基于 Lua 脚本语言的嵌入式 UART 通信的实现

随着变电站智能化程度的逐步提高，对温度、湿度等现场状态参量的采集需求也越来越多。就目前而言，在现场应用中，此类设备多采用 RS232 或 RS485 等 UART 串行通信方式和 IED（Intelligent Electronic Device，智能电子设备）装置进行交互。一般来说，不同的设备采用的通信数据帧格式并不相同。各式各样的串口数据帧格式，对 IED 装置的软件定型造成一定的困难。传统的做法一般是由装置生产厂家指定和其配套的外围设备，装置的灵活性不够理想。本文针对此类问题，提出了一种基于 Lua 脚本语言的解决方案，可有效地提高 IED 装置对各种类型串口数据报文帧格式的适应性。该方案将具体串口报文规约的组建和解析交给 Lua 脚本进行处理，从而使设计者在装置的软件开发中，可仅关注于相关接口的设计，而不用关心具体的串口通信规约，从而方便软件的定型，并提高了装置自身在应用中的灵活性。

1 Lua 脚本语言介绍

Lua 是一种源码开放的、免费的、轻量级的嵌入式脚本语言，源码完全采用 ANSI（ISO）C.这一点使它非常适合融入目前以 C 语言为主的嵌入式开发环境之中。两者之间实现交互的关键在于一个虚拟的栈，通过该虚拟栈和 Lua 提供的可对该栈进行操作的相关接口函数，可以很方便地在它们之间实现各种类型数据的传递。

与其他脚本语言（如 Perl、Tcl、Python 等）相比，Lua 表现出了足够的简单性以及非常高的执行效率，结合其与平台的高度无关以及充分的可扩展性［1］，这使得它越来越多地得到大家的关注。因此，在本文的方案中优先选用 Lua 脚本来进行设计。

2 系统方案概述

本方案主要是围绕着 IED 装置和外围串口设备之间的通信来进行设计的，系统框架如图 1 所示。

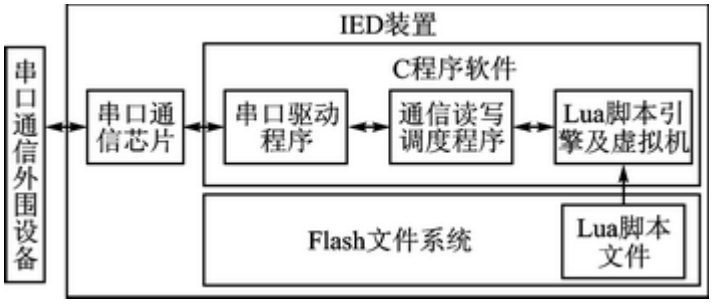

图 1 系统框架

当 IED 装置开始运行时，将创建一个用于 UART 通信的读写调度任务。在该任务中，首先通过 Lua 提供的接口函数来启动其脚本引擎，并创建 Lua 虚拟机。然后即可将用户编写的 C 函数注册到 Lua 虚拟机中去，并将存在于 Flash 文件系统中独立于装置 C 程序的 Lua 脚本文件加载到虚拟机中，从而建立起 Lua 和 C 的交互环境。在系统应用中，将需要发送到外围设备的具体数据内容都放在 Lua 脚本文件中。当装置 C 程序需要发送数据时，通过

通信读写调度程序及虚拟机的配合，将这部分数据取出，并调用串口驱动程序发送给外围设备。当收到外围设备发给 IED 装置的报文时，再将相应数据传给虚拟机中运行的脚本程序进行处理，并由 Lua 根据数据处理结果来调用已注册的 C 函数进行相关业务处理。

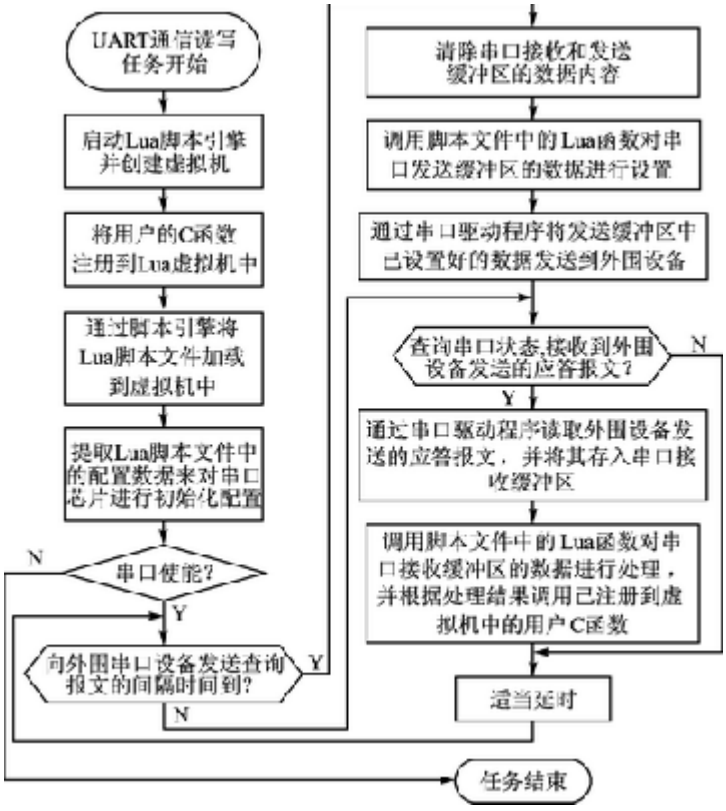

图 2 系统程序流程

本系统的程序流程如图 2 所示。

其中，串口通信芯片采用 TI 公司的带 64 字节 FIFO 的 4 通道可编程 UART 芯片 TL16C754B 来实现。它的 4 个通道可分别独立编程，在 3.3 V 的操作电压下，数据传输速率可高达 2 Mbps，适合多种 UART 通信环境中的应用［2］。基于装置的应用环境，本文采用 RS485 的问答机制并结合查询方式来对该串口通信方案进行设计。在方案实现中，装置将每隔一定时间通过串口芯片发送一次查询报文，当查询到外围设备发送的正确响应报文后，再进行相关业务处理。

3 功能实现

在嵌入式应用领域，串口通信的应用比较成熟，因此，本文将着重介绍 Lua 是如何服务于这一应用的。从图 2 可以看出，Lua 的使用主要体现在如下几个方面：

- Lua 与 C 交互环境的建立；
- 提取脚本中的串口配置数据；
- 调用 Lua 函数设置发送缓冲区；
- 通过 Lua 函数处理接收缓冲区数据。

3.1 Lua 与 C 交互环境的建立

要建立交互环境，首先要启动 Lua 脚本引擎，并创建虚拟机。其机制虽然相对复杂，但对应用来说却比较简单，通过"L=lua_open（NULL）；"即可实现。其中，L 是一个指向结构类型为 lua_State 的指针变量，该结构将负责对 Lua 的运行状态进行维护。

为了实现 Lua 脚本函数对系统程序中串口发送和接收缓存区的数据进行访问，定义了几个 C 函数供脚本调用，即用于设置串口发送缓冲区的函数 set_tx_buf、读取串口接收缓冲区的函数 get_rx_buf，以及在 Lua 脚本中判断串口数据交互正常时调用的结果处理函数 uart_ok_del.

在 Lua 脚本中，要成功调用以上函数，必须将其加载到 Lua 虚拟机中去，本文采用 Lua 提供的一种注册 C 函数库的方法来实现。具体加载过程如下：

① 按以下格式定义调用函数：

static int set_tx_buf（lua_State *L）；

static int get_rx_buf（lua_State *L）；

static int uart_ok_del（lua_State *L）；

② 声明一个结构数组，每个数组元素分别为 C 函数在 Lua 脚本中的调用名字及对应的C函数，即以"name-function"对的形式出现，如下所示：static const struct luaL_reg uartLib［］ ={

{"set_tx_buf"，set_tx_buf}，

{"get_tx_buf"， get_tx_buf}，

{"uart_ok_del"， uart_ok_de}，

{NULL， NULL}

};

③ 调用以下函数对 C 函数库进行注册：luaL_register（L， "ied"， uartLib ）；其中，参数 L 即为创建虚拟机时的函数返回值（以下同），字符串"ied"为注册到虚拟机中的库名称。第 3 个参数 uartLib 即为前面声明的结构数组，对应需要注册的库函数表。

通过以上步骤，即可完成 Lua 脚本中需要调用的 3 个 C 函数的注册过程，从而就可以在 Lua 脚本中通过"库名称。库函数"的形式来对其进行调用，如"ied.set_tx_buf（函数参数）"。

脚本文件本身的加载则相对简单，只需通过如下函数调用即可：

luaL_dofile（L， "uart_script.lua"）；

其中，参数 L 和以上的函数调用相同，第 2 个参数则为脚本文件在 Flash 中的具体存储路径。

至此，就成功建立了一个 Lua 与 C 的交互环境。

3.2 提取脚本中的串口配置数据

要正确地进行 Lua 和 C 的交互过程，首先必须对 Lua 和 C 交互时所采用虚拟栈的作用和操作有比较深入的了解。在 Lua 和 C 的交互中，它们彼此之间函数参数以及返回值都将由该栈来负责传递。Lua 和 C 在栈的操作方式上稍有不同，在 Lua 中采用严格的 LIFO 方式，而 C 则还可以通过索引的方式进行。以 3 个参数为例，参数 1 首先入栈，参数 2、3 随后顺次入栈，Lua 虚拟栈存储结构及索引对应关系如图 3 所示。

栈元素	栈索引
参数3	3(-1)
参数2	2(-2)
参数1	1(-3)

图 3 Lua 虚拟栈结构示例图

如需在 C 中访问参数 1，则既可以通过索引号 1 进行，也可通过索引号-3 进行。其中，正索引按入栈顺序从 1 依次递增，负索引按出栈顺序从-1 依次递减。

通常情况下，串口的配置主要有以下几项：是否使能、数据位数、停止位数、奇偶校验标志位和波特率。因此，在 Lua 脚本中，本文采用 Lua 的表结构对其进行设置，示例如下（本文中斜体代码表示为 Lua 脚本，以下同）：

```
uart_p0={
enable=1，--使能位
dataBits=8 ，  --数据位数
stopBits=1 ，  --停止位数
parityBit=2 ，  --奇偶校验
baudRate=9600 --波特率
};
```

该例表示对 UART 芯片的 P0 口进行使能，并且采用 8 位数据位、1 位停止位、偶校验（本文定义 parityBit 的值取 0 为无校验，取 1 为奇校验，取 2 为偶校验）的帧格式，波特率为 9 600 bps.

在 C 语言中，要获取表中 enable 属性字段的值，可采用以下步骤：

① 调用接口函数并以表名称作为参数，将该表入栈：

lua_getglobal（L， "uart_p0"）；

② 调用接口函数将 enable 属性字段的属性名称入栈：

lua_pushstring（L， "enable"）；

③ 调用接口函数提取属性值，该操作在 C 中可看作是一个先出栈再入栈的过程，结果将在②中已入栈的属性名称所在位置填入属性值：

lua_gettable（L， -2）；

其中，参数"-2"为栈中的索引号。

④ 调用接口函数取出栈顶中该属性字段的值，并调用出栈函数，以恢复调用环境：

p0_enable = （int）lua_tonumber（L， -1）；

lua_pop（L， 1）；

其中，lua_tonumber 函数的参数"-1"也为栈中的索引号，该操作将取出栈顶元素的数值，鉴于 Lua 中的数据都为浮点数，所以需将其强制转换为整型数据。lua_pop 中参数"1"为非索引，仅说明从栈顶将 1 个元素出栈。

通过以上操作，就可以正确地取出脚本中 p0 口参数设置表中 enable 属性字段的值。其他属性字段的提取与其相同。虚拟栈中的内容变化如图 4 所示。

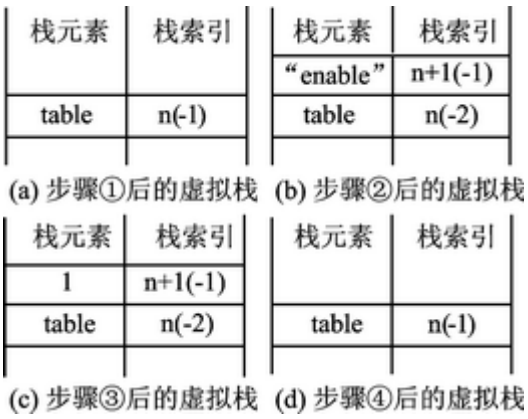

图 4 提取表中属性值时的虚拟栈操作示意图

3.3 调用 Lua 函数设置发送缓冲区

为通过 Lua 脚本对串口发送缓冲区进行设置，在脚本中定义了如下函数：

```
data ={0x11， 0x22， 0x33， 0x44， 0x55 };
function uart_p0_set_txBuf（）
local port=0;
local p0_send_num=5;
for i=1， p0_send_num do
ied.set_tx_buf（port，i-1， data［i］）
end
return p0_send_num
end
```

从脚本内容可以看出，在此采用了一个 Lua 中的循环结构对发送缓冲区进行设置，并返回设置的数据个数。其中，全局变量 data 是 Lua 脚本中的表，类似于数组，在此表示需要设置的缓冲区内容；ied.set_tx_buf（）为在 3.1 节中提到的已注册到虚拟机中的 C 函数库中的一个函数。其参数 port 表示端口号，i-1 表示缓冲区索引号，data［i］表示具体的数据内容。在应用中需要注意的是，在 Lua 中，数组索引默认从 1 开始，而不像 C 中从 0 开始。另外，在 C 中定义 set_tx_buf 函数时并未设置参数，这主要是因为参数的提取必须借助于虚拟栈才能实现。在脚本中调用时，对其参数将按照从左到右的顺序依次入栈，在 C 中要取出参数时，按照其在栈中相应的索引号取出即可。在 Lua 中对每个函数的调用都有一个独立的栈，因此，若以 i 取 2 时调用情况为例，在 C 函数 set_tx_buf 中看到的栈内容将如图 5 所示。

栈元素	栈索引
0x22	3(-1)
1	2(-2)
0	1(-3)

图 5 函数调用时的虚拟栈示例

从而在 C 程序中，只需要调用下面语句即可将该串口发送缓冲区中索引为 1 的内存区域设置成 0x22：

```
port=（int）lua_tonumber（L，1）；//取端口号
```

index=（int）lua_tonumber（L，2）；//取索引

data=（char）lua_tonumber（L，3）；//取数据

uart_port_tx_buf［port］.data［index］=data;

当在 C 程序中需对串口发送缓冲区进行设置时，将按如下方法调用该脚本函数：

lua_getglobal（L， "uart_p0_set_txBuf "）；

lua_pcall（L， 0， 1， 0）；

其中，函数 lua_getglobal 的参数"uart_p0_set_txBuf"为要调用的脚本函数名，函数 lua_pcall 的函数原型为：

int （lua_pcall） （

lua_State *L，

int nargs， //调用函数的参数个数

int nresults， //返回的参数个数

int errfunc //错误处理函数号

）；

因所调用的脚本函数 uart_p0_set_txBuf 没有参数，有一个返回值，所以分别将 nargs、nresults 置为 0、1，而错误处理函数暂不使用，故置为 0.

对于脚本中的返回值，将在脚本函数调用结束时，置于 lua_pcall 调用环境所在的虚拟栈的栈顶中，可由 C 程序根据索引取出。

经以上过程，就完成了对串口发送缓冲区的内容设置，然后就可以通过串口芯片的驱动程序将其发送到外围设备。

在现场应用时，只需根据不同外围设备问询报文的要求来修改脚本中 data 数组以及 p0_send_num 变量的内容即可，而不用对装置的 C 程序进行任何修改。

3.4 通过 Lua 函数处理接收缓冲区数据

通过 Lua 和 C 的交互来对串口接收缓冲区数据的处理方法同发送缓冲区的处理基本相似。

当装置通过串口驱动程序将外围设备发来的数据置入接收缓冲区后，在 C 函数中调用脚本函数：

lua_getglobal（L， "uart_p0_del_rxBuf"）；

lua_pushnumber（L， size）；

ret=lua_pcall（L， 1， 1， 0）；

其中，参数 uart_p0_del_rxBuf 为脚本中定义的缓冲区数据处理函数名，通过 lua_pushnumber 将接收数据的大小入栈，从而传给 Lua 脚本函数，脚本函数的原型如下：

function uart_p0_del_rxBuf（rx_size）

在该函数中，可通过调用注册的 C 函数 get_rx_buf 来获取接收缓冲区中的内容：

data［i］ = ied.get_rx_buf（port，index）

其中，data 为脚本中类似于数组的表类型。port 为串口芯片的端口号，index 为缓冲区的索引号，在 C 程序中通过以下语句对脚本返回所取数据值：

port=（int）lua_tonumber（L，1）；//取端口号

index=（int）lua_tonumber（L，2）；//取索引

data=uart_port_rx_buf［port］.data［index］;

lua_pushnumber（L， data）；//返回值入栈

可以看出，在脚本中也是借助于虚拟栈来获取 C 程序的返回值。通过以上方法成功获取了串口接收缓存区的内容后，就可根据具体的外围设备在脚本中对其接收数据的正确性进行判断，如果判断结果正确，则调用前面注册的 C 函数 uart_ok_del 进行相关业务处理。

ied. uart_ok_del （port）

结语

从本文提供的方案可以看出，从始至终，IED 装置的 C 语言应用程序在 Lua 虚拟机与外围设备之间，除了报文的透明传输功能外，并不负责具体数据业务的处理，这就使在 C 程序的设计中完全不需要考虑外围设备所采用的串口通信数据格式，具体的数据内容都可放在脚本文件中进行设置和处理。在现场应用中，就可以达到仅修改 Lua 脚本文件就能完成 IED 装置与不同的串口通信外围设备之间的数据交互功能，从而实现对装置串口通信规约的现场可配置化