clase UART – bus de comunicación serie dúplex

UART implementa el protocolo estándar de comunicaciones serie dúplex UART/USART. A nivel físico, consta de dos líneas: RX y TX. La unidad de comunicación es un carácter (no confundir con una cadena de caracteres) que puede tener 8 o 9 bits de ancho.

Los objetos UART se pueden crear e inicializar utilizando:

```
from pyb import UART

uart = UART(3, 9600, timeout_char=1000)                     # init with given baudrate
uart.init(9600, bits=8, parity=None, stop=1, timeout_char=1000) # init with given parameters
```

Los bits pueden ser 7, 8 o 9. La paridad puede ser Ninguna, 0 (par) o 1 (impar). El valor de parada puede ser 1 o 2.

Nota: Con paridad = Ninguna, solo se admiten 8 y 9 bits. Con paridad habilitada, solo se admiten 7 y 8 bits.

Un objeto UART actúa como un objeto de flujo y la lectura y escritura se realizan utilizando los métodos de flujo estándar:

```
uart.read(10)       # read 10 characters, returns a bytes object
uart.read()         # read all available characters
uart.readline()     # read a line
uart.readinto(buf)  # read and store into the given buffer
uart.write('abc')   # write the 3 characters
```

Se pueden leer/escribir caracteres individuales mediante:

```
uart.readchar()     # read 1 character and returns it as an integer
uart.writechar(42)  # write 1 character
```

Para comprobar si hay algo para leer, utilice:

```
uart.any()          # returns the number of characters waiting
```

Nota: Las funciones de flujo `read` , `write` , etc. son nuevas en MicroPython v1.3.4. Las versiones anteriores usan `uart.send` y `uart.recv` .

Constructores

clase **pyb.UART** (*bus , ...)*

Construya un objeto UART en el bus indicado. `bus` Puede ser 1/3. Sin parámetros adicionales, el objeto UART se crea, pero no se inicializa (conserva la configuración de la última inicialización del bus, si la hubiera). Si se proporcionan argumentos adicionales, el bus se inicializa. Consulte `init` los parámetros de inicialización.

Los pines físicos del bus UART son para la cámara OpenMV:

- `UART(3)` : `(TX, RX) = (P4, P5) = (PB10, PB11)`

Los pines físicos de los buses UART son para OpenMV Cam M7 y H7:

- `UART(1)` : `(TX, RX) = (P1, P0) = (PB14, PB15)`
- `UART(3)` : `(TX, RX) = (P4, P5) = (PB10, PB11)`

Métodos

UART.init (**velocidad** *en baudios , bits=8 , paridad=Ninguna , parada=1 , * , tiempo de espera=1000 , flujo=0 , carácter_de_tiempo_de_espera=0 , longitud_de_buffer_de_lectura=64)*

Inicialice el bus UART con los parámetros dados:

- `baudrate` es la frecuencia del reloj.
- `bits` es el número de bits por carácter, 7, 8 o 9.
- `parity` es la paridad, `None` , 0 (par) o 1 (impar).
- `stop` es el número de bits de parada, 1 o 2.
- `flow` Establece el tipo de control de flujo. Puede ser 0 `UART.RTS` , `UART.CTS` o . `UART.RTS | UART.CTS`
- `timeout` es el tiempo de espera en milisegundos para esperar a escribir/leer el primer carácter.
- `timeout_char` es el tiempo de espera en milisegundos entre caracteres mientras se escribe o lee.
- `read_buf_len` es la longitud del carácter del búfer de lectura (0 para deshabilitar).

Este método generará una excepción si la tasa de baudios no se puede establecer dentro del 5 % del valor deseado.

Nota: Con paridad = Ninguna, solo se admiten 8 y 9 bits. Con paridad habilitada, solo se admiten 7 y 8 bits.

UART. deinit ( )

Apague el bus UART.

UART. cualquier ( )

Devuelve el número de bytes en espera (puede ser 0).

UART. leer ([*nbytes*])

Leer caracteres. Si `nbytes` se especifica, se lee como máximo esa cantidad de bytes. Si `nbytes` hay disponibles en el búfer, retorna inmediatamente; de lo contrario, retorna cuando llegan suficientes caracteres o se agota el tiempo de espera.

Si `nbytes` no se proporciona, el método lee la mayor cantidad de datos posible. Regresa una vez transcurrido el tiempo de espera.

Nota: para caracteres de 9 bits, cada carácter ocupa dos bytes, `nbytes` debe ser par y el número de caracteres es `nbytes/2`.

Valor de retorno: un objeto de bytes que contiene los bytes leídos. Retorna `None` al expirar el tiempo de espera.

UART.readchar ( )

Recibe un solo personaje en el bus.

Valor de retorno: El carácter leído, como entero. Devuelve -1 al expirar el tiempo de espera.

UART. lectura en (*buf* [, *nbytes*])

Leer bytes en el [número `buf`]. Si `nbytes` se especifica, se lee como máximo esa cantidad de bytes. De lo contrario, se lee como máximo `len(buf)`.

Valor de retorno: número de bytes leídos y almacenados `buf` durante `None` el tiempo de espera.

UART.readline ( )

Lee una línea que termina en un carácter de nueva línea. Si dicha línea existe, el retorno es inmediato. Si transcurre el tiempo de espera, se devuelven todos los datos disponibles, independientemente de si existe un carácter de nueva línea.

Valor de retorno: la línea leída o `None` el tiempo de espera si no hay datos disponibles.

UART. escritura (*buf*)

Escribe el búfer de bytes en el bus. Si los caracteres tienen 7 u 8 bits de ancho, cada byte representa un carácter. Si los caracteres tienen 9 bits de ancho, se utilizan dos bytes por cada carácter (little endian) y `buf` debe contener un número par de bytes.

Valor de retorno: número de bytes escritos. Si se agota el tiempo de espera y no se escribe ningún byte, devuelve `None`.

`UART.writechar(carácter)`

Escribe un solo carácter en el bus. `char` Es un entero a escribir. Valor de retorno: `None`. Consulte la nota a continuación si se utiliza el control de flujo CTS.

`UART.sendbreak()`

Envía una condición de interrupción al bus. Esto activa el bus a nivel bajo durante 13 bits. Valor de retorno: `None`.

Constantes

`UART. RTS`

`UART.CTS`

para seleccionar el tipo de control de flujo.

Control de flujo

`UART(3)` Admite control de flujo de hardware RTS/CTS utilizando los siguientes pines:

- `UART(3)` está en : `(TX, RX, nRTS, nCTS) = (P4, P5, P1, P2) = (PB10, PB11, PB14, PB13)`

En los siguientes párrafos, el término “objetivo” se refiere al dispositivo conectado al UART.

`init()` Cuando se llama al método del UART con `flow` uno o ambos `UART.RTS` pines `UART.CTS` de control de flujo configurados, `nRTS` es una salida baja activa, `nCTS` es una entrada baja activa con pullup habilitado. Para lograr el control de flujo, la señal de la Pyboard `nCTS` debe conectarse a la del objetivo `nRTS` y la de la Pyboard `nRTS` a la del objetivo `nCTS`.

CTS: el objetivo controla el transmisor OpenMV Cam

Si el control de flujo CTS está habilitado, el comportamiento de escritura es el siguiente:

`UART.write(buf)` Si se llama al método de la cámara OpenMV , la transmisión se detendrá durante cualquier periodo en que `nCTS` sea `False`. Esto resultará en un tiempo de espera si no se transmitió todo el búfer durante dicho tiempo. El método devuelve el número de bytes

escritos, lo que permite al usuario escribir el resto de los datos si es necesario. En caso de tiempo de espera, un carácter permanecerá en el UART pendiente de `nCTS`. El número de bytes que componen este carácter se incluirá en el valor de retorno.

Si `UART.writechar()` se llama cuando `nCTS` es, `False` el método expirará a menos que el objetivo se confirme `nCTS` a tiempo. Si se agota el tiempo, se lanzará. El carácter se transmitirá en cuanto el objetivo se confirme . `OSError 116` `nCTS`

RTS: La cámara OpenMV controla el transmisor del objetivo

Si el control de flujo RTS está habilitado, el comportamiento es el siguiente:

Si se utiliza una entrada almacenada en búfer (`read_buf_len` > 0), los caracteres entrantes se almacenan en el búfer. Si el búfer se llena, el siguiente carácter que llegue provocará que `nRTS` se `False` interrumpa la transmisión `nRTS` cuando `True` se lean los caracteres del búfer.

Tenga en cuenta que el `any()` método devuelve el número de bytes en el búfer. Suponga que la longitud del búfer es de `N` bytes. Si el búfer se llena y llega otro carácter, `nRTS` se establecerá en Falso y `any()` devolverá el valor de conteo `N`. Al leer caracteres, el carácter adicional se almacenará en el búfer y se incluirá en el resultado de una `any()` llamada posterior.

Si no se utiliza una entrada almacenada en búfer (`read_buf_len` == 0), la llegada de un carácter provocará `nRTS` que se ejecute `False` hasta que se lea el carácter.