Blynking an IoT Yunshan ESP8266 250V 10A AC/DC WIFI Network Relay Module

Posted on December 18, 2016 by Jim Eli

I purchased a few of these Yunshan Wifi Relays through ebay for approximately $7.50US. The device should be perfect for use in simple IOT projects which require controlling household AC power. The onboard JQC-3FF relay is rated to 250VAC or 30VDC at up to 12A. There are routered slots between the high voltage PCB traces for circuit isolation and arc-over protection. Transient voltage suppression is incorporated on both the board power supply and the photocoupler (see description below) input line.

The device requires a power supply between 7 and 30V DC. I unsuccessfully attempted to run it with an inexpensive 5V, 2A wall-wort, even though the onboard MP2303 buck converter is rated down to 4.8V. I did get it to operate successfully using a 9VDC wall-wort.

The device contains an integrated ESP8266-12E, but appears to only use the GPIO4 and GPIO5 pins. That was a disheartening discovery because it discards a significant amount of functionality inside the ESP8266 WIFI module. Hence the ESP8266 low power, wake from sleep provisions (where GPIO16 and RESET need to be linked together) would require some skillful soldering of the module's exposed pins.

The good news is, programming the module is very easy, as I discuss later. I also found the overall build quality of my device to be above the typical level found on ebay-sourced Chinese electronics.

The ebay listing contained a link to a zip file, entitled *U4648-datasheet*, which contained example programs, schematics, and a Chinese manual. Through the Google translation service I managed to translate the manual, but there's no reason to do that, as there isn't much there. More information can be learned from a quick study of the schematic and the board itself.

Module Description

The Chinese manual presents the following limited module description:

1 – The power input terminals.
2 – The relay output terminals.
3 – IO input terminal.
4 – Enter the status indicator, IO input high when lit, blue light.
5 – The relay output status indicator, the relay is turned on, the red light.
6 – TTL serial output.
7 – Boot mode selection jumper.

Board Connectors

Here are the connections on my board:

A: 7-30V+ DC power supply
B: Power supply ground
C: Normally closed (NC) relay contact
D: Common (COM) relay contact
E: Normally open (NO) relay contact
F: 5V+ out
G: ESP8266 GPIO5 Optocoupler Input
H: Ground (isolated optocoupler input)

AP MODE Webpage

I was easily able to connect a 9V power supply to the A-B connector (see above picture and connector description) and control the device via WIFI. To do this, simply connect your computer or phone to the *yunshan_wifi_xx_xx_xx* network (where it appears the xx are hexadecimal numbers pulled from the ESP8266 MAC address). My device responded to the supplied password of *yunshan123456789*. Once a connection was established, I simply entered the IP address of 192.168.4.1 into my browser. Once there, I was greeted by a Chinese web page, the translation of which appears below. From this webpage, I was able to open and close the relay. The status of the GPIO5 optocoupler input is also displayed on this webpage.

Since I have big IOT home automation plans for these devices, my next task was to attempt a re-program of the onboard ESP8266 module. For a quick test, I uploaded the traditional Arduino IDE ESP8266 blink program, and was rewarded with a 1Hz blinking blue LED on the ESP8266 module.

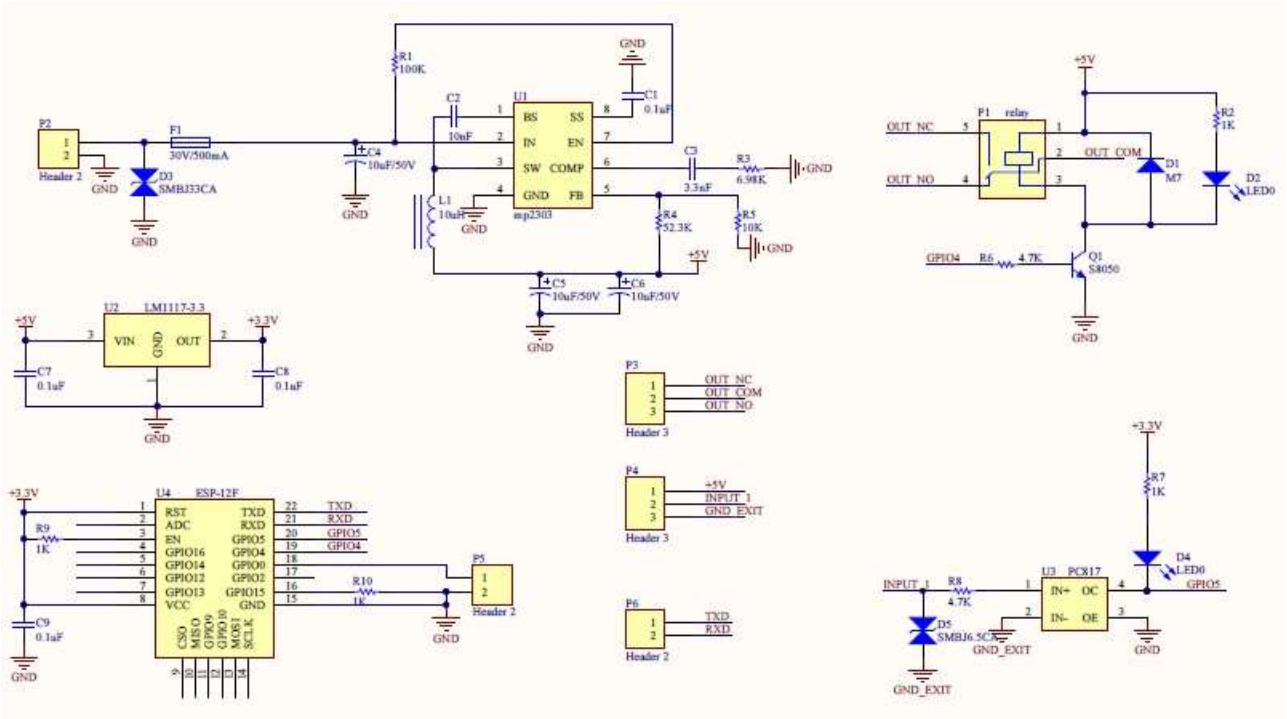
Program Upload

On the lower left portion of the PCB is a section that grants access to the ESP8266 pins for programming (see the above photo). These same pins are also useful for TTL serial output purposes (debugging, etc.). Separate 2 and 3-pin headers will need to be soldered into these connector holes (labeled P5 and P6). The ESP8266 GPIO4 controls the relay through a 2N3904 transistor. Setting GPIO4 high, causes the relay to close the NO contact with Common and the NC contact to open. Additionally, taking connector "G" high causes GPIO5 to also go low isolated via a PC817 photocoupler. On my board the blue LED is connected to GPIO2, and can be illuminated by pulling the pin low.

To program the ESP8266 module, I connected the TX, RX and ground pins of connector P6 to a SparkFun USB FTDI programmer, and jumped the two pins of connector P5 together when I was ready to upload. Connector P5 grounds GPIO0 and GPIO15, sending the device into bootloader mode. If you have trouble programming the ESP8266 like I did on the first attempt, ensure you also ground your FTDI device through the P6 connector.

A very good introduction to the ESP8266 module can be found here. Excellent programming information for the individual ESP8266 modules is also widely available (two examples: ESP8266-01 and ESP8266-12e).

Board Schematic



Blynk Relay Control Application

```
/*************************************************************************

* Title: Simple ESP-8266 blynk/yunshan wifi relay control

* File: esp8266_yunshan_relay.ino

* Author: James Eli

* Date: 12/25/2016

*

* This program controls a Yunshan wifi relay module communicating through

* the onboard esp-8266-12e module. The module is controlled from the

* internet via the Blynk cloud app.

*
```

```
* Notes:
*  (1) Requires the following arduino libraries:
*      ESP8266
*      Blynk
*  (2) Compiled with arduino ide 1.6.12
*  (3) Uses three Blynk app widgets:
*       V0: button configured as a switch.
*       V1: led.
*       V2: led.
************************************************************************
* Change Log:
*   12/25/2016: Initial release. JME
*   12/31/2016: Added input pin status. JME
*   01/15/2017: Added volatile. JME
************************************************************************/
#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>


// Esp8266 pins.
#define ESP8266_GPIO2    2 // Blue LED.
#define ESP8266_GPIO4    4 // Relay control.
```

```
#define ESP8266_GPIO5    5 // Optocoupler input.

#define LED_PIN          ESP8266_GPIO2

// Blynk app authentication code.

char auth[] = "***";

// Wifi SSID.

const char ssid[] = "***";

// Wifi password.

const char password[] = "***";

// Flag for sync on re-connection.

bool isFirstConnect = true;

volatile int relayState = LOW;    // Blynk app pushbutton status.

volatile int inputState = LOW;    // Input pin state.



void setup() {

 pinMode( ESP8266_GPIO4, OUTPUT );      // Relay control pin.

 pinMode( ESP8266_GPIO5, INPUT_PULLUP ); // Input pin.

 pinMode( LED_PIN, OUTPUT );            // ESP8266 module blue LED.

 digitalWrite( LED_PIN, LOW );          // Turn on LED.

 Blynk.begin( auth, ssid, password );    // Initiate Blynk conection.

 digitalWrite( LED_PIN, HIGH );         // Turn off LED.

}
```

```
// This function runs every time Blynk connection is established.
BLYNK_CONNECTED() {
  if ( isFirstConnect ) {
    Blynk.syncAll();
    isFirstConnect = false;
  }
}

// Sync input LED.
BLYNK_READ( V2 ) {
  CheckInput();
}

// Blynk app relay command.
BLYNK_WRITE( V0 ) {
  if ( param.asInt() != relayState ) {
    relayState = !relayState;                // Toggle state.
    digitalWrite( ESP8266_GPIO4, relayState ); // Relay control pin.
    Blynk.virtualWrite( V1, relayState*255 );  // Set Blynk app LED.
  }
```

```
}



// Debounce input pin.

int DebouncePin( void ) {

  // Read input pin.

  if ( digitalRead( ESP8266_GPIO5 ) == HIGH ) {

    // Debounce input.

    delay( 25 );

    if ( digitalRead( ESP8266_GPIO5 ) == HIGH )

      return HIGH;

  }

  return LOW;

}



// Set LED based upon state of input pin.

void CheckInput( void ) {

  if ( DebouncePin() != inputState ) {

    Blynk.virtualWrite( V2, inputState*255 );

    inputState = !inputState;

  }

}
```

```
// Main program loop.
void loop() {
 Blynk.run();
 CheckInput();
 //yield(); //Updated: 3/8/2017
}
```

TCP Client Demo

Here is a basic server which responds to TCP client HTTP GET commands (added 1/8/17):

```
#include <ESP8266WiFi.h>

// Esp8266 pinouts
#define ESP8266_GPIO2    2  // Blue LED.
#define ESP8266_GPIO4    4  // Relay control.
#define ESP8266_GPIO5    5  // Optocoupler input.
#define LED_PIN          ESP8266_GPIO2
// WiFi Definitions.
const char ssid[] = "***";
const char pswd[] = "***";
WiFiServer server( 80 );
```

```
volatile int relayState = 0;      // Relay state.



void setup() {

 initHardware();

 connectWiFi();

 server.begin();

}



void GetClient( WiFiClient client ) {

 // Read the first line of the request.

 String req = client.readStringUntil( '\r' );

 Serial.println( req );

 client.flush();



 String s = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n<!DOCTYPE
HTML>\r\n<html>\r\n";



 if ( req.indexOf( "OPTIONS" ) != -1 ) {

   s += "Allows: GET, OPTIONS";



 } else if ( req.indexOf( "GET" ) != -1 ) {
```

```
if ( req.indexOf( "open" ) != -1 ) {

  // relay on!

  s += "relay on!";

  relayState = 1;

  digitalWrite( ESP8266_GPIO4, 1 ); // Relay control pin.



} else if ( req.indexOf( "close" ) != -1 ) {

  // relay off!

  s += "relay off!";

  relayState = 0;

  digitalWrite( ESP8266_GPIO4, 0 ); // Relay control pin.



} else if ( req.indexOf( "relay" ) != -1 ) {

  if ( relayState == 0 )

    // relay off!

    s += "relay off!";

  else

    // relay on!

    s += "relay on!";



} else if ( req.indexOf( "io" ) != -1 ) {
```

```
    if ( digitalRead( ESP8266_GPIO5 ) == 0 )

      s += "input io is:0!";

    else

      s += "input io is:1!";



  } else if ( req.indexOf( "MAC" ) != -1 ) {

    uint8_t mac[WL_MAC_ADDR_LENGTH];

    WiFi.softAPmacAddress( mac );

    String macID = String( mac[WL_MAC_ADDR_LENGTH - 5], HEX) + String(
mac[WL_MAC_ADDR_LENGTH - 4], HEX) +

                String( mac[WL_MAC_ADDR_LENGTH - 3], HEX) + String( mac[WL_MAC_ADDR_LENGTH -
2], HEX) +

                String( mac[WL_MAC_ADDR_LENGTH - 1], HEX) + String(
mac[WL_MAC_ADDR_LENGTH], HEX);

    macID.toUpperCase();

    s += "MAC address: " + macID;



  } else

    s += "Invalid Request.<br> Try: open/close/relay/io/MAC";



 } else

  s = "HTTP/1.1 501 Not Implemented\r\nContent-Type: text/html\r\n\r\n<!DOCTYPE
HTML>\r\n<html>\r\n";
```

```
  client.flush();
  s += "</html>\n";

  // Send the response to the client.
  client.print( s );
  delay( 1 );
  Serial.println( "Client response sent." );
}

void loop() {
  // Check if a client has connected.
  WiFiClient client = server.available();
  if ( client )
    GetClient( client );
}

void connectWiFi() {
  byte ledStatus = LOW;
  Serial.println();
  Serial.println( "Connecting to: " + String( ssid ) );
```

```
 // Set WiFi mode to station (as opposed to AP or AP_STA).
 WiFi.mode( WIFI_STA );

 // WiFI.begin([ssid], [passkey]) initiates a WiFI connection.
 // to the stated [ssid], using the [passkey] as a WPA, WPA2, or WEP passphrase.
 WiFi.begin( ssid, pswd );

 while ( WiFi.status() != WL_CONNECTED ) {
  // Blink the LED.
  digitalWrite( LED_PIN, ledStatus ); // Write LED high/low.
  ledStatus = ( ledStatus == HIGH ) ? LOW : HIGH;
  delay( 100 );
 }

 Serial.println( "WiFi connected" );
 Serial.println( "IP address: " );
 Serial.println( WiFi.localIP() );
}

void initHardware() {
 Serial.begin( 9600 );
```

```
 pinMode( ESP8266_GPIO4, OUTPUT );      // Relay control pin.

 pinMode( ESP8266_GPIO5, INPUT_PULLUP ); // Input pin.

 pinMode( LED_PIN, OUTPUT );            // ESP8266 module blue LED.

 digitalWrite( ESP8266_GPIO4, 0 );      // Set relay control pin low.

}
```