ANEXO A Legislación relativa a los contadores electrónicos

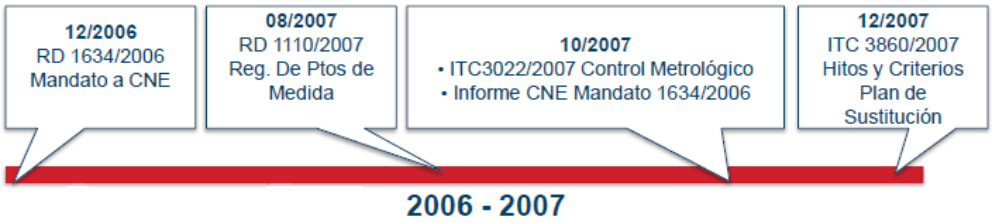

Fig. A-0-1 Cronología, Hitos y Hechos Relevantes 2006 y 2007

Fuente: Taller Contadores Inteligentes (Smart Metering) de Gas Natural Fenosa.

En el **Real Decreto (RD) 809/2006, de 30 Junio** (1), tenemos la primera referencia al contador electrónico. Este RD establece que los contadores eléctricos en nuevos suministros (potencia contratada <15 kW) deberán permitir la discriminación horaria de las medidas así como la telegestión.

Disposición adicional segunda. Instalación de equipos de medida:

"A partir del 1 de julio de 2007, los equipos de medida a instalar para nuevos suministros de energía eléctrica hasta una potencia contratada de 15 kW y los que se sustituyan para los antiguos suministros deberán permitir la discriminación horaria de las medidas así como la telegestión en los términos y condiciones técnicas que establezca el Ministerio de Industria, Turismo y comercio"

Después, en el **RD1634/2006,29 Diciembre**. Mandato a la CNE: Plan de instalación de equipos de medida.

Disposición adicional vigésima segunda. Plan de instalación de equipos de medida.

"Antes del 1 de julio de 2007, la Comisión Nacional de Energía remitirá a la Dirección General de Política Energética y Minas un informe donde se establezca un plan para la sustitución a nivel nacional de contadores que permitan la discriminación horaria de las medidas y la telegestión en todos los suministros de energía eléctrica hasta una potencia contratada de 15 kW.

En el mencionado plan se recogerán los criterios para la sustitución de dichos equipos de medida, así como el número de equipos a instalar anualmente, entendido como un porcentaje del total del parque nacional de contadores correspondientes a este tipo de suministros"

Real Decreto **RD 1110/2007, 24 Agosto**. Reglamento unificado de puntos de medida del sistema eléctrico español:

Establece toda la normativa relacionada con los equipos de medida (Normas generales, equipos de medida, verificación e inspección, sistemas y protocolos de comunicaciones,…).

Respecto a los equipos de medida, fija unas funciones mínimas que quedan resumidas en la siguiente tabla aunque puede consultarse una descripción detallada en los anexos.

FUNCIONES MÍNIMAS
• Lectura remota de la potencia activa, reactiva y máxima • Control de la potencia contratada • Programación remota de parámetros del contrato, como la potencia máxima • Interruptor integrado para operaciones remotas de conexión/desconexión • Lectura remota de parámetros de calidad de la red • Sincronización remota del tiempo • Capacidad de gestión de la carga, para reducir la demanda en momentos críticos • Seguridad y control de acceso a los datos • Alarma y registro de eventos

Tabla A.0.1 Resumen funciones mínimas de los contadores

ORDEN ITC/3022/2007,10 Octubre. Se regula el control metrológico del Estado sobre instrumentos de medida.

ORDEN ITC/3860/2007,28 Diciembre. Publicación de los criterios para el plan de sustitución de equipos de medida, donde se establece que todos los contadores deberán ser sustituidos antes de 31 de Diciembre de 2018 (11 años), cada distribuidor tiene que presentar su propio plan y presentar el diseño del sistema de telegestión, para ser aprobado por el Ministerio.

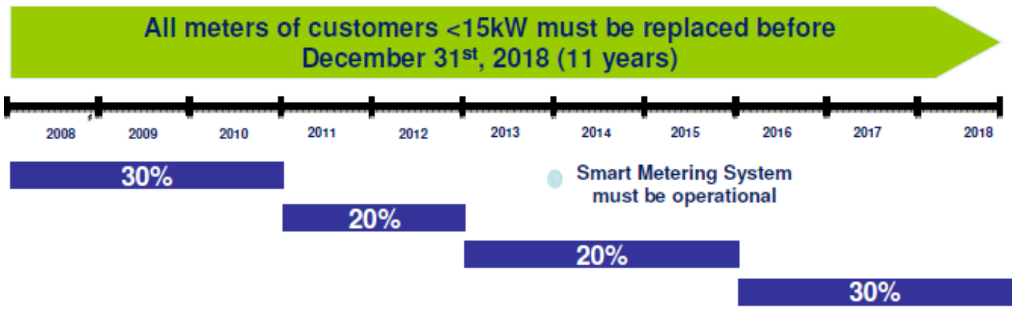

Fig. A-0-2 Plan de sustitución de equipos de medida

Fuente: Endesa's Smart Metering Roll-out Programme.

Principales características de los contadores según RD 1110/2007

Ampliación de características que deberán cumplir todos los equipos de medida.

MEDIDA:
Energía activa y energía reactiva Máxima potencia demandada (15') Discriminación horaria Capacidad para almacenar datos de 3 meses Capacidad para gestionar 6 periodos tarifarios >> Almacenar 3 facturas.
REGISTRO
Parámetros de calidad (Interrupciones de más de 3 minutos de duración y superación de los límites de tensión) Eventos (alarmas, cambios configuración de facturación, detección de fraude,…) Mostrar información al usuario
CONTROL DE LA POTENCIA
Limitador de la potencia contratada Interruptor de control de potencia Reconexión Manual o Automática
TELEMEDIDA Y TELEGESTIÓN. Requisitos obligatorios:
Medidas remotas de energía y potencia, correspondientes a los cierres de facturación. Lectura remota de parámetros de calidad. Modificación de la parametrización del equipo (tarifas, potencias contratadas, tipo de contrato,…) Sincronización remota (al menos una vez en cada ciclo de lectura). Actualización del software del equipo. Lectura remota de eventos. Corte y reconexión del suministro (gestión de altas/bajas y ejecución de planes de control de la demanda) Capacidad de gestión de cargas: reducción de la demanda en momentos críticos Capacidad para remitir mensajes al consumidor (consulta online de información)

Tabla A.0.2 Características de los contadores según RD

ANEXO B FUENTE DE ALIMENTACIÓN CAPACITIVA

1) Principio básico de funcionamiento

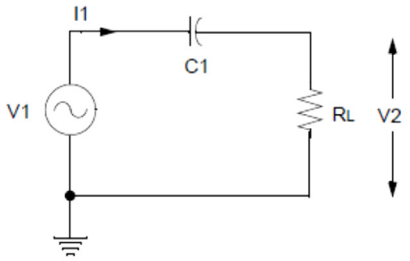

Fig. B-0-1 Principio funcionamiento FA capacitiva

Cuando se conectan en serie un condensador y una resistencia, una corriente constante puede ser mantenida a través de la resistencia mientras la capacitancia del condensador sea mucho mayor que la resistencia. Este valor de corriente dependerá del valor de C, R y de las tensiones de entrada y salida Vi e Vo. Asumiendo que Vi >> Vo lo cual se cumple 230V >> 5V, el valor aproximado de la corriente vendrá dado por:

$$I_{RMS} = \frac{V1}{Xc} = \frac{230V}{Xc}$$

$$Xc = \frac{1}{C\omega} = \frac{1}{C \times 2\pi f}$$

Después para conseguir una tensión continua (DC) en la carga añadiremos un par de rectificadores y condensadores.

Componente	Función
Condensador (C)	Sirve para atenuar la tensión de entrada
Resistencia (R)	Limita la corriente inicial de entrada
Diodos (D1 y D2)	Conforman un rectificador de media onda
Condensador C1	Filtrado y suministro de posibles picos de corriente
Resistencia Zener (Rz)	Para establecer la corriente min y max por el zener
Diodo Zener (Dz)	Fija la tensión de salida de la carga, Vo

Tabla B.0.1 Función componentes FA capacitiva

El voltaje en la carga permanecerá constante mientras la corriente de salida (Iout) sea menor o igual que la de entrada (Iin). La corriente de entrada Iin está limitada por R1 y la reactancia de C1 aunque principalmente por esta última dado su mayor valor.

Nota: R1 limita la corriente de entrada. El valor de R1 se selecciona de tal forma que no disipe demasiada potencia pero lo suficiente grande para limitar la corriente inicial (Inrush current) de entrada.

El problema de este tipo de FA es que la selección de los componentes es muy crítica, ya que al trabajar con tensiones de red, la potencia disipada resulta bastante elevada y

dado que la tensión de salida se regula con un diodo zener, la FA podrá dar corriente en un rango limitado (habrá un máximo y un mínimo de corriente posibles).

Por ello, este tipo de FA se diseña para alimentar un circuito específico y con variaciones de carga pequeñas ya que no permite grandes variaciones en la corriente que puede entregar por problemas de disipación de potencia.

2) Características de corriente de salida:

Primero hemos calculado el consumo estimado de todos los componentes del sistema, para a partir de ahí, diseñar la fuente de alimentación.

CONSUMOS		
	Modo ACTIVO	**BAJO consumo**
Micro XLP	23 mA	2 mA y menor
Cirrus CS5463	2,8 mA	1,93 mA
Módulo ZigBee	35,5 mA	1,5 uA
Relé	72 mA	-
TOTAL	**133,3 mA**	**4 mA**

Tabla B.0.2 Consumo del sistema

Quedando un consumo máximo aproximado de 133mA y mayorándolo un 10%, nos quedará un consumo de 160mA.

Entonces, el voltaje en la carga permanecerá constante mientras la corriente de entrada sea mayor de 150mA.

3) Precauciones respecto a los componentes:

Como ya se ha mencionado en las especificaciones, se debe conectar el sistema a través de un fusible para asegurar la protección de éste en caso de una conexión equivocada y también se incluye, para mejorar la protección global de la FA frente a picos de tensión, un varistor.

Respecto al condensador de AC y los diodos rectificadores, estos deben ser capaces de soportar la tensión de pico del sistema (325V).

4) Precauciones de manejo:

Se debe tener especial precaución ya que estamos trabajando con tensiones de red sin aislamiento. Respecto a ello tendremos que tomar las siguientes precauciones:

Se tiene que asegurar que los componentes no estén accesibles (puedan ser tocados) cuando el circuito esté en funcionamiento.

Diseño de la Fuente de Alimentación

Para el diseño de la FA se ha partido de los valores datasheet del CS5463 y de la nota de aplicación de Microchip, sobre diferentes topologías de FA sin transformador (1). Sin embargo, se ha teniendo en cuenta que nuestra fuente deberá proporcionar una corriente tres veces mayor que la calculada en dichos documentos, dificultando en gran medida el diseño de ésta.

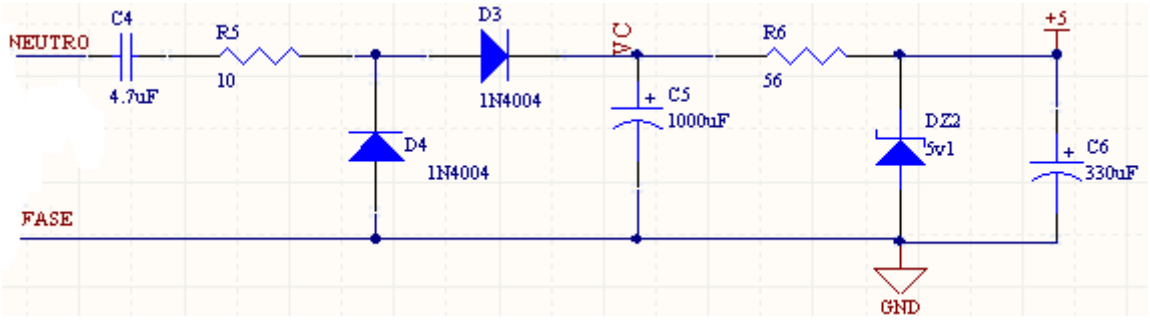

Fig. B-0-2Diseño FA capacitiva

1) Cálculo de C1

Seleccionamos **Vz = 5V1**

$$V_{ef}(media\_onda) = \frac{Vpico}{2} = \frac{230\sqrt{2}}{2} \cong 160V$$

$$I_{IN} = \frac{V_{ef}(media\_onda)}{Xc_1 + R_1} \geq I_{OUT} \quad \rightarrow \quad Xc_1 \leq \frac{V_{ef}(media\_onda)}{I_{OUT}} - R_1$$

Seleccionamos:
R1=10 Ω
Iout=160mA

$$Xc_1 \leq \frac{160V}{160mA} - 10\Omega = 990\Omega$$

$$Xc_1 = \frac{1}{c_1\omega} = \frac{1}{c_1 \times 2\pi f} \quad \rightarrow$$

$$c_1 \geq \frac{1}{Xc_1 \times 2\pi f} = \frac{1}{990\Omega \times 2\pi 50Hz} \cong 3,22\mu F$$

Considerando la tolerancia del condensador y la resistencia, dado que el valor de $c_1 = 3,3\mu F$ quedaría muy justo:

Seleccionamos $c_1 = 4,7\mu F$

2) **<u>Cálculo de C2:</u>** Ahora para el cálculo de C2 consideramos una descarga de C2 de 1V para que nos quede de un tamaño que no sea excesivamente grande.

$$c_2 \geq \frac{I_{descargaMAX} \times t_{descarga}}{\Delta V c_2} \cong \frac{160mA \times 10msg}{1V} \cong 1600\mu F$$

Seleccionamos: $c_2 = 2200\mu F$

3) **<u>Cálculo de Rz:</u>** Le dejamos a Rz un par de voltios por lo que:

$$\Delta V_{Rz} = 2V$$

$$(I_{Rz})_{\min} > (I_{OUT})_{\max} + (I_z)_{\min} \qquad \rightarrow \qquad \frac{2V}{Rz} > 160mA + 1mA$$

$$Rz < \frac{2V}{161mA} \cong 12,4\Omega$$

Seleccionamos: $Rz = 10\Omega$

5) **<u>Cálculo de C3</u>**

El condensador C3 está para proporcionar los posibles picos de corriente necesarios para el circuito, por lo que lo seleccionaremos de un valor menor que C2 para que sea rápido pero que no se descargue demasiado.

Seleccionamos $c_3 = 330\mu F$

6) **<u>Disipación de potencia en los componentes:</u>**

La impedancia de entrada al circuito será:

$$Xc_1 = \frac{1}{4,7\mu F \times 2\pi f} \cong 677\Omega \qquad R_1 = 10\Omega$$

Por lo que la corriente vendrá dada por:

$$Iin \leq \frac{230V}{687\Omega} \cong 335mA$$

- <u>Potencia en R1</u>

$$P_{R1} = I_{IN}^2 \times R_1 = (335mA)^2 \times 10\Omega = 1,12W$$

- <u>Potencia en D1 y D2</u>

$$P_{R1} = I_{OUT} \times V_{Diodo} = 160mA \times 1V = 160mW$$

- Potencia en Diodo Zener

$$P_{R1} = I_{OUT} \times V_Z = 160mA \times 5,1V = 0,816W$$

(Potencia máxima en el Zener si la carga se queda en circuito abierto)

- Potencia en Rz

$$P_{R1} = I_{OUT}^2 \times R_1 = (160mA)^2 \times 10\Omega = 0,256W$$

Nota: Aproximadamente, se han seleccionado los componentes doblando el valor de potencia calculado.

Diseño final, con elementos de protección (fusible y varistor) incluidos:

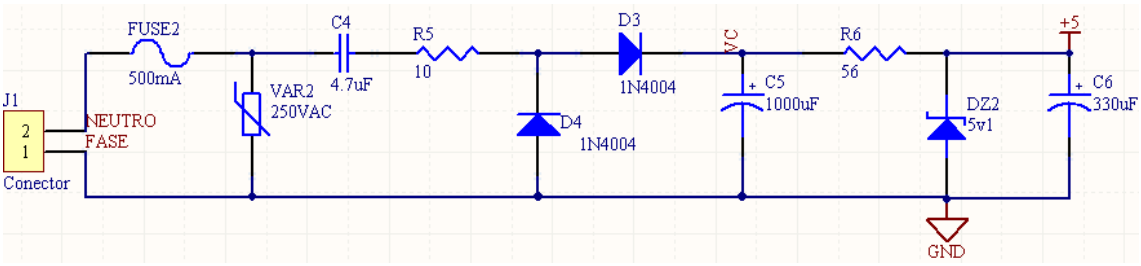

Fig. B-3 Final FA capacitiva

Ref. Farnell	Descripción	Qty.	Precio unitario	Total €
1357898	RESISTENCIA, 3W 5% 10R	2	0,36 €	0,72 €
1781899	CONDENSADOR, CLASE X2, 4,7 UF, 305 VCA	1	4,39 €	4,39 €
1843708	DIODO, RECTIFICADOR,, 400V, 1A, DO-41	2	0,05 €	0,10 €
1469419	1N4733A-TR - DIODO, ZENER, 5,1V, 1,3 W	1	0,05 €	0,05 €
1057192	VARISTOR, 21.0J, 250VCA	1	0,30 €	0,30 €
1822618	CAPACITOR ALUM ELEC, 1000UF, 50V	1	1,00 €	1,00 €
9451102	CONDENSADOR, 330 uF, 16V	1	0,07 €	0,07 €
9762540	TC1185-3.3VCT713 - REG. DE TENSIÓN LDO +3,3 V, SOT-23A-5	1	0,47 €	0,47 €
1123206	MCF06G-500MA - FUSIBLE 500MA	1	0,11 €	0,11 €
1705671	1N5357BRLG - ZENER DIODE, 5W, 20V	1	0,18 €	0,18 €
1737719	RESISTOR, METAL FILM, 56OHM, 3W, 5%	1	0,17 €	0,17 €
				7,57 €

Tabla B.3 Componentes fuente alimentación capacitiva

ANEXO C Cirrus CS5463

Métodos numéricos de cálculo

Como se ha comentado al principio de éste apartado, en el datasheet del CS5463 el esquema de cálculo se encuentra explicado muy visualmente, por ello se mostrará ahora, para dar una visión general muy clara de todo el proceso de cálculo. Primero se definirán todas las magnitudes que aparecen en la figura.

Magnitud	Significado
V*, I*	Tensión y corriente instantánea
Vacoff*, Iacoff*	Tensión y corriente de offset de AC
Vrms*, Irms*	Tensión y corriente eficaz (RMS)
S*	Potencia Aparente
P*	Potencia instantánea
Poff*	Offset para la potencia
Pactive*	Potencia Activa
PF*	Factor de potencia
Qtrig*	Potencia Reactiva del Triangulo de potencias
Q*	Potencia Reactiva instantánea
Qavg*	Potencia Reactiva media (AVG)
PulseRate*	Frecuencia de los pulsos de salida

Tabla C.1 Definición de magnitudes

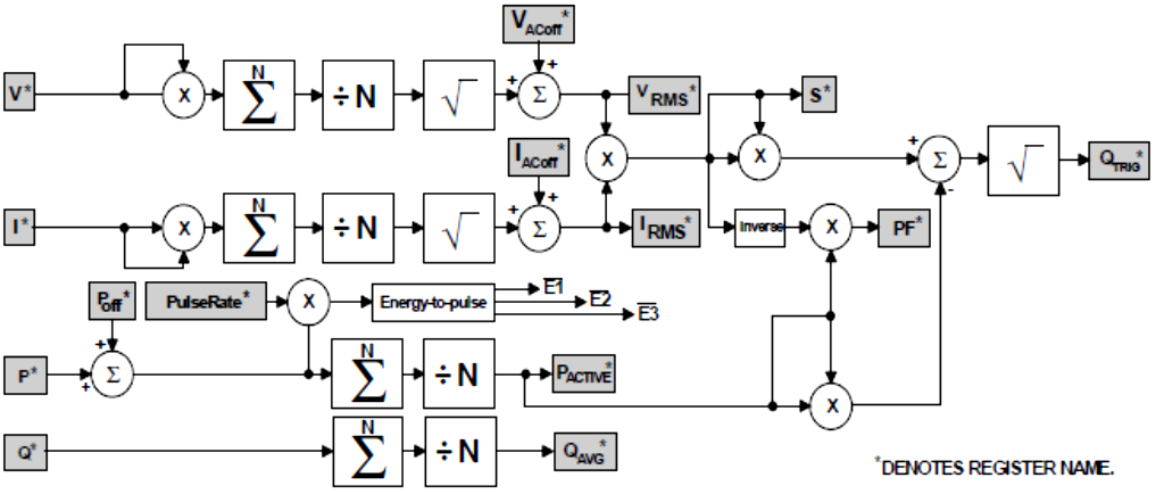

Fig. C-0-1 Data Measurement Flow Diagram.

Lista de registros del Cirrus

Registros implementados para utilizar con la función cs_readRegister (addrees,page), donde addrees representa la dirección del registro en hexadecimal y page la página en la que se encuentra.

REGISTER	ADDRESS
CS_CONFIGURATION	0x00
CS_CURRENT_DC_OFFSET	0x02
CS_CURRENT_GAIN	0x04
CS_VOLTAGE_DC_OFFSET	0x06
CS_VOLTAGE_GAIN	0x08
CS_CYCLE_COUNT	0x0A
CS_PULSE_RATE_E	0x0C
CS_INSTANTANEOUS_CURRENT	0x0E
CS_INSTANTANEOUS_VOLTAGE	0x10
CS_INSTANTANEOUS_POWER	0x12
CS_ACTIVE_POWER	0x14
CS_RMS_CURRENT	0x16
CS_RMS_VOLTAGE	0x18
CS_RATIO	0x1A
CS_POWER_OFFSET	0x1C
CS_STATUS	0x1E
CS_CURRENT_AC_OFFSET	0x20
CS_VOLTAGE_AC_OFFSET	0x22
CS_OPERATION_MODE	0x24
CS_TEMPERATURE	0x26
CS_AVERAGE_REACTIVE_POWER	0x28
CS_INST_REACTIVE_POWER	0x2A
CS_PEAK_CURRENT	0x2C
CS_PEAK_VOLTAGE	0x2E
CS_REACTIVE_POWER	0x30
CS_POWER_FACTOR	0x32
CS_INTERRUPT_MASK	0x34
CS_APPARENT_POWER	0x36
CS_CONTROL	0x38
CS_HARMONIC_ACTIVE_POWER	0x3A
CS_FUNDAMENTAL_ACTIVE_POWER	0x3C
CS_FUNDAMENTAL_REACTIVE	0x3E
CS_PAGE	0x3E

Tabla C.2 Lista registros page 0

- **cs_writeRegister:** Este es el diagrama de flujo de otra de las funciones importantes implementadas. Es similar a la **cs_ReadRegister**, pero para escribir cualquier registro del Cirrus, pasándole como argumento el número de registro, la página en que se encuentra y los datos a escribir.

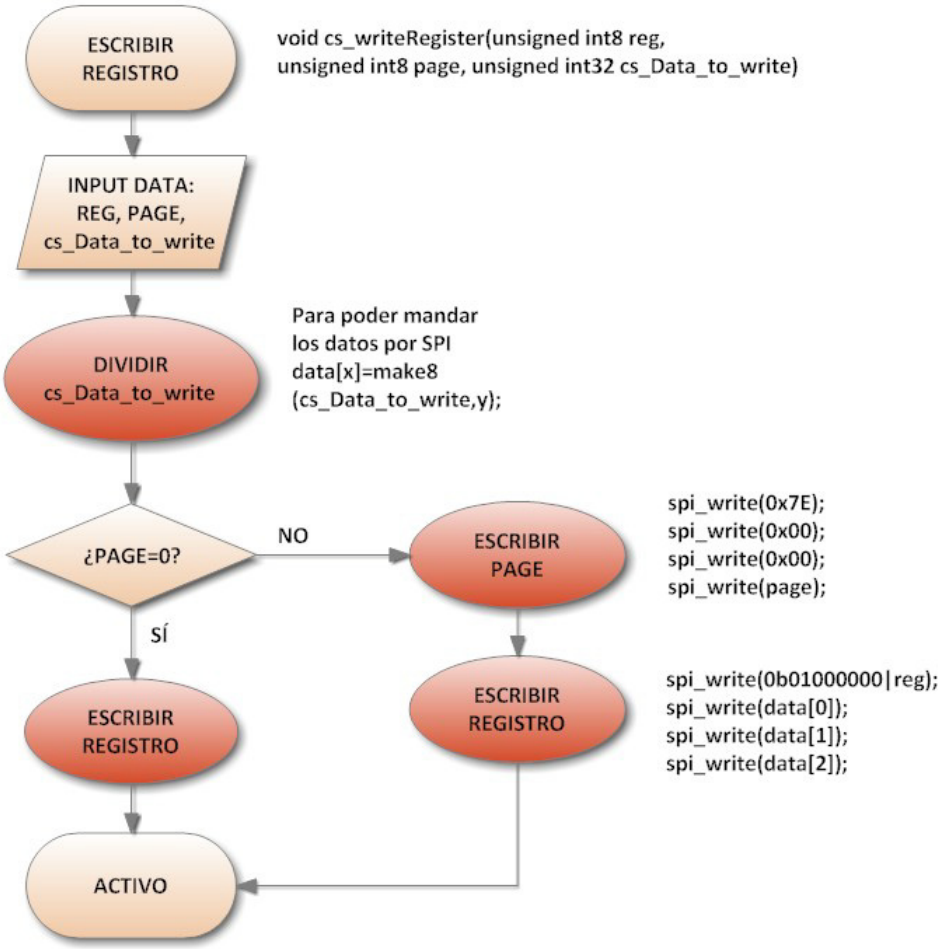

Fig. 0-2 Diagrama flujo cs_writeRegister

ANEXO D Librería ZigBee

Tabla que incluye todas las funciones implementadas por HOWLab para el manejo del ZigBee.

void	**ZB_GeTPowerLevel** (void) Ask for the supply voltage of the device See pag 72 of the AT-Command 3.02 of Telegesis.
void	**ZB_NETDisassociate** (void) Disassociate Local Device From PAN See pag 17 of the AT-Command 3.02 of Telegesis.
void	**ZB_NETJoin** (void) Join Network The local node scans all channels selected in register S00 for the existence of a PAN. When finding any PAN which allows joining it will automatically join in via the remote node with the highest RSSI. See page 16 of the AT-Command 3.02 of Telegesis.
void	**ZB_SSink** (void) Search For A Sink Search for a sink on the network by sending a broadcast causing all sinks to reply. See page 35 of the AT-Command 3.02 of Telegesis.
void	**ZB_SendToSinkD** (int8 nDatos, int8 *Datos) Send data in binary to the Sink.
void	**ZB_StartUpNet** (void) Connet the Zigbee device to the preconfigured network.
void	**ZB_TGBootloader_DefaultConfig** (void) Function to configure the Zigbee Preferred PanID and Password
void	**ZB_TGBootloader** (void) Function to configure the Zigbee device the first time. Today the content of the register must be change by hand in order to fix the correct parameters
void	**ZB_PowerMode** (int8 PowerMode) Change the power level.
void	**ZB_PowerConf** (int8 PowerMode) Configure the Zigbee device in order to use a power level.

ANEXO E Comandos PC >> Nodo implementados

Ahora se va a mostrar la forma en que se ejecutan los comandos, para dar una visión general del funcionamiento de éstos, se puede observar la siguiente imagen, la cual ya ha sido mostrada en apartados anteriores:

COMANDO TIPO	
Comando a ejecutar	***Respuesta***
AT+UCASTB:XX,<address> <end_point> <cluster_id> <length> <command>	<comando ejecutado> SEQ:XX OK o ERROR:<errorcode> ACK:XX o NACK:XX ***Respuesta del comando*** <Según el comando ejecutado>

Comando	Función implementada	Descripción	Longitud
00	readRegister	Lee el registro indicado	2
01	writeRegister	Escribe el registro indicado	3
02	SoftwareReset	Resetea el Cirrus	1
03	PowerState	Modifica el estado del Cirrus	2
04	Initialization	Inicializa los registros del Cirrus	1
05	WriteCalibrationData	Escribe los datos de calibración	1
06	StartConversions	Empezar a tomar medidas	1
07	ReadInstantMeasures	Lee variables instantáneas	1
08	ReadPeakMeasures	Lee variables de pico	1
09	ReadAverageMeasures	Lee variables medias, AVG	1
0A	ReadRMSmeasures	Lee variables eficaces, RMS	1
0B	ReadCalibrations	Lee los registros de calibración	1
0C	SystemCalibration	Inicia la calibración indicada	2
0D	OffsetCalibration	Inicia la calibración de offset	1
0E	GainCalibration	Inicia la calibración de ganacia	1
0F	cs_5463Status	Comprueba el estado del Cirrus	1
20	ReadPage0	Lee registros de la página 0	1
21	ReadPage1	Lee registros de la página 1	1
22	ReadPage3	Lee registros de la página 3	1
23	SetSendTime	Fija el tiempo de lectura periódica	2
30	Control_Rele	Controla el estado del relé	1

Tabla 0.1 Resumen de comandos

A partir de este punto, se explicarán en detalle los comandos de la tabla anterior, teniendo en cuenta que para simplificar la interpretación de los mismos, se sustituirá el <end_point><cluster_id> por <10F1>, que son los números que se han asignado.

ReadRegister – Leemos el registro indicado en la variable DATA	
Comando a ejecutar	***Respuesta***
AT+UCASTB:05,<address> <10F1> <02> <00> <data>	<comando ejecutado> SEQ:XX OK or ERROR:<errorcode> ACK:XX o NACK:XX
<data> registro[1byte] del 0x00-0x30 Número asignado al registro, ver primera tabla del anexo E.	***Respuesta del comando*** <datos del registro leído> [HEX]
Ejemplo de comando: 10F102000A	***Ejemplo de respuesta:*** *000FA0*

WriteRegister – Escribimos el registro indicado en la variable DATA	
Comando a ejecutar	***Respuesta***
AT+UCASTB:08,<address> <10F1> <04> <01> <data>	<comando ejecutado> SEQ:XX OK or ERROR:<errorcode> ACK:XX o NACK:XX
<data> = registro[1] + datos[3] Número del registro (Tabla anexo E) y datos que se quieran escribir en él.	
Ejemplo de comando: 10F1040104400000	***Ejemplo de respuesta:*** <EVENT ACK> [ASCII]

SoftwareReset – Resetea el Cirrus	
Comando a ejecutar	***Respuesta***
AT+UCASTB:04,<address> <10F1> <01> **<02>**	<comando ejecutado> SEQ:XX OK or ERROR:<errorcode>
<02> = Comando asignado a SoftwareReset	ACK:XX o NACK:XX ***Respuesta del comando***
Ejemplo de comando: 10F10102	<EVENT_ACK> [ASCII]

PowerState – Modifica el estado del Cirrus	
Comando a ejecutar AT+UCASTB:05,<address> <10F1> <01> **<03>** <data> **<data>** Selecciona el modo de energía **01=**Standby **10=**Sleep **11=**Power up	***Respuesta*** <comando ejecutado> SEQ:XX OK or ERROR:<errorcode> ACK:XX o NACK:XX ***Respuesta del comando*** <Stanby/Sleep/PowerUP> [ASCII]

Initialization – Inicializa los registros del Cirrus	
Comando a ejecutar AT+UCASTB:04,<address> <10F1> <01> <04> **<04>** = Comando asignado a Initialization ***Ejemplo de comando:*** 10F10104	***Respuesta*** <comando ejecutado> SEQ:XX OK or ERROR:<errorcode> ACK:XX o NACK:XX ***Respuesta del comando*** <Cirrus OK> [ASCII]

WriteCalibrationData – Escribe los datos de calibración	
Comando a ejecutar AT+UCASTB:04,<address> <10F1> <01> <05> **<05>** = Comando asignado a WriteCalibrationData ***Ejemplo de comando:*** 10F10105	***Respuesta*** <comando ejecutado> SEQ:XX OK or ERROR:<errorcode> ACK:XX o NACK:XX ***Respuesta del comando*** <Writing calibration data> [ASCII]

StartConversions – Comando para empezar a tomar medidas	
Comando a ejecutar	***Respuesta***
AT+UCASTB:04,<address> <10F1> <01> <06> **<06>** = Comando asignado a StartConversions ***Ejemplo de comando:*** 10F10106	<comando ejecutado> SEQ:XX OK or ERROR:<errorcode> ACK:XX o NACK:XX ***Respuesta del comando*** <Ninguna>

ReadInstantaneousmeasures – Lee valores instantáneos	
Comando a ejecutar	***Respuesta***
AT+UCASTB:04,<address> <10F1> <01> <07> ***Ejemplo comando:*** 10F10107 ***Ejemplo de respuesta:*** Instantaneous Current 123.5 Instantaneous Voltage 327.1 Instantaneous Power 56.2 Inst Reactive Power 23.8	<comando ejecutado> SEQ:XX OK or ERROR:<errorcode> ACK:XX o NACK:XX ***Respuesta del comando*** < Instantaneous Current > [ASCII] <datos leídos I> [mA] < Instantaneous Voltage> [ASCII] < datos leídos V> [V] < Instantaneous Power> [ASCII] < datos leídos P> [W] < Inst Reactive Power > [ASCII] < datos leídos P> [VAr]

ReadPeakMeasures – Lee los valores de pico de tensión y corriente	
Comando a ejecutar	***Respuesta***
AT+UCASTB:04,<address> <10F1> <01> <08> ***Ejemplo comando:*** 10F10108 ***Ejemplo de respuesta:*** PEAK Current 983.7 PEAK Voltage 325.3	<comando ejecutado> SEQ:XX OK or ERROR:<errorcode> ACK:XX o NACK:XX ***Respuesta del comando*** <Peak Current> [ASCII] <datos leídos del registro Ipeak> [mA] <Peak Voltage> [ASCII] < datos leídos del registro Vpeak > [V]

ReadAVGmeasures – Lee los valores medios de tensión y corriente	
Comando a ejecutar	***Respuesta***
AT+UCASTB:04,<address> <10F1> <01> <09> ***Ejemplo comando:*** 10F10109 ***Ejemplo de respuesta:*** AVG Current 583.1 AVG Voltage 207.07	<comando ejecutado> SEQ:XX OK or ERROR:<errorcode> ACK:XX o NACK:XX ***Respuesta del comando*** <AVG Current> [ASCII] <datos leídos del registro Iavg> [mA] <AVG Voltage> [ASCII] < datos leídos del registro Vavg > [V]
ReadRMSmeasures – Lee los valores eficaces de tensión y corriente	
Comando a ejecutar	***Respuesta***
AT+UCASTB:04,<address> <10F1> <01> <0A> ***Ejemplo comando:*** 10F1010A ***Ejemplo de respuesta:*** RMS Current 783.1 RMS Voltage 229.6	<comando ejecutado> SEQ:XX OK or ERROR:<errorcode> ACK:XX o NACK:XX ***Respuesta del comando*** <RMS Current> [ASCII] <datos leídos del registro Irms> <RMS Voltage> [ASCII] < datos leídos del registro Vrms >

cs_5463Status – Comprueba el estado del Cirrus	
Comando a ejecutar	***Respuesta***
AT+UCASTB:04,<address> <10F1> <01> <0F> **<0F>** = Comando asignado a ReadPage0 ***Ejemplo de comando:*** 10F1010F	<comando ejecutado> SEQ:XX OK or ERROR:<errorcode> ACK:XX o NACK:XX ***Respuesta del comando*** Manda el valor del registro de estado [ASCII]

ReadPage0 – Lee registros de la página 0	
Comando a ejecutar	***Respuesta***
AT+UCASTB:04,<address> <10F1> <01> <20>	<comando ejecutado> SEQ:XX OK or ERROR:<errorcode>
<20> = Comando asignado a ReadPage0	ACK:XX o NACK:XX
Ejemplo de comando: 10F10120	***Respuesta del comando*** Manda los datos leídos de todos los registros de la página 0 [HEX]

ReadPage1 – Lee registros de la página 1	
Comando a ejecutar	***Respuesta***
AT+UCASTB:04,<address> <10F1> <01> <21>	<comando ejecutado> SEQ:XX OK or ERROR:<errorcode>
<21> = Comando asignado a ReadPage1	ACK:XX o NACK:XX
Ejemplo de comando: 10F10121	***Respuesta del comando*** Manda los datos leídos de los registros: "Energy Pulse Output Width" [HEX] "No Load Threshold" [HEX] "Temperature Sensor Gain" [HEX] "Temperature Sensor Offset" [HEX]

ReadPage3 – Lee registros de la página 3	
Comando a ejecutar	***Respuesta***
AT+UCASTB:04,<address> <10F1> <01> <22>	<comando ejecutado> SEQ:XX OK or ERROR:<errorcode>
<22> = Comando asignado a ReadPage3	ACK:XX o NACK:XX
Ejemplo de comando: 10F10122	***Respuesta del comando*** Manda los datos leídos de los registros: "Voltage sample interval" [HEX] "Voltage sag level" [HEX] "I fault sample interval" [HEX] "Current fault level" [HEX]

SetSendTime – Fija el tiempo de lectura periódica	
Comando a ejecutar	***Respuesta***
AT+UCASTB:04,<address> <10F1> <01> <23> <data> **<data>** Selecciona tiempo **00=**15 segundos **01=**30 segundos **10=**1 minuto **11=**desactivado	<comando ejecutado> SEQ:XX OK or ERROR:<errorcode> ACK:XX o NACK:XX ***Respuesta del comando*** <Tiempo establecido> [ASCII]

Control_Rele – Controla el estado del relé	
Comando a ejecutar	***Respuesta***
AT+UCASTB:04,<address> <10F1> <01> <30> **<30>** = Comando asignado a Control_Rele ***Ejemplo de commando:*** 10F10130	<comando ejecutado> SEQ:XX OK or ERROR:<errorcode> ACK:XX o NACK:XX ***Respuesta del comando*** <CARGA conectada> o [ASCII] <CARGA desconectada> [ASCII]

END_POINT	CLUSTER_ID	LENGTH	COMMAND	DATA	RESULT
0x10	0xF1	0x02	0x00	0x16	0x10F1020016
0x10	0xF1	0x01	0x02	XX	0x10F10102

Ejemplo comandos PC>>Nodo

1) Comando 0x00 por lo que utiliza la función cs_readRegister (Viendo la tabla de abajo), con el argumento 0x16, que hace referencia a un registro, en concreto el CS_RMS_CURRENT (según la tabla que se puede consultar en los anexos). En resumen, al enviar por ZigBee el comando 0x10F1020016, leeríamos el registro que guarda el valor medido de Irms.

2) Comando 0x02, que como observamos en la tabla de abajo implementa el comando cs_SoftwareReset, que resetea el CS5463.

ANEXO F CÓDIGO FIRMWARE

Este anexo recoge todo el código del proyecto desarrollado por el autor, es decir, las librerías del Cirrus CS5463, las funciones inteligentes y el protocolo de comunicaciones. También puede consultarse el código y la documentación creada por Doxygen, en el CD anexo.

Código librería Cirrus CS5463

```
//*****************************************************************
***************
/*! \file CS_5463.c
*  \brief The next file provides the functions needed in order to
handle Cirrus CS5463.
*
* Description:
*
* Author: Nestor Carruesco (ncarruesco@gmail.com)\n
* Copyright: HowLab UNIVERSITY OF ZARAGOZA (Spain)\n
* Date: 19/07/2011\n
* Update record:\n
* _____________________________________________________\n
* Author: Nestor Carruesco (ncarruesco@gmail.com)\n
* Update: Añadidas varias funciones:
*                 - Calibraciones
*                 - cs_5463Status
* _____________________________________________________\n
*/

#include "CS_5463.h"
```

- **Funciones Capa 0**

```
/*! \fn void cs_readRegister(unsigned int8 reg, unsigned int8 page)
 *  \brief Reads the register indicated in reg. See pag 24 of CS5463
datasheet
 *  \param reg Indicates register to read from
 *  \param page Indicates page to read from
 *  \return Data read from register */

void cs_readRegister(unsigned int8 reg,unsigned int8 page)
{
   if                    (page                    ==                    0)
//Comprobamos la page (0,1,3)
   {
         spi_write(reg);                                                 //
solicita el valor del registro
         cs_AppData[0] = spi_read(0xFF);     //MSB
         cs_AppData[1] = spi_read(0xFF);      //Escribir 0xFF para que
el micro genere la señal SCLK
         cs_AppData[2] = spi_read(0xFF);    //write  0xFF  to  SPI  the
same time you are reading a value.

         cs_numByteAppData = 3;
```

```
         //Mandamos por ZigBee los datos del reg leido
         ZB_SendToSinkD(cs_numByteAppData,cs_AppData);
//cs_AppData=puntero a la variable

         //Guardamos el nombre del registro a leer, en el array
nombre_registro
         unsigned int8 numero_registro;
         numero_registro=reg;
         numero_registro=(numero_registro>>1);  //Desplazamos los bits
a la derecha para quitar el 0

         char nombre_registro[30];
         unsigned int8 j=0;
         for(j=0;j<30;j++)
            {

nombre_registro[j]=matriz_registros[numero_registro][j];
            }
         ZB_SendToSinkD(STRLEN(nombre_registro),nombre_registro);
   }
   else
   {
         //(0x7E) Lee de PAGE, la pagina en la que queremos leer (por
defecto 0)
         spi_write(0x7E);     //spi_write(0b01000000||CS_PAGE);
         spi_write(0x00);
         spi_write(0x00);
         spi_write(page);

         spi_write(reg);                        // solicita el valor del
registro
         cs_AppData[0] = spi_read(0xFF);
         cs_AppData[1] = spi_read(0xFF);
         cs_AppData[2] = spi_read(0xFF);
         cs_numByteAppData = 3;

         ZB_SendToSinkD(cs_numByteAppData,cs_AppData);  //Mandar datos
reg leido por ZigBee

            spi_write(0x7E);        //Escribimos page=0 (Configuracion
por defecto)
            spi_write(0x00);
            spi_write(0x00);
            spi_write(0);                  //page
   }
}


/*! \fn void cs_writeRegister(unsigned int8 reg,unsigned int8 page,
unsigned int32 cs_Data_to_write)
* \brief Writes the register indicated in address. See pag 24 of
CS5463 datasheet
* \param reg Indicates register to write
* \param page Indicates register to write
* \param cs_Data_to_write Pointer to a byte array with the data to
write in the register */

void cs_writeRegister(unsigned int8 reg,unsigned int8 page, unsigned
int32 cs_Data_to_write)
{
```

```
   //Dividimos int32 cs_Data_to_write en 3 int8 para poder mandarlas
por SPI
   unsigned int8 data[3];
   data[0]=make8(cs_Data_to_write,2);
   data[1]=make8(cs_Data_to_write,1);
   data[2]=make8(cs_Data_to_write,0);

   if (page == 0)
   {
         spi_write(0b01000000|reg);           //Solicita el valor del
registro
         spi_write(data[0]);                        //parámetros
         spi_write(data[1]);
         spi_write(data[2]);
   }
   else
   {
         spi_write(0b01000000|CS_PAGE);    //Página en la que queremos
escribir (por defecto 0)
         spi_write(0x00);
         spi_write(0x00);
         spi_write(page);

         spi_write(0b01000000|reg);          // solicita el valor del
registro
         spi_write(data[0]);                        // parametros
         spi_write(data[1]);
         spi_write(data[2]);
   }
}


/*! \fn cs_SoftwareReset (void)
    \brief After a Software Reset, the internal registers will be
reset to their default values.
    It is a command. See pag 20 of CS5463 datasheet */

void cs_SoftwareReset(void)
{
      spi_write(0b10000000);     //0x80
      delay_ms(200);
}


/*! \fn cs_StartConversions (void)
    \brief Initiates CS5463 continuous computation cycles
    It is a command. See pag 23 of CS5463 datasheet
*/
void cs_StartConversions(void)
{
      spi_write(0xE8);              //0b11101000  Perform  continuous
computation cycles
}


/*! \fn cs_PowerState (unsigned int8 cs_PowerMode)
    \brief Initiates CS5463 continuous computation cycles
    01 = Halt and enter stand-by power saving state. This state allows
quick power-on
    10 = Halt and enter sleep power saving state.
    It is a command. See pag 23 of CS5463 datasheet
```

```
*/

void cs_PowerState(unsigned int8 cs_PowerMode)
{
      switch (cs_PowerMode)
      {
      case 1: spi_write(0b10001000); //01 = Halt and enter stand-by
            break;
      case 2: spi_write(0b10010000); //10 = Halt and enter sleep.
            break;
      case 3: spi_write(0b10100000); //11 = PowerUP --> Will initiate
a power on reset.
            break;                                                  //If the
part is already powered-on, all computations will be halted.
      }
}
```

- **Capa 1: Funciones de inicialización y configuración**

```
/*! \fn void cs_Initialization (void)
    \brief Configurates Cirrus CS5463 registers
    See pages 23-36 of CS5463 datasheet
*/
void cs_Initialization(void)
{
   //Software reset (Pag23)
      cs_SoftwareReset();

   //Configuration Register
      cs_writeRegister(CS_CONFIGURATION,0,0x000001);

   //Cycle Count Register
      cs_writeRegister(CS_CYCLE_COUNT,0,0x000000);

   //Operational Model Register
      cs_writeRegister(CS_OPERATION_MODE,0,0x000FC0);

   //Control Register
      cs_writeRegister(CS_CONTROL,0,0x000000);
}


/*! \fn void cs_WriteCalibrationData (void)
    \brief Calibrates the cirrus CS5463
    See pag 25 of CS5463 datasheet */

void cs_WriteCalibrationData(void)
{
   //Current DC Offset Register
      //cs_writeRegister(CS_CURRENT_DC_OFFSET,0,0x800001);       //  NO
utilizar si IHPF

   //Voltage DC Offset Register
      //cs_writeRegister(CS_VOLTAGE_DC_OFFSET,0,0x9F2EFA);       //  NO
utilizar si VHPF

   //Current Gain Register
      //cs_writeRegister(CS_CURRENT_GAIN,0,0x410000);

   //Voltage Gain Register
      cs_writeRegister(CS_VOLTAGE_GAIN,0,0x500000);
```

```
   //Power Offset Register
      //cs_writeRegister(CS_POWER_OFFSET,0,0x000000);

   //Current AC Offset Register
      //cs_writeRegister(CS_CURRENT_AC_OFFSET,0,0xFFE000);
//GAIN=0x ff e0 00

   //Voltage AC Offset Register
      //cs_writeRegister(CS_VOLTAGE_AC_OFFSET,0,0xFFC76F);      //GAIN=
ff c7 6f

strcopy (DataOut,"Writing Calibration Data");
   ZB_SendToSinkD(STRLEN(DataOut),DataOut);
}
```

- **Capa 1: Funciones de lectura de variables**

```
/*! \fn void cs_ReadInstantaneousMeasures (void)
    \brief Reads instantaneous measures registers (I,V,P,Q)
      See pag 28 of CS5463 datasheet
*/
void cs_ReadInstantaneousMeasures(void)
{
   //Instantaneous Current Register
      cs_readRegister(CS_INSTANTANEOUS_CURRENT ,0);
      SignedDataConversion(cs_AppData,0);//0=Current 1=Signed Measure

   //Instantaneous Voltage Register
      cs_readRegister(CS_INSTANTANEOUS_VOLTAGE,0);
      SignedDataConversion(cs_AppData,1);

   //Instantaneous Power Register
      cs_readRegister(CS_INSTANTANEOUS_POWER,0);
      SignedDataConversion(cs_AppData,3);

   //Instantaneous Reactive Power Register
      cs_readRegister(CS_INST_REACTIVE_POWER,0);
      SignedDataConversion(cs_AppData,3);
}


/*! \fn void cs_ReadRMSmeasures (void)
    \brief Reads RMS measures register (Irms,Vrms)
    See pag 28 of CS5463 datasheet
*/
void cs_ReadRMSmeasures(void)
{
   //RMS Current Register
      cs_readRegister(CS_RMS_CURRENT,0); //0=Page
        DataConversion(cs_AppData,0);                //0=Current

   //RMS Voltage Register
      cs_readRegister(CS_RMS_VOLTAGE,0); //0=Page
        DataConversion(cs_AppData,1);                //1=Voltage
}
```

```
/*! \fn void cs_ReadPeakMeasures (void)
    \brief Reads PEAK measures registers (I_peak,V_peak)
    See pag 31 of CS5463 datasheet
*/
void cs_ReadPeakMeasures(void)
{
   //Peak Current Register
      cs_readRegister(CS_PEAK_CURRENT,0);
        SignedDataConversion(cs_AppData,0);     //0=Current

   //Peak Voltage Register
      cs_readRegister(CS_PEAK_VOLTAGE,0);
        SignedDataConversion(cs_AppData,1);
}

/*! \fn void cs_ReadAverageMeasures (void)
    \brief Reads average(AVG) measures registers (P_avg,Q_avg,s)
    See CS5463 datasheet
*/
void cs_ReadAverageMeasures(void)
{
   //Active Power Register
      cs_readRegister(CS_ACTIVE_POWER,0);
      SignedDataConversion(cs_AppData,4);

   //AVG Reactive Power Register
      cs_readRegister(CS_AVERAGE_REACTIVE_POWER,0);
      SignedDataConversion(cs_AppData,3);

   //Apparent Power Register
      cs_readRegister(CS_APPARENT_POWER,0);
      DataConversion(cs_AppData,4);

   //Power Factor Register
      cs_readRegister(CS_POWER_FACTOR,0);
      SignedDataConversion(cs_AppData,3);
}
```

- **Capa 1: Funciones de calibración**

```
/*! \fn void cs_ReadCalibrations (void)
    \brief Reads calibration registers
    See pag 25 of CS5463 datasheet
*/
void cs_ReadCalibrations(void)
{
         cs_readRegister(CS_CURRENT_DC_OFFSET,0);
         cs_readRegister(CS_VOLTAGE_DC_OFFSET,0);
         cs_readRegister(CS_CURRENT_AC_OFFSET,0);
         cs_readRegister(CS_VOLTAGE_AC_OFFSET,0);
         cs_readRegister(CS_CURRENT_GAIN,0);
         delay_ms(50);
         cs_readRegister(CS_VOLTAGE_GAIN,0);
         cs_readRegister(CS_STATUS,0);
         cs_readRegister(CS_CONFIGURATION,0);
         cs_readRegister(CS_OPERATION_MODE,0);
}
```

```
/*! \fn void cs_SystemCalibration(unsigned int8 cs_channel)
    \brief Performs the designated calibration
    See pag 25 of CS5463 datasheet
*/
void cs_SystemCalibration(unsigned int8 cs_channel)
{
switch (cs_channel){
      case  0x00:spi_write(0b11001001);  //01001  Current  channel  DC
offset
         break;
      case 0x01:spi_write(0b11001010); //01010 Current channel DC gain
         break;
      case  0x02:spi_write(0b11001101);  //01101  Current  channel  AC
offset
         break;
      case 0x03:spi_write(0b11001110); //01110 Current channel AC gain
         break;
      case  0x04:spi_write(0b11010001);  //10001  Voltage  channel  DC
offset
         break;
      case 0x05:spi_write(0b1110010);  //10010 Voltage channel DC gain
         break;
      case  0x06:spi_write(0b11010101);  //10101  Voltage  channel  AC
offset
         break;
      case 0x07:spi_write(0b11010110); //10110 Voltage channel AC gain
         break;
      case  0x08:spi_write(0b11011001);  //11001  Current  and  Voltage
channel DC offset
         break;
      case  0x09:spi_write(0b11011010);  //11010  Current  and  Voltage
channel DC gain
         break;
      case  0x0A:spi_write(0b11011101);  //11101  Current  and  Voltage
channel AC offset
         break;
      case  0x0B:spi_write(0b11011110);  //11110  Current  and  Voltage
channel AC gain
         break;
      }
}


/*! \fn void cs_OffsetCalibration (void)
    \brief Performs the necessary actions for an offset calibration
    See pag 37 of CS5463 datasheet
*/
void cs_OffsetCalibration(void)
{
      //Desactivar IHPF y VHPF
      //cs_SystemCalibration(8);
      //cs_writeRegister(CS_CURRENT_DC_OFFSET,0,(0x7FFFFF));
      //cs_writeRegister(CS_VOLTAGE_DC_OFFSET,0,(0x7FFFFF));
      //delay_ms (70);
      cs_writeRegister(CS_CURRENT_AC_OFFSET,0,(0x000000));
      cs_writeRegister(CS_VOLTAGE_AC_OFFSET,0,(0x000000));
      cs_SystemCalibration(10);
      delay_ms (70);
}
```

```
/*! \fn void cs_GainCalibration (void)
    \brief performs the necessary actions for a gain calibration
    See pag 38 of CS5463 datasheet
*/
void cs_GainCalibration(void)
{
      //Desactivar Igain x50 / IHPF y VHPF
      //cs_writeRegister(CS_CONFIGURATION,0,(0x000001));
      //cs_writeRegister(CS_OPERATION_MODE,0,(0x000000));

      cs_writeRegister(CS_CURRENT_GAIN,0,(0x400000));
      cs_writeRegister(CS_VOLTAGE_GAIN,0,(0x400000));
      cs_SystemCalibration(11);
      delay_ms (70);
      cs_SystemCalibration(9);
      delay_ms (70);
}
```

- **Capa 1: Estado CS5463**

```
/*! \fn void cs_5463Status (void)
    \brief Checks STATUS BITS, DRDY,CRDY,IOR,VOR
    See pag 29 of CS5463 datasheet
*/
void cs_5463Status(void)
{
   if ((flag_EXT==0)&&(flag_TIMER==1))
      {

         spi_write(CS_STATUS);
//Solicita el valor del registro CS_STATUS
         cs_AppData[0] = spi_read(0xFF);          //Así  para  que  no
mande el valor por ZigBee
         cs_AppData[1] = spi_read(0xFF);
         cs_AppData[2] = spi_read(0xFF);
         int1 DRDY,IOR,VOR;

         DRDY=bit_test(cs_AppData[0],7);        //MSB (0x90)
         CRDY=bit_test(cs_AppData[0],4);
         IOR=bit_test(cs_AppData[0],1);
         VOR=bit_test(cs_AppData[0],0);

              if (CRDY==0)
              {
                        strcopy (DataOut,"CRDY Error");
                        ZB_SendToSinkD(STRLEN (DataOut),DataOut);
                        cs_writeRegister(CS_STATUS,0,(0xFFFFFF));
                        flag_EXT=1;
              }

              if (IOR==1)
              {
                        strcopy (DataOut,"Current Out of Range");
                        ZB_SendToSinkD(STRLEN (DataOut),DataOut);
                        flag_EXT=1;
              }
  }}
```

Código Funciones Inteligentes

```
//*****************************************************************
***************
/*! \file Intelligent_functions.c
* \brief The next file provides the functions needed in order to read
*        all registers and for convert values
* Description:
*
*
* Author: Nestor Carruesco (ncarruesco@gmail.com)\n
* Copyright: Tecnodiscap UNIVERSITY OF ZARAGOZA (Spain)\n
* Date: 19/01/2012\n
* Update record:\n
* _____________________________________________________\n
* Author: Nestor Carruesco (ncarruesco@gmail.com)\n
* Update: For explaining a change in firmware\n
* _____________________________________________________\n
*/

#include "Intelligent_functions.h"
```

- **Lectura REGISTROS en hexadecimal**

```
/*! \fn void cs_ReadPage0(void)
    \brief Reads all Page 0 registers
     See pag 24 of CS5463 datasheet */

void cs_ReadPage0(void)
{
   unsigned int8 k=0;
   unsigned int8 k_rotate_left=0;
   for(k=0;k<=31;++k)
   {
      k_rotate_left=k;
      k_rotate_left=(k_rotate_left<<1);
      cs_readRegister(k_rotate_left,0);
      delay_ms(15);
   }
}

/*! \fn void cs_ReadPage1(void)
    \brief Reads all Page 1 registers
    See pag 25 of CS5463 datasheet */

void cs_ReadPage1(void)
{
   cs_readRegister(CS_PULSE_WIDTH,1);
   cs_readRegister(CS_LOAD_MIN,1);
   cs_readRegister(CS_TEMPERATURE_SENSOR_GAIN,1);
   cs_readRegister(CS_TEMPERATURE_SENSOR_OFFSET,1);
}
```

```
/*! \fn void cs_ReadPage3(void)
    \brief Reads Page 3 registers
    It is a register. See pag 25 of CS5463 datasheet */

void cs_ReadPage3(void)
{
   cs_readRegister(CS_VSAG_DURATION,3);
   cs_readRegister(CS_VSAG_LEVEL,3);
   cs_readRegister(CS_ISAG_DURATION,3);
   cs_readRegister(CS_ISAG_LEVEL,3);
}
```

- **Comando para fijar el tiempo para el envío automático de DATOS**

```
/*! \fn void cs_SetSendTime(unsigned int8 Set_time)
 *  \brief Set the time for automatic send of data */

void cs_SetSendTime(unsigned int8 Set_time)
{
   switch(Set_time)
   {
        case 0:Set_counter=2;
               break;
        case 1:Set_counter=4;
               break;
        case 3:Set_counter=6;
                    break;
        case 4:Set_counter=Set_time;
                    break;
        default:Set_counter=3;
                     break;
   }
}
```

- **Código de conversión de datos de hexadecimal a decimal**

```
/*! \fn void DataConversion(unsigned int8 DatosVAR[3],unsigned int8
Gain)
    \brief Converts the UNSIGNED registers value in a decimal number*/

void DataConversion(unsigned int8 DatosVAR[3],unsigned int8 Gain)
{
//Inicializa el valor de las variables
unsigned int8 DatosMedida[3]={0x00,0x00,0x00};
int1 RealMeasure[24]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
unsigned int8 k=0;
float32 PesoBit=1;
float32 Resultado=0;
char Resultado_string[7];

    DatosMedida[2]=DatosVAR[0];     //LSB
    DatosMedida[1]=DatosVAR[1];
    DatosMedida[0]=DatosVAR[2];     //MSB
```

```
//Multiplicación de cada bit por su peso relativo
    for(k=0;k<=24;++k){
      RealMeasure[k]=shift_left(DatosMedida,3,0);
      PesoBit=PesoBit/2;
      Resultado=Resultado+(RealMeasure[k]*PesoBit);
    }

   //Conversión de float a string, para el envío de datos por ZigBee
     Resultado=(Resultado*250);
     sprintf(Resultado_string,"%6.2f",Resultado);
     ZB_SendToSinkD(STRLEN(Resultado_string),Resultado_string);


   switch (Gain)
   {
       case 0:      //Configuración ganancia en corriente
             {
                   if (Resultado<100)
                         Resultado=40*Resultado-45;
                   else if (Resultado>=150)
                         Resultado=57*Resultado;
                   else Resultado=82*Resultado-6647;
             }
          break;
       //Configuración ganancia en tensión
       case 1:Resultado=Resultado*2.15;
          break;
       //Configuración magnitudes SIN ganancia
       case 3:Resultado=Resultado/250;
            break;
       //Configuración ganancia potencias
       case 4:Resultado=Resultado*28;
          break;
   }
   //Conversión de float a string, para el envío de datos por ZigBee
   sprintf(Resultado_string,"%6.2g",Resultado);
   strcpy (DataOut,Resultado_string);
   ZB_SendToSinkD(STRLEN(DataOut),DataOut);
}


/*! \fn void SignedDataConversion(int8 DatosVAR[3],unsigned int8 Gain)
    \brief Converts the SIGNED register´s value in a decimal number
*/
Esta function es igual que el anterior, pero cambiando el peso del
primer BIT

          for(k=0;k<=24;++k)
          {
                RealMeasure[k]=shift_left(DatosMedida,3,0);
                PesoBit=PesoBit/2;
                if (k==0)
                      Resultado=Resultado-(RealMeasure[0]*1);
                else
                      Resultado=Resultado+(RealMeasure[k]*PesoBit);
          }
```

Código protocolo de comunicaciones (CCP)

```
//*****************************************************************************
***************
/*! \file CCP.c
* \brief The next file provides the functions needed in order to
handle ZigBee commands
*
* Description:
*
*
* Author: Nestor Carruesco (ncarruesco@gmail.com)\n
* Copyright: Tecnodiscap UNIVERSITY OF ZARAGOZA (Spain)\n
* Date: 12/02/2012\n
* Update record:\n
* _____________________________________________________\n
* Author: Nestor Carruesco (ncarruesco@gmail.com)\n
* Update: For explaining a change in firmware\n
* _____________________________________________________\n
*/

#include "CCP.h"
```

- **Función para la decodificación del mensaje recibido**

```
/*! \fn void SelectFunction(void)
 *  \brief This function checks the incoming message and selects the
function
 *
 *   at+ucastb:04,000D6F0000354A07 10F1000B
*/

void SelectFunction(void)
{
      cmd_error++;      //1
      if (End_Point==0x10)
      {
            cmd_error++;      //2
            if (Cluster_ID==0xF1)
            {
                  cmd_error++;      //3
                  unsigned int8 ByteData=0;
                ByteData=Data_CCP[0];

                  switch (Command)
                    {
                          //Funtions related wich Cirrus CS5463
                        case 0:cs_readRegister(ByteData,0);
                           break;
                        case 1:
                                 {
            unsigned int32 WriteData;
            WriteData=make32(0,Data_CCP[1],Data_CCP[2],Data_CCP[3]);
            cs_writeRegister(ByteData,0,WriteData);
                                  }
                        break;
```

```
                        case 2:cs_SoftwareReset();
                           break;
                        case 3:cs_PowerState(ByteData);
                           break;
                        case 4:Initiate_CS5463();
                           break;
                        case 5:cs_WriteCalibrationData();
                           break;
                        case 6:cs_StartConversions();
                           break;
                        case 7:cs_ReadInstantaneousMeasures();
                           break;
                        case 8:cs_ReadPeakMeasures();
                           break;
                        case 9:cs_ReadAverageMeasures();
                           break;
                        case 0x0A:cs_ReadRMSmeasures();
                           break;
                        case 0x0B:cs_ReadCalibrations();
                           break;
                        case 0x0C:cs_SystemCalibration(ByteData);
                           break;
                        case 0x0D:cs_OffsetCalibration();
                           break;
                        case 0x0E:cs_GainCalibration();
                           break;
                        case 0x0F:cs_5463Status();
                           break;
                        // Intelligent Funtions
                        case 0x20:cs_ReadPage0();
                           break;
                        case 0x21:cs_ReadPage1();
                           break;
                        case 0x22:cs_ReadPage3();
                           break;
                        case 0x23:cs_SetSendTime(ByteData);
                           break;
                        case 0x30:output_toggle(CONTROL_RELE);
                           break;
                        case 0x40:
                                    {
                                          Initiate_uC();
                                          Initiate_Interrupts();
                                          Welcome();
                                          Initiate_Zigbee();
                                          Initiate_CS5463();
                                    }
                           break;
                        default:cmd_error++;    //4
                           break;
                     }
                  }
               }
            }
```

- **EVENTO: Respuesta al mensaje enviado**

```
/*! \fn void Event(void)
    \brief This function indicates an error/ACK in the message */

void Event(void)
{
      switch (cmd_error)
      {
            case 1:strcpy (CodeCCP,"End_Point ERROR");
                     break;
            case 2:strcpy (CodeCCP,"Cluster_ID ERROR");
                    break;
            case 3:strcpy (CodeCCP,"*** EVENT ACK ***");
                    break;
            case 4:strcpy (CodeCCP,"Command ERROR");
                    break;
      }
      ZB_SendToSinkD(STRLEN(CodeCCP),CodeCCP);
}
```

Función principal

- **Inclusión del resto de ficheros**

```
#include "C:\PFC\pickit\main.h"
#include "C:\PFC\pickit\ZB_ATLibrary30X.c"
#include "C:\PFC\pickit\CS_5463.c"
#include "C:\PFC\pickit\Intelligent_functions.c"
#include "C:\PFC\pickit\CCP.c"
```

- **Configuración de interrupciones**

```
#build (stack=256)
#priority RDA,EXT,TIMER0

#int_TIMER0
void  TIMER0_isr(void)
{
//Overflow 4,3 sg (14 veces=1min)
contador_timer+=1;
}

#int_EXT
void  EXT_isr(void)
{
flag_EXT=0;
output_toggle(LED7);
}

#int_RDA
void  RDA_isr(void)
{
 int16 time_over = 0;
```

```
   datos_HARD = 0;

   while ( kbhit (UART_ZB) || (time_over < TIMEOUT_UART_HARD)) {
      time_over++;
      if (Kbhit (UART_ZB)) {
         time_over = 0;
         Buffer_Hard[datos_HARD++] = fgetc(UART_ZB);
         }
      //restart_wdt ();
   }
  flag_HW=1;
}
```

- **Inicializaciones**

```
void Initiate_uC(void){

   setup_adc_ports(NO_ANALOGS|VSS_VDD); //Sets up the ADC pins
   setup_adc(ADC_CLOCK_DIV_2|ADC_TAD_MUL_0);    /Disable A/D converter
   setup_psp(PSP_DISABLED);         //Disable Parallel Slave Port
   setup_spi(SPI_MASTER|SPI_MODE_0|SPI_CLK_DIV_64);  //Initiates(SPI)

   setup_wdt(WDT_ON);               //Sets up the watchdog timer.
   setup_timer_0(RTCC_INTERNAL|RTCC_DIV_256);  //Sets up the timer 0
   setup_timer_1(T1_DISABLED);                 //Disable timer 1
   setup_timer_2(T2_DISABLED,0,1);             //Disable timer 2
   setup_timer_3(T3_DISABLED|T3_DIV_BY_1);     //Disable timer 3
   setup_comparator(NC_NC_NC_NC);     //Disable the analog comparator
}


void Initiate_Interrupts()
{
   clear_interrupt(INT_RDA);        //Clears the RDA interrupt flag
clear_interrupt(INT_EXT);
   clear_interrupt(INT_TIMER0);

   ext_int_edge(0,H_TO_L);                //Set up PIC EXT0
   enable_interrupts(INT_TIMER0);  //Timer 0 (RTCC) overflow
   enable_interrupts(INT_EXT);            //External interrupt
   enable_interrupts(INT_RDA);            //RS232 receive data
   enable_interrupts(GLOBAL);
}



void Initiate_Zigbee(){

   ZB_PowerConf(AWAKE);       //ZB modo de energía >> AWAKE

      ZB_TGBootloader();      //(ver librerias)
      delay_ms(200);

   ZB_StartUpNet();

  #ifdef ZB_DEBUG             //Comprueba que reconoce el ID(ZB_DEBUG
```

```
delay_ms(20);
   strcopy (ZB_DebugStr," **** SmartMeter connected ****");
   ZB_SendToSinkD(STRLEN(ZB_DebugStr),ZB_DebugStr);
//STRLEN=obtiene la longitud de la cadena s1
   restart_wdt();
   delay_ms(20);
   #endif

  flag_HW = 0;    // Desatendemos cualquier mensaje anterior
}


//Inicialización del CS5463 ----------------------------------------

 void Initiate_CS5463(void)
{
  //Sincronizacion de la comunicacion SPI
      spi_write(0xFF);spi_write(0xFF);
      spi_write(0xFF);spi_write(0xFF);
      spi_write(0xFE);

  //Software reset
      spi_write(0b10000000);   //COMANDO 0x80
      delay_ms(200);
      restart_wdt();

  //Inicializa registros de configuración
      cs_writeRegister(CS_CONFIGURATION,0,(0x000001));//Igain=0(250x)
      cs_writeRegister(CS_OPERATION_MODE,0,(0x000060));  //HPF ON

      cs_WriteCalibrationData();//Writes Gain and AC Offset Registers

      cs_writeRegister(CS_STATUS,0,(0x800001));
      cs_writeRegister(CS_INTERRUPT_MASK,0,(0x000000));

      cs_StartConversions(); //Initiates continuous computation cycles
 }


void Welcome(void){
   unsigned int8 i;
   unsigned int8 RotateLED=0x01;

      for (i=0;i<32;i++){
        output_d(RotateLED);
        delay_ms (100);
        rotate_left(&RotateLED,1);
   }
}
```

- **Main y bucle de programa**

```
void main(){                  //Inicializamos el sistema
   Initiate_uC();
```

```
   Initiate_Interrupts();

   Welcome();

   Initiate_Zigbee();

   Initiate_CS5463();

while(true){

   // comprobamos error de buffer hard
      if (OERR) {
         clear_interrupt(INT_RDA);
         CREN=0;
         CREN=1;
        }

   //Comprobamos contador_timer >> Overflow 4,3 sg
       if (contador_timer>=Set_counter)
           {
             contador_timer=0;
             output_toggle(LED6);
             flag_TIMER=1;
         }

   //Comprobamos Error CS5463 >> CRDY,IOR,VOR
            cs_5463Status();

      //Estado ACTIVO >> Mandamos periodicamente datos por ZigBee
            if ((flag_TIMER)&&(flag_HW==0))
                  {
                        cs_readRegister(CS_STATUS,0);
                  cs_ReadRMSmeasures();
                  flag_TIMER=0;
                  }

   //Comprobamos mensaje uart hard (ZigBee)
      if (flag_HW)
            {

       //Guardamos datos del protocolo de comunicaciones CCP
                  End_Point=Buffer_Hard[0];
                        Cluster_ID=Buffer_Hard[1];
                        Length=Buffer_Hard[2];
                        Command=Buffer_Hard[3];
                        Data_CCP[0]=Buffer_Hard[4];
                        Data_CCP[1]=Buffer_Hard[5];
                        Data_CCP[2]=Buffer_Hard[6];
                        Data_CCP[3]=Buffer_Hard[7];
                        cmd_error=0;//Variable control error mensaje

      //Comprobamos mensaje respecto al CCP y mandamos respuesta
                     SelectFunction();
                     Event();

        //Borramos la flag del ZigBee
         flag_HW = 0;
       }
   }
}
```