Universidad Zaragoza

1542

Trabajo Fin de Grado

Dispositivo registrador de temperatura y humedad con comunicación Bluetooth

Autor/es

Paloma Royo Cascajares

Director/es

Bonifacio Martín del Brío

Departamento de Ingeniería electrónica y comunicaciones
Escuela de ingeniería y Arquitectura de Zaragoza
Septiembre 2016

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. Paloma Royo Cascajares,

con nº de DNI 25200289V 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) Electrónica y Automática, (Título del Trabajo) Dispositivo registrador de temperatura y humedad con comunicación Bluetooth,

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

Zaragoza, 1 de SEPTIEMBRE 2016

Fdo: [illegible]

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

AGRADECIMIENTOS

A mis PADRES y a mi HERMANA por todo su apoyo.

Gracias.

Resumen

Este proyecto nace con el objetivo de desarrollar un dispositivo registrador de datos de temperatura y humedad de bajo consumo, reducido tamaño y con comunicación Bluetooth.

En el mercado hay gran cantidad de dispositivos registradores de datos sin embargo es escasa la variedad de dataloggers inalámbricos de ahí la motivación de este trabajo. El Bluetooth nos ofrece una comunicación inalámbrica de bajo coste y accesible a todo el mundo hoy en día.

Siguiendo los objetivos citados anteriormente se ha desarrollado un prototipo basado en un microcontrolador de bajo consumo que realiza mediciones de temperatura y humedad cada cinco minutos, siendo este tiempo configurable por el usuario, guarda los datos en memoria y realiza la transmisión de datos por Bluetooth a un dispositivo móvil, cuándo el usuario lo desee, mediante un pulsador externo. La aplicación móvil desarrollada permite la visualización de todos los datos almacenados.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. Introducción

1.1 Descripción general

Un registrador de datos, conocido con el término inglés *datalogger*, es un dispositivo electrónico autónomo que mide y almacena distintos parámetros físicos o eléctricos durante un periodo de tiempo. Dependiendo de los sensores integrados en el registrador las medidas pueden incluir: temperatura, humedad, corriente y voltaje AC/DC, luminosidad, concentración de gases (CO,CO2,O2), temperatura del agua, velocidad y dirección del viento… [1] Un ejemplo típico de aplicación es el registro de temperaturas en el transporte de alimentos para garantizar el mantenimiento de la cadena de frío.

Figura 1: Ejemplo de datalogger comercial, con puerto USB

Los registradores de datos son generalmente compactos, utilizan baterías como fuente de alimentación, están equipados con un microprocesador, memoria interna y uno o más sensores. Una de las principales características y punto fuerte de estos dispositivos es su bajo consumo, lo que permite mediciones continuas durante largos periodos de tiempo (si bien debido a la variedad de condiciones ambientales la vida de la batería puede variar).

Estos dispositivos se utilizan en un amplio rango de aplicaciones medioambientales y mantenimiento de edificios. Algunos de los campos en los que se emplean actualmente son:

Medio ambiente y energía	• Medición del aire, tierra y clima • Estaciones meteorológicas
Sector de salud	• Monitorización de constantes vitales
Industria	• Transporte de alimentos • Control del clima en instalaciones
Deportes	• Monitorización del movimiento • Aceleración

Tabla 1: Campos de uso *data-loggers*

Los registradores de datos tradicionales almacenan las mediciones en la memoria interna y éstos deben ser posteriormente transferidos a un PC para su visualización, análisis y almacenamiento permanente. Según el tipo de comunicación utilizado podemos encontrar:

- USB: hacen uso de conexión USB para la descarga de los datos a un PC. Son los habituales.
- *Web-based*: permiten acceder a los datos mediante *Wifi* o comunicación Ethernet. Estos datos pueden ser enviados a un servidor web para permitir el fácil acceso a los datos.
- *Wireless:* transmiten datos de distintos nodos a un ordenador central en tiempo real, eliminando la necesidad de descargar los datos de manera individual.

- Bluetooth *data-logger*: la comunicación *bluetooth* es la base de estos dispositivos para la transmisión de datos, ya sea a un PC o a un dispositivo móvil [2].

La inmensa mayoría de los *data-logger* "estándar", disponibles comercialmente, hacen uso de cables USB, siendo las conexiones sin hilos poco habituales o destinadas a aplicaciones muy concretas que se desarrollan a medida (por ejemplo, monitorización de datos en un bosque). La idea básica de este trabajo es que en la actualidad la tecnología inalámbrica puede también facilitar el desarrollo de aplicaciones convencionales (como transporte de alimentos), por lo que se va a desarrollar un *data-logger* con conexión inalámbrico, de tamaño y consumo reducidos y operable mediante dispositivo móvil.

1.2 Planificación

Fases de desarrollo del proyecto:

En la **fase inicial** se ha realizado una búsqueda de sensores de temperatura y humedad disponibles en el mercado, estudiando sus características para elegir el que mejor se adapta a nuestros requerimientos.

A continuación en una **segunda fase** se ha diseñado el circuito completo basado en el microcontrolador con las correspondientes etapas de acondicionamiento de los sensores y del módulo Bluetooth.

La **tercera fase** continúa con el desarrollo del programa principal utilizando el entorno de trabajo que ofrece Texas Instruments para sus placas de desarrollo, y las pruebas correspondientes tanto de consumo como de funcionalidad.

Una vez obtenido el funcionamiento deseado en la placa de pruebas se ha pasado a la realización de la placa de circuito impreso y su posterior fabricación.

En paralelo a la implementación del registrador de datos, se ha llevado a cabo el diseño y desarrollo de la aplicación Android para mostrar los datos en gráficas en un dispositivo móvil.

Finalmente se recoge toda la información y se documenta todo lo realizado en el proyecto.

A continuación se muestra un diagrama de Gantt para la visualización de la planificación y programación de tareas a lo largo del proyecto.

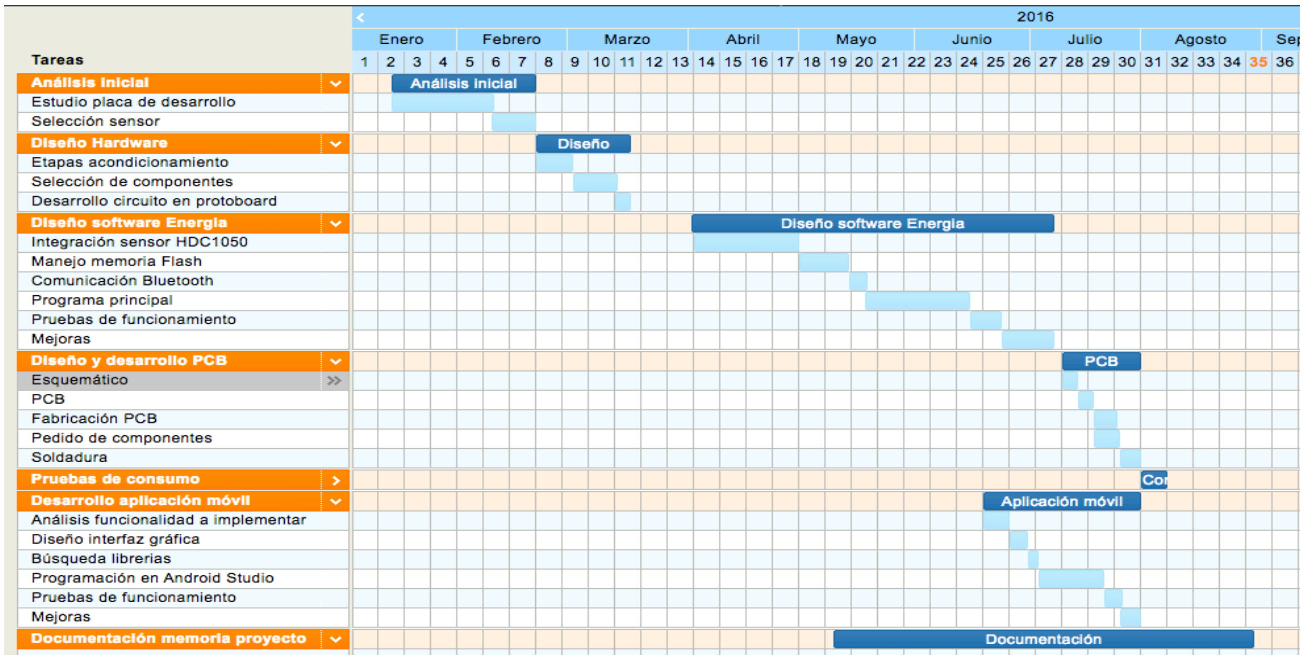


Figura 2: Diagrama de Gantt del desarrollo del proyecto

1.3 Organización de la memoria

La memoria de este proyecto está dividida en seis capítulos, sin incluir la introducción, a continuación se expone un breve resumen del contenido de cada uno.

- **Objetivos**: este capítulo expone los objetivos que se van a abordar en el desarrollo de este proyecto.

- **Diseño del Hardware**: en este apartado se documenta todo lo referente al hardware externo necesario para el desarrollo del dispositivo registrador de datos, tanto el sensor de temperatura y humedad seleccionado como el módulo Bluetooth integrado en el proyecto. Se incluye una descripción de los protocolos de comunicación empleados para la integración del sensor y el Bluetooth.

- **Diseño del Software**: este capítulo explica toda la parte software del proyecto, desde el programa principal del microcontrolador para la implementación del registrador de datos hasta la aplicación móvil desarrollada en la plataforma Android.

- **Prototipo y funcionamiento**: presenta el diseño del esquemático y de la placa PCB para la realización de un prototipo del dispositivo.

- **Conclusiones y trabajo futuro**: se exponen las conclusiones del proyecto así como posibles mejoras futuras, referentes tanto al consumo con la integración de Bluetooth Low Energy, así como de reducción de tamaño, con el diseño de una placa más pequeña para facilitar y demostrar su viabilidad comercial.

2. Objetivos

El objetivo principal de este proyecto es implementar un dispositivo registrador de datos de temperatura y humedad de bajo coste, tamaño reducido y bajo consumo, basado en la comunicación *Bluetooth*. A continuación se explica en detalle cada uno de los objetivos:

- **Bajo coste**: existen numerosos dispositivos de este tipo en el mercado, uno de los retos que plantea este proyecto es el estudio y selección de componentes de bajo coste que permitan obtener la precisión y calidad del dispositivo similares a otros productos en el mercado, pensando en las aplicaciones más típicas de estos dispositivos, como transporte de alimentos o control de clima en instalaciones. En estos casos, la exactitud requerida suele ser de 1°C en el caso de la medida de temperaturas.

- **Reducido tamaño**: un registrador de datos es un dispositivo de pequeño tamaño, siendo una característica imprescindible en el desarrollo de este proyecto, para ello se seleccionarán componentes de soldadura SMD reduciendo el tamaño del prototipo lo máximo posible.

- **Bajo consumo**: se tendrá en cuenta tanto en la selección de componentes como en las estrategias de programación (modos de bajo consumo, microcontrolador "dormido"); se empleará una batería de un tamaño adecuado que permita una capacidad de registro válida para su uso tanto doméstico como industrial.

- **Comunicación inalámbrica**: se hará uso de comunicación *Bluetooth* por su amplia disponibilidad, lo que permitirá la operabilidad del *datalogger* mediante dispositivos móviles estándar, como teléfonos y tabletas.

- **Aplicación Android**: para la consulta de los datos almacenados desde un móvil o tableta se desarrollará una aplicación Android. El usuario podrá conectarse al dispositivo por *Bluetooth* y visualizar en gráficas todos los datos guardados en la memoria flash del registrador de datos. También podrá configurar el tiempo entre dos capturas de datos (permaneciendo el microcontrolador dormido el resto de tiempo). Asimismo, se debe incluir la opción de generar un fichero .txt para exportar los datos, para poder enviarlos por correo o almacenarlos para su posterior consulta y tratamiento desde un computador.

3. Diseño del Hardware

3.1 Descripción general

El registrador de datos diseñado permite la monitorización de parámetros ambientales mediante un sensor de temperatura y humedad, por lo que puede usarse en un amplio rango de aplicaciones tales como el control de estos valores en una cadena de frío o en el transporte de alimentos entre otras muchas.

Todos los datos recogidos por el dispositivo se guardan en la memoria flash del micro controlador, su posterior transmisión a otro dispositivo móvil, para su visualización y tratamiento ,se realizará mediante un módulo de comunicación Bluetooth.

Con el objeto de extender la vida del dispositivo se ha empleado un microcontrolador específico de bajo consumo (MSP430G2553 de Texas Instruments), así como etapas de acondicionamiento de los sensores para controlar su alimentación y activarlos sólo cuándo se realicen las mediciones.

A continuación se presenta un diagrama de bloques para una visualización general del sistema completo:

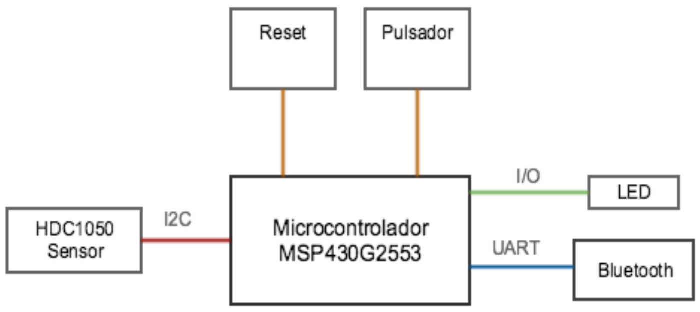


Figura 3: Diagrama de bloques y conexionado

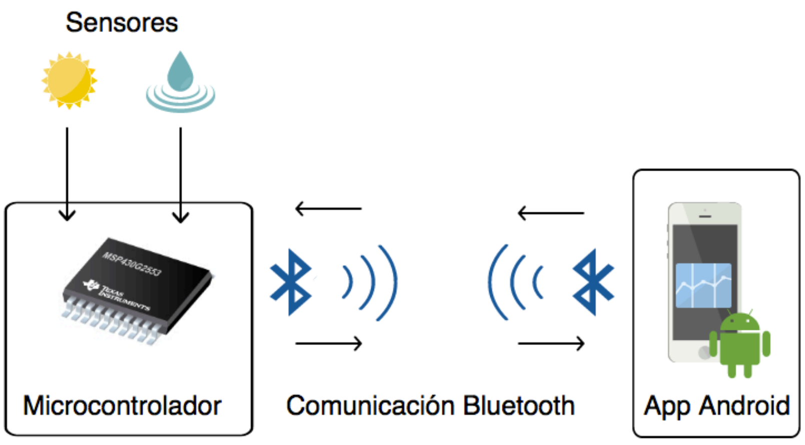


Figura 4 : Diagrama general

3.2 Placa de desarrollo empleada

Para el desarrollo del proyecto se ha utilizado el microcontrolador MSP430G2553 debido a sus características de bajo consumo; se ha trabajado con el kit de desarrollo Launchpad-MSP430 proporcionado por Texas Instruments. Sus principales características son [3]:

- **Microcontrolador:** MSP430G2553
- **Frecuencia:** 16Mhz
- **Flash:** 16KB
- **RAM:** 512B
- 8 canales 10-bit ADC
- Comparador
- 2 contadores de 16-bit
  Protocolos comunicación: I2C, UART, SPI

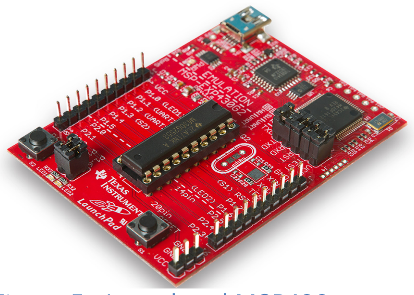


Figura 5 : Launchpad MSP430

Además, para el prototipado e integración de los sensores se ha empleado una protoboard.

3.3 Integración sensor temperatura y humedad

Para la elección del sensor de temperatura y humedad a integrar en el proyecto se realizó un estudio previo de distintos sensores disponibles en el mercado, tanto analógicos como digitales. Los sensores digitales presentaban una mayor precisión así como bajo consumo, reducido coste y la posibilidad de integrar sensor de temperatura y humedad en un único dispositivo, a continuación se muestra una comparativa entre varios sensores digitales:

Sensor	**T**	**H**	**Rango de trabajo**		**Precisión**		**Precio**
			T	**H**	**T**	**H**	
DS18B20	X		-55°C / 125°C	-	± 0.5°C	-	3€
DHT11	X	X	0°C / 50°C	20-80% RH	± 2°C	±5% RH	7€
DHT22	X	X	-40°C / 125°C	0-100% RH	±0.2°C	±2% RH	9.95€
SENSIRION SHT75	X	X	-40°C / 125°C	0-100% RH	±0.2°C	±1.8% RH	40.58€
HDC1050	X	X	-40°C / 125°C	0-100% RH	±0.2°C	±3% RH	7.35€

Tabla 2: Comparativa de sensores

Por su precisión, bajo coste, reducido tamaño y bajo consumo se ha seleccionado el sensor HDC1050 de Texas Instruments; su relación precio/prestaciones supera a las otras opciones.

El **HDC1050** es un sensor digital de humedad que integra sensor de temperatura. Una de las principales ventajas de este sensor es su bajo consumo, haciéndolo una buena elección para aplicaciones en las que se desee prolongar la vida de la batería al máximo. Otra ventaja que ofrece es que permite monitorizar la tensión de alimentación, indicando cuándo dicha tensión es menor de 2.8V. Esta información será de utilidad en este proyecto para conocer el estado de la batería.

Características principales:

- **Tensión de alimentación**: 2.7-5.5V
- Reducido tamaño
- **Comunicación I2C**
- **Consumo en modo dormido**: 100nA
- Resolución de medida14 bit

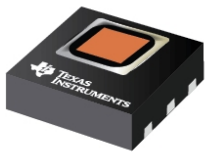
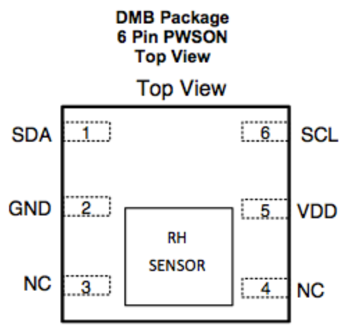


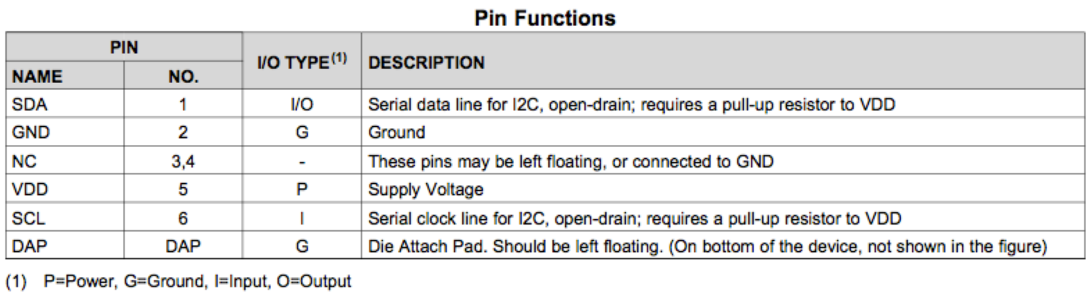
Figura 6: Sensor HDC1050

Algunos parámetros característicos del sensor:

- **Manufacturer ID:** identificador de 16 bits establecido por el fabricante, en este caso Texas Instruments.
- **Serial ID**: identificador numérico de 40 bits único para cada sensor.
- **Device ID**: identificador de 16 bits establecido por el fabricante que determina que el dispositivo es un sensor HDC1050.

El sensor se conecta al microcontrolador a través del bus I2C como esclavo a través de las líneas SDA y SCL, para acondicionar la línea de I2C se deberán añadir dos resistencias de pull-up en ambos pines. El microcontrolador actuará como maestro solicitando los datos al sensor, más adelante, en la descripción del software , se explica como se realiza esta lectura.




Pin Functions

PIN		I/O TYPE[1]	DESCRIPTION
NAME	NO.		
SDA	1	I/O	Serial data line for I2C, open-drain; requires a pull-up resistor to VDD
GND	2	G	Ground
NC	3,4	-	These pins may be left floating, or connected to GND
VDD	5	P	Supply Voltage
SCL	6	I	Serial clock line for I2C, open-drain; requires a pull-up resistor to VDD
DAP	DAP	G	Die Attach Pad. Should be left floating. (On bottom of the device, not shown in the figure)

(1) P=Power, G=Ground, I=Input, O=Output

Figura 7: Huella y pines del sensor HDC1050

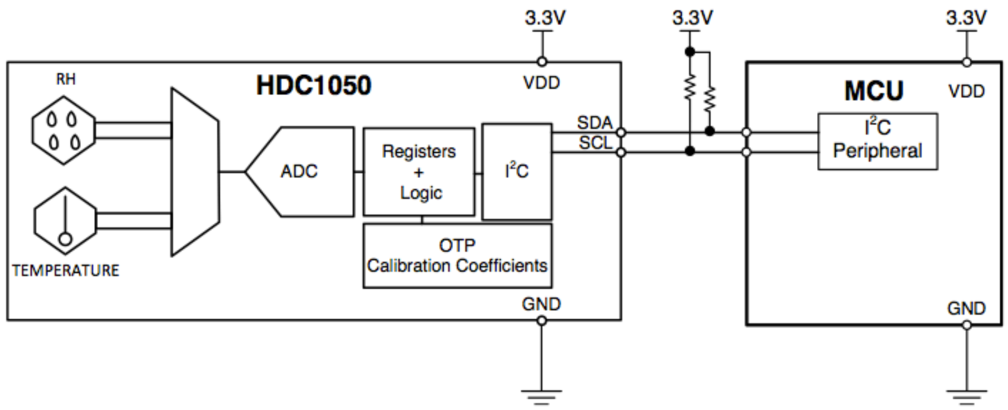


Figura 8: Esquemático de un ejemplo de aplicación del sensor HDC1050

En caso de exponer el sensor a valores altos de humedad de manera prolongada se deberá activar el calentador que integra para ayudar a reducir el offset adquirido. El calentador se controla con un bit del registro de configuración. [4]

Etapa de acondicionamiento:

Para la integración del sensor se ha incluido una etapa de control de la alimentación con el objetivo de minimizar el consumo, de esta manera recibirá alimentación sólo cuando se desee realizar la medición de temperatura y humedad. El control se realiza mediante un pin externo del microcontrolador llevando al transistor a estado de saturación o corte, para encender o apagar el sensor respectivamente.

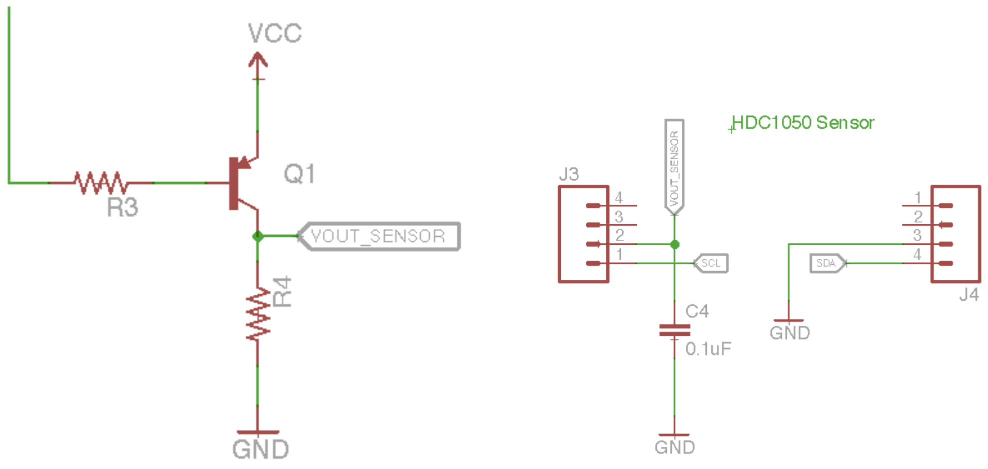


Figura 9: Etapa acondicionamiento del sensor HDC1050

3.4 Protocolo I2C

I2C es un protocolo síncrono que permite la comunicación de múltiples dispositivos esclavos a uno o varios maestros, se pueden conectar hasta 1028 esclavos a un mismo bus I2C. Su uso está dirigido a comunicaciones de corta distancia.

Este protocolo soporta sistemas multi-maestro, permitiendo a más de un maestro comunicarse con todos los dispositivos conectados al bus aunque la comunicación por parte de los maestros no se puede realizar al mismo tiempo, se deben establecer turnos para el empleo del bus.

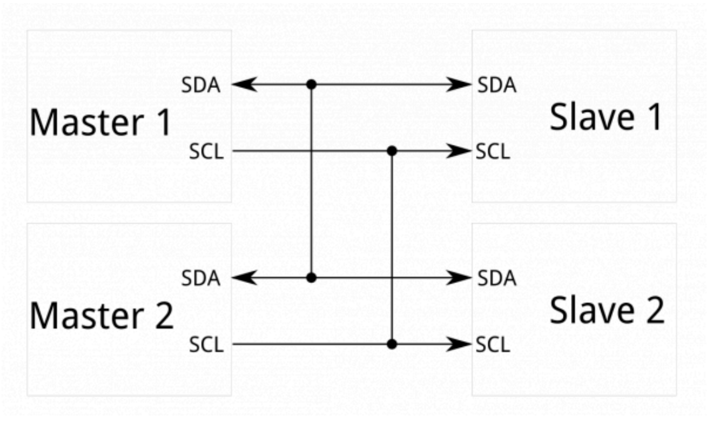


Figura 10: Ejemplo de sistema multi-maestro

La transmisión de datos se realiza a través de dos líneas de señal: SCL (Serial clock line) y SDA (Serial data line). SCL es la señal de reloj, y SDA la señal de datos. La frecuencia de reloj es siempre determinada por el maestro, algunos esclavos pueden fijar dicha frecuencia durante un tiempo determinado en caso de que requieran más tiempo para preparar el dato antes de que el master lo reclame, esta funcionalidad se conoce con el nombre de *"clock stretching"*.

A diferencia de las conexiones UART o SPI, las líneas del bus I2C son del tipo drenaje abierto, es decir un estado similar al de colector abierto pero asociadas a un transistor de efecto de campo o FET. Se deben polarizar en estado alto conectando a la alimentación por medio de resistencias *pull-up* lo que define una estructura de bus que permite conectar en paralelo múltiples entradas y salidas.[5]

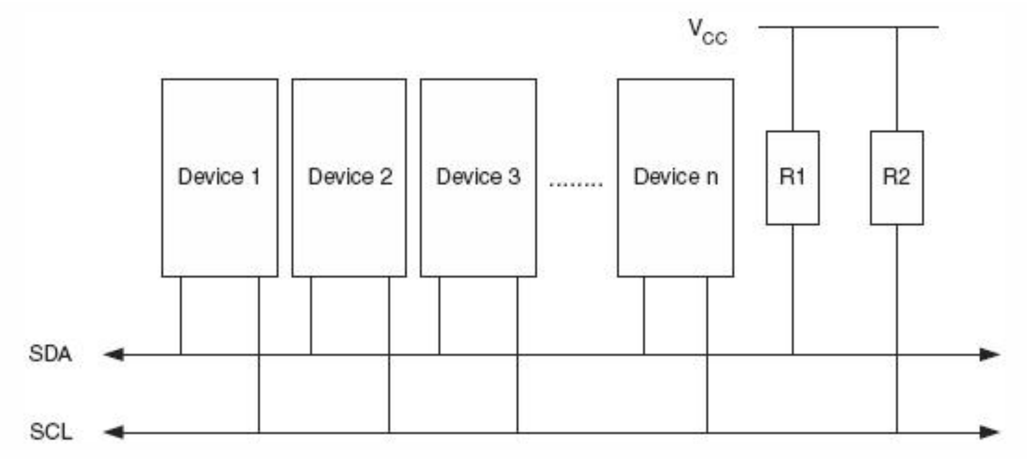


Figura 11: Ejemplo de conexionado de distintos dispositivos al bus.

3.5 Comunicación *Bluetooth*

El Bluetooth [6] es un protocolo de comunicación estandarizado orientado a la comunicación inalámbrica, emisión y recepción de datos, entre diferentes dispositivos. Opera en la banda ISM (médico-científica internacional), abierta en todo el mundo sin necesidad de licencia, a una frecuencia de 2.4GHz.

Se trata de un protocolo seguro, perfecto para distancias cortas, ofrece un bajo consumo y bajo coste permitiendo establecer una comunicación inalámbrica entre dispositivos electrónicos. Además de todo ello ofrece la ventaja de que en la actualidad esta tecnología está embebida en la mayoría de dispositivos, incluyendo todos los *Smart-phones,* es por ello que se ha decidido emplear este protocolo de comunicación para el desarrollo del proyecto.

Algunos parámetros importantes a tener en cuenta:

Nombre: nombre del módulo establecido por el fabricante para su sencilla identificación.

Dirección MAC: identificador numérico único del dispositivo, está compuesto por 12 dígitos hexadecimales separados por ":". Los primeros 24 bits están definidos por el IEEE y los últimos 24 bits son definidos por el fabricante.

El módulo empleado para el desarrollo es Bluetooth HC-06, su correspondiente configuración se indica en la tabla siguiente:

Nombre	HC-06
MAC	98:D3:31:FC:31:14

Tabla 3: Parámetros del módulo Bluetooth empleado

Las redes Bluetooth utilizan el modelo maestro/esclavo para controlar cuando y dónde pueden enviar datos. De esta manera cada maestro puede conectarse a cualquier esclavo para enviar o pedirle datos de la misma manera, pero el esclavo realizará la conexión con un único maestro, y sólo podrá intercambiar datos con él. En nuestro caso el módulo Bluetooth HC-06 actuará como esclavo, será el móvil el que se conecte como maestro para la transmisión de datos.

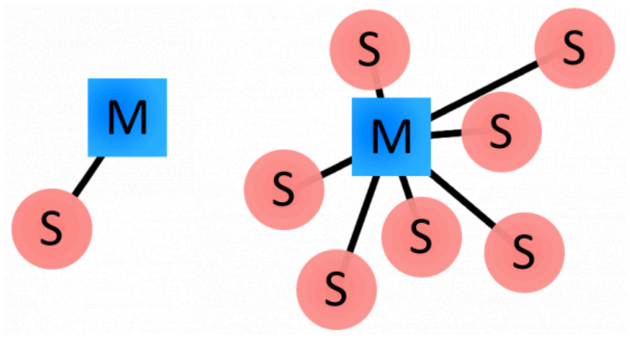


Figura 12: Configuración de redes Bluetooth

La potencia de transmisión y el rango de un módulo Bluetooth es definida por su clase de potencia. Hay módulos que pueden operar sólo en una clase específica y otros que pueden operar en varias clases variando su potencia de transmisión. Hay definidas tres clases:

Clase	Max. potencia salida (dBm)	Máximo rango
1	20 dBm	100m
2	4dBm	10m
3	0dBm	10cm

Tabla 4: Clases de potencia de transmisión Bluetooth

Los perfiles de Bluetooth son protocolos adicionales definidos sobre las bases de estandarización que definen que tipos de archivos se pueden transmitir entre distintos dispositivos. Para que dos dispositivos sean compatibles deben soportar el mismo tipo de perfil.

Hay gran variedad de perfiles, algunos de los más comunes son los siguientes:

- **SPP** (*Serial Port Profile*): emula una línea de comunicación serie y provee una interfaz de comunicaciones basadas en RS-232.
- **HID** (*Human Interface Device Profile*): da soporte a dispositivos tales como ratones, joysticks y teclados.
- **HFP** (*Hands-Free Profile*): usado comúnmente para permitir la comunicación con teléfonos móviles dentro de un coche.
- **HSP** (*Headset Profile*): uno de los perfiles más comunes, permite el uso de los auriculares Bluetooth con los teléfonos móviles.
- **A2DP** (*Advanced Audio Distributed Profile*): define cómo se puede propagar un *stream* de audio, mono o estéreo, entre dispositivos a través de una conexión Bluetooth

En este proyecto se ha empleado el perfil SPP; a partir del puerto UART del microcontrolador establecemos la comunicación Bluetooth mediante las líneas de comunicación serie RX y TX.

A continuación se muestra el módulo utilizado:

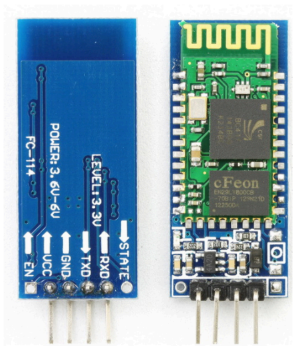


Figura 13: Módulo Bluetooth HC-06

Etapa de acondicionamiento:

Se ha implementado una etapa de acondicionamiento del módulo *Bluetooth* para su conexión al microcontrolador, esta etapa permite controlar la alimentación del módulo mediante un pulsador externo. La señal de alimentación está controlada por un transistor, el módulo Bluetooth se activará solo cuando el usuario desee obtener datos, minimizando de esta manera el consumo.

La etapa se compone de dos resistencias R6 y R7 y un transistor PNP, al recibir el pulso del botón externo el pin 2.1 del microcontrolador pasará a bajo, de esta manera el transistor entrará en saturación permitiendo la alimentación del Bluetooth, una vez realizada la transmisión de datos el pin se configurará en alto dejando el transistor en corte.

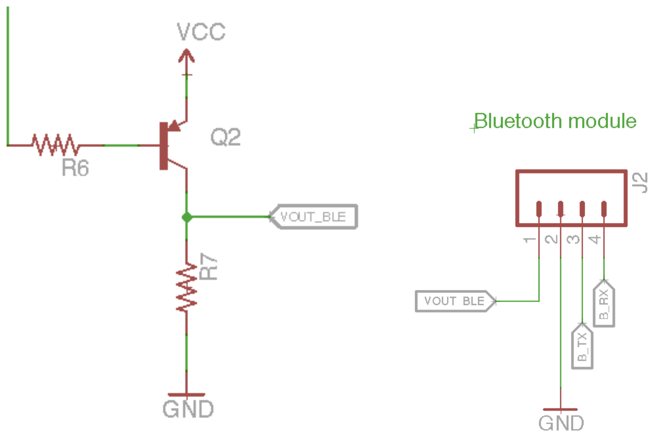


Figura 14: Etapa de acondicionamiento del módulo Bluetooth

3.6 Alimentación y consumo

El microcontrolador empleado está diseñado específicamente para aplicaciones de consumo reducidos, por ello implementa cuatro modos de bajo consumo. [7]

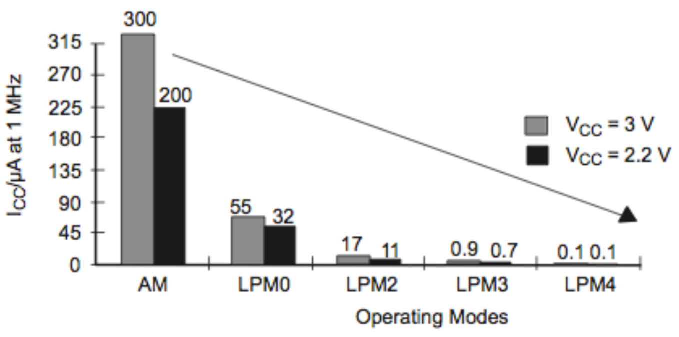


Figura 15: Modos de bajo consumo del MSP430G2553

Hay que tener en cuenta que para obtener estos valores de consumo se deben cumplir las condiciones detalladas en el datasheet , en nuestro caso el IDE de Energia configura el reloj a la máxima frecuencia, 16MHz, lo que incrementa bastante el consumo mínimo.

Para la medición del consumo se ha conectado un *jumper* en serie con el cable de alimentación de la batería, permitiendo realizar medidas de corriente continua con el polímetro en serie.

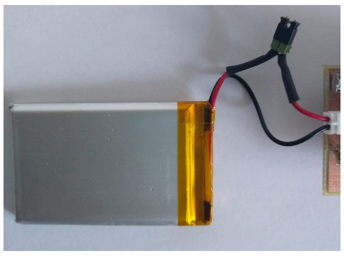


Figura 16: Adaptación de la batería para la medición del consumo

A continuación se muestra una tabla con los consumos medidos en el prototipo desarrollado para los distintos estados posibles con el polímetro:

Ciclo	Tiempo	Corriente media	Ponderación
Sleep	300s	0,9mA	270mAs
Medición sensor	1,2s	43mA	51,6mAs
Calentamiento sensor	2,5s	6mA	15mAs
TOTAL	303,7s	**1,1mA**	336,6mAs

Tabla 5: Estudio de consumos del dispositivo implementado

Para conocer con exactitud el tiempo empleado por el sensor para la medición y el calentamiento inicial se ha empleado el osciloscopio, a continuación se muestra una foto correspondiente a la activación y medición del sensor.

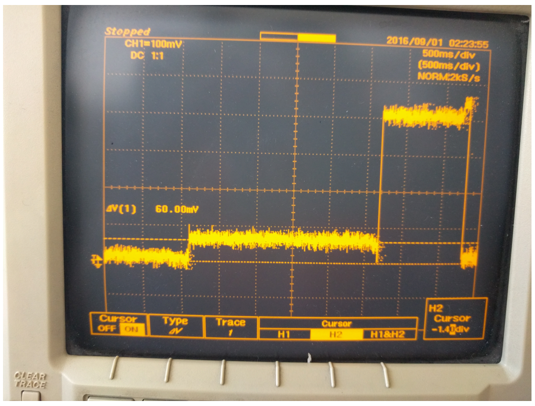


Figura 17 : Captura de tiempos de inicialización y medición del sensor.

Los consumos correspondientes a la comunicación Bluetooth se han tratado por separado ya que dicha conexión se realizará esporádicamente, hemos considerado para el consumo una conexión diaria, a continuación se muestra la tabla de consumos:

Ciclo	Tiempo	Corriente media	Ponderación
Encendido y conexión	120s(máximo)	60mA	7200mAs
Transmisión	1,8s	56mA	100,8mAs
TOTAL	121,8s	**59,94mA**	7300,8mAs

Tabla 6: Estudio de consumo de la comunicación Bluetooth

Considerado un tiempo de consumo de 24h obtenemos:

Ciclo	Tiempo	Corriente media	Ponderación
Registrador datos mediciones	86400-121,8s	1,1mA	94906mAs
Bluetooth	121,8s	59,94mA	7300,8mAs
TOTAL	86400s	**1,18mA**	102206,8mAs

Tabla 7: Consumo total de las mediciones y la comunicación Bluetooth.

Se obtiene una corriente media de 1,18mA teniendo en cuenta una conexión Bluetooth diaria. He incorporado para la alimentación del dispositivo una batería de Litio de 3,7V y una capacidad de 1100mAh, por lo que la autonomía para un tiempo de *sleep* de 5 minutos será de:

$$\frac{1100mAh}{1.18mA} = 932{,}20h$$

Estudiados los consumos, se obtiene una autonomía de 38 días en el caso extremo de una medición cada 5 minutos, lo cual nos parece adecuado. Usando la misma metodología de cálculo obtendríamos una autonomía de 45 días tomando una medida cada hora, y 46 días realizando una medición al día.

4. Diseño del Software

4.1 Programa principal

Para el desarrollo de la programación del microcontrolador he empleado el IDE Energia que ofrece Texas Instruments.

Energia es un software de código abierto, está basado en el entorno de programación Arduino ofreciéndonos numerosas librerías que dan soporte a las placas de desarrollo de Texas Instruments [8].

A continuación se muestran los diagramas de flujo sobre la aplicación principal y la interrupción bluetooth.

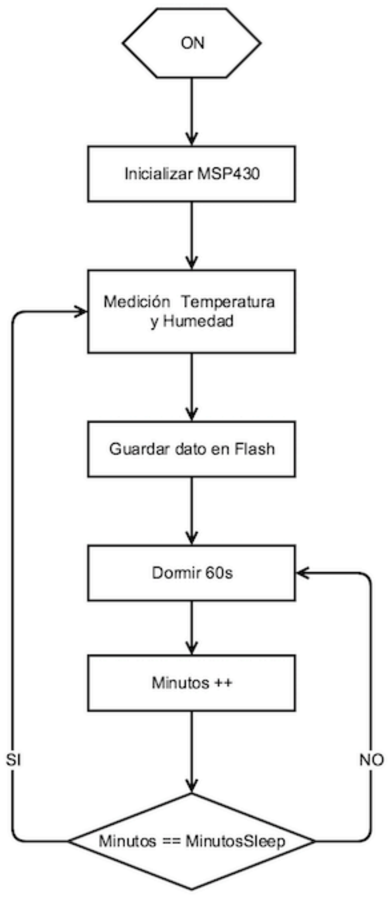


Figura 18: Diagrama de flujo del funcionamiento de la aplicación principal

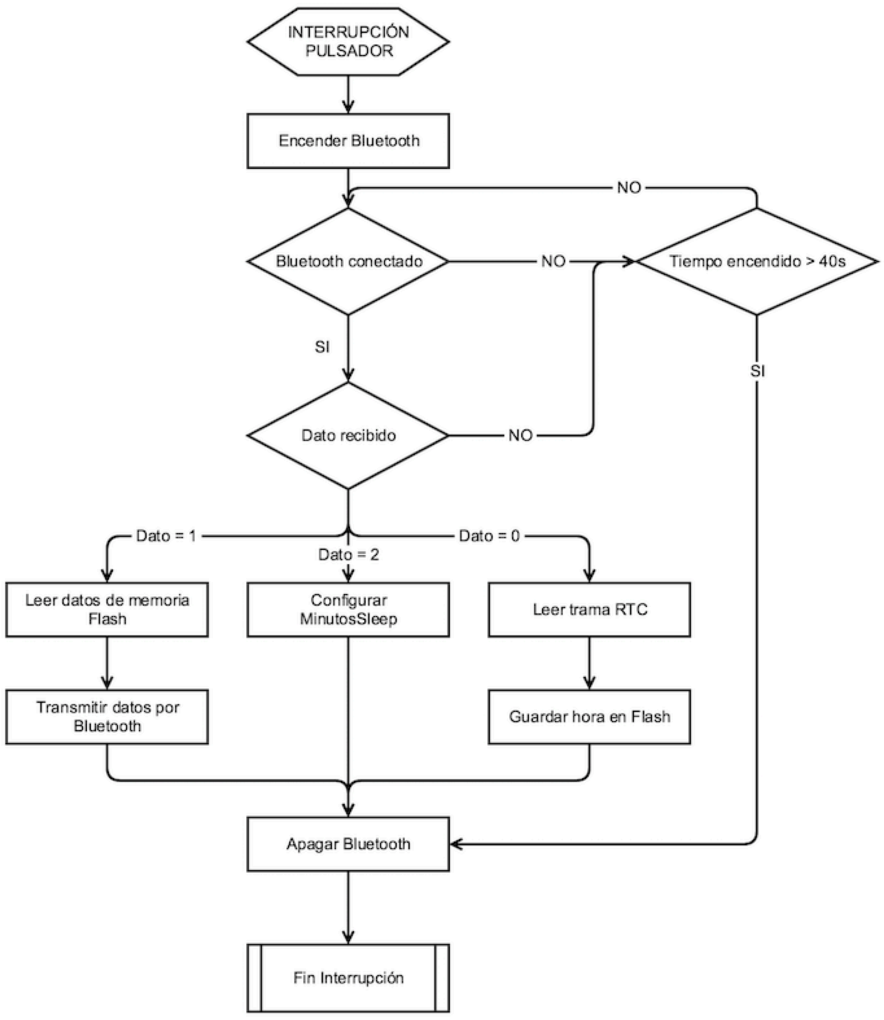


Figura 19: Diagrama de flujo, interrupción asíncrona Bluetooth

Comunicación I2C

Para la comunicación I2C se ha empleado la librería Wire integrada en Energia [9].

Funciones empleadas:

- **beginTransmission(**adress**)**: inicia la comunicación I2C con el esclavo dada su dirección.
- **endTransmission()**: termina la comunicación con el esclavo, admite un boolean como parámetro, en caso de ser true mandará un mensaje de stop una vez finalizada la conexión, necesario para el correcto funcionamiento de algunos dispositivos.
- **available()**: devuelve el número de bytes disponibles para su lectura.
- **write()**: envía datos al esclavo.
- **requestFrom()**: utilizada por el master para pedir un número de bytes al esclavo, a continuación se empleará la función read() o available() para leer los bytes recibidos.
- **read():** lee los bytes transmitidos por el esclavo después de llamar a la función requestFrom().

HDC1050- Sensor temperatura y humedad

El sensor HDC1050 opera únicamente como esclavo en el bus I2C, la conexión al bus I2C se realiza mediante las líneas SDA y SCL.

Una vez alimentado el sensor, se necesita un tiempo de 15ms para poder comenzar a realizar mediciones de temperatura y humedad, esta espera se implementa en la librería mediante una espera de 50ms para asegurar que cumple el tiempo mínimo.

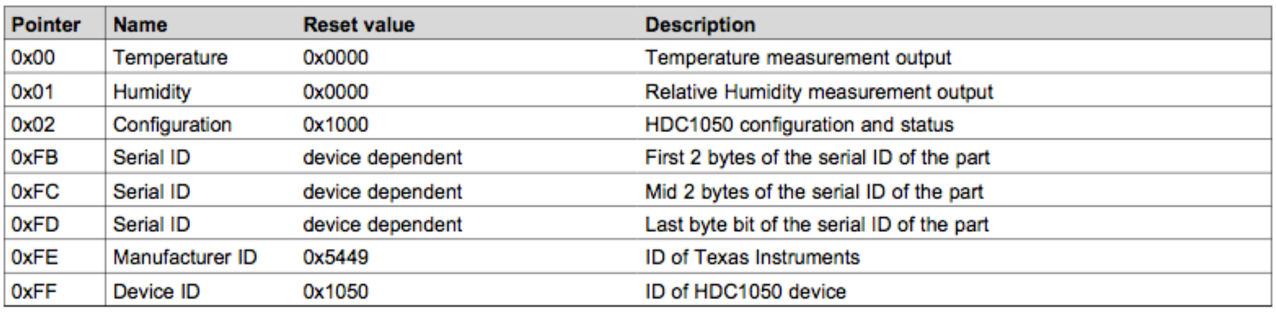
Para iniciar la comunicación con los dispositivos esclavos, el maestro debe enviar el byte de dirección del esclavo. Esta dirección consiste en 7 bits únicos para el esclavo, el sensor HDC1050 tiene configurada la dirección 1000000 (dirección de 7bits). Una vez enviada el esclavo estará preparado para realizar cualquier operación de escritura o lectura con el maestro [4].

Configuración inicial del sensor:

Por defecto ambas medidas, temperatura y humedad, se realizan con una resolución de 14 bits y está configurado en el modo de lectura de ambas medidas al mismo tiempo.

Mapa de registros:

El HDC1050 tiene registros de datos que guardan información sobre la configuración, resultados de las medidas de temperatura y humedad e información del dispositivo.



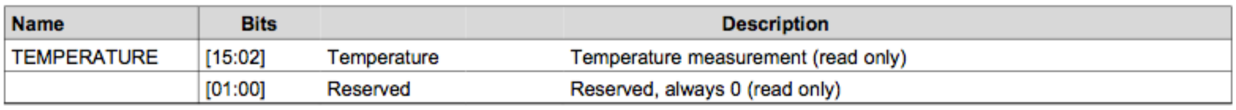
Pointer	Name	Reset value	Description
0x00	Temperature	0x0000	Temperature measurement output
0x01	Humidity	0x0000	Relative Humidity measurement output
0x02	Configuration	0x1000	HDC1050 configuration and status
0xFB	Serial ID	device dependent	First 2 bytes of the serial ID of the part
0xFC	Serial ID	device dependent	Mid 2 bytes of the serial ID of the part
0xFD	Serial ID	device dependent	Last byte bit of the serial ID of the part
0xFE	Manufacturer ID	0x5449	ID of Texas Instruments
0xFF	Device ID	0x1050	ID of HDC1050 device

Figura 20 Tabla de registros del sensor HDC1050

Registro de temperatura:

El registro de temperatura almacena 16 bits del resultado de la medición, los bits menos significativos D1 y D0 son 0, el resultado de la adquisición de datos son siempre 14 bits. La temperatura se calcula a partir de los datos obtenidos de la siguiente manera, especificada en el datasheet del sensor:

$$Temperatura(\,ºC) = \left(\frac{TEMPERATURA[15:00]}{2^{16}}\right) * 165ºC - 40ºC$$



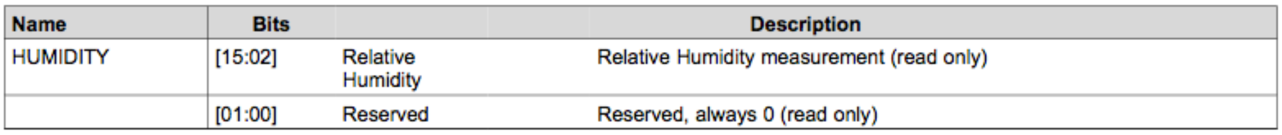
Name	Bits	Description	
TEMPERATURE	[15:02]	Temperature	Temperature measurement (read only)
	[01:00]	Reserved	Reserved, always 0 (read only)

Figura 21: Descripción del registro de temperatura

Registro de humedad:

El registro de humedad contiene, al igual que el de temperatura, 16 bits con el resultado de la medida, los dos bits menos significativos D1 y D0 son 0. Cálculo para obtener la humedad, especificado en el datasheet del sensor, es el siguiente:

$$Humedad\ Relativa(\ \%\ RH) = \left(\frac{HUMEDAD[15:00]}{2^{16}}\right) * 100\ \%\ RH$$



Name	Bits	Description	
HUMIDITY	[15:02]	Relative Humidity	Relative Humidity measurement (read only)
	[01:00]	Reserved	Reserved, always 0 (read only)

Figura 22: Descripción del registro de humedad

Operaciones de lectura y escritura:

Para acceder a un registro en particular debemos escribir la dirección de dicho registro en el puntero de pila, el valor del puntero será el primer byte transferido después de enviar la dirección del esclavo con el bit R/W configurado en bajo. Cada escritura en el esclavo requiere un valor de registro en el registro de pila.

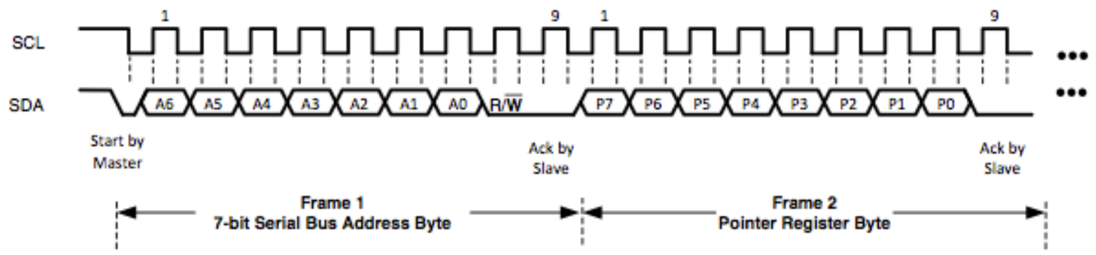


Figura 23: Trama de escritura

La operación de lectura se realizará sobre el último valor guardado en el puntero de pila por una operación de escritura. Para cambiar el puntero de registro para una nueva lectura se deberá realizar una escritura en el registro. Esta operación se realiza escribiendo la dirección del esclavo con el bit R/W en bajo seguido del valor del puntero de registro, como se ha explicado anteriormente.

El Master puede entonces generar una señal de START para iniciar la lectura.

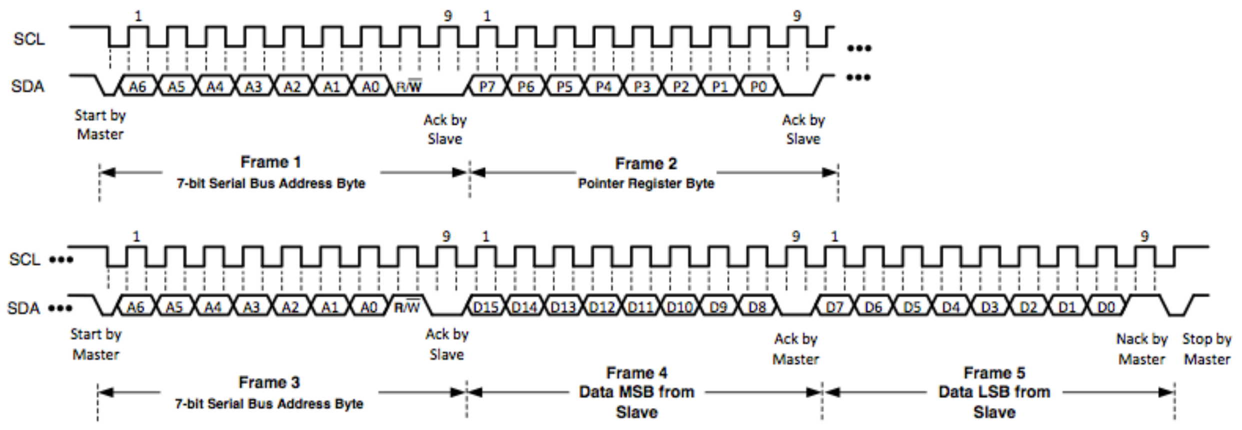


Figura 24: Trama de lectura

Funciones librería manejo HDC1050:

- **getManufacturerID():** devuelve el Manufacturer ID.
- **getDeviceID():** devuelve el Device ID.
- **getSerialID():** devuelve el Serial ID.
- **batteryOK():** devuelve verdadero si la alimentación del sensor es mayor que 2.7V en caso contrario devuelve falso.
- **getTemperatureHumidity():** realiza una medición de temperatura y humedad y devuelve ambos valores como float.

Librerías relacionadas: HDC1050.h y HDC1050.cpp

El código de dichas librerías se encuentra en los anexos.

Memoria Flash

Para el almacenamiento de los datos obtenidos en memoria Flash he hecho uso de la librería *MspFlash* que ofrece TI dentro del paquete de librerías incluidas en Energia [10].

MspFlash permite escribir y leer en la memoria Flash. Para un fácil manejo de los datos y control de las direcciones de memoria disponibles he creado otra librería, una capa superior denominada *MyFlash.*

El microcontrolador utilizado ofrece 16Kb de memoria Flash [7], es muy importante tener en cuenta que toda la información referente a los vectores de interrupción se almacena en el segmento 0 y, por tanto, no debe ser escrito. Además, el código del programa también se guarda en memoria Flash, en este caso el programa desarrollado ocupa 10Kb por lo que se dispone de 6Kb libres de memoria para almacenar datos (menos los 512B correspondientes al segmento 0).

A continuación se muestra una captura del *datasheet* del microcontrolador que refleja la estructura de la memoria Flash:

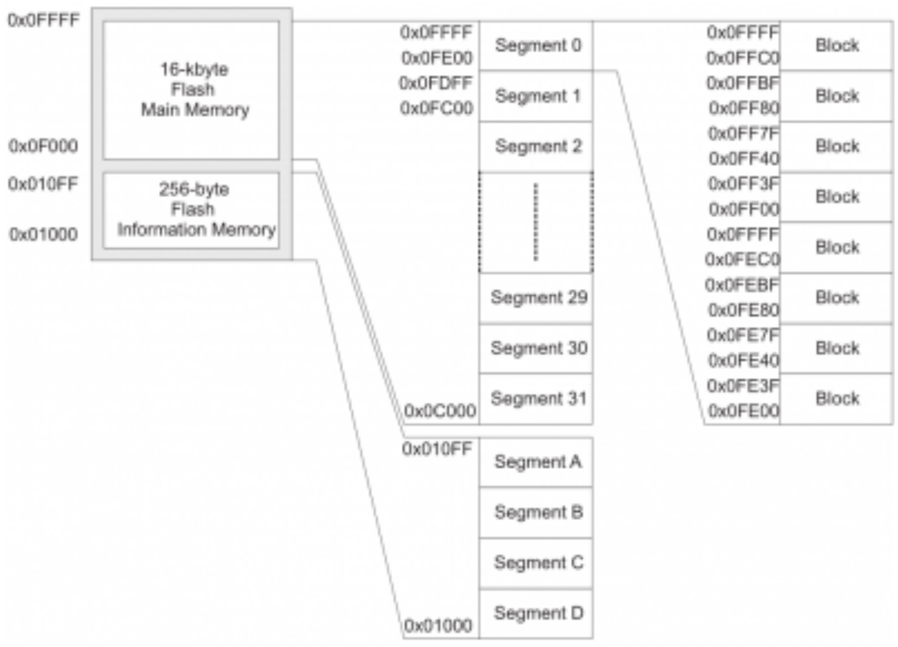


Figura 25: Organización de la memoria del MSP430G2553

La estructura de cada dato guardado es la siguiente: "**TTtt:HHhh:**" correspondiendo los 4 primeros dígitos a la temperatura y los cuatro siguientes a la humedad; en total se almacenan 10 bytes por medición. Dado el espacio del que disponemos esto permitirá almacenar 614 medidas.

Librerías relacionadas: MyFlash.cpp y MyFlash.h

El código de dichas librerías se encuentra en los anexos.

Comunicación Bluetooth

Tal como se ha explicado anteriormente el perfil implementado en la comunicación Bluetooth corresponde a la comunicación serie [12]. De esta manera para transmitir datos por Bluetooth utilizaremos la librería de hardware serial que nos ofrece Energia para el manejo de la UART del microcontrolador. Estableceremos la comunicación serie a 9600 baudios, adecuada para la comunicación Bluetooth.

Funciones empleadas:

- ***available()***: devuelve el número de bytes disponibles para su lectura en el puerto serie.
- ***flush()***: vacía el buffer de datos recibidos.
- ***print()***: imprime datos en el puerto serie en formato ASCII.
- ***println()***: misma funcionalidad que *print()* incluyendo salto de línea.

Para inicializar el puerto serie se debe llamar a la función ***begin()*** antes de transmitir o recibir datos, pasándole como parámetro la tasa de baudios a la que deseas configurar la comunicación serie.

Modos bajo consumo

El IDE Energia nos proporciona la función *sleep()* para dormir el microcontrolador:

- ***sleep() :*** configura el microcontrolador en el modo de bajo consumo LPM3, como parámetro hemos de pasarle el tiempo que deseamos que duerma en milisegundos.

Uno de los principales objetivos de este proyecto es minimizar el consumo del dispositivo, por ello entre mediciones el microcontrolador se configura en modo dormido, *alargando al máximo la vida de la batería.*

Para despertar el microcontrolador de este estado dormido emplearemos la función ***wakeup()***.

Ejemplo de uso en la rutina de interrupción del pulsador:

```
//Interrupt function, launched when push button pressed

void blink(){

        bluetooth = 1;

        wakeup();

}
```

RTC – Reloj tiempo real

Se ha desarrollado una librería para el control del tiempo real con precisión de minutos. Esto permite obtener la hora de cada medición.

Para configurar la hora del comienzo de las mediciones deberemos utilizar la aplicación móvil, esta nos mandará una trama por Bluetooth del tipo **"MMddhhmm"** en la cual los dos primeros dígitos corresponden al número del mes, los dos siguientes al día, hora y minutos sucesivamente. Este dato se guardará en memoria Flash para su posterior consulta.

Como se ha explicado en el apartado anterior se ha configurado el microcontrolador para permanecer en modo dormido cinco minutos, valor configurable por el usuario a través de la aplicación móvil , de esta manera una vez obtengamos los datos almacenados a partir de la fecha de inicio guardada en memoria y con el conocimiento del tiempo a dormir establecido, podremos obtener la hora de cada medición.

Librerías relacionadas: MyRTC.cpp y MyRTC.h

Toda la información sobre estas librerías se encuentra en los anexos.

4.2 Aplicación móvil Android

El software empleado para el desarrollo de la aplicación móvil, basada en el sistema operativo Android, es Android Studio.

La aplicación móvil ofrece al usuario la posibilidad de:

- Visualizar los datos de las medidas registradas en el dispositivo en gráficas
- Configurar el tiempo entre mediciones
- Generar y almacenar un fichero .txt para poder exportar los datos

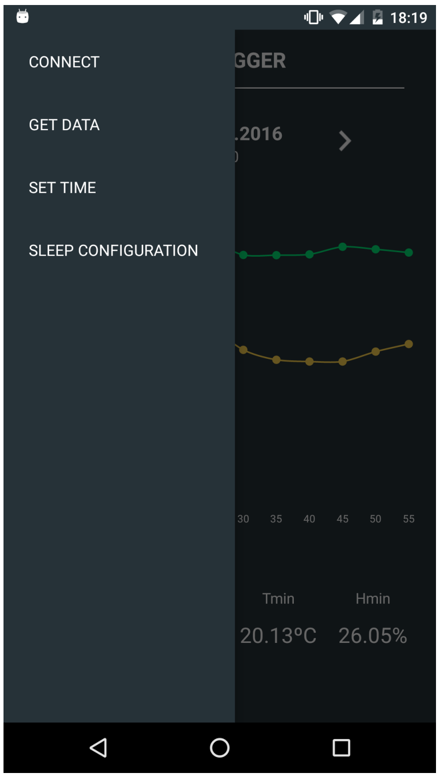


Figura 26: Pantalla del menú de la aplicación móvil

Comunicación Bluetooth

Para establecer la comunicación Bluetooth con el dispositivo implementado empleo la API de Bluetooth que ofrece la plataforma de Android. Esta API nos da acceso al Bluetooth del dispositivo móvil, habilitando una conexión punto a punto o multipunto.

La API nos da la posibilidad de:

- Buscar otros módulos Bluetooth
- Enlazar con un módulo Bluetooth
- Establecer un canal de conexión
- Transferir datos
- Manejar múltiples conexiones

Para hacer uso del Bluetooth debemos incluir los permisos requeridos en nuestra aplicación en el fichero AndroidManifest.xml, el permiso BLUETOOTH permite que la aplicación se conecte, desconecte y transfiera datos a otro dispositivo Bluetooth y el permiso BLUETOOTH_ADMIN permite modificar la configuración Bluetooth y descubrir nuevos dispositivos [13].

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

En primer lugar se debe comprobar si el dispositivo móvil empleado soporta la comunicación Bluetooth, para ello emplearemos la clase *BluetoothAdapter* que representa el Bluetooth del dispositivo móvil. Se instancia una referencia a dicha clase, en caso de que sea nulo significa que el Bluetooth no es soportado por el dispositivo móvil mostrándose un mensaje informativo al usuario. Muestra de la referencia a la clase BluetoothAdapter:

```
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if (mBluetoothAdapter == null) {
    // Device does not support Bluetooth

}else {
    //Connect to the HC-06 module known its mac
}
```

A continuación obtendremos una lista de *BluetoothDevice*s asociados, con el nombre y dirección MAC de cada uno.

Se debe tener en cuenta que si no se ha realizado conexión con el módulo Bluetooth anteriormente debe parearse manualmente antes de poder establecer la conexión. Para parearlo se deberá introducir una contraseña, el módulo HC-06 tiene por defecto la contraseña 1234.

Conocida la dirección MAC del módulo Bluetooth integrado en el registrador de datos se realizará la conexión directa con él en caso de que esté disponible.

Para iniciar una conexión con un dispositivo remoto se debe utilizar el *BluetoothDevice* obtenido anteriormente para adquirir un *BluetoothSocket* y realizar la conexión, para obtener el BluetoothSocket llamaremos a la función createRFcommSocketToServiceRecord(UUID) pasándole como parámetro el identificador UUID del módulo HC-06 empleado, el parámetro UUID es un identificador de 16 bytes único universal.

Una vez obtenido el BluetoothSocket se inicia la conexión mediante la función connect(), establecida la conexión la aplicación ya estará lista para recibir y transmitir datos.

La recepción de datos se realiza mediante un *Listener, método que escucha si hay algún dato para leer y en caso de haberlo lo almacena en un buffer.*

Para la conexión entre dispositivos y transmisión de datos he creado la clase *BluetoothService* que integra todas las funciones referentes a ello y permite utilizarlas de manera más sencilla en la actividad principal.

En los anexos se incluye la clase *BluetoothService.*

Visualización datos

Una vez obtenidas las mediciones empleo la librería *WilliamChart* para mostrar los datos en gráficas.

La aplicación incluye una única gráfica en la que se visualizan los datos de temperatura y humedad con su hora de medición.

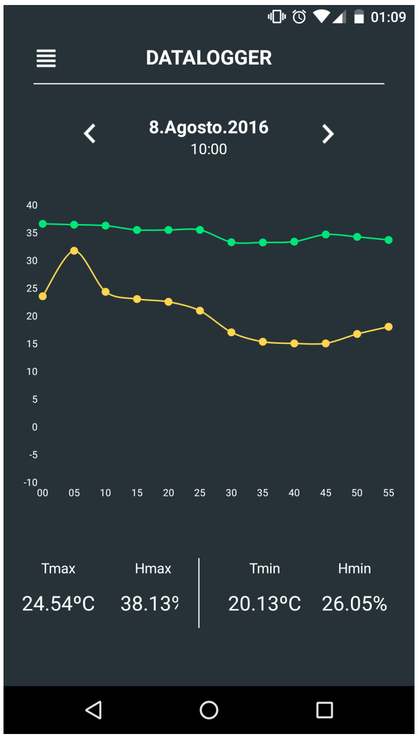


Figura 27: Pantalla de datos en la aplicación móvil

Generación fichero .txt

Una característica de los data-loggers es que permiten generar un archivo .txt para poder exportar los datos, por ello en la aplicación móvil hemos incluido esta funcionalidad.

Para la creación del fichero empleo la clase *FileWriter* [14] que permite añadir texto a un fichero, creado anteriormente con la clase File, mediante la función *append(String data)* a la cuál le paso como parámetro un String con todas las mediciones obtenidas en un formato adecuado.

Para guardar el fichero generado en el dispositivo móvil se debe elegir entre almacenamiento interno o externo, debido a que algunos dispositivos no tienen almacenamiento externo empleo el interno [11].

5. Prototipo

Para el diseño del esquemático y la placa de circuito impreso se ha utilizado el programa de diseño electrónico EAGLE, la versión 7.5.0.

5.1 Esquemático

A continuación se presenta una captura del esquemático.

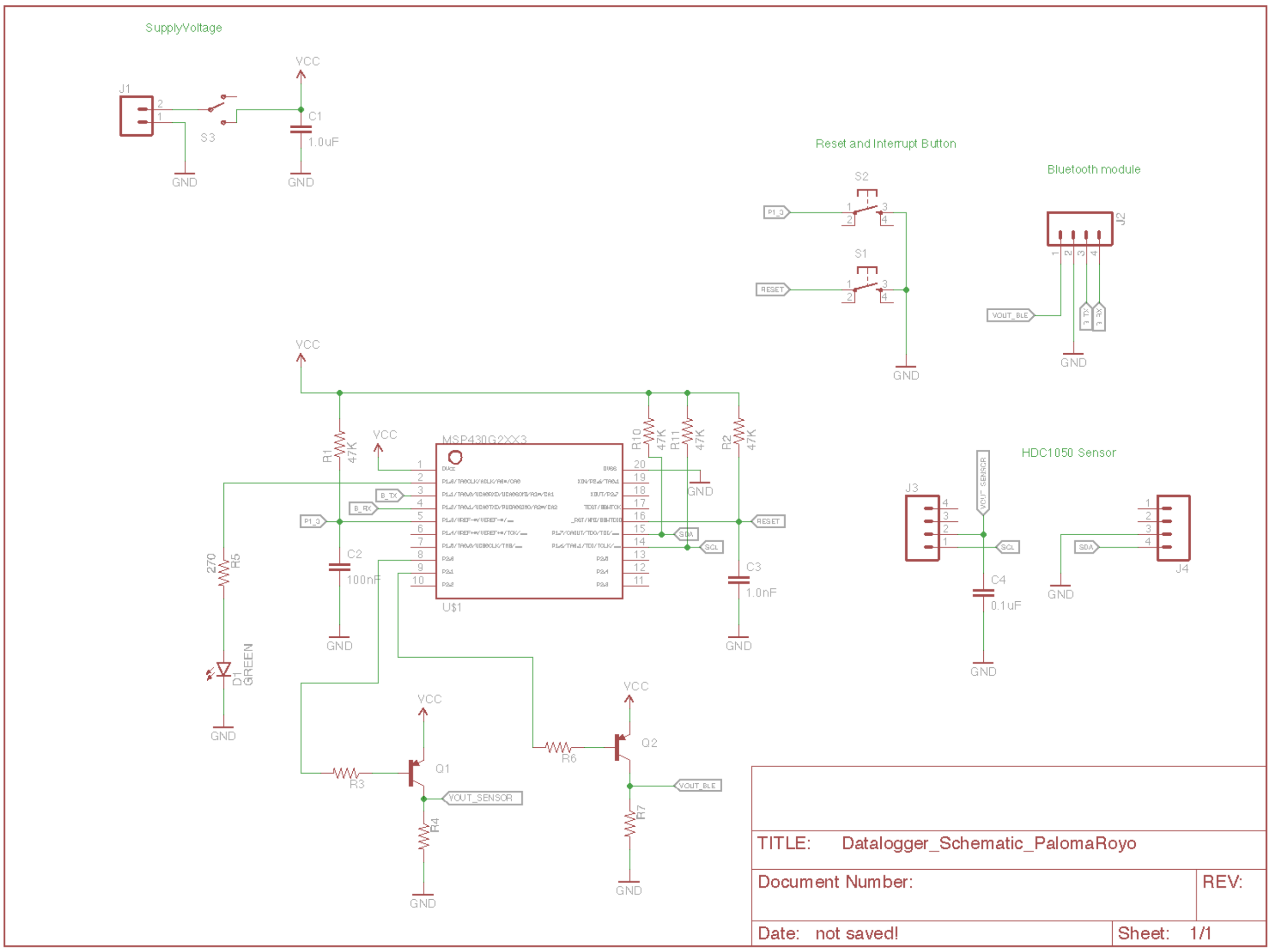


Figura 28: Esquemático del dispositivo

Componente	Valor	Componente	Valor
R1,R2, R10, R11	47KΩ	U$1	MSP430G2553
R3, R6	380Ω	J3,J4	HDC1050
R4, R7	100Ω	J2	Bluetooth HC-06
R5	270Ω	S1, S2	Pulsador
D1	Led verde	D1	Led verde (pruebas)
C1	1 µF	J1	Conector batería
C2, C4	0,1 µF	S3	Interruptor deslizante
C3	1nF		
Q1, Q2	BC557B		

Tabla 8: Tabla de componentes

5.2 Diseño PCB

Otro de los objetivos que queríamos conseguir con este proyecto es el diseño de un dispositivo de reducido tamaño, por eso a la hora de incorporar el módulo de comunicación Bluetooth, el sensor y las etapas de acondicionamiento se ha intentado aprovechar el espacio al máximo, y se han empleado componentes de soldadura SMD (sensor, resistencias, condensadores y led).

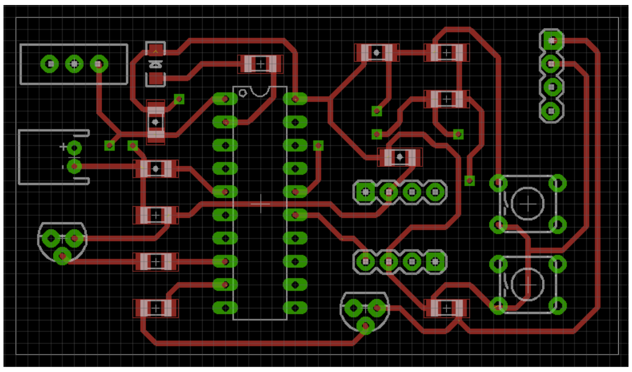


Figura 59: Cara superior placa de circuito impreso

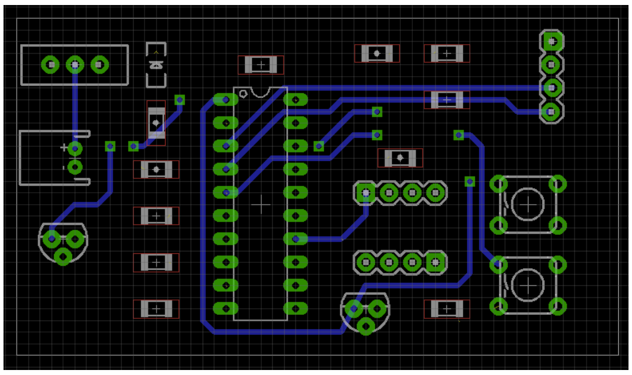


Figura30: Cara inferior placa de circuito impreso

5.3 Presupuesto

Este punto es realmente importante para la rentabilidad económica del dispositivo. A continuación se presenta una tabla con todos los componentes utilizados en el proyecto y el coste total del registrador de datos (tan solo como suma de precio de componentes).

Componente	Referencia	Cantidad	Distribuidor	Precio
MSP430g2553	2113748	1	Farnell	2,06 €
Sensor HDC1050	2499128	1	Farnell	7,35 €
Módulo HC-06	HC-06	1	Amazon	4,00 €
Transistor PNP BC557B	1574384	2	Farnell	0,48 €
Pulsador	DTS61N	2	Diotronic	0,26 €
Conector JST	9491902	1	Farnell	0,24 €
Batería Litio1100mAh	GSP053759	1	Diotronic	13,58 €
Interruptor		1	Farnell	
R- 47KΩ	9241043	4	Farnell	0,06 €
R- 380Ω	-	2	Diotronic	0,08
R- 100Ω	-	2	Diotronic	0,08
C- 1nF	499316	1	Farnell	0,14
C- 100nF	1856626	2	Farnell	1,08 €
C- 1µF	1856626	1	Farnell	0,06
Conector hembra	1218869	1	Farnell	1,27 €
			TOTAL	**30,74 €**

Tabla 9: Presupuesto

Se podría reducir en gran medida el coste del dispositivo empleando otra batería, ya que la elegida (de litio recargable) supone casi la mitad del total. He elegido esa batería por ser compacta y por su relación capacidad/tamaño.

5.4 Funcionamiento

El funcionamiento del dispositivo se podrá comprobar en el mismo momento de la presentación.

Conclusiones y Trabajo futuro

El proyecto desarrollado ha conseguido alcanzar los objetivos planteados inicialmente, disponiendo de un prototipo operativo de *datalogger* de tamaño y consumo reducido, con conexión Bluetooth y que puede ser operado desde un teléfono móvil o *tablet* mediante la aplicación Android desarrollada. La precisión en la medida de temperaturas es de ±0.2°C , la de la medida de humedad es de ±3% RH, tiene capacidad de almacenar hasta 614 datos (fácilmente ampliable sin más que seleccionar un microcontrolador con más memoria flash) y, finalmente, con una autonomía de unas seis semanas registrando datos cada 5 minutos.

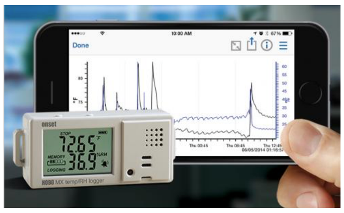
El coste estimado como simple suma de precio de componentes es de unos 31 euros; para comparar se presentan a continuación las características y precio de distintos registradores de datos que se pueden encontrar actualmente en el mercado:

PCE_HT 71N – PCE Instruments
Puerto USB

	Memoria: 32000 valores (16000 por parámetro) **Rango temperatura**: -40, +70°C **Rango humedad**: 0,100% **Precisión**: ± 2ºCd	**59,29 €**

HOBO BLE Data Logger - ONSET

Bluetooth Low Energy

	**Memoria**: 84000 valores **Rango temperatura**: -20, +70°C **Rango humedad**: 0,100% **Precisión**: ± 0.2ºC	**121 €**

Como se puede observar el registrador de datos desarrollado presenta una precisión igual al del datalogger de mayor precio (el sensor que integra ofrece muy buenas prestaciones), y aunque el número de medidas almacenadas en memoria es menor se puede plantear en un desarrollo posterior el aumento de la memoria haciendo uso de un microcontrolador de la misma familia pero con más memoria flash, o bien incluyendo una memoria externa; en cualquiera de los casos no supondría un aumento de precio significativo en el presupuesto total.

En cuanto al tamaño reducido ,que también se establecía como objetivo, se ha desarrollado un prototipo de un tamaño considerablemente pequeño (6.5 cm x 3.5cm) y compacto. Pero se trata de un prototipo; para demostrar la viabilidad de este producto se ha realizado otro diseño de la PCB con componentes de una métrica menor y un microcontrolador de soldadura superficial, el tamaño de la nueva placa es 2,16cm x 3.5m, alrededor de 1/3 del prototipo. A continuación se muestran unas capturas de la cara superior e inferior de la PCB, y la serigrafía de los componentes utilizados:

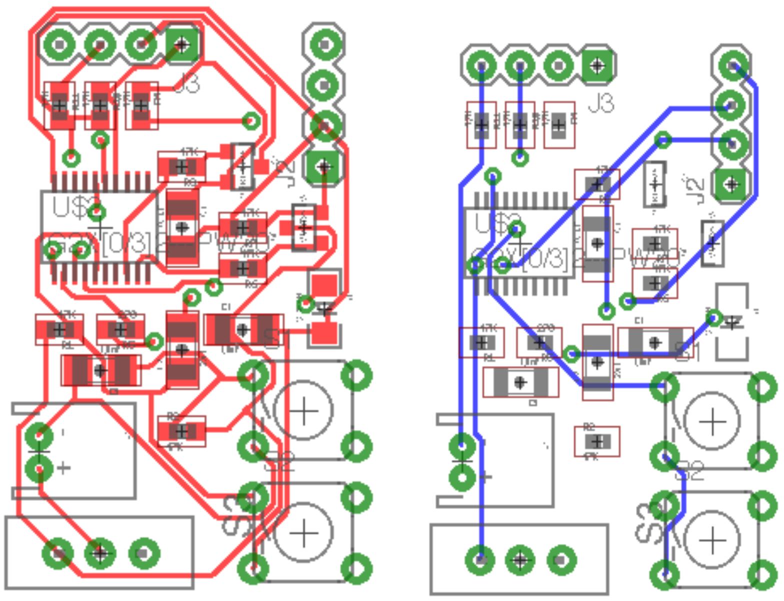


Figura 31: Cara superior e inferior del nuevo diseño de la PCB

En un trabajo futuro cabe considerar el remplazo del módulo Bluetooth empleado por un módulo Bluetooth Low Energy para minimizar el consumo en la comunicación.

El Bluetooth Low Energy también conocido como Bluetooth 4.0 ofrece una comunicación de radio de corta distancia, que a diferencia del Bluetooth clásico está optimizada para un muy bajo consumo, por ello es muy habitual su uso en aplicaciones del internet de las cosas. Su bajo consumo reside en la minimización de la potencia de transmisión de la señal de radio utilizada y el radio de cobertura.

Asimismo, en este proyecto se ha desarrollado un único nodo registrador de datos, pero basándonos en la actualidad y aplicaciones del internet de las cosas sería muy interesante la posibilidad de crear una red inalámbrica de múltiples nodos para monitorizar diferentes zonas de una sala, laboratorio, campo, etc.

La adaptación del diseño realizado sería sencillo. Cada nodo tendría asignado un identificador, permitiéndonos acceder al nodo deseado en cada momento desde la aplicación móvil. A continuación se muestra una ilustración como ejemplo de aplicación de la red de nodos.

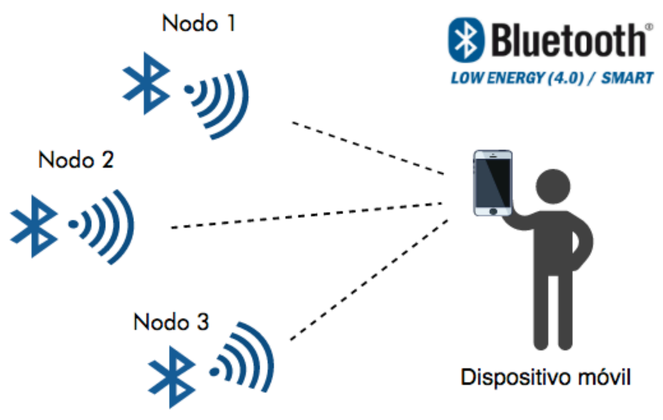


Figura 32: Ejemplo de una red de tres nodos conectados a un dispositivo móvil

Bibliografía

[1] *What is a datalogger?*. From onsetcomp: http://www.onsetcomp.com/what-is-a-data-logger

[2] *Registrador de datos*. From Wikipedia: https://es.wikipedia.org/wiki/Registrador_de_datos

[3] *TI Launchpad*. From Texas Instruments: http://www.ti.com/ww/en/launchpad/launchpads-msp430-msp-exp430g2.html

[4] *HDC1050 Datasheet*. Texas Instruments.

[5] *I2C*. From Sparkfun: https://learn.sparkfun.com/tutorials/i2c

[6] *Bluetooth Basics*. From Sparkfun: https://learn.sparkfun.com/tutorials/bluetooth-basics

[7] *MSP430G2553 Datasheet*. Texas Instruments.

[8] *Energia* . From Github: https://github.com/energia/Energia

[9] *Energia Wire Library*. From Github: https://github.com/energia/Energia/tree/master/libraries/Wire

[10] *Energia Flash Library*. From Github: https://github.com/energia/Energia/tree/master/hardware/msp430/libraries/MspFlash

[11] *Saving Files*. From Android developers: https://developer.android.com/training/basics/data-storage/files.html

[12] *Energia Serial*. From Energia : http://energia.nu/Serial.html

[13] *Bluetooth*. From Android developers: https://developer.android.com/guide/topics/connectivity/bluetooth.html

[14] *File Writer*. From Android developers: https://developer.android.com/reference/java/io/FileWriter.html

Anexos

- Datasheet sensor HDC1050
- Datasheet MSP430G2553
- Librería MyRTC
- Librería MyFlash
- Librería HDC1050
- Código aplicación móvil
- Código Energia

HDC1050 Low Power, High Accuracy Digital Humidity Sensor with Temperature Sensor

1 Features

- Relative Humidity Accuracy ±3% (typical)
- Temperature Accuracy ±0.2°C (typical)
- 14 Bit Measurement Resolution
- 100 nA Sleep Mode Current
- Average Supply Current:
  - 710 nA @ 1sps, 11 bit RH Measurement
  - 1.3 µA @ 1sps, 11 bit RH and Temperature Measurement
- Supply Voltage 2.7 V to 5.5 V
- Small 3 mm x 3 mm Device Footprint
- I^{2}C Interface

2 Applications

- HVAC
- Smart Thermostats and Room Monitors
- White Goods
- Printers
- Handheld Meters
- Medical Devices
- Wireless Sensor (TIDA-00374)

3 Description

The HDC1050 is a digital humidity sensor with integrated temperature sensor that provides excellent measurement accuracy at very low power. The HDC1050 operates over a wide supply range, and is a low cost, low power alternative to competitive solutions in a wide range of common applications. The humidity and temperature sensors are factory calibrated.

Device Information (1)

PART NUMBER	PACKAGE	BODY SIZE (NOM)
HDC1050	PWSON (6-pin) DMB	3.00 mm x 3.00 mm

(1) For all available packages, see the orderable addendum at the end of the datasheet.

4 Typical Application

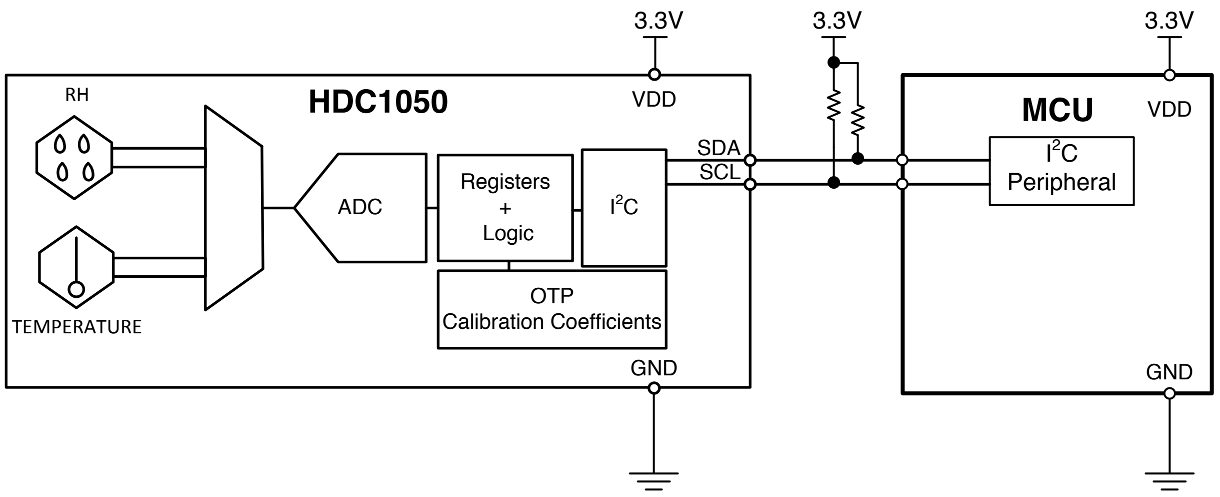


Table of Contents

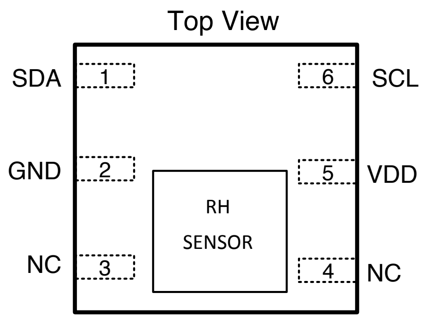
5 Revision History

Changes from Revision A (July 2015) to Revision B	Page
• Added range for Temp sensor only	4
• Added operating range for RH sensor	4
• Added RH condition	5
• Added RH range condition	5
• Added long term drift spec	5
• Changed recommended operating range.	5
• Changed Figure 2	7
• Changed soldering recovery temperature	18

Changes from Original (July 2015) to Revision A	Page
• Added product preview to full datasheet	1

6 Pin Configuration and Functions

DMB Package
6 Pin PWSON
Top View



Pin Functions

PIN		I/O TYPE(1)	DESCRIPTION
NAME	NO.		
SDA	1	I/O	Serial data line for I2C, open-drain; requires a pull-up resistor to VDD
GND	2	G	Ground
NC	3,4	-	These pins may be left floating, or connected to GND
VDD	5	P	Supply Voltage
SCL	6	I	Serial clock line for I2C, open-drain; requires a pull-up resistor to VDD
DAP	DAP	G	Die Attach Pad. Should be left floating. (On bottom of the device, not shown in the figure)

(1) P=Power, G=Ground, I=Input, O=Output

7 Specifications

7.1 Absolute Maximum Ratings[1]

		MIN	MAX	UNIT
Input Voltage	VDD	-0.3	6	V
	SCL	-0.3	6	
	SDA	-0.3	6	
Storage Temperature	T_{STG}	-65	150	°C

(1) Stresses beyond those listed under *Absolute Maximum Ratings* may cause permanent damage to the device. These are stress ratings only, which do not imply functional operation of the device at these or any other conditions beyond those indicated under *Recommended Operating Conditions*. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

7.2 ESD Ratings

			VALUE	UNIT
$V_{(ESD)}$	Electrostatic discharge	Human-body model (HBM), per ANSI/ESDA/JEDEC JS-001[1]	±2000	V
		Charged-device model (CDM), per JEDEC specification JESD22-C101[2]	±500	

(1) JEDEC document JEP155 states that 500-V HBM allows safe manufacturing with a standard ESD control process.
(2) JEDEC document JEP157 states that 250-V CDM allows safe manufacturing with a standard ESD control process.

7.3 Recommended Operating Conditions

over operating range (unless otherwise noted)

		MIN	NOM	MAX	UNIT
V_{DD}	Supply Voltage	2.7	3	5.5	V
T_A, Temperature sensor	Ambient Operating Temperature	-40		125	°C
T_A, Humidity sensor	Ambient Operating Temperature	-20		60	°C

7.4 Thermal Information

THERMAL METRIC[1]		HDC1050	UNIT
		PWSON	
		DMB 6 PINS	
$R_{\theta JA}$	Junction-to-Ambient Thermal Resistance	49.4	°C/W

(1) For more information about traditional and new thermal metrics, see the: *IC Package Thermal Metrics* application report, SPRA953.

7.5 Electrical Characteristics(1)

The electrical ratings specified in this section apply to all specifications in this document, unless otherwise noted. T_A = 30°C, RH = 40%, and V_{DD} = 3V.

PARAMETER		TEST CONDITION(2)	MIN(3)	TYP(4)	MAX(3)	UNIT
POWER CONSUMPTION						
I_{DD}	Supply Current	RH measurement, bit 12 of 0x02 register = 0(5)		190	220	µA
		Temperature measurement, bit 12 of 0x02 register = 0(5)		160	185	µA
		Sleep Mode		100	200	nA
		Average @ 1 measurement/second, RH (11 bit), bit 12 of 0x02 register = 0(5)(6)		710		nA
		Average @ 1 measurement/second, Temp (11 bit), bit 12 of 0x02 register = 0(5)(6)		590		nA
		Average @ 1 measurement/second, RH (11bit) +temperature (11 bit), bit 12 of 0x02 register = 1(5)(6)		1.3		µA
		Startup (average on Start-up time)		300		µA
I_{HEAT}	Heater Current(7)	Peak current		7.2		mA
		Average @ 1 measurement/second, RH (11bit) +temperature (11 bit), bit 12 of 0x02 register = 1(5)(6)		50		µA
RELATIVE HUMIDITY SENSOR						
RH_{ACC}	Accuracy	Refer to Figure 2 in Typical Characteristics section.		±3		%RH
RH_{REP}	Repeatability(7)	14 bit resolution		±0.1		%RH
RH_{HYS}	Hysteresis (8)	20% ≤ RH ≤ 60%		±1		%RH
RH_{RT}	Response Time(9)	$t_{63\%}$ (10)		30		s
RH_{CT}	Conversion Time(7)	8 bit resolution		2.50		ms
		11 bit resolution		3.85		ms
		14 bit resolution		6.50		ms
RH_{OR}	Operating Range(11)	Non-condensing	0		100	%RH
RH_{LTD}	Long Term Drift			±0.5		%RH/yr
TEMPERATURE SENSOR						
$TEMP_{ACC}$	Accuracy(7)	5°C < T_A< 60°C		±0.2	±0.4	°C
$TEMP_{REP}$	Repeatability(7)	14 bit resolution		±0.1		°C
$TEMP_{CT}$	Conversion Time(7)	11 bit accuracy		3.65		ms
		14 bit accuracy		6.35		ms

(1) Electrical Characteristics Table values apply only for factory testing conditions at the temperature indicated. Factory testing conditions result in very limited self-heating of the device such that TJ = TA. No guarantee of parametric performance is indicated in the electrical tables under conditions of internal self-heating where TJ > TA. Absolute Maximum Ratings indicate junction temperature limits beyond which the device may be permanently degraded, either mechanically or electrically.
(2) Register values are represented as either binary (b is the prefix to the digits), or hexadecimal (0x is the prefix to the digits). Decimal values have no prefix.
(3) Limits are ensured by testing, design, or statistical analysis at 30°C. Limits over the operating temperature range are ensured through correlations using statistical quality control (SQC) method.
(4) Typical values represent the most likely parametric norm as determined at the time of characterization. Actual typical values may vary over time and will also depend on the application and configuration. The typical values are not tested and are not guaranteed on shipped production material.
(5) I^2C read/write communication and pull-up resistors current through SCL and SDA not included.
(6) Average current consumption while conversion is in progress.
(7) This parameter is specified by design and/or characterization and it is not tested in production.
(8) The hysteresis value is the difference between an RH measurement in a rising and falling RH environment, at a specific RH point.
(9) Actual response times will vary dependent on system thermal mass and air-flow.
(10) Time for the RH output to change by 63% of the total RH change after a step change in environmental humidity.
(11) Recommended humidity operating range is 20% to 60% RH. Prolonged operation outside this range may result in a measurement offset. The measurement offset will decrease after operating the sensor in this recommended operating range.

7.6 I2C Interface Electrical Characteristics

At T_A=30°C, V_{DD}=3V (unless otherwise noted)

PARAMETER		TEST CONDITION	MIN	TYP	MAX	UNIT
I2C INTERFACE VOLTAGE LEVEL						
VIH	Input High Voltage		0.7xV_{DD}			V
VIL	Input Low Voltage				0.3xV_{DD}	V
VOL	Output Low Voltage	Sink current 3mA			0.4	V
HYS	Hysteresis (1)		0.1xV_{DD}			V
CIN	Input Capacitance on all digital pins			0.5		pF

(1) This parameter is specified by design and/or characterization and it is not tested in production.

7.7 I2C Interface Timing Requirements

PARAMETER		TEST CONDITION	MIN	NOM	MAX	UNIT
I2C INTERFACE VOLTAGE LEVEL						
f_{SCL}	Clock Frequency		10		400	kHz
t_{LOW}	Clock Low Time		1.3			µs
t_{HIGH}	Clock High Time		0.6			µs
t_{SP}	Pulse width of spikes that must be suppressed by the input filter (1)				50	ns
t_{START}	Device Start-up time	From V_{DD} ≥ 2.7 V to ready for a conversion(1)(2)		10	15	ms

(1) This parameter is specified by design and/or characterization and it is not tested in production.
(2) Within this interval it is not possible to communicate to the device.

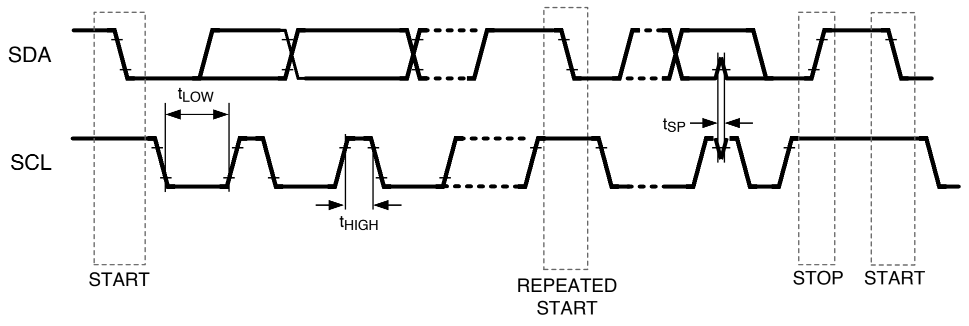


Figure 1. I2C Timing

7.8 Typical Characteristics

Unless otherwise noted. T_A = 30°C, V_{DD} = 3V.

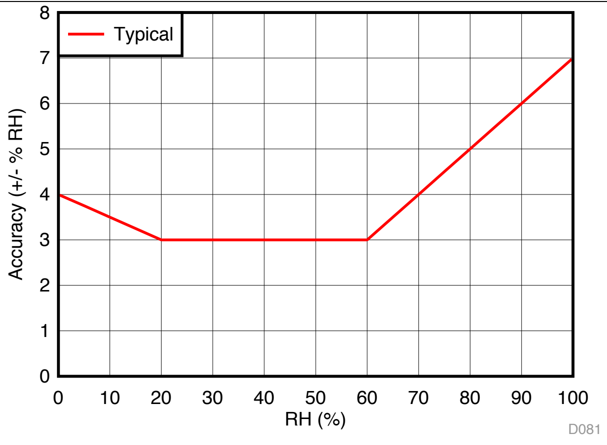


Figure 2. RH Accuracy vs. RH

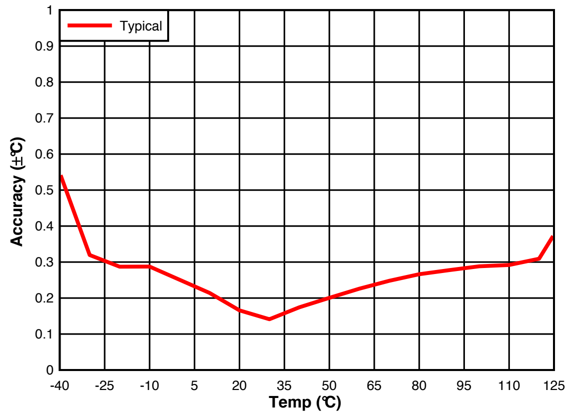


Figure 3. Temperature Accuracy vs. Temperature

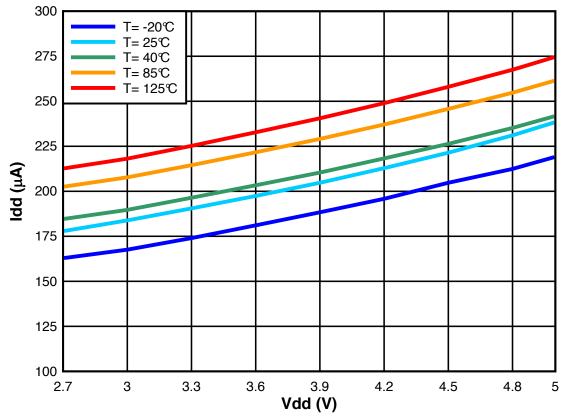


Figure 4. Supply Current vs. Supply Voltage, RH Measurement Active

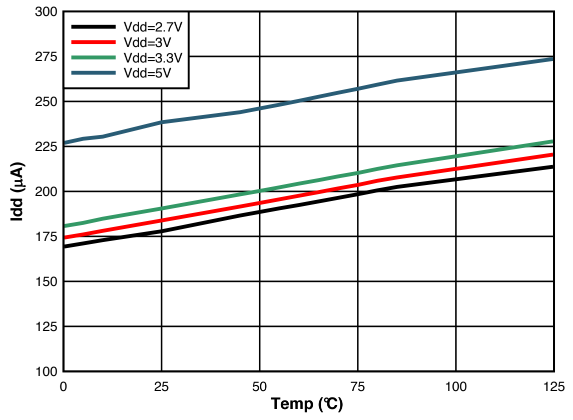


Figure 5. Supply Current vs. Temperature, RH Measurement Active

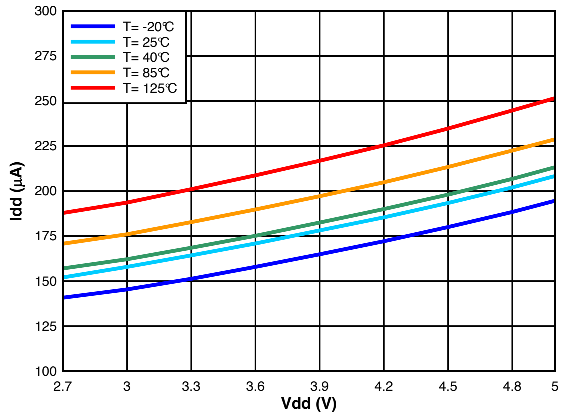


Figure 6. Supply Current vs. Supply Voltage, Temp Measurement Active

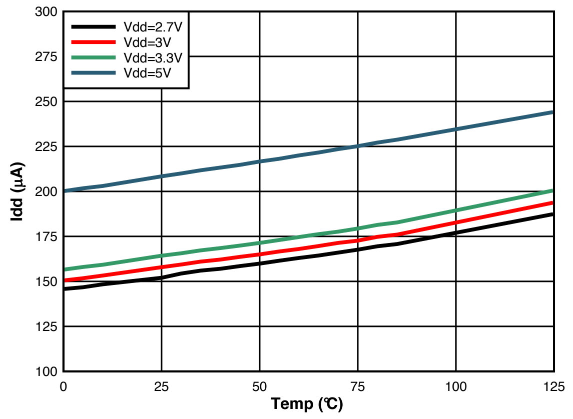


Figure 7. Supply Current vs. Temperature, Temp Measurement Active

Typical Characteristics (continued)

Unless otherwise noted. T_A = 30°C, V_{DD} = 3V.

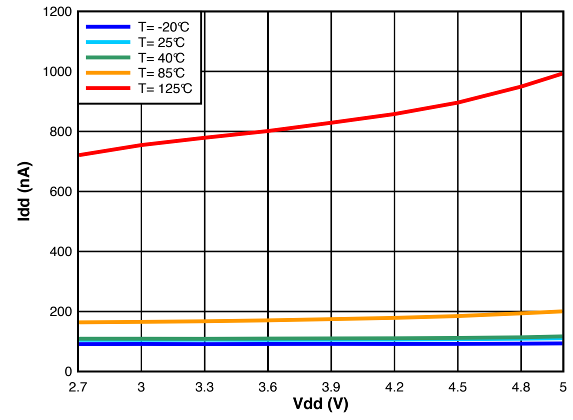


Figure 8. Supply Current vs. Supply Voltage, Sleep Mode

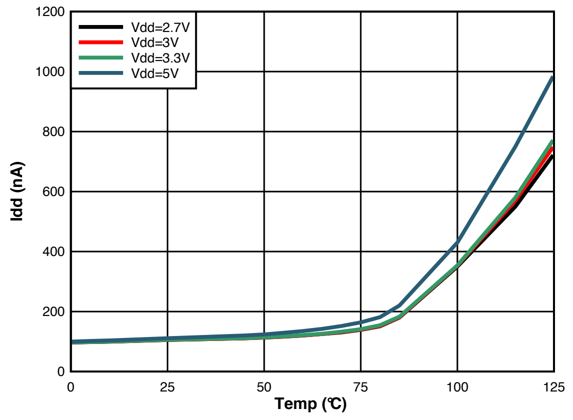


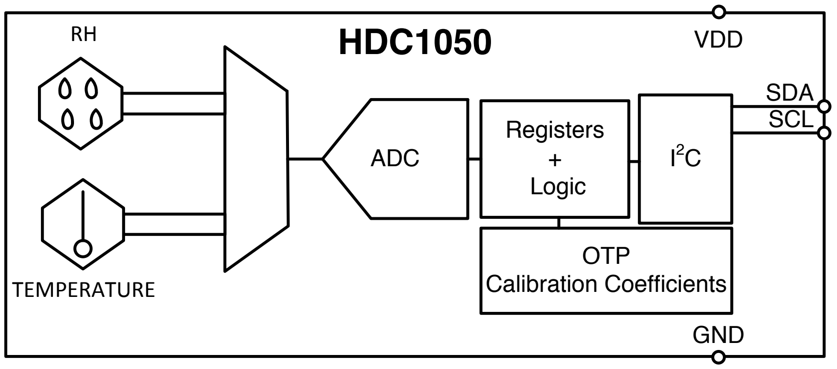
Figure 9. Supply Current vs. Temperature, Sleep Mode

8 Detailed Description

8.1 Overview

The HDC1050 is a digital humidity sensor with integrated temperature sensor that provides excellent measurement accuracy at very low power. The sensing element of the HDC1050 is placed on the top part of the device. Measurement results can be read out through the I2C compatible interface. Resolution is based on the measurement time and can be 8, 11, or 14 bits for humidity; 11 or 14 bits for temperature.

8.2 Functional Block Diagram



8.3 Feature Description

8.3.1 Power Consumption

One of the key features of the HDC1050 is its low power consumption, which makes the device suitable in battery or power harvesting applications. In these applications the HDC1050 spends most of the time in sleep mode: with a typical 100nA of current consumption in sleep mode, the averaged current consumption is minimal. Its low consumption in measurement mode minimizes any self-heating.

8.3.2 Voltage Supply Monitoring

The HDC1050 monitors the supply voltage level and indicates when the voltage supply of the HDC1050 is less than 2.8V. This information is useful in battery-powered systems in order to inform the user to replace the battery. This is reported in the BTST field (register address 0x02:bit[11]) which is updated after POR and after each measurement request.

8.3.3 Heater

The heater is an integrated resistive element that can be used to test the sensor or to drive condensation off the sensor. The heater can be activated using HEAT, bit 13 in the Configuration Register. The heater helps in reducing the accumulated offset after long exposure at high humidity conditions.

Once enabled the heater is turned on only in the measurement mode. To accelerate the temperature increase it is suggested to increase the measurement data rate.

8.4 Device Functional Modes

The HDC1050 has two modes of operation: sleep mode and measurement mode. After power up, the HDC1050 is in sleep mode. In this mode, the HDC1050 waits for I2C input including commands to configure the conversion times, read the status of the battery, trigger a measurement, and read measurements. Once it receives a command to trigger a measurement, the HDC1050 moves from sleep mode to measurement mode. After completing the measurement the HDC1050 returns to sleep mode.

8.5 Programming

8.5.1 I2C Interface

The HDC1050 operates only as a slave device on the I2C bus interface. It is not allowed to have on the I2C bus multiple devices with the same address. Connection to the bus is made via the open-drain I/O lines, SDA, and SCL. The SDA and SCL pins feature integrated spike-suppression filters and Schmitt triggers to minimize the effects of input spikes and bus noise. After power-up, the sensor needs at most 15 ms, to be ready to start RH and temperature measurement. During this power-up time the HDC1050 is only able to provide the content of the serial number registers (0xFB to 0xFF) if requested. After the power-up the sensor is in the sleep mode until a communication or measurement is performed. All data bytes are transmitted MSB first.

8.5.1.1 Serial Bus Address

To communicate with the HDC1050, the master must first address slave devices via a slave address byte. The slave address byte consists of seven address bits, and a direction bit that indicates the intent to execute a read or write operation. The I2C address of the HDC1050 is 1000000 (7-bit address).

8.5.1.2 Read and Write Operations

To access a particular register on the HDC1050, write the desired register address value to the Pointer Register. The pointer value is the first byte transferred after the slave address byte with the R/W bit low. Every write operation to the HDC1050 requires a value for the pointer register (refer to Figure 10).

When reading from the HDC1050, the last value stored in the pointer by a write operation is used to determine which register is accessed by a read operation. To change the pointer register for a read operation, a new value must be written to the pointer register. This transaction is accomplished by issuing the slave address byte with the R/W bit low, followed by the pointer byte. No additional data is required (refer to Figure 11).

The master can then generate a START condition and send the slave address byte with the R/W bit high to initiate the read command. Note that register bytes are sent MSB first, followed by the LSB. A write operation in a read-only register such as (DEVICE ID, MANUFACTURER ID, SERIAL ID) returns a NACK after each data byte; read/write operation to unused address returns a NACK after the pointer; a read/write operation with incorrect I2C address returns a NACK after the I2C address.

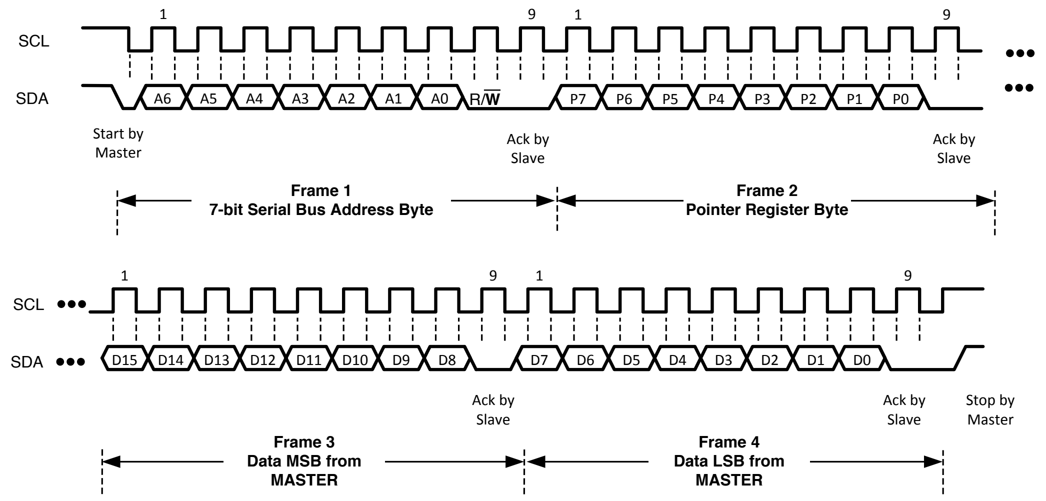


Figure 10. Writing Frame (Configuration Register)

Programming (continued)

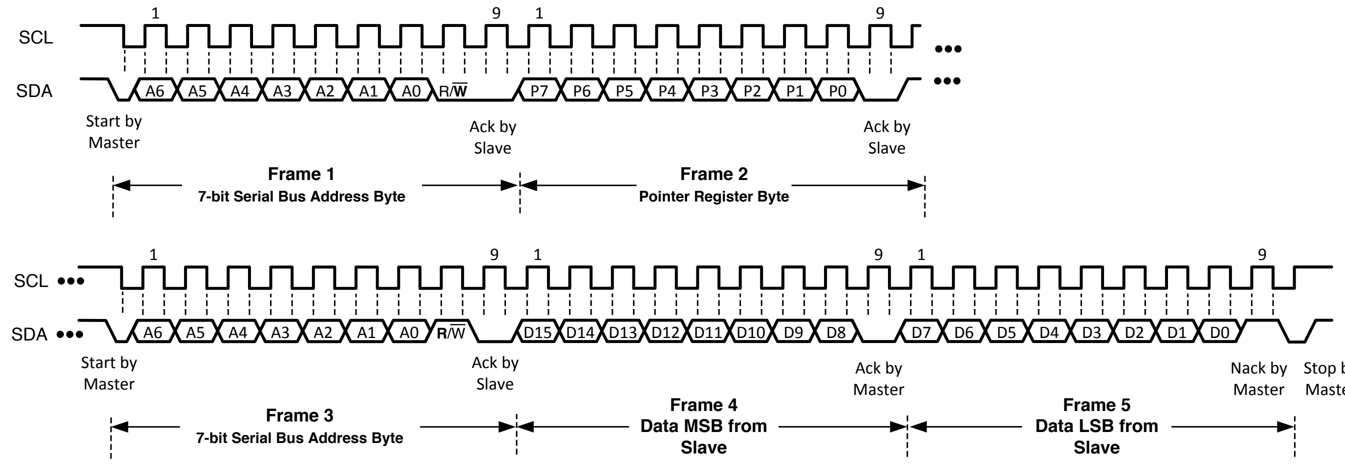


Figure 11. Reading Frame (Configuration Register)

8.5.1.3 Device Measurement Configuration

By default the HDC1050 will first perform a temperature measurement followed by a humidity measurement. On power-up, the HDC1050 enters a low power sleep mode and is not actively measuring. Use the following steps to perform a measurement of both temperature and humidity and then retrieve the results:

1. Configure the acquisition parameters in register address 0x02:
   (a) Set the acquisition mode to measure both temperature and humidity by setting Bit[12] to 1.
   (b) Set the desired temperature measurement resolution:
      – Set Bit[10] to 0 for 14 bit resolution.
      – Set Bit[10] to 1 for 11 bit resolution.
   (c) Set the desired humidity measurement resolution:
      – Set Bit[9:8] to 00 for 14 bit resolution.
      – Set Bit[9:8] to 01 for 11 bit resolution.
      – Set Bit[9:8] to 10 for 8 bit resolution.
2. Trigger the measurements by executing a pointer write transaction with the address pointer set to 0x00. Refer to Figure 12.
3. Wait for the measurements to complete, based on the conversion time (refer to *Electrical Characteristics*(1) for the conversion time).
4. Read the output data:

   Read the temperature data from register address 0x00, followed by the humidity data from register address 0x01 in a single transaction as shown in Figure 14. A read operation will return a NACK if the contents of the registers have not been updated as shown in Figure 13.

To perform another acquisition with the same measurement configuration simply repeat steps 2 through 4.

If only a humidity or temperature measurement is desired, the following steps will perform a measurement and retrieve the result:

1. Configure the acquisition parameters in register address 0x02:
   (a) Set the acquisition mode to independently measure temperature or humidity by setting Bit[12] to 0.
   (b) For a temperature measurement, set the desired temperature measurement resolution:
      – Set Bit[10] to 0 for 14 bit resolution.

(1) Electrical Characteristics Table values apply only for factory testing conditions at the temperature indicated. Factory testing conditions result in very limited self-heating of the device such that TJ = TA. No guarantee of parametric performance is indicated in the electrical tables under conditions of internal self-heating where TJ > TA. Absolute Maximum Ratings indicate junction temperature limits beyond which the device may be permanently degraded, either mechanically or electrically.

Programming (continued)

- Set Bit[10] to 1 for 11 bit resolution.

(c) For a humidity measurement, set the desired humidity measurement resolution:
- Set Bit[9:8] to 00 for 14 bit resolution.
- Set Bit[9:8] to 01 for 11 bit resolution.
- Set Bit[9:8] to 10 for 8 bit resolution.

2. Trigger the measurement by executing a pointer write transaction. Refer to Figure 12
   - Set the address pointer to 0x00 for a temperature measurement.
   - Set the address pointer to 0x01 for a humidity measurement.
3. Wait for the measurement to complete, based on the conversion time (refer to *Electrical Characteristics*[(1)] for the conversion time).
4. Read the output data:

   Retrieve the completed measurement result from register address 0x00 or 0x01, as appropriate, as shown in Figure 11. A read operation will return a NACK if the measurement result is not yet available, as shown in Figure 13.

To perform another acquisition with the same measurement configuration repeat steps 2 through 4.

It is possible to read the output registers (addresses 0x00 and 0x01) during a Temperature or Relative Humidity measurement without affecting any ongoing measurement. Note that a write to address 0x00 or 0x01 while a measurement is ongoing will abort the ongoing measurement.

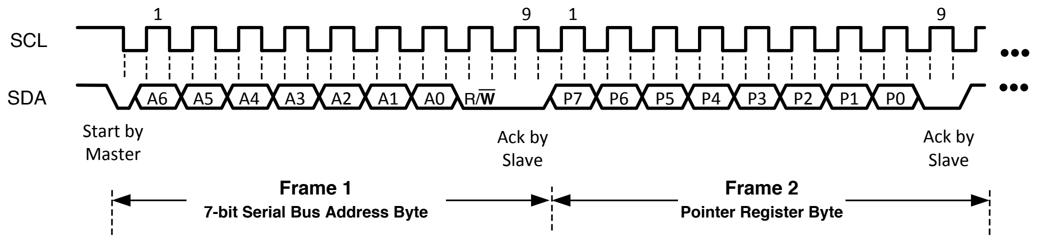


Figure 12. Trigger Humidity/Temperature Measurement

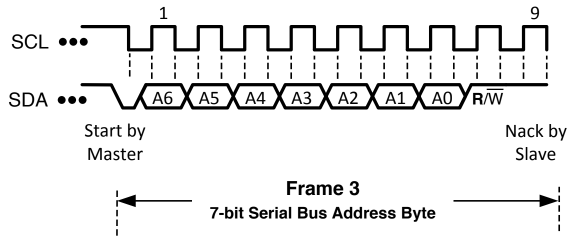


Figure 13. Read Humidity/Temperature Measurement (Data Not Ready)

Programming (continued)

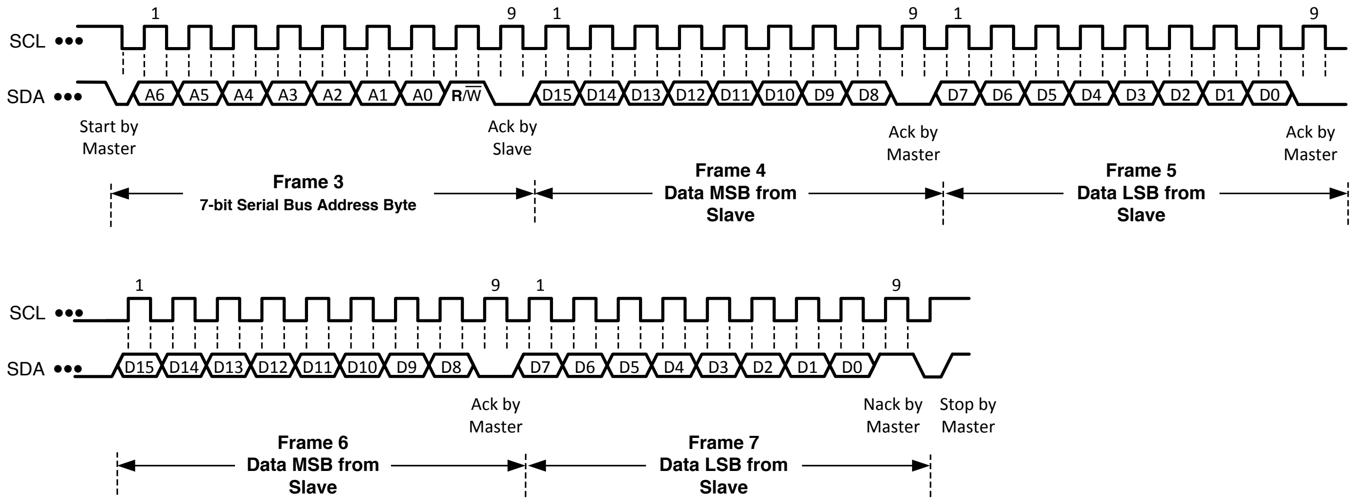


Figure 14. Read Humidity and Temperature Measurement (Data Ready)

8.6 Register Map

The HDC1050 contains data registers that hold configuration information, temperature and humidity measurement results, and status information.

Table 1. Register Map

Pointer	Name	Reset value	Description
0x00	Temperature	0x0000	Temperature measurement output
0x01	Humidity	0x0000	Relative Humidity measurement output
0x02	Configuration	0x1000	HDC1050 configuration and status
0xFB	Serial ID	device dependent	First 2 bytes of the serial ID of the part
0xFC	Serial ID	device dependent	Mid 2 bytes of the serial ID of the part
0xFD	Serial ID	device dependent	Last byte bit of the serial ID of the part
0xFE	Manufacturer ID	0x5449	ID of Texas Instruments
0xFF	Device ID	0x1050	ID of HDC1050 device

Registers addresses 0x03 to 0xFA are reserved and should not be written.

The HDC1050 has an 8-bit pointer used to address a given data register. The pointer identifies which of the data registers should respond to a read or write command on the two-wire bus. This register is set with every write command. A write command must be issued to set the proper value in the pointer before executing a read command. The power-on reset (POR) value of the pointer is 0x00, which selects a temperature measurement.

8.6.1 Temperature Register

The temperature register is a 16-bit result register in binary format (the 2 LSBs D1 and D0 are always 0). The result of the acquisition is always a 14 bit value. The accuracy of the result is related to the selected conversion time (refer to *Electrical Characteristics*(1)). The temperature can be calculated from the output data with:

$$\text{Temperature}(^\circ\text{C}) = \left(\frac{\text{TEMPERATURE}[15:00]}{2^{16}}\right) * 165^\circ\text{C} - 40^\circ\text{C}$$

Table 2. Temperature Register Description (0x00)

Name	Bits		Description
TEMPERATURE	[15:02]	Temperature	Temperature measurement (read only)
	[01:00]	Reserved	Reserved, always 0 (read only)

(1) Electrical Characteristics Table values apply only for factory testing conditions at the temperature indicated. Factory testing conditions result in very limited self-heating of the device such that TJ = TA. No guarantee of parametric performance is indicated in the electrical tables under conditions of internal self-heating where TJ > TA. Absolute Maximum Ratings indicate junction temperature limits beyond which the device may be permanently degraded, either mechanically or electrically.

8.6.2 Humidity Register

The humidity register is a 16-bit result register in binary format (the 2 LSBs D1 and D0 are always 0). The result of the acquisition is always a 14 bit value, while the accuracy is related to the selected conversion time (refer to *Electrical Characteristics*(1)). The humidity can be calculated from the output data with:

$$\text{Relative Humidity}(\%\ \text{RH}) = \left(\frac{\text{HUMIDITY}[15:00]}{2^{16}}\right) * 100\%\text{RH}$$

Table 3. Humidity Register Description (0x01)

Name	Bits		Description
HUMIDITY	[15:02]	Relative Humidity	Relative Humidity measurement (read only)
	[01:00]	Reserved	Reserved, always 0 (read only)

(1) Electrical Characteristics Table values apply only for factory testing conditions at the temperature indicated. Factory testing conditions result in very limited self-heating of the device such that TJ = TA. No guarantee of parametric performance is indicated in the electrical tables under conditions of internal self-heating where TJ > TA. Absolute Maximum Ratings indicate junction temperature limits beyond which the device may be permanently degraded, either mechanically or electrically.

8.6.3 Configuration Register

This register configures device functionality and returns status.

Table 4. Configuration Register Description (0x02)

NAME	Bits	DESCRIPTION		
RST	[15]	Software reset bit	0	Normal Operation, this bit self clears
			1	Software Reset
Reserved	[14]	Reserved	0	Reserved, must be 0
HEAT	[13]	Heater	0	Heater Disabled
			1	Heater Enabled
MODE	[12]	Mode of acquisition	0	Temperature or Humidity is acquired.
			1	Temperature and Humidity are acquired in sequence, Temperature first.
BTST	[11]	Battery Status	0	Battery voltage > 2.8V (read only)
			1	Battery voltage < 2.8V (read only)
TRES	[10]	Temperature Measurement Resolution	0	14 bit
			1	11 bit
HRES	[9:8]	Humidity Measurement Resolution	00	14 bit
			01	11 bit
			10	8 bit
Reserved	[7:0]	Reserved	0	Reserved, must be 0

8.6.4 Serial Number Registers

These registers contain a 40bit unique serial number for each individual HDC1050.

Table 5. Serial Number Register Description (0xFB)

Name	Bits	Description	
SERIAL ID[40:25]	[15:0]	Serial Id bits	Device Serial Number bits from 40 to 25 (read only)

Table 6. Serial Number Register Description (0xFC)

Name	Bits	Description	
SERIAL ID[24:9]	[15:0]	Serial Id bits	Device Serial Number bits from 24 to 9(read only)

Table 7. Serial Number Register Description (0xFD)

Name	Bits	Description	
SERIAL ID[8:0]	[15:7]	Serial Id bits	Device Serial Number bits from 8 to 0 (read only)
	[6:0]	Reserved	Reserved, always 0 (read only)

8.6.5 Manufacturer ID Register

This register contains a factory-programmable identification value that identifies this device as being manufactured by Texas Instruments. This register distinguishes this device from other devices that are on the same I2C bus. The manufacturer ID reads 0x5449.

Table 8. Manufacturer ID Register Description (0xFE)

Name	Bits	Description		
MANUFACTURER ID	[15:0]	Manufacturer ID	0x5449	Texas instruments ID (read only)

8.6.6 Device Register ID

This register contains a factory-programmable identification value that identifies this device as a HDC1050. This register distinguishes this device from other devices that are on the same I2C bus. The Device ID for the HDC1050 is 0x1050.

Table 9. Device ID Register Description (0xFF)

Name	Bits	Description		
DEVICE ID	[15:0]	Device ID	0x1050	HDC1050 Device ID (read only)

MSP430x2xx Family

User's Guide

Literature Number: SLAU144J
December 2004–Revised July 2013

Flash Memory Controller

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

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.

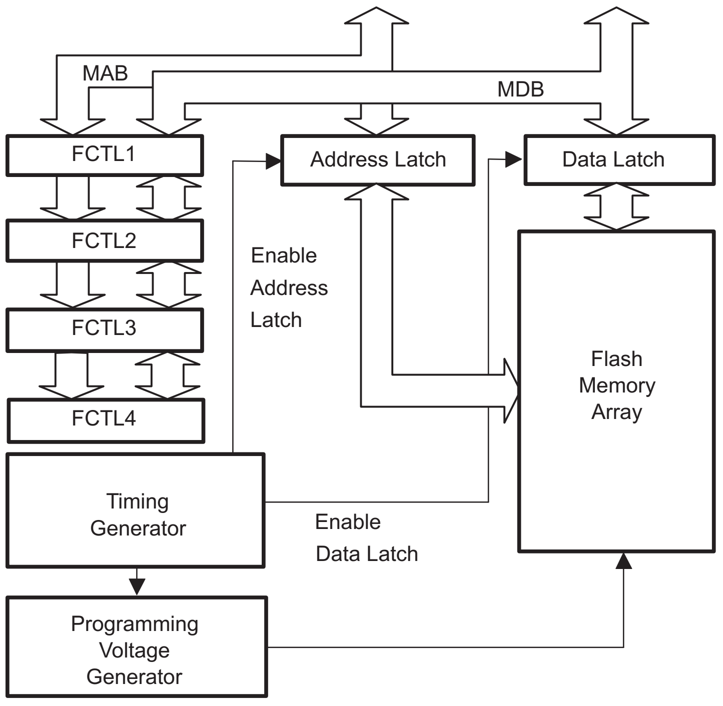


Figure 7-1. Flash Memory Module Block Diagram

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.

Figure 7-2 shows the flash segmentation using an example of 32-KB flash that has eight main segments and four information segments.

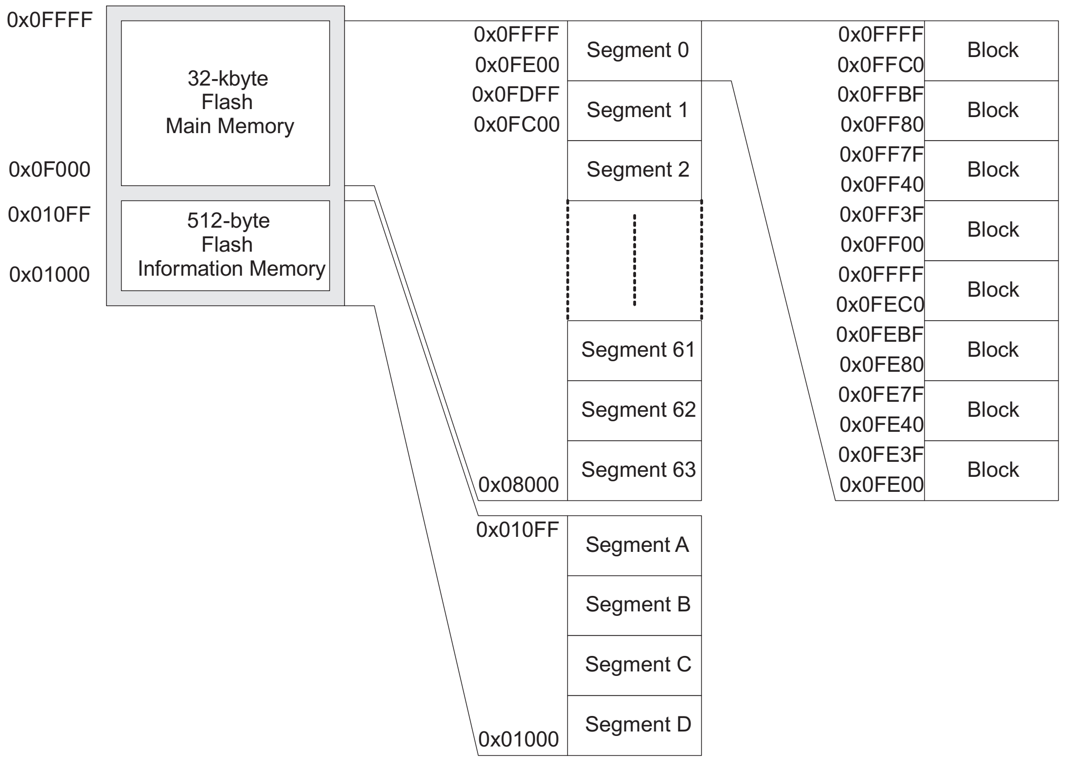


Figure 7-2. Flash Memory Segments, 32-KB Example

7.2.1 SegmentA

SegmentA of the information memory is locked separately from all other segments with the LOCKA bit. When LOCKA = 1, SegmentA cannot be written or erased and all information memory is protected from erasure during a mass erase or production programming. When LOCKA = 0, SegmentA can be erased and written as any other flash memory segment, and all information memory is erased during a mass erase or production programming.

The state of the LOCKA bit is toggled when a 1 is written to it. Writing a 0 to LOCKA has no effect. This allows existing flash programming routines to be used unchanged.

```
; Unlock SegmentA
   BIT   #LOCKA,&FCTL3           ; Test LOCKA
   JZ    SEGA_UNLOCKED           ; Already unlocked?
   MOV   #FWKEY+LOCKA,&FCTL3     ; No, unlock SegmentA
SEGA_UNLOCKED                    ; Yes, continue
; SegmentA is unlocked

; Lock SegmentA
   BIT   #LOCKA,&FCTL3           ; Test LOCKA
   JNZ   SEGA_LOCKED             ; Already locked?
   MOV   #FWKEY+LOCKA,&FCTL3     ; No, lock SegmentA
SEGA_LOCKED                      ; Yes, continue
; SegmentA is locked
```

7.3 Flash Memory Operation

The default mode of the flash memory is read mode. In read mode, the flash memory is not being erased or written, the flash timing generator and voltage generator are off, and the memory operates identically to ROM.

MSP430 flash memory is in-system programmable (ISP) without the need for additional external voltage. The CPU can program its own flash memory. The flash memory write and erase modes are selected with the BLKWRT, WRT, MERAS, and ERASE bits and are:

- Byte or word write
- Block write
- Segment erase
- Mass erase (all main memory segments)
- All erase (all segments)

Reading from or writing to flash memory while it is being programmed or erased is prohibited. If CPU execution is required during the write or erase, the code to be executed must be in RAM. Any flash update can be initiated from within flash memory or RAM.

7.3.1 Flash Memory Timing Generator

Write and erase operations are controlled by the flash timing generator shown in Figure 7-3. The flash timing generator operating frequency, f_{FTG}, must be in the range from approximately 257 kHz to approximately 476 kHz (see device-specific data sheet).

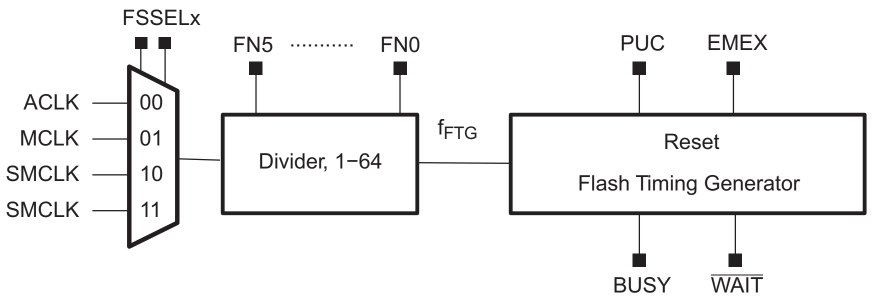


Figure 7-3. Flash Memory Timing Generator Block Diagram

7.3.1.1 Flash Timing Generator Clock Selection

The flash timing generator can be sourced from ACLK, SMCLK, or MCLK. The selected clock source should be divided using the FNx bits to meet the frequency requirements for f_{FTG}. If the f_{FTG} frequency deviates from the specification during the write or erase operation, the result of the write or erase may be unpredictable, or the flash memory may be stressed above the limits of reliable operation.

If a clock failure is detected during a write or erase operation, the operation is aborted, the FAIL flag is set, and the result of the operation is unpredictable.

While a write or erase operation is active the selected clock source can not be disabled by putting the MSP430 into a low-power mode. The selected clock source remains active until the operation is completed before being disabled.

7.3.2 Erasing Flash Memory

The erased level of a flash memory bit is 1. Each bit can be programmed from 1 to 0 individually but to reprogram from 0 to 1 requires an erase cycle. The smallest amount of flash that can be erased is a segment. There are three erase modes selected with the ERASE and MERAS bits listed in Table 7-1.

Table 7-1. Erase Modes

MERAS	ERASE	Erase Mode
0	1	Segment erase
1	0	Mass erase (all main memory segments)
1	1	LOCKA = 0: Erase main and information flash memory. LOCKA = 1: Erase only main flash memory.

Any erase is initiated by a dummy write into the address range to be erased. The dummy write starts the flash timing generator and the erase operation. Figure 7-4 shows the erase cycle timing. The BUSY bit is set immediately after the dummy write and remains set throughout the erase cycle. BUSY, MERAS, and ERASE are automatically cleared when the cycle completes. The erase cycle timing is not dependent on the amount of flash memory present on a device. Erase cycle times are equivalent for all MSP430F2xx and MSP430G2xx devices.

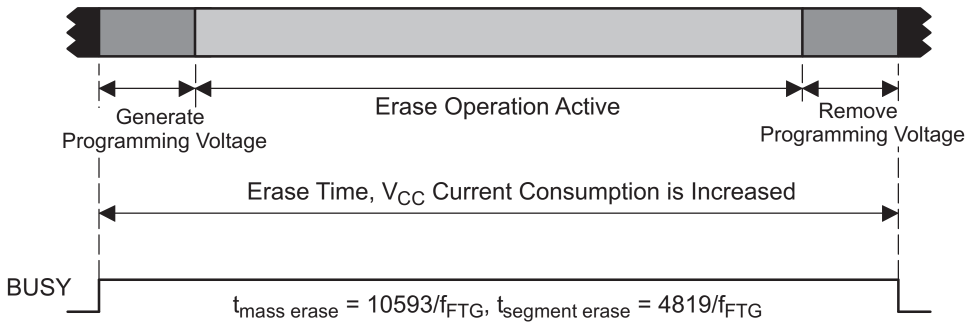

Generate Programming Voltage
Erase Operation Active
Remove Programming Voltage
Erase Time, V_{CC} Current Consumption is Increased
BUSY
$t_{mass\ erase}$ = 10593/f_{FTG}, $t_{segment\ erase}$ = 4819/f_{FTG}

Figure 7-4. Erase Cycle Timing

A dummy write to an address not in the range to be erased does not start the erase cycle, does not affect the flash memory, and is not flagged in any way. This errant dummy write is ignored.

7.3.2.1 Initiating an Erase from Within Flash Memory

Any erase cycle can be initiated from within flash memory or from RAM. When a flash segment erase operation is initiated from within flash memory, all timing is controlled by the flash controller, and the CPU is held while the erase cycle completes. After the erase cycle completes, the CPU resumes code execution with the instruction following the dummy write.

When initiating an erase cycle from within flash memory, it is possible to erase the code needed for execution after the erase. If this occurs, CPU execution is unpredictable after the erase cycle.

The flow to initiate an erase from flash is shown in Figure 7-5.

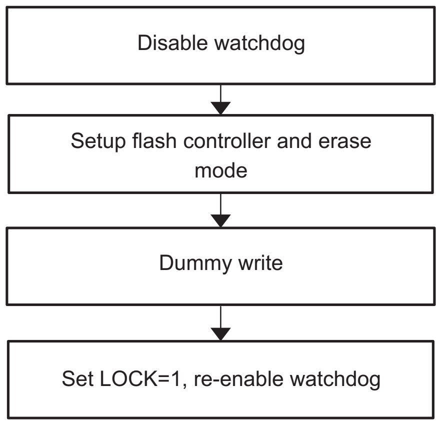

Disable watchdog
Setup flash controller and erase mode
Dummy write
Set LOCK=1, re-enable watchdog

Figure 7-5. Erase Cycle from Within Flash Memory

```
; Segment Erase from flash. 514 kHz < SMCLK < 952 kHz
; Assumes ACCVIE = NMIIE = OFIE = 0.
   MOV      #WDTPW+WDTHOLD,&WDTCTL     ; Disable WDT
   MOV      #FWKEY+FSSEL1+FN0,&FCTL2   ; SMCLK/2
   MOV      #FWKEY, &FCTL3             ; Clear LOCK
   MOV      #FWKEY+ERASE, &FCTL1       ; Enable segment erase
   CLR      &0FC10h                    ; Dummy write, erase S1
   MOV      #FWKEY+LOCK, &FCTL3        ; Done, set LOCK
   ...                                 ; Re-enable WDT?
```

7.3.2.2 Initiating an Erase from RAM

Any erase cycle may be initiated from RAM. In this case, the CPU is not held and can continue to execute code from RAM. The BUSY bit must be polled to determine the end of the erase cycle before the CPU can access any flash address again. If a flash access occurs while BUSY = 1, it is an access violation, ACCVIFG is set, and the erase results are unpredictable.

The flow to initiate an erase from flash from RAM is shown in Figure 7-6.

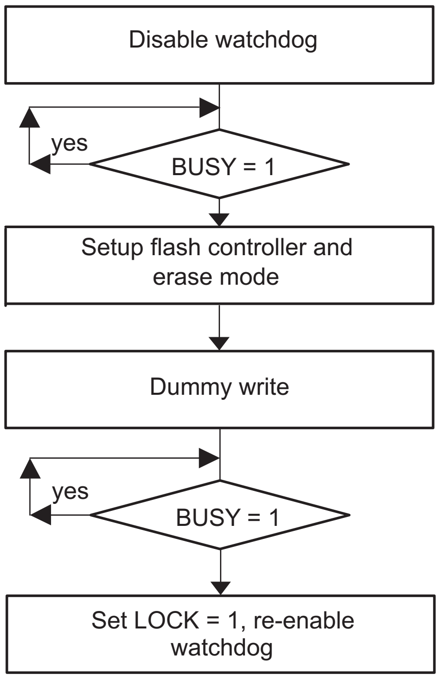

Disable watchdog
yes
BUSY = 1
Setup flash controller and erase mode
Dummy write
yes
BUSY = 1
Set LOCK = 1, re-enable watchdog

Figure 7-6. Erase Cycle from Within RAM

```
; Segment Erase from RAM. 514 kHz < SMCLK < 952 kHz
; Assumes ACCVIE = NMIIE = OFIE = 0.
     MOV   #WDTPW+WDTHOLD,&WDTCTL      ; Disable WDT
L1   BIT   #BUSY, &FCTL3               ; Test BUSY
     JNZ   L1                          ; Loop while busy
     MOV   #FWKEY+FSSEL1+FN0, &FCTL2   ; SMCLK/2
     MOV   #FWKEY&FCTL3                ; Clear LOCK
     MOV   #FWKEY+ERASE, &FCTL1        ; Enable erase
     CLR   &0FC10h                     ; Dummy write, erase S1
L2   BIT   #BUSY, &FCTL3               ; Test BUSY
     JNZ   L2                          ; Loop while busy
     MOV   #FWKEY+LOCK&FCTL3           ; Done, set LOCK
     ...                               ; Re-enable WDT?
```

7.3.3 Writing Flash Memory

The write modes, selected by the WRT and BLKWRT bits, are listed in Table 7-2.

Table 7-2. Write Modes

BLKWRT	WRT	Write Mode
0	1	Byte or word write
1	1	Block write

Both write modes use a sequence of individual write instructions, but using the block write mode is approximately twice as fast as byte or word mode, because the voltage generator remains on for the complete block write. Any instruction that modifies a destination can be used to modify a flash location in either byte or word write mode or block write mode. A flash word (low and high bytes) must not be written more than twice between erasures. Otherwise, damage can occur.

The BUSY bit is set while a write operation is active and cleared when the operation completes. If the write operation is initiated from RAM, the CPU must not access flash while BUSY = 1. Otherwise, an access violation occurs, ACCVIFG is set, and the flash write is unpredictable.

7.3.3.1 Byte or Word Write

A byte or word write operation can be initiated from within flash memory or from RAM. When initiating from within flash memory, all timing is controlled by the flash controller, and the CPU is held while the write completes. After the write completes, the CPU resumes code execution with the instruction following the write. The byte or word write timing is shown in Figure 7-7.

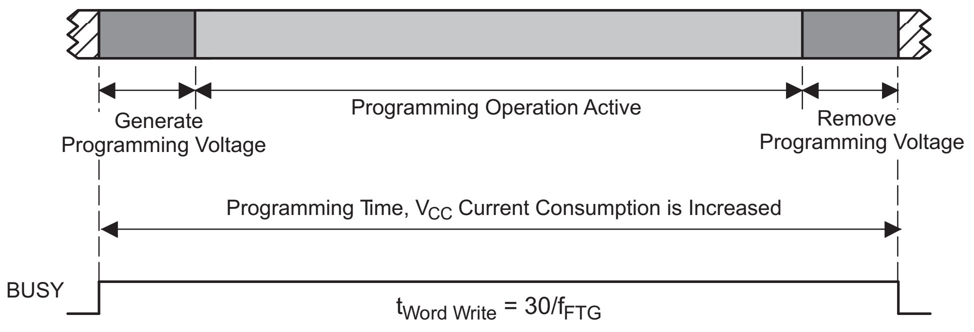

Generate Programming Voltage
Programming Operation Active
Remove Programming Voltage
Programming Time, V_{CC} Current Consumption is Increased
BUSY
$t_{Word\ Write} = 30/f_{FTG}$

Figure 7-7. Byte or Word Write Timing

When a byte or word write is executed from RAM, the CPU continues to execute code from RAM. The BUSY bit must be zero before the CPU accesses flash again, otherwise an access violation occurs, ACCVIFG is set, and the write result is unpredictable.

In byte or word mode, the internally-generated programming voltage is applied to the complete 64-byte block, each time a byte or word is written, for 27 of the 30 f_{FTG} cycles. With each byte or word write, the amount of time the block is subjected to the programming voltage accumulates. The cumulative programming time, t_{CPT}, must not be exceeded for any block. If the cumulative programming time is met, the block must be erased before performing any further writes to any address within the block. See the device-specific data sheet for specifications.

7.3.3.2 Initiating a Byte or Word Write From Within Flash Memory

The flow to initiate a byte or word write from flash is shown in Figure 7-8.

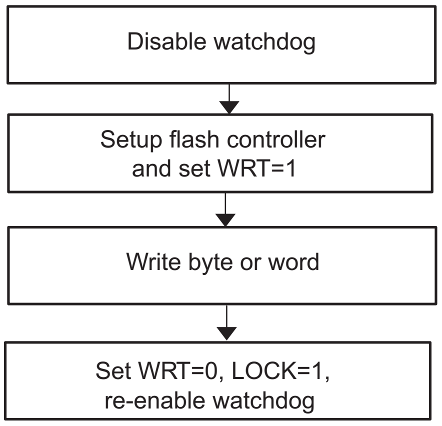

Disable watchdog
Setup flash controller and set WRT=1
Write byte or word
Set WRT=0, LOCK=1, re-enable watchdog

Figure 7-8. Initiating a Byte or Word Write From Flash

```
; Byte/word write from flash. 514 kHz < SMCLK < 952 kHz
; Assumes 0FF1Eh is already erased
; Assumes ACCVIE = NMIIE = OFIE = 0.
   MOV   #WDTPW+WDTHOLD,&WDTCTL     ; Disable WDT
   MOV   #FWKEY+FSSEL1+FN0,&FCTL2   ; SMCLK/2
   MOV   #FWKEY,&FCTL3              ; Clear LOCK
   MOV   #FWKEY+WRT,&FCTL1          ; Enable write
   MOV   #0123h,&0FF1Eh             ; 0123h    -> 0FF1Eh
   MOV   #FWKEY,&FCTL1              ; Done. Clear WRT
   MOV   #FWKEY+LOCK,&FCTL3         ; Set LOCK
   ...                              ; Re-enable WDT?
```

7.3.3.3 Initiating a Byte or Word Write From RAM

The flow to initiate a byte or word write from RAM is shown in Figure 7-9.

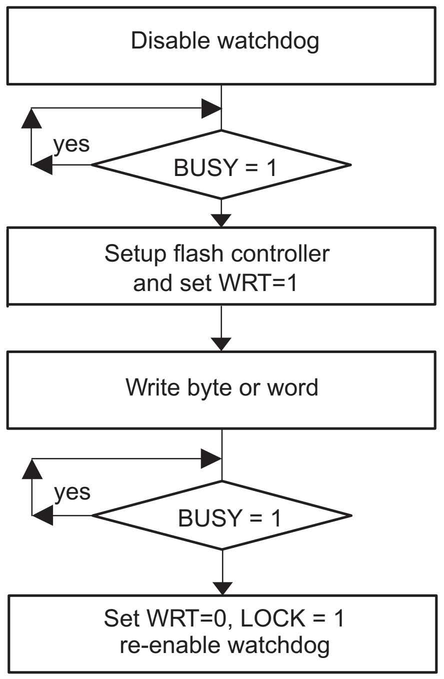

Disable watchdog
yes
BUSY = 1
Setup flash controller and set WRT=1
Write byte or word
yes
BUSY = 1
Set WRT=0, LOCK = 1 re-enable watchdog

Figure 7-9. Initiating a Byte or Word Write from RAM

```
; Byte/word write from RAM. 514 kHz < SMCLK < 952 kHz
; Assumes 0FF1Eh is already erased
; Assumes ACCVIE = NMIIE = OFIE = 0.
     MOV   #WDTPW+WDTHOLD,&WDTCTL      ; Disable WDT
L1   BIT   #BUSY,&FCTL3                ; Test BUSY
     JNZ   L1                          ; Loop while busy
     MOV   #FWKEY+FSSEL1+FN0,&FCTL2    ; SMCLK/2
     MOV   #FWKEY,&FCTL3               ; Clear LOCK
     MOV   #FWKEY+WRT,&FCTL1           ; Enable write
     MOV   #0123h,&0FF1Eh              ; 0123h -> 0FF1Eh
L2   BIT   #BUSY,&FCTL3                ; Test BUSY
     JNZ   L2                          ; Loop while busy
     MOV   #FWKEY,&FCTL1               ; Clear WRT
     MOV   #FWKEY+LOCK,&FCTL3          ; Set LOCK
     ...                               ; Re-enable WDT?
```

7.3.3.4 Block Write

The block write can be used to accelerate the flash write process when many sequential bytes or words need to be programmed. The flash programming voltage remains on for the duration of writing the 64-byte block. The cumulative programming time t_{CPT} must not be exceeded for any block during a block write.

A block write cannot be initiated from within flash memory. The block write must be initiated from RAM only. The BUSY bit remains set throughout the duration of the block write. The WAIT bit must be checked between writing each byte or word in the block. When WAIT is set the next byte or word of the block can be written. When writing successive blocks, the BLKWRT bit must be cleared after the current block is complete. BLKWRT can be set initiating the next block write after the required flash recovery time given by t_{end}. BUSY is cleared following each block write completion indicating the next block can be written. Figure 7-10 shows the block write timing.

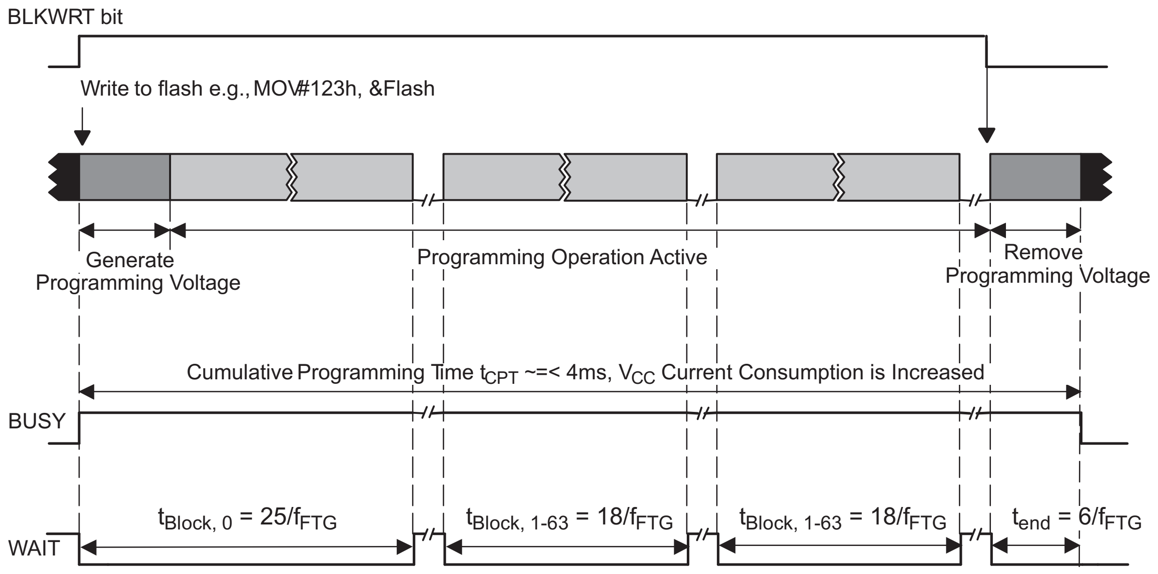

BLKWRT bit
Write to flash e.g., MOV#123h, &Flash
Generate Programming Voltage
Programming Operation Active
Remove Programming Voltage
Cumulative Programming Time t_{CPT} ~=< 4ms, V_{CC} Current Consumption is Increased
BUSY
WAIT
$t_{Block, 0} = 25/f_{FTG}$
$t_{Block, 1-63} = 18/f_{FTG}$
$t_{Block, 1-63} = 18/f_{FTG}$
$t_{end} = 6/f_{FTG}$

Figure 7-10. Block-Write Cycle Timing

7.3.3.5 Block Write Flow and Example

A block write flow is shown in Figure 7-11 and the following example.

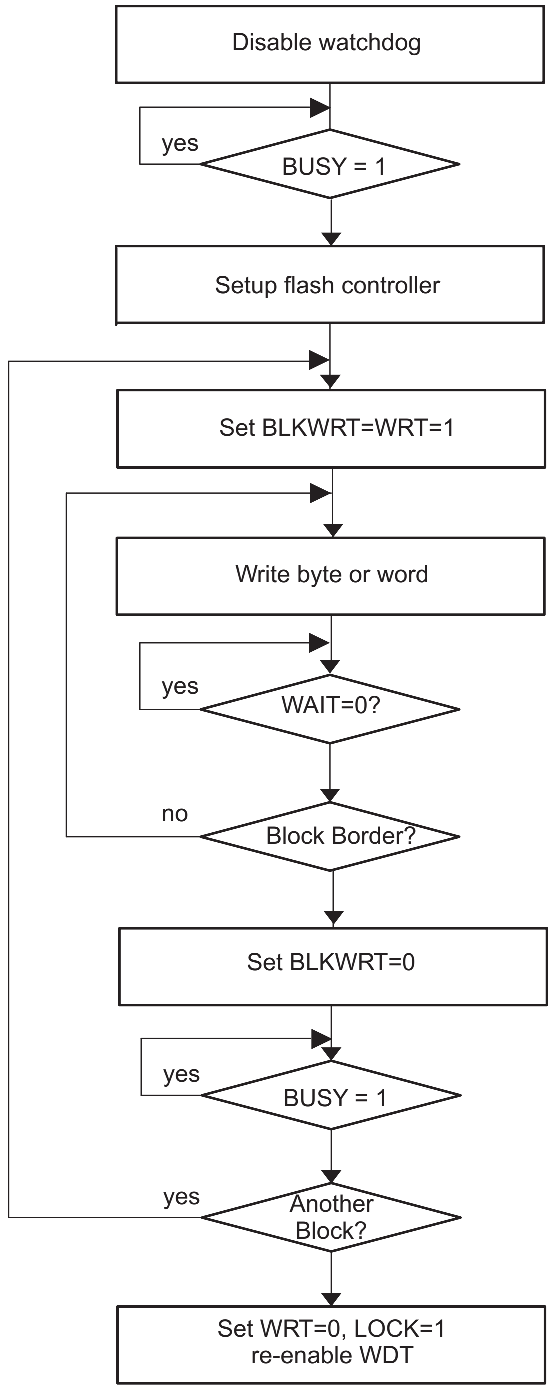


Figure 7-11. Block Write Flow

```
; Write one block starting at 0F000h.
; Must be executed from RAM, Assumes Flash is already erased.
; 514 kHz < SMCLK < 952 kHz
; Assumes ACCVIE = NMIIE = OFIE = 0.
     MOV    #32,R5                      ; Use as write counter
     MOV    #0F000h,R6                  ; Write pointer
     MOV    #WDTPW+WDTHOLD,&WDTCTL      ; Disable WDT
L1   BIT    #BUSY,&FCTL3                ; Test BUSY
     JNZ    L1                          ; Loop while busy
     MOV    #FWKEY+FSSEL1+FN0,&FCTL2    ; SMCLK/2
     MOV    #FWKEY,&FCTL3               ; Clear LOCK
     MOV    #FWKEY+BLKWRT+WRT,&FCTL1    ; Enable block write
L2   MOV    Write_Value,0(R6)           ; Write location
L3   BIT    #WAIT,&FCTL3                ; Test WAIT
     JZ     L3                          ; Loop while WAIT = 0
     INCD   R6                          ; Point to next word
     DEC    R5                          ; Decrement write counter
     JNZ    L2                          ; End of block?
     MOV    #FWKEY,&FCTL1               ; Clear WRT,BLKWRT
L4   BIT    #BUSY,&FCTL3                ; Test BUSY
     JNZ    L4                          ; Loop while busy
     MOV    #FWKEY+LOCK,&FCTL3          ; Set LOCK
     ...                                ; Re-enable WDT if needed
```

7.3.4 Flash Memory Access During Write or Erase

When any write or any erase operation is initiated from RAM and while BUSY = 1, the CPU may not read or write to or from any flash location. Otherwise, an access violation occurs, ACCVIFG is set, and the result is unpredictable. Also if a write to flash is attempted with WRT = 0, the ACCVIFG interrupt flag is set, and the flash memory is unaffected.

When a byte or word write or any erase operation is initiated from within flash memory, the flash controller returns op-code 03FFFh to the CPU at the next instruction fetch. Op-code 03FFFh is the JMP PC instruction. This causes the CPU to loop until the flash operation is finished. When the operation is finished and BUSY = 0, the flash controller allows the CPU to fetch the proper op-code and program execution resumes.

The flash access conditions while BUSY = 1 are listed in Table 7-3.

Table 7-3. Flash Access While BUSY = 1

Flash Operation	Flash Access	WAIT	Result
Any erase, or byte or word write	Read	0	ACCVIFG = 0. 03FFFh is the value read.
	Write	0	ACCVIFG = 1. Write is ignored.
	Instruction fetch	0	ACCVIFG = 0. CPU fetches 03FFFh. This is the JMP PC instruction.
Block write	Any	0	ACCVIFG = 1, LOCK = 1
	Read	1	ACCVIFG = 0. 03FFFh is the value read.
	Write	1	ACCVIFG = 0. Write is written.
	Instruction fetch	1	ACCVIFG = 1, LOCK = 1

Interrupts are automatically disabled during any flash operation when EEI = 0 and EEIEX = 0 and on MSP430x20xx and MSP430G2xx devices where EEI and EEIEX are not present. After the flash operation has completed, interrupts are automatically re-enabled. Any interrupt that occurred during the operation has its associated flag set and generates an interrupt request when re-enabled.

When EEIEX = 1 and GIE = 1, an interrupt immediately aborts any flash operation and the FAIL flag is set. When EEI = 1, GIE = 1, and EEIEX = 0, a segment erase is interrupted by a pending interrupt every 32 f_{FTG} cycles. After servicing the interrupt, the segment erase is continued for at least 32 f_{FTG} cycles or until it is complete. During the servicing of the interrupt, the BUSY bit remains set but the flash memory can be accessed by the CPU without causing an access violation occurs. Nested interrupts and using the RETI instruction inside interrupt service routines are not supported.

The watchdog timer (in watchdog mode) should be disabled before a flash erase cycle. A reset aborts the erase and the results are unpredictable. After the erase cycle has completed, the watchdog may be re-enabled.

7.3.5 Stopping a Write or Erase Cycle

Any write or erase operation can be stopped before its normal completion by setting the emergency exit bit EMEX. Setting the EMEX bit stops the active operation immediately and stops the flash controller. All flash operations cease, the flash returns to read mode, and all bits in the FCTL1 register are reset. The result of the intended operation is unpredictable.

7.3.6 Marginal Read Mode

The marginal read mode can be used to verify the integrity of the flash memory contents. This feature is implemented in selected 2xx devices; see the device-specific data sheet for availability. During marginal read mode marginally programmed flash memory bit locations can be detected. Events that could produce this situation include improper f_{FTG} settings, or violation of minimum V_{CC} during erase or program operations. One method for identifying such memory locations would be to periodically perform a checksum calculation over a section of flash memory (for example, a flash segment) and repeating this procedure with the marginal read mode enabled. If they do not match, it could indicate an insufficiently programmed flash memory location. It is possible to refresh the affected Flash memory segment by disabling marginal read mode, copying to RAM, erasing the flash segment, and writing back to it from RAM.

The program checking the flash memory contents must be executed from RAM. Executing code from flash automatically disables the marginal read mode. The marginal read modes are controlled by the MRG0 and MRG1 register bits. Setting MRG1 is used to detect insufficiently programmed flash cells containing a 1 (erased bits). Setting MRG0 is used to detect insufficiently programmed flash cells containing a 0 (programmed bits). Only one of these bits should be set at a time. Therefore, a full marginal read check requires two passes of checking the flash memory content's integrity. During marginal read mode, the flash access speed (MCLK) must be limited to 1 MHz (see the device-specific data sheet).

7.3.7 Configuring and Accessing the Flash Memory Controller

The FCTLx registers are 16-bit password-protected read/write registers. Any read or write access must use word instructions and write accesses must include the write password 0A5h in the upper byte. Any write to any FCTLx register with any value other than 0A5h in the upper byte is a security key violation, sets the KEYV flag and triggers a PUC system reset. Any read of any FCTLx registers reads 096h in the upper byte.

Any write to FCTL1 during an erase or byte or word write operation is an access violation and sets ACCVIFG. Writing to FCTL1 is allowed in block write mode when WAIT = 1, but writing to FCTL1 in block write mode when WAIT = 0 is an access violation and sets ACCVIFG.

Any write to FCTL2 when the BUSY = 1 is an access violation.

Any FCTLx register may be read when BUSY = 1. A read does not cause an access violation.

7.3.8 Flash Memory Controller Interrupts

The flash controller has two interrupt sources, KEYV, and ACCVIFG. ACCVIFG is set when an access violation occurs. When the ACCVIE bit is re-enabled after a flash write or erase, a set ACCVIFG flag generates an interrupt request. ACCVIFG sources the NMI interrupt vector, so it is not necessary for GIE to be set for ACCVIFG to request an interrupt. ACCVIFG may also be checked by software to determine if an access violation occurred. ACCVIFG must be reset by software.

The key violation flag KEYV is set when any of the flash control registers are written with an incorrect password. When this occurs, a PUC is generated immediately resetting the device.

7.3.9 Programming Flash Memory Devices

There are three options for programming an MSP430 flash device. All options support in-system programming:

- Program via JTAG
- Program via the bootstrap loader
- Program via a custom solution

7.3.9.1 Programming Flash Memory via JTAG

MSP430 devices can be programmed via the JTAG port. The JTAG interface requires four signals (five signals on 20- and 28-pin devices), ground and, optionally, V_{CC} and $\overline{RST}$/NMI.

The JTAG port is protected with a fuse. Blowing the fuse completely disables the JTAG port and is not reversible. Further access to the device via JTAG is not possible. For details, see the *MSP430 Programming Via the JTAG Interface User's Guide* (SLAU320).

7.3.9.2 Programming Flash Memory via the Bootstrap Loader (BSL)

Most MSP430 flash devices contain a bootstrap loader. See the device-specific data sheet for implementation details. The BSL enables users to read or program the flash memory or RAM using a UART serial interface. Access to the MSP430 flash memory via the BSL is protected by a 256-bit user-defined password. For more details see the *MSP430 Programming Via the Bootstrap Loader User's Guide* (SLAU319).

7.3.9.3 Programming Flash Memory via a Custom Solution

The ability of the MSP430 CPU to write to its own flash memory allows for in-system and external custom programming solutions as shown in Figure 7-12. The user can choose to provide data to the MSP430 through any means available (UART, SPI, etc.). User-developed software can receive the data and program the flash memory. Since this type of solution is developed by the user, it can be completely customized to fit the application needs for programming, erasing, or updating the flash memory.

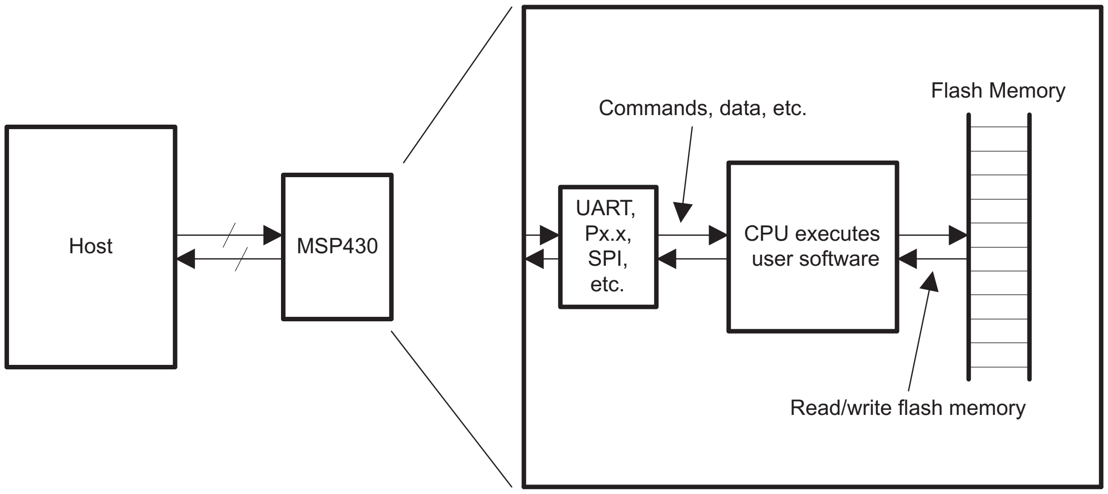


Figure 7-12. User-Developed Programming Solution

7.4 Flash Memory Registers

The flash memory registers are listed in Table 7-4.

Table 7-4. Flash Memory Registers

Register	Short Form	Register Type	Address	Initial State
Flash memory control register 1	FCTL1	Read/write	0x0128	0x9600 with PUC
Flash memory control register 2	FCTL2	Read/write	0x012A	0x9642 with PUC
Flash memory control register 3	FCTL3	Read/write	0x012C	0x9658 with PUC(1)
Flash memory control register 4(2)	FCTL4	Read/write	0x01BE	0x0000 with PUC
Interrupt Enable 1	IE1	Read/write	0x0000	Reset with PUC
Interrupt Flag 1	IFG1	Read/write	0x0002	

(1) KEYV is reset with POR.
(2) Not present in all devices. See device-specific data sheet.

7.4.1 FCTL1, Flash Memory Control Register

15	14	13	12	11	10	9	8
FRKEY, Read as 096h **FWKEY**, Must be written as 0A5h							

7	6	5	4	3	2	1	0
BLKWRT	**WRT**	**Reserved**	**EEIEX**(1)	**EEI**(1)	**MERAS**	**ERASE**	**Reserved**
rw-0	rw-0	r0	rw-0	rw-0	rw-0	rw-0	r0

FRKEY FWKEY Bits 15-8 FCTLx password. Always reads as 096h. Must be written as 0A5h. Writing any other value generates a PUC.

BLKWRT Bit 7 Block write mode. WRT must also be set for block write mode. BLKWRT is automatically reset when EMEX is set.

0 Block-write mode is off

1 Block-write mode is on

WRT Bit 6 Write. This bit is used to select any write mode. WRT is automatically reset when EMEX is set.

0 Write mode is off

1 Write mode is on

Reserved Bit 5 Reserved. Always read as 0.

EEIEX Bit 4 Enable Emergency Interrupt Exit. Setting this bit enables an interrupt to cause an emergency exit from a flash operation when GIE = 1. EEIEX is automatically reset when EMEX is set.

0 Exit interrupt disabled.

1 Exit on interrupt enabled.

EEI Bits 3 Enable Erase Interrupts. Setting this bit allows a segment erase to be interrupted by an interrupt request. After the interrupt is serviced the erase cycle is resumed.

0 Interrupts during segment erase disabled.

1 Interrupts during segment erase enabled.

MERAS Bit 2
ERASE Bit 1 Mass erase and erase. These bits are used together to select the erase mode. MERAS and ERASE are automatically reset when EMEX is set.

MERAS	ERASE	Erase Cycle
0	0	No erase
0	1	Erase individual segment only
1	0	Erase all main memory segments
1	1	LOCKA = 0: Erase main and information flash memory. LOCKA = 1: Erase only main flash memory.

Reserved Bit 0 Reserved. Always read as 0.

(1) Not present on MSP430x20xx and MSP430G2xx devices.

7.4.2 FCTL2, Flash Memory Control Register

15	14	13	12	11	10	9	8
FWKEYx, Read as 096h Must be written as 0A5h							

7	6	5	4	3	2	1	0
FSSELx		**FNx**					
rw-0	rw-1	rw-0	rw-0	rw-0	rw-0	rw-1	rw-0

FWKEYx Bits 15-8 FCTLx password. Always reads as 096h. Must be written as 0A5h. Writing any other value generates a PUC.

FSSELx Bits 7-6 Flash controller clock source select

00 ACLK

01 MCLK

10 SMCLK

11 SMCLK

FNx Bits 5-0 Flash controller clock divider. These six bits select the divider for the flash controller clock. The divisor value is FNx + 1. For example, when FNx = 00h, the divisor is 1. When FNx = 03Fh, the divisor is 64.

7.4.3 FCTL3, Flash Memory Control Register

15	14	13	12	11	10	9	8
FWKEYx, Read as 096h Must be written as 0A5h							

7	6	5	4	3	2	1	0
FAIL	**LOCKA**	**EMEX**	**LOCK**	**WAIT**	**ACCVIFG**	**KEYV**	**BUSY**
r(w)-0	r(w)-1	rw-0	rw-1	r-1	rw-0	rw-(0)	r(w)-0

FWKEYx	Bits 15-8	FCTLx password. Always reads as 096h. Must be written as 0A5h. Writing any other value generates a PUC.
FAIL	Bit 7	Operation failure. This bit is set if the fFTG clock source fails, or a flash operation is aborted from an interrupt when EEIEX = 1. FAIL must be reset with software.
		0 No failure
		1 Failure
LOCKA	Bit 6	SegmentA and Info lock. Write a 1 to this bit to change its state. Writing 0 has no effect.
		0 Segment A unlocked and all information memory is erased during a mass erase.
		1 Segment A locked and all information memory is protected from erasure during a mass erase.
EMEX	Bit 5	Emergency exit
		0 No emergency exit
		1 Emergency exit
LOCK	Bit 4	Lock. This bit unlocks the flash memory for writing or erasing. The LOCK bit can be set any time during a byte or word write or erase operation, and the operation completes normally. In the block write mode if the LOCK bit is set while BLKWRT = WAIT = 1, then BLKWRT and WAIT are reset and the mode ends normally.
		0 Unlocked
		1 Locked
WAIT	Bit 3	Wait. Indicates the flash memory is being written to.
		0 The flash memory is not ready for the next byte/word write
		1 The flash memory is ready for the next byte/word write
ACCVIFG	Bit 2	Access violation interrupt flag
		0 No interrupt pending
		1 Interrupt pending
KEYV	Bit 1	Flash security key violation. This bit indicates an incorrect FCTLx password was written to any flash control register and generates a PUC when set. KEYV must be reset with software.
		0 FCTLx password was written correctly
		1 FCTLx password was written incorrectly
BUSY	Bit 0	Busy. This bit indicates the status of the flash timing generator.
		0 Not Busy
		1 Busy

7.4.4 FCTL4, Flash Memory Control Register

This register is not available in all devices. See the device-specific data sheet for details.

15	14	13	12	11	10	9	8
FWKEYx, Read as 096h Must be written as 0A5h							

7	6	5	4	3	2	1	0
		MRG1	**MRG0**				
r-0	r-0	rw-0	rw-0	r-0	r-0	r-0	r-0

FWKEYx	Bits 15-8	FCTLx password. Always reads as 096h. Must be written as 0A5h. Writing any other value generates a PUC.
Reserved	Bits 7-6	Reserved. Always read as 0.
MRG1	Bit 5	Marginal read 1 mode. This bit enables the marginal 1 read mode. The marginal read 1 bit is cleared if the CPU starts execution from the flash memory. If both MRG1 and MRG0 are set MRG1 is active and MRG0 is ignored. 0 Marginal 1 read mode is disabled. 1 Marginal 1 read mode is enabled.
MRG0	Bit 4	Marginal read 0 mode. This bit enables the marginal 0 read mode. The marginal mode 0 is cleared if the CPU starts execution from the flash memory. If both MRG1 and MRG0 are set MRG1 is active and MRG0 is ignored. 0 Marginal 0 read mode is disabled. 1 Marginal 0 read mode is enabled.
Reserved	Bits 3-0	Reserved. Always read as 0.

7.4.5 IE1, Interrupt Enable Register 1

7	6	5	4	3	2	1	0
		ACCVIE					
		rw-0					

	Bits 7-6	These bits may be used by other modules. See the device-specific data sheet.
ACCVIE	Bit 5	Flash memory access violation interrupt enable. This bit enables the ACCVIFG interrupt. Because other bits in IE1 may be used for other modules, it is recommended to set or clear this bit using BIS.B or BIC.B instructions, rather than MOV.B or CLR.B instructions. 0 Interrupt not enabled 1 Interrupt enabled
	Bits 4-0	These bits may be used by other modules. See the device-specific data sheet.

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.

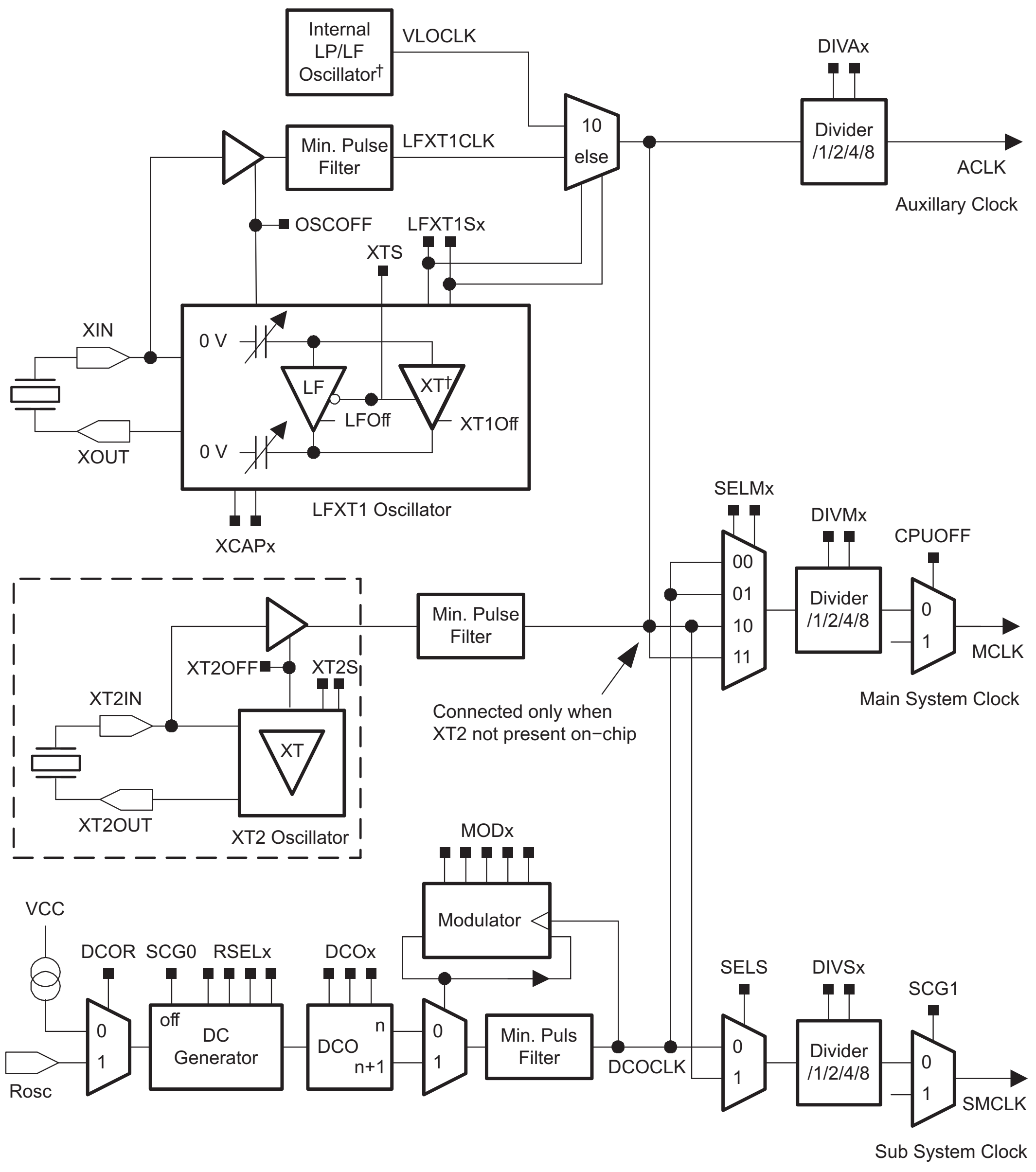


Figure 5-1. Basic Clock Module+ Block Diagram − MSP430F2xx

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.

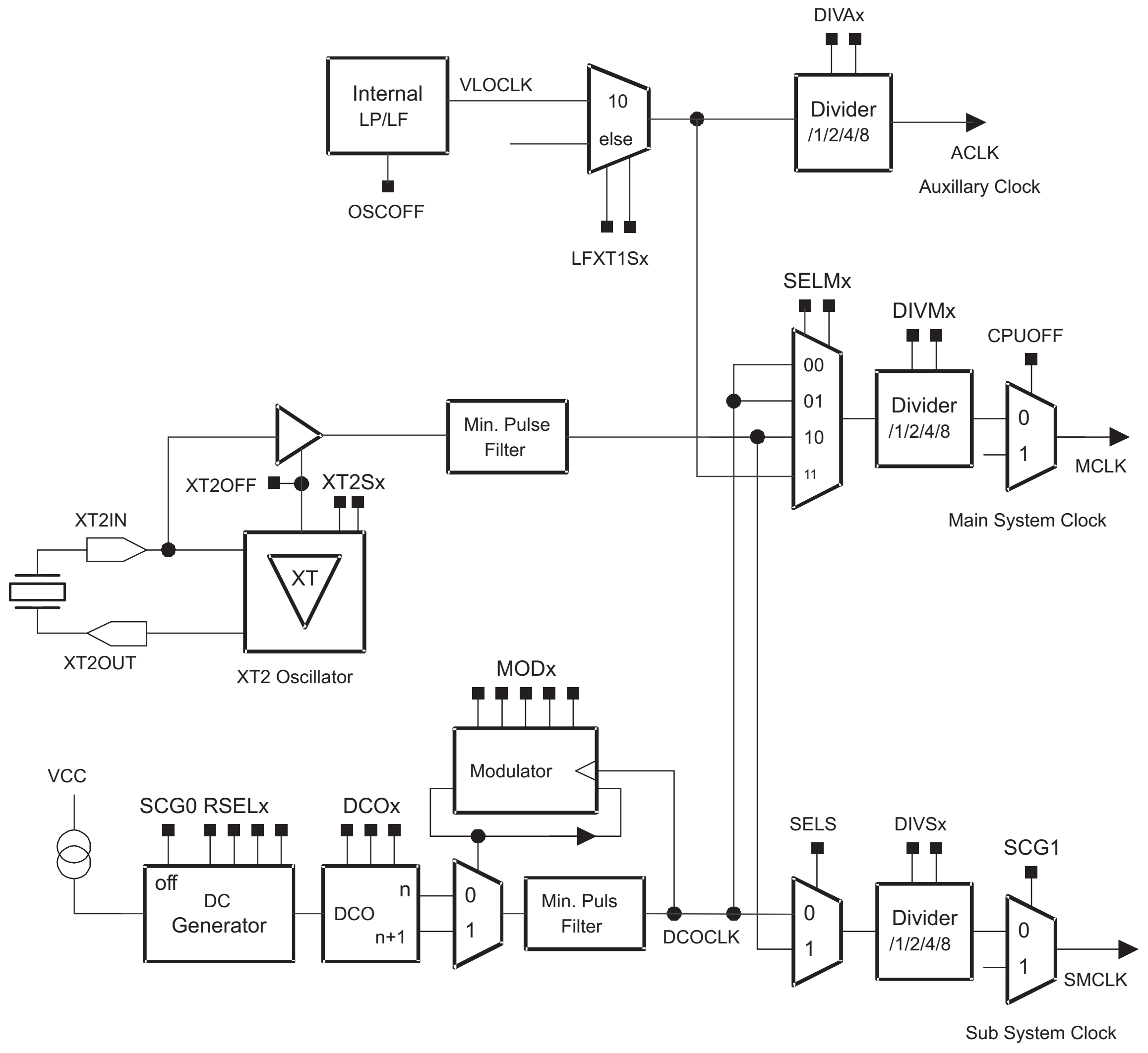


Figure 5-2. Basic Clock Module+ Block Diagram − MSP430AFE2xx

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
```

5.2.1 Basic Clock Module+ Features for Low-Power Applications

Conflicting requirements typically exist in battery-powered applications:

- Low clock frequency for energy conservation and time keeping
- High clock frequency for fast reaction to events and fast burst processing capability
- Clock stability over operating temperature and supply voltage

The basic clock module+ addresses the above conflicting requirements by allowing the user to select from the three available clock signals: ACLK, MCLK, and SMCLK. For optimal low-power performance, ACLK can be sourced from a low-power 32768-Hz watch crystal (if available), providing a stable time base for the system and low-power standby operation, or from the internal low-frequency oscillator when crystal-accurate time keeping is not required. The MCLK can be configured to operate from the on-chip DCO that can be activated when requested by interrupt-driven events. The SMCLK can be configured to operate from a crystal or the DCO, depending on peripheral requirements. A flexible clock distribution and divider system is provided to fine tune the individual clock requirements.

5.2.2 Internal Very-Low-Power Low-Frequency Oscillator (VLO)

The internal very-low-power low-frequency oscillator (VLO) provides a typical frequency of 12 kHz (see device-specific data sheet for parameters) without requiring a crystal. VLOCLK source is selected by setting LFXT1Sx = 10 when XTS = 0. The OSCOFF bit disables the VLO for LPM4. The LFXT1 crystal oscillators are disabled when the VLO is selected reducing current consumption. The VLO consumes no power when not being used.

Devices without LFXT1 (for example, the MSP430G22x0) should be configured to use the VLO as ACLK.

5.2.3 LFXT1 Oscillator

The LFXT1 oscillator is not implemented in the MSP430G22x0 device family.

The LFXT1 oscillator supports ultra-low current consumption using a 32768-Hz watch crystal in LF mode (XTS = 0). A watch crystal connects to XIN and XOUT without any other external components. The software-selectable XCAPx bits configure the internally provided load capacitance for the LFXT1 crystal in LF mode. This capacitance can be selected as 1 pF, 6 pF, 10 pF, or 12.5 pF typical. Additional external capacitors can be added if necessary.

The LFXT1 oscillator also supports high-speed crystals or resonators when in HF mode (XTS = 1, XCAPx = 00). The high-speed crystal or resonator connects to XIN and XOUT and requires external capacitors on both terminals. These capacitors should be sized according to the crystal or resonator specifications. When LFXT1 is in HF mode, the LFXT1Sx bits select the range of operation.

LFXT1 may be used with an external clock signal on the XIN pin in either LF or HF mode when LFXT1Sx = 11, OSCOFF = 0, and XCAPx = 00. When used with an external signal, the external frequency must meet the data sheet parameters for the chosen mode. When the input frequency is below the specified lower limit, the LFXT1OF bit may be set preventing the CPU from being clocked with LFXT1CLK.

Software can disable LFXT1 by setting OSCOFF, if LFXT1CLK does not source SMCLK or MCLK, as shown in Figure 5-3.

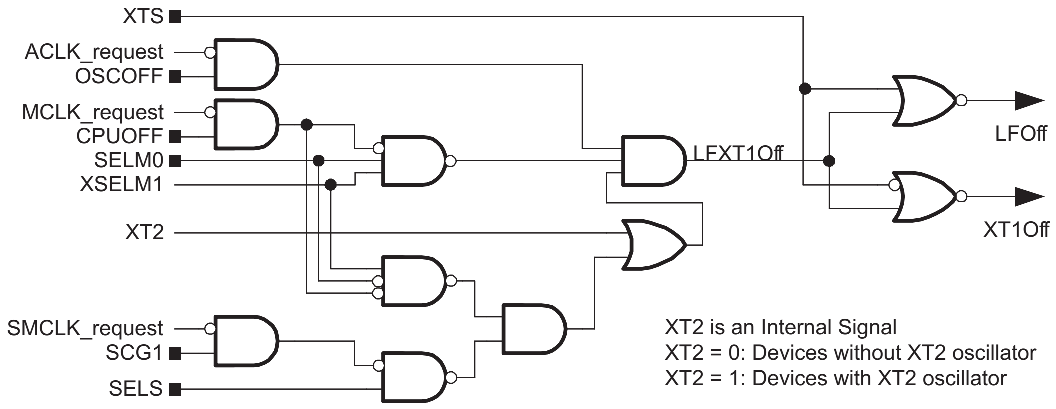

XTS
ACLK_request
OSCOFF
MCLK_request
CPUOFF
SELM0
XSELM1
XT2
SMCLK_request
SCG1
SELS
LFXT1Off
LFOff
XT1Off
XT2 is an Internal Signal
XT2 = 0: Devices without XT2 oscillator
XT2 = 1: Devices with XT2 oscillator

Figure 5-3. Off Signals for the LFXT1 Oscillator

NOTE: LFXT1 Oscillator Characteristics

Low-frequency crystals often require hundreds of milliseconds to start up, depending on the crystal.

Ultralow-power oscillators such as the LFXT1 in LF mode should be guarded from noise coupling from other sources. The crystal should be placed as close as possible to the MSP430 with the crystal housing grounded and the crystal traces guarded with ground traces.

5.2.4 XT2 Oscillator

Some devices have a second crystal oscillator, XT2. XT2 sources XT2CLK and its characteristics are identical to LFXT1 in HF mode. The XT2Sx bits select the range of operation of XT2. The XT2OFF bit disables the XT2 oscillator if XT2CLK is not used for MCLK or SMCLK as shown in Figure 5-4.

XT2 may be used with external clock signals on the XT2IN pin when XT2Sx = 11 and XT2OFF = 0. When used with an external signal, the external frequency must meet the data sheet parameters for XT2. When the input frequency is below the specified lower limit, the XT2OF bit may be set to prevent the CPU from being clocked with XT2CLK.

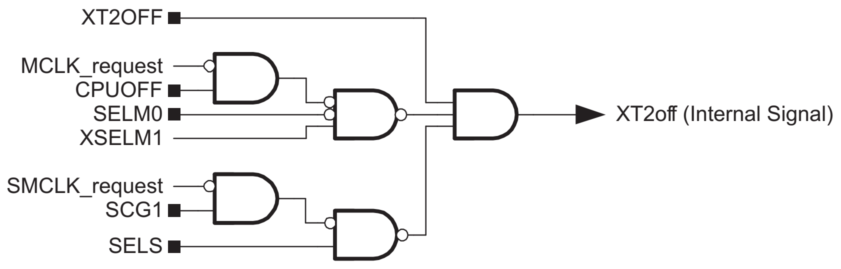


Figure 5-4. Off Signals for Oscillator XT2

5.2.5 Digitally-Controlled Oscillator (DCO)

The DCO is an integrated digitally controlled oscillator. The DCO frequency can be adjusted by software using the DCOx, MODx, and RSELx bits.

5.2.5.1 Disabling the DCO

Software can disable DCOCLK by setting SCG0 when it is not used to source SMCLK or MCLK in active mode, as shown in Figure 5-5.

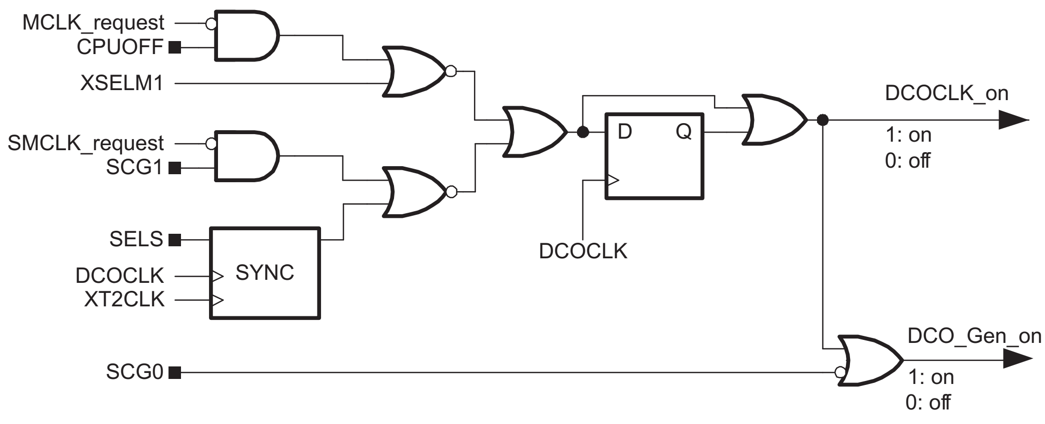


Figure 5-5. On/Off Control of DCO

5.2.5.2 Adjusting the DCO Frequency

After a PUC, RSELx = 7 and DCOx = 3, allowing the DCO to start at a mid-range frequency. MCLK and SMCLK are sourced from DCOCLK. Because the CPU executes code from MCLK, which is sourced from the fast-starting DCO, code execution typically begins from PUC in less than 2 μs. The typical DCOx and RSELx ranges and steps are shown in Figure 5-6.

The frequency of DCOCLK is set by the following functions:

- The four RSELx bits select one of sixteen nominal frequency ranges for the DCO. These ranges are defined for an individual device in the device-specific data sheet.
- The three DCOx bits divide the DCO range selected by the RSELx bits into 8 frequency steps, separated by approximately 10%.
- The five MODx bits, switch between the frequency selected by the DCOx bits and the next higher frequency set by DCOx+1. When DCOx = 07h, the MODx bits have no effect because the DCO is already at the highest setting for the selected RSELx range.

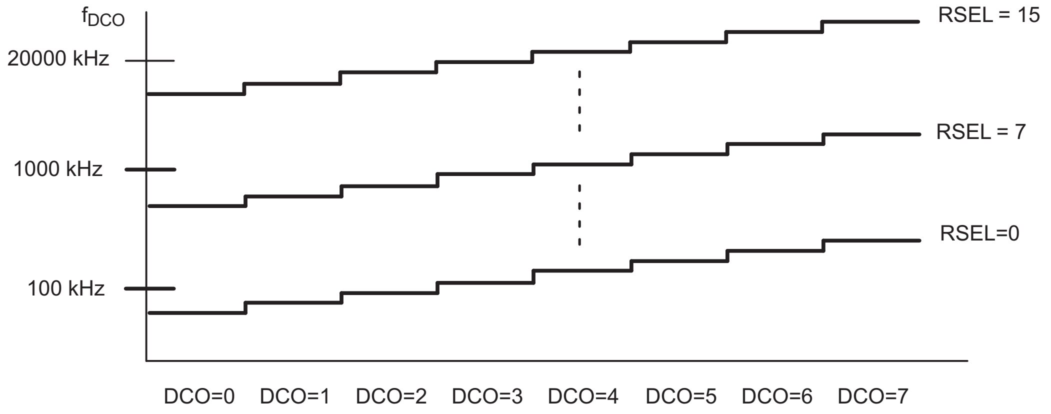


Figure 5-6. Typical DCOx Range and RSELx Steps

Each MSP430F2xx device (and most MSP430G2xx devices; see device-specific data sheets) has calibrated DCOCTL and BCSCTL1 register settings for specific frequencies stored in information memory segment A. To use the calibrated settings, the information is copied into the DCOCTL and BCSCTL1 registers. The calibrated settings affect the DCOx, MODx, and RSELx bits, and clear all other bits, except XT2OFF which remains set. The remaining bits of BCSCTL1 can be set or cleared as needed with BIS.B or BIC.B instructions.

```
; Set DCO to 1 MHz:
CLR.B   &DCOCTL                  ; Select lowest DCOx
                                 ; and MODx settings
```

```
MOV.B    &CALBC1_1MHZ,&BCSCTL1    ; Set range
MOV.B    &CALDCO_1MHZ,&DCOCTL     ; Set DCO step + modulation
```

5.2.5.3 Using an External Resistor (R_{OSC}) for the DCO

Some MSP430F2xx devices provide the option to source the DCO current through an external resistor, R_{OSC}, tied to $DV_{CC,}$ when DCOR = 1. In this case, the DCO has the same characteristics as MSP430x1xx devices, and the RSELx setting is limited to 0 to 7 with the RSEL3 ignored. This option provides an additional method to tune the DCO frequency by varying the resistor value. See the device-specific data sheet for parameters.

5.2.6 DCO Modulator

The modulator mixes two DCO frequencies, f_{DCO} and f_{DCO+1} to produce an intermediate effective frequency between f_{DCO} and f_{DCO+1} and spread the clock energy, reducing electromagnetic interference (EMI). The modulator mixes f_{DCO} and f_{DCO+1} for 32 DCOCLK clock cycles and is configured with the MODx bits. When MODx = 0 the modulator is off.

The modulator mixing formula is:

$$t = (32 - MODx) \times t_{DCO} + MODx \times t_{DCO+1}$$

Because f_{DCO} is lower than the effective frequency and f_{DCO+1} is higher than the effective frequency, the error of the effective frequency integrates to zero. It does not accumulate. The error of the effective frequency is zero every 32 DCOCLK cycles. Figure 5-7 shows the modulator operation.

The modulator settings and DCO control are configured with software. The DCOCLK can be compared to a stable frequency of known value and adjusted with the DCOx, RSELx, and MODx bits. See http://www.msp430.com for application notes and example code on configuring the DCO.

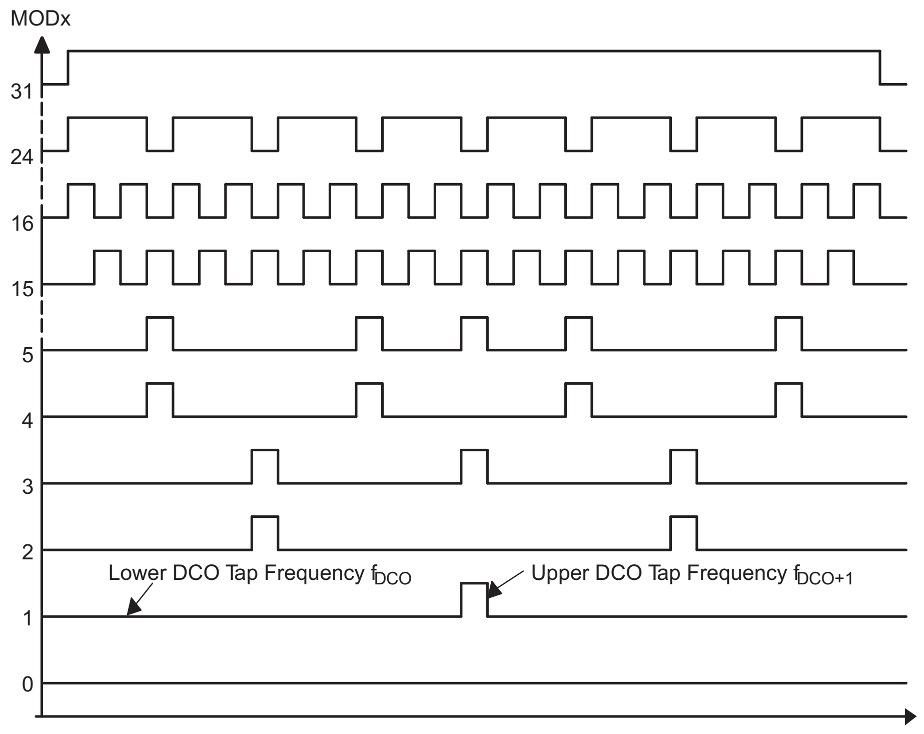

MODx
31
24
16
15
5
4
3
2
1
0
Lower DCO Tap Frequency f_{DCO}
Upper DCO Tap Frequency f_{DCO+1}

Figure 5-7. Modulator Patterns

5.2.7 Basic Clock Module+ Fail-Safe Operation

The basic clock module+ incorporates an oscillator-fault fail-safe feature. This feature detects an oscillator fault for LFXT1 and XT2 as shown in Figure 5-8. The available fault conditions are:

- Low-frequency oscillator fault (LFXT1OF) for LFXT1 in LF mode

- High-frequency oscillator fault (LFXT1OF) for LFXT1 in HF mode
- High-frequency oscillator fault (XT2OF) for XT2

The crystal oscillator fault bits LFXT1OF, and XT2OF are set if the corresponding crystal oscillator is turned on and not operating properly. The fault bits remain set as long as the fault condition exists and are automatically cleared if the enabled oscillators function normally.

The OFIFG oscillator-fault flag is set and latched at POR or when an oscillator fault (LFXT1OF, or XT2OF) is detected. When OFIFG is set, MCLK is sourced from the DCO, and if OFIE is set, the OFIFG requests an NMI interrupt. When the interrupt is granted, the OFIE is reset automatically. The OFIFG flag must be cleared by software. The source of the fault can be identified by checking the individual fault bits.

If a fault is detected for the crystal oscillator sourcing the MCLK, the MCLK is automatically switched to the DCO for its clock source. This does not change the SELMx bit settings. This condition must be handled by user software.

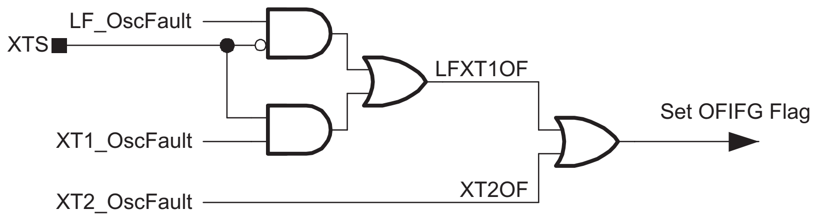


Figure 5-8. Oscillator-Fault Logic

5.2.7.1 Sourcing MCLK from a Crystal

After a PUC, the basic clock module+ uses DCOCLK for MCLK. If required, MCLK may be sourced from LFXT1 or XT2 - if available.

The sequence to switch the MCLK source from the DCO clock to the crystal clock (LFXT1CLK or XT2CLK) is:

1. Turn on the crystal oscillator and select the appropriate mode
2. Clear the OFIFG flag
3. Wait at least 50 μs
4. Test OFIFG, and repeat steps 2 through 4 until OFIFG remains cleared.

```
; Select LFXT1 (HF mode) for MCLK
     BIC.W    #OSCOFF,SR                  ; Turn on osc.
     BIS.B    #XTS,&BCSCTL1               ; HF mode
     MOV.B    #LFXT1S0,&BCSCTL3           ; 1-3MHz Crystal
L1   BIC.B    #OFIFG,&IFG1                ; Clear OFIFG
     MOV.W    #0FFh,R15                   ; Delay
L2   DEC.W    R15                         ;
     JNZ      L2                          ;
     BIT.B    #OFIFG,&IFG1                ; Re-test OFIFG
     JNZ      L1                          ; Repeat test if needed
     BIS.B    #SELM1+SELM0,&BCSCTL2       ; Select LFXT1CLK
```

5.2.8 *Synchronization of Clock Signals*

When switching MCLK or SMCLK from one clock source to another, the switch is synchronized to avoid critical race conditions as shown in Figure 5-9:

- The current clock cycle continues until the next rising edge.
- The clock remains high until the next rising edge of the new clock.
- The new clock source is selected and continues with a full high period.

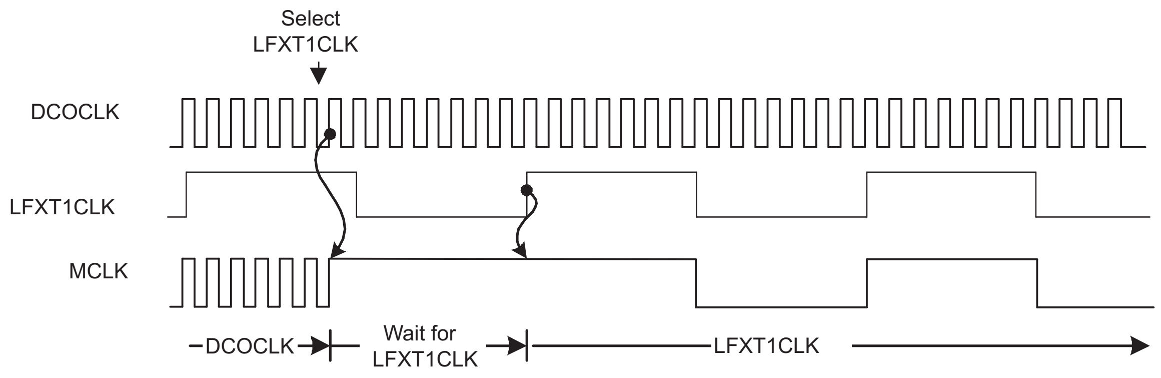


Figure 5-9. Switch MCLK from DCOCLK to LFXT1CLK

5.3 Basic Clock Module+ Registers

The basic clock module+ registers are listed in Table 5-1.

Table 5-1. Basic Clock Module+ Registers

Register	Short Form	Register Type	Address	Initial State
DCO control register	DCOCTL	Read/write	056h	060h with PUC
Basic clock system control 1	BCSCTL1	Read/write	057h	087h with POR[1]
Basic clock system control 2	BCSCTL2	Read/write	058h	Reset with PUC
Basic clock system control 3	BCSCTL3	Read/write	053h	005h with PUC[2]
SFR interrupt enable register 1	IE1	Read/write	000h	Reset with PUC
SFR interrupt flag register 1	IFG1	Read/write	002h	Reset with PUC

(1) Some of the register bits are also PUC initialized (see Section 5.3.2).
(2) The initial state of BCSCTL3 is 000h in the MSP430AFE2xx devices.

5.3.1 DCOCTL, DCO Control Register

7	6	5	4	3	2	1	0
DCOx			MODx				
rw-0	rw-1	rw-1	rw-0	rw-0	rw-0	rw-0	rw-0

DCOx	Bits 7-5	DCO frequency select. These bits select which of the eight discrete DCO frequencies within the range defined by the RSELx setting is selected.
MODx	Bits 4-0	Modulator selection. These bits define how often the f_{DCO+1} frequency is used within a period of 32 DCOCLK cycles. During the remaining clock cycles (32-MOD) the f_{DCO} frequency is used. Not useable when DCOx = 7.

5.3.2 BCSCTL1, Basic Clock System Control Register 1

7	6	5	4	3	2	1	0
XT2OFF	XTS[1][2]	DIVAx		RSELx			
rw-(1)	rw-(0)	rw-(0)	rw-(0)	rw-0	rw-1	rw-1	rw-1

XT2OFF	Bit 7	XT2 off. This bit turns off the XT2 oscillator	
		0	XT2 is on
		1	XT2 is off if it is not used for MCLK or SMCLK.
XTS	Bit 6	LFXT1 mode select.	
		0	Low-frequency mode
		1	High-frequency mode
DIVAx	Bits 5-4	Divider for ACLK	
		00	/1
		01	/2
		10	/4
		11	/8
RSELx	Bits 3-0	Range select. Sixteen different frequency ranges are available. The lowest frequency range is selected by setting RSELx = 0. RSEL3 is ignored when DCOR = 1.	

(1) XTS = 1 is not supported in MSP430x20xx and MSP430G2xx devices (see Figure 5-1 and Figure 5-2 for details on supported settings for all devices).
(2) This bit is reserved in the MSP430AFE2xx devices.

5.3.3 BCSCTL2, Basic Clock System Control Register 2

7	6	5	4	3	2	1	0
SELMx		**DIVMx**		**SELS**	**DIVSx**		**DCOR**[1][2]
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

SELMx	Bits 7-6	Select MCLK. These bits select the MCLK source.	
		00	DCOCLK
		01	DCOCLK
		10	XT2CLK when XT2 oscillator present on-chip. LFXT1CLK or VLOCLK when XT2 oscillator not present on-chip.
		11	LFXT1CLK or VLOCLK
DIVMx	Bits 5-4	Divider for MCLK	
		00	/1
		01	/2
		10	/4
		11	/8
SELS	Bit 3	Select SMCLK. This bit selects the SMCLK source.	
		0	DCOCLK
		1	XT2CLK when XT2 oscillator present. LFXT1CLK or VLOCLK when XT2 oscillator not present
DIVSx	Bits 2-1	Divider for SMCLK	
		00	/1
		01	/2
		10	/4
		11	/8
DCOR	Bit 0	DCO resistor select. Not available in all devices. See the device-specific data sheet.	
		0	Internal resistor
		1	External resistor

[1] Does not apply to MSP430x20xx or MSP430x21xx devices.
[2] This bit is reserved in the MSP430AFE2xx devices.

5.3.4 BCSCTL3, Basic Clock System Control Register 3

7	6	5	4	3	2	1	0
XT2Sx		**LFXT1Sx**[1]		**XCAPx**[2]		**XT2OF**[3]	**LFXT1OF**[2]
rw-0	rw-0	rw-0	rw-0	rw-0	rw-1	r0	r-(1)

XT2Sx	Bits 7-6	XT2 range select. These bits select the frequency range for XT2.
		00 0.4- to 1-MHz crystal or resonator
		01 1- to 3-MHz crystal or resonator
		10 3- to 16-MHz crystal or resonator
		11 Digital external 0.4- to 16-MHz clock source
LFXT1Sx	Bits 5-4	Low-frequency clock select and LFXT1 range select. These bits select between LFXT1 and VLO when XTS = 0, and select the frequency range for LFXT1 when XTS = 1.
		When XTS = 0:
		00 32768-Hz crystal on LFXT1
		01 Reserved
		10 VLOCLK (Reserved in MSP430F21x1 devices)
		11 Digital external clock source
		When XTS = 1 (Not applicable for MSP430x20xx devices, MSP430G2xx1/2/3)
		00 0.4- to 1-MHz crystal or resonator
		01 1- to 3-MHz crystal or resonator
		10 3- to 16-MHz crystal or resonator
		11 Digital external 0.4- to 16-MHz clock source
		LFXT1Sx definition for MSP430AFE2xx devices:
		00 Reserved
		01 Reserved
		10 VLOCLK
		11 Reserved
XCAPx	Bits 3-2	Oscillator capacitor selection. These bits select the effective capacitance seen by the LFXT1 crystal when XTS = 0. If XTS = 1 or if LFXT1Sx = 11 XCAPx should be 00.
		00 ~1 pF
		01 ~6 pF
		10 ~10 pF
		11 ~12.5 pF
XT2OF	Bit 1	XT2 oscillator fault
		0 No fault condition present
		1 Fault condition present
LFXT1OF	Bit 0	LFXT1 oscillator fault
		0 No fault condition present
		1 Fault condition present

(1) MSP430G22x0: The LFXT1Sx bits should be programmed to 10b during the initialization and start-up code to select VLOCLK (for more details refer to Digital I/O chapter). The other bits are reserved and should not be altered.
(2) This bit is reserved in the MSP430AFE2xx devices.
(3) Does not apply to MSP430x2xx, MSP430x21xx, or MSP430x22xx devices.

5.3.5 IE1, Interrupt Enable Register 1

7	6	5	4	3	2	1	0
						OFIE[(1)]	
						rw-0	

	Bits 7-2	These bits may be used by other modules. See device-specific data sheet.
OFIE	Bit 1	Oscillator fault interrupt enable. This bit enables the OFIFG interrupt. Because other bits in IE1 may be used for other modules, it is recommended to set or clear this bit using BIS.B or BIC.B instructions, rather than MOV.B or CLR.B instructions. 0 Interrupt not enabled 1 Interrupt enabled
	Bits 0	This bit may be used by other modules. See device-specific data sheet.

(1) MSP430G22x0: This bit should not be set.

5.3.6 IFG1, Interrupt Flag Register 1

7	6	5	4	3	2	1	0
						OFIFG[(1)]	
						rw-1	

	Bits 7-2	These bits may be used by other modules. See device-specific data sheet.
OFIFG	Bit 1	Oscillator fault interrupt flag. Because other bits in IFG1 may be used for other modules, it is recommended to set or clear this bit using BIS.B or BIC.B instructions, rather than MOV.B or CLR.B instructions. 0 No interrupt pending 1 Interrupt pending
	Bits 0	This bit may be used by other modules. See device-specific data sheet.

(1) MSP430G22x0: The LFXT1 oscillator pins are not available in this device. The oscillator fault flag will always be set by hardware. The interrupt enable bit should not be set.

Digital I/O

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

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.

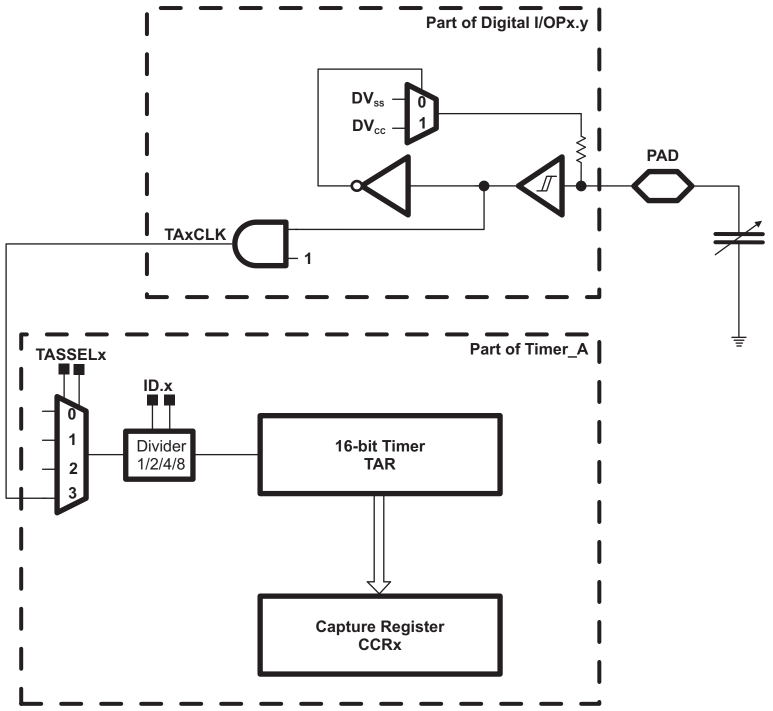


Figure 8-1. Example Circuitry and Configuration using the Pin Oscillator

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.

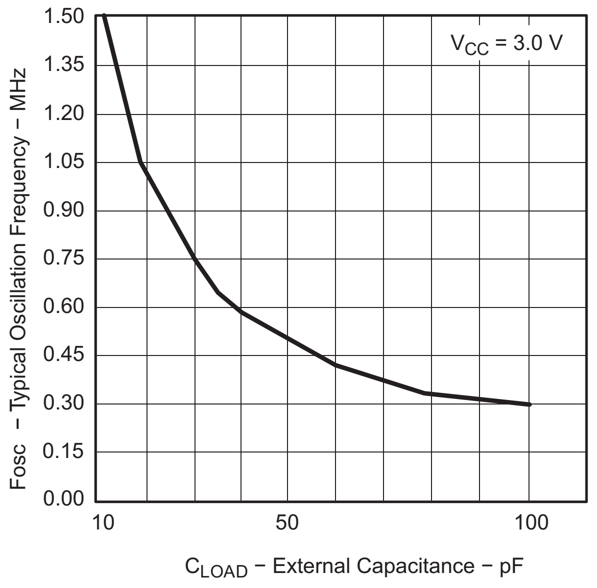


Figure 8-2. Typical Pin-Oscillation Frequency

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

MyRTC.cpp

```
/*
MyRTC.cpp
Librería para almacenar la fecha y hora del comienzo de las mediciones
en memoria Flash
*/

#include <Energia.h>
#include <MspFlash.h>
#include <MyRTC.h>


//Dirección de memoria Flash para guardad la fecha
#define flash SEGMENT_D
unsigned char* timeRTC;

/*
 * setTime(String)
 * Guarda fecha y hora en memoria
 */

void MyRTCClass::setTime(String timeData)
{
  //Once received the string w
  timeRTC= (unsigned char*)timeData.c_str();
  Flash.erase(SEGMENT_D);
  Flash.write(flash,timeRTC, timeData.length());
}

/*
 * getTime()
 * Lee fecha y hora almacenada memoria
 */

void MyRTCClass::getTime()
{
  unsigned char p = 0;
  int i=0;
  do
{
  Flash.read(flash+i, &p, 1);
  Serial.write(p);
  i++;
} while (p && (i<9) );
}



MyRTCClass MyRTC;
```

MyRTC.h

```
/*
  MyRTC.h
*/

#ifndef MyRTC_h
#define MyRTC_h

#include <Energia.h>


class MyRTCClass
{
  public:
    void setTime(String timeData);
    void getTime();

};

extern MyRTCClass MyRTC;


#endif
```

MyFlash.cpp

```
/*
  MyFlash.cpp
  Esta librería se encarga del manejo de la memoria Flash a partir de la
  Librería MspFlash integrada en Energia.
*/

#include <Energia.h>
#include <MspFlash.h>
#include "MyFlash.h"

unsigned char* addr = (unsigned char*) 0xF400;
unsigned char* initialAddr = (unsigned char*) 0xF400;
unsigned char* finalAddr = (unsigned char*) 0x0FDFF;
unsigned char* actualPointer;
int totalWritten;

//Variable j stores how many times we write data in flash memory
int j;
int counter;
#define flash ((unsigned char*) 0xF600)


/*
 * doRead()
 *
 * Lee todos los datos almacenados en la memoria Flash
 */

void MyFlashClass::doRead()
{
unsigned char p = 0;
int i=0;
do
{
  Flash.read(initialAddr+i, &p, 1);
  Serial.write(p);
  i++;
} while (p && (i<(j*14)) );
}

/*
 * doEraseSegment(unsigned char*)
 *
 * Borra un segmente de la memoria Flash
 * Parámetro: dirección inicial del segmento a borrar
 */

void MyFlashClass::doEraseSegment(unsigned char* segmentAddress)
{
    Flash.erase(segmentAddress);
}

/*
 * doEraseSegment(unsigned char*)
 *
 * Borra  la memoria Flash disponible para almacenamiento de datos
 */

void MyFlashClass::doEraseAllFlash(){
    Flash.erase(SEGMENT_11);
    Flash.erase(SEGMENT_10);
    Flash.erase(SEGMENT_9);
    Flash.erase(SEGMENT_8);
    Flash.erase(SEGMENT_7);
    Flash.erase(SEGMENT_6);
    Flash.erase(SEGMENT_5);
    Flash.erase(SEGMENT_4);
    Flash.erase(SEGMENT_3);
    Flash.erase(SEGMENT_2);
    Flash.erase(SEGMENT_1);
    j=0;
}

/*
 * doWrite(unsigned char*, int)
 *
 * Parámetro: valor del dato a almacenar
 * Parámetro: nº de bytes del parámetro a almacenar
 */
void MyFlashClass::doWrite(unsigned char* valueToWrite, int len)
{
    if(addr+((j+1)*len) < finalAddr){
     Serial.println("dentro del if dowrite");
     Flash.write(addr + (j*len)  ,valueToWrite, len);
     j++;

    }else{
     //DONOTHING NOT SAVING DATA
    }
}

MyFlashClass MyFlash;
```

MyFlash.h

```
/*
  MyFlash.h
*/
#ifndef MyFlash_h
#define MyFlash_h

#include <Energia.h>


//variables flash public methods
#define SEG_SIZE 512

#define SEGMENT_60 ( (unsigned char*) 0xFFFF - (60)*SEG_SIZE )


#define SEGMENT_1 (unsigned char*)  0xFC00
#define SEGMENT_2 (unsigned char*)  0xFA00
#define SEGMENT_3 (unsigned char*)  0xF800
#define SEGMENT_4 (unsigned char*)  0xF600
#define SEGMENT_5 (unsigned char*)  0xF400
#define SEGMENT_6 (unsigned char*)  0xF200
#define SEGMENT_7 (unsigned char*)  0xF000
#define SEGMENT_8 (unsigned char*)  0xEE00
#define SEGMENT_9 (unsigned char*)  0xEC00
#define SEGMENT_10 (unsigned char*) 0xEA00
#define SEGMENT_11 (unsigned char*) 0xE800



class MyFlashClass
{
  public:
    void doRead();
    void doEraseSegment(unsigned char* segmentAddress);
     void doEraseAllFlash();
    void doWrite(unsigned char* valueToWrite, int len);
    void doHelp();

};

extern MyFlashClass MyFlash;

#endif
```

HDC1050.cpp

```
/*
 * HDC1050 Temperature/Humidity Sensor Library
 * Kerry D. Wong
 * http://www.kerrywong.com
 * 10/2015
 */

#include "HDC1050.h"
#include <Wire.h>

HDC1050::HDC1050()
{
    configReg = 0x10; //POR default
}

void HDC1050::readRegister(byte regAddr, byte numOfBytes)
{
    Wire.beginTransmission(Addr);
    Wire.write(regAddr);
    Wire.endTransmission();

    if (regAddr == REG_Temperature || regAddr == REG_Humidity) {
     delay(50);
    }

    Wire.requestFrom(Addr, numOfBytes);

    while (!Wire.available());

    for (int i=0; i < numOfBytes; i++) {
     buf[i] = Wire.read();
    }
}

unsigned int HDC1050::getManufacturerID()
{
    readRegister(REG_ManufactureID, 2);
    return buf[0] << 8 | buf[1];
}

unsigned int HDC1050::getDeviceID()
{
    readRegister(REG_DeviceID, 2);
    return buf[0] << 8 | buf[1];
}

String HDC1050::getSerialID()
{
    readRegister(REG_SER_1, 2);
    String s1 = String(buf[0] << 8 | buf[1], HEX);
    readRegister(REG_SER_2, 2);
    String s2 = String(buf[0] << 8 | buf[1], HEX);
    readRegister(REG_SER_3, 2);
    String s3 = String(buf[0] << 8 | buf[1], HEX);

    return String(s1 + s2 + s3);
}

void HDC1050::updateConfigRegister()
{
    Wire.beginTransmission(Addr);
    Wire.write(REG_Config);
    Wire.write(configReg);
    Wire.write(byte(0x00));
    Wire.endTransmission();
}

void HDC1050::turnOnHeater(bool heaterOn)
{
    if (heaterOn)
     configReg |= 1 << BIT_HEATER;
    else
     configReg &= ~(1 << BIT_HEATER);
}

bool HDC1050::batteryOK()
{
    readRegister(REG_Config, 2);
    configReg = buf[0];

    return (configReg & (1 << BIT_BATTERY_OK)) == 0;
}

float HDC1050::getTemperatureHumidity(float &t, float &h)
{
    readRegister(REG_Temperature, 4);
    unsigned int th, tl, hh, hl;

    th = buf[0];
    tl = buf[1];
    hh = buf[2];
    hl = buf[3];

    t = (th << 8 | tl) * 165.0 / 65536.0 - 40.0;
    h = (hh << 8 | hl) * 100.0 / 65536.0;

    return 1.8 * t + 32.0;
}
```

HDC1050.h

```
#ifndef HDC1050_h
#define HDC1050_h


#include <Energia.h>

class HDC1050 {
public:
    static const byte Addr = 0x40;

    static const byte REG_Temperature = 0x00;
    static const byte REG_Humidity = 0x01;
    static const byte REG_Config = 0x02;
    static const byte REG_SER_1 = 0xfb;
    static const byte REG_SER_2 = 0xfc;
    static const byte REG_SER_3 = 0xfd;
    static const byte REG_ManufactureID = 0xfe; //0x5449
    static const byte REG_DeviceID = 0xff; //0x1050

    static const byte BIT_T_RES = 2;
    static const byte BIT_H_RES = 0;
    static const byte BIT_BATTERY_OK = 3;
    static const byte BIT_ACQ_MODE = 4;
    static const byte BIT_HEATER = 5;
    static const byte BIT_RST = 7;

    static const byte T_RES_14 = 0;
    static const byte T_RES_11 = 1;

    static const byte H_RES_14 = 0;
    static const byte H_RES_11 = 1;
    static const byte H_RES_8 = 2;

    HDC1050();
    unsigned int getManufacturerID();
    unsigned int getDeviceID();
    String getSerialID();
    void setTemperatureRes(byte res);
    void setHumidityRes(byte res);
    void turnOnHeater(bool heaterOn);
    bool batteryOK();
    float getTemperatureHumidity(float &t, float &h);
    void updateConfigRegister();
    void readRegister(byte regAddr, byte numOfBytes);

    byte configReg; //higher 8 bits of the configuration register
    byte buf[4];
private:

};
#endif
```

Código Aplicación Móvil Android

```
package com.palomaroyocascajares.datalogger;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.DialogInterface;
import android.os.Build;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.support.v4.app.FragmentManager;
import android.support.v4.widget.DrawerLayout;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import java.text.DecimalFormat;
import java.util.Calendar;
import java.util.Set;

public class MainActivity extends AppCompatActivity {


    private Button mMenu;

    private DrawerLayout mDrawerLayout;
    private DrawerLayout.DrawerListener mDrawerListener;
    private ListView mDrawerList;

    private BluetoothDevice mDevice;
    private BluetoothService bluetoothService;
    Handler mHandler;

    private String mMessageBle;
    private String[] mMenuTitles = {" CONNECT", " GET DATA", " SET TIME", " SLEEP
CONFIGURATION"};


    private final float[] mValues = {23.5f, 31.7f, 24.3f, 23f, 22.5f, 20.9f, 17f, 15.3f, 15.0f, 15.0f, 16.7f, 18f};
    private float[] mTemperatureData;
    private float[] mHumidityData;
    private boolean mBluetoothDataCompleter= false;

    private String mMinutes;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        DataFragment fragment = new DataFragment();
        Bundle bundle = new Bundle();
        bundle.putFloatArray(DataFragment.ARG_DATA_FRAGMENT,mValues);
        fragment.setArguments(bundle);

        FragmentManager fragmentManager = getSupportFragmentManager();
        fragmentManager.beginTransaction().replace(R.id.content_frame,fragment).commit();


        final View view = this.getWindow().getDecorView();
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
            view.setBackgroundColor(getResources().getColor(R.color.backgroundDark,getTheme()));
        }else{
            view.setBackgroundColor(getResources().getColor(R.color.backgroundDark));
        }
 //       View viewMenu = view.findViewById(R.id.menuid_layout);
//        mMenu = (Button) viewMenu.findViewById(R.id.menu_button);
//        mMenu.setOnClickListener(new View.OnClickListener() {
//            @Override
//            public void onClick(View view) {
//                openDrawer();
//            }
//        });
        mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
        mDrawerList = (ListView) findViewById(R.id.left_drawer);
        // Set the adapter for the list view
        ArrayAdapter<String > adapter = new
ArrayAdapter<String>(this,R.layout.drawer_list_item,mMenuTitles);
        mDrawerList.setAdapter(adapter);
        // Set the list's click listener
        mDrawerList.setOnItemClickListener(new DrawerItemClickListener());
//        mDrawerLayout.setDrawerListener(mDrawerListener);


        mHandler = new Handler() {
            @Override
            public void handleMessage(Message msg) {

                switch (msg.what) {

                    case BluetoothService.MESSAGE_READ:

                        byte[] readBuf = (byte[]) msg.obj;
                        // construct a string from the valid bytes in the buffer

                        String readMessage = new String(readBuf, 0, msg.arg1);
                        if(!readMessage.contains("#")){
                            mMessageBle += readMessage;
                        }else{
                            mMessageBle += readMessage;
                            readMessage = "";
                            mBluetoothDataCompleter = true;
                            Parse.parseData();
                        }
                        Log.e("Bluetooth message: ", readMessage);
                        break;
```

```java
                }
            }
        };

    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }

    public class DrawerItemClickListener implements ListView.OnItemClickListener {

        @Override
        public void onItemClick(AdapterView parent, View view, int position, long id) {
            switch(position){
                case 0:
                    //Connect to the bluetooth device
                    Toast toastConnect = Toast.makeText(getApplicationContext(),"Bluetooth connection",
Toast.LENGTH_SHORT);
                    toastConnect.show();
                    //Check if this device supports Bluetooth
                    try{
                        BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
                        if (mBluetoothAdapter == null) {
                            // Device does not support Bluetooth
                            Toast toastBluetooth = Toast.makeText(getApplicationContext(), "Bluetooth not
supported", Toast.LENGTH_SHORT);
                            toastBluetooth.show();
                        }else {
                            //Connect to the HC-06 module known its mac

                            Set<BluetoothDevice> pairedDevices = mBluetoothAdapter
                                    .getBondedDevices();
                            if (pairedDevices.isEmpty()) {
                                Log.e("Paired Empty...",
                                        "No devices paired...");
                                return;
                            }

                            for (BluetoothDevice device : pairedDevices) {
                                Log.d("Device paired", "Device : address : " + device.getAddress() + " name :"
                                        + device.getName());
                                if (BluetoothService.HCO6_ADRESS.equals(device.getAddress())) {
                                    mDevice = device;
                                    Log.e("Eguals if", "HC06 device obtained");
                                    break;
                                }
                            }

                            bluetoothService = new BluetoothService(mBluetoothAdapter, mDevice, mHandler);
                            bluetoothService.connect();
                        }
                    }catch(Exception e){
                        //Bluetooth not connected
                        break;
                    }
                    break;
                case 1:
                    //Get Data
                    //Get the data the chart will be updated on the handler
                    byte red = 1;
                    byte[] myByte = new byte[red] ;
                    try{
                        bluetoothService.write("1");
                        Toast toast = Toast.makeText(getApplicationContext(),"Receiving
data...",Toast.LENGTH_LONG);
                        toast.show();

                    }catch (Exception e){
                        //Bluetooth not connected
                        break;
                    }
                    //Update graphics with new data
                    LineChart.updateGraphics(Parse.getTemperatureValues(mMessageBle),
Parse.getHumidityValues(mMessageBle));

                    //Generate the datalogger.txt file and save it in the internal storage
                    File.generateDataFile(Parse.getTemperatureValues(mMessageBle),
Parse.getHumidityValues(mMessageBle));
                    break;
                case 2:
                    //Configure Time
                    //Send time to the device
                    try{
                        bluetoothService.write(getTime());
                    }catch (Exception e){
                        //Bluetooth not connected
                        break;
                    }

                    break;
                case 3:
                    //Configure Sleep
                    //Set minutes to sleep
                    //CustomDialog dialog = new CustomDialog(MainActivity.this);
                   // dialog.show();
                    showMinutesDialog();
                    break;
                case 4:
                    //Send data.txt to mail
                    Mail.sendMail();
                default:
                    break;
            }
            // Highlight the selected item, update the title, and close the drawer
            mDrawerList.setItemChecked(position, true);
```

```
            setTitle(mMenuTitles[position]);
            mDrawerLayout.closeDrawer(mDrawerList);
        }
    }

    public String getTime(){
        DecimalFormat decimalFormat = new DecimalFormat("00");
        Calendar calendar= Calendar.getInstance();
        String minutes = decimalFormat.format(Double.valueOf(calendar.get(Calendar.MINUTE)));
        Log.d("minutes",minutes);
        String hours = decimalFormat.format(Double.valueOf(calendar.get(Calendar.HOUR)));
        Log.d("horas", hours);
        String day= decimalFormat.format(Double.valueOf(calendar.get(Calendar.DAY_OF_MONTH)));
        Log.d("diadelmes",day);
        String month = decimalFormat.format(Double.valueOf(calendar.get(Calendar.MONTH)+1));
        Log.d("mes", month);

        String actualTime = "0" + month + day + hours +minutes;
        return actualTime;
    }

    public void setTitle(CharSequence title) {
        TextView menu = (TextView) findViewById(R.id.list_item_text);
        menu.setText(title);
    }




    public void showMinutesDialog() {
        AlertDialog.Builder dialogBuilder = new AlertDialog.Builder(this);
        LayoutInflater inflater = this.getLayoutInflater();
        final View dialogView = inflater.inflate(R.layout.dialog_layout, null);
        dialogBuilder.setView(dialogView);

        final EditText edt = (EditText) dialogView.findViewById(R.id.edit_minutes);

        //dialogBuilder.setTitle("Custom dialog");
        //dialogBuilder.setMessage("Enter text below");
        dialogBuilder.setPositiveButton("Done", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {
                //do something with edt.getText().toString();
                mMinutes = edt.getText().toString();
                Log.d("Minutes", mMinutes);
                String data = "2" + mMinutes;
                try {
                    bluetoothService.write(data);

                } catch (Exception e) {
                    //Error
                }
            }
        });
        dialogBuilder.setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {
                //pass
            }
        });
        AlertDialog b = dialogBuilder.create();
        b.show();
    }
    public void openDrawer(){
        mDrawerLayout.openDrawer(mDrawerLayout);
    }




}
```

Código Energia

```
/* MSP430G2553 & HDC1050 DATALOGGER:
* This code reads the sensor(HDC1050 TI) temperature and humidity data
* each 5 minutes and store it on flash memory.
* When the user press button p1.3 then an interrupt occurs and all the
* data stored on flash memory will be sent trough Bluetooth.
*
*
* Developer: Paloma Royo Cascajares
* 2016
*/

//Include libraries for this project
#include <HDC1050.h>
#include <Wire.h>
#include <MyFlash.h>
#include <MspFlash.h>
#include <MyRTC.h>
#include <Parse.h>

//variables to create string data
char str[80];
int intTemp;
int intHum;
String dots = ":";

//HDC1050 variables
HDC1050 hdc1050;
float tc;
float tf;
float hum;
String dataToSend;

//Send data trough bluetooth
char cadena[20];
char serialBuffer[13];
String serialString;
int bluetooth=0;
char datoMobile;
int bluetoothTimer;

unsigned char *data;
int i;
int minutes;
int userMinutes;
int totalBytes;
int n;

String defaultTime = "08072245";

void setup() {
    //Initialice the launchpad
    initialice();
    MyFlash.doEraseAllFlash();
    pinMode(P2_0, OUTPUT);
    pinMode(P2_1, OUTPUT);
    attachInterrupt(PUSH2, blink, FALLING);
    minutes = 5;
    userMinutes = 5;
    bluetoothTimer = 120; //If Bluetooth module is not connected to any
device in two minutes it will be turned off
    MyRTC.setTime(defaultTime);

}

void loop() {

    Serial.begin(9600);
    Serial.println("loop");
    digitalWrite(P2_0, HIGH);
    digitalWrite(P2_1, HIGH);

    if(minutes== userMinutes){
     minutes = 0;

     //Power the sensor
     digitalWrite(P2_0, LOW);
     delay(1000);

     //Read sensor data
     Wire.begin();
     tf = hdc1050.getTemperatureHumidity(tc,hum);
     dataToSend = getDataString(tc, hum);
     //dataToSend = "t:21.57h:34.67";
     data =(unsigned char*)dataToSend.c_str();

     Serial.println("data measurement");
     Serial.println(dataToSend);

     //Write data to flash
     if(dataToSend.length()!=0){

          MyFlash.doWrite(data,dataToSend.length());
          Serial.print("length: ");
          Serial.println(dataToSend.length());
     }
     //Turn off the sensor
     digitalWrite(P2_0,HIGH);
    }

    //Bluetooth connection
    if(bluetooth==1){
     Serial.begin(9600);
     digitalWrite(P2_1, LOW);
     delay(1000);
     Serial.flush();
     while(!Serial.available() && bluetoothTimer>0){
```

```
                delay(1000);
                bluetoothTimer--;
        }
        while(Serial.available()>0)
        {
                totalBytes = Serial.available();
                Serial.print("totalBytes: ");
                Serial.println(totalBytes);
                datoMobile = Serial.read();
                cadena[i++]=datoMobile;
                if(datoMobile !=1 && totalBytes > 1){
                        serialBuffer[0] = datoMobile;
                        serialBuffer[1] = Serial.read();
                        serialBuffer[2] = Serial.read();
                        serialBuffer[3] = Serial.read();
                        serialBuffer[4] = Serial.read();
                        serialBuffer[5] = Serial.read();
                        serialBuffer[6] = Serial.read();
                        serialBuffer[7] = Serial.read();
                }

                Serial.flush();
        }

        delay(1000);

        if(datoMobile == '1')
        {
                Serial.println("BluetoothData: ");
                MyRTC.getTime();
                MyFlash.doRead();
                Serial.println("#");
                delay(1000);
                bluetooth = 0;
                digitalWrite(P2_1, HIGH);
        }if(datoMobile == '2'){
                userMinutes = Parse.getData(serialBuffer);
        }else{
                Serial.println("Setting time... ");
                MyRTC.setTime(serialBuffer);
                MyRTC.getTime();
                bluetooth = 0;
                delay(1000);
                digitalWrite(P2_1, HIGH);
        }
        bluetoothTimer= 120;
       }



      minutes ++;
      // Go to sleep
      sleep(5000);
}

//Interrupt function, launched when push button pressed
```

```
void blink(){
    bluetooth = 1;
    wakeup();
}



String getDataString(float temperature, float humidity){
    String datoTotal="";

    intTemp = int(temperature*100.0);
    Serial.println(intTemp);

    int entero1 = intTemp/1000;
    int entero2 = (intTemp%1000)/100;
    int decimal1 = (intTemp%100)/10;
    int decimal2 =(intTemp%10);

    unsigned char uno = char(entero1);
    unsigned char dos = char(entero2);
    unsigned char tres = char(decimal1);
    unsigned char cuatro = char(decimal2);


    intHum = int(humidity*100.0);
    Serial.println(intHum);

    int enteroH1 = intHum/1000;
    int enteroH2 = (intHum%1000)/100;
    int decimalH1 = (intHum%100)/10;
    int decimalH2 =(intHum%10);

    unsigned char unoH = char(enteroH1);
    unsigned char dosH = char(enteroH2);
    unsigned char tresH = char(decimalH1);
    unsigned char cuatroH = char(decimalH2);

    datoTotal = uno + dos + tres + cuatro + dots + unoH + dosH +tresH +
cuatroH + dots;

    return datoTotal;
}

void initialice(){

    P1DIR |= (BIT0 + BIT6); // Set P1.0 to output direction

    P1OUT &= ~(BIT0 + BIT6); // set P1.0 to 0 (LED OFF)
    P1IE |=  BIT3;                          // P1.3 interrupt enabled
    P1IES |= BIT3;                          // P1.3 Hi/lo edge
    P1REN |= BIT3;                          // Enable Pull Up on SW2
(P1.3)
    P1IFG &= ~BIT3;
}
```