador auxiliar para el *watchdog*, de forma que no se reinicie continuamente: llamada a `TimerWDT.startPeriodic(512)`: provoca un reseteo del temporizador cada $500\,\mathrm{ms}$.
- En caso de un nodo de recogida de datos, se llama a `Arrancar(ADCVI)`, para inicializar el convertidor analógico-digital para las medidas de tensión e intensidad. Cuando se haya iniciado, se recibe el evento `ResourceVI.granted`. En ese momento se llama a la tarea `Conversion` (véase 4.22).

4.6 Gestión entradas/salidas digitales

En este punto se trata cómo configurar ciertos pines del chip para que funcionen como entradas o salidas digitales.

4.6.1 HplMsp430GeneralIOC

Proporciona una abstracción de bajo nivel para las entradas/salidas de propósito general.

INTERFACES PROPORCIONADAS

- `HplMsp430GeneralIO`. Permite interactuar con los pines configurables del microcontrolador. Algunos de los comandos para usar: `selectIOFunc` fija un pin determinado como entrada/salida; `makeOutput` hace que un terminal sea de salida; y `clr` y `set` fijan un pin (de salida) a nivel bajo y alto respectivamente.

Debido a que existe más de un pin, es necesario distinguir las interfaces de cada uno de ellos, para ello se le asigna un alias a cada conector (interfaz genérica), creando instancias para cada pin. Algunas de ellas:

```
Interface HplMsp430GeneralIO as ADC0
Interface HplMsp430GeneralIO as ADC1
...
Interface HplMsp430GeneralIO as Port20
Interface HplMsp430GeneralIO as Port21
...
Interface HplMsp430GeneralIO as UTXD0
Interface HplMsp430GeneralIO as UTXD1
```

* * *

Preparación de las salidas, que pueden usarse para controlar la carga conectada o para activar algún dispositivo mediante una señal digital. Concretamente, serán los pines 23 y 26 de la CPU del mote, los cuales pueden conectarse a través del puerto de expansión U28 que existe en el sensor (ver Figura 2.2). Es decir, los alias

```
Interface HplMsp430GeneralIO as Port23
Interface HplMsp430GeneralIO as Port26
```

se asignan respectivamente a `CargaA` y `CargaB`.

Mediante `selectIOFunc` se fija el pin concreto para que funcione como entrada/salida. Después, `makeOutput` establecerá el pin como salida.

4.7 Conversión analógico-digital

La conversión analógico-digital es otro aspecto importante en el diseño del sistema. TinyOS proporciona varios componentes para este propósito, dependiendo de los requerimientos de muestreo.

En la Figura 4.2 se muestra un esquema con los componentes del convertidor interno del mote.

Se trata de un convertidor con 12 bits de resolución, lo cual quiere decir que devolverá un resultado entre $\left[0, 2^{12} - 1\right] = \left[0, 4095\right]$. La expresión analítica para obtener el valor es

$$N_{ADC} = 4095 \frac{V_{IN} - V_{R-}}{V_{R+} - V_{R-}} \tag{4.1}$$

Donde V_{IN} es la tensión de entrada y V_{R+} y V_{R-} son las referencias de tensión (positiva y negativa respectivamente).

Para la monitorización de la potencia eléctrica (aparte de otros parámetros) hay que muestrear dos canales analógicos, uno correspondiente a tensión y el otro a intensidad. Además, por el Principio de tiempo de muestreo reducido[33], hay que

[33] *Para más información, consultar 4.23.1*

tomarlas alternativamente. Todo esto sin olvidar la frecuencia de muestreo elegida: $F_S = 1600\,\text{Hz}$, que equivale a tomar una muestra cada $T = 625\,\text{us}$[34].

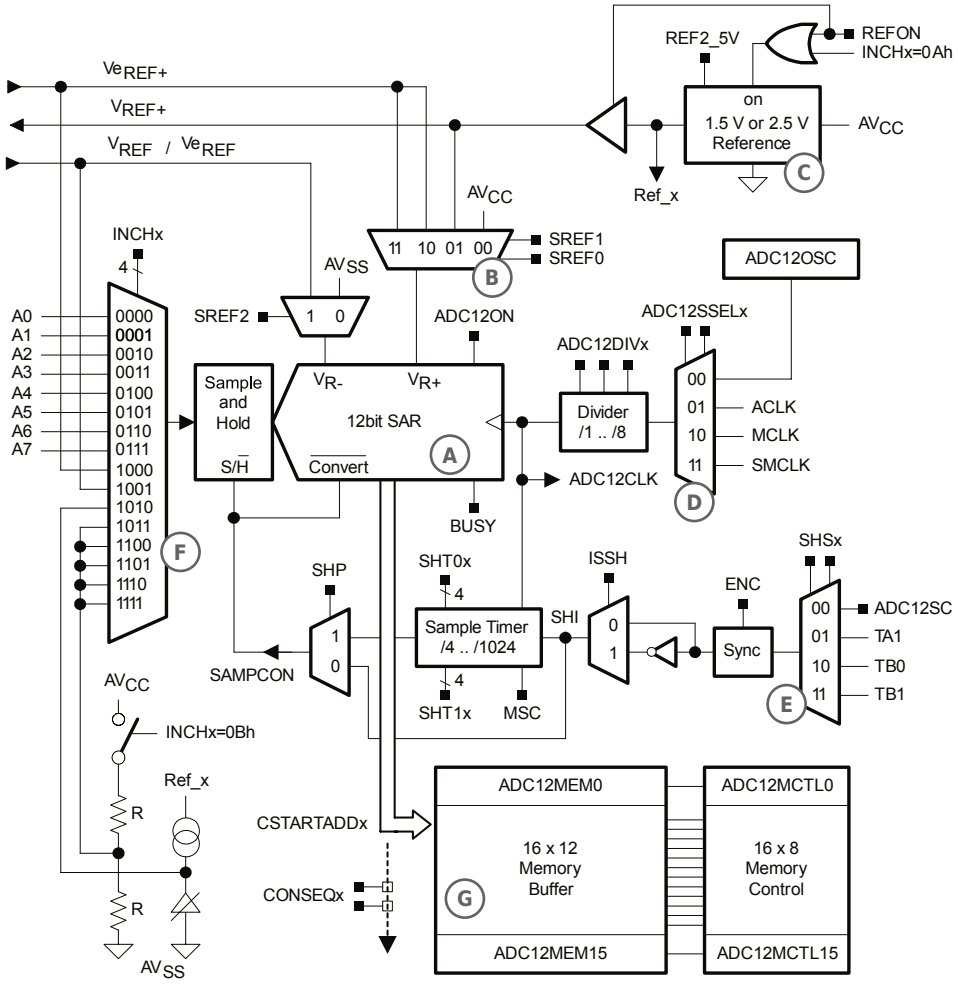


Figura 4.2 Diagrama de bloques del convertidor A-D de 12bits

El convertidor está representado por el bloque 12bit de tipo SAR (A). Necesita referencias de tensión (V_{R+} y V_{R-}), seleccionables mediante los bits SREFx (B). Una opción es utilizar el generador de tensión interno de $1.5\,\text{V}$ o $2.5\,\text{V}$ (C). Por otro lado, utiliza una señal de reloj configurable entre las fuentes disponibles (D). También hay que controlar el tiempo que se muestrea y retiene la señal (S/H) (E). La señal a convertir se puede seleccionar usando los bits INCHx (los analógicos A0 hasta A7 son externos) (F). Los resultados de las conversiones se almacenan en un buffer interno ADC12MEMx (G).

Existen componentes para este propósito que son independientes de la plataforma en la que se usan, pero no poseen tantas opciones de configuración. Es por ello por lo que se recurre a componentes propios del chip del mote Tmote Sky.

Todos ellos se nombran como `Msp430Adc12Client...` la diferencia entre ellos está en si generan automáticamente la referencia de voltaje interna (denomina-

[34] *La justificación de estos valores se puede encontrar en 4.24.2.*

da AutoRVG) y/o si tienen acceso directo a memoria (o DMA). Esto da lugar por tanto a cuatro posibles configuraciones: Msp430Adc12ClientC, Msp430Adc12ClientAutoRVGC, Msp430Adc12ClientAutoDMAC y Msp430Adc12ClientAutoDMA_RVGC.

Para el proyecto, se desechan las opciones con DMA (los dos últimos componentes), ya que no soportan el muestreo de varios canales. De entre los otros dos, se usará Msp430Adc12ClientAutoRVGC, que configura automáticamente la referencia de tensión.

4.7.1 Msp430Adc12ClientAutoRVGC

INTERFACES PROPORCIONADAS

- Msp430Adc12MultiChannel. Manejo del convertidor A-D para que trabaje con varios canales. Contiene dos comandos: configure, en el cuál se especifican los tiempos de muestreo, canales a usar, dónde guardar los resultados... y getData que inicia el proceso de conversión según se haya especificado en la configuración. El evento dataReady indica que la conversión ha finalizado y los resultados ya están disponibles en memoria.
- Msp430Adc12Overflow. Esta interfaz se usa para controlar posibles fallos de desbordamiento, tanto en memoria (con el evento memOverflow) como en los valores del temporizador (evento conversionTimeOverflow).
- Resource. Esta interfaz es proporcionada por los periféricos de la CPU, los cuáles pueden estar compartidos por varios componentes. Mediante el comando request se hace una petición para acceder al recurso (en este caso el convertidor). El componente avisará si está disponible mediante el evento granted. Para liberar al convertidor, una vez se termine de usar, se utiliza el comando release.

INTERFACES USADAS

- `AdcConfigure`. Sólo contiene un comando: `getConfiguration`. Al tratarse de una interfaz usada, habrá que programar dicho comando. En este caso, sólo hay que devolver una estructura con la configuración (a continuación).

4.7.2 Configuración del convertidor

La configuración se realiza a través de estructuras de tipo `msp430adc12_channel_config_t`. Los campos de ésta son los siguientes (las constantes simbólicas que aparecen están definidas en el fichero de cabecera correspondiente `Msp430Adc12.h`).

- `inch`. Especifica el canal de entrada (analógico) del convertidor. Existen varias opciones para la entrada (Tabla 4.1).

Valor de INCH	Descripción	Selector canales Figura 4.2
INPUT_CHANNEL_An	Indica una entrada externa analógica del integrado. El parámetro "n" puede variar entre 0 y 7 (ocho canales en total).	0 a 7
EXTERNAL_REF_VOLTAGE_CHANNEL	Entrada conectada directamente al terminal VeREF+ del chip.	8
REF_VOLTAGE_NEG_TERMINAL_CHANNEL	Está conectado al pin denominado VeREF-/VREF-.	9
TEMPERATURE_DIODE_CHANNEL	Conectado al sensor interno de temperatura (diodo) con el que cuenta el microcontrolador.	10
SUPPLY_VOLTAGE_HALF_CHANNEL	Mantiene la mitad de la tensión de alimentación del chip, esto es: (AVCC-AVSS)/2.	11

Tabla 4.1 Valores posibles para el campo INCH

- `sref`. Referencia de voltaje para el convertidor. En teoría, se pueden especificar hasta seis opciones diferentes, recogidas en la Tabla 4.2.

Valor de SREF	Referencia positiva	Referencia negativa
REFERENCE_AVcc_AVss	AVCC	AVSS
REFERENCE_VREFplus_AVss	VREF+	AVSS
REFERENCE_VeREFplus_AVss	VeREF+	AVSS
REFERENCE_AVcc_VREFnegterm	AVCC	VREF-/VeREF-
REFERENCE_VREFplus_VREFnegterm	VREF+	VREF-/VeREF-
REFERENCE_VeREFplus_VREFnegterm	VeREF+	VREF-/VeREF-

Tabla 4.2 Valores posibles para el campo SREF

Sin embargo, para el modelo Tmote Sky, hay que tener en cuenta que los pines Ve_{REF+} y V_{REF-}/Ve_{REF-} están conectados entre sí y a tierra, por lo que no es posible usarlos como referencias. Con esto, se pueden descartar dos opciones, pues tanto la referencia positiva como la negativa serían la misma: `REFERENCE_VeREFplus_AVss` y `REFERENCE_VeREFplus_VREFnegterm`.

- `ref2_5v`. Controla el generador de tensión interno del microcontrolador. Cuenta con tres opciones: `REFVOLT_LEVEL_1_5`, `REFVOLT_LEVEL_2_5` y `REFVOLT_LEVEL_NONE`, según se requieran $1.5\,\mathrm{V}$, $2.5\,\mathrm{V}$ o ninguna (respectivamente). Las opciones de este parámetro están muy relacionadas con las del anterior. Si en la referencia positiva aparece AV_{CC}, la referencia de voltaje interna no será necesaria, y bastará con establecerla a `REFVOLT_LEVEL_NONE`. Sin embargo, si la referencia positiva elegida es V_{REF+}, aquí si es necesario especificar una tensión interna de referencia: $1.5\,\mathrm{V}$ o $2.5\,\mathrm{V}$.
- `adc12ssel`. Selección de la fuente de tiempo (reloj) para realizar las conversiones (muestreo y retención). Se permiten cuatro valores, según la Tabla 4.3.

Valor de ADC12SSEL	Descripción
SHT_SOURCE_ADC12OSC	Oscilador interno a frecuencia de 5MHz. Notar que su frecuencia depende en gran medida de factores como la temperatura, tensión de alimentación...
SHT_SOURCE_ACLK	Conectado a un oscilador externo en los pines XIN y XOUT. En el caso del Tmote Sky, se trata de un cristal a 32kHz. Posee un divisor de frecuencia configurable por SW a 1, 2, 4 y 8.
SHT_SOURCE_MCLK	Representa al reloj maestro, usado por la CPU y el sistema principal. Permite seleccionar entre distintas bases de tiempo, por ejemplo, el oscilador anterior. También tiene un divisor de frecuencia configurable.
SHT_SOURCE_SMCLK	Reloj maestro del subsistema, útil para módulos periféricos del microcontrolador. Puede elegir entre un oscilador externo o uno digital. También es posible escalar la frecuencia de salida.

Tabla 4.3 Valores posibles para el campo ADC12SSEL

- `adc12div`. Divisor programable de la base de tiempos que recibe el convertidor. La entrada de tiempo (elegida en el parámetro anterior) puede ser escalada en 8 valores: `SHT_CLOCK_DIV_n`, donde `n` indica el factor por el que dividir, desde 1 hasta 8.
- `sht`. Tiempo de muestreo y retención. Se especifica el número de ciclos de reloj en los cuáles se está muestreando la señal de entrada. El valor es `SAMPLE_HOLD_n_CYCLES`. `n` puede ser 4, 8, 16, 32, 64, 128, 192, 256, 384, 512, 768 o 1024 ciclos. El tiempo de muestreo y retención no puede ser todo lo pequeño que se desee: existe una restricción que depende de la resistencia del sensor y la circuitería interna (expresión dada en la hoja de características del MSP430F1611).
- `sampcon_ssel`. Selección de la fuente de tiempo para fijar los instantes de conversión (periodo de muestreo). En la Tabla 4.4 se recogen los posibles valores.

Valor de SAMPCON_SSEL	Descripción
SAMPCON_SOURCE_TACLK	Entrada externa para el TimerA del microcontrolador. En el modelo Tmote Sky no es posible usarlo, pues está destinado a comunicarse con el chip de radio.
SAMPCON_SOURCE_ACLK	Reloj auxiliar. Conectado a un oscilador externo en los pines XIN y XOUT. Mismo reloj que el visto anteriormente.
SAMPCON_SOURCE_SMCLK	Reloj principal de subsistema, útil para módulos periféricos del microcontrolador. Igual que el visto anteriormente.
SAMPCON_SOURCE_INCLK	Entrada de reloj (externo) general. En Tmote Sky está usado para otro fin.

Tabla 4.4 Valores posibles para el campo SAMPCON_SSEL

- `sampcon_id`. Divisor de entrada para el contador, según lo seleccionado en el punto anterior. El valor se especifica mediante `SAMPCON_CLOCK_DIV_n`, con `n` igual a 1, 2, 4 u 8.

En caso de que se quieran muestrear varios canales adicionales (de forma secuencial), se utilizará otra estructura, de tipo `adc12memctl_t`. En ella sólo habrá que indicar dos parámetros: el canal de entrada `inch` y la referencia de tensión `sref`. Los demás parámetros se toman de la estructura ya vista.

A continuación se trata la conversión de canales de tensión e intensidad por una parte, y sensores auxiliares por otra.

4.7.3 Conversión de tensión e intensidad

En primer lugar, hay que declarar las estructuras que definan los parámetros del convertidor. En primer lugar, el tipo `msp430adc12_channel_config_t`:

```
const msp430adc12_channel_config_t configVI {
...
};
```

Los campos particularizados para este caso quedarán

- `inch`. El canal de tensión se asociará al analógico 0: `INPUT_CHANNEL_A0`.
- `sref`: `REFERENCE_VREFplus_AVss`. Indica que se usará el generador interno de tensión como referencia positiva del convertidor. La negativa será tierra.
- `ref_2_5v`: `REFVOLT_LEVEL_2_5`. La referencia se fija a $2.5\,\mathrm{V}$.
- `adc12ssel`. El reloj a utilizar es el denominado SMCLK: `SHT_SOURCE_SMCLK`.
- `adc12div`: `SHT_CLOCK_DIV_1`. No se altera la frecuencia base del reloj, con lo que quedaría a $1\,\mathrm{MHz}$ binario.
- `sht`: `SAMPLE_HOLD_4_CYCLES`. La retención de la señal para poder realizar la conversión será de 4 ciclos de reloj SMCLK, lo que equivale a unos $3.8147\,\mathrm{us}$. Se ha fijado al mínimo posible, ya que las señales con armónicos pueden variar muy rápidamente. No obstante, dependiendo del posible transductor aplicado al sensor, podría ser necesario aumentar este tiempo.
- `sampcon_ssel`: `SAMPCON_SOURCE_SMCLK`. La fuente de tiempo para el muestreo es también SMCLK.
- `sampcon_id`: `SAMPCON_CLOCK_DIV_1`. Tampoco aquí se escala la frecuencia del reloj: $1\,\mathrm{MHz}$ binario.

Hay que especificar el otro canal a muestrear (intensidad). Para ello, se usa la otra estructura auxiliar del tipo `adc12memctl_t`. y que se define como

```
adc12memctl_t memctlVI[1];
```

En ella solo se indican dos parámetros:

- `inch`. Valor `INPUT_CHANNEL_A1` (canal analógico 1).
- `sref`: `REFERENCE_VREFplus_AVss`, la misma elegida antes.

La conversión se prepara usando la interfaz ya comentada en 4.7.1 `Msp430Adc12MultiChannel`, llamando al comando `configure` con los siguientes parámetros:

```
command error_t configure(msp430adc12_channel_config_t *config,
  adc12memctl_t *memctl, uint8_t numMemctl, uint16_t *buffer,
  uint16_t numSamples, uint16_t jiffies)
```

- `*config`. Configuración principal del convertidor, según la estructura vista en el apartado anterior `msp430adc12_channel_config_t` (se envía un puntero a ella).
- `memctl`. Lista de posibles canales adicionales a muestrear. En el caso de medidas de potencia, sólo se añade uno (el A1-intensidad) con la estructura auxiliar `adc12memctl_t`.
- `numMemctl`. Indica el número de canales añadidos en la lista anterior (sin contar el primero). En este caso, 1.
- `*buffer`. Puntero donde se almacenarán los resultados de las conversiones. Los valores se guardan alternativamente. Por ejemplo, si se usan los canales A0 y A1, los resultados quedarían como canal 0-1-0-1-0-1...
- `numSamples`. Número total de muestras a tomar. El número de muestras totales debe ser un múltiplo de los canales usados, con el fin de que el *buffer* se llene completamente[35].
- `jiffies`. Periodo de muestreo, según el tiempo especificado en los parámetros `sampcon_ssel` y `sampcon_id`. Como se ha tomado SMCLK, se tienen $2^{20} = 1048576$ por cada segundo. El tiempo entre dos muestras consecutivas de la misma magnitud había de ser $T = 625\,\text{us}$, sin embargo, al estar muestreando alternativamente los canales, el tiempo de muestreo se reduce a la mitad: $312.5\,\text{us}$. El valor entero más próximo para este tiempo es 328 jiffies. Esto supone muestrear cada $312.8052\,\text{us}$ aprox. cometiéndose un error del 0.0977%.

[35] *De forma matemática, el resto de dividir $numSamples$ entre los canales totales $numMemctl + 1$ ha de ser cero.*

Para muestrear cuatro periodos consecutivos[36] (equivalentes a $80\,\mathrm{ms}$), a una frecuencia $F_S = 1600\,\mathrm{Hz}$ serán necesarias $80\,\mathrm{m}\cancel{\mathrm{s}}\cdot 1600\cancel{\mathrm{s}^{-1}} = 128$ muestras para cada canal, luego el *buffer* de memoria tendrá 256 muestras.

4.7.4 Conversión de sensores auxiliares

Se ha pensado también en tomar medidas de otros sensores, como pueden ser la temperatura interna del microcontrolador, la tensión de alimentación del circuito y la posibilidad de añadir posteriormente más entradas.

La forma de configuración es similar al caso anterior. Para el primer canal a tratar (será el de temperatura interna) se usa la estructura `msp430adc12_channel_config_t`, quedando como sigue

```
const msp430adc12_channel_config_t configAux = {
  inch: TEMPERATURE_DIODE_CHANNEL,
  sref: REFERENCE_VREFplus_AVss,
  ref2_5v: REFVOLT_LEVEL_2_5,
  adc12ssel: SHT_SOURCE_SMCLK,
  adc12div: SHT_CLOCK_DIV_1,
  sht: SAMPLE_HOLD_64_CYCLES,
  sampcon_ssel: SAMPCON_SOURCE_SMCLK,
  sampcon_id: SAMPCON_CLOCK_DIV_1
};
```

Para canales adicionales, se usa el otro tipo de estructura (`adc12memctl_t`). Se añaden tres más, por lo que necesita un *array* (o tabla) de tres elementos.

```
adc12memctl_t memctlAux[3];
```

Y la configuración quedaría:

```
memctlAux[0].inch=SUPPLY_VOLTAGE_HALF_CHANNEL;
memctlAux[0].sref=REFERENCE_VREFplus_AVss;
memctlAux[1].inch=INPUT_CHANNEL_A2;
memctlAux[1].sref=REFERENCE_VREFplus_AVss;
memctlAux[2].inch=INPUT_CHANNEL_A3;
memctlAux[2].sref=REFERENCE_VREFplus_AVss;
```

[36] *Justificación en Tarea CalcPotencia (4.23).*

Es decir, se controla la tensión de alimentación, y después se añaden los canales analógicos A2 y A3 de propósito general (en principio no están asignados a ninguna magnitud).

4.8 Procesamiento sensores auxiliares

Cada vez que se realice una conversión, se tomarán las medidas proporcionadas por los sensores de temperatura, tensión y otros dos adicionales. Para cada sensor se toman $16 = 2^4$ muestras.

Después, se aplica un proceso de *sobremuestreo* (*oversampling*): (Texas Instruments, 2006), (ATMEL, 2005). Consiste en sumar todas las muestras del sensor correspondiente. Al ser las muestras de $12\,\mathrm{bit}$, la suma podrá almacenarse en una variable de $12 + 4 = 16\,\mathrm{bit}$. El resultado obtenido se divide entre $4 = 2^2$, consiguiendo así un valor de $14\,\mathrm{bit}$. De esta forma se han conseguido $2\,\mathrm{bit}$ adicionales de resolución; el rango de medidas se encontrará ahora entre $\left[0, \frac{\left(2^{12}-1\right)\cdot 16}{4}\right] = \left[0, 16380\right]$.

4.8.1 Temperatura interna

Para el caso de la temperatura, se recurre a la ecuación de transferencia que proporciona el manual del microcontrolador:

$$T_C = \frac{V_D - 0.986}{0.00355} \tag{4.2}$$

que relaciona la tensión medida en el diodo interno V_D con la temperatura T_C en grados Celsius. Como la resolución es ahora de $14\,\mathrm{bit}$, la conversión entre tensión del sensor y valor digital será:

$$V_D = \frac{N_{CAD}\cdot 2.5}{16380} \tag{4.3}$$

Introduciendo la expresión (4.3) en (4.2) resulta:

$$T_C = \frac{\frac{N_{CAD}}{6552} - 0.986}{0.00355} = \frac{N_{CAD} - 6460.272}{23.2596} \tag{4.4}$$

Como no se trabaja con números decimales, será necesario multiplicar (en numerador y denominador) por el factor 10^4:

$$T_C\left(^{\circ}\mathrm{C}\right) = \frac{10000 N_{CAD} - 64602720}{232596} \tag{4.5}$$

Para conseguir un valor más preciso, se multiplica la expresión por 10, resultando el valor de la temperatura en décimas de grado:

$$T_C\left(^{\circ}\mathrm{C}{\cdot}10\right) = \frac{10^5 N_{CAD} - 646027200}{232596} \tag{4.6}$$

Por ejemplo, un valor 204 indicará $20.4^{\circ}\mathrm{C}$.

4.8.2 Tensión de alimentación

El sensor proporciona la tensión de alimentación dividida entre dos:

$$\frac{A\,V_{CC} - A\,V_{SS}}{2} \tag{4.7}$$

Para conseguir la tensión total a partir de la lectura del convertidor (ahora de $14\,\mathrm{bit}$) se hará

$$V\left(\mathrm{V}\right) = \frac{N_{ADC}{\cdot}2.5}{16380}{\cdot}2 = \frac{N_{ADC}}{3276} \tag{4.8}$$

Si se requiere más precisión, por ejemplo, la medida en mV bastará multiplicar la expresión por 10^3:

$$V\left(\mathrm{mV}\right) = \frac{250{\cdot}N_{ADC}}{819} \tag{4.9}$$

Es decir, un valor 2850 equivale a $2.850\,\mathrm{V}$.

4.8.3 Otros sensores

Existen dos canales analógicos adicionales de propósito general. Los valores que se obtienen tras el proceso de sobremuestreo también estarán en el rango $\left[0,16380\right]$. Dependiendo de la magnitud, se usarán unos factores determinados.

4.9 Gestión memoria flash externa

Se trata de almacenar datos en el chip de memoria externa (ST M25P80). Por ejemplo, se guardan los umbrales máximos para la distorsión, temperatura interna y tensión de alimentación, de forma que si existe una interrupción en la alimentación del nodo, dichos datos se mantendrán la próxima vez que arranque.

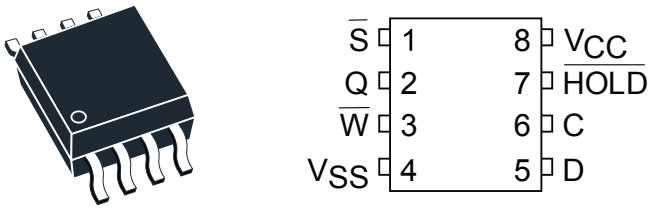


Figura 4.3 Encapsulado (a) y conexiones (b) del M25P80 (ST)

Memoria de tipo flash de $8\,\mathrm{Mb}$. Está organizada en 16 sectores, cada uno de ellos contiene 256 páginas. En cada página se programan $256\,\mathrm{B}$. Alimentación desde $2.7\,\mathrm{V}$ hasta $3.6\,\mathrm{V}$ Consultar (STMicroelectronics, 2002).

4.9.1 ConfigStorageC

Control de escritura, lectura... en memoria externa. El componente está parametrizado con un valor:

```
ConfigStorageC(volume_id_t volume_id)
```

En `volume_id` hay que especificar el nombre que se ha dado al bloque de memoria para trabajar sobre él. En este caso será `VOLUME_CONFIG`.

INTERFACES PROPORCIONADAS

- `Mount`. Su comando `mount` prepara un volumen de memoria en particular. Antes de usar la memoria, debe usarse este comando. Una vez hecha la llamada (y que se haya aceptado), hay que esperar al evento `mountDone`, indicando que el montado se ha realizado (`SUCCESS`) o si por el contrario ha habido algún error (`FAIL`).
- `ConfigStorage`. Manejo de los volúmenes de memoria. El comando `read` lee un bloque de memoria a partir de una dirección dada y con un tamaño de datos a leer; también necesita un *buffer* de memoria donde guardar los datos leídos. Puede devolver error si el volumen no ha sido montado, parámetros incorrectos, ocupado... El evento `readDone` se recibe cuando se hayan leído los datos sin problemas, o si ha habido algún error al realizar la lectura.
- Por su parte, el comando `write` se encarga de guardar en memoria determinados datos. Igual que antes, hay que indicar una dirección de memoria, el *buffer* donde están dichos datos y el tamaño de los mismos. Puede devolver distintos códigos de error. `writeDone` es el evento para informar que se ha escrito o se ha intentado escribir pero ha habido algún fallo.
- El comando `commit` (confirmación) obliga al componente a finalizar cualquier operación de escritura. Es necesario para asegurar los datos en la memoria. Si la petición es correcta, devolverá `SUCCESS`. Hay que esperar al evento `commitDone`. Si no ha habido ningún error, se puede asegurar que los datos han sido grabados en la memoria no volátil.
- El comando `valid` indica mediante un valor booleano si el volumen contiene datos válidos y es funcional. En caso contrario, debe llamarse a `commit` para validarlo.

4.10 Colección de datos

Para la recuperación de datos de una red de sensores existían dos alternativas vistas en el capítulo anterior: MultihopLQI y CTP. En el trabajo que se desarrolla, se ha optado por MultihopLQI (aprovechamiento de características propias del CC2420).

A continuación se describen los componentes implicados en este protocolo y sus interfaces.

4.10.1 ActiveMessageC

Proporciona la mayoría de interfaces necesarias para el control de mensajes radio.

INTERFACES PROPORCIONADAS

- `SplitControl`. Permite activar o desactivar el funcionamiento del componente correspondiente (y de sus componentes asociados). Por ejemplo, mediantes los comandos `start` y `stop` se realizan las mencionadas acciones. A su vez, se tienen sendos eventos `startDone` y `stopDone`. El primero avisa de que el componente ha sido iniciado y está listo para recibir más comandos; el segundo, que el componente ha sido detenido.

4.10.2 CollectionC

Funcionalidad específica para poner en marcha el protocolo de colección de datos en una red de nodos.

INTERFACES PROPORCIONADAS

- `StdControl`. Es similar a la interfaz `SplitControl`, si bien ésta sólo contiene dos comandos: `start` (para arrancar el componente y sus posibles componentes asociados) y `stop` (para detenerlo).

- `RootControl`. Permite configurar un nodo de la red para que se comporte como raíz o principal (y que el resto de la red tenga constancia de ello). Para ello, basta usar el comando `setRoot`. El comando `unsetRoot` libera a un nodo de su condición de raíz. Por último, el comando `isRoot` devuelve mediante una variable booleana si un nodo es principal o no.
- `Receive`. Esta interfaz permite la recepción de mensajes. En el caso de una red de colección de datos, este evento está justificado para el nodo raíz. Su único evento es `receive`. Conviene resaltar se trata de una interfaz parametrizada. Hay que especificar un valor concreto para cada tipo de mensaje que se vaya a recibir.

4.10.3 CollectionSenderC

Envío de mensajes a una red de colección de datos. Este componente también está parametrizado. Es necesario incluir un identificador para los mensajes que se envíen con este componente. De esta forma, la interfaz antes comentada `Receive` puede distinguirlos.

INTERFACES PROPORCIONADAS

- `Send`. Envío de un paquete de datos (comando `send`). En el caso que nos ocupa, será a través de la interfaz radio. No hay que especificar dirección de destino: el propio protocolo de colección de datos se encargará de llevarlo hacia el nodo apropiado (nodo raíz del árbol).

4.10.4 Archivo Makefile

A la hora de realizar la compilación de la aplicación, hay que añadir unas reglas al archivo `Makefile` para incluir el protocolo (versión LQI):

```
CFLAGS += -I$(TOSDIR)/lib/net \
          -I$(TOSDIR)/lib/net/lqi
```

4.11 Diseminación de datos

Aunque se usan los componentes que implementan el protocolo de difusión de datos en una red, también se va a dotar de funcionalidad para que el nodo raíz o principal envíe mensajes destinados a un nodo en concreto (y no a toda la red).

Al igual que con la colección, existían distintas alternativas de implementar este protocolo. En este caso, se ha optado por DHV (los tipos de mensajes están bien definidos, también se obtiene una convergencia de la red rápida).

4.11.1 DisseminationC

Funcionalidad para hacer funcionar el protocolo de difusión de datos. Básicamente se utiliza para activar el protocolo.

INTERFACES PROPORCIONADAS

- `StdControl`. Posee dos comandos: `start`, para arrancar el componente y sus posibles componentes asociados; y `stop` para detenerlo.

4.11.2 DisseminatorC

Permite la difusión (y posterior recepción del resto de nodos) de valores desde el nodo principal. Este componente está parametrizado con dos valores:

```
DisseminatorC(typedef t, uint16_t key)
```

El valor `t` indica qué tipo de datos se van a difundir; `key` es un valor único $[0, 65535]$ para distinguir posibles dominios de difusión (podría interesar difundir unos valores a ciertos nodos de la red y otros valores a otros nodos).

INTERFACES PROPORCIONADAS

- `DisseminationUpdate`. Cuando se llama a su único comando `change`, el valor especificado en él se transfiere a todos los nodos de la red a través de la radio. Es una interfaz genérica i.e. hay que especificar en ella el tipo de dato

(typedef t) que se quiere enviar: entero de 8 bit, entero de 16 bit, o incluso una estructura de datos.

- DisseminationValue. Complementa a la anterior. Cuando se recibe el evento changed indica que el nodo ha recibido una actualización del valor de una variable (el nodo raíz habrá usado el comando change anterior). Si se desea conocer el nuevo valor diseminado, basta con llamar a get. Lógicamente, esta interfaz también es genérica: el tipo de datos typedef t con el que trabaje ha de coincidir con el establecido en la interfaz DisseminationUpdate.

4.11.3 Archivo Makefile

Las siguientes líneas en el archivo de compilación incluyen los componentes necesarios para la diseminación:

```
CFLAGS += -I$(TOSDIR)/lib/net \[37]
          -I$(TOSDIR)/lib/net/dhv \
          -I$(TOSDIR)/lib/net/dhv/interfaces \
```

4.12 Datos puerto serie

Para lograr la comunicación entre un sensor y un equipo que procese datos se usará la comunicación por puerto serie (el conector USB emula un puerto de este tipo). El componente que se encarga de estas operaciones es SerialActiveMessageC. Su nombre recuerda a ActiveMessageC: interfaces similares, pero en este caso los mensajes no se envían/reciben por radio. Esta similitud entre componentes es muy útil a la hora de programar, pues las interfaces que se manejan son prácticamente las mismas.

[37] *Esta línea es la misma que el protocolo de colección: no es necesario repetirla.*

4.12.1 SerialActiveMessageC

Contiene interfaces para el manejo de los mensajes transmitidos y recibidos por el puerto serie.

INTERFACES PROPORCIONADAS

- `SplitControl`. Activación o desactivación el funcionamiento del componente correspondiente (y de sus componentes asociados). Por ejemplo, mediantes los comandos `start` y `stop` se realizan las mencionadas acciones. A su vez, se tienen sendos eventos: `startDone` y `stopDone`. El primero avisa de que el componente ha sido iniciado y está listo para recibir más comandos; el segundo, que el componente ha sido detenido.
- `Packet`. Accede a la estructura básica de mensajes en TinyOS 2.x: `message_t`. El comando más usado es `getPayload`, que devuelve un puntero al área útil del paquete de datos.
- `AMSend`. Envía un mensaje (en este caso por puerto serie). Contiene el comando igual al anterior `getPayload`; `send`, que envía un determinado mensaje a una dirección concreta; o `cancel`, para cancelar una transmisión en proceso. El evento `sendDone` informa si el mensaje se ha podido transmitir (éxito) o si por el contrario ha habido algún error. Remarcar que se trata de una interfaz parametrizada: es necesario dar un identificador para la transmisión. Esto es útil si se envían distintos tipos de mensajes a través del mismo puerto: el receptor los diferencia con este parámetro.
- `Receive`. Recepción de mensajes. Su único evento es `receive` (la carga útil la indica con un puntero). Esta interfaz también está parametrizada. Tiene la utilidad complementaria a la anterior interfaz. Cada interfaz `Receive` envía el evento solo para los mensajes que tengan su mismo identificador.

4.13 Variables y constantes globales

Usadas en ambos tipos de nodos (tanto de datos como raíz):

- `radiopkt`. Variable del tipo `message_t`, es decir, *buffer* de mensajes de TinyOS 2.x. Como su nombre indica, se trata del paquete usado para enviar mensajes de datos vía radio desde un nodo de datos, y recibirlo en un nodo raíz.

Variables usadas únicamente por el nodo base:

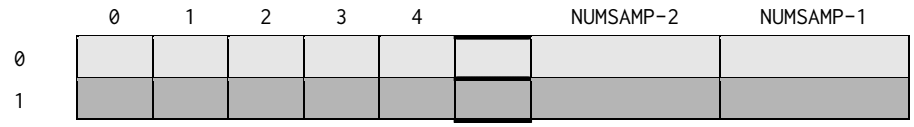
- `serialpkt`. Al igual que el caso anterior, variable del tipo `message_t`. En este caso, será el paquete que transporte información a través del puerto serie (físicamente USB) entre un ordenador y el nodo raíz.
- `serial_send_status`. Variable bandera para indicar el estado del envío por puerto serie.

Valor	Significado
0	Puerto serie libre (sin enviar)
1	Puerto serie ocupado (enviando mensajes)

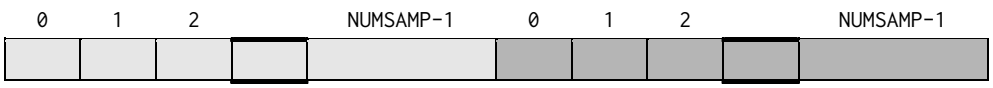
Tabla 4.5 Variable serial_send_status

El último bloque, son variables usadas únicamente por los nodos de datos.

- `raw`. *Array* o tabla bidimensional. Al declararlo como `uint16_t raw[MAXBUF][NUMSAMP]` se está indicando que se tienen `MAXBUF` filas y `NUMSAMP` columnas. Permite almacenar los datos leídos por el convertidor analógico-digital sin procesamiento alguno. La idea de declararlo como una matriz bidimensional es poder trabajar (procesar) una serie de datos mientras que por otro lado se están escribiendo nuevos. Por ejemplo, suponiendo `MAXBUF=2`, la matriz quedaría como sigue:



En memoria, esta matriz se almacenaría por filas, quedando así `NUMSAMP` muestras de la misma conversión, seguidas de otras `NUMSAMP` muestras:



- `sensor`. Vector para almacenar las conversiones de las medidas de sensores auxiliares (temperatura interna, tensión de alimentación...). Se toman 16 mues-

tras por cada sensor, con el fin de aplicar posteriormente un procesamiento de sobremuestreo. Las componentes totales serán por tanto `16*NUMSENS` (según el número de sensores que se deseen procesar).

- `bufesc` y `buflec`. Variables para indicar sobre qué *buffer* de recogida de datos se está trabajando. La tabla `raw` contiene varios *buffer* (como mínimo dos) con el fin de realizar varias tareas simultáneamente. `bufesc` indica sobre cuál se están escribiendo nuevas conversiones (convertidor A-D). Por su parte, `buflec` indica sobre qué *buffer* se pueden realizar cálculos o procesamientos. La idea que está detrás es poder estar realizando conversiones mientras se procesan los datos tomados anteriormente, disminuyendo así el tiempo entre dos conversiones consecutivas.
- `adc_status`. Bandera para indicar el estado del convertidor analógico digital.

Valor	Significado
0	Recurso libre. No se está realizando ninguna conversión
1	Recurso ocupado. Está convirtiendo medidas

Tabla 4.6 Variable adc_status

- `calcp_status`. Indicación del estado de la tarea de cálculo de valores medios, eficaces, potencia... (`CalcPotencia`).

Valor	Significado
0	Tarea libre (sin empezar o ya finalizada)
1	Ejecutando tarea (cálculos)

Tabla 4.7 Variable calcp_status

- `sens_status`. Indica el estado de la tarea de procesamiento de los sensores auxiliares (`ProcSensor`).

Valor	Significado
0	Tarea libre (sin empezar o ya finalizada)
1	Ejecutando tarea (cálculos)

Tabla 4.8 Variable sens_status

- `dft_status`. Variable bandera para indicar en qué estado se encuentra la tarea de cálculo de la DFT de las señales (`CalcDFT`). Esta tarea se ejecuta en varias partes, con el fin de no bloquear al resto de procesos del microcontrolador. Para conocer en todo momento cómo se encuentra dicha tarea se tienen varios valores de estado.

Valor	Significado
0	Tarea libre (sin empezar o ya finalizada)
1	Copiando datos a un vector interno de trabajo
2	Datos copiados correctamente (pueden sobrescribirse los datos originales)
3	Calculando DFT de la secuencia compleja
4	DFT de la secuencia calculada
5	Calculando DFT de la secuencia original
6	DFT secuencia original calculada
7	Calculando parámetros de armónicos
8	Parámetros armónicos calculados
9	Calculando THD

Tabla 4.9 Variable dft_status

- `send_status`. Variable bandera análoga a `serial_send_status`. En este caso se aplica al sistema de radio y el envío de paquetes.

Valor	Significado
0	Radio libre (sin enviar)
1	Radio ocupada (transmitiendo mensaje)

Tabla 4.10 Variable send_status

- `wr_memory`. Indica si es necesario escribir en la memoria flash (externa) nuevos datos (como límites de distorsión, umbral de temperatura...).

Valor	Significado
FALSE	No es necesario escribir en memoria
TRUE	Es necesario actualizar la memoria flash

Tabla 4.11 Variable wr_memory

- `vi_overflow`. Indicación de desbordamiento en las medidas de tensión e intensidad. Este desbordamiento puede producirse por el tiempo de conversión

inadecuado o bien por problemas de escritura en memoria. La variable se activa cuando se recibe alguno de estos eventos `OverflowVI.conversionTimeOverflow` y `OverflowVI.memOverflow` (respectivamente).

Valor	Significado
FALSE	No ha habido desbordamiento
TRUE	La conversión ha tenido problemas

Tabla 4.12 Variable vi_overflow

- `sens_overflow`. Variable análoga a la anterior, pero para las conversiones de los sensores auxiliares. En este caso, los eventos para activar esta bandera son `OverflowSENS.conversionTimeOverflow` y `OverflowSENS.memOverflow`

Valor	Significado
FALSE	No ha habido desbordamiento
TRUE	La conversión ha tenido problemas

Tabla 4.13 Variable sens_overflow

- `thd_umbral`. Vector de dos componentes, para guardar los valores máximos o umbrales para la THD. El primer valor (`thd_umbral[0]`) se usa para la distorsión de onda de tensión y el segundo (`thd_umbral[1]`) para intensidad.
- `temp_umbral`. Almacenamiento del valor máximo permitido para la temperatura interna del microcontrolador.
- `volt_umbral`. Umbral máximo para la tensión de alimentación del nodo.
- `configVI`. Se trata de una estructura constante del tipo `msp430adc12_channel_config_t`. Permite configurar correctamente el convertidor analógico-digital para las medidas de tensión e intensidad de la red (aquí solo se podía configurar el primero, tensión).
- `memctlVI`. Estructura en *array* de tipo `adc12memctl_t` para configurar el resto de canales del convertidor (en este caso solo uno más, el de intensidad).
- `configSENS`. Igual que el caso anterior, estructura constante del tipo `msp430adc12_channel_config_t`, pero ahora para los sensores auxiliares. Parámetros de configuración del primer canal (en este caso, temperatura interna).

- memctlSENS. Estructura para el resto de canales. Se tendrán NUMSENS-1 componentes.
- Las siguientes variables se utilizan para ir almacenando en forma de *array* (conforme se van calculando) los valores de tensiones medias (vmeddata), intensidades medias (imeddata), tensiones eficaces (vrmsdata), intensidades eficaces (irmsdata), potencias activas (pactdata), factores de potencia (fdpdata), distorsión armónica (thddata) y temperatura y tensión de alimentación (tempvoltdata). Cuando se tenga que enviar el paquete de datos, la información se tomará de estas variables.
- permdata. Se trata de una estructura de tipo storage_struct (declarada en el código). Contiene los campos que se quedarán almacenados en la memoria externa (no volátil). Entre ellas: version, que incluye la versión del firmware instalado; thd_threshold, contiene los umbrales máximos de distorsión de tensión e intensidad; temp_threshold, umbral para temperatura o volt_threshold, para la tensión del nodo.

4.14 Función IsRoot

Comprobación de si un nodo está configurado correctamente como nodo raíz o base. Como se explicó en 4.1, para que un nodo se considere base, su identificador TOS_NODE_ID debe ser divisible entre la constante ROOT_DIVIDER (fijada por defecto a 20).

Argumento	Tipo	Significado
(salida)	bool	TRUE cuando el identificador de nodo es divisible entre ROOT_DIVIDER. FALSE en caso contrario

Tabla 4.14 Argumentos función IsRoot

4.15 Función Arrancar

Inicia diferentes componentes que se usarán a lo largo del programa.

Argumento	Tipo	Significado
device (entrada)	uint8_t	Dispositivo del mote de desea iniciar. Se usan constantes simbólicas.
(salida)	error_t	Devuelve SUCCESS si no ha habido ningún problema. En caso contrario, se devuelve FAIL

Tabla 4.15 Argumentos función Arrancar

Están definidas algunas constantes simbólicas, con el fin de facilitar la compresión y programación del código. Por ejemplo, mediante

```
Arrancar( RADIO );
```

se inicia el componente `ActiveMessageC`, que maneja los mensajes vía radio.

Si un arranque no tuvo éxito, la propia función lo intentará un número determinado de veces, definido en `MAXATTEMPT`. Si pasados estos intentos, el módulo no se inicia, la función devolverá el código de error `FAIL`.

Algunos valores solo se tienen en cuenta en nodos de datos (como `ADCVI`, para iniciar el convertidor A-D para tensión e intensidad, que no tendría sentido en un nodo base).

4.16 Función InicioMote

Inicialización de algunas variables de la aplicación. Como *buffer* de datos, se configura el convertidor analógico-digital y las salidas digitales de propósito general. Esta función no tiene argumentos (ni de entrada ni de salida). Las variables que utiliza se describen en Tabla 4.16.

Nombre	Tipo	Inicial	Funcionalidad
i	uint8_t var.	0	Contador para bucle FOR
j	uint16_t var.	0	Contador para bucle FOR

Tabla 4.16 Variables locales función InicioMote

4.17 Función increase

Permite incrementar una variable de tipo entero de $8\,\mathrm{bit}$, rango $[0,255]$. A diferencia de la operación de incremento ++, cuando la variable alcanza su valor máximo, permanece en éste y no se modifica (saturación).

Argumento	Tipo	Significado
x (entrada)	uint8_t *	Puntero a la variable entera para aumentar en una unidad.

Tabla 4.17 Argumentos función increase

4.18 Función decrease

Decrementa una variable de tipo entero de $8\,\mathrm{bit}$, rango $[0,255]$. A diferencia de la operación de decremento --, cuando la variable alcanza su valor mínimo, permanece en éste y no se modifica.

Argumento	Tipo	Significado
x (entrada)	uint8_t *	Puntero a la variable entera para disminuir en una unidad.

Tabla 4.18 Argumentos función decrease

4.19 Tarea ProcSensor

Tarea para el procesamiento de los sensores auxiliares: temperatura interna del microcontrolador, tensión de alimentación y otros posibles para añadir. Para más información, puede consultarse el punto 4.8.

Nombre	Tipo	Inicial	Funcionalidad
temp	uint32_t var.	0	Temperatura interna del microcontrolador
volt	uint32_t var.	0	Tensión de alimentación del dispositivo
sen1	uint32_t var.	0	Medida para sensor adicional 1
sen2	uint32_t var.	0	Medida para sensor adicional 2
i	uint8_t var.	0	Control de bucle FOR

Tabla 4.19 Variables y/o constantes locales tarea ProcSensor

Según la configuración elegida para la aplicación, se realizan un total de 64 conversiones de los sensores (16 para cada uno). El bucle que recorre estas medidas se encarga de ir almacenándolas en cada una de las cuatro variables, teniendo en cuenta que las muestras se toman entrelazadas, esto es:

```
temp-volt-sen1-sen2-temp-volt-sen1-sen2-temp-...-sen1-sen2
```

En cada iteración se va acumulando el valor recogido del ADC12 (en el rango $\left[0, 4095\right]$).

Una vez hecho esto, se procede a aplicar el procesado de *sobremuestreo* y a corregir los valores para obtener unos resultados más legibles.

Por ejemplo, para el caso de la temperatura interna se aplicará la expresión:

$$T_C\left(^{\circ}\mathrm{C}{\cdot}10\right) = \frac{100000{\cdot}N_{CAD} - 646027200}{232596} \tag{4.10}$$

Los factores 100000, 646027200 y 232596 están definidos en el archivo de cabecera `Sensores.h`.

Una vez hecho esto, se guardan estos datos en la variable `tempvoltdata`, que se usará para transmitir esta información vía radio.

También se comprueban los umbrales para temperatura y tensión de alimentación. En el sistema no está programada ninguna acción, si bien puede configurarse para que active una salida digital, envíe un mensaje de aviso...

En este punto ya se tendrían procesadas las medidas. A continuación se intenta liberar el recurso `ResourceSENS` (convertidor A-D para los sensores auxiliares). En caso positivo, se llama a la tarea de envío de datos por radio `EnvioDatos`.

Simultáneamente, se comprueba si existen nuevos datos para ser almacenados en memoria, a través de la bandera `wr_memory`. Si es así, se debe llamar a `ConfigSto.write`, para que comience la escritura en la memoria flash externa.

Estos pasos se recogen en el diagrama de flujo de la Figura 4.4.

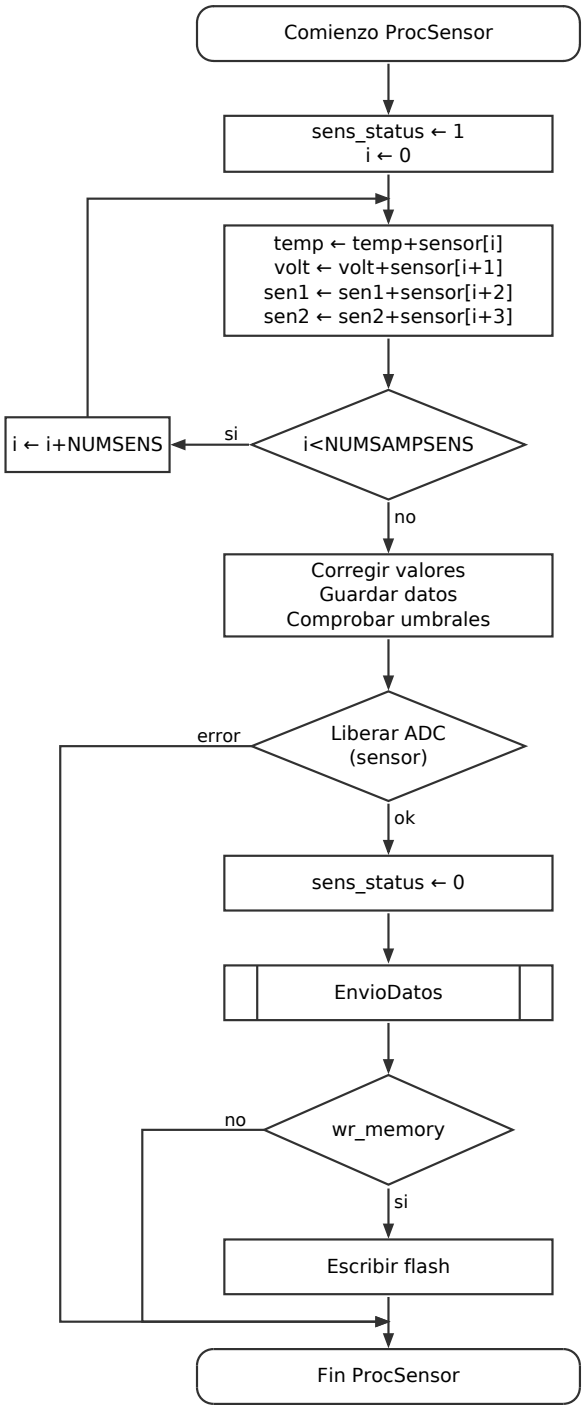


Figura 4.4 Diagrama de flujo tarea ProcSensor

4.20 Tarea ActValores

Se encarga de recibir valores procedentes del nodo raíz, procesarlos y actuar según éstos.

Nombre	Tipo	Inicial	Funcionalidad
newdissmsg	Puntero a estructura DissMsg const.	Recibido por Dissemination	Datos del paquete que se envía desde el nodo base a un nodo de datos (diseminación)

Tabla 4.20 Variables y/o constantes locales tarea ActValores

La primera comprobación es el cuál es el destinatario del mensaje. En la estructura de datos `DissMsg`, concretamente el campo `nodeid`, indica a qué nodo va dirigido el mensaje. Existen dos casos para aceptar el mensaje: que el destinatario sea el nodo en cuestión o que sea un mensaje de difusión (destinado a toda la red).

El siguiente campo que se chequea es `function`. Indica qué se desea hacer en el nodo remoto. Según su valor:

- `1`: activación/desactivación de salidas digitales. En este caso se trabaja con la salida digital cuya interfaz se denomina en el código `CargaA`. Si el valor especificado en el campo `valueA` es igual a `0`, la salida queda desactivada (nivel bajo de tensión). En caso de que sea mayor que `0`, se activa (nivel alto). En este caso, `valueB` no tiene ningún efecto programado.
- `2`: actualización de los umbrales máximos de distorsión armónica. Primeramente se comprueba que los valores recibidos son diferentes a los que ya existen (si no fuera así, no sería necesaria la actualización). Se toman los nuevos datos y se guardan en el vector `thd_umbral`. También se prepara la estructura `permdata` que permite almacenar estos datos en la memoria no volátil. Por último, activa la bandera `wr_memory`, que indica que hay nuevos datos para guardar en memoria flash.
- `3`: actualización de los umbrales de temperatura y tensión. Al igual que antes, se comprueba si alguno de ellos ha cambiado respecto a los valores actuales. Si es así, se procede a recoger estos datos y guardarlos en `temp_umbral`

y `volt_umbral` respectivamente. Después, se guardan en la estructura `permdata` y se activa la bandera de guardado en memoria externa `wr_memory`.

El resto de valores de `function` no tendrán ningún efecto. Puede ampliarse la funcionalidad simplemente añadiendo nuevos valores posibles al campo `function` (hasta 255).

En la Figura 4.5 se muestra el diagrama de flujo de esta tarea.

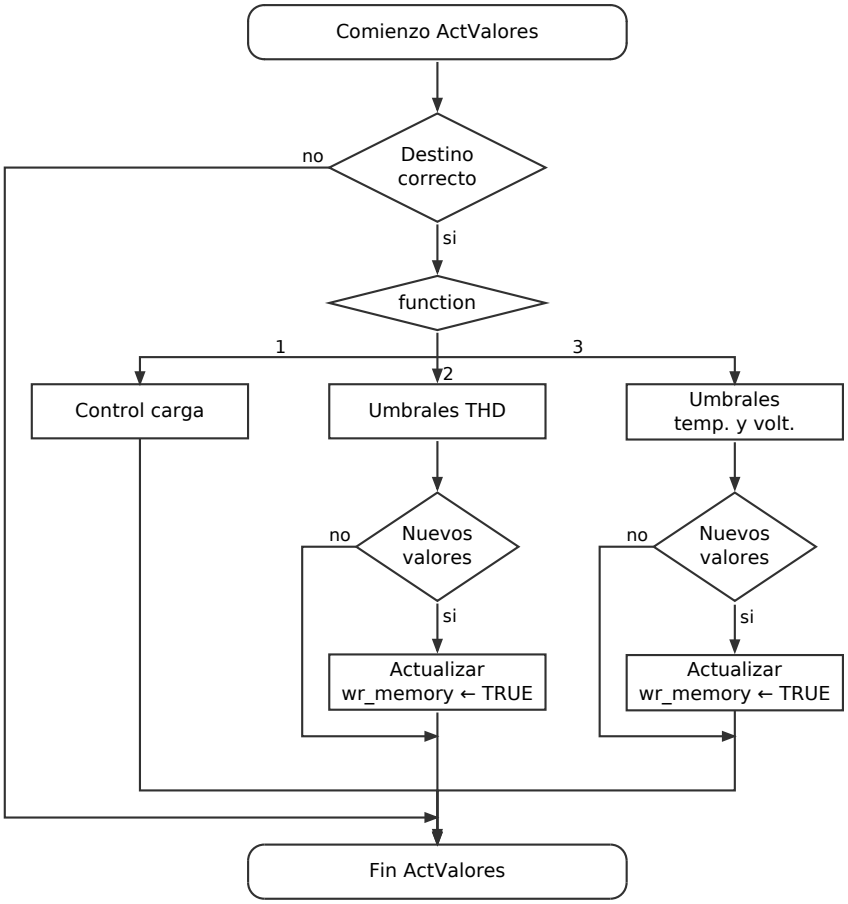


Figura 4.5 Diagrama de flujo tarea ActValores

4.21 Tarea EnvioDatos

Prepara el paquete de datos a transmitir desde un nodo de datos al nodo raíz.

Nombre	Tipo	Inicial	Funcionalidad
i	uint8_t var.	0	Control de bucle FOR
potpkt	Puntero a estructura PotenciaMsg (paquete datos)	NULL	Acceso a los campos internos del paquete de datos (estructura)

Tabla 4.21 Variables y/o constantes locales tarea EnvioDatos

Para poder enviar un mensaje, se exigen tres condiciones: las tareas de cálculo de potencia y DFT han de estar terminadas (liberadas); y el transmisor radio no puede estar ocupado. Esto se controla a través de las variables `calcp_status`, `dft_status` y `send_status` respectivamente.

En caso de que estas condiciones no se cumplan (simultáneamente) se vuelve a programar la ejecución de la tarea, es decir, se pospone el envío.

Si por el contrario se cumple, se accede al bloque de datos de un paquete (`message_t`) realizando la llamada a `Send.getPayload` (como se explicó, es más seguro trabajar con interfaces que acceder directamente a los campos del mensaje).

El siguiente paso es preparar los datos a enviar. El primero es el identificador de nodo (variable `TOS_NODE_ID`). Después, mediante dos bucles FOR se pasan los valores locales (tensión e intensidad medias, valores eficaces, potencia activa...) al paquete de datos, así como las tasas de distorsión armónica, temperatura interna y tensión de alimentación.

Con `Send.send` se realiza la petición para enviar el mensaje por radio. Si ha resultado satisfactoria, se activa la bandera `send_status`, indicando radio ocupada transmitiendo. Habrá que esperar al evento `Send.sendDone`, que realmente informará si se ha enviado o no (y si ha habido algún problema durante el envío).

Diagrama de flujo de la tarea `EnvioDatos` en Figura 4.6.

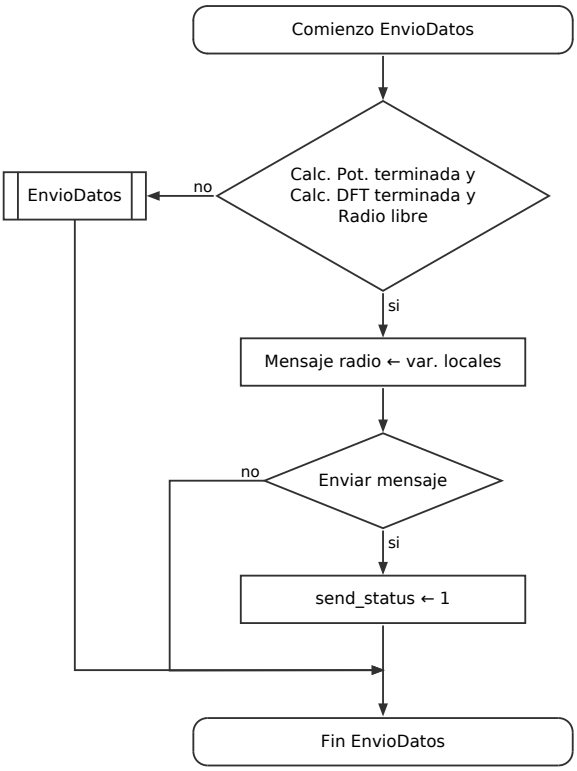


Figura 4.6 Diagrama de flujo tarea EnvioDatos

4.22 Tarea Conversion

Organiza las conversiones (tanto de tensión-intensidad como de los sensores auxiliares). También se encarga de actualizar los indicadores de *buffer* de lectura/escritura y de lanzar las tareas de procesamiento (`CalcPotencia`).

Nombre	Tipo	Inicial	Funcionalidad
num_conv	static uint8_t var.	0	Cuenta el número de conversiones realizadas (sin problemas).

Tabla 4.22 Variables y/o constantes locales tarea Conversion

Al comenzar la tarea, lo primero que se comprueba es la variable `num_conv`. Si es igual a `MAXCONV` (constante simbólica definida previamente), se pasarán a leer los sensores auxiliares y posteriormente enviar el mensaje radio. Se vuelve a 0 esta variable, después se libera el recurso `ResourceVI.release`, y una vez hecho esto, se

pide acceder al convertidor A-D para los sensores auxiliares: `ResourceSENS.request` (esperar al evento `ResourceSENS.granted`).

Si `num_conv` todavía no ha llegado al número máximo, se pasa a comprobar las siguientes condiciones: que no haya habido desbordamiento en la última conversión, la tarea de cálculo de potencia haya finalizado, y la tarea de cálculo DFT ya haya tomado los datos necesarios. Si se cumplen todas ellas, se actualiza la variable `buflec`, se llama a la tarea de cálculo de potencia y se prepara el nuevo valor de `bufesc` (para la próxima conversión). Si no se cumpliera alguna, no habría llamada a la tarea ni actualización de las variables anteriores.

Después, se llama a `MultiChannelVI.configure` para preparar el buffer de memoria para la siguiente conversión de tensión e intensidad. Si ha tenido éxito, se desactiva la bandera de desbordamiento `vi_overflow` y se hace `MultiChannelVI.getData`, para la recogida de datos. De nuevo, si ha tenido éxito, se activa la bandera `adc_status`, indicando que el convertidor está ocupado. Si en cualquiera de estas llamadas se produjera algún error, se vuelve a preparar la tarea de conversión. El diagrama de la Figura 4.7 muestra este proceso.

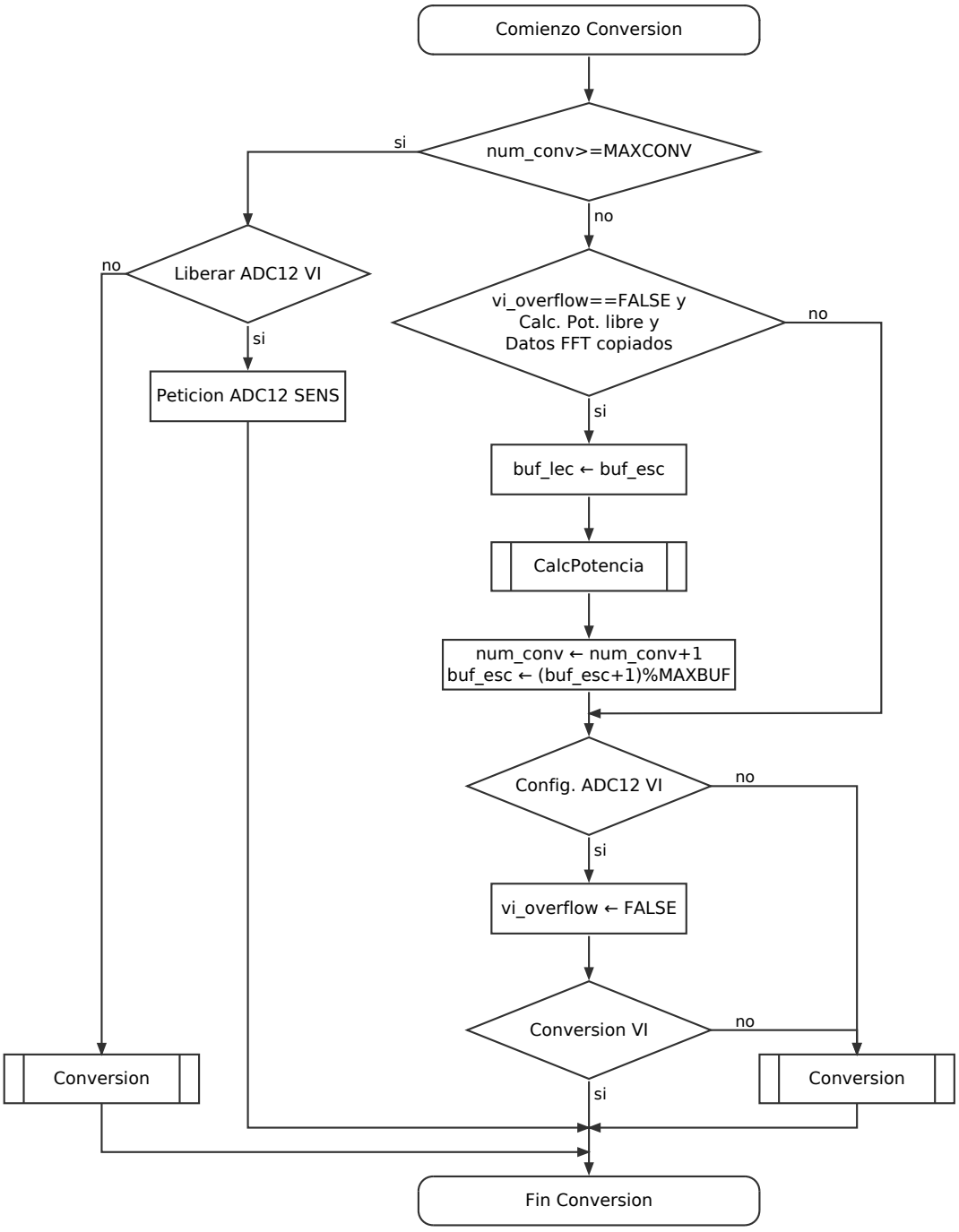


Figura 4.7 Diagrama de flujo tarea Conversion

4.23 Tarea CalcPotencia

Se encarga de calcular los valores más significativos asociados a las ondas de tensión e intensidad: valor medio, eficaz, potencias, energía...

Nombre	Tipo	Inicial	Funcionalidad
vmed	uint32_t var.	0	Valor medio (D.C.) de tensión
imed	uint32_t var.	0	Valor medio (D.C.) de intensidad
vrms	uint64_t var.	0	Valor eficaz (RMS) de tensión al cuadrado
irms	uint64_t var.	0	Valor eficaz (RMS) de intensidad al cuadrado
pact	uint64_t var.	0	Potencia activa (P)
prea	uint64_t var.	0	Potencia reactiva (Q) al cuadrado
ener	uint64_t var.	0	Energía eléctrica (E)
fdp	uint16_t var.	0	Factor de potencia en % al cuadrado
i	uint8_t var.	0	Control de bucle FOR
contador	static uint8_t var.	0	Controla los resultados de las conversiones que se almacenan en el paquete de datos radio

Tabla 4.23 Variables y/o constantes locales tarea CalcPotencia

El primer paso es activar la bandera `calcp_status=1`, indicando así que la tarea ha comenzado a ejecutarse.

A continuación se tiene un bucle FOR encargado de realizar las operaciones de integración de las señales. En todos los casos se aplica la integración mediante la regla de Simpson. La expresión usada viene dada en (4.11).

$$\int_a^b x(t)\,dt \approx \frac{h}{3}\left[x(a) + 4x(m) + x(b)\right] \tag{4.11}$$

En particular, el tamaño de paso $h \approx 625\,\text{us}$ corresponde a la frecuencia de muestreo $F_S \approx 1600\,\text{Hz}$ [38]. El buffer donde se encuentran almacenadas las medidas tiene una longitud total de $128 + 128 = 256$ muestras entre tensión e intensidad.

[38] *El hecho de escribir $\approx$ significa que la precisión del sensor no permite establecer los valores teóricos exactamente.*

- Tensión e intensidad medias. El bucle comienza en el segundo punto, es decir $x(m)$, multiplicándose así por cuatro. Como el siguiente punto $x(b)$ coincide con el inicial del próximo intervalo $x(a)$, basta multiplicarlo por dos. El bucle finaliza al llegar a la penúltima muestra (con índice 252 para tensión y 253 para intensidad). La fórmula que se aplica es:

$$X_m = \frac{1}{T_0}\int_{T_0} x(t)\,dt \tag{4.12}$$

- Valores eficaces de tensión e intensidad. El proceso es igual al bucle anterior, solo que ahora se multiplica cada muestra por sí misma[39]. De hecho, la expresión usada es muy parecida a (4.12):

$$X_{RMS}^2 = \frac{1}{T_0}\int_{T_0} x^2(t)\,dt \tag{4.13}$$

- Potencia activa[40]. Se utilizará el Principio de tiempo de muestreo reducido, que se explica a continuación.

4.23.1 Potencia en señales discretas. Principio de tiempo de muestreo reducido

Como se ha visto anteriormente, para calcular valores de potencia, es necesario hallar la potencia instantánea con el producto:

$$p(t) = v(t)\cdot i(t) \tag{4.14}$$

O si se expresa en forma de señal discretizada:

$$p[n] = v[n]\cdot i[n] \tag{4.15}$$

---

[39] Al realizar este proceso, el resultado obtenido será el valor eficaz al cuadrado (RMS^2). El uso de la función matemática de raíz cuadrada en un sensor de estas características no sería lo ideal: necesitará usar decimales y las operaciones implicadas consumirían recursos del microcontrolador. Simplemente se transmite el valor al cuadrado y el equipo de monitorización se encargará (si es necesario) de hallar su raíz cuadrada.

[40] Capacidad que tiene el dispositivo eléctrico para realizar un trabajo, transformando la energía eléctrica a otra forma (mecánica, lumínica, térmica…). Se mide en Watios (W).

Sin embargo, el convertidor analógico-digital del mote tiene una limitación: no es posible tomar dos medidas simultáneas del canal de tensión y del canal de intensidad. Dicho de otro modo, las conversiones hay que realizarlas alternativamente en cada canal. Es necesario usar algún mecanismo que permita aproximarse al valor real de la potencia instantánea, y éste es el Principio de tiempo de muestreo reducido[41].

Se supone que cada señal (tensión e intensidad) se muestrea cada cierto tiempo denominado T. La idea del método es desfasar la lectura de las señales un tiempo $T/2$. En la Figura 4.8 se ilustra este sistema.

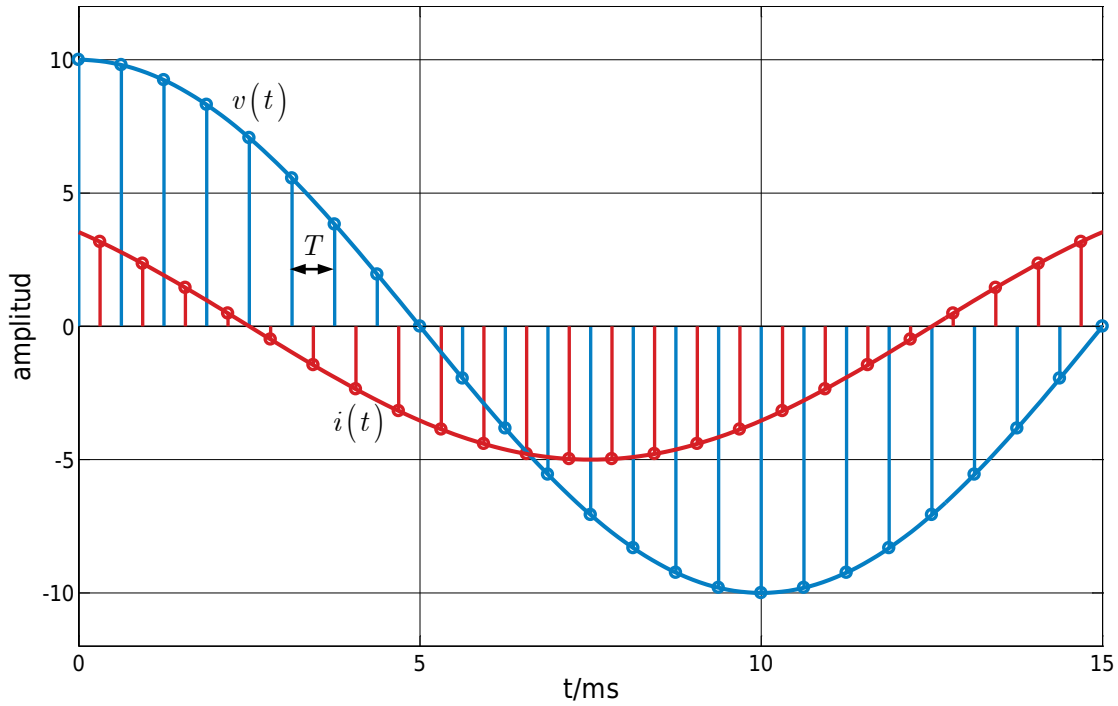


Figura 4.8 Principio de tiempo de muestreo reducido

Se muestra la señal de tensión (azul) y de intensidad (en rojo). El tiempo entre dos medidas consecutivas (de tensión o intensidad) es el tiempo de muestreo T. Entre dos medidas de tensión, se realiza una de intensidad, justo a la mitad del intervalo, en $T/2$.

Como la potencia instantánea necesita que las dos medidas se realicen en el mismo instante de tiempo, se realiza la siguiente aproximación:

[41] *Este algoritmo es propiedad intelectual de Texas Instruments. Según se detalla en la nota de aplicación correspondiente (Texas Instruments, 2000), puede ser usado únicamente en microcontroladores de Texas Instruments.*

$$p[n] = i[n] \cdot \frac{v[n] + v[n+1]}{2} \tag{4.16}$$

Esto es, la potencia es un instante dado n, se calcula multiplicando el valor de la intensidad $i[n]$ por la media aritmética de la tensión anterior y posterior. Este procedimiento es válido siempre y cuando la señal no varíe demasiado entre muestra y muestra. En la Figura 4.9 muestra con más detalle el procedimiento (los intervalos temporales son los que se utilizan en la aplicación).

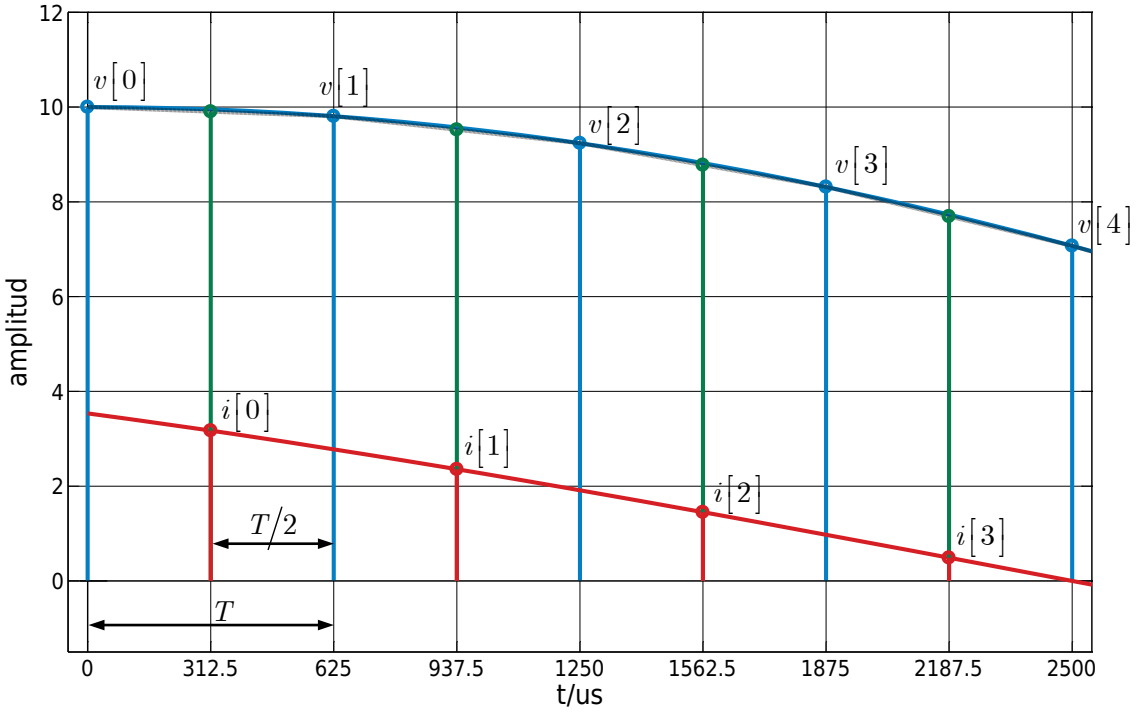


Figura 4.9 Principio de tiempo de muestreo reducido (detalle)

El primer valor de potencia instantánea se calculará como $p[0] = i[0] \cdot (v[0] + v[1])/2$ *. El siguiente vendrá dado por* $p[1] = i[1] \cdot (v[1] + v[2])/2$ *y así sucesivamente. Como se puede comprobar, existe un error entre el valor medio (de color verde)* $(v[n] + v[n+1])/2$ *y el valor real que tiene la función (que no puede medirse).*

Hay que tener en cuenta que este sistema de medida tiene un error inherente, debido al desfase que se produce entre medidas. Puede demostrarse que el valor de este error (en porcentaje) viene dado por

$$e(\%) = \left[\cos\left(2\pi \frac{T}{2} F_0\right) - 1\right] \cdot 100 = \left[\cos\left(\pi \frac{F_0}{F_S}\right) - 1\right] \cdot 100 \tag{4.17}$$

Donde T es el tiempo de muestreo y $F_S = 1/T$ la frecuencia de muestreo. F_0 es la frecuencia fundamental de la sinusoide, en el caso que nos ocupa será

$F_0 = 50\,\text{Hz}$. Es interesante remarcar el hecho de que el error no depende del desfase ϕ que pueda existir entre las señales de tensión e intensidad (según la naturaleza inductiva/capacitiva del circuito).

* * *

Retomando la tarea `CalcPotencia`, para implementar este principio se utiliza de nuevo la integración numérica de Simpson, dejando las primeras y últimas muestras fuera del bucle FOR.

Faltan por tratar algunas muestras de las señales. Debido a que el periodo de integración para las señales ha de ser de un período fundamental ($20\,\text{ms}$) o un múltiplo de éste, se toman $4{\cdot}20\,\text{ms} = 80\,\text{ms}$. Esto obliga a que se deba trabajar con $128 + 1$ muestras (con el fin de completar así el periodo). A la hora de manejar datos en memoria, es recomendable usar, siempre que sea posible, tamaños potencia de dos, de ahí el *buffer* de 256 componentes. Para solventar este problema, se usa la siguiente aproximación: se asume que la última muestra del período de integración es similar a la primera (la señal no ha cambiado mucho en los $80\,\text{ms}$). Esto es lo que se aplica para todos los valores a calcular.

Una vez hecho esto, es necesario ajustar los valores. Para empezar, la regla de Simpson incluye el factor $h/3$ en su expresión. También hay que dividir los resultados entre el tiempo de integración. Por último, se multiplica el resultado por diez para poder aplicar un redondeo posterior. En resumen (se utilizan los valores exactos del mote):

$$\frac{h}{3}\cdot\frac{1}{4T}\cdot 10 = \frac{328/2^{19}}{3}\cdot\frac{1}{4\cdot\left(328/2^{14}\right)}\cdot 10 = \frac{10{\cdot}2^{14}}{12{\cdot}2^{19}} = \frac{5}{192} \qquad (4.18)$$

En el caso de la potencia activa, al haber usado el Principio de tiempo de muestreo reducido, hay que dividir además por dos: factor $5/384$.

Después, se procede a usar la macro de redondeo de cantidades (sin usar variables decimales).

La energía eléctrica se obtiene multiplicando la potencia activa por el tiempo de integración (aprox. $80\,\mathrm{ms}$) y por diez (redondeo):

$$4T{\cdot}10 = 4{\cdot}\left(328\Big/2^{14}\right){\cdot}10 = \frac{205}{256} \tag{4.19}$$

Es importante destacar que este cálculo hay que hacerlo después de tener el valor de potencia activa correcto.

La potencia reactiva[42] se calcula usando su definición (4.20).

$$S = \sqrt{P^2 + Q^2} \rightarrow Q^2 = S^2 - P^2 \tag{4.20}$$

Donde $S^2 = U_{RMS}^2{\cdot}I_{RMS}^2$ [43], ya calculados anteriormente. Para el factor de potencia[44], también se utiliza la definición de (4.21).

$$f.d.p. = \frac{P}{S} \rightarrow f.d.p.^2 = \frac{P^2}{S^2} \rightarrow f.d.p.\left(\%\right)^2 = 100^2\,\frac{P^2}{S^2} \tag{4.21}$$

El f.d.p. se da en tanto por ciento (al cuadrado) para no perder demasiada precisión (no se trabaja con decimales)[45].

El siguiente bloque de la tarea consiste en ir almacenando los distintos resultados obtenidos en las variables (globales) correspondientes: `vmeddata`, `imeddata`, `vrmsdata`, etcétera. Por cada bloque de conversiones (definido en la constante simbólica `MAXCONV`) se realizan cuatro guardados de datos. Es decir, si `MAXCONV` se establece a cuatro, se enviarán por radio todas las conversiones y cálculos realizados, si bien el envío de mensajes será excesivo (muchos datos

---

[42] *Aparece cuando existen inductores o condensadores en el circuito. Se produce un almacenamiento de energía eléctrica en forma de campo magnético o eléctrico. Se mide en Voltios-Amperios reactivos (VAR).*

[43] *S representa la potencia aparente, también denominada compleja o total. Es la suma vectorial o geométrica de las potencias activa y reactiva. Se mide en Voltios-Amperios (VA).*

[44] *Mide la componente de potencia reactiva que tiene una instalación:* $f.d.p. = \cos\phi = P\big/S$ *. Interesará que esté lo más cercano a* 1 *.*

[45] *Al igual que ocurría con los valores eficaces, la potencia reactiva se calcula al cuadrado. Así ocurre también con el factor de potencia. Esto permite el ahorro de recursos (y de consumo eléctrico) en el mote.*

cada poco tiempo). Por el contrario, si se aumenta `MAXCONV` no se enviarán algunas conversiones, pero el tráfico de mensajes será menor.

Después, se actualiza la variable `contador`. Además, se comprueba si la tarea `CalcDFT` (algoritmo FFT) está libre i.e. no se está ejecutando. Si es así, se llama a dicha tarea para trate los nuevos datos[46].

Para finalizar, se baja la bandera `calcp_status`, tomando el valor cero.

El proceso completo se recoge en la Figura 4.10.

[46] *Puede ocurrir (de hecho ocurre) que la tarea de cálculos armónicos de las señales tarde más tiempo que el que existe entre dos llamadas de la tarea de cálculo de potencia. Si no ha finalizado, se esperará hasta la siguiente vez para intentarlo (el cálculo DFT no es excesivamente prioritario).*

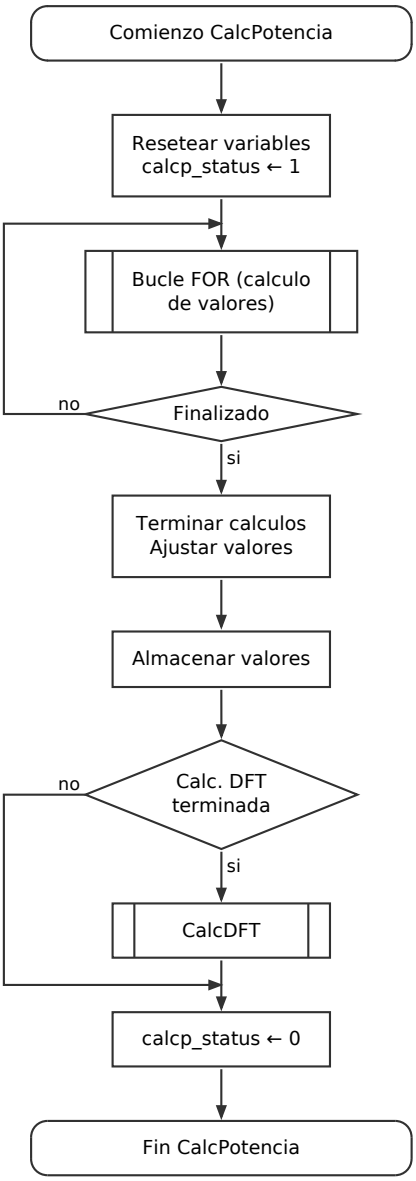


Figura 4.10 Diagrama de flujo tarea CalcPotencia

4.24 Tarea CalcDFT

4.24.1 Armónicos a estudiar

Los armónicos más frecuentes en las redes de distribución son los impares. Las amplitudes de los armónicos suelen disminuir a medida que aumenta la frecuencia. Por encima del armónico de rango 50º, los armónicos son insignificantes y las me-

diciones no serán significativas. Se obtienen mediciones suficientemente precisas midiendo los armónicos hasta el rango 30º.

Las instalaciones supervisan los armónicos de rangos 3º, 5º, 7º, 11º y 13º. Por lo general, es suficiente mejorar los niveles armónicos de los rangos más inferiores (hasta 13º). Una optimización más completa tendría en cuenta los armónicos hasta el rango 25º (Schneider Electric España, 2008).

En la Figura 4.11 se muestran los niveles máximos permisibles de los armónicos.

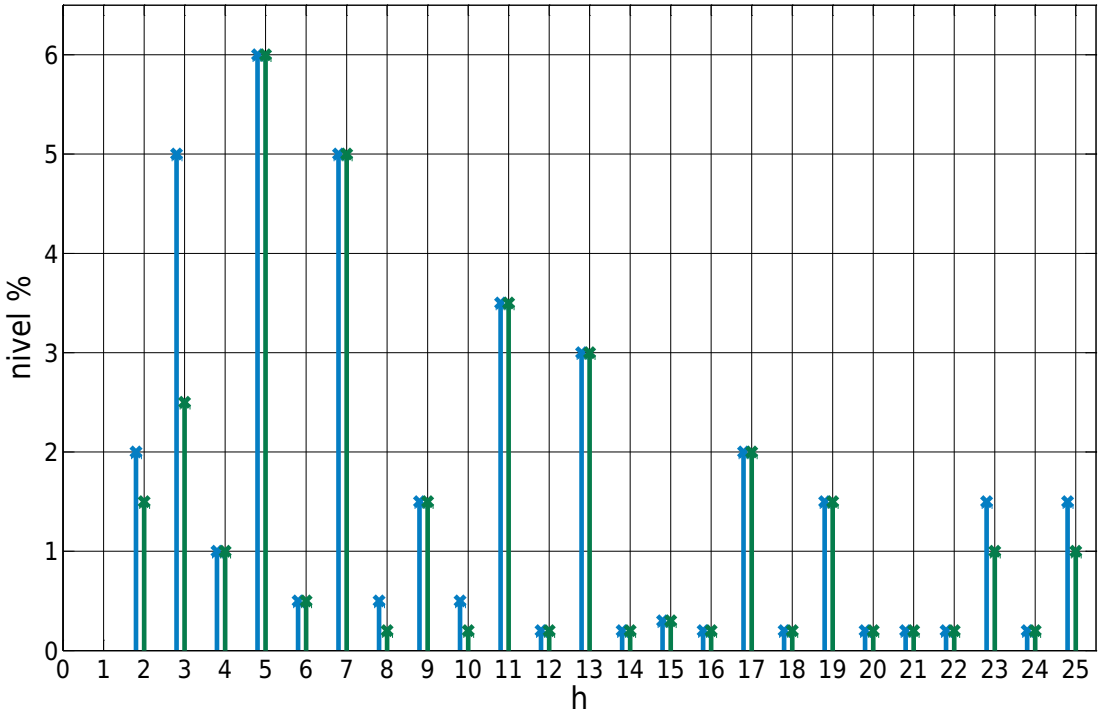


Figura 4.11 Niveles de armónicos máximos permisibles

El armónico fundamental es el de orden $h = 1$. Se representan dos clases de instalación: Baja Tensión (azul) y Media Tensión (en verde).

4.24.2 Planteamiento del problema

Para el estudio de los armónicos de red, es necesario tomar un número significativo de ellos, de forma que no se pierda demasiada información pero tampoco se exija gran capacidad de procesamiento digital.

Antes de continuar, es recomendable la lectura del Anexo A para una mejor comprensión de este apartado.

En el caso que nos ocupa, se ha tomado hasta el armónico 15º, o lo que es lo mismo, hasta frecuencia $f_{15} = 15 \cdot 50 = 750\,\text{Hz}$ [47]. De acuerdo al Teorema de muestreo de Nyquist-Shannon, el ancho de banda de la señal será $B = 750\,\text{Hz}$. La frecuencia de muestreo mínima será $F_{S_{\min}} = 1500\,\text{Hz}$.

La secuencia que se tome del proceso de muestreo se deberá enventanar obligatoriamente (no puede tener longitud infinita), lo que implica elegir una ventana adecuada.

Por otra parte, el procesamiento digital se debe hacer a través de la transformada discreta de Fourier (DFT), lo que implicaba muestras en el dominio de la frecuencia. Dicho de otro modo, el espectro resultante de enventanar la secuencia $\bar{V}\left(F/F_S\right)$ hay que muestrearlo, concretamente cada $F = \frac{F_S}{N}k \quad k = 0,1,2...N-1$. El valor N puede ser cualquier natural, sin embargo, debido al algoritmo empleado para su cálculo, conviene que sea una potencia de 2: de la forma $N = 2^R$.

La idea es que al muestrear el espectro, haya muestras que se encuentren en las frecuencias de los armónicos $F = 50,100,150...\,\text{Hz}$, con el fin de tomar su valor con la mayor exactitud posible. Con una frecuencia $F_S = 1500\,\text{Hz}$ y N potencia de 2 no es posible hacer coincidir las muestras con dichos armónicos. Realizando diversas pruebas, se llega a que con frecuencia $F_S = 1600\,\text{Hz}$ (mayor que la mínima admisible) sí es posible conseguir muestras a frecuencias de nuestro interés. Por ejemplo, con $N = 32$ se tendrían precisamente $F = 50,100,150...\,\text{Hz}$.

El siguiente punto a tratar es la periodicidad que existe en la DFT. Un muestreo en frecuencia implica una periodicidad en la secuencia temporal. Esto puede calcularse usando la relación

$$T_0 = NT = \frac{N}{F_S} \tag{4.22}$$

El periodo de repetición no es más que el número de muestras de la DFT multiplicado por el tiempo de muestreo entre ellas. Para los valores ya calculados resul-

[47] *Puede consultarse la introducción a los armónicos de red (1.3).*

ta $T_0 = 32/1600\,\text{Hz} = 20\,\text{ms}$. Esto es, la secuencia que se toma para realizar la DFT abarca exactamente un periodo de la señal analógica a $50\,\text{Hz}$[48]. De hecho, la condición para que una señal discreta (muestreada) tenga periodicidad es que el cociente F_0/F_S sea fraccionario, i.e. pueda expresarse como cociente de dos números naturales: $50/1600 = 1/32$[49].

En principio, con $N = 32$ se cumplen las especificaciones actuales del problema. A continuación, hay que elegir un modo de enventanado adecuado. Debido a que los armónicos de la señal tendrán una amplitud muy pequeña comparada con el principal, es fundamental que la ventana presente la mayor atenuación lobular posible. Tal y como se describe en A.2.2 (Anexo A), a mayor atenuación, también aumenta la resolución espectral, lo que puede hacer que los picos de los armónicos se confundan. La separación mínima que tendrán los armónicos será de $\Delta F = 50\,\text{Hz}$.

Ventana	Atenuación	Resolución N=32	Resolución N=64	Resolución N=128
Rectangular	13	50	25	12.5
Triangular/Bartlett	27	100	50	25
Hanning	31	100	50	25
Hamming	41	100	50	25
Blackman	57	150	75	37.5
Blackman-Harris	92	200	100	50

Tabla 4.24 Características de las ventanas para secuencias

Para este trabajo interesa una ventana con la mayor atenuación lobular posible, teniendo en cuenta la resolución que presenta (se supone una frecuencia de $1600\,\text{Hz}$).

En la Tabla 4.24 no se han tomado más de $N = 128$, pues la capacidad de procesamiento del sensor es limitada. Para escoger la ventana, se comienza desde abajo. La ventana Blackman-Harris con $N = 128$ proporciona una resolución $\Delta F = 50\,\text{Hz}$. Sin embargo, no es recomendable ajustar tanto las frecuencias. Si por algún motivo la frecuencia fundamental de la red quedara por debajo de los

[48] *El hecho de que el procesamiento abarque periodo/s completo/s de la señal es de gran utilidad para los cálculos de potencia, energía… (tarea CalcPotencia).*

[49] *La fracción $1/32$ significa que existen 32 muestras por cada periodo de la señal.*

$F_0 = 50\,\mathrm{Hz}$, en el espectro quedarían solapados los armónicos y no habría forma de resolverlos.

La siguiente ventana es Blackman (genérica). Con $N = 128$ se tiene una resolución $\Delta F = 37.5\,\mathrm{Hz}$. Esta ventana sí sería válida para el problema, si bien se ha perdido atenuación respecto de la anterior: $35\,\mathrm{dB}$.

La ventana resultante particularizada para estos valores, se muestra en la Figura 4.12 (dominio temporal) y en la Figura 4.13 (espectro en frecuencia).

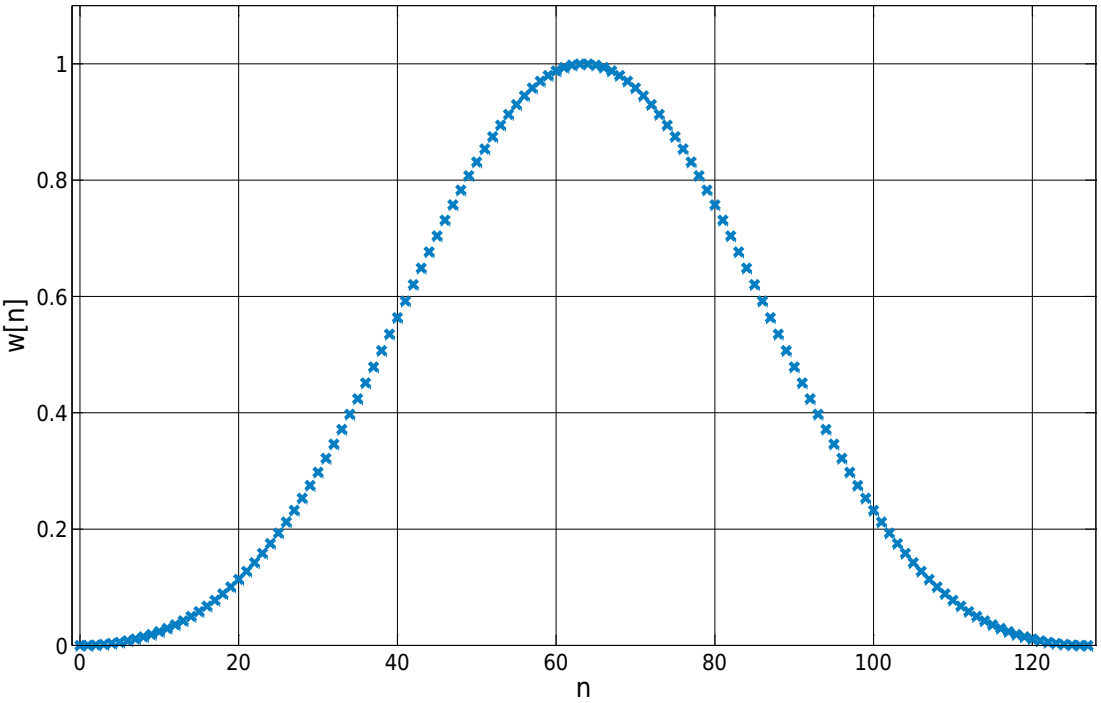


Figura 4.12 Ventana Blackman usada

Representación de la ventana que se va a usar para el control de armónicos de red. $M + 1 = 128 = 2^7$ *muestras.*

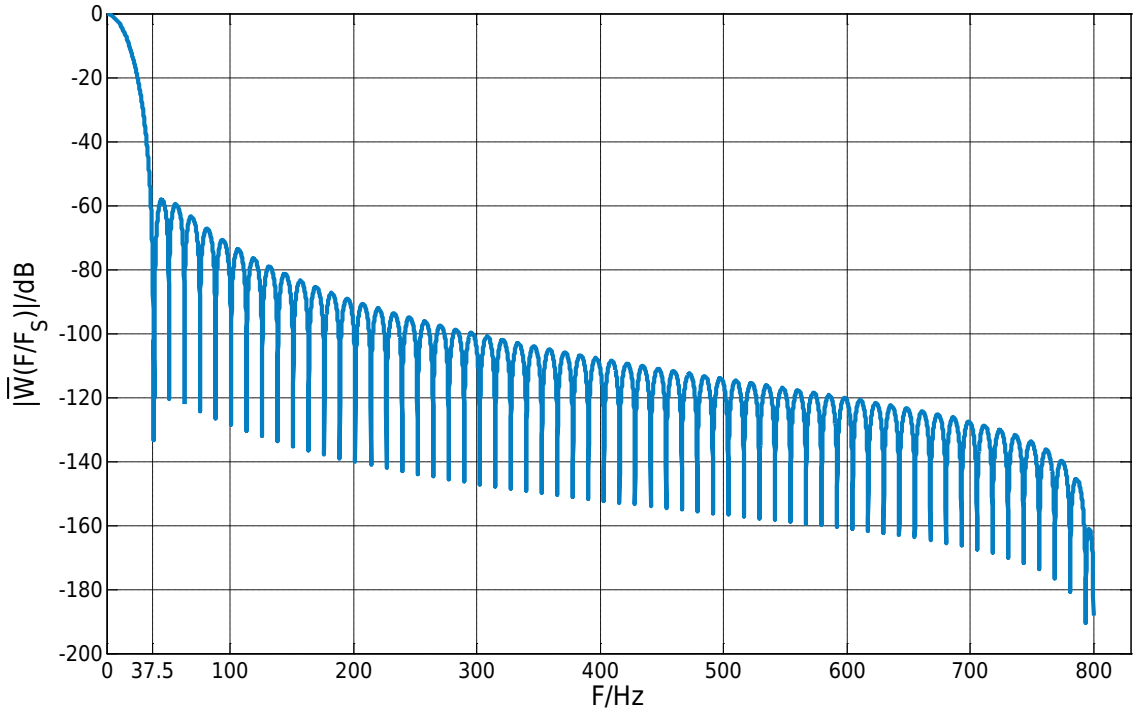


Figura 4.13 Respuesta en frecuencia de la ventana Blackman usada

Magnitud en unidades logarítmicas del espectro de $\bar{W}\left(F/F_S\right)$ *(posee simetría en frecuencias negativas). La pérdida que aporta la ventana es de* $57\,\mathrm{dB}$ *aprox. Usando una frecuencia de muestreo* $F_S = 1600\,\mathrm{Hz}$ *la resolución espectral resultará* $\Delta F = 3{\cdot}1600/128 = 37.5\,\mathrm{Hz}$ *, apta para las condiciones del problema.*

Un aspecto a tener en cuenta es que la ventana posee simetría respecto a su elemento central, lo que ahorra memoria a la hora de programarla en un microcontrolador.

* * *

En la Tabla 4.25 se recogen las variables y constantes utilizadas en la tarea `CalcDFT`.

Nombre	Tipo	Inicial	Funcionalidad
wndw	uint32_t const. vector (n)	Coeficientes de la ventana a aplicar	Cada muestra leída se multiplica por un coeficiente de la ventana
order	uint8_t const. vector (n)	Índices de las muestras a reordenar	Reordenación de las muestra iniciales para conseguir el resultado (DFT) en el orden correcto
expcomp	uint8_t const. vector (n)	Coeficientes parte real de exponencial compleja	Cálculo de la parte real e imaginaria para el algoritmo FFT (no hay que calcular senos y/o cosenos)
pot	uint8_t const. vector (r)	Potencias naturales de 2 (1,2,4...)	Cálculo de las potencias enteras de 2 (no hay que calcularlas)
y	static int32_t y var. matriz (n+1,2)	Se inicializa en algoritmo	Matriz bidimensional para alojar la transformada compleja de Fourier. La segunda dimensión indica parte real (0) o imaginaria (1).
modulus	static uint32_t var. vector (n)	Se inicializa en algoritmo	Almacenamiento de los módulos al cuadrado de las componentes de la DFT.
modulusdc	static uint32_t var.	Se inicializa en algoritmo	Módulo al cuadrado de la componente de continua (DC) de la DFT.
chan	static uint8_t var.	0	Canal analógico sobre el que se calculará la DFT. 0 para tensión, 1 para intensidad (alternativamente)
freq	static uint16_t var. vector (2)	Se inicializa en algoritmo	Estimación de la frecuencia fundamental (tensión e intensidad)
inddist	static uint32_t var. matriz (2,arm)	Se inicializa en algoritmo	Almacenamiento de indicadores de distorsión armónica. La primera dimensión indica si es tensión (0) o intensidad (1).
thd	static uint64_t var. vector (2)	Se inicializa en algoritmo	Guardado de las tasas de distorsión armónica total, para onda de tensión (0) e intensidad (1)
lbcounter	static uint8_t	0	Variable contador del algoritmo de control leaky bucket.
dft_calc	static uint8_t	0	Contador del número de procesamientos DFT realizados (tanto de tensión como intensidad)
m	uint8_t	0	Índice auxiliar del algoritmo FFT
q	uint8_t	0	Índice auxiliar del algoritmo FFT
k	uint8_t	0	Índice auxiliar del algoritmo FFT
n	uint8_t	0	Índice auxiliar del algoritmo FFT
inda	uint8_t	0	Índice auxiliar del algoritmo FFT
indb	uint8_t	0	Índice auxiliar del algoritmo FFT
ur	int32_t	0	Parte real de la exponencial compleja
ui	int32_t	0	Parte imaginaria de la exponencial compleja
auxa	int32_t	0	Almacenamiento de valores intermedios
auxb	int32_t	0	Almacenamiento de valores intermedios
auxc	int32_t	0	Almacenamiento de valores intermedios
auxd	int32_t	0	Almacenamiento de valores intermedios

Tabla 4.25 Variables y/o constantes locales tarea CalcDFT

Para comentar el funcionamiento de esta tarea, se van a tratar las cinco partes que la componen, tal y como se tiene en el código fuente[50].

4.24.3 Preparación de datos

El algoritmo FFT trabaja con el mismo buffer de medidas que el usado por la tarea de cálculo de potencia, es decir `raw`. Con el fin de evitar tener ocupada esta zona de memoria, lo primero que se hace es preprocesar y copiar estos valores en un buffer interno, denominado `y`.

Este preprocesamiento implica varias operaciones:

- Enventanado. Como se dedujo anteriormente, se deben de multiplicar por una ventana de tipo **Blackman** de 128 muestras. En el código solo se tienen la mitad, pues la otra mitad es simétrica.
- Selección del canal. El algoritmo solo opera en cada ejecución con una señal, es decir, se alternan tensión e intensidad.
- Aplicación de inversión binaria. Consiste en reordenar las muestras iniciales (en el dominio temporal) para obtener así una DFT en el dominio frecuencial ordenada.
- Separación de partes reales e imaginarias. Con el fin de reducir la carga computacional, dos muestras consecutivas pasan a formar parte real e imaginaria (respectivamente) de una muestra compleja. Así, se tendrá que calcular la DFT de una secuencia de $128/2 = 64$ puntos.
- Redondeo de valores, usando la macro creada a tal efecto.

Estos valores quedarán almacenados en su posición correcta en el buffer `y`.

[50] *Una tarea de estas características puede llegar a consumir demasiados recursos de CPU. Es por ello por lo que está dividida en varios bloques, de forma que en cada llamada de la tarea se va ejecutando uno de ellos (secuencialmente).*

4.24.4 Algoritmo FFT

Este bloque es el que lleva a cabo el algoritmo FFT en sí, tal y como se detalla en el pseudocódigo de A.3. El diagrama de flujo de la Figura 4.14 muestra los pasos a seguir.

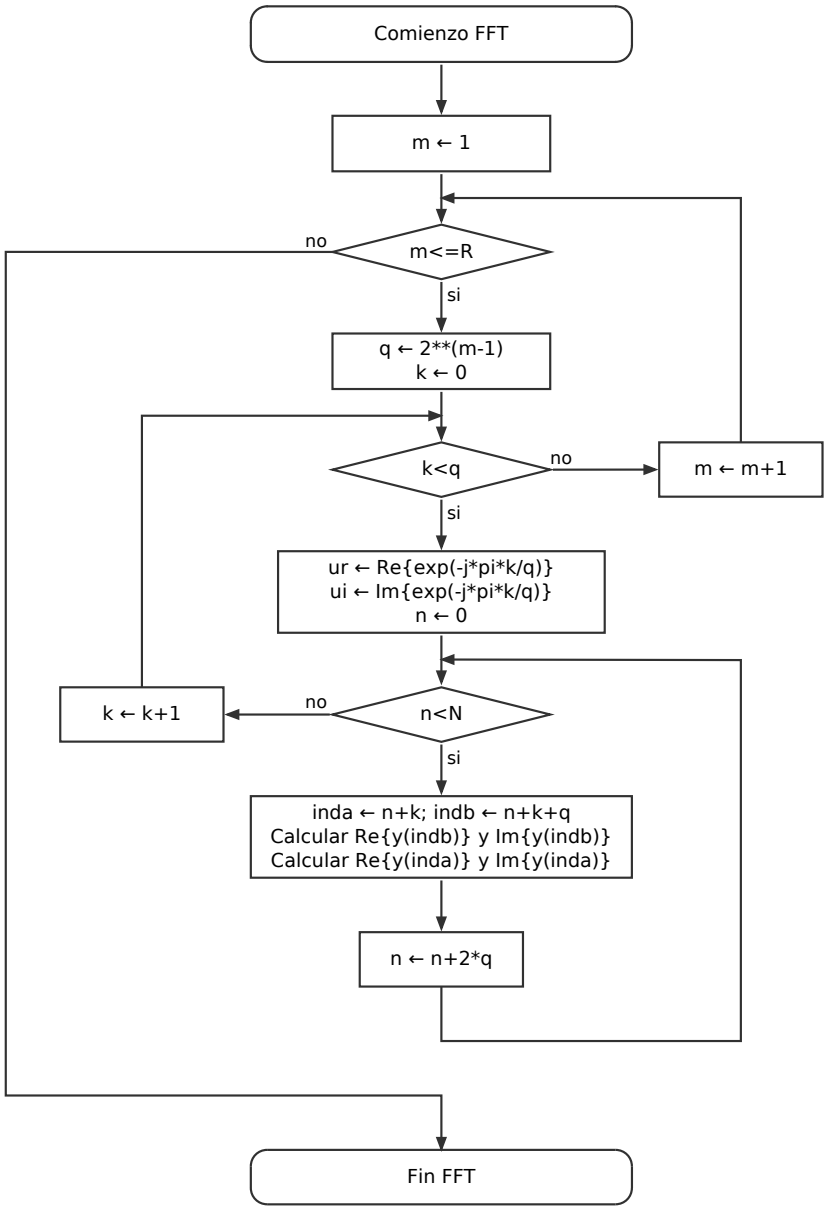


Figura 4.14 Diagrama de flujo algoritmo FFT

Como punto a destacar está el uso de la exponencial compleja. Al trabajar con partes reales e imaginarias, habrá que usar la expresión

$$W_N = e^{-j\frac{2\pi}{N}} = \cos\left(\frac{2\pi}{N}\right) - j\sin\left(\frac{2\pi}{N}\right) \tag{4.23}$$

Aunque las bibliotecas del compilador de nesC incluyen funciones trigonométricas, no se han utilizado por dos motivos: implican el uso de bibliotecas y funciones adicionales, incrementando la carga computacional del mote; necesitan variables con decimales que si bien nesC permite su uso, el microcontrolador MSP430F1611 no posee *hardware* nativo para ellos, lo que puede ocasionar problemas de cálculo.

La solución pasa por tener almacenados (en una tabla) los valores de la exponencial compleja escalados (para evitar el uso de decimales), constante `expcomp`. Además, para ahorrar espacio en memoria, solo se tienen los valores correspondientes a la parte real (coseno). El seno se calcula desfasando las muestras del coseno.

Los cálculos intermedios se realizan aplicando un redondeo. Al finalizar este bloque se tiene calculada, en el mismo vector y, la Transformada Discreta de Fourier (aunque no todavía la que se necesita).

4.24.5 DFT de la secuencia original

Las muestras originales son reales. Para ahorrar tiempo de cálculo de la DFT, se transforman en una secuencia de la mitad de longitud, pero en este caso compleja.

En este punto hay que deshacer ese cambio: a partir de la DFT de la secuencia compleja, calcular la DFT de los datos originales. Las expresiones (4.24) y (4.25) (comentadas en A.3.1) permiten este cambio.

$$\begin{aligned} X_1[k] &= \frac{1}{2}\{X[k] + X^*[N-k]\} \\ X_2[k] &= \frac{1}{2j}\{X[k] - X^*[N-k]\} \end{aligned} \tag{4.24}$$

$$\begin{aligned} G[k] &= X_1[k] + W_{2N}^k X_2[k] \\ G[k+N] &= X_1[k] - W_{2N}^k X_2[k] \end{aligned} \quad k = 0,1,\ldots,N-1 \tag{4.25}$$

$X[k]$ es la DFT de la secuencia compleja $x[n]$, generada a partir de las muestras reales ($g[n]$). Precisamente la transformada discreta buscada es $G[k]$.

$G[k]$ tendrá en principio $2N = 128$ componentes. Sin embargo, como la secuencia original es real, solo aportaran información espectral $N + 1 = 65$ muestras. De ahí que el vector y tenga 65 componentes.

Primero se calculan algunas muestras particulares, como

- Primera componente $G[0]$. Solo se modifica su parte real (la imaginaria es nula).
- Última componente $G[N]$. Solo se modifica su parte real (imaginaria nula). En este caso, hay que multiplicar por dos (para ajustar los módulos).
- Componente $G[N/2]$. Cambio de signo en su parte imaginaria.

Después, el resto de valores, mediante un bucle FOR. Aprovechando las propiedades de simetría, en cada iteración se calculan dos componentes: $G[k]$ y $G[N-k]$.

Además de otros escalados, es importante tener en cuenta la denominada ganancia coherente[51] de la ventana. Se ha elegido una ventana Blackman de 128 componentes, luego su ganancia coherente es de $CG = 53.34/128 \approx 0.4167$. Como no se utilizan decimales, se multiplicará por 2^{16} y después se dividirá entre 10 (redondeos). El resultado, 2731 se guarda en la constante `CGAIN_ADJ`.

Cuando concluye este bloque, se tiene calculada la DFT de la señal digitalizada original (en el vector y).

[51] *En inglés, coherent gain. Grosso modo se trata de un factor a aplicar en el resultado de la DFT calculada, para corregir los valores absolutos de módulos que se obtienen.*

Se halla como $CG = \dfrac{\sum_{n=0}^{M} w[n]}{M+1}$, *siendo* $w[n]$ *la ventana y* $M+1$ *las muestras totales (Schmid, 2009).*

4.24.6 Parámetros armónicos

Básicamente se calculan los módulos o valores absolutos de cada una de las componentes espectrales[52].

$$\left|y\left[k\right]\right|^2 = \mathfrak{Re}\left\{y\left[k\right]\right\}^2 + \mathfrak{Im}\left\{y\left[k\right]\right\}^2 \qquad (4.26)$$

También se realiza una estimación de la frecuencia fundamental (debería estar en torno a los $50\,\mathrm{Hz}$), usando un polinomio interpolador de Lagrange de segundo grado.

Los indicadores de distorsión armónica indican qué porcentajes de armónicos existen en la señal respecto del fundamental: U_h para la onda de tensión e I_h para la de intensidad. Debido a que los módulos están elevados al cuadrado, el resultado se expresa en tanto por mil y elevado al cuadrado: $U_h\left(‰\right)^2$ e $I_h\left(‰\right)^2$.

4.24.7 Tasa de distorsión total armónica

El cálculo de la THD se realiza con un bucle FOR que suma todos los módulos (al cuadrado) de los armónicos de la frecuencia fundamental. Para después dividir el resultado entre el principal. De la misma forma que antes, el resultado se expresa en tanto por mil al cuadrado.

La última operación que se realiza es el control de la distorsión (de tensión e intensidad) mediante el algoritmo denominado cubo agujereado, o *leaky bucket*. La Figura 4.15 ilustra este sistema.

[52] *La expresión* (4.26) *permite hallar el módulo al cuadrado. Tal y como ocurría con los cálculos de potencia, se prefiere el resultado así y no utilizar una función adicional para extraer su raíz cuadrada.*

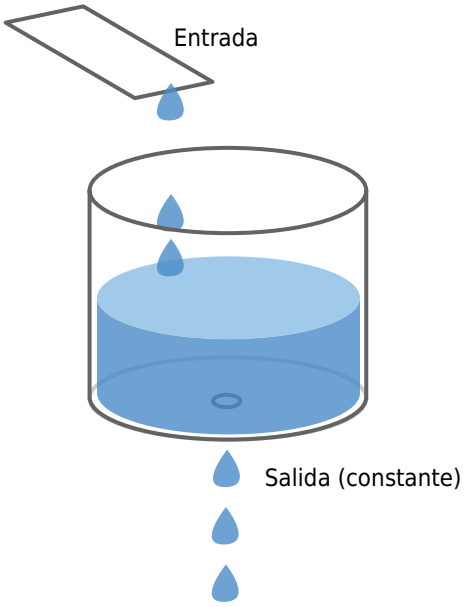


Figura 4.15 Ilustración del algoritmo del cubo agujereado

La entrada representa en este caso cada detección de distorsión elevada en el mote. El agujero inferior que posee el cubo permite que el nivel descienda cada cierto tiempo (y de forma constante): de lo contrario siempre acabaría por llenarse y rebosar. Si la tasa de distorsión es elevada, llegará un momento en el que supere la capacidad del cubo y el sensor dará un aviso. Si la tasa se mantiene por debajo de los límites (o los sobrepasa en momentos aislados), no llegará a rebosar.

Así, los parámetros usados en el código son

- `LB_UPT`. Límite superior. En caso de que se llegue a dicho límite, se activa una salida digital de propósito general (se puede programar otro comportamiento distinto). Esto indica que existe una distorsión que se mantiene en el tiempo, es decir, no es aislada. Equivaldría al rebosamiento del cubo.
- `LB_LWT`. Límite inferior. Permite añadir histéresis al sistema. Una vez se ha activado la salida indicando distorsión, la variable de control (`lb_counter`) debe descender hasta llegar a este límite. Esto evita que haya demasiadas conmutaciones de la salida digital.
- `LB_D`. Factor de olvido. Cada `LB_D` cálculos DFT, se decrementa una unidad la variable de control. Equivale al agujero que tiene el cubo, que hace descender el nivel de líquido.

4.25 Significado de los LED

La aplicación implementa funcionalidad para los LED de propósito general, con el fin de informar de ciertos eventos, tal y como se recoge en Tabla 4.26 (para un nodo raíz) y Tabla 4.27 (para nodo de datos). El manejo y funcionamiento de los mismos se describió en 4.5.1.

LED	Estado	Significado
0 (rojo)	ON	Error al enviar datos por el puerto serie
0 (rojo)	OFF	Datos por puerto serie enviados correctamente
1 (verde)	ON	Recepción de datos vía radio
1 (verde)	OFF	Procesamiento de los datos radio realizado
2 (azul)	ON	Recepción de datos vía puerto serie
2 (azul)	OFF	Datos recibidos por puerto serie diseminados a la red

Tabla 4.26 Significado de los LED para un nodo raíz

LED	Estado	Significado
0 (rojo)	ON	Desbordamiento en el convertidor A-D (tanto en medidas de V/I como de los sensores auxiliares)
0 (rojo)	OFF	Ausencia de desbordamiento en el convertidor A-D
1 (verde)	ON	Convertidor A-D ocupado
1 (verde)	OFF	Convertidor A-D libre (datos de conversión ya disponibles)
2 (azul)	ON	Radio enviando datos hacia nodo raíz / Escritura en memoria flash
2 (azul)	OFF	Datos radio enviados / Escritura en flash terminada

Tabla 4.27 Significado de los LED para un nodo de datos

4.26 Representación gráfica de la aplicación

En el Capítulo 3 se comentó la forma de representar gráficamente los componentes que conforman una aplicación, y las interfaces existentes entre ellos.

Estos diagramas pueden crearse de forma automática, gracias a la aplicación de código abierto Graphviz. Para generar el gráfico (además de documentación acerca

de los componentes e interfaces en formato *html*) se escribe en Cygwin (para el caso Tmote Sky)

```
make telosb docs
```

En la Figura 4.16 se muestra el diagrama generado para el componente de configuración `MaxwellAppC.nc` (realiza el cableado de todos los componentes que conforman la aplicación).

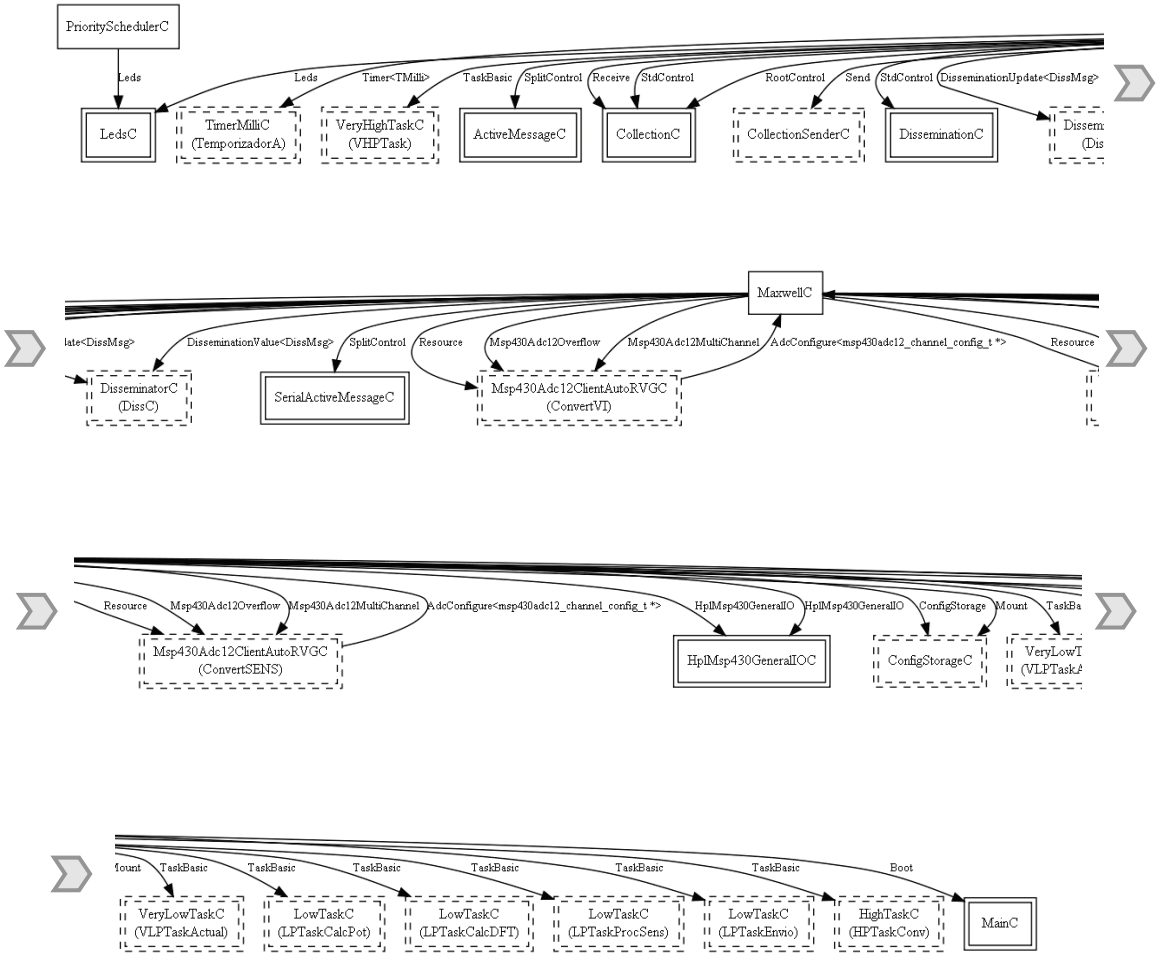


Figura 4.16 Diagrama de aplicación (Graphviz)

Estructura del componente (de configuración) MaxwellAppC. Se muestran las relaciones del módulo MaxwellC con el resto de componentes utilizados en la aplicación, así como las interfaces que permiten el cableado o wiring.

4.27 Comunicación Ethernet

Se trata de poder conectar un mote (puede ser por ejemplo el nodo raíz de una red) a una pasarela, de forma que los mensajes que se transfieran por el puerto serie del mismo pasen a una red de área local o LAN. De esta forma, cualquier equipo que esté conectado a esta red podría recibir (y/o enviar) información de forma remota.

La Figura 4.17 muestra la conexión de esta pasarela con un *router*.

Figura 4.17 Conexión pasarela Tmote Connect con router

El NSLU se conecta al router a través de un cable Ethernet. En la imagen se aprecia un mote conectado (al puerto USB1). La conexión inferior es la alimentación de la pasarela. En este caso, la comunicación con la pasarela a través del router se realiza inalámbricamente (802.11g).

4.27.1 Tmote Connect

Este *firmware* permite que el dispositivo NSLU2 pueda funcionar como pasarela interconectando un nodo (a través de su puerto USB) a una red de área local.

En la Figura 4.18 se muestra una captura de la interfaz web de Tmote Connect. En la parte izquierda se muestra información acerca del dispositivo físico: código de identificación, dirección IP asignada y la dirección física (MAC). También se inclu-

yen opciones para resetear el servidor de datos y los posibles clientes que hubiese, así como el propio sistema.

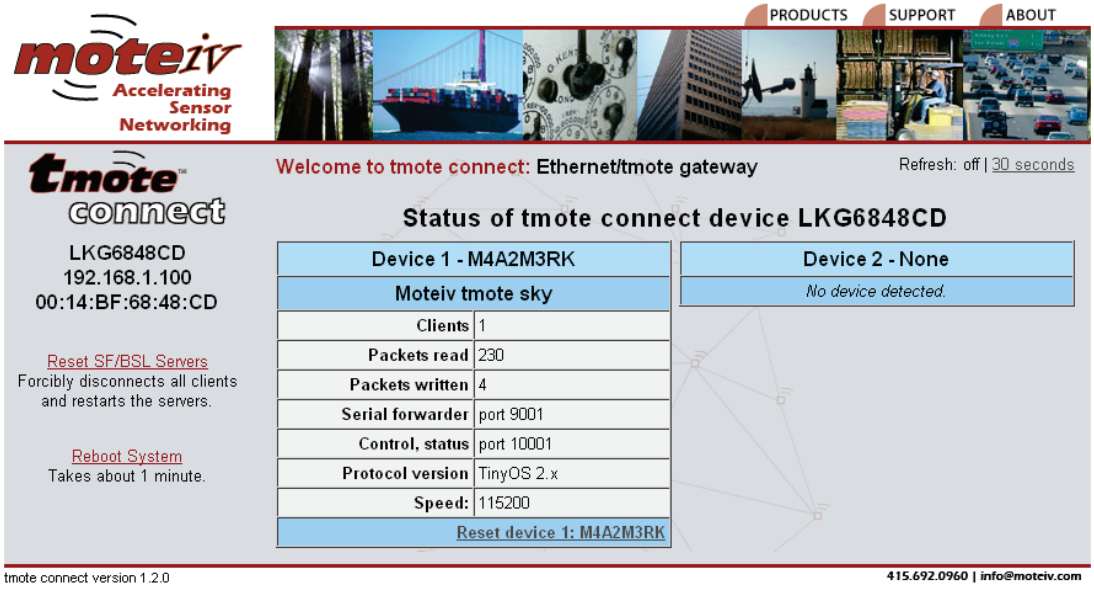


Figura 4.18 Captura de pantalla Tmote Connect

Pantalla principal de la pasarela Tmote Connect. Muestra información básica tanto del dispositivo como de los motes conectados.

La parte fundamental muestra el estado de los dos posibles motes que se pueden conectar a la pasarela. En la imagen solo existe un único sensor conectado (modelo Tmote Sky). Los campos de información:

- **Clients**. Muestra el número de clientes TCP/IP que están haciendo uso de la pasarela (transmitiendo y/o recibiendo datos).
- **Packets read**. Número de paquetes leídos por el mote, por ejemplo, los recibidos a través de un protocolo de recolección de datos.
- **Packets written**. Paquetes enviados hacia el mote. En este caso, serían paquetes enviados desde un equipo conectado a la red Ethernet.
- **Serial forwarder**. Indica el puerto TCP por el cual se puede tanto acceder a la información que recibe el sensor como enviar paquetes de datos hacia él. Está configurado el número 9001 para el mote conectado al USB 1 y el 9002 para el USB 2.
- **Control, status**. Puerto TCP para acceder a datos de estado del mote, así como para enviar comandos de control del mismo. Se asignan sendos números 10001 y 10002.

- **Protocol version**. Versión del sistema operativo utilizado.
- **Speed**. Velocidad (baudios) de transmisión de datos.

Se ofrece la posibilidad de reiniciar cada uno de los motes en particular.

Por último, en la parte derecha se puede activar una actualización de esta página automáticamente cada 30 segundos.

4.27.2 Operaciones básicas

Se describen las operaciones fundamentales que pueden realizarse en el dispositivo NSLU2 desde un equipo externo (PC).

RECUPERAR EL ESTADO DEL MOTE

Desde la consola de Cygwin basta escribir

```
echo status | nc <DirIP> <PuertoControl>
```

Para el ejemplo de la figura, quedaría

```
echo status | nc 192.168.1.100 10001
```

El comando devuelve la información

```
MotePresent Yes
Manufacturer Moteiv
Product tmote sky
SerialNumber M4A2M3RK
Errors none
NumClients 1
PacketsRead 230
PacketsWritten 4
ProtocolVersion 2
Speed 115200
ActualProtocolVersion 2
ActualSpeed 115200
```

REINICIAR EL SERVIDOR DE CONTROL

El servidor de control de los dispositivos funciona mediante instancias. Cuando se llama al comando `exit`, se elimina la instancia actual y se crea una nueva. Uso:

```
echo exit | nc <DirIP> <PuertoControl>
```

REINICIAR EL DISPOSITIVO TMOTE CONNECT

Puede ocurrir que en determinadas ocasiones haya que reiniciar el dispositivo físico completo. Una vez llamado, no puede cancelarse. Aproximadamente después de un minuto, se muestra una confirmación de que se ha reiniciado.

```
echo unfriendly_system_reboot | nc <DirIP> <PuertoControl>
```

CONTROL DEL PROTOCOLO Y DE LA VELOCIDAD

Para modificar la versión del protocolo que se usa, se puede llamar al comando `protocol`. Acepta tres posibles opciones: `auto`, `tinyos1.x` y `tinyos2.x`:

```
echo “protocol <opcion>” | nc <DirIP> <PuertoControl>
```

Por su parte, la velocidad de transmisión de datos también puede modificarse, ahora usando `baudrate`. Hay que especificar una velocidad (en baudios). Si se escribe `0`, se toma automáticamente según el protocolo que se esté usando.

```
echo “baudrate <velocidad>” | nc <DirIP> <PuertoControl>
```

PROGRAMACIÓN DEL MOTE

La forma más sencilla para realizar esta operación es usar la herramienta de TinyOS denominada `netbsl`.

Una vez se tenga compilada la aplicación a instalar, hay que escribir en la consola de comandos

```
make <modelo> reinstall,<IDnodo> netbsl,<DirIP:PuertoControl>
```

Por ejemplo, para un mote Tmote Sky que se quiera asignar un identificador de red `10`, quedaría

```
make tmote reinstall,10 netbsl,192.168.1.100:10001
```

Y la salida que se muestra

```
tos-set-symbols --objcopy msp430-objcopy --objdump msp430-objdump
--target ihex build/telosb/main.ihex build/telosb/main.ihex.out-10
  TOS_NODE_ID=10 ActiveMessageAddressC__addr=10
    installing telosb binary using netbsl
```

```
/opt/tinyos-2.x/support/make/msp/netbsl 192.168.1.100:10001
  build/telosb/main.ihex.out-10 --telosb -r -e -I -p

Using mote  on port /dev/ttyUSB0.
(...)
Reset
rm -f build/telosb/main.exe.out-10 build/telosb/main.ihex.out-10
```

Este es el procedimiento más sencillo. Sin embargo, a través de la aplicación `msp430-bsl` es posible realizar las operaciones individualmente. La forma general de usarla es

```
(echo "msp430-bsl <opciones>"; echo) | nc <DirIP> <PuertoControl>
```

Entre las opciones disponibles se encuentran: borrar toda la memoria flash del dispositivo (`-e`), comprobar el borrado (`-E`), programar un archivo previamente compilado (`-p`), reiniciar un mote (`-r`) o verificar el contenido de la memoria (`-v`). La opción `-h` muestra la ayuda disponible.

Adicionalmente, se puede especificar la plataforma en la cual se va a instalar la aplicación: Tmote Sky (`--tmote`), TelosA (`--telos`) o bien TelosB (`--telosb`).

Por ejemplo, para reiniciar remotamente un sensor modelo Tmote Sky, bastaría con escribir

```
(echo "msp430-bsl --tmote -r"; echo) | nc 192.168.1.100 10001
```

4.27.3 Configuración del NSLU2

Se trata de acceder a operaciones de mantenimiento y configuración internas del *hardware* NSLU2 (y no al *firmware* Tmote Connect). Para ello, basta acceder desde un navegador web a la dirección IP que tenga asignada la pasarela, y añadir a la dirección `/linksys.cgi`. Por ejemplo: `192.168.1.100/linksys.cgi`. Se muestra la pantalla de configuración como la Figura 4.19.

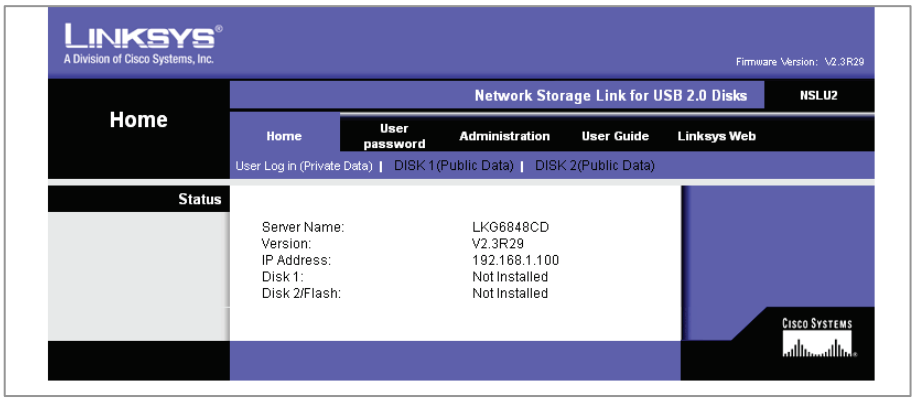


Figura 4.19 Pantalla principal de configuración NSLU2

Por ejemplo, en el apartado `Administration > LAN` se puede establecer una dirección IP fija para el dispositivo (Figura 4.20).

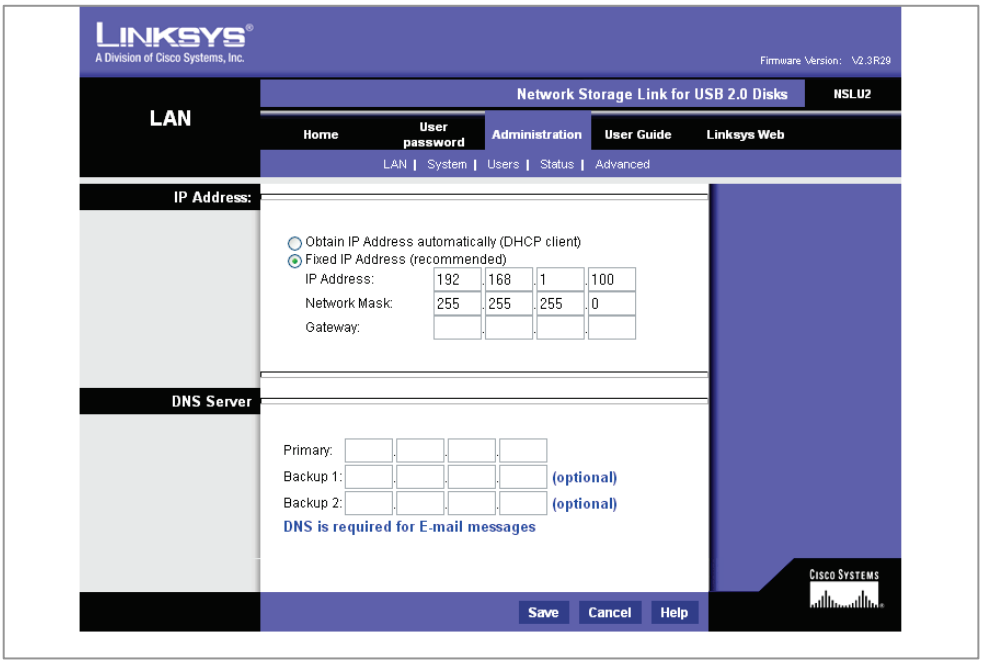


Figura 4.20 Configuración LAN del NSLU2

4.28 Comunicación TCP/IP. MATLAB

Mediante el software MATLAB es posible establecer una comunicación en una red de área local usando los protocolos de comunicaciones TCP/IP.

Aunque MATLAB incluye funciones para abrir, establecer, cerrar una conexión TCP, como `tcpip`, `fopen`, `fwrite`[53]... existe una función desarrollada por terceros que trabaja con este protocolo y aporta más funcionalidad. Se denomina `pnet`, y pertenece al toolbox TCP/UDP/IP Toolbox 2.0.6. (Rydesäter, y otros).

Esta función está escrita en lenguaje C, y para poder utilizarla en MATLAB es necesario compilarla y generar un archivo `.mex` (*MATLAB EXecutable*). Esto dependerá del compilador y sistema operativo usado (Faria):

- En sistemas UNIX, hay que ejecutar en MATLAB:

```
mex -O pnet.c
```

- En sistemas bajo Windows, se puede usar el compilador LCC:

```
mex -O pnet.c {DIR_INSTAL_MATLAB}\sys\lcc\lib\wsock32.lib -DWIN32
```

  O bien Visual C++:

```
mex -O pnet.c ws2_32.lib -DWIN32
```

La forma de utilizar la función `pnet` es

```
pnet('comando',parametros);
pnet(con,'comando',parametros);
```

La variable `con` es un manejador de conexión. Cuando se establece una, devuelve un entero con el que se pueden modificar las propiedades de dicha conexión.

A continuación se describen las operaciones más importantes que se incluyen en la función `pnet`.

```
con=pnet('tcpconnect','nombre_equipo',puerto);
```

Crea una conexión TCP/IP con el equipo indicado en `nombre_equipo` y en el número de puerto. El nombre de equipo puede ser una cadena de caracteres (por ejemplo, `'localhost'`) o una dirección IP decimal.
Devuelve un entero mayor o igual a cero que identifica la conexión (negativo si hubo un error).

[53] *Deberá instalarse el toolbox "Instrument Control".*

```
sockcon=pnet('tcpsocket',puerto);
```

Abre un *socket* TCP en el puerto indicado. Al igual que ocurre con las conexiones, devuelve un entero (negativo en caso de error) con el manejador del socket.

```
con=pnet(sockcon,'tcplisten', ['noblock'] );
```

Si algún equipo se ha conectado al socket TCP especificado en `sockcon`, devuelve un identificador con para dicha conexión (si no hubiera conexión alguna, devuelve -1). `'noblock'` es opcional, y evita bloqueos en la operación.

```
elementos=pnet(con,'write', datos [,orden_bytes]);
```

Envía los datos especificados a la conexión indicada en `con`. El tipo de datos puede ser entero, con o sin signo, de 8 b , 16 b , 32 b , caracteres... `orden_bytes` (opcional) permite indicar el criterio de ordenación de los bytes más significativos (*little/big endian*). Retorna el número de datos escritos correctamente.

```
pnet(con,'printf','formato',...);
```

Envía a la conexión `con` una cadena con formato. Muchos de los indicadores de formato de MATLAB son similares a lenguaje C. Por ejemplo, `\n` introduce un salto de línea, `\t` una tabulación, `%s` para escribir cadenas, `%f` para números con punto fijo...

```
data=pnet(con,'read' [,tamaño] [,tipo_datos] [,orden_bytes]
  [,'view'] [,'noblock']);
```

Lee y devuelve los datos desde una conexión determinada `con`. En tamaño se especifica la cantidad de datos a recuperar. En `tipo_datos` se indica la forma de interpretar los datos (igual que en comando `'write'`). `orden_bytes` también es similar a `'write'`. Si se especifica `'view'`, ofrece una vista previa de los datos, pero éstos permanecerían todavía en el *buffer* de lectura. La opción `'noblock'` es muy útil junto a `'view'` para comprobar los datos que existen para leer sin bloquear el resto de operaciones.

```
estado=pnet(con,'status');
```

Devuelve un código con el estado en el que se encuentra la conexión dada por con. Los valores se encuentran definidos en el fichero fuente pnet.c. Por ejemplo, el valor 0 indica que el servidor se ha desconectado.

```
estado=pnet(con,'setreadtimeout'/'setwritetimeout',seg);
```

Permite especificar los tiempos máximos de espera (en segundos) en una operación de lectura/escritura (respect.). Es útil para evitar grandes tiempos de bloqueo en las operaciones.

```
[DirIP,puerto]=pnet(con,'gethost');
```

Devuelve la dirección IP y el número de puerto del último equipo que esté asociado a la conexión con.

```
pnet(con,'close');
```

Cierra la conexión o *socket* TCP identificados mediante con.

```
pnet('closeall');
```

Cierra todas las conexiones y *sockets* establecidos en la sesión actual de MATLAB.

4.29 Monitorización y control en MATLAB

Para implementar un sistema de recogida y procesamiento de datos, así como de control básico de los sensores remotos a través de MATLAB, se usará la función pnet (perteneciente al toolbox TCP/UDP/IP 2.0.6). También se empleará el dispositivo NSLU2 (con Tmote Connect instalado), que hace las veces de pasarela.

4.29.1 Arranque del programa

Lo primero que se hace es comprobar si existe una conexión anterior. Si es así, se comprueba el estado de la misma: puede ser una conexión fallida, o que el servi-

dor remoto se haya desconectado, en ambos casos se borran todos los datos y se termina el programa; si no tuviera errores, se utiliza ese identificador.

En caso de que no exista ninguna conexión, se pasa a crear una, el programa muestra un cuadro de diálogo como el de la Figura 4.21.

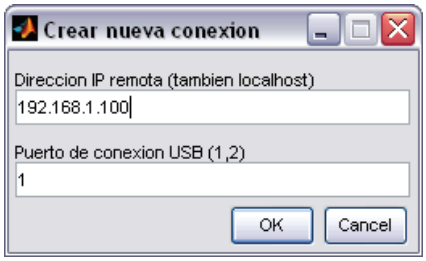


Figura 4.21 Cuadro de diálogo Crear nueva conexión

En la ventana se pide la dirección IP remota desde donde recoger/enviar los datos, esto es, la pasarela Tmote Connect (es recomendable dar una IP fija al dispositivo a través de su pantalla de configuración, ver 4.27.3). Se debe especificar en formato decimal. Después, hay que especificar en qué puerto USB está conectado el nodo raíz (la pasarela solo dispone de dos puertos).

Si los datos son correctos, se intenta establecer la conexión. Una vez hecho esto, se debe de enviar una *cookie* (o *handshake*), para iniciar la sesión con la pasarela Tmote Connect. Si el servidor la devuelve, la conexión se habrá establecido sin problemas (Figura 4.22).

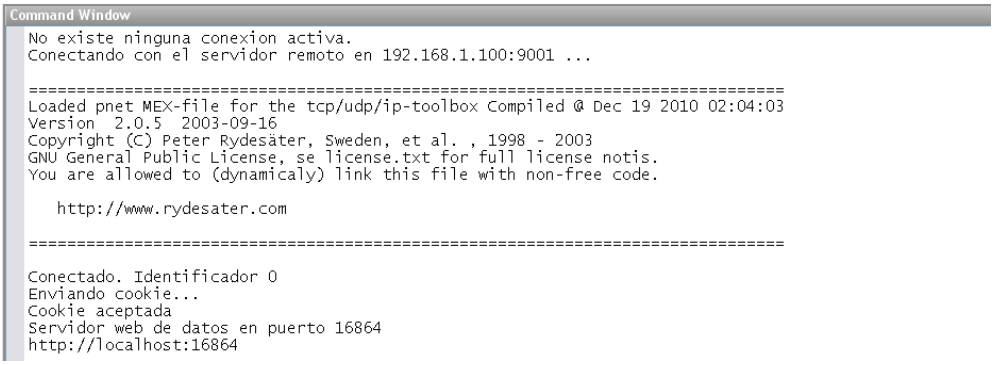


Figura 4.22 Ventana de comandos MATLAB (I)

Se muestran mensajes indicando el estado de la conexión.

A su vez, se genera un puerto TCP aleatorio para que se pueda acceder a la información recibida vía web (a través de HTTP). Después, el programa muestra el menú principal (Figura 4.23).

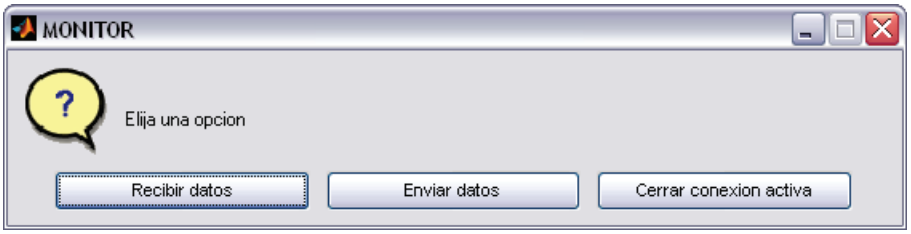


Figura 4.23 Menú principal programa Monitor

Una vez se ha establecido una conexión correctamente con la pasarela, se dan las opciones de Recibir datos (desde la red de sensores a través del nodo raíz), Enviar datos (a un nodo concreto o a toda la red en modo difusión) o Cerrar la conexión activa creada.

4.29.2 Recepción de datos. Gráficas. Servidor web de datos

Al seleccionar esta opción, se pregunta al usuario si desea mostrar ventanas gráficas de evolución temporal para cada uno de los nodos remotos de la red.

Después, se llega a un bucle WHILE infinito, el cual está continuamente comprobando el *buffer* de datos de lectura de la pasarela. Los paquetes de datos siempre comienzan con la longitud del mismo, por lo que lo primero es comprobar la longitud, y a continuación leer esa cantidad de datos. A veces pueden existir problemas de sincronismo con la pasarela, en cuyo caso aparecerá un mensaje indicándolo.

La información se recibe en forma de grupos de $1\,\mathrm{B}$. Como las variables que se envían (valores medios, eficaces, potencias...) ocupan más de $1\,\mathrm{B}$, es necesario un procesamiento para recuperar los valores originales. Esta operación se ha implementado en una función auxiliar, denominada `procdata`.

Por ejemplo, si la potencia activa contiene $4\,\mathrm{B}$, y se reciben (en decimal) `10 130 240 84`, el resultado final será

$$\underline{10}\cdot 256^3 + \underline{130}\cdot 256^2 + \underline{240}\cdot 256^1 + \underline{84}\cdot 256^0 = 176353364 \qquad (4.27)$$

El paso siguiente es el procesamiento. Por ejemplo, para la tensión media habría que deshacer el cambio del convertidor analógico-digital:

$$V_m\left(\mathrm{mV}\right) = \frac{2.5\cdot 1000}{4095}\cdot vmed \qquad (4.28)$$

O el factor de potencia, que es enviado en forma de porcentaje al cuadrado, es en MATLAB donde se aplica la raíz cuadrada al valor recibido.

Los resultados se formatean y se van mostrando por la ventana de comandos de MATLAB, tal y como se aprecia en la Figura 4.24.

```
Cookie aceptada
Servidor web de datos en puerto 16864
http://localhost:16864

Para detener el programa, CTRL+C

 Tiempo   CTR  GRP  HID  MotID  Vmed  Imed  Vrms  Irms    P.apar     P.act      P.reac    fdp     THD v     THD i    Temp  Volt
hh:mm:ss   #    #    #     #     mV    mA    mV    mA       uVA        uW        uVAR      %        %         %       ºC    mV
--:--:--  ---  ---  ---  -----  ----  ----  ----  ----  ---------  ---------  ---------  ----  --------  --------  ----  ----
14:26:03    0    0   18     23  1491  1497  1491  1497  2233082.4  2209900.4   320932.1  99.0    620.25    323.60  74.9  2529
14:26:05    0    0   18     78  1526  1532  1528  1534  2343370.3  2323589.9   303832.8  99.2    331.94    633.24  27.5  2929
14:26:06    0    0   18     23  1491  1497  1491  1497  2233082.4  2209900.4   320932.1  99.0    479.34    199.31  74.3  2532
14:26:10    0    0   18     78  1526  1532  1528  1534  2343370.3  2323589.9   303832.8  99.2   1880.31    144.66  27.5  2929
14:26:11    0    0   18     23  1496  1494  1496  1494  2234962.4  2213349.5   310066.6  99.0    490.48    284.48  74.8  2531
14:26:12    0    0   18     78  1537  1529  1538  1529  2351955.5  2323967.1   361762.0  98.8   1020.39    268.75  27.6  2934
14:26:16    0    0   18     78  1528  1527  1529  1528  2336629.5  2313374.3   328841.7  99.0    323.29    317.74  27.6  2933
14:26:17    0    0   18     23  1493  1491  1494  1492  2229538.9  2207590.3   312071.4  99.0    575.52    241.99  74.9  2530
14:26:20    0    0   18     78  1517  1521  1519  1523  2312814.7  2293585.9   297616.0  99.2    189.72    186.44  27.5  2929
14:26:22    0    0   18     23  1493  1491  1494  1492  2229538.9  2207590.3   312071.4  99.0    126.16    140.34  75.1  2531
14:26:24    0    0   18     78  1517  1521  1519  1523  2312814.7  2293585.9   297616.0  99.2    235.42    286.65  27.4  2929
14:26:26    0    0   18     23  1499  1498  1499  1498  2246579.7  2222634.1   327136.4  98.9    143.10    155.98  74.9  2531
14:26:28    0    0   18     78  1535  1536  1536  1537  2360943.1  2335501.8   345664.2  98.9    257.73    624.76  27.4  2929
14:26:29    0    0   18     23  1487  1495  1488  1495  2223559.0  2200569.6   318916.8  99.0    165.63    413.68  74.9  2532
14:26:32    0    0   18     78  1535  1533  1536  1534  2355760.8  2333912.9   320093.2  99.1    405.14    280.38  27.5  2933
14:26:33    0    0   18     23  1496  1498  1497  1499  2243666.6  2223938.9   296876.0  99.1    486.62    131.18  75.0  2530
```

Figura 4.24 Ventana de comandos MATLAB (II)

Si se escoge la opción de Recepción de datos, comienzan a recibirse los datos de todos los sensores presentes en la red actual (en el orden en el que vayan llegando al nodo raíz o base).

Además, se ha implementado una cola o *buffer* de datos circular, en el que van quedando las últimas medidas recibidas (el tamaño del *buffer* se puede modificar). Esto se utiliza a la hora de enviar los datos a un navegador web.

Si el usuario indicó la opción de representación gráfica, en este punto se generan las representaciones de algunos de los datos. Es importante conocer qué nodo envió los datos, pues cada uno de ellos tiene sus propias ventanas gráficas (Figura 4.25). Estas ventanas se actualizan en tiempo real (tal y como van llegando los paquetes de datos).

El último bloque es la implementación del servidor web. Lo primero es comprobar si existe alguna conexión establecida con MATLAB, que hace las veces de servidor. Si es así, se envía el código fuente de la página web.

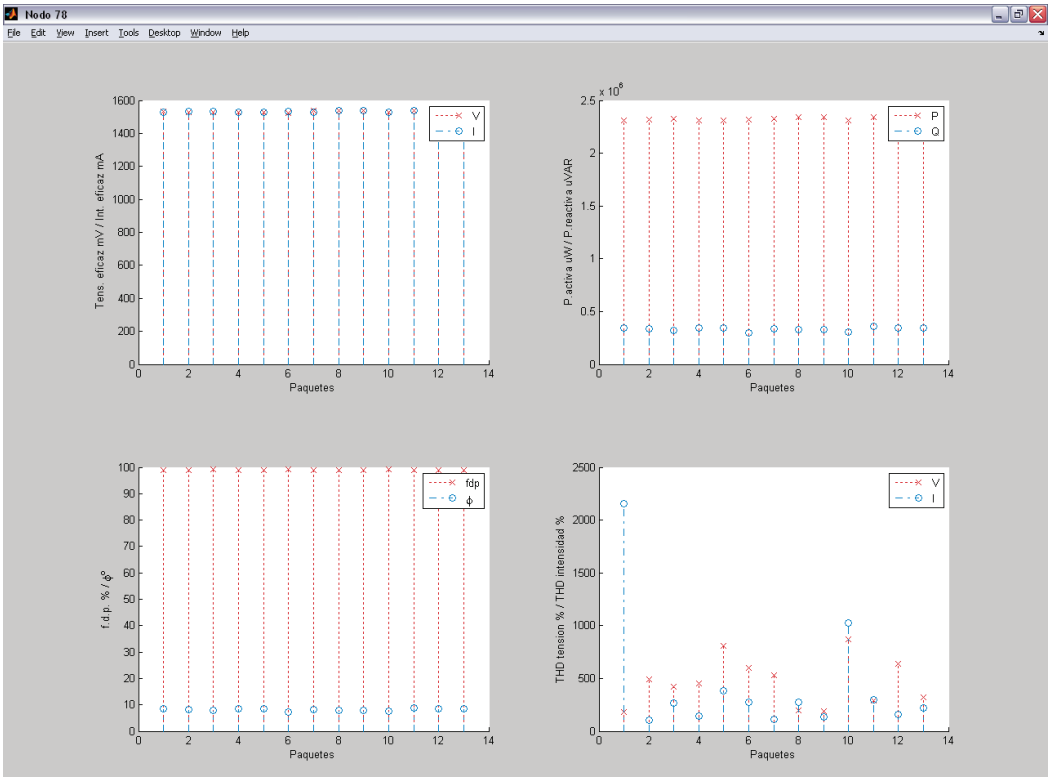
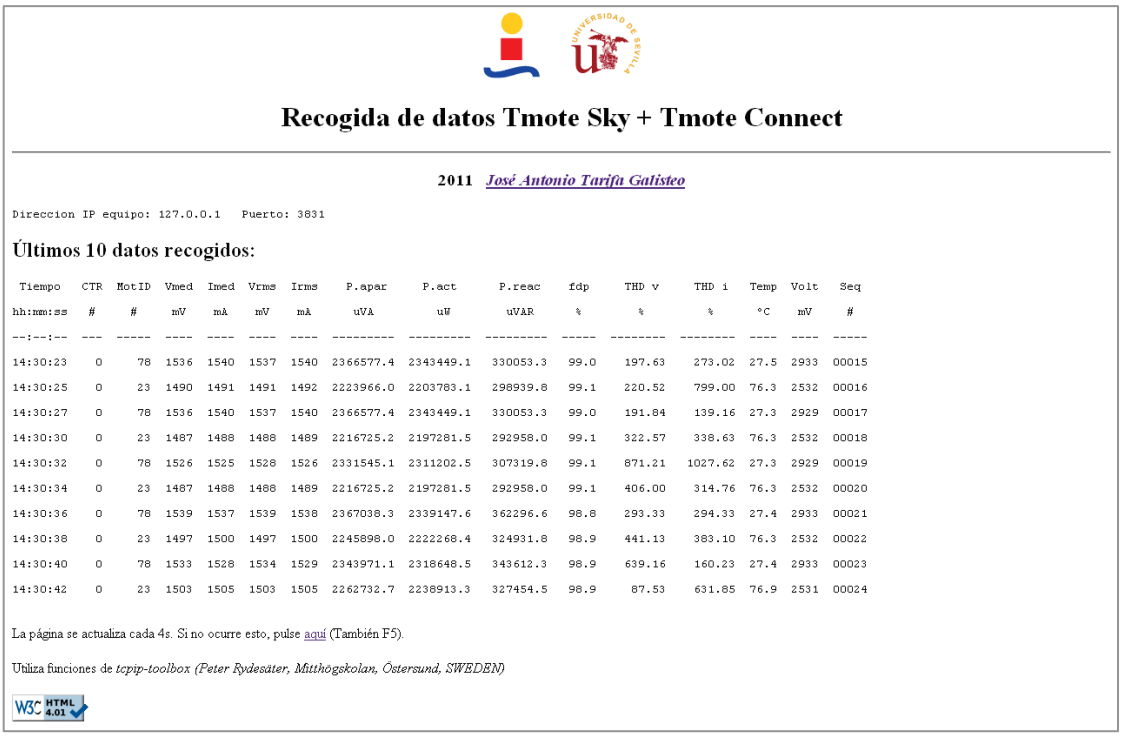


Figura 4.25 Gráficas de datos en MATLAB

Se tiene una ventana gráfica para cada nodo de la red (en la figura se muestra el de identificador 78). De izquierda a derecha y de arriba a abajo: (1) tensión e intensidad eficaz; (2) potencias activa y reactiva; (3) factor de potencia en % y ángulo ϕ en grados; (4) tasas de distorsión armónica total (%) de tensión e intensidad.

El código fuente está escrito en lenguaje *HTML 4.01 Transitional*, cumpliendo las reglas de dicho estándar. Las imágenes que aparecen en la página web están insertadas directamente en el código, es decir, no están vinculadas a ningún archivo externo. Dicha página puede verse usando un navegador web cualquiera, y su apariencia es similar a la de la Figura 4.26.

En la ventana de comandos de MATLAB aparece un mensaje cada vez que un navegador accede al servidor web, indicando su dirección IP y puerto remotos.



Recogida de datos Tmote Sky + Tmote Connect

2011 *José Antonio Tarifa Galisteo*

Direccion IP equipo: 127.0.0.1 Puerto: 3831

Últimos 10 datos recogidos:

Tiempo	CTR	MotID	Vmed	Imed	Vrms	Irms	P.apar	P.act	P.reac	fdp	THD v	THD i	Temp	Volt	Seq
hh:mm:ss	#	#	mV	mA	mV	mA	uVA	uW	uVAR	%	%	%	°C	mV	#
14:30:23	0	78	1536	1540	1537	1540	2366577.4	2343449.1	330053.3	99.0	197.63	273.02	27.5	2933	00015
14:30:25	0	23	1490	1491	1491	1492	2223966.0	2203783.1	298939.8	99.1	220.52	799.00	76.3	2532	00016
14:30:27	0	78	1536	1540	1537	1540	2366577.4	2343449.1	330053.3	99.0	191.84	139.16	27.3	2929	00017
14:30:30	0	23	1487	1488	1488	1489	2216725.2	2197281.5	292958.0	99.1	322.57	338.63	76.3	2532	00018
14:30:32	0	78	1526	1525	1528	1526	2331545.1	2311202.5	307319.8	99.1	871.21	1027.62	27.3	2929	00019
14:30:34	0	23	1487	1488	1488	1489	2216725.2	2197281.5	292958.0	99.1	406.00	314.76	76.3	2532	00020
14:30:36	0	78	1539	1537	1539	1538	2367038.3	2339147.6	362296.6	98.8	293.33	294.33	27.4	2933	00021
14:30:38	0	23	1497	1500	1497	1500	2245898.0	2222268.4	324931.8	98.9	441.13	383.10	76.3	2532	00022
14:30:40	0	78	1533	1528	1534	1529	2343971.1	2318648.5	343612.3	98.9	639.16	160.23	27.4	2933	00023
14:30:42	0	23	1503	1505	1503	1505	2262732.7	2238913.3	327454.5	98.9	87.53	631.85	76.9	2531	00024

La página se actualiza cada 4s. Si no ocurre esto, pulse aquí (También F5).

Utiliza funciones de *tcpip-toolbox (Peter Rydesäter, Mitthögskolan, Östersund, SWEDEN)*

Figura 4.26 Web de recogida de datos

Aspecto que presenta la página web usando el servidor en MATLAB. Se muestran los últimos datos llegados a MATLAB (a través de la pasarela). La actualización de la página es automática.

4.29.3 Envío de datos

Otra opción es la de enviar datos a los motes, pudiendo realizar acciones sobre ellos. Esta opción no es compatible con la de recepción. Aparece un cuadro de diálogo como el de la Figura 4.27. Los datos a enviar son iguales a los del paquete o estructura definida en la aplicación como `DissMsg`.

La primera vez aparecen unos datos predeterminados. Las siguientes veces aparecerán los últimos datos enviados. Si los valores son correctos (entran dentro de los rangos definidos) se procede al envío del mensaje hacia la pasarela (y desde ésta al nodo raíz de la red). En este caso, se debe realizar la operación contraria de cuando se leían los datos: hay que convertir el valor a grupos de 1 B para poder enviarlos.

Cuando se realice el envío, vuelve a aparecer el cuadro (pulsando sobre `Cancel` se finaliza el programa).

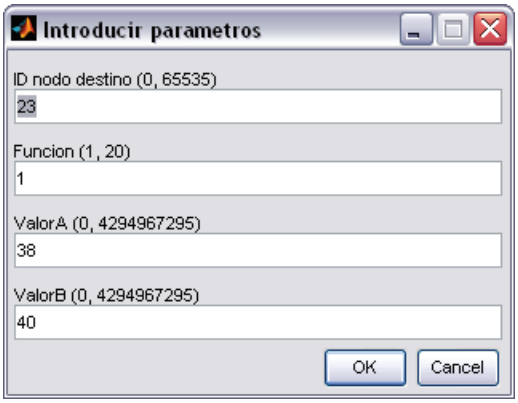


Figura 4.27 Cuadro de diálogo Introducir parámetros

Se piden los cuatro parámetros para enviar al nodo base de la red (y éste posteriormente al/a los nodo/s necesario/s). El primero es el nodo destino, por defecto está fijado el valor 65535 como difusión (llega a todos los nodos pertenecientes a la red actual); el segundo especifica qué función ejecuta el nodo remoto (activar/desactivar salida digital, definir nuevos umbrales...); los dos últimos son los dos valores de propósito general, cuyo significado depende de la función elegida.

4.29.4 Cierre de conexión

Esta opción finaliza la conexión activa y borra todas las variables que pudieran existir.