CAN is easy to use!

Bernd Westhoff

Author
Bernd Westhoff
Renesas Electronics Europe
Product Marketing
Arcadiastr. 10
DE-40472 Düsseldorf
Tel.: +49-211-6503-1686
bernd.westhoff @renesas.com

Link
www.renesas.eu

Introduction
Today, CAN, extensively used in automotive designs, gained acceptance in industrial automation, medical equipment, vending machines and other embedded system networking applications. Thus, a growing number of engineers are being faced with the task of implementing CAN-based solutions. For networked embedded system applications that require reliable communication and good noise immunity, CAN technology is a robust alternative to EIA-232/-422/-485, I²C and SPI serial interface links. Therefore Renesas provides the RX micro-controllers (MCUs) for CAN-based designs ranging from low-end to high-end flash-based CAN controllers with automated fault detection with correction capabilities.

The CAN interface may be used with an easy accessible CAN API (application programming interface) programming layer. Coupled with the fact that the CAN peripherals execute the low-level protocol work, Renesas employees mean that CAN is very easy to use and enables a fast implementation for your application. This article provides technical introduction in a possible CAN usage case and shows details about CAN's ability to provide low latency, flexible routing of messages and error handling.

CAN controller take care of the low level transmit and receive details enabling the programmer to concentrate on the application. The main focus for the application and programmer is the mailbox, also known as "message box", "buffer" or "slot". When an API is used, the mailbox is the point of bus interaction. The more mailboxes are provided the greater is the design flexibility for the application developer. Also the peripheral will need less runtime for reconfiguration.

Using the mailbox, an application developer may only read/write the CAN-ID, data length code (DLC), and the data fields (and a timestamp if used) of a CAN message, as the application software reads/writes only these fields. Table 1 shows the required interaction with a mailbox when configuring it to communicate. After configuration, the hardware layer takes care of the ▷

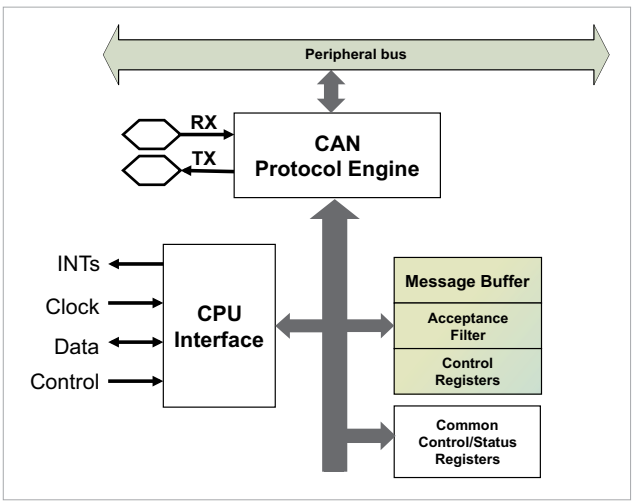


Figure 1: RX600 CAN interface

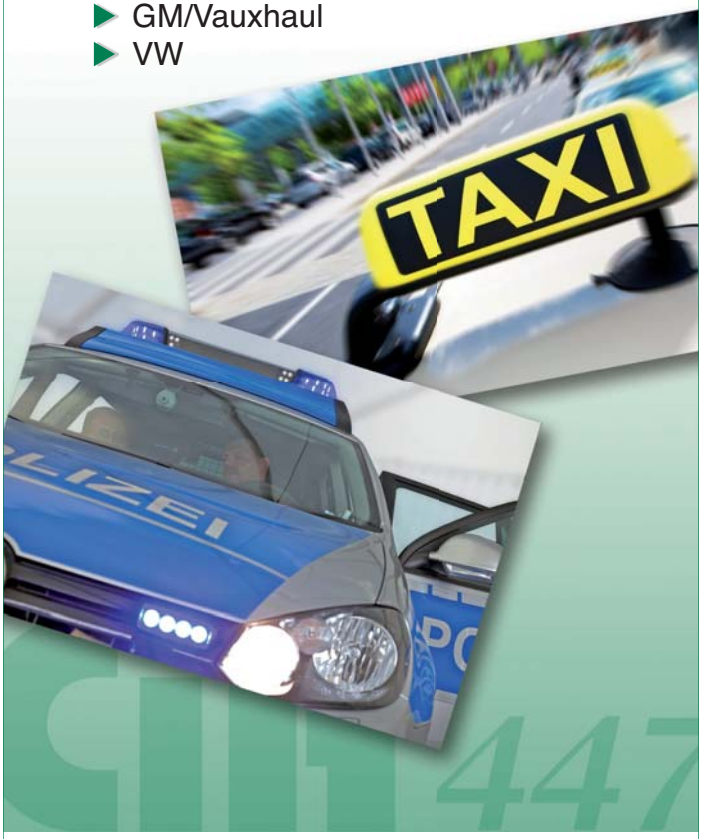
CiA® 447

CAN in Automation

This is the open standard for special purpose vehicles like taxi, emergency vehicles and vehicles for persons with disabilities.

It is supported partly by the following major OEMs in their current cars and will be supported completely in their next generation models:

- Audi
- BMW
- Daimler/Mercedes
- GM/Opel
- GM/Vauxhaul
- VW



For more details please contact the CiA office at service@can-cia.org

www.can-cia.org

Field	Transmit	Receive (mailbox setup)	Receive (get message)
ID	Write	Write	Read
DLC	Write	-	Read
Data	Write	-	Read

Table 1: Required interaction with the mailboxes

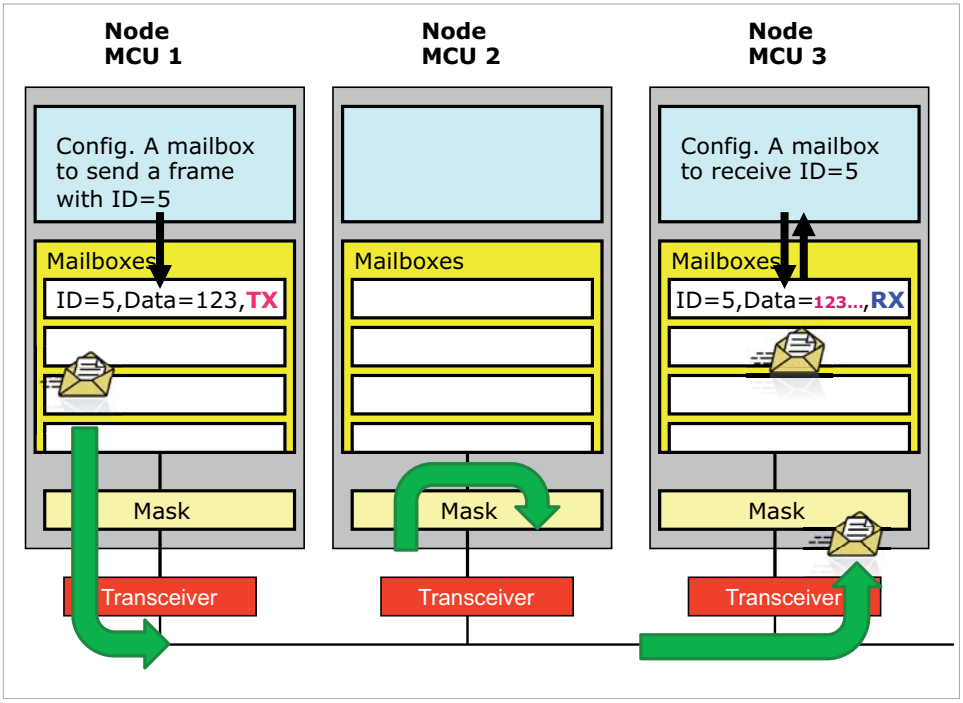


Figure 2: Communication process in a network with three nodes

rest. The transmitter mailbox will manage the low-level transmit details and on the receiver side the mailbox hardware only captures messages with the set CAN-ID. It should be noted that a reconfiguration of the mailbox could also be done during run-time.

Figure 2 illustrates the basic data flow process in a communication network where three CAN nodes are used. Node 1 is sending a message with the CAN-ID of 5. Since node 3 does have a previous setup of their mailboxes to accept messages with the CAN-ID of 5, node 3 accepts the message. Node 2, where no prior setup of mailbox has been done would not accept this message. Renesas's RX MCU does offer up to 32 mailboxes for each CAN interface. Thus, it is capable to receive 32 different messages with different CAN-IDs. It should be noted that all messages are broadcast and received by all participants on the bus. All nodes check the CRC (cyclic redundancy check) sum and then check if any mailbox is configured to receive the sent CAN-ID. If this is true, the received message is copied to the mailbox of a CAN node.

The CAN API

In a layered model, the CAN peripherals control and the CAN API would be placed above the two lower ISO/OSI layers (CAN physical and data link layer). What it required to initialize the CAN peripherals? Setting the CAN operation mode to reset/initialization, enabling the CAN ports, setting the operating mode, setting bit timing and bit-rate and setting the mask registers for acceptance filtering. The procedure for enabling the CAN interrupts followed by setting of a mailbox for send or receive, fetching data from a mailbox etc. should be considered as well. The ▷

The RX600 platform

With up to 128 KiB of RAM and up to 2-MiB embedded flash, the RX600 series offers up to three CAN channels, supporting standard (11-bit CAN-ID) and extended (29-bit CAN-ID) frames. Each CAN module includes 32 mailboxes, of which eight can also be configured as FIFO mailboxes. An acceptance filter mask provides up to eight different masks to be individually set up for each of four mailboxes, which may be enabled and disabled separately. In addition, a 16-bit counter offers a time stamp function. The CAN modules may interact with the RX CPU by using such interrupts as reception complete, transmission complete, receive FIFO, transmit FIFO, and error interrupts. The MCUs have a maximum operating frequency of 100 MHz. In combination with the enhanced RX CPU core architecture, it provides an overall processing performance of 165 DMIPS executing code from the embedded zero-wait state flash. The MCUs also incorporate an on-chip 32-bit multiplier, single-precision floating-point unit (FPU) and a 32-bit enhanced barrel shifter for improved operation processing performance.

The on-chip peripheral functions include timers, four DMA controller channels, Ethernet MAC and up to two USB units. Up to 13 scalable SCIs (UART, SIO and I2C), several A/D and D/A converters and a CRC calculation circuit are also available. The chips with improved EMI/EMS performance comes in 48-pin to 176-pin packages with on-chip flash memory from 64 KiB to 2 MiB and RAM memory from 8 KiB to 128 KiB. The RX600 products are covered by one-tool chain concept.

ifm electronic

7844 x passion!

A mobile dialogue: Robust and simple to operate

One of 7844 products we developed for you with passion: Dialogue module PDM360 NG for mobile vehicles.

The powerful process and dialogue unit of the latest generation has a scratch-resistant high-resolution 7" TFT colour display. 9 backlit function keys with tactile feedback guarantee intuitive operation in the field.
Its robust diecast housing with protection rating IP 67 ensures the highest reliability for mounting inside or outside the cabin. It is easy to program and allows immediate fast and flexible use.

ifm electronic – close to you!

www.ifm.com/gb/mobile

CAN API covers this kind of dealing with the CAN peripherals by doing required settings of related CAN registers.

Following function-call blocks are offered by the CAN API:

- Initialization, port and peripheral control
- Send
- Receive
- Error check

The functions for initializing the CAN peripherals e.g. the function 'R_CAN_Create' is used to create a CAN connection and will call the other 'Init' functions by default. The 'Send' functions are used for set up of a mailbox to transmit and to check that it was sent successfully. The 'Receive' functions are used to set up a mailbox to receive and to retrieve a message from it. The 'Error check' function shows in which CAN state a node is (error active, error passive, or bus-off). This should be used by the application to send messages to the user and restart the application if the node returns from a bus-off state.

Sending first frame

To send first data, the data-frame structure in the memory consisting of CAN-ID, DLC and the actual data should be prepared. Then such a data object should be allocated by defining a structure. Here (as an example) it would be: "my_tx_frame"

```
struct can_std_data_s
{
    uint16 id;
    uint8  dlc;
    uint8  data[8];
};
struct can_std_frame_t
my_tx_frame;
Next, an element of
this structure is de-
fined:
my_tx_frame.id = 0x700;
my_tx_frame.dlc = 2;
my_tx_frame.data[0] =
0x11;
my_tx_frame.data[1] =
0x22;
```

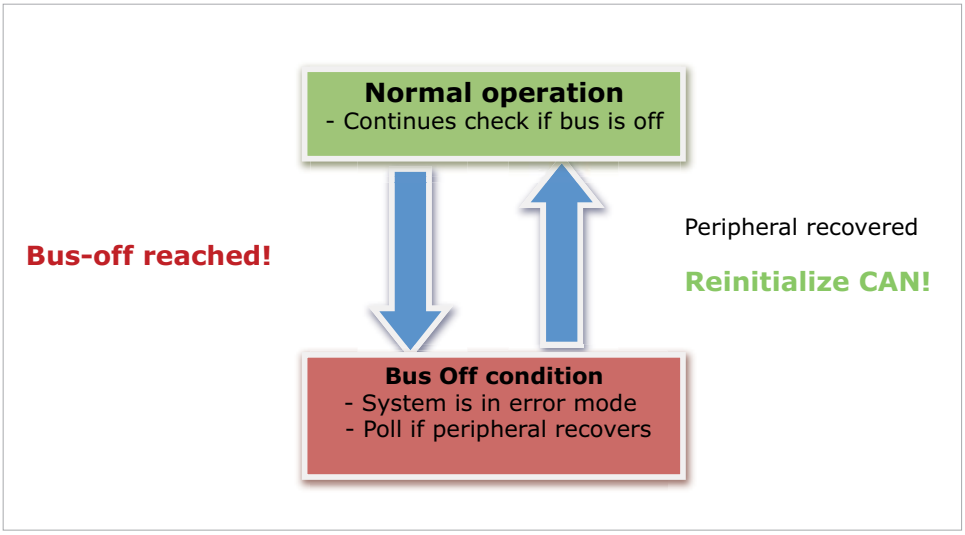


Figure 3: Bus-off handling

Then, the frame using the API function call is sent: `'R_CAN_TxSet(0, mbox_nr, &my_tx_frame, DATA_FRAME);'`. It is not needed to bother with what the function does in detail. For a reference, the API function will:

- Wait for previous frame to finish transmit
- Clear message control register
- Disable interrupts for mailbox
- Set the CAN-ID
- Set the DLC
- Set transmit data
- Transmit a data frame

Receiving first frame

The API function for setting a slot to receive messages would be `'R_CAN_RxSet(ch_nr, mbox_nr, stid, frame_type);'`. In this case the API will (unless 'USE_CAN_POLL' is defined):

- Wait for any previous transmission/reception to complete
- Disable interrupts for mailbox
- Clear mailbox control register
- Set standard CAN-ID for selected mailbox
- Set data frame/remote frame
- Set receive interrupt enabled

Polling versus Interrupts

There is a choice of polling mailboxes for received (and perhaps sent) messages or using the CAN interrupts. Polling may run in background and will not compete with any other interrupts. It is recommended to use this mode, if the interested data is not required at a certain point in time. In case of a certain timing requirement, processing of messages in order or urgent request of data progressing (e.g. emergency message) an interrupt handling would be the better choice.

When polling is used, 'USE_CAN_POLL' is defined in the CAN configuration header (config_r_can_rapi.h), the check transmit function should be called to confirm a transmission. If CAN interrupts are used instead ('USE_CAN_POLL' is not defined) the 'CAN Transmit ISR' is triggered by the peripheral and a mailbox may then be polled by the ISR for a successful transmission. Afterwards the application has to be ▷

CANopen for RX600

The availability of a CANopen source code library for the RX600 has been established by Port (Germany). The library including NMT master and NMT slave functionality contains the services as defined in the CiA 301 version 4.2 and the CiA 302 specifications. The library has been coded in ANSI-C and hardware-specific interfaces have been placed in separate driver packages (also available in C source code). This should facilitate adaptation to different systems. The scope of delivery includes one driver package for one CPU and one CAN controller.

A fitting CANopen design tool (CDT) enables development of CANopen devices and applications. It automatically generates an object dictionary and an initialization function in C code, an EDS (electronic data sheet) and the documentation of the project. Furthermore, it should simplify the configuration of the CANopen library and of the CANopen driver packages. The library provider offers support packages for the standard Renesas' starter kits. The CANopen stack and the CDT are already available.

informed that the message was sent.

To receive data frames using polling the API function 'R_CAN_RxPoll(ch_nr, mbox_nr, frame_p);' has to be called. It returns an OK, if a message is waiting. In this case the message needs to be copied to the RAM, by calling the API function 'R_CAN_RxRead (ch_nr, mbox_nr, frame_p);'. The main application needs to the interaction with the received frame.

To receive data frames using interrupt the following should be applied. When CAN data arrives with the CAN-ID set by the API, the receive ISR triggers. The first item to be done in the ISR, should be to call 'R_CAN_RxPoll (ch_nr, mbox_nr, frame_p);' to check which mailbox caused the interrupt. If there is a new message, the ISR should call 'R_CAN_RxRead (ch_nr, mbox_nr, frame_p);', which will copy the data to the frame structure in RAM indicated by the frame pointer argument. Then, a flag should be set to tell the main application that data has been received.

Bus-off state

To treat a bus-off situation the CAN error interrupt or poll with the 'Check Error' function of the API should be used. It should be done once every cycle in the main routine what state the node is in. If the node has reached bus-off a certain number of times within a certain time period, one may want to send a warning message, light an LED, etc. in an application. If this state is reached, the communication should be stopped and polling continued to see when the peripheral has returned to the normal Error Active state. When the node has recovered, it is important to reinitialize the CAN peripheral and the application to make sure that the slots are in a known state. ◀

Summary

This article provides a basic understanding of sending and receiving CAN frames using the CAN API. Each node on a CAN network may have several buffers or message mailboxes. Each mailbox may be assigned a CAN-ID (identifier) that is either unique or is shared with certain other nodes. All nodes receive the message and perform a filtering operation to determine if the message (and thus its content) is relevant to that particular node. Only the node(s) for which the message is relevant will process it.