Using CAN

Protocol Introduction

CAN CAN is the abbreviation of Controller Area Network, and its Chinese name is Controller Area Network. It is an ISO internationally standardized serial communication protocol and a serial communication protocol bus for real-time applications. It can use twisted pair cables to transmit signals and is one of the most widely used field buses in the world.

Advantages of CAN bus:

- High reliability and strong anti-interference ability. The phy chip periphery is based on differential signals and twisted pair cables for transmission, which effectively offsets electromagnetic interference; the hardware-cured data link layer can automatically solve the problem of clock asynchrony between multiple nodes.
- Strong error detection capability. The hardware-cured data link layer includes detection capabilities such as CRC and bit detection, which can almost 100% detect communication anomalies.
- Perfect error management mechanism. If the message frame arbitration fails or is damaged during transmission, it can be automatically retransmitted; in the case of serious errors, the node has the function of automatically disconnecting from the bus, cutting off its connection with the bus without affecting the normal operation of the bus.
- Strong real-time performance. The CAN bus has a high data transmission rate and can process and transmit large amounts of data in real time, meeting the real-time requirements of automotive electronic control systems.
- Low cost The hardware cost of CAN bus is relatively low, which can reduce the overall cost of automotive electronic control system.
- Long communication distance and fast message transmission speed The direct communication distance can reach up to 10km (rate below 4Kbps); the communication rate can reach up to 1Mbps (the maximum distance is 40m at this time).
- Support multi-host communication and lossless arbitration technology When two nodes publish information at the same time, the high priority message can transmit data without being affected.
- Abolition of the "address" concept, efficient and flexible One of the biggest features of CAN protocol is the abolition of traditional station address encoding, instead of communication

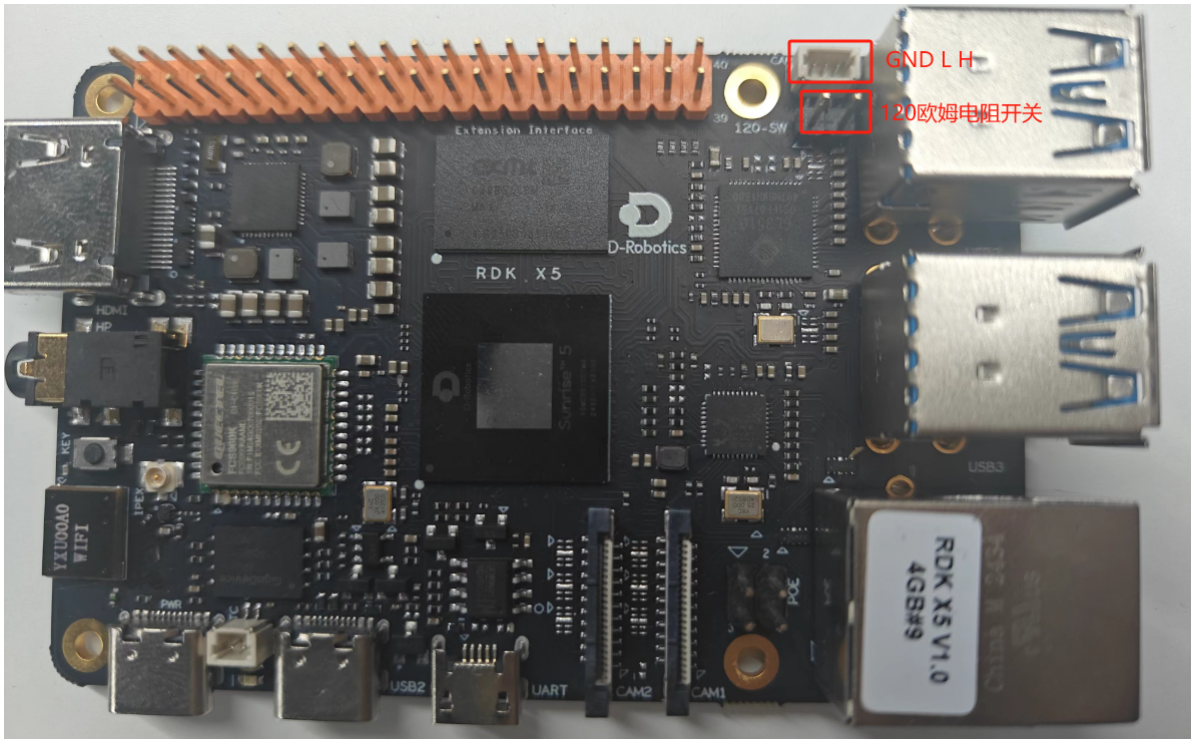
message encoding.

- Line encoding 1) The number of network nodes constituted by CAN is unlimited, and it has great flexibility in use; 2) When adding nodes to the bus, it will not affect the software, hardware and application layer of the existing nodes on the bus.
- Simple structure, convenient networking, strong scalability When adding new devices, just put them on the twisted pair, without additional line modification (such as SPI, etc.).

CAN FD The strong market demand for improving the performance of the CAN bus has led to the emergence of CAN-FD (Flexible Data rate). CAN-FD promotes the advantages of CAN and makes up for its shortcomings. Its main features are as follows:

- Extended effective data field The effective data field of each frame message of the CAN-FD protocol can reach 64 bytes5, while the traditional CAN protocol has only 8 bytes. This greatly increases the data transmission capacity, allowing more data to be transmitted in a single message.
- Dual bit rate mode The CAN-FD protocol introduces a dual bit rate mode, which uses the nominal bit rate (up to 1 Mbit/s) in the arbitration segment and a higher data bit rate (up to 5 Mbit/s) in the data segment. This mode can increase the data transmission rate while ensuring compatibility.
- Improved cyclic redundancy check and stuffing bit counter The CAN-FD protocol uses an improved cyclic redundancy check (CRC) and stuffing bit counter (SBC) to improve error detection capabilities. These improvements enhance data integrity and reliability.
- Cancel support for remote frames To simplify the frame structure, the CAN-FD protocol cancels support for remote frames. This means that in a CAN-FD network, communication is done via data frames, and control and request via remote frames are no longer required.

Interface Description



- RDK X5 provides a CAN communication interface and is equipped with a 120 ohm terminal resistor switch. When the terminal resistor needs to be enabled, just close the switch.
- Terminal interface model: SH1.0 1X3P

Module Introduction

RDK X5 integrates TCAN4550 TCAN4550 is a CAN FD controller with an integrated CAN FD transceiver, supporting data rates up to 8Mbps. This CAN FD controller complies with the ISO11898-1:2015 High-speed Controller Area Network (CAN) data link layer specification and meets the physical layer requirements of the ISO11898–2:2016 High-speed CAN specification. TCAN4550 provides an interface between the CAN bus and the system processor through a serial peripheral interface (SPI), supports both classic CAN and CAN FD, and implements port expansion or CAN support for processors that do not support CAN FD. TCAN4550 provides CAN FD transceiver functions: differential transmission capability to the bus and differential reception capability from the bus.

- Supports 1M CAN 2M CAN FD

Driver Guide

dts

```
&spi5 {
  status = "okay";
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_spi5 &lsio_gpio0_7 &lsio_gpio0_12>;

  tcan4x5x: tcan4x5x@0 {
    compatible = "ti,tcan4x5x";
    reg = <0>;
    #address-cells = <1>;
    #size-cells = <1>;
    spi-max-frequency = <10000000>;
    bosch,mram-cfg = <0x0 0 0 16 0 0 1 1>;
    interrupt-parent = <&ls_gpio0_porta>;
    interrupts = <12 IRQ_TYPE_EDGE_FALLING>;
    reset-gpios = <&ls_gpio0_porta 7 GPIO_ACTIVE_HIGH>;
  };
};
```

Drive code

```
kernel\drivers\net\can\m_can\tcan4x5x-core.c
```

can-utils tool introduction

can-utils is a set of open source tools for Linux operating systems, specifically designed to handle tasks related to the CAN (Controller Area Network) bus. The CAN bus is widely used in automotive and industrial automation for communication between devices. This toolset provides a variety of command-line tools for sending, receiving and processing data on the CAN network. For example: cansend: Send a single CAN frame. candump: Capture and display data passing through the CAN interface. canplayer: Replay data recorded by candump. cansniffer: Display changes in CAN data. can-utils also includes some tools for advanced functions, such as setting CAN hardware filters or debugging CAN devices and networks. These tools are provided through a command-line interface and can be flexibly integrated into scripts and automation systems.

Basic usage

1. candump Display, filter and record CAN data

Basic usage:

```
candump can0
```

- Displays all CAN data passing through the can0 interface.

Filter for a specific ID:

```
candump can0,123:7FF
```

- Display CAN frame with ID 123.

Log data to file:

```
candump -l can0
```

- This will log the data through can0 to a file, the default file name format is candump-date.log.

2. canplayer canplayer is used to play back the CAN data log recorded with candump.

Basic usage:

```
canplayer -I candump.log
```

- Will replay the CAN data recorded in the file candump.log.

3. cansend is used to send the specified CAN frame

Basic usage:

```
cansend can0 123#1122334455667788
```

- Send a CAN frame with ID 123 and data content 1122334455667788 to the can0 interface.

4. cangen Generates random or regular CAN traffic for testing or simulation.

Basic usage:

```
cangen can0 -I 1A -L 8 -D i -g 10 -n 100
```

- Generate 100 incremental packets with ID 1A and length 8 bytes on can0, with 10 ms interval between each packet.

5. cansequence Send a series of CAN frames with increasing payload and check if there is frame loss.

Basic usage:

```
cansequence can0
```

- Send and inspect a series of CAN frames with increasing payload on can0.

6. cansniffer is used to display changes in CAN data, which is very helpful for debugging and understanding data flow.

Basic usage:

```
cansniffer can0
```

- Monitor and display any changes in CAN data on the can0 interface.

Test Guide

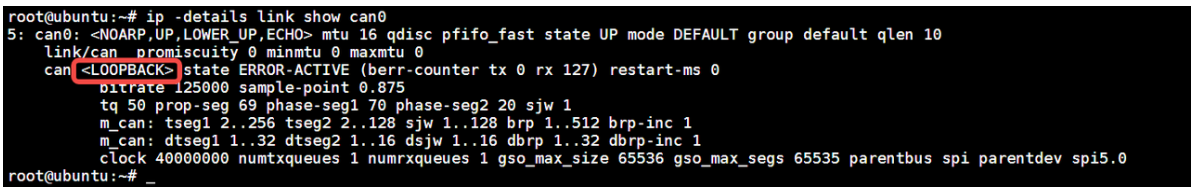
Loopback Test

Configure the can bus bit rate and loopback mode

```
ip link set down can0
ip link set can0 type can bitrate 125000
ip link set can0 type can loopback on
ip link set up can0
```

View can0 configuration information

```
ip -details link show can0
```



Continue to enter the receiving command (background receiving, can not block the serial port, the subsequent need to enter the sending command):

```
candump can0 -L &
```

Send test input, and under normal circumstances the test data will be received immediately:

```
cansend can0 123#1122334455667788
```

Test Results

```
root@ubuntu:~# cansend can0 123#1122334455667788
root@ubuntu:~# (0946687937.396240) can0 123#1122334455667788
(0946687937.396255) can0 123#1122334455667788
```

CANFD loopback test

The arbitration segment baud rate is 500K, and the data field baud rate is 2M

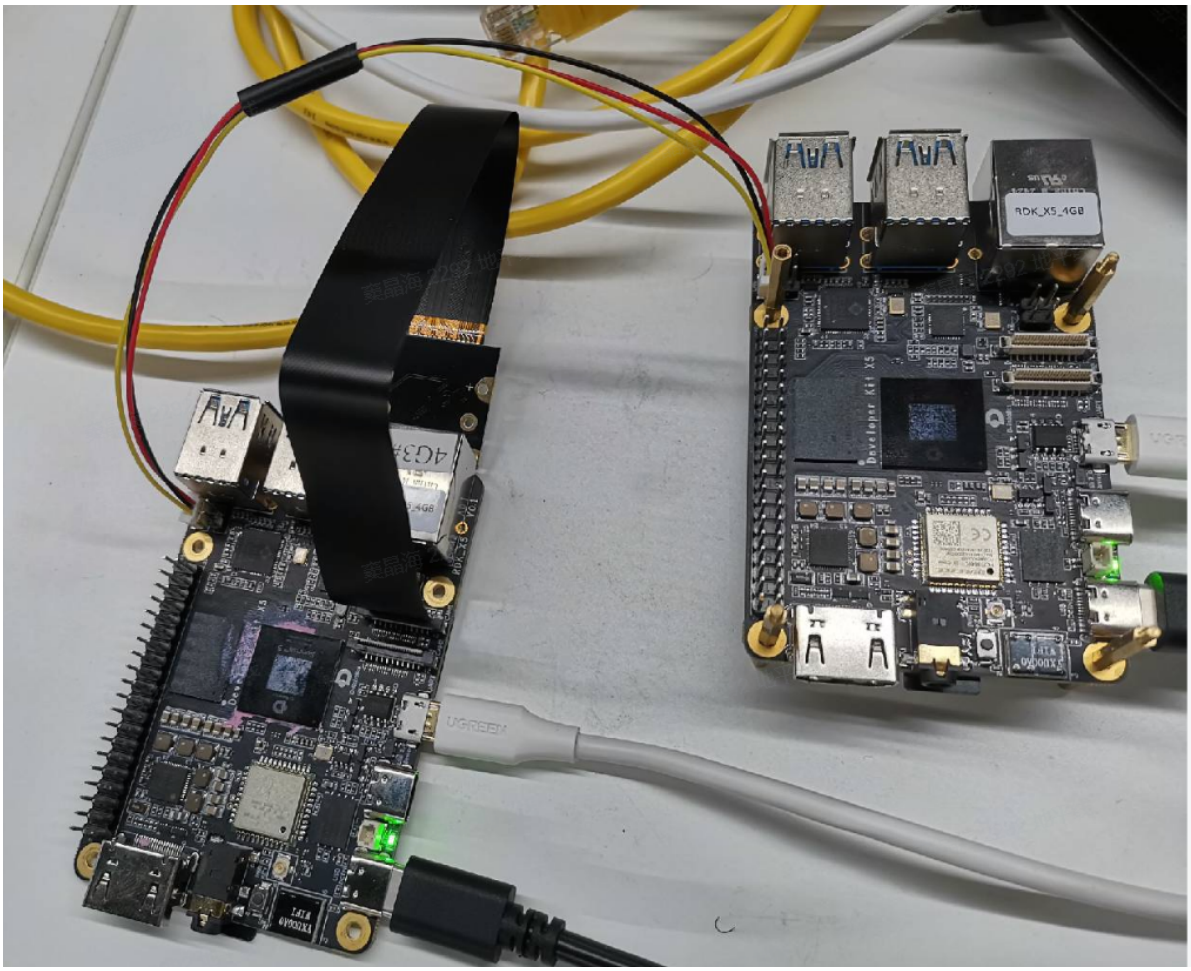
```
ip link set can0 down
ip link set can0 up type can bitrate 500000 dbitrate 2000000  fd on
ip link set can0 type can loopback on
ip link set can0 up
```

Send, receive CAN FD data

```
candump can0 -L &
cansend can0 123##300112233445566778899aabbccddeeff
```

Dual device communication test

Hardware connection



- GND to GND L to L H to H

Test command Two devices are configured with the same CAN bus bit rate

```
ip link set down can0
ip link set can0 type can bitrate 125000
ip link set up can0
```

A device receives

```
candump can0 -L
```

A device sends

```
cansend can0 123#1122334455667788
```

Application Guide

Linux provides the SocketCAN interface, which makes CAN bus communication similar to Ethernet communication, and the application development interface is more universal and more flexible. Using socketCAN is just like using TCP/IP. Here is a simple sending and receiving routine:

First, configure the CAN bus bit rate and loopback mode

```
ip link set down can0
ip link set can0 type can bitrate 125000
ip link set can0 type can loopback on
ip link set up can0
```

Writing a program

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <linux/can.h>
#include <linux/can/raw.h>
#include <sys/socket.h>
#include <net/if.h>
#include <sys/ioctl.h>
#include <fcntl.h>

int main() {
  int sock;
```

```
    struct sockaddr_can addr;
    struct can_frame frame;

    // Create Socket
    sock = socket(PF_CAN, SOCK_RAW, CAN_RAW);
    if (sock < 0) {
        perror("Socket");
        return 1;
    }

    // Get can0 interface
    struct ifreq ifr;
    strcpy(ifr.ifr_name, "can0");
    ioctl(sock, SIOCGIFINDEX, &ifr);

    // Binding Socket
    addr.can_family = PF_CAN;
    addr.can_ifindex = ifr.ifr_ifindex;
    bind(sock, (struct sockaddr *)&addr, sizeof(addr));

    // Send Message
    frame.can_id = 0x123;
    frame.can_dlc = 4;
    memcpy(frame.data, "\xde\xad\xbe\xef", 4);
    write(sock, &frame, sizeof(struct can_frame));

    // Receiving Messages
    while (1) {
        int nbytes = read(sock, &frame, sizeof(struct can_frame));
        if (nbytes > 0) {
            printf("Received: ID: 0x%X Data: ", frame.can_id);
            for (int i = 0; i < frame.can_dlc; i++) {
                printf("%02X ", frame.data[i]);
            }
            printf("\n");
        }
    }

    close(sock);
    return 0;
}
```

Compile and run Save the code to the file `can_loopback.c`.

Compile the program:

```
gcc -o can_loopback can_loopback.c
```

Run the program:

```
sudo ./can_loopback
```

The above code will send a CAN message and continuously receive and print the received messages.