

# Case Study on Communication between Embedded Linux Environment and Microcontroller

Ziheng Gao, Yizhun Peng \*, Shuo Wang

*College of Electronic Information and Automation, Tianjin University of Science and Technology,  
China*

*Binhai International Advanced Structural Integrity Research Centre, Tianjin University of Science  
and Technology, China*

*E-mail: \* pengyizhun@tust.edu.cn*

## Abstract

In many embedded development scenarios, we need to combine the real-time microcontroller and non-real-time embedded Linux environment for collaborative development. In a high real-time environment (such as: vehicles, production lines), it can complete kinds of features relying on huge Linux ecosystem, such as hot fixable functional modules, improved network communication, easier OTA firmware updates, more efficient algorithm capabilities, and more objects. This paper aims to enumerate and explore several methods that can be used to implement similar duplex communications. For ease of understanding, a single-board computer called Raspberry Pi running Raspbian Linux and an STM32F103 32-bit ARM microcontroller are used in experiment.

*Keywords:* communication, embedded, real-time, Raspberry Pi, ARM

## 1. Introduction

IoT (Internet of things) has become closer to people's daily life, such as smart speakers and remoting control table lamps. As the working basis of the intelligent equipment to meet the products' demands, embedded development is more complicated. Lots of terminal equipment need to connect the Internet to run high computational tasks and high IO tasks. They will be used to develop deep learning models online for predictions.

Excellent native support for huge ecosystems and asynchronous tasks in an embedded Linux environment can accommodate all of the above. In addition, native compatibility of different types of scripting languages optimizes workflow.

However, traditional embedded development must operate in a real-time environment to meet reliability and other requirements. Obviously, the contradiction between them cannot be solved in the same environment. Therefore,

using two controllers to process real-time and non-real-time tasks separately is a good solution.

Communication is an inevitable problem between two controllers. This article focuses on how to build a complete and reliable communication solution. We ran the Raspbian Linux distribution of the single-board computer Raspberry Pi and the 32-bit ARM microcontroller-STM32F103 as an example. It is worth mentioning that Raspbian is a customized version of Debian. It was created to accommodate the ARM commands of the Raspberry pi. Essentially, it's a full-featured Linux distribution.

## 2. Choices

The communication protocol is an inevitable part when it comes to communication. The protocol is also divided into application layer protocols and underlying protocols according to the level. Firstly, we will consider the most comprehensive multi-layer communication protocol stack,

such as the TCP/IP<sup>1</sup> stack, which is used widely in Internet, and the CAN bus, which is commonly used in industrial control. This article will introduce four communication protocols.

### 2.1 Ethernet protocol

Ethernet is one of the most popular computer LAN technologies<sup>2</sup>. The advantage is that the ecosystem and protocol stack is mature. For example, TCP/IP is well known, but its standard topology is the bus topology.

### 2.2 CAN bus

The Controller Area Network is a feature-rich automotive bus standard based on the Broadcast Communication Mechanism. According to the content of the message, message Identifier is used to define the priority of content and messages for delivery. It is a multi-master serial bus standard for connecting ECUs<sup>4</sup>. The CAN network includes multiple ECU nodes which can be input/output devices, embedded devices that include CAN interchangers or gateways. It is concluded that CAN bus communication is generally applicable in large industrial equipment, but the disadvantages in the underlying two-machine communication are similar to Ethernet.

### 2.3 I<sup>2</sup>C bus

Inter-Integrated Circuit is a serial communication bus that uses a master-slave architecture. For simple two-wire communication, serial data line (SDA) and serial clock line (SCL) are recommended to use when the amount of data is not large and data structure is not responsible.

### 2.4 Serial communication bus: SPI and UART

Serial communication is essentially on the bus and other data channels and continuously performs the communication process of the above single process. The corresponding method is parallel communication. This article only describes two examples of SPI and UART.

#### 2.4.1 Serial Peripheral Interface

SPI is a high-speed, full-duplex, synchronous, serial communication bus, working in master-slave mode, independent transceiver. The SPI bus consists of SCLK (serial clock), SDI (serial data input) and SDO (serial data output). CS.SPI is a protocol that allows a master device to initiate a synchronous communication with several slave devices. The data lines of the SPI for input and output are independent, so it is allowed to complete the input and output of data at the same time.

#### 2.4.2 Universal Asynchronous Receiver Transmitter

UART is a two-wire, full-duplex, asynchronous serial communication bus. There are only two lines, one for sending and one for receiving. The timing requirements for both parties are relatively strict, and the communication speed is not very fast. However, it is simple and easy, and it is suitable for transmitting data between two better-performing controllers.

## 3. CASE

### 3.1 Choose plan

The use of serial communication UART to implement the two-machine communication we describe is a very cost-effective opinion. Below, we will explain how to make this solution practical by designing a simple two-machine communication protocol. We will use the above-mentioned single-board computer Raspberry Pi and STM32F103 microcontroller to demonstrate.

### 3.2 Implementation plan

#### 3.2.1 Configuration

On the Raspberry Pi<sup>8</sup>, we will use the highly acclaimed Python to finish the programming. The Raspberry Pi 3 Model B has two sets of UART serial ports, one hardware serial port for Bluetooth, and one software serial port for GPIO pins. For performance considerations, we have to exchange them. Then, we need bind the GPIO pins to the hardware serial port instead of BT. Specific steps are as follows<sup>9</sup>:

**A.** Turn off the onboard Bluetooth feature. The steps are as described above.

**B.** Restore the serial port and set it as a universal serial port. Edit document: */boot/config.txt* at the end of the document, add a statement: *dtoverlay=pi3-miniuart-bt*.

Edit document: */boot/cmdline.txt*, next step replace the contents of the document with the following:

```
dwc_otg.lpm_enable=0
```

```
console=ttyL
```

```
root=/dev/mmcblk0p2
```

```
rootfstype=ext4
```

```
elevator=deadline
```

```
fsck.repair=yes rootwait
```

Above we have completed the basic configuration of the serial port.

### 3.2.2 Programming

**A.** Reference the pyserial<sup>6</sup> library for serial programming. It has been shown in the figure1.

PySerial encapsulates the serial communication module and supports different platforms such as Linux, Windows and BSD. Also, it is a python support module. This module encapsulates the access rights of the serial port. The module named “Stand” will automatically select the appropriate backend.

**B.** On the STM32<sup>7</sup>, the communication part code is as figure 2.



Fig1. pyserial library for serial programming

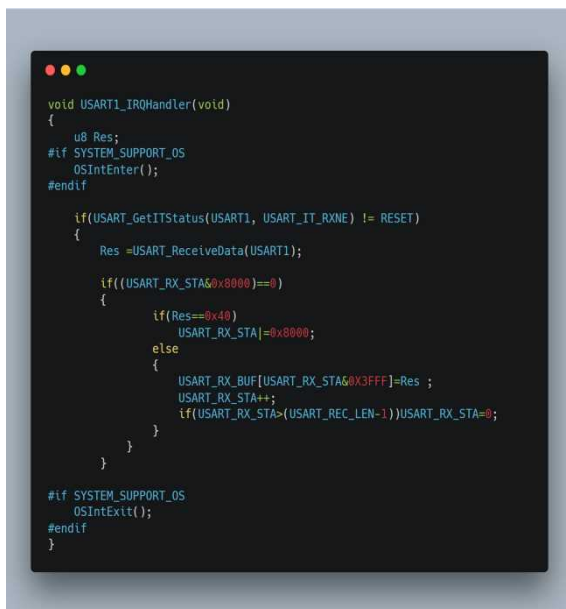


Figure 2: the communication part code

### 3.2.3 Design

**A.** After solving the problem of how to initiate communication on both sides, it is necessary to design a set of application layer communication protocol to unify the standard of multi-party hardware communication. In order to make the communication content can be recognized bilaterally. Also, easy to increase or decrease the modification function. This article thinks that the protocol standard needs to meet the following.

- The unit of communication content is a message frame, including the frame header, the message body, and the end of the frame, all of which are ASCII characters.
- The message body includes a variety of data, each of which has its corresponding flag, data body, separator between the data and the data body, and a separator between the data.
- Each type of data is distinguished by a identifier, and the meaning and data type of its representative are negotiated in advance.
- The final assembled message frame can be divided, recognized, converted, and processed in the form of a string or an array of characters as both ends.

**B.** As an example, this article will introduce a set of application layer communication protocols designed in a vehicle project.

Basic definition of interface:

**a.** Serial communication using the UART protocol, the baud rate is 115200.

**b.** In the form of a packet, the header is defined as \*(ASCII code 0x2A) and the trailer is defined as # (ASCII code 0x23)

The data in data packet must have an identifier,

**a)** Which is any uppercase letter in ASCII, followed by the data body.

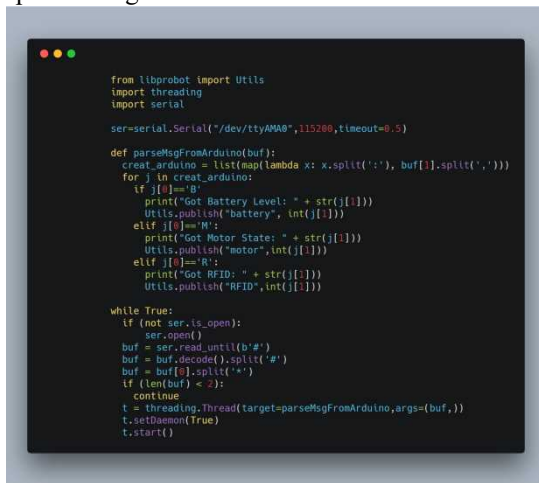
**b)** The data identifier is separated from the data body by ‘.’ (ASCII code 0x3A)

**c)** Multiple data is separated by ‘,’ (ASCII code 0x2C) in the same package.

Identifier	Type	Description	When
A	null	Package of sync request, empty body	Request micro-controller sync data
B	uint8	Battery level of percentage system	Battery level is updated
M	uint8	Motor state, 1-5 means go/back/left/right/stop	Motor state (need) update
R	string	RFID Tag's UID of ten system	Sensor have read a new tag
Z	null	Sync header, empty body, represent the frame used by sync data	The micro-controller receive request('A' package)

Figure 3: The custom identifier

Data return confirmation and synchronization: In order to make the communication reliable, two ends can send the packet back to the secondary station after receiving the data that needs to be consistent at both ends. Besides, aim to synchronize the upper and lower data, the logical state can still obtain the underlying state after the restart, and the bottom layer should send all the latest data to the logical end the synchronization data frame flag 'Z' should be added to the header of the post back message. The certain example is as figure 4.



```
from librobot import Utils
import threading
import serial

ser=serial.Serial("/dev/ttyAMA0",115200,timeout=0.5)

def parseMsgFromArduino(buf):
    creat_arduino = list(map(lambda x: x.split(':'), buf[1].split(',')))
    for j in creat_arduino:
        if j[0]=='B':
            print("Got Battery Level: " + str(j[1]))
            Utils.publish("battery", int(j[1]))
        elif j[0]=='M':
            print("Got Motor State: " + str(j[1]))
            Utils.publish("motor",int(j[1]))
        elif j[0]=='R':
            print("Got RFID: " + str(j[1]))
            Utils.publish("RFID",int(j[1]))

while True:
    if (not ser.is_open):
        ser.open()
    buf = ser.read_until(b'#')
    buf = buf.decode().split('#')
    buf = buf[0].split(',')
    if (len(buf) < 2):
        continue
    t = threading.Thread(target=parseMsgFromArduino,args=(buf,))
    t.setDaemon(True)
    t.start()
```

Figure 4.Certain example of programming

#### 4. Conclusion

The above is an implementation scheme that we proposed and completed. We use the UART serial port as the

transport layer protocol to design the application layer protocol.

#### 5. Acknowledgement

This research was partially supported by Student's Platform for Innovation and Entrepreneur Training Program, the Ministry of education of China (201810057019) . We are also very grateful to Yang Jiasheng for his help in the revision of the manuscript.

#### REFERENCE:

1. LiMing. Research and development of embedded TCP/IP protocol stack. Computer engineering and Application, 2002, (8): 18-19
2. IEEE 802.3-2015 Ethernet protocol official document
3. CJ1W Defined CAN Unit Operational Manual
4. Wang Weixin. Principle and interface technology of single chip microcomputer: China Agriculture Press,2013.08
5. Song Qinhua. Shipboard Electronic Countermeasure. Comparison of online programming mechanism between SPI and UART, 2015, (10): 66-67
6. pySerial 3.0 documentation. <https://pythonhosted.org/pyserial/>.
7. Wang Wei. Design of communication system without host computer based on STM32F103. Tianjin Polytechnic University,2017.
8. Raspberry official technical documents. <https://www.raspberrypi.org/blog/>.
9. THE RASPBERRY PI UARTS. <https://www.raspberrypi.org/documentation/configuration/uart.md>.