Rugged Board

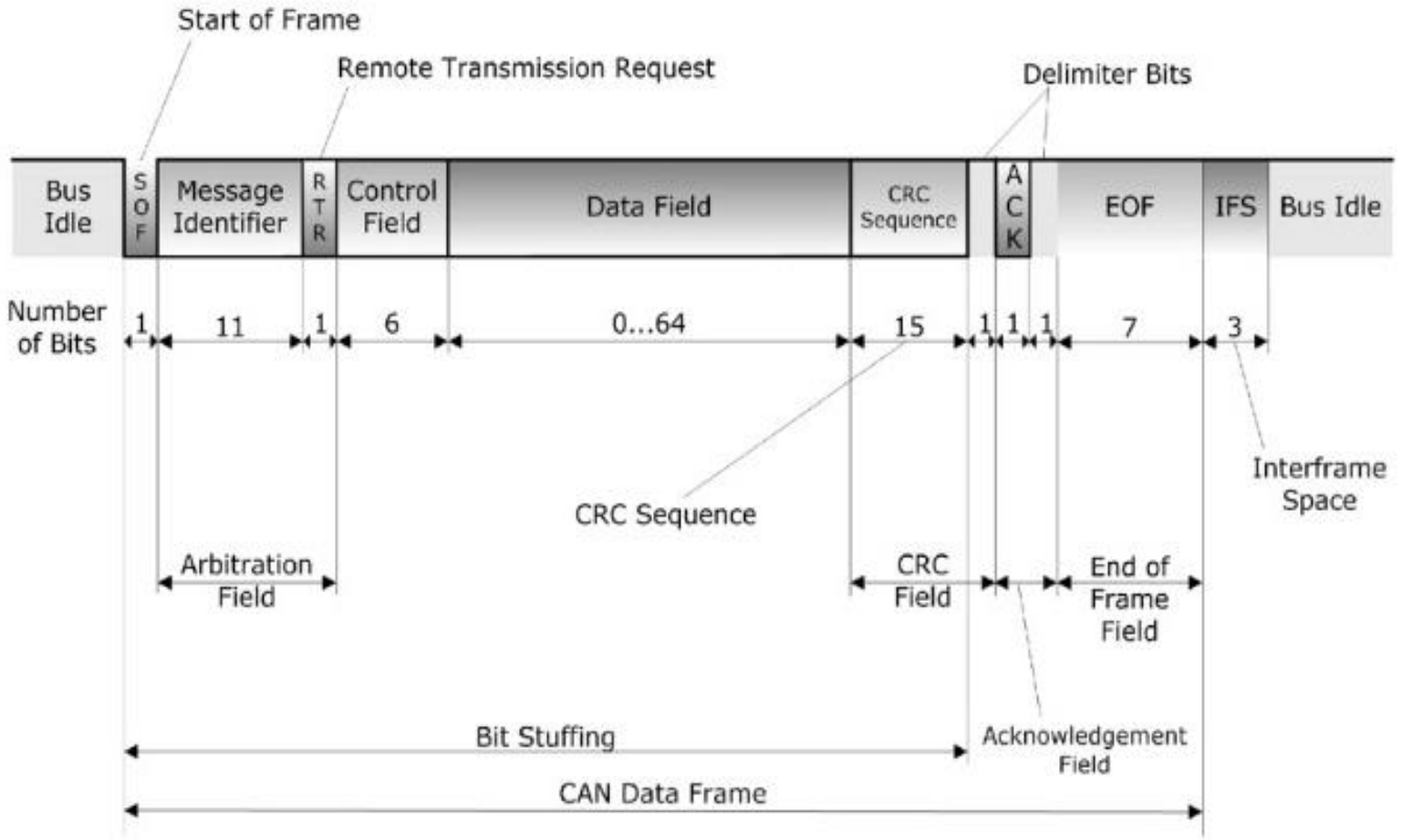
CAN

https://community.ruggedboard.com

RB-CAN

- Development of the CAN bus started in 1983 at Robert Bosch GmbH.
- The protocol was officially released in 1986 at the Society of Automotive Engineers (SAE) conference in Detroit, Michigan.
- The first CAN controller chips were introduced by Intel in 1987, and shortly thereafter by Philips. Released in 1991, the Mercedes-Benz W140 was the first production vehicle to feature a CAN-based multiplex wiring system.
- Bosch published several versions of the CAN specification and the latest is CAN 2.0 published in 1991.
- This specification has two parts; part A is for the standard format with an 11-bit identifier, and part B is for the extended format with a 29-bit identifier.
- A CAN device that uses 11-bit identifiers is commonly called CAN 2.0A and a CAN device that uses 29-bit identifiers is commonly called CAN 2.0B.
- These standards are freely available from Bosch along with other specifications and white papers.

RB-CAN


Start of Frame
Remote Transmission Request
Delimiter Bits
Bus Idle
SOF
Message Identifier
RTR
Control Field
Data Field
CRC Sequence
ACK
EOF
IFS
Bus Idle
Number of Bits
1
11
1
6
0...64
15
1
1
1
7
3
Interframe Space
CRC Sequence
Arbitration Field
CRC Field
End of Frame Field
Bit Stuffing
Acknowledgement Field
CAN Data Frame

Field	Length (bits)	Description
Start of Frame (SOF)	1	Must be dominant
Identifier	11	Unique identifier indicates priority
Remote Transmission Request (RTR)	1	Dominant in data frames; recessive in remote frames
Reserved	2	Must be dominant
Data Length Code (DLC)	4	Number of data bytes (0–8)
Data Field	0–8 bytes	Length determined by DLC field
Cyclic Redundancy Check (CRC)	15	
CRC Delimiter	1	Must be recessive
Acknowledge (ACK)	1	Transmitter sends recessive; receiver asserts dominant
ACK Delimiter	1	Must be recessive
End of Frame (EOF)	7	Must be recessive

SocketCAN

- The socketcan package is an implementation of CAN protocols (Controller Area Network) for Linux.
- While there have been other CAN implementations for Linux based on character devices, SocketCAN uses the Berkeley socket API, the Linux network stack and implements the CAN device drivers as network interfaces.
- The CAN socket API has been designed as similar as possible to the TCP/IP protocols to allow programmers, familiar with network programming, to easily learn how to use CAN sockets.

How to use SocketCAN?

Like TCP/IP, first need to open a socket for communicating over a CAN network.

int socket(int *domain*, **int** *type*, **int** *protocol*);

Since SocketCAN implements a new protocol family,
First argument PF_CAN
Second argument SOCK_RAW
Third argument CAN_RAW

socket(PF_CAN, SOCK_RAW, CAN_RAW);

RB-CAN

- After the successful creation of the socket, use the bind(2) system call to bind the socket to a CAN interface
- After binding (CAN_RAW) the socket, we can read(2) and write(2) from/to the socket

The basic CAN frame structure and the sockaddr structure are defined in include/linux/can.h:

```
struct can_frame {
        canid_t can_id; /* 32 bit CAN_ID + EFF/RTR/ERR flags */
        __u8 can_dlc; /* frame payload length in byte (0 .. 8) */
        __u8 __pad; /* padding */
        __u8 __res0; /* reserved / padding */
        __u8 __res1; /* reserved / padding */
        __u8 data[8] __attribute__((aligned(8)));
};
```

The sockaddr_can structure has an interface index like the PF_PACKET socket, that also binds to a specific interface:

```
struct sockaddr_can {
        sa_family_t can_family;
        int can_ifindex;
        union {
                /* transport protocol class address info (e.g. ISOTP) */
                struct { canid_t rx_id, tx_id; } tp;
                /* reserved for future CAN protocols address information */
        } can_addr;
};
```

To determine the interface index an appropriate ioctl() has to be used :

```
int s;
struct sockaddr_can addr;
struct ifreq ifr;
 s = socket(PF_CAN, SOCK_RAW, CAN_RAW);
strcpy(ifr.ifr_name, "can0" );
ioctl(s, SIOCGIFINDEX, &ifr);
addr.can_family = AF_CAN;
addr.can_ifindex = ifr.ifr_ifindex;
bind(s, (struct sockaddr *)&addr, sizeof(addr));
```

Reading CAN frames from a bound CAN_RAW socket consists of reading a struct can_frame:

```
struct can_frame frame;
nbytes = read(s, &frame, sizeof(struct can_frame));
if (nbytes < 0) {
        perror("can raw socket read");
        return 1;
 }

if (nbytes < sizeof(struct can_frame))
{
    fprintf(stderr, "read: incomplete CAN frame\n");
    return 1;
 }
```

Connect-us

Developer Wiki

Open Discussions