GU620_OPEN_AT DevKit
V1.0

前言

合方圆OPEN AT是本公司自主研发的开发包，用户可以根据自身实际需要结合本公司的开发工具包开发上层应用。

产品版本

与本文档相对应的产品版本如下所示。

产品名称	产品版本
GU620	

读者对象

本文档(远程配置需求)主要适用于以下工程师：

- 技术研发工程师
- 技术支持工程师
- 维护工程师

1.功能概述

OPEN AT 开发工具包可以使用户直接开发应用嵌入到 GU620 的通讯软件里。因而，用户可以创建更低成本，更高效率的系统。通过系统简单而有效的开发接口，使得你在将来持续的硬件升级也无需做太多软件改动而直接发布新产品。

GU620 OPEN AT 具有以下特点：

- **开发高效**
  - 扩展 API 提供了几乎所有平台功能
  - 开发工具包简单紧凑，十分易于上手
  - 编译过程简单有效，用户仅仅通过增加代码文件以及修改一个配置即可编译
  - OPEN AT 是降低硬件成本以及开发周期最佳方案
- **灵活简便**
  - 用户可以灵活定制、添加自定义 AT 命令
  - 用户可以接收来自短信息、TCP/IP 的请求，访问系统的 I/O、ADC 等资源
  - 可以作为数据采集系统使用
- **更加稳定可靠**
  - OPEN AT 运行在独立的用户空间，通过任务消息和系统通讯
  - 健壮稳定的软件结构，保证系统具有极高可扩展性以及稳定性
  - 根据用户不同需求，可对系统硬件资源作灵活定制
- **平台特性**
  - ARM7 硬件架构，主频 360MHZ
  - 提供高达 300KB 用户数据配置，提供超大栈缓存（32KB）等
  - 提供 800K Bytes 代码空间
  - 支持 1 个 10 位 ADC 输入
  - 1 个 UART 端口
  - 5 个硬件中断
  - SPI LCD 接口
  - 24 个以上可编程 GPIO
  - 超低功耗，待机 < 1 mA
- **系统 API 接口**

最新的发布版本支持以下接口：

- 音频 API
  键盘 DTMF 按键音，其它频率声音
  内部提供 10 首和弦音乐以及其它音源

- 消息 API
  AT 命令执行结果消息
  键盘事件
  中断事件
  串口数据事件
  定时器消息

- Flash API
  Flash 管理函数
  同步读写
  全面的错误码

- 系统 API
  系统相关函数，包括模块关机、读取系统 tick、OPEN AT 模式切换等
  进入休眠、模块唤醒，实现超低功耗
  防软件拷贝技术，防止非法盗版

- 外围设备 API
  读取 ADC
  SPI LCD
  GPIO 管理
  中断

- 定时器 API
  开始、停止定时器

- 调试 API
  串口格式化打印调试信息

- **安装环境**

  - 仅支持 WIN XP 操作系统
  - 编译环境为 RVCT 3.1 Build 569
  - ActivePerl-5.8.9.829 以后版本

2. SDK

2.1 GU620 发布规则：

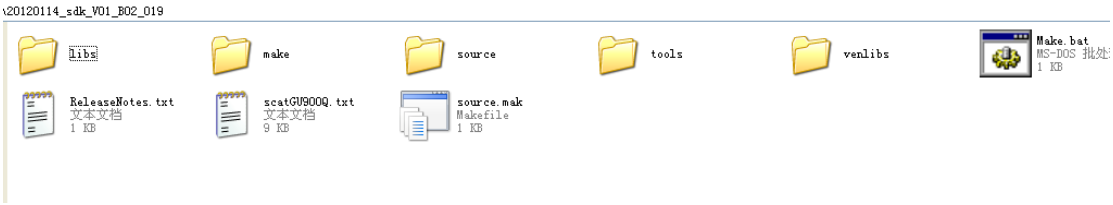
我们目前有多个软件版本：B01,B02，每次模块启动的时候会打印一个软件版本号，比如，V01_B01_015，其中的 V01 是数据库的版本，每次我们的数据参数配置有变化，这个 015 会递增 1，B01 是硬件版本分支，目前有以下分支版本：

B01 — 是普通用户版本，可带硬件流控。

模块启动会在串口输出“V01_B01_015-OPEN-01, Build Time:2014/05/15 22:12”，Build Time 是我们的编译号，每次软件有更新发布，都会按照当时的时间保存到 SDK 里。**奇数编译号是发布版本号，偶数版本是内部测试版本。**

注意：硬件版本分支以及编译号会经常更新。

把我司提供的压缩文件解压到一目录，如图



3. 编译

3.1 编译 DEMO

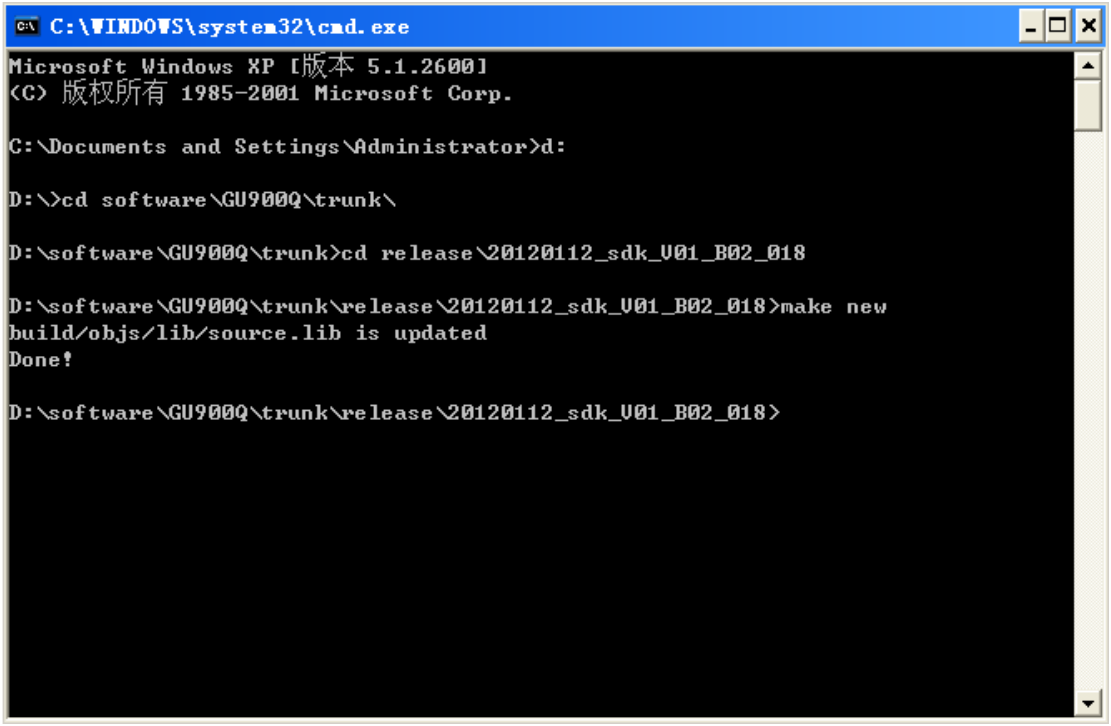
首先，解开 SDK 代码后，根据你的安装路径看看 SDK 根目录下的 make.bat 文件是否需要更改，如下：

```
@SET ARMHOME=C:\Progra~1\ARM
@SET ARMROOT=C:\Progra~1\ARM
@SET RVCT31BIN=%ARMROOT%\RVCT\Programs\3.1\569\win_32-pentium
@SET RVCT31INC=%ARMROOT%\RVCT\Data\3.1\569\include\windows
@SET RVCT31LIB=%ARMROOT%\RVCT\Data\3.1\569\lib
@SET LM_LICENSE_FILE=%ARMROOT%\Licenses\rvds.dat
@SET
PATH=C:\Perl\bin\;%CD%\tools\MinGW\bin;%CD%\tools\MinGW\lib\gcc-lib\mingw32\3.3.1;C:
\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Program Files\Microsoft
Visual Studio\COMMON\Tools;C:\Program Files\Microsoft Visual
```

```
Studio\VC98\Bin;%ARMROOT%\RVCT\Programs\3.1\569\win_32-pentium

tools\make.exe -fmake\main.mak -k -r -R %*
rem perl make2.pl %*
```

进入命令行，cd 到 SDK 安装目录，执行 make new，如图

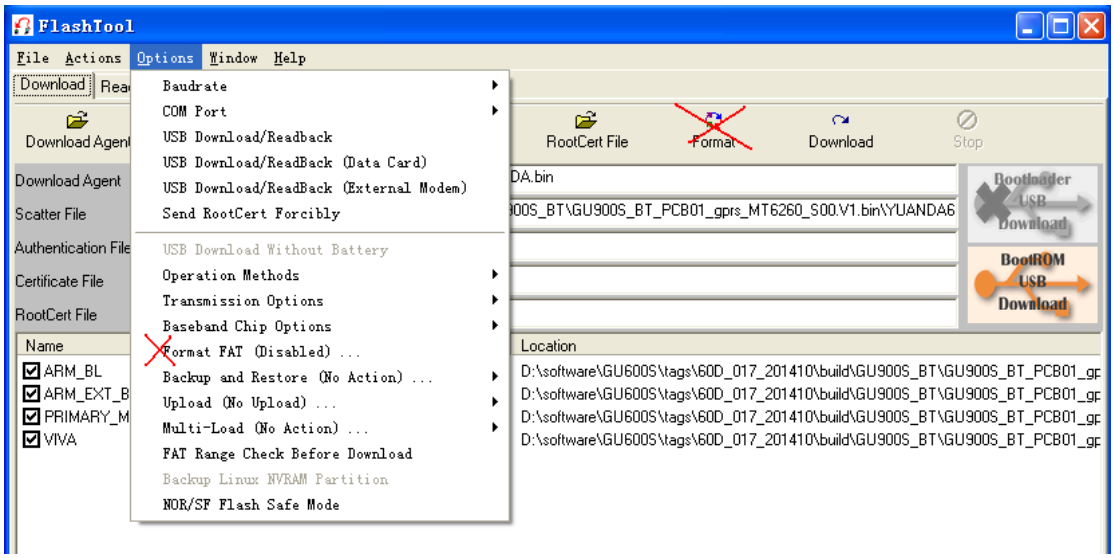


成功后，将生成 GU620.bin 文件。

如果发生编译错误，请看 build/source/的日志文件。
如果发生链接错误，请看 build/ link.log 文件。

3.2 烧录软件

模块在发布的时候，已经会先预先烧写一个普通版本的软件，模块内部有很多重要参数是不能被用户改写的。因此，用户不能对模块的 Flash 进行格式化操作，否则，需要退回修理！



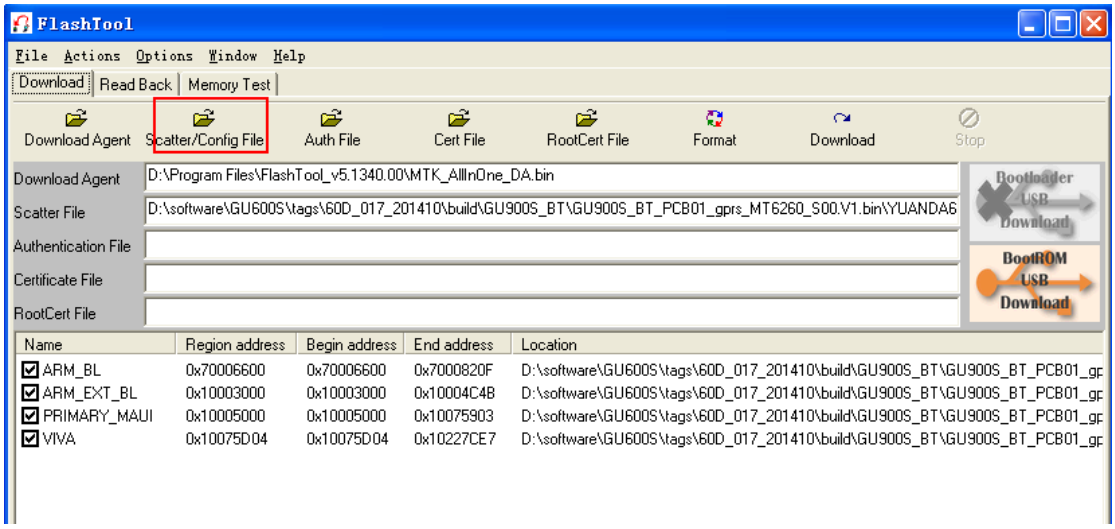
烧录过程：

1） 准备好模块，先处于断电状态，连接 TTL 转 USB 或者串口到 PC 机

2） 启动 Flash Tool 程序，检查各项设置（COM Port, Baudrate 等）。

COM Port：选择模块的第一个数据口（即 AT 指令命令接收口）。
如果是串口，选择 115200
如果是 TTL 转 USB，选择 921600

3） 在存放 BIN 文件的目录选择 scatGU620.txt 文件，准备下载



4） 点击“Download”，然后模块上电

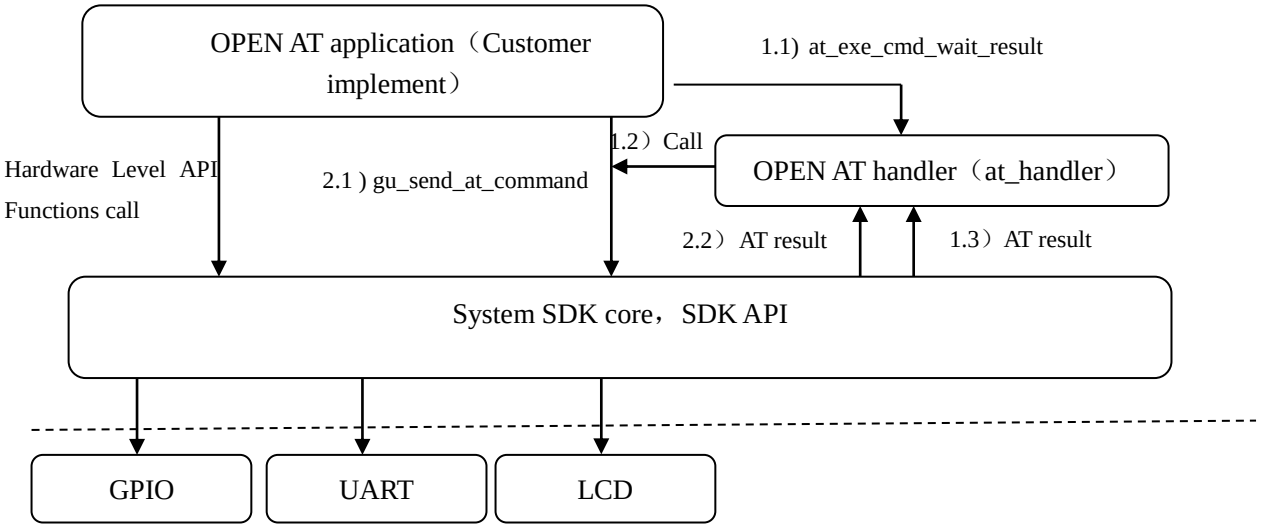
5） 最后会弹出一个“OK”的窗口，表示下载已经完成，您可以运行新软件了。

4. 软件实现框架

HFY OPEN AT 实现从整体上看可分为三层：

- System SDK core ， SDK API
- OPEN AT handler(OPEN AT handler, at_handler)
- OPEN AT application (Customer implement)

SDK 不仅仅向用户提供了完整的 AT 指令处理，并且提供了一些 API 供用户直接和底层通讯，比如，GPIO 编程配置，LCD 接口，串口服务等。



其中，OPEN AT 应用和 OPEN AT 处理框架是完全开放给用户，由用户完成自己的编程工作，用户可以参考 SDK 的示例框架代码开发。

5. 应用开发

5.1 增加新代码

开发者直接拷贝新代码到 source 目录下，通过编辑 source.mak 来增加新代码文件、包含头文件路径、宏定义等，最后所有代码会打包为 source.lib，和 GU620 CORE 进行链接。

如果发生编译错误，请看 build/source/的相应的编译日志。
如果发生链接错误，请看 build/ link.log 文件。

5.2 GPIO 功能复用

下表为 GU620 模块支持的 GPIO 功能复用表，其中带土黄色背景的为默认配置，如果用户需要在实际应用中更改，可以通过调用 gu_gpio_mode_setup 函数重新配置。

Pin No	Program Var	Reset	Mode0	Mode1	Mode2	Mode3	Mode4	Mode5
10	GU_PIN_GPIO1	PD	GPIO					
11	GU_PIN_GPIO2	PU	GPIO					
12	GU_PIN_GPIO3	PU	GPIO					
13	GU_PIN_GPIO4	PD	GPIO	EINT1				
14	GU_PIN_CON_OK	PD	GPIO	EINT2				
15	GU_PIN_DTR	PU	GPIO	EINT3				
16	GU_PIN_RI	PD	GPIO					
17	GU_PIN_NET_LIGHT	PU	GPIO					
20	GU_PIN_KBR3	PD	GPIO		EINT4			
21	GU_PIN_KBC3	PD	GPIO		EINT5			

5.21 GPIO 函数

```
/*****************************************************************************
* FUNCTION
*      gu_gpio_mode_setup()
* DESCRIPTION
*      config GPIO function mode.
* PARAMETERS
*      port[in]          -    GPIO port number
*      mode[in]      -    GPIO function mode.
*
* RETURNS
*      None
* EXAMPLE:
*****************************************************************************/
extern void gu_gpio_mode_setup(gu_pin_enum port, gu_io_mode_enum mode);


/*****************************************************************************
* FUNCTION
*      gu_set_io_direct()
* DESCRIPTION
*      Set IO directly. In gernally, you don't need to config the direction of GPIO since GU620
*          has configured it for you.
* PARAMETERS
*      direction[in]    -      GPIO direction, INPUT = 0, OUTPUT = 1.
*          port[in]              -      GPIO port number
*
* RETURNS
*      None
* EXAMPLE:
```

```
*****************************************************************************/
extern void gu_set_io_direct(gu_char direction, gu_pin_enum port);

/*****************************************************************************
* FUNCTION
*     gu_read_io_val()
* DESCRIPTION
*     read GPIO value.
* PARAMETERS
*     port[in] - GPIO port number.
*
* RETURNS
*     None
* EXAMPLE:
*****************************************************************************/
extern gu_char gu_read_io_val(gu_pin_enum port);

/*****************************************************************************
* FUNCTION
*     gu_write_io_val()
* DESCRIPTION
*     write GPIO value.
* PARAMETERS
*     data[in]   -   Expected data in byte, 1 or 0
*     port[in]    -   GPIO port number.
*
* RETURNS
*     None
* EXAMPLE:
*****************************************************************************/
extern void gu_write_io_val(gu_char data, gu_pin_enum port);
```

例子：

配置 GU_PIN_CTS 管脚为模式 5，使能中断 EINT2：
gu_gpio_mode_setup(GU_PIN_CTS, GU_IO_MODE_5);

程序调用后，在 GU_PIN_CTS 管脚发生电平变化后，系统底层会返回一个 GU_MSG_EINT_EVT 的消息，其对应的数据结构为：gu_eint_evt_struct

5.2 LCD 编程接口

GU620 模块提供了一个 SPI 的硬件接口，如果需要使用 GU620 模块现有的 SPI 接口，只需要接上规格书定义的管脚即可，但要注意的是这些管脚电压是 1.8 V。对于一些需要 2.8 V 电压驱动的 SPI LCD 屏，可以使用 GPIO 模拟 SPI 的做法，模块已经提供了 GPIO 模拟 SPI 的驱动函数。该 GPIO 模拟 SPI 驱动对应的管脚定义如下：

Pin	Program Var	LCD PIN

No		
2	GU_PIN_GPIO0	LCD_A0
6	GU_PIN_DSR	LCD_RST
51	GU_PIN_GPIO7	LCD_CLK
52	GU_PIN_GPIO1	LCD_DATA
68	GU_PIN_ GPIO9	LCD_CS

编程示例：

```
// 初始化 LCD
gu_lcd_init(0x00);
static void LCD_display_fb(gu_uint8 *fb)
{
    gu_uint8 Pages,Columns;
    gu_uint16 k=0;

    gu_lcd_send_cmd(0x40);                          //display start line set *?*
    for (Pages=0; Pages<=7; Pages++)
    {
        gu_lcd_send_cmd(0xb0 + Pages);              //page address set
        gu_lcd_send_cmd(0x10);
        gu_lcd_send_cmd(0x00);                      //set column address

        for (Columns=0; Columns < 128; Columns++)
        {
            gu_lcd_send_data(fb[k++]);
        }
    }
}
```

5.3 时间相关函数

5.3.1 获取系统时间

```
/*****************************************************************************
* FUNCTION
*    gu_get_tick_count()
* DESCRIPTION
*    Get system ticks after booting up.
* PARAMETERS
*    None.
*
* RETURNS
*    system ticks after booting up
* EXAMPLE:
*****************************************************************************/
extern gu_uint32 gu_get_tick_count();

/*****************************************************************************
* FUNCTION
*    gu_get_systicks_ms()
```

```
* DESCRIPTION
*      Get system milliseconds after booting up.
* PARAMETERS
*      None.
*
* RETURNS
*      system milliseconds after booting up
* EXAMPLE:
*****************************************************************************/
extern gu_uint32 gu_get_systicks_ms();

/*****************************************************************************
* FUNCTION
*      gu_get_systicks_s()
* DESCRIPTION
*      Get system seconds after booting up.
* PARAMETERS
*      None.
*
* RETURNS
*      system seconds after booting up
* EXAMPLE:
*****************************************************************************/
extern gu_uint32 gu_get_systicks_s();
```

5.3.2 系统定时器

GU620 SDK 带有两种定时器编程模式供用户使用，一种是可回调函数，另一种是消息队列返回。
第一种方式可提供最多 16 个定时器，第二种方式只有一个。函数定义如下：

```
/*****************************************************************************
* FUNCTION
*      gu_start_timer()
* DESCRIPTION
*      Start a timer, when timer is timeout, it shall generate a timer expire message in the message queue.
* PARAMETERS
*          timer[in]                    - 0 <= timer id < GU_MAX_TIMERS(16)
*      func[in]                -    call this func if time is expire.
*      func_arg[in]             -    function arguments attached to the callback func.
*      timeout_ms[in]        -    timer timeout setting..
*
* RETURNS
*      None
* EXAMPLE:
*****************************************************************************/
extern void gu_start_timer(gu_int32 timer, gu_timer_func func, void *func_arg, gu_uint32 timeout_ms);

/*****************************************************************************
* FUNCTION
*      gu_stop_timer()
```

```
* DESCRIPTION
*      Stop the running/starting timer before it is timeout.
* PARAMETERS
*      timer[in]                  - timer id < GU_MAX_TIMERS(16)
*
* RETURNS
*      None
* EXAMPLE:
*****************************************************************************/
extern void gu_stop_timer(gu_int32 timer);


/*****************************************************************************
* FUNCTION
*      gu_app_start_timer()
* DESCRIPTION
*      Start a timer, when timer is timeout, it shall generate a timer expire message in the message
queue.
* PARAMETERS
*      timeout_ms[in]      -     timer timeout setting.
*
* RETURNS
*      None
* EXAMPLE:
*****************************************************************************/
extern void gu_app_start_timer(gu_uint32 timeout_ms);

/*****************************************************************************
* FUNCTION
*      gu_app_stop_timer()
* DESCRIPTION
*      Stop the running/starting timer before it is timeout.
* PARAMETERS
*      None.
*
* RETURNS
*      None
* EXAMPLE:
*****************************************************************************/
extern void gu_app_stop_timer();

/*****************************************************************************
* FUNCTION
*      gu_app_sleep_ms()
* DESCRIPTION
*      Let this task sleep x ms.
* PARAMETERS
*      sleep_ms[in]      -     sleep ms.
*
* RETURNS
*      None
* EXAMPLE:
*****************************************************************************/
extern void gu_app_sleep_ms(gu_uint32 sleep_ms);
```

5.4 声音播放函数

系统内置 10 首和弦铃声以及各种双频音，如下函数：

```
/*****************************************************************************
* FUNCTION
*     gu_app_play_audio()
* DESCRIPTION
*     send data to lcd driver
* PARAMETERS
*     audio_id[in]        - audio id
*     audio_style[in]     - audio play style
*     audio_dev[in]       - audio device path, refers to audio_dev macro
*
* RETURNS
*     None
* EXAMPLE:
*****************************************************************************/
extern gu_bool gu_app_play_audio(gu_audio_id_enum audio_id,
      gu_audio_play_style_enum audio_style, gu_uint8 audio_dev);

/*****************************************************************************
* FUNCTION
*     gu_app_stop_audio()
* DESCRIPTION
*     send data to lcd driver
* PARAMETERS
*     audio_id[in]        - audio id
*
* RETURNS
*     None
* EXAMPLE:
*****************************************************************************/
extern gu_bool gu_app_stop_audio(gu_uint16 audio_id);
```

如果需要把声音文件编译到程序里，直接在程序调用来播放，可以先使用工具将声音资源文件转换成 C 文件，在 C 文件里可以看到有该资源文件的数组定义，拷贝这个数组定义到工程文件作为全局变量来使用，即可在特定的函数调用。举例：

1. 在 DOS 命令行使用工具 

(GU620_resgen.exe)将声音资源文件转换成 C 文件；

   GU620_resgen.exe -h -a Effect01.wav Effect01.c

2. 在 C 文件里可以看到有该资源文件的数组定义:

   __align(4) const gu_uint8 audio_13[]

3. 使用播放声音函数调用，如果需要知道播放结束或者播放状况，可以增加一个回调函数来获取具体情况。

```
// 声音播放回调函数定义，如果该函数被回调，说明播放已经结束
void audio_play_callback(gu_int32 result)
{
        if (result == -9006) // the user stop play
        {
            return;
        }
        if (result == -9005)
        {
            // MDI_AUDIO_SUCCESS;
        }
}

// 外部函数定义
extern  gu_int32  mdi_audio_play_string_with_vol_path(
          void          *audio_data,
          gu_uint32     len,
          gu_uint8      format,
          gu_uint8      play_style,
          void          *handle_p,
          void          *play_callback,
          gu_uint8      volume,
          gu_uint8      path);

//////////////////////////// 调用样例代码段//////////////////////////////////////////
const gu_uint8 *audio_data;
gu_uint8 audio_type;
gu_uint32   audio_len;

audio_data = audio_13;
if (audio_data[0] == '\0')
{
    return GU_FALSE;
}

audio_type = audio_data[0];
audio_len   = 0;
audio_len |= (U32) (audio_data[1]);
audio_len |= (U32) ((U32) audio_data[2] << 8);
audio_len |= (U32) ((U32) audio_data[3] << 16);
audio_len |= (U32) ((U32) audio_data[4] << 24);

return mdi_audio_play_string_with_vol_path(
        (void*)&audio_data[8],
        audio_len,
        audio_type,
        GU_AUDIO_PLAY_ONCE,
        NULL,
        audio_play_callback,
        4,                  /* 0 ~ 6*/
        GU_TRUE);
```

5.5 Flash 存取函数

FLASH 的空间是通过内部的一个 ID 值来访问（ID < 256），用户可以根据来命名 ID 值，具体的函数如下：

```
/*****************************************************************************
* FUNCTION
*     gu_flash_alloc()
* DESCRIPTION
*     User can create a new or exist space for the user data, if the flash id had been created previously, it
*         tries to rewrite the existed flash space.
* PARAMETERS
*     id[in]    -    flash id to be created
*     max_size[in]    -    the max size of data will be allocated
*
* RETURNS
*     result code - refer to [flash access error codes]
* EXAMPLE:
*****************************************************************************/
extern gu_int16 gu_flash_alloc(gu_uint8 id, gu_uint32 max_size);

/*****************************************************************************
* FUNCTION
*     gu_flash_read()
* DESCRIPTION
*     .
* PARAMETERS
*     id[in]    -    flash id
*     offset[in]    -    offset
*     data_buf[in, out]    -    data buf to be read
*     data_buf_len[in]    -    data buf length
*
* RETURNS
*     RTF_NO_ERROR - Succeed
*     Others - Refer to [flash access error codes]
* EXAMPLE:
*****************************************************************************/
extern gu_int16 gu_flash_read(gu_uint8 id, gu_uint32 offset,
        gu_uint8 *data_buf, gu_uint16 data_buf_len);

/*****************************************************************************
* FUNCTION
*     gu_flash_write()
* DESCRIPTION
*     .
* PARAMETERS
*     id[in]    -    flash id
*     offset[in]    -    offset
*     data_buf[in]    -    data buf to be written
*     data_buf_len[in]    -    data buf length
*
* RETURNS
*     RTF_NO_ERROR - Succeed
```

```
*      Others - Refer to [flash access error codes]
* EXAMPLE:
*****************************************************************************/
extern gu_int16 gu_flash_write(gu_uint8 id, gu_uint32 offset,
      gu_uint8 *data_buf, gu_uint16 data_buf_len);
```

5.6 AT 指令编程接口

在 SDK 里发送 AT 指令或者数据，类似用一个 MCU 向 GU620 模块发送 AT 指令。可以用同步方式函数（at_exe_cmd_wait_result）发送，也可以用异步方式函数（gu_send_at_command）发送。一般来说，同步方式发送 AT 指令是比较符合程序逻辑的做法，这种方法在编写软件的时候可维护性较强，但有些场合（发送通讯数据）下，需要用异步函数来发送，因为它并不关心返回的结果。

5.7 编程实例

我们发布的 SDK 已经带有丰富的编程实例，用户可以结合 SDK 的 API 来理解各个应用的编程细节。

1） 用户需要实现应用入口函数 gu_main_task:

```
void gu_main_task()
{
      gu_message_req_struct *msg_req;

      while (1) {
            if ((msg_req = gu_get_message())) {

                  // Process the current message
                  .....

                  // Free the memory of the message
                  gu_free_message(msg_req);
            }
      }
}
```

2） 系统采用消息驱动机制，所有的异步数据返回、硬件事件和系统事件都是通过消息队列返回，用户在处理这些消息时，不能阻塞整个队列。因此，用户在编程时，处理完消息后，就要马上处理下一个消息。这个和 WINDOWS 的消息机制有些类似。