黑客派

基于ESP8266芯片的物联网解决方案

作者：liuxin

原文链接：https://hacpai.com/article/1459084029196

来源网站：黑客派

     好久没来更新文章了，知道忙只是一种借口，所以我就不说了。最近改开源的目改的头大，开始进入正题。

     让一硬件设备联网有很多种实现方案，今天我来说一下我利用DDPUSH+ESP866实现设备入网的方案。
<h2 id="toc_h2_0">   通信框图 </h2>

      上面这张图是方案的通信流程，至于心跳包的时间这个数是不一定就是10s。
      ESP8266这款芯片我比较看重它的价格已经开放的SDK特性如下：
 l 支持无线802.11 b/g/n 标准
 l 支持STA/AP/STA+AP 三种工作模式
 l 内置TCP/IPan>协议栈，支持多路TCP Client连接
 l 支持丰富的Socket AT指令
 l 支持UARTn>/GPIO数据通信接口
 l 支持Smart Link 智能联网功能
 l 支持远程固件升级（OT）
 l 内置32位MCU，可兼作应用处理器/span>
 l 超低能耗，适合电池供电应用an>
 l 3.3V 单电源供电
     在这个方案中，wifi新片作为一个数据透传模块，编程实现smartconfi功能，用户在手机端配置wifi联网，wifi联网成功后启动连接tcp服务器的一个任务，连接成功后开启跳进程，与服务器保持联系，接下来进入数据透传模式。当中途服务器出现意外断掉后，wifi模块会己不断尝试重连服务器。在路由器出现意外断掉之后，wifi芯片也会自己一直重连路由。如果用户更了wifi密码，只需按下wifi芯片上的一个按钮，给IO13口输入一个高电平产生一个下降沿脉冲，wifi芯就会进入智能连接模式，IO14口会输出一个200ms的脉冲，如果在IO14接入一个二极管会发现二极不停闪烁。这个时候用户需要在手机端输入wifi的名称和密码，进行智能连接。wifi芯片收到账号和密后会去尝试连接路由器，接下来依次进入正常工作流程。忘了补充，这款wifi芯片有512K/1M的flash储空间，它每次在连接路由器成功之后会可以程序实现将账号密码保存到flash以便在重启的时候用此号自动连接路由器。为了提高wifi芯片和服务器之间的通信稳定性，在wifi芯片的一个定时器(也是唯一个可用的定时器)实现如下逻辑：

```
<p> <span> </span> </p>
<pre class="prettyprint lang-cpp">timer(void)
{
   smart_led_count++;
   count_2s++;
   count_30s++;
   if(count_2s==20)
   {
      status = wifi_station_get_connect_status();
      count_2s=0;
      os_printf("\r\n tcp_state %d \r\n",tcp_connect_flag);
      os_printf("\r\n smart_config state %d \r\n",_smart_config);
   }</pre>
<pre class="prettyprint lang-cpp">&amp;nbsp;&amp;nbsp;&amp;nbsp;/********30秒定时*******************
*/
   if(count_30s==300){
        status = wifi_station_get_connect_status();
/**************************TCP检测状态**************************/
      if(tcp_connect_flag==5||tcp_connect_flag==7||tcp_connect_flag==2){
         if(status == STATION_GOT_IP){
             os_printf("\r\n check tcp is not connect begin connect tcp \r\n");
            tcp_connect_flag=2;
            disConnect();
            system_os_task(TCP_init,1,&amp;amp;Taskqueue,1); //建立远程连接
            system_os_post(1,0,0);
         }
      }
/**************************TCP检测状态 end**************************/
      count_30s=0;
      if(tcp_connect_flag==3){
//          发送心跳包
         InterNet_TCP_SendData(heart,21);
      }
   }
   if(smart_led_count==2){//当前是智能匹配状态
      smart_led_count=0;
      if(_smart_config==1){
         if(led==0){
            led=1;
            GPIO_OUTPUT_SET(GPIO_ID_PIN(14), 0);
         }
         else {
            led=0;
            GPIO_OUTPUT_SET(GPIO_ID_PIN(14), 1);
         }
      }
   }
   if(tcp_connect_flag==1){
       os_printf("\r\n begin connect tcp \r\n");
       tcp_connect_flag=2;
       system_os_task(TCP_init,1,&amp;amp;Taskqueue,1); //建立远程连接
       system_os_post(1,0,0);
   }
}
```

/*****************100 ms定时器 end*********************/</pre>
<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></scr
pt>
<!-- 黑客派PC帖子内嵌-展示 -->
<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342"
data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></in
>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>
<h2 id="toc_h2_1"> 定时器里面在做什么事情？ </h2>
<h4 id="toc_h4_2"> 一、发送心跳包 </h4>
 在这里每间隔30秒给服务器发送一个心跳包，数据长度为21：

<pre class="prettyprint lang-cpp">uint8 heart[21]={0x01,0x01,0x00,0xda,0x75,0x0f,0xf9,0x98,
x6a,0x86,0x6a,0x12,0x21,0x83,0xa0,0x6c,0xcd,0x38,0x91,0x00,0x00};</pre> 心跳包的内容是：
是谁(16位uuid)？我发的是什么数据类型（心跳包）？

 发送心跳包的原因在上篇文章里面也说了，待会还会再补充。在这个定时器里面，</sp
n>

<h4 id="toc_h4_3"> 二、检测路由器异常 </h4>
 每2秒检测一次，查看是否正常连接路由器，主要是为了避免路由器突然挂掉了，如果状态出现
异常可以实现wifi状态灯的指示，并且尝试重新连接路由器。
<h4 id="toc_h4_4"> 三、检测TCP连接异常 </h4>
 tcp连接状态设置了一个变量，在任何tcp故障的时候这个标志位会发生改变，所以在定时器里
每间隔30秒会检测一次TCP的连接标志位，如果出现了连接异常会启动重新建立连接的任务。
<h2 id="toc_h2_5"> 如何实现智能连接？ </h2>
 有一个智能连接的调用函数，还有一个按键中断。IO13如果出现下降沿会启动智能连接函数，I
14会输出200ms的脉冲。

<pre class="prettyprint lang-cpp">/******************按键初始化**************************/
/*设置13号角为输入，14号角为输出
 */
void key_Init(void)
{
    PIN_FUNC_SELECT(PERIPHS_IO_MUX_MTCK_U,FUNC_GPIO13);
    GPIO_DIS_OUTPUT(GPIO_ID_PIN(13));
<pre><code class="highlight-chroma">PIN_FUNC_SELECT(PERIPHS_IO_MUX_MTMS_U, FUN
_GPIO14);
GPIO_OUTPUT_SET(GPIO_ID_PIN(14), 0);
</code></pre>
}
/smart config</stro
g>**********************/
void ICACHE_FLASH_ATTR
smartconfig_done(sc_status status, void *pdata)
{
switch(status) {
case SC_STATUS_WAIT:
os_printf("SC_STATUS_WAIT\n");
break;
case SC_STATUS_FIND_CHANNEL:
_smart_config=1;

os_printf("SC_STATUS_FIND_CHANNEL\n");
break;
case SC_STATUS_GETTING_SSID_PSWD:
os_printf("SC_STATUS_GETTING_SSID_PSWD\n");
sc_type *type = pdata;
if (*type == SC_TYPE_ESPTOUCH) {
os_printf("SC_TYPE:SC_TYPE_ESPTOUCH\n");
} else {
os_printf("SC_TYPE:SC_TYPE_AIRKISS\n");
}
break;
case SC_STATUS_LINK:
os_printf("SC_STATUS_LINK\n");
struct station_config *sta_conf = pdata;
<pre><code class="highlight-chroma">   wifi_station_set_config(sta_conf);
        wifi_station_disconnect();
        wifi_station_connect();
        break;
    case SC_STATUS_LINK_OVER:
        os_printf("SC_STATUS_LINK_OVER\n");
        _smart_config=0;
        tcp_connect_flag=1;
        if (pdata != NULL) {
            uint8 phone_ip[4] = {0};

}
</code></pre>

}
/smart config end</stron
>*****************/
void smart_config_(){
<pre><code class="highlight-chroma">smartconfig_set_type(SC_TYPE_ESPTOUCH); //SC_TY
E_ESPTOUCH,SC_TYPE_AIRKISS,SC_TYPE_ESPTOUCH_AIRKISS
wifi_set_opmode(STATION_MODE);
smartconfig_start(smartconfig_done);
</code></pre>
}
/***

FunctionName : GPIO13 上升沿中断
***/
void GPIO_INTER(void)
{
status = wifi_station_get_connect_status();
if(status == STATION_GOT_IP){
_smart_config=0;
return ;
}

</pre>

<h4 id="toc_h4_6"> 智能连接成功后里面干了些什么 </h4>
<pre class="prettyprint lang-cpp">   case SC_STATUS_LINK_OVER://路由器连接完成
            os_printf("SC_STATUS_LINK_OVER\n");
            _smart_config=0;
            tcp_connect_flag=1;
            smartconfig_stop();
            break;</pre> 连接路由器成功后会设置tcp连接标志位，在定时器里面会检测到并开始准备接TCP了。

<h4 id="toc_h4_7"> 智能连接失败会干什么 </h4>
 它会一直重启smartconfig模式

<h2 id="toc_h2_8"> TCP接受数据处理部分 </h2>

<pre class="prettyprint lang-cpp">static void ICACHE_FLASH_ATTR
InterNet_Receive(void *arg, char *pdata, unsigned short len)
{
os_printf("%s",pdata);//串口输出
if(pdata[2]==0x20){
akServer0x20();//如果类型为0x20则需要回执服务器已经收到数据了,为什么要返回？由你设计的通协议所决定
}
}</pre>
<h2 id="toc_h2_9"> 服务器端思路 </h2>
 服务器主要是TCP服务器。我改了一个开源的代码，DDPUSH上次在微信里面推荐过。官网为:http://www.ddpush.net/overview。这玩意本来是用做移动推送的，就是我们手机QQ啊一类的APP接受消息的和发送消息的，我把它们的方案移植到这来。先来简单介绍下DDPUSH的通信流程吧。
 <img src="https://static.hacpai.com/images/img-loading.svg" alt="" data-src="http://liuin.lxliu.cn/blog/images/ddpush.png">
 它是在服务端建了两个TCP监听端口，一个9966TCP或者UDP，一个是TCP的listenpush端口999.listenpush主要用来推送消息的。客户端也是两个端口，一个是与9966立长连接，一个是向9999推送信息。他们给的客户端程序是Android，wifi用不了，所以我折腾了好天把客户端改成了直接用与服务器9966建立的tcp连接发送心跳包和数据包，9999客户端我扔掉了。为他们的流程是建立9966的连接只负责发送心跳数据和接受数据，而客户端每次要向服务器发送数据要新建一次连接，也就是没发一次数据都要三次握手，这对于移动推送倒没什么，对于这种实时性要比较高的系统延时还是比较长的，这是wifi跟远程服务器建立tcp连接，你想想每次我发个数据都要和程服务器进行三次握手才能发和我直接利用现有的已经建立好的数据通道发数据，哪个更快？我这样的当然也有一点不好的地方，就是服务器负载变点大了，牺牲并发性能来提升实时性能，值了。服务也要对接受到的数据进行处理，所以服务器端的9966以前只接受心跳数据现在也被我改成同时接受自义数据，收到后创建新的线程立即处理。
<h2 id="toc_h2_10"> 总结 </h2>
 如此让wifi芯片连上了因特网，wifi的串口又可以和硬件设备进行一个直连。最终达到的目的是

硬件上网，如何让硬件变得智能，就需要在服务端进行开发了。
 有兴趣的可以去买块板子测试完一完，提供测试程序：
 烧录程序链接：http://pan.baidu.com/s/1dFdiJcP 密码：63zb
 smart config app链接：http://pan.baidu.com/s/1c2NRWvm 密码：51jf
<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></script>
<!-- 黑客派PC帖子内嵌-展示 -->
<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342" data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></ins>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>

 配置：
 1、 IO13下降沿触发进入smartconfig模式
 2、 等待IO14输出脉冲后开始手机APP下发账号密码
 3、 正常工作串口会打印tcp_state=3;smart_config_=0;
 服务器接口
 控制命令：
  http://liuxin.lxliu.cn /tcp/controllo?uuid=20160001&level=1&io=14
 uuid：设备编号，字符串
 io:需要获取的IO端口号
 level:控制电平
 获取状态：
 <a href="https://link.hacpai.com/forward?goto=http%3A%2F%2Fliuxin.lxliu.cn%Ftcp%2FgetIoState%3Fuuid%3D20160001%26io%3D14" target="_blank" rel="nofollow ugc" http://liuxin.lxliu.cn/tcp/getIoState?uuid=20160001&io=14
 uuid：设备编号，字符串
 io:需要获取的IO端口号

<h2 id="toc_h2_11">   </h2>