Embedded Systems

CS 397

TRIMESTER 3, AY 2021/22

Controller Area Network (CAN) (1/3)

[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

Introduction

- BOSCH (www.bosch.com) developed the Controller Area Network (CAN) bus in the late 1980, and the CAN specification version 2.0 (parts A and B) was released in Sep 1991.

- CAN is a multicast, shared serial bus standard for connecting electronic control units.

- CAN bus is a multi-master message broadcast system, and in a CAN network, many small blocks of data (short messages) are broadcasted to the entire network.

- CAN is also an International Standardization Organization (ISO) defined serial communications bus originally developed for the automotive industry to replace the complex wiring harness with a two-wire bus.

- The CAN specification calls for high immunity to electrical interference and the ability to self-diagnose and repair data errors.

- These features have led to CAN's popularity in a variety of industries including building automation, medical, and manufacturing.

Introduction: a Brief History of CAN

1983	Start of the Bosch internal project to develop an in-vehicle network
1986	Official introduction of the CAN protocol
1987	First CAN controller chips available from Intel and Philips Semiconductor
1991	Bosch publishes CAN specification 2.0
1992	CAN in Automation (CiA) established as the international users and manufacturers group
1992	CAN Application Layer (CAL) protocol published by CiA
1992	First cars by Mercedes-Benz are being equipped with CAN
1993	ISO 11898 standard published
1994	First International CAN Conference (iCC) organized by CiA
1994	DeviceNet protocol introduced by Allen-Bradley
1995	ISO 11898 amendment (extended frame format) published
1995	CANopen protocol published by CiA

Source: A Comprehensible Guide to Controller Area Network

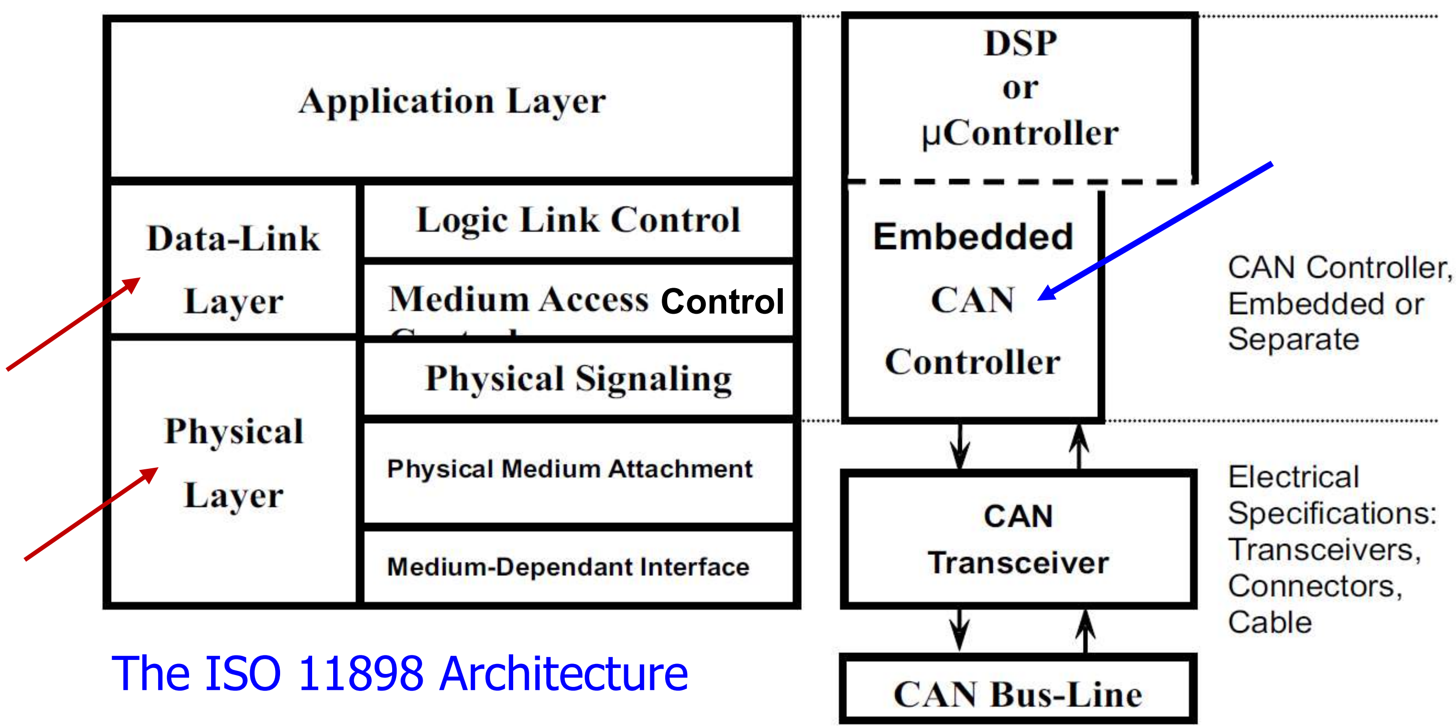
https://copperhilltech.com/

The CAN Standard

ISO: International Standardization Organization

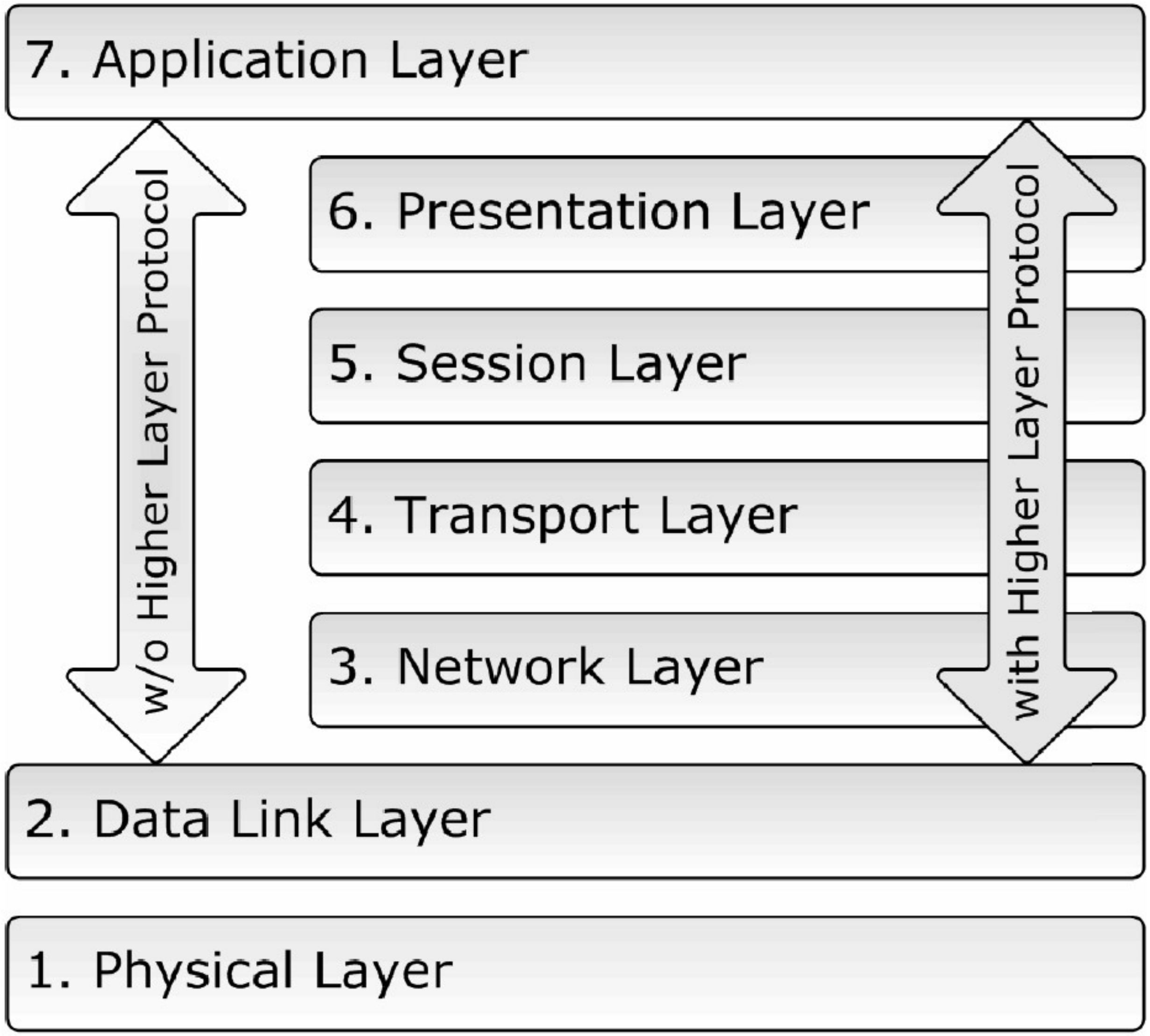
- The CAN communications protocol, ISO 11898: 2003, describes information passing between devices on a network and conforms to the Open Systems Interconnection (OSI) model, which is defined in terms of layers.

- Actual communication between devices connected by the physical medium is defined by the physical layer of the model. The ISO 11898 architecture defines the lowest two layers of the seven layer ISO/OSI model as the data-link layer and physical layer.



The ISO 11898 Architecture

The CAN Standard: ISO/OSI 7 Layer Reference Model



Source: A Comprehensible Guide to Controller Area Network

https://copperhilltech.com/

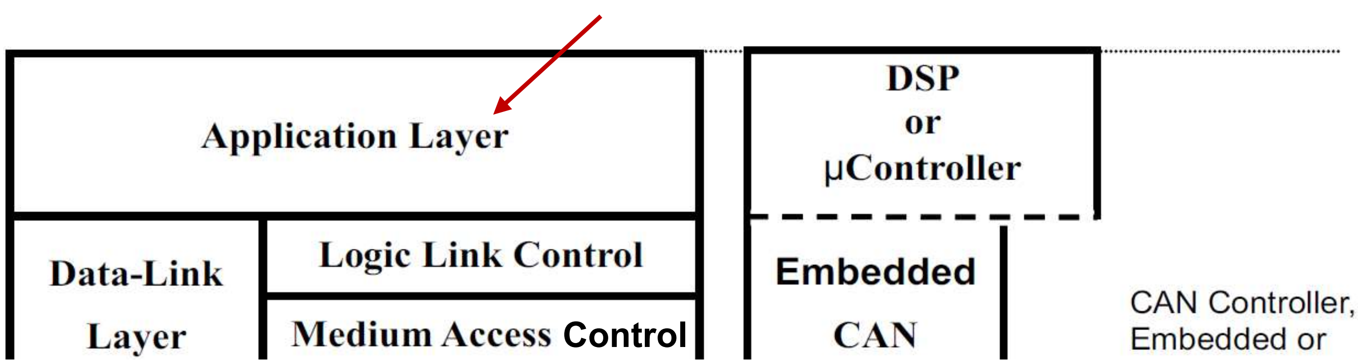
The CAN Standard

- The application layer (see figure) sets up the communication link to an application specific protocol such as the vendor-independent CANopen protocol.

- https://en.wikipedia.org/wiki/CANopen

- This CANopen protocol is supported by the international users and manufacturers group, CAN in Automation (CiA). Additional CAN information can be found at the CiA Web site, https://www.can-cia.org/

- Many protocols (CAN higher-layer protocols) are dedicated to applications like industrial automation, aviation industry, and automotive in-vehicle network.



The CAN Lower-Layer Standards

CAN is internationally standardized in **ISO 11898**. This standard has seen several revisions. The current CAN protocol standards include:

- ISO 11898-1:2015 – Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signaling

- ISO 11898-2:2016 – Road vehicles – Controller area network (CAN) – Part 2: High-speed medium access unit

- ISO 11898-3:2006 – Road vehicles – Controller area network (CAN) – Part 3: Low-speed, fault-tolerant, medium-dependent interface

- ISO 11898-3/Cor1:2006 provides a replacement for Figure 9 on page 17.

- ISO 11898-4:2004 – Road vehicles – Controller area network (CAN) – Part 4: Time-triggered communication

https://blog.ansi.org/2017/02/controller-area-network-can-standards-iso-11898/#gref

All ISO 11898 series standards for the CAN protocol are available on the ANSI (American National Standards Institute) webstore.

The CAN Higher-Layer Protocols

https://en.wikipedia.org/wiki/CAN_bus

The CAN standard does not include tasks of the application layer protocols. Each car manufacturer therefore created its own standard, the CAN higher-layer protocols, which include the following standardized approaches:

- ARINC 812 or ARINC 825 (aviation industry)
- CANopen - EN 50325-4 (industrial automation)
- DeviceNet (industrial automation)
- EnergyBus - CiA 454 (light electrical vehicles)
- ISOBUS - ISO 11783 (agriculture)
- ISO-TP - ISO 15765-2 (transport protocol for automotive diagnostics)
- MilCAN (military vehicles)
- NMEA 2000 - IEC 61162-3 (marine industry)
- SAE J1939 (in-vehicle network for buses and trucks)
- SAE J2284 (in-vehicle networks for passenger cars)
- Unified Diagnostic Services (UDS) - ISO 14229 (automotive diagnostics)

https://en.wikipedia.org/wiki/SAE_J1939

The CAN Higher-Layer Protocols

Other approaches of the CAN higher-layer protocols are not limited to:

- CANaerospace - Stock (for the aviation industry)
- CAN Kingdom - Kvaser (embedded control system)
- CCP/XCP (automotive ECU calibration)
- GMLAN - General Motors (for General Motors)
- RV-C - RVIA (used for recreational vehicles)
- SafetyBUS p - Pilz (used for industrial automation)
- UAVCAN (aerospace and robotics)
- CSP (CubeSat Space Protocol)

  https://en.wikipedia.org/wiki/Cubesat_Space_Protocol

The CAN Protocol Standards – Example of Specifications

Standard	Common Name	Baud Rate	Max nodes	Max Length
ISO 11783	ISOBUS	250 KBit/s	30	40m
ISO 11898-2	High speed-CAN	max. 1 MBit/s	110	6500 m
ISO 11898-2 2015	CAN FD	max.12 MBit/s ?	110	10 m
ISO 11898-3	Fault Tolerant CAN	max. 125 KBit/s	32	500 m
ISO 11992	Truck/Trailer CAN	max. 125 KBit/s		40 m
ISO 15765	Diagnostics On CAN	max 1 MBit/s	110	
SAE J1939	J1939	250 KBit/s	30 ECUs	40m
SAE J2284		max. 1 MBit/s	110	
	Single Wire CAN	33,3 KBit/s	32	

http://www.computer-solutions.co.uk/info/Embedded_tutorials/can_tutorial.htm

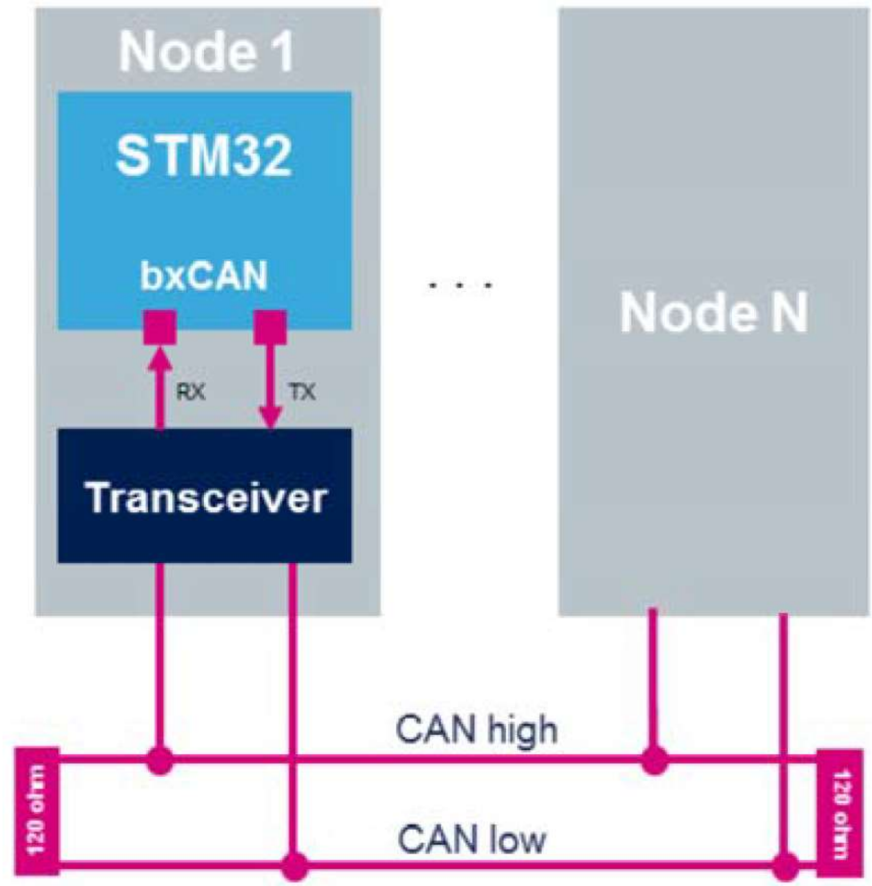
The STMicroelectronics' STM32F7 bxCAN (Basic Extended Controller Area Network)

- The bxCAN is a standard serial differential bus, allowing the microcontroller to communicate with external devices connected to the same network bus.

- The bxCAN interface is highly configurable, allowing each node to be easily connected using two wires via an external CAN transceiver.

Application benefits

- multi-master concept with message priority,
- object-oriented communication (no node addressing, but content identification),
- real-time capability with low message transfer latency, and
- system wide message consistency (error detection & management mechanism).



An example of bxCAN Network

The bxCAN Main Features (1/2)

General

- Support CAN protocol version 2.0 (A and B) with CAN 2.0B active core
- Bit rates up to 1 Mbit/s

Transmission

- Three transmit mailboxes
- Configurable transmit priority
- Time Stamp on SOF transmission

Reception

- Two receive FIFOs, each with three stages
- Scalable filter banks:
  - 28 filter banks shared between CAN1 and CAN2 for dual CAN
  - 14 filter banks for single CAN (CAN3)
- Identifier list feature
- Configurable FIFO overrun
- Time Stamp on SOF reception

The bxCAN Main Features (2/2)

Possible options

- Debug freeze
- Automatic bus-off management (Error Management)
- Automatic wakeup mode
- No automatic retransmission

Time-triggered communication mode

- 16-bit free running timer
- Time Stamp sent in last two data bytes

Management

- Maskable interrupts
- Software-efficient mailbox mapping at a unique address space

Dual CAN peripheral configuration

- CAN1: Master bxCAN for managing the communication between a Slave bxCAN and the 512-byte SRAM memory, which is shared by CAN2
- CAN2: Slave bxCAN, with no direct access to the SRAM memory.

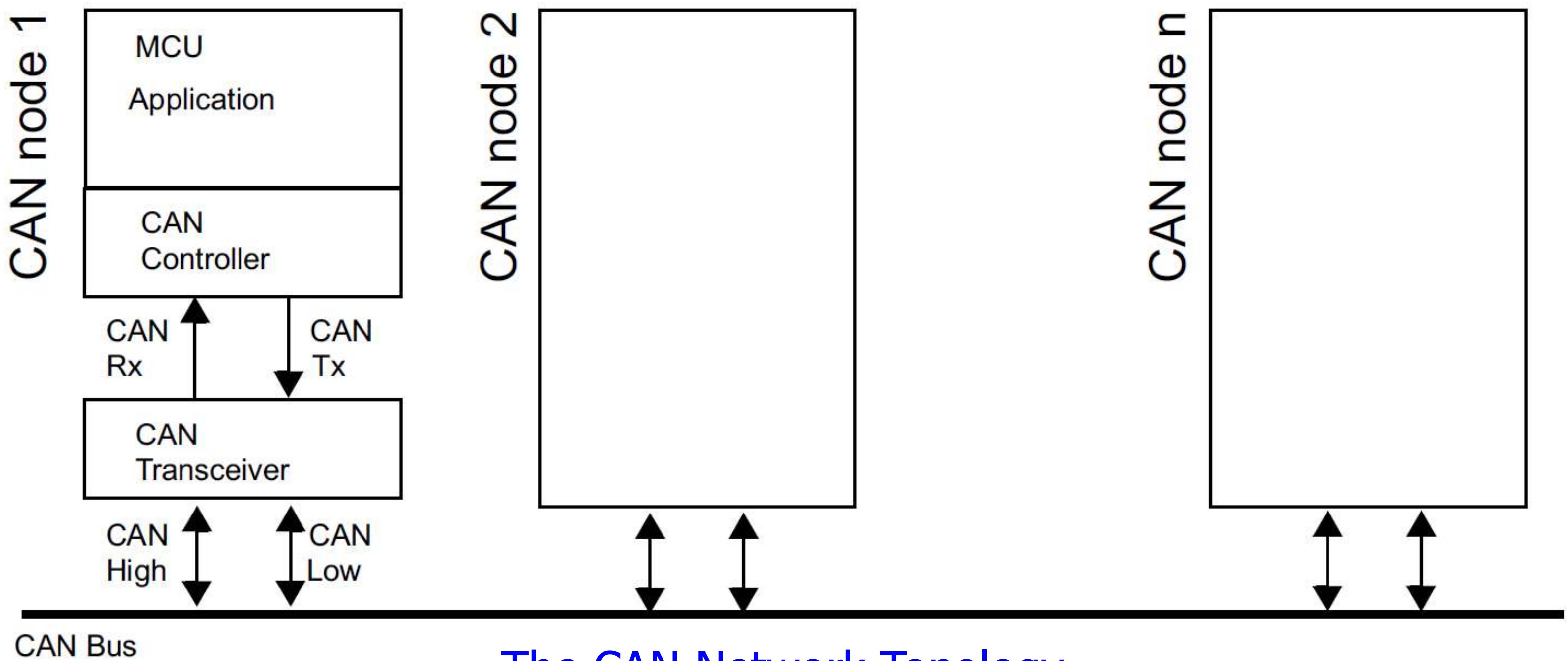
Single CAN peripheral configuration

- CAN3: Master bxCAN with dedicated Memory Access Control unit and 512-byte SRAM memory

Four dedicated interrupt vectors

- Transmit interrupt
- FIFO 0 interrupt
- FIFO 1 interrupt
- Status change error interrupt

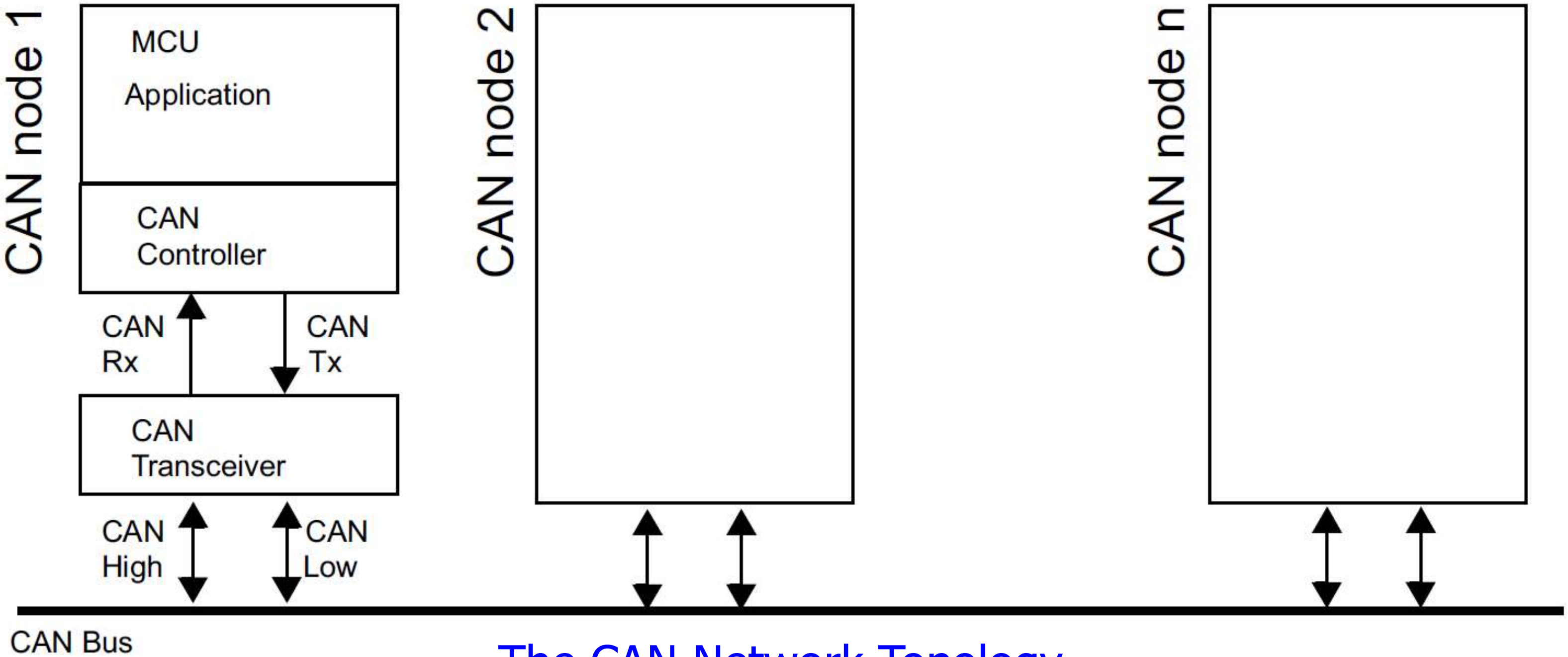
The bxCAN General Description



The CAN Network Topology

- In today CAN applications, the number of nodes in a network is increasing and often several networks are linked together via gateways (or bridges) . Typically, the number of messages in the system (to be handled by each node) has significantly increased. In addition to the application messages, Network Management and Diagnostic messages have been introduced.
    - An enhanced filtering mechanism is required to handle each type of message.

Controller Area Network



The CAN Network Topology

- Furthermore, application tasks require more CPU time, therefore real-time constraints caused by message reception become an issue.
    - A receive FIFO scheme allows the CPU to be dedicated to application tasks for a longer time period without losing messages

- The standard HLP (Higher Layer Protocol) based on standard CAN drivers requires an efficient interface to the CAN controller.

Controller Area Network

- The bxCAN module with CAN 2.0B active core handles the transmission and the reception of CAN messages fully autonomously.

- Standard identifiers (11-bit) and extended identifiers (29-bit) are fully supported by the hardware.

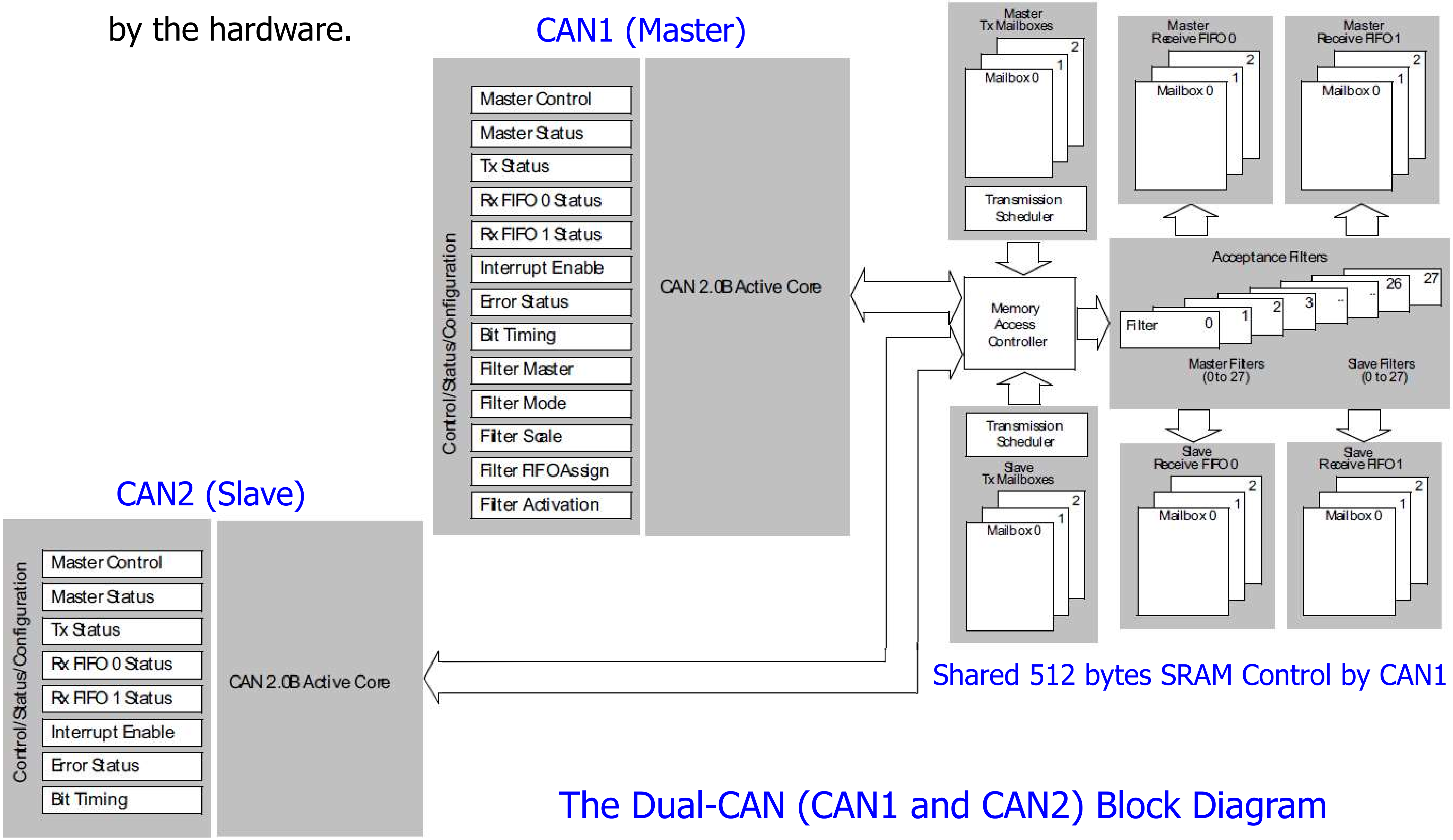
- The application uses control, status and configuration registers to
  - Configure CAN parameters, e.g., baud rate
  - Request transmissions
  - Handle receptions
  - Manage interrupts
  - Get diagnostic information

- For each CAN, three transmit mailboxes are provided to the software for setting up messages. The transmission scheduler decides which mailbox has to be transmitted first.

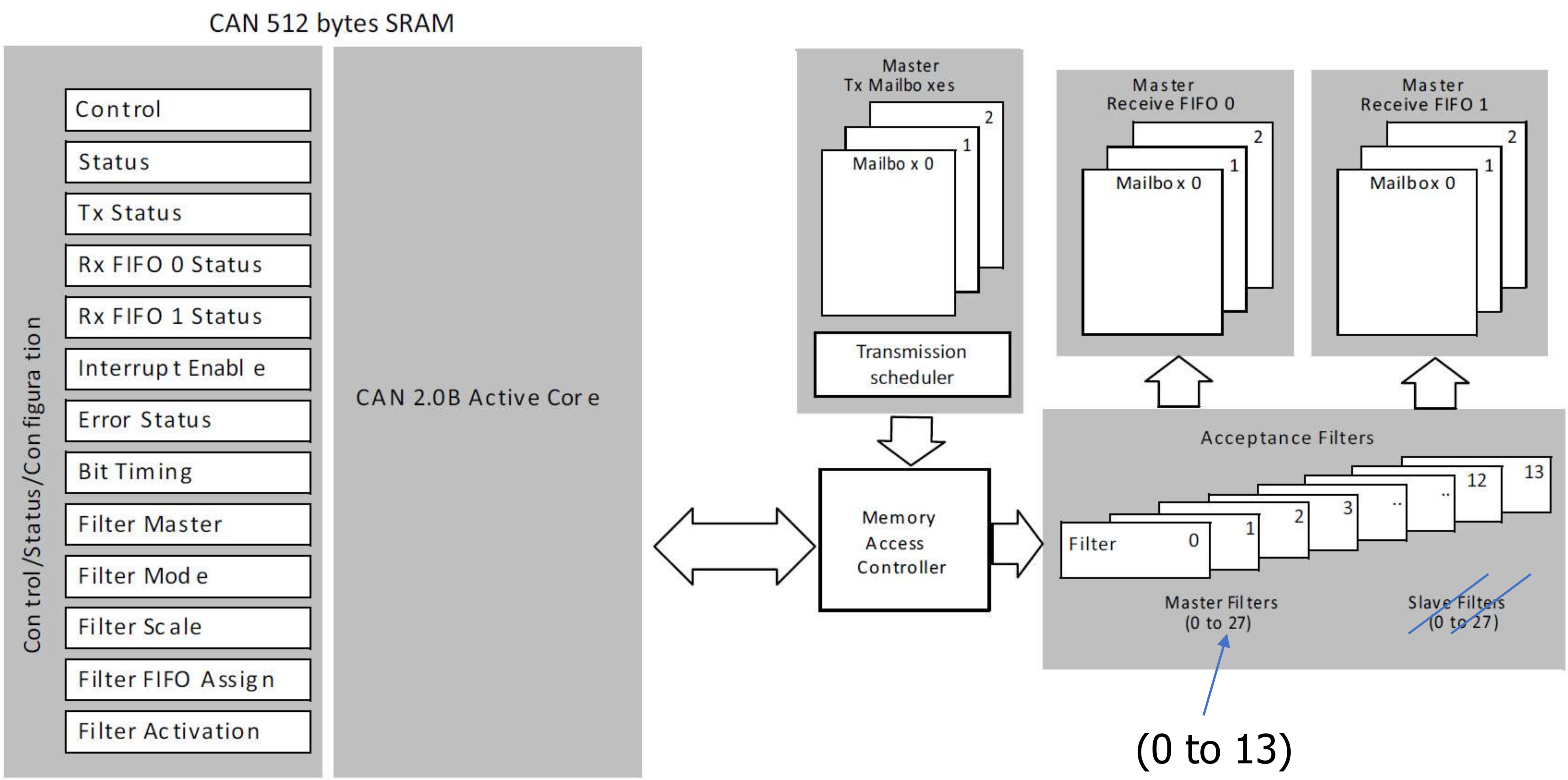
- The bxCAN provides up to 28 scalable/configurable identifier filter banks in dual CAN configuration, for selecting the incoming messages, that the software needs and discarding the others. (14 scalable/configurable identifier filter banks in single CAN conf.)

Controller Area Network

- For each CAN, two receive FIFOs are used by the hardware to store the incoming messages. Three complete messages can be stored in each FIFO. The FIFOs are managed completely by the hardware.



The Dual-CAN (CAN1 and CAN2) Block Diagram



The Single-CAN (CAN3) Block Diagram

CAN Control and Status Registers:

1.1 CAN_MCR (CAN master control register)
1.2 CAN_MSR (CAN master status register)
1.3 CAN_TSR (CAN transmit status register)
1.4 CAN_RF0R (CAN receive FIFO 0 register)
1.5 CAN_RF1R (CAN receive FIFO 1 register)
1.6 CAN_IER (CAN interrupt enable register)
1.7 CAN_ESR (CAN error status register)
1.8 CAN_BTR (CAN bit timing register)

CAN Mailbox Registers:

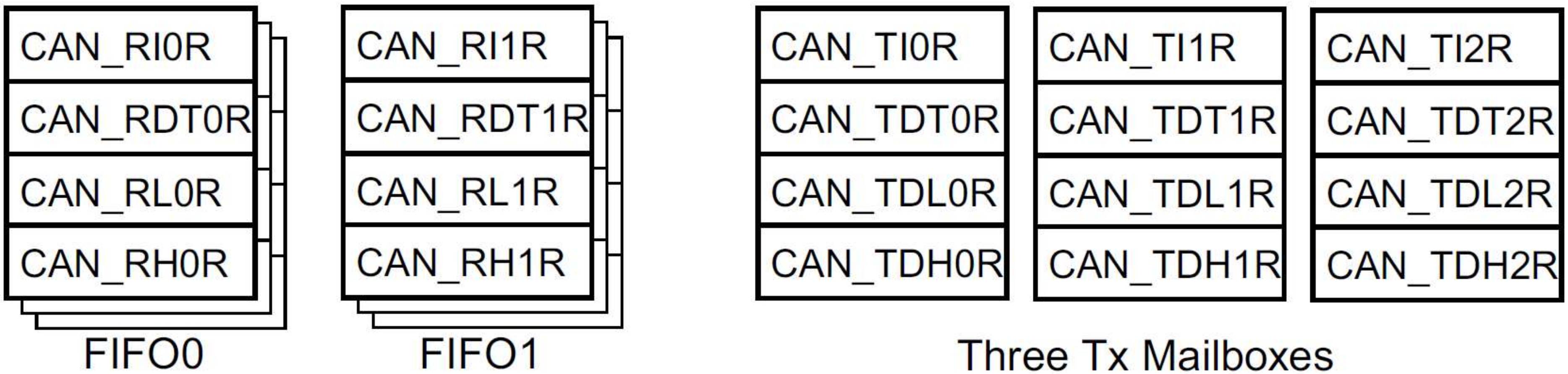
2.1 CAN_TIxR (CAN transmit mailbox identifier register, x = 0..2 for 3 TX mailboxes)
2.2 CAN_TDTxR (CAN transmit mailbox data length control and time stamp register, x = 0..2)
2.3 CAN_TDLxR (CAN transmit mailbox data low register, x = 0..2)
2.4 CAN_TDHxR (CAN transmit mailbox data high register, x =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

CAN Filter Registers:

3.1 CAN_FMR (CAN filter master register)
3.2 CAN_FM1R (CAN filter mode register)
3.3 CAN_FS1R (CAN filter scale register)
3.4 CAN_FFA1R (CAN filter FIFO assignment register)
3.5 CAN_FA1R (CAN filter activation register)
3.6 CAN_FiRx (CAN filter bank i register x, i = 0..27, x = 1,2)

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)

The bxCAN Operating Modes

- The bxCAN has three main operating modes: **Initialization**, **Normal** and **Sleep**.
- After a hardware reset, bxCAN is in **Sleep** mode to reduce power consumption and an internal pull-up is active on CANTX (CAN Tx or CAN_TX) pin.
- The software requests bxCAN to enter **Initialization** or **Sleep** mode by setting the INRQ or SLEEP bit in the CAN Master Control Register (CAN_MCR).
- Once the mode has been entered, bxCAN confirms it by setting the INAK or SLAK bit in the CAN Master Status Register (CAN_MSR).
- When neither INAK nor SLAK is set, bxCAN is in **Normal** mode.
- Before entering **Normal** mode bxCAN always must synchronize with the CAN bus.
- To synchronize, bxCAN waits until the CAN bus is idle, this means **11 consecutive recessive** ("1" or high) bits have been monitored on CANRX (CAN Rx or CAN_RX) pin.
- When the CAN is in **Normal** mode, the user can select to run the **Test** mode.

recessive ("1" or high) vs **dominant** ("0" or low)

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

CAN master status register (CAN_MSR)

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.	RX	SAMP	RXM	TXM	Res.	Res.	Res.	SLAKI	WKUI	ERRI	SLAK	INAK
				r	r	r	r				rc_w1	rc_w1	rc_w1	r	r

RX: CAN Rx signal

SAMP: Last sample point

RXM: Receive mode

TXM: Transmit mode

SLAKI: Sleep acknowledge interrupt

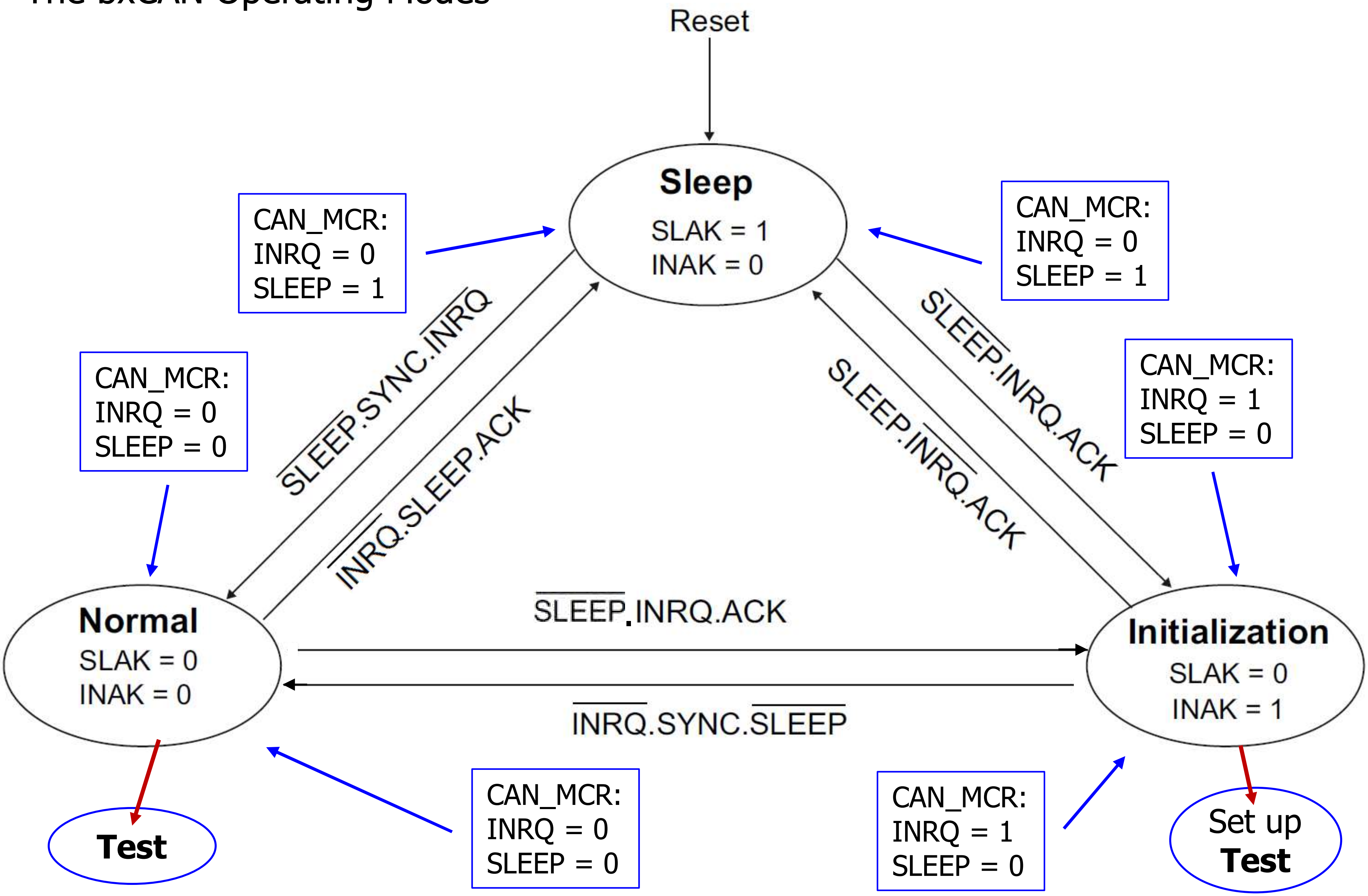
WKUI: Wakeup interrupt

ERRI: Error interrupt

SLAK: Sleep acknowledge

INAK: Initialization acknowledge

The bxCAN Operating Modes



The bxCAN Operating Modes – Initialization mode (1/2)

- The registers can only be configured while the hardware is in Initialization mode.
- To enter this mode, the software sets the INRQ bit in the CAN Master Control Register (CAN_MCR) and waits until the hardware has confirmed the request by setting the INAK bit in the CAN Master Status Register (CAN_MSR).
- To leave Initialization mode, the software clears the INRQ bit. bxCAN has left Initialization mode once the INAK bit has been cleared by hardware.
- While in Initialization mode, all message transfers to and from the CAN bus are stopped and the status of the CAN bus output CANTX pin is recessive (high).
- Entering Initialization mode does not change any of the configuration registers.

The bxCAN Operating Modes – Initialization mode (2/2)

- To initialize the CAN Controller, software must set up the Bit Timing Register (CAN_BTR) and CAN Master Control Register (CAN_MCR).
- To initialize the registers associated with the CAN filter banks (filter mode, filter scale, filter FIFO assignment, filter activation and filter values), software must set the FINIT (filter initialization) bit in the Filter Master Register (CAN_FMR).
- Filter initialization also can be done outside the Initialization mode.
- When FINIT = 1, CAN reception is deactivated.
- The filter values (in CAN_FiRx) is modified by deactivating the associated filter activation bits (FACTi) in the Filter Activation Register (CAN_FA1R).
- If a filter bank is not used, it is recommended to leave it non active, i.e., leave the corresponding FACTi (filter active) bit cleared.

Controller Area Network

CAN bit timing register (CAN_BTR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SILM	LBKM	Res.	Res.	Res.	Res.	SJW[1:0]		Res.	TS2[2:0]			TS1[3:0]			
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.	BRP[9:0]									
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

SILM: Silent mode (debug)

LBKM: Loop back mode (debug)

SJW[1:0]: Resynchronization jump width — tRJW = tq x (SJW[1:0] + 1)

TS2[2:0]: Time segment 2 — tBS2 = tq x (TS2[2:0] + 1)

TS1[3:0]: Time segment 1 — tBS1 = tq x (TS1[3:0] + 1)

BRP[9:0]: Baud rate prescaler — tq = (BRP[9:0]+1) x tPCLK

Recessive: high
Dominant: low

The bxCAN Operating Modes – Normal mode

- Once the initialization is complete, the software must request the hardware to enter Normal mode so that it is able to synchronize with the CAN bus and start reception and transmission.
- The request to enter Normal mode is issued by clearing the INRQ bit in CAN_MCR register.
- The bxCAN enters Normal mode and is ready to take part in bus activities when it has synchronized with the data transfer on the CAN bus.
- This is done by waiting for the occurrence of a sequence of 11 consecutive recessive bits (Bus Idle state).
- The switch to Normal mode is confirmed by the hardware by clearing the INAK bit in the CAN_MSR register.
- The initialization of the filter values is independent from Initialization mode but must be done while the filter is not active (corresponding FACTi bit cleared). The filter mode, scale and FIFO assignment registers must be configured before entering Normal mode.

The bxCAN Operating Modes – Sleep mode (low-power)

- To reduce power consumption, the bxCAN has a low-power mode called **Sleep** mode.
- This mode is entered on software request by setting the SLEEP bit in CAN_MCR register.
- In this mode, the bxCAN clock is stopped, however software can still access the bxCAN mailboxes.
- If software requests entry to **initialization** mode by setting the INRQ bit while bxCAN is in **Sleep** mode, it must also clear the SLEEP bit.
- The bxCAN can be woken up (exit Sleep mode) either by software clearing the SLEEP bit or on the detection of CAN bus activity.
- On the CAN bus activity detection, hardware automatically performs the wakeup sequence by clearing the SLEEP bit if the AWUM bit in the CAN_MCR register is set.
- If the AWUM bit is cleared, software must clear the SLEEP bit when a wakeup interrupt occurs, in order to exit from the Sleep mode.

The bxCAN Operating Modes – Sleep mode (low-power)

- If the wakeup interrupt is enabled [WKUIE bit set in CAN Interrupt Enable Register (CAN_IER)], a wakeup interrupt will be generated on detection of CAN bus activity, even if the bxCAN automatically performs the wakeup sequence.

- After the SLEEP bit has been cleared, Sleep mode is exited once bxCAN has synchronized with the CAN bus.

- The Sleep mode is exited once the SLAK bit has been cleared by hardware.

The bxCAN Test Mode

- The bxCAN supports three test modes: **Silent** mode, **Loop Back** mode, and **Loop Back combined with Silent** mode.

- Test mode can be selected by the SILM (= 1, silent mode) and LBKM (= 1, loop back mode) bits in the Bit Timing Register (CAN_BTR).

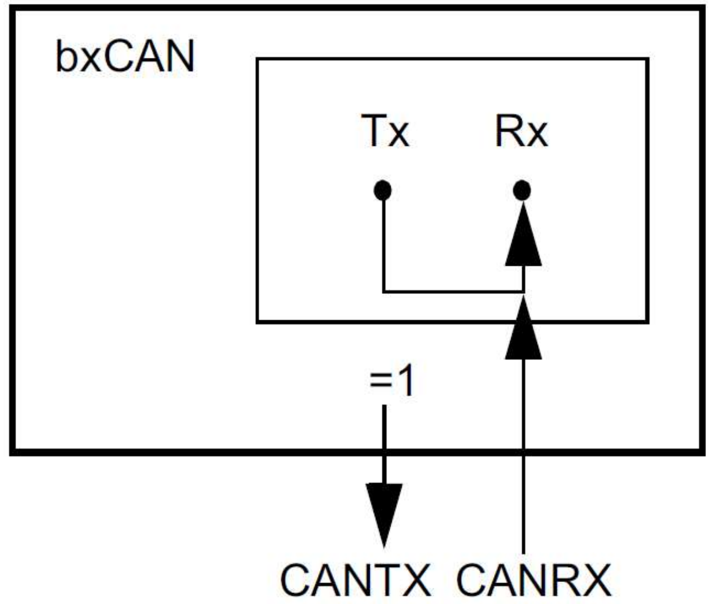
- These bits must be configured while the bxCAN is in Initialization mode.

- Once test mode has been selected, the INRQ bit in the CAN_MCR register must be reset to enter Normal (Test) mode.

The bxCAN Test Mode – Silent mode

- The bxCAN can be put in Silent mode by setting the SILM bit in the CAN_BTR register.

- In Silent mode, the bxCAN can receive valid data frames and valid remote frames, but it sends only recessive (high) bits on the CAN bus, and it cannot start a transmission.



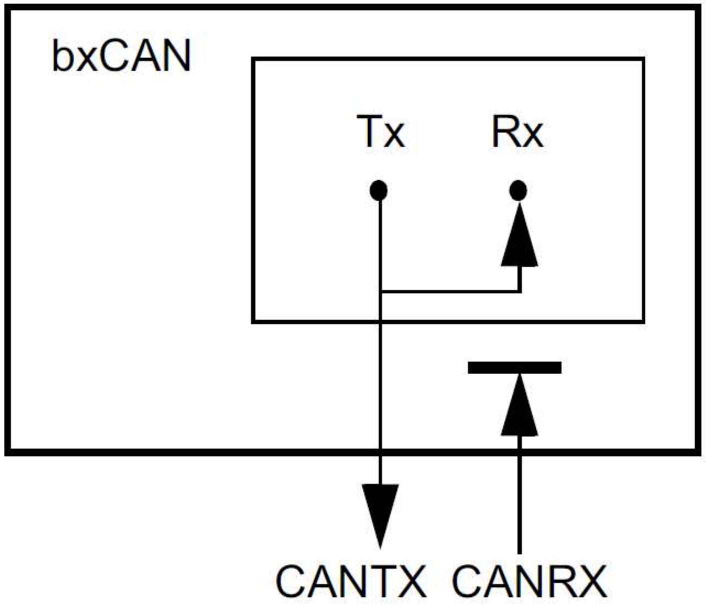
The bxCAN in Silent Mode

- If the bxCAN has to send a dominant (low) bit (ACK bit, overload flag, active error flag), this bit will be rerouted internally to the CAN Core for monitoring purposes. Note that, the CANTX pin is always set at the recessive (high) state.

- The Silent mode can be used to analyze the traffic on a CAN bus without affecting the bus as there is no transmission of dominant (low) bits such as the Acknowledge Bits and Error Frames.

The bxCAN Test Mode – Loop Back mode

- The bxCAN can be set in Loop Back mode by setting the LBKM bit in the CAN_BTR register.
- In Loop Back mode, the bxCAN treats its own transmitted messages as received messages and stores them (if they pass acceptance filtering) in a Receive mailbox.
- This mode is provided for self-test functions.



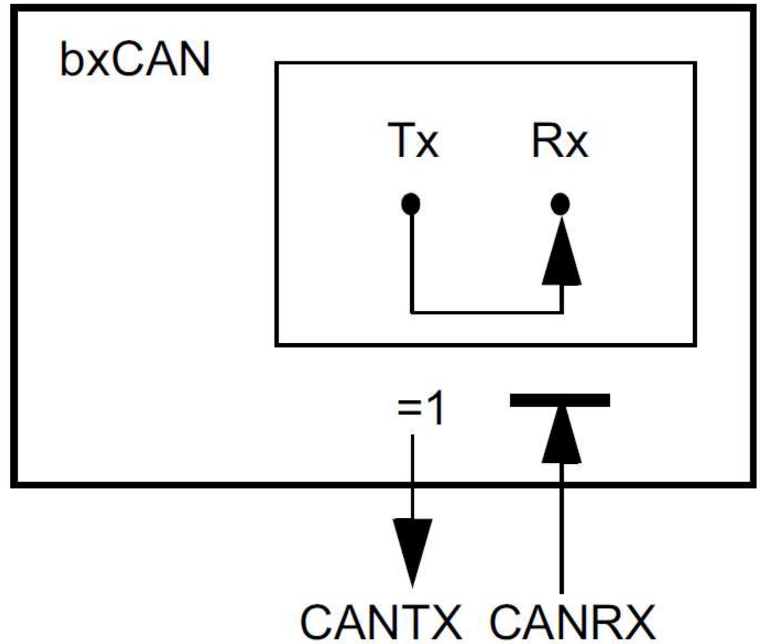
The bxCAN in Loop Back Mode

- To be independent of external events, the CAN Core ignores acknowledge errors (no dominant bit sampled in the acknowledge slot of a data / remote frame) in the Loop Back mode.
- In this mode, the bxCAN performs an internal feedback from its Tx output to its Rx input. The actual value of the CANRX input pin is ignored by the bxCAN. The transmitted messages can be monitored on the CANTX pin.

The bxCAN Test Mode – Loop Back combined with Silent mode

- It is also possible to combine Loop Back mode and Silent mode by setting the LBKM and SILM bits in the CAN_BTR register.

- This mode can be used for a "Host Self-test", i.e., the bxCAN can be tested like in Loop Back mode but without affecting the running CAN system connected to the CANTX and CANRX pins.



The bxCAN in Loop Back combined with Silent Mode

- In this mode, the transmission (Tx) is internally looped to Rx.

- The CANRX pin is disconnected from the bxCAN and the CANTX pin is held recessive (high).

The bxCAN Behavior in Debug Mode

- When the microcontroller enters the debug mode (microcontroller core halted), the bxCAN continues to work normally or stop, depending on:
  - **DBG_CAN1_STOP** bit for CAN1 or **DBG_CAN2_STOP** bit for CAN2 or **DBG_CAN3_STOP** bit for CAN3 in the Debug MCU APB1 freeze (**DBGMCU_APB1_FZ**) register

    Bit 26 **DBG_CAN2_STOP:** Debug CAN2 stopped when Core is halted
    0: Same behavior as in normal mode
    1: The CAN2 receive registers are frozen
    Bit 25 **DBG_CAN1_STOP:** Debug CAN1 stopped when Core is halted
    0: Same behavior as in normal mode
    1: The CAN1 receive registers are frozen
    Bit 13 **DBG_CAN3_STOP:** Debug CAN3 stopped when Core is halted
    0: Same behavior as in normal mode
    1: The CAN3 receive registers are frozen

  - DBF (debug freeze) bit in the CAN Master Control Register (CAN_MCR). The DBF bit is cleared (DBF = 0) for CAN to work during debug. The DBF bit is set (DBF = 1) for CAN to freeze its Rx/Tx during debug. When DBF = 1, reception FIFOs can still be accessed/controlled normally.