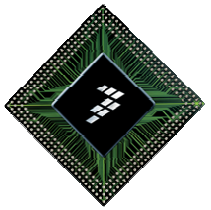
Sept, 2010

FlexCAN



freescale™
semiconductor

FlexCAN Training Outline

1. Module Overview
2. On-chip interconnects and inter-module dependencies
3. Software configuration
4. Hardware configuration/considerations
5. Demo code explanation
6. Frequently asked question list
7. Reference material

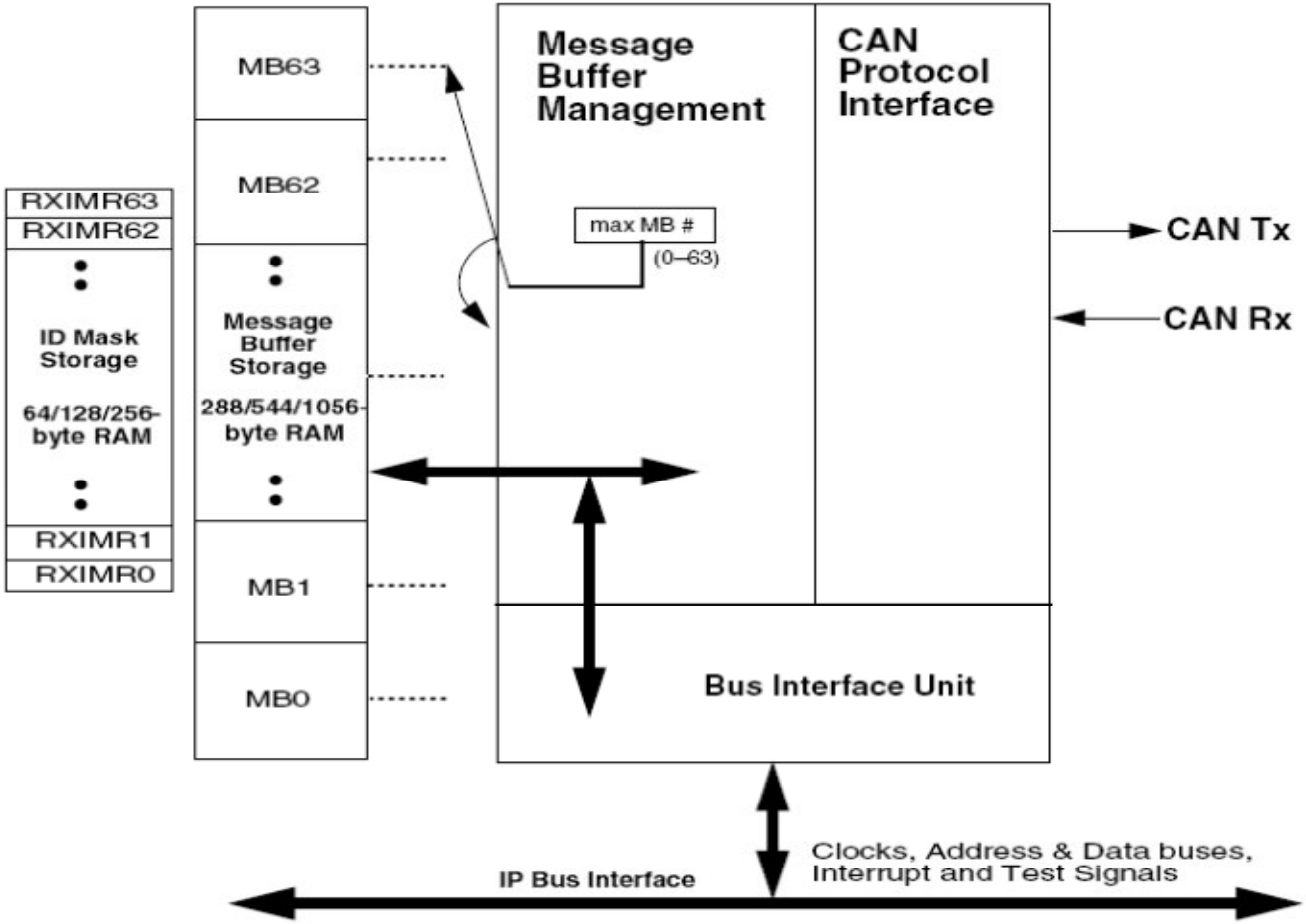
freescale™ semiconductor

FlexCAN Training Outline

1. **Module Overview**
2. On-chip interconnects and inter-module dependencies
3. Software configuration
4. Hardware configuration/considerations
5. Demo code explanation
6. Frequently asked question list
7. Reference material

freescale™ semiconductor

Block Diagram



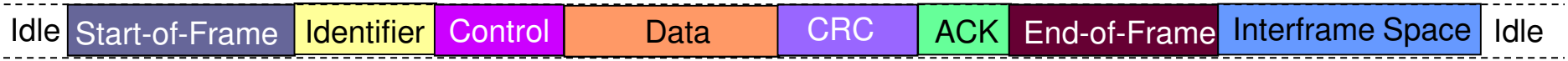
NOTE:

Kinetis only supports16 MBs.

freescale semiconductor

Feature list

- Full Implementation of the CAN protocol specification, Version 2.0B
  - Standard data and remote frames
  - Extended data and remote frames
  - Zero to eight bytes data length
  - Programmable bit rate up to 1 Mb/sec
  - Content-related addressing
- Flexible Mailboxes (up to 16 for Kinetis) of zero to eight bytes data length
- Each Mailbox configurable as Rx or Tx, all supporting standard and extended messages
- Individual Rx Mask Registers per Mailbox
- Full featured Rx FIFO with storage capacity for up to 6 frames and automatic internal pointer handling



freescale™ semiconductor

Feature list

- Transmission abort capability
- Programmable clock source to the CAN Protocol Interface, either bus clock or crystal oscillator
- Unused structures space can be used as general purpose RAM space
- Listen-only mode capability
- Programmable transmission priority scheme: lowest ID, lowest buffer number or highest priority
- Time Stamp based on 16-bit free-running timer
- Global network time, synchronized by a specific message
- Mask-able interrupts

Feature list

- Independent of the transmission medium (an external transceiver is assumed)
- Short latency time due to an arbitration scheme for high-priority messages
- Low power modes with programmable wake up on bus activity
- Rich status bits for Error conditions and fault confinement
  - Tx Buffer status (lowest priority buffer or empty buffer)
  - CRC status for just transmitted message
  - Synchronization status between FlexCAN and the CAN bus

Feature list

- Multiple Rx Mailboxes can be configured as reception queue with the same acceptance criteria to allow more time for the CPU to service received messages
- Remote request frames can be handled automatically or by software
- Safe mechanism for ID filter re-configuration
- Selectable priority of message reception between Mailboxes and Rx FIFO
- Powerful Rx FIFO ID filtering capable of matching incoming IDs against either 128 extended IDs, 256 standard IDs or 512 partial IDs, with up to 16 individual masking capability

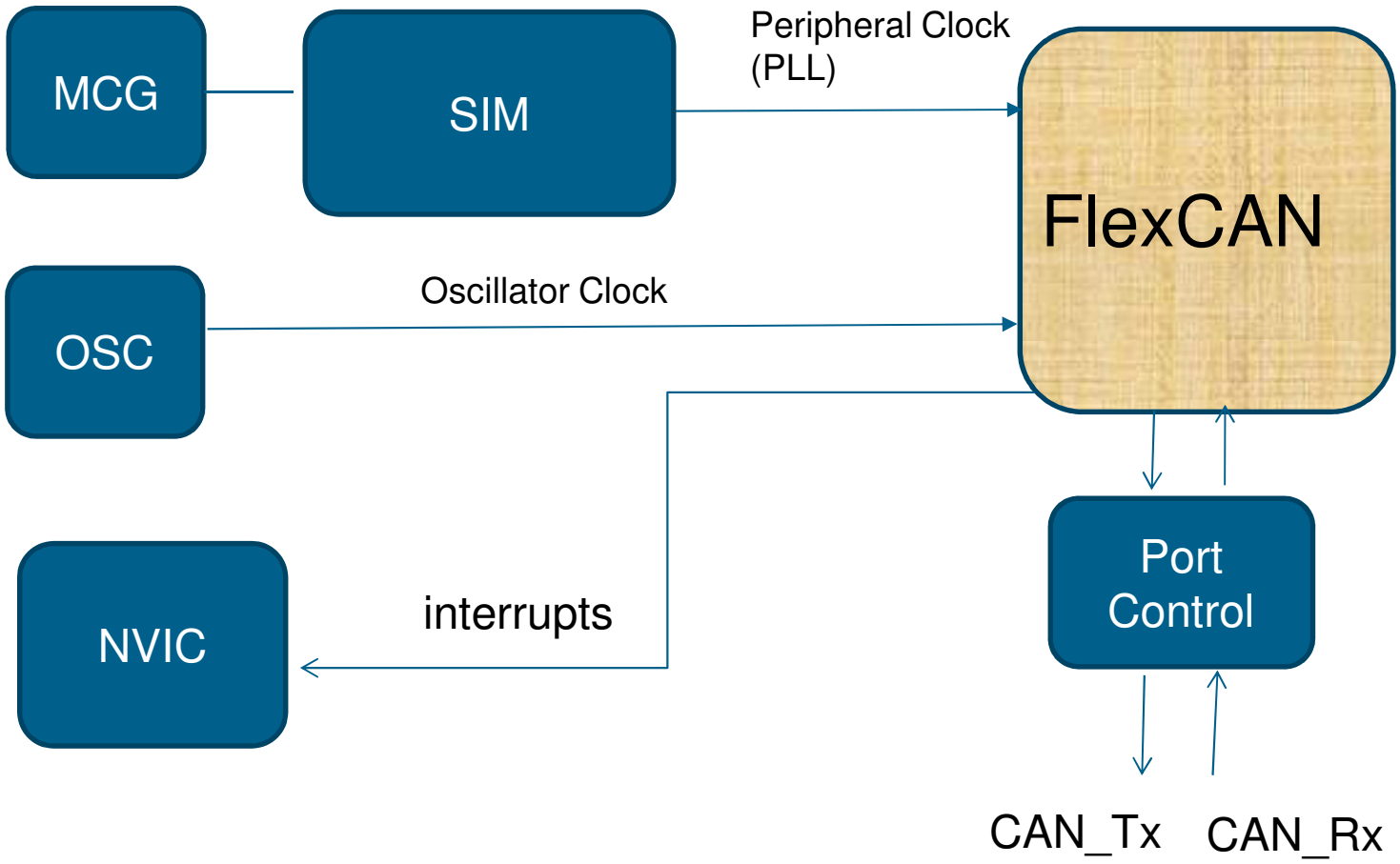
freescale ™
semiconductor

FlexCAN Training Outline

1. Module Overview
2. **On-chip interconnects and inter-module dependencies**
3. Software configuration
4. Hardware configuration/considerations
5. Demo code explanation
6. Frequently asked question list
7. Reference material

freescale™ semiconductor

SoC interconnect diagram



freescale™ semiconductor

Module dependencies

- *Clock source setup*
  - *FlexCAN requires both Bus clock (from PLL) and External reference clock (ERCLK)*
    - *External crystal from 1 to 32MHz must be connected*
    - *Configure MCG to enable PLL to the desired bus frequency*
    - *Enable OSC (Oscillator) to output ERCLK clock by setting OSC_CR[ERCLKEN] bit*
  - *If bus clock is from FLL, the high bit rate may not be achieved due to CAN clock tolerance requirement 1.58%*

Module dependencies

- *Clock gating*
  - Prior to using FlexCAN1, bit 4 (FLEXCAN1) in System Clock Gating Control Register 3 (SIM_SCGC3) must be set
  - Prior to using FlexCAN0, bit 4 (FLEXCAN0) in SIM_SCGC6 must be set
  - enable the clock(s) to the corresponding port(s) (refer to the following slides) whose pins are to function as FlexCAN pins
  - These clocks can be disabled
    - if the ports are not used after FlexCAN pin configuration
    - that is, other pins of the ports are not used as I/O port pins

Module dependencies

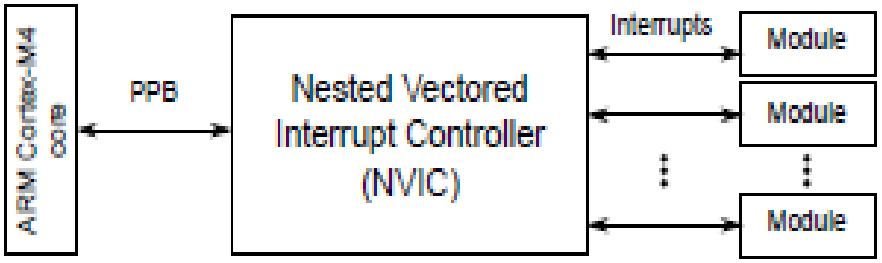
I/O configuration

- CAN pins are multiplexed with other peripheral modules via Port Control Module
- The corresponding CAN pins must be configured as CAN pins by selecting appropriate ALTn function

Port Pin name	ALT2	Correct Settings for CAN
PTE24	CAN1_TX	PORTE_PCR24[MUX] =2
PTE25	CAN1_RX	PORTE_PCR25[MUX] =2
PTC16	CAN1_RX	PORTC_PCR16[MUX] =2
PTC17	CAN1_TX	PORTC_PCR17[MUX] =2
PTA12	CAN0_TX	PORTA_PCR12[MUX] =2
PTA13	CAN0_RX	PORTA_PCR13[MUX] =2
PTB18	CAN0_TX	PORTB_PCR18[MUX] =2
PTB19	CAN0_RX	PORTB_PCR19[MUX] =2

Module dependencies

- *Nested Vectored Interrupt Controller (NVIC) Configuration for FlexCAN*
  - *The corresponding register bits*
  - *Must be set*



FlexCAN0 Interrupt Vectors:

Address Offset	Vector	IRQ	Source	Source Description	NVIC Register Bits
0x0000_00B4	45	29	CAN0	OR'ed Message buffer (0-15)	NVICISER0[29]
0x0000_00B8	46	30	CAN0	Bus Off	NVICISER0[30]
0x0000_00BC	47	31	CAN0	Error	NVICISER0[31]
0x0000_00C0	48	32	CAN0	Transmit Warning	NVICISER1[0]
0x0000_00C4	49	33	CAN0	Receive Warning	NVICISER1[1]

freescale semiconductor

Module dependencies

Address Offset	Vector	IRQ	Source	Source Description	NVIC Register Bits
0x0000_00C8	50	34	CAN0	Wake Up	NVICISER1[2]
0x0000_00CC	51	35	CAN0	Individual Matching Elements Update (IMEU)	NVICISER1[3]
0x0000_00D0	52	36	CAN0	Lost Receive	NVICISER1[4]

NOTE:
Before setting the corresponding bits in NVICISERn, set the corresponding bits In NVICICPRn

FlexCAN1 Interrupt Vectors:

Address Offset	Vector	IRQ	Source	Source Description	NVIC Register Bits
0x0000_00D4	53	37	CAN1	OR'ed Message buffer (0-15)	NVICISER1[5]
0x0000_00D8	54	38	CAN1	Bus Off	NVICISER1[6]
0x0000_00DC	55	39	CAN1	Error	NVICISER1[7]
0x0000_00E0	56	40	CAN1	Transmit Warning	NVICISER1[8]
0x0000_00E4	57	41	CAN1	Receive Warning	NVICISER1[9]
0x0000_00E8	58	42	CAN1	Wake Up	NVICISER1[10]
0x0000_00EC	59	43	CAN1	Individual Matching Elements Update (IMEU)	NVICISER1[11]
0x0000_00F0	60	44	CAN1	Lost Receive	NVICISER1[12]

Module dependencies

- *Sequencing requirements / recommendation for initializing the above dependencies*
  - *Initialize MCG & OSC to enable PLL and ERCLK*
  - *Initialize the clock gating in SIM to enable clocks to FlexCAN module(s) and to the corresponding ports whose pins are to be used as FlexCAN pins*
  - *Configure the corresponding port pins for FlexCAN through Port Control*
  - *Initialize the NVIC registers to enable corresponding interrupts (NOTE: the initialization of NVIC registers for FlexCAN interrupts can be done later before starting CAN communication)*

FlexCAN Training Outline

1. Module Overview
2. On-chip interconnects and inter-module dependencies
3. **Software configuration**
4. Hardware configuration/considerations
5. Demo code explanation
6. Frequently asked question list
7. Reference material

freescale™ semiconductor

Sequence of register setup

Step1

- Make sure FlexCAN module is disbled (after reset, it is disabled)
- Select clock source for FlexCAN by setting/clearing CTRL1[CLK_SRC] bit

Step2

- Enable FlexCAN module by clearing MCR[MDIS] bit
- Wait till FlexCAN module out of low power mode (MCR[LPM_ACK] = 0)

Step3

- Wait till FlexCAN goes into Freeze mode (MCR[FRZ_ACK] = 1)

Sequence of register setup

- Initialize other MCR bits as needed
  - enable the individual filtering per MB and reception queue features by setting MCR[IRMQ] bit
  - enable the warning interrupts by setting the MCR[WRN_EN] bit
  - disable self reception by setting the MCR[SRX_DIS] bit
  - enable the RxFIFO by setting MCR[RFEN] bit
  - enable the abort mechanism by setting the MCR[AEN] bit
  - enable the local priority feature by setting the MCR[LPRIO_EN] bit

freescale™ semiconductor

Sequence of register setup

- Configure baud rate and initialize CTRL1 and CTRL2 as needed
  - Determine the bit timing parameters: PROPSEG,PSEG1,PSEG2,RJW
  - Determine the bit rate by programming the PRESDIV field
  - Determine the internal arbitration mode (LBUF bit)
  - Initialize CTRL2[RFFN], CTRL2[MRP], CTRL2[RRS],CTRL2[EACEN] and keep CTRL2[TASD] as default

freescale™ semiconductor

Sequence of register setup

- Initialize the Message Buffers (MB)
  - must initialize control/status word of all MBs
  - must initialize the ID filter table if Rx FIFO was enabled
  - Initialize other field of MBs as required

freescale™ semiconductor

Sequence of register setup

Step7

- Initialize the Rx Individual Mask Registers (RXIMRn) if individual Rx masking & queue is enabled (MCR[IRMQ]=1)

Step8

- enable the corresponding interrupts by setting required interrupt mask bits in IMASKn register (for all MB interrupts), CTRLn register (for Bus off & Error interrupts), and MCR register (for wakeup interrupt)

Step9

- Negate the MCR[HALT] bit
- Wait till FlexCAN is out of freeze mode (MCR[FRZ_ACK] = 0)

Message Buffer Structure

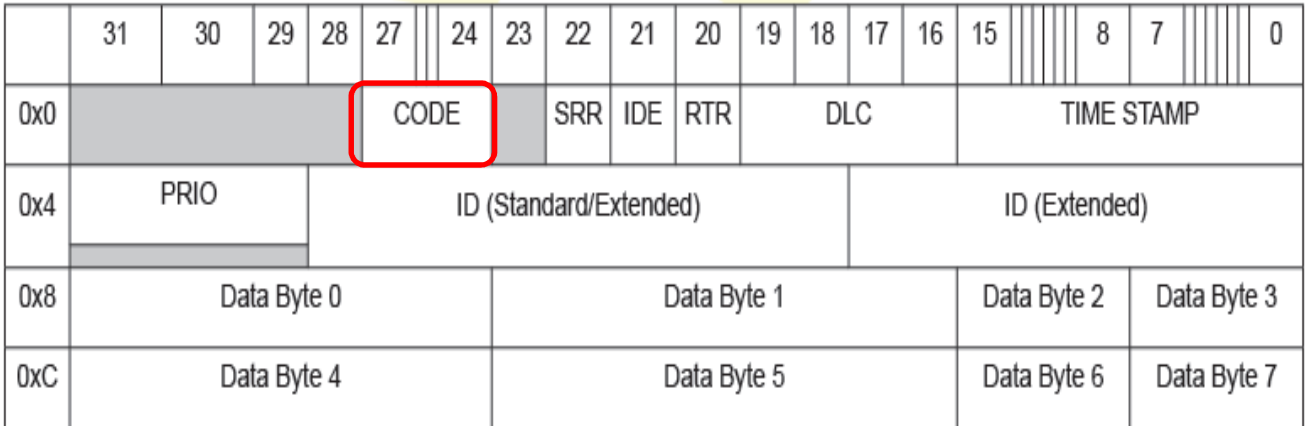
Field Group

Control/Status(C/S)

ID

Data Bytes word0

word1



- *Each consists of Control/Status field, ID field, and Data Bytes field as shown above*
- *ID fields stores the identifier of a message frame to be transmitted or received and also function as an ID acceptance code for a receive MB*
  - *For a Rx MB, if the ID field of the received message matches the ID acceptance code of this Rx MB, this received message will be accepted and stored in this Rx MB*
  - *For a Tx MB, the ID field will be transmitted as identifier field of the message frame*
- *IDE indicates it is standard frame or extended frame*
- *DLC stores the # of data bytes (<=8) of a message*
- *RTR indicates it is remote frame or data frame*
- *SRR is s*ubstitute remote request bit
- *Time stamp stores the* sampled free running timer at the beginning of the Identifier field on the CAN bus
- *PRIO specifies local transmit priority*

- ► *Transmit / Receive of a message is based on CODE bits*
- ► *Data Bytes are in Big Endian format and consists of word 0 and word 1*
- ► *Up to 16 MBs are supported by Kinetis*

freescale™ semiconductor

Message Buffer Structure

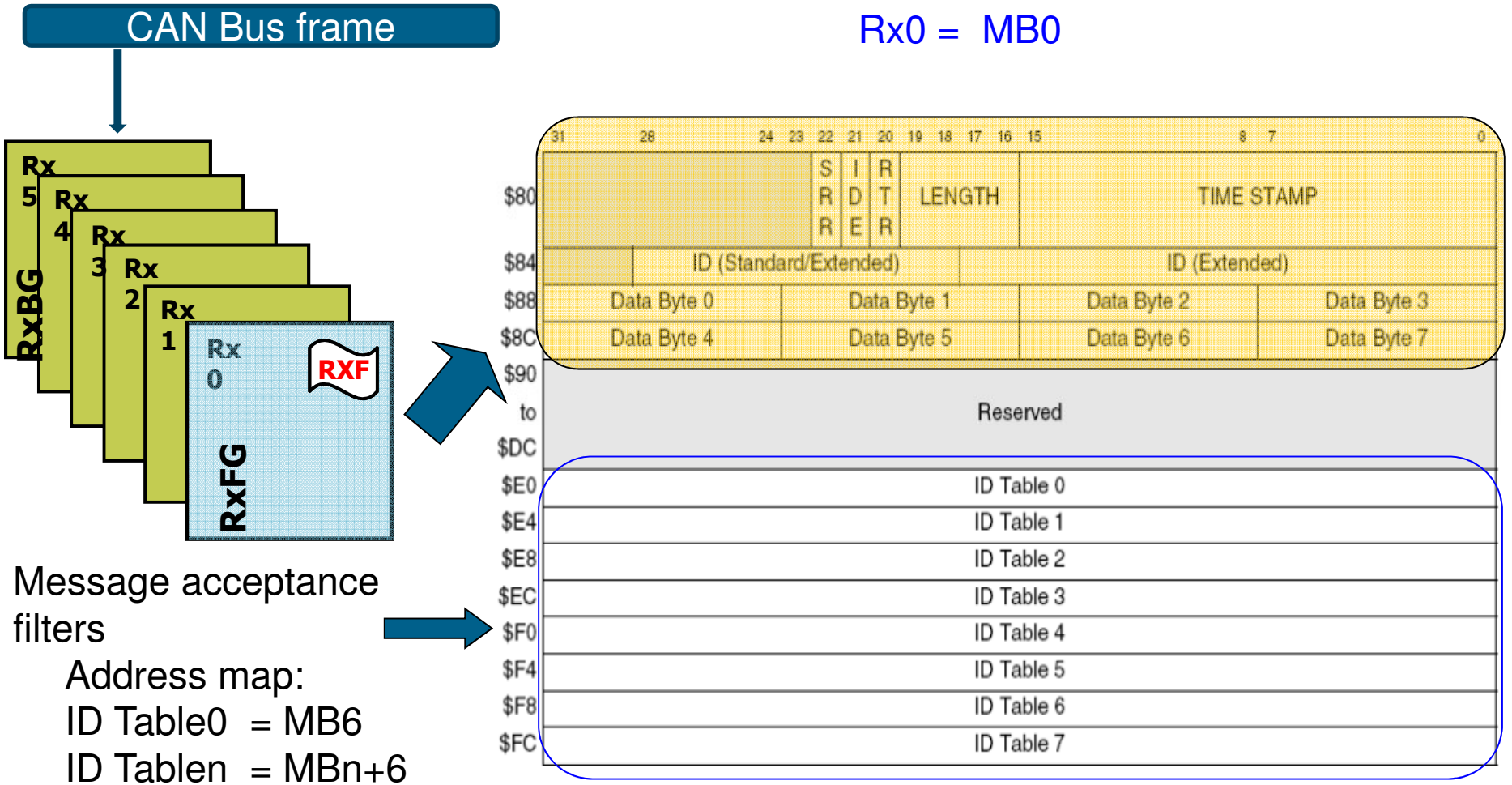
- CODE: Message Buffer Code indicating control/status code of a MB
  - For a Rx MB:
    - 0000 -- Inactive; 0100 – Empty and Active; 0010 – Full
    - 0110 -- Overrun; 0XY1 -- Busy ;
    - 1010 – RANSWER: a response frame is configured in this MB to recognize a remote request frame; after matching, code is automatically changed to TANSWER to transmit this response frame
  - For a Tx MB:
    - 1000 -- Inactive; 1001 – Abort; 1100 – Tx a data frame once then inactive if RTR = 0;Tx remote frame once then as a Rx MB with the same ID if RTR = 1;
    - 1110 – TANSWER: Tx a response frame as a remote response frame then return code to RANSWER (1010)

Macros for Message Buffer Fields

- #define FLEXCAN0_MBn_CS(n) \
  (*(vuint32_t*)(FLEXCAN0_BASE+0x80+n*0x10))
- #define FLEXCAN0_MBn_ID(n) \
  (*(vuint32_t*)(FLEXCAN0_BASE+0x84+n*0x10))
- #define FLEXCAN0_MBn_WORD0(n) \
  (*(vuint32_t*)(FLEXCAN0_BASE+0x88+n*0x10))
- #define FLEXCAN0_MBn_WORD1(n) \
  (*(vuint32_t*)(FLEXCAN0_BASE+0x8C+n*0x10))
- #define FLEXCAN1_MBn_CS(n) \
  (*(vuint32_t*)(FLEXCAN1_BASE+0x80+n*0x10))
- #define FLEXCAN1_MBn_ID(n) \
  (*(vuint32_t*)(FLEXCAN1_BASE+0x84+n*0x10))
- #define FLEXCAN1_MBn_WORD0(n) \
  (*(vuint32_t*)(FLEXCAN1_BASE+0x88+n*0x10))
- #define FLEXCAN1_MBn_WORD1(n) \
  (*(vuint32_t*)(FLEXCAN1_BASE+0x8C+n*0x10))

Rx FIFO Structure

- MCR[RFEN] = 1



Message acceptance filters

Address map:

ID Table0 = MB6

ID Tablen = MBn+6

Rx0 is MB0, output of the rx FIFO

freescale semiconductor

Rx FIFO Structure

- ID Table Structure (selected by MCR[IDAM] bit)

Format	31	30	29				24	23						16	15	14	13					8	7					1	0
A	RTR	IDE	RXIDA (Standard = 29-19, Extended = 29-1)																										
B	RTR	IDE	RXIDB_0 (Standard = 29-19, Extended = 29-16)												RTR	IDE	RXIDB_1 (Standard = 13-3, Extended = 13-0)												
C	RXIDC_0 (Std/Ext = 31-24)								RXIDC_1 (Std/Ext = 23-16)								RXIDC_2 (Std/Ext = 15-8)								RXIDC_3 (Std/Ext = 7-0)			

- NOTE:
- Kinetis supports up to 40 RxFIFO filter table elements (ID Tables)
- Maximum value of CTRL2[RFFN] = 4
- CPU must enable and configure the FIFO during Freeze Mode

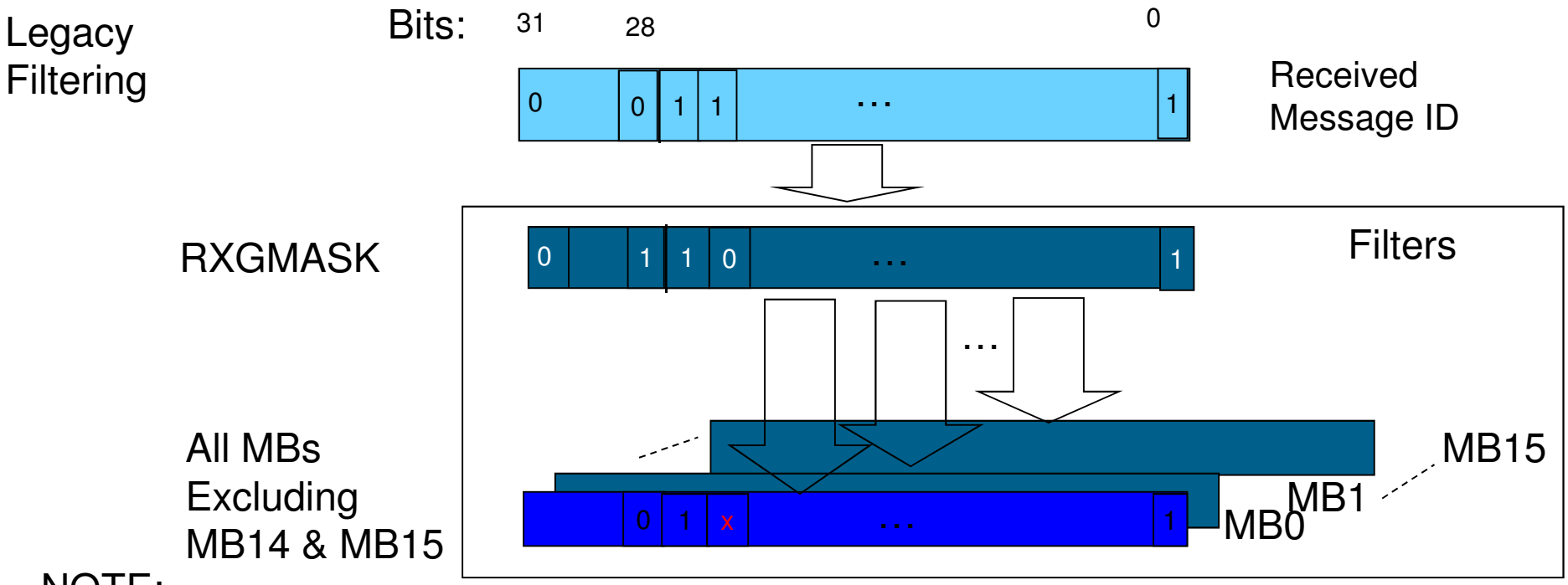
Interrupt Flags for Rx FIFO

- IFLAG1[5] is frame available in FIFO flag
- IFLAG1[6] is FIFO Warning flag
- IFLAG1[7] is FIFO overflow flag
- IFLAG1[4] to IFLAG1[0] not used

▶ Macros for ID Filter Table Elements

```
#define FLEXCAN0_IDFLT_TAB(n)
        (*(vuint32_t*)(FLEXCAN0_BASE+0xE0+(n*4)))
#define FLEXCAN1_IDFLT_TAB(n)
        (*(vuint32_t*)(FLEXCAN1_BASE+0xE0+(n*4)))
```

FlexCAN Message Filtering – Global Masking



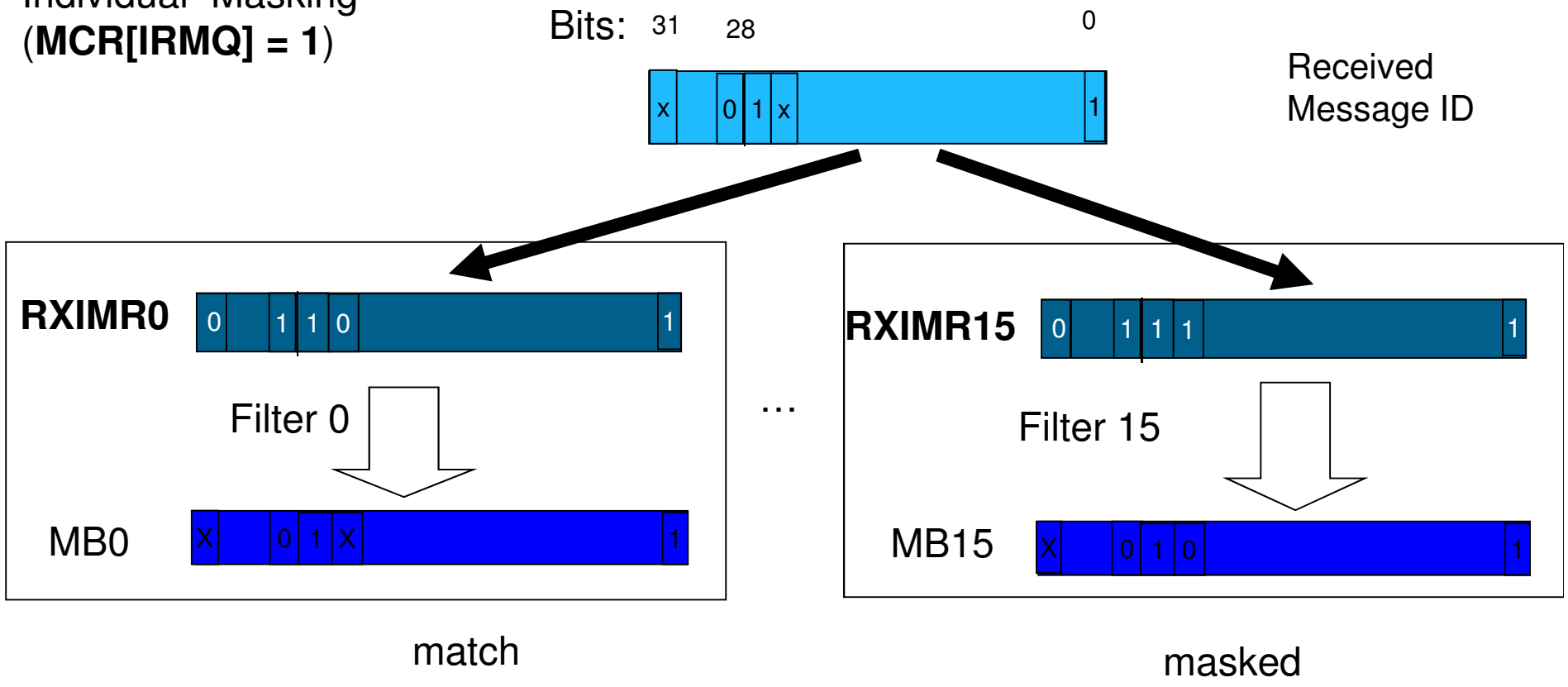
NOTE:

- MB14 uses **RX14MASK,** MB15 uses **RX15MASK** instead of **RXGMASK**
- Each MB acts as both filter acceptance register and the message storage
- When matching a MB, the received message is stored in the MB; masked otherwise
- Rx Mask Register bits:
  - 1 = The corresponding bit in the filter is checked against the one received
  - 0 = the corresponding bit in the filter is “don’t care”

freescale semiconductor

FlexCAN Message Filtering -- Individual Masking per MB

Individual Masking
(**MCR[IRMQ] = 1**)

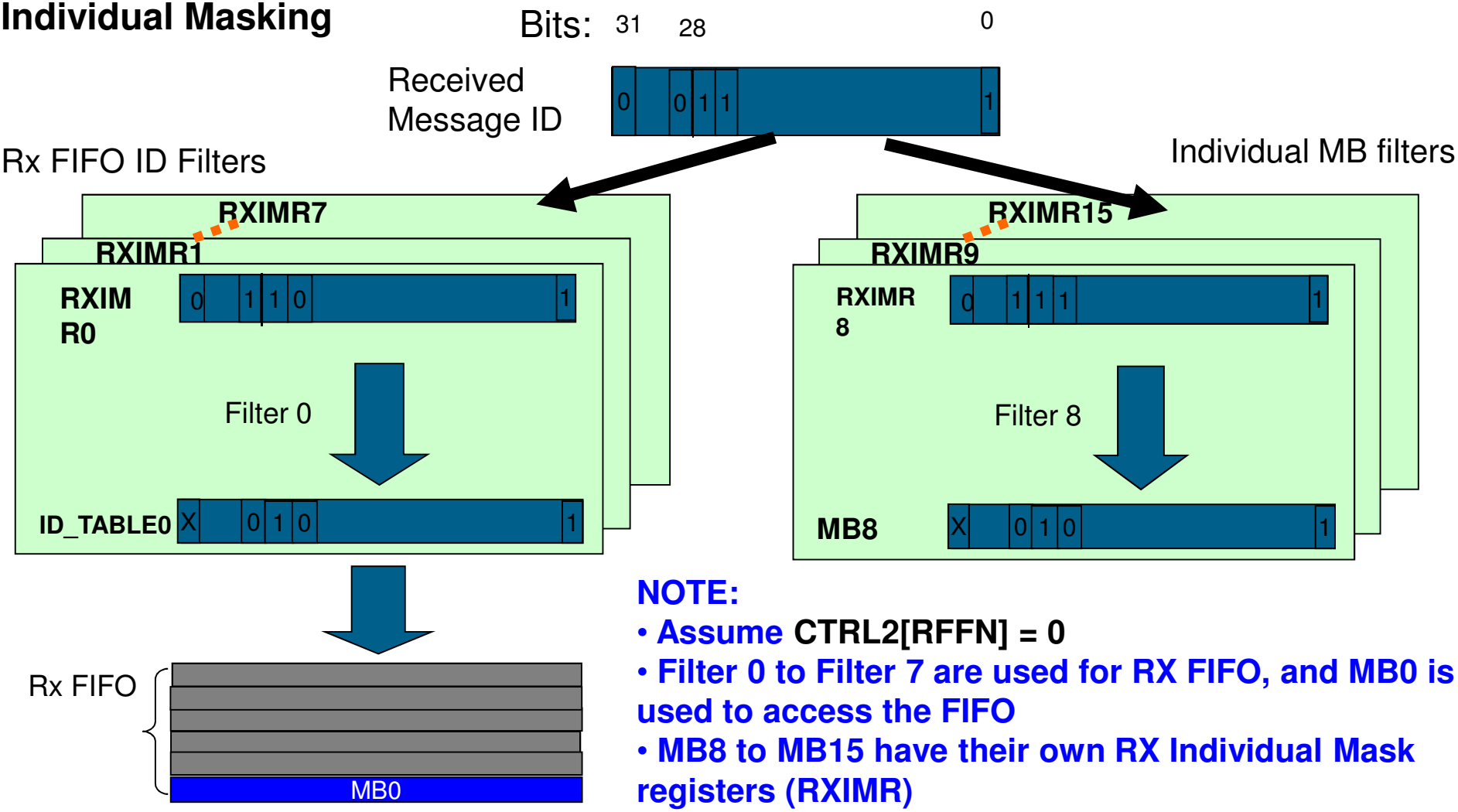


NOTE: Each MB has its own RX Individual Mask register (RXIMR)
"X" means don't care bit

freescale™ semiconductor

FlexCAN Message Filtering -- Mixed Filters

Mixed Rx FIFO & Individual Masking



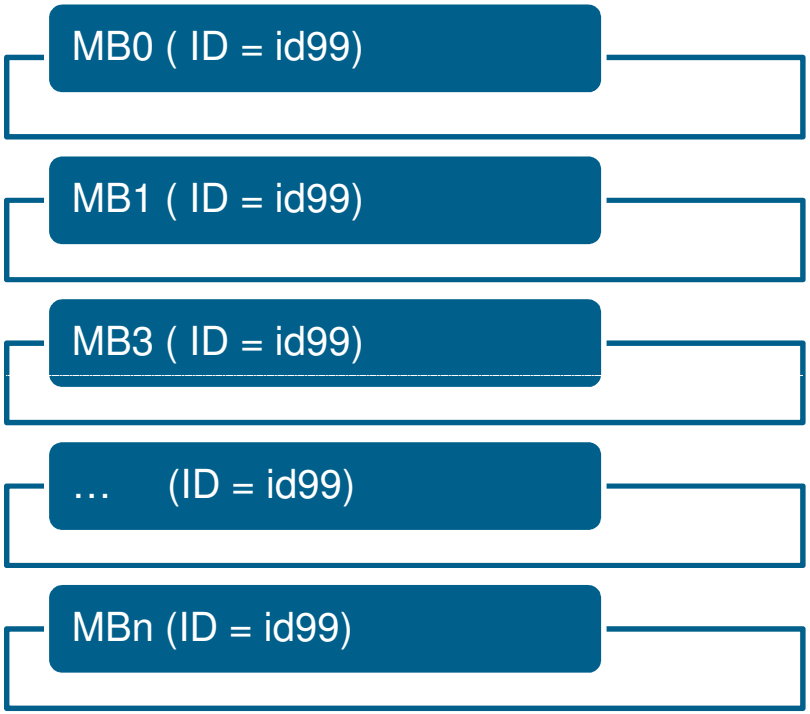
NOTE:

- **Assume CTRL2[RFFN] = 0**
- **Filter 0 to Filter 7 are used for RX FIFO, and MB0 is used to access the FIFO**
- **MB8 to MB15 have their own RX Individual Mask registers (RXIMR)**
- **Scan priority: RxFIFO first or MB first controlled by CTRL2[MRP]**

freescale™ semiconductor

Message Queue

- Allow more time for the CPU to service the MBs
- Condition:
  - MCR[IRMQ] = 1
- All MBs with same ID will be queued together
- If the previous MBs with same IDs are not “free-to-receive”, the current message will be stored to the next “free-to-receive” MB
- If the queue is full, store the message to the end of queue with “OVERRUN” status code
- The CPU can examine the Time Stamp field of the MBs to determine the order in which the messages arrived



freescale™ semiconductor

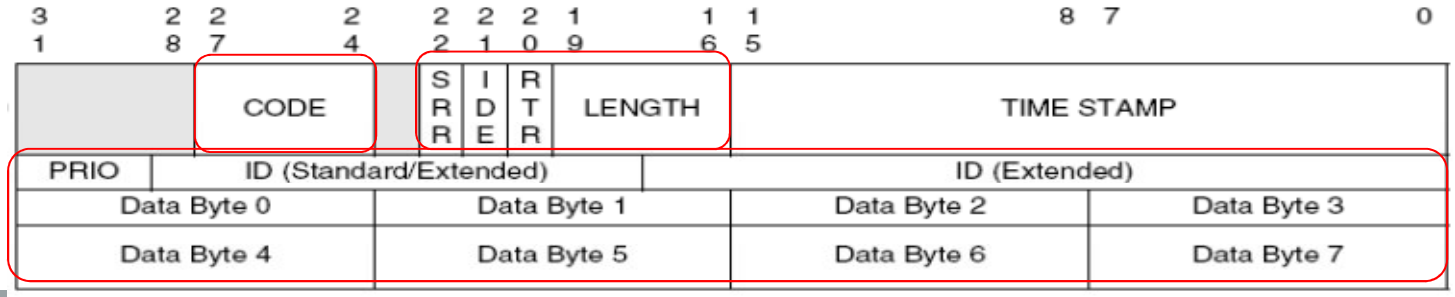
Priority Scheme

- Three methods:
  - Lowest number buffer first with highest priority
  - Lowest ID first with highest priority
  - Programmable 8-level local priority with the lowest value the highest priority

- When CTRL1[LBUF]= 1, the lowest number buffer is transmitted first.
- When CTRL1[LBUF] = 0 and CTRL1[LPRIO_EN] = 0, the MB with the lowest ID is transmitted first
- If CTRL1[LBUF] = 0 and CTRL1[LPRIO_EN] = 1, the PRIO bits augment the ID used during the arbitration process
- If two or more MBs have the same priority, the regular ID will determine the priority of transmission. If two or more MBs have the same priority (3 extra bits) and the same regular ID, the lowest MB will be transmitted first.

Transmit Process

1. If the MB is active (transmission pending), write an ABORT code (‘1001’) to the Code field of the Control and Status word to request an abortion of the transmission, then read back the Code field and the IFLAG register to check if the transmission was aborted. If backwards compatibility is desired (AEN in MCR = 0), just write ‘1000’ to the Code field to inactivate the MB but then the pending frame may be transmitted without notification
2. Write the ID word.
3. Write the data bytes.
4. Write the Length, Control and Code fields of the Control and Status word to activate the MB.



freescale™ semiconductor

Abort Transmission Process

- The abort mechanism must be explicitly enabled by asserting the MCR[AEN]
- The abort procedure can be summarized as follows:
  - CPU writes 1001 into the code field of the C/S word
  - CPU waits for the corresponding IFLAG indicating that the frame was either transmitted or aborted
  - CPU reads the CODE field to check if the frame was either transmitted (CODE=0b1000) or aborted (CODE=0b1001)
  - It is necessary to clear the corresponding IFLAG in order to allow the MB to be reconfigured

Code Example for Transmit Process

- With Inactivate code

```
FLEXCAN0_MBn_CS(i) = 0x08000000;          // write inactivate code to C/S
FLEXCAN0_MBn_ID(i) = ((i+1)<<18);         // write ID to be transmitted
FLEXCAN0_MBn_WORD0(i) = 0x12345678+i;     // write data bytes
FLEXCAN0_MBn_WORD1(i) = 0x11112222+i;
FLEXCAN0_MBn_CS(i) = 0x0C080000;          // write Tx code and data length to C/S
```

- With Abort code

```
FLEXCAN0_MBn_CS(i) = 0x09000000;          // write abort code to C/S
While(!FLEXCAN_IFLAG1 & (1<<i)) {};       // wait till abort or tx is done
FLEXCAN0_MBn_ID(i) = ((i+1)<<18);         // write ID to be transmitted
FLEXCAN0_MBn_WORD0(i) = 0x12345678+i;     // write data bytes
FLEXCAN0_MBn_WORD1(i) = 0x11112222+i;
FLEXCAN0_MBn_CS(i) = 0x0C080000;          // write Tx code and data length to C/S
```

Code Example for Accessing Rx FIFO

- Code snippet for configuring ID filter table elements

```
for(i = 0; i< iNoOfFilterTableElements;i++)
{

        FLEXCAN0_IDFLT_TAB(i) = (i+1L)<<19          //11msb of ID: 29 - 19

}
```

- Code snippet for reading message from Rx FIFO

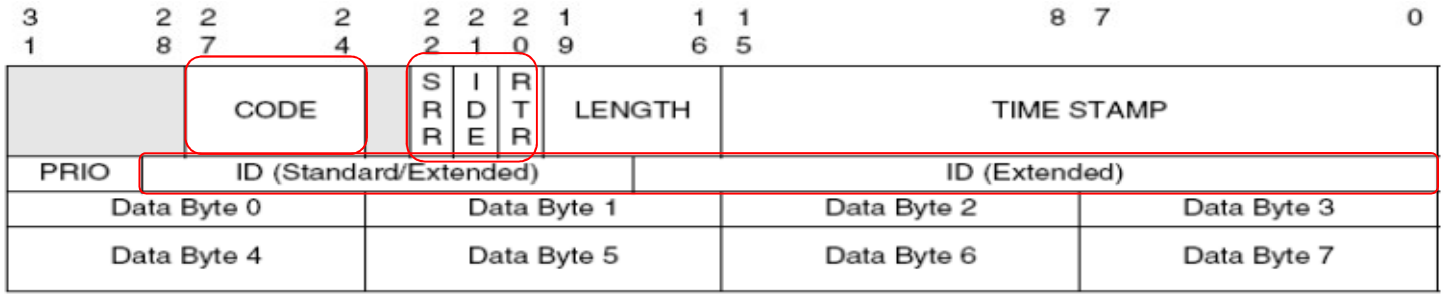
```
if(FLEXCAN0_IFLAG1 & (FLEXCAN_IFLAG1_BUF5I))     // check IFLAG1[5] bit
{
        // read FIFO
       cs= FLEXCAN0_MBn_CS(0);
       id= FLEXCAN0_MBn_ID(0);
       data0= FLEXCAN0_MBn_WORD0(0);
       data1= FLEXCAN0_MBn_WORD1(0);

        // read RXFIR to see which hit occurs
        iMatchHit = FLEXCAN0_RXFIR & 0x1FF;

        // clear flag to allow update of FIFO
        FLEXCAN0_IFLAG1 = FLEXCAN_IFLAG1_BUF5I;
}
```

Receive Process

- **Configure a MB as a Rx MB:**
  1. If the MB has a pending transmission, write an ABORT code (‘1001’) to the Code field of the Control and Status (C/S) word to request an abortion of the transmission, then read back the Code field and the IFLAG register to check if the transmission was aborted. If backwards compatibility is desired (AEN in MCR = 0), just write ‘1000’ to the Code field to inactivate the MB, but then the pending frame may be transmitted without notification. If the MB is already programmed as a receiver, just write ‘0000’ to the Code field of the Control and Status word to keep the MB inactive.
  2. Write the ID word
  3. Write ‘0100’ to the Code field of the Control and Status word to activate the MB



freescale™ semiconductor

Interrupt vs Poll for reading a message

- **Receive Interrupt:**
  1. Lock the MB by reading the Control and Status word (mandatory)
  2. Read the ID field (optional – needed only if a mask was used)
  3. Read the Data field
  4. Unlock the MB by reading the Free Running Timer (optional – releases the internal lock), or lock another MB
- Receive Poll:
  1. Check IFLAG1[MBn] bit until it is set
  2. Lock the MB by reading the Control and Status word (mandatory)
  3. Read the ID field (optional – needed only if a mask was used)
  4. Read the Data field
  5. Unlock the MB by reading the Free Running Timer (optional – releases the internal lock), or lock another MB
- Note:
  - CPU can receive self-transmitted messages if match. Or do not receive self-transmitted messages if MCR[SRX_DIS] = 1

Example code for configuring a MB as a Rx MB

- Configure a MB as a Rx MB

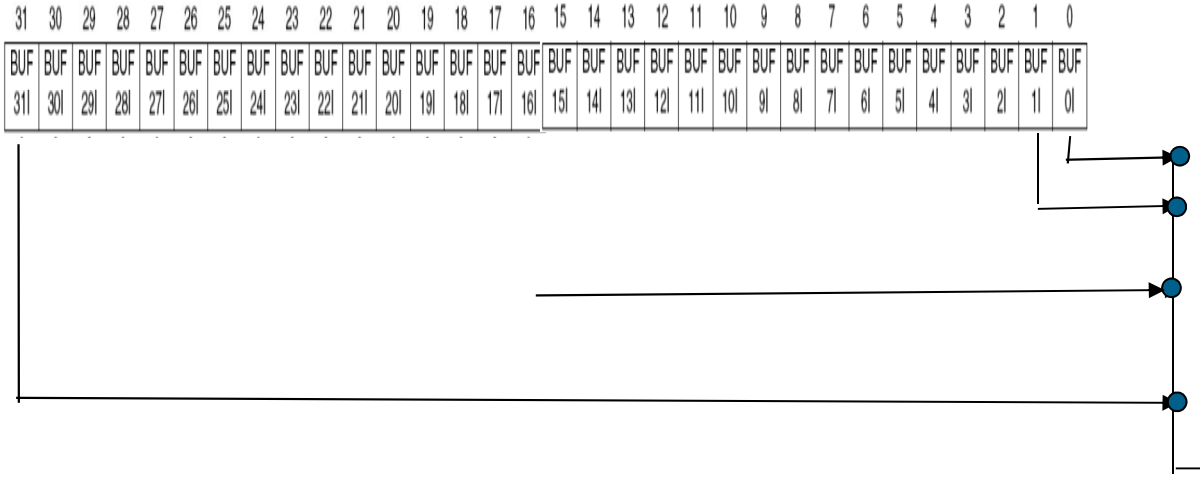
```
FLEXCAN0_MBn_CS(i) = 0x00000000;        // inactivate the MB
FLEXCAN0_MBn_ID(i) = ((i+1)<<18);       // write ID
FLEXCAN0_MBn_CS(i) = 0x04000000;        // write Rx code
```

- Check if a frame is received in a Rx MB and read the frame

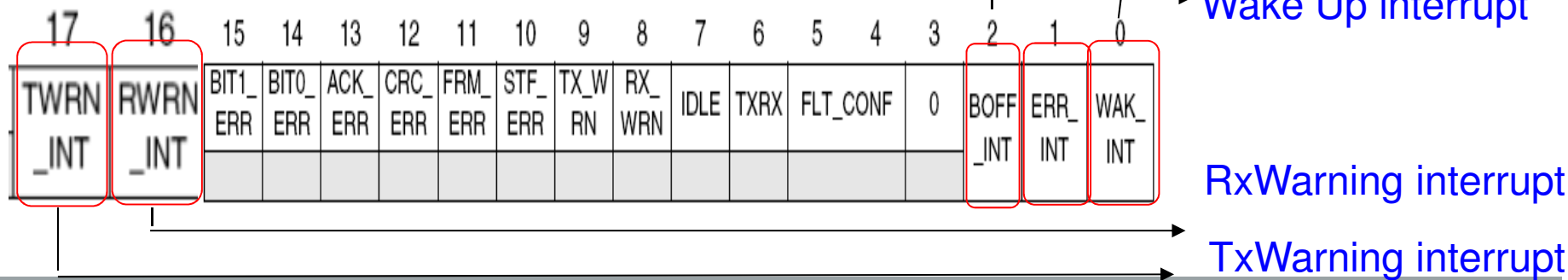
```
If(FLEXCAN0_IFLAG1 & (1<<i))
{
      cs = FLEXCAN0_MBn_CS(i);                  // lock the MB
      id = FLEXCAN0_MBn_ID(i);                  // read ID
      word0 = FLEXCAN0_MBn_WORD0(i);            // read data bytes
      word= FLEXCAN0_MBn_WORD1(i);
      timer= FLEXCAN0_TIMER;                    // unlock the MB
      FLEXCAN0_IFLAG1 = (1<<i);                 // must clear the flag
}
```

Interrupts

IFLAG1



ESR1



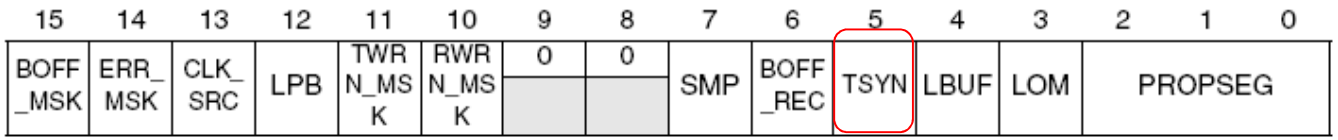
freescale semiconductor

Interrupts

- To enable the MB interrupt in FlexCAN module
  - IMASK1[MB] = 1
- To enable other interrupts
  - CTRL1[xx_MSK] = 1, MCR[WAK_MSK] = 1
  - xx_MSK:
    - BOFF_MSK, ERR_MSK, TWRN_MSK, RWRN_MSK

Global Timer Synchonization

- This feature provides means to synchronize multiple FlexCAN stations with a special “SYNC” message (i.e., global network time) which is required by high-level protocols such as CANopen.
- CTRL1[TSYN] enables this mechanism that resets the free-running timer each time a message is received in MB0 if FIFO is disabled or the first available MB (if FIFO is enabled).



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BOFF_MSK	ERR_MSK	CLK_SRC	LPB	TWRN_MSK	RWRN_MSK	0	0	SMP	BOFF_REC	TSYN	LBUF	LOM	PROPSEG		

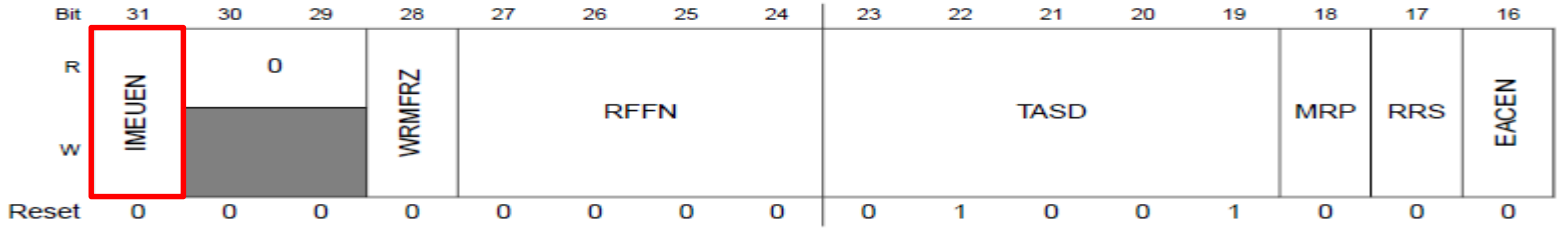
Control Register (CTRL1)

Safe Inactivation Process for a MB/FIFO

- Purpose:
  - Avoid loss of a received frame
  - Avoid the transmit of a frame without notice
- To perform a safe inactivation the CPU must use the following mechanisms along with the inactivation itself:
    - for Tx Mailboxes, the Transmission Abort;
    - for Rx Mailboxes, the IMEU (**Individual Matching Elements Update)**;

Safe Inactivation Process for a MB/FIFO

- **Individual Matching Elements Update**
  - Safe reconfiguration of both Mailbox and Rx FIFO individual matching elements in Normal Mode through the use of IMEUREQ and IMEUP bit fields in IMEUR register
  - Is enabled by setting CTRL2[IMEUEN] bit



CTRL2

- Individual matching elements
  - is every element used in the matching process which is associated to a specific Mailbox or Rx FIFO ID Filter Table element.
  - This includes the ID field, IDE bit and RTR bit of Mailboxes, Rx FIFO Identifier Acceptance Filters (ID filter table elements) and Individual Masks (RXIMRn).
  - It excludes RXMGMASK, RXFGMASK, RX14MASK and RX15MASK
  - For MBs,only one Individual Matching Element can be updated at a time
  - For FIFO Filter elements update, the reconfiguration is done every four

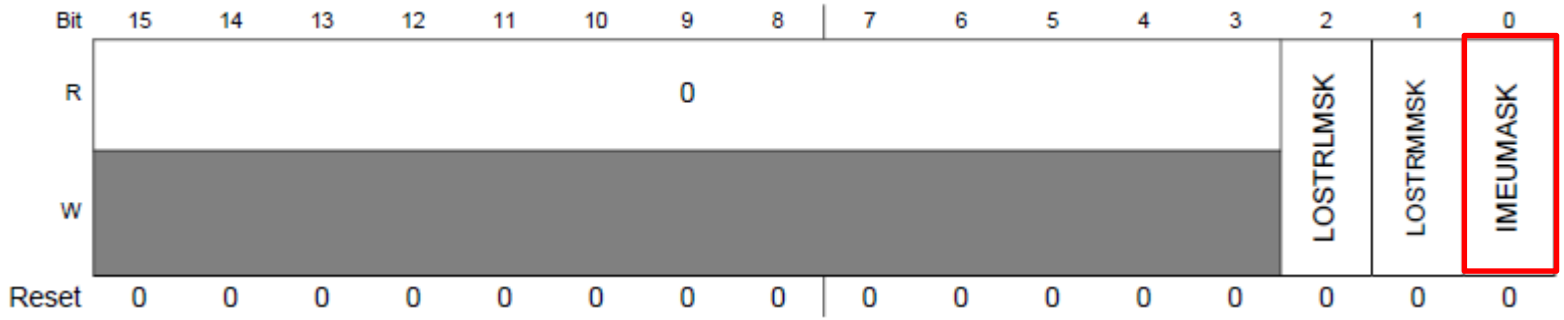
freescale™ semiconductor

Safe Inactivation Process for a MB/FIFO

- For polling, CPU must proceed as follows:
  1. check if the target element has its interrupt flag (IFLAG) is negated. If IFLAG is asserted, CPU must service this element and clear the corresponding IFLAG;
  2. wait for both IMEUR[IMEUREQ] and IMEUR[IMEUACK] bits to be negated;
  3. configure IMEUR[IMEUP] bits with the respective element address pointer;
  4. request IMEU by asserting IMEUR[IMEUREQ] bit;
  5. check IMEUR[IMEUREQ] and IMEUR[IMEUACK] bits, if both are asserted
     a) update the Mailbox/RxFIFO matching elements;
     b) negate IMEUR[IMEUREQ] bit;
  6. it is recommended to clear ESR2[IMEUF] flag to keep the coherence with the corresponding IMEUR[IMEUREQ] and IMEUR[IMEUACK] assertion;

Safe Inactivation Process for a MB/FIFO

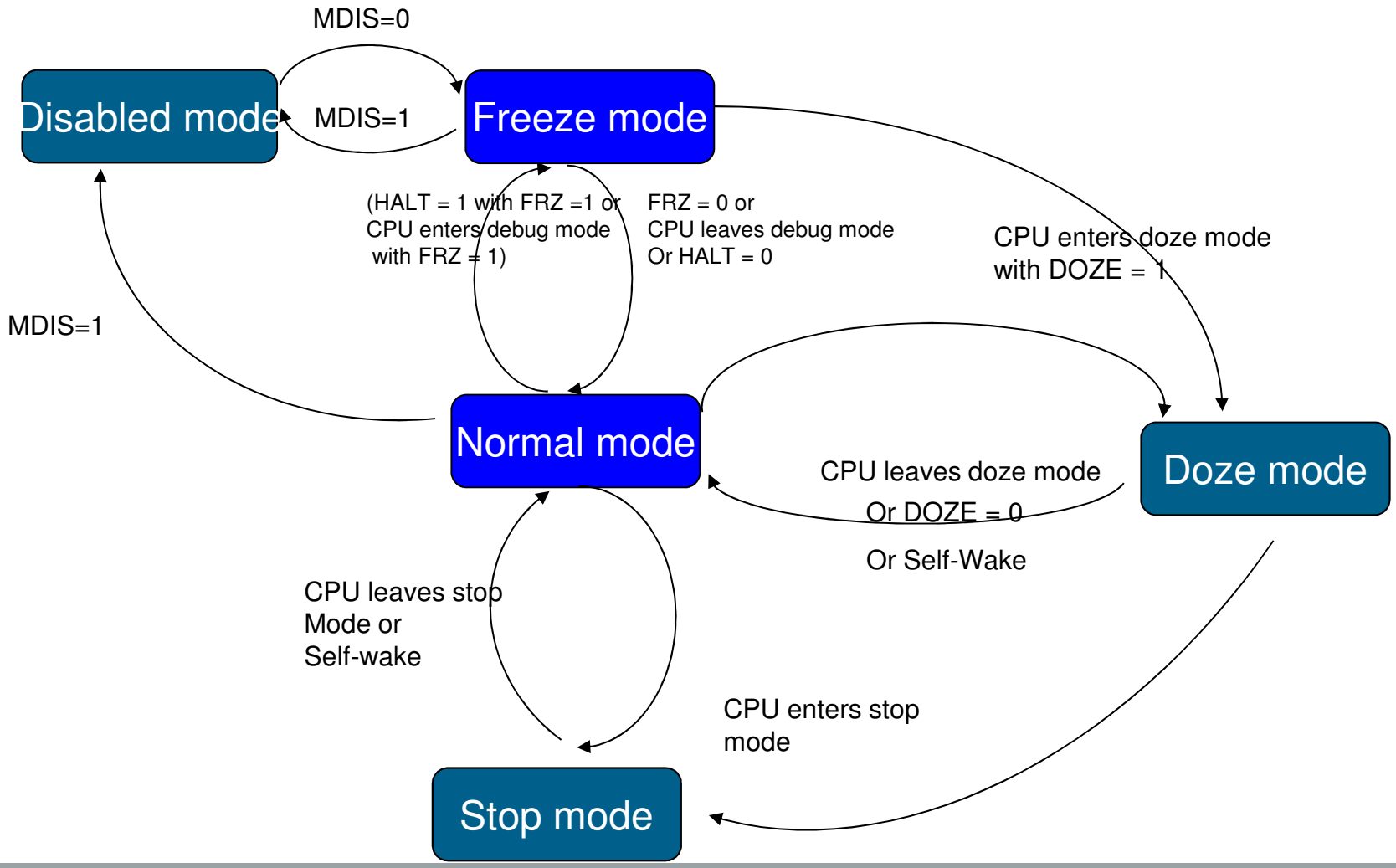
- For interrupt, CPU must proceed as follows:
  1. check if the target element has its interrupt flag (IFLAG) is negated. If IFLAG is asserted, CPU must service this element and clear the corresponding IFLAG;
  2. wait for both IMEUR[IMEUREQ] and IMEUR[IMEUACK] bits to be negated;
  3. configure IMEUR[IMEUP] bits with the respective element address pointer;
  4. request IMEU by asserting IMEUR[IMEUREQ] bit;
  5. wait for either IMEU Interrupt or ESR2[IMEUF] bit assertion. ESR2[IMEUF] is asserted when IMEUR[IMEUACK] is asserted;
     a) update the Mailbox/RxFIFO matching elements;
     b) negate IMEUR[IMEUREQ]bit;
  6. clear ESR2[IMEUF] flag;
- To enable IMEU interrupt, set CTRL2[IMEUMASK] bit:



CTRL2

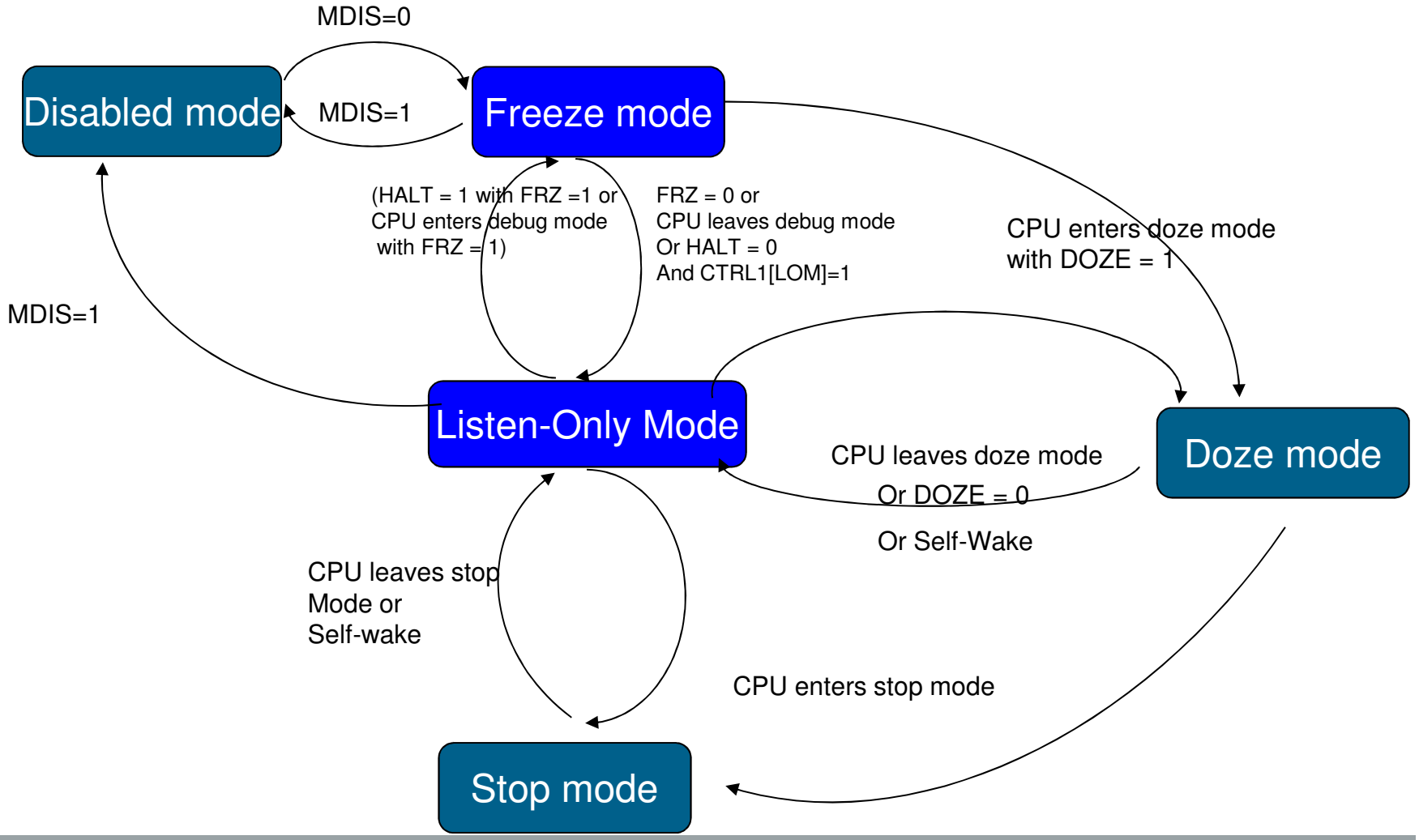
freescale™ semiconductor

Modes of operation overview



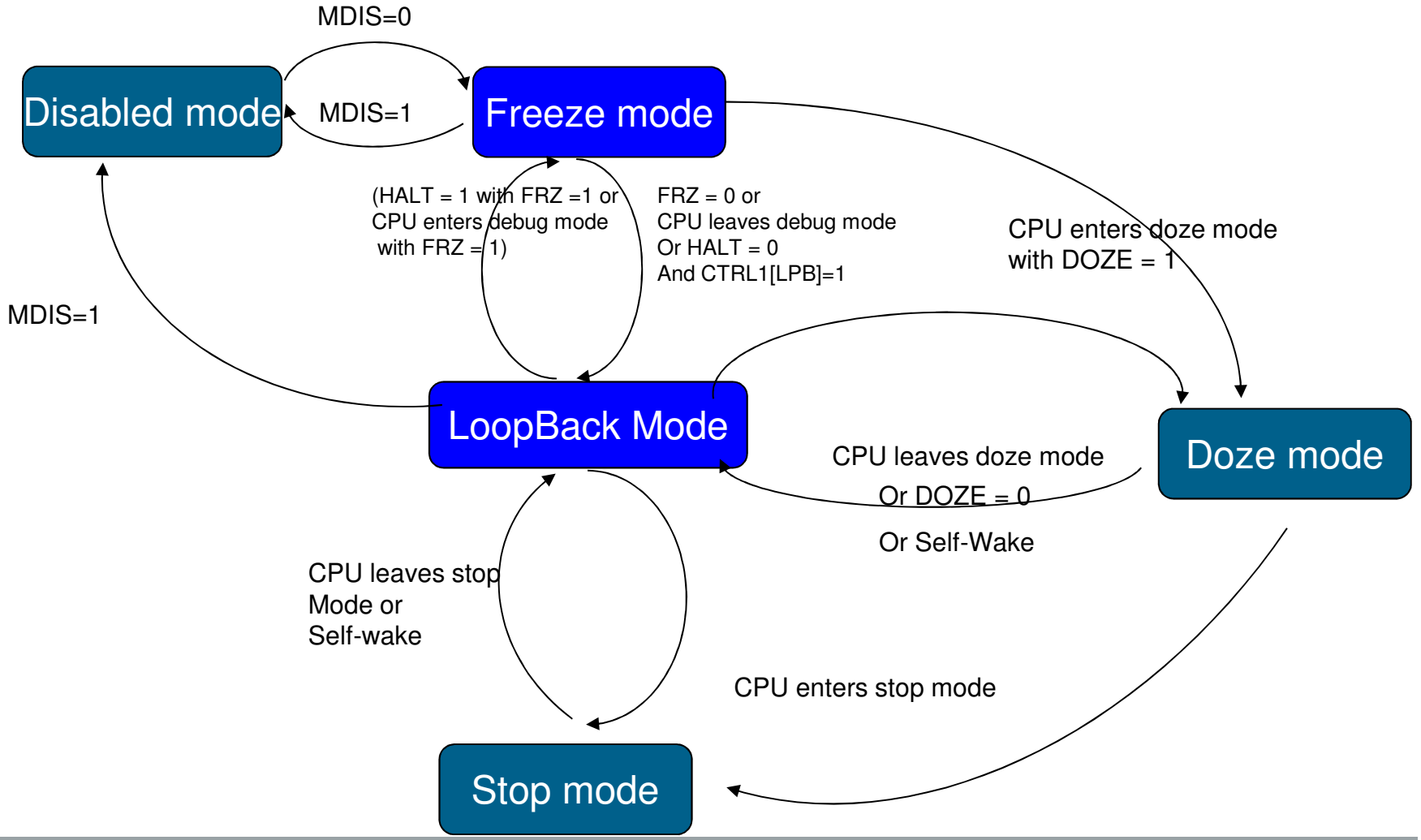
freescale™ semiconductor

Modes of operation overview



freescale™ semiconductor

Modes of operation overview



freescale™ semiconductor

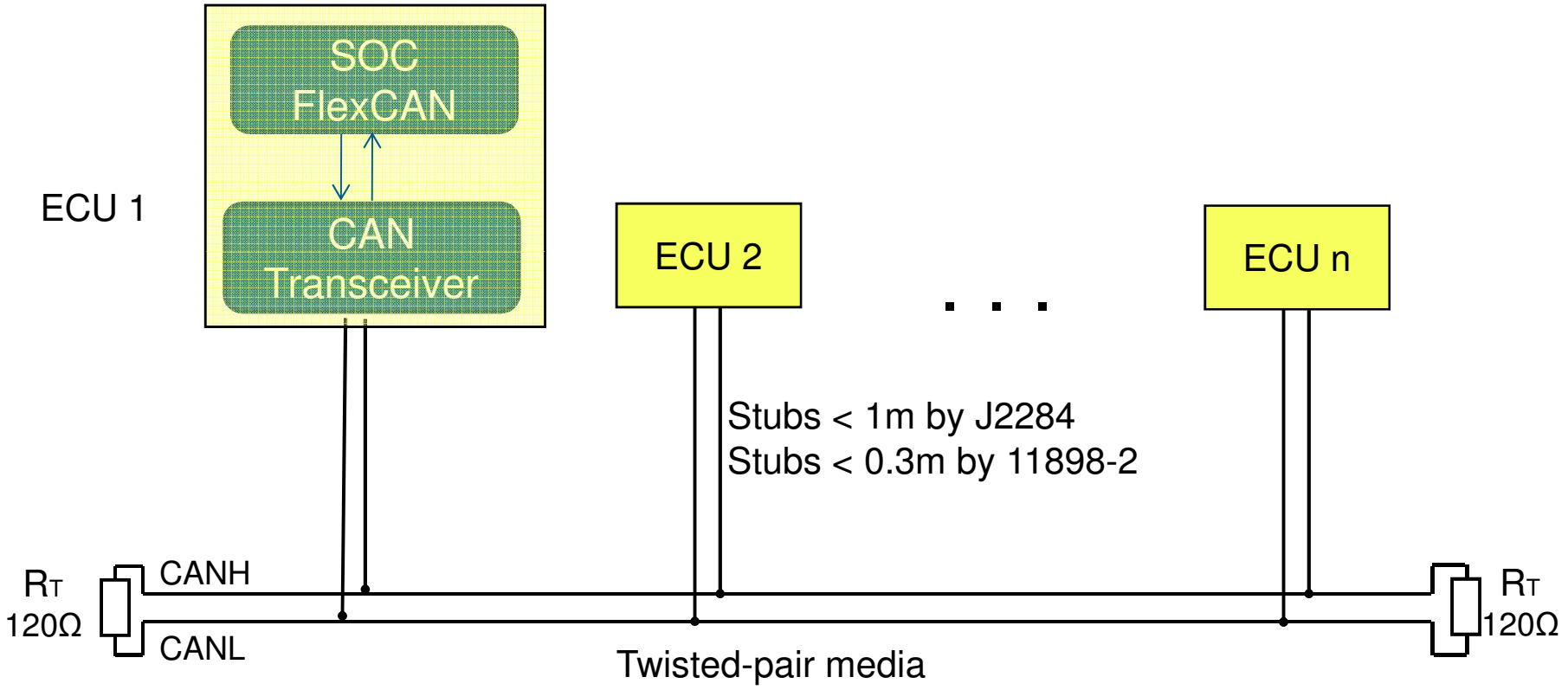
FlexCAN Training Outline

1. Module Overview
2. On-chip interconnects and inter-module dependencies
3. Software configuration
4. **Hardware configuration/considerations**
5. Demo code explanation
6. Frequently asked question list
7. Reference material

freescale™ semiconductor

Hardware Configuration

- *Both CAN_TX and CAN_RX pins can be directly connected to CAN transceiver*
- *Typical High Speed CAN-C connection as below*



freescale™ semiconductor

FlexCAN Training Outline

1. Module Overview
2. On-chip interconnects and inter-module dependencies
3. Software configuration
4. Hardware configuration/considerations
5. Demo code explanation
6. Frequently asked question list
7. Reference material

FlexCAN Training Outline

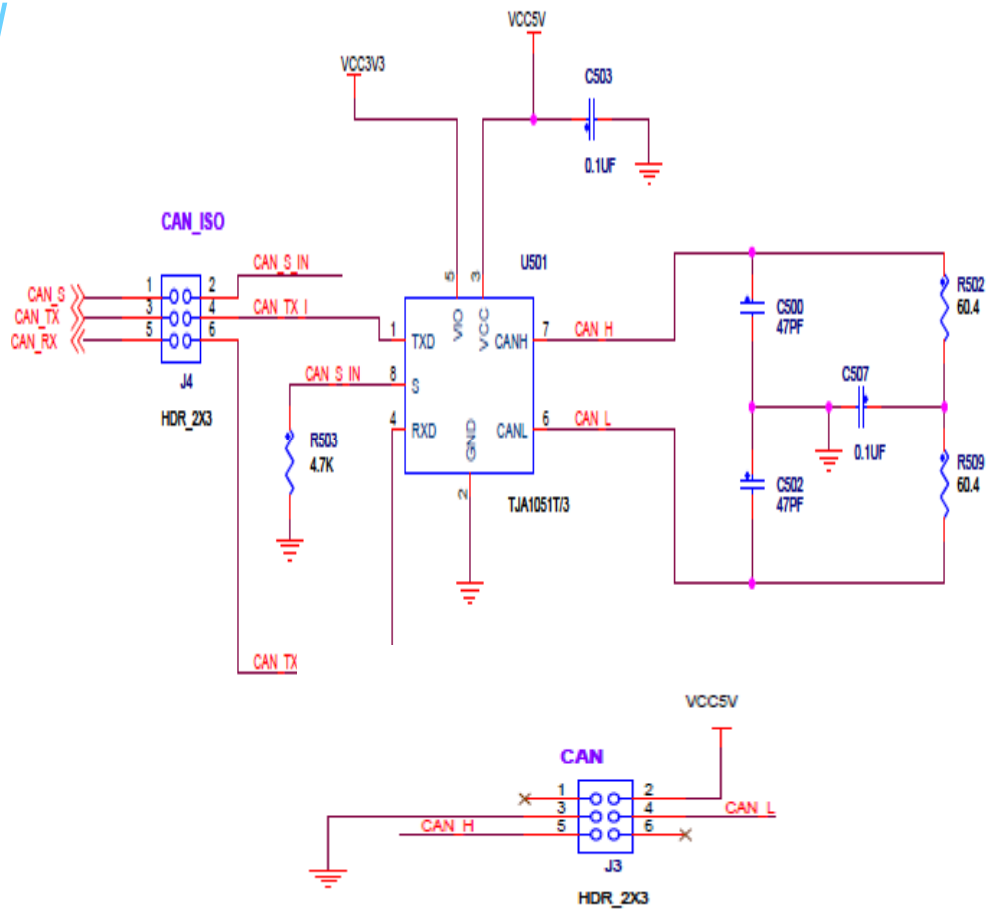
1. Module Overview
2. On-chip interconnects and inter-module dependencies
3. Software configuration
4. Hardware configuration/considerations
5. **Demo code explanation**
6. Frequently asked question list
7. Reference material

Hands on Demo

- ***SCI2CAN bridge demo***
  - *a character entered in Terminal will be transmitted to remote CAN node which echoes it to the local CAN node that will display it in the Terminal*
  - *call CAN driver to do FlexCAN initalization, configure acceptance filters, prepare transmit , receive, and read messages*
  - *can run in loopback mode or with external node*
  - *Terminal is used to input a character and output a character received*
  - *Baud rate of UART is 115200bps with 8-N-1 format*
  -
- *RxFIFO demo*
  - *call CAN driver to set up 8 ID filter table elements and 1 MB*
  - *send 9 messages*
  - *receive 9 messages: 8 messages from RxFIFO, 1 from MB*

Board Configuration

- *CAN1 pins PTE24/25 are connected out of tower board to CAN transceiver*
- *Shorten pin 3 & 4, 5 & 6 of*
- *HDR_2X3 on TWR-SER2*
- *Board*
- *NOTE:*
- Because the TXD dominant time-out time on TJA1051T, the minimum possible bit rate of **40 kbit/s**.



freescale semiconductor

FlexCAN Training Outline

1. Module Overview
2. On-chip interconnects and inter-module dependencies
3. Software configuration
4. Hardware configuration/considerations
5. Demo code explanation
6. **Frequently asked question list**
7. Reference material

freescale™ semiconductor

FlexCAN Preliminary FAQ

- **Q: Can FlexCAN automatically respond to remote request frame ?**
  A: Yes, FlexCAN can automatically respond to remote request frame with either a data frame or remote request frame. This reduces CPU intervention.

- **Q: Can FlexCAN receive self-transmitted frames and how can we disable it?**
  A: Yes, FlexCAN can receive self-transmitted frame. To disable this feature, set MCR[SRX_DIS] bit

- **Q: How many Message Buffers supported by Kinetis?**
  A: Kinetis supports up to 16 message buffers (MBs) and 16 individual masking registers (RXIMR).

- **Q: How many Rx FIFO filter table elements does FlexCAN support?**
  A: It is up to CTRL2[RFFN] bit field. The maximum value of RFFN for Kinetis is 4, so it supports up to 40 Rx FIFO filter table elements.

- **Q: How can I determine when a frame is received? How to start a frame transmission?**
- **A: You can determine when a frame is received either by polling the IFLAG[MBn] bit or by using MB interrupt. To start a frame transmission, a MB must be configured as a Tx MB and follow the Transmit Process.**

FlexCAN Training Outline

1. Module Overview
2. On-chip interconnects and inter-module dependencies
3. Software configuration
4. Hardware configuration/considerations
5. Demo code explanation
6. Frequently asked question list
7. **Reference material**

freescale™ semiconductor

FlexCAN References

Application Notes:

- **AN1798**/D - CAN Bit Timing Requirements
- **AN3690**/D - CAN2USB bridge
- **AN1828**/D - Flash Programming via CAN
- **AN1776**/D - Stereo Audio Transmission with TouCAN

freescale™ semiconductor


freescale™
semiconductor

Kinetis Rx FIFO Filters vs corresponding Masks

CTRL2[RFFN]	# of Rx FIFO Filters	MBs occupied by Rx FIFO & ID Filter Table	MBs available	RxFIFO ID Filter Table Elements affected by Rx Individual Mask (RXIMRn)	RxFIFO ID Filter Table Elements affected by Rx FIFO Global Mask (RXFGMASK)
0	8	MB0 – 7	MB8-15	Elements 0 - 7	none
1	16	MB0 – 9	MB10 – 15	Elements 0 – 9	Elements 10 – 15
2	24	MB0 – 11	MB12 – 15	Elements 0 – 11	Elements 12 – 15
3	32	MB0 – 13	MB14 – 15	Elements 0 – 13	Elements 14 – 15
4	40	MB0 – 15	None	Elements 0 - 15	None

NOTE:

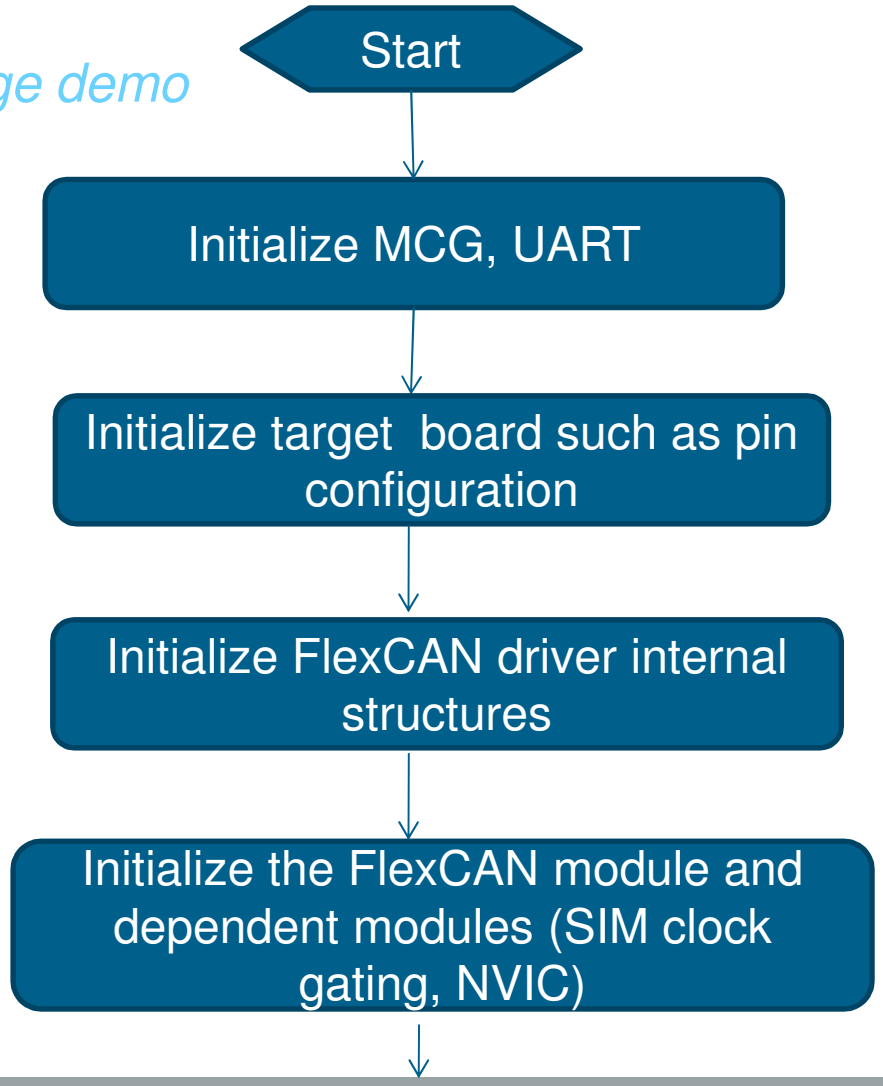
- RXIMRn where n is the Elements #
- For CTRL2[RFFN]=0, RXIMR0 – 7 applies to Elements 0 – 7
- For CTRL2[RFFN]=1, RXIMR0 – 9 applies to Elements 0 – 9
- …

Matching Architecture

Structure	Rx SMB [RTR]	CTRL2[RSS]	CTRL2[EACEN]	MB[IDE]	MB[RTR]	MB[ID]	MB[CODE]	Comments
MB	0	-	0	Cmp	No_cmp	Cmp_msk	Empty /FULL/OVERRUN	Data frame
MB	0	-	1	Cmp_msk	Cmp_msk	Cmp_msk	Empty /FULL/OVERRUN	Data frame
MB	1	0	-	Cmp	No_cmp	Cmp	RANSWER	Remote frame
MB	1	1	0	Cmp	No_cmp	Cmp_msk	Empty /FULL/OVERRUN	Remote frame
MB	1	1	1	Cmp_msk	Cmp_msk	Cmp_msk	Empty /FULL/OVERRUN	Data frames
FIFO	-	-	-	Cmp_msk	Cmp_msk	Cmp_msk	-	Rx FIFO Table elements

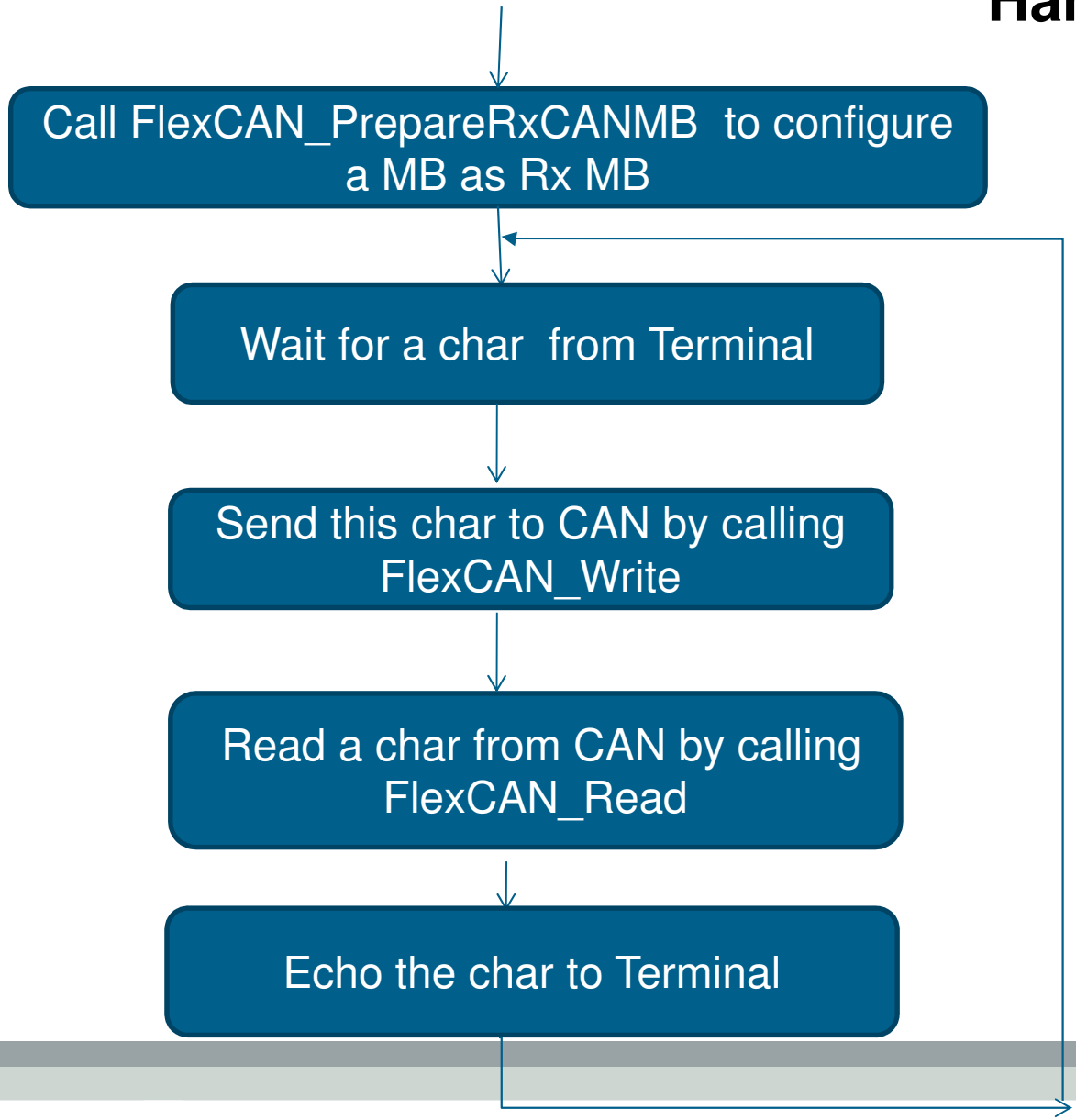
Hands on Demo

- *SCI2CAN bridge demo*



freescale™ semiconductor

Hands on Demo



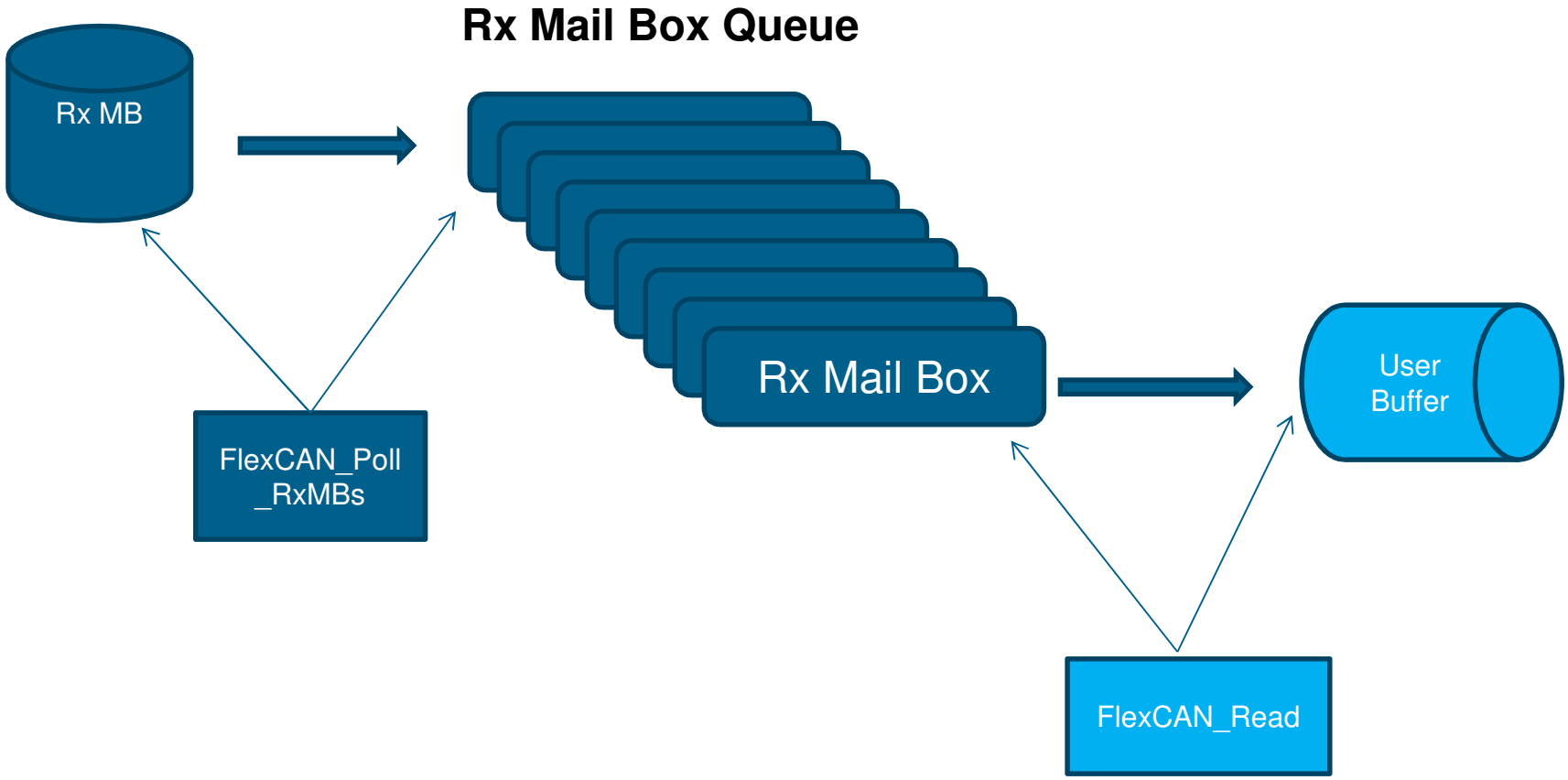
freescale™ semiconductor

CAN Driver Structure

```
// mail box
typedef volatile struct {
  uint8       dev_num;                  /*  FlexCAN device number */
  uint16      mailbox_number;           /*  mailbox number */
  uint32      identifier;               /*  message ID */
  uint8       format;                   /*  mailbox format (FLEXCAN_STANDARD OR
FLEXCAN_EXTENDED) */
  uint8       direction;                /*  transmission or reception direction (FLEXCAN_TX or
FLEXCAN_RX) */
  uint8       remote_req_flag;          /*  is remote request?   */
  uint8       data_len;                 /*  number of bytes to write to or read from the mailbox (0 to
8) */
  uint8       data[8];                  /*  data bytes */
  uint8       code;                     /* control code */
  uint8       priority;                 /*  priority of the message in the mailbox */
  uint16      timestamp;                /*  timestamp */
  uint16      crc;                      /*  crc of the message frame */
  uint32      imask;                    /* individual mask */
 } TFCAN_MailBox, *PTFCAN_MailBox;
```

Note:
The identifier field is used to indicate the ID of a message and also includes indication of frame format (standard or extended format) and type (data frame or remote frame):
If an ID with CAN_MSG_IDE_MASK is 1 (bit 31 = 1), it is extended format; otherwise, it is standard format.
If an ID with CAN_MSG_TYPE_MASK is 1 (bit 30 = 1), it is remote frame; otherwise, it is data frame.

CAN Driver Structure

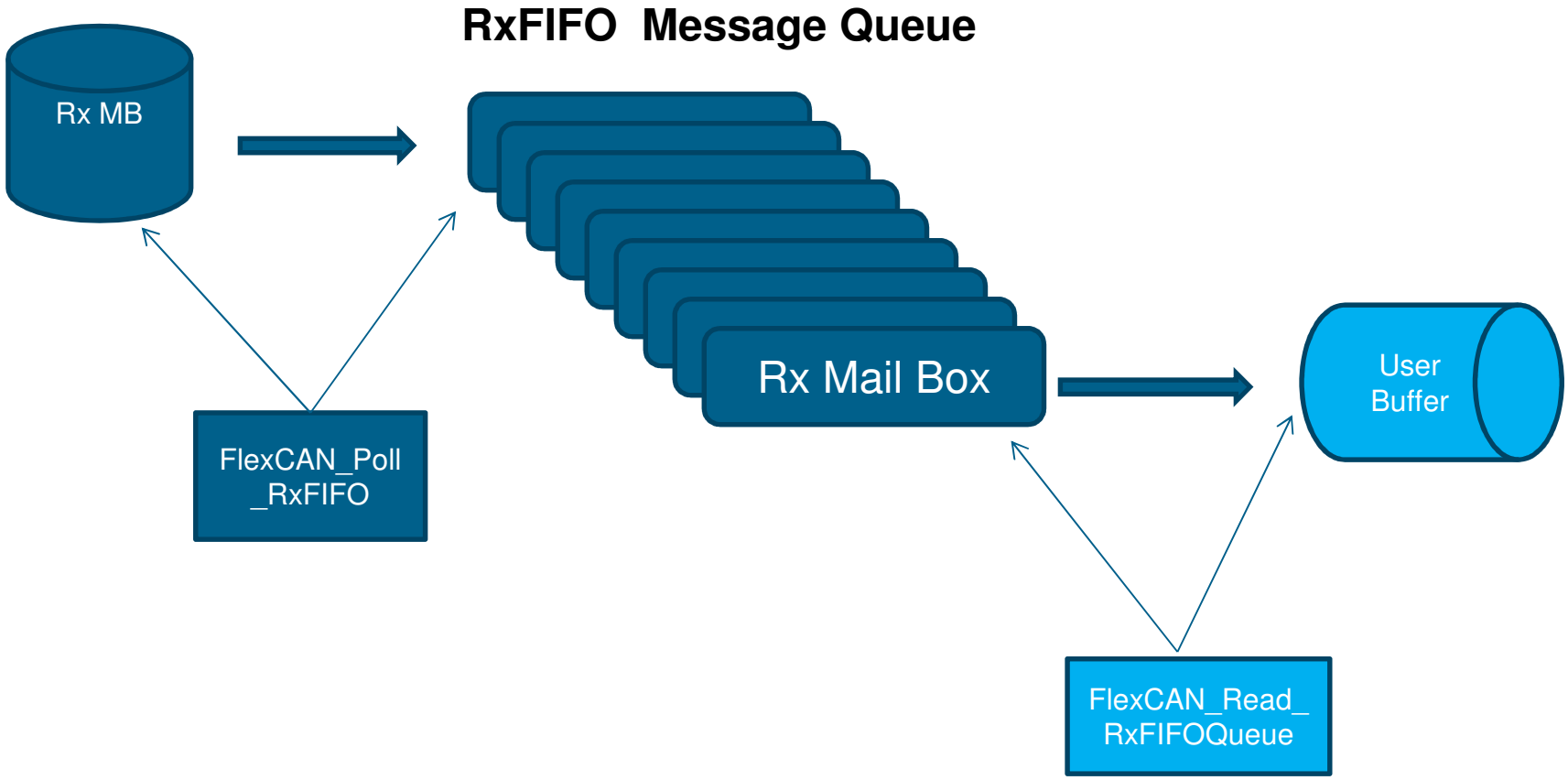


NOTE:

- Driver calls FlexCAN_PollRxMBs to queue messages read from a Rx MB to Rx Mail Box queue \
- Application calls FlexCAN_Read to deque messages from Rx Mail Box queue to user's buffer

freescale™ semiconductor

CAN Driver Structure

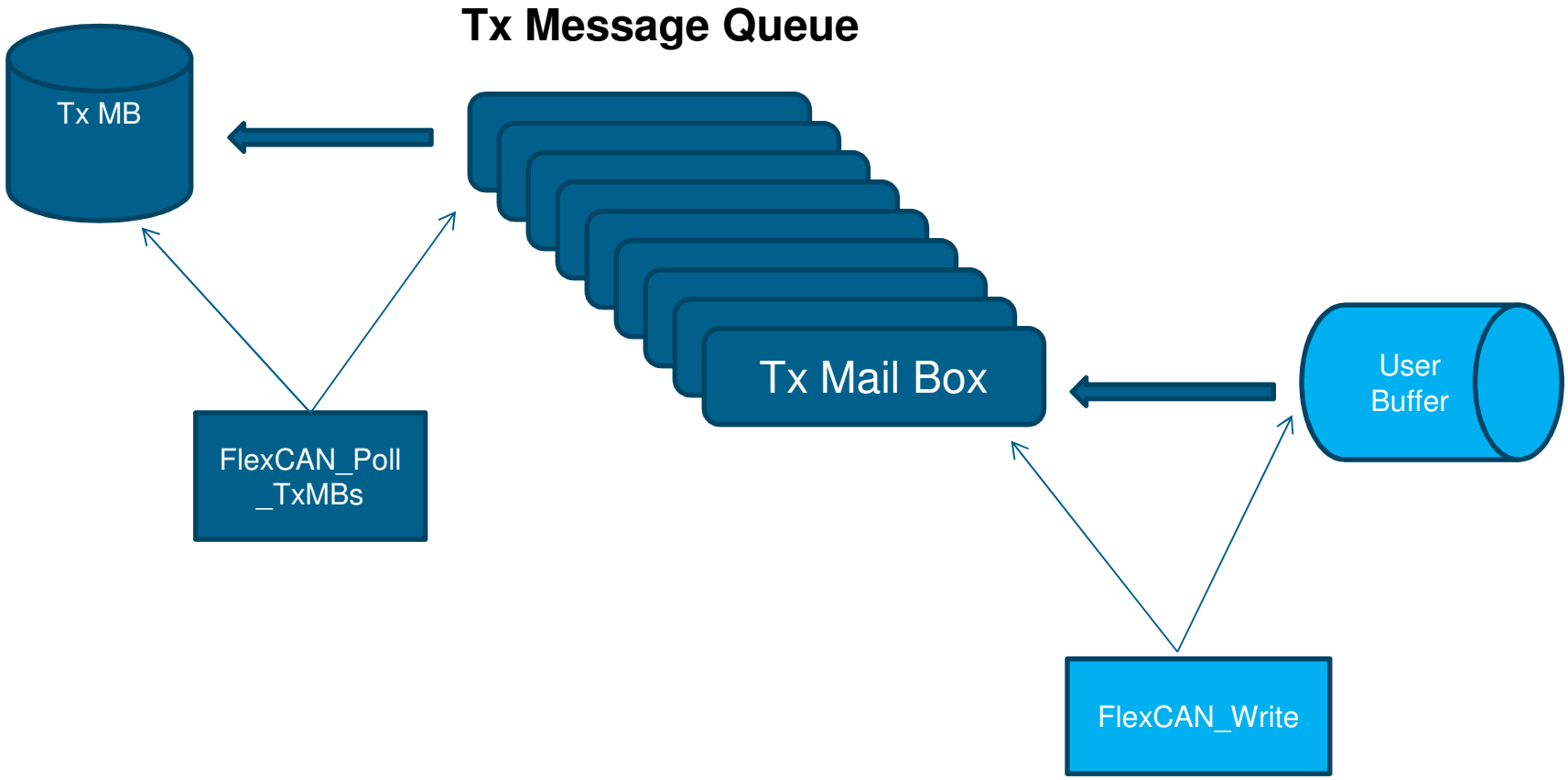


NOTE:

- Driver calls FlexCAN_PollRxFIFO to queue messages read from a Rx MB to Rx FIFO message queue \
- Application calls FlexCAN_ReadRxFIFOQueue to deque messages from Rx FIFO message queue to user's buffer

freescale™ semiconductor

CAN Driver Structure



NOTE:

- Application calls FlexCAN_Write to queue messages from user's buffer to the Tx Mail Box queue for transmit
- Driver calls FlexCAN_PollRxMBs to deque messages from Tx message queue to the Tx MB for transmit

freescale™ semiconductor

Custom CAN Parameters

- The values of the following macros can be redefined to specify CAN parameters suitable for applications in can_config.h:

```
// Definition of FlexCAN control bits
#define FLEXCAN_CLOCK_SOURCE      1       /* clock source for FlexCAN: 0 -- external oscillator, 1 -- bus
clock from PLL */
#define FLEXCAN_SELF_RECEPTION    0       /* enable self-reception: 0 - disable self-reception; 1 --
enable self-reception */
#define FLEXCAN_SCAN_PRIORITY     0       /* scan prio: 0 -- RxFIFO first, 1 -- MB first */
#define FLEXCAN_STORE_RTR         0       /* remote request storing: 0 -- do not store RTR, 1 -- store RTR
as data frame */
#define FLEXCAN_ENTIRE_ARB_CMP    0       /* entire frame arbitration filed comparision including
IDE,RTR, excluding RxFIFO
                                  * 0 -- always compare ID of Rx MB regardless of masks, but not compare RTR
                                  * 1 -- mask compare both Rx MB ID and RTR
                                  */
#define FLEXCAN_INDIVIDUAL_MASK   0       /* individual Rx Masking and queue enable:
                                  * 0 -- message queue is enabled, and masking with
RXMGMASK/RX14MASK/RX15MASK,RXFGMASK
                                  * 1 -- message queue is disabled, and individual masking with RXIMRn
                                  */

#define FLEXCAN_LOCAL_PRIO        0       /* local priority enable: 0 -- disabled, 1 -- enabled */
#define FLEXCAN_LOOP_BACK         0       /* loop back for single node: 0 -- no loopback, 1 -- loopback */
/* NOTE: if use LOOPBACK, self-reception must be enabled by defining
 * #define FLEXCAN_SELF_RECEPTION  1
 */
```

Custom CAN Parameters

```
#define FLEXCAN_NO_RXFIFO_FILTERS 8       /* # of Rx FIFO filters: 0 -- Rx FIFO disabled, or
8/16/24/32/40 */
#define FLEXCAN_ID_TAB_FORMAT     0       /* Rx FIFO ID table format: 0 -- Format A, 1 -- Format B, 2 --
Format C, 3 -- Format D */

// Configuration of MBs
#define NUMBER_OF_RX_MB                   12                    /* # of Receive MBs, can
be changed per application */
#define NUMBER_OF_TX_MB                   (NUMBER_OF_MB-NUMBER_OF_RX_MB)
                /* # of MBs are  for Transmit MBs */

// Definition of Receive MBs index
#define         FLEXCAN_RX_MB_START             8 //0              /* start # of Rx MB */
#define         FLEXCAN_RX_MB_END   8 //(NUMBER_OF_RX_MB-1)       /* end # of Rx MB */

// Definition of Transmit MBs index
#define FLEXCAN_TX_MB_START       9 //(FLEXCAN_RX_MB_END+1)
#define         FLEXCAN_TX_MB_END   (NUMBER_OF_MB-1)

// Definition of max # of custom mails
#define MAX_RX_MAILS     20
#define MAX_TX_MAILS      20
#define MAX_RXFIFO_MAILS  10
```