Universidad Zaragoza
1542

Trabajo Fin de Grado

Estación meteorológica inalámbrica, de muy bajo consumo e inteligencia embebida

Autor:

Leyre Morlas Funes

Director:

Bonifacio Martín del Brío

Departamento de Ingeniería Electrónica y Comunicaciones
Escuela de Ingeniería y Arquitectura de Zaragoza
Septiembre 2015

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

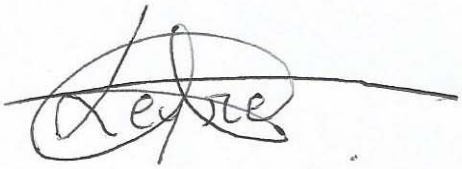
D./Dª. Leyre Morlas Funes,

con nº de DNI 73132628B en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado en Ingeniería Electrónica y Automática, (Título del Trabajo) Estación meteorológica inalámbrica, de muy bajo consumo e inteligencia embebida

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 1 septiembre 2015



Fdo: Leyre Morlas Funes

Agradecimientos

En primer lugar me gustaría agradecer la posibilidad que me brindó mi director para crear un proyecto partiendo de cero y poder disfrutar de su crecimiento, obteniendo un prototipo final fruto de todo el trabajo y esfuerzo realizado. Con el desarrollo del mismo he podido evolucionar dentro de mi formación como ingeniero al enfrentarme y solventar los problemas que han ido surgiendo, preparándome para el futuro inmediato que se abre ante mí.

Por otro lado, destacar el apoyo recibido por el profesor Rubén Blasco en la parte final del proyecto.

Agradecer a todos los compañeros y familiares que han estado apoyándome y aconsejándome en los momentos más duros, porque gracias a ellos he aprendido a encontrar el lado bueno en cada situación.

Finalmente, mención especial merece mi padre, que aunque es difícil compaginar diferentes formas de pensar, me ha contagiado la felicidad de observar la magia que un ingeniero puede lograr.

"Estación meteorológica inalámbrica, de muy bajo consumo e inteligencia embebida"

RESUMEN

En este trabajo se desarrolla una plataforma de bajo consumo para la medida y visualización en internet de los parámetros meteorológicos más relevantes.

Así, se ha desarrollado un prototipo basado en un microcontrolador que captura los datos recogidos por sensores y transmite estas medidas por radiofrecuencia a un computador, el cual sube automáticamente los datos a internet para posibilitar la consulta desde cualquier lugar. Asimismo, se han implementado algoritmos sencillos de predicción meteorológica a partir del histórico de datos recogidos. Todo el sistema se alimenta mediante baterías cargadas con paneles solares.

Si bien la idea inicial del proyecto parte del cuidado y gestión de huertos urbanos, con las modificaciones adecuadas esta plataforma podría aplicarse en otros campos, como domótica (en particular, el control automático de la climatización) o redes de sensores inalámbricos (desplegados en el campo o la ciudad).

ABSTRACT

This work develops a low power platform whose objective is the measurement of the most relevant meteorological parameters and display them online.

It is built as a prototype based on a microcontroller which obtain data from sensors and transmit the measurements to a computer by using radiofrequency. This computer automatically uploads the data to allow the user consulting them. In addition, some weather forecasting algorithms (based on historic data), have been implemented. The systems is powered by batteries, which are charged by solar panels.

Although the initial idea begin from the care and control of urban orchards, it is possible to expand it to another fields such as domotics, for example in Home Automation, or wireless sensor net along the city.

Índice General

Tercera parte: ANEXOS

E. Software Code Composer
    a. Access Point
    b. End Device
    c. Sensor de humedad
        i. SHT.c
        ii. SHT.h
    d. Sensor de Presión
        i. Presion.c
        ii. Presion.h
F. Software NetBeans
    a. Driver_Serie.java
    b. Sensor.java
    c. Prediccion.java
G. Software pluggins ThingSpeak
    a. Valores Actuales
    b. Predicción
H. Lanzamiento de la ejecución:
I. Estudio de la incorporación de la cámara OV7670

PRIMERA PARTE

LISTADOS

Lista de Figuras

Página

Lista de Tablas

Lista de Siglas y Abreviaturas

USB	Universal Serial Bus
UART	Universal Asynchronous Receiver-Transmitter
USCI	Universal Serial Communication Interface
RTC	Real Time Counter
SVS	Supply Voltage Supervisor
I2C	Inter Integrated Circuit
SPI	Serial Peripheral Interface
IrDA	Infrared Data Association
VDD	Power Supply
GND	Ground
SDI	Serial Data Input
SDO	Serial Data Output

PS	Prescaler
SCCB	Serial Camera Control Bus
SDIOD	SCCB Serial Interface Data
SDOC	SCCB Serial Interface Clock
VSYNC	Vertical Synchronization
HREF	Horizontal Synchronization
CSB	Chip Select
SCK	Serial Clock
XCLK	External Clock
PCLK	Pixels Clock
PWDN	Power Down
VGA	Video Graphics Array
QVGA	Quarter Video Graphics Array
CIF	Common Intermediate Format
QCIF	Quarter Common Intermediate Format
AWB	Automatic White Balance
AEC	Automatic Exposure Control
AGC	Automatic Gain Control
ABF	Automatic Band Filter
ABLC	Automatic Black-Level Calibration
RGB	Red Green Blue
ED	End Device
AP	Access Point
PC	Personal Computer
COM	Communication Port
AAA	Pilas triple A
HDMI	High Definition Multimedia Interface
RSSI	Received Signal Strength Indication

CPU	Central Processing Unit
RAM	Random Access Memory
PROM	Programmable Read-Only Memory
TTL	Transistor- Transistor Logic
LPM3	Low Power Mode 3
LDR	Light Dependent Resistance
LED	Light Emitting Diode
ADC	Analog to Digital Converter
HTML	HyperText Markup Language
CCS	Cascading Style Sheets
Temp	Temperatura
API	Application Programming Interface
R_{adjust}	Adjust Resistance
V_{drop}	Dropout Voltage
GHz	Gigahercios
MHz	Megahercios
mbar	Milibares
mA	Miliamperios
mAs	Miliamperios segundo
s	Segundos
h	Horas
Ω	Ohmios
mW	Miliwatios
mV	Milivoltios
^{o}C	Grados Celsius
R	Resistencia
L	Luminosidad
LnR	Logaritmo neperiano de Resistencia

LnL	Logaritmo neperiano de Luminosidad
T_{act}	Temperatura Actual
T_{ant}	Temperatura Anterior
P_{act}	Presión Actual
P_{ant}	Presión Anterior

SEGUNDA PARTE

MEMORIA

1. Introducción

Actualmente ha proliferado la implantación de huertos urbanos, que el usuario atiende con esmero día a día. Sin embargo, durante un periodo de vacaciones el huerto puede sufrir percances a causa de factores meteorológicos. De esta idea parte el presente trabajo, en el que se pretende medir variables meteorológicas y su monitorización a distancia por el usuario.

No obstante, el objetivo del trabajo es más general: desarrollar una plataforma genérica, no restringida a una aplicación concreta, que en el futuro podría aplicarse al caso de los huertos urbanos, en aplicaciones domóticas o en redes de sensores inalámbricos desplegados en ciudades.

En el caso de la domótica, por ejemplo, el prototipo podría realizar una gestión eficiente de la climatización de las viviendas desplegando varios nodos sensores a lo largo de la casa y realizar un control sin necesidad de la aparatosidad de los cables actuales.

En el caso de las redes de sensores inalámbricas en ciudades, el objetivo sería aprovechar la elevada autonomía del dispositivo, de manera que mediante extensores de rango sea posible llevar la información a través de diferentes nodos por toda la ciudad hasta el punto de acceso que se encarga de almacenar los datos.

El prototipo diseñado y realizado constituye una estación meteorológica que transmite la información entre dos tarjetas, una emisora y otra receptora. La placa emisora cuenta con una serie de sensores que dan información acerca de la temperatura, la humedad relativa, la presión y la iluminación del entorno que se desea conocer.

Los datos son recibidos en el computador mediante una transmisión basada en radio frecuencia a 2.4GHz. Con la información de estos datos se implementan algoritmos básicos de predicción del tiempo.

La información de las magnitudes y el resultado de la predicción son visualizados por el usuario en internet a través de una página web, mediante el uso de cualquier dispositivo con conexión a internet.

Uno de los objetivos fundamentales es que el sistema resultante sea de muy bajo consumo para que no suponga un gasto extra al usuario y pueda actuar autónomamente en ausencia de éste por lo que es alimentado mediante unas baterías que se recargan a partir de placas solares.

Adicionalmente, se ha realizado un estudio para la posible incorporación de una cámara que en un futuro podrá implementarse aumentando las posibilidades del producto de forma que pudiera visualizarse a tiempo real la situación que vive el huerto, los diferentes espacios de la vivienda o de una zona de la ciudad concreta.

La planificación realizada se concreta en las siguientes etapas (ver también Anexo D).

En la **primera fase** se analizan las características de los diferentes sensores a incorporar junto con los requerimientos de hardware para su acondicionamiento, se seleccionan los componentes y se realizan los pedidos necesarios.

En la **segunda fase** se realiza la programación en lenguaje C de los dispositivos (Tarjeta emisora y receptora). Para ello, en una placa de prototipos se montan los sensores conectados al dispositivo emisor.

Tras recibir las magnitudes de los sensores en la tarjeta receptora se cambia el entorno de programación y se pasa a trabajar durante la **tercera fase** en lenguaje Java. En esta fase se tratan los datos recibidos y se realiza un estudio de algoritmos básicos de predicción meteorológica para su implementación.

En la **siguiente fase** se comienzan a realizar los planos necesarios para la fabricación del prototipo y se desarrolla un circuito cargador de baterías. Tras su implementación, se finalizan los planos y se mandan a fabricar para posteriormente proceder a su montaje.

En la **penúltima fase**, se estudia el uso de una plataforma *online* basada en el internet de las cosas como medio de visualización de los datos obtenidos, se programa la página web y se realizan las pruebas necesarias para asegurar el correcto funcionamiento del prototipo. Paralelamente se analiza la incorporación de la cámara web al proyecto.

Finalmente, con la información que se ha recogido durante todo el proyecto, se realizan los documentos requeridos para el depósito final sin dejar de realizar diversas pruebas para ir comprobando el funcionamiento de los algoritmos de predicción.

El coste del prototipo viene determinado en gran medida por el kit de Texas Instruments utilizado, ya que los sensores incorporados no resultan excesivamente caros (aunque ofrecen buenos resultados).

2. Objetivos

El trabajo desarrollado cumple con una serie de objetivos planteados para la demostración de la perfecta aplicación del producto en una situación real y en un futuro inmediato. Estos objetivos son:

1º- Estudio de las posibilidades de transmisión inalámbrica con la búsqueda del menor consumo y la mayor simplicidad para la aplicación que se desea desarrollar.

2º- Estudio del protocolo seleccionado y los elementos que configuran el kit elegido para la recepción y transmisión de la información.

3º - Búsqueda y selección de los sensores más adecuados para la captación de la información del entorno.

4º- Estudio del consumo global del prototipo para el diseño de un sistema de alimentación eficiente que permita la autonomía requerida por la aplicación.

5º- Implementación en una placa de prototipos de los sensores con el montaje de la circuitería necesaria y programación de cada uno de los sensores para la recepción de la información.

6º- Búsqueda de algoritmos sencillos de predicción meteorológica que permitan ilustrar el funcionamiento global del prototipo

7º- Tratamiento de la información recibida de los sensores para implementar los algoritmos de predicción seleccionados y para poder visualizarlos.

8º- Diseño de una página web accesible desde cualquier dispositivo con acceso a internet donde se observan las predicciones y las variables meteorológicas y con el objetivo de mostrar la utilidad final del sistema.

9º - Estudio de la posibilidad de la incorporación de una cámara web al prototipo.

10º- Fabricación del prototipo en una placa de circuito impreso.

3. Diseño del Hardware

3.1. Diagrama de bloques

En este capítulo presentamos el diseño del software del sistema desarrollado. Para ello, en primer lugar mostramos en la Figura 1 el diagrama de bloques del hardware. Un conjunto de sensores tomarán medidas de variables meteorológicas, señales que convenientemente acondicionadas serán llevadas a un microcontrolador el cual, mediante una etapa de radiofrecuencia lo enviará a un receptor de RF que suministrará los datos a un segundo microcontrolador. Éste, haciendo uso de un puerto serie estándar, los enviará a un computador convencional para su tratamiento y subida de datos y gráficas a internet para su consulta desde cualquier dispositivo (móvil, tablet, otro computador).

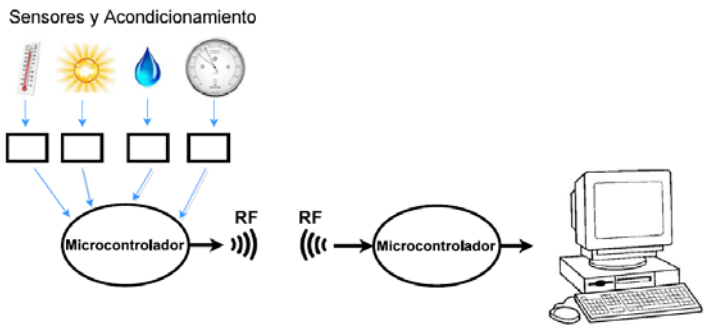


Figura 1: Diagrama de bloques

3.2. Módulo ez430-RF2500-SEH Solar Energy Harvesting

El proyecto se ha llevado a cabo tomando como base el kit ez430-RF2500-SEH Solar Energy Harvesting, de Texas Instruments. Este kit constituye una herramienta de desarrollo para la creación de una red inalámbrica de sensores de bajo consumo. Está compuesto por dos dispositivos denominados End Device (dispositivo emisor) y Access Point (receptor, conectable al puerto USB de un ordenador mediante el adaptador incluido), cuya explicación se realizará en el apartado 3.2.1. Ambos dispositivos incorporan:

- Un microcontrolador
- El hardware requerido para la transmisión inalámbrica

MICRONTROLADOR

El microcontrolador que incorporan los dispositivos es el MSP430 de Texas, bien conocido por sus características de muy bajo consumo (la razón principal para adoptar para este proyecto este microcontrolador y el kit correspondiente proporcionado por Texas). Sus características más relevantes son:

- Frecuencia de trabajo 16MHz
- Cinco modos de bajo consumo
- 32 kB de memoria Flash y 1kB de memoria RAM
- Registros USCI disponibles para interfaz: UART, SPI, I2C e IrDA.
- Conversor Analógico digital de 10 bits

TRANSMISIÓN INALÁMBRICA

El sistema se basa en la transmisión inalámbrica de la información, para ello los dispositivos incorporan el transceptor inalámbrico CC2500 que permite una comunicación por radiofrecuencia caracterizada por:

- Frecuencia de flujo de datos a 2.4 GHz.
- Posibilidad de realizar varios tipos de modulación hasta los 500Kbaudios.

3.2.1. Los Dispositivos

A nivel particular, el dispositivo Access Point (receptor) no requiere de ningún hardware adicional para el desarrollo del prototipo. Como se observa en la Figura 2, se conecta a un adaptador extraíble que se caracteriza por tener una interfaz USB. Este adaptador se conecta también al dispositivo End Device y de esta forma se pueden programar ambos dispositivos y depurar en tiempo real su comportamiento mediante un ordenador.

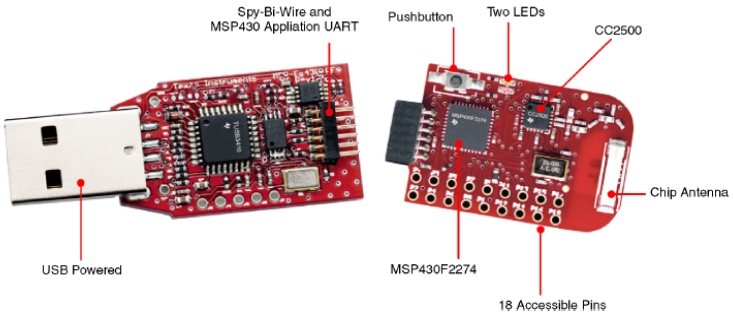


Figura 2: Dispositivo Access Point

El dispositivo emisor End Device por su parte, viene adaptado para ser alimentado mediante una placa solar de alta eficiencia equivalente a la de la Figura 3.

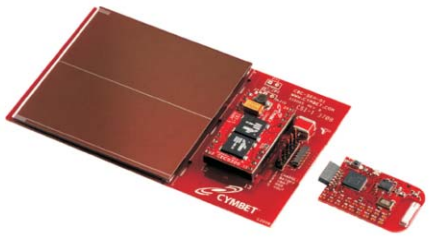


Figura 3: Dispositivo End Device y Placa Solar

Adicionalmente se incluyen dos LEDS verdes que permiten comprobar la emisión y recepción de datos.

El dispositivo End Device tiene disponible 18 pines para poder incorporar hardware externo:

Pin	Function	Description
1	GND	Ground reference
2	VCC	Supply voltage
3	P2.0 / ACLK / A0 / OA0I0	General-purpose digital I/O pin / ACLK output / ADC10, analog input A0
4	P2.1 / TAINCLK / SMCLK / A1 / A0O	General-purpose digital I/O pin / ADC10, analog input A1 Timer_A, clock signal at INCLK, SMCLK signal output
5	P2.2 / TA0 / A2 / OA0I1	General-purpose digital I/O pin / ADC10, analog input A2 Timer_A, capture: CCI0B input/BSL receive, compare: OUT0 output
6	P2.3 / TA1 / A3 / VREF– / VeREF– / OA1I1 / OA1O	General-purpose digital I/O pin / Timer_A, capture: CCI1B input, compare: OUT1 output / ADC10, analog input A3 / negative reference voltage output/input
7	P2.4 / TA2 / A4 / VREF+ / VeREF+ / OA1I0	General-purpose digital I/O pin / Timer_A, compare: OUT2 output / ADC10, analog input A4 / positive reference voltage output/input
8	P4.3 / TB0 / A12 / OA0O	General-purpose digital I/O pin / ADC10 analog input A12 / Timer_B, capture: CCI0B input, compare: OUT0 output
9	P4.4 / TB1 / A13 / OA1O	General-purpose digital I/O pin / ADC10 analog input A13 / Timer_B, capture: CCI1B input, compare: OUT1 output
10	P4.5 / TB2 / A14 / OA0I3	General-purpose digital I/O pin / ADC10 analog input A14 / Timer_B, compare: OUT2 output
11	P4.6 / TBOUTH / A15 / OA1I3	General-purpose digital I/O pin / ADC10 analog input A15 / Timer_B, switch all TB0 to TB3 outputs to high impedance
12	GND	Ground reference
13	P2.6 / XIN (GDO0)	General-purpose digital I/O pin / Input terminal of crystal oscillator
14	P2.7 / XOUT (GDO2)	General-purpose digital I/O pin / Output terminal of crystal oscillator
15	P3.2 / UCB0SOMI / UCB0SCL	General-purpose digital I/O pin USCI_B0 slave out/master in when in SPI mode, SCL I2C clock in I2C mode
16	P3.3 / UCB0CLK / UCA0STE	General-purpose digital I/O pin USCI_B0 clock input/output / USCI_A0 slave transmit enable
17	P3.0 / UCB0STE / UCA0CLK / A5	General-purpose digital I/O pin / USCI_B0 slave transmit enable / USCI_A0 clock input/output / ADC10, analog input A5
18	P3.1 / UCB0SIMO / UCB0SDA	General-purpose digital I/O pin / USCI_B0 slave in/master out in SPI mode, SDA I2C data in I2C mode

Tabla 1: Descripción de pines del dispositivo End Device

Sin embargo, seis de los pines mostrados en la Tabla 1 se utilizan para la conexión con el chip C2500 por lo que no pueden ser configurados para otra utilización.

Los pines 15, 16, 17 y 18, correspondientes al puerto tres del microcontrolador, están conectados para realizar una comunicación vía SPI con el chip C2500 y los pines 13 y 14 se utilizan para despertar al chip por medio de una interrupción cuando hay que hacer una transmisión. Esta conexión puede observarse en la Figura 4.

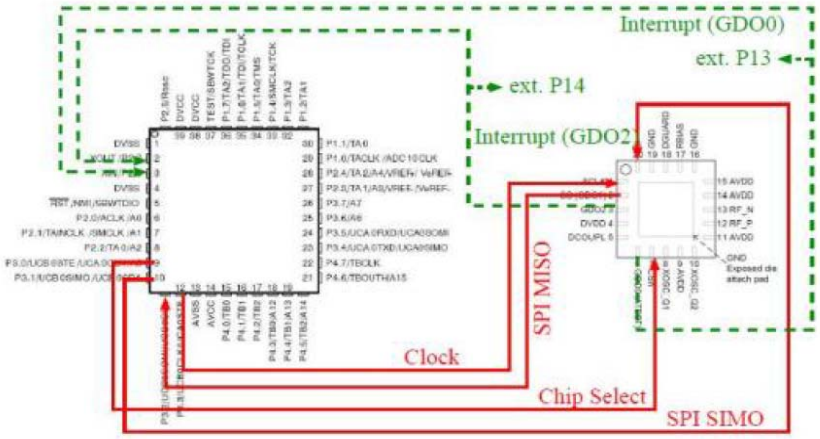


Figura 4: Conexión entre el chip C2500 y el microcontrolador MSP430

Los nueve pines restantes que están disponibles pueden ser configurados como entradas o salidas digitales y como entradas analógicas que acceden al conversor analógico- digital del microcontrolador.

Figura 5: Adaptador de pilas para alimentar el dispositivo End Device

En caso de no utilizar la placa solar, para alimentación del End Device se incluyen dos baterías de 1.5V (Figura 5). Estas baterías son suficientes para probar la demo que el kit tiene incorporada, pero en cuanto las exigencias del circuito son levemente superiores y el nivel de tensión baja mínimamente por debajo de los 3V resultan insuficientes y es necesario utilizar medios auxiliares de alimentación donde se asegure un nivel mínimo de 3.3 V.

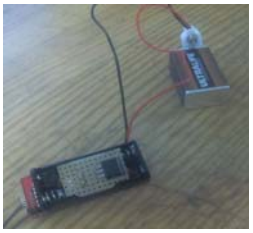


Figura 6: Adaptador casero para alimentar el dispositivo End Device

Tal y como se observa en la Figura 6, se incorporó un regulador de tensión fija de 3.3 V que regulaba la tensión de una pila de 9V, lo que permitió trabajar durante todo el proyecto sin necesidad de estar pendiente de que las pilas de 1.5 V se descargasen.

3.2.2. Protocolo de transmisión

El kit viene pre-programado con una demo que muestra la transmisión de datos entre dos dispositivos: End Device y Access Point.

El protocolo utilizado por los transceptores CC2500 es SimpliTI, el cual constituye un protocolo propiedad de Texas Instruments para aplicaciones de baja potencia. El protocolo SimpliciTI, tal y como aparece en el anexo A.a (herramienta de desarrollo eZ430-RF2500), fue creado con el objetivo de transmitir mensajes cortos y puede ser implementado en diferentes entornos de desarrollo como Code Composer Studio o IAR y es exclusivo para dispositivos de Texas Instruments.

Se distinguen tres tipos de dispositivos:

- **End Device:** Dispositivo final. Se trata de los diferentes puntos finales de la red. En el prototipo únicamente existe un End Device al cual se conectan los sensores pero como se comentará en las líneas futuras (apartado 6 de la memoria) en caso de ampliar el prototipo se utilizarían otros dispositivos End Device.
- **Access Point**: Punto de acceso. Este dispositivo es el encargado de realizar las funciones de mayor dificultad ya que le llega toda la información de los diferentes End Device conectados a la red y debe retransmitir dicha información entre otras funciones.
- **Range extender** (extensor de rango) Estos dispositivos se encargan de retrasmitir los mensajes recibidos aumentando la cobertura de red antes de llegar a un punto de acceso.

En cuanto a la topología de red existen de dos tipos:

- Por un lado, la denominada **peer to peer** comunica dos a dos los dispositivos finales y en algún caso se comunican con un extensor de rango.
- Por otro lado se encuentra la topología en **estrella**, donde el punto de acceso es el nodo central al cual se comunican todos los dispositivos finales.

En la Figura 7 pueden observarse estas dos tipologías.

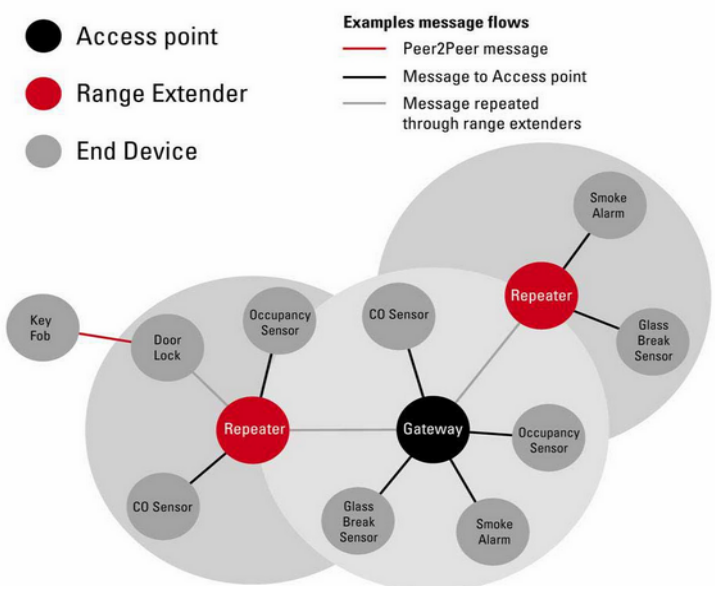


Figura 7: Topologías del protocolo SimpliciTI

En el caso del prototipo de estación meteorológica únicamente existe un dispositivo End Device, éste se conecta al hardware requerido por la aplicación y manda la información al Access Point, el cual la recibe y la envía mediante su aplicación UART al puerto COM del ordenador al que está conectado, estableciendo de este modo una comunicación serie con el mismo. En el caso de querer incorporar más dispositivos End Device la topología programada será en estrella.

El rango de transmisión de información entre dispositivos es de un máximo de quince metros cuando no hay ningún obstáculo que interfiera la transmisión y el rango de frecuencias en el que opera va de 2400 MHz a los 2483,5 MHz.

La principal ventaja de este protocolo es su reducido consumo en comparación con otros protocolos como Bluetooth o Wifi, así como su sencillez de implementación, lo que ha conducido a su elección como protocolo de transmisión.

En la Tabla 2 se adjunta una comparativa de los protocolos mencionados para confirmar su elección.

Protocolo	Pico de consumo (mA)	Alcance (m)
Wifi	80	300
Bluetooth 3.0	40	100
Zigbee	30	10 a 75
SimpliciTI	21.2	15

Tabla 2: Comparativa de consumos y alcance entre protocolos

Se ha considerado un dispositivo de clase uno para el caso del estándar Bluetooth porque aunque los dispositivos de clase dos y tres tienen un consumo inferior el alcance es inferior a 10 metros.

Tal y como se observa en la Tabla 2, y dado que la aplicación de nuestro prototipo no requiere un gran alcance, el protocolo seleccionado es una elección adecuada por ser el de menor consumo y de sencilla implementación y manejo.

Actualmente se ha lanzado al mercado la versión 4.0 del estándar Bluetooth (Bluetooth Low Energy) que sería el principal competidor con el protocolo seleccionado ya que con un consumo de 20 miliamperios garantiza una cobertura de 100 metros. Sin embargo, la complejidad de este estándar es muy superior y presenta problemas de compatibilidad.

3.3. Sensores

Para la puesta en marcha del proyecto se utilizó una placa protoboard donde se implementó el hardware asociado a cada sensor y se interconectó con el dispositivo End Device (Figura 8). Para construir una estación meteorológica de tipo medio se decidió escoger como variables a medir temperatura, humedad, presión y luz.

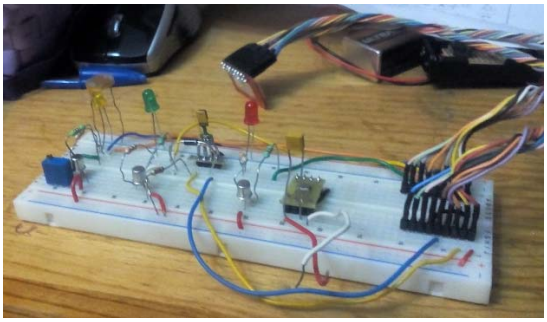


Figura 8: Protoboard con los sensores en funcionamiento

Tras un proceso de análisis y estudio, se seleccionaron los siguientes componentes para la adquisición de la información requerida.

3.3.1. Sensor de Luz

La luz ambiental es un factor a medir que nos proporciona información adicional acerca de las características del momento del día en el que se están tomando las medidas.

La radiación luminosa provoca cambios en algunos materiales semiconductores y en esencia se hallan tres formas de detectar estos cambios en la Luz.

- Fotodiodo
- Fototransistor
- Resistencia LDR (Light Dependent Resistence)

Los dos primeros se caracterizan por una mayor sensibilidad a los cambios de luz, por lo que son más precisos a la hora de realizar medidas.

La resistencia LDR se caracteriza por una variación en su resistencia al incidir la luz, de manera que cuando es iluminada por una gran cantidad de luz su resistencia puede disminuir hasta 100Ω y cuando está a oscuras aumentar hasta los 50kΩ. En este caso, la variación no es instantánea ante un cambio repentino, pero en el proyecto que nos ocupa no se suelen dar este tipo de cambios ya que está continuamente en exposición y estudia la evolución a lo largo de todo el día. Asimismo, la implementación de una resistencia LDR y su posterior linealización para obtener los luxes asociados resulta más sencillo que los otros dos dispositivos y por ello se escoge como sensor de luz para el proyecto.

Inicialmente, para poder conocer la relación de la resistencia de la LDR con los luxes de la luz ambiente se realizó un calibrado casero utilizando para ello un luxómetro.

La dependencia entre resistencia e iluminación es de la forma:

$$R = A \cdot L^{-\alpha}$$

Siendo R la resistencia medida en ohmios, L la Luminosidad medida en luxes y A y α parámetros constantes dependientes de la fabricación de la LDR.

Por tanto, si se mide la resistencia de la LDR con un polímetro y se recogen los luxes asociados a diferentes niveles de iluminación mediante el luxómetro puede obtenerse la relación entre ambas magnitudes.

Se tomaron medidas a lo largo del día, se obtuvieron valores en estancias oscuras y durante la noche y luego a partir del punto de la mañana. Los valores obtenidos se observan en la Tabla 3 y la representación de los mismos en la Figura 9.

Iluminación (Luxes)	Resistencia (Ω)
145	2850
235	1950
320	1550
400	1300
560	1190
630	1000
740	935
800	890
12000	780
27000	695
37500	620
52000	485
67000	377
77500	285
92000	240

Tabla 3: Medidas obtenidas de la LDR

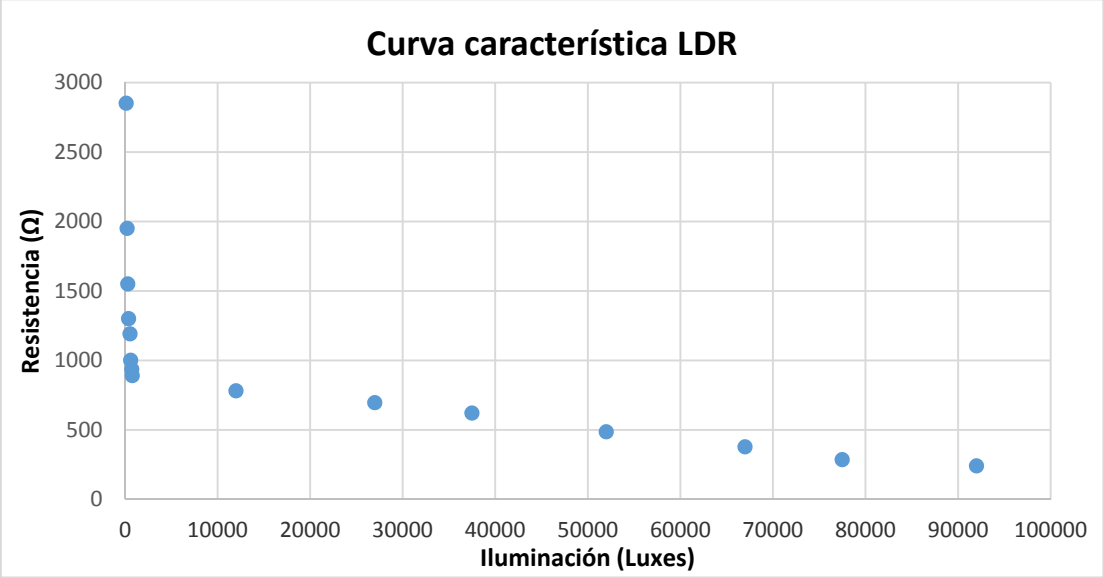


Figura 9: Curva característica LDR

Tal y como puede observarse aparecen dos tramos diferenciados. Cuando la luz del sol incide sobre la LDR a lo largo del día el comportamiento es lineal, sin embargo, cuando la luz es escasa la LDR se comporta de manera exponencial. Por esta razón, se ha decidido diferenciar estos dos tramos y calcular la iluminación con dos fórmulas diferentes en función de la resistencia que presenta la LDR.

Si la resistencia es superior a 800 Ω se aplican logaritmos naturales a los datos obtenidos y se obtiene una relación lineal entre las variables:

$$LnR = Ln(AL^{-\alpha}) \rightarrow LnR = LnA - \alpha LnL$$

El resultado gráfico se observa en la Figura 10:

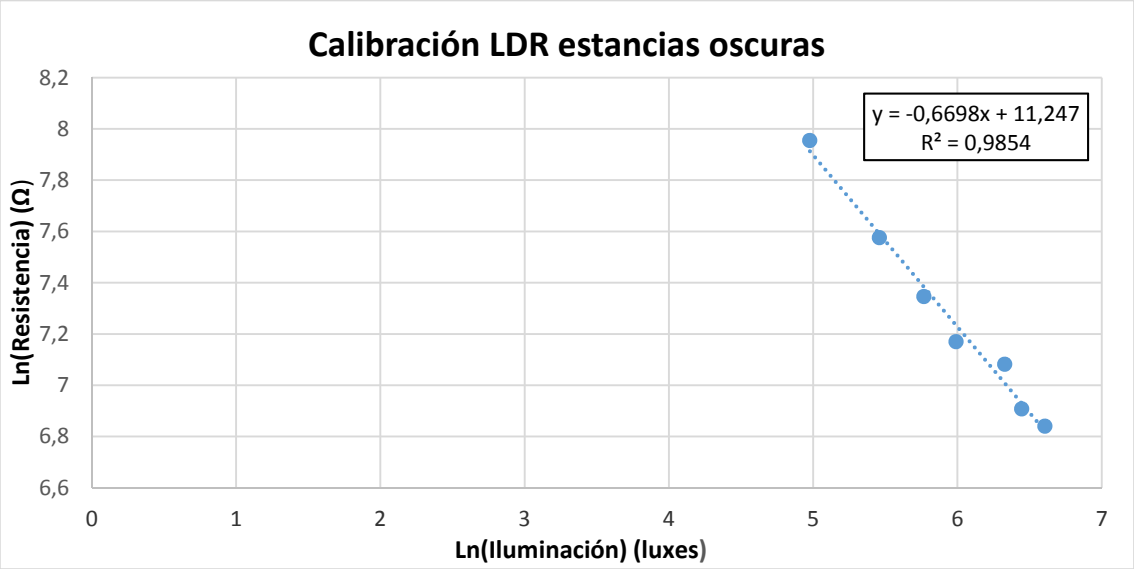


Figura 10: Calibración LDR en estancias oscuras

A partir de la Figura 10 se obtiene la siguiente relación:

$$\mathbf{LnR} = -0.6698\ \mathbf{LnL} + 11.247 \rightarrow \mathbf{LnL} = 16.79157 - 1.4929\ \mathbf{LnR}$$

$$\mathbf{L} = e^{16.79157 - 1.4929\ \mathbf{LnR}}$$

Si la resistencia es inferior debido a la incidencia de luz, los datos de resistencia e iluminación se representan directamente en la Figura 11 gracias a su correlación lineal.

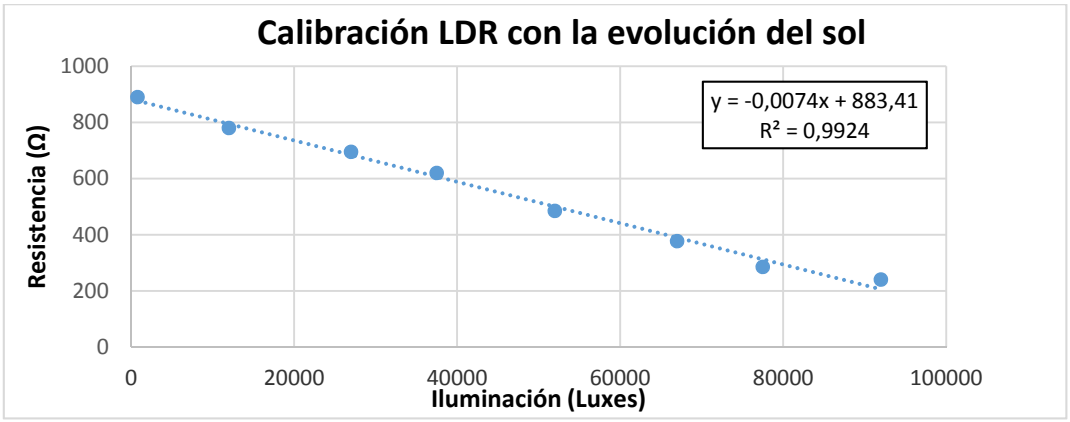


Figura 11: Calibración LDR con la evolución del sol

A partir de la Figura 11 se obtiene la siguiente relación:

$$\mathbf{R} = -0.0074\,\mathbf{L} + 883.41 \rightarrow \mathbf{L} = 119379.73 - 135.135\,\mathbf{R}$$

Circuito de Acondicionamiento:

Para su incorporación en el circuito se utiliza un potenciómetro de 5kΩ en serie con la resistencia LDR, éste permite ajustar la corriente que circula por ella. El valor del potenciómetro se ajustó para que se detectara la diferencia de iluminación en un día nublado respecto de un día soleado: R_{pot}= 3740Ω

Finalmente, en el prototipo se coloca un LED amarillo en paralelo con el transistor que alimenta la LDR para visualizar cuándo se está realizando la medida. El resultado se muestra en la Figura 12 que hace referencia a la parte del esquema general del circuito relacionada con el circuito de acondicionamiento de la LDR.

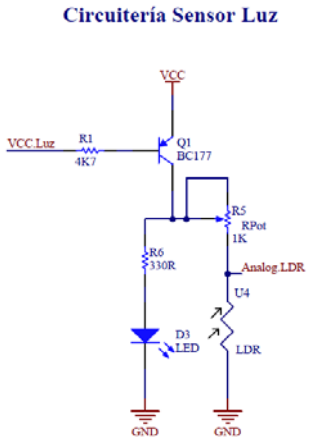


Figura 12: Esquema general del circuito: LDR

En todos los sensores la alimentación se realiza a través de un transistor operando como interruptor on-off, de manera que hasta el momento en el que se precisa realizar la medida, el sensor permanece desconectado con un consumo nulo. Adicionalmente, en paralelo con la alimentación se ha incorporado un LED que indica visualmente el instante en que entra en funcionamiento el correspondiente sensor.

3.3.2. Sensor de Humedad Relativa

La humedad relativa del aire hace referencia a la cantidad de vapor de agua que contiene la masa de estudio en relación a la cantidad que tendría si estuviera completamente saturada (ver, por ejemplo, referencia [12]). Se expresa en porcentaje, de manera que una humedad relativa del 0% hace referencia a una masa de aire seco, mientras que una humedad relativa del 100% hace referencia a un ambiente completamente húmedo.

$$HR = \frac{p_{(H_2O)}}{p^*_{(H_2O)}} \cdot 100\%$$

Donde: HR es la humedad relativa de la mezcla de aire (%), $P_{(H2O)}$ es la presión parcial de vapor de agua en la mezcla de aire (Pa) y $P^*_{(H2O)}$ es la presión de saturación de agua a la temperatura de la mezcla de aire (Pa).

Como se observa en la fórmula, para la obtención de la humedad relativa se debe tener en cuenta la temperatura y la presión del volumen analizado, siendo muy útil para la predicción de la niebla y las precipitaciones.

Existen diferentes tecnologías y métodos para medir la humedad relativa (ver referencia [14]):

- **Psicometría por bulbo húmedo/bulbo seco:**

Están constituidos por dos termómetros, uno de bulbo húmedo y otro de bulbo seco. El termómetro de bulbo húmedo es sensible a la evaporación del agua. Esta evaporación enfría el termómetro de manera que se establece una diferencia de temperaturas entre los dos termómetros. Si la diferencia es pequeña indica una humedad relativa alta ya que hay poca evaporación.

La exactitud de los instrumentos que utilizan esta tecnología depende de la proximidad con el punto de saturación, es decir 100% de humedad relativa. Cuando la humedad relativa es baja pierden precisión, además dado que el propio psicómetro es una fuente de humedad no puede ser utilizado en entornos pequeños o cerrados. A pesar de no ser el caso que ocupa nuestro proyecto, sí que pueden darse ambientes muy secos donde la precisión sería insuficiente.

- **Sensor por condensación:**

Estos sensores se basan en el uso de la medida del punto de rocío para el cálculo de la humedad relativa. El procedimiento es bastante complejo y precisa de un dispositivo denominado higrómetro óptico de espejo frío.

- **Sensor mecánico (absorción o deformación) :**

Estos sensores aprovechan que ciertas fibras orgánicas modifican sus dimensiones ante variaciones en la humedad del entorno en que se encuentran. Cuando la humedad se incrementa, aumentan su tamaño de manera que amplificando adecuadamente este alargamiento puede relacionarse con los valores de humedad relativa.

El problema que presentan es que se requiere que el aire circule a cierta velocidad para poder dar una medida fiable y ese parámetro es algo que no puede ser controlado en el entorno de estudio del prototipo ya que si se forzara influiría en otros sensores presentes como el de presión.

- **Sensor de bloque de polímero resistivo:**

Estos sensores se fabrican sobre una lámina de polímero que es capaz de absorber agua y sobre la cual se establecen dos contactos de un material conductor. Se mide la resistencia eléctrica a través del polímero.

- **Sensores capacitivos:**

En estos sensores se evalúa el cambio que se produce en la constante dieléctrica del material que se encuentra entre las placas paralelas que forman el sensor. Este cambio se produce porque el material dieléctrico absorbe o elimina vapor de agua al cambiar la humedad del entorno. Al variar la constante dieléctrica cambia el valor de la capacitancia y ésta es la magnitud que finalmente se mide y se relaciona con la humedad relativa.

Los sensores capacitivos son adecuados cuando se requiere una alta sensibilidad en ambientes poco húmedos aunque para altos valores de humedad relativa pierden precisión.

A partir de los sensores capacitivos, y con el objetivo de mejorar sus carencias, se desarrollan los sensores integrados de temperatura y humedad, caracterizados por una gran estabilidad a largo plazo; se fabrican en reducidos tamaños ya que la tecnología ha permitido integrarlos en un chip.

Estos últimos sensores son la tecnología seleccionada para implementar nuestro prototipo; en concreto se ha seleccionado el sensor digital SHT71 de la casa Sensirion. Este integrado proporciona de manera sencilla la humedad relativa y la temperatura del entorno en el que se encuentra. Se presenta en un encapsulado con cuatro terminales (VDD, GND, DATA, SCK).

Se basa en el uso de un sensor capacitivo para medir la humedad relativa, mientras un sensor de tipo band-gap se encarga de la medida de la temperatura. Ambos sensores, mediante un convertidor analógico digital también integrado en el chip, proporcionan la información en formato de 8, 12 o 14 bits por medio de una interfaz serie. Cada sensor SHT71 es calibrado individualmente, los coeficientes de calibración están almacenados en la memoria interna que integra.

El dispositivo cuenta con dos características generales que hacen de él la elección óptima. Por un lado, presenta una interfaz serie, la cual permite que con tan solo dos cables y aplicando el protocolo específico del sensor se pueda obtener la información deseada. Por otro lado, destaca su bajo consumo, como puede apreciarse de sus características eléctricas:

- Alimentación entre 2.4V y 5.5V con una tensión típica de 3.3V
- Consumo medio de 90µW. Durante la medida alcanza un máximo 3mW.
- Rango de medida de Temperatura : -40ºC a 123.8 ºC
- Rango de medida de la humedad : 0% - 100%
- Precisión en la medida de la Temperatura:$\pm$ 0.4ºC
- Precisión en la medida de la Humedad: :$\pm$ 3.0 %RH

Circuito de Acondicionamiento:

El hardware necesario para implementar el sensor se observa en la Figura 13 y su incorporación al esquema general en la Figura 14:

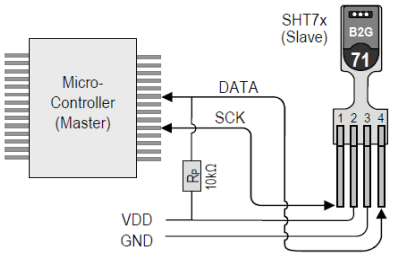


Figura 13: Circuito de Acondicionamiento del sensor SHT71

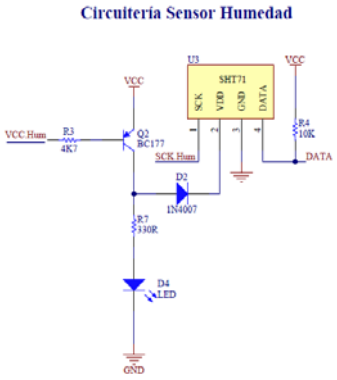


Figura 14: Esquema general del circuito: SHT71

3.3.3. Sensor de Presión

La presión atmosférica hace referencia a la presión (Fuerza por unidad de superficie) que ejerce la atmósfera sobre los objetos que están en contacto con ella. En concreto, se ha decidido medir la presión barométrica que se corresponde con la presión atmosférica más cierta corrección relacionada con la altitud geopotencial del punto de estudio (veáse, por ejemplo, la ref. [16]).

Para la selección del tipo de sensor a utilizar existe una gran diversidad debido a que pueden darse diferentes condiciones en el entorno y a que existen diversos materiales para su fabricación. Cabe destacar los siguientes tipos (veáse, por ejemplo, [15])

- **Sensores Capacitivos:**

Estos sensores miden la variación en la capacitancia entre un diafragma de metal y una fuente de metal fija. Al modificarse la distancia entre el diafragma y la fuente fija, se modifica la constante dieléctrica y esa modificación en la distancia se debe al desplazamiento producido al aplicar la presión sobre el diafragma.

- **Sensores Piezoresistivos:**

Estos sensores se basan en el uso de varias láminas de silicio conectadas a un puente Wheaststone. Cuando se modifica la resistencia del sensor resistivo del puente, indica un cambio de presión. Esto se debe a que los materiales piezoresistivos modifican su resistencia al ejercer una presión sobre ellos. Estos materiales resultan de gran durabilidad y por lo tanto son los más comúnmente utilizados.

Se selecciona un integrado que se basa en este último método de medida pero con algunos matices adicionales que mejoran su precisión y funcionamiento. Este integrado es el sensor digital MS5607-02BA de la casa MEAS Switzerland.

Este sensor proporciona una presión barométrica de alta resolución incluyendo un módulo con un sensor de presión de alta linealidad y un conversor de 24 bits de muy bajo consumo que incluye los coeficientes de calibración. Adicionalmente incorpora un sensor de temperatura de alta resolución con salida digital.

La tecnología utilizada se denomina MEMS technology y permite trabajar con una rango de histéresis muy reducido y con una alta estabilidad en las señales de salida.

El protocolo de comunicación es simple y puede emplearse tanto con puerto SPI como I2C. El funcionamiento del sensor se centra en la conversión y compensación de la tensión analógica que genera el sensor piezoresistivo que integra a través de un conversor analógico digital de 24 bits.

Cada módulo es individualmente calibrado para dos temperaturas y dos presiones y, como resultado, se obtienen seis coeficientes de compensación que se almacenan en la PROM de 128 bit que incorpora cada módulo.

Las características eléctricas más relevantes son:

- Alimentación entre 1.8V y 3.6V con una tensión típica de 3V
- Consumo máximo durante la conversión de 1.4mA.
- Rango de medida de Temperatura : -40ºC a 85 ºC
- Rango de medida de la Presión : 300 a 1100 mbar
- Precisión en la medida de la Temperatura:$\pm\ 4^0$C
- Precisión en la medida de la Humedad: 0.13 mbar

Circuito de Acondicionamiento:

Tal y como posteriormente se comentará se ha configurado para protocolo SPI por lo que el pin PS, observable en la Figura 15, se conecta a masa.

Se ha incorporado en la alimentación un transistor para que el sensor esté activo únicamente cuando va a medir y, de esta forma, minimizar el consumo. En paralelo se coloca en el prototipo un LED para visualizar cuando está midiendo.

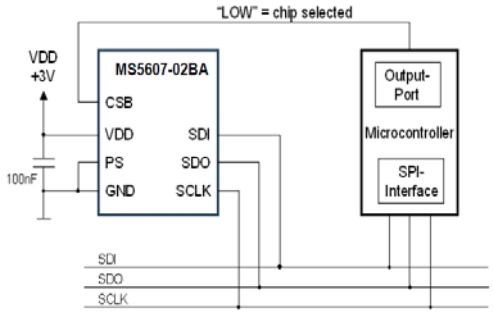


Figura 15: Circuito de Acondicionamiento del sensor MS5607-02BA

En la Figura 16 se muestra la parte del esquema general del circuito correspondiente a este sensor.

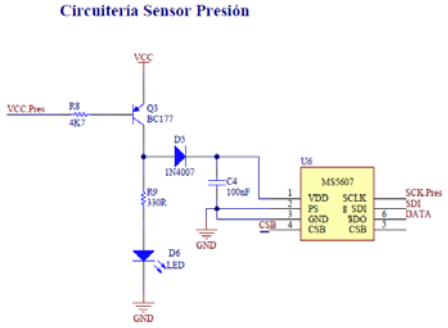


Figura 16: Esquema general del circuito: MS5607-02BA

Finalmente queremos remarcar que a lo largo de este apartado 3 se ha incluido para cada sensor un LED para una mejor visualización del funcionamiento del prototipo, sin embargo, como se explicará más adelante, esto incrementa el consumo, por lo que en el producto final no se implementarán los LED con objeto de minimizar al máximo dicho consumo.

4. Sistema de Alimentación

En este apartado se analiza el consumo del prototipo y a continuación se desarrolla un sistema de alimentación adecuado.

4.1. Estudio de Consumos

El consumo del sistema fue analizado en el laboratorio mediante osciloscopio y utilizando la placa protoboard. Se analizó inicialmente el consumo en el caso de utilizar los LEDs incluidos en el prototipo y en caso de retirarlos.

CONSUMOS CON LEDS:

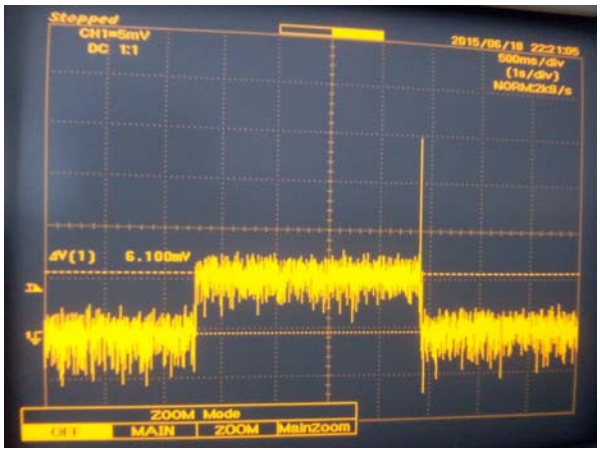


Figura 17: Captura medición y transmisión LDR con LED conectado

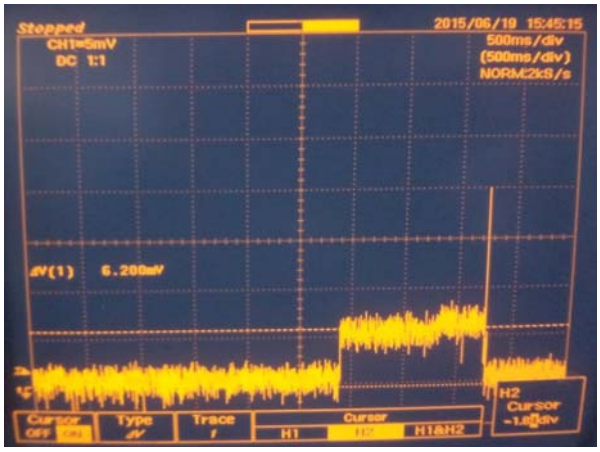


Figura 18: Captura medición y transmisión sensor SHT71 con LED conectado

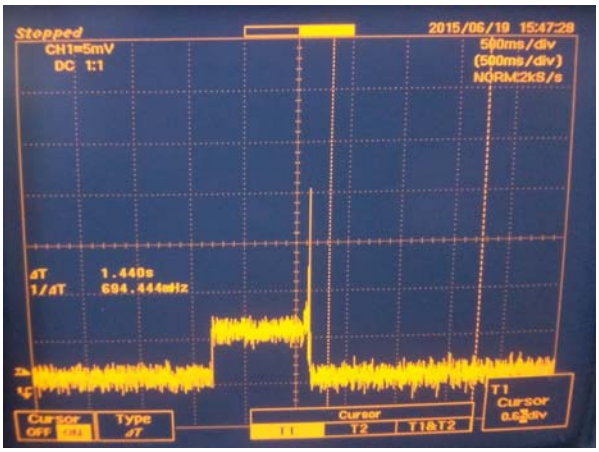


Figura 19: Captura medición y transmisión sensor MS5607-02BA con LED conectado

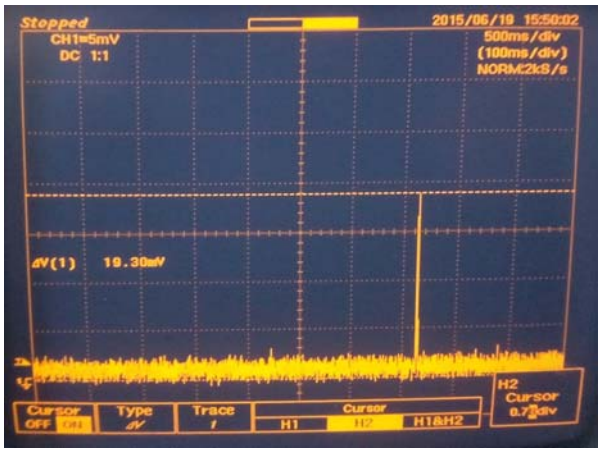


Figura 20: Captura medición y transmisión del sensor de temperatura integrado

Si bien se observa cierta distorsión en las medidas, se aprecia claramente el consumo en cada caso. El resumen del consumo en un ciclo de ejecución se presenta en la Tabla 4:

Sección de ciclo	Tiempo	Corriente media	Ponderación
Idle	900 s (15 min)	0.0006mA	0.54mAs
Medir LDR	2.145 s	6.1 mA	13.0845 mAs
Transmitir LDR	0.005 s	20.5 mA	0.1025 mAs
Medir Sensor Humedad	1.340 s	6.2 mA	8.308 mAs
Transmitir Sensor Humedad	0.005 s	21.2 mA	0.106 mAs
Medir Sensor Presión	0.942 s	5.1 mA	4.8042 mAs
Transmitir Sensor Presión	0.045 s	20 mA	0.9 mAs
Transmitir Sensor Temp. integrado	0.005 s	19.3 mA	0.0965 mAs
Total	**904.487 s**		**27.9417 mAs**

Tabla 4: Resumen de consumos en un ciclo con LEDs conectados

El consumo medio obtenido es de 0.03089mA, si se usan baterías AAA con capacidad 200 mAh y considerando un funcionamiento ideal de las baterías, se tiene

Tiempo de funcionamiento = (200/0.03089)= 6474.5872 h

Por lo tanto, idealmente funcionaría sin recarga más de 269 días, más de 8 meses.

CONSUMOS SIN LEDS:

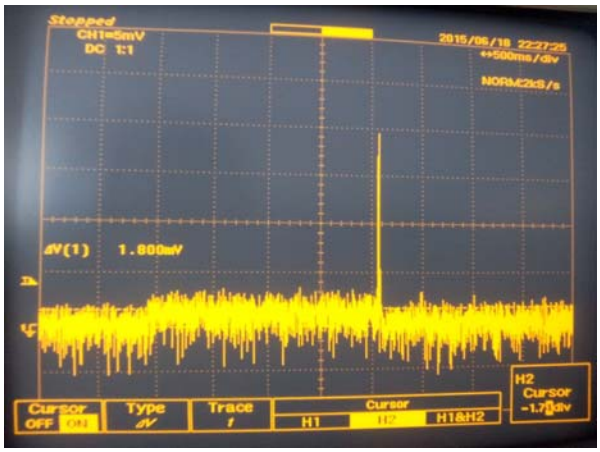


Figura 21: Captura medición y transmisión LDR sin LED conectado

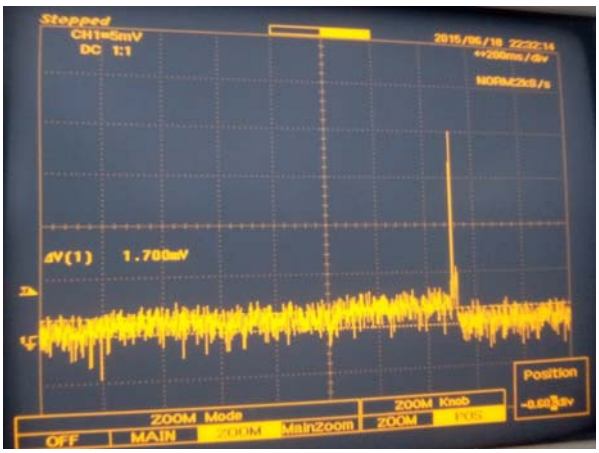


Figura 22: Captura medición y transmisión sensor SHT71 sin LED conectado

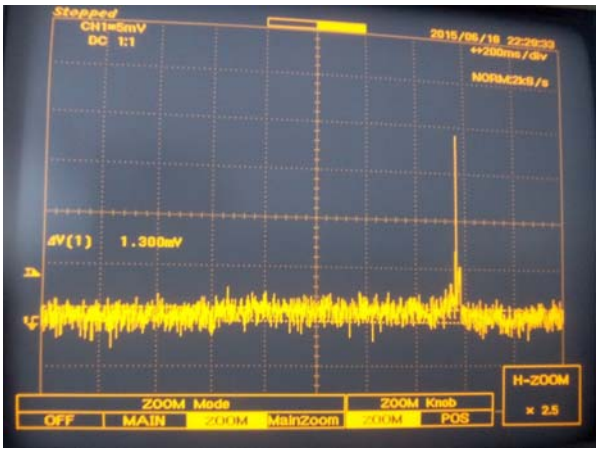


Figura 23: Captura medición y transmisión sensor MS5607-02BA con LED conectado

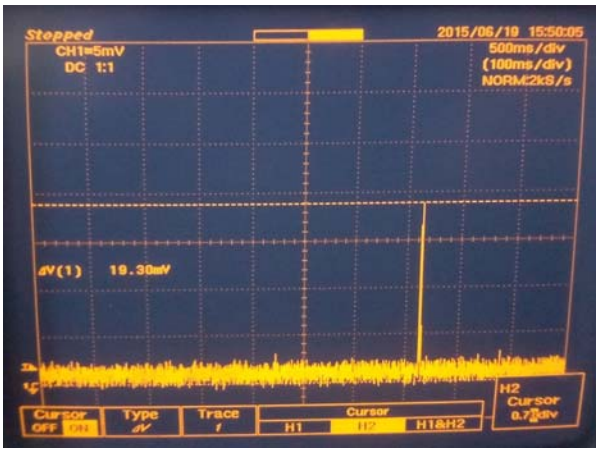


Figura 24: Captura medición y transmisión del sensor de temperatura integrado

Resumen del consumo en un ciclo (Tabla 5):

Sección de ciclo	Tiempo	Corriente media	Ponderación
Idle LPM3	900 s (15 min)	0.0006mA	0.54mAs
Medir LDR	2.145 s	1.8 mA	3.861 mAs
Transmitir LDR	0.005 s	20.5 mA	0.1025 mAs
Medir Sensor Humedad	1.340 s	1.7 mA	2.278 mAs
Transmitir Sensor Humedad	0.005 s	21.2 mA	0.106 mAs
Medir Sensor Presión	0.942 s	1.3 mA	1.2246 mAs
Transmitir Sensor Presión	0.045 s	20 mA	0.9 mAs
Transmitir Sensor Temp. integrado	0.005 s	19.3 mA	0.0965 mAs
Total	**904.487 s**		**9.1086 mAs**

Tabla 5: Resumen de consumos en un ciclo sin LEDs conectados

El consumo medio es de 0.01mAs, si se utilizan unas baterías AAA de capacidad 200mAh se puede estimar que el tiempo de funcionamiento sería:

(200/0.01)= 20000 h

Idealmente más de 833 días sin recarga, lo que son más de 2 años. Este tiempo de funcionamiento se refiere al caso de utilizar unas baterías y usarlas hasta que se descarguen.

Al comparar los dos casos es evidente que la no incorporación de los LEDs al circuito aumenta en gran medida la autonomía del sistema. Por esta razón, en un sistema real no se implementarían, sin embargo, para ayudar a la comprensión del funcionamiento del sistema a la hora de presentar el prototipo se han decidido incorporar ya que la autonomía sigue siendo bastante generosa.

En definitiva, el sistema se caracteriza por presentar un reducido consumo. Sin embargo, para ampliar esta autonomía se ha desarrollado un cargador de baterías que las recargará mediante los paneles solares en el momento en que el nivel de tensión baje, dotando al sistema de una autonomía virtualmente infinita (solo limitada por la vida útil de las baterías).

4.2. Cargador de Baterías

Una vez analizado el consumo del sistema se propone un cargador de baterías para el prototipo. Dado que se trata de una estación meteorológica que se encuentra en contacto con el aire libre, se ha diseñado un cargador que toma la energía de unos paneles solares. En la Figura 25 se muestra la parte del esquema general del circuito destinada al cargador de baterías y en la Figura 26 se muestra el prototipo final con el cargador solar integrado.

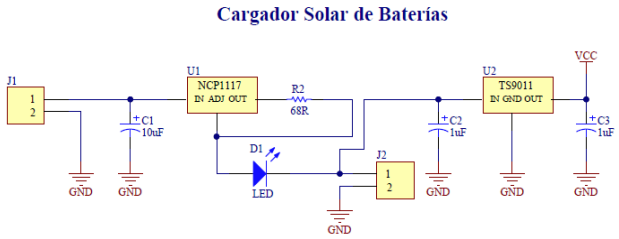


Figura 25: Esquema general del circuito: Cargador de Baterías

Figura 26: Prototipo final

El cargador se estructura en dos etapas, que se pasan a describir.

En **la primera etapa** se pueden observar 6 células solares de 1.5V/célula puestas en serie proporcionando un total de 9V de tensión cuando están perfectamente expuestas al sol. Esta tensión entra a un regulador de tensión ajustable actuando como regulador de corriente.

Para la recarga de las baterías se ha seleccionado una carga lenta, esto es cargar con una corriente correspondiente a la décima parte de la capacidad de las baterías. Esta corriente de carga es de 20mA. La regulación de la corriente de salida del regulador se realiza mediante la resistencia colocada entre los terminales OUTPUT y ADJUST ya que la tensión entre estos terminales es fija y es de 1.25V.

Realmente la carga se verá incrementada por la corriente que circula por el terminal ADJUST, pero esta corriente tiene un valor en torno a los 3mA por lo que se considera despreciable ya que además ejerce un efecto positivo en la carga.

$$I_{carga} = \frac{1.25V}{R_{adjust}} + I_{adj}$$

Si se desea una corriente de carga de 20mA → $R_{adjust} = \frac{1.25V}{0.02\ A} = 62.5\ \Omega$

Los valores comerciales más próximos a esta resistencia son 47Ω y 68Ω. Se elige el valor de 68Ω ya que es preferible una corriente de carga menor y que le cueste más tiempo a que se produzca una sobrecarga que pueda afectar a las baterías. Además cabe recordar el bajo consumo del sistema, y el hecho de que no se va a descargar completamente en ningún caso, por lo que el factor tiempo no es crítico.

A la salida del regulador se ha colocado un LED que indica visualmente el momento en que las células están cargando.

En **la segunda etapa** del sistema de alimentación se trata la tensión proporcionada por las baterías. Se han escogido cuatro baterías recargables NI-MH de capacidad 200mAh y una tensión de 1.2V/Batería, por lo tanto la tensión total proporcionada es de 4.8V. Esta tensión se ajusta a la alimentación del sistema mediante un regulador de tensión fijo de 3.3V. Este regulador se caracteriza por tener una baja tensión V_{drop}, en torno a 400mV, que permite que con tan solo cuatro baterías el sistema quede perfectamente alimentado.

5. Diseño del Software

5.1. Recepción de Datos

En una primera fase se lleva a cabo la recepción de los datos de los sensores mediante los dos dispositivos incluidos en el kit de Texas Instrument: Access Point (dispositivo receptor conectado al ordenador) y End Device (dispositivo emisor, con los sensores, alimentado mediante las baterías y paneles solares); se va a realizar una descripción independiente del tratamiento de cada una de ellos.

5.1.1. Punto de Acceso (*Access Point*)

El software básico proporcionado por Texas Instruments (Demo) incluye un código asociado para implementar en Code Composer o en IAR, en ella se muestra cómo se realiza el envío de datos entre End Device y Access Point. En este caso se ha elegido el programa Code Composer Studio 6.1 para implementar la recepción y transmisión de datos entre el End Device y el Access Point.

En el código de la Demo viene incorporada la programación del protocolo SimpliciTI, así como del envío de datos mediante el envío de la temperatura de un sensor que tiene integrado. Esta parte del código ha sido aprovechada para tener un valor adicional sobre la temperatura y se ha tomado como referencia para el modo en el que se envían el resto de los datos.

El funcionamiento general del Access Point (receptor) se muestra en la Figura 27:

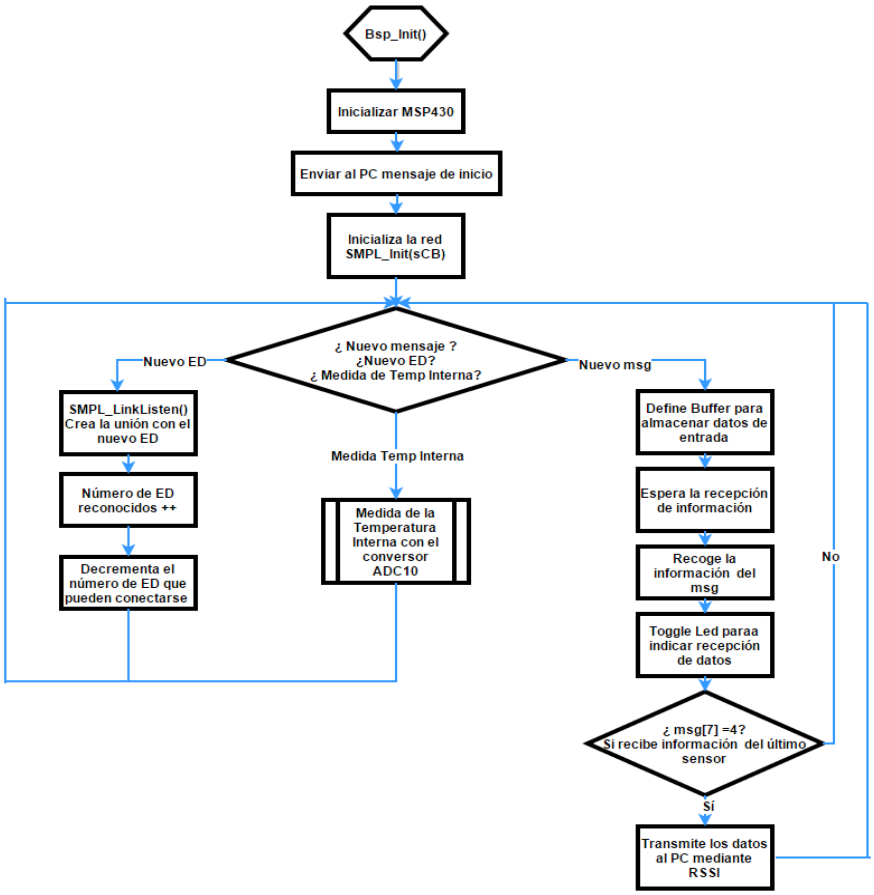


Figura 27: Diagrama de bloques del funcionamiento del Access Point

El receptor diferencia entre la conexión de un nuevo dispositivo emisor End Device (ED) o la recepción de un nuevo mensaje. Adicionalmente, al igual que el dispositivo ED, lleva incorporado un sensor de temperatura, por lo que cada cierto tiempo fijado mide la temperatura en ese punto. Esta temperatura no es de interés en el proyecto ya que no corresponde con la temperatura de la estación, por lo que se ha retirado su transmisión al ordenador.

Una vez ha recibido el mensaje, crea un buffer donde va almacenando la información que recibe de los sensores. Este buffer corresponde a una trama que se transmite mediante RSSI al ordenador al recibir la información del último sensor.

El aspecto de la trama que transmite el Access Point es la siguiente:

"S1:-XX.X,S2:X.X,S3:-XX.X,S4:XXX.X,S5:-XX.X,S6:XXXX.X"

S1: Temperatura en grados centígrados medida por el dispositivo ED

S2: Tensión LDR

S3: Temperatura en grados centígrados medida por el sensor de humedad

S4: Porcentaje de humedad, se ha multiplicado por 10 para mandar mayor precisión

S5: Temperatura en grados centígrados medida por el sensor de presión

S5: Presión en milibares

El dispositivo receptor Access Point recibe por radiofrecuencia el mensaje de bytes y en función del séptimo byte reconoce el sensor y desagrega la información almacenando los datos de las magnitudes leídas en las posiciones de la trama que le corresponden. Cuando obtiene la información de todos los sensores es cuando realiza la transmisión de la trama.

5.1.2. Dispositivo Final (*End Device*)

El procedimiento general de funcionamiento del emisor End Device (al que se conecta el prototipo con los sensores) en la lectura y envío de datos se muestra en la Figura 28:

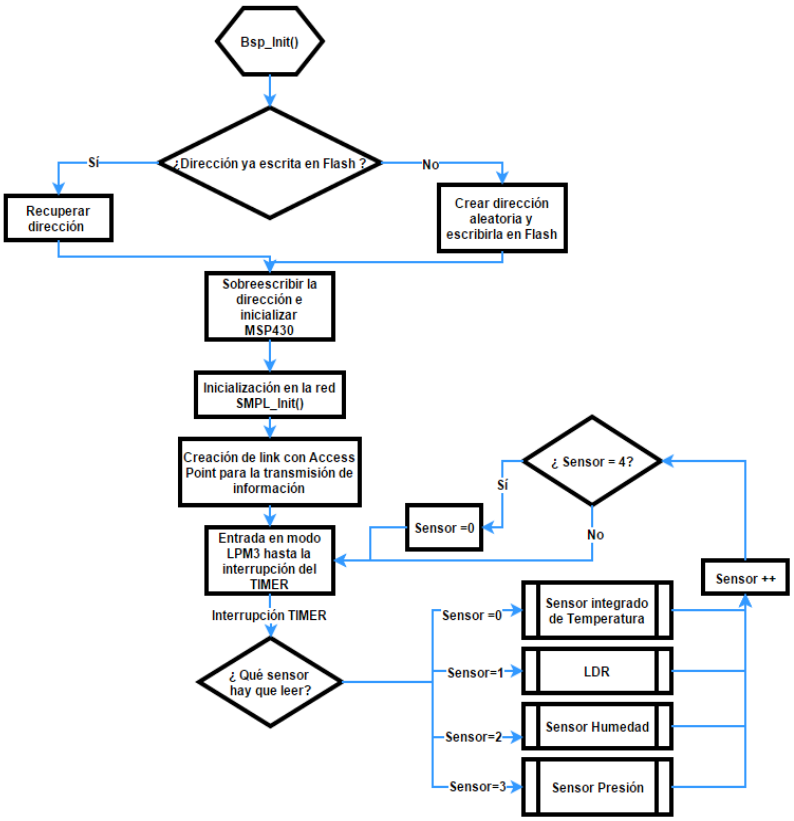


Figura 28: Diagrama de bloques del funcionamiento del End Device

Tal y como se observa en el diagrama tras el Reset e inicialización del microcontrolador y de la creación del nodo en la red, el dispositivo End Device entra en un modo de bajo consumo propio del microcontrolador: LPM3. Este modo se caracteriza por un consumo de 0.6 µA y la deshabilitación de la CPU y de los periféricos denominados como Fast. Únicamente quedan habilitados los periféricos que operan a 32 kHz (RTC, Watchdog y SVS)

Una vez realizado el proceso particular de medida de cada sensor se realiza la medida de la tensión de la batería a través del conversor analógico digital. El resultado de las medidas junto con el nivel de batería se envía al receptor codificado en un mensaje de bytes. En el programa se envían 10 bytes con información en cada transmisión.

Para todos los sensores los bytes 2, 3, 4, 5 y 6 se utilizan para enviar cierta información al Access Point sobre el estado de la batería o la célula solar si está conectada. Esta información es necesaria en caso de utilizarse la célula proporcionada en el kit, en el proyecto desarrollado no se utiliza por lo que esta información carece de utilidad, aunque para evitar entrar en conflicto con ciertas partes del código del protocolo SimpliciTI se deja sin modificar.

A continuación se va a detallar el procedimiento llevado a cabo para adquirir la información de cada sensor en particular.

SENSOR INTEGRADO DE TEMPERATURA

Para obtener la medida de la temperatura del sensor integrado se selecciona como canal del conversor analógico digital el 1010. Al seleccionar este canal se activa automáticamente la alimentación del sensor integrado de manera que se muestrea dicha tensión con el conversor.

La relación entre el valor digital obtenido y los grados centígrados asociados es:

$$^{0}\mathrm{C} = \frac{\left(\left(\left(\frac{\mathrm{V_{Digital}}}{1024}\right) \mathrm{x}\ 1500\mathrm{mV}\right) - 986\ \mathrm{mV}\right)}{3.55\mathrm{mV}}$$

Sin embargo, el valor digital obtenido está multiplicado por 10 dado que es un entero, de manera que para obtener el valor de temperatura en grados centígrados finalmente se aplica:

$$^{0}\mathrm{C} = \frac{(\mathrm{V_{Digital}} - 673)\ \mathrm{x}\ 4230}{1024}$$

Una vez obtenido el valor de la temperatura, ésta se divide en dos bytes para su transmisión y se establece como valor uno al séptimo byte del mensaje para que sea reconocido por el Access Point. La Figura 29 muestra un resumen de los datos enviados en el mensaje.

MSByte Temp	LSByte Temp	Tensión Batería	Modo Alimentación	Núm Transmisiones	Núm Transmisiones	On/OFF	Sensor 1	0	0
msg 0	msg1	msg2	msg3	msg4	msg5	msg6	msg7	msg8	msg9

Figura 29: Mensaje transmitido al Access Point al leer el sensor integrado de temperatura

SENSOR DE LUZ

Para realizar la medida de la LDR, inicialmente se activa el transistor que la alimenta y se realiza un retardo de dos segundos antes de iniciar la medida para permitir que la LDR se estabilice, ya que como se ha comentado no tiene mucha sensibilidad.

Se conecta la entrada analógica AN0 disponible en End Device a la resistencia LDR para medir el nivel de tensión en cada medida.

Una vez alimentada se mide con el conversor analógico digital la tensión y se manda al receptor Access Point. La tensión se envía codificada en dos bytes más un byte que lo identifica como LDR, es decir, el byte séptimo tiene como valor 2.

El mensaje enviado al Access Point se muestra en la Figura 30:

Tensión sin Filtrar	0	Tensión Batería	Modo Alimentación	Núm Transmisiones	Núm Transmisiones	On/OFF	Sensor 2	Tensión Filtrada	0
msg 0	msg1	msg2	msg3	msg4	msg5	msg6	msg7	msg8	msg9

Figura 30: Mensaje transmitido al Access Point al leer la LDR

Conocida V_{AN0} se conoce R_{LDR} y será sencillo conocer posteriormente los luxes asociados:

$$V_{AN0} = I \times R_{LDR} \qquad I = \frac{VDD}{R_{LDR}+R_{POT}} \rightarrow V_{AN0} = \frac{VDD}{R_{LDR}+R_{POT}} \times R_{LDR} \rightarrow$$

$$R_{POT} \times V_{AN0} = (VDD - V_{AN0}) \times R_{LDR} \rightarrow R_{LDR} = \frac{V_{AN0}}{VDD - V_{AN0}} \times R_{POT}$$

SENSOR DE HUMEDAD

El sensor de humedad seleccionado es un sensor que proporciona por su línea DATA el valor de la temperatura y de la humedad de manera digital en formato de 14 bits. Para establecer la comunicación con él se debe implementar el protocolo particular que precisa dado que no se corresponde con ningún protocolo estándar.

Por un lado se conecta la línea de reloj (SCK) al microcontrolador a una patilla configurada como modo salida y, por otro lado, la línea DATA a una patilla configurable como salida o entrada.

Para implementar el protocolo se han programado en Code Composer Studio unas funciones incluidas en unos ficheros a parte del proyecto principal, SHT.h y SHT.c, donde se realiza la configuración de los pines y se programan el envío de los comandos de lectura, escritura y reset, así como el tratamiento de los datos recibidos, adecuándolos para obtener en el programa principal el valor de la temperatura en grados centígrados y el porcentaje de humedad relativa.

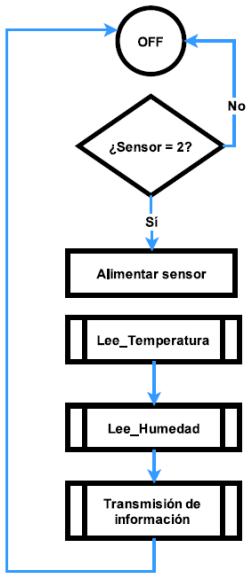


Figura 31: Diagrama de bloques del funcionamiento del sensor de humedad

Tras la activación del transistor que alimenta al sensor, se inicializa mandando el comando de Reset por la línea de datos utilizando el método de escritura programado. Una vez inicializado se transmite el comando de lectura de la temperatura y por la línea DATA se obtiene el valor digital de la temperatura. Este valor se traduce en grados centígrados aplicando la relación proporcionada por el fabricante:

$$^{0}\mathrm{C} = (\mathrm{V_{Digital}}\ \mathrm{x}\ 0.01) - 40$$

El valor obtenido por la línea DATA corresponde a una humedad no lineal que conviene ser compensada. Para ello se aplica la siguiente fórmula:

$$\mathrm{RH_{linear}} = (\mathrm{V_{Digital}}\ \mathrm{x}\ 0.0405) - (\mathrm{V_{Digital}}^{2}\ \mathrm{x}\ 0.0000028) - 4$$

Además, cuando la temperatura difiere de los 25 º C se precisa una compensación adicional sobre el valor linealizado de humedad. Esta compensación precisa conocer el valor de temperatura que previamente se ha medido:

$$\text{Humedad} = (\text{Temp}(^{o}C) - 25) \times (0.01 + 0.00008 \times V_{Digital}) + RH_{linear}$$

Tras obtener los valores de temperatura y humedad se dividen en bytes para poder transmitir la información en el mensaje y se asigna al byte séptimo el número 3 para que sea reconocido por el Access Point. El mensaje enviado se muestra en la Figura 32:

MSByte Temp	LSByte Temp	MSByte Humedad	Modo Alimentación	Núm Transmisiones	Núm Transmisiones	On/OFF	Sensor 3	LSByte Humedad	1=- 0=+
msg 0	msg1	msg2	msg3	msg4	msg5	msg6	msg7	msg8	msg9

Figura 32: Mensaje transmitido al Access Point al leer el sensor de humedad

Una vez que la información es recibida en el Access Point la temperatura y la humedad relativa se utilizan para la obtención del valor del punto de rocío, es decir, la temperatura a la cual el vapor de agua contenido en el aire comienza a condensarse. Este valor es utilizado posteriormente en los algoritmos de predicción.

La fórmula que permite obtener este valor junto con las mostradas en este apartado, es extraída del anexo A.d (datasheet del sensor de humedad SHT71):

$$PR = T_n x \frac{\text{Ln}\left(\frac{rh}{100}\right) + \frac{mxT}{T_n + T}}{m - \text{Ln}\left(\frac{RH}{100}\right) - \frac{mxT}{T_n + T}}$$

El valor de las constantes depende del rango de temperatura (Tabla 6):

Rango de Temperatura	**T_n (ºC)**	**m**
0 a 50 ºC	243.12	17.62
-40 a 0 ºC	272.62	22.46

Tabla 6: Constantes para el punto de rocío

Para evitar cargar la librería Math.h para la aplicación de logaritmos se programó una tabla y una función de interpolación para el cálculo del punto de rocío. Sin embargo, posteriormente se decidió que para minimizar el consumo y cómputo del Access Point sería la plataforma de desarrollo NetBeans, que posteriormente se comentará, la que se encargará de esta operación.

SENSOR DE PRESIÓN

El sensor digital de presión seleccionado incluye la posibilidad de transmitir los datos mediante protocolo SPI o I2 C. Se ha elegido el protocolo SPI de manera que se precisan dos líneas de datos SDI y SDO a parte de la línea de CSB.

Para poder implementar este protocolo ha sido necesario que la línea DATA del sensor de humedad fuera compartida con SDO de este sensor ya que, como se describió al inicio de la memoria, el número de pines está limitado.

Para implementar este protocolo se han programado en Code Composer Studio los códigos Presion.h y Presion.c. En ellos se ha programado la lectura y escritura de datos ya que no se han utilizado los registros del microcontrolador.

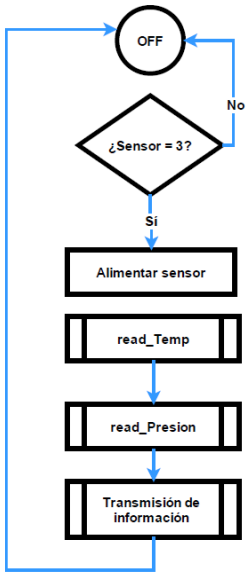


Figura 33: Diagrama de bloques del funcionamiento del sensor de presión

Tras la alimentación del sensor mediante su correspondiente transistor se lanza la función para leer la temperatura. Esta función pude esquematizarse en cuatro pasos:

1º Reset del sensor mediante el envío del comando RESET

2º Lectura de los coeficientes de calibración que se encuentran en la PROM del chip

3º Lectura del resultado de la conversión del ADC interno que proporciona los valores digitales de la presión y la temperatura descompensada

4º Aplicación de fórmulas para compensar la temperatura y conversión a grados centígrados.

A continuación, se lanza la función para obtener la presión en milibares. En esta función se utiliza la presión descompensada anteriormente calculada y se aplican los coeficientes de calibración proporcionados por el fabricante para su compensación

El mensaje que se transmite al Access Point se muestra en la Figura 34:

MSByte Temp	LSByte Temp	MSByte Presión	Modo Alimentación	Núm Transmisiones	Núm Transmisiones	On/OFF	Sensor 4	LSByte Presión	0
msg 0	msg1	msg2	msg3	msg4	msg5	msg6	msg7	msg8	msg9

Figura 34: Mensaje transmitido al Access Point al leer el sensor de presión

5.2. Tratamiento de los Datos

En este apartado se va a describir el tratamiento de los datos sensoriales una vez que el Access Point conectado al ordenador los recibe y los vuelca al puerto COM serie del mismo. Para llevar a cabo este tratamiento en el ordenador se ha utilizado la plataforma de desarrollo NetBeans IDE 8.0.2 en lenguaje Java y se han creado tres clases para estructurar la recepción y tratamiento.

En primer lugar, está la clase Sensor, en la que se crean los objetos sensor junto con una serie de métodos que permiten obtener su valor, su identificación y su almacenamiento en una tabla (“treemap”) de manera que se tiene un registro de la magnitud del sensor recibida junto con la fecha y la hora en que se ha recibido. Este histórico tiene una extensión de 12 valores y será utilizado para los algoritmos de predicción.

Por otro lado, se encuentra la clase Predicción donde se implementan los métodos que permiten actualizar el resultado de las variables de predicción.

En último lugar se encuentra la clase Driver_Serie que a su vez constituye la clase principal donde se encuentra el main o ejecutable del proyecto.

De las tres clases definidas destacan la segunda y la última por lo que se va a incidir más en su descripción.

CLASE PREDICCIÓN:

La clase predicción tiene como objetivo implementar dos tipos de algoritmos básicos que permiten predecir la lluvia, el tiempo, la niebla y la temperatura mínima que habrá por la noche.

El **primer tipo** de predicción (ver referencia [22]) utiliza las variables temperatura y presión para predecir el estado del tiempo y de la lluvia. Se realizan medidas cada 15 minutos y se evalúa cada tres horas lo sucedido con respecto a las tres horas anteriores. Por esta razón, se configura la estación para que tome medidas cada 15 minutos y las almacene en el histórico particular de cada sensor.

A las tres horas el histórico tiene almacenados doce valores de temperatura y doce valores de presión. Se realiza una media para tener un valor que represente a las variables en esas tres horas y se evalúa con el valor de las tres horas anteriores.

Para determinar que variación han sufrido las magnitudes se considera lo establecido en la Tabla 7:

Variable	Presión	Temperatura
Desciende	Pact – Pant <= - 0.67 mbar	Tact – Tant <= - 2ºC
Estable	|Pact –Pant| < 0.67 mbar	|Tact –Tant| < 2 ºC
Aumenta	Pact – Pant >= 0.67 mbar	Tact – Tant > = 2ºC

Tabla 7: Cuantificación de la interpretación de incremento y decremento en la Presión y Temperatura

A partir de la información de si ha descendido, ha aumentado o ha permanecido estable, se realiza la predicción atendiendo a los algoritmos desarrollados a partir de las reglas de Gachons (ver referencia [22]) cuyo resumen se muestra en la Tabla 8:

Temperatura	Presión	Lluvia	Tiempo
Desciende	Desciende	Abundante	Malo
Aumenta	Aumenta	Improbable	Si Humedad < 50% Caluroso y Seco Si Humedad > 50% Bueno
Aumenta	Desciende	Impredecible	Variable
Desciende	Aumenta	Impredecible	Malo
Aumenta	Estable	Improbable	Bueno
Desciende	Estable	Probable	Malo
Aumenta	Estable	Improbable	Bueno
Estable	Desciende	Probable	Malo
Estable	Estable	Impredecible	Variable

Tabla 8: Resultado de los Algoritmos de Predicción

El segundo tipo de predicción (ver referencia [23]) actualiza las variables Niebla y Temperatura mínima de la noche. Esta predicción se basa en la información que proporciona el punto de rocío calculado a las ocho de la tarde. Si la Temperatura a las ocho de la tarde y el punto de rocío son similares se formarán bancos de niebla, pero si además la humedad relativa está entorno al 100%, aparecerá una niebla abundante.

Por otro lado, si la humedad relativa a esa hora está en torno al 50% se puede predecir que la temperatura mínima por la noche corresponderá con el punto de rocío a las ocho de la tarde.

CLASE DRIVER_SERIE:

Como ya se ha comentado esta clase corresponde a la clase principal desde la cual se ejecuta el main del proyecto, pero adicionalmente se encarga de otras tareas.

Por un lado, en esta clase se obtiene del puerto serie COM el mensaje que el Access Point manda. Este mensaje se descompone en subtramas para localizar la información de cada sensor y actualizar los objetos de la clase sensor.

Por otro lado, se crea un TIMER para la activación de tres tareas periódicas.

La primera tarea periódica se ejecuta tras un retraso de tres horas tras la activación del sistema y con un periodo de tres horas. Esta tarea es la encargada de lanzar la predicción de la lluvia y el tiempo cada tres horas.

La segunda tarea periódica se activa todos los días a las ocho de la tarde. Esta tarea se encarga de lanzar la función que realiza la predicción de la niebla y la temperatura mínima de la noche.

Finalmente, la tercera tarea periódica se activa todos los días a las cinco de la mañana y se encarga de resetear la predicción de la niebla y de la temperatura mínima, ya que hasta por la tarde no se pueden predecir y son válidas únicamente durante la noche.

Además de estas dos tareas, la clase Driver_Serie es la encargada de comunicarse con la plataforma utilizada para visualizar los datos en internet.

6. Acceso a los Datos a través de Internet

El objetivo final de este proyecto es que el usuario de la estación pueda tener información acerca de ella sin necesidad de estar físicamente junto a la estación. Con este objetivo en mente, se ha buscado el desarrollo de una aplicación para el móvil o la realización de una página web accesible para el mismo.

Finalmente, se ha desarrollado una página web por no tener restricciones de sistema operativo, cuestión que podría afectar en el desarrollo de una aplicación, y por lo tanto es accesible desde cualquier dispositivo con acceso a internet, bien sea un ordenador de mesa, un portátil, un móvil o una Tablet y sea cual sea su sistema operativo.

Para la creación de esta página web se ha utilizado la plataforma online ThingSpeak (https://thingspeak.com/). ThingSpeak es una aplicación open source para el Internet de las cosas y una API que permite almacenar datos en Internet. Para ello, ThingSpeak permite la creación de un canal el cual cuenta con un máximo de ocho variables, que constituyen el almacén de los datos que se desea enviar desde el programa que se esté desarrollando.

ThingSpeak permite mostrar de manera sencilla la evolución de las variables a lo largo del tiempo en una serie de gráficos, permitiendo además la inserción de pluggins para mostrar una visualización diferente.

En el proyecto realizado se han asociado a los diferentes campos las variables temperatura media, media de las temperaturas recibidas por los diversos sensores, tensión LDR, presión, humedad, lluvia, tiempo, niebla y temperatura mínima por la noche.

Se representa la evolución de la temperatura, la presión y la humedad; para la iluminación se ha insertado un plugging que muestra en tanto porciento la iluminación del entorno.

Por otro lado se han insertado dos tablas con la información en cada momento de las variables de predicción y de las magnitudes temperatura, humedad, presión e iluminación (en luxes). Esta tabla ha sido insertada mediante la programación de la misma en javascript ya que la plataforma ThingSpeak permite programar online en html, CCS o javascript los pluggings que se desean insertar.

Adicionalmente a la información sobre las variables, se muestra la localización del parque de Macanaz en Zaragoza, posible zona de trabajo de la estación meteorológica, así como un video del funcionamiento de la misma.

El resultado aparece en la Figura 35 y en el enlace que posteriormente se comenta.

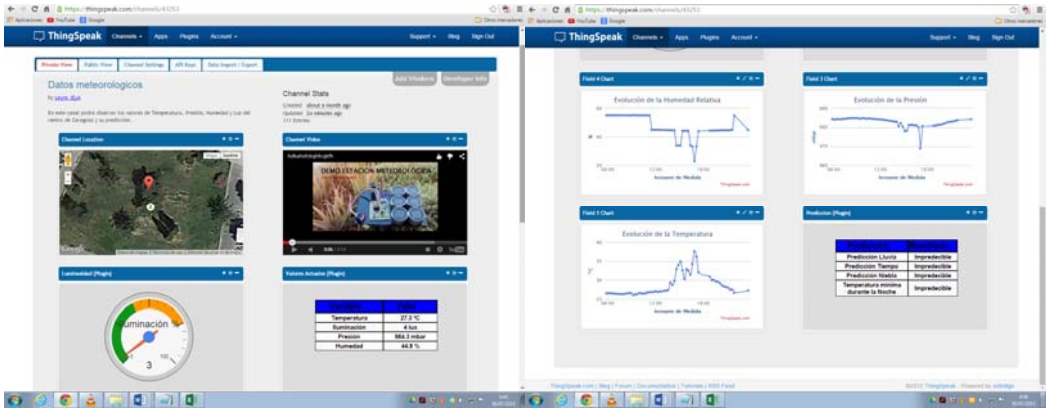


Figura 35: Capturas de la página web donde se visualizan los resultados

7. Prototipo

Tras la explicación de cómo se ha desarrollado el proyecto, se presenta en este apartado la materialización del prototipo en la forma de un circuito impreso. Para la realización del esquemático y de la placa de circuito impreso se ha utilizado el programa de diseño de circuitos electrónicos Altium Designer versión 14.0.

7.1. Esquemático

En el esquemático presentado en la Figura 36 pueden observarse las diferentes partes que constituyen el prototipo:

- Cargador de baterías
- Conector Módulo de RF (Conexión a los pines proporcionados por el dispositivo End Device de Texas Instrument)
- Circuito del sensor de iluminación
- Circuito del sensor presión y temperatura
- Circuito del sensor humedad y temperatura

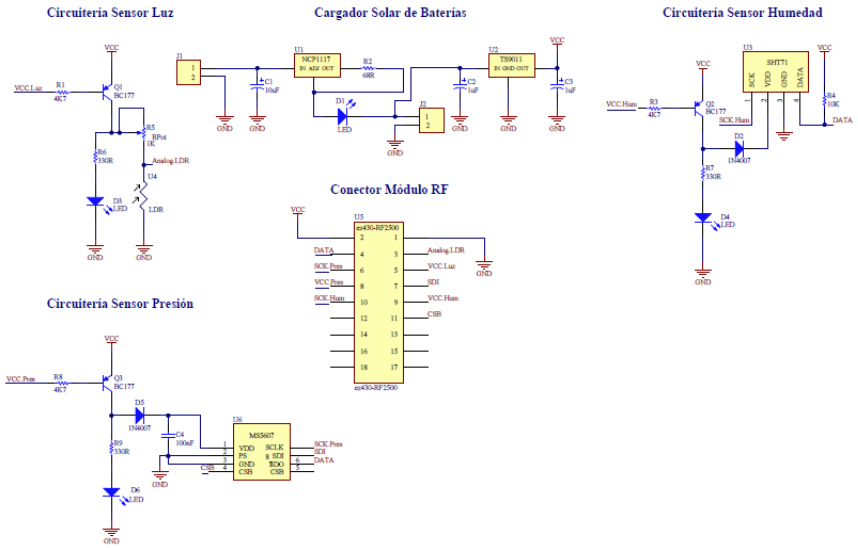


Figura 36: Pantallazo del plano esquema general del circuito

7.2. Placa de circuito impreso

Se muestran a continuación en la Figura 37, un pantallazo de la cara Top y Bottom de la placa de circuito impreso del prototipo. Ambas caras cuentan con un plano de masa para minimizar las interferencias.

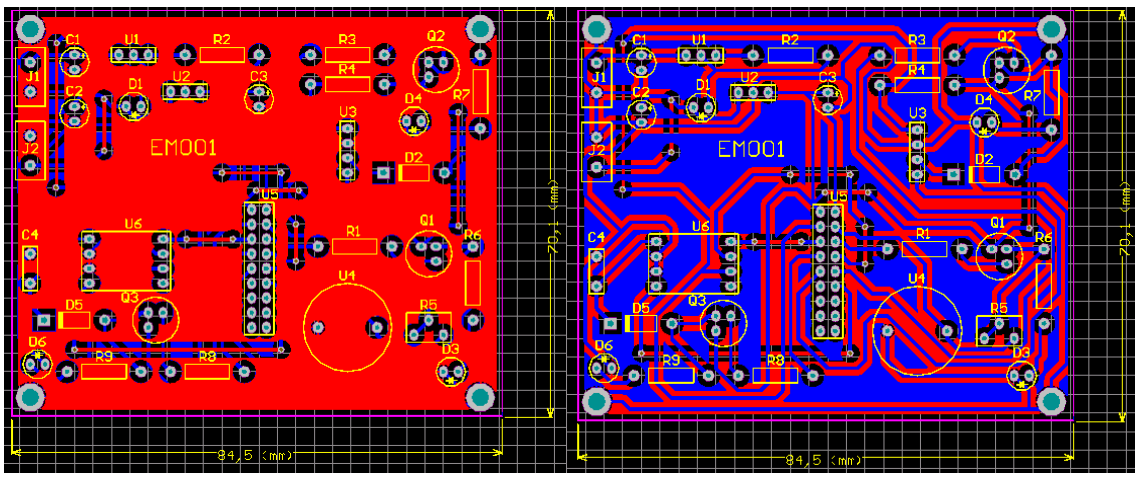


Figura 37: Pantallazo de la cara Top y Bottom

Finalmente en la Figura 38 se incorpora una visualización en 3D de la placa. El funcionamiento del prototipo podrá verse si se desea in situ en la presentación y en el video que aparece en la página web que aparece en el apartado 7.3.

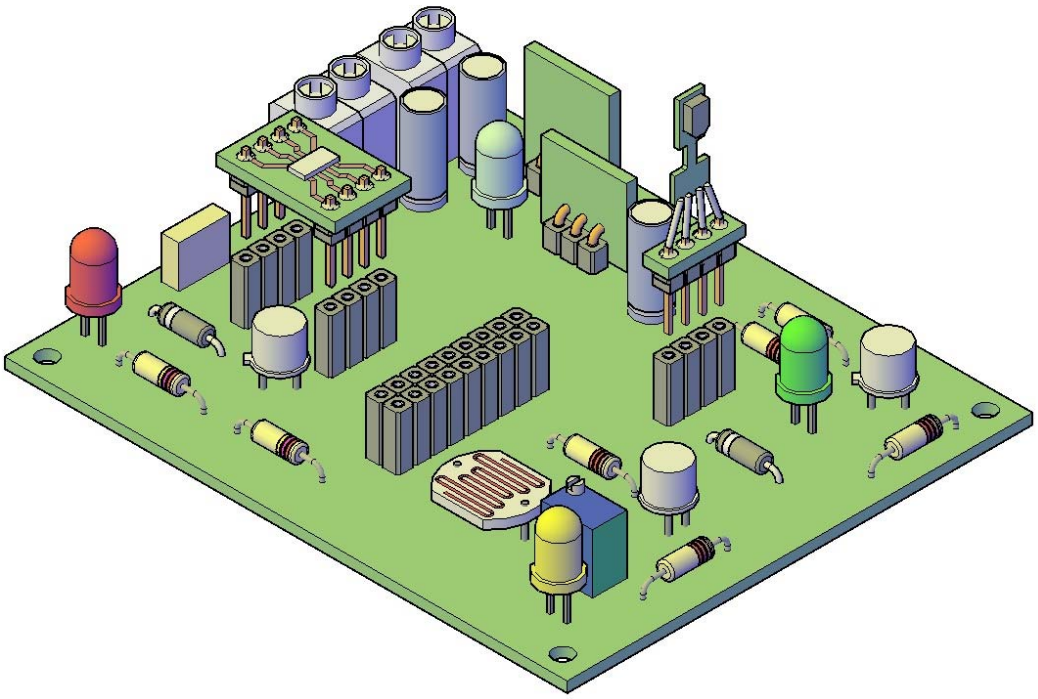


Figura 38: Visualización 3D del prototipo

7.3. Pruebas realizadas

Tras el montaje completo del prototipo se procede a realizar las pruebas necesarias para verificar su correcto comportamiento. El resultado obtenido puede ser visualizado en la página web desarrollada.

Enlace: https://thingspeak.com/channels/43253

8. Conclusiones y Trabajo Futuro

Para finalizar con la memoria, se presentan dos cuestiones importantes acerca de las posibilidades de este proyecto.

Por un lado, cabe destacar que el prototipo se ha desarrollado considerando que el receptor Access Point está conectado al puerto USB de un ordenador Windows convencional. Sin embargo, existen otras posibilidades, como por ejemplo la utilización de un mini PC Android o un mini PC Windows.

Estos mini PC's son ordenadores extraordinariamente pequeños y que se comercializan en algunos casos por debajo de los treinta euros; pueden conectarse a un monitor o televisor vía HDMI e incluyen un puerto USB. A pesar de su limitada memoria, el almacenamiento de datos se realiza online y no en el PC, por lo que no es problemático. Estos dispositivos permiten que la estación meteorológica pueda ser comercializada como un dispositivo completo donde el usuario no precisa de ningún elemento extra para ponerlo en funcionamiento y, por lo tanto, añade valor al concepto general del producto.

Finalmente, otra línea de investigación de cara a la mejora de las posibilidades de la estación es la posible incorporación de una pequeña cámara. Esta cámara tendría la misión de tomar instantáneas del entorno de la estación y, por lo tanto, proporcionar una información adicional al usuario sobre el estado meteorológico o sobre el entorno (por ejemplo, puede ser utilizada como medio de seguridad).

Se ha realizado un estudio (ver Anexo I) sobre la posible implementación de la cámara OV7670 (ver referencia [24]), muy difundida por su uso en plataformas de amplia distribución como Arduino. La cámara OV7670 es un módulo diseñado por la casa Omnivisión y está especialmente diseñada para una interfaz TTL. Se caracteriza por un bajo requerimiento de tensión de alimentación, máximo 3.3 voltios, lo cual resulta adecuado para el prototipo que se quiere llevar a cabo. El control del funcionamiento de la cámara se realiza bajo el estándar SCCB compatible con una interfaz I2C.

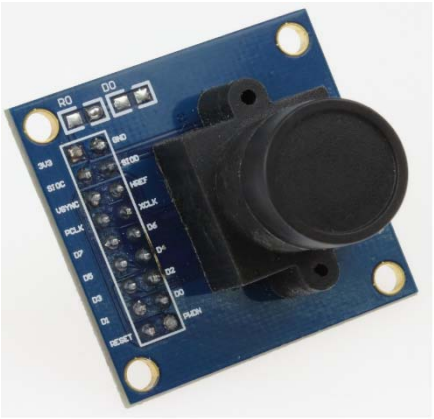


Figura 39: Cámara OV7670

Del estudio que hemos realizado (ver Anexo I) se deduce que se requerirían cuatro patillas para el manejo de la cámara, incorporar al sistema una RAM de 64K y añadir otro dispositivo End Device.

Como conclusión del proyecto, se ha desarrollado un prototipo totalmente operativo que permite estudiar las posibilidades que presentan los sensores inalámbricos alimentados con energía solar; la plataforma desarrollada permitirá en el futuro desarrollar aplicaciones autónomas de consumo muy reducido y gran autonomía. Además, el prototipo se ha conectado a una plataforma on-line del campo del "internet de las cosas" (https://thingspeak.com/) que permite monitorizar los datos con cualquier dispositivo y desde cualquier lugar (basta únicamente una conexión a internet)

Por otro lado, dado el período de desarrollo del proyecto, no se han podido comprobar los algoritmos de predicción meteorológica en todas las estaciones del año (trabajo en curso). Sin embargo, el hecho de realizar predicciones locales mediante medidas locales favorece la posibilidad de que los algoritmos implementados superen a las predicciones que realizan los servicios meteorológicos utilizando modelos meteorológicos automatizados (como ocurre con http://www.aemet.es/).

9. Bibliografía

- Referencia sobre el kit ez430-RF2500:

[1] Thomas Watteyne, "*eZWSN: Experimenting with Wireless Sensor Networks using eZ430-RF2500*", Rice University, Houston, Texas, 2012
http://cnx.org/exports/35441aa1-e93c-4c5b-82fe-bc1aa16bf6fd@10.6.pdf/ezwsn-experimenting-with-wireless-sensor-networks-using-the-ez430-rf2500-10.6.pdf

- Referencia sobre el microcontrolador MSP430:

[2] Rafael Charro, "*Volverine. MSP430FR59xx Family. Texas Instrument* ", Seminario proporcionado por Arrow Argentina, 2012
http://www.sase.com.ar/2012/files/2012/10/Microcontroladores-ultra-low-power.pdf

- Referencias sobre los protocolos de transmisión de información:

[3] Texas Instrument, "*TI Low Power RF*", Designer's Guide to LPRF, 2010
http://www.ti.com/lit/sg/slya020a/slya020a.pdf

[4] "*La etapa de radiofrecuencia*", Universidad de Andalucía
http://ocw.unia.es/ciencias-tecnologicas/tecnologia-del-ocio/materiales-basicos-folder/html/B3_UD02/la_etapa_de_radiofrecuencia.html

[5] "*Funciones básicas del chip radio*", Universidad de Andalucía
http://ocw.unia.es/ciencias-tecnologicas/tecnologia-del-ocio/materiales-basicos-folder/html/B3_UD02/funciones_bsicas_del_chip_radio.html/skinless_view

[6] Bluegiga technologies, "*Bluetooth Smart. Getting started*", Introduction to Bluetooth Smart technology and to Bluegiga's Bluetooth Smart products, Finland, 2012
http://www.farnell.com/datasheets/1691422.pdf

[7] Cristian López Arancibia y Carlos Cofré Vásquez, "*Estándar de comunicación inalámbrica Bluetooth*", Universidad técnica Federico Santa María. Dpto de Electrónica, Chile, 2012
http://profesores.elo.utfsm.cl/~agv/elo322/1s12/project/reports/LopezCofre/BLUETOOTH.pdf

[8] Phil Smith,"Comparing Low-Power Wireless Technologies", Artículo desarrollado en colaboración de Convergence Promotion LLC, 2011
http://www.digikey.com/es/articles/techzone/2011/aug/comparing-low-power-wireless-technologies

- Referencias sobre los sensores:

LDR:

[9] Via Satelital, "*Apuntes de electrónica básica. Fotorresistencia y otros*", Proyectos de electrónica
http://www.viasatelital.com/proyectos_electronicos/fotoresistencia.htm

[10] Carlos Barbado Herrera y otros. "*Tecnología 4º Eso*", Editorial SM-FSM, Madrid

Sensor humedad:

[11] Antonio Serna Ruiz y otros, "*Guía práctica de sensores*", Creaciones Copyright, 2010

[12] A.Cromer, "FÍsica para las ciencias de la vida", Editorial Reverté, Barcelona, 2007

[13] Francesc Xavier Martínez de Osés, "*Meteorología aplicada a la navegación*", Ediciones de la Universidad Politécnica de Cataluña, Barcelona, 2006

[14] Metas y Metrólogos asociados, "*Sensores de humedad.Tipos y aplicaciones*", La Guía Metas, México, 2008
http://www.metas.com.mx/guiametas/la-guia-metas-08-05-sensores-de-humedad.pdf

Sensor de presión:

[15] National Instruments, "*Fundamentos de medidas de presión*", Nota técnica, Madrid, 2011
http://www.ni.com/white-paper/13034/es/

[16] Metas y Metrólogos asociados, " *Presión atmosférica, presión barométrica y altitud Conceptos y aplicaciones*", La Guía Metas, México, 2005
http://www.metas.com.mx/guiametas/La-Guia-MetAs-05-02-presion-atmosferica.pdf

General:

[17] Ramón Pallás Areny, "*Sensores y Acondicionadores de señal*", Ediciones Marcombo, Barcelona, 2003

[18] John G.Webster y Halit Eren, "*Measurement,Instrumentation and Sensors Handbook*", Editorial CRC PRESS,U.S.A, 2014

- Referencias sobre la lectura de un puerto serie con java y de los ficheros .jar

[19] Steven Lim, "*Read from a COM port using Java program*", Tutorial
http://www.java-samples.com/showtutorial.php?tutorialid=11

[20] Henry Poom, "*Serial Communication in Java with Example Program*", Informatics Blog, 2011
https://www.henrypoon.com/blog/2011/01/01/serial-communication-in-java-with-example-program/

[21]Luis Fernando García Llinás, "*Todo lo básico que debería saber sobre Programación básica orientada a objetos en Java*", Ediciones Uninorte, Colombia, 2010

[22] Oracle, "*Java documentation. Path and ClassPath*", Tutorial
https://docs.oracle.com/javase/tutorial/essential/environment/paths.html

- Referencias sobre los algoritmos de predicción:

[23] Victoriano Reinoso Gomez, "*Meteorology and Climatology*", Atlantic International University, Honolulu, Hawaii, 2008
http://escuelamaritima.com/media/noticias/4422_file.pdf

[24] Asociación Ars Creatio,"*Nociones básicas de meteorología*", Proyecto Sorel, Alicante
http://arscreatio.com/_sorel/PDF/NOCIONES_BASICA_METEO.PDF

- Referencia sobre la cámara de estudio:

[25] Jorge Aparicio," Hacking the OV7670 camera module.SCCB cheat sheet inside", Electronics and Robotics Blog, 2012
http://embeddedprogrammer.blogspot.com.es/2012/07/hacking-ov7670-camera-module-sccb-cheat.html

- Referencia sobre la aplicación ThingSpeak:

[26] Google Developers, "*Code Examples*", Servicios ofrecidos por google para desarrollar aplicaciones, 2015
https://developers.google.com/chart/interactive/docs/examples

TERCERA PARTE

ANEXOS

A. Datasheet

a. Herramienta de desarrollo ez430-RF2500

eZ430-RF2500 Development Tool

User's Guide

Literature Number: SLAU227E
September 2007–Revised April 2009

Contents

List of Figures

List of Tables

Read This First

If You Need Assistance

Support for MSP430 devices and the eZ430-RF2500 is provided by the Texas Instruments Product Information Center (PIC). Contact information for the PIC can be found on the TI web site at www.ti.com. Additional device-specific information can be found on the MSP430 web site at www.ti.com/msp430 and www.ti.com/ez430-rf.

Note: IAR Embedded Workbench® KickStart is supported by Texas Instruments.

Although IAR Embedded Workbench KickStart is a product of IAR, Texas Instruments provides support for KickStart. Therefore, please do not request support for KickStart from IAR. Please consult all provided documentation with KickStart before requesting assistance.

We Would Like to Hear from You

If you have any comments, feedback, or suggestions, please let us know by contacting us at support@ti.com.

Trademarks

SimpliciTI is a trademark of Texas Instruments.

All other trademarks are the property of their respective owners.

eZ430-RF2500 Development Tool

1 eZ430-RF2500 Overview. Wireless Made Easy.

The eZ430-RF2500 is a complete USB-based MSP430 wireless development tool providing all the hardware and software to evaluate the MSP430F2274 microcontroller and CC2500 2.4-GHz wireless transceiver.

The eZ430-RF2500 uses the IAR Embedded Workbench Integrated Development Environment (IDE) or Code Composer Essentials (CCE) to write, download, and debug an application. The debugger is unobtrusive, allowing the user to run an application at full speed with both hardware breakpoints and single stepping available while consuming no extra hardware resources.

The eZ430-RF2500T target board is an out-of-the box wireless system that may be used with the USB debugging interface, as a stand-alone system with or without external sensors, or may be incorporated into an existing design.

The new USB debugging interface enables the eZ430-RF2500 to remotely send and receive data from a PC using the MSP430 Application UART.

eZ430-RF2500 features:

- USB debugging and programming interface featuring a driverless installation and application backchannel
- 21 available development pins
- Highly integrated, ultra-low-power MSP430 MCU with 16-MHz performance
- Two general-purpose digital I/O pins connected to green and red LEDs for visual feedback
- Interruptible push button for user feedback

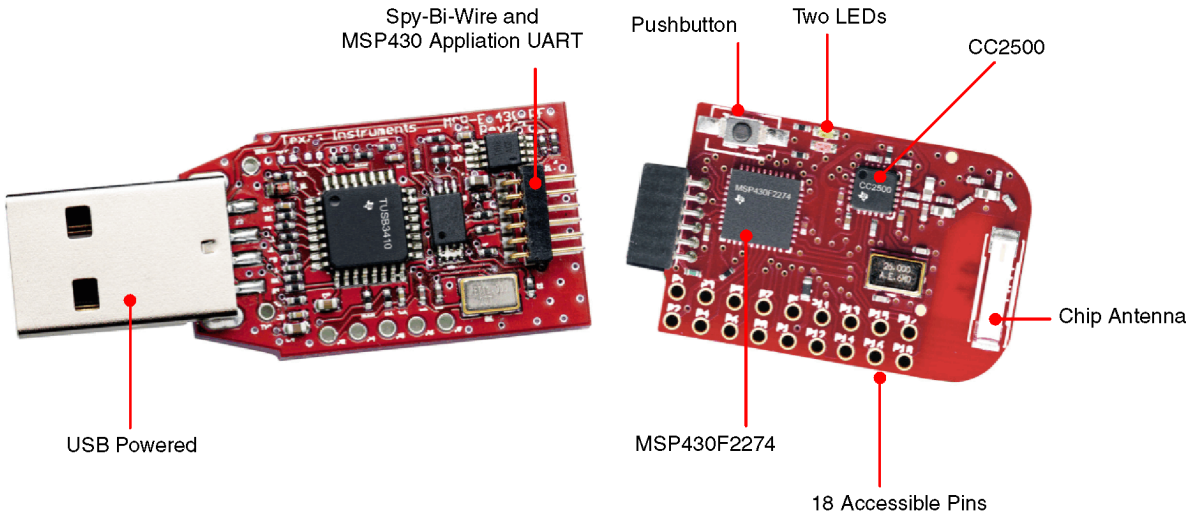


Figure 1. eZ430-RF2500

Figure 2. eZ430-RF2500 Battery Board

2 Kit Contents, eZ430-RF2500

- The hardware includes:
  - Two eZ430-RF2500T target boards
  - One eZ430-RF USB debugging interface
  - One AAA battery pack with expansion board (batteries included)
- One MSP430 Development Tool CD-ROM containing documentation and new development software for eZ430-RF2500:
  - MSP430x2xx Family User's Guide, SLAU144
  - eZ430-RF2500 User's Guide, SLAU227
  - Code Composer Essentials (CCE), SLAC063
  - IAR Embedded Workbench (KickStart Version), SLAC050
  - eZ430-RF2500 Sensor Monitor (Code and Visualizer), SLAC139

Note: Visit Texas Instrument's website for the latest versions: www.ti.com/msp430

3 Developing With eZ430-RF2500T Target Board

The eZ430-RF2500 can be used as a stand-alone development tool. Additionally, the eZ430-RF2500T target board also may be detached from the debugging interface and integrated into another design by removing the plastic enclosure. The target board features an MSP430F2274 and most of its pins are easily accessible. The pins are shown in Figure 3 and described in Table 1 and Table 2:

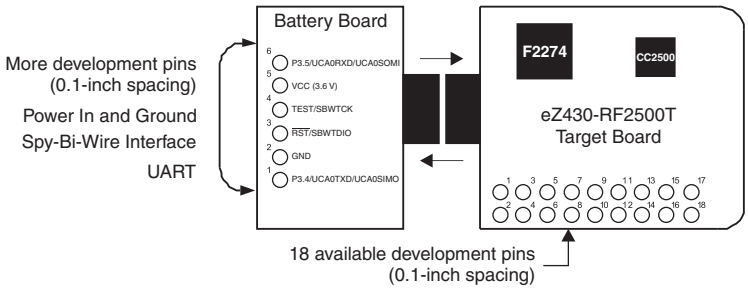


Figure 3. eZ430-RF2500 Development Tool

Table 1. eZ430-RF2500T Target Board Pinouts

Pin	Function	Description
1	GND	Ground reference
2	VCC	Supply voltage
3	P2.0 / ACLK / A0 / OA0I0	General-purpose digital I/O pin / ACLK output / ADC10, analog input A0
4	P2.1 / TAINCLK / SMCLK / A1 / A0O	General-purpose digital I/O pin / ADC10, analog input A1 Timer_A, clock signal at INCLK, SMCLK signal output
5	P2.2 / TA0 / A2 / OA0I1	General-purpose digital I/O pin / ADC10, analog input A2 Timer_A, capture: CCI0B input/BSL receive, compare: OUT0 output
6	P2.3 / TA1 / A3 / VREF– / VeREF– / OA1I1 / OA1O	General-purpose digital I/O pin / Timer_A, capture: CCI1B input, compare: OUT1 output / ADC10, analog input A3 / negative reference voltage output/input
7	P2.4 / TA2 / A4 / VREF+ / VeREF+ / OA1I0	General-purpose digital I/O pin / Timer_A, compare: OUT2 output / ADC10, analog input A4 / positive reference voltage output/input
8	P4.3 / TB0 / A12 / OA0O	General-purpose digital I/O pin / ADC10 analog input A12 / Timer_B, capture: CCI0B input, compare: OUT0 output
9	P4.4 / TB1 / A13 / OA1O	General-purpose digital I/O pin / ADC10 analog input A13 / Timer_B, capture: CCI1B input, compare: OUT1 output
10	P4.5 / TB2 / A14 / OA0I3	General-purpose digital I/O pin / ADC10 analog input A14 / Timer_B, compare: OUT2 output
11	P4.6 / TBOUTH / A15 / OA1I3	General-purpose digital I/O pin / ADC10 analog input A15 / Timer_B, switch all TB0 to TB3 outputs to high impedance
12	GND	Ground reference
13	P2.6 / XIN (GDO0)	General-purpose digital I/O pin / Input terminal of crystal oscillator
14	P2.7 / XOUT (GDO2)	General-purpose digital I/O pin / Output terminal of crystal oscillator
15	P3.2 / UCB0SOMI / UCB0SCL	General-purpose digital I/O pin USCI_B0 slave out/master in when in SPI mode, SCL I2C clock in I2C mode
16	P3.3 / UCB0CLK / UCA0STE	General-purpose digital I/O pin USCI_B0 clock input/output / USCI_A0 slave transmit enable
17	P3.0 / UCB0STE / UCA0CLK / A5	General-purpose digital I/O pin / USCI_B0 slave transmit enable / USCI_A0 clock input/output / ADC10, analog input A5
18	P3.1 / UCB0SIMO / UCB0SDA	General-purpose digital I/O pin / USCI_B0 slave in/master out in SPI mode, SDA I2C data in I2C mode

Table 2. Battery Board Pinouts

Pin	Function	Description
1	P3.4 / UCA0TXD / UCA0SIMO	General-purpose digital I/O pin / USCI_A0 transmit data output in UART mode (UART communication from 2274 to PC), slave in/master out in SPI mode
2	GND	Ground reference
3	$\overline{\text{RST}}$ / SBWTDIO	Reset or nonmaskable interrupt input Spy-Bi-Wire test data input/output during programming and test
4	TEST / SBWTCK	Selects test mode for JTAG pins on Port1. The device protection fuse is connected to TEST. Spy-Bi-Wire test clock input during programming and test
5	VCC (3.6V)	Supply voltage
6	P3.5 / UCA0RXD / UCA0SOMI	General-purpose digital I/O pin / USCI_A0 receive data input in UART mode (UART communication from 2274 to PC), slave out/master in when in SPI mode

4 Specifications

MSP430F2274

- 16-MIPS performance
- 200-ksps 10-bit SAR ADC
- Two built-in operational amplifiers
- Watchdog timer, 16-bit Timer_A3 and Timer_B3
- USCI module supporting UART/LIN, (2) SPI, I2C, or IrDA
- Five low-power modes drawing as little as 700 nA in standby

PARAMETER	MIN	TYP	MAX	UNIT
OPERATING CONDITIONS				
Operating supply voltage	1.8		3.6	V
Operating free-air temperature range	-40		85	°C
CURRENT CONSUMPTION				
Active mode at 1 MHz, 2.2 V		270	390	µA
Standby mode		0.7	1.4	µA
Off mode with RAM retention		0.1	0.5	µA
OPERATING FREQUENCY				
VCC ≥ 3.3 V			16	MHz

CC2500

- 2.4-GHz radio-frequency (RF) transceiver
- Programmable data rate up to 500 kbps
- Low current consumption

PARAMETER	CONDITION	MIN	TYP	MAX	UNIT
OPERATING CONDITIONS					
Operating supply voltage		1.8		3.6	V
CURRENT CONSUMPTION					
RX input signal at the sensitivity limit, 250 kbps	Optimized current		16.6		mA
	Optimized sensitivity		18.8		mA
RX input signal 30 dB above the sensitivity limit, 250 kbps	Optimized current		13.3		mA
	Optimized sensitivity		15.7		mA
Current consumption TX (0 dBm)			21.2		mA
Current consumption TX (-12 dBm)			11.1		mA
RF CHARACTERISTICS					
Frequency range		2400		2483.5	MHz
Data rate (programmable)		1.2		500	kbps
Output power (programmable)		-30		0	dBm
Sensitivity, 10 kbps	Optimized current, 2-FSK, 230-kHz RX filter bandwidth, 1% PER		-99		dBm
	Optimized sensitivity		-101		dBm
Sensitivity, 250 kbps	Optimized current, 500-kHz RX filter bandwidth, 1% PER		-87		dBm
	Optimized sensitivity		-89		dBm

5 Supported Devices

The eZ430-RF USB debugging interface may be used as a standard Flash Emulation Tool through its Spy-Bi-Wire interface. The eZ430-RF USB debugging interface supports the following MSP430 families:

- MSP430F20xx
- MSP430F22xx

The connector on the USB debugging interface is backward compatible with the eZ430-F2013 and T2012 target boards.

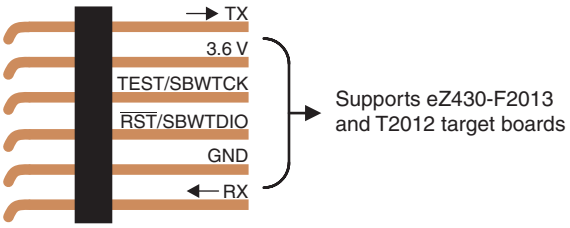


Figure 4. eZ430-RF2500 USB Debugging Interface 6-Pin Male Header

6 MSP430 Application UART

The eZ430-RF USB debugging interface features a back channel MSP430 Application UART that may be used independently of a debug session. This allows the user to transfer serial data to a terminal window at a fixed rate of 9600 bps with no flow control. See Figure 5 for typical settings.

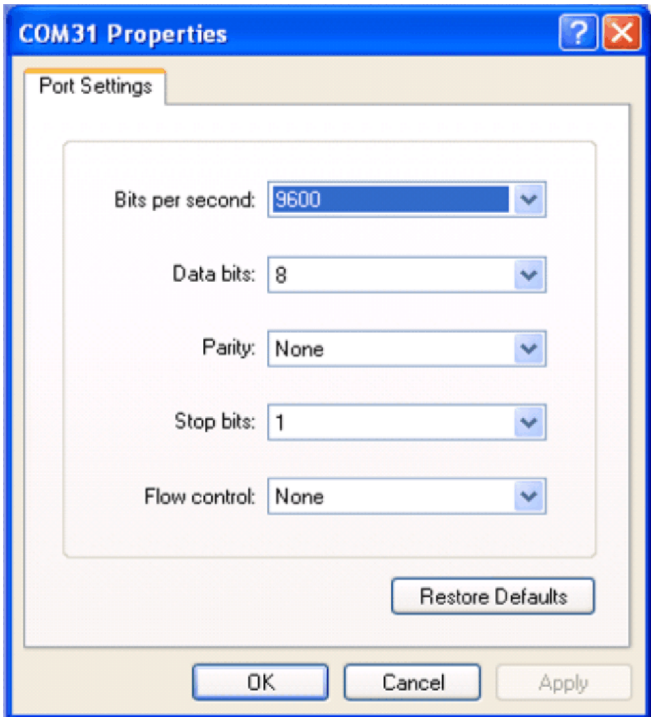


Figure 5. 9600 bps With No Flow Control

Check the Device Manager for COM port assignment of the MSP430 Application UART. For more details, see Section 14.

7 Software Installation

The CD-ROM includes two different development software tools for the MSP430: IAR Embedded Workbench KickStart and Code Composer Essentials (CCE). The term "KickStart" refers to the limited version of Embedded Workbench that allows up to 4 KB of C-code compilation. The included CCE is also limited, but it allows up to 16 KB of code compilation. The full version of CCE Pro offers unlimited code compilation and can be purchased from www.ti.com/msp430.

7.1 Installing the IDE

1. Insert the eZ430-RF2500 CD-ROM into the computer. The eZ430-RF2500 start page should automatically display. If it does not display, use a browser to open "index.htm", which is located in the root directory of the eZ430-RF2500 CD-ROM.

   The eZ430-RF2500 is compatible with Windows XP and Windows Vista.
2. Select **Software → IAR Workbench KickStart / Code Composer Essentials**.
3. Respond to the prompts to install the software. The installation procedure installs the IDE and TI files.

7.2 Installing the Sensor Monitor Visualizer Application

1. Select **Software → Sensor Monitor Visualizer** and follow the instructions.
2. Choose the installation path for the software.
3. Open the eZ430-RF2500 Sensor Monitor using the shortcut installed on the desktop.

8 Hardware Installation

1. Insert the eZ430-RF into USB port. The debugging interface automatically installs itself.
2. When prompted for the software for the MSP430 Application UART, allow Windows to **Install the software automatically**. This is only possible if either IAR KickStart R4.64 (or higher) or the Sensor Monitor Visualizer has already been installed. For more information, see Section 14.

9 SimpliciTI™ Network Protocol

The SimpliciTI network protocol is a proprietary, low-power radio-frequency (RF) protocol targeting simple, small RF networks (<100 nodes). The SimpliciTI network protocol is designed for easy implementation with minimal microcontroller resource requirements. The protocol runs out of the box on TI's MSP430 ultra-low-power microcontrollers and multiple RF transceivers.

Small low-power RF networks typically contain battery-operated devices, which require long battery life, low data rate, and low duty cycle, and have a limited number of nodes talking directly to each other. With the SimpliciTI network protocol, MCU resource requirements are minimal, resulting in lower system cost for low-power RF networks. More complex mesh networks that need routing typically require 10ıthe program memory and RAM to implement.

Despite the modest resources required, SimpliciTI network protocol supports End Devices in a peer-to-peer network topology, the option to use an Access Point to store and forward messages, and Range Extenders to extend the range of the network up to four hops. Future releases will add more sophisticated features such as frequency agility, an ETSI-compliant listen-before-talk discipline, and a software security routine for message encryption.

The SimpliciTI network protocol supports a wide range of low-power applications including alarm and security (smoke detectors, glass breakage detectors, carbon monoxide sensors, and light sensors), automated meter reading (gas meters and water meters), home automation (appliances, garage door openers, and environmental devices), and active RFID.

The SimpliciTI network protocol is provided as source code under a free license without royalties.

Developers are encouraged to adapt the protocol to their own specific application needs. For information on compatibility, updates, and the latest version of the SimpliciTI protocol, visit www.ti.com/simpliciti.

The eZ430-RF2500 demonstration application uses the SimpliciTI protocol to demonstrate a temperature sensor network application that provides a starting point to develop a wireless applications.

10 Demo – eZ430-RF2500 Sensor Monitor

eZ430-RF2500 is preloaded with a wireless temperature sensor network firmware and may be reprogrammed at any time. This network consists of an Access Point that measures its own temperature and also wirelessly receives temperature measurements from End Devices. End Devices measure their temperature once per second and then enter a low-power mode to reduce battery usage. The Access Point transmits all measured data to the PC through the UART backchannel. The included PC Sensor Monitor Visualizer provides a demonstration of the eZ430-RF2500 using the SimpliciTI protocol across a star network. In the PC Sensor Monitor Visualizer, the center node is the Access Point and the attached bubbles are the End Devices. The PC application displays the temperature of both the End Devices and Access Point. Additionally, the PC application is capable of simulating distance from its access point when the End Devices are moved. The number of End Devices can be expanded by adding more target boards in the star network as seen in Figure 6.

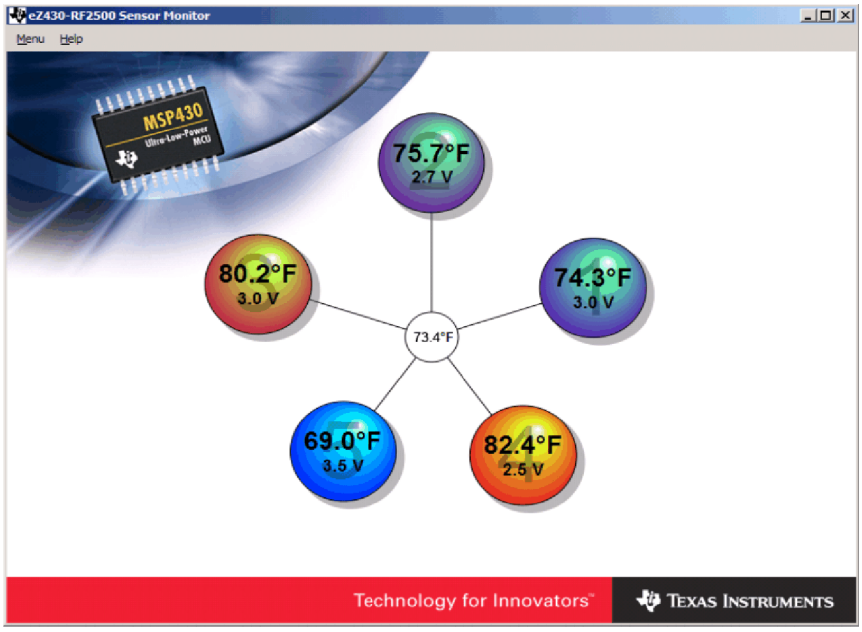


Figure 6. eZ430-RF2500 Sensor Monitor

10.1 Demo Hardware Setup

1. Connect the eZ430-RF2500 to a USB port on the PC.
2. Connect the second eZ430-RF2500T target board to the battery board. Insert the jumper on the board to power up the device.

10.2 Demo Firmware Download

The following steps describe how to update the demo application firmware on the eZ430-RF2500 target boards and are not required out of the box.

1. Open IAR Workbench KickStart.
2. Select Open Existing Workspace, and browse for the demo application workspace (*.eww) file. The project is available on the CD or at http://www.ti.com/lit/zip/slac139.
3. To download demo firmware, follow steps 3a for Access Point firmware and 3b for End Device firmware.
   a. Right click on **Access Point** project in the workspace and click **Set as Active** as shown in Figure 7.
   b. Right click on **End Device** project in the workspace and click **Set as Active**.

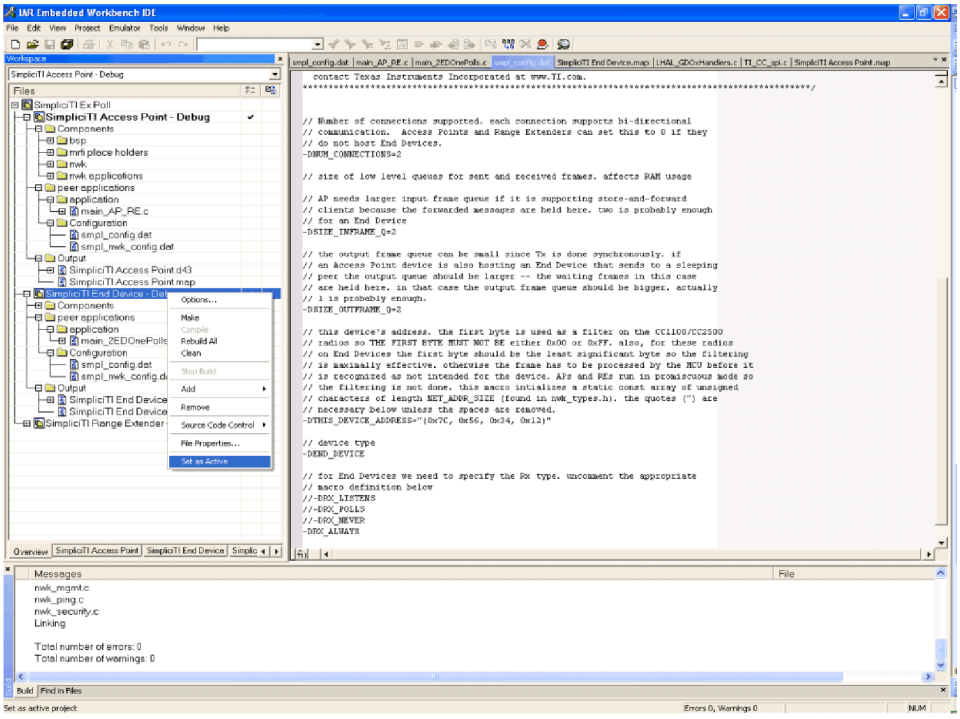


Figure 7. IAR Embedded Workbench KickStart Workspace

4. Select **Project** → **Debug in IAR** to download the code for the target boards.
5. Select **Debug** → **Go** to start running code while in debug mode.
6. Select **Debug** → **Stop Debugging** exits the debug mode while leaving the target board executing code.

10.3 Demo Software GUI Setup

1. Ensure the Access Point is connected to the PC.
2. Apply power to the End Device.
3. Launch eZ430-RF2500 Sensor Monitor Demo Visualizer. After installation, a shortcut is placed on the desktop. It is available on the CD and online at http://www.ti.com/lit/zip/slac139. The application should automatically display End Devices when in range.

10.4 Demo Options

1. Go to **Menu** → **Settings**.
2. Under the settings menu, the demo application is capable of displaying values in Celsius or Fahrenheit.
3. Checking the box **Disable Animations** disables the dynamic distance change, thus decreasing CPU processing on PC.
4. See the demo application help file by clicking **Help** for more detailed options.

11 Suggested Reading

The primary sources of MSP430 information are the device-specific data sheets and user's guides. The most up-to-date versions of the user's guide documents available at the time of production have been provided on the CD-ROM included with this tool. The most current information is found at www.ti.com/msp430. Information specific to the eZ430-RF2500 development tool can be found at www.ti.com/ez430-rf.

MSP430 device user's guides and the FET user's guide may be accessed from the main page on the CD-ROM under the User's Guides section. The FET user's guide includes detailed information on setting up a project for the MSP430 using IAR.

Documents describing the IAR tools (Workbench/C-SPY, the assembler, the C compiler, the linker, and the library) are located in common\doc and 430\doc. The documents are in PDF format. Supplements to the documents (i.e., the latest information) are available in HTML format within the same directories. 430\doc\readme_start.htm provides a convenient starting point for navigating the IAR documentation.

12 Frequently Asked Questions (FAQ)

1. **Does the eZ430-RF2500 support fuse blow?**

   The eZ430-RF USB debugging interface lacks the JTAG security fuse-blow capability. To ensure firmware security on devices going to production, the USB Flash Emulation Tool or the Gang Programmer, which include the fuse-blow feature, are recommended.

2. **What is the voltage supplied to the eZ430-RF2500T target board from the debugging interface?**

   The eZ430-RF USB debugging interface supplies a regulated 3.6 V to the eZ430-RF2500T target board.

3. **Can other programming tools interface to the eZ430-RF2500T target board?**

   The eZ430-RF2500T target board works with any programming tool supporting the 2-wire Spy-Bi-Wire interface. Both the MSP430 USB FET (MSP-FET430UIF) and the Gang Programmer (MSP-GANG430) support these devices. See MSP-FET430 Flash Emulation Tool User's Guide (SLAU138) for details on using MSP430 USB FET and the Gang Programmer for a 2-wire Spy-Bi-Wire interface.

4. **What versions of IAR Embedded Workbench and Code Composer Essentials are supported?**

   The eZ430-RF2500 hardware is supported by IAR Embedded Workbench KickStart Release 4.64 (IAR 3.42F) and Code Composer Essentials v2.03 (SP3) or higher.

   At the time of print, CCE is currently not supported by the SimpliciTI protocol or the Sensor Monitor Demo. Please check the TI web site for updates.

5. **What are the part numbers for the connectors between the eZ430-RF USB debugger and the eZ430-RF2500T target board?**

   Header: Mill-Max 850-10-006-20-001000

   Socket: Mill-Max 851-93-006-20-001000

6. **Where can I obtain more information about the 2.4-GHz chip antenna?**

   Part Number: 7488910245

   Würth Electronik Group: www.we-online.com

7. **I am not able to select the MSP430 Application UART, cannot receive data, or the demo app doesn't appear to change.**

   Ensure that the Application UART driver is correctly installed. This is done by either running the installer for the Sensor Monitor Visualizer or IAR KickStart 3.42F or higher and following the directions in Section 14.

   To determine if the driver is correctly installed:

   a. Plug in the eZ430-RF USB debugging interface.
   b. Right click **My Computer** and select **Properties.**
   c. Select the **Hardware** tab and click on **Device Manager**.
   d. Under **Ports (COM & LPT)** should be an entry for "MSP430 Application UART (COM xx)".

   If the entry is there, but no characters are received, restart the PC.

   If the Application UART is not listed, please install the driver by following the instructions in Section 14.

8. **When trying to compiler the Sensor Monitor Demo project in IAR, I receive the following error:**

   Error[e117]: Incompatible runtime models. Module ISR specifies that '__rt_version' must be '3', but module LHAL_GDOxHandlers has the value '2'

   Please use the latest version of the demo source code off the web (http://www.ti.com/lit/zip/slac139) and use IAR KickStart 4.x.

   Early versions of the demo code included a precompiled version of the SimpliciTI library for IAR 3.x. IAR 4.x changes the calling conventions, which returns Error[e117] when trying to build libraries for an older version of the compiler.

9. **What kind of range should I expect to get with the eZ430-RF2500?**

   Based on practical rage testing with one node connected to a PC and the other node connected to the battery board, we have measured indoor line-of-sight range of more than 50 meters. This range can be significantly affected by the orientation of the boards and the environment. Note that the eZ430-RF2500 target board was designed to optimize for factor and does not focus on maximizing RF range. Please visit the TI website for additional reference designs and antenna options.

10. **Why is my battery board different than in the documentation?**

    Since introduction, the eZ430RF-2500 battery board was slightly modified. The connections and function remain the same.

13 eZ430-RF2500 Schematics

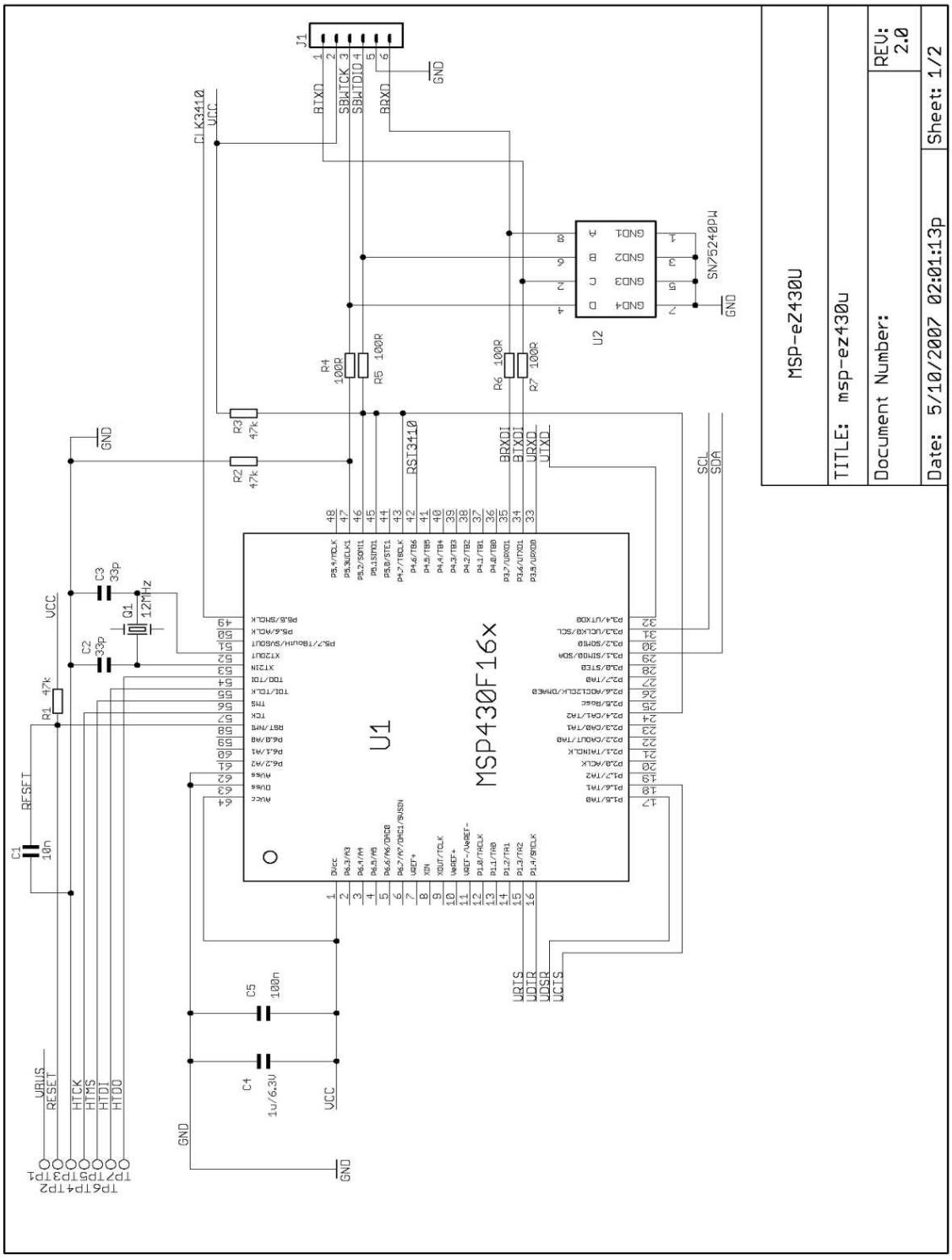


Figure 8. eZ430-RF, USB Debugging Interface, Schematic

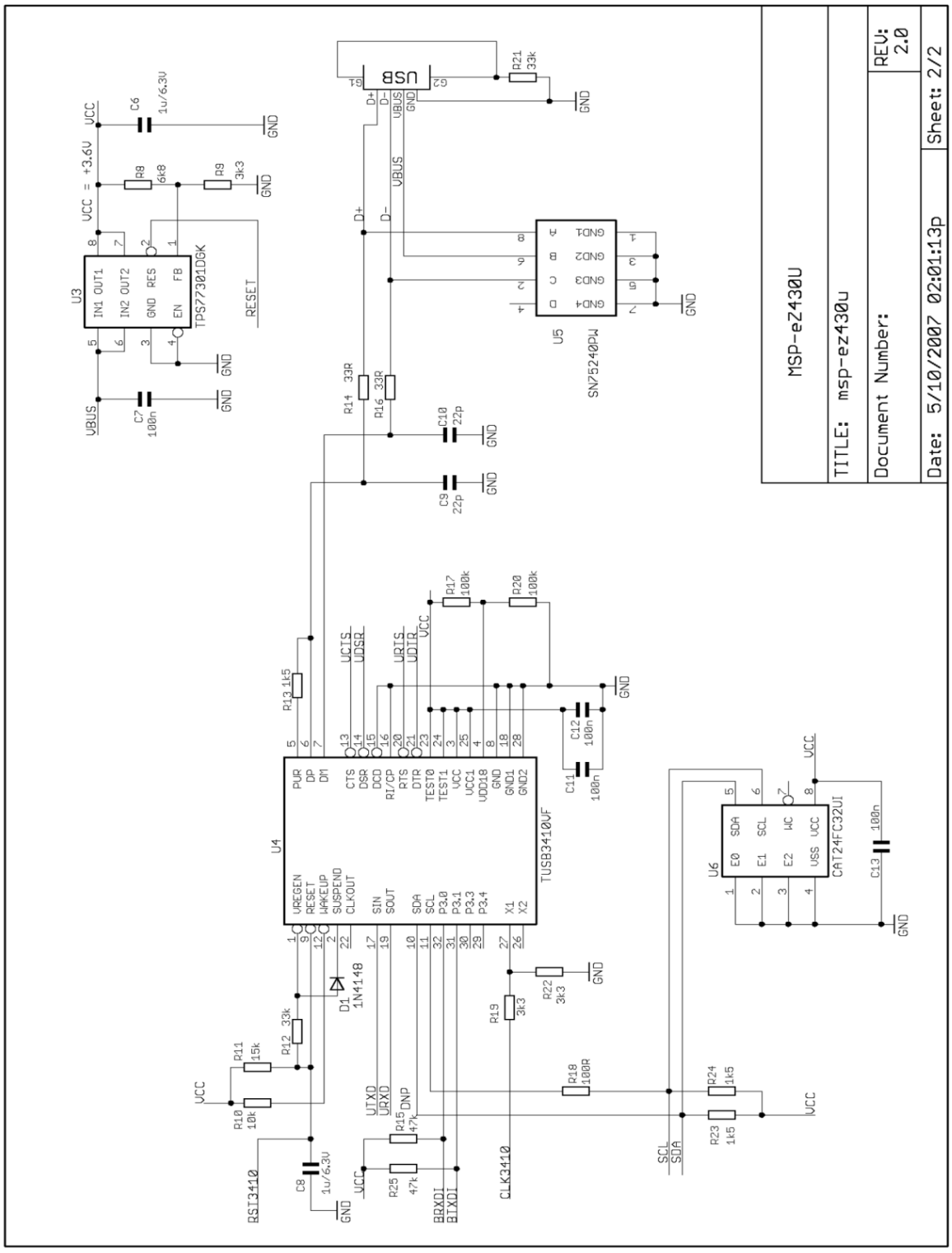


Figure 9. eZ430-RF, USB Debugging Interface, Schematic

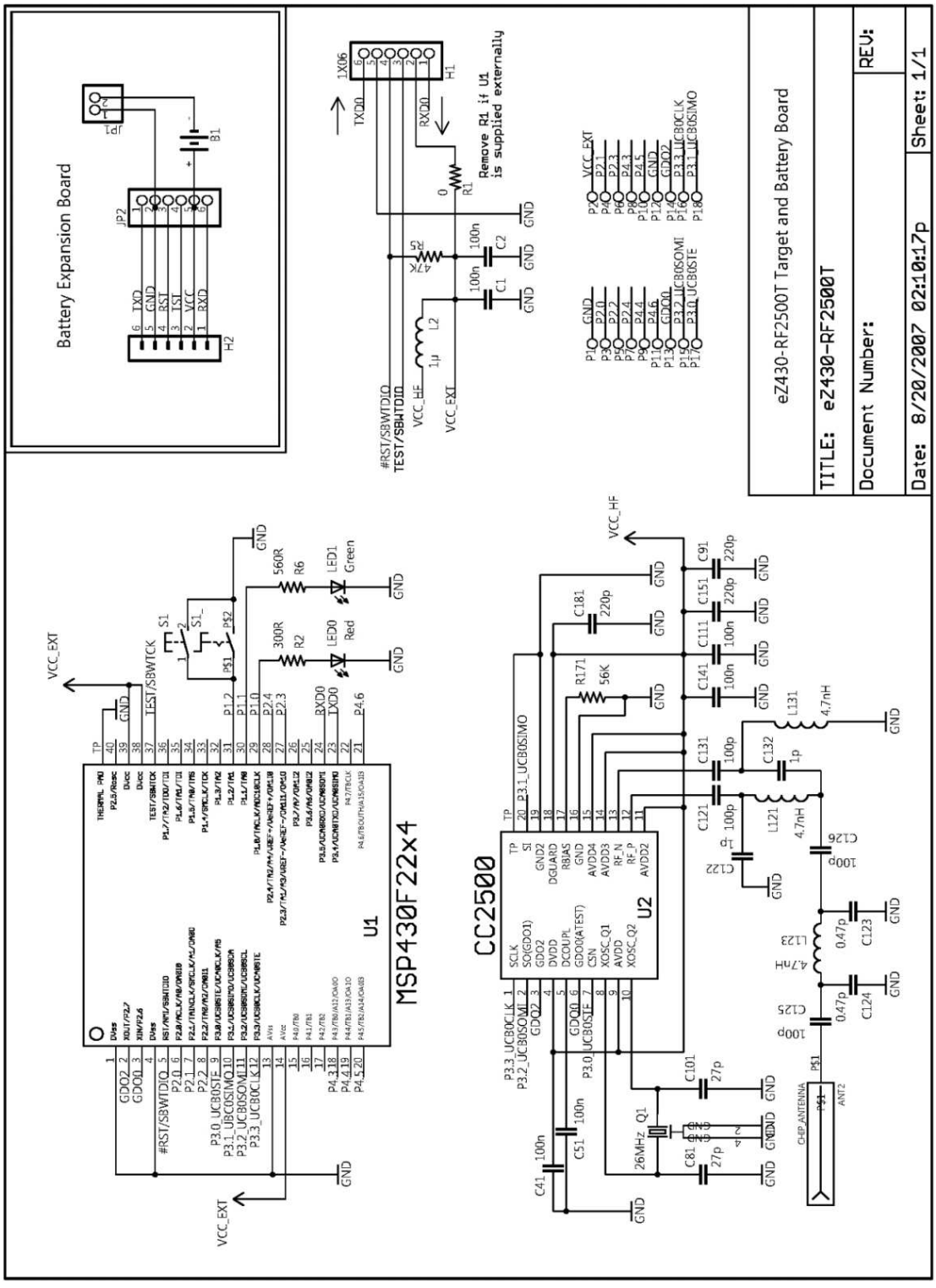


Figure 10. eZ430-RF2500T, Target Board and Battery Board, Schematic

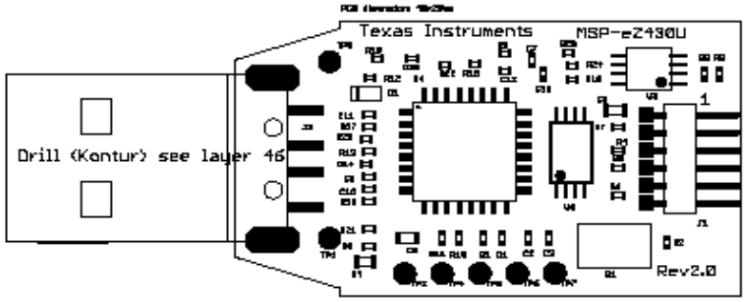


Figure 11. eZ430-RF, USB Debugger, PCB Components Layout

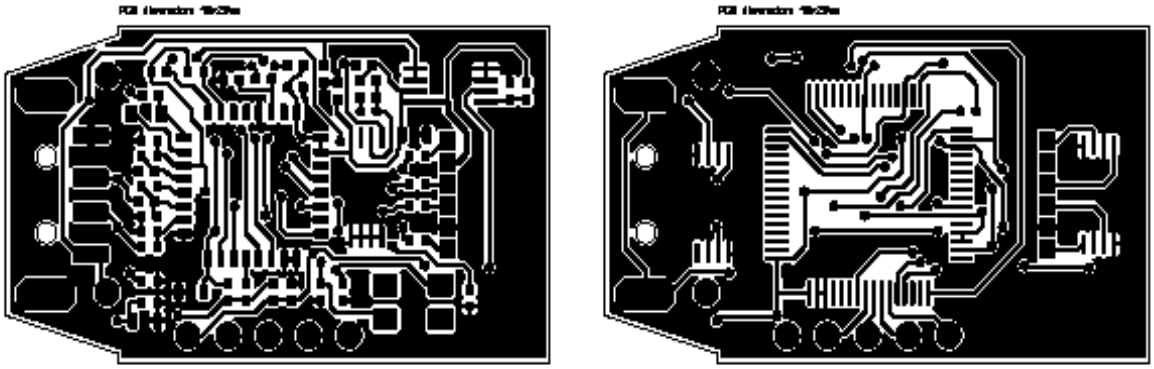


Figure 12. eZ430-RF, USB Debugger, PCB Layout

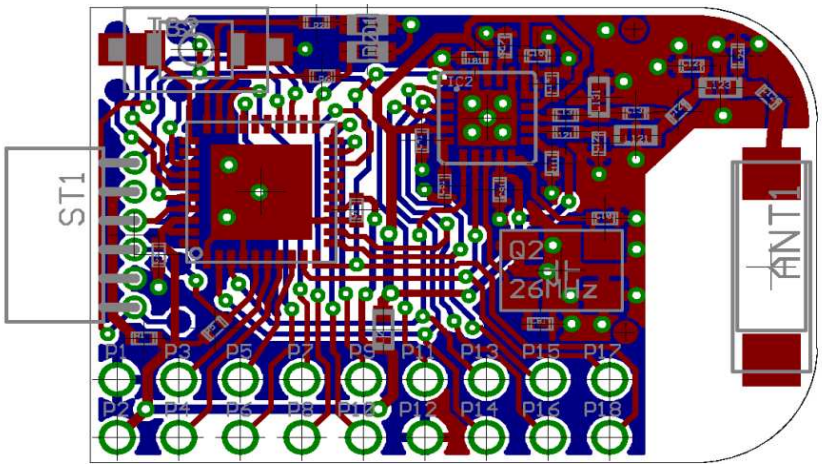


Figure 13. eZ430-RF2500T, Target Board, PCB Layout

14 Detailed Hardware Installation Guide

1. Insert the eZ430-RF2500 CD-ROM into a CD drive.
2. Install the eZ430-RF2500 Sensor Monitor Demo Visualizer. It is available on the CD and online at http://www.ti.com/lit/zip/slac139. This installs the necessary drivers on your system.
3. Insert the eZ430-RF2500 into a USB port of the PC.
4. Windows should recognize the new hardware as **Texas Instruments MSP-FET430UIF** (see Figure 14). Windows should automatically install the drivers for the MSP-FET430UIF as a HID tool.

Figure 14. Windows XP Hardware Recognition

5. Windows recognizes another new hardware driver to be installed called **MSP430 Application UART** (see Figure 15).

Figure 15. Windows XP Hardware Recognition for MSP430 Application UART

Note: This Installation Step is **Optional**. The USB debugging interface works without the MSP430 Application UART as long as (R4.64 or newer) IAR Workbench is used.

6. The Found New Hardware Wizard opens a dialog window. Select No, not this time and click Next (see Figure 16).

Figure 16. Windows XP Found New Hardware Wizard

7. Select **Install the software automatically (Recommended)** (see Figure 17), if IAR KickStart R4.64 or higher has already been installed.

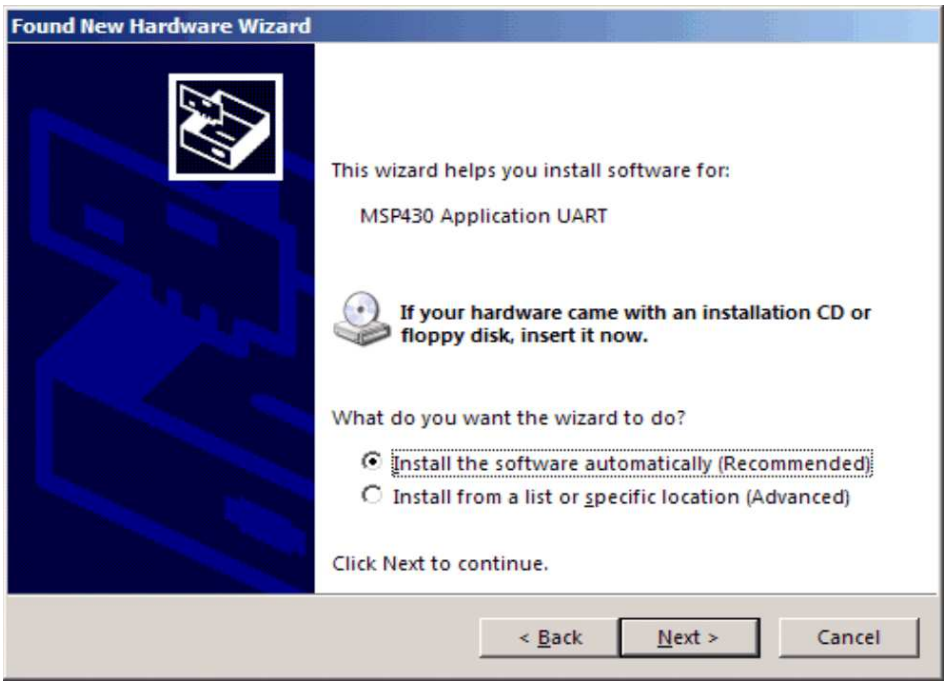


Figure 17. Windows XP Hardware Wizard

8. The Wizard should find the appropriate driver for a Windows XP system; it shows a warning that Microsoft did not certify the driver. The drivers have been tested exhaustively, and this warning may be ignored. Click **Continue Anyway** (see Figure 18).

Figure 18. Windows XP Warning

9. The Wizard continues to install the driver and then provides notification when it has finished the installation of the software.

10. The eZ430-RF2500 is now installed and ready to use. The assigned COM port for the MSP430 Application UART is shown in the Windows Device Manager (see Figure 19).

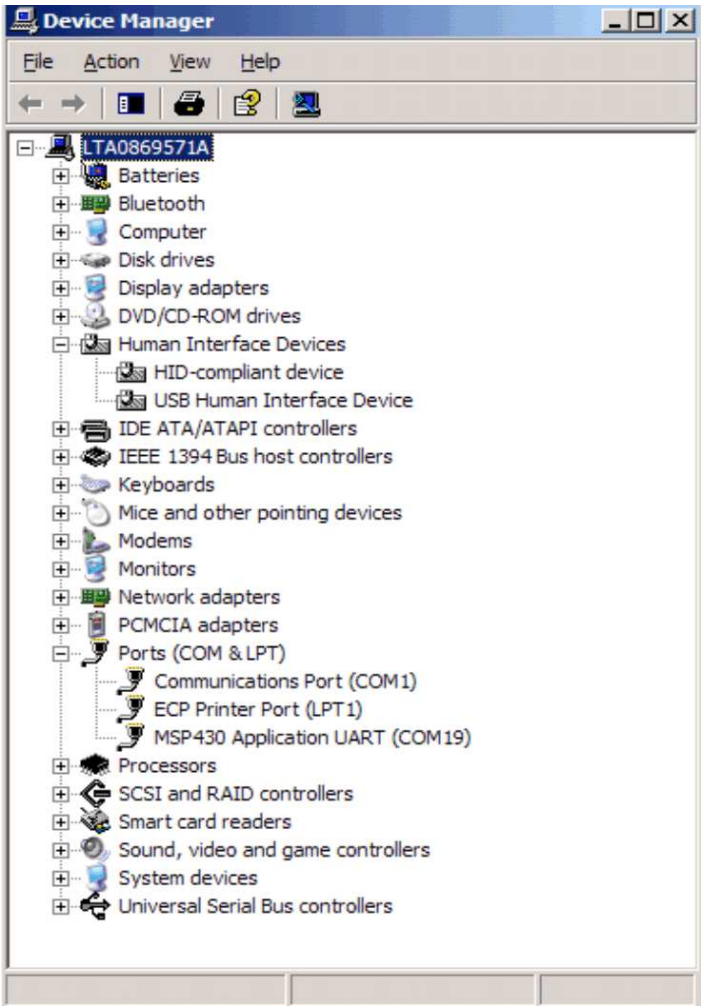


Figure 19. Device Manager

15 IAR Workbench Compatibility Guide

Note: In this document, "IAR version" refers to the IAR compiler version. This can be obtained by clicking **Help** → **About** → **Product Info**.

IAR KickStart version 3.42F (FET_R4.64)

- Minimum version compatible with eZ430-RF USB debugging interface board
- Compatible with eZ430-RF2500 Sensor Monitor demo v1.00

IAR KickStart version 4.09A+ (FET_R5.10+)

- Compatible with eZ430-RF USB debugging interface board
- Compatible with SimpliciTI libraries 1.0.3+
- Compatible with eZ430-RF2500 Sensor Monitor demo v1.02+

EVALUATION BOARD/KIT IMPORTANT NOTICE

Texas Instruments (TI) provides the enclosed product(s) under the following conditions:

This evaluation board/kit is intended for use for **ENGINEERING DEVELOPMENT, DEMONSTRATION, OR EVALUATION PURPOSES ONLY** and is not considered by TI to be a finished end-product fit for general consumer use. Persons handling the product(s) must have electronics training and observe good engineering practice standards. As such, the goods being provided are not intended to be complete in terms of required design-, marketing-, and/or manufacturing-related protective considerations, including product safety and environmental measures typically found in end products that incorporate such semiconductor components or circuit boards. This evaluation board/kit does not fall within the scope of the European Union directives regarding electromagnetic compatibility, restricted substances (RoHS), recycling (WEEE), FCC, CE or UL, and therefore may not meet the technical requirements of these directives or other related directives.

Should this evaluation board/kit not meet the specifications indicated in the User's Guide, the board/kit may be returned within 30 days from the date of delivery for a full refund. THE FOREGOING WARRANTY IS THE EXCLUSIVE WARRANTY MADE BY SELLER TO BUYER AND IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED, IMPLIED, OR STATUTORY, INCLUDING ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE.

The user assumes all responsibility and liability for proper and safe handling of the goods. Further, the user indemnifies TI from all claims arising from the handling or use of the goods. Due to the open construction of the product, it is the user's responsibility to take any and all appropriate precautions with regard to electrostatic discharge.

EXCEPT TO THE EXTENT OF THE INDEMNITY SET FORTH ABOVE, NEITHER PARTY SHALL BE LIABLE TO THE OTHER FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES.

TI currently deals with a variety of customers for products, and therefore our arrangement with the user **is not exclusive.**

TI assumes **no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein.**

Please read the User's Guide and, specifically, the Warnings and Restrictions notice in the User's Guide prior to handling the product. This notice contains important safety information about temperatures and voltages. For additional information on TI's environmental and/or safety programs, please contact the TI application engineer or visit www.ti.com/esh.

No license is granted under any patent right or other intellectual property right of TI covering or relating to any machine, process, or combination in which such TI products or services might be or are used.

FCC Warning

This evaluation board/kit is intended for use for **ENGINEERING DEVELOPMENT, DEMONSTRATION, OR EVALUATION PURPOSES ONLY** and is not considered by TI to be a finished end-product fit for general consumer use. It generates, uses, and can radiate radio frequency energy and has not been tested for compliance with the limits of computing devices pursuant to part 15 of FCC rules, which are designed to provide reasonable protection against radio frequency interference. Operation of this equipment in other environments may cause interference with radio communications, in which case the user at his own expense will be required to take whatever measures may be required to correct this interference.

EVM WARNINGS AND RESTRICTIONS

It is important to operate this EVM within the input voltage range of 1.8 V to 3.6 V and the output voltage range of 1.8 V to 3.6 V.

Exceeding the specified input range may cause unexpected operation and/or irreversible damage to the EVM. If there are questions concerning the input range, please contact a TI field representative prior to connecting the input power.

Applying loads outside of the specified output range may result in unintended operation and/or possible permanent damage to the EVM. Please consult the EVM User's Guide prior to connecting any load to the EVM output. If there is uncertainty as to the load specification, please contact a TI field representative.

During normal operation, some circuit components may have case temperatures greater than 60°C. The EVM is designed to operate properly with certain components above 60°C as long as the input and output ranges are maintained. These components include but are not limited to linear regulators, switching transistors, pass transistors, and current sense resistors. These types of devices can be identified using the EVM schematic located in the EVM User's Guide. When placing measurement probes near these devices during operation, please be aware that these devices may be very warm to the touch.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Broadband	www.ti.com/broadband
DSP	dsp.ti.com	Digital Control	www.ti.com/digitalcontrol
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Military	www.ti.com/military
Logic	logic.ti.com	Optical Networking	www.ti.com/opticalnetwork
Power Mgmt	power.ti.com	Security	www.ti.com/security
Microcontrollers	microcontroller.ti.com	Telephony	www.ti.com/telephony
RFID	www.ti-rfid.com	Video & Imaging	www.ti.com/video
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265

A. Datasheet

b. Microcontrolador

Introduction

This chapter describes the architecture of the MSP430.

Topic **Page**

1.1 Architecture

The MSP430 incorporates a 16-bit RISC CPU, peripherals, and a flexible clock system that interconnect using a von-Neumann common memory address bus (MAB) and memory data bus (MDB) (see Figure 1-1). Partnering a modern CPU with modular memory-mapped analog and digital peripherals, the MSP430 offers solutions for demanding mixed-signal applications.

Key features of the MSP430x2xx family include:

- Ultralow-power architecture extends battery life
  - 0.1 µA RAM retention
  - 0.8 µA real-time clock mode
  - 250 µA/MIPS active
- High-performance analog ideal for precision measurement
  - Comparator-gated timers for measuring resistive elements
- 16-bit RISC CPU enables new applications at a fraction of the code size.
  - Large register file eliminates working file bottleneck
  - Compact core design reduces power consumption and cost
  - Optimized for modern high-level programming
  - Only 27 core instructions and seven addressing modes
  - Extensive vectored-interrupt capability
- In-system programmable Flash permits flexible code changes, field upgrades and data logging

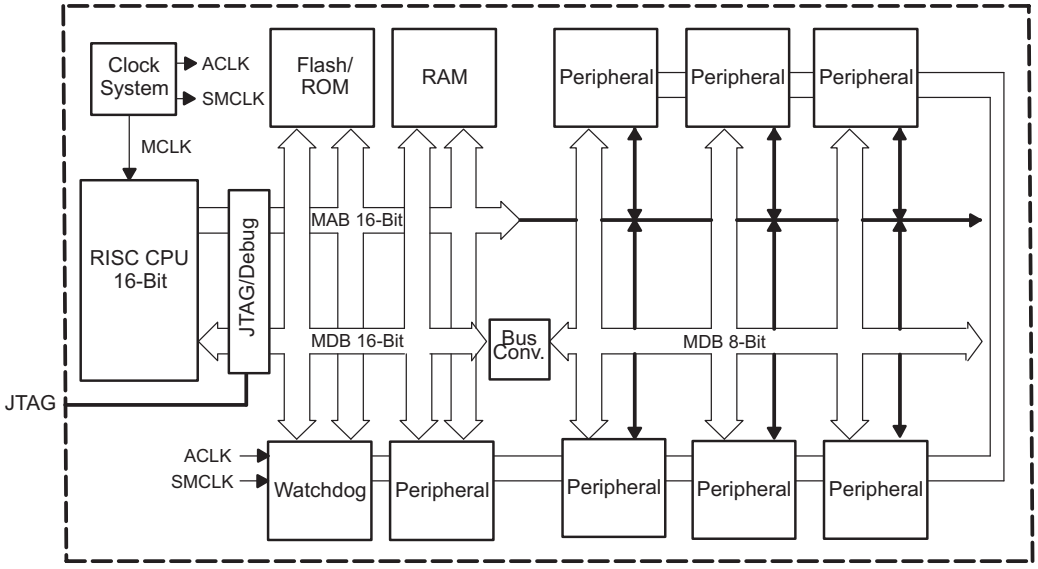


Figure 1-1. MSP430 Architecture

1.2 Flexible Clock System

The clock system is designed specifically for battery-powered applications. A low-frequency auxiliary clock (ACLK) is driven directly from a common 32-kHz watch crystal. The ACLK can be used for a background real-time clock self wake-up function. An integrated high-speed digitally controlled oscillator (DCO) can source the master clock (MCLK) used by the CPU and high-speed peripherals. By design, the DCO is active and stable in less than 2 µs at 1 MHz. MSP430-based solutions effectively use the high-performance 16-bit RISC CPU in very short bursts.

- Low-frequency auxiliary clock = Ultralow-power stand-by mode
- High-speed master clock = High performance signal processing

1.3 Embedded Emulation

Dedicated embedded emulation logic resides on the device itself and is accessed via JTAG using no additional system resources.

The benefits of embedded emulation include:

- Unobtrusive development and debug with full-speed execution, breakpoints, and single-steps in an application are supported.
- Development is in-system subject to the same characteristics as the final application.
- Mixed-signal integrity is preserved and not subject to cabling interference.

1.4 Address Space

The MSP430 von-Neumann architecture has one address space shared with special function registers (SFRs), peripherals, RAM, and Flash/ROM memory as shown in Figure 1-2. See the device-specific data sheets for specific memory maps. Code access are always performed on even addresses. Data can be accessed as bytes or words.

The addressable memory space is currently 128 KB.

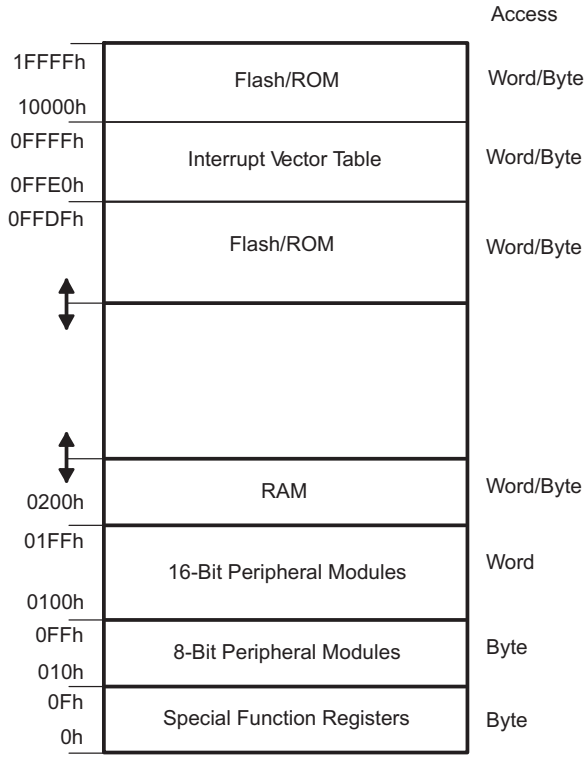


Figure 1-2. Memory Map

1.4.1 Flash/ROM

The start address of Flash/ROM depends on the amount of Flash/ROM present and varies by device. The end address for Flash/ROM is 0x0FFFF for devices with less that 60KB of Flash/ROM. Flash can be used for both code and data. Word or byte tables can be stored and used in Flash/ROM without the need to copy the tables to RAM before using them.

The interrupt vector table is mapped into the upper 16 words of Flash/ROM address space, with the highest priority interrupt vector at the highest Flash/ROM word address (0x0FFFE).

1.4.2 RAM

RAM starts at 0200h. The end address of RAM depends on the amount of RAM present and varies by device. RAM can be used for both code and data.

1.4.3 Peripheral Modules

Peripheral modules are mapped into the address space. The address space from 0100 to 01FFh is reserved for 16-bit peripheral modules. These modules should be accessed with word instructions. If byte instructions are used, only even addresses are permissible, and the high byte of the result is always 0.

The address space from 010h to 0FFh is reserved for 8-bit peripheral modules. These modules should be accessed with byte instructions. Read access of byte modules using word instructions results in unpredictable data in the high byte. If word data is written to a byte module only the low byte is written into the peripheral register, ignoring the high byte.

1.4.4 Special Function Registers (SFRs)

Some peripheral functions are configured in the SFRs. The SFRs are located in the lower 16 bytes of the address space, and are organized by byte. SFRs must be accessed using byte instructions only. See the device-specific data sheets for applicable SFR bits.

1.4.5 Memory Organization

Bytes are located at even or odd addresses. Words are only located at even addresses as shown in Figure 1-3. When using word instructions, only even addresses may be used. The low byte of a word is always an even address. The high byte is at the next odd address. For example, if a data word is located at address xxx4h, then the low byte of that data word is located at address xxx4h, and the high byte of that word is located at address xxx5h.

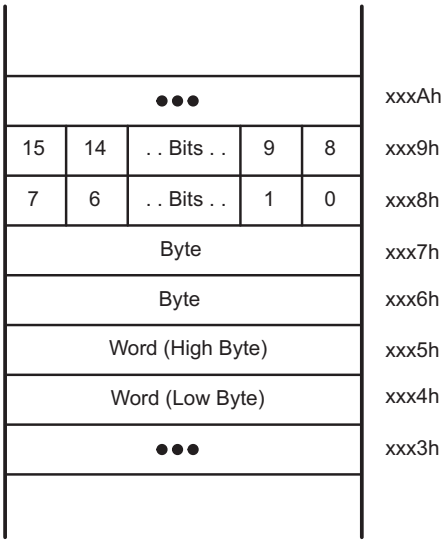


Figure 1-3. Bits, Bytes, and Words in a Byte-Organized Memory

1.5 MSP430x2xx Family Enhancements

Table 1-1 highlights enhancements made to the MSP430x2xx family. The enhancements are discussed fully in the following chapters, or in the case of improved device parameters, shown in the device-specific data sheet.

Table 1-1. MSP430x2xx Family Enhancements

Subject	Enhancement
Reset	• Brownout reset is included on all MSP430x2xx devices. • PORIFG and RSTIFG flags have been added to IFG1 to indicate the cause of a reset. • An instruction fetch from the address range 0x0000 - 0x01FF will reset the device.
Watchdog Timer	• All MSP430x2xx devices integrate the Watchdog Timer+ module (WDT+). The WDT+ ensures the clock source for the timer is never disabled.
Basic Clock System	• The LFXT1 oscillator has selectable load capacitors in LF mode. • The LFXT1 supports up to 16-MHz crystals in HF mode. • The LFXT1 includes oscillator fault detection in LF mode. • The XIN and XOUT pins are shared function pins on 20- and 28-pin devices. • The external R_{OSC} feature of the DCO not supported on some devices. Software should not set the LSB of the BCSCTL2 register in this case. See the device-specific data sheet for details. • The DCO operating frequency has been significantly increased. • The DCO temperature stability has been significantly improved.
Flash Memory	• The information memory has 4 segments of 64 bytes each. • SegmentA is individually locked with the LOCKA bit. • All information if protected from mass erase with the LOCKA bit. • Segment erases can be interrupted by an interrupt. • Flash updates can be aborted by an interrupt. • Flash programming voltage has been lowered to 2.2 V • Program/erase time has been reduced. • Clock failure aborts a flash update.
Digital I/O	• All ports have integrated pullup/pulldown resistors. • P2.6 and P2.7 functions have been added to 20- and 28- pin devices. These are shared functions with XIN and XOUT. Software must not clear the P2SELx bits for these pins if crystal operation is required.
Comparator_A	• Comparator_A has expanded input capability with a new input multiplexer.
Low Power	• Typical LPM3 current consumption has been reduced almost 50% at 3 V. DCO startup time has been significantly reduced.
Operating frequency	• The maximum operating frequency is 16 MHz at 3.3 V.
BSL	• An incorrect password causes a mass erase. • BSL entry sequence is more robust to prevent accidental entry and erasure.

System Resets, Interrupts, and Operating Modes

This chapter describes the MSP430x2xx system resets, interrupts, and operating modes.

Topic **Page**

2.1 System Reset and Initialization

The system reset circuitry shown in Figure 2-1 sources both a power-on reset (POR) and a power-up clear (PUC) signal. Different events trigger these reset signals and different initial conditions exist depending on which signal was generated.

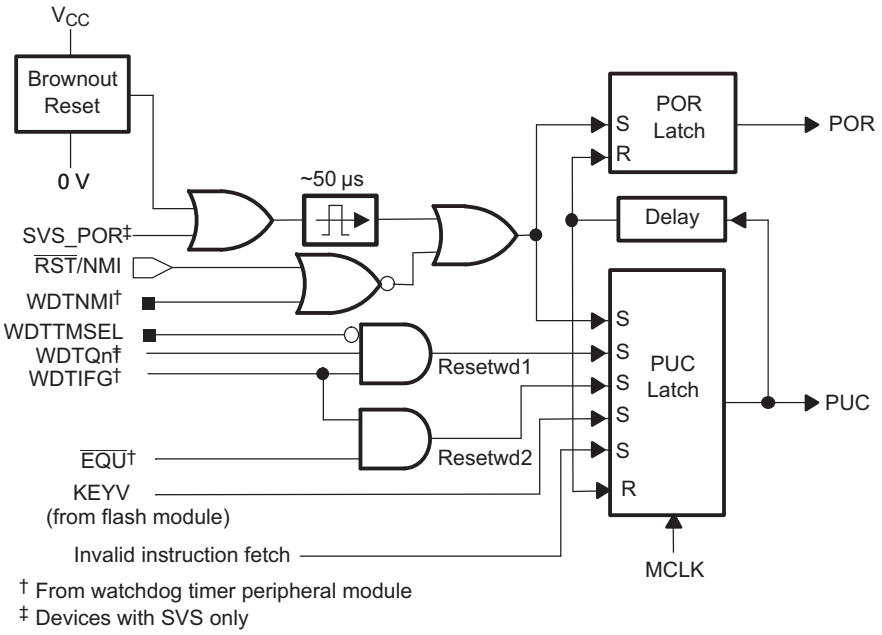


Figure 2-1. Power-On Reset and Power-Up Clear Schematic

A POR is a device reset. A POR is only generated by the following three events:

- Powering up the device
- A low signal on the $\overline{\text{RST}}$/NMI pin when configured in the reset mode
- An SVS low condition when PORON = 1.

A PUC is always generated when a POR is generated, but a POR is not generated by a PUC. The following events trigger a PUC:

- A POR signal
- Watchdog timer expiration when in watchdog mode only
- Watchdog timer security key violation
- A Flash memory security key violation
- A CPU instruction fetch from the peripheral address range 0h to 01FFh

2.1.1 Brownout Reset (BOR)

The brownout reset circuit detects low supply voltages such as when a supply voltage is applied to or removed from the V_{CC} terminal. The brownout reset circuit resets the device by triggering a POR signal when power is applied or removed. The operating levels are shown in Figure 2-2.

The POR signal becomes active when V_{CC} crosses the $V_{CC(start)}$ level. It remains active until V_{CC} crosses the $V_{(B\_IT+)}$ threshold and the delay $t_{(BOR)}$ elapses. The delay $t_{(BOR)}$ is adaptive being longer for a slow ramping V_{CC}. The hysteresis $V_{hys(B\_IT-)}$ ensures that the supply voltage must drop below $V_{(B\_IT-)}$ to generate another POR signal from the brownout reset circuitry.

CPU

This chapter describes the MSP430 CPU, addressing modes, and instruction set.

Topic **Page**

3.1 CPU Introduction

The CPU incorporates features specifically designed for modern programming techniques such as calculated branching, table processing, and the use of high-level languages such as C. The CPU can address the complete address range without paging.

The CPU features include:

- RISC architecture with 27 instructions and 7 addressing modes.
- Orthogonal architecture with every instruction usable with every addressing mode.
- Full register access including program counter, status registers, and stack pointer.
- Single-cycle register operations.
- Large 16-bit register file reduces fetches to memory.
- 16-bit address bus allows direct access and branching throughout entire memory range.
- 16-bit data bus allows direct manipulation of word-wide arguments.
- Constant generator provides six most used immediate values and reduces code size.
- Direct memory-to-memory transfers without intermediate register holding.
- Word and byte addressing and instruction formats.

The block diagram of the CPU is shown in Figure 3-1.

CPUX

This chapter describes the extended MSP430X 16-bit RISC CPU with 1-MB memory access, its addressing modes, and instruction set. The MSP430X CPU is implemented in all MSP430 devices that exceed 64-KB of address space.

Topic **Page**

4.1 CPU Introduction

The MSP430X CPU incorporates features specifically designed for modern programming techniques such as calculated branching, table processing and the use of high-level languages such as C. The MSP430X CPU can address a 1-MB address range without paging. In addition, the MSP430X CPU has fewer interrupt overhead cycles and fewer instruction cycles in some cases than the MSP430 CPU, while maintaining the same or better code density than the MSP430 CPU. The MSP430X CPU is backward compatible with the MSP430 CPU.

The MSP430X CPU features include:

- RISC architecture
- Orthogonal architecture
- Full register access including program counter, status register and stack pointer
- Single-cycle register operations
- Large register file reduces fetches to memory
- 20-bit address bus allows direct access and branching throughout the entire memory range without paging
- 16-bit data bus allows direct manipulation of word-wide arguments
- Constant generator provides the six most often used immediate values and reduces code size
- Direct memory-to-memory transfers without intermediate register holding
- Byte, word, and 20-bit address-word addressing

The block diagram of the MSP430X CPU is shown in Figure 4-1.

Basic Clock Module+

The basic clock module+ provides the clocks for MSP430x2xx devices. This chapter describes the operation of the basic clock module+ of the MSP430x2xx device family.

Topic **Page**

5.1 Basic Clock Module+ Introduction

The basic clock module+ supports low system cost and ultralow power consumption. Using three internal clock signals, the user can select the best balance of performance and low power consumption. The basic clock module+ can be configured to operate without any external components, with one external resistor, with one or two external crystals, or with resonators, under full software control.

The basic clock module+ includes two, three or four clock sources:

- LFXT1CLK: Low-frequency/high-frequency oscillator that can be used with low-frequency watch crystals or external clock sources of 32768 Hz or with standard crystals, resonators, or external clock sources in the 400-kHz to 16-MHz range.
- XT2CLK: Optional high-frequency oscillator that can be used with standard crystals, resonators, or external clock sources in the 400-kHz to 16-MHz range.
- DCOCLK: Internal digitally controlled oscillator (DCO).
- VLOCLK: Internal very low power, low frequency oscillator with 12-kHz typical frequency.

Three clock signals are available from the basic clock module+:

- ACLK: Auxiliary clock. ACLK is software selectable as LFXT1CLK or VLOCLK. ACLK is divided by 1, 2, 4, or 8. ACLK is software selectable for individual peripheral modules.
- MCLK: Master clock. MCLK is software selectable as LFXT1CLK, VLOCLK, XT2CLK (if available on-chip), or DCOCLK. MCLK is divided by 1, 2, 4, or 8. MCLK is used by the CPU and system.
- SMCLK: Sub-main clock. SMCLK is software selectable as LFXT1CLK, VLOCLK, XT2CLK (if available on-chip), or DCOCLK. SMCLK is divided by 1, 2, 4, or 8. SMCLK is software selectable for individual peripheral modules.

The block diagram of the basic clock module+ in the MSP430F2xx devices is shown in Figure 5-1.

The block diagram of the basic clock module+ in the MSP430AFE2xx devices is shown in Figure 5-2.

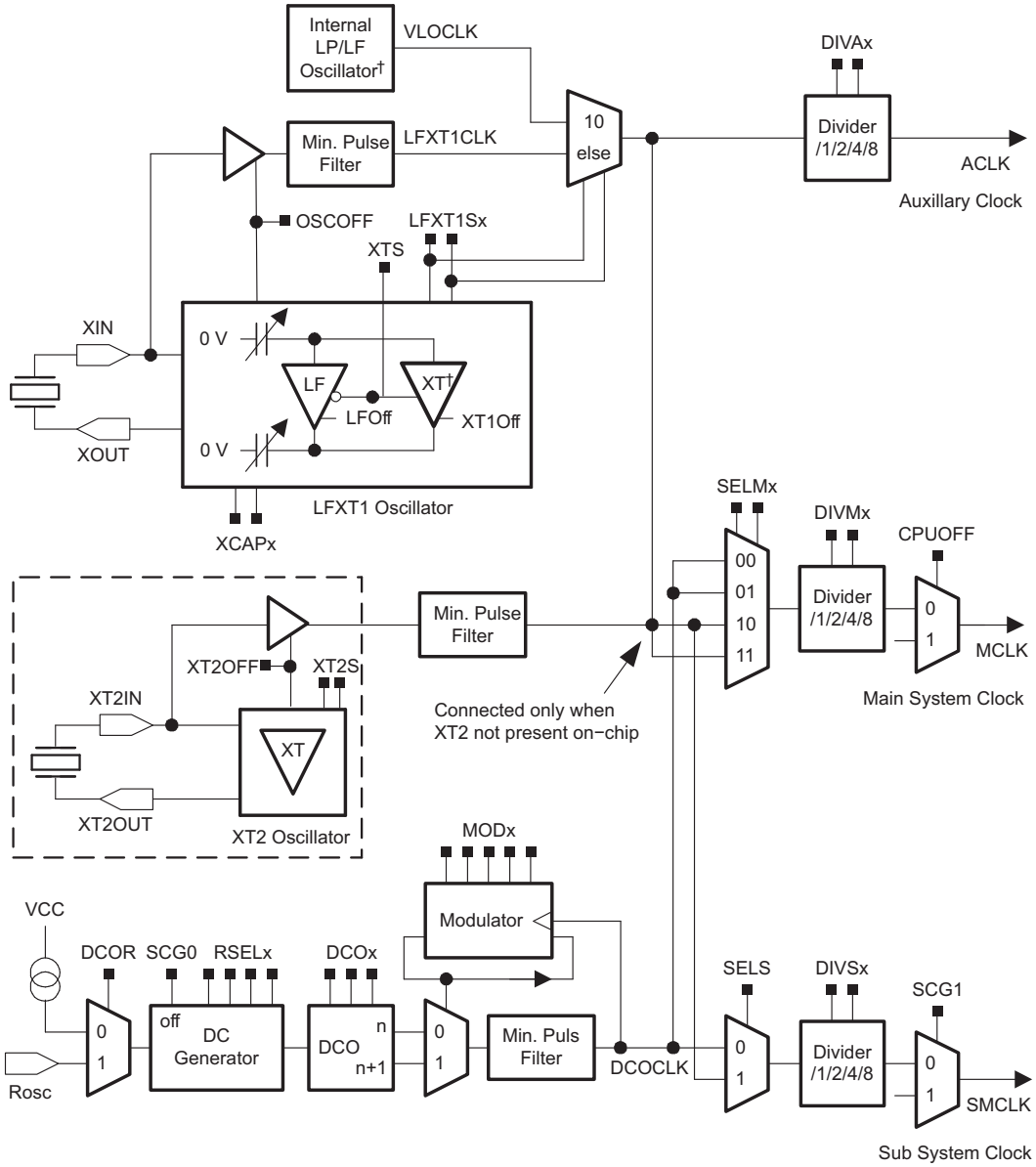


Figure 5-1. Basic Clock Module+ Block Diagram − MSP430F2xx

NOTE: † Device-Specific Clock Variations

Not all clock features are available on all MSP430x2xx devices:
MSP430G22x0: LFXT1 is not present, XT2 is not present, ROSC is not supported.

MSP430F20xx, MSP430G2xx1, MSP430G2xx2, MSP430G2xx3: LFXT1 does not support HF mode, XT2 is not present, ROSC is not supported.
MSP430x21x1: Internal LP/LF oscillator is not present, XT2 is not present, ROSC is not supported.
MSP430x21x2: XT2 is not present.
MSP430F22xx, MSP430x23x0: XT2 is not present.

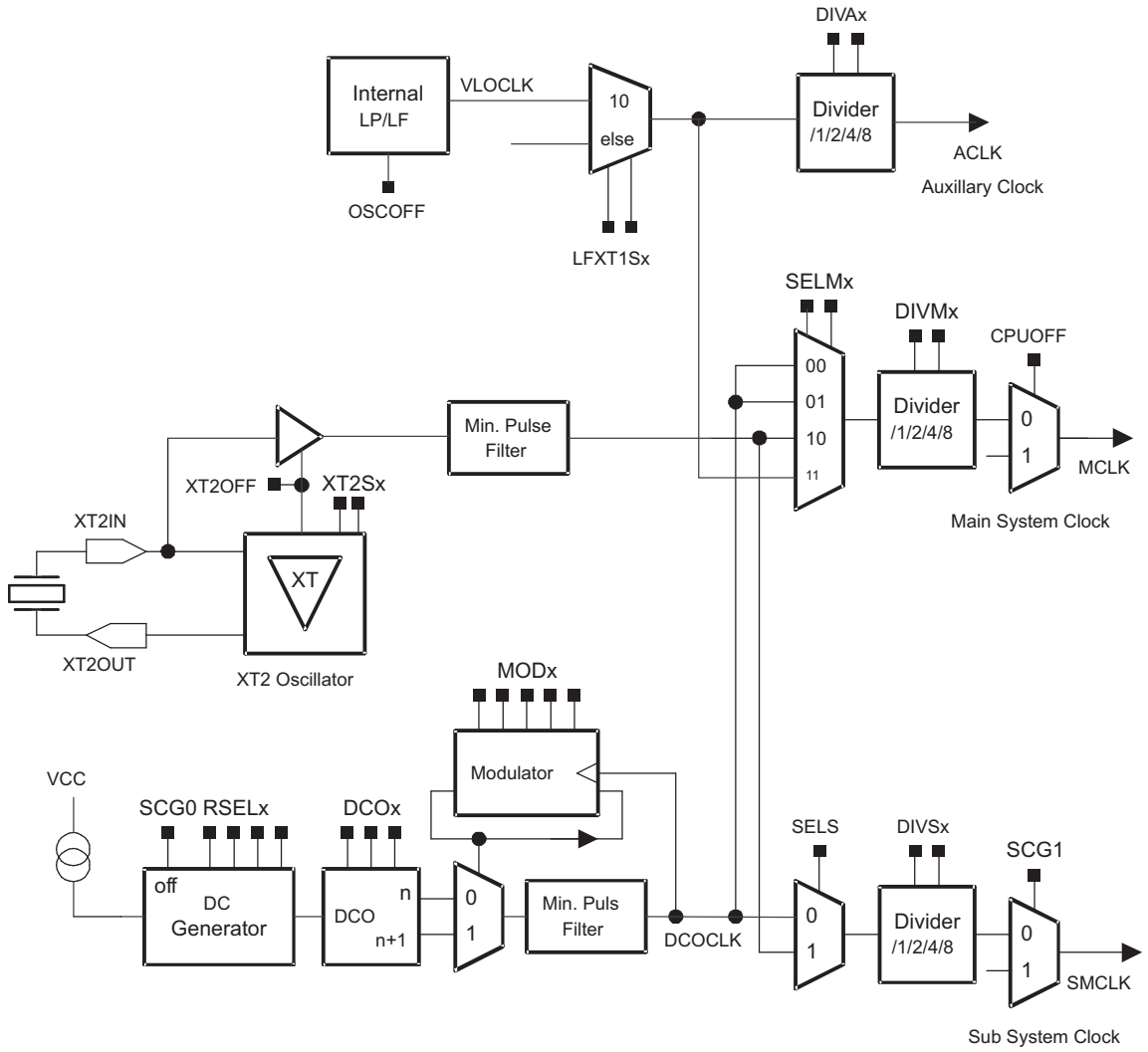


Figure 5-2. Basic Clock Module+ Block Diagram − MSP430AFE2xx

NOTE: LFXT1 is not present in MSP430AFE2xx devices.

5.2 Basic Clock Module+ Operation

After a PUC, MCLK and SMCLK are sourced from DCOCLK at ~1.1 MHz (see the device-specific data sheet for parameters) and ACLK is sourced from LFXT1CLK in LF mode with an internal load capacitance of 6 pF.

Status register control bits SCG0, SCG1, OSCOFF, and CPUOFF configure the MSP430 operating modes and enable or disable portions of the basic clock module+ (see the *System Resets, Interrupts and Operating Modes* chapter). The DCOCTL, BCSCTL1, BCSCTL2, and BCSCTL3 registers configure the basic clock module+.

The basic clock module+ can be configured or reconfigured by software at any time during program execution, for example:

```
CLR.B   &DCOCTL                     ; Select lowest DCOx
                                    ; and MODx settings
BIS.B   #RSEL2+RSEL1+RSEL0,&BCSCTL1 ; Select range 7
BIS.B   #DCO2+DCO1+DCO0,&DCOCTL     ; Select max DCO tap
```

DMA Controller

The DMA controller module transfers data from one address to another without CPU intervention. This chapter describes the operation of the DMA controller of the MSP430x2xx device family.

Topic **Page**

6.1 DMA Introduction

The direct memory access (DMA) controller transfers data from one address to another, without CPU intervention, across the entire address range. For example, the DMA controller can move data from the ADC12 conversion memory to RAM.

Devices that contain a DMA controller may have one, two, or three DMA channels available. Therefore, depending on the number of DMA channels available, some features described in this chapter are not applicable to all devices.

Using the DMA controller can increase the throughput of peripheral modules. It can also reduce system power consumption by allowing the CPU to remain in a low-power mode without having to awaken to move data to or from a peripheral.

The DMA controller features include:

- Up to three independent transfer channels
- Configurable DMA channel priorities
- Requires only two MCLK clock cycles per transfer
- Byte or word and mixed byte/word transfer capability
- Block sizes up to 65535 bytes or words
- Configurable transfer trigger selections
- Selectable edge or level-triggered transfer
- Four addressing modes
- Single, block, or burst-block transfer modes

The DMA controller block diagram is shown in Figure 6-1.

7.1 Flash Memory Introduction

The MSP430 flash memory is bit-, byte-, and word-addressable and programmable. The flash memory module has an integrated controller that controls programming and erase operations. The controller has four registers, a timing generator, and a voltage generator to supply program and erase voltages.

MSP430 flash memory features include:

- Internal programming voltage generation
- Bit, byte, or word programmable
- Ultralow-power operation
- Segment erase and mass erase
- Marginal 0 and marginal 1 read mode (optional, see the device-specific data sheet)

Figure 7-1 shows the block diagram of the flash memory and controller.

NOTE: Minimum V_{cc} during flash write or erase

The minimum V_{cc} voltage during a flash write or erase operation is 2.2 V. If V_{cc} falls below 2.2 V during write or erase, the result of the write or erase is unpredictable.

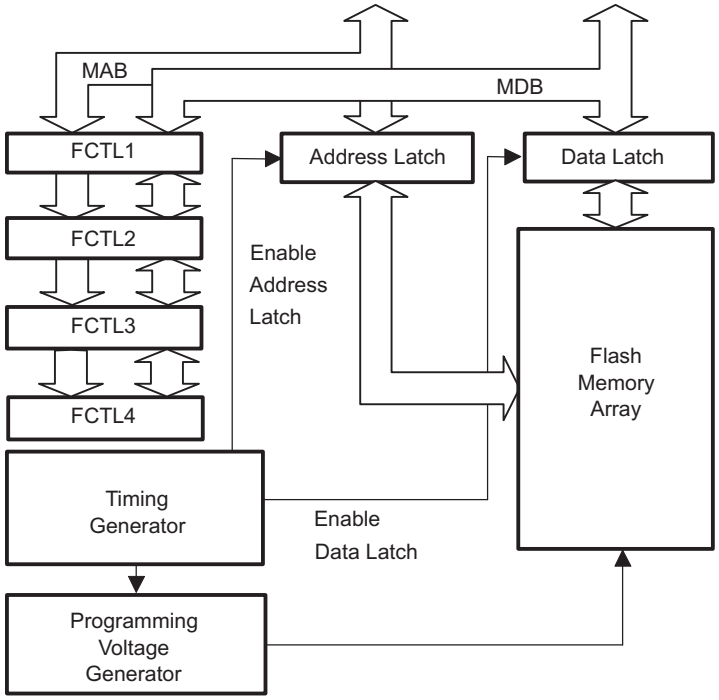


Figure 7-1. Flash Memory Module Block Diagram

7.2 Flash Memory Segmentation

MSP430 flash memory is partitioned into segments. Single bits, bytes, or words can be written to flash memory, but the segment is the smallest size of flash memory that can be erased.

The flash memory is partitioned into main and information memory sections. There is no difference in the operation of the main and information memory sections. Code or data can be located in either section. The differences between the two sections are the segment size and the physical addresses.

The information memory has four 64-byte segments. The main memory has one or more 512-byte segments. See the device-specific data sheet for the complete memory map of a device.

The segments are further divided into blocks.

Flash Memory Controller

This chapter describes the operation of the MSP430x2xx flash memory controller.

Topic **Page**

Digital I/O

This chapter describes the operation of the digital I/O ports.

Topic **Page**

8.1 Digital I/O Introduction

MSP430 devices have up to eight digital I/O ports implemented, P1 to P8. Each port has up to eight I/O pins. Every I/O pin is individually configurable for input or output direction, and each I/O line can be individually read or written to.

Ports P1 and P2 have interrupt capability. Each interrupt for the P1 and P2 I/O lines can be individually enabled and configured to provide an interrupt on a rising edge or falling edge of an input signal. All P1 I/O lines source a single interrupt vector, and all P2 I/O lines source a different, single interrupt vector.

The digital I/O features include:

- Independently programmable individual I/Os
- Any combination of input or output
- Individually configurable P1 and P2 interrupts
- Independent input and output data registers
- Individually configurable pullup or pulldown resistors
- Individually configurable pin-oscillator function (some MSP430 devices)

NOTE: **MSP430G22x0** : These devices feature digital I/O pins P1.2, P1.5, P1.6 and P1.7. The GPIOs P1.0, P1.1, P1.3, P1.4, P2.6, and P2.7 are implemented on this device but not available on the device pin-out. To avoid floating inputs, these GPIOs, these digital I/Os should be properly initialized by running a start-up code. See initialization code below:
mov.b #0x1B, P1REN; ; Terminate unavailable Port1 pins properly ; Config as Input with pull-down enabled
xor.b #0x20, BCSCTL3; ; Select VLO as low freq clock
The initialization code configures GPIOs P1.0, P1.1, P1.3, and P1.4 as inputs with pull-down resistor enabled (that is, P1REN.x = 1) and GPIOs P2.6 and P2.7 are terminated by selecting VLOCLK as ACLK – see the Basic Clock System chapter for details. The register bits of P1.0, P1.1, P1.3, and P1.4 in registers P1OUT, P1DIR, P1IFG, P1IE, P1IES, P1SEL and P1REN should not be altered after the initialization code is executed. Also, all Port2 registers are should not be altered.

8.2 Digital I/O Operation

The digital I/O is configured with user software. The setup and operation of the digital I/O is discussed in the following sections.

8.2.1 Input Register PxIN

Each bit in each PxIN register reflects the value of the input signal at the corresponding I/O pin when the pin is configured as I/O function.

Bit = 0: The input is low

Bit = 1: The input is high

NOTE: **Writing to Read-Only Registers PxIN**

Writing to these read-only registers results in increased current consumption while the write attempt is active.

8.2.2 Output Registers PxOUT

Each bit in each PxOUT register is the value to be output on the corresponding I/O pin when the pin is configured as I/O function, output direction, and the pullup/down resistor is disabled.

Bit = 0: The output is low

Bit = 1: The output is high

If the pin's pullup/pulldown resistor is enabled, the corresponding bit in the PxOUT register selects pullup or pulldown.

Bit = 0: The pin is pulled down

Bit = 1: The pin is pulled up

8.2.3 Direction Registers PxDIR

Each bit in each PxDIR register selects the direction of the corresponding I/O pin, regardless of the selected function for the pin. PxDIR bits for I/O pins that are selected for other functions must be set as required by the other function.

Bit = 0: The port pin is switched to input direction

Bit = 1: The port pin is switched to output direction

8.2.4 Pullup/Pulldown Resistor Enable Registers PxREN

Each bit in each PxREN register enables or disables the pullup/pulldown resistor of the corresponding I/O pin. The corresponding bit in the PxOUT register selects if the pin is pulled up or pulled down.

Bit = 0: Pullup/pulldown resistor disabled

Bit = 1: Pullup/pulldown resistor enabled

8.2.5 Function Select Registers PxSEL and PxSEL2

Port pins are often multiplexed with other peripheral module functions. See the device-specific data sheet to determine pin functions. Each PxSEL and PxSEL2 bit is used to select the pin function - I/O port or peripheral module function.

Table 8-1. PxSEL and PxSEL2

PxSEL2	PxSEL	Pin Function
0	0	I/O function is selected.
0	1	Primary peripheral module function is selected.
1	0	Reserved. See device-specific data sheet.
1	1	Secondary peripheral module function is selected.

Setting PxSELx = 1 does not automatically set the pin direction. Other peripheral module functions may require the PxDIRx bits to be configured according to the direction needed for the module function. See the pin schematics in the device-specific data sheet.

NOTE: Setting PxREN = 1 When PxSEL = 1

On some I/O ports on the MSP430F261x and MSP430F2416/7/8/9, enabling the pullup/pulldown resistor (PxREN = 1) while the module function is selected (PxSEL = 1) does not disable the logic output driver. This combination is not recommended and may result in unwanted current flow through the internal resistor. See the device-specific data sheet pin schematics for more information.

```
;Output ACLK on P2.0 on MSP430F21x1
  BIS.B   #01h,&P2SEL   ; Select ACLK function for pin
  BIS.B   #01h,&P2DIR   ; Set direction to output *Required*
```

NOTE: P1 and P2 Interrupts Are Disabled When PxSEL = 1

When any P1SELx or P2SELx bit is set, the corresponding pin's interrupt function is disabled. Therefore, signals on these pins will not generate P1 or P2 interrupts, regardless of the state of the corresponding P1IE or P2IE bit.

When a port pin is selected as an input to a peripheral, the input signal to the peripheral is a latched representation of the signal at the device pin. While PxSELx = 1, the internal input signal follows the signal at the pin. However, if the PxSELx = 0, the input to the peripheral maintains the value of the input signal at the device pin before the PxSELx bit was reset.

8.2.6 Pin Oscillator

Some MSP430 devices have a pin oscillator function built-in to some pins. The pin oscillator function may be used in capacitive touch sensing applications to eliminate external passive components. Additionally, the pin oscillator may be used in sensor applications.

- No external components to create the oscillation
- Capacitive sensors can be connected directly to MSP430 pin
- Robust, typical built-in hysteresis of ~0.7 V

When the pin oscillator function is enabled, other pin configurations are overwritten. The output driver is turned off while the weak pullup/pulldown is enabled and controlled by the voltage level on the pin itself. The voltage on the I/O is fed into the Schmitt trigger of the pin and then routed to a timer. The connection to the timer is device specific and, thus, defined in the device-specific data sheet. The Schmitt-trigger output is inverted and then decides if the pullup or the pulldown is enabled. Due to the inversion, the pin starts to oscillate as soon as the pin oscillator pin configuration is selected. Some of the pin-oscillator outputs are combined by a logical OR before routing to a timer clock input or timer capture channel. Therefore, only one pin oscillator should be enabled at a time. The oscillation frequency of each pin is defined by the load on the pin and by the I/O type. I/Os with analog functions typically show a lower oscillation frequency than pure digital I/Os. See the device-specific data sheet for details. Pins without external load show typical oscillation frequencies of 1 MHz to 3 MHz.

Pin oscillator in a cap touch application

A typical touch pad application using the pin oscilator is shown in Figure 8-1.

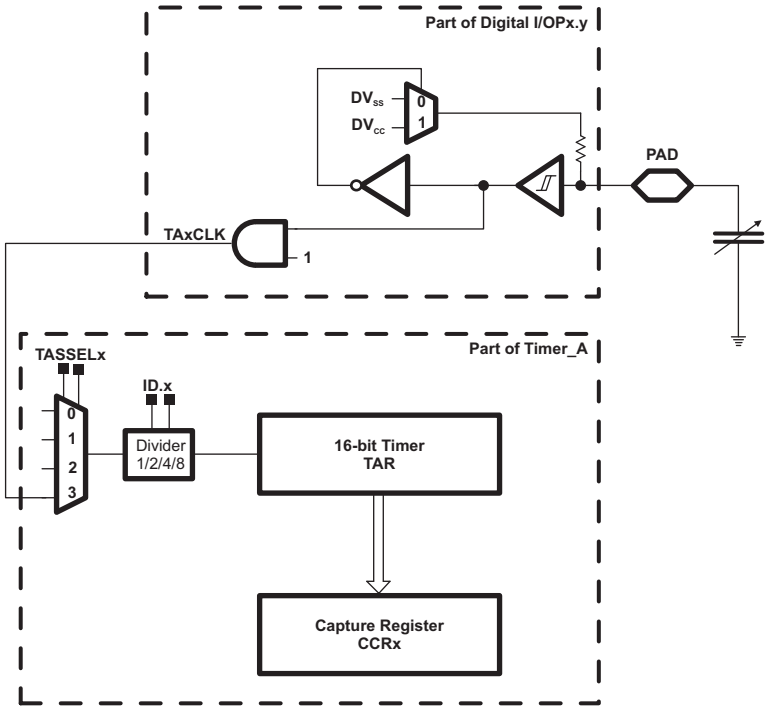


Figure 8-1. Example Circuitry and Configuration using the Pin Oscillator

A change of the capacitance of the touch pad (external capacitive load) has an effect on the pin oscillator frequency. An approaching finger tip increases the capacitance of the touch pad thus leads to a lower self-oscillation frequency due to the longer charging time. The oscillation frequency can directly be captured in a built-in Timer channel. The typical sensitivity of a pin is shown in Figure 8-2.

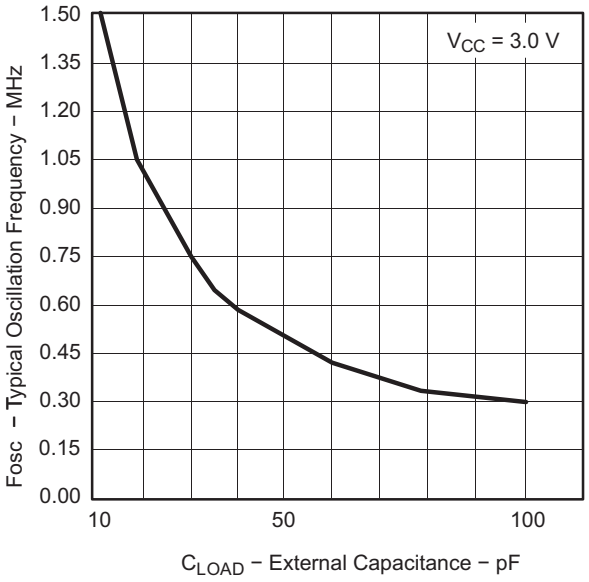


Figure 8-2. Typical Pin-Oscillation Frequency

8.2.7 P1 and P2 Interrupts

Each pin in ports P1 and P2 have interrupt capability, configured with the PxIFG, PxIE, and PxIES registers. All P1 pins source a single interrupt vector, and all P2 pins source a different single interrupt vector. The PxIFG register can be tested to determine the source of a P1 or P2 interrupt.

8.2.7.1 Interrupt Flag Registers P1IFG, P2IFG

Each PxIFGx bit is the interrupt flag for its corresponding I/O pin and is set when the selected input signal edge occurs at the pin. All PxIFGx interrupt flags request an interrupt when their corresponding PxIE bit and the GIE bit are set. Each PxIFG flag must be reset with software. Software can also set each PxIFG flag, providing a way to generate a software initiated interrupt.

Bit = 0: No interrupt is pending

Bit = 1: An interrupt is pending

Only transitions, not static levels, cause interrupts. If any PxIFGx flag becomes set during a Px interrupt service routine, or is set after the RETI instruction of a Px interrupt service routine is executed, the set PxIFGx flag generates another interrupt. This ensures that each transition is acknowledged.

NOTE: PxIFG Flags When Changing PxOUT or PxDIR

Writing to P1OUT, P1DIR, P2OUT, or P2DIR can result in setting the corresponding P1IFG or P2IFG flags.

8.2.7.2 Interrupt Edge Select Registers P1IES, P2IES

Each PxIES bit selects the interrupt edge for the corresponding I/O pin.

Bit = 0: The PxIFGx flag is set with a low-to-high transition

Bit = 1: The PxIFGx flag is set with a high-to-low transition

NOTE: Writing to PxIESx

Writing to P1IES, or P2IES can result in setting the corresponding interrupt flags.

PxIESx	PxINx	PxIFGx
0 → 1	0	May be set
0 → 1	1	Unchanged
1 → 0	0	Unchanged
1 → 0	1	May be set

8.2.7.3 Interrupt Enable P1IE, P2IE

Each PxIE bit enables the associated PxIFG interrupt flag.

Bit = 0: The interrupt is disabled.

Bit = 1: The interrupt is enabled.

8.2.8 Configuring Unused Port Pins

Unused I/O pins should be configured as I/O function, output direction, and left unconnected on the PC board, to prevent a floating input and reduce power consumption. The value of the PxOUT bit is irrelevant, since the pin is unconnected. Alternatively, the integrated pullup/pulldown resistor can be enabled by setting the PxREN bit of the unused pin to prevent the floating input. See the *System Resets, Interrupts, and Operating Modes* chapter for termination of unused pins.

8.3 Digital I/O Registers

The digital I/O registers are listed in Table 8-2.

Table 8-2. Digital I/O Registers

Port	Register	Short Form	Address	Register Type	Initial State
P1	Input	P1IN	020h	Read only	-
	Output	P1OUT	021h	Read/write	Unchanged
	Direction	P1DIR	022h	Read/write	Reset with PUC
	Interrupt Flag	P1IFG	023h	Read/write	Reset with PUC
	Interrupt Edge Select	P1IES	024h	Read/write	Unchanged
	Interrupt Enable	P1IE	025h	Read/write	Reset with PUC
	Port Select	P1SEL	026h	Read/write	Reset with PUC
	Port Select 2	P1SEL2	041h	Read/write	Reset with PUC
	Resistor Enable	P1REN	027h	Read/write	Reset with PUC
P2	Input	P2IN	028h	Read only	-
	Output	P2OUT	029h	Read/write	Unchanged
	Direction	P2DIR	02Ah	Read/write	Reset with PUC
	Interrupt Flag	P2IFG	02Bh	Read/write	Reset with PUC
	Interrupt Edge Select	P2IES	02Ch	Read/write	Unchanged
	Interrupt Enable	P2IE	02Dh	Read/write	Reset with PUC
	Port Select	P2SEL	02Eh	Read/write	0C0h with PUC
	Port Select 2	P2SEL2	042h	Read/write	Reset with PUC
	Resistor Enable	P2REN	02Fh	Read/write	Reset with PUC
P3	Input	P3IN	018h	Read only	-
	Output	P3OUT	019h	Read/write	Unchanged
	Direction	P3DIR	01Ah	Read/write	Reset with PUC
	Port Select	P3SEL	01Bh	Read/write	Reset with PUC
	Port Select 2	P3SEL2	043h	Read/write	Reset with PUC
	Resistor Enable	P3REN	010h	Read/write	Reset with PUC
P4	Input	P4IN	01Ch	Read only	-
	Output	P4OUT	01Dh	Read/write	Unchanged
	Direction	P4DIR	01Eh	Read/write	Reset with PUC
	Port Select	P4SEL	01Fh	Read/write	Reset with PUC
	Port Select 2	P4SEL2	044h	Read/write	Reset with PUC
	Resistor Enable	P4REN	011h	Read/write	Reset with PUC
P5	Input	P5IN	030h	Read only	-
	Output	P5OUT	031h	Read/write	Unchanged
	Direction	P5DIR	032h	Read/write	Reset with PUC
	Port Select	P5SEL	033h	Read/write	Reset with PUC
	Port Select 2	P5SEL2	045h	Read/write	Reset with PUC
	Resistor Enable	P5REN	012h	Read/write	Reset with PUC
P6	Input	P6IN	034h	Read only	-
	Output	P6OUT	035h	Read/write	Unchanged
	Direction	P6DIR	036h	Read/write	Reset with PUC
	Port Select	P6SEL	037h	Read/write	Reset with PUC
	Port Select 2	P6SEL2	046h	Read/write	Reset with PUC
	Resistor Enable	P6REN	013h	Read/write	Reset with PUC

Table 8-2. Digital I/O Registers (continued)

Port	Register	Short Form	Address	Register Type	Initial State
P7	Input	P7IN	038h	Read only	-
	Output	P7OUT	03Ah	Read/write	Unchanged
	Direction	P7DIR	03Ch	Read/write	Reset with PUC
	Port Select	P7SEL	03Eh	Read/write	Reset with PUC
	Port Select 2	P7SEL2	047h	Read/write	Reset with PUC
	Resistor Enable	P7REN	014h	Read/write	Reset with PUC
P8	Input	P8IN	039h	Read only	-
	Output	P8OUT	03Bh	Read/write	Unchanged
	Direction	P8DIR	03Dh	Read/write	Reset with PUC
	Port Select	P8SEL	03Fh	Read/write	Reset with PUC
	Port Select 2	P8SEL2	048h	Read/write	Reset with PUC
	Resistor Enable	P8REN	015h	Read/write	Reset with PUC

Supply Voltage Supervisor (SVS)

This chapter describes the operation of the SVS. The SVS is implemented in selected MSP430x2xx devices.

Topic **Page**

9.1 Supply Voltage Supervisor (SVS) Introduction

The SVS is used to monitor the AV_{CC} supply voltage or an external voltage. The SVS can be configured to set a flag or generate a POR reset when the supply voltage or external voltage drops below a user-selected threshold.

The SVS features include:

- AV_{CC} monitoring
- Selectable generation of POR
- Output of SVS comparator accessible by software
- Low-voltage condition latched and accessible by software
- 14 selectable threshold levels
- External channel to monitor external voltage

The SVS block diagram is shown in Figure 9-1.

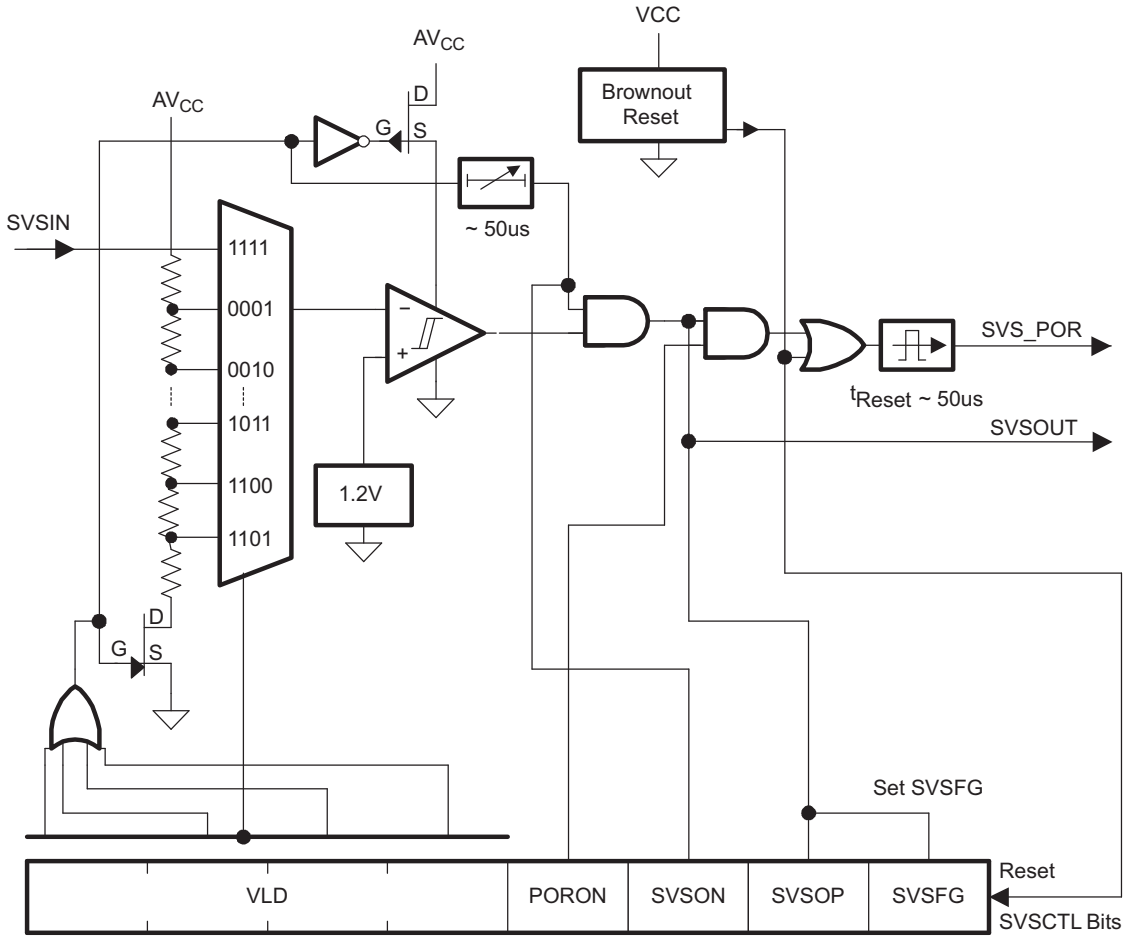


Figure 9-1. SVS Block Diagram

Watchdog Timer+ (WDT+)

The watchdog timer+ (WDT+) is a 16-bit timer that can be used as a watchdog or as an interval timer. This chapter describes the WDT+ The WDT+ is implemented in all MSP430x2xx devices.

Topic **Page**

10.1 Watchdog Timer+ (WDT+) Introduction

The primary function of the WDT+ module is to perform a controlled system restart after a software problem occurs. If the selected time interval expires, a system reset is generated. If the watchdog function is not needed in an application, the module can be configured as an interval timer and can generate interrupts at selected time intervals.

Features of the watchdog timer+ module include:

- Four software-selectable time intervals
- Watchdog mode
- Interval mode
- Access to WDT+ control register is password protected
- Control of $\overline{\text{RST}}$/NMI pin function
- Selectable clock source
- Can be stopped to conserve power
- Clock fail-safe feature

The WDT+ block diagram is shown in Figure 10-1.

NOTE: Watchdog Timer+ Powers Up Active

After a PUC, the WDT+ module is automatically configured in the watchdog mode with an initial 32768 clock cycle reset interval using the DCOCLK. The user must setup or halt the WDT+ prior to the expiration of the initial reset interval.

Hardware Multiplier

This chapter describes the hardware multiplier. The hardware multiplier is implemented in some MSP430x2xx devices.

Topic **Page**

11.1 Hardware Multiplier Introduction

The hardware multiplier is a peripheral and is not part of the MSP430 CPU. This means, its activities do not interfere with the CPU activities. The multiplier registers are peripheral registers that are loaded and read with CPU instructions.

The hardware multiplier supports:

- Unsigned multiply
- Signed multiply
- Unsigned multiply accumulate
- Signed multiply accumulate
- 16x16 bits, 16x8 bits, 8x16 bits, 8x8 bits

The hardware multiplier block diagram is shown in Figure 11-1.

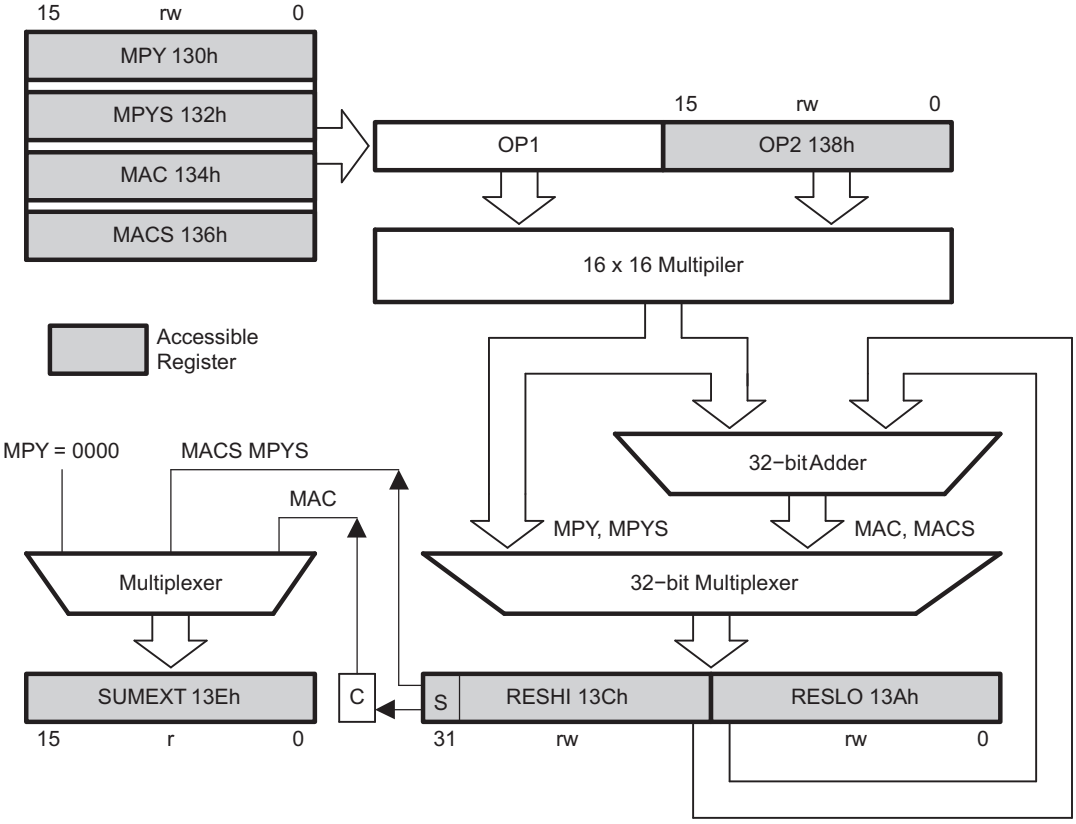


Figure 11-1. Hardware Multiplier Block Diagram

11.2 Hardware Multiplier Operation

The hardware multiplier supports unsigned multiply, signed multiply, unsigned multiply accumulate, and signed multiply accumulate operations. The type of operation is selected by the address the first operand is written to.

The hardware multiplier has two 16-bit operand registers, OP1 and OP2, and three result registers, RESLO, RESHI, and SUMEXT. RESLO stores the low word of the result, RESHI stores the high word of the result, and SUMEXT stores information about the result. The result is ready in three MCLK cycles and can be read with the next instruction after writing to OP2, except when using an indirect addressing mode to access the result. When using indirect addressing for the result, a NOP is required before the result is ready.

Timer_A

Timer_A is a 16-bit timer/counter with multiple capture/compare registers. This chapter describes the operation of the Timer_A of the MSP430x2xx device family.

Topic **Page**

12.1 Timer_A Introduction

Timer_A is a 16-bit timer/counter with three capture/compare registers. Timer_A can support multiple capture/compares, PWM outputs, and interval timing. Timer_A also has extensive interrupt capabilities. Interrupts may be generated from the counter on overflow conditions and from each of the capture/compare registers.

Timer_A features include:

- Asynchronous 16-bit timer/counter with four operating modes
- Selectable and configurable clock source
- Two or three configurable capture/compare registers
- Configurable outputs with PWM capability
- Asynchronous input and output latching
- Interrupt vector register for fast decoding of all Timer_A interrupts

The block diagram of Timer_A is shown in Figure 12-1.

NOTE: Use of the Word *Count*

Count is used throughout this chapter. It means the counter must be in the process of counting for the action to take place. If a particular value is directly written to the counter, then an associated action will not take place.

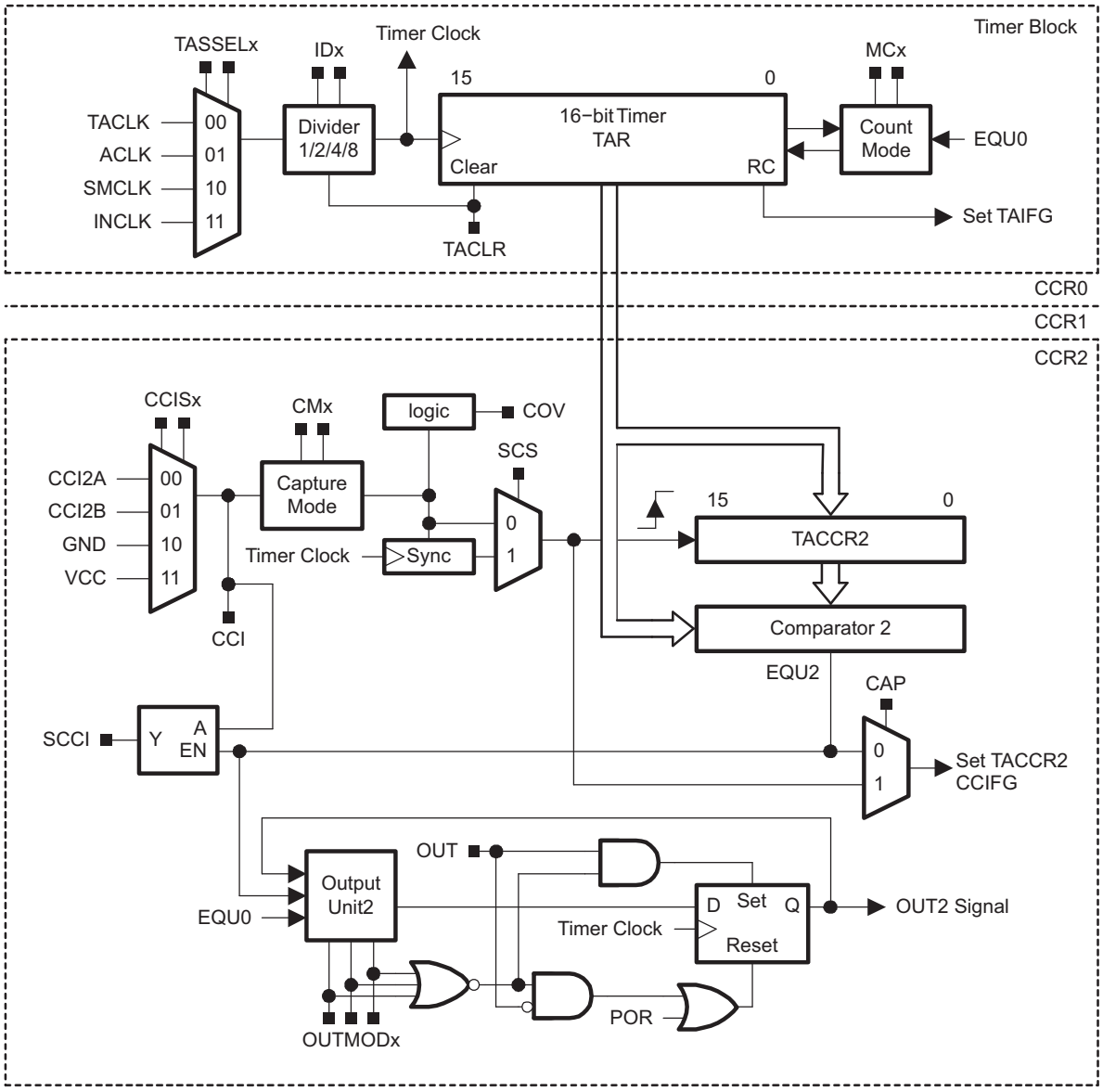


Figure 12-1. Timer_A Block Diagram

12.2 Timer_A Operation

The Timer_A module is configured with user software. The setup and operation of Timer_A is discussed in the following sections.

12.2.1 16-Bit Timer Counter

The 16-bit timer/counter register, TAR, increments or decrements (depending on mode of operation) with each rising edge of the clock signal. TAR can be read or written with software. Additionally, the timer can generate an interrupt when it overflows.

TAR may be cleared by setting the TACLR bit. Setting TACLR also clears the clock divider and count direction for up/down mode.

Timer_B

Timer_B is a 16-bit timer/counter with multiple capture/compare registers. This chapter describes the operation of the Timer_B of the MSP430x2xx device family.

Topic **Page**

13.1 Timer_B Introduction

Timer_B is a 16-bit timer/counter with three or seven capture/compare registers. Timer_B can support multiple capture/compares, PWM outputs, and interval timing. Timer_B also has extensive interrupt capabilities. Interrupts may be generated from the counter on overflow conditions and from each of the capture/compare registers.

Timer_B features include :

- Asynchronous 16-bit timer/counter with four operating modes and four selectable lengths
- Selectable and configurable clock source
- Three or seven configurable capture/compare registers
- Configurable outputs with PWM capability
- Double-buffered compare latches with synchronized loading
- Interrupt vector register for fast decoding of all Timer_B interrupts

The block diagram of Timer_B is shown in Figure 13-1.

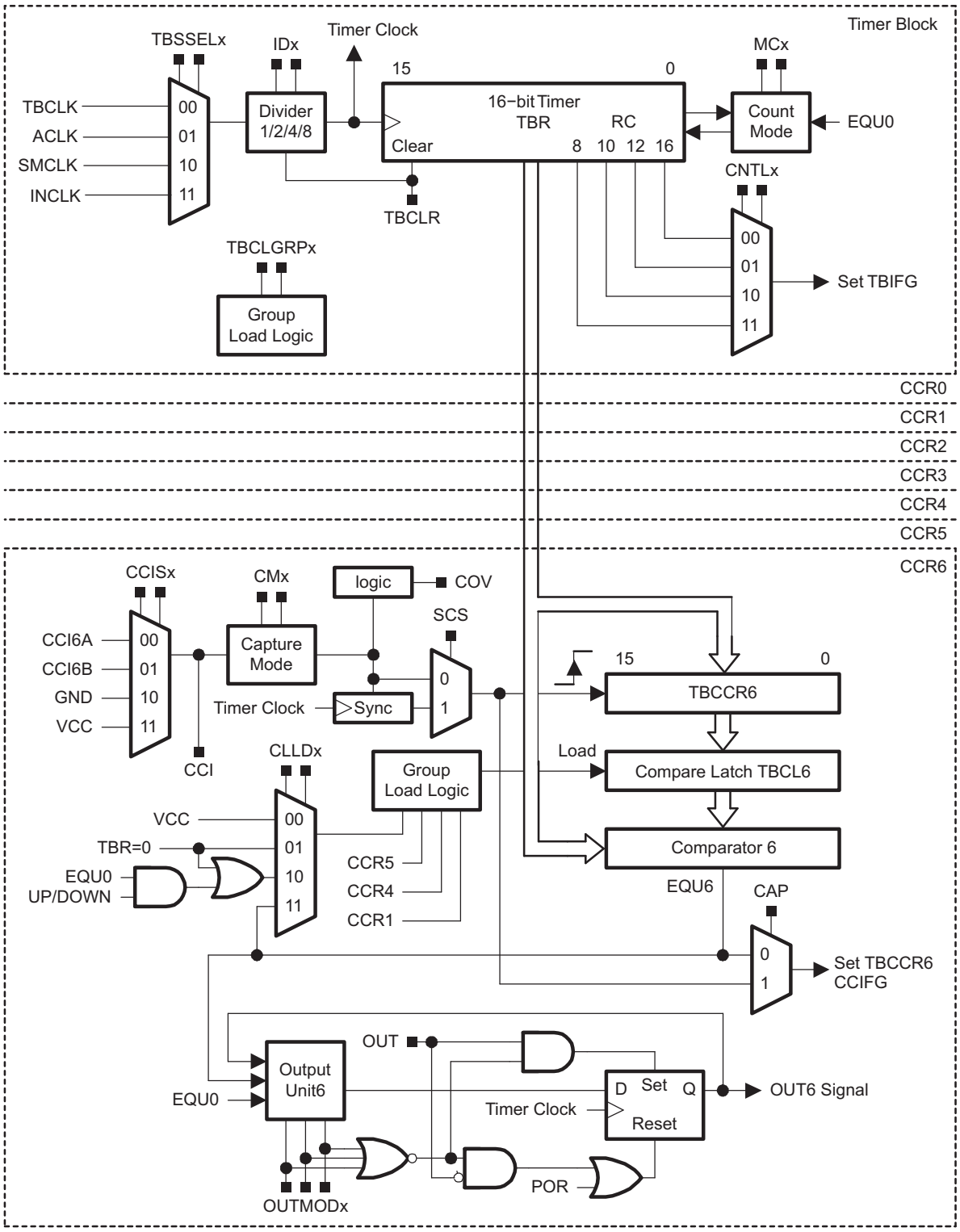
NOTE: Use of the Word *Count*

Count is used throughout this chapter. It means the counter must be in the process of counting for the action to take place. If a particular value is directly written to the counter, then an associated action does not take place.

13.1.1 Similarities and Differences From Timer_A

Timer_B is identical to Timer_A with the following exceptions:

- The length of Timer_B is programmable to be 8, 10, 12, or 16 bits.
- Timer_B TBCCRx registers are double-buffered and can be grouped.
- All Timer_B outputs can be put into a high-impedance state.
- The SCCI bit function is not implemented in Timer_B.



NOTE: INCLK is device-specific, often assigned to the inverted TBCLK, refer to device-specific data sheet.

Figure 13-1. Timer_B Block Diagram

Universal Serial Interface (USI)

The Universal Serial Interface (USI) module provides SPI and I^2C serial communication with one hardware module. This chapter discusses both modes.

Topic **Page**

14.1 USI Introduction

The USI module provides the basic functionality to support synchronous serial communication. In its simplest form, it is an 8- or 16-bit shift register that can be used to output data streams, or when combined with minimal software, can implement serial communication. In addition, the USI includes built-in hardware functionality to ease the implementation of SPI and I^2C communication. The USI module also includes interrupts to further reduce the necessary software overhead for serial communication and to maintain the ultra-low-power capabilities of the MSP430.

The USI module features include:

- Three-wire SPI mode support
- I^2C mode support
- Variable data length
- Slave operation in LPM4; no internal clock required
- Selectable MSB or LSB data order
- START and STOP detection for I^2C mode with automatic SCL control
- Arbitration lost detection in master mode
- Programmable clock generation
- Selectable clock polarity and phase control

Figure 14-1 shows the USI module in SPI mode. Figure 14-2 shows the USI module in I^2C mode.

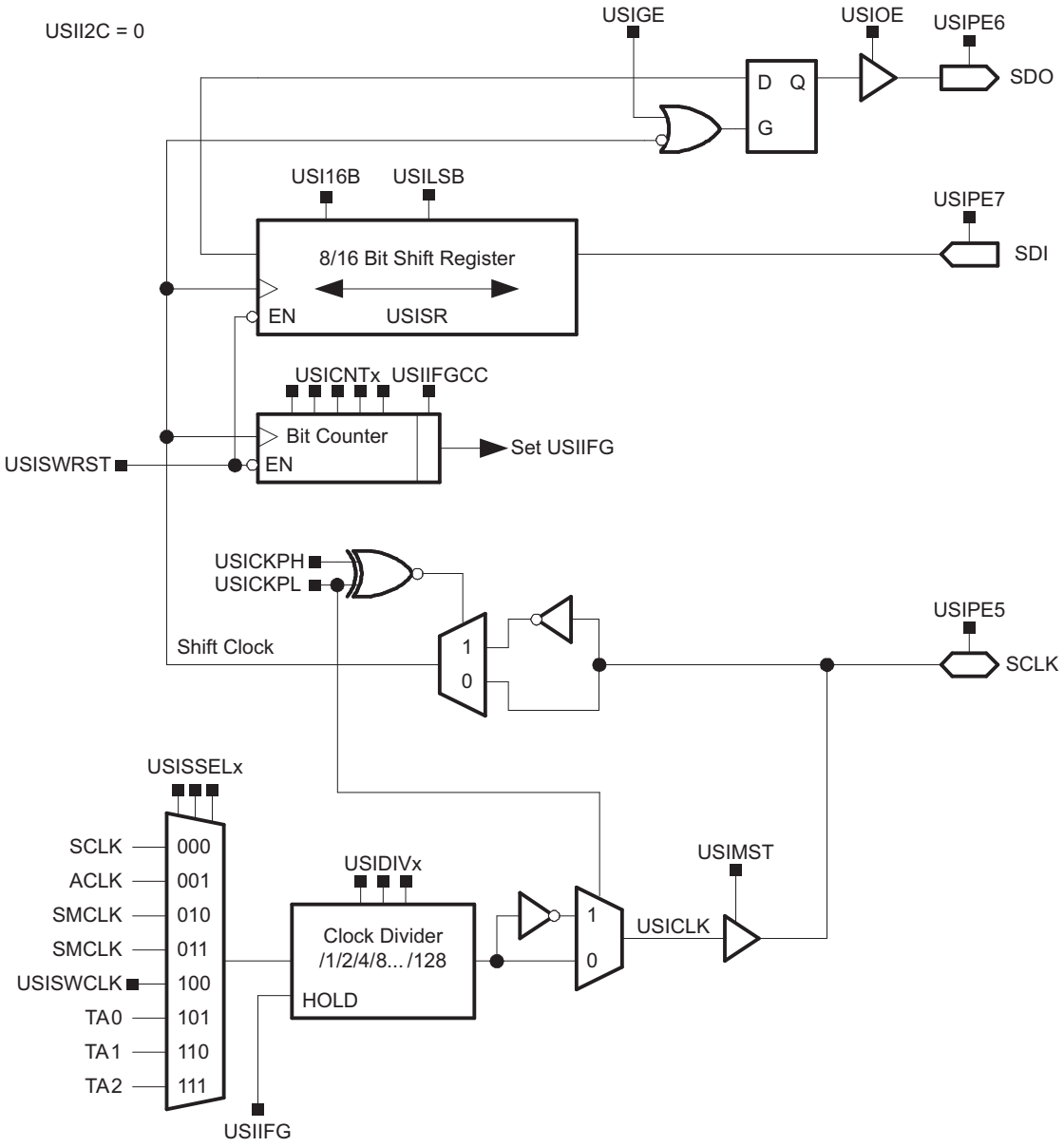


Figure 14-1. USI Block Diagram: SPI Mode

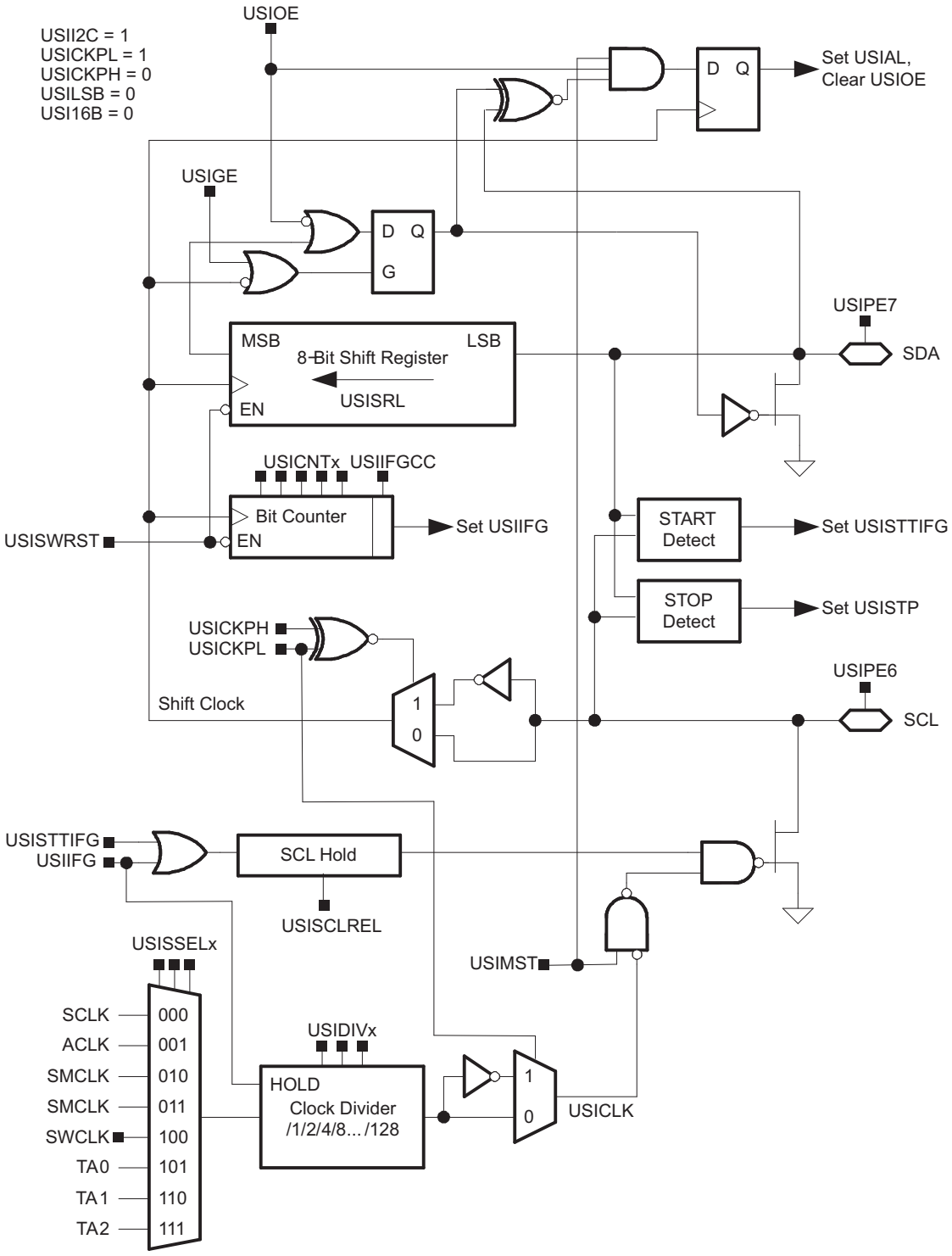


Figure 14-2. USI Block Diagram: I²C Mode

Universal Serial Communication Interface, UART Mode

The universal serial communication interface (USCI) supports multiple serial communication modes with one hardware module. This chapter discusses the operation of the asynchronous UART mode.

Topic **Page**

15.1 USCI Overview

The universal serial communication interface (USCI) modules support multiple serial communication modes. Different USCI modules support different modes. Each different USCI module is named with a different letter. For example, USCI_A is different from USCI_B, etc. If more than one identical USCI module is implemented on one device, those modules are named with incrementing numbers. For example, if one device has two USCI_A modules, they are named USCI_A0 and USCI_A1. See the device-specific data sheet to determine which USCI modules, if any, are implemented on which devices.

The USCI_Ax modules support:

- UART mode
- Pulse shaping for IrDA communications
- Automatic baud rate detection for LIN communications
- SPI mode

The USCI_Bx modules support:

- I^2C mode
- SPI mode

15.2 USCI Introduction: UART Mode

In asynchronous mode, the USCI_Ax modules connect the MSP430 to an external system via two external pins, UCAxRXD and UCAxTXD. UART mode is selected when the UCSYNC bit is cleared.

UART mode features include:

- 7- or 8-bit data with odd, even, or non-parity
- Independent transmit and receive shift registers
- Separate transmit and receive buffer registers
- LSB-first or MSB-first data transmit and receive
- Built-in idle-line and address-bit communication protocols for multiprocessor systems
- Receiver start-edge detection for auto-wake up from LPMx modes
- Programmable baud rate with modulation for fractional baud rate support
- Status flags for error detection and suppression
- Status flags for address detection
- Independent interrupt capability for receive and transmit

Figure 15-1 shows the USCI_Ax when configured for UART mode.

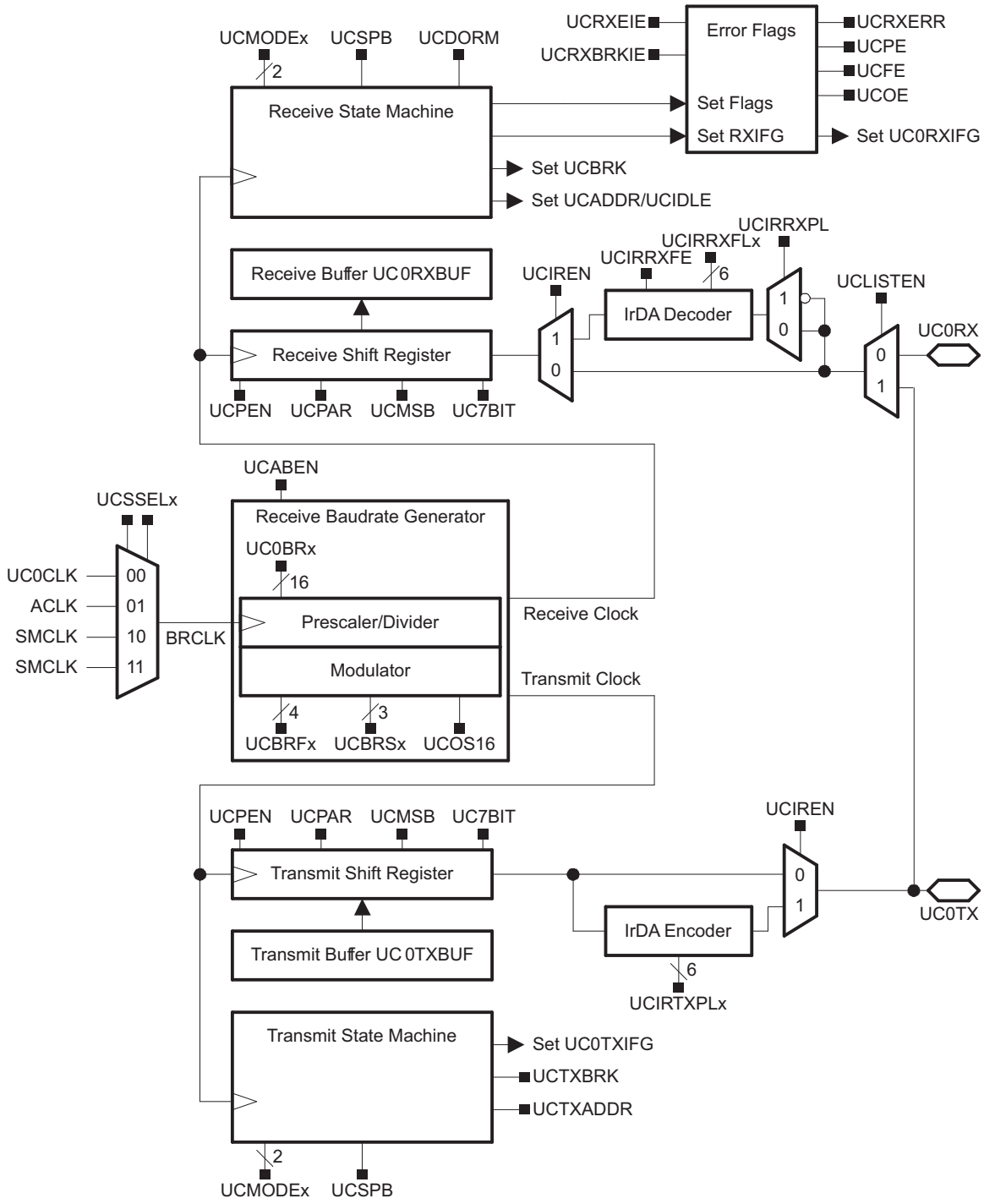


Figure 15-1. USCI_Ax Block Diagram: UART Mode (UCSYNC = 0)

Universal Serial Communication Interface, SPI Mode

The universal serial communication interface (USCI) supports multiple serial communication modes with one hardware module. This chapter discusses the operation of the synchronous peripheral interface or SPI mode.

Topic **Page**

16.1 USCI Overview

The universal serial communication interface (USCI) modules support multiple serial communication modes. Different USCI modules support different modes. Each different USCI module is named with a different letter (for example, USCI_A is different from USCI_B). If more than one identical USCI module is implemented on one device, those modules are named with incrementing numbers. For example, if one device has two USCI_A modules, they are named USCI_A0 and USCI_A1. See the device-specific data sheet to determine which USCI modules, if any, are implemented on each device.

The USCI_Ax modules support:

- UART mode
- Pulse shaping for IrDA communications
- Automatic baud rate detection for LIN communications
- SPI mode

The USCI_Bx modules support:

- I^2C mode
- SPI mode

16.2 USCI Introduction: SPI Mode

In synchronous mode, the USCI connects the MSP430 to an external system via three or four pins: UCxSIMO, UCxSOMI, UCxCLK, and UCxSTE. SPI mode is selected when the UCSYNC bit is set and SPI mode (3-pin or 4-pin) is selected with the UCMODEx bits.

SPI mode features include:

- 7- or 8-bit data length
- LSB-first or MSB-first data transmit and receive
- 3-pin and 4-pin SPI operation
- Master or slave modes
- Independent transmit and receive shift registers
- Separate transmit and receive buffer registers
- Continuous transmit and receive operation
- Selectable clock polarity and phase control
- Programmable clock frequency in master mode
- Independent interrupt capability for receive and transmit
- Slave operation in LPM4

Figure 16-1 shows the USCI when configured for SPI mode.

Universal Serial Communication Interface, I²C Mode

The universal serial communication interface (USCI) supports multiple serial communication modes with one hardware module. This chapter discusses the operation of the I²C mode.

Topic **Page**

17.1 USCI Overview

The universal serial communication interface (USCI) modules support multiple serial communication modes. Different USCI modules support different modes. Each different USCI module is named with a different letter. For example, USCI_A is different from USCI_B, etc. If more than one identical USCI module is implemented on one device, those modules are named with incrementing numbers. For example, if one device has two USCI_A modules, they are named USCI_A0 and USCI_A1. See the device-specific data sheet to determine which USCI modules, if any, are implemented on which devices.

The USCI_Ax modules support:

- UART mode
- Pulse shaping for IrDA communications
- Automatic baud rate detection for LIN communications
- SPI mode

The USCI_Bx modules support:

- I^2C mode
- SPI mode

17.2 USCI Introduction: I^2C Mode

In I^2C mode, the USCI module provides an interface between the MSP430 and I^2C-compatible devices connected by way of the two-wire I^2C serial bus. External components attached to the I^2C bus serially transmit and/or receive serial data to/from the USCI module through the 2-wire I^2C interface.

The I^2C mode features include:

- Compliance to the Philips Semiconductor I^2C specification v2.1
  - 7-bit and 10-bit device addressing modes
  - General call
  - START/RESTART/STOP
  - Multi-master transmitter/receiver mode
  - Slave receiver/transmitter mode
  - Standard mode up to 100 kbps and fast mode up to 400 kbps support
- Programmable UCxCLK frequency in master mode
- Designed for low power
- Slave receiver START detection for auto-wake up from LPMx modes
- Slave operation in LPM4

Figure 17-1 shows the USCI when configured in I^2C mode.

USART Peripheral Interface, UART Mode

The universal synchronous/asynchronous receive/transmit (USART) peripheral interface supports two serial modes with one hardware module. This chapter discusses the operation of the asynchronous UART mode. USART0 is implemented on the MSP430AFE2xx devices.

Topic **Page**

18.1 USART Introduction: UART Mode

In asynchronous mode, the USART connects the MSP430 to an external system via two external pins, URXD and UTXD. UART mode is selected when the SYNC bit is cleared.

UART mode features include:

- 7- or 8-bit data with odd parity, even parity, or non-parity
- Independent transmit and receive shift registers
- Separate transmit and receive buffer registers
- LSB-first data transmit and receive
- Built-in idle-line and address-bit communication protocols for multiprocessor systems
- Receiver start-edge detection for auto-wake up from LPMx modes
- Programmable baud rate with modulation for fractional baud rate support
- Status flags for error detection and suppression and address detection
- Independent interrupt capability for receive and transmit

Figure 18-1 shows the USART when configured for UART mode.

USART Peripheral Interface, SPI Mode

The universal synchronous/asynchronous receive/transmit (USART) peripheral interface supports two serial modes with one hardware module. This chapter discusses the operation of the synchronous peripheral interface or SPI mode. USART0 is implemented on the MSP430AFE2xx devices.

Topic **Page**

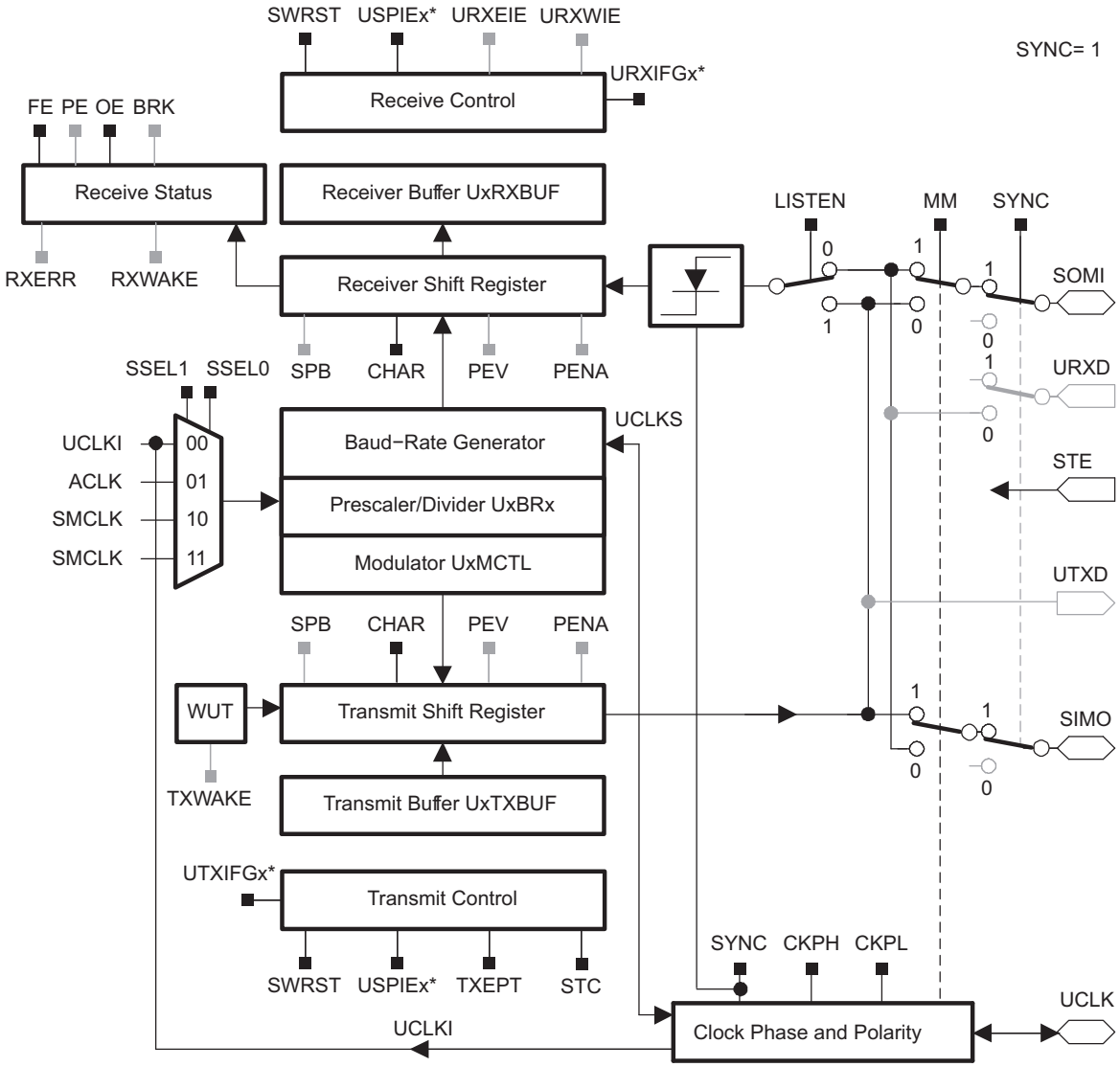
19.1 USART Introduction: SPI Mode

In synchronous mode, the USART connects the MSP430 to an external system via three or four pins: SIMO, SOMI, UCLK, and STE. SPI mode is selected when the SYNC bit is set and the I2C bit is cleared.

SPI mode features include:

- 7-bit or 8-bit data length
- 3-pin and 4-pin SPI operation
- Master or slave modes
- Independent transmit and receive shift registers
- Separate transmit and receive buffer registers
- Selectable UCLK polarity and phase control
- Programmable UCLK frequency in master mode
- Independent interrupt capability for receive and transmit

Figure 19-1 shows the USART when configured for SPI mode.



* See the device-specific data sheet for SFR locations.

Figure 19-1. USART Block Diagram: SPI Mode

OA

The OA is a general purpose operational amplifier. This chapter describes the OA. Two OA modules are implemented in the MSP430x22x4 devices.

Topic **Page**

20.1 OA Introduction

The OA operational amplifiers support front-end analog signal conditioning prior to analog-to-digital conversion.

Features of the OA include:

- Single supply, low-current operation
- Rail-to-rail output
- Programmable settling time vs. power consumption
- Software selectable configurations
- Software selectable feedback resistor ladder for PGA implementations

NOTE: Multiple OA Modules

Some devices may integrate more than one OA module. If more than one OA is present on a device, the multiple OA modules operate identically.

Throughout this chapter, nomenclature appears such as OAxCTL0 to describe register names. When this occurs, the x is used to indicate which OA module is being discussed. In cases where operation is identical, the register is simply referred to as OAxCTL0.

The block diagram of the OA module is shown in Figure 20-1.

Comparator_A+

Comparator_A+ is an analog voltage comparator. This chapter describes the operation of the Comparator_A+ of the 2xx family.

Topic **Page**

21.1 Comparator_A+ Introduction

The Comparator_A+ module supports precision slope analog-to-digital conversions, supply voltage supervision, and monitoring of external analog signals.

Features of Comparator_A+ include:

- Inverting and non-inverting terminal input multiplexer
- Software selectable RC-filter for the comparator output
- Output provided to Timer_A capture input
- Software control of the port input buffer
- Interrupt capability
- Selectable reference voltage generator
- Comparator and reference generator can be powered down
- Input Multiplexer

The Comparator_A+ block diagram is shown in Figure 21-1.

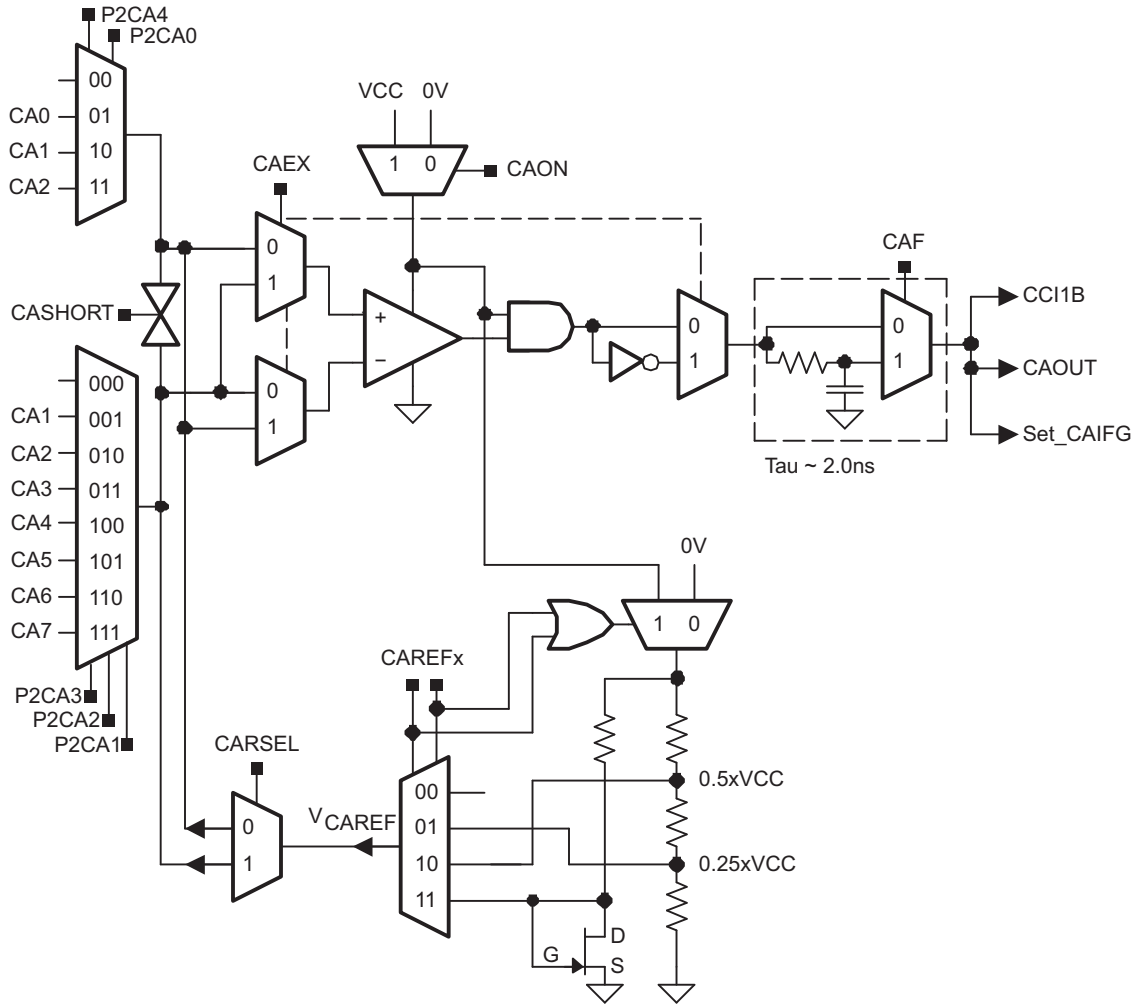


Figure 21-1. Comparator_A+ Block Diagram

NOTE: **MSP430G2210:** Channels 2, 5, 6, and 7 are available. Other channels should not be enabled.

ADC10

The ADC10 module is a high-performance 10-bit analog-to-digital converter. This chapter describes the operation of the ADC10 module of the 2xx family in general. There are device with less than eight external input channels.

Topic **Page**

22.1 ADC10 Introduction

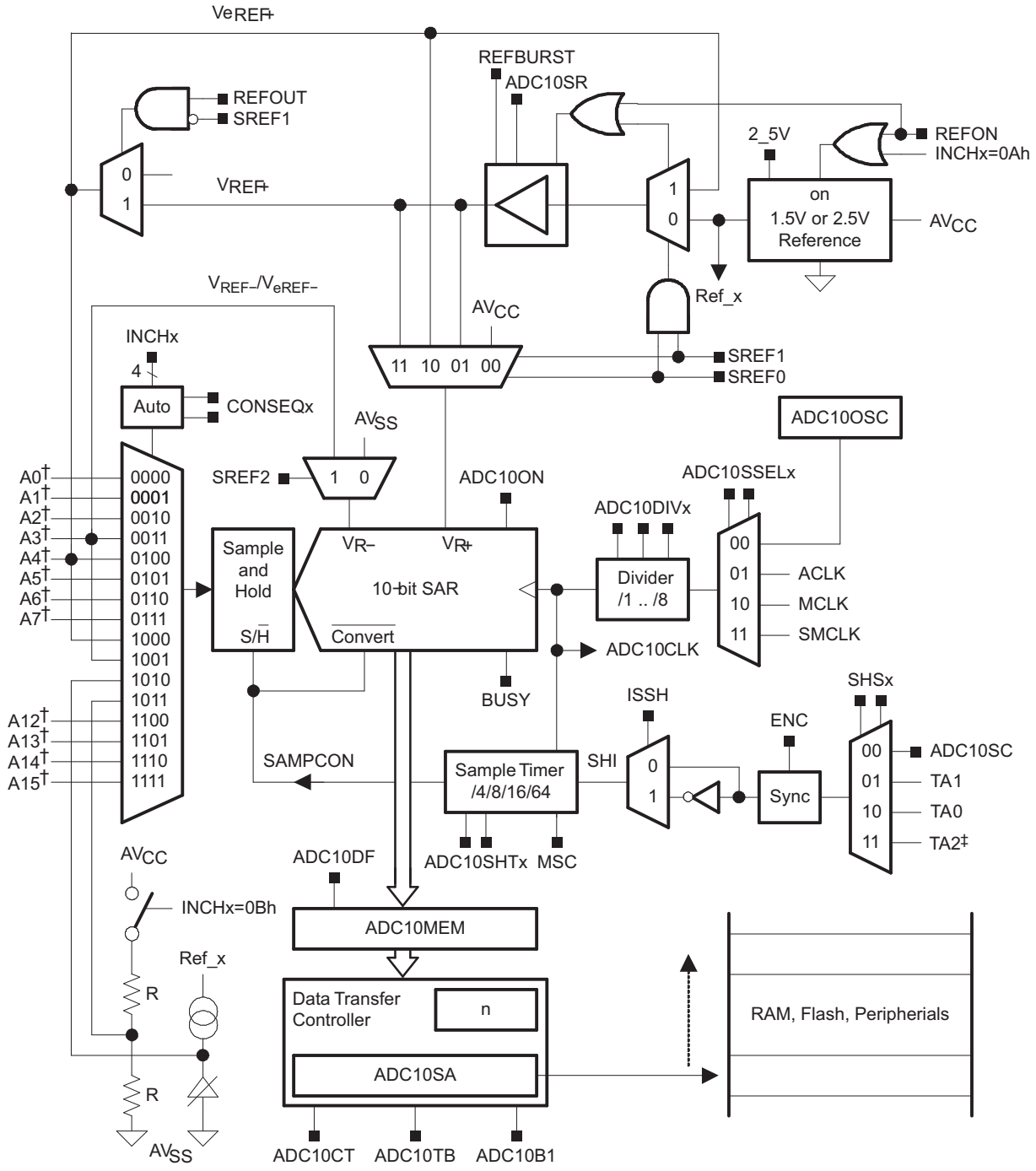
The ADC10 module supports fast, 10-bit analog-to-digital conversions. The module implements a 10-bit SAR core, sample select control, reference generator, and data transfer controller (DTC).

The DTC allows ADC10 samples to be converted and stored anywhere in memory without CPU intervention. The module can be configured with user software to support a variety of applications.

ADC10 features include:

- Greater than 200-ksps maximum conversion rate
- Monotonic 10-bit converter with no missing codes
- Sample-and-hold with programmable sample periods
- Conversion initiation by software or Timer_A
- Software selectable on-chip reference voltage generation (1.5 V or 2.5 V)
- Software selectable internal or external reference
- Up to eight external input channels (twelve on MSP430F22xx devices)
- Conversion channels for internal temperature sensor, V_{CC}, and external references
- Selectable conversion clock source
- Single-channel, repeated single-channel, sequence, and repeated sequence conversion modes
- ADC core and reference voltage can be powered down separately
- Data transfer controller for automatic storage of conversion results

The block diagram of ADC10 is shown in Figure 22-1.



†Channels A12-A15 are available in MSP430F22xx devices only. Channels A12-A15 tied to channel A11 in other devices. Not all channels are available in all devices.
‡TA1 on MSP430F20x2, MSP430G2x31, and MSP430G2x30 devices

Figure 22-1. ADC10 Block Diagram

22.2 ADC10 Operation

The ADC10 module is configured with user software. The setup and operation of the ADC10 is discussed in the following sections.

22.2.1 10-Bit ADC Core

The ADC core converts an analog input to its 10-bit digital representation and stores the result in the ADC10MEM register. The core uses two programmable/selectable voltage levels (V_{R+} and V_{R-}) to define the upper and lower limits of the conversion. The digital output (N_{ADC}) is full scale (03FFh) when the input signal is equal to or higher than V_{R+}, and zero when the input signal is equal to or lower than V_{R-}. The input channel and the reference voltage levels (V_{R+} and V_{R-}) are defined in the conversion-control memory. Conversion results may be in straight binary format or 2s-complement format. The conversion formula for the ADC result when using straight binary format is:

$$N_{ADC} = 1023 \times \frac{V_{IN} - V_{R-}}{V_{R+} - V_{R-}}$$

The ADC10 core is configured by two control registers, ADC10CTL0 and ADC10CTL1. The core is enabled with the ADC10ON bit. With few exceptions the ADC10 control bits can only be modified when ENC = 0. ENC must be set to 1 before any conversion can take place.

22.2.1.1 Conversion Clock Selection

The ADC10CLK is used both as the conversion clock and to generate the sampling period. The ADC10 source clock is selected using the ADC10SSELx bits and can be divided from 1 to 8 using the ADC10DIVx bits. Possible ADC10CLK sources are SMCLK, MCLK, ACLK, and internal oscillator ADC10OSC .

The ADC10OSC, generated internally, is in the 5-MHz range, but varies with individual devices, supply voltage, and temperature. See the device-specific data sheet for the ADC10OSC specification.

The user must ensure that the clock chosen for ADC10CLK remains active until the end of a conversion. If the clock is removed during a conversion, the operation does not complete, and any result is invalid.

22.2.2 ADC10 Inputs and Multiplexer

The eight external and four internal analog signals are selected as the channel for conversion by the analog input multiplexer. The input multiplexer is a break-before-make type to reduce input-to-input noise injection that can result from channel switching (see Figure 22-2). The input multiplexer is also a T-switch to minimize the coupling between channels. Channels that are not selected are isolated from the A/D, and the intermediate node is connected to analog ground (V_{SS}) so that the stray capacitance is grounded to help eliminate crosstalk.

The ADC10 uses the charge redistribution method. When the inputs are internally switched, the switching action may cause transients on the input signal. These transients decay and settle before causing errant conversion.

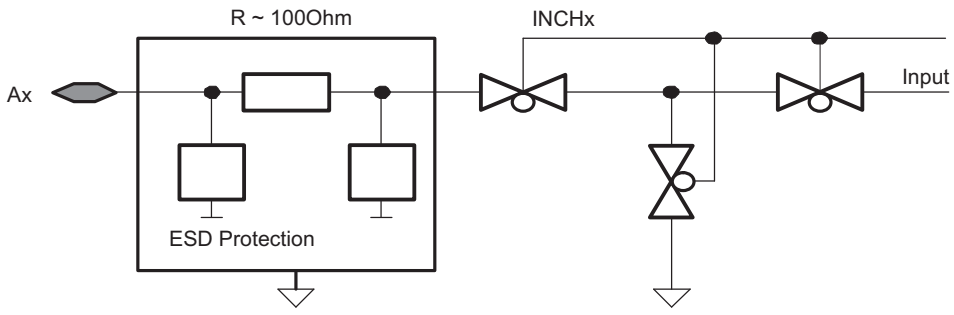


Figure 22-2. Analog Multiplexer

22.2.2.1 Analog Port Selection

The ADC10 external inputs Ax, V_{eREF+},and V_{REF-} share terminals with general purpose I/O ports, which are digital CMOS gates (see the device-specific data sheet). When analog signals are applied to digital CMOS gates, parasitic current can flow from VCC to GND. This parasitic current occurs if the input voltage is near the transition level of the gate. Disabling the port pin buffer eliminates the parasitic current flow and therefore reduces overall current consumption. The ADC10AEx bits provide the ability to disable the port pin input and output buffers.

```
; P2.3 on MSP430F22xx device configured for analog input
   BIS.B   #08h,&ADC10AE0   ; P2.3 ADC10 function and enable
```

Devices which don't have all the ADC10 external inputs channels Ax or V_{eREF+}/V_{REF+} and V_{eREF-}/V_{REF-} available at device pins must not alter the default register bit configuration of the not available pins. See device specific data sheet.

22.2.3 Voltage Reference Generator

The ADC10 module contains a built-in voltage reference with two selectable voltage levels. Setting REFON = 1 enables the internal reference. When REF2_5V = 1, the internal reference is 2.5 V. When REF2_5V = 0, the reference is 1.5 V. The internal reference voltage may be used internally (REFOUT = 0) and, when REFOUT = 1, externally on pin V_{REF+}. REFOUT = 1 should only be used if the pins $V_{REF+}$and V_{REF-} are available as device pins.

External references may be supplied for V_{R+} and V_{R-} through pins A4 and A3 respectively. When external references are used, or when V_{CC} is used as the reference, the internal reference may be turned off to save power.

An external positive reference V_{eREF+} can be buffered by setting SREF0 = 1 and SREF1 = 1 (only devices with V_{eREF+} pin). This allows using an external reference with a large internal resistance at the cost of the buffer current. When REFBURST = 1 the increased current consumption is limited to the sample and conversion period.

External storage capacitance is not required for the ADC10 reference source as on the ADC12.

22.2.3.1 Internal Reference Low-Power Features

The ADC10 internal reference generator is designed for low power applications. The reference generator includes a band-gap voltage source and a separate buffer. The current consumption of each is specified separately in the device-specific data sheet. When REFON = 1, both are enabled and when REFON = 0 both are disabled. The total settling time when REFON becomes set is approximately 30 µs.

When REFON = 1, but no conversion is active, the buffer is automatically disabled and automatically re-enabled when needed. When the buffer is disabled, it consumes no current. In this case, the bandgap voltage source remains enabled.

When REFOUT = 1, the REFBURST bit controls the operation of the internal reference buffer. When REFBURST = 0, the buffer is on continuously, allowing the reference voltage to be present outside the device continuously. When REFBURST = 1, the buffer is automatically disabled when the ADC10 is not actively converting and is automatically re-enabled when needed.

The internal reference buffer also has selectable speed versus power settings. When the maximum conversion rate is below 50 ksps, setting ADC10SR = 1 reduces the current consumption of the buffer approximately 50%.

22.2.4 Auto Power-Down

The ADC10 is designed for low power applications. When the ADC10 is not actively converting, the core is automatically disabled and is automatically re-enabled when needed. The ADC10OSC is also automatically enabled when needed and disabled when not needed. When the core or oscillator is disabled, it consumes no current.

22.2.5 Sample and Conversion Timing

An analog-to-digital conversion is initiated with a rising edge of sample input signal SHI. The source for SHI is selected with the SHSx bits and includes the following:

- The ADC10SC bit
- The Timer_A Output Unit 1
- The Timer_A Output Unit 0
- The Timer_A Output Unit 2

The polarity of the SHI signal source can be inverted with the ISSH bit. The SHTx bits select the sample period t_{sample} to be 4, 8, 16, or 64 ADC10CLK cycles. The sampling timer sets SAMPCON high for the selected sample period after synchronization with ADC10CLK.Total sampling time is t_{sample} plus t_{sync}.The high-to-low SAMPCON transition starts the analog-to-digital conversion, which requires 13 ADC10CLK cycles as shown in Figure 22-3.

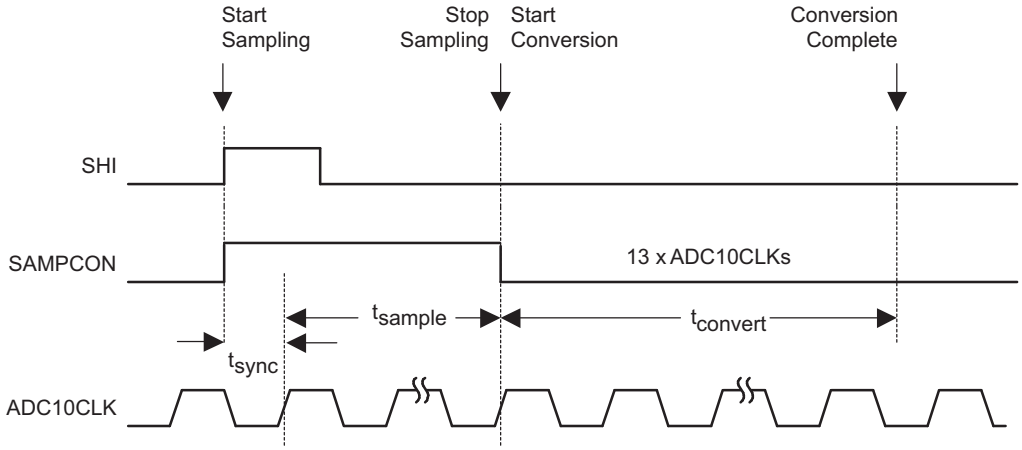


Figure 22-3. Sample Timing

22.2.5.1 Sample Timing Considerations

When SAMPCON = 0 all Ax inputs are high impedance. When SAMPCON = 1, the selected Ax input can be modeled as an RC low-pass filter during the sampling time t_{sample}, as shown in Figure 22-4. An internal MUX-on input resistance R_I (2 kΩ maximum) in series with capacitor C_I (27 pF maximum) is seen by the source. The capacitor C_I voltage V_C must be charged to within ½ LSB of the source voltage V_S for an accurate 10-bit conversion.

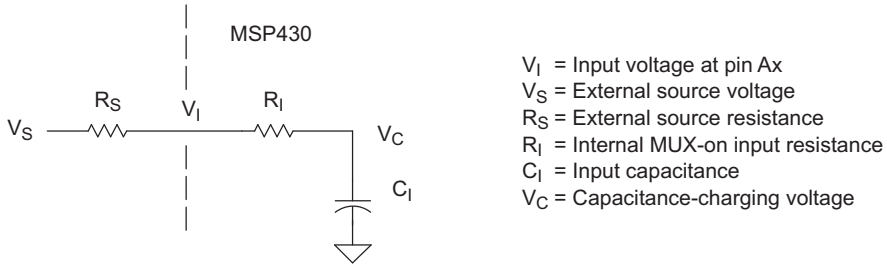


Figure 22-4. Analog Input Equivalent Circuit

The resistance of the source R_S and R_I affect t_{sample}.The following equations can be used to calculate the minimum sampling time for a 10-bit conversion.

$$t_{sample} > (R_S + R_I) \times \ln(2^{11}) \times C_I$$

Substituting the values for R_I and C_I given above, the equation becomes:

$$t_{sample} > (R_S + 2\ k\Omega) \times 7.625 \times 27\ pF$$

For example, if R_S is 10 kΩ, t_{sample} must be greater than 2.47 µs.

When the reference buffer is used in burst mode, the sampling time must be greater than the sampling time calculated and the settling time of the buffer, $t_{REFBURST}$:

$$t_{sample} > \begin{cases} (R_S + R_I) \times \ln(2^{11}) \times C_I \\ t_{REFBURST} \end{cases}$$

For example, if V_{Ref} is 1.5 V and R_S is 10 kΩ, t_{sample} must be greater than 2.47 µs when ADC10SR = 0, or 2.5 µs when ADC10SR = 1. See the device-specific data sheet for parameters.

To calculate the buffer settling time when using an external reference, the formula is:

$t_{REFBURST} = S_R \times V_{Ref} - 0.5\ \mu s$

Where:

SR = Buffer slew rate (~1 µs/V when ADC10SR = 0 and ~2 µs/V when ADC10SR = 1)

V_{Ref} = External reference voltage

22.2.6 Conversion Modes

The ADC10 has four operating modes selected by the CONSEQx bits as discussed in Table 22-1.

Table 22-1. Conversion Mode Summary

CONSEQx	Mode	Operation
00	Single channel single-conversion	A single channel is converted once.
01	Sequence-of-channels	A sequence of channels is converted once.
10	Repeat single channel	A single channel is converted repeatedly.
11	Repeat sequence-of-channels	A sequence of channels is converted repeatedly.

22.2.6.1 Single-Channel Single-Conversion Mode

A single channel selected by INCHx is sampled and converted once. The ADC result is written to ADC10MEM. Figure 22-5 shows the flow of the single-channel, single-conversion mode. When ADC10SC triggers a conversion, successive conversions can be triggered by the ADC10SC bit. When any other trigger source is used, ENC must be toggled between each conversion.

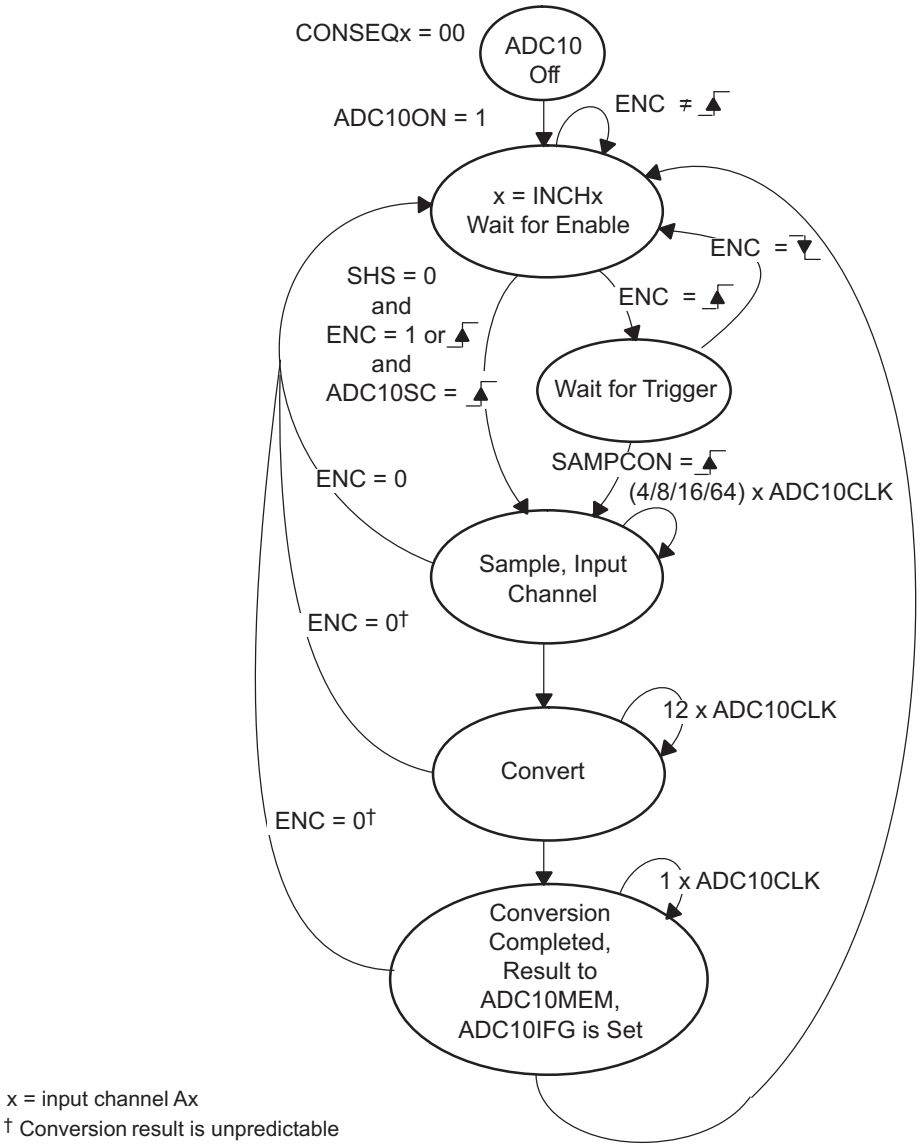


Figure 22-5. Single-Channel Single-Conversion Mode

22.2.6.2 Sequence-of-Channels Mode

A sequence of channels is sampled and converted once. The sequence begins with the channel selected by INCHx and decrements to channel A0. Each ADC result is written to ADC10MEM. The sequence stops after conversion of channel A0. Figure 22-6 shows the sequence-of-channels mode. When ADC10SC triggers a sequence, successive sequences can be triggered by the ADC10SC bit. When any other trigger source is used, ENC must be toggled between each sequence.

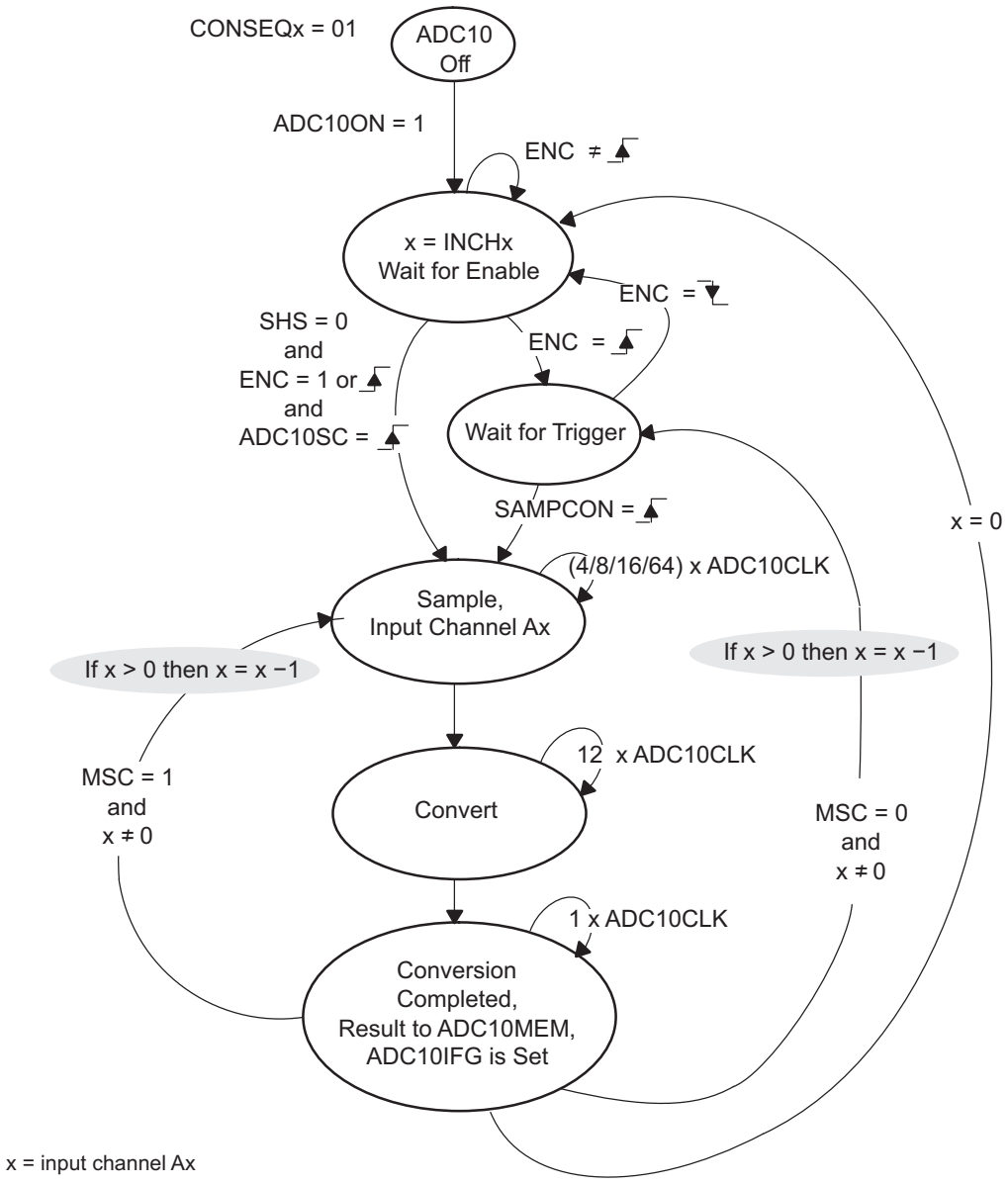


Figure 22-6. Sequence-of-Channels Mode

22.2.6.3 Repeat-Single-Channel Mode

A single channel selected by INCHx is sampled and converted continuously. Each ADC result is written to ADC10MEM. Figure 22-7 shows the repeat-single-channel mode.

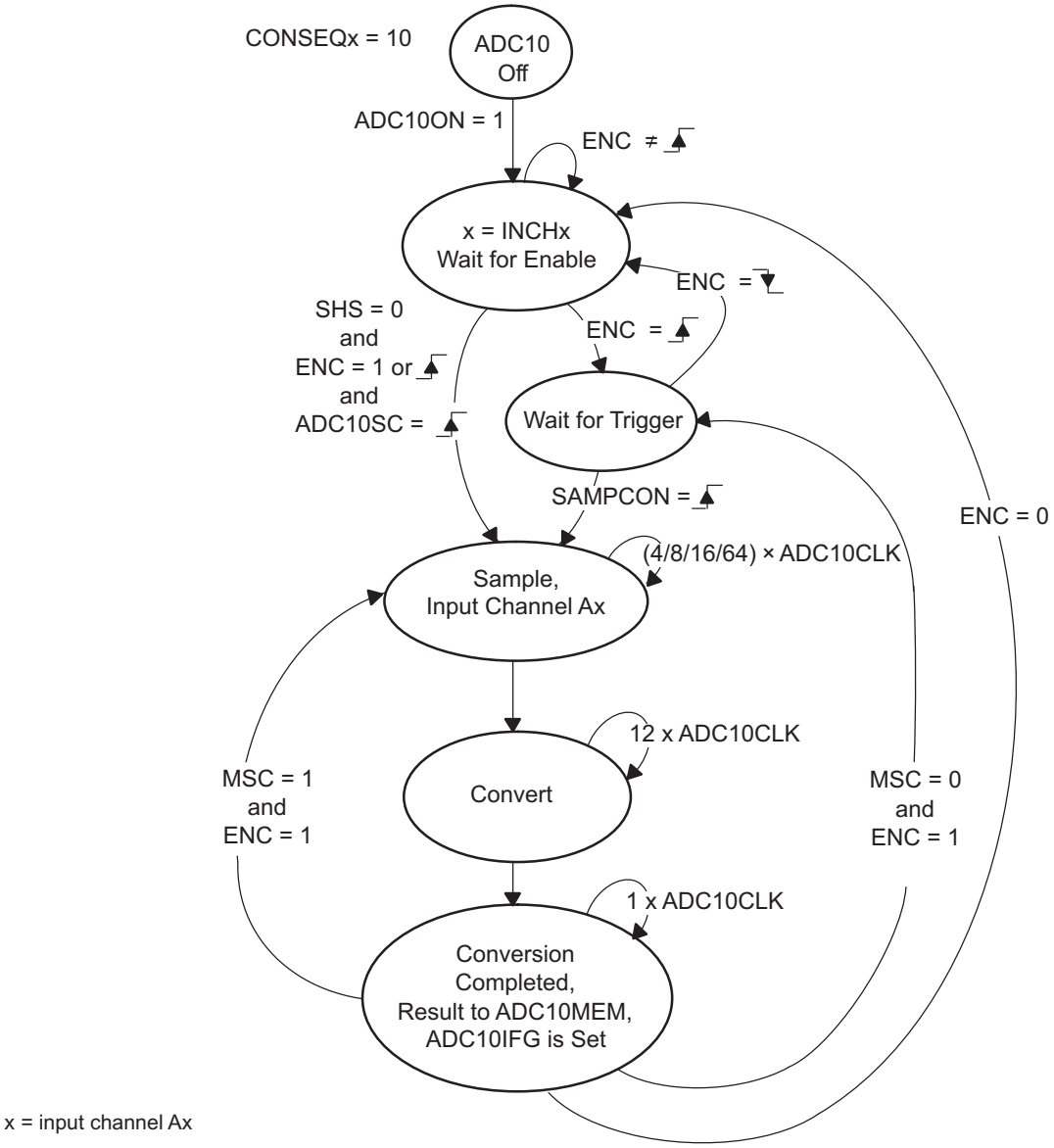


Figure 22-7. Repeat-Single-Channel Mode

22.2.6.4 Repeat-Sequence-of-Channels Mode

A sequence of channels is sampled and converted repeatedly. The sequence begins with the channel selected by INCHx and decrements to channel A0. Each ADC result is written to ADC10MEM. The sequence ends after conversion of channel A0, and the next trigger signal re-starts the sequence. Figure 22-8 shows the repeat-sequence-of-channels mode.

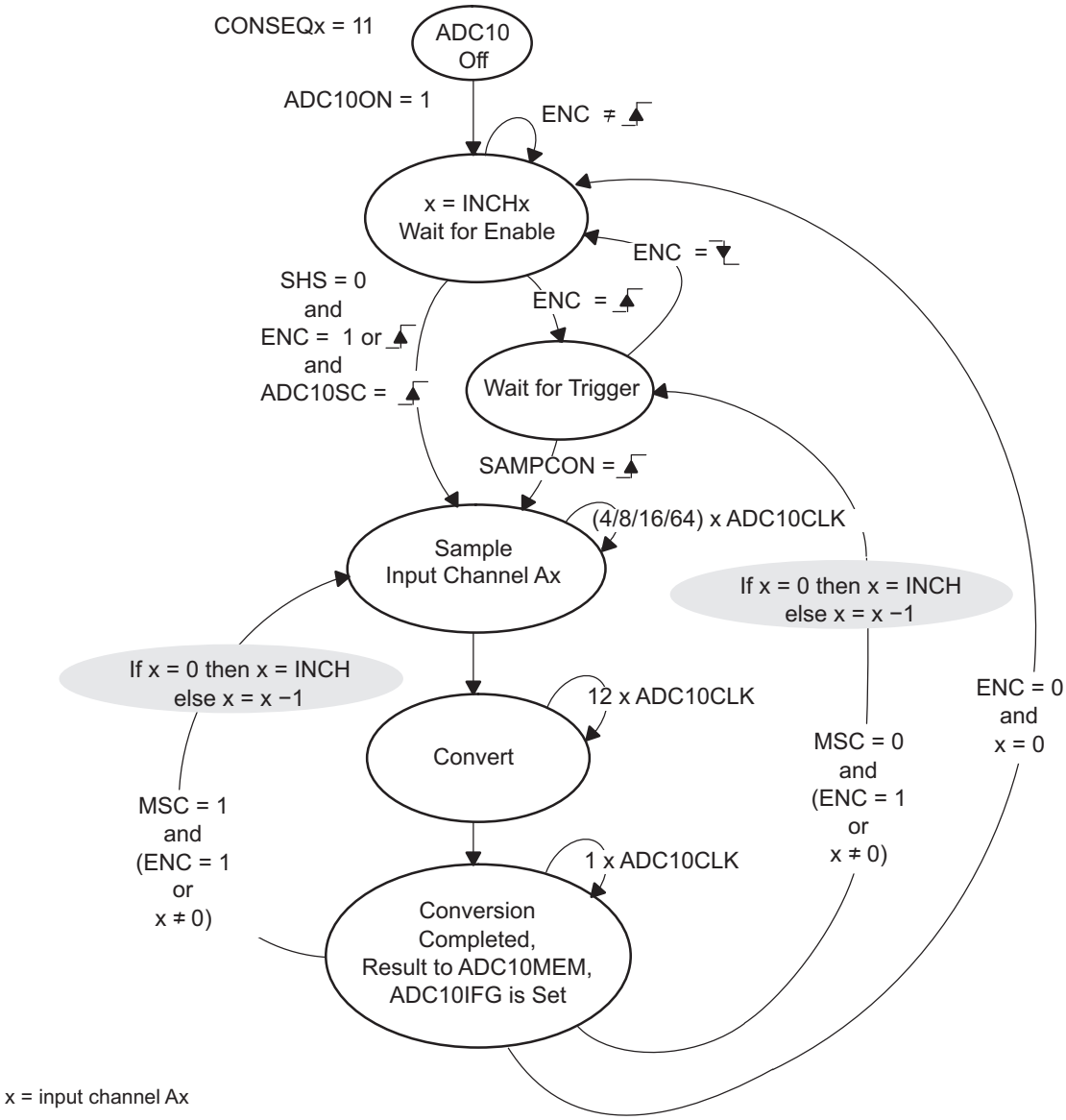


Figure 22-8. Repeat-Sequence-of-Channels Mode

22.2.6.5 Using the MSC Bit

To configure the converter to perform successive conversions automatically and as quickly as possible, a multiple sample and convert function is available. When MSC = 1 and CONSEQx > 0, the first rising edge of the SHI signal triggers the first conversion. Successive conversions are triggered automatically as soon as the prior conversion is completed. Additional rising edges on SHI are ignored until the sequence is completed in the single-sequence mode or until the ENC bit is toggled in repeat-single-channel, or repeated-sequence modes. The function of the ENC bit is unchanged when using the MSC bit.

22.2.6.6 Stopping Conversions

Stopping ADC10 activity depends on the mode of operation. The recommended ways to stop an active conversion or conversion sequence are:

- Resetting ENC in single-channel single-conversion mode stops a conversion immediately and the results are unpredictable. For correct results, poll the ADC10BUSY bit until reset before clearing ENC.
- Resetting ENC during repeat-single-channel operation stops the converter at the end of the current conversion.
- Resetting ENC during a sequence or repeat sequence mode stops the converter at the end of the sequence.
- Any conversion mode may be stopped immediately by setting the CONSEQx = 0 and resetting the ENC bit. Conversion data is unreliable.

22.2.7 ADC10 Data Transfer Controller

The ADC10 includes a data transfer controller (DTC) to automatically transfer conversion results from ADC10MEM to other on-chip memory locations. The DTC is enabled by setting the ADC10DTC1 register to a nonzero value.

When the DTC is enabled, each time the ADC10 completes a conversion and loads the result to ADC10MEM, a data transfer is triggered. No software intervention is required to manage the ADC10 until the predefined amount of conversion data has been transferred. Each DTC transfer requires one CPU MCLK. To avoid any bus contention during the DTC transfer, the CPU is halted, if active, for the one MCLK required for the transfer.

A DTC transfer must not be initiated while the ADC10 is busy. Software must ensure that no active conversion or sequence is in progress when the DTC is configured:

```
; ADC10 activity test
           BIC.W   #ENC,&ADC10CTL0   ;
busy_test  BIT.W   #BUSY,&ADC10CTL1  ;
           JNZ     busy_test        ;
           MOV.W   #xxx,&ADC10SA     ; Safe
           MOV.B   #xx,&ADC10DTC1    ;
; continue setup
```

22.2.7.1 One-Block Transfer Mode

The one-block mode is selected if the ADC10TB is reset. The value n in ADC10DTC1 defines the total number of transfers for a block. The block start address is defined anywhere in the MSP430 address range using the 16-bit register ADC10SA. The block ends at ADC10SA + 2n – 2. The one-block transfer mode is shown in Figure 22-9.

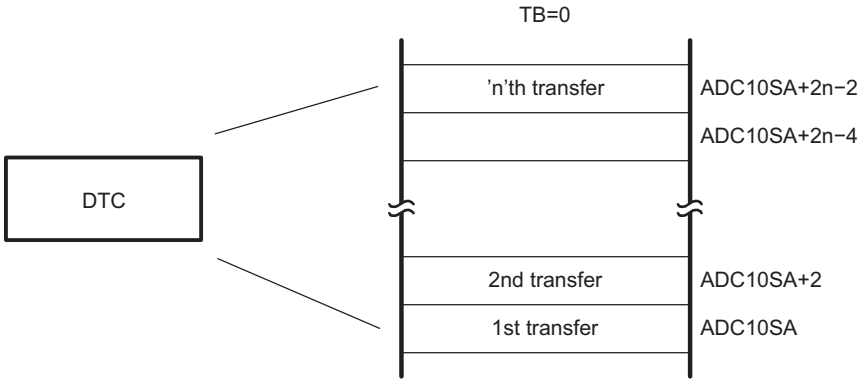


Figure 22-9. One-Block Transfer

The internal address pointer is initially equal to ADC10SA and the internal transfer counter is initially equal to 'n'. The internal pointer and counter are not visible to software. The DTC transfers the word-value of ADC10MEM to the address pointer ADC10SA. After each DTC transfer, the internal address pointer is incremented by two and the internal transfer counter is decremented by one.

The DTC transfers continue with each loading of ADC10MEM, until the internal transfer counter becomes equal to zero. No additional DTC transfers occur until a write to ADC10SA. When using the DTC in the one-block mode, the ADC10IFG flag is set only after a complete block has been transferred. Figure 22-10 shows a state diagram of the one-block mode.

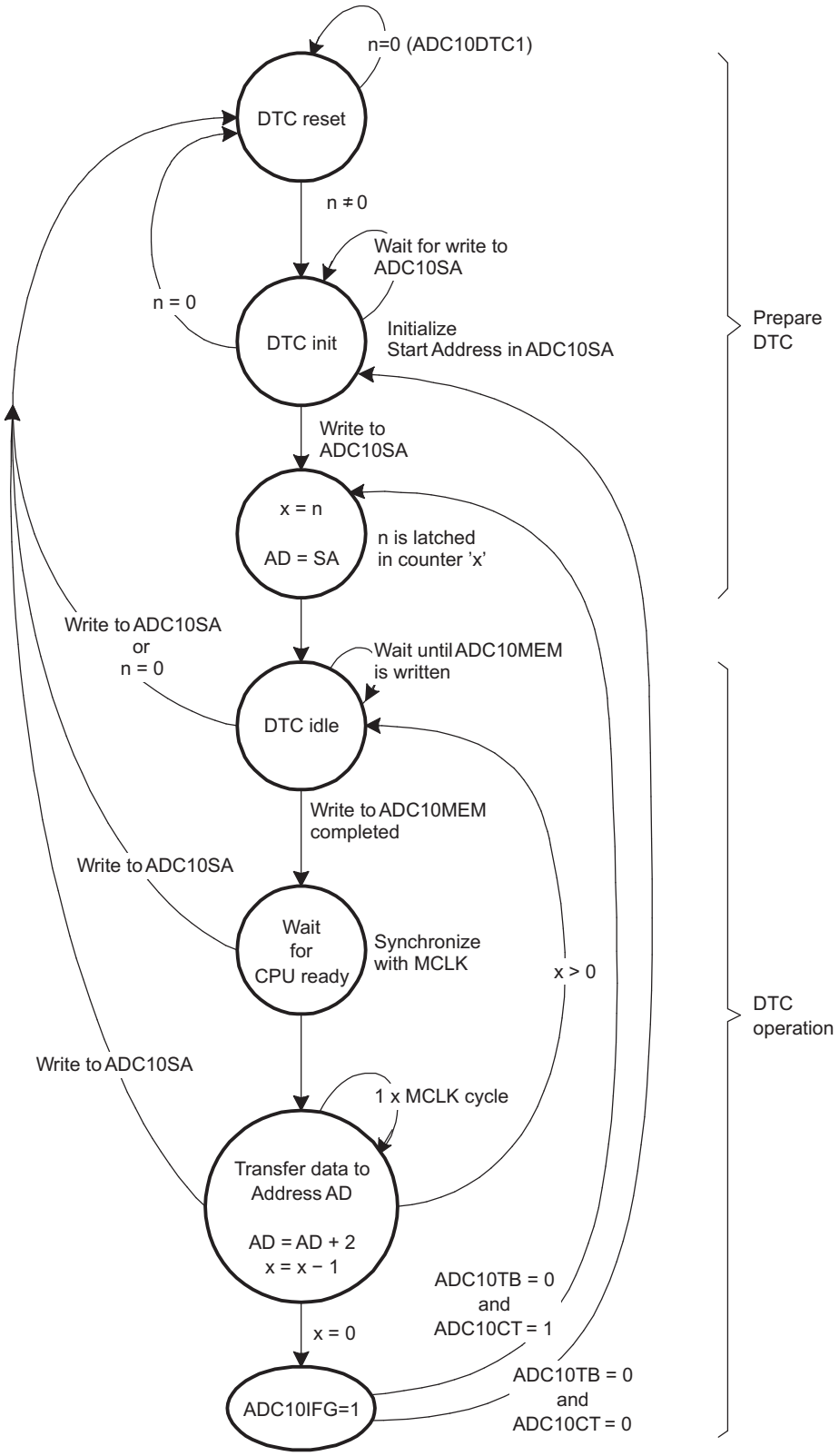


Figure 22-10. State Diagram for Data Transfer Control in One-Block Transfer Mode

22.2.7.2 Two-Block Transfer Mode

The two-block mode is selected if the ADC10TB bit is set. The value n in ADC10DTC1 defines the number of transfers for one block. The address range of the first block is defined anywhere in the MSP430 address range with the 16-bit register ADC10SA. The first block ends at ADC10SA+2n-2. The address range for the second block is defined as SA+2n to SA+4n-2. The two-block transfer mode is shown in Figure 22-11.

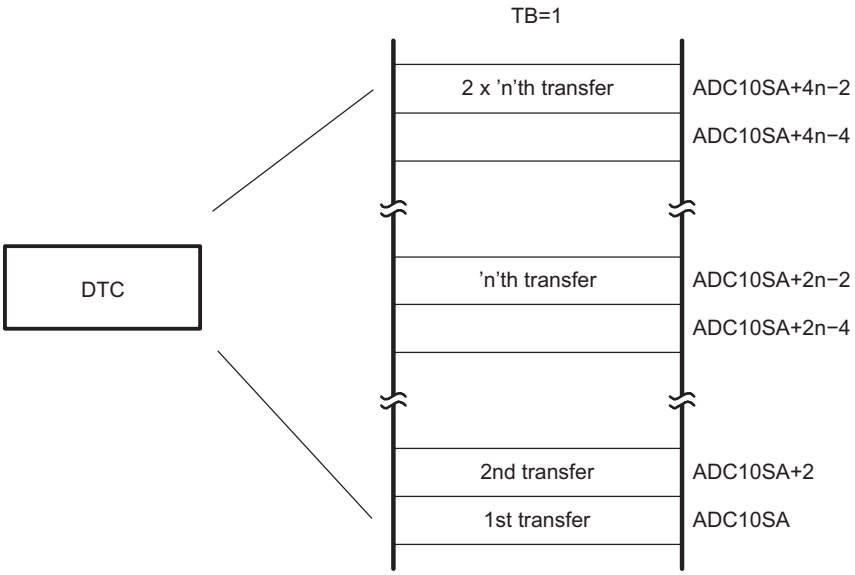


Figure 22-11. Two-Block Transfer

The internal address pointer is initially equal to ADC10SA and the internal transfer counter is initially equal to 'n'. The internal pointer and counter are not visible to software. The DTC transfers the word-value of ADC10MEM to the address pointer ADC10SA. After each DTC transfer the internal address pointer is incremented by two and the internal transfer counter is decremented by one.

The DTC transfers continue, with each loading of ADC10MEM, until the internal transfer counter becomes equal to zero. At this point, block one is full and both the ADC10IFG flag the ADC10B1 bit are set. The user can test the ADC10B1 bit to determine that block one is full.

The DTC continues with block two. The internal transfer counter is automatically reloaded with 'n'. At the next load of the ADC10MEM, the DTC begins transferring conversion results to block two. After n transfers have completed, block two is full. The ADC10IFG flag is set and the ADC10B1 bit is cleared. User software can test the cleared ADC10B1 bit to determine that block two is full. Figure 22-12 shows a state diagram of the two-block mode.

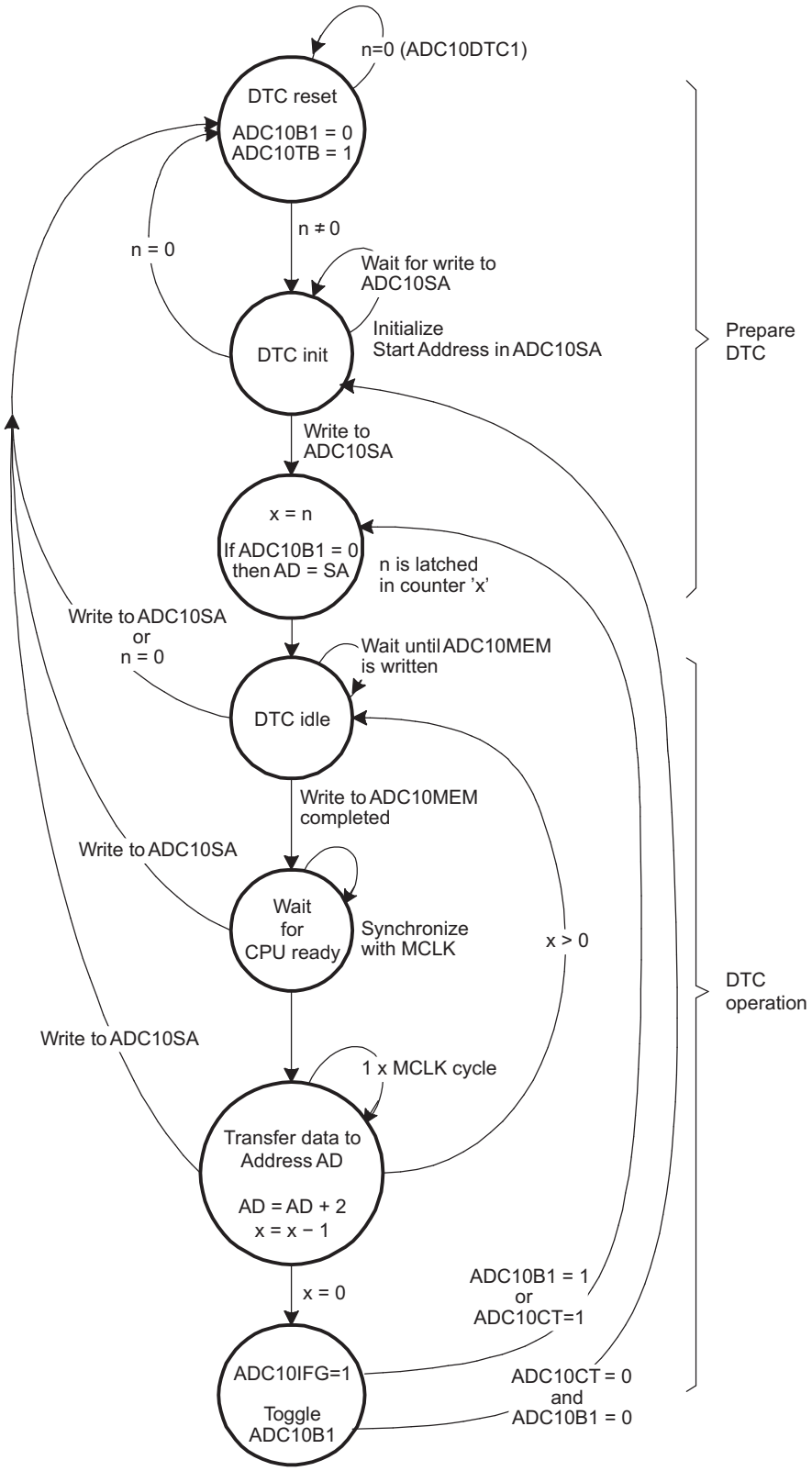


Figure 22-12. State Diagram for Data Transfer Control in Two-Block Transfer Mode

22.2.7.3 Continuous Transfer

A continuous transfer is selected if ADC10CT bit is set. The DTC does not stop after block one (in one-block mode) or block two (in two-block mode) has been transferred. The internal address pointer and transfer counter are set equal to ADC10SA and n respectively. Transfers continue starting in block one. If the ADC10CT bit is reset, DTC transfers cease after the current completion of transfers into block one (in one-block mode) or block two (in two-block mode) have been transferred.

22.2.7.4 DTC Transfer Cycle Time

For each ADC10MEM transfer, the DTC requires one or two MCLK clock cycles to synchronize, one for the actual transfer (while the CPU is halted), and one cycle of wait time. Because the DTC uses MCLK, the DTC cycle time is dependent on the MSP430 operating mode and clock system setup.

If the MCLK source is active but the CPU is off, the DTC uses the MCLK source for each transfer, without re-enabling the CPU. If the MCLK source is off, the DTC temporarily restarts MCLK, sourced with DCOCLK, only during a transfer. The CPU remains off, and MCLK is again turned off after the DTC transfer. The maximum DTC cycle time for all operating modes is show in Table 22-2.

Table 22-2. Maximum DTC Cycle Time

CPU Operating Mode	Clock Source	Maximum DTC Cycle Time
Active mode	MCLK = DCOCLK	3 MCLK cycles
Active mode	MCLK = LFXT1CLK	3 MCLK cycles
Low-power mode LPM0/1	MCLK = DCOCLK	4 MCLK cycles
Low-power mode LPM3/4	MCLK = DCOCLK	4 MCLK cycles + 2 µs(1)
Low-power mode LPM0/1	MCLK = LFXT1CLK	4 MCLK cycles
Low-power mode LPM3	MCLK = LFXT1CLK	4 MCLK cycles
Low-power mode LPM4	MCLK = LFXT1CLK	4 MCLK cycles + 2 µs(1)

(1) The additional 2 µs are needed to start the DCOCLK. See the device-specific data sheet for parameters.

22.2.8 Using the Integrated Temperature Sensor

To use the on-chip temperature sensor, select the analog input channel INCHx = 1010. Any other configuration is done as if an external channel was selected, including reference selection, conversion-memory selection, etc.

The typical temperature sensor transfer function is shown in Figure 22-13. When using the temperature sensor, the sample period must be greater than 30 µs. The temperature sensor offset error is large. Deriving absolute temperature values in the application requires calibration. See the device-specific data sheet for the parameters. See Section 24.2.2.1 for the calibration equations.

Selecting the temperature sensor automatically turns on the on-chip reference generator as a voltage source for the temperature sensor. However, it does not enable the V_{REF+} output or affect the reference selections for the conversion. The reference choices for converting the temperature sensor are the same as with any other channel.

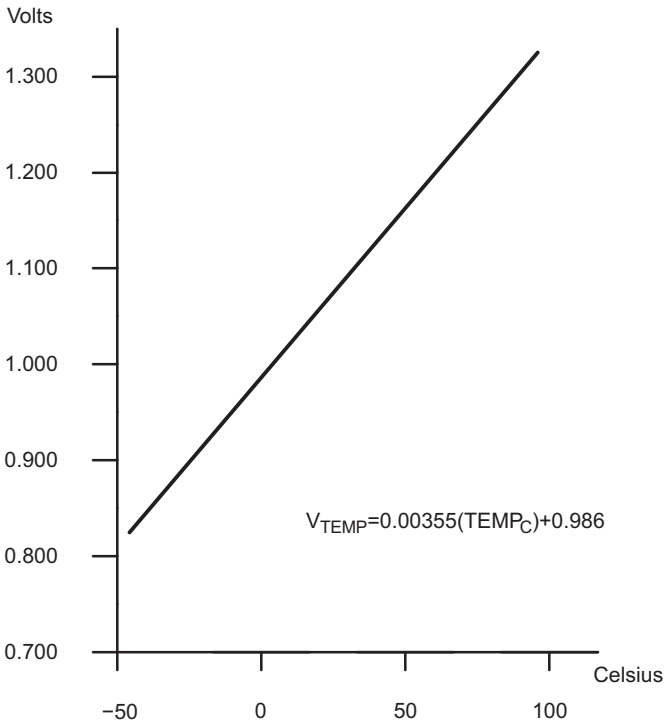


Figure 22-13. Typical Temperature Sensor Transfer Function

22.2.9 ADC10 Grounding and Noise Considerations

As with any high-resolution ADC, appropriate printed-circuit-board layout and grounding techniques should be followed to eliminate ground loops, unwanted parasitic effects, and noise.

Ground loops are formed when return current from the A/D flows through paths that are common with other analog or digital circuitry. If care is not taken, this current can generate small, unwanted offset voltages that can add to or subtract from the reference or input voltages of the A/D converter. The connections shown in Figure 22-14 and Figure 22-15 help avoid this.

In addition to grounding, ripple and noise spikes on the power supply lines due to digital switching or switching power supplies can corrupt the conversion result. A noise-free design is important to achieve high accuracy.

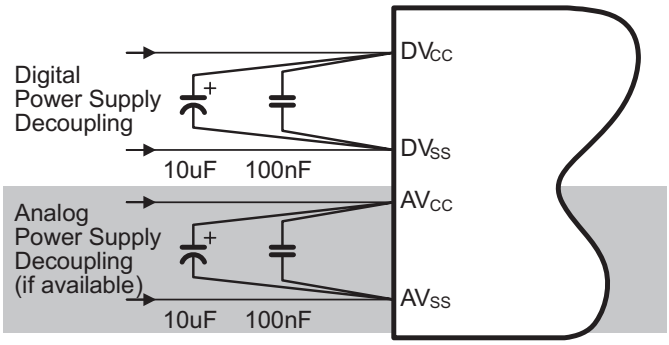


Figure 22-14. ADC10 Grounding and Noise Considerations (Internal V_{REF})

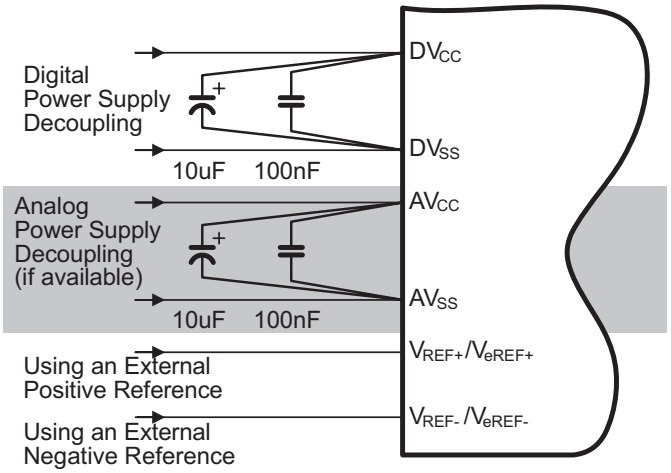


Figure 22-15. ADC10 Grounding and Noise Considerations (External V_{REF})

22.2.10 ADC10 Interrupts

One interrupt and one interrupt vector are associated with the ADC10 as shown in Figure 22-16. When the DTC is not used (ADC10DTC1 = 0), ADC10IFG is set when conversion results are loaded into ADC10MEM. When DTC is used (ADC10DTC1 > 0), ADC10IFG is set when a block transfer completes and the internal transfer counter n = 0. If both the ADC10IE and the GIE bits are set, then the ADC10IFG flag generates an interrupt request. The ADC10IFG flag is automatically reset when the interrupt request is serviced, or it may be reset by software.

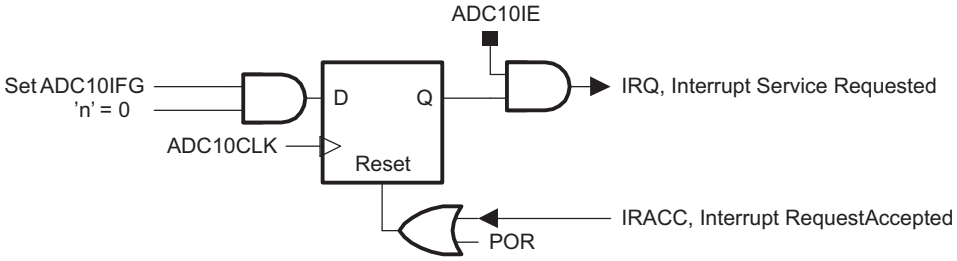


Figure 22-16. ADC10 Interrupt System

22.3 ADC10 Registers

The ADC10 registers are listed in Table 22-3.

Table 22-3. ADC10 Registers

Register	Short Form	Register Type	Address	Initial State
ADC10 input enable register 0	ADC10AE0	Read/write	04Ah	Reset with POR
ADC10 input enable register 1	ADC10AE1	Read/write	04Bh	Reset with POR
ADC10 control register 0	ADC10CTL0	Read/write	01B0h	Reset with POR
ADC10 control register 1	ADC10CTL1	Read/write	01B2h	Reset with POR
ADC10 memory	ADC10MEM	Read	01B4h	Unchanged
ADC10 data transfer control register 0	ADC10DTC0	Read/write	048h	Reset with POR
ADC10 data transfer control register 1	ADC10DTC1	Read/write	049h	Reset with POR
ADC10 data transfer start address	ADC10SA	Read/write	01BCh	0200h with POR

22.3.1 ADC10CTL0, ADC10 Control Register 0

15	14	13	12	11	10	9	8
SREFx			**ADC10SHTx**		**ADC10SR**	**REFOUT**	**REFBURST**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
MSC	**REF2_5V**	**REFON**	**ADC10ON**	**ADC10IE**	**ADC10IFG**	**ENC**	**ADC10SC**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Can be modified only when ENC = 0

SREFx Bits 15-13 Select reference.

- 000 $V_{R+} = V_{CC}$ and $V_{R-} = V_{SS}$
- 001 $V_{R+} = V_{REF+}$ and $V_{R-} = V_{SS}$
- 010 $V_{R+} = V_{eREF+}$ and $V_{R-} = V_{SS}$. Devices with V_{eREF+} only.
- 011 V_{R+} = Buffered V_{eREF+} and $V_{R-} = V_{SS}$. Devices with V_{eREF+} pin only.
- 100 $V_{R+} = V_{CC}$ and $V_{R-} = V_{REF-}/ V_{eREF-}$. Devices with V_{eREF-} pin only.
- 101 $V_{R+} = V_{REF+}$ and $V_{R-} = V_{REF-}/ V_{eREF-}$. Devices with $V_{eREF+/-}$ pins only.
- 110 $V_{R+} = V_{eREF+}$ and $V_{R-} = V_{REF-}/ V_{eREF-}$. Devices with $V_{eREF+/-}$ pins only.
- 111 V_{R+} = Buffered V_{eREF+} and $V_{R-} = V_{REF-}/ V_{eREF-}$. Devices with $V_{eREF+/-}$ pins only.

ADC10SHTx Bits 12-11 ADC10 sample-and-hold time

- 00 4 × ADC10CLKs
- 01 8 × ADC10CLKs
- 10 16 × ADC10CLKs
- 11 64 × ADC10CLKs

ADC10SR Bit 10 ADC10 sampling rate. This bit selects the reference buffer drive capability for the maximum sampling rate. Setting ADC10SR reduces the current consumption of the reference buffer.

- 0 Reference buffer supports up to ~200 ksps
- 1 Reference buffer supports up to ~50 ksps

REFOUT Bit 9 Reference output

- 0 Reference output off
- 1 Reference output on. Devices with V_{eREF+} / V_{REF+} pin only.

REFBURST Bit 8 Reference burst.

- 0 Reference buffer on continuously
- 1 Reference buffer on only during sample-and-conversion

MSC Bit 7 Multiple sample and conversion. Valid only for sequence or repeated modes.

- 0 The sampling requires a rising edge of the SHI signal to trigger each sample-and-conversion.
- 1 The first rising edge of the SHI signal triggers the sampling timer, but further sample-and-conversions are performed automatically as soon as the prior conversion is completed

REF2_5V Bit 6 Reference-generator voltage. REFON must also be set.

- 0 1.5 V
- 1 2.5 V

REFON Bit 5 Reference generator on

- 0 Reference off
- 1 Reference on

ADC10ON Bit 4 ADC10 on

- 0 ADC10 off
- 1 ADC10 on

ADC10IE Bit 3 ADC10 interrupt enable

- 0 Interrupt disabled
- 1 Interrupt enabled

ADC10IFG	Bit 2	ADC10 interrupt flag. This bit is set if ADC10MEM is loaded with a conversion result. It is automatically reset when the interrupt request is accepted, or it may be reset by software. When using the DTC this flag is set when a block of transfers is completed. 0 No interrupt pending 1 Interrupt pending
ENC	Bit 1	Enable conversion 0 ADC10 disabled 1 ADC10 enabled
ADC10SC	Bit 0	Start conversion. Software-controlled sample-and-conversion start. ADC10SC and ENC may be set together with one instruction. ADC10SC is reset automatically. 0 No sample-and-conversion start 1 Start sample-and-conversion

22.3.2 ADC10CTL1, ADC10 Control Register 1

15	14	13	12	11	10	9	8
INCHx				**SHSx**		**ADC10DF**	**ISSH**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC10DIVx			**ADC10SSELx**		**CONSEQx**		**ADC10BUSY**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-0

Can be modified only when ENC = 0

INCHx Bits 15-12 Input channel select. These bits select the channel for a single-conversion or the highest channel for a sequence of conversions. Only available ADC channels should be selected. See device specific data sheet.

- 0000 A0
- 0001 A1
- 0010 A2
- 0011 A3
- 0100 A4
- 0101 A5
- 0110 A6
- 0111 A7
- 1000 V_{eREF+}
- 1001 V_{REF-}/V_{eREF-}
- 1010 Temperature sensor
- 1011 $(V_{CC} - V_{SS}) / 2$
- 1100 $(V_{CC} - V_{SS}) / 2$, A12 on MSP430F22xx devices
- 1101 $(V_{CC} - V_{SS}) / 2$, A13 on MSP430F22xx devices
- 1110 $(V_{CC} - V_{SS}) / 2$, A14 on MSP430F22xx devices
- 1111 $(V_{CC} - V_{SS}) / 2$, A15 on MSP430F22xx devices

SHSx Bits 11-10 Sample-and-hold source select.

- 00 ADC10SC bit
- 01 Timer_A.OUT1 (1)
- 10 Timer_A.OUT0 (1)
- 11 Timer_A.OUT2 (Timer_A.OUT1 on MSP430F20x0, MSP430G2x31, and MSP430G2x30 devices) (1)

ADC10DF Bit 9 ADC10 data format

- 0 Straight binary
- 1 2s complement

ISSH Bit 8 Invert signal sample-and-hold

- 0 The sample-input signal is not inverted.
- 1 The sample-input signal is inverted.

ADC10DIVx Bits 7-5 ADC10 clock divider

- 000 /1
- 001 /2
- 010 /3
- 011 /4
- 100 /5
- 101 /6
- 110 /7
- 111 /8

ADC10SSELx Bits 4-3 ADC10 clock source select

- 00 ADC10OSC
- 01 ACLK
- 10 MCLK
- 11 SMCLK

(1) Timer triggers are from Timer0_Ax if more than one timer module exists on the device.

CONSEQx	Bits 2-1	Conversion sequence mode select
	00	Single-channel-single-conversion
	01	Sequence-of-channels
	10	Repeat-single-channel
	11	Repeat-sequence-of-channels
ADC10BUSY	Bit 0	ADC10 busy. This bit indicates an active sample or conversion operation
	0	No operation is active.
	1	A sequence, sample, or conversion is active.

22.3.3 ADC10AE0, Analog (Input) Enable Control Register 0

7	6	5	4	3	2	1	0
ADC10AE0x							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

ADC10AE0x	Bits 7-0	ADC10 analog enable. These bits enable the corresponding pin for analog input. BIT0 corresponds to A0, BIT1 corresponds to A1, etc. The analog enable bit of not implemented channels should not be programmed to 1.
	0	Analog input disabled
	1	Analog input enabled

22.3.4 ADC10AE1, Analog (Input) Enable Control Register 1 (MSP430F22xx only)

7	6	5	4	3	2	1	0
ADC10AE1x				**Reserved**			
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

ADC10AE1x	Bits 7-4	ADC10 analog enable. These bits enable the corresponding pin for analog input. BIT4 corresponds to A12, BIT5 corresponds to A13, BIT6 corresponds to A14, and BIT7 corresponds to A15. The analog enable bit of not implemented channels should not be programmed to 1.
	0	Analog input disabled
	1	Analog input enabled
Reserved	Bits 3-0	Reserved

22.3.5 ADC10MEM, Conversion-Memory Register, Binary Format

15	14	13	12	11	10	9	8
0	0	0	0	0	0	**Conversion Results**	
r0	r0	r0	r0	r0	r0	r	r

7	6	5	4	3	2	1	0
Conversion Results							
r	r	r	r	r	r	r	r

Conversion Results	Bits 15-0	The 10-bit conversion results are right justified, straight-binary format. Bit 9 is the MSB. Bits 15-10 are always 0.

22.3.6 ADC10MEM, Conversion-Memory Register, 2s Complement Format

15	14	13	12	11	10	9	8
Conversion Results							
r	r	r	r	r	r	r	r
7	6	5	4	3	2	1	0
Conversion Results		0	0	0	0	0	0
r	r	r0	r0	r0	r0	r0	r0

Conversion Results Bits 15-0 The 10-bit conversion results are left-justified, 2s complement format. Bit 15 is the MSB. Bits 5-0 are always 0.

22.3.7 ADC10DTC0, Data Transfer Control Register 0

7	6	5	4	3	2	1	0
Reserved				**ADC10TB**	**ADC10CT**	**ADC10B1**	**ADC10FETCH**
r0	r0	r0	r0	rw-(0)	rw-(0)	r-(0)	rw-(0)

Reserved Bits 7-4 Reserved. Always read as 0.

ADC10TB Bit 3 ADC10 two-block mode

- 0 One-block transfer mode
- 1 Two-block transfer mode

ADC10CT Bit 2 ADC10 continuous transfer

- 0 Data transfer stops when one block (one-block mode) or two blocks (two-block mode) have completed.
- 1 Data is transferred continuously. DTC operation is stopped only if ADC10CT cleared, or ADC10SA is written to.

ADC10B1 Bit 1 ADC10 block one. This bit indicates for two-block mode which block is filled with ADC10 conversion results. ADC10B1 is valid only after ADC10IFG has been set the first time during DTC operation. ADC10TB must also be set.

- 0 Block 2 is filled
- 1 Block 1 is filled

ADC10FETCH Bit 0 This bit should normally be reset.

22.3.8 ADC10DTC1, Data Transfer Control Register 1

7	6	5	4	3	2	1	0
DTC Transfers							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

DTC Transfers Bits 7-0 DTC transfers. These bits define the number of transfers in each block.

- 0 DTC is disabled
- 01h-0FFh Number of transfers per block

22.3.9 ADC10SA, Start Address Register for Data Transfer

15	14	13	12	11	10	9	8
ADC10SAx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC10SAx							0
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r0

ADC10SAx	Bits 15-1	ADC10 start address. These bits are the start address for the DTC. A write to register ADC10SA is required to initiate DTC transfers.
Unused	Bit 0	Unused, Read only. Always read as 0.

ADC12

The ADC12 module is a high-performance 12-bit analog-to-digital converter. This chapter describes the ADC12 of the MSP430x2xx device family.

Topic **Page**

23.1 ADC12 Introduction

The ADC12 module supports fast 12-bit analog-to-digital conversions. The module implements a 12-bit SAR core, sample select control, reference generator, and a 16-word conversion-and-control buffer. The conversion-and-control buffer allows up to 16 independent ADC samples to be converted and stored without any CPU intervention.

ADC12 features include:

- Greater than 200-ksps maximum conversion rate
- Monotonic 12-bit converter with no missing codes
- Sample-and-hold with programmable sampling periods controlled by software or timers
- Conversion initiation by software, Timer_A, or Timer_B
- Software selectable on-chip reference voltage generation (1.5 V or 2.5 V)
- Software selectable internal or external reference
- Eight individually configurable external input channels
- Conversion channels for internal temperature sensor, AV_{CC}, and external references
- Independent channel-selectable reference sources for both positive and negative references
- Selectable conversion clock source
- Single-channel, repeat-single-channel, sequence, and repeat-sequence conversion modes
- ADC core and reference voltage can be powered down separately
- Interrupt vector register for fast decoding of 18 ADC interrupts
- 16 conversion-result storage registers

The block diagram of ADC12 is shown in Figure 23-1.

TLV Structure

The Tag-Length-Value (TLV) structure is used in selected MSP430x2xx devices to provide device-specific information in the device's flash memory SegmentA, such as calibration data. For the device-dependent implementation, see the device-specific data sheet.

Topic **Page**

24.1 TLV Introduction

The TLV structure stores device-specific data in SegmentA. The SegmentA content of an example device is shown in Table 24-1.

Table 24-1. Example SegmentA Structure

Word Address	Upper Byte	Lower Byte	Tag Address and Offset
0x10FE	CALBC1_1MHZ	CALDCO_1MHZ	0x10F6 + 0x0008
0x10FC	CALBC1_8MHZ	CALDCO_8MHZ	0x10F6 + 0x0006
0x10FA	CALBC1_12MHZ	CALDCO_12MHZ	0x10F6 + 0x0004
0x10F8	CALBC1_16MHZ	CALDCO_16MHZ	0x10F6 + 0x0002
0x10F6	0x08 (LENGTH)	TAG_DCO_30	0x10F6
0x10F4	0xFF	0xFF	
0x10F2	0xFF	0xFF	
0x10F0	0xFF	0xFF	
0x10EE	0xFF	0xFF	
0x10EC	0x08 (LENGTH)	TAG_EMPTY	0x10EC
0x10EA	CAL_ADC_25T85		0x10DA + 0x0010
0x10E8	CAL_ADC_25T30		0x10DA + 0x000E
0x10E6	CAL_ADC_25VREF_FACTOR		0x10DA + 0x000C
0x10E4	CAL_ADC_15T85		0x10DA + 0x000A
0x10E2	CAL_ADC_15T30		0x10DA + 0x0008
0x10E0	CAL_ADC_15VREF_FACTOR		0x10DA + 0x0006
0x10DE	CAL_ADC_OFFSET		0x10DA + 0x0004
0x10DC	CAL_ADC_GAIN_FACTOR		0x10DA + 0x0002
0x10DA	0x10 (LENGTH)	TAG_ADC12_1	0x10DA
0x10D8	0xFF	0xFF	
0x10D6	0xFF	0xFF	
0x10D4	0xFF	0xFF	
0x10D2	0xFF	0xFF	
0x10D0	0xFF	0xFF	
0x10CE	0xFF	0xFF	
0x10CC	0xFF	0xFF	
0x10CA	0xFF	0xFF	
0x10C8	0xFF	0xFF	
0x10C6	0xFF	0xFF	
0x10C4	0xFF	0xFF	
0x10C2	0x16 (LENGTH)	TAG_EMPTY	0x10C2
0x10C0	2s complement of bit-wise XOR		0x10C0

The first two bytes of SegmentA (0x10C0 and 0x10C1) hold the checksum of the remainder of the segment (addresses 0x10C2 to 0x10FF).

The first tag is located at address 0x10C2 and, in this example, is the TAG_EMPTY tag. The following byte (0x10C3) holds the length of the following structure. The length of this TAG_EMPTY structure is 0x16 and, therefore, the next tag, TAG_ADC12_1, is found at address 0x10DA. Again, the following byte holds the length of the TAG_ADC12_1 structure.

The TLV structure maps the entire address range 0x10C2 to 0x10FF of the SegmentA. A program routine looking for tags starting at the SegmentA address 0x10C2 can extract all information even if it is stored at a different (device-specific) absolute address.

DAC12

The DAC12 module is a 12-bit voltage-output digital-to-analog converter (DAC). This chapter describes the operation of the DAC12 module of the MSP430x2xx device family.

Topic **Page**

25.1 DAC12 Introduction

The DAC12 module is a 12-bit voltage-output DAC. The DAC12 can be configured in 8-bit or 12-bit mode and may be used in conjunction with the DMA controller. When multiple DAC12 modules are present, they may be grouped together for synchronous update operation.

Features of the DAC12 include:

- 12-bit monotonic output
- 8-bit or 12-bit voltage output resolution
- Programmable settling time vs power consumption
- Internal or external reference selection
- Straight binary or 2s compliment data format
- Self-calibration option for offset correction
- Synchronized update capability for multiple DAC12 modules

NOTE: Multiple DAC12 Modules

Some devices may integrate more than one DAC12 module. If more than one DAC12 is present on a device, the multiple DAC12 modules operate identically.

Throughout this chapter, nomenclature appears such as DAC12_xDAT or DAC12_xCTL to describe register names. When this occurs, the x is used to indicate which DAC12 module is being discussed. In cases where operation is identical, the register is simply referred to as DAC12_xCTL.

The block diagram of the DAC12 module is shown in Figure 25-1.

SD16_A

The SD16_A module is a single-converter 16-bit sigma-delta analog-to-digital conversion module with high impedance input buffer. This chapter describes the SD16_A. The SD16_A module is implemented in the MSP430x20x3 devices.

Topic **Page**

26.1 SD16_A Introduction

The SD16_A module consists of one sigma-delta analog-to-digital converter with a high-impedance input buffer and an internal voltage reference. It has up to eight fully differential multiplexed analog input pairs including a built-in temperature sensor and a divided supply voltage. The converter is based on a second-order oversampling sigma-delta modulator and digital decimation filter. The decimation filter is a comb type filter with selectable oversampling ratios of up to 1024. Additional filtering can be done in software.

The high impedance input buffer is not implemented in MSP430x20x3 devices.

Features of the SD16_A include:

- 16-bit sigma-delta architecture
- Up to eight multiplexed differential analog inputs per channel(The number of inputs is device dependent, see the device-specific data sheet.)
- Software selectable on-chip reference voltage generation (1.2 V)
- Software selectable internal or external reference
- Built-in temperature sensor
- Up to 1.1-MHz modulator input frequency
- High impedance input buffer(not implemented on all devices, see the device-specific data sheet)
- Selectable low-power conversion mode

The block diagram of the SD16_A module is shown in Figure 26-1.

SD24_A

The SD24_A module is a multichannel 24-bit sigma-delta analog-to-digital converter (ADC). This chapter describes the SD24_A of the MSP430x2xx family.

Topic **Page**

27.1 SD24_A Introduction

The SD24_A module consists of up to seven independent sigma-delta analog-to-digital converters, referred to as channels, and an internal voltage reference. Each channel has up to eight fully differential multiplexed analog input pairs including a built-in temperature sensor and a divided supply voltage. The converters are based on second-order oversampling sigma-delta modulators and digital decimation filters. The decimation filters are comb-type filters with selectable oversampling ratios of up to 1024. Additional filtering can be done in software.

The digital filter output of SD24_A can range from 15 bits to 30 bits, based on the oversampling ratio. The default oversampling ratio is 256, which results in 24-bit output from the digital filter. The 16 most significant bits of the filter are captured in the SD24_A conversion memory register and, by setting SD24LSBACC = 1, the 16 least significant bits of the filter output can be read (see Section 27.2.7 for details).

Features of the SD24_A include:

- Up to seven independent, simultaneously-sampling ADC channels (the number of channels is device dependent, see the device-specific data sheet)
- Up to eight multiplexed differential analog inputs per channel (the number of inputs is device dependent, see the device-specific data sheet)
- Software selectable on-chip reference voltage generation (1.2 V)
- Software selectable internal or external reference
- Built-in temperature sensor accessible by all channels
- Up to 1.1-MHz modulator input frequency
- High impedance input buffer (not implemented on all devices, see the device-specific data sheet)
- Selectable low-power conversion mode

The block diagram of the SD24_A module is shown in Figure 27-1.

Embedded Emulation Module (EEM)

This chapter describes the Embedded Emulation Module (EEM) that is implemented in all MSP430 flash devices.

Topic **Page**

28.1 EEM Introduction

Every MSP430 flash-based microcontroller implements an embedded emulation module (EEM). It is accessed and controlled through JTAG. Each implementation is device dependent and is described in section 1.3 *EEM Configurations* and the device-specific data sheet.

In general, the following features are available:

- Non-intrusive code execution with real-time breakpoint control
- Single step, step into and step over functionality
- Full support of all low-power modes
- Support for all system frequencies, for all clock sources
- Up to eight (device dependent) hardware triggers/breakpoints on memory address bus (MAB) or memory data bus (MDB)
- Up to two (device dependent) hardware triggers/breakpoints on CPU register write accesses
- MAB, MDB ,and CPU register access triggers can be combined to form up to eight (device dependent) complex triggers/breakpoints
- Trigger sequencing (device dependent)
- Storage of internal bus and control signals using an integrated trace buffer (device dependent)
- Clock control for timers, communication peripherals, and other modules on a global device level or on a per-module basis during an emulation stop

Figure 28-1 shows a simplified block diagram of the largest currently available 2xx EEM implementation.

For more details on how the features of the EEM can be used together with the IAR Embedded Workbench™ debugger see the application report *Advanced Debugging Using the Enhanced Emulation Module* (SLAA263) at www.msp430.com. Code Composer Essentials (CCE) and most other debuggers supporting MSP430 have the same or a similar feature set. For details see the user's guide of the applicable debugger.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265

A. Datasheet

c. Transceptor de radio frecuencia : CC250

CC2500
Low-Cost Low-Power 2.4 GHz RF Transceiver

Applications

- *2400-2483.5 MHz ISM/SRD band systems*
- *Consumer electronics*
- *Wireless game controllers*
- *Wireless audio*
- *Wireless keyboard and mouse*
- *RF enabled remote controls*

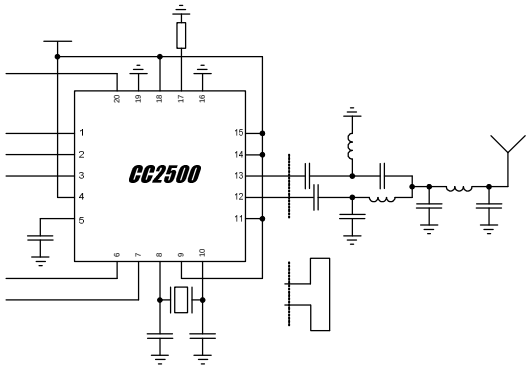
Product Description

The **CC2500** is a low-cost 2.4 GHz transceiver designed for very low-power wireless applications. The circuit is intended for the 2400-2483.5 MHz ISM (Industrial, Scientific and Medical) and SRD (Short Range Device) frequency band.

The RF transceiver is integrated with a highly configurable baseband modem. The modem supports various modulation formats and has a configurable data rate up to 500 kBaud.

CC2500 provides extensive hardware support for packet handling, data buffering, burst transmissions, clear channel assessment, link quality indication, and wake-on-radio.

The main operating parameters and the 64-byte transmit/receive FIFOs of **CC2500** can be controlled via an SPI interface. In a typical system, the **CC2500** will be used together with a microcontroller and a few additional passive components.



Key Features

RF Performance

- High sensitivity (–104 dBm at 2.4 kBaud, 1% packet error rate)
- Low current consumption (13.3 mA in RX, 250 kBaud, input well above sensitivity limit)
- Programmable output power up to +1 dBm
- Excellent receiver selectivity and blocking performance
- Programmable data rate from 1.2 to 500 kBaud
- Frequency range: 2400 – 2483.5 MHz

Analog Features

- OOK, 2-FSK, GFSK, and MSK supported
- Suitable for frequency hopping and multi-channel systems due to a fast settling frequency synthesizer with 90 us settling time
- Automatic Frequency Compensation (AFC) can be used to align the frequency synthesizer to the received centre frequency
- Integrated analog temperature sensor

Digital Features

- Flexible support for packet oriented systems: On-chip support for sync word detection, address check, flexible packet length, and automatic CRC handling
- Efficient SPI interface: All registers can be programmed with one “burst” transfer
- Digital RSSI output
- Programmable channel filter bandwidth
- Programmable Carrier Sense (CS) indicator

- Programmable Preamble Quality Indicator (PQI) for improved protection against false sync word detection in random noise
- Support for automatic Clear Channel Assessment (CCA) before transmitting (for listen-before-talk systems)
- Support for per-package Link Quality Indication (LQI)
- Optional automatic whitening and de-whitening of data

Low-Power Features

- 400 nA SLEEP mode current consumption
- Fast startup time: 240 us from SLEEP to RX or TX mode (measured on EM design)
- Wake-on-radio functionality for automatic low-power RX polling
- Separate 64-byte RX and TX data FIFOs (enables burst mode data transmission)

General

- Few external components: Complete on-chip frequency synthesizer, no external filters or RF switch needed
- Green package: RoHS compliant and no antimony or bromine
- Small size (QLP 4x4 mm package, 20 pins)
- Suited for systems compliant with EN 300 328 and EN 300 440 class 2 (Europe), FCC CFR47 Part 15 (US), and ARIB STD-T66 (Japan)
- Support for asynchronous and synchronous serial receive/transmit mode for backwards compatibility with existing radio communication protocols

A. Datasheet

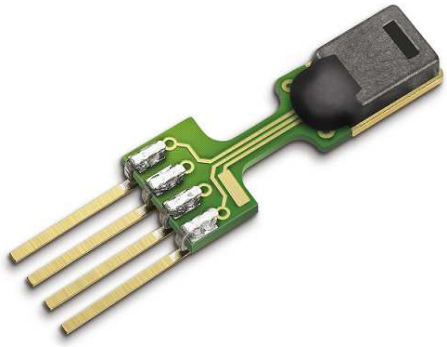
d. Sensor de humedad: SHT71

SENSIRION
THE SENSOR COMPANY

Datasheet SHT7x (SHT71, SHT75)

Humidity and Temperature Sensor IC

- Fully calibrated
- Digital output
- Low power consumption
- Excellent long term stability
- Pin type package – easy integration



Product Summary

SHT7x (including SHT71 and SHT75) is Sensirion's family of relative humidity and temperature sensors with pins. The sensors integrate sensor elements plus signal processing in compact format and provide a fully calibrated digital output. A unique capacitive sensor element is used for measuring relative humidity while temperature is measured by a band-gap sensor. The applied CMOSens® technology guarantees excellent reliability and long term stability. Both sensors are seamlessly coupled to a 14bit analog to digital converter and a serial interface circuit. This results in superior signal quality, a fast response time and insensitivity to external disturbances (EMC).

Each SHT7x is individually calibrated in a precision humidity chamber. The calibration coefficients are programmed into an OTP memory on the chip. These coefficients are used to internally calibrate the signals from the sensors. The 2-wire serial interface and internal voltage regulation allows for easy and fast system integration. The small size and low power consumption makes SHT7x the ultimate choice for even the most demanding applications.

SHT7x is supplied on FR4 with pins which allows for easy integration or replacement. The same sensor is also available as surface mountable packaging (SHT1x) or on flex print (SHTA1).

Dimensions

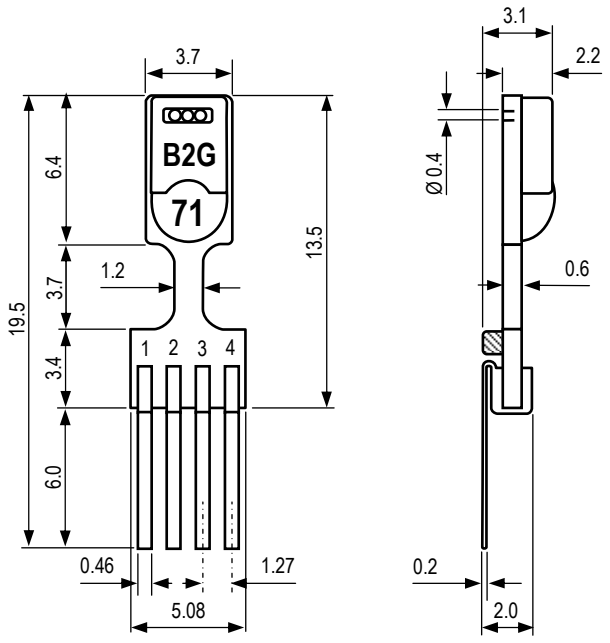


Figure 1: Drawing of SHT7x (applies to SHT71 and SHT75) sensor packaging, dimensions in mm (1mm = 0.039inch). Contact assignment: 1: SCK, 2: VDD, 3: GND, 4: DATA. Hatched item on backside of PCB is a 100nF capacitor – see Section 2.1 for more information.

Sensor Chip

SHT7x V4 – for which this datasheet applies – features a version 4 Silicon sensor chip. Besides a humidity and a temperature sensor the chip contains an amplifier, A/D converter, OTP memory and a digital interface. V4 sensors can be identified by the alpha-numeric traceability code on the sensor cap – see example "B2G" code on Figure 1.

Material Contents

While the sensor is made of a CMOS chip the sensor housing consists of an LCP cap with epoxy glob top on an FR4 substrate. Pins are made of a Cu/Be alloy coated with Ni and Au. The device is fully RoHS and WEEE compliant, thus it is free of Pb, Cd, Hg, Cr(6+), PBB and PBDE.

Evaluation Kits

For sensor trial measurements, for qualification of the sensor or even experimental application (data logging) of the sensor there is an evaluation kit *EK-H4* available including SHT71 (same sensor chip as SHT1x) and 4 sensor channels, hard and software to interface with a computer. For other evaluation kits please check www.sensirion.com/humidity.

Sensor Performance

Relative Humidity

Parameter	Condition	min	typ	max	Units
Resolution[1]		0.4	0.05	0.05	%RH
		8	12	12	bit
Accuracy[2] SHT71	typ		±3.0		%RH
	max	see Figure 2			
Accuracy[2] SHT75	typ		±1.8		%RH
	max	see Figure 2			
Repeatability			±0.1		%RH
Hysteresis			±1		%RH
Nonlinearity	raw data		±3		%RH
	linearized		<<1		%RH
Response time[3]	tau 63%		8		s
Operating Range		0		100	%RH
Long term drift[4]	normal		< 0.5		%RH/yr

Temperature

Parameter	Condition	min	typ	max	Units
Resolution[1]		0.04	0.01	0.01	°C
		12	14	14	bit
Accuracy[2] SHT71	typ		±0.4		°C
	max	see Figure 3			
Accuracy[2] SHT75	typ		±0.3		°C
	max	see Figure 3			
Repeatability			±0.1		°C
Operating Range		-40		123.8	°C
		-40		254.9	°F
Response Time [6]	tau 63%	5		30	s
Long term drift			< 0.04		°C/yr

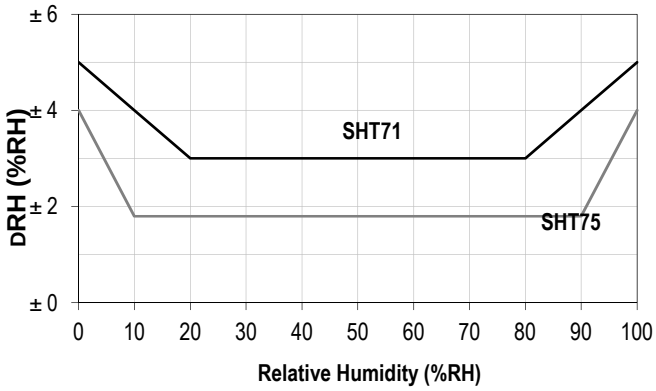


Figure 2: Maximal RH-tolerance at 25°C per sensor type.

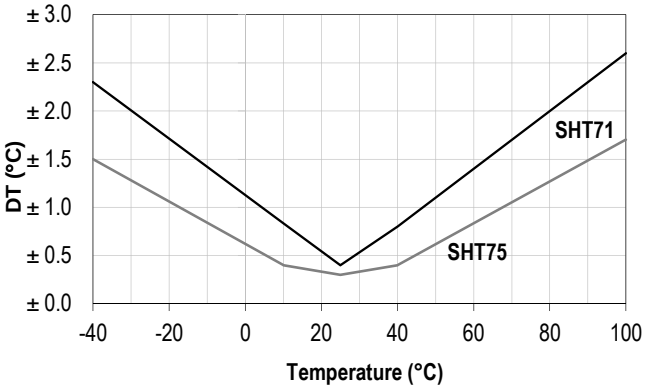


Figure 3: Maximal T-tolerance per sensor type.

Electrical and General Items

Parameter	Condition	min	typ	max	Units
Source Voltage		2.4	3.3	5.5	V
Power Consumption[5]	sleep		2	5	µW
	measuring		3		mW
	average		90		µW
Communication	digital 2-wire interface, see Communication				
Storage	10 – 50°C (0 – 80°C peak), 20 – 60%RH				

Packaging Information

Sensor Type	Packaging	Quantity	Order Number
SHT71	Tape Stripes	50	1-100092-04
SHT75	Tape Stripes	50	1-100071-04

This datasheet is subject to change and may be amended without prior notice.

[1] The default measurement resolution of is 14bit for temperature and 12bit for humidity. It can be reduced to 12/8bit by command to status register.

[2] Accuracies are tested at Outgoing Quality Control at 25°C (77°F) and 3.3V. Values exclude hysteresis and are only applicable to non-condensing environments.

[3] Time for reaching 63% of a step function, valid at 25°C and 1 m/s airflow.

[4] Value may be higher in environments with high contents of volatile organic compounds. See Section 1.3 of Users Guide.

[5] Values for VDD=3.3V at 25°C, average value at one 12bit measurement per second.

[6] Response time depends on heat capacity of and thermal resistance to sensor substrate.

Users Guide SHT7x

1 Application Information

1.1 Operating Conditions

Sensor works stable within recommended normal range – see Figure 4. Long term exposures to conditions outside normal range, especially at humidity >80%RH, may temporarily offset the RH signal (+3 %RH after 60h). After return to normal range it will slowly return towards calibration state by itself. See Section 1.4 "Reconditioning Procedure" to accelerate eliminating the offset. Prolonged exposure to extreme conditions may accelerate ageing.

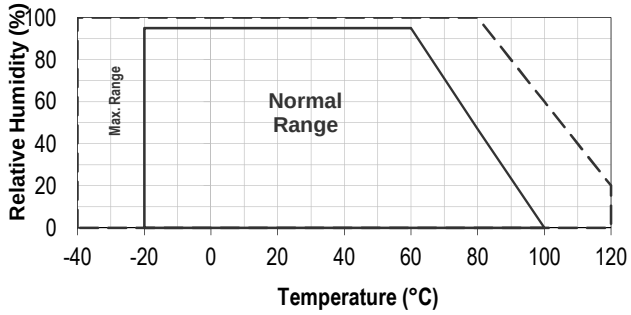


Figure 4: Operating Conditions

1.2 Sockets and Soldering

For maintain high accuracy specifications the sensor shall not be soldered. Sockets may be used such as "Preci-dip / Mill-Max R851-83-004-20-001" or similar.

Standard wave soldering ovens may be used at maximum 235°C for 20 seconds. For manual soldering contact time must be limited to 5 seconds at up to 350°C[7].

After wave soldering the devices should be stored at >75%RH for at least 12h to allow the polymer to re-hydrate. Alternatively the re-hydration process may be performed at ambient conditions (>40%RH) during more than 5 days.

In no case, neither after manual nor wave soldering, a board wash shall be applied. In case of application with exposure of the sensor to corrosive gases the soldering pads of pins and PCB shall be sealed to prevent loose contacts or short cuts.

1.3 Storage Conditions and Handling Instructions

It is of great importance to understand that a humidity sensor is not a normal electronic component and needs to be handled with care. Chemical vapors at high concentration in combination with long exposure times may offset the sensor reading.

For these reasons it is recommended to store the sensors in original packaging including the sealed ESD bag at following conditions: Temperature shall be in the range of 10°C – 50°C (0 – 80°C for limited time) and humidity at 20 – 60%RH (sensors that are not stored in ESD bags). For sensors that have been removed from the original packaging we recommend to store them in ESD bags made of metal-in PE-HD[8].

In manufacturing and transport the sensors shall be prevented of high concentration of chemical solvents and long exposure times. Out-gassing of glues, adhesive tapes and stickers or out-gassing packaging material such as bubble foils, foams, etc. shall be avoided. Manufacturing area shall be well ventilated.

For more detailed information please consult the document *"Handling Instructions"* or contact Sensirion.

1.4 Reconditioning Procedure

As stated above extreme conditions or exposure to solvent vapors may offset the sensor. The following reconditioning procedure may bring the sensor back to calibration state:

Baking: 100 – 105°C at < 5%RH for 10h
Re-Hydration: 20 – 30°C at ~ 75%RH for 12h [9].

1.5 Temperature Effects

Relative humidity reading strongly depends on temperature. Therefore, it is essential to keep humidity sensors at the same temperature as the air of which the relative humidity is to be measured. In case of testing or qualification the reference sensor and test sensor must show equal temperature to allow for comparing humidity readings.

The packaging of SHT7x is designed for minimal heat transfer from the pins to the sensor. Still, if the SHT7x shares a PCB with electronic components that produce heat it should be mounted in a way that prevents heat transfer or keeps it as low as possible.

Furthermore, there are self-heating effects in case the measurement frequency is too high. Please refer to Section 3.3 for detailed information.

[7] 235°C corresponds to 455°F, 350°C corresponds to 662°F

[8] For example, 3M antistatic bag, product "1910" with zipper .

[9] 75%RH can conveniently be generated with saturated NaCl solution. 100 – 105°C correspond to 212 – 221°F, 20 – 30°C correspond to 68 – 86°F

1.6 Light

The SHT7x is not light sensitive. Prolonged direct exposure to sunshine or strong UV radiation may age the housing.

1.7 Materials Used for Sealing / Mounting

Many materials absorb humidity and will act as a buffer increasing response times and hysteresis. Materials in the vicinity of the sensor must therefore be carefully chosen. Recommended materials are: Any metals, LCP, POM (Delrin), PTFE (Teflon), PE, PEEK, PP, PB, PPS, PSU, PVDF, PVF.

For sealing and gluing (use sparingly): Use high filled epoxy for electronic packaging (e.g. glob top, underfill), and Silicone. Out-gassing of these materials may also contaminate the SHT7x (see Section 1.3). Therefore try to add the sensor as a last manufacturing step to the assembly, store the assembly well ventilated after manufacturing or bake at 50°C for 24h to outgas contaminants before packing.

1.8 Wiring Considerations and Signal Integrity

SHT7x are often applied using wires. Carrying the SCK and DATA signal parallel and in close proximity more than 10cm may result in cross talk and loss of communication. This may be resolved by routing VDD and/or GND between the two data signals and/or using shielded cables. Furthermore, slowing down SCK frequency will possibly improve signal integrity.

Please see the Application Note “ESD, Latch-up and EMC” for more information.

1.9 ESD (Electrostatic Discharge)

ESD immunity is qualified according to MIL STD 883E, method 3015 (Human Body Model at ±2 kV).

Latch-up immunity is provided at a force current of ±100mA with T_{amb} = 80°C according to JEDEC78A. See Application Note “ESD, Latch-up and EMC” for more information.

2 Interface Specifications

Pin	Name	Comment
1	SCK	Serial Clock, input only
2	VDD	Source Voltage
3	GND	Ground
4	DATA	Serial Data, bidirectional

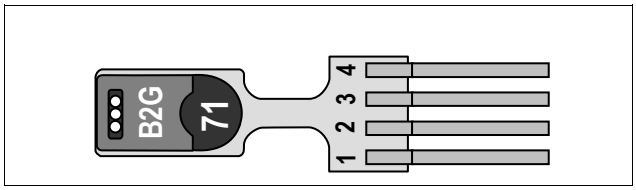


Table 1: SHT7x pin assignment.

2.1 Power Pins (VDD, GND)

The supply voltage of SHT7x must be in the range of 2.4 and 5.5V, recommended supply voltage is 3.3V. Decoupling of VDD and GND by a 100nF capacitor is integrated on the backside of the sensor packaging.

The serial interface of the SHT7x is optimized for sensor readout and effective power consumption. The sensor cannot be addressed by I²C protocol, however, the sensor can be connected to an I²C bus without interference with other devices connected to the bus. Microcontroller must switch between protocols.

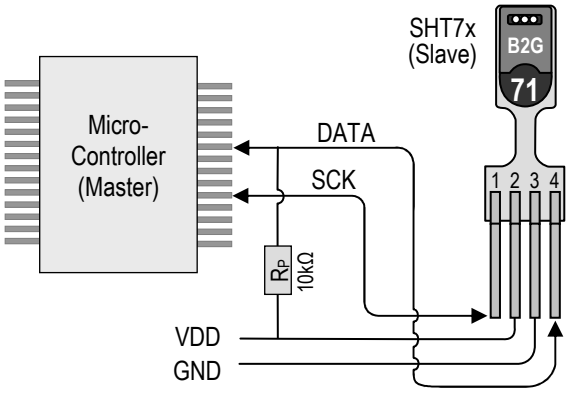


Figure 5: Typical application circuit, including pull up resistor R_P.

2.2 Serial clock input (SCK)

SCK is used to synchronize the communication between microcontroller and SHT7x. Since the interface consists of fully static logic there is no minimum SCK frequency.

2.3 Serial data (DATA)

The DATA tri-state pin is used to transfer data in and out of the sensor. For sending a command to the sensor, DATA is valid on the rising edge of the serial clock (SCK) and must remain stable while SCK is high. After the falling edge of SCK the DATA value may be changed. For safe communication DATA valid shall be extended T_{SU} and T_{HO} before the rising and after the falling edge of SCK, respectively – see Figure 6. For reading data from the sensor, DATA is valid T_V after SCK has gone low and remains valid until the next falling edge of SCK.

To avoid signal contention the microcontroller must only drive DATA low. An external pull-up resistor (e.g. 10 kΩ) is required to pull the signal high – it should be noted that pull-up resistors may be included in I/O circuits of

microcontrollers. See Table 2 for detailed I/O characteristic of the sensor.

2.4 Electrical Characteristics

The electrical characteristics such as power consumption, low and high level, input and output voltages depend on the supply voltage. Table 2 gives electrical characteristics of SHT7x with the assumption of 5V supply voltage if not stated otherwise. For proper communication with the sensor it is essential to make sure that signal design is strictly within the limits given in Table 3 and Figure 6. Absolute maximum ratings for VDD versus GND are +7V and -0.3V. Exposure to absolute maximum rating conditions for extended periods may affect the sensor reliability (e.g. hot carrier degradation, oxide breakdown).

Parameter	Conditions	min	typ	max	Units
Power supply DC[10]		2.4	3.3	5.5	V
Supply current	measuring		0.55	1	mA
	average[11]	2	28		µA
	sleep		0.3	1.5	µA
Low level output voltage	I_{OL} < 4 mA	0		250	mV
High level output voltage	R_P < 25 kΩ	90%		100%	VDD
Low level input voltage	Negative going	0%		20%	VDD
High level input voltage	Positive going	80%		100%	VDD
Input current on pads				1	µA
Output current	on			4	mA
	Tri-stated (off)		10	20	µA

Table 2: SHT7x DC characteristics. R_P stands for pull up resistor, while I_{OL} is low level output current.

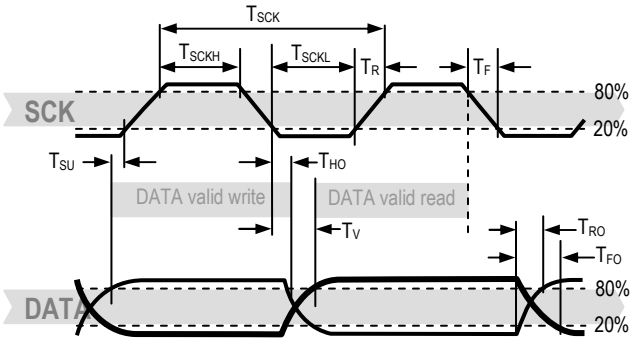


Figure 6: Timing Diagram, abbreviations are explained in Table 3. Bold DATA line is controlled by the sensor, plain DATA line is controlled by the micro-controller. Note that DATA valid read time is triggered by falling edge of anterior toggle.

[10] Recommended voltage supply for highest accuracy is 3.3V, due to sensor calibration.

[11] Minimum value with one measurement of 8 bit resolution without OTP reload per second, typical value with one measurement of 12bit resolution per second.

	Parameter	Conditions	min	typ	max	Units
F_{SCK}	SCK Frequency	VDD > 4.5V	0	0.1	5	MHz
		VDD < 4.5V	0	0.1	1	MHz
T_{SCKx}	SCK hi/low time		100			ns
T_R/T_F	SCK rise/fall time		1	200	*	ns
T_{FO}	DATA fall time	OL = 5pF	3.5	10	20	ns
		OL = 100pF	30	40	200	ns
T_{RO}	DATA rise time		**	**	**	ns
T_V	DATA valid time		200	250	***	ns
T_{SU}	DATA setup time		100	150	***	ns
T_{HO}	DATA hold time		10	15	****	ns

* $T_{R\_max} + T_{F\_max} = (F_{SCK})^{-1} - T_{SCKH} - T_{SCKL}$

** T_{R0} is determined by the $R_P * C_{bus}$ time-constant at DATA line

*** T_{V_max} and T_{SU_max} depend on external pull-up resistor (R_P) and total bus line capacitance (Cbus) at DATA line

**** $T_{H0\_max} < T_V - \max(T_{R0}, T_{F0})$

Table 3: SHT7x I/O signal characteristics, OL stands for Output Load, entities are displayed in Figure 6.

3 Communication with Sensor

3.1 Start up Sensor

As a first step the sensor is powered up to chosen supply voltage VDD. The slew rate during power up shall not fall below 1V/ms. After power-up the sensor needs 11ms to get to Sleep State. No commands must be sent before that time.

3.2 Sending a Command

To initiate a transmission, a Transmission Start sequence has to be issued. It consists of a lowering of the DATA line while SCK is high, followed by a low pulse on SCK and raising DATA again while SCK is still high – see Figure 7.

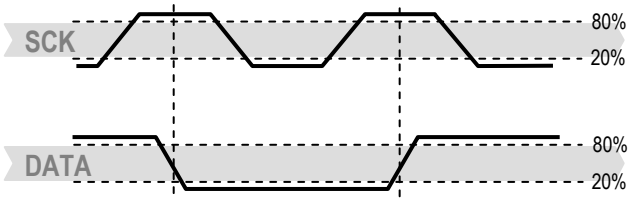


Figure 7: "Transmission Start" sequence

The subsequent command consists of three address bits (only '000' is supported) and five command bits. The SHT7x indicates the proper reception of a command by pulling the DATA pin low (ACK bit) after the falling edge of the 8th SCK clock. The DATA line is released (and goes high) after the falling edge of the 9th SCK clock.

Command	Code
Reserved	0000x
Measure Temperature	**00011**
Measure Relative Humidity	**00101**
Read Status Register	00111
Write Status Register	00110
Reserved	0101x-1110x
Soft reset, resets the interface, clears the status register to default values. Wait minimum 11 ms before next command	**11110**

Table 4: SHT7x list of commands

3.3 Measurement of RH and T

After issuing a measurement command ('00000101' for relative humidity, '00000011' for temperature) the controller has to wait for the measurement to complete. This takes a maximum of 20/80/320 ms for a 8/12/14bit measurement. The time varies with the speed of the internal oscillator and can be lower by up to 30%. To signal the completion of a measurement, the SHT7x pulls data line low and enters Idle Mode. The controller must wait for this Data Ready signal before restarting SCK to readout the data. Measurement data is stored until readout, therefore the controller can continue with other tasks and readout at its convenience.

Two bytes of measurement data and one byte of CRC checksum (optional) will then be transmitted. The micro controller must acknowledge each byte by pulling the DATA line low. All values are MSB first, right justified (e.g. the 5th SCK is MSB for a 12bit value, for a 8bit result the first byte is not used).

Communication terminates after the acknowledge bit of the CRC data. If CRC-8 checksum is not used the controller may terminate the communication after the measurement data LSB by keeping ACK high. The device automatically returns to Sleep Mode after measurement and communication are completed.

Important: To keep self heating below 0.1°C, SHT7x should not be active for more than 10% of the time – e.g. maximum one measurement per second at 12bit accuracy shall be made.

3.4 Connection reset sequence

If communication with the device is lost the following signal sequence will reset the serial interface: While leaving DATA high, toggle SCK nine or more times – see Figure 8. This must be followed by a Transmission Start sequence preceding the next command. This sequence resets the interface only. The status register preserves its content.

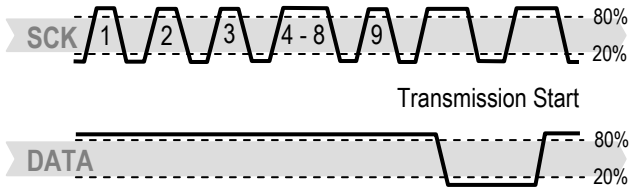


Figure 8: Connection Reset Sequence

3.5 CRC-8 Checksum calculation

The whole digital transmission is secured by an 8bit checksum. It ensures that any wrong data can be detected and eliminated. As described above this is an additional feature of which may be used or abandoned. Please consult Application Note "CRC Checksum" for information on how to calculate the CRC.

3.6 Status Register

Some of the advanced functions of the SHT7x such as selecting measurement resolution, end-of-battery notice, use of OTP reload or using the heater may be activated by sending a command to the status register. The following section gives a brief overview of these features.

After the command Status Register Read or Status Register Write – see Table 4 – the content of 8 bits of the status register may be read out or written. For the communication compare Figure 9 and Figure 10 – the assignation of the bits is displayed in Table 5.

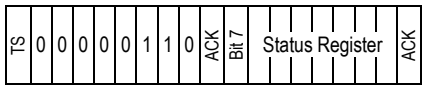


Figure 9: Status Register Write

Figure 10: Status Register Read

Examples of full communication cycle are displayed in Figure 11 and Figure 12.

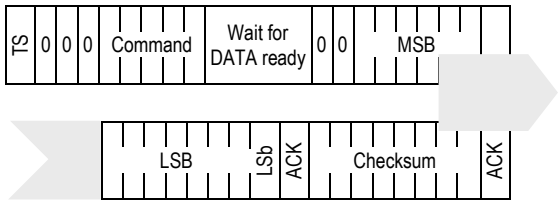


Figure 11: Overview of Measurement Sequence. TS = Transmission Start, MSB = Most Significant Byte, LSB = Last Significant Byte, LSb = Last Significant Bit.

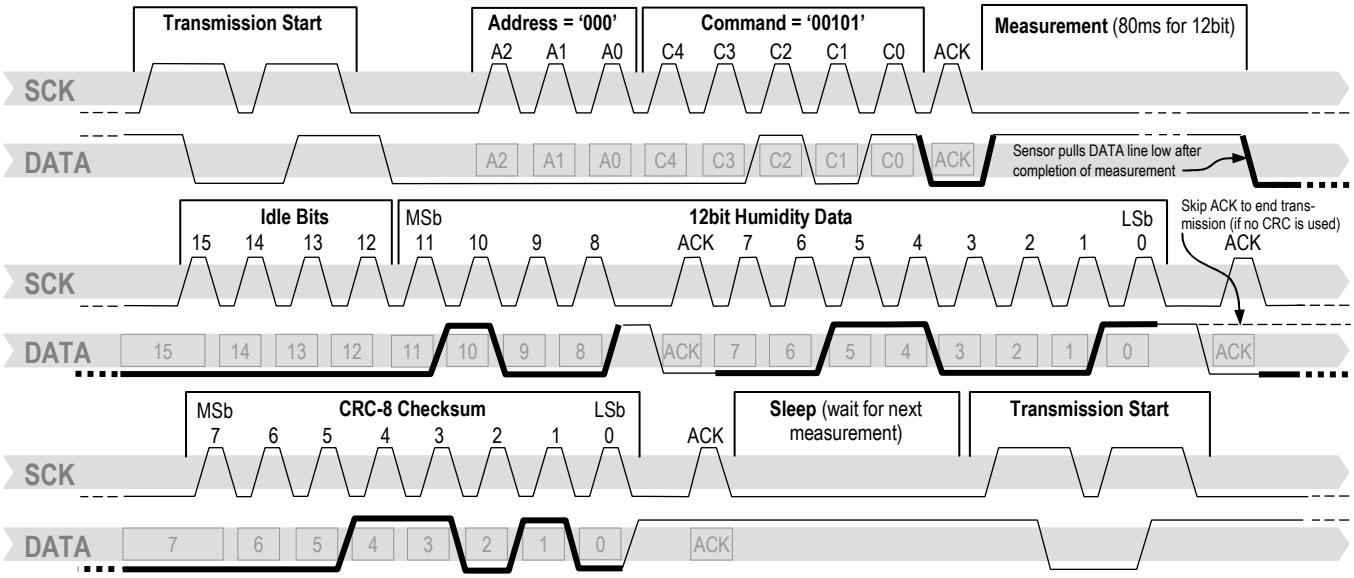


Figure 12: Example RH measurement sequence for value "0000'0100"0011'0001" = 1073 = 35.50%RH (without temperature compensation). DATA valid times are given and referenced in boxes on DATA line. Bold DATA lines are controlled by sensor while plain lines are controlled by the micro-controller.

Bit	Type	Description	Default	
7		reserved	0	
6	R	End of Battery (low voltage detection) '0' for VDD > 2.47 '1' for VDD < 2.47	X	No default value, bit is only updated after a measurement
5		reserved	0	
4		reserved	0	
3		For Testing only, do not use	0	
2	R/W	Heater	0	off
1	R/W	no reload from OTP	0	reload
0	R/W	'1' = 8bit RH / 12bit Temp. resolution '0' = 12bit RH / 14bit Temp. resolution	0	12bit RH 14bit Temp.

Table 5: Status Register Bits

Measurement resolution: The default measurement resolution of 14bit (temperature) and 12bit (humidity) can be reduced to 12 and 8bit. This is especially useful in high speed or extreme low power applications.

End of Battery function detects and notifies VDD voltages below 2.47 V. Accuracy is ±0.05 V.

Heater: An on chip heating element can be addressed by writing a command into status register. The heater may increase the temperature of the sensor by 5 – 10°C[12] beyond ambient temperature. The heater draws roughly 8mA @ 5V supply voltage.

For example the heater can be helpful for functionality analysis: Humidity and temperature readings before and after applying the heater are compared. Temperature shall increase while relative humidity decreases at the same time. Dew point shall remain the same.

Please note: The temperature reading will display the temperature of the heated sensor element and not ambient temperature. Furthermore, the sensor is not qualified for continuous application of the heater.

OTP reload: With this operation the calibration data is uploaded to the register before each measurement. This may be deactivated for reducing measurement time by about 10ms.

4 Conversion of Signal Output

4.1 Relative Humidity

For compensating non-linearity of the humidity sensor – see Figure 13 – and for obtaining the full accuracy of the sensor it is recommended to convert the humidity readout (SO_{RH}) with the following formula with coefficients given in Table 6:

$$RH_{linear} = c_1 + c_2 \cdot SO_{RH} + c_3 \cdot SO_{RH}^2 \ (\%RH)$$

SO_{RH}	c_1	c_2	c_3
12 bit	-2.0468	0.0367	-1.5955E-6
8 bit	-2.0468	0.5872	-4.0845E-4

Table 6: Humidity conversion coefficients

[12] Corresponds to 9 – 18°F

Values higher than 99%RH indicate fully saturated air and must be processed and displayed as 100%RH[13]. Please note that the humidity sensor has no significant voltage dependency.

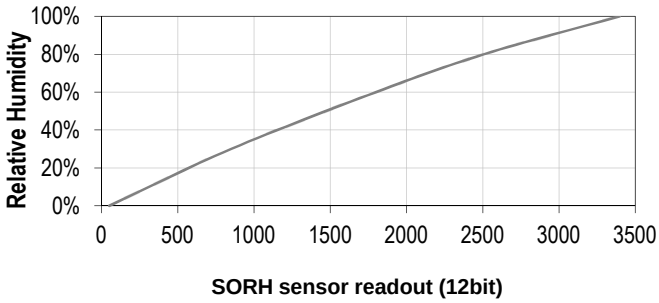


Figure 13: Conversion from SO_{RH} to relative humidity

4.2 Temperature compensation of Humidity Signal

For temperatures significantly different from 25°C (~77°F) the humidity signal requires temperature compensation. The temperature correction corresponds roughly to 0.12%RH/°C @ 50%RH. Coefficients for the temperature compensation are given in Table 8.

$$RH_{true} = (T_{°C} - 25) \cdot (t_1 + t_2 \cdot SO_{RH}) + RH_{linear}$$

SO_{RH}	t_1	t_2
12 bit	0.01	0.00008
8 bit	0.01	0.00128

Table 7: Temperature compensation coefficients

4.3 Temperature

The band-gap PTAT (Proportional To Absolute Temperature) temperature sensor is very linear by design. Use the following formula to convert digital readout (SO_T) to temperature value, with coefficients given in Table 9:

$$T = d_1 + d_2 \cdot SO_T$$

VDD	d_1 (°C)	d_1 (°F)
5V	-40.1	-40.2
4V	-39.8	-39.6
3.5V	-39.7	-39.5
3V	-39.6	-39.3
2.5V	-39.4	-38.9

SO_T	d_2 (°C)	d_2 (°F)
14bit	0.01	0.018
12bit	0.04	0.072

Table 8: Temperature conversion coefficients.

4.4 Dew Point

SHT7x is not measuring dew point directly, however dew point can be derived from humidity and temperature readings. Since humidity and temperature are both measured on the same monolithic chip, the SHT7x allows superb dew point measurements.

For dew point (T_d) calculations there are various formulas to be applied, most of them quite complicated. For the temperature range of -40 – 50°C the following approximation provides good accuracy with parameters given in Table 10:

$$T_d(RH,T) = T_n \cdot \frac{\ln\left(\frac{RH}{100\%}\right) + \frac{m \cdot T}{T_n + T}}{m - \ln\left(\frac{RH}{100\%}\right) - \frac{m \cdot T}{T_n + T}}$$

Temperature Range	Tn (°C)	m
Above water, 0 – 50°C	243.12	17.62
Above ice, -40 – 0°C	272.62	22.46

Table 9: Parameters for dew point (T_d) calculation.

Please note that "ln(…)" denotes the natural logarithm. For RH and T the linearized and compensated values for relative humidity and temperature shall be applied.

For more information on dew point calculation see Application Note "Introduction to Humidity".

5 Environmental Stability

If sensors are qualified for assemblies or devices, please make sure that they experience same conditions as the reference sensor. It should be taken into account that response times in assemblies may be longer, hence enough dwell time for the measurement shall be granted. For detailed information please consult Application Note "Testing Guide".

SHT7x have been tested according to the test conditions given in Table 11. Sensor performance under other test conditions cannot be guaranteed and is not part of the sensor specifications. Especially, no guarantee can be given for sensor performance in the field or for customer's specific application.

Please contact Sensirion for detailed information.

[13] If wetted excessively (strong condensation of water on sensor surface), sensor output signal can drop below 100%RH (even below 0%RH in some cases), but the sensor will recover completely when water droplets evaporate. The sensor is not damaged by water immersion or condensation.

Environment	Standard	Results [14]
HTOL	125°C, 1000 h	Within specifications
TC	-40°C - 125°C, 500 cycles Acc. JESD22-A104-C	Within specifications
THU	85°C / 85%RH, 1000h	Within specifications
ESD immunity	MIL STD 883E, method 3015 (Human Body Model at ±2kV)	Qualified
Latch-up	force current of ±100mA with T_{amb} = 80°C, acc. JEDEC 17	Qualified

Table 10: Qualification tests: HTSL = High Temperature Storage Lifetime, TC = Temperature Cycles, UHST = Unbiased Highly accelerated temperature and humidity Test, THU = Temperature humidity unbiased

6 Packaging

6.1 Packaging type

The device is supplied in a single-in-line pin type package. The sensor housing consists of a Liquid Crystal Polymer (LCP) cap with epoxy glob top on a standard 0.6 mm FR4 substrate. The sensor head is connected to the pins, by a small bridge to minimize heat conduction and response times. The pins are made of Cu/Be alloy coated with 1.3µm Ni and 0.5µm Au, which are soldered to the FR4 substrate by lead-free solder paste. The gold plated back side of the sensor head is connected to the GND pin. A 100nF capacitor is mounted on the back side between VDD and GND. The device is fully RoHS and WEEE compliant – thus it is free of of Pb, Cd, Hg, Cr(6+), PBB and PBDE.

Size including pins is 19.5 x 5.08 x 3.1mm. Total weight: 168 mg, weight of sensor head: 73 mg.

All pins are Au plated to avoid corrosion. They can be soldered or mate with most 1.27 mm (0.05'') sockets, for example: Preci-dip / Mill-Max R851-83-004-20-001 or similar.

6.2 Traceability Information

All SHT7x are marked with an alphanumeric, three digit code on the chip cap – see "B2G" on Figure 1. The lot numbers allow full traceability through production, calibration and testing. No information can be derived from the code directly, respective data is stored at Sensirion.

Labels on the reels are displayed in Figure 14 and Figure 15, they both give traceability information.

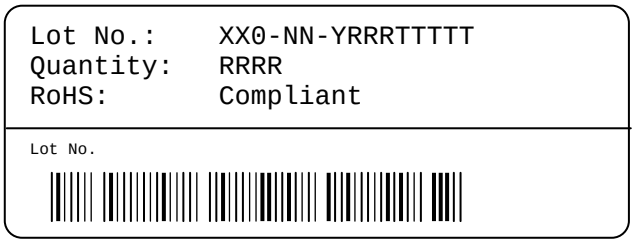


Figure 14: First label on reel: XX = Sensor Type (71 for SHT71), NN = Chip Version (04 for V4), Y = last digit of year, RRR = number of sensors on reel divided by 10 (200 for 2000 units), TTTTT = Traceability Code.

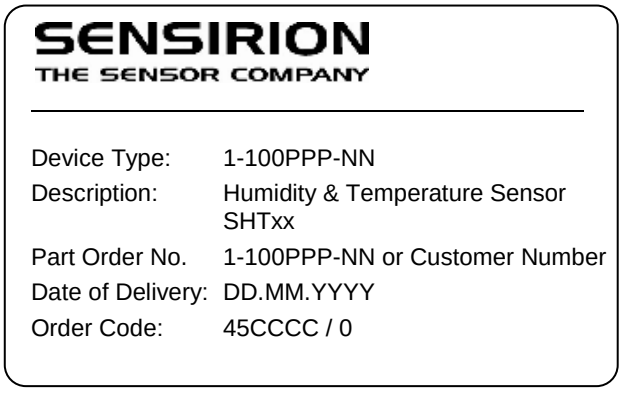


Figure 15: Second label on reel: For Device Type and Part Order Number please refer to Table 12, Delivery Date (also Date Code) is date of packaging of sensors (DD = day, MM = month, YYYY = year), CCCC = number of order.

6.3 Shipping Package

SHT7x are shipped in 32mm tape at 50pcs each – for details see Figure 16 and Table 12. Reels are individually labeled with barcode and human readable labels, see section 6.2.

Sensor Type	Packaging	Quantity	Order Number
SHT71	Tape Stripes	50	1-100092-04
SHT75	Tape Stripes	50	1-100071-04

Table 11: Packaging types per sensor type.

Dimensions of packaging tape are given in Figure 16. All tapes have a 7 pockets empty leader tape (first pockets of the tape) and a 7 pockets empty trailer tape (last pockets of the tape).

[14] According to accuracy and long term drift specification given on Page 2.

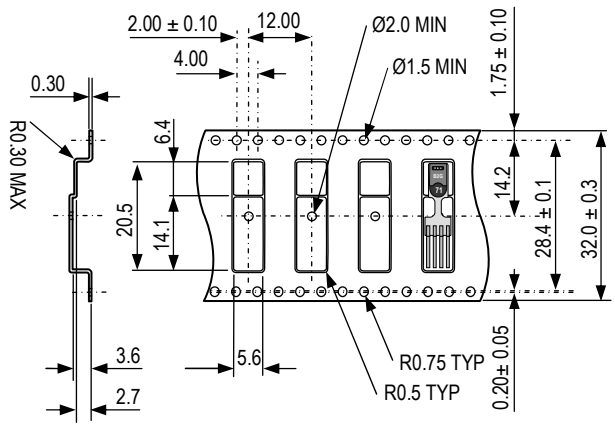


Figure 16: Tape configuration and unit orientation within tape, dimensions in mm (1mm = 0.039inch). Leader tape is to the right of the figure, trailer tape to the left.

Revision History

Date	Version	Page(s)	Changes
March 2007	3.0	1 – 10	Data sheet valid for SHTxx-V4 and SHTxx-V3
July 2008	4.0	1 – 10	New release, rework of datasheet
April 2009	4.2	2, 7	Amended foot note 2, communication diagram changed (Figure 12)
May 2010	4.3	1 – 11	Errors eliminated, information added – for details please ask for change protocol.
December 2011	5	1, 7-9	References to V3 sensors eliminated.

Important Notices

Warning, Personal Injury

Do not use this product as safety or emergency stop devices or in any other application where failure of the product could result in personal injury. Do not use this product for applications other than its intended and authorized use. Before installing, handling, using or servicing this product, please consult the data sheet and application notes. Failure to comply with these instructions could result in death or serious injury.

If the Buyer shall purchase or use SENSIRION products for any unintended or unauthorized application, Buyer shall defend, indemnify and hold harmless SENSIRION and its officers, employees, subsidiaries, affiliates and distributors against all claims, costs, damages and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if SENSIRION shall be allegedly negligent with respect to the design or the manufacture of the product.

ESD Precautions

The inherent design of this component causes it to be sensitive to electrostatic discharge (ESD). To prevent ESD-induced damage and/or degradation, take customary and statutory ESD precautions when handling this product.
See application note "ESD, Latchup and EMC" for more information.

Warranty

SENSIRION warrants solely to the original purchaser of this product for a period of 12 months (one year) from the date of delivery that this product shall be of the quality, material and workmanship defined in SENSIRION's published specifications of the product. Within such period, if proven to be defective, SENSIRION shall repair and/or replace this product, in SENSIRION's discretion, free of charge to the Buyer, provided that:

- notice in writing describing the defects shall be given to SENSIRION within fourteen (14) days after their appearance;
- such defects shall be found, to SENSIRION's reasonable satisfaction, to have arisen from SENSIRION's faulty design, material, or workmanship;
- the defective product shall be returned to SENSIRION's factory at the Buyer's expense; and
- the warranty period for any repaired or replaced product shall be limited to the unexpired portion of the original period.

This warranty does not apply to any equipment which has not been installed and used within the specifications recommended by SENSIRION for the intended and proper use of the equipment. EXCEPT FOR THE WARRANTIES EXPRESSLY SET FORTH HEREIN, SENSIRION MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THE PRODUCT. ANY AND ALL WARRANTIES, INCLUDING WITHOUT LIMITATION, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, ARE EXPRESSLY EXCLUDED AND DECLINED.
SENSIRION is only liable for defects of this product arising under the conditions of operation provided for in the data sheet and proper use of the goods. SENSIRION explicitly disclaims all warranties, express or implied, for any period during which the goods are operated or stored not in accordance with the technical specifications.
SENSIRION does not assume any liability arising out of any application or use of any product or circuit and specifically disclaims any and all liability, including without limitation consequential or incidental damages. All operating parameters, including without limitation recommended parameters, must be validated for each customer's applications by customer's technical experts. Recommended parameters can and do vary in different applications.
SENSIRION reserves the right, without further notice, (i) to change the product specifications and/or the information in this document and (ii) to improve reliability, functions and design of this product.

Headquarters and Subsidiaries

SENSIRION AG
Laubisruetistr. 50
CH-8712 Staefa ZH
Switzerland

phone: +41 44 306 40 00
fax: +41 44 306 40 30
info@sensirion.com
www.sensirion.com

Sensirion AG (Germany)
phone: +41 44 927 11 66
info@sensirion.com
www.sensirion.com

Sensirion Inc., USA
phone: +1 805 409 4900
info_us@sensirion.com
www.sensirion.com

Sensirion Japan Co. Ltd.
phone: +81 3 3444 4940
info@sensirion.co.jp
www.sensirion.co.jp

Sensirion Korea Co. Ltd.
phone: +82 31 345 0031 3
info@sensirion.co.kr
www.sensirion.co.kr

Sensirion China Co. Ltd.
phone: +86 755 8252 1501
info@sensirion.com.cn
www.sensirion.com.cn

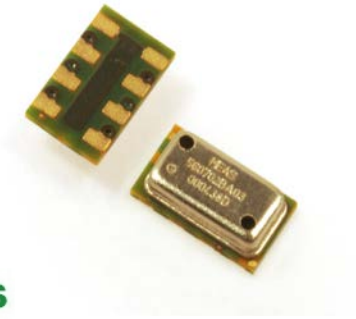
To find your local representative, please visit www.sensirion.com/contact

A. Datasheet

e. Sensor de Presión:

MS5607-02BA

MS5607-02BA03 Barometric Pressure Sensor, with stainless steel cap



- **High resolution module, 20cm**
- **Fast conversion down to 1 ms**
- **Low power, 1 µA (standby < 0.15 µA)**
- **QFN package 5.0 x 3.0 x 1.0 mm^3**
- **Supply voltage 1.8 to 3.6 V**
- **Integrated digital pressure sensor (24 bit ΔΣ ADC)**
- **Operating range: 10 to 1200 mbar, -40 to +85 °C**
- **I^{2}C and SPI interface up to 20 MHz**
- **No external components (Internal oscillator)**
- **Excellent long term stability**

DESCRIPTION

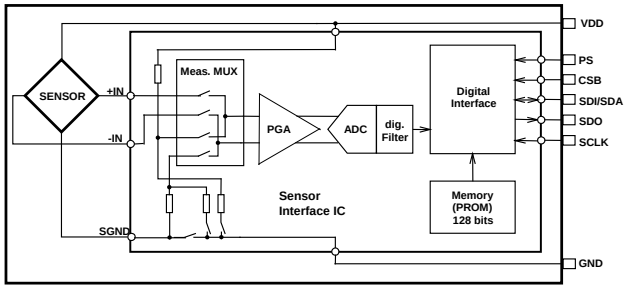
The MS5607-02BA is a new generation of high resolution altimeter sensors from MEAS Switzerland with SPI and I^{2}C bus interface. This barometric pressure sensor is optimized for altimeters and variometers with an altitude resolution of 20 cm. The sensor module includes a high linearity pressure sensor and an ultra low power 24 bit ΔΣ ADC with internal factory calibrated coefficients. It provides a precise digital 24 bit pressure and temperature value and different operation modes that allow the user to optimize for conversion speed and current consumption. A high resolution temperature output allows the implementation of an altimeter/thermometer function without any additional sensor. The MS5607-02BA can be interfaced to virtually any microcontroller. The communication protocol is simple, without the need of programming internal registers in the device. Small dimensions of only 5.0 mm x 3.0 mm and a height of only 1.0 mm allow for integration in mobile devices. This new sensor module generation is based on leading MEMS technology and latest benefits from MEAS Switzerland proven experience and know-how in high volume manufacturing of altimeter modules, which have been widely used for over a decade. The sensing principle employed leads to very low hysteresis and high stability of both pressure and temperature signal.

FEATURES

FIELD OF APPLICATION

Mobile altimeter / barometer systems
Bike computers
Variometers
Dataloggers
Mobile phones / GPS

FUNCTIONAL BLOCK DIAGRAM



TECHNICAL DATA

Sensor Performances (V_{DD} = 3 V)				
Pressure	**Min**	**Typ**	**Max**	**Unit**
Range	10		1200	mbar
ADC		24		bit
Resolution (1)		0.13 / 0.084 / 0.054 / 0.036 / 0.024		mbar
Accuracy 25°C, 750 mbar	-1.5		+1.5	mbar
Error band, -20°C to + 85°C 300 to 1100 mbar (2)	-2.5		+2.5	mbar
Response time (1)		0.5 / 1.1 / 2.1 / 4.1 / 8.22		ms
Long term stability		±1		mbar/yr
Temperature	**Min**	**Typ**	**Max**	**Unit**
Range	-40		+85	°C
Resolution		<0.01		°C
Accuracy	-0.8		+0.8	°C

Notes: (1) Oversampling Ratio: 256 / 512 / 1024 / 2048 / 4096
(2) With autozero at one pressure point

PERFORMANCE SPECIFICATIONS

ABSOLUTE MAXIMUM RATINGS

Parameter	Symbol	Conditions	Min.	Typ.	Max	Unit
Supply voltage	V_{DD}		-0.3		+4.0	V
Storage temperature	T_S		-40		+125	°C
Overpressure	P_{max}				6	bar
Maximum Soldering Temperature	T_{max}	40 sec max			250	°C
ESD rating		Human Body Model	-4		+4	kV
Latch up		JEDEC standard No 78	-100		+100	mA

ELECTRICAL CHARACTERISTICS

Parameter	Symbol	Conditions	Min.	Typ.	Max	Unit
Operating Supply voltage	V_{DD}		1.8	3.0	3.6	V
Operating Temperature	T		-40	+25	+85	°C
Supply current (1 sample per sec.)	I_{DD}	OSR 4096		12.5		µA
		2048		6.3		
		1024		3.2		
		512		1.7		
		256		0.9		
Peak supply current		during conversion		1.4		mA
Standby supply current		at 25°C		0.02	0.14	µA
VDD Capacitor		From VDD to GND	100			nF

ANALOG DIGITAL CONVERTER (ADC)

Parameter	Symbol	Conditions	Min.	Typ.	Max	Unit
Output Word				24		bit
Conversion time	t_c	OSR 4096	7.40	8.22	9.04	ms
		2048	3.72	4.13	4.54	
		1024	1.88	2.08	2.28	
		512	0.95	1.06	1.17	
		256	0.48	0.54	0.60	

PERFORMANCE SPECIFICATIONS (CONTINUED)

PRESSURE OUTPUT CHARACTERISTICS (V_{DD} = 3 V, T = 25°C UNLESS OTHERWISE NOTED)

Parameter	Conditions		Min.	Typ.	Max	Unit
Operating Pressure Range	P_{range}	Full Accuracy	300		1100	mbar
Extended Pressure Range	P_{ext}	Linear Range of ADC	10		1200	mbar
Total Error band, no autozero	at 25°C, 700..1100 mbar at 0..50°C, 300..1100 mbar at -20..85°C, 300..1100 mbar		-1.5 -2.0 -3.5		+1.5 +2.0 +3.5	mbar
Total Error band, autozero at one pressure point	at 25°C, 700..1100 mbar at 0..50°C, 300..1100 mbar at -20..85°C, 300..1100 mbar		-0.5 -1.0 -2.5		+0.5 +1.0 +2.5	mbar
Maximum error with supply voltage	V_{DD} = 1.8 V … 3.6 V		-2.5		+2.5	mbar
Resolution RMS	OSR 4096 2048 1024 512 256			0.024 0.036 0.054 0.084 0.130		mbar
Long-term stability				±1		mbar/yr
Reflow soldering impact	IPC/JEDEC J-STD-020C (See application note AN808)			+0.4		mbar
Recovering time after reflow (1)				7		days

(1) Time to recovering at least 66% of the reflow impact

TEMPERATURE OUTPUT CHARACTERISTICS (V_{DD} = 3 V, T = 25°C UNLESS OTHERWISE NOTED)

Parameter	Conditions	Min.	Typ.	Max	Unit
Absolute Accuracy	at 25°C -20..85°C -40..85°C	-0.8 -2.0 -4.0		+0.8 +2.0 +4.0	°C
Maximum error with supply voltage	V_{DD} = 1.8 V … 3.6 V	-0.5		+0.5	°C
Resolution RMS	OSR 4096 2048 1024 512 256		0.002 0.003 0.005 0.008 0.012		°C

PERFORMANCE SPECIFICATIONS (CONTINUED)

DIGITAL INPUTS (CSB, I^2C, DIN, SCLK)

Parameter	Symbol	Conditions	Min.	Typ.	Max	Unit
Serial data clock	SCLK	SPI protocol			20	MHz
		I^2C protocol			400	KHz
Input high voltage	V_{IH}	Pins CSB	80% V_{DD}		100% V_{DD}	V
Input low voltage	V_{IL}		0% V_{DD}		20% V_{DD}	V
Input leakage current	$I_{leak25°C}$ $I_{leak85°C}$	at 25°C			0.15	µA
Input capacitance	C_{IN}				6	pF

PRESSURE OUTPUTS (I^2C, DOUT)

Parameter	Symbol	Conditions	Min.	Typ.	Max	Unit
Output high voltage	V_{OH}	I_{source} = 1.0 mA	80% V_{DD}		100% V_{DD}	V
Output low voltage	V_{OL}	I_{sink} = 1.0 mA	0% V_{DD}		20% V_{DD}	V
Load capacitance	C_{LOAD}				16	pF

FUNCTIONAL DESCRIPTION

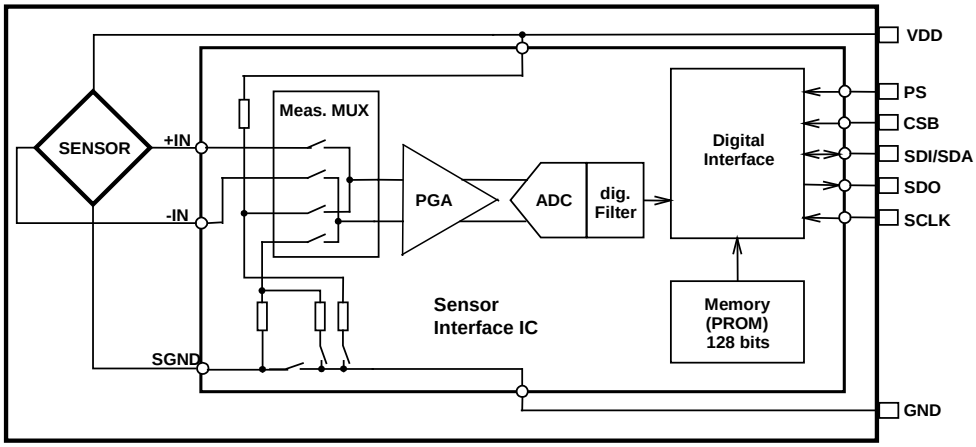


Figure 1: Block diagram of MS5607-02BA

GENERAL

The MS5607-02BA consists of a piezo-resistive sensor and a sensor interface IC. The main function of the MS5607-02BA is to convert the uncompensated analogue output voltage from the piezo-resistive pressure sensor to a 24-bit digital value, as well as providing a 24-bit digital value for the temperature of the sensor.

FACTORY CALIBRATION

Every module is individually factory calibrated at two temperatures and two pressures. As a result, 6 coefficients necessary to compensate for process variations and temperature variations are calculated and stored in the 128-bit PROM of each module. These bits (partitioned into 6 coefficients) must be read by the microcontroller software and used in the program converting D1 and D2 into compensated pressure and temperature values.

SERIAL INTERFACE

The MS5607-02BA has built in two types of serial interfaces: SPI and I^2C. Pulling the Protocol Select pin PS to low selects the SPI protocol, pulling PS to high activates the I^2C bus protocol.

Pin PS	Mode	Pins used
High	I^2C	SDA
Low	SPI	SDI, SDO, CSB

SPI MODE

The external microcontroller clocks in the data through the input SCLK (Serial CLocK) and SDI (Serial Data In). In the SPI mode module can accept both mode 0 and mode 3 for the clock polarity and phase. The sensor responds on the output SDO (Serial Data Out). The pin CSB (Chip Select) is used to enable/disable the interface, so that other devices can talk on the same SPI bus. The CSB pin can be pulled high after the command is sent or after the end of the command execution (for example end of conversion). The best noise performance from the module is obtained when the SPI bus is idle and without communication to other devices during the ADC conversion.

I²C MODE

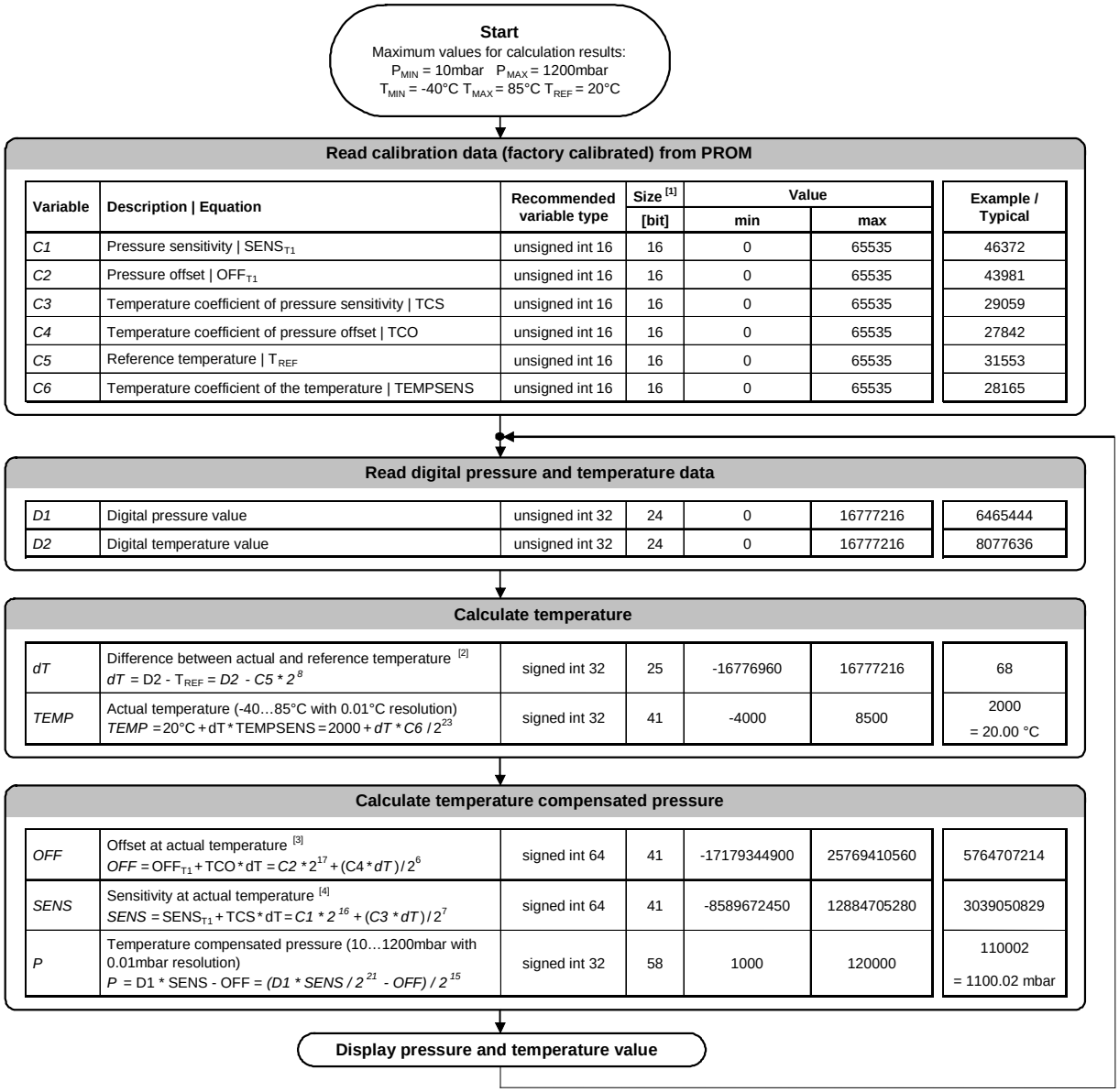
The external microcontroller clocks in the data through the input SCLK (Serial CLocK) and SDA (Serial DAta). The sensor responds on the same pin SDA which is bidirectional for the I²C bus interface. So this interface type uses only 2 signal lines and does not require a chip select, which can be favourable to reduce board space. In I²C-Mode the complement of the pin CSB (Chip Select) represents the LSB of the I²C address. It is possible to use two sensors with two different addresses on the I²C bus. The pin CSB shall be connected to VDD or GND (do not leave unconnected!).

COMMANDS

The MS5607-02BA has only five basic commands:

1. Reset
2. Read PROM (128 bit of calibration words)
3. D1 conversion
4. D2 conversion
5. Read ADC result (24 bit pressure / temperature)

PRESSURE AND TEMPERATURE CALCULATION



Read calibration data (factory calibrated) from PROM

Variable	Description | Equation	Recommended variable type	Size [1] [bit]	Value min	Value max	Example / Typical
C1	Pressure sensitivity | $SENS_{T1}$	unsigned int 16	16	0	65535	46372
C2	Pressure offset | OFF_{T1}	unsigned int 16	16	0	65535	43981
C3	Temperature coefficient of pressure sensitivity | TCS	unsigned int 16	16	0	65535	29059
C4	Temperature coefficient of pressure offset | TCO	unsigned int 16	16	0	65535	27842
C5	Reference temperature | T_{REF}	unsigned int 16	16	0	65535	31553
C6	Temperature coefficient of the temperature | TEMPSENS	unsigned int 16	16	0	65535	28165

Read digital pressure and temperature data

D1	Digital pressure value	unsigned int 32	24	0	16777216	6465444
D2	Digital temperature value	unsigned int 32	24	0	16777216	8077636

Calculate temperature

dT	Difference between actual and reference temperature [2] $dT = D2 - T_{REF} = D2 - C5 * 2^8$	signed int 32	25	-16776960	16777216	68
TEMP	Actual temperature (-40…85°C with 0.01°C resolution) $TEMP = 20°C + dT * TEMPSENS = 2000 + dT * C6 / 2^{23}$	signed int 32	41	-4000	8500	2000 = 20.00 °C

Calculate temperature compensated pressure

OFF	Offset at actual temperature [3] $OFF = OFF_{T1} + TCO * dT = C2 * 2^{17} + (C4 * dT) / 2^6$	signed int 64	41	-17179344900	25769410560	5764707214
SENS	Sensitivity at actual temperature [4] $SENS = SENS_{T1} + TCS * dT = C1 * 2^{16} + (C3 * dT) / 2^7$	signed int 64	41	-8589672450	12884705280	3039050829
P	Temperature compensated pressure (10…1200mbar with 0.01mbar resolution) $P = D1 * SENS - OFF = (D1 * SENS / 2^{21} - OFF) / 2^{15}$	signed int 32	58	1000	120000	110002 = 1100.02 mbar

Notes
[1] Maximal size of intermediate result during evaluation of variable
[2] min and max have to be defined
[3] min and max have to be defined
[4] min and max have to be defined

Figure 2: Flow chart for pressure and temperature reading and software compensation.

SECOND ORDER TEMPERATURE COMPENSATION

In order to obtain best accuracy over temperature range, particularly in low temperature, it is recommended to compensate the non-linearity over the temperature. This can be achieved by correcting the calculated temperature, offset and sensitivity by a second order correction factor and will be recalculated with the standard calculation. The second-order factors are calculated as follows:

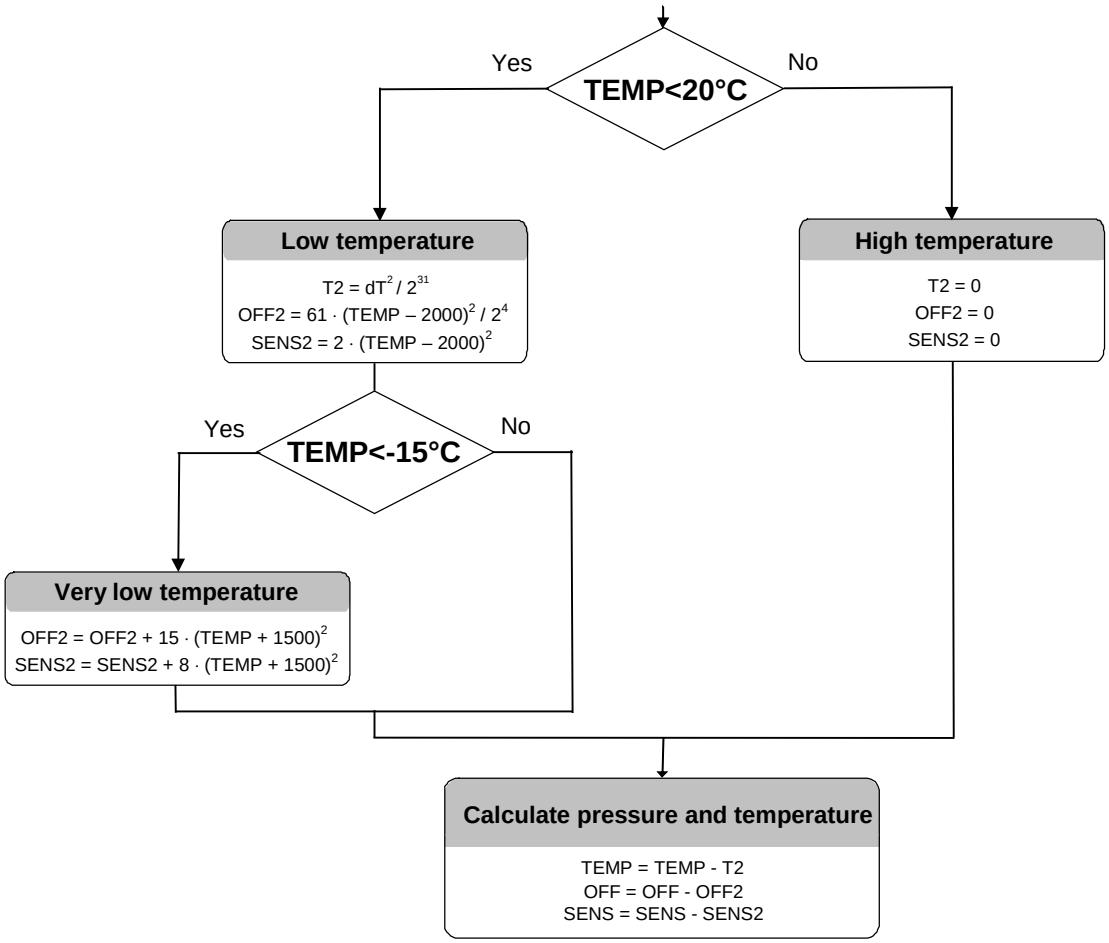


Figure 3: Flow chart for pressure and temperature to the optimum accuracy.

SPI INTERFACE

COMMANDS

Size of each command is 1 byte (8 bits) as described in the table below. After ADC read commands the device will return 24 bit result and after the PROM read 16bit result. The address of the PROM is embedded inside of the PROM read command using the a2, a1 and a0 bits.

	Command byte								hex value
Bit number	0	1	2	3	4	5	6	7	
Bit name	PRM	COV	-	Typ	Ad2/ Os2	Ad1/ Os1	Ad0/ Os0	Stop	
Command									
Reset	0	0	0	1	1	1	1	0	0x1E
Convert D1 (OSR=256)	0	1	0	0	0	0	0	0	0x40
Convert D1 (OSR=512)	0	1	0	0	0	0	1	0	0x42
Convert D1 (OSR=1024)	0	1	0	0	0	1	0	0	0x44
Convert D1 (OSR=2048)	0	1	0	0	0	1	1	0	0x46
Convert D1 (OSR=4096)	0	1	0	0	1	0	0	0	0x48
Convert D2 (OSR=256)	0	1	0	1	0	0	0	0	0x50
Convert D2 (OSR=512)	0	1	0	1	0	0	1	0	0x52
Convert D2 (OSR=1024)	0	1	0	1	0	1	0	0	0x54
Convert D2 (OSR=2048)	0	1	0	1	0	1	1	0	0x56
Convert D2 (OSR=4096)	0	1	0	1	1	0	0	0	0x58
ADC Read	0	0	0	0	0	0	0	0	0x00
PROM Read	1	0	1	0	Ad2	Ad1	Ad0	0	0xA0 to 0xAE

Figure 4: Command structure

RESET SEQUENCE

The Reset sequence shall be sent once after power-on to make sure that the calibration PROM gets loaded into the internal register. It can be also used to reset the device ROM from an unknown condition

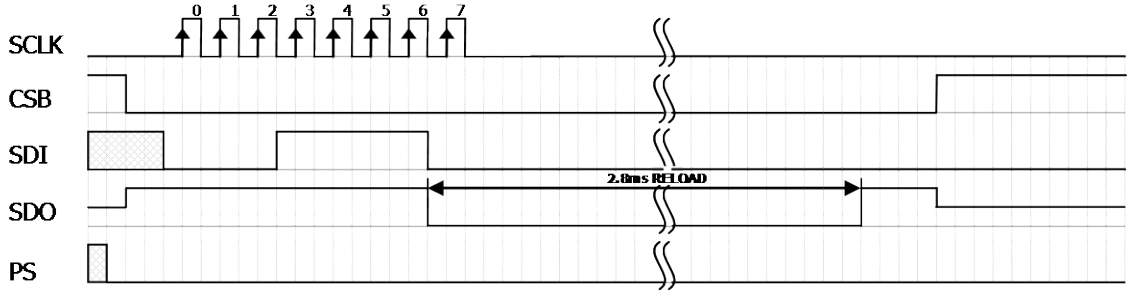


Figure 5: Reset command sequence SPI mode 0

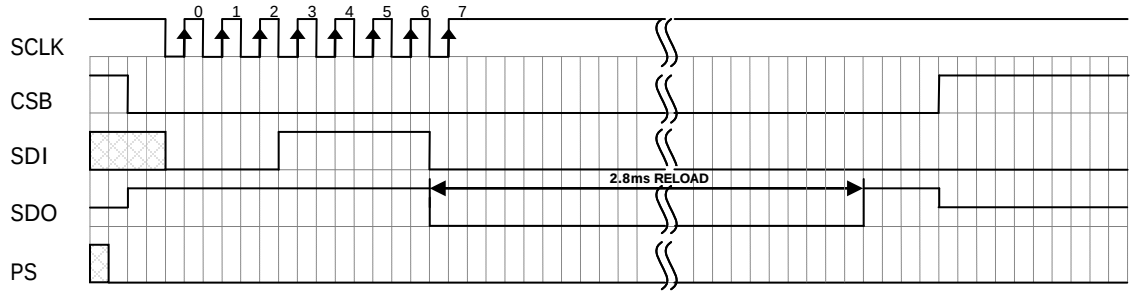


Figure 6: Reset command sequence SPI mode 3

CONVERSION SEQUENCE

The conversion command is used to initiate uncompensated pressure (D1) or uncompensated temperature (D2) conversion. The chip select can be disabled during this time to communicate with other devices.
After the conversion, using ADC read command the result is clocked out with the MSB first. If the conversion is not executed before the ADC read command, or the ADC read command is repeated, it will give 0 as the output result. If the ADC read command is sent during conversion the result will be 0, the conversion will not stop and the final result will be wrong. Conversion sequence sent during the already started conversion process will yield incorrect result as well.

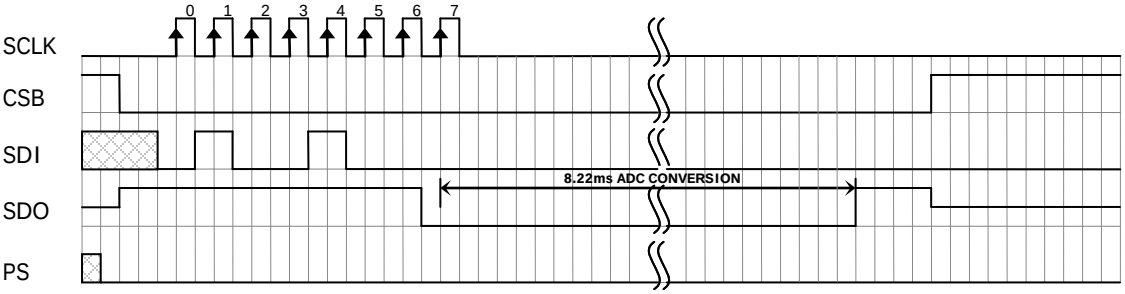


Figure 7: Conversion out sequence, Typ=d1, OSR = 4096

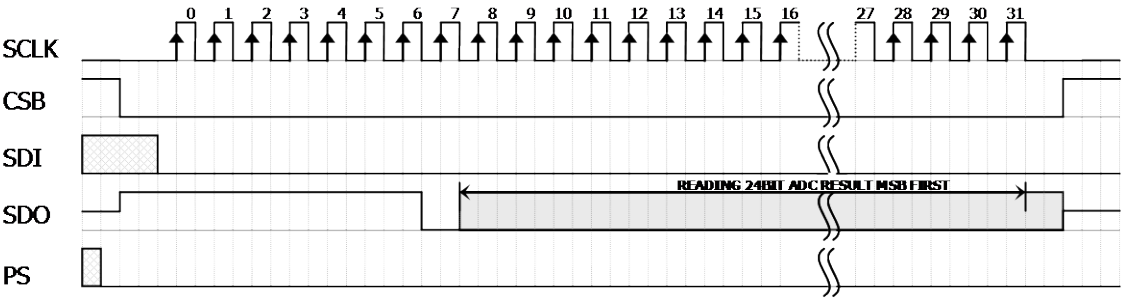


Figure 8: ADC Read sequence

PROM READ SEQUENCE

The read command for PROM shall be executed once after reset by the user to read the content of the calibration PROM and to calculate the calibration coefficients. There are in total 8 addresses resulting in a total memory of 128 bit. Address 0 contains factory data and the setup, addresses 1-6 calibration coefficients and address 7 contains the serial code and CRC. The command sequence is 8 bits long with a 16 bit result which is clocked with the MSB first.

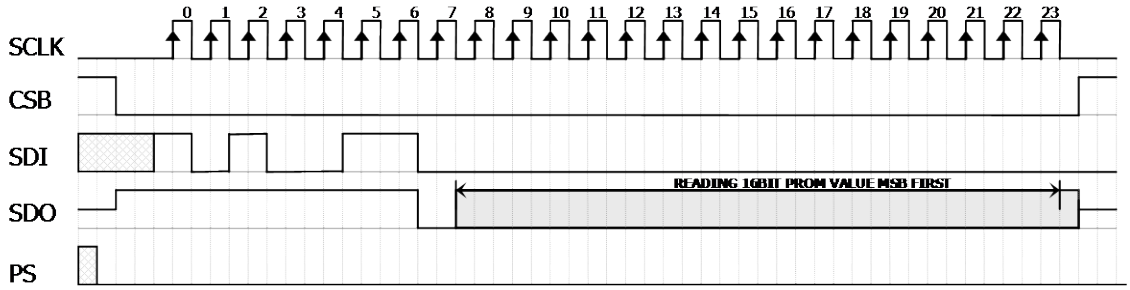


Figure 9: PROM Read sequence, address = 011 (Coefficient 3).

I²C INTERFACE

COMMANDS

Each I²C communication message starts with the start condition and it is ended with the stop condition. The MS5607-02BA address is 111011Cx, where C is the complementary value of the pin CSB. Since the IC does not have a microcontroller inside, the commands for I²C and SPI are quite similar.

RESET SEQUENCE

The reset can be sent at any time. In the event that there is not a successful power on reset this may be caused by the SDA being blocked by the module in the acknowledge state. The only way to get the MS5607-02BA to function is to send several SCLKs followed by a reset sequence or to repeat power on reset.

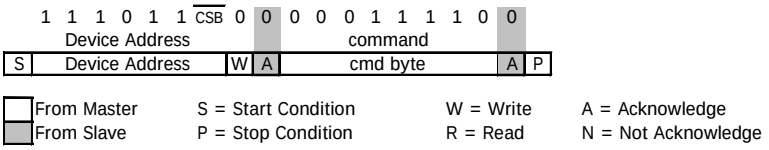


Figure 10: I²C Reset Command

PROM READ SEQUENCE

The PROM Read command consists of two parts. First command sets up the system into PROM read mode. The second part gets the data from the system.

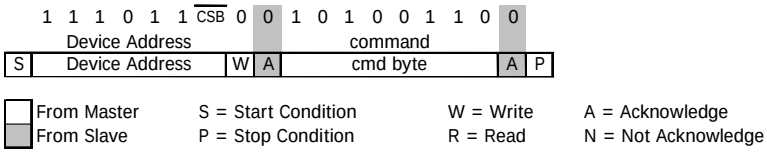


Figure 11: I²C Command to read memory address= 011 (Coefficient 3)

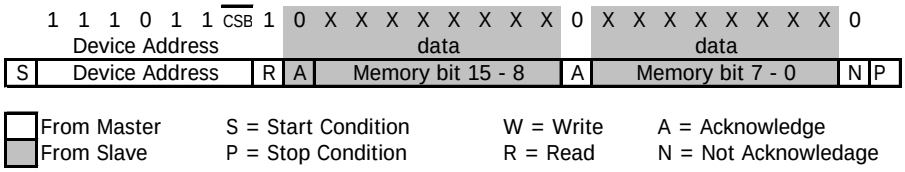


Figure 12: I²C answer from MS5607-02BA

CONVERSION SEQUENCE

A conversion can be started by sending the command to MS5607-02BA. When command is sent to the system it stays busy until conversion is done. When conversion is finished the data can be accessed by sending a Read command, when an acknowledge appears from the MS5607-02BA, 24 SCLK cycles may be sent to receive all result bits. Every 8 bit the system waits for an acknowledge signal.

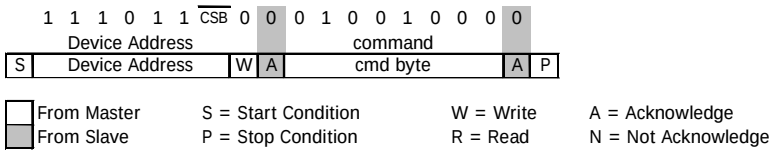


Figure 13: I²C Command to initiate a pressure conversion (OSR=4096, typ=D1)

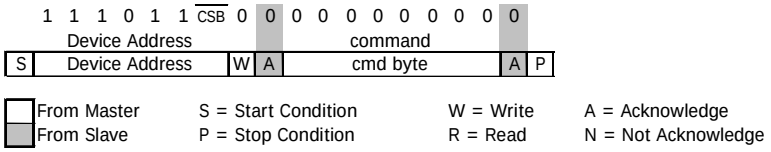


Figure 14: I²C ADC read sequence

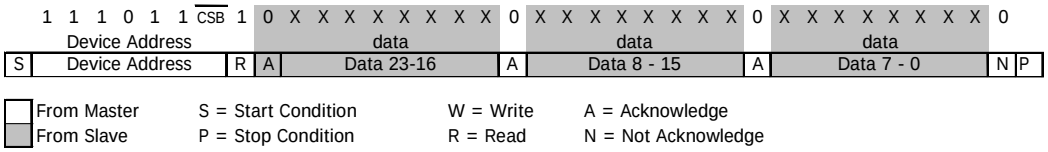


Figure 15: I²C answer from MS5607-02BA

CYCLIC REDUNDANCY CHECK (CRC)

MS5607-02BA contains a PROM memory with 128-Bit. A 4-bit CRC has been implemented to check the data validity in memory. The application note AN520 describes in detail CRC-4 code used.

Add	DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	16 bit reserved for manufacturer															
1	Coefficient 1 (16 bit unsigned)															
2	Coefficient 2 (16 bit unsigned)															
3	Coefficient 3 (16 bit unsigned)															
4	Coefficient 4 (16 bit unsigned)															
5	Coefficient 5 (16 bit unsigned)															
6	Coefficient 6 (16 bit unsigned)															
7														CRC		

Figure 16: Memory PROM mapping

APPLICATION CIRCUIT

The MS5607-02BA is a circuit that can be used in conjunction with a microcontroller in mobile altimeter applications. It is designed for low-voltage systems with a supply voltage of 3 V.

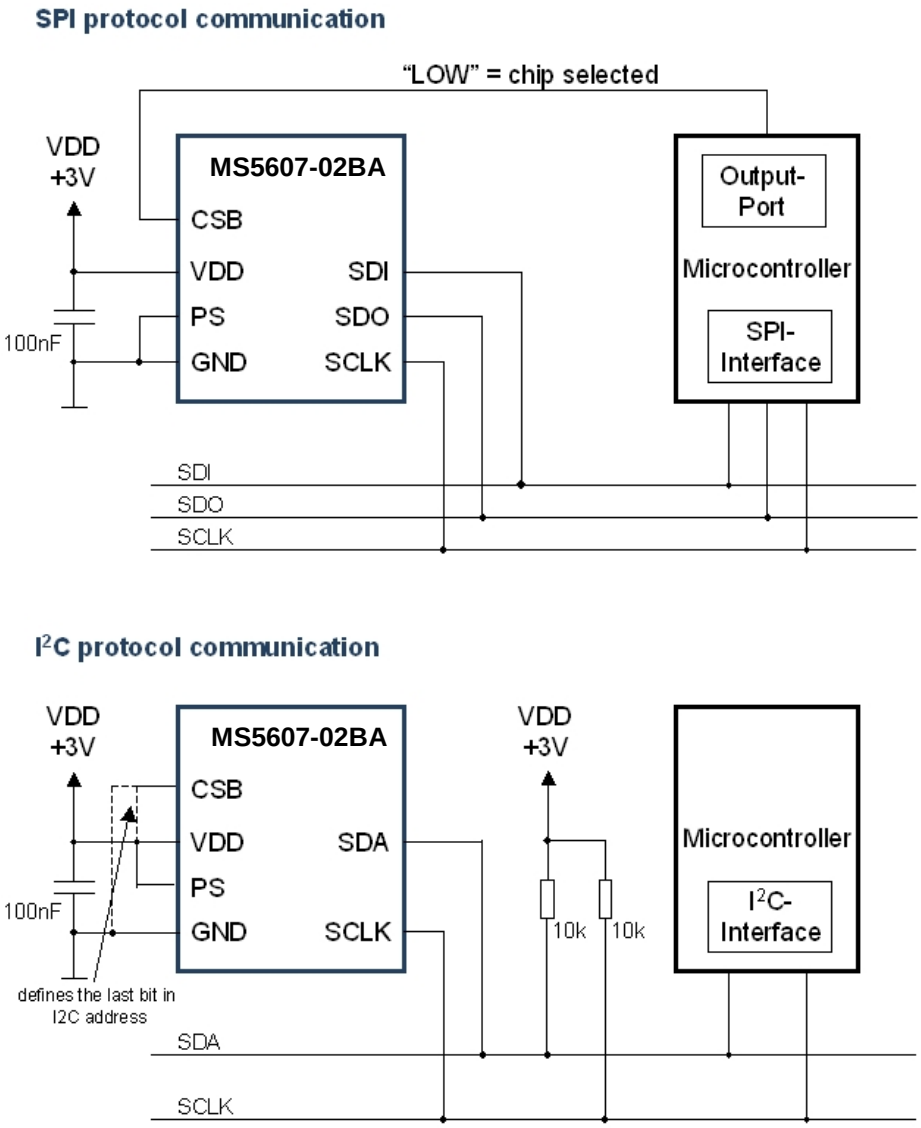
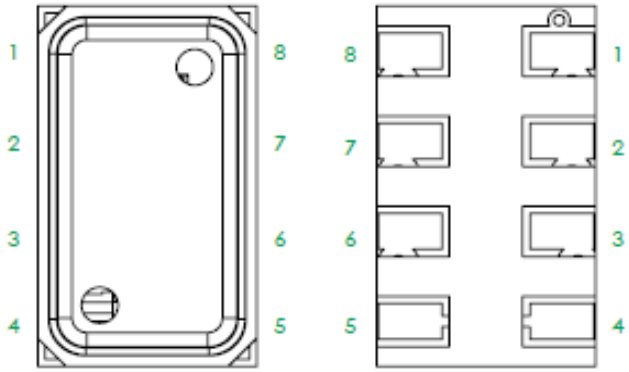


Figure 17: Typical application circuit with SPI / I^2C protocol communication

PIN CONFIGURATION

Pin	Name	Type	Function
1	VDD	P	Positive supply voltage
2	PS	I	Protocol select PS high (VDD) → I^2C PS low (GND) → SPI
3	GND	G	Ground
4	CSB	I	Chip select (active low), internal connection
5			
6	SDO	O	Serial data output
7	SDI / SDA	I / IO	Serial data input / I^2C data IO
8	SCLK	I	Serial data clock



DEVICE PACKAGE OUTLINE

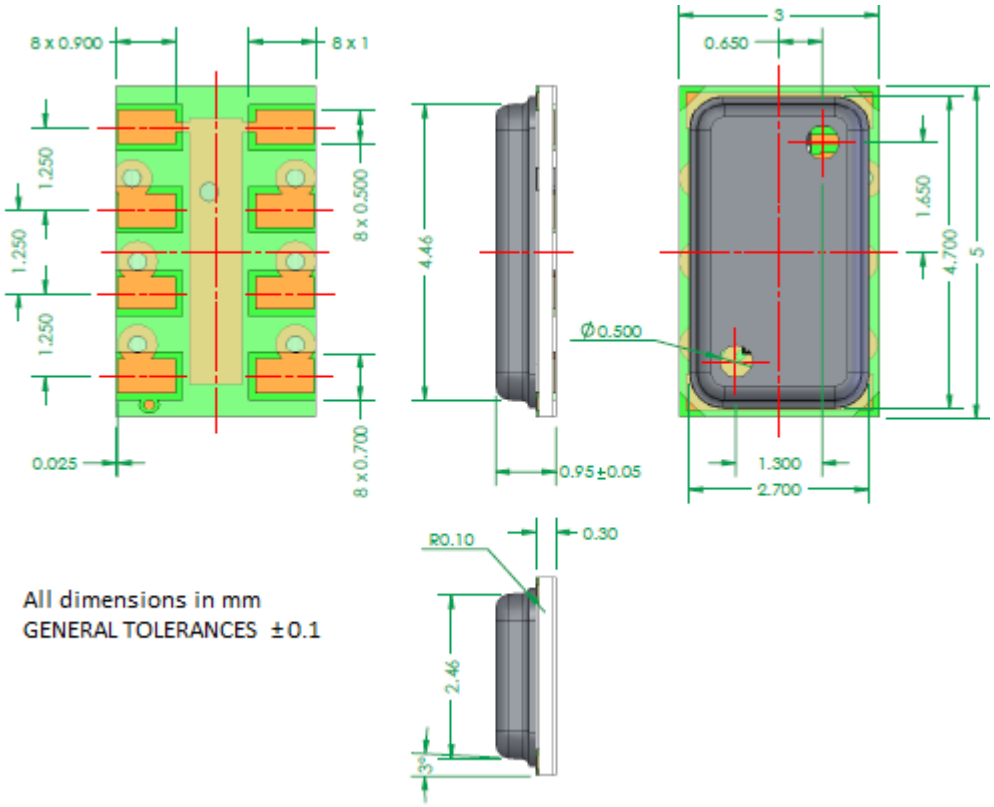
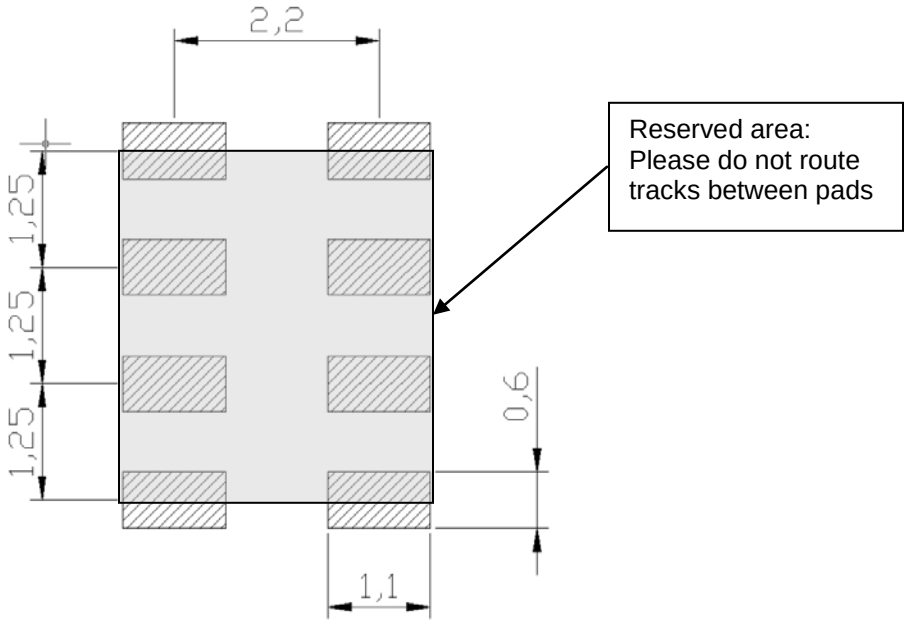


Figure 18: MS5607-02BA03 package outline

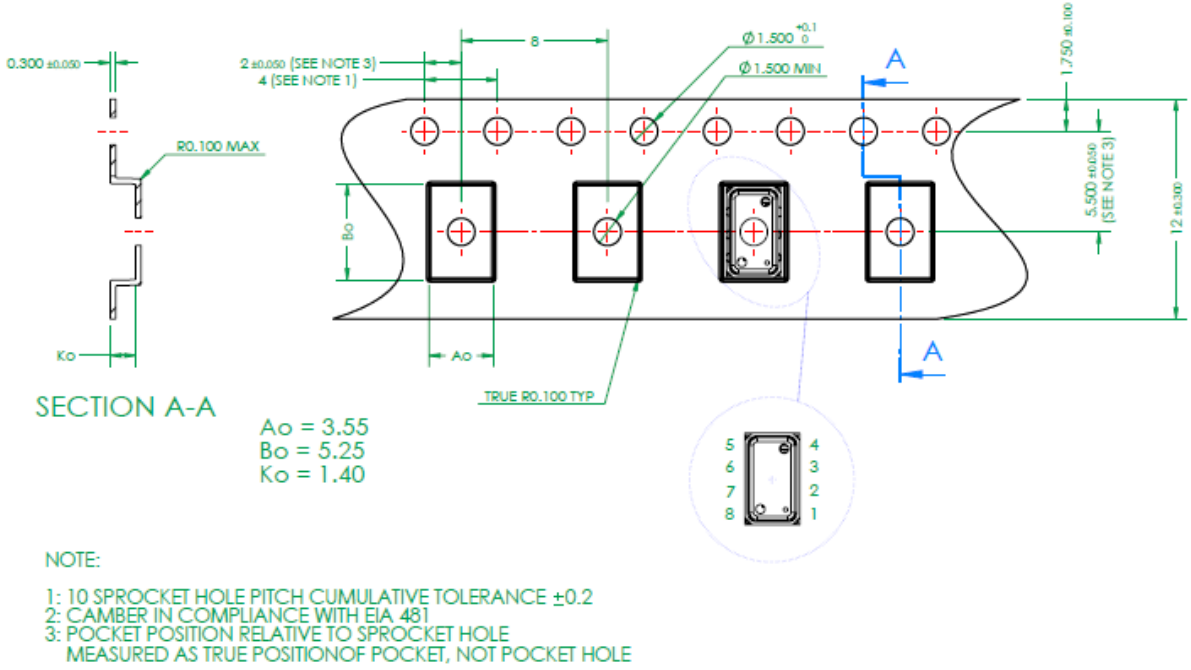
RECOMMENDED PAD LAYOUT

Pad layout for bottom side of the MS5607-02BA soldered onto printed circuit board.



SHIPPING PACKAGE

Tape and Tape and reel



NOTE:

1: 10 SPROCKET HOLE PITCH CUMULATIVE TOLERANCE ±0.2
2: CAMBER IN COMPLIANCE WITH EIA 481
3: POCKET POSITION RELATIVE TO SPROCKET HOLE
MEASURED AS TRUE POSITIONOF POCKET, NOT POCKET HOLE

MOUNTING AND ASSEMBLY CONSIDERATIONS

SOLDERING

Please refer to the application note AN808 available on our website for all soldering issues.

MOUNTING

The MS5607-02BA can be placed with automatic Pick & Place equipment using vacuum nozzles. It will not be damaged by the vacuum. Due to the low stress assembly the sensor does not show pressure hysteresis effects. It is important to solder all contact pads.

CONNECTION TO PCB

The package outline of the module allows the use of a flexible PCB for interconnection. This can be important for applications in watches and other special devices.

CLEANING

The MS5607-02BA has been manufactured under cleanroom conditions. It is therefore recommended to assemble the sensor under class 10'000 or better conditions. Should this not be possible, it is recommended to protect the sensor opening during assembly from entering particles and dust. To avoid cleaning of the PCB, solder paste of type "no-clean" shall be used. Cleaning might damage the sensor!

ESD PRECAUTIONS

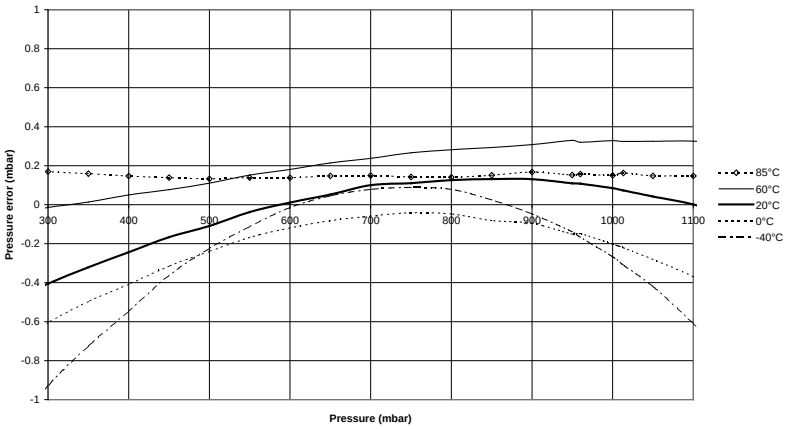
The electrical contact pads are protected against ESD up to 4 kV HBM (human body model). It is therefore essential to ground machines and personnel properly during assembly and handling of the device. The MS5607-02BA is shipped in antistatic transport boxes. Any test adapters or production transport boxes used during the assembly of the sensor shall be of an equivalent antistatic material.

DECOUPLING CAPACITOR

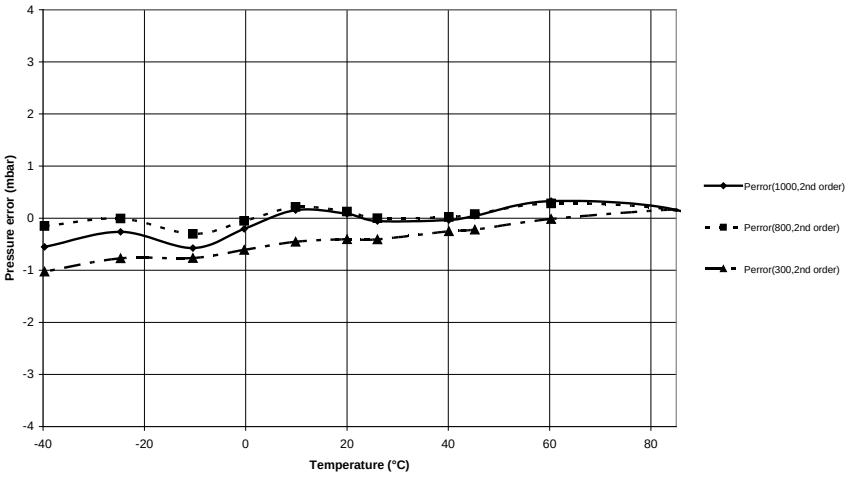
Particular care must be taken when connecting the device to the power supply. A 100 nF ceramic capacitor must be placed as close as possible to the MS5607-02BA VDD pin. This capacitor will stabilize the power supply during data conversion and thus, provide the highest possible accuracy.

TYPICAL PERFORMANCE CHARACTERISTICS

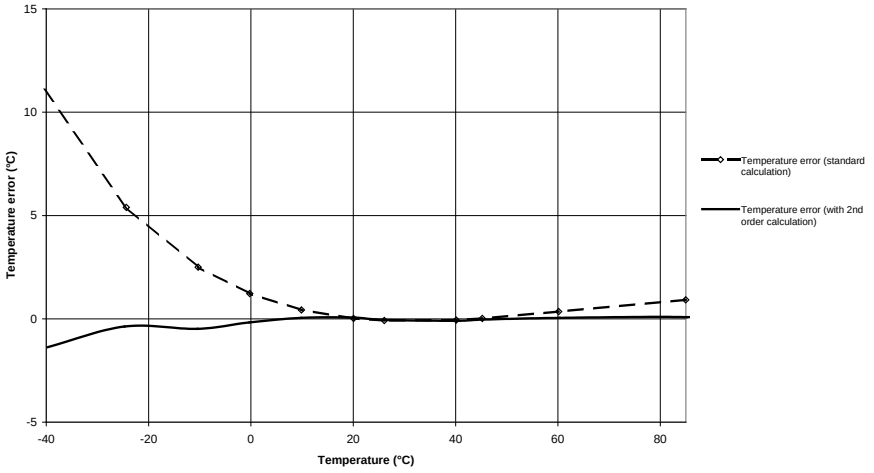
Absolute Pressure Accuracy, 2nd order compensation



Pressure Error Accuracy vs temperature

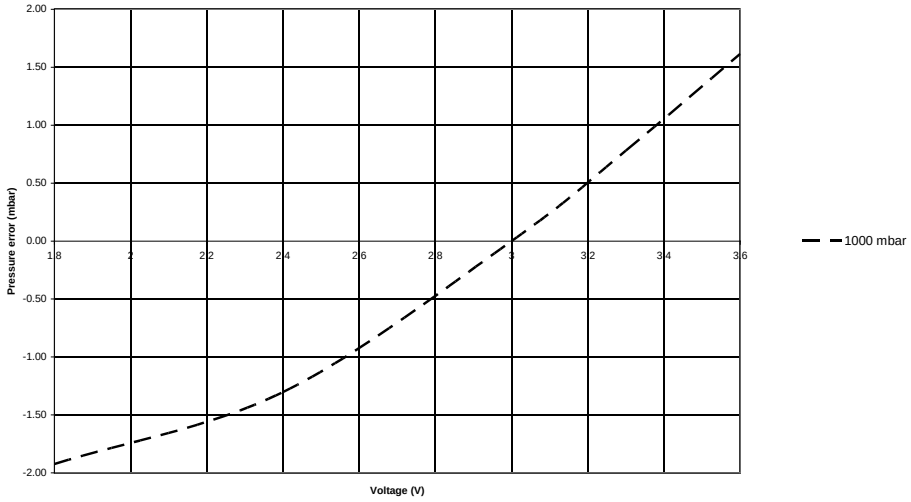


Temperature Error Accuracy vs temperature

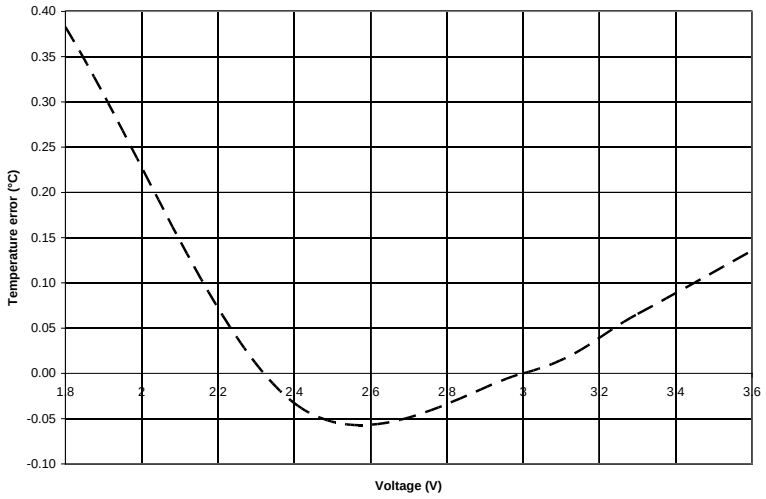


TYPICAL PERFORMANCE CHARACTERISTICS (CONTINUED)

Pressure error vs supply voltage (typical)



Temperature error vs supply voltage (typical)



ORDERING INFORMATION

Product Code	Product	Art. No	Delivery Form
MS5607-02BA03	Barometric Pressure Sensor Thin Metal Cap	MS560702BA03-00	Waffle pack
MS5607-02BA03	Barometric Pressure Sensor Thin Metal Cap	MS560702BA03-50	Tape and reel

FACTORY CONTACTS

NORTH AMERICA

Measurement Specialties
45738 Northport Loop West
Fremont, CA 94538

Tel: +1 800 767 1888
Fax: +1 510 498 1578
e-mail: pfg.cs.amer@meas-spec.com
Website: www.meas-spec.com

EUROPE

MEAS Switzerland Sàrl
Ch. Chapons-des-Prés 11
CH-2022 Bevaix

Tel: +41 32 847 9550
Fax: + 41 32 847 9569
e-mail: sales.ch@meas-spec.com
Website: www.meas-spec.com

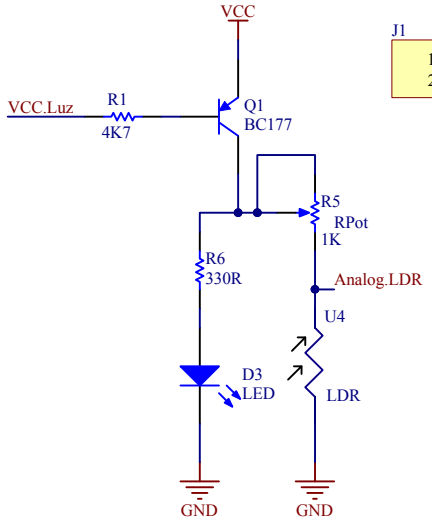
ASIA

Measurement Specialties (China), Ltd.
No. 26 Langshan Road
Shenzhen High-Tech Park (North)
Nanshan District, Shenzhen, 518057
China
Tel: +86 755 3330 5088
Fax: +86 755 3330 5099
e-mail: pfg.cs.asia@meas-spec.com
Website: www.meas-spec.com

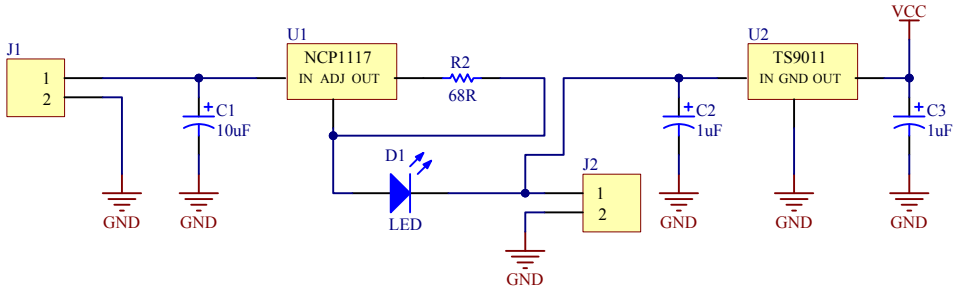
B. Planos

a. Plano esquema general del circuito

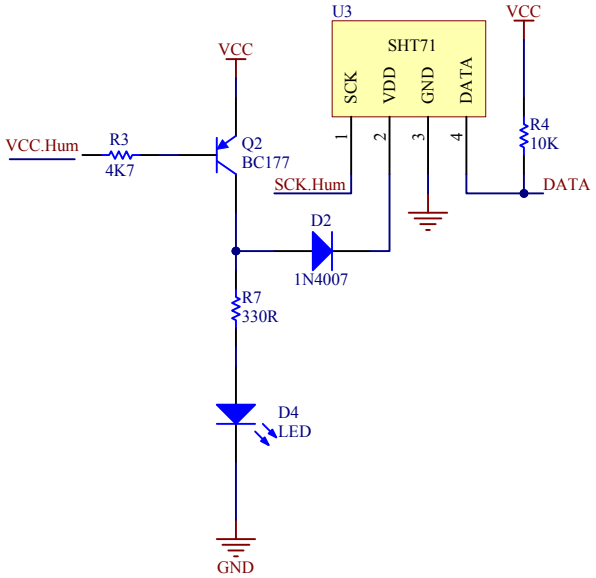
Circuitería Sensor Luz



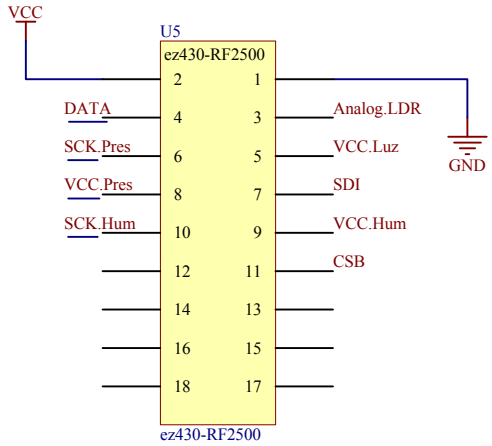
Cargador Solar de Baterías



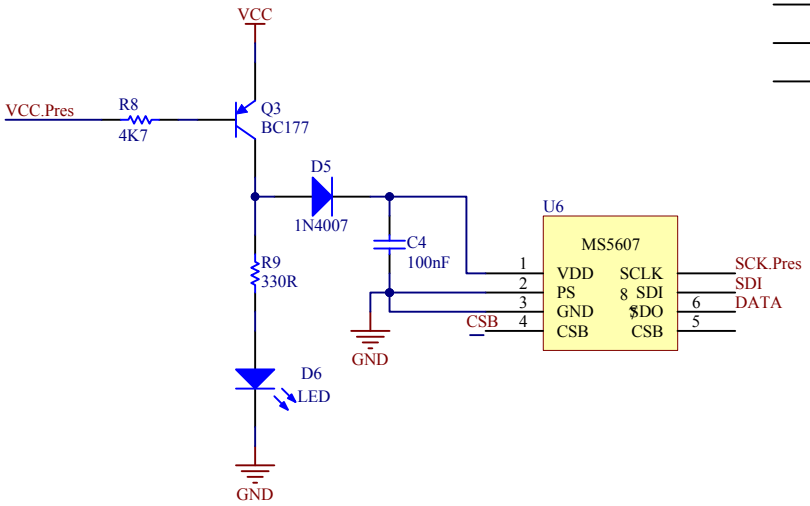
Circuitería Sensor Humedad



Conector Módulo RF



Circuitería Sensor Presión



	Nombre	Fecha	Firma
Dibujado	Leyre Morlas	20/06/2015	
Comprobado	Bonifacio Martín del Brío	03/08/2015	

Escuela de Ingeniería y Arquitectura UniversidadZaragoza

Escala: S/E

Título: Estación meteorológica inalámbrica, de muy bajo consumo e inteligencia embebida

Esquema general del circuito

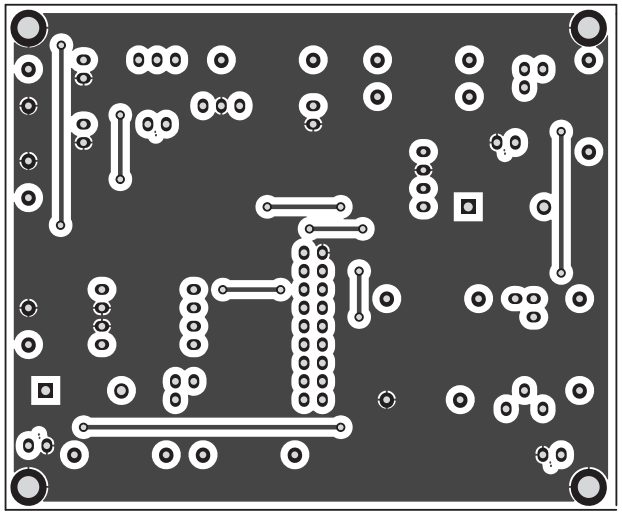
Nº de Plano: 1/4

NIA: 651076

Curso: 4º Electrónicos y Automáticos

B. Planos

b. Plano de circuito impreso cara TOP



	Nombre	Fecha	Firma
Dibujado	Leyre Morlas	20/06/2015	
Comprobado	Bonifacio Martín del Brío	03/08/2015	

Escuela de
Ingeniería y Arquitectura
UniversidadZaragoza

Escala

1:2

Título

Estación meteorológica inalámbrica, de muy bajo consumo e inteligencia embebida

Plano de circuito impreso cara TOP

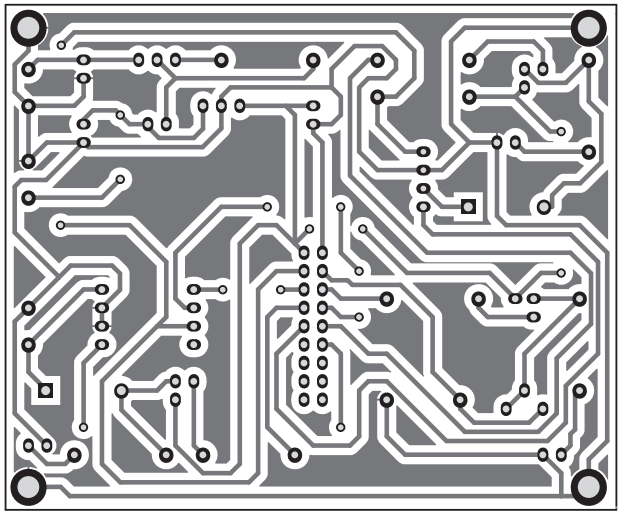
Nº de Plano: 2/4

NIA: 651076

Curso: 4º Electrónicos y Automáticos

B. Planos

c. Plano de circuito impreso cara BOTTOM



	Nombre	Fecha	Firma
Dibujado	Leyre Morlas	20/06/2015	
Comprobado	Bonifacio Martín del Brío	03/08/2015	

Escuela de
Ingeniería y Arquitectura
UniversidadZaragoza

Escala

1:2

Título

Estación meteorológica inalámbrica, de muy bajo consumo e inteligencia embebida

Plano de circuito impreso cara BOTTOM

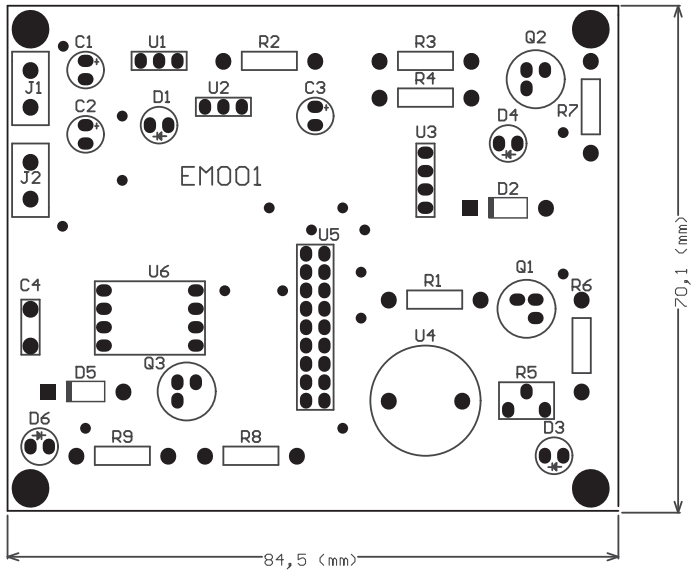
Nº de Plano: 3/4

NIA: 651076

Curso: 4º Electrónicos y Automáticos

B. Planos

d. Plano de serigrafía



	Nombre	Fecha	Firma
Dibujado	Leyre Morlas	20/06/2015	
Comprobado	Bonifacio Martín del Brío	03/08/2015	

Escuela de Ingeniería y Arquitectura UniversidadZaragoza

Escala

1:2

Título

Estación meteorológica inalámbrica, de muy bajo consumo e inteligencia embebida

Plano de serigrafía de componentes

Nº de Plano: 4/4

NIA: 651076

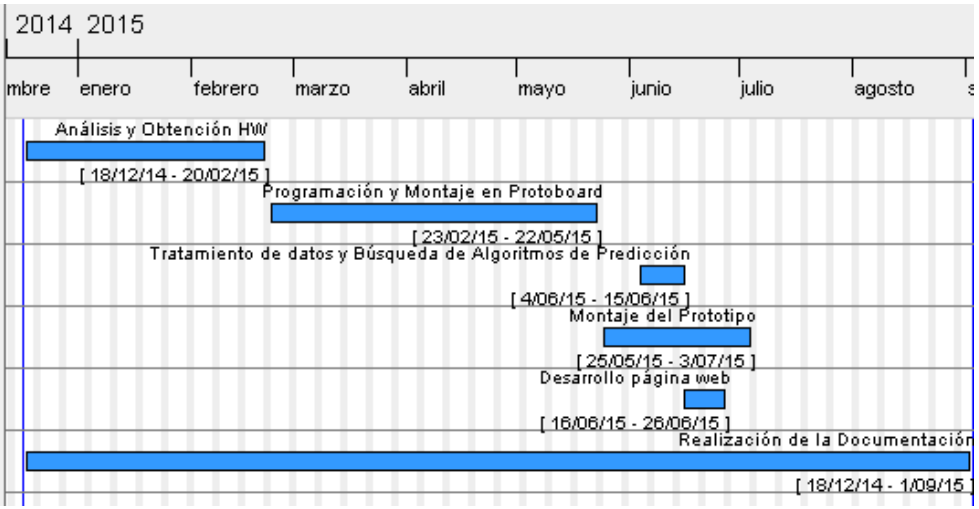
Curso: 4º Electrónicos y Automáticos

C. Presupuesto

Descripción	Identificación Particular	Fabricante	Proveedor	Referencia	Cantidad	Precio Unitario	Total (€)
Resistencia	330 Ω	Multicomp	Farnell	1128099	3	0,008	0,024
Resistencia	4k7 Ω	Multicomp	Farnell	1129205	3	0,005	0,015
Resistencia	10 kΩ	Multicomp	Farnell	1128067	1	0,008	0,008
Resistencia	68 Ω	Te Connectivity	RS	131968	1	0,036	0,036
Potenciómetro	5 kΩ	Vishay Spectrol	Farnell	9608613	1	1,9	1,9
Condensador	1µF	Nichicon	RS	5194481	2	0,354	0,708
Condensador	10µF	Panasonic	RS	8073551	1	0,031	0,031
Condensador	100 nF	AVX	Farnell	1398033	1	0,033	0,033
Diodo	1N4007	Multicomp	Farnell	2306357	3	0,02	0,06
Transistor PNP	BC1774A	Multicomp	Farnell	9206825	3	2,35	7,05
Transistor PNP	2N3251A	Multicomp	Farnell	186529	3	0,568	1,704
Led	Rojo	Vishay	Farnell	1045457	1	0,288	0,288
Led	Verde	Vishay	Farnell	1045478	1	0,135	0,135
Led	Amarillo	Vishay	Farnell	1045482	1	0,127	0,127
Regulador de tensión ajustable	NCP1117STAT3G	On Semiconductor	RS	785-7200	1	0,342	0,342
Regulador de tensión fijo	TS9011	Taiwan Semiconductor	RS	398625	1	0,31	0,31
Resistencia	LDR	Excelitas tech	Farnell	1652638	1	0,471	0,471
Sensor de Humedad y Temperatura	SHT71	Sensirion	Farnell	1590513	1	23,67	23,67
Sensor de Presión y Temperatura	MS5607	Measurement Specialties	Farnell	2362660	1	3,05	3,05
Baliza	Baliza Solar Inspire Tobago	Leroy Merlin	Leroy Merlin	16592520	6	1,49	8,94
Módulo Microcontrolador con antena de Radio frecuencia	MSP430 eZ430-RF2500-SEH	Texas Instrument	Farnell	1740342	1	161,8	161,8
Portabaterías	4xAAA	RS	RS	512-3568	1	0,97	0,97
Bloques Terminales para PCB	2x5,08	Weidmuller	RS	425-8720	2	0,192	0,384
				Coste Total Componentes Prototipo			212,056

D. Planificación

Nombre	Fecha de inicio	Fecha de fin
Análisis y Obtención HW	18/12/14	20/02/15
Programación y Montaje en Protoboard	23/02/15	22/05/15
Tratamiento de datos y Búsqueda de Algoritmos de Predicción	4/06/15	15/06/15
Montaje del Prototipo	25/05/15	3/07/15
Desarrollo página web	16/06/15	26/06/15
Realización de la Documentación	18/12/14	1/09/15



E. Software Code Composer

a. Access Point

```
//******************************************************************************
// THIS PROGRAM IS PROVIDED "AS IS". TI MAKES NO WARRANTIES OR
// REPRESENTATIONS, EITHER EXPRESS, IMPLIED OR STATUTORY,
// INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS
// FOR A PARTICULAR PURPOSE, LACK OF VIRUSES, ACCURACY OR
// COMPLETENESS OF RESPONSES, RESULTS AND LACK OF NEGLIGENCE.
// TI DISCLAIMS ANY WARRANTY OF TITLE, QUIET ENJOYMENT, QUIET
// POSSESSION, AND NON-INFRINGEMENT OF ANY THIRD PARTY
// INTELLECTUAL PROPERTY RIGHTS WITH REGARD TO THE PROGRAM OR
// YOUR USE OF THE PROGRAM.
//
// IN NO EVENT SHALL TI BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
// CONSEQUENTIAL OR INDIRECT DAMAGES, HOWEVER CAUSED, ON ANY
// THEORY OF LIABILITY AND WHETHER OR NOT TI HAS BEEN ADVISED
// OF THE POSSIBILITY OF SUCH DAMAGES, ARISING IN ANY WAY OUT
// OF THIS AGREEMENT, THE PROGRAM, OR YOUR USE OF THE PROGRAM.
// EXCLUDED DAMAGES INCLUDE, BUT ARE NOT LIMITED TO, COST OF
// REMOVAL OR REINSTALLATION, COMPUTER TIME, LABOR COSTS, LOSS
// OF GOODWILL, LOSS OF PROFITS, LOSS OF SAVINGS, OR LOSS OF
// USE OR INTERRUPTION OF BUSINESS. IN NO EVENT WILL TI'S
// AGGREGATE LIABILITY UNDER THIS AGREEMENT OR ARISING OUT OF
// YOUR USE OF THE PROGRAM EXCEED FIVE HUNDRED DOLLARS
// (U.S.$500).
//
// Unless otherwise stated, the Program written and copyrighted
// by Texas Instruments is distributed as "freeware".  You may,
// only under TI's copyright in the Program, use and modify the
// Program without any charge or restriction.  You may
// distribute to third parties, provided that you transfer a
// copy of this license to the third party and the third party
// agrees to these terms by its first use of the Program. You
// must reproduce the copyright notice and any other legend of
// ownership on each copy or partial copy, of the Program.
//
// You acknowledge and agree that the Program contains
// copyrighted material, trade secrets and other TI proprietary
// information and is protected by copyright laws,
// international copyright treaties, and trade secret laws, as
// well as other intellectual property laws.  To protect TI's
// rights in the Program, you agree not to decompile, reverse
// engineer, disassemble or otherwise translate any object code
// versions of the Program to a human-readable form.  You agree
// that in no event will you alter, remove or destroy any
// copyright notice included in the Program.  TI reserves all
// rights not specifically granted under this license. Except
// as specifically provided herein, nothing in this agreement
// shall be construed as conferring by implication, estoppel,
// or otherwise, upon you, any license or other right under any
// TI patents, copyrights or trade secrets.
//
// You may not use the Program in non-TI devices.
//
//******************************************************************************
//  eZ430-RF2500-SEH Temperature Sensor Access Point using Cymbet Solar Energy
//  Harvesting Board
//
//  Description:
//      This is the Access Point software for the eZ430-RF2500-SEH Temperature
//      Sensing demo when hooked up to a Cymbet solar Energy Harvester board.
//
//      The Energy Harvester Access Point samples its local temperature and
//      voltage and listens to any incoming End Device connections.
//
//      This firmware is based off of Texas Instrument's SimpliciTI network
//      protocol version 1.06.
//
// W. Goh
// Version 1.5
// Texas Instruments, Inc
// March 2009
// Built with IAR Embedded Workbench Version: 4.11B
// Built with Code Composer Essentials Version: v3.1 build 3.2.3.6.4
//******************************************************************************
// Change Log:
//******************************************************************************
// Version:  1.50
// Comments: Fixed an un-initialized bug inside SimpliciTI
// Version:  1.40
//           Application files now compiles on both IAR and CCE
// Version:  1.30
// Comments: Initial public release
// Version:  1.00 using Simpliciti ver 1.06
// Comments: Initial Release Version
//******************************************************************************
// Modificado por Leyre Morlas Funes
// 2015
//******************************************************************************

#include "bsp.h"
#include "mrfi.h"
#include "bsp_leds.h"
#include "bsp_buttons.h"
#include "nwk_types.h"
#include "nwk_api.h"
#include "nwk_frame.h"
#include "nwk.h"

#include "msp430x22x4.h"
#include "vlo_rand.h"

#define MESSAGE_LENGTH 3
void TXString( char* string, int length );
void MCU_Init(void);
void transmitData(int addr, signed char rssi,  char msg[MESSAGE_LENGTH] );
void transmitDataString(char addr[4],char rssi[3], char msg[MESSAGE_LENGTH]);
void createRandomAddress(void);
```

```
float Calcula_Rocio(float H, float T);

// data for terminal output
const char splash[] = {"\r\n-------------------------------------------------\r\n     ****\r\n     ****
eZ430-RF2500\r\n     ******o****    Temperature Sensor Network\r\n*******_///_****   Copyright 2007\r\n ******/_//_/*****
Texas Instruments Incorporated\r\n  ** ***(__/*****   All rights reserved.\r\n      *********     Version 1.03\r\n
*****\r\n        ***\r\n-------------------------------------------------\r\n"};

// reserve space for the maximum possible peer Link IDs
static linkID_t sLID[NUM_CONNECTIONS] = {0};
static uint8_t  sNumCurrentPeers = 0;

// callback handler
static uint8_t sCB(linkID_t);

// work loop semaphores
static uint8_t sPeerFrameSem = 0;
static uint8_t sJoinSem = 0;
static uint8_t sSelfMeasureSem = 0;

// mode data verbose = default, deg F = default
char verboseMode = 1;
char degCMode =1;

// -------------------------------------------------------------------------------------------------------------------
// Variables globales añadidas
// -------------------------------------------------------------------------------------------------------------------
int temperature[4] = {0};                                                   // Array de temperaturas
char output_sensores[] =
    {"\r\nS1:-XX.X,S2:X.X,S3:-XX.X,S4:XXX.X,S5:-XX.X,S6:XXXX.X"};           // Formato de la cadena de los sensores

// ===================================================================================================================
// Main. Entrada del programa
// ===================================================================================================================
void main (void) {
    addr_t lAddr;
    bspIState_t intState;
    char *Flash_Addr;                                                       // Initialize radio address location
    Flash_Addr = (char *)0x10F0;

    WDTCTL = WDTPW + WDTHOLD;                                               // Stop WDT
    // delay loop to ensure proper startup before SimpliciTI increases DCO
    // This is typically tailored to the power supply used, and in this case
    // is overkill for safety due to wide distribution.
    __delay_cycles(65000);

    if( CALBC1_8MHZ == 0xFF && CALDCO_8MHZ == 0xFF ) {                      // Do not run if cal values
        P1DIR |= 0x03;
        BSP_TURN_ON_LED1();
        BSP_TURN_OFF_LED2();
        while(1) {
            __delay_cycles(65000);
            BSP_TOGGLE_LED2();
            BSP_TOGGLE_LED1();
        }
    }

    BSP_Init();

    if( Flash_Addr[0] == 0xFF &&
    Flash_Addr[1] == 0xFF &&
    Flash_Addr[2] == 0xFF &&
    Flash_Addr[3] == 0xFF ) {
        createRandomAddress();                                              // Create Random device address at
    }                                                                       // initial startup if missing

    lAddr.addr[0]=Flash_Addr[0];
    lAddr.addr[1]=Flash_Addr[1];
    lAddr.addr[2]=Flash_Addr[2];
    lAddr.addr[3]=Flash_Addr[3];

    // SMPL_Init();
    SMPL_Ioctl(IOCTL_OBJ_ADDR, IOCTL_ACT_SET, &lAddr);
    MCU_Init();

    // Transmit splash screen and network init notification
    TXString( (char*)splash, sizeof splash);
    TXString( "\r\nInitializing Network....", 26 );
    SMPL_Init(sCB);

    // network initialized
    TXString( "Done\r\n", 6);

    // main work loop
    while(1)  {
        // Wait for the Join semaphore to be set by the receipt of a Join frame from a
        // device that supports and End Device.

        if (sJoinSem && (sNumCurrentPeers < NUM_CONNECTIONS)) {
            // listen for a new connection
            SMPL_LinkListen(&sLID[sNumCurrentPeers]);
            sNumCurrentPeers++;
            BSP_ENTER_CRITICAL_SECTION(intState);

            if (sJoinSem) {
                sJoinSem--;
            }
            BSP_EXIT_CRITICAL_SECTION(intState);
        }

        // if it is time to measure our own temperature...
        if (sSelfMeasureSem) {
            char msg [8];
            char addr[] = {"HUB0"};
            char rssi[] = {"000"};
            int degC, volt;
```

```
            volatile long temp;
            int results[2];
            int *tempOffset;                                            // Initialize temperature offset
            tempOffset = (int *)0x10F4;                                 // coefficient

            ADC10CTL1 = INCH_10 + ADC10DIV_4;                           // Temp Sensor ADC10CLK/5
            ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON + ADC10IE + ADC10SR;
            __delay_cycles(250);                                        // delay to allow reference to settle

            ADC10CTL0 |= ENC + ADC10SC;                                 // Sampling and conversion start
            __bis_SR_register(CPUOFF + GIE);                            // LPM0 with interrupts enabled
            results[0] = ADC10MEM;

            ADC10CTL0 &= ~ENC;
            ADC10CTL1 = INCH_11;                                        // AVcc/2
            ADC10CTL0 = SREF_1 + ADC10SHT_2 + REFON + ADC10ON + ADC10IE + REF2_5V;
            __delay_cycles(250);                                        // delay to allow reference to settle

            ADC10CTL0 |= ENC + ADC10SC;                                 // Sampling and conversion start
            __bis_SR_register(CPUOFF + GIE);                            // LPM0 with interrupts enabled
            results[1] = ADC10MEM;
            ADC10CTL0 &= ~ENC;
            ADC10CTL0 &= ~(REFON + ADC10ON);                            // turn off A/D to save power

            // Collect VCC Sample
            temp = results[1];
            volt = (temp*25)/512;

            // oC = ((A10/1024)*1500mV)-986mV)*1/3.55mV = A10*423/1024 - 278
            // the temperature is transmitted as an integer where 32.1 = 321
            // hence 4230 instead of 423
            temp = results[0];
            degC = ((temp * 4230) / 1024) - 2780;
            if( *tempOffset != 0xFFFF ) degC += *tempOffset;

            // Moving window of samples
            temperature[0] = temperature[1];
            temperature[1] = temperature[2];
            temperature[2] = temperature[3];
            temperature[3] = degC;

            // If the low-pass filter buffer is not yet full, do not send data to AP
            if(temperature[0] != 0) {
                // Average the last 4 samples
                temp = temperature[0] + temperature[1] + temperature[2] + temperature[3];
                temp = temp >> 2;                                       // Divide by 4

                // Load the message
                msg[0] = temp&0xFF;
                msg[1] = (temp>>8)&0xFF;
                msg[2] = volt;
                msg[4] = 0;
                msg[5] = 0;
                msg[6] = 0;
                msg[7]= 0;
                transmitDataString(addr, rssi, msg );
            }

            BSP_TOGGLE_LED1();
            sSelfMeasureSem = 0;
        }

        // Have we received a frame on one of the ED connections?
        // No critical section -- it doesn't really matter much if we miss a poll
        if (sPeerFrameSem) {
            uint8_t     msg[MAX_APP_PAYLOAD], len, i;

            // process all frames waiting
            for (i=0; i<sNumCurrentPeers; ++i) {
                if (SMPL_Receive(sLID[i], msg, &len) == SMPL_SUCCESS) {
                    ioctlRadioSiginfo_t sigInfo;
                    sigInfo.lid = sLID[i];
                    SMPL_Ioctl(IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SIGINFO, (void *)&sigInfo);
                    transmitData( i, (signed char)sigInfo.sigInfo.rssi, (char*)msg );
                    BSP_TURN_ON_LED2();                                 // Toggle LED2 when received packet
                    BSP_ENTER_CRITICAL_SECTION(intState);
                    sPeerFrameSem--;
                    BSP_EXIT_CRITICAL_SECTION(intState);
                    __delay_cycles(10000);
                    BSP_TURN_OFF_LED2();
                }
            }
        }
    }
}

// ============================================================================================================================
// BEGHDR
// NAME: createRandomAddress()
// DESCRIPTION: generate random address
// ============================================================================================================================
void createRandomAddress(void) {
    unsigned int rand, rand2;
    char *Flash_Addr;                                                   // Initialize radio address location
    Flash_Addr = (char *)0x10F0;

    do  {
        rand = TI_getRandomIntegerFromVLO();                            // first byte can not be 0x00 of 0xFF
    }

    while ((rand & 0xFF00)==0xFF00 || (rand & 0xFF00)==0x0000);
    rand2 = TI_getRandomIntegerFromVLO();

    BCSCTL1 = CALBC1_1MHZ;                                              // Set DCO to 1MHz
    DCOCTL = CALDCO_1MHZ;
    FCTL2 = FWKEY + FSSEL0 + FN1;                                       // MCLK/3 for Flash Timing Generator
```

```
    FCTL3 = FWKEY + LOCKA;                                                  // Clear LOCK & LOCKA bits
    FCTL1 = FWKEY + WRT;                                                    // Set WRT bit for write operation

    Flash_Addr[0]=(rand>>8) & 0xFF;
    Flash_Addr[1]=rand & 0xFF;
    Flash_Addr[2]=(rand2>>8) & 0xFF;
    Flash_Addr[3]=rand2 & 0xFF;

    FCTL1 = FWKEY;                                                          // Clear WRT bit
    FCTL3 = FWKEY + LOCKA + LOCK;                                           // Set LOCK & LOCKA bit
}

// ================================================================================================================================
// transmitData
// ================================================================================================================================
void transmitData(int addr, signed char rssi,  char msg[MESSAGE_LENGTH] ) {
    char addrString[4];
    char rssiString[3];
    volatile signed int rssi_int;

    addrString[0] = '0';
    addrString[1] = '0';
    addrString[2] = '0'+(((addr+1)/10)%10);
    addrString[3] = '0'+((addr+1)%10);
    rssi_int = (signed int) rssi;
    rssi_int = rssi_int+128;
    rssi_int = (rssi_int*100)/256;
    rssiString[0] = '0'+(rssi_int%10);
    rssiString[1] = '0'+((rssi_int/10)%10);
    rssiString[2] = '0'+((rssi_int/100)%10);

    transmitDataString( addrString, rssiString, msg );
}

// ================================================================================================================================
// transmitDataString
// ================================================================================================================================
void transmitDataString(char addr[4],char rssi[3], char msg[MESSAGE_LENGTH] ) {
    char temp_string[]={"XX.XC"};
    int temp=0;
    int tcnt=0;
    int TempSHT=0, HumSHT=0;
    int TempPr=0, Pres=0;
    float TempSHTF=0.0,HumSHTF=0.0;

    if(msg[7]==0) {                                                         // Sensor de temperatura del Access Point
        int temp = msg[0] + (msg[1]<<8);
        int tcnt=msg[4] + (msg[5]<<8);
        if( !degCMode ) {
            temp = (int) ((((float)temp)*1.8)+320);
            temp_string[5] = 'F';
        }
        if( temp < 0 ) {
            temp_string[0] = '-';
            temp = temp * -1;
        } else {
            if (((temp/1000)%10) != 0 ) {
                temp_string[0] = '0'+((temp/1000)%10);
            } else temp_string[0] ='+';
        }
        temp_string[4] = '0'+(temp%10);
        temp_string[2] = '0'+((temp/10)%10);
        temp_string[1] = '0'+((temp/100)%10);

        char output_verbose[] = {"\r\nNode:XXXX,Temp:-XX.XC,Battery:X.XV,Strength:XXX%,RE:XXXXXXX "};

        output_verbose[46] = rssi[2];
        output_verbose[47] = rssi[1];
        output_verbose[48] = rssi[0];
        output_verbose[17] = temp_string[0];
        output_verbose[18] = temp_string[1];
        output_verbose[19] = temp_string[2];
        output_verbose[20] = temp_string[3];
        output_verbose[21] = temp_string[4];
        output_verbose[22] = temp_string[5];
        output_verbose[32] = '0'+(msg[2]/10)%10;
        output_verbose[34] = '0'+(msg[2]%10);
        output_verbose[7] = addr[0];
        output_verbose[8] = addr[1];
        output_verbose[9] = addr[2];
        output_verbose[10] = addr[3];
        output_verbose[54]= '0'+(msg[3]/100)%10;
        output_verbose[55] = '0'+(msg[3]/10)%10;
        output_verbose[56] = '0'+(msg[3]%10);
        output_verbose[57] = '0'+(msg[6]%10);
        output_verbose[58] = '0'+((tcnt/100)%10);
        output_verbose[59] = '0'+((tcnt/10)%10);
        output_verbose[60] = '0'+(tcnt%10);
    } else {
        if(msg[7]==1) {                                                     // Sensor integrado de Temperatura
            int temp = msg[0] + (msg[1]<<8);
            int tcnt=msg[4] + (msg[5]<<8);
            if( !degCMode ) {
                temp = (int) ((((float)temp)*1.8)+320);
                temp_string[5] = 'F';
            }

            if (temp < 0) {
                temp_string[0] = '-';
                temp = temp * -1;
            } else {
                if (((temp/1000)%10) != 0) {
                    temp_string[0] = '0'+((temp/1000)%10);
                } else temp_string[0] = '+';
            }
```

```
            temp_string[4] = '0'+(temp%10);
            temp_string[2] = '0'+((temp/10)%10);
            temp_string[1] = '0'+((temp/100)%10);

            output_sensores[5] = temp_string[0];
            output_sensores[6] = temp_string[1];
            output_sensores[7] = temp_string[2];
            output_sensores[9] = temp_string[4];
        }

        if (msg[7]==2) {                                                    // Tensión LDR
            output_sensores[14] = '0'+ (msg[8]/10)%10 ;
            output_sensores[16] = '0'+ (msg[8]%10);
        }

        if (msg[7]==3) {                                                    // Sensor Humedad
            TempSHT=msg[0] + (msg[1]<<8);
            HumSHT=msg[2] + (msg[8]<<8);
            HumSHTF=HumSHT/10.0;
            TempSHTF=TempSHT/10.0;

            if (msg[9]==1) {
                output_sensores[21] = '-';
            } else {
                output_sensores[21] = '+';
            }

            output_sensores[22] = '0'+((TempSHT/100)%10);
            output_sensores[23] = '0'+((TempSHT/10)%10);
            output_sensores[25] = '0'+(TempSHT%10);

            output_sensores[30] = '0'+(HumSHT/100);
            output_sensores[31] = '0'+((HumSHT/100)%10);
            output_sensores[32] ='0'+((HumSHT/10)%10);
            output_sensores[34] = '0'+(HumSHT%10);
        }

        if (msg[7]==4) {                                                    // Sensor de Presión
            TempPr=msg[0] + (msg[1]<<8);
            Pres=msg[2] + (msg[8]<<8);

            if (msg[9]==1) {
                output_sensores[39] = '-';
            } else {
                output_sensores[39] = '+';
            }

            output_sensores[40] = '0'+((TempPr/100)%10);
            output_sensores[41] = '0'+((TempPr/10)%10);
            output_sensores[43] = '0'+(TempPr%10);
            output_sensores[48] = '0'+(Pres/10000);
            output_sensores[49] = '0'+((Pres/1000)%10);
            output_sensores[50] = '0'+((Pres/100)%10);
            output_sensores[51] ='0'+((Pres/10)%10);
            output_sensores[53] = '0'+(Pres%10);

            TXString(output_sensores, sizeof output_sensores );

            char output_sensores[]={"\r\nS1:-XX.X,S2:X.X,S3:-XX.X,S4:XXX.X,S5:-XX.X,S6:XXXX.X"};
        }
    }
}

// ================================================================================================================
// TXString
// ================================================================================================================
void TXString(char* string, int length) {
    int pointer;
    for (pointer = 0; pointer < length; pointer++) {
        UCA0TXBUF = string[pointer];
        while (!(IFG2&UCA0TXIFG));                                          // USCI_A0 TX buffer ready?
    }
}

// ================================================================================================================
// MCU_Init
// ================================================================================================================
void MCU_Init() {
    BCSCTL1 = CALBC1_8MHZ;                                                  // Set DCO
    DCOCTL = CALDCO_8MHZ;
    BCSCTL3 |= LFXT1S_2;                                                    // LFXT1 = VLO
    TBCCTL0 = CCIE;                                                         // TCCR0 interrupt enabled
    TBCCR0 = 12000;                                                         // ~1 second
    TBCTL = TBSSEL_1 + MC_1;                                                // ACLK, upmode
    P3SEL |= 0x30;                                                          // P3.4,5 = USCI_A0 TXD/RXD
    UCA0CTL1 = UCSSEL_2;                                                    // SMCLK
    UCA0BR0 = 0x41;                                                         // 9600 from 8Mhz
    UCA0BR1 = 0x3;
    UCA0MCTL = UCBRS_2;
    UCA0CTL1 &= ~UCSWRST;                                                   // **Initialize USCI state machine**
    IE2 |= UCA0RXIE;                                                        // Enable USCI_A0 RX interrupt
    __enable_interrupt();
}

// ================================================================================================================
// Runs in ISR context. Reading the frame should be done in the application thread not in the ISR thread.
// ================================================================================================================
static uint8_t sCB(linkID_t lid) {
    if (lid) {
        sPeerFrameSem++;
    } else {
        sJoinSem++;
    }
    // leave frame to be read by application.
    return 0;
}
```

```
// ================================================================================================================
// ADC10 interrupt service routine
// ================================================================================================================
#pragma vector=ADC10_VECTOR

__interrupt void ADC10_ISR(void) {
    __bic_SR_register_on_exit(LPM0_bits);                                   // Clear CPUOFF bit from 0(SR)
}

// ================================================================================================================
// Timer B0 interrupt service routine
// ================================================================================================================
#pragma vector=TIMERB0_VECTOR

__interrupt void Timer_B (void) {
    sSelfMeasureSem = 1;
}

// ================================================================================================================
// USCIA interrupt service routine
// ================================================================================================================
#pragma vector=USCIAB0RX_VECTOR

__interrupt void USCI0RX_ISR(void) {
    char rx = UCA0RXBUF;
    if (rx == 'V' || rx == 'v') {
        verboseMode = 1;
    } else if ( rx == 'M' || rx == 'm' )  {
        verboseMode = 0;
    } else if ( rx == 'F' || rx == 'f' )  {
        degCMode = 0;
    } else if ( rx == 'C' || rx == 'c' )  {
        degCMode = 1;
    }
}

// ================================================================================================================
// Función cálculo del punto de rocío
// ================================================================================================================
float Calcula_Rocio(float H, float T) {
    float m;
    float Tn;
    float lnRH;
    float Rocio;
    float TablaLN[11]={-15, -2.302585093, -1.609437912,
                       -1.203972804, -0.916290732,
                       -0.693147181, -0.510825624,
                       -0.356674944, -0.223143551,
                       -0.105360516,  0.0};
    uint8_t i;
    float RH;
    float TermMT;

    // Primero asignamos los parámetros m y Tn
    if (T<0) {                                                              // Para temperaturas negativas
        m=22.46;
        Tn=272.62;
    } else {                                                                // Para temperaturas positivas
        m=17.62;
        Tn=243.12;
    }

    // Calculamos el término (m*T)/(Tn+T)
    TermMT = (m*T)/(Tn+T);                                                  // Calcula el término

    // Ahora se calcula el logaritmo natural con la tabla. Primero se busca los valores de interpolación
    for (i=0;i<11;i++) {                                                    // Recorre la tabla
        RH=(float)i*10;                                                     // Calcula la humedad
        if (H<=RH) break;                                                   // Si la humedad es menor o igual sale del
        bucle
    }

    // En TablaLN[i] se tiene el valor superior y en TablaLN[i-1] el valor inferior.Se realiza una regla de tres
    lnRH = (TablaLN[i]-TablaLN[i-1])*(H-(float)(i-1)*10);                   // Calcula el numerador
    lnRH = lnRH/((float)i*10 - (float)(i-1)*10);                            // Lo divide por el denominador
    lnRH = TablaLN[i-1]+lnRH;                                               // Termina el cálculo

    // Finalmente se calcula el resultado
    Rocio = Tn * (lnRH + TermMT) / (m - lnRH - TermMT);                     // Calcula el Punto de rocío
    return(Rocio);                                                          // Y lo devuelve
}
```

E. Software Code Composer

b. End Device

```
//*****************************************************************************
// THIS PROGRAM IS PROVIDED "AS IS". TI MAKES NO WARRANTIES OR
// REPRESENTATIONS, EITHER EXPRESS, IMPLIED OR STATUTORY,
// INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS
// FOR A PARTICULAR PURPOSE, LACK OF VIRUSES, ACCURACY OR
// COMPLETENESS OF RESPONSES, RESULTS AND LACK OF NEGLIGENCE.
// TI DISCLAIMS ANY WARRANTY OF TITLE, QUIET ENJOYMENT, QUIET
// POSSESSION, AND NON-INFRINGEMENT OF ANY THIRD PARTY
// INTELLECTUAL PROPERTY RIGHTS WITH REGARD TO THE PROGRAM OR
// YOUR USE OF THE PROGRAM.
//
// IN NO EVENT SHALL TI BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
// CONSEQUENTIAL OR INDIRECT DAMAGES, HOWEVER CAUSED, ON ANY
// THEORY OF LIABILITY AND WHETHER OR NOT TI HAS BEEN ADVISED
// OF THE POSSIBILITY OF SUCH DAMAGES, ARISING IN ANY WAY OUT
// OF THIS AGREEMENT, THE PROGRAM, OR YOUR USE OF THE PROGRAM.
// EXCLUDED DAMAGES INCLUDE, BUT ARE NOT LIMITED TO, COST OF
// REMOVAL OR REINSTALLATION, COMPUTER TIME, LABOR COSTS, LOSS
// OF GOODWILL, LOSS OF PROFITS, LOSS OF SAVINGS, OR LOSS OF
// USE OR INTERRUPTION OF BUSINESS. IN NO EVENT WILL TI'S
// AGGREGATE LIABILITY UNDER THIS AGREEMENT OR ARISING OUT OF
// YOUR USE OF THE PROGRAM EXCEED FIVE HUNDRED DOLLARS
// (U.S.$500).k
//
// Unless otherwise stated, the Program written and copyrighted
// by Texas Instruments is distributed as "freeware".  You may,
// only under TI's copyright in the Program, use and modify the
// Program without any charge or restriction.  You may
// distribute to third parties, provided that you transfer a
// copy of this license to the third party and the third party
// agrees to these terms by its first use of the Program. You
// must reproduce the copyright notice and any other legend of
// ownership on each copy or partial copy, of the Program.
//
// You acknowledge and agree that the Program contains
// copyrighted material, trade secrets and other TI proprietary
// information and is protected by copyright laws,
// international copyright treaties, and trade secret laws, as
// well as other intellectual property laws.  To protect TI's
// rights in the Program, you agree not to decompile, reverse
// engineer, disassemble or otherwise translate any object code
// versions of the Program to a human-readable form.  You agree
// that in no event will you alter, remove or destroy any
// copyright notice included in the Program.  TI reserves all
// rights not specifically granted under this license. Except
// as specifically provided herein, nothing in this agreement
// shall be construed as conferring by implication, estoppel,
// or otherwise, upon you, any license or other right under any
// TI patents, copyrights or trade secrets.
//
// You may not use the Program in non-TI devices.
//
//*****************************************************************************
//  eZ430-RF2500 Temperature Sensor End Device using Cymbet Solar Energy
//  Harvesting Board
//
//  Description:
//      This is the End Device software for the eZ430-RF2500-SEH Temperature
//      Sensing demo when hooked up to a Cymbet solar Energy Harvester board.
//
//      The Energy Harvester End Device (EHED) will join the traditional
//      Access Point (AP).  The EHED was optimized to reduce active time
//      especially during start up.
//
//   W. Goh
//   Version 1.5
//   Texas Instruments, Inc
//   March 2009
//   Built with IAR Embedded Workbench Version: 4.11B
//   Built with Code Composer Essentials Version: 3.1 build 3.2.3.6.4
//*****************************************************************************
// Change Log:
//*****************************************************************************
// Version:  1.5 using SimpliciTI ver 1.06
// Comments: Fixed an un-initialized bug inside SimpliciTI
//           Removed unnecessary port initialization in code
// Version:  1.4 using simpliciTI ver 1.06
// Comments: Fixed several bugs.
//           Added blinking LED on power-up
//           Application files now compiles on both IAR and CCE
// Version:  1.3 using Simpliciti ver 1.06
// Comments: Added count battery used count fields
//           Added number_transmits counts up and down
//           Added check if battery charged for 1 hour
// Version:  1.0
// Comments: Inital Release date
//*****************************************************************************
// Modificado por Leyre Morlas Funes
// 2015
//*****************************************************************************

#include "bsp.h"
#include "mrfi.h"
#include "bsp_leds.h"
#include "bsp_buttons.h"
#include "nwk_types.h"
#include "nwk_api.h"
#include "nwk_frame.h"
#include "nwk.h"

#include "msp430x22x4.h"
#include "vlo_rand.h"

#include "SHT.h"
#include "Presion.h"
```

```
#define WakeupPeriod      15000                                   // ~10 sec (=15000/(12000/8))
#define a_d_wakeup_time   4500                                    // ~3 sec
#define TXPeriod          7500                                    // ~5 sec  (=7500/(12000/8))
#define delay_time        500                                     // led delay time
#define debounce_time     750                                     // key debounce

//Timer count for time between transmit
#define sec1              1500                                    // ~1 sec
#define sec2              2610
#define sec5              7500                                    // ~5 sec  (=7500/(12000/8))
#define sec10             15000                                   // ~10 sec
#define sec20             30000                                   // ~20 swec
#define sec40             60000                                   // ~40 sec
#define sec30_2           43000                                   // ~30sec 2 min?
#define sec30_4           50434                                   // ~30sec 4 min?
#define min_15            65000                                   // ~15min
#define one_hour          5400000

#define port_delay        10                                      // 6ms - 1.5 msec

#define status_one        1
#define status_two        2
#define status_three      3
#define status_four       4
#define status_five       5
#define status_six        6

#define timer_state_1     1
#define timer_state_2     2
#define timer_state_3     3
#define timer_state_4     4
#define timer_state_5     5
#define timer_state_6     6
#define timer_state_7     7

#define run_voltage       29                                      // Minimum voltage to execute 2.9V
#define ad_check_voltage  29
#define key_down_count    12                                      // # times to check if button is
                                                                  // still button pressed

#define battery_time_test 174                                     // 3 min count at 10 sec for testing
#define running_on_battery 100                                    // Tells GUI that it is running on
                                                                  // battery
#define xmt_count         400                                     // # max transmit on battery - 400
                                                                  // magic number

#define ON                1
#define OFF               0

unsigned int timer_state;
unsigned char change_mode;
unsigned char ftt_flag;
unsigned int battery_ready = 0;
unsigned int in_delay = 0;
char status = 0;
unsigned int battery_full_flag = 0;
unsigned long battery_full_timer = 0;

unsigned int number_transmits;

void linkTo(void);
void StatusBlink_led1(int BlinkCount);
void StatusBlink_led2(int BlinkCount);
void status_indicator(char status, int status_led);
void delay(unsigned int BlinkCount);
void button_still_pressed(void);
unsigned int get_voltage(void);
void transmit_time_delay(void);
void display_mode(void);
void check_bat_full(void);
void createRandomAddress(void);

// ============================================================================================================================
// Main. Punto de entrada al programa
// ============================================================================================================================
void main (void) {
    addr_t lAddr;
    char *Flash_Addr;
    unsigned int current_voltage;

    WDTCTL = WDTPW + WDTHOLD;                                     // Stop WDT

    P1DIR |= 0x03;                                                // Set P1.0,1 Output
    if( CALBC1_1MHZ == 0xFF && CALDCO_1MHZ == 0xFF &&
    CALBC1_8MHZ == 0xFF && CALDCO_8MHZ == 0xFF ) {                // Do not run if cal values
                                                                  // are erased and set LEDs ON
        P1OUT |= 0x03;                                            // Set P1.0,1 High
        __bis_SR_register(LPM4_bits);                             // Enter LPM4 if Cal missing
    }

    // Blink LED for startup feedback
    P1OUT |= 0x03;                                                // Set P1.0,1 High
    __delay_cycles(10000);
    P1OUT &= ~0x03;                                               // Set P1.0,1 Low

    Flash_Addr = (char *)0x10F0;                                  // RF Address = 0x10F0
    if( Flash_Addr[0] == 0xFF &&                                  // Check if device Address is missing
    Flash_Addr[1] == 0xFF &&
    Flash_Addr[2] == 0xFF &&
    Flash_Addr[3] == 0xFF )  {
        createRandomAddress();                                    // Create Random device address at
    }                                                             // initial startup if missing

    lAddr.addr[0] = Flash_Addr[0];
```

```
    lAddr.addr[1] = Flash_Addr[1];
    lAddr.addr[2] = Flash_Addr[2];
    lAddr.addr[3] = Flash_Addr[3];

    SMPL_Ioctl(IOCTL_OBJ_ADDR, IOCTL_ACT_SET, &lAddr);

    BSP_Init();                                                     // Initialize eZ430 hardware
    BCSCTL3 |= LFXT1S_2;                                            // LFXT1 = ACLK = VLO
    TBCCTL0 = CCIE;                                                 // TBCCR0 interrupt enabled
    TBCCR0 =  WakeupPeriod;                                         // ~10 sec (=15000/(12000/8))
    TBCTL = TBSSEL_1 + MC_1 + ID_3;                                 // ACLK, upmode, Divider = 8

    status = status_four;                                           // Set status to 4

    // Initialize SimpliciTI
    while(SMPL_NO_JOIN == SMPL_Init((uint8_t (*)(linkID_t))0)) {
        __bis_SR_register(LPM3_bits + GIE);                         // LPM3 with interrupts enabled
    }

    // Put radio to sleep
    SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SLEEP, "" );

    ftt_flag = 1;                                                   // first time thru the program flag

    BCSCTL1 = CALBC1_1MHZ;                                          // Set DCO = 1MHz
    DCOCTL = CALDCO_1MHZ;

    // SimpliciTI will change port pin settings as well
    P1DIR = 0xFB;                                                   // P1.2 (button) = input
    P1OUT = 0x04;                                                   // P1.2 pullup
    P1REN |= 0x04;                                                  // P1.2 pullup
    P1IE |= 0x04;                                                   // P1.3 interrupt enabled
    P1IES |= 0x04;                                                  // P1.3 Hi/lo edge
    P1IFG &= ~0x04;                                                 // P1.3 IFG cleared
    P2DIR = 0x2E;                                                   // P2.2 Está configurado como salida
    P2REN |= 0x01;
    P2OUT = 0x05;
    P3DIR |= 0xD0;                                                  // port 3 set after initilization
    P3OUT &= ~0x38;                                                 // set up port 3
    P3REN |= 0x20;                                                  // Enable Pull-Down Res for /Charge
    P4DIR = 0xFF;                                                   // setup port 4. P4.4 Está config como salida
    P4OUT = 0x18;                                                   // Config para interruptor off

    timer_state = timer_state_7;                                    // set timer state to 2 ~ 15 min
    change_mode = 10;                                               // Default GUI display mode set to 10 sec

    TBCTL |= TBCLR;                                                 // Clear TBR counter
    TBCCR0 = a_d_wakeup_time;                                       // set timer to wakeup time ~ 3 sec

    // added to check if battery voltage stable before linking
    current_voltage = get_voltage();                                // get current battery voltage
    if(current_voltage < run_voltage)  {
        current_voltage = 0;
        while (current_voltage < ad_check_voltage) {
            __bis_SR_register(LPM3_bits + GIE);   // Enter LPM3 w/ interrupts
            current_voltage = get_voltage();
        }
    }

    SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_AWAKE, "" );

    // unconditional link to AP which is listening due to successful join.
    linkTo();
}

// =============================================================================================================================
// @fn          linkTo
// =============================================================================================================================
void linkTo(void) {
    linkID_t linkID1;
    uint8_t  msg[10];                                               // Añadimos char eran 7 para seleccionar
    visualización
    unsigned int *tempOffset;                                       // Initialize temperature offset
    tempOffset = (unsigned int *)0x10F4;                            // coefficient
    number_transmits = xmt_count;                                   // Initialize to max transmit #
    unsigned int sensor=0;                                          // Variable selecciona sensor
    unsigned int m=0;
    int TempSHTentero, HumedadEntero;
    float Humedad,TemperaturaH;
    int TempPrEntero, PresionEntero;
    float Presion,TemperaturaP;
    int Tension;

    // keep trying to link... Uses Timer B to wake up periodically
    while (SMPL_SUCCESS != SMPL_Link(&linkID1))  {
        __bis_SR_register(LPM3_bits + GIE);                         // LPM3 with interrupts enabled
    }

    // put radio to sleep once a successfull connection has been established
    SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SLEEP, "" );

    while(1) {
        volatile long temp;
        int degC, volt;
        int results[2];

        // If battery charging, go back to sleep  (if P3.5 = 1, Sleep)
        P3REN &= ~0x20;                                             // turn off pulldown resistor
        delay(port_delay);

        if (sensor==0) {                                            // Sensor=0 implica Temperatura interna
            // Measure Temperature
            ADC10CTL1 = INCH_10 + ADC10DIV_4;                       // Temp Sensor ADC10CLK/5
            ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON + ADC10IE + ADC10SR;
            __delay_cycles(350);                                    // delay to allow reference to settle
            ADC10CTL0 |= ENC + ADC10SC;                             // Sampling and conversion start
```

```
    __bis_SR_register(LPM0_bits + GIE);                                 // LPM0 with interrupts enabled
    results[0] = ADC10MEM;
    ADC10CTL0 &= ~ENC;

    // oC = ((A10/1024)*1500mV)-986mV)*1/3.55mV = A10*423/1024 - 278
    // the temperature is transmitted as an integer where 32.1 = 321
    // hence 4230 instead of 423
    temp = results[0];
    degC = ((temp - 673) * 4230) / 1024;
    if( *tempOffset != 0xFFFF ) {
        degC += *tempOffset;
    }

    // message format,  UB = upper Byte, LB = lower Byte
    // ------------------------------------------------------------------------
    // |degC LB | degC UB |  volt LB | Mode  | # transmit LB |# transmit UB |  ?  |
    // ------------------------------------------------------------------------
    //     0         1          2          3          4               5         6


    // Measure Battery Voltage
    ADC10CTL1 = INCH_11;                                                // AVcc/2
    ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON + ADC10IE + REF2_5V;
    __delay_cycles(350);                                                // delay to allow reference to settle
    ADC10CTL0 |= ENC + ADC10SC;                                         // Sampling and conversion start
    __bis_SR_register(LPM0_bits + GIE);                                 // LPM0 with interrupts enabled
    results[1] = ADC10MEM;
    ADC10CTL0 &= ~ENC;
    ADC10CTL0 &= ~(REFON + ADC10ON);                                    // turn off A/D to save power

    temp = results[1];
    volt = (temp*25)/512;

    msg[2] = volt;
    msg[0] = degC&0xFF;
    msg[1] = (degC>>8)&0xFF;
    msg[7]=1;                                                           // Avisa al AP que es sensor temperatura
    msg[8]=0;
    msg[9]=0;
}

if(sensor==1) {                                                         // Sensor=1  es LDR
    // Encender Led de la LDR
    P2OUT&= 0xFB;                                                       // Sale un 0 que enciende el interruptor
    for (m=0; m<12; m++) {                                              // Retraso de 2 segundos para estabilizar LDR
        delay(250);
    }
    m=0;

    // Medir tensión LDR
    ADC10CTL1 = INCH_0 + ADC10DIV_4;
    ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON + ADC10IE  + REF2_5V;
    __delay_cycles(350);                                                // delay to allow reference to settle
    ADC10CTL0 |= ENC + ADC10SC;                                         // Sampling and conversion start
    __bis_SR_register(LPM0_bits + GIE);                                 // LPM0 with interrupts enabled
    results[0] = ADC10MEM;
    ADC10CTL0 &= ~ENC;

    // Cálculo tensión
    Tension=(results[0]*25)/1024;
    msg[0]=Tension;                                                     // Toma de la primera medida antes de filtrar

    for(m=0;m<10;m++) {
        // Medir tensión LDR Filtrando
        ADC10CTL1 = INCH_0 + ADC10DIV_4;
        ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON + ADC10IE  + REF2_5V;
        __delay_cycles(350);                                            // delay to allow reference to settle
        ADC10CTL0 |= ENC + ADC10SC;                                     // Sampling and conversion start
       __bis_SR_register(LPM0_bits + GIE);                              // LPM0 with interrupts enabled
        results[0] = ADC10MEM;
        ADC10CTL0 &= ~ENC;

        Tension=Tension+((results[0]*25)/1024);
    }

    Tension=(int)(Tension/10.0);                                        // Cálculo tensión filtrada con 10 medidas


    // Measure Battery Voltage
    ADC10CTL1 = INCH_11;                                                // AVcc/2
    ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON + ADC10IE + REF2_5V;
    __delay_cycles(350);                                                // delay to allow reference to settle
    ADC10CTL0 |= ENC + ADC10SC;                                         // Sampling and conversion start
    __bis_SR_register(LPM0_bits + GIE);                                 // LPM0 with interrupts enabled
    results[1] = ADC10MEM;
    ADC10CTL0 &= ~ENC;
    ADC10CTL0 &= ~(REFON + ADC10ON);                                    // turn off A/D to save power

    temp = results[1];
    volt = (temp*25)/512;

    msg[2] = volt;
    msg[8]=Tension;
    msg[1]=0;
    msg[7]=2;                                                           // Avisa al AP que es LDR
    msg[9]=0;

    P2OUT|= 0x04;                                                       // Apagar Led LDR, saca un 1 que deshabilita
    el interruptor
}                                                                       // FIN LDR

if (sensor==2) {                                                        // Sensor=2 es SHT
    P4OUT &= 0xEF;                                                      // Pone un 0 por lo que enciende el sensor

    for(m=0;m<5;m++) {                                                  // Retraso de 0.8 s
        delay(250);
```

```
    }

    m=0;
    SensorInit ();                                          // Inicializa el sensor

    // Operaciones para tener Temp y Humedad en entero a partir de float
    TemperaturaH=Lee_Temperatura();                         // Lee  Temperatura
    TempSHTentero=(int)(TemperaturaH*10.0);                 // Convierte a entero multiplicando por 10
    para coger decimal

    if(TempSHTentero<0) {                                   // En caso de T<0 invierte para obtener
    módulo
        TempSHTentero=-TempSHTentero;
        msg[9]=1;                                           // Marca negativa
    } else {
        msg[9]=0;                                           // Marca positiva
    }

    Humedad=Lee_Humedad(TemperaturaH);                      // Lee  Humedad
    HumedadEntero=(int)(Humedad*10.0);                      // Convierte a entero multiplicando por 10
    para coger decimal

    msg[0] = TempSHTentero&0xFF;
    msg[1] = (TempSHTentero>>8)&0xFF;
    msg[2] = HumedadEntero&0xFF;
    msg[8] = (HumedadEntero>>8)&0xFF;
    msg[7] = 3;                                             // Avisa al AP que es SHT

    P4OUT |= 0x10;                                          //  Apagar SHT, saca un 1 que deshabilita el
    interruptor
}

if (sensor==3) {                                            // Sensor=3 implica sensor de Presión
    P4OUT &= 0XF7;                                          // Pone un 0 por lo que enciende el sensor

    for(m=0;m<5;m++) {                                      // Retraso de 0.8 s
        delay(250);
    }
    m=0;

    // Operaciones para tener Temp y Presion en entero a partir de float
    TemperaturaP = read_Temp();                             // Lee  Temperatura
    TempPrEntero=(int)(TemperaturaP*10.0);                  // Convierte a entero multiplicando por 10
    para coger decimal

    if (TempPrEntero<0) {                                   // En caso de T<0 invierte para obtener
    módulo
        TempPrEntero=-TempPrEntero;
        msg[9]=1;                                           // Marca negativa
    } else {
        msg[9]=0;                                           // Marca positiva
    }

    Presion=read_Presion();                                 // Lee  Presión
    PresionEntero=(int)(Presion*10.0);                      // Convierte a entero multiplicando por 10
    para coger decimal
                                                            // En datasheet dice que va de 10 a 1200

    msg[0] = TempPrEntero&0xFF;
    msg[1] = (TempPrEntero>>8)&0xFF;
    msg[2] = PresionEntero&0XFF;
    msg[8] = (PresionEntero>>8)&0xFF;
    msg[7] = 4;                                             // Avisa al AP que es sensor de Presion
    P4OUT |= 0x08;                                          // Apagar sensor de Presion, saca un 1 que
    deshabilita el interruptor
}

// Wake radio-up
SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_AWAKE, "" );

// If using Solar & not first time through, set battery as ready
if(((P3IN & 0x20) == 0x00) && (ftt_flag == 0) && (battery_ready == 0)) {
    battery_ready = 1;
}

// If battery is ready or first time through, transmit packets
if(battery_ready == 1 || ftt_flag == 1) {
    if((P3IN & 0x20)) {                                     // If P3.5 = 1, then
        msg[3] = (change_mode + running_on_battery);        // +100 is for GUI to know is on battery
        if( number_transmits != 0) number_transmits--;      // # transmit countdown
    } else {                                                // else using solar cells
        msg[3] = change_mode;
        if(number_transmits != xmt_count) {
            number_transmits++ ;
            if(number_transmits >= xmt_count) {             // If max # of transmits achieved,
                number_transmits = xmt_count;               // Reset counter to 400.
            }
        }
        if (battery_full_flag == 1) {                       // If battery is fully charged,
            number_transmits = xmt_count;                   // reset counter to 400
        }
    }

    msg[4] = number_transmits&0xFF;
    msg[5] = (number_transmits>>8)&0xFF;

    // used as a spare bit to indicate on and off
    if (P3IN & 0x01) {
        msg[6] = ON;
    } else {
        msg[6] = OFF;
    }

    // if end of battery, turn off battery
    if(number_transmits == 0) {
        in_delay = 1;
```

```
                while((P3IN & 0x20)) {                                  // Continue sleeping if still on battery
                    __bis_SR_register(LPM4_bits);
                }
                in_delay = 0;
            }

            // Send message
            if (SMPL_SUCCESS == SMPL_Send(linkID1, msg, sizeof(msg))) {
                delay(port_delay);

                if(P3IN & 0x20) {                                       // Using Battery, Blink Red
                    status_indicator(status_one, 1);
                } else {                                                // Using Solar & Blink Green
                    status_indicator(status_one, 2);
                }
            } else {                                                    // Blink both LED if transmission failed
                status_indicator(status_one, 1);
                status_indicator(status_one, 2);
            }
        }

        ftt_flag = 0;                                                   // first time thru the program flag
        check_bat_full();
        status = status_six;
        P3REN |= 0x20;                                                  // Set /Charge pulldown resistor
        SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SLEEP, "" );
        sensor ++;

        if (sensor==4) {
            sensor=0;                                                   // Vuelve a iniciar el ciclo de medidas
            transmit_time_delay();                                      // sleep time between transmits
        } else {
            for(m=0;m<12;m++) {                                         // Retraso de 1 segundo para no saturar
            transmisiones
                delay(125);
            }
            m=0;
        }
    }
}

// ================================================================================================================
// BEGHDR
// NAME:createRandomAddress()
// DESCRIPTION: generate random address
// ================================================================================================================
void createRandomAddress() {
    unsigned int rand, rand2;
    char *Flash_Addr;
    Flash_Addr = (char *)0x10F0;

    do  {
        rand = TI_getRandomIntegerFromVLO();                            // first byte can not be 0x00 of 0xFF
    }

    while ((rand & 0xFF00)==0xFF00 || (rand & 0xFF00)==0x0000);
    rand2 = TI_getRandomIntegerFromVLO();

    BCSCTL1 = CALBC1_1MHZ;                                              // Set DCO to 1MHz
    DCOCTL = CALDCO_1MHZ;
    FCTL2 = FWKEY + FSSEL0 + FN1;                                       // MCLK/3 for Flash Timing Generator
    FCTL3 = FWKEY + LOCKA;                                              // Clear LOCK & LOCKA bits
    FCTL1 = FWKEY + WRT;                                                // Set WRT bit for write operation

    Flash_Addr[0]=(rand>>8) & 0xFF;
    Flash_Addr[1]=rand & 0xFF;
    Flash_Addr[2]=(rand2>>8) & 0xFF;
    Flash_Addr[3]=rand2 & 0xFF;

    FCTL1 = FWKEY;                                                      // Clear WRT bit
    FCTL3 = FWKEY + LOCKA + LOCK;                                       // Set LOCK & LOCKA bit
}

// ================================================================================================================
// BEGHDR
// Function:    void StatusBlink_led1(int BlinkCount)
// DESCRIPTION: Blinks LED 1 - Red based on specified delay
// INPUTS:      BlinkCount
// PROCESSING:  Turns on and off the RED LED with specified blink time
// OUTPUTS:     VOID
// ================================================================================================================
void StatusBlink_led1(int BlinkCount) {
    BSP_TURN_ON_LED1();
    delay(BlinkCount);
    BSP_TURN_OFF_LED1();
}

// ================================================================================================================
// BEGHDR
// Function:    void StatusBlink_led1(int BlinkCount)
// DESCRIPTION: Blinks LED 1 - Green based on specified delay
// INPUTS:      BlinkCount
// PROCESSING:  Turns on and off the Green LED with specified blink time
// OUTPUTS:     VOID
// ================================================================================================================
void StatusBlink_led2(int BlinkCount) {
    BSP_TURN_ON_LED2();
    delay(BlinkCount);
    BSP_TURN_OFF_LED2();
}

// ================================================================================================================
// BEGHDR
// Function:    void delay(unsigned int BlinkCount)
// DESCRIPTION: Creates a low-power delay by entering LPM3 using Timer B.
//              Timer B frequency = VLO/8 = 1500 Hz.
```

```
// INPUTS:      BlinkCount
// PROCESSING:  Delay length of time of BlinkCount
// OUTPUTS:     VOID
// ============================================================================================================================
void delay(unsigned int BlinkCount) {
    int TimerTemp;
    TimerTemp = TBCCR0;                                                     // Save current content of TBCCR0
    TBCCR0 = BlinkCount;                                                    // Set new TBCCR0 delay
    TBCTL |= TBCLR;                                                         // Clear TBR counter
    TBCCTL0 &= ~CCIFG;                                                      // Clear CCIFG Flag
    TBCTL |= MC_1;                                                          // Start Timer B
    __bis_SR_register(LPM3_bits + GIE);                                     // Enter LPM3
    TBCTL &= ~(MC_1);                                                       // Stop Timer B
    TBCCR0 = TimerTemp;
}

// ============================================================================================================================
// BEGHDR
// Function:    void status_indicator(char status , int status_led)
// DESCRIPTION: This can be usefull to blink the LED to indicate where the
//              program is executing for debugging purposes. It blinks the red or
//              green LED the number of times in status. For example, status_five
//              blinks the LED 5 times.
// INPUTS:      status, status_led
// PROCESSING:  Blinks the red or green led the number of times in status and the
//              correct led in status_led
// OUTPUTS:     VOID
// ============================================================================================================================
void status_indicator(char status , int status_led) {
    volatile unsigned int i = 0;

    switch (status) {
        case status_one:
            if (status_led == 1) StatusBlink_led1(15);
            if (status_led == 2) StatusBlink_led2(15);
            break;

        case status_two:
            if (status_led == 1) {
                StatusBlink_led1(15);
                delay(delay_time);
                StatusBlink_led1(15);
            }
            if (status_led==2) {
                StatusBlink_led2(15);
                delay(delay_time);
                StatusBlink_led2(15);
            }
            break;

        case status_three:
            if(status_led == 1) {
                for(i=0 ; i < (status-1) ; i++) {
                    StatusBlink_led1(15);
                    delay(delay_time);
                }
                StatusBlink_led1(15);
            }

            if(status_led == 2) {
                for(i=0 ; i < (status-1) ; i++) {
                    StatusBlink_led2(15);
                    delay(delay_time);
                }
                StatusBlink_led2(15);
            }
            break;

        case status_four:
            if(status_led == 1) {
                for(i=0 ; i < (status-1) ; i++) {
                    StatusBlink_led1(15);
                    delay(delay_time);
                }
                StatusBlink_led1(15);
            }
            if(status_led == 2) {
                for(i=0 ; i < (status-1) ; i++) {
                    StatusBlink_led2(15);
                    delay(delay_time);
                }
                StatusBlink_led2(15);
            }
            break;

        case status_five:
            if(status_led == 1) {
                for(i=0 ; i < (status-1) ; i++) {
                    StatusBlink_led1(15);
                    delay(delay_time);
                }
                StatusBlink_led1(15);
            }
            if(status_led == 2) {
                for(i=0 ; i < (status-1) ; i++) {
                    StatusBlink_led2(15);
                    delay(delay_time);
                }
                StatusBlink_led2(15);
            }
            break;

        default:
            break;
    }
}
```

```
// ============================================================================================================================
// BEGHDR
// Function:    unsigned int get_voltage(void)
// DESCRIPTION: Get battery voltage with A/D
// INPUTS:      void
// PROCESSING:  Read battery voltage from ADC10 and returns the value
// OUTPUTS:     Battery voltage from A/D
// ============================================================================================================================
unsigned int get_voltage(void) {
    unsigned int rt_volts;

    ADC10CTL1 = INCH_11;                                                    // AVcc/2
    ADC10CTL0 = SREF_1 + ADC10SHT_2 + REFON + ADC10ON + ADC10IE + REF2_5V;
    __delay_cycles(250);                                                    // delay to allow reference to settle
    ADC10CTL0 |= ENC + ADC10SC;                                             // Sampling and conversion start
    __bis_SR_register(LPM0_bits + GIE);                                     // LPM0 with interrupts enabled
    rt_volts = ADC10MEM;
    ADC10CTL0 &= ~ENC;
    ADC10CTL0 &= ~(REFON + ADC10ON);                                        // turn off A/D to save power
    rt_volts = (rt_volts*25)/512;
    return (rt_volts);
}

// ============================================================================================================================
// BEGHDR
// NAME:        void transmit_time_delay(void)
// DESCRIPTION: Sets timer to transmit time based on timer_state
// INPUTS:      void
// PROCESSING:  Sets timer to transmit time, for 2 min and 4 min transmit times
//              loop number of 30 sec times to make 2 min and 4 min.
// OUTPUTS:     void
// ============================================================================================================================
void transmit_time_delay(void) {
    volatile unsigned int i = 0;
    in_delay = 1;

    switch (timer_state) {
        case timer_state_1:                                                 // Timer State == 1; 5 Secs
            delay(sec5);
            in_delay = 0;
            battery_full_timer += sec5;
            break;

        case timer_state_2:                                                 // Timer State == 2; 10 Secs
            delay(sec10);
            in_delay = 0;
            battery_full_timer += sec10;
            break;

        case timer_state_3:                                                 // Timer State == 3; 20 Secs
            delay(sec20);
            in_delay = 0;
            battery_full_timer += sec20;
            break;

        case timer_state_4:                                                 // Timer State == 4; 40 Secs
            delay(sec40);
            in_delay = 0;
            battery_full_timer += sec40;
            break;

        case timer_state_5:                                                 // Timer State == 5; 2 mins
            while((i++ < 4) && (timer_state == timer_state_5)) {
                delay(sec30_2);
                in_delay = 0;
                battery_full_timer += sec30_2;
            }
            break;

        case timer_state_6:                                                 // Timer State == 6; 4 mins
            while((i++ < 8) && (timer_state == timer_state_6)) {
                delay(sec30_4);
                in_delay = 0;
                battery_full_timer += sec30_4;
            }

        case timer_state_7:                                                 // Timer State == 7; 15 mins
            while((i++ < 20) && (timer_state == timer_state_7)) {
                delay(min_15);
                in_delay = 0;
                battery_full_timer += min_15;
            }
            break;

        default:
            break;
    }
}

// ============================================================================================================================
// BEGHDR
// NAME:        void display_mode(void)
// DESCRIPTION: Sets mode time to be displayed on the GUI in the voltage stage
//              for the first display
// INPUTS:      void
// PROCESSING:  Sets change_mode number based on timer_state
// OUTPUTS:     void
// ============================================================================================================================
void display_mode(void) {
    switch(timer_state) {
        case timer_state_1:
            change_mode=5;                                                  // ~=5 sec
            break;
        case timer_state_2:
            change_mode=10;                                                 // ~=10 sec
```

```
            break;
        case timer_state_3:
            change_mode=20;                                             // ~=20 sec
            break;
        case timer_state_4:
            change_mode=40;                                             // ~=40 sec
            break;
        case timer_state_5:
            change_mode=2;                                              // ~=2 min
            break;
        case timer_state_6:
            change_mode=4;                                              // ~=4 min
            break;
        default:
            break;
    }
}

// ===================================================================================================================
// BEGHDR
// NAME:        void check_batt_full(void)
// DESCRIPTION: Check if the battery has been charging for 1 hour using
//              battery_full_timer as the counter. battery_full_timer is updated
//              after each delay inside transmit_time_delay() function.
// INPUTS:      void
// PROCESSING:  If battery_full_timer has arrived, switch off batteries.
//              battery_full_flag set if charging time is matched.
// OUTPUTS:     void
// ===================================================================================================================
void check_bat_full(void) {
    // If /Charge is high, No Solar, turn on battery
    if(P3IN & 0x20) {                                                   // Charge=1; battery, Blink Red
        P3OUT &= ~0x10;                                                 // turn on battery
        battery_full_flag = 0;
        battery_full_timer = 0;
    } else {                                                            // Charge=0; Solar charging, blink green
        if(battery_full_timer >= one_hour) {                            // If battery has been charging for an hour,
        turn off battery
            battery_full_flag = 1;
            battery_full_timer = 0;
            P3OUT |= 0x10;                                              // turn off battery
        }
    }
}

// ===================================================================================================================
// BEGHDR
// NAME:        __interrupt void ADC10_ISR(void)
// DESCRIPTION: ADC10 interrupt service routine
// INPUTS:      void
// PROCESSING:  Exit from LPM after interrupt
// OUTPUTS:     void
// ===================================================================================================================
#pragma vector=ADC10_VECTOR

__interrupt void ADC10_ISR(void) {
    __bic_SR_register_on_exit(LPM0_bits);                               // Clear CPUOFF bit from 0(SR)
}

// ===================================================================================================================
// BEGHDR
// NAME:        __interrupt void Timer_B (void)
// DESCRIPTION: Timer B0 interrupt service routine
// INPUTS:      Void
// PROCESSING:  Exit from LPM after interrupt
// OUTPUTS:     Void
// ===================================================================================================================
#pragma vector=TIMERB0_VECTOR

__interrupt void TimerB_ISR (void) {
    __bic_SR_register_on_exit(LPM3_bits);                               // Clear LPM3 bit from 0(SR)
}

// ===================================================================================================================
// BEGHDR
// NAME:        __interrupt void Port_1(void)
// DESCRIPTION: Port 1 interrupt service routine function key
// INPUTS:      void
// PROCESSING:  process the push button to switch to the next time mode
// OUTPUTS:     void
// ===================================================================================================================
#pragma vector=PORT1_VECTOR

__interrupt void Port_1(void) {
    if((P3IN & 0x20)) {                                                 // Charge=1; battery, Blink Red
        BSP_TURN_ON_LED1();
        __delay_cycles(10000);
        BSP_TURN_OFF_LED1();
    }  else {                                                           // Charge=0; Solar, blink green
        BSP_TURN_ON_LED2();
        __delay_cycles(10000);
        BSP_TURN_OFF_LED2();
    }

    // If successful link, change timer state.
    if(status == status_six || status == status_five) {
        if(timer_state >= timer_state_6) {                              // If transmit time is == 6,
            timer_state = timer_state_1;                                // Set timer_state = 1
            display_mode();                                             // Change GUI display time
        } else {
            timer_state++;                                              // Change transmit time state
            display_mode();                                             // Change GUI display time
        }
        if (in_delay) {                                                 // If in transmit delay, exit and
                                                                        // send a new packet with new time
            __bic_SR_register_on_exit(LPM4_bits);                       // Clear LPM3 bit from 0(SR)
```

```
        }
    }
    __delay_cycles(150000);                                 // Debounce software delay

    while(!(P1IN & 0x04));                                  // Loop if button is still pressed
    P1IFG &= ~0x04;                                         // P1.2 IFG cleared key interuped
}
```

E. Software Code Composer

c. Sensor de Humedad

i. SHT.c

```
// ***************************************************************************************************************************
// Driver for SHT71 Sensor by Sensirion
//
// Autor : Stefano Cappello. Modified by Leyre Morlas,2015
// ***************************************************************************************************************************

// ---------------------------------------------------------------------------------------------------------------------------
// Módulos incluídos
// ---------------------------------------------------------------------------------------------------------------------------
#include "msp430x22x4.h"
#include "SHT.h"

// ---------------------------------------------------------------------------------------------------------------------------
//  Definiciones y variables globales
// ---------------------------------------------------------------------------------------------------------------------------
#define data_pin   1
#define clk_pin    0


// ===========================================================================================================================
// Funciones del código
// ===========================================================================================================================

// ---------------------------------------------------------------------------------------------------------------------------
// output_bit
// Asigna como salida un pin y lo pone en alto o en bajo
// ---------------------------------------------------------------------------------------------------------------------------
void output_pin(unsigned int Pin, unsigned int valor) {
    if(Pin==1) {                                                        // Corresponde a data_pin
        P2DIR |=0x02;                                                   // Pone como salida P2.7 (Pone a 1 el pin)
        if(valor==1) {                                                  // Si queremos poner un 1
            P2OUT |= 0x02;                                              // Lo hacemos
        } else {                                                        // Si queremos poner un 0
            P2OUT &= 0xFD;                                              // Lo hacemos
        }
    } else {                                                            // Corresponde a clk_pin
        P4DIR |= 0x20;                                                  // Pone como salida P4.5 (Pone a 1 el pin)
        if (valor==1) {                                                 // Si queremos poner un 1
            P4OUT |= 0x20;                                              // Lo hacemos
        } else {                                                        // Si queremos poner un 0
            P4OUT &= 0xDF;                                              // Lo hacemos
        }
    }
}

// ---------------------------------------------------------------------------------------------------------------------------
// input_pin
// Pone el pin Recibe en el modo de alta impedancia (lectura)
// ---------------------------------------------------------------------------------------------------------------------------
void input_data(void) {
    P2DIR &= 0xFD;                                                      // Pone como entrada P2.7
}

// ---------------------------------------------------------------------------------------------------------------------------
// DelayMS
// Retardo del nº de milisegundos deseado
// ---------------------------------------------------------------------------------------------------------------------------
void DelayMs(long count) {
    volatile long _dcnt;                                                // Contador

    for (_dcnt=0;_dcnt<count;_dcnt++) {                                 // Bucle de conteo
        __delay_cycles(1000);                                           // Cada delay_cycles es 1us aproximadamente
    }
}

// ---------------------------------------------------------------------------------------------------------------------------
// comstart
// Initialize the communication with the sensor
// ---------------------------------------------------------------------------------------------------------------------------
void comstart (void) {
    input_data();                                                       // data high
    output_pin(clk_pin, 0);                                             // clk low
    DelayMs(1);
    output_pin(clk_pin, 1);                                             // clk high
    DelayMs(1);
    output_pin(data_pin, 0);                                            // data low
    DelayMs(1);
    output_pin(clk_pin, 0);                                             // clk low
    DelayMs(1);
    output_pin(clk_pin, 1);                                             // clk high
    DelayMs(1);
    input_data();                                                       // data high
    DelayMs(1);
    output_pin(clk_pin, 0);                                             // clk low
}

// ---------------------------------------------------------------------------------------------------------------------------
// comwrite
// Writes a byte on bus
// ---------------------------------------------------------------------------------------------------------------------------
unsigned char comwrite (unsigned char iobyte) {
    unsigned char i, mask = 0x80;
    unsigned char ack;

    // Shift out command
    DelayMs(1);                                                         // Delay for 1 ms
    for(i=0; i<8; i++) {
        output_pin(clk_pin, 0);                                         // clk low
        if((iobyte & mask) > 0) {
            input_data();                                               // data high if MSB high
        } else {
            output_pin(data_pin, 0);                                    // data low if MSB low
        }
        DelayMs(1);
        output_pin(clk_pin, 1);                                         // clk high
        DelayMs(1);
```

```
        mask = mask >> 1;                                                   // shift to next bit
    }

    // Shift in ack
    output_pin(clk_pin, 0);                                                 // clk low
    DelayMs(1);
    input_data();                                                           // data high

    if (( P2IN & 0x02)== 0x02) {
        ack=1;
        } else {
            ack=0;
        }
        output_pin(clk_pin, 1);                                             // clk high
        DelayMs(1);
        output_pin(clk_pin, 0);                                             // clk low
        return(ack);
    }

// ---------------------------------------------------------------------------------------------------------------------
// comread
// Reads and returns the byte from bus
// ---------------------------------------------------------------------------------------------------------------------
unsigned int comread (void) {
    unsigned char i;
    unsigned int  iobyte = 0;
    const unsigned int mask0 = 0x0000;
    const unsigned int mask1 = 0x0001;

    // shift in MSB data
    for(i=0; i<8; i++) {
        iobyte = iobyte << 1;
        output_pin(clk_pin, 1);                                             // clk high
        DelayMs(1);
        input_data();                                                       // data high

        if ((P2IN & 0x02)==0x02) {
            iobyte |= mask1;                                                // shift in data bit
        } else {
            iobyte |= mask0;
        }

        output_pin(clk_pin, 0);                                             // clk low
        DelayMs(1);
    }

    // send ack 0 bit
    output_pin(data_pin, 0);                                                // data low
    DelayMs(1);

    output_pin(clk_pin, 1);                                                 // clk high
    DelayMs(1);

    output_pin(clk_pin, 0);                                                 // clk low
    DelayMs(1);

    input_data();

    // shift in LSB data
    for(i=0; i<8; i++) {
        iobyte = iobyte << 1;
        output_pin(clk_pin, 1);                                             // clk high
        DelayMs(1);
        input_data();                                                       // data high

        if ((P2IN & 0x02)==0x02) {
            iobyte |= mask1;                                                // shift in data bit
        } else {
            iobyte |= mask0;
        }

        output_pin(clk_pin, 0);                                             // clk low
        DelayMs(1);
    }

    // send ack 1 bit
    input_data();                                                           // data high
    DelayMs(1);
    output_pin(clk_pin, 1);                                                 // clk high
    DelayMs(1);
    output_pin(clk_pin, 0);                                                 // clk low
    return(iobyte);
}


// ---------------------------------------------------------------------------------------------------------------------
// comwait
// Waits until the sensor isn't ready
// ---------------------------------------------------------------------------------------------------------------------
void comwait (void) {
    unsigned int sht_delay;

    input_data();                                                           // data high
    output_pin(clk_pin, 0);                                                 // clk low
    DelayMs(1);

    for(sht_delay=0; sht_delay<300; sht_delay++) {                          // wait for max 300ms
        if ((P2IN & 0X02)== 0x00 ) break;                                   // if sht_data_pin low, SHT75 ready
        DelayMs(1);
    }
}

// ---------------------------------------------------------------------------------------------------------------------
// comreset
// Resets a communication
// ---------------------------------------------------------------------------------------------------------------------
```

```
void comreset (void) {
    unsigned char i;

    input_data();                                                   // data high
    output_pin(clk_pin, 0);                                         // clk low
    DelayMs(1);

    for(i=0; i<9; i++) {
        output_pin(clk_pin, 1);                                     // toggle clk 9 times
        DelayMs(1);
        output_pin(clk_pin, 0);
        DelayMs(1);
    }

    comstart();
}


// ----------------------------------------------------------------------------------------------------------------------
// sht_soft_reset
// Resets a communication for a long time
// ----------------------------------------------------------------------------------------------------------------------
void sht_soft_reset (void) {
    comreset();                                                     // SHT75 communication reset
    comwrite(0x1E);                                                 // send SHT75 reset command
    DelayMs(15);                                                    // Pause 15 ms
}


// ----------------------------------------------------------------------------------------------------------------------
// measuretemp
// Reads the byte of temperature
// ----------------------------------------------------------------------------------------------------------------------
unsigned int measuretemp (void) {
    unsigned char ack;
    unsigned int iobyte;

    comstart();                                                     // alert SHT71
    ack = comwrite(0x03);                                           // send measure temp command and read ack
    status
    if(ack == 1) return 0;

    comwait();                                                      // wait for SHT71 measurement to complete
    iobyte = comread();                                             // read SHT71 temp data
    return iobyte;
}

// ----------------------------------------------------------------------------------------------------------------------
// measurehumid
// Reads the byte of temperature
// ----------------------------------------------------------------------------------------------------------------------
unsigned int measurehumid (void) {
    unsigned char ack;
    unsigned int iobyte;

    comstart();                                                     // alert SHT71
    ack = comwrite(0x05);                                           // send measure RH command and read ack
    status
    if(ack == 1) return 0;

    comwait();                                                      // wait for SHT71 measurement to complete
    iobyte = comread();                                             // read SHT75 temp data
    return iobyte;
}

// ----------------------------------------------------------------------------------------------------------------------
// Lee_Temperatura
// Devuelve el valor de la temperatura en grados centígrados
// ----------------------------------------------------------------------------------------------------------------------

float Lee_Temperatura(void) {
    float Temperatura;
    unsigned int restemp;

    restemp = measuretemp();                                        // measure temp

    // calculate temperature reading
    Temperatura = ((float) restemp * 0.01) - 40.0;
    return Temperatura;
}

// ----------------------------------------------------------------------------------------------------------------------
// Lee_Humedad
// Devuelve el valor de la humedad en porcentaje de humedad relativa
// ----------------------------------------------------------------------------------------------------------------------
float Lee_Humedad(float Temperatura) {
    float Humedad;
    unsigned int reshumid;
    float rhlin,rh;

    reshumid = measurehumid();                                      // measure RH

    // Calculate Real RH reading
    rh = (float) reshumid;
    rhlin = (rh * 0.0405) - (rh * rh * 0.0000028) - 4.0;

    // Calculate True RH reading
    Humedad = (((Temperatura - 25.0) * (0.01 + (0.00008 * rh))) + rhlin);

    if (Humedad>100) {
        Humedad = 100.0;
    }

    return Humedad;
}
```

```
// ------------------------------------------------------------------------------------------------------------------------
// SensorInit                                                |
// Main function- Initialize the sensor                        |
// ------------------------------------------------------------------------------------------------------------------------
void SensorInit (void) {
    comreset();                                                             // reset SHT71
    DelayMs(20);                                                            // delay for power-up
}
```

E. Software Code Composer

c. Sensor de Humedad

ii. SHT.h

```
// ************************************************************************************************************************
// Driver for SHT71 Sensor by Sensirion
//
// Autor : Stefano Cappello. Modified by Leyre Morlas,2015
// ************************************************************************************************************************

// ------------------------------------------------------------------------------------------------------------------------
// Prototipos de las funciones
// ------------------------------------------------------------------------------------------------------------------------
void output_pin(unsigned int Pin, unsigned int valor);
void input_data(void);
void comstart (void);
unsigned char comwrite (unsigned char iobyte);
unsigned int comread (void);
void comwait (void);
void comreset (void);
void sht_soft_reset (void);
unsigned int measuretemp (void);
unsigned int measurehumid (void);
float Lee_Temperatura(void);
float Lee_Humedad(float Temperatura);
void SensorInit (void);
```

E. Software Code Composer

d. Sensor de Presión

i. Presion.c

```
// ***************************************************************************************************************************
// Sensor de presión MS5607
// Código del sensor para medir Presión y Temperatura
//
// Autor : Leyre Morlas + Nota de aplicación: AN520 C-code example for M56xx pressure sensors by Measurement Specialties
// ***************************************************************************************************************************

// ---------------------------------------------------------------------------------------------------------------------------
// Módulos incluídos
// ---------------------------------------------------------------------------------------------------------------------------
#include "msp430x22x4.h"                                                    // Fichero de inclusión del micro
#include "Presion.h"                                                        // Fichero de inclusión del sensor


// ---------------------------------------------------------------------------------------------------------------------------
//  Definiciones y variables globales
// ---------------------------------------------------------------------------------------------------------------------------
#define     Envia_pin       1                                               // Pin de envío
#define     clk             0                                               // Pin de clock

// ...........................................................................................................................
// Comandos del sensor
// ...........................................................................................................................
#define     CMD_RESET       0x1E                                            // Comando Reset
#define     CMD_ADC_READ    0x00                                            // Comando ADC read
#define     CMD_ADC_CONV    0x40                                            // Comando ADC conversion
#define     CMD_ADC_D1      0x00                                            // Conversión ADC D1
#define     CMD_ADC_D2      0x10                                            // Conversión ADC D2
#define     CMD_ADC_256     0x00                                            // ADC OSR=256
#define     CMD_ADC_512     0x02                                            // ADC OSR=512
#define     CMD_ADC_1024    0x04                                            // ADC OSR=1024
#define     CMD_ADC_2048    0x06                                            // ADC OSR=2056
#define     CMD_ADC_4096    0x08                                            // ADC OSR=4096
#define     CMD_PROM_RD     0xA0                                            // Comando Prom read

// ...........................................................................................................................
// Definición de las potencias utilizadas para evitar incluir la librería math.h que ocupa mucho espacio
// ...........................................................................................................................
#define     POW_2_6         64.0                                            // 2^6
#define     POW_2_7         128.0                                           // 2^7
#define     POW_2_8         256.0                                           // 2^8
#define     POW_2_15        32768.0                                         // 2^15
#define     POW_2_16        65536.0                                         // 2^16
#define     POW_2_17        131072.0                                        // 2^17
#define     POW_2_21        2097152.0                                       // 2^21
#define     POW_2_23        8388608.0                                       // 2^23


// ...........................................................................................................................
// Variables globales. Con float minimizamos uso de la memoria frente a double
// ...........................................................................................................................
float           OFF;                                                        // Offset a la temperatura actual
float           SENS;                                                       // Sensibilidad a la temperatura actual
unsigned long   D1;                                                         // Valor ADC value de la conversión

// ===========================================================================================================================
// Funciones del código
// ===========================================================================================================================

// ---------------------------------------------------------------------------------------------------------------------------
// output_bit
// Asigna como salida un pin y lo pone en alto o en bajo
// ---------------------------------------------------------------------------------------------------------------------------
void output_Pin(unsigned int Pin, unsigned int valor) {
    if(Pin==1) {                                                            // Se trata de Envia_pin
        P2DIR |=0x10;                                                       // Pone como salida P2.4 (Pone a 1 el pin)
        if(valor==1) {                                                      // Si lo queremos poner a 1
            P2OUT |= 0x10;                                                  // Lo hacemos
        } else {                                                            // Si lo queremos poner a 0
            P2OUT &= 0xEF;                                                  // Lo hacemos
        }
    } else {                                                                // Se trata de clk_pin
        P2DIR |= 0x08;                                                      // Pone como salida P2.3 (Pone a 1 el pin)
        if(valor==1) {                                                      // Si lo queremos poner a 1
            P2OUT |= 0x08;                                                  // Lo hacemos
        } else {                                                            // Si lo queremos poner a 0
            P2OUT &= 0xF7;                                                  // Lo hacemos
        }
    }
}

// ---------------------------------------------------------------------------------------------------------------------------
// input_pin
// Pone el pin Recibe en el modo de alta impedancia (lectura)
// ---------------------------------------------------------------------------------------------------------------------------
void input_Pin(void) {
    P2DIR &= 0xFD;                                                          // Pone como entrada P2.1
}

// ---------------------------------------------------------------------------------------------------------------------------
// selecciona
// Selecciona el sensor de presión
// ---------------------------------------------------------------------------------------------------------------------------
void selecciona (void) {
    P4DIR |= 0x40;                                                          // Pone como salida P4.6 (pone a 1 el pin)
    P4OUT &= 0xBF;                                                          // Pone un 0  el pin --> Sensor seleccionado
}

// ---------------------------------------------------------------------------------------------------------------------------
// no_selecciona
// De-selecciona el sensor de presión
// ---------------------------------------------------------------------------------------------------------------------------
void no_selecciona (void) {
    P4DIR |= 0x40;                                                          // Pone como salida P4.6 (pone a 1 el pin)
    P4OUT |= 0x40;                                                          // Pone a 1 el pin --> Sensor No seleccionado
}
```

```
// ---------------------------------------------------------------------------------------------------------------------
// spi_write
// Manda el comando por Envia_pin
// ---------------------------------------------------------------------------------------------------------------------
void spi_write (char cmd) {
    char i;                                                             // Contador
    char SPIData;                                                       // Dato a envíar (Auxiliar)

    SPIData = cmd;                                                      // Copia el comando en SPIData

    for (i=0; i<8; i++)  {                                              // Para los 8 bits
        if (SPIData & 0x80) {                                           // Comprueba si es 1
            output_Pin(Envia_pin,1);                                    // Pone en alto la línea SDI
        } else {                                                        // Y si es cero
            output_Pin(Envia_pin,0);                                    // Pone en bajo la línea SDI
        }

        output_Pin(clk,1);                                              // Pone el clock a 1
        output_Pin(clk,0);                                              // Pone el clock a 0

        SPIData <<= 1;                                                  // Rota SPIData para el siguiente bit
    }
}

// ---------------------------------------------------------------------------------------------------------------------
// spi_read
// Lee lo que recibe del sensor
// ---------------------------------------------------------------------------------------------------------------------
unsigned int spi_read (void) {
    char i;                                                             // Contador
    unsigned int  iobyte = 0;                                           // Byte a recibir

    for(i=0; i<8; i++) {                                                // Para los 8 bits
        iobyte  <<= 1;                                                  // Rotamos a la izquierda
        output_Pin(clk,1);                                              // Pone el clock a 1
        input_Pin();                                                    // Pone el pin de data como entrada
        if((P2IN & 0x02) == 0x02) {                                     // Si está a 1
            iobyte ++;                                                  // Pone a 1 el bit 0 de iobyte
        }
        output_Pin(clk,0);                                              // Pone el clock a 0
    }

    return(iobyte);                                                     // Devuelve el byte leído
}

// ---------------------------------------------------------------------------------------------------------------------
// DelayMS
// Retardo del nº de milisegundos deseado
// ---------------------------------------------------------------------------------------------------------------------
void Delay_Ms(long count) {
    volatile long _dcnt;                                                // Contador

    for (_dcnt=0;_dcnt<count;_dcnt++) {                                 // Bucle de conteo
         __delay_cycles(1000);                                          // Cada delay_cycles es 1us aproximadamente
    }
}

// ---------------------------------------------------------------------------------------------------------------------
// cmd_reset
// Aplica comando reset
// ---------------------------------------------------------------------------------------------------------------------
void cmd_reset(void) {
    selecciona();                                                       // Selecciona el sensor
    spi_write(CMD_RESET);                                               // Envía comando para la secuencia de reset
    Delay_Ms(3);                                                        // Tiempo de espera para la secuencia de
    Reset
    no_selecciona();                                                    // De-selecciona el sensor
}

// ---------------------------------------------------------------------------------------------------------------------
// Obtiene el valor de la temperatura (D1) y la Presión (D2) tras la conversión. Resultado 24 bits
// ---------------------------------------------------------------------------------------------------------------------
unsigned long  cmd_adc(char cmd) {
    unsigned int ret;                                                   // Valores de retorno del spi
    unsigned long  temp=0;                                              // Temperatura

    selecciona();                                                       // Selecciona el sensor
    spi_write(CMD_ADC_CONV + cmd);                                      // Envía el comando de conversión

    // .....................................................................................................
    // Comprueba el último Nibble del comando para determinar el tiempo de espera
    // .....................................................................................................
    switch (cmd & 0x0F) {                                               // Test del Nibble bajo
        case CMD_ADC_256 :                                              // Tiempo de 256
            _delay_cycles(250);
            _delay_cycles(250);
            _delay_cycles(250);
            _delay_cycles(150);
            break;
        case CMD_ADC_512 :                                              // Tiempo de 512
            Delay_Ms(3);
            break;
        case CMD_ADC_1024:                                              // Tiempo de 1024
            Delay_Ms(4);
            break;
        case CMD_ADC_2048:                                              // Tiempo de 2048
            Delay_Ms(6);
            break;
        case CMD_ADC_4096:                                              // Tiempo de 4096
            Delay_Ms(10);
            break;
    }
    no_selecciona();                                                    // De-selecciona el sensor

    // .....................................................................................................
```

```
    // Calcula la temperatura que es un valor de 24 bits
    // ..........................................................................................................
    selecciona();                                                   // Selecciona el sensor de nuevo
    spi_write(CMD_ADC_READ);                                        // Envía el comando para leer el resultado
    ret = spi_read();                                               // Lee el valor en ret
    temp = 65536*ret;                                               // Valor de la temperatura (Byte alto-alto)

    ret = spi_read();                                               // Lee el siguiente valor en ret
    temp = temp + 256 * ret;                                        // Acumula el valor de la temperatura (Byte
    alto)

    ret = spi_read();                                               // Lee el siguiente valor en ret
    temp = temp + ret;                                              // Acumula el valor de la temperatura (Byte
    bajo)

    no_selecciona();                                                // De-selecciona el sensor

    return temp;                                                    // Devuelve la Temperatura
}

// ------------------------------------------------------------------------------------------------------------------
// Leer coeficientes de calibración. Resultado en 16 bits
// ------------------------------------------------------------------------------------------------------------------
unsigned int cmd_prom(char coef_num) {
    unsigned int ret;                                               // Valores de retorno del spi
    unsigned int rC=0;                                              // Valor del coeficiente

    selecciona();                                                   // Selecciona el sensor de nuevo

    spi_write(CMD_PROM_RD + coef_num*2);                            // Manda el comando PROM READ
    ret=spi_read();                                                 // Lee el byte alto del coeficiente
    rC=256*ret;                                                     // Lo acumula

    ret=spi_read();                                                 // Lee el byte bajo del coeficiente
    rC=rC+ret;                                                      // Lo acumula

    no_selecciona();                                                // De-selecciona el sensor

    return rC;                                                      // Devuelve el coeficiente
}

// ------------------------------------------------------------------------------------------------------------------
// read_Temp
// Cálculo de la temperatura
// ------------------------------------------------------------------------------------------------------------------
float read_Temp(void) {
    unsigned long  D2;                                              // Valor de D2
    unsigned int Calib[8];                                          // Array de coeficientes
    float T;                                                        // Valor de T
    float dT;                                                       // valor de dT
    char i;                                                         // Contador

    cmd_reset();                                                    // Reset del sensor

    for (i=0;i<8;i++){                                              // Para los ocho coeficientes
        Calib[i]=cmd_prom(i);                                       // Lee los coeficientes de calibración
    }

    D1 = cmd_adc(CMD_ADC_D1 + CMD_ADC_256);                         // Lee el resultado de la Presión
    descompensada
    D2 = cmd_adc(CMD_ADC_D2 + CMD_ADC_4096);                        // Lee el resultado de la Temperatura
    descompensada

    dT = D2 - Calib[5] * POW_2_8;                                   // Cálculo de dT
    OFF = Calib[2] * POW_2_17 + dT * Calib[4] / POW_2_6;            // Cálculo del Offset
    SENS = Calib[1] * POW_2_16 + dT * Calib[3] / POW_2_7;           // Cálculo de la sensibilidad
    T = (2000 + (dT * Calib[6]) / POW_2_23) / 100.0;                // Rango de medida de -40 a +85 grados
    centígrados

    return T;                                                       // Devuelve la temperatura compensada
}

// ------------------------------------------------------------------------------------------------------------------
// read_Presion
// Cálculo de la Presión
// ------------------------------------------------------------------------------------------------------------------

float read_Presion(void) {
    float P;                                                        // Valor de la Presión

    P = (((D1 * SENS) / POW_2_21 - OFF) / POW_2_15) / 100;          // Cálcula la presión
    return P;                                                       // La devuelve
}
```

E. Software Code Composer

d. Sensor de Presión

ii. Presion.h

```
// ************************************************************************************************************************
// Sensor de presión MS5607
// Fichero de inclusión
//
// Autor : Leyre Morlas,2015
// ************************************************************************************************************************

// ------------------------------------------------------------------------------------------------------------------------
// Prototipos de las funciones
// ------------------------------------------------------------------------------------------------------------------------
void output_Pin(unsigned int Pin, unsigned int valor);
void input_Pin(void);
void selecciona (void);
void no_selecciona (void);
void spi_write ( char cmd);
unsigned int spi_read (void);
void Delay_Ms(long count);
void cmd_reset(void);
unsigned long cmd_adc(char cmd) ;
unsigned int cmd_prom(char coef_num);
unsigned char crc4(unsigned int n_prom[]);
float read_Temp(void);
float read_Presion(void);
```

F. Software NetBeans

a. Driver_Serie.java

```java
// ************************************************************************************************************************
// Esta clase se encarga de la recepción de los datos de los sensores por el puerto serie.
// Separa los distintos valores y los interpreta.
//
// Modificada y adaptada por: Leyre Morlas
// ************************************************************************************************************************

package Paquete_Serie;                                                          // Nombre del paquete

// ........................................................................................................................
// Importamos todas las librerías necesarias para el proyecto
// ........................................................................................................................
import com.angryelectron.thingspeak.Channel;
import com.angryelectron.thingspeak.Entry;
import com.angryelectron.thingspeak.ThingSpeakException;
import gnu.io.*;
import java.io.*;
import java.util.*;
import java.text.SimpleDateFormat;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;
import com.mashape.unirest.request.GetRequest;
import org.apache.log4j.Level;
import org.apache.log4j.Logger;
import org.apache.log4j.BasicConfigurator;
import org.junit.BeforeClass;
import org.junit.Test;

// ========================================================================================================================
//                                Clase Driver_Serie
// ========================================================================================================================
public class Driver_Serie implements Runnable, SerialPortEventListener {

    static CommPortIdentifier portId;                                           // portId
    static Enumeration portList;                                                // portList

    InputStream inputStream;                                                    // inputStream
    SerialPort serialPort;                                                      // serialPort
    Thread readThread;                                                          // readThread
    String misDatos = "";                                                       // Cadena global

    static Sensor sensorTemp = new Sensor("Temp");                              // Creamos sensorTemp
    static Sensor sensorLDR = new Sensor("LDR");                                // Creamos sensorLDR
    static Sensor sensorTempH = new Sensor("TempH");                            // Creamos sensorTempH
    static Sensor sensorHum = new Sensor("Hum");                                // Creamos sensorHum
    static Sensor sensorTempP = new Sensor("TempP");                            // Creamos sensorTempP
    static Sensor sensorPresion = new Sensor("Presion");                        // Creamos sensorPresion

    static String LluviaS = "Impredecible";                                     // Asignamos las predicciones
    static String TiempoS ="Impredecible";                                      // Asignamos las predicciones
    static String TminS = "Impredecible";                                       // Asignamos las predicciones
    static String NieblaS = "Impredecible";                                     // Asignamos las predicciones

    static ArrayList<Sensor> listaSensores = new ArrayList <>
        (Arrays.asList(sensorTemp,sensorLDR,sensorHum, sensorPresion));         // Creamos ArrayList

    static Prediccion prediccion = new Prediccion();                            // Creamos prediccion

    // ------------------------------------------------------------------------------------------------------------------------
    // Main. Punto de entrada de la clase
    // ------------------------------------------------------------------------------------------------------------------------
    public static void main(String[] args) {
        Date horaTarde = new Date(System.currentTimeMillis());                  // Creamos horaTarde con la actual
        Date horaMañana = new Date(System.currentTimeMillis());                 // Creamos horaMañana con la actual
        Calendar c = Calendar.getInstance();                                    // Asignamos c
        Calendar cm = Calendar.getInstance();                                   // Asignamos cm
        String NombrePuerto = "";                                               // De momento vacío

        c.setTime(horaTarde);                                                   // Asignamos la hora de la tarde
        cm.setTime(horaMañana);                                                 // Asignamos la hora de la mañana

        if (c.get(Calendar.HOUR_OF_DAY) >= 22) {                                // Si son más de las 22:00
            c.set(Calendar.DAY_OF_YEAR,
            c.get(Calendar.DAY_OF_YEAR) + 1);                                   // Añadimos un día a c
        }

        if (cm.get(Calendar.HOUR_OF_DAY) >= 22) {                               // Si son más de las 22:00
            cm.set(Calendar.DAY_OF_YEAR,
            cm.get(Calendar.DAY_OF_YEAR) + 1);                                  // Añadimos un día a cm
        }

        c.set(Calendar.HOUR_OF_DAY, 20);                                        // Ponemos c a las 20:00
        c.set(Calendar.MINUTE, 0);                                              //
        c.set(Calendar.SECOND, 0);                                              //

        cm.set(Calendar.HOUR_OF_DAY, 5);                                        // Ponemos cm a las 05:00
        cm.set(Calendar.MINUTE, 0);                                             //
        cm.set(Calendar.SECOND, 0);                                             //

        horaTarde = c.getTime();                                                // Reasignamos horaTarde (20:00)
        horaMañana = cm.getTime();                                              // Reasignamos horaMañana (05:00)

        MyTimerTask tareaTresHoras = new MyTimerTask();                         // Creamos la tareaTresHoras
        MyTimerTaskDos tareaTarde = new MyTimerTaskDos();                       // Creamos la tareaTarde
        MyTimerTaskTres tareaMañana = new MyTimerTaskTres();                    // Creamos la tareaMañana

        Timer timer = new Timer(true);                                          // Creamos un timer
        timer.scheduleAtFixedRate(tareaTresHoras,10800000,10800000);           // Real cuando recibe cada 15 min
        timer.scheduleAtFixedRate(tareaTarde,horaTarde,(24*60*60*1000));        // Comienzo a las 20:00
        timer.scheduleAtFixedRate(tareaMañana,horaMañana,(24*60*60*1000));      // Comienzo a las 05:00

        // ....................................................................................................
```

```
    // Una vez creadas las tareas pasamos a configurar el puerto serie utilizado.
    // El puerto de trabajo debe de ser asignado mediante un argumento de la línea de comando
    // ..............................................................................................

    portList = CommPortIdentifier.getPortIdentifiers();                     // Creamos la lista de puertos

    try {                                                                   // Utilizamos un try/catch
        String argumento = args[0];                                         // Tomamos el argumento
        argumento = argumento.toUpperCase();                                // Lo pasamos a mayúsculas

        if((argumento.substring(0,3)).equals("COM")) {                      // Si comienza por 'COM'
            while (portList.hasMoreElements()) {                           // Mientras queden elementos
                portId = (CommPortIdentifier) portList.nextElement();       // Asignamos el siguiente elemento
                if (portId.getPortType() ==
                CommPortIdentifier.PORT_SERIAL ) {                          // Si es un puerto serie
                    if (portId.getName().equals (argumento)) {              // Y se llama como el argumento
                        Driver_Serie reader = new Driver_Serie();           // Creamos el reader
                    }
                }
            }
        } else {
            // ..........................................................................................
            // Si hay algún error ponemos el mensaje correspondiente en la cónsola
            // ..........................................................................................
            System.out.println ("El argumento introducido en el fichero arranque.cmd");
            System.out.print ("no es correcto.");
            System.out.println ("Introduzca como argumento:COM_ ,sustituyendo la barra baja");
            System.out.println ("por el puerto COM utilizado por el AP.");
        }
    } catch (Exception e){
        // ..........................................................................................
        // Si no se ha puesto el puerto COM
        // ..........................................................................................
                System.out.println ("Introduzca en el fichero de arranque el COM utilizado");
    }
}

// -------------------------------------------------------------------------------------------------------
// Driver_Serie
// -------------------------------------------------------------------------------------------------------
public Driver_Serie() {
    try {                                                                   // Abrimos el puerto
        serialPort = (SerialPort) portId.open("DriverSerial", 2000);
    } catch (PortInUseException e) {
    }

    try {
        inputStream = serialPort.getInputStream();                          // Leemos la cadena de entrada
    } catch (IOException e) {
    }

    try {
        serialPort.addEventListener(this);                                  // Añadimos el evento
    } catch (TooManyListenersException e) {
    }

    serialPort.notifyOnDataAvailable(true);                                 // Datos habilitados

    try {                                                                   // Parámetros del puerto serie
        serialPort.setSerialPortParams(19600,
            SerialPort.DATABITS_8,
            SerialPort.STOPBITS_1,
            SerialPort.PARITY_NONE);

    } catch (UnsupportedCommOperationException e) {
    }

    readThread = new Thread(this);                                          // Creamos readThread
    readThread.start();                                                     // Y lo arrancamos
}

// -------------------------------------------------------------------------------------------------------
// Como es Runnable debe de tener el método run() aunque no es necesario que haga nada
// -------------------------------------------------------------------------------------------------------
public void run() {
}

// -------------------------------------------------------------------------------------------------------
// Eventos del puerto serie. Solo procesaremos SerialPortEvent.DATA_AVAILABLE
// -------------------------------------------------------------------------------------------------------
public void serialEvent(SerialPortEvent event) {
    String apiWriteKey = "6TA2OFU7JWFEO7RE";                                // Contraseña del canal asignado

    Channel channel = new Channel(43253, apiWriteKey);                      // Abrimos el canal

    switch (event.getEventType()) {                                         // Comprobamos los eventos
        // Con los siguientes eventos no hacemos nada
        case SerialPortEvent.BI:
        case SerialPortEvent.OE:
        case SerialPortEvent.FE:
        case SerialPortEvent.PE:
        case SerialPortEvent.CD:
        case SerialPortEvent.CTS:
        case SerialPortEvent.DSR:
        case SerialPortEvent.RI:
        case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
            break;

        // Procesamos el evento SerialPortEvent.DATA_AVAILABLE
        case SerialPortEvent.DATA_AVAILABLE:
            byte[] readBuffer = new byte[1];                                // Cadena de un byte
            Entry entry = new Entry();                                      // Creamos entry

            try {
                while (inputStream.available() > 0) {                       // Mientras haya datos en el buffer
                    int numBytes = inputStream.read(readBuffer);            // Leemos el buffer
```

```
                    misDatos += (char) readBuffer[0];                       // Y acumulamos los datos
                }

                if (misDatos.length() >= 54) {                              // Cuando es mayor ya tiene todo
                    System.out.println(misDatos.trim());                    // Imprime la cadena

                    // Datos de Temperatura / Humedad
                    String tempeH = (misDatos.trim()).substring(19, 24);    // Extrae la subcadena
                    double tempH = Double.parseDouble(tempeH);              // Asigna el valor de tempH

                    if (tempH > -40.0) {                                    // Si el dato es válido
                        sensorTempH.setValue(tempH);                        // Asigna el valor de tempH

                        // Datos del sensor LDR
                        String tensionLDRs =
                        (misDatos.trim()).substring(12, 15);                // Extrae la subcadena
                        double tensionLDR =
                        Double.parseDouble(tensionLDRs);                    // Asigna el valor de tensionLDR
                        sensorLDR.setValue(tensionLDR);                     // Asigna el valor
                        entry.setField(2, tensionLDRs);                     // Asigna el campo tensionLDRs

                        // Datos del sensor de Humedad
                        String hume =
                        (misDatos.trim()).substring(28, 33);                // Extrae la subcadena
                        double hum = (Double.parseDouble(hume)) / 10.0;     // Asigna el valor de hum
                        String humed = String.valueOf(hum);                 // Asigna el valor de humed
                        sensorHum.setValue(hum);                            // Asigna el valor de hum
                        entry.setField(4, humed);                           // Asigna el campo humed

                        // Datos de Temperatura / Presión
                        String tempeP =
                        (misDatos.trim()).substring(37, 42);                // Extrae la subcadena
                        double tempP = Double.parseDouble(tempeP);          // Asigna el valor de tempP
                        sensorTempP.setValue(tempP);                        // Asigna el valor

                        // Datos del sensor de Presión
                        String pres = (misDatos.trim()).substring(46, 52);  // Extrae la subcadena
                        double presion = Double.parseDouble(pres);          // Asigna el valor de presion
                        sensorPresion.setValue(presion);                    // Asigna el valor
                        entry.setField(3, pres);                            // Asigna el campo pres

                        // Datos del sensor de Temperatura
                        String tempe1 = (misDatos.trim()).substring(3, 8);  // Extrae la subcadena
                        double temp1 = Double.parseDouble(tempe1);          // Asigna el valor de tempe1
                        double temp = (temp1 + tempH + tempP) / 3.0;        // Calcula la media de las temperaturas
                        String tempe = String.valueOf(temp);                // Asigna la media a tempe
                        sensorTemp.setValue(temp);                          // Asigna el valor
                        entry.setField(1, tempe);                           // Asigna el campo tempe

                        entry.setField(5,LluviaS);                          // Asigna el campo LluviaS
                        entry.setField(6,TiempoS);                          // Asigna el campo TiempoS
                        entry.setField(7,NieblaS);                          // Asigna el campo NieblaS
                        entry.setField(8,TminS);                            // Asigna el campo TminS

                        // Ponemos el mensaje de los datos añadidos
                        System.out.println("Datos añadidos: " + channel.update(entry));

                    } else {
                        // Si los datos no son válidos ponemos el mensaje
                        System.out.println("Datos malos");
                    }

                    misDatos = "";                                          // Borramos la cadena
                }
            } catch (IOException e) {                                       // Si hay error
                System.out.println(e);                                      // Lo imprimimos

            } catch (UnirestException ex) {                                 // Si hay error
                java.util.logging.Logger.getLogger(Driver_Serie.class
                .getName()).log(java.util.logging.Level.SEVERE, null, ex);  // Lo identificamos

            } catch (ThingSpeakException ex) {                              // Si hay error
                java.util.logging.Logger.getLogger(Driver_Serie.class
                .getName()).log(java.util.logging.Level.SEVERE, null, ex);  // Lo identificamos
            }

    }
}

// ------------------------------------------------------------------------------------------------------------------
// Tarea MyTimerTask (Se ejecuta cada tres horas)
// ------------------------------------------------------------------------------------------------------------------
static class MyTimerTask extends TimerTask {                                // Tarea periódica ejecutable
    @Override

    public void run() {
         prediccion.calcularPrediccion(listaSensores);                      // Calcula la predicción
         LluviaS=prediccion.getLluvia();                                    // Asigna la cadena
         TiempoS=prediccion.getTiempo();                                    // Asigna la cadena
         NieblaS=prediccion.getNiebla();                                    // Asigna la cadena
         TminS=prediccion.getTminS();                                       // Asigna la cadena
    }

}

// ------------------------------------------------------------------------------------------------------------------
// Tarea MyTimerTaskDos (Se ejecuta a las 20:00 de cada día)
// ------------------------------------------------------------------------------------------------------------------

static class MyTimerTaskDos extends TimerTask {                             // Tarea periódica ejecutable
    @Override

    public void run() {
        prediccion.calcularPrediccionTarde(listaSensores);                  // Calcula la predicción de la tarde
    }
}
```

```
    // ------------------------------------------------------------------------------------------------------------
    // Tarea MyTimerTaskTres (Se ejecuta a las 05:00 de cada día)
    // ------------------------------------------------------------------------------------------------------------
        static class MyTimerTaskTres extends TimerTask {                        // Tarea periódica ejecutable
        @Override

        public void run() {
            prediccion.setNieblaNum(0);                                         // Asigna NieblaNum valor inicial tras la
            noche
            prediccion.setTmin(300);                                            // Asigna Tmin valor inicial tras la noche
        }
    }
}
```

F. Software NetBeans

b. Sensor.java

```java
// ************************************************************************************************************************
// Esta clase se encarga de la creación de los objetos sensor
// Autor: Leyre Morlas
// ************************************************************************************************************************

package Paquete_Serie;                                                          // Nombre del paquete

// ........................................................................................................................
// Importamos todas las librerías necesarias para el proyecto
// ........................................................................................................................
import java.util.Date;
import java.util.Map;
import java.util.TreeMap;

// ========================================================================================================================
//                                  Clase Sensor
// ========================================================================================================================
public class Sensor {
    private final String ID;                                                    // Creamos la cadena
    private Double value;                                                       // Definimos el valor
    private Date timeStamp;                                                     // Para valores de hora / fecha
    private TreeMap<Date, Double> historico;                                    // Definimos el Histórico

    // ------------------------------------------------------------------------------------------------------------------------
    // Elemento Sensor
    // ------------------------------------------------------------------------------------------------------------------------
    public Sensor(String ID) {
        this.ID = ID;                                                           // Asignamos el ID
        historico = new TreeMap<Date, Double>();                                // Creamos el histórico
    }

    // ------------------------------------------------------------------------------------------------------------------------
    // Devuelve el valor
    // ------------------------------------------------------------------------------------------------------------------------
    public Double getValue() {
        return value;                                                           // Valor devuelto
    }

    // ------------------------------------------------------------------------------------------------------------------------
    // Asigna el valor
    // ------------------------------------------------------------------------------------------------------------------------
    public void setValue(Double value) {                                        // Valor a asignar
        updateHistorico(timeStamp, this.value);                                 // Actualiza el histórico
        this.value = value;                                                     // Asigna el valor
        this.timeStamp = new Date(System.currentTimeMillis());                  // Asigna la hora y fecha
    }

    // ------------------------------------------------------------------------------------------------------------------------
    // Devuelve el ID
    // ------------------------------------------------------------------------------------------------------------------------
    public String getID() {
        return ID;                                                              // ID devuelto
    }

    // ------------------------------------------------------------------------------------------------------------------------
    // Devuelve el Histórico
    // ------------------------------------------------------------------------------------------------------------------------
    public TreeMap getTree() {
        return historico;                                                       // histórico devuelto
    }

    // ------------------------------------------------------------------------------------------------------------------------
    // Devuelve la media
    // ------------------------------------------------------------------------------------------------------------------------
    public double getMedia() {
        double avg = 0;                                                         // Media calculada
        int numData = historico.size();                                         // Nº de datos en el histórico

        for (Map.Entry<Date, Double> entry : historico.entrySet()) {            // Para cada entrada del histórico
            Double value = entry.getValue();                                    // Carga el valor
            avg+=value;                                                         // Y lo suma a la variable avg
        }

        return avg/numData;                                                     // Devuelve la media
    }

    // ------------------------------------------------------------------------------------------------------------------------
    // Sobrecarga de la clase. Devuelve el ID y el valor en forma de cadena
    // ------------------------------------------------------------------------------------------------------------------------
    @Override
    public String toString() {
        return this.ID + ":" + this.value;                                      // Devuelve la cadena
    }

    // ------------------------------------------------------------------------------------------------------------------------
    // Actualiza el Histórico
    // ------------------------------------------------------------------------------------------------------------------------
    private void updateHistorico(Date date, Double sensorValue) {
        if (date == null) {                                                     // Si no hay dato
            return;                                                             // termina
        }

        if (this.historico.size() < 12) {                                       // Si el tamaño es menor de 12
            this.historico.put(date, value);                                    // Carga el valor en el histórico
        } else {                                                                // Si es mayor de 12
            this.historico.remove(this.historico.firstKey());                   // Elimina el primer valor
            this.historico.put(date, value);                                    // y carga el nuevo valor
        }
    }
}
```

F. Software NetBeans

c. Prediccion.java

```java
// ********************************************************************************************************************************
// Esta clase incluye las variables de Predicción y su actualización
//
// Autor: Leyre Morlas
// ********************************************************************************************************************************

package Paquete_Serie;                                                          // Nombre del paquete

// ................................................................................................................
// Importamos todas las librerías necesarias para el proyecto
// ................................................................................................................
import static java.lang.Math.abs;
import java.util.ArrayList;
import static java.lang.Math.log;

// ================================================================================================================================
//                                  Clase Prediccion
// ================================================================================================================================
public class Prediccion {
    private String Tiempo;                                                      // Cadena Tiempo
    private String Niebla;                                                      // Cadena Niebla
    private String Lluvia;                                                      // Cadena Lluvia
    private String TminS = "";                                                  // Cadena TminS (Vacía)

    private double TempAnt;                                                     // Valor de TempAnt
    private double PresAnt;                                                     // Valor de PresAnt
    private double TempAct;                                                     // Valor de TempAct
    private double PresAct;                                                     // Valor de TempAct
    private double HumAct;                                                      // Valor de TempAct
    private double Tmin;                                                        // Valor de Tmin
    private double NieblaNum = 0;                                               // Valor de NieblaNum (0)

    // --------------------------------------------------------------------------------------------------------------------------
    // Prediccion. Pone los valores iniciales
    // --------------------------------------------------------------------------------------------------------------------------
        public Prediccion() {
        this.Tiempo = "";                                                       // Inicialmente vacía
        this.Niebla = "";                                                       // Inicialmente vacía
        this.Lluvia = "";                                                       // Inicialmente vacía
        this.TempAnt = 0;                                                       // Inicialmente nula
        this.PresAnt = 0;                                                       // Inicialmente nula
        this.Tmin = 300;                                                        // Valor Inicial
    }

    // --------------------------------------------------------------------------------------------------------------------------
    // Devuelve la predicción de la Lluvia en string
    // --------------------------------------------------------------------------------------------------------------------------
    public String getLluvia() {
        return Lluvia;
    }

    // --------------------------------------------------------------------------------------------------------------------------
    // Asigna la lluvia en string
    // --------------------------------------------------------------------------------------------------------------------------
    public void setLluvia(String Lluvia) {
        this.Lluvia = Lluvia;
    }

    // --------------------------------------------------------------------------------------------------------------------------
    // Devuelve la predicción de la valoración del Tiempo en string
    // --------------------------------------------------------------------------------------------------------------------------
    public String getTiempo() {
        return Tiempo;
    }

    // --------------------------------------------------------------------------------------------------------------------------
    // Asigna la valoración del tiempo en string
    // --------------------------------------------------------------------------------------------------------------------------
    public void setTiempo(String Tiempo) {
        this.Tiempo = Tiempo;
    }

    // --------------------------------------------------------------------------------------------------------------------------
    // Devuelve la predicción de la temperatura mínima nocturna en string
    // --------------------------------------------------------------------------------------------------------------------------
    public String getTminS() {
        return TminS;
    }

    // --------------------------------------------------------------------------------------------------------------------------
    // Asigna la temperatura mínima nocturna en valor numérico
    // --------------------------------------------------------------------------------------------------------------------------
    public void setTmin(double Tmin) {
        this.Tmin = Tmin;
    }

    // --------------------------------------------------------------------------------------------------------------------------
    // Devuelve la predicción de la temperatura mínima nocturna en valor numérico
    // --------------------------------------------------------------------------------------------------------------------------
    public double getTmin() {
        return Tmin;
    }

    // --------------------------------------------------------------------------------------------------------------------------
    // Devuelve la predicción de la niebla
    // --------------------------------------------------------------------------------------------------------------------------
    public String getNiebla() {
        return Niebla;
    }

    // --------------------------------------------------------------------------------------------------------------------------
    // Devuelve la predicción de la niebla en valor numérico
    // --------------------------------------------------------------------------------------------------------------------------
    public double getNieblaNum() {
        return NieblaNum;
    }
```

```
// ------------------------------------------------------------------------------------------------------------------------
// Asigna la niebla en string
// ------------------------------------------------------------------------------------------------------------------------
public void setNiebla(String Niebla) {
    this.Niebla = Niebla;
}

// ------------------------------------------------------------------------------------------------------------------------
// Asigna la niebla en valor numerico
// ------------------------------------------------------------------------------------------------------------------------
public void setNieblaNum(int NieblaNum) {
    this.NieblaNum = NieblaNum;
}


// ------------------------------------------------------------------------------------------------------------------------
// Calcula la predicción
// ------------------------------------------------------------------------------------------------------------------------
public int calcularPrediccion (ArrayList<Sensor> sensores) {
    for (Sensor sensor:sensores){                                       // Obtención del valor medio
        if (sensor.getID().equals("Temp"))  TempAct = sensor.getMedia();    // en las últimas 3 horas
        if (sensor.getID().equals("Presion")) PresAct = sensor.getMedia();  // de la temperatura, presión
        if (sensor.getID().equals("Hum"))HumAct=sensor.getMedia();          // y humedad
    }

    if ((TempAnt == 0) && (PresAnt==0)) {                               // Tras un Reset o al conectar
        Lluvia = "Impredecible";                                        // la primera vez el dispositivo
        Tiempo = "Impredecible";                                        // no hay valores anteriores
        TempAnt = TempAct;                                              // para hacer predicción
        PresAnt = PresAct;
    } else {                                                            // Si no es así
        if (((PresAct-PresAnt) <= -0.67)
        &&((TempAct-TempAnt)<=-12)) {                                   // Presión y Temp bajan
            Lluvia = "Abundante" ;                                      // Lluvia Abundante
            Tiempo = "Malo";                                            // Tiempo Malo
        } else {
            if(((PresAct-PresAnt) >= 0.67)
            &&((TempAct-TempAnt)>= 12)) {                               // Presión y Temp suben
                Lluvia = "Improbable";                                  // No hay Lluvia
                if (HumAct < 50 ) Tiempo = "Caluroso y Seco";           // Tiempo  caluroso y seco
                else Tiempo = "Bueno";                                  // Tiempo Bueno
            } else {
                if (((PresAct-PresAnt)<= -0.67)
                &&((TempAct-TempAnt)>= 12)) {                           // Presión baja y Temp sube
                    Lluvia = "Impredecible";                            // Lluvia Impredecible
                    Tiempo = "Variable";                                // Tiempo variable
                } else {
                    if(((PresAct-PresAnt)>= 0.67)
                    &&((TempAct-TempAnt)<= -12)){                       // Presión sube y Temp baja
                        Lluvia = "Impredecible";                        // Lluvia Impredecible
                        Tiempo = "Malo";                                // Tiempo variable
                    } else {
                        if((abs(PresAct-PresAnt)<0.67)
                        &&((TempAct-TempAnt)>=12)){                     // Presión estable y Temp sube
                            Lluvia = "Improbable";                      // No hay Lluvia
                            Tiempo = "Bueno";                           // Tiempo Bueno
                        } else {
                            if((abs(PresAct-PresAnt)<0.67)
                            &&((TempAct-TempAnt)<=-12)){                // Presión estable y Temp baja
                                Lluvia = "Probable";                    // Lluvia probable
                                Tiempo = "Malo";                        // Tiempo malo
                            } else {
                                if(((PresAct-PresAnt)>= 0.67)
                                &&(abs(TempAct-TempAnt)< 12)){          // Presión sube y Temp estable
                                    Lluvia = "Improbable";              // No hay Lluvia
                                    Tiempo = "Bueno";                   // Tiempo Bueno
                                } else {
                                    if(((PresAct-PresAnt)<= -0.67)
                                    &&(abs(TempAct-TempAnt)< 12)){      // Presión baja y Temp estable
                                        Lluvia = "Probable";            // Lluvia probable
                                        Tiempo = "Malo";                // Tiempo malo
                                    } else {                            // Presión est y Temp est
                                        Lluvia = "Impredecible";        // Lluvia impredecible
                                        Tiempo = "Variable";            // Tiempo Variable
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }

    if (NieblaNum == 0) {                                               // Si es 0
        Niebla = "Impredecible";                                        // No se puede hacer la predicción
    }

    if(Tmin == 300) {                                                   // Si es 300
        TminS = "Impredecible";                                         // No se puede hacer la predicción
    }
    return 0;
}


// ------------------------------------------------------------------------------------------------------------------------
// Calcula la predicción de la tarde. Permite evaluar la posibilidad de niebla
// ------------------------------------------------------------------------------------------------------------------------
public int calcularPrediccionTarde (ArrayList<Sensor> sensores){
    double TempEv = 0;                                                  // Inicialmente a 0
    double HumEv = 0;                                                   // Inicialmente a 0
    double Rocio = 0;                                                   // Inicialmente a 0

    for (Sensor sensor:sensores){                                       // Para cada uno de los sensores
        if (sensor.getID().equals("Temp"))
```

```
            TempEv  = sensor.getMedia();                                    // Asigna la media de la temperatura
            if (sensor.getID().equals("Hum"))
            HumEv  = sensor.getMedia();                                     // Asigna la media de la humedad
        }

        if (TempEv > 0){                                                    // Si la temperatura es positiva
            double num = log(HumEv/100)+((17.62*TempEv)/(243.12+TempEv));   // se aplica esta función para el
            double den = 17.62 - log(HumEv/100) -
            ((17.62*TempEv)/(243.12+TempEv));                               // cálculo del punto de rocío
            Rocio = 243.12 *(num/den);                                      // Valor para TempEv>0
        } else {
            double num = log(HumEv/100)+((22.46*TempEv)/(272.62+TempEv));   // Si la temperatura es negativa
            double den = 22.46 - log(HumEv/100) -
            ((22.46*TempEv)/(272.62+TempEv));                               // se aplican otras constantes
            Rocio = 272.62 *(num/den);                                      // Valor para TempEv<=0
        }

        // ..............................................................................................
        // Si la temperatura al atardecer es similar al punto de rocío al atardecer y la humedad relativa es máxima
        // la niebla por la noche será abundante. En otros casos será impredecible o se formarán bancos de niebla
        // ..............................................................................................
        if (abs(TempEv - Rocio)<2) {                                        // Temperaturas cercanas
            if (abs(HumEv - 98)<2) {                                        // Humedad prácticamente 100%
                Niebla = "Abundante";                                       // Niebla abundante
                NieblaNum = 2;                                              // Valor numérico de la niebla
            } else {                                                        // Si la humedad relativa no es máxima
                Niebla = "Bancos de Niebla";                                // sólo se formarán bancos de niebla
                NieblaNum= 1;                                               // Valor numérico de los bancos de niebla
            }
        } else {                                                            // Si no son temperaturas similares
            Niebla = "Impredecible";                                        // No se puede predecir la niebla
            NieblaNum = 0;                                                  // Valor numérico de la niebla impredecible
        }

        // ..............................................................................................
        // Si la humedad relativa al atardecer es del 50%, la temperatura mínima por la noche coincide con el punto de
        // rocío al atardecer
        // ..............................................................................................
        if (abs(HumEv-50) < 2) Tmin = Rocio;                                // Asigna Tmin

        // ..............................................................................................
        // Si podemos estimar la temperatura mínima por la noche lo hacemos, en caso contrario es impredecible
        // ..............................................................................................
        if (Tmin < 300) {
            TminS = String.valueOf(Tmin);                                   // Temperatura mínima estimada
        } else {
            TminS = "Impredecible";                                         // Temperatura mínima impredecible
        }
        return 0;                                                           // La función devuelve 0
    }
}
```

G. Software pluggins ThingSpeak

a. Valores Actuales

HTML

```
<html>
  <head>

  <title>Google Gauge - ThingSpeak</title>

  %%PLUGIN_CSS%%
  %%PLUGIN_JAVASCRIPT%%

  </head>

  <body>
    <div id="container">
        <div id="table_div"></div>
      </div>
  </body>
</html>
```

CCS

```
<style type="text/css">
  body { background-color: #ddd; }
  #container { height:100%; width:100%; display: table; }
  #table_div { width: 300px; margin: 3 auto; }

  .bold-green-font {
    font-weight: bold;
    color: green;
  }

  .bold-font {
    font-weight: bold;
  }

  .right-text {
    text-align: center;
  }

  .large-font {
    font-size: 20px;
  }
  .medio-font {
    font-size: 15px;
  }
```

```
.italic-darkblue-font {
  font-style: italic;
  color: darkblue;
}

.italic-purple-font {
  font-style: italic;
  color: purple;
}

.underline-blue-font {
  text-decoration: underline;
  color: blue;
}

.gold-border {
  border: 3px solid gold;
}
.blue-border {
  border: 3px solid blue;
}
.blueL-border {
  border: 1px solid blue;
}
.deeppink-border {
  border: 3px solid deeppink;
}
.borderGrande{
  border: 2px solid
  }
  .borderFino{
  border: 1px solid
  }
.orange-background {
  background-color: orange;
}

.orchid-background {
  background-color: blue;
}
.fondoAzul{
   background-color:blue;
  }

.beige-background {
  background-color: beige;
}
```

```
</style>

-----------------------------------------------------------------------------------------------------------------------------------------
                                                                    JavaScript
-----------------------------------------------------------------------------------------------------------------------------------------

<script type='text/javascript' src='https://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js'></script>
<script type='text/javascript' src='https://www.google.com/jsapi'></script>
<script type='text/javascript'>
 // set your channel id here
  var channel_id = 43253;
 // set your channel's read api key here if necessary
  var api_key = '6TA2OFU7JWFEO7RE';
 // global variables
  var tabla, data;
 // variable for the data point
  var temp,lum,pres,hum;
 // load the google gauge visualization
  google.load('visualization', '1', {packages:['table']});
  google.setOnLoadCallback(DrawTable);

  var cssClassNames = {
       'headerCell': 'borderGrande right-text large-font fondoAzul',
     'tableCell': ' borderFino right-text medio-font bold-font'};

  // display the data
   function displayData() {

    data.setCell(0, 0,'Temperatura');
    data.setCell(0, 1, temp+' ºC');
    data.setCell(1, 0,'Iluminación');
    data.setCell(1, 1, lum+' lux');
    data.setCell(2, 0,'Presión');
    data.setCell(2, 1, pres+' mbar');
    data.setCell(3, 0,'Humedad');
    data.setCell(3, 1, hum+' %');
    tabla.draw(data, options);
  }

  // load the data
  function loadData() {
  // get the data from thingspeak
   $.getJSON('https://api.thingspeak.com/channels/' + channel_id + '/feed/last.json?api_key=' + api_key, function(data) {
  // get the data point
    temp = data.field1;
    tempdos= (Math.round(parseFloat(temp*10)))/10;
    temp = tempdos.toString();
    lum = data.field2;
```

```
    ResLDR= Math.round((374*parseFloat(lum))/(3.3-parseFloat(lum)));
    if(ResLDR > 800) LUZ= Math.round(Math.E^(16.79 - (1.49 * Math.log(ResLDR))));
    else LUZ = Math.round(119379.73 - (135.135 * ResLDR));
    lum = LUZ.toString();
    pres = data.field3;
    presdos = parseFloat(pres);
    pres = presdos.toString();
    hum = data.field4;
    humdos= (Math.round(parseFloat(hum)*10))/10;
    hum = humdos.toString();

    displayData();  });
  }

  // initialize the chart
  function DrawTable() {

    data = new google.visualization.DataTable();
    data.addColumn('string', 'Variable');
    data.addColumn('string', 'Valor');
    data.addRows(4);

    // chart = new google.visualization.Gauge(document.getElementById('table_div'));
    tabla = new google.visualization.Table(document.getElementById('table_div'));
     options = {width: 300, height: 210,'cssClassNames': cssClassNames};
    loadData();

    // load new data every 15 seconds
    setInterval('loadData()', 15000);
  }

</script>
```

G. Software plugggins ThingSpeak

b. Predicción

```
-----------------------------------------------------------------------------------------------------------------------------------
                                                        HTML
-----------------------------------------------------------------------------------------------------------------------------------
<html>
 <head>

 <title>Google Gauge - ThingSpeak</title>

 %%PLUGIN_CSS%%
 %%PLUGIN_JAVASCRIPT%%

 </head>

 <body>
   <div id="container">
       <div id="table_div"></div>
     </div>
 </body>
</html>
-----------------------------------------------------------------------------------------------------------------------------------
                                                        CCS
-----------------------------------------------------------------------------------------------------------------------------------
<style type="text/css">
 body { background-color: #ddd; }
 #container { height: 100%; width: 100%; display: table; }
 #table_div { width: 300px; margin: 3 auto; }

 .bold-green-font {
   font-weight: bold;
   color: green;
 }

 .bold-font {
   font-weight: bold;
 }

 .right-text {
   text-align: center;
 }

 .large-font {
   font-size: 20px;
 }
 .medio-font {
   font-size: 15px;
 }

 .italic-darkblue-font {
```

```
    font-style: italic;
    color: darkblue;
  }

  .italic-purple-font {
    font-style: italic;
    color: purple;
  }

  .underline-blue-font {
    text-decoration: underline;
    color: blue;
  }

  .gold-border {
    border: 3px solid gold;
  }
  .blue-border {
    border: 3px solid blue;
  }
 .blueL-border {
    border: 1px solid blue;
  }
  .deeppink-border {
    border: 3px solid deeppink;
  }
  .borderGrande{
    border: 2px solid
    }
    .borderFino{
    border: 1px solid
    }
  .orange-background {
    background-color: orange;
  }

  .orchid-background {
    background-color: blue;
  }
  .fondoAzul{
     background-color:blue;
    }

  .beige-background {
    background-color: beige;
  }

</style>
```

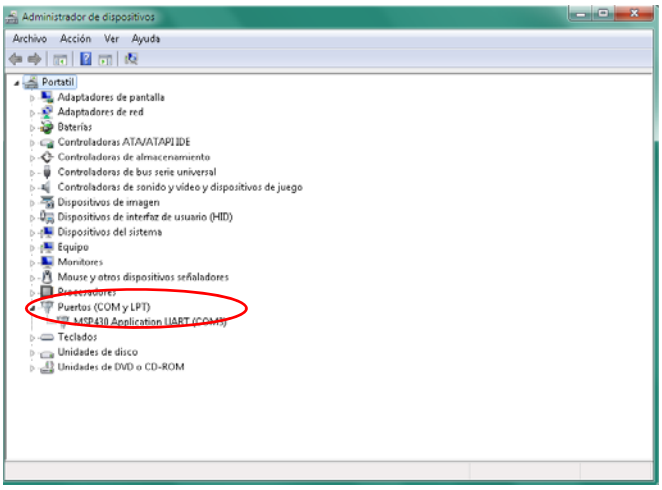
```
 ------------------------------------------------------------------------------------------------------------------------------------
                                                                    JavaScript
 ------------------------------------------------------------------------------------------------------------------------------------
 <script type='text/javascript' src='https://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js'></script>
<script type='text/javascript' src='https://www.google.com/jsapi'></script>
<script type='text/javascript'>
 // set your channel id here
  var channel_id = 43253;
 // set your channel's read api key here if necessary
  var api_key = '6TA2OFU7JWFEO7RE';
 // global variables
  var tabla, data;
 // variable for the data point
  var Lluvia,Tiempo,Niebla,Tmin;
 // load the google gauge visualization
  google.load('visualization', '1', {packages:['table']});
  google.setOnLoadCallback(DrawTable);

  var cssClassNames = {
     'headerCell': 'borderGrande right-text large-font fondoAzul',
     'tableCell': ' borderFino right-text medio-font bold-font'
  };


  // display the data
   function displayData() {
    data.setCell(0, 0,'Predicción Lluvia');
    data.setCell(0, 1, Lluvia );
    data.setCell(1, 0,'Predicción Tiempo');
    data.setCell(1, 1, Tiempo);
    data.setCell(2, 0,'Predicción Niebla');
    data.setCell(2, 1, Niebla);
    data.setCell(3, 0,'Temperatura mínima durante la Noche');
    data.setCell(3, 1, Tmin);

    tabla.draw(data, options);
  }

  // load the data
  function loadData() {
  // get the data from thingspeak
   $.getJSON('https://api.thingspeak.com/channels/' + channel_id + '/feed/last.json?api_key=' + api_key, function(data) {
  // get the data point
    Lluvia = data.field5;
    Tiempo = data.field6;
    Niebla = data.field7;
    Tmin = data.field8;
```

```
    displayData();  });
  }

  // initialize the chart
  function DrawTable() {

    data = new google.visualization.DataTable();
    data.addColumn('string', 'Predicción');
    data.addColumn('string', 'Resultado');
    data.addRows(4);

    // chart = new google.visualization.Gauge(document.getElementById('table_div'));
    tabla = new google.visualization.Table(document.getElementById('table_div'));
    options = {width: 300, height: 200,'cssClassNames': cssClassNames};
    loadData();

    // load new data every 15 seconds
    setInterval('loadData()', 15000);
  }

</script>
```

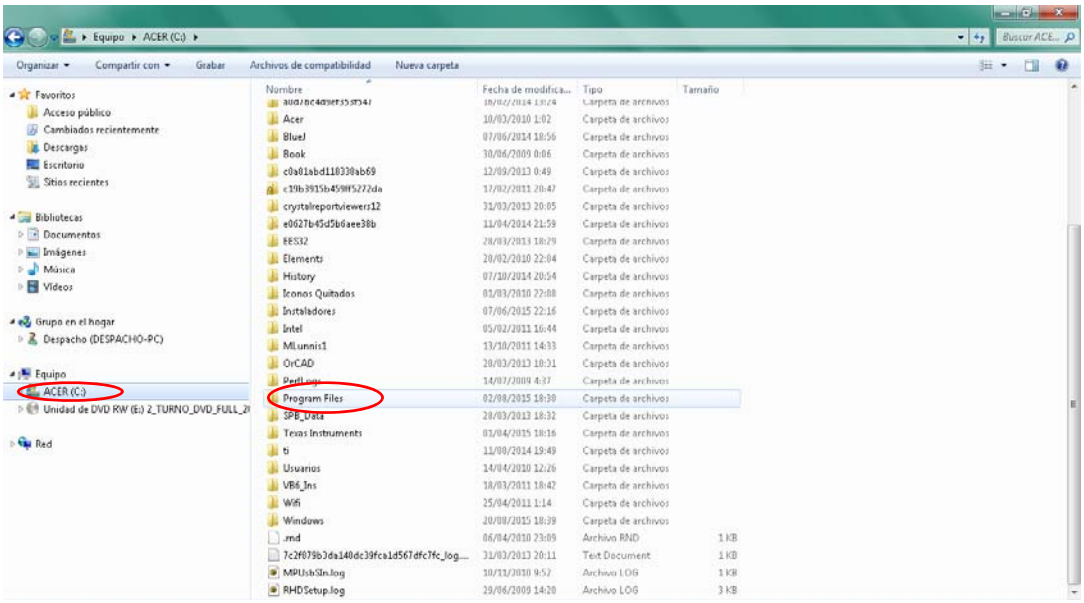
H. Lanzamiento de la ejecución

En este anexo se describen los pasos a realizar para lanzar la aplicación cuando el usuario recibe un fichero comprimido en formato zip denominado, por ejemplo, Estacion_metereologica (caso de la presentación) para su correcta ejecución:

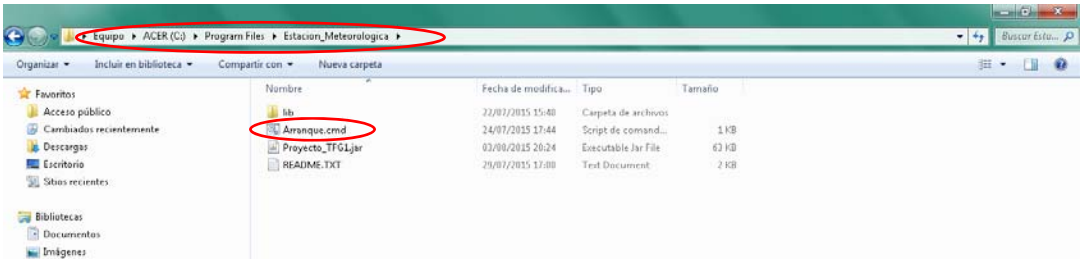
1. Conectar el dispositivo Access Point a un puerto USB del ordenador habiendo instalado los controladores necesarios de la placa de Texas Instrument.
2. Buscar el *COM PORT* al cual ha sido conectado.

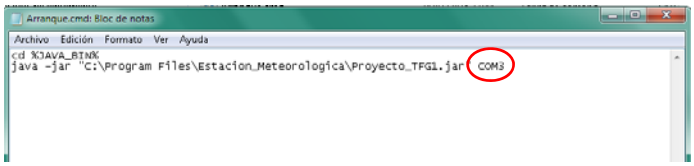


3. Extraer el fichero zip "Estacion_Metereologica" en C:Program Files.

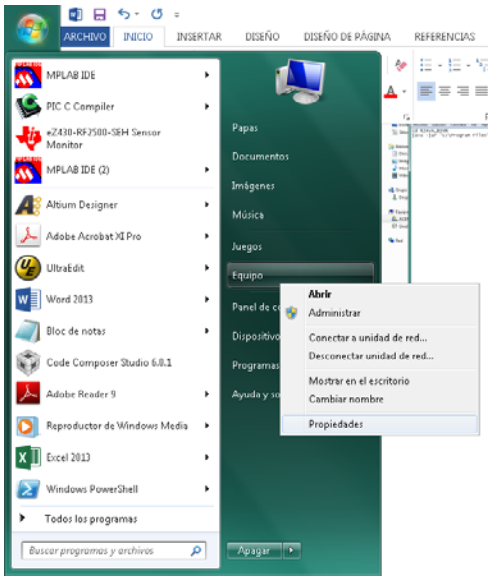


4. Editar el fichero Arranque.cmd modificando el número *COM PORT* al cual se ha conectado el dispositivo AP, por defecto está puesto el COM3 a modo de ejemplo.

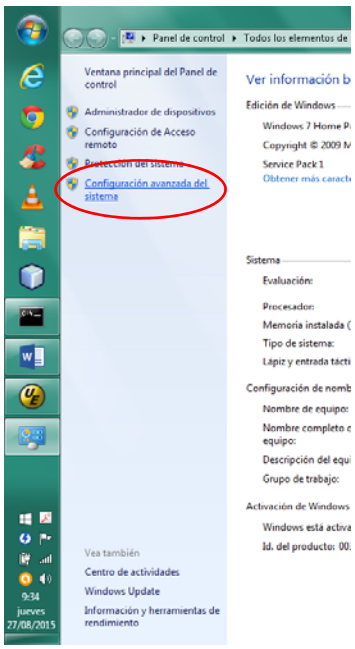




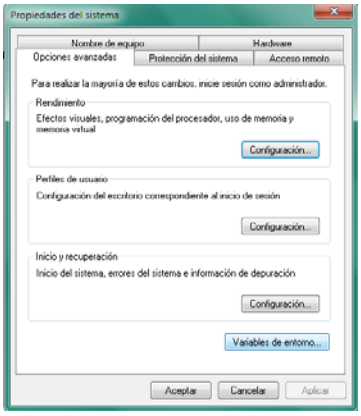
5. Asegurarse de que se tiene creada la variable de entorno JAVA_BIN:
   - Ir a equipo y hacer click con el botón derecho y seleccionar propiedades



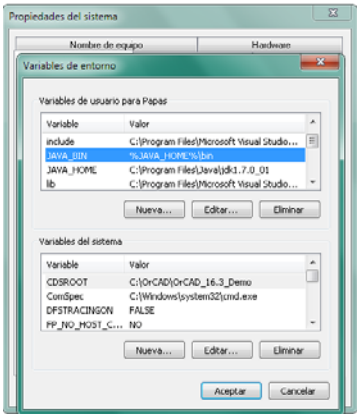
   - En la parte izquierda de la ventana emergente seleccionar configuración avanzada del sistema.



- En la pestaña opciones avanzadas, seleccionar en la parte de abajo: variables de entorno.



- Buscar variables *JAVA_BIN* y *JAVA_HOME.*



Si no están creadas visitar el siguiente enlace para su creación:
enlace: http://www.letkowski. us/simple/java/envar/winxp/

ADVERTENCIA: En la creación de JAVA_HOME hay que copiar la ruta de acceso propia de cada ordenador

ejemplo C:\Program Files\Java\jdk1.7.0_01

6. Copiar el fichero Arranque.cmd en la carpeta Inicio para una ejecución automática de la aplicación al arrancar el ordenador.

ADVERTENCIA: En caso de que la ruta de la carpeta Estacion_Metereologica añada algo en Program Files, modificar el fichero arranque añadiendo al lado de Program Files lo añadido.

ejemplo C:\Program Files(x86)\Estacion_Metereologica

I. Estudio de la incorporación de la cámara OV7670

La cámara OV7670 de Omnivisión incorpora:

1. Funciones automáticas para el control de la imagen, es decir permite un pre-procesamiento de la imagen gracias a un DSP integrado:

AWB → Auto balance de blanco

AE →Exposición automática

AGC → Control automático de la ganancia

ABF →Filtro de banda automático

ABLC→ Calibración automática del nivel de negro

2. Funciones para el control de la calidad de la imagen como la saturación del color, la nitidez, la anti decoloración etc.

Reducción de ruido y corrección de defectos

3. Posibilidad de trabajar con diferentes formatos de trama:

- VGA (640 x 480)
- QVGA (320 x 240)
- CIF (352 x 240)
- QCIF (176 x 144)
- Escalamiento Mnual

4. Posibilidad de trabajar con diferentes formatos de pixel:

- **Monocromo**: Cada pixel se codifica con 8 bits de manera que en función del número que se almacena representa de blanco (0) a negro (255).
- **RGB**: Este tipo de formato se basa en que todos los colores pueden descomponerse en los tres colores primarios (rojo , verde y azul).

La cámara OV7670 permite trabajar con diferentes formatos RGB en función del número de bits que se requieren para almacenar la información de esta intensidad:
RGB565→Cada pixel se compone de 16 bits donde la intensidad del color rojo se representa con cinco bits, la del verde con seis y la del azul con cinco. La intensidad se codifica como cero para la ausencia de ella y el máximo para la máxima intensidad.
Del mismo modo puede utilizarse el formato RGB555 y RGB444.

YCbCr→Formato que codifica la información RGB. Y representa la iluminación (Cantidad de luz blanca en un color), Cb y Cr son los componentes de crominancia, diferencia de azul (Cb) y rojo (Cr). En el caso de la cámara OV7670 se utiliza el formato YCbCr422 de manera que la información se almacena en palabras (cuatro bytes).

	Byte 0	Byte 1	Byte 2	Byte 3
Word 0	Cb0	Y0	Cr0	Y1
Word 1	Cb2	Y2	Cr2	Y3
Word 2	Cb4	Y4	Cr4	Y5

Es decir, en la línea de datos la información llega de la forma que se representa en la siguiente tabla:

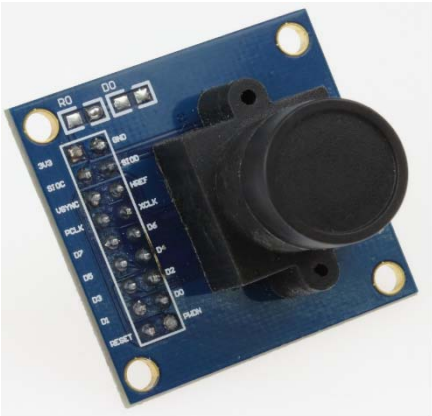
N	1º	2º	3º	4º	5º	6º	7º	8º	…
Byte	Cb0	Y0	Cr0	Y1	Cb2	Y2	Cr2	Y3	…

Y los pixeles reales obtenidos serían:

Pixel 0	Y0 Cb0 Cr0
Pixel 1	Y1 Cb0 Cr0
Pixel 2	Y2 Cb2 Cr2
Pixel 3	Y3 Cb2 Cr2
Pixel 4	Y4 Cb4 Cr4
Pixel 5	Y5 Cb4 Cr4

Se puede observar que dos píxeles consecutivos comparten Cb y Cr es decir, se podría decir que cada pixel en promedio viene representado por dos bytes por lo que el requerimiento de memoria es inferior.

La interfaz física de la cámara se muestra en la figura y en ella se observan catorce pines:



3V3 → Alimentación del módulo

GNG → Masa del módulo

SDIOC→ Reloj de SSCB, similar al protocolo I2C

VSYNC→ Sincronización vertical

HREF→ Sincronización horizontal

SDIOD → Datos de SSCB, similar al protocolo I2C

PCLK → Por defecto es la señal introducida por el pin XCLK aunque es configurable a través de las líneas de control SCCB, reloj de la emisión de cada pixel. Como ya se ha comentado, la velocidad máxima (independientemente del formato de trama elegido) es de 30fps y esto sucede para un PCLK=24MHz.

XCLK→ Pin para incorporar un cristal de cuarzo entre 10 y 48 MHZ para la transmisión de la señal PCLK a 30fps, reloj del sistema

D7-D0→ Transmisión de los datos, información de los pixeles

RESET→ Reinicio de módulo

PWDN→ Apagado del módulo

Una vez definida la interfaz, se escoge el formato QCIF para la imagen capturada, ya que es el formato con menor requerimiento de memoria. A pesar de ello el número de bytes resultantes es demasiado elevado para el dispositivo End Device.

QCIF (176x 144):

1 pixel = 16 bits → QCIF =405504 bits → 50688 Bytes, por lo tanto es necesario la incorporación de una memoria externa de 64k para su almacenaje o el uso de otro microcontrolador con la memoria suficiente como para almacenarlo.

Modo de funcionamiento:

El dispositivo End Device es el encargado de comunicarse con el módulo a través de las líneas SCCB para lanzar lanzar la orden de captura de la imagen. Una vez realizada la imagen los pixeles son almacenados en una memoria externa. Esta memoria se conecta al dispositivo End Device mediante interfaz serie y éste envía por radiofrecuencia de diez en diez bytes la información al Access Point.

Como se puede deducir, se requieren al menos cuatro pines del End Device para poder trabajar con el módulo de la cámara además de los necesarios para su alimentación. Además también es importante detectar que si se utiliza el dispositivo End Device que está conectado a los sensores, el envío de datos es muy lento ya que cada 15 segundos envía 10 bytes.

Esto se traduce en que los 50688 bytes tardarían en enviarse un total de aproximadamente 200 días, algo totalmente desorbitado. Es por ello necesario el uso de otro dispositivo End Device que se encargue exclusivamente de la comunicación con el módulo y que, independientemente al otro dispositivo End Device, envíe, a la velocidad máxima de 10 bytes por segundo, información sobre la imagen.

Esto permite reducir el tiempo de adquisición de la imagen por parte del dispositivo Access Point a una hora y media, algo razonable para la aplicación que se está tratando.

La adición de un dispositivo End Device a la red creada por el protocolo SimpliciTI es perfectamente válida por lo que no sería difícil implementarlo en un producto real.

Éste dispositivo se alimentaría con baterías recargables a partir de placas solares. Esto es posible ya que el consumo se produce al tomar la imagen (máximo de 20 mA) pero después durante el envío de los pixeles la cámara permanece desconectada y el consumo restante es únicamente el de transmisión.