Radio

> Página extraída de Federico Coca Guia de Trabajo de Microbit CC-BY-SA

El módulo de `radio` permite que los dispositivos trabajen juntos a través de redes inalámbricas sencillas.

El módulo de radio es conceptualmente muy sencillo:

- Los mensajes broadcast o de difusión tienen una longitud configurable (hasta 251 bytes).
- Los mensajes recibidos se leen de una cola de tamaño configurable (cuanto mayor sea la cola, más memoria RAM se utilizará). Si la cola está llena, se ignoran los mensajes nuevos. La lectura de un mensaje lo elimina de la cola.
- Los mensajes se emiten y reciben en un canal preseleccionado (numerado de 0 a 83).
- Las emisiones tienen un determinado nivel de potencia: más potencia significa más alcance.
- Los mensajes se filtran por dirección (como un número de casa) y grupo (como un destinatario con nombre en la dirección especificada)
- La velocidad de transmisión puede ser una de las tres predeterminadas.
- Se envían y reciben bytes para trabajar con datos arbitrarios.
- Utilizando `receive_full` se obtiene todos los detalles sobre un mensaje entrante: los datos como tales, la intensidad de la señal de recepción y una marca de tiempo en microsegundos cuando llegó el mensaje.
- Es fácil enviar y recibir mensajes como cadenas.

Para acceder a este módulo se necesita:

- `import radio`

Las constantes son:

- `radio.RATE_1MBIT`. Es una constante utilizada para indicar un caudal de 1 Mbit por segundo.
- `radio.RATE_2MBIT`. Es una constante utilizada para indicar un caudal de 2 Mbit por segundo.

Las funciones disponibles son:

- `radio.on()`. Enciende el módulo de radio. Desde MicroPython-on-micro:bit v1.1 la radio se activa por defecto cuando se importa el módulo de radio. En versiones anteriores, para reducir el consumo de energía, esta función tenía que ser llamada explícitamente. Para esos casos `radio.off()` puede ser llamada después de la importación.
- `radio.off()`. Apaga la radio. Esto ahorra energia y memoria.
- `radio.config(**kwargs)`. Configura varios ajustes basados en palabras clave relacionados con la radio. A continuación se enumeran los ajustes disponibles y sus valores predeterminados.
- `length`. La longitud (por defecto=32) define la longitud máxima, en bytes, de un mensaje enviado por radio. Puede tener una longitud máxima de 251 bytes (254 - 3 bytes para los preámbulos S0, LENGTH y S1).
- `queue`. La cola (por defecto=3) especifica el número de mensajes que se pueden almacenar en la cola de mensajes entrantes. Si no hay espacio en la cola para mensajes entrantes, el mensaje entrante se descarta.
- `channel`. El canal (por defecto=7) puede ser un valor entero de 0 a 83 (inclusive) que define un "canal" arbitrario al que se sintoniza la radio. Los mensajes se enviarán a través de este canal y sólo los mensajes recibidos a través de este canal se pondrán en la cola de mensajes entrantes. Cada paso tiene un ancho de 1MHz, basado en 2400MHz.
- `power`. La potencia (por defecto=6) es un valor entero de 0 a 7 (ambos inclusive) que indica la intensidad de la señal utilizada al emitir un mensaje. Cuanto mayor sea el valor, más potente será la señal, pero más potencia consumirá el dispositivo. La numeración se traduce en posiciones en la siguiente lista de valores dBm (decibelios milivatio): -30, -20, -16, -12, -8, -4, 0, 4.
- `address`. La dirección (por defecto=0x75626974) es un nombre arbitrario, expresado como una dirección de 32 bits, que se utiliza para filtrar los paquetes entrantes a nivel de hardware, manteniendo sólo aquellos que coinciden con la dirección que establezca. El valor por defecto utilizado por otras plataformas relacionadas con micro:bit es el valor por defecto utilizado aquí.
- `group`. El grupo (por defecto=0) es un valor de 8 bits (0-255) que se utiliza con la dirección al filtrar los mensajes. Conceptualmente, "dirección" es como una dirección de casa/oficina y "grupo" es como la persona de esa dirección a la que se quiere enviar el mensaje.
- `data_rate`. La tasa_de_datos (por defecto=radio.RATE_1MBIT) indica la velocidad a la que se produce el flujo de datos. Puede ser uno de los siguientes contantes definidos en el módulo de radio : `RATE_1MBIT` o `RATE_2MBIT`.

Nota Una velocidad de datos menor de 250 kbit/seg es compatible con micro:bit V1, y puede ser posible con micro:bit V2, pero no se garantiza que funcione en todos los dispositivos. Para acceder a esta característica oculta para la compatibilidad con V1

> ponemos 2 en el argumento `data_rate`.

Si **no se llama** a `config` **se asumen** los valores por defecto descritos anteriormente.

- `radio.reset()`. Restablece los valores por defecto (como se indica en la documentación de la función de configuración). Ninguno de los siguientes métodos de envío o recepción funcionará hasta que la radio esté encendida.
- `radio.send_bytes(message)`. Envía `message` conteniendo bytes.
- `radio.receive_bytes()`. Recibe el siguiente mensaje entrante en la cola de mensajes. Devuelve `None` (Ninguno) si no hay mensajes pendientes. Los mensajes se devuelven como bytes.
- `radio.receive_bytes_into(buffer)`. Recibe el siguiente mensaje entrante en la cola de mensajes. Copia el mensaje en el búfer, recortando el final del mensaje si es necesario. Devuelve `None` si no hay mensajes pendientes; en caso contrario, devuelve la longitud del mensaje (que puede ser superior a la longitud del búfer).
- `radio.send(message)`. Envía una cadena de mensajes. Esto es el equivalente de `send_bytes(bytes(message, 'utf8'))` pero con `b'\x01\x00\x01'` antepuesto (para hacerlo compatible con otras plataformas que apuntan al micro:bit).
- `radio.receive()`. Funciona exactamente igual que `receive_bytes` pero devuelve lo que se haya enviado. Es es equivalente a `str(receive_bytes(), 'utf8')` pero con una comprobación de que los tres primeros bytes son `b'\x01\x00\x01'` (para hacerlo compatible con otras plataformas que puedan tener como objetivo el micro:bit). Elimina los bytes añadidos antes de convertir a cadena Se lanza una excepción `ValueError` si falla la conversión a cadena.
- `radio.receive_full()`. Devuelve una tupla que contiene tres valores que representan el siguiente mensaje entrante en la cola de mensajes. Si no hay mensajes pendientes se devuelve `None`.

Los tres valores de la tupla representan:

- el siguiente mensaje entrante en la cola de mensajes en bytes.
- el RSSI (intensidad de la señal): un valor entre 0 (más fuerte) y -255 (más débil) medido en dBm.
- una marca de tiempo en microsegundos: el valor devuelto por `time.ticks_us()` cuando se recibió el mensaje.

Envío de la temperatura

Vamos a enviar la temperatura medida por la placa y en el valor recibido vamos a calcular la diferencia entre la temperatura de la placa que recibe y la que envía, estableciendo así la

diferencia de temperaturas entre, por ejemplo, una zona al sol y otra a la sombra.

El código es el siguiente

El código del programa es el siguiente:

```
from microbit import *
import radio
radio.on()
radio.config(channel=50, group=90)

while True:
    if button_a.is_pressed():
       radio.send(str(temperature()))
    recibido = radio.receive()
    if recibido is not None:
        display.show(recibido)
        sleep(50)
    display.clear()
```

El resultado (se visualiza en Makecode que permite una simulación en dos placas que https://python.microbit.org/ no permite)

A11_brujula_MC.png

Autor Federico Coca Fuente : Guía de Trabajo de Microbit Licencia CC-BY-SA

Enviar "Agitar" cuando se agita

vamos a realizar una especie de juego en el que se envía el mensaje "agitar (shake)" a un grupo de micro:bits (podemos poner otras en un grupo diferente) cuando se realiza justamente ese gesto. Usaremos uno de los botones para salir del programa.

El código del programa es el siguiente:

```
from microbit import *
import radio
radio.on()
radio.config(channel=50, group=90)
```

```
while True:
    if accelerometer.is_gesture('shake'):
        radio.send("Agitar")
    recibido = radio.receive()
    if recibido is not None:
        display.show(recibido)
        sleep(50)
    display.clear()
```

El resultado (se visualiza en Makecode que permite una simulación en dos placas que https://python.microbit.org/ no permite)

A11_brujula_MC.png

Autor Federico Coca Fuente : Guía de Trabajo de Microbit Licencia CC-BY-SA

> Página extraída de Federico Coca Guia de Trabajo de Microbit CC-BY-SA

Revision #5
Created 2024-09-27 12:29:19 CEST by Javier Quintana
Updated 2024-10-01 19:42:14 CEST by Javier Quintana