RL78/F24

RS-CANFD lite Module Software Integration System

Introduction

This application note describes the RS-CANFD lite Module.

Target Device

RL78/F24 Group

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Supported Compilers

- Renesas Electronics C/C++ Compiler Package for RL78 Family
- IAR C/C++ Compiler for Renesas RL78

Contents

1. Overview

This module uses the RS-CANFD lite to implement CAN frame transmission and reception.

2. API Information

Operations of this module has been confirmed under the following conditions.

2.1 Hardware Requirements

The MCU used in the development must support the following function.

- RS-CANFD lite

2.2 Hardware Resource Requirements

In addition to the RS-CANFD lite, this module requires:

- Two pins allocated for the CAN channel

2.3 Software Requirements

This module depends on the following module:

- Board support package (r_bsp) v1.20 and above

2.4 Limitations

RS-CANFD lite functions that this module does not support are listed below.

- PNF (Pretended Network Filter List)
- Disable receive rule entries
- Global reset by GRSTC register
- One-shot transmission function
- Capture of Error Occurrence Counter
- Capture of Successful Occurrence Counter
- Transmit history function
- Test function (Listen-only mode, Loopback, RAM test, etc.)
- CAN RAM ECC

2.5 Supported Toolchains

Module operations have been confirmed on the following toolchains:

- Renesas CS+ for CC V8.07.00
- IAR Embedded Workbench for Renesas RL78 4.21.3

2.6 Header Files

All API calls and their supporting interface definitions are located in “r_rscanfd_rl78_if.h”. Build-time configuration options are selected or defined in the file “r_rscanfd_rl78_config.h”.

Both of these files should be included by the user's application.

2.7 Integer Types

This module uses ANSI C99. These types are defined in “stdint.h”.

2.8 Configuration Overview

Configuration options for this module are set by the user via the file "r_rscanfd_rl78_config.h".

The following table provides the names and setting values for the configuration option settings used this module.

Define	Default Value	Description	Target Register
CAN_CFG_PARAM_CHECKING_ENABLE	BSP_CFG_PARAM_CHECKING_ENABLE	Setting to 1 includes parameter checking in the code. Setting to 0 removes parameter checking from the code. Setting to "BSP_CFG_PARAM_CHECKING_ENABLE" depends on BSP setting.	-
CAN_CFG_CLOE_AND_FDOE	0	Setting to 1 enables FD only mode. Setting to 2 enables Classical only mode.	C0FDCFGH, C0FDCFGL
CAN_CFG_REFE	0	Setting to 1 enables RX edge filter. For details, refer to the User's Manual: Hardware.	C0FDCFGH, C0FDCFGL
CAN_CFG_TDCO	0x00	Set the offset of TDC (Transceiver Delay Compensation). For details, refer to the User's Manual: Hardware.	C0FDCFGH, C0FDCFGL
CAN_CFG_ESIC	0	Set the ESI. For details, refer to the User's Manual: Hardware.	C0FDCFGH, C0FDCFGL
CAN_CFG_TDCE_AND_TDCOC	0	Setting to 1 or 2 enables TDC (Transceiver Delay Compensation). (1: SSP offset = Measured delay + CAN_CFG_TDCO 2: SSP offset = CAN_CFG_TDCO) For details, refer to the User's Manual: Hardware.	C0FDCFGH, C0FDCFGL
CAN_CFG_ITRCP	0	Set definition of a reference clock for the FIFO interval timer source clock. Setting to 0, timer is disabled.	GCFGH, GCFGL
CAN_CFG_TSSS	0	Select the clock source for the Timestamp counter. Setting to 0, clock source for Timestamp counter is peripheral clock. Setting to 1, clock source for Timestamp counter is bit time clock. Do not set to 1, when CAN-FD communication will be used.	GCFGH, GCFGL
CAN_CFG_TSP	0	Set the period of the clock source used for the Timestamp counter. For details, refer to the User's Manual: Hardware.	GCFGH, GCFGL
CAN_CFG_CMPOC	0	Control the message payload acceptance mechanism in the case when the received payload is higher than the Message Buffer payload size. Setting to 0, message is rejected. Setting to 1, message payload is cut to fit to configured message size.	GCFGH, GCFGL
CAN_CFG_DCS	0	Select the clock source for the CAN communications. Setting to 0, clock source is internal clock Setting to 1, clock source is external clock source (X1 clock direct).	GCFGH, GCFGL

CAN_CFG_MME	0	Setting to 1 enables the Mirror Mode for all CAN channels.	GCFGH, GCFGL
CAN_CFG_DRE	0	If this setting is 1 and CAN_CFG_DCE is 1, then RS-CANFD lite will store the configured value (CAN_CFG_RULEx_GAFLDLC) of DLC in the destination RX Message Buffer or FIFO buffer if the DLC check passes. Otherwise the DLC value in the destination RX Message Buffer or FIFO buffer is unchanged.	GCFGH, GCFGL
CAN_CFG_DCE	0	Setting to 1 enables the DLC check for all CAN channels.	GCFGH, GCFGL
CAN_CFG_TPRI	0	Select the transmission priority for all CAN channels. Setting to 0, transmission priority is ID Priority. Setting to 1, transmission priority is Message Buffer Number Priority.	GCFGH, GCFGL
CAN_CFG_NBRP	5	Set Nominal Baud Rate Prescaler division ratio. (Prescaler division ratio = CAN_CFG_NBRP + 1) For details, refer to the User's Manual: Hardware.	C0NCFGH, C0NCFGL
CAN_CFG_NMNL_TSEG1	63	Set Nominal Timing Segment 1. For details, refer to the User's Manual: Hardware.	C0NCFGH, C0NCFGL
CAN_CFG_NMNL_TSEG2	16	Set Nominal Timing Segment 2. For details, refer to the User's Manual: Hardware.	C0NCFGH, C0NCFGL
CAN_CFG_NMNL_SJW	16	Set Nominal Synchronization Jump Width. For details, refer to the User's Manual: Hardware.	C0NCFGH, C0NCFGL
CAN_CFG_DBRP	5	Set Data Baud Rate Prescaler division ratio. (Prescaler division ratio = CAN_CFG_DBRP + 1) For details, refer to the User's Manual: Hardware.	C0DCFGH, C0DCFGL
CAN_CFG_DATA_TSEG1	13	Set Data Timing Segment 1. For details, refer to the User's Manual: Hardware.	C0DCFGH, C0DCFGL
CAN_CFG_DATA_TSEG2	6	Set Data Timing Segment 2. For details, refer to the User's Manual: Hardware.	C0DCFGH, C0DCFGL
CAN_CFG_DATA_SJW	6	Set Data Synchronization Jump Width. For details, refer to the User's Manual: Hardware.	C0DCFGH, C0DCFGL
CAN_CFG_RMPLS	0	Set the RX message buffer payload data size. For details, refer to the User's Manual: Hardware.	RMNB
CAN_CFG_NRXMB	0	Set the number of RX Message Buffers. Setting to 0 makes RX Message Buffers unavailable.	RMNB

CAN_CFG_RFx_RFIGCV (x=0, 1)	0	Select the counter value of the RX FIFO for generation of RX FIFO Interrupt. This value represent fractions of the RX FIFO depth for which Interrupt is generated. For details, refer to the User's Manual: Hardware.	RFCCk
CAN_CFG_RFx_RFIM (x=0, 1)	0	Select the Interrupt generation condition for the RX FIFO. Setting to 0, Interrupt generated when RX FIFO counter reaches CAN_CFG_RFx_RFIGCV value from values smaller than CAN_CFG_RFx_RFIGCV. Setting to 1, Interrupt generated at the end of every received message storage.	RFCCk
CAN_CFG_RFx_RFDC (x=0, 1)	0	Select the depth of the RX FIFO in terms of number of Messages. If the RX FIFO depth is configured to 0 Messages then the RX FIFO cannot be used. For details, refer to the User's Manual: Hardware.	RFCCk
CAN_CFG_RFx_RFPLS (x=0, 1)	0	Set the max number of Bytes which can be received by RX FIFO. For details, refer to the User's Manual: Hardware.	RFCCk
CAN_CFG_RFx_RFIE (x=0, 1)	0	Setting to 1 enables generation of the RX FIFO Interrupt.	RFCCk
CAN_CFG_CFITT	0	Select the delay in the start of transmission for all messages transmitted from Common FIFO when configured in TX mode. For details, refer to the User's Manual: Hardware.	CFCC
CAN_CFG_CFDC	0	Select the depth of the Common FIFO in terms of number of Messages. If the Common FIFO depth is configured to 0 Messages then the Common FIFO cannot be used. For details, refer to the User's Manual: Hardware.	CFCC
CAN_CFG_CFTML	0	Select the normal transmit Message Buffer position where the TX FIFO is linked to, for transmission scanning.	CFCC
CAN_CFG_CFIGCV	0	Select the message counter value for the generation of the Common FIFO Interrupt. This value represent fractions of the Common FIFO depth at which Interrupt is to be generated. For details, refer to the User's Manual: Hardware.	CFCC

CAN_CFG_CFIM	0	Select the Interrupt generation condition for the Common FIFO. • Setting to 0 — RX FIFO Mode: RX Interrupt generated when Common FIFO counter reaches CAN_CFG_CFIGCV value from a lower value. — TX FIFO Mode: TX Interrupt generated when Common FIFO transmits the last message successfully. • Setting to 1 — RX FIFO Mode: RX Interrupt generated at the end of every received message storage. — TX FIFO Mode: TX Interrupt generated for every successfully transmitted message.	CFCC
CAN_CFG_CFITR	0	Select the resolution of the Reference Clock for the Interval Transmission Timer (Peripheral Clock is the source for the Reference Clock). Setting to 0, Reference Clock Period x1. Setting to 1, Reference Clock Period x10.	CFCC
CAN_CFG_CFITSS	0	Select the basic clock source for the Interval Transmission Timer. Setting to 0, Reference Clock (x1 / x10 period). Setting to 1, Bit Time Clock of related channel (FIFO is linked to fixed channel). Do not set to 1, when CAN-FD communication will be used.	CFCC
CAN_CFG_CFM	0	Select the Mode of the Common FIFO. Setting to 0, RX FIFO Mode. Setting to 1, TX FIFO Mode.	CFCC
CAN_CFG_CFPLS	0	Set the max number of Bytes which can be received or transmitted by Common FIFO. For details, refer to the User's Manual: Hardware.	CFCC
CAN_CFG_CFTXIE	0	Setting to 1 enables generation of the Common FIFO Interrupt when the Interrupt flag is set after transmission of a frame from the corresponding FIFO.	CFCC
CAN_CFG_CFRXIE	0	Setting to 1 enables generation of the Common FIFO Interrupt when the Interrupt flag is set after reception of a frame in the corresponding FIFO.	CFCC
CAN_CFG_THLDTE	0	Select the conditions for storing an entry in the TX History list after successful transmission. Setting to 0, TX FIFO. Setting to 1, Flat TX MB + TX FIFO.	THLCC

CAN_CFG_THLIM	0	Select the Interrupt generation condition for the FIFO. Setting to 0, Interrupt generated if TX History List level reaches 3/4 of the TX History List depth. Setting to 1, Interrupt generated for every successfully stored entry.	THLCC
CAN_CFG_THLIE	0	Setting to 1, TX history list generated.	THLCC
CAN_CFG_THLE	0	Setting to 1, TX history list enabled.	THLCC
CAN_CFG_CMPOFIE	0	Setting to 1, an Interrupt will be generated when a CAN-FD message payload overflow condition occurs.	GCTRH, GCTRL
CAN_CFG_THLEIE	0	Setting to 1, an Interrupt will be generated when a TX History List Entry Lost condition occurs.	GCTRH, GCTRL
CAN_CFG_MEIE	0	Setting to 1, an Interrupt will be generated when a Message Lost condition occurs.	GCTRH, GCTRL
CAN_CFG_DEIE	0	Setting to 1, an interrupt will be generated when a DLC error is detected in received frames.	GCTRH, GCTRL
CAN_CFG_ERRD	0	Set the display mode of the error flag bits (bits 14 to 8) in the Channel Error Flag Register (C0ERFL). Setting to 0, Only the 1st set of error codes displayed. Setting to 1, Accumulated error codes displayed.	C0CTRH, C0CTRL
CAN_CFG_BOM	0	Set the timing of the recovery from Bus-Off mode of the RS-CANFD lite Channel. For details, refer to the User's Manual: Hardware.	C0CTRH, C0CTRL
CAN_CFG_TDCVFIE	0	Setting to 1 enables Transceiver Delay Compensation Violation Interrupt. Do not set to 1 when Classical only mode.	C0CTRH, C0CTRL
CAN_CFG_TAIE	0	Setting to 1 enables Transmission Abort Interrupt.	C0CTRH, C0CTRL
CAN_CFG_ALIE	0	Setting to 1 enables Arbitration Lost Interrupt.	C0CTRH, C0CTRL
CAN_CFG_BLIE	0	Setting to 1 enables Bus Lock Interrupt.	C0CTRH, C0CTRL
CAN_CFG_OLIE	0	Setting to 1 enables Overload Interrupt.	C0CTRH, C0CTRL
CAN_CFG_BORIE	0	Setting to 1 enables Bus-Off Recovery Interrupt.	C0CTRH, C0CTRL
CAN_CFG_BOEIE	0	Setting to 1 enables Bus-Off Entry Interrupt.	C0CTRH, C0CTRL
CAN_CFG_EPIE	0	Setting to 1 enables Error Passive Interrupt.	C0CTRH, C0CTRL
CAN_CFG_EWIE	0	Setting to 1 enables Error Warning Interrupt.	C0CTRH, C0CTRL
CAN_CFG_BEIE	0	Setting to 1 enables Bus Error Interrupt.	C0CTRH, C0CTRL

CAN_CFG_TMIEx (x=0 to 3)	0	Setting to 1, an interrupt will be generated at the end of a successful transmission from the corresponding Message Buffer. Do not set to 1 if the corresponding TX Message Buffer is linked to a Common FIFO via CAN_CFG_CFTML.	TMIEC
CAN_CFG_TSCCFG	0	These bits configure the different capture points of the timestamp. For details, refer to the User's Manual: Hardware.	GFDCFG
CAN_CFG_RPED	0	Setting to 1 disables the protocol exception event detection.	GFDCFG
CAN_CFG_RMIE_ALL	0	Setting to 1, interrupts will be generated at the end of a successful reception from the all Receive Message Buffer. Setting to 2 for enable each buffer. (Enables CAN_CFG_RMIE_VALUE)	RMIEC
CAN_CFG_RMIE_VALUE	0x0000	Setting bit n to 1, an interrupt will be generated at the end of a successful reception from the corresponding Receive Message Buffer n.	RMIEC
CAN_CFG_INTRCANGRFR_USE	0	Select use / no use of CAN global receive FIFO interrupt. Setting to 0, no use. Setting to 1, use.	RCANGRFRMK
CAN_CFG_INTRCANGRFR_LEVEL	2	Set the priority level for CAN global receive FIFO interrupt.	RCANGRFRPR1, RCANGRFRPR0
CAN_CFG_INTRCANGRVC_USE	0	Select use / no use of CAN global receive message buffer interrupt. Setting to 0, no use. Setting to 1, use.	RCANGRVCMK
CAN_CFG_INTRCANGRVC_LEVEL	2	Set the priority level for CAN global receive message buffer interrupt.	RCANGRVCPR1, RCANGRVCPR0
CAN_CFG_INTRCANGERR_USE	0	Select use/no use of CAN global error interrupt. Setting to 0, no use. Setting to 1, use.	RCANGERRMK
CAN_CFG_INTRCANGERR_LEVEL	2	Set the priority level for CAN global error interrupt	RCANGERRPR1, RCANGERRPR0
CAN_CFG_INTRCAN0TRM_USE	0	Select use / no use of CAN0 channel transmit interrupt. Setting to 0, no use. Setting to 1, use.	RCAN0TRMMK
CAN_CFG_INTRCAN0TRM_LEVEL	2	Set the priority level for CAN0 channel transmit interrupt.	RCAN0TRMPR1, RCAN0TRMPR0

CAN_CFG_INTRCAN0CFR_USE	0	Select use / no use of CAN0 Common FIFO receive interrupt. Setting to 0, no use. Setting to 1, use.	RCAN0CFRMK
CAN_CFG_INTRCAN0CFR_LEVEL	2	Set the priority level for CAN0 Common FIFO receive interrupt.	RCAN0CFRPR1, RCAN0CFRPR0
CAN_CFG_INTRCAN0ERR_USE	0	Select use / no use of CAN0 channel error interrupt. Setting to 0, no use. Setting to 1, use.	RCAN0ERRMK
CAN_CFG_INTRCAN0ERR_LEVEL	2	Set the priority level for CAN0 channel error interrupt.	RCAN0ERRPR1, RCAN0ERRPR0
CAN_CFG_INTRCAN0WUP_USE	0	Select use / no use of CAN0 wakeup interrupt. Setting to 0, no use. Setting to 1, use.	RCAN0WUPMK
CAN_CFG_INTRCAN0WUP_LEVEL	2	Set the priority level for CAN0 wakeup interrupt.	RCAN0WUPPR1, RCAN0WUPPR0
CAN_CFG_CALLBACK_XXX	my_can_xxx_callback	Set the user callback function names	-
CAN_CFG_CRXD0_PU	0	Set the CRXD0 pin on-chip pull-up resistor.	PUxx
CAN_CFG_CRXD0_PITHL	0	Set the input buffer threshold for CRXD0 pin. (0: Schmitt1, 1: Schmitt 3)	PITHLx

2.9 Code Size

The code size is shown below.

Condition: - Parameter check enabled
- RXMB and RX FIFO used
- Common FIFO used in TX mode
- All interrupt handlers are valid

[CC-RL]

Tools used

Renesas Electronics CS+ for CC V8.07.00

Renesas Electronics C/C++ compiler for R78 Family V.1.11.0

(Optimization level: Perform the default optimization (None))

ROM size: 3409 bytes

RAM size: 0 byte

Stack: 22 bytes

[IAR]

Tools used

IAR Systems IAR Embedded Workbench for Renesas RL78 4.21.3

IAR Systems IAR C/C++ Compiler for Renesas RL78 4.21.3.2447

(Optimization level: Medium)

ROM size: 3722 bytes

RAM size: 0 byte

Stack: 20 bytes

2.10 API Data Types

The data types, structures, etc. used by this module are shown below.

2.10.1 Data Types

The data types used by this module are shown below.

Data types are defined in "r_rscanfd_rl78_if.h".

Data Type	Actual Data Type	Description
int8_t	signed char	BSP calls stdint.h.
int16_t	signed short	Same as above.
uint8_t	unsigned char	Same as above.
uint16_t	unsigned short	Same as above.
e_can_err_t e_can_txb_result_t	unsigned int	Error code, return value.
can_rxfifo_t	unsigned char	RX FIFO buffer number.
can_txbuf_t	unsigned char	TX Message Buffer number.
can_rxbuf_t	unsigned char	RX Message Buffer number.
can_length_t	unsigned char	CAN data length.
can_storage_t	unsigned short	Receive frame storage buffer type
st_can_tx_frame_t st_can_rx_frame_t	union	CAN send/receive data. Refer 2.10.2 Structure, Union for detail.

2.10.2 Structure, Union

The structure, union used by this module are shown below.

Structure, union are defined in "r_rscanfd_rl78_if.h".

2.10.2.1 u_can_data_t

Data type name

u_can_data_t

Description

A union for storing data bytes for sending and receiving at CAN.

Defines the area for storing 64 bytes (32 word) of CAN frame data.

Use DW for word access and DB for byte access.

Definitions

```
typedef union
{
    uint16_t   DW[32u];       /* Data Word   */
    uint8_t    DB[64u]; /* Data Byte   */
} u_can_data_t;
```

2.10.2.2 u_can_tx_head_t, u_can_rx_head_t

Data type name

u_can_tx_head_t

u_can_rx_head_t

Description

A union for storing ID for sending and receiving at CAN.

Defines the area for storing CAN ID, IDE, RTR, etc. of the CAN frame.

Definitions

```
typedef union
{
    uint16_t        Word[CAN_TX_HEAD_WORD_NUM]; /* Word access */
    struct
    {
        /* ---- ID, THLEN, RTR and IDE (2 words) ---- */
        uint32_t    ID   :29;   /* CAN ID                          */
        uint32_t    THLEN:1;    /* THLEN  if 1 then store in THL   */
        uint32_t    RTR  :1;    /* RTR  0:Data 1:Remote(Classical) */
        uint32_t    IDE  :1;    /* IDE  0:Standard 1:Extended      */

        /* ---- Classical/FD, DLC (0.5 word) ---- */
        uint8_t     FDCTR:3;    /* FDF/BRS/ESI                     */
        uint8_t     :1;
        uint8_t     DLC  :4;    /* DLC  0-15                       */

        /* ---- Label and Time Stamp (1.5 words) ---- */
        uint8_t     IFL:2;      /* Information label               */
        uint8_t     :6;
        uint16_t    LBL;        /* TX label                        */
    } Bits;                                     /* Bit access */
} u_can_tx_head_t;

typedef union
{
    uint16_t        Word[CAN_RX_HEAD_WORD_NUM]; /* Word access */
    struct
    {
        /* ---- ID, RTR and IDE (2 words) ---- */
        uint32_t    ID :29;     /* CAN ID                      */
        uint32_t    :1;
        uint32_t    RTR:1;      /* RTR  0:Data 1:Remote(Classical) */
        uint32_t    IDE:1;      /* IDE  0:Standard 1:Extended */

        /* ---- Classical/FD, DLC (0.5 words) ---- */
        uint8_t     ESI:1;      /* ESI  0:Error Active 1:Error Passive */
        uint8_t     BRS:1;      /* BRS  0:Only Nominal 1:Use Data Baud Rate */
        uint8_t     FDF:1;      /* FDF  0:Classical 1:CAN-FD  */
        uint8_t     :1;
        uint8_t     DLC:4;      /* DLC  0-15                  */

        /* ---- Label and Time Stamp (2.5 words) ---- */
        uint8_t     IFL:2;      /* Information label          */
        uint8_t     :6;
        uint16_t    LBL;        /* RX label                   */
        uint16_t    TS;         /* Time Stamp                 */
    } Bits;                                     /* Bit access */
} u_can_rx_head_t;
```

	When sending u_can_tx_head_t	When receiving u_can_rx_head_t
IDL	The 29 bits of the ID. For the standard ID, specify 0 for the upper 18 bits.	The 29 bits of the ID. For the standard ID, the value read from the upper 18 bits is 0.
THLEN	Whether TX history is stored 0: Not store 1: Store Macro CAN_THL_XXX are available at the specified time.	Nothing
RTR	Message frame format (RTR bit). 0: Data Frame, 1: Remote Frame. Remote frame is used only for the classical CAN frame.	
IDE	Message ID format (IDE bit). 0: Standard ID, 1: Extended ID.	
DLC	DLC value of sending message. The data length corresponding to the DLC value depends on the value of the FDF bit. You can use the following macros to set the values: FDF=0：CAN_DLC_LEN0 to CAN_DLC_LEN8 FDF=1：CAN_DLC_LEN0 to CAN_DLC_LEN8, CAN_FD_DLC_LEN12 to CAN_FD_DLC_LEN64	DLC value of receiving message.
FDCTR	FDF/BRS/ESI Bit 000b: Classical CAN Frame 100b to 111b: CAN FD frame 100b: No data bitrate and ESI 101b: Do not use data bitrate 110b: Do not use ESI 111b: Using data bitrate and ESI Macro CAN_FDCTR_XXX are available at the specified time.	Nothing
ESI	Nothing	ESI bit 0: Error Active Node, 1: Error Passive Node. When FDF = 0, it is fixed to 0 when receiving and Specify 0 when sending.
BRS	Nothing	BRS bit 0: no bit rate switch, 1: bit rate switch. When FDF = 0, it is fixed to 0 when receiving and Specify 0 when sending.
FDF	Nothing	FDF bit 0: Classical CAN frame, 1: CAN-FD frame.
IFL	Label information of TX history. (2bits)	Label information of receiving message. (2bits)
LBL	Label information of TX history. (16bits)	Label information of receiving message. (16bits)
TS	Nothing.	Timestamp value of receiving message.

2.10.2.3 st_can_tx_frame_t, st_can_rx_frame_t

Data type name

st_can_tx_frame_t

st_can_rx_frame_t

Description

A union for storing data for sending and receiving at CAN.

Defines the area for storing CAN frame information for one frame.

Definitions

```
typedef struct
{
    u_can_tx_head_t  Head;
    u_can_data_t     Data;
} st_can_tx_frame_t;

typedef struct
{
    u_can_rx_head_t  Head;
    u_can_data_t     Data;
} st_can_rx_frame_t;
```

2.10.2.4 st_can_filter_t, st_can_filter_opt_t

Data type name

st_can_filter_t

st_can_filter_opt_t

Description

A structure for storing CAN receive rules.

Receive filter: st_can_filter_t

Receive filter options: st_can_filter_opt_t

Defines the area where one frame of CAN frame information is stored.

Definitions

```
typedef struct
{
    uint32_t ID     :29;    /* CAN ID                                          */
    uint32_t :3;
    uint32_t ID_MASK:29;    /* CAN ID Mask                                     */
    uint32_t :3;
    uint8_t  RTR_TYPE;      /* RTR_TYPE  0:Data 1:Remote (classical) 2:Any */
    uint8_t  IDE_TYPE;      /* IDE_TYPE  0:Standard 1:Extend 2:Any         */
} st_can_filter_t;

/* RX rule filter option */
typedef struct
{
    uint16_t LBL;           /* RX label                    */
    uint16_t IFL :2;        /* Information label           */
    uint16_t DLC :4;        /* DLC (Effective if DCE=1)    */
    uint16_t LB  :1;        /* LB  0:RX frames  1:TX frames (Effective if
MME=1) */
    uint16_t :9;
} st_can_filter_opt_t;
```

		Description	remarks
st_can_filter_t	ID	For id 29-bit standard IDs, the top 18 bits must be 0.	
	ID_MASK	ID comparison bit Bits specified with 1 are compared. ID_MASK=0 stores frames that correspond to the RTR_TYPE and IDE_TYPE, regardless of the value specified for the ID.	If you do not want to compare or compare all id bits, you can use the following macros: CAN_MATCH_NO_ID_BIT CAN_MATCH_ALL_ID_BIT
	RTR_TYPE	Data format of stored messages (RTR bit) 0: Data frame 1: Remote frame 2: Optional * Remote frame is classical CAN frame only	The following macros are available when specified: CAN_RTR_DATA_FRAME CAN_RTR_RMT_RTR1_FRAME CAN_RTR_ANY_FRAME
	IDE_TYPE	Stored message ID format (IDE bit) 0: Standard ID 1: Extended ID 2: Optional	The following macros are available when specified: CAN_IDE_STD_FORMAT CAN_IDE_EXT_FORMAT CAN_IDE_ANY_FORMAT
st_can_filter_opt_t	LBL	Label information (16bit)	
	IFL	Label information (2bit)	
	DLC	DLC value When CAN_CFG_DCE =1, rs-CANFD refers to DLC checking. When CAN_CFG_DCE = 0, setting anything other than 0 has no effect.	The following macros are available when specified: CAN_DLC_LEN0 , ..., CAN_FD_DLC_LEN64
	LB	Loopback When CAN_CFG_MME=1, lb=1 stores the sending frame according to the receive rules instead of the receive frame. When CAN_CFG_MME = 0, if 1 is set, both the transmission frame and the reception frame are not stored.	The following macros are available when specified: CAN_AFL_LB_NOT_LOOPBACK CAN_AFL_LB_LOOPBACK

2.10.2.5 st_can_txhist_t

Data type name

st_can_txhist_t

Description

A structure for storing TX history list entry.

Defines the area where one entry of TX history information is stored.

Definitions

```
typedef struct
{
    can_txbuf_t     txbuf_idx;  /* TX buffer index      */
    uint16_t        TS;         /* Time Stamp           */
    uint16_t        LBL;        /* TX label             */
    uint16_t        IFL :2;     /* Information label    */
    uint16_t        :14;
} st_can_txhist_t;
```

		Description	**remarks**
st_can_txhist_t	txbuf_idx	0-3: TX messege buffer 0xFF: Not TX message buffer (Common FIFO)	
	TS	Timestamp value of TX history	
	LBL	Label information (16bit) of TX history	
	IFL	Label information (2bit) of TX history	

2.10.3 Macro

The macro used by this module are shown below.

2.10.3.1 Parameter Macro

The macro used as arguments of the API function of this module is shown below.

Macros are defined in "r_rscanfd_rl78_if.h".

Macro Name	Value	Description	Functions that use macros
CAN_WUP_UNUSE to CAN_WUP_USE	0 to 1	Use / No use of CAN Wakeup.	R_CAN_Sleep
CAN_RXFIFO0 to CAN_RXFIFO1	0 to 1	RX FIFO buffer number. (Example: RX FIFO buffer 0 -> CAN_RXFIFO0)	R_CAN_ReadRXFIFO
CAN_TXBUF0 to CAN_TXBUF3	0 to3	TX Message Buffer number.	R_CAN_SendByTXMB others
CAN_TXBUF_NOT	0xFF	Not TX message buffer (Common FIFO)	R_CAN_TxHistory
CAN_RXBUF0 to CAN_RXBUF15	0 to 15	RX Message Buffer number.	R_CAN_ReadRXMB
CAN_DLC_LEN0 to CAN_DLC_LEN8	0 to 8	DLC value (Classical CAN frame)	
CAN_DLC_LEN0 to CAN_DLC_LEN8, CAN_FD_DLC_LEN12 to CAN_FD_DLC_LEN64	0 to 8, 9 to 15	DLC value (CAN-FD frame) CAN_FD_DLC_LENx x=12,16,20,24,32,48,64	
CAN_STORE_XX	0x8000 0x0001 0x0002 0x0100	See R_CAN_AddRxRule	R_CAN_AddRxRule
CAN_STORE_XX_AND_YY	0x8001 0x8002 0x8100 0x0003 0x0101 0x0102	See R_CAN_AddRxRule	R_CAN_AddRxRule
CAN_MAX_WORD_NUM	32	Maximum number of words in CAN frame data.	

2.10.3.2 Configuration Macro

Refer 2.8 Configuration Overview.

2.11 Return Values

The API function returns a return value of type e_can_err_t except for some functions. The e_can_err_t type is found in “r_rscanfd_rl78_if.h” along with the API function declarations.

The following is a list of return values of API functions.

Type	Macro Name	Description
e_can_err_t	CAN_SUCCESS	Completed successfully.
	CAN_SUCCESS_WITH_LOST	Complete successfully, but message was lost.
	CAN_ERR_WAITING	Waiting for transition to complete
	CAN_ERR_INVALID_ARG	Argument error.
	CAN_ERR_INVALID_MODE	Invalid CAN mode
	CAN_ERR_ILLEGAL_STS	Status error. Some error occurred in the CAN module.
	CAN_ERR_BUF_BUSY	Buffer is busy.
	CAN_ERR_BUF_FULL	Buffer is full.
	CAN_ERR_BUF_EMPTY	No unread message. (Buffer is empty.)
	CAN_ERR_OVERWRITE	Overwrite has occurred.
e_can_txb_result	CAN_TXB_TRANSMITTING	Transmitting, or no transmit request. Note.
	CAN_TXB_ABORT_OVER	Transmit abort completed. Note.
	CAN_TXB_END	Transmit completed, no transmit abort request. Note.
	CAN_TXB_END_WITH_ABORT	Transmit ended, with transmit abort request. Note.
	CAN_TXB_ARG_ERROR	Argument error. Note.

Note: R_CAN_GetTXMBResult only.

3. API Functions

3.1 Summary

This module has the following API functions.

Function name	Description
R_CAN_Create	Initializes CAN peripheral.
R_CAN_SetConfig	Initializes the CAN module according to the configuration.
R_CAN_AddRxRule	Adds a receive rule of the CAN module.
R_CAN_StartComm	Operates the CAN module and transition channel 0 to communicable mode.
R_CAN_StopComm	Transitions channel 0 to reset mode and stops the CAN module.
R_CAN_Sleep	Transitions the CAN module to global sleep mode.
R_CAN_SendByTXMB	Sends data from the TX Message Buffer on channel 0.
R_CAN_AbortTXMB	Aborts sending from the TX Message Buffer on channel 0.
R_CAN_GetTXMBResult	Gets the transmission result from the transmission buffer on channel 0.
R_CAN_SendByCFIFO	Sends data from the Common FIFO on channel 0.
R_CAN_AbortCFIFO	Aborts sending from the Common FIFO on channel 0.
R_CAN_ReadTxHistory	Reads entry from TX history list.
R_CAN_ReadRXMB	Reads received data from the RX Message Buffer.
R_CAN_ReadRXMBInHandler	Reads received data from the RX Message Buffer. (In interrupt handler)
R_CAN_ReadRXFIFO	Reads received data from the receive FIFO.
R_CAN_ReadCFIFO	Reads received data from the Common FIFO on channel 0.
r_can_glb_rxfifo_isr	CAN global receive FIFO interrupt handler.
r_can_glb_rxmb_isr	CAN global receive message buffer interrupt handler.
r_can_glb_error_isr	CAN global error interrupt handler.
r_can_ch0_transmit_isr	CAN0 channel transmit interrupt handler.
r_can_ch0_cfifo_rx_isr	CAN0 common FIFO receive interrupt handler.
r_can_ch0_error_isr	CAN0 channel error interrupt handler.
r_can_ch0_wakeup_isr	CAN0 wakeup interrupt handler.
CAN_CFG_CALLBACK_XXXX	Callback from the interrupt handler.
R_CAN_ReadChStatus	Gets the status of channel 0.
R_CAN_ReadChBusErrFlag	Gets the bus error flag of channel 0.
R_CAN_GetTDCResult	Gets the Transceiver Delay Compensation (TDC) result of channel 0.
R_CAN_GetTSCounter	Gets timestamp counter value
R_CAN_GetVersion	Gets the version of this module

3.2 R_CAN_Create

Initializes CAN peripheral.

Format

void R_CAN_Create(void);

Parameters

None.

Return Values

None.

Properties

The prototype is declared in "r_rscanfd_rl78_if.h".

Description

Initialize peripheral functions to use the CAN module.

- Supply input clock to CAN. CAN0EN=1
- Disable the interrupt mask flag related to CAN.
  RCANGRVCMK, RCAN0ERMK, RCAN0WUPMK, RCAN0CFRMK, RCAN0TRMMK, RCANGRFRMK, RCANGERRMK
- Clear the interrupt request flag related to CAN
  RCANGRVCIF, RCAN0ERIF, RCAN0WUPIF, RCAN0CFRIF, RCAN0TRMIF, RCANGRFRIF, RCANGERRIF

Reentrant

Non-reentrant.

Example

R_CAN_Create();

Special Notes:

1. The interrupt priority flag is set with the function R_CAN_SetConfig.

3.3 R_CAN_SetConfig

Initializes the CAN module according to the configuration.

Format

e_can_err_t R_CAN_SetConfig(void);

Parameters

None.

Return Values

CAN_SUCCESS	Completed successfully.
CAN_ERR_INVALID_MODE	Global reset mode has been released.
CAN_ERR_ ILLEGAL_STS	Failure global mode or channel mode transition.

Properties

The prototype is declared in "r_rscanfd_rl78_if.h".

Description

Make initial settings for the CAN module.

With this function, the global mode transitions to the global reset mode, and the channel mode of the channel to be used transitions to the channel reset mode.

After the transition is completed, set the register of the CAN module.

- Channel mode selection: CAN-FD mode / FD only mode / Classical CAN only mode
- Baud rate setting
- Acceptance Filter setting
- Settings for each buffer (Depth, Data length, Interrupt enable / disable, etc.)

etc.

After setting the CAN register, makes initial setting of the interrupt register.

- Set the interrupt priority flag related to CAN.
  RCANGRVCPRx, RCAN0ERPRx, RCAN0WUPPRx, RCAN0CFRPRx, RCAN0TRMPRx, RCANGRFRPRx, RCANGERRPRx
  (x=0, 1)

Reentrant

Non-reentrant.

Example

```
e_can_err rtn;
rtn = R_CAN_SetConfig ();
```

Special Notes:

Call this function after calling the function R_CAN_Create.

3.4 R_CAN_AddRxRule

Adds a receive rule of the CAN module.

Format

```
e_can_err_t R_CAN_AddRxRule(const st_can_filter_t *     p_filter,
                            const can_storage_t      storage,
                            const can_rxbuf_t        rxbuf_idx,
                            const st_can_filter_opt_t * p_opt);
```

Parameters

p_filter

Address of the structure that stores the receive rule. For details, refer to "2.10.2 Structure, Union".

storage

Specifies the storage (a maximum of two) of the frame that matched with a receive rule.

CAN_STORE_RM	RX buffer
CAN_STORE_RF0	RXFIFO0
CAN_STORE_RF1	RXFIFO1
CAN_STORE_CF	Common FIFO
CAN_STORE_RM_AND_RF0	RX buffer and RXFIFO0
CAN_STORE_RM_AND_RF1	RX buffer and RXFIFO1
CAN_STORE_RM_AND_CF	RX buffer and Common FIFO
CAN_STORE_RF0_AND_RF1	RXFIFO0 and RXFIFO1
CAN_STORE_RF0_AND_CF	RXFIFO0 and Common FIFO
CAN_STORE_RF1_AND_CF	RXFIFO1 and Common FIFO

rxbuf_idx

Specifies the receive buffer number when stored in the receive buffer.

This is useful when storage is set to store in the receive buffer, such as CAN_STORE_RM..

p_opt

Address of the structure that stores the receive rule options.

For details, refer to "2.10.2 Structure, Union".

Set Null if option is not required

Return Values

CAN_SUCCESS	Completed successfully.
CAN_ERR_INVALID_ARG	Argument error
CAN_ERR_INVALID_MODE	Not in channel reset mode
CAN_ERR_BUF_FULL	The number of rules has reached the upper limit

Properties

The prototype is declared in "r_rscanfd_rl78_if.h".

Description

Add a receive rule for the CAN module.

The number of receiving rules after calling R_CAN_SetConfig function is 0, and the receive rules are added one by one by a normal call to this function. Up to 16 rules can be set.

Reentrant

Non-reentrant.

Example

```
e_can_err rtn;
st_can_filter_t filter;
st_can_filter_opt_t opt;
filter.IDE_TYPE = CAN_IDE_STD_FORMAT;
filter.RTR_TYPE = CAN_RTR_ANY_FRAME;
filter.ID = 0x700u;
filter.ID_MASK = 0x700u;
opt.DLC = 0u;
opt.LB  = 0u;
opt.LBL = 0xA5A5u;
opt.IFL = 2u;
rtn = R_CAN_AddRxRule(&filter, CAN_STORE_RM, rxbuf_idx, &opt);
```

Special Notes:

Call this function after calling the function R_CAN_SetConfig.

3.5 R_CAN_StartComm

Operates the CAN module and transition channel 0 to communicable mode.

Format

e_can_err_t R_CAN_StartComm(void);

Parameters

None.

Return Values

CAN_SUCCESS	Completed successfully.
CAN_ERR_INVALID_MODE	Invalid global mode or channel mode
CAN_ERR_ILLEGAL_STS	Failure global mode or channel mode transition.
CAN_ERR_WAITING	Wait to change global mode or channel mode

Properties

The prototype is declared in "r_rscanfd_rl78_if.h".

Description

Transitions the global mode to the global operation mode, and if successful, transitions the channel mode of channel 0 to the channel operation mode.

After confirming that the mode transition was successful, setting the following permission.

- RX FIFO buffer to use
- Common FIFO buffer to use

Setting the interrupt register enable before transitioning to the global operation mode. (Excluding INTRCAN0WUP.)

- Clear the interrupt request flag related to CAN.
  RCANGRVCIF, RCAN0ERIF, RCAN0CFRIF, RCAN0TRMIF, RCANGRFRIF, RCANGERRIF
- Enable the interrupt mask flag related to CAN.
  RCANGRVCMK, RCAN0ERMK, RCAN0CFRMK, RCAN0TRMMK, RCANGRFRMK, RCANGERRMK

Reentrant

Non-reentrant.

Example

```
e_can_err rtn;
rtn = CAN_ERR_WAITING;
while (CAN_ERR_WAITING == rtn)
{
    rtn = R_CAN_StartComm();
}

if (CAN_SUCCESS == rtn)
{
    while (0x0080u != R_CAN_GetChStatus()); /* Wait until communication is ready */
}
else
{
    /* error process */
}
```

Special Notes:

1. Call this function after calling the function R_CAN_SetConfig.
2. Setting to use of the RX FIFO buffer and Common FIFO buffer is performed only when the use is set in the configuration.
3. After the successful completion, make sure that the function R_CAN_GetChStatus is ready to communicate.

3.6 R_CAN_StopComm

Transitions channel 0 to reset mode and stops the CAN module.

Format

e_can_err_t R_CAN_StopComm(void);

Parameters

None.

Return Values

CAN_SUCCESS	Completed successfully.
CAN_ERR_WAITING	Wait to change global mode or channel mode

Properties

The prototype is declared in "r_rscanfd_rl78_if.h".

Description

Transitions the channel mode of channel 0 to the channel reset mode, and if successful, transitions the global mode to global reset mode.

By transitioning to the channel reset mode, CAN communication of the channel is stopped.

Before transitioning to channel reset mode, transition to channel halt mode and wait for the end of transmission / reception.

All interrupt registers related to CAN are disabled before channel mode transition.

- Disable the interrupt mask flag related to CAN.
  RCANGRVCMK, RCAN0ERMK, RCAN0WUPMK, RCAN0CFRMK, RCAN0TRMMK, RCANGRFRMK, RCANGERRMK
- Clear the interrupt request flag related to CAN.
  RCANGRVCIF, RCAN0ERIF, RCAN0WUPIF, RCAN0CFRIF, RCAN0TRMIF, RCANGRFRIF, RCANGERRIF

Reentrant

Non-reentrant.

Example

```
e_can_err rtn;
rtn = R_CAN_StopComm ();
```

Special Notes:

None.

3.7 R_CAN_Sleep

Transitions the CAN module to global sleep mode.

Format

e_can_err_t R_CAN_Sleep(const unsigned char wup);

Parameters

wup

Use / No use of CAN Wakeup. (Specify CAN_WUP_UNUSE to CAN_WUP_USE (0 to 1).)

Return Values

CAN_SUCCESS	Completed successfully.
CAN_ERR_INVALID_MODE	Invalid global mode or channel mode
CAN_ERR_ ILLEGAL_STS	Failure global mode or channel mode transition.

Properties

The prototype is declared in "r_rscanfd_rl78_if.h".

Description

Transition the CAN module to global sleep mode.

This mode stops the clock of the entire module and achieves low power consumption.

After transitioning to global sleep mode, the clock to the CAN module is stopped.

- Stop supplying the input clock to the CAN. CAN0EN = 0 (for CAN_WUP_UNUSE)
- Stop supplying the X1 clock to the CAN. CAN0MCKE = 0

Clock needs to supply to the CRXD0 pin by CAN0EN to detect CAN wakeup, the argument wup specifies whether to stop CAN0EN. If CAN_WUP_USE is specified, CAN0EN = 0 will not be set.

If INTRCAN0WUP is set to enable in the configuration, enable the interrupt register.

- Clear the interrupt request flag of INTRCAN0WUP. (RCAN0WUPIF)
- Enable the interrupt mask flag of INTRCAN0WUP. (RCAN0WUPMK)

Reentrant

Non-reentrant.

Example

```
e_can_err rtn;
rtn = R_CAN_Sleep(CAN_WUP_UNUSE);
```

Special Notes:

None.

3.8 R_CAN_SendByTXMB

Sends data from the TX Message Buffer on channel 0.

Format

e_can_err_t R_CAN_SendByTXMB(const can_txbuf_t txbuf_idx, const st_can_tx_frame_t * p_frame);

Parameters

txbuf_idx

TX Message Buffer number. (Specify CAN_TXBUF0 to CAN_TXBUF3 (0 to 3).)

p_frame

Address of the structure that stores the sending message. For details, refer to "2.10.2 Structure, Union".

Return Values

CAN_SUCCESS	Completed successfully.
CAN_ERR_BUF_BUSY	Status busy of TX Message Buffer.
CAN_ERR_INVALID_ARG	Argument error. (TX Message Buffer number is not 0 to 3.)

Properties

The prototype is declared in "r_rscanfd_rl78_if.h".

Description

Stores message data in the specified TX Message Buffer and issues a transmit request.

Before storing the data, the TX Message Buffer transmission result flag is cleared.

If the status of the TX Message Buffer is not 0, such as when there is already a transmit request or the TX Message Buffer transmission result flag fails to be cleared, CAN_ERR_BUF_BUSY is returned without performing transmitting.

Reentrant

Reentrant, for different TX Message Buffer numbers.

Example

```
s_tx_frame.Head.Bits.IDE   = CAN_IDE_STD_FORMAT;
s_tx_frame.Head.Bits.ID    = 0x7FEuL;
s_tx_frame.Head.Bits.RTR   = CAN_RTR_DATA_FRAME;
s_tx_frame.Head.Bits.FDCTR = CAN_FDCTR_CLASSICAL;
s_tx_frame.Head.Bits.DLC   = CAN_DLC_LEN8;
s_tx_frame.Head.Bits.THLEN   = CAN_THL_DISABLE;
s_tx_frame.Head.Bits.LBL  = 0u;
s_tx_frame.Head.Bits.IFL   = 0u;
for (i = 0u; i < CAN_DLC_LEN8; i++)  s_tx_frame.Data.DB[i] = 0x11u * (i + 1);
rtn = R_CAN_SendByTXMB(CAN_TXBUF0, &s_tx_frame);
```

Special Notes:

1. Call this function after the successful completion of calling the function R_CAN_StartComm.
2. Do not specify the TX Message Buffer number linked to the Common FIFO buffer.

3.9 R_CAN_AbortTXMB

Aborts sending from the TX Message Buffer on channel 0.

Format

e_can_err_t R_CAN_AbortTXMB(const can_txbuf_t txbuf_idx);

Parameters

txbuf_idx

TX Message Buffer number. (Specify CAN_TXBUF0 to CAN_TXBUF3 (0 to 3).)

Return Values

CAN_SUCCESS	Completed successfully.
CAN_ERR_INVALID_ARG	Argument error. (TX Message Buffer number is not 0 to 3.)

Properties

The prototype is declared in "r_rscanfd_rl78_if.h".

Description

Issue a transmit abort request to the specified TX Message Buffer.

Reentrant

Reentrant, for different TX Message Buffer numbers.

Example

```
e_can_err rtn;
rtn = R_CAN_AbortTXMB(CAN_TXBUF0);
```

Special Notes:

Do not specify the TX Message Buffer number linked to the Common FIFO buffer.

3.10 R_CAN_GetTXMBResult

Gets the transmission result from the transmission buffer on channel 0.

Format

e_can_txb_result_t R_CAN_GetTXMBResult(const can_txbuf_t txbuf_idx);

Parameters

txbuf_idx

TX Message Buffer number. (Specify CAN_TXBUF0 to CAN_TXBUF3 (0 to 3).)

Return Values

CAN_TXB_TRANSMITTING	Transmitting, or no transmit request.
CAN_TXB_ABORT_OVER	Transmit abort completed.
CAN_TXB_END	Transmit completed, no transmit abort request.
CAN_TXB_END_WITH_ABORT_REQ	Transmit ended, with transmit abort request.
CAN_ERR_ ARG_ERROR	Argument error. (TX Message Buffer number is not 0 to 3.)

Properties

The prototype is declared in "r_rscanfd_rl78_if.h".

Description

Reads the status register of the specified TX Message Buffer and returns the transmission result.

After reading the transmission result, the status register is cleared to clear the transmission result.

Reentrant

Reentrant, for different TX Message Buffer numbers.

Example

```
e_can_txb_result_t rtn;
rtn = R_CAN_GetTXMBResult(CAN_TXBUF0);
```

Special Notes:

By calling this function, the interrupt request flag (TMTRF [1:0] in TMSTSm) of CANi transmission completion / CANi transmission abort is cleared.

3.11 R_CAN_SendByCFIFO

Sends data from the Common FIFO on channel 0.

Format

e_can_err_t R_CAN_SendByCFIFO(const st_can_tx_frame_t * p_frame);

Parameters

p_frame

Address of the structure that stores the sending message. For details, refer to "2.10.2 Structure, Union".

Return Values

CAN_SUCCESS	Completed successfully.
CAN_ERR_BUF_FULL	Common FIFO buffer is full.
CAN_ERR_INVALID_ARG	Argument error. (Common FIFO buffer is not in TX mode.)

Properties

The prototype is declared in "r_rscanfd_rl78_if.h".

Description

Stores message data in the Common FIFO buffer.

Before storing the message data, check the status of the Common FIFO buffer, and if the Common FIFO buffer is full, return CAN_ERR_BUF_FULL without storing.

Reentrant

Non-reentrant.

Example

```
s_tx_frame.Head.Bits.IDE   = CAN_IDE_STD_FORMAT;
s_tx_frame.Head.Bits.ID    = 0x7FEuL;
s_tx_frame.Head.Bits.RTR   = CAN_RTR_DATA_FRAME;
s_tx_frame.Head.Bits.FDCTR = CAN_FDCTR_CLASSICAL;
s_tx_frame.Head.Bits.DLC   = CAN_DLC_LEN8;
s_tx_frame.Head.Bits.THLEN   = CAN_THL_DISABLE;
s_tx_frame.Head.Bits.LBL  = 0u;
s_tx_frame.Head.Bits.IFL   = 0u;
for (i = 0u; i < CAN_DLC_LEN8; i++)  s_tx_frame.Data.DB[i] = 0x11u * (i + 1);
rtn = R_CAN_SendByCFIFO(&s_tx_frame);
```

Special Notes:

1. Call this function after the successful completion of calling the function R_CAN_StartComm.
2. This function does not clear the interrupt request flag (CFTXIF in CFSTS) of RS-CANFD lite for the transmission completion interrupt of the Common FIFO buffer.
3. If the Common FIFO buffer is not set to be used in TX mode in the configuration, only argument error return processing is enabled when this function is compiled.

3.12 R_CAN_AbortCFIFO

Aborts sending from the Common FIFO on channel 0.

Format

e_can_err_t R_CAN_ AbortCFIFO(void);

Parameters

None.

Return Values

CAN_SUCCESS	Completed successfully.
CAN_ERR_BUF_BUSY	Common FIFO was not emptied.
CAN_ERR_INVALID_ARG	Argument error. (Common FIFO buffer is unused.)

Properties

The prototype is declared in "r_rscanfd_rl78_if.h".

Description

By disabling the Common FIFO once, the messages in the Common FIFO will be aborted.

After disabling the Common FIFO, make sure that the Common FIFO is empty, and then enable the Common FIFO.

Reentrant

Non-reentrant.

Example

```
e_can_err_t  rtn;
rtn = R_CAN_AbortCFIFO();
```

Special Notes:

1. This function waits until the Common FIFO buffer is empty. If the message in the Common FIFO buffer is being transmitted or is determined to be the next transmission, it is emptied after transmission completion, CAN bus error detection, or arbitration lost. Therefore, waits may time out and CAN_ERR_BUF_BUSY may return before one frame's transmission is complete.
2. If there is no Common FIFO buffer set for use in the configuration, only argument error return processing is enabled when this function is compiled.

3.13 R_CAN_ReadTxHistory

Reads entry from the TX history list on channel 0.

Format

e_can_err_t R_CAN_ReadTxHistory(st_can_txhist_t *p_entry);

Parameters

p_entry

Address of the structure that stores the TX history entry. For details, refer to "2.10.2 Structure, Union".

Return Values

CAN_SUCCESS	Completed successfully.
CAN_SUCCESS_WITH_LOST	Complete successfully, but entry was lost.
CAN_ERR_BUF_EMPTY	No unread message. (Buffer is empty.)
CAN_ERR_INVALID_ARG	Argument error. (p_entryr is NULL.)

Properties

The prototype is declared in "r_rscanfd_rl78_if.h".

Description

Reads the received data stored in the specified TX history list for one entry.

Check the status of the TX history list and clear the overflow flag if overflow has occurred. At this time, the entry is read and CAN_SUCCESS_WITH_LOST is returned.

CAN_ERR_BUF_EMPTY is returned if the RX FIFO buffer is empty.

Reentrant

Non-reentrant.

Example

```
e_can_err_t       rtn;
st_can_txhist_t entry;

rtn = R_CAN_ReadTxHistory(&entry);
if (rtn == CAN_SUCCESS)
{
    if (entry.txbuf_idx == CAN_TXBUF_NOT)
    {
        /* process for TX history of CFIFO */
    }
    else
    {
        /* process for TX history of TXMB */
    }
}
```

Special Notes:

1. This function does not clear the interrupt request flag (THLIF in THLSTS) of RS-CANFD lite.
2. Calling this function clears the THLIF in THLSTS.
3. If the TX history list is not set to be used in the configuration, CAN_ERR_BUF_EMPTY is always returned.

3.14 R_CAN_ReadRXMB

R_CAN_ReadRXMB Reads received data from the RX Message Buffer.

R_CAN_ReadRXMBInHandler Reads received data from the RX Message Buffer. (In interrupt handler)

Format

```
e_can_err_t R_CAN_ReadRXMB(const can_rxbuf_t rxbuf_idx,
                           st_can_rx_frame_t * p_frame,
                           can_length_t * p_length);
e_can_err_t R_CAN_ReadRXMBInHandler(const can_rxbuf_t rxbuf_idx,
                                    st_can_rx_frame_t * p_frame,
                                    can_length_t * p_length);
```

Parameters

rxbuf_idx

RX Message Buffer number. (Specify CAN_RXBUF0 to CAN_RXBUF15 (0 to 15).)

p_frame

Address of the structure that stores the receiving message. For details, refer to "2.10.2 Structure, Union".

p_length

Receiving data length.

Return Values

CAN_SUCCESS	Completed successfully.
CAN_ERR_BUF_EMPTY	No unread message. (Buffer is empty.)
CAN_ERR_BUF_BUSY	Status error. (Failed to clear the reception completion flag.)
CAN_ERR_OVERWRITE	Overwrite has occurred during reading.
CAN_ERR_INVALID_ARG	Argument error. (RX Message Buffer number is not 0 to [Number of use - 1].)

Properties

The prototype is declared in "r_rscanfd_rl78_if.h".

Description

Reads the received message data stored in the specified RX Message Buffer.

The function R_CAN_ReadRXMBInHandler checks the reception completion flag before reading the message, and if the flag is 1, clears it. If the clearing of reception completion flag has failed, returns CAN_ERR_BUF_BUSY. If the buffer is overwritten by the next received message in the middle of message reading, returns CAN_ERR_OVERWRITE. If the flag is 0, returns CAN_ERR_BUF_EMPTY.

The function R_CAN_ReadRXMBInHandler assumes that the reception completion flag has been cleared before the call, and does not check or clear the reception completion flag before reading the message. Because reading on the assumption that message is received and stored, CAN_ERR_BUF_EMPTY is not returned.

Reentrant

Non-reentrant.

Example

```
e_can_err_t       rtn;
st_can_rx_frame_t rx_frame;
can_length_t      length;

rtn = R_CAN_ReadRXMB(CAN_RXBUF0, &rx_frame, &length);
if (rtn == CAN_SUCCESS)
{
    /* process for receive message */
}
```

Special Notes:

If the number of RX Message Buffers used is set to 0 in the configuration, only argument error return processing is enabled when this function is compiled.

3.15 R_CAN_ReadRXFIFO

Reads received data from the receive FIFO.

Format

```
e_can_err_t R_CAN_ReadRXFIFO(const can_rxfifo_t rxfifo_idx,
                             st_can_rx_frame_t * p_frame,
                             can_length_t * p_length);
```

Parameters

rxfifo_idx

RX FIFO buffer number. (Specify CAN_RXFIFO0 to CAN_RXFIFO1 (0 to 1).)

p_frame

Address of the structure that stores the receiving message. For details, refer to "2.10.2 Structure, Union".

p_length

Receiving data length.

Return Values

CAN_SUCCESS	Completed successfully.
CAN_SUCCESS_WITH_LOST	Complete successfully, but message was lost.
CAN_ERR_BUF_EMPTY	No unread message. (Buffer is empty.)
CAN_ERR_INVALID_ARG	Argument error. (RX FIFO buffer number is out of range, or specified RX FIFO buffer is unused.)

Properties

The prototype is declared in "r_rscanfd_rl78_if.h".

Description

Reads the received data stored in the specified RX FIFO buffer for one message.

Check the status of the RX FIFO buffer and clear the message lost flag if message lost has occurred. At this time, the message is read and CAN_SUCCESS_WITH_LOST is returned.

CAN_ERR_BUF_EMPTY is returned if the RX FIFO buffer is empty.

Reentrant

Reentrant, for different RX FIFO buffer numbers.

Example

```
e_can_err_t       rtn;
st_can_rx_frame_t rx_frame;
can_length_t      length;

rtn = R_CAN_ReadRXMB(CAN_RXFIFO0, &rx_frame, &length);
```

```
if (rtn == CAN_SUCCESS)
{
    /* process for receive message */
}
```

Special Notes:

1. This function does not clear the interrupt request flag (RFIF in RFSTSk) of RS-CANFD lite for the receiption completion interrupt of the RX FIFO buffer.
2. Calling this function clears the RFMLT in RFSTSk. If there is no other RX FIFO buffer or Common FIFO buffer in which the FIFO message is lost, the interrupt request flag of the FIFO message lost interrupt (MES in GERFLL) becomes 0.
3. If there is no RX FIFO buffer set for use in the configuration, only argument error return processing is enabled when this function is compiled.

3.16 R_CAN_ReadCFIFO

Reads received data from the Common FIFO on channel 0.

Format

e_can_err_t R_CAN_ReadCFIFO(st_can_rx_frame_t * p_frame, can_length_t * p_length);

Parameters

p_frame

Address of the structure that stores the receiving message. For details, refer to "2.10.2 Structure, Union".

p_length

Receiving data length.

Return Values

CAN_SUCCESS	Completed successfully.
CAN_SUCCESS_WITH_LOST	Complete successfully, but message was lost.
CAN_ERR_BUF_EMPTY	No unread message. (Buffer is empty.)
CAN_ERR_INVALID_ARG	Argument error. (Common FIFO buffer is not in RX mode.)

Properties

The prototype is declared in "r_rscanfd_rl78_if.h".

Description

When the Common FIFO buffer is in RX mode, reads the received data stored in the specified Common FIFO buffer for one message.

Check the status of the Common FIFO buffer and clear the message lost flag if message lost has occurred. At this time, the message is read and CAN_SUCCESS_WITH_LOST is returned.

CAN_ERR_BUF_EMPTY is returned if the RX FIFO buffer is empty.

Reentrant

Non-reentrant.

Example

```
e_can_err_t       rtn;
st_can_rx_frame_t rx_frame;
can_length_t      length;

rtn = R_CAN_ReadCFIFO(&rx_frame, &length);
if (rtn == CAN_SUCCESS)
{
    /* process for receive message */
}
```

Special Notes:

4. This function does not clear the interrupt request flag (CFIF in CFSTSm) of RS-CANFD lite for the reception completion interrupt of the Common FIFO buffer.
5. Calling this function clears the CFMLT in CFSTSm. If there is no other RX FIFO buffer or Common FIFO buffer in which the FIFO message is lost, the interrupt request flag of the FIFO message lost interrupt (MES in GERFLL) becomes 0.
6. If the Common FIFO buffer is not set to be used in RX mode in the configuration, only argument error return processing is enabled when this function is compiled.

3.17 r_can_glb_xxxx_isr

r_can_glb_rxfifo_isr CAN global receive FIFO interrupt handler.

r_can_glb_rxmb_isr CAN global receive message buffer interrupt handler.

r_can_glb_error_isr CAN global error interrupt handler.

Format

R_BSP_PRAGMA_STATIC_INTERRUPT(r_can_glb_rxfifo_isr, CAN_INTRCANGRFR_VECT);

R_BSP_PRAGMA_STATIC_INTERRUPT(r_can_glb_rxmb_isr, CAN_INTRCANGRVC_VECT);

R_BSP_PRAGMA_STATIC_INTERRUPT(r_can_glb_error_isr, CAN_INTRCANGERR_VECT);

Parameters

None.

Return Values

None.

Properties

The prototype is declared in "r_rscanfd_rl78.c".

Description

Interrupt handler for CAN global interrupts.

Clears the interrupt request flag of the CAN module that causes the interrupt, and calls the user callback function. At this time, those for which interrupt enable is not set in the configuration are excluded from clearing.

Example: RMIEC configuration value is 1, RMND=3 -> Clears the interrupt request flag -> RMND=2

Clear only bit0. Bit1 is not cleared because interrupt enable is not set.

The RL78/F2x CAN global interrupt does not generate the next interrupt until the interrupt request is cleared. Also, the OR values of multiple interrupt sources are assigned to one interrupt source. Therefore, this function repeatedly clears until it can be confirmed that all interrupt sources have become 0.

The interrupt request flag to be cleared and the interrupt enable (configuration value) to be referenced are as follows.

r_can_glb_rxfifo_isr

Interrupt request flag: RFIF in RFSTSk. (The target of reading is RFISTS.)

Interrupt enable: RFIE in RFCCk.

r_can_glb_rxmb_isr

Interrupt request flag: RMND.

Interrupt enable: RMIEC.

r_can_glb_error_isr

Of the following, the bit that is the interrupt enable setting in the GCTRL.

Interrupt request flag:

DEF in GERFL.

Bit that sets the MSE in GERFL to 1. (RFMLT in RFSTSk, CFMLT in CFSTS)

CMPOF in GERFL.

Note: Because transmission history is not supported, THLES is not supported.
Interrupt enable: DEIE, MEIE, CMPOFIE in GCTR

Reentrant

Non-reentrant.

Example

This function is not called by the user.

Special Notes:

1. All that is done in the interrupt handler is to clear the flag.
2. To read the RX FIFO and RX Message Buffer, it is necessary to call the functions R_CAN_ReadRXFIFO and R_CAN_ReadRXMBInHandler in the user callback.

3.18 r_can_ch0_xxxx_isr

r_can_ch0_transmit_isr	CAN0 channel transmit interrupt handler.
r_can_ch0_cfifo_rx_isr	CAN0 common FIFO receive interrupt handler.
r_can_ch0_error_isr	CAN0 channel error interrupt handler.
r_can_ch0_wakeup_isr	CAN0 wakeup interrupt handler.

Format

```
R_BSP_PRAGMA_STATIC_INTERRUPT(r_can_ch0_transmit_isr, CAN_INTRCAN0TRM_VECT);
R_BSP_PRAGMA_STATIC_INTERRUPT(r_can_ch0_cfifo_rx_isr, CAN_INTRCAN0CFR_VECT);
R_BSP_PRAGMA_STATIC_INTERRUPT(r_can_ch0_error_isr, CAN_INTRCAN0ERR_VECT);
R_BSP_PRAGMA_STATIC_INTERRUPT(r_can_ch0_wakeup_isr, CAN_INTRCAN0WUP_VECT);
```

Parameters

None.

Return Values

None.

Properties

The prototype is declared in "r_rscanfd_rl78.c".

Description

Interrupt handler for CAN channel interrupts.

Clears the interrupt request flag of the CAN module that causes the interrupt, and calls the user callback function. At this time, those for which interrupt enable is not set in the configuration are excluded from clearing. (INTRCAN0WUP is not have a target interrupt request flag, so there is no clearing.)

The RL78/F2x CAN channel interrupt does not generate the next interrupt until the interrupt request is cleared. Also, INTRCAN0TRM and INTRCAN0ERR are assigned OR values of multiple interrupt sources to one interrupt source. Therefore, the INTRCAN0TRM and INTRCAN0ERR interrupt handlers repeatedly clears until it can be confirmed that all interrupt sources have become 0.

The interrupt request flag to be cleared and the interrupt enable (configuration value) to be referenced are as follows.

r_can_ch0_transmit_isr

Interrupt request flag:

TMTRF in TMSTS. (The target of reading is TMTCSTS, TMTASTS.)

CFTXIF in CFSTS. (When TFIE = 1.)

Note: Because transmission history is not supported, THLIF is not supported.

Interrupt enable:

TMIEC, TAIE in C0CTRH.

CFTXIE in CFCC.

r_can_ch0_cfifo_rx_isr

Interrupt request flag: CFRFIF in CFSTS.

Interrupt enable: Does not refer. (Because the target request flag is CFRFIF only.)

r_can_ch0_error_isr

Interrupt request flag: Lower 8 bits of C0ERFL, TDCVF in C0FDSTS

(Does not support EOCO and SOCO bits in C0FDSTS.)

Interrupt enable: Upper 8 bits of C0CTRL, TDCVFIE in C0CTRH.

r_can_ch0_wakeup_isr

None.

Reentrant

Non-reentrant.

Example

This function is not called by the user.

Special Notes:

None.

3.19 CAN_CFG_CALLBACK_XXXX

Callback from the interrupt handler. The user_callback function name is arbitrary.

Format

```
#define CAN_CFG_CALLBACK_GLB_RXFIFO        user_callback
#define CAN_CFG_CALLBACK_GLB_RXMB          user_callback
#define CAN_CFG_CALLBACK_GLB_ERROR         user_callback
#define CAN_CFG_CALLBACK_CH0_TRANSMIT      user_callback
#define CAN_CFG_CALLBACK_CH0_CFIFO_RX      user_callback
#define CAN_CFG_CALLBACK_CH0_ERROR         user_callback
#define CAN_CFG_CALLBACK_CH0_WAKEUP        user_callback
void CAN_CFG_CALLBACK_XXXX(uint16_t arg);
```

Parameters

arg

Interrupt sources flag.

Return Values

None.

Properties

The prototype is declared in "r_rscanfd_rl78_config.h".

Description

These are user callback functions called from the interrupt handler of CAN global interrupt and CAN channel interrupt. Specify any function name in the configuration.

The caller of each callback and the interrupt source flag passed are as follows.

CAN_CFG_CALLBACK_GLB_RXFIFO

Called from r_can_glb_rxfifo_isr.

Argument - Bit0: Received with RX FIFO 0, Bit1: Received with RX FIFO 1, Bit2 to 15: Fixed to 0.

CAN_CFG_CALLBACK_GLB_RXMB

Called from r_can_glb_rxmb_isr.

Argument - Bit0: Received with RX Buffer 0, Bit1: Received with RX Buffer 1, to
Bit15: Received with RX Buffer 15.

CAN_CFG_CALLBACK_GLB_ERROR

Called from r_can_glb_error_isr.

Argument - Bit0: DLC error, Bit1: Message lost, Bit2: TX history overflow,
Bit3: Payload overflow,
Bit4 to 5, 8: Details of message lost,
Bit4: Message lost with RX FIFO 0, Bit5: Message lost with RX FIFO 1,
Bit8: Message lost with Common FIFO.

Others: Fixed to 0.

CAN_CFG_CALLBACK_CH0_TRANSMIT

Called from r_can_ch0_transmit_isr.

Argument - Bit0 to 3: Transmission completed with TX Message Buffer 0 to 3,

(Example: If bit0 = 1, transmission completed with TX Message Buffer 0)

Bit4 to 7: Transmission abort completed with TX Message Buffer 0 to 3,

(Example: If bit4 = 1, transmission abort completed with TX Message Buffer 0)

Bit8: Transmission interrupt with Common FIFO,

Bit12: TX history list interrupt,

Bit9 to 11, Bit13 to 15: Fixed to 0.

CAN_CFG_CALLBACK_CH0_CFIFO_RX

Called from r_can_ch0_cfifo_rx_isr.

Argument - Bit0: Received with Common FIFO, Bit1 to 15: Fixed to 0.

CAN_CFG_CALLBACK_CH0_ERROR

Called from r_can_ch0_error_isr.

Argument - Bit0: Bus Error detected, Bit1: Error Warning detected, Bit2: Error Passive detected,

Bit3: Bus-Off Entry detected, Bit4: Bus-Off Recovery detected, Bit5: Overload detected,

Bit6: Bus Lock detected, Bit7: Arbitration Lost detected,

Bit8: TDC violation detected, Bit9 to 15: Fixed to 0.

CAN_CFG_CALLBACK_CH0_WAKEUP

Called from r_can_ch0_wakeup_isr.

Argument - Bit0: Wakeup, Bit1 to 15: Fixed to 0.

Reentrant

Non-reentrant.

Example

This function is not called by the user.

Special Notes:

None.

3.20 R_CAN_GetChStatus

Gets the status of channel 0.

Format

uint16_t R_CAN_GetChStatus(void);

Parameters

None.

Return Values

Channel status (lower 9 bits).

Bit8: The bit is 1 if the ESI bit was sampled recessive for a reception CAN message

Bit7: The bit is 1 if communication is ready.

Bit6: The bit is 1 in reception, and 0 in bus idle, transmission or bus off state.

Bit5: The bit is 1 in transmission or bus off state, and 0 in bus idle or reception.

Bit4: The bit is 1 if in bus off state.

Bit3: The bit is 1 if in error passive state.

Bit2: The bit is 1 if in channel sleep mode.

Bit1: The bit is 1 if in channel halt mode.

Bit0: The bit is 1 if in channel reset mode.

Properties

The prototype is declared in "r_rscanfd_rl78_if.h".

Description

Reads the status of the channel and returns it.

Clears the bit8 of the status with register clearing after reading if bit8 is 1.

.

Reentrant

Non-reentrant.

Example

```
uint16_t sts;
sts = R_CAN_ReadChStatus();
```

Special Notes:

None.

3.21 R_CAN_GetChBusErrFlag

Gets the bus error flags of channel 0.

Format

uint16_t R_CAN_GetChStatus(void);

Parameters

None.

Return Values

Channel bus error flags (lower 7 bits).

Bit6: The bit is 1 if the acknowledge delimiter error detected.

Bit5: The bit is 1 if the bit 0 error delimiter error detected.

Bit4: The bit is 1 if the bit 1 error detected.

Bit3: The bit is 1 if the CRC error detected.

Bit2: The bit is 1 if the acknowledge error detected.

Bit1: The bit is 1 if the form error detected.

Bit0: The bit is 1 if the stuff error detected.

Properties

The prototype is declared in "r_rscanfd_rl78_if.h".

Description

Reads the bus error flags of the channel and returns it.

Clears the bits of the bus error flags after reading if bits are 1.

.

Reentrant

Non-reentrant.

Example

```
uint8_t errflag;
errflag = R_CAN_ReadChBusErrFlag();
```

Special Notes:

None.

3.22 R_CAN_GetTDCResult

Gets the Transceiver Delay Compensation (TDC) result of channel 0.

Format

uint8_t R_CAN_GetTDCResult(void);

Parameters

None.

Return Values

Transceiver Delay Compensation result

Properties

The prototype is declared in "r_rscanfd_rl78_if.h".

Description

Reads the Transceiver Delay Compensation result of the channel and returns it.

.

Reentrant

Non-reentrant.

Example

```
uint8_t result;
result = R_CAN_GetTDCResult();
```

Special Notes:

None.

3.23 R_CAN_GetTSCounter

Gets the timestamp counter value.

Format

uint16_t R_CAN_GetTSCounter(void);

Parameters

None.

Return Values

Timestamp counter value

Properties

The prototype is declared in "r_rscanfd_rl78_if.h".

Description

Reads the current value of the timestamp counter and returns it.

.

Reentrant

Non-reentrant.

Example

```
uint8_t value;
value = R_CAN_ GetTSCounter();
```

Special Notes:

None.

3.24 R_CAN_GetVersion

Gets the version of this module.

Format

uint16_t R_CAN_GetVersion(void);

Parameters

None.

Return Values

Version number (upper 8bits: Major version, lower 8bits: Minor version)

Properties

The prototype is declared in "r_rscanfd_rl78_if.h".

Description

Reads the version of this module and returns it.

.

Reentrant

Non-reentrant.

Example

```
uint16_t result;
result = R_CAN_GetVersion();
```

Special Notes:

None.

4. Appendix

4.1 Confirmed Operating Environment

The environment in which the operation of the module has been confirmed is shown below.

Table 4.1 Confirmed Operating Environment (Rev. 1.00)

Item	Description
Integrated development environment	Renesas Electronics CS+ for CC V8.07.00 IAR Systems IAR Embedded Workbench for Renesas RL78 4.21.3
C compiler	Renesas Electronics C/C++ compiler for R78 Family V.1.11.0 IAR Systems IAR C/C++ Compiler for Renesas RL78 4.21.3.2447
Module revision	Rev.1.00
Board used	RL78/F24 Target Board (Product type: RTK7F124FPC01000BJ)

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Apr.20.22	-	1st edition issued

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. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time 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 time 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 time 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 time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. 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 the 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.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is 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. Additionally, 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.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of 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.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.