Instituto Tecnológico de Buenos Aires

Proyecto Final de Ingeniería Electrónica

Sistema de medición del desempeño de colectores solares

Alumnos:
Rodrigo Iribarren 53171
Rocco Rondinella 53201
Julián Tachella 53233

Tutor:
Alejandro Ugarte

12 de diciembre de 2016

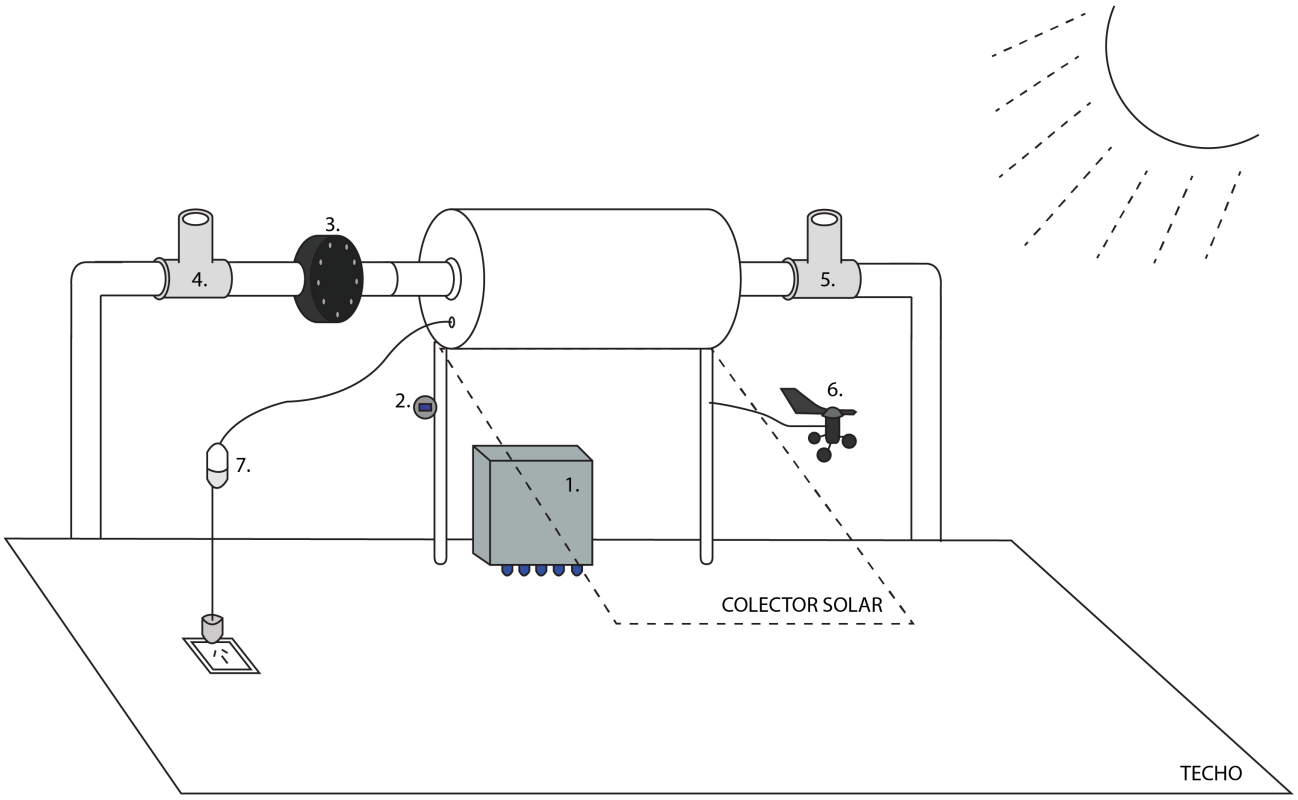
Índice

1. Agradecimientos

Queremos agradecer a los profesores Alejandro Ugarte, Nicolás Nemirovsky y Ricardo Pingitore, cuyas correcciones, aportes y sugerencias enriquecieron notablemente este trabajo. También agradecerle al profesor Claudio Muñoz por considerarnos para este proyecto y ayudarnos en la fase inicial del mismo.

2. Resumen

Este trabajo documenta el diseño de un sistema de medición de las variables relacionadas al rendimiento de colectores solares térmicos. El trabajo fue realizado en el marco de un proyecto multidisplinario propuesto por el ITBA y la fundación FOVISEE, que busca promover la instalación de colectores solares en barrios carenciados de Argentina. El producto desarrollado es una alternativa de bajo costo y bajo consumo para medir el desempeño de los colectores solares instalados en el barrio de La Perla, Moreno, Buenos Aires, Argentina.



1. Gabinete
2. Sensor de radiación
3. Caudalímetro
4. Sensor de temperatura de entrada
5. Sensor de temperatura de salida
6. Anemómetro
7. Amperímetro

Figura 1: Esquema del sistema de medición y sus sensores

3. Introducción

3.1. Historia, antecedentes

La energía térmica proveniente del sol, posee múltiples aplicaciones, una de las más relevantes es la obtención de agua caliente para uso doméstico. En la última década en Argentina, surgieron muchas PyMes dedicas al diseño y fabricación de colectores solares térmicos. Desde el año 2008, el Instituto Nacional de Tecnología Industrial, INTI, brinda asistencia a la industria nacional a través de la Plataforma Solar Térmica ubicada en la sede de San Martín. Dicha plataforma posee el equipamiento necesario para realizar mediciones de los productos de los fabricantes a partir de simulaciones de consumo.

En el año 2010, se llevó a cabo la iniciativa 'Casas por + energía', en el cual el INTI, la municipalidad de Moreno, EDENOR y la fundación FOVISEE (Foro de Vivienda Social y Eficiencia Energética) se unieron para instalar 10 colectores solares en el barrio La Perla de la localidad de Moreno (ver figura 2). La experiencia obtenida en ese primer trabajo, desembocó el año siguiente en el proyecto 'Energía solar en la vivienda social', con el propósito de desarrollar más de 2000 modelos económicos de colectores solares para barrios de casas construidas por el Estado. Sin embargo, a fin de estimar la potencialidad de la iniciativa, surgió la necesidad de evaluar el desempeño de los colectores solares ya instalados en Moreno.

Figura 2: Instalación de colectores solares en Moreno

Debido a que la plataforma solar térmica del INTI no permite evaluar el verdadero rendimiento de las prestaciones de los colectores, dado que las mismas se ven afectadas en gran medida por la zona geográfica en que se encuentran y la frecuencia de uso, surgió la necesidad de un sistema de medición económico, de bajo consumo eléctrico, que pueda ser instalado en Moreno y suministrar los datos de las mediciones a través de Internet.

A principios del año 2015, la fundación FOVISEE se comunicó con el Instituto Tecnológico de Buenos Aires, ITBA, con la propuesta de diseñar dicho sistema de medición. Los directores del proyecto, la Dra. Smoglie y el ingeniero De Bernardez, armaron un equipo multidisciplinario de alumnos de quinto año de Ingeniería Industrial, Mecánica y Electrónica, para desarrollar la tarea en cuestión. El equipo de Ingeniería Electrónica fue encargado con el diseño del sistema de medición y los equipos de Ingeniería Mecánica e Ingeniería Industrial fueron asignados con el análsis estadístico de las mediciones tomadas por dicho sistema. A mediados de octubre del mismo año, el equipo realizó una visita a la plataforma solar térmica del INTI y tomó nota de cuales eran las necesidades que el sistema de medición debía cumplir. Con el conocimiento adquirido en esta visita, los alumnos

del Departamento de Electrónica comenzaron a trabajar en el diseño de un sistema de medición de bajo costo y bajo consumo, que pudiera ser instalado en los colectores del barrio de La Perla.

3.2. Definiciones, glosario de términos

Anemómetro

Sensor capaz de medir la velocidad y dirección del viento. La medición de la velocidad del viento se logra mediante un conjunto de copas, denominado molinete. La medición de la dirección del viento se lleva a cabo con una veleta.

Bit

Unidad mínima de información de un sistema binario. En electrónica digital suele representarse como una tensión alta y una tensión baja.

C++

Es un lenguaje de programación de medio y/o alto nivel, que permite programar desde microcontroladores hasta complejos programas que corren en diferentes dispositivos, como computadoras, celulares, etc.

CFM

Cubic Feet per Minute, es una unidad utilizada en la industria para medir el flujo de aire de un sistema de ventilación. Mide la catidad de pies cúbicos por minuto de aire que el sistema es capaz de sumistrar.

Chars

Es un tipo de datos de 1 byte de tamaño, es decir 8 bits, cuyo tamaño es el mínimo necesario para representar el set completo de caracteres pertenecientes al código UTF-8. Este código contiene todas las letras del idioma inglés (desde la a hasta la z), y otros caracteres especiales.

CIDIM

La sigla CIDIM hace referencia al Centro Integrado de Desarrollo en Ingeniería Mecánica del Instituto Tecnológico de Buenos Aires, un laboratorio de mecánica ubicado en la sede de Parque Patricios del ITBA.

CIPSR

La CIPSR es una normativa europea que provee una regulación estandar para las emisiones electro-magnéticas de equipos multimedia.

Colector solar

Es un dispositivo que permite convertir radiación solar en energía térmica. Se utiliza para calentar agua mediante diversos metodos. Existen dos tipos de colectores solares: De placa plana y de tubos de vacío. Los colectores de tubos de vacío poseen mayor eficiencia que los colectores de placa plana, ya que el metal receptor de la energía solar se encuentra dentro de tuberías de vidrio al vacío que reducen las pérdidas por convección y conducción.

Colector de placa plana

Un colector solar plano consta de un radiador y un tanque de agua aislados térmicamente. El radiador posee cañerías por donde fluye el agua, que es calentada por la energía solar. Debido al efecto termosifón, el agua caliente sube y se almacena en el tanque, mientras que la fría se mantiene en el radiador. En la figura 3 se muestra un colector solar de placa plana.

Figura 3: Colector solar de placa plana.

Colector solar de tubos de vacío

Un colector solar de tubos de vacío está formado por conductos de metal (en su mayoría cobre) alojadas en tubos de vidrio al vacío. Dentro de cada conducto circula un fluido caloportador que recibe la energía solar captada por el metal. Existen dos categorías de colectores de tubos de vacío: De flujo directo y flujo indirecto o *'heat pipe'*. En los colectores de flujo directo, el agua ingresa al conducto dentro del tubo de vidrio, se calienta por la energía solar y sale del conducto debido al efecto termosifón. En los colectores de flujo indirecto o *'heat pipe'*, el agua no circula dentro de los tubos de vacío. Cada conducto contiene un líquido de transferencia de calor (típicamente propilenglicol), que se evapora al recibir calor del sol y asciende al extremo superior del tubo. Luego el líquido caloportador evaporado transfiere el calor al agua circulante por el tanque, se enfría y vuelve al extremo inferior del tubo. Los colectores de tecnología 'heat pipe' son más resistentes a sobretemperaturas en climas calurosos que los colectores de flujo indirecto. En la figura 4 se pueden ver los colectores solares de tubos de vacío instalados en el Centro Integrado de Desarrollo en Ingeniería Mecánica, CIDIM, del ITBA.

Figura 4: Colector solar de tubos de vacío.

Comando AT

Es el lenguaje desarrolado por la compañia Hayes Communications. Estos comandos son utilizados en modems y cualquier dispositivo que sea compatible. El acrónimo AT significa "Atention", y precede todas las intrucciones en la posible lista de comandos.

Conversor digital-analógico

Un conversor digital-analógico, DA, es un dispositivo de hardware que se encarga de convertir una señal digital en una analógica, a una frecuencia de muestreo y tasa de bits específica de entrada.

Conversor analógico-digital

Un conversor analógico-digital, AD, es un dispositivo de hardware que se encarga de convertir una señal analógica en una digital, a una frecuencia de muestreo y tasa de bits específica de salida.

Corriente de inrush

La corriente de inrush es el valor pico de corriente que requiere un dispositivo apenas se lo enciende o enchufa a la red.

EMI

El término ElectroMagnetic Interference se refiere a la perturbación que ocurre en un circuito, componente o sistema electrónico causada por una fuente de radiación electromagnética externa o interna.

EMS

El término ElectroMagnetic Sensitivity se refiere a la sensibilidad de un sistema electrónico frente a radiación electromagnética en cuanto a su durabilidad y degradación de desempeño.

Efecto Termosifón

Es un fenómeno que se produce en los fluidos cuando se calientan. Una sustancia se dilata al calentarse y disminuye su densidad. La porción más caliente del fluido tiene menor densidad, de modo que asciende sobre la porción de fluido más fría. Este efecto es responsable del intercambio de calor por convección. También puede servir para provocar una circulación natural en los ambientes habitados o en redes de tuberías.

Evaluation Board

La frase Evaluation Board proviene del inglés y significa placa de desarrollo. Es un circuito impreso que incluye un microprocesador y cuenta con hardware (numerosos pines de entrada y salida, una comunicación serie para programar el microprocesador, una interfaz de depuración de software, LED indicadores de estado, etc) dedicado a la prueba de dicho microprocesador, que facilitan la programación, prueba de módulos y depuración de errores.

FOVISEE

El Foro de la Vivienda Social y Eficiencia Energética es una fundación sin fines de lucro que tiene como objetivo proveer soluciones energéticas a hogares de bajos recursos. Entre sus proyectos actuales, se encuentra la instalación de colectores solares en hogares carenciados. (Un enlace a su página web se encuentra en 13.3).

FPGA

Field Programmable Gate Array (FPGA) es un dispositivo programable que contiene bloques de lógica cuya interconexión y funcionalidad puede ser configurada mediante un lenguaje de descripción especializado (VHDL, Verilog u otro). La lógica programable puede reproducir desde funciones sencillas como una compuerta lógica hasta sistemas complejos como un microprocesador.

GPIO

General Pourpose Input-Output (GPIO) es un módulo del microcontrolador encargado de administrar los pines, configurándolos como entradas o salidas según lo indique el software. El sistema GPIO también es capaz de realizar lecturas o escrituras en los pines según la funcionalidad tengan asignados.

I2C

Inter-Integrated Circuit (I2C) es un bus de comunicación sincrónica serie que es utilizado para la comunicación entre microprocesadores y periféricos de baja velocidad de corta distancia. Fue desarrollado por Philips en la década de 1980, pero en la actualidad no existen cargos de licenciamiento sobre el mismo.

IDE

Integrated Development Environment (IDE) es un entorno de desarrollo que permite programar aplicaciones. Normalmente es provisto por el fabricante del hardware y otras veces simplemente por terceros.

IEC529

La norma IEC529 especifica la protección que posee un sistema respecto de cuerpos extraños, objetos sólidos y el ingreso de agua u otros fluidos. El grado de protección se indica como IP XY donde X es la protección contra sólidos y Y protección contra el ingreso perjudicial de agua.

Internet of Things

Internet of Things (IoT) es un concepto que se refiere a la utilización de tecnologías de comunicaciones y sistemas embebidos, para medir y controlar variables automáticamente a través de Internet.

INTI

La sigla INTI hace referencia al Instituto Nacional de Tecnología Industrial, una institución del Estado argentino dedicada a brindar ayuda a productores industriales, especialmente de pequeña escala. Este organismo estatal ofrece servicios de capacitación y soporte para el desarrollo tecnológico de la industria argentina.

PCB

El acrónimo PCB proviene del inglés donde sus siglas significan Printed Circuit Board y en español hace referencia a los circuitos impresos.

PLC Programmable Logic Controller (PLC), controlador lógico programable en español, es una computadora para automatizar procesos electromecánicos, que está diseñada para manejar múltiples señales de entrada y de salida, y trabajar en rangos de temperatura ampliados, con mayor inmunidad al ruido eléctrico y resistencia a la vibración y al impacto.

Pull Up

El término es utilizado en el campo de la electrónica digital para indicar que en algún punto del circuito la tensión está fijo a un nivel de referencia y sólo va a modificarse si algún dispositivo conectado en ese punto fuerza una cambio de tensión. Normalmente se implementa con una resistencia o divisor resistivo conectado a la alimentación.

Red GPRS

El General Packet Radio Service, es una red comunicaciones celulares creado en 1980 como extensión del sistema Global System for Mobile Communications (GSM). Esta red permite conectarse a Internet para poder subir datos con el uso de una tarjeta SIM. Se la conoce tambien como red 2.5G, ya que es el punto intermedio entre 2G y 3G.

Shield

Shield es el nombre que se le da a un circuito impreso que une diferentes partes del hardware, usualmente montado sobre la placa principal haciendo las veces de un ¨escudo¨.

SPI

Serial Peripheral Interface, SPI, es un bus de comunicación sincrónico serie que es utilizado en comunicaciones de corta distancia en sistemas embebidos. Fue desarrollado por Motorola en la década de 1980, pero luego se convirtió en un estándar de la industria de sistemas embebidos.

UART

El módulo Universal Asynchronous Receiver-Transmitter, UART, es un dispositivo de hardware que viene integrado en microcontroladores u otros dispositivos electrónicos, que se encarga de manejar una comunicación asincrónica serie, siguiendo los estándares RS-232, RS-422 o RS-485 (para más información ver [2]).

3.3. Justificación del proyecto

Este proyecto comenzó a partir de la iniciativa del ITBA y la organización FOVISEE, que crearon un equipo multidisciplinario de alumnos del último año de Ingeniería Industrial, Mecánica y Electrónica con el fin de promover y desarrollar la utilización de colectores solares térmicos en sectores carenciados del país. El Departamento de Ingeniería Electrónica fue encargado con el desarrollo de un sistema de medición del desempeño de colectores solares, que pueda ser instalado en los barrios donde los colectores ya fueron colocados. De esta forma, se busca desarrollar una alternativa de bajo costo que sustituya la simulación del desempeño en las instalaciones del INTI, y provea datos reales del rendimiento de los colectores instalados. Cabe destacar que el proyecto cuenta con el apoyo del Banco Santader Río, que otorgó un presupuesto a cada Ingeniería.

El Departamento de Ingeniería Mecánica va a poder estimar con los datos recibidos la eficiencia de cada colector según el lugar de instalación, y luego elegir el colector correcto para cada región. Asimismo, el Departamento de Ingeniería Industrial utilizará la información relevada para calcular la rentabilidad de instalar dichos colectores en cada sector y la reducción de emisiones de dióxido de carbono relacionada al uso de colectores solares en comparación con alternativas no renovables. Los esfuerzos de cada equipo confluyen en la necesidad de demostrar que la inversión en colectores solares es una alternativa rentable y eficiente que debe ser utilizada por el Estado Argentino para otorgar agua caliente a sectores carenciados.

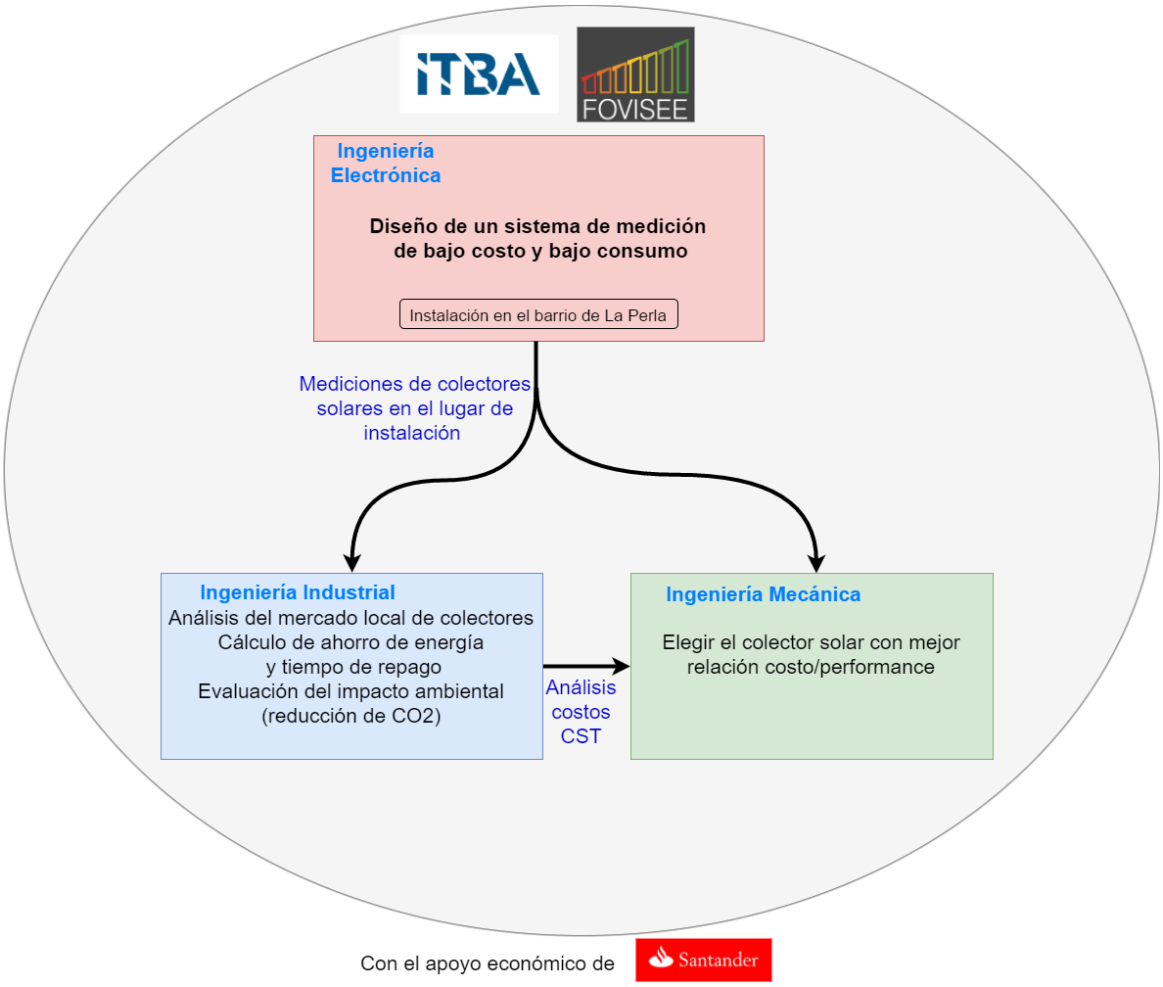


Figura 5: Diagrama de alcance del proyecto

4. Objetivos

Se debe diseñar un sistema de medición de las variables relacionadas al desempeño de colectores solares térmicos, que sea económico, de bajo consumo, sencillo de instalar y que entregue los datos relevados directamente al usuario a través de Internet.

4.1. Finalidad del proyecto

Desarrollar un sistema de adquisición de datos de colectores solares que entregue las mediciones en tiempo real a los equipos de Ingeniería Industrial y Mecánica del ITBA.

4.2. Planteamiento del problema a resolver

Crear un sistema versátil, económico, instalable en la intemperie y de bajo consumo que pueda medir las variables relacionadas al desempeño de un colector solar térmico en su lugar de instalación.

5. Definición del producto

5.1. Requerimientos

El principal cliente de este proyecto es el ITBA y FOVISEE, que mediante una serie de reuniones y la ayuda del INTI, nos transmitieron los siguientes requerimientos:

1. Se debe medir el caudal de agua del colector.
2. Se debe medir la temperatura de entrada y salida de un colector de una casa promedio de 4 personas.
3. Se debe medir la radiación solar que recibe el colector, pero sin una precisión elevada.
4. El sistema debe obtener la velocidad y dirección del viento donde esta instalado el colector.
5. El sistema debe poder medir el consumo de corriente eléctrica de un colector que posee la ayuda de un termotanque.
6. El sistema debe ser de bajo consumo de potencia, y debe disponer de un modo de bajo consumo que apague el sistema cuando hay menos de $1\frac{litro}{min}$ de caudal circulando por el colector.
7. El período de las mediciones debe ser configurable de forma remota, con un tiempo mínimo entre muestras de 3 minutos y un máximo de 1 medición por día. Si hay caudal circulando por el colector, las mediciones se deben tomar cada un mínimo de 3 minutos o menos, independientemente del tiempo configurado.
8. El costo del sistema debe no ser significativo frente al costo del colector.
9. El relevamiento de datos debe ser remoto y no depender de una conexión tipo Ethernet o WiFi. Se debe utilizar la red celular para transmitir los datos, con tecnología GPRS u otra que tenga igual o mayor área de cobertura en la Argentina.
10. Los datos obtenidos deben poder leerse en un celular, mediante una aplicación compatible con Android que los baje de Internet. Además, deben poder descargarse desde cualquier computadora para realizar estudios estadísticos. Se deben poder almacenar todas las mediciones correspondientes a un período mayor a 2 años.
11. El sistema debe poder realizar todas sus funciones en el techo de una vivienda, es decir a la intemperie, por períodos mayores a 2 años.
12. Los datos, además de ser enviados por la conexión remota, deberán ser almacenados en una memoria dentro del gabinete.
13. El sistema debe recibir alimentación de la tensión domiciliaria de $220V_{rms}$.
14. La distancia máxima entre los sensores del sistema y el gabinete debe ser mayor a 1 metro.

5.1.1. Construcción de la casa de calidad

La casa de calidad se muestra en la figura 6.

Figura 6: Casa de calidad

5.2. Especificaciones funcionales y de diseño

En esta sección se analizan las especificaciones de hardware y software de diseño. Junto al nombre de cada especificación se aclara su fuente entre paréntesis, ya sea un requerimiento de cliente (req. xx), propio de diseño (propio), o una normativa nacional (norm.).

5.2.1. Especificaciones de hardware

1. Tensión de alimentación (req. 13): Conexión a la línea domiciliaria de $220V_{rms}/50Hz$ con una variación de +10 % -20 % en la tensión nominal y ±1 % en la frecuencia nominal.

2. Grados de protección $IEC529$ (req. 11): El gabinete cerrado con todos los cables conectados debe cumplir con IP55.

3. Temperaturas (req. 11): Se debe poder soportar una temperatura ambiente máxima de $60°C$ y una temperatura mínima de $-5°C$ en funcionamiento continuo para mantener las prestaciones.

4. Medición de intensidad solar (req. 3): Debe medirse la irradiancia total en un rango mayor o igual a las longitudes de onda comprendidas entre $[500, 600]$ nm con una máxima diferencia de atenuación de 0,5. El error absoluto, a fondo de escala, debe ser menor a $50\frac{W}{m^2}$ en el rango espectral mencionado. Debe poder medirse en el rango de $0\frac{W}{m^2}$ a $200000\frac{W}{m^2}$.

5. Medición de temperatura del agua de entrada y salida del colector solar (req. 2): Debe medirse la temperatura del agua en el rango de $[0, 84]°C$, con un error absoluto menor a $0{,}8°C$ y entre $(84, 100]°C$ con un error absoluto menor a $4°C$.

6. Medición de caudal de agua (req. 1): Se debe medir el caudal de agua en la entrada o salida del colector en el rango de $[0, 50]\frac{litros}{minuto}$, con un error absoluto menor a $1\frac{litro}{minuto}$. El caudalímetro debe soportar agua con un máximo de dureza de 180 ppm de $CaCO_3$.

7. Medición del viento (req. 4): Se debe medir la velocidad del viento en el rango de $[0, 300]\frac{km}{hora}$, con un error absoluto máximo de $5\frac{km}{hora}$. También se debe medir la dirección del viento con una precisión de 1 octante.

8. Medición de consumo del termotanque (req. 5): Se debe medir la corriente instantánea consumida por el termotanque en el rango de $[0, 10]A_{rms}$ con un error menor a $0{,}5A_{rms}$.

9. GPRS (req. 9): El sistema debe poseer conexión con Internet mediante la red de datos móviles $GPRS$.

10. Memoria interna (req. 12): El sistema debe poseer una memoria interna que almacene los datos del desempeño del colector de un período mayor a 2 años.

11. Potencia (req. 6): Debe consumir menos de $1W$ promedio por dia, y picos de potencia (de duración máxima de $1\mu seg$) menores a $15W$. Durante los intervalos en los que no se realizan mediciones, el consumo debe ser menor a $300mW$. La corriente de inrush debe ser inferior a $400mA$.

    a) Corrección/Acta de acuerdo (Ver sección 9.1.5): Debe consumir menos de $1W$ promedio por dia, y picos de potencia (de duración máxima de $1\mu seg$) menores a $15W$. La corriente de inrush debe ser inferior a $400mA$.

12. Etiquetado (propio): Todos los sensores deben poseer etiquetas que identifiquen su función y ubicación en la instalación. Además, cada sensor del sistema debe tener sus cables de diferentes colores.

13. Vida útil (req. 11): el producto debe funcionar independientemente por un período mayor a 2 años.

14. Precio al consumidor (req. 14): el producto final debe tener un precio menor a 400 U$D. Aclaración: No se consideran los costos del plan datos necesario para la comunicación.

15. EMI (norm.): Se debe cumplir con el estándar EN 55022/CIPSR 22 para equipos de tecnología de la información.

16. EMS (norm.): Se debe cumplir con el estándar EN 55024 para equipos de tecnología de la información.

17. Peso (req. 11): El peso del producto incluyendo todas sus partes debe ser menor a $10kg$.

18. Dimensiones (req. 11): El gabinete debe tener dimensiones menores a $50cm$ de largo, $50cm$ de ancho y $30cm$ de alto.

19. Distancia máxima sensores (req. 14): La máxima distancia entre un sensor y el gabinete debe ser menor a $1m$.

20. LED indicador de estado (propio): Un LED debe indicar cuando el sistema esta midiendo y cuando está en modo de bajo consumo.

21. Arranque del sistema (req. 13): El sistema debe comenzar a medir al ser conectado a la alimentación, sin la necesidad de realizar un encendido específico.

22. Funcionamiento (req. 11): El sistema debe permanecer realizando mediciones siempre que esté conectado a la alimentación.

23. Apagado del sistema (req. 13): Se debe perder como máximo una medición (la última) al desconectar el sistema.

24. Documentación de hardware (propio): El diseño de cada módulo de hardware debe estar documentado.

5.2.2. Especificaciones de software

1. Aplicación de Android para usuario (req. 10): La aplicación debe mostrar los gráficos correspondientes a todos los datos adquiridos para el colector solar seleccionado por el usuario. El acceso a la información de cada colector debe estar protegida por una contraseña. La aplicación debe ser compatible con la versión *Lollipop*($v5{,}0/v5{,}1$) o posterior.

2. Tratamiento de las mediciones (req. 9): Los datos relevados no serán procesados, pero deben poder descargarse por los usuarios autorizados.

3. Presentación de las mediciones (req. 10): La aplicación debe mostrar los gráficos correspondientes a todas las mediciones con una base de tiempo configurable por el usuario.

4. Almacenamiento remoto (req. 9): Se deben almacenar los datos relevados del colector en un servidor remoto por un período mayor a 2 años. La cuenta del servidor remoto debe poder ser modificada mediante un mensaje SMS a la cuenta de celular asociada con el colector.

5. Almacenamiento interno (req. **12**): Las mediciones se deben almacenar en formato ASCII, con un formato fijo por medición.

6. Precisión del instante de medición (req. 7): El instante de cada medición debe almacenarse con precisión mayor a 1 segundo.

7. Relevamiento de datos (req. 7): Se deben realizar mediciones cada intervalos menores a 2 minutos si el colector presenta caudal mayor a $1\frac{L}{minuto}$, iniciando la medición en menos de 5 minutos posterior al aumento del caudal por encima de $1\frac{L}{minuto}$. El intervalo de medición con caudal menor a $1\frac{L}{minuto}$ debe poder ser configurado por el usuario en valores desde 2 minutos hasta 1440 minutos (24 horas) mediante un mensaje SMS a la cuenta asociada con el colector. Los intervalos de medición no pueden variar más de 1 minuto respecto del valor configurado por el usuario.

8. Contraseñas (req. 10): Los datos de cada colector deben ser accedidos mediante el uso de una contraseña de longitud mayor a 5 caracteres alfanuméricos. Por cada colector debe haber una contraseña configurable.

9. Documentación de software (propio): Todas las rutinas deben tener un comentario al comienzo que explique sus entradas, sus salidas y sus restricciones.

10. Sensores (propio): Si se desconecta un sensor, el resto del sistema debe continuar funcionando independientemente.

6. Análisis de factibilidad

6.1. Factibilidad tecnológica

6.1.1. Propuesta de alternativas de diseño

Las decisiones de diseño más importantes relacionadas al sistema de relevamiento de datos son: el dispositivo que toma los datos, el almacenamiento de datos y los sensores.

En cuanto al sistema embebido encargado del centro de procesamiento, existen tres posibilidades: un microcontrolador, un PLC o una FPGA (Field Programmable Gate Array). El primero presenta la ventaja de ya poseer los módulos de comunicaciones (SPI, UART, I2C) y conversores AD necesarios para la tarea, por lo que la programación de los mismos es más simple y está secundada por una extensa documentación. El PLC ofrece la posibilidad de utilizar sensores estándar y operar en un rango de condiciones más amplio que el microcontrolador, pero su costo es mayor. La FPGA presenta la ventaja de poder paralelizar las mediciones, logrando mayor velocidad de procesamiento. Sin embargo, posee la desventaja de tener que realizar algunos módulos a muy bajo nivel y lidiar con diagramas de tiempos complejos.

Dentro de la opción del microcontrolador, surgen dos alternativas: Se puede utilizar un dispositivo comercial de alto nivel como Arduino o Raspberry Pi, o un microcontrolador de las principales empresas de semiconductores como Texas Instr., NXP, etc. La primera opción cuenta con la ventaja del gran número de librerías de alto nivel gratuitas disponibles en Internet, que simplifican fuertemente la programación y disminuyen drásticamente los tiempos de desarrollo. Su principal desventaja es el mayor costo frente a la segunda opción, cuyos valores comerciales se encuentran unas decenas de dólares por debajo. Además, los microcontroladores profesionales presentan mayor flexibilidad, pero necesitan de una programación en bajo nivel sin la ayuda de numerosas librerías.

Relativo al almacenamiento remoto, existen dos posibilidades: Se puede desarrollar un servidor propio que reciba los datos directamente de la cuenta del colector, interprete las mediciones con un formato propio y las almacene en un disco duro propio. A pesar de independizar al diseño de un tercero, esta opción tiene la desventaja de implicar un mayor costo de armado y mantenimiento del servidor propio. La segunda opción es la de utilizar un servicio de almacenamiento en la nube otorgado por un tercero, y adecuarse al formato de comunicación impuesto por dicho tercero. Esta variante ofrece la ventaja de reducir los tiempos de diseño considerablemente, pero la desventaja de necesitar adecuarse a los posibles cambios impuestos por el proveedor del servicio de almacenamiento.

Los sensores pueden ser tanto industriales como no industriales: Los primeros ofrecen la seguridad de estar certificados, poder medir con menor error y operar en un mayor rango de condiciones, pero en muchas ocasiones necesitan un acondicionamiento de la señal sensada, y son de mayor costo. En cambio, los no industriales operan en un rango reducido de condiciones, pero otorgan una señal de salida en niveles de tensión acordes a un microcontrolador ($5V$ o $3{,}3V$) o mediante protocolos digitales estándar (I2C, SPI, etc.), y tienen un precio menor.

6.1.2. Elección de una solución

El dispositivo de relevamiento de datos más adecuado para este producto es el microcontrolador. Esta opción disminuye el costo y el tiempo de diseño, ya que permite utilizar librerías de drivers para las funciones básicas de cada submódulo, y se encuentra disponible una vasta documentación de cada uno. El microcontrolador tiene prestaciones similares a un PLC considerando la precisión con la que se desean obtener las mediciones, y su precio es significativamente menor (el PLC más económico tiene un costo de 200U$D, mientras que el costo de un microcontrolador promedio ronda los 5 U$D). A pesar de que la programación de la lectura de sensores es más simple en un PLC que en un microcontrolador (por la disponibilidad de librerías dedicadas más fáciles de utilizar), el manejo de la interfaz con el servidor remoto resulta más complejo que con el microcontrolador. Teniendo en cuenta las especificaciones de diseño, no es necesario realizar mediciones en intervalos muy cortos (sólo deben ser menores a 1 minuto), por lo que el uso de una FPGA implicaría un injustificado aumento en la complejidad del diseño. Además, un microcontrolador presenta un costo menor que una FPGA de rendimiento similar, siendo una mejor alternativa para cumplir el requerimiento de que sea un sistema bajo costo.

En relación a la elección de la familia de microcontroladores, se prioriza el precio para poder cumplir el requerimiento de bajo costo, aunque esta elección signifique un número significativamente mayor de horas de desarrollo de drivers y librerías de bajo nivel. Además, los microcontroladores profesionales poseen flexibilidad en cuanto a su consumo de potencia, y simplifican el cumplimiento del requerimiento de bajo consumo del sistema.

Un almacenamiento en la nube, en un servidor provisto por un tercero, es la mejor opción para este diseño, por las siguientes razones: En primer lugar, reduce significativamente los costos involucrados en el armado y mantenimiento de un servidor, y hasta puede obtenerse un almacenamiento gratuito, considerando que los datos

de las mediciones son de poco tamaño (en la sección de ingeniería de detalle se analiza este punto en mayor profundidad). En segundo lugar, disminuye críticamente el tiempo de diseño y desarrollo del producto.

En cuanto a los sensores, en esta aplicación se prioriza la variable económica sobre el mayor rango de operación, por lo que se utilizan sensores no estándares. Esta decisión se basa en que los requerimientos de diseño no suponen una excesiva precisión en las mediciones, pero marcan la importancia del costo final del producto en relación al valor de un colector solar térmico. Un sensor industrial típico, como puede ser el sensor de temperatura PT100 clase B, tiene un costo mayor a 100 U$D, mientras que la sensor para microcontrolador DS18B20 tiene un precio menor a los 3 U$D. Además, la diferencia de precisión entre ambos no es determinante para la aplicación, siendo de $\pm$0,3°C para el sensor industrial y de $\pm$0,5°C para el no industrial.

Proyecto: Estación de medición del desempeño de un colector solar

Participantes: Iribarren, Rodrigo

Rondinella, Rocco

Tachella, Julián

Fecha: 20/08/2016

Versión DFMEA: 2.0

Listado de componentes del diseño:

- Fuentes
  - Fuente regulada de 3.3 y 5 Volt.
- Sensores
  - Caudalímetro.
  - Sensor de temperatura.
  - Medidor de radiación solar.
  - Medidor de dirección del viento.
  - Medidor de velocidad del viento.
  - Sensor de corriente eléctrica.
- Control
  - Microcontrolador.
  - Software de obtención de datos.
- Comunicaciones
  - Placa GPRS.
  - Software de manejo de comunicaciones
- Seguridad
  - Sobretensión de la alimentación de línea.
  - Exposición a la intemperie.
  - Peligros de instalación y mantenimiento.
- Interfaz con el usuario
  - Software para usuario (front end).

Nivel de RPN	
Aceptable	RPN≤27
Bajar hasta razonablemente práctico	27<RPN≤48
No Aceptable	RPN>48

Severidad	Ocurrencia	Detectabilidad	Puntaje
Insignificante	Remota	Completa	1
Poco significativa	Poco probable	Mayor	2
Moderado	Media	Moderada	3
Grave	Alta	Pequeña	4
Muy grave	Muy alta	Mínima	5

Partes	Subsistemas	Posible efecto	Falla potencial	Causa	Severidad	Ocurrencia	Detección	RPN	Control de prevención Control de detección Acciones	Severidad	Ocurrencia	Detección	RPN
Fuentes	Fuente regulada de 3.3 Volt	Mal funcionamiento del microcontrolador, daño irreversible a circuitos internos.	Tensión de salida mayor a 5V.	Sobretensión de alimentación de entrada.	4	3	5	60	Utilizar una fuente que soporte mayores variaciones de tensión. Protección contra sobretensión en la entrada del microcontrolador. Realizar ensayos de sobretensión.	4	1	5	20
		Mal funcionamiento del microcontrolador. Las mediciones no se toman correctamente.	Tensión de salida menor a 3V	Cortocircuito de la salida de la fuente. Corte en la red de línea.	3	2	5	30	Verificar la actualización de datos en el servidor remoto. Utilizar un LED que indique si el microcontrolador está recibiendo alimentación.	3	2	2	12
Sensores	Caudalímetro	Mediciones con error.	Rotura por caudal excesivo	Sobrecarga del caudal del colector.	4	3	4	48	Prevenir al usuario de no exceder el límite de carga del colector. Limitar el uso a colectores que no puedan superar el límite. Guardar en la memoria interna una advertencia si se supera el límite especificado. Manual de Usuario	4	1	4	16
		Se cortan las mediciones del caudalímetro	La paleta del sensor deja de girar correctamente.	Incrustaciones de carbonatos en el caudalímetro por aguas duras.	4	3	3	36	Asegurar que el caudalímetro utilizado soporte un máximo de dureza de 180 ppm de $CaCO_3$	4	1	3	12
	Sensor de temperatura	Mediciones con error	Rotura por alta temperatura.	Sobrepico de temperatura de salida. Zona de calor muy elevado.	4	1	4	16					

Partes	Subsistemas	Posible efecto	Falla potencial	Causa	Severidad	Ocurrencia	Detección	RPN	Control de prevención Control de detección Acciones	Severidad	Ocurrencia	Detección	RPN
Sensores	Sensor de temperatura	Mediciones con error	Rotura por bajas temperatura.	Congelamiento de la tubería.	4	3	4	48	Asegurar que el sensor soporte temperaturas inferiores a -5°C.	4	1	4	16
	Medidor de radiación solar.	Los datos obtenidos no se condicen con la radiación que recibió el colector.	No se relevan datos de la radiación que llega al colector correctamente.	Se colocó en un lugar que no llega la radiación directamente.	3	4	3	36	Proveer un manual con instrucciones claras de instalación de la estación. Proveer cables de 1 metro de largo desde el microcontrolador hasta el sensor.	3	2	3	18
	Todos los sensores	Los datos obtenidos del sensor no se condicen con el valor real.	Descalibración de un sensor.	La ganancia o el offset de un sensor difieren del valor por programado.	4	4	3	48	Proveer la posibilidad de calibrar la ganancia y el offset de cada sensor.	2	4	3	24
	Medidor de dirección del viento	No se relevan datos de la medición correctamente.	Mediciones con un error fijo.	Se colocó con una referencia equivocada.	3	4	3	36	Proveer un manual con instrucciones claras de instalación del anemómetro.	4	1	3	12
	Medidor de velocidad del viento	No se relevan datos de la medición correctamente.	Mediciones más bajas/altas según la dirección	Se colocó en un sector donde no llega el viento.	4	3	4	36	Proveer un manual con instrucciones claras de instalación del anemómetro	4	1	4	16
Control	Microcontrolador	Mal funcionamiento del microcontrolador.	Descarga electrostática	Colocación de integrados sin prevención de ESD.	5	2	3	30	Fabricar las placas con terceros certificados.	5	1	3	15

Partes	Subsistemas	Posible efecto	Falla potencial	Causa	Severidad	Ocurrencia	Detección	RPN	Control de prevención Control de detección Acciones	Severidad	Ocurrencia	Detección	RPN
Control	Software de obtención de datos	Mediciones tomadas erróneamente. Cuelgue del programa principal.	Bugs en la programación del relevamiento de datos.	Mala programación de la adquisición de datos	5	3	4	60	Revisión del código por varios programadores. Ensayos extensivos de funcionamiento previo a la venta. Período de prueba del prototipo en un colector propio.	5	1	3	15
Comunicaciones	Placa GPRS	Daño severo a la placa GPRS, corte en las comunicaciones con el servidor remoto.	Sobrecorriente a la entrada	Sobrepico de corriente por falta de desacople.	5	3	5	75	Utilizar capacitores de desacople. Ensayos de Sobrecorriente sobre la placa GPRS	5	1	5	25
		No se actualizan las mediciones en el servidor remoto	El módulo no tiene acceso a internet.	No hay señal GPRS en el lugar de colocación del sistema.	4	3	3	36	En el manual de usuario aclarar que la necesidad de tener señal GPRS en el lugar del colector. Marcar con un LED la presencia de señal del módulo GPRS. Utilizar una memoria SD interna que releve los datos.	4	1	3	12
	Software de manejo de comunicaciones	Pérdida de algunas mediciones enviadas.	Error en la comunicación.	Mala programación, casos especiales no tenidos en cuenta.	4	3	3	36	Pruebas extensivas de funcionamiento previo a la venta.	4	2	3	24
		No se pueden acceder mediciones de cierta antigüedad.	Pérdida de mediciones.	El servidor remoto desecha datos antiguos.	3	3	5	45	Revisar contrato con el servidor remoto, asegurar la durabilidad de los datos subidos.	3	1	5	15

Partes	Subsistemas	Posible efecto	Falla potencial	Causa	Severidad	Ocurrencia	Detección	RPN	Control de prevención Control de detección Acciones	Severidad	Ocurrencia	Detección	RPN
Comunicaciones		No se actualizan los datos en el servidor remoto.	Pérdida del acceso al servidor remoto.	Cambia la contraseña de acceso al servidor	4	4	4	64	Proveer la posibilidad de modificar la contraseña del servidor mediante un SMS.	4	1	4	16
Seguridad	Sobretensión de la alimentación de línea.	No se actualizan datos en el servidor remoto.	Daño irreversible a la placa GPRS.	Rayos. Sobretensión en la red eléctrica domiciliaria.	4	2	5	40	Agregar fusibles y varistores en la entrada del circuito. Rodear al equipo con una jaula de Faraday.	4	1	5	20
		Se dejan de relevar datos.	Sobretensión en la entrada del microcontrolador.	Rayos. Sobretensión en la red eléctrica domiciliaria.	5	2	5	50	Agregar fusibles y varistores en la entrada del circuito. Rodear al equipo con una jaula de Faraday.	5	1	5	25
	Exposición a la intemperie	Mal funcionamiento, se dejan de tomar mediciones.	Circuitería interna dañada.	Degradación del gabinete por excesiva humedad y baja temperatura.	5	3	3	45	Utilizar un material adherido al circuito impreso que proteja el equipo ante la condensación de gotas de agua.	5	1	3	15
		Mal funcionamiento, se dejan de tomar mediciones.	Circuitería interna dañada.	Mala instalación del gabinete. Sobreexposición al sol.	5	3	5	75	Explicitar el lugar de colocación del gabinete en el manual de usuario. El gabinete debe situarse debajo del colector solar, tal que reciba la menor cantidad posible de radiación directa del sol.	5	1	4	20

Partes	Subsistemas	Posible efecto	Falla potencial	Causa	Severidad	Ocurrencia	Detección	RPN	Control de detección	Severidad	Ocurrencia	Detección	RPN
Seguridad	Exposición a la intemperie	Mal funcionamiento, se dejan de tomar mediciones.	Circuitería interna dañada	Temperaturas muy elevadas.	5	2	5	50	Registrar la temperatura de operación del sistema y apagar el equipo en caso de superar el límite de seguridad de 50°C.	5	1	4	20
	Peligros de instalación y mantenimiento.	Lesiones en el instalador.	Descargas eléctricas al instalador.	Conector a la red eléctrica no seguro.	5	2	5	50	Utilizar conectores polarizados que sigan la normativa de seguridad eléctrica nacional. Colocar instrucciones claras en manual de usuario, advertir peligros.	5	1	4	20
		Daños severos a los sensores, malfuncionamiento de ellos.	Error de instalación de sensores	Mala conexión por parte del instalador. Conexión de la polaridad al revés.	5	4	3	60	Colocar instrucciones claras de instalación en el manual de usuario. Exigir la utilización de un buscapolos para instalar el amperímetro. Proveer conectores con etiquetas y/o códigos de colores.	5	1	3	15
Interfaz con el usuario	Software para usuario.	Dejan de almacenarse nuevas mediciones en la memoria interna.	Se llena la capacidad de la memoria interna.	La capacidad de la memoria no es suficiente para almacenar los datos durante 2 años.	5	3	3	45	Incluir en el producto una tarjeta de memoria que tenga espacio para almacenar mediciones durante más de 2 años.	5	1	3	15

Partes	Subsistemas	Posible efecto	Falla potencial	Causa	Severidad	Ocurrencia	Detección	RPN	Control de detección	Severidad	Ocurrencia	Detección	RPN
Interfaz con el usuario	Software para usuario.	El usuario no puede observar las mediciones en tiempo real.	La página web del servidor remoto no es accesible.	La página web del servidor remoto pierde acceso a Internet.	4	4	2	32	Utilizar un servidor remoto provisto por un tercero que no presente caídas frecuentes.	4	2	2	16
		El sistema no toma mediciones.	Se desconecta la alimentación del microcontrolador dentro gabinete.	El gabinete recibe un golpe no intencional.	4	3	3	36	Fijar las placas dentro del gabinete. Utilizar conectores con trabas de seguridad.	4	2	3	24
		Las mediciones de uno o más sensores son tomadas con valores erróneos.	Se desconecta uno o más sensores.	Los cables de los sensores sufren un tirón.	4	3	3	36	Utilizar una placa conectora que pueda absorber los esfuerzos mecánicos exteriores que sufren los cables de los sensores del sistema.	4	1	3	12

6.2. Factibilidad de tiempos

El desarrollo del proyecto estuvo dividido en una etapa de inicio y dos etapas de desarrollo. En septiembre del año 2015, surgió la propuesta del ITBA y FOVISEE, se realizó un estudio del proyecto y se dividieron las tareas entre las distintas ingenierías involucradas. La primera etapa de desarrollo, entre enero y febrero de 2016, consistió en el diseño y armado del prototipo para cumplir con una entrega parcial al Banco Santander en abril de 2016. La segunda etapa de desarrollo, entre abril y noviembre de 2016, estuvo dedicada a mejorar el prototipo diseñado, realizar las pruebas y validaciones del mismo y escribir la documentación del proyecto.

6.2.1. Planificación

En el cuadro 1 se muestra la planificación de las tareas del proyecto. En dicha planificación se consideró que la redacción del informe comenzó de manera temprana. El diagrama de Gant y las redes PERT fallan en mostrar la relación de dependencia de la redacción del informe, ya que ésta depende de todas las etapas de desarrollo del proyecto, ocurre en paralelo al resto de las tareas y se extiende más allá de ellas. No obstante, el informe incluirá todo dato relevante que se haya obtenido en el desarrollo del proyecto, aún cuando finalice su duración en el diagrama.

Tarea	Inicio	Fin	Duración	T_{opt}	T_{est}	T_{pes}	µ	σ	Tareas que la preceden
Estudio del proyecto	28-09-15	01-11-15	34,16	15	35	50	34,17	5,83	Ninguna
Análisis de requerimientos	01-11-15	17-11-15	16,66	10	15	30	16,67	3,33	Estudio del proyecto
Encargo de componentes	17-11-15	23-11-15	5,16	1	5	10	5,17	1,50	Análisis de requerimientos
Arribo de componentes	23-11-15	06-12-15	13,83	7	14	20	13,83	2,17	Encargo de componentes
Testeo de microcontroladores	06-12-15	31-12-15	25	10	25	40	25,00	5,00	Arribo de componentes
Testeo de sensores	06-12-15	27-12-15	20,66	10	21	30	20,67	3,33	Arribo de componentes
Testeo de módulo de comunicación	06-12-15	27-12-15	20,67	10	21	30	20,67	3,33	Arribo de componentes
Análisis eléctrico de cada bloque	27-12-15	15-01-16	18,66	7	20	25	18,67	3,00	Testeo de sensores
Diseño de interfaces eléctricas	15-01-16	13-02-16	29	14	30	40	29,00	4,33	Análisis eléctrico de cada bloque. Testeo de módulo de comunicación
Programación de drivers	27-12-15	07-02-16	41,5	21	42	60	41,50	6,50	Testeo de microcontroladores
Diseño de sistema de encapsulado	15-01-16	25-02-16	41,5	21	42	60	41,50	6,50	Diseño de interfaces eléctricas
Diseño de interfaz de monitoreo	07-02-16	13-03-16	35,83	25	35	50	35,83	4,17	Programación de drivers
Diseño de montado	25-02-16	28-03-16	31,66	20	30	50	31,67	5,00	Diseño de sistema de encapsulado
Puesta en marcha del microcontrolador	13-03-16	02-04-16	19,83	7	20	32	19,83	4,17	Diseño de interfaz de monitoreo
Prueba de comunicación	02-04-16	17-04-16	14,5	7	15	20	14,50	2,17	Puesta en marcha del microcontrolador
Prueba de sensado	02-04-16	19-04-16	16,83	7	18	22	16,83	2,50	Puesta en marcha del microcontrolador
Prueba de sensado y comunicación	19-04-16	04-05-16	14,5	7	15	20	14,50	2,17	Prueba de comunicación. Prueba de sensado
Prueba de funcionamiento continuo	04-05-16	18-05-16	14,5	7	15	20	14,50	2,17	Prueba de sensado. Prueba de sensado y comunicación
Montado	18-05-16	16-08-16	90	60	90	120	90,00	10,00	Prueba de funcionamiento continuo. Prueba de comunicación
Relevamiento de datos	16-08-16	11-11-16	86,66	60	90	100	86,67	6,67	Montado
Documentado de funcionamiento	16-08-16	15-09-16	30	20	30	40	30,00	3,33	Relevamiento de datos
Ajustes y calibraciones	11-11-16	28-11-16	17,16	7	14	40	17,17	5,50	Relevamiento de datos
Redacción de informe	27-12-15	24-10-16	301,66	250	300	360	301,67	18,33	Testeo de microcontroladores
Presentación Aula Magna	28-11-16		0	0	0	0	0,00	0,00	Redacción de informe. Documentado de funcionamiento. Ajustes y calibraciones

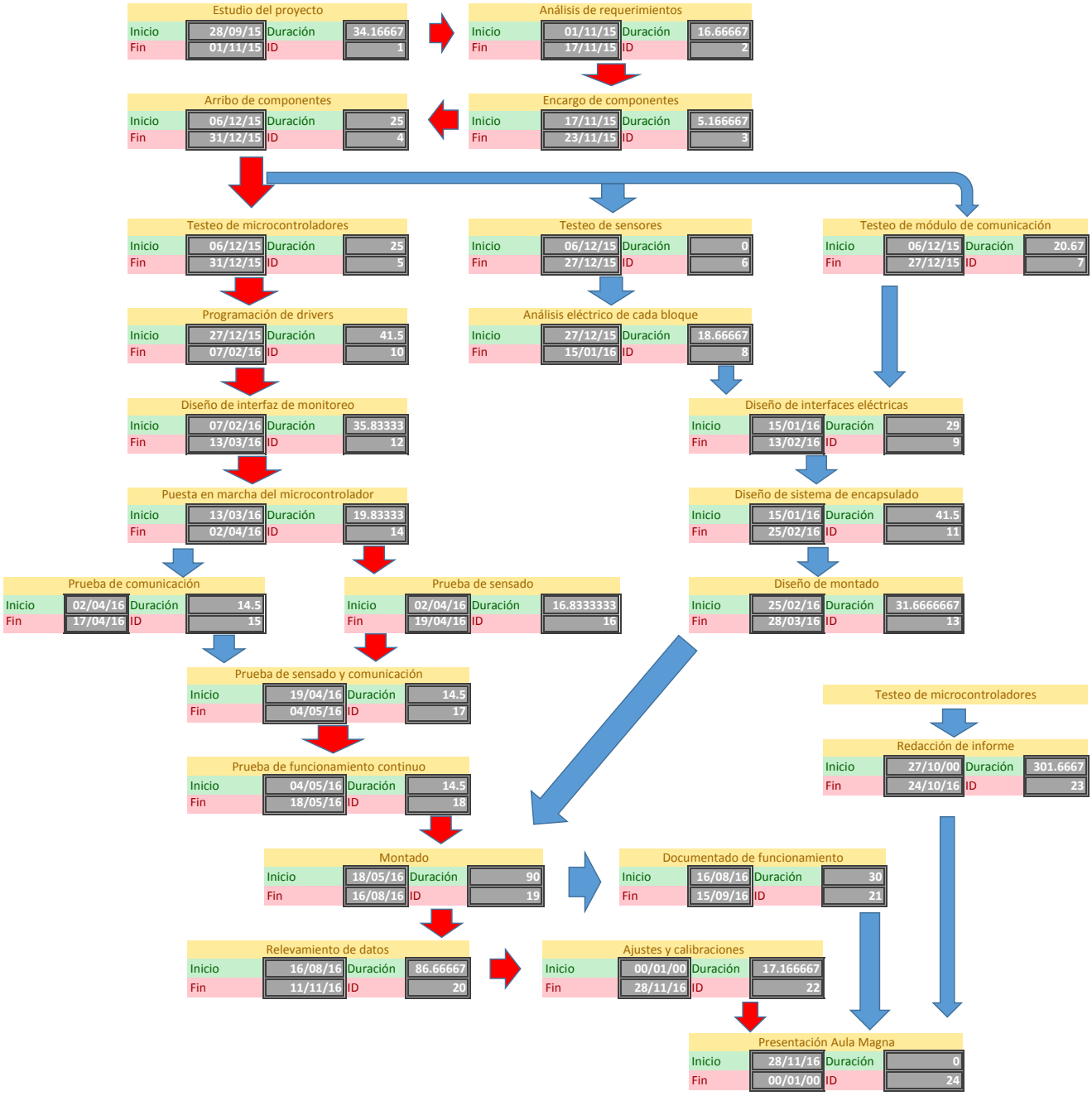
Cuadro 1: Diagrama de Tiempos

Para más detalle se explica cada actividad:

- Estudio del proyecto: Durante esta etapa se determinó qué actividades era necesarias para poder llevar a cabo el proyecto, cuáles eran las prioridades de estas y qué requería el proyecto para poder satisfacer al cliente.
- Análisis de requerimientos: Se llevó a cabo un estudio sobre los requerimientos del cliente, y en base a éstos se determinó qué componentes iban a necesitarse.
- Encargo de componentes: Se hizo una busqueda de proveedores para poder determinar un presupuesto, para luego realizar los encargos de materias primas necesarias.
- Arribo de Componentes: Si bien ésta no es una actividad, se la agrega para denotar un tiempo muerto en el cual se espera el arribo de los recursos para comenzar a trabajar.

- Testeo general: Una vez que se obtuvieron los componentes encargados, se realizó un testeo de general de cada uno para corroborar su funcionamiento.
- Testeo de microcontroladores: En esta actividad se tomaron las placas de desarrollo y se las dejó aptas para comenzar a trabajar, es decir se agregó el bootloader, pines macho para conexionado, y se instaló la interfaz de desarrollo del microcontrolador para comenzar a prototipar.
- Testeo de sensores: El testeo de los sensores se llevó a cabo para verificar si su funcionamiento era el esperado y especificado, qué tipo de manejo requería por parte del microcontrolador, y un estudio acerca de cómo debían implementarse los drivers.
- Testeo de sistema de comunicación: El sistema de comunicación requiere una tarjeta SIM, por lo que se corroboró mediante el comando apropiado que acepte la tarjeta.
- Análisis eléctrico de cada bloque: Cada bloque posee una interfaz de comunicación, ya sea digital (con un protocolo propio o estándar) o análogica. Además, cada bloque requiere un nivel lógico para funcionar y un nivel de tensión para ser alimentado.
- Diseño de interfaces eléctricas: El sistema debe manejar diversos niveles de tensión y protocolos, por lo que fue necesario compatibilizar todos los componentes.
- Programación de drivers: Durante esta actividad se programaron todos los drivers que permiten el manejo de los perifericos a ser utilizados por el microcontrolador.
- Diseño del sistema de encapsulado: El sistema de encapsulado encuadró todo lo relacionado a la protección del sistema en la intemperie.
- Diseño de montado: La disposición dentro del encapsulado, las conexiones con el exterior del encapsulado y placas de interfaz con el microcontrolador.
- Diseño de interfaz de monitoreo: Durante esta actividad se encontró un sistema que ya cumplía todos los requisitos que eran necesarios, por lo que la actividad se redujo a configurar este sistema en base a las necesidades del producto.
- Puesta en marcha del microcontrolador: Una vez que se conectaron todos los sensores, se analizaron las interfaces eléctricas y se programaron los drivers, fue posible comenzar a armar el sistema que maneja la recolección de datos, el modo de bajo consumo y las operaciones necesarias para hacer llegar los datos al cliente.
- Pruebas de comunicación y sensado: Una vez que el microcontrolador ha sido puesto en marcha se comprueba que funcione en conjunto con los modulos, sensando y comunicando los datos pedidos.
- Montado: El sistema fue instalado en el sitio de prueba.
- Relevamiento de datos: Durante esta actividad se realizó todo el relevamiento de datos necesario para la fase de pruebas.
- Documentado, ajustes y calibraciones: Luego de obtener mediciones fue necesario documentar lo medido para corroborar los requerimientos y, de ser necesario, realizar ajustes y calibraciones del sistema.

El diagrama de redes PERT se muestra a continuación:


Estudio del proyecto
Inicio 28/09/15 Duración 34.16667
Fin 01/11/15 ID 1
Análisis de requerimientos
Inicio 01/11/15 Duración 16.66667
Fin 17/11/15 ID 2
Encargo de componentes
Inicio 17/11/15 Duración 5.166667
Fin 23/11/15 ID 3
Arribo de componentes
Inicio 06/12/15 Duración 25
Fin 31/12/15 ID 4
Testeo de microcontroladores
Inicio 06/12/15 Duración 25
Fin 31/12/15 ID 5
Testeo de sensores
Inicio 06/12/15 Duración 0
Fin 27/12/15 ID 6
Testeo de módulo de comunicación
Inicio 06/12/15 Duración 20.67
Fin 27/12/15 ID 7
Programación de drivers
Inicio 27/12/15 Duración 41.5
Fin 07/02/16 ID 10
Análisis eléctrico de cada bloque
Inicio 27/12/15 Duración 18.66667
Fin 15/01/16 ID 8
Diseño de interfaz de monitoreo
Inicio 07/02/16 Duración 35.83333
Fin 13/03/16 ID 12
Diseño de interfaces eléctricas
Inicio 15/01/16 Duración 29
Fin 13/02/16 ID 9
Puesta en marcha del microcontrolador
Inicio 13/03/16 Duración 19.83333
Fin 02/04/16 ID 14
Diseño de sistema de encapsulado
Inicio 15/01/16 Duración 41.5
Fin 25/02/16 ID 11
Prueba de comunicación
Inicio 02/04/16 Duración 14.5
Fin 17/04/16 ID 15
Prueba de sensado
Inicio 02/04/16 Duración 16.8333333
Fin 19/04/16 ID 16
Diseño de montado
Inicio 25/02/16 Duración 31.6666667
Fin 28/03/16 ID 13
Prueba de sensado y comunicación
Inicio 19/04/16 Duración 14.5
Fin 04/05/16 ID 17
Testeo de microcontroladores
Redacción de informe
Inicio 27/10/00 Duración 301.6667
Fin 24/10/16 ID 23
Prueba de funcionamiento continuo
Inicio 04/05/16 Duración 14.5
Fin 18/05/16 ID 18
Montado
Inicio 18/05/16 Duración 90
Fin 16/08/16 ID 19
Documentado de funcionamiento
Inicio 16/08/16 Duración 30
Fin 15/09/16 ID 21
Relevamiento de datos
Inicio 16/08/16 Duración 86.66667
Fin 11/11/16 ID 20
Ajustes y calibraciones
Inicio 00/01/00 Duración 17.166667
Fin 28/11/16 ID 22
Presentación Aula Magna
Inicio 28/11/16 Duración 0
Fin 00/01/00 ID 24

Como puede observarse en las redes PERT, el camino crítico depende en gran parte del microcontrolador, las pruebas de los módulos y el montado. También se observa que la redacción del informe debería acompañar el proyecto hasta el finalización del mismo, es decir la presentación en el Aula Magna.

6.2.2. Programación

La figura 7 muestra el diagrama de Gantt del proyecto. Se puede visualizar qué actividades dan huelgo y permiten un margen para su cumplimiento, a la hora de paralelizar el trabajo entre integrantes del grupo y establecer prioridades. Se le debe dar especial atención a las actividades que conforman el camino crítico para poder entregar el proyecto a fines del año 2016.

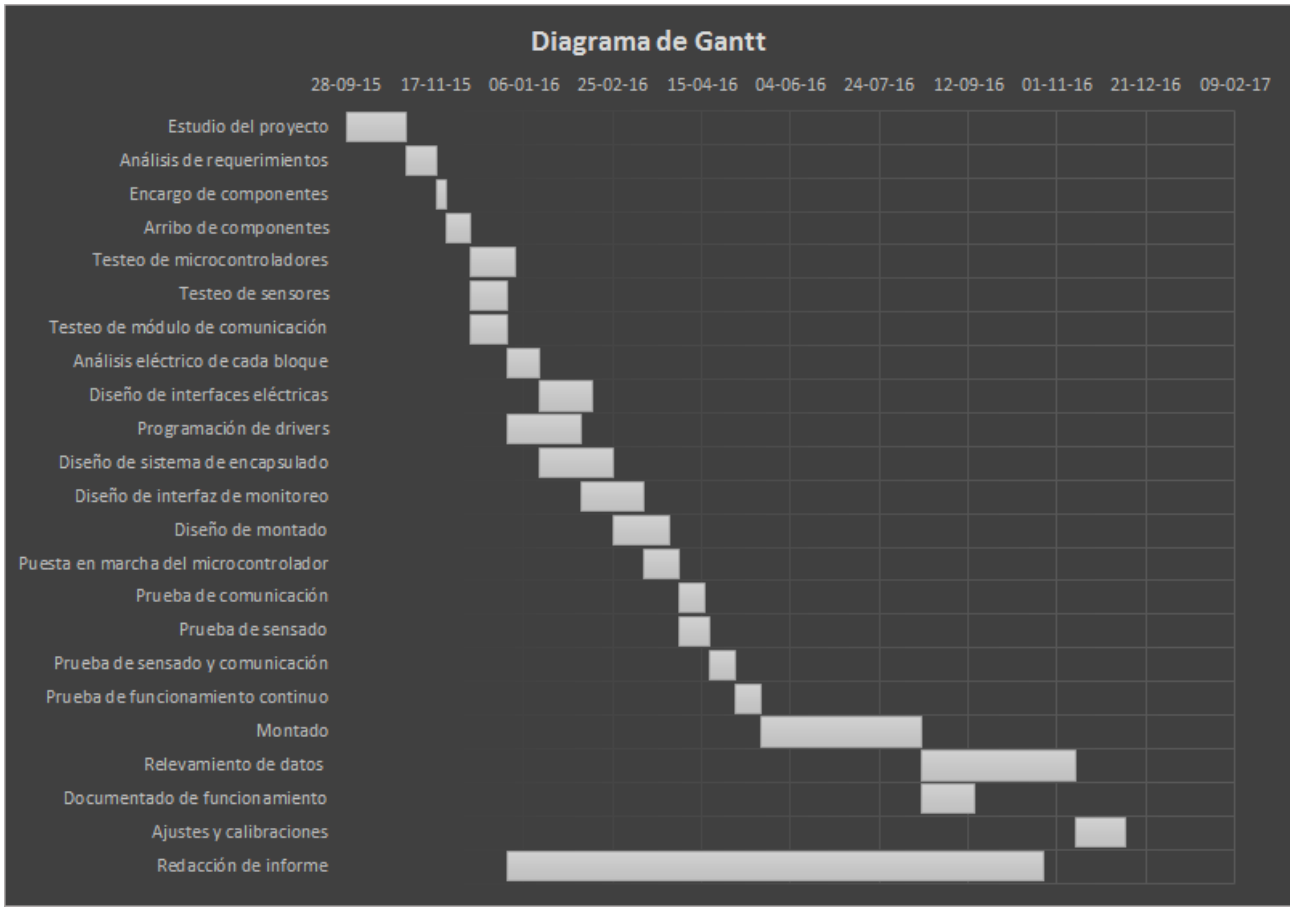


Figura 7: Diagrama de Gantt

La programación muestra como distribuir el trabajo en caso de solapamiento de actividades, hay un caso donde se solapan cuatro actividades, esto se debe al hecho que el informe esta una actividad que conlleva documentado. Por lo que las actividades realizadas siempre son llevadas al informe de forma contemporanea como documentado.

6.3. Factibilidad económica

En esta sección se realiza un análisis económico del proyecto. Es importante destacar que este sistema de adquisición de datos no posee fines de lucro, ya que se encuentra bajo un proyecto de caracter social impulsado por el ITBA y FOVISEE. El proyecto recibió del banco Santander, un presupuesto inicial de 8000 pesos argentinos para desarrollar el prototipo.

Mercado El proyecto está orientado a un mercado acotado, que se resume a instalar 10 sistemas de adquisición de datos en los colectores solares del barrio de La Perla, Moreno. Los costos del proyecto se pueden dividir en el desarrollo y construcción del prototipo, y luego la fabricación de 10 unidades de producto final para instalar en Moreno. Si el producto obtiene resultados positivos en esa instalación, el proyecto podría extenderse a otros barrios carenciados de la Argentina, siempre que sus inversores (el banco Santander Río, el ITBA y FOVISEE) así lo dispongan.

Costos del prototipo En el cuadro 2, se muestran los costos directos e indirectos del prototipo, siendo los costos directos aquellos asociados con la producción de una unidad y los costos indirectos aquellos asociados al desarrollo del prototipo. No se consideraron los costos de mano de obra e instalaciones, ya que el prototipo fue realizado en su totalidad en el ITBA, y los alumnos no esperaban una retribución económica a cambio de su trabajo. Todos los precios que se encuentran en la tabla que no son de origen nacional y cotizan en dólares estadounidenses, fueron convertidos a su equivalente en peso argentino, considerando que 15 pesos equivalen a 1 dólar y se indican con un asterísco (*). El costo total del prototipo es de 3730 pesos, menor a la mitad de los 8000 pesos de presupuesto para el desarrollo del prototipo. Este excedente puede ser utilizado para la fabricación de la primera tira de producto final.

Costos		Precio Unitario [Pesos]	Cantidad	Precio total [Pesos]
Directos				
	Kinetis Freedom KL25z Evaluation Board*	350	1	350
	Level Shifter*	100	1	100
	Prensa-cables y tapones	10	5	50
	Gabinete	121	1	121
	Lector de tarjeta de memoria*	55	1	55
	Cables	21	1	21
	Sensor de temperatura*	100	2	200
	Caudalímetro	699	1	699
	Anemómetro*	700	1	700
	Sensor de radiación*	100	1	100
	Placa EFcomProV10*	450	1	450
	Fuente de alimentación de 9V	300	1	300
	Fuente de alimentación de 5V y 3V3*	55	1	55
	Componentes varios + repuestos*	500	1	500
	Chip Claro	50	1	50
	Memoria SD 8GBy	79	1	79
Indirectos				
	Cable Mini-USB*	150	1	150
	Ácido para placa	99	1	99
Total General				3730

Cuadro 2: Costos

Costos del producto final Los costos aproximados del producto final se detallan en la tabla 4. La mayoría de los costos del producto final son inferiores a los del prototipo, debido a la sustitución de las placas de evaluación por los componentes que las integran. Además, el factor de escala de 10 unidades también reduce el costo de cada unidad. Los precios de valor internacional fueron tomados del distribuidor *Digikey*, y los precios de fabricación e importación de PCB (con la colocación de componentes incluida) se tomaron del fabricante de PCB chino, *PCBway*. Sin embargo, en este caso es necesario considerar los costos asociados al armado, verificación e instalación del sistema en los colectores solares de Moreno, que en el desarrollo del prototipo fueron realizados por los alumnos de forma gratuita. Dicha tarea puede realizarla un técnico electrónico contratado en la condición de personal temporario, mediante una agencia como *Manpower*. La tabla 3 muestra una estimación del costo total de mano de obra para instalar 10 equipos de medición. El costo estimado para el producto final es de aproximadamente los 4571 pesos, equivalente a 305 dólares, que se encuentra por debajo del límite de 400 dólares requerido por la especificación 14. El costo del producto final supera el costo del prototipo debido a la necesidad de contratar un técnico para realizar las tareas de armado del gabinete, verificación del sistema e instalación. Este costo podría reducirse si hubiese una demanda mayor de productos, y de este modo el técnico pudiera ser contratado en condición de relación de dependencia.

Tarea	Horas necesarias
Start-up y entrenamiento	8:00 (1 día)
Armado del gabinete	entre 1:00 y 1:30
Verificación del producto	entre 1:30 y 2:00
Instalación	entre 3:00 y 4:00
Total promedio (1 producto)	6:75
Total promedio (10 producto)	75:30 hs ≅10 días

(a) Horas hombre necesarias para el armado, verificación e instalación

Costo en mano	$18000 \frac{pesos}{mes}$
Factor personal temporario	$1{,}9 * 18000 = 34200 \frac{pesos}{mes}$
Semanas necesarias	$2\ semanas$
Total	$17100 pesos$

(b) Costos asociados al armado, verificación e instalación de 10 equipos.

Cuadro 3: Cálculo de costos de armado, verificación e instalación.

La mayor proporción del costo de los insumos se atribuye al anemómetro y al caudalímetro, cuyos distribuidores no ofrecen descuentos por comprar varias unidades. Para una producción mayor a 10 unidades sería necesario negociar con los distribuidores una reducción del precio unitario, o cambiar el caudalímetro y el anemómetro por otros modelos de funcionalidad similar que ofrezcan descuentos por compras mayoristas.

Costos		Precio por unidad si se compran 10 unidades [Pesos]	Cantidad	Precio total [Pesos]
Directos				
	Microcontrolador MKL25Z*	85	1	85
	Gabinete	121	1	121
	Prensa-cables y tapones	10	5	50
	Cables	21	1	21
	Lector de tarjeta de memoria*	55	1	55
	Sensor de temperatura*	100	2	200
	Caudalímetro	699	1	699
	Anemómetro*	700	1	700
	Sensor de radiación*	44	1	44
	SIM900*	150	1	150
	Fuente de alimentación de 9V*	193	1	193
	Reguladores lineales de 5V y 3V3*	29	1	29
	Fabricación del PCB y colocación de componentes en China*	240	1	240
	Componentes*	100	1	100
	Chip Claro	50	1	50
	Memoria SD 8GBy	79	1	79
	Mano de obra para armado, verificación e instalación de los equipos	1710	1	1710
Indirectos				
	Envío de PCB más impuestos*	45	1	45
Total General				4571

Cuadro 4: Costos estimados del producto final

Ciclo de vida Desde el punto de vista económico, el ciclo de vida de este producto se estima en 3 años (No implica que el producto va a funcionar durante sólo 3 años, sino que va a satisfacer la demanda durante 3 años).

La tecnología de los sensores y el microcontrolador es relativamente actual y no es probable que sufra grandes modificaciones en los próximos 5 años (3 años del ciclo de vida económico más 2 años de vida útil). El módulo de comunicaciones móviles es el factor limitante del ciclo de vida del producto, ya que es probable que la red GPRS sea reemplazada en los próximos años. Sin embargo, el diseño modular del sistema permite modificar la tecnología del módulo de comunicaciones sin tener que realizar cambios significativas del resto de los módulos. Además, los recursos del microcontrolador (memorias Flash y RAM, pines libres, conversores y módulos de comunicaciones) y la memoria interna se encuentran utilizados en menos del 50 % de su capacidad total, por lo que es posible realizar desarrollos incrementales en el futuro. De esta forma, reemplazando el módulo de comunicaciones se podría extender el ciclo de vida del producto otros 5 años (usando una estimación de la duración promedio de una tecnología de comunicación celular).

El producto pertenece a la categoría de 'Internet of Things', que se encuentra en pleno auge en la actualidad, debido a la reducción de costos de los microcontroladores y la facilidad de acceso a las comunicaciones móviles e Internet. Aunque actualmente no existen empresas de IoT enfocadas en la medición de colectores solares, es posible que en el futuro alguna de ellas decida adaptar su producto para la medición de colectores solares. Esta competencia podría significar una reducción en el ciclo de vida del producto, ya que FOVISEE y el ITBA podrían optar por un producto de otra empresa.

Costos de operación La comunicación GPRS requiere un plan de datos móviles con acceso a Internet. Aunque la elección de la compañía proveedora del plan de datos es realizada por el usuario, a continuación se analiza una estimación del costo de operación del sistema. Si se considera que el envío de cada medición requiere aproximadamente 250 Bytes (ver sección 7.1.2), la confirmación del servidor remoto requiere otros $300 Bytes$ y se envían un máximo de 720 mediciones por día (si se configura en el período mínimo de una medición cada 2 minutos), se obtiene un máximo de $12 MBy$ de datos enviados por mes. Por ejemplo, si se elige la empresa Claro Argentina, el plan de datos más económico ofrece $1 GBy$ de datos por 250 pesos mensuales, y supera con un margen de de $1012 MBy$ el máximo de datos necesarios por mes. Además el plan incluye SMS ilimitados que pueden ser utilizados para mandar los SMS de configuración necesarios. Siguiendo este razonamiento, el costo mínimo de operación del sistema es 250 pesos mensuales. Las empresas Personal y Movistar ofrecen planes similares por 260 pesos mensuales y 250 pesos mensuales respectivamente.

6.4. Factibilidad legal y responsabilidad civil

El diseño debe cumplir las normas IRAM2086, IEC 529 IP55, EN 55022/CIPSR 22 y EN 55024/CIPSR 24. Todas las normas se encuentran adjuntas en el anexo de este documento.

La norma IRAM2086 establece los requisitos que debe cumplir el tomacorrientes que se conecta a la red eléctrica. El diseño debe considerar esta norma para que sea compatible con los conectores estándares de Argentina y respete las leyes del país.

La norma IEC 529 IP55, establece los grados de protección que debe tener el sistema: La sigla '5' indica la protección contra polvo, y la sigla '5' se refiere a la protección contra chorros de agua a baja presión. Estas protecciones son las mínimas necesarias para el correcto funcionamiento del sistema en el techo de una casa. El gabinete del sistema debe evitar las filtraciones de agua de lluvia y el ingreso de polvo y objetos sólidos indeseados.

La norma europea EN 55022/CIPSR 22 se refiere a la máxima emisión electromagnética que puede generar el producto. La norma EN 55024/CIPSR 24 indica la mínima inmunidad que debe tener el sistema ante la recepción de radiación electromagnética externa. Ambas normas aplican a los equipos de tecnología de la información, incluyendo hardware y software de computadoras, teléfonos y otros productos de telecomunicaciones. El sistema diseñado debe cumplir con éstas debido a que debe manejar comunicaciones GPRS y entra dentro del grupo de productos de tecnología de la información.

7. Ingeniería del detalle

En esta sección se realiza el análisis en detalle del diseño, explicando los diferentes módulos y sus componentes. La elección de cada módulo se basa en las especificaciones de diseño fijadas en la sección 5.2.

El esquema principal del sistema se muestra en la figura 8, característico de una aplicación IoT. Los sensores del sistema toman los mediciones de las variables del colector solar, que son comunicados al microprocesador, que guarda las mediciones en una memoria SD interna y también las transmite al módulo de comunicaciones. Luego, el módulo de comunicaciones envía las mediciones al servidor remoto online mediante la red celular GPRS. Las mediciones almacenadas en el servidor son accedidas por una PC o una aplicación de un smartphone, que posea conexión a Internet. Las mediciones se muestran al usuario en tiempo real. En caso de ser necesario, las mediciones de la memoria interna pueden subirse al servidor online mediante un programa de MATLAB que viene incluido con el producto o de forma manual por el usuario.

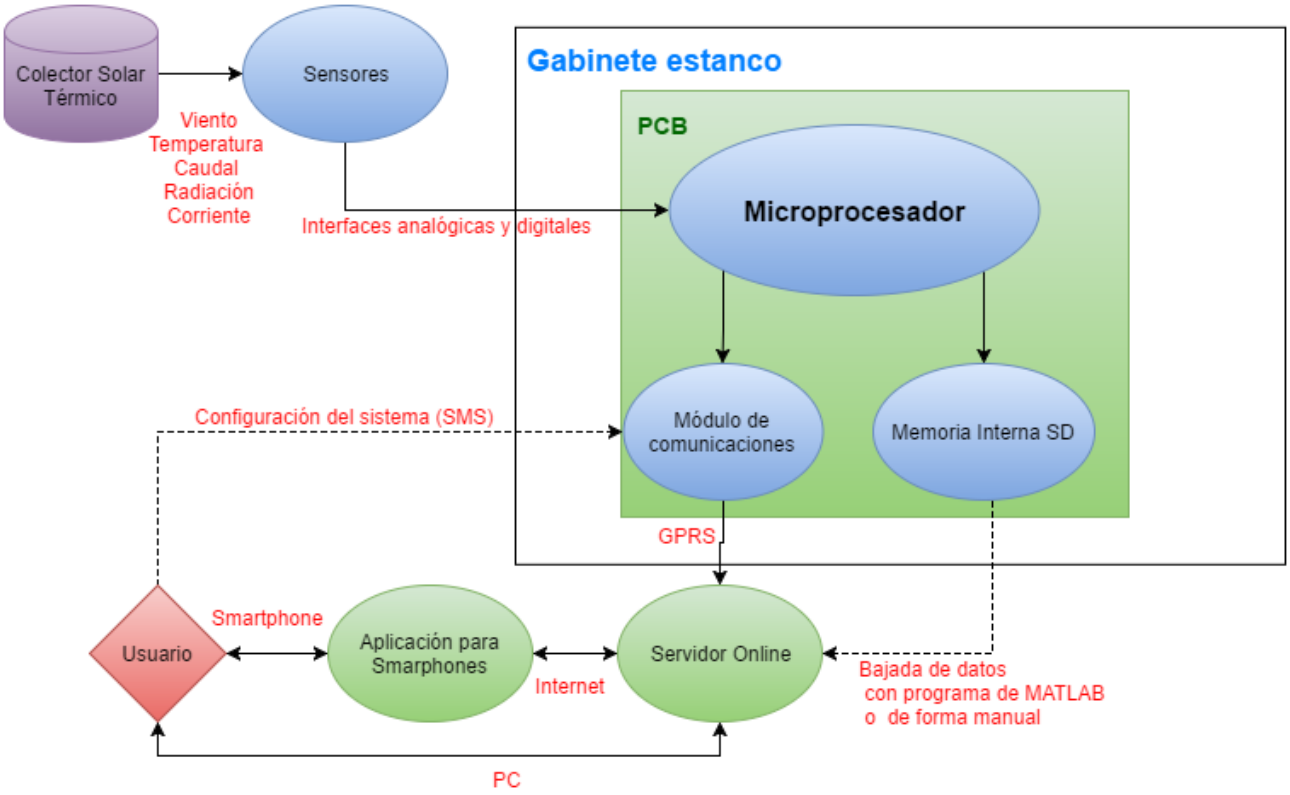


Figura 8: Diagrama de funcionamiento general

7.0.1. Hardware

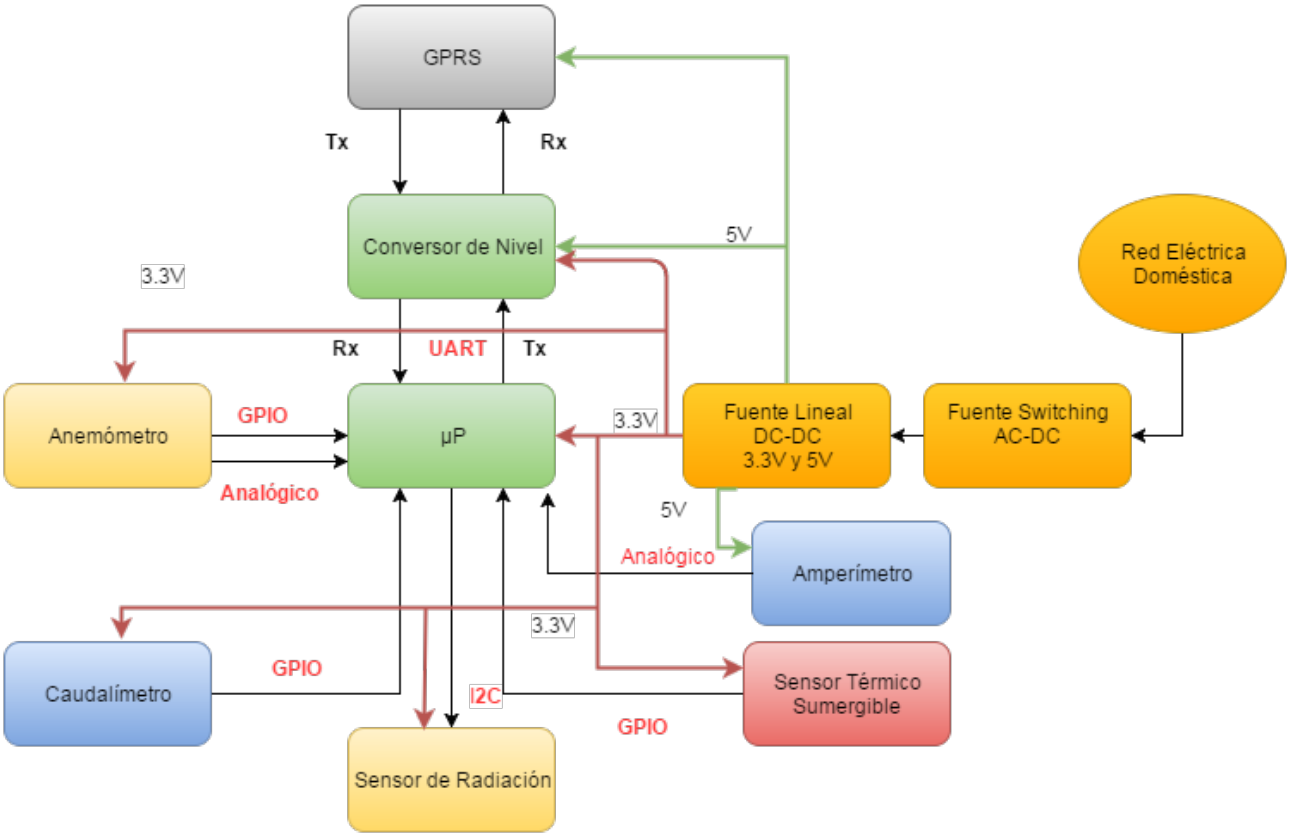


Figura 9: Diagrama de Hardware

Diagrama de bloques

Descripción detallada de cada bloque

- Módulo GPRS

La comunicación con la red de datos móviles es manejada con el IC SIM900 que posee un transreceptor para transmitir en bandas de 850/900/1800/1900 MHz. Este dispositivo trabaja con una tensión de alimentación de $5V$. El módulo es controlado utilizando el protocolo UART con niveles lógicos de $5V$. Además, tiene la opción de permancer apagado cuando no trasmite, que se controla con la señal de control PWR ("Power"). Esta señal de control no requiere $5V$ de nivel lógico, pudiendo ser manejada directamente con los $3{,}3V$ del controlador. El integrado verifica la tensión de entrada, asegurándose que esté dentro del rango permitido, sino apaga el módulo para prevenir daños.

El SIM900 presenta picos de corriente durante la trasmisión de hasta $1{,}5A$, por lo que es necesario agregar un capacitor electrolítico de $4{,}7\mu F$ a la entrada para filtrarlos, no dañar la fuente del sistema y cumplir los requerimientos de picos de potencia. Para poder trasmitir datos por la red celular, el integrado requiere de una tarjeta SIM estándar cuya portadora transmita en las bandas de frecuencia en la cual el SIM900 funciona y también un plan de datos que le permita acceder al servidor remoto.

- Microcontrolador

Se utiliza la familia de microcontroladores de NXP (originalmente Freescale) KL25, especialmente el modelo MCU MKL25Z128VLK4. El microcontrolador se comunica con los distintos módulos utilizando los protocolos I2C, UART y GPIO. Este bloque recibe una tensión de alimentación de $3{,}3V$ y todas sus entradas y salidas poseen ese mismo nivel de tensión.

El microcontrolador se encarga de tomar las mediciones de cada sensor y luego comunicarlas tanto al módulo de comunicaciones y a la memoria interna. Además, el microcontrolador se encarga de apagar el módulo SIM900 y entrar en modo de bajo consumo cuando termina una medición. Este módulo controla período de medición con un Real Time Clock interno que lo interrumpe cuando es necesario realizar una nueva medición. Una descripción detallada del software del microcontrolador se encuentra en la sección 7.1.

- Caudalímetro

El caudalimetro $YF-G1$ posee tres conexiones cableadas, una para la referencia, otra para la alimentación y la última para la señal de interés. La tensión de alimentación es de 3,3V y los niveles lógicos manejados son de ese mismo valor. Por especificación del sensor la señal de interés debe poseer un resistor de pull-up de $10k\Omega$. La tecnología utilizada es de tipo paleta con un sensor que genera una señal cuadrada de frecuencia proporcional al caudal circulante con una sensibilidad de $\frac{1}{420}\frac{litros}{puslo}$. Este sensor tiene una pulgada de diámetro y dos conectores de rosca hembra.

- Sensor de radiación

El sensor BH1750FVI utiliza el protocolo I2C para comunicarse, y maneja tensiones de 3,3V como niveles lógicos y de alimentación. Además, este sensor posee una dirección que puede configurarse en dos alternativas, según como se encuentre conectado el pin de address, que modifica el bit más significativo de la dirección. El software del microcontrolador tiene configurada la dirección del dispositivo con un '0' en la posición más significativa, por lo que el pin de address se encuentra conectado a referencia ($0V$). Además, el sensor mide la radiación en unidades de lux y comunica su valor mediante $I2C$. Dicho valor de radiación es convertido a $\frac{W}{m^2}$ por el microcontrolador, utilizando la constante aproximada de conversión para luz solar $1lux = 4{,}02\frac{W}{m^2}$.

- Sensor de temperatura:

El sensor de temperatura consiste del IC DS18B20 envainado para que sea apto para la inmersión. Este sensor posee tres cables de conexión para referencia, alimentación y señal de interés. La tensión manejada es de 3.3V y posee un protocolo propio por lo que el manejo de éste se realiza mediante el módulo GPIO del microcontrolador. La tecnología es de la categoría de circuito integrado, estos varian un cierto parámetro medible interno, usualmente la tensión de un semiconductor, a partir de la cual derivan un valor de temperatura.

- Sensor de viento

Se utiliza el anemómetro Davis 7911 que posee cuatro cables de conexión: Uno para referencia, otro para alimentación y dos para señales de interés. Las señales de interés son para enviar la dirección y sentido del viento sensados. La dirección se indica analógicamente con un valor de tensión entre $0V$ y 3,3V para valores entre 0 y 360 grados respectivamente. Para la velocidad del viento el sensor utiliza una tecnología similar a la del caudalímetro, que comunica el dato sensado en base a una señal cuadrada de frecuencia proporcional a la velocidad del viento. Las señales requieren un capacitor anti jitter de $10\mu F$ para la velocidad, un resistor de pull-up de 4k7Ω y un capacitor anti rebote de $15nF$ para la dirección.

- Amperímetro

Se utiliza el sensor de efecto hall ACS712, que mide corriente instantánea. Este dispositivo funciona con una tensión de alimentación de $5V$, mide en un rango de $\pm 20A$ y otorga una salida analógica de tensión entre $0V$ y $5V$. El sensor tiene una salida de 2,5V para marcar los $0A$, y una sensibilidad de $100\frac{mV}{A}$.

- Fuente de alimentación

La alimentación del sistema consta de dos partes: La primera parte es una fuente switching con aislación galvánica, que es utilizada para obtener $9V$ de tensión continua a partir de los $220V_{rms}$ de alterna de la red domiciliaria. La segunda fuente recibe los $9V$ y genera tensiones de 3,3V y $5V$ simultaneamente. Esta fuente se encuentra compuesta por dos reguladores lineales independientes: El AMS1117-3.3 para la salida de 3,3V y el AMS1117-5 para la salida de $5V$. La ventaja de tener 2 fuentes independientes es una mejor regulación de carga y una menor variación de cada tensión de alimentación. La elección de estos se dió por rapida disponibilidad y facildiad de implementación, ademas de tener garantizado el funcionamiento del producto por parte del fabricante, ahorrando horas-hombre.

- Conversor de nivel

El conversor de nivel es necesario para adaptar la tensión entre el microcontrolador de 3,3V y el dispositivo SIM900 que se maneja con $5V$. Debido a que las tasas de datos de comunicación serial son bajas (menores a 10 kbps), el conversor de nivel no genera retardos que afecten y comprometan la transmisión de información. El conversor elegido es el BC547, de tecnología MOSFET que funciona de forma bidireccional.

Detalles de selección y cálculo de los elementos circuitales de cada bloque

- Módulo GPRS

El módulo SIM900 es estándar y el más económico de los módulos que permiten comunicación GPRS. Este chip cumple los requerimientos de comunicación para las bandas utilizadas en el país y utiliza el protocolo TCP/IP. Además, presenta la ventaja de ser de bajo consumo y tener un rango de operación entre -30°C y 90°C (presentando un margen de 30°C para calentamiento interno del gabinete respecto de la especificación de diseño). La comunicación se realiza mediante los comandos AT, que son estándar en la industria de las telecomunicaciones. Se comunica utilizando UART, que es compatible con el microcontrolador KL25. Este integrado recibe una alimentación de 5V.

- Microcontrolador

Se eligió el MKL25Z128VLK4 basándose en el bajo consumo de potencia, bajo costo y la prestación que brinda. Este microcontrolador presenta la posibilidad de apagarse casi por completo hasta que suceda una condición de encendido. Su unidad LLWU (Low Leakage Wakeup Unit) permite elegir entre diferentes opciones de bajo consumo, que dan flexibilidad a la hora de disminuir el consumo cuando no se están relevando mediciones. Por otro lado se eligió en base a la cantidad de pines y modos de comunicación que posee, ya que ofrece numerosas variantes de protocolos de comunicación, tanto SPI, como I2C y UART, que son necesarias para poder recibir los datos de los sensores. Posee un módulo ADC de alta resolución (hasta 14 bits), de resolución suficiente para los sensores que transmiten datos analógicos. Además tiene la posibilidad de utilizar un RTC (Real Time Clock) interno para contar los intervalos entre mediciones, que evita la necesidad de un RTC externo. Este microcontolador provee un entorno de programación bien documentado, reciente (2013) y fácil para un rápido desarrollo. El precio del integrado es muy bajo, sale 2.43 USD si se compran más de 100 unidades o 2.65 USD si se compran más de 25 unidades. Su rango de operación es de -40°C a 105°C, que supone un margen de 45°C para que se caliente el interior del gabinete en base a las especificaciones de diseño.

- Caudalímetro

El caudalímetro es capaz de medir en el rango de 0 a 100 $\frac{litros}{min}$ con una precisión de $0.16\frac{litros}{min}$, que cumple con el mínimo especificado de $1\frac{litro}{min}$. Su rango de temperaturas de trabajo va de 0°C a 75°C. Cabe aclarar que es necesario instalarlo en la entrada del colector donde se el agua es fría y no supera el límite de 75°C. Circuitalmente se utilizó la configuración de pull-up del fabricante utilizando un resistor de $10K\Omega$.

- Sensor de radiación

El sensor de radiación es capaz de medir desde 0 a 65535 lx y con resolución ajustable entre 0.5 lx y 4 lx. Para la radiación de origen solar, $4{,}02\frac{W}{m^2}$ son aproximadamente equivalentes a 1 lx, por lo que la precisión del sensor equivale a $16{,}08\frac{W}{m^2}$, cumpliendo con la mínima precisión especificada de $50\frac{W}{m^2}$. Además, este sensor mide en el rango de $[500, 600]$ nm, según la curva de la figura 10, con una máxima diferencia de atenuación de 0.8, por lo que cumple con las especificación de diseño.

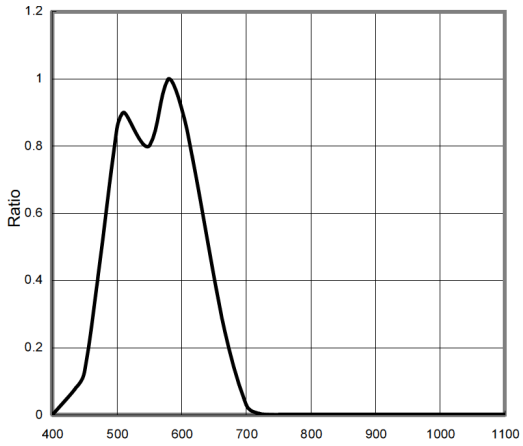


Figura 10: Respuesta Espectral del sensor

- Sensor de temperatura:

El sensor de temperatura DS18B20 presenta un rango de medición entre $-55°C$ y $125°C$. Dentro del rango de $0°C$ a $85°C$, el sensor puede medir con una tolerancia de $0{,}5°C$, mientras que para el resto del rango la precisión es $2°C$. Estas características cumplen con la mínima precisión impuesta por la especificación, de $\pm 0{,}8$°C en el rango de $0°C$ a $85°C$ y 4°C en el rango de 85°C a 100°C.

- Sensor de viento

El anemómetro Davis 7911 posee un rango de medición de $[0; 322]\frac{km}{h}$, un error absoluto de $3{,}21\frac{km}{h}$, que cumple la especificación de la medición de la velocidad del viento con un margen de $1{,}79\frac{km}{hora}$. Además, puede determinar la dirección del viento con un error de hasta 7°, que también cumple con la especificación referida a la medición de la dirección de viento (precisión de 1 octante / 45°).

- Amperímetro

El sensor ACS712 mide la corriente con una precisión de $\pm 1{,}5\,\%$ en el rango de -20 a 20 A, que es equivalente a una precisión mínima de $\pm 0{,}15A = \pm 0{,}106A_{rms}$, lo que cumple la especificación de la medición de corriente con un margen de $0{,}4A_{rms}$. Los calculos fueron hechos suponiendo corriente senoidal.

- Fuente de alimentación

La fuente seleccionada marca $full - energy$ es de tipo switching con aislación galvánica para aislar el resto del circuito y proveer mayor protección. La fuente posee una entrada que acepta tensiones de 100 a 250 V_{rms}, que cumple lo previsto por las especificaciones (de $176V_{rms}$ a $242V_{rms}$) . Posee una salida de hasta $2A$ y $9V$, pudiendo proveer los picos de corriente especificados. Además, la fuente tiene especificado un límite de entrada de $300mA$ de inrush, menor a los $400mA$ especificados.

Plan de pruebas de cada módulo

- Microcontrolador

1. Conectar el microcontrolador a la fuente de alimentación de $3{,}3V$.
2. Verificar que el LED de estado se encienda en color verde. (El microcontrolador de funcionar bien prende el LED con ese color)
3. En un intervalo de tiempo menor a 2 minutos, el LED de estado debe cambiar de color verde a rojo (cuando termina el intervalo de medición y el microcontrolador pasa al modo de bajo consumo).

- Módulo GPRS

1. Alimentar el módulo con $5V$.
2. Colocar una tensión mayor a $3{,}3V$ en el pin de PWR.
3. Verificar que el LED de estado se encienda en color rojo.
4. Enviar el comando 'AT' a través de la interfaz UART del controlador . Este comando verifica que el dispositvo este disponible.
5. Verificar que el sistema responda 'OK'.

- Caudalímetro

1. Alimentar el sensor con $3{,}3V$.
2. Girar la paleta del caudalímetro, introduciendo un flujo de aire a través de la apertura en el sentido que indica la flecha del caudalímetro. Un soplido suave es suficiente para la prueba.
3. Verificar que aparezcan pulsos de $3{,}3V$ de frecuencia mayor a $0{,}1Hz$ y menor a $300Hz$ en el pin de señal del sensor.

- Sensor de radiación

1. Tapar el sensor, tal que no reciba luz.
2. Alimentar el sensor con $3{,}3V$.
3. Colocar el pin de address a $0V$.
4. Enviar los comandos 0x01 y 0x20 a través de la interfaz I2C. El primer comando sirve para configurar la resolución a un lux de precisión y el segundo para tomar una medición.
5. Verificar que el sensor envíe una medición menor a $20lx$.

- Sensor de temperatura

1. Sumergir el sensor en un recipiente con agua a temperatura ambiente con un termometro de mercurio
2. Alimentar el sensor con 3,3V.
3. Realizar una lectura de temperatura.
4. Verificar que el sensor envíe una medición que no difiera en mas de $\pm$0,1°.C

- Sensor de viento

1. Alimentar el anemómetro con 3,3V.
2. Colocar el aspa de dirección a $0° \pm 2°$.
3. Verificar que la tensión del pin de señal de dirección sea menor a 100mV.
4. Girar las copas del anemómetro.
5. Verificar que aparezcan pulsos de 3,3V de frecuencia mayor a 0,1Hz y menor a 500Hz en la señal de velocidad.

- Sensor de corriente

1. Alimentar el sensor con 5V .
2. Cortocircuitar las entradas
3. Verificar que la salida analógica éste en $2{,}5 \pm 0{,}2V$.

- Fuente de alimentación

1. Conectar la fuente a la red eléctrica.
2. Verificar que la salida de la fuente switching tenga 9V con un grado de tolerancia de 5 %.
3. Verificar que la salidas de los reguladores lineales tengan 3,3V y 5V con un grado de tolerancia de 5 %.

- Conversor de nivel

1. Conectar las alimentaciones de 5V y 3,3V.
2. Colocar $5 \pm 0{,}1V$ en el pin de Tx de la terminal de 5V y $3{,}3 \pm 0{,}1V$ en el pin de Tx de la terminal de 3,3V.
3. Verificar que el pin de Rx de la terminal 3,3V tenga una tensión superior a 3,1V y una tensión inferior a 3,5V , y que el pin de Rx de la terminal de 5V tenga una tensión superior a 4,7V e inferior a 5,1V .

7.1. Software

El desarrollo de software del producto corresponde a la programación del microcontrolador KL25Z, que se encarga de tomar los datos de los sensores, subirlos a un servidor y guardarlos en la memoria SD interna. Todo el código está escrito en el lenguaje C++ dentro del entorno de programación de Kinetis Design Studio, un IDE dedicado a microcontroladores Freescale. Se utiliza este entorno debido a que provee una capa de abstracción de hardware (Kinetis Processor Expert) que agiliza el desarollo de drivers específicos. El software se separa en los drivers dedicados a cada sensor, el driver de comunicaciones, el manejo de estado de bajo consumo y el programa principal que integra los anteriores.

El servidor remoto elegido es ThingSpeak, que pertenece a una empresa dedicada a proveer servicio de red a aplicaciones IoT. Esta elección se basa en el costo nulo del servidor, el almacenamiento sin límites de tamaño ni tiempo (ThingSpeak asegura guardar todos los datos que se envíen por tiempo indefinido), y porque es una plataforma pensada para su integración con MATLAB. Esta última característica es importante para que el cliente pueda bajar las mediciones directamente desde MATLAB y realice las estadísticas que considere necesarias. Además, existen varias aplicaciones gratuitas para Android e iOS dedicadas a la lectura de datos del servidor, que reduce los costos y tiempos del diseño de una aplicación propia. De estas posibilidades, se elige la aplicación Thingviewer de Android que es compatible con la versión *Lollipop*(v5,0/v5,1) y posteriores, y permite visualizar todos los datos subidos al servidor de ThingSpeak de manera interactiva, como se muestra en la figura 11.

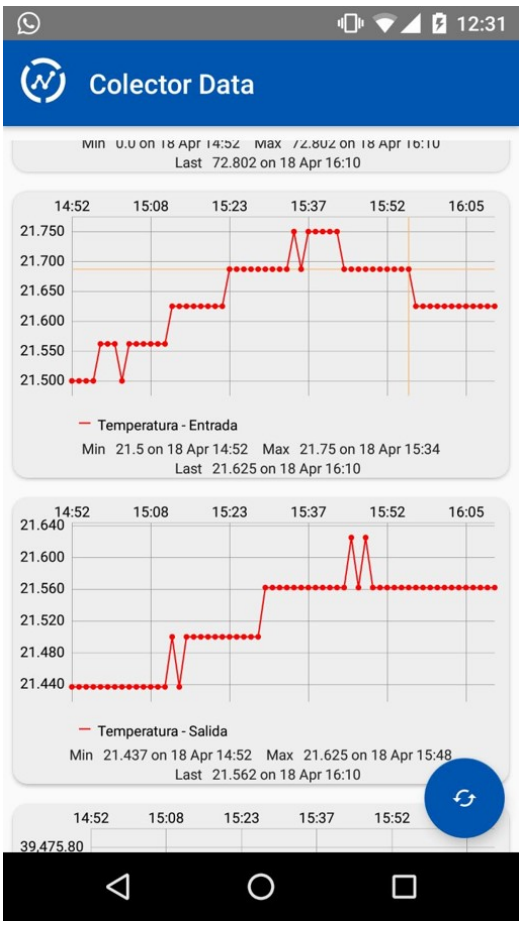


Figura 11: Aplicación para android

El cliente puede configurar tanto la contraseña del servidor remoto (por si decide cambiar de cuenta), como la frecuencia de las mediciones. Para cambiar la contraseña, debe enviar un SMS al número asignado al sistema correspondiente, con la el formato «API KEY: [código de la contraseña que desea modificar]». Para modificar la frecuencia de mediciones, se debe enviar un SMS con el formato «Periodo Mediciones: [frecuencia de mediciones que quiere fijar, en minutos]». Cabe aclarar que esta configuración es sensible a las mayúsculas, por lo que el usuario debe respetar dicho formato. Estas directivas y algunos ejemplos de uso están presentes en el manual de usuario del producto.

En caso de registrar alguna descalibración en las mediciones de uno o varios sensores del sistema, un técnico autorizado debe configurar los sensores enviando un SMS de configuración al sistema, ajustando la ganancia y offset de cada sensor según la ecuación 1, de modo que cumplan las especificaciones. Dicho SMS debe seguir el siguiente formato: «Calibracion [referencia sensor de la tabla 5] Ganancia: [valor de ganancia en ASCII] Offset: [valor de offset en ASCII] Password: [contraseña del sistema]», donde tanto la ganancia y el offset deben ser especificado utilizando hasta 4 decimales y el separador decimal debe ser el caracter punto '.', y la contraseña por default es 'SlrLggr'. Luego de efectuar la calibración, el sistema responde al técnico con el SMS «Calibracion OK». Los valores por default para todos los sensores son $ganancia = 1$ y $offset = 0$. La corrección de la dirección del viento no se aplica directamente sobre el octante, sino a los grados obtenidos del sensor. De esta forma, el técnico puede realizar un ajuste más preciso del sensor.

$$Valor\ Corregido = Valor\ Medido * ganancia + offset \tag{1}$$

Sensor	Referencia
Anemómetro - velocidad del viento	ANV
Anemómetro - dirección del viento	AND
Amperímetro	AM
Caudalímetro	C
Sensor de temperatura de entrada	TIN
Sensor de temperatura de salida	TOUT

Cuadro 5: Referencias para calibrar los sensores

7.1.1. Diagrama de estados y flujogramas

El diagrama de flujo del programa principal se presenta en la figura 12. Cuando se enciende el dispositivo, en primer lugar se inicializan los drivers y luego se entra en el loop principal. A continuación, se entra en el estado de bajo consumo del KL25Z, particularmente el modo LLS (Low Leakage Stop Mode) que es provisto por el módulo LLWU (Low Leakage Wakeup Unit) del KL25Z. En este estado, el consumo de corriente de entrada está limitado a un máximo de $2{,}09\mu A$, siendo $3{,}3V$ la entrada, lo que limita la potencia consumida por el microcontrolador a $7\mu W$ y se reduce el consumo del microcontrolador varios órdenes de magnitud. En este estado todos los módulos del KL25Z se apagan, es decir no reciben clock, menos los módulos de LPTMR (Low Power Timer) y RTC (Real Time Clock). El primero se encarga de verificar la actividad en el sensor de caudal, que al superar un límite prefijado de $1\frac{L}{min}$ se encarga de despertar al microprocesador, sacándolo del estado LLS. El segundo es un reloj que sigue contando a pesar del modo de bajo consumo, y despierta a la KL25Z en caso de haberse completado un intervalo prefijado por el usuario.

Al salir del estado de bajo consumo, ya sea por un aumento del caudal o la finalización de un intervalo de espera, el software toma una marca de tiempo (para almacenar el momento de medición) y procede a tomar las mediciones de los sensores. Debido a que los sensores de caudal y velocidad del viento se basan en un sistema de conteo de pulsos, es necesario que la medición dure al menos 2,25 segundos para lograr la precisión especificada: Para el caudalímetro, se puede calcular la precisión obtenida, estimando el error cometido al omitir o añadir un pulso. Si se añade un pulso, el error es de $1pulso * \frac{1}{7}\frac{litros}{min}\frac{seg}{pulso} * \frac{1}{2{,}25seg} = 0{,}063\frac{litros}{min}$, mientras que si se omite un pulso el error es de $-0{,}063\frac{litros}{min}$. Ambos valores, sumados al error del caudalímetro ($0{,}16\frac{litros}{min}$), alcanzan un error de $0{,}223\frac{litros}{min}$, que se encuentra por debajo del mínimo necesario por especificación($1\frac{litro}{min}$). En el caso de la dirección del viento, según el valor de precisión brindado por la hoja de datos del anemómetro, a un período de 2,25 segundos de muestreo le corresponde una precisión de $2\frac{millas}{hora} = 3{,}21\frac{km}{hora}$ (ver hoja de datos del anemómetro del anexo). El conteo de pulsos se realiza mediante interrupciones, por lo que durante ese intervalo también se realizan el resto de las mediciones. Una vez finalizado ese intervalo, procede a enviar todas las mediciones tomadas al servidor remoto: Si el sistema se encuentra en un proceso de muchas mediciones seguidas (debido a que el caudal se encuentra por encima del límite), el sistema de comunicaciones se mantiene prendido y conectado al servidor remoto para agilizar la subida de datos. En caso contrario, el sistema debe prender al módulo SIM900 y apagarlo una vez terminada la medición para reducir el consumo del mismo. Luego de enviar los datos, se verifica la llegada de SMS nuevos, ya que estos sirven de herramienta de configuración para el usuario, específicamente para configurar la clave del servidor remoto y el lapso entre mediciones. Finalmente se completa el ciclo guardando las mediciones en la tarjeta SD interna. Si el caudal se encuentra por arriba del límite de caudal se repite el ciclo, mientras que si se encuentra por debajo del límite, se vuelve al estado de bajo consumo LLS.

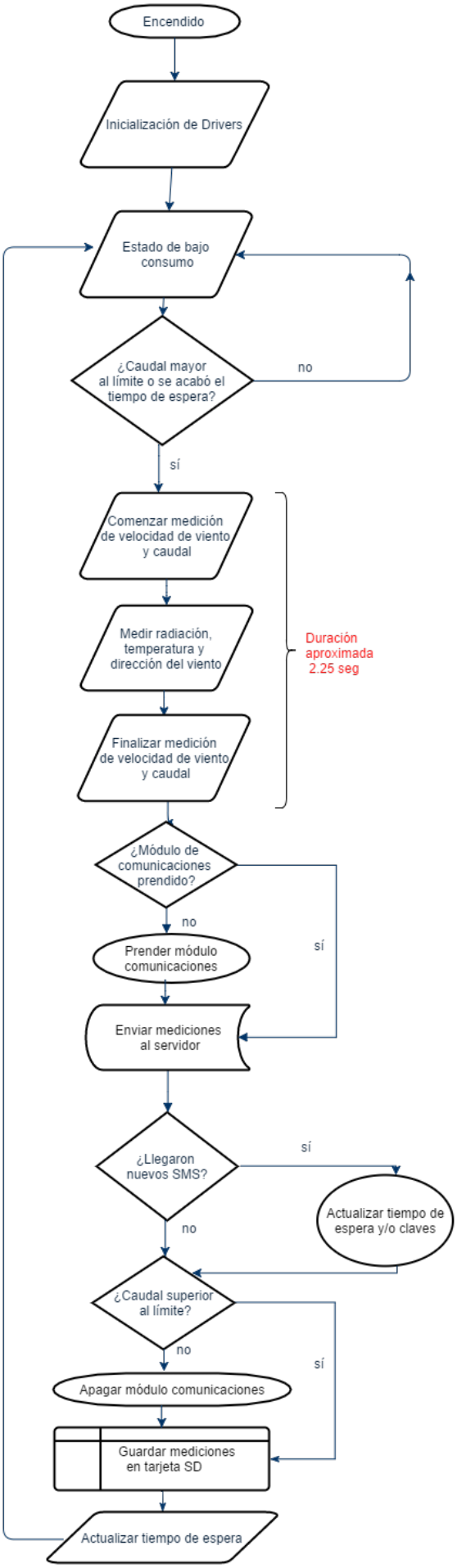


Figura 12: Diagrama de flujo del Software

7.1.2. Descripción de subrutinas

En esta sección se describen las subrutinas más importantes del sistema.

Librería de manejo de consumo Power.h

1. void PowerOffSystem(void,);

   Se encarga de cambiar el modo de consumo del microcontrolador, de Normal a LLS. Deja activadas las interrupciones por Caudal y tiempo. Antes de entrar en LLS, prende un LED rojo para que quede un indicador de que el sistema sigue funcionando correctamente. El sistema activa los módulos LPTMR y RTC para que posteriormente despierten al microcontrolador.

Librería de comunicaciones GSM.h

1. void SendThingSpeak(const char * APIKEY, rtc_datetime_t *fecha, float field1,float field2, float field3, float field4, uint32_t field5, float field6,float field7);

   Esta función recibe 7 mediciones en formato float (desde field1 hasta field7), el momento de medición (rtc_datetime_t fecha) y las envía al servidor ThingSpeak, específicamente a la cuenta que utiliza la clave APIKEY. Para que la función tenga efecto, es necesario que el SIM900 se encuentre prendido. El ordenamiento de las mediciones implementado es el siguiente: field1-Caudal, field2-Temp IN, field3-Temp OUT, field4-Radiacion, field5-VientoDir, field6-VientoVel, field7-Corriente. Igualmente, este orden puede cambiarse fácilmente de ser necesario. La cadena de chars enviada al servidor remoto utiliza 250Bytes de memoria.

2. void ReceiveSMS(uint32_t * Frecuencia, char * APIKEY);

   Si el SIM900 se encuentra prendido, recibe todos los SMS de la cuenta de celular definida, y verifica si algún SMS corresponde a un cambio de configuración del sistema, ya sea para cambiar la frecuencia de las mediciones (en minutos) o la contraseña del servidor ThingSpeak (la APIKEY).

3. void SendSMS(char *text);

   Manda un SMS con el texto text. Sirve para confirmar la recepción de un cambio de configuración.

4. void TurnSIMon(void);

   Se encarga de prender el módulo de comunicaciones, mandando una señal de PWR al SIM900. Si el encendido es efectivo, guarda el valor True en la variable global sim_state.

5. void TurnSIMoff(void);

   Apaga el módulo de comunicaciones, mandando una señal de PWR y modifica el estado de la variable global sim_state a False.

Librería de medición Measure.h

1. void DoWholeMedition(void);

   Función principal del programa, que inicia un ciclo completo de mediciones y posterior envío de las mismas.

2. void SendMeditions(void);

   Función que envía las mediciones realizadas mediante SendThingSpeak() y también las guarda en la memoria interna, mediante SendToSD(...).

3. void SetMeasuringMinutes(uint32_t minutos);

   Esta función modifica la frecuencia de mediciones en minutos.

4. void configureRTCclock(void);

   Función que obtiene el horario actual de un servidor NTP, y configura el RTC del microcontrolador con dicho horario. Esta función se utiliza en el inicio del programa.

Librería de almacenamiento interno SDcard.h

1. void SendToSD(rtc_datetime_t fecha, float field1,float field2, float field3, float field4, float field5, float field6, float field7);

   Con esta función se guardan las mediciones de los 7 sensores y el momento de medición (rtc_datetime_t fecha). Se reciben en formato float y se almacenan en la SD en el archivo *data.log*. Dicho archivo consiste de cadenas ASCII, con el formato "Fecha:[dd/mm/aa hh:mm];Caudal=[valor medido en $\frac{litros}{min}$];TempIn=[valor medido en °C];TempOut=[valor medido en °C];Radiacion=[valor medido en $\frac{W}{m^2}$];VientoDir=[valor medido en $\frac{km}{hora}$];VientoVel=[octante];Corriente=[valor medido en A_{rms}];\n" por cada medición completa.

Librería del anemómetro Viento.h

1. float GetWindSpeed(void);

   Esta función devuelve la velocidad del viento en $\frac{km}{hora}$, una vez que ya transcurrió el intervalo de medición. Si no terminó dicho intervalo, el valor retornado es erróneo.

2. float GetWindDirection(void);

   Se retorna la dirección del viento en float, en grados entre $[0°, 360°]$. El dato se lee analógicamente, utilizando el ADC del microcontrolador.

3. void StartMeasuringWind(void);

   Se inicia el intervalo de medición de la velocidad del viento, activando las lecturas y acumulando los pulsos provenientes del sensor de viento.

4. void StopMeasuringWind(void);

   Una vez transcurridos los 2.25 segundos, se dejan de contar pulsos provenientes del sensor de viento con esta función.

Librería del sensor de radición Radiación.h

1. float GetRadiacion(void);

   Devuelve la radición en unidades de $\frac{W}{m^2}$ en formato float. Se comunica con el sensor de radiación mediante el protocolo I2C, del cual obtiene la luminosidad en lx y la convierte a $\frac{W}{m^2}$ utilizando una tabla de correspondencia entre ambos.

Librería del sensor de temperatura Temperatura.h

1. float LeerTemperatura(uint32_t pin);

   Recibe el pin del microcontrolador al cual esta conectado el sensor de temperatura, y devuelve la temperatura medida en formato float en unidades de °C. La obtiene comunicándose a través de un puerto GPIO en configuración pull up, siguiendo el protocolo establecido por el sensor DS18B20.

Librería del sensor de caudal Caudal.h

1. void StartMeasuringCaudal(void);

   Se inicia el intervalo de medición del caudal, activando las lecturas y acumulando los pulsos provenientes del sensor.

2. void StopMeasuringCaudal(void);

   Una vez transcurridos los 2.25 segundos, se dejan de contar pulsos provenientes del sensor de caudal con esta función.

3. float GetCaudal(void);

   Devuelve el caudal en $\frac{litros}{min}$ en formato float, una vez que ya transcurrió el intervalo de medición. Si no terminó dicho intervalo, el valor retornado es erróneo.

Librería del sensor de corriente Corriente.h

1. float GetCurrent(void);

   Esta función se encarga de leer el valor analógico proveniente del sensor de corriente y devolverlo en *Amperes* en formato float. En la instrucción se realizan mediciones de corriente instantánea durante 500 milisegundos, y se almacenan el valor máximo y mínimo obtenido. Luego, se calcula el valor de corriente RMS como $I_{rms} = \frac{I_{max}+I_{min}}{2\sqrt{2}}$.

7.1.3. Plan de prueba de módulos y de depuración del soft

1. Librería Power.h

   Para validar este módulo, se debe llamar a la función *PowerOffSystem(void)*, y verificar que el LED rojo se prenda y que el consumo de corriente de entrada del microcontrolador disminuya al 10 % o menos. Además se debe probar que el sistema se prenda tanto ante la aparición de pulsos en el pin del módulo LPTMR o la finalización del intervalo fijado por el RTC. Este cambio se verifica si se prende el LED verde.

2. Librería GSM.h

   Debido a su mayor complejidad, este módulo necesita varias pruebas. Primero se debe verificar el funcionamiento de *TurnSIMon(void)* y *TurnSIMoff(void)* para comprabar el correcto encendido y apagado del módulo de comunicaciones. A continuación, se debe probar el envío y recepción de SMS: Llamar a la función *SendSMS(char *text)* para asegurar el funcionamiento de los mensajes de texto. Finalmente, se debe comprobar la conexión a Internet y la subida de datos al servidor remoto, llamando a la función *SendThingSpeak(...),* comprabando la llegada de datos en el servidor remoto.

3. Librería Measure.h

   La librería que maneja los sensores debe ser verificada de a un sensor a la vez, y finalmente con todos al mismo tiempo. Esto significa primero probar las funciones de cada sensor por separado, y una vez comprobado su funcionamiento, llamar a *DoWholeMedition(void)* y *SendMeditions(void)* (primero debe funcionar la librería GSM.h para esta última). Se debe verificar que los datos que arrivan al servidor remoto coincidan, en orden de magnitud, con los medidos de manera independiente por cada sensor.

4. Librería SDcard.h

   La validación debe realizarse comprobando que la función *SendToSD(...)* sube efectivamente los datos relevados a la memoria SD, con el formato especificado en ingenería de detalle. Dicha verificación se puede llevar a cabo sacando la tarjeta SD del sistema para leerla en una computadora.

5. Librería Viento.h

   Se debe comprobar tanto la medición de la velocidad del viento, como la dirección del viento. Para la primera, se debe llamar a la función *StartMeasuringWind(void),* luego comprobar el tiempo de espera dado por *IsMeasuringCaudal(void)*, donde se debe mover la paleta del anemómetro, y finalmente, validar el dato sensado en *GetWindSpeed(void).* Para la segunda, es necesario llamar a *GetWindDirection(void)* y comprobar que el dato relevado se condiga con la dirección del aspa del anemómetro.

6. Librería Radiación.h

   Este módulo se verifica realizando una llamada a *GetRadiacion(void)*, corroborando que el valor recibido sea proporcional a la luz que recibe el sensor.

7. Librería Temperatura.h

   Esta librería tiene dos etapas de prueba: En la primer etapa, se debe verificar el correcto funcionamiento del protocolo de comunicación específico de los sensores DS18B20, para lo que primero se debe resetear el sensor con *ResetTempSensor(uint32_t pin)* y luego mandar la palabra 0xCC (SKIP ROM) con la función *SendWordtoTemp(0xCC,uint32_t pin),* a la que se debe esperar una respuesta de ACK por parte del sensor, mediante una señal activo baja que se recibe con *ReadFromSensor(uint32_t pin).* Luego de verificar dicho funcionamiento, se debe correr la función *LeerTemperatura(uint32_t pin)* y comprobar que el valor numérico recibido en formato flotante coincida con la temperatura ambiente.

8. Librería Caudal.h

   La medición de caudal se prueba de manera similar a la velocidad del viento. Primero se llama a *StartMeasuringCaudal(void),* luego se debe mover el aspa del caudalímetro mientras la función *IsMeasuringCaudal(void)* devuelve *True.* Al final el intervalo de medición, *IsMeasuringCaudal(void)* devuelve *False,* y el dato medido, que se obtiene con *GetCaudal(void)*, debe ser corroborado con los giros del aspa realizados.

8. Construcción del prototipo

8.1. Definición de los módulos

En la figura 13 se presenta el diagrama en bloques del prototipo. En este prototipo se eligieron utilizar placas de evaluación (EB), tanto para el microcontrolador, como el módulo de comunicaciones. El shield, que recibe su nombre por servir de 'escudo' a la placa del microcontrolador, es el PCB principal del sistema, que se encarga de interconectar todos los módulos. A la tensión de línea de $220V_{rmns}$ se conecta un conversor AC/DC switching con gabinete propio, que luego transmite $9V$ de continua hasta el gabinete principal. Dicha tensión es recibida por dos reguladores lineales independientes, de salidas de $3{,}3V$ y $5V$, ambos soldados en el shield. La línea de $3{,}3V$ alimenta tanto al microcontrolador, como a los sensores digitales de caudal, radiación, de velocidad del viento y de temperatura. La línea de $5V$ se encarga de alimentar a la placa de comunicaciones EFcomPro. Todos los sensores se conectan al shield mediante conectores en el gabinete. La memoria SD interna se conecta a una lectora soldada en el shield, y para el módulos de comunicaciones se utiliza la placa aparte, la EFcomPro, que contiene el integrado SIM900. Tanto el conjunto microcontrolador más shield como la placa de comunicaciones se encuentran fijadas al gabinete principal.

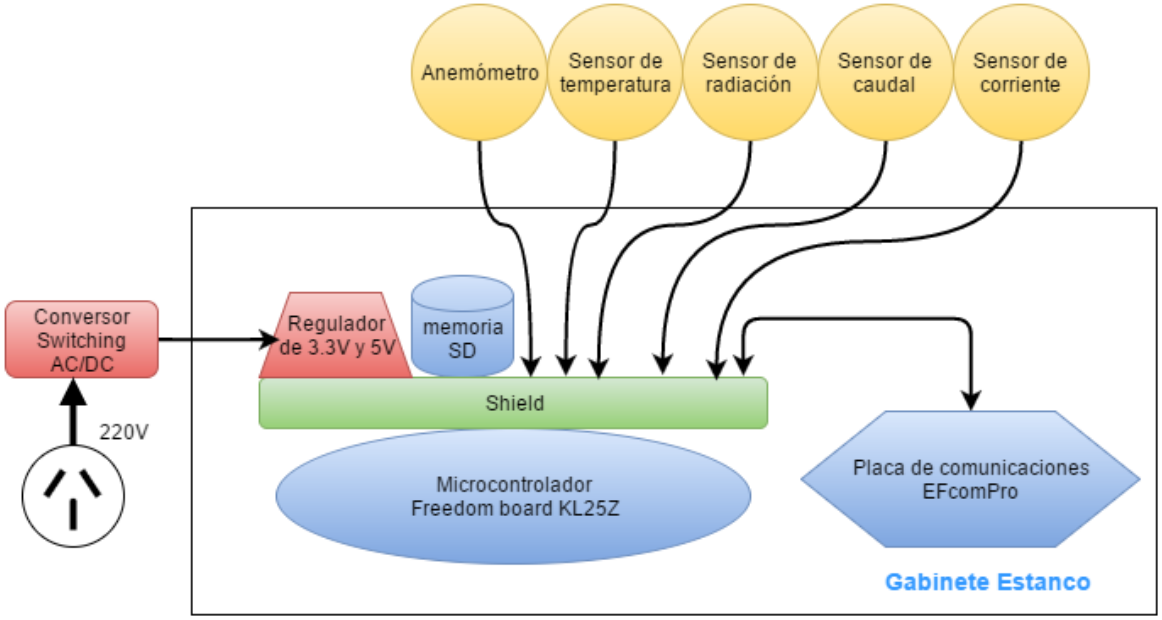


Figura 13: Módulos del prototipo

Microcontrolador

Se eligió la placa de evaluación de NXP KL25Z, perteneciente a la familia de las Freedom Boards, ya que permite acceder a todas los módulos necesarias del microcontrolador elegido en la ingeniería de detalle, a un bajo costo (15 USD). La placa KL25Z posee 80 pines multipropósito, que alcanzan y sobran para el diseño realizado, y su disposición es ideal para realizar placas del tipo shield que se fijan mediante la conexión con dichos pines. Además, esta placa de desarrollo presenta una interfaz de debugging, que facilita y acelera el desarrollo del prototipo. Adicionalmente, cuenta de 3 LEDs (rojo, verde y azul) que sirven como señalizadores del estado del sistema (bajo consumo o realizando mediciones).

Figura 14: Módulo de evaluación KL25Z

Módulo de comunicaciones

La placa EFcomProV10 es una opción de hardware libre que posee el SIM900 y un regulador de tensión para su alimentación. Se eligió esta placa de evaluación, ya que contiene el integrado SIM900 (elegido en la ingeniería de detalle), a bajo costo y porque estaba disponible en el Departamento de Electrónica (no hubo que importarla).

Figura 15: Módulo EFcomProV10

Sensores

Se consiguieron los sensores elegidos en la ingeniería de detalle, de acuerdo a las especificaciones de diseño. Éstos se instalaron de la siguiente manera:

1. Temperatura: Se instalaron 2 sensores DS18B20. Éstos se conectan al microcontrolador en los pines GPIO J1_2 para la temperatura de entrada, y J1_4 para la temperatura de salida.

2. Caudal: Se conectó la señal inteligente del sensor YF-G1 al pin J1_9, correspondiente al módulo LPTMR del microcontrolador.

3. Radiación: El sensor BH1750FVI se conectó al módulo I2C del microcontrolador, mediante 4 pines: SDA (datos digitales) en el pin J10_4, SCK (reloj de la comunicación) en el pin J10_2, GND (tierra del circuito) y VCC (Alimentación de 3.3V).

4. Anemómetro: El sensor de velocidad del viento se conectó con el módulo TPM TIMER del microcontrolador, mediante el pin J10_10. La dirección del viento se transmite de manera analógica, con tensiones entre 0 y 3.3V que son procesadas por el conversor AD del microcontrolador, correspondiente al pin J10_8. Ambos sensores se encuentran integrados en el Davis 7911 SS.

5. Corriente: Un sensor ACS712 que se conectó al módulo del conversor AD del microcontrolador, mediante el pin J10_1.

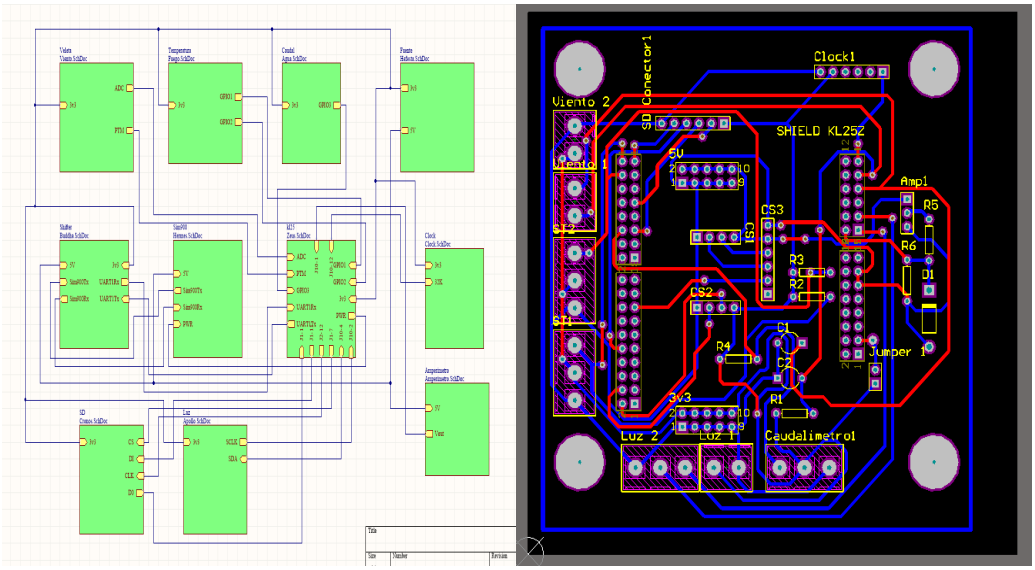
Memoria interna

Se instaló una memoria interna de 8 GBy en un zócalo para tarjetas SD que se conecta por SPI con el módulo microcontrolador, utilizando los pines J1_1 para MOSI, J1_11 para MISO y J2_12 para SCK.

Reguladores lineales

Los reguladores AMS1117-5 y AMS1117-3.3, con salidas de 5V y 3.3V respectivamente, se consiguieron en una única placa para arduino, con pines en forma de protoboard, que se conectan directamente al shield.

Se diseñaron dos circuitos impresos, cada uno con diferentes finalidades. El primer circuito impreso es un shield o una placa interface, en la cual se conectan todos los evaluation board. La figura 16a muestra el esquemático correspondiente al shield y la figura 16b exhibe el circuito impreso. (El esquemático interno de cada bloque verde se encuentra en el material anexo).

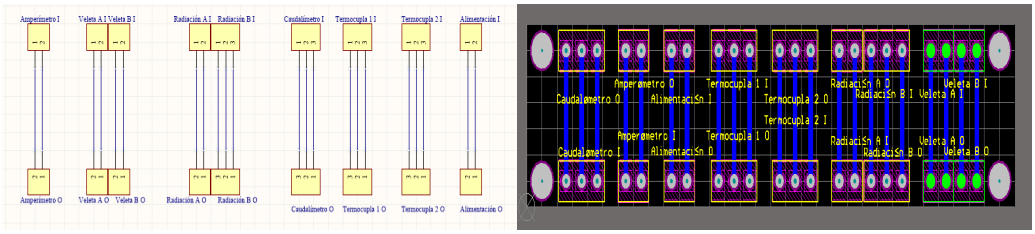


(a) Esquemático (b) Circuito impreso

Figura 16: Esquemático y circuito impreso.

El circuito impreso está conformado por dos planos formando una placa doble faz, el plano superior se grafica en rojo y el inferior en azul. El dibujo de los componentes sobre la placa se indica en color amarillo y su nombre está posicionado sobre o cerca de los mismos.

El segundo circuito impreso está dedicado a ser una placa de conexión entre los sensores que van en el exterior del gabinete, con la placa mencionada anteriormente (Shield). El próposito principal de este circuito impreso es soportar los esfuerzos mecánicos, en la figura 17a se muestra el esquemático y en la figura 17b el circuito impreso.



(a) Esquemático (b) Circuito impreso

Figura 17: Placa conectora

8.1.1. Diseño mecánico

El diseño mecánico tiene como principal objetivo generar una estructura sólida, con un grado aceptable de protección contra polvo y agua, que cumpla con las especificaciones 2 y 18. Para esto se adquirió un gabinete Genrod de $115mm\ X\ 165mm\ X\ 80mm$ de exterior, que está certificado con la norma IP65.

Se realizaron 5 perforaciones al gabinete para poder comunicar los cables de los sensores con el exterior y alimentar eléctricamente al sistema, colocando tapones y prensa-cables para evitar filtraciones de agua por dichas perforaciones. En la figura 18 se puede ver el gabinete con el hardware instalado en su interior y con los cables de los sensores que salen a través de los agujeros.

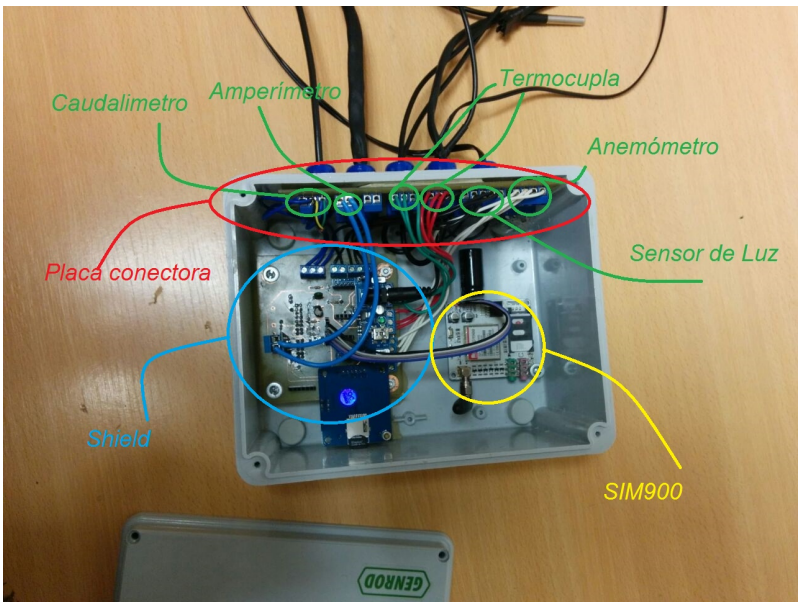


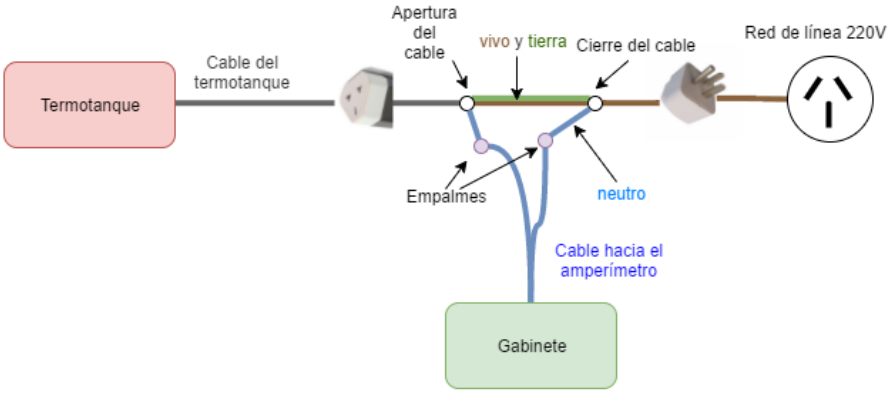
Figura 18: Gabinete

Como se puede observar en dicha imagen, los sensores ingresan al gabinete por los conectores circulares, que están conectados a la placa conectora. Luego, los cables de diferentes colores se conectan al shield, donde a cada sensor se le asignó un color y una etiqueta, para hacer más simple su identificación de acuerdo a las especificaciones de diseño.

8.2. Detalles de construcción y precauciones especiales de montaje

El prototipo cuenta con dos PCB. El primero es un shield que permite la interconexión entre todos los dispositivos que conforman al sistema de medición, mientras que el segundo simplemente se utiliza para hacer de interfaz entre los sensores que van en el exterior y el shield. De esta manera, se busca contener todos los esfuerzos mecánicos que se producen al tirar de los sensores en el PCB de interfaz, y que dichos esfuerzos no se propaguen a la placa principal, causando daños en los dispositivos que se conectan al mismo. Es importante destacar que el gabinete en su conjunto fue diseñado para ser instalado debajo del colector solar, disminuyendo la exposición solar de manera directa y a las precipitaciones.

Conector del amperímetro El conector del amperímetro, figura 19, está formado por un conector hembra y un conector macho. Los cables correspondientes a la tierra y el vivo no ingresan al gabinete, mientras que el cable correspondiente al neutro sí lo hace para que el amperímetro ACS712 mida la corriente que circula por el termotanque. La elección del neutro busca evitar que ingrese la tensión de la red eléctrica ($220V_{rms}$) al gabinete, evitando riesgos de electrocución para las personas que lo manipulen y riesgos de arcos de sobretensión para las placas del sistema. Antes de instalar el amperímetro se debe verificar con un buscapolos que el enchufe de la red eléctrica respete la norma de asignación de vivo, neutro y tierra. En caso de estar invertido, el sistema cuenta con un adaptador que invierte los cables del vivo y el neutro del conector macho. Para mayor detalle sobre las instrucciones de instalación del amperímetro referirse al manual de usuario.

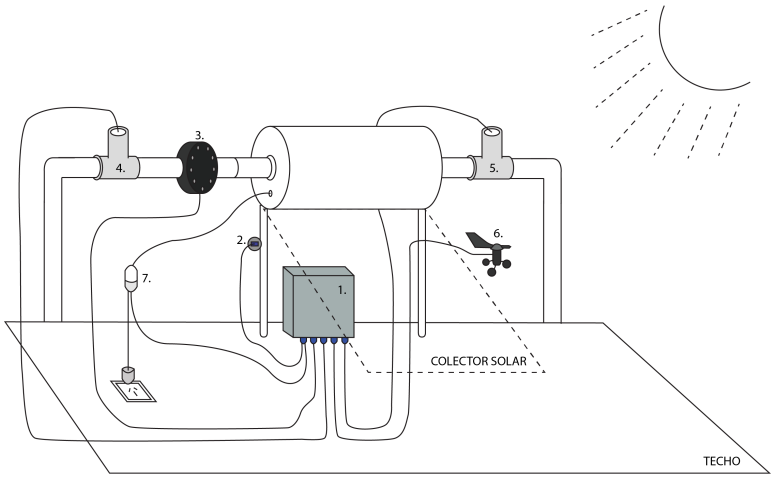


(a) Esquema

(b) Conector

Figura 19: Conector del amperímetro

El gabinete debe ser instalado con los prensa-cables hacia abajo, para evitar posibles filtraciones a través de los conectores por donde salen los cables de los sensores. La figura 20 esquematiza la instalación de los sensores del sistema. Una descripción detallada de la instalación de cada parte puede hallarse en el manual de usuario.



1. Gabinete
2. Sensor de radiación
3. Caudalímetro
4. Sensor de temperatura de entrada
5. Sensor de temperatura de salida
6. Anemómetro
7. Amperímetro

Figura 20: Instalación del sistema de medición

9. Validación del prototipo

9.1. Validación del hardware

9.1.1. Bancos de pruebas

La precisión de los instrumentos de medición (incluyendo los sensores patrón), debe cumplir la inecuación 2 para que la prueba asegure el cumplimiento de la precisión especificada. En cada banco de pruebas, los instrumentos de medición tienen cotas de precisión mínima de acuerdo a dicha inecuación.

$$Precisión\ especificada \leq Precisión\ referencia + precisión\ criterio\ de\ aprobación \tag{2}$$

Banco de pruebas 1 El primer banco de pruebas, figura 21, consiste en la configuración básica del sistema, con un variac y un amperímetro agregados entre el conector a $220V_{rms}$ y la red eléctrica. El sistema debe estar configurado para tomar mediciones cada 2 minutos. Todos los elementos involucrados deben estar expuestos a temperatura ambiente. Además, se debe asegurar que el módulo de comunicaciones tenga la tarjeta SIM colocada, y la cuenta de celular asociada debe tener crédito para acceder a Internet. Los sensores pueden estar no instalados, pero deben permanecer conectados al sistema. El amperímetro de banco o un multímetro digital entre el variac y la fuente AC/DC debe tener una precisión superior al $\pm 3\,\%$ en un rango superior a $100mA_{rms}$.

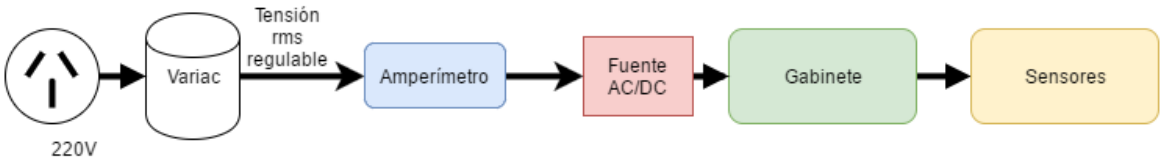


Figura 21: Banco de pruebas 1

Banco de pruebas 2 El segundo banco de pruebas, figura 22, sirve para validar el funcionamiento del prototipo. En esta configuración, la fuente AC/DC debe estar conectada directamente a la red eléctrica. Al igual que en el banco de pruebas 1, el sistema debe trabajar a temperatura ambiente, y debe estar colocada la tarjeta SIM con acceso a Internet. Además, el sistema debe estar configurado para tomar mediciones a intervalos de 2 minutos, salvo en casos donde se requiera otra configuración (test 11). Todos los sensores deben colocarse a una distancia de$1m$ o mayor del gabinete, de acuerdo con la especificación 19. Cada sensor debe estar dispuesto específicamente:

1. Anemómetro: Debe fijarse en una mesa, a una distancia menor de $1m$ de un ventilador. Las aspas del anemómetro deben estar orientadas a 0°del ventilador, de modo que se maximice su giro al prender el ventilador. Debe insertarse una marca fija en la mesa indicando los 0° para poder realizar la prueba de dirección. El compás utilizado para marcar la dirección debe tener una precisión de $\pm 5°$ o superior.

2. Sensor de radiación: Tanto el sensor de luz de referencia como el sensor de radiación del sistema deben colocarse a una distancia menor a $5cm$ de un monitor de 15 pulgadas (o mayor). Ambos sensores debe orientarse de modo que reciban la mayor radiación del monitor posible, y la distancia entre sensores debe ser menor a $10cm$. El sensor de referencia debe tener una precisión superior a $10\frac{W}{m^2}$.

3. Caudalímetro: El caudalímetro debe colocarse en serie con el caudalímetro patrón, y ambos deben formar parte del circuito de un banco de bombas. El caudalímetro patrón debe tener un diámetro de 1 pulgada y una precisión superior a $0{,}25\frac{litros}{min}$. El banco de bombas debe poder ser regulable en el intervalo entre $1\frac{litros}{min}$ y $50\frac{litros}{min}$ o superior.

4. Sensores de temperatura: Los sensores de temperatura deben colocarse dentro de un envase térmico, tal que la totalidad del sensor permanezca en el envase. El diámetro del envase térmico debe ser inferior a 2 pulgadas.

5. Amperímetro: Debe conectarse en serie con un amperímetro de banco y recibir la corriente de una carga que ofrezca corrientes entre $[0, 10]A_{rms}$. Para corrientes menores a $5A_{rms}$ se puede utilizar un variac con una resistencia en serie y para corrientes superiores a $5A_{rms}$ se pueden utilizar una carga (por ejemplo un caloventor) que consuma $5A_{rms}$ o más. La corriente no debe superar los $10A_{rms}$ en ningún momento. El amperímetro de banco utilizado debe soportar corrientes superiores a $10A_{rms}$ y tener una precisión de $0{,}1A_{rms}$ o mayor.

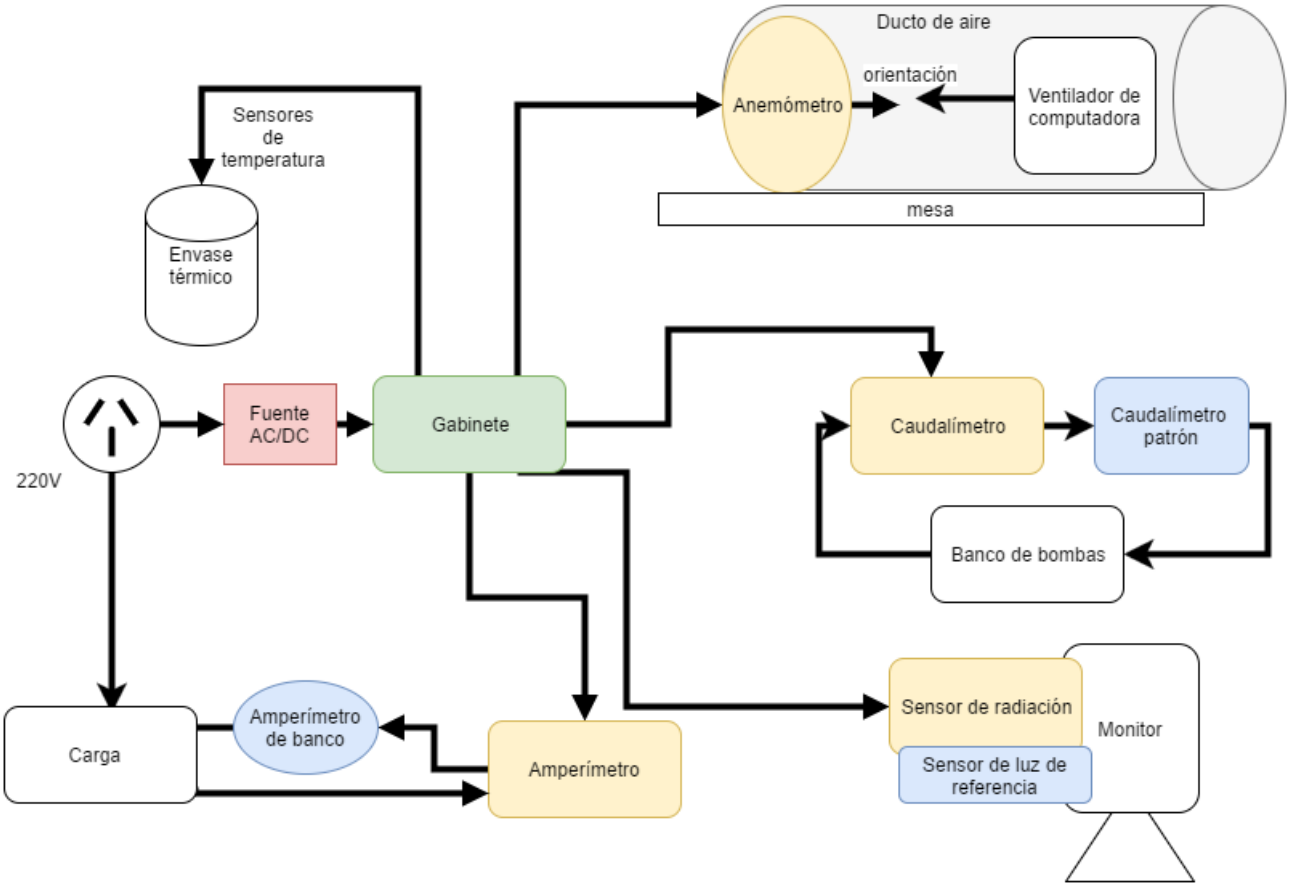


Figura 22: Banco de pruebas 2

Banco de pruebas 3 El tercer banco de pruebas, figura 23, sirve para validar el cumplimiento de las especificaciones de diseño del producto final, específicamente la precisión de los sensores. La configuración de este banco es idéntica al banco 2, en cuanto a la fuente AC/DC, la configuración de la frecuencia de medición, la distancia de sensores al gabinete y la tarjeta SIM con acceso a Internet. Los sensores patrón pueden ser contolados en conjunto por un PLC o individualmente si tienen un display digital, siempre que se respete la precisión especificada para cada sensor patrón. Todo el sistema debe estar dentro de un ambiente con temperatura regulable entre y$61 \pm 1°C$ y $-1 \pm 1°C$. Tanto los sensores del sistema como los sensores patrón deben disponerse específicamente:

1. Anemómetros: Ambos anemómetros deben fijarse en una mesa, a una distancia menor de $1m$ de un ventilador y una distancia menor a $20cm$ entre ellos. El anemómetro patrón debe medir la velocidad del viento con una precisión de $1\frac{km}{hora}$ o superior. Además, el ventilador debe tener una velocidad regulable en el intervalo entre $0\frac{km}{hora}$ y $300\frac{km}{hora}$ o mayor. Las aspas de cada anemómetro deben estar orientadas a 0° del ventilador, de modo que se maximice su giro al prender el ventilador. Debe insertarse una marca fija en la mesa indicando los 0° para poder realizar la prueba de dirección. El compás utilizado para marcar la dirección debe tener una precisión de ±5° o superior.

2. Sensor de radiación: Debe colocarse en una cámara anecoica junto a un piranómetro y ambos deben tener la misma orientación respecto de la fuente de radiación. El piranómetro debe tener una precisión superior a $5\frac{W}{m^2}$ y la fuente de radiación debe tener un espectro de emisión que simule el del sol.

3. Caudalímetro: El caudalímetro debe colocarse en serie con el caudalímetro patrón, y ambos deben formar parte del circuito de un banco de bombas. El caudalímetro patrón debe tener un diámetro de 1 pulgada y una precisión superior a $0{,}25\frac{litros}{min}$. El banco de bombas debe poder ser regulable en el intervalo entre $1\frac{litros}{min}$ y $50\frac{litros}{min}$ o superior.

4. Sensores de temperatura: Los sensores de temperatura deben colocarse dentro de un horno, junto a un sensor patrón de temperatura. La distancia entre los sensores no debe superar los $10cm$. La precisión del sensor patrón debe ser mayor a 0,2°C en el rango $[-10, 85]°C$ y mayor a $0{,}5°C$ en el rango de $[85, 125]°C$.

5. Amperímetro: Debe conectarse en serie con un amperímetro de banco y recibir la tensión de un variac. La corriente no debe superar los $10A_{rms}$ en ningún momento. El amperímetro de banco utilizado debe

soportar corrientes superiores a $10A_{rms}$ y tener una precisión de $0{,}1A_{rms}$ o mayor.

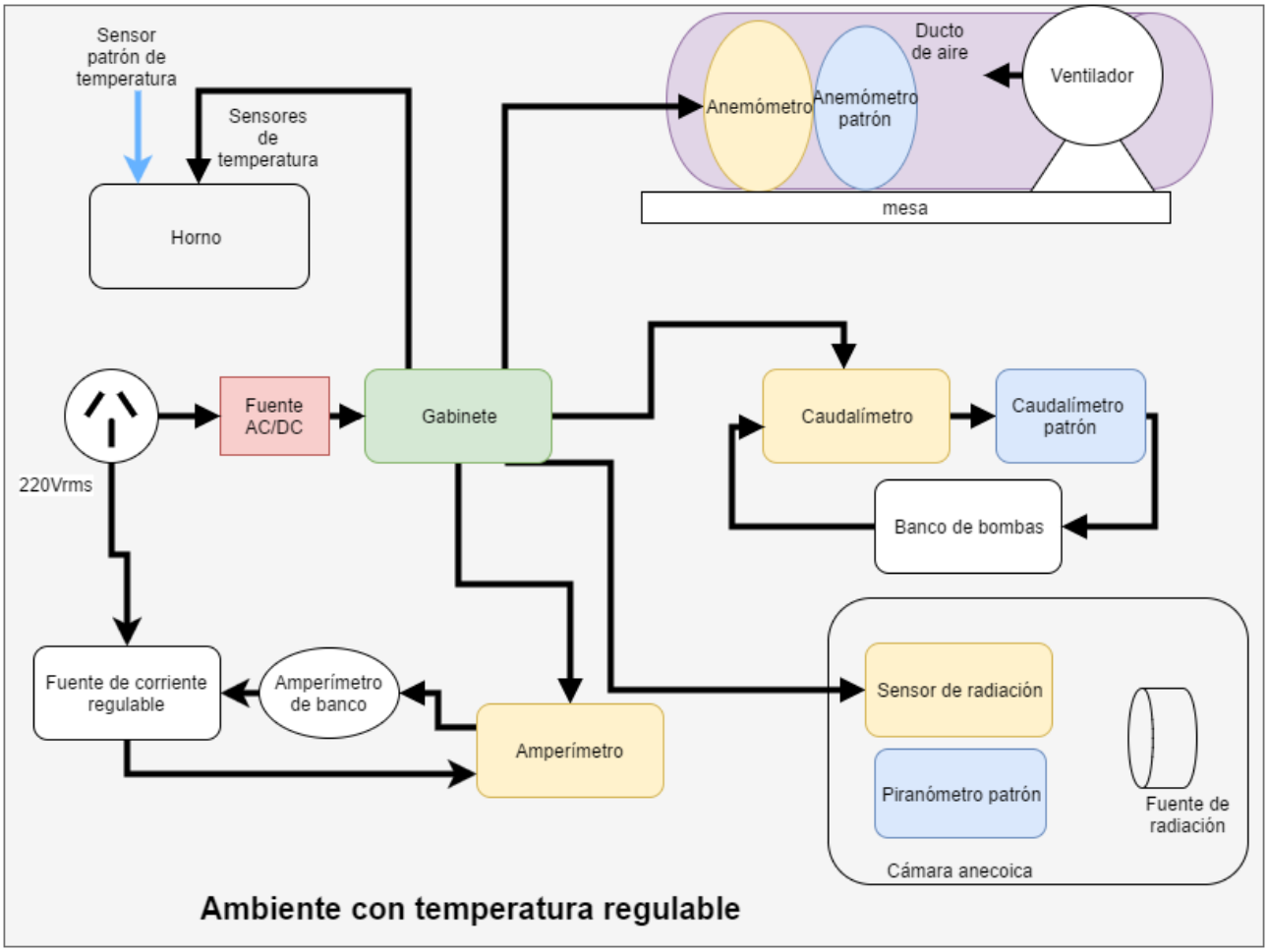


Figura 23: Banco de pruebas 3

9.1.2. Tests

En esta sección se analiza la validación de cada especificación de diseño de hardware: En relación a las especificaciones de los sensores, la validación de la precisión que requiera sensores patrón se restringe al producto final. Para el caso del prototipo, se valida el funcionamiento y repetibilidad de las mediciones, justificando la precisión con las hojas de datos (en ingeniería de detalle). En la sección siguiente, se detallan las mediciones de todas aquellas especificaciones que necesitan una medición en el prototipo.

Test ID	Espec. de diseño	Condiciones de prueba	Test previos	Procedimiento	Criterio de aprobación	Aplic.
1	Tensión de alimentación máxima- 1 Arranque del sistema - 21 Apagado del sistema -23	Banco de pruebas 1 (9.1.1)	24	1. Colocar una tensión de $247 \pm 5V_{rms}$ como alimentación del sistema utilizando el variac. 2. Conectar el sistema a salida del variac. 3. Registrar el tiempo hasta el encendido de los LED. 4. Registrar el tiempo hasta la actualización del servidor remoto. 5. Mantener el sistema conectado durante al menos 10 minutos. 6. Desconectar el sistema	En un tiempo menor a 5 segundos, se deben encender tanto el LED del microcontrolador, el LED de la placa de comunicaciones y el LED del regulador lineal. Además, se deben actualizar las mediciones en el servidor remoto en un tiempo menor a 5 minutos. Durante los 10 minutos de funcionamiento, los LED deben mantenerse prendidos y el servidor remoto debe recibir actualizaciones. La última medición registrada debe haber ocurrido en el intervalo entre la desconexión y los 4 minutos previos.	Prototipo y producto final
2	Tensión de alimentación mínima-1 Arranque del sistema - 21 Apagado del sistema -23	Banco de pruebas 1 (9.1.1)	1	1. Colocar una tensión de $171 \pm 5V$ como alimentación del sistema utilizando el variac. 2. Conectar el sistema a salida del variac. 3. Registrar el tiempo hasta el encendido de los LED. 4. Registrar el tiempo hasta la actualización del servidor remoto. 5. Mantener el sistema conectado durante al menos 10 minutos. 6. Desconectar el sistema	En un tiempo menor a 5 segundos, se deben encender tanto el LED del microcontrolador, el LED de la placa de comunicaciones y el LED del regulador lineal. Además, se deben actualizar las mediciones en el servidor remoto en un tiempo menor a 5 minutos. Durante los 10 minutos de funcionamiento, los LED deben mantenerse prendidos y el servidor remoto debe recibir actualizaciones. La última medición registrada debe haber ocurrido en el intervalo entre la desconexión y los 4 minutos previos.	Prototipo y producto final
3	Grados de protección $IEC529$ - 2	Establecidas por la norma $IEC529$.	5, 7, 10, 14, 15	Se debe realizar el procedimiento según la norma $IEC529$ para el grado de protección $IP34$.	Establecido por la norma $IEC529$ para $IP34$.	Producto final

Test ID	Espec. de diseño	Condiciones de prueba	Test previos	Procedimiento	Criterio de aprobación	Aplic.
4	Temperaturas soportadas - 3	Banco de pruebas 3 (9.1.1)	5, 7, 10, 14, 15	1. Colocar el gabinete junto a todos los sensores dentro del horno. 2. Conectar el sistema a una tensión de $220V_{rms}$. 3. Calentar el horno una temperatura a $61 \pm 1°C$. Esperar que se establezca la temperatura. 4. Realizar los tests 5, 7, 10, 14 y 15. 5. Repetir el procedimiento, pero colocando el horno a $-1 \pm 1°C$.	Se deben aprobar los tests 5, 7, 10, 14 y 15 tanto para la máxima temperatura de operación de $61 \pm 1°C$ como la mínima temperatura $-1 \pm 1°C$.	Producto final
5	Medición de radiación - 4	Banco de pruebas 3 (9.1.1)	6	1. Colocar el sensor del sistema dentro de la cámara anecoica, junto al piranómetro. 2. Realizar 30 mediciones de radiación con una potencia variando linealmente entre$50\frac{W}{m^2}$ y $200000\frac{W}{m^2}$. Cada variación de potencia debe realizarse luego de que se hayan registrado las mediciones del piranómetro y sensor de radiación del sistema.	Todas las mediciones realizadas por el sensor de radiación deben difererir en menos de $40\frac{W}{m^2}$.	Producto final
6	Medición de radiación - 4Relevamiento de datos - 7 Memoria Interna - 10	Banco de pruebas 2 (9.1.1)	19	1. Conectar el sistema a la red eléctrica. 2. Realizar un barrido lineal de escala de grises en el monitor, entre el color #000000 (en formato estándar RGB de 24 bits para cada píxel) y color blanco (color #FFFFFF en formato estándar de 8 bits por color para cada píxel), que tenga una duración total de 25 minutos. Tomar mediciones con el sensor de luz patrón cada intervalos menores a 1 minuto y registrar el momento de cada medición con precisión superior a 30 segundos.	Todas las mediciones subidas al servidor remoto, como las almacenadas en la memoria interna, deben variar menos de $30\frac{W}{m^2}$ respecto de la más cercana temporalmente obtenida por el sensor de referencia. El intervalo entre mediciones debe ser mayor a 1 minuto y 30 segundos y menor a 2 minutos y 30 segundos.	Prototipo

Test ID	Espec. de diseño	Condiciones de prueba	Test previos	Procedimiento	Criterio de aprobación	Aplic.
7	Medición de temperatura de agua de entrada y salida - 5	Banco de pruebas 3 (9.1.1)	8, 9	1. Colocar los sensores de temperatura y el sensor patrón en un horno. El gabinete debe permanecer fuera del horno. 2. Conectar el sistema a la red eléctrica. 3. Tomar 30 mediciones, variando la temperatura linealmente entre $0°C$ y $125°C$. Cada cambio de temperatura debe realizarse luego de que se hayan registrado las mediciones del piranómetro y sensor de radiación del sistema. Para cada temperatura, esperar a que se establezca la medición en ambos sensores.	Las mediciones tomadas por los sensores de temperatura de entrada y salida deben distar en menos de 0,4°C respecto del sensor patrón para el rango $[-10, 85]°C$ y en menos de $1°C$ para el rango de $[85, 125]°C$. Se deben comparar las mediciones con los tiempos de medición más cercanos entre el sensor patrón y el momento de medición guardado en la memoria interna.	Producto final
8	Medición de temperatura de agua de entrada y salida - 5	Banco de pruebas 2 (9.1.1)	6	1. Colocar ambos sensores de temperatura en envase térmico con agua y hielo. 2. Conectar el sistema a la red eléctrica. 3. Esperar que las mediciones de temperatura registradas en el servidor remoto se establezcan. 3. Registrar 5 mediciones en el servidor remoto.	Las 5 mediciones tomadas deben relevar una temperatura de entrada y salida en el intervalo de $0 \pm 0{,}8°C$.	Prototipo
9	Medición de temperatura de agua de entrada y salida - 5	Banco de pruebas 2 (9.1.1)	6	1. Colocar ambos sensores de temperatura en envase con agua hirviendo. 2. Conectar el sistema a la red eléctrica. 3. Esperar que las mediciones de temperatura registradas en el servidor remoto se establezcan. 4. Conectar el sistema a la red eléctrica y esperar que se actualicen 5 mediciones en el servidor remoto.	Las 5 mediciones tomadas deben relevar una temperatura de entrada y salida en el intervalo de$100 \pm 2°C$.	Prototipo

Test ID	Espec. de diseño	Condiciones de prueba	Test previos	Procedimiento	Criterio de aprobación	Aplic.
10	Medición caudal de agua - 6 LED indicador - 20 Relevamiento de datos - 7	Banco de pruebas 3 (9.1.1)	11	1. Colocar el caudalímetro del sistema y el caudalímetro en serie en un banco de bombas. 2. Conectar el sistema a la red eléctrica. 3. Configurar la frecuencia de mediciones en 1440 minutos, tal que el sistema sólo tome mediciones por caudal mayor al mínimo durante la prueba. Apagar el caudal del banco de bombas y esperar que el LED indicador de estado tenga color rojo. 4. Tomar 15 mediciones, variando el caudal linealmente entre $1\frac{litros}{min}$ y $50\frac{litros}{min}$. Para cada presión, esperar a que se establezca la medición en ambos sensores.	Las datos relevados por el caudalímetro del sistema en la memoria interna deben diferir en menos de $0{,}9\frac{litros}{min}$ respecto del caudalímetro patrón. El LED indicador de estado debe cambiar de color rojo a verde en menos de 2 segundos luego de iniciado el paso 4.	Prototipo y producto final
12	Medición de la velocidad del viento - 7	Banco de pruebas 3 (9.1.1)	13, 14	1. Colocar el anemómetro junto a un anemómetro patrón. Ambos sensores deben tener la misma orientación ($\pm 5°$, frente al ventilador) y deben estar a una distancia menor a $50cm$. 2. Conectar el sistema a la red eléctrica. 3. Tomar 30 mediciones, variando la velocidad del ventilador entre $0\frac{km}{hora}$ y $300\frac{km}{hora}$. Para cada velocidad, esperar a que se establezca la medición en ambos sensores.	Las mediciones tomadas el anemómetro del sistema deben distar en menos de $3\frac{km}{hora}$ respecto del sensor patrón.	Producto final
13	Medición de la velocidad del viento - 7	Banco de pruebas 2 (9.1.1)	6	1. Colocar el anemómetro frente al ventilador de computadora, con un error máximo de orientación$\pm 5°$. Colocar un ducto de aire entre ambos. 2. Conectar el sistema a la red de línea. 3. Registrar 10 mediciones en el servidor remoto.	Las velocidad registradas en el servidor remoto deben variar en menos de $3\frac{km}{hora}$ respecto del valor del ventilador de computadora.	Prototipo

Test ID	Espec. de diseño	Condiciones de prueba	Test previos	Procedimiento	Criterio de aprobación	Aplic.
14	Medición de la dirección del viento - 7	Banco de pruebas 2 ó 3 (9.1.1,9.1.1)	6	1. Orientar el anemómetro a 0°, realizando una marca en la mesa en dicha dirección. 2. Conectar el sistema a la red de línea. 3. Con un compás, colocar la paleta de dirección del anemómetro a $22{,}5 \pm 5°$ y esperar a que se actualice la medición en el servidor remoto. 4. Repetir el paso anterior, colocando la paleta a 67,5°, 112,5°, 157,5°, 202,5°, 247,5°, 292,5° y 337,5°	Los octantes relevados en el servidor remoto deben ser los siguientes: 1, 2, 3, 4, 5, 6, 7 y 8. Las mediciones tomadas deben respetar el orden indicado.	Prototipo y producto final
15	Medición del consumo del termotanque - 8	Banco de pruebas 2 ó 3 (9.1.1,9.1.1)	6	1. Colocar la tensión del variac a $0V_{rms}$. 2. Conectar el sistema a la red eléctrica. 3. Tomar 15 mediciones con el amperímetro de banco y el amperímetro del sistema, variando la corriente linealmente entre $0A_{rms}$ y $10A_{rms}$. Modificar la corriente de cada medición luego de haber realizado la medición con ambos amperímetros.	Las mediciones tomadas por el servidor remoto, no deben variar en más de $0{,}3A_{rms}$ respecto de la corriente registrada por el amperímetro de banco.	Prototipo y producto final
16	EMI - 15	Establecidas por la norma $EN55022/CIPSR22$.	5, 7, 10, 14, 15	Se debe realizar el procedimiento establecido por la norma $EN55022/CIPSR22$.	Establecidas por la norma $EN55022/CIPSR22$.	Producto final
17	EMS - 16	Establecidas por la norma $EN55024$.	5, 7, 10, 14, 15	Se debe realizar el procedimiento establecido por la norma $EN55024$.	Establecidas por la norma $EN55024$.	Producto final
18	Vida útil - 13	Ensayo acelerado	5, 7, 10, 14, 15	Utilizar el modelo de Arrhenius para acelerar el desgaste subiendo la temperatura.	La vida útil estimada debe superar los 2 años.	Producto final
19	Funcionamiento - 22 Potencia - 11 GPRS - 9	Banco de pruebas 1 (9.1.1)	2	1. Conectar el sistema a la red eléctrica. 2. En un lapso de 6 horas, relevar 30 mediciones mientras el LED del microcontrolador tenga luz roja y 30 mediciones mientras el LED del microcontrolador tenga luz verde. Las mediciones deben tomarse distanciadas a 15 minutos o más.[1]	Las mediciones tomadas cuando el LED del microcontrolador tenía luz roja, deben tener una media por debajo de los $300mW$. Las mediciones obtenidas cuando el LED tenía luz verde deben tener una media por debajo de $0{,}9W$ y todas deben ser inferiores a $15W$. La corriente de inrush debe ser inferior a $382mA$.	Prototipo y producto final

[1] Se presupone que las condiciones de operación de los sensores no modifican en más de $100mW$ la potencia consumida por el sistema.

Test ID	Espec. de diseño	Condiciones de prueba	Test previos	Procedimiento	Criterio de aprobación	Aplic.
20	Peso - 17	Utilizar una balanza electrónica de precisión superior a 500 gramos.	23	1. Colocar el gabinete y todos los sensores en la balanza. 2. Medir el peso total.	El peso medido debe ser menor a 9,5 kg.	Prototipo y producto final.
21	Dimensiones gabinete - 18	Utilizar una centímetro de precisión superior a $1cm$	23	1. Cerrar la tapa del gabinete 2. Medir el largo, ancho y alto del gabinete con el centímetro.	Las dimensiones medidas deben ser menores a $50cm$ de largo, $50cm$ de ancho y $30cm$ de alto.	Prototipo y producto final.
22	Sensores - 10 Distancia máxima sensores - 19	Banco de pruebas 2 ó 3 (9.1.1,9.1.1)	19	1. Conectar el sistema a la red eléctrica. 2. Excitar todos los sensores con una carga fija. 3. Esperar a que se actualicen 5 mediciones. 4. Sin cambiar las condiciones de los sensores, desconectar el sensor de radiación. Esperar a que se actualicen 5 mediciones 5. Conectar nuevamente el sensor de radiación. 6. Repetir el mismo procedimiento para el resto de los sensores.	Las 10 mediciones tomadas de los sensores que se mantuvieron conectados no deben diferir en más de $0{,}5A_{rms}$ para el amperímetro, $50\frac{W}{m^2}$ para el sensor de radiación, $5\frac{km}{hora}$ y 1 octante para el anemómetro, $1\frac{litros}{min}$ para el caudalímetro y 0,8°C para los sensores de temperatura.	Prototipo y producto final
23	Etiquetado - 12	Banco de pruebas 1 (9.1.1)	-	1. Realizar una inspección visual de los cables de cada sensor del sistema.	Los cables de los sensores deben estar etiquetados de la siguiente forma: 'C' para el caudalímetro, 'AN' para el anemómetro, 'T IN' para el sensor de temperatura de entrada, 'T OUT' para el sensor de temperatura de salida, 'R' para el sensor de radiación y 'AM' para el amperímetro.	Prototipo y producto final
24	Memoria Interna - 10	Banco de pruebas 1 (9.1.1)	20, 21	1. Realizar una inspección visual de la tarjeta SD.	La memoria de la tarjeta SD debe superar los 2 GBy.	Prototipo y producto final

9.1.3. Matriz de trazabilidad y plan de medidas

En el cuadro 7 se resumen las validaciones necesarias para cada especificación de hardware, tanto para el prototipo como para el producto final. El diagrama PERT de la figura 24 organiza las dependencias entre mediciones, y esquematiza el plan de medidas. Primero se realizarán las validaciones más simples, como la medición de peso y dimensiones, luego se realizarán las pruebas de funcionamiento básico con el banco de pruebas 1 y finalmente se validarán las especificaciones de los sensores con el banco de pruebas 2. La validación del producto final con el banco de pruebas 3 queda para un trabajo futuro.

Nro. de especif.	Nombre de especificación	Test ID prototipo	Test ID producto final
1	Tensión de alimentación	1, 2	1, 2
2	Grados de protección $IEC529$	-	3
3	Temperaturas	-	4
4	Medición de intensidad solar	6	5
5	Medición de temperatura del agua de entrada y salida del colector solar	8, 9	7
6	Medición de caudal de agua	10	10
7	Medición del viento	13, 14	12, 14
8	Medición de consumo del termotanque	6	6
9	Conexión GPRS	19	19
10	Memoria Interna	6, 24	6, 24
11	Potencia	19	19
12	Etiquetado	23	23
13	Vida útil	-	18
14	Precio al consumidor	Ver documentación 6.3	Ver documentación 6.3
15	EMI	-	16
16	EMS	-	17
17	Peso	20	20
18	Dimensiones	21	21
19	Distancia máxima sensores	22	22
20	LED indicador de estado	11	10
21	Arranque del sistema	1, 2	1, 2
22	Funcionamiento	19	19
23	Apagado del sistema	1,2	1,2
24	Documentación de hardware	Ver secciones 7 y 8	

Cuadro 7: Matriz de trazabilidad

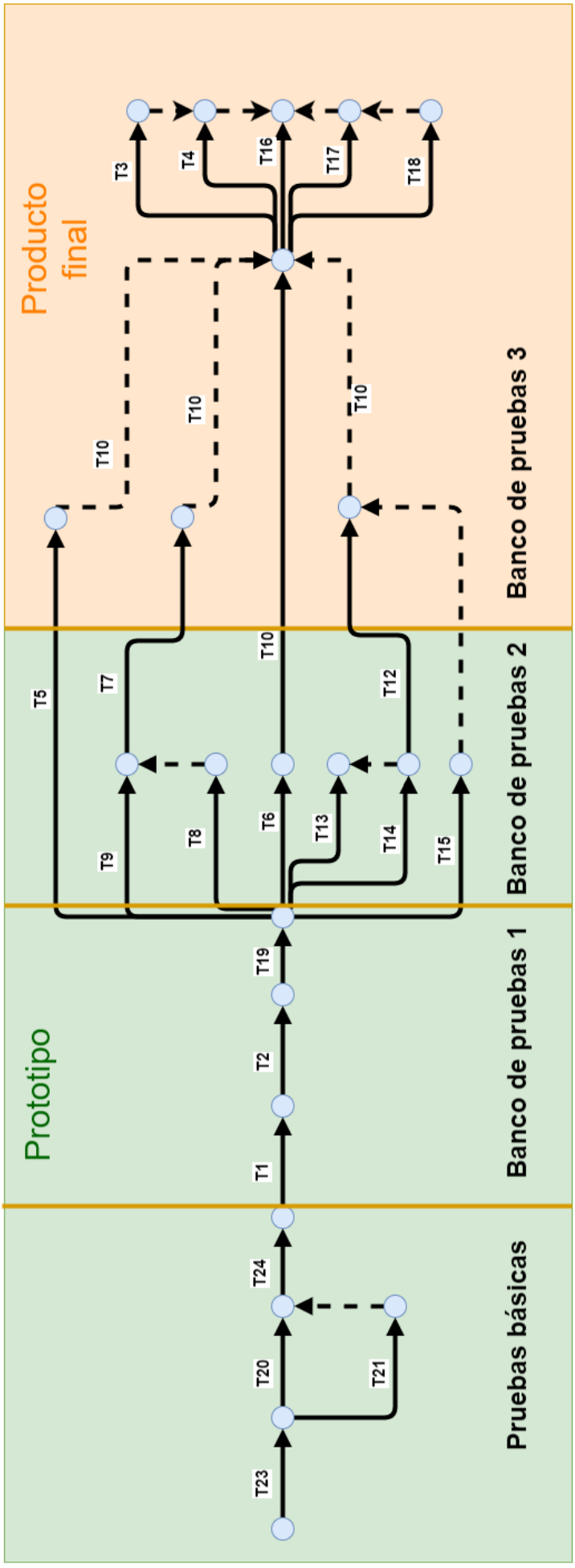


Figura 24: Diagrama PERT de las pruebas de validación

9.1.4. Medidas

Test ID	Espec. de diseño	Criterio de aprobación	Resultados
1	Tensión de alimentación máxima- 1 Arranque del sistema - 21 Apagado del sistema -23	En un tiempo menor a 5 segundos, se deben encender tanto el LED del microcontrolador, el LED de la placa de comunicaciones y el LED del regulador lineal. Además, se deben actualizar las mediciones en el servidor remoto en un tiempo menor a 5 minutos. Durante los 10 minutos de funcionamiento, los LED deben mantenerse prendidos y el servidor remoto debe recibir actualizaciones. La última medición registrada debe haber ocurrido en el intervalo entre la desconexión y los 4 minutos previos.	Aprobada con una tensión de $248V_{rms}$.
2	Tensión de alimentación mínima-1 Arranque del sistema - 21 Apagado del sistema -23	En un tiempo menor a 5 segundos, se deben encender tanto el LED del microcontrolador, el LED de la placa de comunicaciones y el LED del regulador lineal. Además, se deben actualizar las mediciones en el servidor remoto en un tiempo menor a 5 minutos. Durante los 10 minutos de funcionamiento, los LED deben mantenerse prendidos y el servidor remoto debe recibir actualizaciones. La última medición registrada debe haber ocurrido en el intervalo entre la desconexión y los 4 minutos previos.	Aprobada con una tensión de $164V_{rms}$.
6	Medición de radiación - 4Relevamiento de datos - 7 Mediciones - ?? Memoria Interna - 10	Todas las mediciones subidas al servidor remoto, como las almacenadas en la memoria interna, deben variar menos de $30\frac{W}{m^2}$ respecto de la más cercana temporalmente obtenida por el sensor de referencia. El intervalo entre mediciones debe ser mayor a 1 minuto y 30 segundos y menor a 2 minutos y 30 segundos.	Aprobada. Las mediciones de los sensores variaron $16\frac{W}{m^2}$, el intervalo mínimo fue de 1minuto y 41 segundos y el intervalo máximo de 2 minutos y 20 segundos. La memoria interna y el servidor remoto almacenaron las mismas mediciones.
8	Medición de temperatura de agua de entrada y salida - 5	Las 5 mediciones tomadas deben relevar una temperatura de entrada y salida en el intervalo de $0 \pm 0{,}8$°C.	Aprobada. El sensor de temperatura entrada tiene un margen de 0,425ºC y el sensor de salida presenta un margen de $0{,}3^{o}C$.
9	Medición de temperatura de agua de entrada y salida - 5	Las 5 mediciones tomadas deben relevar una temperatura de entrada y salida en el intervalo de$100 \pm 2°C$.	Aprobada. El sensor de temperatura de entrada presenta un margen de 1,0625ºC y el sensor de temperatura de salida presenta un margen de 1,1250ºC.

Test ID	Espec. de diseño	Criterio de aprobación	Resultados
10	Medición caudal de agua - 6 LED indicador - 20 Relevamiento de datos - 7	Las datos relevados por el caudalímetro del sistema en la memoria interna deben diferir en menos de $0{,}5\frac{litros}{min}$ respecto del caudalímetro patrón. El LED indicador de estado debe cambiar de color rojo a verde en menos de 2 segundos luego de iniciado el paso 4.	Aprobado. Mediciones de caudal tienen una precisión de $0{,}24\frac{litros}{min}$. El LED se cambió de color rojo a verde en menos de 1 segundo.
13	Medición de la velocidad del viento - 7	Las velocidad registradas en el servidor remoto deben variar en menos de $3\frac{km}{hora}$ respecto del valor del ventilador de computadora.	Aprobada con un margen de $1{,}81\frac{km}{hora}$.
14	Medición de la dirección del viento - 7	Los octantes relevados en el servidor remoto deben ser los siguientes: 1, 2, 3, 4, 5, 6, 7 y 8. Las mediciones tomadas deben respetar el orden indicado.	Aprobada.
15	Medición del consumo del termo-tanque - 8	Las mediciones tomadas por el servidor remoto, no deben variar en más de $0{,}3A_{rms}$ respecto de la corriente registrada por el amperímetro de banco.	Aprobada con un margen de $0{,}17A_{rms}$.
19	Funcionamiento - 22 Potencia - 11 GPRS - 9	Las mediciones tomadas cuando el LED del microcontrolador tenía luz roja, deben tener una media por debajo de los $300mW$. Las mediciones obtenidas cuando el LED tenía luz verde deben tener una media por debajo de $0{,}9W$ y todas deben ser inferiores a $15W$.	Aprobada parcialmente. La potencia media es inferior a $1W$ con un margen de $265mW$, pero en intervalos de bajo consumo es mayor a $300mW$, con un excedente de$146mW$. (Ver 9.1.5)
20	Peso - 17	El peso medido debe ser menor a $10kg$.	Aprobada con un margen de $8{,}5kg$.
21	Dimensiones gabinete - 18	Las dimensiones medidas deben ser menores a $50cm$ de largo, $50cm$ de ancho y $30cm$ de alto.	Aprobada con un margen de $28{,}9cm$ para el largo, $33{,}5cm$ para el ancho y $20{,}8cm$ para el alto.
22	Sensores - 10 Distancia máxima sensores - 19	Las 10 mediciones tomadas de los sensores que se mantuvieron conectados no deben diferir en más de $0{,}5A_{rms}$ para el amperímetro, $50\frac{W}{m^2}$ para el sensor de radiación, $5\frac{km}{hora}$ y 1 octante para el anemómetro, $1\frac{litros}{min}$ para el caudalímetro y 0,8°C para los sensores de temperatura.	Aprobado con un margen de $0{,}46A_{rms}$ para el amperímetro, $4{,}48\frac{km}{hora}$ y 1 octante para el anemómetro, $0{,}2\frac{litros}{min}$ para el caudalímetro, $40{,}90\frac{W}{m^2}$ para el sensor de radiación y 0,4875°C para los sensores de temperatura.
23	Etiquetado - 12	Los cables de los sensores deben estar etiquetados de la siguiente forma: 'C' para el caudalímetro, 'A' para el anemómetro, 'T IN' para el sensor de temperatura de entrada, 'T OUT' para el sensor de temperatura de salida, 'R' para el sensor de radiación y 'A' para el amperímetro.	Aprobada.
24	Memoria Interna - 10	La memoria de la tarjeta SD debe superar los 2 GBy.	Aprobada con un margen de 6GBy.

Test 1 Se conectó el sistema a un variac monofásico de laboratorio marca 'El Toroide' (ver figura 25). El variac tiene un voltímetro que mide la tensión de salida con una precisión superior a $1V_{rms}$. En primera instancia, se probó encender el sistema con $243V_{rms}$, para validar el mínimo requerido por la especificación de diseño. El sistema arrancó correctamente y se actualizaron las mediciones en un tiempo menor a 3 minutos. El sistema se dejó funcionando durante 10 minutos, y no se comprobaron anomalías: Las mediciones se actualizaron correctamente en la página de ThingSpeak y en la memoria interna. A continuación, se realizó el mismo procedimiento con una tensión de $248V_{rms}$ y el resultado fue el mismo. En ambos casos, el LED del microcontrolador, el LED del módulo de comunicaciones y el LED del regulador lineal se prendieron en menos de 1 segundo. Los resultados obtenidos superan la prueba, con un margen de $5V_{rms}$ de sobretensión. La última medición registrada ocurrió 1 minuto y 20 segundos antes de la desconexión del sistema.

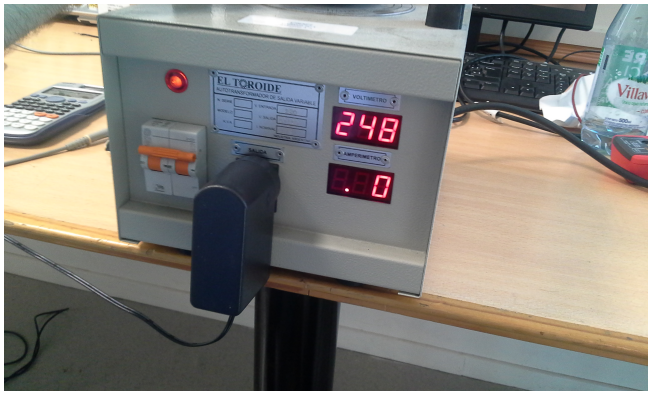


Figura 25: Conexión del sistema a la salida de un variac con sobretensión.

Test 2 Al igual que en el test 1, se conectó el sistema a un variac monofásico de precisión superior a $1V_{rms}$ (ver figura 26). En primer lugar se conectó el sistema a una tensión de $176V_{rms}$, para verificar el funcionamiento del sistema y cumplir el mínimo requerido por la especificación. Se prendió el LED del microcontrolador, el LED del módulo de comunicaciones y el LED del regulador lineal en menos de 1 segundo y se actualizaron mediciones en el servidor remoto en menos de 3 minutos. La prueba se repitió para una tensión menor de $164V_{rms}$, y los resultados obtenidos fueron los mismos. La prueba fue exitosa, ya que superó en $8V_{rms}$ la especificación de funcionamiento a baja tensión. La última medición registrada ocurrió 1 minuto y 40 segundos antes de la desconexión del sistema.

Figura 26: Conexión del sistema a la salida de un variac con baja tensión

Test 6 En esta prueba se colocó el sensor de radiación del sistema junto al sensor de luz TSL2561, ambos adheridos al centro de un monitor (ver figura 28). Las mediciones del sensor de radiación se guardaron en una computadora (a través de una conexión USB) a intervalos de 2 segundos. El sistema fue configurado para tomar mediciones cada 2 minutos, su mínimo intervalo de medición. Se realizó un barrido de escala de grises con un programa de MATLAB, que empezó con el color negro y aumentó su intensidad linealmente hasta llegar al blanco luego de 30 minutos. Luego de los 30 minutos, la pantalla volvió a color negro. Los datos del sensor de radiación del sistema se descargaron de ThingSpeak y la memoria interna, y no hubieron variaciones entre ambas fuentes.

Las mediciones de ambos sensores se muestran en el gráfico de la figura 27. A pesar de que la escala de grises se varió linealmente, la curva medida no es una recta, ya que el monitor aplicó una corrección gamma.

La diferencia máxima entre las mediciones ambos sensores es $16\frac{W}{m^2}$, que se encuentra por debajo del mínimo necesario para aprobar la prueba, $30\frac{W}{m^2}$ (que considera el error del sensor patrón).

En relación al intervalo de tiempo entre mediciones subidas al servidor remoto, el intervalo mínimo fue de 1 minuto y 41 segundos y el intervalo máximo fue de 2 minutos y 20 segundos. Estos resultados se encuentran dentro de los valores permitidos por la especificación, por lo que la prueba se considera aprobada.

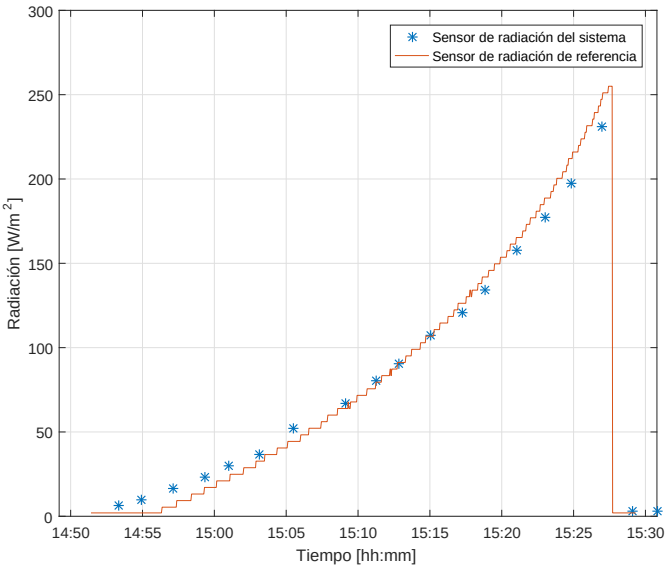


Figura 27: Mediciones de radiación

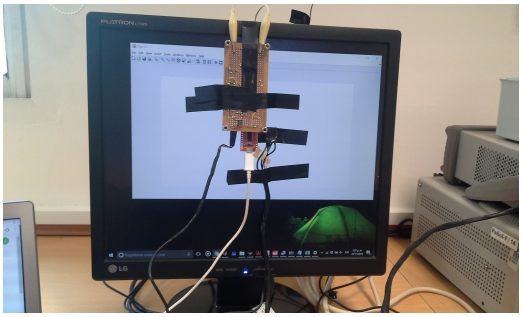


Figura 28: Medición de la radiación

Test 8 Se colocaron ambos sensores de temperatura en un envase térmico con una mezcla de 80 % hielo y 20 % agua fría (ver figura 29). Los sensores se ingresaron en el envase con $10cm$ de profundidad y con una distancia menor a $3cm$ entre ellos. En estas condiciones, se esperó a que se establecieran las mediciones de temperatura, y luego se registraron 5 mediciones que se muestran en la tabla 9. Ambos sensores presentan mediciones dentro del rango de $0 \pm 0{,}8$°C, por lo que la prueba se considera aprobada. El sensor de temperatura entrada tiene un margen de 0,425ºC y el sensor de salida presenta un margen de $0{,}3^{o}C$.

Sensor de temperatura de entrada [°C]	Sensor de temperatura de salida [°C]
0.3750	0.3125
0.2500	0.5000
0.3750	0.3750
0.2500	0.3125
0.3125	0.5000

Cuadro 9: Medición de agua con hielo

Figura 29: Medición de 0°C

Test 9 En esta prueba, se colocaron los sensores de temperatura en un envase térmico con agua hirviendo (ver figura 30). Los sensores se ingresaron en el envase con $10cm$ de profundidad y con una distancia menor a $3cm$ entre ellos. Una vez que las mediciones de temperatura se estabilizaron, se registraron 5 mediciones de cada sensor (tabla 10). Todas las mediciones se encuentran dentro del intervalo de aprobación, $100 \pm 2°C$, por lo que la prueba se considera aprobada. El sensor de temperatura de entrada presenta un margen de 1,0625ºC respecto al límite de aprobación, y el sensor de temperatura de salida presenta un margen de 1,1250ºC.

Sensor de temperatura de entrada [°C]	Sensor de temperatura de salida [°C]
99.0625	99.3125
99.5000	99.1250
99.2500	98.9375
98.9375	98.8750
99.2500	99.1250

Cuadro 10: Mediciones de agua hirviendo

Figura 30: Medición de 100°C

Test 10 El test se llevó a cabo en el banco de bombas del ITBA (ver figura 31), donde se colocó el caudalímetro del sistema en serie con el sensor $SV8050$, que mide con una precisión del 0.5 % en el rango de $[0, 150]\frac{litros}{min}$. En el rango de interés de la prueba, $[0, 50]\frac{litros}{min}$, la precisión a fondo de escala es $0{,}25\frac{litros}{min}$, igual al valor mínimo necesario para realizar la prueba. Se tomaron 15 mediciones en el rango de $[0, 50]\frac{litros}{min}$ y los resultados se muestran en la tabla 11. El LED indicador de estado cambió de color rojo a verde en menos de 1 segundo después de haber iniciado el caudal. La máxima diferencia entre la medición del sensor patrón y del caudalímetro del sistema, fue de $0{,}24\frac{litros}{min}$, por lo que la prueba se considera aprobada con un margen de $0{,}26\frac{litros}{min}$.

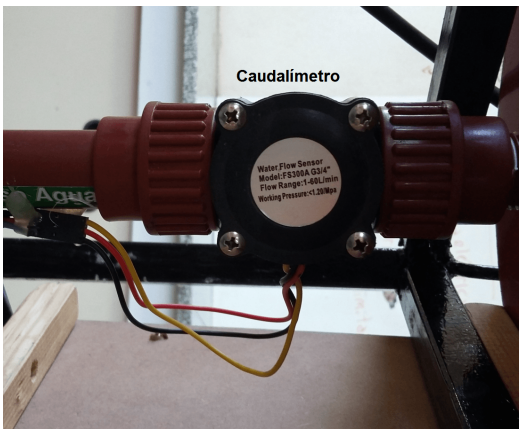


Figura 31: Medición del caudal de agua en un banco de bombas

Caudalímetro del sistema$[\frac{litros}{min}]$	Sensor patron$[\frac{litros}{min}]$
0.00	0.00
3.53	3.55
6.12	5.65
9.11	8.95
10.51	10.75
12.32	12.30
15.21	15.15
19.87	19.93
25.12	25.25
31.32	31.43
33.80	33.95
39.22	39.41
40.11	40.32
49.22	49.34

Cuadro 11: Mediciones de caudal

Test 13 Se colocó un ventilador de computadora en el extremo de un ducto de aire de11,4cm x 11,4cm (ver figura 32). El anemómetro del sistema se fijó a una mesa, tal que sus copas queden justo en el otro extremo del ducto.

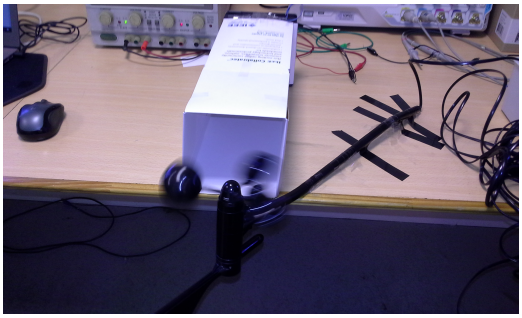


Figura 32: Medición de la velocidad del viento

Se tomaron 3 mediciones a 9 velocidades diferentes, aplicando tensiones al ventilador en el rango de $[0, 16]V_{DC}$, como muestra la tabla 12. A partir de dichas mediciones, se elaboró la tabla 13, donde se calculó la media y diferencia máxima para cada tensión aplicada al ventilador. De esta tabla, se pueden sacar dos conclusiones:

- La curva es monótona, indicando que a mayor velocidad de giro del ventilador se mide mayor velocidad del viento
- Las mediciones presentan una repetibildad con un error máximo de 1,09$\frac{km}{hora}$.

Por estas razones, la prueba se considera aprobada con un margen de 1,81 $\frac{km}{hora}$.

Tensión aplicada al ventilador [V_{DC}]	Velocidad medida [$\frac{km}{hora}$]
16	12.79
16	13.29
16	12.20
13.8	10.55
13.8	10.50
13.8	10.16
13.1	9.99
13.1	10.3
13.1	10.1
11.65	9.60
11.65	9.55
11.65	9.59
10.65	8.27
10.65	8.85
10.65	8.20
10.05	8.34
10.05	8.36
10.05	7.72
9.65	7.72
9.65	7.38
9.65	7.38
6.70	2.34
6.70	2.71
6.70	2.80
0	0
0	0
0	0

Cuadro 12: Velocidades de viento medidas

Tensión aplicada al ventilador [V_{DC}]	Media de las mediciones [$\frac{km}{hora}$]	Máxima diferencia entre mediciones [$\frac{km}{hora}$]
16	12.76	1.09
13.8	10.40	0.39
13.1	10.13	0.31
11.65	9.58	0.05
10.65	8.44	0.65
10.05	8.13	0.64
9.65	7.49	0.34
6.70	2.61	0.46
0	0	0

Cuadro 13: Media y diferencia máxima para cada tensión aplicada al ventilador

Test 14 Se imprimió un compás en un hoja y se fijó el anemómetro en el centro de la misma, tal como lo muestra la figura 33. Luego se conectó el sistema la red eléctrica y se relevaron 8 mediciones, colocando el aspa del anemómetro con las direcciones de la tabla 14. Las mediciones obtenidas siguieron el resultado esperado por cuadrante, y la prueba fue aprobada.

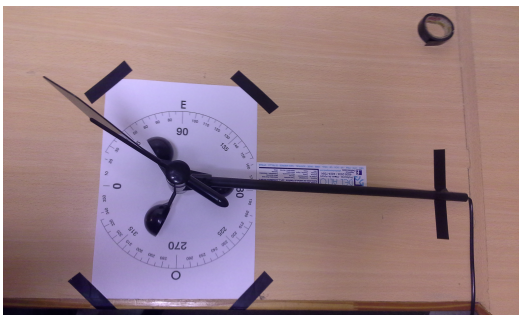


Figura 33: Validación de la medición de la dirección del viento

Orientación del aspa del anemómetro	Octante medido
22,5°	1
67,5°	2
112,5°	3
157,5°	4
202,5°	5
247,5°	6
292,5°	7
337,5°	8

Cuadro 14: Mediciones de octante

Test 15 Para realizar esta prueba se conectó el amperímetro del sistema al cable de una estufa que ofrece 3 potencias diferentes (ver figura 34b). Además, se colocó la pinza amperométrica EXTECH Instruments 380947 midiendo la corriente de la estufa, para poder comparar las mediciones realizadas por el amperímetro del sistema (ver figura 34a). La pinza amperométrica tiene una precisión de 1.5 % y mide hasta 400A. En el rango de medición de la prueba, $[0, 10A_{rms}]$, la precisión mínima de la pinza es de $0{,}15A_{rms}$ que supera la precisión mínima necesaria para la prueba ($0{,}2A_{rms}$). En esta configuración, se realizaron 5 mediciones para cada potencia posible de la estufa, y el resultado se muestra en la tabla 15. El error máximo de las mediciones realizadas es de $0{,}13A_{rms}$, que se encuentra por debajo de $0{,}3A_{rms}$, por lo que la prueba se considera aprobada con un margen de $0{,}17A_{rms}$.

(a) Medición de la corriente con una pinza amperométrica

(b) Configuración de la estufa

Figura 34: Mediciones de consumo de corriente con una estufa.

Potencia del calefactor [W]	Amperímetro del sistema [A_{rms}]	Pinza amperométrica [A_{rms}]
800	3.41	3.40
800	3.38	3.39
800	3.42	3.40
800	3.38	3.40
800	3.41	3.40
1200	5.06	5.13
1200	5.04	5.12
1200	5.03	5.12
1200	5.03	5.11
1200	5.01	5.10
1600	6.67	6.82
1600	6.67	6.79
1600	6.65	6.78
1600	6.68	6.79
1600	6.66	6.79

Cuadro 15: Mediciones de corriente

Test 19 Se midió el consumo de corriente del sistema durante 6 horas (mientras se realizaron otras pruebas), tomando 30 mediciones en los intervalos de medición y 30 mediciones en los intervalos de bajo consumo. Las mediciones se realizaron colocando un multímetro UNI-T UT39C entre el variac con $220V_{rms}$ de salida y la fuente del sistema (ver figura 35). El multímetro tiene una precisión de 1.8 % en el rango de corrientes alternas menores a $500mA_{rms}$, que cumple con la precisión necesaria para la prueba de 3 %. En los intervalos de medición, se registró una corriente media de $3{,}34mA_{rms}$ y un máximo de $4{,}64A_{rms}$, mientras que en los intervalos de bajo consumo, se registró una corriente media de $2{,}03mA_{rms}$. Además, se midieron $215mA$ de corriente de encendido (inrush) del sistema, midiendo con un osciloscopio los bornes de una resistencia de 1Ω entre la fuente switching y la red eléctrica.

Considerando la tensión del variac, la potencia media durante intervalos de medición fue $735mW$, los picos de potencia (menos en el inicio) menores a $1W$ y la potencia media durante los intervalos de bajo consumo fue de $446mW$. La prueba se considera parcialmente aprobada: Por un lado, la potencia media en los intervalos de medición es inferior a $0{,}9W$, la corriente de inrush se encontró por debajo de $388mA$ y los picos de potencia son inferiores a $15W$. Por otro lado, la potencia consumida durante el intervalo de bajo consumo supera los $300mW$. La evaluación de las implicancias de este resultado se encuentra en la sección 9.1.5.

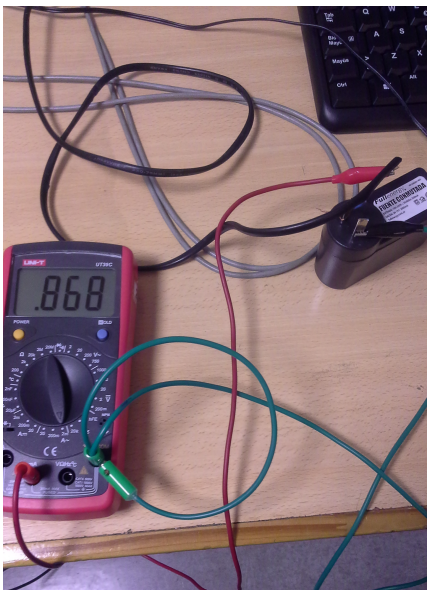


Figura 35: Medición del consumo del sistema

Test 20 Se colocó el gabinete y todos los sensores del sistema en una balanza electrónica de precisión mayor a $100gramos$, obteniendo un peso total de $1{,}4kg$. La prueba se considera aprobada con un margen de $8{,}5kg$.

Test 21 Se midió el gabinete del prototipo con un centímetro de precisión de 0,5mm. Las medidas obtenidas fueron:

$$Largo = 21{,}1cm$$

$$Ancho = 16{,}5cm$$

$$Alto = 9{,}2cm$$

Estas medidas se encuentran por debajo del límite de 50cmx50cmx30cm, por lo que la prueba se encuentra aprobada, con un margen de 28,9cm para el largo, 33,5cm para el ancho y 20,8cm para el alto.

Test 22 En esta prueba, los sensores del sistema se mantuvieron con cargas constantes, de modo de asegurar la repitibilidad de las mediciones. El anemómetro se colocó en las condiciones de los test 13 y 14, aplicando 13,8V_{DC} al ventilador de computadora. Además, se colocó el caudalímetro dentro del ducto de aire para obtener una caudal constante. Los sensores de temperatura se colocaron en el envase térmico de los tests 8 y 9, pero con agua a temperatura ambiente. El sensor de radiación se mantuvo adherido a la pantalla como en el test 6 y la pantalla se mantuvo en color blanco. El amperímetro se conectó a la estufa al igual que en el test 15, que se mantuvo prendida con una potencia de 800W.

Los sensores se fueron desconectando de a uno a la vez y se tomaron 10 mediciones en cada caso, elaborando la tabla 36. Las máximas variaciones de los sensores fueron:

- Amperímetro: 0,04A_{rms}. Aprobado con un margen de 0,46A_{rms}.
- Velocidad del viento: 0,52$\frac{km}{hora}$. Aprobado con un margen de 4,48$\frac{km}{hora}$.
- Dirección del viento: 1 octantes. Aprobado.
- Caudalímetro:
- Radiación: 10,10$\frac{W}{m^2}$. Aprobado con un margen de 40,90$\frac{W}{m^2}$
- Temperatura de entrada: 0,3125°C. Aprobado con un margen de 0,4875°C.
- Temperatura de salida: 0,3125°C. Aprobado con un margen de 0,4875°C.

Máxima variación\Sensor desconectado	Amperímetro	An. vel. del viento	An. dir. del viento	Caudalímetro	Radiación	Temp. entrada	Temp. salida
Corriente [A_{rms}]	X	0.04	0.03	0.04	0.04	0.02	0.03
Velocidad del viento [$\frac{km}{hora}$]	0.38	X	0.41	0.35	0.52	0.43	0.36
Dirección del viento [octante]	0	0	X	0	0	0	0
Caudal de agua [$\frac{litros}{min}$]	0.14	0.20	0.13	X	0.17	0.17	0.19
Radiación [$\frac{W}{m^2}$]	5.50	10.10	7.30	9.20	X	8.30	7.20
Temperatura entrada [°C]	0.2500	0.3125	0.1875	0.3125	0.1250	X	0.1875
Temperatura salida [°C]	0.1500	0.1250	0.3125	0.375	0.1250	0.3125	X

Figura 36: Mediciones con un sensor desconectado

Test 23 Se verificó que las etiquetas de los sensores respeten la siguiente convención: C' para el caudalímetro, 'A' para el anemómetro, 'T IN' para el sensor de temperatura de entrada, 'T OUT' para el sensor de temperatura de salida, 'R' para el sensor de radiación y 'A' para el amperímetro. La prueba se considera aprobada.

Test 24 Se realizó una inspección visual del gabinete, y validó la presencia de una tarjeta de memoria SD *Kingston* de 8GBy. Como el valor de la memoria supera los 2GBy, la prueba se considera aprobada.

9.1.5. Evaluación

Todas las pruebas de validación del prototipo fueron superadas, menos el test 19 (consumo de potencia del sistema) que se aprobó parcialmente. Las pruebas básicas, (tests 20, 21, 22 y 24) fueron superadas holgadamente, con un margen de tamaño y peso que podrían ser utilizados para incorporar más funcionalidades en una versión futura del proyecto.

Las pruebas relacionadas a las variaciones de tensión de la red eléctrica fueron aprobadas con un margen de 5 V_{rms}, y aseguran el funcionamiento del producto en la red eléctrica de la mayoría de las distribuidoras en Argentina.

El test 6, relacionado al intervalo de medición y al sensor de radiación, fue superado con un buen margen para el sensor de radiación, pero se obtuvieron resultados ajustados para el intervalo de medición (un margen de $10 segundos$). Al realizar un análisis más detallado, se encontró que la principal causa de la alta variabilidad del intervalo de medición se debe a la incertidumbre del tiempo de conexión a la red GPRS, Este último punto podría ser sujeto de mejoras en versiones futuras del producto, que busquen controlar con mayor precisión la conexión a la red GPRS.

Los tests 8, 9, 10, 13, 14 y 15, referidos a los sensores del sistema, fueron aprobados con buen margen, obteniendo precisiones similares o superiores a las indicadas en las hojas de datos de los sensores. Se puede concluir que la adquisición y envío de las mediciones no aportan un error significativo. La precisión medida de los sensores de temperatura, 0,425°C, es similar a los 0,5°C de la hoja de datos del sensor DS18B20. Las mediciones de corriente presentaron una precisión de $0{,}13 A_{rms}$, también similar a los $0{,}15 A_{rms}$ especificados por la hoja de datos del ACS712. Además, la precisión de las mediciones de velocidad del viento, $1{,}09 \frac{km}{hora}$, superaron la precisión dada por la hoja de datos del anemómetro, $3 \frac{km}{hora}$. El test 22 confirmó el correcto desarrollo del software, ya que la variabilidad de las mediciones de los sensores no se vió afectada por la desconexión de uno de los sensores.

En relación a la corriente de inrush del test 19, primero se midió un máximo de $1A$ que excedía el límite especificado. Sin embargo, se encontró que el problema era causado por la fuente switching del sistema, que no poseía un termistor NTC que limitara la carga de los capacitores en el momento de encendido. Por esta razón, se debió cambiar la fuente original por otra de características y precio similar que tuviera especificada la corriente de inrush máxima (ver 7.0.1). La modularidad del sistema permitió modificar la fuente switching sin tener que cambiar los demás módulos. Con el cambio de fuente, se midieron $215mA$ de corriente de inrush, que cumplieron con la especificación de diseño. Además, la prueba relacionada al consumo de potencia del sistema aprobó el límite de consumo de $1W$ impuesto por la especificación 11 con un margen considerable ($265mW$), pero no supero el máximo de potencia admisible para intervalos de bajo consumo por un exceso de $146mW$. Luego de realizar un análisis de potencia de cada módulo por separado, se halló que el exceso provenía en su mayoría por la baja eficiencia de las fuentes de alimentación que consumían $217mW$, aún cuando el resto de los módulos estaba apagado o en modo de bajo consumo. Sin embargo, el incumplimiento de este punto de la especificación no es crítico para el cumplimiento de los requerimientos del cliente, ya que proviene de un error de sobreespecificación. La especificación de diseño debería indicar únicamente el límite de potencia promedio que debería consumir el producto, sin marcar cómo es necesario llevarla a cabo: No es necesario cumplir el límite de potencia en los intervalos de bajo consumo, siempre que se cumpla que la potencia promedio total del sistema es menor a $1W$. También vale remarcar que la potencia pico medida durante el funcionamiento del sistema fue dos órdenes de magnitud inferior al límite especificado.

9.2. Validación del software

En el caso de las validaciones de software, se analizarán todas las especificaciones, realizando las validaciones correspondientes para cada caso.

1. Aplicación de Android para usuario: La aplicación ThingView cumple con la especificación de ser compatible con la versión *Lollipop*($v5{,}0/v5{,}1$) o posterior. Esta información puede verificarse en *https://play.google.com/store/ap*

2. Los datos relevados se pueden descargar por los usuarios autorizados, quienes poseen la contraseña de acceso, directamente de la página de ThingSpeak, que provee soporte para MATLAB. Además, se pueden exportar al entorno de MATLAB directamente con una función nativa del lenguaje (para la versión 2016 o posterior)*: thingSpeakRead(chId)*, donde chId es el número de identificación de la cuenta.

3. La página de ThingSpeak ofrece, para cada variable medida, una base de tiempo configurable por el usuario. El menú de configuración mencionado se muestra en la figura 37.

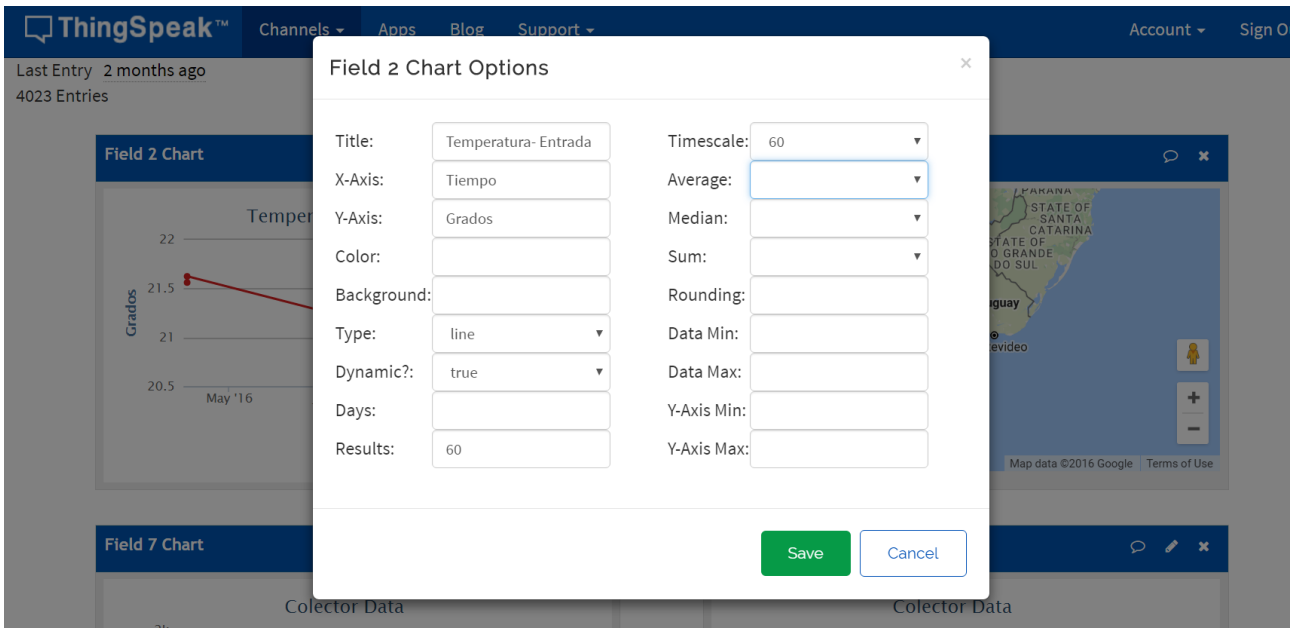


Figura 37: Configuración de la base de tiempos en la página de ThingSpeak

4. Almacenamiento remoto: El servicio de ThingSpeak almacena los datos sin limitación de capacidad según su política actual, como explica en su página de Internet. Para comprobar que la cuenta del servidor remoto puede modificarse mediante un mensaje SMS a la cuenta de celular asociada con el colector, se mandaron una serie de mensajes de prueba para configurar la contraseña del sistema (tabla 16).

Entradas	Respuesta correcta	Respuesta obtenida
API KEY: VXVFJ56XT07D9C06	API KEY cambiada a VXVFJ56XT07D9C06	API KEY cambiada a VXVFJ56XT07D9C06
API KEY VXVFJ56XT07D9C06	El sistema no debe responder a este mensaje.	No se obtuvo respuesta.
API KEY:VXVFJ56XT07D9C06	El sistema no debe responder a este mensaje.	No se obtuvo respuesta.
APIKEY: VXVFJ56XT07D9C06	El sistema no debe responder a este mensaje.	No se obtuvo respuesta.
API KEY: VAAFJAAXT0AA9CPP	API KEY cambiada a VAAFJAAXT0AA9C0PP	API KEY cambiada a VAAFJAAXT0AA9C0PP
APIKEY: VXVFJ56XT07D9C067	El sistema no debe responder a este mensaje.	No se obtuvo respuesta.
APIKEY: VXVFJ56XT07D9C0	El sistema no debe responder a este mensaje.	No se obtuvo respuesta.

Cuadro 16: Pruebas de configuración de la contraseña de ThingSpeak

Como las respuestas del sistema coincidieron con las respuestas esperadas, la prueba se considera aprobada.

5. Almacenamiento interno: Las mediciones realizadas durante las validaciones del hardware, se guardaron en el archivo *data.log* en la tarjeta SD en formato ASCII, con un formato fijo por medición, como se muestra en la figura 38. El archivo respeta el formato especificado en la ingeniería de detalle, por lo que la prueba se considera aprobada.

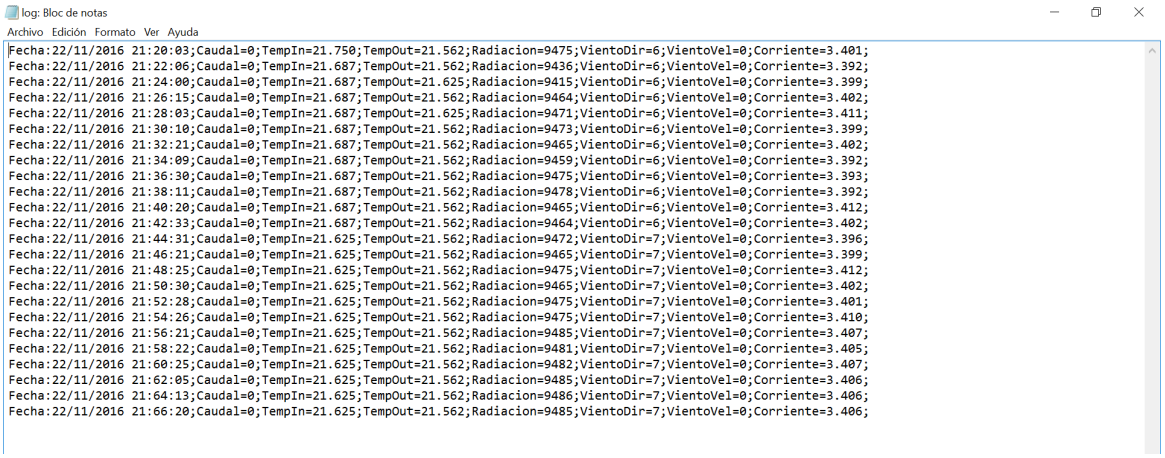


Figura 38: Extracto del archivo data.log almacenado en la tarjeta SD

6. En el servidor de ThingSpeak, el instante de cada medición se almacena con precisión de 1 segundo, como se muestra en la figura 39. Las mediciones almacenadas en la tarjeta SD también poseen una precisión de 1 segundo, como se observa en la figura 38.

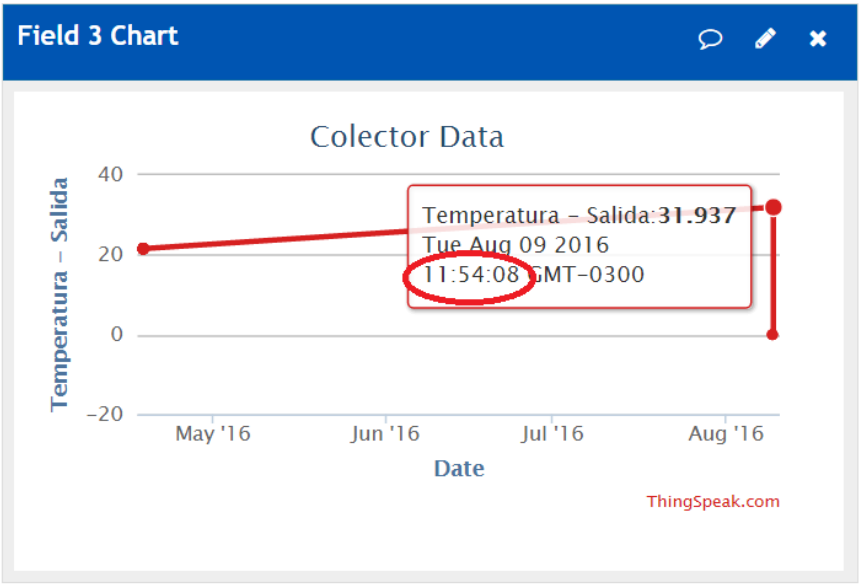


Figura 39: Precisión en las mediciones almacenadas en ThingSpeak

7. Relevamiento de datos: Deben realizarse los tests 11 y 15 de las validaciones de hardware para asegurar el cumplimiento de la especificación. En relación a la configuración de la frecuencia de mediciones, se debe validar las pruebas correspondientes a la tabla 17. Las respuestas del sistema coincidieron con las respuestas esperadas, por lo que la prueba se considera aprobada.

Entradas	Respuesta correcta	Respuesta obtenida
Periodo Mediciones: 2	Periodo cambiado a 2	Periodo cambiado a 2
Periodo Mediciones:2	El sistema no debe responder a este mensaje.	No se obtuvo respuesta.
PeriodoMediciones: 2	El sistema no debe responder a este mensaje.	No se obtuvo respuesta.
Periodo Mediciones:2	El sistema no debe responder a este mensaje.	No se obtuvo respuesta
Periodo mediciones: 2	El sistema no debe responder a este mensaje.	No se obtuvo respuesta
Periodo Mediciones: 2	El sistema no debe responder a este mensaje.	No se obtuvo respuesta
Periodo Mediciones: 1	El sistema no debe responder a este mensaje.	No se obtuvo respuesta
Periodo Mediciones: 1440	Periodo cambiado a 1440	Periodo cambiado a 1440
Periodo Mediciones: 1441	El sistema no debe responder a este mensaje.	No se obtuvo respuesta

Cuadro 17: Pruebas de configuración del período de mediciones

8. Contraseñas: los datos de almacenados en el servidor remoto se encuentran protegidos por una contraseña que es configurable en la página de ThingSpeak, con un email de referencia de seguridad, como muestra la figura 40. Además, cada canal de datos puede configurarse como público o privado: Los canales públicos pueden ser accedidos por cualquier usuario, mientras que los canales privados solo brindan acceso a sus creadores. De esta forma, el usuario posee flexibilidad para compartir o no sus datos.

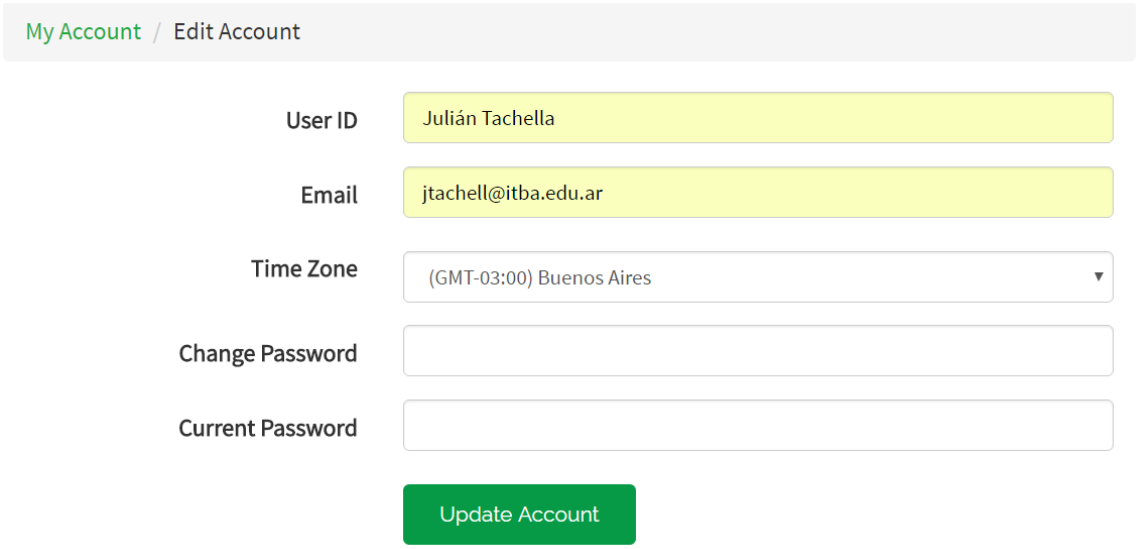


Figura 40: Extracto del archivo data.log almacenado en la tarjeta SD

9. Documentación necesaria: Todas las rutinas tienen un comentario al comienzo que explique sus entradas, sus salidas y sus restricciones, como se puede observar en el código fuente adjuntado en el anexo.
10. Sensores: El test 22 de hardware (ver 9.1.4) asegura el funcionamiento independiente de cada sensor.

10. Estudios de confiabilidad de hardware y software

10.1. Confiabilidad de hardware

La confiabilidad de hardware se calculó siguiendo el método de las cargas de la norma HDBK 217. En este marco, la tasa de fallas de cada elemento λ_{item} se calcula siguiendo la ecuación 3, donde λ_0 es la tasa de fallas base y π_j son los multiplicadores que dependen de las condiciones de uso del elemento. Cuando los elementos individuales se encuentran asociados en serie en cuanto a su confiabilidad, la tasa de falla del sistema que los integra viene dado por la ecuación 4. Las fallas consideradas en este análisis son catastróficas, ya que son repentinas, completas y definitivas en el elemento que las sufras. Una falla es considerada cuando implica el no cumplimiento de una especificación de diseño o requerimiento del cliente. Finalmente, una vez calculada la tasa de fallas del sistema $\lambda_{sistema}$, el tiempo a la falla $MTTF$ viene dado por la ecuación 5.

$$\lambda_{item} = \lambda_0 \prod_{j=1}^{C} \pi_j \tag{3}$$

$$\lambda_{serie} = \sum_{i=1}^{n} \lambda_i \tag{4}$$

$$MTTF = \frac{1}{\lambda_{sistema}} \tag{5}$$

Dada la complejidad del diseño, se tomó una estrategia *top-down*, partiendo de la asociación de los módulos más generales, y luego desglosando cada uno en submódulos cuya tasa de fallas es más simple de calcular. La figura 41 muestra el esquema de más alto nivel, donde los módulos principales del sistema se encuentran asociados en serie, debido a que la falla de cualquiera implica el no cumplimiento de los requerimientos del cliente. En las subsecciones posteriores se analiza cada módulo en detalle.

Figura 41: Diagrama de confiabilidad del sistema

10.1.1. Fuentes de alimentación

El sistema cuenta con 3 fuentes de alimentación: La fuente switching que recibe la tensión de la red eléctrica, el regulador lineal con salida de $5V$ y el regulador lineal con salida de$3{,}3V$. La falla de cualquiera de estas fuentes implica la falla completa del sistema: Si la fuente switching falla, los demás módulos no reciben tensión de alimentación. Si el regulador de salida de $5V$ falla, el módulo de comunicaciones deja de funcionar, y si el regulador de salida de $3{,}3V$ falla, el microcontrolador deja de funcionar y por ende no se registran nuevas mediciones. Para los capcitores se utilizó la aproximación de tecnología tantalio por falta de capacitores multicapa, ya que es sabido que el desempeño del capcitor de tantalio es peor lo que terminaría dando una cota de seguridad para la confiabilidad.

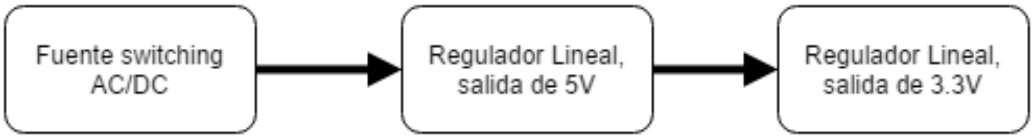


Figura 42: Diagrama de confiabilidad de las fuentes de alimentación

Fuente switching La fuente switching no es un diseño propio, pero su fabricante no ofrece datos de confiabilidad. Por esta razón, a continuación se realiza una estimación de confiabilidad, analizando sus partes principales. La figura 43 muestra los elementos principales de una fuente switching, que se encuentran asociados en serie en relación a su confiabilidad.

El rectificador a la entrada de la fuente viene dada por un puente de diodos y un capacitor, y su tasa de fallas puede calcularse como $\lambda_{rect} = \lambda_{diodo} + \lambda_{cap} = 0{,}0147 \frac{fallas}{10^6 hrs}$.

Parámetro	Valor	Justificación
λ_0	0.001	El diodo se utiliza para una aplicación de switching.
π_T	1.4	La temperatura de juntura es aproximadamente 40°C
π_S	0.29	La tensión de estrés que sufre el diodo $V_S = \frac{VoltajeAplicado}{VoltajeMaximo} = 0{,}6$
π_C	1	Factor constructivo
π_Q	5.5	Factor de calidad del LOWER encapsulado
π_E	6	Factor de ambiente, Ground Fixed.

(a) Diodo

Parámetro	Valor	Justificación
λ_0	0.012	El capacitor es electrolítico de tantalio. Se utiliza con un estrés de 0.5 a aprox. 30°C.
π_{CV}	1.9	Se utiliza de capacitancia aproximadamente $210\mu C$
π_S	0.29	La tensión de estrés que sufre el diodo $V_S = \frac{VoltajeAplicado}{VoltajeMaximo} = 0{,}6$
π_{SR}	0.66	Resistencia parásita serie mayor a $0{,}8\Omega$
π_Q	1.5	Factor de calidad L del encapsulado
π_E	2	Factor de ambiente, Ground Fixed.

(b) Capacitor

Cuadro 18: Tasa de fallas diodo y capacitor del rectificador en la entrada

Los interruptores de potencia vienen dados por un mosfet, y su tasa de fallas puede calcularse como $\lambda_{int} = \lambda_b \pi_T \pi_A \pi_R \pi_S \pi_Q \pi_E = 0{,}0215 \frac{fallas}{10^6 hrs}$

Parámetro	Valor	Justificación
λ_b	0.00074	Valor base estipulado
π_T	2.1	Peor caso de temperatura estipulado el cual es $T = 60$°C
π_A	0.7	Aplicación switching
π_R	1.8	Potencia en watts consumida por la fuente, en base a especificación
π_S	1	Estrés voltaíco
π_Q	5.5	Lower
π_E	2	Fijo a tierra

Cuadro 19: Transitores de potencia

Para el transformador de potencia se calculó utilziando la siguiente formula del manual HDBK 217: $\lambda_{tr} = \lambda_b \pi_E \pi_Q = 0{,}288 \frac{fallas}{10^6 hrs}$

Parámetro	Valor	Justificación
λ_b	0.006	Valor base
π_Q	8	Transformador de potencia
π_E	6	Fijo a tierra

Cuadro 20: Transformador

El rectificador de salida es igualado al de entrada y el filtro LC es un capictor en serie con una bobina, de forma tal que su confiabilidad es: $\lambda = 0{,}0371 \frac{fallas}{10^6 hrs}$

Parámetro	Valor	Justificación
λ_0	0.001	El diodo se utiliza para una aplicación de switching.
π_C	1	Factor constructivo
π_Q	4	Factor de calidad del MIL-C-15305
π_E	6	Factor de ambiente, Ground Fixed.

(a) Inductor

Parámetro	Valor	Justificación
λ_0	0.012	El capacitor es electrolítico de tantalio. Se utiliza con un estrés de 0.5 a aprox. 30°C.
π_{CV}	1.9	Se utiliza de capacitancia aproximadamente $210\mu C$
π_S	0.29	La tensión de estrés que sufre el diodo $V_S = \frac{VoltajeAplicado}{VoltajeMaximo} = 0{,}6$
π_{SR}	0.66	Resistencia parásita serie mayor a $0{,}8\Omega$
π_Q	1.5	Factor de calidad L del encapsulado
π_E	2	Factor de ambiente, Ground Fixed.

(b) Capacitor

Cuadro 21: Tasa de fallas diodo y capacitor del rectificador en la entrada

Esto da una confiabilidad total de $\lambda_T = 0{,}3681 \frac{fallas}{10^6 hrs}$

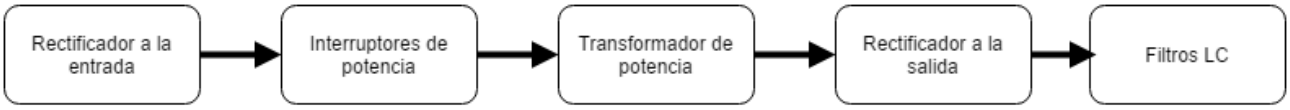


Figura 43: Diagrama de confiabilidad de la fuente switching

10.1.2. Microcontrolador

La confiabilidad del microcontrolador, esquema 44, puede estimarse como 2 elementos en serie, la electrónica digital del microcontrolador y la memoria flash interna que contiene el programa. La tasa de fallas de la electrónica del microcontrolador se estima según la ecuación 44, y los parámetros se resumen en la tabla a continuación:

$$\lambda_{micro} = \lambda_{BD}\pi_{MFG}\pi_T\pi_{CD} + \lambda_{BP}\pi_E\pi_Q\pi_{PT} + \lambda_{EOS} = 0{,}1266\frac{fallas}{10^6 hrs} \tag{6}$$

Parámetro	Valor	Justificación
λ_{BD}	0.16	El circuito es un microcontrolador con más de 60000 compuertas.
π_{MFG}	2.0	El fabricante no es QML o QPL.
π_T	0.63	Para el peor caso estipulado, $T = 60C$
π_{CD}	0.117	Calculado según HDBK para las dimensiones del ARM Cortex M0
π_E	2	Fijo a tierra
π_{PT}	2.2	Grid Array Hermetico
π_Q	2.4	JAN
λ_{EOS}	0.065	Vth desconocido
λ_{BP}	0.0036	Cantidad de pines

Cuadro 22: Diodo

Memoria interna Para la memoria flash interna se calcúlo utilizando la siguiente formula

$$\lambda_{flash} = (C_1\pi_T + C_2\pi_E + \lambda_{cyc})\pi_Q\pi_L = 0{,}025\frac{fallas}{10^6 hrs}$$

Parámetro	Valor1	Justificación
C_1	0.0068	Valor según memoria
λ_{cyc}	0	Por tecnología de fabricación
π_T	0.63	Peor caso de temperatura esperado $T = 60°C$
π_Q	0.25	Clase S
C_2	0.0041	Número funcionales de pines
π_E	2	Fijo a tierra
π_L	1	Años de producción

Cuadro 23: Amperimetro

Confiabilidad total Sumando los elementos en serie se obtiene la siguiente confiabilidad total

$$\lambda_{microcontrolador} = 0{,}025 + 0{,}1266 = 0{,}1516 \frac{fallas}{10^6 hrs}$$

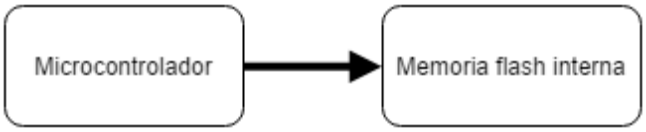


Figura 44: Diagrama de confiabilidad del microcontrolador

10.1.3. Sensores

La confiabilidad de los sensores, esquema 45, puede estimarse como todos los sensores en serie, pues para tener un dato valido es necesaria la presencia de la información relevada por todos estos. Para cada sensor se calculó su MTTF de forma estimada, asimilandolos a componentes del manual HDBK-217. Para el caso de los sensores de temperatura, se posee un reporte de confiabilidad del productor.

Para el sensor de radiación se lo estimo como un foto-diodo, cuya confiabilidad se calcula con:

$$\lambda_{rad} = \lambda_d \pi_T \pi_Q \pi_E = 0{,}1728 \frac{fallas}{10^6 hrs}$$

Parámetro	Valor	Justificación
λ_d	0.004	Es un foto diodo
π_T	2.7	Peor caso de temperatura estipulado el cual es $T = 60°C$
π_Q	8	Es plástico
π_E	2	Fijo a tierra

Cuadro 24: Foto diodo y led

Caudal Para el caudalímetro se usa tanto un foto-diodo para detectar y un led para emitir, ambos se calculan de la misma forma y se consideran en serie para el funcionamiento:

$$\lambda_{caudal} = \lambda_{b1} \pi_{T1} \pi_{Q1} \pi_{E1} + \lambda_{b2} \pi_{T2} \pi_{Q2} \pi_{E2} = 0{,}1827 \frac{fallas}{10^6 hrs}$$

Parámetro	Valor1	Valor 2	Justificación
λ_d	0.004	0.00023	Valor de foto diodo y led
π_T	2.7	2.7	Peor caso de temperatura esperado $T = 60°C$
π_Q	8	8	Plástico
π_E	2	2	Fijo a tierra

Cuadro 25: Diodo

Sensor de viento Para el sensor de viento se utilizan los mismos componentes que el caudalímetro para la velocidad y una resistencia variable para la dirección, por lo que se tiene:

$$\lambda_{viento} = \lambda_{b1}\pi_{T1}\pi_{Q1}\pi_{E1} + \lambda_{b2}\pi_{T2}\pi_{Q2}\pi_{E2} + \lambda_{br}\pi_{TAPS}\pi_R\pi_V\pi_Q\pi_E = 9{,}646\frac{fallas}{10^6 hrs}$$

Parámetro	Valor1	Valor 2	Justificación
λ_d	0.004	0.00023	Valor de foto diodo y led
π_T	2.7	2.7	Peor caso de temperatura esperado $T = 60°C$
π_Q	8	8	Plástico
π_E	2	2	Fijo a tierra

Cuadro 26: Fotodiodo y LED

Parámetro	Valor	Justificación
λ_{br}	0.096	Valor de falla base, para el peor caso de temperatura
π_{TAPS}	4.4	En base al número de vueltas
π_R	2.0	Factor de resistencia
π_V	1.4	Factor de voltaje en base a la tensión aplicada
π_Q	4	Factor de calidad
π_E	2	Fijo a tierra

Cuadro 27: Resistencia variable

Ampérimetro Para el amperímetro se lo toma como un IC de mas de 100 transistores y menos de 100:

$$\lambda = (C_1\pi_T + C_2\pi_E)\pi_Q\pi_L = 0{,}0017\frac{fallas}{10^6 hrs}$$

Parámetro	Valor1	Justificación
C_1	0.0025	Valor según compuertas
π_T	0.63	Peor caso de temperatura esperado $T = 60°C$
π_Q	0.25	Clase S
C_2	0.0026	Dip sellado de 8 pines
π_E	2	Fijo a tierra
π_L	1	Años de producción

Cuadro 28: Amperimetro

Sensor de temperatura Según la hoja de datos (ver anexo), los sensores de temperatura poseen un MTTF de 25120 años.

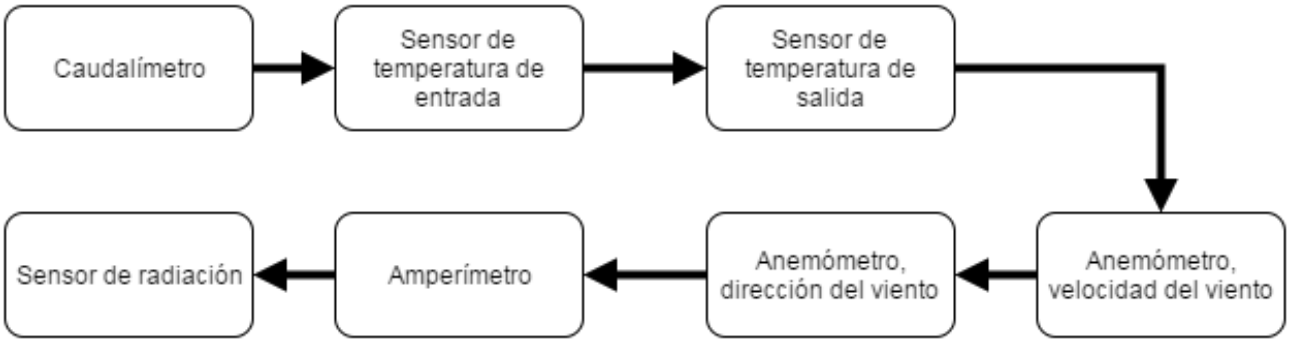


Figura 45: Diagrama de confiabilidad del sistema

Total Habiendo calculado todas las tasas de falla λ, es posible estimar la confiabilidad del conjunto de sensores:

$$\lambda_{sensores} = \sum \lambda_i = 10{,}0032\frac{fallas}{10^6 hrs}$$

10.1.4. Módulo GPRS

El modulo GPRS se puede representar mediante una conexión en serie del SIM900 y la antena.
El SIM900 es un microcontrolador, por lo que se puede calcular su confiabilidad como:

$$\lambda = (C_1\pi_T + C_2\pi_E)\pi_Q\pi_L = 0{,}0027\frac{fallas}{10^6 hrs}$$

Parámetro	Valor1	Justificación
C_1	0.004	Valor según compuertas
π_T	0.63	Peor caso de temperatura esperado $T = 60°C$
π_Q	0.25	Clase S
C_2	0.0026	Dip sellado de 8 pines
π_E	2	Fijo a tierra
π_L	1	Años de producción

Cuadro 29: Amperimetro

Para la antena se buscó ejemplos de calculos de confiabilidad similares que usasen el mismo manual para el calcúlo de la confiabilidad. Ya que no hay un objeto determinado como antena lo mas próximo a esta es un inductor, por lo que se utilizó este como modelo de fallas:

$$\lambda_p = \lambda_b\pi_C\pi_Q\pi_E = 0{,}0088\frac{fallas}{10^6 hrs}$$

Parámetro	Valor1	Justificación
λ_b	0.0011	Valor base
π_C	1	Factor de construcción fijo
π_Q	4	Por ser usada en RF
π_E	2	Fijo a tierra

Cuadro 30: Amperimetro

Poniendo en serie ambos sistema se obtiene:

$$\lambda = 0{,}0088 + 0{,}0027 = 0{,}0115\frac{fallas}{10^6 hrs}$$

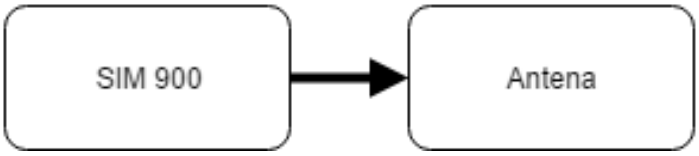


Figura 46: Diagrama de confiabilidad del módulo GPRS

10.1.5. Memoria interna

La memoria esta especificada en el manual HDBK 217 para ser calculada como:

$$\lambda = (C_1\pi_T + C_2\pi_E + \lambda_{cyc})\pi_Q\pi_L = 0{,}025\frac{fallas}{10^6 hrs}$$

Parámetro	Valor1	Justificación
C_1	0.0068	Valor según memoria
λ_{cyc}	0	Por tecnología de fabricación
π_T	0.63	Peor caso de temperatura esperado $T = 60°C$
π_Q	0.25	Clase S
C_2	0.0041	Número funcionales de pines
π_E	2	Fijo a tierra
π_L	1	Años de producción

Cuadro 31: Amperimetro

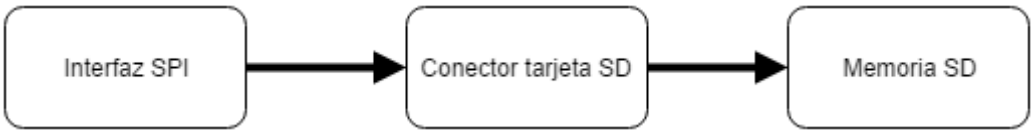


Figura 47: Diagrama de confiabilidad del sistema

10.1.6. Confiabilidad total

Sumando las tasas de fallas de cada módulo, se obtiene una confiabilidad de:

$$\lambda_{sistema} = 10{,}771 \frac{fallas}{10^6 hrs}$$

La tasa de fallas obtenida equivale a un $MTTF = 93362hrs$, cuyo valor aproximado es de diez años. Este valor se encuentra por debajo de los 2 años especificados de vida útil. Utilizando la fórmula 7, se obtiene una probabilidad de 0.83 de que el sistema funcione sin fallas durante 2 años. Si se considera la antigüedad de la norma HDBK, se puede dividir la tasa de fallas obtenida por 4 debido a que la confiabilidad de los componentes actuales es superior a la confiabilidad estimada por la norma HDBK. Aplicando este factor de corrección, se obtiene una probabilidad de 0.96 de que el sistema no sufra fallas en el intervalo de 2 años.

$$R_{sistema}(t) = e^{-\lambda_{sistema} t} \tag{7}$$

Cabe destacar que casi el 90 % de la confiabilidad del sistema yace en el anemómetro, específicamente en el sensor encargado de la dirección del viento. Si se quisiese mejorar la confiabilidad del sistema, sería necesario modificar la tecnología de este sensor u ofrecer un servicio de reemplazo para el mismo.

10.2. Confiabilidad de software

Se realizó un análisis de confiabilidad de software basado en el método de Schooman que se encuentra detallado en su libro *Reliability of Computer Systems and Networks*[1]. Dicho modelo que toma las siguientes suposiciones:

- Una mayor cantidad de errores residuales del software generan una mayor cantidad de fallas de software.
- La tasa de descubrimiento de errores residuales puede ser modelado como un proceso estocástico que depende de las entradas y condiciones iniciales del sistema.
- La tasa de descubrimiento de errores es proporcional al número de errores residuales.

La tasa de errores residuales del software al instante τ, $\varepsilon_r(\tau)$, se modela con la ecuación 8, donde E_T es la cantidad inicial de errores, $\varepsilon_r(\tau)$ es la tasa de corrección de errores e I_T es la cantidad de instrucciones del programa.

$$\varepsilon_r(\tau) = \frac{E_T}{I_T} - \varepsilon_c(\tau) \tag{8}$$

Siguiendo la suposición de que la tasa de fallas es proporcional a los errores residuales, se llega a la ecuación 9, donde k es una constante de proporcionalidad. Además, como el modelo es exponencial, la tiempo medio entre fallas viene dado por la inversa de la tasa de fallas: $MTBF = \frac{1}{\lambda}$.

$$\lambda = k(\frac{E_T}{I_0} - \varepsilon_c(\tau)) \tag{9}$$

La estimación de los parámetros del modelo se puede realizar utilizando información de desarrollos pasados o con el método de los momentos. Debido a que este proyecto es el primer desarrollo importante que llevaron a cabo los alumnos, no habían antecedentes disponibles y se debió utilizar el método de los momentos. El método de los momentos consiste en estimar los parámetros mediante los momentos estadísticos de primer y segundo orden. A partir de la expresión del MTBF para 2 períodos de testeo del software distintos, ecuaciones 10 y 11, se puede resolver un sistema de ecuaciones para obtener estimaciones de k y E_T (ecuaciones 12 y 13).

$$MTBF_a = \frac{1}{\lambda_a} = \frac{1}{k(\frac{E_T}{I_0} - \varepsilon_c(\tau_a))} \tag{10}$$

$$MTBF_b = \frac{1}{\lambda_b} = \frac{1}{k(\frac{E_T}{I_0} - \varepsilon_c(\tau_b))} \tag{11}$$

$$\hat{E}_T = -\frac{\frac{\lambda_b}{\lambda_a}\varepsilon_c(\tau_a)I_T - \varepsilon_c(\tau_b)I_T}{\frac{\lambda_b}{\lambda_a} - 1} \tag{12}$$

$$\hat{k} = \lambda_a \frac{I_T}{\hat{E}_T - \varepsilon_c(\tau_a)I_T} \tag{13}$$

El software del sistema fue testeado durante dos semanas en la fase de validación del prototipo. Durante ese período, el programa se corrió n veces, y se registraron la cantidad de veces que hubieron fallas del software (r), las horas de operación hasta la falla (t_i), las horas de operación de las corridas sin errores (T_i) y los errores corregidos en ese intervalo, $E_c(\tau_i)$. Los datos obtenidos se muestran en la tabla , donde la cantidad total de horas de funcionamiento viene dada por la suma de las horas de operación hasta la falla y las horas de operación sin fallas, como muestra la ecuación 14. Además, la estimación de la tasa de fallas se calculó siguiendo la ecuación 15.

$$H_i = \sum_{i=1}^{n-r} T_i + \sum_{i=1}^{r} t_i \tag{14}$$

$$\lambda_i = \frac{r}{H_i} \tag{15}$$

Parámetro	Valor
n[veces]	23
r[veces]	2
H_a[horas]	296
$\lambda_a[\frac{fallas}{hora}]$	0,006756
$E_c(\tau_a)$[correcciones]	10

(a) Fallas detectadas durante la primera semana

Parámetro	Valor
n[veces]	26
r[veces]	1
H_b[horas]	306
$\lambda_b[\frac{fallas}{hora}]$	0,003267
$E_c(\tau_b)$[correcciones]	21

(b) Fallas detectadas durante la segunda semana

Cuadro 32: Fallas registradas durante el período de prueba de software

Considerando que el programa tiene 1322 líneas de código en C/C++, que equivalen aproximadamente a $I_T = 2{,}5{*}1322 = 3305$ instrucciones de máquina y que la tasa de errores corregidos se calcula como $\varepsilon_c(\tau) = \frac{E_c(\tau)}{I_T}$, se estimaron los parámetros E_T y k (ver tabla 33). Sustituyendo los valores obtenidos en la ecuación 9, se obtuvo un MTBF de 3256 horas.

$\hat{k}$	$\hat{E}_T$
1.0149	32

Cuadro 33: Estimación de los parámetros E_T y k

Si se tiene en cuenta que el sistema realiza un $soft-reset$ al pasar al modo de bajo consumo, la probabilidad de falla del software disminuye considerablemente, debido a que los registros del microcontrolador se reinician en cada $soft-reset$.

11. Conclusiones

Este trabajo desarrolla detalladamente el diseño de un sistema de medición de las variables de interés de colectores solares térmicos. Los requerimientos del cliente (FOVISEE y el ITBA), fueron traducidos a especificaciones de diseño concretas, que luego se transformaron en decisiones de ingeniería de detalle. Finalmente se planearon y ejecutaron pruebas de validación del prototipo, que fueron aprobadas con buen margen, asegurando el cumplimiento de los requerimientos del cliente. Considerando que el producto tiene un único cliente, las especificaciones del prototipo coinciden con los resultados de las pruebas de validación. A continuación se presenta la matriz de trazabilidad final.

Req. cliente	Nro. de esp.	Nombre de especificación	Test ID prototipo	Resultado Test prototipo	Test ID producto final
Especificaciones de Hardware					
13	1	Tensión de alimentación	1, 2	Aprobada con una tensión máxima de $248V_{rms}$ y mínima de$164V_{rms}$.	1, 2
11	2	Grados de protección $IEC529$	-		3
11	3	Temperaturas	-		4
3	4	Medición de intensidad solar	6	Aprobada. Las mediciones de los sensores variaron $16\frac{W}{m^2}$, el intervalo mínimo fue de 1minuto y 41 segundos y el intervalo máximo de 2 minutos y 20 segundos. La memoria interna y el servidor remoto almacenaron las mismas mediciones.	5
2	5	Medición de temperatura del agua de entrada y salida del colector solar	8, 9	Aprobada. El sensor de temperatura de entrada presenta un margen de 1,0625ºC y el sensor de temperatura de salida presenta un margen de 1,1250ºC.	7
1	6	Medición de caudal de agua	10	Aprobado. Mediciones de caudal tienen una precisión de $0{,}24\frac{litros}{min}$. El LED se cambió de color rojo a verde en menos de 1 segundo.	10
4	7	Medición del viento	13, 14	Aprobada con un margen de $1{,}81\frac{km}{hora}$.	12, 14
5	8	Medición de consumo del termotanque	6	Aprobada con un margen de $0{,}17A_{rms}$.	6
9	9	Conexión GPRS	19	Aprobada	19
12	10	Memoria Interna	6, 24	Aprobada con un margen de 6GBy.	6, 24
6	11	Potencia	19	Aprobada. La potencia media es inferior a $1W$ con un margen de $265mW$. (Ver 9.1.5)	19
propio	12	Etiquetado	23	Aprobada	23
11	13	Vida útil	-		18
8	14	Precio al consumidor	Ver documentación 6.3		Ver documentación 6.3
Regulación	15	EMI	-		16
Regulación	16	EMS	-		17
11	17	Peso	20	Aprobada con un margen de $8{,}5kg$.	20
11	18	Dimensiones	21	Aprobada con un margen de $28{,}9cm$ para el largo, $33{,}5cm$ para el ancho y $20{,}8cm$ para el alto.	21

Req. cliente	Nro. de esp.	Nombre de especificación	Test ID prototipo	Resultado Test prototipo	Test ID producto final
14	19	Distancia máxima sensores	22	Aprobada.	22
propio	20	LED indicador de estado	11	Aprobada.	10
13	21	Arranque del sistema	1, 2	Aprobada.	1, 2
10	22	Funcionamiento	19	Aprobada.	19
13	23	Apagado del sistema	1, 2	Aprobada.	1, 2
propio	24	Documentación hardware	Ver secciones 7 y 8		Ver sección 7
Especificaciones de Software					
10	1	Aplicación de Android para usuario	1	Aprobado.	1
9	2	Tratamiento de las mediciones	2	Aprobado.	2
10	3	Presentación de las mediciones	2	Aprobado.	2
9	4	Almacenamiento remoto	4	Aprobado.	4
12	5	Almacenamiento interno	5	Aprobado.	5
7	6	Precisión del instante de medición	3	Aprobado con precisión de 1 segundo.	3
7	7	Relevamiento de datos	7	Aprobado.	7
10	8	Contraseñas	8	Aprobado.	8
propio	9	Documentación de software	9	Ver anexo 12.4	
propio	10	Sensores	10	Aprobado con un margen de $0{,}46 A_{rms}$ para el amperímetro, $4{,}48 \frac{km}{hora}$ y 1 octante para el anemómetro, $0{,}2 \frac{litros}{min}$ para el caudalímetro, $40{,}90 \frac{W}{m^2}$ para el sensor de radiación y 0,4875°C para los sensores de temperatura.	10

La experiencia de realizar el trabajo produjo varios aprendizajes, que pueden dividirse en técnicos y generales. Los aprendizajes técnicos se refieren al *know-how* obtenido en los campo de colectores solares e Internet of Things, mientras que los aprendizajes generales giran en torno a los errores y aciertos de la primera experiencia en el diseño profesional de un producto. Las próximas subsecciones analizan ambos tipos de aprendizajes.

11.1. Excelencias. Objetivos alcanzados

El prototipo fabricado fue validado integralmente y es apto para la instalación en un colector solar térmico. Esta primera versión es fundamental para promover el avance del proyecto social del ITBA y FOVISEE, y mantener el apoyo económico del banco Santander Río. Además, en el trabajo se analizaron los costos, modificaciones y pruebas necesarias para el producto final, por lo que la primera tira de productos se puede llevar a cabo fácilmente. El desarrollo del prototipo se realizó con un costo total de 3730 pesos, por debajo de la mitad del presupuesto recibido de los clientes (8000 pesos), lo que deja un importante excedente que puede destinarse a la producción de las primeras unidades del producto final.

En relación a los aprendizajes técnicos, el microcontrolador elegido (KL25Z) resultó ser muy flexible, de bajo costo y apto para desarrollos rápidos. Además, el servicio técnico de su fabricante respondió a las consultas en

tiempo y forma, por lo que se recomienda esta opción para futuros diseños. Los sensores elegidos demostraron tener una precisión superior o similar a sus valores de hojas de datos, y también se recomiendan para futuros diseños. Este proyecto sirvió como una muy buena introducción al diseño de dispositivos dedicados a Internet Of Things, que se encuentra en constante auge en la actualidad.

En cuanto a los aprendizajes generales, los alumnos aprendieron la importancia de detallar correctamente los requerimientos y especificaciones de un producto, a diagramar y realizar pruebas de validación y a realizar el planeamiento económico y de tiempos de un proyecto de tamaño significativamente mayor al resto de los trabajos de la carrera. Este aprendizaje es fundamental para el futuro desempeño de los alumnos en el mundo profesional. El proyecto fue muy útil para entender las diferencias entre cumplir con un trabajo para un profesor y cumplir con un trabajo para un cliente (incluyendo al Estado y la sociedad).

11.2. Fallos. Recomendaciones para futuros diseños

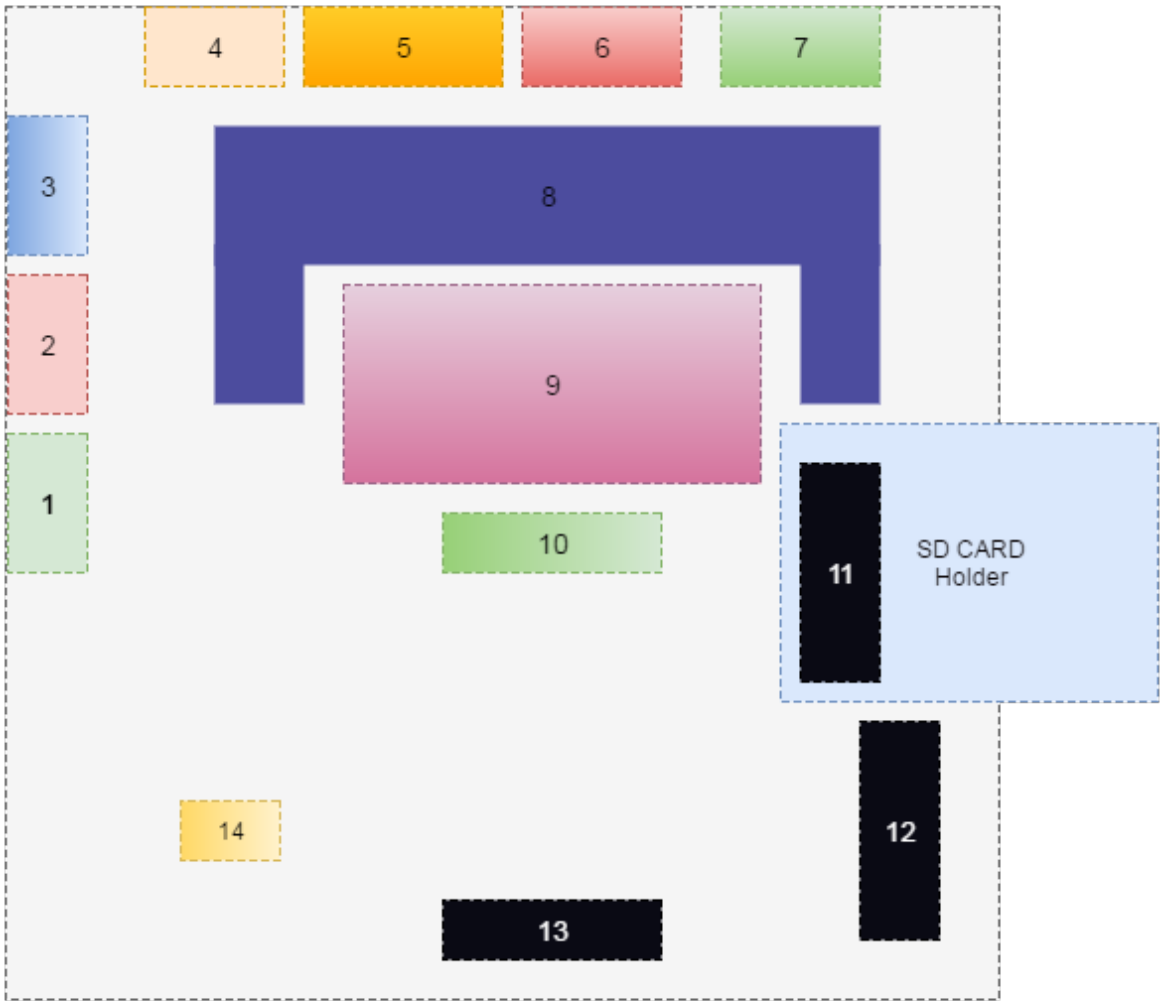
En relación a los aprendizajes técnicos, el proceso de diseño no tuvo en cuenta el arranque del circuito y el prototipo exhibió una corriente de inrush superior al límite especificado. Sin embargo, se halló que la causa del problema era la fuente switching, que no poseía un termistor NTC. Una vez reemplazada la fuente, el pico de corriente disminuyó a un valor que cumple las especificaciones de diseño. También se midió una potencia superior a la especificada en los intervalos de bajo consumo, pero se pudo hallar la causa del exceso para considerar en futuros diseños. Se encontró una falencia en dicha especificación, ya que ésta no tendría que especificar la forma en que debería cumplirse el límite de consumo. Esta especificación fue escrita erróneamente pensando al sistema como caja blanca y no como caja negra. La elección del módulo de comunicaciones no se recomienda para futuros diseños, no sólo por la desactualización de la tecnología GPRS, sino también por la complejidad involucrada en el manejo del módulo SIM900.

En cuanto a los aprendizajes generales, el equipo debió adaptarse a un trabajo cuyas premisas presentaron mucha varianza. En primer lugar, fue encargado de realizar un prototipo rápido para presentar al banco Santander en abril del 2016, que lo obligó a llevar a cabo un desarrollo ágil sin tener requerimientos y objetivos claros. Los alumnos debieron trabajar arduamente durante enero y febrero para entregar la versión a tiempo. Una vez terminado el prototipo, el plan del proyecto debió modificarse para respetar un esquema de requerimientos y especificaciones más puntuales, que no eran cumplidos con rigor inicialmente, por lo que el equipo debió adaptar el prototipo a dicho esquema. El trabajo fue más complejo de lo esperado debido a la deuda técnica del prototipo inicial.

Los alumnos que trabajaron en este proyecto esperan que pueda ser continuado por otras generaciones de ingenieros del ITBA y que brinde un aporte al desarrollo de sectores de bajos recursos y la promoción de alternativas renovables.

12. Anexos

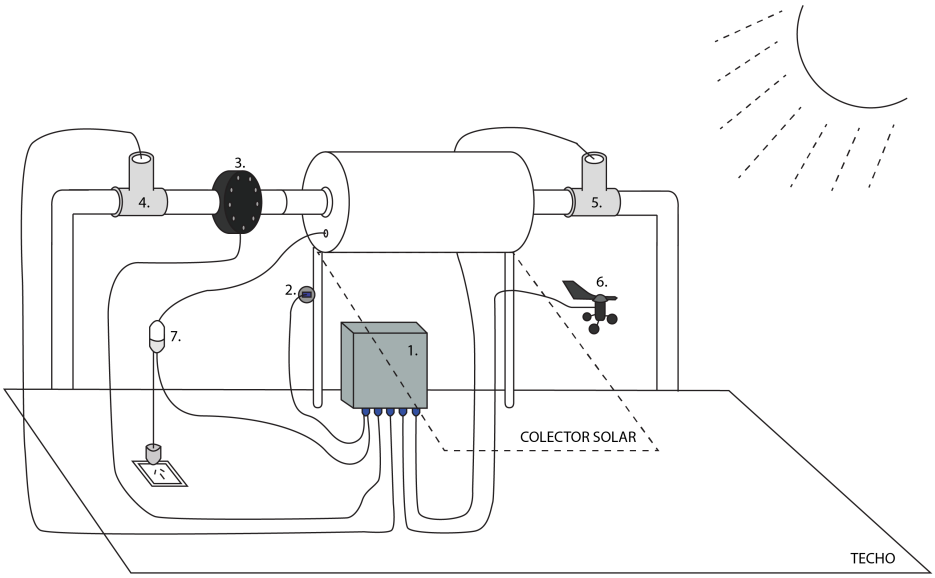
12.1. Planos



1-Bornera caudalímetro
2 y 3 - Bornera sensor de radiación
4-Bornera sensor de temperatura entrada
5-Bornera sensor de temperatura salida
6 y 7-Bornera anemómetro
8 - Regulador lineal
9- Conversor de nivel
10 - Bornera modulo GPRS
11- Conexión tarjeta SD
12 Conexión RTC - Opcional
13-Bornera Amperímetro
14-Jumper alimentación microcontrolador

Figura 48: Placa Shield

12.2. Esquemas



1. Gabinete
2. Sensor de radiación
3. Caudalímetro
4. Sensor de temperatura de entrada
5. Sensor de temperatura de salida
6. Anemómetro
7. Amperímetro

Figura 49: Esquema de conexión del sistema

12.2.1. Anexo Shield

Todos los esquemáticos de cada una de los bloques verdes que componen el esquemático original se puede visualizar en la figura 50.

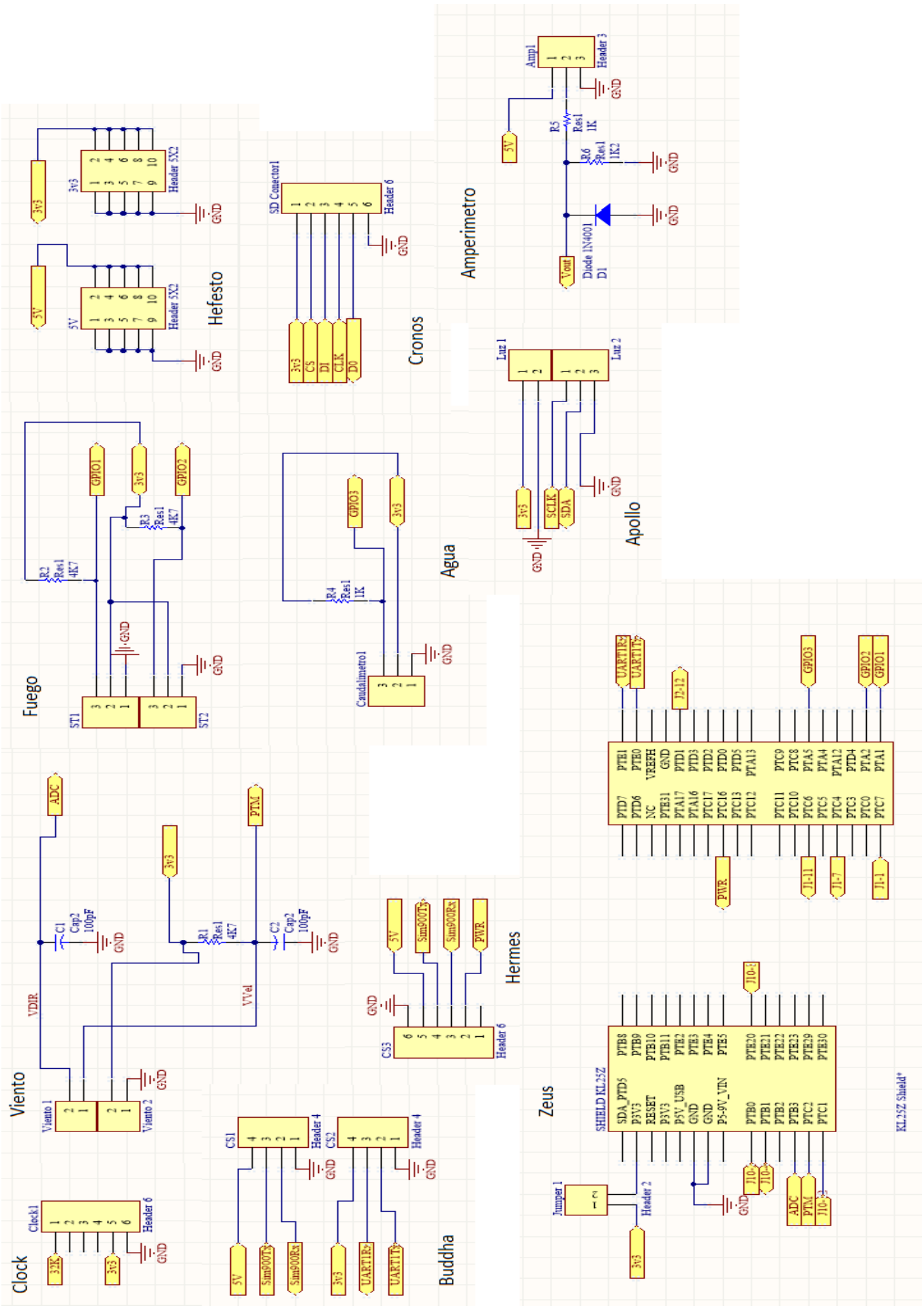


Figura 50: Esquemáticos internos

12.3. Listado de partes

12.3.1. Sensores

- Sensor de temperatura sumergible DS18B20 x2
- Caudalímetro YF G1 x1
- Anemómetro Davis x1
- Amperímetro ACS712 x1
- Sensor de radiación BH1750VHI x1

12.3.2. PCB

- EB KL25Z
- Placa Shield
- Placa conectora con bornes
- Adaptador de nivel
- Regulador lineal 3.3V / 5V

12.3.3. Miscelaneo

- Fuente de alimentación 220Vrms 50Hz a 9V DC
- Cableado para cada sensor
- Gabinete de protección

12.4. Códigos del software

'

```
/* ###################################################################
**     Filename    : main.c
**     Project     : ColectorSolar
**     Processor   : MKL25Z128VLK4
**     Version     : Driver 01.01
**     Compiler    : GNU C Compiler
**     Date/Time   : 2015-12-30, 10:26, # CodeGen: 0
** ###################################################################*/
/*!
** @file main.c
/* MODULE main */


/* Including needed modules to compile this module/procedure */
#include "Cpu.h"
#include "Events.h"
#include "clockMan1.h"
#include "pin_init.h"
#include "osa1.h"
#include "gpio1.h"
#include "tpmTmr1.h"
#include "adConv1.h"
#include "uartCom1.h"
#include "i2cCom1.h"
#include "memoryCard1.h"
#include "fsl_spi1.h"
#include "rtcTimer1.h"
#include "lpTmr1.h"
#include "pwrMan1.h"
#include "temperatura.h"
#include "AppRun.h"
#include "Measure.h"
#include "Radiacion.h"

#if CPU_INIT_CONFIG
  #include "Init_Config.h"
#endif
/* User includes (#include below this line is not maintained by Processor Expert) */
#include "fsl_smc_hal.h"


rtc_repeat_alarm_state_t repeatAlarmState;


/*lint -save  -e970 Disable MISRA rule (6.3) checking. */
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{
  /* Write your local variable definition here */

  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/


  INT_SYS_DisableIRQ(LLWU_IRQn);
```

```
  /*
   rtc_datetime_t fecha = {
              .year = 2016,
              .month = 2,
              .day = 1,
              .hour = 21,
              .minute = 0,
              .second = 0
    };*/


   POWER_SYS_SetWakeupModule(kPowerManagerWakeupRtcAlarm, true);
   POWER_SYS_SetWakeupModule(kPowerManagerWakeupLptmr, true);
  // RTC_DRV_SetDatetime(FSL_RTCTIMER1,&fecha);


   configureRTCclock();

   RTC_DRV_InitRepeatAlarm(FSL_RTCTIMER1, &repeatAlarmState);

   set_alarm_interval();

   //g_xtalRtcClkFreq=32768U;
   LPTMR_DRV_Start(FSL_LPTMR1);

    GPIO_DRV_ClearPinOutput(LEDRGB_GREEN);

    for(;;)
        DoWholeMedition();




    //t1=LeerTemperatura(J1_2);
    //t2=LeerTemperatura(J1_4);
    //GetCurrent();GetRadiacion();

  /*** Don't write any code pass this line, or it will be deleted during code generation. ***/
  /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component. DON'T MODIFY
THIS CODE!!! ***/
  #ifdef PEX_RTOS_START
    PEX_RTOS_START();                  /* Startup of the selected RTOS. Macro is defined by the
RTOS component. */
  #endif
  /*** End of RTOS startup code.  ***/
  /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
  for(;;){}
  /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
} /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/

/* END main */
/*!
** @}
*/


/*
```

'

```
 * Measure.h
 *
 *  Created on: 7/1/2016
 *      Author: julit
 */
#include "stdint.h"
#include "stdbool.h"
#include "rtcTimer1.h"

// esta funcion realiza el loop principal del programa,
//mide todos los sensores, los envia a la memoria y servidor remoto
//y pone al sistema en bajo consumo
void DoWholeMedition(void);

//Esta funcion envia las mediciones al servidor remoto y la mem interna con un timestamp (fecha)
de medición
void SendMeditions(rtc_datetime_t *fecha);

// esta función fija el intervalo de la alarma.
void set_alarm_interval(void);

//esta funcion devueve la frecuencia de  medicion actual
uint32_t getMeasuringMinutes(void);

//esta funcion configura la frecuencia de  medicion actual
void SetMeasuringMinutes(uint32_t minutos);

//esta funcion devueve el puntero al valor de frecuencia de  medicion actual
uint32_t * GetFrequencyPtr(void);

//esta funcion devueve el puntero al string que contiene la contraseña API actual
char ** GetAPIPtr(void);

//esta función recibe un intervalo de alarma en minutos (minutes)
//y la fecha actual (current_time) y actualiza la alarma del RTC.
void update_alarm(uint32_t minutes,rtc_datetime_t *current_time);

//esta función configura el RTC con la hora actual,
//que es descargada de la red celular a través del SIM900
void configureRTCclock(void);

/*
 * Measure.c
 *
 *  Created on: 7/1/2016
 *      Author: julit
 */
#include "Measure.h"
#include "temperatura.h"
#include "Viento.h"
#include "Circular.h"
#include "GSM.h"
#include "Power.h"
#include "gpio1.h"
#include "Radiacion.h"
#include "Caudal.h"
#include "rtcTimer1.h"
```

```
#include "SDcard.h"
#include "Corriente.h"

#define MEAS_PER_SMS 1
#define CAUDAL_THRESHOLD 0.1

typedef struct {
    uint32_tCircular Direccion;
    floatCircular Velocidad;
} Viento_t;

typedef struct {
    floatCircular Entrada;
    floatCircular Salida;
}Temperatura_t;

typedef floatCircular Caudal_t;

typedef floatCircular Corriente_t;

typedef floatCircular Radiacion_t;


rtc_datetime_t repeat_interval = {
        .year = 0,
        .month = 0,
        .day = 0,
        .hour = 0,
        .minute = 1,
        .second = 0
};

//loaded measured data
static Viento_t Viento;
static Caudal_t Caudal;
static Radiacion_t Radiacion;
static Temperatura_t Temperaturas;
static Corriente_t Corriente;

//API key for thingspeak.com
static char API[20]="VXVFJ56XT07D9C06";
static uint32_t Frecuencia=2;//in minutes

void DoWholeMedition(void){
    rtc_datetime_t fecha;
    RTC_DRV_GetDatetime(FSL_RTCTIMER1, &fecha);

    StartMeasuringCaudal();
    StartMeasuringWind();

    while(IsMeasuringCaudal() || IsMeasuringWind());

    push_float(&Radiacion,GetRadiacion());
    push_float(&Viento.Velocidad,GetWindSpeed());
    push_uint32(&Viento.Direccion,GetWindDirection());
    push_float(&Caudal,GetCaudal());
    push_float(&Corriente,GetCurrent());
```

```
    push_float(&Temperaturas.Entrada,LeerTemperatura(J1_2));
    push_float(&Temperaturas.Salida,LeerTemperatura(J1_4));
    SendMeditions(&fecha);
    pop_float(&Temperaturas.Entrada);pop_float(&Temperaturas.Salida);
    pop_float(&Radiacion);pop_uint32(&Viento.Direccion);
    pop_float(&Viento.Velocidad);pop_float(&Corriente);

    if(pull_float(&Caudal)<CAUDAL_THRESHOLD){
        PowerOffSystem();
    }

}


void set_alarm_interval(void){
    bool resp;
    rtc_datetime_t fecha;
    RTC_DRV_GetDatetime(FSL_RTCTIMER1, &fecha);
    update_alarm(10, &fecha);
    repeat_interval.minute=Frecuencia;
    resp=RTC_DRV_SetAlarmRepeat(FSL_RTCTIMER1, &fecha, &repeat_interval);
}


void update_alarm(uint32_t seconds, rtc_datetime_t *current_time){
    uint32_t minutes, hours;
    RTC_DRV_GetDatetime(FSL_RTCTIMER1,current_time);

    if((*current_time).second+seconds>=60){
        minutes=seconds/60;
        (*current_time).second=(*current_time).second+seconds-60*minutes;
        if((*current_time).minute+minutes>=60){
            hours=minutes/60;
            (*current_time).minute = (*current_time).minute+minutes-60*hours;
            if((*current_time).hour+hours>=24){
                (*current_time).hour=(*current_time).hour+hours-24;
                (*current_time).day+=1;
            }
            else
                (*current_time).hour+=1;
        }
        else
            (*current_time).minute += minutes;
    }
    else
        (*current_time).second += seconds;
}



uint32_t * GetFrequencyPtr(void){
    return &Frecuencia;
}

char ** GetAPIPtr(void){
    return &API;
}
```

'

```
void configureRTCclock(void){
    rtc_datetime_t fecha;
    bool done=false;
    while(done==false){
        if(!isSIMon())
            TurnSIMon();
        if(isSIMon()){
            done=getTime(&fecha);
            RTC_DRV_SetDatetime(FSL_RTCTIMER1,&fecha);
        }
        TurnSIMoff();
    }
}

void SendMeditions(rtc_datetime_t *fecha){
    //field1 Caudal//field2 Temp IN//field3 Temp OUT//field4 Radiacion//field5 VientoDir//field6
VientoVel //field7 Corriente
    if(!isSIMon())
        TurnSIMon();
    if(isSIMon()){
        SendThingSpeak(API,fecha,front_float(&Caudal),
                front_float(&Temperaturas.Entrada),front_float(&Temperaturas.Salida),
                front_float(&Radiacion), front_uint32(&Viento.Direccion),
                front_float(&Viento.Velocidad),front_float(&Corriente));
        if (ReceiveSMS(&Frecuencia,API))
            set_alarm_interval();
    }
    if(front_float(&Caudal)<CAUDAL_THRESHOLD){
        TurnSIMoff();
    }
    SendToSD(fecha,front_float(&Caudal),front_float(&Temperaturas.Entrada),
            front_float(&Temperaturas.Salida),front_float(&Radiacion),
front_uint32(&Viento.Direccion),
            front_float(&Viento.Velocidad),front_float(&Corriente));
}

uint32_t getMeasuringMinutes(void){
    return Frecuencia;
}


void SetMeasuringMinutes(uint32_t minutos){
    Frecuencia=minutos;
}

/*
 * Power.h
 *
 *  Created on: 7/1/2016
 *      Author: julit
 */

// esta funcion cambia el color del LED del microcontrolador de verde a rojo
//y coloca al microcontrolador en low leakage stop mode
void PowerOffSystem(void);
```

'

```c
/*
 * Power.c
 *
 *  Created on: 7/1/2016
 *      Author: julit
 */
#include "Power.h"
#include "Cpu.h"
#include "gpio1.h"

void PowerOffSystem(void){

    GPIO_DRV_SetPinOutput(LEDRGB_GREEN);
    GPIO_DRV_ClearPinOutput(LEDRGB_RED);
    if (kPowerManagerSuccess != POWER_SYS_SetMode(0, kPowerManagerPolicyForcible))
       while (1) {
          /* Error when setting low power mode */
           GPIO_DRV_TogglePinOutput(LEDRGB_RED);
           OSA_TimeDelay(500);
       }

    GPIO_DRV_ClearPinOutput(LEDRGB_GREEN);
}


/*
 * GSM.h
 *
 *  Created on: 5/1/2016
 *      Author: julit
 */

#include "stdint.h"
#include "stdbool.h"
#include "rtcTimer1.h"

//esta función envía el texto por SMS(text) al numero (user_number)
void SendSMS(char *text,char *user_number);

//esta función lee la hora actual de la proveedora de datos y lo devuelve en (día).
// Devuelve True si la lectura fue efectiva y false en caso contrario
bool getTime(rtc_datetime_t *dia);

//esta función envía el texto (text)
void SendTextMessage(char * text);

// esta funcion apaga el echo del modulo SIM900
void SendEchoOff(void);

// esta funcion inicia la comunicacion con el modulo SIM900
void SendSyncComm(void);

//esta funcion recibe la respuesta del SIM900 y la guarda en el Buffer.
//La funcion bloquea hasta obtener una respuesta de tamaño size o hasta que se acabe el tiempo
marcado por los milisegundos de (timeblocking)
void ReceiveText(uint8_t *Buffer, uint32_t size,uint32_t time_blocking);
```

'

```
//esta funcion recibe la respuesta del SIM900 y la guarda en el Buffer.
//La funcion bloquea hasta obtener una respuesta de tamaño size o hasta que se acabe el tiempo
marcado por los milisegundos de (timeblocking)
void ReceiveText2(uint8_t *Buffer, uint32_t size,uint32_t time_blocking);

//esta funcion envia al SIM900 el texto (text), y devuelve True si la respuesta es la esperada
(ExpectedResponse). Por cada intento bloquea durante (timeblocking) milisegundos.
//En caso de obtener una respuesta incorrecta, repite el procedimiento (chances) veces.
bool SendAndCheck(char *text, char *ExpectedResponse, uint32_t chances,uint32_t canti_espacios,
uint32_t timeblocking);

//esta función devuelve el estado de el SIM900 (si esta prendido devuelve True)
bool isSIMon(void);

//esta función prende el SIM900. Puede ser inefectivo, el estado queda almacenado en sim_state
void TurnSIMon(void);

//esta función apaga el SIM900.
void TurnSIMoff(void);

// esta función sirve de auxiliar a SendSMS, envia el texto de SendSMS.
void SendText(char *text);

//field1 Caudal
//field2 Temp IN
//field3 Temp OUT
//field4 Radiacion
//field5 VientoDir
//field6 VientoVel
void SendThingSpeak(const char * APIKEY, rtc_datetime_t *fecha ,
        float field1,float field2, float field3, float field4,
        uint32_t field5, float field6,float field7);

//esta función recibe un dato en formato float, y
//lo devuelve en formato string en (str), con MAX_PARAMETER_DIGITS digitos (incluyendo el '.'
decimal)
void floatTostring(float dato, char *str);

//recibe todos los SMS y asigna a frecuencia los cambios pertinentes.
//Devuelve True si hubo un cambio de frecuencia de mediciones.
bool ReceiveSMS(uint32_t * Frecuencia, char * APIKEY);

//ESTO NECESITA RECIBIR UN PARAMETER CON MAX_PARAMETER_STRING lugares vacios
bool SearchForParameterString(uint8_t *Buffer,const char * texto, char *parameter, char *
numero);

//funcion que busca un mensaje con el key texto y luego le parametro numerico.
//De existir el key y su valor numerico, lo asigna a parameter.
bool SearchForParameterInt(uint8_t *Buffer,const char * texto, uint32_t *parameter, char *
numero);

/*
 * GSM.c
 *
 *  Created on: 5/1/2016
 *      Author: julit
 */
```

```
#include "GSM.h"
#include "uartCom1.h"
#include "fsl_uart_driver.h"
#include "string.h"
#include "gpio1.h"
#include "rtcTimer1.h"

#define MAX_PARAMETER_DIGITS 10
#define MAX_PARAMETER_STRING 20
#define MAX_DECIMALES 3
#define BUFFER_SIZE 1024
#define TIEMPO_ESPERA 10000
#define FRECUENCIA_MAX 1440
#define FRECUENCIA_MIN 2
#define LARGO_NUMERO 11

uint8_t buffer[BUFFER_SIZE];

static bool sim_state=false;

//static char user_number[LARGO_NUMERO]="1160359338";

bool isSIMon(void){
    return sim_state;
}

void TurnSIMon(){
    uint32_t loops=0;
    char *aux=NULL;
    sim_state=false;
    while(!SendAndCheck("AT\r\n","\r\n",2,2,100) && loops<3){
        GPIO_DRV_ClearPinOutput(J2_5);
        OSA_TimeDelay(1100);
        GPIO_DRV_SetPinOutput(J2_5);
        OSA_TimeDelay(3000);
        loops++;
    }
    if(loops<3){
        sim_state=SendAndCheck("ATE0\r\n","\r\n",2,4,100);
        OSA_TimeDelay(1000);
    }

}

void TurnSIMoff(){
    SendAndCheck("AT+CIPCLOSE\r\n","OK",2,2,1000);
    OSA_TimeDelay(1000);
    SendText("AT+CPOWD=1\r\n");
    UART_DRV_AbortReceivingData(FSL_UARTCOM1);
    memset((void *)buffer,0,BUFFER_SIZE);
    sim_state=false;
}


void SendText(char *text){
    UART_DRV_SendData(FSL_UARTCOM1,text,strlen(text));
```

```
                                                '
}


void SendSMS(char *text,char *user_number){
    char str[32];
    if(SendAndCheck("AT+CMGF=1\r\n","OK",5,2,100)){

        sprintf(str,"AT+CMGS=\"%s\"\r",user_number);
        SendText(str);
        OSA_TimeDelay(1000);
        SendTextMessage(text);

        //OSA_TimeDelay(10000);
    }
}


bool getTime(rtc_datetime_t *dia){
    char *aux;
    uint32_t valor;
    rtc_datetime_t fecha;
    bool done=false;

    OSA_TimeDelay(2000);
    if(SendAndCheck("AT+CGATT=1\r\n","OK",5,4,2500)){
        if(SendAndCheck("AT+CIPMUX=0\r\n","OK",5,4,2000)){
            OSA_TimeDelay(4000);
            if(SendAndCheck("AT+CSTT=\"igprs.claro.com.ar\",\"clarogprs\",
\"clarogprs999\"\r\n","OK",10,4,4500)){
                if(SendAndCheck("AT+CIICR\r\n","OK",5,4,2500)){
                    if(SendAndCheck("AT+CLTS=1\r\n","OK",5,4,2500)){
                            SendText("AT+CCLK?\r\n");
                            UART_DRV_ReceiveData(FSL_UARTCOM1,buffer,BUFFER_SIZE);
                            OSA_TimeDelay(3000);
                            aux=strstr(buffer,"+CCLK: \"");
                            aux+=8;
                            valor=(*aux-0x30)*10;
                            aux++;
                            valor+=(*aux-0x30);
                            fecha.year=valor+2000;
                            aux+=2;
                            valor=(*aux-0x30)*10;
                            aux++;
                            valor+=(*aux-0x30);
                            fecha.month=valor;
                            aux+=2;
                            valor=(*aux-0x30)*10;
                            aux++;
                            valor+=(*aux-0x30);
                            fecha.day=valor;
                            aux+=2;
                            valor=(*aux-0x30)*10;
                            aux++;
                            valor+=(*aux-0x30);
                            fecha.hour=valor;
                            aux+=2;
                            valor=(*aux-0x30)*10;
```

```
                                '

                                aux++;
                                valor+=(*aux-0x30);
                                fecha.minute=valor;
                                aux+=2;
                                valor=(*aux-0x30)*10;
                                aux++;
                                valor+=(*aux-0x30);
                                fecha.second=valor;
                                *dia=fecha;
                                done=true;
                            }
                        }
                    }
                }
            }
            return done;
}

//field1 Caudal
//field2 Temp IN
//field3 Temp OUT
//field4 Radiacion
//field5 VientoDir
//field6 VientoVel
void SendThingSpeak(const char * APIKEY, rtc_datetime_t *fecha,
        float field1,float field2, float field3, float field4, uint32_t field5, float
field6,float field7){
    char *aux=NULL;
    //char str[256], field1str[MAX_PARAMETER_DIGITS], field2str[MAX_PARAMETER_DIGITS],
field3str[MAX_PARAMETER_DIGITS],field4str[MAX_PARAMETER_DIGITS], field6str[MAX_PARAMETER_DIGITS],
field5str[MAX_PARAMETER_DIGITS], field7str[MAX_PARAMETER_DIGITS];
    char str[256], field1str[MAX_PARAMETER_DIGITS];
    uint32_t paso=0;
    OSA_TimeDelay(2000);
    if(SendAndCheck("AT+CGATT=1\r\n","OK",5,4,2500)){
        paso++;
        if(SendAndCheck("AT+CIPMUX=0\r\n","OK",5,4,2000)){
            paso++;
            OSA_TimeDelay(4000);
            if(SendAndCheck("AT+CSTT=\"igprs.claro.com.ar\",\"clarogprs\",
\"clarogprs999\"\r\n","OK",10,4,4500)){
                paso++;
                if(SendAndCheck("AT+CIICR\r\n","OK",5,4,2500)){
                    paso++;
                    if(SendAndCheck("AT+CIFSR\r\n","\r\n",5,4,2500)){
                        paso++;
                        if(SendAndCheck("AT+CIPSTART=\"TCP\",\"api.thingspeak.com\",
\"80\"\r\n","OK",5,4,2500)){
                            paso++;
                            OSA_TimeDelay(2000);
                            sprintf(str,"GET /update?key=%s&",APIKEY);
                            floatTostring(field1,field1str);
                            sprintf(str,"%sfield1=%s&",str,field1str);
                            floatTostring(field2,field1str);
                            sprintf(str,"%sfield2=%s&",str,field1str);
                            floatTostring(field3,field1str);
                            sprintf(str,"%sfield3=%s&",str,field1str);
```

```
                                '

                                floatTostring(field4,field1str);
                                sprintf(str,"%sfield4=%s&",str,field1str);
                                sprintf(str,"%sfield5=%d&",str,field5);
                                floatTostring(field6,field1str);
                                sprintf(str,"%sfield6=%s&",str,field1str);
                                floatTostring(field7,field1str);
                                sprintf(str,"%sfield7=%s&",str,field1str);

                                //&created_at=2013-02-05T21:49:00+01:00
                                sprintf(str,"%screated_at=%d-%02d-%02dT%02d:%02d:%02d-03:00 ",
                                        str,fecha->year,fecha->month,fecha->day,fecha->hour,fecha->mi
nute,fecha->second);

                                sprintf(str,"%sHTTP/1.1\r\nHOST: api.thingspeak.com\r\nConnection:
close\r\nAccept: */*\r\n\r\n",str);
                                //sprintf(str,"GET /update?key=%s&field1=%s&field2=%s&field3=
%s&field4=%s&field5=%s&field6=%s&field7=%s HTTP/1.1\r\nHOST: api.thingspeak.com\r\nConnection:
close\r\nAccept:
*/*\r\n\r\n",APIKEY,field1str,field2str,field3str,field4str,field5str,field6str,field7str);


                                OSA_TimeDelay(2000);
                                if(SendAndCheck("AT+CIPSEND\r\n","\r\n>",1,4,1000)){
                                    paso++;
                                    UART_DRV_AbortReceivingData(FSL_UARTCOM1);
                                    memset((void *)buffer,0,BUFFER_SIZE);

                                    UART_DRV_ReceiveData(FSL_UARTCOM1,buffer,BUFFER_SIZE);
                                    UART_DRV_SendDataBlocking(FSL_UARTCOM1,str,strlen(str),1000);

                                    uint8_t term=0x1a;
                                    UART_DRV_SendData(FSL_UARTCOM1,&term,1);

                                }
                            }
                        }
                    }
                }
            }
        }
    }
    paso=0;
}



bool SendAndCheck(char *text, char *ExpectedResponse, uint32_t chances,
        uint32_t canti_espacios, uint32_t timeblocking){
    char *aux=NULL,*aux2=NULL;
    uint32_t start=OSA_TimeGetMsec();
    uint32_t loops=chances;
    while(loops-- && aux==NULL){
        UART_DRV_AbortReceivingData(FSL_UARTCOM1);
        memset((void *)buffer,0,BUFFER_SIZE);

        UART_DRV_ReceiveData(FSL_UARTCOM1,buffer,BUFFER_SIZE);
        SendText(text);
```

```
        while(aux==NULL && OSA_TimeDiff(start,OSA_TimeGetMsec())<TIEMPO_ESPERA){
            aux=strstr(buffer,"\r\n");
        }
        aux+=2;

        for(uint32_t i=1; i<canti_espacios;i++){
            while(aux2==NULL && OSA_TimeDiff(start,OSA_TimeGetMsec())<TIEMPO_ESPERA){
                    aux2=strstr(aux,"\r\n");
            }
            aux=aux2+2;
        }

        aux=strstr(buffer,ExpectedResponse);
        if(aux==NULL)
            OSA_TimeDelay(timeblocking);
    }
    if(aux==NULL)
        return false;
    else{
        return true;
    }
}

void SendTextMessage(char * text){
    UART_DRV_SendDataBlocking(FSL_UARTCOM1,text,strlen(text),1000);
    uint8_t aux=0x1a;
    UART_DRV_SendDataBlocking(FSL_UARTCOM1,&aux,1,1000);
}

//recibe todos los SMS y asigna a frecuencia los cambios pertinentes
bool ReceiveSMS(uint32_t * Frecuencia, char * APIKEY){
uint32_t p;
bool changed_freq=false;
char numero [LARGO_NUMERO];

    if(SendAndCheck("AT+CMGF=1\r\n","OK",5,2,1000)){
    //uint8_t *Buffer= calloc(BUFFER_SIZE,sizeof(uint8_t));

        UART_DRV_AbortReceivingData(FSL_UARTCOM1);
        memset((void *)buffer,0,BUFFER_SIZE);
        SendText("AT+CMGL=\"ALL\"\r\n");

        UART_DRV_ReceiveData(FSL_UARTCOM1,buffer,BUFFER_SIZE);
        OSA_TimeDelay(3000);

        //Parse the messages
        if(SearchForParameterInt(buffer,"Periodo Mediciones: ", Frecuencia,numero)){
            char str[40];
            changed_freq=true;
            sprintf(str,"Periodo cambiado a %d", *Frecuencia);
            SendSMS(str,numero);
        }

        if(SearchForParameterString(buffer,"API KEY: ", APIKEY,numero)){
            char str[40];
            sprintf(str,"API KEY cambiada a %s", APIKEY);
            SendSMS(str,numero);
```

```
        }

        //delete all msgs
        OSA_TimeDelay(4000);
        SendText("AT+CMGD=1,4\r\n");
        ReceiveText(buffer,BUFFER_SIZE,1000);
        p=1;
    }

    return changed_freq;

}

void floatTostring(float dato, char *str){
    float base=1;

    uint32_t digitos=0,i=0;
    if(dato<0.0001 && dato>-0.0001){
        str[i++]='0';
    }else{
        while(base<=dato){
            base*=10;
            digitos++;
        }
        if (digitos==0){
            str[i++]='0';
            digitos++;
            str[i++]='.';
            digitos++;
            str[i++]='0'+(int)(dato/base)%10;
            digitos++;
        }

        while(digitos<=MAX_PARAMETER_DIGITS){
            base/=10;
            str[i++]='0'+(int)(dato/base)%10;
            if(base<10 && base>=1){
                str[i++]='.';
                digitos++;
            }
            digitos++;
        }
    }

    str[i]=0x00;
}

//ESTA FUNCION NECESITA RECIBIR UN PARAMETER CON MAX_PARAMETER_STRING lugares vacios
bool SearchForParameterString(uint8_t *Buffer,const char * texto,
        char *parameter,char * numero){
    bool ret=false;
    char * aux, *aux2;
    aux=strstr(Buffer,texto);
    uint32_t p;

    while(aux!=NULL){
        aux2=aux-40;
```

```
        aux2=strstr(Buffer,"+54911");
        aux2+=4;
        p=0;
        while(p<LARGO_NUMERO-1){
            numero[p++]=(*aux2);
            aux2++;
        }
        numero[p]=0x00;

        aux+=strlen(texto);
        uint32_t count=0;
        while((*aux)!=0x0D && (*aux)!=0x0A && count<MAX_PARAMETER_STRING){
            parameter[count]=(*aux);
            count++;
            aux++;
        }
        aux=strstr(aux,texto);
        parameter[count]=0x00;
        ret=true;
    }

    return ret;
}

bool SearchForParameterInt(uint8_t *Buffer,const char * texto, uint32_t *parameter,char *
numero){
    bool ret=false;
    char * aux, *aux2;
    aux=strstr(Buffer,texto);
    uint32_t valor;
    uint32_t p;

    while(aux!=NULL){
        aux2=aux-40;
        aux2=strstr(Buffer,"+54911");
        aux2+=4;
        p=0;
        while(p<LARGO_NUMERO-1){
            numero[p++]=(*aux2);
            aux2++;
        }
        numero[p]=0x00;

        aux+=strlen(texto);

        uint32_t index=0;
        char digitos[MAX_PARAMETER_DIGITS];
        while((*aux)>='0' && (*aux)<='9' && index<MAX_PARAMETER_DIGITS){
            digitos[index]=(*aux)-'0';
            index++;
            aux++;
        }
        if(index){
            valor=0;
            //*parameter=0;
            ret=true;
        }
```

```
        uint32_t j=0;
        while(index){
            uint32_t factor=1, index2=index;
            while((index2--)>1)
                factor*=10;
            //(*parameter)+=factor*digitos[j++];
            valor+=factor*digitos[j++];
            index--;
        }
        aux=strstr(aux,texto);
    }
    if(valor<=FRECUENCIA_MAX && valor>=FRECUENCIA_MIN)
        *parameter=valor;
    else
        ret=false;

    return ret;
}


void ReceiveText(uint8_t *Buffer, uint32_t size,uint32_t time_blocking){
    char *aux=NULL,*aux2=NULL;
    uint32_t start=OSA_TimeGetMsec();

    UART_DRV_ReceiveData(FSL_UARTCOM1,Buffer,size);

    while(aux==NULL && OSA_TimeGetMsec()-start<time_blocking){
        aux=strstr(Buffer,"\r\n");
    }
    aux+=2;

    while(aux2==NULL && OSA_TimeGetMsec()-start<time_blocking){
        aux2=strstr(aux,"\r\n");
    }
}

void ReceiveText2(uint8_t *Buffer, uint32_t size,uint32_t time_blocking){
    char *aux=NULL;
    uint32_t start=OSA_TimeGetMsec();

    memset((void *)buffer,0,BUFFER_SIZE);
    UART_DRV_AbortReceivingData(FSL_UARTCOM1);

    UART_DRV_ReceiveData(FSL_UARTCOM1,buffer,size);

    while(aux==NULL && OSA_TimeGetMsec()-start<time_blocking){
        aux=strstr(buffer,"\r\n");

    }
    OSA_TimeDelay(100);
}


void SendSyncComm(void){
    SendText("AT\r\n");
}
```

'

```
void SendEchoOff(void){
    SendText("ATE0\r\n");
}


/*
 * SDcard.h
 *
 *  Created on: 14/1/2016
 *      Author: julit
 */

// esta función recibe una cadena de texto (text)
//y la adjunta al final del archivo log.txt de la tarjeta SD
void append(char *fileNameAndPath, char *text);

//field1 Caudal
//field2 Temp IN
//field3 Temp OUT
//field4 Radiacion
//field5 VientoDir
//field6 VientoVel
// esta función recibe un conjunto de mediciones field1-field7,
//el momento de medicion (datetime) y lo adjunta al archivo log.txt de la tarjeta SD
void SendToSD(rtc_datetime_t *datetime, float field1,
        float field2, float field3, float field4,
        uint32_t field5, float field6, float field7);

/*
 * SDcard.c
 *
 *  Created on: 14/1/2016
 *      Author: julit
 */


#include "ff.h"
#include "string.h"
#include "GSM.h"
#include "rtcTimer1.h"
//#include "fsl_spi_hal.h"
#include "fsl_spi1.h"

#define SD_DRIVE_NUMBER 1
#define MAX_PARAMETER_DIGITS 10
// debe ser igual a MAX_PARAMETER_DIGITS de gsm


//field1 Caudal
//field2 Temp IN
//field3 Temp OUT
//field4 Radiacion
//field5 VientoDir
//field6 VientoVel
void SendToSD(rtc_datetime_t *datetime, float field1,float field2,
        float field3, float field4, uint32_t field5, float field6, float field7){
    char str[200], field1str[MAX_PARAMETER_DIGITS],
```

```
    field2str[MAX_PARAMETER_DIGITS], field3str[MAX_PARAMETER_DIGITS],
    field4str[MAX_PARAMETER_DIGITS], field6str[MAX_PARAMETER_DIGITS],
    field7str[MAX_PARAMETER_DIGITS];
    //rtc_datetime_t datetime;
    floatTostring(field1,field1str);
    floatTostring(field2,field2str);
    floatTostring(field3,field3str);
    floatTostring(field4,field4str);
    floatTostring(field6,field6str);
    floatTostring(field7,field7str);

    //RTC_DRV_GetDatetime(FSL_RTCTIMER1, &datetime);

    sprintf(str,"Fecha: %d/%d/%d %d:%d:%d Caudal=%s;TempIn=%s;TempOut=%s;Radiacion=%s;VientoDir=
%d;VientoVel=%s;Corriente=%s;
\n",datetime->day,datetime->month,datetime->year,datetime->hour,datetime->minute,datetime->second,
field1str,field2str,field3str,field4str,field5,field6str,field7str);

    //SPI_HAL_Enable(g_spiBase[FSL_FSL_SPI1]);
    append("/data.log", str);
    SPI_HAL_Disable(g_spiBase[FSL_FSL_SPI1]);
}

void append(char *fileNameAndPath, char *text)
{
    FATFS fileSystem;
    // mount SD card
    FRESULT fResult = f_mount(SD_DRIVE_NUMBER, &fileSystem);

    if (fResult == FR_OK)
    {
        fResult = f_chdrive(SD_DRIVE_NUMBER);

        if (fResult == FR_OK)
        {
            FIL file;
            static char pathWithDriveLetter[255];
            sprintf(pathWithDriveLetter, "%u:%s", SD_DRIVE_NUMBER, fileNameAndPath);

            /* Open a text file */
            fResult = f_open(&file, pathWithDriveLetter, FA_OPEN_ALWAYS | FA_WRITE);
            if (fResult == FR_OK)
            {
                /* Move to end of the file to append data */
                fResult = f_lseek(&file, file.fsize);

                /* Write to the file */
                uint32_t bytesWritten;
                fResult = f_write(&file, text, strlen(text), &bytesWritten);
            }
            /* Close the file */
            fResult = f_close(&file);
        }

        // unmount SD card
        fResult = f_mount(SD_DRIVE_NUMBER, NULL);
    }
```

```
}
/*
 * temperatura.h
 *
 *  Created on: 4/1/2016
 *      Author: julit
 */

#include "stdint.h"

// funcion que efectua un delay de (microsecs) microsegundos
void WaitMicrosecs(uint32_t microsecs);

//funcion que escribe '0' al sensor de temperatura conectado en el pin (pin)
void Write0toTempSensor(uint32_t pin);

//funcion que escribe '1' al sensor de temperatura conectado en el pin (pin)
void Write1toTempSensor(uint32_t pin);

//Recibe la respuesta del sensor de temperatura del pin (pin)
uint32_t ReadFromSensor(uint32_t pin);

// Resetea el sensor de temperatura
void ResetTempSensor(uint32_t pin);

//Envia la palabra (word) de 8bits al sensor de temperatura del pin (pin)
void SendWordtoTemp(uint8_t word, uint32_t pin);

//Lee la medicion de 12 bits del sensor de temperatura del pin (pin)
uint16_t ReadTemp(uint32_t pin);

//esta función es la que devuelve el dato de la
//temperatura en la configuración de 12 bits del datasheet,
//incluyendo el signo en el MSNy
float LeerTemperatura(uint32_t pin);

/*
 * temperatura.c
 *
 *  Created on: 30/12/2015
 *      Author: julit
 */

#include "fsl_gpio_driver.h"
#include "fsl_os_abstraction.h"
#include "temperatura.h"
#include "gpio1.h"
#include "calibracion.h"

void WaitMicrosecs(uint32_t microsecs){
// si el clock no esta seteado a un microseg esto no funcionará
    for(int i=0;i<(uint32_t)(microsecs);i++);
}

void Write0toTempSensor(uint32_t pin){
    GPIO_DRV_SetPinDir(pin,kGpioDigitalOutput);
    GPIO_DRV_WritePinOutput(pin,0);
```

```
    //wait for 100microseg
    WaitMicrosecs(100);
    GPIO_DRV_SetPinDir(pin,kGpioDigitalInput);
    //wait for >1microseg
    WaitMicrosecs(2);
}

void Write1toTempSensor(uint32_t pin){
    uint32_t start=OSA_TimeGetMsec();

    GPIO_DRV_SetPinDir(pin,kGpioDigitalOutput);
    GPIO_DRV_WritePinOutput(pin,0);
    //wait for >1microseg
    WaitMicrosecs(5);
    GPIO_DRV_SetPinDir(pin,kGpioDigitalInput);
    while(GPIO_DRV_ReadPinInput(pin)==0 && (start+1)<OSA_TimeGetMsec());
    //wait for >1microseg
    WaitMicrosecs(60);
}

uint32_t ReadFromSensor(uint32_t pin){
    GPIO_DRV_SetPinDir(pin,kGpioDigitalOutput);
    GPIO_DRV_WritePinOutput(pin,0);
    //wait for >1microseg
    WaitMicrosecs(2);
    GPIO_DRV_SetPinDir(pin,kGpioDigitalInput);
    //wait for 15-1 microseg
    WaitMicrosecs(12);
    uint32_t retorno=GPIO_DRV_ReadPinInput(pin);
    //wait for recovery
    WaitMicrosecs(50);
    return retorno;
}

void ResetTempSensor(uint32_t pin){
    //set as tx
    //uint32_t dir=GPIO_DRV_GetPinDir(pin);
    GPIO_DRV_SetPinDir(pin,kGpioDigitalOutput);
    //reset pulse
    GPIO_DRV_WritePinOutput(pin,0);
    //wait for 480-960microseg
    WaitMicrosecs(700);
    //set pin as input to receive confirmation
    GPIO_DRV_SetPinDir(pin,kGpioDigitalInput);
    //wait for confirmation
    //while(GPIO_DRV_ReadPinInput(pin));
    //wait for >400microseg
    WaitMicrosecs(450);

}

void SendWordtoTemp(uint8_t word, uint32_t pin){
    for(int i=0;i<8;i++){
        if( (0x01 << i) & word)
            Write1toTempSensor(pin);
        else
            Write0toTempSensor(pin);
```

```
    }
}

//returns true if the read is succesfull
uint16_t ReadTemp( uint32_t pin){
    uint16_t retorno=0x00;

    for(int i=0; i<16 ;i++){
        retorno+= ( ReadFromSensor(pin) << i);
    }

/* in case of using CRC code
    for(int i=0; i<7 ;i++)
        for(int j=0; j<8; j++)
            ReadFromSensor(pin);

    return true;
    */

    return retorno;
}

//devuelve la temperatura actual
float LeerTemperatura(uint32_t pin){
    uint16_t temp;
    float retorno=0;

    ResetTempSensor(pin);

    //send CCh (SKIP ROM)
    SendWordtoTemp(0xCC,pin);

    //Send 44h (Convert Temperature)
    SendWordtoTemp(0x44,pin);

    //wait for conversion (full conv 750ms)
    OSA_TimeDelay(850);

    //reset
    ResetTempSensor(pin);

    //send CCh (SKIP ROM)
    SendWordtoTemp(0xCC,pin);

    //send BEh (Read ScratchPad)
    SendWordtoTemp(0xBE,pin);

    //wait for 1 ms
    OSA_TimeDelay(1);

    temp=ReadTemp(pin);

    uint32_t i;
    float base=0.0625;
    for(i=0;i<11;i++){
        retorno+=(((temp >> i) & 0x01)*base);
```

```
        base*=2;
    }

    retorno-=(((temp >> i) & 0x01)*base);

    if(pin==J1_2)
        return retorno*ganancia_temp_in+offset_temp_in;
    else
        return retorno*ganancia_temp_out+offset_temp_out;

}

/*
 * Viento.h
 *
 *  Created on: 4/1/2016
 *      Author: julit
 */
/*
 * Viento.c
 *
 *  Created on: 4/1/2016
 *      Author: julit
 */
#include "Viento.h"
#include "fsl_adc16_driver.h"
#include "fsl_os_abstraction.h"
#include "adConv1.h"
#include "calibracion.h"

#define CTE_PULSOS 3621
#define CTE_A_GRADOS 0.08789
#define MEDITION_SECS 3 //3 segundos que se cuentan pulsos

//indicador de estado de medición

static bool Measuring;
static uint32_t pulses;
static uint32_t StartTime;
static float WindSpeed;
static float direccion;
static int32_t offset=0;

bool IsMeasuringWind(void){
    uint32_t tiempo=OSA_TimeGetMsec()-StartTime;

    if(tiempo> MEDITION_SECS*1000){
        StopMeasuringWind();

        //get wind directio
        ADC16_DRV_ConfigConvChn(FSL_ADCONV1,0,&adConv1_ChnConfig0);
        direccion=(ADC16_DRV_GetConvValueRAW(FSL_ADCONV1,0))*CTE_A_GRADOS+offset;

        //get windspeed
        WindSpeed=pulses*CTE_PULSOS/(float)tiempo;
    }
    return Measuring;
```

```
}

void StartMeasuringWind(void){
    if(!IsMeasuringWind()){
        Measuring=true;
        pulses=0;
        StartTime=OSA_TimeGetMsec();
    }
}

void StopMeasuringWind(void){
    Measuring=false;
}

void NewWindPulse(void){
    uint32_t tiempo=OSA_TimeGetMsec()-StartTime;

    pulses++;

    if(tiempo> MEDITION_SECS*1000){
        StopMeasuringWind();

        //get wind direction
        ADC16_DRV_ConfigConvChn(FSL_ADCONV1,0,&adConv1_ChnConfig0);
        direccion=(ADC16_DRV_GetConvValueRAW(FSL_ADCONV1,0))*CTE_A_GRADOS+offset;

        //get windspeed
        WindSpeed=pulses*CTE_PULSOS/(float)tiempo;
    }
}


void CalibrateWindDir(int32_t offs){
    offset=offs;
}

//devuelve valor entre 0 y 360 grados
uint32_t GetWindDirection(void){
    uint32_t octante;
    uint32_t dir;

    dir=direccion*ganancia_dir_viento+offset_dir_viento;
    if(dir<45)
        octante=1;
    else if (dir<90)
        octante=2;
    else if (dir<135)
        octante=3;
    else if (dir<180)
        octante=4;
    else if (dir<225)
        octante=5;
    else if (dir<270)
        octante=6;
    else if (dir<315)
        octante=7;
    else
```

```
        octante=8;
    return octante;
}

float GetWindSpeed(void){
    return WindSpeed*ganancia_vel_viento+offset_vel_viento;
}



#include "stdbool.h"
#include "stdint.h"

#define WIND_CHANNEL 1

// función que devuele True en caso de estar en un intervalo de medicion de velocidad
bool IsMeasuringWind(void);

// función que devuele la velocidad del viento en float.
//Es necesario llamar a StartMeasuringWind previamente y esperar a que IsMeasuringWind sea false
float GetWindSpeed(void);

// devuelve la direccion del viento en 32 bits
//Es necesario llamar a StartMeasuringWind previamente y esperar a que IsMeasuringWind sea false
uint32_t GetWindDirection(void);

// función que comienza el intervalo de medición de 2.25 segundos
void StartMeasuringWind(void);

// función que finaliza el intervalo de medición de 2.25 segundos
void StopMeasuringWind(void);

// función que agrega un pulso a la cuenta de pulsos para
void NewWindPulse(void);

/*
 * Radiacion.h
 *
 *  Created on: 12/1/2016
 *      Author: julit
 */

// esta funcion se comunica mediante I2C con el sensor de radiación
//y devuelve el valor obtenido en float.
float GetRadiacion(void);

/*
 * Radiacion.c
 *
 *  Created on: 12/1/2016
 *      Author: julit
 */

#include "i2cCom1.h"
#include "fsl_i2c_master_driver.h"
#include "fsl_os_abstraction.h"
#include "Radiacion.h"
```

'

```
#include "calibracion.h"

#define POWERON_CMD 0x01
#define ONE_TIME_CONV_H_CMD 0x20
#define UN_SEGUNDO 1000
#define LUX_TO_WM2 4.02

//devuelve la iluminacion en lx
float GetRadiacion(void){
    uint8_t RxBuffer[3];
    uint8_t TxBuffer[2];
    float dato;

    TxBuffer[0]=POWERON_CMD;
    TxBuffer[1]=0x00;
    i2c_status_t status;
    status=I2C_DRV_MasterSendDataBlocking(FSL_I2CCOM1,
            &i2cCom1_MasterConfig0,NULL,0,TxBuffer,1,UN_SEGUNDO);

    TxBuffer[0]=ONE_TIME_CONV_H_CMD;
    status=I2C_DRV_MasterSendDataBlocking(FSL_I2CCOM1,
            &i2cCom1_MasterConfig0,NULL,0,TxBuffer,1,UN_SEGUNDO);
    OSA_TimeDelay(UN_SEGUNDO);

    status=I2C_DRV_MasterReceiveDataBlocking(FSL_I2CCOM1,
            &i2cCom1_MasterConfig0,NULL,0,RxBuffer,2,UN_SEGUNDO);

    dato=(RxBuffer[0]*256+RxBuffer[1])/1.2; //siguiendo la formulita del datasheet

    dato*=LUX_TO_WM2; // lux to W/m^2 conversion for sunlight
    return dato*ganancia_radiacion+offset_radiacion;
}

/*
 * Corriente.h
 *
 *  Created on: 26/1/2016
 *      Author: julit
 */

// función que mide la corriente instantánea durante 500 milisegundos
//y estima la corriente rms con el valor maximo y minimo obtenido, que es devuelto en float.
float GetCurrent(void);
/*
 * Corriente.c
 *
 *  Created on: 26/1/2016
 *      Author: julit
 */
#include "Corriente.h"
#include "adConv1.h"
#include "calibracion.h"
// 0A-1.38V- 1770
//1ADC -1.32V - 1648
```

'

```
#define CTE_A_CORRIENTE 0.01442
#define OFFSET_CORRIENTE 1770
#define TIEMPO_MEDICION 500 //500mseg= 25 periodos de 50Hz
#define PEAK_TO_RMS 1.4142135

float GetCurrent(void){

    uint32_t start=OSA_TimeGetMsec(), cte;
    float valor, max=-10, min=10, retorno;
    while(OSA_TimeGetMsec()-start<TIEMPO_MEDICION){
        ADC16_DRV_ConfigConvChn(FSL_ADCONV1,0,&adConv1_ChnConfig1);
        cte=ADC16_DRV_GetConvValueRAW(FSL_ADCONV1,0);
        valor=((float)cte-OFFSET_CORRIENTE)*CTE_A_CORRIENTE;
        if(valor>max)
            max=valor;
        if(valor<min)
            min=valor;
    }

    retorno=(max-min)/2.0/PEAK_TO_RMS*ganancia_amperimetro+offset_amperimetro;
    return retorno;
}


/*
 * Caudal.h
 *
 *  Created on: 5/1/2016
 *      Author: julit
 */
#include "stdbool.h"
#define CAUDAL_CHANNEL 0

//verifica si se esta midiendo el caudal, devuele True en caso positivo
bool IsMeasuringCaudal(void);

//Empieza a medir el caudal, el lptimer cuenta pulsos hasta que se llame StopMeasuringCaudal
void StartMeasuringCaudal(void);

//Deja de medir caudal, el lptimer deja de contar pulsos
void StopMeasuringCaudal(void);

//Aumenta en uno el contador de pulsos
void NewCaudalPulse(void);

//Devuelve el valor de caudal medido. Debe llamarse despues de StopMeasuringCaudal
float GetCaudal(void);
/*
 * Caudal.c
 *
 *  Created on: 5/1/2016
 *      Author: julit
 */

#include "Caudal.h"
#include "fsl_os_abstraction.h"
#include "calibracion.h"
```

'

```
#define CTE_PULSOS 3621
#define MEDITION_SECS 2.25
//indicador de estado de medición

static bool Measuring;
static uint32_t pulses;
static uint32_t StartTime;
static float Caudal;

bool IsMeasuringCaudal(void){
    uint32_t tiempo=OSA_TimeGetMsec()-StartTime;

    if(tiempo> MEDITION_SECS*1000){
        StopMeasuringCaudal();
        //get Caudal
        Caudal=pulses*CTE_PULSOS/(float)tiempo;
    }
    return Measuring;
}

void StartMeasuringCaudal(void){
    if(!IsMeasuringCaudal()){
        Measuring=true;
        pulses=0;
        StartTime=OSA_TimeGetMsec();
    }
}

void StopMeasuringCaudal(void){
    Measuring=false;
}

void NewCaudalPulse(void){
    uint32_t tiempo=OSA_TimeGetMsec()-StartTime;

    pulses++;

    if(tiempo> MEDITION_SECS*1000){
        StopMeasuringCaudal();
        //get Caudal
        Caudal=pulses*CTE_PULSOS/(float)tiempo;
    }
}

float GetCaudal(void){
    return Caudal*ganancia_caudal+offset_caudal;
}

/* Archivo de calibración de sensores:
 * Los sensores toman mediciones según la ecuación
 *          Valor_Corredigo=Valor_Medido*ganancia+offset
 */

// caudalímetro
static const float ganancia_caudal=1;
static const float offset_caudal=0;
```

'

```
// sensor de temperatura de entrada
static const float ganancia_temp_in=1;
static const float offset_temp_in=0;

// sensor de temperatura de salida
static const float ganancia_temp_out=1;
static const float offset_temp_out=0;

// anemómetro- velocidad del viento
static const float ganancia_vel_viento=1;
static const float offset_vel_viento=0;

// anemómetro- dirección del viento
static const float ganancia_dir_viento=1;
static const float offset_dir_viento=0;

// amperímetro
static const float ganancia_amperimetro=1;
static const float offset_amperimetro=0;

// sensor de radiación
static const float ganancia_radiacion=1;
static const float offset_radiacion=0;


/* ###################################################################
**     Filename    : Events.h
**     Project     : ColectorSolar
**     Processor   : MKL25Z128VLK4
**     Component   : Events
**     Version     : Driver 01.00
**     Compiler    : GNU C Compiler
**     Date/Time   : 2015-12-30, 10:26, # CodeGen: 0
**     Abstract    :
**         This is user's event module.
**         Put your event handler code here.
**     Settings    :
**     Contents    :
**         No public methods
**
** ###################################################################*/
/*!
** @file Events.h
** @version 01.00
** @brief
**         This is user's event module.
**         Put your event handler code here.
*/
/*!
**  @addtogroup Events_module Events module documentation
**  @{
*/

/* MODULE Events */

#include "fsl_device_registers.h"
```

'

```c
#include "clockMan1.h"
#include "pin_init.h"
#include "osa1.h"
#include "gpio1.h"
#include "tpmTmr1.h"
#include "adConv1.h"
#include "uartCom1.h"
#include "i2cCom1.h"
#include "memoryCard1.h"
#include "fsl_spi1.h"
#include "rtcTimer1.h"
#include "lpTmr1.h"
#include "pwrMan1.h"

/*! tpmTmr1 IRQ handler */
void tpmTmr1_IRQHandler(void);

void PIT_IRQHandler(void);

/*! adConv1 IRQ handler */
void ADC0_IRQHandler(void);


/*! UART Rx IRQ handler */
void uartCom1_RxCallback(uint32_t instance, void * uartState);

/*! UART Tx IRQ handler */
void uartCom1_TxCallback(uint32_t instance, void * uartState);


/*! PORTA IRQ handler */
void PORTA_IRQHandler(void);

/*! fsl_spi1 IRQ handler */
void fsl_spi1_IrqHandler(void);

/*! i2cCom1 IRQ handler */
void I2C0_IRQHandler(void);

/*! spiCom1 IRQ handler */
void SPI0_IRQHandler(void);

/*! i2cCom2 IRQ handler */
void I2C1_IRQHandler(void);

/*! RTC TIMER IRQ handler */
void rtcTimer1_IRQHandler(void);


/*! RTC TIMER  IRQ handler */
void RTC_Seconds_IRQHandler(void);


/*! LPTIMER IRQ handler */
void lpTmr1_OnTimerCompare(void);
```

```
/*! LLWU IRQ handler */
void LLWU_IRQHandler(void);

/* ifndef __Events_H*/
/*!
** @}
*/



/* ###################################################################
**     Filename    : Events.c

*/
/* MODULE Events */

#include "Cpu.h"
#include "Events.h"
#include "Viento.h"
#include "Caudal.h"
#include "uartCom1.h"
#include "fsl_uart_driver.h"


/*! Input Capture IRQ handler */
void tpmTmr1_IRQHandler(void)
{
  if(IsMeasuringWind())
    NewWindPulse();
 // uint32_t status=TPM_HAL_GetStatusRegVal(g_tpmBase[FSL_TPMTMR1]);
  TPM_HAL_ClearStatusReg(g_tpmBase[FSL_TPMTMR1],2);
}


/*! adConv1 IRQ handler */
void ADC0_IRQHandler(void)
{
}


/*! i2cCom1 IRQ handler */
void I2C0_IRQHandler(void)
{
  I2C_DRV_IRQHandler(FSL_I2CCOM1);
}


/*! fsl_spi1 IRQ handler */
void SPI0_IRQHandler(void)
{
#if FSL_SPI1_DMA_MODE
  SPI_DRV_DmaIRQHandler(FSL_FSL_SPI1);
#else
  SPI_DRV_IRQHandler(FSL_FSL_SPI1);
#endif
  /* Write your code here ... */
}
```

```
void rtcTimer1_IRQHandler(void)
{
    RTC_DRV_AlarmIntAction(FSL_RTCTIMER1);
    GPIO_DRV_SetPinOutput(LEDRGB_RED);
}

void RTC_Seconds_IRQHandler(void){
  /* Write your code here ... */
    //RTC_DRV_SecsIntAction(FSL_RTCTIMER1);

}

void lpTmr1_OnTimerCompare(void)
{
   if(IsMeasuringCaudal())
        NewCaudalPulse();

   GPIO_DRV_SetPinOutput(LEDRGB_RED);

}

void LLWU_IRQHandler(void)
{

}


void PORTA_IRQHandler(void)
{
  /* Clear interrupt flag.*/
  PORT_HAL_ClearPortIntFlag(PORTA_BASE_PTR);
}

/* END Events */
/*
 * Circular.h
 *
 *  Created on: 7/1/2016
 *      Author: julit
 */


#include "stdint.h"

#define BufferSize 2

//buffer circular que almacena BufferSize enteros de 32 bits
typedef struct{
    uint32_t medido[BufferSize];
    uint32_t pointer;
} uint32_tCircular;

//buffer circular que almacena BufferSize floats
typedef struct{
    float medido[BufferSize];
    uint32_t pointer;
```

'

```
} floatCircular;

//push al buffer circular (buff) de un elemento de 32 bits
void push_uint32(uint32_tCircular *buff, uint32_t elem);

//push al buffer circular (buff) de un elemento float
void push_float(floatCircular *buff, float elem);

//pull al buffer circular (buff) de un elemento de 32 bits
uint32_t pull_uint32(uint32_tCircular *buff);

//pull al buffer circular (buff) de un elemento float
float pull_float(floatCircular *buff);

//toma el valor del frente del buffer circular sin modificar los punteros del buffer
uint32_t front_uint32(uint32_tCircular *buff);

//toma el valor del frente del buffer circular sin modificar los punteros del buffer
float front_float(floatCircular *buff);

//pop al buffer circular (buff)
void pop_uint32(uint32_tCircular *buff);

//pop al buffer circular (buff)
void pop_float(floatCircular *buff);
 /* SOURCES_CIRCULAR_H_ */
/*
 * Circular.c
 *
 *  Created on: 7/1/2016
 *      Author: julit
 */
#include "Circular.h"



void push_uint32(uint32_tCircular *buff, uint32_t elem){
    buff->medido[buff->pointer]=elem;

    if(buff->pointer==BufferSize-1)
        buff->pointer=0;
    else
        buff->pointer++;

}

void push_float(floatCircular *buff, float elem){
    buff->medido[buff->pointer]=elem;

    if(buff->pointer==BufferSize-1)
        buff->pointer=0;
    else
        buff->pointer++;

}
```

```
                                                '

uint32_t pull_uint32(uint32_tCircular *buff){
    if(buff->pointer>BufferSize-1)
        return 0;

    return buff->medido[--buff->pointer];

}

float pull_float(floatCircular *buff){
    if(buff->pointer>BufferSize-1)
        return 0;

    return buff->medido[--buff->pointer];

}


uint32_t front_uint32(uint32_tCircular *buff){
    return buff->medido[0];
}

float front_float(floatCircular *buff){
    return buff->medido[0];
}

void pop_uint32(uint32_tCircular *buff){
    if(buff->pointer>0)
            buff->pointer--;
}

void pop_float(floatCircular *buff){
    if(buff->pointer>0)
        buff->pointer--;
}
```

12.5. Experiencias accesorias

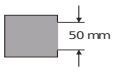
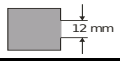
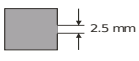
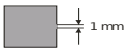
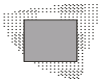
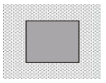
Protección según IEC

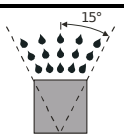
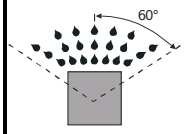
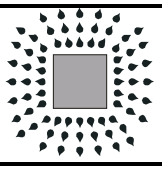
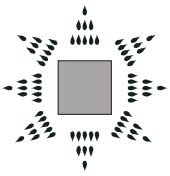
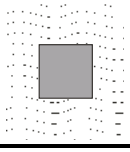
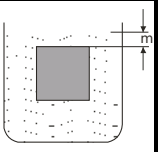
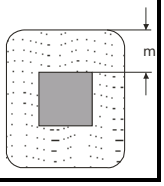
IEC 529 vs. NEMA

IEC 529 no especifica un grado de protección contra el riesgo de explosión o efectos producidos por la humedad (por ejemplo condensación), vapores corrosivos, hongos o insectos. NEMA no esta probado para condiciones ambientales como corrosión, oxidación, aceite y líquidos refrigerantes. Por esta razón, y debido a que las prueba y evaluaciones para otras características son totalmente diferentes, el sistema de clasificación de protecciones según IEC no puede ser exactamente comparada con los tipos de protección según NEMA.

La tabla que se muestra debajo de estas líneas provee una referencia cruzada entre los tipos de protección según NEMA y el sistema de clasificación de protecciones según IEC. Esta referencia cruzada es una aproximación basada en la información disponible sobre la performance de las pruebas y no esta sancionada por NEMA, IEC, VDE o cualquier agencia afiliada.

Para usar esta tabla, primero seleccione el tipo de protección según NEMA más apropiado, en la primer columna de la tabla, y luego lea a través de la fila y obtendrá el grado de protección IP correspondiente. No utilice esta tabla para la conversión del sistema de clasificación de protecciones según IEC al tipo de protección NEMA.

1° dígito	Grados de protección con respecto a cuerpos extraños y objetos sólidos	
0	Sin Protección	
1	Protección contra objetos sólidos con Ø 50 mm y superior	
2	Protección contra objetos sólidos con Ø 15.5 mm y superior	
3	Protección contra objetos sólidos con Ø 2.5 mm y superior	
4	Protección contra objetos sólidos con Ø 1.0 mm y superior	
5	Protección contra polvo a una presión de 200 mm de columna de agua	
6	Protección total contra polvo	

2° dígito	Grado de protección con respecto al ingreso perjudicial de agua	
0	Sin protección	
1	Protección contra caídas verticales de agua	
2	Protección contra caídas verticales de gotas de agua para una inclinación máxima de 15°	
3	Protección contra salpicaduras de agua de un ángulo hasta 60° en ambos lados	
4	Protección contra salpicaduras de agua en todas direcciones	
5	Protección contra chorros de agua a baja presión	
6	Protección contra agua que por momentos inunda el equipo por ejemplo mares embravecidos	
7	Protección contra inmersión	
8	Protección contra sumersión	

ROHM SEMICONDUCTOR

Technical Note

Ambient Light Sensor IC Series

Digital 16bit Serial Output Type Ambient Light Sensor IC

BH1750FVI

No.09046EBT01

● Descriptions

BH1750FVI is an digital Ambient Light Sensor IC for I^2C bus interface. This IC is the most suitable to obtain the ambient light data for adjusting LCD and Keypad backlight power of Mobile phone. It is possible to detect wide range at High resolution. (1 - 65535 lx).

● Features

1) I^2C bus Interface (f / s Mode Support)
2) Spectral responsibility is approximately human eye response
3) Illuminance to Digital Converter
4) Wide range and High resolution. (1 - 65535 lx)
5) Low Current by power down function
6) 50Hz / 60Hz Light noise reject-function
7) 1.8V Logic input interface
8) No need any external parts
9) Light source dependency is little. (ex. Incandescent Lamp. Fluorescent Lamp. Halogen Lamp. White LED. Sun Light)
10) It is possible to select 2 type of I^2C slave-address.
11) Adjustable measurement result for influence of optical window
(It is possible to detect min. 0.11 lx, max. 100000 lx by using this function.)
12) Small measurement variation (+/- 20%)
13) The influence of infrared is very small.

● Applications

Mobile phone, LCD TV, NOTE PC, Portable game machine, Digital camera, Digital video camera, Car navigation, PDA, LCD display

● Absolute Maximum Ratings

Parameter	Symbol	Limits	Units
Supply Voltage	Vmax	4.5	V
Operating Temperature	Topr	-40～85	℃
Storage Temperature	Tstg	-40～100	℃
SDA Sink Current	Imax	7	mA
Power Dissipation	Pd	260※	mW

※ 70mm × 70mm × 1.6mm glass epoxy board. Derating in done at 3.47mW/℃ for operating above Ta=25℃.

● Operating Conditions

Parameter	Symbol	Min.	Typ.	Max.	Units
Vcc Voltage	Vcc	2.4	3.0	3.6	V
I^2C Reference Voltage	VDVI	1.65	-	Vcc	V

●Electrical Characteristics (V_{CC} = 3.0V, DVI = 3.0V, Ta = 25℃, unless otherwise noted)

Parameter	Symbol	Min.	Typ.	Max.	Units	Conditions
Supply Current	Icc1	—	120	190	μA	Ev = 100 lx ※1
Powerdown Current	Icc2	—	0.01	1.0	μA	No input Light
Peak Wave Length	λp	—	560	—	nm	
Measurement Accuracy	S/A	0.96	1.2	1.44	times	Sensor out / Actual lx EV = 1000 lx ※1, ※2
Dark (0 lx) Sensor out	S0	0	0	3	count	H-Resolution Mode ※3
H-Resolution Mode Resolution	r_{HR}	—	1	—	lx	
L-Resolution Mode Resolution	r_{LR}	—	4	—	lx	
H-Resolution Mode Measurement Time	t_{HR}	—	120	180	ms	
L-Resolution Mode Measurement Time	t_{LR}	—	16	24	ms	
Incandescent / Fluorescent Sensor out ratio	r_{IF}	—	1	—	times	EV = 1000 lx
ADDR Input ‘H’ Voltage	V_{AH}	0.7 * VCC	—	—	V	
ADDR Input ‘L’ Voltage	V_{AL}	—	—	0.3 * VCC	V	
DVI Input ‘L’ Voltage	V_{DVL}	—	—	0.4	V	
SCL, SDA Input ‘H’ Voltage 1	$V_{IH}1$	0.7 * DVI	—	—	V	DVI ≧ 1.8V
SCL, SDA Input ‘H’ Voltage 2	$V_{IH}2$	1.26	—	—	V	1.65V ≦ DVI ＜1.8V
SCL, SDA Input ‘L’ Voltage 1	$V_{IL}1$	—	—	0.3 * DVI	V	DVI ≧ 1.8V
SCL, SDA Input ‘L’ Voltage 2	$V_{IL}2$	—	—	DVI – 1.26	V	1.65V ≦ DVI ＜ 1.8V
SCL, SDA, ADDR Input ‘H’ Current	I_{IH}	—	—	10	μA	
SCL, SDA, ADDR Input ‘L’ Current	I_{IL}	—	—	10	μA	
I^2C SCL Clock Frequency	f_{SCL}	—	—	400	kHz	
I^2C Bus Free Time	t_{BUF}	1.3	—	—	μs	
I^2C Hold Time (repeated) START Condition	t_{HDSTA}	0.6	—	—	μs	
I^2C Set up time for a Repeated START Condition	t_{SUSTA}	0.6	—	—	μs	
I^2C Set up time for a Repeated STOP Condition	t_{SUSTD}	0.6	—	—	μs	
I^2C Data Hold Time	t_{HDDAT}	0	—	0.9	μs	
I^2C Data Setup Time	t_{SUDAT}	100	—	—	ns	
I^2C ‘L’ Period of the SCL Clock	t_{LOW}	1.3	—	—	μs	
I^2C ‘H’ Period of the SCL Clock	t_{HIGH}	0.6	—	—	μs	
I^2C SDA Output ‘L’ Voltage	V_{OL}	0	—	0.4	V	I_{OL} = 3 mA

※1 White LED is used as optical source.
※2 Measurement Accuracy typical value is possible to change '1' by "Measurement result adjustment function".
※3 Use H-resolution mode or H-resolution mode2 if dark data (less than 10 lx) is need.

●Reference Data

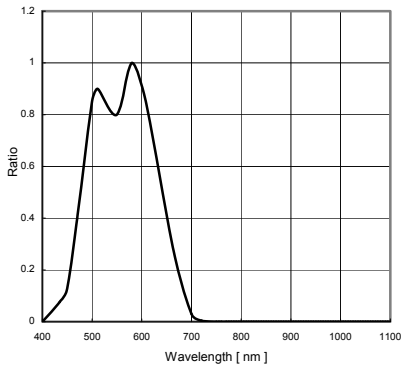


Fig.1 Spectral Response

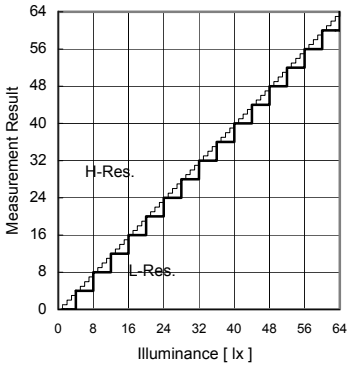


Fig.2 Illuminance - Measurement Result 1

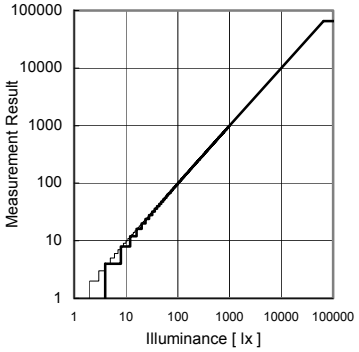


Fig.3 Illuminance - Measuremnet Result 2

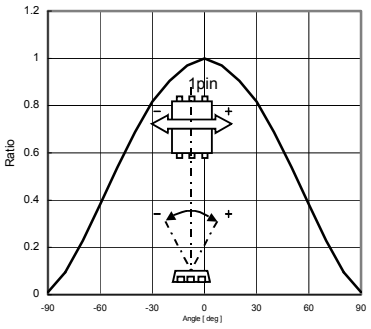


Fig.4 Directional Characteristics 1

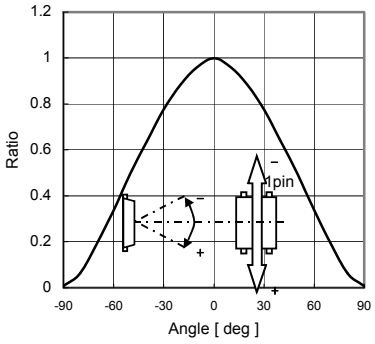


Fig.5 Directional Characteristics 2

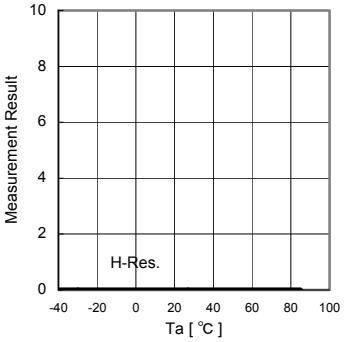


Fig.6 Dark Response

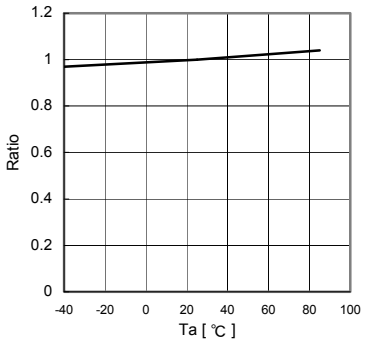


Fig.7 Measurement Accuracy Temperature Dependency

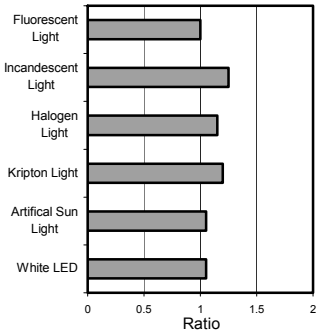


Fig.8 Light Source Dependency (Fluorescent Light is set to '1')

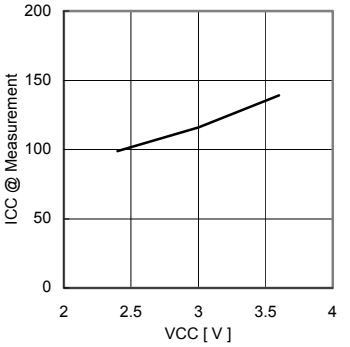


Fig.9 VCC - ICC (During measurement)

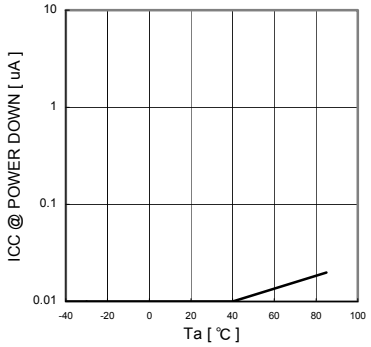


Fig.10 VCC – ICC@0 Lx (POWER DOWN)

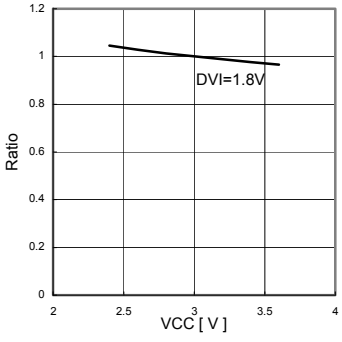


Fig.11 Measurement Result VCC Dependency

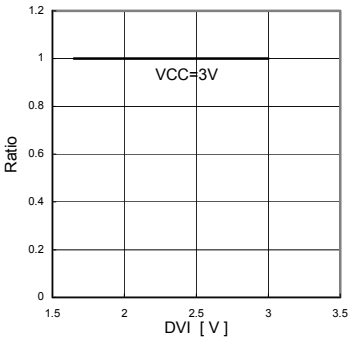
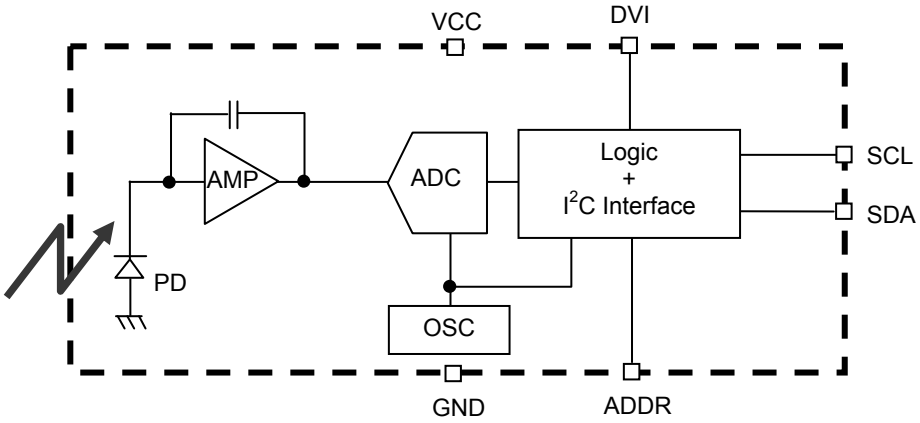


Fig.12 Measurement Result DVI Dependency

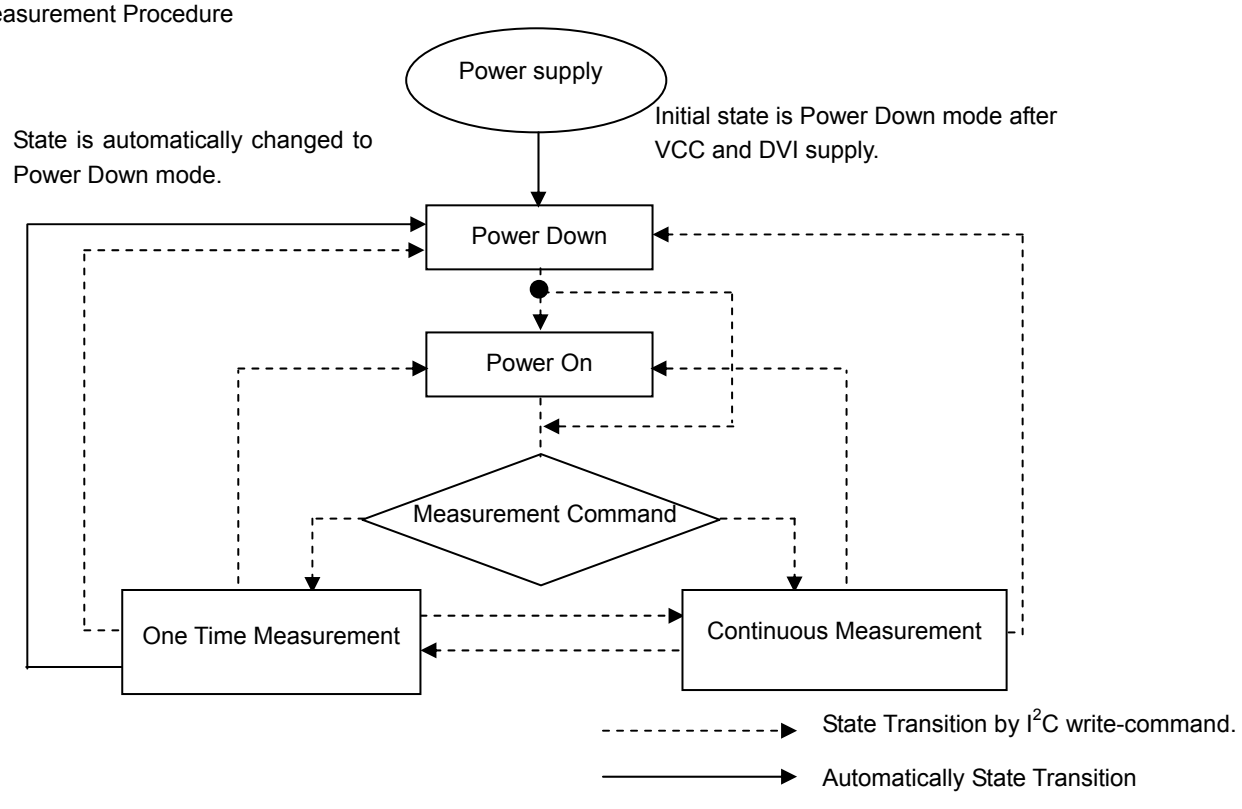
●Block Diagram



●Block Diagram Descriptions

・PD
Photo diode with approximately human eye response.
・AMP
Integration-OPAMP for converting from PD current to Voltage.
・ADC
AD converter for obtainment Digital 16bit data.
・Logic + I^2C Interface
Ambient Light Calculation and I^2C BUS Interface. It is including below register.
Data Register → This is for registration of Ambient Light Data. Initial Value is "0000_0000_0000_0000".
Measurement Time Register → This is for registration of measurement time. Initial Value is "0100_0101".
・OSC
Internal Oscillator (typ. 320kHz). It is CLK for internal logic.

●Measurement Procedure



* "Power On" Command is possible to omit.

●Instruction Set Architecture

Instruction	Opecode	Comments
Power Down	0000_0000	No active state.
Power On	0000_0001	Waiting for measurement command.
Reset	0000_0111	Reset Data register value. Reset command is not acceptable in Power Down mode.
Continuously H-Resolution Mode	0001_0000	Start measurement at 1lx resolution. Measurement Time is typically 120ms.
Continuously H-Resolution Mode2	0001_0001	Start measurement at 0.5lx resolution. Measurement Time is typically 120ms.
Continuously L-Resolution Mode	0001_0011	Start measurement at 4lx resolution. Measurement Time is typically 16ms.
One Time H-Resolution Mode	0010_0000	Start measurement at 1lx resolution. Measurement Time is typically 120ms. It is automatically set to Power Down mode after measurement.
One Time H-Resolution Mode2	0010_0001	Start measurement at 0.5lx resolution. Measurement Time is typically 120ms. It is automatically set to Power Down mode after measurement.
One Time L-Resolution Mode	0010_0011	Start measurement at 4lx resolution. Measurement Time is typically 16ms. It is automatically set to Power Down mode after measurement.
Change Measurement time (High bit)	01000_MT[7,6,5]	Change measurement time. ※ Please refer "adjust measurement result for influence of optical window."
Change Masurement time (Low bit)	011_MT[4,3,2,1,0]	Change measurement time. ※ Please refer "adjust measurement result for influence of optical window."

※ Don't input the other opecode.

●Measurement mode explanation

Measurement Mode	Measurement Time.	Resolurtion
H-resolution Mode2	Typ. 120ms.	0.5 lx
H-Resolution Mode	Typ. 120ms.	1 lx.
L-Resolution Mode	Typ. 16ms.	4 lx.

We recommend to use H-Resolution Mode.
Measurement time (integration time) of H-Resolution Mode is so long that some kind of noise(including in 50Hz / 60Hz noise) is rejected. And H-Resolution Mode is 1 l x resolution so that it is suitable for darkness (less than 10 lx)
H-resolution mode2 is also suitable to detect for darkness.

●Explanation of Asynchronous reset and Reset command "0000_0111"

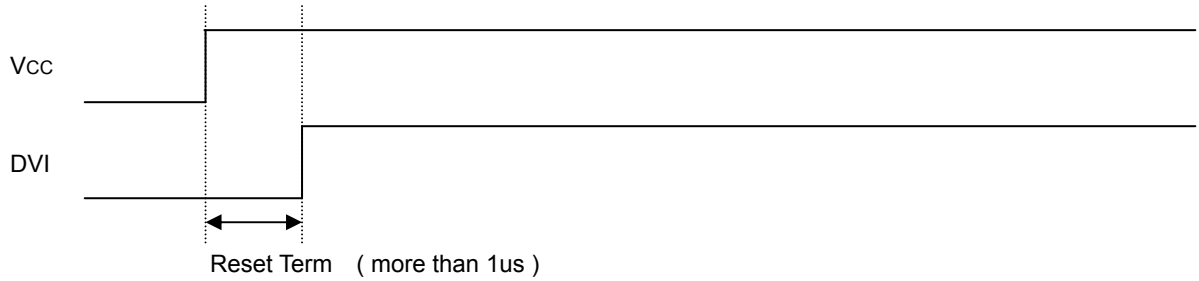
1) Asynchronous reset
All registers are reset. It is necessary on power supply sequence. Please refer "Timing chart for VCC and DVI power supply sequence" in this page. It is power down mode during DVI = 'L'.

2) Reset command
Reset command is for only reset Illuminance data register. (reset value is '0') It is not necessary even power supply sequence.It is used for removing previous measurement result. This command is not working in power down mode, so that please set the power on mode before input this command.

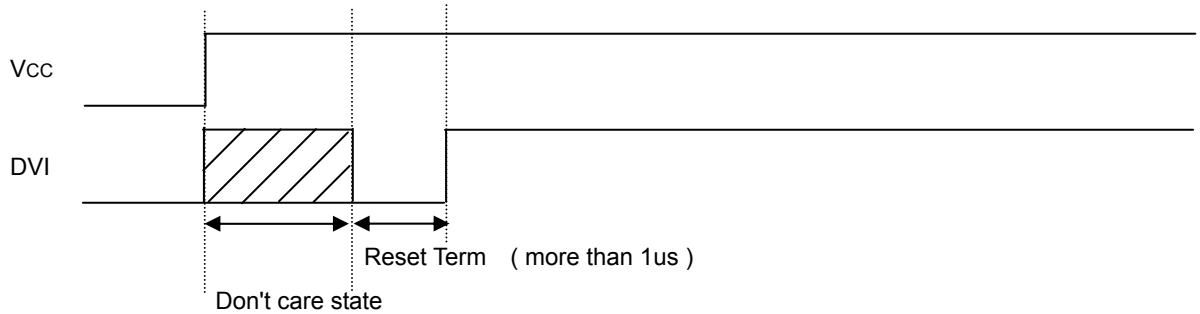
●Timing chart for VCC and DVI power supply sequence

DVI is I^2C bus reference voltage terminal. And it is also asynchronous reset terminal. It is necessary to set to 'L' after VCC is supplied. In DVI 'L' term, internal state is set to Power Down mode.

1) Recommended Timing chart1 for VCC and DVI supply.



2) Timing chart2 for VCC and DVI supply.
(If DVI rises within 1μs after VCC supply)



Don't care state
ADDR, SDA, SCL is not stable if DVI 'L' term (1us) is not given by systems.
In this case, please connect the resisters (approximately 100kOhm) to ADDR without directly connecting to VCC or GND,
because it is 3 state buffer for Internal testing.

●Measurement sequence example from "Write instruction" to "Read measurement result"

ex1) Continuously H-resolution mode (ADDR = 'L')

from Master to Slave

from Slave to Master

① Send "Continuously H-resolution mode " instruction

ST	0100011	0	Ack	00010000	Ack	SP

② Wait to complete 1st H-resolution mode measurement.(max. 180ms.)

③ Read measurement result.

ST	0100011	1	Ack	High Byte [15:8]	Ack

Low Byte [7:0]	$\overline{\text{Ack}}$	SP

How to calculate when the data High Byte is "10000011" and Low Byte is "10010000"
($2^{15} + 2^9 + 2^8 + 2^7 + 2^4$) / 1.2 ≒ 28067 [lx]

The result of continuously measurement mode is updated.(120ms.typ at H-resolution mode, 16ms.typ at L-resolution mode)

ex2) One time L-resolution mode (ADDR = 'H')

① Send "One time L-resolution mode " instruction

ST	1011100	0	Ack	00100011	Ack	SP

② Wait to complete L-resolution mode measurement.(max. 24ms.)

③ Read measurement result

ST	1011100	1	Ack	High Byte [15:8]	Ack

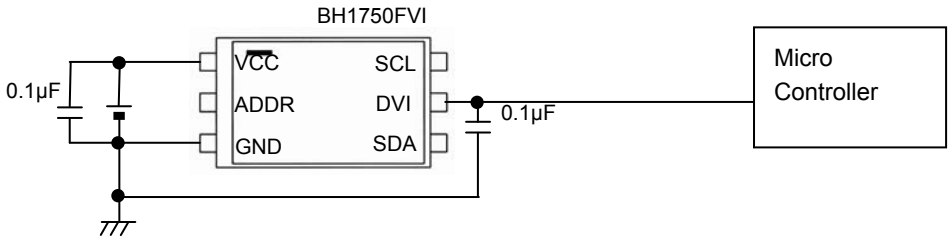
Low Byte [7:0]	$\overline{\text{Ack}}$	SP

How to calculate when the data High Byte is "00000001" and Low Byte is "00010000"
($2^8 + 2^4$) / 1.2 ≒ 227 [lx]
In one time measurement, Statement moves to power down mode after measurement completion.If updated result is need then please resend measurement instruction.

●Application circuit example of DVI terminal

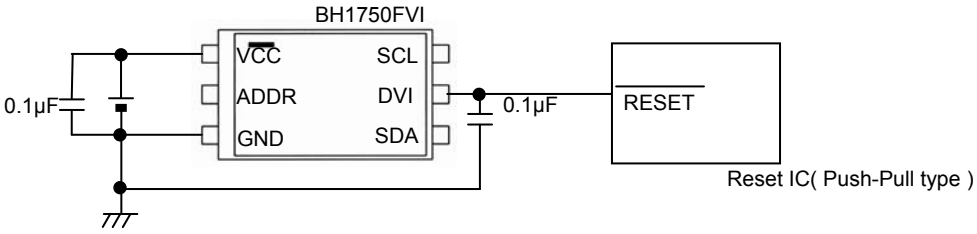
The DVI terminal is an asynchronous reset terminal. Please note that there is a possibility that IC doesn't operate normally if the reset section is not installed after the start-up of VCC.
(Please refer to the paragraph of "Timing chart for VCC and DVI power supply sequence")
The description concerning SDA and the terminal SCL is omitted in this application circuit example. Please design the application the standard of the I2C bus as it finishes being satisfactory. Moreover, the description concerning the terminal ADDR is omitted. Please refer to the paragraph of "Timing chart for VCC and DVI power supply sequence" about the terminal ADDR design.

ex 1) The control signal line such as CPU is connected.

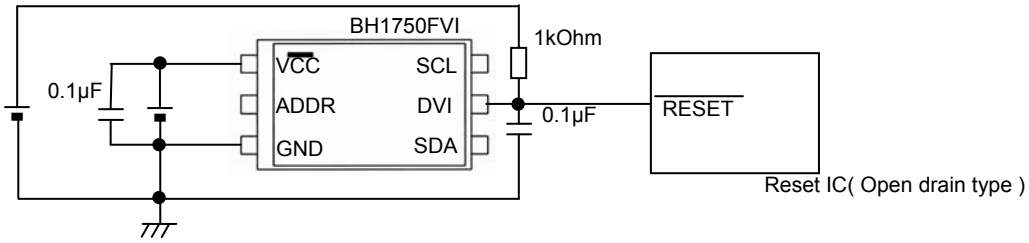


ex 2) Reset IC is used.

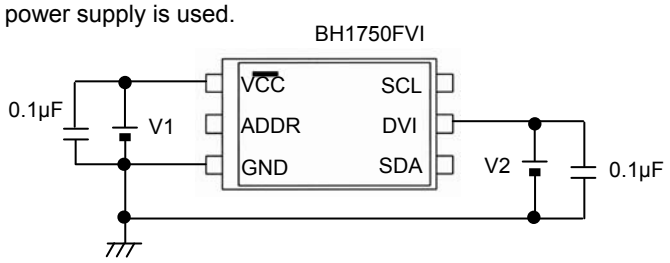
1, For Reset IC of the Push-Pull type



2, For Reset IC of the Open drain output

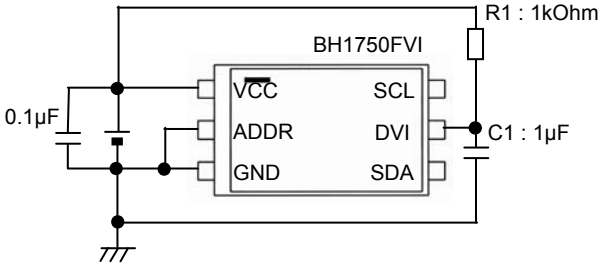


ex 3) A different power supply is used.



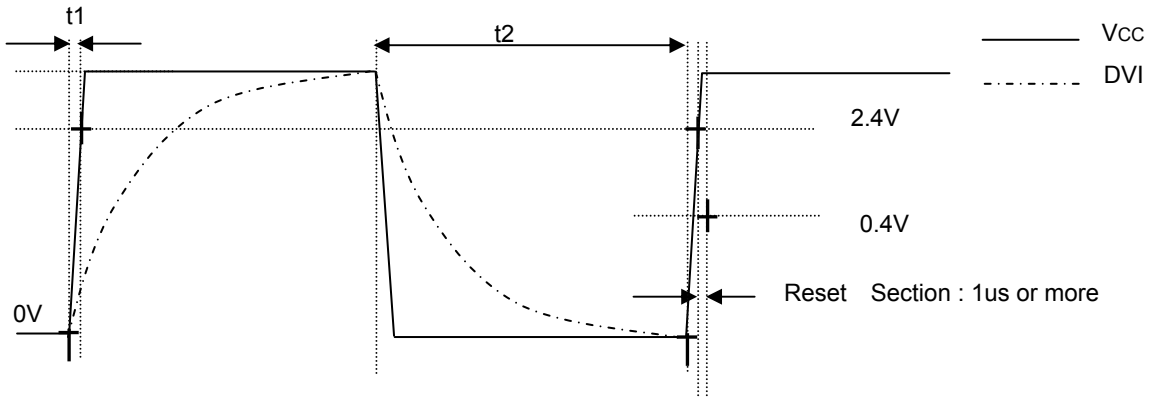
※ Power supply of DVI must stand up later than power supply of VCC stand up, because it is necessary to secure reset section (1μs or more).

ex 4) LPF using CR is inserted between VCC and DVI.
This method has the possibility that the Reset section of turning on the power supply can not satisfied. cannot be satisfied.
Please design the set considering the characteristic of the power supply enough.



◆ Notes when CR is inserted between VCC and DVI

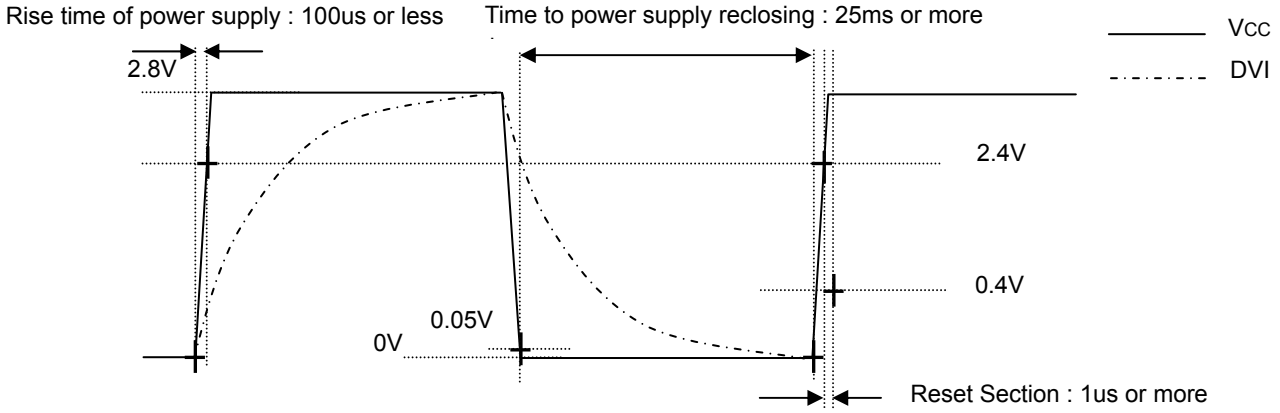
※ Please note that there is a possibility that reset section (1μs) can not be satisfied because the power supply is turned on when the rise time of VCC is slow
※ When VCC is turned off, the DVI voltage becomes higher than VCC voltage but IC destruction is not occred if recommended constant (R1 = 1kOhm, C1 = 1μF) is used.
※ Please note that there is a possibility that Reset section (1μsec) cannot be satisfied if wait time is not enough long after turning off VCC. (It is necessary to consider DVI voltage level after turning off VCC.)



* Please do the application design to secure Reset section 1us or more after the reclosing of the power supply.

◆ Example of designing set when CR (C = 1μF, R = 1kΩ) is inserted between VCC and DVI with VCC=2.8V

①The rise time to 0→2.4V of VCC must use the power supply of 100μs or less.
②Please wait 25ms or more after VCC turn off (VCC <= 0.05V), because it is necessary to secure reset section (1μs or more).

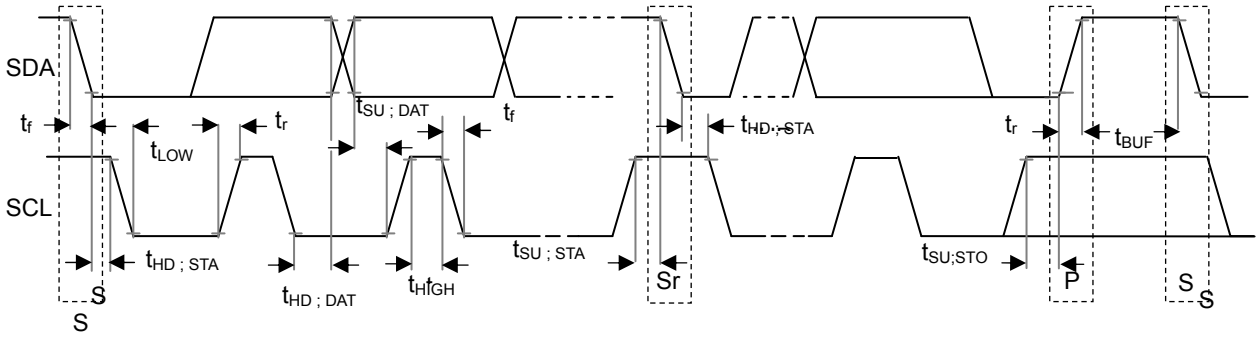


* Please do the application design to secure Reset section 1us or more after the reclosing of the power supply.

●I^2C Bus Access

1) I^2C Bus Interface Timing chart

Write measurement command and Read measurement result are done by I^2C Bus interface. Please refer the formally specification of I^2C Bus interface, and follow the formally timing chart.



2) Slave Address

Slave Address is 2 types, it is determined by ADDR Terminal

ADDR = 'H' (ADDR ≧ 0.7VCC) → "1011100"

ADDR = 'L' (ADDR ≦ 0.3VCC) → "0100011"

3) Write Format

BH1750FVI is not able to accept plural command without stop condition. Please insert SP every 1 Opecode.

ST	Slave Address	R/W 0	Ack	Opecode	Ack	SP

4) Read Format

ST	Slave Address	R/W 1	Ack	High Byte [15:8] 2^{15} 2^{14} 2^{13} 2^{12} 2^{11} 2^{10} 2^9 2^8	Ack

Low Byte [7:0] 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0	$\overline{Ack}$	SP

from Master to Slave

from Slave to Master

ex)

High Byte = "1000_0011"

Low Byte = "1001_0000"

($2^{15} + 2^9 + 2^8 + 2^7 + 2^4$) / 1.2 ≒ 28067 [lx]

* I^2C BUS is trademark of Phillips Semiconductors. Please refer formality specification.

●Adjust measurement result for influence of optical window. (sensor sensitivity adjusting)

BH1750FVI is possible to change sensor sensitivity. And it is possible to cancel the optical window influence (difference with / without optical window) by using this function. Adjust is done by changing measurement time. For example, when transmission rate of optical window is 50% (measurement result becomes 0.5 times if optical window is set), influence of optical window is ignored by changing sensor sensitivity from default to 2 times

Sensor sensitivity is shift by changing the value of MTreg (measurement time regisiter). MTreg value has to set 2 times if target of sensor sensitivity is 2 times. Measurement time is also set 2 times when MTreg value is changed from default to 2 times.

ex) Procedure for changing target sensor sensitivity to 2 times.

Please change Mtreg from ”0100_0101” (default) to ”1000_1010” (default * 2).

1) Changing High bit of MTreg

ST	Slave Address	R/W 0	Ack	01000_100	Ack	SP

2) Changing Low bit of MTreg

ST	Slave Address	R/W 0	Ack	011_01010	Ack	SP

3) Input Measurement Command

ST	Slave Address	R/W 0	Ack	0001_0000	Ack	SP

* This example is High Resolution mode, but it accepts the other measurement.

4) After about 240ms, measurement result is registered to Data Register.
(High Resolution mode is typically 120ms, but measurement time is set twice.)

The below table is seeing the changable range of MTreg.

		Min.	Typ.	Max.
changable range of MTreg	binary	0001_1111 (sensitivity : default * 0.45)	0100_0101 default	1111_1110 (sensitivity : default * 3.68)
	decimal	31 (sensitivity : default * 0.45)	69 default	254 (sensitivity : default * 3.68)

It is possilbe to detect 0.23lx by using this function at H-resolution mode. And it is possilbe to detect 0.11lx by using this function at H-resolution mode2.

The below formula is to calculate illuminance per 1 count.

H-reslution mode : Illuminance per 1 count (lx / count) = 1 / 1.2 *(69 / X)
H-reslution mode2 : Illuminance per 1 count (lx / count) = 1 / 1.2 *(69 / X) / 2

1.2 : Measurement accuracy
69 : Default value of MTreg (dec)
X : MTreg value

The below table is seeing the detail of resolution.

Mtreg の値	lx / count at H-resolutin mode	lx / count at H-resolution mode2
0001_1111	1.85	0.93
0100_0101	0.83	0.42
1111_1110	0.23	0.11

●H-Resolution Mode2

H-resolution mode2 is 0.5lx (typ.) resolution mode. It is suitable if under less than 10 lx measure ment data is necessary. This measurement mode supports " Adjust measurement result for influence of optical window ". Please refer it. It is possible to detect min. 0.11 lx by using H-resolution mode2.

○ Instruction set architecture for H-resolution mode2

Instruction	Opecode	Comments
Continuously H-Resolution Mode2	0001_0001	Start measurement at 0.5lx resolution. Measurement Time is typically 120ms.
One Time H-Resolution Mode2	0010_0001	Start measurement at 0.5lx resolution. Measurement Time is typically 120ms. It is automatically set to Power Down mode after measurement.

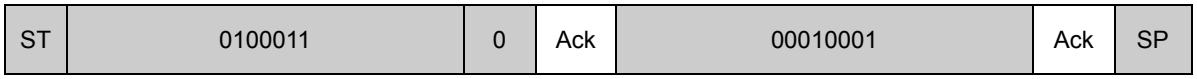
○ Measurement sequence example from "Write instruction" to "Read measurement result"

ex) Continuously H-resolution mode2 (ADDR = 'L')

from Master to Slave

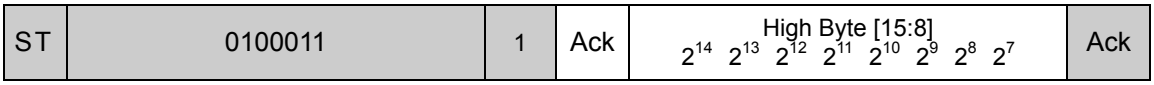
from Slave to Master

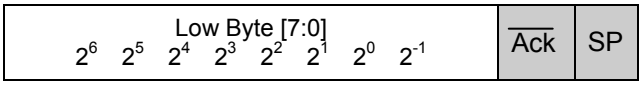
① Send "Continuously H-resolution mode2 " instruction



② Wait to complete 1st H-resolution mode2 measurement.(max. 180ms.)

③ Read measurement result.

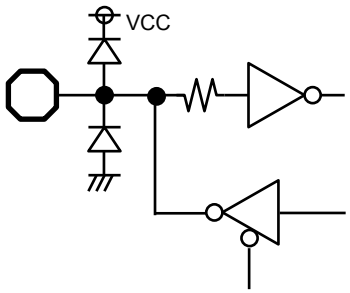
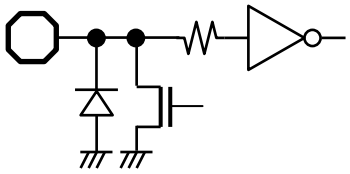
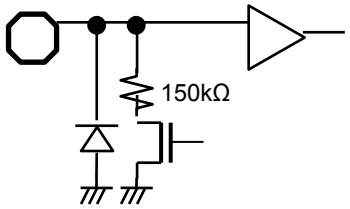
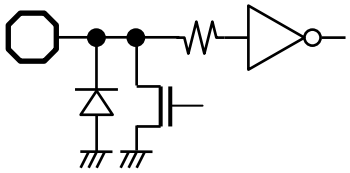




How to calculate when the data High Byte is "00000000" and Low Byte is "00010010"

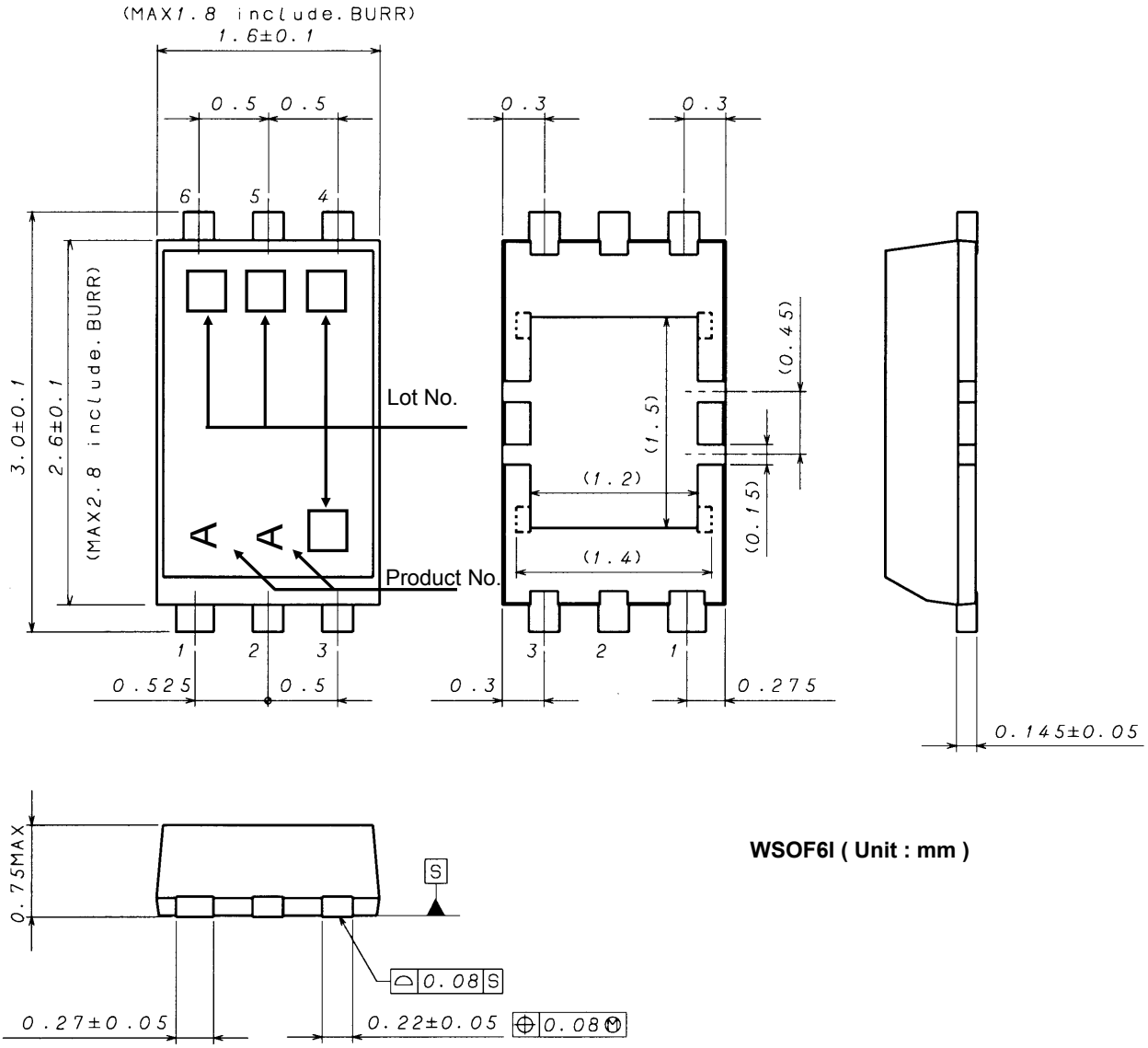
$(2^{3} + 2^{0}) / 1.2 \fallingdotseq 7.5$ [lx]

●Terminal Description

PIN No.	Terminal Name	Equivalent Circuit	Function
1	VCC		Power Supply Terminal
2	ADDR		I^2C Slave-address Terminal ADDR = 'H' (ADDR ≧ 0.7Vcc) "1011100" ADDR = 'L' (ADDR ≦ 0.3Vcc) "0100011" ADDR Terminal is designed as 3 state buffer for internal test. So that please take care of Vcc and DVI supply procedure.Please see P6.
3	GND		GND Terminal
4	SDA		I^2C bus Interface SDA Terminal
5	DVI		SDA, SCL Reference Voltage Terminal And DVI Terminal is also asynchronous Reset for internal registers.So that please set to 'L' (at least 1μs, DVI <= 0.4V) after Vcc is supplied. BH1750FVI is pulled down by 150kOhm while DVI = 'L'.
6	SCL		I^2C bus Interface SCL Terminal

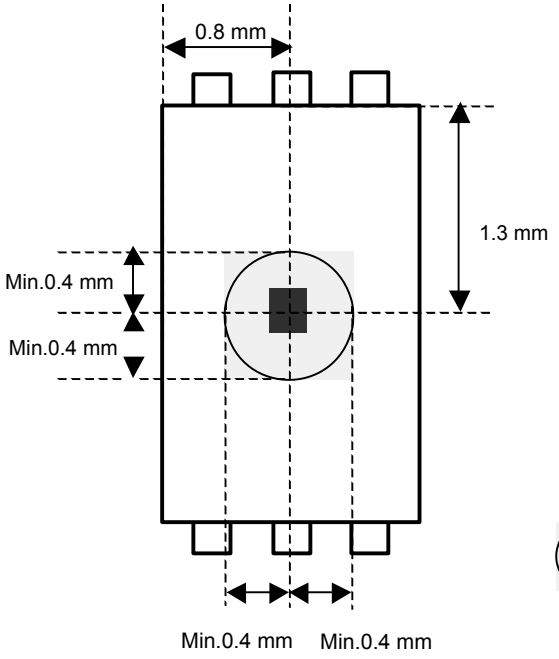
※These values are design-value, not guaranteed.

●Package Outlines



WSOF6I (Unit : mm)

●About an optical design on the device



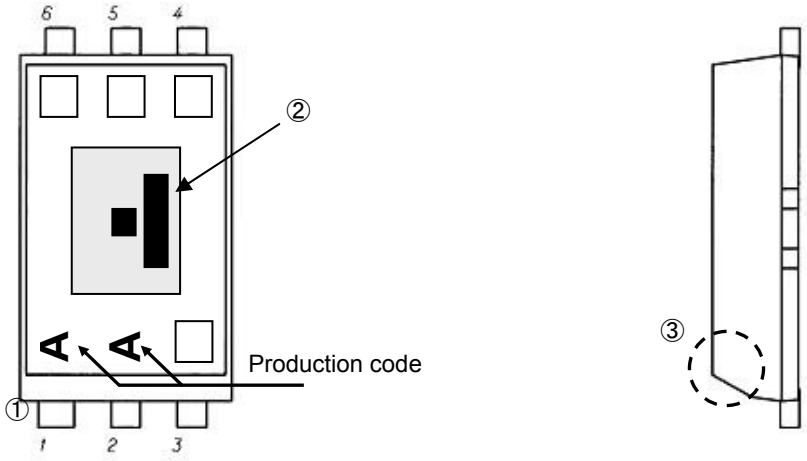
PD area (0.25 mm x 0.3 mm)

Please design the optical window so that light can cover at least this area.

●The method of distinguishing 1pin.

There is some method of distinguishing 1pin.
① Distinguishing by 1Pin wide-lead
② Distinguishing by die pattern
③ Distinguishing by taper part of 1-3pin side

② (by die patern) is the easiest method to distinguish by naked eye.



●Cautions on use

1) Absolute Maximum Ratings
An excess in the absolute maximum ratings, such as supply voltage (Vmax), temperature range of operating conditions (Topr), etc., can break down devices, thus making impossible to identify breaking mode such as a short circuit or an open circuit. If any special mode exceeding the absolute maximum ratings is assumed, consideration should be given to take physical safety measures including the use of fuses, etc.

2) GND voltage
Make setting of the potential of the GND terminal so that it will be maintained at the minimum in any operating state. Furthermore, check to be sure no terminals are at a potential lower than the GND voltage including an actual electric transient.

3) Short circuit between terminals and erroneous mounting
In order to mount ICs on a set PCB, pay thorough attention to the direction and offset of the ICs. Erroneous mounting can break down the ICs. Furthermore, if a short circuit occurs due to foreign matters entering between terminals or between the terminal and the power supply or the GND terminal, the ICs can break down.

4) Operation in strong electromagnetic field
Be noted that using ICs in the strong electromagnetic field can malfunction them.

5) Inspection with set PCB
On the inspection with the set PCB, if a capacitor is connected to a low-impedance IC terminal, the IC can suffer stress. Therefore, be sure to discharge from the set PCB by each process. Furthermore, in order to mount or dismount the set PCB to/from the jig for the inspection process, be sure to turn OFF the power supply and then mount the set PCB to the jig. After the completion of the inspection, be sure to turn OFF the power supply and then dismount it from the jig. In addition, for protection against static electricity, establish a ground for the assembly process and pay thorough attention to the transportation and the storage of the set PCB.

6) Input terminals
In terms of the construction of IC, parasitic elements are inevitably formed in relation to potential. The operation of the parasitic element can cause interference with circuit operation, thus resulting in a malfunction and then breakdown of the input terminal. Therefore, pay thorough attention not to handle the input terminals; such as to apply to the input terminals a voltage lower than the GND respectively, so that any parasitic element will operate. Furthermore, do not apply a voltage to the input terminals when no power supply voltage is applied to the IC. In addition, even if the power supply voltage is applied, apply to the input terminals a voltage lower than the power supply voltage or within the guaranteed value of electrical characteristics.

7) Thermal design
Perform thermal design in which there are adequate margins by taking into account the power dissipation (Pd) in actual states of use.

8) Treatment of package
Dusts or scratch on the photo detector may affect the optical characteristics. Please handle it with care.

9) Rush current
When power is first supplied to the CMOS IC, it is possible that the internal logic may be unstable and rush current may flow instantaneously. Therefore, give special consideration to power coupling capacitance, power wiring, width of GND wiring, and routing of connections.

10) The exposed central pad on the back side of the package
There is an exposed central pad on the back side of the package. But please do it non connection. (Don't solder, and don't do electrical connection) Please mount by Footprint dimensions described in the Jisso Information for WSOF6I. This pad is GND level, therefore there is a possibility that LSI malfunctions and heavy-current is generated.

●Ordering part number

B	H	1	7	5	0	F	V	I	-	T	R

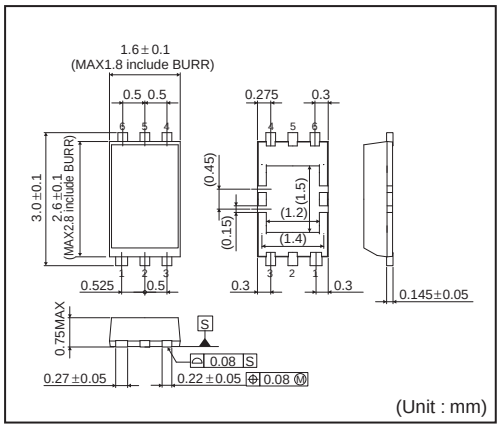
Part No. (BH)

Part No. (1750)

Package
FVI: WSOF6I

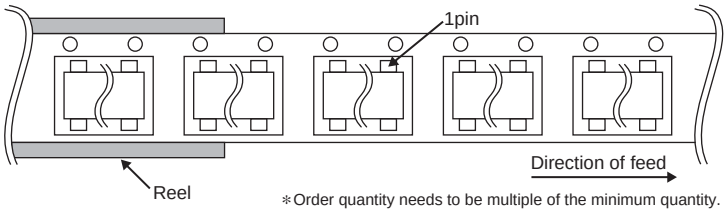
Packaging and forming specification
TR: Embossed tape and reel

WSOF6I



<Tape and Reel information>

Tape	Embossed carrier tape
Quantity	3000pcs
Direction of feed	TR (The direction is the 1pin of product is at the upper right when you hold reel on the left hand and you pull out the tape on the right hand)



∗Order quantity needs to be multiple of the minimum quantity.

Notes

No copying or reproduction of this document, in part or in whole, is permitted without the consent of ROHM Co.,Ltd.

The content specified herein is subject to change for improvement without notice.

The content specified herein is for the purpose of introducing ROHM's products (hereinafter "Products"). If you wish to use any such Product, please be sure to refer to the specifications, which can be obtained from ROHM upon request.

Examples of application circuits, circuit constants and any other information contained herein illustrate the standard usage and operations of the Products. The peripheral conditions must be taken into account when designing circuits for mass production.

Great care was taken in ensuring the accuracy of the information specified in this document. However, should you incur any damage arising from any inaccuracy or misprint of such information, ROHM shall bear no responsibility for such damage.

The technical information specified herein is intended only to show the typical functions of and examples of application circuits for the Products. ROHM does not grant you, explicitly or implicitly, any license to use or exercise intellectual property or other rights held by ROHM and other parties. ROHM shall bear no responsibility whatsoever for any dispute arising from the use of such technical information.

The Products specified in this document are intended to be used with general-use electronic equipment or devices (such as audio visual equipment, office-automation equipment, communication devices, electronic appliances and amusement devices).

The Products specified in this document are not designed to be radiation tolerant.

While ROHM always makes efforts to enhance the quality and reliability of its Products, a Product may fail or malfunction for a variety of reasons.

Please be sure to implement in your equipment using the Products safety measures to guard against the possibility of physical injury, fire or any other damage caused in the event of the failure of any Product, such as derating, redundancy, fire control and fail-safe designs. ROHM shall bear no responsibility whatsoever for your use of any Product outside of the prescribed scope or not in accordance with the instruction manual.

The Products are not designed or manufactured to be used with any equipment, device or system which requires an extremely high level of reliability the failure or malfunction of which may result in a direct threat to human life or create a risk of human injury (such as a medical instrument, transportation equipment, aerospace machinery, nuclear-reactor controller, fuel-controller or other safety device). ROHM shall bear no responsibility in any way for use of any of the Products for the above special purposes. If a Product is intended to be used for any such special purpose, please contact a ROHM sales representative before purchasing.

If you intend to export or ship overseas any Product or technology specified herein that may be controlled under the Foreign Exchange and the Foreign Trade Law, you will be required to obtain a license or permit under the Law.

ROHM SEMICONDUCTOR

Thank you for your accessing to ROHM product informations.
More detail product informations and catalogs are available, please contact us.

ROHM Customer Support System

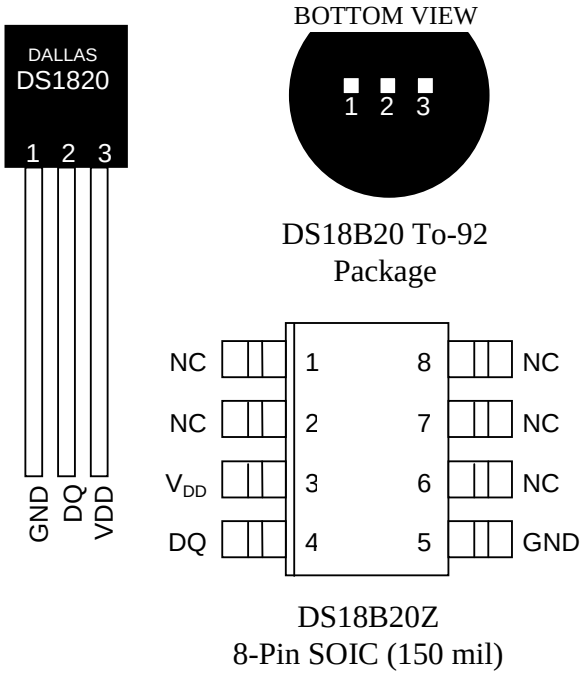
http://www.rohm.com/contact/

DS18B20
Programmable Resolution 1-Wire® Digital Thermometer

FEATURES

- Unique 1-Wire interface requires only one port pin for communication
- Multidrop capability simplifies distributed temperature sensing applications
- Requires no external components
- Can be powered from data line. Power supply range is 3.0V to 5.5V
- Zero standby power required
- Measures temperatures from -55°C to +125°C. Fahrenheit equivalent is -67°F to +257°F
- ±0.5°C accuracy from -10°C to +85°C
- Thermometer resolution is programmable from 9 to 12 bits
- Converts 12-bit temperature to digital word in 750 ms (max.)
- User-definable, nonvolatile temperature alarm settings
- Alarm search command identifies and addresses devices whose temperature is outside of programmed limits (temperature alarm condition)
- Applications include thermostatic controls, industrial systems, consumer products, thermometers, or any thermally sensitive system

PIN ASSIGNMENT



DS18B20 To-92 Package

DS18B20Z
8-Pin SOIC (150 mil)

PIN DESCRIPTION

GND - Ground
DQ - Data In/Out
V_{DD} - Power Supply Voltage
NC - No Connect

DESCRIPTION

The DS18B20 Digital Thermometer provides 9 to 12-bit (configurable) temperature readings which indicate the temperature of the device.

Information is sent to/from the DS18B20 over a 1-Wire interface, so that only one wire (and ground) needs to be connected from a central microprocessor to a DS18B20. Power for reading, writing, and performing temperature conversions can be derived from the data line itself with no need for an external power source.

Because each DS18B20 contains a unique silicon serial number, multiple DS18B20s can exist on the same 1-Wire bus. This allows for placing temperature sensors in many different places. Applications where this feature is useful include HVAC environmental controls, sensing temperatures inside buildings, equipment or machinery, and process monitoring and control.

DETAILED PIN DESCRIPTION Table 1

PIN 8PIN SOIC	PIN TO92	SYMBOL	DESCRIPTION
5	1	GND	Ground.
4	2	DQ	**Data Input/Output pin.** For 1-Wire operation: Open drain. (See "Parasite Power" section.)
3	3	V_{DD}	**Optional V_{DD} pin.** See "Parasite Power" section for details of connection. V_{DD} must be grounded for operation in parasite power mode.

DS18B20Z (8-pin SOIC): All pins not specified in this table are not to be connected.

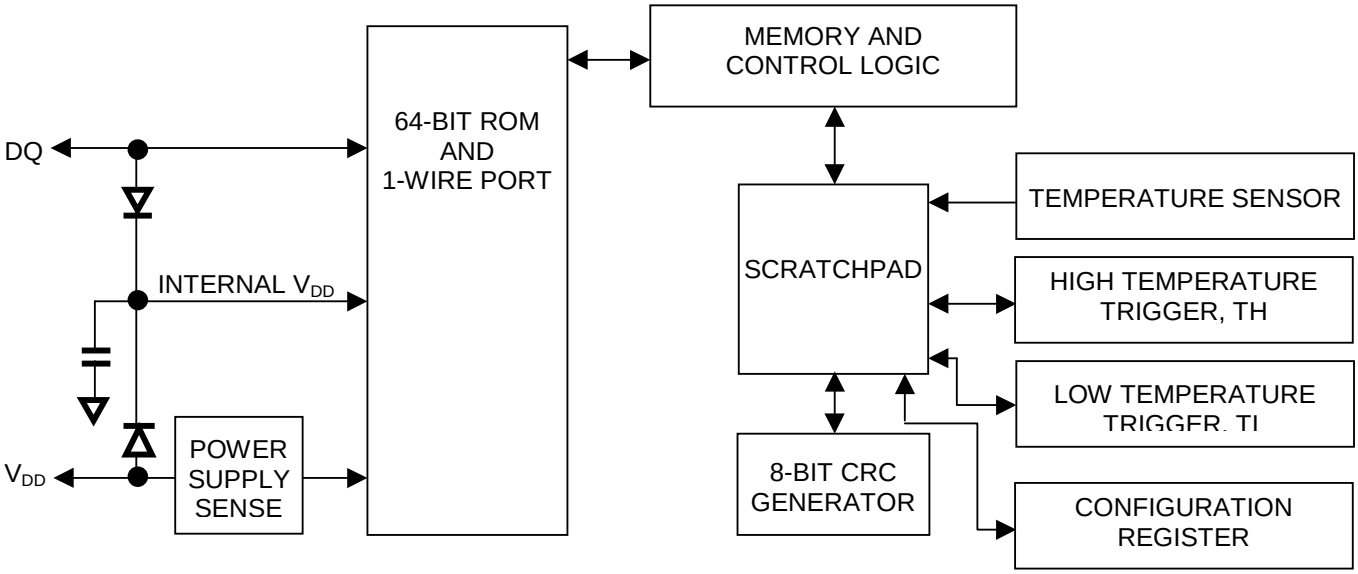
OVERVIEW

The block diagram of Figure 1 shows the major components of the DS18B20. The DS18B20 has four main data components: 1) 64-bit lasered ROM, 2) temperature sensor, 3) nonvolatile temperature alarm triggers TH and TL, and 4) a configuration register. The device derives its power from the 1-Wire communication line by storing energy on an internal capacitor during periods of time when the signal line is high and continues to operate off this power source during the low times of the 1-Wire line until it returns high to replenish the parasite (capacitor) supply. As an alternative, the DS18B20 may also be powered from an external 3 volt - 5.5 volt supply.

Communication to the DS18B20 is via a 1-Wire port. With the 1-Wire port, the memory and control functions will not be available before the ROM function protocol has been established. The master must first provide one of five ROM function commands: 1) Read ROM, 2) Match ROM, 3) Search ROM, 4) Skip ROM, or 5) Alarm Search. These commands operate on the 64-bit lasered ROM portion of each device and can single out a specific device if many are present on the 1-Wire line as well as indicate to the bus master how many and what types of devices are present. After a ROM function sequence has been successfully executed, the memory and control functions are accessible and the master may then provide any one of the six memory and control function commands.

One control function command instructs the DS18B20 to perform a temperature measurement. The result of this measurement will be placed in the DS18B20's scratch-pad memory, and may be read by issuing a memory function command which reads the contents of the scratchpad memory. The temperature alarm triggers TH and TL consist of 1 byte EEPROM each. If the alarm search command is not applied to the DS18B20, these registers may be used as general purpose user memory. The scratchpad also contains a configuration byte to set the desired resolution of the temperature to digital conversion. Writing TH, TL, and the configuration byte is done using a memory function command. Read access to these registers is through the scratchpad. All data is read and written least significant bit first.

DS18B20 BLOCK DIAGRAM Figure 1



PARASITE POWER

The block diagram (Figure 1) shows the parasite-powered circuitry. This circuitry "steals" power whenever the DQ or V_{DD} pins are high. DQ will provide sufficient power as long as the specified timing and voltage requirements are met (see the section titled "1-Wire Bus System"). The advantages of parasite power are twofold: 1) by parasiting off this pin, no local power source is needed for remote sensing of temperature, and 2) the ROM may be read in absence of normal power.

In order for the DS18B20 to be able to perform accurate temperature conversions, sufficient power must be provided over the DQ line when a temperature conversion is taking place. Since the operating current of the DS18B20 is up to 1.5 mA, the DQ line will not have sufficient drive due to the 5k pullup resistor. This problem is particularly acute if several DS18B20s are on the same DQ and attempting to convert simultaneously.

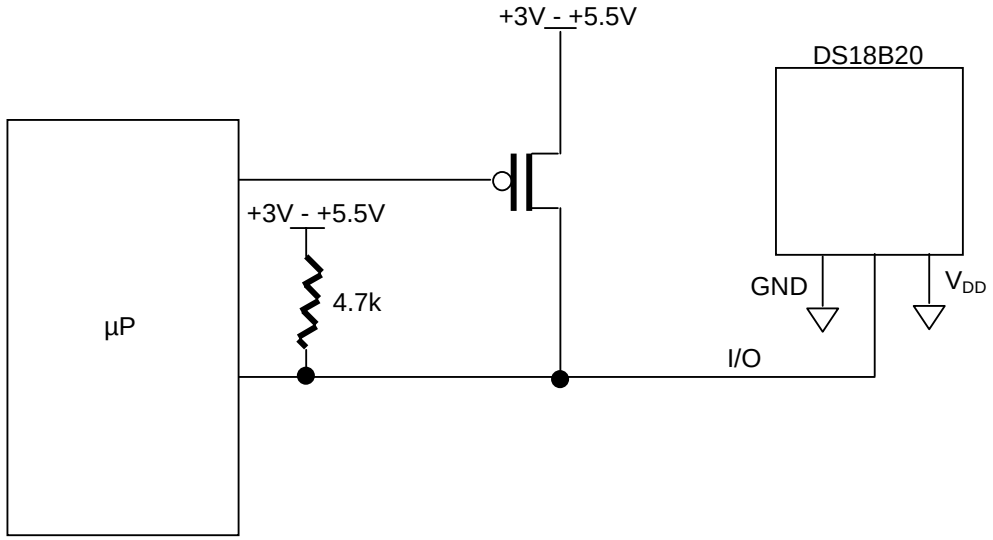
There are two ways to assure that the DS18B20 has sufficient supply current during its active conversion cycle. The first is to provide a strong pullup on the DQ line whenever temperature conversions or copies to the E^2 memory are taking place. This may be accomplished by using a MOSFET to pull the DQ line directly to the power supply as shown in Figure 2. The DQ line must be switched over to the strong pull-up within 10 µs maximum after issuing any protocol that involves copying to the E^2 memory or initiates temperature conversions. When using the parasite power mode, the V_{DD} pin must be tied to ground.

Another method of supplying current to the DS18B20 is through the use of an external power supply tied to the V_{DD} pin, as shown in Figure 3. The advantage to this is that the strong pullup is not required on the DQ line, and the bus master need not be tied up holding that line high during temperature conversions. This allows other data traffic on the 1-Wire bus during the conversion time. In addition, any number of DS18B20s may be placed on the 1-Wire bus, and if they all use external power, they may all simultaneously perform temperature conversions by issuing the Skip ROM command and then issuing the Convert T command. Note that as long as the external power supply is active, the GND pin may not be floating.

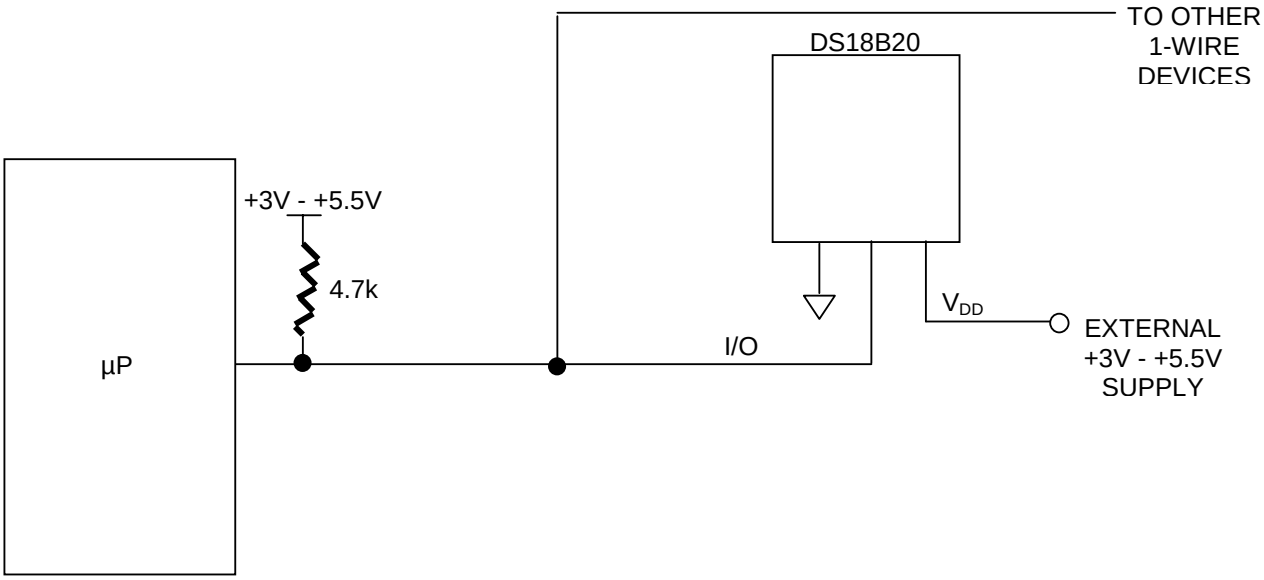
The use of parasite power is not recommended above 100°C, since it may not be able to sustain communications given the higher leakage currents the DS18B20 exhibits at these temperatures. For applications in which such temperatures are likely, it is strongly recommended that V_{DD} be applied to the DS18B20.

For situations where the bus master does not know whether the DS18B20s on the bus are parasite powered or supplied with external V_{DD}, a provision is made in the DS18B20 to signal the power supply scheme used. The bus master can determine if any DS18B20s are on the bus which require the strong pullup by sending a Skip ROM protocol, then issuing the read power supply command. After this command is issued, the master then issues read time slots. The DS18B20 will send back "0" on the 1-Wire bus if it is parasite powered; it will send back a "1" if it is powered from the V_{DD} pin. If the master receives a "0," it knows that it must supply the strong pullup on the DQ line during temperature conversions. See "Memory Command Functions" section for more detail on this command protocol.

STRONG PULLUP FOR SUPPLYING DS18B20 DURING TEMPERATURE CONVERSION Figure 2



USING V_{DD} TO SUPPLY TEMPERATURE CONVERSION CURRENT Figure 3



OPERATION - MEASURING TEMPERATURE

The core functionality of the DS18B20 is its direct-to-digital temperature sensor. The resolution of the DS18B20 is configurable (9, 10, 11, or 12 bits), with 12-bit readings the factory default state. This equates to a temperature resolution of 0.5°C, 0.25°C, 0.125°C, or 0.0625°C. Following the issuance of the Convert T [44h] command, a temperature conversion is performed and the thermal data is stored in the scratchpad memory in a 16-bit, sign-extended two's complement format. The temperature information can be retrieved over the 1-Wire interface by issuing a Read Scratchpad [BEh] command once the conversion has been performed. The data is transferred over the 1-Wire bus, LSB first. The MSB of the temperature register contains the "sign" (S) bit, denoting whether the temperature is positive or negative.

Table 2 describes the exact relationship of output data to measured temperature. The table assumes 12-bit resolution. If the DS18B20 is configured for a lower resolution, insignificant bits will contain zeros. For Fahrenheit usage, a lookup table or conversion routine must be used.

Temperature/Data Relationships Table 2

2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	LSB
MSb			(unit = °C)				LSb	
S	S	S	S	S	2^6	2^5	2^4	MSB

TEMPERATURE	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+125°C	0000 0111 1101 0000	07D0h
+85°C	0000 0101 0101 0000	0550h*
+25.0625°C	0000 0001 1001 0001	0191h
+10.125°C	0000 0000 1010 0010	00A2h
+0.5°C	0000 0000 0000 1000	0008h
0°C	0000 0000 0000 0000	0000h
-0.5°C	1111 1111 1111 1000	FFF8h
-10.125°C	1111 1111 0101 1110	FF5Eh
-25.0625°C	1111 1110 0110 1111	FF6Fh
-55°C	1111 1100 1001 0000	FC90h

*The power on reset register value is +85°C.

OPERATION - ALARM SIGNALING

After the DS18B20 has performed a temperature conversion, the temperature value is compared to the trigger values stored in TH and TL. Since these registers are 8-bit only, bits 9-12 are ignored for comparison. The most significant bit of TH or TL directly corresponds to the sign bit of the 16-bit temperature register. If the result of a temperature measurement is higher than TH or lower than TL, an alarm flag inside the device is set. This flag is updated with every temperature measurement. As long as the alarm flag is set, the DS18B20 will respond to the alarm search command. This allows many DS18B20s to be connected in parallel doing simultaneous temperature measurements. If somewhere the temperature exceeds the limits, the alarming device(s) can be identified and read immediately without having to read non-alarming devices.

64-BIT LASERED ROM

Each DS18B20 contains a unique ROM code that is 64-bits long. The first 8 bits are a 1-Wire family code (DS18B20 code is 28h). The next 48 bits are a unique serial number. The last 8 bits are a CRC of the first 56 bits. (See Figure 4.) The 64-bit ROM and ROM Function Control section allow the DS18B20 to operate as a 1-Wire device and follow the 1-Wire protocol detailed in the section "1-Wire Bus System." The functions required to control sections of the DS18B20 are not accessible until the ROM function protocol has been satisfied. This protocol is described in the ROM function protocol flowchart (Figure 5). The 1-Wire bus master must first provide one of five ROM function commands: 1) Read ROM, 2) Match ROM, 3) Search ROM, 4) Skip ROM, or 5) Alarm Search. After a ROM function sequence has been successfully executed, the functions specific to the DS18B20 are accessible and the bus master may then provide one of the six memory and control function commands.

CRC GENERATION

The DS18B20 has an 8-bit CRC stored in the most significant byte of the 64-bit ROM. The bus master can compute a CRC value from the first 56-bits of the 64-bit ROM and compare it to the value stored within the DS18B20 to determine if the ROM data has been received error-free by the bus master. The equivalent polynomial function of this CRC is:

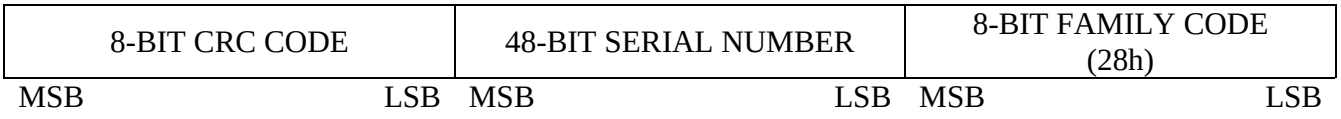
$$CRC = X^8 + X^5 + X^4 + 1$$

The DS18B20 also generates an 8-bit CRC value using the same polynomial function shown above and provides this value to the bus master to validate the transfer of data bytes. In each case where a CRC is used for data transfer validation, the bus master must calculate a CRC value using the polynomial function given above and compare the calculated value to either the 8-bit CRC value stored in the 64-bit ROM portion of the DS18B20 (for ROM reads) or the 8-bit CRC value computed within the DS18B20 (which is read as a ninth byte when the scratchpad is read). The comparison of CRC values and decision to continue with an operation are determined entirely by the bus master. There is no circuitry inside the DS18B20 that prevents a command sequence from proceeding if the CRC stored in or calculated by the DS18B20 does not match the value generated by the bus master.

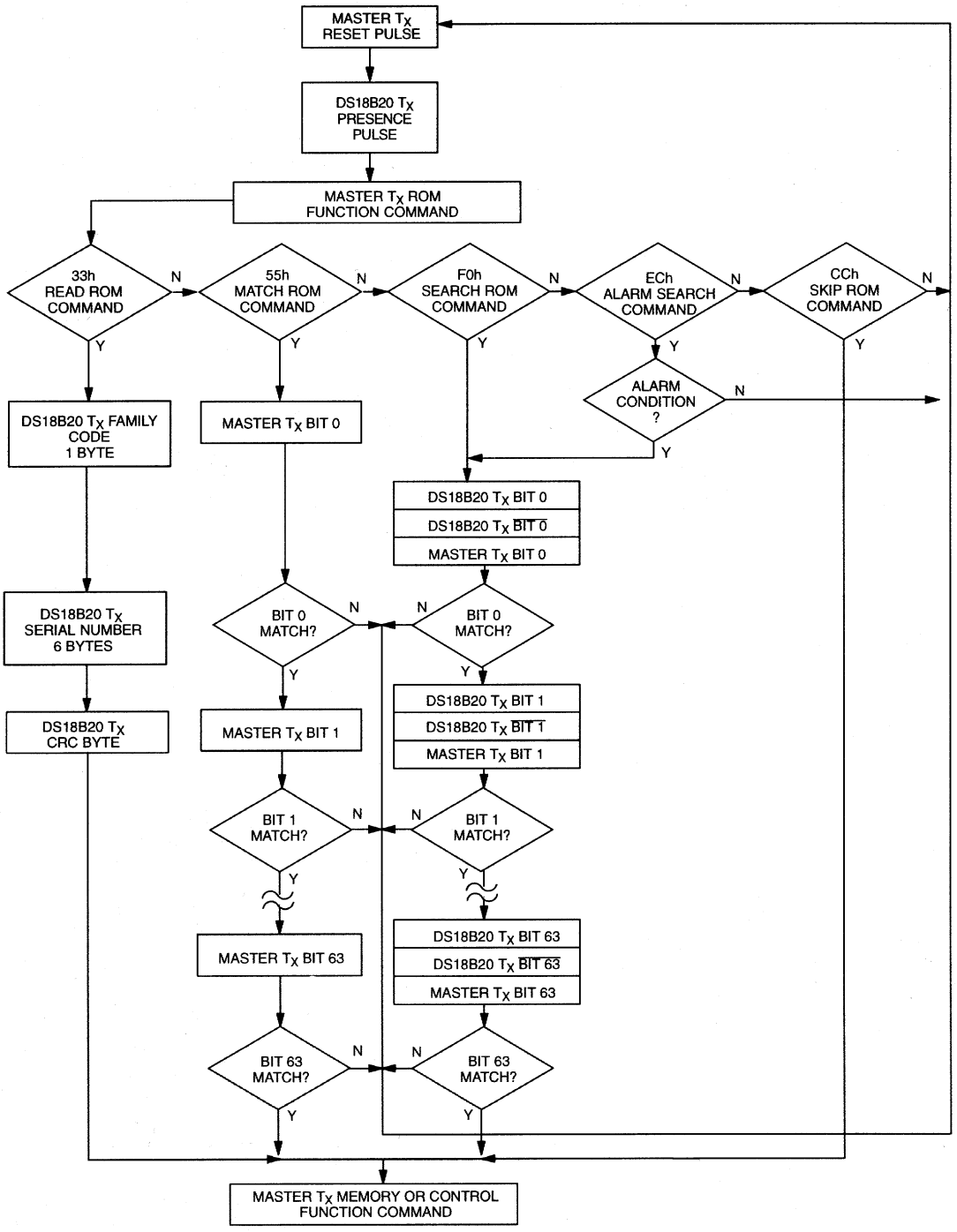
The 1-Wire CRC can be generated using a polynomial generator consisting of a shift register and XOR gates as shown in Figure 6. Additional information about the Dallas 1-Wire Cyclic Redundancy Check is available in Application Note 27 entitled "Understanding and Using Cyclic Redundancy Checks with Dallas Semiconductor Touch Memory Products."

The shift register bits are initialized to 0. Then starting with the least significant bit of the family code, 1 bit at a time is shifted in. After the 8th bit of the family code has been entered, then the serial number is entered. After the 48th bit of the serial number has been entered, the shift register contains the CRC value. Shifting in the 8 bits of CRC should return the shift register to all 0s.

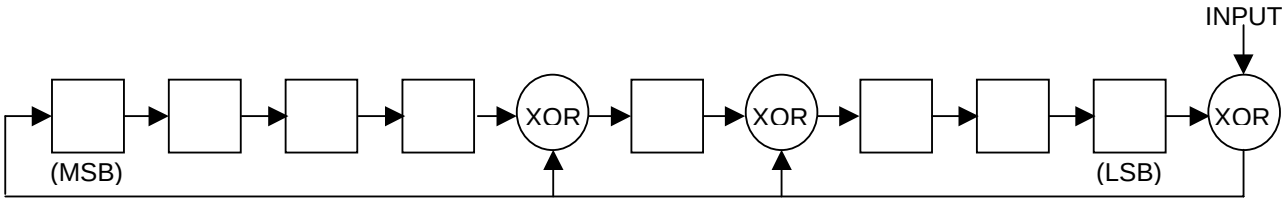
64-BIT LASERED ROM Figure 4



ROM FUNCTIONS FLOW CHART Figure 5



1-WIRE CRC CODE Figure 6



MEMORY

The DS18B20's memory is organized as shown in Figure 8. The memory consists of a scratchpad RAM and a nonvolatile, electrically erasable (E^2) RAM, which stores the high and low temperature triggers TH and TL, and the configuration register. The scratchpad helps insure data integrity when communicating over the 1-Wire bus. Data is first written to the scratchpad using the Write Scratchpad [4Eh] command. It can then be verified by using the Read Scratchpad [BEh] command. After the data has been verified, a Copy Scratchpad [48h] command will transfer the data to the nonvolatile (E^2) RAM. This process insures data integrity when modifying memory. The DS18B20 EEPROM is rated for a minimum of 50,000 writes and 10 years data retention at T = +55°C.

The scratchpad is organized as eight bytes of memory. The first 2 bytes contain the LSB and the MSB of the measured temperature information, respectively. The third and fourth bytes are volatile copies of TH and TL and are refreshed with every power-on reset. The fifth byte is a volatile copy of the configuration register and is refreshed with every power-on reset. The configuration register will be explained in more detail later in this section of the datasheet. The sixth, seventh, and eighth bytes are used for internal computations, and thus will not read out any predictable pattern.

It is imperative that one writes TH, TL, and config in succession; i.e. a write is not valid if one writes only to TH and TL, for example, and then issues a reset. If any of these bytes must be written, all three must be written before a reset is issued.

There is a ninth byte which may be read with a Read Scratchpad [BEh] command. This byte contains a cyclic redundancy check (CRC) byte which is the CRC over all of the eight previous bytes. This CRC is implemented in the fashion described in the section titled "CRC Generation".

Configuration Register

The fifth byte of the scratchpad memory is the configuration register.

It contains information which will be used by the device to determine the resolution of the temperature to digital conversion. The bits are organized as shown in Figure 7.

DS18B20 CONFIGURATION REGISTER Figure 7

0	R1	R0	1	1	1	1	1

MSb LSb

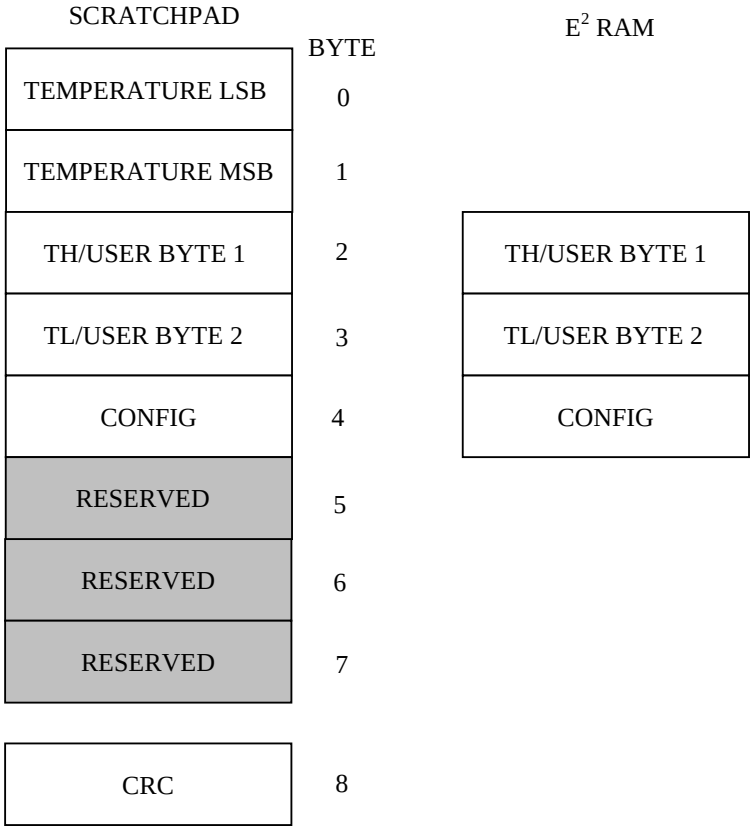
Bits 0-4 are don't cares on a write but will always read out "1".
Bit 7 is a don't care on a write but will always read out "0".

R0, R1: Thermometer resolution bits. Table 3 below defines the resolution of the digital thermometer, based on the settings of these 2 bits. There is a direct tradeoff between resolution and conversion time, as depicted in the AC Electrical Characteristics. The factory default of these EEPROM bits is R0=1 and R1=1 (12-bit conversions).

Thermometer Resolution Configuration Table 3

R1	R0	Thermometer Resolution	Max Conversion Time	
0	0	9 bit	93.75 ms	(t_{conv}/8)
0	1	10 bit	187.5 ms	(t_{conv}/4)
1	0	11 bit	375 ms	(t_{conv}/2)
1	1	12 bit	750 ms	(t_{conv})

DS18B20 MEMORY MAP Figure 8



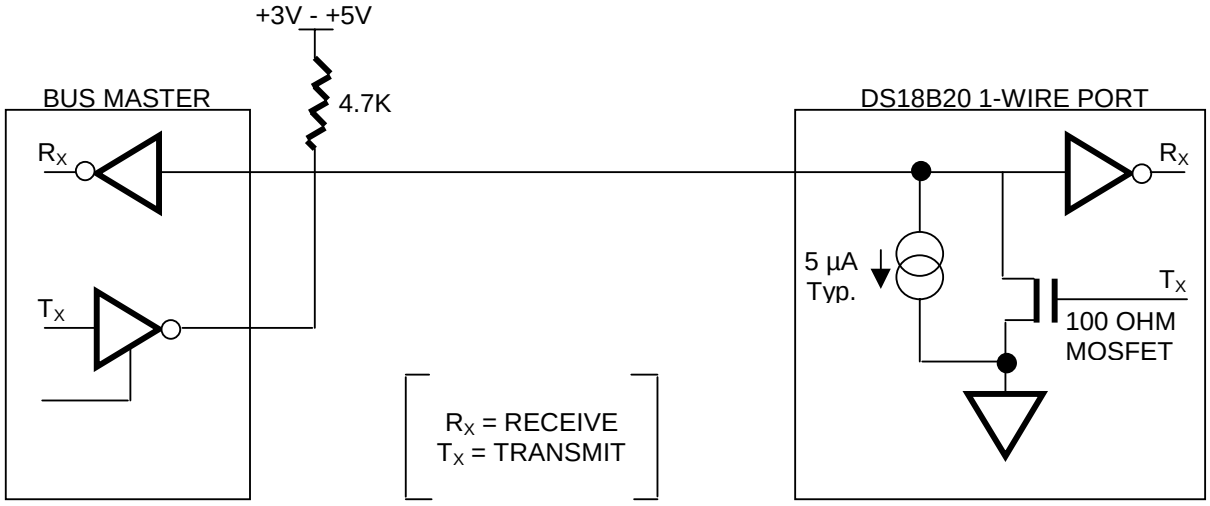
1-WIRE BUS SYSTEM

The 1-Wire bus is a system which has a single bus master and one or more slaves. The DS18B20 behaves as a slave. The discussion of this bus system is broken down into three topics: hardware configuration, transaction sequence, and 1-Wire signaling (signal types and timing).

HARDWARE CONFIGURATION

The 1-Wire bus has only a single line by definition; it is important that each device on the bus be able to drive it at the appropriate time. To facilitate this, each device attached to the 1-Wire bus must have open drain or 3-state outputs. The 1-Wire port of the DS18B20 (DQ pin) is open drain with an internal circuit equivalent to that shown in Figure 9. A multidrop bus consists of a 1-Wire bus with multiple slaves attached. The 1-Wire bus requires a pullup resistor of approximately 5 kΩ.

HARDWARE CONFIGURATION Figure 9



The idle state for the 1-Wire bus is high. If for any reason a transaction needs to be suspended, the bus MUST be left in the idle state if the transaction is to resume. Infinite recovery time can occur between bits so long as the 1-Wire bus is in the inactive (high) state during the recovery period. If this does not occur and the bus is left low for more than 480 µs, all components on the bus will be reset.

TRANSACTION SEQUENCE

The protocol for accessing the DS18B20 via the 1-Wire port is as follows:

- Initialization
- ROM Function Command
- Memory Function Command
- Transaction/Data

INITIALIZATION

All transactions on the 1-Wire bus begin with an initialization sequence. The initialization sequence consists of a reset pulse transmitted by the bus master followed by presence pulse(s) transmitted by the slave(s).

The presence pulse lets the bus master know that the DS18B20 is on the bus and is ready to operate. For more details, see the "1-Wire Signaling" section.

ROM FUNCTION COMMANDS

Once the bus master has detected a presence, it can issue one of the five ROM function commands. All ROM function commands are 8 bits long. A list of these commands follows (refer to flowchart in Figure 5):

Read ROM [33h]

This command allows the bus master to read the DS18B20's 8-bit family code, unique 48-bit serial number, and 8-bit CRC. This command can only be used if there is a single DS18B20 on the bus. If more than one slave is present on the bus, a data collision will occur when all slaves try to transmit at the same time (open drain will produce a wired AND result).

Match ROM [55h]

The match ROM command, followed by a 64-bit ROM sequence, allows the bus master to address a specific DS18B20 on a multidrop bus. Only the DS18B20 that exactly matches the 64-bit ROM sequence will respond to the following memory function command. All slaves that do not match the 64-bit ROM sequence will wait for a reset pulse. This command can be used with a single or multiple devices on the bus.

Skip ROM [CCh]

This command can save time in a single drop bus system by allowing the bus master to access the memory functions without providing the 64-bit ROM code. If more than one slave is present on the bus and a Read command is issued following the Skip ROM command, data collision will occur on the bus as multiple slaves transmit simultaneously (open drain pulldowns will produce a wired AND result).

Search ROM [F0h]

When a system is initially brought up, the bus master might not know the number of devices on the 1-Wire bus or their 64-bit ROM codes. The search ROM command allows the bus master to use a process of elimination to identify the 64-bit ROM codes of all slave devices on the bus.

Alarm Search [ECh]

The flowchart of this command is identical to the Search ROM command. However, the DS18B20 will respond to this command only if an alarm condition has been encountered at the last temperature measurement. An alarm condition is defined as a temperature higher than TH or lower than TL. The alarm condition remains set as long as the DS18B20 is powered up, or until another temperature measurement reveals a non-alarming value. For alarming, the trigger values stored in EEPROM are taken into account. If an alarm condition exists and the TH or TL settings are changed, another temperature conversion should be done to validate any alarm conditions.

Example of a ROM Search

The ROM search process is the repetition of a simple three-step routine: read a bit, read the complement of the bit, then write the desired value of that bit. The bus master performs this simple, three-step routine on each bit of the ROM. After one complete pass, the bus master knows the contents of the ROM in one device. The remaining number of devices and their ROM codes may be identified by additional passes.

The following example of the ROM search process assumes four different devices are connected to the same 1-Wire bus. The ROM data of the four devices is as shown:

ROM1	00110101...
ROM2	10101010...
ROM3	11110101...
ROM4	00010001...

The search process is as follows:

1. The bus master begins the initialization sequence by issuing a reset pulse. The slave devices respond by issuing simultaneous presence pulses.

2. The bus master will then issue the Search ROM command on the 1-Wire bus.

3. The bus master reads a bit from the 1-Wire bus. Each device will respond by placing the value of the first bit of their respective ROM data onto the 1-Wire bus. ROM1 and ROM4 will place a 0 onto the 1-Wire bus, i.e., pull it low. ROM2 and ROM3 will place a 1 onto the 1-Wire bus by allowing the line to stay high. The result is the logical AND of all devices on the line, therefore the bus master sees a 0. The bus master reads another bit. Since the Search ROM data command is being executed, all of the devices on the 1-Wire bus respond to this second read by placing the complement of the first bit of their respective ROM data onto the 1-Wire bus. ROM1 and ROM4 will place a 1 onto the 1-Wire, allowing the line to stay high. ROM2 and ROM3 will place a 0 onto the 1-Wire, thus it will be pulled low. The bus master again observes a 0 for the complement of the first ROM data bit. The bus master has determined that there are some devices on the 1-Wire bus that have a 0 in the first position and others that have a 1.

   The data obtained from the two reads of the three-step routine have the following interpretations:

00	There are still devices attached which have conflicting bits in this position.
01	All devices still coupled have a 0-bit in this bit position.
10	All devices still coupled have a 1-bit in this bit position.
11	There are no devices attached to the 1-Wire bus.

4. The bus master writes a 0. This deselects ROM2 and ROM3 for the remainder of this search pass, leaving only ROM1 and ROM4 connected to the 1-Wire bus.

5. The bus master performs two more reads and receives a 0-bit followed by a 1-bit. This indicates that all devices still coupled to the bus have 0s as their second ROM data bit.

6. The bus master then writes a 0 to keep both ROM1 and ROM4 coupled.

7. The bus master executes two reads and receives two 0-bits. This indicates that both 1-bits and 0-bits exist as the 3rd bit of the ROM data of the attached devices.

8. The bus master writes a 0-bit. This deselects ROM1, leaving ROM4 as the only device still connected.

9. The bus master reads the remainder of the ROM bits for ROM4 and continues to access the part if desired. This completes the first pass and uniquely identifies one part on the 1-Wire bus.

10. The bus master starts a new ROM search sequence by repeating steps 1 through 7.

11. The bus master writes a 1-bit. This decouples ROM4, leaving only ROM1 still coupled.

12. The bus master reads the remainder of the ROM bits for ROM1 and communicates to the underlying logic if desired. This completes the second ROM search pass, in which another of the ROMs was found.

13. The bus master starts a new ROM search by repeating steps 1 through 3.

14. The bus master writes a 1-bit. This deselects ROM1 and ROM4 for the remainder of this search pass, leaving only ROM2 and ROM3 coupled to the system.

15. The bus master executes two Read time slots and receives two 0s.

16. The bus master writes a 0-bit. This decouples ROM3 leaving only ROM2.

17. The bus master reads the remainder of the ROM bits for ROM2 and communicates to the underlying logic if desired. This completes the third ROM search pass, in which another of the ROMs was found.

18. The bus master starts a new ROM search by repeating steps 13 through 15.

19. The bus master writes a 1-bit. This decouples ROM2, leaving only ROM3.

20. The bus master reads the remainder of the ROM bits for ROM3 and communicates to the underlying logic if desired. This completes the fourth ROM search pass, in which another of the ROMs was found.

NOTE:

The bus master learns the unique ID number (ROM data pattern) of one 1-Wire device on each ROM Search operation. The time required to derive the part's unique ROM code is:

$$960\ \mu s + (8 + 3 \times 64)\ 61\ \mu s = 13.16\ ms$$

The bus master is therefore capable of identifying 75 different 1-Wire devices per second.

I/O SIGNALING

The DS18B20 requires strict protocols to insure data integrity. The protocol consists of several types of signaling on one line: reset pulse, presence pulse, write 0, write 1, read 0, and read 1. All of these signals, with the exception of the presence pulse, are initiated by the bus master.

The initialization sequence required to begin any communication with the DS18B20 is shown in Figure 11. A reset pulse followed by a presence pulse indicates the DS18B20 is ready to send or receive data given the correct ROM command and memory function command.

The bus master transmits (TX) a reset pulse (a low signal for a minimum of 480 µs). The bus master then releases the line and goes into a receive mode (RX). The 1-Wire bus is pulled to a high state via the 5k pullup resistor. After detecting the rising edge on the DQ pin, the DS18B20 waits 15-60 µs and then transmits the presence pulse (a low signal for 60-240 µs).

MEMORY COMMAND FUNCTIONS

The following command protocols are summarized in Table 4, and by the flowchart of Figure 10.

Write Scratchpad [4Eh]

This command writes to the scratchpad of the DS18B20, starting at the TH register. The next 3 bytes written will be saved in scratchpad memory at address locations 2 through 4. All 3 bytes must be written before a reset is issued.

Read Scratchpad [BEh]

This command reads the contents of the scratchpad. Reading will commence at byte 0 and will continue through the scratchpad until the ninth (byte 8, CRC) byte is read. If not all locations are to be read, the master may issue a reset to terminate reading at any time.

Copy Scratchpad [48h]

This command copies the scratchpad into the E^2 memory of the DS18B20, storing the temperature trigger bytes in nonvolatile memory. If the bus master issues read time slots following this command, the DS18B20 will output 0 on the bus as long as it is busy copying the scratchpad to E^2; it will return a 1 when the copy process is complete. If parasite-powered, the bus master has to enable a strong pullup for at least 10 ms immediately after issuing this command. The DS18B20 EEPROM is rated for a minimum of 50,000 writes and 10 years data retention at T=+55°C.

Convert T [44h]

This command begins a temperature conversion. No further data is required. The temperature conversion will be performed and then the DS18B20 will remain idle. If the bus master issues read time slots following this command, the DS18B20 will output 0 on the bus as long as it is busy making a temperature conversion; it will return a 1 when the temperature conversion is complete. If parasite-powered, the bus master has to enable a strong pullup for a period greater than t_{conv} immediately after issuing this command.

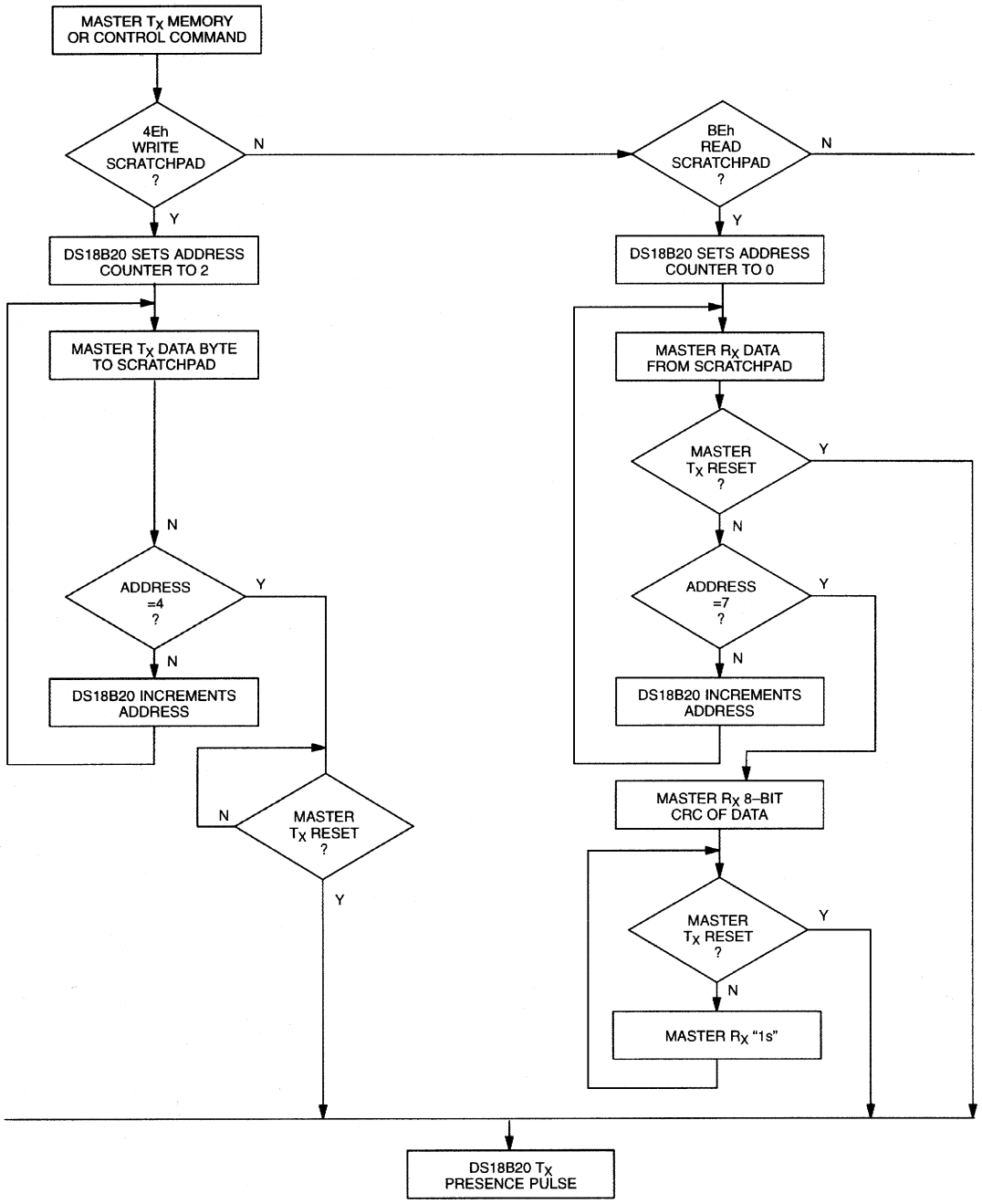
Recall E2 [B8h]

This command recalls the temperature trigger values and configuration register stored in E^2 to the scratchpad. This recall operation happens automatically upon power-up to the DS18B20 as well, so valid data is available in the scratchpad as soon as the device has power applied. With every read data time slot issued after this command has been sent, the device will output its temperature converter busy flag: 0=busy, 1=ready.

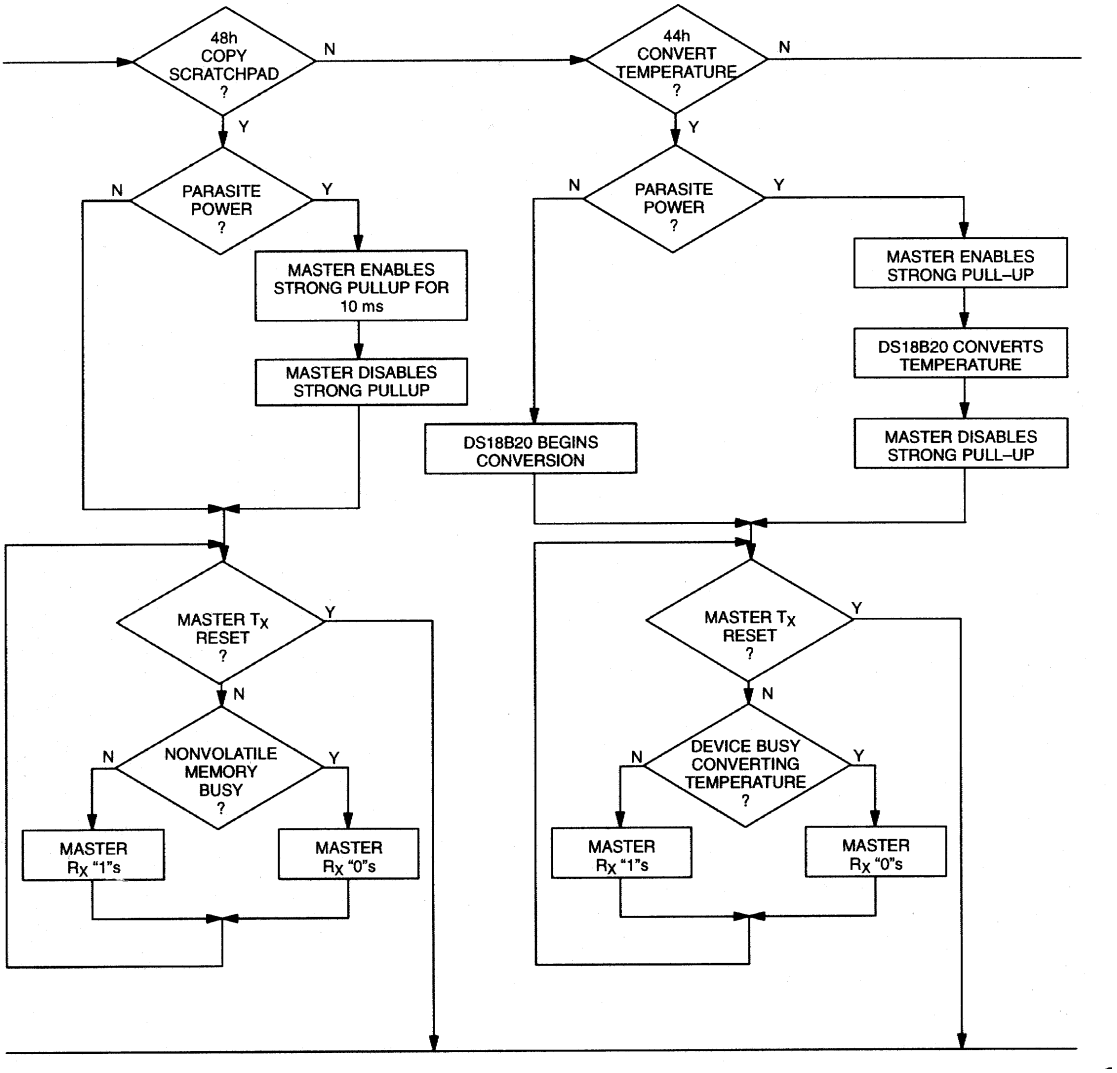
Read Power Supply [B4h]

With every read data time slot issued after this command has been sent to the DS18B20, the device will signal its power mode: 0=parasite power, 1=external power supply provided.

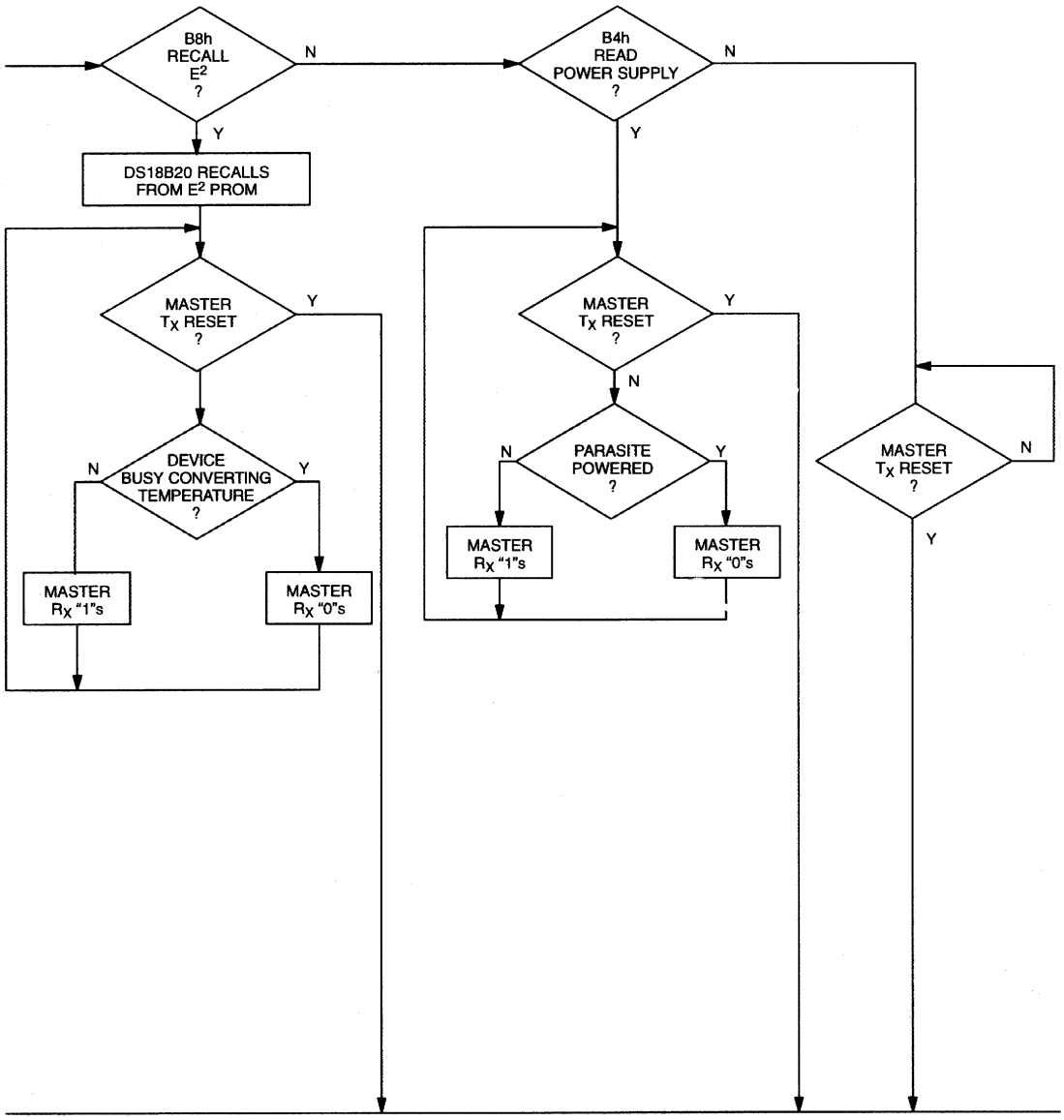
MEMORY FUNCTIONS FLOW CHART Figure 10



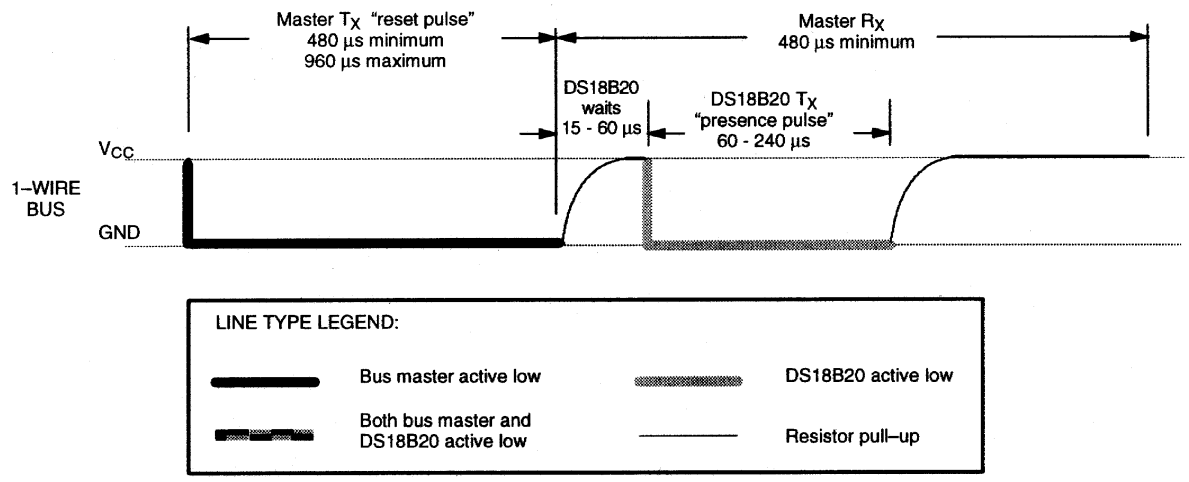
MEMORY FUNCTIONS FLOW CHART Figure 10 (cont'd)



MEMORY FUNCTIONS FLOW CHART Figure 10 (cont'd)



INITIALIZATION PROCEDURE "RESET AND PRESENCE PULSES" Figure 11



DS18B20 COMMAND SET Table 4

INSTRUCTION	DESCRIPTION	PROTOCOL	1-WIRE BUS AFTER ISSUING PROTOCOL	NOTES
TEMPERATURE CONVERSION COMMANDS				
Convert T	Initiates temperature conversion.	44h	<read temperature busy status>	1
MEMORY COMMANDS				
Read Scratchpad	Reads bytes from scratchpad and reads CRC byte.	BEh	<read data up to 9 bytes>	
Write Scratchpad	Writes bytes into scratchpad at addresses 2 through 4 (TH and TL temperature triggers and config).	4Eh	<write data into 3 bytes at addr. 2 through. 4>	3
Copy Scratchpad	Copies scratchpad into nonvolatile memory (addresses 2 through 4 only).	48h	<read copy status>	2
Recall E^2	Recalls values stored in nonvolatile memory into scratchpad (temperature triggers).	B8h	<read temperature busy status>	
Read Power Supply	Signals the mode of DS18B20 power supply to the master.	B4h	<read supply status>	

NOTES:

1. Temperature conversion takes up to 750 ms. After receiving the Convert T protocol, if the part does not receive power from the V_{DD} pin, the DQ line for the DS18B20 must be held high for at least a period greater than t_{conv} to provide power during the conversion process. As such, no other activity may take place on the 1-Wire bus for at least this period after a Convert T command has been issued.

2. After receiving the Copy Scratchpad protocol, if the part does not receive power from the V_{DD} pin, the DQ line for the DS18B20 must be held high for at least 10 ms to provide power during the copy process. As such, no other activity may take place on the 1-Wire bus for at least this period after a Copy Scratchpad command has been issued.

3. All 3 bytes must be written before a reset is issued.

READ/WRITE TIME SLOTS

DS18B20 data is read and written through the use of time slots to manipulate bits and a command word to specify the transaction.

Write Time Slots

A write time slot is initiated when the host pulls the data line from a high logic level to a low logic level. There are two types of write time slots: Write 1 time slots and Write 0 time slots. All write time slots must be a minimum of 60 µs in duration with a minimum of a 1-µs recovery time between individual write cycles.

The DS18B20 samples the DQ line in a window of 15 µs to 60 µs after the DQ line falls. If the line is high, a Write 1 occurs. If the line is low, a Write 0 occurs (see Figure 12).

For the host to generate a Write 1 time slot, the data line must be pulled to a logic low level and then released, allowing the data line to pull up to a high level within 15 µs after the start of the write time slot.

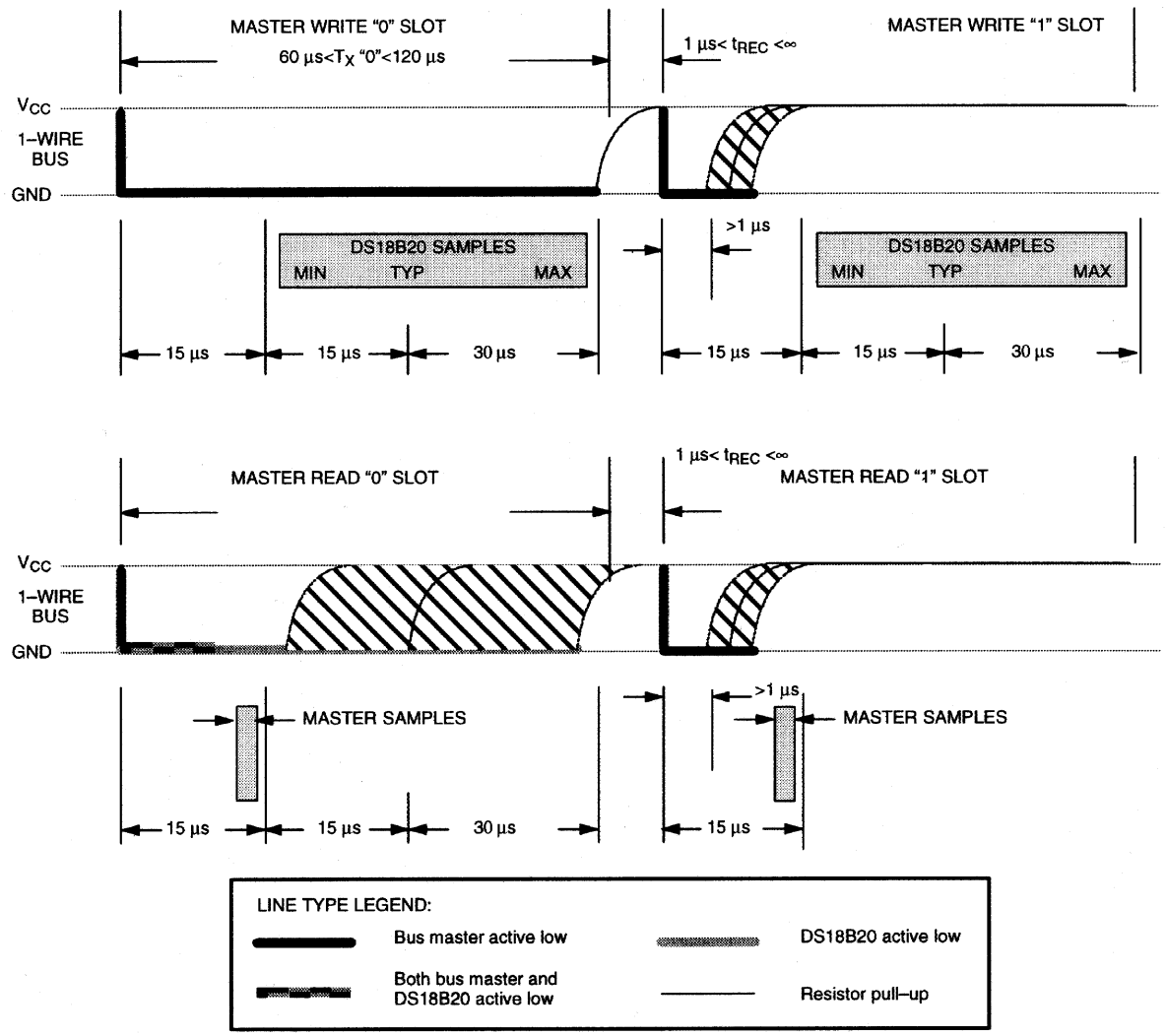
For the host to generate a Write 0 time slot, the data line must be pulled to a logic low level and remain low for 60 µs.

Read Time Slots

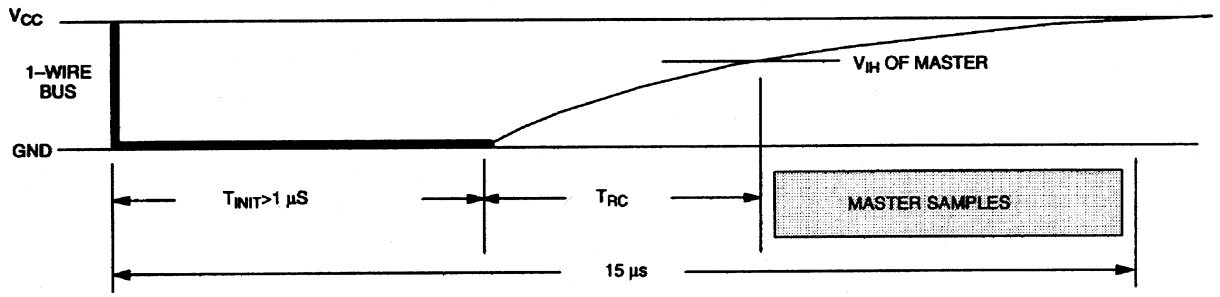
The host generates read time slots when data is to be read from the DS18B20. A read time slot is initiated when the host pulls the data line from a logic high level to logic low level. The data line must remain at a low logic level for a minimum of 1 µs; output data from the DS18B20 is valid for 15 µs after the falling edge of the read time slot. The host therefore must stop driving the DQ pin low in order to read its state 15 µs from the start of the read slot (see Figure 12). By the end of the read time slot, the DQ pin will pull back high via the external pullup resistor. All read time slots must be a minimum of 60 µs in duration with a minimum of a 1-µs recovery time between individual read slots.

Figure 12 shows that the sum of T_{INIT}, T_{RC}, and T_{SAMPLE} must be less than 15 µs. Figure 14 shows that system timing margin is maximized by keeping T_{INIT} and T_{RC} as small as possible and by locating the master sample time towards the end of the 15-µs period.

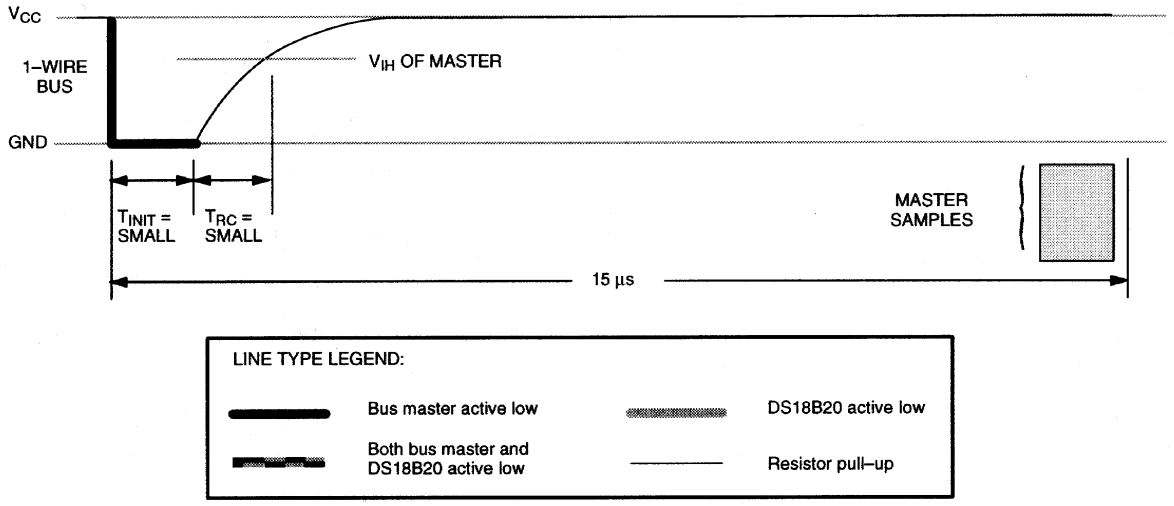
READ/WRITE TIMING DIAGRAM Figure 12



DETAILED MASTER READ 1 TIMING Figure 13



RECOMMENDED MASTER READ 1 TIMING Figure 14



Related Application Notes

The following Application Notes can be applied to the DS18B20. These notes can be obtained from the Dallas Semiconductor "Application Note Book," via our website at http://www.dalsemi.com/.

Application Note 27: "Understanding and Using Cyclic Redundancy Checks with Dallas Semiconductor Touch Memory Product"

Application Note 55: "Extending the Contact Range of Touch Memories"

Application Note 74: "Reading and Writing Touch Memories via Serial Interfaces"

Application Note 104: "Minimalist Temperature Control Demo"

Application Note 106: "Complex MicroLANs"

Application Note 108: "MicroLAN - In the Long Run"

Sample 1-Wire subroutines that can be used in conjunction with AN74 can be downloaded from the website or our Anonymous FTP Site.

MEMORY FUNCTION EXAMPLE Table 5

Example: Bus Master initiates temperature conversion, then reads temperature (parasite power assumed).

MASTER MODE	DATA (LSB FIRST)	COMMENTS
TX	Reset	Reset pulse (480-960 µs).
RX	Presence	Presence pulse.
TX	55h	Issue "Match ROM" command.
TX	<64-bit ROM code>	Issue address for DS18B20.
TX	44h	Issue " Convert T" command.
TX	<I/O LINE HIGH>	I/O line is held high for at least a period of time greater than t_{conv} by bus master to allow conversion to complete.
TX	Reset	Reset pulse.
RX	Presence	Presence pulse.
TX	55h	Issue "Match ROM" command.
TX	<64-bit ROM code>	Issue address for DS18B20.
TX	BEh	Issue "Read Scratchpad" command.
RX	<9 data bytes>	Read entire scratchpad plus CRC; the master now recalculates the CRC of the eight data bytes received from the scratchpad, compares the CRC calculated and the CRC read. If they match, the master continues; if not, this read operation is repeated.
TX	Reset	Reset pulse.
RX	Presence	Presence pulse, done.

MEMORY FUNCTION EXAMPLE Table 6

Example: Bus Master writes memory (parasite power and only one DS18B20 assumed).

MASTER MODE	DATA (LSB FIRST)	COMMENTS
TX	Reset	Reset pulse.
RX	Presence	Presence pulse.
TX	CCh	Skip ROM command.
TX	4Eh	Write Scratchpad command.
TX	<3 data bytes>	Writes three bytes to scratchpad (TH, TL, and config).
TX	Reset	Reset pulse.
RX	Presence	Presence pulse.
TX	CCh	Skip ROM command.
TX	BEh	Read Scratchpad command.
RX	<9 data bytes>	Read entire scratchpad plus CRC. The master now recalculates the CRC of the eight data bytes received from the scratchpad, compares the CRC and the two other bytes read back from the scratchpad. If data match, the master continues; if not, repeat the sequence.
TX	Reset	Reset pulse.
RX	Presence	Presence pulse.
TX	CCh	Skip ROM command.
TX	48h	Copy Scratchpad command; after issuing this command, the master must wait 10 ms for copy operation to complete.
TX	Reset	Reset pulse.
RX	Presence	Presence pulse, done.

ABSOLUTE MAXIMUM RATINGS*

Voltage on Any Pin Relative to Ground	-0.5V to +6.0V
Operating Temperature	-55°C to +125°C
Storage Temperature	-55°C to +125°C
Soldering Temperature	See J-STD-020A specification

* This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operation sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods of time may affect reliability.

RECOMMENDED DC OPERATING CONDITIONS

PARAMETER	SYMBOL	CONDITION	MIN	TYP	MAX	UNITS	NOTES
Supply Voltage	V_{DD}	Local Power	3.0		5.5	V	1
Data Pin	DQ		-0.3		+5.5	V	1
Logic 1	V_{IH}		2.2		V_{CC}+ 0.3	V	1,2
Logic 0	V_{IL}		-0.3		+0.8	V	1,3,7

DC ELECTRICAL CHARACTERISTICS (-55°C to +125°C; V_{DD}=3.0V to 5.5V)

PARAMETER	SYMBOL	CONDITION	MIN	TYP	MAX	UNITS	NOTES
Thermometer Error	t_{ERR}	-10°C to +85°C			±½	°C	
		-55°C to +125°C			±2		
Input Logic High	V_{IH}	Local Power Parasite Power	2.2 3.0		5.5	V V	1,2 1,2
Input Logic Low	V_{IL}		-0.3		+0.8	V	1,3,7
Sink Current	I_L	$V_{I/O}$=0.4V	-4.0			mA	1
Standby Current	I_{DDS}			750	1000	nA	6,8
Active Current	I_{DD}			1	1.5	mA	4
DQ-Input Load Current	I_{DQ}			5		µA	5

AC ELECTRICAL CHARACTERISTICS: NV MEMORY

(-55°C to +125°C; V_{DD}=3.0V to 5.5V)

PARAMETER	SYMBOL	CONDITION	MIN	TYP	MAX	UNITS	NOTES
NV Write Cycle Time	t_{wr}			2	10	ms	
EEPROM Writes	N_{EEWR}	-55°C to +55°C	50k			writes	
EEPROM Data Retention	t_{EEDR}	-55°C to +55°C	10			years	

AC ELECTRICAL CHARACTERISTICS: (-55°C to +125°C; V_{DD}=3.0V to 5.5V)

PARAMETER	SYMBOL	CONDITION	MIN	TYP	MAX	UNITS	NOTES
Temperature Conversion Time	t_{CONV}	9 bit			93.75	ms	
		10 bit			187.5		
		11 bit			375		
		12 bit			750		
Time Slot	t_{SLOT}		60		120	µs	
Recovery Time	t_{REC}		1			µs	
Write 0 Low Time	r_{LOW0}		60		120	µs	
Write 1 Low Time	t_{LOW1}		1		15	µs	
Read Data Valid	t_{RDV}				15	µs	
Reset Time High	t_{RSTH}		480			µs	
Reset Time Low	t_{RSTL}		480			µs	9
Presence Detect High	t_{PDHIGH}		15		60	µs	
Presence Detect Low	t_{PDLOW}		60		240	µs	
Capacitance	$C_{IN/OUT}$				25	pF	

NOTES:

1. All voltages are referenced to ground.

2. Logic one voltages are specified at a source current of 1 mA.

3. Logic zero voltages are specified at a sink current of 4 mA.

4. Active current refers to either temperature conversion or writing to the E^2 memory. Writing to E^2 memory consumes approximately 200 µA for up to 10 ms.

5. Input load is to ground.

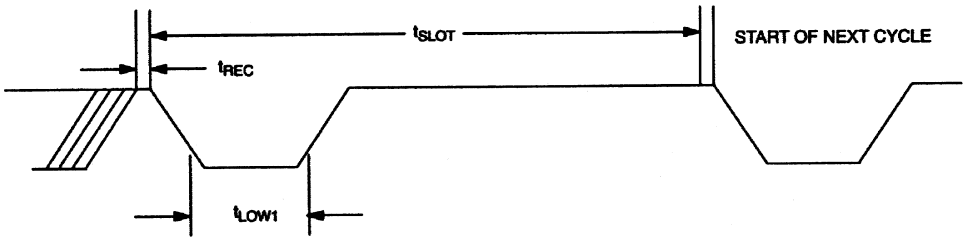
6. Standby current specified up to 70°C. Standby current typically is 3 µA at 125°C.

7. To always guarantee a presence pulse under low voltage parasite power conditions, V_{ILMAX} may have to be reduced to as much as 0.5V.

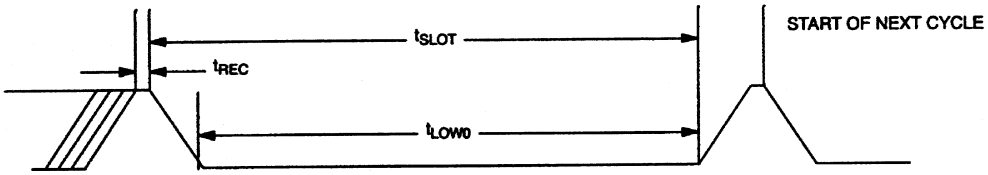
8. To minimize I_{DDS}, DQ should be: GND ≤ DQ ≤ GND +0.3V or V_{DD} – 0.3V ≤ DQ ≤ V_{DD}.

9. Under parasite power, the max t_{RSTL} before a power on reset occurs is 960 µS.

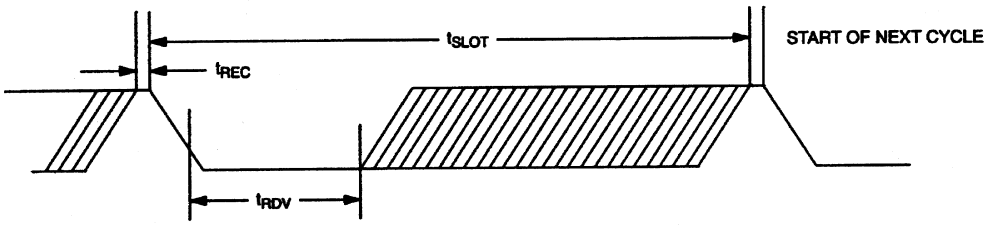
1–WIRE WRITE ONE TIME SLOT



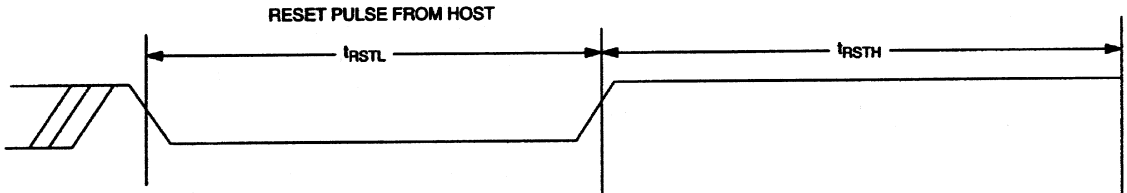
1–WIRE WRITE ZERO TIME SLOT



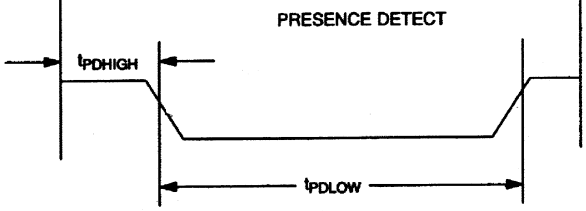
1–WIRE READ ZERO TIME SLOT



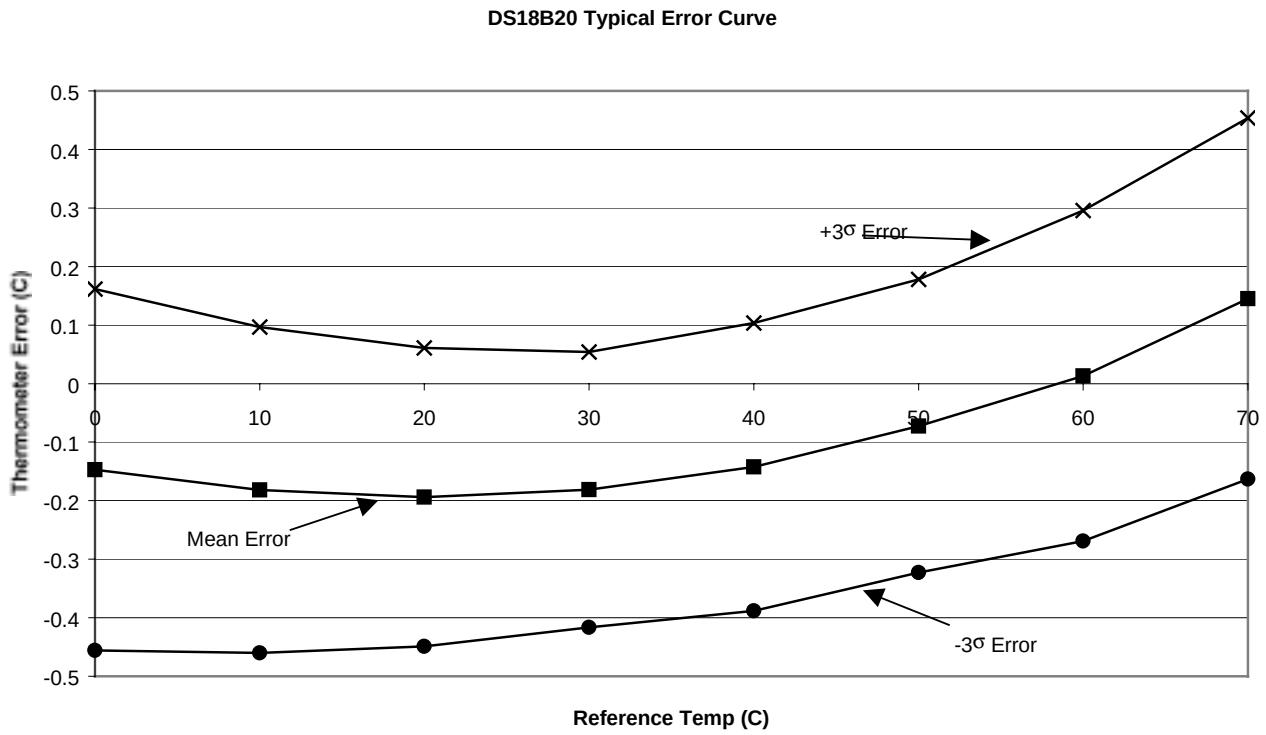
1–WIRE RESET PULSE



1–WIRE PRESENCE DETECT



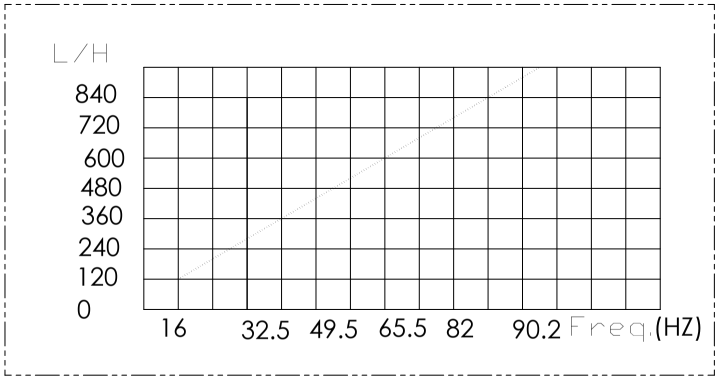
TYPICAL PERFORMANCE CURVE

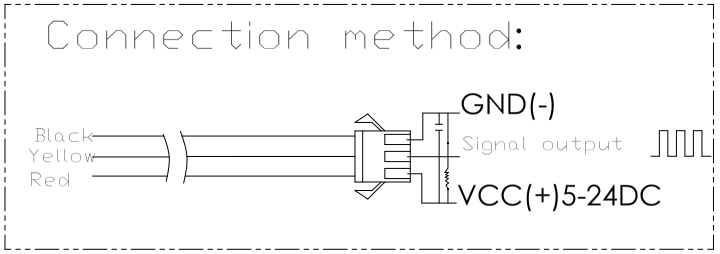


Flow-Pulse	Flow-Curve	Connection method

一.Flow Range:100L/H-/1800H-L/H

Flow (L/H)	Frezq.(HZ)	Erro range
120L/H	16	±10
240L/H	32.5	
360L/H	49.3	
480L/H	65.5	
600L/H	82	
720L/H	90.2	





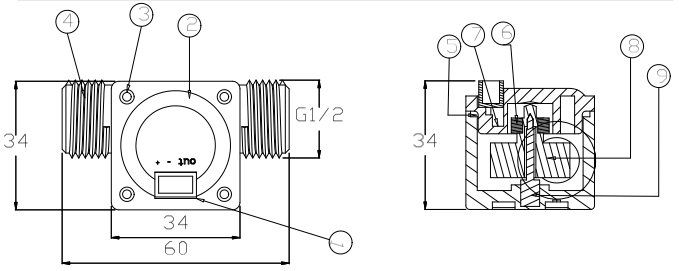
YIFA the plastics Ltd
Prodcut Introduction

1.Modle:YF-21
2.Product Name:Hall sensor
3.Flow Range: 1-30L/MIN
4.(1)Connection Method

(2)Voltage Range 3.5-24VDC, Pluse Characteristic:F=7Q(L/MIN).
(3)Extent of error:±5%.
(4)Flow-Pulse
2L/MIN=16HZ 4L/MIN=32.5HZ 6L/MIN=49.3HZ
8L/MIN=65.5HZ 10L/MIN=82HZ

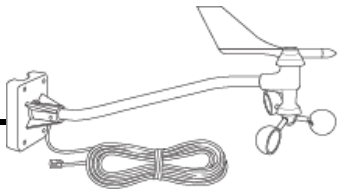
5.Bom

No.	Item	Material	Qty.
1	Connection wire		1
2	Bonnet	PA	1
3	Screw		4
4	Valve body	PA	1
5	Leak press valve		1
6	Magnet		1
7	Hall		1
8	Impeller	POM	1
9	Rustless steel axis	SUS304	1
10			
11			



Anemometer

7911



For Monitor and Wizard Stations

The anemometer includes both wind speed and wind direction sensors. Rugged components stand up to hurricane-force winds, yet are sensitive to a light breeze. Includes sealed stainless-steel bearings for long life. The range and accuracy specifications have been verified in wind-tunnel tests. Digital filtering, with time constant as specified below, is applied to wind direction measurements.

General

Sensor Type
- Wind Speed . . . . . Solid state magnetic sensor
- Wind Direction . . . . . Wind vane and potentiometer

Attached Cable Length . . . . . 40' (12 m)

Note: Cable lengths longer than 140' (42 m) between sensors and console may artificially limit wind speed readings. That is, beyond that length, maximum recordable wind speed decreases as cable length increases. For example, with a length of 140' (42 m), the maximum recordable speed exceeds 175 mph. At 240' (72 m), however, the maximum recordable speed drops to less than 140 mph. Below that upper limit, however, the anemometer's accuracy is not affected.

Cable Type . . . . . 4-conductor, 26 AWG

Connector . . . . . Modular connector (RJ-11)

Recommended Maximum Cable Length . . . . . 140' (42 m) Sensor to Console

Material
- Wind Vane and Control Head . . . . . UV-resistant ABS
- Wind Cups . . . . . Polycarbonate
- Anemometer Arm . . . . . Black-anodized aluminum

Dimensions . . . . . 18.5" long x 7.5" high x 4.75" wide (470 mm x 191 mm x 121 mm)

Weight . . . . . 2 lbs. 15 oz. (1.332 kg)

Console Data

Note: These specifications apply to sensor output as converted by Davis Instruments weather station consoles.

Range
- Wind Speed (large wind cups) (See Note 1) . . . . . 1 to 200 mph, 1 to 173 knots, 0.5 to 89 m/s, 1 to 322 km/h
- Wind Direction . . . . . 0° to 360° or 16 compass points
- Wind Run . . . . . 0 to 1999.9 miles (1999.9 km)

Accuracy
- Wind Speed . . . . . ±2 mph (2 kts, 3 km/h, 1 m/s) or ±5%, whichever is greater
- Wind Direction . . . . . ±7°
- Wind Run . . . . . ±5%

Resolution
- Wind Speed . . . . . 1 mph (1 knot, 0.1 m/s, 1 km/hr)
- Wind Direction . . . . . 1° (0° to 355°), 22.5° between compass points
- Wind Run . . . . . 0.1 m (0.1 km)

Measurement Timing
- Wind Speed Sample Period . . . . . 2.25 seconds
- Wind Speed Sample and Display Interval . . . . . 2.25 seconds (Monitor & Wizard)
- Wind Direction Sample Interval . . . . . 1 second (Monitor & Wizard)
- Wind Direction Filter Time Constant (typical) . . . . . 8 seconds (Monitor & Wizard)
- Wind Direction Display Update Interval . . . . . 2 seconds (Monitor & Wizard)

Davis Instruments 3465 Diablo Ave., Hayward, CA 94545-2778
(510) 732-9229 • FAX (510) 670-0589 • sales@davisnet.com • www.davisnet.com
DS7911 (Rev. G, 2/26/13)

WeatherLink® Data

Note: These specifications apply to sensor output as logged and displayed by the WeatherLink.

Wind Speed	Average during archive interval
High Wind Speed	Maximum during archive interval
Wind Direction	Dominant wind direction during archive interval

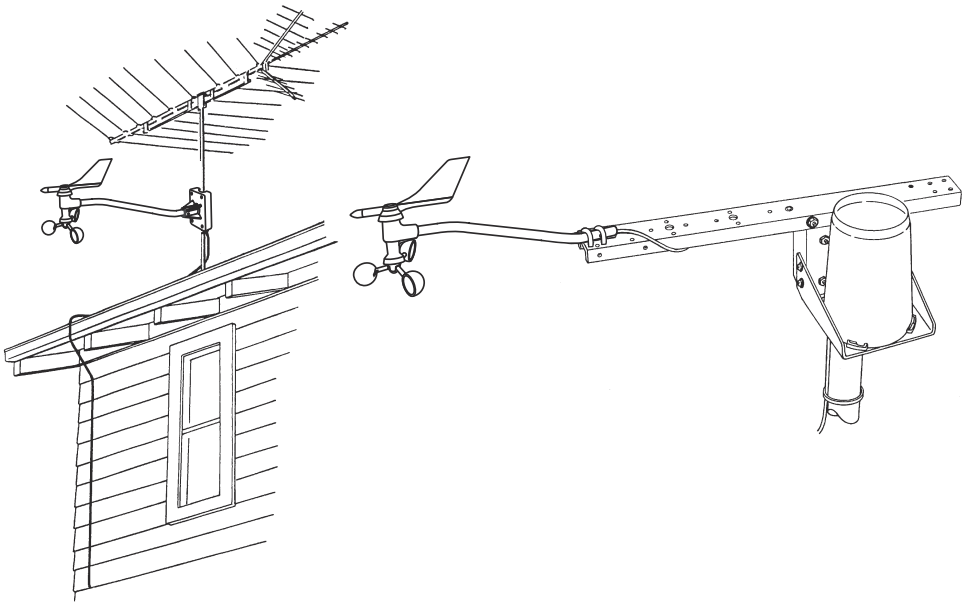
Input/Output Connections

Black	Wind speed contact closure to ground
Red	Ground
Green	Wind direction pot wiper (20KΩ potentiometer)
Yellow	Pot supply voltage

Sensor Output

Wind Speed	1600 rev/hr = 1 mph V = P(2.25/T) *V = speed in mph* *P = no. of pulses per sample period* *T = sample period in seconds*
Wind Direction	Variable resistance 0 - 20KΩ; 10KΩ = south, 180°

Installation Options



Package Dimensions

Product #	Package Dimensions (Length x Width x Height)	Package Weight	UPC Codes
7911	17.50" x 5.75" x 2.50" (445 mm x 146 mm x 64 mm)	1.7 lbs. (.7 kg)	011698 79110 1

ACS712

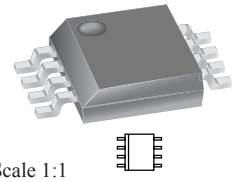
Fully Integrated, Hall Effect-Based Linear Current Sensor IC with 2.1 kVRMS Isolation and a Low-Resistance Current Conductor

Features and Benefits

- Low-noise analog signal path
- Device bandwidth is set via the new FILTER pin
- 5 µs output rise time in response to step input current
- 80 kHz bandwidth
- Total output error 1.5% at T_A = 25°C
- Small footprint, low-profile SOIC8 package
- 1.2 mΩ internal conductor resistance
- 2.1 kVRMS minimum isolation voltage from pins 1-4 to pins 5-8
- 5.0 V, single supply operation
- 66 to 185 mV/A output sensitivity
- Output voltage proportional to AC or DC currents
- Factory-trimmed for accuracy
- Extremely stable output offset voltage
- Nearly zero magnetic hysteresis
- Ratiometric output from supply voltage

TÜV America
Certificate Number:
U8V 06 05 54214 010

Package: 8 Lead SOIC (suffix LC)



Approximate Scale 1:1

Description

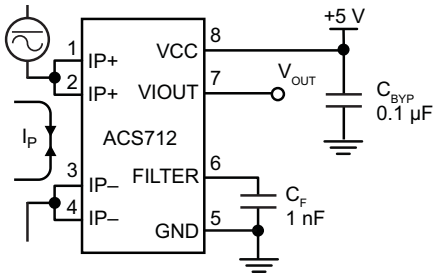
The Allegro™ ACS712 provides economical and precise solutions for AC or DC current sensing in industrial, commercial, and communications systems. The device package allows for easy implementation by the customer. Typical applications include motor control, load detection and management, switch-mode power supplies, and overcurrent fault protection. The device is not intended for automotive applications.

The device consists of a precise, low-offset, linear Hall circuit with a copper conduction path located near the surface of the die. Applied current flowing through this copper conduction path generates a magnetic field which the Hall IC converts into a proportional voltage. Device accuracy is optimized through the close proximity of the magnetic signal to the Hall transducer. A precise, proportional voltage is provided by the low-offset, chopper-stabilized BiCMOS Hall IC, which is programmed for accuracy after packaging.

The output of the device has a positive slope (>$V_{IOUT(Q)}$) when an increasing current flows through the primary copper conduction path (from pins 1 and 2, to pins 3 and 4), which is the path used for current sampling. The internal resistance of this conductive path is 1.2 mΩ typical, providing low power loss. The thickness of the copper conductor allows survival of

Continued on the next page…

Typical Application



Application 1. The ACS712 outputs an analog signal, V_{OUT}. that varies linearly with the uni- or bi-directional AC or DC primary sampled current, I_P, within the range specified. C_F is recommended for noise management, with values that depend on the application.

Description (continued)

the device at up to 5× overcurrent conditions. The terminals of the conductive path are electrically isolated from the signal leads (pins 5 through 8). This allows the ACS712 to be used in applications requiring electrical isolation without the use of opto-isolators or other costly isolation techniques.

The ACS712 is provided in a small, surface mount SOIC8 package. The leadframe is plated with 100% matte tin, which is compatible with standard lead (Pb) free printed circuit board assembly processes. Internally, the device is Pb-free, except for flip-chip high-temperature Pb-based solder balls, currently exempt from RoHS. The device is fully calibrated prior to shipment from the factory.

Selection Guide

Part Number	Packing*	T_A (°C)	Optimized Range, I_P (A)	Sensitivity, Sens (Typ) (mV/A)
ACS712ELCTR-05B-T	Tape and reel, 3000 pieces/reel	–40 to 85	±5	185
ACS712ELCTR-20A-T	Tape and reel, 3000 pieces/reel	–40 to 85	±20	100
ACS712ELCTR-30A-T	Tape and reel, 3000 pieces/reel	–40 to 85	±30	66

*Contact Allegro for additional packing options.

Absolute Maximum Ratings

Characteristic	Symbol	Notes	Rating	Units
Supply Voltage	V_{CC}		8	V
Reverse Supply Voltage	V_{RCC}		–0.1	V
Output Voltage	V_{IOUT}		8	V
Reverse Output Voltage	V_{RIOUT}		–0.1	V
Output Current Source	$I_{IOUT(Source)}$		3	mA
Output Current Sink	$I_{IOUT(Sink)}$		10	mA
Overcurrent Transient Tolerance	I_P	1 pulse, 100 ms	100	A
Nominal Operating Ambient Temperature	T_A	Range E	–40 to 85	ºC
Maximum Junction Temperature	T_J(max)		165	ºC
Storage Temperature	T_{stg}		–65 to 170	ºC

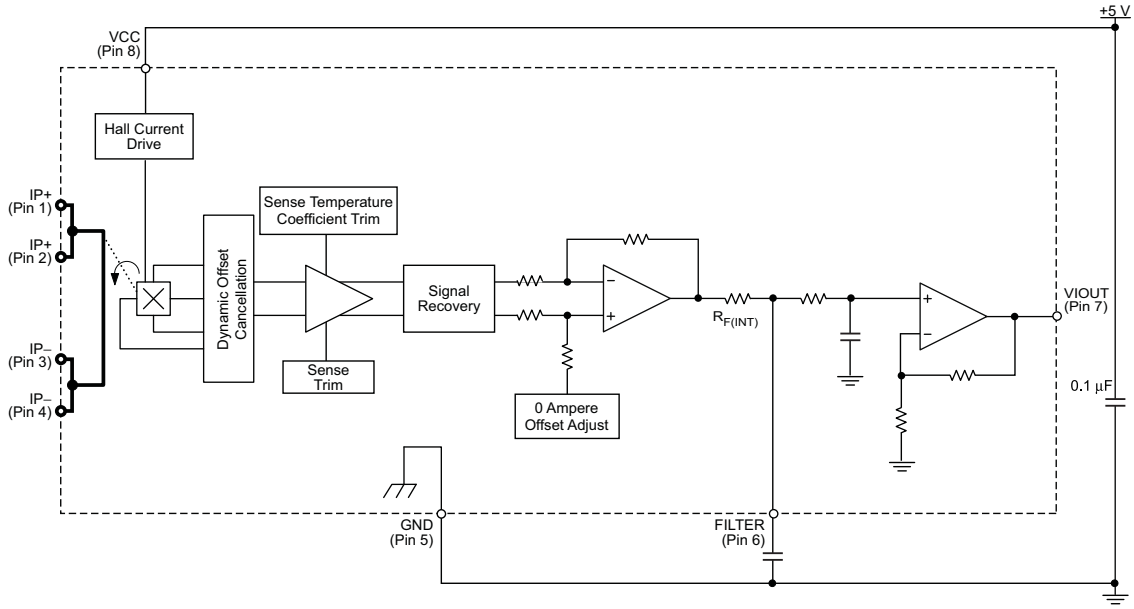
Isolation Characteristics

Characteristic	Symbol	Notes	Rating	Unit
Dielectric Strength Test Voltage*	V_{ISO}	Agency type-tested for 60 seconds per UL standard 60950-1, 1st Edition	2100	VAC
Working Voltage for Basic Isolation	V_{WFSI}	For basic (single) isolation per UL standard 60950-1, 1st Edition	354	VDC or V_{pk}
Working Voltage for Reinforced Isolation	V_{WFRI}	For reinforced (double) isolation per UL standard 60950-1, 1st Edition	184	VDC or V_{pk}

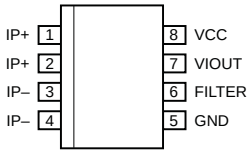
* Allegro does not conduct 60-second testing. It is done only during the UL certification process.

Parameter	Specification
Fire and Electric Shock	CAN/CSA-C22.2 No. 60950-1-03 UL 60950-1:2003 EN 60950-1:2001

Functional Block Diagram



Pin-out Diagram



Terminal List Table

Number	Name	Description
1 and 2	IP+	Terminals for current being sampled; fused internally
3 and 4	IP–	Terminals for current being sampled; fused internally
5	GND	Signal ground terminal
6	FILTER	Terminal for external capacitor that sets bandwidth
7	VIOUT	Analog output signal
8	VCC	Device power supply terminal

COMMON OPERATING CHARACTERISTICS[1] over full range of T_A, C_F = 1 nF, and V_{CC} = 5 V, unless otherwise specified

Characteristic	Symbol	Test Conditions	Min.	Typ.	Max.	Units
ELECTRICAL CHARACTERISTICS						
Supply Voltage	V_{CC}		4.5	5.0	5.5	V
Supply Current	I_{CC}	V_{CC} = 5.0 V, output open	–	10	13	mA
Output Capacitance Load	C_{LOAD}	VIOUT to GND	–	–	10	nF
Output Resistive Load	R_{LOAD}	VIOUT to GND	4.7	–	–	kΩ
Primary Conductor Resistance	$R_{PRIMARY}$	T_A = 25°C	–	1.2	–	mΩ
Rise Time	t_r	I_P = I_P(max), T_A = 25°C, C_{OUT} = open	–	3.5	–	µs
Frequency Bandwidth	f	–3 dB, T_A = 25°C; I_P is 10 A peak-to-peak	–	80	–	kHz
Nonlinearity	E_{LIN}	Over full range of I_P	–	1.5	–	%
Symmetry	E_{SYM}	Over full range of I_P	98	100	102	%
Zero Current Output Voltage	$V_{IOUT(Q)}$	Bidirectional; I_P = 0 A, T_A = 25°C	–	V_{CC} × 0.5	–	V
Power-On Time	t_{PO}	Output reaches 90% of steady-state level, T_J = 25°C, 20 A present on leadframe	–	35	–	µs
Magnetic Coupling[2]			–	12	–	G/A
Internal Filter Resistance[3]	$R_{F(INT)}$			1.7		kΩ

[1]Device may be operated at higher primary current levels, I_P, and ambient, T_A, and internal leadframe temperatures, T_A, provided that the Maximum Junction Temperature, T_J(max), is not exceeded.

[2]1G = 0.1 mT.

[3]$R_{F(INT)}$ forms an RC circuit via the FILTER pin.

COMMON THERMAL CHARACTERISTICS[1]

			Min.	Typ.	Max.	Units
Operating Internal Leadframe Temperature	T_A	E range	–40	–	85	°C
					Value	Units
Junction-to-Lead Thermal Resistance[2]	$R_{\theta JL}$	Mounted on the Allegro ASEK 712 evaluation board			5	°C/W
Junction-to-Ambient Thermal Resistance	$R_{\theta JA}$	Mounted on the Allegro 85-0322 evaluation board, includes the power consumed by the board			23	°C/W

[1]Additional thermal information is available on the Allegro website.

[2]The Allegro evaluation board has 1500 mm^2 of 2 oz. copper on each side, connected to pins 1 and 2, and to pins 3 and 4, with thermal vias connecting the layers. Performance values include the power consumed by the PCB. Further details on the board are available from the Frequently Asked Questions document on our website. Further information about board design and thermal performance also can be found in the Applications Information section of this datasheet.

x05B PERFORMANCE CHARACTERISTICS[1] T_A = –40°C to 85°C, C_F = 1 nF, and V_{CC} = 5 V, unless otherwise specified

Characteristic	Symbol	Test Conditions	Min.	Typ.	Max.	Units
Optimized Accuracy Range	I_P		–5	–	5	A
Sensitivity	Sens	Over full range of I_P, T_A = 25°C	180	185	190	mV/A
Noise	$V_{NOISE(PP)}$	Peak-to-peak, T_A = 25°C, 185 mV/A programmed Sensitivity, C_F = 47 nF, C_{OUT} = open, 2 kHz bandwidth	–	21	–	mV
Zero Current Output Slope	$\Delta V_{OUT(Q)}$	T_A = –40°C to 25°C	–	–0.26	–	mV/°C
		T_A = 25°C to 150°C	–	–0.08	–	mV/°C
Sensitivity Slope	ΔSens	T_A = –40°C to 25°C	–	0.054	–	mV/A/°C
		T_A = 25°C to 150°C	–	–0.008	–	mV/A/°C
Total Output Error[2]	E_{TOT}	I_P =±5 A, T_A = 25°C	–	±1.5	–	%

[1]Device may be operated at higher primary current levels, I_P, and ambient temperatures, T_A, provided that the Maximum Junction Temperature, $T_{J(max)}$, is not exceeded.

[2]Percentage of I_P, with I_P = 5 A. Output filtered.

x20A PERFORMANCE CHARACTERISTICS[1] T_A = –40°C to 85°C, C_F = 1 nF, and V_{CC} = 5 V, unless otherwise specified

Characteristic	Symbol	Test Conditions	Min.	Typ.	Max.	Units
Optimized Accuracy Range	I_P		–20	–	20	A
Sensitivity	Sens	Over full range of I_P, T_A = 25°C	96	100	104	mV/A
Noise	$V_{NOISE(PP)}$	Peak-to-peak, T_A = 25°C, 100 mV/A programmed Sensitivity, C_F = 47 nF, C_{OUT} = open, 2 kHz bandwidth	–	11	–	mV
Zero Current Output Slope	$\Delta V_{OUT(Q)}$	T_A = –40°C to 25°C	–	–0.34	–	mV/°C
		T_A = 25°C to 150°C	–	–0.07	–	mV/°C
Sensitivity Slope	ΔSens	T_A = –40°C to 25°C	–	0.017	–	mV/A/°C
		T_A = 25°C to 150°C	–	–0.004	–	mV/A/°C
Total Output Error[2]	E_{TOT}	I_P =±20 A, T_A = 25°C	–	±1.5	–	%

[1]Device may be operated at higher primary current levels, I_P, and ambient temperatures, T_A, provided that the Maximum Junction Temperature, T_J(max), is not exceeded.

[2]Percentage of I_P, with I_P = 20 A. Output filtered.

x30A PERFORMANCE CHARACTERISTICS[1] T_A = –40°C to 85°C, C_F = 1 nF, and V_{CC} = 5 V, unless otherwise specified

Characteristic	Symbol	Test Conditions	Min.	Typ.	Max.	Units
Optimized Accuracy Range	I_P		–30	–	30	A
Sensitivity	Sens	Over full range of I_P, T_A = 25°C	63	66	69	mV/A
Noise	$V_{NOISE(PP)}$	Peak-to-peak, T_A = 25°C, 66 mV/A programmed Sensitivity, C_F = 47 nF, C_{OUT} = open, 2 kHz bandwidth	–	7	–	mV
Zero Current Output Slope	$\Delta V_{OUT(Q)}$	T_A = –40°C to 25°C	–	–0.35	–	mV/°C
		T_A = 25°C to 150°C	–	–0.08	–	mV/°C
Sensitivity Slope	ΔSens	T_A = –40°C to 25°C	–	0.007	–	mV/A/°C
		T_A = 25°C to 150°C	–	–0.002	–	mV/A/°C
Total Output Error[2]	E_{TOT}	I_P = ±30 A, T_A = 25°C	–	±1.5	–	%

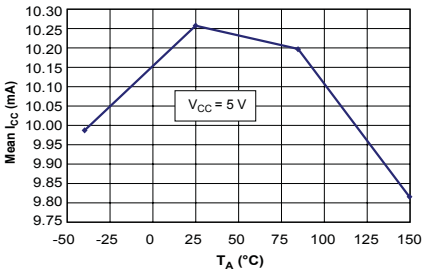
[1]Device may be operated at higher primary current levels, I_P, and ambient temperatures, T_A, provided that the Maximum Junction Temperature, T_J(max), is not exceeded.

[2]Percentage of I_P, with I_P = 30 A. Output filtered.

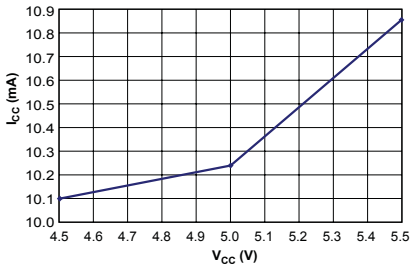
Characteristic Performance

I_P = 5 A, unless otherwise specified

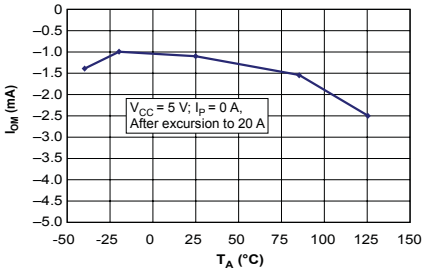
Mean Supply Current versus Ambient Temperature



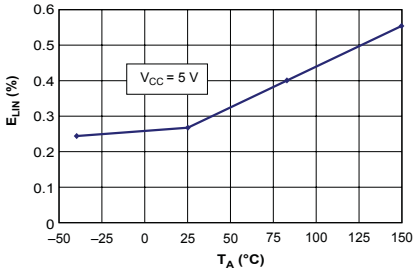
Supply Current versus Supply Voltage



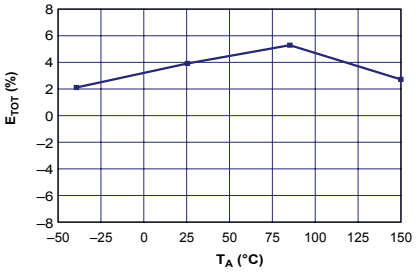
Magnetic Offset versus Ambient Temperature



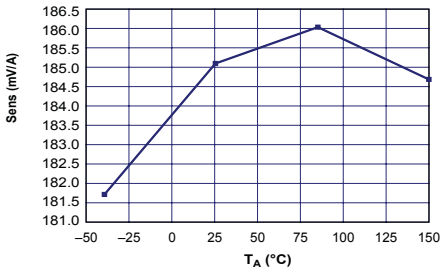
Nonlinearity versus Ambient Temperature



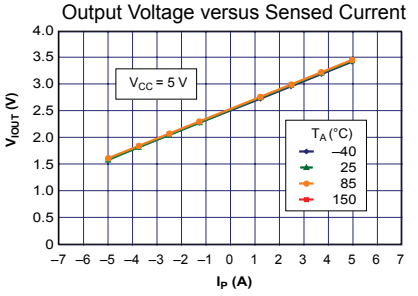
Mean Total Output Error versus Ambient Temperature



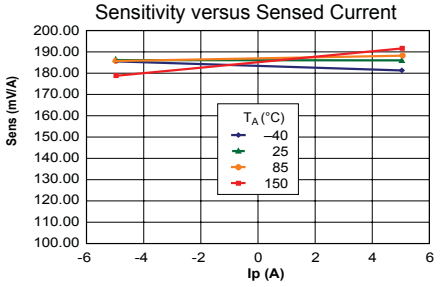
Sensitivity versus Ambient Temperature



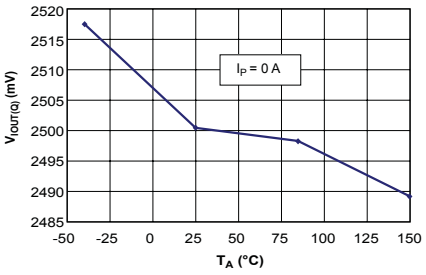
Output Voltage versus Sensed Current



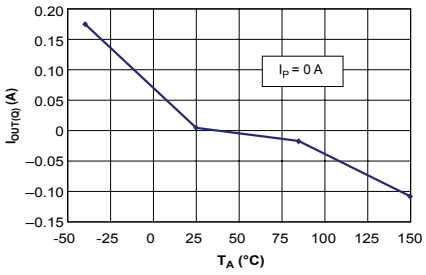
Sensitivity versus Sensed Current



0 A Output Voltage versus Ambient Temperature



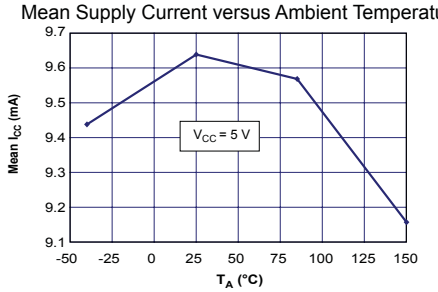
0 A Output Voltage Current versus Ambient Temperature



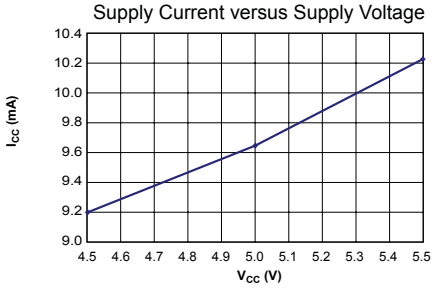
Characteristic Performance

I_P = 20 A, unless otherwise specified

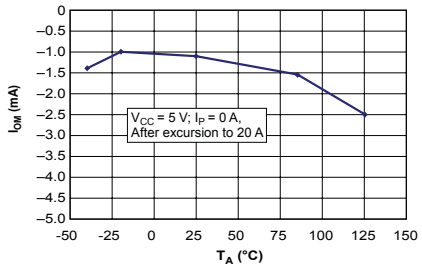
Mean Supply Current versus Ambient Temperature



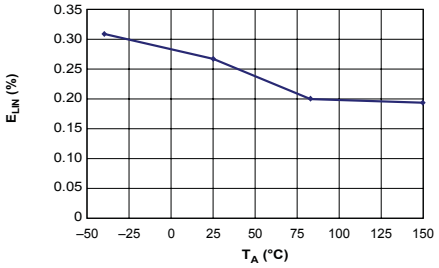
Supply Current versus Supply Voltage



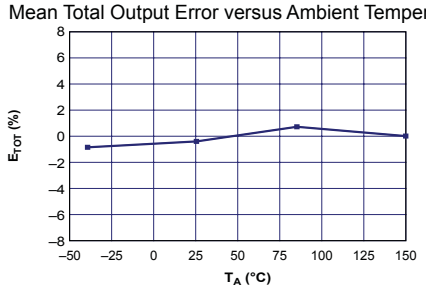
Magnetic Offset versus Ambient Temperature



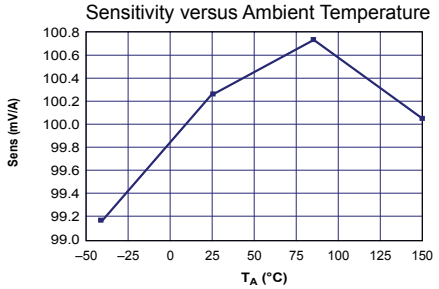
Nonlinearity versus Ambient Temperature



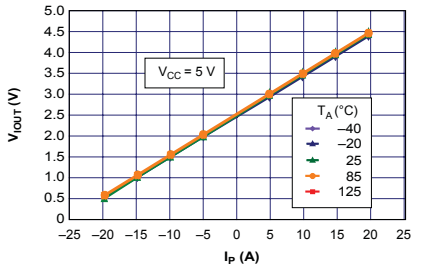
Mean Total Output Error versus Ambient Temperature



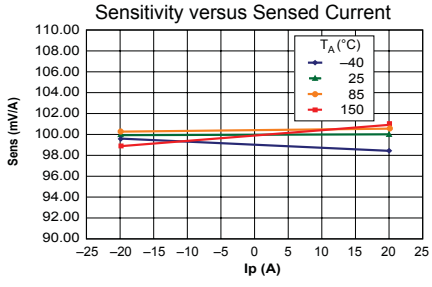
Sensitivity versus Ambient Temperature



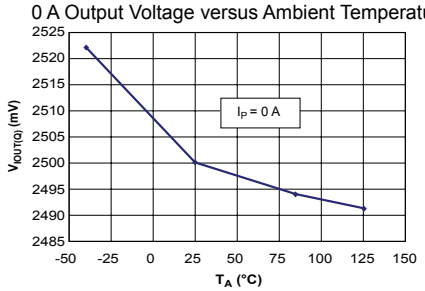
Output Voltage versus Sensed Current



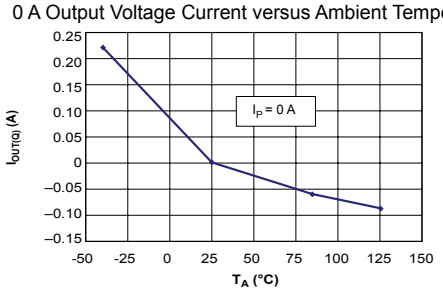
Sensitivity versus Sensed Current



0 A Output Voltage versus Ambient Temperature



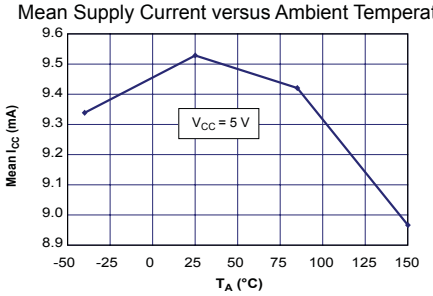
0 A Output Voltage Current versus Ambient Temperature



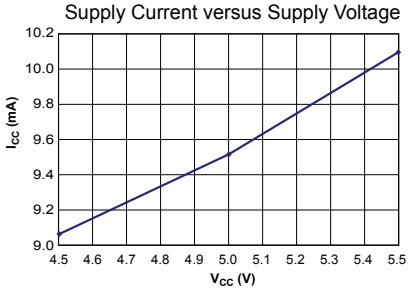
Characteristic Performance

I_P = 30 A, unless otherwise specified

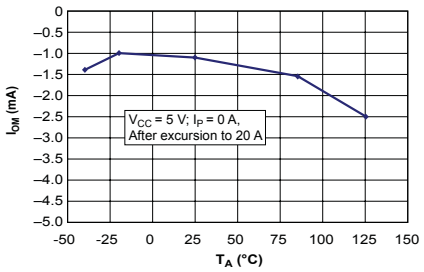
Mean Supply Current versus Ambient Temperature



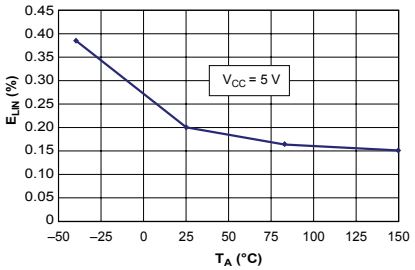
Supply Current versus Supply Voltage



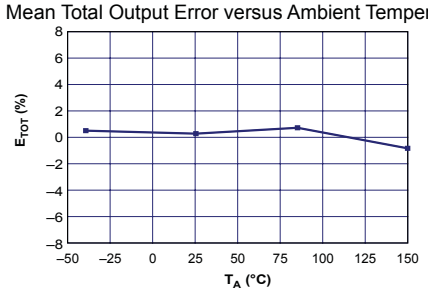
Magnetic Offset versus Ambient Temperature



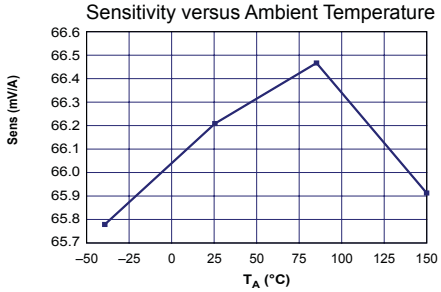
Nonlinearity versus Ambient Temperature



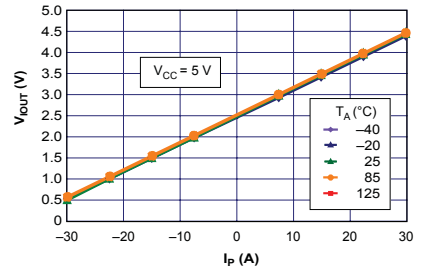
Mean Total Output Error versus Ambient Temperature



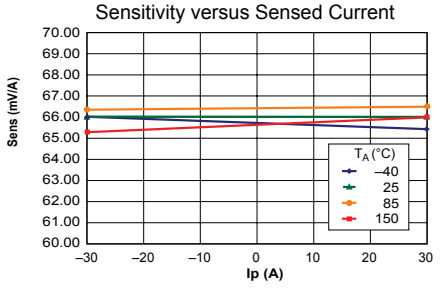
Sensitivity versus Ambient Temperature



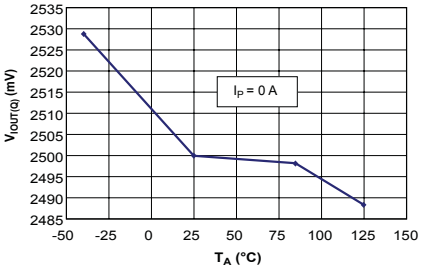
Output Voltage versus Sensed Current



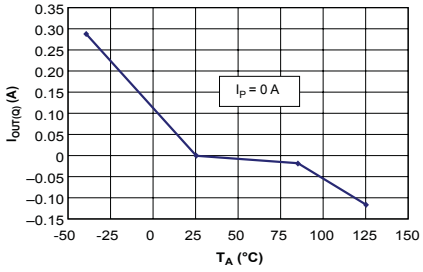
Sensitivity versus Sensed Current



0 A Output Voltage versus Ambient Temperature



0 A Output Voltage Current versus Ambient Temperature



Definitions of Accuracy Characteristics

Sensitivity (Sens). The change in device output in response to a 1 A change through the primary conductor. The sensitivity is the product of the magnetic circuit sensitivity (G/A) and the linear IC amplifier gain (mV/G). The linear IC amplifier gain is programmed at the factory to optimize the sensitivity (mV/A) for the full-scale current of the device.

Noise (V_{NOISE}). The product of the linear IC amplifier gain (mV/G) and the noise floor for the Allegro Hall effect linear IC (≈1 G). The noise floor is derived from the thermal and shot noise observed in Hall elements. Dividing the noise (mV) by the sensitivity (mV/A) provides the smallest current that the device is able to resolve.

Linearity (E_{LIN}). The degree to which the voltage output from the IC varies in direct proportion to the primary current through its full-scale amplitude. Nonlinearity in the output can be attributed to the saturation of the flux concentrator approaching the full-scale current. The following equation is used to derive the linearity:

$$100\left\{1-\left[\frac{\Delta\text{ gain}\times\%\text{ sat}\,(V_{\text{IOUT\_full-scale amperes}}-V_{\text{IOUT(Q)}})}{2\,(V_{\text{IOUT\_half-scale amperes}}-V_{\text{IOUT(Q)}})}\right]\right\}$$

where $V_{\text{IOUT\_full-scale amperes}}$ = the output voltage (V) when the sampled current approximates full-scale $\pm I_P$.

Symmetry (E_{SYM}). The degree to which the absolute voltage output from the IC varies in proportion to either a positive or negative full-scale primary current. The following formula is used to derive symmetry:

$$100\left(\frac{V_{\text{IOUT\_+ full-scale amperes}}-V_{\text{IOUT(Q)}}}{V_{\text{IOUT(Q)}}-V_{\text{IOUT\_–full-scale amperes}}}\right)$$

Quiescent output voltage ($V_{IOUT(Q)}$). The output of the device when the primary current is zero. For a unipolar supply voltage, it nominally remains at $V_{CC}/2$. Thus, V_{CC} = 5 V translates into $V_{IOUT(Q)}$ = 2.5 V. Variation in $V_{IOUT(Q)}$ can be attributed to the resolution of the Allegro linear IC quiescent voltage trim and thermal drift.

Electrical offset voltage (V_{OE}). The deviation of the device output from its ideal quiescent value of $V_{CC}/2$ due to nonmagnetic causes. To convert this voltage to amperes, divide by the device sensitivity, Sens.

Accuracy (E_{TOT}). The accuracy represents the maximum deviation of the actual output from its ideal value. This is also known as the total output error. The accuracy is illustrated graphically in the output voltage versus current chart at right.

Accuracy is divided into four areas:

- **0 A at 25°C.** Accuracy at the zero current flow at 25°C, without the effects of temperature.
- **0 A over Δ temperature.** Accuracy at the zero current flow including temperature effects.
- **Full-scale current at 25°C.** Accuracy at the the full-scale current at 25°C, without the effects of temperature.
- **Full-scale current over Δ temperature.** Accuracy at the full-scale current flow including temperature effects.

Ratiometry. The ratiometric feature means that its 0 A output, $V_{IOUT(Q)}$, (nominally equal to $V_{CC}/2$) and sensitivity, Sens, are proportional to its supply voltage, V_{CC}. The following formula is used to derive the ratiometric change in 0 A output voltage, $\Delta V_{IOUT(Q)RAT}$ (%).

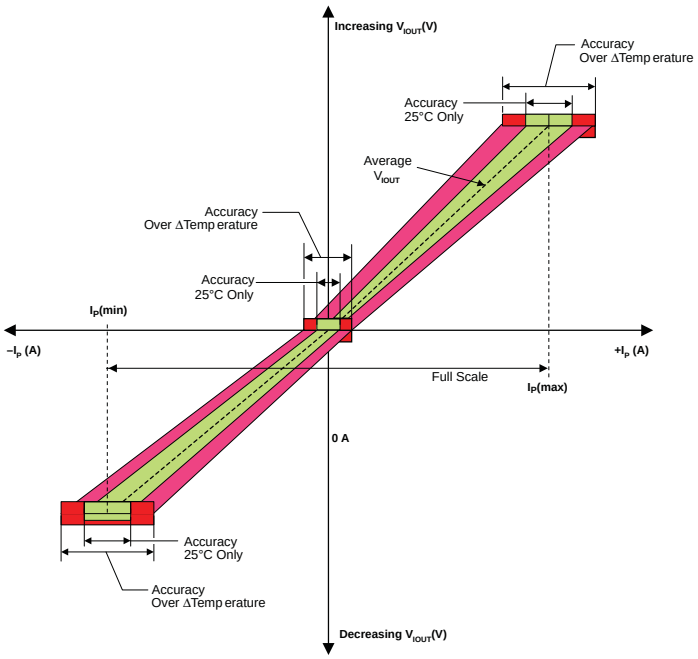
$$100\left(\frac{V_{\text{IOUT(Q)VCC}}/V_{\text{IOUT(Q)5V}}}{V_{CC}/5\text{ V}}\right)$$

The ratiometric change in sensitivity, $\Delta Sens_{RAT}$ (%), is defined as:

$$100\left(\frac{Sens_{VCC}/Sens_{5V}}{V_{CC}/5\text{ V}}\right)$$

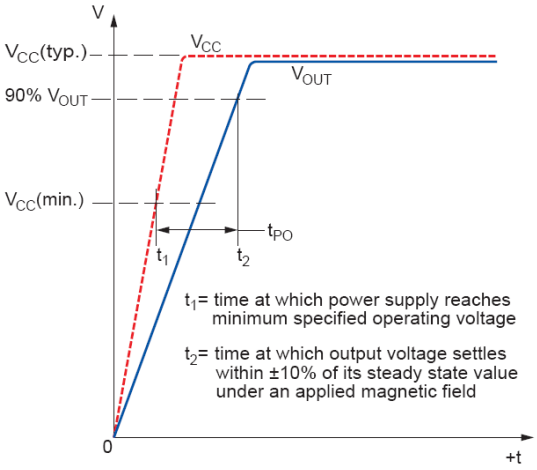
Output Voltage versus Sampled Current

Accuracy at 0 A and at Full-Scale Current

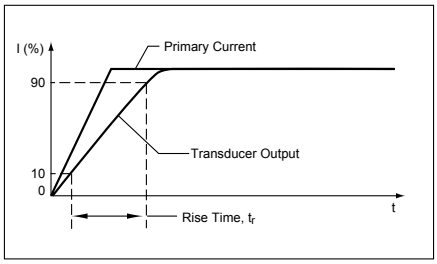


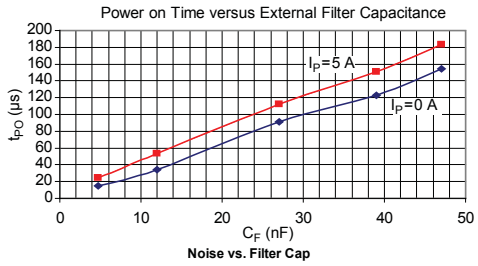
Definitions of Dynamic Response Characteristics

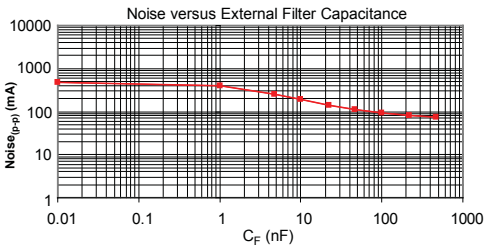
Power-On Time (t_{PO}). When the supply is ramped to its operating voltage, the device requires a finite time to power its internal components before responding to an input magnetic field. Power-On Time, t_{PO} , is defined as the time it takes for the output voltage to settle within ±10% of its steady state value under an applied magnetic field, after the power supply has reached its minimum specified operating voltage, V_{CC}(min), as shown in the chart at right.

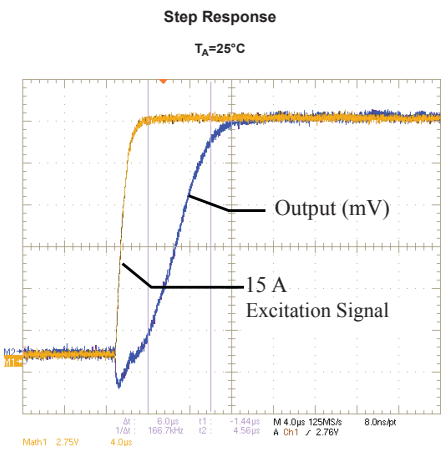


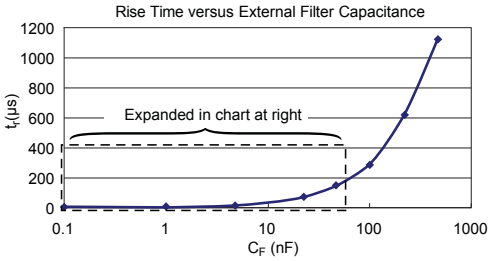
Rise time (t_r). The time interval between a) when the device reaches 10% of its full scale value, and b) when it reaches 90% of its full scale value. The rise time to a step response is used to derive the bandwidth of the device, in which $f(-3\ dB) = 0.35/t_r$. Both t_r and $t_{RESPONSE}$ are detrimentally affected by eddy current losses observed in the conductive IC ground plane.



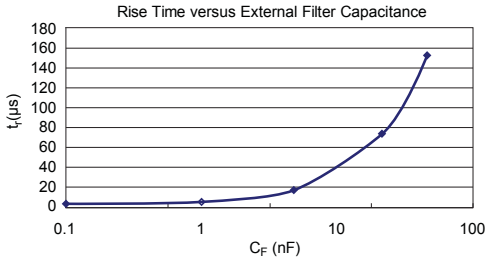








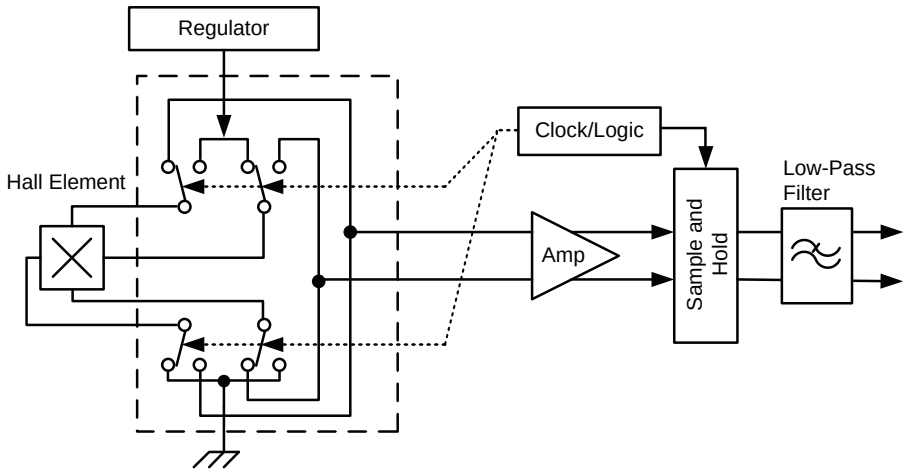
C_F (nF)	t_r (µs)
Open	3.5
1	5.8
4.7	17.5
22	73.5
47	88.2
100	291.3
220	623
470	1120



Chopper Stabilization Technique

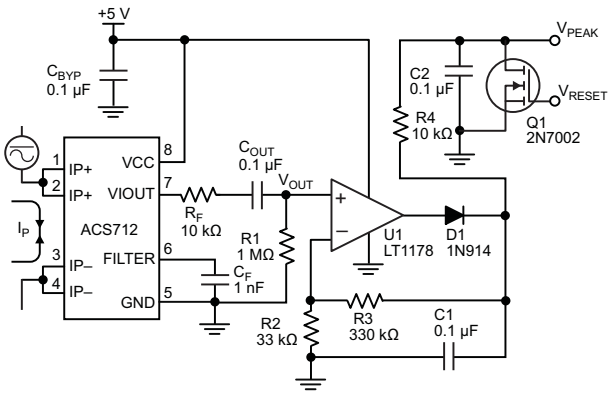
Chopper Stabilization is an innovative circuit technique that is used to minimize the offset voltage of a Hall element and an associated on-chip amplifier. Allegro patented a Chopper Stabilization technique that nearly eliminates Hall IC output drift induced by temperature or package stress effects. This offset reduction technique is based on a signal modulation-demodulation process. Modulation is used to separate the undesired DC offset signal from the magnetically induced signal in the frequency domain. Then, using a low-pass filter, the modulated DC offset is suppressed while the magnetically induced signal passes through the filter. As a result of this chopper stabilization approach, the output voltage from the Hall IC is desensitized to the effects of temperature and mechanical stress. This technique produces devices that have an extremely stable Electrical Offset Voltage, are immune to thermal stress, and have precise recoverability after temperature cycling.

This technique is made possible through the use of a BiCMOS process that allows the use of low-offset and low-noise amplifiers in combination with high-density logic integration and sample and hold circuits.

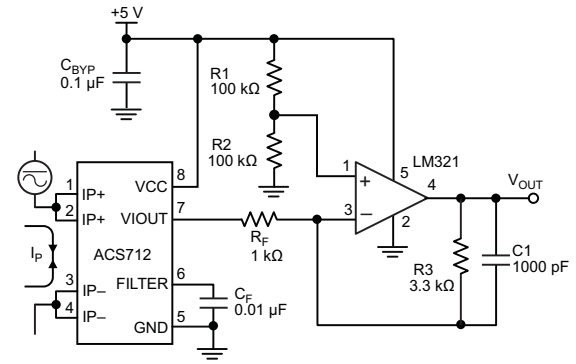


Concept of Chopper Stabilization Technique

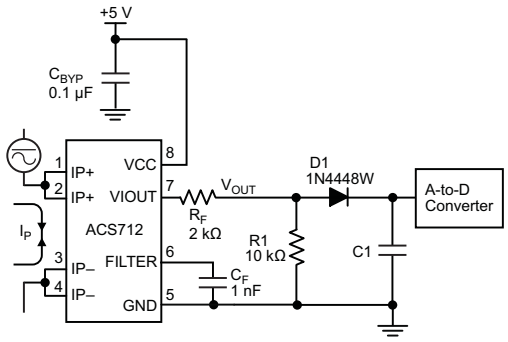
Typical Applications



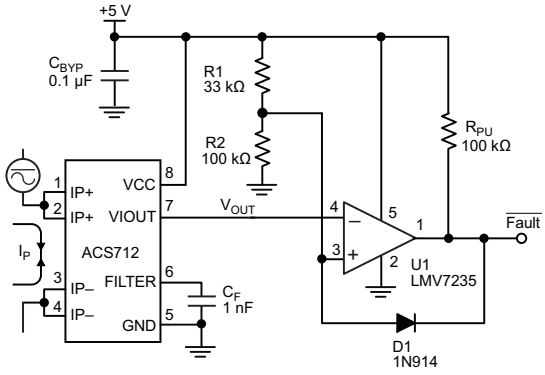
Application 2. Peak Detecting Circuit



Application 3. This configuration increases gain to 610 mV/A (tested using the ACS712ELC-05A).



Application 4. Rectified Output. 3.3 V scaling and rectification application for A-to-D converters. Replaces current transformer solutions with simpler ACS circuit. C1 is a function of the load resistance and filtering desired. R1 can be omitted if the full range is desired.



Application 5. 10 A Overcurrent Fault Latch. Fault threshold set by R1 and R2. This circuit latches an overcurrent fault and holds it until the 5 V rail is powered down.

Improving Sensing System Accuracy Using the FILTER Pin

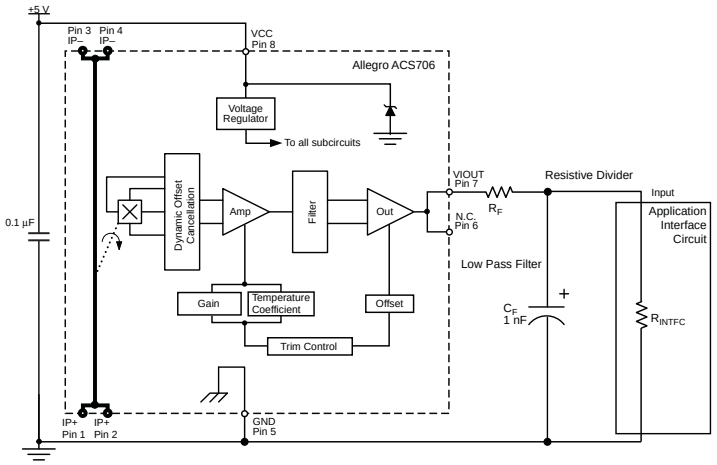
In low-frequency sensing applications, it is often advantageous to add a simple RC filter to the output of the device. Such a low-pass filter improves the signal-to-noise ratio, and therefore the resolution, of the device output signal. However, the addition of an RC filter to the output of a sensor IC can result in undesirable device output attenuation — even for DC signals.

Signal attenuation, ΔV_{ATT}, is a result of the resistive divider effect between the resistance of the external filter, R_F (see Application 6), and the input impedance and resistance of the customer interface circuit, R_{INTFC}. The transfer function of this resistive divider is given by:

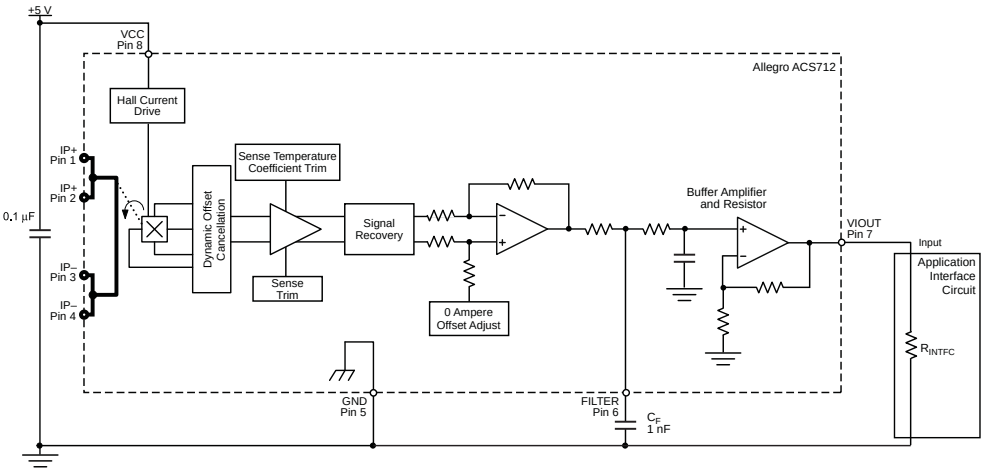
$$\Delta V_{ATT} = V_{IOUT}\left(\frac{R_{INTFC}}{R_F + R_{INTFC}}\right) .$$

Even if R_F and R_{INTFC} are designed to match, the two individual resistance values will most likely drift by different amounts over temperature. Therefore, signal attenuation will vary as a function of temperature. Note that, in many cases, the input impedance, R_{INTFC}, of a typical analog-to-digital converter (ADC) can be as low as 10 kΩ.

The ACS712 contains an internal resistor, a FILTER pin connection to the printed circuit board, and an internal buffer amplifier. With this circuit architecture, users can implement a simple RC filter via the addition of a capacitor, C_F (see Application 7) from the FILTER pin to ground. The buffer amplifier inside of the ACS712 (located after the internal resistor and FILTER pin connection) eliminates the attenuation caused by the resistive divider effect described in the equation for ΔV_{ATT}. Therefore, the ACS712 device is ideal for use in high-accuracy applications that cannot afford the signal attenuation associated with the use of an external RC low-pass filter.

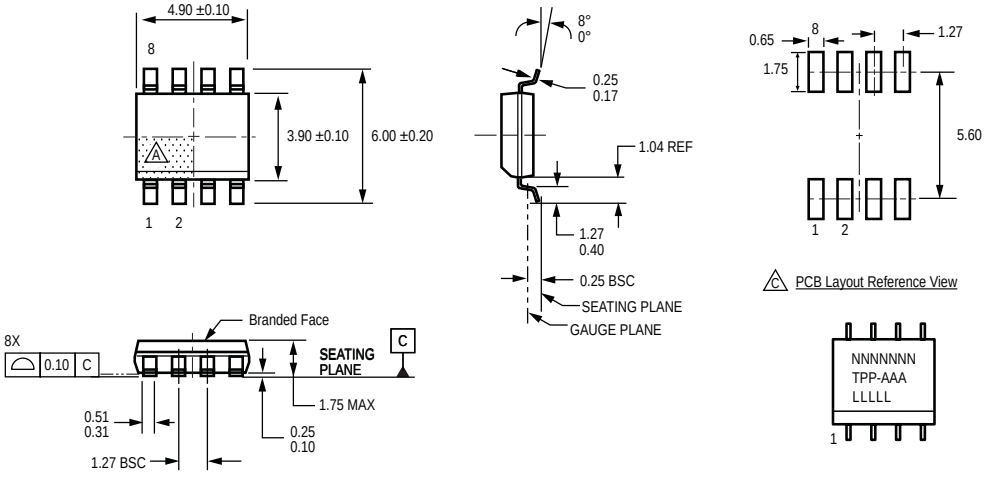


Application 6. When a low pass filter is constructed externally to a standard Hall effect device, a resistive divider may exist between the filter resistor, R_F, and the resistance of the customer interface circuit, R_{INTFC}. This resistive divider will cause excessive attenuation, as given by the transfer function for ΔV_{ATT}.



Application 7. Using the FILTER pin provided on the ACS712 eliminates the attenuation effects of the resistor divider between R_F and R_{INTFC}, shown in Application 6.

Package LC, 8-pin SOIC



For Reference Only; not for tooling use (reference MS-012AA)
Dimensions in millimeters
Dimensions exclusive of mold flash, gate burrs, and dambar protrusions
Exact case and lead configuration at supplier discretion within limits shown

A Terminal #1 mark area

B Branding scale and appearance at supplier discretion

C Reference land pattern layout (reference IPC7351 SOIC127P600X175-8M); all pads a minimum of 0.20 mm from all adjacent pads; adjust as necessary to meet application process requirements and PCB layout tolerances

Revision History

Revision	Revision Date	Description of Revision
Rev. 15	November 16, 2012	Update rise time and isolation, I_{OUT} reference data, patents

For the latest version of this document, visit our website:

www.allegromicro.com

Freescale Semiconductor
Application Note

Document Number: AN4503
Rev. 2, 04/2015

Power Management for Kinetis MCUs

When and how to use Kinetis low-power modes

Contents

1 Introduction

Applications strive for high performance within constrained energy budgets. Increasing system level requirements does not allow for compromises on performance, but pushes for low energy budgets to extend product use times. The Kinetis microcontroller family includes internal power management features that can be used to control the microcontroller's power usage and assist reaching the targets of embedded designs.

This application note discusses how to use the power management systems, provides use case examples, and shows real-time current measurement results for these use cases. A section on how to use the DDR memory controller to maintain the low power nature of your application is in this release.

Also included is a discussion of the differences between power management features on the various microcontrollers, along with drivers demonstrating these low-power states.Tips are given for using each of the power modes.

Power management methods discussed here do not include details on the clock generator module. Refer to the MCU reference manual for a description of the clock modes as well as the explanation of Multipurpose Clock Generator(MCG) acronyms. This application note focuses on the Power Management Controller (PMC), the Low Leakage Wakeup Unit (LLWU), the Reset Control Module (RCM), and the System Mode Controller (SMC).

For a one stop shop for all things low power on Kinetis refer to http://www.freescale.com/lowpower.

1.1 System Mode Controller Revisions - MC and SMC

The mode control is accomplished with a state machine in the MCU called the mode controller or system mode controller (SMC).

The code in this application note is compatible with Kinetis devices with the some differences. The mode controller module has evolved over time adding new low power modes and new run modes keeping mode entry and exit compatible. In this application note MC1 refers to the Mode Controller Module (MC) on Kinetis Rev 1.x 100 MHz devices. The terms MC2 - MC4 refer to the System Mode Controller modules (SMC) on subsequent Kinetis MCUs. With the introduction of the Kinetis L MC3(SMC) was created. Mode Controller version 4 was introduced with a reduced number of low power modes in the Kinetis E. With the introduction of Kinetis devices in 2014, including the K64, K22FN and KV, came MC5(SMC) with a high speed run power mode. See table below for summary.

Table 1. Kinetis Mode Controller Versions

Mode Controller version	Kinetis Series	Description
Mode Conctroller v1	K40,K60- 100 MHz	Kinetis Rev 1.x 100 MHz devices, with VBAT domain and low power OSC
SMC1 Mode Controller v2	Kinetis L 48 MHz	Added VLLS0 power modes No separate VBAT domain
SMC2 Mode controller v3	Kinetis K 72 MHz, K64/K24 120 MHz	Larger Memories M4 cores added LLS2, LLS3
SMC3 Mode Controller v4	Kinetis E	Only has RUN, WAIT, STOP
SMC4 Mode Controller v5	K65, K22FN, KV40 -some up to 180MHz	larger Flash and RAM some Include High Speed Run mode

1.2 Serialization of operation to guarantee correct sequence of operations

Consider the code segment below:

```
SCB_SCR |= SCB_SCR_SLEEPDEEP_MASK;
SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK;
SMC_PMCTRL |=  SMC_PMCTRL_STOPM(0x3);
asm("WFI");
```

The ARM Cortex-M architecture does not delay the execution of the WFI instruction until the previous register write is completed. As a result, the core can assert the SLEEP output to the mode controller before the previous write to the mode controller register is complete. If this happens, the MCU may enter the wrong low-power mode.

The Kinetis MCU takes a minimum of 3 clocks to write to most peripherals. For peripherals like the SMC, that are powered in low power modes, it takes a minimum of 4 clocks for the data to be written to the SMC register. This is assuming no delay through the cross-bar. While the WFI asserts the SLEEP/SLEEPDEEP output in just one clock.

```
volatile unsigned int dummyread;
SCB_SCR |= SCB_SCR_SLEEPDEEP_MASK;
SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK;
SMC_PMCTRL |=  SMC_PMCTRL_STOPM(0x3);
dummyread = SMC_PMCTRL;   /* read-after-write sequence */
```

```
        dummyread++;    /* not required but added this statement to eliminate the compile
warning */
        asm("WFI");
```

The simplest (and guaranteed) way to avoid this problem is to follow the write to the control register, in this case the SMC_PMCTRL register, with a read of the same register. The read-after-write sequence guarantees that the write completes before the read, thus ensuring that the correct low-power mode is latched before the WFI instruction is executed.

This peripheral register read-after-write serialization sequence is applicable across all real-time sequence control. For the sake of consistency for software constructs, the peripheral register read-after-write sequence is a perfectly acceptable memory serialization approach for all control MCUs.

Warning: If the compiler optimization dial is turned in an attempt to improve code density, code sequences like this may get optimized out.

2 Power Management Techniques

2.1 Application Design Introduction

The design team coined the term "chasing nanowatts" to refer to the relentless process of going after every piece of circuitry in the chip, making sure the circuit was at the lowest power possible.

The following details just a few of the areas of focus during the application design process that resulted in the ultra low-power performance of the Kinetis MCUs.

The figure shows the idea of “reducing the area underneath the energy curve”.

In an embedded system where low power is critical it is all about being more energy efficient across the multiple tasks necessary to complete your application. These tasks can typically be summarized in 3 phases:

- Initialization phase
- Control phase which can include data collection, communication and control
- Compute phase

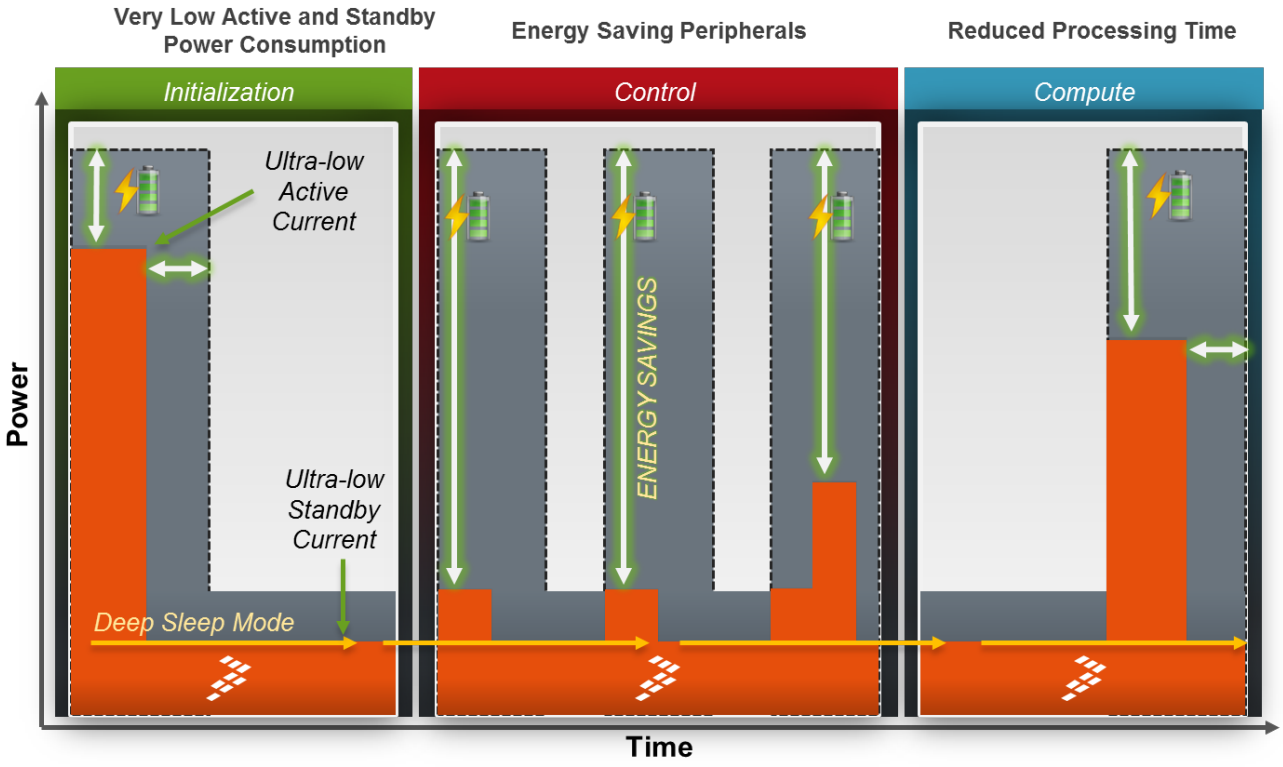


Figure 1. Energy Efficiency: Energy = Power x Time

To reduce energy across these phases the equation is simple; do more with less power and time (Energy = Power x Time). This is achieved by being low power in not just 1 phase, but in all. You must have very low active and standby power consumption.

The Kinetis MCUs can achieve both of these with RUN mode and a speed reduced Very Low Power RUN (VLPR) mode. Looking at the MCU data sheet (DS) “power consumption operating behaviors” section you will see specifications that can represent your use case. The test descriptions given in the DS show you the expected current when you use the controls to reduce the power consumed by your application. Just to name a few of these controls:

- frequency
- clock source
- voltage
- clock gating of peripherals
- execution from flash or RAM
- use of compute mode
- deep sleep modes

Use the DS nmbers as guides to your low power design efforts.

You can follow the graph in Figure 1 by seeing the energy savings comparing the Kinetis L and K2 Series energy curve in orange versus other competitive products energy curve in gray.

Second, you must have energy saving peripherals that are intelligent enough to collect, process, and store data without ever waking up the CPU. Other products would be required to wake-up into full RUN mode to activate a peripheral and complete the data collection phase later to return to a deep sleep mode.

For the Kinetis MCUs, the data collection phase starts in deep sleep mode and shows 3 periodic events triggered by a low power timer. The timer triggers the start of low power ADC conversions where the result is compared to a pre-programmed threshold value using built-in compare feature in ADC.

A feature like this avoids the need to store a result if the value is not within your desired result. Notice the first two events do not trigger the result to be stored. However, the last one does and instead of waking up the CPU to store the data, we have a much smaller energy spike since the Kinetis energy saving peripherals supports an asynchronous DMA wake-up feature that can store the ADC result to SRAM for later processing while the CPU is still sleeping. Once the DMA transfer is completed the MCU automatically returns to the deep sleep mode. Once you have collected sufficient data or transmitted the data using the low power UART you can then wake-up the CPU and begin the compute phase.

This is just one example usage of the available energy saving peripherals.

Last, you must reduce processing time in the compute phase in order to get back into deep sleep mode and start the whole sequence again. This is possible on Kinetis MCUs because of the ultra-efficient Cortex cores combined with an energy saving architecture including a cross bar allowing concurrent accesses between DMA and CPU to slave peripherals and a bit-manipulation engine simplifying bit oriented tasks and a flash memory controller eliminating wait states from flash accesses. The next few paragraphs will break down these concepts in a bit more detail.

2.2 Energy Saving Peripherals

Peripheral	*Low Power Functionality*
DMA	Allows energy-saving peripherals (ex. ADC, UART and Timer/PWM) to trigger asynchronous DMA request in STOP/VLPS modes to perform DMA transfer and return to current power mode with no CPU intervention
UART	Supports asynchronous transmit and receive operations to the bus clock supporting communication down to STOP/VLPS modes. Configurable receiver baud rate oversampling ratio from 4x to 32x allowing higher baud rates with lower clock sources
SPI	Supports slave mode address match wake-up function and first message capture down to STOP/VLPS modes
I2C	Supports multiple address match wake-up function down to STOP/VLPS modes
USB	Supports asynchronous wakeup on resume signaling down to STOP/VLPS
LPTPM (Timer/PWM)	Supports 16-bit timer input capture, output compare and PWM functions down to STOP/VLPS modes
LPTMR (Timer/Pulse Counter)	Supports 16-bit timer and pulse counter functions in all power modes
RTC	Supports 32-bit seconds counter with seconds interrupt and programmable alarm in all power modes with include temperature and voltage compensation

Figure 2. Energy Saving Peripherals 1

UART0 – can receive and transmit in VLPS mode. Can use DMA to receive data and store in a buffer or vice versa with the core shutdown. The address match allows characters to be received and ignored with no action being taken. For example, in say a networked smoke alarm system several nodes could be on the bus and a central system can individually address each detector to check status. Also, the UART supports a single wire system. Also, can receive the first character when in VLPS.

SPI and I2C support address match to wake-up functions. USB supports a wake-up on a resume signal.

LPTPM – supports timer, IC,OC and PWM down to VLPS. Useful for custom serial protocols. PWM output can be modified via DMA in VLPS in applications that provide a control voltage which may be updated occasionally from an ADC reading (light sensor controlling lighting brightness).

LPTMR provides for timer operation down to VLLS1 – wake up events and system “tick” function. Commonly used to trigger events (adc readings etc) or DMA transfers. Can also provide a pulse counting function down to VLLS1 – useful in flow meter applications, counting pulses from sensor, wake up x thousand pulses, otherwise always in VLLS1.

RTC – For devices without a dedicated RTC oscillator, this provides low power, less than 1uA, time keeping capability along with wake up capability down to VLLS1 with 32 KHz very low power crystal oscillator and down to VLLS0 if an external square wave clock is used.

Note that the asynchronous operation of the DMA and certain peripherals down to VLPS is now Kinetis L and new K device specific.

Peripheral	*Low Power Functionality*
ADC	Supports single conversions in multiple result registers down to STOP/VLPS modes with hardware averaging and automatic compare modes
CMP (Analog Comparator)	Supports threshold crossing detection in all power modes along with a triggered compare mode for lower average power compares
DAC	Supports static reference in all power modes
Segment LCD	Supports alternate displays and blink capability in all power modes
TSI (Capacitive Touch Sense Interface)	Supports wake-on capacitive touch on single channel in all power modes
LLWU (Low-Leakage Wake-up Unit)	Supports 8 wake-up pins, RESET and NMI wakeup pins, and some energy-saving peripherals in LLS and VLLSx modes

Figure 3. Energy Saving Peripherals 2

Continuing with the energy-saving peripherals the ADC supports single conversions with hardware averaging and auto compare modes. For example, in a thermostat, the MCU can be in low power mode with the LPTMR triggering an ADC reading regularly, only waking the MCU if the temperature exceeds some threshold.

The analog comparator has threshold crossing detection. It also has a trigger mode that allows the LPTMR to turn the CMP on for long enough to perform a comparison and then shut it down again.

The DAC supports a static reference. This can be updated in VLPS mode via DMA.

The segment LCD has built in blink mode and does not require CPU intervention and alternate blink mode that cuts in half the frequency of CPU intervention. These allow for power reduction. Operational down to VLLS1.

The capacitive touch sense interface supports wake-on touch from a single channel.

2.3 Architectural overview of power modes

The typical power modes in the legacy cores and other embedded systems are Run, Wait, and Stop. The ARM® Cortex™-M4 and M0+ power modes are Run, Sleep, and Deep Sleep. The extended power modes and their relationship to the typical and the ARM Cortex-M4 and M0+ cores are depicted in Figure 4.

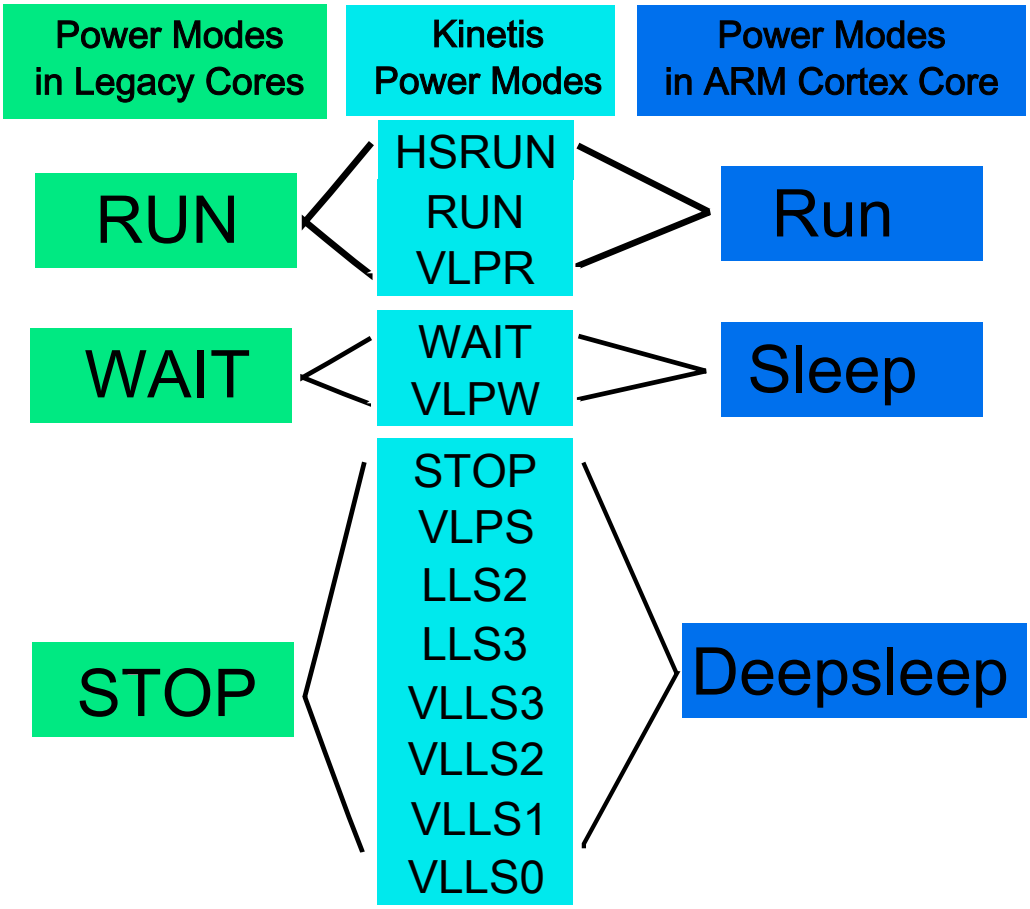


Figure 4. Power modes comparison

2.3.1 ARM Cortex-M4 and M0+ low-power implementation

The ARM Cortex-M4 and M0+ cores have three primary modes of operation: Run, Sleep, and Deep Sleep. On Kinetis, the cores use the wake from interrupt (WFI) instruction to invoke Sleep and Deep Sleep modes.

Figure 5 is a description of the Cortex -M4 and M0 low power implementation. The sleep and deep sleep states are architected in hardware to control the clock to the core and interrupt controller. When entering sleep the NVIC logic remains active and interrupts or a reset wake the core from sleep. When entering deep-sleep an Asynchronous Wakeup Interrupt Controller (AWIC) is used to wake the MCU from a select group of sources. These sources are described in the Low Leakage Wakeup Unit (LLWU) module.

Using the Sleep-On-Exit feature the ARM Cortex cores have one more way to enter low power modes. In the System Control Block of the Cortex processor is a register called the System Control Register (SCR) that contains several control bits related to sleep operation. Setting the control bit SLEEPONEXIT to 1 enables an interrupt driven application to avoid returning to a main application with every wake up event. When SLEEPONEXIT is enabled, the MCU can enter the low power mode as soon as it completes all pending exception handlers (interrupt service routines). It is just like execution a WFI immediately after the exception exit. The sleep-on-exit feature reduces unnecessary stack push and pop operations.

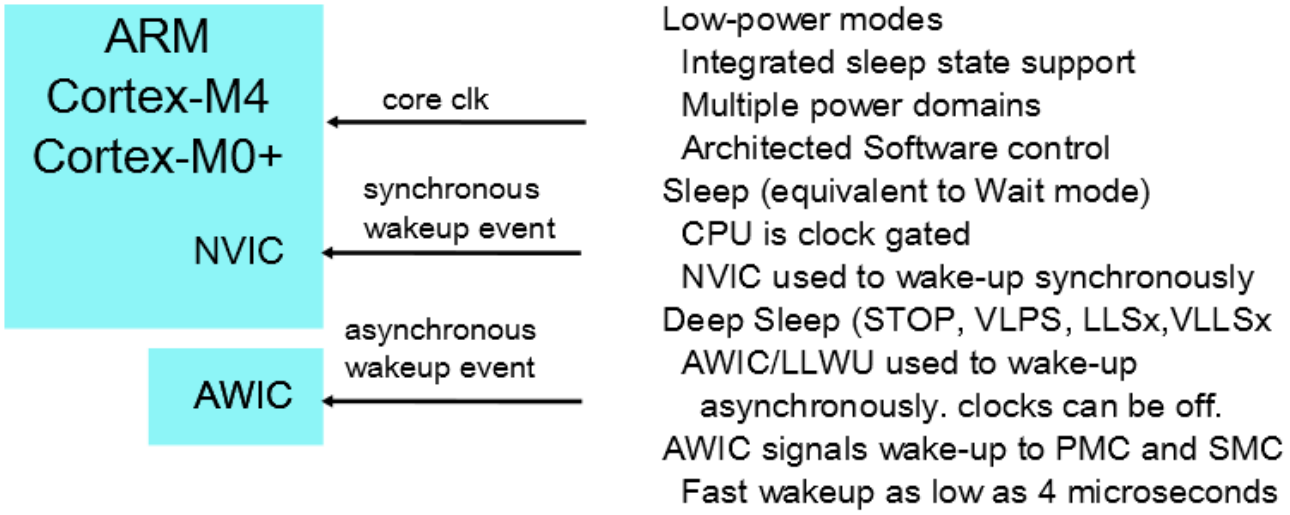


Figure 5. ARM Cortex-M4 and M0+ power modes

2.3.2 Mode description

Each of the power modes is described in Table 2 with details about the mode and the basics of mode entry and exit. The Kinetis MCU may have all of these low power modes or a subset. Some of the products revert to the traditional low power modes of Run, WAIT and STOP. Some do not have a LLWU module. See the MCU reference manual for the list of specific modes supported. For more advanced information, including what would prevent you from entering low-power mode, consult the reference manual power mode transitions section. The measurement data, frequencies, and other limit data given below are guidelines only. Make sure to use the MCU data sheet for the official values.

The following table describes the power modes available for most Kinetis devices. Refer to the individual device reference manual for details on the chip specific mode.

Table 2. Chip power modes

Chip mode	Description
Normal RUN	The MCU can be run at full speed and the internal supply is fully regulated, that is, in run regulation. This mode is also referred to as Normal Run mode.
High speed RUN	Allows maximum performance of the chip. The MCU can be run at a faster frequency compared with RUN mode and the internal supply is fully regulated. See the Power Management chapter for details about the maximum allowable frequencies.
Normal WAIT via WFI	Allows peripherals to function while the core is in sleep mode, reducing power. NVIC remains sensitive to interrupts; peripherals continue to be clocked. Run regulation is maintained.
Normal STOP via WFI	Places chip in static state. Lowest power mode that retains all registers while maintaining LVD protection. NVIC is disabled; AWIC is used to wake up from interrupt; peripheral clocks are stopped. System clocks to other masters and bus clocks are gated off after all stop acknowledge signals from supporting peripherals are valid. The internal voltage regulator is in run regulation.
VLPR (Very Low Power Run)	On-chip voltage regulator is in a low power mode that supplies only enough power to run the chip at a reduced frequency. Reduced frequency Flash access mode (1 MHz); LVD off; internal oscillator provides a low power 4 MHz source core, the bus, and the peripheral clocks.
VLPW (Very Low Power Wait) via WFI	Same as VLPR but with the core in sleep mode to further reduce power; NVIC remains sensitive to interrupts (FCLK = ON). On-chip voltage regulator is in a low power mode that supplies only enough power to run the chip at a reduced frequency.

Table continues on the next page...

Table 2. Chip power modes (continued)

Chip mode	Description
VLPS Very Low Power Stop) via WFI	Places chip in static state with LVD operation off. Lowest power mode with ADC and pin interrupts functional. Peripheral clocks are stopped, but LPTimer, RTC, CMP, TSI, DAC can be used. NVIC is disabled (FCLK = OFF); AWIC is used to wake up from interrupt. On-chip voltage regulator is in a low power mode that supplies only enough power to run the chip at a reduced frequency. All SRAM is operating (content retained and I/O states held).
LLS (Low Leakage Stop)	State retention power mode. Most peripherals are in state retention mode (with clocks stopped), but LLWU, LPTimer, RTC, CMP, TSI, DAC can be used. NVIC is disabled; LLWU is used to wake up. **NOTE:** The LLWU interrupt must not be masked by the interrupt controller to avoid a scenario where the system does not fully exit stop mode on an LLS recovery. All SRAM is operating (content retained and I/O states held).
LLS3 (Low Leakage Stop3)	State retention power mode. Most peripherals are in state retention mode (with clocks stopped), but LLWU, LPTimer, RTC, CMP, TSI,DAC can be used. NVIC is disabled; LLWU is used to wake up. **NOTE:** The LLWU interrupt must not be masked by the interrupt controller to avoid a scenario where the system does not fully exit stop mode on an LLS recovery. All SRAM is operating (content retained and I/O states held).
LLS2 (Low Leakage Stop2)	State retention power mode. Most peripherals are in state retention mode (with clocks stopped), but LLWU, LPTimer, RTC, CMP, TSI, DAC can be used. NVIC is disabled; LLWU is used to wake up. **NOTE:** The LLWU interrupt must not be masked by the interrupt controller to avoid a scenario where the system does not fully exit stop mode on an LLS recovery. A portion of SRAM_U remains powered on (content retained and I/O states held). RAM2 partition. The MCU is placed in a low leakage mode by reducing the voltage to internal logic and powering down the system RAM3 partition. The system RAM2 partition can be optionally retained using STOPCTRL[RAM2PO].The system RAM1 partition, internal logic and I/O states are retained.
VLLS3 (Very Low Leakage Stop3)	Most peripherals are disabled (with clocks stopped), but LLWU, LPTimer, RTC, CMP, TSI, DAC can be used. NVIC is disabled; LLWU is used to wake up. SRAM_U and SRAM_L remain powered on (content retained and I/O states held).
VLLS2 (Very Low Leakage Stop2)	Most peripherals are disabled (with clocks stopped), but LLWU, LPTimer, RTC, CMP, TSI, DAC can be used. NVIC is disabled; LLWU is used to wake up. SRAM_L is powered off. A portion of SRAM_U remains powered on (content retained and I/O states held). The system RAM2 partition can be optionally retained using STOPCTRL[RAM2PO]. The system RAM1 partition contents are retained in this mode. Internal logic states are not retained.
VLLS1 (Very Low Leakage Stop1)	Most peripherals are disabled (with clocks stopped), but LLWU, LPTimer, RTC, CMP, TSI, DAC can be used. NVIC is disabled; LLWU is used to wake up. All of SRAM_U and SRAM_L are powered off. The 32-byte system register file and the 128-byteVBAT register file remain powered for customer-critical data. I/O states are held. Internal logic states are not retained.
VLLS0 (Very Low Leakage Stop0)	Most peripherals are disabled (with clocks stopped), but LLWU and RTC can be used. NVIC is disabled; LLWU is used to wake up. All of SRAM_U and SRAM_L are powered off. The 32-byte system register file and the 128-byte VBAT register file remain powered for customer-critical data. I/O states are held. Internal logic states are not retained. The 1kHz LPO clock is disabled and the power on reset (POR) circuit can be optionally enabled using CTRL[PORPO].
BAT (backup battery only)	This chip is powered down except for the VBAT supply. The RTC and the 128-byte VBAT register file for customer-critical data remain powered.

2.3.3 Run mode

- Selected after any reset unless LPRUN is selected by an FOPT control.
- On-chip voltage regulator is On, full capability,unless LPRUN is selected by an FOPT control.
- Stack pointer (SP), Program Counter (PC), and link register are set.

- ARM Cortex-M4 and M0+ processor exits reset and reads the start SP.
- ARM Cortex-M4 and M0+ processor reads the start PC from vector table.
- If recovering from VLLSx exit through reset, OSC and output pins are held state until ACKISO written.
- If NMI is low at de-assertion of reset exception processing for the NMI vector is executed prior to instruction at start PC.

2.3.4 High Speed Run mode

- On-chip voltage regulator is in High-Speed RUN mode.
- Selected by a write to RUNM control bits.
- Flash programming and erasing is not allowed.
- Will exit HSRUN to run mode when RUNM bit written or Reset.
- Cannot enter low power modes without first moving to Run.

2.3.5 Very Low Power Run mode (VLPR)

- Can enter VLPR from RUN or from reset if LPBOOT bit in FOPT is set.
- On-chip voltage regulator is in a mode that supplies only enough power to run the MCU at a reduced frequency.
- Core and bus frequency limited to 4 MHz.
- Flash frequency limited to 800 kHz–1 MHz.
- Clock changes are not allowed.
- Flash programming and erasing is not allowed.
- Flex Memory (EEPROM) programming is not allowed.
- Can enter VLPR using the internal fast IRC, or LIRC if using the MCG-Lite.
- Can enter VLPR using an external clock.
- Can enter all low power modes directly except normal STOP and WAIT.
- Can exit VLPR via a RUNM control bit write or, on some devices, by means of an interrupt.

2.3.6 Wait mode

- ARM Cortex-M4 and M0+ core enters Sleep mode.
- ARM Cortex-M4 and M0+ core is clock gated.
- NVIC remains sensitive to interrupts.
- Peripherals continue to be clocked.
- Reduce power by clearing clock gating bits in SCGCx registers.
- Flex memory (EEPROM) programming is not allowed.
- On interrupt, the ARM Cortex-M4 and M0+ core exits Sleep mode, Run mode processing resumes.
- Will exit WAIT in the same MCG mdoe present when WAIT was entered.

2.3.7 Very Low Power Wait mode (VLPW)

- Can only enter VLPW from VLPR.
- On-chip voltage regulator stays in a mode that supplies only enough power to run the MCU in a reduced frequency.
- ARM Cortex-M4 and M0+ core enters Sleep mode.
- ARM Cortex-M4 and M0+ core is clock gated.
- NVIC remains sensitive to interrupts.
- Peripherals continue to be clocked.
- Reduce power by clearing clock gating bits in SCGCx registers allow individual module clock control.
- Flex memory (EEPROM) programming is not allowed.
- Flash memory programming/erase is not allowed.

- Will exit VLPW in the same MCG mode present when VLPW was entered.
- Can optionally keep the external reference clock enabled (up to 16 MHz max).
- On interrupt, the ARM Cortex-M4 and M0+ core exits Sleep mode, VLPR or Run mode processing resumes.

2.3.8 Normal Stop mode

- ARM Cortex-M4 and M0+ core enters Deep Sleep mode.
- ARM Cortex-M4 and M0+ core is clock gated.
- INTC/NVIC is disabled.
- Wakeup Interrupt Controller (WIC) is used to wake from interruptions.
- System and peripheral clocks are stopped unless the debug module is active. On some MCUs asynchronous clock available to low power modules and the DMA for operation in STOP.
- Low Voltage Detect(LVD) and Low Voltage Warning(LVW) are fully operational in STOP.
- All SRAM content retained. Register file, I/O and oscillator states held.
- Can enter Stop from any MCG mode.
- MCU can be configured to leave reference clocks running.
- Can optionally keep the PLL enabled but the output will be gated off—MCG_C5[PLLSTEN].
- Can optionally keep the internal reference clock enabled—MCG_C1[IREFSTEN].
- Can optionally keep the external reference clock enabled—OSC_CR[EREFSTEN].
- Will exit Stop in the same clock mode when Stop was entered, except if Stop is entered when in PLL Engaged External (PEE) mode and PLLSTEN = 0 the MCG will be in PLL Bypassed External (PBE) mode when Stop is exited.

MCU current will be higher when using a debugger since the ARM Cortex-M4 core will need to have clocks alive in Stop with the debugger enabled.

2.3.9 Very Low Power Stop mode (VLPS)

- ARM Cortex-M4 and M0+ core enters Deep Sleep mode.
- ARM Cortex-M4 and M0+ core is clock gated.
- INTC/NVIC is disabled.
- WIC is used to wake from interruptions.
- System and peripheral clocks are stopped unless the debug module is active. On some MCUs asynchronous clock available to low power modules and the DMA for operation in VLPS.
- All SRAM content retained. Register file, I/O and oscillator states held.
- Can enter VLPS from any clock mode.
- MCU can be configured to leave reference clocks running.
- Can optionally keep the internal reference clock enabled—MCG_C1[IREFSTEN].
- Can optionally keep the external reference clock enabled—OSC_CR[EREFSTEN].
- Will exit VLPS in the same clock mode when VLPS was entered, except if VLPS is entered when in PLL Engaged External (PEE) mode and PLLSTEN = 0 the MCG will be in PLL Bypassed External (PBE) mode when VLPS is exited.

MCU current will be higher when using a debugger since the ARM Cortex-M4 core will need to have clocks alive in VLPS with the debugger enabled.

2.3.10 Low Leakage Stop modes (LLS ,LLS2 and LLS3)

- ARM Cortex-M4 and M0+ core enters Deep Sleep mode.
- ARM Cortex-M4 and M0+ core is clock gated.
- INTC/NVIC is disabled and interrupts need not be enabled.
- LLWU must be configured to enable the desired wake-up source.
- System and peripheral clocks are stopped. OSCERCLK is optionally available for use by some peripherals.

- All SRAM (LLS3) or partial SRAM (LLS2) contents retained. Register File retained, I/O and oscillator states held.
- Most peripherals are in state retention mode (can't operate).
- Can enter LLS, LLS2 or LLS3 from any clock mode.
- MCU is static with no clocks active (IREFSTEN and PLLSTEN have no effect).
- Can optionally keep the external reference clock enabled—OSC_CR[EREFSTEN].
- Will exit LLSx in the same clock mode when LLSx was entered, except if LLSx is entered when in PLL Engaged External (PEE) mode and PLLSTEN = 0 the MCG will be in PLL Bypassed External (PBE) mode when LLSx is exited.
- Upon a wake-up event, the WAKEUP bit in the SRS register is set.
- After taking the LLWU interrupt code, execution continues at the next instruction following the LLS mode entry.

2.3.11 Very Low Leakage Stop 3 mode (VLLS3)

- ARM Cortex-M4 and M0+ core enters Deep Sleep mode.
- ARM Cortex-M4 and M0+ core is clock gated.
- INTC/NVIC is disabled and interrupts need not be enabled.
- LLWU must be configured to enable the desired wake-up source.
- System and peripheral clocks are stopped as in VLLS3. OSCERCLK is opntionally available for use by some peripherals.
- Most modules are disabled.
- All SRAM content retained. Register File retained, I/O and oscillator states held.
- Can enter VLLS3 from any clock mode.
- Upon a wake-up event, exit VLLS3 through the reset flow. MCU comes up in FLL Engaged Internal (FEI) mode.
- Upon a wake-up event, the WAKEUP bit in the SRS register is set and the MCU executes the code from the reset vector.
- For Kinetis MCUs, the first enabled interrupt with the highest priority will execute.
- Can optionally keep the external reference clock enabled—OSC_CR[EREFSTEN].

2.3.12 Very Low Leakage Stop 2 mode (VLLS2)

- ARM Cortex-M4 and M0+ core enters Deep Sleep mode.
- ARM Cortex-M4 and M0+ core is clock gated.
- INTC/NVIC is disabled.
- LLWU must be configured to enable the desired wake-up source.
- System and peripheral clocks are stopped as in VLLS2.
- Only a few modules are able to operate.
- 4–8K SRAM content retained, Register File retained, I/O and oscillator states held.
- Can enter VLLS2 from any clock mode.
- Can optionally keep the external reference clock enabled.
- MCG is off with no clocks active (IREFSTEN and PLLSTEN have no effect).
- Upon a wake-up event, exit VLLS2 through the reset flow. MCU comes up in FEI mode.
- Upon a wake-up event, the WAKEUP bit in the SRS register is set and the MCU executes the code from reset unless awakened by the NMI interrupt.
- For Kinetis MCUs, the first enabled interrupt with the highest priority will execute.
- Can optionally keep the external reference clock enabled—OSC_CR[EREFSTEN].

2.3.13 Very Low Leakage Stop 1 mode (VLLS1)

- ARM Cortex-M4 and M0+ core enters Deep Sleep mode.
- ARM Cortex-M4 and M0+ core is clock gated.
- INTC/NVIC is disabled.

- LLWU must be configured to enable the desired wake-up source.
- System and peripheral clocks are stopped as in VLLS1.
- Only a few modules are able to operate.
- No SRAM content retained, Register File retained, I/O and oscillator states held.
- Can enter VLLS1 from any clock mode.
- Can optionally keep the external reference clock enabled.
- MCG is off with no clocks active (IREFSTEN and PLLSTEN have no effect).
- Upon a wake-up event, exit VLLS1 through the reset flow. MCU comes up in FEI mode.
- Upon a wake-up event, the WAKEUP bit in the SRS register is set and the MCU executes the code from Reset.
- For Kinetis MCUs, the first enabled interrupt with the highest priority will execute.
- Can optionally keep the external reference clock enabled—OSC_CR[EREFSTEN].

2.3.14 Very Low Leakage Stop 0 mode (VLLS0)

- ARM Cortex-M4 and M0+ core enters Deep Sleep mode.
- ARM Cortex-M4 and M0+ core is clock gated.
- INTC/NVIC is disabled.
- LLWU must be configured to enable the desired wake-up source.
- System and peripheral clocks are stopped as in VLLS0.
- Power on Protection (POR) optionally enabled.
- No SRAM content retained, Register File retained, I/O and oscillator states held.
- Can enter VLLS0 from any clock mode.
- The Low power Oscillator (1 KHz) clock is off.
- The external reference clock is disabled.
- MCG is off with no clocks active (IREFSTEN and PLLSTEN have no effect).
- RTC can remain active when 32,768 Hz square wave input is provided to the MCU.
- Upon a wake-up event, exit VLLS0 through the reset flow. MCU comes up in FEI mode.
- Upon a wake-up event, the WAKEUP bit in the SRS register is set and the MCU executes the code from reset.
- The first enabled interrupt with the highest priority will execute.

2.4 Power Efficiency – Typical Power Consumption

Mode	K70 – 150 MHz	K20 – 50 MHz	KL46z – 48 MHz
Run	89.9 mA	13.9 mA	6.3 mA *
VLP Run (VLPR)	1.4 mA	867 uA – 1.1 mA	292 ** - 522 uA ***
Wait	40.9 / 19.6^ mA	7.5 mA	3.3 - 4.4mA
VLP Wait (VLPW)	926 uA	509 uA	261 uA
Stop	1.3 mA	310 uA	212 uA
VLP Stop (VLPS)	250 uA	3.5 uA	2.8 uA
LL Stop (LLS)	250 uA	2.1 uA	2.0 uA
VLL Stop 3 (VLLS3)	5.6 uA	2.9 uA	1.5 uA
VLL Stop 1 (VLLS1)	2.8 uA	1.5 uA	650 nA
VLL Stop 0 (VLLS0)	-	176 - 367 nA	130 - 330 nA

Cortex-M0+

* Compute Operation enabled: 4.1mA @ 48MHz core / 24MHz bus)
** Compute Operation enabled: 188uA @ 4MHz core / 800kHz bus)
*** Running Coremark algorithm, KEIL 4.54 optimized for speed
^Reduced Frequency Wait

Figure 6. Power Efficiency - Typical Power Consumption

Note these are all typical current estimates at 3V and 25C. RUN and VLPR numbers are with all modules off(compute Operation) clocking option enabled. In RUN mode at maximum core frequency of 48MHz the current consumed is 4mA. In VLPR mode at maximum core frequency of 4MHz the current consumed is 200uA. In both these modes with all modules off clocking option enabled the energy saving peripherals can still be functional.

2.5 Ultra Low Power Modes Check List

2.5.1 Items to do before entering low power mode

- The SMC_PMPROT must be set to allow the stop modes and low power run mode you wish to enter. This register is a write once after reset register. If no low power mode is allowed and SLEEPDEEP = 1, executing the WFI instruction enters Normal STOP low power mode.
- If using an external clock source then the clock monitor must be disabled before entering any low power mode except WAIT.
- A wake up source should be enabled before entering a low power mode. An interrupt for VLPS, WAIT, or VLPW and an LLWU source for LLSx, VLLSx modes. If none are enabled Reset and NMI will still wake the MCU from all low power modes..

- Select the desired low power mode in the SMC_PMCTRL and SMC_STOPCTRL registers.
- Set/Clear ARM SLEEPDEEP bit for STOP/WAIT.
- All of the steps above can be done in an initialization phase. If there are other operations, taking at least 12 core clock cycles, in-between the set of the stop mode and WFI instruction execution no serialization is needed.
- The Wait For Interrupt instruction, WFI, causes immediate entry to sleep mode unless the wake-up condition is true.
- If the low power mode is switched immediately before entering the sleep mode then you must ensure the write to the SMC_PMCTRL or SMC_STOPCTRL register has completed before executing the WFI. This can be done with a read of the SMC_PMCTRL register prior to execution of the WFI instruction. This is serialization.

2.5.2 Low power mode using sleep-on-exit:

- SCR register Sleep-On-Exit mode is set with SLEEPDEEP=1.
- Execute WFI to enter the first time.
- Normal STOP, VLPS or LLS can be used with Sleep-on-Exit operation.
- If the SLEEPONEXIT bit of the SCR is set to 1, when the processor completes the execution of all exception handlers it immediately re-enters sleep mode.
- Low power peripheral and asynchronous DMA can operate in VLPS with Sleep-On-Exit mode set.
- Use this mechanism in applications that require the core to run only when a wake up exception occurs.

2.5.2 To enter VLPR you must:

- Be in BLPI (using fast IRC for the MCG or LIRC is using MCG_Lite) or BLPE clock mode.
- Set the core frequency to 4MHz or less and flash clock to 1MHz or less.
- Not have the slow IRC enabled.
- Disable the clock monitor.
- Disable FIRC on devices using the MCG_Lite.
- Write the SMC_PMCTRL RUNM bits to enter VLPR.

2.5.4 Exiting VLPR

- By Writing the SMC_PMCTRL RUNM bits to exit VLPR
- On K series with an interrupt.
- Exiting VLPR with a write to the RUNM bits does not change the flow of the code. No exception is generated.

2.5.5 Exiting LLS, LLS2, or LLS3

- By a low leakage wake up (LLWU) event, LLWU interrupt is pending after wake up and takes priority.
- Internal module flags can be cleared in the LLWU exception handlers (ISR) or if you leave the module flag set then after the LLWU ISR is complete you enter the module ISR to clear the flag. If you clear the module flag and do not want to enter the module ISR the NVIC clear pending interrupt flag bit must be cleared for the module. Serialization of operations is necessary to ensure the flag are cleared before returning from the LLWU or module ISRs.
- Code re-starts at the instruction following the WFI instruction after completing of all exception handlers.

2.5.6 Exiting VLLS3, VLLS2, VLLS1, or VLLS0

- VLLSx recovery is via the reset flow.
- After exiting any of the VLLSx modes, the IO remain latched until they are released by writing “1” to the ACKISO bit in the PMC.
- The GPIO must be re-configured before ACKISO is written.

- If there is no requirement to make use of any external IO, There is no need to re-configure the IO before re-entering VLLSx if you do not write the ACKISO bit.
- If the oscillator has been kept running in VLLS1,VLLS2, or VLLS3, it must be re-configured before ACKISO is written (unless it was configured in RTC OSC).
- RAM contents need to be re-initialized if using VLLS1 or VLLS0.
- RAM contents need to be re-initialized for RAMs not powered in VLLS2.
- The LLWU interrupt will be pending.
- If the wake up was by a module then the associated Module interrupt will also be pending.
- The interrupt taken will be the first one enabled in the reset flow.
- If the LLWU interrupt is the first enabled then the module interrupt flag must be cleared in the LLWU ISR.
- If the Module interrupt is the first enabled then the module ISR will clear the interrupt flag and the LLWU ISR will no longer be pending.
- Pin wake up flags must be cleared in the LLWU.
- VLLS wakeup will be reported in the RCM.

NOTE

VLLS0 mode is only available on some Kinetis devices.

2.6 Kinetis Energy Savings – Tips and Tricks

1. The absolute lowest power mode is "powered VDD OFF". This mode uses the RTC_WAKEUP to control the MCU VDD. On some later Kinetis devices with isolated VBAT power domains, like the K22F, K64, K24 and K65, there is a control output called RTC_WAKEUP_B that can be used to gate on the MCU VDD using an external component.
2. Use the FOPT options: There are multiple FOPT register bits such as LPBOOT to help reduce current at startup or disable an unwanted function like EZPORT mode entry.
3. Using the asynchronous DMA, in conjunction with a low power peripheral is technically another low power mode. While in the low power mode the peripheral, such as the LPUART, can send and receive data until buffers need tending. In STOP or VLPS with Core clock disabled:
    - UART data transfers still possible
    - PWM output changes possible with DMA
    - ADC data sampling possible with DMA reading result register
4. UART0/LPUART – Low Power UART:
    - Over-Sampling, Address Match, Asynchronous operation in conjunction with asynchronous DMA.
5. Low Power Timer (LPTMR) running off of the LPO offers a low resolution(typical 1ms) wakeup timer in all but the lowest power mode VLLS0.
    - Very low current added in all power modes.
    - Operates through all resets except POR and LVD.
    - Can run from the internal 1 KHz low power oscillator in all power modes except VLLS0.
6. Real Time Clock (RTC) running off an external square wave clock source is the lowest power option for a periodic wake up source.
    - Lowest current adder for a wakeup source.
    - Can operate in all power modes
    - Operates through all resets except POR and LVD.
    - Operates from external 32,768 Hz low power crystal oscillator in all power modes except VLLS0.
7. Unused pins should be configured in the disabled state, mux(0), to prevent unwanted leakage (potentially caused by floating inputs).
8. Turn off unnecessary module clock gates in SIM.
9. Use compiler options to reduce the current spikes from flash access for the literal pool. The literal pool is used as the base address of a module's register bank. The compiler could optimize the literal pool address.
10. Keep loops as small as possible to keep the code access within the size of the cache.
11. Execute commonly used loops or functions out of RAM instead of FLASH.
12. Use explicit register writes rather than Read-Modify-Writes.

13. ADC: Use 4x h/w averaging if 32x is not needed.
14. Use lower power mode with a slower clock speed if possible.
15. For GPIO accesses use the IOPORT where available. It takes less clock cycles to propagate the write to the output pin.
16. For battery applications with USB capability, add an isolation diodes in the supply line and use the USB to power the application when connected to the USB bus (can use on chip regulator).
17. Also for battery applications with USB capability, use the VBUS presences to wakeup the MCU for low power modes by connecting the VBUS through appropriately sized divide resistors to an LLWU wake-up input pin.
18. Compiler optimization levels can affect Idd.
19. Lowest Idd is not necessarily at the lowest power. Figure 7 shows a typical IDD versus Voltage curve for a Kinetis L MCU. Figure 8 below shows the overall power curve power = IDD * VDD. Notice that running the MCU at lower VDD values does reduce power consumption but at the expense of higher IDD's.

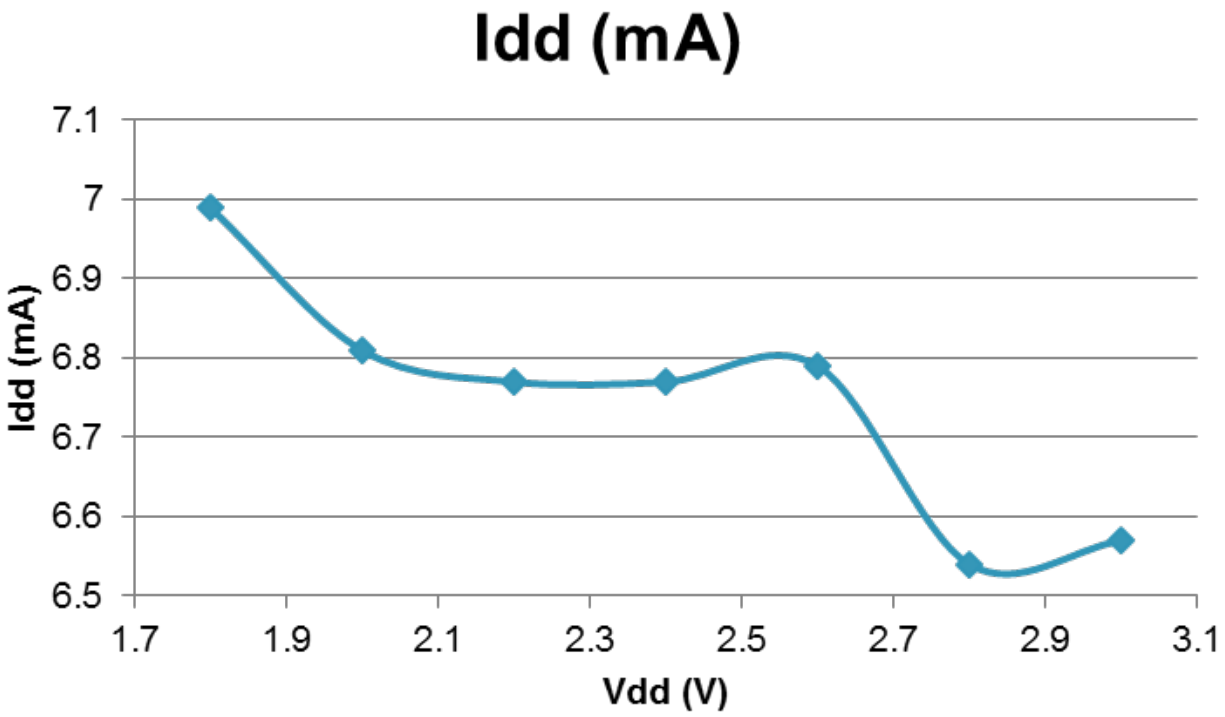


Figure 7. Kinetis MCUs typical Idd versus Vdd

Lowest Power is achieved at lowest voltage

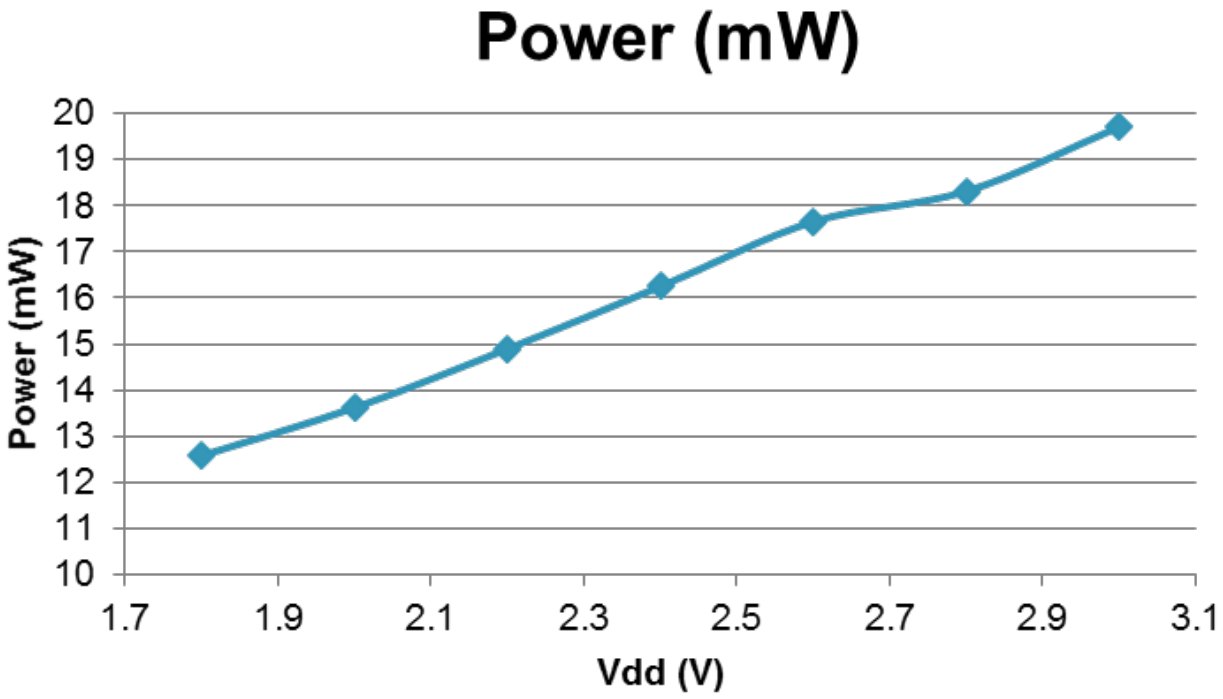


Figure 8. Kinetis MCUs Power versus Vdd

3 Quick Start Kinetis SDK

With the release of the Kinetis SDK, a growing number of Kinetis MCUs are supported with low-level software to set up and operate the MCU. The lowest level code in the SDK is the Hardware Abstraction Layer (HAL). The next level up from the HAL is the reference peripheral drivers and system services drivers. Power mode control is part of the system services driver set and allows for advanced device management when entering and exiting power modes, whereas the HAL functions allow you create your own customized driver and offer more flexibility.

The SDK release contains demo projects that provide use-case based examples that you can re-use for you own application. One such demo is the power_manager_demo. This code provides you a working power mode transition example for your study and re-use.

This quick start section demonstrates how you can use either the HAL or the system services to put the Kinetis MCU into the LLS low power mode from RUN mode. If your application began from a non-SDK based project or started as an SDK HAL based project, you will likely want to continue using the HAL (and the HAL example will assume this). Otherwise, if your project began from an SDK driver or system services based application, you will probably want to continue using the reference peripheral drivers (and the power manager system service example will assume this). First we will focus on using the HAL layer and then we will focus on using the prebuilt drivers.

3.1 LLS mode entry and exit - Kinetis Software Development Kit (KSDK) code example

3.1.1 MCU preparation using the SDK HAL functions

The first step in moving to a low power mode is to prepare the MCU for that low power mode. This includes turning off any peripheral modules that are not needed in the desired mode and any unused GPIO pins should be put in the disabled state. The SDK HAL provides easy to use functions to allow for this. An example of one such statement is shown below.

```
/* Disable PTC7 */
PORT_HAL_SetMuxMode(PORTC,7u,kPortPinDisabled );
```

This should be done for every pin that is used in the application.

3.1.2 Wakeup source configuration

Then, the exit method must be configured. This example uses a push button connected to a GPIO pin for the exit method. The first step in configuring the exit method is to initialize the GPIO pin connected to the push button. The code below demonstrates this using the KSDK HAL functions. Specifically, the following code will:

- **Configure a GPIO pin that will be used to wake from LLS mode**.
- **Configure pin as digital input**
- **Configure LLWU module pin PORTC6 (LLWU_P10) as a valid wake-up source**.

```
void main (void){
  /* Enable Port C6 to be a digital pin.   */
  SIM_HAL_EnablePortClock(SIM_BASE, HW_PORTC);

  /* PORTC_PCR6 */
  PORT_HAL_SetMuxMode(PORTC_BASE,6u,kPortMuxAsGpio);

  /* Configure GPIO input features. */
  PORT_HAL_SetPullCmd(PORTC_BASE, 6u, true);
  PORT_HAL_SetPullMode(PORTC_BASE, 6u, kPortPullUp);

  /* Set the LLWU pin enable bits to enable the PORTC6 input
  * to be a wake-up source.
  * WUPE10 is used in this case since it is associated with PTC6.
  * This information is in the Chip Configuration chapter of the Reference Manual.
  * 0b10: External input pin enabled with falling edge detection
  */
  SIM_HAL_EnableLlwuClock(SIM_BASE, HW_LLWU); // this bit may not exist on some MCUs
  LLWU_HAL_ClearExternalPinWakeupFlag(LLWU_BASE, 10u);

  //falling edge detection
  LLWU_HAL_SetExternalInputPinMode(LLWU_BASE, kLlwuExternalPinFallingEdge, kLlwuWakeupPin10);
```

3.1.3 SMC configuration using the SDK HAL functions

After preparing the MCU peripherals for entry into low power mode and configuring the wakeup source, it is time to configure the SMC (or MC depending on your specific MCU). Follow these steps:

- First, the power mode protection register must be configured to allow entry into the desired mode. This is a write once register and can only be written once after every reset. The recommended location for writing this register is in your startup code.

```
/* Power mode protection initialization */
SMC_HAL_SetProtection(SMC, kAllowPowerModeLls);
```

NOTE

Multiple power modes may be enabled by simply OR'ing the appropriate masks together in the function call. For example, if VLPS mode were to be allowed as well, the function call would become SMC_HAL_SetProtection(SMC, kAllowPowerModeLls | kAllowPowerModeVlps);

- Next the stop mode, and sub-mode must be configured. Once these are configured, the sleep bit in the ARM System Control Block (SCB) should be set and then the Wait For Interrupt (WFI) instruction should be executed. The SDK HAL provides a convenient way of performing these tasks in a single function call. This is the SMC_HAL_SetMode function and an example of its use is shown below:

```
/* First, setup an smc_power_mode_config_t variable to be passed to the SMC
 * HAL function
 */
smc_power_mode_config_t smc_power_config = {
    .powerModeName = kPowerModeLls,          // Set the desired power mode
    .stopSubMode = kSmcStopSub3,             // Set the sub-mode if necessary
#if FSL_FEATURE_SMC_HAS_LPWUI
    .lpwuiOption = false,                   // Set the LPWUI option if available
    .lpwuiOptionValue = kSmcLpwuiDisabled,
#endif
#if FSL_FEATURE_SMC_HAS_PORPO
    .porOption = false,                  // Set the PORPO option.  Only needed
    .porOptionValue = kSmcPorEnabled,   //  if entering VLLS0
#endif
#if FSL_FEATURE_SMC_HAS_PSTOPO
    .pstopOption = false,               // Only needed if entering a PSTOP mode
    .pstopOptionValue = kSmcPstopStop,
#endif
};

/* Disable clock monitor before moving to a STOP mode */
CLOCK_HAL_DisableOsc0Monitor(MCG_BASE);

/* Now call the SMC HAL SetMode function to move to the desired power mode. */
SMC_HAL_SetMode(SMC_BASE, &smc_power_config);
```

NOTE

The SMC_HAL_SetMode function simply configures the SMC control registers and executes the WFI instruction and performs the same actions every time regardless of whether or not the registers need to be written. If your application calls for more advanced functions, such as faster low power entry times or sleep-on-exit usage, there are ways to accomplish these tasks. They are presented here:

- Faster low power entry times: Once the SMC_HAL_SetMode function has been called, the SMC's control registers are still in the same configuration once the function exits. So if the same low power mode is desired on subsequent entries, you can simply execute the WFI instruction as shown:

  ```
  __WFI();
  ```

- Sleep-on-exit: To use the sleep on exit feature, simply set the sleep-on-exit bit in the System Control Register (SCB->SCR) before calling the SMC_HAL_SetMode function. The SMC_HAL_SetMode function performs only read-modify write operations on the SCB->SCR register so the configuration of the sleep-on-exit bit will be preserved. An example of how to do this is shown below:

  ```
  /* Set the Sleep-on-Exit bit */
  SCB->SCR |= SCB_SCR_SLEEPONEXIT_Msk;

  /* Then call the SMC_HAL_SetMode function */
  SMC_HAL_SetMode(SMC_BASE, &smc_power_config);
  ```

3.1.4 Exiting Low Leakage Stop (LLS) low-power mode

The MCU is now in LLS mode. The wakeup pin must be asserted to wake the MCU from LLS (in this case, a falling edge on Port C pin 6). At a minimum, the following code must be executed to clear the wake-up event flag in the LLWU register (note that for LLS or VLLSx modes, interrupts do not need to be enabled but can be to simplify your LLS or VLLS exit procedure).

```
  /* after exiting LLS (can be in LLWU interrupt service Routine)
  * clear the wake-up flag in the LLWU-write one to clear the flag
  */
  if (LLWU_HAL_GetExternalPinWakeupFlag(LLWU_BASE, 10u)) {
      LLWU_HAL_ClearExternalPinWakeupFlag(LLWU_BASE, 10u);
  }
} /* end of main */
```

NOTE

If interrupts are enabled, the execution resumes with the appropriate interrupt service routine. If interrupts are disabled, the execution resumes with the instruction after the WFI, or STOP instruction.

3.1.5 MCU preparation using the power manager system service

MCU preparation for low power entry is the same using the power manager system service as it is when using the HAL with one difference. Any peripheral module that is not needed in the desired mode should still be turned off and any GPIO pin that will not be used should still be put in the disabled state. The difference is that a callback function can be used. An example of a callback function to do this is as follows:

```
/* Structure for callback data */
typedef struct {
    uint32_t counter;
    uint32_t status;
    uint32_t err;
} callback_data_t;

/* Callback structure definition to be used by the Power Manager variables */
typedef struct {
    callback_data_t none;
    callback_data_t before;
    callback_data_t after;
    power_manager_callback_type_t lastType;
    uint32_t err;
} user_callback_data_t;

/* This is the declaration of the callback function that can be used
     * in power mode transitions
     */
    user_callback_data_t callbackData0;

    power_manager_static_callback_user_config_t callbackCfg0 = { callback0,
            kPowerManagerCallbackBeforeAfter,
            (power_manager_callback_data_t*) &callbackData0 };

    power_manager_static_callback_user_config_t *callbacks[] =
            { &callbackCfg0 };

/*
 * Power manager callback implementation code
 *
 * This section defines what the power manager functions do before
```

```
 * and after the power manager mode transitions occur.  It also defines
 * what to do if no callback is provided.
 *
 */
power_manager_error_code_t callback0(power_manager_callback_type_t type,
        power_manager_user_config_t * configPtr,
        power_manager_callback_data_t * dataPtr) {

    user_callback_data_t * userData = (user_callback_data_t*) dataPtr;
    power_manager_error_code_t ret = kPowerManagerError;

    switch (type) {

    case kPowerManagerCallbackNone:

        userData->none.counter++;
        ret = kPowerManagerSuccess;
        break;
    case kPowerManagerCallbackBefore:

        /* Disable PTC7 and any other pins that are not used */
        PORT_HAL_SetMuxMode(PORTC,7u,kPortPinDisabled );

        userData->before.counter++;
        ret = kPowerManagerSuccess;
        break;
    case kPowerManagerCallbackAfter:

        userData->after.counter++;
        ret = kPowerManagerSuccess;
        break;
    default:
        userData->err++;
        break;
    }

    userData->lastType = type;

    return ret;
}
```

3.1.6 Wakeup source configuration using the power manager system service

The first step in configuring a wakeup source is adding the desired pin in the _gpio_pins enumeration. The code below is an example of this:

```
/*! @brief gpio pin names.*/
/*!*/
/*! This should be defined according to board setting.*/
enum _gpio_pins
{
    kGpioSW1         = GPIO_MAKE_PIN(GPIOC_IDX, 6),  /* TWR-K22F120M SW1 */
};
```

Next, the pin must be included in an input pin structure that defines all of the necessary parameters for that pin to work in your application. This structure is used by the GPIO driver to properly initialize the desired pin:

```
  /* Declare Switch pins */
gpio_input_pin_user_config_t switchPins[] = {
    {
        .pinName = kGpioSW1,
        .config.isPullEnable = true,
        .config.pullSelect = kPortPullUp,
        .config.isPassiveFilterEnabled = false,
        .config.interrupt = kPortIntDisabled,
    },
```

```
    {
        .pinName = GPIO_PINS_OUT_OF_RANGE,
    }
};
```

Another important action in enabling a GPIO pin as a low-power wakeup source is to configure the port mux field correctly. This is done in the hardware_init function in hardware_init.c. It is also important to only enable those pins that will be used (for minimum current draw). An example of how to enable a pin for a GPIO function is presented below.

```
/* enable clock for PORTC */
CLOCK_SYS_EnablePortClock(2);

/* enable PORTC pin 6 as a GPIO */
PORT_HAL_SetMuxMode(PORTC,6u,kPortMuxAsGpio);
```

Now the GPIO driver may be called to initialize the desired pins.

```
 /* Initialize the pins used for switches and LEDs only */
    GPIO_DRV_Init(switchPins, NULL);
```

Next, since the low-leakage power modes (LLS, VLLSx) use the Low-Leakage Wakeup Unit (LLWU) as the recovery method, the LLWU must be configured. To do this, we use the LLWU HAL functions as shown below.

```
/*************************************************************
     * LLWU pin initialization
     *
     * First, clear the associated flag just to be sure the device
     * doesn't immediately enter the LLWU interrupt service routine
     * (ISR). Then enable the interrupt
     *************************************************************/
    LLWU_HAL_ClearExternalPinWakeupFlag(LLWU_BASE, pinEn);

    LLWU_HAL_SetExternalInputPinMode(LLWU_BASE, riseFall, pinEn);
```

In the function calls above, LLWU_BASE is the register location of the LLWU (which is defined in the device specific header file), risefall is a variable of type llwu_external_pin_modes_t (which is defined in fsl_llwu_hal.h), and pinEn is a uint8_t variable that specifies which pin in the LLWU to enable.

3.1.7 Entering Low Leakage Stop (LLS) low-power mode using the power manager system service

The Kinetis SDK power manager system service utilizes an array of power configurations to interface to the application. The user is expected to create an array of all of the power configurations to be used in the application. An example of an array definition to be used by the power manager is shown below.

```
    /* Power manager configuration variables that configure
     * the named low power mode
     */
    power_manager_user_config_t llsConfig;
    power_manager_user_config_t runConfig;

    /* This is the power manager configuration variable.  The order of this
     * list determines the index that should be passed to the Setmode
     * function to enter the desired power mode.
     */
    power_manager_user_config_t const *powerConfigs[] = { &llsConfig, &runConfig };
```

After declaring the power manager user configurations and defining the array for the power manager function calls, the power manager configuration variables must be configured. An example is shown below.

```
    /* Define the power mode configurations */
    /* Define all of the parameters for the LLS configuration */
    llsConfig.mode = kPowerManagerLls;
    llsConfig.policy = kPowerManagerPolicyAgreement;
```

```
#if FSL_FEATURE_SMC_HAS_LPWUI
    llsConfig.lowPowerWakeUpOnInterruptOption = true;
    llsConfig.lowPowerWakeUpOnInterruptValue = kSmcLpwuiEnabled;
#endif
    llsConfig.sleepOnExitValue = false;
    llsConfig.sleepOnExitOption = false;
#if FSL_FEATURE_SMC_HAS_PORPO
    llsConfig.powerOnResetDetectionOption = true;
    llsConfig.powerOnResetDetectionValue = kSmcPorEnabled;
#endif
#if FSL_FEATURE_SMC_HAS_LPOPO
    llsConfig.lowPowerOscillatorOption = true;
    llsConfig.lowPowerOscillatorValue = kSmcLpoEnabled;
#endif
```

NOTE

If the device you are working with has LLS submodes (i.e., LL2 and LL3) then you will want to specify that by using the *.mode variables kPowerManagerLls2 and kPowerManagerLls3.

After configuring the power manager user configuration variables, the power manager needs to be initialized and the callbacks need to be registered. The following code is an example of how to do this. .

```
/* Initialize the power manager module */
POWER_SYS_Init(&powerConfigs,
     sizeof(powerConfigs)/sizeof(power_manager_user_config_t *),
     &callbacks,
     sizeof(callbacks)/sizeof(power_manager_callback_user_config_t *));
```

The above function simply informs the power manager system service of what configurations are available. To use the drivers to move to the low power mode, use the POWER_SYS_SetMode function and pass the index to the configuration in the powerConfigs array defined in the code above. This function returns a status variable to acknowledge that the mode change occurred successfully or unsuccessfully (note that this code will not be executed if moving to a VLLSx mode, as these modes are exited through the reset sequence. The following is an example.

```
/* Now Issue power mode change */
ret = POWER_SYS_SetMode(powerModeLls1Index);

/* The code execution should not get here on successful entries.
 * However, unsuccessful entries should make it to this code and
 * report errors
 */
if (ret != kPowerManagerSuccess)
{
    printf("POWER_SYS_SetMode(powerModeLls1Index) returns : %u\n\r", ret);
}
```

To explore these drivers further, please refer to the power_manager_hal_demo in your SDK installation found at <SDK root>/apps/<board name>/demos/power_manager_hal_demo.

4 Quick Start

4.1 LLS mode entry and exit - bare metal code example

For those who want to quickly see low-power operation, simply add the following lines of code to enter Low Leakage Stop (LLS) mode. To enable a wake-up pin, follow the steps in the initialization section below. The initialization section also disables the clock monitor, which is needed only if the application is running from an external clock source.

I chose to demonstrate LLS mode since it is one of the more useful low power mode. It exits the low power mode on the next instruction following the WFI instruction that put the MCU into the low power mode. LLS is very fast to wakeup, as quickly at 4 us in FLL engaged internal (FEI) clock mode at 100 MHz. It is the first mode in the sequence of low power modes to need the LLWU to wake up the MCU from low power mode. LLS mode maintains all register and SRAM though the low power mode.

4.1.1 Initialization

The code below will:

- **Configure a GPIO pin that will be used to wake from LLS mode**.
- **Configure pin as digital input** (Kinetis device shown)
- **Configure LLWU module pin PORTE1 (LLWU_P0) as a valid wake-up source**.

```
void main (void){
  unsigned int dummyread;
  /* Enable Port E1 to be a digital pin.   */
  SIM_SCGC5 = SIM_SCGC5_PORTE_MASK;
  PORTE_PCR1 =(PORT_PCR_ISF_MASK |  //clear flag if there
              PORT_PCR_MUX(01) |   //set pin functionality -GPIO
              PORT_PCR_IRQC(0x0A)| //falling edge interrupt enable
              PORT_PCR_PE_MASK |   // pull enable
              PORT_PCR_PS_MASK);   // pullup enable

  /* Set the LLWU pin enable bits to enable the PORTE1 input
  * to be a wake-up source.
  * WUPE0 is used in this case since it is associated with PTE1.
  * This information is in the Chip Configuration chapter of the Reference Manual.
  * 0b10: External input pin enabled with falling edge detection
  */
  SIM_SCGC4 = SIM_SCGC4_LLWU_MASK; // this bit may not exist on some MCUs
  LLWU_PE1 = LLWU_PE1_WUPE0(2); //falling edge detection
```

- **If an external clock source is used, disable the clock monitor(s)**

```
  /* Need to make sure the clock monitor(s) is disabled if using
  * an external clock source. This assumes OSC0 is used as the
  * external clock source and that the clock gate for MCG has
  * already been enabled.
  */
  MCG_C6 &= ~MCG_C6_CME0_MASK; //CME=0 clock monitor disable
```

4.1.2 Entering Low Leakage Stop (LLS) low-power mode

The code below will:

- **Allow the MCU to go into LLS mode. The PMPROT register is a write-once register and should be written to allow all needed low-power modes. If only LLS and VLLS1 are needed, then enable only those modes.**

```
/* Write to PMPROT to allow LLS power modes */
  #ifdef MC1
  MC_PMPROT = MC_PMPROT_ALLS_MASK; // (MC1)
  #else
  SMC_PMPROT = SMC_PMPROT_ALLS_MASK;    // (MC2)
  #endif
```

- **Set the LPLLSM (for MC1) or STOPM (for MC2) bits to select the LLS low-power mode. If VLLS1 mode is desired there is one more write for MC2 MCUs.**
- **Set the VLLSM bits to 0b11 for LLS3. If LLS2 with reduced SRAM retention is desired choose 0b10.**

```
#ifdef MC1
MC_PMCTRL  =  MC_PMCTRL_LPLLSM(3); // (MC1) set LPLLSM = 0b11
#else
SMC_PMCTRL =  SMC_PMCTRL_STOPM(3); // (MC2) Set STOPM = 0b11
#endif

#ifdef MC4
SMC_VLLSCTRL =  SMC_VLLSCTRL_VLLSM(3);  // (MC4) Set VLLSM = 0b11 for LLS3
#endif
```

- **For Kinetis devices, set the SLEEPDEEP bit in the core, and then execute the Wake From Interrupt (WFI) instruction to enter LLS mode.**
- **Serialization read of the PMCTRL register is needed to ensure that the core does not stop operation before the write to set the low-power mode is completed. See Power mode transition code section for details on serialization.**

```
/*wait for write to complete to SMC before stopping core */
dummyread = SMC_PMCTRL;

/* Set the SLEEPDEEP bit to enable deep sleep mode */
SCB_SCR |= SCB_SCR_SLEEPDEEP_MASK;

/* WFI instruction will start entry into low-power mode */
asm("WFI");
```

- **Serialization read of the PMCTRL register is needed to ensure that the core does not stop operation before the write to set the low-power mode is completed. See Power mode transition code section for details on serialization.**

```
/*wait for write to complete to SMC before stopping core */
dummyread = SMC_PMCTRL;

SIM_SOPT4 = SIM_SOPT4_STOPE_MASK;  //enable stop mode entry
asm (  stop #0x2000; )
```

4.1.3 Exiting Low Leakage Stop (LLS) low-power mode

- **The MCU is now in LLS mode.**
- **The LLWU wake-up circuitry operates independently of interrupts.**
- **Interrupts do not need to be enabled nor does the LLWU interrupt vector need to be enabled for the MCU to wake from LLS or VLLSx low-power modes.**
- **If a falling edge event occurs on PORTE1, the MCU will exit LLS and return to Run mode.**
- **After exiting from LLS, the code below clears the wake-up event flag in the LLWU flag register.**
- **If interrupts and the LLWU interrupt are enabled, execution resumes with the LLWU interrupt service routine. Then code execution returns to the instruction following the WFI or STOP instruction.**
- **If interrupts are disabled, the execution resumes with the instruction after the WFI, or STOP instruction.**

```
/* after exiting LLS (can be in LLWU interrupt service Routine)
* clear the wake-up flag in the LLWU-write one to clear the flag
*/
if (LLWU_F1 & LLWU_F1_WUF0_MASK) {
    LLWU_F1 |= LLWU_F1_WUF0_MASK;
}
} /* end of main */
```

5 Reset Management

5.1 Resets and holds on I/O for low-power modes

Managing resets in the Kinetis MCU has become more complicated than previous microcontrollers with the introduction of the new low-power modes that recover through the reset flow. A reset is not a simple “return all modules to a default state” function anymore either. Many of the modules that are used to wake the MCU from low power modes will not clear the register state with certain reset types.

In the Reset section is an extended discussion on all of the types of reset. Here is a short list of Reset types:

1. Power On Reset (POR) and Low Voltage Detect (LVD) resets that occur when power is initially applied to the MCU. With multiple power domains there is a POR reset for both the VDD power domain and the VBAT power domain. A POR occurance in the VBAT can only be detected with the RTC invalid flag.
2. External pin reset (PIN) This pin is open drain and has an internal pullup device. Asserting RESET wakes the device from any mode. If waking from a VLLSx low power mode the hold on the I/O pins is disabled.
3. Computer operating properly (COP) watchdog timer
4. Technically not a reset source but if waking from VLLSx low power mode the LLWU Low leakage wakeup Unit does result in a reset. The (WAKEUP) bit is set.
5. Multipurpose clock generator loss-of-clock (LOC). The LOC reset will assert if the clock monitor is enabled when entering stop or low power modes. This is cause by the internal slow IRC stopping in these low power modes. The slow IRC is used as a reference to the clock monitor.
6. MCG loss-of-lock (LOL) reset
7. Stop mode acknowledge error (SACKERR)
8. Software reset (SW)
9. Core Lockup reset (LOCKUP)
10. EzPort reset
11. MDM-AP system reset request

During the reset flow that occurs with the recovery from VLLSx low power modes, the WAKEUP bit in the SRS register is set and the following modules, LPTMR, RTC, CMP, TSI, PMC, SMC, are not reset. This detail is identified in the note before each register bit descriptions in the reference manual of the MCU. It is worded like this:

" NOTE This register is reset on Chip POR not VLLS and by reset types that trigger Chip POR not VLLS. It is unaffected by reset types that do not trigger Chip POR not VLLS. See the Reset section details for more information."

As noted in the LLWU reset the LPTMR registers are reset only on POR and LVD type resets. So this module can operate continuously through all power modes and all reset types except the POR and LVD resets. There are a number of other modules like this.

The LLWU reset occurs when recovering from VLLS0, VLLS1, VLLS2, or VLLS3. Using the SRS status register and mode controller control registers, you can identify whether the reset is from the LVD, POR, or an LLWU wake-up event. If it is an LLWU reset, the WAKEUP bit in the SRS register is set and the values in the mode controller control registers will identify which mode the MCU was recovering from.

How to manage all of the different reset types requires some additional consideration. Some modules do not get reset to default states with reset. Some or all of the RAM can be retained and will not require re-initialization by the startup code.

5.1.1 Hold on I/O and oscillator

When in VLLSx and LLS low power modes, and while exiting VLLSx low-power modes without the Reset pin, there is a hold on all of the I/O states and the system oscillator. The hold is released automatically when LLS is exited, but most I/O pins must be released manually when exiting from VLLSx modes.

GPIO: Most of the digital I/O pins on all Kinetis MCUs default to a high-impedance mode after a non-VLLSx reset recovery. However, upon a VLLSx (LLWU) reset recovery, the state of the pins is held. For instance, if a port pin was configured as an output driving high, it would continue to drive high until the hold was released. If the output port was not re-initialized before the hold was released, then the pin would temporarily become a high-impedance pin. This might glitch the output low for a short time until the software sets the pin to drive the correct output state.

OSC: If the oscillator is enabled to operate in VLLSx or LLS modes, then it continues to function when the MCU is in these modes and when exiting the VLLSx low-power modes. The only reason to enable the OSC in these low-power modes is to clock a module from the oscillator's ERCLK output.The registers that control the oscillator are reset when VLLSx modes are exited. The actual control signals are latched while in VLLSx modes. When the hold is released the oscillator control signals will be updated with the register values. If the oscillator is not configured correctly before releasing the hold it will stop running.

```
PMC_REGSC |= PMC_REGSC_ACKISO_MASK; //write to release hold on I/O
```

5.1.1 Releasing the hold: IF or WHEN

The hold on I/O and OSC is maintained in LLS mode and automatically released upon mode exit without software intervention.

VLLSx modes: The hold on I/O and OSC is maintained in VLLSx mode. During the reset recovery the hold is released automatically on the pins used for debug purposes and for the rest of the pins, when the software writes to the ACKISO bit in the PMC module.

The IF question:

There may be a time when you do not want to release the hold at all. You might want to check an internal condition, like the real time clock, and then go right back into one of the VLLSx low-power modes.

The WHEN Question:

As the MCU is waking from the VLLSx modes, and if you do not want to glitch your I/O or temporarily stop the oscillator, the code needs to sequence through the initialization process so that all of the I/O and the oscillator are re-initialized prior to releasing the hold. This, of course, is very application dependent..

1. Initialize the oscillator module and the MCG oscillator control bits as in this example

```
// OSC_CR must enable external clock and enable in stop
OSC_CR = OSC_CR_EREFSTEN_MASK | OSC_CR_ERCLKEN_MASK;
```

2. Initialize all digital I/O including GPIO, serial interfaces, comparators driving outputs, timers, and so on.
3. Release hold with write to ACKISO.

Because the required sequence is application dependent, none of the sample code or bare-bones projects have an example of this sequence.

5.2 Identifying reset types

There are methods to identify the new reset types shown in the previous section. The SRS registers, now residing in the Reset Control Module (RCM), along with the settings in the SMC control registers can identify which mode the MCU is recovering from.

The power-up states of the SRS register will indicate POR and LVD resets. If the POR and LVD bits are set in the SRS register, then you know that all MCU registers are set to their default state and the RAM and register file register are not initialized.

If there is another source of reset identified in the SRS registers, then you have some choices to make about how you handle the reset recovery. The reset flow after a recovery from VLLS0, VLLS1, VLLS2, or VLLS3 sets the WAKEUP bit in the SRS register. The WAKEUP bit indicates that the MCU is waking up via reset from these low-power modes. Using the fact that the WAKEUP bit is set you can interpret the control registers in the SMC, such as PMPROT, PMCTRL,STOPCTRL and on some MCUs the VLLSCTRL to identify which low-power modes you are exiting. These registers reflect the state that the MCU low power mode was entered and are not cleared by the LLWU reset.

5.3 Initialization of variables heap and stack space in C

Low-power modes LLS, LLS2, VLLS2, and VLLS3 have some or all of the RAM retained throughout the entry, the time in the low-power mode, and recovery. The Register File Modules, the system register file, and the RTC register file are maintained in all power-down modes including VLLS0 and VLLS1. The MCU also has built-in flash or EEPROM memory that can be used for non-volatile data storage.

The management of these memories can get creative. If the variable space, heap, and/or stack is retained, it need not be re-initialized during a reset flow.

5.3.1 RAM and register file retention

The entire RAM is not retained in all low-power modes; however, the register file contents are maintained. There is a handy table that describes the status of each of the modes in all of the low-power modes. It is the “Module operation in low power modes” table in the Power Management section of the reference manual. If you refer to this table, then you will note that RAM is retained in VLLS2, but no RAM is retained in VLLS1, VLLS0, or when the MCU is powered off.

5.3.2 Testing for low-power mode recovery

The C code below tests the SRS register bits for each reset source. Assuming that the wake-up bit was set in the SRS register, the SMC module registers are read to determine which low-power mode the MCU is recovering from. With this information, the recovering MCU can choose to bypass some of the tradition reset initialization.

NOTE

This is an example for rev 2.x Kinetis silicon, please consult the reference manual for the bit and register definitions for the MCU of choice.

```
if (RCM_SRS1 & RCM_SRS1_SW_MASK)
  printf("Software Reset\n");)
if (RCM_SRS1 & RCM_SRS1_LOCKUP_MASK)
  printf("Core Lockup Event Reset\n");
if (RCM_SRS1 & RCM_SRS1_JTAG_MASK)
  printf("JTAG Reset\n");
if (RCM_SRS0 & RCM_SRS0_POR_MASK)
  printf("Power-on Reset\n");
if (RCM_SRS0 & RCM_SRS0_PIN_MASK)
  printf("External Pin Reset\n");
if (RCM_SRS0 & RCM_SRS0_WDOG_MASK)
  printf("Watchdog(COP) Reset\n");
if (RCM_SRS0 & RCM_SRS0_LOC_MASK)
  printf("Loss of Clock Reset\n");
if (RCM_SRS0 & RCM_SRS0_LVD_MASK)
  printf("Low-voltage Detect Reset\n");
if (RCM_SRS0 & RCM_SRS0_WAKEUP_MASK)
{
  printf("[outSRS]Wakeup bit set from low power mode exit\n");
```

```
  printf("[outSRS]SMC_PMPROT    = %#02X \r\n", (SMC_PMPROT))   ;
  printf("[outSRS]SMC_PMCTRL    = %#02X \r\n", (SMC_PMCTRL))   ;
  printf("[outSRS]SMC_VLLSCTRL = %#02X \r\n", (SMC_VLLSCTRL));
  printf("[outSRS]SMC_PMSTAT    = %#02X \r\n", (SMC_PMSTAT))   ;

if ((SMC_PMCTRL & SMC_PMCTRL_STOPM_MASK)== 3)
          printf("[outSRS] LLS exit \n") ;
if (((SMC_PMCTRL & SMC_PMCTRL_STOPM_MASK)== 4) &&
    ((SMC_VLLSCTRL & SMC_VLLSCTRL_VLLSM_MASK)== 1))
          printf("[outSRS] VLLS1 exit \n") ;
if (((SMC_PMCTRL & SMC_PMCTRL_STOPM_MASK)== 4) &&
    ((SMC_VLLSCTRL & SMC_VLLSCTRL_VLLSM_MASK)== 2))
          printf("[outSRS] VLLS2 exit \n") ;
if (((SMC_PMCTRL & SMC_PMCTRL_STOPM_MASK)== 4) &&
   ((SMC_VLLSCTRL & SMC_VLLSCTRL_VLLSM_MASK)== 3))
          printf("[outSRS] VLLS3 exit \n") ;

}
```

5.4 Boot sequence

There are four basic boot sequences: normal boot from flash memory, boot from ROM, boot up in EZPORT mode, and reset with the debugger enabled. Consult the reference manual for your Kinetis device for the boot details.

Booting from flash has two options, normal boot and low-power boot. The main difference is how quickly the boot sequences happen and the current consumed during boot. To control this boot option, you must leave the LPBOOT bit. in the FOPT register set for normal boot, or configure the LPBOOT bits for low-power boot mode.

For Kinetis devices with EZPORT, EZPORT boot will occur if the EZPORT chip select input is held low during a POR or LVD reset. If this mode is not desired, you can opt out of the EZPORT boot mode by clearing the corresponding EZPORT disable bit in the FOPT flash register. With this bit clear, there will be no inadvertent EZPORT boot mode entries. If the EZPORT mode is needed at a later time, the MCU can be mass erased and the new setting for the FOPT register can be programmed.

The debug mode boot is not technically a separate boot mode, but is referenced here with reference to how it affects low power modes. If you reset the MCU with the debugger connected and active, some of the MCU low power modes act differently than they would in the normal operating mode of your application. The current reading of the MCU in STOP and VLPS modes is higher than in the applications mission mode (with the debugger disconnected) because the debugger module clocks are kept alive to allow for debugging through STOP and VLPS.

To ensure that your application is in mission mode, disconnect the debugger and power cycle the VDD to cause a POR reset. If the state of the RAM requires to be maintained a POR is not possible, so enter VLLS3 low power mode and recover and the debugger will be disabled.

6 Dynamic and Static Power Management

6.1 Power management through clock control

Clocking the circuitry of the MCU takes power. Careful control of the system clocking can make a large impact in the average current consumption of the application.

The faster the clock frequency the more power the circuit consumes. The MCU has many clocks and clock controls to enable the power management of the MCU through clock control. The MCU can function from internal or external clock sources and can use the internal reference clocks, FLL or PLLs to set the clock speed to the CPU, bus, the external bus, and the flash

memory and on some Kinetis MCUs the peripherals. The RTC module and RTC_OSC on some MCUs are independent clock domains from the core, platform and other peripherals. This means that even with the VDD or the MCU turned off, the RTC can retain autonomous operation to keep time or provide a wake-up event for the MCU.

Internal module power management control is possible. The peripheral modules have clock gate controls in the SIM that can disable an unused or idle module clock. This is an effective power management control on a per-module level.

6.2 Power management using low-power modes

Kinetis MCUs have up to eleven distinct power modes. There are four dynamic power modes and up to seven static power modes. To use these power modes for power management most effectively, it is important to understand the various trade-offs of each of the modes: what modules are available, how fast is the wake-up time, what memory is retained, what sections of the MCU are powered off, and what is the state of the I/O pins.

The resources needed to understand these trade-offs are identified throughout the MCU reference manuals.

7 Clock Operation in Low-power Modes

There are a number of clocks in the MCU. All can function in Run and Wait power modes. Some can be optionally turned on when operating in some of the low-power modes.

The clocks for the MCU are generated in one of the following type of clock modules. Refer to the reference manual of your Kinetis device for clock specifics.

1. Multipurpose Clock Generator Module (MCG)
2. Multipurpose Clock Generator Lite Module (MCG-Lite)
3. Internal Clock Source (ICS)

There are clocks generated in other modules including but not limited to:

1. System Oscillator (OSC)
2. Real Time Clock (RTC)
3. Power Management Controller (PMC)
4. USB crystal-less clock source (48MHz IRC)

These, along with the SIM, control the clock selection and distribution of these clocks in the various power modes. An example of how these modules fit together is shown in figure 9. For further study, and to understand the strict rules for clock mode transitions, please see the reference manuals.

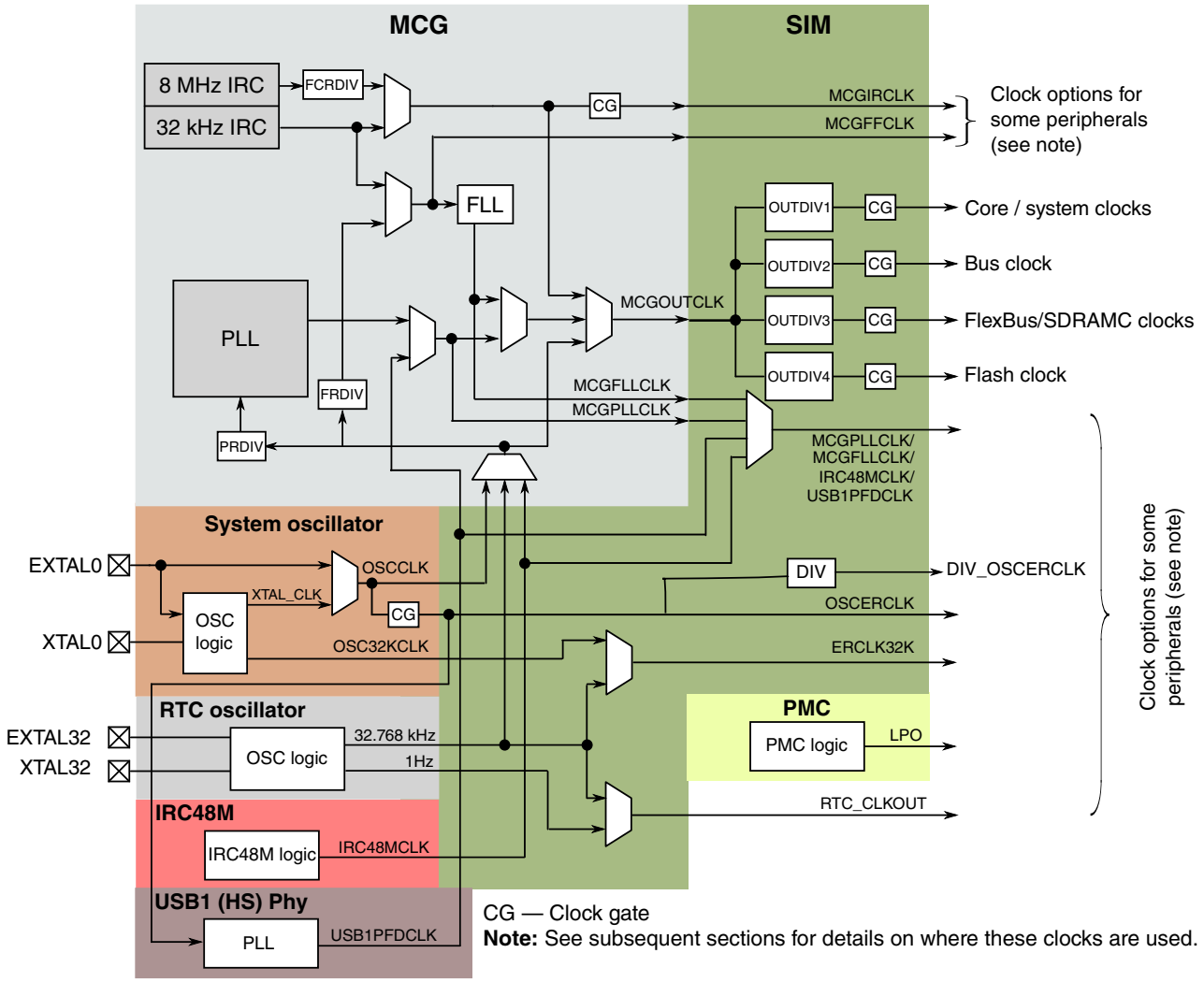


Figure 9. SOC clock diagram Example - Kinetis K66

7.1 Multipurpose Clock Generator (MCG)

The MCG is the main clock module. It works in conjunction with the oscillator modules to fulfill the clock needs of the MCU and peripheral modules. The MCG contains an FLL and up to two PLL clock generators. The FLL is less precise and is used to clock the MCU when there are no critical interface timing needs. The PLL is used to provide an accurate, low jitter clock to the CPU and peripherals when there is a need for this kind of accurate clock. The PLL is only capable of running with an external crystal or clock input source as the reference clock.

The MCG main system clock source can range from 0 to 150 MHz. There is also a separate PLL clock path available for the USB, I2S, and Ethernet modules. There is a separate FLL clock path available for other modules. MCGIRCLK is an internally generated clock that is available for use by the segment LCD, LPT, and TSI modules. MCGFFCLK is available for use by the Flex Timer (FTM).

A high speed IRC, called the IRC48, is available on some Kinetis MCUs. This can clock the MCU and also provides a means of clocking the USB in device mode without the need for an external crystal.

7.2 The MCG lite

The block diagram of MCG_Lite is provide. The first part to see this module was the Kinetis KL03

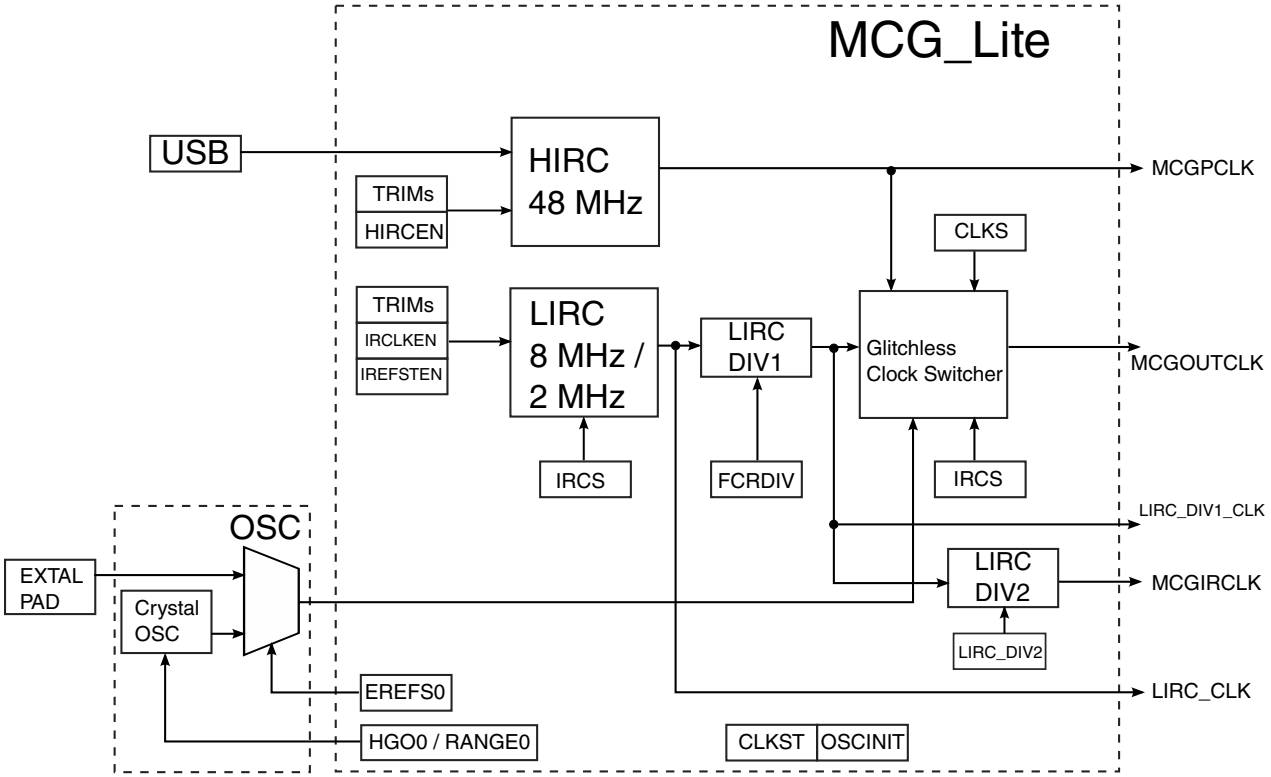


Figure 10. SOC clock diagram MCG LITE - Kinetis KL03

7.3 System Oscillator (OSC)

The OSCERCLK, from the OSC, is available for the FlexCAN, LPT, and ADC. The internal ERCLK32K is available for use by the segment LCD, LPT, and TSI.

7.4 RTC Oscillator (RTC)

The output of the OSC logic is sent to the Real Time Clock (RTC) and can be made available through ERCLK32K to other modules. The RTC oscillator can also be the clock source for the FLL (but not the PLL) and can be routed out to the rest of the system through MCGOUTCLK.

7.5 Power Management Controller (PMC) clocks

The PMC is the source of the low-power oscillator (LPO) that is made available as a clock source to modules. The LPO runs at a nominal frequency of 1 KHz +/– 30%. It is not very accurate over temperature and voltage but it can operate in all power modes. This clock can be the source of clock to a number of modules in the MCU including the LPTMR, the reset filter, the LLWU pin filters, and the LCD controller, so that they in turn can have a clock in the lowest of power modes. See the reference manual configuration sections for each module. The clock options available are detailed in tables.

7.6 Clock control in the SIM

The SIM provides control of the system clock dividers and various clock source multiplexors. The SIM also provides clock gate control of the individual modules in the device. Refer to the SIM and Clock distribution chapters of the individual device reference manual for details.

7.6.1 Clock gate control for peripheral modules in the SIM

There are up to seven clock gate control registers in the SIM. Most of the clock gate enable bits get cleared out of reset with a few exceptions. One is the bit that controls the FTFx—the clock enable for the flash memory.

You need to enable the clock gate to any module before you access it with a read or write, otherwise a hard fault reset will be generated. If you no longer need a module, or if it will be idle for an extended duration, the clock gate bit can be cleared. This will reduce power consumption from that module.

To minimize power consumption of the MCU in general, it is common practice to enable the clock gate only to the modules needed for the operation at hand. In this way, you can tune the power budget to the tasks required at that time.

If a module requires clocks for operation, such as a UART, disabling the clock gate will stop the module operation.

NOTE

The FTFx clock gate bit requires special handling. If entering LLS mode with FTFx cleared, the LLWU wake-up service routine and the interrupt vector must be located in internal RAM for the MCU to wake-up properly. If the flash is required upon exit from the LLWU service routine the FTFx clock gate must be re-enabled before exiting the ISR.

7.7 Clock operation in HSRun, Run and Wait power modes

High Speed Run allows maximum performance of chip. In this state, the MCU is able to operate at a faster frequency compared to normal run mode

The SIM has a number of clock controls. One of the most powerful is the SIMCLKDIV register, which allows clock divide control in Run mode, enabling dynamic frequency scaling to adjust the system power consumption according to the existing performance requirements. The maximum clock speed in Run and Wait modes is determined by the MCU specifications.

7.8 VLPR clock considerations

To enter VLPR from RUN, you have to ensure the VLPR clocking requirements are met. Since clock speeds are limited, so are the serial baud rates and timer timebases.

VLPW can only be entered from VLPR and share the exact same clock limitations. Even though VLPS can be entered directly from RUN mode, any peripherals that are being clocked in VLPS must not exceed the maximum frequencies as specified for VLPR. Refer to the individual device data sheet for device specific information.

The SIMCLKDIV system clock dividers must not be changed in VLPR mode. The desired system clock frequencies must be configured before entering VLPR. You cannot throttle the clock rates as you can in Run mode once you have entered VLPR.

The clock modes that can be used for VLPR and VLPW modes are BLPE, BLPI, or LIRC. BLPE modes uses an external clock source.

On some devices the LPBOOT bits in the FOPT register can be configured to directly enter VLPR mode after exiting from reset. The SIMCLKDIV dividers are configured to ensure the maximum VLPR system clock frequencies are not exceeded.

8 Power Mode Transitions

8.1 Entering low-power modes

Figure 11 shows the allowed power mode transitions. Any reset brings the chip back to the default run state as configured by the LPBOOT bits. In Run, Wait, and Stop modes, active power regulation is enabled. The VLPR and VLPW modes are limited in frequency, but offer a lower power operating mode than normal modes. The LLS and VLLSx modes are the lowest power Stop modes, and should be selected based on the amount of logic or memory that is required to be retained by the application.

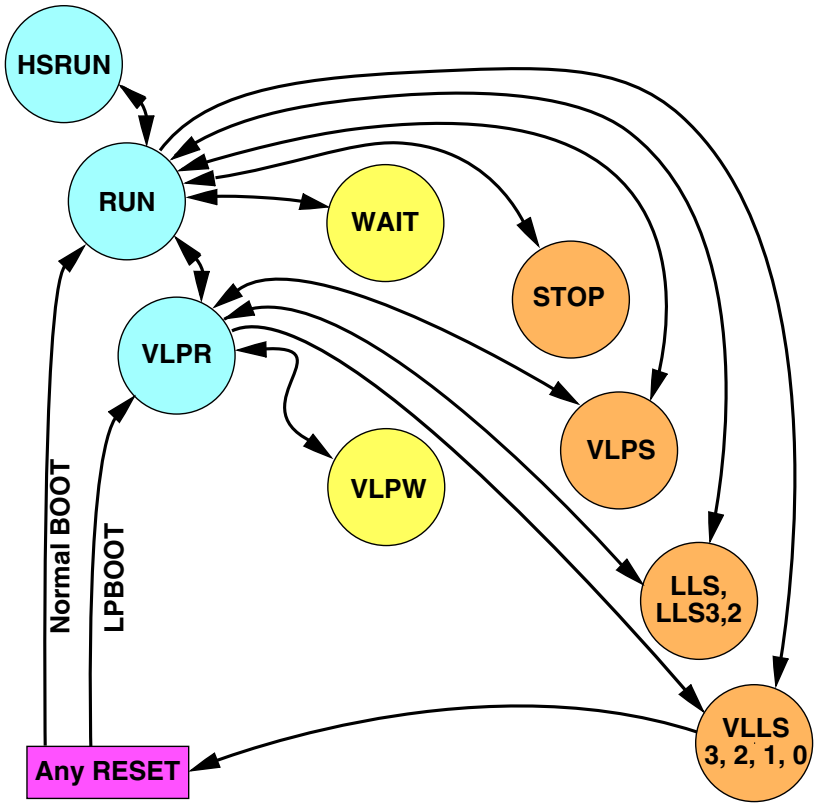


Figure 11. Power mode state transition diagram

8.1.1 Entering and exit conditions for each low power mode

Table 3. Power mode transition triggers

Transition #	From	To	Trigger conditions
1	RUN	WAIT	Sleep-now or sleep-on-exit modes entered with SLEEPDEEP clear, controlled in System Control Register in ARM core. See note.[1]
	WAIT	RUN	Interrupt or Reset
2	RUN	STOP	PMCTRL[RUNM]=00, PMCTRL[STOPM]=000 [2] Sleep-now or sleep-on-exit modes entered with SLEEPDEEP set, which is controlled in System Control Register in ARM core.
	STOP	RUN	Interrupt or Reset
3	RUN	VLPR	The core, system, bus and flash clock frequencies and MCG,MCG_Lite clocking mode are restricted in this mode. See the Power Management chapter for the maximum allowable frequencies and MCG,MCG_Lite modes supported. Set PMPROT[AVLP]=1, PMCTRL[RUNM]=10. **NOTE:** In order to limit peak current, PMPROT[AVLP] and PMCTRL[RUNM] bits can be set on any Reset via Flash IFR settings, causing the SMC to transition the MCU from RUN->VLPR during the reset recovery sequence.
	VLPR	RUN	Set PMCTRL[RUNM]=00 or Interrupt with PMCTRL[LPWUI] =1 or Reset. **NOTE:** Not all devices have the PMCTRL[LPWUI] bit.
4	VLPR	VLPW	Sleep-now or sleep-on-exit modes entered with SLEEPDEEP clear, which is controlled in System Control Register in ARM core.
	VLPW	VLPR	Interrupt with PMCTRL[LPWUI]=0 **NOTE:** Not all devices have the PMCTRL[LPWUI] bit.
5	VLPW	RUN	Interrupt with PMCTRL[LPWUI]=1 or Reset **NOTE:** Not all devices have the PMCTRL[LPWUI] bit. For these devices the cases with LPWUI = 1 do not exist.
6	VLPR	VLPS	PMCTRL[STOPM]=000 [3] or 010, Sleep-now or sleep-on-exit modes entered with SLEEPDEEP set, which is controlled in System Control Register in ARM core.
	VLPS	VLPR	Interrupt with PMCTRL[LPWUI]=0 **NOTE:** If VLPS was entered directly from RUN (transition #7), hardware forces exit back to RUN and does not allow a transition to VLPR. Not all devices have the PMCTRL[LPWUI] bit.

Table continues on the next page...

Table 3. Power mode transition triggers (continued)

Transition #	From	To	Trigger conditions
7	RUN	VLPS	PMPROT[AVLP]=1, PMCTRL[STOPM]=010 Sleep-now or sleep-on-exit modes entered with SLEEPDEEP set, which is controlled in System Control Register in ARM core.
	VLPS	RUN	Interrupt with PMCTRL[LPWUI]=1 or Interrupt with PMCTRL[LPWUI]=0 and VLPS mode was entered directly from RUN or Reset **NOTE:** Not all devices have the PMCTRL[LPWUI] bit.
8	RUN	VLLSx	PMPROT[AVLLS]=1, PMCTRL[STOPM]=100, STOPCTRL[VLLSM]=x (VLLSx), STOPE=1, WAITE=0 PMPROT[AVLLS]=1, PMCTRL[STOPM]=100,STOPCTRL[VLLSM]=x (VLLSx), Sleep-now or sleep-on-exit modes entered with SLEEPDEEP set, which is controlled in System Control Register in ARM core.
	VLLSx	RUN	Wakeup from enabled LLWU input source or RESET pin
9	VLPR	VLLSx	PMPROT[AVLLS]=1, PMCTRL[STOPM]=100, STOPCTRL[VLLSM]=x (VLLSx), STOPE=1, WAITE=0 PMPROT[AVLLS]=1, PMCTRL[STOPM]=100, STOPCTRL[VLLSM]=x (VLLSx), Sleep-now or sleep-on-exit modes entered with SLEEPDEEP set, which is controlled in System Control Register in ARM core.
10	RUN	LLS	PMPROT[ALLS]=1, PMCTRL[STOPM]=011, STOPE=1, WAITE=0 Sleep-now or sleep-on-exit modes entered with SLEEPDEEP set, which is controlled in System Control Register in ARM core.
11	LLS	RUN	Wakeup from enabled LLWU input source and LLS mode was entered directly from RUN or RESET pin.
12	VLPR	LLS	PMPROT[ALLS]=1, PMCTRL[STOPM]=011, STOPE=1, WAITE=0 Sleep-now or sleep-on-exit modes entered with SLEEPDEEP set, which is controlled in System Control Register in ARM core.
	LLS	VLPR	Implemented on some devices. Wakeup from enabled LLWU input source and LLS mode was entered directly from VLPR **NOTE:** If LLS was entered directly from RUN, hardware will not allow this transition and will force exit back to RUN
13	RUN	HSRUN	Set PMPROT[AHSRUN]=1, PMCTRL[RUNM]=11.
	HSRUN	RUN	Set PMCTRL[RUNM]=00 Reset

1. If debug is enabled, the core clock remains to support debug.

2. If PMCTRL[STOPM]=000 and STOPCTRL[PSTOPO]=01 or 10, then only a Partial Stop mode is entered instead of STOP
3. If PMCTRL[STOPM]=000 and STOPCTRL[PSTOPO]=00, then VLPS mode is entered instead of STOP. If PMCTRL[STOPM]=000 and STOPCTRL[PSTOPO]=01 or 10, then only a Partial Stop mode is entered instead of VLPS

9 Power Mode Entry Code

9.1 Power mode transition code

9.1.1 What is included and what is assumed

Example code for entering the different low-power modes is given in the next few sections. The short snippets demonstrate the simplicity of low-power mode entry.

The functions that do the work of entering the mode are included. Any required pre-mode initialization has been omitted, although there are notes indicating what is expected to be done by the time this code is executed.

9.1.2 Function Prototypes for the power mode entry drivers

```
/*****************************************************************/
// function prototypes
void sleep(void);
void deepsleep(void);
void enter_wait(void);
void enter_stop(void);
int enter_vlpr(char lpwui_value);
void exit_vlpr(void);
void enter_vlps(void);
void enter_lls(void);
void enter_lls2(void);
void enter_lls3(void);
void enter_vlls3(void);
void enter_vlls2(void);
void enter_vlls1(void);
void enter_vlls0(unsigned char PORPO_value);
/*****************************************************************/
```

9.1.3 Entering Sleep—create a function to enter Sleep

The ARM Cortex-M4 and M0+ cores uses the state of the SLEEPDEEP bit in the SCR to control which state the core platform enters when the WFI instruction is executed. If the SLEEPDEEP bit is clear, Sleep or Wait mode is entered.

```
/*****************************************************************/
/*
 * Configures the ARM system control register for WAIT(sleep)mode
 * and then executes the WFI instruction to enter the mode.
 *
 * Parameters:
 * none
 *
 */
```

```
void sleep (void)
{
/* Clear the SLEEPDEEP bit to make sure we go into WAIT (sleep)
 * mode instead of deep sleep.
 */
    SCB_SCR &=  ~SCB_SCR_SLEEPDEEP_MASK;
#ifdef CMSIS
    __wfi();
#else
/* WFI instruction will start entry into WAIT mode */
asm("WFI");
#endif
}
/*******************************************************************/
```

9.1.4 Entering Deep Sleep—create a function to enter Deep Sleep

The ARM Cortex-M4 and M0+ cores uses the state of the SLEEPDEEP bit in the SCR to control which state the core platform enters when the WFI instruction is executed. If the SLEEPDEEP bit is set Deep Sleep or core Stop mode is entered.

```
/*******************************************************************/
/*
 * Configures the ARM system control register for STOP
 * (deepsleep) mode and then executes the WFI instruction
 * to enter the mode.
 *
 * Parameters:
 * none
 *
 */

void deepsleep (void)
{
  /* Set the SLEEPDEEP bit to enable deep sleep mode (STOP) */
  SCB_SCR |= SCB_SCR_SLEEPDEEP_MASK;

#ifdef CMSIS
  __wfi();
#else
  /* WFI instruction will start entry into STOP mode */
  asm("WFI");
#endif
}
/*******************************************************************/
```

9.1.5 Entering Wait mode or VLPW

The MCU enters Wait or VLPW mode with the code below. The mode entered depends on the run mode in use at the time. If in run mode you enter Wait, if in VLPR mode you enter VLPW.

If the watchdog is enabled, then you must have a method to wake the device to service the watchdog, else the wake-up will be from a watchdog reset.

Prerequisites:

- The clock monitor(s) in the MCG must be disabled before calling this function if you are going to enter VLPW. For example, CME0 should also be written to a logic 0 before entering VLPR or VLPW power modes if the MCG is in BLPE mode.
- The wake-up events that are to be used to wake the MCU from Wait or VLPW must be set up before calling this function.

```
/*****************************************************************/
/* WAIT mode entry routine. Puts the processor into Wait mode.
 * In this mode the core clock is disabled (no code executing),
 * but bus clocks are enabled (peripheral modules are
 * operational)
 *
 * Mode transitions:
 * RUN to WAIT
 * VLPR to VLPW
 *
 * This function can be used to enter normal wait mode or VLPW
 * mode. If you are executing in normal run mode when calling
 * this function, then you will enter normal wait mode.
 * If you are in VLPR mode when calling this function,
 * then you will enter VLPW mode instead.
 *
 * NOTE: Some modules include a programmable option to disable
 * them in wait mode. If those modules are programmed to disable
 * in wait mode, they will not be able to generate interrupts to
 * wake the core.
 *
 * WAIT mode is exited using any enabled interrupt or RESET,
 * so no exit_wait routine is needed.
 * For Kinetis K:
 * If in VLPW mode, the statue of the SMC_PMCTRL[LPWUI] bit
 * determines if the processor exits to VLPR (LPWUI cleared)
 * or normal run mode (LPWUI set). The enable_lpwui()
 * and disable_lpwui()functions can be used to set this bit
 * to the desired option prior to calling enter_wait().
 * For Kinetis L:
 * LPWUI does not exist.
 * Exits with an interrupt from VLPW will always be back to VLPR.
 * Exits from an interrupt from Wait will always be back to Run.
 *
 * Parameters:
 * none
 */
void enter_wait(void)
{
    sleep();
}
/*****************************************************************/
```

9.1.6 Entering normal Stop mode

The MCU enters normal Stop mode from run with the code below.

For Kinetis L the MCU enters VLPS instead of Stop mode if starting from VLPR mode.

If the watchdog is enabled, then you must have a method to wake the device to service the watchdog, else the wake-up will be from a watchdog Reset.

The MCU will always exit Stop mode into Run mode.

Prerequisites:

- The clock monitor(s) in the MCG must be disabled before calling this function.
- All modules that would not acknowledge a Stop mode entry must be handled. Refer to the reference manual for which modules need to handled.
- To enter Stop mode PMCTRL[RUNM]=00, PMCTRL[STOPM]=02.
- If the PSTOPO bit in the STOPCTRL register='b01 or 'b10, then only a Partial Stop mode is entered instead of STOP
- The wake-up events that are to be used to wake the MCU from Stop or VLPS must be set up before calling this function.

```
/*****************************************************************/
/* STOP mode entry routine.
 * if in Run mode puts the processor into normal stop mode.
 * If in VLPR mode puts the processor into VLPS mode.
 * In this mode core, bus and peripheral clocks are disabled.
 *
 * Mode transitions:
 * RUN to STOP
 * VLPR to VLPS
 *
 * This function can be used to enter normal stop mode.
 * If you are executing in normal run mode when calling this
 * function and AVLP = 0, then you will enter normal stop mode.
 * If AVLP = 1 with previous write to PMPROT
 * then you will enter VLPS mode instead.
 *
 * STOP mode is exited using any enabled interrupt or RESET,
 * so no exit_stop routine is needed.
 *
 * Parameters:
 * none
 */
void enter_stop(void)
{
  volatile unsigned int dummyread;
  /*The PMPROT register may have already been written by init
     code. If so, then this next write is not done since
     PMPROT is write once after RESET
     this write-once bit allows the MCU to enter the
     normal STOP mode.
     If AVLP is already a 1, VLPS mode is entered
     instead of normal STOP
     is SMC_PMPROT = 0  */

    /* Set the STOPM field to 0b000 for normal STOP mode
       For Kinetis L: if trying to enter Stop from VLPR user
       forced to VLPS low power mode */
    SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK;
    SMC_PMCTRL |=  SMC_PMCTRL_STOPM(0);
    /*wait for write to complete to SMC before stopping core */
    dummyread = SMC_PMCTRL;
    deepsleep();
}
/*****************************************************************/
```

9.1.7 Entering VLPR mode

The MCU enters VLPR mode with the code below. In Kinetis K, the value of LPWUI is passed as a parameter to this function. The state of LPWUI is static during VLPR, VLPW, and VLPS modes and should not be modified while in VLPR. For Kinetis L, the value you pass to this routine is a don't care.

Prerequisites:

- The frequency of the clock must be reduced. Some MCUs can operate with the system, bus, and FlexBus clocks at 4 MHz and the flash clock at 1 MHz.

```
//core = /1 bus = /2 flexbus = /2 flash clk =/4
SIM_CLKDIV1  =  (SIM_CLKDIV1_OUTDIV1(1) |
                 SIM_CLKDIV1_OUTDIV2(1) |
                 SIM_CLKDIV1_OUTDIV3(1) |
                 SIM_CLKDIV1_OUTDIV4(3));
```

- The clock monitor(s) in the MCG must be disabled before calling this function. For example, CME0 should also be written to a logic 0 before entering VLPR or VLPW power modes if the MCG is in BLPE mode.
- All modes that would not acknowledge a Stop mode entry must be disabled.

```
/*****************************************************************/
/* VLPR mode entry routine.Puts the processor into Very Low Power
 * Run Mode. In this mode, all clocks are enabled,
 * but the core, bus, and peripheral clocks are limited
 * to 2 or 4 MHz or less.
 * The flash clock is limited to 1MHz or less.
 *
 * Mode transitions:
 * RUN to VLPR
 *
 * For Kinetis K:
 * While in VLPR, VLPW or VLPS the exit to VLPR is determined by
 * the value passed in from the calling program.
 * LPWUI is static during VLPR mode and
 * should not be written to while in VLPR mode.
 *
 * For Kinetis L:
 * LPWUI does not exist. the parameter pass is a don't care
 * Exits with an interrupt from VLPW will always be back to VLPR.
 * Exits from an interrupt from Wait will always be back to Run.
*
 * Parameters:
 * lpwui_value - The input determines what is written to the
 *               LPWUI bit in the PMCTRL register
 *               Clear LPWUI and interrupts keep you in VLPR
 *               Set LPWUI and interrupts return you to Run mode
 * Return value : PMSTAT value or error code
 *                PMSTAT = 000_0100 Current power mode is VLPR
 *                ERROR Code =  0x14 - already in VLPR mode
 *                           =  0x24 - REGONS never clears
 *                                     indicating stop regulation
 */
int enter_vlpr(char lpwui_value)
{
  int i;
  if ((SMC_PMSTAT & SMC_PMSTAT_PMSTAT_MASK)== 4){
         return 0x14;
         }

  /* The PMPROT register may have been written by init code
     If so, then this next write is not done
     PMPROT is write once after RESET
     This write-once bit allows the MCU to enter the
     very low power modes: VLPR, VLPW, and VLPS   */
  SMC_PMPROT = SMC_PMPROT_AVLP_MASK;

  /* Set the (for MC1)LPLLSM or (for MC2)STOPM field
     to 0b010 for VLPS mode -
     and RUNM bits to 0b010 for VLPR mode
     Need to set state of LPWUI bit */
  lpwui_value &= 1;
  SMC_PMCTRL &= ~SMC_PMCTRL_RUNM_MASK;
  SMC_PMCTRL |= SMC_PMCTRL_RUNM(0x2) |
                (lpwui_value << SMC_PMCTRL_LPWUI_SHIFT);
  /* OPTIONAL Wait for VLPS regulator mode to be confirmed */
  for (i = 0 ; i < 10 ; i++)
    {     /* Check that the value of REGONS bit is not 0
             When it is a zero, you can stop checking */
      if ((PMC_REGSC & PMC_REGSC_REGONS_MASK) ==0x04){
       /* 0 Regulator is in stop regulation or in transition
             to/from it
          1 MCU is in Run regulation mode */
      }
      else  break;
    }
  if ((PMC_REGSC & PMC_REGSC_REGONS_MASK) ==0x04) {
       return 0x24;
     }
  /* SMC_PMSTAT register only exist in Mode Controller 2 */
```

```
  if ((SMC_PMSTAT & SMC_PMSTAT_PMSTAT_MASK) == 4) {
       return 0x04;
     }
}
/*****************************************************************/

/*  Use this statement in-line with your code to put
 *  the MCU into VLPR
 *  This assumes
 *  1)that the clocks are setup properly
 *  2)that you are entering VLPR from Run mode */

/*  add the next line if not already written. */
//  SMC_PMPROT = SMC_PMPROT_AVLP_MASK;

  SMC_PMCTRL = SMC_PMCTRL_RUNM(0x2);
```

9.1.8 Entering VLPS mode

The MCU enters VLPS mode from Run or VLPR modes with the code below.

If the watchdog is enabled, then the you must have a method to wake the part to service the watchdog, else the wake-up will be from a watchdog reset.

Prerequisites:

- The clock monitor(s) in the MCG must be disabled before calling this function. For example, the CME0 bit should be written to a logic 0 before the MCG enters any Stop mode. Otherwise, a reset request may occur while in Stop mode.
- To enter VLPS mode, the ALVP bit in the PMPROT register must be set and the STOPM bit in the PMCTRL register must be 'b10.

```
/*****************************************************************/
/* VLPS mode entry routine. Puts the processor into VLPS mode
 * directly from run or VLPR modes.
 *
 * Mode transitions:
 * RUN  to VLPS
 * VLPR to VLPS
 *
 * Kinetis K:
 * when VLPS is entered directly from RUN mode,
 * exit to VLPR is disabled by hardware and the system will
 * always exit back to RUN.
 *
 * If however VLPS mode is entered from VLPR the state of
 * the LPWUI bit determines the state the MCU will return
 * to upon exit from VLPS.If LPWUI is 1 and an interrupt
 * occurs you will exit to normal run mode instead of VLPR.
 * If LPWUI is 0 and an interrupt occurs you will exit to VLPR.
 *
 * For Kinetis L:
 * when VLPS is entered from run an interrupt will exit to run.
 * When VLPS is entered from VLPR an interrupt will exit to VLPS
 * Parameters:
 * none
 */
/*****************************************************************/

void enter_vlps(void)
{
  volatile unsigned int dummyread;
  /*The PMPROT register may have already been written by init
     code. If so then this next write is not done since
     PMPROT is write once after RESET.
```

```
      This Write allows the MCU to enter the VLPR, VLPW,
      and VLPS modes. If AVLP is already written to 0
      Stop is entered instead of VLPS*/
  SMC_PMPROT = SMC_PMPROT_AVLP_MASK;
  /* Set the STOPM field to 0b010 for VLPS mode */
  SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK;
  SMC_PMCTRL |=  SMC_PMCTRL_STOPM(0x2);
  /*wait for write to complete to SMC before stopping core */
  dummyread = SMC_PMCTRL;
  /* Now execute the stop instruction to go into VLPS */
  deepsleep();
}
/*****************************************************************/
```

9.1.9 Entering LLS mode

The MCU enters LLS mode from Run or VLPR with the following code.

Prerequisites:

- All modes that would not acknowledge a Stop mode entry must be disabled.
- The clock monitor(s) in the MCG must be disabled before calling this function.
- To enter LLS mode, the ALLS bit in the PMPROT register must be set.
- If the ALLS and AVLP bits in the PMPROT register = 0 then a write to bits in the PMCTRL is ignored.
- All I/O is held at state while the MCU is in LLS mode and is released automatically upon wake-up.
- The wake-up events that are to be used to wake the MCU from LLS must be set up in the LLWU and wake-up module before calling this function. See LLS mode entry and exit - bare metal code example section of this application note for an example of setting up the LLWU to wake with the PTE1 pin.

```
/*****************************************************************/
/* LLS mode entry routine. Puts the processor into LLS mode from
 * normal Run mode or VLPR.
 *
 * Mode transitions:
 * RUN to LLS
 * VLPR to LLS
 *
 * Wake-up from LLS mode is controlled by the LLWU module. Most
 * modules cannot issue a wake-up interrupt in LLS mode, so make
 * sure to set up the desired wake-up sources in the LLWU before
 * calling this function.
 *
 * Parameters:
 * none
 */
void enter_lls(void)
{
   volatile unsigned int dummyread;
   /* Write to PMPROT to allow LLS power modes this write-once
      bit allows the MCU to enter the LLS low power mode*/
   SMC_PMPROT = SMC_PMPROT_ALLS_MASK;
   /* Set the (for MC1) LPLLSM or
     (for MC2)STOPM field to 0b011 for LLS mode
      Retains LPWUI and RUNM values */
   SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK ;
   SMC_PMCTRL |= SMC_PMCTRL_STOPM(0x3) ;
    /* set LLSM = 0b00 in SMC_VLLSCTRL (for MC4) */
     SMC_VLLSCTRL =  SMC_VLLSCTRL_LLSM(0);
   /*wait for write to complete to SMC before stopping core */
   dummyread = SMC_PMCTRL;
   /* Now execute the stop instruction to go into LLS */
   deepsleep();
}
/*****************************************************************/
```

9.1.10 Entering LLS2 mode

The MCU enters LLS2 mode from Run or VLPR with the following code.

The mode that the MCU will exit from LLS2 is always Run mode.

Prerequisites:

- All modes that would not acknowledge a Stop mode entry must be disabled.
- The clock monitor(s) in the MCG must be disabled before calling this function.
- To enter LLS2 mode, the ALLS bit in the PMPROT register must be set.
- If the ALLS and AVLP bits in the PMPROT register = 0 then a write to bits in the PMCTRL is ignored.
- All I/O is held at state while the MCU is in LLS mode and is released automatically upon wake-up.
- The wake-up events that are to be used to wake the MCU from LLS must be set up in the LLWU and wake-up module before calling this function. See LLS mode entry and exit - bare metal code example section of this application note for an example of setting up the LLWU to wake with the PTE1 pin.

```
/*****************************************************************/
/* LLS mode entry routine. Puts the processor into LLS mode from
 * normal Run mode or VLPR.
 *
 * Mode transitions:
 * RUN to LLS2
 * VLPR to LLS2
 *
 * NOTE: LLS2 mode will always exit to Run mode even if you were
 * in VLPR mode before entering LLS2.
 *
 * Wake-up from LLS2 mode is controlled by the LLWU module. Most
 * modules cannot issue a wake-up interrupt in LLS mode, so make
 * sure to set up the desired wake-up sources in the LLWU before
 * calling this function.
 *
 * Parameters:
 * none
 */
void enter_lls2(void)
{
   volatile unsigned int dummyread;
   /* Write to PMPROT to allow LLS power modes this write-once
      bit allows the MCU to enter the LLS low power mode*/
   SMC_PMPROT = SMC_PMPROT_ALLS_MASK;
   /* Set the (for MC1) LPLLSM or
     (for MC2)STOPM field to 0b011 for LLS3 mode
      Retains LPWUI and RUNM values */
   SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK ;
   SMC_PMCTRL |= SMC_PMCTRL_STOPM(0x3) ;
    /* set LLSM = 0b10 in SMC_VLLSCTRL (for MC4) */
     SMC_VLLSCTRL =  SMC_VLLSCTRL_LLSM(2);

   /*wait for write to complete to SMC before stopping core */
   dummyread = SMC_PMCTRL;
   /* Now execute the stop instruction to go into LLS */
   deepsleep();
}
/*****************************************************************/
```

9.1.11 Entering LLS3 mode

The MCU enters LLS3 mode from Run or VLPR with the following code.

Prerequisites:

- All modes that would not acknowledge a Stop mode entry must be disabled.
- The clock monitor(s) in the MCG must be disabled before calling this function.
- To enter LLS3 mode, the ALLS bit in the PMPROT register must be set.
- If the ALLS and AVLP bits in the PMPROT register = 0 then a write to bits in the PMCTRL is ignored.
- All I/O is held at state while the MCU is in LLS3 mode and is released automatically upon wake-up.
- The wake-up events that are to be used to wake the MCU from LLS3 must be set up in the LLWU and wake-up module before calling this function. See LLS mode entry and exit - bare metal code example section of this application note for an example of setting up the LLWU to wake with the PTE1 pin.

```
/*****************************************************************/
/* LLS3 mode entry routine. Puts the processor into LLS3 mode from
 * normal Run mode or VLPR.
 *
 * Mode transitions:
 * RUN to LLS3
 * VLPR to LLS3
 *
 * Wake-up from LLS3 mode is controlled by the LLWU module. Most
 * modules cannot issue a wake-up interrupt in LLS3 mode, so make
 * sure to set up the desired wake-up sources in the LLWU before
 * calling this function.
 *
 * Parameters:
 * none
 */
void enter_lls3(void)
{
   volatile unsigned int dummyread;
   /* Write to PMPROT to allow LLS power modes this write-once
      bit allows the MCU to enter the LLS low power mode*/
   SMC_PMPROT = SMC_PMPROT_ALLS_MASK;
   /* Set the (for MC1) LPLLSM or
     (for MC2)STOPM field to 0b011 for LLS3 mode
      Retains LPWUI and RUNM values */
   SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK ;
   SMC_PMCTRL |= SMC_PMCTRL_STOPM(0x3) ;
    /* set LLSM = 0b11 in SMC_VLLSCTRL (for MC4) */
     SMC_VLLSCTRL =  SMC_VLLSCTRL_LLSM(3);
   /*wait for write to complete to SMC before stopping core */
   dummyread = SMC_PMCTRL;
   /* Now execute the stop instruction to go into LLS */
   deepsleep();
}
/*****************************************************************/
```

9.1.12 Entering VLLS3 mode

The MCU enters VLLS3 mode from Run or VLPR with the code below.

The mode that the MCU will exit from VLLS3 is always a reset.

Prerequisites:

- To enter VLLS3 mode, the AVLLS bit in the PMPROT register must be set.
- If the AVLLS and AVLP bits in the PMPROT register = 0 then normal Stop is entered instead of VLLS3.
- The clock monitor(s) in the MCG must be disabled before calling this function.

- All I/O is held at state while the MCU is in VLLS3 mode and is released after reset when the code writes to the ACKISO bit.
- The wake-up events that are to be used to wake the MCU from VLLS3 must be set up in the LLWU and wake-up module before calling this function. See LLS mode entry and exit - bare metal code example section of this application note for an example of setting up the LLWU to wake with the PTE1 pin.

```
/*****************************************************************/
/* VLLS3 mode entry routine. Puts the processor into
 * VLLS3 mode from normal Run mode or VLPR.
 *
 * Mode transitions:
 * RUN to VLLS3
 * VLPR to VLLS3
 *
 * NOTE: VLLSx modes will always exit to Run mode even if you were
 *       in VLPR mode before entering VLLSx.
 *
 * Wake-up from VLLSx mode is controlled by the LLWU module. Most
 * modules cannot issue a wake-up interrupt in VLLSx mode, so make
 * sure to setup the desired wake-up sources in the LLWU before
 * calling this function.
 *
 * Parameters:
 * none
 */
void enter_vlls3(void)
{
   volatile unsigned int dummyread;
    /* Write to PMPROT to allow VLLS3 power modes */
    SMC_PMPROT = SMC_PMPROT_AVLLS_MASK;

    /* Set the VLLSM field to 0b100 for VLLSx(for MC1)
      or STOPM field to 0b100 for VLLSx (for MC2)
    - Retain state of LPWUI and RUNM */
     SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK ;
     SMC_PMCTRL |= SMC_PMCTRL_STOPM(0x4) ;
 //    MC_PMCTRL &= ~MC_PMCTRL_VLLSM_MASK ; //(for MC1)
 //    MC_PMCTRL |= MC_PMCTRL_VLLSM(0x4) ; //(for MC1)
    /* set VLLSM = 0b11 in SMC_VLLSCTRL (for MC2) */
     SMC_VLLSCTRL =  SMC_VLLSCTRL_VLLSM(3);
    /*wait for write to complete to SMC before stopping core */
    dummyread = SMC_CLLSCTRL;
    /* Now execute the stop instruction to go into VLLS3 */
    deepsleep();
}
/*****************************************************************/
```

9.1.13 Entering VLLS2 mode

The MCU enters VLLS2 mode from Run or VLPR with the code below.

The mode that the MCU will exit from VLLS is always a reset.

For Kinetis L, MCU enters VLLS1 mode; VLLS2 mode is not supported on Kinetis L.

Prerequisites:

- To enter VLLS2 mode, the AVLLS bit in the PMPROT register must be set.
- If the AVLLS and AVLP bits in the PMPROT register = 0 then normal Stop is entered instead of VLLS2.
- The clock monitor(s) in the MCG must be disabled before calling this function.

- All I/O is held at state while the MCU is in VLLS2 mode and is released after reset when the code writes to the ACKISO bit.
- The wake-up events that are to be used to wake the MCU from VLLS2 must be set up in the LLWU and wake-up module before calling this function. See LLS mode entry and exit - bare metal code example section of this application note for an example of setting up the LLWU to wake with the PTE1 pin.

```
/*****************************************************************/
/* VLLS2 mode entry routine. Puts the processor into
 * VLLS2 mode from normal Run mode or VLPR.
 *
 * Mode transitions:
 * RUN to VLLS2
 * VLPR to VLLS2
 *
 * NOTE: VLLSx modes will always exit to Run mode even
 *       if you were in VLPR mode before entering VLLSx.
 *
 * Wake-up from VLLSx mode is controlled by the LLWU module. Most
 * modules cannot issue a wake-up interrupt in VLLSx mode, so make
 * sure to setup the desired wake-up sources in the LLWU before
 * calling this function.
 *
 * Parameters:
 * none
 */
void enter_vlls2(void)
{
    volatile unsigned int dummyread;
     /* Write to PMPROT to allow VLLS2 power modes */
    SMC_PMPROT = SMC_PMPROT_AVLLS_MASK;

    /* Set the VLLSM field to 0b100 for VLLSx(for MC1)
      or STOPM field to 0b100 for VLLSx (for MC2)
    - Retain state of LPWUI and RUNM */
     SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK ;
     SMC_PMCTRL |= SMC_PMCTRL_STOPM(0x4) ;
 //    MC_PMCTRL &= ~MC_PMCTRL_VLLSM_MASK ;
 //    MC_PMCTRL |= MC_PMCTRL_VLLSM(0x4) ;
   /* set VLLSM = 0b10 in SMC_VLLSCTRL (for MC2) */
   SMC_VLLSCTRL =  SMC_VLLSCTRL_VLLSM(2);
   /*wait for write to complete to SMC before stopping core */
   dummyread = SMC_VLLSCTRL;
   /* Now execute the stop instruction to go into VLLS2 */
   deepsleep();
}
/*****************************************************************/
```

9.1.14 Entering VLLS1 mode

The MCU enters VLLS1 mode from Run or VLPR with the code below.

The mode that the MCU will exit from VLLS1 is always a reset.

Prerequisites:

- To enter VLLS1 mode, the AVLLS bit in the PMPROT register must be set.
- If the AVLLS and AVLP bits in the PMPROT register = 0 then normal Stop is entered instead of VLLS1.
- The clock monitor(s) in the MCG must be disabled before calling this function.
- All I/O is held at state while the MCU is in VLLS1 mode and is released after reset when the code writes to the ACKISO bit.
- The wake-up events that are to be used to wake the MCU from VLLS1 must be set up in the LLWU and wake-up module before calling this function. See LLS mode entry and exit - bare metal code example section of this application note for an example of setting up the LLWU to wake with the PTE1 pin.

```
/*****************************************************************/
/* VLLS1 mode entry routine. Puts the processor into
 * VLLS1 mode from normal Run mode or VLPR.
 *
 * Mode transitions:
 * RUN to VLLS1
 * VLPR to VLLS1
 *
 * NOTE:VLLSx modes will always exit to Run mode even if you were
 * in VLPR mode before entering VLLSx.
 *
 * Wake-up from VLLSx mode is controlled by the LLWU module. Most
 * modules cannot issue a wake-up interrupt in VLLSx mode, so make
 * sure to setup the desired wake-up sources in the LLWU before
 * calling this function.
 *
 * Parameters:
 * none
 */
void enter_vlls1(void)
{
   volatile unsigned int dummyread;
   /* Write to PMPROT to allow all possible power modes */
   SMC_PMPROT = SMC_PMPROT_AVLLS_MASK;
   /* Set the VLLSM field to 0b100 for VLLSx(for MC1)
     or STOPM field to 0b100 for VLLSx (for MC2)
   - Retain state of LPWUI and RUNM */
   SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK ;
   SMC_PMCTRL |= SMC_PMCTRL_STOPM(0x4) ;
 //    MC_PMCTRL &= ~MC_PMCTRL_VLLSM_MASK ;
 //    MC_PMCTRL |= MC_PMCTRL_VLLSM(0x4) ;
   /* set VLLSM = 0b01 in SMC_VLLSCTRL (for MC2) */
   SMC_VLLSCTRL =  SMC_VLLSCTRL_VLLSM(1);
   /*wait for write to complete to SMC before stopping core */
   dummyread = SMC_VLLSCTRL;
   /* Now execute the stop instruction to go into VLLS1 */
   deepsleep();
}
/*****************************************************************/
```

9.1.15 Entering VLLS0 mode

The MCU enters VLLS0 mode from Run or VLPR with the code below.

The mode that the MCU will exit from VLLS0 is always a reset.

Prerequisites:

- To enter VLLS0 mode, the AVLLS bit in the PMPROT register must be set.
- If the AVLLS and AVLP bits in the PMPROT register = 0 then normal Stop is entered instead of VLLS1.
- The clock monitor(s) in the MCG must be disabled before calling this function.
- All I/O is held at state while the MCU is in VLLS0 mode and is released after reset when the code writes to the ACKISO bit.
- The wake-up events that are to be used to wake the MCU from VLLS0 must be set up in the LLWU and wake-up module before calling this function. See LLS mode entry and exit - bare metal code example section of this application note for an example of setting up the LLWU to wake the PTE1 pin.

```
/*****************************************************************/
/* VLLS0 mode entry routine. Puts the processor into
 * VLLS0 mode from normal run mode or VLPR.
 *
 * Mode transitions:
 * RUN to VLLS0
```

```
 * VLPR to VLLS0
 *
 * NOTE: VLLSx modes will always exit to RUN mode even if you were
 * in VLPR mode before entering VLLSx.
 *
 * Wake-up from VLLSx mode is controlled by the LLWU module. Most
 * modules cannot issue a wake-up interrupt in VLLSx mode, so make
 * sure to setup the desired wake-up sources in the LLWU before
 * calling this function.
 *
 * Parameters:
 * PORPO_value - 0 POR detect circuit is enabled in VLLS0
 *               1 POR detect circuit is disabled in VLLS0
 */
 /*****************************************************************/

void enter_vlls0(unsigned char PORPO_value )
{
    volatile unsigned int dummyread;
    /* Write to PMPROT to allow all possible power modes */
    SMC_PMPROT = SMC_PMPROT_AVLLS_MASK;
    /* Set the STOPM field to 0b100 for VLLS0 mode */
    SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK;
    SMC_PMCTRL |=  SMC_PMCTRL_STOPM(0x4);
    /* set VLLSM = 0b00 */
    SMC_VLLSCTRL =  (PORPO_value <<SMC_VLLSCTRL_PORPO_SHIFT)
                  | SMC_VLLSCTRL_VLLSM(3);
    /*wait for write to complete to SMC before stopping core */
    dummyread = SMC_VLLSCTRL;
    stop();
}
```

10 Power Mode Exit Transitions

10.1 More Notes about exiting low-power modes

The normal exit procedure is to enable and detect an event that initiates a mode change.

For Wait, Stop, VLPS, VLPR, and VLPW the wake-up event is typically an interrupt from a pin state change or module. The LLWU is not enabled and not used for these mode exits.

For LLS,LLS2,LLS3, VLLS3, VLLS2, VLLS1, and VLLS0 the wake-up events are limited to the enabled LLWU wake-up pins or modules, and the reset and NMI pins. The pin can be a wake-up source as long as the pin is "enabled as a digital input source" in the Pin Control Register. For MCU's without a LLWU, the module or pin wakeup is a wake-up source if included in the wake-up source table in the MCU reference manual and are set up as interrupt wake-up sources.

10.1.1 Pin Interrupt and LLWU Wake-up functionality integration

For LLWU wake-up input pins, it is recommended to also enable the pin interrupt functionality on the pin. In the event that the pin interrupt is not enabled, there is a window of time while the MCU transitions into the low leakage mode, that an edge transition might be missed. If the pin interrupt functionality was enabled and an edge occurred during this small transition time the low power entry would be aborted and the MCU would service the pin interrupt.

In Kinetis K devices all LLWU wake-up pins are also capable of being pin interrupt sources. In Kinetis L devices some LLWU wake-up pins do not have this dual functionality.

If the pin can also function as a pin interrupt to wake the MCU it is recommended to use both the interrupt and LLWU wakeup features. This is recommended since there is a very tiny window of time (~4ns) as the MCU transitions into the low leakage power mode, like LLSx or VLLSx, that an edge transition could be missed if the interrupt was not enabled for that pin. If the edge occurred during this small transition time the low power entry would be averted and the MCU would take the pin interrupt.

If the pin is configured as both a pin interrupt and an LLWU wake-up pin, the corresponding ISF flag bit in the port control register may be set and can be cleared in the LLWU. If the ISF flag is set and is not cleared in the LLWU ISR then the port ISR will be taken after the LLWU isr completes.

10.1.2 Reset as LLWU Wake-up

For all modes, the wake-up event could be a reset caused by any of the reset sources. These reset sources are listed in the reference manuals of the Kinetis MCU.

All references to the LPWUI bit are typically for Kinetis K MCUs. The Kinetis L series does not have the LPWUI bit.

10.1.3 Power Mode Transition Times

The interrupt latency of the M4 core based Kinetis devices is 12 cycles. The interrupt latency of the M0+ core based Kinetis devices is 15 cycles.

Table 4. Power Mode Transition Times

Transition number	Transition	Transition time
1	WAIT - RUN	Interrupt Latency
2	STOP - RUN	2us + Interrupt Latency
3	VLPW - VLPR	Interrupt Latency
4	VLPW - RUN	2 - 4 us + Interrupt Latency[1]
5	VLPS - VLPR	2 - 4 us + Interrupt Latency
6	VLPS - RUN	2 - 4 us + Interrupt Latency [2]
7	VLLSx - RUN	BOOT(LP) + 53 us to 115 us[3]
8	LLS - RUN	2 us + Interrupt Latency

1. The VLPW to RUN transition is only possible with Kinetis devices with LPWUI bit
2. The VLPS to RUN transition is only possible with Kinetis devices with LPWUI bit or Kinetis K22F MCUs
3. Exit from VLLSx modes is through the Wakeup Reset Flow

10.1.4 What can slow the exit from low power modes

- the MCG clock mode is not using the FLL. Using the MCG clock mode PEE causes the wakeup to start only after the external clock sources has started. This clock source is typically much slower than the PEE clock frequency.
- pin filtering is being used on the input trigger.
- The rise time of the input trigger is slow. (rising edge detected at VIH, falling edge at VIL).
- The MCU clock is slow or stopped.
- The wakeup source is the NMI pin and it stays low for a while. NMI is level sensitive and will stay in NMI ISR until NMI input returns high.

- The debugger is active during mode transitions. Interaction of the debug module can halt or stall processor.
- The external clock source is a crystal and it stops during low power mode. The crystal startup time is added to mode recovery time.

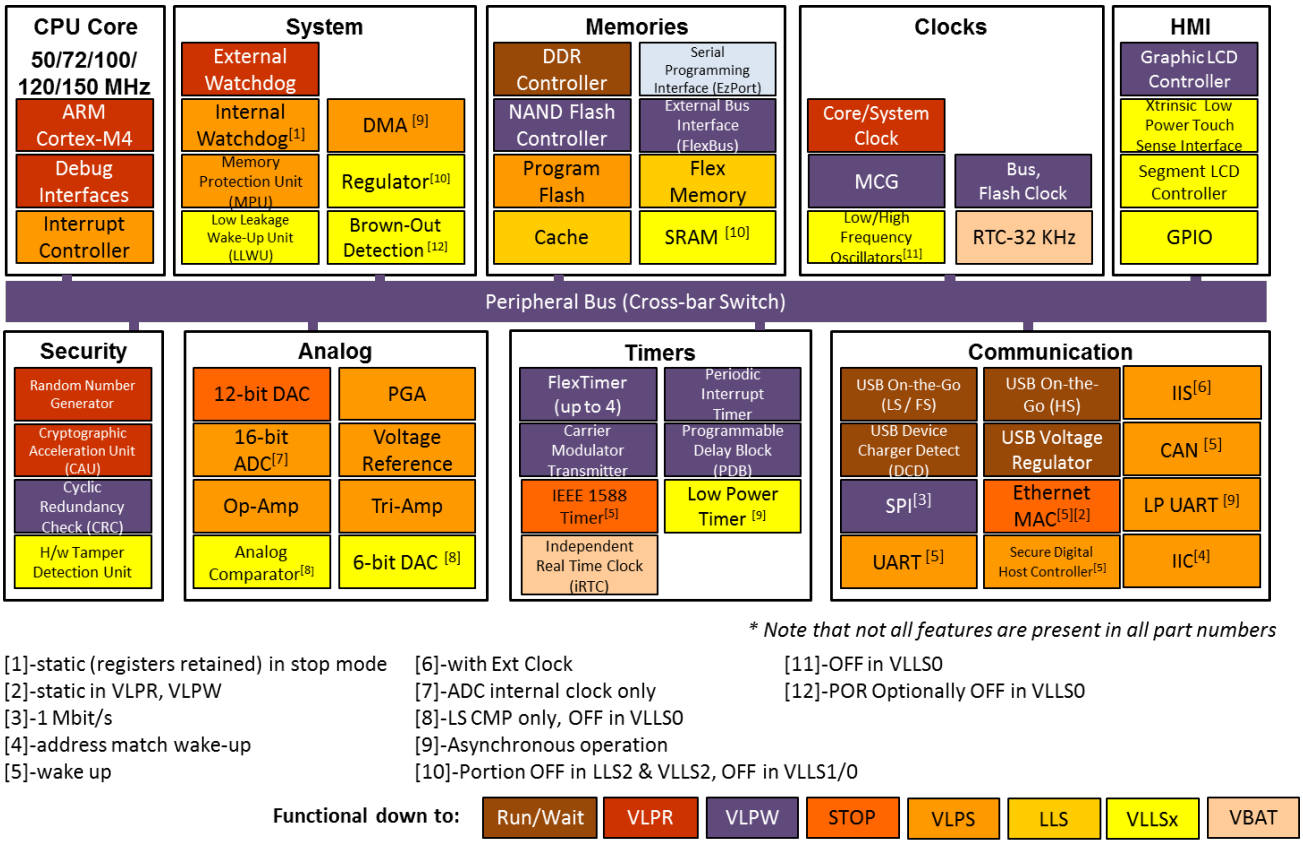
10.1.5 What can cause the MCU not to recognize an interrupt or LLWU input.

- The MCU is in RUN, WAIT, VLPR or VLPW and the input pulse is less than 1 and 1/2 bus clocks long. (i.e. A 250 ns input pulse with a 1 MHz bus won't work)
- pin filtering is being used on the input trigger and the input is not long enough. If LPO is used to clock the filter the pulse is less than 3 milliseconds
- The rise time of the input trigger is slow. (rising edge detected at VIH, falling edge at VIL).
- The MCU clock is slow or stopped.
- If you are in WAIT, VLPW, STOP, VLPS and global interrups are disabled while RESET and NMI are not used to wake up the MCU.
- If a pin is not a digital input, it is disabled or is an output the LLWU input function will not wake the MCU.

11 Modules in Power Modes

11.1 Module operation in low-power modes

The chart below illustrates the Kinetis K series module operation in the various low power modes.

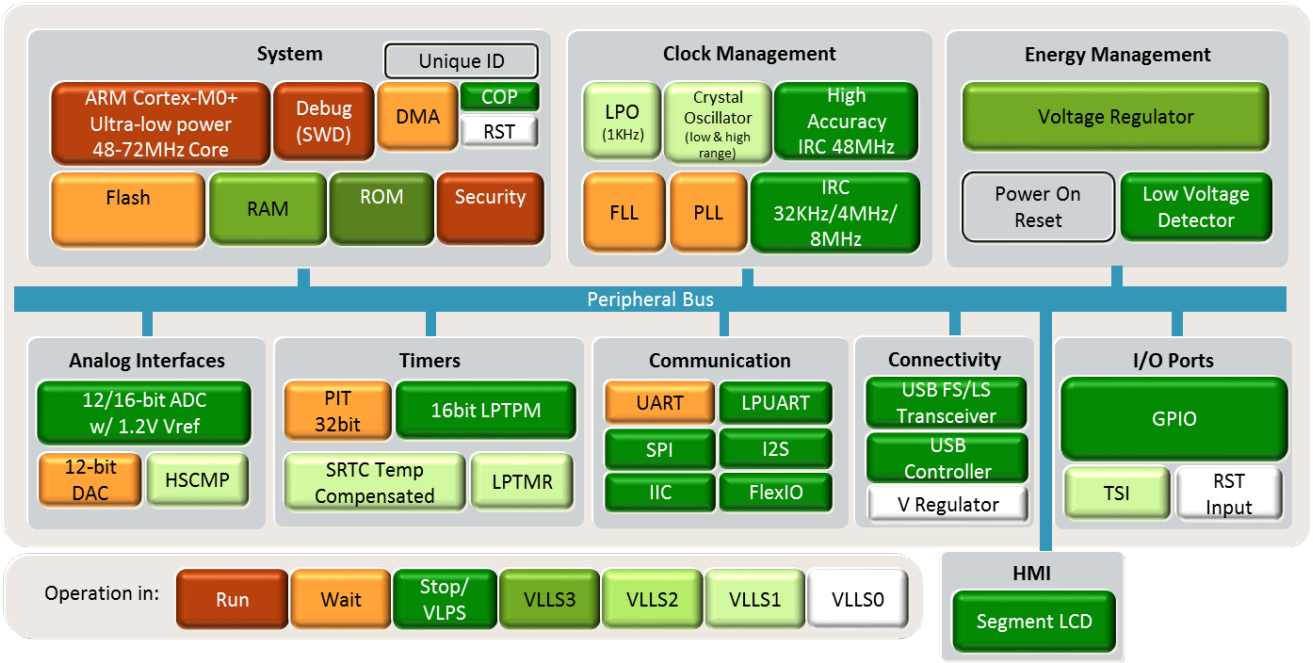


** Note that not all features are present in all part numbers*

[1]-static (registers retained) in stop mode
[2]-static in VLPR, VLPW
[3]-1 Mbit/s
[4]-address match wake-up
[5]-wake up
[6]-with Ext Clock
[7]-ADC internal clock only
[8]-LS CMP only, OFF in VLLS0
[9]-Asynchronous operation
[10]-Portion OFF in LLS2 & VLLS2, OFF in VLLS1/0
[11]-OFF in VLLS0
[12]-POR Optionally OFF in VLLS0

Functional down to: Run/Wait | VLPR | VLPW | STOP | VLPS | LLS | VLLSx | VBAT

Figure 12. Kinetis K-Modules Power Modes Chart



- FF—full functionality. In VLPR and VLPW, the system frequency is limited, but if a module does not have a limitation in its functionality, it is still listed as FF.
- Async operation = Kinetis L and later Kinetis K
- Static—module register states and associated memories are retained.

- Powered—memory is powered to retain contents.
- Low power—flash has a low-power state that retains configuration registers to support faster wake-up.
- OFF—modules are powered off; module is in reset state upon wake-up.
- Wake-up—modules can serve as a wake-up source for the chip.
- CPO - Compute Only Mode - Kinetis L and later Kinetis K
- PSTOPx - Partial Stop Mode 1 or 2 - Kinetis L and later Kinetis K
- IOPORT - Single Cycle IOPORT - Kinetis L and later Kinetis K

Table 5. Module operation in low-power modes

Modules	Stop	VLPR	VLPW	VLPS	LLSx	VLLSx
Core Platform modules						
CORE	static	FF	static	static	static	OFF
NVIC	static	FF	FF	static	static	OFF
FPU	static	FF	static	static	static	OFF
HDQR	static	FF	static	static	static	OFF
System modules						
Mode Controller	FF	FF	FF	FF	FF	FF
LLWU[1]	static	static	static	static	FF	FF[2]
Regulator	ON	low power	low power	low power	low power	low power in VLLS2/3, OFF in VLLS0/1
LVD	ON	disabled	disabled	disabled	disabled	disabled
Brown-out Detection	ON	ON	ON	ON	ON	ON in VLLS1/2/3, optionally disabled in VLLS0[3]
DMA	Async operation	FF Async operation in CPO	FF	Async operation	static	OFF
Watchdog	static FF in PSTOP2	FF static in CPO	FF	FF	static	OFF
EWM	static FF in PSTOP2	FF static in CPO	static	static	static	OFF
Clocks						
1kHz LPO	ON	ON	ON	ON	ON	ON in VLLS1/2/3, OFF in VLLS0
System oscillator (OSC)	OSCERCLK optional	OSCERCLK max of 16 MHz crystal	OSCERCLK max of 16 MHz crystal	OSCERCLK max of 16 MHz crystal	limited to low range/low power	limited to low range/low power in VLLS1/2/3, OFF in VLLS0
MCG PLL	PLL optionally on but gated	static	static	static	static	OFF
MCG FLL	static—MCGIRCLK optional	static	static	static	static—no clock output	OFF

Table continues on the next page...

Table 5. Module operation in low-power modes (continued)

Modules	Stop	VLPR	VLPW	VLPS	LLSx	VLLSx
MCG Slow IRC	static - MCGIRCLK optional	static	static	static - MCGIRCLK optional	static—no clock output	OFF
MCG Fast IRC	MCGIRCLK optional	2 or 4 MHz IRC	2 or 4 MHz IRC	MCGIRCLK optional	static—no clock output	OFF
Core clock	OFF except in debug	2 or 4 MHz IRC	OFF	OFF	OFF	OFF
System clock	OFF except in debug	2 or 4 MHz max OFF in CPO	2 or 4 MHz IRC	OFF	OFF	OFF
Bus clock	OFF except in debug	1 MHz L series, 2 or 4 MHz IRC OFF in CPO	2 or 4 MHz IRC	OFF	OFF	OFF
Memory and memory interfaces						
Flash	powered	1 MHz max access—no pgm	low power	low power	OFF	OFF
Portion of SRAM_U[4]	low power	low power	low power	low power	low power	low power in VLLS3,2
Remaining SRAM_U and all of SRAM_L	low power	low power	low power	low power	low power	low power in VLLS3
Cache	low power	low power	low power	low power	low power	OFF
FlexMemory[5]	low power	low power[6]	low power	low power	low power	low power in VLLS3, OFF in VLLS2 and VLLS1
VBAT Register files[7]	powered	powered	powered	powered	powered	powered
System Register file	powered	powered	powered	powered	powered	powered
DDR controller	low power	low power	low power	low power	low power	OFF
SDRAM controller	low power	FF disabled in CPO	FF	low power	low power	OFF
NFC	static	FF	FF	static	static	OFF
FlexBus	static	FF disabled in CPO	FF	static	static	OFF
EzPort	disabled	disabled	disabled	disabled	disabled	disabled
Communication interfaces						
USB HS Phy	static	static	static	static	OFF	OFF
USB HS Controller	static, wakeup on resume	static, wakeup on resume	static, wakeup on resumeF	static	static	OFF
USB FS/LS	static, wakeup on resume	static, wakeup on resume	static, wakeup on resume	static	static	OFF
USB DCD	static	FF	FF	static	static	OFF

Table continues on the next page...

Table 5. Module operation in low-power modes (continued)

Modules	Stop	VLPR	VLPW	VLPS	LLSx	VLLSx
USB Voltage Regulator	optional	optional	optional	optional	optional	optional
Ethernet	wake-up	static	static	static	static	OFF
UART0, UART1	static, wakeup on edge	up to 250kbits/s static, wakeup on edge in CPO	up to 250kbits/s	static, wakeup on edge	static	OFF
LPUART Kinetis L & later Kinetis K	Async operation FF in PSTOP2	1-4 Mbps Async operation in CPO	1-4 Mbps	Async operation	static	OFF
FLEXIO Kinetis K2	Async operation FF in PSTOP2	1-4 Mbps Async operation in CPO	1-4 Mbps	Async operation	static	OFF
UART(other)	static, wake-up on edge FF in PSTOP2	125-250 kbit/s statis, wakeup on edge in CPO	125-250 kbit/s	static, wake-up on edge	static	OFF
SPI Kinetis L K2	static FF in PSTOP2	static, slave mode receive static in CPO	master mode 500 kbps, slave mode 250 kbps static, slave mode receive in CPO	static, slave mode receive	static	OFF
SPI	static FF in PSTOP2	1 Mbit/s (slave) 2 Mbit/s (master) static in CPO	1 Mbit/s (slave) 2 Mbit/s (master)	static	static	OFF
I^2C	static, address match wake-up	100 kbit/s	100 kbit/s	static, address match wake-up	static	OFF
CAN	wake-up FF in PSTOP2	250-500 kbit/s	250-500 kbit/s	wake-up	static	OFF
I^2C Kinetis L	50 kbps	static, address match wake-up FF in PSTOP2	50 kbps static, address match wake-up in CPO	static, address match wake-up	static	OFF
I^2C Kinetis K2	static, address match wake-up FF in PSTOP2	200 kbit/s static, address match wake-up in CPO	200 kbps	static, address match wake-up	static	OFF
I^2S	Async operation with external clock[8]	FF Async operation in CPO	FF	FF with external clock[9]	static	OFF
SDHC	wake-up	FF	FF	wake-up	static	OFF
Security						
CRC	static	FF	FF	static	static	OFF

Table continues on the next page...

Table 5. Module operation in low-power modes (continued)

Modules	Stop	VLPR	VLPW	VLPS	LLSx	VLLSx
RNG	static	FF	FF	static	static	OFF
DryIce [7]	FF	FF	FF	FF	FF	FF
MMCAU	static	FF	FF	static	static	OFF
Timers						
TPM Kinetis L K2	Async operation FF in PSTOP2	FF Async operation in CPO	FF	Async operation	static	OFF
FTM	static FF in PSTOP2	FF	FF	static	static	OFF
PIT	static FF in PSTOP2	FF static in CPO	FF	static	static	OFF
PDB	static FF in PSTOP2	FF static in CPO	FF	static	static	OFF
LPTMR Kinetis L K2	Async operation FF in PSTOP2	FF	FF	Async operation	Async operation	Async operation[10]
LPTMR	FF	FF	FF	FF	FF[11]	Async operation[10]
RTC Kinetis L K2	Async operation FF in PSTOP2	FF Async operation in CPO	FF	Async operation	Async operation	FF[12]
RTC - 32kHz OSC	FF	FF	FF	FF	FF[13]	FF
CMT	static	FF	FF	static	static	OFF
Analog						
16-bit ADC	ADC internal clock only	FF	FF	ADC internal clock only	static	OFF
CMP - Kinetis L[14]	HS or LS compare FF in PSTOP2	FF HS or LS compare in CPO	FF	HS or LS compare	LS compare	LS compare in VLLS1/3, OFF in VLLS0
CMP[15]	HS or LS compare	FF	FF	HS or LS compare	LS compare	LS compare in VLLS1/2/3, OFF in VLLS0
6-bit DAC	static FF in PSTOP2	FF static in CPO	FF	static	static	static, OFF in VLLS0
VREF	FF	FF	FF	FF	static	OFF
OPAMP	FF	FF	FF	FF	static	OFF
TRIAMP	FF	FF	FF	FF	static	OFF
PGA	FF	FF	FF	FF	static	OFF
12-bit DAC	static FF in PSTOP2	FF static in CPO	FF	static	static	static

Table continues on the next page...

Table 5. Module operation in low-power modes (continued)

Modules	Stop	VLPR	VLPW	VLPS	LLSx	VLLSx
Human-machine interfaces						
GPIO	static output, wake-up input FF in PSTOP2	FF IOPORT write only if in CPO	FF	static output, wake-up input	static output, pins latched, wake-up input	static output, pins latched, wake-up input
Segment LCD	FF	FF	FF	FF	FF[16]	FF[17]
Graphic LCD	static	FF	FF	static	static	OFF
TSI Kinetis L	Async operation FF in PSTOP2	FF Async operation in CPO	Async operation	Async operation	Async operation	Async operation
TSI	wake-up	FF	FF	wake-up	wake-up[17]	wake-up[11]

1. Using the LLWU module, the external pins available for this chip do not require the associated peripheral function to be enabled. It only requires the function controlling the pin (GPIO or peripheral) to be configured as a digital input to allow a transition to pass to the LLWU.
2. Since LPO clock source is disabled, filters will be bypassed during VLLS0.
3. [MC2]The VLLSCTRL[PORPO] bit in the SMC module controls this option.
4. A 4, 8, 16 32 KB portion of SRAM_U and 32KB SRAM L (with devices with PORPO) block is left powered on in low power mode VLLS2. There is no VLLS2 mode on Kinetis L.
5. FlexRAM is always powered in VLLS3. When the FlexRAM is configured for traditional RAM, optionally powered in VLLS2 mode. When the FlexRAM is configured for EEPROM, off in VLLS2 mode.
6. FlexRAM enabled as EEPROM is not writable in VLPR and writes are ignored. Read accesses to FlexRAM as EEPROM while in VLPR are allowed. There are no access restrictions for FlexRAM configured as traditional RAM.
7. These components remain powered in BAT power mode.
8. Use an externally generated bit clock or an externally generated audio master clock (including EXTAL).

   FF in PSTOP2
9. Use an externally generated bit clock or an externally generated audio master clock (including EXTAL).
10. LPO clock source is not available in VLLS0. Also, to use system OSC in VLLS0 it must be configured for bypass (external clock) operation. Pulse counting is available in all modes.
11. System OSC and LPO clock sources are not available in VLLS0
12. In VLLS0 the only clocking option is from RTC_CLKIN.
13. RTC_CLKOUT is not available.
14. CMP in stop or VLPS supports high speed or low speed external pin to pin or external pin to DAC compares. CMP in LLS or VLLSx supports only low speed external pin to pin or external pin to DAC compares. Windowed, sampled & filtered modes of operation are not available while in stop, VLPS, LLS, or VLLSx modes.
15. CMP in Stop or VLPS supports high speed or low speed external pin to pin or external pin to DAC compares. CMP in LLS or VLLSx supports only low speed external pin to pin or external pin to DAC compares. Windowed, sampled & filtered modes of operation are not available while in Stop, VLPS, LLS, or VLLSx modes.
16. End-of-frame wake-up not supported in LLS and VLLSx.
17. TSI wakeup from LLS and VLLSx modes is limited to a single selectable pin.

12 Using external memories and peripherals - Use case with DDR Memory Controller and DDR Low Power Modes

12.1 Controlling DDR memory and interface code example

The Kinetis K70/K61 is integrated with a dram (DDR) memory controller that has additional low power modes that will be referred to as DDR low power modes. The DDR controls can help achieve low power operation even when executing code from DDR memory. When operating with a DDR2 or LPDDR1 memory, the key to the lowest power system is managing the transitions between the MCU run modes and the memories and the MCUs low power modes.

This section of the application note explores some techniques to use that will allow low power operation of the Kinetis K70/K61 with the integrated DDR memory controller driving DDR2 or LPDDR memories. Sections below describe the DDR memory controls and the entry and exit steps to use to transition between DDR and MCU low power modes and run modes.

Use these descriptions, in conjunction with the application note demo code, to explore the controls and evaluate some different scenarios of the MCU and memory operation.

What about using the DDR in VLPR mode. Note that the DDR runs from the PLL. Therefore, before entering any MCU mode that would turn off the PLL such as VLPR, VLPS, STOP, LLS, VLLSx, the DDR must be put into a low power mode.

The same approach to low power mode entry can be applied to other external memory and peripheral types such as SDRAM, serial flash and sensors. Most of these device have low power modes that need to be set up prior to the MCU entering it's low power mode.

Note: The SDK low power mode entry functions have before and after callback functions that are ideal for this kind of code. The before setting up the LPDDR memory or peripheral before allowing the MCU to enter the low power mode and the after for the recovery operations. Please refer to the SDK API Reference Manual for details.

12.1.1 DDR power mode controls and Use

There are 5 low power modes for the External DDR Memories and Controller, LP modes 1 and 2 do not retain the contents of the memory.

Table 6. DDR Memory Low Power Mode Control with Register DDR_CR16_LPCTRL

Mode	Memory Type	Value	DDR Low Power Mode
1	DDR2 and LPDDR	0x10	DDR Memory Power-Down
2	DDR2 and LPDDR	0x08	DDR Memory Power-Down with Memory Clock Gating
3	DDR2 and LPDDR	0x04	DDR Memory Self-Refresh
4	DDR2 and LPDDR	0x02	DDR Memory Self-Refresh with Memory Clock Gating
5	LPDDR	0x01	DDR Memory Self-Refresh with Memory and Controller Clock Gating. This LP mode 5 cannot be entered Manually

12.1.2 Before MCU low power Mode Entry

The software flow would normally call a function to place the MCU into one of the low power modes like, Stop, VLPS, LLS or VLLS3. If using external memories that allow low power operation, the function sets the memory and memory controller state before entering the requested low power mode. When using external DDR memories in conjunction with the MCU low power modes, the entry into the low power modes is preceded with the sequence to put the DDR memory into it's low power mode, the DDR memory controller is disable and the MCU DDR pads are set to a lower power state.

The LLS low power mode entry code below includes the DDR controller and SIM_MCR register writes to put the DDR memory into mode 4 – DDR self refresh mode with clock gating. This is the lowest power mode for a DDR2 type memory.

```
    /* send command to memory to enter ddr low power mode 4 */
    DDR_CR16 = DDR_CR16_LPCTRL(0x02);     //bit 17 set of DDR_CR16
    SIM_MCR =   SIM_MCR_DDRDQSDIS_MASK | SIM_MCR_DDRCFG(1)
              | SIM_MCR_DDRS_MASK ;
    for (i = 0;i < 100;i++){
         if ( SIM_MCR & SIM_MCR_DDRS_MASK == 0x02)
         break;
         }
    SIM_SCGC3 &= ~SIM_SCGC3_DDR_MASK;
   // turned off the clock gate for the DDR Controller
```

12.1.3 After MCU low power Mode Exit

The MCU is now in LLS mode.

If a falling edge event occurs on PORTE1, the MCU will exit LLS and return to Run mode.

After exiting from LLS, code clears the wake-up event flag in the LLWU flag register

If interrupts and the LLWU interrupt are enabled, execution resumes with the LLWU interrupt service routine. Then code execution returns to the instruction following the WFI or STOP instruction.

If interrupts are disabled, the execution resumes with the instruction after the WFI, or STOP instruction.

At this point the DDR memory is disabled. The DDR cannot be accessed by any of the code below until the PLL restarts and the DDR memory exits self refresh. Be careful not to have any interrupts enabled that might attempt to access the DDR until the PLL is restarted and this code has completed.

```
  /* after exiting LLS (can be in LLWU interrupt service Routine)
  * clear the wake-up flag in the LLWU-write one to clear the flag
  */
  if (LLWU_F1 & LLWU_F1_WUF0_MASK) {
      LLWU_F1 |= LLWU_F1_WUF0_MASK;

 /* Enable DDR controller clock gate */
      SIM_SCGC3 |= SIM_SCGC3_DDR_MASK; /* Enable DDR controller clock gate */
     DDR_RCR = DDR_RCR_RST_MASK;     // reset DDR PHY

     DDR_RCR = 0;
     /* soft reset to DDR RCR, DDR_DQS analog circuit enabled,
        lpddr half strength, DDRPEN = 1 */
     SIM_MCR &= ~SIM_MCR_RCRRSTEN_MASK;
     SIM_MCR &= ~SIM_MCR_DDRSREN_MASK;
     SIM_MCR &= ~SIM_MCR_DDRDQSDIS_MASK;
     DDR_CR50 |= DDR_CR50_CLKSTATUS_MASK;
     /* disable the DDR memory low power mode  */
     DDR_CR16 &= ~DDR_CR16_LPCTRL_MASK;  // all bits are zero
     DDR_CR16 |= DDR_CR16_QKREF_MASK;    // since set by lpddr_init code
```

```
    DDR_CR15 &= ~DDR_CR15_SREF_MASK;    // Disable self-refresh mode
    SIM_MCR |=  SIM_MCR_DDRPEN_MASK;
   /*wait for write to complete to before continuing */
   dummyread = SIM_MCR;
}
```

13 Power Measurement

13.1 Power measurement results and tips

13.1.1 Power measurement in real time

The typical way of measuring the current consumption of the application is with a digital multimeter. Measuring the current of a Kinetis MCU in these lowest power modes requires +/– 10 nA accuracy over a wide range.

A much better view of the real-time current consumption is available if you measure the voltage drop across a resistor or a 10 turn current loop in series with the VDD of the MCU.

Figure 13 and Figure 14 show the real-time current measurements across a 10 ohm resistor using an active differential probe and oscilloscope.

The software used to take these measurements was ported from a piece of demonstration code that repeatedly performed an ADC measurement of a single input, placing the results in a table. Every 256 samples, an FFT calculation using the Kinetis CMSIS DSP library was initiated. The ADC measurement and the FFT number crunching were done in either Run or VLPR mode. While not in Run mode, the MCU was put into VLPS mode.

Figure 13 shows the real time measurement results for two devices, K40 100 MHz MCU in Run mode and a K20 50 MHz MCU in VLPR mode.

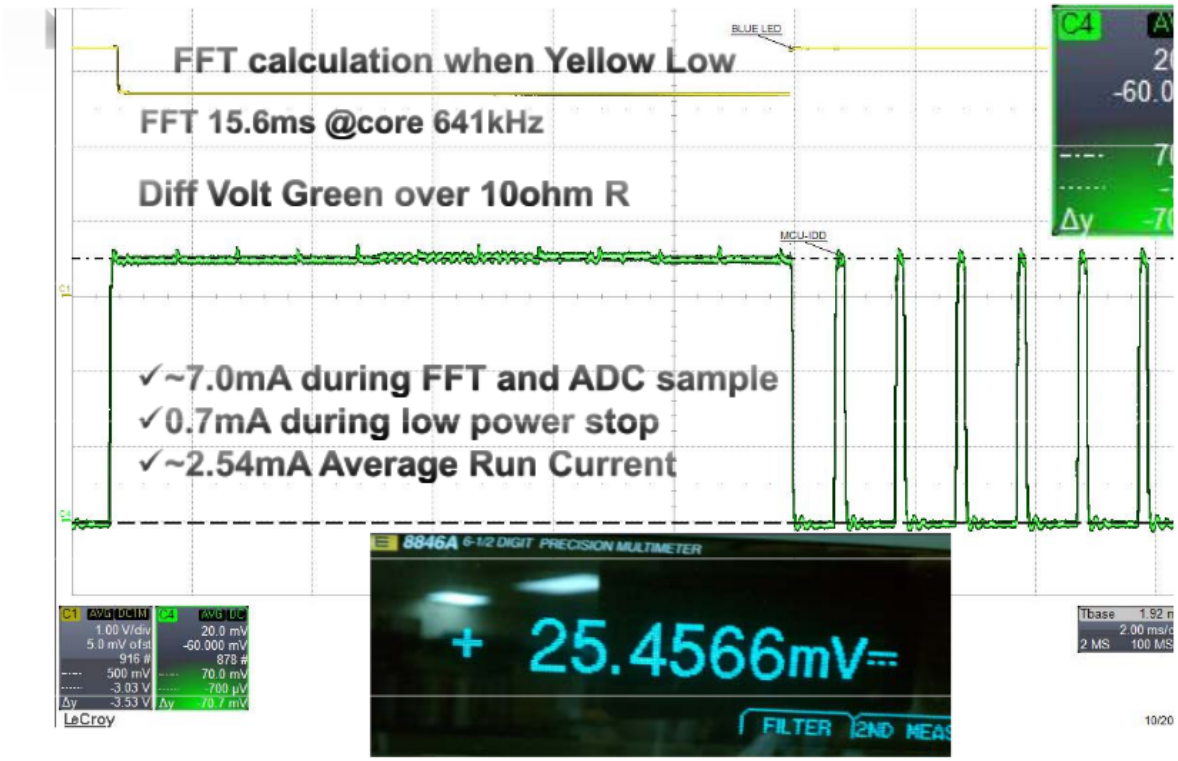


Figure 13. Real-time measurement of Run and VLPS modes

The measurements taken above were across a precision 10 ohm resistor using a differential probe. Divide the voltage measurement by 10 to get the value for current.

Run currents were approximately 7 mA and VLPS currents were approximately 700 µA. The average voltage taken with a precision digital multimeter was 2.54 mA.

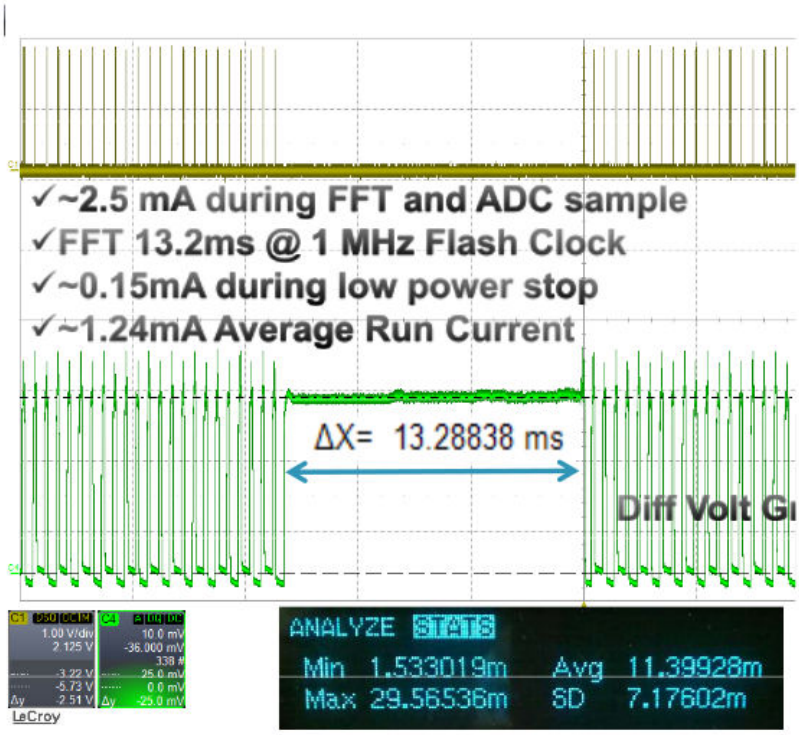


Figure 14. Real-time measurement of VLPR and VLPS modes

The measurement above is across a precision 10 ohm resistor using a differential probe. Divide the voltage measurement by 10 to get the value for current.

Measuring voltage across the same 10 ohm precision resistor using a precision digital multimeter with statistical calculations ability, the minimum, maximum, and average currents, as well as the standard deviation are superimposed at the bottom of the diagram.

Very Low Power Run currents ~ 2.5 mA and VLPS currents are ~ 150 µA. The average voltage taken with a precision digital multimeter (DMM) is 1.2 mA.

13.1.2 Tips for making low-power measurements on the bench

- **External.** The suggestions below address the most common issues encountered when trying to duplicate the data sheet current specs.
  a. **When using a digital multimeter (DMM), use "Manual Range Mode."** Using a DMM with an auto-ranging function enabled may cause LVD and POR resets. This is most common when you are exiting from one of the low-power modes like LLS or VLPS back to Run. The DMM has changed the range to a micro-amp or nano-amp range while the MCU is in the low-power mode and the sudden inrush of current requires the DMM to change range. The range change does not happen fast enough and the MCU starves and pulls the VDD level below the LVD or POR limits.
  b. **Disconnect the debugger and power cycle the MCU.** With the JTAG debugger for Kinetis is attached, the MCU may have the debugger module in the MCU active, clocking and consuming power. The external debugger hardware may also load the I/O of the JTAG port when attached. As a result, your low-power measurements will be higher than expected.

c. **Isolate the MCU VDDs.** If you want to measure the current draw of the MCU, then remove the other IC and component networks that are sourced by the voltage supply sourcing the MCU. For example, the early tower boards provided for Kinetis have a potentiometer connected between MCU_VDD and ground. A 5 K potentiometer across a 3.6 V supply pulls 720 µA. This is huge when considering that the MCU consumes around 1.5 µA in VLLS1.
d. **Measure VBAT or RTC VDD by itself.** The current supplying the RTC VBAT domain is typically much less than 1 µA while the RTC is running, keeping time. This very small IDD can be swamped by Run or VLPR currents.
e. **Match impedance of inputs.** If the impedance of high speed signals (fast edge transitions) are not well matched then the signals can "ring" and exceed the Vdd supply of the device. This can result in the signal providing current to the device through the input protection diodes. This is particularly true for high speed input clocks. This issue can result in negative IDD measurements while in the lowest power modes.
f. **Match voltage levels.** Although the MCU input pins are 5 V tolerant on some parts, when the MCU goes into the low-power modes, measurement of the current through MCU_VDD will be affected by any input higher than MCU_VDD. The higher input pin will back power the MCU through the input pin, resulting in negative IDD reading in low-power modes.
g. **Reduce pin loading of the MCU.** When the MCU sources current through the output pins, the power is being sourced through MCU_VDD. This is most evident when you output high frequency signals to an output pin as you might with the FlexBus clock and address/data pins.
h. Build code with debug hooks that do not need the Debugger or high speed serial interface that consume a lot of current.

- **Internal to the MCU.** Below is a list of the most common issues that can prevent you from getting to the lowest data sheet current specs.
  a. Watchdog is not disabled, causing resets. Disable or service the watchdog.
  b. The clock monitor is not disabled which may cause resets. Disable all of the clock monitors.
  c. A crystal oscillator is enabled in low power mode. The RTC oscillator, typically consumes <500 na of current.
  d. The CLKOUT signal is being output to a pin. Any pin that is constantly changing state will draw power.
  e. The requested low-power mode is not allowed with a corresponding bit in the PMPROT register. For example, if ALLS is not set in the PMPROT and the WFI instruction is executed, you will enter Stop mode and not LLS.
  f. The clock gate for a module is not enabled before it is read or written. This causes a reset before the MCU tries to enter the low-power mode.
  g. The clock gate for a module that must acknowledge the mode controller mode entry request is turned off prior to low-power mode entry. This will result in a Stop mode Acknowledge reset.
  h. Failure to un-comment out the call to the stop or sleep function after debugging is complete will keep you in the higher run current mode.
  i. If the MCU software is polling the RTC registers or VBAT Register File module, the current of the VBAT domain will increase 2–5 times over normal because of the constant accesses from the code. Therefore, to keep the lowest current, the RTC registers should not be polled.
  j. The frequency of wake-up events is too high, which means that the MCU spends more time in Run or VLPR mode than in a low-power mode. As shown above, the transition time from low-power mode to Run mode is quick. If the MCU only spends 9 ms in run and 1 ms in a low-power mode, the average current of the system will be considerably higher than if the MCU was running only 1 ms every 1 second.
  k. The MCU is running at a much higher frequency than is need to accomplish the work. Throttle the clock with the SIM CLK dividers or reduce the clock. Obviously, the higher the MCU frequency the higher the IDD of the MCU. Reduce the clock and lower the Run or Wait current. However there is some trade-off. If the current of Run mode can be tolerated, then getting work done as quickly as possible and going right back to low power mode is more advantageous than running a slower clock in Run mode.
  l. An input pin is floating without an internal or external pull device. This can result in 50-80 uA of current per pin. This include the JTAG or SWD pins. Disable the JTAG pins on portA or properly terminate the inputs.

14 LLWU pin and module Wakeup

14.1 LLWU hardware considerations

14.1.1 Pin wake-up events

The MCU wake-up logic is designed to use interrupts to wake the MCU from low power modes. It is recommended that interrupts be unmasked and the LLWU interrupt enabled. If global interrupts are masked or the LLWU interrupt is not enabled, the application may have unpredictable operation.

LLWU_Px are external pin inputs. Any digital input function multiplexed on the pin can be selected as the wake-up source.

You can choose whether a pin will cause a wake-up event with a rising edge, falling edge, or any edge. The LLWU controls the actions of each of the pins identified in the LLWU wake-up sources table. This is a select group of pins that, for the most part, does not change across all of the Kinetis family of devices. Kinetis L devices may only have a subset of the pins defined.

The pins listed in the table below can act as wake-up sources as long as the pin mux is actively selecting a digital input pin function such as a GPIO input or UART RX or RTS, or SPI slave select input in slave mode The wake-up event will not wake-up the MCU if the pin mux is not an LLWU wake-up pin, is a digital output or is selecting an analog function. When the pin is a digital input function, there is an asynchronous path from the pad through the Asynchronous Wakeup Interrupt Controller (AWIC) to the LLWU. The asynchronous path allows the wake-up event to pass to the LLWU without clocks and clock synchronizers. See the chip's signal multiplexing table for the digital input signal options and the LLWU wake-up pin list.

14.1.2 Integrating Pin Interrupt and LLWU Wake-up functionality

For LLWU wake-up input pins, it is recommended to enable the pin interrupt functionality on the LLWU wake-up pin. In the event that the pin interrupt is not enabled, there is a window of time while the MCU transitions into the low leakage mode, that an edge transition might be missed. If the pin interrupt functionality was enabled and an edge occurred during this small transition time the low power entry would be aborted and the MCU would service the pin interrupt.

In Kinetis K devices all LLWU wake-up pins are also capable of being pin interrupt sources. In Kinetis L devices some LLWU wake-up pins do not have this dual functionality.

14.1.3 Module wake-up events

LLWU_M0IF–M7IF are the LLWU internal peripheral interrupt flags.You can choose up to eight module wake-up sources in the LLWU. To complete the initialization of the wake-up source, the module must be enabled and set up to create an interrupt and the LLWU Module Flag enabled as a LLWU wake-up source. The interrupt from an enabled module will create a wake-up event.

After a wake-up with a peripheral, both the LLWU interrupt flag and the waking module's flag will be set in the NVIC pending interrupt register. If waking from LLS, the LLWU interrupt routine will be serviced first. If the LLWU service routine code does not clear the module flag and the module pending interrupt flag in the NVIC, then the module interrupt will be called at the completion of the LLWU interrupt service routine. If a module flag is cleared a read-back of the flag is recommended to make sure it is cleared before leaving the interrupt service routine.(see Serialization of operations section for more detail.

14.1.4 Filtered wake-up inputs

In Kinetis K there is a glitch filter function available for up to two external GPIO wake-up pins. In subsequent MCUs the glitch filter is based upon the internal LPO clock and therefore will act to reject any pulse shorter than three LPO cycles.

You can write the two filter control registers, LLWU_FILT1 and LLWU_FILT2, to select which of the 16 input pins would be sent to the two filters. It does not make much sense to enable a pin to one of the filters and also enable the same pin enable functions. Enabling both the pin and the filter enables bypasses the filter functionality since the MCU will always wake on the edge event. Do not choose the edge flag wake-up if the filter is being enabled for a pin. For example, to use LLWU_P7 for filter 1, write a 0b111 to the FILTSEL bits and enable the filter edge (FILTE) in the LLWU_FILT1 register. Then write 0b00 to the pin enable bits of the LLWU_PE2 register to disable any edges from causing a wake-up event.

14.1.5 Wake-up sources

NOTE

$\overline{\text{RESET}}$ is also a wake-up source, depending on the bit setting in the LLWU_CS register for MC1 or the LLWU_RST register for MC2. See the Reset filter setting in the Reset Control Module (RCM) for more information on how to configure the filtering on the Reset pin input.

Table 7. Example wake-up sources for LLWU inputs (see the reference manual Chip Configuration section for MCU-specific implementation)

Input	Wake-up source	Input	Wake-up source
LLWU_P0	PTE1/LLWU_P0 pin	LLWU_P12	PTD0/LLWU_P12 pin
LLWU_P1	PTE2/LLWU_P1 pin	LLWU_P13	PTD2/LLWU_P13 pin
LLWU_P2	PTE4/LLWU_P2 pin	LLWU_P14	PTD4/LLWU_P14 pin
LLWU_P3	PTA4/LLWU_P3 pin[1, 2]	LLWU_P15	PTD6/LLWU_P15 pin
LLWU_P4	PTA13/LLWU_P4 pin	LLWU_M0IF	LPTMR[3]
LLWU_P5	PTB0/LLWU_P5 pin	LLWU_M1IF	CMP0
LLWU_P6	PTC1/LLWU_P6 pin	LLWU_M2IF	CMP1
LLWU_P7	PTC3/LLWU_P7 pin	LLWU_M3IF	CMP2
LLWU_P8	PTC4/LLWU_P8 pin	LLWU_M4IF	TSI
LLWU_P9	PTC5/LLWU_P9 pin	LLWU_M5IF	RTC Alarm
LLWU_P10	PTC6/LLWU_P10 pin	LLWU_M6IF	DryIce (tamper detect)
LLWU_P11	PTC11/LLWU_P11 pin	LLWU_M7IF	RTC Seconds

1. For Kinetis Rev 1.x devices, a falling edge input that remains low on this pin when waking up the MCU from VLLSx modes with the EzPort enabled causes entry into EzPort mode during the reset sequence. A falling edge input that remains low on this pin when waking up the MCU from any non-VLLSx mode with the NMI function selected in its port control register asserts an NMI exception on low power mode recovery. The same occurs when recovering from VLLSx modes if EzPort is disabled; otherwise, EzPort mode is entered.
2. For MC2 devices the $\overline{\text{EZP\_CS}}$ signal is checked only on *Chip Reset not VLLS*, so a VLLS wake-up via a non-reset source does not cause EzPort mode entry. If NMI was enabled on entry to LLS/VLLS, asserting the NMI pin generates an NMI interrupt on exit from the low power mode. On Kinetis devices NMI can also be disabled via the FOPT[NMI_DIS] bit.
3. Requires the peripheral and the peripheral interrupt to be enabled. The LLWU's WUME bit enables the internal module flag as a wake-up input. After wake-up, the flags are cleared based on the peripheral clearing mechanism.

15 References and Revision History

15.1 References

The references listed below have additional information regarding power management for Kinetis. You can find a particular reference manual, data sheet, or errata report by choosing a device on the Kinetis (www.freescale.com/Kinetis) pages, and then selecting the Documentation tab. To find the application notes below, search for the document number.

15.1.1 Kinetis Microcontrollers (MCUs)

Kinetis MCUs consist of multiple hardware and software-compatible ARM® Cortex®-M0+ and -M4-based MCU series with exceptional low-power performance, scalability and feature integration. At the time of this application note writing there are 8 families

1. Kinetis K - Broad and Scalable and Compatibility from 50-180MHz, 32 KB to 2 M Flash,up to 256 KB RAM, floating point unit, security, analog, timers and serial interfaces
2. Kinetis L - Worlds most energy efficient microcontrollers, up to 48 MHz, 8KB-256KB Flash,up to 32 KB RAM,low power timer and smart peripherals
3. Kinetis E - Enhanced ESD/EMC performance, up to 48 MHz, 8 KB-128 KB Flash,up to 8K RAM,ADC, Flextimers,high current output.
4. Kinetis EA - Automotive grade qualification, up to 48 MHz, 8 KB-128 KB Flash,up to 8 K RAM,ADC, Flextimers,high current output.
5. Kinetis MINI - the worlds smallest ARM-based microcontrollers packaged in chip-scale packages.
6. Kinetis M - Metering and measurement applications, 50 MHz, 32 KB-128 KB Flash,up to 32 KB SRAM,high accuracy Sigma Delta analog front end, security and serial interfaces.
7. Kinetis V - Designed for motor control and digital power conversion, 75-200 MHz, 16 KB-2MB Flash,up to 256 KB SRAM, high speed analog& timing peripherals, math accelerators.
8. Kinetis W - Integrated with sub-1 GHz and 2.4 GHz RF transceivers, 48-50 MHz, 32 KB-512 KB Flash, up to 64 KB SRAM, optimized for embedded wireless solutions.

Realize that the system mode controllers on all of these MCUs are not the same. Even within one family grouping, such as the K20 family for instance, there are some significant improvements in the system mode controller over time. Please be diligent in reviewing each of the reference manuals and data sheets of the MCU you are designing with to make sure it has the features you need for your application.

Please see http://www.freescale.com/kinetis for the detailed documentation and updates.

15.1.2 Reference documents and links

There is a growing community of MCU applications designers that you can access through the following references.

- **MCU Reference Manuals.** The reference manuals contain MCU-specific implementation details in the Chip Configuration chapters and the module specific sections. Look for a detailed description of the Clocks, Resets and Power Management Features of each MCU.
- **MCU Data Sheet Specifications.** The data sheet includes all of the MCU specifications, including clock rates, low-power mode power consumption expectations.
- **Errata for MCUs.** Device errata identify what functionality and/or specification is not being met due to a problem with the MCU.
- **Kinestis SDK.** Software HAL and Driver Reference manuals in the KSDK downloaded docs folder.

- **AN4447, *Freescale MQX™ Low-Power Management*.** Freescale offers MQX, the full-featured and complimentary Real-Time Operating System (RTOS). Starting with version 3.8, MQX integrates a Low-Power Management (LPM) driver to take advantage of the low-power operating modes in MQX applications.
- **AN4470, *Using Low Power modes on the Kinetis Family***. Includes low-power mode entry demonstration code that can be used across Kinetis devices with mode controller 2.
- **Kinetis Solutions Advisor,**. Access and run the Kinetis Solutions Advisor to help select the MCU you need http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=SOLUTION_ADVISOR.
- **Kinetis Community.** Go here to access the Kinetis Community. https://community.freescale.com/community/kinetis
- **MBED Community** Go here to access the MBED community for Kinetis. http://developer.mbed.org/teams/Freescale/ or
- You can evaluate the low-power modes with the many Tower and Freedom kits available for Kinetis.

15.1.3 Revision History

The following table summarizes content changes since the previous release of this document.

Table 8. Revision History

Rev. No.	Date	Substantial Changes
1	Nov. 2011	Added details of newer Kinetis devices and new Low Power Modes including Kinetis L series MCUs.
2	Mar. 2015	Add details of newer Kinetis devices included K20, K61, K70, K22F, KVx, K65, K66, K24... Added Power Management Techniques section Remove power technology details Add Kinetis SDK example code for low power mode entry and exit Removed references to Coldfire+ MCUs Add updates to MCU clocks and diagrams including MCG Lite. Update document with new features and ideas throughout. Add section on what to do before entering Low Power modes. Use case includes DDR and LPDDR memories. Add tables on mode entry and exit Add reference to SDK API reference manual. Added mode entry notes and code for LLS2 and LLS3. Update power mode state diagram.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN4503
Rev. 2
4/2015

13. Bibliografía

13.1. Libros

[1] Shooman, M. L. (1968). Probabilistic reliability: an engineering approach. McGraw-Hill.

[2] Valvano, J. W. (2011). Embedded microcomputer systems: real time interfacing. Cengage Learning.

[3] Pérez García, M. A., Álvarez Antón, J. C., Campo Rodríguez, J. C., Ferrero Martín, F. J., & Grillo Ortega, G. J. (2004). Instrumentación electrónica. 2004) España: Thomson.

[4] Jones, C., & Ertas, A. (1993). The engineering design process. New York: John Wiley& Sons, Inc. Otto K. and Wood K.(2001), Product Design, New Jersey: Prentice Hall. Boothroyd G.(2002), Product Design for Manufacture and assembly, 2.

[5] APA Shtub, A., Bard, J. F., & Globerson, S. (1994). Project management: engineering, technology, and implementation. Prentice-Hall, Inc..

13.2. Notas de aplicación

[6] Freescale Semiconductor Inc. FRDM-KL25Z (2012) User's Manual. Rev. 1.0

[7] Freescale Semiconductor Inc. AN4503 (2015) Rev 2.0

13.3. Links útiles

Página web de FOVISEE
http://www.clarin.com/medio_ambiente/instalaron-alimentadas-energia-Moreno-quieren_0_624537769.html