集群板程序指南

surenyi82@163.com

2018 年 7 月 23 日

修订记录

修订日期	版本	作者	修订说明
2018 / 07 / 22	初稿	苏仁义	开始撰写该文档。

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

目 录

1 简介 4

2 代码编译 4

3 对讲模块协议实现 4

3.1 实现文件 4
3.2 关键数据结构 5
3.3 代码示例 6
3.4 高层 API 介绍 8
3.4.1 discip_open 8
3.4.2 discip_t discip_open_with_fd 8
3.4.3 discip_close 8
3.4.4 discip_getfd 8
3.4.5 discip_get_tmofd 9
3.4.6 discip_ingress 9
3.4.7 discip_timeout 9
3.4.8 discip_set_timeout 9
3.4.9 discip_set_reqhandler 10
3.4.10 discip_flush 10
3.4.11 discip_statics 10
3.4.12 discip_clean_cache 10
3.4.13 discip_set_channel 11
3.4.14 discip_set_freq 11
3.4.15 discip_set_power 11
3.5 低层 API 12
3.5.1 srb_alloc 12
3.5.2 srb_free 12
3.5.3 srb_set_completion 12
3.5.4 srb_set_timeout 13
3.5.5 srb_reset 13

目 录

3.5.6 srb_submit 13
3.5.7 srb_set_channel 13
3.5.8 srb_set_freq 13
3.5.9 srb_set_power 13
3.5.10 srb_ack 14
3.5.11 srb_get_block 14
3.5.12 示例: 实现信道切换指令 14

4 测试程序 15

5 集群板 `icd` 程序 15

5.1 发布包中的文件 15
5.2 Lua 模块 16
5.2.1 discip 16
5.2.2 udp 17
5.2.3 mcu 17

1 简介

集群板实现通过对讲机与语音终端通信的功能。该设备中一个支持 UHF 和 VHF 的无线模块，通过串口（`/dev/ttymxc1`）与主控 CPU (i.MX 6) 连接。通过一个厂商自定义的协议使用模块功能。其完整的交互协议定义请参阅代码目录下的 "`docs/dmr818.pdf`" 文件。

2 代码编译

解压源码压缩包 `icd-rf-src.tar.gz`，然后执行命令：

```
$> tar xvf icd-rf-src.tar.gz
$> cd icd-rf/src
$> make
$> make pkg
```

执行完上面命令后，在 icd-rf 目录会有一个 `icd-rf-2.1.tar.gz` 的文件。可以通过 x-tool 点击 "升级对讲"，然后选择该文件来升级集群板。

3 对讲模块协议实现

对讲模块协议在 `docs/dmr818.pdf` 文件中有完整定义。目前代码中只具体实现了其中的三个指令: 信道切换 0x01, 设置收发频率 0x0D, 设置发射功率 0x17。代码协议框架是完整的，新增加其他指令是很容易的，通过对照协议文档简单添加几行代码即可实现。

3.1 实现文件

模块通信协议由下面几个文件实现：

- `src/discip.c`
- `src/discip.h`
- `src/dmr.c`

- `src/dmr.h`

这几个文件写的很独立，不依赖别的东西，如果其他项目用到该模块，可以把这几个文件直接加入到新项目来使用。

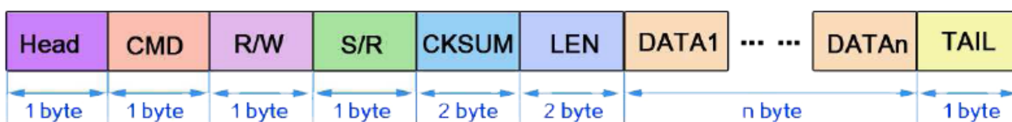
3.2 关键数据结构

设计原则是尽可能隐藏实现细节，让使用者只需要关注接口，目前只会暴露一个数据结构给使用者：

```
src/dmr.h:

#define DMR_BLOCK_SIZE (128)
struct dmr_block {
    uint8_t     cmd;
    uint8_t     rw;
    uint8_t     sr;
    uint16_t    checksum;
    uint16_t    len;
    union {
        uint8_t    data[DMR_BLOCK_SIZE];
        uint8_t    *raw;
    };
};
```

它对应协议中除 “Head” 和 “TAIL” 字段外所有字段：



注意: 结构 `struct dmr_block` 中，如果 `len` > `DMR_BLOCK_SIZE`，数据部分使用 `raw` 字段。如果 `len` ≤ `DMR_BLOCK_SIZE`, 数据部分使用 `data` 字段。没有特别的理由，不要直接使用 `data` 和 `raw`，请使用工具函数：

```
const uint8_t *dmr_block_data(const struct dmr_block *drb, int *len);
```

3.3 代码示例

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <poll.h>

#include "discip.h"

static void __completion(const struct dmr_block *drb, void *ud)
{
    int *done = ud;

    *done = 1;

    if (drb == NULL) {
        fprintf(stderr, "set timeout.\n");
        return;
    }
    if (drb->sr == DMR_ST_SUCC) {
        fprintf(stderr, "set succ.\n");
    } else {
        fprintf(stderr, "set failed.\n");
    }
}

int main(int argc, char *argv[]
{
    struct pollfd pfd[2];
    int tfd, tmfd;
    int n, done = 0;

    discip_t dsc = discip_open("/dev/ttymxc1");
    tfd = discip_getfd(dsc);
    tmfd = discip_get_tmofd(dsc);

    /* set channel to 6 */
    discip_set_channel(dsc, 6, &done, __completion);

    while (!done) {
        memset(&pfd, 0, sizeof pfd);
        pfd[0].fd = tfd;
        pfd[0].events = POLLIN;
        pfd[1].fd = tmfd;
        pfd[1].events = POLLIN;
        n = poll(pfd, 2, -1);
```

```
        if (n <= 0)
            continue;

        if (pfd[0].revents & POLLIN) {
            discip_ingress(dsc);
        }
        if (pfd[1].revents & POLLIN) {
            discip_timeout(dsc);
        }
    }

    done = 0;
    /* set freq to 421.125M */
    discip_set_freq(dsc, 421125000, 421125000,
     &done, __completion);
    while (!done) {
        memset(&pfd, 0, sizeof pfd);
        pfd[0].fd = tfd;
        pfd[0].events = POLLIN;
        pfd[1].fd = tmfd;
        pfd[1].events = POLLIN;
        n = poll(pfd, 2, -1);
        if (n <= 0)
            continue;

        if (pfd[0].revents & POLLIN) {
            discip_ingress(dsc);
        }
        if (pfd[1].revents & POLLIN) {
            discip_timeout(dsc);
        }
    }

    discip_close(dsc);
    return 0;
}
```

代码展示了如何设置信道和频率。在已有的基于 select/poll/epoll 事件循环的程序中，添加少量代码，就可以整合进该功能。

3.4 高层 API 介绍

在应用中首选使用该层接口。它使用相对简单，关注的细节最少。但是能完成的功能有限。

该接口中所有函数不是线程安全的，请在同一线程中使用这些函数，或者自行保证函数调用的原子性。

3.4.1 discip_open

discip_t discip_open(const char *tty);

功能：创建 discip_t , 它保存着通信的状态，后面的函数调用，大多以它为第一个参数。

参数: tty 串口设备名，例如: /dev/ttymxc1。

返回值：创建成功返回 discip_t 对象；创建失败返回 NULL 。

3.4.2 discip_t discip_open_with_fd

discip_t discip_open_with_fd(int fd);

功能：与 discip_open 一样，参数是文件描述符，用户可以自己 open 串口设备，然后用成功的文件描述符来创建 discip_t 对象。

参数：fd 已打开的串口文件描述符。

返回值：创建成功返回 discip_t 对象；创建失败返回 NULL 。

3.4.3 discip_close

void discip_close(discip_t dsc);

功能: 释放 discip_t 占用的资源。

参数：dsc 是 discip_open 创建的对象。

返回值：无。

3.4.4 discip_getfd

int discip_getfd(discip_t dsc);

功能：获取串口设备的文件描述符，以便用于 select/poll/epoll 检测数据事件。

参数: dsc 是 discip_open 创建的对象。

返回值: tty 设备文件描述符，< 0 表示失败。

3.4.5 discip_get_tmofd

int discip_get_tmofd(discip_t dsc);

功能：获取超时文件描述符，以便用于 select/poll/epoll 检测超时事件。

参数: dsc 是 discip_open 创建的对象。

返回值: 超时文件描述符，< 0 表示失败。

3.4.6 discip_ingress

int discip_ingress(discip_t dsc);

功能：处理串口输入时间，用户不能直接调用 read 来读取串口设备数据。在检测到串口文件描述符可读时，必须调用该函数。请参考示例代码。

参数: dsc 是 discip_open 创建的对象。

返回值: 超时文件描述符，< 0 表示失败。

3.4.7 discip_timeout

int discip_timeout(discip_t dsc);

功能: 处理超时事件，如果有正在等待的响应指令，它的回调函数会被调用。用户在侦测到 tmofd 文件描述符可读事件时，需要调用该函数。请参考示例代码。

参数: dsc 是 discip_open 创建的对象。

返回值: 成功返回 0，失败返回 < 0 。

3.4.8 discip_set_timeout

int discip_set_timeout(discip_t dsc, int ms);

功能：设置全局的超时时间。如果指令没有单独设置超时时间，则使用该值。默认为 2 秒。

参数: dsc 是 discip_open 创建的对象；ms 超时时间，单位毫秒。

返回值: 成功返回 0，失败返回 < 0 。

3.4.9 discip_set_reqhandler

void discip_set_reqhandler(discip_t dsc, void *ud, void (*on_request)(void *ud, const struct dmr_block *));

功能：设置回调函数。当从串口数据中解析到正确的协议数据包，且没有指令正在等待响应时，该回调函数会被调用。

参数: dsc 是 discip_open 创建的对象；ud 回调函数的第一个参数；on_request 回调函数。

返回值: 无。

3.4.10 discip_flush

int discip_flush(discip_t dsc);

功能: 刷新 discip_t 中需要处理的指令。该函数几乎不会用到。

参数: dsc 是 discip_open 创建的对象。

返回值: 成功返回 0，失败返回 < 0 。

3.4.11 discip_statics

void discip_statics(discip_t dsc, int *queue, int *free);

功能: 获取 dsc 中等待队列中的指令数和可以利用的指令缓存。用于调试目的。

参数: dsc 是 discip_open 创建的对象；queue 返回等待队列中的指令数；free 返回缓存中的指令数量。

返回值: 成功返回 0，失败返回 < 0 。

3.4.12 discip_clean_cache

void discip_clean_cache(discip_t);

功能: 释放指令缓存，用于调试目的。

参数: dsc 是 discip_open 创建的对象。

返回值: 无。

3.4.13 discip_set_channel

int discip_set_channel(discip_t dsc, int ch, void *ud, void (*cb)(const struct dmr_block *, void *ud));

功能：信道切换。

参数：dsc 是 discip_open 创建的对象；ch 需要切换到的信道号；ud ud 回调函数的最后一个参数; cb 对方响应或者超时的回调函数。

返回值: 成功返回 0，失败返回 < 0 。该返回值并不表示指令执行是否成功。指令的成功需要在回调函数中检测。当超时发生时，回到函数的第一个参数为 NULL；当它不是 NULL 时，需要从 struct dmr_block 的 `sr` 字段来判断对方响应是否成功。

3.4.14 discip_set_freq

int discip_set_freq(discip_t dsc, unsigned int rx, unsigned int tx, void *ud, void (*cb)(const struct dmr_block *, void *ud));

功能：设置收发频率。

参数：dsc 是 discip_open 创建的对象；rx 接收频率，单位 HZ；tx 发送频率，单位 HZ；ud 回调函数的最后一个参数; cb 对方响应或者超时的回调函数。

返回值: 成功返回 0，失败返回 < 0 。该返回值并不表示指令执行是否成功。指令的成功需要在回调函数中检测。当超时发生时，回到函数的第一个参数为 NULL；当它不是 NULL 时，需要从 struct dmr_block 的 `sr` 字段来判断对方响应是否成功。

3.4.15 discip_set_power

int discip_set_power(discip_t dsc, int pwr, void *ud, void (*cb)(const struct dmr_block *, void *ud));

功能：设置发射功率。

参数：dsc 是 discip_open 创建的对象；pwr 功率，$pwr \in [0, 255]$；ud 回调函数的最后一个参数; cb 对方响应或者超时的回调函数。

返回值: 成功返回 0，失败返回 < 0 。该返回值并不表示指令执行是否成功。指令的成功需要在回调函数中检测。当超时发生时，回到函数的第一个参数为

NULL；当它不是 NULL 时，需要从 struct dmr_block 的 `sr` 字段来判断对方响应是否成功。

3.5 低层 API

请先参考 3.4，在 3.4 的接口不能实现或者需要更精细的控制的时候再考虑这些接口。

这是一个异步的协议，发了一个指令之后，需要等待对方响应或者超时。连续发送多个指令时，需要一个机制来保证命令的串行化。这里抽象出来一个概念 “srb” (serial request block) 来表示每一个指令。其使用流程是: 先分配资源 (alloc), 再设置指令及其参数 (set), 然后提交 (submit)。对方响应或者超时后，该 srb 上设置的回调函数将被执行。

3.5.1 srb_alloc

srb_t srb_alloc(discip_t dsc);

功能: 分配 srb 资源。

参数: dsc 是 discip_open 创建的对象。

返回值: srb_t 对象，或者 NULL。

3.5.2 srb_free

void srb_free(srb_t srb);

功能: 释放 srb 资源。

3.5.3 srb_set_completion

void srb_set_completion(srb_t srb, void *ud, void (*f)(const struct dmr_block *, void *ud, srb_t));

功能: 设置指令完成后的回调函数。

参数: ud 回调函数的第 2 个参数；f 为完成时的回调函数, 该函数的参数规则，请参考 3.4.13。另外注意: 如果 f 不是 NULL, 在 f 函数里需要处理 srb 资源，调用 srb_free() 或者提交新的请求。

3.5.4 srb_set_timeout

void srb_set_timeout(srb_t srb, int ms);

功能：设置本指令的超时时间，它的优先级比 3.4.8 高，优选这里设置的时间。

参数: ms 超时时间，单位毫秒。

返回值: 无。

3.5.5 srb_reset

void srb_reset(srb_t srb);

功能：重置 srb 资源，回到 alloc 出来的状态。

3.5.6 srb_submit

int srb_submit(srb_t srb);

功能：提交 srb 请求。

返回值：0 成功，< 0 失败。

3.5.7 srb_set_channel

int srb_set_channel(srb_t srb, int ch);

功能: 设置信道值。

参数: ch 的信道号码。

3.5.8 srb_set_freq

int srb_set_freq(srb_t srb, unsigned int rx, unsigned int tx);

功能: 设置收发频率。

参数: rx 接收频率；tx 发送频率；频率单位为 HZ。

3.5.9 srb_set_power

int srb_set_power(srb_t srb, int pwr);

功能：设置发射功率。

参数：pwr 新的功率参数, $pwr \in [0, 255]$。

3.5.10 srb_ack

int srb_ack(srb_t srb, const struct dmr_block *req, int code, const void *data, int len);

功能: 响应模块主动发出的请求，在 3.4.9 设置的回调函数中使用。

3.5.11 srb_get_block

struct dmr_block * srb_get_block(srb_t);

功能：获取 srb 中的 dmr_block 对象。然后可以修改该对象的参数值。如果需要扩展新的指令，可以通过该函数获取 dmr_block，填写好参数后，提交出去 (submit)。这里有一个使用规则需要注意，如果 data 的长度 $\geq$ DMR_BLOCK_SIZE。则需要 malloc 并赋值给 raw 字段。

返回值: struct dmr_block * 对象。

3.5.12 示例: 实现信道切换指令

下面演示用这些 API 来实现 “信道切换” 功能:

```
static void __helper(const struct dmr_block *drb, void *ud, srb_t srb)
{
    if (drb == NULL) {
        fprintf(stderr, "timeout\n");
    } else {
        if (drb->sr == DMR_ST_SUCC) {
            fprintf(stderr, "succ\n");
        } else {
            fprintf(stderr, "failure\n");
        }
    }
    srb_free(srb);
}

int set_channel(discip_t dsc, int ch)
{
    srb_t srb = srb_alloc(dsc);
```

```
    if (srb) {
        srb_set_channel(srb, ch);
        srb_set_completion(srb, ud, __helper);
        srb_submit(srb);
        return 0;
    }
    return -1;
}
```

4 测试程序

在代码目录下执行 make ptymux 命令可以生成一个测试程序，该程序可以在没有硬件模块的情况下，模拟对方指令。可以用它来测试前面两节中的 API 接口的正确性。

执行 ptymux 程序:

```
user@debian:~$ ./ptymux
/dev/pts/1
```

将打印出可供使用的伪终端名，用它替代串口的设备名 (/dev/ttymxc1) 来调用 discip_open()，来验证程序中指令的正确性。

5 集群板 icd 程序

集群板的 icd 是一个以 C 与 Lua 的混合程序。Lua 脚本可以很容易的被 C/C++ 代码调用，也可以反过来调用 C/C++ 的函数，这使得 Lua 在应用程序中可以被广泛应用。Lua 由标准 C 编写而成，代码简洁优美，几乎在所有操作系统和平台上都可以编译，运行。一个完整的 Lua 解释器不过 200k，在目前所有脚本引擎中，Lua 的速度是最快的。icd 可执行程序是被扩展过的 Lua 解析器。

5.1 发布包中的文件

icd 发布包中包含的文件及作用:

文件名	功能
/etc/icd/iphone.lua	对讲语音逻辑实现
/etc/icd/iphone.conf	配置文件，动态生成，发布包中不包含该文件
/usr/bin/icd	扩展过的 Lua 解释器
/usr/share/lua/5.3/icd.lua	海地语音 icd 定义
/usr/lib/lua/5.3/cjson.so	lua 的 json 解析库

5.2 Lua 模块

核心功能用 C 编写，以 Lua 模块的形式导出给 Lua 脚本使用。icd 可执行程序除了可以使用标准 Lua 库，还可以使用下列模块。

5.2.1 discip

该模块是前面章节对应的 Lua API。

- `userdata->disc discip.open(string)`
- `disc:set_timeout(integer)`
- `disc:set_reqhandler(function (integer->cmd, integer->rw, integer->sr, nil or string))`
- `(integer, integer) disc:statics()`
- `disc:clean()`
- `disc:close()`
- `boolean disc:flush()`
- `disc:set_channel(integer, function)`
- `disc:set_freq(integer->rx, integer->tx, function)`
- `disc:set_power(integer, function)`
- `userdata->srb disc:alloc_srb()`
- `srb:set_timeout(integer)`
- `srb:set_completion(function("timeout" or nil, integer->cmd, integer->rw, integer->sr, nil or string))`
- `srb:free()`

- srb:submit()
- srb:reset()
- srb:set_channel(integer)
- srb:set_freq(integer->rx, integer->tx)
- srb:set_power(integer)

5.2.2 udp

给 Lua 提供 udp 功能:

- userdata->udp udp.open()
- boolean udp:bind(string->ip, integer->port)
- integer udp:sendto(string->dest, integer->port, string->data)
- integer udp:send(string->data)
- boolean udp:add_multicast_membership(string->group, string->ip)
- boolean udp:drop_multicast_membership(string->group, string->ip)
- boolean udp:add_multicast_if(string->ip)
- boolean udp:set_loop(boolean)
- udp:set_reader(function)
- udp:close()

5.2.3 mcu

EMIF 的 FPGA 通信接口:

- init()
- exit()
- set_isr(function)
- ptt(boolean)
- uhf()
- vhf()
- string->"uhf" or "vhf" is_xhf()