LoRa

comunicación inalámbrica de largo alcance y bajo consumo

Integrantes: Juan Martín Chiale y Mariana Siniscalchi
Tutor: Javier Schandy

Facultad de ingeniería

Resumen

LoRa es un protocolo de comunicación inalámbrica de baja potencia y largo alcance. Está enfocado a pequeños dispositivos autónomos. En este proyecto se formó una red de dos nodos utilizando el protocolo LoRa en la banda de 868 MHz. Cada nodo está basado en el launchpad MSP4030 que incluye el microprocesador MSP430G2553 y un módulo inalámbrico LoRa SX1276. Uno de los nodos, dotado de un sensor de temperatura, toma una medida periódicamente y envía el dato al otro nodo. Este último a su vez envía a través de la UART el dato recibido a un PC, que sirve de concentrador de la información e interfaz de usuario. Se utilizó una arquitectura Round Robin con interrupciones.Se obtuvieron apenas 10 m de alcance, mucho menor a 5 km especificados por el fabricante para entornos urbanos. Se obtuvieron 15.8 mA de consumo máximo, donde 1.7 mA corresponden al sensor de temperatura activo y la diferencia a la radio en recepción y el launchpad en su funcionamiento normal. El throughput de datos está limitado por la implementación del software a 8 bps, pero es adecuado al caso de aplicación.

Índice

Introducción **4**

Objetivos **4**

Alcance **5**

Diseño **5**

Descripción del sistema 5

Consideraciones de hardware 6

Implementación **7**

Software del nodoTemp 7

Módulo main 8

Módulo radio 8

Módulo sx1276 9

Modificaciones de esta versión del módulo sx1276 9

Módulo spi 10

Sensor de temperatura 10

Módulo I2C 11

Módulo temp 11

Módulo timer 11

Módulo mcu 12

Software del nodoBase 12

Módulo main 12

Módulo uart 12

Pruebas **13**

Pruebas de la comunicación inalámbrica 13

Medida de alcance 13

Caracterización del consumo y velocidad de transmisión de datos 13

Conclusiones **15**

Referencias **16**

Anexo **17**

Anexo I: Conceptos del curso aplicados al proyecto 17

Anexo II: Especificación del proyecto 20

Anexo III: Planificación del proyecto 26

Introducción

Con el avance de la tecnología, el número de dispositivos y sensores que se conectarán y comunicarán entre sí incrementará considerablemente, creándose así grandes redes inteligentes, lo que hoy se conoce por el internet de las cosas IoT. Es por esto que una serie de líderes mundiales de la industria de las comunicaciones crearon “LoRa-Alliance” con el objetivo de desarrollar una nueva tecnología para lograr una comunicación de dispositivos IoT de forma más eficiente. Esta organización tiene como objetivo estandarizar una tecnología de comunicación de red que cumpla con una serie de características necesarias para IoT.

LoRa es un protocolo de comunicación por radio de baja potencia, su propósito es conseguir establecer conexiones de largo alcance [1]. Estas comunicaciones están enfocadas para pequeños dispositivos que desean transmitir poca información y por tanto sin necesidad de grandes velocidades pero con la intención de emplear el menor consumo de energía, para así poder tener un mayor tiempo de vida con pequeñas baterías. La intención es establecer una red de dispositivos conectados que se les denominará nodos, los cuales establecerán un enlace inalámbrico con un elemento más potente capaz de comunicarse con todos estos elementos de manera gestionada. El alcance estimado es de 15 km en zonas despobladas y unos 5 km para ciudades [2].

Este proyecto implica el diseño e implementación de una aplicación sencilla para probar y caracterizar el protocolo LoRa. Se establece una comunicación inalámbrica entre dos nodos bajo el protocolo LoRa. Uno de los nodos, dotado de un sensor de temperatura, toma medidas de temperatura periódicamente y las transmite a través del enlace inalámbrico al otro nodo. Este otro nodo opera como base y simplemente recibe los datos de temperatura remota. Para verificar el funcionamiento, se conecta el nodo base a un PC y se despliegan en pantalla los mensajes recibidos.

En estas condiciones se caracteriza el sistema, en términos del alcance del enlace, el consumo de corriente y el throughput de datos en ese caso. Se analizan los resultados obtenidos.

Objetivos

Se estudiarán en profundidad todos los modos de funcionamiento de la radio y se elegirá la configuración óptima de los registros para la aplicación planteada, logrando comunicar dos nodos utilizando el protocolo LoRa. Se desarrollará una aplicación que se comunique con un sensor de temperatura, tome medidas y las envíe periódicamente al nodo base. El nodo base recibirá los paquetes y enviará por la UART el valor de temperatura recibido, para verificar el funcionamiento. Luego se evaluará el desempeño del sistema, midiendo el alcance, consumo y velocidad de transmisión.

Alcance

Se armará una red de dos nodos, utilizando el kit de desarrollo MSP-EXP430G2 [3], basado en el microcontrolador de Texas Instruments MSP430G2553 [4]. Se utilizará el módulo

LoRa inalámbrico SX1276 [5]. Se deberá resolver la comunicación SPI con el módulo LoRa para configurarlo correctamente y enviar y recibir paquetes. Se deberá resolver la comunicación I^2C con el sensor de temperatura. Se deberá dar una medida del throughput y el consumo en ese caso.

Diseño

Descripción del sistema

El sistema implementado consiste de dos nodos comunicados a través de un enlace inalámbrico bajo el protocolo LoRa, según se muestra en la Fig. 1. Cada nodo está compuesto del launchpad MSP4030 y un módulo de radio SX1276 que implementa LoRa. El launchpad tiene incorporado el microprocesador MSP430G2553 de Texas Instruments. Cuenta con un puerto USB y pines para implementar comunicación serial SPI e I^2C, entre otras prestaciones, que se utilizan en este proyecto.

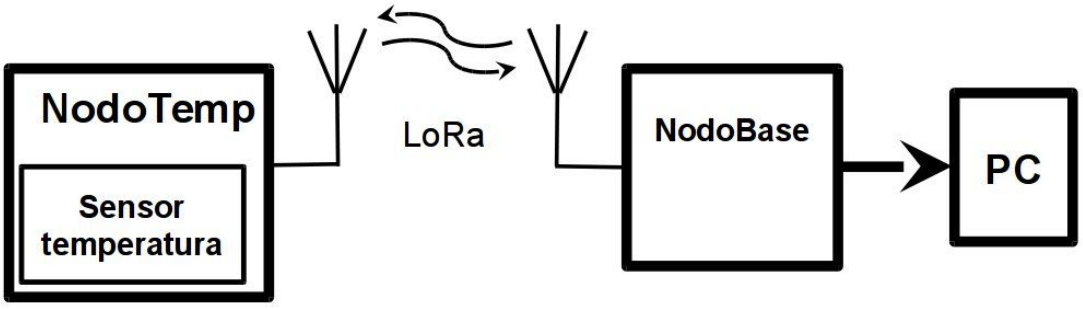
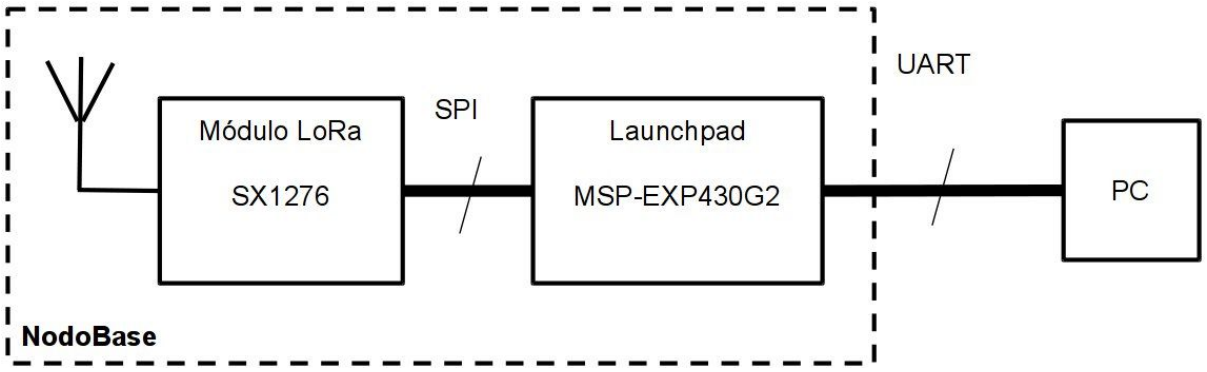


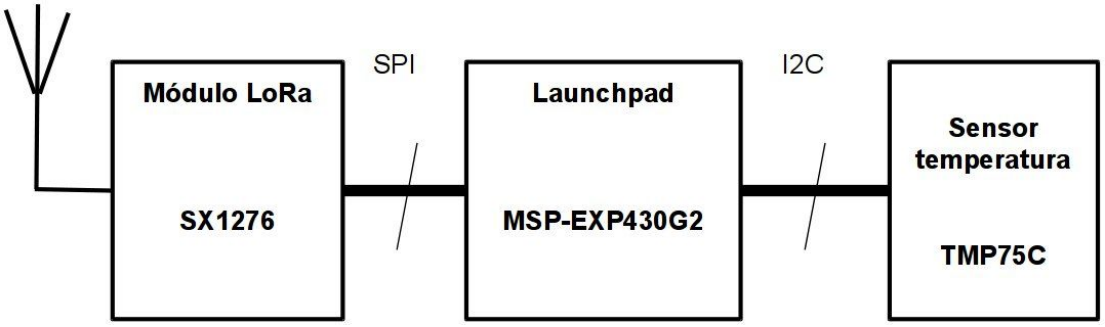
Figura 1: diagrama cualitativo del sistema implementado

El nodoTemp incorpora un sensor de temperatura (TMP75C) [6], tomando periódicamente una medida. Luego envía a través del enlace inalámbrico el dato de temperatura al nodoBase. El nodoBase implementa la comunicación UART con un PC a través del puerto USB del launchpad. El contenido de todo paquete que recibe el nodoBase, lo comunica por la UART. A su vez, envía un paquete de reconocimiento, a través del enlace inalámbrico, al nodoTemp. Por lo tanto el nodoBase debe estar continuamente esperando recibir un nuevo paquete a través del enlace inalámbrico.

El microprocesador y el módulo SX1276 se comunican vía SPI. El sensor de temperatura y el microprocesador se comunican vía I2C. Las Fig. 2 a y b muestran un diagrama del nodoTemp y nodoBase, respectivamente.



(a) nodoBase y conexión al PC.



(b) nodoTemp.

Figura 2: diagramas de bloques de los nodos

Consideraciones de hardware

La Fig. 3 muestra el esquemático del nodoTemp. El nodoBase es similar, pero prescindiendo del sensor de temperatura e incorporando la conexión del USB del launchpad al USB de un PC.

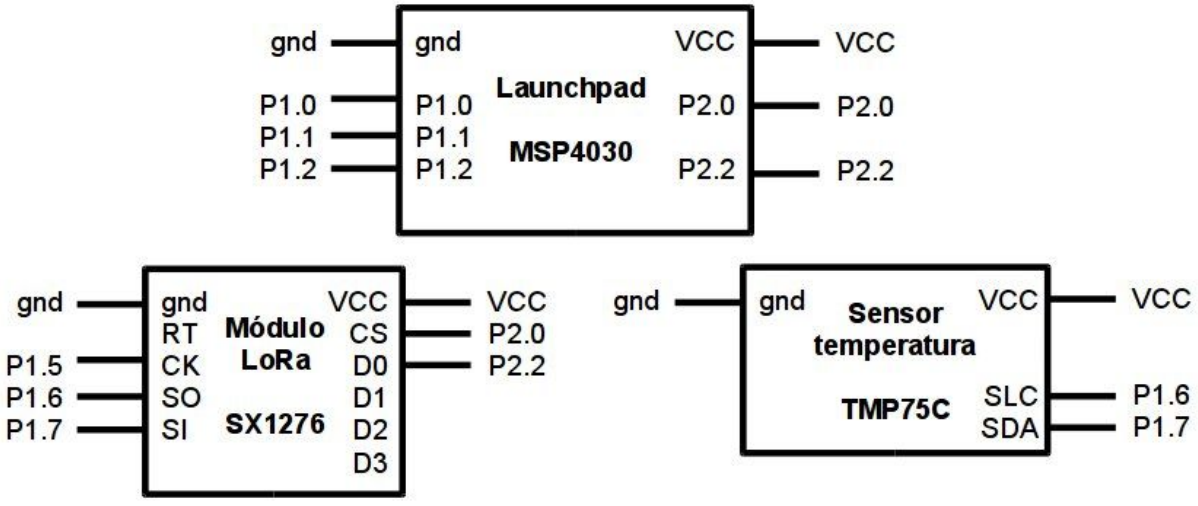


Figura 3: circuito esquemático del nodoTemp

Los pines que se omiten del launchpad, no se utilizan. Los pines 6 y 7 del puerto 1 (P1.6 y P1.7) se utilizan para la comunicación con el módulo LoRa y con el sensor de temperatura.

El pin de reset (RT) del módulo LoRa debe permanecer sin conectar, por ser una entrada de alta impedancia. En caso de querer implementar un reset por software del módulo LoRa, se debe cablear RT a un pin del launchpad. Se alimentan el módulo LoRa y el sensor de temperatura con la tensión Vcc = 3.3 V, disponible en un pin del launchpad.

Implementación

Se desarrollaron las aplicaciones para el nodoTemp y el nodoBase, utilizando la versión 6.40.2 del IAR como entorno de desarrollo. Como herramienta de debug, además del IAR, se utilizó la comunicación UART del nodo con un PC, utilizando la conexión USB del launchpad. Para esto se implementó un módulo uart que se utiliza también en la aplicación nodoBase en su funcionamiento normal. Lo que el nodo escribe en la UART se despliega en la pantalla de un PC utilizando el software Realterm.

Software del nodoTemp

Se presenta un diagrama del software del nodoTemp en la Fig. 4.

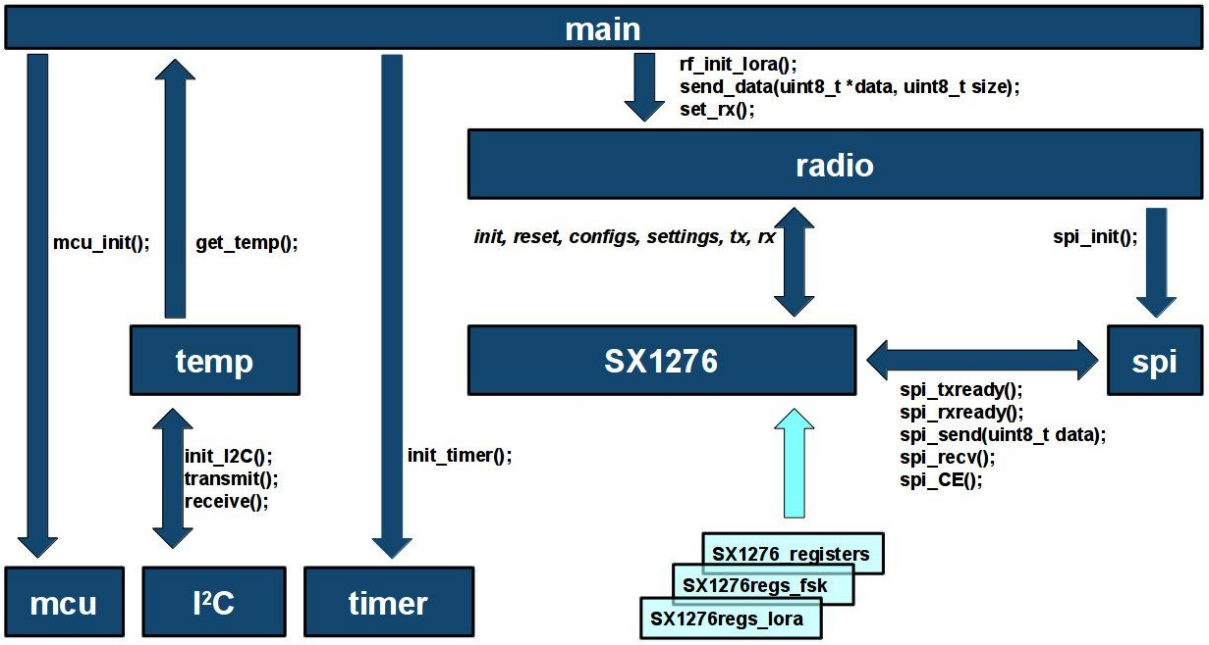


Figura 4: diagrama del software del nodoTemp.

Cada bloque del diagrama corresponde a un módulo de software, es decir un archivo *.c y un archivo *.h, en este caso. Las flechas que interconectan los módulos, dan una idea de las funciones que cada módulo disponibiliza a los demás módulos, de acuerdo a su interfaz pública. El sentido de las flechas representa hacia dónde es el flujo de datos, pero es meramente ilustrativo. Cada módulo maneja las interrupciones asociadas, es decir, implementa las rutinas de atención a la interrupción que estén relacionadas con el módulo. La altura a la que se posicionan los módulos da una idea de la jerarquía de éstos y a su vez presenta a menor altura los módulos de más bajo nivel, es decir, más cerca del hardware. Los módulos I^2C y SPI implementan la comunicación serial correspondiente, disponibilizando funciones para transmitir y recibir, independientemente de la implementación de la comunicación en sí. De la misma manera, el módulo temp, que

implementa el sensado de temperatura, utiliza las funciones del módulo I^2C para comunicarse con el sensor y disponibiliza la función *get_temp()* que se llama para obtener un nuevo dato de temperatura. El módulo I^2C está basado en un código de ejemplo disponible en línea [7]. Similarmente el módulo radio disponibiliza las funciones necesarias para configurar la radio y trasmitir y recibir paquetes a través del enlace inalámbrico, independizando la aplicación del hardware utilizado (capa de abstracción de hardware). De este modo, se podría generar un módulo radio nuevo que implemente las mismas funciones paro para un módulo diferente al SX1276, sin necesidad de modificar la aplicación. También se implementaron dos módulos, mcu y timer que permiten inicializar el microprocesador (pines, watchdog, leds) y el timer de hardware para contar tiempo. Sendos módulos disponibilizan funciones de inicialización (e implementan, por ejemplo una rutina de atención a interrupción en el caso del timer), para que la aplicación las utilice en forma transparente.

El módulo SX1276 es una nueva versión del módulo del mismo nombre disponible en Github [8], que incorpora algunas modificaciones que se explicarán. El programa de prueba está basado en las bibliotecas para LoRa también disponibles en Github [9]. Los bloques pequeños utilizados por el modulo SX1276 (SX1276_registers, SX1276regs_fsk, SX_1276regs_lora) corresponden a archivos de cabecera que asocian nombres de registros de la radio con direcciones en las que se accede a éstos, específicos de este hardware. Las funciones que disponibiliza el módulo SPI son mayoritariamente usadas por el módulo SX1276. El módulo SPI también está disponible, como un módulo del programa de prueba, y fue utilizado sin modificaciones en este proyecto.

Módulo main

Es el encargado de inicializar todo el sistema, utilizando las funciones de inicialización de los módulos timer, radio, temp y mcu. A su vez, debe declarar la bandera timeout. Luego pide un dato de temperatura, utilizando la función get_temp() del módulo temp y ejecuta un loop infinito. En este loop se verifica si el timer caducó, es decir, si la bandera timeout está alta. De ser así, se pide un nuevo dato de temperatura, se llena el buffer con el dato, se llama a la función send_data del módulo radio y se baja la bandera timeout.

La rutina de atención a la interrupción del timer, implementada en el módulo timer, se encarga de modificar la bandera timeout.

Módulo radio

Disponibiliza las funciones necesarias para inicializar la radio y trasmitir paquetes a través del enlace inalámbrico. Este módulo permite independizar la aplicación del hardware utilizado (en este caso, del módulo LoRa sx1276). La aplicación declara un buffer en el main, reservando memoria para un arreglo de tamaño fijo configurable. De acuerdo a la dinámica de la aplicación, en determinado momento se tienen datos en ese buffer, listos para ser transmitidos. Para una trasmisión, la aplicación se encarga de pasarle al módulo radio un puntero al primer dato y el largo de ese arreglo que implementa el buffer. Luego el módulo radio se sirve de las funciones del módulo sx1276 para la trasmisión.

Por otra parte, este módulo implementa una rutina de atención a interrupción. La interrupción es generada por el módulo LoRa sx1276, levantando un pin que se conecta al pin 2 del puerto 2 del launchpad. Cuando se recibe la interrupción, se ejecutan distintos

fragmentos de código según sea el modo de operación en el que está configurada la radio, recepción o transmisión, señalizado con la bandera op_mode. Esto es necesario porque el módulo LoRa sx1276 sube ese pin cuando terminó de transmitir un paquete y también cuando recibió un paquete nuevo y está listo para ser leído.

Módulo sx1276

Esta es una nueva versión del módulo del mismo nombre disponible en Github, que incorpora algunas modificaciones que se explican al final de esta sección. Tanto en su versión original como en esta nueva versión, el módulo sx1276 disponibiliza múltiples funciones, utilizadas por el módulo radio. Estas funciones permiten inicializar la radio, resetearla por software, realizar las configuraciones para el funcionamiento de la radio en modo recepción y transmisión, y llenar una FIFO con los datos para trasmitir a través del enlace inalámbrico.

Los parámetros configurables más notables incluyen:

- comunicación: LoRa o FSK
- banda de frecuencias
- canal
- ancho de banda
- baudios
- potencia de transmisión
- habilitación de detección de errores con CRC
- habilitación de saltos de frecuencia
- modo de recepción en forma permanente

Para una trasmisión, se debe llenar adecuadamente la FIFO del módulo sx1276.

Modificaciones de esta versión del módulo sx1276

El código obtenido de Github no compilaba en el entorno de desarrollo utilizado para este proyecto. Tenía sentencias que no eran reconocidas por el IAR versión 6.40.2, por lo que fue necesario adaptarlas. En particular, en una ISR se cambió la línea:

void __attribute__ ((interrupt(PORT2_VECTOR))) port_2 (void)

por:

#pragma vector=PORT2_VECTOR

__interrupt void port_2(void)

Hecha la modificación, el programa de prueba compiló. Se programaron los dos nodos para probar el programa y no funcionó. El programa de prueba debe implementar un ping-pong. Primero se inicializan el MCU, el módulo LoRa y se configura la radio en modo transmisión. Luego se llena el buffer en el main, se llama a una función para transmitir el paquete a la que se le pasa el puntero al buffer y el largo del buffer. Esta función llama a su vez a la función de llenado del buffer del módulo sx1276. Anidada con esta, se llama a la función de llenado de la FIFO SPI del módulo spi. Luego que finaliza la comunicación SPI, se configura la radio en modo recepción y se vuelve al main (esto requiere unos cuantos pasos

intermedios que no vienen al caso). Se verificó con el IAR en modo debug que el microprocesador se reiniciaba al llamar a la función de llenado de la FIFO. También se verificó que si en lugar de llamar a la función para llenar la FIFO, simplemente se llenaba la FIFO (con las mismas líneas de código), el programa sí funcionaba. Se concluyó que el stack se llenaba al guardar los valores necesarios antes de hacer un salto a la función de llenado de la FIFO. Entonces se buscaron estrategias para hacer un mejor uso del stack. Se buscó identificar las prestaciones del programa de prueba que no fueran necesarias en el caso de aplicación. Se modificó la función de llenado del buffer en el main. Esta función se ubica en la versión final de la aplicación en el módulo radio. Las modificaciones consisten en simplificar esta función. La función original llena el array del main con la palabra clave 'ping'. Luego llama a una función de un módulo auxiliar, base64, que codifica la palabra 'ping', haciendo uso de dos funciones anidadas. En el caso de aplicación no es de interés la codificación del dato de temperatura, por lo que la aplicación en su versión final prescinde del módulo base64 y toda codificación. En la nueva versión, la función que llena el buffer únicamente lo llena y llama a una función del módulo sx1276 de transmisión. De ahí en adelante se ejecuta como se explicó para la versión original del programa de prueba del ping-pong.

Para el desarrollo de la aplicación final, se tuvo en cuenta estas consideraciones, evitando anidar funciones innecesariamente. Las funciones anidadas relacionadas con la comunicación inalámbrica, se recorren y se sale de ellas hasta volver al main y desde ahí se anidan otras funciones relativas a la medida de temperatura. De esta manera se aprovecha mejor el uso del stack.

Teniendo en cuenta que la versión original del programa de prueba del ping-pong fue desarrollado para compilar con gcc, parece razonable pensar que el programa sí debería compilar, pero con optimizaciones del compilador que en el IAR no se configuraron. Por los mismos motivos, seguramente las optimizaciones permiten que el anidamiento de funciones no llene el stack.

Módulo spi

Este módulo está disponible en Github y fue utilizado sin modificaciones en este proyecto. Implementa funciones para transmitir y recibir datos por SPI, que son utilizadas por los módulos sx1276 y radio. Estas son las funciones de transmisión y recepción en sí, la función que consulta si está listo el esclavo para recibir un dato nuevo o que la transmisión del esclavo fue finalizada y una función para habilitar el esclavo (chip enable).

Sensor de temperatura

El nodo encargado de medir la temperatura tiene un sensor de temperatura digital TMP75C. Este sensor tiene varios modos de funcionamiento. El utilizado en este proyecto es el modo continuo donde el sensor está continuamente tomando una medida de temperatura y la almacena en su registro de temperatura. Para comunicarse con el sensor se utiliza el protocolo de comunicación serial I^2C, el cual utiliza un hilo de datos y hilo para la señal de reloj. De acuerdo a la conexión del hardware presentada en la Fig. 3, los pines del launchpad conectados a estos dos hilos también están conectados al módulo LoRa. Por lo tanto se debe utilizar los periféricos en forma alternada. Esto implica que cada vez que se

vaya a tomar una medida de temperatura, se debe deshabilitar el módulo LoRa, configurar el sensor de temperatura, tomar la medida y al final habilitar el módulo LoRa y volverlo a configurar adecuadamente.

Las principales consideraciones a tener en cuenta en relación a la configuración del sensor de temperatura son las siguientes. Para comunicarse con un dispositivo cualquiera, se debe conocer la dirección del mismo, en este caso el sensor tiene la dirección 0x48 . El launchpad utilizado contiene un módulo (Universal serial communication interface USCI) que resuelve varias de las tareas necesarias para el funcionamiento de dicho protocolo, solo basta con configurar unos registros para establecer qué tipo de comunicación se requiere y la dirección del dispositivo con el que se quiera comunicar. Por ejemplo el registro UCB0CTL0 en su bit 7 se asigna si se va a usar direcciones de 10 o 7 bits y en los bits 1 y 2 se establece qué tipo de comunicación utilizará el módulo, que para este caso corresponde a I^2C. También se configura si la comunicación será asíncrona o sincrónica. Luego resta configurar el registro UCB0I2CSA el cual debe tener la dirección del dispositivo con el que se quiera comunicar. Una vez configurados los registros, ya es posible la comunicación con el sensor, ya sea para configurarlo o para tomar una medida de temperatura.

Módulo I^2C

Este módulo cuenta con la función de inicialización de I^2C, la cual configura los registros explicados anteriormente para poder comunicarse con el sensor.

Módulo temp

El módulo temp agrupa las funciones necesarias para obtener una medida de temperatura, en su interfaz pública está la función get_temp() que es la que retorna la medida. Es necesario cada vez que se requiera una medida de temperatura configurar los registros de comunicación I^2C ya que se utiliza el mismo módulo de comunicación serial del launchpad que utiliza la comunicación SPI, por lo tanto get_temp lo primero que hace es realizar la configuración. Luego el módulo involucra dos funciones, la primera es Transmit(), que es la responsable de iniciar la comunicación y enviar la condición de start que utiliza el módulo del launchpad para iniciar una comunicación I^2C y la siguiente función es Receive(), la cual configura el registro necesario para recibir un dato. Cuando un dato está disponible, se utiliza el registro de interrupción de la USCI para almacenar ese dato en la memoria correspondiente. Como el dato nuevo se almacena cuando la interrupción lo indica, se utiliza la sentencia __delay_cycles(1000) para asegurar que el sensor tenga el tiempo necesario de enviar una medida. Ésta sentencia es oportuna utilizarla en nuestro programa ya que solo se espera una medida de temperatura y nada más, en otras aplicaciones más complejas donde se involucren varios procesos no es conveniente utilizarla porque el micro se detiene durante ese periodo.

Módulo timer

Implementa la rutina de atención a la interrupción del timer de hardware. Esta rutina deshabilita las interrupciones, levanta la bandera timeout que utiliza la aplicación, reinicia el conteo de tiempo, habilita nuevamente las interrupciones y retorna al programa principal.

Módulo mcu

Inicializa los puertos deshabilitando aquellos que no se utilizan configurándolos como salidas. Deshabilita el watchdog timer y configura los leds utilizados para el debug.

Software del nodoBase

Se presenta un diagrama del software del nodoBase en la Fig. 5. El nodoBase utiliza muchos de los bloques del nodoTemp. Adicionalmente se tiene el módulo uart, que implementa la comunicación serial con el PC.

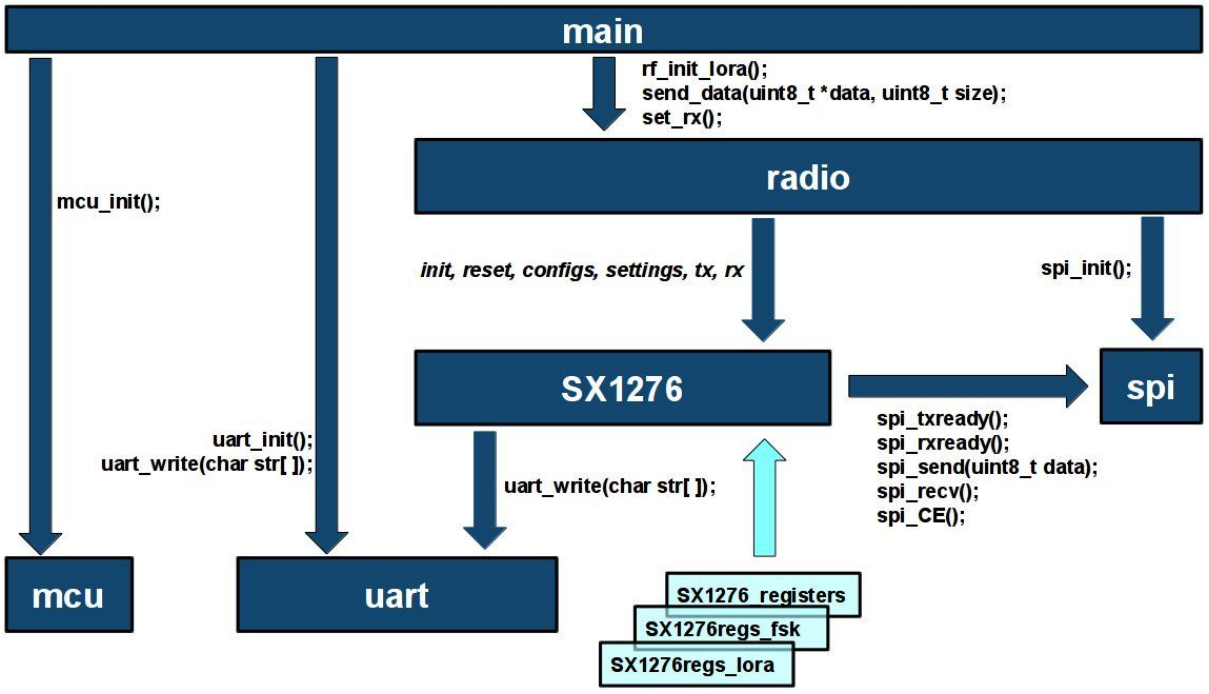


Figura 5: diagrama del software del nodoBase.

Módulo main

Es el encargado de inicializar todo el sistema, utilizando las funciones de inicialización de los módulos radio, uart y mcu. Ejecuta un loop infinito vacío. La dinámica de la aplicación viene dada por las interrupciones al recibir un nuevo paquete. De ser así, se trasmite un reconocimiento, se vuelve a modo de recepción y se imprime por UART el contenido del paquete recibido. Esto se hace utilizando las funciones send_data y set_rx del módulo radio y la función uart_write del módulo uart.

Módulo uart

Este módulo implementa la comunicación UART. Además de la inicialización del módulo, éste disponibiliza una función para escribir una cadena de caracteres. Los módulos sx1276 y el main llaman esta función para imprimir en la pantalla de un PC, utilizando el puerto USB del launchpad, el contenido de un paquete recibido, así como otros datos de interés para el caso de aplicación y para debug.

Pruebas

Se realizaron pruebas para verificar el funcionamiento del software y para caracterizar el sistema en términos del alcance, el consumo de corriente y throughput de datos.

Pruebas de la comunicación inalámbrica

Se realizaron pruebas de la red de dos nodos, para estresar la comunicación inalámbrica. Primero se resetean los nodos, éstos se comunican, es decir, el nodoTemp envía el dato de temperatura, el nodoBase recibe el dato y responde un reconocimiento, que es recibido por el nodoTemp. Se verificó que se intercambiaran paquetes en esta secuencia.

Luego se observó cómo el nodoBase recibía los paquetes, pero el nodoTemp dejaba de recibir los reconocimientos. De acuerdo a la implementación, esto es factible de ocurrir, y la aplicación del nodoTemp sólo envía paquetes cuando caduca el timer y toma un nuevo dato de temperatura. Finalmente se observó cómo el nodoBase dejó de recibir paquetes. Logs que muestran este comportamiento, están disponibles en el anexo. Se verificó que el comportamiento era el adecuado a las especificaciones.

Medida de alcance

Se hizo una prueba a la intemperie, con línea vista entre los dos nodos, a 3 m de altura, con una potencia de trasmisión de 14 dBm y la portadora en 868 MHz, para medir el alcance. Los nodos estaban alimentación a través del conector USB del launchpad a sendas computadoras portátiles. En este caso se obtuvo un alcance menor a 10 m, mientras el alcance estimado en entornos urbanos es de 5 km.

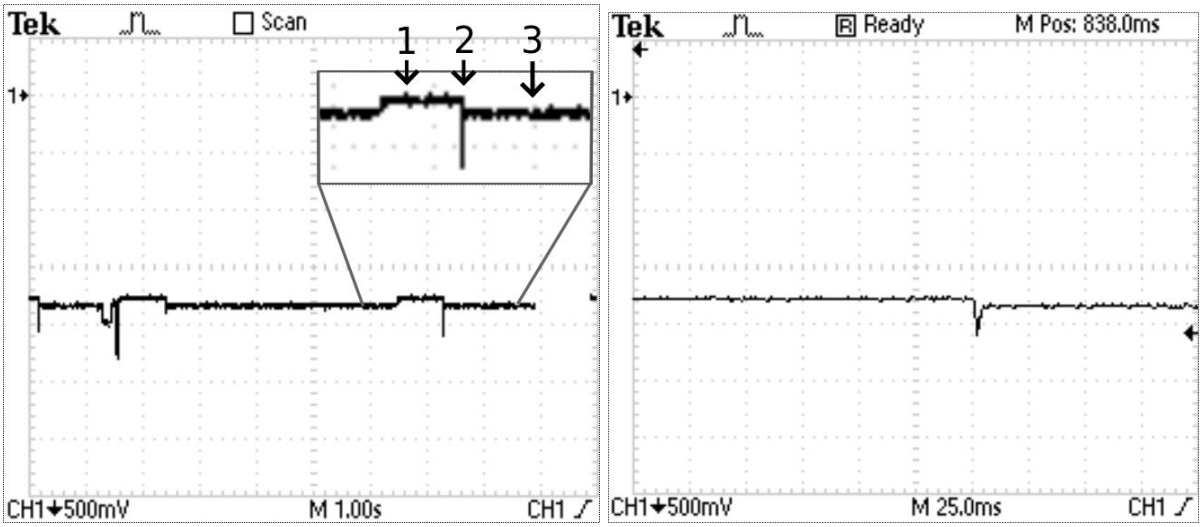
Luego se hizo una prueba en el laboratorio con línea vista entre los dos nodos y potencia de trasmisión de 14 dBm, pero con los launchpad alimentados vía USB a la red eléctrica con una fuente switcheada de 5V y 2A. Se consiguió un alcance menor a 5 m, consistente con la medida a la intemperie. Con este resultado se descartó la posibilidad de que la falta de alcance se debiera a la falta de corriente para un correcto funcionamiento del módulo LoRa.

Se concluyó que el mal funcionamiento debe estar relacionado con la configuración de la radio. No se encontró el origen del problema ni cómo resolverlo. Si bien la interconexión del hardware se hizo con cables y protoboard, no parece razonable pensar que esto justifique completamente el mal funcionamiento.

Caracterización del consumo y velocidad de transmisión de datos

Utilizando un amplificador logarítmico [10] se caracterizó el consumo de corriente del nodoTemp. La dinámica de la aplicación del nodoTemp es la siguiente. Luego de un reset, se toma una medida de temperatura, se transmite un paquete con el dato y se pasa a modo de recepción. Luego de del período establecido se vuelve a tomar una medida de temperatura, y repetir la secuencia. Estando en modo recepción, el nodoTemp puede recibir un reconocimiento del nodoBase.

El consumo medido incluye el consumo del launchpad, de la radio y del sensor. La Fig. 6 muestra capturas del osciloscopio a la salida del amplificador. El logaritmo en base diez del consumo de corriente corresponde a la tensión medida.



(a) (b)

Figura 6: capturas de osciloscopio de medidas de consumo de corriente utilizando un amplificador logarítmico.

La Fig. 6a muestra el consumo a lo largo de más de un período, en este caso configurado para ser de cinco segundos. El pico donde se tiene menor consumo y las variaciones anteriores, están asociados a un reset utilizando el botón del launchpad. Se muestra un detalle de las variaciones del consumo representativo del comportamiento, que se tiene periódicamente cada vez que se toma una medida de temperatura, se transmite un paquete y se espera en recepción. Se distinguen tres valores de consumo distintos asociados a tres estados, enumerados del 1 al 3.

En el estado 1 el nodo pide al sensor una medida de temperatura y espera la interrupción del sensor que llega cuando tiene el nuevo dato de temperatura. La aplicación tiene una espera ociosa de un segundo, mientras llega esta interrupción. Durante el estado 1, la radio está configurada en modo recepción. Por lo tanto el consumo del estado 1 es el de la radio en recepción, el sensor activo y el launchpad con el microprocesador corriendo.

Al finalizar la espera ociosa, se selecciona la radio (chip select) y se la configura para transmisión. Esta transición corresponde al estado 2, cuya duración del estado 2 es de 5 ms. El detalle del estado 2, ampliando la escala de los tiempos, se muestra en la Fig. 6b. El consumo durante el estado 2 corresponde al consumo del sensor de temperatura en espera (comunicación I^2C no tiene lugar), la radio fuera de los estados de transmisión o recepción, y el launchpad con el microprocesador ejecutando.

Cuando la radio comienza la transmisión, se pasa al estado 3. En este estado el nodo transmite un paquete con el dato de temperatura y configura la radio para recepción. La radio permanece en recepción, esperando un reconocimiento, hasta tanto se cumpla el tiempo para volver a medir temperatura, pasando al estado 1. Se observa que el estado de la radio en transmisión y recepción son indistinguibles.

El consumo en cada uno de los estados 1, 2 y 3, corresponde a 15.8 mA, 7.94 mA y 14.1 mA, respectivamente. De esto se deduce que el consumo del sensor de temperatura mientras se le consulta el dato y se sostiene la comunicación I2C es en media la diferencia entre el consumo de los estados 1 y 3, es decir 1.7 mA. El consumo durante el estado 2 no toma un valor estable, sino que va disminuyendo y luego se ve la transición al estado 3. Por lo tanto no se puede garantizar que el consumo mínimo medido durante el estado 2 (7.94

mA) sea el que se tendría con la radio fuera de transmisión y recepción, y el sensor en espera en régimen permanente, sino que es una cota superior. Por otro lado, a juzgar por el consumo del estado 3, no se tienen variaciones en el consumo cuando la radio está configurada en transmisión o recepción. Para poder diferenciar dos estados, se debería configurar la radio en transmisión durante un tiempo más largo (no sólo la transmisión de un único paquete de 1 byte).

En el caso de aplicación el throughput de datos es muy bajo, porque se transmite un único paquete de 1 byte cada un período preestablecido de unos segundos, especificado como diez segundos. La aplicación involucra una espera ociosa de un segundo que no afecta el desempeño del sistema para este caso particular, pero que acota el throughput de datos necesariamente a un paquete de 1 byte por segundo, o sea 8 baudios. No obstante, esta espera ociosa podría ser suprimida en una versión nueva del módulo temp, según se explica en el Anexo I. En este caso, el throughput de datos queda limitado al tiempo que demora la radio en transmitir un paquete de 1 byte, desde que se llama a la función para transmitir desde el main, hasta que la radio indica en su salida digital D0 que la transmisión finalizó; sin tener en cuenta la espera del reconocimiento. Cuando se llama a la función para transmitir desde el main, se selecciona la radio con el pin CS, se configura adecuadamente y se transmite el paquete. Desde la selección de la radio hasta el comienzo de la transmisión, transcurren 5 ms, de acuerdo a las medidas de consumo presentadas (estado 2). Del comienzo de la transmisión hasta que la radio indica en su salida D0 que terminó, transcurre algo menos de 1 ms, de acuerdo a la hoja de datos del módulo LoRa. Por lo tanto se tendrá en el peor de los casos un throughput de 1 byte por cada 6 ms, es decir 1.3 kbps, para el caso en que el paquete sea de 1 byte. Para otras aplicaciones puede tener interés aumentar el throughput de datos, en cuyo caso se deberá pensar en aumentar el payload del paquete e implementar piggybacking, si en el caso de aplicación esto tiene sentido.

Conclusiones

Se implementó la red de dos nodos que se comunican entre ellos utilizando el protocolo LoRa. Se encontraron y superaron dificultades para hacer funcionar el programa de prueba, debido a que tiene algunas sentencias que el IAR no reconoce, a que el stack se llena y resetea el microprocesador y a que no hay documentación disponible. Luego de tener este ejemplo en funcionamiento se implementó la aplicación propuesta en el proyecto que verifica las especificaciones. En particular se pudo implementar correctamente la comunicación con el sensor de temperatura por I^2C y con la radio vía SPI, compartiendo algunos pines del launchpad. Se caracterizó el sistema a través de medidas de desempeño en cuanto a alcance, consumo y throughput de datos. El alcance obtenido es menor a 10 m, lo que muestra que se debe depurar la configuración de la radio para poder obtener un alcance similar al que promete el fabricante. Los resultados de medida de consumo son adecuados a una aplicación de este tipo: 15.8 mA de consumo del launchpad, el sensor en espera y la radio en recepción/transmisión y 1.7 mA del sensor activo y la radio funcionando fuera de transmisión o recepción. Estos resultados de medida de consumo no dejan en evidencia, en primera instancia, cuál es el origen del problema del corto alcance. El throughput de datos en el caso de aplicación es de 8 baudios, que está bien en esta aplicación, pero que no puede aumentarse dada la implementación del software tal como está propuesta. En general se cumplió con lo planificado al inicio del proyecto.

Referencias

[1] Semtech, *Presentación del protocolo LoRa*, disponible en www.semtech.com/wireless-rf/internet-of-things/what-is-lora/

[2]Libelium Comunicaciones Distribuidas, *Waspmote-LoRa-868MHz_915MHz-SX1272, Networking Guide, v7.0*, febrero 2017.

[3] Texas Instruments, *MSP-EXP430G2 LaunchPadTM Development Kit*, Manuales disponibles en www.ti.com.

[4] Texas instruments, *MSP430x2xx Family User´s Guide*, revisión julio de 2013.

[5] Semtech Corporation, *SX1276/77/78/79 Wireless, Sensing and Timing DATASHEET, SX1276/77/78/79 - 137 MHz to 1020 MHz Low Power Long Range Transceiver. Rev. 4,* disponible en http://modtronix.com/inair9.html, marzo 2015.

[6] Texas instruments, *TMP75C Digital Temperature Sensor*, revisión 2014.

[7] *Msp430 I²C single write-read example,* disponible en electronics.stackexchange.com/questions/67524/msp430-i%C2%B2c-single-write-read-example, abril de 2013.

[8] *msp430-radio, SX1276 transceiver driver implementation for MSP430G2553*, disponible en github.com/Show-vars/msp430-radio.

[9] *LoRaWAN endpoint stack implementation and example projects*, disponible en github.com/Lora-net/LoRaMac-node, noviembre 2013.

[10] Ee-quipment, *201 Real-Time Current Monitor Datasheet, Rev. 2.0*, disponible en www.ee-quipment.com/pages/manuals-and-documents, 2013.

Anexo

Anexo I: Conceptos del curso aplicados al proyecto

Modularización y abstracción de hardware

Tanto la aplicación del nodoTemp como del nodoBase, fueron desarrolladas utilizando un conjunto de módulos. Cada módulo implementa una serie de funcionalidades y deja disponibles funciones en su interfaz pública para que otro módulos las utilicen. Esto permite tener un orden jerárquico de los módulos desde los que dependen más del hardware específico a los que son independientes.

El módulo sx1276 es un ejemplo claro de la dependencia con el hardware. Este incluye las direcciones de los registros de la radio, así como parámetros relativos a la comunicación (portadora, canales, potencias), que varían de una radio a otra. Además, este módulo LoRa utiliza comunicación SPI, lo que puede variar de un módulo a otro. Es por esto que el módulo sx1276 se sirve de las funciones SPI y no es necesario que la aplicación a nivel del main sepa qué protocolo de comunicación utiliza la radio. Para poder independizar el main del hardware de la radio, se implementa el módulo radio, que resuelve en forma transparente la transmisión y recepción de paquetes. Esto permite que si se utiliza otra radio, con otro protocolo de comunicación inalámbrica o con otro protocolo de comunicación con el procesador distinto de SPI, sea factible implementar una versión distinta del módulo radio y conservar el main sin modificaciones.

Pines compartidos para comunicación I^2C con sensor y SPI con la radio

Los pines 6 y 7 del puerto 1 (P1.6 y P1.7) se utilizan para la comunicación SPI con el módulo LoRa e I^2C con el sensor de temperatura. Se tuvo cuidado en la aplicación con la secuencia de ejecución alternada de comunicación por parte del microprocesador y también de esperas por eventos del sensor y la radio. Cuando se solicita al sensor una medida y se espera una interrupción, se debe tener la radio deshabilitada. Similarmente, no se debe comenzar una comunicación con el sensor cuando se espera una interrupción de la radio. Esto es porque los pines compartidos son de reloj y datos y la comunicación con el sensor podría ser interpretada por la radio, y viceversa, generando errores esporádicamente difíciles de rastrear.

Espera ociosa para la medida de temperatura

La función que llama el main para obtener un nuevo dato de temperatura, es provista por el módulo temp. Esta función realiza los siguientes pasos:

i) configuración de los registros para I^2C,

ii) pide dato de temperatura: vía I^2C con el sensor, que es el esclavo en la dirección 0x48,

iii) espera interrupción I^2C implementada con un retardo Td,

iv) devuelve el dato de temperatura.

Esta implementación funciona bien en el caso de aplicación, pero es poco portable a otras aplicaciones, debido a la espera ociosa introducida por el retardo Td en el paso (iii). Cuando se pide un dato de temperatura, el sensor toma la medida y cuando está lista para ser leída genera una interrupción. Mientras, la aplicación no ejecuta código, sino que espera durante el tiempo Td. En esa situación se tiene el sistema consumiendo sin hacer nada. Además, el tiempo Td que se configura debe ser suficientemente largo para dar lugar con certeza a la interrupción, haciendo que en general se sobreestime. Esto hace que se limiten las aplicaciones a aquellas en que las restricciones de tiempos permitan estas esperas. Para mantener la implementación para aplicaciones con mayores restricciones de tiempos, se puede reducir el Td, pero es una mala solución de ingeniería. También tiene otro defecto, que es que si la interrupción no se produce por alguna falla, el programa devuelve basura o un dato anterior de temperatura (en el mejor de los casos) en lugar de una medida de temperatura.

Una implementación más robusta implica implementar una función de solicitud de temperatura, que ejecute los pasos (i) y (ii) y no devuelva nada. Luego cuando ocurre la interrupción I^2C, se debe levantar una bandera que permita al main reconocer que hay un nuevo dato de temperatura para leer. Por otro lado, debe haber un timer que caduque si transcurre un cierto tiempo (por ejemplo Td), para volver a pedir un dato de temperatura o levantar una bandera de que el último dato de temperatura no es válido. Esta implementación alternativa no enlentece la ejecución innecesariamente y no genera datos basura en caso de que la lectura de temperatura falle. Ambas implementaciones implican mantener el módulo LoRa deshabilitado hasta que se dé por terminado el sensado de temperatura (sea por timeout, sea porque se obtuvo un dato), porque se tienen compartidos los pines de comunicación del sensor con los del módulo LoRa. Es decir que de cualquier manera, no se va a poder estar recibiendo ni transmitiendo datos a través del enlace inalámbrico mientras se sensa la temperatura.

Llenado del stack

El programa de prueba de la comunicación inalámbrica llenaba el stack, reiniciando el procesador. Entonces se buscaron estrategias para hacer un mejor uso del stack. Se buscó identificar las prestaciones del programa de prueba que no fueran necesarias en el caso de aplicación. Para el desarrollo de la aplicación final, se tuvo en cuenta estas consideraciones, evitando anidar funciones innecesariamente. Las funciones anidadas relacionadas con la comunicación inalámbrica, se recorren y se sale de ellas hasta volver al main y desde ahí se anidan otras funciones relativas a la medida de temperatura.

Teniendo en cuenta que la versión original del programa de prueba del ping-pong fue desarrollado para compilar con gcc, parece razonable pensar que el programa sí debería correr sin llenar el stack, pero con optimizaciones del compilador que en el IAR no se

configuraron.

Anexo II: Especificación del proyecto

LoRa – comunicación inalámbrica de largo alcance y bajo consumo.

Integrantes: Juan Martín Chiale y Mariana Siniscalchi
Tutor: Javier Schandy

1. Descripción del problema a ser resuelto

Se busca entender el funcionamiento del protocolo LoRa [1]. Se implementará una red de dos nodos utilizando una radio y un microcontrolador adecuado para la aplicación. Se desarrollará una aplicación que tome medidas de un sensor y las mande a otro nodo vía RF con el protocolo mencionado. Para eso se desarrollarán los drivers para la comunicación con la radio, y se configurarán los registros de la misma para que funcione adecuadamente. Para tomar las medidas se utilizará un sensor de temperatura digital con comunicación I2C. Se evaluará qué throughput de datos y distancias de transmisión se pueden lograr, y el consumo energético en cada caso.

2. Antecedentes

Está disponible una aplicación de prueba de la comunicación LoRa para la radio SX1276 [2] en Github [3] basado en las bibliotecas disponibles [4].

3. Objetivos del proyecto:

Se estudiarán en profundidad todos los modos de funcionamiento de la radio y se elegirá la configuración óptima de los registros para la aplicación planteada, logrando comunicar dos nodos utilizando el protocolo LoRa. Se desarrollará una aplicación que se comunique con un sensor de temperatura, tome medidas y las envíe periódicamente al nodo vecino. El nodo vecino recibirá los paquetes y enviará por la UART el valor de temperatura recibido, para verificar el funcionamiento.

4. Alcance del proyecto:

Se armará una red de dos nodos, utilizando el kit de desarrollo MSP-EXP430G2, basado en el microcontrolador de Texas Instruments MSP430G2553 [5]. Se utilizará el módulo LoRa inalámbrico SX1276 [2]. Se deberá resolver la comunicación SPI con el módulo LoRa para configurarlo correctamente y enviar y recibir paquetes. Se deberá resolver la comunicación I2C con el sensor de temperatura. Se deberá dar una medida del throughput y el consumo en ese caso.

5. Descripción del sistema

El sistema estará formado por dos nodos que se comunicarán utilizando LoRa.

Uno de ellos estará dotado de un sensor de temperatura, además del transceptor de radio. A ese nodo le llamaremos NodoTemp. El NodoTemp tomará medidas de temperatura periódicamente y las enviará al otro nodo, al que llamaremos NodoUART y estará conectado a un PC via UART. El NodoUART recibirá el dato de temperatura y lo enviará por la UART al PC.

La Figura 1 muestra un diagrama de bloques del sistema.

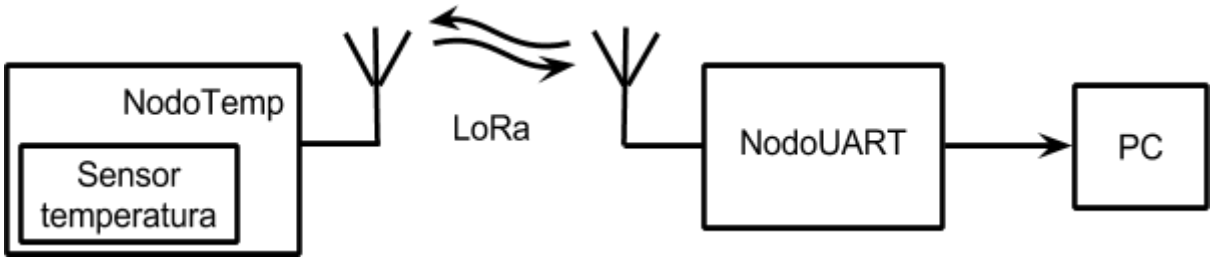


Figura 1: diagrama de bloques de una red formada por dos nodos, para sensar temperatura remotamente.

6. Requerimientos y restricciones del sistema

- Módulo SX1276: Requiere una tensión de alimentación de 3.3V y una interfaz de comunicación SPI.
- Procesamiento y memoria: el nodo NodoUART necesita el módulo de comunicación por uart y un timer que ocupan aproximadamente 3,5 Kb de memoria más el módulo de comunicación por radio que necesita 7,5 Kb de memoria, requiriendo un total de 11 kb de memoria para su funcionamiento. Mientras que el NodoTemp necesita un módulo de comunicación SPI que se estima de igual tamaño que el uart y el módulo de comunicación por radio, requiriendo aproximadamente la misma cantidad de memoria que el NodoUART.
- Tiempos de respuesta: la aplicación muestreará cada 10s una medida de temperatura, en este tiempo el sistema deberá leer la temperatura en el NodoTemp y enviarla al NodoUART para que ésta la transmita por su uart y envíe la confirmación hacia el NodoTemp.

7. Diseño preliminar

Se desea implementar una red inalámbrica compuesta por dos nodos, a los que en adelante llamaremos NodoTemp y NodoUART. Ambos nodos estarán implementados sobre el kit de desarrollo MSP-EXP430G2 launchpad, basado en el microcontrolador de Texas Instruments MSP430G2553 [5].

La comunicación inalámbrica se realizará utilizando el protocolo LoRa. Para esto, el kit de desarrollo estará conectado al módulo LoRa SX1276. Este módulo funciona opcionalmente entre la banda de 868 MHz y 915 MHz, y se consiguen 15 km de alcance. Se utilizará en la banda de 868 MHz en principio.

El NodoTemp tomará medidas de temperatura periódicamente, del sensor de temperatura TMP75C disponible para la aplicación [6]. Luego enviará el dato de temperatura al NodoUART. El NodoUART recibirá el dato y lo comunicará a un PC a través de la UART.

El sistema estará implementado según se muestra en la Figura 2. El NodoTemp comprende el
El launchpad se comunicará vía SPI con el módulo LoRa y vía I2C con el sensor de temperatura. Por otra parte el NodoUART comprende el launchpad, el módulo LoRa y el manejo de la UART para la comunicación con launchpad, el módulo LoRa y el sensor de temperatura.
el PC.

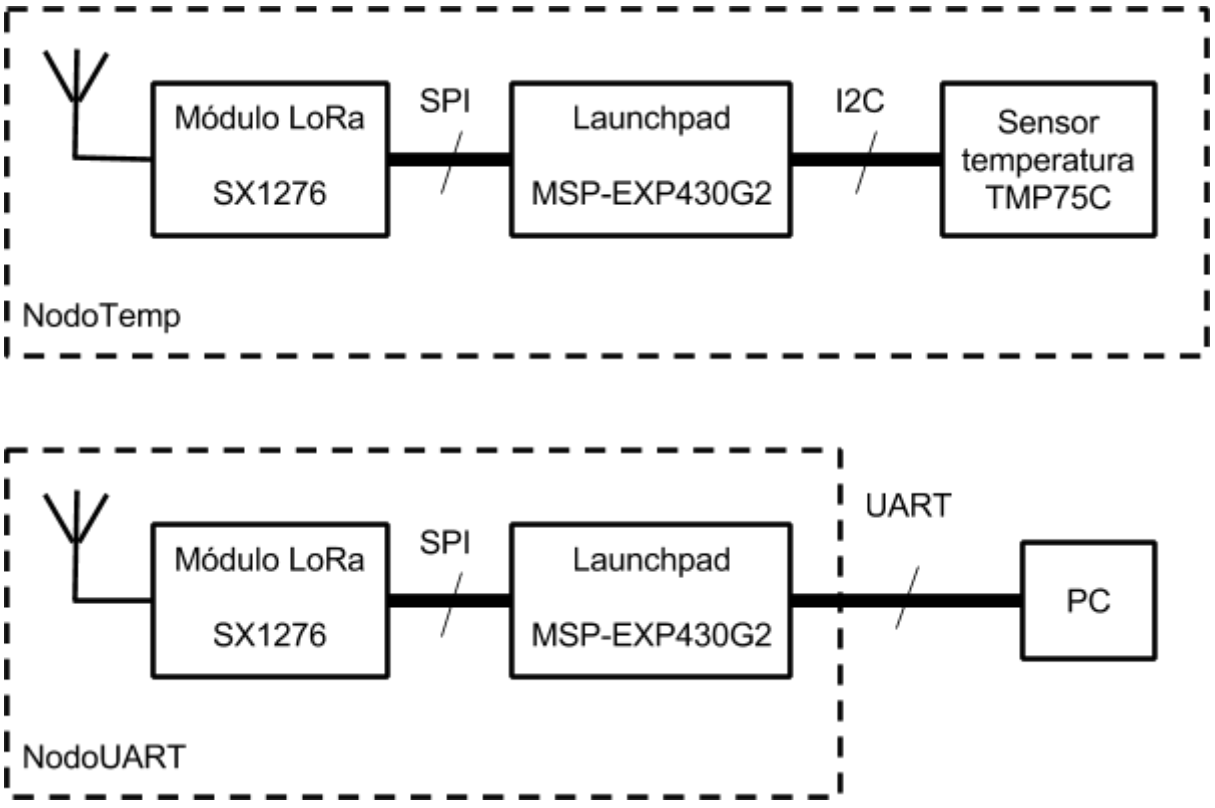


Figura 2: NodoTemp y NodoUART formando una red de dos nodos para sensar temperatura remotamente en el NodoTemp y enviar los datos utilizando LoRa al NodoUART conectado a un PC para desplegar los datos recibidos.

Para poder aprovechar la implementación de los módulos timer y uart del laboratorio, y dado que se tienen disponibles las bibliotecas del stack de protocolo LoRa, se utilizará una arquitectura Round Robin con interrupciones.

NodoTemp

El NodoTemp deberá tener configurado un período de muestreo de la temperatura. Se implementará un módulo timer que cuente ese tiempo y, cada vez que se haya cumplido, tome una medida de la temperatura y avise al programa principal que hay una medida nueva. El programa principal deberá leer el dato de temperatura e iniciar el envío del dato por la red. Para esto se tendrá un módulo radio, que disponibiliza para el programa principal

las utilidades del stack del módulo LoRa necesarias para enviar y recibir. El programa principal deberá reenviar el dato cada cierto período de tiempo hasta recibir la confirmación del NodoUART. Para esto, el módulo radio debe avisar al programa principal que hay un nuevo dato recibido por el módulo LoRa, listo para ser leído. El programa principal verifica que esto sea una confirmación para cesar los reintentos.

NodoUART

El NodoUART también utilizará el módulo radio. Ante un aviso del módulo radio de un nuevo dato recibido, el programa principal verifica que esto sea un valor de temperatura válido y lo saca por la UART, utilizando el módulo uart. El programa principal envía una confirmación de recepción al NodoTemp, utilizando el módulo radio. En la Figura 3, se muestra un diagrama de bloques de los programas de sendos nodos.

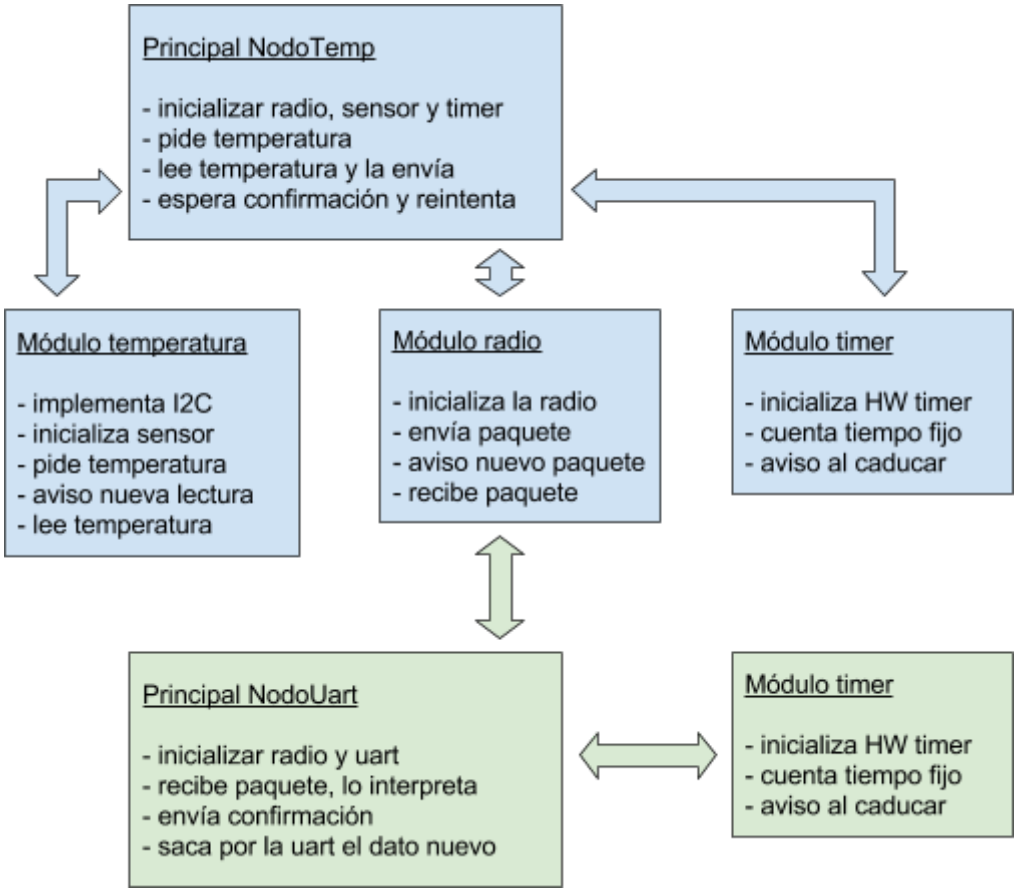


Figura 3: Módulos que conforman los programas de NodoTemp y NodoUART.

5. Planificación

A continuación, se describen las tareas del proyecto. Algunas tareas tienen asociadas las pruebas a realizar que se detallan al final. El hito intermedio y las presentaciones de avance se incluyen en el listado.

Tarea 1: preparar presentación inicial
Tarea 2: estudiar en profundidad la hoja de datos del módulo de radio seleccionado (SX1276), analizando los distintos modos de funcionamiento y parámetros a configurar.
Tarea 3: comunicación SPI con la radio y configuración de los registros. Aquí se hará la prueba básica del módulo LoRa.
Tarea 4: implementar la red de dos nodos y hacer la prueba completa del módulo LoRa.
Hito 1. Se planifica para tener las tareas del 1 al 4 completas para la fecha del hito 1.
Tarea 5: implementar comunicación I2C con el sensor y realizar pruebas.
Tarea 6: desarrollar la aplicación que correrá en el NodoTemp. La aplicación toma una medida de temperatura periódicamente y la envía al nodo vecino utilizando LoRa. Probar la aplicación.
Tarea 7: desarrollar el programa que corre en el NodoUART que debe recibir los paquetes utilizando LoRa y sacar por la UART el contenido del paquete (que es la medida de temperatura del otro nodo). Probar la aplicación. Se puede agregar eventualmente la prestación de sacar también por la UART una identificación del nodo que envió el paquete.
Tarea 8: desarrollar un programa medida_performance para caracterizar el throughput de datos y el consumo en ese caso. Este programa deberá enviar paquetes de tamaño fijo a una tasa fija configurable, utilizando los módulos radio y timer. Hacer pruebas de medida_performance.
Tarea 9: preparar defensa (presentación y demo) y documentación final.

<u>Pruebas a realizar:</u>

Para verificar el correcto funcionamiento del proyecto se realizarán distintas pruebas a cada módulo.

- Sensor de Temperatura y comunicación I2C.

Comprobar que la configuración es correcta obteniendo una medida de temperatura. Luego para verificar si es coherente la medida, se compara con una medida de temperatura tomada por el sensor del microprocesador.

- Módulo Lora.

i) Prueba básica: implementar un programa sencillo que lea los registros de la radio para verificar que se está configurando como se quiere.

ii) Prueba completa: enviar un Ping desde el NodoTemp y esperar una respuesta Pong del NodoUart. Aquí se prueba el correcto funcionamiento de la comunicación SPI entre el launchpad y el módulo LoRa y la comunicación inalámbrica.

- Módulo uart.

Se conecta el launchpad a un PC y se envía por la uart una palabra predefinida para ser leída con el RealTerm, si se visualiza la palabra enviada, el módulo funciona correctamente.

Luego se realizarán pruebas sobre el programa completo del NodoTemp y del NodoUART por separado, verificando que la dinámica del round robin es la esperada, de acuerdo al diseño preliminar. Por último se probará el sistema completo, verificando que el intercambio de paquetes se hace en la secuencia preestablecida, de acuerdo al diseño preliminar. En particular se verificará que el NodoTemp reenvía los paquetes hasta tener confirmación de que fueron recibidos.

Finalmente se caracterizará el throughput y se medirá el consumo utilizando el amplificador logarítmico. Para esto se cargará el programa medida_performance.

Cronograma:

Semana 1: tareas 1 y 2.
Semana 2: tareas 3 y 4.
Semana 3: tarea 5. Hito 1. Tarea 6.
Semana 4: tareas 7 y 8.
Semana 5: tarea 9.

Referencias:

[1] Semtech, *Presentación del protocolo LoRa*, disponible en www.semtech.com/wireless-rf/internet-of-things/what-is-lora/
[2] Semtech, *Wireless SX1276 LoRa Module, 868MHz and 915MHz, 3.3V, SMA Connector*. Descripción y especificaciones disponibles en http://modtronix.com/inair9.html.
[3] SX1276 transceiver test code for MSP430G2553, available in https://github.com/Show-vars/msp430-radio.
[4] Reference implementation and documentation of a LoRa network node, available in https://github.com/Lora-net/LoRaMac-node.
[5] Texas Instruments, MSP-EXP430G2 LaunchPadTM Development Kit. Manuales disponibles en www.ti.com.
[6] Texas Instruments, *TMP75C 1.4V-Capable Temperature Sensor with I2C/SMBus Interface with One-Shot Mode in LM75 Pinout*, available in: www.ti.com/product/TMP75C.

Anexo III: Planificación del proyecto

El proyecto se desarrolló esencialmente de acuerdo a la planificación explicada en la especificación del Anexo II. Las horas dedicadas fueron acordes a la planificación.

El desarrollo del programa de prueba para las medidas de desempeño y la realización de las pruebas para caracterizar el desempeño, estaban previstas para una semana. Se realizaron las pruebas en ese plazo, pero los resultados obtenidos plantearon la necesidad de revisión del código de la aplicación y nuevas pruebas de las medidas de alcance. Estas pruebas adicionales y las modificaciones necesarias para su éxito, no se realizaron en este proyecto. Una semana adicional para esas tareas podría haber sido suficiente, para lo que hubiera hecho falta modificar el alcance; por ejemplo implementando la comunicación inalámbrica únicamente, sin el sensor de temperatura. Esto hubiera simplificado mucho la aplicación, como contrapartida.