GoKit3 DEV SDK
Common 版移植说明

机智云

编制人	Ture Zhang	审核人	Andy Gao	批准人	
产品名称		产品型号		文档编号	
会签日期			版本	V0.1.0	

GizWits

修改记录：

修改时间	修改记录	修改人	版本	备注
20160927	初建	TureZhang	0.1.0	
20160930	二版修改	TureZhang	0.1.1	

目录

1. 前文需知

1.1 什么是“代码自动生成工具”？

为了降低开发者的开发门槛，缩短开发周期，降低开发资源投入，机智云推出了代码自动生成服务。云端会根据产品定义的数据点生成对应产品的设备端代码。

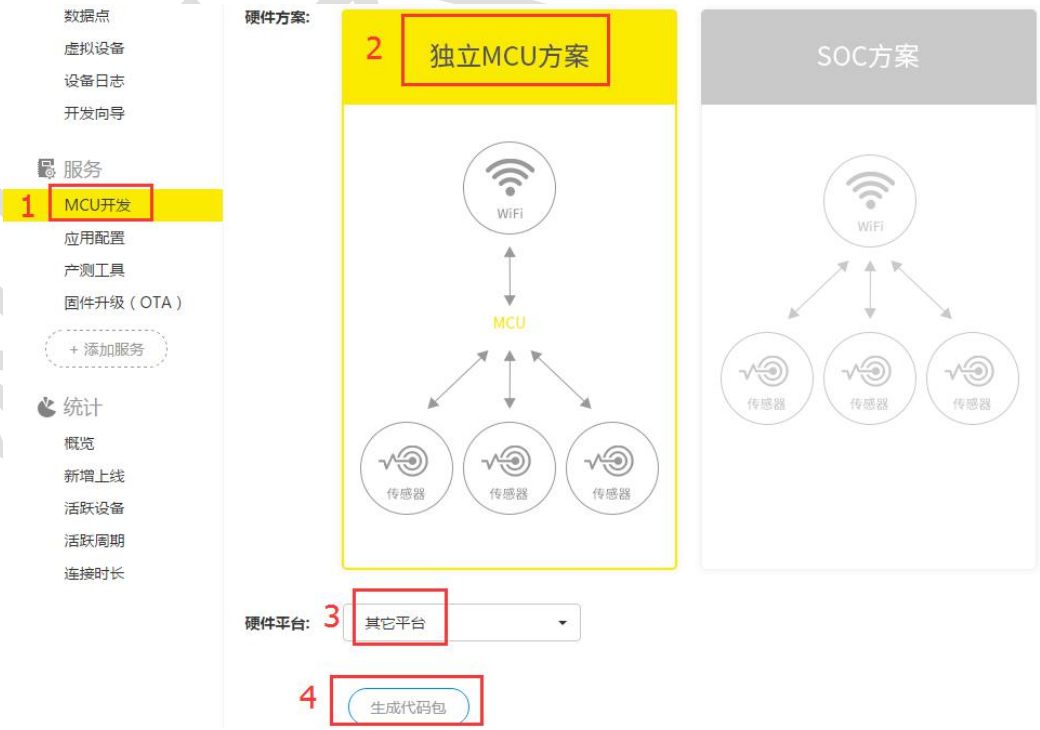
自动生成的代码实现了机智云通信协议的解析与封包、传感器数据与通信数据的转换逻辑，并封装成了简单的 API，且提供了多种平台的实例代码。当设备收到云端或 APP 端的数据后，程序会将数据转换成对应的事件并通知到应用层，开发者只需要在对应的事件处理逻辑中添加传感器的控制函数，就可以完成产品的开发。

MCU 方案默认支持 **STM32F103C8x** 平台，如果是其他 MCU 芯片，可以将我们生成好的**通用平台版代码**移植到符合条件的平台，从而实现机智云所提供的各种功能。

本文将主要说明**通用平台版**的移植。

1.2 如何自动生成“通用平台代码”？

在机智云平台定义一个产品后，选择左侧服务中的“MCU 开发”，选中硬件方案中的“独立 MCU 方案”，再选中“硬件平台”中的“其他平台”，最后点击“生成代码包”，等待生成完毕下载即可。



下载完成后解压如下：

1.3 通用平台移植需知

开发者在移植前要确保被移植平台的硬件参数满足以下的要求：

A.平台支持两个串口接口（至少一个），一个负责与 wifi 模组间的数据收发（必须），一个用于调试信息打印（可复用数据收发串口）。

B.平台支持定时器功能（1ms 精确定时）。

C.平台支持至少 2K 的 RAM 空间（可调整环形缓冲区大小来解决此问题，但易导致数据协议的处理异常）。

注：环形缓冲区修改位置: **gokit_mcu_stm32_xxx\Gizwits\gizwits_protocol.h**

```
42  #define MAX_PACKAGE_LEN      950                    ///< 数据缓冲区最大长度
43  #define RB_MAX_LEN           (MAX_PACKAGE_LEN*2)    ///< 环形缓冲区最大长度
```

原代码中 MAX_PACKAGE_LEN = 950，即 环形缓冲区所占 RAM 空间大小为 950*2 = 1900 字节，开发者可以此来调整程序所占 RAM 空间的大小。

2. 通用平台版代码移植说明

2.1 文件介绍

```
.
├── Gizwits                                  //机智云协议层,包括通信协议和用户事件处理
│   ├── gizwits_product.c
│   ├── gizwits_product.h
│   ├── gizwits_protocol.c
│   └── gizwits_protocol.h
└── User                                     //APP层,main为程序入口函数
    └── main.c
```

重要文件解读：

1. gizwits_product.c
   该文件为产品相关处理函数，如 gizwitsEventProcess()。
2. gizwits_product.h
   该文件为 gizwits_product.c 的头文件，如 HARDWARE_VERSION、SOFTWARE_VERSION。
3. gizwits_protocol.c
   该文件为 SDK API 接口函数定义文件。
4. gizwits_protocol.h
   该文件为 gizwits_protocol.c 对应头文件，相关 API 的接口声明均在此文件中。
5. 其他文件
   a) User/main.c
   MCU 程序入口函数所在文件，入口函数为 main(void)。

2.2 API 介绍

• void gizwitsInit(void)

gizwits 协议初始化接口。

用户调用该接口可以完成 Gizwits 协议相关初始化（包括协议相关定时器、串口的初始化）。

• void gizwitsSetMode(uint8_t mode)

参数 mode[in]：仅支持 0,1 和 2,其他数据无效。

参数为 0，恢复模组出厂配置接口，调用会清空所有配置参数，恢复到出厂默认配置。

参数为 1 或 2，配置模式切换接口，支持 SoftAP 和 AirLink 模式。参数为 1 时配置模组进入 SoftAp 模式，参数为 2 配置模组进入 AirLink 模式。

• void gizwitsHandle(dataPoint_t *dataPoint)

参数 dataPoint[in]:用户设备数据点。
该函数中完成了相应协议数据的处理即数据上报的等相关操作。

- **int8_t gizwitsEventProcess(eventInfo_t *info, uint8_t *data, uint32_t len)**

参数 info[in]:事件队列
参数 data[in]:数据
参数 len [in]:数据长度
用户数据处理函数,包括 wifi 状态更新事件和控制事件。

a) Wifi 状态更新事件

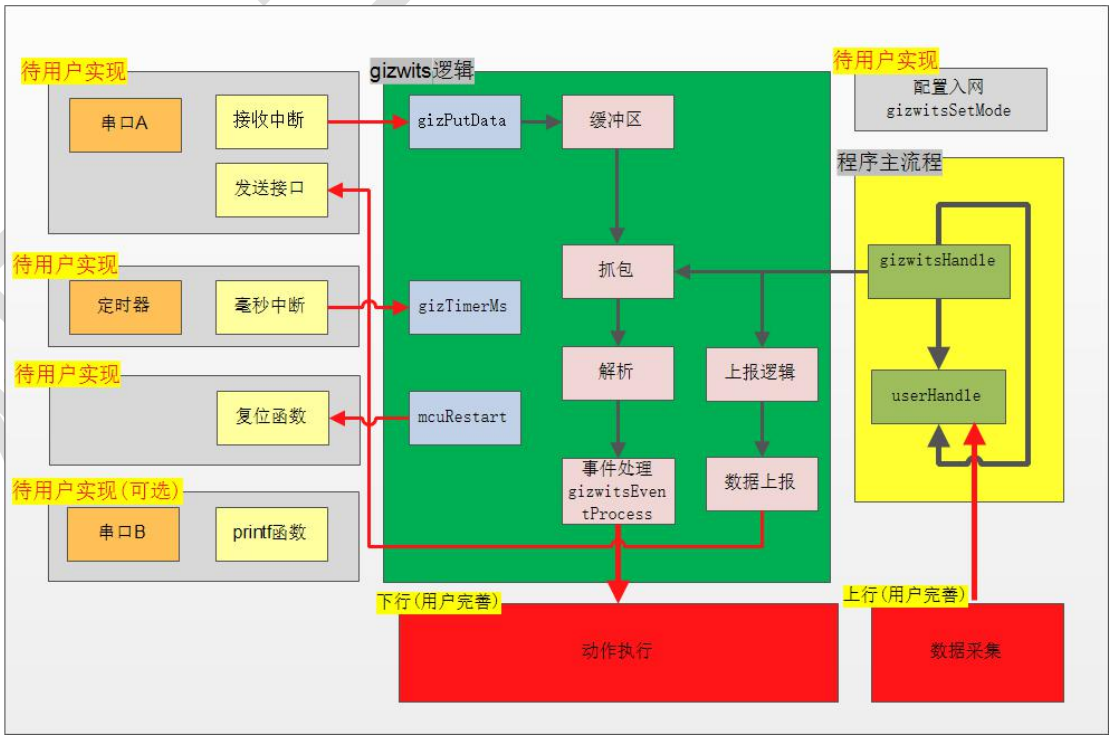
WIFI_开头的事件为 wifi 状态更新事件，data 参数仅在 WIFI_RSSI 有效，data 值为 RSSI 值,数据类型为 uint8_t，取值范围 0~7。

b) 控制事件

与数据点相关,本版本代码会打印相关事件信息，相关数值也一并打印输出，用户只需要做命令的具体执行即可。

2.3 代码结构说明

自动化代码生成工具已经根据用户定义的产品数据点信息，生成了对应的机智云串口协议层代码，用户需要移植代码到自己的工程中，完成设备的接入工作。如图如下：



代码绿色部分的协议逻辑和程序主流程已经帮用户实现，图中用黄色字体注标的部分待

用户实现并完成代码的移植。用户的移植分以下几步进行：

a.搭建最小平台工程（必要）。

b.实现串口驱动（必要）：包括通信与打印功能。

c.实现定时器驱动（必要）。

d.实现芯片复位函数（可选）。

e.实现应用层逻辑开发（必要）：包括数据上下行、入网配置等。

3. 移植步骤介绍

下面我们以 **MSP430 平台的移植**为例来介绍移植步骤。

3.1 搭建最小平台工程（必要）

首先完成目标平台的最小工程搭建，以 MSP430 为例，我们将通信协议处理的源码文件(Gizwits 目录下所有文件)导入到工程中，并将 **User 目录**下的示例 **main.c** 文件整合到工程中的主文件中，如下所示：

```
/** 用户区当前设备状态结构体*/
dataPoint_t currentDataPoint;

void Sys_Init(void)
{
    //关软件看门狗 Stop WDT
    WDTCTL = WDTPW + WDTHOLD;

    //设置时钟
    InitClock();
}

/************************************************************
** 函数名称: main(void)
** 函数功能: 主函数
************************************************************/
void main(void)
{
    //System space init
    Sys_Init();

    //Gizwits protocol init
    userInit();
    gizwitsInit();

    while(1)
    {
        userHandle();
        gizwitsHandle((dataPoint_t *)&currentDataPoint);
    }
}
```

3.2 实现串口驱动（必要）

MCU 方案需要用户实现一个串口，用于设备 MCU 与 WIFI 模组之间数据通信。用户首先需要实现串口接收中断服务函数接口 UART_IRQ_FUN（MSP430 平台函数接口为：**USCI0RX_ISR**），该接口调用 **gizPutData()**函数实现串口数据的接收并且写入协议层数据缓冲区。

下面以 MSP430 平台为例，本例使用 **USCI0** 与模组通信，串口初始化实现如下：

```
void Sys_Init(void)
{
    //关软件看门狗 Stop WDT
    WDTCTL = WDTPW + WDTHOLD;

    //设置时钟
    InitClock();

    //串口设置-9600bps
    serial_init(9600);
    cio_printf(" Start system \n");
}
```

中断服务函数和串口发送报文函数实现如下：

```
/*************************************************************
******
** 函数名称: void USCI0RX_ISR(void)
** 函数功能: Echo back RXed character, confirm TX buffer is ready
first
** 入口参数: 无
** 出口参数: 无
*************************************************************
******/
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
  while (!(IFG2&UCA0TXIFG));                // USCI_A0 TX buffer
ready?

  gizPutData((uint8_t *)&UCA0RXBUF,1);

  return;
}
```

另外，用户需要实现串口的发送接口，uartWrite()函数调用该接口实现设备数据的发送。需要特别注意的是 **gizwits_product.c** 文件中 uartWrite()函数是伪函数，用户需根据自己实现的串口发送接口完善 ***uartWrite()***，请注意相关注释信息，以防出错。实现如下：

```
/**
* @brief 串口写操作，发送数据到 WiFi 模组
*
* @param buf      : 数据地址
* @param len       : 数据长度
*
* @return : 正确返回有效数据长度;-1，错误返回
*/
int32_t uartWrite(uint8_t *buf, uint32_t len)
{
    uint32_t i = 0;

    if(NULL == buf)
    {
        return -1;
    }

    for(i=0; i<len; i++)
    {
       serial_send_blocking(buf[i]);

       //实现串口发送函数,将 buf[i]发送到模组
        if(i >=2 && buf[i] == 0xFF)
        {
          //实现串口发送函数,将 0x55 发送到模组
          serial_send_blocking(0x55);
        }
    }

    return len;
}
```

注：注意示例中的 0x55 条件处理，即出现 0xFF 的数据时后面要加 0x55，这个操作一定要保留。

如果用户需要打印日志调试信息，用户需实现 GIZWITS_LOG 函数，只需修改 **gizwits_protocol.h** 中对应的宏定义即可，如下：

```
#define GIZWITS_LOG cio_printf  ///<运行日志打印
```

3.3 实现定时器驱动（必要）

协议层使用到了一个系统时间，该事件单位为毫秒，所以要求用户实现一个毫秒定时器（必须是 1ms 精确定时，若不准确，会影响到超时重发、定时上报等处理），并且实现中断服务函数 TIMER_IRQ_FUN（MSP430 平台函数接口为：**Timer_A**），该函数调用**gizTimerMs()**实现协议层系统时间的维护。

下面以 MSP430 平台为例，本例使用 Timer_A 实现时间维护，定时器初始化如下：

```
void Sys_Init(void)
{
    //关软件看门狗 Stop WDT
    WDTCTL = WDTPW + WDTHOLD;

    //设置时钟
    InitClock();

    //定时器
    BCSCTL3 |= LFXT1S_2; // Set LFXT1 为 vol 时钟即 12kHZ
    CCTL0|= CCIE;  //设置捕获/比较控制寄存器，CCIE=0x0010，使能捕获比较中断
    CCR0 =12;  //设置捕获/比较寄存器，初始值为 12000，对于 ACLK 时钟频率为 12khz 的频率，相当于 1s 120 相当于 1ms
    TA0CTL = TASSEL_1 +TACLR+MC_1; // 设置定时器 A 控制寄存器，

    //串口设置-9600bps
    serial_init(9600);
    cio_printf(" Start system \n");
}
```

中断服务函数实现如下：

```
#pragma vector=TIMER0_A0_VECTOR//固定的格式
__interrupt void Timer_A (void) //定时器 A 的 CC0 中断处理程序 必须是没有返回值的
{
    gizTimerMs();
}
```

3.4 实现芯片复位（可选）

根据串口协议文档规定，模组可以发送命令复位设备 MCU，所以用户需要实现 **gizwits_product.c** 中的 **mcuRestart()**接口即可。下面以 MSP430 平台为例，实现如下：

```
/**
* @brief MCU 复位函数

* @param none
* @return none
*/
void mcuRestart(void)
{
    ((void (*)())0xFFFE)();
}
```

至此便完成了平台的移植，后续的配置入网、上下行操作属于应用逻辑开发。

3.5 应用逻辑开发

3.5.1 数据下行控制

数据点方式将转换成数据点事件，开发者只需要在 **gizwits_product.c** 文件的 **gizwitsEventProcess()**相应事件下作具体处理即可。

下面使用 MSP430 平台，以 APP 实现控制 LED 为例，如下：

```
/**
* @brief 事件处理接口

* 说明：

* 1.用户可以对 WiFi 模组状态的变化进行自定义的处理

* 2.用户可以在该函数内添加数据点事件处理逻辑，如调用相关硬件外设的操作接口

* @param[in] info : 事件队列
* @param[in] data : 协议数据
* @param[in] len : 协议数据长度
* @return NULL
* @ref gizwits_protocol.h
*/
int8_t gizwitsEventProcess(eventInfo_t *info, uint8_t *data,
uint32_t len)
{
```

```
  uint8_t i = 0;
  dataPoint_t *dataPointPtr = (dataPoint_t *)data;
  moduleStatusInfo_t *wifiData = (moduleStatusInfo_t *)data;

  if((NULL == info) || (NULL == data))
  {
    return -1;
  }

  for(i=0; i<info->num; i++)
  {
    switch(info->event[i])
    {
      case EVENT_LED_ONOFF:
      currentDataPoint.valueLED_ONOFF =
dataPointPtr->valueLED_ONOFF;

      if(0x01 == currentDataPoint.valueLED_ONOFF)
      {
        //user handle
         P1OUT |= BIT6;
      }
      else
      {
        //user handle
         P1OUT &= ~BIT6;
      }
      break;
```

3.5.2 数据上行控制

该工程代码默认在 **userHandle()**中实现传感器数据采集，并且该函数在 while 中循环执行，原则上用户只需要关心如何采集数据。特别提醒，默认 while 循环执行速度较快，需要针对不同的需求，用户可调整数据点数据的采集周期和接口实现位置，预防由于传感器数据采集过快引发的不必要的问题（具体可查看 STM32 的详解篇）。

下面使用 MSP430 平台，以灯的状态赋值为例（**只读型数据点**的操作会被云端自动生成），如下：

```
/**
* 用户数据获取

* 此处需要用户实现除可写数据点之外所有传感器数据的采集,可自行定义采集
```

```
频率和设计数据过滤算法
* @param none
* @return none
*/
void userHandle(void)
{
    currentDataPoint.valueLED_ONOFF = 0x01;
}
```

3.5.3 配置入网功能（必要）

根据串口协议文档规定，MCU 可以向模组发送命令使其进入相应的配置模式，所以用户可以调用 **gizwitsSetMode** 接口（在 **gizwits_protocol.c** 中）完成相应的操作（例如按键控制），接口调用说明如下：

```
/**
* @brief WiFi 配置接口

* 用户可以调用该接口使 WiFi 模组进入相应的配置模式或者复位模组

* @param[in] mode 配置模式选择：0x0， 模组复位 ;0x01， SoftAp 模式 ;0x02， AirLink 模式
* @return 错误命令码
*/
int32_t gizwitsSetMode(uint8_t mode)
```

3.5.4 实现模组状态处理功能（可选）

开发者可以在 **gizwits_product.c** 文件的 **gizwitsEventProcess()**函数内获得 WIFI 状态，并做相应的逻辑处理。

下面使用 MSP430 平台为例，添加一个逻辑：当 WiFi 模块成功连接路由后关闭 LED 灯，代码中示例如下：

```
    case WIFI_CON_ROUTER:
        P1OUT &= ~BIT6;
        break;
```

4. 相关支持

1) 如果您是开发者

GoKit 是面向智能硬件开发者限量免费开放，注册我们的论坛或关注我们的官方微信均可发起申请即可。

开发者论坛： http://club.gizwits.com/forum.php

文档中心：http://docs.gizwits.com/hc/

2) 如果您是团体

GizWits 针对团体有很多支持计划，您可以和 GizWtis 联系，快速得到 GoKit 以及技术支持；

网站地址：http://www.gizwits.com/about-us

官方二维码：