Embedded Systems

CS 397

TRIMESTER 3, AY 2021/22

Controller Area Network (CAN) (2/3)

[Ref 03] RM0410, Reference Manual STM32F76xxx, Rev 4, Mar2018, ch 40, pp. 1531 – 1576.

Dr. LIAW Hwee Choo

Department of Electrical and Computer Engineering

DigiPen Institute of Technology Singapore

HweeChoo.Liaw@DigiPen.edu

Contents

RQCP = request completed
TXOK = transmission OK
ALST = arbitration lost
TERR = transmission error

The bxCAN Transmission Handling

- To transmit a message, the application must select one **empty** transmit mailbox, set up the identifier, the data length code/control (DLC) and the data before requesting the transmission by setting the TXRQ (transmit mailbox request) bit in the CAN TX Mailbox Identifier (CAN_TIxR, x=0..2) register.

- Once the mailbox **has left empty state**, the software no longer has "write" access to that mailbox registers. After the TXRQ bit has been set, the mailbox enters **pending** state and waits to become the highest priority mailbox. Once the mailbox has the highest priority it will be **scheduled** for transmission. The transmission of message of the scheduled mailbox will start (enter **transmit** state) when the CAN bus becomes idle.

- Once the mailbox successfully transmitted, it becomes **empty** again. The hardware indicates a successful transmission by setting the RQCPx and TXOKx bits in the CAN Transmit Status Register (CAN_TSR). If the transmission fails, the cause is indicated by the ALSTx bit in the CAN_TSR register in case of an Arbitration Lost, and/or the TERRx bit, in case of transmission error detection.

CAN transmit status register (CAN_TSR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LOW2	LOW1	LOW0	TME2	TME1	TME0	CODE[1:0]		ABRQ2	Res.	Res.	Res.	TERR2	ALST2	TXOK2	RQCP2
r	r	r	r	r	r	r	r	rs				rc_w1	rc_w1	rc_w1	rc_w1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABRQ1	Res.	Res.	Res.	TERR1	ALST1	TXOK1	RQCP1	ABRQ0	Res.	Res.	Res.	TERR0	ALST0	TXOK0	RQCP0
rs				rc_w1	rc_w1	rc_w1	rc_w1	rs				rc_w1	rc_w1	rc_w1	rc_w1

CAN TX mailbox identifier register (CAN_TIxR) (x = 0..2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
STID[10:0]/EXID[28:18]											EXID[17:13]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXID[12:0]													IDE	RTR	TXRQ
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

IDE: identifier extension (0=std/1=ext) , RTR: remote transmission request (0=data/1=remote)

CAN mailbox data length control and time stamp register (CAN_TDTxR) (x = 0..2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TIME[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DLC[3:0]			
												rw	rw	rw	rw

CAN mailbox data high register (CAN_TDHxR) (x = 0..2)

CAN mailbox data low register (CAN_TDLxR) (x = 0..2)

The bxCAN Transmission Handling – Transmit Priority

By Identifier (CAN_MCR, TXFP=0)

TXFP = Transmit FIFO Priority

- When more than one transmit mailbox is pending, the transmission order is given by the identifier of the message in the mailbox.
- The message with the lowest identifier value has the highest priority. If the identifier values are equal, the lower mailbox number will be scheduled first.

By Transmit Request Order (CAN_MCR, TXFP=1)

- In this mode the priority order is given by the transmit request order. This mode is very useful for segmented transmission.

Controller Area Network

CAN master control register (CAN_MCR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DBF
															rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESET	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TTCM	ABOM	AWUM	NART	RFLM	TXFP	SLEEP	INRQ
rs								rw	rw	rw	rw	rw	rw	rw	rw

DBF: Debug freeze

RESET: bxCAN software master reset

TTCM: Time triggered communication mode

ABOM: Automatic bus-off management

AWUM: Automatic wakeup mode

NART: No automatic retransmission

RFLM: Receive FIFO locked mode

TXFP: Transmit FIFO priority

SLEEP: Sleep mode request

INRQ: Initialization request

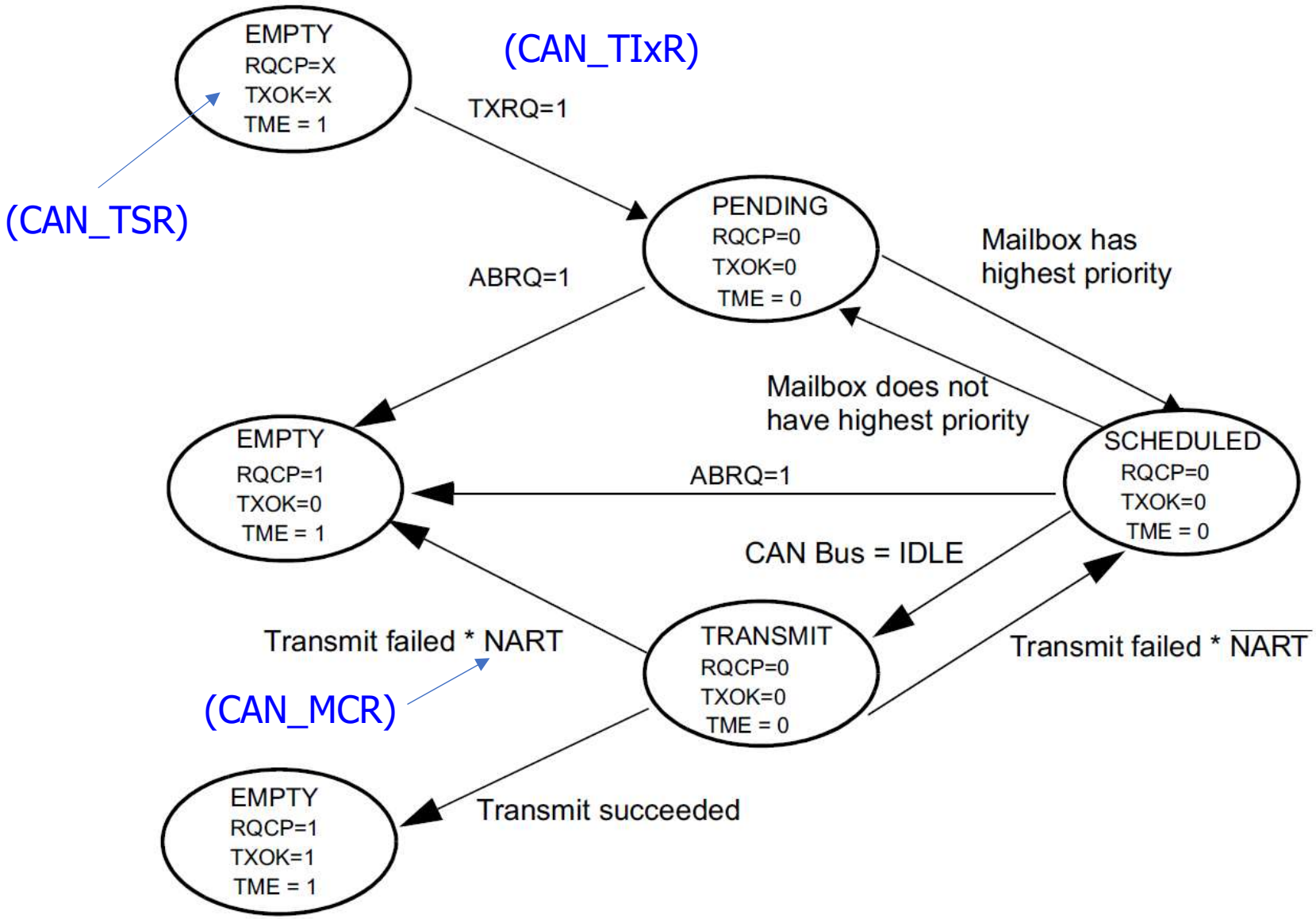
The bxCAN Transmission Handling – Abort

- A transmission request can be aborted by the user setting the ABRQx (abort request) bit in the CAN Transmit Status Register (CAN_TSR).
- In **pending** or **scheduled** state, the mailbox is aborted immediately.
- An abort request while the mailbox is in **transmit** state can have two results.
  - If the mailbox is transmitted successfully the mailbox becomes **empty** with the TXOK bit set in the CAN_TSR register.
  - If the transmission fails, the mailbox becomes **scheduled,** the transmission is aborted and becomes **empty** with TXOK cleared.
- In all cases the mailbox will become **empty** again at least at the end of the current transmission.

The bxCAN Transmission Handling – Non Automatic Retransmission Mode

- This mode is implemented according to the Time Triggered Communication option of the CAN standard.
- To configure the hardware in this mode the NART (no automatic retransmission) bit in the CAN_MCR register must be set.
- In this mode, each transmission is started only once.
- If the first attempt fails, due to an arbitration loss or an error, the hardware will not automatically restart the message transmission.
- At the end of the first transmission attempt, the hardware considers the request as completed and sets the RQCP (request completed) bit in the CAN_TSR register.
- The result of the transmission is indicated in the CAN_TSR register by the TXOK (transmission OK), ALST (arbitration error) and TERR (transmission error) bits.

The bxCAN Transmit Mailbox States



NART = no automatic retransmission

The bxCAN Time Triggered Communication Mode

- In this mode, the internal counter of the CAN hardware is activated and used to generate the Time Stamp value for storing in the

(1) CAN Receive FIFO Mailbox Data Length Control and Time Stamp register (CAN_RDTxR), &

(2) CAN Mailbox Data Length Control and Time Stamp register (CAN_TDTyR), i.e.,

(CAN_RDTxR, x=0,1) and (CAN_TDTyR, y=0..2), respectively, for Rx and Tx mailboxes.

- The internal counter is incremented by each CAN bit time (refer to Bit Timing).
- The internal counter is captured on the sample point of the Start Of Frame bit in both reception and transmission.
- Note that the above is enabled by setting the TTCM (time triggered communication mode) in CAN_MCR.

The bxCAN Reception Handling

- For the reception of CAN messages, 2 x three mailboxes organized as a FIFO are provided.
- The FIFO is managed completely by hardware. The messages stored in the FIFO through the FIFO output mailbox.

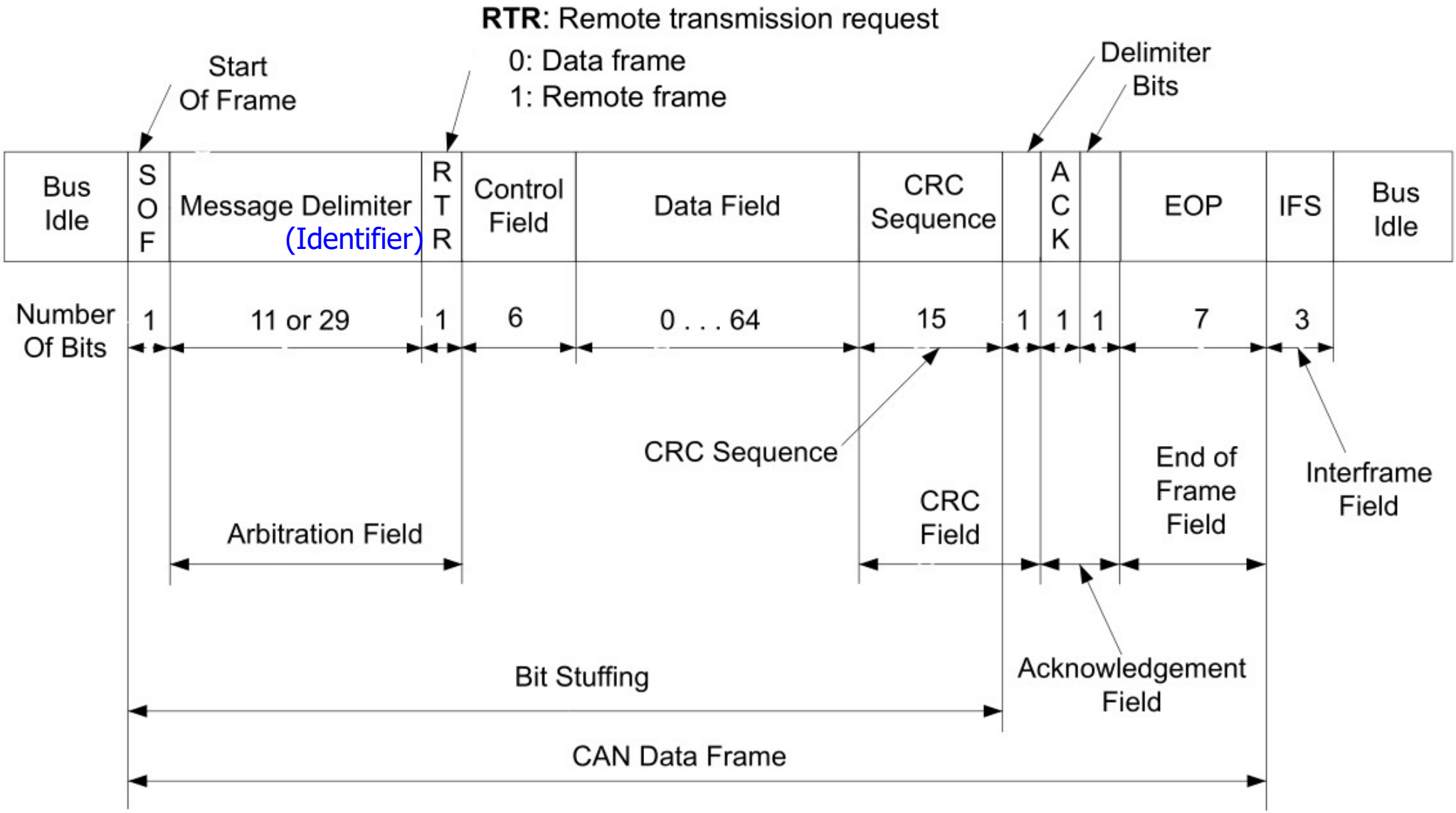
CAN receive FIFO mailbox data low register (CAN_RDLxR) (x = 0,1)

CAN receive FIFO mailbox data high register (CAN_RDHxR) (x = 0,1)

Valid message

A received message is considered as valid **when** it has been received correctly according to the CAN protocol (no error until the last but one bit of the EOF field) **and** it passed through the identifier filtering successfully, see Identifier filtering.

CAN Data/Remote Frame



The bxCAN Reception Handling – FIFO Management

- Starting from the **empty** state, the first valid message received is stored in the FIFO which becomes **pending_1**.
- The hardware sets the **FMPx**[1:0] (**FIFO x message pending**, x=0,1) **bits** in the **CAN Receive FIFO x** (CAN_RFxR, x=0,1) **register** to the value **01b**.
- The message is available in the FIFO output mailbox.
- The software reads the mailbox content and releases it by setting the **RFOMx** (**release FIFO x output mailbox**) **bit** in the CAN_RFxR register.
- The FIFO becomes **empty** again. FMPx[1:0] = **00b**.
- If a new valid message has been received in the meantime, the FIFO stays in **pending_1** state and the new message is available in the output mailbox.

Note: The FMPx[1:0] bits indicate how many messages are pending in the receive FIFO. FMPx is increased each time the hardware stores a new message into the FIFO. FMPx is decreased each time the software releases the output mailbox by setting the RFOMx bit.

Controller Area Network

CAN receive FIFO 0 register (CAN_RF0R)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RFOM0	FOVR0	FULL0	Res.	FMP0[1:0]	
										rs	rc_w1	rc_w1		r	r

RFOM0: Release FIFO 0 output mailbox

FOVR0: FIFO 0 overrun

FULL0: FIFO 0 full

FMP0[1:0]: FIFO 0 message pending

CAN receive FIFO 1 register (CAN_RF1R)

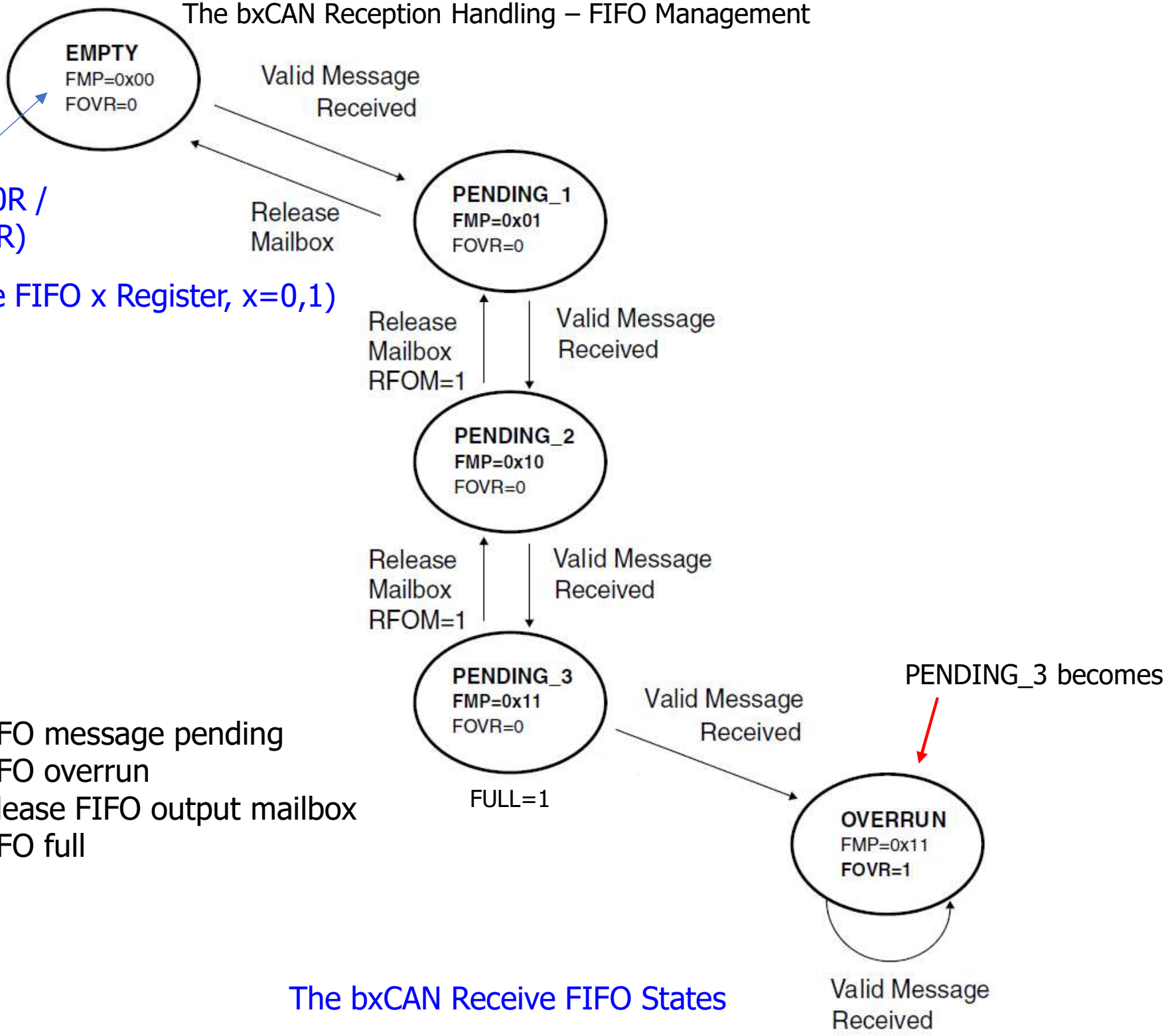
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RFOM1	FOVR1	FULL1	Res.	FMP1[1:0]	
										rs	rc_w1	rc_w1		r	r

The bxCAN Reception Handling – FIFO Management

- If the application does not release the mailbox, the next valid message will be stored in the FIFO which enters **pending_2** state (FMPx[1:0] = **10b**).
- The storage process is repeated for the next valid message putting the FIFO into **pending_3** state (FMPx[1:0] = **11b**).
- At this point, the software must read the data and release the output mailbox by setting the RFOMx bit, so that a mailbox is free to store the next valid message.
- Otherwise, the next valid message received will cause a loss of message.

The bxCAN Reception Handling – FIFO Management



(CAN_RF0R / CAN_RF1R)

(CAN Receive FIFO x Register, x=0,1)

FMP : FIFO message pending
FOVR : FIFO overrun
RFOM : release FIFO output mailbox
FULL : FIFO full

The bxCAN Receive FIFO States

The bxCAN Reception Handling – FIFO Overrun

- Once the FIFO is in **pending_3** state (i.e., the three mailboxes are full) the next valid message reception will lead to an **overrun** and a message will be lost.

- The hardware signals the overrun condition by setting the FOVRx (FIFO x overrun) bit in the CAN_RFxR register. Which message is lost depends on the configuration of the FIFO:

  - If the FIFO lock function is disabled [RFLM (receive FIFO locked mode) bit in the CAN_MCR register cleared], the last message stored in the FIFO will be overwritten by the new incoming message. In this case the latest messages will be always available to the application.

  - If the FIFO lock function is enabled (RFLM bit in the CAN_MCR register set), the most recent message will be discarded, and the software will have the three oldest messages in the FIFO available.

The bxCAN Reception Handling – Reception Related Interrupts

- Once a message has been stored in the FIFO, the FMPx[1:0] bits are updated, and an interrupt request is generated if the FMPIEx (FIFO x message pending interrupt enable) bit in the CAN Interrupt Enable Register (CAN_IER) is set.

- When the FIFO becomes full (i.e., a third message is stored) the FULLx (FIFO x full) bit in the CAN_RFxR register is set, and an interrupt is generated if the FFIEx (FIFO x full interrupt enable) bit in the CAN_IER register is set.

- On overrun condition, the FOVRx (FIFO x overrun) bit in the CAN_RFxR register is set, and an interrupt is generated if the FOVIEx (FIFO x overrun interrupt enable) bit in the CAN_IER register is set.

CAN_IER: CAN Interrupt Enable Register
CAN_RFxR: CAN Receive FIFO x Register, x=0,1

Controller Area Network

CAN interrupt enable register (CAN_IER)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SLKIE	WKUIE
														rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERRIE	Res.	Res.	Res.	LEC IE	BOF IE	EPV IE	EWG IE	Res.	FOV IE1	FF IE1	FMP IE1	FOV IE0	FF IE0	FMP IE0	TME IE
rw				rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

SLKIE: Sleep interrupt enable

WKUIE: Wakeup interrupt enable

ERRIE: Error interrupt enable

LECIE: Last error code interrupt enable

BOFIE: Bus-off interrupt enable

EPVIE: Error passive interrupt enable

EWGIE: Error warning interrupt enable

FOVIE1: FIFO 1 overrun interrupt enable

FFIE1: FIFO 1 full interrupt enable

FMPIE1: FIFO 1 message pending interrupt enable

FOVIE0: FIFO 0 overrun interrupt enable

FFIE0: FIFO 0 full interrupt enable

FMPIE0: FIFO 0 message pending interrupt enable

TMEIE: Transmit mailbox empty interrupt enable

The bxCAN Identifier Filtering

- In the CAN protocol, the identifier of a message is related to the content of the message.
- A transmitter broadcasts its message to all receivers.
- On message reception, a receiver node decides, depending on the identifier value, whether the software needs the message or not.
- If the message is needed, it is copied into the SRAM. If not, the message must be discarded without intervention by the software.
- To fulfill this requirement the bxCAN Controller provides 28 configurable and scalable filter banks (0-27) to the application, to receive only the messages that are needed.
- This hardware filtering saves CPU resources. Each filter bank i (i = 0..27) consists of two 32-bit registers, CAN_FiR1 and CAN_FiR2, or (CAN_FiRx, i=0..27, x=1,2).

(Filter bank i register x) (32 filter bits)

The bxCAN Identifier Filtering – Filter Bank with Scalable Widths

- Each filter bank can be scaled independently.

- Depending on the filter scale, a filter bank provides:

    - One 32-bit filter for the STDID[10:0] or EXTID[28:0], IDE and RTR bits.

    - Two 16-bit filters for the STDID[10:0], RTR, IDE and EXTID[17:15] bits.

    Refer to diagram - Filter Bank Configuration & Register Organization.

- The filters can be configured as Identifier Mask mode or Identifier List mode.

STDID : standard identifier
EXTID : extended identifier
IDE : identifier extension
RTR : remote transmission request

Controller Area Network

CAN filter master register (CAN_FMR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	CANSB[5:0]						Res.	Res.	Res.	Res.	Res.	Res.	Res.	FINIT
		rw	rw	rw	rw	rw	rw								rw

CANSB[5:0]: CAN start bank

FINIT: Filter initialization mode

CAN filter mode register (CAN_FM1R)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	FBM27	FBM26	FBM25	FBM24	FBM23	FBM22	FBM21	FBM20	FBM19	FBM18	FBM17	FBM16
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FBM15	FBM14	FBM13	FBM12	FBM11	FBM10	FBM9	FBM8	FBM7	FBM6	FBM5	FBM4	FBM3	FBM2	FBM1	FBM0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

FBMx: Filter mode (identifier mask mode or identifier list mode)

(0 or 1)

CAN filter scale register (CAN_FS1R)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	FSC27	FSC26	FSC25	FSC24	FSC23	FSC22	FSC21	FSC20	FSC19	FSC18	FSC17	FSC16
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FSC15	FSC14	FSC13	FSC12	FSC11	FSC10	FSC9	FSC8	FSC7	FSC6	FSC5	FSC4	FSC3	FSC2	FSC1	FSC0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

FSCx: Filter scale configuration (2 x 16-bit or 32-bit)

(0 or 1)

Controller Area Network

CAN filter FIFO assignment register (CAN_FFA1R)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	FFA27	FFA26	FFA25	FFA24	FFA23	FFA22	FFA21	FFA20	FFA19	FFA18	FFA17	FFA16
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FFA15	FFA14	FFA13	FFA12	FFA11	FFA10	FFA9	FFA8	FFA7	FFA6	FFA5	FFA4	FFA3	FFA2	FFA1	FFA0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

FFAx: Filter FIFO assignment for filter x (0 or 1)

CAN filter activation register (CAN_FA1R)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	FACT 27	FACT 26	FACT 25	FACT 24	FACT 23	FACT 22	FACT 21	FACT 20	FACT 19	FACT 18	FACT 17	FACT 16
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FACT 15	FACT 14	FACT 13	FACT 12	FACT 11	FACT 10	FACT9	FACT8	FACT7	FACT6	FACT5	FACT4	FACT3	FACT2	FACT1	FACT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

FACTx: Filter active (1= active)

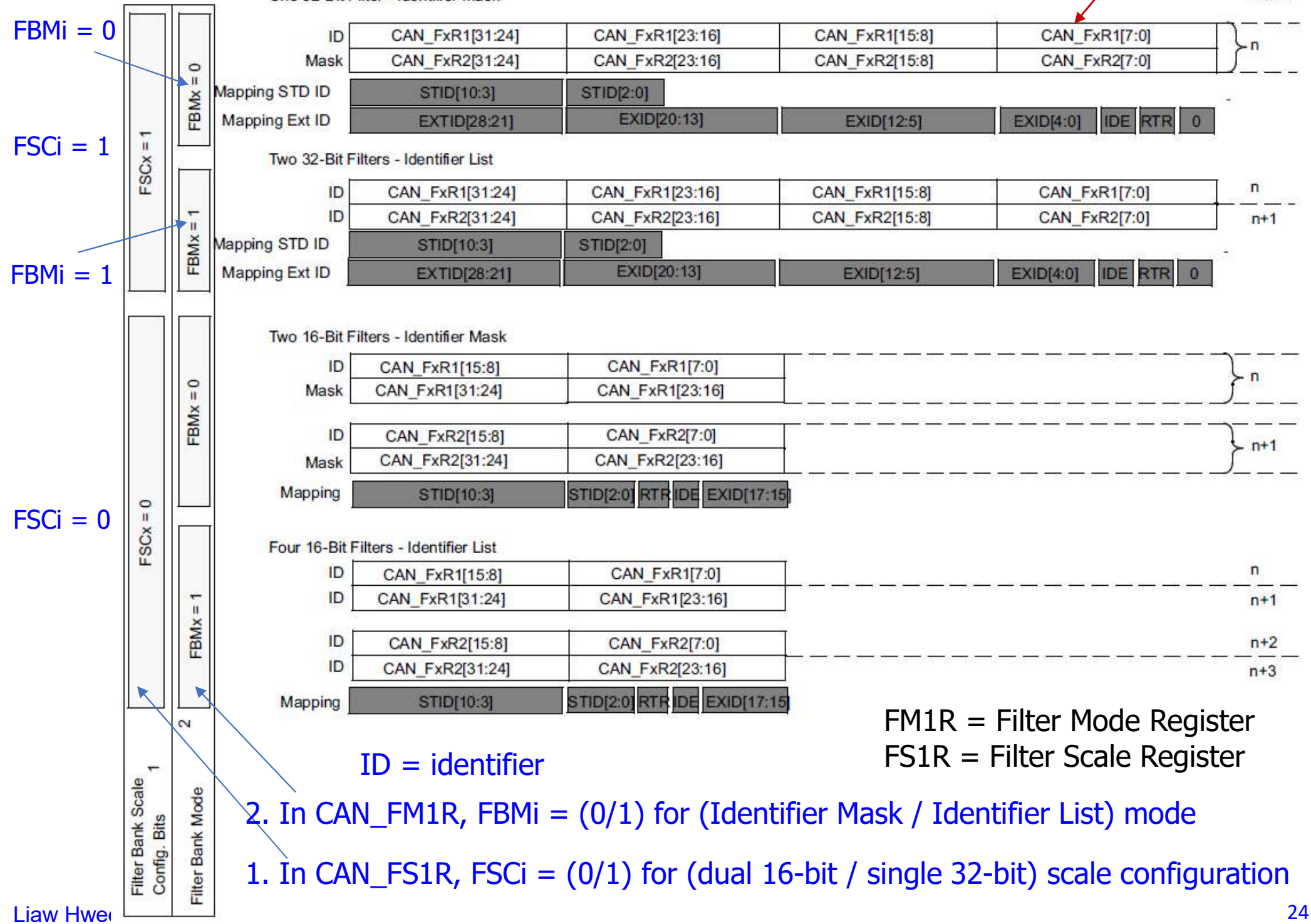
Filter bank i register x (CAN_FiRx) (i = 0..27, x = 1, 2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FB31	FB30	FB29	FB28	FB27	FB26	FB25	FB24	FB23	FB22	FB21	FB20	FB19	FB18	FB17	FB16
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FB15	FB14	FB13	FB12	FB11	FB10	FB9	FB8	FB7	FB6	FB5	FB4	FB3	FB2	FB1	FB0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

FB[31:0]: Filter bits

Filter bank number
x = i = 0..27

Filter Bank Configuration & Register Organization



FM1R = Filter Mode Register
FS1R = Filter Scale Register

ID = identifier

2. In CAN_FM1R, FBMi = (0/1) for (Identifier Mask / Identifier List) mode

1. In CAN_FS1R, FSCi = (0/1) for (dual 16-bit / single 32-bit) scale configuration

The bxCAN Identifier Filtering – Identifier Mask Mode or Identifier List Mode

Identifier Mask Mode

- In identifier mask mode the identifier registers are associated with mask registers specifying which bits of the identifier are handled as "must match" or as "don't care".

1 0

Identifier List Mode

- In identifier list mode, the mask registers are used as identifier registers. Thus, instead of defining an identifier and a mask, two identifiers are specified, doubling the number of single identifiers. All bits of the incoming identifier must match the bits specified in the filter registers.

The bxCAN Identifier Filtering – Filter Bank Scale and Mode Configuration

- The assignment of filter banks are configured by the CANSB[5:0] (CAN start bank) bits in the CAN Filter Master Register (CAN_FMR). Further, its FINIT (filter initialization mode) bit is set to initialize the filters.

- In the configuration of the filter bank i (i = 0..27), it must be deactivated by clearing the FACTi (filter active, i=0..27) bit in the CAN Filter Activation Register (CAN_FA1R).

- The filter scale is configured by the FSCi (filter scale configuration, i=0..27) bit in the CAN Filter Scale Register (CAN_FS1R).

- The identifier mask mode or identifier list mode is configured by the FBMi (filter mode, i=0..27) bit in the CAN Filter Mode Register (CAN_FM1R).

- The two receive FIFOs, FIFO 0 & FIFO 1, need to be assigned by FFAi (filter FIFO assignment for filter i, i=0..27) bit in the CAN Filter FIFO Assignment Register (CAN_FFA1R).

Note: CANSB[5:0] sets the CAN start bank of CAN2 for dual CAN, sets to zero for one CAN.

- The registers, CAN_FS1R, CAN_FM1R, and CAN_FFA1R are configured only when the filter initialization mode is set, i.e., FINIT=1 in CAN_FMR.
- To filter a group of identifiers, configure the identifier/mask registers in the identifier mask mode.
- To select a single identifier, configure the identifier/mask registers in the identifier list mode.
- Filters not used by the application should be left deactivated.
- Each filter within a filter bank is numbered (called the Filter Number) from 0 to a maximum number dependent on the mode and the scale of each of the filter bank.

CAN Filter Master Register (CAN_FMR)
CAN Filter Activation Register (CAN_FA1R)
CAN Filter Scale Register (CAN_FS1R)
CAN Filter Mode Register (CAN_FM1R)
CAN Filter FIFO Assignment Register (CAN_FFA1R)

The bxCAN Identifier Filtering – Filter Match Index

- Once a message has been received in the FIFO, it is copied into SRAM locations. To copy the data to the right location, the application must identify the data by means of the identifier.
- To ease the access to the SRAM locations, the CAN controller provides a **Filter Match Index.**
- This Filter Match Index is stored in (CAN_RDTxR) according to the filter priority rules.

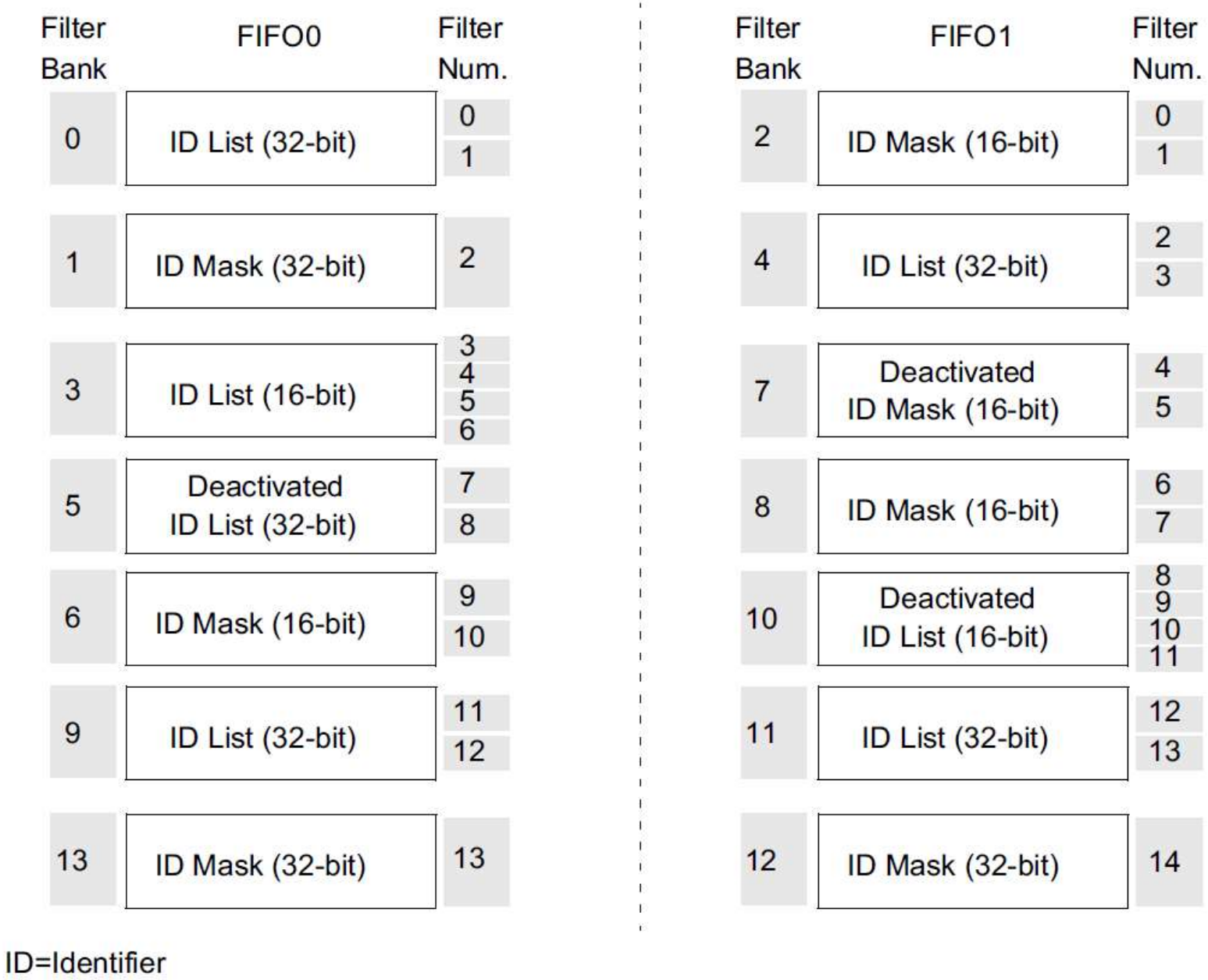
CAN receive FIFO mailbox data length control and time stamp register (CAN_RDTxR) (x = 0..1)

- Each received message has its associated Filter Match Index.
- The Filter Match Index can be used in two ways:
  - Compare the Filter Match index with a list of expected values.
  - Use the Filter Match Index as an index to access the data location.

The bxCAN Identifier Filtering – Remark

- For non masked filters, Mask = 0x0000, the software no longer needs to compare the identifier.
- If the filter is masked, the software reduces the comparison to the masked bits only.
- The value of the filter number does not consider the activation state of the filter banks. In addition, two independent numbering schemes are used, one for each FIFO.

Controller Area Network



ID=Identifier

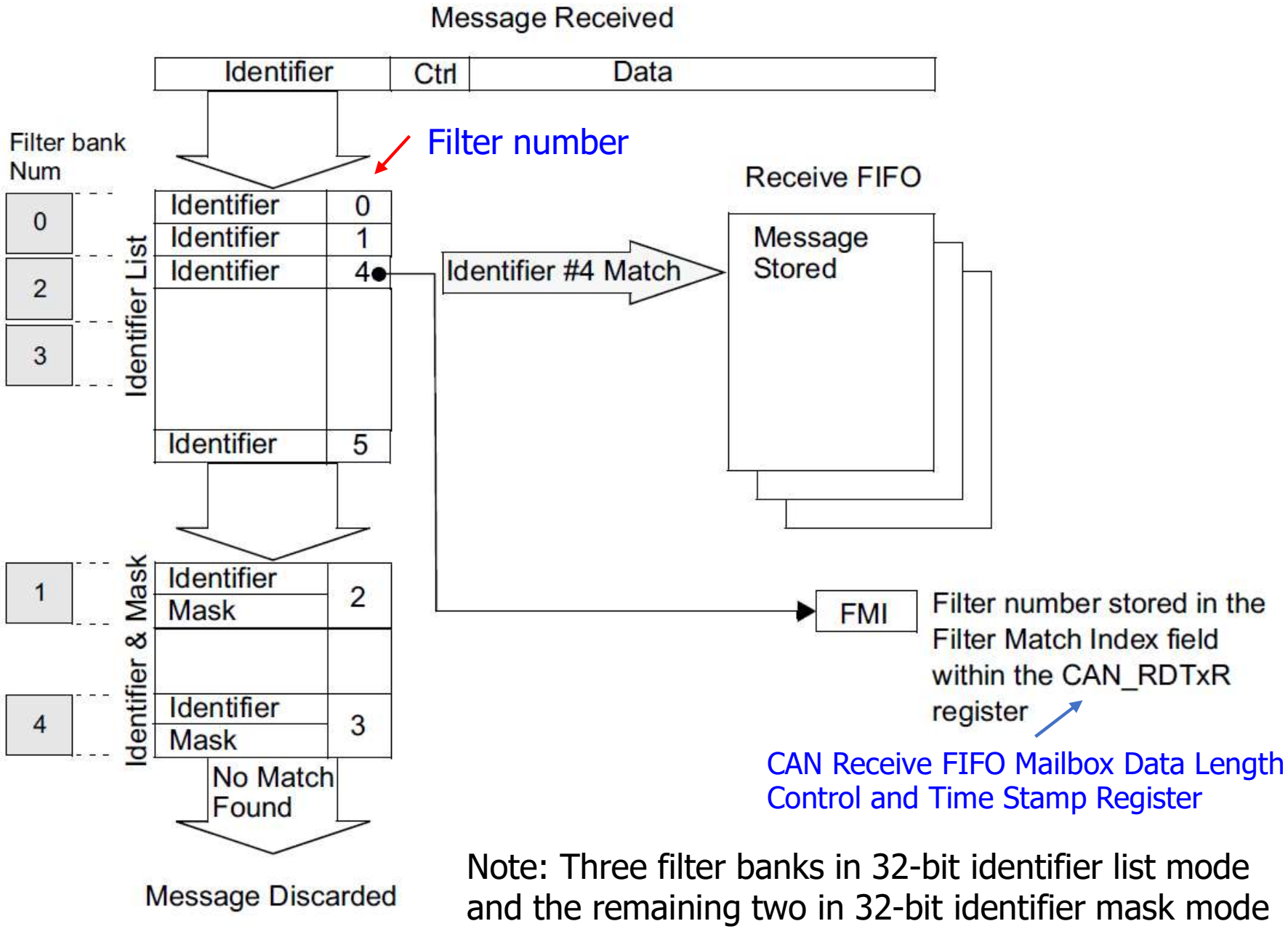
Example of Filter Numbering

Note: Each ID in the filter bank has an assigned filter number.

The bxCAN Identifier Filtering – Filter Priority Rules

- Depending on the filter combination, it may occur that an identifier passes successfully through several filters.
- In this case, the filter match value/index stored in the receive mailbox is chosen according to the following priority rules:
  - A 32-bit filter takes priority over a 16-bit filter.
  - For filters of equal scale, priority is given to the identifier list mode over the identifier mask mode
  - For filters of equal scale and mode, priority is given by the filter number (the lower the number, the higher the priority).

The bxCAN Identifier Filtering – Example of the Filtering Mechanism



Note: Three filter banks in 32-bit identifier list mode and the remaining two in 32-bit identifier mask mode

The bxCAN Identifier Filtering – Example of the Filtering Mechanism

- The example above shows the filtering principle of the bxCAN.
- On reception of a message, the identifier is compared first with the filters configured in identifier list mode.
- If there is a match, the message is stored in the associated FIFO and the index (filter number) of the matching filter is stored in the FMI[7:0] (filter match index) bits in the CAN Receive FIFO Mailbox Data Length Control and Time Stamp Register (CAN_RDTxR, x=0,1).
- In this example, the identifier matches with Identifier #4 thus the message content and FMI 4 is stored in the FIFO, i.e., in the CAN_RDLxR, CAN_RDHxR, and CAN_RDTxR. (CAN_RDLxR = CAN Receive FIFO Mailbox Data Low Register)
- If there is no match, the incoming identifier is then compared with the filters configured in identifier mask mode. If the identifier does not match any of the identifiers configured in the filters, the message is discarded by hardware without disturbing the software.

The bxCAN Message Storage

- The interface between the software and the hardware for the CAN messages is implemented by means of mailboxes. A mailbox contains all information related to a message; identifier, data, control, status and time stamp information.

Transmit mailbox

- The software sets up the message to be transmitted in an empty transmit mailbox. The status of the transmission is indicated by hardware in the CAN_TSR.

Receive mailbox

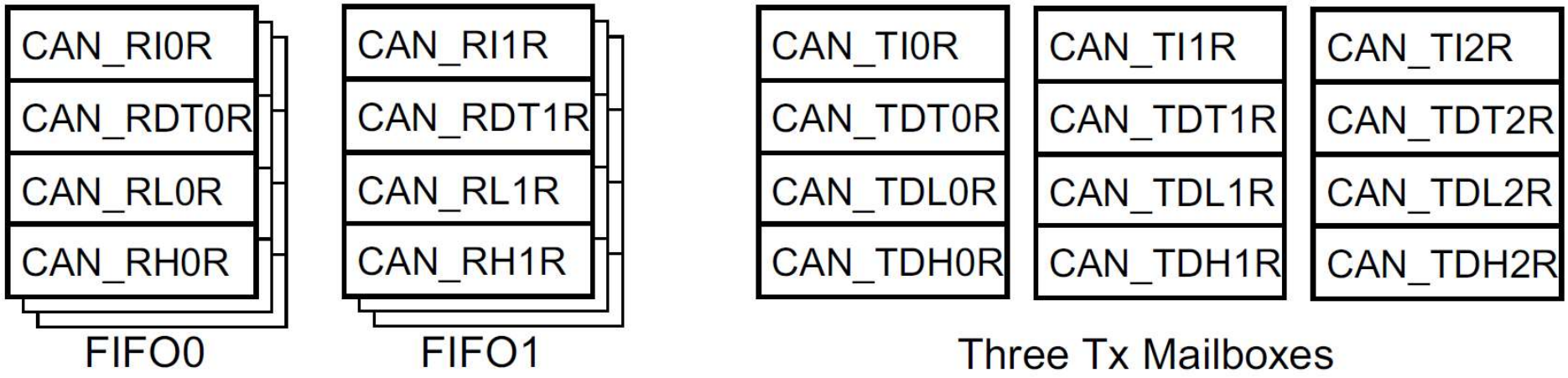
RFOM0: Release FIFO 0 output mailbox

- When a message has been received, it is available to the software in the FIFO output mailbox. Once the software has handled the message (e.g., read it), the software must release the FIFO output mailbox by means of the RFOM0 or RFOM1 bit in the CAN_RF0R or CANR1R register to make the next incoming message available. The filter match index and the 16-bit time stamp value are stored in the FMI[7:0] field and TIME[15:0] field, respectively, of the CAN_RDT0R or CAN_RDT1R.

CAN receive FIFO mailbox data length control and time stamp register

The bxCAN Message Storage

CAN_TSR (CAN transmit status register)
CAN_RF0R (CAN receive FIFO 0 register)
CAN_RF1R (CAN receive FIFO 1 register)



bxCAN mailbox registers

CAN_TIxR (CAN transmit mailbox identifier register, x = 0..2 for 3 TX mailboxes)
CAN_TDTxR (CAN transmit mailbox data length control and time stamp register, x = 0..2)
CAN_TDLxR (CAN transmit mailbox data low register, x = 0..2)
CAN_TDHxR (CAN transmit mailbox data high register, x = 0..2)
CAN_RIxR (CAN receive FIFO mailbox identifier register, x = 0,1 for 2 RX FIFOs)
CAN_RDTxR (CAN receive FIFO mailbox data length control & time stamp register, x=0,1)
CAN_RDLxR (CAN receive FIFO mailbox data low register, x = 0,1)
CAN_RDHxR (CAN receive FIFO mailbox data high register, x = 0,1)

Controller Area Network

CAN receive FIFO 0 register (CAN_RF0R)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RFOM0	FOVR0	FULL0	Res.	FMP0[1:0]	
										rs	rc_w1	rc_w1		r	r

RFOM0: Release FIFO 0 output mailbox

FOVR0: FIFO 0 overrun

FULL0: FIFO 0 full

FMP0[1:0]: FIFO 0 message pending

CAN receive FIFO 1 register (CAN_RF1R)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RFOM1	FOVR1	FULL1	Res.	FMP1[1:0]	
										rs	rc_w1	rc_w1		r	r

Controller Area Network

CAN receive FIFO mailbox identifier register (CAN_RIxR) (x = 0..1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
STID[10:0]/EXID[28:18]											EXID[17:13]				
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXID[12:0]													IDE	RTR	Res
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

STID[10:0]/EXID[28:18]: Standard identifier or extended identifier

EXID[17:0]: Extended identifier

IDE: Identifier extension

RTR: Remote transmission request

CAN receive FIFO mailbox data length control and time stamp register (CAN_RDTxR) (x = 0..1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TIME[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FMI[7:0]								Res.	Res.	Res.	Res.	DLC[3:0]			
r	r	r	r	r	r	r	r					r	r	r	r

TIME[15:0]: Message time stamp

FMI[7:0]: Filter match index

DLC[3:0]: Data length code

CAN receive FIFO mailbox data low register (CAN_RDLxR) (x = 0..1)

CAN receive FIFO mailbox data high register (CAN_RDHxR) (x = 0..1)

The bxCAN Error Management

- The error management is handled by hardware using a **Transmit Error Counter** (TEC[7:0] value) and a **Receive Error Counter** (REC[7:0] value) in the CAN Error Status Register (CAN_ESR), which will be incremented or decremented according to the error condition.

- TEC and REC can be read by software to determine the stability of the network.

- The CAN hardware provides detailed information on the current error status in the CAN_ESR. The error information includes the LEC[2:0] (3-bit last error code), BOFF (bus-off flag, set when TEC > 255), EPVF (error passive flag, set when TEC or REC > 127), and EWGF (error warning flag, set when TEC or REC >= 96).

- The CAN Interrupt Enable Register (CAN_IER), (ERRIE, LECIE, BOFIE, EPVIE or/and EWGIE bit), enable the interrupt generation about the error detection.

Controller Area Network

CAN error status register (CAN_ESR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
REC[7:0]								TEC[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	LEC[2:0]			Res.	BOFF	EPVF	EWGF
									rw	rw	rw		r	r	r

REC[7:0]: Receive error counter

TEC[7:0]: transmit error counter

LEC[2:0]: Last error code

BOFF: Bus-off flag

EPVF: Error passive flag

EWGF: Error warning flag

This field is set by hardware and holds a code which indicates the error condition of the last error detected on the CAN bus. If a message has been transferred (reception or transmission) without error, this field will be cleared to '0'.
The LEC[2:0] bits can be set to value 0b111 by software. They are updated by hardware to indicate the current communication status.
000: No Error
001: Stuff Error
010: Form Error
011: Acknowledgment Error
100: Bit recessive Error
101: Bit dominant Error
110: CRC Error
111: Set by software

Controller Area Network

CAN interrupt enable register (CAN_IER)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SLKIE	WKUIE
														rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERRIE	Res.	Res.	Res.	LEC IE	BOF IE	EPV IE	EWG IE	Res.	FOV IE1	FF IE1	FMP IE1	FOV IE0	FF IE0	FMP IE0	TME IE
rw				rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

SLKIE: Sleep interrupt enable

WKUIE: Wakeup interrupt enable

ERRIE: Error interrupt enable

LECIE: Last error code interrupt enable

BOFIE: Bus-off interrupt enable

EPVIE: Error passive interrupt enable

EWGIE: Error warning interrupt enable

FOVIE1: FIFO 1 overrun interrupt enable

FFIE1: FIFO 1 full interrupt enable

FMPIE1: FIFO 1 message pending interrupt enable

FOVIE0: FIFO 0 overrun interrupt enable

FFIE0: FIFO 0 full interrupt enable

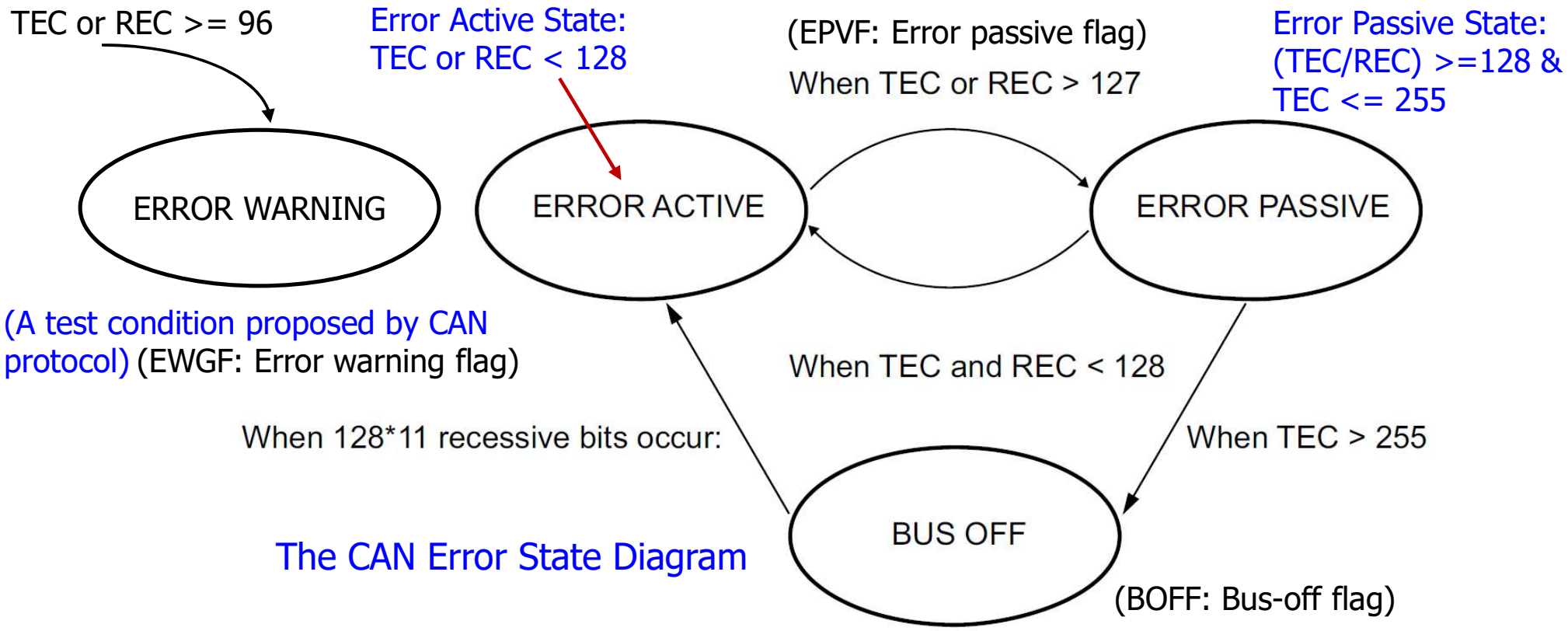
FMPIE0: FIFO 0 message pending interrupt enable

TMEIE: Transmit mailbox empty interrupt enable

The bxCAN Error Management – Bus-Off Recovery

- The Bus-Off state is reached when TEC is greater than 255, this state is indicated by BOFF (Bus-off flag) bit in CAN_ESR.
- In Bus-Off state, the bxCAN is no longer able to transmit and receive messages.
- Depending on the ABOM (automatic bus-off management) bit in the CAN_MCR, bxCAN will recover from the Bus-Off (become error active again) either automatically (ABOM = 1) or on software request (ABOM = 0).
- But in both cases, the bxCAN must wait at least for the recovery sequence specified in the CAN standard (128 occurrences of 11 consecutive recessive bits monitored on CANRX pin).
- If ABOM is set, the bxCAN will start the recovering sequence automatically after it has entered the Bus-Off state.
- If ABOM is cleared, the software must initiate the recovering sequence by requesting bxCAN to enter and to leave Initialization mode.

- Note: In Initialization mode, bxCAN does not monitor the CANRX signal, therefore it cannot complete the recovery sequence. To recover, bxCAN must leave the Initialization mode.



The CAN Error State Diagram

TEC & **REC**: Implement the fault confinement mechanism of the CAN protocol/specification.

REC: In case of an error during reception, this counter is incremented by 1 or by 8 depending on the error condition as defined by the CAN standard. After every successful reception, the counter is decremented by 1 or reset to 120 if its value was higher than 128. When the counter value exceeds 127, the CAN controller enters the error passive state. (refer CAN Fault Confinement)