Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Electrónica

BUAP

"Adquisición y Transmisión de Datos de Microsensores de Flujo"

Tesis Profesional

Qué Para Obtener el Título de:
Licenciado en Electrónica

PRESENTA:
Mauricio Huixtlaca Quintana

Dr. Salvador Antonio Arroyo Díaz
ASESOR INTERNO BUAP

Dr. Alejandro Díaz Sánchez
ASESOR EXTERNO INAOE

Puebla, México. **Febrero 2019**

Dedicatoria

Este logro se lo dedico a mis padres, que siempre han creído en mí, por alentarme a seguir adelante y por hacerme sentir orgulloso de tenerlos conmigo

A mis hermanos por estar siempre pendientes de mi

A mi esposa, por estar a mi lado apoyándome en todas mis decisiones
Agradezco a mi suegra, cuñado y familia por toda su ayuda

A mis sobrinas que son mi alegría

A ustedes amigos y compañeros que me han acompañado en esta aventura, compartiendo conmigo muchos momentos alegres y algunos tristes

Agradecimientos

A la directora de la Facultad de Ciencias de la Electrónica, Dra. Luz del Carmen Gómez Pavón y al secretario académico, Dr. Emilio Miguel Soto García por su intervención

Al Instituto Nacional de Astrofísica Óptica y Electrónica que me apoyó y permitió trabajar en sus instalaciones durante mis cursos, servicio social y la tesis, a los técnicos del taller mecánico, laboratorios y biblioteca por su colaboración

Agradezco mi asesor interno de la BUAP, Dr. Salvador Antonio Arroyo Díaz, por compartirme su conocimiento en el área de sistemas digitales y por su disposición en la revisión de este trabajo

En especial agradezco al Dr. Alejandro Díaz Sánchez y al Dr. José Miguel Rocha Pérez del INAOE, por todo el aprendizaje que me transmitieron, por darme la confianza de trabajar en su laboratorio, por todo su apoyo durante mi estancia y por brindarme su amistad

....

BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA ELECTRÓNICA

ACTA DE COLOQUIO DE TESIS

31/OCTUBRE/2012 10:00 Horas

ALUMNO: Mauricio Huixtlaca Quintana

PRESIDENTE: M.C. RODRIGO LUCIO MAYA RAMÍREZ
SECRETARIO: M.C. RICARDO ÁLVAREZ GONZÁLEZ
VOCAL: M.C. VÍCTOR RODOLFO GONZÁLEZ DÍAZ
Asesor Externo: Dr. Alejandro Díaz Sánchez
Asesor Interno: Dr. Salvador Antonio Arroyo Sánchez

TEMA DE TESIS: ***"Adquisición y Transmisión de Datos de Microsensores de Flujo"***

OBSERVACIONES:
Documentar el principio de funcionamiento de los microsensores de flujo.
Incluir diagramas de bloques del sistema y de flujo del programa

APROBADA PARA SU IMPRESIÓN	
APROBADA PARA SU IMPRESIÓN POSTERIOR A LAS CORRECCIONES	X
DEBERA VOLVER A PRESENTAR COLOQUIO	

FIRMA DEL PRESIDENTE

FIRMA DEL SECRETARIO

FACULTAD DE CIENCIAS DE LA ELECTRÓNICA BENEMÉRITA Secretaría Académica

FIRMA DEL VOCAL

FIRMA DEL ASESOR INTERNO

FIRMA DEL ASESOR INTERNO

FIRMA DEL ASESOR EXTERNO

C.c.p.- Sustentante.
C.c.p.- Expediente.
C.c.p.- Minutario.
**doh.*

Tonantzintla, Pue., 07 de Abril del 2011.

M.C. José Francisco Portillo Robledo
Secretario Académico de la
Facultad de Ciencias de la Electrónica
BUAP

Por este medio le comunico que el INSTITUTO NACIONAL DE ASTROFÍSICA ÓPTICA Y ELECTRÓNICA, certifica que el pasante de Licenciatura en Electrónica **HUIXTLACA QUINTANA MAURICIO con número de matrícula 200113709,** ha sido aceptado para su realización de Tesis de Licenciatura en el Área de Electrónica de este Instituto, con el siguiente titulo y asesores el cual complementa a sus asesores en la Benemérita Universidad Autónoma de Puebla.

Tesista:

MAURICIO HUIXTLACA QUINTANA

Titulo:

"ADQUISICIÓN Y TRANSMISIÓN DE DATOS DE MICROSENSORES DE FLUJO"

Asesor:

DR. ALEJANDRO DÍAZ SÁNCHEZ

Agradeciendo de antemano por su atención, quedo de usted como su atento y seguro servidor.

MARTHA A. OLMOS FLORES
JEFE DEL DEPTO. DE SERVICIOS ESCOLARES

Instituto Nacional de Astrofísica Óptica y Electrónica
Luis Enrique Erro No. 1 Santa María Tonantzintla, Puebla C.P. 72840 Apartado Postal 51 y 216 Puebla C.P. 72000 Conmutador 266-31-00

Dirección General Tel. 247 40 44 Fax. 247 25 80
Dirección de Investigación Tel. 247 43 06
Dirección de Administración y Finanzas Tel. 247 43 21 Fax. 247 01 81
Dirección de Desarrollo Tecnológico Tel. 247 43 14
Coordinación Astrofísica Tel. 247 22 31
Coordinación Óptica Tel. 247 29 40
Coordinación Electrónica Tel. 247 05 17
Dirección de Formación Académica Tel. 247 27 42

Contenido

Índice de figuras

Índice de tablas

Resumen

Como parte del proyecto de diseño y fabricación de microsensores de flujo para cuidados neonatales, se ha diseñado y fabricado una parte del sensor de flujo, caracterizándose hasta el momento, las variables eléctricas.

Esta parte del sensor, está fabricada sobre una superficie de silicio mecanizada para dejar pasar un flujo de gas, y sobre esta rejilla se deposita metal por medio de procesos de fotolitografía. Para este sensor (Sistema micro-electromecánico), se utiliza la técnica de alambre caliente o *Hot-Wire*, que consiste en que el flujo del gas enfríe el metal previamente elevado a cierta temperatura por efecto Joule, de tal manera que el cambio de temperatura será proporcional al flujo que circula por él. Se requiere para la etapa de caracterización dinámica del sistema micro-electromecánico, una etapa de procesamiento y de comunicación digital hacia la computadora, para analizar los datos de la caracterización dinámica.

Para la caracterización dinámica se adquirió el kit de desarrollo eZ430-RF2500 de Texas Instruments, cuyas tarjetas están integradas por un microcontrolador, un transceptor inalámbrico y un programador e interfaz USB, con el fin de adquirir, procesar, transmitir y analizar los datos de la caracterización. El kit incluye 3 tarjetas eZ430-RF2500, de las cuales una opera como punto de acceso de la red inalámbrica que mediante una interfaz USB incluida, comunica los datos a la computadora, las otras tarjetas operan como terminales que envían los datos al punto de acceso. El programa de demostración, se basa en una red inalámbrica con topología tipo estrella, donde las terminales periféricas de la red, le envían los datos al punto de acceso central vía USB a la computadora.

Estudiando al microcontrolador de la tarjeta y analizando el código fuente del programa de demostración, se logra modificar parte de este código predefinido, para adquirir por un puerto externo a la tarjeta, las señales que deseemos analizar y con esto, procesarlas y transmitirlas hacia el punto de acceso en la computadora. Finalmente se adapta por medio de un PCB la señal final de la caracterización y mediante la programación modificada se mapea la señal adquirida de la rejilla a los valores correspondientes de flujo del gas.

Introducción

Como parte del proyecto de colaboración, "Diseño y fabricación de microsensores de flujo para cuidados neonatales" en el que participan los investigadores, Dr. Alejandro Díaz Sánchez y Dr. Joel Molina Reyes de los grupos de Diseño de Circuitos Integrados y Microelectrónica del Instituto Nacional de Astrofísica Óptica y Electrónica. Se desarrolla el diseño de un microsensor de flujo, con la idea de fabricar un sistema micro electromecánico (o MEMS por sus siglas en ingles) y un circuito integrado de aplicación específica (o ASIC por sus siglas en inglés) que en conjunto censarán el flujo de gases utilizados en cuidado neonatal, con el fin de registrar y controlar la cantidad, así como el tiempo que se le suministren estos gases a un recién nacido, de tal manera que el porcentaje de oxígeno en la sangre sea el necesario para su desarrollo durante la incubación.

El estado actual del proyecto se formó con más trabajos de licenciatura, los cuales incluyeron lo siguiente:

- Modelado de la rejilla o MEMS - Se estudió la viabilidad de la propuesta sobre la rejilla tomando en cuenta los materiales, la estructura, y que el arreglo no perturbara y causara turbulencias del flujo al cruzar por la rejilla, todo esto usando el software de análisis ANSYS.

- Layout - Se obtuvo el patrón geométrico de la rejilla para la generación de las mascarillas para el maquinado y la metalización, todo esto con la herramienta L-Edit del software de diseño de circuitos integrados Tanner EDA y siguiendo las reglas de diseño del proceso.

- Fabricación - La etapa de fabricación requirió un entrenamiento en el proceso de 10μm del INAOE para realizar las modificaciones necesarias para próximos diseños y para poderse comunicar con los técnicos del laboratorio de microelectrónica quienes llevarían a cabo la fabricación.

- Caracterización de la rejilla - Se realizaron mediciones de temperatura y voltaje en la rejilla respecto al cambio de corriente, y se calculó el valor de la resistencia, se utilizó

una pistola IR para medir la temperatura. De igual forma con 3 diferentes versiones de la rejilla con modificaciones del número de los canales, el ancho de las pistas de metal y el tamaño de la rejilla para ajustarse a la tubería de prueba, obteniendo los resultados de la Figura 1.

Temperatura °C	Voltaje (V)	Corriente (A)	Resistencia (Ω)
58.30	0.33	0.30	1.09
64.70	0.39	0.35	1.11
70.40	0.45	0.40	1.12
76.40	0.52	0.45	1.16
82.60	0.58	0.50	1.16
89.00	0.66	0.55	1.19
94.00	0.72	0.60	1.20
99.50	0.80	0.65	1.22
104.90	0.87	0.70	1.24
110.70	0.94	0.75	1.26
115.00	1.02	0.80	1.27
120.10	1.09	0.85	1.28
124.90	1.17	0.90	1.30
130.70	1.26	0.95	1.33
135.40	1.34	1.00	1.34
140.70	1.43	1.05	1.36
146.00	1.53	1.10	1.39
150.50	1.61	1.15	1.40
154.00	1.70	1.20	1.41
159.20	1.81	1.25	1.45
164.00	1.95	1.30	1.50

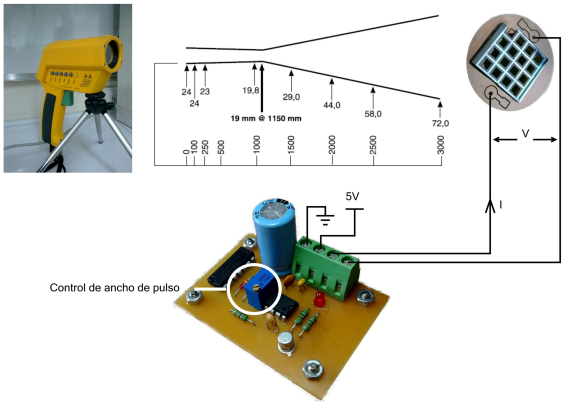


Figura 1: Caracterización de variables eléctricas

Para la caracterización dinámica del MEMS se adquirió la herramienta eZ430-RF2500 de Texas Instruments para comunicar los datos adquiridos, analizarlos en la computadora y obtener la función de transferencia respecto del flujo del gas que se esté utilizando. La figura 2 muestra el *setup* de pruebas, que consta del regulador a la salida del tanque de gas, un medidor de flujo comercial con salida de datos como referencia y por último la rejilla a caracterizar.

Figura 2: *Setup* de pruebas para la caracterización dinámica

Planteamiento del problema

Para la caracterización dinámica del microsensor de flujo, se comparará la respuesta eléctrica de la rejilla, respecto al flujo de gas que se encuentre circulando a través de ella. Para la calibración se empleará un sensor de flujo comercial, dicha referencia análoga se podrá registrar y analizar a la par de la respuesta de la rejilla.

Para lograr obtener la señal de la rejilla como respuesta al flujo de gas, se usará un sistema que: acondicione, adquiera y procese los datos de tal manera que la señal de respuesta, coincida con el valor del sensor de referencia. El diagrama de flujo de la figura 3 muestra el proceso para la caracterización dinámica del microsensor de flujo y el diagrama a bloques del sistema de procesamiento de señales a usar.

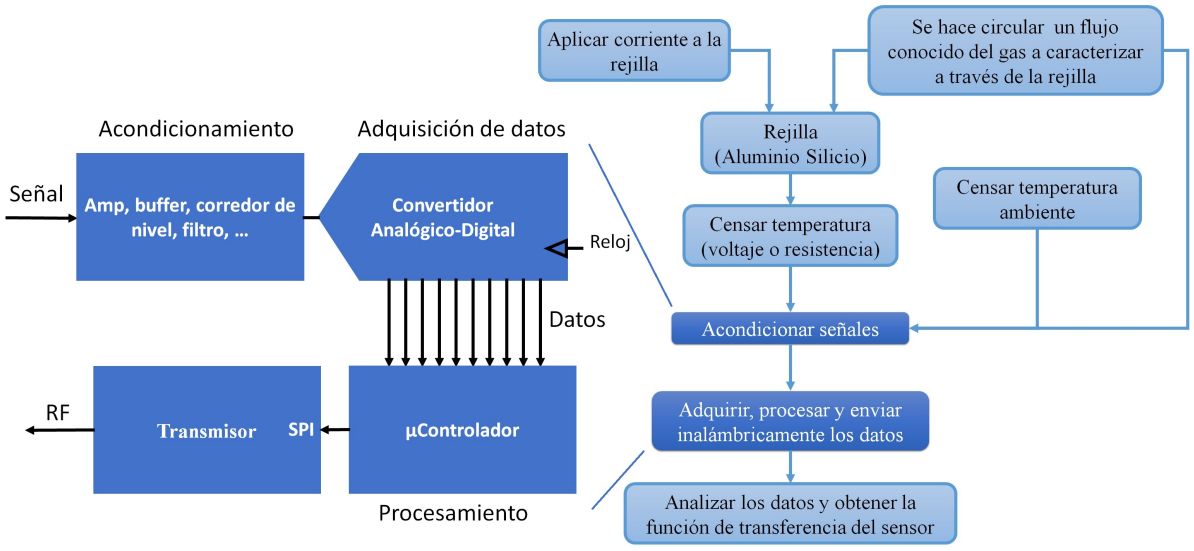


Figura 3: Diagrama de flujo para la caracterización dinámica del sensor

Los bloques que se resaltan en el diagrama, corresponden para esta etapa a la tarjeta de desarrollo, y en un futuro corresponderá al circuito integrado de aplicación específica.

Objetivo General

Desarrollar un sistema para la adquisición y transmisión inalámbrica de datos a partir de una tecnología comercial

Objetivos Particulares

- *Utilizar una tecnología existente en el mercado para la adquisición y transmisión inalámbrica de datos*
- *Diseñar la etapa de acondicionamiento y fabricar la tarjeta de circuito impreso.*
- *Programar la herramienta de desarrollo para adquirir la señal a través de un puerto disponible en la tarjeta, procesar la señal y transmitirla.*
- *Alimentar tanto la tarjeta de acondicionamiento como a la herramienta de desarrollo con una pila de 3V.*
- *Controlar el número de muestras enviadas, para conseguir un bajo consumo de energía durante el tiempo total de transmisión de los datos, controlando el tiempo de descarga de la batería*
- *Emplear un programa de cómputo, que adquiera de los datos recibidos, los almacene en un espacio de memoria, y que estén disponibles para analizarlos en cualquier momento.*

Justificación

El uso de oxígeno en el tratamiento de enfermedades respiratorias en recién nacidos (neonatos), ha sido reportado por más de un siglo. Hay muchos métodos diferentes para suministrar oxígeno de forma no invasiva, a principios del siglo XX se propuso emplear oxígeno, suministrando este a un recién nacido directamente sobre la cara con una mascarilla durante lapsos de tiempo (Figura 4)

La saturación de oxígeno arterial es determinante en el tratamiento de enfermedades respiratorias sobre todo cuando los pulmones de los neonatos no se han terminado de desarrollar, o cuando se presentan enfermedades en pulmones que desembocan en una cianosis, la cual es la mayor complicación fatal de una pulmonía con una insuficiencia de oxígeno en la sangre.

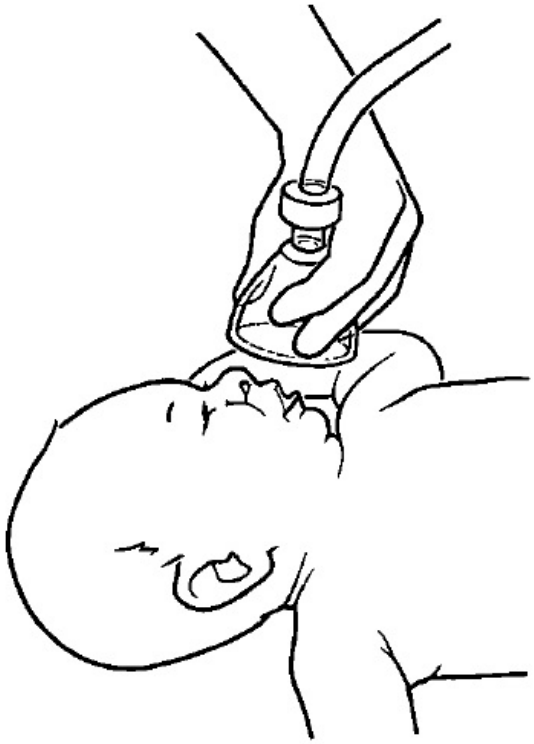


Figura 4: Suministro de oxígeno a una neonato

En 1934 el Doctor Julius Hess [1][2] desarrolló una incubadora (Figura 5) capaz de suministrar una concentración de oxígeno al 40 % cantidad usual en el tratamiento de enfermedades respiratorias en infantes, además de la capacidad de poder controlar la humedad y temperatura y de poder proveer una cantidad de oxígeno extra. Ya en 1940 se encontraban de manera comercial incubadoras que tenían la capacidad de proveer oxígeno para el tratamiento de cianosis, apneas o para un patrón respiratorio periódico en el recién nacido.

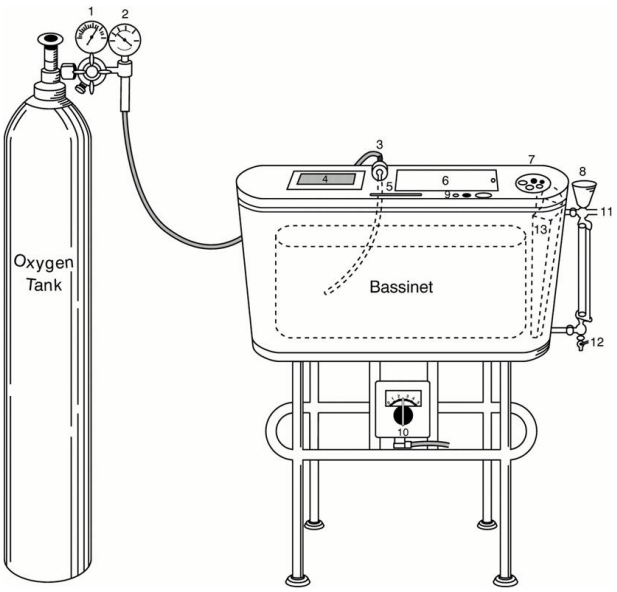


Figura 5: Incubadora de Hess 1934

Sin embargo, desde entonces ya se sabía de la toxicidad del oxígeno ya que, suministrado en exceso al organismo, éste produce radicales libres.

En recién nacidos con hipoxia, una enfermedad que priva de oxígeno a una parte o al cuerpo entero, se encuentran elevadas concentraciones de estos, si en estas condiciones se le suministra oxígeno sin control, se produciría un exceso de radicales libres en el organismo del neonato que debilitaría las defensas del organismo, volviéndolo vulnerable a otras enfermedades [3].

En los 50's se demostró que el oxígeno era seguro si se suministraba en concentraciones menores al 40 % y al mismo tiempo se validó la hipótesis de que la exposición a él en exceso, era al menos una de las causas de la retinopatía del prematuro [4].

Hoy entendemos que debe ser considerado como un medicamento, esto es, debe ser dosificado y monitorizado, por sus posibles efectos adversos y/o complicaciones.

Capítulo 1

Marco teórico

Para poder trabajar con datos digitales, los cuales provienen de alguna fuente de señales analógicas, es necesario conocer el proceso que requieren tener dichas señales antes de convertirse en muestras discretas. Antes de hablar de la adquisición de datos es necesario definir primero algunos conceptos básicos que involucran los sistemas de procesamiento de señales, y el tratamiento previo del mundo analógico hacia el digital que requieren estas para optimizar el funcionamiento del sistema.

El primer bloque que aparece en la literatura [5] sobre los sistemas de procesamiento de señales, mencionan al acondicionamiento de las señales, las cuales provienen de las fuentes de información, enseguida se habla de la etapa de conversión analógica a digital que transforma o entrega una representación binaria de la señal analógica previamente acondicionada, que finalmente serán los datos discretos esperados para su procesamiento.

En el caso que nos ocupa, nos interesa adquirir, procesar y transmitir los datos provenientes de un sistema micro electromecánico, el cual contiene la información del flujo dentro de una conexión de gases para el tratamiento de enfermedades respiratorias en recién nacidos. La adquisición y procesamiento se producirá dentro del microcontrolador MSP430F2274 y la transmisión de los datos por el tranceptor CC2500, ambos incluidos en la herramienta de desarrollo eZ430-RF2500 de Texas Instruments.

Tópicos de sistemas de comunicación digitales

- Acondicionamiento de señales - En el acondicionamiento de señales [6], se le realiza un tratamiento a la señal para que ésta cubra el mayor rango de resolución en la conversión A/D en un sistema de procesamiento de señales, utilizando alguna etapa de amplificación o haciendo un corrimiento de nivel para centrar la señal, realizar

filtrado, linealizar la señal de un sensor, atenuar alguna señal que sobrepase los niveles de referencia del ADC, aislar señales que pongan en riesgo la operación de los circuitos, obteniendo el mejor resultado en la conversión.

- Adquisición de datos -Es el proceso de conversión una señal analógica a digital y comunicarla a un sistema digital para el procesamiento de los datos digitales, el convertidor analógico-digital [7][8] se encarga de hacer una representación binaria de una señal analógica, está formado por un conjunto de circuitos analógicos y digitales, incluidos: amplificadores operacionales, comparadores, contadores, circuito red R2R, registros, entre otros.

- Procesamiento de señales digitales - Es el proceso que reciben los datos de acuerdo a las tareas asignadas a los circuitos digitales por donde estos datos circulan, generalmente dentro de un procesador, microcontrolador, FPGA, o dentro de la programación, análogo a las funciones de amplificación, corrimientos, filtros, mezcla de señales, entre otras, pero en forma discreta, con todas las ventajas de los sistemas digitales [9].

- Transmisión de datos - La transmisión de datos, transmisión digital o comunicaciones digitales [10], son las distintas formas para referirse al envío de información en datos a través de una red de comunicaciones entre sistemas digitales en un bus de datos o de forma serial por medio de transmisores entre alguna red inalámbrica.

- Topologías de red - Cuando se necesita establecer una comunicación entre varios nodos, se deben formar canales que aseguren la comunicación hacia todas las terminales, estos arreglos son llamados topologías [11] y existen algunos arreglos básicos, los cuales, dependiendo del modo de comunicación y las necesidades, se puede elegir alguna topología de red en especial.

- Protocolos de comunicación inalámbricos - Los protocolos de comunicación son formatos y reglas empleados para el intercambio de datos entre sistemas de cómputo que aseguren un correcto envío de información [12]. Existe un mercado de protocolos para redes inalámbricas, diseñados para cumplir con condiciones específicas para cada aplicación, los cuales se diferencian básicamente por la capacidad de datos enviados por unidad de tiempo y el rango de cobertura de la comunicación. Además de los protocolos estándar existen protocolos propiedad de alguna empresa, estos

son llamados protocolos propietarios y se adaptan para cumplir con una necesidad específica o para explotar una característica de un producto [13].

- Banda ISM - (del inglés Industrial Scientific and Medical) son espectros de frecuencia reservados para uso no comercial de radiofrecuencia en las áreas industrial, científica y médica [14]. Los últimos años han sido testigos del surgimiento de muchos estándares inalámbricos operando en la banda ISM, estos estándares juntos con protocolos privados proveen enormes oportunidades para desarrollar un amplio rango de productos inalámbricos. Estos difieren entre sí en: la tasa de transmisión de datos, rango de cobertura de comunicación, dominio de aplicación, potencia de transmisión, consumo, entre otros.

- Punto de acceso - Es la interfaz de comunicación entra las terminales de una red inalámbrica y la computadora, un ejemplo simple de un AP (por sus siglas en ingles Access Point) es el *pendrive* de un ratón o de un teclado inalámbrico y funciona como una extensión del puerto USB vía inalámbrica, para comunicar a la computadora la información de las terminales de la red inalámbrica.

- Terminal - Es una extensión inalámbrica de un periférico de computadora, análogo al ejemplo anterior, un teclado o ratón son las terminales o EP (por sus siglas en ingles End Point) en la red.

Una vez definido los principales temas sobre los que se desarrollará este trabajo, continuaremos con la descripción de la herramienta de desarrollo utilizada para resolver la caracterización dinámica del microsensor de flujo.

Capítulo 2

Herramienta de desarrollo

Para resolver el problema de comunicación digital de la información proveniente del MEMS hacia la computadora con el fin de analizar los datos y realizar la caracterización dinámica del microsensor de flujo, se adquirió la tarjeta de desarrollo eZ430-RF2500, cuya descripción se realiza a continuación.

2.1. Kit de desarrollo

La eZ430-RF2500 [15] de Texas Instruments es una completa herramienta de desarrollo inalámbrica que incluye todo el software y hardware necesario para evaluar al microcontrolador MSP430F2274 y al transceptor CC2500 a 2.4GHz.

Figura 2.1: Kit eZ430-RF2500

2.1.1. Tarjeta eZ430-RF2500

Como ya se mencionó, se requiere adquirir, procesar y transmitir los datos de la rejilla para la caracterización respecto al flujo de gas que esté circulando a través de ella. Dichas tareas son posibles de obtener con esta tarjeta, ya que incluye un microcontrolador de señal mixta, es decir, tiene la capacidad de trabajar con señales analógicas y digitales, debido a que cuenta con terminales accesibles a un convertidor analógico digital de 10 bits dentro del microcontrolador MSP430F2274, y bien el procesamiento lo puede realizar el microcontrolador.

Además del microcontrolador, la tarjeta incluye un transceptor CC2500 a 2.4GHz para la comunicación inalámbrica, y también incluye la aplicación UART USB para comunicar a una de las tarjetas eZ430-RF2500 como punto de acceso a la computadora y de esta manera se logrará capturar en la computadora los datos que se adquieran en la tarjeta conectada al sistema de caracterización que operará como una terminal de la red.

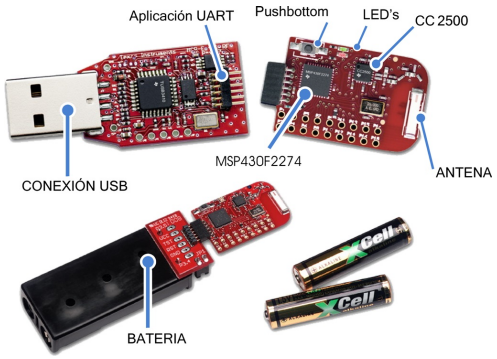


Figura 2.2: Componentes de la herramienta de desarrollo

La herramienta es capaz de operar con una batería de 3V debido a los modos de operación de bajo consumo en el microcontrolador y en el transceptor, por el protocolo de comunicación diseñado para aplicaciones de ultra bajo consumo, volviendo a la herramienta de desarrollo una aplicación portátil.

A continuación, mostraremos las principales características de operación tanto del microcontrolador como del transceptor, las cuales nos ayudarán al realizar cambios en la programación y para entender el porqué de la operación del código de demostración.

2.1.2. Parámetros eléctricos del MSP430F2274

Los siguientes parámetros del microcontrolador muestran los valores en el consumo según el modo de operación en que se opere, la temperatura A continuación podemos ver en la tabla 2.1 los principales parámetros de trabajo y consumo mínimo, típico y máximos del microcontrolador MSP430F2274 a tener en cuenta [16].

Parámetro	Mínimo	Típico	Máximo	Unidades
Voltaje de operación	1.8		3.6	V
Rango de temperatura	-40		85	°C
Consumo modo activo a 1MHz y a 2.2V		270	390	μA
Consumo en modo espera		0.7	1.4	μA
Consumo en modo apagado con retención de memoria			0.5	μA
Frecuencia de operación on VCC $\geq$ 3.3V			16	MHz

Tabla 2.1: Parámetros eléctricos del MSP430F2274

2.1.3. Parámetros del transceptor CC2500

El circuito dentro de la tarjeta eZ430-RF2500 encargado de la comunicación inalámbrica CC2500[17] entre las terminales y el punto de acceso, presenta la siguiente información de los parámetros de consumo, operación, recepción y transmisión mostrado en la tabla 2.2.

Parámetro	Condición	Mín.	Típ.	Máx.	Unidad
Voltaje de operación		1.8		3.6	
Consumo de corriente Rx, 250kbps	Corriente optimizada		16.6		mA
	Sensibilidad optimizada		18.8		mA
Consumo señal de entrda Rx 30dB sobre la sensibilidad límite 250kbps	Corriente optimizada		13.3		mA
	Sensibilidad optimizada		15.7		mA
Consumo de corriente Tx (0dBm			21.2		mA
Consumo de corriente Tx (-12dBm)			11.1		mA
Rango de frecuencia		2400		2483.5	MHz
Tasa de transferencia de datos (Programable)		1.2		500	kbps
Potencia de salida (Programable)		-30		0	dBm
Sensibilidad, 10kbps	Corriente optimizada, 2fsk, 230-kHz Rx de ancho de banda filtrado, 1% per		-99		dBm
	Sensibilidad optimizada		-101		dBm
Sensibilidad, 250kbps	Corriente optimizada, 500 kHz Rx ancho de banda filtrado 1% per		-87		dBm

Tabla 2.2: Parámetros eléctricos del transceptor CC2500

2.1.4. Instalación del software de la tarjeta de desarrollo

La herramienta incluye el entorno de desarrollo integrado *Code composer studio* para editar el código, compilar, y programar la tarjeta de desarrollo. Este programa se incluye en el disco y su instalación es amigable con el usuario. Cuando se conecta la tarjeta con la aplicación UART USB, se instalan los controladores necesarios de forma automática.

2.1.5. Demo eZ430-RF2500 Monitor

Además del compilador CCS antes mencionado, el kit incluye el programa *Sensor Monitor*, el cual despliega la interfaz gráfica que se muestra en la figura 2.3. En donde podemos ver una topología de red tipo estrella, en el que las terminales son representadas en la periferia, las terminales miden su temperatura una vez cada segundo durante 7ms y después de esto, entran en modo de bajo consumo de energía para reducir el uso de la batería, al igual que el punto de acceso central, que a su vez es el puente de comunicación con la computadora.

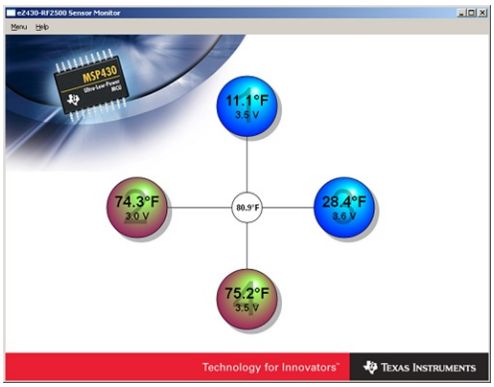


Figura 2.3: Interfaz Gráfica *Sensor monitor*

Además de mostrar la información de cada nodo de la red, los datos pueden ser desplegados en la consola del programa como se puede ver en la figura 2.4

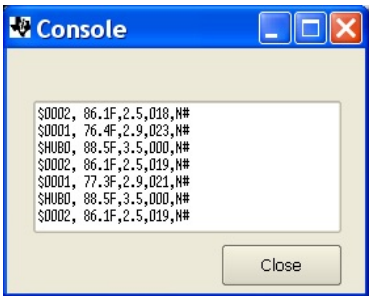


Figura 2.4: Consola de la interfaz gráfica

2.1.6. Protocolo de red SimpliciTI

SimpliciTI es un protocolo de red propietario, para interconexión simple de redes pequeñas de RF, de bajo consumo (<100 nodos) y de bajo rango de operación. El protocolo de red SimpliciTI está diseñado para implementarse fácilmente con un requerimiento de recursos del microcontrolador mínimos.

Pequeñas redes de RF de bajo consumo, contienen típicamente dispositivos que operan con baterías, lo cual requiere que la vida útil de la batería sea grande, que la tasa de transmisión de datos de la tasa de transmisión se a baja y de un corto ciclo de trabajo durante la transmisión, así como un número limitado de nodos comunicándose directamente unos con otros.

2.1.7. Instalación del código de demostración en CCS

A continuación, se detalla el proceso de instalación del entorno de desarrollo y del código fuente [18].

- El espacio de trabajo deberá ser creado en el siguiente directorio: smallskip ~/CCE_Source/Projects/Examples/peer_applications/eZ430RF\
- Ir a: "Project >Import Existing CCS/CCE Eclipse Project"
- Navegar para encontrar la carpeta eZ430-RF2500_WSM Asegúrese de tener seleccionado el proyecto eZ430-RF2500_WSM y finalice.
- Ignore el error "C/C++ indexin"
- Seleccione "Window >Preferences... >General >Workspace >Linked Resources >New" para crear una nueva variable de entorno.
- Nombre de la variable: eZ430_WSM_ROOT
- Seleccione la opción de folder y busque la ruta ~\eZ430-RF2500 Wireless Sensor Monitor\CCS_Source
- Finalmente construya y descargue el nuevo proyecto, eligiendo entre Access Point y End Point.

Ahora que se conocen los dispositivos que integran la tarjeta de desarrollo y después de repasar las características internas del microcontrolador, así como de los bloques que lo forman, podremos entender el código fuente de programación, el cual se analizará en el siguiente capítulo para reprogramarlo con las tareas que se requieran durante el proceso de caracterización dinámica de los microsensores de flujo.

Capítulo 3

Programación

3.1. Código fuente de la aplicación de demostración

El código de programación de la versión de demostración, está formado por 3 programas, uno asignado para el nodo principal que recibe la información de los nodos periféricos y se comunica directamente con la computadora a través del puerto USB, otro para la(s) terminal(es), que envían de manera inalámbrica hacia el punto de acceso la información que recibe y procesa el microcontrolador MSP430F2274 y un tercero que contiene el formato del mensaje enviado hacia el punto de acceso y es recibido por la computadora.

El código del punto de acceso, está representado en el diagrama de flujo de la figura 3.1, el programa comienza inicializando los bloques periféricos que se usarán en el microcontrolador y las tareas que realizarán, a continuación el programa envía un paquete hacia la computadora para entablar la comunicación, enseguida se inicializa la red, comenzando con la espera de comunicación con las terminales, continuando con la lectura interna de temperatura y voltaje, genera un formato para enviar los datos en un mensaje, espera los paquetes de las terminales y finalmente envía hacia la computadora la información de todo lo procesado y recibido.

Para las terminales figura 3.2, el programa difiere del código del punto de acceso por el hecho de necesitar una identificación de las terminales que quieran comunicarse con el punto de acceso, es por esto que antes de todo, se debe generar esa identificación, una vez que se generó se inicializa el microcontrolador de la misma forma que en el programa del AP, enseguida se inicia la comunicación en red con el punto de acceso, ahora se realizan las lecturas de voltaje y temperatura del dispositivo y se llena el formato para enviar el mensaje, se envían los datos e inmediatamente después de enviar el mensaje, el programa entra en modo de bajo consumo reduciendo al mínimo el consumo de energía de la tarjeta, hasta que se interrumpa el estado de bajo consumo para tomar una nueva muestra.

3.1.1. Código de programación del punto de acceso

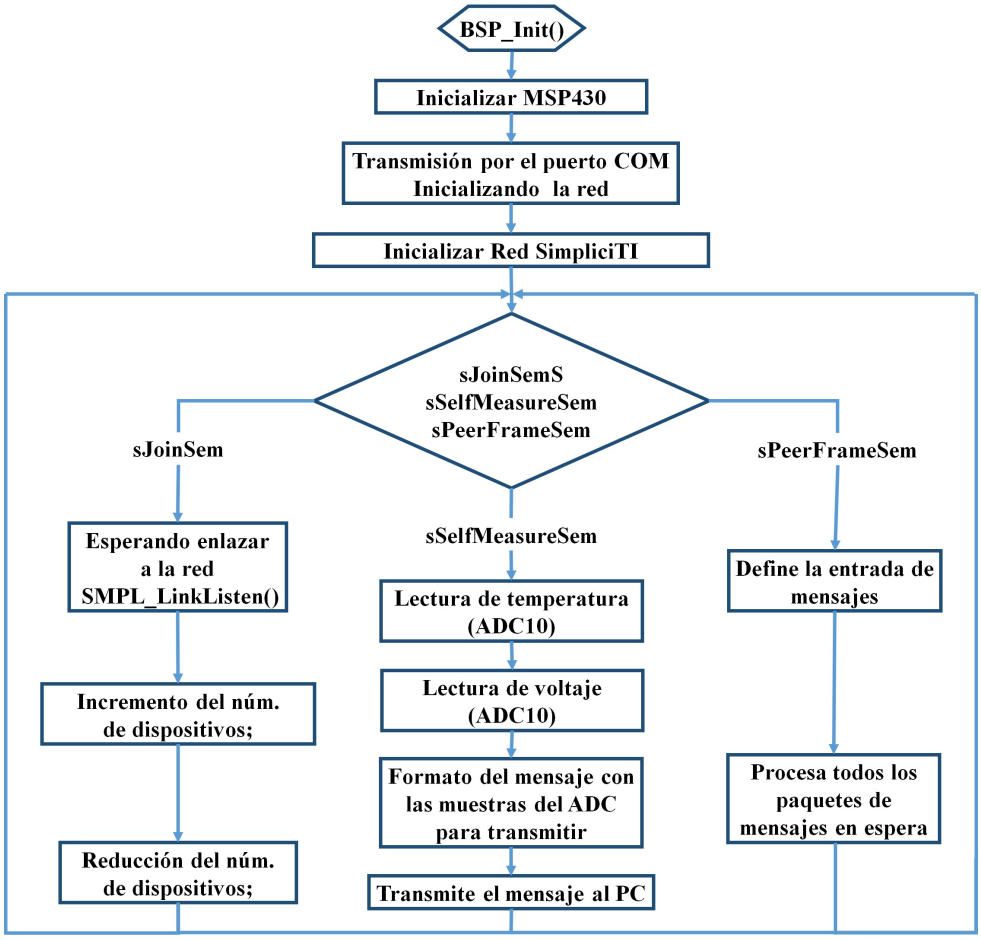


Figura 3.1: Diagrama de flujo del punto de acceso

El siguiente código muestra la sección del programa donde se obtiene la temperatura y el voltaje del microcontrolador, así como el procesamiento que se le pueden dar a los datos adquiridos por el convertidor analógico digital.

Primero se definen las variables a donde se enviarán los datos de temperatura y voltaje, se definen los puertos de entrada hacia el convertidor analógico-digital y enseguida se configura el convertidor a través de los registros definidos para ello, después se da inicio a la conversión y el resultado de la conversión que se almacena inmediatamente en el registro ADC10MEM el cual se copia a la variable definida “result[]”, posteriormente cada dato obtenido se procesa y se envía a una variable final que se comienza a empaquetar según un formato para enviarse finalmente al código del mensaje.

Ésta parte del programa del punto de acceso será eliminada para recibir solamente información de las terminales. Hasta este momento no se han usado puertos externos de la tarjeta, sin embargo, para las tareas del proyecto se modificará para tener acceso a las terminales externas de la tarjeta.

```
// if it is time to measure our own temperature...
 if(sSelfMeasureSem)
 {
   char msg [6];
   char addr[] = {"HUB0"};
   char rssi[] = {"000"};
   int degC, volt;
   volatile long temp;
   int results[2];

   ADC10CTL1 = INCH_10 + ADC10DIV_4;     // Temp Sensor ADC10CLK/5
   ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON + ADC10IE + ADC10SR;
   for( degC = 240; degC > 0; degC-- );  // delay to allow reference to settle
   ADC10CTL0 |= ENC + ADC10SC;           // Sampling and conversion start
   __bis_SR_register(CPUOFF + GIE);      // LPM0 with interrupts enabled
   results[0] = ADC10MEM;
   ADC10CTL0 &= ~ENC;

   ADC10CTL1 = INCH_11;                  // AVcc/2
   ADC10CTL0 = SREF_1 + ADC10SHT_2 + REFON + ADC10ON + ADC10IE + REF2_5V;
   for( degC = 240; degC > 0; degC-- );  // delay to allow reference to settle
   ADC10CTL0 |= ENC + ADC10SC;           // Sampling and conversion start
   __bis_SR_register(CPUOFF + GIE);      // LPM0 with interrupts enabled
   results[1] = ADC10MEM;
   ADC10CTL0 &= ~ENC;
   ADC10CTL0 &= ~(REFON + ADC10ON);      // turn off A/D to save power

   // oC = ((A10/1024)*1500mV)-986mV)*1/3.55mV = A10*423/1024 - 278
   // the temperature is transmitted as an integer where 32.1 = 321
   // hence 4230 instead of 423

   temp = results[0];
   degC = (((temp - 673) * 4230) / 1024);
   if( (*tempOffset) != 0xFFFF )
     {
       degC += (*tempOffset);
     }

   temp = results[1];
   volt = (temp*25)/512;

   msg[0] = degC&0xFF;
   msg[1] = (degC>>8)&0xFF;
   msg[2] = volt;
   transmitDataString(1, addr, rssi, msg );
   BSP_TOGGLE_LED1();
   sSelfMeasureSem = 0;
 }
```

3.1.2. Código del programa de la terminal

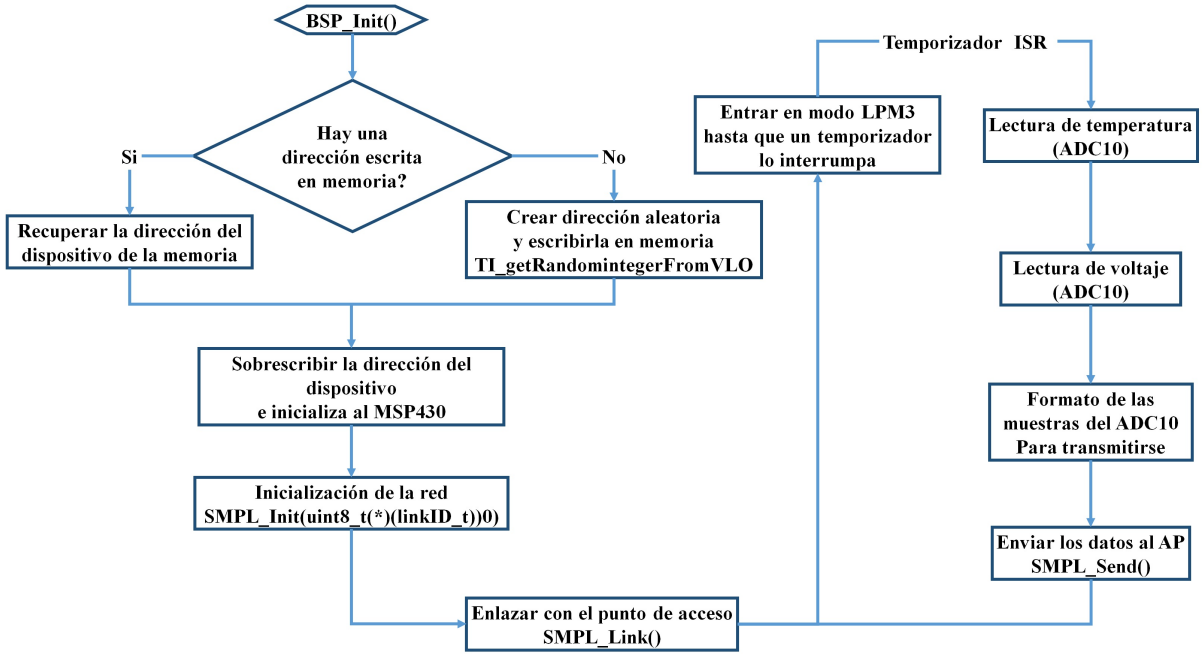


Figura 3.2: Diagrama de flujo de la terminal

Para las terminales al igual que para el punto de acceso en la versión de demostración, se toman muestras para calcular la alimentación, la cual es suministrada por baterías, y de la temperatura interna del microcontrolador.

```
/* get radio ready...awakens in idle state */
      SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_AWAKE, 0);

      ADC10CTL1 = INCH_10 + ADC10DIV_4;       // Temp Sensor ADC10CLK/5
      ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON + ADC10IE + ADC10SR;
      for( degC = 240; degC > 0; degC-- );    // delay to allow reference to settle
      ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
      __bis_SR_register(CPUOFF + GIE);        // LPM0 with interrupts enabled

      results[0] = ADC10MEM;
      ADC10CTL0 &= ~ENC;
      ADC10CTL1 = INCH_11;                     // AVcc/2
      ADC10CTL0 = SREF_1 + ADC10SHT_2 + REFON + ADC10ON + ADC10IE + REF2_5V;
      for( degC = 240; degC > 0; degC-- );    // delay to allow reference to settle
      ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
      __bis_SR_register(CPUOFF + GIE);        // LPM0 with interrupts enabled
      results[1] = ADC10MEM;
      ADC10CTL0 &= ~ENC;
      ADC10CTL0 &= ~(REFON + ADC10ON);        // turn off A/D to save power

      // oC = ((A10/1024)*1500mV)-986mV)*1/3.55mV = A10*423/1024 - 278
      // the temperature is transmitted as an integer where 32.1 = 321
```

```
      // hence 4230 instead of 423
      temp = results[0];
      degC = ((temp - 673) * 4230) / 1024;
      if( (*tempOffset) != 0xFFFF )
      {
        degC += (*tempOffset);
      }
      /* message format,  UB = upper Byte, LB = lower Byte
      -------------------------------
      |degC LB | degC UB |  volt LB |
      -------------------------------
         0         1          2
      */

      temp = results[1];
      volt = (temp*25)/512;
      msg[0] = degC&0xFF;
      msg[1] = (degC>>8)&0xFF;
      msg[2] = volt;
```

3.1.3. Código con el formato del mensaje

El siguiente código define la posición y el orden que tendrán los datos y el formato del mensaje, además de asignarle unidades a los datos procesados y agrega etiquetas para definir de qué se trata la información. En este caso se trata de la información de las terminales como la temperatura, el nivel de la batería, la potencia de la señal y si es un repetidor y su identificador.

```
void transmitDataString(char data_mode, char addr[4],char rssi[3], char msg[MESSAGE_LENGTH] )
{
  char temp_string[] = {" XX.XC"};
  int temp = msg[0] + (msg[1]<<8);

  if(!(data_mode & degCMode))
  {
    temp = (((float)temp)*1.8)+320;
    temp_string[5] = 'F';
  }
  if( temp < 0 )
  {
    temp_string[0] = '-';
    temp = temp * -1;
  }
  else if( ((temp/1000)%10) != 0 )
  {
    temp_string[0] = '0'+((temp/1000)%10);
  }
  temp_string[4] = '0'+(temp%10);
  temp_string[2] = '0'+((temp/10)%10);
  temp_string[1] = '0'+((temp/100)%10);

  if(data_mode & verboseMode)
  {
    char output_verbose[] = {"\r\nNode:XXXX,Temp:-XX.XC,Battery:X.XV,Strength:XXX%,RE:no "};

    output_verbose[46] = rssi[2];
```

```
  output_verbose[47] = rssi[1];
  output_verbose[48] = rssi[0];

  output_verbose[17] = temp_string[0];
  output_verbose[18] = temp_string[1];
  output_verbose[19] = temp_string[2];
  output_verbose[20] = temp_string[3];
  output_verbose[21] = temp_string[4];
  output_verbose[22] = temp_string[5];

  output_verbose[32] = '0'+(msg[2]/10)%10;
  output_verbose[34] = '0'+(msg[2]%10);
  output_verbose[7] = addr[0];
  output_verbose[8] = addr[1];
  output_verbose[9] = addr[2];
  output_verbose[10] = addr[3];
  TXString(output_verbose, sizeof output_verbose );
}
else
{
  char output_short[] = {"\r\n$ADDR,-XX.XC,V.C,RSI,N#"};

  output_short[19] = rssi[2];
  output_short[20] = rssi[1];
  output_short[21] = rssi[0];


  output_short[8] = temp_string[0];
  output_short[9] = temp_string[1];
  output_short[10] = temp_string[2];
  output_short[11] = temp_string[3];
  output_short[12] = temp_string[4];
  output_short[13] = temp_string[5];

  output_short[15] = '0'+(msg[2]/10)%10;
  output_short[17] = '0'+(msg[2]%10);
  output_short[3] = addr[0];
  output_short[4] = addr[1];
  output_short[5] = addr[2];
  output_short[6] = addr[3];
  TXString(output_short, sizeof output_short );
}
```

3.2. Microcontroladores

Son sistemas embebidos [15] construidos alrededor de una unidad central de procesamiento. Éstos sistemas se conforman por periféricos que pueden ofrecer funciones analógicas y digitales tales como: temporizadores, convertidores, amplificadores, módulos de comunicación, memoria volátil y fija etc. El microcontrolador realiza instrucciones almacenadas en localidades de memoria, dichas instrucciones obedecen a un programa el cual es realizado en algún lenguaje de bajo o alto nivel, usando algún procesador de código base, enseguida es compilado, depurado y almacenado en lenguaje máquina por un circuito programador.

3.2.1. MSP430F2274

La familia de microcontroladores MSP430 de Texas Instruments de bajo consumo, cuenta con varios dispositivos con características diferentes para diversas aplicaciones. Su arquitectura combinada con 5 modos de ultra bajo consumo de potencia es optimizada para lograr extender el periodo de vida útil de la batería en dispositivos de medición portátiles. Las características del dispositivo son una unidad de procesamiento de 16 bits tipo RISC, registros de 16 bits, 4 registros dedicados, y una respuesta a modo activo de 1μs cuando está en bajo consumo.

La serie MSP430x22xx es un microcontrolador de ultra bajo consumo construido con: 2 temporizadores, una interfaz de comunicación serial universal, convertidor A/D de 10 bits con voltaje de referencia integrado y un controlador de transferencia de datos, dos amplificadores de propósito general configurables, y 32 pines de entrada y salida.

La figura 3.3 muestra el diagrama a bloques funcional del microcontrolador MSP430F2274.

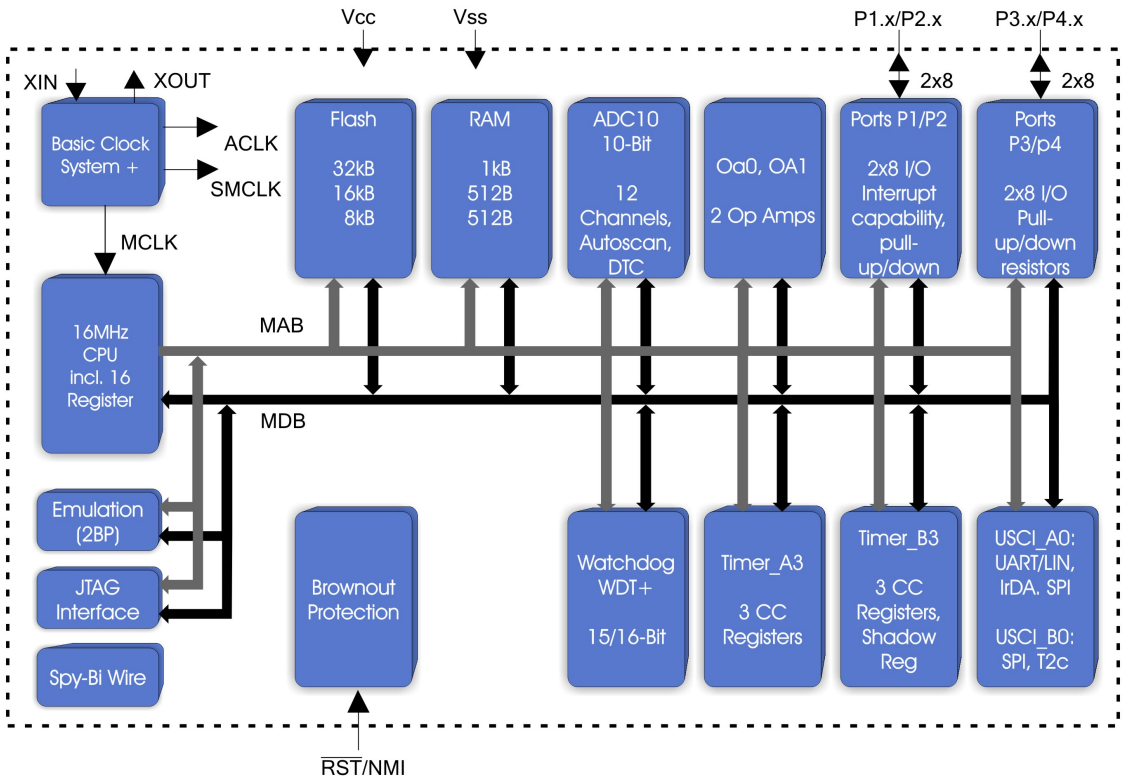


Figura 3.3: Diagrama a bloques funcional del microcontrolador MSP43022F74

3.2.2. Arquitectura del microcontrolador MSP430F2274

La CPU del MSP430 cuenta con una arquitectura tipo RISC de 16-bits que es altamente transparente a la aplicación. Todas las operaciones, salvo las instrucciones de flujo de programa, se realizan como operaciones de registro junto con siete modos de direccionamiento para el operador de origen y cuatro modos de direccionamiento para el operador de destino. La CPU está integrada con 16 registros que proporcionan una reducción del tiempo de ejecución de las instrucciones. El tiempo de ejecución de la operación registro a registro, es un ciclo de reloj de la CPU. Cuatro de los registros, R0 a R3, son dedicados como contador de programa, apuntador de pila, registro de estado, y generador de constante respectivamente. Los registros restantes son de propósito general. Los periféricos se interconectan a la CPU a través de los buses de control, direcciones y datos y pueden ser manejados con todas las instrucciones.

3.2.3. Variables definidas

Para reconocer las definiciones de los registros y bits estándar para el microcontrolador existe el código *MSP430x22x4.h* que se incluye en la compilación del programa y que describe con claridad cómo están definidos los registros en el código de demostración.

Es necesario el repaso debido a que solo se manejan los nombres de las variables, pero no así el valor que ocupan dentro de los registros de las instrucciones además que el nombramiento de variables es arbitrario y se debe conocer cada variable que aparezca involucrada dentro de la programación.

```
/************************************************************************/
/* Legacy Header File                                                   */
/* Not recommended for use in new projects.                             */
/* Please use the msp430.h file or the device specific header file      */
/************************************************************************/

/************************************************************
*
* Standard register and bit definitions for the Texas Instruments
* MSP430 microcontroller.
*
* This file supports assembler and C development for
* MSP430x22x4 devices.
*
* Texas Instruments, Version 1.5
*
/* ADC10CTL0 */
#define ADC10SC                 (0x001)        /* ADC10 Start Conversion */
#define ENC                     (0x002)        /* ADC10 Enable Conversion */
#define ADC10IFG                (0x004)        /* ADC10 Interrupt Flag */
```

```
#define ADC10IE                 (0x008)         /* ADC10 Interrupt Enalbe */
#define ADC10ON                 (0x010)         /* ADC10 On/Enable */
#define REFON                   (0x020)         /* ADC10 Reference on */
#define REF2_5V                 (0x040)         /* ADC10 Ref 0:1.5V / 1:2.5V */
```

3.2.4. Convertidor analógico digital

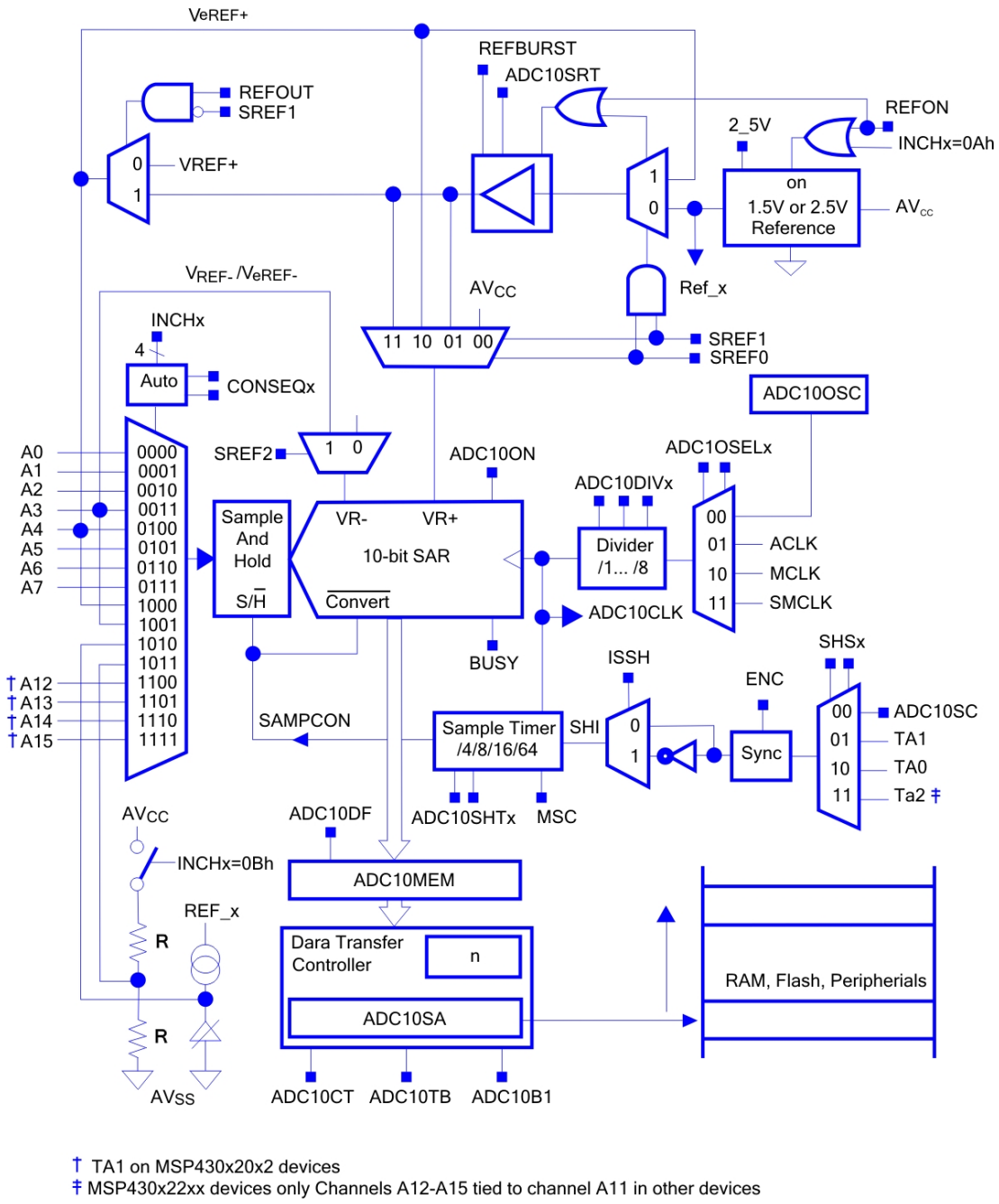
El módulo ADC10 es un convertidor analógico digital de 10 bits. El módulo consiste de un convertidor por aproximación sucesiva con control de transferencia de datos, voltajes de referencia y control de selección de datos.

El control de transferencia de datos permite a la muestra ser convertida y almacenada en la memoria sin la intervención de la unidad central de procesamiento El módulo puede ser configurado para ejecutar una variedad de tareas.

Características:

- Tasa de conversión máxima 200-ksps
- Monótono sin pérdida de código
- Muestreo y retención con periodos de muestra programable
- Inicio de muestreo por software o temporizador
- Voltajes de referencia en chip programable por software (1.5 V o 2.5 V)
- Selección de voltaje de referencia externo o interno
- 12 canales de entrada externos
- Canales internos para el sensor de temperatura y Vcc

El corazón del ADC (Figura 3.4) convierte una señal de entrada analógica en una representación digital de 10 bits y almacena el resultado en el registro ADC10MEM. Usa dos voltajes seleccionables de referencia (V_{R+} y V_{R-}) para definir los límites de conversión. La salida digital (N_{ADC}) cuando alcanza el límite superior V_{R+} es (03FFh) y cero cuando la señal de entrada es igual al voltaje V_{R-}



† TA1 on MSP430x20x2 devices
‡ MSP430x22xx devices only Channels A12-A15 tied to channel A11 in other devices

Figura 3.4: Bloques que pertenecen al convertidor analógico digital ADC10

La ecuación de conversión de ADC es:

$$N_{ADC} = 1023 * \frac{V_{in} - V_{R-}}{V_{R+} - V_{R-}} \tag{3.1}$$

El núcleo del ADC10 configurado por 2 registros de control ADCTL0 (Tabla 9) y ADCTL1 (Tabla 10). Y es habilitado con el bit ADC10ON. Bajo muy pocas excepciones los bits de control pueden ser modificados cuando ENC=0. ENC debe estar en uno lógico antes de que cualquier conversión pueda realizarse.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SREFx			ADC10SHT		ADC10SR	REFOUT	REFBURST	MSC	REF2_5V	REFON	ADC10ON	ADC10IE	ADC10IFG	ENC	ADC10SC

Tabla 3.1: Registro de control ADCTL0

SREFx	Bits 15-13	Select reference.
	000	VR+ = VCC and VR- = VSS
	001	VR+ = VREF+ and VR- = VSS
	010	VR+ = VeREF+ and VR- = VSS. Devices with VeREF+ only.
	011	VR+ = Buffered VeREF+ and VR- = VSS. Devices with VeREF+ pin only.
	100	VR+ = VCC and VR- = VREF-/ VeREF-. Devices with VeREF- pin only.
	101	VR+ = VREF+ and VR- = VREF-/ VeREF-. Devices with VeREF+/- pin only.
	110	VR+ = VeREF+ and VR- = VREF-/ VeREF-. Devices with VeREF+/- pin only.
	111	VR+ = Buffered VeREF+ and VR- = VREF-/ VeREF-. Devices with VeREF+/- pin only.
ADC10SHTx	Bits 12-11	ADC10 sample-and-hold time
	00	4 * ADC10CLKs
	01	8 * ADC10CLKs
	10	16 * ADC10CLKs
	11	64 * ADC10CLKs
ADC10SR	Bit 10	ADC10 sampling rate. This bit selects the reference buffer drive capability for the maximum sampling rate. Setting ADC10SR reduces the current consumption of the reference buffer.
	0	Reference buffer supports up to ~200 ksps
	1	Reference buffer supports up to ~50 ksps
REFOUT	Bit 9	Reference output
	0	Reference output off
	1	Reference output on. Devices with VeREF+ / VREF+ pin only.
REFBURST	Bit 8	Reference burst.
	0	Reference buffer on continuously
	1	Reference buffer on only during sample-and-conversion
MSC	Bit 7	Multiple sample and conversion. Valid only for sequence or repeated modes.
	0	The sampling requires a rising edge of the SHI signal to trigger each sample-and-conversion.
	1	The first rising edge of the SHI signal triggers the sampling timer, but further sample-and-conversions are performed automatically as soon as the prior conversion is completed
REF2_5V	Bit 6	Reference-generator voltage. REFON must also be set.
	0	1.5 V
	1	2.5 V
REFON	Bit 5	Reference generator on
	0	Reference off
	1	Reference on
ADC10ON	Bit 4	ADC10 on
	0	ADC10 off
	1	ADC10 on
ADC10IE	Bit 3	ADC10 interrupt enable
	0	Interrupt disabled
	1	Interrupt enabled
ADC10IFG	Bit 2	ADC10 interrupt flag. This bit is set if ADC10MEM is loaded with a conversion result. It is automatically reset when the interrupt request is accepted, or it may be reset by software. When using the DTC this flag is set when a block of transfers is completed.
	0	No interrupt pending
	1	Interrupt pending
ENC	Bit 1	Enable conversion
	0	ADC10 disabled
	1	ADC10 enabled
ADC10SC	Bit 0	Start conversion. Software-controlled sample-and-conversion start. ADC10SC and ENC may be set together with one instruction. ADC10SC is reset automatically.
	0	No sample-and-conversion start
	1	Start sample-and-conversion
	}	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SREFx			ADC10SHT		ADC10SR	REFOUT	REFBRURST	MSC	REF2_5V	REFON	ADC10ON	AD10IE	ADC10FG	ENC	ADC10SC

Tabla 3.2: Registro de control ADCTL1

INCHx	Bits 5-12	Input channel select. These bits select the channel for a single-conversion or the highest channel for a sequence of conversions. Only available ADC channels should be selected. See device specific datasheet.
	0000	A0
	0001	A1
	0010	A2
	0011	A3
	0100	A4
	0101	A5
	0110	A6
	0111	A7
	1000	VeREF+
	1001	VREF-/VeREF-
	1010	Temperature sensor
	1011	(VCC - VSS) / 2
	1100	(VCC - VSS) / 2, A12 on MSP430F22xx devices
	1101	(VCC - VSS) / 2, A13 on MSP430F22xx devices
	1110	(VCC - VSS) / 2, A14 on MSP430F22xx devices
	1111	(VCC - VSS) / 2, A15 on MSP430F22xx devices
SHSx	Bits 11-10	Sample-and-hold source select.
	00	ADC10SC bit
	01	Timer_A.OUT1
	10	Timer_A.OUT0
	11	Timer_A.OUT2 (Timer_A.OUT1 on MSP430F20x0, MSP430G2x31, and MSP430G2x30 devices)
ADC10DF	Bit 9	ADC10 data format
	0	Straight binary
	1	2s complement
ISSH	Bit 8	Invert signal sample-and-hold
	0	The sample-input signal is not inverted.
	1	The sample-input signal is inverted.
ADC10DIVx	Bits 7-5	ADC10 clock divider
	000	/1
	001	/2
	010	/3
	011	/4
	100	/5
	101	/6
	110	/7
	111	/8
ADC10SSELx	Bits 4-3	ADC10 clock source select
	00	ADC10OSC
	01	ACLK
	10	MCLK
	11	SMCLK
CONSEQx	Bits 2-1	Conversion sequence mode select
	00	Single-channel-single-conversion
	01	Sequence-of-channels
	10	Repeat-single-channel
	11	Repeat-sequence-of-channels
ADC10BUSY	Bit 0	ADC10 busy. This bit indicates an active sample or conversion operation
	0	No operation is active.
	1	A sequence, sample, or conversion is active.

Ejemplo del uso de los registros ADCTL0 y ADCTL1 en el Convertidor analógico digital.

```
ADC10CTL0 = ADC10SHT_2 + ADC10ON + ADC10IE; // ADC10ON, interrupt enabled
ADC10CTL1 = INCH_1;                         // input A1
```

3.3. Programación de los puertos de entrada

El firmware de la eZ430-RF2500 no hace uso de los puertos de entrada, ya que tanto el sensor de temperatura como el voltaje de referencia que se usan en la demostración,

se encuentran dentro del MSP430F2274, es por esto que se deberán de tener en cuenta las instrucciones necesarias, para habilitar los puertos que manejen señales analógicas y de esta manera se puedan adquirir las señales de forma externas a través de los pines asignados (Figura 3.5).

La configuración de los puertos para adquirir una señal analógica, viene implícita en los registros de control al acceder a través del convertidor, debido a la conexión directa que hay en las terminales y el multiplexor del ADC.

3.3.1. Código de ejemplo del bloque ADC10

```
WDTCTL = WDTPW + WDTHOLD;                    // Stop WDT
ADC10CTL0 = ADC10SHT_2 + ADC10ON + ADC10IE; // ADC10ON, interrupt enabled
ADC10CTL1 = INCH_1;                          // input A1
ADC10AE0 |= 0x02;                            // PA.1 ADC option select
P1DIR |= 0x01;                               // Set P1.0 to output direction
```

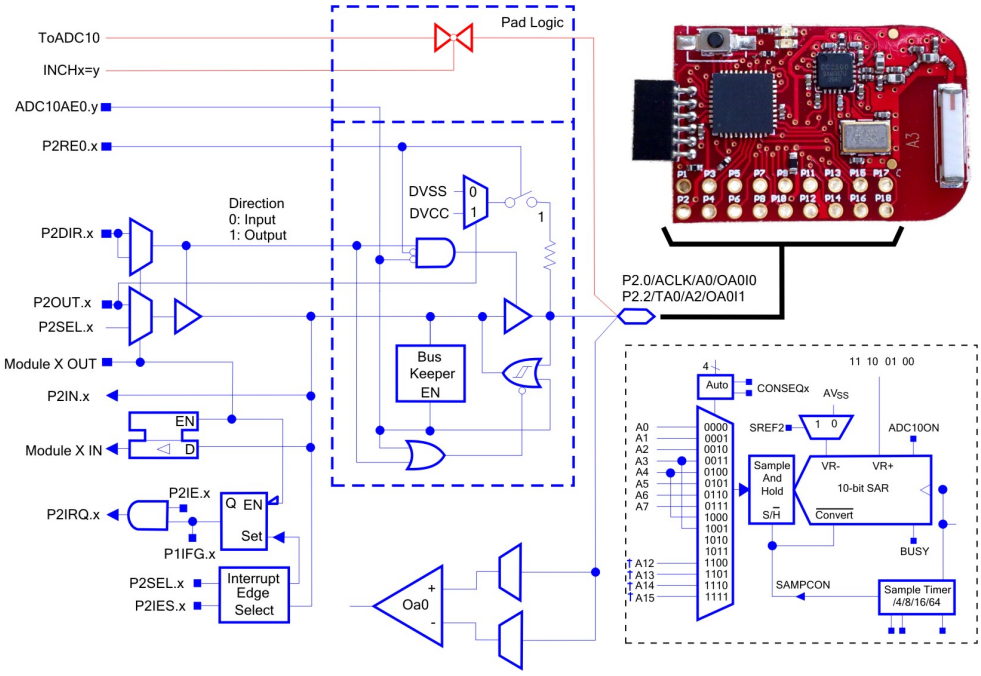


Figura 3.5: Diagrama esquemático de los puertos de entrada salida del MSP43022F74

3.4. Análisis del código fuente de la terminal

Es necesario hacer un estudio de las instrucciones contenidas en el código del programa original y comprender las tareas que se están ejecutando en cada bloque del microcontro-

lador, para poder realizar las modificaciones que creamos necesarias a partir del código preinstalado para modificarlo y realizar otras tareas.

Ya se analizó el código de los principales bloques que se utilizarán para realizar la tarea de adquisición procesamiento y transmisión del transceptor CC2500 a partir del microcontrolador MSP430F2274.

Ahora enfocaremos la atención al cuerpo del programa donde se toma la señal interna del sensor de temperatura y como se lleva a cabo el proceso de conversión de la señal analógica a digital y el procesamiento que recibe esta señal digital para obtener finalmente los datos que serán enviados dentro de un mensaje hacia el transceptor.

A continuación, se describen paso a paso las líneas de código del bloque principal del programa

El objetivo principal del programa, es el de obtener muestras del sensor de temperatura y cada segundo bastarían para nuestra aplicación y registrar el nivel de voltaje en la batería que alimenta al sistema de adquisición y de la tarjeta de acondicionamiento.

La señal del sensor de temperatura está definida por la función de transferencia que se muestra en la figura 3.6 contenida en las especificaciones del microcontrolador.

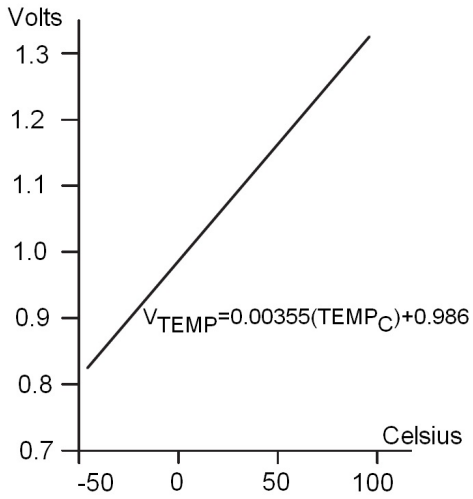


Figura 3.6: Curva de transferencia del sensor de temperatura interno

3.4.1. Análisis del código del procesamiento de la señal

Estudiando la ecuación que describe el comportamiento de la temperatura del sensor interno en el microcontrolador, se puede procesar los datos y obtener los valores que deseemos para cualquier otra curva de otro sensor o fuente de señal.

A continuación, se analiza el código para calcular la temperatura del microcontrolador.

```
degC= ((temp - 673) * 4230) / 1024;
```

Partiendo con la función de transferencia del sensor de temperatura del microcontrolador

$$\mathtt{V_{TEMP}} = 0{,}00355 * (\mathtt{TEMP_C}) + 0{,}986\mathtt{V} \tag{3.2}$$

Tenemos que para 0 $°C$ el $\mathtt{V_{TEMP}}$ es el siguiente

$$\mathtt{V_{TEMP}} = 0{,}00355 * (0) + 0{,}986\mathtt{V} = 0{,}986\mathtt{V} \tag{3.3}$$

En este caso, utilizando los voltajes de referencia internos del ADC10 que mejor se ajustan al rango de voltaje del sensor de temperatura del microcontrolador tenemos:

$V_{REF+} = 1{,}5V V_{REF-} = 0V$

Despejando de la ecuación (3.2) $\mathtt{TEMP_C}$ con $\mathtt{V_{TEMP}} = 1.5\mathrm{V}$, el cual es el mayor voltaje para conversión según el V_{REF+}, obtenemos una temperatura de:

$$\mathtt{TEMP_C} = \frac{1{,}5\mathtt{V} - 0{,}986\mathtt{V}}{0{,}00355} = 144{,}8\mathtt{°C} \tag{3.4}$$

y para $\mathtt{V_{TEMP}} = 0\mathrm{V}$

$$\mathtt{TEMP_C} = \frac{0\mathtt{V} - 0{,}986\mathtt{V}}{0{,}00355} = -277{,}7\mathtt{°C} \tag{3.5}$$

Nota: La temperatura que puede alcanzar el microcontrolador sin comprometer la operación por causar daños a la programación va desde -55 $°C$ hasta 105 $°C$

Para obtener el valor binario de la señal de entrada del ADC10 respecto a los voltajes de referencia usamos la fórmula de conversión.

$$N_{ADC} = 1023 * \frac{V_{in} - V_{R-}}{V_{R+} - V_{R-}} \tag{3.6}$$

Calculando dos puntos para V_{in}= 0.986V y para V_{in}= 1.5V obtenemos:

$$N_{ADC} = 1023 * \frac{0{,}986V - 0V}{1{,}5V - 0V} = 672 \tag{3.7}$$

$$N_{ADC} = 1023 * \frac{1{,}5V - 0V}{1{,}5V - 0V} = 1023 \tag{3.8}$$

V_{TEMP}	N_{ADC}	$TEMP_C$
0.986V	672	0 $°C$
1.5V	1023	144.8 $°C$

Tabla 3.3: Voltaje, conversión digital y valor de temperatura del μControlador

Como queremos procesar los datos adquiridos en el microcontrolador, tenemos que trabajar con los valores que arroje la conversión N_{ADC}, enviada al registro ADC10MEM donde se cargará temporalmente la actual o última conversión.

A continuación, tomaremos 2 puntos de la recta de temperatura contra N_{ADC} para obtener la función de transferencia la cual debe coincidir con la función que encontramos en el programa de demostración.

N_{ADC}	$TEMP_C$
x_1	y_1
672	0
x_2	y_2
1023	1448

Tabla 3.4: Valores para obtener la función de transferencia de temperatura.

Para no manejar datos con punto decimal multiplicamos la temperatura 144.8 $°C$ por 10 obteniendo 1448.

Ahora con los datos anteriores obtenemos la pendiente de la recta.

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{1448 - 0}{1023 - 672} = 4{,}125 \tag{3.9}$$

De la ecuación de la recta que pasa por 2 puntos tenemos que

$$(Y - y_1) = m(X - x_1) \tag{3.10}$$

Despejando Y obtenemos:

$$Y = m(X - x_1) + y_1 \tag{3.11}$$

Sustituyendo valores:

$$Temperatura = 4{,}125(\mathtt{N_{ADC}} - 672) + 0 \tag{3.12}$$

Volvemos a trabajar con enteros, multiplicando y dividiendo a la pendiente por 1024:

$$Temperatura = (\mathtt{N_{ADC}} - 672) * 4224/1024 \tag{3.13}$$

Llegando a la misma solución que del código de demostración.

```
degC= ((temp - 673) * 4230) / 1024;
```

Para 0 °C

Temp = 672 bits

Sustituyendo el valor decimal correspondiente a 0 °C que es 672, en la fórmula de procesamiento debemos recuperar la misma temperatura

$$degC = \frac{(672 - 672) * 4224}{1024} = 0 \tag{3.14}$$

Lo mismo cuando la señal de entrada alcance el valor de referencia positivo 1023 obtenemos Sustituyendo.

$$degC = \frac{(1023 - 672) * 4224}{1024} = 1447{,}87 \tag{3.15}$$

Cuyos valores coinciden con los de la tabla 3.3
XXX.XC cumpliendo con el formato para le temperatura el dato que se envía es 144.7C

3.4.2. Formato para asignar posición a cada bit

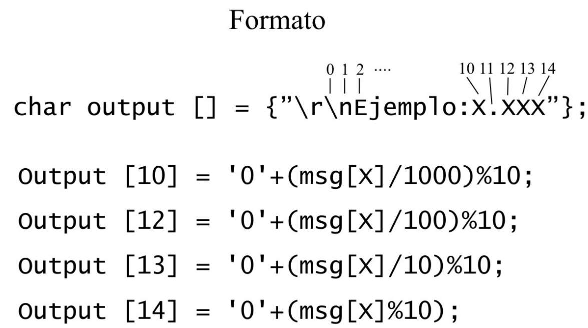


Figura 3.7: Formato del mensaje enviado por la tarjeta ez430-RF2500

Se puede ver que el punto decimal solo es una etiqueta y la posición realmente le asigna el valor a cada número.

Dentro del código para las terminales se asigna el mensaje que solo es llamado en el formato del mensaje de hecho, si se modifica el formato solo se debe programar el punto de acceso, ya que el mensaje es comunicado en código ASCCI, se debe indicar que los numero empiezan desde cero, por eso es que en el mensaje se asigna '0' sumado con el propio mensaje y la posición correspondiente, de otro modo enviaría datos erróneos.

Es por eso que aparece el cero, ya que si el número en binario empezaría desde cero que en código ASCCI tomaría el valor correspondiente en dicho código. De la misma manera hacemos el análisis para el voltaje de la batería.

```
/* Get temperature */
ADC10CTL1 = INCH_10 + ADC10DIV_4;       // Temp Sensor ADC10CLK/5
ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON + ADC10IE + ADC10SR;
/* Allow ref voltage to settle for at least 30us (30us * 8MHz = 240 cycles)
 * See SLAS504D for settling time spec
 */
__delay_cycles(240);
ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
__bis_SR_register(CPUOFF + GIE);        // LPM0 with interrupts enabled
results[0] = ADC10MEM;                  // Retrieve result
ADC10CTL0 &= ~ENC;

/* Get voltage */
ADC10CTL1 = INCH_11;                     // AVcc/2
ADC10CTL0 = SREF_1 + ADC10SHT_2 + REFON + ADC10ON + ADC10IE + REF2_5V;
__delay_cycles(240);
ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
__bis_SR_register(CPUOFF + GIE);        // LPM0 with interrupts enabled
results[1] = ADC10MEM;                  // Retrieve result

/* Stop and turn off ADC */
ADC10CTL0 &= ~ENC;
ADC10CTL0 &= ~(REFON + ADC10ON);
```

```
/* oC = ((A10/1024)*1500mV)-986mV)*1/3.55mV = A10*423/1024 - 278
 * the temperature is transmitted as an integer where 32.1 = 321
 * hence 4230 instead of 423
 */
temp = results[0];
degC = ((temp - 673) * 4230) / 1024;
if( (*tempOffset) != 0xFFFF )
{
  degC += (*tempOffset);
}

/* message format,  UB = upper Byte, LB = lower Byte
-------------------------------
|degC LB | degC UB |  volt LB |
-------------------------------
   0         1          2
*/
temp = results[1];
volt = (temp*25)/512;
msg[0] = degC&0xFF;
msg[1] = (degC>>8)&0xFF;
msg[2] = volt;
```

3.4.3. Formato del mensaje enviado

En el archivo virtual_com_cmds.c, se define el mensaje que se va a enviar, el cual está formado por una etiqueta y el valor esperado. Es primordial poder entender como modificarlo para asociarle el valor adecuado y personalizarlo.

El mensaje es transmitido y depende de la longitud de éste, en la velocidad de comunicación ya que entre más largo sea, más tiempo tomará transmitir la siguiente muestra.

La siguiente línea muestra el mensaje original de la aplicación del sensor de temperatura.

```
char output_verbose[] = {"\r\nNode:XXXX,Temp:-XX.XC,Battery:X.XV,Strength:XXX%"};
```

Junto a esta línea se define el valor que tomará cada mensaje dependiendo de su posición, es por eso que, dentro de los corchetes, aparece un número que pertenece a la posición dentro del mensaje.

El siguiente código pertenece al formato de demostración.

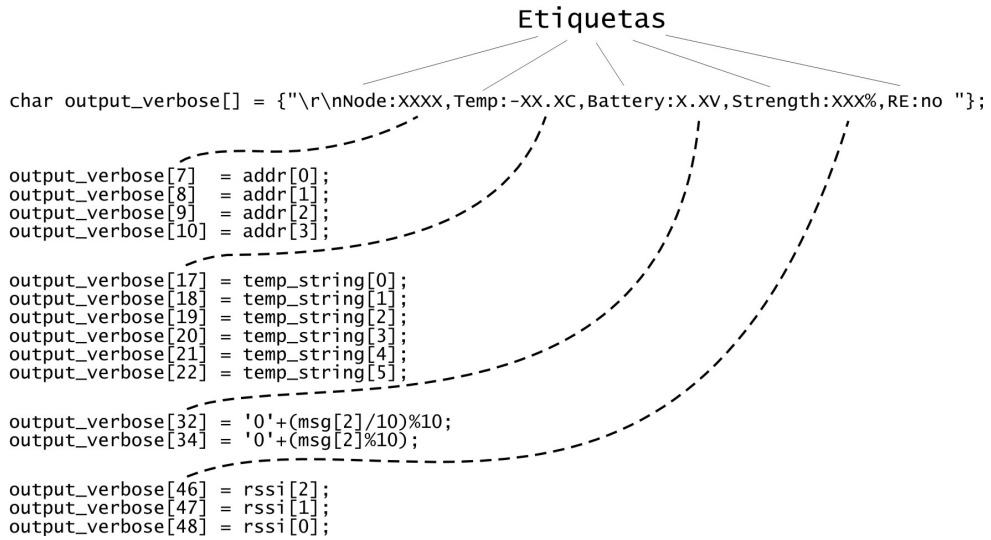


Figura 3.8: Formato del programa para el mensaje

Y el mensaje recibido en la consola aparece de la siguiente manera

```
    Node:HUB0,Temp: 88.5F,Battery:3.5V,Strength:000%,RE:no
```

Enseguida se muestran las modificaciones que se le pueden hacer al código y como llega el mensaje a la consola

```
char output_verbose[] = {"\r\nTerminal:XXXX  Sensor:X.XXXV  Vdd:X.XV  Humedad:XX%"};

    output_verbose[11] = addr[0];
    output_verbose[12] = addr[1];
    output_verbose[13] = addr[2];
    output_verbose[14] = addr[3];

    output_verbose[24] = temp_string[0];
    output_verbose[26] = temp_string[2];
    output_verbose[27] = temp_string[3];
    output_verbose[28] = temp_string[4];

    output_verbose[36] = '0'+(msg[2]/10)%10;
    output_verbose[38] = '0'+(msg[2]%10);

    output_verbose[50] = '0'+(msg[3]/10)%10;
    output_verbose[51] = '0'+(msg[3]%10);
```

3.4.4. Formato de mensaje modificado

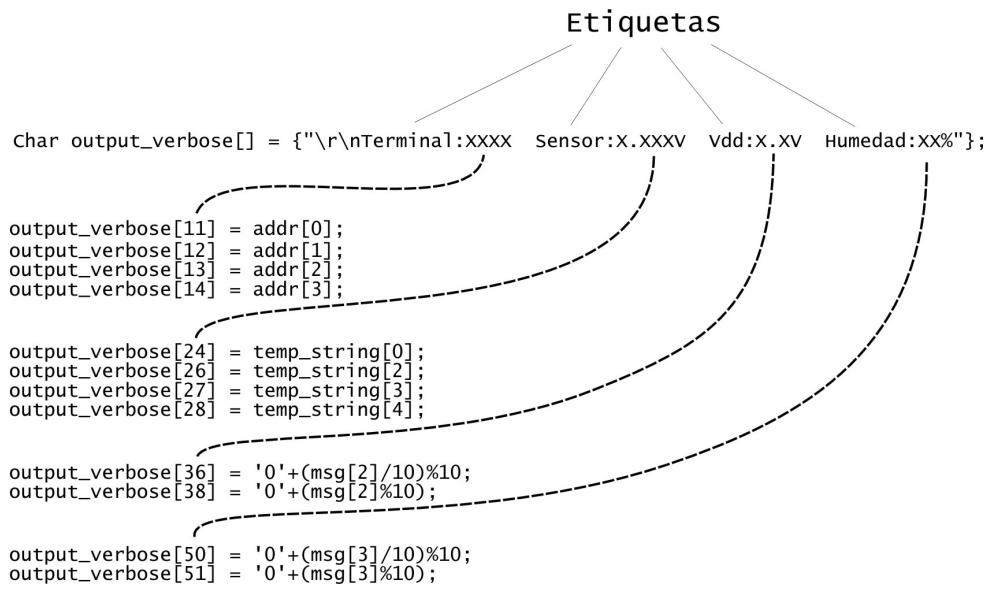


Figura 3.9: Formato del programa para el mensaje modificado

```
    Terminal:0001  Sensor:2.572V  Vdd:2.8V  Humedad:48%
```

3.4.5. Modificación para el número de transmisiones

Con el propósito de mantener un bajo consumo de energía, la aplicación del eZ430-RF2500 transmite muestras cada segundo, sin embargo, si se desea tomar muestras en un rango diferente de tiempo, se debe modificar la frecuencia de la toma de muestras para poder reconstruir de manera correcta la señal que se encuentre bajo prueba.

```
/* Initialize TimerA and oscillator */

  BCSCTL3 |= LFXT1S_2;                      // LFXT1 = VLO
  TACCTL0 = CCIE;                           // TACCR0 interrupt enabled
  TACCR0 = 12000;                           // ~1 second
  TACTL = TASSEL_1 + MC_1;                  // ACLK, upmode
```

La magnitud del registro TACCR0 define el tiempo que existe entre cada transmisión, ya que se ejecuta el conteo ascendente en el temporizador y cuanto éste llega al valor de TACCR0 el conteo comienza en 0 nuevamente y ocurre una nueva transmisión.

3.5. Programación con los cambios realizados

En la documentación de la aplicación de demostración (SLA378), de la herramienta de desarrollo Ez430-RF2500, se explica de forma detallada el código para el muestreo de la temperatura y alimentación

Los siguientes códigos pertenecen a los cambios realizados para el envío de 3 señales externas, a través de las terminales P3, P4, P5 de la herramienta de desarrollo eZ430-RF2500.

3.5.1. Punto de acceso

El siguiente código corresponde al propio mensaje del punto de acceso, el cual fue deshabilitado para recibir únicamente la información de las terminales sin desplegar su propia información ya que no es práctico conectar señales externas.

main_AP.c

```
// if it is time to measure our own temperature...
    if(sSelfMeasureSem)
    {
      char msg [6];
      char addr[] = {"HUB0"};
      char rssi[] = {"000"};
      int degC, volt;
      volatile long temp;
      int results[2];

      ADC10CTL1 = INCH_10 + ADC10DIV_4;     // Temp Sensor ADC10CLK/5
      ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON + ADC10IE + ADC10SR;
      for( degC = 240; degC > 0; degC-- );  // delay to allow reference to settle
      ADC10CTL0 |= ENC + ADC10SC;           // Sampling and conversion start
      __bis_SR_register(CPUOFF + GIE);      // LPM0 with interrupts enabled
      results[0] = ADC10MEM;

      ADC10CTL0 &= ~ENC;

      ADC10CTL1 = INCH_11;                  // AVcc/2
      ADC10CTL0 = SREF_1 + ADC10SHT_2 + REFON + ADC10ON + ADC10IE + REF2_5V;
      for( degC = 240; degC > 0; degC-- );  // delay to allow reference to settle
      ADC10CTL0 |= ENC + ADC10SC;           // Sampling and conversion start
      __bis_SR_register(CPUOFF + GIE);      // LPM0 with interrupts enabled
      results[1] = ADC10MEM;
      ADC10CTL0 &= ~ENC;
      ADC10CTL0 &= ~(REFON + ADC10ON);      // turn off A/D to save power

      // oC = ((A10/1024)*1500mV)-986mV)*1/3.55mV = A10*423/1024 - 278
      // the temperature is transmitted as an integer where 32.1 = 321
      // hence 4230 instead of 423
      temp = results[0];
      degC = (((temp - 673) * 4230) / 1024);
      if( (*tempOffset) != 0xFFFF )
      {
        degC += (*tempOffset);
      }
```

```
      temp = results[1];
      volt = (temp*25)/512;

      msg[0] = degC&0xFF;
      msg[1] = (degC>>8)&0xFF;
      msg[2] = volt;
      transmitDataString(1, addr, rssi, msg );
      BSP_TOGGLE_LED1();
      sSelfMeasureSem = 0;
    }

    /* Have we received a frame on one of the ED connections?
     * No critical section -- it doesn't really matter much if we miss a poll */
```

3.5.2. Terminal

El código de las terminales fue modificado conservando activo el canal 11 para medir el voltaje de alimentación, añadiendo el canal 1 y 2 para señales externas

main_ED.c

```
      /* :) :) :) Comienza el proceso de muestreo */

ADC10CTL1 = INCH_11;                       // Primer registro de control para el ADC
ADC10CTL0 = SREF_1 + ADC10SHT_2 + REFON + ADC10ON + ADC10IE + REF2_5V; //Segundo registro
for( degC = 240; degC > 0; degC-- );    // Retraso para el establecimiento de la referencia
ADC10CTL0 |= ENC + ADC10SC;             // Conversion y muestreo
__bis_SR_register(CPUOFF + GIE);        // Modo LPM0 con interrupciones habilitadas

results[1] = ADC10MEM;

ADC10CTL0 &= ~ENC;
ADC10CTL0 &= ~(REFON + ADC10ON);        // Apagado del ADC para reducir consumo

ADC10CTL1 = INCH_1;                     // Primer registro de control para el ADC
ADC10CTL0 = SREF_1 + ADC10SHT_2 + REFON + ADC10ON + ADC10IE + REF2_5V;//Segundo registro
for( degC = 240; degC > 0; degC-- );    // Retraso para el establecimiento de la referencia
ADC10CTL0 |= ENC + ADC10SC;             // Conversion y muestreo
__bis_SR_register(CPUOFF + GIE);        // Modo LPM0 con interrupciones habilitadas

results[0] = ADC10MEM;

ADC10CTL0 &= ~ENC;
ADC10CTL0 &= ~(REFON + ADC10ON);        // Apagado del ADC para reducir consumo
ADC10CTL1 = INCH_2;                     // Primer registro de control para el ADC
ADC10CTL0 = SREF_1 + ADC10SHT_2 + REFON + ADC10ON + ADC10IE + REF2_5V; //Segundo registro
for( degC = 240; degC > 0; degC-- );    // Retraso para el establecimiento de la referencia
ADC10CTL0 |= ENC + ADC10SC;             // Conversion y muestreo
__bis_SR_register(CPUOFF + GIE);        // Modo LPM0 con interrupciones habilitadas

results[2] = ADC10MEM;

ADC10CTL0 &= ~ENC;
ADC10CTL0 &= ~(REFON + ADC10ON);        // Apagado del ADC para reducir consumo

temp = results[0];
degC = (((temp) * 3000) / 1024);

temp = results[1];
volt = (temp*50)/1024;
```

```
temp = results[2];
sma = (temp - 205)* 123/1024;

/* message format,  UB = upper Byte, LB = lower Byte
-------------------------------
|degC LB | degC UB |  volt LB |
-------------------------------
    0         1          2
      */

      msg[0] = degC&0xFF;
      msg[1] = (degC>>8)&0xFF;
      msg[2] = volt;
      msg[3] = sma;

      /* :( :( :( :( */
```

3.5.3. Mensaje

El mensaje enviado se modificó y se puede cambiar las veces y de la forma que sea necesaria revisando la parte en que se habla del análisis del formato. El siguiente código pertenece a la versión final del mensaje y la figura 3.11 muestra el mensaje recibido en la terminal *Console*.

Virtual_com_cmds.c

```
char temp_string[] = {"X.XXXV"};
  int temp = msg[0] + (msg[1]<<8);

  temp_string[4] = '0'+(temp%10);
  temp_string[3] = '0'+((temp/10)%10);
  temp_string[2] = '0'+((temp/100)%10);
  temp_string[0] = '0'+((temp/1000)%10);

  if(data_mode & verboseMode)
  {
    char output_verbose[] = {"\r\nTerminal:XXXX  Sensor:X.XXXV  Vdd:X.XV  Humedad:XX%"};

    output_verbose[11] = addr[0];
    output_verbose[12] = addr[1];
    output_verbose[13] = addr[2];
    output_verbose[14] = addr[3];

    output_verbose[24] = temp_string[0];
    output_verbose[26] = temp_string[2];
    output_verbose[27] = temp_string[3];
    output_verbose[28] = temp_string[4];

    output_verbose[36] = '0'+(msg[2]/10)%10;
    output_verbose[38] = '0'+(msg[2]%10);

    output_verbose[50] = '0'+(msg[3]/10)%10;
    output_verbose[51] = '0'+(msg[3]%10);
    }
```

3.6. Compilación del código en CCS

Una vez que se cargó el proyecto y se modificó el código original, se compila con las nuevas características y se programan las tarjetas (Figura 3.10).

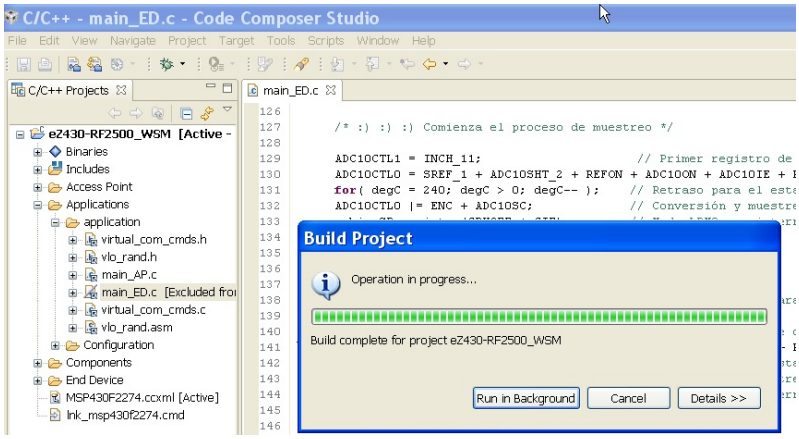


Figura 3.10: Compilación en Code Composser Estudio

3.6.1. Visualización con la interfaz *Console*

Dentro de la interfaz gráfica de prueba existe una opción que muestra los datos que se están adquiriendo en la computadora (Figura 3.11), usaremos esta herramienta como prueba para verificar que se estén transmitiendo los datos, más adelante plantearemos el uso de otro programa que nos permita manipular los datos recibidos.

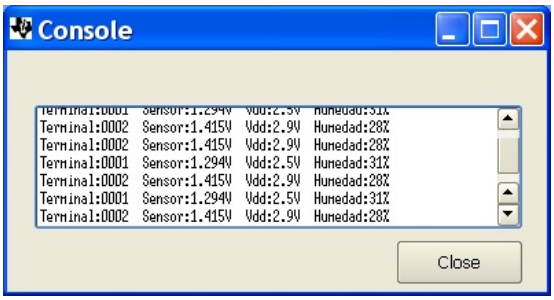


Figura 3.11: Aplicación Console de TI

Después de haber estudiado las instrucciones, registros, y variables definidas del microcontrolador, se analizaron los códigos del punto de acceso y terminales del programa de demostración, logrando cambiar los códigos para realizar las tareas de adquisición y procesamiento de los datos, quedando por trabajar en la etapa de acondicionamiento.

Capítulo 4

Tarjeta de acondicionamiento

4.1. Circuito propuesto

Una etapa previa a la adquisición de datos, es el acondicionamiento de la señal eléctrica. Esto es: un tratamiento que recibe la señal, mediante amplificadores, atenuadores, divisores de voltaje, corredores de nivel, filtros etc. Todo lo necesario para ajustar la señal dentro de niveles compatibles con la etapa de adquisición de datos.

Después de realizar el estudio de los tipos de puertos de entrada del microcontrolador MSP430 y de haber analizado la etapa de conversión analógica/digital ADC10, sabemos cómo habilitar las terminales externas de la tarjeta y dentro de que rango deben estar los niveles de las señales de entrada, también logramos manipular los datos de los registros temporales, para programar al microcontrolador y así adquirir en la computadora los datos con la información correcta de todo el sistema que formara el microsensor de flujo.

Para probar la tarjeta con sensores externos al microcontrolador y acoplar la señal proveniente de la rejilla a través de un conector SMA, se propuso fabricar un PCB con las mismas dimensiones de la eZ430-RF2500, que además de contener un amplificador de instrumentación, integrará un sensor de humedad y un diodo como sensor de temperatura. El diagrama a bloques se muestra en la figura 4.2, el sensor de humedad nos sirve para practicar con la programación ya que se debe ajustar la ecuación para obtener el valor correcto de humedad, ya que según su hoja de especificaciones el voltaje de la señal de transferencia de salida, sobrepasa los 2 volts a diferencia del programa de demostración, por lo que también es necesario saber cómo cambiar el registro de instrucciones del ADC10 para modificar el voltaje de referencia V_{REF+}. Como aún no se ha caracterizado dinámicamente a la rejilla, no se cuenta con la curva de voltaje o corriente de la rejilla respecto a la cantidad de flujo, es por eso que se decidieron incluir estas señales de prueba.

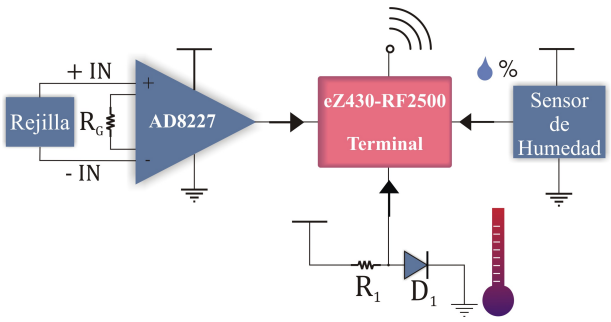


Figura 4.1: Diagrama a bloques propuesto para la tarjeta de acondicionamiento.

El número de bits del convertidor analógico-digital determina la resolución de conversión de la señal analógica adquirida, por otro lado, para que esto ocurra es necesario que la señal se ajuste a los límites de voltaje de referencia del convertidor, en este caso V_{REF+} y V_{REF-}, En el convertidor ADC10 dentro del microcontrolador estos voltajes de referencia se pueden ajustar en la programación entre VCC, 2.5V, y 1.5V para V_{REF+}, y V_{ss} o (0V) para V_{REF-}

La figura 4.2 muestra el proceso de una señal antes, durante y después de la conversión analógica digital, se puede apreciar que, a mayor cantidad de muestras, más se ajustará la curva final a la original.

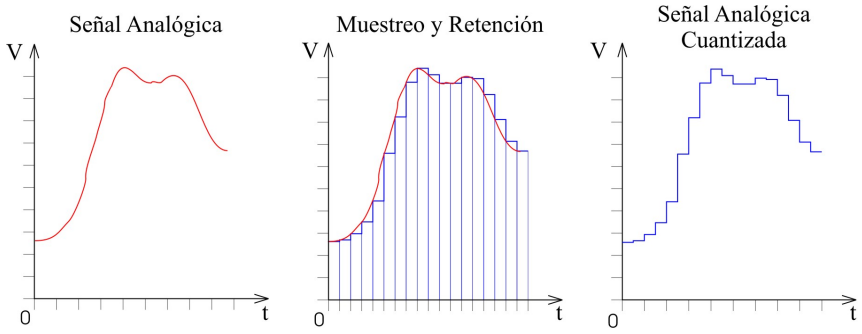


Figura 4.2: Muestreo y retención.

Para propósito de las primeras pruebas de la tarjeta, además de la señal del sensor se propuso integrar un sensor de humedad y se realizó un arreglo para tomar la temperatura ambiente, tomando como respuesta, la caída de voltaje en un diodo en polarización directa la cual debe ser proporcional al cambio de la temperatura. Estas tres muestras serán adquiridas por las terminales eZ430-RF2500 y los datos procesados, serán enviados hacia la computadora a través del punto de acceso vía inalámbrica.

El diagrama esquemático de la Figura 28, muestra los dispositivos y la forma en que se encuentran conectados los componentes en la tarjeta.

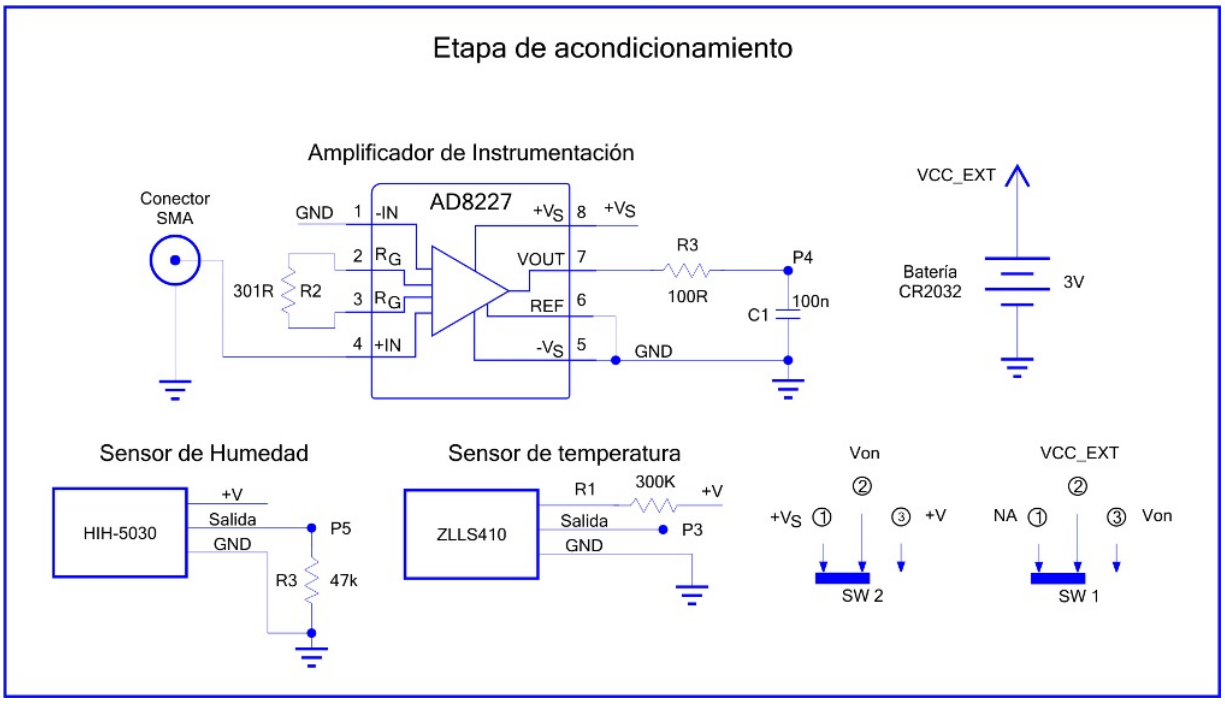


Figura 4.3: Diagrama esquemático de la tarjeta de acondicionamiento

4.2. Descripción del diagrama

Los interruptores se agregaron para controlar y dirigir el consumo de energía entre los sensores y el amplificador, o para apagar el consumo total en el PCB cuando no se utilice.

El AD2227, es un amplificador de instrumentación con un rango de ganancia programable de entre 5 y 1000 veces la señal de entrada (Ecuación 4.1), por medio de un resistor externo RG, con una razón de rechazo en modo común de 100dB cuando la ganancia es 5.

$$G = 5 + \frac{80k\Omega}{R_G} \tag{4.1}$$

El amplificador se polariza con un voltaje de entre 2.2 a 36 volts, la señal de entrada es tomada del conector coaxial Sub-Miniatura clase A (SMA) y la salida del amplificador es enviada a la terminal P4 que corresponde al puerto 2.1-A1.

El HIH-5030 es un sensor de humedad de bajo voltaje de 2.7V a 3.3V el cual envía una señal a la salida en un rango de 0.5V a 2.2V para 0 % y 100 % de humedad relativa respectivamente, la curva es compensada con una resistencia de carga de 65kΩ La respuesta corresponde a la terminal P5 o puerto 2.2 A2

Por último, un diodo en polarización directa refleja la temperatura ambiente de manera inversamente proporcional al aumento de esta, por la caída en la barrera de potencial de la unión NP, la terminal P3 correspondiente al puerto 2.0-A0 recibirá la señal de salida.

4.3. Placa de circuito impreso

Para acoplar la señal entre la rejilla y la terminal, se diseñó una placa de circuito impreso que además de servir de soporte para la pila de 3V, también tenía que incluir interruptores, el amplificador de instrumentación y los sensores de humedad y temperatura.

El PCB se interconectó físicamente a las terminales de la tarjeta eZ430-RF2500, cuidando las dimensiones para mantener al sistema portátil y del mismo tamaño que la terminal.

La figura 4.4 muestra el aspecto final del diseño del PCB listo para la fabricación y con la ubicación de los componentes y su dimensión

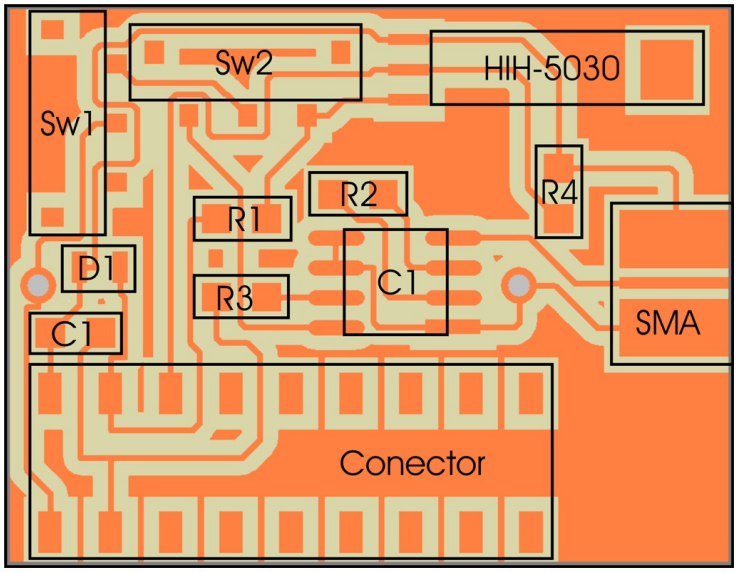


Figura 4.4: Layout de la tarjeta de acondicionamiento (3 cm X 2.3 cm)

La tarjeta se diseñó usando el programa Orcad Layout y se fabricó en una máquina de control numérico por computadora (CNC) gracias a la ayuda del taller mecánico del INAOE (Figura 4.5), usando los archivos Gerber exportados del layout.

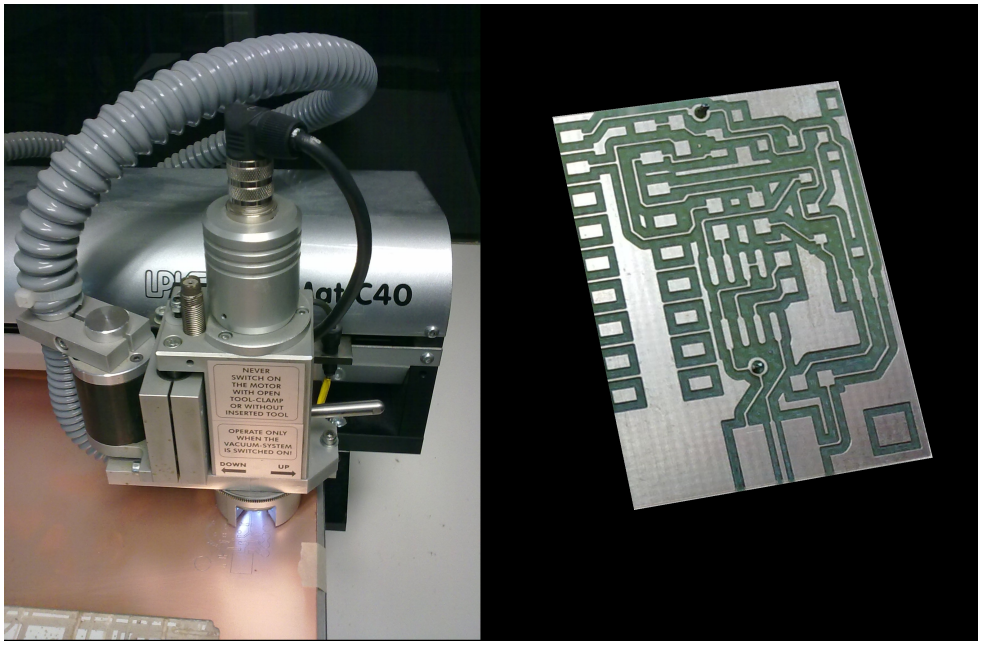


Figura 4.5: Fabricación de la tarjeta de circuito impreso

Finalmente se presentan la tarjeta completa con los componentes montados en la figura 4.6.

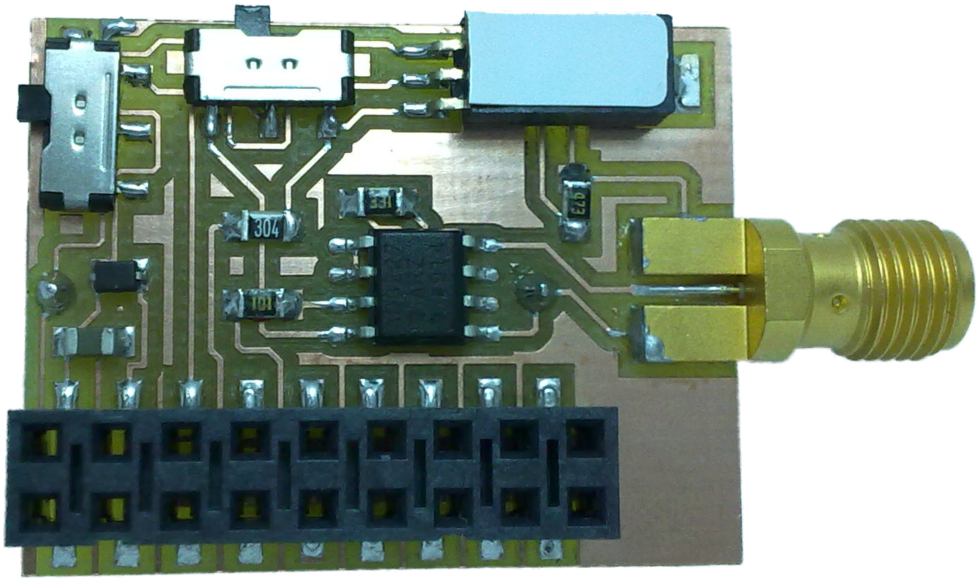


Figura 4.6: Tarjeta de acondicionamiento

4.4. Componentes de la tarjeta

La tabla 4.1 muestras los componentes empleados y su descripción, todos los dispositivos fueron adquiridos teniendo en cuenta las dimensiones de la tableta, el voltaje de alimentación y el consumo que soportara la pila CR2032.

Tarjeta de acondicionamiento	
Componentes	Descripción
AD8227ARZ-RL	Amplificador de instrumentación
HIH-5030,5031	Sensor de humedad
AYZ0102AGRLC	Interruptor de un polo dos tiros
J799-ND	Conector SMA
3M9568CT-ND	Conector hembra dual 18 terminales
ZLLS410CT-ND	Diodo
CONN HEADER 18POS .100"	Conector macho dual 18 terminales
LMK212SD104JG-T	Capacitor de 0.1 μF
RR1220P-304-D	Resistencia de 300 kΩ
RR1220P-473-D	Resistencia de 47 kΩ
RR1220P-101-D	Resistencia de 100 Ω
RR1220P-301-D	Resistencia de 300 Ω

Tabla 4.1: Lista de componentes de la tarjeta de acondicionamiento

4.4.1. Pila CR2032

La fuente de alimentación para la tarjeta eZ430-RF2500 como para la placa de acondicionamiento será suministrada por una pila de 3V la cual cuenta con las siguientes características (Tabla 4.2).

Especificaciones pila CR2032	
Voltaje nominal	3V
Corriente nominal	240 mAh
Corriente de descarga estándar	0.3 mA
Corriente de descarga máx.	Continua 4 mA
	Pulso 20mA
Rango de temperatura de operación	-20°C a 70 °C

Tabla 4.2: Características eléctricas de la Pila CR2032

4.4.2. Amplificador de instrumentación

Considerando la ganancia predefinida en la tarjeta de acondicionamiento para la rejilla, se simuló y estimó la característica que debería de tener la señal a la salida del puente Wheatstone, la cual se calculó de la siguiente manera. Teniendo en cuenta la ganancia G=6 para el amplificador de instrumentación y un voltaje de alimentación de 3 volts, una señal con un sexto de la alimentación y con un offset tal que a la salida tengamos la mitad del voltaje de alimentación. Resulta una señal pico a pico de 460mV con un offset de 250mV. Se simuló con estos valores por medio de un generador se señales inyectando al amplificador de instrumentación dicha señal y se obtuvo la respuesta de la figura 4.7.

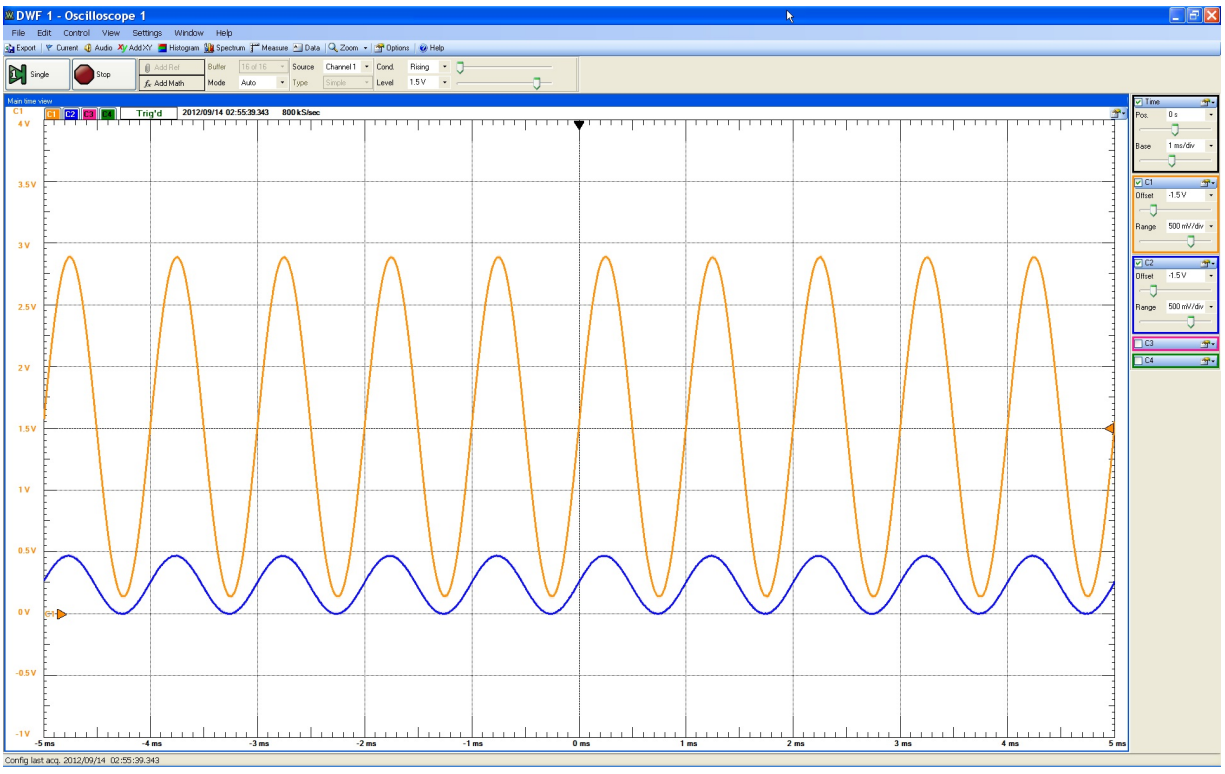


Figura 4.7: Señal transitoria del amplificador de instrumentación

En la figura 4.8 se presenta la etapa de acondicionamiento montada en una tarjeta *Digilent Electronics Explorer Board*, en la cual se hicieron las mediciones de prueba para comprobar que los circuitos de humedad, temperatura y amplificación, estuvieran funcionando antes de acoplarla a la tarjeta eZ430-RF2500.

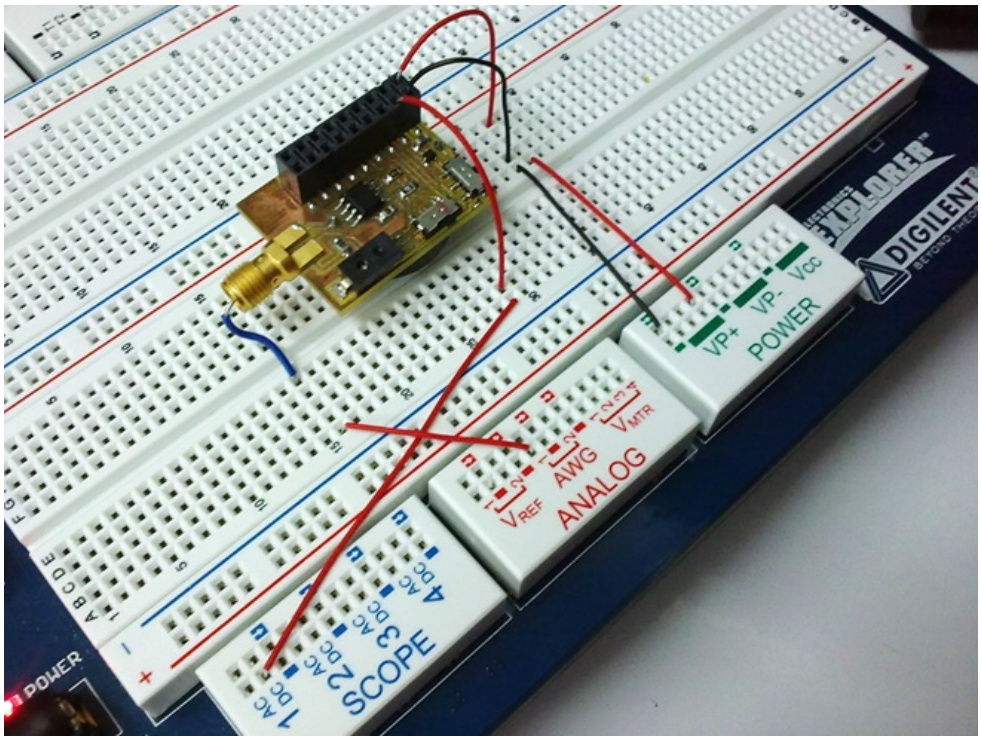


Figura 4.8: Pruebas con el amplificador de instrumentación

4.5. Sensor de humedad relativa

Para probar la tarjeta eZ430-RF2500 con una señal externa, se propuso agregar un sensor de humedad comercial a la tarjeta de acondicionamiento, para hacer pruebas de programación del microcontrolador. Para probar la curva de humedad que reporta el fabricante del sensor de humedad relativa HIH-5030, se alimentó al sensor con 3V y se realizaron las siguientes acciones.

- Se introdujo en un contenedor de desecante comercial para el control de humedad, logrando obtener 0.5 Volts prácticamente de inmediato, valor proporcional a 0 % de humedad relativa.
- Se saturó con vapor de agua un contenedor con el sensor de humedad, con una fuente de calor para evitar que se condensara el vapor, el valor de la señal cumplió con lo propuesto por la curva mostrada en la figura 4.9 de la hoja de datos del fabricante, de aproximadamente 2.5V debido a la temperatura superior a la temperatura ambiente dentro del contenedor.

Todo esto solo para comprobar la curva propuesta. El sensor de humedad tiene un consumo típico de 200μA y máximo 500μA, el voltaje a la salida está definido como: $V_{OUT} =$

$(V_{SUPPLY})(0.00636(sensorRH) + 0.1515)$ típico a 25 °C.

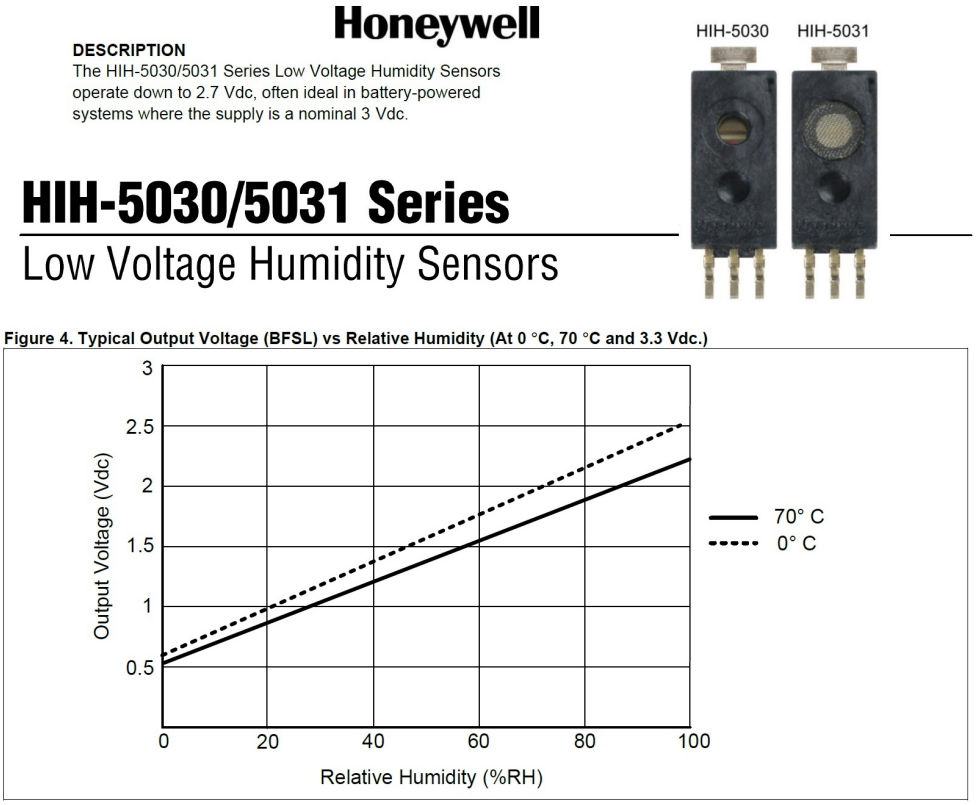


Figura 4.9: Sensor de humedad empleado en las pruebas

4.6. Sensor de temperatura

Otra propuesta fue la de censar el cambio de temperatura de la rejilla respecto al flujo de gas que la cruzara, empleando un diodo en polarización directa como sensor de temperatura. Como se muestra en las curvas de la figura 4.10 la temperatura afecta el comportamiento de la corriente en los diodos y en cualquier unión NP. Por lo tanto, es un excelente instrumento para detectar las variaciones de temperatura. Sin embargo, el coeficiente de temperatura negativo, provoca que el voltaje en la unión sea inversamente proporcional al incremento de temperatura o lo que es lo mismo, a mayor temperatura menor será la caída de voltaje que midamos en las terminales del diodo, obteniendo voltajes menores al de la barrera de potencial de alrededor de 0.6V.

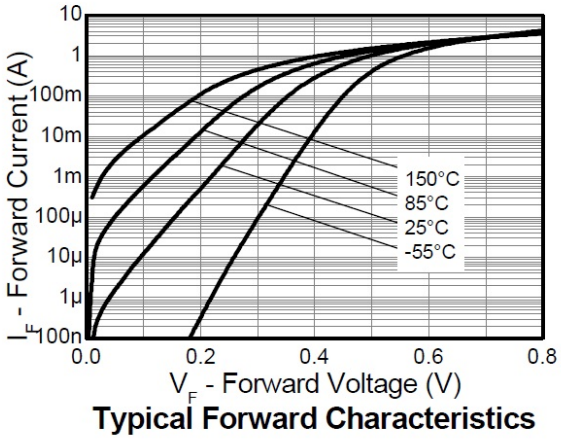


Figura 4.10: Corriente de polarización directa vs Temperatura

Un inconveniente de utilizar este circuito, es el pequeño rango en el que cambia el voltaje en el diodo, lo cual nos obliga a contar con una etapa previa de amplificación, o de ajustar las señales de referencia en el ADC o de lo contrario al entrar a la etapa de adquisición, la señal tendría una muy baja resolución que a pesar de que se pudiera amplificar (multiplicar) dentro del procesamiento, nos encontraríamos con una señal escalonada por la razón de no haber acondicionado la señal como podemos ver en la figura 4.11.

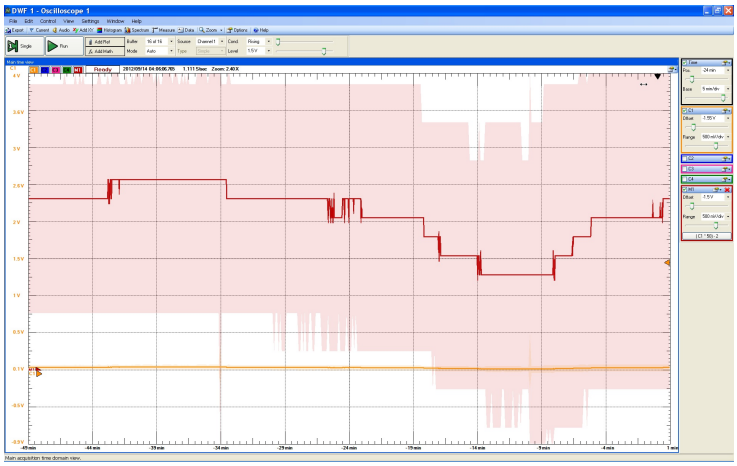


Figura 4.11: Error de adquisición de la señal del diodo por falta de acondicionamiento

Después de probar la tarjeta que recibirá la señal de la rejilla y de los otros sensores, se acoplará con la tarjeta eZ430-RF2500 para realizar la adquisición y procesamiento de las 3 señales y poder desplegar la información en la computadora como se verá en el siguiente capítulo, que muestra los resultados de la programación.

Capítulo 5

Resultados

5.1. Acoplamiento con la tarjeta eZ430-RF2500

Después de la caracterización de la etapa de acondicionamiento y de modificar los códigos de programación, tenemos finalmente el sistema de adquisición y transmisión de datos. El diagrama de la figura 5.1 muestra a la izquierda de manera detallada el diagrama esquemático de la terminal y a la derecha, el diagrama de la etapa de acondicionamiento.

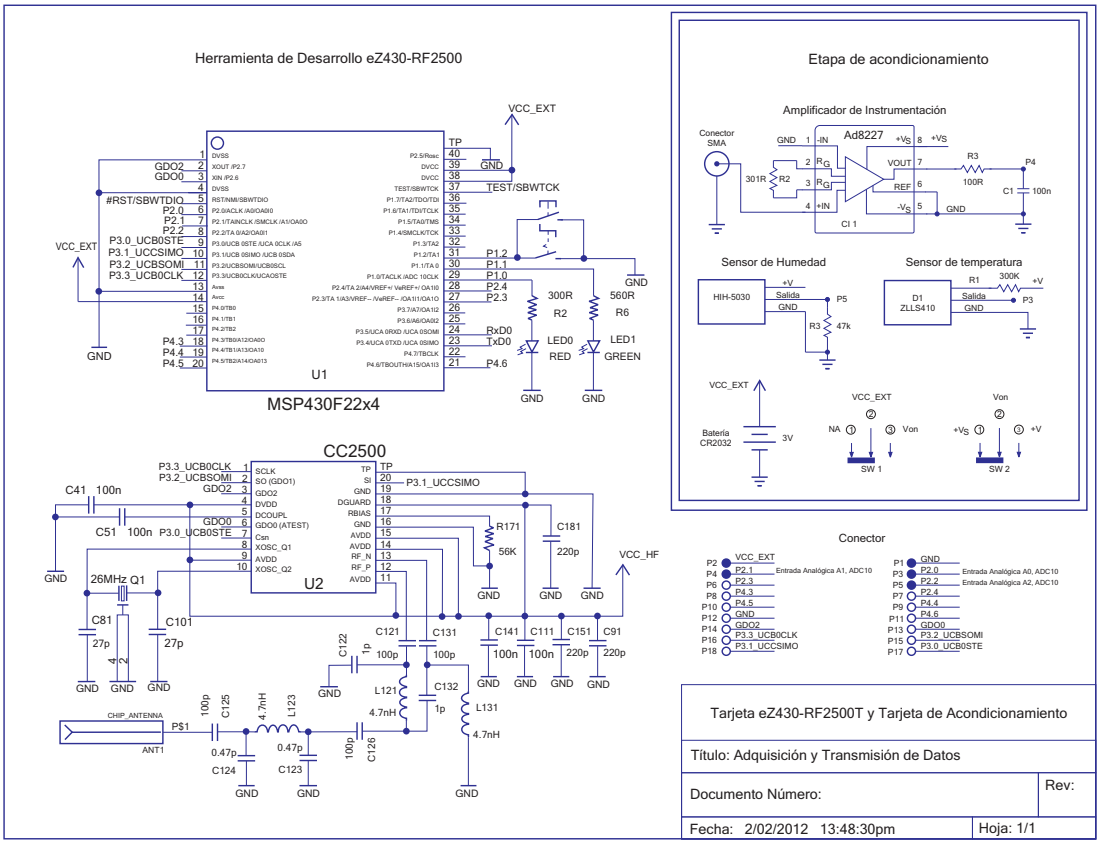


Figura 5.1: Esquema del sistema de adquisición y transmisión de datos

Teniendo el sistema completo como se muestra en la figura 5.2, se puede caracterizar la tarjeta de acondicionamiento con cualquier fuente, por el momento de manera independiente a la señal de la rejilla de flujo, por medio de un generador de señales.

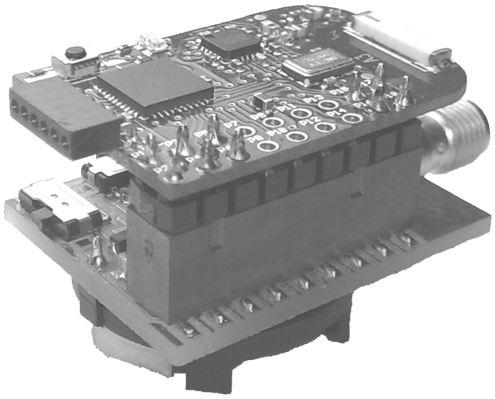


Figura 5.2: Sistema de adquisición y transmisión de datos

Ahora podemos enviar una señal externa y adquirirla por la tarjeta eZ430-RF2500, utilizando un generador de señales y transmitir esta señal a través de la terminal, hacia el punto de acceso conectado en la computadora por USB, usando el programa de comunicación *HyperTerminal* para capturar los datos recibidos almacenándolos en un archivo de texto. La figura 5.3 muestra el montaje para realizar las pruebas.

Figura 5.3: Montaje de pruebas para la adquisición y transmisión de datos

5.2. Datos recibidos vía USB

Puertos de conexión

Para la comunicación entre el punto de acceso y la computadora se asigna automáticamente un puerto COM numerado a la tarjeta USB, el cual debe ser nombrado cuando se intente hacer la comunicación y configuración con HyperTerminal. La figura 5.4 muestra la manera en que se puede verificar el puerto asignado.

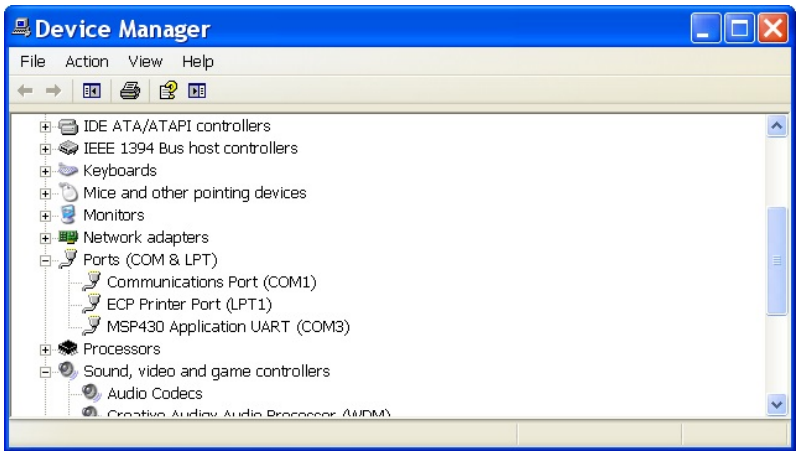


Figura 5.4: Puerto asignado al circuito UART de la eZ430-RF2500

Configuración del puerto COM

Una vez reconocido el puerto asignado MSP430 application UART (COM1), se abre una nueva conexión en HyperTerminal y una vez iniciado la aplicación se debe seleccionar el puerto asignado y se continúa con la configuración como se muestra en la figura 5.5

Figura 5.5: Conexión al puerto COM correcto

Para configurar las propiedades del puerto, se asigna una velocidad de comunicación de 9600 bits por segundo, para que detecte y se sincronicen los datos recibidos con los enviados para una lectura correcta tal como se muestra en la figura 5.6.

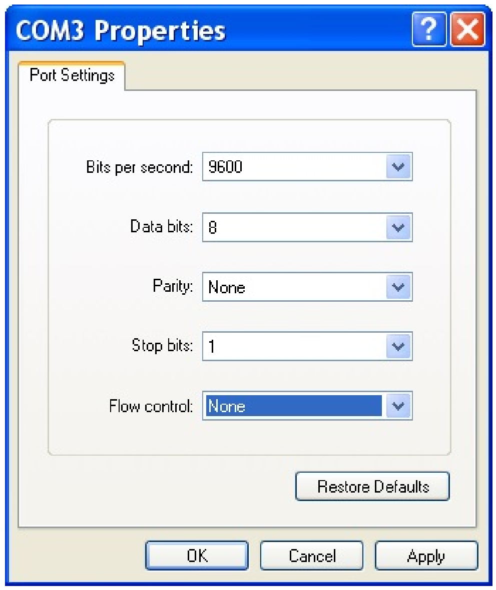


Figura 5.6: Configuración de la comunicación

Una vez establecida la conexión, automáticamente empiezan a desplegarse los mensajes enviados, los cuales portan la información que en este caso corresponde a la señal del generador (figura 5.7).

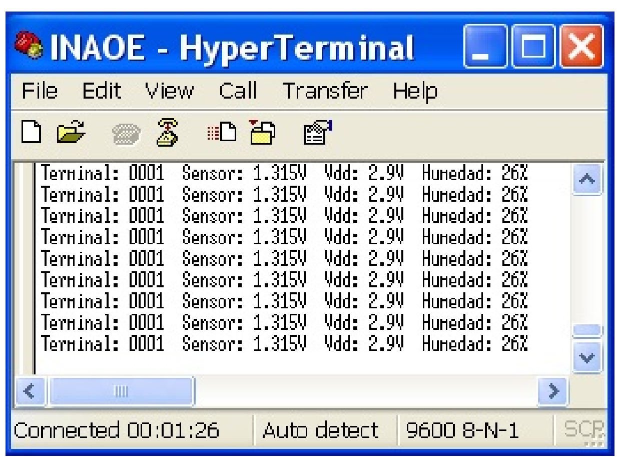


Figura 5.7: Mensaje desplegado en la *HyperTerminal*

HyperTerminal puede generar un archivo con el texto de los mensajes recibidos durante un periodo, donde el usuario elige cuando comienza o termina la toma de muestras.

Después del almacenamiento se puede disponer de los datos en cualquier momento y estos pueden ser analizados y procesados por alguna herramienta de análisis de datos, la figura 5.8 muestra en el bloc de notas una parte del registro de los datos.

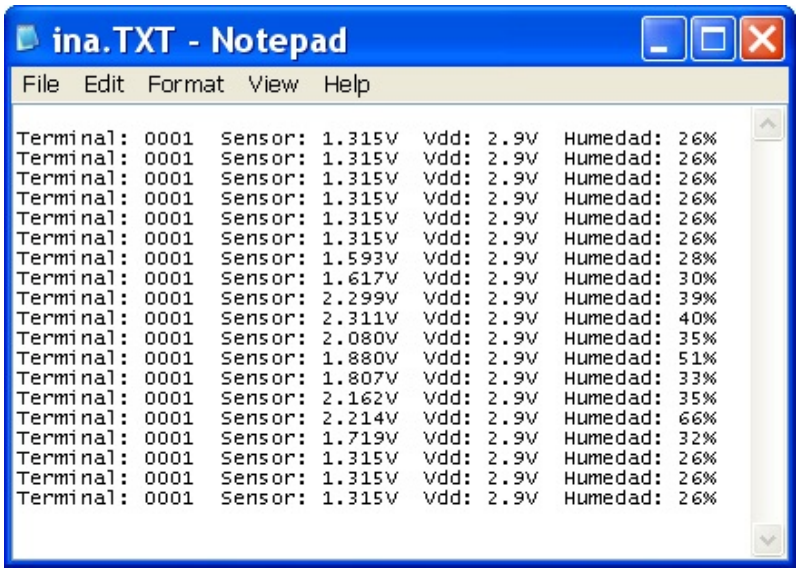


Figura 5.8: Datos almacenados

Los datos adquiridos y almacenados pueden ser analizados o se pueden manipular para obtener la información relevante de acuerdo a las necesidades que se tengan en específico y graficarlas mediante algún programa. La figura 5.11 muestra la señal reconstruida a partir de los datos recuperados y graficados en Excel, previamente separando los datos requeridos de las demás columnas de humedad voltaje de batería y número de la terminal.

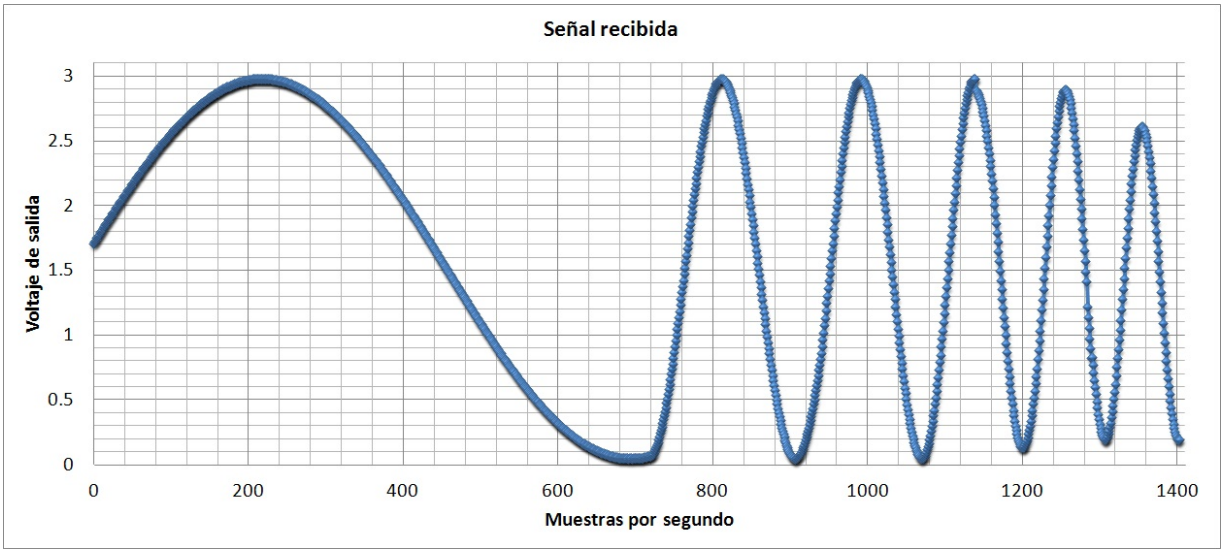


Figura 5.9: Señal recuperada de los datos almacenados

Considerando que el envío de datos se produce cada segundo, la cantidad de muestras recuperadas dependerá de la frecuencia de la señal de entrada.

Para esta prueba se transmitió una onda senoidal con un barrido en frecuencia de 1 mHz a 10 mHz. Los cambios abruptos en la gráfica se deben, al momento que se realizó el cambio de frecuencia en el generador de señales, una vez que se estableció la frecuencia no se presentaron discontinuidades.

Se puede observar la disminución de muestras al incrementar la frecuencia de la señal senoidal, debido a que al momento de registrarse la señal y almacenarse no existe una retroalimentación visual, no fue posible percibir el detalle que, como se continuó aumentando la frecuencia a partir del minuto 20 aproximadamente, se creó el efecto de una señal atenuada. La muestra fue tomada durante 23.33 minutos de registro o 1400 muestras.

5.3. Consumo de energía

Tomando en cuenta la capacidad de la pila y el consumo de corriente del sistema, se puede estimar el tiempo de funcionamiento de la siguiente manera.

$$\frac{\text{Capacidad de la pila mAh}}{\text{Consumo del sistema mA}} = \text{Vida útil de la pila h}$$

Con una capacidad de la fuente de 340 mAh y un consumo por el sistema de 390 µA, (considerando solo a la tarjeta eZ430-RF2500 y al amplificador de instrumentación), el tiempo de operación sería de.

$$\frac{340mA}{390\mu A} = 871h = 36días \tag{5.1}$$

Este análisis es una aproximación muy simple de las pilas de 3V empleadas, ya que las baterías tienen curvas de descarga con capacidades de carga muy limitadas además de que el voltaje de las mismas baja hasta los 2 volts y decae más rápido si no se cumplen con las corrientes recomendadas por el fabricante de descarga

La siguiente imagen (Figura 5.10) muestra la relación entre la vida útil de la fuente de alimentación con el número de muestras tomadas y trasmitidas.

El consumo de la tarjeta de adquisición contrasta con el tiempo esperado de vida en años para la tarjeta de desarrollo que presenta la siguiente curva, la cual muestra la dependencia en función del periodo en segundos entre cada transmisión de datos.

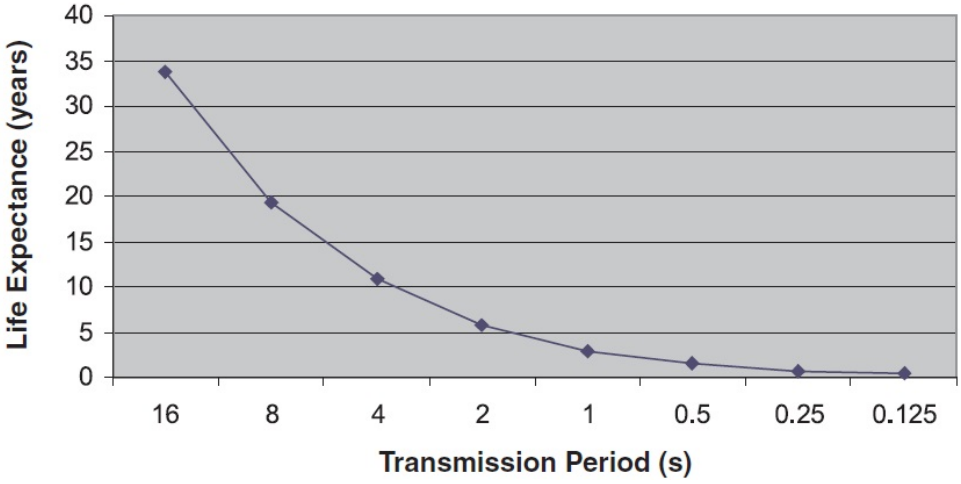


Figura 5.10: Años de operación vs Intervalo entre transmisiones

Lo reportado tiene mucho sentido si observamos la curva de consumo de las tarjetas de desarrollo (figura 5.11), donde podemos notar que efectivamente el tiempo que dura la transmisión es "despreciable" debido a que el microprocesador entra en modo de ultra bajo consumo, así como el transceptor, mientras que el consumo del amplificador de instrumentación y de los sensores que agregamos en la tarjeta de acondicionamiento, siempre permanecerán activos por mínimo consumo que tengan.

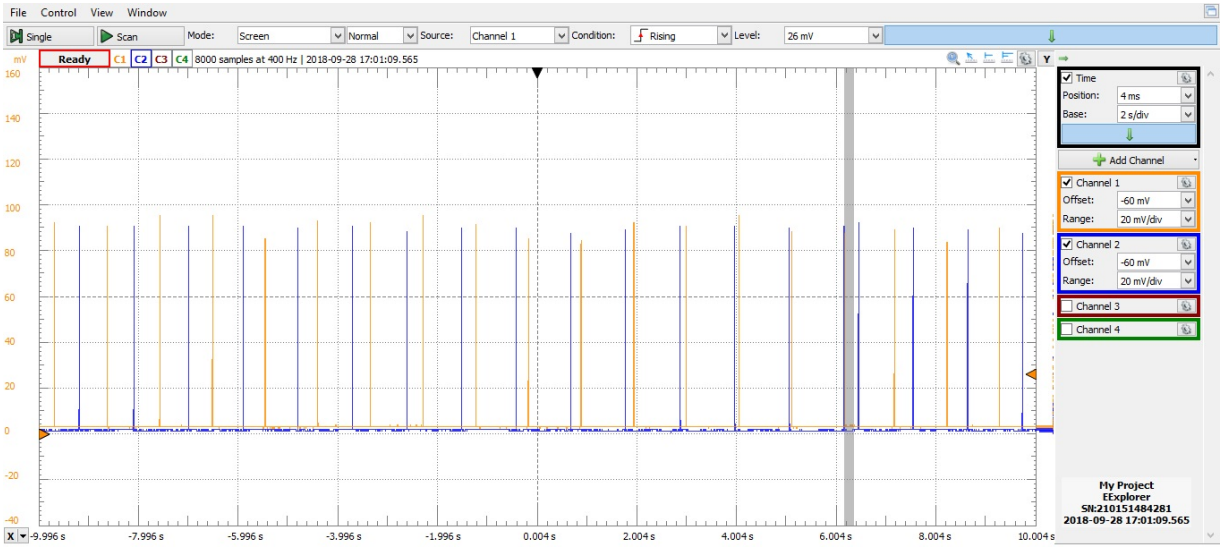


Figura 5.11: Consumo de corriente en las 2 terminales

Para observar en el osciloscopio, el consumo de las terminales, se introdujo una resistencia de 50Ω en serie con la batería y se midió el voltaje que caía entre las terminales de la resistencia, como se sugiere en el reporte de la tarjeta de desarrollo.

Cambiando el tiempo por división de cuadro a 1ms en el osciloscopio y activar la sincronización por disparo de flanco positivo, se pudo notar el consumo de las terminales, observando el perfil del consumo que tiene una duración de 6.5ms y una forma como se puede observar en la figura 5.12

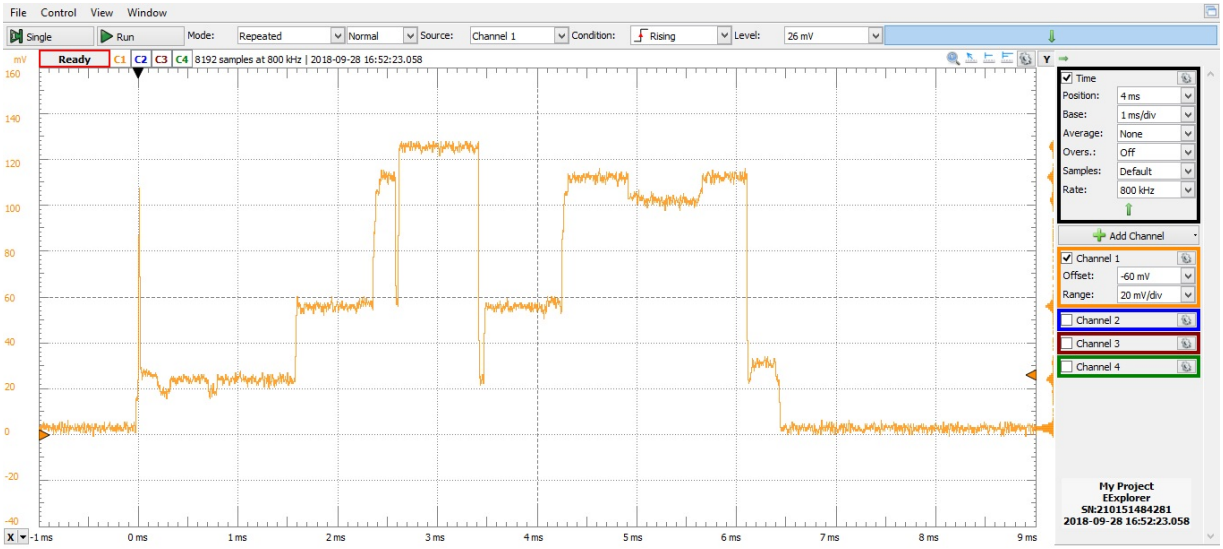


Figura 5.12: Consumo de una terminal a detalle

5.4. Respuesta de temperatura de la rejilla al escalón

Durante la caracterización de las variables eléctricas de la rejilla, se observó que le tomaba minutos alcanzar el valor máximo de temperatura, por una parte, debido al substrato de silicio sobre el que se encuentra depositado el metal, además del chasis donde se alambra y sostiene la rejilla, que en este caso fue una tableta de circuito impreso, que también disipaba parte del calor.

Por otra parte, el suministro de oxígeno es controlado por un regulador a la salida del tanque de gas, lo que significa que el flujo que se registrará se encontrará prácticamente sin variaciones, salvo por las perturbaciones que puedan ocurrir al cruzar los gases por la rejilla y por las variaciones debido a la temperatura ambiente en ese momento.

La figura 5.13 muestra la rejilla de prueba a la que se le aplicó un voltaje de 1.9V, consumiendo 740 mA, que alcanza una temperatura de 90 °C censada con una pistola infrarroja.

Figura 5.13: Respuesta de temperatura al escalón

A continuación se presenta la respuesta de temperatura al escalón de voltaje, el cual se capturó en la interfaz gráfica de la tarjeta *Digilent* utilizada anteriormente.

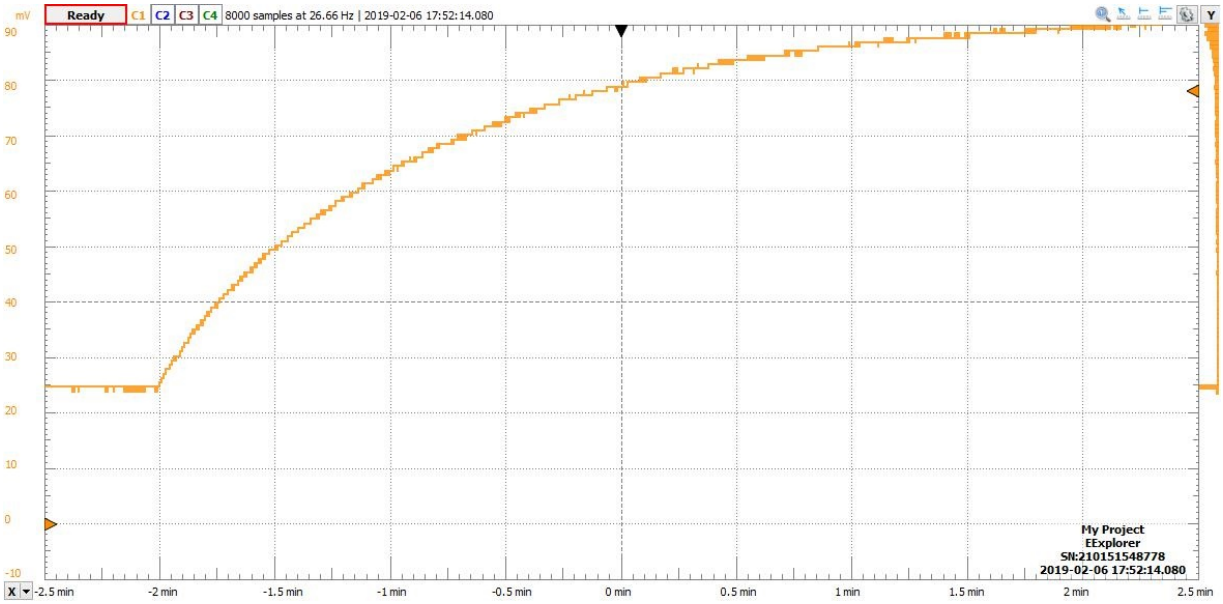


Figura 5.14: Respuesta de temperatura al escalón

Como podemos observar en la figura 5.14 le toma 4 minutos y medio alcanzar la máxima temperatura con el voltaje aplicado, es por esta razón que la frecuencia de la toma de muestras, se decidió mantener en una por segundo sin embargo, si es necesario modificar la frecuencia de la toma de las muestras, este cambio se puede realizar.

5.5. Puente Wheatstone

Todos los conductores poseen una dependencia entre la temperatura y su conductividad, que es llamado coeficiente de temperatura, en el caso del aluminio este coeficiente es de 3.9 x 10^{-3}, y al ser positivo significa que a mayor temperatura la resistencia del aluminio aumentará proporcionalmente. Es gracias a esta propiedad que empleando el puente de Wheatstone podemos obtener un cambio de voltaje respecto al cambio de flujo ya que, al inyectar corriente al puente, la temperatura de la rejilla aumenta, pero cuando se presente un flujo de gas, éste flujo tenderá a enfriar el conductor y con esto, se provocará un cambio de la resistencia S, lo que se verá reflejado en el voltaje V_{AB} de la figura 5.15.

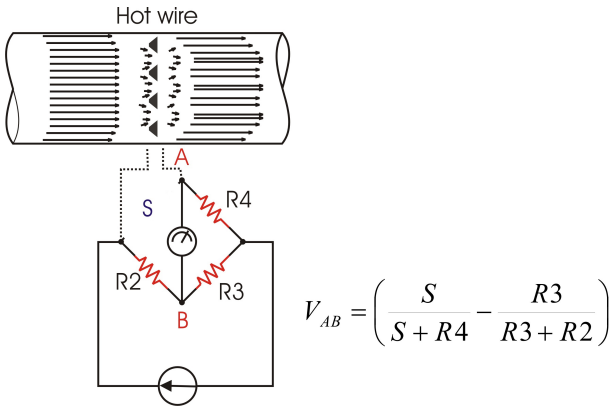


Figura 5.15: Respuesta de voltaje V_{AB} vs Flujo de gas

Hasta el momento no se ha decidido si se usará el puente de Wheatstone o censar directamente la temperatura de la rejilla y calcular de esta manera el flujo de gas, ya que para que fluya una corriente de la magnitud que consume la rejilla por todo el puente, se requerirá mucho consumo de energía, calculando solo para la rejilla multiplicando el voltaje y la corriente, de la última medición se tendría, 1.9V x 0.740 A = 1.4Watts, multiplicado por las 4 resistencias, el circuito consumiría 5.6 Watts

5.6. Transmisión y recepción

El circuito CC2500 encargado de la comunicación inalámbrica en la tarjeta de desarrollo opera en el rango de frecuencia entre 2400MHz y 2483.5MHz. Para poder entender cómo se lleva a cabo la comunicación entre los componentes del sistema, se realizó una prueba empleando un analizador de espectros en el laboratorio de diseño de circuitos integrados del INAOE que se muestra a continuación.

Debido a la duración del "pulso" de transmisión visto en la figura 5.11, resulta imposible reconocer y mostrar el espectro de nuestra fuente en el modo normal del analizador debido a que es una señal intermitente, sin embargo, existe la herramienta de retención de valores máximos. Por otro lado, la potencia máxima del CC2500 es de apenas +1dBm por lo que las 2 terminales usadas como el punto de acceso, se colocaron aún lado de la antena del analizador de espectros como se aprecia en la figura 5.18, solo con el fin de conocer la o las frecuencias en las que se transmiten los datos y como se establecen y relacionan las terminales con el punto de acceso.
Para observar el espectro del transmisor se utilizó el analizador de espectros Rode&Schwarz

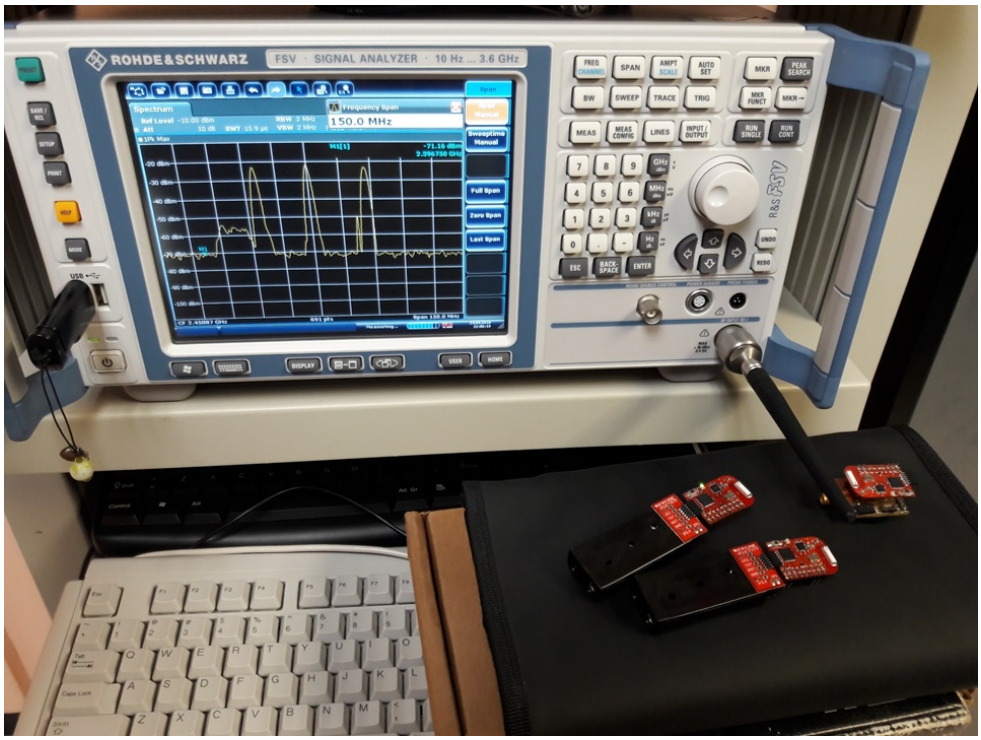


Figura 5.16: Espectro de radiofrecuencia de 2 terminales y un punto de acceso

Para observar y entender cómo es que se entablaba la comunicación entre el punto de acceso y las terminales, así como la frecuencia de trabajo, después de un tiempo encendiendo y apagando el punto de acceso, encontramos lo siguiente:

- Si no existe ninguna terminal encendida el punto de acceso se alimenta, este transmite a cierta frecuencia una señal dirigida hacia las terminales para entablar la comunicación

- Cuando las terminales se polarizan con el punto de acceso activo previamente, estas son reconocidas por el punto de acceso, a la misma frecuencia de él y las 2 terminales continúan enviando la información por el mismo canal.
- Si se reinicia el punto de acceso, la frecuencia cambia al siguiente "canal" o frecuencia más alta y las terminales escanean y encuentran la frecuencia del punto de acceso.

Del proceso anterior encontramos 4 canales que cambian de una frecuencia menor a una mayor cada vez que se reinicia el punto de acceso, tal vez podría haber más canales, sin embargo con solo 2 terminales logramos observar y registrar estos canales.

Podemos ver en la figura 5.18 que a unos centímetros de la antena la señal es de -21dBm (7.94μW) mientras que el promedio de las señales del mismo espectro ISM era de -60dBm (1nW) con una sensibilidad de receptor de hasta -100dBm(100fW) considerando y un ruido de piso de -120dBm(1fW)

Las frecuencias recurrentes fueron de: 2425.7MHz, 2450.7MHz, 2475.4MHz y 2477.9MHz

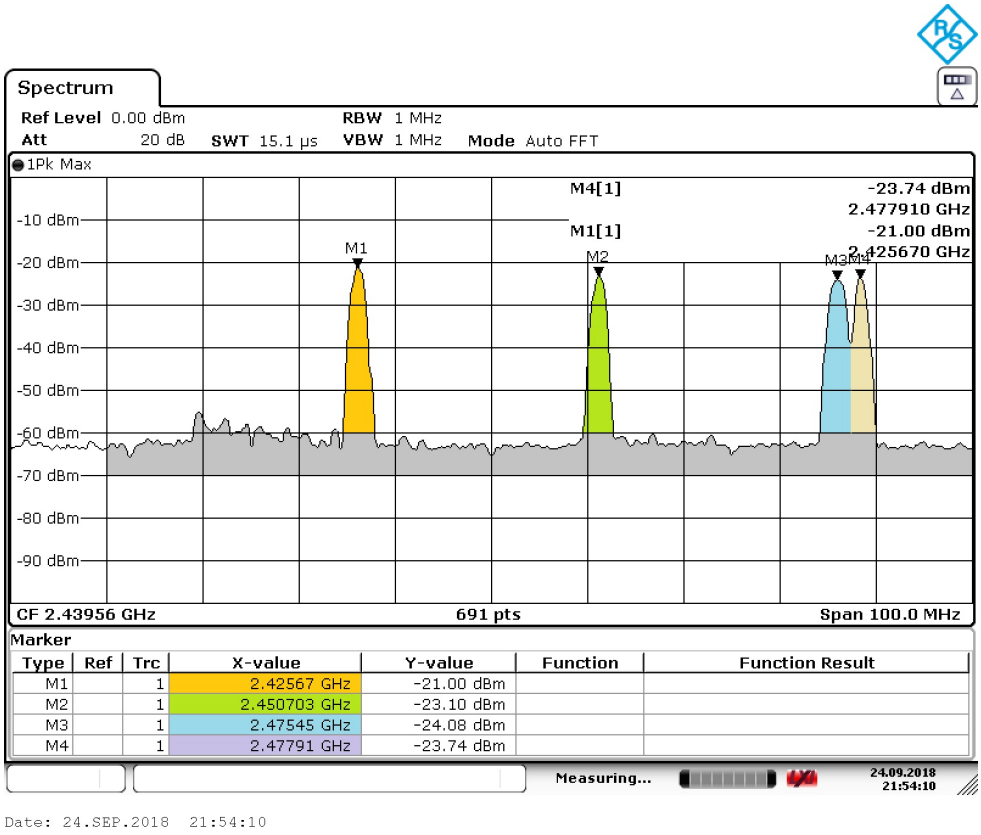


Figura 5.17: Frecuencias de los diferentes canales de comunicación

La imagen capturada de la pantalla del analizador, muestra el espectro de cada una de

las frecuencias donde se entabló la comunicación entre las 2 terminales y el punto de acceso.

Dentro de los protocolos de comunicación inalámbricos existentes, podemos ver uno llamado SimpliciTI, del mismo fabricante del microcontrolador y del transceptor, el cual se encarga de la comunicación de la eZ430-RF2500. Mostrando el lugar que ocupa entre los protocolos más reconocidos de los sistemas inalámbricos, el cual se caracteriza por un bajo rango de transmisión debido a que está designado para dispositivos de corto alcance (SRD Short Range Devices) dentro de la banda libre de transmisión ISM (Industrial Scientific and Medical) la cual es una porción del espectro electromagnético reservado internacionalmente para la industria la investigación y la medicina, y debido a que simpliciTI está diseñado para bajo consumo de potencia la tasa de transmisión de datos también es reducido.

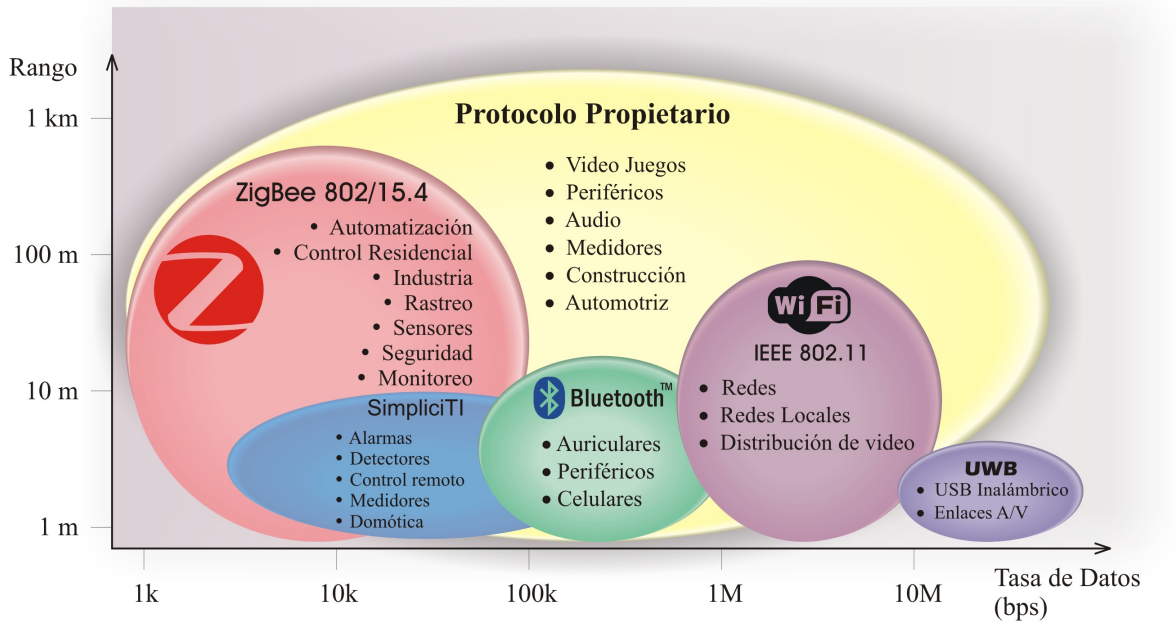


Figura 5.18: Protocolos de comunicación en la banda ISM

Capítulo 6

Conclusiones

El programa de demostración envía desde las terminales, la temperatura y voltajes internos en el microcontrolador de la tarjeta eZ430-RF2500 segundo. En base al estudio del microcontrolador y del código de programación original, se logró reprogramar parte del código, que era necesario para adquirir y procesar las señales que se requerirán para la caracterización dinámica del sensor de flujo.

Para la adquisición de los datos, es necesario acondicionar la señal antes de realizar la conversión analógica-digital, para obtener la mayor resolución. Este acondicionamiento es un tratamiento de la señal analógica para colocarla dentro de los niveles de referencia del convertidor analógico digital, para esto se propuso el diseño y fabricación de la etapa de acondicionamiento y del PCB, para realizar pruebas con otros sensores antes de la caracterización de los sensores de flujo.

El procesamiento de los datos en la CPU mapea los datos adquiridos del MEMS para la caracterización dinámico del sensor respecto al flujo y así obtener la información del flujo de gas con los valores y unidades correctas.

Se deja a disposición de trabajos futuros, todo el análisis que se llevó a cabo para modificar el código de programación y poder realizar nuevas tareas, así como de resolver la forma de acoplar otras señales a la tarjeta.

Referente a la caracterización dinámica del sensor respecto al flujo, el sistema es lo suficientemente capaz de adaptarse a los cambios de respuesta y poder tomar el número de muestras por segundo necesarias que requiera la medición.

Los datos se adquirieron y almacenaron, usando la herramienta *Hyperterminal* de Windows, la información almacenada como vector de datos, se pudo graficar en Excel.

Bibliografía

[1] P. M. Dunn, "Julius hess, md,(1876–1955) and the premature infant," *Archives of Disease in Childhood-Fetal and Neonatal Edition*, vol. 85, no. 2, pp. F141–F144, 2001.

[2] J. P. Baker, "The incubator and the medical discovery of the premature infant," *Journal of Perinatology*, vol. 20, no. 5, p. 321, 2000.

[3] O. Saugstad, "Mechanisms of tissue injury by oxygen radicals: implications for neonatal disease," *Acta Paediatrica*, vol. 85, no. 1, pp. 1–4, 1996.

[4] D. S. McLeod, S. N. Crone, and G. A. Lutty, "Vasoproliferation in the neonatal dog model of oxygen-induced retinopathy.," *Investigative ophthalmology & visual science*, vol. 37, no. 7, pp. 1322–1333, 1996.

[5] L. Tan and J. Jiang, *Fundamentals of analog and digital signal processing*. AuthorHouse, 2007.

[6] R. Pallas-Areny and J. G. Webster, *Sensors and signal conditioning*. John Wiley & Sons, 2012.

[7] D. H. Sheingold and A. Devices, *Analog-digital conversion handbook*, vol. 16. Prentice-Hall Englewood Cliffs, NJ, 1986.

[8] N. V. Kirianaki, S. Y. Yurish, N. O. Shpak, and V. P. Deynega, *Data acquisition and signal processing for smart sensors*. Wiley New York, 2002.

[9] L. R. Rabiner and B. Gold, "Theory and application of digital signal processing," *Englewood Cliffs, NJ, Prentice-Hall, Inc., 1975. 777 p.*, 1975.

[10] S. Benedetto and E. Biglieri, *Principles of digital transmission: with wireless applications*. Springer Science & Business Media, 1999.

[11] J. Espina, T. Falck, A. Panousopoulou, L. Schmitt, O. Mülhens, and G.-Z. Yang, "Network topologies, communication protocols, and standards," in *Body sensor networks*, pp. 189–236, Springer, 2014.

[12] J.-S. Lee, Y.-W. Su, and C.-C. Shen, "A comparative study of wireless protocols: Bluetooth, uwb, zigbee, and wi-fi," in *Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE*, pp. 46–51, Ieee, 2007.

[13] K. Mikhaylov, N. Plevritakis, and J. Tervonen, "Performance analysis and comparison of bluetooth low energy with ieee 802.15. 4 and simpliciti," *Journal of Sensor and Actuator Networks*, vol. 2, no. 3, pp. 589–613, 2013.

[14] A. Sikora and V. F. Groza, "Coexistence of ieee802. 15.4 with other systems in the 2.4 ghz-ism-band," in *2005 IEEE Instrumentationand Measurement Technology Conference Proceedings*, vol. 3, pp. 1786–1791, IEEE, 2005.

[15] T. Instruments, "ez430-rf2500 development tool user's guide," *Texas Instruments SLAU227E*, 2009.

[16] T. Instruments, "Msp430f2274 low power solutions [online], datasheet," 2008.

[17] T. Instruments, "Cc2500 low-cost low-power 2.4 ghz rf transceiver," 2011.

[18] C. C. Studio, "Getting started guide," *Texas Instruments.–2001*, 2006.

Capítulo 7

Apéndice

7.1. Código original del punto de acceso.

```
//*****************************************************************************
// THIS PROGRAM IS PROVIDED "AS IS". TI MAKES NO WARRANTIES OR
// REPRESENTATIONS, EITHER EXPRESS, IMPLIED OR STATUTORY,
// INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS
// FOR A PARTICULAR PURPOSE, LACK OF VIRUSES, ACCURACY OR
// COMPLETENESS OF RESPONSES, RESULTS AND LACK OF NEGLIGENCE.
// TI DISCLAIMS ANY WARRANTY OF TITLE, QUIET ENJOYMENT, QUIET
// POSSESSION, AND NON-INFRINGEMENT OF ANY THIRD PARTY
// INTELLECTUAL PROPERTY RIGHTS WITH REGARD TO THE PROGRAM OR
// YOUR USE OF THE PROGRAM.
//
// IN NO EVENT SHALL TI BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
// CONSEQUENTIAL OR INDIRECT DAMAGES, HOWEVER CAUSED, ON ANY
// THEORY OF LIABILITY AND WHETHER OR NOT TI HAS BEEN ADVISED
// OF THE POSSIBILITY OF SUCH DAMAGES, ARISING IN ANY WAY OUT
// OF THIS AGREEMENT, THE PROGRAM, OR YOUR USE OF THE PROGRAM.
// EXCLUDED DAMAGES INCLUDE, BUT ARE NOT LIMITED TO, COST OF
// REMOVAL OR REINSTALLATION, COMPUTER TIME, LABOR COSTS, LOSS
// OF GOODWILL, LOSS OF PROFITS, LOSS OF SAVINGS, OR LOSS OF
// USE OR INTERRUPTION OF BUSINESS. IN NO EVENT WILL TI'S
// AGGREGATE LIABILITY UNDER THIS AGREEMENT OR ARISING OUT OF
// YOUR USE OF THE PROGRAM EXCEED FIVE HUNDRED DOLLARS
// (U.S.$500).
//
// Unless otherwise stated, the Program written and copyrighted
// by Texas Instruments is distributed as "freeware".  You may,
// only under TI's copyright in the Program, use and modify the
// Program without any charge or restriction.  You may
// distribute to third parties, provided that you transfer a
// copy of this license to the third party and the third party
// agrees to these terms by its first use of the Program. You
// must reproduce the copyright notice and any other legend of
// ownership on each copy or partial copy, of the Program.
//
// You acknowledge and agree that the Program contains
// copyrighted material, trade secrets and other TI proprietary
// information and is protected by copyright laws,
// international copyright treaties, and trade secret laws, as
// well as other intellectual property laws.  To protect TI's
// rights in the Program, you agree not to decompile, reverse
// engineer, disassemble or otherwise translate any object code
// versions of the Program to a human-readable form.  You agree
```

```
// that in no event will you alter, remove or destroy any
// copyright notice included in the Program.  TI reserves all
// rights not specifically granted under this license. Except
// as specifically provided herein, nothing in this agreement
// shall be construed as conferring by implication, estoppel,
// or otherwise, upon you, any license or other right under any
// TI patents, copyrights or trade secrets.
//
// You may not use the Program in non-TI devices.
//
//******************************************************************************
//   eZ430-RF2500 Temperature Sensor Access Point
//
//   Description: This is the Access Point software for the eZ430-2500RF
//                Temperature Sensing demo
//
//
//   L. Westlund
//   Version    1.03
//   Texas Instruments, Inc
//   August 2009
//   Built with IAR Embedded Workbench Version: 4.20
//******************************************************************************
//Change Log:
//******************************************************************************
//Version:  1.03
//Comments: Added support for SimpliciTI 1.1.0
//          Added support for Code Composer Studio
//          Added security (Enabled with -DSMPL_SECURE in smpl_nwk_config.dat)
//          Added acknowledgement (Enabled with -DAPP_AUTO_ACK in smpl_nwk_config.dat)
//          Based the modifications on the AP_as_Data_Hub example code
//Version:  1.02
//Comments: Changed Port toggling to abstract method
//          Removed ToggleLED
//          Fixed comment typos/errors
//          Changed startup string to 1.02
//Version:  1.01
//Comments: Added support for SimpliciTI 1.0.3
//          Changed RSSI read method
//          Added 3 digit temperature output for 100+F
//          Changed startup string to 1.01
//Version:  1.00
//Comments: Initial Release Version
//******************************************************************************
#include <string.h>
#include "bsp.h"
#include "mrfi.h"
#include "bsp_leds.h"
#include "bsp_buttons.h"
#include "nwk_types.h"
#include "nwk_api.h"
#include "nwk_frame.h"
#include "nwk.h"
#include "virtual_com_cmds.h"

//#include "app_remap_led.h"

#ifndef APP_AUTO_ACK
#error ERROR: Must define the macro APP_AUTO_ACK for this application.
#endif

void toggleLED(uint8_t);

/*************************** COMMENTS ON ASYNC LISTEN APPLICATION ***********************
Summary:
  This AP build includes implementation of an unknown number of end device peers in
  addition to AP functionality. In this scenario all End Devices establish a link to
  the AP and only to the AP. The AP acts as a data hub. All End Device peers are on
```

```
  the AP and not on other distinct ED platforms.

  There is still a limit to the number of peers supported on the AP that is defined
  by the macro NUM_CONNECTIONS. The AP will support NUM_CONNECTIONS or fewer peers
  but the exact number does not need to be known at build time.

  In this special but common scenario SimpliciTI restricts each End Device object to a
  single connection to the AP. If multiple logical connections are required these must
  be accommodated by supporting contexts in the application payload itself.

Solution overview:
  When a new peer connection is required the AP main loop must be notified. In essence
  the main loop polls a semaphore to know whether to begin listening for a peer Link
  request from a new End Device. There are two solutions: automatic notification and
  external notification. The only difference between the automatic notification
  solution and the external notification solution is how the listen semaphore is
  set. In the external notification solution the sempahore is set by the user when the
  AP is stimulated for example by a button press or a commend over a serial link. In the
  automatic scheme the notification is accomplished as a side effect of a new End Device
  joining.

  The Rx callback must be implemented. When the callback is invoked with a non-zero
  Link ID the handler could set a semaphore that alerts the main work loop that a
  SMPL_Receive() can be executed successfully on that Link ID.

  If the callback conveys an argument (LinkID) of 0 then a new device has joined the
  network. A SMPL_LinkListen() should be executed.

  Whether the joining device supports ED objects is indirectly inferred on the joining
  device from the setting of the NUM_CONNECTIONS macro. The value of this macro should
  be non-zero only if ED objects exist on the device. This macro is always non-zero
  for ED-only devices. But Range Extenders may or may not support ED objects. The macro
  should be be set to 0 for REs that do not also support ED objects. This prevents the
  Access Point from reserving resources for a joinng device that does not support any
  End Device Objects and it prevents the AP from executing a SMPL_LinkListen(). The
  Access Point will not ever see a Link frame if the joining device does not support
  any connections.

  Each joining device must execute a SMPL_Link() after receiving the join reply from the
  Access Point. The Access Point will be listening.

************************** END COMMENTS ON ASYNC LISTEN APPLICATION ******************/

/************  THIS SOURCE FILE REPRESENTS THE AUTOMATIC NOTIFICATION SOLUTION ************/

/* reserve space for the maximum possible peer Link IDs */
static linkID_t sLID[NUM_CONNECTIONS] = {0};
static uint8_t  sNumCurrentPeers = 0;

/* callback handler */
static uint8_t sCB(linkID_t);

/* received message handler */
static void processMessage(linkID_t, uint8_t *, uint8_t);

/* Frequency Agility helper functions */
static void    checkChangeChannel(void);
static void    changeChannel(void);

/* work loop semaphores */
static volatile uint8_t sPeerFrameSem = 0;
static volatile uint8_t sJoinSem = 0;
static volatile uint8_t sSelfMeasureSem = 0;

#ifdef FREQUENCY_AGILITY
/*     ************** BEGIN interference detection support */

#define INTERFERNCE_THRESHOLD_DBM (-70)
```

```
#define SSIZE    25
#define IN_A_ROW  3
static int8_t  sSample[SSIZE];
static uint8_t sChannel = 0;
#endif  /* FREQUENCY_AGILITY */

/* blink LEDs when channel changes... */
static volatile uint8_t sBlinky = 0;

//data for terminal output
const char splash[] = {"\r\n-------------------------------------------------\r\n
****\r\n     ****           eZ430-RF2500\r\n     ******o****    Temperature Sensor
Network\r\n********_///_****   Copyright 2009\r\n ******/_//_/*****  Texas Instruments
Incorporated\r\n  ** ***(__/*****   All rights reserved.\r\n      *********
SimpliciTI1.1.0\r\n       *****\r\n        ***\r\n-----------------------------------
---------------\r\n"};
volatile int * tempOffset = (int *)0x10F4;

__interrupt void ADC10_ISR(void);
__interrupt void Timer_A (void);

/*     ************** END interference detection support                        */

#define SPIN_ABOUT_A_QUARTER_SECOND   NWK_DELAY(250)

void main (void)
{
  bspIState_t intState;

  memset(sSample, 0x0, sizeof(sSample));

  BSP_Init();

  BCSCTL3 |= LFXT1S_2;                      // LFXT1 = VLO
  TACCTL0 = CCIE;                           // TACCR0 interrupt enabled
  TACCR0 = 12000;                           // ~1 second
  TACTL = TASSEL_1 + MC_1;                  // ACLK, upmode

  COM_Init();
  //Transmit splash screen and network init notification
  TXString( (char*)splash, sizeof splash);
  TXString( "\r\nInitializing Network....", 26 );

  SMPL_Init(sCB);

  // network initialized
  TXString( "Done\r\n", 6);

  /* green and red LEDs on solid to indicate waiting for a Join. */
  if (!BSP_LED2_IS_ON())
  {
    toggleLED(2);
  }
  if (!BSP_LED1_IS_ON())
  {
    toggleLED(1);
  }

  /* main work loop */
  while (1)
  {
    /* Wait for the Join semaphore to be set by the receipt of a Join frame from a
     * device that supports an End Device.
     *
     * An external method could be used as well. A button press could be connected
     * to an ISR and the ISR could set a semaphore that is checked by a function
     * call here, or a command shell running in support of a serial connection
     * could set a semaphore that is checked by a function call.
```

```
 */
if (sJoinSem && (sNumCurrentPeers < NUM_CONNECTIONS))
{
  /* listen for a new connection */
  while (1)
  {
    if (SMPL_SUCCESS == SMPL_LinkListen(&sLID[sNumCurrentPeers]))
    {
      break;
    }
    /* Implement fail-to-link policy here. otherwise, listen again. */
  }

  sNumCurrentPeers++;

  BSP_ENTER_CRITICAL_SECTION(intState);
  sJoinSem--;
  BSP_EXIT_CRITICAL_SECTION(intState);
}


// if it is time to measure our own temperature...
if(sSelfMeasureSem)
{
  char msg [6];
  char addr[] = {"HUB0"};
  char rssi[] = {"000"};
  int degC, volt;
  volatile long temp;
  int results[2];

  ADC10CTL1 = INCH_10 + ADC10DIV_4;     // Temp Sensor ADC10CLK/5
  ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON + ADC10IE + ADC10SR;
  for( degC = 240; degC > 0; degC-- );  // delay to allow reference to settle
  ADC10CTL0 |= ENC + ADC10SC;           // Sampling and conversion start
  __bis_SR_register(CPUOFF + GIE);      // LPM0 with interrupts enabled
  results[0] = ADC10MEM;

  ADC10CTL0 &= ~ENC;

  ADC10CTL1 = INCH_11;                  // AVcc/2
  ADC10CTL0 = SREF_1 + ADC10SHT_2 + REFON + ADC10ON + ADC10IE + REF2_5V;
  for( degC = 240; degC > 0; degC-- );  // delay to allow reference to settle
  ADC10CTL0 |= ENC + ADC10SC;           // Sampling and conversion start
  __bis_SR_register(CPUOFF + GIE);      // LPM0 with interrupts enabled
  results[1] = ADC10MEM;
  ADC10CTL0 &= ~ENC;
  ADC10CTL0 &= ~(REFON + ADC10ON);      // turn off A/D to save power

  // oC = ((A10/1024)*1500mV)-986mV)*1/3.55mV = A10*423/1024 - 278
  // the temperature is transmitted as an integer where 32.1 = 321
  // hence 4230 instead of 423
  temp = results[0];
  degC = (((temp - 673) * 4230) / 1024);
  if( (*tempOffset) != 0xFFFF )
  {
    degC += (*tempOffset);
  }

  temp = results[1];
  volt = (temp*25)/512;

  msg[0] = degC&0xFF;
  msg[1] = (degC>>8)&0xFF;
  msg[2] = volt;
  transmitDataString(1, addr, rssi, msg );
  BSP_TOGGLE_LED1();
  sSelfMeasureSem = 0;
```

```
    }

    /* Have we received a frame on one of the ED connections?
     * No critical section -- it doesn't really matter much if we miss a poll
     */
    if (sPeerFrameSem)
    {
      uint8_t     msg[MAX_APP_PAYLOAD], len, i;

      /* process all frames waiting */
      for (i=0; i<sNumCurrentPeers; ++i)
      {
        if (SMPL_SUCCESS == SMPL_Receive(sLID[i], msg, &len))
        {
          ioctlRadioSiginfo_t sigInfo;

          processMessage(sLID[i], msg, len);

          sigInfo.lid = sLID[i];

          SMPL_Ioctl(IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SIGINFO, (void *)&sigInfo);

          transmitData( i, sigInfo.sigInfo.rssi, (char*)msg );
          BSP_TOGGLE_LED2();

          BSP_ENTER_CRITICAL_SECTION(intState);
          sPeerFrameSem--;
          BSP_EXIT_CRITICAL_SECTION(intState);
        }
      }
    }
    if (BSP_BUTTON1())
    {
      SPIN_ABOUT_A_QUARTER_SECOND;  /* debounce */
      changeChannel();
    }
    else
    {
      checkChangeChannel();
    }
    BSP_ENTER_CRITICAL_SECTION(intState);
    if (sBlinky)
    {
      if (++sBlinky >= 0xF)
      {
        sBlinky = 1;
        toggleLED(1);
        toggleLED(2);
      }
    }
    BSP_EXIT_CRITICAL_SECTION(intState);
  }

}

void toggleLED(uint8_t which)
{
  if (1 == which)
  {
    BSP_TOGGLE_LED1();
  }
  else if (2 == which)
  {
    BSP_TOGGLE_LED2();
  }

  return;
}
```

```
/* Runs in ISR context. Reading the frame should be done in the */
/* application thread not in the ISR thread. */
static uint8_t sCB(linkID_t lid)
{
  if (lid)
  {
    sPeerFrameSem++;
    sBlinky = 0;
  }
  else
  {
    sJoinSem++;
  }

  /* leave frame to be read by application. */
  return 0;
}

static void processMessage(linkID_t lid, uint8_t *msg, uint8_t len)
{
  /* do something useful */
  if (len)
  {
    toggleLED(*msg);
  }
  return;
}

static void changeChannel(void)
{
#ifdef FREQUENCY_AGILITY
  freqEntry_t freq;

  if (++sChannel >= NWK_FREQ_TBL_SIZE)
  {
    sChannel = 0;
  }
  freq.logicalChan = sChannel;
  SMPL_Ioctl(IOCTL_OBJ_FREQ, IOCTL_ACT_SET, &freq);
  BSP_TURN_OFF_LED1();
  BSP_TURN_OFF_LED2();
  sBlinky = 1;
#endif
  return;
}

/* implement auto-channel-change policy here... */
static void  checkChangeChannel(void)
{
#ifdef FREQUENCY_AGILITY
  int8_t dbm, inARow = 0;

  uint8_t i;

  memset(sSample, 0x0, SSIZE);
  for (i=0; i<SSIZE; ++i)
  {
    /* quit if we need to service an app frame */
    if (sPeerFrameSem || sJoinSem)
    {
      return;
    }
    NWK_DELAY(1);
    SMPL_Ioctl(IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_RSSI, (void *)&dbm);
    sSample[i] = dbm;

    if (dbm > INTERFERNCE_THRESHOLD_DBM)
```

```
    {
      if (++inARow == IN_A_ROW)
      {
        changeChannel();
        break;
      }
    }
    else
    {
      inARow = 0;
    }
  }
#endif
  return;
}

/*-------------------------------------------------------------------------
* ADC10 interrupt service routine
-------------------------------------------------------------------------*/
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
  __bic_SR_register_on_exit(CPUOFF);        // Clear CPUOFF bit from 0(SR)
}

/*-------------------------------------------------------------------------
* Timer A0 interrupt service routine
-------------------------------------------------------------------------*/
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
  sSelfMeasureSem = 1;
}
```

7.2. Código original de las terminales.

```
//*****************************************************************************
// THIS PROGRAM IS PROVIDED "AS IS". TI MAKES NO WARRANTIES OR
// REPRESENTATIONS, EITHER EXPRESS, IMPLIED OR STATUTORY,
// INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS
// FOR A PARTICULAR PURPOSE, LACK OF VIRUSES, ACCURACY OR
// COMPLETENESS OF RESPONSES, RESULTS AND LACK OF NEGLIGENCE.
// TI DISCLAIMS ANY WARRANTY OF TITLE, QUIET ENJOYMENT, QUIET
// POSSESSION, AND NON-INFRINGEMENT OF ANY THIRD PARTY
// INTELLECTUAL PROPERTY RIGHTS WITH REGARD TO THE PROGRAM OR
// YOUR USE OF THE PROGRAM.
//
// IN NO EVENT SHALL TI BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
// CONSEQUENTIAL OR INDIRECT DAMAGES, HOWEVER CAUSED, ON ANY
// THEORY OF LIABILITY AND WHETHER OR NOT TI HAS BEEN ADVISED
// OF THE POSSIBILITY OF SUCH DAMAGES, ARISING IN ANY WAY OUT
// OF THIS AGREEMENT, THE PROGRAM, OR YOUR USE OF THE PROGRAM.
// EXCLUDED DAMAGES INCLUDE, BUT ARE NOT LIMITED TO, COST OF
// REMOVAL OR REINSTALLATION, COMPUTER TIME, LABOR COSTS, LOSS
// OF GOODWILL, LOSS OF PROFITS, LOSS OF SAVINGS, OR LOSS OF
// USE OR INTERRUPTION OF BUSINESS. IN NO EVENT WILL TI'S
// AGGREGATE LIABILITY UNDER THIS AGREEMENT OR ARISING OUT OF
// YOUR USE OF THE PROGRAM EXCEED FIVE HUNDRED DOLLARS
// (U.S.$500).
```

```
//
// Unless otherwise stated, the Program written and copyrighted
// by Texas Instruments is distributed as "freeware".  You may,
// only under TI's copyright in the Program, use and modify the
// Program without any charge or restriction.  You may
// distribute to third parties, provided that you transfer a
// copy of this license to the third party and the third party
// agrees to these terms by its first use of the Program. You
// must reproduce the copyright notice and any other legend of
// ownership on each copy or partial copy, of the Program.
//
// You acknowledge and agree that the Program contains
// copyrighted material, trade secrets and other TI proprietary
// information and is protected by copyright laws,
// international copyright treaties, and trade secret laws, as
// well as other intellectual property laws.  To protect TI's
// rights in the Program, you agree not to decompile, reverse
// engineer, disassemble or otherwise translate any object code
// versions of the Program to a human-readable form.  You agree
// that in no event will you alter, remove or destroy any
// copyright notice included in the Program.  TI reserves all
// rights not specifically granted under this license. Except
// as specifically provided herein, nothing in this agreement
// shall be construed as conferring by implication, estoppel,
// or otherwise, upon you, any license or other right under any
// TI patents, copyrights or trade secrets.
//
// You may not use the Program in non-TI devices.
//
//******************************************************************************
//   eZ430-RF2500 Temperature Sensor End Device
//
//   Description: This is the End Device software for the eZ430-2500RF
//                Temperature Sensing demo
//
//
//   L. Westlund
//   Version    1.02
//   Texas Instruments, Inc
//   November 2007
//   Built with IAR Embedded Workbench Version: 4.09A
//******************************************************************************
//Change Log:
//******************************************************************************
//Version:  1.03
//Comments: Added support for SimpliciTI 1.1.0
//          Added support for Code Composer Studio
//          Added security (Enabled with -DSMPL_SECURE in smpl_nwk_config.dat)
//          Added acknowledgement (Enabled with -DAPP_AUTO_ACK in smpl_nwk_config.dat)
//          Based the modifications on the AP_as_Data_Hub example code
//Version:  1.02
//Comments: Changed Port toggling to abstract method
//          Fixed comment typos
//Version:  1.01
//Comments: Added support for SimpliciTI 1.0.3
//          Added Flash storage/check of Random address
//          Moved LED toggle to HAL
//Version:  1.00
//Comments: Initial Release Version
//******************************************************************************
#define I_WANT_TO_CHANGE_DEFAULT_ROM_DEVICE_ADDRESS_PSEUDO_CODE
#include "bsp.h"
#include "mrfi.h"
#include "nwk_types.h"
#include "nwk_api.h"
#include "bsp_leds.h"
#include "bsp_buttons.h"
#include "vlo_rand.h"
```

```
#ifndef APP_AUTO_ACK
#error ERROR: Must define the macro APP_AUTO_ACK for this application.
#endif

void toggleLED(uint8_t);

static void linkTo(void);

static linkID_t sLinkID1 = 0;

#define SPIN_ABOUT_A_SECOND   NWK_DELAY(1000)
#define SPIN_ABOUT_A_QUARTER_SECOND   NWK_DELAY(250)

/* How many times to try a Tx and miss an acknowledge before doing a scan */
#define MISSES_IN_A_ROW  2

void createRandomAddress(void);

volatile int * tempOffset = (int *)0x10F4; // Temperature offset set at production
char * Flash_Addr = (char *)0x10F0;        // Initialize radio address location

__interrupt void ADC10_ISR(void);
__interrupt void Timer_A (void);

/* work loop semaphores */
static volatile uint8_t sSelfMeasureSem = 0;

void main (void)
{
  addr_t lAddr;

  BSP_Init();

  if(Flash_Addr[0] == 0xFF && Flash_Addr[1] == 0xFF &&
     Flash_Addr[2] == 0xFF && Flash_Addr[3] == 0xFF )
  {
    createRandomAddress(); // set Random device address at initial startup
  }
  lAddr.addr[0] = Flash_Addr[0];
  lAddr.addr[1] = Flash_Addr[1];
  lAddr.addr[2] = Flash_Addr[2];
  lAddr.addr[3] = Flash_Addr[3];
  SMPL_Ioctl(IOCTL_OBJ_ADDR, IOCTL_ACT_SET, &lAddr);

  /* Keep trying to join (a side effect of successful initialization) until
   * successful. Toggle LEDS to indicate that joining has not occurred.
   */
  while (SMPL_SUCCESS != SMPL_Init(0))
  {
    toggleLED(1);
    toggleLED(2);
    SPIN_ABOUT_A_SECOND;
  }

  /* LEDs on solid to indicate successful join. */
  if (!BSP_LED2_IS_ON())
  {
    toggleLED(2);
  }
  if (!BSP_LED1_IS_ON())
  {
    toggleLED(1);
  }

  BCSCTL3 |= LFXT1S_2;                      // LFXT1 = VLO
  TACCTL0 = CCIE;                           // TACCR0 interrupt enabled
  TACCR0 = 12000;                           // ~ 1 sec
```

```
  TACTL = TASSEL_1 + MC_1;                  // ACLK, upmode

  /* Unconditional link to AP which is listening due to successful join. */
  linkTo();

  while (1) ;
}

static void linkTo()
{
  uint8_t     msg[3];
  uint8_t     misses, done;

  /* Keep trying to link... */
  while (SMPL_SUCCESS != SMPL_Link(&sLinkID1))
  {
    toggleLED(1);
    toggleLED(2);
    SPIN_ABOUT_A_SECOND;
  }

  /* Turn off LEDs. */
  if (BSP_LED2_IS_ON())
  {
    toggleLED(2);
  }
  if (BSP_LED1_IS_ON())
  {
    toggleLED(1);
  }

  SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SLEEP, 0);

  while (1)
  {


    __bis_SR_register(LPM3_bits+GIE);  // LPM3 with interrupts enabled

    if (sSelfMeasureSem) {
      volatile long temp;
      int degC, volt;
      int results[2];
      uint8_t      noAck;
      smplStatus_t rc;

      /* get radio ready...awakens in idle state */
      SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_AWAKE, 0);

      ADC10CTL1 = INCH_10 + ADC10DIV_4;       // Temp Sensor ADC10CLK/5
      ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON + ADC10IE + ADC10SR;
      for( degC = 240; degC > 0; degC-- );    // delay to allow reference to settle
      ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
      __bis_SR_register(CPUOFF + GIE);        // LPM0 with interrupts enabled

      results[0] = ADC10MEM;
      ADC10CTL0 &= ~ENC;
      ADC10CTL1 = INCH_11;                     // AVcc/2
      ADC10CTL0 = SREF_1 + ADC10SHT_2 + REFON + ADC10ON + ADC10IE + REF2_5V;
      for( degC = 240; degC > 0; degC-- );    // delay to allow reference to settle
      ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
      __bis_SR_register(CPUOFF + GIE);        // LPM0 with interrupts enabled
      results[1] = ADC10MEM;
      ADC10CTL0 &= ~ENC;
      ADC10CTL0 &= ~(REFON + ADC10ON);        // turn off A/D to save power

      // oC = ((A10/1024)*1500mV)-986mV)*1/3.55mV = A10*423/1024 - 278
      // the temperature is transmitted as an integer where 32.1 = 321
```

```
      // hence 4230 instead of 423
      temp = results[0];
      degC = ((temp - 673) * 4230) / 1024;
      if( (*tempOffset) != 0xFFFF )
      {
        degC += (*tempOffset);
      }
      /* message format,  UB = upper Byte, LB = lower Byte
      -------------------------------
      |degC LB | degC UB |  volt LB |
      -------------------------------
         0         1          2
      */

      temp = results[1];
      volt = (temp*25)/512;
      msg[0] = degC&0xFF;
      msg[1] = (degC>>8)&0xFF;
      msg[2] = volt;

      done = 0;
      while (!done)
      {
        noAck = 0;

        /* Try sending message MISSES_IN_A_ROW times looking for ack */
        for (misses=0; misses < MISSES_IN_A_ROW; ++misses)
        {
          if (SMPL_SUCCESS == (rc=SMPL_SendOpt(sLinkID1, msg, sizeof(msg), SMPL_TXOPTION_ACKREQ)))
          {
            /* Message acked. We're done. Toggle LED 1 to indicate ack received. */
            toggleLED(1);  // Toggle On LED1
            __delay_cycles(2000);
            toggleLED(1);
            break;
          }
          if (SMPL_NO_ACK == rc)
          {
            /* Count ack failures. Could also fail becuase of CCA and
             * we don't want to scan in this case.
             */
            noAck++;
          }
        }
        if (MISSES_IN_A_ROW == noAck)
        {
          /* Message not acked. Toggle LED 2. */
          toggleLED(2);  // Turn On LED2
          __delay_cycles(2000);
          toggleLED(2);
#ifdef FREQUENCY_AGILITY
          /* Assume we're on the wrong channel so look for channel by
           * using the Ping to initiate a scan when it gets no reply. With
           * a successful ping try sending the message again. Otherwise,
           * for any error we get we will wait until the next button
           * press to try again.
           */
          if (SMPL_SUCCESS != SMPL_Ping(sLinkID1))
          {
            done = 1;
          }
#else
          done = 1;
#endif  /* FREQUENCY_AGILITY */
        }
        else
        {
          /* Got the ack or we don't care. We're done. */
```

```
          done = 1;
        }
      }

      /* radio back to sleep */
      SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SLEEP, 0);
    }
  }
}


void toggleLED(uint8_t which)
{
  if (1 == which)
  {
    BSP_TOGGLE_LED1();
  }
  else if (2 == which)
  {
    BSP_TOGGLE_LED2();
  }
  return;
}

void createRandomAddress()
{
  unsigned int rand, rand2;
  do
  {
    rand = TI_getRandomIntegerFromVLO();    // first byte can not be 0x00 of 0xFF
  }
  while( (rand & 0xFF00)==0xFF00 || (rand & 0xFF00)==0x0000 );
  rand2 = TI_getRandomIntegerFromVLO();

  BCSCTL1 = CALBC1_1MHZ;                    // Set DCO to 1MHz
  DCOCTL = CALDCO_1MHZ;
  FCTL2 = FWKEY + FSSEL0 + FN1;             // MCLK/3 for Flash Timing Generator
  FCTL3 = FWKEY + LOCKA;                    // Clear LOCK & LOCKA bits
  FCTL1 = FWKEY + WRT;                      // Set WRT bit for write operation

  Flash_Addr[0]=(rand>>8) & 0xFF;
  Flash_Addr[1]=rand & 0xFF;
  Flash_Addr[2]=(rand2>>8) & 0xFF;
  Flash_Addr[3]=rand2 & 0xFF;

  FCTL1 = FWKEY;                            // Clear WRT bit
  FCTL3 = FWKEY + LOCKA + LOCK;             // Set LOCK & LOCKA bit
}

/*-------------------------------------------------------------------------
* ADC10 interrupt service routine
-------------------------------------------------------------------------*/
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
  __bic_SR_register_on_exit(CPUOFF);        // Clear CPUOFF bit from 0(SR)
}

/*-------------------------------------------------------------------------
* Timer A0 interrupt service routine
-------------------------------------------------------------------------*/
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
  sSelfMeasureSem = 1;
  __bic_SR_register_on_exit(LPM3_bits);        // Clear LPM3 bit from 0(SR)
}
```

7.3. Código del puerto virtual COM de comunicación

```
/*-------------------------------------------------------------------------
 *  Demo Application for SimpliciTI
 *
 *  L. Friedman
 *  Texas Instruments, Inc.
 *-------------------------------------------------------------------------
 */

/*******************************************************************************************
  Copyright 2007-2009 Texas Instruments Incorporated. All rights reserved.

  IMPORTANT: Your use of this Software is limited to those specific rights granted under
  the terms of a software license agreement between the user who downloaded the software,
  his/her employer (which must be your employer) and Texas Instruments Incorporated (the
  "License"). You may not use this Software unless you agree to abide by the terms of the
  License. The License limits your use, and you acknowledge, that the Software may not be
  modified, copied or distributed unless embedded on a Texas Instruments microcontroller
  or used solely and exclusively in conjunction with a Texas Instruments radio frequency
  transceiver, which is integrated into your product. Other than for the foregoing purpose,
  you may not use, reproduce, copy, prepare derivative works of, modify, distribute,
  perform, display or sell this Software and/or its documentation for any purpose.

  YOU FURTHER ACKNOWLEDGE AND AGREE THAT THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS"
  WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY
  WARRANTY OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
  IN NO EVENT SHALL TEXAS INSTRUMENTS OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER CONTRACT,
  NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE
  THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO ANY
  INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST
  DATA, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY
  THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

  Should you have any questions regarding your right to use this Software,
  contact Texas Instruments Incorporated at www.TI.com.
*******************************************************************************************/
#include <string.h>
#include "bsp.h"
#include "virtual_com_cmds.h"

static char verboseMode = 1;
static char degCMode = 0;

/*****************************************************************************/
// End Virtual Com Port Communication
/*****************************************************************************/
void COM_Init(void)
{
  P3SEL |= 0x30;                            // P3.4,5 = USCI_A0 TXD/RXD
  UCA0CTL1 = UCSSEL_2;                      // SMCLK
  UCA0BR0 = 0x41;                           // 9600 from 8Mhz
  UCA0BR1 = 0x3;
  UCA0MCTL = UCBRS_2;
  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
  IE2 |= UCA0RXIE;                          // Enable USCI_A0 RX interrupt
  __enable_interrupt();
}
```

```
void TXString( char* string, int length )
{
  int pointer;
  for( pointer = 0; pointer < length; pointer++)
  {
    volatile int i;
    UCA0TXBUF = string[pointer];
    while (!(IFG2&UCA0TXIFG));              // USCI_A0 TX buffer ready?
  }
}

void transmitDataString(char data_mode, char addr[4],char rssi[3], char msg[MESSAGE_LENGTH] )
{
  char temp_string[] = {" XX.XC"};
  int temp = msg[0] + (msg[1]<<8);

  if(!(data_mode & degCMode))
  {
    temp = (((float)temp)*1.8)+320;
    temp_string[5] = 'F';
  }
  if( temp < 0 )
  {
    temp_string[0] = '-';
    temp = temp * -1;
  }
  else if( ((temp/1000)%10) != 0 )
  {
    temp_string[0] = '0'+((temp/1000)%10);
  }
  temp_string[4] = '0'+(temp%10);
  temp_string[2] = '0'+((temp/10)%10);
  temp_string[1] = '0'+((temp/100)%10);

  if(data_mode & verboseMode)
  {
    char output_verbose[] = {"\r\nNode:XXXX,Temp:-XX.XC,Battery:X.XV,Strength:XXX%,RE:no "};

    output_verbose[46] = rssi[2];
    output_verbose[47] = rssi[1];
    output_verbose[48] = rssi[0];

    output_verbose[17] = temp_string[0];
    output_verbose[18] = temp_string[1];
    output_verbose[19] = temp_string[2];
    output_verbose[20] = temp_string[3];
    output_verbose[21] = temp_string[4];
    output_verbose[22] = temp_string[5];

    output_verbose[32] = '0'+(msg[2]/10)%10;
    output_verbose[34] = '0'+(msg[2]%10);
    output_verbose[7] = addr[0];
    output_verbose[8] = addr[1];
    output_verbose[9] = addr[2];
    output_verbose[10] = addr[3];
    TXString(output_verbose, sizeof output_verbose );
  }
  else
  {
    char output_short[] = {"\r\n$ADDR,-XX.XC,V.C,RSI,N#"};

    output_short[19] = rssi[2];
    output_short[20] = rssi[1];
    output_short[21] = rssi[0];


    output_short[8] = temp_string[0];
    output_short[9] = temp_string[1];
```

```
    output_short[10] = temp_string[2];
    output_short[11] = temp_string[3];
    output_short[12] = temp_string[4];
    output_short[13] = temp_string[5];

    output_short[15] = '0'+(msg[2]/10)%10;
    output_short[17] = '0'+(msg[2]%10);
    output_short[3] = addr[0];
    output_short[4] = addr[1];
    output_short[5] = addr[2];
    output_short[6] = addr[3];
    TXString(output_short, sizeof output_short );
  }
}

void transmitData(int addr, signed char rssi,  char msg[MESSAGE_LENGTH] )
{
  char addrString[4];
  char rssiString[3];
  volatile signed int rssi_int;

  addrString[0] = '0';
  addrString[1] = '0';
  addrString[2] = '0'+(((addr+1)/10)%10);
  addrString[3] = '0'+((addr+1)%10);
  rssi_int = (signed int) rssi;
  rssi_int = rssi_int+128;
  rssi_int = (rssi_int*100)/256;
  rssiString[0] = '0'+(rssi_int%10);
  rssiString[1] = '0'+((rssi_int/10)%10);
  rssiString[2] = '0'+((rssi_int/100)%10);

  transmitDataString( degCMode, addrString, rssiString, msg );
}

/*-------------------------------------------------------------------------
* USCIA interrupt service routine
-------------------------------------------------------------------------*/
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
  char rx = UCA0RXBUF;
  if ( rx == 'V' || rx == 'v' )
  {
    verboseMode = 1;
  }
  else if ( rx == 'M' || rx == 'm' )
  {
    verboseMode = 0;
  }
  else if ( rx == 'F' || rx == 'f' )
  {
    degCMode = 0;
  }
  else if ( rx == 'C' || rx == 'c' )
  {
    degCMode = 1;
  }
}
```

7.4. Código original #include <string.h>

```
/*-------------------------------------------------------------------------
 *  Demo Application for SimpliciTI
 *
 *  L. Friedman
 *  Texas Instruments, Inc.
 *-------------------------------------------------------------------------
 */

/*******************************************************************************************
  Copyright 2007-2009 Texas Instruments Incorporated. All rights reserved.

  IMPORTANT: Your use of this Software is limited to those specific rights granted under
  the terms of a software license agreement between the user who downloaded the software,
  his/her employer (which must be your employer) and Texas Instruments Incorporated (the
  "License"). You may not use this Software unless you agree to abide by the terms of the
  License. The License limits your use, and you acknowledge, that the Software may not be
  modified, copied or distributed unless embedded on a Texas Instruments microcontroller
  or used solely and exclusively in conjunction with a Texas Instruments radio frequency
  transceiver, which is integrated into your product. Other than for the foregoing purpose,
  you may not use, reproduce, copy, prepare derivative works of, modify, distribute,
  perform, display or sell this Software and/or its documentation for any purpose.

  YOU FURTHER ACKNOWLEDGE AND AGREE THAT THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS"
  WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY
  WARRANTY OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
  IN NO EVENT SHALL TEXAS INSTRUMENTS OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER CONTRACT,
  NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE
  THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO ANY
  INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST
  DATA, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY
  THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

  Should you have any questions regarding your right to use this Software,
  contact Texas Instruments Incorporated at www.TI.com.
*******************************************************************************************/
#include <string.h>

/*****************************************************************************/
// Virtual Com Port Communication
/*****************************************************************************/
#define MESSAGE_LENGTH 3

void COM_Init(void);
void TXString( char* string, int length );
void transmitData(int addr, signed char rssi,  char msg[MESSAGE_LENGTH] );
void transmitDataString(char data_mode, char addr[4],char rssi[3], char msg[MESSAGE_LENGTH]);
```

7.5. Código original <vlo_rand.h>

```
#ifndef VLO_Rand_Library
#define VLO_Rand_Library

extern int TI_getRandomIntegerFromADC( void );
extern int TI_getRandomIntegerFromVLO( void );

#endif
```