

类别	内容
关键词	LUA、自由串口协议
摘要	

修订历史

版本	日期	原因	编制	审查
V1.0	2021/03/24	创建文档	陈鹏	刘启鑫



销售与服务

广州大彩光电科技有限公司

电话：020-82186683

传真：020-82187676

Email: hmi@gz-dc.com（公共服务）

网站: www.gz-dc.com

地址：广州高新技术产业开发区玉树工业园富康西街 8 号 C 栋 303 房

官网零售淘宝店: www.gz-dc.taobao.com

目录

1. 适合范围.....	1
2. 开发环境版本.....	2
3. 概述.....	3
4. 参考资料.....	4
5. 教程实现.....	5
5.1.1 准备工程素材.....	5
5.2 画面配置.....	6
5.3 LUA串口接收.....	7
5.3.1 帧头+帧尾.....	7
5.3.2 帧头+数据长度.....	10
5.3.3 报文定长.....	12
5.4 LUA串口发送.....	13
6. 免责声明.....	15



1. 适合范围

本文档适合大彩支持 lua 脚本的串口屏产品使用。如 W 联型、M 系列、F 系列（固件版本 \geq V4.2.401.0）等串口屏。

2. 开发环境版本

1. VisualTFT 软件版本: V3.0.1.1111 及以上的版本。

版本查看:

1) 打开 VisualTFT 软件启动页面如图 2-1 软件版本, 右上角会显示的软件版本号;



图 2-1 软件版本

2) 打开 VisualTFT, 在软件右下角可以查看软件版本图 2-2 软件版本, 最新版本可登录 <http://www.gz-dc.com/> 进行下载。

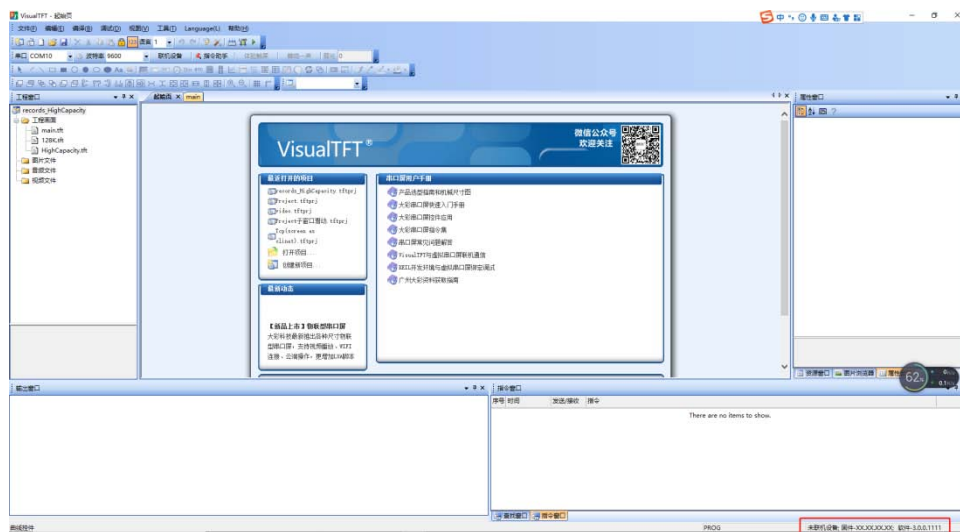


图 2-2 软件版本

2. 串口屏硬件版本:

版本查看:

- 1) 查看屏幕背面版本号贴纸;
- 2) VisualTFT 与屏幕联机成功后, 右下角显示的版本号。

3. 概述

大彩串口屏支持的协议有以下几类：

- 大彩协议：报文格式 **EE ... FF FC FF FF**。例外，用户可以自定义大彩协议，使用 **EE B5 ... FF FC FF FF** 格式，也会触发 LUA 脚本的串口接收函数：**on_uart_recv_data(packet)**，且 **packet** 是完整的一帧数据。
- MODBUS-RTU：标准 MODBUS RTU 协议，主机、从机均可以实现
- 三菱协议：标准 Mitsubishi FX2N 协议
- XGUS：自定义帧头，通过寻址方式实现
- 自由串口协议：LUA 脚本处理，可供用户二次开发实现

如果用户设备是第三方，且不支持二次开发，需要屏幕来处理交互报文，此时可以通过 LUA 脚本实现自由串口协议，本文档通过介绍常见的几种报文结构，来说明 LUA 如何开发自由串口协议。

4. 参考资料

1. 《LUA 脚本 API V1.4》可通过以下链接下载物联型开发包获取：
<http://www.gz-dc.com/index.php?s=/List/index/cid/19.html>
2. 《LUA 基础学习》可通过以下链接下载物联型开发包获取：
<http://www.gz-dc.com/index.php?s=/List/index/cid/19.html>
3. LUA 脚本初学者可以通过下面链接进行学习。
<http://www.runoob.com/lua/lua-arrays.html>

5. 教程实现

LUA 脚本接收自由串数据时，触发 `on_uart_recv_data(packet)` 函数，`packet` 不一定完整一帧报文，可能存在大概率存在分包。且帧和帧之间的也可能存在粘包的情况。所以不能用串口超时来接收每一帧，需要在初始化 `on_init()`，设置超时为 0，`uart_set_timeout(0, 0)`。在 LUA 中需要声明一个全局数组，每次触发 LUA 接收数据时候，将报文的每一个字节存在数组中。通过报文中的标识（帧头、帧尾或固定长度），把报文一帧一帧提取处理。

本例程将用灯的控制做说明，如控制灯的开关和亮度。并举例说明常见的报文格式的解析，常见的格如下所示：

- 帧头 + 帧尾
- 帧头 + 数据长度
- 报文定长

5.1.1 准备工程素材

在实现例程前需要作以下 3 个准备：

1. 硬件平台；
2. 软件平台；
3. UI 素材；

1. 硬件平台

该例程使用大彩 M 系列 7 寸串口屏 DC48048M070_1111_0C 为验证开发平台。如图 5-1 所示：

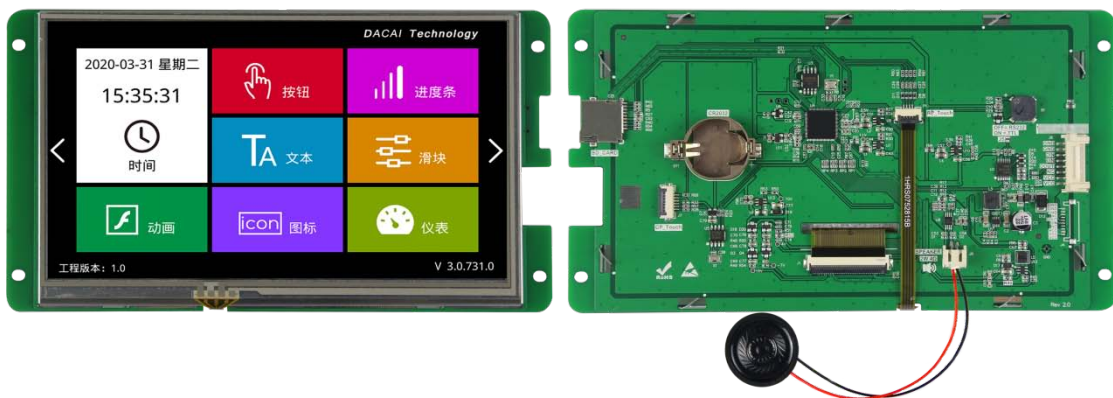


图 5-1 M 系列 7 寸串口屏

其他支持 lua 型号的串口屏均可借鉴此教程。

2. VisualTFT 上位机软件

使用大彩自主研发的上位机软件 VisualTFT 配置工程，登录 <http://www.gz-dc.com/> 下载。如图 5-2 所示：

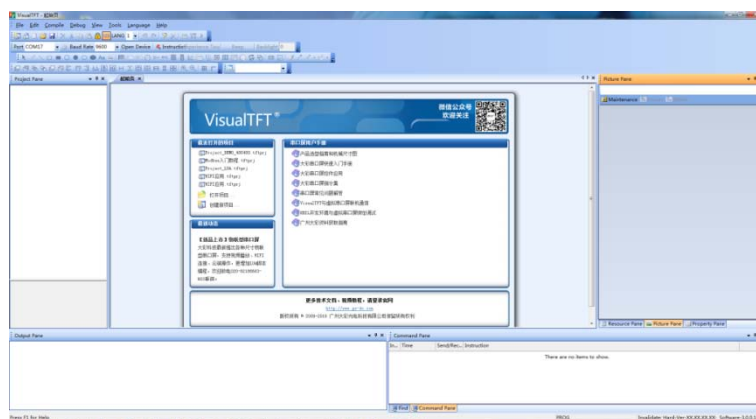


图 5-2 上位机软件

3. UI 素材

本案例涉及到的 UI 素材如所示。



图 5-3 素材准备

5.2 画面配置

在画面 ID0 中，添加两个主要控制控件。一个按钮控件（控件 ID1），作为灯开关的控制；一个进度条控件（控件 ID2），作为灯的亮度的控制。如图 5-4 所示：



图 5-4 画面配置

5.3 LUA 串口接收

LUA 中自由串口协议中，接收回调函数为 `on_uart_recv_data(packet)`

函数名: `on_uart_recv_data(packet)`

形参: `packet`，下表从 0 开始，表示接收的字节数组

5.3.1 帧头+帧尾

以帧头 `0x5A`、帧尾 `0xA5 0x5A 0xA5 0xA5`、含 `CRC16` 校验为例，屏幕对该帧结构做以下解析示例。源码可参考《1. 帧头+帧尾 (Header+End).rar》

基本思路：

1. 触发 `on_uart_recv_data(packet)`，检索是否有帧头 `0x5A`；当检索到帧头且标志 `cmd_head_tag=0` 时，将帧头标志位置位 `cmd_head_tag = 1`
2. 开始填充缓存 `buff`
3. 检测到帧尾 `cmd_head_tag = 0xA55AA5A5`，表示接收到完整一帧数据
4. 调用 `add_crc16(start, n, data)`，进行 `CRC16` 检验
5. 检验成功，调用 `my_processmessage(msg)` 执行相关操作
6. 最后清除相关的标记和缓冲，完成一帧的读取，继续获取下一帧数据

代码清单如程序清单 1 所示：

程序清单 1 帧头+帧尾

```

local cmd_end      = 0xA55AA5A5    -- 帧尾
local buff         = {}             -- 缓冲区
local cmd_length   = 0              -- 帧长度
local cmd_head_tag = 0              -- 帧头标识
local cmd_end_tag  = 0              -- 帧尾标识

-- calculate CRC16

```

```
--@data : t, data to be verified
--@n    : number of verified
--@return : check result
function add_crc16(start, n, data)

    local carry_flag, a = 0
    local result = 0xffff
    local i = start

    while(true )
    do
        result = result ~ data[i]
        for j = 0, 7
        do
            a = result
            carry_flag = a & 0x0001
            result = result >> 1
            if carry_flag == 1
            then
                result = result ~ 0xa001
            end
        end

        i = i + 1
        if i == start + n
        then
            break
        end
    end
    return result
end

--Instruction analysis
--@msg: data table
function my_processmessage(msg)

    local funccode = msg[1]
    if funccode == Func_lampState
    then

        local xth_lamp = msg[2]
        local xth_lamp_state = msg[3]
        my_set_lamp_state(xth_lamp, xth_lamp_state)
```

```
elseif funccode == Func_lampLight
then
    local xth_lamp = msg[2]
    local xth_lamp_light = msg[3]
    my_set_lamp_light(xth_lamp, xth_lamp_light)
end
end

-- 系统函数: 初始化
function on_init()
    uart_set_timeout(0, 0)
end

-- 系统函数: 串口接收函数
function on_uart_recv_data(packet)

    local recv_packet_size = (#(packet))
    local check16          = 0

    for i = 0, recv_packet_size
    do
        if packet[i] == cmd_head and cmd_head_tag == 0
        then
            cmd_head_tag = 1
        end

        if cmd_head_tag == 1
        then
            buff[cmd_length] = packet[i]
            cmd_length = cmd_length + 1
            cmd_end_tag = (cmd_end_tag << 8) | (packet[i])

            if (cmd_end_tag & cmd_end) == cmd_end
            then

                check16 = ((buff[cmd_length - 6] << 8) |
                           buff[cmd_length - 5]) & 0xFFFF

                if check16 == add_crc16(1, cmd_length - 7, buff)
                then
                    my_processmessage(buff)
                    buff          = {}
                    cmd_length    = 0
                    cmd_end_tag    = 0
                end
            end
        end
    end
end
```

```

        cmd_head_tag = 0
    else
        buff          = {}
        cmd_length    = 0
        cmd_end_tag    = 0
        cmd_head_tag = 0
    end
end
end
end
end
end

```

5.3.2 帧头+数据长度

以帧头 0x5A + 长度 LEN + DATA0...DATAn (LEN 表示 DATA 的长度) 为例，屏幕对该帧结构做以下解析示例。源码可参考《2. 帧头+数据长度 (Header+dataLen). rar》

基本思路：

1. 触发 `on_uart_recv_data(packet)`，检索是否有帧头 0x5A；当检索到帧头且标志 `cmd_head_tag=0` 时，将帧头标志位置位 `cmd_head_tag = 1`
2. 开始填充缓存 buff
3. 记录 data 的长度 `data_length = packet[1]`
4. 检测到 `data_length + 2 = cmd_length`，表示接收到完整一帧数据
5. 调用 `my_processmessage(msg)` 执行相关操作
6. 最后清除相关的标记和缓冲，完成一帧的读取，继续获取下一帧数据

代码清单如程序清单 2 所示：

程序清单 2 帧头+数据长度

```

--Instruction analysis
--@msg: data table
function my_processmessage(msg)

    local funccode = msg[2]
    print(' my_processmessage')

    print(' msg len = '..(#(msg)))
    if funccode == Func_lampState
    then

        local xth_lamp = msg[3]
        local xth_lamp_state = msg[4]
        my_set_lamp_state(xth_lamp, xth_lamp_state)

    elseif funccode == Func_lampLight
    then

```

```
        local xth_lamp = msg[3]
        local xth_lamp_light = (msg[4] << 8) | msg[5]
        my_set_lamp_light(xth_lamp, xth_lamp_light)
    end
end

-- 系统函数: 初始化
function on_init()
    uart_set_timeout(0, 0)
end

local cmd_head    = 0x5A          -- 帧头
local buff        = {}           -- 缓冲区
local cmd_length  = 0            -- 帧长度
local data_length = 0
local cmd_head_tag = 0

-- 系统函数: 串口接收函数
function on_uart_recv_data(packet)

    local recv_packet_size = #(packet)
    local check16          = 0

    for i = 0, recv_packet_size
    do
        if packet[i] == cmd_head and cmd_head_tag == 0
        then
            cmd_head_tag = 1
        end

        if cmd_head_tag == 1
        then
            buff[cmd_length] = packet[i]
            cmd_length = cmd_length + 1

            if cmd_length == 2
            then
                data_length = packet[1]
            end

            if (data_length + 2) == cmd_length
            then
                my_processmessage(buff)
            end
        end
    end
end
```

```
        buff      = {}
        cmd_length = 0
        data_length = 0
        cmd_head_tag = 0
    end
end
end
end
```

5.3.3 报文定长

没有特殊的帧头或帧尾标识，且每一帧的报文都是固定长度的，本章节假设指令报文长度就 3 个字节，进行解析。源码可参考《3. 报文定长. rar》

基本思路：

1. 触发 `on_uart_recv_data(packet)`，开始填充缓存 `buff`
2. 检测到 `cmd_length = 3`，表示接收到完整一帧数据
3. 调用 `my_processmessage(msg)` 执行相关操作
4. 最后清除相关的标记和缓冲，完成一帧的读取，继续获取下一帧数据

代码清单如程序清单 3 所示

程序清单 3 报文定长

```
--Instruction analysis
--@msg: data table
function my_processmessage(msg)

    local funccode = msg[0]

    if funccode == Func_lampState
    then

        local xth_lamp = msg[1]
        local xth_lamp_state = msg[2]
        my_set_lamp_state(xth_lamp, xth_lamp_state)

    elseif funccode == Func_lampLight
    then
        local xth_lamp = msg[1]
        local xth_lamp_light = msg[2]
        my_set_lamp_light(xth_lamp, xth_lamp_light)
    end
end

-- 系统函数：初始化
function on_init()
```



```

    uart_set_timeout(0, 0)
end

local buff      = {}          -- 缓冲区
local cmd_length = 0          -- 帧长度

-- 系统函数: 串口接收函数
function on_uart_recv_data(packet)

    local recv_packet_size = (#(packet))
    local check16          = 0

    for i = 0, recv_packet_size
    do
        buff[cmd_length] = packet[i]
        cmd_length = cmd_length + 1

        if cmd_length == 3
        then
            my_processmessage(buff)

            buff      = {}
            cmd_length = 0
        end
    end
end
end

```

5.4 LUA 串口发送

LUA 中, 串口发送函数为 `uart_send_data(packet)`

函数名: `uart_send_data (packet)`

形参: `packet`, 下表从 0 开始, 表示发送的字节数组

以程序清单 4 为例, 用户发送数据时候, 建议封装一个函数, 将变量设置成形参传递, 给对应数组元素赋值。

程序清单 4 发送示例

```

--send notify of lamp status
--@xth_lamp: which lamp
--@state: state of lamp, 0-close, 1-open

function my_uartsend_lampstate_notify(xth_lamp, state)

    local lamp_state_notify = {}

```

```
local send_crc16      = 0

lamp_state_notify[0]  = 0xA5
lamp_state_notify[1]  = Func_lampState
lamp_state_notify[2]  = xth_lamp
lamp_state_notify[3]  = state

send_crc16 = add_crc16(1, 3, lamp_state_notify)
lamp_state_notify[4] = (send_crc16 >> 8) & 0xFF
lamp_state_notify[5] = (send_crc16 >> 0) & 0xFF

lamp_state_notify[6] = 0x5A
lamp_state_notify[7] = 0xA5
lamp_state_notify[8] = 0x5A
lamp_state_notify[9] = 0x5A

uart_send_data(lamp_state_notify)
end
```

6. 免责声明

本文档提供有关广州大彩光电科技有限公司（以下简称：大彩科技）产品的信息，旨在协助客户加速产品的研发进度，在服务过程中或者其他渠道所提供的任何例程程序、技术文档、CAD 图等资料和信息都仅供参考，客户有权不使用或自行参考修改。本公司不提供任何的完整性、可靠性等保证，若是客户使用过程中因任何原因造成的特别的、偶然的或间接的损失，本公司不承担任何责任。大彩科技产品不能在用于军事、医疗、救生或维生等用途中作为唯一控制设备。

本文档并未授予任何知识产权的许可，并未以明示或暗示，或以禁止发言或其它方式授予任何知识产权许可。除大彩科技在其产品的销售条款和条件中声明的责任之外，大彩科技概不承担任何其它责任。并且，大彩科技对大彩科技产品的销售和/或使用不作任何明示或暗示的担保，包括对产品的特定用途适用性、适销性或对任何专利权、版权或其它知识产权的侵权责任等，均不作担保。大彩科技可能随时对产品规格及产品描述做出修改，恕不另行通知。