RZ/T1 Group

CAN Interface Sample Program

R01AN3109EJ0120
Rev.1.20
Dec. 12, 2018

Summary

This application note explains a sample program for handling communications by using CAN0, which is one of the two channels (CAN0 and CAN1) of the on-chip CAN controller of the MCU mounted on the RZ/T1 evaluation board.

The features of the CAN interface sample program are listed below.

Sending messages:

- Send messages by using the transmission buffers
- Send messages by using the transmission-and-reception FIFO buffers in transmission mode

Receiving messages:

- Receive messages by using the reception buffers
- Receive messages by using the reception FIFO buffers
- Receive messages by using the transmission-and-reception FIFO buffers in reception mode

Self-test modes:

- Self-test mode 0 (external loop-back)
  Send from a transmission buffer and receive at a reception buffer
  Send from a transmission buffer and receive at a reception FIFO buffer
  Send from a transmission buffer and receive at a transmission-and-reception FIFO buffer in reception mode
  Send from a transmission-and-reception FIFO buffer in transmission mode and receive at a transmission-and-reception FIFO buffer in reception mode

- Self-test mode 1 (internal loop-back)
  Send from a transmission buffer and receive at a reception buffer
  Send from a transmission buffer and receive at a reception FIFO buffer
  Send from a transmission buffer and receive at a transmission-and-reception FIFO buffer in reception mode
  Send from a transmission-and-reception FIFO buffer in transmission mode and receive at a transmission-and-reception FIFO buffer in reception mode

The allowed transfer rates:

Three rates; 1 Mbps, 500 Kbps, 125 Kbps can be selected from the menu provided for the program.

Restrictions

The following restrictions apply to this sample program.

(1) The channel in use is fixed to CAN0

(2) The allowed message format is data frames with a standard ID (0x120)

(3) Reception rules are predetermined as follows:
Page number for the reception rule table: 0
Number of reception rules: 1 (described in table 0)
Reception rule ID: data frames with a standard ID (0x120)

(4) Buffers to be used are fixed as follows:
Transmission buffer number: 0
Reception buffer number: 1
Number of the transmission-and-reception FIFO buffer in transmission mode: 0
Number of the transmission buffer to be linked to the transmission-and-reception FIFO buffer in transmission mode: 2
Number of the transmission-and-reception FIFO buffer in reception mode: 1

(5) Other
This sample program does not support the following capabilities.
Transmission abort, transmission by using a transmission queue, function to save transmission history, gateway function, following test functions; standard test mode, listen-only mode, RAM test, inter-channels transfer test, and error detection and correction for RSCAN RAM

Target Devices

RZ/T1 Group

When applying the sample program covered in this application note to another microconrtoller, modify the program according to the specifications for the target microcontroller and conduct an extensive evaluation of the modified program.

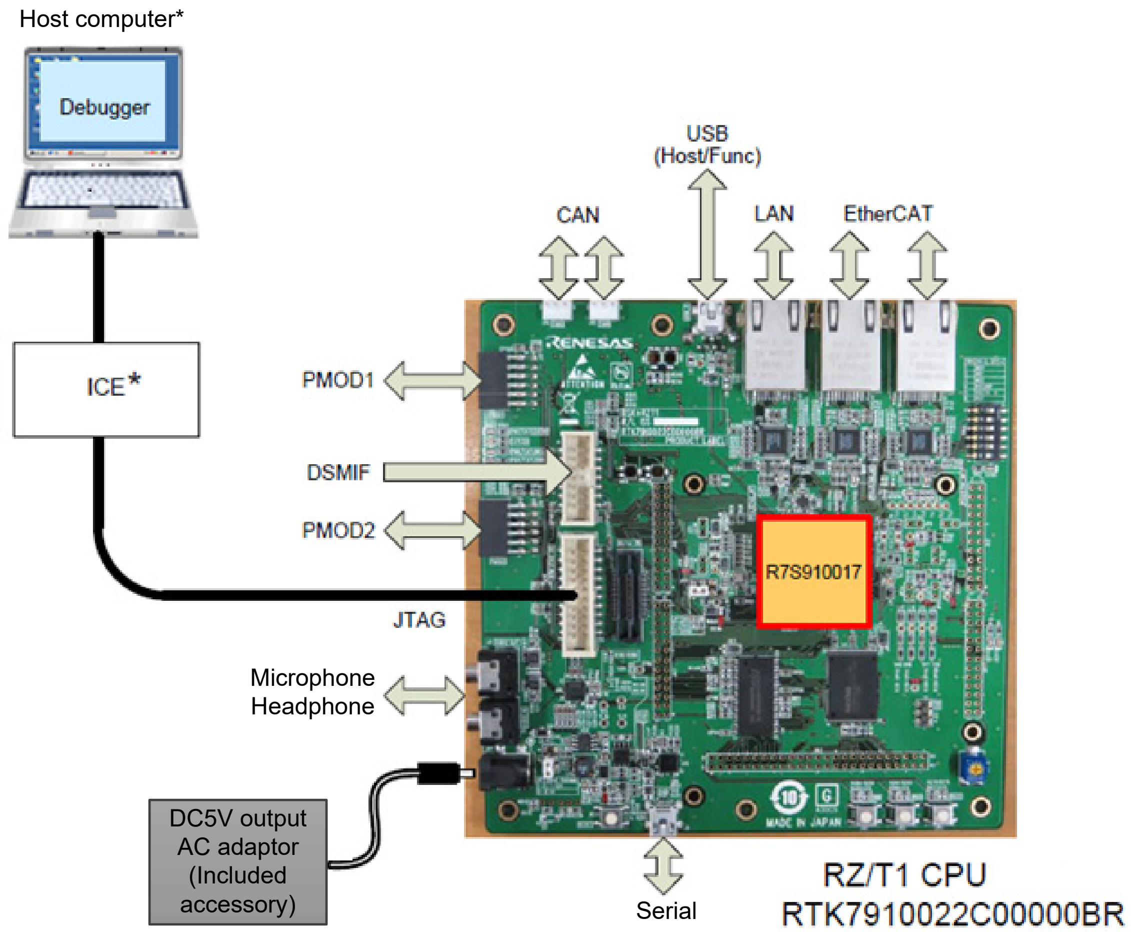
Table of Contents

1. Specifications

Table 1.1 lists the peripheral modules to be used and their applications and Figure 1.1 shows the operating environment for execution of the sample code.

Table 1.1 Peripheral Modules and Applications

Peripheral Modules	Application
CAN interface (RSCAN) CAN0	Transmission and reception of data through the CAN bus by using these LSI chips.
Power consumption reducer	For starting and stopping the RSCAN module (MSTPCRB1)
Interrupt controller (ICUA)	Controlling the following RSCAN interrupt sources: CAN global error (vector 262) CAN0 error (vector 263) CAN1 error (vector 264) CAN reception FIFO (vector 104) CAN0 transmission-and-reception FIFO buffer reception completion (vector 105) CAN0 transmission (vector 106) CAN1 transmission-and-reception FIFO buffer reception completion (vector 107) CAN1 transmission (vector 108)
I/O ports	CAN0: CRXD0 (input) PC6 CAN0: CTXD0 (output) P67 CAN1: CRXD1 (input) PC7 CAN1: CTXD1 (output) P66



Note 1. Indicates the device that the user needs to prepare.

Figure 1.1 Operating Environment

2. Operating Environment

The sample code covered in this application note is for the environment below.

Table 2.1 Operating Environment

Item	Description
Microcontroller	RZ/T1 group
Operating frequency	CPU clock (CPUCLK): 450 MHz
Operating voltage	Power-supply voltage (I/O): 3.3 V
Integrated development environment	• Embedded Workbench® for Arm (version 8.20.2) from IAR Systems • Arm® integrated environment: Arm Development Studio 5 (DS-5™) (version 5.26.2) from Arm • e2studio (version 6.1.0) from Renesas
Operating modes	• SPI boot mode (serial flash memory) • 16-bit bus boot mode (NOR flash memory)
CAN operating modes	Global stop mode Global reset mode Global test mode Global operating mode Channel stop mode Channel reset mode Channel halt mode Channel transfer mode
Settings for communication for the terminal software	• Transfer rate: 115200 bps • Data length: 8 bits • Parity: none • Stop bit length: 1 bit • Flow control: not supported • New line code (reception): CR • New line code (transmission): CR
Board	RZ/T1 evaluation board (RTK7910022C00000BR)
Devices (functions to be used on the board)	Serial interface (USB-mini B connector J8) CAN controller (RSCAN) which conforms the ISO11898-1 specification (for standard frame and extended frame)

3. Related Application Note

The application note related to this application note is listed below for reference.

- Application Note: RZ/T1 Group Initial Settings (R01AN2554EJ)

Note: Settings for registers of the microcontroller which are not stated in this application note are as described in the above application note.

4. Peripheral Modules

Refer to *RZ/T1 Group User's Manual: Hardware* for the functions related to the CAN interface including power-consumption reducer, I/O port, and multi-function pin controller (MPC).

5. Hardware

5.1 Pins

Table 5.1 shows the pins used and their functions.

Table 5.1 Pins Used and Their Functions

Channel	Pin Name	Input/Output	Description
CAN0	CRXD0	Input	CAN0 reception data input pin
	CTXD0	Output	CAN0 transmission data output pin
CAN1	CRXD1	Input	CAN1 reception data input pin
	CTXD1	Output	CAN1 transmission data output pin

5.2 Sample Circuit

Figure 5.1 shows a block diagram.

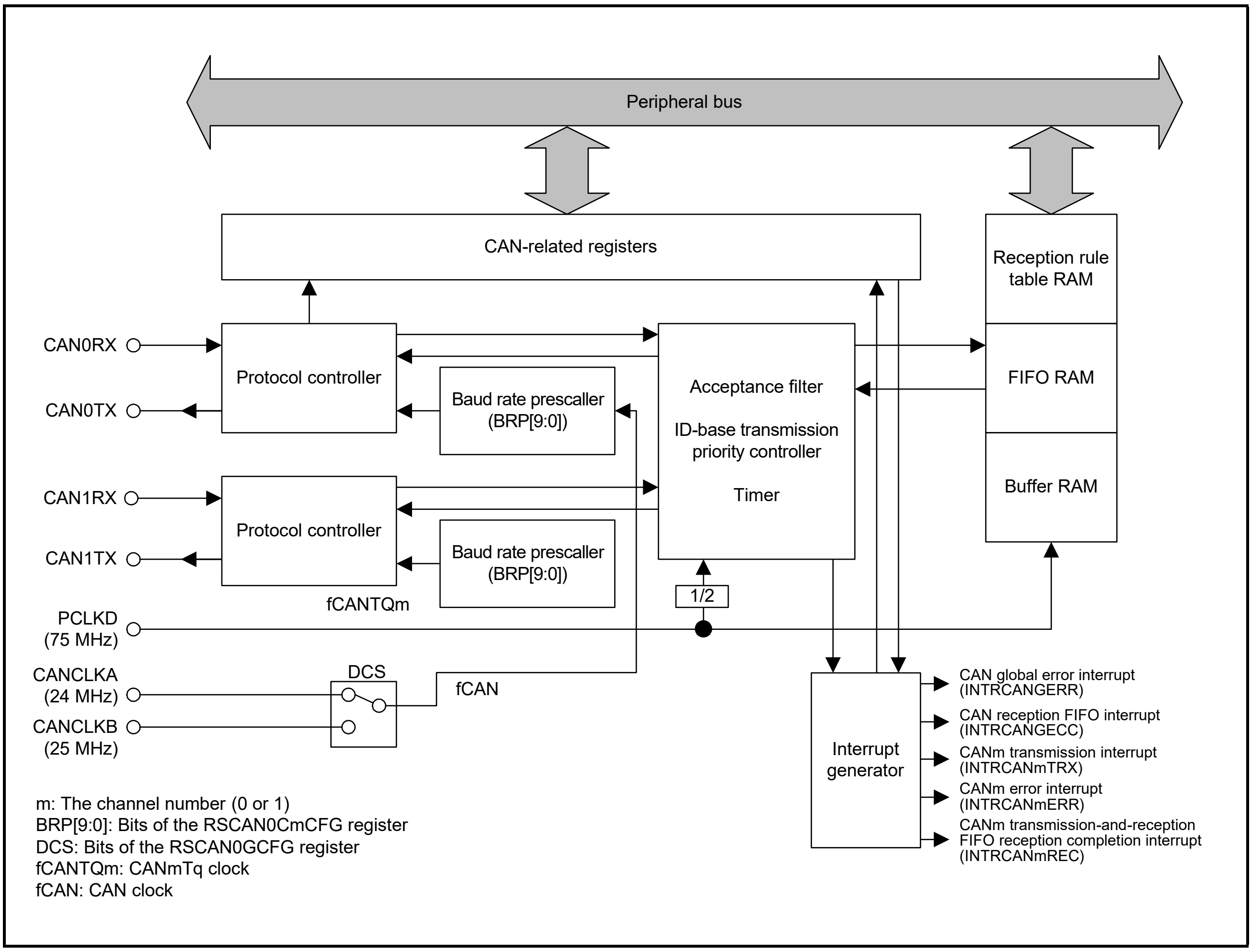


Figure 5.1 RSCAN Block Diagram

6. CAN Configuration

6.1 Configuring the CAN Module

This section describes how to configure the features required for handling communications by using the CAN module ("CAN communications"). Configuration is required before starting or restarting CAN communications after the MCU is reset, any bus error is detected, or a wakeup signal is generated.

Configuration is allowed in the following modes. See Section 6.2, CAN State (Mode) Transitions for details on the CAN states (modes).

- Global reset mode
- Channel reset mode
- Channel halt mode

The following aspects of the CAN module are configured in the initial processing. See the subsequent sections for details on processing for each of the items.

- CAN state (mode)
- Transfer rates
- Global facilities
- Reception rule table
- Buffers
- Global error interrupts
- Channel functions

(1) Configuring the CAN module after the MCU is reset

Initialize the whole CAN module after the MCU is reset.

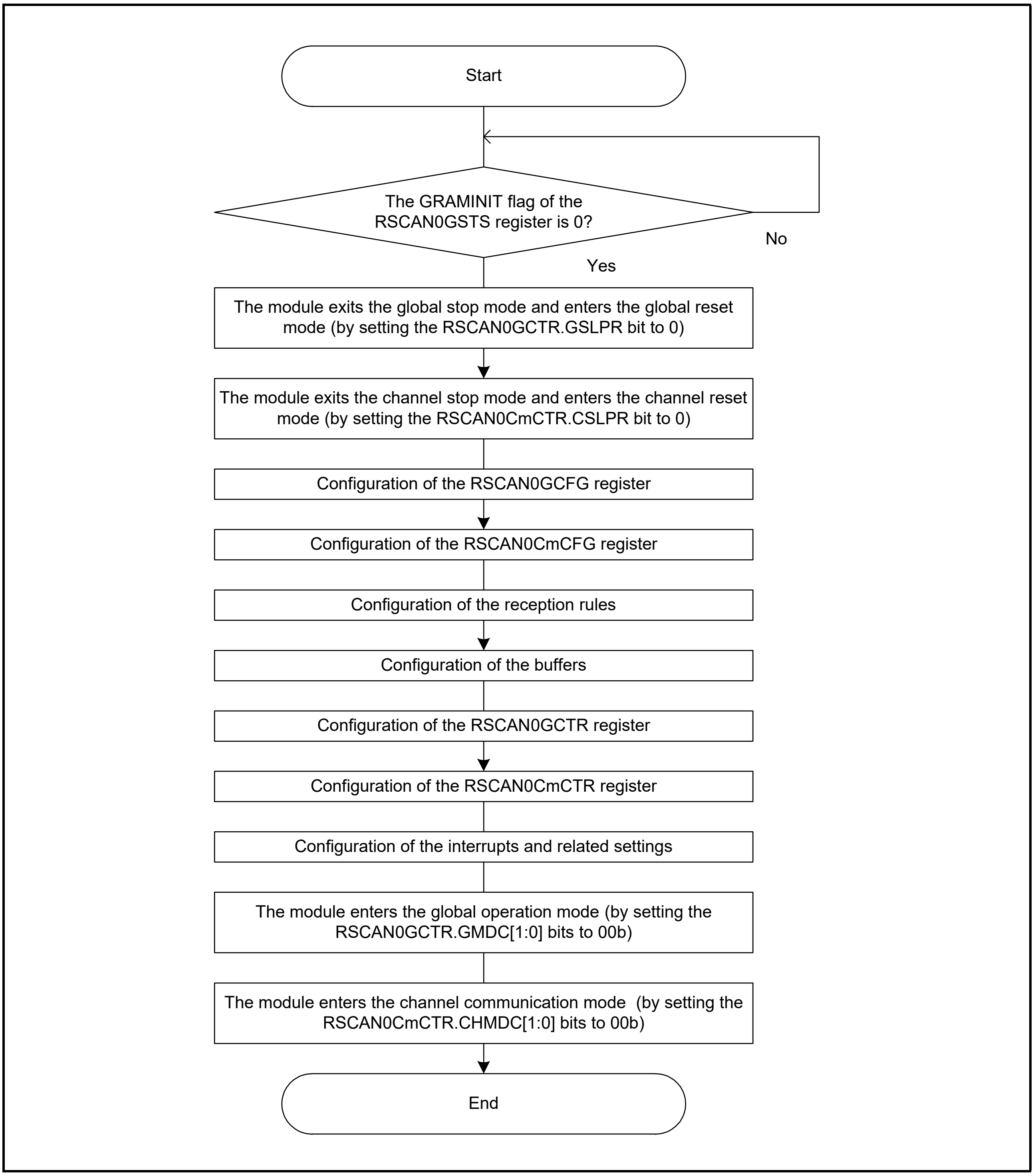


Figure 6.1 Processing of Configuring the CAN Module after the MCU Reset

6.2 CAN State (Mode) Transitions

The CAN module has four global modes to control the state of the module as a whole (its global modes) and four channel modes to control the individual channels (the modes of each channel) as listed below.

Global modes:

- Global stop mode
- Global reset mode
- Global test mode
- Global operation mode

Channel modes:

- Channel stop mode
- Channel reset mode
- Channel halt mode
- Channel transfer mode

6.2.1 Global Modes

These modes involve control of the CAN module as a whole.

Figure 6.2 shows transitions between the global modes. Transition from one global mode to another may also affect the current channel modes.

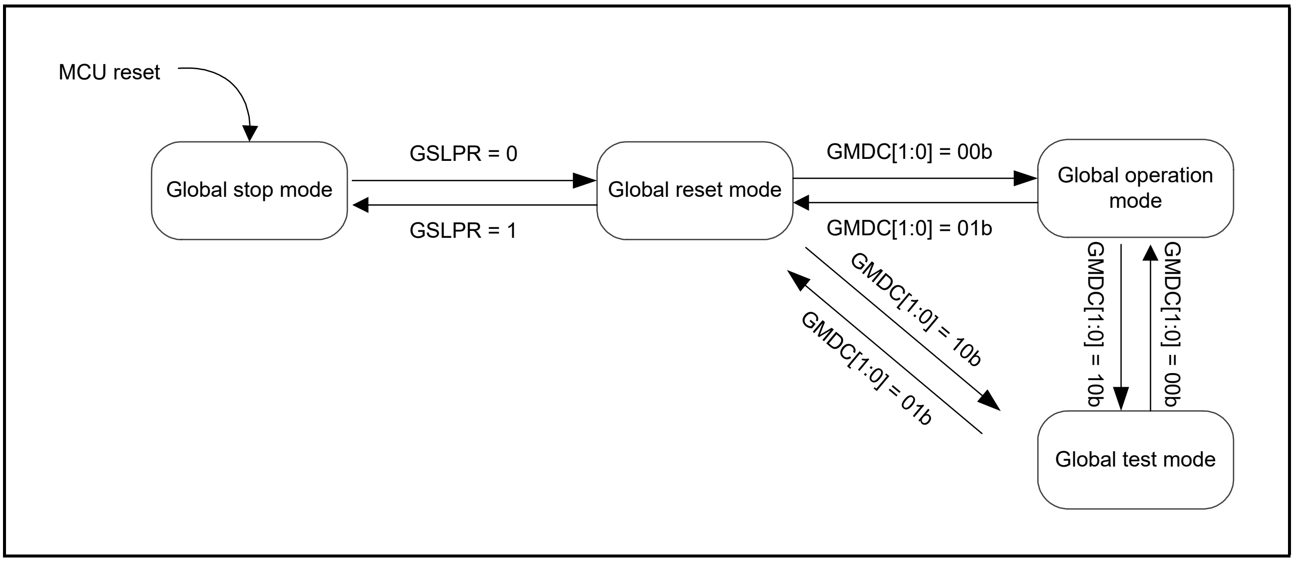


Figure 6.2 Transitions between Global Modes

(1) Global stop mode
The clock for the CAN module stops in this mode so that power consumption can be reduced. Reading from the CAN-related registers is possible but writing to them is not allowed while in this mode. The register values from before the transition are retained.

(2) Global reset mode
The CAN module is configured as a whole in this mode. Making a transition to this mode from another mode initializes part of the registers.

(3) Global test mode
Registers related to test functions are configured in this mode. Making a transition to this mode from another mode stops communication with this module.

(4) Global operation mode
The whole CAN module is operational in this mode. Communications involving the CAN module proceed in this mode.

6.2.2 Channel Modes

The channels of the CAN module are controlled in these modes.

Figure 6.3 shows a transition diagram between channel modes.

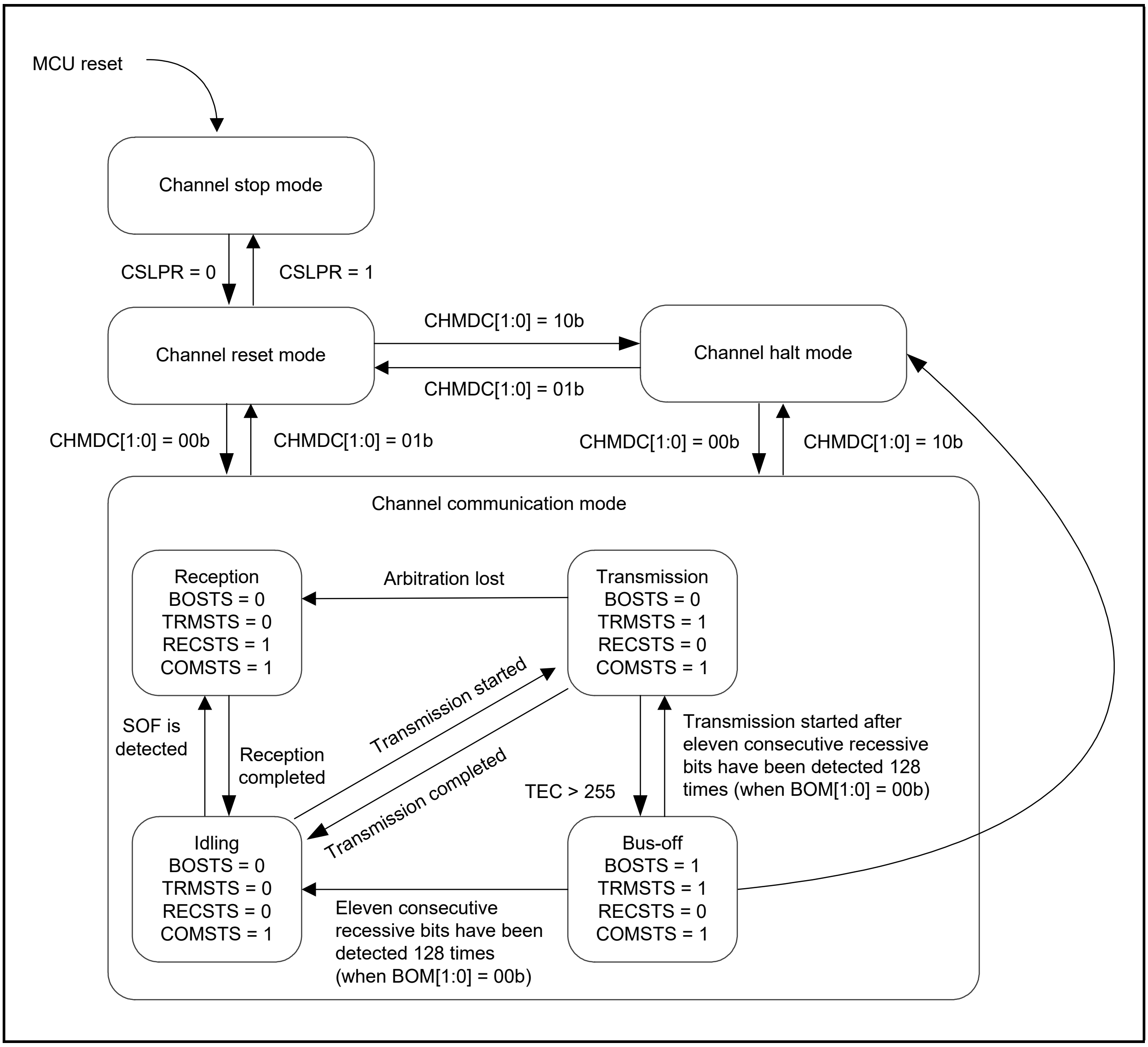


Figure 6.3 Transition Diagram among Channel Modes

(1) Channel stop mode
The clock supplied to the channel currently selected is stopped in this mode. Therefore, power consumption can be reduced. Reading from the CAN-related registers through the concerned channel is possible but writing to them is not allowed while in this mode. The register values from before the transition are retained.

(2) Channel reset mode
The individual channels of the CAN module are configured in this mode. Making a transition to this mode from another mode initializes part of the registers related to the currently selected channel.

(3) Channel halt mode
Registers related to test functions are configured in this mode. Making a transition to this mode from another mode stops communication with this module by using the currently selected channel.

(4) Channel transfer mode
Communications involving the CAN module are done in this mode. The channels of this module are in any of the following states while in this mode:

- Idle state
  Neither reception nor transmission is in progress.
- Reception state
  The channel is receiving a message from another node.
- Transmission state
  The channel is sending a message.
- Bus-off state
  The channel is cut off from the CAN bus.

6.2.3 Changes of Channel Mode Caused by Transitions between Global Modes

Transition from one global mode to another may change the current channel modes. Table 6.1 lists changes of channel mode before and after entering each global mode.

Table 6.1 Changes of Channel Mode Caused by Transitions between Global Modes

Channel Mode Before Entering Any Global Mode	Corresponding Change of Channel Mode After Entering Each Global Mode			
	Global operation	Global test	Global reset	Global stop
Channel transfer	Channel transfer	Channel halt	Channel reset	Transition not allowed
Channel halt	Channel halt	Channel halt	Channel reset	Transition not allowed
Channel reset	Channel reset	Channel reset	Channel reset	Channel stop
Channel stop	Channel stop	Channel stop	Channel stop	Channel stop

6.3 Transfer Rate

The following settings determine the CAN module's transfer rate.

- Bit time setting
- Calculation of the bit rate

6.3.1 CAN Bit Time Setting

In the CAN module of these microcontrollers, one-bit communication frame is divided into three segments as shown in Figure 6.4. Two time segments (TSEG1 and TSEG2) are used to determine the sampling point. The user can set the sampling time by changing the setting values for these segments.

The sampling time is set by using the time quantum (Tq), a fixed unit of time which can be obtained from the clock frequency and the baud rate prescaler value input to the CAN module.

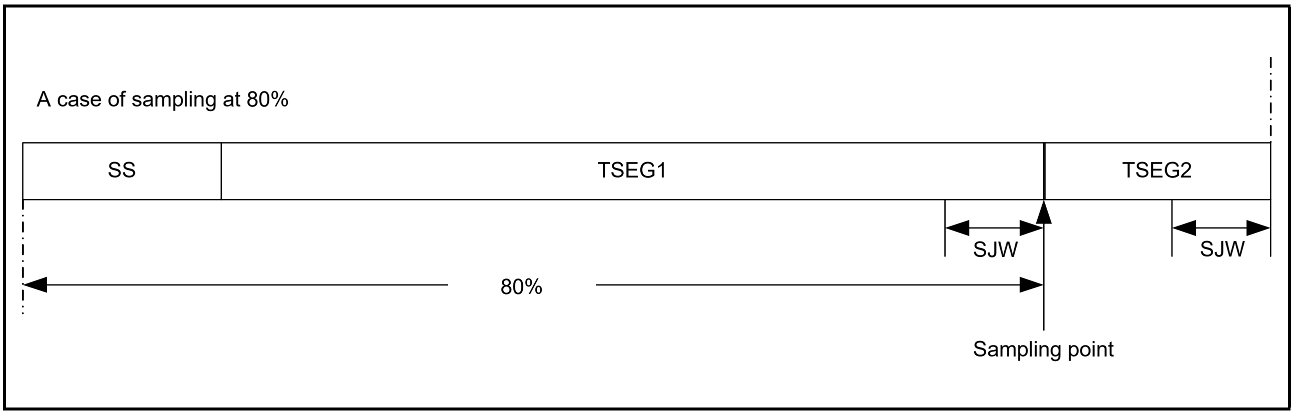


Figure 6.4 Structure of Bit Segments and A Sample Point

Descriptions of the segments in the above figure are given below.

- SS: Synchronization segment
  This segment controls synchronization by monitoring the recessive to dominant edge within the interframe space. The interframe space contains three subfields, which are intermission, suspend transmission, and bus idle. All nodes are able to start transmission of data during the bus idle time.
- TSEG1: Time segment 1
  This segment absorbs the physical delay on the CAN bus. Physical delay on the bus is twice the total of the following three delays: a delay on the CAN bus, a delay in the input comparator, and a delay in the output driver.
- TSEG2: Time segment 2
  This segment compensates phase errors due to clock frequency errors.
- SJW: Resynchronization jump width
  This is a length to extend or reduce a time segment to compensate for an error in phase.

(1) Conditions for setting bit time

The ranges and limitations for the setting values for each segment are as follows.

Ranges of setting values:

- SS : fixed to 1Tq
- TSEG1 : from 4 to 16 Tq
- TSEG2 : from 2 to 8 Tq
- SJW : from 1 to 4 Tq
- SS + TSEG1 + TSEG2 : from 8 to 25 Tq

Limitations on the settings:

- TSEG1 > TSEG2 ≥ SJW (with the added condition that when SJW = 1, TSEG2 ≥ 2)

6.3.2 Calculating Transfer Rates

The transfer rate is determined by the CAN clock (f_{CAN}) which is a clock source for the CAN module, the baud rate prescaler value, and Tq count per bit time. Either one of the following clocks can be used as f_{CAN}: the clock obtained by dividing the CPU/peripheral hardware clock by 2 or the X1 clock.

Table 6.2 and Table 6.3 are examples of basic transfer rates and bit times.

Table 6.2 Examples of Basic Transfer Rates

Transfer Rate \ fCAN	40 MHz	32 MHz	24 MHz	16 MHz	8 MHz
1 Mbps	8 Tq (5) 20 Tq (2)	8 Tq (4) 16 Tq (2)	8 Tq (3) 12 Tq (2) 24 Tq (1)	8 Tq (2) 16 Tq (1)	8 Tq (1)
500 Kbps	8 Tq (10) 20 Tq (4)	8 Tq (8) 16 Tq (4)	8 Tq (6) 12 Tq (4) 24 Tq (2)	8 Tq (4) 16 Tq (2)	8 Tq (2) 16 Tq (1)
250 Kbps	8 Tq (20) 20 Tq (8)	8 Tq (16) 16 Tq (8)	8 Tq (12) 12 Tq (8) 24 Tq (4)	8 Tq (8) 16 Tq (4)	8 Tq (4) 16 Tq (2)
125 Kbps	8 Tq (40) 20 Tq (16)	8 Tq (32) 16 Tq (16)	8 Tq (24) 12 Tq (16) 24 Tq (8)	8 Tq (16) 16 Tq (8)	8 Tq (8) 16 Tq (4)

Note: Figures in parentheses indicate baud rate prescaler values.

Table 6.3 Bit Time Example

1 Bit	Setting Value (Tq)				Sampling Point (%) * See Figure 6.4
	SS	TSEG1	TSEG2	SJW	
8 Tq	1	4	3	1	62.50
	1	5	2	1	75.00
10 Tq	1	6	3	1	70.00
	1	7	2	1	80.00
16 Tq	1	10	5	1	68.75
	1	11	4	1	75.00
20 Tq	1	12	7	1	65.00
	1	13	6	1	70.00

6.3.3 Procedure for Setting CAN Bit Time and Transfer Rates

Figure 6.5 shows the procedure for setting the CAN bit time and transfer rate. Make these settings during CAN configuration.

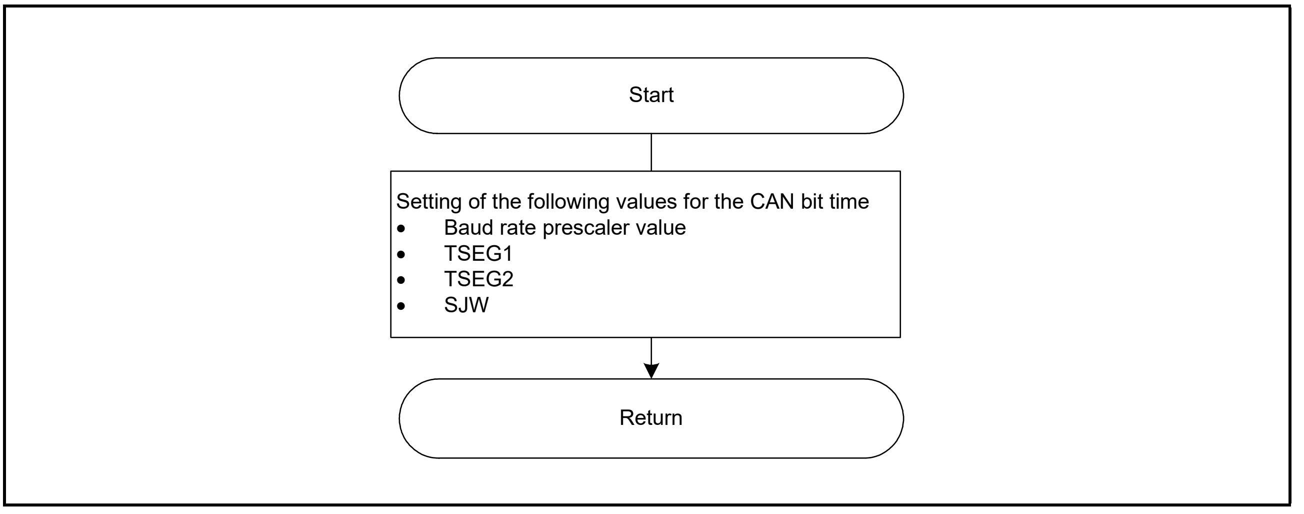


Figure 6.5 Procedure for Setting CAN Bit Time and Transfer Rates

6.4 Global Facilities

The following functionalities are configured for the CAN module as a whole. These are the settings common to both channels.

- Transmission priority
- DLC checking
- DLC replacement
- Mirroring function
- CAN clock source
- Timestamp clock

6.4.1 Transmission Priority

This function sets priorities for transmission requests issued from two or more transmission buffers of the same channel. The same priority setting applies to both channels; that is, priority cannot be configured per channel. Priority is judged based on the following two options:

- ID-base
  Messages stored in the buffers are transmitted in the order based on their IDs, according to the CAN bus arbitration method. This option applies to messages in transmission buffers and transmission-and-reception FIFO buffers which are set in transmission mode. The oldest message is highest in the order of priority for a reception-and-transmission FIFO buffer. When a message is being transmitted from a transmission-and-reception FIFO buffer, the next message in the buffer is judged to have the next highest priority. When the same message ID is set for two or three of the buffers, the buffer with the lower or lowest number takes priority.
- Based on transmission buffer numbers
  The message in the transmission buffer with the lowest number among the transmission buffers having a transmission request takes priority. When the transmission-and-reception FIFO buffer is linked to transmission buffers, priority is judged according to the buffer numbers of the buffers of the link destinations.

When messages are to be resent after arbitration losses or any errors, the priority order is judged again regardless of the selection of the priority setting rules.

6.4.2 DLC Checking

Enable or disable the DLC (data length code) checking function during configuration.

When this function is enabled, DLC filtering is applied to the messages that have passed through the acceptance filter. When this function is disabled, DLC filtering is not applied to those.

When DLC filtering is applied to a received message that is equal to or larger than the DLC value specified in the reception rule, it passes through the filter. Meanwhile, when DLC filtering is applied to a received message that is smaller than the DLC value specified in the reception rule, it does not pass through the filter. In this case, the message will not be stored in the reception buffer or transmission-and-reception FIFO buffer, which means a DLC error has occurred.

6.4.3 DLC Replacement

Enable or disable the DLC replacement function during configuration. This function is enabled only when DLC checking is enabled.

If a message has passed through DLC filtering while DLC replacement is enabled, the number of bytes corresponding to the DLC value in the reception rule table instead of the DLC value of the received message is stored in the reception buffer. If the size of the message exceeds the replacement value in the table, H'00 is stored in the corresponding bytes in the reception buffer.

If a message passed DLC filtering while DLC replacement is disabled, the DLC value of the received message is stored in the reception buffer. Here, all data bytes of the received message are stored in the buffer.

6.4.4 Mirroring Function

Enable or disable the mirroring function during configuration. When this function is enabled, a CAN node is able to receive messages sent by itself.

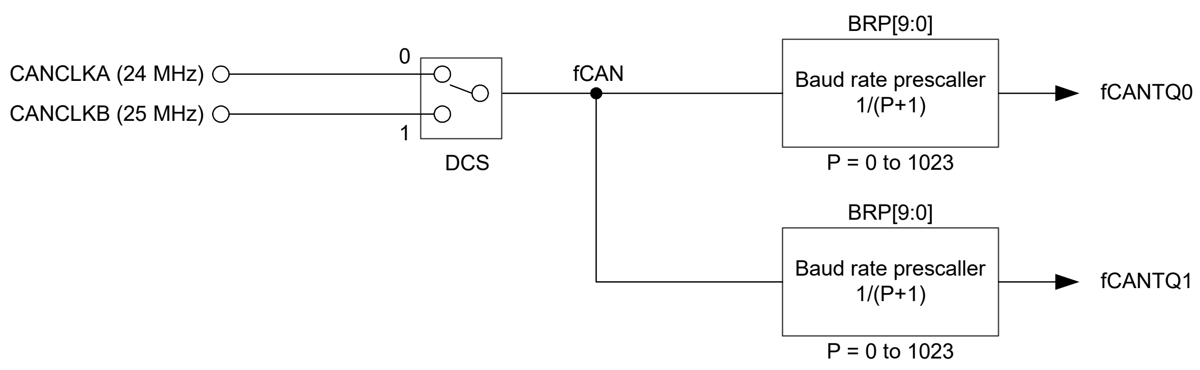
When a CAN node receives messages sent from a different node, the reception rule without mirroring function is used for processing data. When a CAN node receives messages sent by its own node, the reception rule with mirroring function is used for processing data.

6.4.5 CAN Clock Source

The CAN clock (fCAN) is configured as a clock source for the CAN module. The following two clocks can be used as the source.

- Clock obtained by frequency-dividing the CPU/peripheral hardware clock by 2
- X1 clock

Figure 6.6 illustrates the CAN clock generator.



$$\text{Transfer rate} = \frac{\text{fCAN}}{\text{Baud rate prescaler division ratio} \times (\text{Tq count of 1 bit time})}$$

m = 0, 1: Channel number
BRP[9:0] Bits of the RSCAN0CmCFG register
DCS: Bits of the RSCAN0GCFG register
fCAN: CAN clock

Figure 6.6 CAN Clock Generator

6.4.6 Timestamp Clock

The settings of the clock source and division ratios used for the timestamp counter are described below.

The timestamp is a 16-bit free-running counter clock used for recording message receiving time. The value of the timestamp counter is fetched at the StartOfFrame*1 timing of a message and then stored in a reception buffer or a FIFO buffer together with the message ID and its data.

The clock used for the timestamp counter can be selected from the following:

- Clock obtained by frequency-dividing the CPU/peripheral hardware clock by 2
- CANi bit time clock

Note 1. StartOfFrame: A field indicating a start of a frame.

Figure 6.7 is a block diagram of the timestamp function.

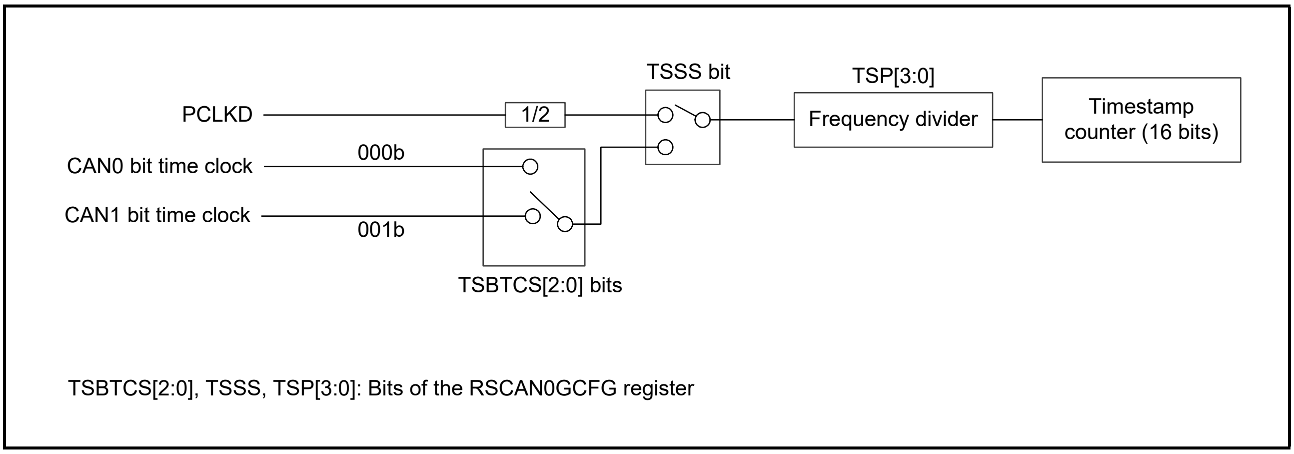


Figure 6.7 Timestamp Function

6.4.7 Global Facilities

Figure 6.8 shows the procedure for setting the global facilities. Make these settings during CAN configuration.

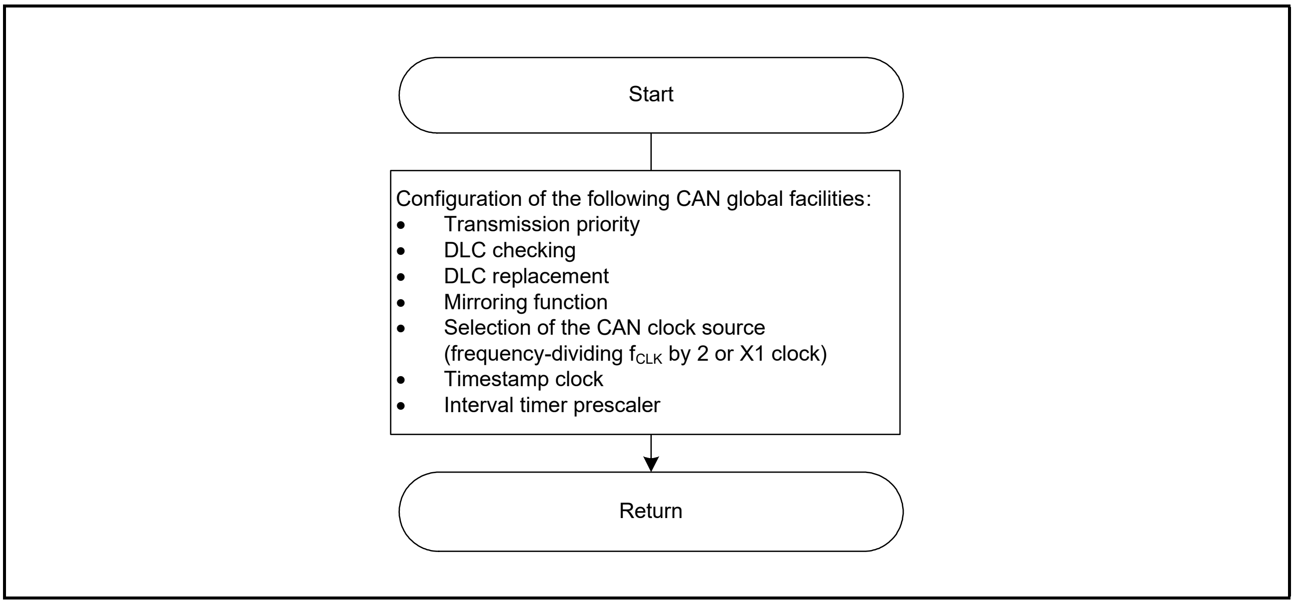


Figure 6.8 Setting Procedure for Global Facilities

6.5 Reception Rule Table

Messages received by the CAN module are filtered based on the reception rule table.

According to the settings in the table, received data may be processed by the acceptance filter, DLC filter, routing, labeling, or mirroring functions, before being stored in the specified buffer.

The reception rule table includes the following settings:

- Number of the reception rules
- Setting of the IDE, RTR, and ID bits
- Whether to apply reception rules or not
- Weather to mask the IDE, RTR, and ID bits or not
- DLC checking
- Labeling for reception rules
- Buffers for storing data

6.5.1 Number of Reception Rules

Number of reception rules are set for each channel. Up to 16 rules can be registered in one page.

In filtering process, received messages are checked with the reception rules from the minimum rule number. Filtering stops when the bits for the target received messages match all the reception rules or when checking of all bits ended without having any match with the reception rules. If no reception rules matched, the message is not stored in the reception buffers or FIFO buffers.

6.5.2 Setting of the IDE, RTR, and ID Bits

Setting of the ID format (standard or extended), the frame format (data or remote), and the reception ID in each received message is required.

6.5.3 Processing Using Reception Rules

Setting the GAFLLB bit of the RSCAN0GAFLIDj register to 0 allows data processing by using the reception rules for the messages received from a different CAN node.

Setting the GAFLLB bit of the RSCAN0GAFLIDj register to 1 while mirroring function is enabled allows data processing by using the reception rules for the messages received from its own node.

6.5.4 Settings to Mask the IDE, RTR, and ID Bits

The values set in the IDE mask, RTR mask, and ID mask bits are used to mask the values set in the corresponding IDE, RTR, and ID bits. The bits not masked by these bits are enabled when acceptance filter is applied.

6.5.5 Values for DLC Checking

The DLC values set in the reception rule table are compared with that in the received message when DLC checking is enabled.

6.5.6 Reception Rule Labeling

Users can set a 12-bit information label for messages which have passed through the filter. The label is attached to the message when it is stored in the buffer. The label may be set as described and labeling may also be handled under program control. For example, the channel through which messages with the same ID in a reception FIFO buffer were received can be identified by their labels by setting the channel number in the label.

6.5.7 Buffer for Storing Messages

Messages passed through the DLC filtering are stored in the buffers specified from the followings.

- Reception buffer n (a single buffer is designated for a single reception rule)
- Reception FIFO buffer m
- Transmission-and-reception FIFO buffer k in reception mode

Up to two buffers are selected for a single reception rule but only one buffer can be designated for storing the messages.

6.5.8 Usage Example of Reception Rule

The following are usage examples of the reception rules.

[Example 1]

Required settings for the registers for receiving the message with the following conditions are given below.

- ID format : standard ID
- Message format : data frames
- Mirroring function : disabled (receiving messages from a different CAN node)
- Reception IDs : 120h, 121h, 122h, 123h
- DLC : the DLC value of the reception message is equal to or greater than 6
- Labeling : 010h
- Destination buffers : reception buffer 3 and reception FIFO buffers 0 and 1

Reception Rule ID Register (RSCAN0GAFLIDj)

Target Reception ID	GAFLIDE	GAFLRTR	GAFLLB	GAFLID[28:0]
120h	0	0	0	B'- --- ---- ---- ---- -001 0010 0000
121h				B'- --- ---- ---- ---- -001 0010 0001
122h				B'- --- ---- ---- ---- -001 0010 0010
123h				B'- --- ---- ---- ---- -001 0010 0011

Reception Rule Mask Register (RSCAN0GAFLMj)

GAFLIDEM	GAFLRTRM	GAFLIDM[28:0]
1	1	B'0 0000 0000 0000 0000 0111 1111 1100

Reception Rule Pointer 0 Register (RSCAN0GAFLP0j)

GAFLDLC[3:0]	GAFLPTR[11:0]	GAFLRMV	GAFLRMDP[6:0]
6	010h	1	3

Reception Rule Pointer1 Register (RSCAN0GAFLP1j)

RSCAN0GAFLP1j[17:0]	GAFLFDPr[7:0]
B'00 0000 0000 0000 0000	B'0000 0011

[Example 2]

Required settings for the registers for receiving the message with the following conditions are given below.

- ID format : standard ID
- Message format : remote frames, data frames
- Mirroring function : disabled (receiving messages from a different CAN node)
- Reception ID : 130h
- DLC : DLC is disabled
- Labeling : 130h
- Destination buffers : reception FIFO buffer 0, transmission-and-reception FIFO buffer 0

Reception Rule ID Register (RSCAN0GAFLIDj)

Target Reception ID	GAFLIDE	GAFLRTR	GAFLLB	GAFLID[28:0]
130h (data frames)	0	0	0	B'- --- ---- ---- ---- -001 0011 0000
130h (remote frames)	0	1	0	B'- --- ---- ---- ---- -001 0011 0000

Reception Rule Mask Register (RSCAN0GAFLMj)

GAFLIDEM	GAFLRTRM	GAFLIDM[28:0]
1	0	B'0 0000 0000 0000 0000 0111 1111 1111

Reception Rule Pointer 0 Register (RSCAN0GAFLP0j)

GAFLDLC[3:0]	GAFLPTR[11:0]	GAFLRMV	GAFLRMDP[6:0]
0	130h	0	0

Reception Rule Pointer1 Register (RSCAN0GAFLP1j)

RSCAN0GAFLP1j[17:0]	GAFLFDPr[7:0]
B'00 0000 0000 0000 0001	B'0000 0001

6.5.9 Procedure for Setting the Reception Rule Table

Figure 6.9 shows the setting flow of the reception rule table. Make these settings during CAN configuration.

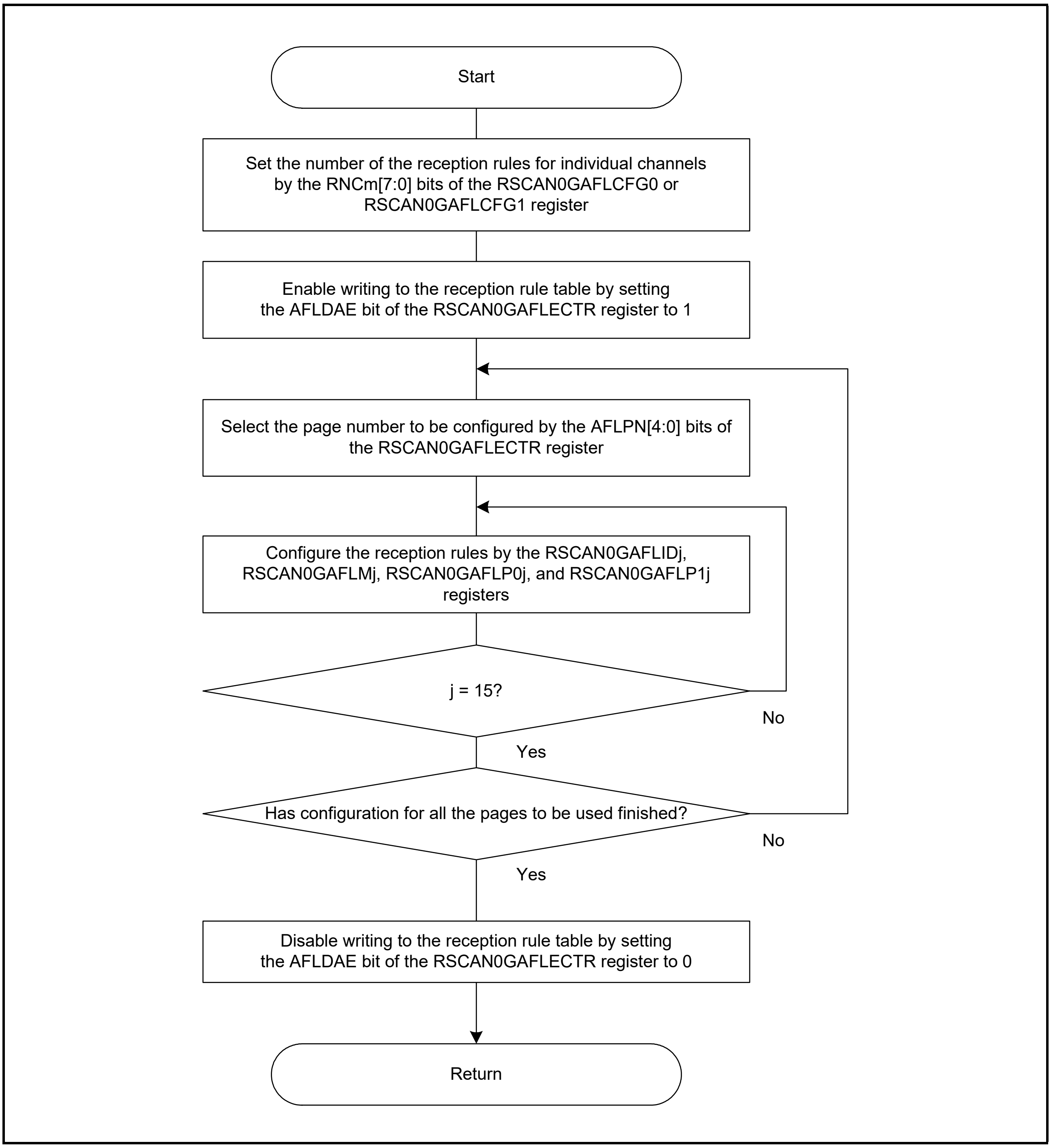


Figure 6.9 Setting Procedure for the Reception Rule Table

6.6 Buffers and FIFO Buffers

Configuration of the following buffers and FIFO buffers are required for sending and receiving messages.

- Reception buffers
- Reception FIFO buffers
- Transmission-and-reception FIFO buffers
- Transmission buffers
- Transmission history buffers

Figure 6.10 shows a buffer structure.

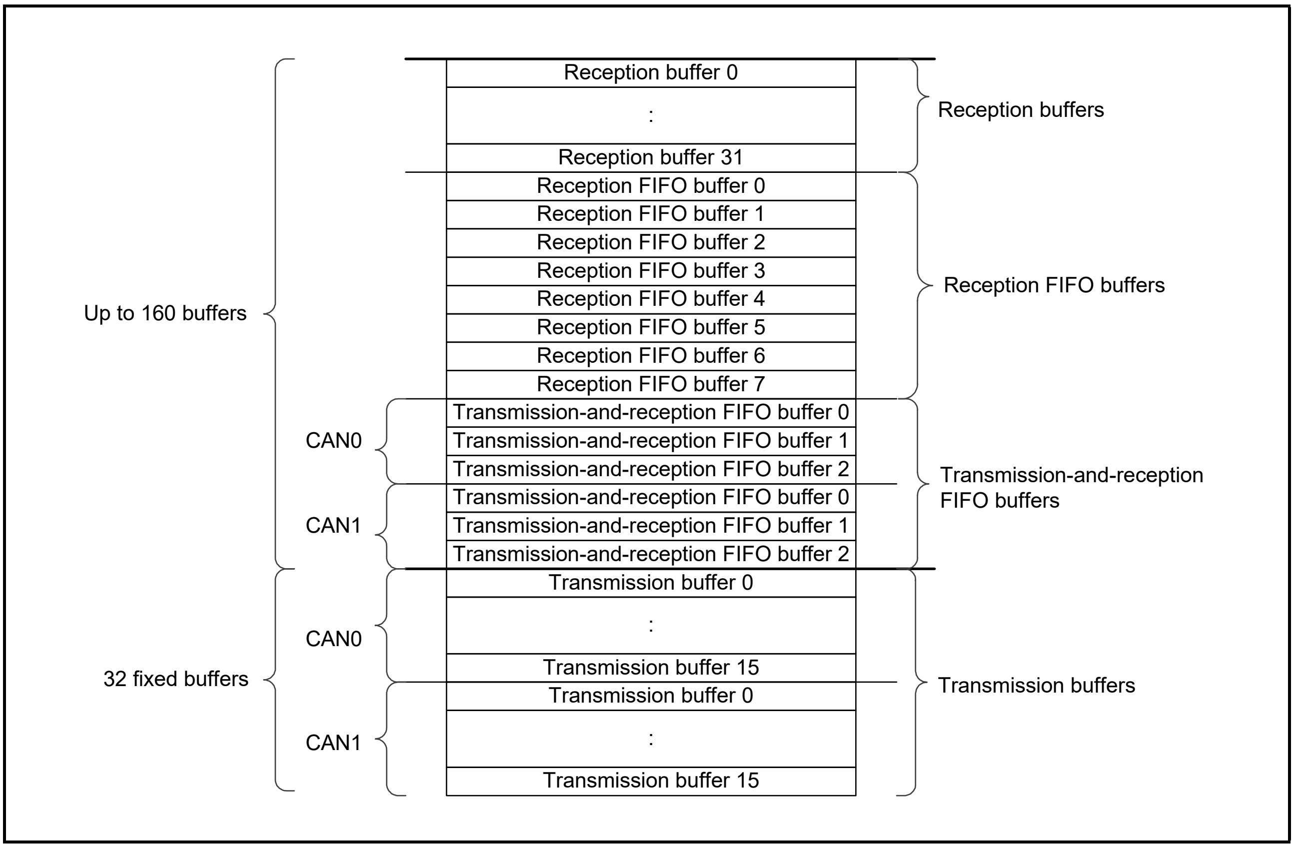


Figure 6.10 Buffer Structure

6.6.1 Reception Buffer

The number of buffers to be used as reception buffers is specified in the range from 0 to 31. No reception buffer can be used if the specified number of the reception buffers is 0. Interrupt settings are not needed as there are no interrupts related to reception buffers.

6.6.2 Reception FIFO Buffer

The following settings are required to use the reception FIFO buffers:

- The number of buffers
- Enabling and disabling of interrupts and setting of interrupt sources

(1) The number of buffers
The number of buffers to be used as reception FIFO buffers is specified in the range from 0 to 8.
If no reception FIFO buffers are to be used, set the reception FIFO buffer enable bit (RFE) and the reception FIFO buffer depth configuration bits (RFDC[2:0]) of the reception FIFO buffer configuration/control register (RSCAN0RFCCx) to 0 and 000, respectively.

(2) Enabling and disabling interrupts and setting interrupt sources
Enable or disable the reception FIFO interrupts during configuration. When the interrupt is enabled, the interrupt sources are selected from the following.

- An interrupt is generated (the RFIM bit of the RSCAN0RFCCx register is set to 0) when the conditions set in the RFIGCV[2:0] bits of the reception FIFO buffer configuration/control register (RSCAN0RFCCx) met.
- An interrupt is generated (the RFIM bit of the RSCAN0RFCCx register is set to 1) every time message reception completes.

6.6.3 Transmission-and-Reception FIFO Buffer

The following settings are required to use the transmission-and-reception FIFO buffers.

- The number of the buffers
- Enabling and disabling of interrupts and setting of interrupt sources
- Mode of the transmission-and-reception FIFO buffer
- Interval timer counter (when used in transmission mode)
- Transmission buffer link (when used in transmission mode)

(1) The number of buffers

The number of buffers to be used as the transmission-and-reception FIFO buffers is specified in the range from 0 to 5, up to three for each channel (CAN0: 0 to 2, CAN1: 3 to 5).

If no transmission-and-reception FIFO buffers are to be used, set the transmission-and-reception FIFO buffer enable bit (CFE) and the transmission-and-reception FIFO buffer depth configuration bits (CFDC[2:0]) of the transmission-and-reception FIFO buffer configuration/control register (RSCAN0CFCCk) to 0 and 000, respectively.

(2) Enabling and disabling interrupts and setting interrupt sources

Transmission-and-reception FIFO interrupts are enabled and disabled. The interrupt sources for each mode (transmission or reception) are shown below.

Mode	CFIM Bit	Interrupt Source
Reception	0	A FIFO reception interrupt request is issued when the number of received messages reached the value set in the CFIGCV[2:0] bits.
	1	A FIFO reception interrupt request is issued every time message reception completes.
Transmission	0	A FIFO transmission interrupt request is issued when the buffer becomes empty after completion of message transmission.
	1	A FIFO transmission interrupt request is issued every time message transmission completes.

Generation of a transmission-and-reception FIFO transmission interrupt triggers generation of the following CANi transmission interrupt sources:

- CANi transmission completion interrupt
- CANi transmission abort interrupt
- CANi transmission-and-reception FIFO transmission completion interrupt
- CANi transmission history interrupt

(3) Mode of the transmission-and-reception FIFO buffer

Transmission-and-reception FIFO buffers are used in either the reception mode or the transmission mode.

- Reception mode
  In this mode, the buffer serves as a reception FIFO buffer.
- Transmission mode
  In this mode, the buffer serves as a transmission FIFO buffer.

(4) Interval timer counter (when used in transmission mode)
The source for the counter and transmission interval are specified. The counter is enabled only in transmission mode.

(5) Transmission buffer link (when used in transmission mode)
A transmission-and-reception FIFO buffer is linked to a transmission buffer. Linking is enabled only in transmission mode.

6.6.4 Transmission Buffers

Enable or disable transmission completion interrupts for each transmission buffer during configuration. A single channel contains sixteen transmission buffers which are used as transmission buffers or the buffers linked to transmission-and-reception FIFO buffers in transmission mode.

Generation of a transmission completion interrupt triggers generation of the following CANi transmission interrupt sources:

- CANi transmission completion interrupt
- CANi transmission abort interrupt
- CANi transmission-and-reception FIFO transmission completion interrupt
- CANi transmission history interrupt

6.6.5 Transmission History Buffers

The following settings are required to use a transmission history buffer. A single channel contains one transmission history buffer which can hold history data on sixteen transmissions.

- Buffers for which transmission histories are to be stored are selectable
- Enabling and disabling of interrupts and setting of interrupt sources

(1) Buffers for which transmission histories are to be stored
The buffers (transmission source) for which transmission history data will be stored in the transmission history buffer are selected from the following two options. It is also possible to select whether or not to store the transmission history at each transmission.
- Transmission-and-reception FIFO buffers
- Transmission buffers and transmission-and-reception FIFO buffers

(2) Enabling and disabling interrupts and setting interrupt sources
Enable or disable transmission history interrupts during configuration. The interrupt sources are shown below.
- CANi transmission completion interrupt
- CANi transmission abort interrupt
- CANi transmission-and-reception FIFO transmission completion interrupt
- CANi transmission history interrupt

6.6.6 Procedures for Setting Buffers

Figure 6.11 shows a procedure for setting the reception buffers and reception FIFO buffers, and Figure 6.12 shows a procedure for setting the transmission-and-reception FIFO buffers, transmission buffers, and transmission history buffers.

Make these settings during CAN configuration.

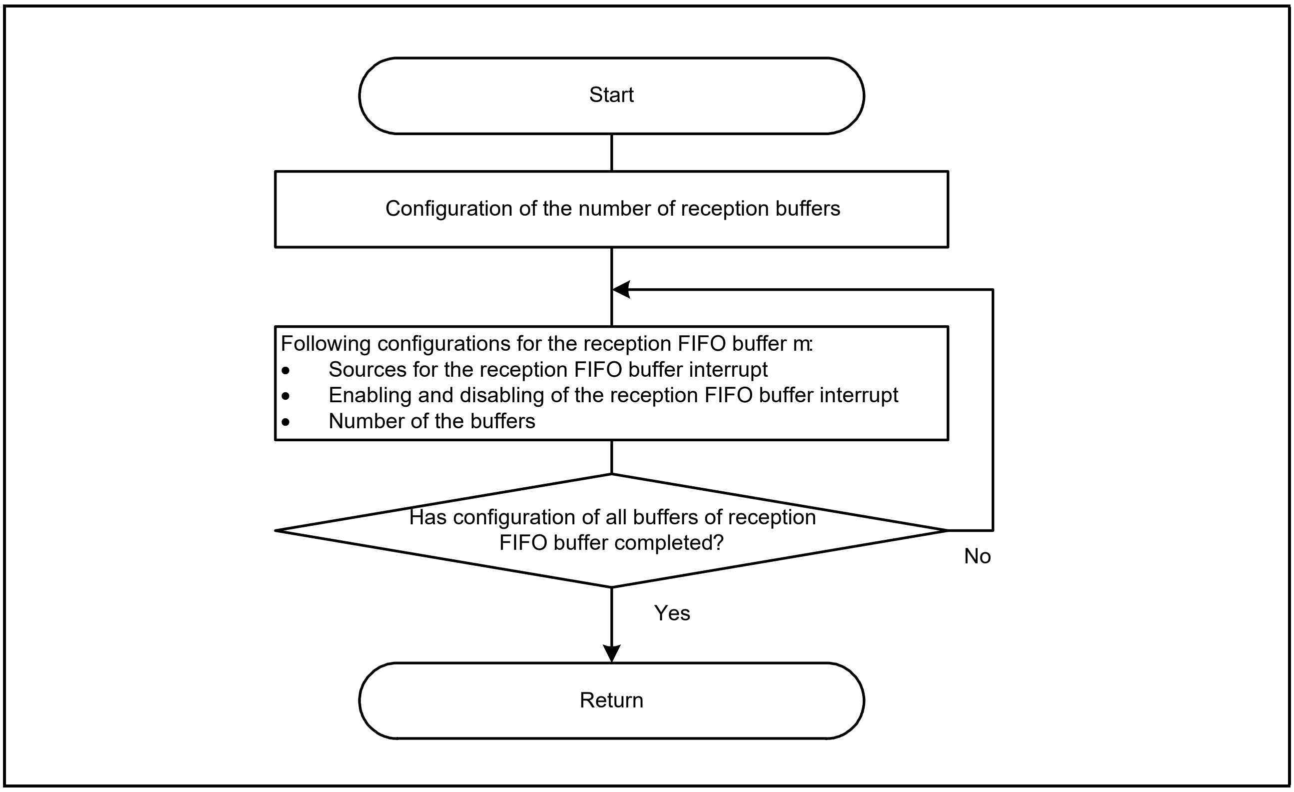


Figure 6.11 Procedure for Setting the Reception Buffers and Reception FIFO Buffers

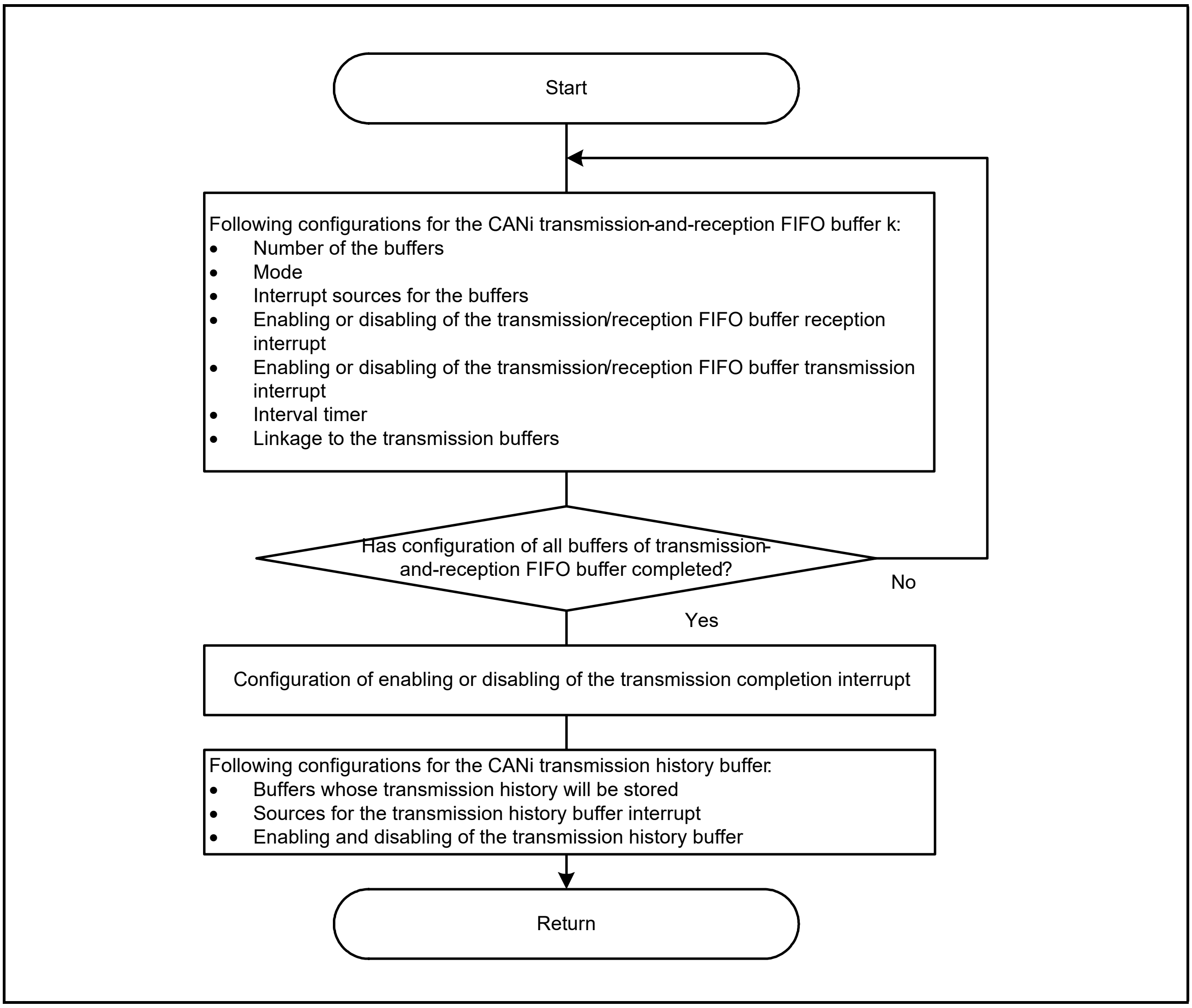


Figure 6.12 Procedure for Setting the Transmission-and-Reception FIFO Buffers, Transmission Buffers, and Transmission History Buffers

6.7 Global Error Interrupt

Settings for the global error interrupt is described below. The CAN module outputs an interrupt request for the interrupt enabling bit which is being enabled. Generation of interrupts also depends on the settings of the interrupt control registers of the interrupt controller.

6.7.1 Global Error Interrupts

There are following sources for global error interrupts.

- DLC checking error
- FIFO message loss
- Transmission history buffer overflow

(1) DLC checking error
In DLC checking, this error is detected if the DLC value of the received message, which has passed through the acceptance filter, is smaller than that of the reception rule.

(2) FIFO message is lost
This is detected if storing of a further received message is attempted while the reception FIFO buffer or the transmission-and-reception FIFO buffer is full.

(3) Transmission history buffer overflow
This is detected if storing of further transmission history is attempted while the transmission history buffer is full.

6.7.2 Procedure for Setting the Global Error Interrupt

Figure 6.13 shows the procedure for setting the global error interrupts. Make these settings during CAN configuration.

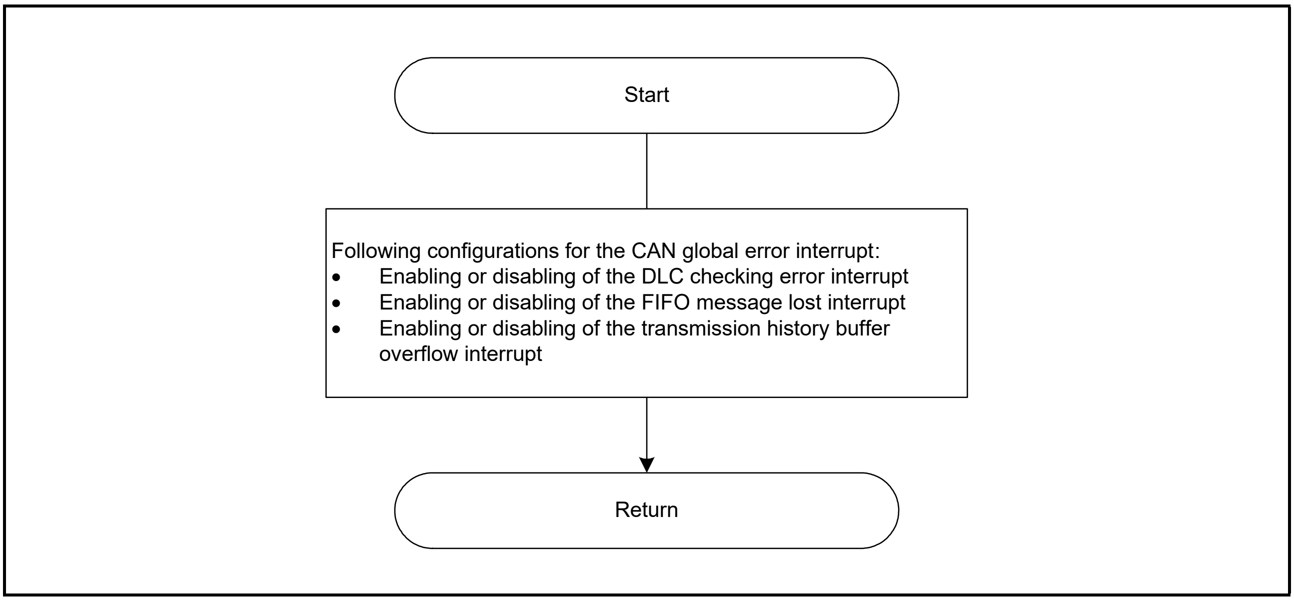


Figure 6.13 Procedure for Setting the Global Error Interrupt

6.8 Channel Functions

Configure the following features provided for the individual channels.

- Channel error interrupt
- Transmission abort interrupt
- Bus-off recovery mode
- Error display mode
- Transfer test mode

6.8.1 CANi Error Interrupts

The CANi error interrupts are enabled or disabled. The sources for these channel error interrupts are shown below.

- Bus error
- Error warning
- Error passive
- Bus-off entry
- Bus-off recovery
- Overload frame transmission
- Bus lockup
- Arbitration lost

(1) Bus error interrupt

This interrupt is generated on detection of any of the followings:

- A form error is detected in the ACK delimiter.
  The ADERR bit of the channel error flag register (RSCAN0CmERFL) is set to 1.
- A recessive bit is detected although a dominant bit has been transmitted.
  The B0ERR bit of the channel error flag register (RSCAN0CmERFL) is set to 1.
- A dominant bit is detected although a recessive bit has been transmitted.
  The B1ERR bit of the channel error flag register (RSCAN0CmERFL) is set to 1.
- A CRC error is detected.
  The CERR bit of the channel error flag register (RSCAN0CmERFL) is set to 1.
- An ACK error is detected.
  The AERR bit of the channel error flag register (RSCAN0CmERFL) is set to 1.
- A form error is detected.
  The FERR bit of the channel error flag register (RSCAN0CmERFL) is set to 1.
- A stuff error is detected.
  The SERR bit of the channel error flag register (RSCAN0CmERFL) is set to 1.

(2) Error warning interrupt

This interrupt is generated when an error warning state, where the value in the reception error counter or the transmission error counter exceeds 95, is first detected.

(3) Error passive interrupt

This interrupt is generated when an error passive state, where the value in the reception error counter or the transmission error counter exceeds 127, is first detected.

(4) Bus-off entry interrupt
This interrupt is generated on detection of a bus-off state, where the value in the transmission error counter exceeds 255. Entering a bus-off state as a result of setting the bus-off recovery mode to “transmission to channel halt mode at bus-off state” also causes this interrupt.

(5) Bus-off recovery interrupt
This interrupt is generated on detection of recovery from the bus-off state after eleven consecutive recessive bits have been detected 128 times.

(6) Overload frame transmission interrupt
This interrupt is generated on detection of a condition for transmitting the overload frame in reception or transmission.

(7) Bus lockup interrupt
This interrupt is generated on detection of the CAN bus being locked up, which is determined by the detection of 32 consecutive dominant bits on the CAN bus during channel transfer.

(8) Arbitration lost interrupt
This interrupt is generated on detection of a case of a loss in arbitration.

6.8.2 CANi Transmission Abort Interrupts

Enable or disable transmission abort interrupts during configuration. When this interrupt is enabled, it is generated when completion of transmission abort is detected.

Generation of a transmission abort interrupt triggers generation of the following CANi transmission interrupts:

- CANi transmission completion interrupt
- CANi transmission abort interrupt
- CANi transmission-and-reception FIFO transmission completion interrupt
- CANi transmission history interrupt

6.8.3 Bus-Off Recovery Mode

Behavior of the CAN module in bus-off recovery mode is selected by the BOM[1:0] bits of the channel control register (RSCAN0CmCTR) as follows.

- 00: The CAN module behaves in compliance with the ISO11898-1 specifications.
- 01: The CAN module makes a transition to the channel halt mode as it enters the bus-off state.
- 10: The CAN module makes a transition to the channel halt mode as it exits the bus-off state.
- 11: The CAN module makes a transition to the channel halt mode by a request from the program during bus-off state

6.8.4 Error Display Modes

Content of the errors on the CAN bus are displayed on the corresponding bits (bits 14 to 8) of the channel error flag register (RSCAN0CmERFL). The display mode of the errors is selected from the following.

- Displays the first error only (ERRD bit of the RSCAN0CmCTR register is 0)
  In this mode, only the flag for the first error event is set to 1. If two or more errors occur in the first error event, all the flags of the detected errors are set to 1.
- Displays all errors (ERRD bit of the RSCAN0CmCTR register is 1)
  In this mode, the flags for all error events are set to 1 regardless of the order of their occurrence.

6.8.5 Transfer Test Mode

A transfer test mode is selectable. The test functions are run by the CAN transceiver or the MCU for self-diagnosis of CAN communications and of the RAM.

6.8.6 Procedures for Setting the Channel Functions

Figure 6.14 shows a procedure for setting the channel functions. Make these settings during CAN configuration.

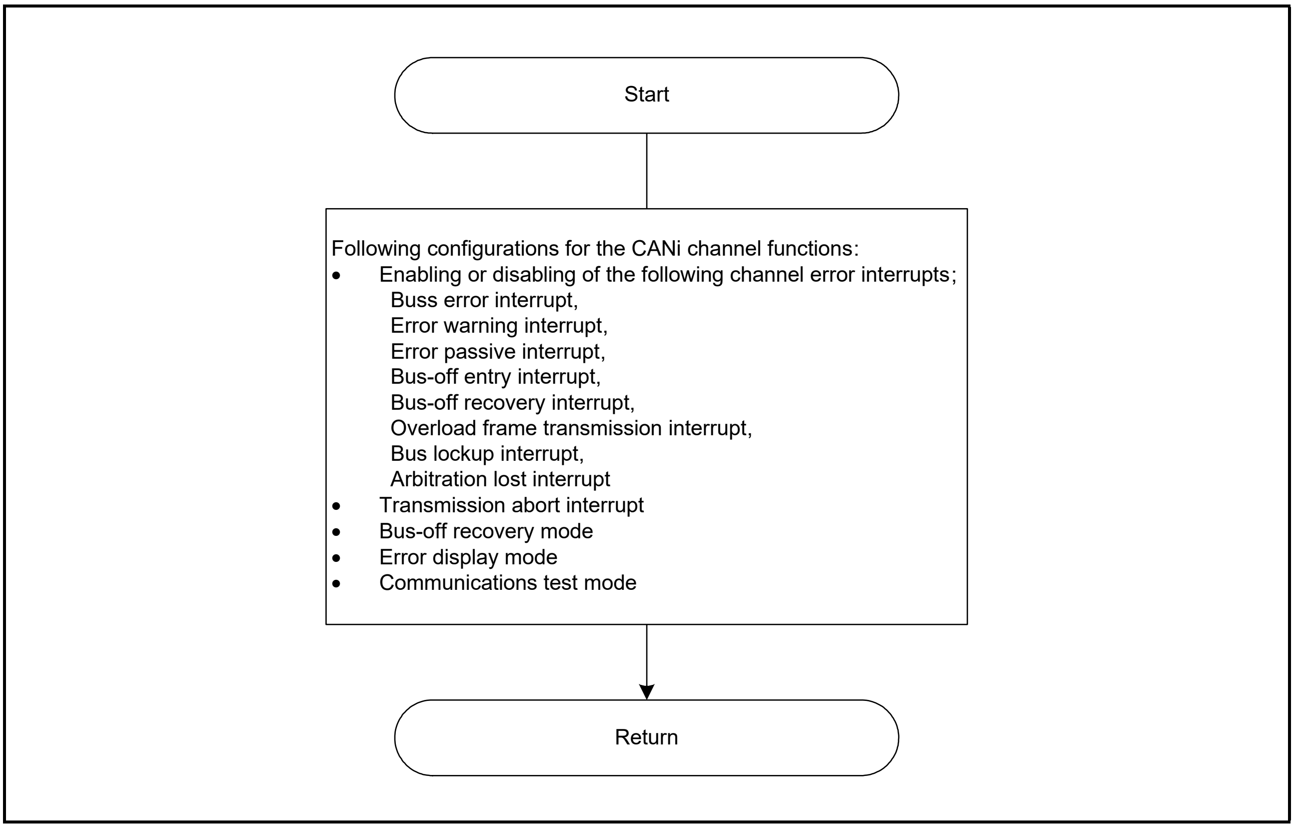


Figure 6.14 Procedure for Setting the Channel Functions

6.9 Configurations Required for Each CAN State (Mode)

Configurations required at each CAN state (mode) are shown in the tables below.

Note: "R": setting is required, "N/R": setting is not required, "N/A": setting is not allowed.

6.9.1 CAN State (Mode) Transition

Configuration Processing	State of the CAN Module			
	After MCU reset	After transition to global reset mode	After transition to channel reset mode	After transition to channel halt mode
Transition between global modes	R	R	N/A	N/A
Transition between channel modes	R	R	R	R

6.9.2 Global Facilities

Configuration Processing	State of the CAN Module			
	After MCU reset	After transition to global reset mode	After transition to channel reset mode	After transition to channel halt mode
Transmission priority	R	N/R	N/A	N/A
DLC checking	R	N/R	N/A	N/A
DLC replacement	R	N/R	N/A	N/A
Mirroring function	R	N/R	N/A	N/A
Clock	R	N/R	N/A	N/A
Timestamp clock	R	N/R	N/A	N/A
Interval timer prescaler	R	N/R	N/A	N/A

6.9.3 Transfer Rate

Configuration Processing	State of the CAN Module			
	After MCU reset	After transition to global reset mode	After transition to channel reset mode	After transition to channel halt mode
Bit time	R	N/R	N/R	N/R
Transfer rate	R	N/R	N/R	N/R

6.9.4 Reception Rule Table

Configuration Processing	State of the CAN Module			
	After MCU reset	After transition to global reset mode	After transition to channel reset mode	After transition to channel halt mode
Reception rule table	R	N/R	N/A	N/A

6.9.5 Buffers

Configuration Processing	State of the CAN Module			
	After MCU reset	After transition to global reset mode	After transition to channel reset mode	After transition to channel halt mode
Reception buffers	R	N/R	N/A	N/A
Reception FIFO buffers	R	N/R	N/A	N/A
Transmission-and-reception FIFO buffers	R	N/R	N/R	N/R
Transmission buffers	R	N/R	N/R	N/R
Transmission history buffers	R	N/R	N/R	N/R

6.9.6 Global Error Interrupts

Configuration Processing	State of the CAN Module			
	After MCU reset	After transition to global reset mode	After transition to channel reset mode	After transition to channel halt mode
Global error interrupts	R	N/R	N/A	N/A

6.9.7 Channels

Configuration Processing	State of the CAN Module			
	After MCU reset	After transition to global reset mode	After transition to channel reset mode	After transition to channel halt mode
Channel functions	R	N/R	N/R	N/R

7. Reception

7.1 Receiving Functions

CAN messages are received by using the following reception types. See the subsequent sections for the details on each type.

- Reception by using the reception buffers
- Reception by using the reception FIFO buffers
- Reception by using the transmission-and-reception FIFO buffers

7.2 Reception by Using the Reception Buffers

Reception buffer 0 to n + 1 are shared by both channels. Data (message) in a reception buffer will be overwritten when a new message is stored in the same reception buffer. Thus, the latest received data can be read. No interrupt is generated on reception of a message by a reception buffer.

Once storing of message to a reception buffer begins, the RMNS bit of the reception buffer new data register 0 (RSCAN0RMND0) is set to 1, which means reception buffer n contains a new message. Then, the data can be read from the reception buffer ID register (RSCAN0RMIDq), the reception buffer pointer register (RSCAN0RMPTRq), the reception buffer data field 0 register (RSCAN0RMDF0q), and the reception buffer data field 1 register (RSCAN0RMDF1q).

7.2.1 Procedures for Reading from a Reception Buffer

Figure 7.1 shows a procedure for reading from a reception buffer.

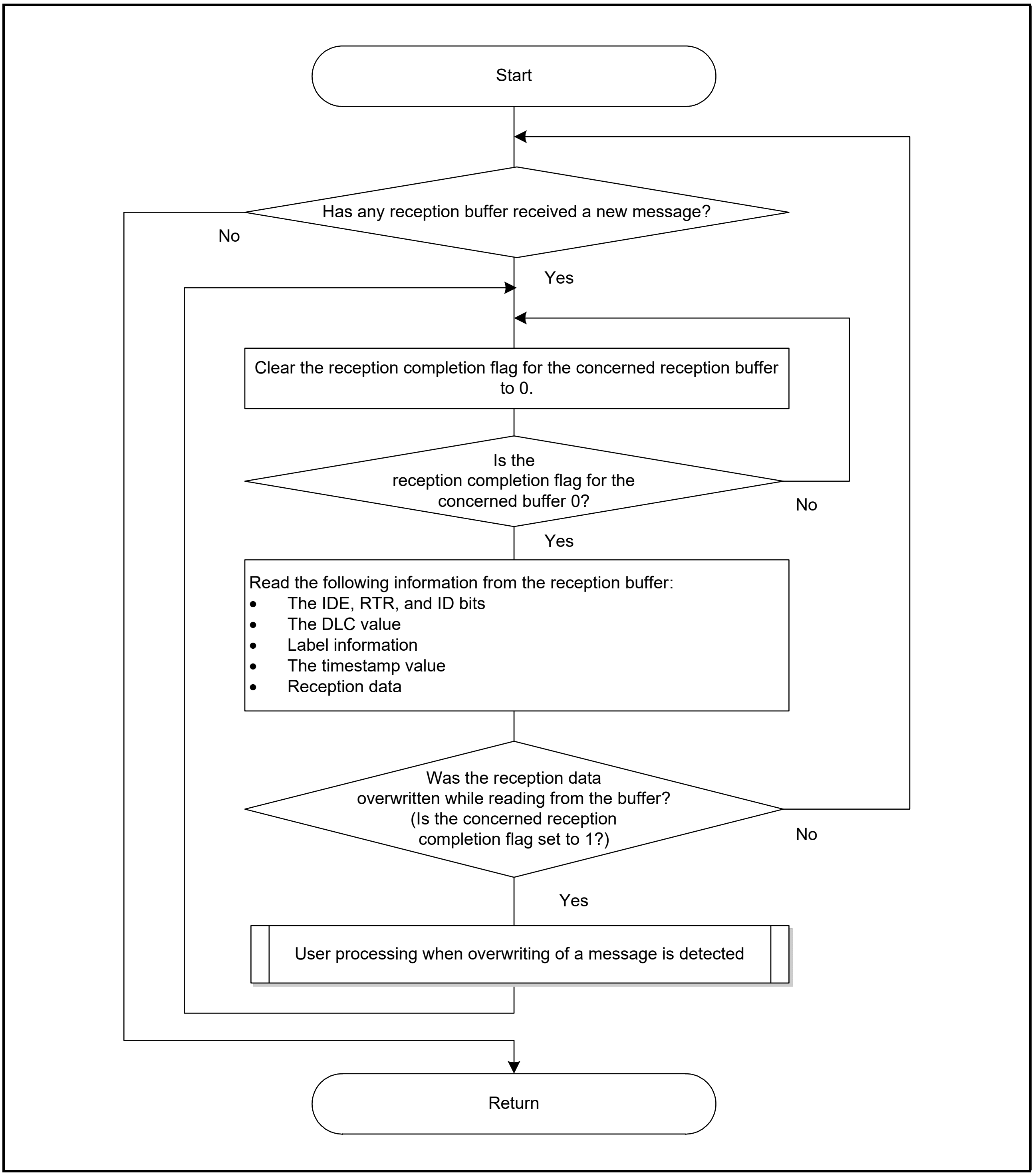


Figure 7.1 Procedure for Reading from a Reception Buffer

7.3 Reception by Using the Reception FIFO Buffers

Eight reception FIFO buffers are shared by both channels. Each reception FIFO buffer can retain messages up to the number equal to the number of reception buffers that each reception FIFO buffer has.

Once the received message has been stored in the reception FIFO buffer, the value of the corresponding message count display counter (the RFMC[7:0] bits in the reception FIFO buffer status register (RSCAN0RFSTSx)) is incremented.

The received message is read from the reception FIFO buffer access ID register (RSCAN0RFIDx), the reception FIFO buffer access pointer register (RSCAN0RFPTRx), the reception FIFO buffer access data field 0 register (RSCAN0RFDF0x), and the reception FIFO buffer access data field 1 register (RSCAN0RFDF1x).

When the value of the message count display counter matches the number of messages that can be stored in a single reception FIFO buffer (a value set by the RFDC bit of the RSCAN0RFCCx register), the buffer is full (the RFFL bit of the RSCAN0RFSTSx register is set to 1).

When all the messages have been read from the reception FIFO buffer, it is empty (the RFEMP bit of the RSCAN0RFSTSx register is set to 1).

7.3.1 Procedure for Reading from the Reception FIFO Buffers

Figure 7.2 shows a procedure for reading from a reception FIFO buffer.

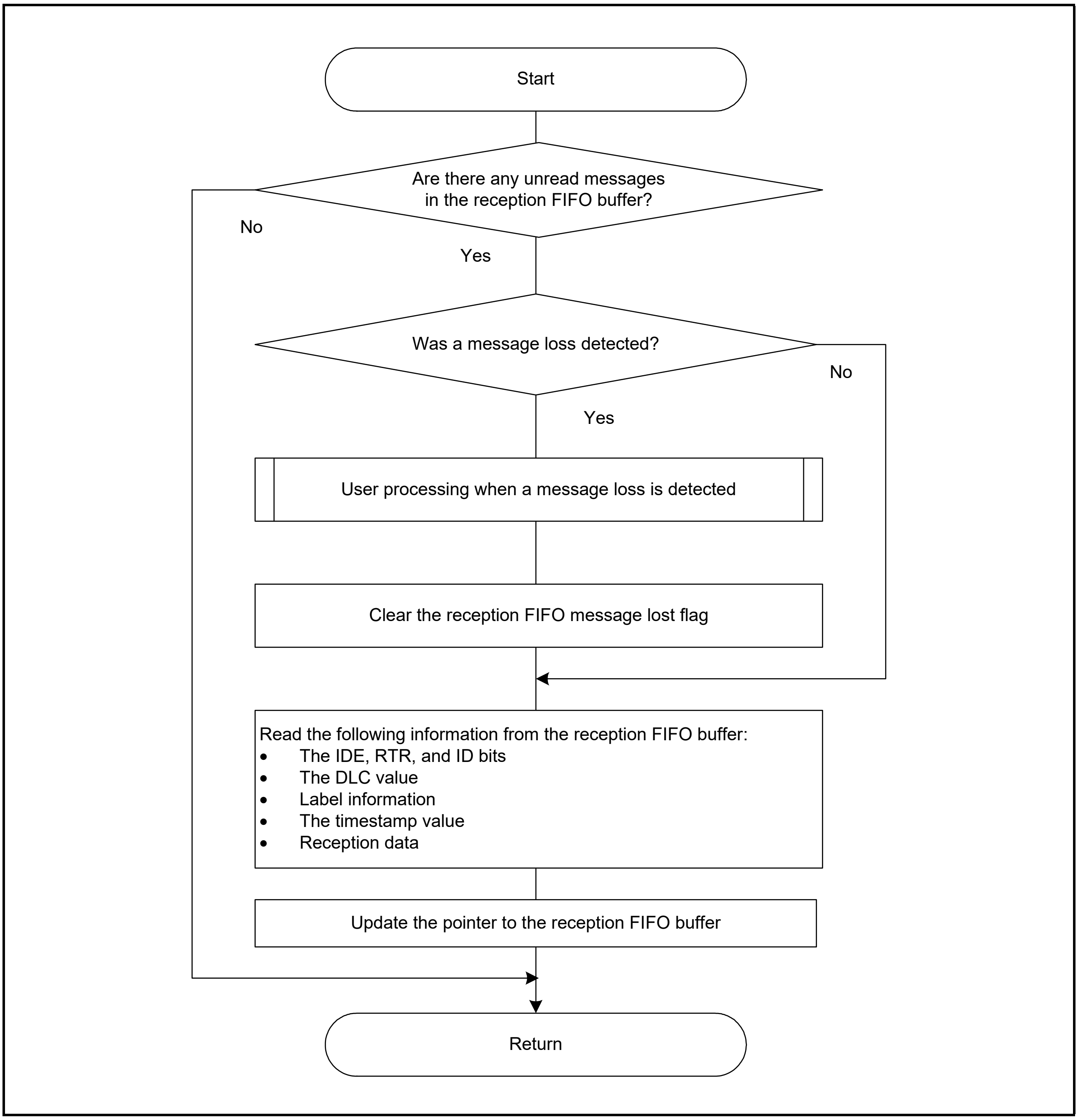


Figure 7.2 Procedure for Reading from a Reception FIFO Buffer

7.3.2 Handling of Reception FIFO-Related Interrupts

(1) Handling of reception FIFO interrupt

While this interrupt is enabled, it is generated when the conditions set by the RFIM bit of the RSCAN0RFCCx register are met.

Even if the reception FIFO buffers are disabled while an interrupt request is present (the RFIF bit of the RSCAN0RFSTSx register is set to 1), the interrupt request flag (the RFIR flag) is not automatically cleared to 0, so clear it by a program.

Each reception FIFO buffer is enabled and disabled individually by the RFIE bit of the RSCAN0RFCCx register.

The sources for this interrupt are shown below.

- An interrupt request is issued when the condition selected by the CFIGCV[2:0] bits of the RSCAN0RFCCx register met (this source is selected by setting the RFIM bit of the RSCAN0RFCCx register to 0).
- An interrupt request is issued every time reception of message completes (this source is selected by setting the RFIM bit of the RSCAN0RFCCx register to 1)

All the interrupt request flags for the reception FIFO interrupts which you want to use need to be set to 0 while the corresponding interrupt enable bits are being set to 1.

(2) Handling of global error interrupt

While this interrupt is enabled, it is generated on detection of a message loss in the reception FIFO buffer. This interrupt is enabled and disabled collectively for the whole CAN module by using the MEIE bit of the RSCAN0GCTR register.

7.4 Reception by Using the Transmission-and-Reception FIFO Buffers

The transmission-and-reception FIFO buffers are used either in reception mode or transmission mode. This section describes the reception mode only.

Each channel has three dedicated transmission-and-reception FIFO buffers. In reception mode, these buffers serve similarly as reception FIFO buffers and can retain messages up to the number equal to the number of the buffers that each transmission-and-reception FIFO buffer has.

Once the received message has stored in the transmission-and-reception FIFO buffer, the value of the corresponding message count display counter (the CFMC[7:0] bits in the transmission-and-reception FIFO buffer status register (RSCAN0CFSTSk)) is incremented.

The received message is read from the transmission-and-reception FIFO buffer access ID register (RSCAN0CFIDk), the transmission-and-reception FIFO buffer access pointer register (RSCAN0CFPTRk), the transmission-and-reception FIFO buffer access data field 0 register (RSCAN0CFDF0k), and the transmission-and-reception FIFO buffer access data field 1 register (RSCAN0CFDF1k). The data are sequentially read from each FIFO on a first-in, first-out basis.

When the value of the message count display counter matches the number of messages that can be stored in a single transmission-and-reception FIFO buffer (a value set by the CFDC[2:0] bits of the RSCAN0CFCCk register), the buffer is full (the CFFLL flag of the RSCAN0CFSTSk register is set to 1).

When all the messages have been read from the transmission-and-reception FIFO buffer, it is empty and the CFEMP bit of the RSCAN0CFSTSk register is set to 1.

7.4.1 Procedure for Reading from the Transmission-and-Reception FIFO Buffers

Figure 7.3 shows a procedure for reading from the transmission-and-reception FIFO buffers.

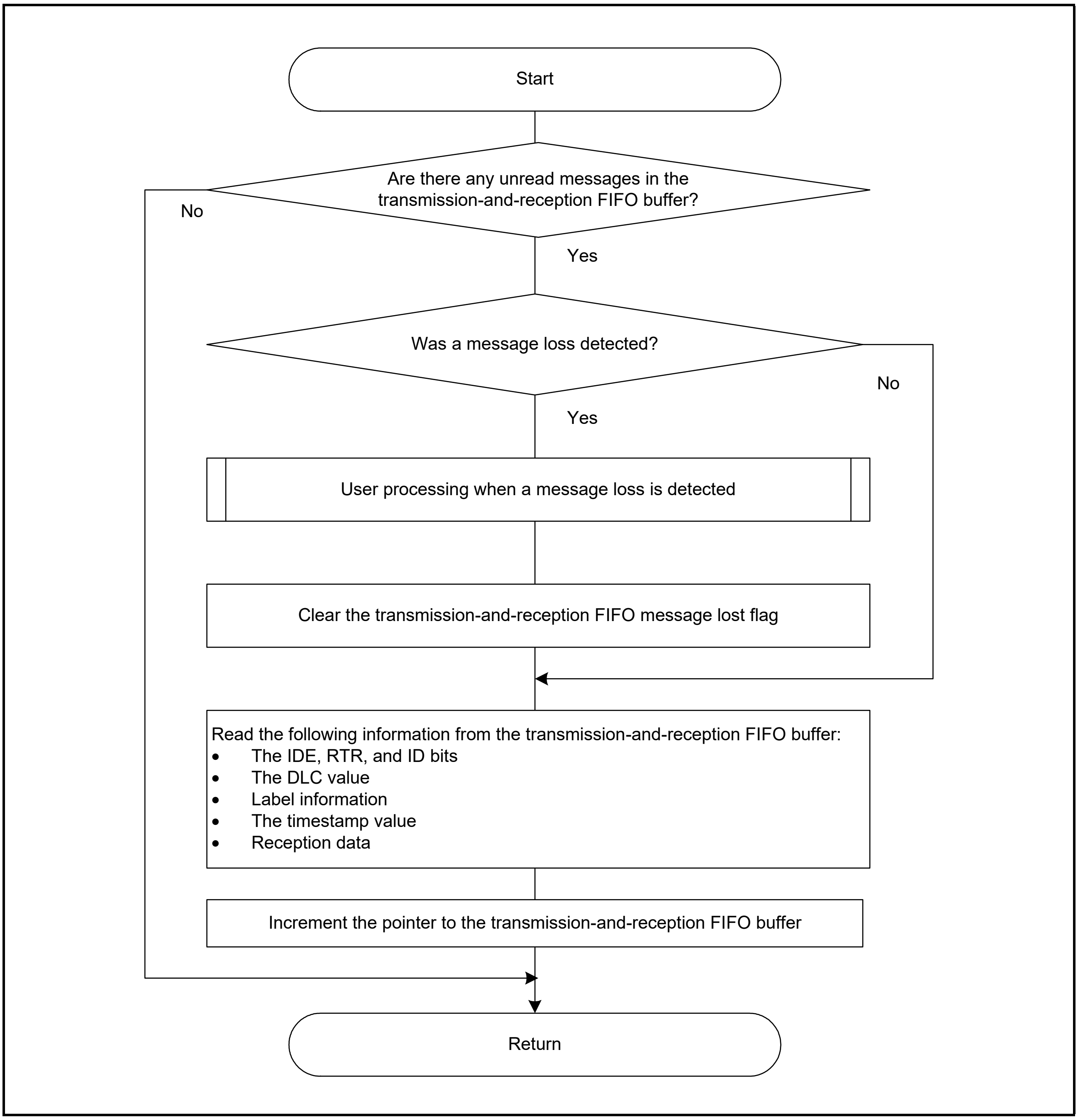


Figure 7.3 Procedure for Reading from the Transmission-and-Reception FIFO Buffers

7.4.2 Handling of Transmission-and-Reception FIFO Buffer-Related Interrupts (When Used in Reception Mode)

(1) Handling of transmission-and-reception FIFO reception completion interrupt
While this interrupt is enabled, it is generated when the conditions set by the CFIM bit of the RSCAN0CFCCk register are met.
Even if the transmission-and-reception FIFO buffers are disabled while an interrupt request is present (the CFRXIF bit of the RSCAN0CFSTSk register is set to 1), the interrupt request flag (the CFTXIF flag) is not automatically cleared to 0, so clear it by a program.
Each transmission-and-reception FIFO buffer is enabled and disabled individually by the CFRXIE bit of the RSCAN0CFCCk register. The sources for this interrupt are shown below.

- An interrupt request is issued when the condition selected by the CFIGCV[2:0] bits of the RSCAN0CFCCk register met (this source is selected by setting the CFIM bit of the RSCAN0CFCCk register to 0).
- An interrupt request is issued every time reception of message completes (this source is selected by setting the CFIM bit of the RSCAN0CFCCk register to 1)

All the interrupt request flags for the transmission-and-reception FIFO interrupts which you want to use need to be set to 0 while the corresponding interrupt enable bits are being set to 1.

(2) Handling of global error interrupt
While this interrupt is enabled, it is generated on detection of a message loss in the transmission-and-reception FIFO buffer. This interrupt is enabled and disabled collectively for the whole CAN module by using the MEIE bit of the RSCAN0GCTR register.

8. Transmission

8.1 Transmitting Functions

CAN messages are transmitted by using the following transmission types. See the subsequent sections for the details on each type.

- Transmission by using the transmission buffers
- Transmission by using the transmission-and-reception FIFO buffers
- Transmission by using the transmission history buffers

8.2 Transmission by Using the Transmission Buffers

Transmission of data frames and remote frames are possible by using the transmission buffers. A single channel contains sixteen transmission buffers which are used as transmission buffers or the buffers for linking to the transmission-and-reception FIFO buffers.

The transmission buffers are provided with the following features.

- Message transmission
- Transmission abort
- One-shot transmission (disables retransmission)

8.2.1 Message Transmission

This is a function to transmit data frames or remote frames. Issuing a transmission request for the target transmission buffer (by setting the TMTR bit of the RSCAN0TMCp register to 1) enables transmission of the message. The result of transmission is read from the TMTRF[1:0] flag of the RSCAN0TMSTSp register as follows:

- Transmission has been completed without a request for aborting transmission (TMTRF[1:0] flag is B’10)
- Transmission has been completed with a request for aborting transmission (TMTRF[1:0] flag is B’11)

Each transmission completion interrupt is enabled and disabled individually by the TMIEp bit of the RSCAN0TMIEC0 register.

8.2.2 Procedure for Transmitting Messages from the Transmission Buffer

Figure 8.1 shows a procedure for transmitting messages from the transmission buffer.

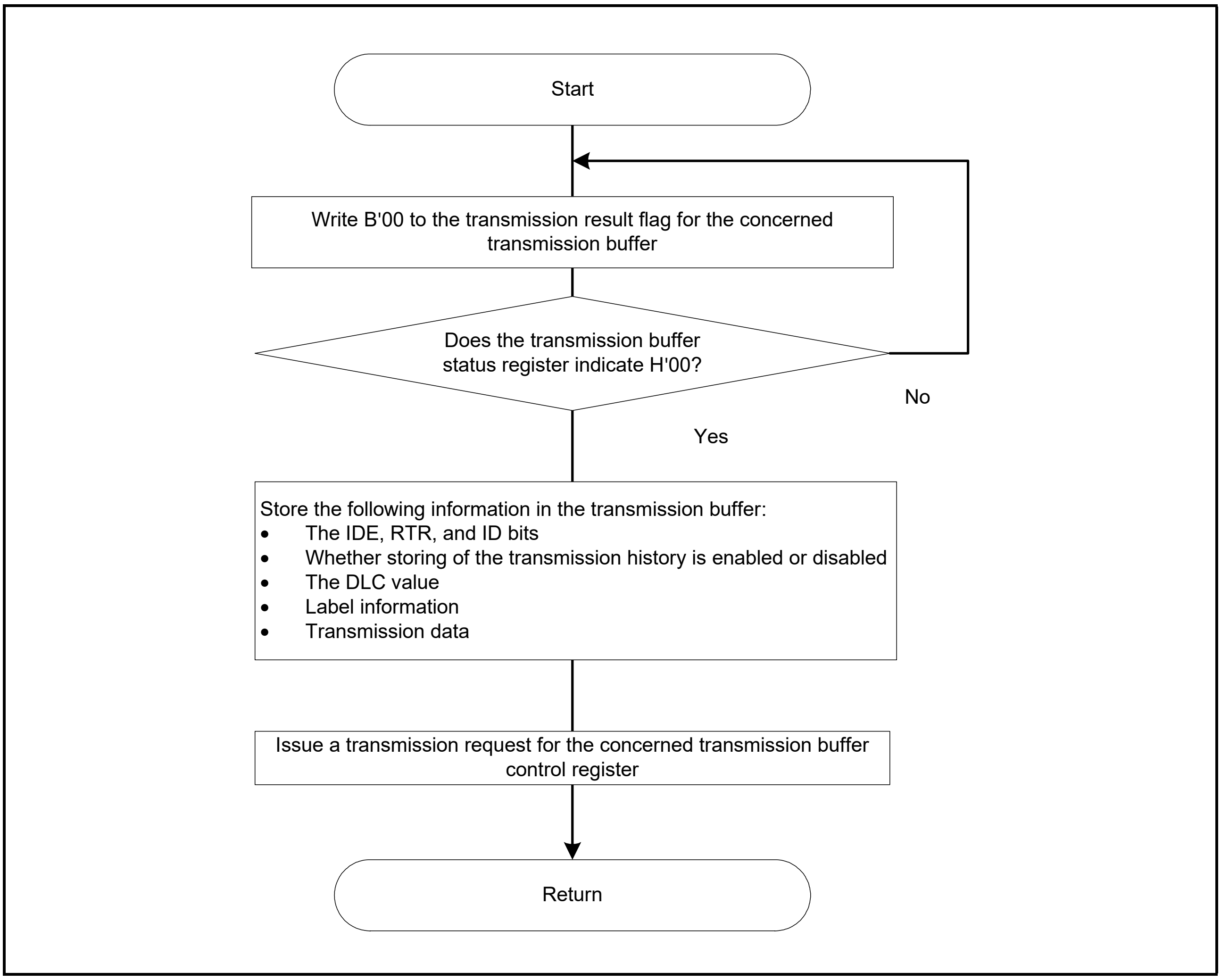


Figure 8.1 Procedure for Transmitting a Message from the Transmission Buffer

8.2.3 Transmission Abort

Aborting transmission refers to the discarding of a message for which transmission was being retried.

When two or more nodes begin transmission at the same time, the nodes containing the messages with lower-priority CAN IDs lose in arbitration and cannot complete transmission unless they subsequently win in arbitration or retry transmission while the CAN bus is idle. Messages for which transmission is being retried are cancelled by aborting transmission.

This function can be used to set up a time limit for the transmission of a message or for the transmission of a message as urgent (i.e., by giving it a higher priority).

The transmission request which has been issued for a transmission buffer (by setting the TMTRM bit of the RSCAN0TMSTSp register to 1) will be cancelled by issuing a request for aborting the transmission to the concerned buffer (by setting the TMTAR bit of the RSCAN0TMCp register to 1).

Once a request to abort transmission is issued, transmission of the message concerned is aborted at the following times depending on its state.

The message for which transmission is in progress or which has the second highest priority of transmission:

- When a loss in arbitration occurs
- When an error occurs

Other than above:

- On issuing of an explicit request to abort transmission

Once the transmission abort is completed, the TMTRF[1:0] flag of the RSCAN0TMSTSp register is set to B'01 and the transmission request is cancelled (the TMTRM bit is cleared to 0).

After a request for aborting the transmission is issued for a message for which transmission is in progress or which has the second highest priority of transmission, if the concerned message is successfully transmitted without arbitration losses or any errors, the result of transmission is read as follows.

- Transmission has been completed with a request for aborting the transmission (TMTRF[1:0] flag is set to B'11)

8.2.4 Procedure for Aborting Message Transmission

Figure 8.2 shows a procedure for aborting message transmission.

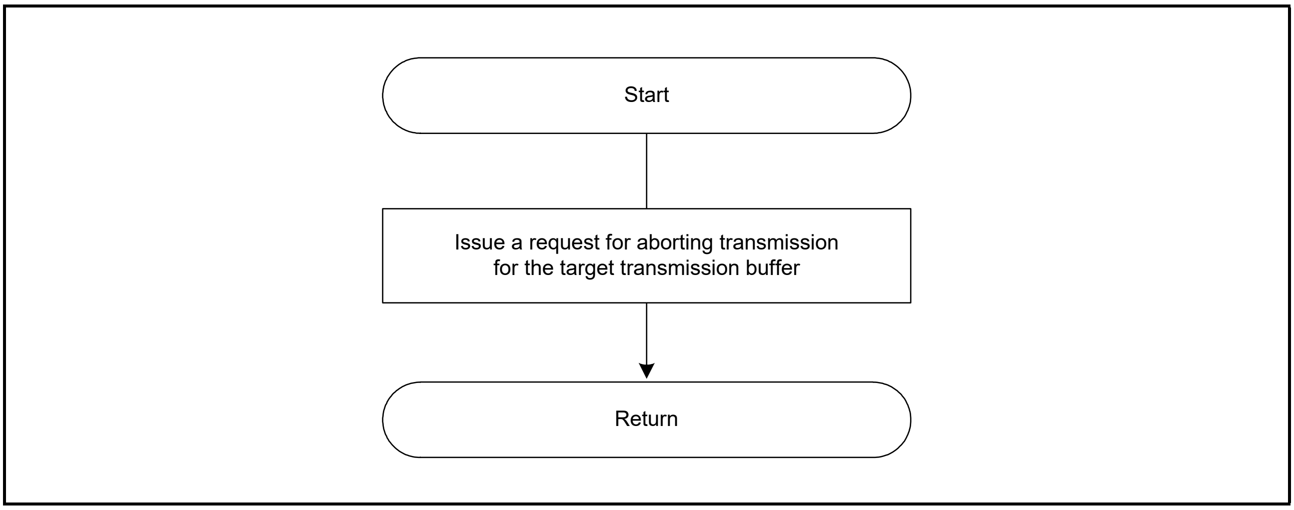


Figure 8.2 Procedure for Aborting Message Transmission

8.2.5 One-Shot Transmission Function

By enabling this function (by setting the TMOM bit of the RSCAN0TMCp register to 1), only a single transmission is attempted for the message for which a transmission request has been issued. This means that transmission will not be retried after an arbitration loss or any errors.

The result of one-shot transmission is read from the TMTRF[1:0] flag of the RSCAN0MSTSp register as follows.

At successful transmission:

- Transmission has been completed without a request for aborting the transmission (TMTRF[1:0] flag is B’10)
- Transmission has been completed with a request for aborting the transmission (TMTRF[1:0] flag is B’11)

At occurrence of an arbitration loss or an error:

- Transmission abort has been completed (TMTRF[1:0] flag is set to B’01)

8.2.6 Procedure for Transmission by Using the One-Shot Transmission Function

Figure 8.3 shows a procedure for transmission by using the one-shot transmission function.

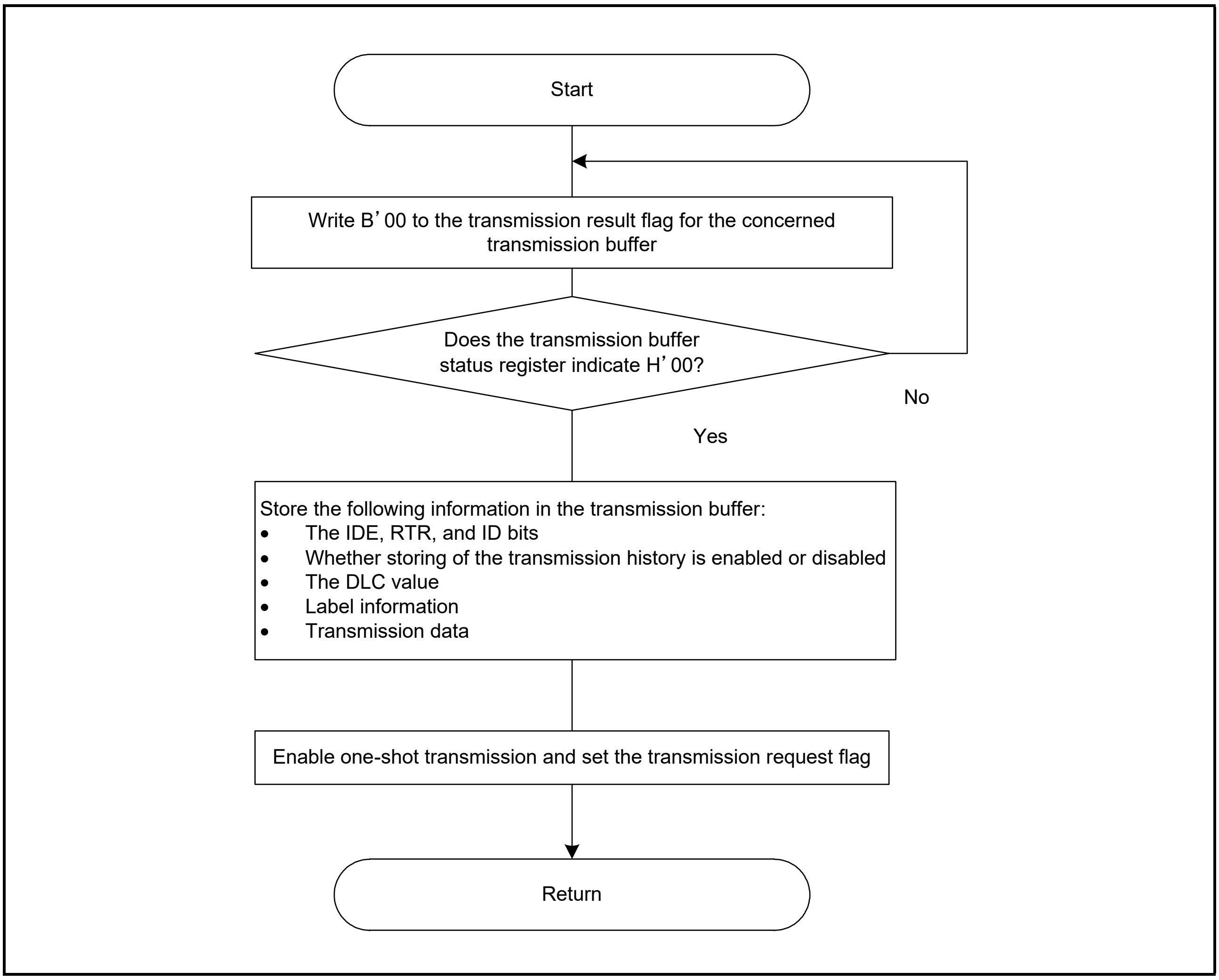


Figure 8.3 Procedure for Transmission by Using the One-Shot Transmission Function

8.2.7 Handling of Transmission Buffer-Related Interrupts

(1) Handling of transmission completion interrupt

While this interrupt is enabled, a CANi transmission interrupt is generated on completion of transmission of a message.

Transmission completion interrupt is enabled and disabled for the individual transmission buffers by the TMIEq bit of the RSCAN0TMIEC0 register.

The sources for the CANi transmission interrupt are shown below. If the user uses two or more interrupt sources, identify each source while the interrupt is being handled as required. These source flags are also read from the RSCAN0GTINTSTS0 register.

- CANi transmission completion interrupt
- CANi transmission abort interrupt
- CANi transmission-and-reception FIFO transmission completion interrupt
- CANi transmission history interrupt

(2) Handling of transmission abort interrupt

While this interrupt is enabled, a CANi transmission completion interrupt is generated on completion of aborting a transmission. Transmission abort interrupt is enabled and disabled for the individual channels by the TAIE bit of the RSCAN0CmCTR register. However, if the transmission for which a request for abortion has been issued is already successfully completed (the TMTRF[1:0] flag is set to B'11), a transmission completion interrupt will be generated instead of a transmission abort interrupt.

The sources for the CANi transmission interrupt are shown below. If the user uses two or more interrupt sources, identify each source while the interrupt is being handled as required. The source flags are also read from the RSCAN0GTINTSTS0 register.

- CANi transmission completion interrupt
- CANi transmission abort interrupt
- CANi transmission-and-reception FIFO transmission completion interrupt
- CANi transmission history interrupt

8.2.8 Processing after Completion of Message Transmission or Transmission Abort

(1) Processing after completion of message transmission or transmission abort when interrupt is disabled

Figure 8.4 shows a processing after completion of message transmission or transmission abort when interrupt is disabled.

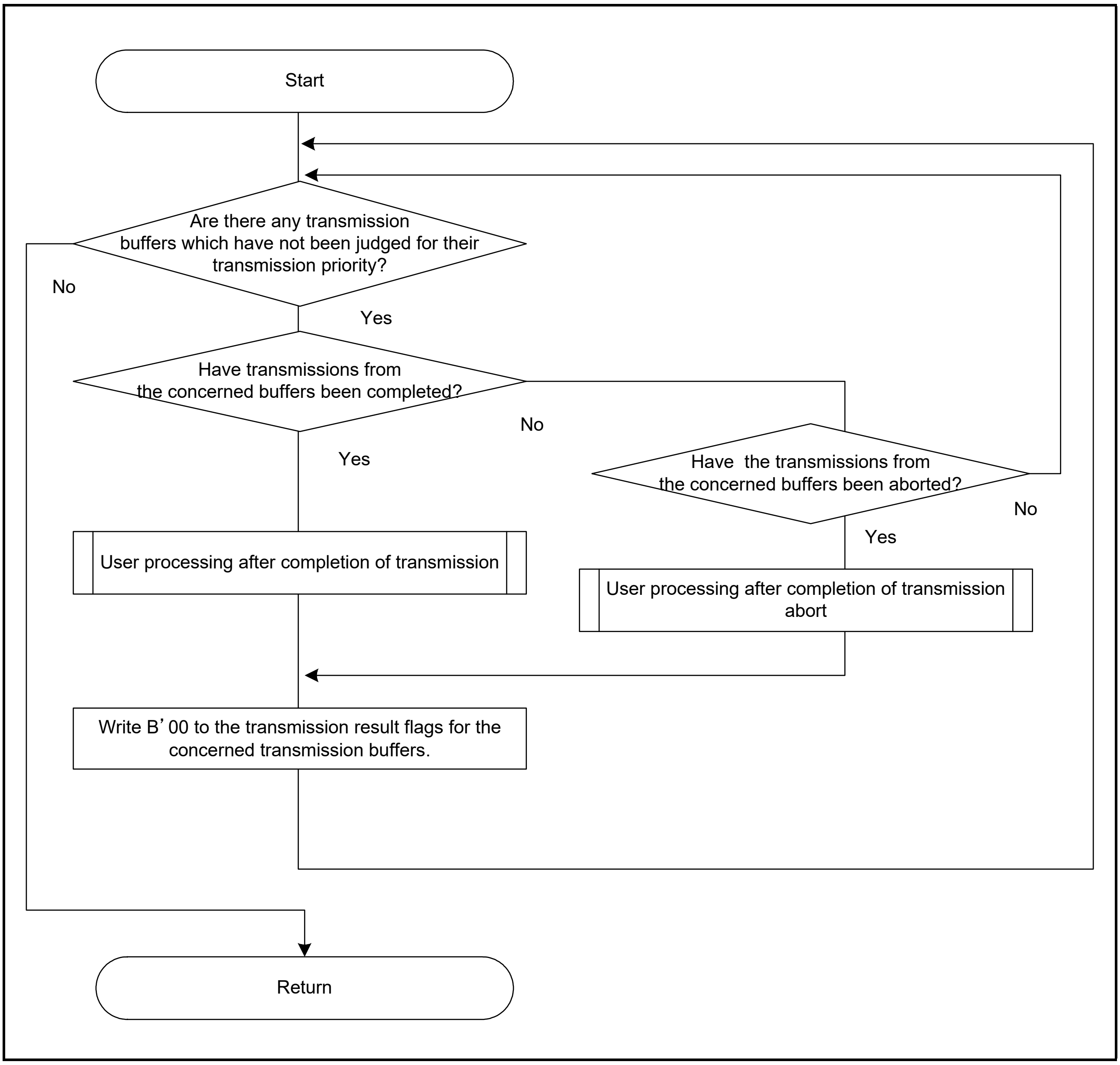


Figure 8.4 Processing after Completion of Message Transmission or Transmission Abort when Interrupt is Disabled

(2) Processing after completion of message transmission when interrupt is enabled
Figure 8.5 shows a processing after completion of message transmission when interrupt is enabled.

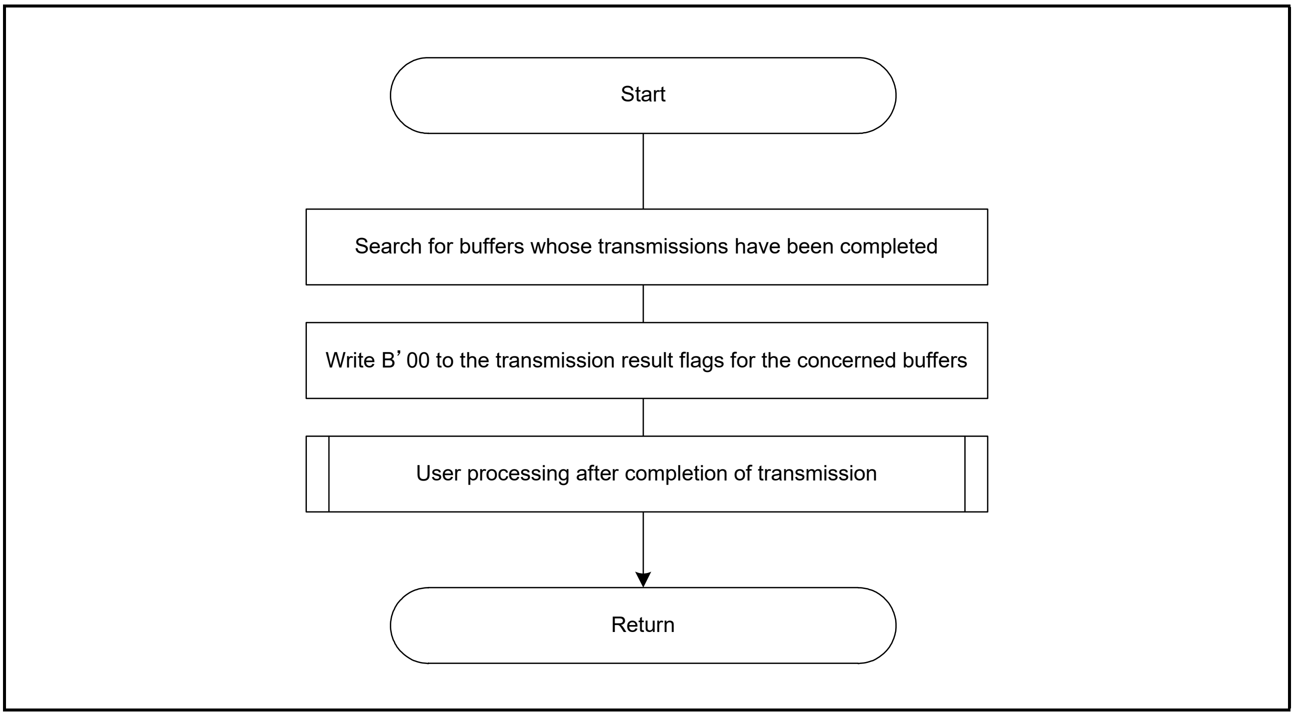


Figure 8.5 Processing after Completion of Message Transmission when Interrupt is Enabled

(3) rocessing after completion of transmission abort when interrupt is enabled
Figure 8.6 shows a processing after completion of transmission abort when interrupt is enabled.

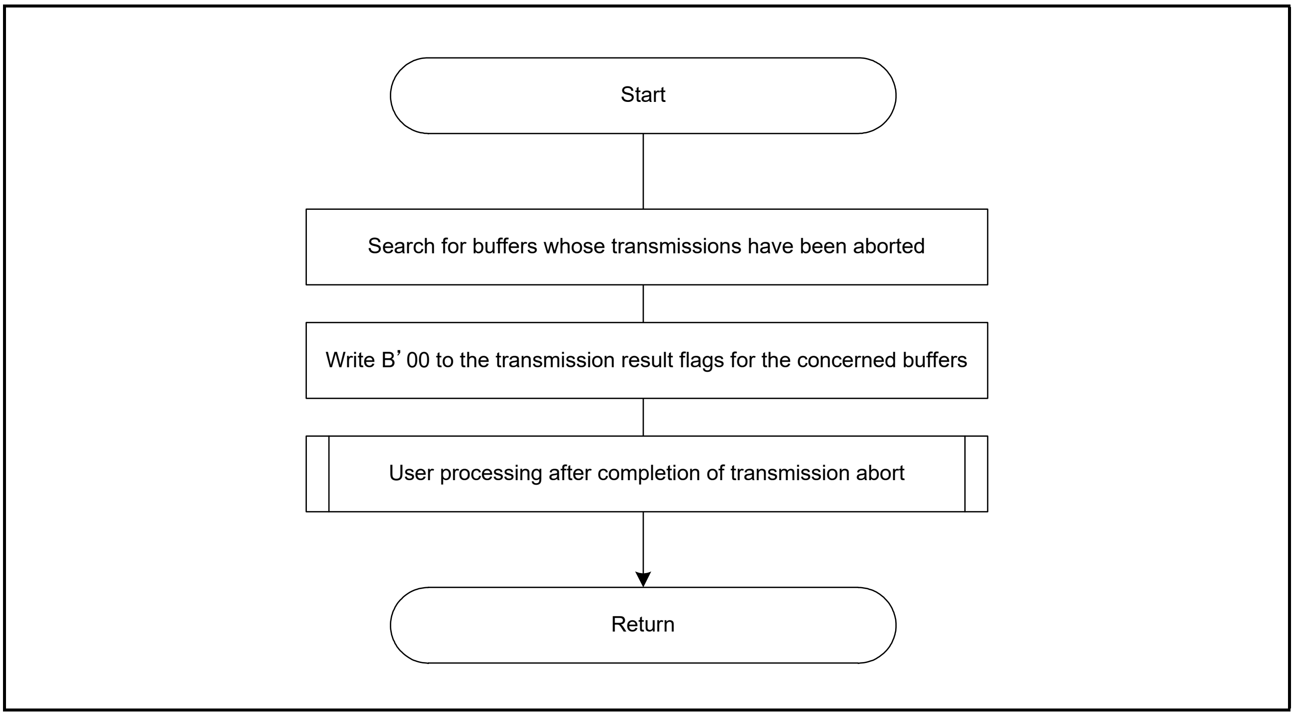


Figure 8.6 Processing after Completion of Transmission Abort when Interrupt is Enabled

8.3 Transmission by Using the Transmission-and-Reception FIFO Buffers

Transmission of data frames and remote frames are possible by using the transmission-and-reception FIFO buffers. Each channel has three transmission-and-reception FIFO buffers, each of which can retain up to 128 messages. The messages are transmitted sequentially on a first-in, first-out basis.

The transmission-and-reception FIFO buffers are used in either the reception mode or the transmission mode. This section describes the transmission mode only.

The transmission-and-reception FIFO buffers are provided with the following features.

- Message transmission
- Transmission abort
- Interval transmission

8.3.1 Message Transmission

This is a function to transmit data frames or remote frames. The messages stored in the transmission-and-reception FIFO buffers are transmitted sequentially on a first-in, first-out basis.

8.3.2 Procedure for Transmitting Messages from a Transmission-and-Reception FIFO Buffer

Figure 8.7 shows a procedure for transmitting messages from a transmission-and-reception FIFO buffer.

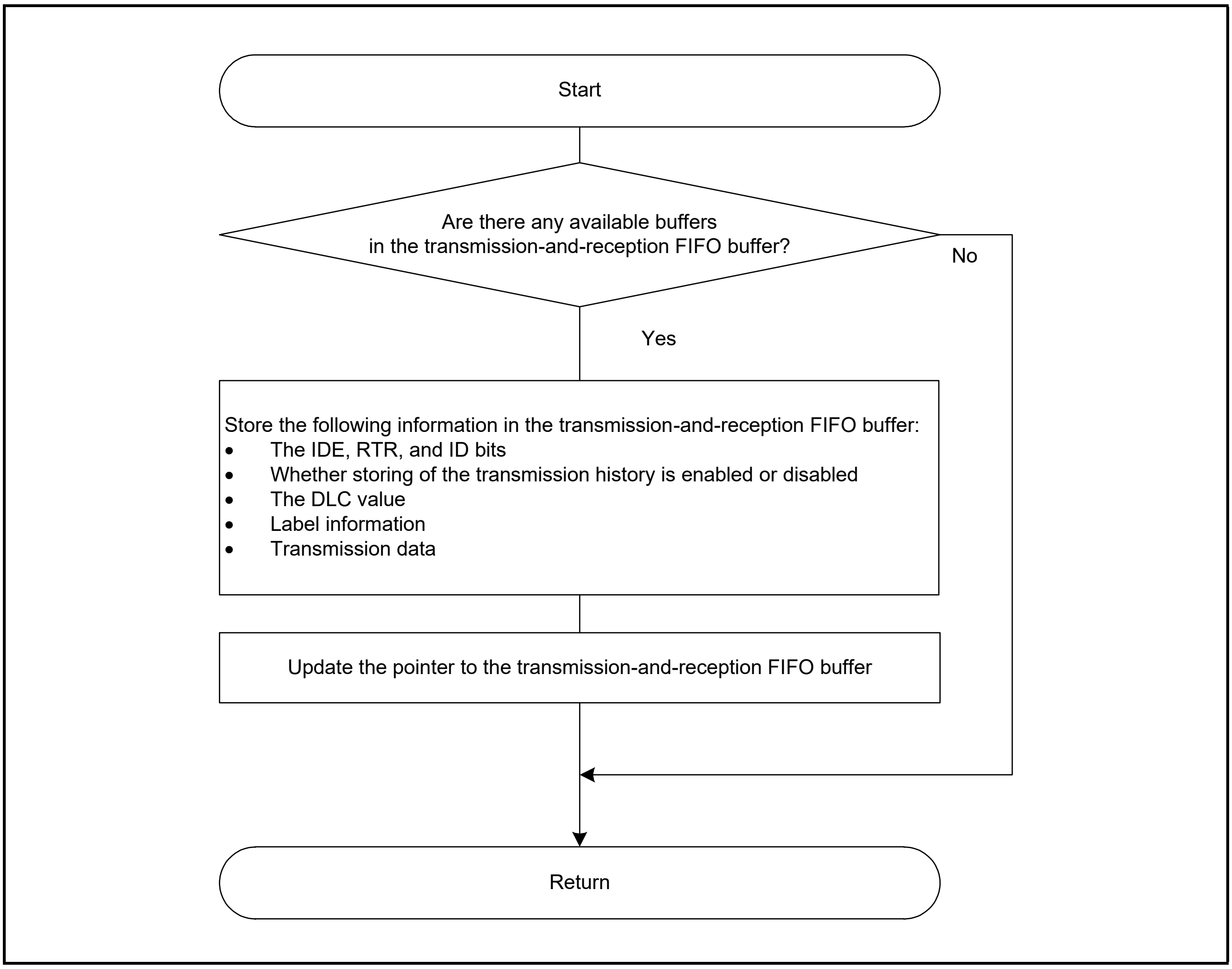


Figure 8.7 Procedure for Transmitting Messages from a Transmission-and-Reception FIFO Buffer

8.3.3 Transmission Abort

Disabling the transmission-and-reception FIFO buffers leads to abortion of the transmission of all the messages in the buffers regardless of whether transmission of any is in progress or not. Once the abort is completed, the transmission-and-reception FIFO buffer becomes empty.

Completion of transmission abort for the transmission-and-reception FIFO buffers does not cause an interrupt. Still, it may cause a transmission-and-reception FIFO transmission completion interrupt if a message for which a request for abortion has been issued was successfully transmitted.

8.3.4 Interval Transmission

In transmission mode, a transmission interval can be specified for sequential transmissions from the same transmission-and-reception FIFO buffer.

8.3.5 Handling of Transmission-and-Reception FIFO Interrupts (Transmission Mode)

(1) Handling of transmission-and-reception FIFO interrupt

While the transmission-and-reception FIFO transmission completion interrupt is enabled, a CANi transmission interrupt is generated according to the setting in the CFIM bit of the RSCAN0CFCCk register.

The sources for the CANi transmission interrupt are shown below. If the user uses two or more interrupt sources, identify each source while the interrupt is being handled as required. The source flags are also read from the RSCAN0GTINTSTS0 register.

- CANi transmission completion interrupt
- CANi transmission abort interrupt
- CANi transmission-and-reception FIFO transmission completion interrupt
- CANi transmission history interrupt

Transmission-and-reception FIFO transmission completion interrupts are enabled and disabled. When the interrupt is enabled, the interrupt sources are selected from the following.

- An interrupt request is issued when the buffer becomes empty upon completion of transmission.
- An interrupt request is issued every time message transmission completes.

8.4 Transmission History Buffers

Users can select whether or not to store the information of the transmitted messages (transmission history data) in the transmission history buffers. A single channel contains one transmission history buffer which can hold history data of sixteen transmissions.

8.4.1 Storing Transmission History Data

Whether or not to store the transmission history is decided for the individual transmission sources (transmission buffers) at the time of configuration. For the transmission buffers which are configured for storage of their transmission history, whether or not to store the transmission history and which label to attach to the history data is selectable each time a message is transmitted.

The following data are stored in the transmission history buffer on successful transmission:

Buffer type

This is the type of buffer (transmission buffer or transmission-and-reception FIFO buffer) for which the item of transmission history has been stored.

Buffer number

This is the number of the transmission buffer or the transmission-and-reception FIFO buffer for which transmission history has been stored.

Label data

This is the information of the transmitted message. Users can freely set the label for the messages when they are stored in the reception buffers.

8.4.2 Procedure for Reading From a Transmission History Buffer

Figure 8.8 shows a procedure for reading from a transmission history buffer.

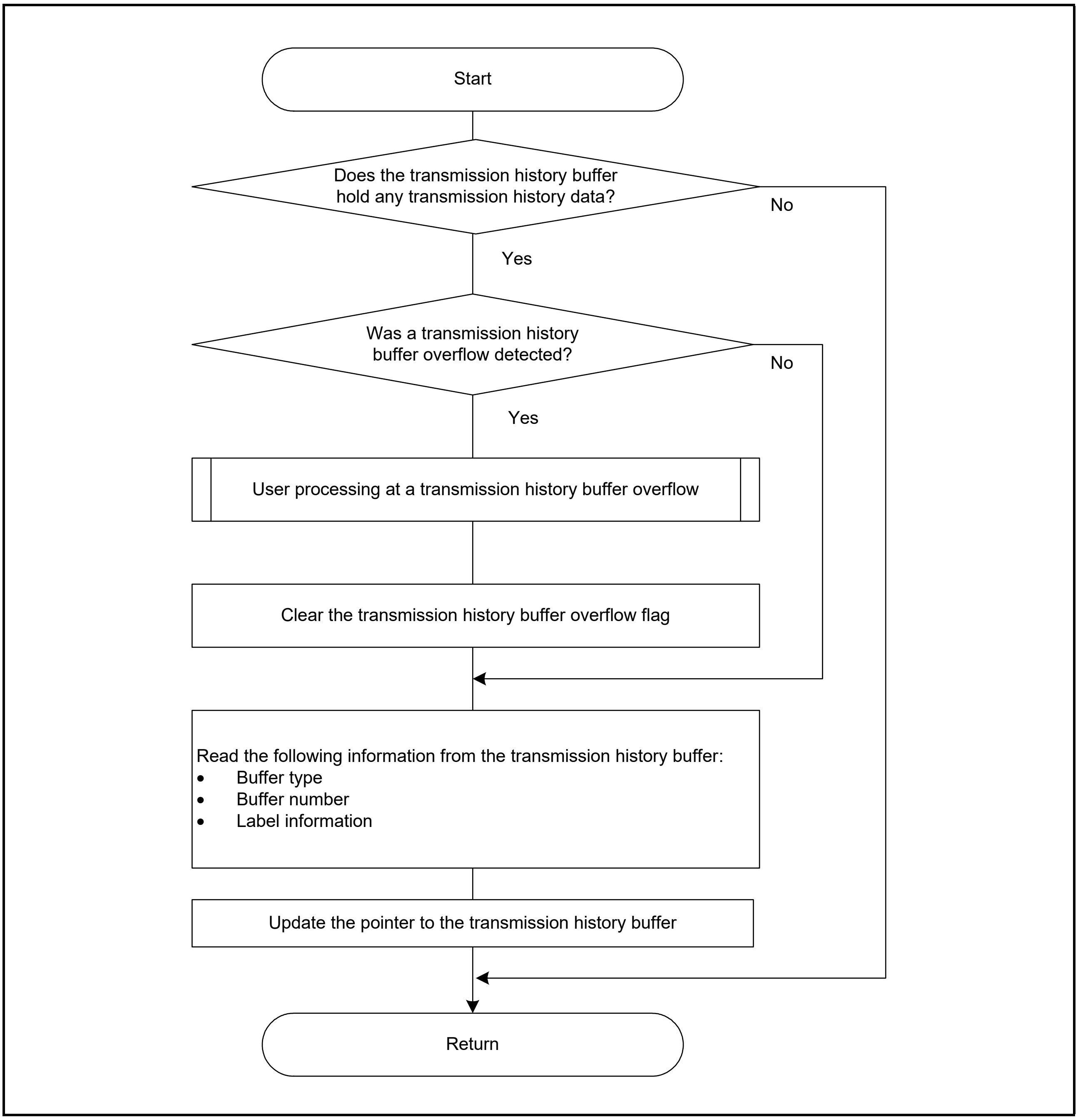


Figure 8.8 Procedure for Reading from a Transmission History Buffer

8.4.3 Handling of Transmission History Interrupts

(1) Handling of transmission history interrupt

While the transmission history interrupt is enabled, a CANi transmission interrupt occurs when the condition selected in the THLIM bit of the RSCAN0THLCCm register is satisfied.

The sources for the CANi transmission interrupt are shown below. If the user uses two or more interrupt sources, identify each source while the interrupt is being handled as required. The source flags are also read from the RSCAN0GTINTSTS0 register.

- CANi transmission completion interrupt
- CANi transmission abort interrupt
- CANi transmission-and-reception FIFO transmission completion interrupt
- CANi transmission history interrupt

(2) Handling of global error interrupt

While this interrupt is enabled, it is generated on detection of a message overflow error in the transmission history buffer. This interrupt is enabled and disabled collectively for the whole CAN module by using the THLEIE bit of the RSCAN0GCTR register.

9. CAN-Related Interrupts

9.1 CAN-Related Interrupts

The following CAN-related interrupts are available for the module. Each interrupt is enabled or disabled by the settings for the corresponding interrupt request.

Global interrupts:

CAN reception FIFO interrupt
CAN global error interrupt

Channel interrupts:

CANi transmission interrupts

- CANi transmission completion interrupt
- CANi transmission abort interrupt
- CANi transmission-and-reception FIFO transmission completion interrupt
- CANi transmission history interrupt
- CANi transmission queue interrupt

CANi transmission-and-reception FIFO reception completion interrupt
CANi error interrupt

9.1.1 Procedure for Setting the CAN Related Interrupts

Figure 9.1 shows a procedure for setting the can related interrupts.

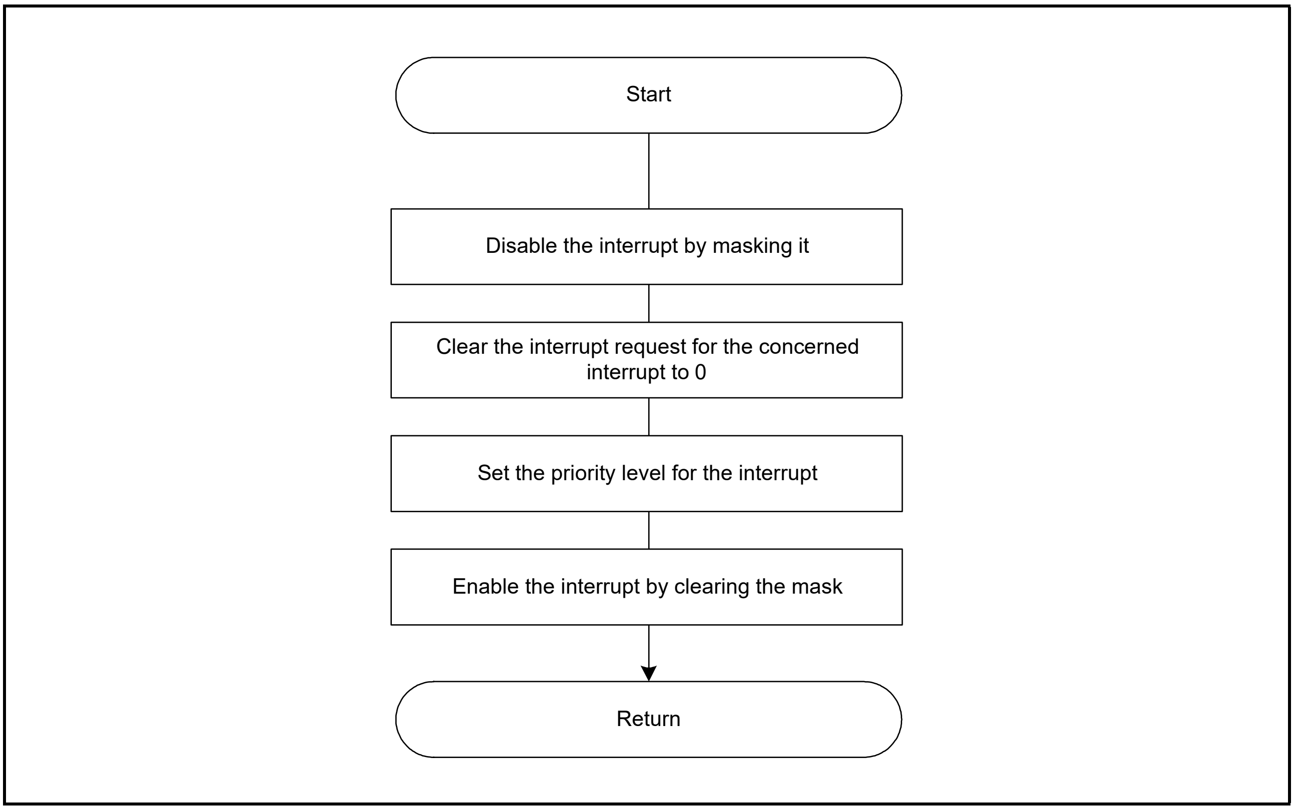


Figure 9.1 Procedure for Setting the CAN-Related Interrupts

10. Software

10.1 Operational Outline

Table 10.1 is the outline of the features of a sample program for the RSCAN module and Figure 10.1 is a system block diagram.

Table 10.1 Outline of the Features

Function	Outline
Alternative pins	PC6: CAN0 CRXD0 P67: CAN0 CTXD0 PC7: CAN1 CRXD1 P66: CAN1 CTXD1
Channel for CAN communications	Channel 0 (CAN0)
Interrupt sources (number in parentheses indicates the priority order)	CAN global error (3) CAN0 error (4) CAN1 error (4) CAN reception FIFO (5) CAN0 transmission-and-reception FIFO reception completion (5) CAN1 transmission-and-reception FIFO reception completion (5) CAN0 transmission (5) CAN1 transmission (5)
Transfer rate	1 Mbps
Operational modes	Transmission mode (sending side of the two connected evaluation boards): • Message transmission by using the transmission buffers • Message transmission by using the transmission-and-reception FIFO buffers in transmission mode Reception mode (receiving side of the two connected evaluation boards): • Message reception by using the reception buffers • Message reception by using the reception FIFO buffers • Message reception by using the transmission-and-reception FIFO buffers in reception mode Test for transmission while receiving data at the same time Test mode (test in a single evaluation board): • Self-test mode 0 (external loopback mode) • Self-test mode 1 (internal loopback mode)
Operational outline	Operating modes are selected from the menu.
Operation result display	The result of an operation is output to the console.

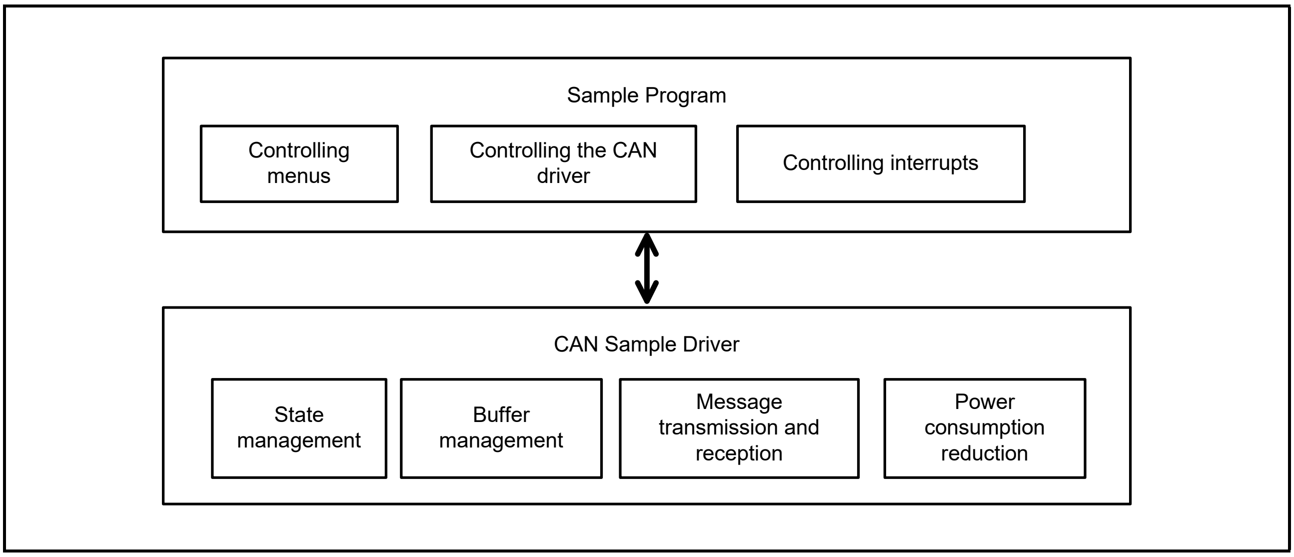


Figure 10.1 System Block Diagram

10.1.1 Setting of Projects

The projects for the development environments EWARM, DS-5, and e2studio are described in the RZ/T1 Group Application Note: Initial Settings.

10.1.2 Preparation for Self-Test

This sample program provides self-testing for CAN communications. A test for a single evaluation board which is connected to the development environment is possible.

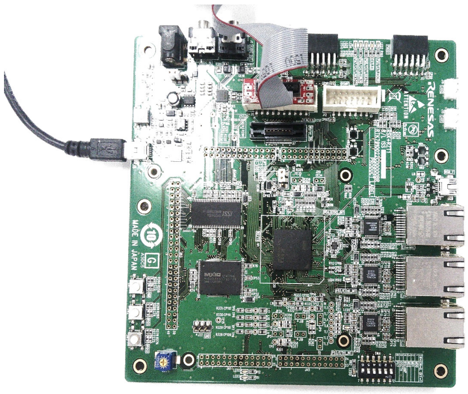


Figure 10.2 Configuration for Self-Testing by a Sample Program

10.1.3 Preparation for Transmission and Reception Tests

These tests require two evaluation boards (boards A and B) which are respectively connected to different development environments on different PCs. Evaluation boards A and B are connected to each other by a CAN cable*1 in their CAN 1 connectors.

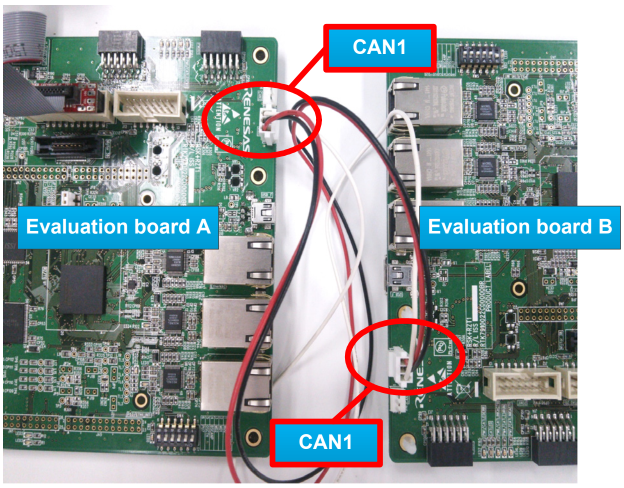


Figure 10.3 Configuration for Testing Transmission and Reception by a Sample Program

Note 1. The cable connects the CAN-H pins and CAN-L pins, respectively, of CAN connector 1 (J15) on boards A and B to each other.

10.1.4 Terminal Software (Tera Term)

In the sample program, data are transferred between a COM port of the host PC and the RS-232C interfaces of the boards by using the synchronous communications protocol of the serial communications interface with FIFO (SCIFA).

Start up the Tera Term terminal software on the host PC and configure the serial ports for a baud rate of 115200 with CR as the new line character.

- Transfer rate : 115200 bps
- Character lengths : 8 bits
- Stop bit length : 1 bit
- Parity : None
- Hardware flow control : Not supported

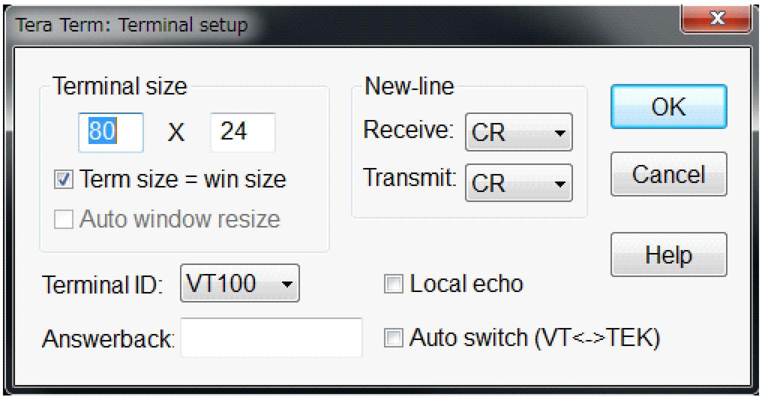


Figure 10.4 Terminal Setup of Tera Term

An example of serial port setup with "COM4" is shown below.

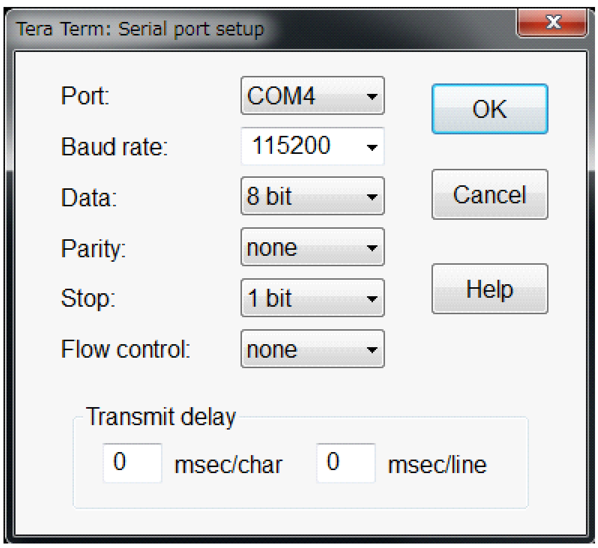


Figure 10.5 Serial Port Setup of Tera Term

10.1.5 Sample Program Menu

Start up the Tera Term terminal software and then start up the sample program. Select the program you want to use from the main menu on the console window.

Main menu

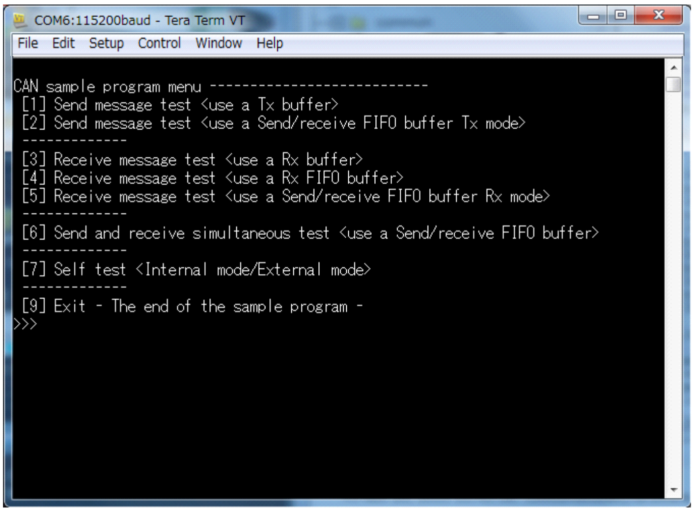


Figure 10.6 Main Menu of the Sample Program

Content of each menu item is described below.

[1] Send message test <uses a Tx buffer>

This is a test of the transmission of a message from a transmission buffer.

[2] Send message test <uses a Send/receive FIFO buffer Tx mode>

This is a test of the transmission of a message from a transmission-and-reception FIFO buffer.

[3] Receive message test <uses a Rx buffer>

This is a test of the reception of a message at a reception buffer.

[4] Receive message test <uses a Rx FIFO buffer>

This is a test of the reception of a message at a reception FIFO buffer.

[5] Receive message test <uses a Send/receive FIFO buffer Rx mode>

This is a test of the reception of a message at a transmission-and-reception FIFO buffer in reception mode.

[6] Send and receive simultaneous test <uses a Send/receive FIFO buffer >

This is a test of the transmission and reception of messages at the same time.

[7] Self-test <Internal mode/External mode>

This is a menu item for selecting self-tests.

[9] Exit – The end of the sample program –

Select this function to exit the sample program.

Self-test menu

This is a set of self-tests of operation in external and internal loopback modes by using a single evaluation board (evaluation board A). The modes are switched by the menu.

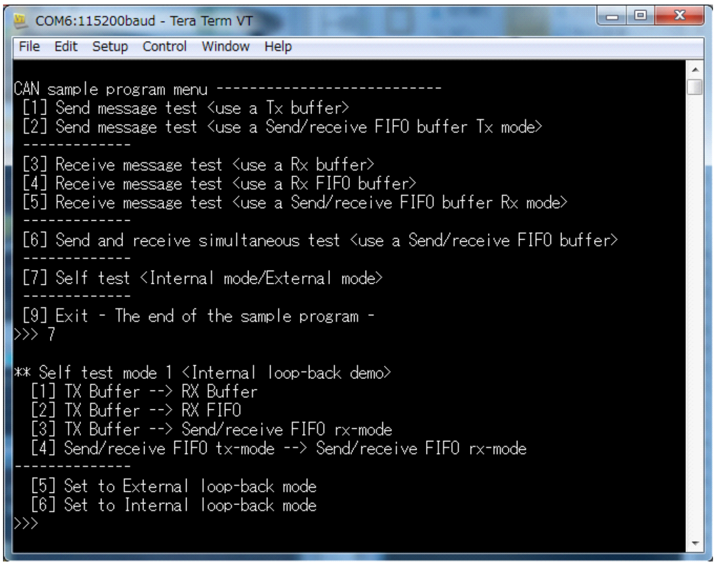


Figure 10.7 Selection of Self-Tests after Selecting Menu Item “[7] Self-Test”

Content of each self-test is described below.

[1] Tx Buffer → Rx Buffer

This is a test of the transmission of a message from a transmission buffer and reception of it at a reception buffer.

[2] Tx Buffer → Rx FIFO buffer

This is a test of the transmission of a message from a transmission buffer and reception of it at a reception FIFO buffer.

[3] Tx Buffer → Send/receive FIFO buffer Rx mode

This is a test of the transmission of a message from a transmission buffer and reception of it at a transmission-and-reception FIFO buffer in reception mode.

[4] Send/receive FIFO buffer Tx mode → Send/receive FIFO buffer Rx mode

This is a test of the transmission of a message from a transmission-and-reception FIFO buffer in reception mode and reception of it at a transmission-and-reception FIFO buffer in reception mode.

[5] Set to External loop back mode

Self-test mode 0 (external loopback mode) is selected.

[6] Set to Internal loop back mode

Self-test mode 1 (internal loopback mode) is selected.

10.1.6 Setting Values for the Sample Program

This sample program runs with the following settings:

Channel: CAN0 (fixed)
Baud rate: 1 Mbps (fixed)
Transmission message: Repetition of a 8-byte message
Reception rule: 1 (reception of the messages with the ID 0x120 only) (fixed)
Transmission buffer number: 0 (fixed)
Reception buffer number: 1 (fixed)
Reception FIFO buffer number: 0 (fixed)
Number of the transmission-and-reception FIFO buffer in transmission mode: 0 (fixed)
Number of the transmission-and-reception FIFO buffer in reception mode: 1 (fixed)
Number of the transmission buffer which is linked to the transmission-and-reception FIFO buffer: 2 (fixed)

Sample data settings:

Message ID: 0x120 (fixed)
Message type: Standard ID (fixed)
Data format: Data frame (fixed)
Message data size: 00h to FFh (8 bytes of data are transmitted in one message)

Reception buffer setting:

Buffer size: 1024 bytes. Data in excess of this size are overwritten from the beginning of the buffer.

10.1.7 Transmission Test

Preparation

Connect evaluation boards A and B, which are respectively connected to different development environments on different PCs, with the CAN cable. See Section 10.1.3 for details.

Operating procedure

1. Set the receiving side (evaluation board B) in reception mode by selecting [3], [4], or [5] from the main menu.
2. Select [1] or [2] from the main menu on the sending side (evaluation board A).
3. Transmit messages sequentially from the sending side.
4. The transmission test is terminated by pressing any key on the sending side. The receiving side exits the reception mode at this time.

Transmission data settings:

Message ID: 0x120
Message type: standard ID (a value of 0)
Data format: data frame (a value of 0)
Message data size: 8 bytes of data are transmitted in one message (the 8-byte sequences from 0x00 to 0xFF are used in the sample program)
Delimiter code: message end code (transmission of 0x00 8 times consecutively is judged to indicate the end of a message)

Test result

The following shows an example of transmission test by using the transmission-and-reception FIFO buffer in transmission mode.

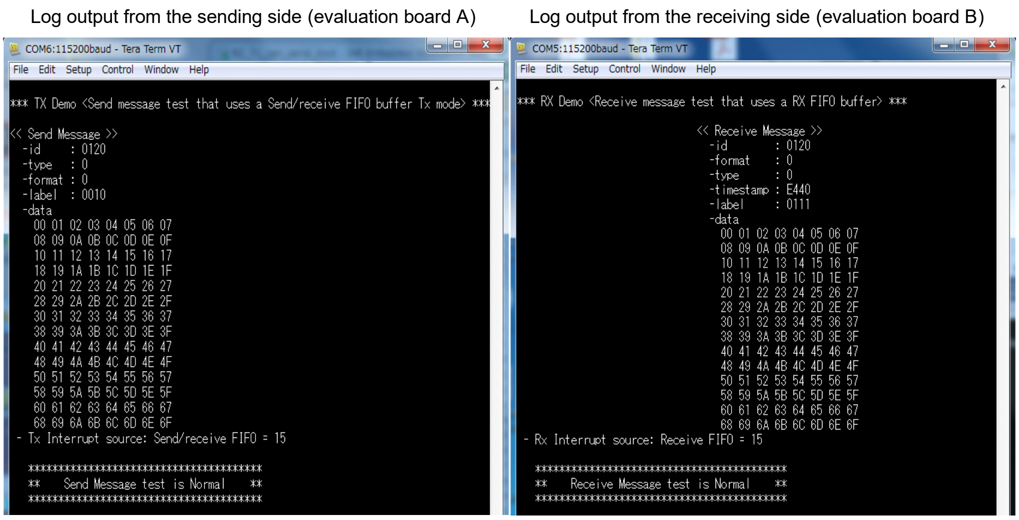


Figure 10.8 Example of Transmission Test Result

Note

In the operating procedure described earlier, if the messages are transmitted from the sending side (evaluation board A) while the receiving side (evaluation board B) has not entered the reception mode, an error will occur as shown in the window below. Be sure to start transmission of the test messages after completing preparation for the receiving side.

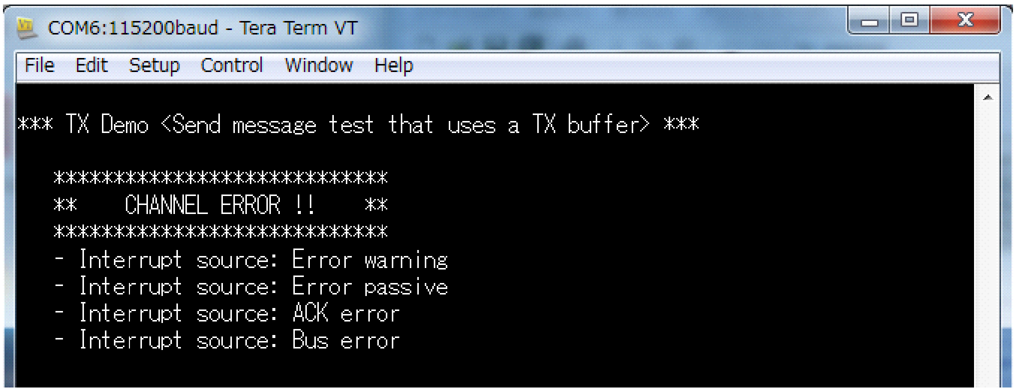


Figure 10.9 Example of Error in the Transmission Test

10.1.8 Reception Test

Preparation

Connect evaluation boards A and B, which are respectively connected to different development environment on different PCs, with the CAN cable. See Section 10.1.3 for details.

Operating procedure

1. Set the receiving side (evaluation board A) in reception mode by selecting [3], [4], or [5] from the main menu.
2. Select [1] or [2] from the main menu on the sending side (evaluation board B).
3. Transmit messages sequentially from the sending side.
4. The transmission test on the sending side is terminated by pressing any key on the sending side. The receiving side exits the reception mode at this time and the reception test is terminated accordingly.

Test result

The following shows an example of reception test by using the transmission-and-reception FIFO buffer in reception mode.

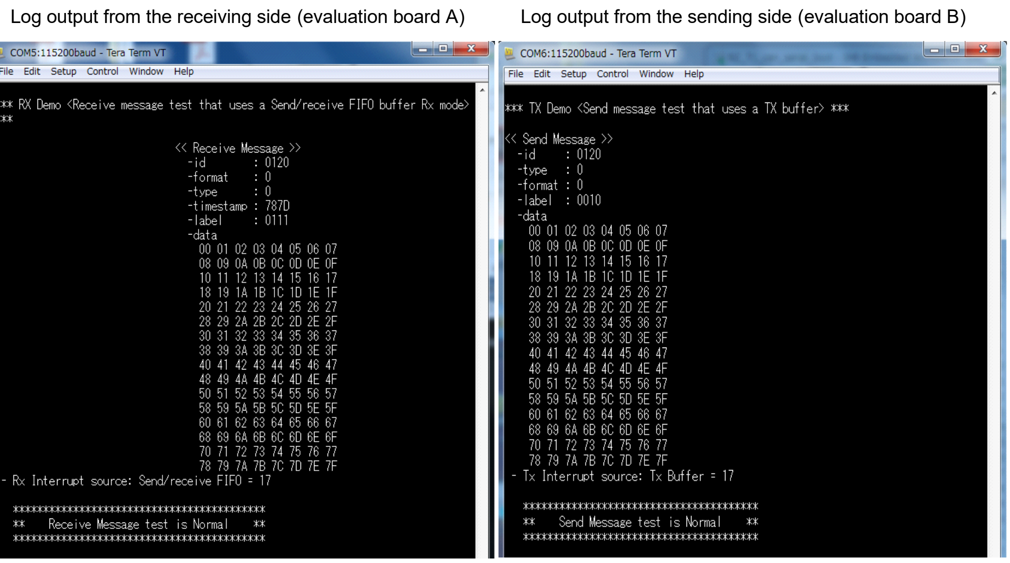


Figure 10.10 Example of Reception Test Result

10.1.9 Test for Transmission While Receiving Data at the Same Time

Preparation

Connect evaluation boards A and B, which are respectively connected to different development environment on different PCs, with the CAN cable. See Section 10.1.3 for details.

Operating procedure

(1) Select [6] from the main menu on the evaluation board A.

(2) A request that the user press any key is output.

(3) Select [6] from the main menu on the evaluation board B on the other PC.

(4) A request that the user press any key is output.

(5) Press any key on the evaluation board B after confirming that the request message was output on both sides.

By taking the procedure described above, the evaluation boards A and B sequentially transmit messages.

This test is terminated by pressing any key on either of the evaluation board A or B.

Test result

The following shows an example of transmission test while receiving a different message at the same time.

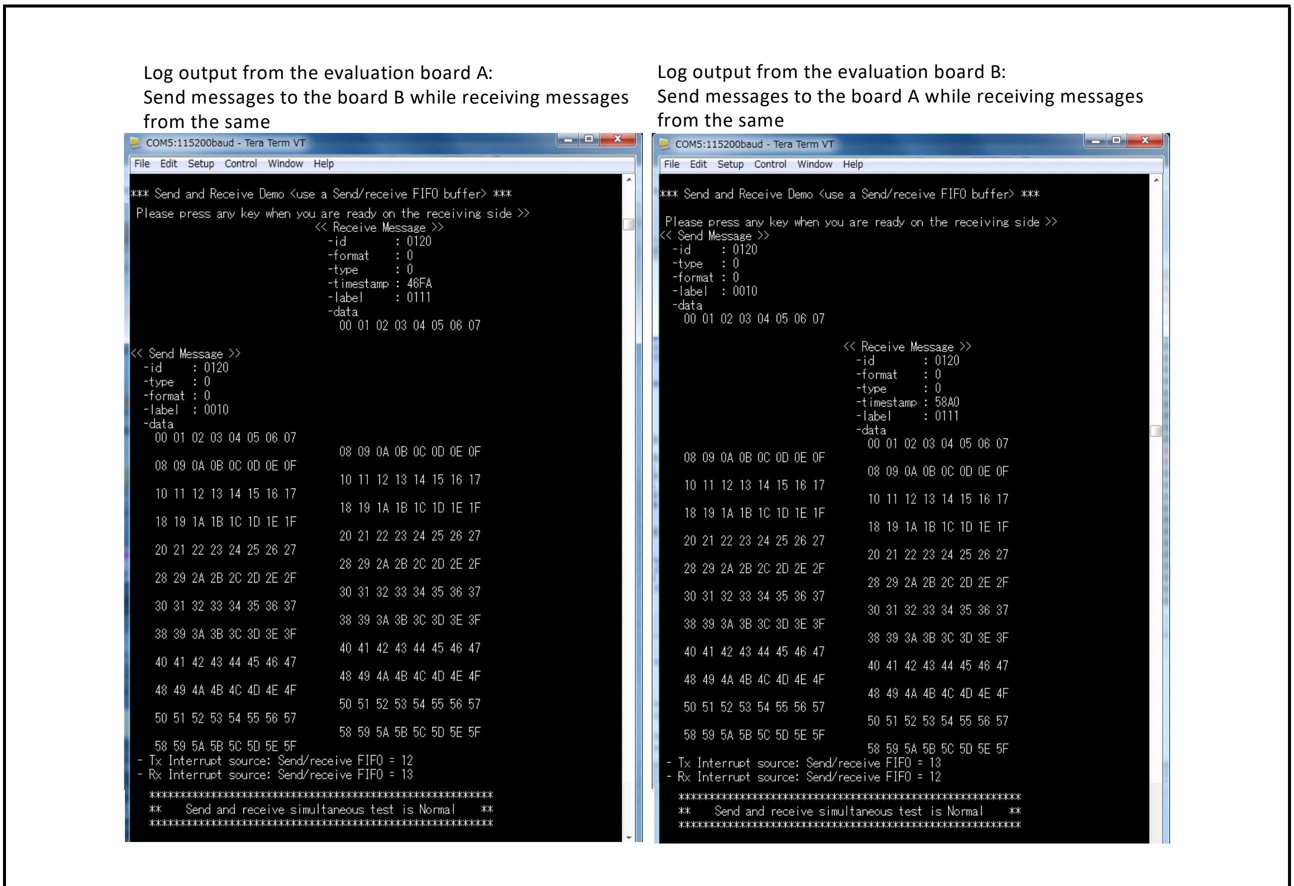


Figure 10.11 Example of Result of Transmission Test While Receiving Data

10.1.10 Self-Test

Preparation

This test requires only evaluation board A which is connected to a development environment. See Section 10.1.2 for details.

Operating procedure

1. Select [7] from the main menu for evaluation board A to show the self-test menu.
2. Select the operation you want to test from the menu, either [5] external loopback mode or [6] internal loopback mode (default).

Test result

The following shows an example of self-test by transmitting a message from a transmission buffer and receiving it at the transmission-and-reception FIFO buffer in reception mode.

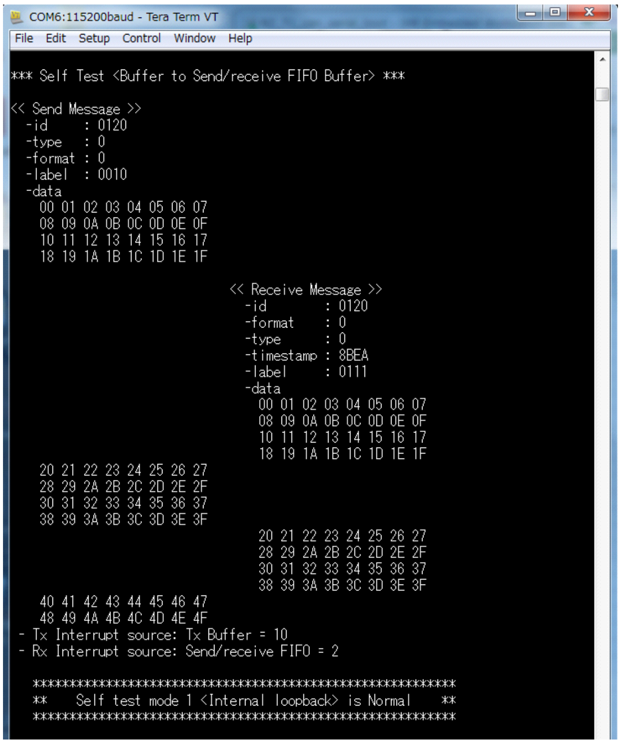


Figure 10.12 Example of Self-Testing Result

10.2 Interrupts

Table 10.2 lists the interrupts used in the sample code.

Table 10.2 Interrupts Used in the Sample Code

Interrupt (source ID)	Priority	Processing Outline
CAN global error (CANGE)	CAN_IR_PRIORITY_262_CANERR_GL	Generation of a global error on detection of the following sources (vector number 262): • DLC error • FIFO message loss • Transmission history buffer overflow
CAN0 error (CANIE0)	CAN_IR_PRIORITY_263_CANERR_CH0	Generation of a channel 0 error on detection of the following sources: (vector number 263): • Channel bus error • Error warning state • Error passive state • Bus-off entry • Bus-off recovery • Overload • Channel bus lockup • Arbitration lost • Stuff error • Form error • ACK error • CRC error • Recessive bit error • Dominant bit error • ACK delimiter error
CAN1 error (CANIE1)	CAN_IR_PRIORITY_264_CANERR_CH1	Generation of a channel 0 error on detection of the same sources as above (vector number 264)
CAN reception FIFO (CANRFI)	CAN_IR_PRIORITY_104_CANRFI	Message reception by using the reception FIFO buffers (vector number 104)
CAN0 transmission-and-reception FIFO reception completion (CANFIR0)	CAN_IR_PRIORITY_105_CANFIR0	Message reception by using the transmission-and-reception FIFO buffers in reception mode (vector number 105)
CAN0 transmission (CANTI0)	CAN_IR_PRIORITY_106_CANTI0	End of transmission on detection of the following source conditions (vector number 106): • Transmission completion • Transmission abort completion • Transmission interrupt request for the transmission-and-reception FIFO in transmission mode
CAN1 transmission-and-reception FIFO reception completion (CANFIR1)	CAN_IR_PRIORITY_107_CANFIR1	Message reception by using the transmission-and-reception FIFO in reception mode (vector number 107)
CAN1 transmission (CANTI1)	CAN_IR_PRIORITY_108_CANTI1	End of transmission on detection of the following source conditions (vector number 108): • Transmission completion • Transmission abort completion • Transmission interrupt request for the transmission-and-reception FIFO in transmission mode

10.3 Fixed-Width Integer Types

Table 10.3 lists the fixed-width integer types used for the sample code. These types are defined in the standard library.

Table 10.3 Fixed-Width Integer Types Used for the Sample Code

Symbol	Description
int8_t	8-bit signed integer
int16_t	16-bit signed integer
int32_t	32-bit signed integer
int64_t	64-bit signed integer
uint8_t	8-bit unsigned integer
uint16_t	16-bit unsigned integer
uint32_t	32-bit unsigned integer
uint64_t	64-bit unsigned integer

10.4 Constants and Error Codes

Table 10.4 lists the constants to be used in the sample program.

Table 10.4 Constants to be Used in the Sample Program (1 / 4)

Constant Name	Setting Value	Description
CAN_NUM	2	The number of channels of the CAN module
CAN_CH_0	0	Channel 0 (CAN0)
CAN_CH_1	1	Channel 1 (CAN1)
CH_BUFFER_MAX	16	The number of transmission buffers available for each channel
CH_FIFO_BUFFER_MAX	3	The number of transmission-and-reception FIFO buffers available at each channel
DATA_MAX	8	The number of message data that can be transmitted at the same time
CAN_TX_BUFFER	0	A state flag (the transmission buffers are in use)
CAN_TX_FIFO	1	A state flag (the transmission-and-reception FIFO buffers are in use in transmission mode)
CAN_TX_HISTORY	2	A state flag (for transmission history)
CAN_TX_QUEUE	3	A state flag (for transmission queue)
CAN_RX_BUFFER	0	A state flag (reception buffers are in use)
CAN_RX_RX_FIFO	1	A state flag (reception FIFO buffers are in use)
CAN_RX_FIFO	2	A state flag (transmission-and-reception FIFO buffers are in use in reception mode)
CAN_MODULE_ON	0	Exits the stop state
CAN_MODULE_OFF	1	Enters the stop state
CAN_STANDARD	0	Standard ID
CAN_EXTENDED	1	Extended ID
CAN_DATA_FRAME	0	Data frame
CAN_REMOTE_FRAME	1	Remote frame
CAN_RULE_PAGE_MAX	8	The maximum number of the reception rule pages
CAN_RULE_TABLE_MAX	16	The maximum number of the reception rule tables
CAN_RX_FIFO_BUFFER_MAX	8	The maximum number of the reception FIFO buffers
CAN_RX_BUFFER_MAX	32	The maximum number of the reception buffers
CAN_RULE_NUM_MAX	64	The maximum number of the reception rules
CAN_RX_MODE	0	Reception mode
CAN_TX_MODE	1	Transmission mode
CAN_GATEWAY_MODE	2	Gateway mode
CAN_FIFO_MSG_0	0	The number of the transmission-and-reception FIFO buffer stages (0 message)
CAN_FIFO_MSG_4	1	The number of the transmission-and-reception FIFO buffer stages (4 messages)
CAN_FIFO_MSG_8	2	The number of the transmission-and-reception FIFO buffer stages (8 messages)
CAN_FIFO_MSG_16	3	The number of the transmission-and-reception FIFO buffer stages (16 messages)
CAN_FIFO_MSG_32	4	The number of the transmission-and-reception FIFO buffer stages (32 messages)
CAN_FIFO_MSG_48	5	The number of the transmission-and-reception FIFO buffer stages (48 messages)
CAN_FIFO_MSG_64	6	The number of the transmission-and-reception FIFO buffer stages (64 messages)

Table 10.4 Constants to be Used in the Sample Program (2 / 4)

Constant Name	Setting Value	Description
CAN_FIFO_MSG_128	7	The number of the transmission-and-reception FIFO buffer stages (128 messages)
GL_MODE_STOP	0	Global stop mode
GL_MODE_RESET	1	Global reset mode
GL_MODE_TEST	2	Global test mode
GL_MODE_OPE	3	Global operation mode
CAN_GL_OPE	0	Enters the global operating mode
CAN_GL_RESET	1	Enters the global reset mode
CAN_GL_TEST	2	Enters the global test mode
CH_MODE_STOP	0	Channel stop mode
CH_MODE_RESET	1	Chanel reset mode
CH_MODE_WAIT	2	Channel halt mode
CH_MODE_COMM	3	Channel transfer mode
CAN_CH_COMM	0	Enters the channel transfer mode
CAN_CH_RESET	1	Enters the channel reset mode
CAN_CH_WAIT	2	Enters the channel halt mode
GL_TEST_RAMTEST	0	RAM test
GL_TEST_COMMTEST	1	Inter-channels transfer test
CH_TEST_STANDARD	0	Standard test mode
CH_TEST_LISTENONLY	1	Listen-only mode
CH_TEST_SELF0	2	Self-test mode 0 (external loopback mode)
CH_TEST_SELF1	3	Self-test mode 1 (internal loopback mode)
CANCLKA_CLK	24000000µ	CAN clock runs at 24 MHz
CANCLKB_CLK	25000000µ	CAN clock runs at 25 MHz
CAN_INTR_DISABLE	0	Interrupt is disabled
CAN_INTR_ENABLE	1	Interrupt is enabled
CAN_IR_PRIORITY_262_CANERR_GL	3	Priority order (CAN global error)
CAN_IR_PRIORITY_263_CANERR_CH0	4	Priority order (CAN0 error)
CAN_IR_PRIORITY_264_CANERR_CH1	4	Priority order (CAN1 error)
CAN_IR_PRIORITY_104_CANRFI	5	Priority order (CAN reception FIFO)
CAN_IR_PRIORITY_105_CANFIR0	5	Priority order (CAN0 transmission-and-reception FIFO reception completion)
CAN_IR_PRIORITY_106_CANTI0	5	Priority order (CAN0 transmission)
CAN_IR_PRIORITY_107_CANFIR1	5	Priority order (CAN1 transmission-and-reception FIFO reception completion)
CAN_IR_PRIORITY_108_CANTI1	5	Priority order (CAN1 transmission)
CAN_HVA_WRITE_DATA	0µ	HVA write data
CAN_OK	0µ	Returned value for successful operation
CAN_EMPTY	1µ	Returned value for the case of buffer empty
CAN_NG	0xFFFFFFFFµ	Returned value when an error occurred
CAN_INTR_TX_END	1	A source for the channel transmission interrupt: transmission completion
CAN_INTR_ABORT_END	2	A source for the channel transmission interrupt: abort transmission completion
CAN_INTR_FIFO_REQ	3	A source for the channel transmission interrupt: completion of transmission from the transmission-and-reception FIFO in transmission mode

Table 10.4 Constants to be Used in the Sample Program (3 / 4)

Constant Name	Setting Value	Description
CAN_INTR_QUEUE_REQ	4	A source for the channel transmission interrupt: transmission queue request is issued
CAN_INTR_HISTORY_REQ	5	A source for the channel transmission interrupt: transmission history request is issued
CAN_INTR_FIFO_EMPTY	1	Reception FIFO buffer empty
CAN_INTR_FIFO_FULL	2	Reception FIFO buffer full
CAN_INTR_FIFO_LOST	3	Reception FIFO buffer message lost
CAN_INTR_FIFO_TX_MESSAGE	4	Transmission-and-reception FIFO transmission interrupt request
CAN_INTR_FIFO_RX_MESSAGE	5	Transmission-and-reception FIFO reception interrupt request
CAN_BUS_ERR	1	Error flag (bus error)
CAN_ERR_WARNING	2	Error flag (error warning)
CAN_ERR_PASSIVE	3	Error flag (error passive)
CAN_BUS_OFF_START	4	Error flag (bus-off entry)
CAN_BUS_OFF_RETURN	5	Error flag (bus-off recovery)
CAN_OVER_LOAD	6	Error flag (overload)
CAN_BUS_LOCK	7	Error flag (channel bus lockup)
CAN_ARBITRATION_LOST	8	Error flag (arbitration lost)
CAN_STAFF_ERR	9	Error flag (staff error)
CAN_FORM_ERR	10	Error flag (form error)
CAN_ACK_ERR	11	Error flag (ACK error)
CAN_CRC_ERR	12	Error flag (CRC error)
CAN_RECESSIVE_BIT_ERR	13	Error flag (recessive bit error)
CAN_DOMINANT_BIT_ERR	14	Error flag (dominant bit error)
CAN_ACK_DELIMITER_ERR	15	Error flag (ACK delimiter error)
CAN_DLC_ERR	1	Error flag (DLC error)
CAN_FIFO_MSG_LOST_ERR	2	Error flag (FIFO message lost)
CAN_HISTORY_OVERFLOW_ERR	3	Error flag (transmission history buffer overflow)
CAN0_CRXD0_P30_VAL	0x10	MPC: setting value for the CAN0 CRXD0 (not used in this sample program)
CAN0_CRXD0_PC6_VAL	0x10	MPC: setting value for the CAN0 CRXD0
CAN0_CTXD0_P60_VAL	0x10	MPC: setting value for the CAN0 CTXD0 (not used in this sample program)
CAN0_CTXD0_P67_VAL	0x10	MPC: setting value for the CAN0 CTXD0
CAN1_CRXD1_PC3_VAL	0x10	MPC: setting value for the CAN1 CRXD1 (not used in this sample program)
CAN1_CRXD1_PC7_VAL	0x10	MPC: setting value for the CAN1 CRXD1
CAN1_CTXD1_P61_VAL	0x10	MPC: setting value for the CAN1 CTXD1 (not used in this sample program)
CAN1_CTXD1_P66_VAL	0x10	MPC: setting value for the CAN1 CTXD1
CAN1_CTXD1_PB3_VAL	0x10	MPC: setting value for the CAN1 CTXD1 (not used in this sample program)
CAN_GL_STATUS_BIT	0x00000007µ	RSCAN0GSTS register mask bit
CAN_CH_STATUS_BIT	0x00000007µ	RSCAN0CmSTS register mask bit
GCFG_REG_INIT	0x00000013µ	The initial value for the RSCAN0GCFG register
TMIEC0_REG_DISABLE_LOW	0x0000FFFFµ	TMIEp (p = 15 to 0) mask bit of the RSCAN0TMIEC0 register
TMIEC0_REG_DISABLE_HIGH	0xFFFF0000µ	TMIEp (p = 31 to 16) mask bit of the RSCAN0TMIEC0 register
CAN_CH_STOP_MODE	0x00000004µ	RSCAN0CmCTR register channel stop mode

Table 10.4 Constants to be Used in the Sample Program (4 / 4)

Constant Name	Setting Value	Description
CAN_REL_CH_STOP_MODE	0xFFFFFFFBμ	Release the module from the RSCAN0CmCTR register channel stop mode
TIME_QUANTUM_MIN	8	(*1) Value range for the bit time Set the value within the range obtained by SS + TSEG1 + TSEG2 = 8 to 25 Tq
TIME_QUANTUM_MAX	25	(*1) Value range for the bit time Set the value within the range obtained by SS + TSEG1 + TSEG2 = 8 to 25 Tq
SAMPLE_POINT	0.666666667	(*1) Sample point (%) The two third of one-bit communication frame is set as the sampling point in this sample program.
FIFO_UPDATE	0x000000FFμ	The value for controlling the pointer to the FIFO buffers

Note 1. Refer to Section 6.3.1, CAN Bit Time Setting of this application note and *Section 35.9.1.2 Bit Timing Setting* of the *RZ/T1 Group User's Manual: Hardware* for details.

10.5 Functions

Table 10.5 lists the functions used in this sample program.

Table 10.5 List of Functions

Function Name	Description
R_CAN_Open	Starts up the CAN module
R_CAN_Close	Stops the CAN module
R_CAN_GlobalControl	Makes a transition between the global modes
R_CAN_ChannelControl	Makes a transition between the channel modes
R_CAN_SetBitrate	Sets the transfer rates
R_CAN_UseBufferEntry	Registers the information of the buffers for use in transmission and reception
R_CAN_SetRxFifoBuffer	Enables the reception FIFO buffer
R_CAN_SetFifoBuffer	Enables the transmission-and-reception FIFO buffer
R_CAN_ReleaseFifoBuffer	Releases the transmission-and-reception FIFO buffer
R_CAN_ReleaseRxFifoBuffer	Releases the reception FIFO buffer
R_CAN_ReleaseBuffer	Releases the transmission buffer or the reception buffer
R_CAN_GetTxBufferStatus	Reads the state of the transmission buffer
R_CAN_WriteBuffer	Writes messages to be transmitted to the transmission buffer
R_CAN_GetFifoStatus	Reads the state of the transmission-and-reception FIFO buffer
R_CAN_WriteFifo	Writes messages to be transmitted to the transmission-and-reception FIFO buffer
R_CAN_Tx	Starts transmission
R_CAN_RxSet	Makes settings for reception
R_CAN_ReadBuff	Reads received messages from the reception buffer
R_CAN_ReadRxFifo	Reads received messages from the reception FIFO buffer
R_CAN_ReadFifo	Reads received messages from the transmission-and-reception FIFO buffer
R_CAN_GetFifoMessageNum	Get the number of unread messages in the transmission-and-reception FIFO buffer
R_CAN_GetRxFifoMessageNum	Get the number of unread messages in the reception FIFO buffer
R_CAN_SetCommTestMode	Makes settings for transfer tests
R_CAN_ResetTestMode	The module is released from the test mode and enters the channel transfer mode
R_CAN_SetInterruptHandler	Registers the interrupt handler
R_CAN_SetInterruptEnableDisable	Controls enabling and disabling of the CAN module interrupt vectors
R_CAN_GetInterruptSource	Gets the interrupt source
R_CAN_ClearInterruptSource	Clears the interrupt source
main	The main processing of the sample program

10.6 Structures/Unions/Enumerated Types

The following shows the structures, unions, and enumerated types used in this sample code.

- can_vector_t

  A structure for selecting whether to use the RSCAN interrupt vector or not

```
typedef struct {
    union {
        uint8_t     BYTE;
        struct {
            uint8_t     CANGE:1;     /* CAN global error interrupt */
            uint8_t     CANIE0:1;    /* CAN0 error interrupt */
            uint8_t     CANIE1:1;    /* CAN1 error interrupt */
            uint8_t     CANRFI:1;    /* CAN reception FIFO interrupt */
            uint8_t     CANFIR0:1;   /* CAN0 transmission-and-reception FIFO interrupt */
            uint8_t     CANTI0:1;    /* CAN0 transmission interrupt */
            uint8_t     CANFIR1:1;   /* CAN1 transmission-and-reception FIFO interrupt */
            uint8_t     CANTI1:1;    /* CAN1 transmission interrupt */
        } BIT;
    } SELECT;
} can_vector_t;
```

- can_callback_t

  A structure for registering a callback function

```
typedef struct {
    void    (*pintr_ge)(void);     /* Pointer to user callback function */
    void    (*pintr_ie0)(void);    /* Pointer to user callback function */
    void    (*pintr_ie1)(void);    /* Pointer to user callback function */
    void    (*pintr_rfi)(void);    /* Pointer to user callback function */
    void    (*pintr_fir0)(void);   /* Pointer to user callback function */
    void    (*pintr_ti0)(void);    /* Pointer to user callback function */
    void    (*pintr_fir1)(void);   /* Pointer to user callback function */
    void    (*pintr_ti1)(void);    /* Pointer to user callback function */
} can_callback_t;
```

- can_handle_t

  A structure for registering a callback function

```
typedef struct {
    bool            ch_opened;
    can_callback_t  can_callback;
} can_handle_t;
```

- can_rx_rule_t
  A structure for the reception rule table

```
typedef struct {
    uint32_t    buf_type;       /* Type of the buffer */
                                /* Transmission: CAN_TX_BUFFER, CAN_TX_FIFO */
                                /* Reception: CAN_RX_BUFFER, CAN_RX_RX_FIFO, CAN_RX_FIFO */
    uint32_t    rule_page;      /* Reception rule page number */
    uint32_t    rule_table;     /* Reception rule table number */
    uint32_t    rule_id;        /* Message ID */
    uint32_t    rule_type;      /* Message type (data frame/remote frame) */
    uint32_t    rule_format;    /* Message format (standard ID/extended ID) */
    uint32_t    rule_label;     /* Message label */
    uint32_t    rule_dlc_check; /* DLC checking */
    uint32_t    rule_mask;      /* Mask */
} can_rx_rule_t;
```

- udata_t
  A union of four-byte long types

```
typedef union udata {
    uint32_t    LONG;
    uint8_t     BYTE[4];
} udata_t;
```

- can_tx_message_t
  A structure of transmission message data

```
typedef struct {
    uint32_t    id;             /* Message ID */
    uint32_t    type;           /* 0: Data frame/1: Remote frame */
    uint32_t    format;         /* 0: Standard ID/1: Extended ID */
    uint32_t    length;         /* Message data length */
    udata_t     data_h;         /* Message data */
    udata_t     data_l;         /* Message data */
    uint32_t    history_label;  /* Label */
    uint32_t    buf_type;       /* Type of the buffer to be used (buffer or FIFO) */
} can_tx_message_t;
```

- can_rx_message_t
  A structure of reception message data

```
typedef struct {
    uint32_t    id;         /* Message ID */
    uint32_t    format;     /* 0: Standard ID/1: Extended ID */
    uint32_t    type;       /* 0: Data frame/1: Remote frame */
    uint16_t    timestamp;  /* Timestamp data */
    uint16_t    label;      /* Label information */
    uint32_t    length;     /* Message length */
    udata_t     data_h;     /* Message data */
    udata_t     data_l;     /* Message data */
} can_rx_message_t;
```

- gl_err_source_t
  A structure for the global error source count

```
typedef struct {
    uint32_t    total_num;              /* Total count of the global errors */
    uint32_t    dlc_error_num;          /* DLC error count */
    uint32_t    fifo_message_lost_num;  /* FIFO message loss count */
    uint32_t    history_overflow_num;   /* Transmission history overflow count */
} gl_err_source_t;
```

- ch_err_source_t
  A structure for the channel error source count

```
typedef struct {
    uint32_t    total_num;                  /* Total count of the channel errors */
    uint32_t    bus_error_num;              /* Bus error count */
    uint32_t    error_warning_num;          /* Error warning count */
    uint32_t    error_passive_num;          /* Error passive count */
    uint32_t    bus_off_start_num;          /* Bus-off start count */
    uint32_t    bus_off_return_num;         /* Bus-off return count */
    uint32_t    overload_num;               /* Overload count */
    uint32_t    bus_lock_num;               /* Bus lock-up count */
    uint32_t    arbitration_lost_num;       /* Arbitration loss count */
    uint32_t    staff_error_num;            /* Staff error count */
    uint32_t    form_error_num;             /* Form error count */
    uint32_t    ack_error_num;              /* ACK error count */
    uint32_t    crc_error_num;              /* CRC error count */
    uint32_t    recessive_bit_error_num;    /* Recessive bit error count */
    uint32_t    dominant_bit_error_num;     /* Dominant bit error count */
    uint32_t    ack_delimiter_error_num;    /* ACK delimiter error count */
} ch_err_source_t;
```

- ch_tx_source_t
  A structure for the transmission interrupt source count

```
typedef struct {
    uint32_t        total_num;          /* Total count of the transmission interrupts */
    uint32_t        intr_sorce;         /* Interrupt source */
    uint32_t        tx_buf_end_num;     /* Count of successful transmissions from the transmission buffer */
    uint32_t        tx_fifo_end_num;    /* Count of successful transmissions from the transmission-and-reception
                                           FIFO buffer in transmission mode */
    uint32_t        tx_abort_num;       /* Transmission abortion count*/
    uint32_t        tx_queue_num;       /* Transmission queue count */
    uint32_t        tx_history_num;     /* Transmission history data count */
} ch_tx_source_t;
```

- can_intr_source_t
  A structure for the information of the interrupt source

```
typedef struct {
    uint32_t          rx_fifo_num;          /* Reception FIFO interrupt count */
    uint32_t          ch_fifo_receive_num;  /* Transmission-and-reception FIFO reception interrupt count */
    ch_tx_source_t    tx_source;            /* Transmission interrupt source */
    gl_err_source_t   gl_err;               /* Global error interrupt source */
    ch_err_source_t   ch_err;               /* Channel error interrupt source */
} can_intr_source_t;
```

- can_used_buffer_t
  A structure for the information of the buffer in use

```
typedef struct {
    uint32_t        use_tx_buf_no;          /* Transmission buffer number */
    uint32_t        use_rx_buf_no;          /* Reception buffer number */
    uint32_t        use_rx_fifo_no;         /* Reception FIFO buffer number */
    uint32_t        use_fifo_txmode_no;     /* Number of the transmission-and-reception FIFO buffer in
                                               transmission mode */
    uint32_t        use_fifo_rxmode_no;     /* Number of the transmission-and-reception FIFO buffer in
                                               reception mode */
    uint32_t        use_fifo_link_buf_no;   /* Number of the buffer to which the FIFO buffer is linked to */
} can_used_buffer_t;
```

- can_tx_intr_sts_t
  A structure for the information of the state of the request for transmission interrupt

```
typedef struct {
    union {
        uint8_t     BYTE;
        struct {
            uint8_t     TMTRF:2;    /* Result of transmission from the transmission buffer */
            uint8_t     CFTXIF:1;   /* Request for transmission-and-reception FIFO transmission interrupt */
            uint8_t     TXQIF:1;    /* Request for transmission queue interrupt */
            uint8_t     THLIF:1;    /* Request for transmission history interrupt */
            uint8_t     :3;
        } BIT;
    } STS;
} can_tx_intr_sts_t;
```

- can_tx_history_t
  A structure for the information of the transmission history

```
typedef struct {
    uint32_t    buf_type;   /* Buffer type */
    uint32_t    buf_no;     /* Buffer number */
    uint32_t    label;      /* Label data */
} can_tx_history_t;
```

10.7 Function Specifications

Specifications of the functions used in the sample code are as follows:

10.7.1 R_CAN_Open

R_CAN_Open

Synopsis	This is the function used first when using the CAN module.
Header	r_can_api.h
Declaration	void R_CAN_Open(uint32_t ch, uint32_t frequency);
Description	This function makes initial settings for starting CAN communications. The channels and the transfer rate used for the communication are specified in the arguments. The following processes are required if the channel to be used has not been initialized. - Initializing the variables used with the API functions - Releasing the CAN module from the stop state - Setting the ports to input or output - Setting the CAN module to the global reset mode - Setting the selected channel to the channel reset mode - Initializing the CAN registers to be used for the CAN communication - Specifying the transfer rate for the CAN communication
Arguments	uint32_t ch : Channel number uint32_t frequency : Transfer rate
Returned value	None
Remarks	None

10.7.2 R_CAN_Close

R_CAN_Close

Synopsis	Stops the CAN communication and releases the CAN module.
Header	r_can_api.h
Declaration	void R_CAN_Close(uint32_t ch);
Description	This function makes settings for ending the current CAN communication. When executed, the channels specified by the arguments are disabled. The following operations are included: - Setting the CAN module to the stop state - Disabling the CAN interrupts Call R_CAN_Open( ) to restart communication after this function has been called. If the ongoing communication is forcibly stopped, the communication is not guaranteed.
Arguments	uint32_t ch : Channel number
Returned value	None
Remarks	None

10.7.3 R_CAN_GlobalControl

R_CAN_GlobalControl	
Synopsis	Controls the RSCAN module as a whole.
Header	r_can_api.h
Declaration	void R_CAN_GlobalControl(uint32_t mode);
Description	This function sets the RSCAN module to whichever of the following global modes is specified in the argument. - Global stop mode GThe clock for the whole module is stopped. Lower-power consumption is possible in this mode. - Global reset mode The initial settings for the whole CAN module are made in this mode. - Global test mode Tests (RAM test and inter-channels transfer test) are carried out in this mode. - Global operation mode Operation of the whole CAN module is enabled in this mode. The CAN module is normally in this mode.
Arguments	uint32_t mode : The module makes a transition to the specified global mode from the list below: GL_MODE_OPE: global operation mode GL_MODE_RESET: global reset mode GL_MODE_STOP: global stop mode GL_MODE_TEST: global test mode
Returned value	None
Remarks	None

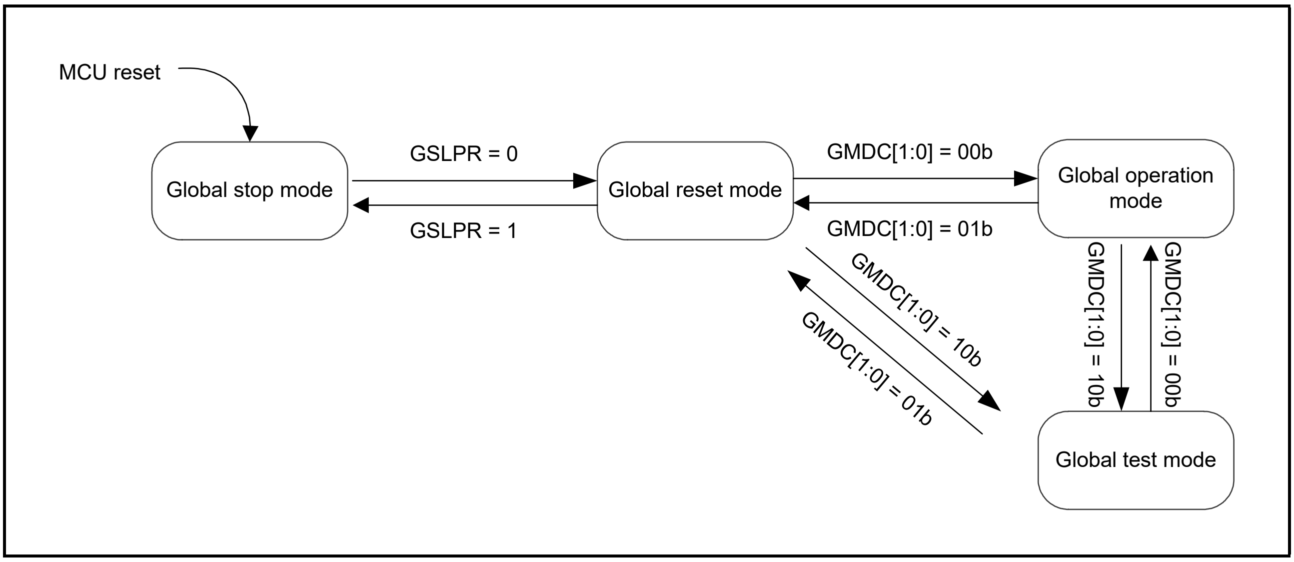


Figure 10.13 Transitions between Global Modes

10.7.4 R_CAN_ChannelControl

R_CAN_ChannelControl	
Synopsis	Controls channels.
Header	r_can_api.h
Declaration	void R_CAN_ChannelControl(uint32_t ch, uint32_t mode);
Description	This function sets the state of the selected channel to whichever of the following channel modes is specified in the arguments: - Channel stop mode The clock for the specified channel is stopped in this mode. - Channel reset mode The initial settings for the channel is made in this mode. - Channel halt mode The CAN module is halted and tests for the specified channels are enabled. - Channel transfer mode CAN communications are normally handled in this mode.
Arguments	uint32_t ch : Channel number uint32_t mode : The selected channel makes a transition to the specified channel mode from the list below: CH_MODE_STOP: channel stop mode CH_MODE_RESET: channel reset mode CH_MODE_WAIT: channel halt mode CH_MODE_COMM: channel transfer mode
Returned value	None
Remarks	None

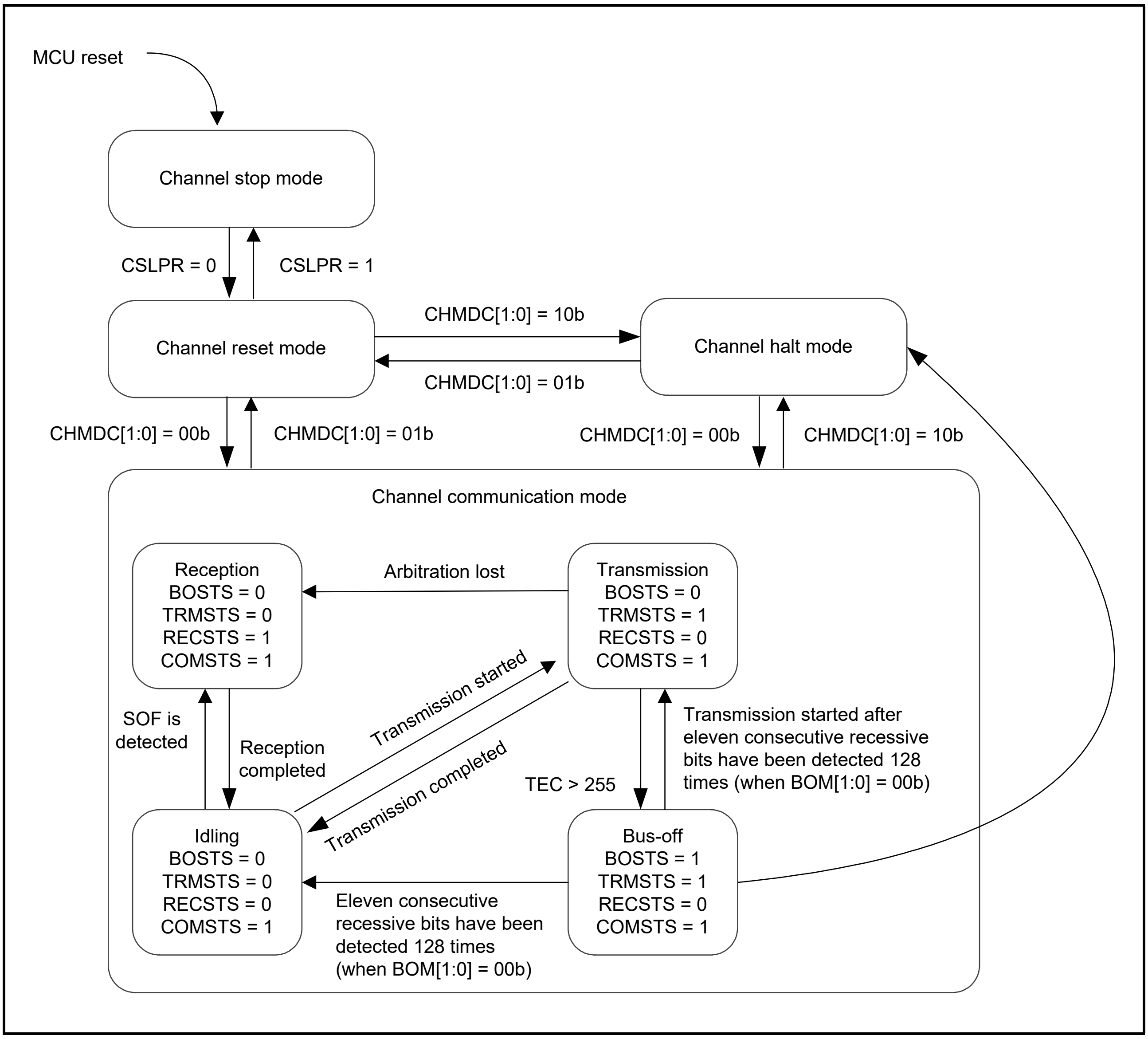


Figure 10.14 Transition Diagram among Channel Modes

10.7.5 R_CAN_SetBitrate

R_CAN_SetBitrate		
Synopsis	Specifies the transfer rate for the CAN communication.	
Header	r_can_api.h	
Declaration	void R_CAN_SetBitrate(uint32_t ch, uint32_t frequency);	
Description	This function specifies the transfer rate for the CAN communication by using the value set in the argument. See Section 6.3, Transfer Rate.	
Arguments	uint32_t ch	: Channel number
	uint32_t frequency	: Transfer rate BAUD_RATE_1MBPS: 1 Mbps BAUD_RATE_500KBPS: 500 Kbps BAUD_RATE_125KBPS: 125 Kbps
Returned value	None	
Remarks	None	

10.7.6 R_CAN_UseBufferEntry

R_CAN_UseBufferEntry		
Synopsis	Registers the information of the buffers for use in the CAN communications.	
Header	r_can_api.h	
Declaration	void R_CAN_UseBufferEntry(can_used_buffer_t * obj);	
Description	This function registers the following information of the buffers for use in the CAN communications. - Transmission buffer number - Reception buffer number - Reception FIFO buffer number - Number of the transmission-and-reception FIFO in transmission mode - Number of the transmission-and-reception FIFO in reception mode - Number of the buffer to which the transmission-and-reception FIFO in transmission mode links.	
Arguments	can_used_buffer_t * obj	: A pointer to the structure which holds the information related to buffers
Returned value	None	
Remarks	None	

10.7.7 R_CAN_SetRxFifoBuffer

R_CAN_SetRxFifoBuffer		
Synopsis	Enables the reception FIFO buffers.	
Header	r_can_api.h	
Declaration	void R_CAN_SetRxFifoBuffer(uint32_t ch, can_rfcc_t * obj);	
Description	This function enables reception of messages by using the reception FIFO buffers. A CAN reception FIFO interrupt occurs on reception of a message. Register the reception FIFO buffer number to be used for the CAN communication by the R_CAN_UseBufferEntry() function before calling this function. The received messages are read by using the R_CAN_ReadRxFifo() function. See Section 7.3, Reception by Using the Reception FIFO Buffers.	
Arguments	uint32_t ch	: Channel number
	can_rfcc_t * obj	: Information of the reception FIFO buffer
Returned value	None	
Remarks	None	

10.7.8 R_CAN_SetFifoBuffer

R_CAN_SetFifoBuffer		
Synopsis	Enables the transmission-and-reception FIFO buffers.	
Header	r_can_api.h	
Declaration	void R_CAN_SetFifoBuffer(uint32_t ch, uint32_t mode, can_cfcc_t * obj);	
Description	This function enables transmission and reception of messages by using the transmission-and-reception FIFO buffers. - Transmission mode A transmission completion interrupt occurs on completion of transmission of message from the transmission-and-reception FIFO buffer in transmission mode, with the source for the interrupt as completion of transmission. See Section 8.3, Transmission by Using the Transmission-and-Reception FIFO Buffers. - Reception mode A reception completion interrupt occurs on completion of reception of message at the transmission-and-reception FIFO buffer in reception mode. Register the transmission-and-reception FIFO buffers to be used for the CAN communication by using the R_CAN_UseBufferEntry() function before calling this function. The received messages are read by using the R_CAN_ReadFifo() function. See Section 7.4, Reception by Using the Transmission-and-Reception FIFO Buffers.	
Arguments	uint32_t ch	: Channel number
	uint32_t mode	: Modes CAN_TX_MODE: transmission mode CAN_RX_MODE: reception mode
	can_cfcc_t * obj	: Information of the transmission-and-reception FIFO buffer
Returned value	None	
Remarks	None	

10.7.9 R_CAN_ReleaseFifoBuffer

R_CAN_ReleaseFifoBuffer		
Synopsis	Releases the transmission-and-reception FIFO buffers used for the CAN communications.	
Header	r_can_api.h	
Declaration	void R_CAN_ReleaseFifoBuffer(uint32_t ch, uint32_t mode);	
Description	This function releases the transmission-and-reception FIFO buffers used for the CAN communications.	
Arguments	uint32_t ch	: Channel number
	uint32_t mode	: Modes CAN_TX_MODE: transmission mode CAN_RX_MODE: reception mode
Returned value	None	
Remarks	None	

10.7.10 R_CAN_ReleaseRxFifoBuffer

R_CAN_ReleaseRxFifoBuffer		
Synopsis	Releases the reception FIFO buffers used for the CAN communications.	
Header	r_can_api.h	
Declaration	void R_CAN_ReleaseRxFifoBuffer(uint32_t ch);	
Description	This function releases the reception FIFO buffer used for the CAN communications.	
Arguments	uint32_t ch	: Channel number
Returned value	None	
Remarks	None	

10.7.11 R_CAN_ReleaseBuffer

R_CAN_ReleaseBuffer		
Synopsis	Releases the buffers used for the CAN communications.	
Header	r_can_api.h	
Declaration	void R_CAN_ReleaseBuffer(uint32_t ch, uint32_t mode);	
Description	This function releases the buffers used for the CAN communications.	
Arguments	uint32_t ch	: Channel number
	uint32_t mode	: Mode CAN_TX_MODE: transmission mode CAN_RX_MODE: reception mode
Returned value	None	
Remarks	None	

10.7.12 R_CAN_GetTxBufferStatus

R_CAN_GetTxBufferStatus	
Synopsis	Reads the state of the transmission buffer.
Header	r_can_api.h
Declaration	uint32_t R_CAN_GetTxBufferStatus(uint32_t ch);
Description	This function is used for reading the state of the transmission buffer.
Arguments	uint32_t ch : Channel number
Returned value	0: No ongoing transmission 1: Transmission is in progress
Remarks	None

10.7.13 R_CAN_WriteBuffer

R_CAN_WriteBuffer	
Synopsis	Writes messages to the transmission buffer.
Header	r_can_api.h
Declaration	void R_CAN_WriteBuffer(uint32_t ch, can_tx_message_t * msg);
Description	This function is used for writing messages to the transmission buffer. The message ID, the data format, the data length, the label information, and the data to be transmitted are stored in the can_tx_message_t structure and set as an argument of this function. See Section 8.2, Transmission by Using the Transmission Buffers.
Arguments	uint32_t ch : Channel number can_tx_message_t * msg : Information of the transmission message
Returned value	None
Remarks	None

10.7.14 R_CAN_GetFifoStatus

R_CAN_GetFifoStatus	
Synopsis	Reads the state of the transmission-and-reception FIFO buffer.
Header	r_can_api.h
Declaration	uint32_t R_CAN_GetFifoStatus(uint32_t ch, uint32_t mode);
Description	This function is used for reading the state of the transmission-and-reception FIFO buffer.
Arguments	uint32_t ch : Channel number uint32_t mode : Modes CAN_TX_MODE: transmission mode CAN_RX_MODE: reception mode
Returned value	0: Transmission-and-reception FIFO buffer is not full 1: Transmission-and-reception FIFO buffer is full
Remarks	None

10.7.15 R_CAN_WriteFifo

R_CAN_WriteFifo	
Synopsis	Writes messages to the transmission-and-reception FIFO buffer in transmission mode.
Header	r_can_api.h
Declaration	void R_CAN_WriteFifo(uint32_t ch, can_tx_message_t * msg);
Description	This function is used for writing messages to the transmission-and-reception FIFO buffer in transmission mode. The message ID, the data format, the data length, the label information, and the data to be transmitted are stored in the can_tx_message_t structure and set as an argument of this function. See Section 8.3, Transmission by Using the Transmission-and-Reception FIFO Buffers.
Arguments	uint32_t ch : Channel number can_tx_message_t * msg : Information of the transmission message
Returned value	None
Remarks	None

10.7.16 R_CAN_Tx

R_CAN_Tx	
Synopsis	Starts transmission in CAN communication.
Header	r_can_api.h
Declaration	void R_CAN_Tx(uint32_t ch, can_tx_message_t * msg);
Description	This function starts transmission in CAN communication. - Transmission by using the transmission buffer Set the transmission request bit for the relative transmission buffer to 1 (transmission is requested). - Transmission by using the transmission-and-reception FIFO buffer in transmission mode Set the transmission-and-reception FIFO buffer enable bit to 1 (transmission-and-reception FIFO buffers are used)
Arguments	uint32_t ch : Channel number can_tx_message_t * msg : Information of the transmission message
Returned value	None
Remarks	None

10.7.17 R_CAN_RxSet

R_CAN_RxSet	
Synopsis	Enables reception.
Header	r_can_api.h
Declaration	void R_CAN_RxSet(uint32_t ch, can_rx_rule_t * rule);
Description	This function is used for setting rules for receiving messages. The information of the reception rules are stored in the can_rx_rule_t structure and set as an argument for the reception rule of this function. See Section 6.5, Reception Rule Table.
Arguments	uint32_t ch : Channel number can_rx_rule_t * rule : Information of the reception rule
Returned value	None
Remarks	None

10.7.18 R_CAN_ReadBuff

R_CAN_ReadBuff	
Synopsis	Reads messages from the reception buffer.
Header	r_can_api.h
Declaration	uint32_t R_CAN_ReadBuff(uint32_t ch, can_rx_message_t * obj);
Description	This function is used for reading messages from the reception buffers. See Section 7.2, Reception by Using the Reception Buffers.
Arguments	uint32_t ch : Channel number can_rx_message_t * obj : A pointer to the area where the reception messages are stored
Returned value	CAN_OK: Data are successfully read from the reception buffer. CAN_EMPTY: There are no new messages in the reception buffer.
Remarks	None

10.7.19 R_CAN_GetRxFifoMessageNum

R_CAN_GetRxFifoMessageNum	
Synopsis	Gets the number of unread messages from the reception FIFO buffer.
Header	r_can_api.h
Declaration	uint32_t R_CAN_GetRxFifoMessageNum(void);
Description	This function returns the number of unread messages in the reception FIFO buffer. See Section 7.3, Reception by Using the Reception FIFO Buffers.
Arguments	None
Returned value	The number of unread messages.
Remarks	None

10.7.20 R_CAN_ReadRxFifo

R_CAN_ReadRxFifo	
Synopsis	Reads the received messages from the reception FIFO buffer.
Header	r_can_api.h
Declaration	uint32_t R_CAN_ReadRxFifo(can_rx_message_t * obj);
Description	This function is used for reading messages from the reception FIFO buffer. See Section 7.3, Reception by Using the Reception FIFO Buffers.
Arguments	can_rx_message_t * obj : The area where the received messages are stored.
Returned value	CAN_OK: Data are successfully read from the reception FIFO buffer. CAN_EMPTY: There are no unread messages in the reception FIFO buffer (buffer empty). CAM_LOST: FIFO message lost.
Remarks	None

10.7.21 R_CAN_GetFifoMessageNum

R_CAN_GetFifoMessageNum	
Synopsis	Get the number of unread messages from the transmission-and-reception FIFO buffer.
Header	r_can_api.h
Declaration	uint32_t R_CAN_GetFifoMessageNum(uint32_t ch);
Description	This function returns the number of unread messages from the transmission-and-reception FIFO buffer. See Section 7.4, Reception by Using the Transmission-and-Reception FIFO Buffers.
Arguments	uint32_t ch : Channel number
Returned value	The number of unread messages
Remarks	None

10.7.22 R_CAN_ReadFifo

R_CAN_ReadFifo		
Synopsis	Reads the received messages from the transmission-and-reception FIFO buffer.	
Header	r_can_api.h	
Declaration	uint32_t R_CAN_ReadFifo(uint32_t ch, can_rx_message_t * obj);	
Description	This function is used for reading the messages from the transmission-and-reception FIFO buffer. See Section 7.4, Reception by Using the Transmission-and-Reception FIFO Buffers.	
Arguments	uint32_t ch	: Channel number
	can_rx_message_t * obj	: A pointer to the area where the received messages are stored.
Returned value	CAN_OK: Data are successfully read from the transmission-and-reception FIFO buffer. CAN_EMPTY: There are no unread messages in the transmission-and-reception FIFO buffer (buffer empty). CAM_LOST: FIFO message lost.	
Remarks	None	

10.7.23 R_CAN_SetCommTestMode

R_CAN_SetCommTestMode		
Synopsis	Selects the transfer test mode.	
Header	r_can_api.h	
Declaration	void R_CAN_SetCommTestMode(uint32_t ch, uint32_t mode);	
Description	This function is used for selecting a test mode from the following: - Standard test mode - Listen-only mode - Self-test mode 0 (external loopback mode) - Self-test mode 1 (internal loopback mode)	
Arguments	uint32_t ch	: Channel number
	uint32_t mode	: Test modes CH_TEST_STANDARD: Standard test mode CH_TEST_LISTENONLY: Listen-only test mode CH_TEST_SELF0: Self-test mode 0 (external loopback mode) CH_TEST_SELF1: Self-test mode 1 (internal loopback mode)
Returned value	None	
Remarks	None	

10.7.24 R_CAN_ResetTestMode

R_CAN_ResetTestMode	
Synopsis	Clears the transfer test state.
Header	r_can_api.h
Declaration	void R_CAN_ResetTestMode(uint32_t ch);
Description	This function clears the test mode set by using the R_CAN_SetCommTestMode() function. After the test, always clear the state by calling this function.
Arguments	uint32_t ch : Channel number
Returned value	None
Remarks	None

10.7.25 R_CAN_SetInterruptHandler

R_CAN_SetInterruptHandler	
Synopsis	Registers the callback function which is called by the interrupt handling routines used in the CAN communications.
Header	r_can_api.h
Declaration	void R_CAN_SetInterruptHandler(uint32_t ch, can_callback_t * pcallback);
Description	This function is used to register the callback function which is called by one of the following interrupt handling routines used in the CAN communications. - CAN global error - CAN0 error - CAN1 error - CAN reception FIFO - CAN0 transmission-and-reception FIFO reception completion - CAN0 transmission - CAN1 transmission-and-reception FIFO reception completion - CAN1 transmission * pcallback: A pointer to the structure which holds the information of the callback function. Write the callback function name to the member of this structure. Write null if the pointer is not used.
Arguments	uint32_t ch : Channel number can_callback_t * pcallback : Information of the callback function
Returned value	None
Remarks	None

10.7.26 R_CAN_SetInterruptEnableDisable

R_CAN_SetInterruptEnableDisable	
Synopsis	Controls enabling and disabling of the interrupt handler used in the CAN communication.
Header	r_can_api.h
Declaration	void R_CAN_SetInterruptEnableDisable(uint32_t enable_disable);
Description	This function controls enabling and disabling of the following interrupt handlers used in CAN communication. - CAN global error - CAN0 error - CAN1 error - CAN reception FIFO - CAN0 transmission-and-reception FIFO reception completion - CAN0 transmission - CAN1 transmission-and-reception FIFO reception completion - CAN1 transmission
Arguments	uint32_t enable_disable : Enables or disables CAN_INTR_DISABLE: Disables CAN_INTR_ENABLE: Enables
Returned value	None
Remarks	None

10.7.27 R_CAN_GetInterruptSource

R_CAN_GetInterruptSource	
Synopsis	Gets the interrupt source
Header	r_can_api.h
Declaration	void R_CAN_GetInterruptSource(can_intr_source_t * obj);
Description	This function returns an indicator of the source of an interrupt.
Arguments	can_intr_source_t * obj : Area where the information of the interrupt source is stored
Returned value	None
Remarks	None

10.7.28 R_CAN_ClearInterruptSource

R_CAN_ClearInterruptSource	
Synopsis	Clears the information of the interrupt source.
Header	r_can_api.h
Declaration	void R_CAN_ClearInterruptSource(void);
Description	This function clears the source for the corresponding interrupt.
Arguments	None
Returned value	None
Remarks	None

10.7.29 main

Main	
Synopsis	The main function of this sample program.
Header	—
Declaration	void main(void)
Description	This is the main processing for this sample program. See Section 10.8, Flowchart for detailed processing.
Arguments	None
Returned value	None
Remarks	None

10.8 Flowchart

10.8.1 Main Processing

In this sample program, the item the user wants to check is selected from the menus. See Section 10.1, Operational Outline for the menus.

Figure 10.15 is the flowchart for the main processing of this sample code.

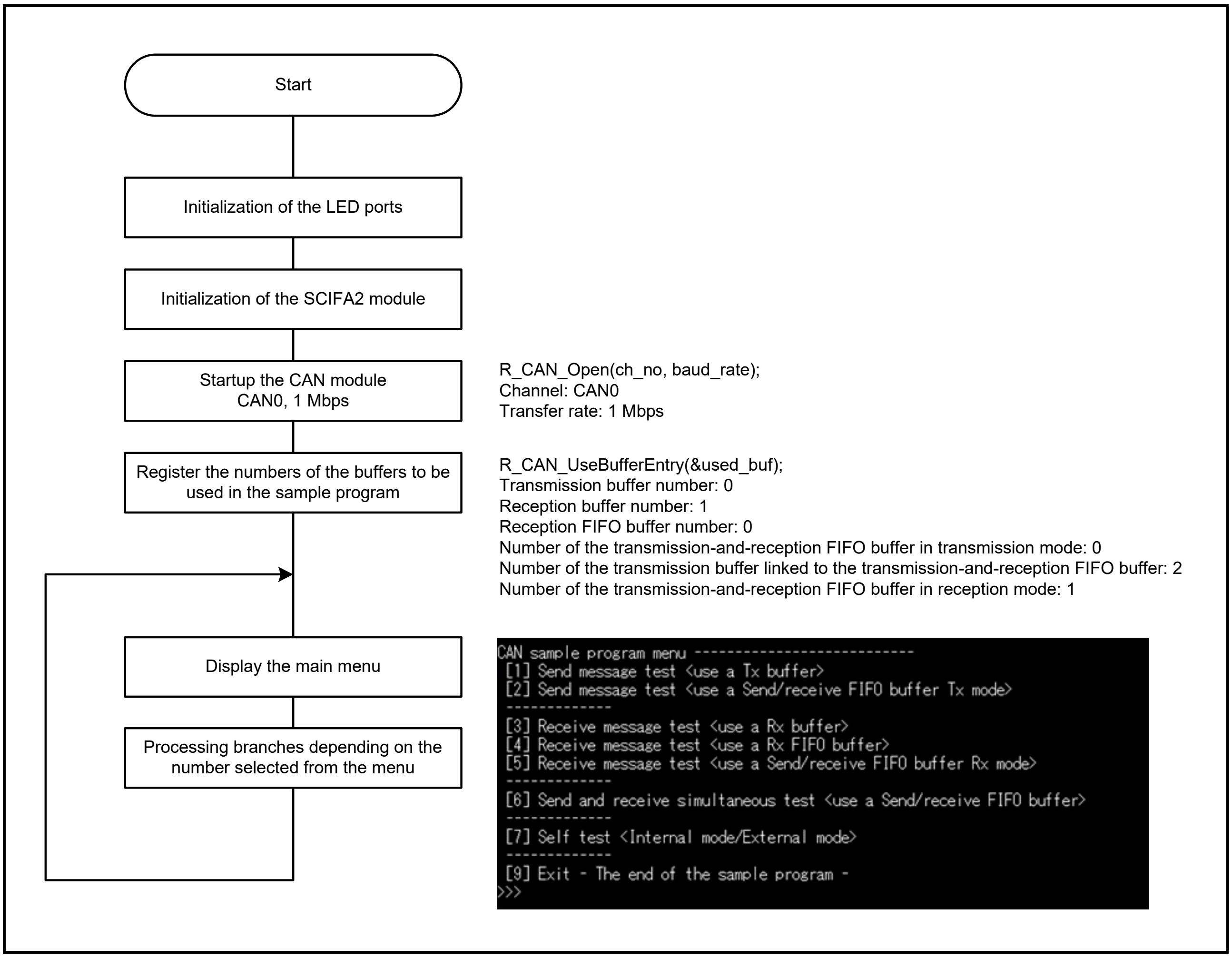


Figure 10.15 Main Processing in the Sample Code

10.8.2 Transmission Test

Two types of the transmission tests, transmission by using the transmission buffer and by using the transmission-and-reception FIFO buffer in transmission mode are available from the menu.

See Section 10.1, Operational Outline for the menus.

The following functions are used for performing each test.

(1) void tx_demo_buffer(void)
Message transmission by using the transmission buffer.

(2) void tx_demo_fifo(void)
Message transmission by using the transmission-and-reception FIFO buffer in transmission mode.

(3) uint32_t write_buffer(uint32_t msg_type, tx_data_t * obj)
Writing transmission messages to the registers related to the transmission buffer.

(4) uint32_t write_fifo(uint32_t msg_type, tx_data_t * obj)
Writing transmission messages to the registers related to the transmission-and-reception FIFO buffer.

Figure 10.16 to Figure 10.19 show the flowcharts for processing by the respective functions.

(1) Message transmission by using the transmission buffer
Function name: void tx_demo_buffer(void)
A flowchart for this test is shown below.

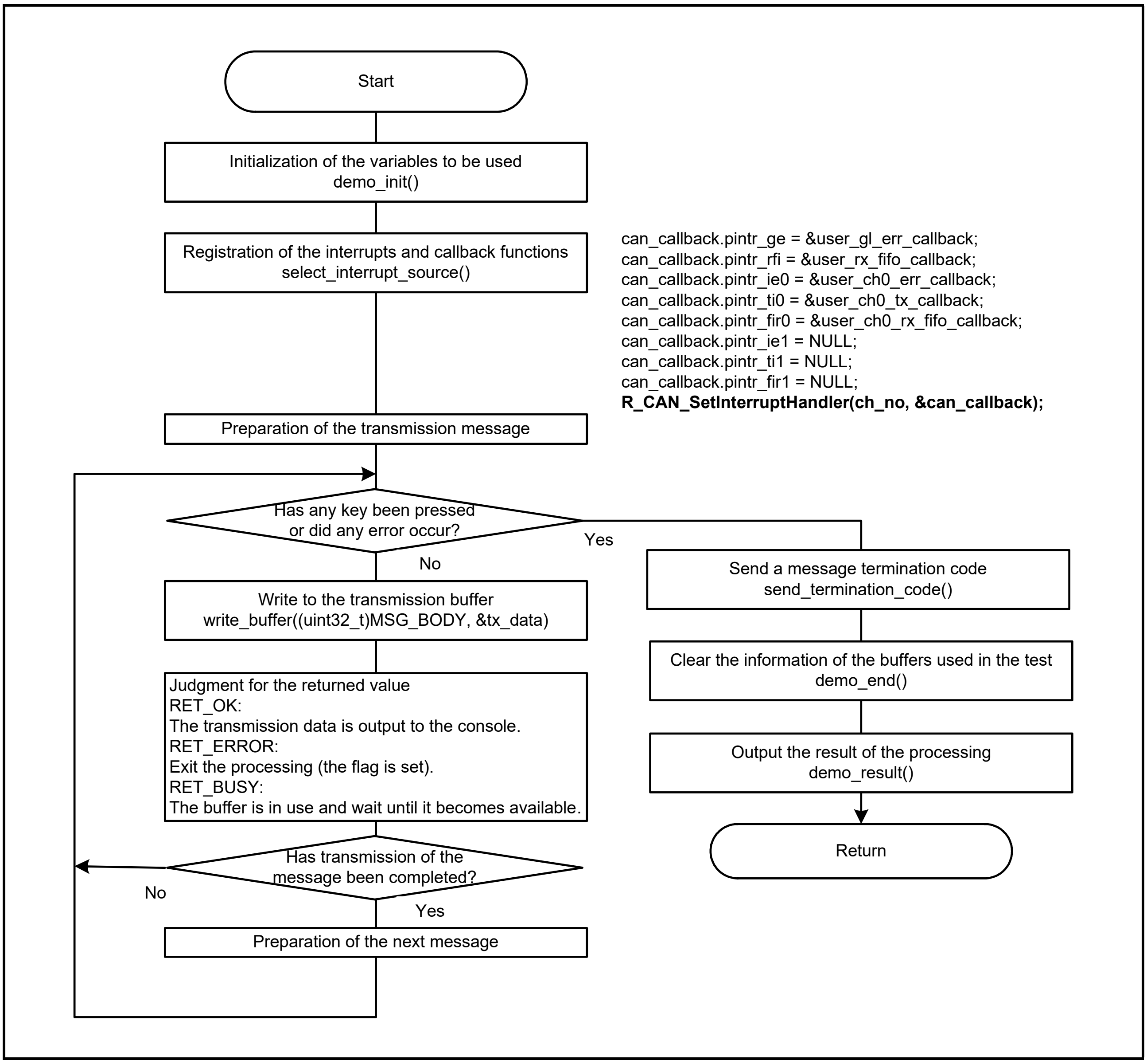


Figure 10.16 Message Transmission by Using the Transmission Buffer

(2) Message transmission by using the transmission-and-reception FIFO buffer in transmission mode
Function name: void tx_demo_fifo(void)
A flowchart for this test is shown below.

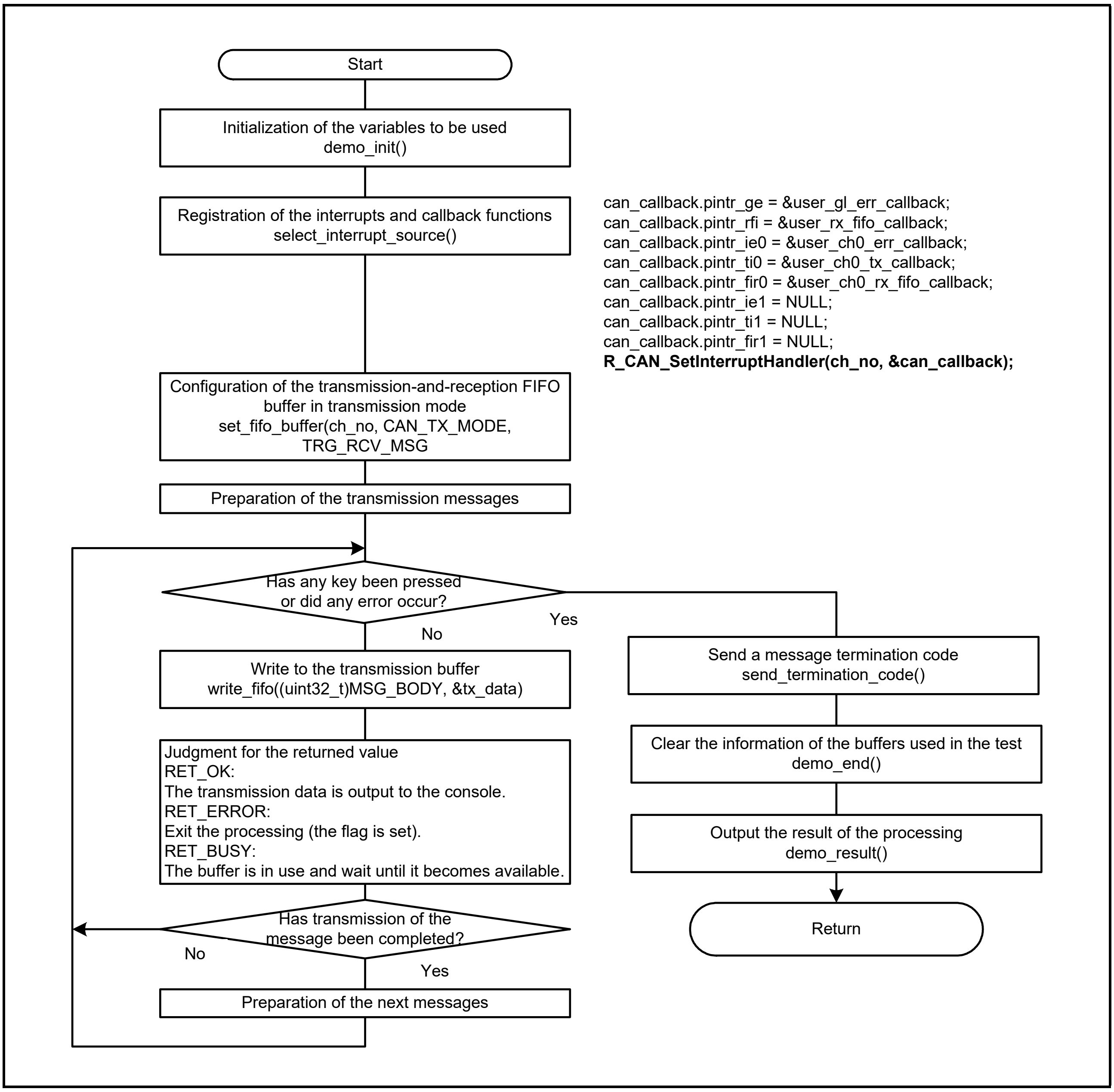


Figure 10.17 Message Transmission by Using the Transmission-and-Reception FIFO Buffer in Transmission Mode

(3) Writing transmission messages to the registers related to the transmission buffer
Function name: uint32_t write_buffer(uint32_t msg_type, tx_data_t * obj)
A flowchart for this test is shown below.

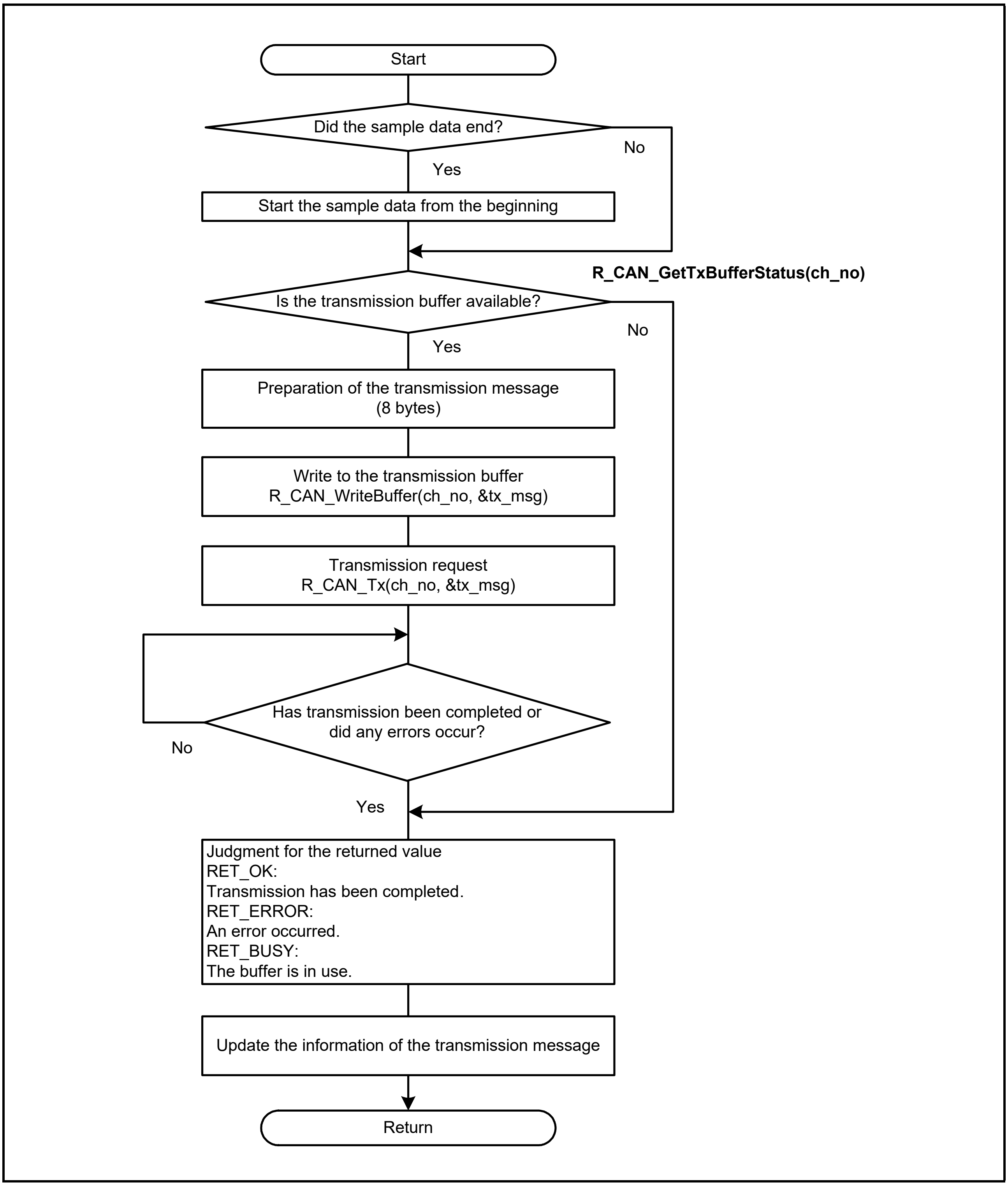


Figure 10.18 Writing Transmission Messages to the Registers Related to the Transmission Buffer

(4) Writing transmission messages to the registers related to the transmission-and-reception FIFO buffer.
Function name: uint32_t write_fifo(uint32_t msg_type, tx_data_t * obj)
A flowchart for this test is shown below.

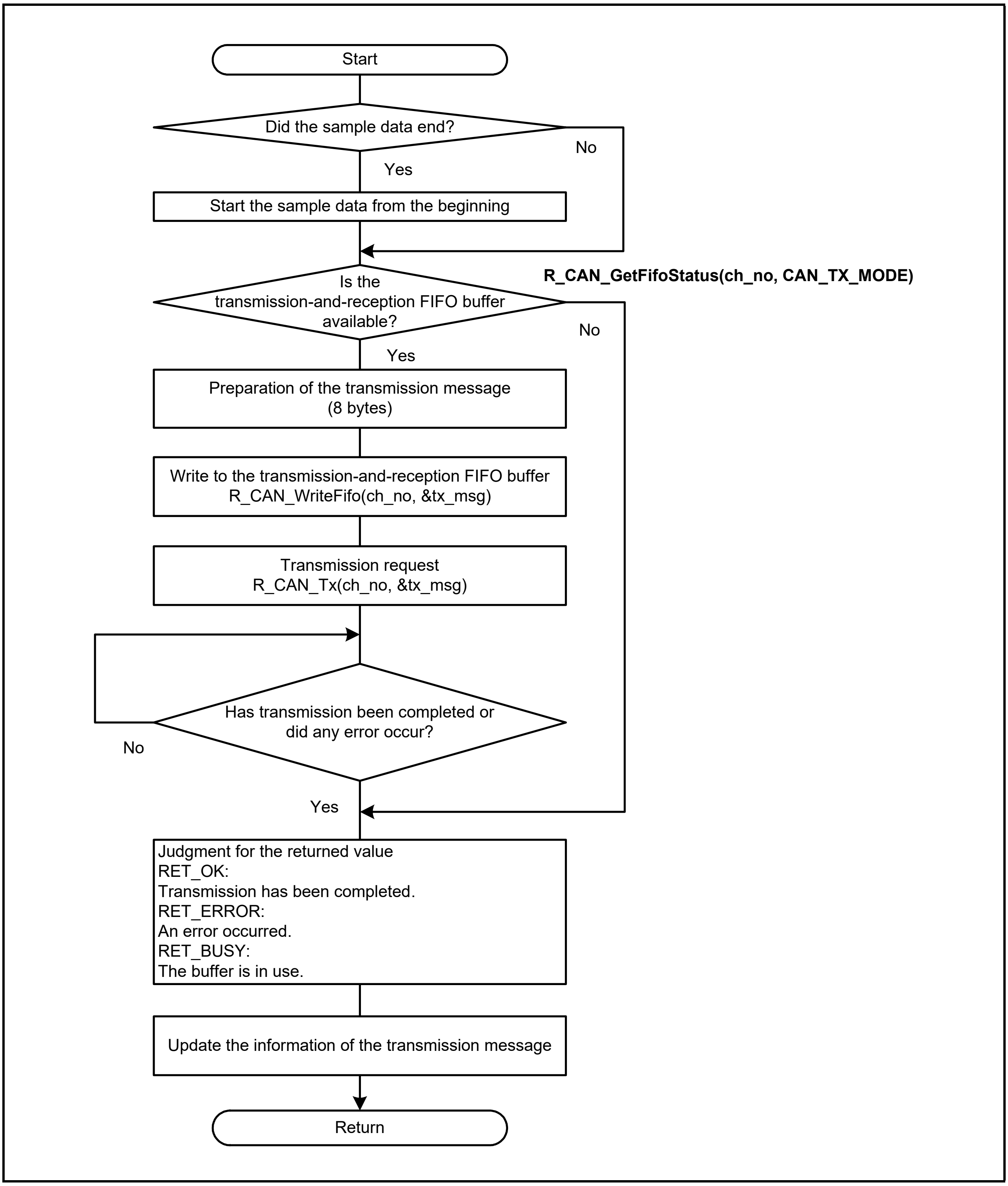


Figure 10.19 Writing Transmission Messages to the Registers Related to the Transmission-and-Reception FIFO Buffer

10.8.3 Reception Test

Two types of reception tests, message reception by using the reception buffer and by using the reception FIFO buffer are available from the menus.

See Section 10.1, Operational Outline for the menus.

The following functions are used for performing each test.

(1) void rx_demo_buffer(void)
Message reception by using the reception buffer

(2) void rx_demo_rx_fifo(void)
Message reception by using the reception FIFO buffer

(3) void rx_demo_fifo(void)
Message reception by using the transmission-and-reception FIFO buffer in reception mode

(4) uint32_t read_buffer(rx_data_t * obj)
Reading received messages from the reception buffer

(5) void read_rx_fifo(rx_data_t * obj)
Reading received messages from the reception FIFO buffer

(6) void read_fifo(uint32_t ch, rx_data_t * obj)
Reading received messages from the transmission-and-reception FIFO buffer

Figure 10.20 to Figure 10.25 show the flowcharts for processing by the respective functions.

(1) Message reception by using the reception buffer
Function name: void rx_demo_buffer(void)
A flowchart for this test is shown below.

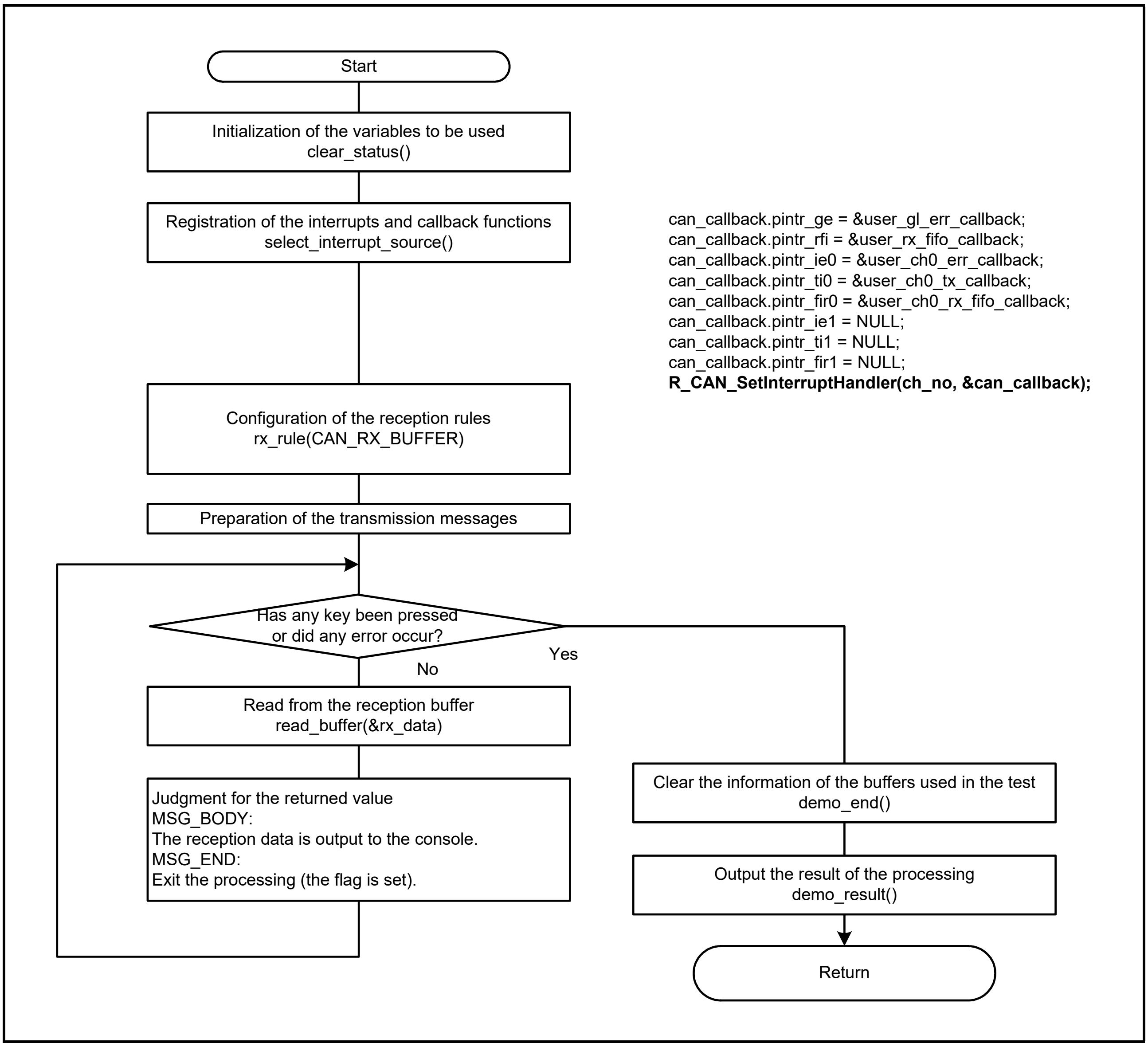


Figure 10.20 Message Reception by Using the Reception Buffer

(2) Message reception by using the reception FIFO buffer
Function name: void rx_demo_rx_fifo(void)
A flowchart for this test is shown below.

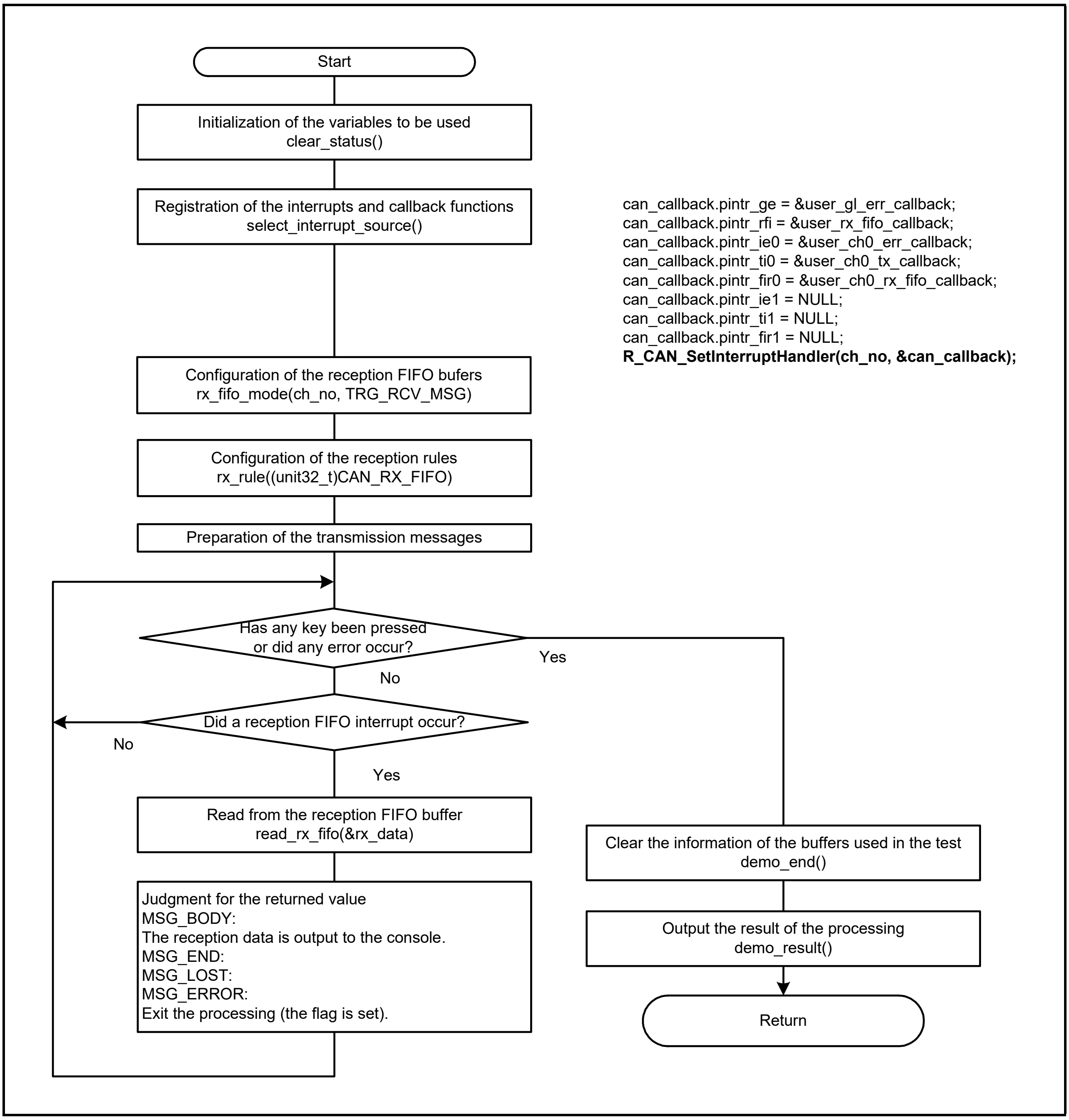


Figure 10.21 Message Reception by Using the Reception FIFO Buffer

(3) Message reception by using the transmission-and-reception FIFO buffer in reception mode
Function name: void rx_demo_fifo(void)
A flowchart for this test is shown below.

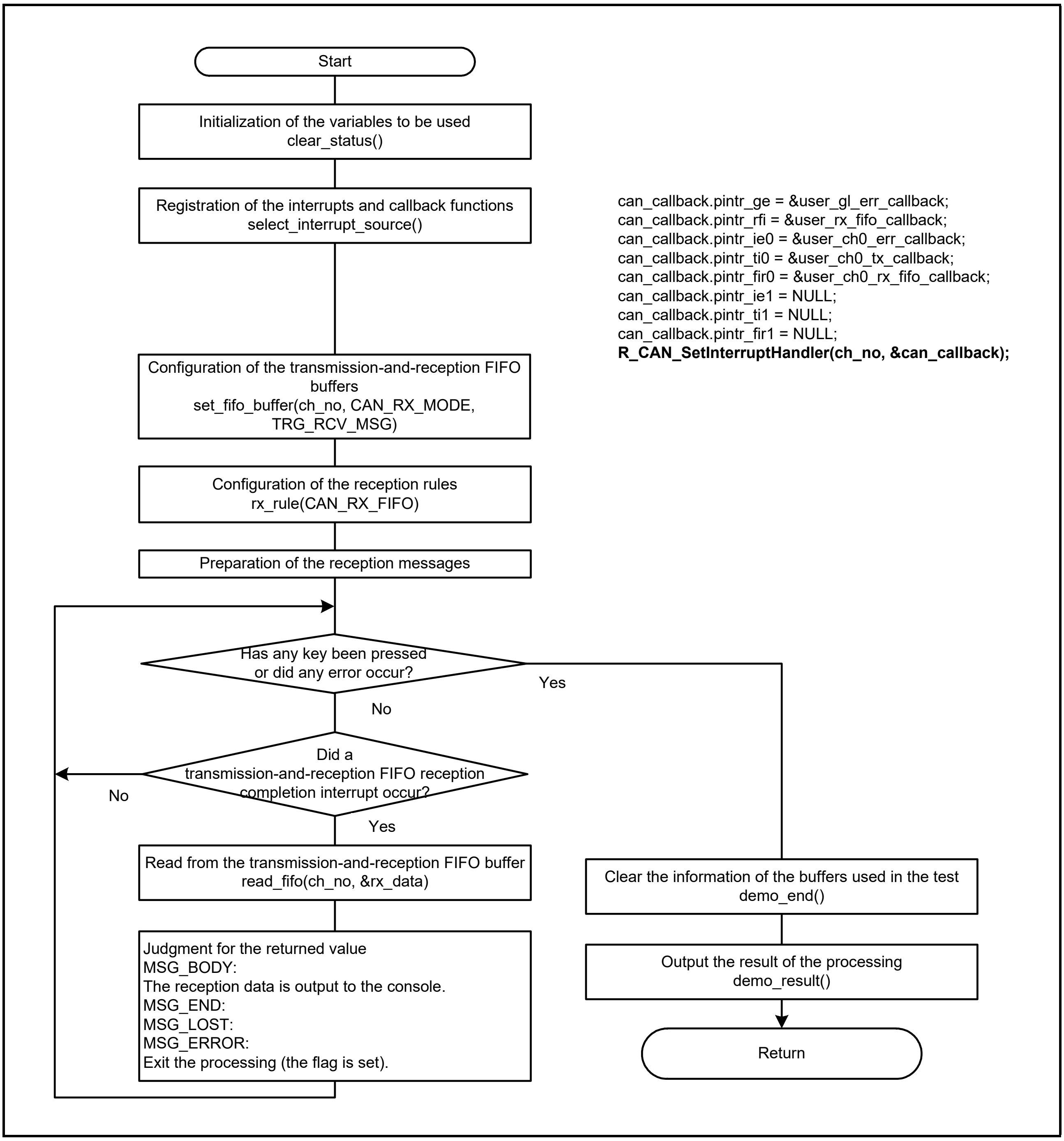


Figure 10.22 Message Reception by Using the Transmission-and-Reception FIFO Buffer in Reception Mode

(4) Reading received messages from the reception buffer
Function name: uint32_t read_buffer(rx_data_t * obj)
A flowchart for this test is shown below.

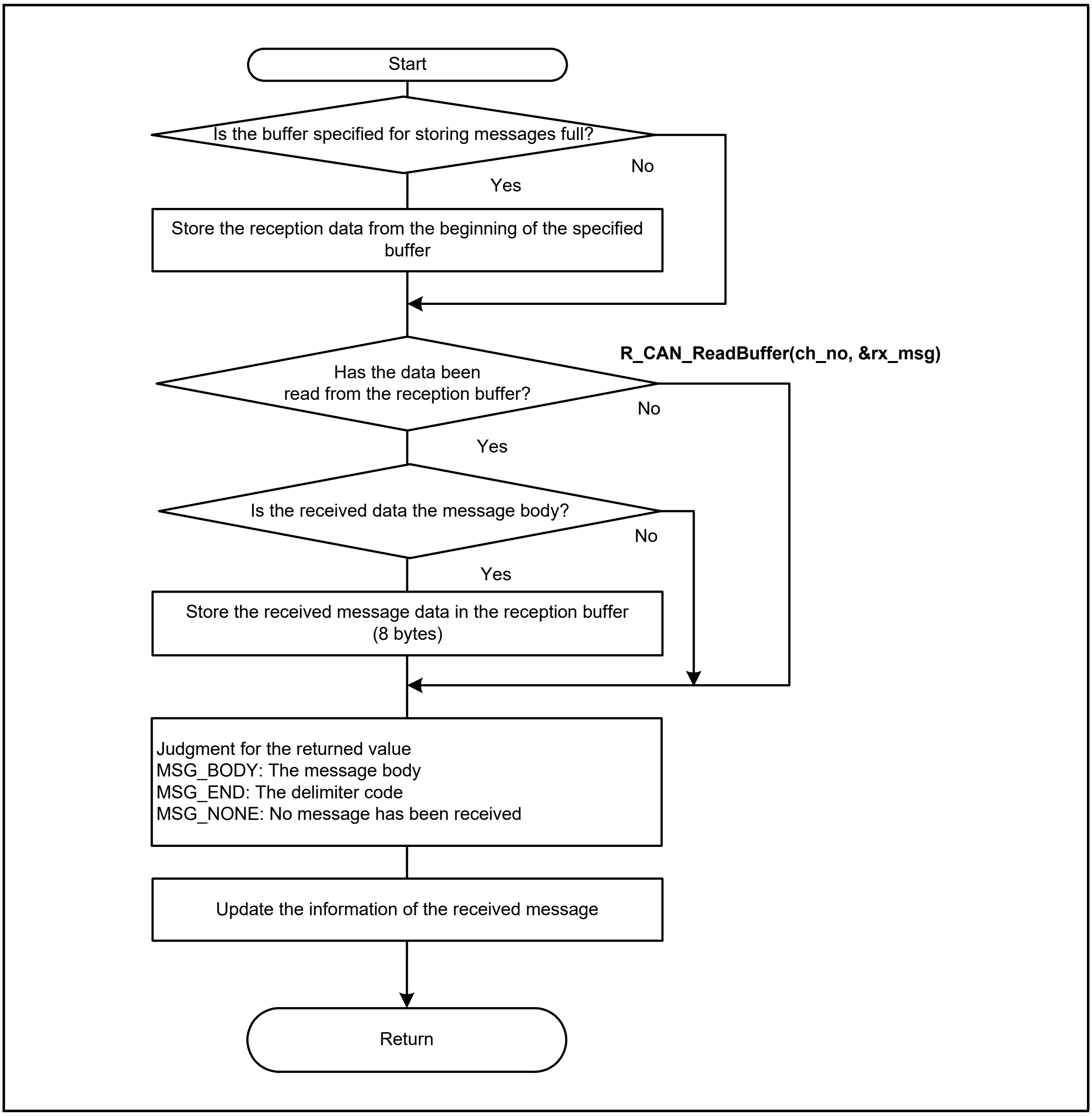


Figure 10.23 Reading Received Messages from the Reception Buffer

(5) Reading Received Messages from the Reception FIFO Buffer
Function name: void read_rx_fifo(rx_data_t * obj)
A flowchart for this test is shown below.

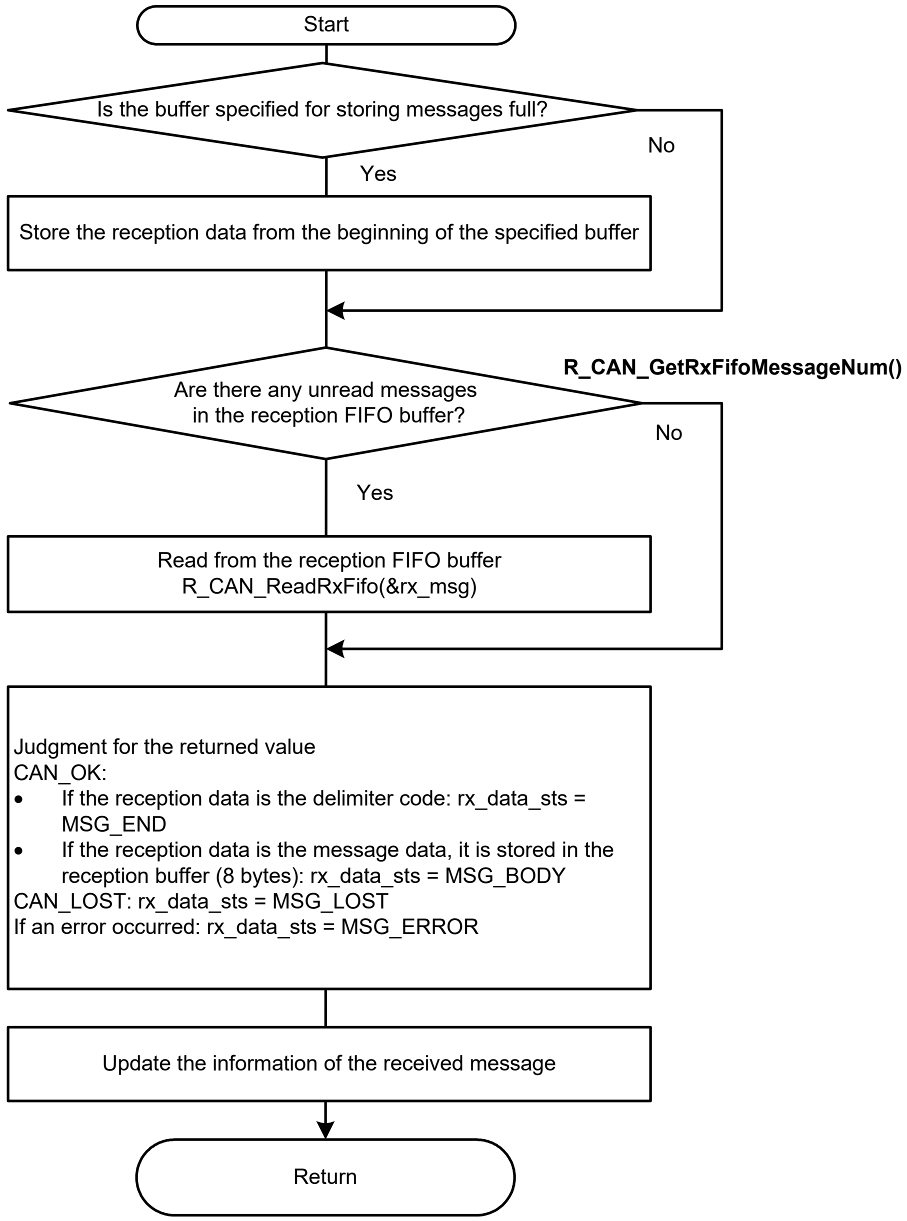


Figure 10.24 Reading Received Messages from the Reception FIFO Buffer

(6) Reading received messages from the transmission-and-reception FIFO buffer
Function name: void read_fifo(uint32_t ch, rx_data_t * obj)
A flowchart for this test is shown below.

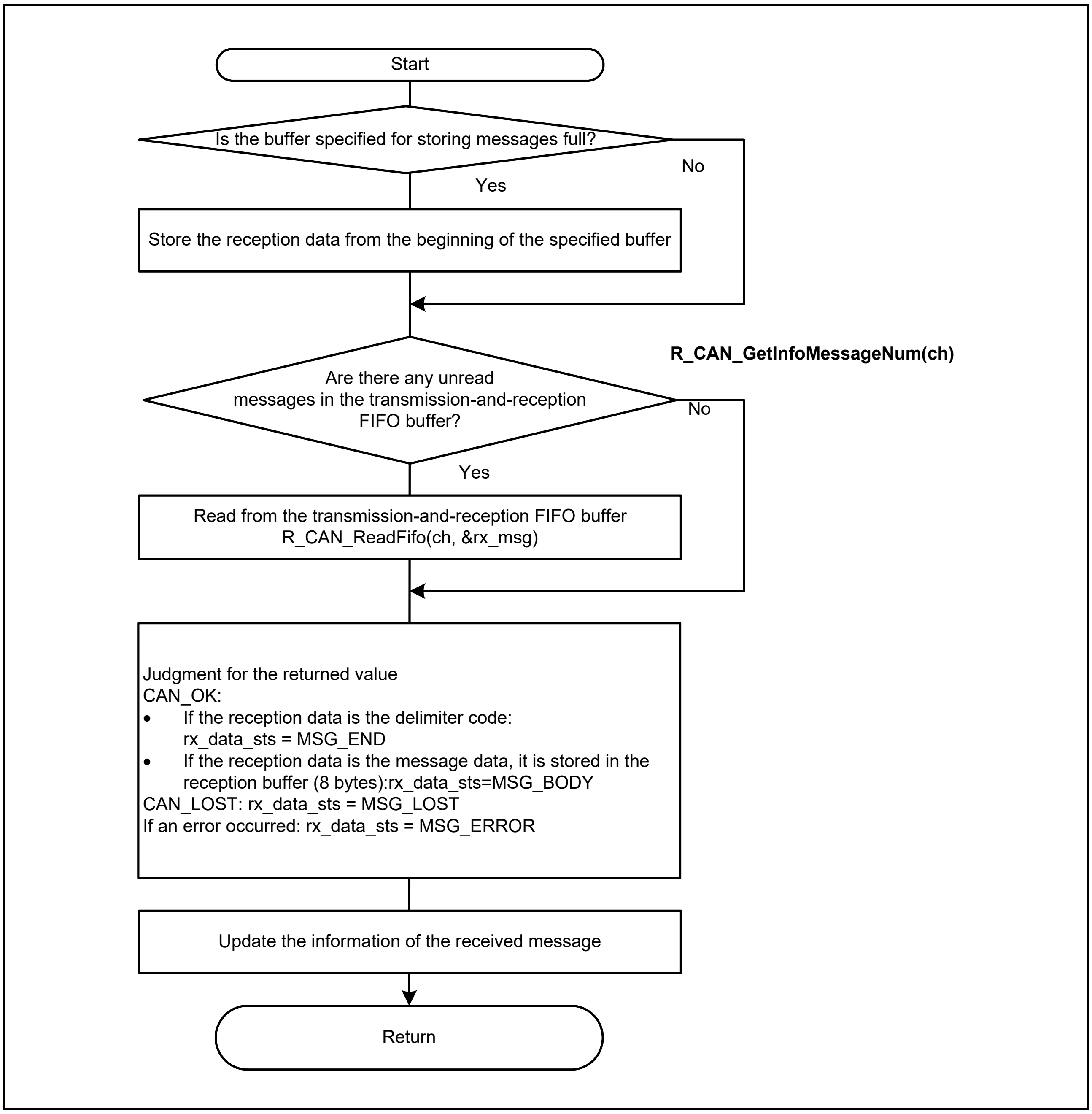


Figure 10.25 Reading Received Messages from the Transmission-and-reception FIFO Buffer

10.8.4 Test for Transmission While Receiving Data at the Same Time

This is to test transmission of messages while receiving a different message at the same time.

In this sample program, the messages are received by using the transmission-and-reception FIFO buffer in reception mode and transmitted by using the transmission-and-reception FIFO buffer in transmission mode.

See Section 10.1, Operational Outline for the menus.

The following function is used for performing this test.

(1) void trx_demo_fifo(void)
Message transmission while receiving a different message at the same time.

Figure 10.26 shows a flowchart for this processing.

(1) Message transmission while receiving a different message at the same time
Function name: void trx_demo_fifo(void)
A flowchart for this test is shown below.

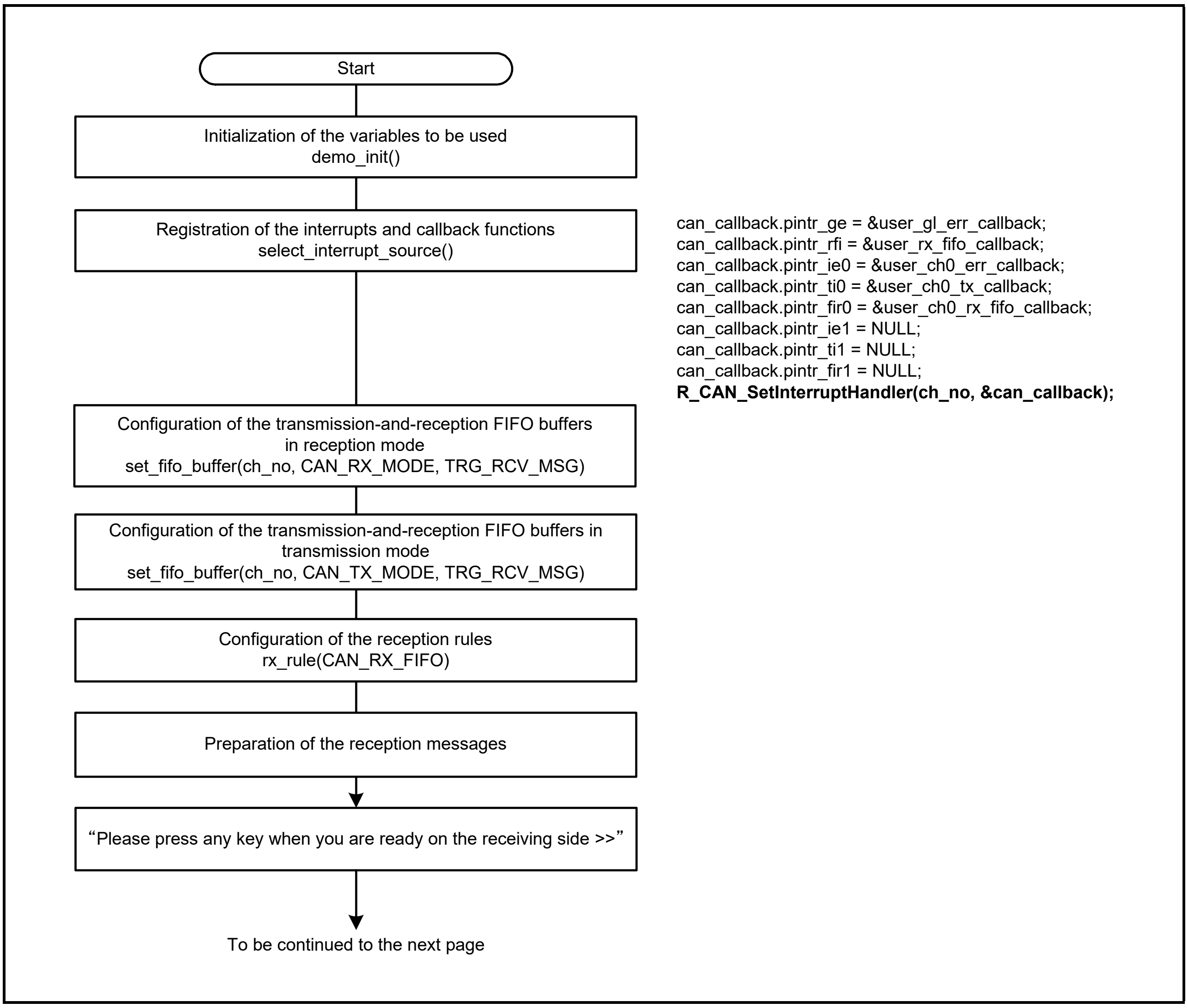


Figure 10.26 Message Transmission While Receiving a Different Message at the Same Time (1/2)

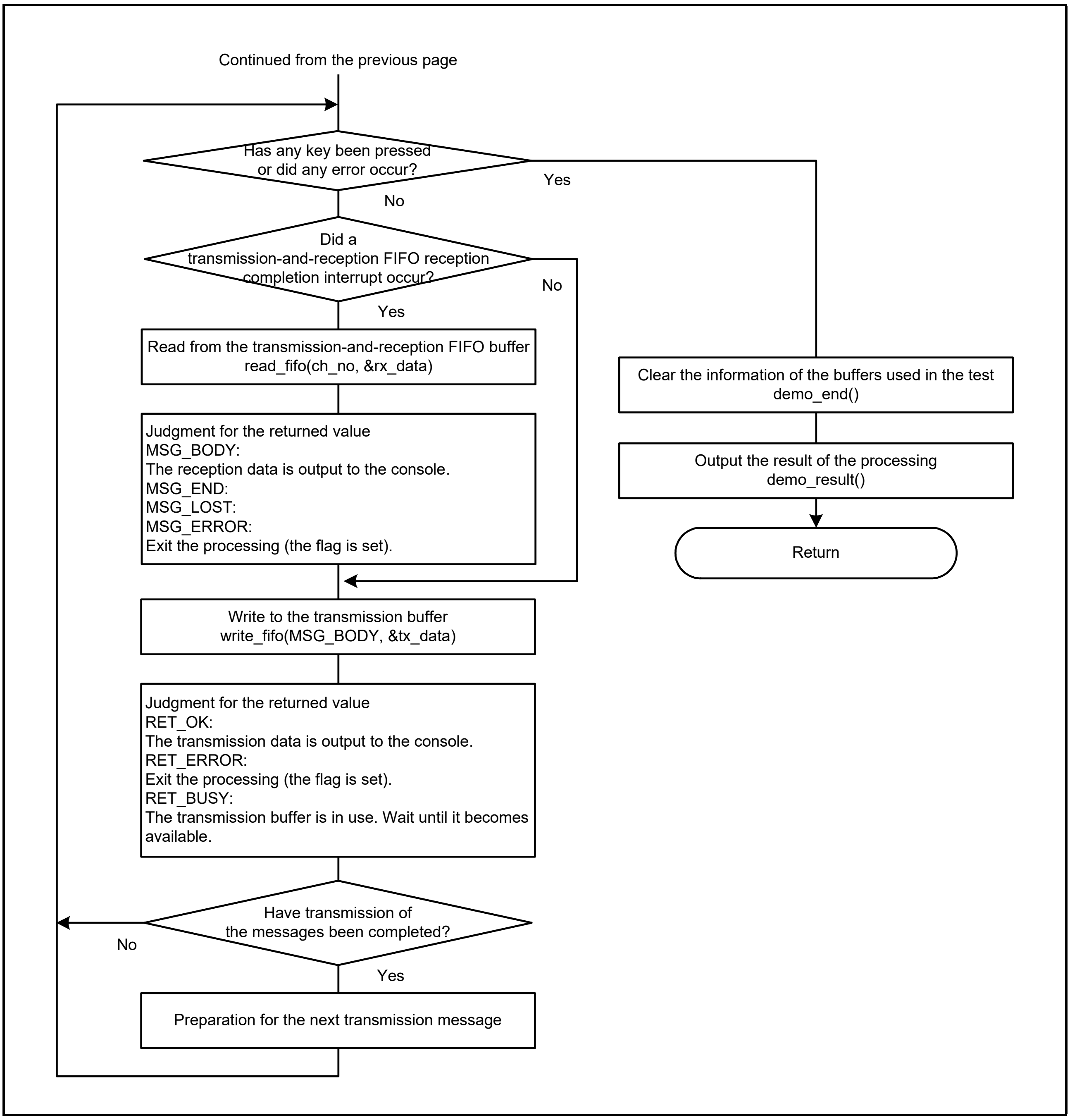


Figure 10.26 Message Transmission While Receiving a Different Message at the Same Time (2/2)

10.8.5 Self-Tests

Testing CAN communications by using a single evaluation board which is connected to the development environment is possible. In this sample program, self-tests for checking transmission and reception are available. Select the test mode from self-test mode 0 for external loopback or self-test mode 1 for internal loopback from the menu.

See Section 10.1, Operational Outline for the menus.

(1) Self-test mode 0 (external loopback)
This is a loopback within a channel including the CAN transceiver. Figure 10.27 shows the connection when self-test mode 0 is selected.

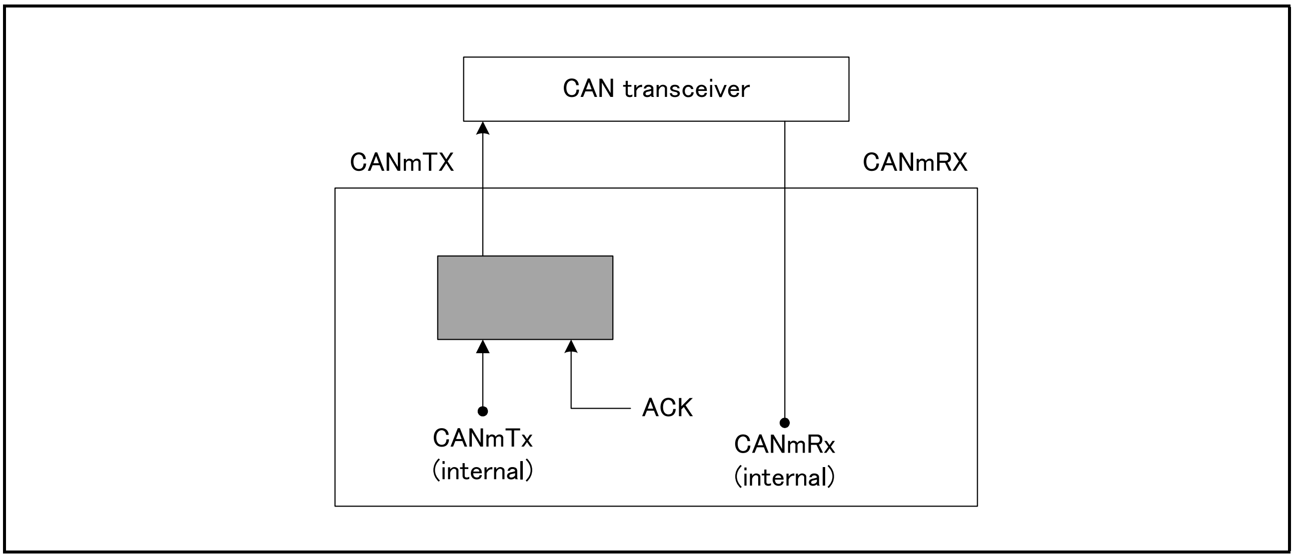


Figure 10.27 Connection for Self-Test Mode 0

(2) Self-test mode 1 (internal loopback)
In this mode, the transmitted messages are handled as the reception messages and stored in the specified buffer. Figure 10.28 shows the connection when self-test mode 1 is selected.

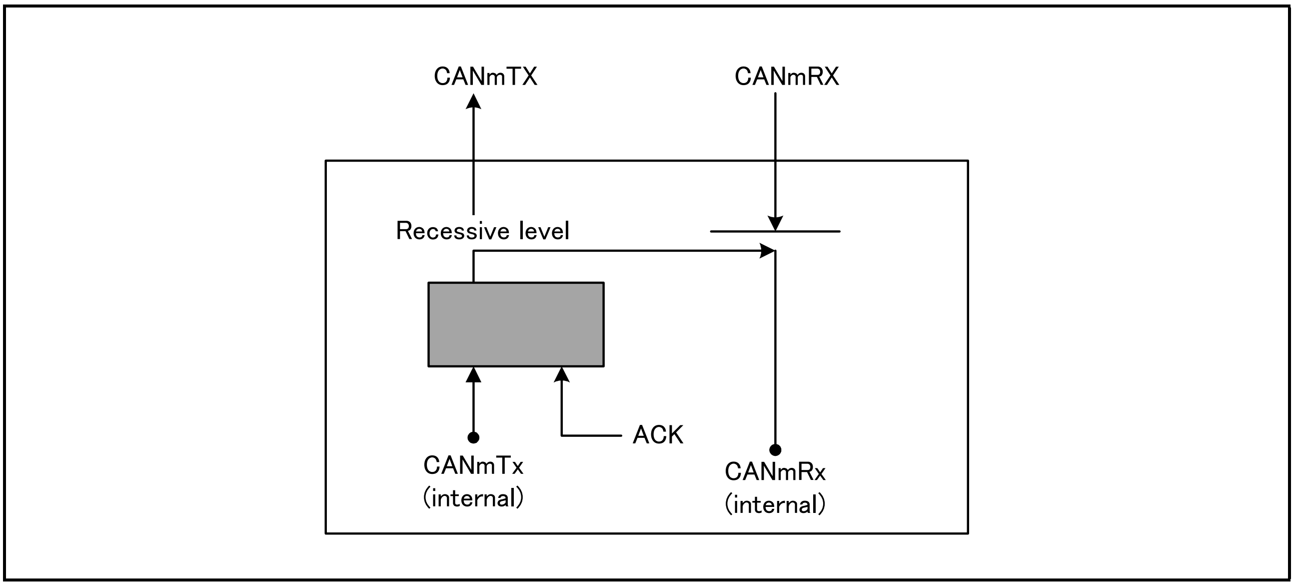


Figure 10.28 Connection for Self-Test Mode 1

Four types of self-tests are available as follows;

- Transmitting a message from the transmission buffer and receiving it at the reception buffer
- Transmitting a message from the transmission buffer and receiving it at the reception FIFO buffer
- Transmitting a message from the transmission buffer and receiving it at the transmission-and-reception FIFO buffer in reception mode
- Transmitting a message from the transmission-and-reception FIFO buffer in transmission mode and receiving it at the transmission-and-reception FIFO buffer in reception mode

The type of transmission depends on the buffer used for transmission as follows.

The transmission buffer: transmission of one message is repeated.

The transmission-and-reception FIFO buffer in transmission mode: transmission of one message is repeated.

The type of reception depends on the buffer used for reception as follows.

The reception FIFO buffer:

An interrupt occurs when the number of messages stored in the reception FIFO buffer matches the specified FIFO buffer depth (four messages).

The transmission-and-reception FIFO buffer in reception mode:

An interrupt occurs when the number of messages stored in the transmission-and-reception FIFO buffer matches the transmission-and-reception FIFO buffer depth (four messages).

The following functions are used for performing each test.

(1) void selftest_buf_to_buf(void)

Sending a message from the transmission buffer and receiving it at the reception buffer

(2) void selftest_buf_to_rx_fifo(void)

Sending a message from the transmission buffer and receiving it at the reception FIFO buffer

(3) void selftest_buf_to_fifo(void)

Sending a message from the transmission buffer and receiving it at the transmission-and-reception FIFO buffer

(4) void selftest_fifo_to_fifo(void)

Sending a message from the transmission-and-reception FIFO buffer in transmission mode and receiving it at the transmission-and-reception FIFO buffer in reception mode

Figure 10.29 to Figure 10.32 show the flowcharts for processing by the respective functions.

(1) Sending a message from the transmission buffer and receiving it at the reception buffer
Function name: void selftest_buf_to_buf(void)
A flowchart for this test is shown below.

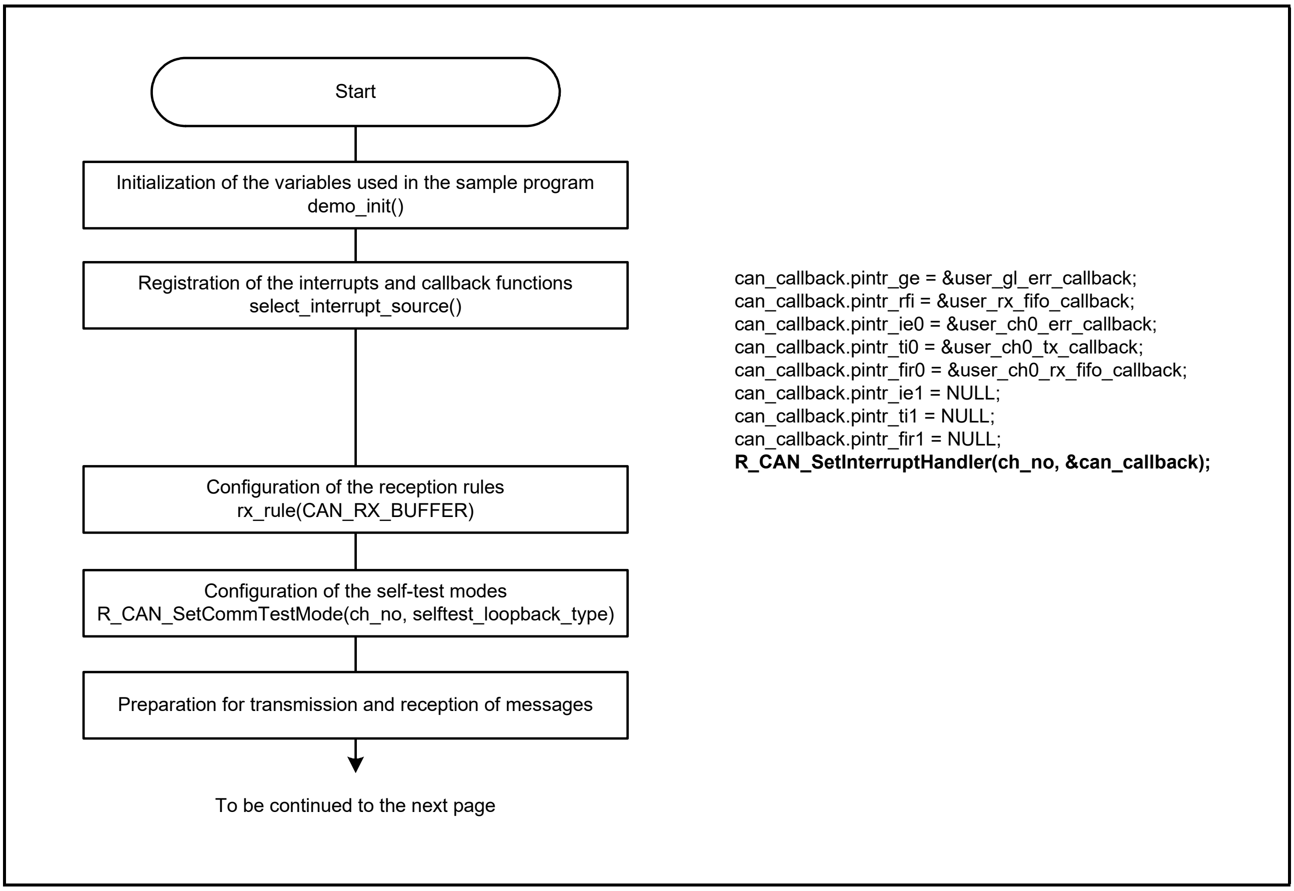


Figure 10.29 Sending a Message from the Transmission Buffer and Receiving It at the Reception Buffer (1/2)

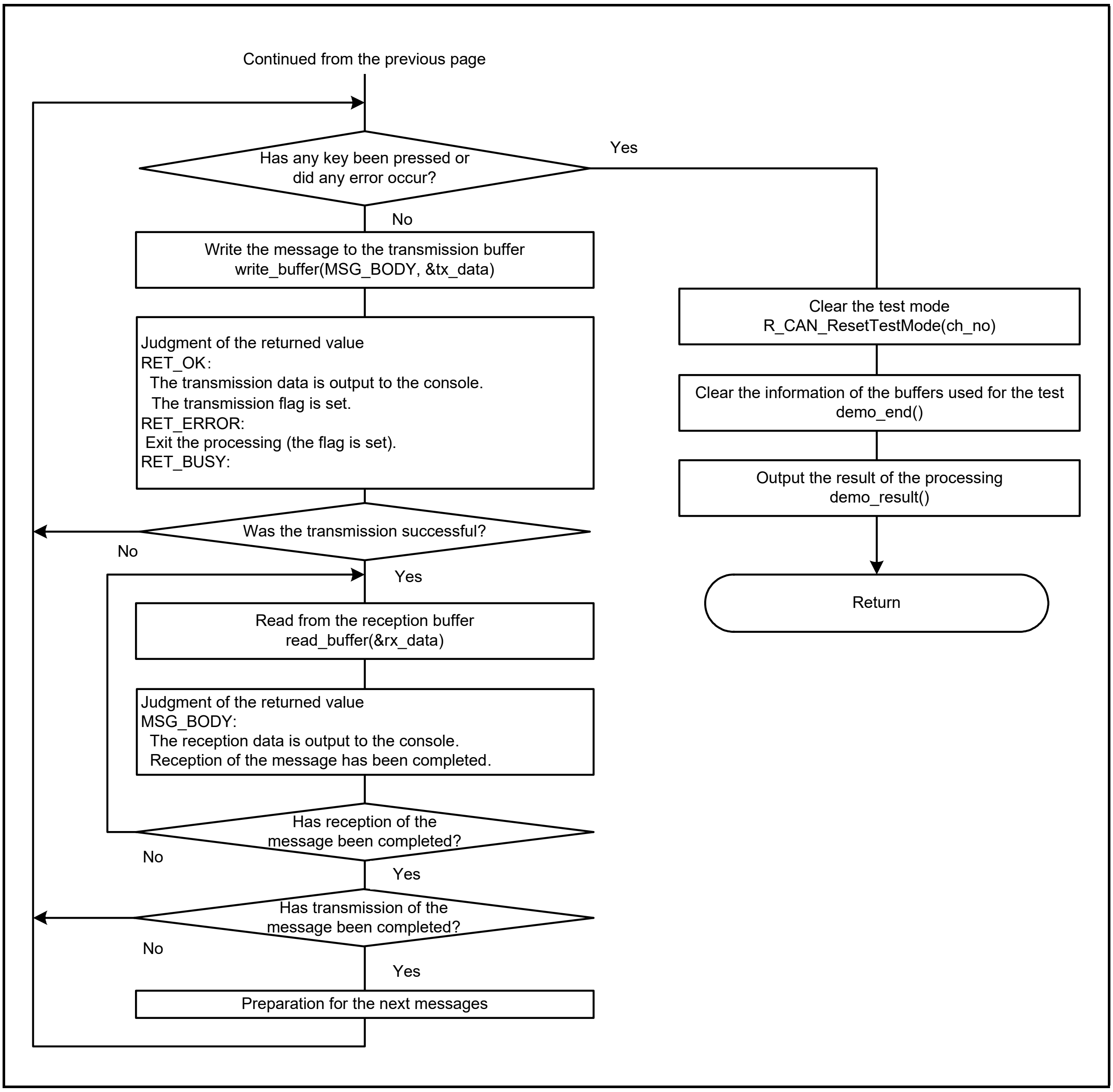


Figure 10.29 Sending a Message from the Transmission Buffer and Receiving It at the Reception Buffer (2/2)

(2) Sending a message from the transmission buffer and receiving it at the reception FIFO buffer
Function name: void selftest_buf_to_rx_fifo(void)
A flowchart for this test is shown below.

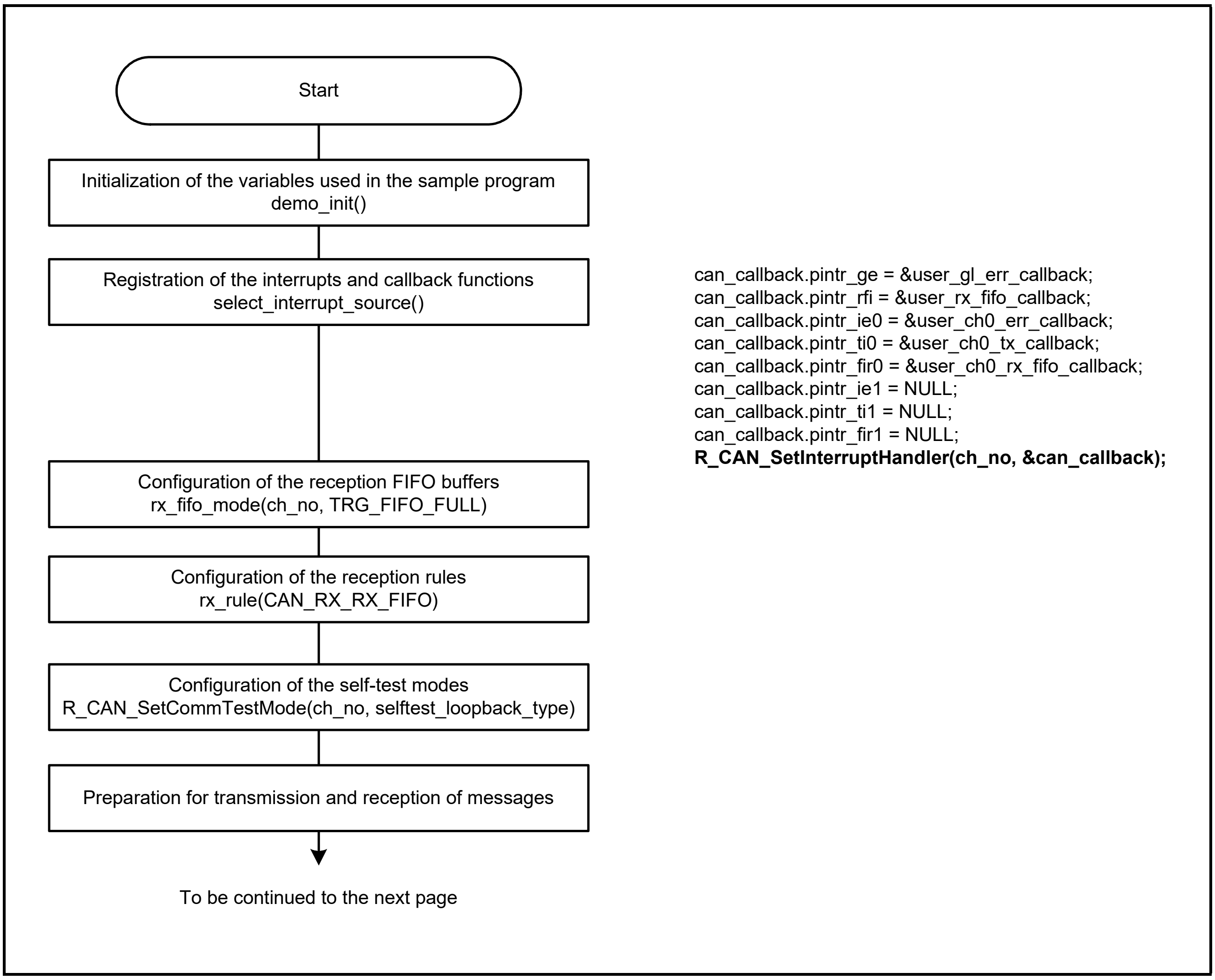


Figure 10.30 Sending a Message from the Transmission Buffer and Receiving It at the Reception FIFO buffer (1/2)

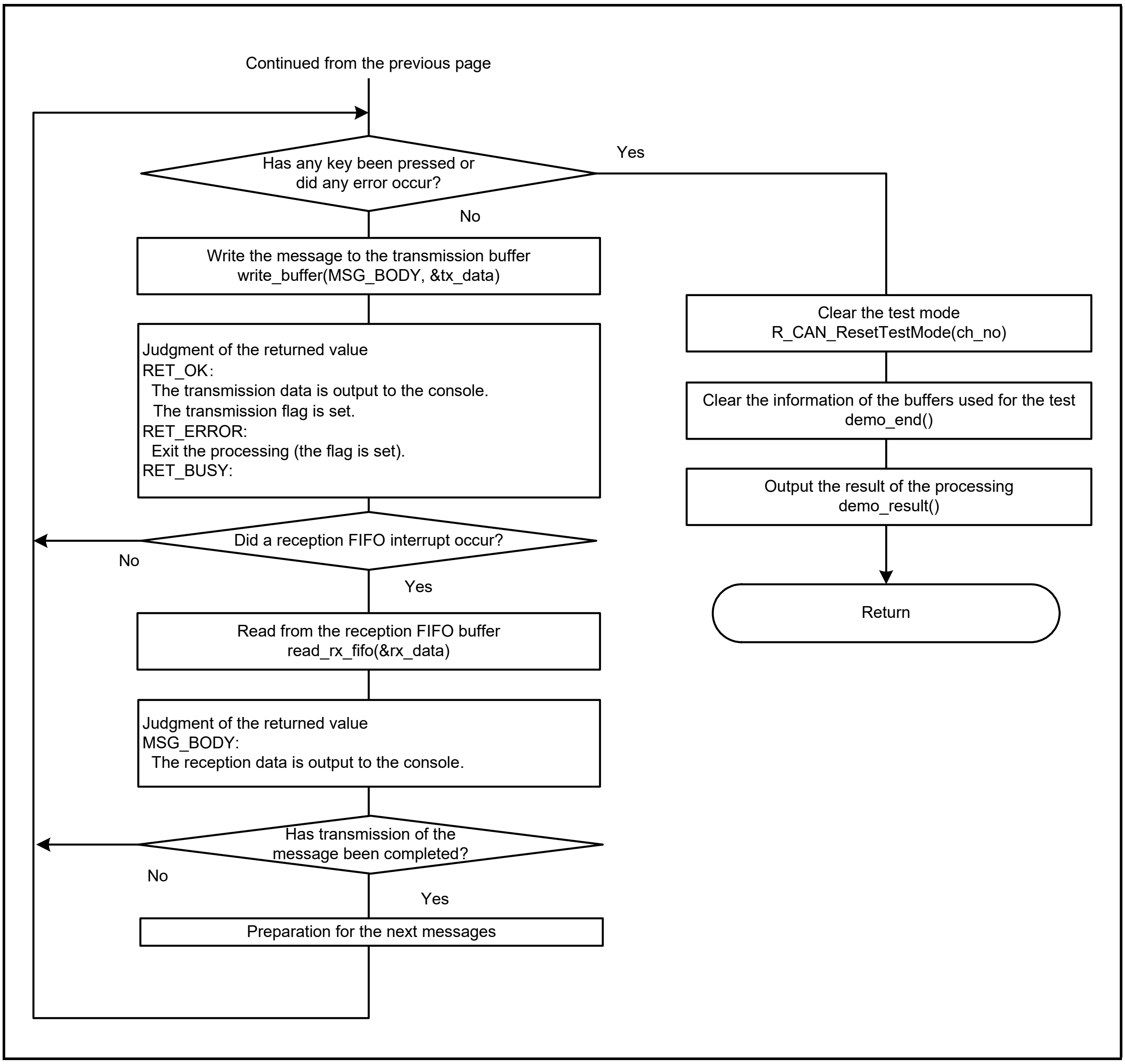


Figure 10.30 Sending a Message from the Transmission Buffer and Receiving It at the Reception FIFO Buffer (2/2)

(3) Sending a message from the transmission buffer and receiving it at the transmission-and-reception FIFO buffer in reception mode

Function name: void selftest_buf_to_fifo(void)

A flowchart for this test is shown below.

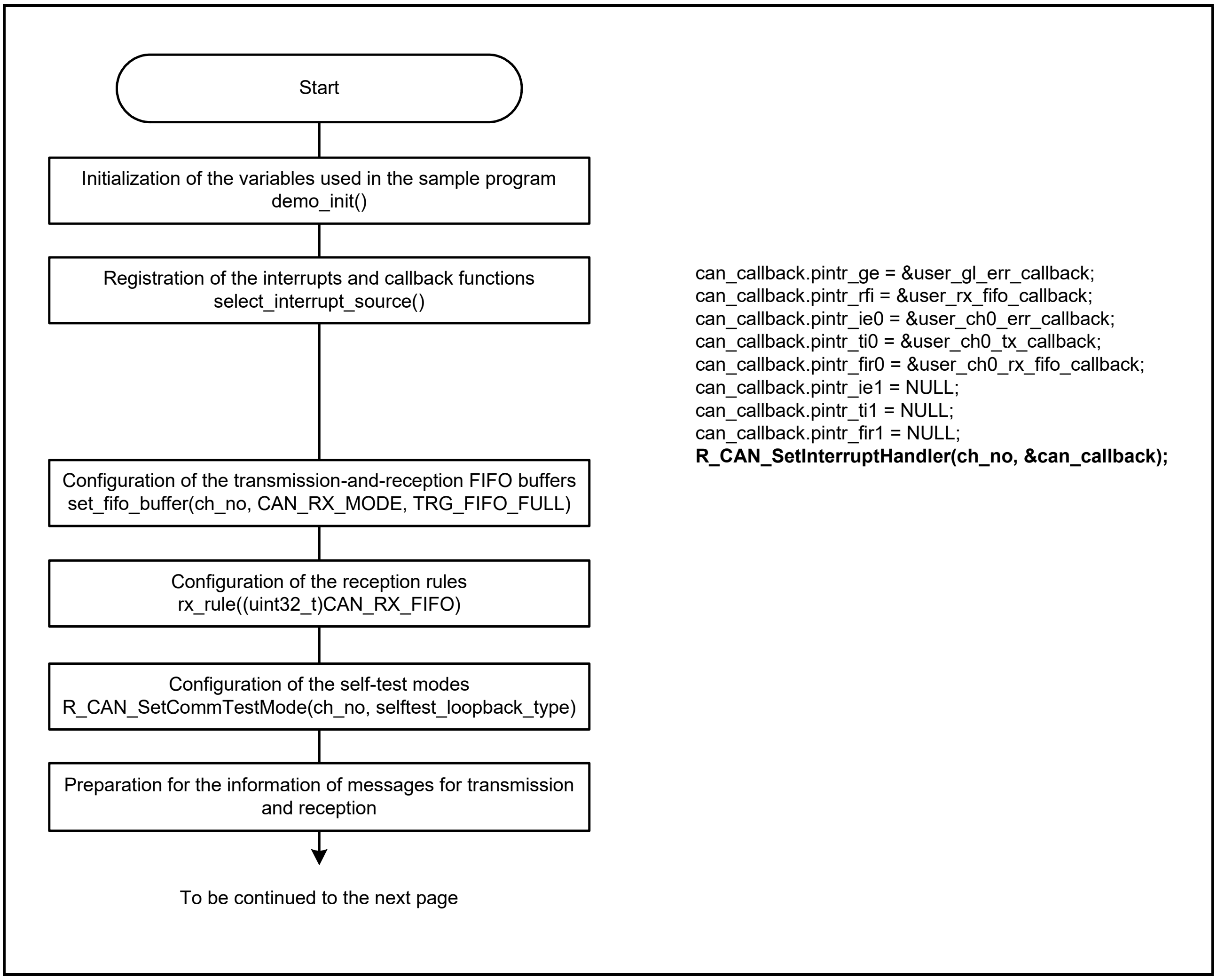


Figure 10.31 Sending a Message from the Transmission Buffer and Receiving It at the Transmission-and-Reception FIFO Buffer in Reception Mode (1/2)

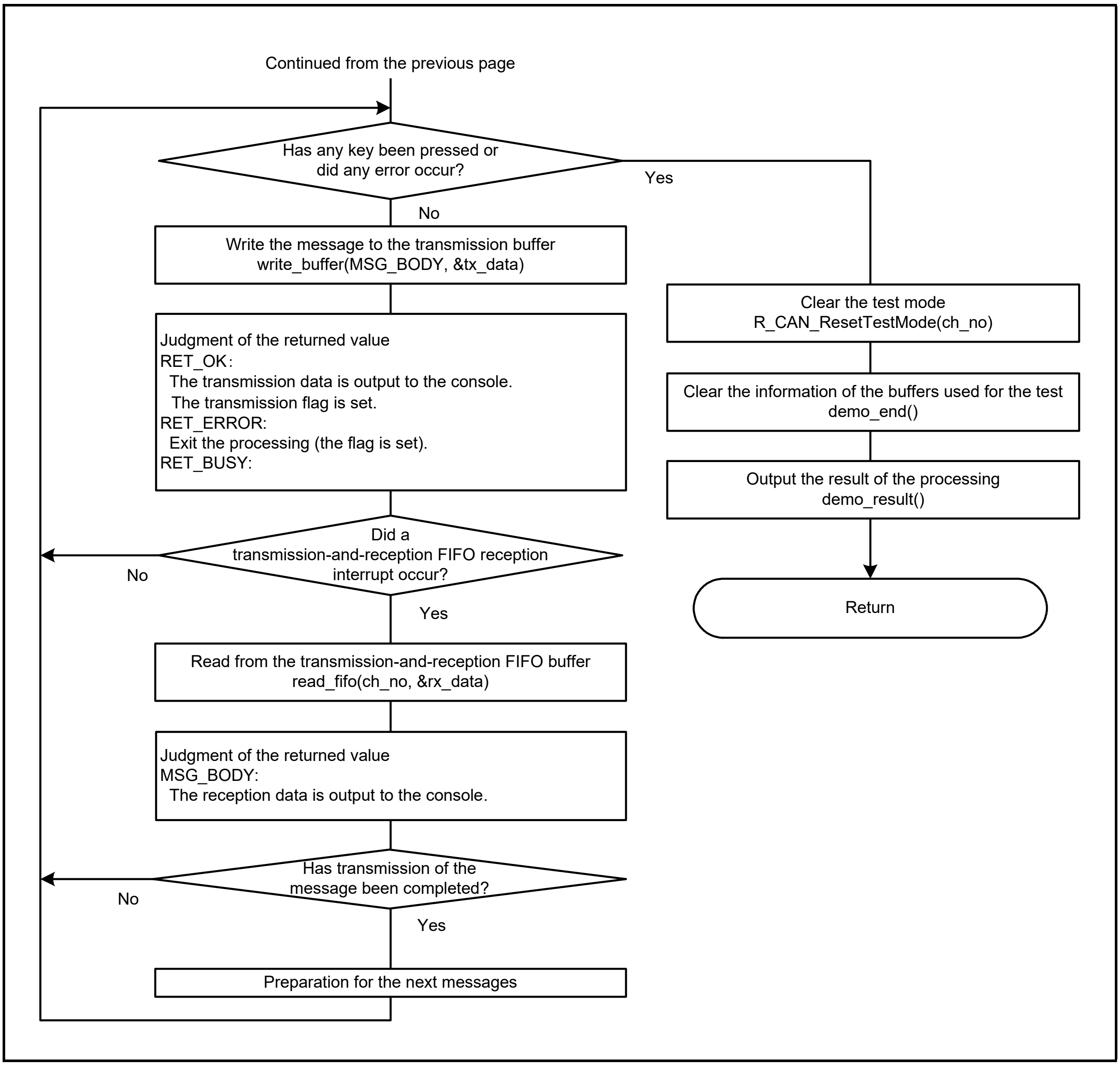


Figure 10.31 Sending a Message from the Transmission Buffer and Receiving It at the Transmission-and-Reception FIFO Buffer in Reception Mode (2/2)

(4) Sending a message from the transmission-and-reception FIFO buffer in transmission mode and receiving it at the transmission-and-reception FIFO buffer in reception mode
Function name: void selftest_fifo_to_fifo(void)
A flowchart for this test is shown below.

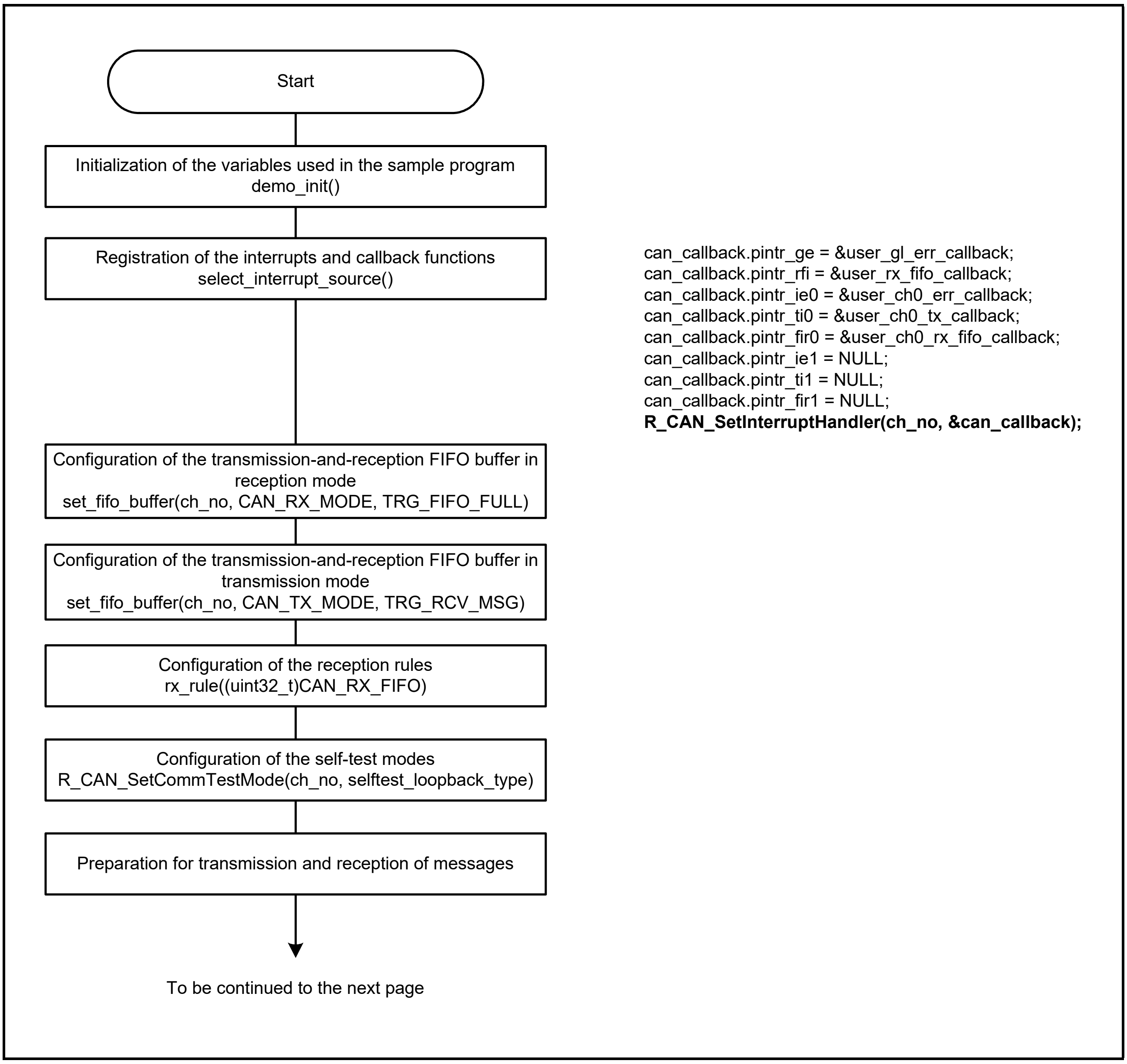


Figure 10.32 Sending a Message from the Transmission-and-Reception FIFO Buffer in Transmission Mode and Receiving It at the Transmission-and-Reception FIFO Buffer in Reception Mode (1/2)

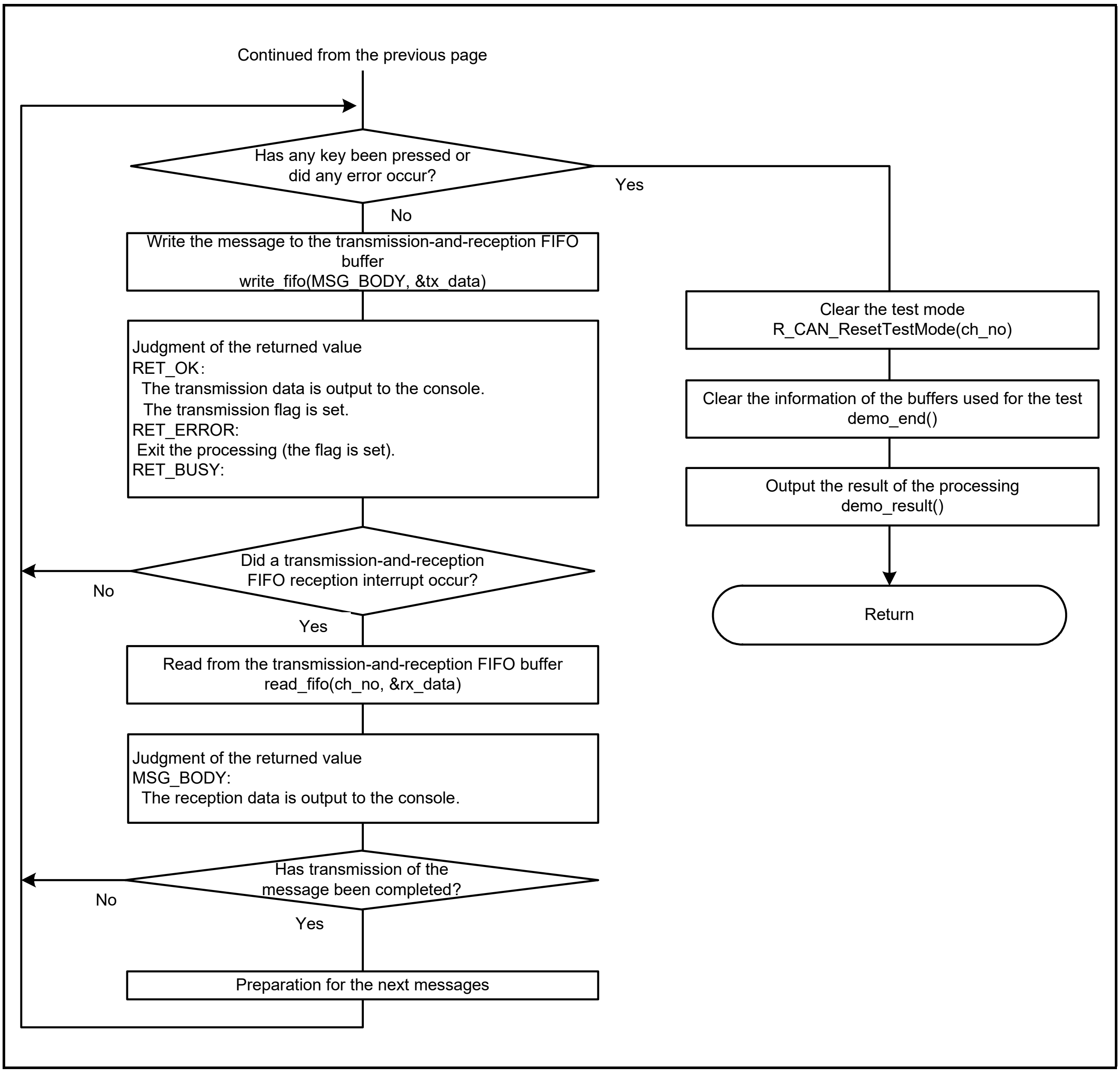


Figure 10.32 Sending a Message from the Transmission-and-Reception FIFO Buffer in Transmission Mode and Receiving It at the Transmission-and-Reception FIFO Buffer in Reception Mode (2/2)

10.8.6 Callback Processing

This sample program includes interrupt handling routines that are activated on occurrence of the various interrupt source conditions. The handling routines and the callback functions called by each are listed below.

(1) RSCAN:CANGE (CAN global error)
Interrupt handler: void user_gl_err_isr(void)
Callback function: void user_gl_err_callback(void)

(2) RSCAN:CANIE0 (CAN0 error)
Interrupt handler: void user_ch0_err_isr(void)
Callback function: void user_ch0_err_callback(void)

(3) RSCAN:CANIE1 (CAN1 error)
Interrupt handler: void user_ch1_err_isr(void)
Callback function: void user_ch1_err_callback(void)

(4) RSCAN:CANRFI (CAN reception FIFO interrupt)
Interrupt handler: void user_rx_fifo_isr(void)
Callback function: void user_rx_fifo_callback(void)

(5) RSCAN:CANTI0 (CAN0 transmission interrupt)
Interrupt handler: void user_ch0_tx_isr(void)
Callback function: void user_ch0_tx_callback(void)

(6) RSCAN:CANTI1 (CAN1 transmission interrupt)
Interrupt handler: void user_ch1_tx_isr(void)
Callback function: void user_ch1_tx_callback(void)

(7) RSCAN:CANFIR0 (CAN0 transmission-and-reception FIFO reception completion interrupt)
Interrupt handler: void user_ch0_rx_fifo_isr(void)
Callback function: void user_ch0_rx_fifo_callback(void)

(8) RSCAN:CANFIR1 (CAN1 transmission-and-reception FIFO reception completion interrupt)
Interrupt handler: void user_ch1_rx_fifo_isr(void)
Callback function: void user_ch1_rx_fifo_callback(void)

(9) SCIFA:RXIF2 (SCIFA reception FIFO data full interrupt)
Interrupt handler: void scifa_key_input_isr(void)
Callback function: void key_handler_callback(void)

How to configure the interrupt handling routines is described here:

Use the API functions below to configure the interrupt handling routines.

void R_ICU_Regist(uint32_t vec_num, uint32_t type, uint32_t priority, uint32_t isr_addr);

uint32_t vec_num: vector number

uint32_t type: interrupt detection type

uint32_t priority: priority level of the interrupt

uint32_t isr_addr: address of the function for the interrupt handling routine

Use the API functions below to enable or disable the interrupt.

void R_ICU_Disable(uint32_t vec_num);

void R_ICU_Enable(uint32_t vec_num);

uint32_t vec_num: vector number

An example of configuring the callback function is described here.

can_handle_t gb_can_handles[CAN_NUM];

```
typedef struct {
    void (*pintr_ge)(void);         /* Pointer to user callback function. */
    void (*pintr_ie0)(void);        /* Pointer to user callback function. */
    void (*pintr_ie1)(void);        /* Pointer to user callback function. */
    void (*pintr_rfi)(void);        /* Pointer to user callback function. */
    void (*pintr_fir0)(void);       /* Pointer to user callback function. */
    void (*pintr_ti0)(void);        /* Pointer to user callback function. */
    void (*pintr_fir1)(void);       /* Pointer to user callback function. */
    void (*pintr_ti1)(void);        /* Pointer to user callback function. */
} can_callback_t;

typedef struct {
    bool            ch_opened;
    can_callback_t  can_callback;
} can_handle_t;
```

Figure 10.33 to Figure 10.38 show the flowcharts for handling the callback functions on occurrence of the respective errors.

(1) (RSCAN: CANGE) – A callback function which is called on occurrence of a can global error
Interrupt handler: void user_gl_err_isr(void)
Callback function: void user_gl_err_callback(void)
This callback function is called by the given interrupt handler on the occurrence of an error in the global portion of the CAN module.
A flowchart for the interrupt handling routine and the corresponding callback function is shown below.

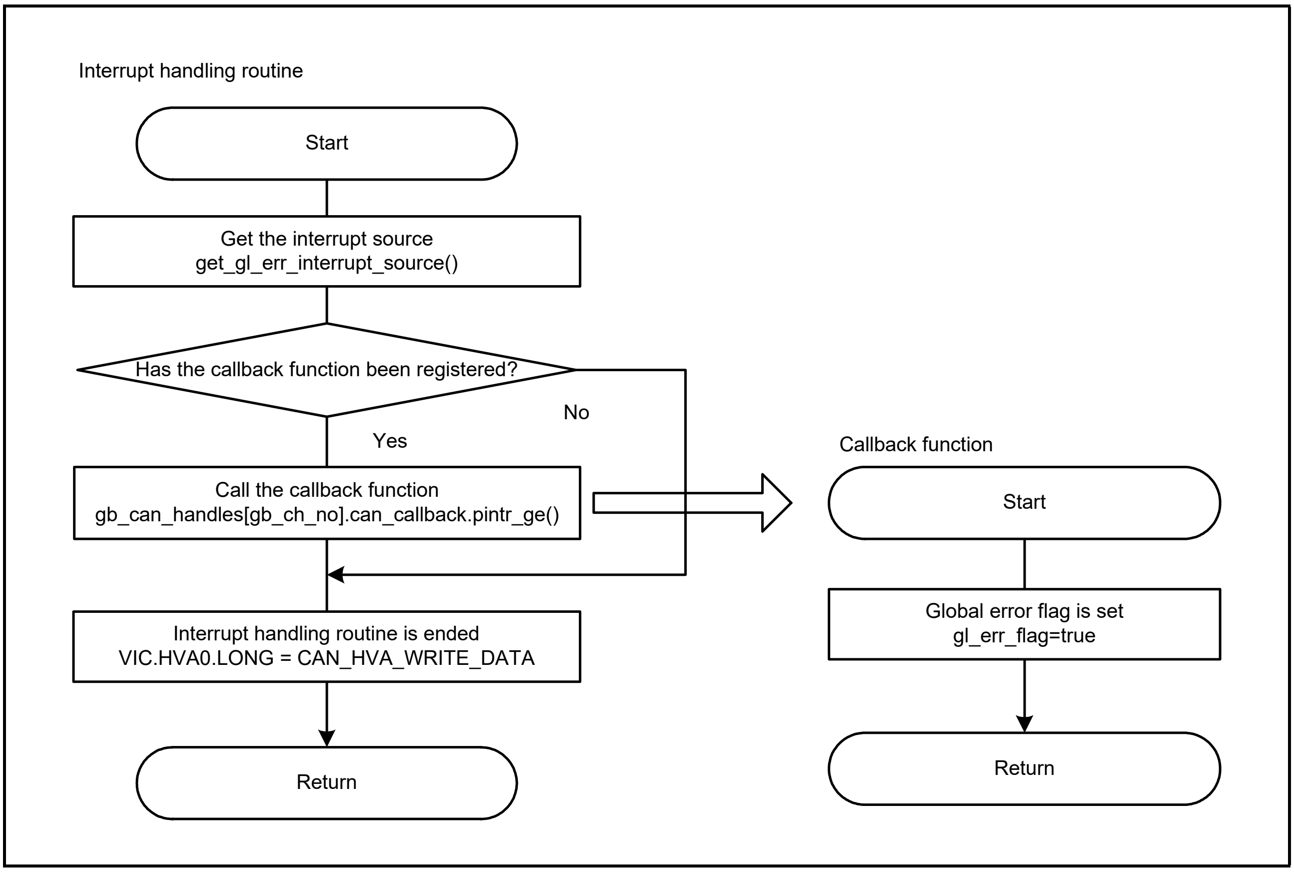


Figure 10.33 Handling of the Callback Function for a CAN Global Error

(2) (RSCAN:CANIEm) – A callback function which is called on occurrence of a CANm error
Interrupt handler: void user_ch0_err_isr(void), void user_ch1_err_isr(void)
Callback function: void user_ch0_err_callback(void), void user_ch1_err_callback(void)
This callback function is called by the given interrupt handler on the occurrence of an error in the channel (CAN0 or CAN1) of the CAN module.
A flowchart for the interrupt handling routine and the corresponding callback function is shown below.

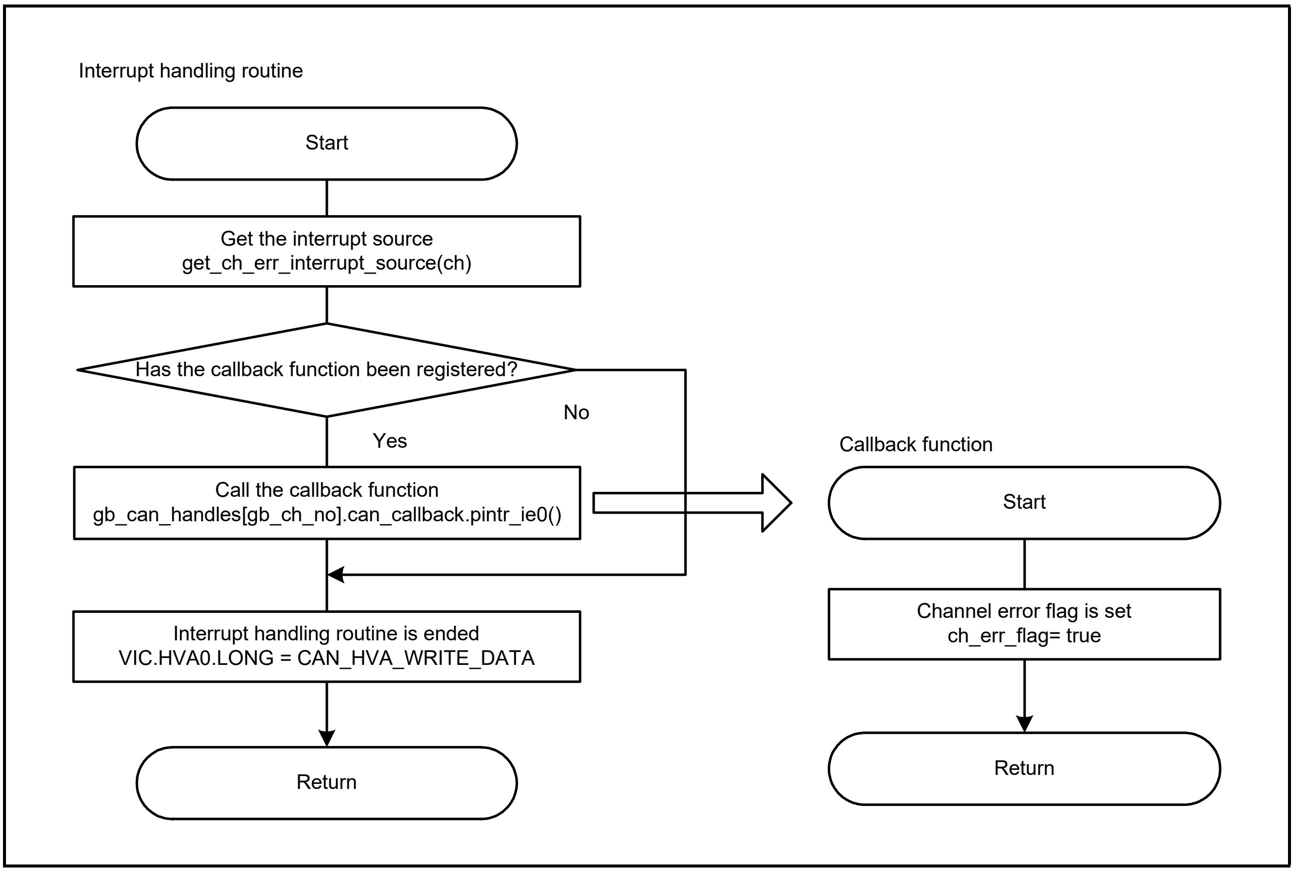


Figure 10.34 Handling of the Callback Function for a CANm Error

(3) (RSCAN:CANRFI) – A callback function which is called on occurrence of a CAN reception FIFO interrupt
Interrupt handler: void user_rx_fifo_isr(void)
Callback function: void user_rx_fifo_callback(void)
With the settings for the reception of messages by the reception FIFO buffer, the interrupt handler calls this callback function when the reception FIFO buffer becomes full of messages.
A flowchart for the interrupt handling routine and the corresponding callback function is shown below.

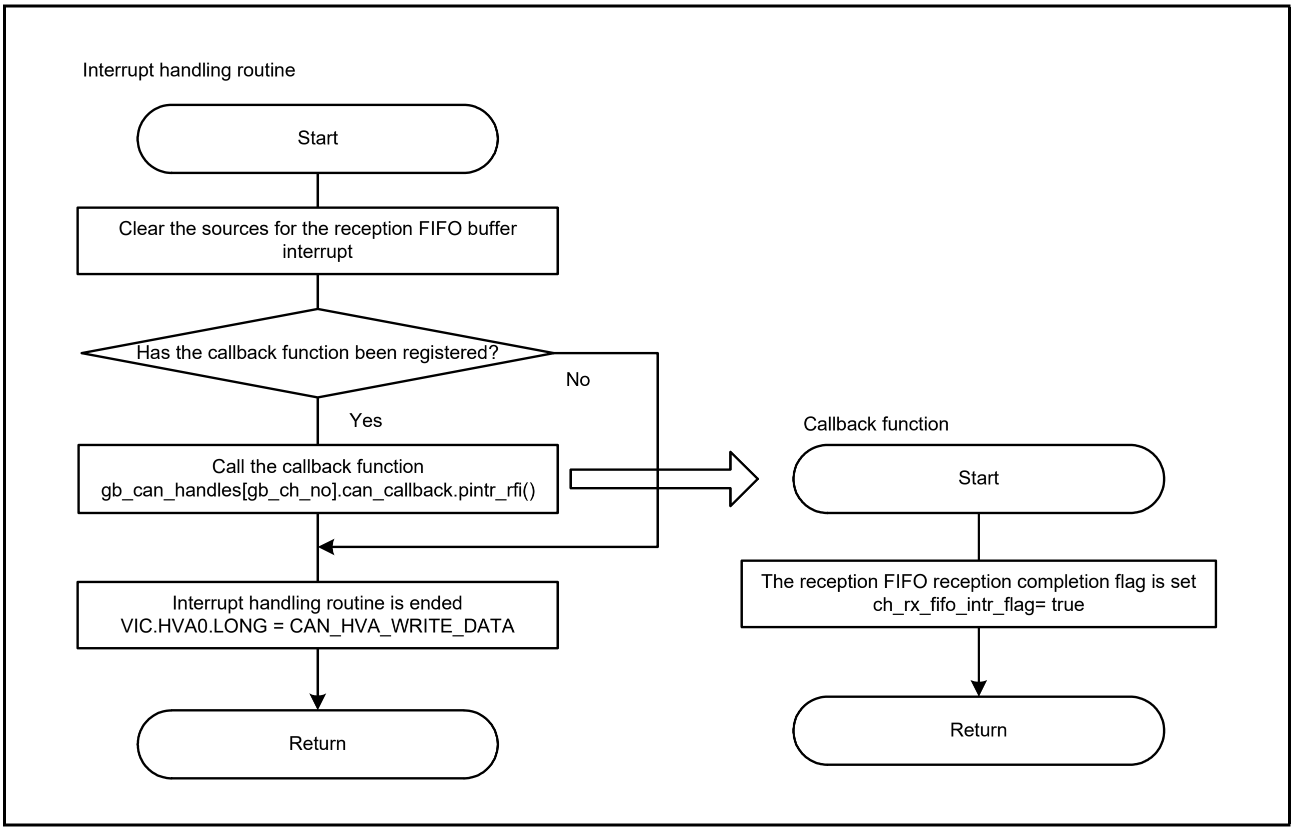


Figure 10.35 Handling of the Callback Function for a CAN Reception FIFO Interrupt

(4) (RSCAN:CANTIm) – A callback function which is called on occurrence of a CANm transmission interrupt
Interrupt handler: void user_ch0_tx_isr(void), void user_ch1_tx_isr(void)
Callback function: void user_ch0_tx_callback(void), void user_ch1_tx_callback(void)
This callback function is called by the given interrupt handler on completion of transmission of a message.
A flowchart for the interrupt handling routine and the corresponding callback function is shown below.

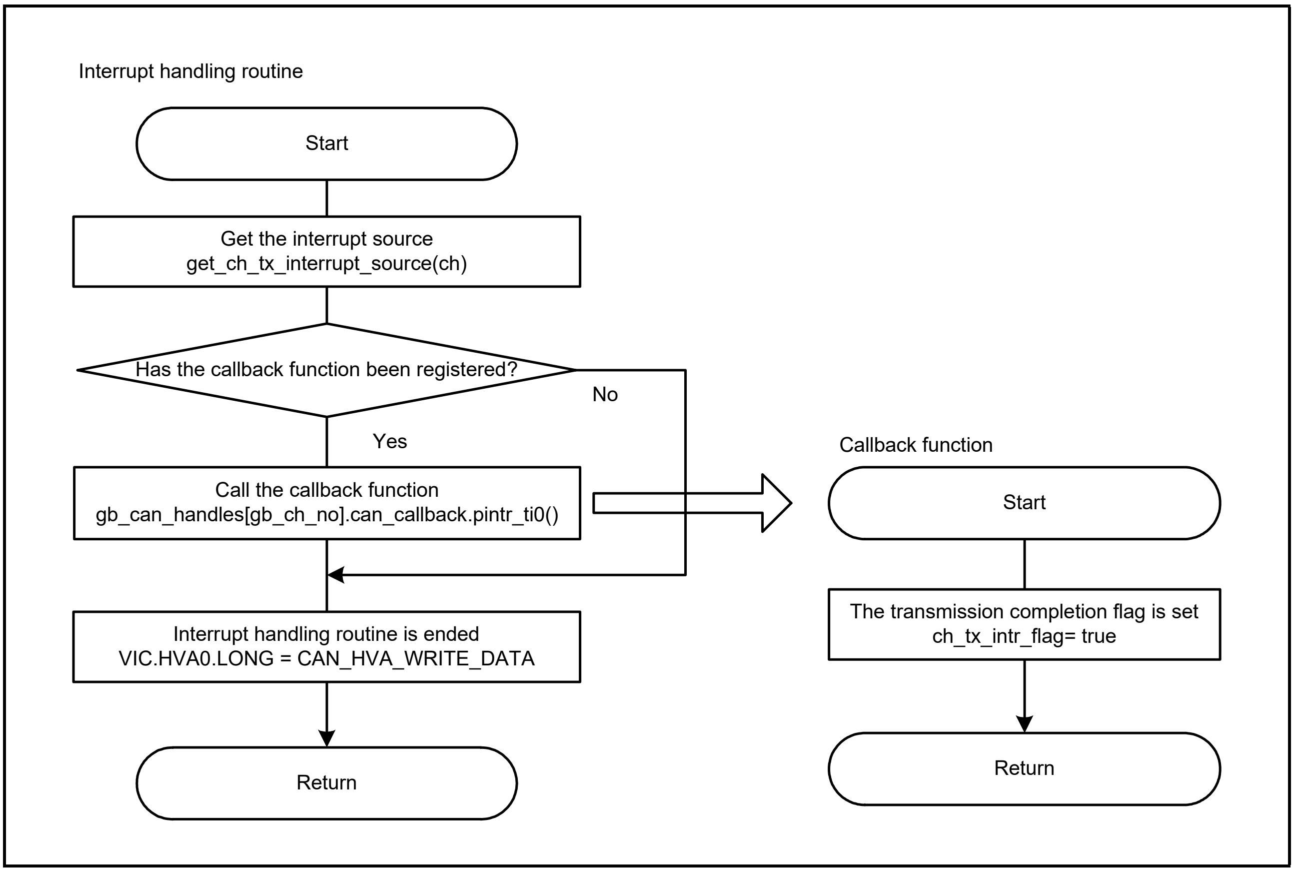


Figure 10.36 Handling of the Callback Function for a CANm Transmission Interrupt

(5) (RSCAN:CANFIRm) – A callback function that is called on occurrence of a CANm transmission-and-reception FIFO reception interrupt
Interrupt handler: void user_ch0_rx_fifo_isr(void), void user_ch1_rx_fifo_isr(void)
Callback function: void user_ch0_rx_fifo_callback(void), void user_ch1_rx_fifo_callback(void)
With the settings for the reception of messages by the transmission-and-reception FIFO buffer, the interrupt handler calls this callback function when the transmission-and-reception FIFO buffer becomes full of messages.
A flowchart for the interrupt handling routine and the corresponding callback function is shown below.

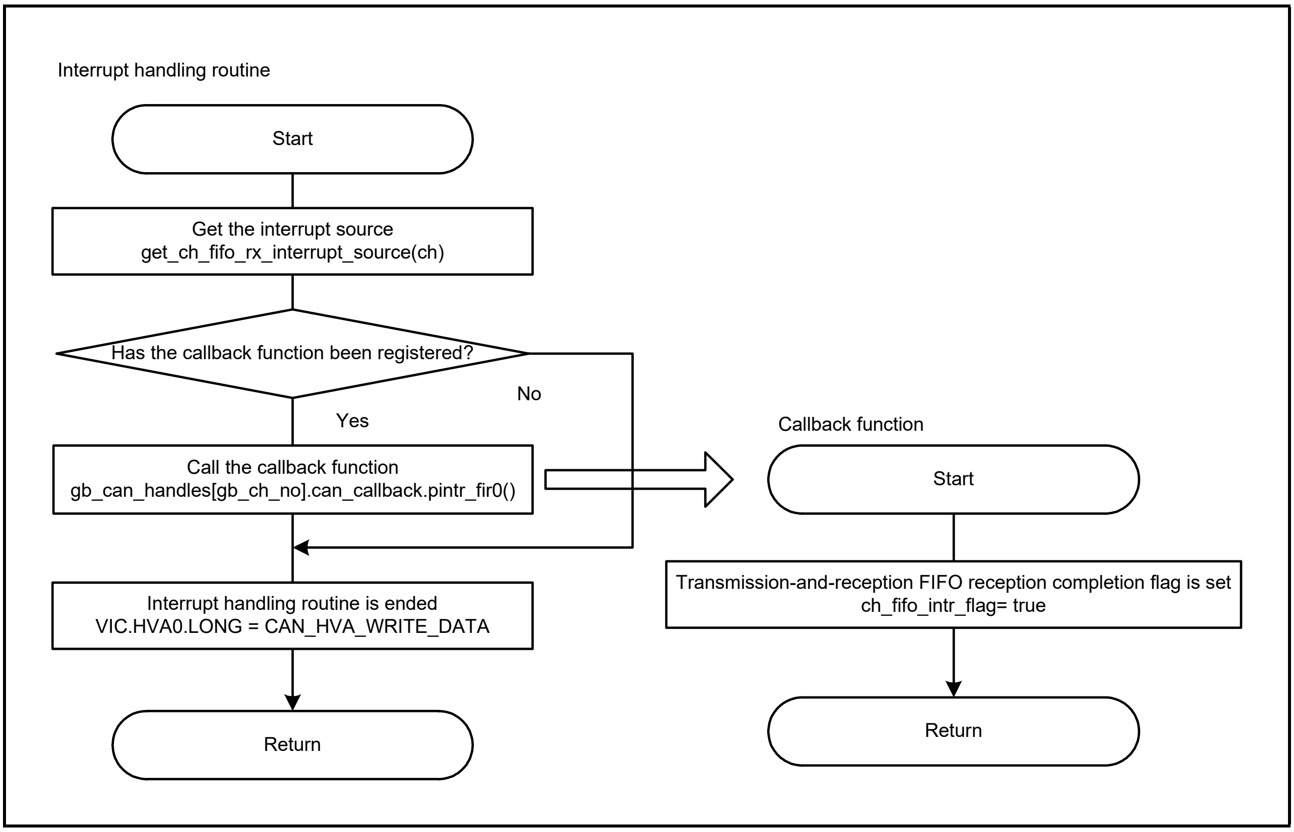


Figure 10.37 Handling of the Callback Function for a CANm Transmission-and-Reception FIFO Reception Completion Interrupt

(6) (SCIFA:RXIF2) – A callback function which is called on occurrence of a reception FIFO data full interrupt
Interrupt handler: void scifa_key_input_isr(void)
Callback function: void key_handler_callback(void)
This callback function is called from the handling routine for the pressing of a key producing an interrupt from the SCIFA module.
When this sample program is being used for transmission, messages are repeatedly transmitted to the receiving side. The repeated transmission is stopped by pressing any key, which causes the SCIFA module to set a flag (key_in_flag).
A flowchart for the interrupt handler and the corresponding callback function is shown below.

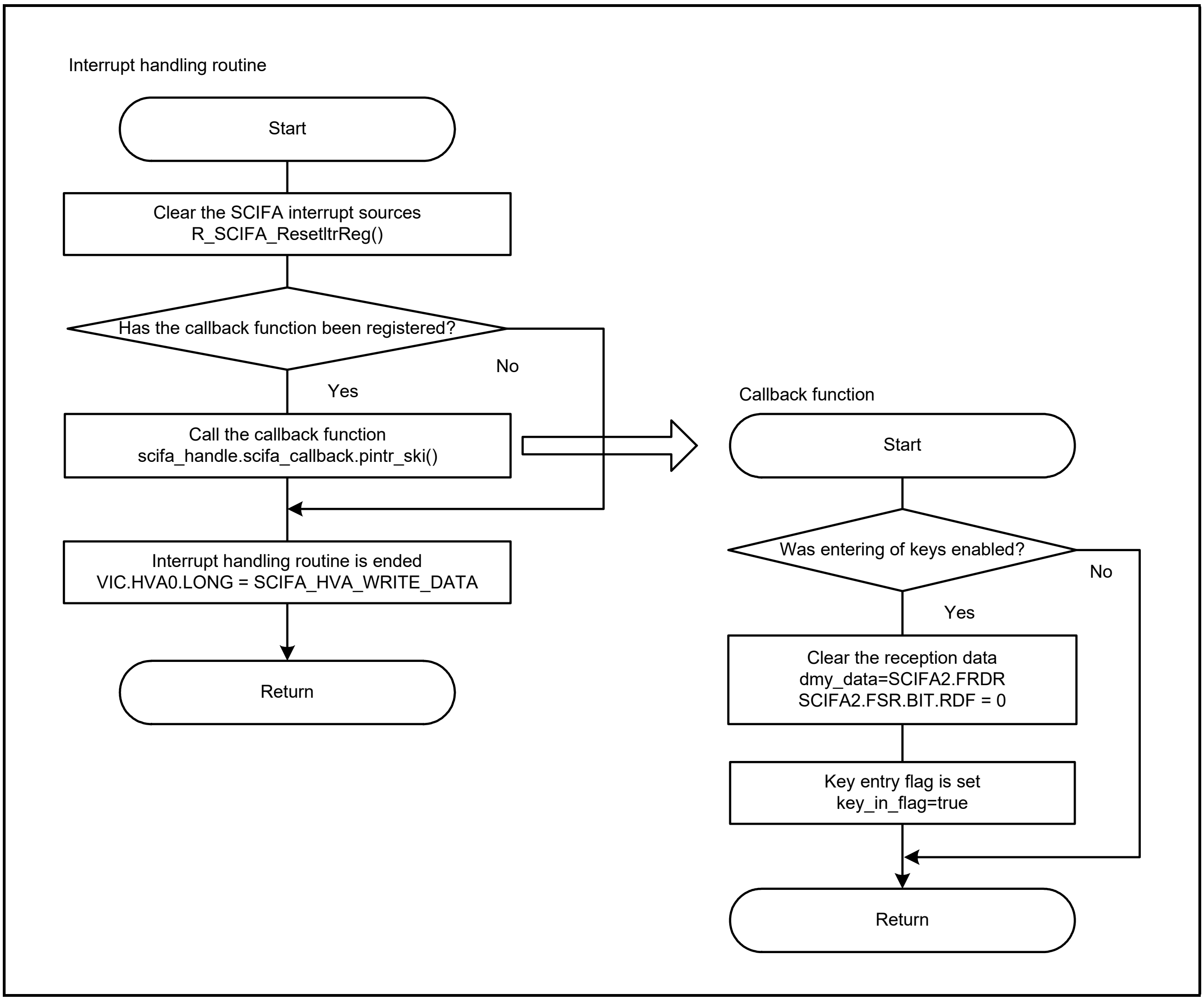


Figure 10.38 Handling of the Callback Function for a Reception FIFO Data Full Interrupt of SCIFA

11. Sample Codes

Download the sample program from the Renesas Electronics website.

12. Reference Documents

- User's Manual: Hardware
  RZ/T1 Group User's Manual: Hardware
  (Download the latest version from the Renesas Electronics website.)

- RZ/T1 CPU Board RTK7910022C00000BR User's Manual
  (Download the latest version from the Renesas Electronics website.)

- Technical Update/Technical News
  (Download the latest version from the Renesas Electronics website.)

- User's Manual: Development Environment
  For IAR integrated development environment (IAR Embedded Workbench® for Arm), download the latest version from the IAR systems website.
  For Arm software development tools (Arm compiler toolchain, Arm DS-5™, etc.), download the latest version from the Arm website.
  For Renesas integrated development environment (e2studio, etc.), download the latest version from the Renesas Electronics website.

Website and Support

Renesas Electronics Website

http://www.renesas.com/

Inquiries

http://www.renesas.com/contact/

Revision History	Application Note: CAN Interface Sample Program

Rev.	Date	Description	
		Page	**Summary**
1.00	May. 24, 2016	—	First Edition issued
1.10	Sep. 19, 2017	2. Operating Environment	
		9	Table 2.1 Operating Environment, modified
		6. CAN Configuration	
		15	Figure 6.2 Transitions between Global Modes: Arrow directions corrected
		17	Figure 6.3 Transition Diagram among Channel Modes: Changed (SCF → SOF, TEC > 256 → TEC > 255)
		10. Software	
		85	10.6, can_rx_rule_t: Comment modified
		90	Figure 10.13 Transitions between Global Modes: Changed (arrow directions corrected)
		92	Figure 10.14 Transition Diagram among Channel Modes: Changed (SCF → SOF, TEC > 256 → TEC > 255)
		117	10.8.4, (1) void trx_demo_fifo(void): Function name changed
		118	10.8.4, (1) Message transmission while receiving a different message at the same time: Function name changed
		129	Figure 10.32 Sending a Message from the Transmission-and-Reception FIFO Buffer in Transmission Mode and Receiving It at the Transmission-and-Reception FIFO Buffer in Reception Mode (2/2): Function name changed (write_buffer → write_fifo)
		131	10.8.6 Callback Processing: can_handle_t gb_can_handles[CAN_NUM]: Comment modified
		12. Reference Documents	
		139	RZ/T1 CPU Board RTK7910022C00000BR User’s Manual; Revision number deleted
1.20	Dec. 12, 2018	2. Operating Environment	
		9	Table 2.1 Operating Environment: The description on the integrated development environment, modified
		10. Software	
		102	10.7.27 R_CAN_GetInterruptSource: Description and Arguments, modified
		12. Reference Documents	
		139	"ARM" changed to "Arm"

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins
   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.
   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.
2. Processing at Power-on
   The state of the product is undefined at the moment when power is supplied.
   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.
3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.
4. Clock Signals
   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.
   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.
5. Differences between Products
   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.
   — The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics Corporation
TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan

Renesas Electronics America Inc.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709 Quantum Plaza, No.27 ZhichunLu, Haidian District, Beijing, 100191 P. R. China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, 200333 P. R. China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 7.2