RIOTOUS User Manual

Realtime Internet Of Things Of Unusual Size

Version 0.2

Table of Contents

Introduction to RIOTOUS

The IoT (internet of things) is rapidly becoming one of the biggest fields of electronics with most house hold devices becoming internet enabled. Even trivial devices such as toasters are being given internet capabilities; allowing users to prepare toast from a remote location using a mobile device. While many microcontrollers available on the market are ideal for such applications, they are not typically hobby friendly being in unfriendly packages such as QFN. IoT modules exist (such as the ESP8266) but are prone to stalling and errors unless they are properly handled.

RIOTOUS is a framework that handles all communication with the ESP8266 and can run on many low end microcontrollers. Written in generic C, the framework uses as little as 140 bytes and 3KB of FLASH making it ideal for low memory, low cost environments. Just some of the devices compatible with RIOTOUS

- PIC16, PIC18, PIC,24, PIC32

But RIOTOUS purpose is not just to be an IoT framework for low end microcontrollers; it is also designed to be simple and user friendly.

The simplicity of RIOTOUS does not end at the microcontroller side; the server side is written in VB.net and can be run on most Windows based machines including tablets. The server class is incredibly simple to use with only a few lines of code needed to get a server operating.

Device Requirements (Client)

For RIOTOUS to function correctly there are a few requirements of the microcontroller running the framework.

- UART peripheral configured to 9600 baud
- At least 4MIPS
- Interrupts (including a UART interrupt)
- 256 bytes of RAM Recommended (140 bytes minimum)
- 3KB ROM
- An ESP8266 module with AT 1.5.4 and SDK 1.1.0.0

User Functions (Client)

RIOTOUS handles 95% of the work needed to get an IoT product working with the ESP8266 module. But since every device is unique, a few RIOTOUS functions have to be defined by the user. The following functions need to be defined by the user which are found in RIOTOUS_IO.c unless you download a specific flavour of RIOTOUS that matches your device (such as RIOTOUS PIC16F1516).

- void interrupt isr(void)
- void riotous_configUART(void)
- void riotous_sendByteUART(unsigned char data)
- void riotous_turnInterruptsOff(void)
- void riotous_turnInterruptsOn(void)
- void riotous_delay(void)

The Main Interrupt - void interrupt isr(void)

RIOTOUS is as interrupt driven as it can possibly be to ensure that there are no infinite loops and/or stalls. When a byte is received over UART, the interrupt service routine must send RIOTOUS framework the received byte by passing the byte to the function riotous_receiveByte(unsigned char byte). For example, the PIC16 would use riotous_receiveByte(RCREG). The interrupt service routine must also handle overflows and framing errors should they occur (PIC devices have a horrible habit of locking up the UART when one of these errors occur).

Delay - void riotous_delay(void)

This delay is imperative for proper ESP8266 communication and should delay for 100ms. This does not disable interrupts and must never allowed to stall (i.e. use a simple while loop with a decrementing counter if possible).

Configure UART - void riotous_configUART(void)

This function configures the UART module to do the following

- Clear the UART buffers
- Configure the baud rate to 9600
- Use 8 bits
- Use 1 stop bit
- Use no parity bits
- Use asynchronous transmission
- Configure the I/O pins for UART transmission / reception
- Fire an interrupt upon receiving a UART byte

Send Byte UART - void riotous_sendByteUART(unsigned char data)

This function needs to send a byte to the UART module for transmission, start the transmission, and wait until the byte has been sent.

Turn Interrupts Off - void riotous_turnInterruptsOff(void)

This function needs to disable any interrupts from firing (i.e. disable global interrupts). This is crucial for RIOTOUS to function properly as C does not do well with atomic instructions. Turning interrupts off for short periods of time guarantees the framework that data buffers and variables are not being altered by external code.

Turn Interrupts On - void riotous_turnInterruptsOn(void)

This function needs to enabled interrupts globally. When RIOTOUS access vital variables, interrupts are disabled and once these sensitive operations have been completed, interrupts are re-enabled with this function.

Using RIOTOUS (Client)

RIOTOUS has been designed with simplicity at its heart keeping the number of user functions lower than 20 (with some of those functions being slight variations of each other). The main functions to the user are given below.

- void riotous_init(void)
- unsigned char riotous_probe(void)
- unsigned char riotous_connectToWiFi(unsigned char *SSID, unsigned char *PASS)
- unsigned char riotous_connectToServer(unsigned char *IP, unsigned char *PORT)
- unsigned char riotous_connectToRiotServer(unsigned char *IP, unsigned char *PORT)
- unsigned char riotous_sendData(unsigned char *sendBuff, unsigned char length)
- unsigned char riotous_sendDataRiotous(unsigned char *sendBuff, unsigned char length)
- unsigned char riotous_getCommandStatus(void)
- unsigned char riotous_wifiStatus(void)
- unsigned char riotous_serverStatus(void)
- unsigned char riotous_dataStatus(void)
- unsigned char riotous_dataSize(void)
- unsigned char *riotous_dataBufferPtr(void)
- unsigned char riotous_timeoutStatus(void)
- void riotous_sendPing(void)

Command Responses

Command related functions return the following

Response	Value (Unsigned char)
Command In progress	0
Command Success	1
Command Fail	2
Command Error	3
Command Timeout	4

void riotous_init(void)

This is the first function that is called after the microcontroller has been configured. This function, when called, will configure the UART module, clear data arrays, set control variables, and send a few commands to the ESP8266 (such as disabling echoing).

unsigned char riotous_timeoutStatus(void)

Earlier version of RIOTOUS had issues with "keep alive" signals. This function returns true if the server has requested a confirmation that the device is still connected. If no response is made within 10 seconds, the connection will be closed by the server.

void riotous_sendPing(void)

Call this function when riotous_timeoutStatus() returns true.

unsigned char riotous_probe(void)

While not entirely important, this function can be used to check if the ESP8266 module is functioning correctly. When called, RIOTOUS will send an AT command to see if the module responds with OK. This function is a command function and returns a command response.

unsigned char riotous_connectToWiFi(unsigned char *SSID, unsigned char *PASS)

This function will connect to the WiFi network whose SSID is equal to the string SSID and use the network key PASS. The strings passed to this function must be 0 terminated but the easiest way to use this function is to write a string into this function. This function is a command function and returns a command response.

unsigned char riotous_connectToServer(unsigned char *IP, unsigned char *PORT)

This function will connect the ESP8266 module to a server whose IP is the string IP and port is the string PORT. Both these strings are zero terminated and C strings are sufficient for this function. This function is a command function and returns a command response.

unsigned char riotous_connectToRiotServer(unsigned char *IP, unsigned char *PORT)

This function will connect the ESP8266 module to a RIOTOUS server whose IP is the string IP and port is the string PORT. Both these strings are zero terminated and C strings are sufficient for this function. This function is a command function and returns a command response.

unsigned char riotous_sendData(unsigned char *sendBuff, unsigned char length)

This function sends the data found in the array sendBuff whose size is defined by length. If a generic data buffer is used, be careful if using sizeof() in place of length as the data put into that buffer may be small in size than the array itself. This function is a command function and returns a command response.

unsigned char riotous_sendDataRiotous(unsigned char *sendBuff, unsigned char length)

This function sends data nfoud in the array sendBuff whose size is defined by length. As opposed to sending just the data found in the array, the data is sent using the basic RIOTOUS data protocol which is required if communicating with a RIOTOUS server. This function is a command function and returns a command response.

unsigned char riotous_getCommandStatus(void)

This function returns the current state of a command currently being executed. If, for example, riotous_connectToWiFi has been called and the module is in the process of connecting to the network, this function would return 0 (command in progress).

unsigned char riotous_wifiStatus(void)

This function returns the current status of the WiFi connection. If a WiFi connection has been established then true (1) is returned, else false (0).

unsigned char riotous_serverStatus(void)

This function returns the current status of the server connection. If a server connection has been established then true (1) is returned, else false (0).

unsigned char riotous_dataStatus(void)

This function returns the current state of the data engine. If there is data in the data buffer that is complete (a complete proper packet), and it has not be dealt with then this function returns true (1), else false (0). Once called, this will return false.

unsigned char *riotous_dataBufferPtr(void)

This function returns a pointer that points to the RIOTOUS data buffer. To ensure that RIOTOUS functions correctly, copy data from the RIOTOUS buffer into your own array buffer immediately or use the data in this buffer as soon as possible as new incoming data will override the data currently stored.

unsigned char riotous_dataLength(void)

This function returns the length of the data in the RIOTOUS data buffer.

Other Functions

RIOTOUS also contains other functions that are required for the framework to work correctly but these functions can also be used by the user safely (and may even be helpful too).

*unsigned char strCompare(unsigned char *strA, unsigned char *strB)*

This functions returns 1 if the two strings strA and strB are equal other wise will return 0. These strings must be zero terminated.

*unsigned char textToByte(unsigned char *str)*

This function converts a base 10 string number (such as 120) to a byte.

RIOTOUS Client Examples

The examples given below are for the PIC16F1516 programmed using MPLAB X IDE, XC8, and utilizing the PICKIT3 for hardware flashing.

RIOTOUS_IO.c Example

```
// ------------------------------------------------------------------------------
// Description  : RIOTOUS User IO : You need to fill these functions in!
// Author       : Robin Mitchell
// ------------------------------------------------------------------------------
// This must configure the UART module to fire an interrupt on reception
// It must also change the baud rate to the ESP8266 rate (9600), use 8 bits,
// no parity, and one stop bit. It may be a good idea to configure the port pins
// here!
// ------------------------------------------------------------------------------
void riotous_configUART(void)
{
    unsigned char garbage;          // Used for clearing registers
    garbage = RCREG;
    garbage = RCREG;
    garbage = RCREG;

    TXSTAbits.BRGH = 0;             // High Baud Rate
    BAUDCONbits.BRG16 = 0;          // 16 Bit SPBRG
    SPBRGH = 0x00;
    SPBRG = 25;

    TXSTAbits.SYNC = 0;             // Async communication
    RCSTAbits.SPEN = 1;             // Enable serial port
    TXSTAbits.TXEN = 1;             // Enable transmission
    RCSTAbits.CREN = 1;             // Enable reception
    PIE1bits.RCIE = 1;              // Enable reception interrupt
    PIR1bits.RCIF = 0;              // Enable reception interrupt

    garbage = RCREG;
    garbage = RCREG;
    garbage = RCREG;
    garbage = RCREG;

    TRISCbits.TRISC6 = 1;
    TRISCbits.TRISC7 = 1;
}


// ------------------------------------------------------------------------------
// This must send a byte to the UART transmitter register
// ------------------------------------------------------------------------------
void riotous_sendByteUART(unsigned char data)
{
    TXREG = data;                   // Transfer byte to UART send register
    while(!TXSTAbits.TRMT);         // Wait until byte sent
}
```

```
// ------------------------------------------------------------------------------
// This turns interrupts off during crucial variable accesses. Probably the
// most important function here!
// ------------------------------------------------------------------------------
void riotous_turnInterruptsOff(void)
{
    INTCONbits.GIE  = 0;
}


// ------------------------------------------------------------------------------
// This turns interrupts on once key variables have been altered
// ------------------------------------------------------------------------------
void riotous_turnInterruptsOn(void)
{
    INTCONbits.GIE  = 1;
}


// ------------------------------------------------------------------------------
// This should delay for 100ms
// ------------------------------------------------------------------------------
void riotous_delay(void)
{
    for(unsigned int i = 0; i < 1000; i++)
    {
    }
}
```

IoT Controlled LED

```
#include "../include/config.h"
#include "../RIOTOUS/RIOTOUS.h"

const char ledon[] = "DLEDON";
const char ledoff[] = "DLEDOFF";

void main(void)
{
    // Configure PIC16F1516
    configure();

    // Initialise RIOTOUS
    riotous_init();

    // Enable Global Interrupts
    INTCONbits.GIE  = 1;

    // Indicator LEDs
    LATCbits.LATC0 = 0;
    LATCbits.LATC1 = 0;
    LATCbits.LATC2 = 0;

    // Check the ESP8266 is connected
    while(riotous_probe() != 1);
    LATCbits.LATC0 = 1;

    // Connect the ESP8266 to the WiFi
    while(riotous_wifiStatus() == 0)
    {
        while(riotous_connectToWiFi("TP-LINK_CA77","06426705") != 1);
    }
    LATCbits.LATC1 = 1;

     // Connect the RIOTOUS Server
    while(riotous_serverStatus() == 0)
    {
        while(riotous_connectToRiotServer("192.168.1.72","333") != 1);
    }
    LATCbits.LATC2 = 1;

    // Indicator LEDs
    LATCbits.LATC0 = 0;
    LATCbits.LATC1 = 0;
    LATCbits.LATC2 = 1;

    // Set unique ID to 'A'
    riotous_sendData("IA", 2);


    while(1)
    {
        if(riotous_dataStatus() == 1)
        {
            if(strCompare(riotous_dataBufferPtr(), ledon))
            {
                LATCbits.LATC1 = 1;
            }
             if(strCompare(riotous_dataBufferPtr(), ledoff))
            {
                LATCbits.LATC1 = 0;
            }
        }

        // Timeout checker - THIS IS COMPULSORY
        if(riotous_timeoutStatus() == 1)
        {
            riotous_sendData("P", 1);
        }
    }
}
```

RIOTOUS Server

While RIOTOUS can operate with most servers it is mainly designed to operate with a RIOTOUS server. The server side code handles data transfers, client connections, timeouts, and port assigning while keeping every connected client in its own thread. Only a few functions are needed to be called by the user to start the server and once started, only a few functions are needed to send and receive data to and from clients.

Since the RIOTOUS server code is a VB.net class, you can use any project type including forms, command line, and even an XNA project if desired.

Server Class

The first task in creating a RIOTOUS server is creating an instance of the server class. Two files require to be imported into you project; Server.vb and client.vb. Server.vb contains the server class itself while client.vb contains the client class which is called by the server. Below is the code needed to create an instance of the RIOTOUS server.

```
Dim IoTServer As Server
IoTServer = New Server
```

With the server instance created, the server needs to be started. This is done by calling the function startServer and passing two arguments; the IP address of the machine executing the server, and the port that the server will advertise on.

```
IoTServer.startServer("192.168.0.1", "333")
```

While you may choose any port to advertise the server on it is best to use 333 as its a commonly unused port. Once the server starts, it will await for device connections and once a device connects, the server will find a free port slot (starting from 60000 upward), tell the client which port to reconnect to, and then close the connection.

Server – Receiving Data

The RIOTOUS server class requires to be probed to see if there is new data ready in the client buffers. The easiest way to do this is to use a timer that ticks frequently (such as 100ms), iterate through each client, and then check for new data. If there is data, then either the buffer can be accessed directly (not advised) or a buffer can be passed to RIOTOUS whereby the awaiting data is transferred to the specified buffer.

```
' ------------------------------------------- '
' Upon a tick, check the server for information '
' ------------------------------------------- '
Private Sub serverTick_Tick(sender As Object, e As EventArgs) Handles serverTick.Tick

    numberOfClients = 0

    Try
        For currentClient As Integer = 0 To Server.MAX_CONNECTION

            If IoTServer.isClientConnected(currentClient) = True Then
                numberOfClients += 1
            End If

            If IoTServer.clientDataAvailable(currentClient) Then

                ' Get the data that has just arrived '
                lightLevel = IoTServer.clientBuffers(currentClient, 0)

                ' Clear the data received flag! '
                IoTServer.clientDataAvailable(currentClient) = False

                ' Plot the data! '
                dataLog.Series("loggedLight").Points.Add(lightLevel)

            End If

        Next
    Catch ex As Exception

    End Try

    numberOfConnections.Text = numberOfClients

End Sub
```