CAN Bus Controller

HT45B3305H

Revision: V1.30 Date: March 31, 2023

www.holtek.com

Table of Contents

Features

- Operating Voltage: 3.0V~5.5V
- Oscillator Type: High Speed External Crystal – HXT
- Sleep Mode and Idle Mode
- 32-byte Write Buffer with Data Check Unit for communicating with Host MCUs
- Serial Communication Interfaces – I^2C or SPI (up to 10MHz SPI data rate)
  - In order to ensure correct data transmission, the SPI and I^2C interfaces cannot be used in one(master MCU)-to-multiple(slave devices) situations
- Clock Out pin with programmable prescaler
- Interrupt output pins with selectable active level configuration
- HT45B3305H CAN Core, contains the following features:
  - Conforms to ISO11898-1 and CAN 2.0A/B
  - 32 Message Objects
  - Each Message Object has its own identifier mask
  - Programmable FIFO mode – concatenation of Message Objects
  - Maskable interrupt
  - Programmable loop-back mode for self-test operation
- Support the SOF (Start of Frame) signal output
- 32×139-bit Message Memory
- Package types: 16-pin NSOP/QFN

Applications

- Networked Automotive Products
- Industrial Automation
- Entertainment Products

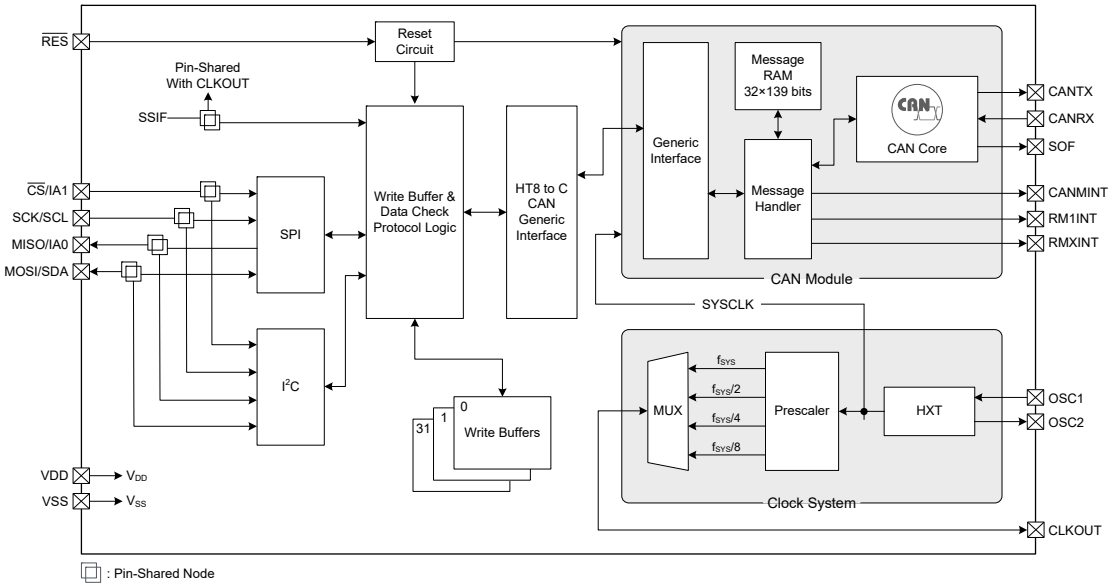
General Description

A complete CAN Node requires a CAN controller, a CAN transceiver and a microcontroller.

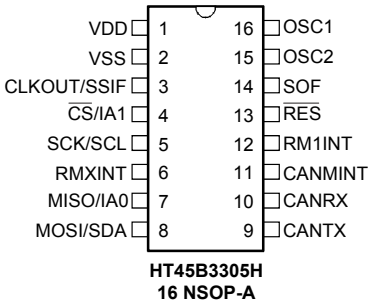
The device is developed as a stand-alone CAN (Controller Area Network) controller. For the connection to the physical layer additional transceiver hardware is required. Two pins of CANTX and CANRX interface to the CAN BUS Transceiver. The device includes a SPI and I^2C interfaces to communicate with a microcontroller.

The HT45B3305H CAN Module licensed from Bosch. The HT45B3305H CAN supports the CAN 2.0 Part A and B protocol specifications and compatible with the ISO11898-1 standards. This CAN Module abbreviated as C_CAN. It is capable of transmitting and receiving standard and extended messages. It also capable of both acceptance filtering and message handler and includes 32 Message Objects which can be concatenated to configurate FIFO buffer with different depth. The SPI interface with data rate of up 10MHz and I^2C serial interface are provided for communication with a SPI or I^2C based 8-bit MCU.

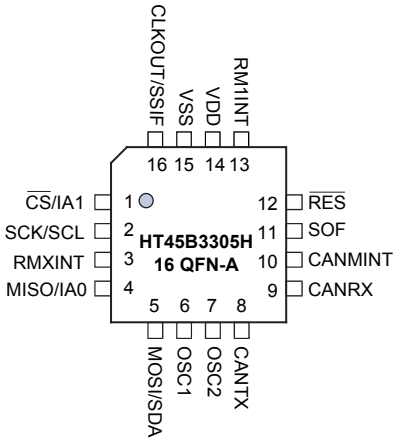
Block Diagram



Pin Assignment



HT45B3305H
16 NSOP-A



HT45B3305H
16 QFN-A

Pin Description

Pin Name	Function	Type	Description
CLKOUT/SSIF	CLKOUT	O	Clock output pin with programmable prescaler
	SSIF	I	SPI or I²C interface selection pin when $\overline{RES}$ was low
$\overline{CS}$/IA1	$\overline{CS}$	I	Chip Select input pin for SPI interface
	IA1	I	Define I²C address bit 1 when $\overline{RES}$ was low
SCK/SCL	SCK	I	Clock input pin for SPI interface
	SCL	I	Clock input pin for I²C interface
MISO/IA0	MISO	O	Master In Slave Out, the device data output for SPI interface
	IA0	I	Defining I²C address bit 0 when $\overline{RES}$ was low
MOSI/SDA	MOSI	I	Master Out Slave In, the device data input for SPI interface
	SDA	I/O	Data input/output for I²C interface
OSC1	OSC1	I	Oscillator input
OSC2	OSC2	O	Oscillator output
SOF	SOF	O	Start of Frame signal
$\overline{RES}$	$\overline{RES}$	I	Reset input, active-low
CANMINT	CANMINT	O	CAN Interrupt output pin
RM1INT	RM1INT	O	Receive a Message into Message Object 1 Successfully Interrupt
RMXINT	RMXINT	O	Receive a Message into Message Object x Successfully Interrupt, the x value is user-defined using the CANCFG Register
CANTX	CANTX	O	Transmit output pin to CAN bus
CANRX	CANRX	I	Receive input pin from CAN bus
VDD	VDD	PWR	Digital positive power supply.
VSS	VSS	PWR	Ground

Legend: I= Input; O= Output; PWR= Power

Absolute Maximum Ratings

Supply Voltage ..... V_{SS}-0.3V to 6.0V

Input Voltage ..... V_{SS}-0.3V to V_{DD}+0.3V

Storage Temperature..... -50°C to 125°C

Operating Temperature..... -40°C to 125°C

I_{OL} Total 80mA

I_{OH} Total..... -80mA

Total Power Dissipation 500mW

Note: These are stress ratings only. Stresses exceeding the range specified under "Absolute Maximum Ratings" may cause substantial damage to the device. Functional operation of this device at other conditions beyond those listed in the specification is not implied and prolonged exposure to extreme conditions may affect device reliability.

D.C. Characteristics

For data in the following tables, note that factors such as oscillator type, operating voltage, operating frequency, pin load conditions, temperature and program instruction type, etc., can all exert an influence on the measured values.

Operating Voltage Characteristics

Ta=-40°C~125°C, unless otherwise specified.

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{DD}	Operating Voltage (HXT)	—	$f_{SYS}=f_{HXT}$=8MHz	3.0	—	5.5	V
			$f_{SYS}=f_{HXT}$=16MHz	3.0	—	5.5	V
			$f_{SYS}=f_{HXT}$=24MHz	4.5	—	5.5	V

Operating Current Characteristics

Ta=-40°C~125°C, unless otherwise specified.

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
I_{DD}	Operating Current (HXT)	3V	No load, all peripherals off, $f_{SYS}=f_{HXT}$=8MHz	—	8	13	mA
		5V		—	13	18	mA
		3V	No load, all peripherals off, $f_{SYS}=f_{HXT}$=16MHz	—	15	20	mA
		5V		—	25	30	mA
		3V	No load, all peripherals off, $f_{SYS}=f_{HXT}$=24MHz	—	23	28	mA
		5V		—	38	41	mA
V_{IL}	Input Low Voltage for SSIF/$\overline{CS}$/SCK/SCL/MOSI/SDA/IA0/IA1/CANRX Pins	—	—	0	—	$0.2V_{DD}$	V
	Input Low Voltage for $\overline{RES}$ Pin	—	—	0	—	$0.4V_{DD}$	V
V_{IH}	Input High Voltage for SSIF/$\overline{CS}$/SCK/SCL/MOSI/SDA/IA0/IA1/CANRX Pins	—	—	$0.8V_{DD}$	—	V_{DD}	V
	Input High Voltage for $\overline{RES}$ Pin	—	—	$0.9V_{DD}$	—	V_{DD}	V
I_{OL}	Sink Current for MISO/SDA/RMXINT/RM1INT/CANMINT/CLKOUT/SOF/CANTX Pins	3V	V_{OL} = $0.1V_{DD}$	4	8	—	mA
		5V		10	20	—	mA
I_{OH}	Source Current for MISO/SDA/RMXINT/RM1INT/CANMINT/CLKOUT/SOF/CANTX Pins	3V	V_{OH}=$0.9V_{DD}$	-2	-4	—	mA
		5V		-5	-10	—	mA

Standby Current Characteristics

Ta=-40°C~125°C, unless otherwise specified.

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Max. 85°C	Max. 105°C	Max. 125°C	Unit
		V_{DD}	Conditions							
I_{STB}	Standby Current (SLEEP mode)	3V	No load, all peripherals off	—	—	1	1	3	5	μA
		5V		—	—	2	2	5	8	μA
	Standby Current (IDLE mode)	3V	No load, all peripherals off, $f_{SYS}=f_{HXT}$=8MHz	—	—	1	1	1	1	mA
		5V		—	—	1.5	1.5	1.5	1.5	mA
		3V	No load, all peripherals off, $f_{SYS}=f_{HXT}$=16MHz	—	—	1.2	1.2	1.2	1.2	mA
		5V		—	—	2.2	2.2	2.2	2.2	mA
		3V	No load, all peripherals off, $f_{SYS}=f_{HXT}$=24MHz	—	—	1.8	1.8	1.8	1.8	mA
		5V		—	—	2.8	2.8	2.8	2.8	mA

A.C. Characteristics

For data in the following tables, note that factors such as oscillator type, operating voltage, operating frequency and temperature etc., can all exert an influence on the measured values.

System Frequency Characteristics

Ta=-40°C~125°C, unless otherwise specified.

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
f_{SYS}	System clock (HXT)	3.0V~5.5V	$f_{SYS}=f_{HXT}$=8MHz	—	8	—	MHz
		3.0V~5.5V	$f_{SYS}=f_{HXT}$=16MHz	—	16	—	MHz
		4.5V~5.5V	$f_{SYS}=f_{HXT}$=24MHz	—	24	—	MHz

Timing Characteristics

Ta=-40°C~125°C, unless otherwise specified.

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
t_{RES}	External Reset Minimum Low Pulse Width	—	—	10	—	—	μs
$t_{START(HXT)}$	HXT Oscillator Startup Time	3V	$f_{SYS}=f_{HXT}$=16MHz	—	—	25	ms
		5V		—	—	10	ms
f_{I2C}	I^2C Standard Mode (100kHz) f_{SYS} Frequency	3.0V~5.5V	No clock debounce	2	—	—	MHz
			2 system clock debounce	4	—	—	MHz
			4 system clock debounce	8	—	—	MHz
	I^2C Fast Mode (400kHz) f_{SYS} Frequency	3.0V~5.5V	No clock debounce	5	—	—	MHz
			2 system clock debounce	10	—	—	MHz
			4 system clock debounce	20	—	—	MHz
f_{SPI}	SPI Mode Frequency	—	—	—	—	10	MHz
t_{ITO}	I^2C Timeout Period	3.0V~5.5V	$f_{SYS}=f_{HXT}$=16MHz	40	—	—	ms
t_{SOF}	SOF signal width	—	$f_{SYS}=f_{HXT}$=24MHz, SOFT[2:0]=101	9.6	10.6	11.6	μs

CAN Electrical Characteristics

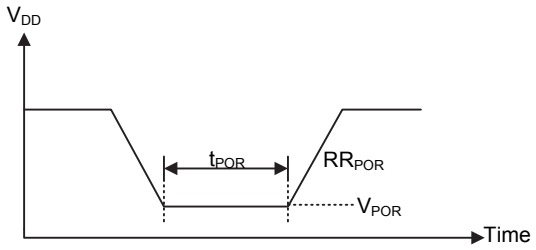
Ta=-40°C~125°C, unless otherwise specified.

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{DD}	Operating Voltage (HXT)	—	f_{CAN}=8MHz	3.0	—	5.5	V
		—	f_{CAN}=16MHz	3.0	—	5.5	V
		—	f_{CAN}=24MHz	4.5	—	5.5	V
f_{CAN}	System Clock (HXT)	3.0V~5.5V	—	—	8	—	MHz
		3.0V~5.5V		—	16	—	MHz
		4.5V~5.5V		—	24	—	MHz
f_{MCLK}	Memory Clock (HXT)	3.0V~5.5V	—	—	8	—	MHz
		3.0V~5.5V		—	16	—	MHz
		4.5V~5.5V		—	24	—	MHz

Power-on Reset Characteristics

Ta=-40°C~125°C, unless otherwise specified.

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{POR}	V_{DD} Start Voltage to Ensure Power-on Reset	—	—	—	—	100	mV
RR_{POR}	V_{DD} Rising Rate to Ensure Power-on Reset	—	—	0.035	—	—	V/ms
t_{POR}	Minimum Time for V_{DD} Stays at V_{POR} to Ensure Power-on Reset	—	—	1	—	—	ms



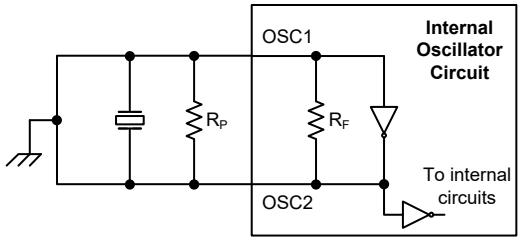
Power Control Function

The device operating clock is from an external crystal oscillator, HXT. The oscillator can be enabled or disabled using a register bit HXTEN. The clock which is a divided version of the system clock can be output on the CLKOUT pin.

External Crystal Oscillator – HXT

There is a high frequency external crystal oscillator for this device. For most crystal oscillator configurations, the simple connection of a crystal across OSC1 and OSC2 will create the necessary phase shift and feedback for oscillation, without requiring external capacitors. It is recommended to connect an 8MHz, 16MHz or 24MHz crystal to the HXT pins for applications.

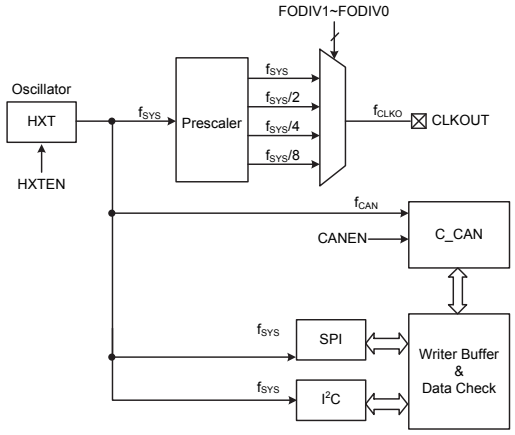
For oscillator stability and to minimise the effects of noise and crosstalk, it is important to ensure that the crystal and any associated resistors and capacitors along with interconnecting lines are all located as close to the device as possible.



Note: 1. R_P is normally not required
2. Although not shown OSC1/OSC2 pins have a parasitic capacitance of around 7pF.

CLKOUT Pin

The clock output pin, CLKOUT is provided to the users for use as a clock input for other devices. The CLKOUT pin has an internal prescaler which can devide f_{SYS} by 1, 2, 4 and 8. The prescaler is selected via the FOCFG register. When clearing the HXTEN bit in the FOCFG register (ADDRESS C0H) to zero, the HXT oscillator is off thus turning off the CLKOUT clock output.



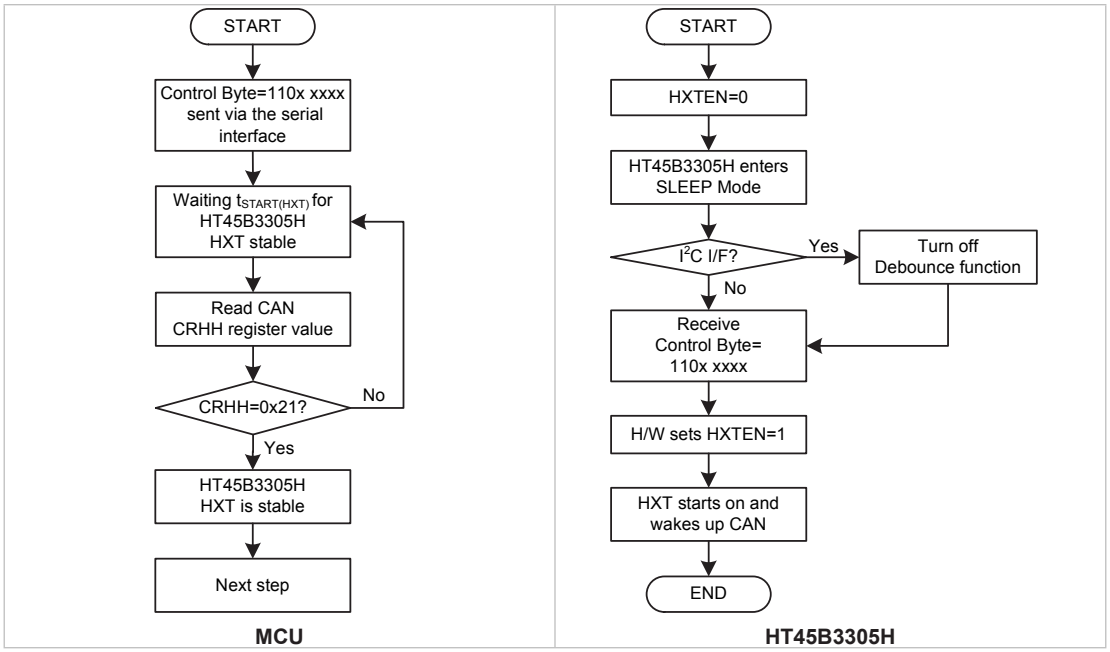
IDLE Mode

The IDLE Mode is entered when the HXTEN bit in the FOCFG register is high while the CANEN bit in the CANCFG register is low. In the IDLE Mode the HXT oscillator is continue to provide a clock. The C_CAN is disabled.

SLEEP Mode and Wake-up

The device has an internal SLEEP mode that is used to minimize the current consumption of the device. In the SLEEP Mode the oscillator is turned off. If the I^2C serial interface is selected, the I^2C interface debounce function will be disabled after entering the SLEEP Mode.

To enter the SLEEP mode, the HXTEN bit in the FOCFG register should be cleared to zero. The master can wake up the device by sending a Wake-up command to set the HXTEN bit high and then read the CRHH register (Address: 0x3B) which indicates whether the HXT oscillator is stable after $t_{START(HXT)}$ time. If the CRHH has a value of 21H, it means the HXT oscillator is stable and the device is success fully waked-up by the master MCU from the SLEEP mode.

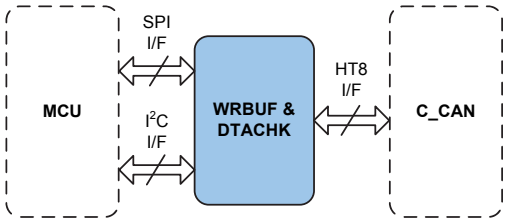


Functional Description

The Controller Area Network, or CAN bus for short, is a standard communication protocol used originally designed for automotive networking applications, however it is also used in other application areas such as industrial automation and some entertainment products. It is a two wire serial bus to which the CAN bus equipped products are connected together using twisted pair cable with a characteristic impedence of 120Ω.

Write Buffer and Data Check

The device provides a 32-byte Write Buffer with Data check function to easier the communication with Host MCUs and reduce the number of the interface pins between the peripheral device and its Host MCU.



The device contains two sets of interface modules which are the four line SPI interface and the two line I²C interface, to allow an easy method of communication with external Master devices. Having relatively simple communication protocols, these serial interface types allow the device to interface to external SPI or I²C based microcontrollers. The choice of whether the SPI or I²C type is used is made using the SSIF pin input signal and with the $\overline{RES}$ pin low. Users can choose using SPI or I²C serial interface for the data transfer, based on their MCU support interfaces and application speed requirements.

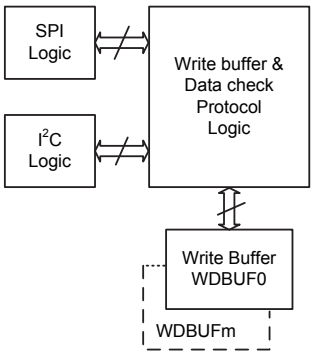
SSIF pin should connect a 500kΩ pull-low resistor. Communication interface switching is implemented via this pin. During the device reset period which is caused by keeping the $\overline{RES}$ pin low, the SSIF pin input can be used to select the serial interface used with the master. The selections are shown in the following table.

When release the $\overline{RES}$ pin, the external reset is

$\overline{RES}$ & SSIF Conditions	Selected Interface
$\overline{RES}$=Low, SSIF=Low	SPI interface
$\overline{RES}$=Low, SSIF=High	I²C interface

Interface Selections

The 32-byte Writter Buffer can be used to store the Control byte, Register Address byte and up to 31 bytes Data which are received or to be transmit in a communication.



Block Diagram

SPI and I²C Frame Fields

Following the Communication Protocol, the SPI or I²C frame should contain five data fields which are Control Byte, Register Address, Control CheckSum, Data and Data CheckSum.

- A 1-byte control field, including:
  - A 3-bit control instruction code defining the operation command.
  - A 5-bit data length code bits defining the size (in bytes) of the data field.
- A 1-byte Register Address field, defining the start register address to read from or to write into
- A 1-byte Control CheckSum field, Detecting errors during the control byte and the address byte transmission.The control checksum is based on a XOR operation
- A Data field of up to 31 bytes
- A 1-byte Data CheckSum field, Detecting errors during the data transmission.The control checksum is based on a XOR operation of all write/read data.

Control Byte

For each data transfer, a Control Byte is initiated to specify which Instruction is executed and how many bytes of data is transferred. The bit7~bit5 of the Control Byte, named INSTR[2:0], define the instruction while the bit4~bit0 of the Control Byte, named SDLC[4:0], is the data length code for setting the number of the data bytes to be received or transmitted.

• Serial Data Length Code

The data byte numbers of 1~31 can be determined by programming the SDLC[4:0] bits in the control byte.

SDLC[4:0]	Serial Data Length	Description
00000	Not Valid	Not Valid
00001~11111	1~31	Valid programmed values 1~31

Note: The data length must be defined correctly when the INSTR[2:0] bits are set as 010 or 101. For other instructions, the defining of the data length is not required.

• Instruction Control Code

The instruction is determined by the INSTR bit field of the control byte.

INSTR[2:0]	Instruction	Description
000 or 001 or 011	Not Valid	Not Valid
010	Write Data	To write data to Buffer
100	Read Status	Read HT45B3305H status
101	Read Data	To read CAN register data
110	Wake Up	Wake CAN Up
111	Reset	Reset CAN Block(can_reset) - Resets internal CAN registers to default state

Note: If set INSTR[2:0]=010 or 101, the SDLC[4:0] bits must be set correctly to define the length of the data to be read or written.

Instruction Description

The instruction byte is sent to the device via the SPI or I²C interface for different operations. Refer to "SPI Interface Timing Diagram" or "I²C Interface Timing Diagram" for detailed input and output timing diagram.

Write Data to Buffer

An instruction code of 010B should be transmitted to the device. The SDLC[4:0] bits in the control byte must be programmed correctly to define the data length to be written. The Register Address field defines the starting register address where the following data will be written into.The register address is automatically incremented by one to store the next byte of data until all the data bytes are written.

A byte Control byte, a byte address byte and the written data bytes will be written into the write buffer. Each transmission of the control and address bytes is followed by a XOR checksum byte for the control and address byte data. And another XOR checksum byte of the data bytes is transferred after the transmission of the data field for detecting errors during the data transmission.

Read CAN Register Data

An instruction code of 101B should be transmitted to the device. The SDLC[4:0] bits in the control byte must be programmed correctly to define the data length of reading CAN registers. The Address field byte defines the starting address of the CAN registers that the master wanted to read from. A control checksum byte which computes the exclusive or (XOR) of the control and address data is transmitted for detecting the control and address byte errors. After the control byte, address byte and the checksum byte are sent, the data stored in the registers at the selected starting address can stored in the buffer. The internal register address is automatically incremented by one to read the data and store it to the buffer until the defined SDLC[4+0] bytes of data all were read. And another checksum byte which computes the XOR of the read data is following the data field for detecting data errors. All the data will be shifed out on the MISO pin. Then the master can read the data.

Before reading out the required register data, the first byte read by the master is a simple status byte which is used to determine whether the device is busy and the reading address is matched or not. Different values of the first byte and the corresponding status they indicate are summaried in the following table:

1st Byte=	Description
Write Address	It means the register address to read has been written correctly by the host MCU.
0xFD	Control CheckSum error. It means a error in writing the reading address
0xFE	HT45B3305H Busy
Others	Don't care

The "Read CAN Register Data" instruction is used to read the device register content and provide brief current error information about the device internal processing status such as access address error and device busy status.

Read Status

The Read Status Instruction allows single instruction access to some bits about the device status.

The part is selected by an instruction code of 100B transmitted to the device. After the read status instruction is sent by the host, the device will return eight bits of data that contain the status.

Status Byte	Description	Initial Value
Bit7	HT45B3305H Buffer Busy flag bit 0: Ready 1: Busy	0
Bit6	Transfer Error 0: Normal 1: INSTR[2:0] value in Control Byte invalid or I²C time-out occurred if I²C interface was selected	0
Bit5	Control CheckSum Error 0: Ok 1: Error	0
Bit4	Data CheckSum Error 0: Ok 1: Error	0
Bit3	Bit3 = $\overline{\text{Bit7}}$	1
Bit2	Bit2 = $\overline{\text{Bit6}}$	1
Bit1	Bit1 = $\overline{\text{Bit5}}$	1
Bit0	Bit0 = $\overline{\text{Bit4}}$	1

In the status byte, Bit3~Bit0 is the Bit3= $\overline{\text{Bit7}}$, Bit2=$\overline{\text{Bit6}}$, Bit1=$\overline{\text{Bit5}}$ and Bit0=$\overline{\text{Bit4}}$. This four bit feild is for the purpose of detecting errors about the status bits. So the Checksum field can be omitted.

The Reading Status instruction should be executed in the following conditions:

- Before the initialisation after a reset, it needs first to determine whether the device is busy or not?
- Reading the device status instruction can be executed after writing data into important registers, to confirm the data was written correctly.
- When using the “Read CAN Register Data” instruction, if the first byte data received by the device has the value of 0xFD or 0xFE and users need complete error information, then the “Read Status Instruction” can be used to determine the actual condition, such as the device busy, Control CheckSum error, Transfer error or Data CheckSum error.

Wake CAN Up

If the HXTEN bit is cleared to zero, the HXT oscillator will stop and the device enters the Sleep Mode. To wake up the device, an instruction code of 110B can be sent. Refer to the “SLEEP Mode and Wake-up” section for detailed wake-up process.

It needs to note for the wake up instruction, the SDLC[4:0] bits in the Control field and the following four fields which are Register Address, Control CheckSum, Data and Data CheckSum are not required. But if the frame contains these four fields, the device will also save them into the buffer without processing them and an error will not happen.

Reset CAN Block Instruction

The Reset CAN Block Instruction can be used to re-initialise the internal CAN registers of the device to default state. Only an instruction code of 111B should be transmitted to the device for the reset operation. The SDLC[4:0] bits in the Control field and the following four fields which are Register Address, Control CheckSum, Data and Data CheckSum are not required. But if the frame contains these four fields, the device will also store them into the buffer without processing them and an error will not happen.

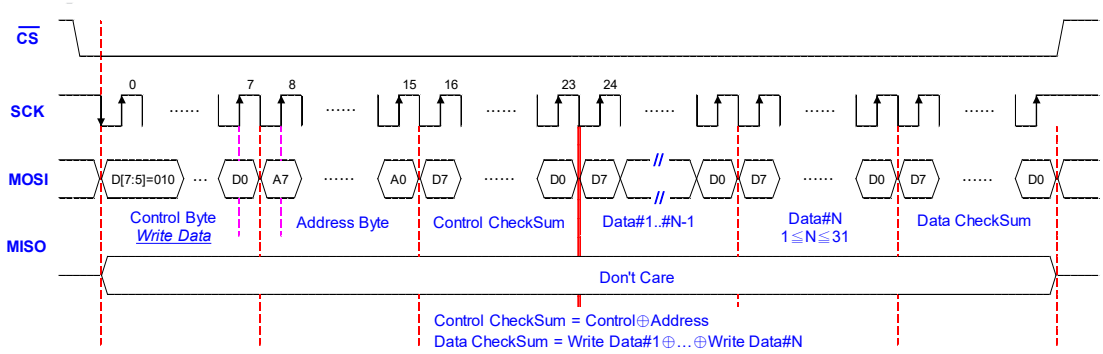
SPI Serial Interface

The device is designed to interface directly with the Serial Peripheral Interface (SPI) port available on many microcontrollers. Commands and data are sent to the device via the MOSI pin, with data being clocked in on the rising edge of SCK. Data is driven out by the device, on the MISO line, on the falling edge of SCK. The $\overline{CS}$ pin must be held low while any operation is performed. When raising the $\overline{CS}$ pin from low to high, the SPI Interface will be reset.

Note: Wait 10 HXT clocks for every 8 bits of command/position/data.

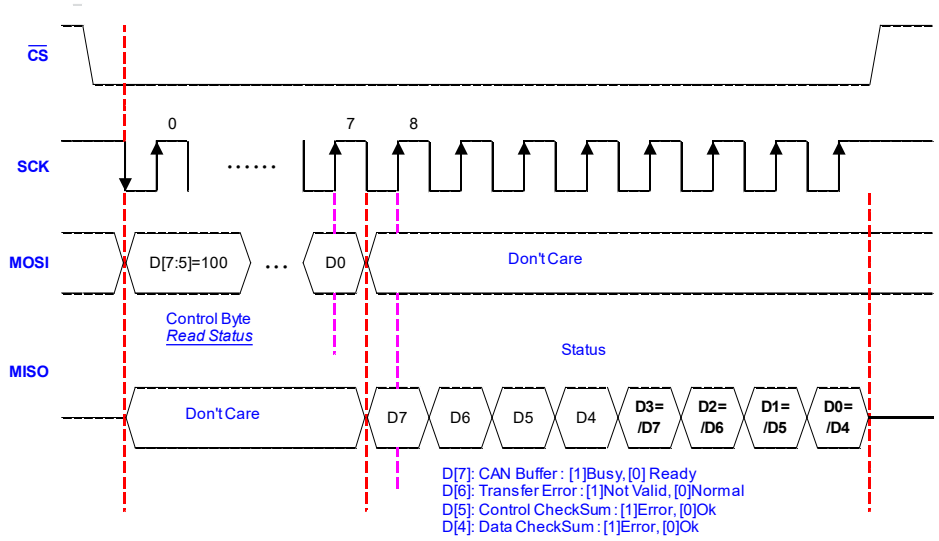
SPI Interface Timing Diagrams

Write Data to Buffer



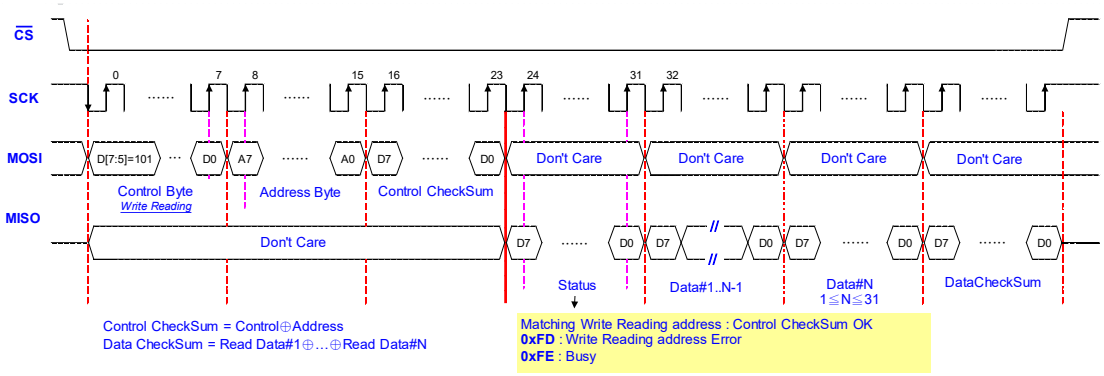
Write Data to Buffer Timing Diagram

Read Status



Read Status Timing Diagram

Read CAN Register Data



Read CAN Register Data Timing Diagram

I²C Serial Interface

During the device reset with the $\overline{RES}$ pin held low, setting the SSIF pin input high, then the I²C interface function together with SCL and SDA pin functions are selected; the I²C interface is used for communication with the master.

The I2CDEB1 and I2CDEB0 bits in the FOCFG register determine the debounce time of the I²C interface. This uses the internal clock to in effect add a debounce time to the external clock to reduce the possibility of glitches on the clock line causing erroneous operation. The debounce time, if selected, can be chosen to be either 2 or 4 system clocks. There is also an option of no debounce can be selected.

I²C Slave Address

The slave address byte is the first byte received following the START condition from the master device. The first seven bits of the first byte make up the slave address. The eighth bit defines a read or write operation to be performed. When this R/$\overline{W}$ bit is "0", then a write operation is selected. A "1" selects a read operation. The device address bits are "10101A_1A_0" where the "A1, A0" value should be the IA1, IA0 pin external level. When an address byte is sent, the device compares the first seven bits after the START condition. If they match, the device outputs an acknowledge on the SDA line.

I²C Slave Address definition:

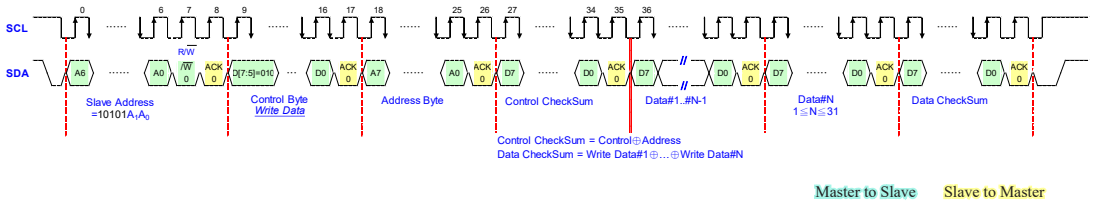
IA1 Pin Level	IA0 Pin Level	I²C Slave Address (10101A1A0)
Low	Low	1010100
Low	High	1010101
High	Low	1010110
High	High	1010111

I²C Timeout Function

In order to reduce the I²C data transfer problem due to reception of erroneous clock sources, a time-out function is provided. When the device is receiving data via the I²C interface, a SCL low level keeps for a time over the specified timeout period of t_{TTO}, the I²C interface will be reset.

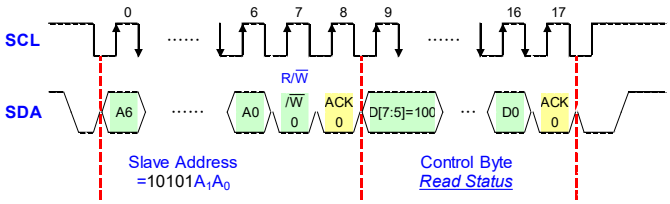
I²C Interface Timing Diagrams

Write Data to Buffer

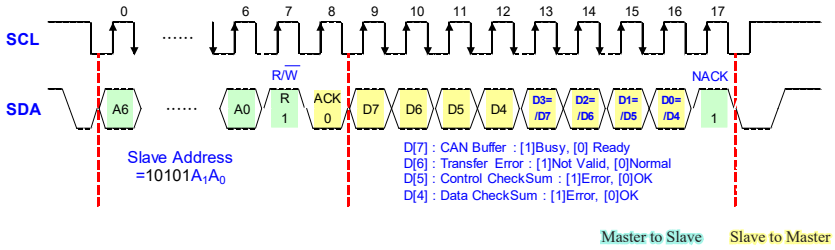


Write Data to Buffer Timing Diagram

Read Status

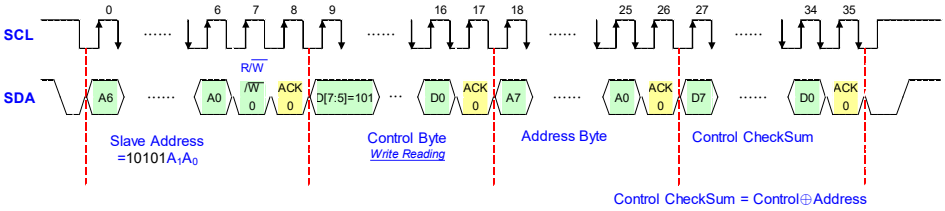


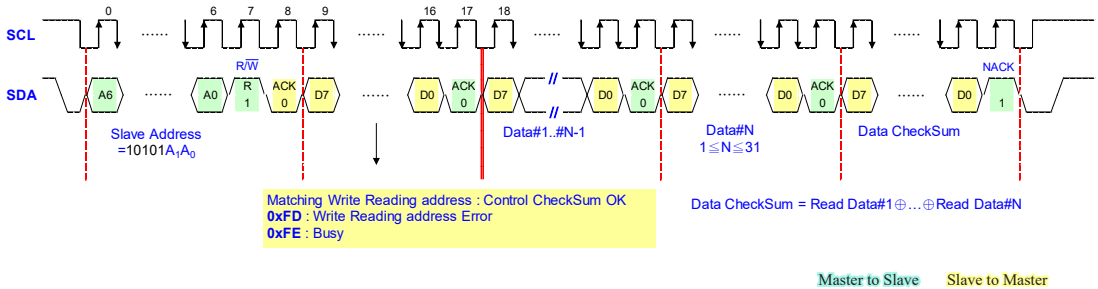
Wait 0 Clocks



Read Status Timing Diagram

Read CAN Register Data

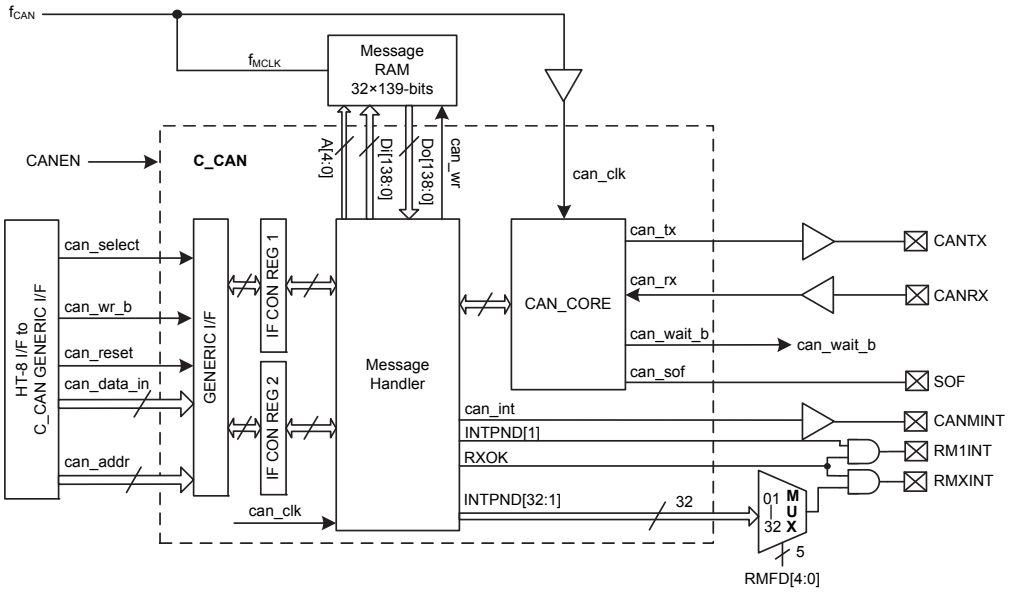




Read CAN Register Data Timing Diagram

HT45B3305H CAN Block Diagram

The Holtek CAN Module licensed from Bosch which supports CAN communication with up to 8 byte data fields. The device implements the HT45B3305H CAN. As the same with the C_CAN, the HT45B3305H CAN consists of the components CAN Core, Message RAM, Message Handler, Interface Control Register sets. A HT-8 Interface to C_CAN generic interface is provided by the HT45B3305H CAN, via which the external 8-bit MCUs could access the C_CAN registers.



HT45B3305H CAN Block Diagram

(1) C_CAN Core

Refer to the following Operating Description and Application section for further CAN module operation details. In this section we give a description about the functional blocks of the C_CAN:

- CAN_Core
  The CAN_Core performs communication according to the CAN protocol version 2.0 A, B and ISO 11898-1.
- Registers
  All registers are used to control and to configure the module.
- Message Handler
  The internal State Machine controls the data transfer between the RX/TX Shift Register of the CAN_Core and the Message RAM as well as the generation of interrupts as programmed in the Control and Configuration Registers. All functions concerning the handling of messages are implemented in the Message Handler. Those functions are the acceptance filtering, the transfer of messages between the CAN Core and the Message RAM, and the handling of transmission requests as well as the generation of the module interrupt.
- Interface Control Register 1 and 2
  The function of the two interface control register sets is identical (except in Basic mode). The interface control register sets are used for the data transfer between the external bus and the Message RAM.
- Message RAM Interface
  Message RAM size: 139-bits×32

(2) Message RAM IP

- Stores 32 Message Objects and Identifier Masks.
- Each Message Object together with Identifier Mask has a length of 139 bits.

(3) HT-8 interface to C_CAN Generic Interface

- The external 8-bit microcontroller can access the C_CAN registers through this transform interface.
- By the firmware, the MCU defined the desired register address. It is only allowed to access one register each time. To implement consecutive access operations, the address should be incremented by one using the firmware.

Interrupt Output Pins

The device has three interrupt output pins, CANMINT, RM1INT and RMXINT, to be used to indicate different conditions. When a CAN interrupt occurs, the CANMINT pin will be driven an active level by the device. When a Message is received into Message Object 1 successfully, an interrupt occurs and the RM1INT pin will output an active level. When a Message is received into Message Object x successfully, an interrupt occurs and the RMXINT pin will output an active level.

Interrupt active level can be selected to be high or low using the FOCFG register bits.

Message RAM and FIFO Buffer Configuration

For communication on a CAN network, individual Message Objects are configured. The Message Objects and Identifier Masks for acceptance filtering of received messages are stored in the Message RAM. The device includes a Message Memory capacity of 139-bit ×32 for storing 32 Message Objects and Identifier Masks. A Message Objects and Identifier Masks is 139 bits which is shown in the following table.

Structure of a Message Object in the Message RAM							
MSKn28~00	MXTDn	MDIRn	UMASKn	TXnIEN	RXnIEN	RMTnEN	EOBn
IDn28~00	XTDn	DIRn	MSGnLST	—	—	—	DLCn[3:0]
DATA0	DATA1	DATA2	DATA3	DATA4	DATA5	DATA6	DATA7

MSKn28~00 Identifier Mask
0: The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering
1: The corresponding identifier bit is used for acceptance filtering

IDn28~00 Message Identifier
IDn28~IDn00: 29-bit Identifier (“Extended Frame”).
IDn28~IDn18: 11-bit Identifier (“Standard Frame”).

MXTDn Mask Extended Identifier
0: The extended identifier bit (IDE) has no effect on the acceptance filtering
1: The extended identifier bit (IDE) is used for acceptance filtering
Note: When 11-bit (“standard”) Identifiers are used for a Message Object, the identifiers of received Data Frames are written into bits **IDn28** to **IDn18**. For acceptance filtering, only these bits together with MASK bits **MSKn28** to **MSKn18** are considered.

XTDn Extended Identifier
0: The 11-bit (“standard”) Identifier will be used for this Message Object
1: The 29-bit (“extended”) Identifier will be used for this Message Object

MDIRn Mask Message Direction
0: The message direction bit (DIR) has no effect on the acceptance filtering
1: The message direction bit (DIR) is used for acceptance filtering
Note: The Arbitration Registers **IDn28-00**, **XTDn**, and **DIRn** are used to define the identifier and type of outgoing messages and are used (together with the mask registers **MSKn28-00**, **MXTDn**, and **MDIRn**) for acceptance filtering of incoming messages. A received message is stored into the valid Message Object with matching identifier and Direction=receive (Data Frame) or Direction=transmit (Remote Frame). Extended frames can be stored only in Message Objects with **XTDn**=one, standard frames in Message Objects with **XTDn**=zero. If a received message (Data Frame or Remote Frame) matches with more than one valid Message Object, it is stored into that with the lowest message number. For details see chapter Acceptance Filtering of Received Messages.

DIRn Message Direction
0: Direction=receive: On **TREQ**, a Remote Frame with the identifier of this Message Object is transmitted. On reception of a Data Frame with matching identifier, that message is stored in this Message Object.
1: Direction=transmit: On **TREQ**, the respective Message Object is transmitted as a Data Frame. On reception of a Remote Frame with matching identifier, the **TREQ** bit of this Message Object is set (if **RMTnEN**=one).

UMASKn Use Acceptance Mask
0: MASK ignored.
1: Use MASK (MSKn28~00, MXTDn and MDIRn) for acceptance filtering.
If the **UMASKn** bit is set to one, the Message Object's mask bits have to be programmed during initialization of the Message Object before **MSGnVA** is set to one.

MSGnLST Message Lost (only valid for Message Objects with direction=receive)
0: No message lost since last time this bit was reset by the CPU.
1: The Message Handler stored a new message into this object when **NDTA** was still set, the CPU has lost a message.

TXnIEN Transmit Interrupt Enable
0: **INTPND** will be left unchanged after the successful transmission of a frame.
1: **INTPND** will be set after a successful transmission of a frame.

RXnIEN Receive Interrupt Enable
0: **INTPND** will be left unchanged after a successful reception of a frame.
1: **INTPND** will be set after a successful reception of a frame.

RMTnEN Remote Enable
0: At the reception of a Remote Frame, **TREQ** is left unchanged.
1: At the reception of a Remote Frame, **TREQ** is set.

EOBn End of Buffer
0: Message Object belongs to a FIFO Buffer and is not the last Message Object of that FIFO Buffer.
1: Single Message Object or last Message Object of a FIFO Buffer.
This bit is used to concatenate two ore more Message Objects (up to 32) to build a FIFO Buffer.
For single Message Objects (not belonging to a FIFO Buffer) this bit must always be set to one.

DLCn3~0 Data Length Code
0~8: CAN: Frame has 0-8 data bytes
9~15: CAN: Frame has 8 data bytes
Note: The Data Length Code of a Message Object must be defined the same as in all the corresponding objects with the same identifier at other nodes. When the Message Handler stores a data frame, it will write the DLC to the value given by the received message.

DATA0 1st data byte of a CAN Data Frame
DATA1 2nd data byte of a CAN Data Frame
DATA2 3rd data byte of a CAN Data Frame
DATA3 4th data byte of a CAN Data Frame
DATA4 5th data byte of a CAN Data Frame
DATA5 6th data byte of a CAN Data Frame
DATA6 7th data byte of a CAN Data Frame
DATA7 8th data byte of a CAN Data Frame

Note: Byte **DATA0** is the first data byte shifted into the shift register of the CAN Core during a reception, byte **DATA7** is the last. When the Message Handler stores a Data Frame, it will write all the eight data bytes into a Message Object. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by **non specified values**.

The 32 Message Objects can be configurated to several sets of FIFO buffer. A FIFO buffer can have a single Message Object or several concatenated Message Objects. The FIFO threshold of the Message Object number is determined by the RMFD[4:0] bits in the HT45B3305H CAN Configuration Register, CANCFG. When the Message Object of the selected number is received successfully, an interrupt active signal will output on the RMXINT pin.

HT45B3305H CAN Operating Modes

The Operating modes can be controlled by the registers. Detailed informations about the operating modes refer to the following contents and the related registers.

Software Initialization

The software initialization is started by setting the bit **INIT** in the CAN Control Register, either by software or by a hardware reset, or by going Bus_Off.

While **INIT** is set, all message transfered from and to the CAN bus is stopped, the status of the CAN bus output **can_tx** is recessive (HIGH). The counters of the EML(Error Management Logic) are unchanged. Setting **INIT** does not change any configuration register.

To initialize the CAN Controller, the CPU has to set up the Bit Timing Register and each Message Object. If a Message Object is not needed, it is sufficient to set it's **MSGnVA** bit to not valid. Otherwise, the whole Message Object has to be initialized.

Access to the Bit Timing Register and to the BRP Extension Register for the configuration of the bit timing is enabled when both bits **INIT** and **CCE** in the CAN Control Register are set.

Resetting **INIT** (by CPU only) finishes the software initialization. Afterwards the Bit Stream Processor(BSP) synchronizes itself to the data transfer on the CAN bus by waiting for the occurrence of a sequence of 11 consecutive recessive bits ($\equiv$ Bus Idle) before it can take part in bus activities and starts the message transfer.

The initialization of the Message Objects is independent of **INIT** and can be done on the fly, but the Message Objects should all be configured to particular identifiers or set to not valid before the BSP starts the message transfer. To change the configuration of a Message Object during normal operation, the CPU has to start by setting **MSGnVA** to not valid. When the configuration is completed, **MSGnVA** is set to valid again.

CAN Message Transfer

Once the HT45B3305H CAN is initialized and **INIT** is reset to zero, the HT45B3305H CAN's CAN Core synchronizes itself to the CAN bus and starts the message transfer.

Received messages are stored into their appropriate Message Objects if they pass the Message Handler's acceptance filtering. The whole message including all arbitration bits, DLC and eight data bytes is stored into the Message Object. If the Identifier Mask is used, the arbitration bits which are masked to "don't care" may be overwritten in the Message Object.

The CPU may read or write each message any time via the Interface Registers, the Message Handler guarantees data consistency in case of concurrent accesses.

Messages to be transmitted are updated by the CPU. If a permanent Message Object (arbitration and control bits set up during configuration) exists for the message, only the data bytes are updated and then **TQnDTA** bit is set to start the transmission. If several transmit messages are assigned to the same Message Object (when the number of Message Objects is not sufficient), the whole Message Object has to be configured before the transmission of this message is requested.

The transmission of any number of Message Objects may be requested at the same time, they are transmitted subsequently according to their internal priority. Messages may be updated or set to not valid any time, even when their requested transmission is still pending. The old data will be discarded when a message is updated before its pending transmission has started.

Depending on the configuration of the Message Object, the transmission of a message may be requested autonomously by the reception of a remote frame with a matching identifier.

Note: Remote frames are always transmitted in Classical CAN format.

Disabled Automatic Retransmission

According to the CAN Specification (see ISO11898-1, 6.3.3 Recovery Management), the HT45B3305H CAN provides means for automatic retransmission of frames that have lost arbitration or that have been disturbed by errors during transmission. The frame transmission service will not be confirmed to the user before the transmission is successfully completed. By default, this means for automatic retransmission is enabled.

The Disabled Automatic Retransmission mode is enabled by programming bit **DAR** in the CAN Control Register to '1'. In this operation mode the programmer has to consider the different behaviour of bits **TREQ** and **NDTA** in the Control Registers of the Message Buffers:

- When a transmission starts, bit **TREQ** of the respective Message Buffer is reset, while bit **NDTA** remains set.
- When the transmission completed successfully bit **NDTA** is reset.

When a transmission failed (lost arbitration or error) bit **NDTA** remains set. To restart the transmission the CPU has to set **TREQ** back to '1'.

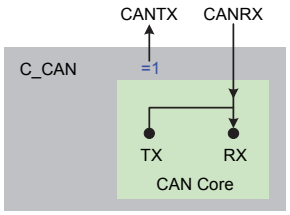
Test Mode

The Test Mode is entered by setting bit **TEST** in the CAN Control Register to one. In Test Mode the bits **TX1**, **TX0**, **LBACK**, **SILENT** and **BASIC** in the Test Register are writable. Bit **RX** monitors the state of pin **CANRX** and therefore is only readable. All Test Register functions are disabled when bit **TEST** is reset to zero. The Test Mode functions as described in the following subsections are intended for device tests outside normal operation. These functions should be used carefully. Switching between Test Mode functions and normal operation while communication is running (**INIT**='0') should be avoided.

Silent Mode

In ISO 11898-1, the Silent Mode is called the Bus Monitoring Mode. The CAN Core can be set in Silent Mode by programming the Test Register bit SILENT to '1'.

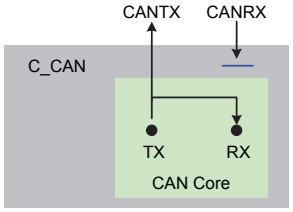
In Silent Mode, the HT45B3305H CAN is able to receive valid data frames and valid remote frames, but it sends only recessive bits on the CAN bus and it cannot start a transmission. If the CAN Core is required to send a dominant bit (ACK bit, overload flag, active error flag), the bit is rerouted internally so that the CAN Core monitors this dominant bit, although the CAN bus may remain in recessive state. The Silent Mode can be used to analyse the traffic on a CAN bus without affecting it by the transmission of dominant bits (Acknowledge Bits, Error Frames).



CAN Core in Silent Mode

Loop Back Mode

The CAN Core can be set in Loop Back Mode by programming the Test Register bit LBACK to '1'. In Loop Back Mode, the CAN Core treats its own transmitted messages as received messages and stores them (if they pass acceptance filtering) into a Receive Buffer.

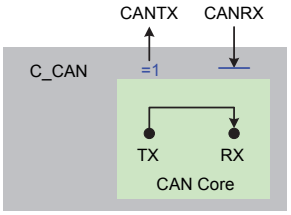


CAN Core in Loop Back Mode

This mode is provided for self-test functions. To be independent from external stimulation, the CAN Core ignores acknowledge errors (recessive bit sampled in the acknowledge slot of a data/remote frame) in Loop Back Mode. In this mode the CAN Core performs an internal feedback from its TX output to its RX input. The actual value of the **CANRX** input pin is disregarded by the CAN Core. The transmitted messages can be monitored at the **CANTX** pin.

Loop Back combined with Silent Mode

It is also possible to combine Loop Back Mode and Silent Mode by programming both bits LBACK and SILENT to '1' at the same time. This mode can be used for a "Hot Selftest", meaning the CAN can be tested without affecting a running CAN system connected to the pins CANTX and CANRX. In this mode the CANRX pin is disconnected from the CAN Core and the CANTX pin is held recessive.



CAN Core in Loop Back combined with Silent Mode

Basic Mode

The CAN Core can be set in Basic Mode by programming the Test Register bit **BASIC** to '1'. In this mode the CAN module runs without the Message RAM.

The IF1 Registers are used as Transmit Buffer. The transmission of the contents of the IF1 Registers is requested by writing the **BUSYn** bit of the IF1 Command Request Register to '1'. The IF1 Registers are locked while the **BUSYn** bit is set. The **BUSYn** bit indicates that the transmission is pending. As soon the CAN bus is idle, the IF1 Registers are loaded into the shift register of the CAN Core and the transmission is started. When the transmission has completed, the **BUSYn** bit is reset and the locked IF1 Registers are released. A pending transmission can be aborted at any time by resetting the **BUSYn** bit in the IF1 Command Request Register while the IF1 Registers are locked. If the CPU has reset the **BUSYn** bit, a possible retransmission in case of lost arbitration or in case of an error is disabled.

The IF2 Registers are used as Receive Buffer. After the reception of a message the contents of the shift register is stored into the IF2 Registers, without any acceptance filtering. Additionally, the actual contents of the shift register can be monitored during the message transfer. Each time a read Message Object is initiated by writing the **BUSYn** bit of the IF2 Command Request Register to '1', the contents of the shift register is stored into the IF2 Registers.

In Basic Mode the evaluation of all Message Object related control and status bits and of the control bits of the IFn Command Mask Registers is turned off. The message number of the Command request registers is not evaluated. The **NnDTA** and **MSGnLST** bits of the IF2 Message Control Register retain their function, **DLCn3~DLCn0** will show the received **DLC**(Data Length Code), the other control bits will be read as '0'.

In Basic Mode the ready output **can_wait_b** is not active.

Software control of Pin CANTX

Four output functions are available for the CAN transmit pin CANTX. Additionally to its default function – the serial data output – it can drive the CAN Sample Point signal to monitor CAN_Core's bit timing and it can drive constant dominant or recessive values. The last two functions, combined with the readable CAN receive pin CANRX, can be used to check the CAN bus' physical layer.

The output mode of pin CANTX is selected by programming the Test Register bits **TX1** and **TX0**. The three test functions for pin CANTX interfere with all CAN protocol functions. CANTX must be left in its default function when CAN message transfer or any of the test modes Loop Back Mode, Silent Mode, or Basic Mode are selected.

CAN Application

Management of Message Objects

The configuration of the Message Objects in the Message RAM will (with the exception of the bits **MSGVA**, **NDTA**, **INTPND**, and **TREQ**) not be affected by resetting the CAN. All the Message Objects must be initialized by the CPU or they must be set not valid (**MSGVA**=0'). The bit timing must be configured before the CPU clears the **INIT** bit in the CAN Control Register.

The configuration of a Message Object is done by programming Mask, Arbitration, Control and Data field of one of the two interface register sets to the desired values. By writing to the corresponding IFn Command Request Register, the IFn Message Buffer Registers are loaded into the addressed Message Object in the Message RAM.

When the **INIT** bit in the CAN Control Register is cleared, the CAN Protocol Controller state machine of the CAN_Core and the Message Handler State Machine control the CAN's internal data flow. Received messages that pass the acceptance filtering are stored into the Message RAM, messages with pending transmission request are loaded into the CAN_Core's Shift Register and are transmitted via the CAN bus.

The CPU reads received messages and updates messages to be transmitted via the IFn Interface Registers. Depending on the configuration, the CPU is interrupted on certain CAN message and CAN error events.

Message Handler State Machine

The Message Handler controls the data transfer between the Rx/Tx Shift Register of the CAN Core, the Message RAM and the IFn Registers.

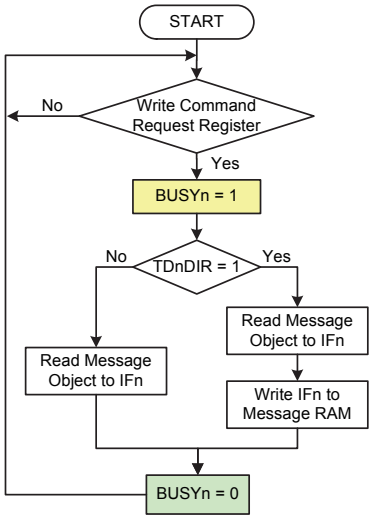
The Message Handler FSM(Finite State Machine) controls the following functions:

- Data Transfer from IFn Registers to the Message RAM
- Data Transfer from Message RAM to the IFn Registers
- Data Transfer from Shift Register to the Message RAM
- Data Transfer from Message RAM to Shift Register
- Data Transfer from Shift Register to the Acceptance Filtering unit
- Scanning of Message RAM for a matching Message Object
- Handling of **TREQ** flags
- Handling of interrupts

Data Transfer from / to Message RAM

When the CPU initiates a data transfer between the IFn Registers and Message RAM, the Message Handler sets the **BUSYn** bit in the respective IFn Command Request Register to '1'. After the transfer has completed, the **BUSYn** bit is set back to '0'.

The respective IFn Command Mask Register specifies whether a complete Message Object or only parts of it will be transferred. Due to the structure of the Message RAM it is not possible to write single bits/bytes of one Message Object, it is always necessary to write a complete Message Object to the Message RAM. Therefore the data transfer from the IFn Message Buffer Registers to the Message RAM (**TDnDIR**="1") requires a read-modify-write cycle. First those parts of the Message Object that are not to be changed are read from the Message RAM to the selected IFn Message Buffer Registers and then the complete contents of the selected IFn Message Buffer Registers are written to the Message Object.



Data Transfer between IFn Registers and Message RAM

After a partial write of a Message Object (**TDnDIR**="1"), the IFn Message Buffer Registers that are not selected by the respective IFn Command Mask Register will be set to the actual contents of the selected Message Object.

After a partial read of a Message Object (**TDnDIR**= "0"), the IFn Message Buffer Registers that are not selected by the respective IFn Command Mask Register will be left unchanged.

Transmission of Messages

If the shift register of the CAN Core cell is ready for loading and if there is no data transfer between the IFn Registers and Message RAM, the MSGVA bits in the Message Valid Register and the TREQ bits in the Transmission Request Register are evaluated. The valid Message Object with the highest priority pending transmission request is loaded into the shift register by the Message Handler and the transmission is started. The Message Object's **NDTA** bit is reset.

After a successful transmission and if no new data was written to the Message Object (**NDTA** ='0') since the start of the transmission, the **TREQ** bit will be reset. If **TXnIEN** is set, **INTPND** will be set after a successful transmission. If the CAN has lost the arbitration or if an error occurred during the transmission, the message will be retransmitted as soon as the CAN bus is free again. If meanwhile the transmission of a message with higher priority has been requested, the messages will be transmitted in the order of their priority.

Acceptance Filtering of Received Messages

When the arbitration and control field (Identifier + DLC) of an incoming message is completely shifted into the Rx/Tx Shift Register of the CAN Core, the Message Handler FSM (Finite State Machine) starts the scanning of the Message RAM for a matching valid Message Object.

To scan the Message RAM for a matching Message Object, the Acceptance Filtering unit is loaded with the arbitration bits from the CAN Core shift register. Then the arbitration and mask fields (including **MSGVA**, **UMASKn**, **NDTA**, and **EOBn**) of Message Object 1 are loaded into the Acceptance Filtering unit and are compared with the arbitration field from the shift register. This is repeated with each following Message Object until a matching Message Object is found or until the end of the Message RAM is reached.

If a match occurs, the scanning is stopped and the Message Handler FSM (Finite State Machine) proceeds depending on the type of frame (Data Frame or Remote Frame) received.

Reception of Data Frame

The Message Handler FSM (Finite State Machine) stores the message from the CAN Core shift register into the respective Message Object in the Message RAM. Not only the data bytes, but also all arbitration bits and the Data Length Code are stored into the corresponding Message Object. This is implemented to keep the data bytes connected with the identifier even if arbitration mask registers are used.

The **NDTA** bit is set to indicate that new data (not yet seen by the CPU) has been received. The CPU should reset **NDTA** when it reads the Message Object. If at the time of the reception, the **NDTA** bit was already set, **MSGLST** is set to indicate that the previous data (supposedly not seen by the CPU) is lost. If the **RXnIE** bit is set, the **INTPND** bit is set, causing the Interrupt Register to point to this Message Object.

The **TREQ** bit of this Message Object is reset to prevent the transmission of a Remote Frame, while the requested Data Frame has just been received.

Reception of Remote Frame

When a Remote Frame is received, three different configurations of the matching Message Object have to be considered:

1) **DIRn**='1' (direction=transmit), **RMTnEN**='1', **UMASKn**='1' or '0'
   At the reception of a matching Remote Frame, the **TREQ** bit of this Message Object is set high. The rest of the Message Object remains unchanged.
2) **DIRn**='1' (direction=transmit), **RMTnEN**='0', **UMASKn**='0'
   At the reception of a matching Remote Frame, the **TREQ** bit of this Message Object remains unchanged; the Remote Frame is ignored.
3) **DIRn**='1' (direction=transmit), **RMTnEN**='0', **UMASKn**='1'

At the reception of a matching Remote Frame, the **TREQ** bit of this Message Object is reset. The arbitration and control field (Identifier + IDE + RTR + DLC) from the shift register is stored into the Message Object in the Message RAM and the **NDTA** bit of this Message Object is set high.

Note: Remote frames are always transmitted in Classical CAN format.

Receive / Transmit Priority

The receive/transmit priority for the Message Objects is attached to the message number. Message Object 1 has the highest priority, while Message Object 32 has the lowest priority. If more than one transmission request is pending, they are serviced according to the priority of the corresponding Message Object.

Configuration of a Transmit Object

MSGnVA	ARBn	—	DATA	MASK	EOBn	DIRn
1	appl.	—	appl.	appl.	1	1
NnDTA	**MSGnLST**	**RXnIEN**	**TXnIEN**	**INTnPND**	**RMTnEN**	**TnREQ**
0	0	0	appl.	0	appl.	0

Note: "**appl.**" means by application.

Initialisation of a Transmit Object

The Arbitration Registers (**IDn28~00** and **XTDn** bit) are given by the application. They define the identifier and type of the outgoing message. If an 11-bit Identifier ("Standard Frame") is used, it is programmed to **IDn28~IDn18**, **IDn17~IDn00** can then be disregarded.

If the **TXnIEN** bit is set, the INTPND bit will be set after a successful transmission of the Message Object.

If the **RMTnEN** bit is set, a matching received Remote Frame will cause the **TREQ** bit to be set; the Remote Frame will autonomously be answered by a Data Frame.

The Data Registers (**DLCn3-0, DATA0-7**) are given by the application, **TnREQ** and **RMTnEN** may not be set before the data is valid.

The Mask Registers (**MSKn28-00, UMASKn, MXTDn** and **MDIRn** bits) may be used (**UMASKn**='1') to allow groups of Remote Frames with similar identifiers to set the **TnREQ** bit. The **DIRn** bit should not be masked.

Updating a Transmit Object

The CPU may update the data bytes of a Transmit Object any time via the IFn Interface registers, neither **MSGVA** nor **TREQ** have to be reset before the update.

Even if only a part of the data bytes are to be updated, all four bytes of the corresponding IFn DATAnA Register or IFn DATAnB Register have to be valid before the content of that register is transferred to the Message Object. Either the CPU has to write all four bytes into the IFn Data Register or the Message Object is transferred to the IFn Data Register before the CPU writes the new data bytes.

When only the (eight) data bytes are updated, first 0x87 is written to the IFn Command Mask Register and then the number of the Message Object is written to the IFn Command Request Register, concurrently updating the data bytes and setting **TQnDTA**.

To prevent the reset of **TREQ** at the end of a transmission that may already be in progress while the data is updated, **NDTA** has to be set together with **TREQ**.

When **NDTA** is set together with **TREQ**, **NDTA** will be reset as soon as the new transmission has started.

Configuration of a Receive Object

MSGnVA	ARBn	—	DATA	MASK	EOBn	DIRn
1	appl.	—	appl.	appl.	1	0
NnDTA	MSGnLST	RXnIEN	TXnIEN	INTnPND	RMTnEN	TnREQ
0	0	appl.	0	0	0	0

Note: "appl." means by application.

Initialisation of a Receive Object

The Arbitration Registers (**IDn[28:00]** and **XTDn** bit) are given by the application. They define the identifier and type of accepted received messages. If an 11-bit Identifier ("Standard Frame") is used, it is programmed to **IDn28~IDn18, IDn17~IDn00** can then be disregarded. When a Data Frame with an 11-bit Identifier is received, **IDn17~IDn00** will be set to '0'.

If the **RXnIEN** bit is set, the **INTPND** bit will be set when a received Data Frame is accepted and stored in the Message Object.

The Data Length Code (**DLCn[3:0]**) is given by the application. When the Message Handler stores a Data Frame in the Message Object, it will store the received Data Length Code and eight data bytes. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by **non specified values**.

The Mask Registers (**MSKn[28:00], UMASKn, MXTDn**, and **MDIRn** bits) may be used (**UMASKn**='1') to allow groups of Data Frames with similar identifiers to be accepted . The **DIRn** bit should not be masked in typical applications.

Handling of Received Messages

The CPU may read a received message any time via the IFn Interface registers. The data consistency is guaranteed by the Message Handler state machine.

Typically the CPU will write first **0x7F** to the IFn Command Mask Register and then the number of the Message Object to the IFn Command Request Register. That combination will transfer the whole received message from the Message RAM into the IFn Message Buffer Register. Additionally, the bits **NDTA** and **INTPND** are cleared in the Message RAM (not in the Message Buffer).

If the Message Object uses masks for acceptance filtering, the arbitration bits show which of the matching messages has been received.

The actual value of **NDTA** shows whether a new message has been received since last time this Message Object was read. The actual value of **MSGLST** shows whether more than one message has been received since last time this Message Object was read. **MSGLST** will not be automatically reset.

By means of a Remote Frame, the CPU may request another CAN node to provide new data for a receive object. Setting the **TREQ** bit of a receive object will cause the transmission of a Remote Frame with the receive object's identifier. This Remote Frame triggers the other CAN node to start the transmission of the matching Data Frame. If the matching Data Frame is received before the Remote Frame could be transmitted, the **TREQ** bit is automatically reset.

Configuration of a FIFO Buffer

With the exception of the **EOBn** bit, the configuration of Receive Objects belonging to a FIFO Buffer is the same as the configuration of a (single) Receive Object.

To concatenate two or more Message Objects into a FIFO Buffer, the identifiers and masks (if used) of these Message Objects have to be programmed to matching values. Due to the implicit priority of the Message Objects, the Message Object with the lowest number will be the first Message Object of the FIFO Buffer. The **EOBn** bit of all Message Objects of a FIFO Buffer except the last have to be programmed to zero. The EOBn bits of the last Message Object of a FIFO Buffer is set to one, configuring it as the End of the Block.

Reception of Messages with FIFO Buffers

Received messages with identifiers matching to a FIFO Buffer are stored into a Message Object of this FIFO Buffer starting with the Message Object with the lowest message number.

When a message is stored into a Message Object of a FIFO Buffer the **NDTA** bit of this Message Object is set. By setting **NDTA** while **EOBn** is zero the Message Object is locked for further write accesses by the Message Handler until the CPU has written the **NDTA** bit back to zero.

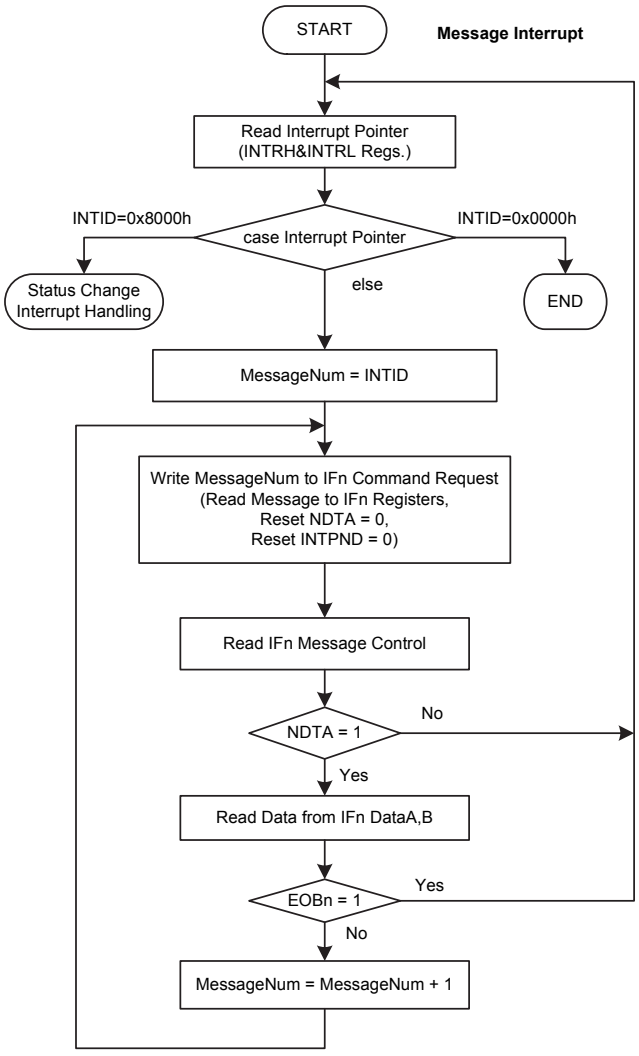
Messages are stored into a FIFO Buffer until the last Message Object of this FIFO Buffer is reached. If none of the preceding Message Objects is released by writing **NDTA** to zero, all further messages for this FIFO Buffer will be written into the last Message Object of the FIFO Buffer and therefore overwrite previous messages.

Reading from a FIFO Buffer

When the CPU transfers the contents of Message Object to the IFn Message Buffer Registers by writing its number to the IFn Command Request Register, the corresponding IFn Command Mask Register should be programmed the way that bits **NDTA** and **INTPND** are reset to zero (**TQnDTA**='1' and **CINTPNDn**='1'). The values of these bits in the IFn Message Control Register always reflect the status before resetting the bits.

To assure the correct function of a FIFO Buffer, the CPU should read out the Message Objects starting at the FIFO Object with the lowest message number.

The following figure shows how a set of Message Objects which are concatenated to a FIFO Buffer can be handled by the CPU.



CPU Handling of a FIFO Buffer

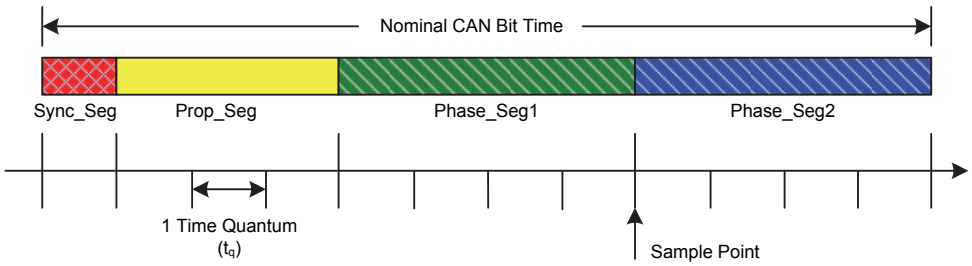
Bit Time and Bit Rate

CAN supports bit rates in the range of 1 kBit/s to 1000 kBit/s. Each member of the CAN network has its own clock generator, usually a quartz oscillator. The timing parameter of the bit time (i.e. the reciprocal of the bit rate) can be configured individually for each CAN node, creating a common bit rate even though the CAN nodes' oscillator periods (f_{OSC}) may be different.

The frequencies of these oscillators are not absolutely stable, small variations are caused by changes in temperature or voltage and by deteriorating components. As long as the variations remain inside a specification oscillator tolerance range (d_F), the CAN nodes are able to compensate for the different bit rates by resynchronising to the bit stream.

According to the CAN specification, the bit time is divided into four segments which are the Synchronisation Segment, the Propagation Time Segment, the Phase Buffer Segment 1 and the Phase Buffer Segment 2. Each segment consists of a specification, programmable number of time quanta. The length of the time quantum (t_q), which is the basic time unit of the bit time, is defined by the CAN controller's system clock f_{SYS} and the Baud Rate Prescaler (BRP): t_q=BRP / f_{SYS}. The CAN's system clock f_{SYS} is the frequency of its can_clk input.

The Synchronisation Segment Sync_Seg is that part of the bit time where edges of the CAN bus level are expected to occur; the distance between an edge that occurs outside of Sync_Seg and the Sync_Seg is called the phase error of that edge. The Propagation Time Segment Prop_Seg is intended to compensate for the physical delay times within the CAN network. The Phase Buffer Segments Phase_Seg1 and Phase_Seg2 surround the Sample Point. The (Re-) Synchronisation Jump Width (SJW) defines how far a resynchronisation may move the Sample Point inside the limits defined by the Phase Buffer Segments to compensate for edge phase errors.



Bit Timing

Parameter	Range	Remark
BRP	[1 .. 32]	Defines the length of the time quantum t_q
Sync_Seg	1 t_q	Fixed length, synchronisation of bus input to CAN system clock
Prop_Seg	[1 .. 8] t_q	Compensates for the physical delay times
Phase_Seg1	[1 .. 8] t_q	May be lengthened temporarily by synchronisation
Phase_Seg2	[1 .. 8] t_q	May be shortened temporarily by synchronisation
SJW	[1 .. 4] t_q	May not be longer than either Phase Buffer Segment
This table describes the minimum programmable ranges required by the CAN protocol		

mtq (minimum time quantum) = system clock period = $1/f_{SYS}$

t_q (time quantum) = (BRPE[3:0] × 0x40 + BRP[5:0] + 1) × mtq

SYNC_SEG = 1 t_q

SEG1 = PROP_SEG + PHASE_SEG1

Bit Time = t_{SYNC_SEG} + t_{SEG1} + t_{PHASE_SEG2}

For example:

f_{SYS}=8MHz , Bit Rate=500Kbps , PROP_SEG=0 , Sample point=50% , SYNC_SEG=1 t_q, then to calculate the SEG1 & PHASE_SEG2 values.

Sol: mtq = $1/f_{SYS}$ = 1/8MHz = 0.125μs

set Baud Rate Prescaler(BRP) = 1 → BRP[5:0] = (1−1) = $0_{\#}$

t_q = (BRPE[3:0] • 0x40 + BRP[5:0]+1) • mtq = 1 • mtq = 0.125μs

Bit Time = 1/Bit Rate = 1/500Kbps = 0.002ms = 2μs

Nominal Bit Time = Bit Rate/tq = 2μs/(0.125μs) = 16tq

(1) PHASE_SEG2 = Nominal Bit Time − (Nominal Bit Time • Sample point) = $16t_q$ − ($16t_q$ • 50%) = $16t_q$ − $8t_q$ = $8t_q$

TSG2D[2:0] = (8−1) = $7_{\#}$

(2) SEG1=(Nominal Bit Time − SYNC_SEG − PHASE_SEG2)=($16t_q$–$1t_q$–$8t_q$)=$7t_q$

TSG1D[3:0]=(7−1)=$6_{\#}$

Register Description

The device is controlled using a series of registers which are described in the following section.

Register Map

The following shows the full register bit map of the device.

Note that the symbol "–" represents an unimplemented bit which is read as zero.

Address	Register Name	Bit							
		7	6	5	4	3	2	1	0
0x00	CTRLRL	TEST	CCE	DAR	—	EIE	SIE	CANIE	INIT
0x02	STATRL	BOFF	EWARN	EPASS	RXOK	TXOK	LEC2	LEC1	LEC0
0x04	ERRCNTL	TEC7	TEC6	TEC5	TEC4	TEC3	TEC2	TEC1	TEC0
0x05	ERRCNTH	RP	REC6	REC5	REC4	REC3	REC2	REC1	REC0
0x06	BTRL	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
0x07	BTRH	—	TSG2D2	TSG2D1	TSG2D0	TSG1D3	TSG1D2	TSG1D1	TSG1D0
0x08	INTRL	INTID7	INTID6	INTID5	INTID4	INTID3	INTID2	INTID1	INTID0
0x09	INTRH	INTID15	INTID14	INTID13	INTID12	INTID11	INTID10	INTID9	INTID8
0x0A	TESTRL	RX	TX1	TX0	LBACK	SILENT	BASIC	—	—
0x0C	BRPERL	—	—	—	—	BRPE3	BRPE2	BRPE1	BRPE0
0x10	IF1CREQL	—	—	MSG1N5	MSG1N4	MSG1N3	MSG1N2	MSG1N1	MSG1N0
0x11	IF1CREQH	BUSY1	—	—	—	—	—	—	—
0x12	IF1CMSKL	TD1DIR	MASK1	ARB1	CTRL1	CINTPND1	TQ1DTA	DATA1A	DATA1B
0x14	IF1MSK1L	MSK107	MSK106	MSK105	MSK104	MSK103	MSK102	MSK101	MSK100
0x15	IF1MSK1H	MSK115	MSK114	MSK113	MSK112	MSK111	MSK110	MSK109	MSK108
0x16	IF1MSK2L	MSK123	MSK122	MSK121	MSK120	MSK119	MSK118	MSK117	MSK116
0x17	IF1MSK2H	MXTD1	MDIR1	—	MSK128	MSK127	MSK126	MSK125	MSK124
0x18	IF1ARB1L	ID107	ID106	ID105	ID104	ID103	ID102	ID101	ID100
0x19	IF1ARB1H	ID115	ID114	ID113	ID112	ID111	ID110	ID109	ID108
0x1A	IF1ARB2L	ID123	ID122	ID121	ID120	ID119	ID118	ID117	ID116
0x1B	IF1ARB2H	MSG1VA	XTD1	DIR1	ID128	ID127	ID126	ID125	ID124
0x1C	IF1MCTRL	EOB1	—	—	—	DLC13	DLC12	DLC11	DLC10
0x1D	IF1MCTRH	N1DTA	MSG1LST	INT1PND	UMASK1	TX1IEN	RX1IEN	RMT1EN	T1REQ
0x1E	IF1DTA1L	D7	D6	D5	D4	D3	D2	D1	D0

Address	Register Name	Bit							
		7	6	5	4	3	2	1	0
0x1F	IF1DTA1H	D7	D6	D5	D4	D3	D2	D1	D0
0x20	IF1DTA2L	D7	D6	D5	D4	D3	D2	D1	D0
0x21	IF1DTA2H	D7	D6	D5	D4	D3	D2	D1	D0
0x22	IF1DTB1L	D7	D6	D5	D4	D3	D2	D1	D0
0x23	IF1DTB1H	D7	D6	D5	D4	D3	D2	D1	D0
0x24	IF1DTB2L	D7	D6	D5	D4	D3	D2	D1	D0
0x25	IF1DTB2H	D7	D6	D5	D4	D3	D2	D1	D0
0x26~37	Note: Reserved, cannot be changed								
0x38	CRLL	DAY7	DAY6	DAY5	DAY4	DAY3	DAY2	DAY1	DAY0
0x39	CRLH	MON7	MON6	MON5	MON4	MON3	MON2	MON1	MON0
0x3A	CRHL	SUBSTEP3	SUBSTEP2	SUBSTEP1	SUBSTEP0	YEAR3	YEAR2	YEAR1	YEAR0
0x3B	CRHH	REL3	REL2	REL1	REL0	STEP3	STEP2	STEP1	STEP0
0x40	IF2CREQL	—	—	MSG2N5	MSG2N4	MSG2N3	MSG2N2	MSG2N1	MSG2N0
0x41	IF2CREQH	BUSY2	—	—	—	—	—	—	—
0x42	IF2CMSKL	TD2DIR	MASK2	ARB2	CTRL2	CINTPND2	TQ2DTA	DATA2A	DATA2B
0x44	IF2MSK1L	MSK207	MSK206	MSK205	MSK204	MSK203	MSK202	MSK201	MSK200
0x45	IF2MSK1H	MSK215	MSK214	MSK213	MSK212	MSK211	MSK210	MSK209	MSK208
0x46	IF2MSK2L	MSK223	MSK222	MSK221	MSK220	MSK219	MSK218	MSK217	MSK216
0x47	IF2MSK2H	MXTD2	MDIR2	—	MSK228	MSK227	MSK226	MSK225	MSK224
0x48	IF2ARB1L	ID207	ID206	ID205	ID204	ID203	ID202	ID201	ID200
0x49	IF2ARB1H	ID215	ID214	ID213	ID212	ID211	ID210	ID209	ID208
0x4A	IF2ARB2L	ID223	ID222	ID221	ID220	ID219	ID218	ID217	ID216
0x4B	IF2ARB2H	MSG2VA	XTD2	DIR2	ID228	ID227	ID226	ID225	ID224
0x4C	IF2MCTRL	EOB2	—	—	—	DLC23	DLC22	DLC21	DLC20
0x4D	IF2MCTRH	N2DTA	MSG2LST	INT2PND	UMASK2	TX2IEN	RX2IEN	RMT2EN	T2REQ
0x4E	IF2DTA1L	D7	D6	D5	D4	D3	D2	D1	D0
0x4F	IF2DTA1H	D7	D6	D5	D4	D3	D2	D1	D0
0x50	IF2DTA2L	D7	D6	D5	D4	D3	D2	D1	D0
0x51	IF2DTA2H	D7	D6	D5	D4	D3	D2	D1	D0
0x52	IF2DTB1L	D7	D6	D5	D4	D3	D2	D1	D0
0x53	IF2DTB1H	D7	D6	D5	D4	D3	D2	D1	D0
0x54	IF2DTB2L	D7	D6	D5	D4	D3	D2	D1	D0
0x55	IF2DTB2H	D7	D6	D5	D4	D3	D2	D1	D0
0x80	TREQR1L	TREQ8	TREQ7	TREQ6	TREQ5	TREQ4	TREQ3	TREQ2	TREQ1
0x81	TREQR1H	TREQ16	TREQ15	TREQ14	TREQ13	TREQ12	TREQ11	TREQ10	TREQ9
0x82	TREQR2L	TREQ24	TREQ23	TREQ22	TREQ21	TREQ20	TREQ19	TREQ18	TREQ17
0x83	TREQR2H	TREQ32	TREQ31	TREQ30	TREQ29	TREQ28	TREQ27	TREQ26	TREQ25
0x90	NEWDT1L	NDTA8	NDTA7	NDTA6	NDTA5	NDTA4	NDTA3	NDTA2	NDTA1
0x91	NEWDT1H	NDTA16	NDTA15	NDTA14	NDTA13	NDTA12	NDTA11	NDTA10	NDTA9
0x92	NEWDT2L	NDTA24	NDTA23	NDTA22	NDTA21	NDTA20	NDTA19	NDTA18	NDTA17
0x93	NEWDT2H	NDTA32	NDTA31	NDTA30	NDTA29	NDTA28	NDTA27	NDTA26	NDTA25
0xA0	INTPND1L	INTPND8	INTPND7	INTPND6	INTPND5	INTPND4	INTPND3	INTPND2	INTPND1
0xA1	INTPND1H	INTPND16	INTPND15	INTPND14	INTPND13	INTPND12	INTPND11	INTPND10	INTPND9
0xA2	INTPND2L	INTPND24	INTPND23	INTPND22	INTPND21	INTPND20	INTPND19	INTPND18	INTPND17
0xA3	INTPND2H	INTPND32	INTPND31	INTPND30	INTPND29	INTPND28	INTPND27	INTPND26	INTPND25
0xB0	MSGVAL1L	MSGVA8	MSGVA7	MSGVA6	MSGVA5	MSGVA4	MSGVA3	MSGVA2	MSGVA1
0xB1	MSGVAL1H	MSGVA16	MSGVA15	MSGVA14	MSGVA13	MSGVA12	MSGVA11	MSGVA10	MSGVA9
0xB2	MSGVAL2L	MSGVA24	MSGVA23	MSGVA22	MSGVA21	MSGVA20	MSGVA19	MSGVA18	MSGVA17
0xB3	MSGVAL2H	MSGVA32	MSGVA31	MSGVA30	MSGVA29	MSGVA28	MSGVA27	MSGVA26	MSGVA25
0xBF	CANCFG	D7	CANEN	—	RMFD4	RMFD3	RMFD2	RMFD1	RMFD0
0xC0	FOCFG	I2CDEB1	I2CDEB0	RMXFIV	RM1FIV	CANIV	HXTEN	FODIV1	FODIV0
0xC1	SFIOSTC	—	SOFT2	SOFT1	SOFT0	—	—	CLKHST	MISOHST

Register Reset Condition

A Reset function is a fundamental part of the device ensuring the device can be set to some predetermined condition irrespective of outside parameters.

To ensure reliable operation, it is important to know what condition the device registers is in after a Power on Reset or an external $\overline{\text{RES}}$ pin reset or receive a "Reset Can Block" instruction. The following table describes how the reset affects each of the internal registers.

Register	Power On Reset / $\overline{\text{RES}}$ Reset / "Reset CAN Block" Instruction Reset
CTRLRL	000- 0001
STATRL	0000 0000
ERRCNTL	0000 0000
ERRCNTH	0000 0000
BTRL	0000 0001
BTRH	-010 0011
INTRL	0000 0000
INTRH	0000 0000
TESTRL	x000 00--
BRPERL	---- 0000
IF1CREQL	--00 0001
IF1CREQH	0--- ----
IF1CMSKL	0000 0000
IF1MSK1L	1111 1111
IF1MSK1H	1111 1111
IF1MSK2L	1111 1111
IF1MSK2H	11-1 1111
IF1ARB1L	0000 0000
IF1ARB1H	0000 0000
IF1ARB2L	0000 0000
IF1ARB2H	0000 0000
IF1MCTRL	0--- 0000
IF1MCTRH	0000 0000
IF1DTA1L	0000 0000
IF1DTA1H	0000 0000
IF1DTA2L	0000 0000
IF1DTA2H	0000 0000
IF1DTB1L	0000 0000
IF1DTB1H	0000 0000
IF1DTB2L	0000 0000
IF1DTB2H	0000 0000
CRLL	0010 0111
CRLH	0000 0010
CRHL	0000 0101
CRHH	0010 0001
IF2CREQL	--00 0001
IF2CREQH	0--- ----
IF2CMSKL	0000 0000
IF2MSK1L	1111 1111
IF2MSK1H	1111 1111
IF2MSK2L	1111 1111

Register	Power On Reset / $\overline{RES}$ Reset / “Reset CAN Block” Instruction Reset
IF2MSK2H	11-1 1111
IF2ARB1L	0000 0000
IF2ARB1H	0000 0000
IF2ARB2L	0000 0000
IF2ARB2H	0000 0000
IF2MCTRL	0--- 0000
IF2MCTRH	0000 0000
IF2DTA1L	0000 0000
IF2DTA1H	0000 0000
IF2DTA2L	0000 0000
IF2DTA2H	0000 0000
IF2DTB1L	0000 0000
IF2DTB1H	0000 0000
IF2DTB2L	0000 0000
IF2DTB2H	0000 0000
TREQR1L	0000 0000
TREQR1H	0000 0000
TREQR2L	0000 0000
TREQR2H	0000 0000
NEWDT1L	0000 0000
NEWDT1H	0000 0000
NEWDT2L	0000 0000
NEWDT2H	0000 0000
INTPND1L	0000 0000
INTPND1H	0000 0000
INTPND2L	0000 0000
INTPND2H	0000 0000
MSGVAL1L	0000 0000
MSGVAL1H	0000 0000
MSGVAL2L	0000 0000
MSGVAL2H	0000 0000
CANCFG	10-0 0000
FOCFG	0000 0100
SFIOSTC	-000 --00

Table Legend: “-” Unimplemented
“x” Unknown

Register Description

The following is the detailed register description. The registers are used for different functions.

Programmer's Model

Register	Description	Note
CTRLRL	CAN Control Register	—
STATRL	Status Register	—
ERRCNTH/ ERRCNTL	Error Counter	Read only
BTRH/ BTRL	Bit Timing Register	Write enabled by CCE
INTRH/ INTRL	Interrupt Register	Read only
TESTRL	Test Register	Write enabled by TEST
BRPERL	BRP Extension Register	Write enabled by CCE
IF1CREQH/ IF1CREQL	IF1 Command Request	—
IF1CMSKL	IF1 Command Mask	—
IF1MSK1H/ IF1MSK1L	IF1 Mask 1	—
IF1MSK2H/ IF1MSK2L	IF1 Mask 2	—
IF1ARB1H/ IF1ARB1L	IF1 Arbitration 1	—
IF1ARB2H/ IF1ARB2L	IF1 Arbitration 2	—
IF1MCTRH/ IF1MCTRL	IF1 Message Control	—
IF1DTA1H/ IF1DTA1L	IF1 Data A 1	—
IF1DTA2H/ IF1DTA2L	IF1 Data A 2	—
IF1DTB1H/ IF1DTB1L	IF1 Data B 1	—
IF1DTB2H/ IF1DTB2L	IF1 Data B 2	—
IF2CREQH/ IF2CREQL	IF2 Command Request	—
IF2CMSKL	IF2 Command Mask	—
IF2MSK1H/ IF2MSK1L	IF2 Mask 1	—
IF2MSK2H/ IF2MSK2L	IF2 Mask 2	—
IF2ARB1H/ IF2ARB1L	IF2 Arbitration 1	—
IF2ARB2H/ IF2ARB2L	IF2 Arbitration 2	—
IF2MCTRH/ IF2MCTRL	IF2 Message Control	—
IF2DTA1H/ IF2DTA1L	IF2 Data A 1	—
IF2DTA2H/ IF2DTA2L	IF2 Data A 2	—
IF2DTB1H/ IF2DTB1L	IF2 Data B 1	—
IF2DTB2H/ IF2DTB2L	IF2 Data B 2	—
CRLH / CRLL	Core Release Low	Read only
CRHH/ CRHL	Core Release High	Read only
TREQR1H/ TREQR1L	Transmission Request 1	Read only
TREQR2H/ TREQR2L	Transmission Request 2	Read only
NEWDT1H/ NEWDT1L	New Data 1	Read only
NEWDT2H/ NEWDT2L	New Data 2	Read only
INTPND1H/ INTPND1L	Interrupt Pending 1	Read only
INTPND2H/ INTPND2L	Interrupt Pending 2	Read only
MSGVAL1H/ MSGVAL1L	Message Valid 1	Read only
MSGVAL2H/ MSGVAL2L	Message Valid 2	Read only
CANCFG	HT45B3305H CAN Configuration	—
FOCFG	Device Output Configuration	—
SFIOSTC	Output pin Configuration	—

Device Output Configuration Register

- **FOCFG Register (ADDRESS: C0H)**

Bit	7	6	5	4	3	2	1	0
Name	I2CDEB1	I2CDEB0	RMXFIV	RM1FIV	CANIV	HXTEN	FODIV1	FODIV0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	1	0	0

Bit 7~6 **I2CDEB1~I2CDEB0**: I2C Debounce Time Selection
00: No debounce
01: 2 system clock debounce
10: 4 system clock debounce
11: 4 system clock debounce
Note: When the device enters SLEEP Mode, the SCL and SDA lines will bypass debounce circuit.

Bit 5 **RMXFIV**: RMXINT interrupt output level selection
0: Active Low
1: Active High

Bit 4 **RM1FIV**: RM1INT interrupt output level selection
0: Active Low
1: Active High

Bit 3 **CANIV**: CANMINT interrupt output level selection
0: Active Low
1: Active High

Bit 2 **HXTEN**: HXT oscillator enable control
0: Disable
1: Enable

Bit 1~0 **FODIV1~FODIV0**: CLKOUT pin prescaler control
00: $f_{CLKO}=f_{SYS}$
01: $f_{CLKO}=f_{SYS}/2$
10: $f_{CLKO}=f_{SYS}/4$
11: $f_{CLKO}=f_{SYS}/8$

- **SFIOSTC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	SOFT2	SOFT1	SOFT0	—	—	CLKHST	MISOHST
R/W	—	R/W	R/W	R/W	—	—	R/W	R/W
POR	—	0	0	0	—	—	0	0

Bit 7 Unimplemented, read as "0"

Bit 6~4 **SOFT2~SOFT0**: SOF signal width selection
SOF signal width=$2^{[(SOFT[2:0]+3)]}\times(1/f_{HXT})$
Here, SOFT[2:0]=000~111

Bit 3~2 Unimplemented, read as "0"

Bit 1 **CLKHST**: CLKOUT Pin output state for HALT mode
0: Output low
1: Output high

Bit 0 **MISOHST**: MISO Pin output state for HALT mode
0: Output low
1: Output high

Note: At HXT off state, CANCFG, FOCFG & SFIOSTC SFRs can be writed, but cannot be read.

HT45B3305H CAN Configuration Register

- **CANCFG Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	CANEN	—	RMFD4	RMFD3	RMFD2	RMFD1	RMFD0
R/W	R/W	R/W	—	R/W	R/W	R/W	R/W	R/W
POR	1	0	—	0	0	0	0	0

Bit 7 **D7**: Reserved, must be fixed at "1"

Bit 6 **CANEN**: CAN Core Enable Control
0: Disable
1: Enable
When this bit is cleared to zero, CAN core remains in reset state and cannot be written in.

Bit 5 Unimplemented, read as "0"

Bit 4~0 **RMFD4~RMFD0**: Set receive FIFO threshold
0x00 : Message Object Number 32.
0x01~0x1F: Message Object Number 1 ~ Message Object Number 31.
These bits are used to select one of the Message Object Number 1~ Message Object Number 32. When the selected Message Object received a Message successfully, a interrrupt signal RMXINT will output on the RMXINT pin.

CAN Protocol Related Registers

These registers are related to the CAN protocol controller in the CAN Core. They control the operating modes and the configuration of the CAN bit timing and provide status information.

HT45B3305H CAN Control Registers

The contents of the control register are used to change the behavior of the device CAN operation. Bits may be set or reset by the connected MCU via a SPI or I^2C interface.

- **CTRLRL Register (ADDRESS: 0x00H)**

Bit	7	6	5	4	3	2	1	0
Name	TEST	CCE	DAR	—	EIE	SIE	CANIE	INIT
R/W	R/W	R/W	R/W	—	R/W	R/W	R/W	R/W
POR	0	0	0	—	0	0	0	1

Bit 7 **TEST**: Test Mode Enable Control
0: Normal Operation
1: Test Mode

Bit 6 **CCE**: Configuration Change Enable
0: The CPU has no write access to protected register bits
1: While **INIT**='1', the CPU has write access to protected register bits

Bit 5 **DAR**: Disable Automatic Retransmission
0: Enable Automatic Retransmission of disturbed messages
1: Disable Automatic Retransmission of disturbed messages

Bit 4 Unimplemented, read as "0"

Bit 3 **EIE**: Error Interrupt Enable
0: Disabled - No Error Status Interrupt will be generated
1: Enabled - A change of bits **BOFF** or **EWARN** in the Status Register will cause the Interrupt Register to be set to Status Interrupt (INTID15~INTID0=0x8000)

Bit 2 **SIE**: Status Change Interrupt Enable
0: Disabled - No Status Change Interrupt will be generated
1: Enabled - The Interrupt Register will be set to Status Interrupt (INTID15~INTID0=0x8000) when the CAN sets **LEC[2:0]** to a value ≠ 7

Bit 1 **CANIE**: Module Interrupt Enable
0: Disabled - Module Interrupt **can_int** is always inactive
1: Enabled - When the Interrupt Register is ≠ zero, the interrupt line **can_int** is set to active. **can_int** remains active until all interrupts are processed (Interrupt Register returns to zero)

Bit 0 **INIT**: Initialization
0: Normal Operation
1: Initialization is started

The busoff recovery sequence (see CAN Specification Rev. 2.0) cannot be shortened by setting or resetting **INIT**. If the device goes busoff, it will set **INIT** of its own accord, stopping all bus activities.

Once **INIT** has been cleared by the CPU, the device will then wait for 129 occurrences of Bus Idle (129×11 consecutive recessive bits) before resuming normal operations. At the end of the busoff recovery sequence, the Error Management Counters will be reset.

Test Register

Write access to the Test Register TESTRL is enabled by setting bit TEST in the HT45B3305H CAN Control Register. The different test functions may be combined, but TX[1:0] ≠ "00" disturbs message transfer. The TESTRL register should be cleared to zero before exiting the Test Mode.

• TESTRL Register (ADDRESS: 0x0AH)

Bit	7	6	5	4	3	2	1	0
Name	RX	TX1	TX0	LBACK	SILENT	BASIC	—	—
R/W	R	R/W	R/W	R/W	R/W	R/W	—	—
POR	x	0	0	0	0	0	—	—

Bit 7 **RX**: Monitors the actual value of the **CANRX** Pin
0: The CAN bus is dominant (**CANRX**='0').
1: The CAN bus is recessive (**CANRX**='1').

Note: The POR value of 'x' signifies the actual POR value of the CANRX pin.

Bit 6~5 **TX1~TX0**: Control of **CANTX** pin
00: Reset value, **CANTX** is controlled by the CAN Core.
01: Sample Point can be monitored at **CANTX** pin
10: **CANTX** pin drives a dominant ('0') value.
11: **CANTX** pin drives a recessive ('1') value.

Bit 4 **LBACK**: Loop Back Mode Control
0: Loop Back Mode is disabled.
1: Loop Back Mode is enabled.

Bit 3 **SILENT**: Silent Mode Control
0: Normal operation.
1: The module is in Silent Mode.

Bit 2 **BASIC**: Basic Mode Control
0: Basic Mode disabled.
1: Basic Mode, IF1 Registers used as TX Buffer, IF2 Registers used as RX Buffer.

Bit 1~0 Unimplemented, read as "0"

Status Register

The contents of the status register reflect the status of the HT45B3305H CAN. Some bits can only be read while some are read/write bits.

• **STATRL Register (ADDRESS: 0x02H)**

Bit	7	6	5	4	3	2	1	0
Name	BOFF	EWARN	EPASS	RXOK	TXOK	LEC2	LEC1	LEC0
R/W	R	R	R	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **BOFF**: Busoff Status
0: The CAN module is not busoff
1: The CAN module is in busoff state

Bit 6 **EWARN**: Error Warning Status
0: Both error counters are below the error warning limit of 96
1: At least one of the error counters in the EML has reached the error warning limit of 96

Bit 5 **EPASS**: Error Passive
0: The CAN Core is error active
1: The CAN Core is error passive as defined in the CAN Specification

Bit 4 **RXOK**: Received a Message Successfully
0: Since this bit was last reset by the CPU, no message has been successfully received.
1: Since this bit was last reset by the CPU, a message has been successfully received (independent of the result of acceptance filtering).
This RXOK bit is never reset by the CAN Core.

Bit 3 **TXOK**: Transmitted a Message Successfully
0: Since this bit was reset by the CPU, no message has been successfully transmitted.
1: Since this bit was last reset by the CPU, a message has been successfully (error free and acknowledged by at least one other node) transmitted
This TXOK bit is never reset by the CAN Core

Bit 2~0 **LEC2~LEC0**: Last Error Code (Type of the last protocol event to occur on the CAN bus)
000: **No Error**
Message successfully transmitted or received.
001: **Stuff Error**
More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed.
010: **Form Error**
Fixed format part of a received frame has the wrong format.
011: **AckError**
The message this CAN Core transmitted was not acknowedged by another node.
100: **Bit1Error**
During the transmission of a message (with the exception of the arbitration field), the device wanted to send a recessive level (bit of logical value '1'), but the monitored bus value was dominant.
101: **Bit0 Error**
During the transmission of a message (or acknowledge bit, or active error flag, or overload flag), the device wanted to send a dominant level (data or identifier bit logical value '0', but the monitored Bus value was recessive. During busoff recovery this status is set each time a sequence of 11 recessive bits has been monitored. This enables the CPU to monitor the proceeding of the busoff recovery sequence (indicating the bus is not stuck at dominant or continuously disturbed). During the waiting time after the resetting of **INIT**, each time a sequence of 11 recessive bits has been monitored, a **Bit0 Error** code is written to the Status Register, enabling the CPU to readily check up whether the CAN bus is stuck at dominant or continuously disturbed and to monitor the proceeding of the busoff recovery sequence.

110: **CRC Error**
The CRC check sum was incorrect in the message received, the CRC received for an incoming wanted to send a recessive level (bit of logical value '1'), but the monitored bus value was dominant.

111: **No Change**
When the LEC bit field shows the value '7', no CAN bus event was detected since the CPU wrote this value to the LEC bit field.

The LEC field holds a code which indicates the type of the last error to occur on the CAN bus. This field will be cleared to '0' when a message has been transferred (reception or transmission) without error. The code '7' may be written by the CPU to check for updates.

Note: A Status Interrupt is generated by bits BOFF and EWARN (Error Interrupt) or by RXOK, TXOK and LEC (Status Change Interrupt) assumed that the corresponding enable bits in the CAN Control Register are set. A change of bit EPASS or a write to RXOK, TXOK or LEC will never generate a Status Interrupt.

Reading the Status Register will clear the Status Interrupt value (INTID15~INTID0=0x8000) in the Interrupt Register, if it is pending.

CAN Error Counter Registers

• ERRCNTL Register (ADDRESS: 0x04H)

Bit	7	6	5	4	3	2	1	0
Name	TEC7	TEC6	TEC5	TEC4	TEC3	TEC2	TEC1	TEC0
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **TEC7~TEC0**: Transmit Error Counter
Actual state of the Transmit Error Counter.

• ERRCNTH Register (ADDRESS: 0x05H)

Bit	7	6	5	4	3	2	1	0
Name	RP	REC6	REC5	REC4	REC3	REC2	REC1	REC0
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7 **RP**: Receive Error Passive
0: The Receive Error Counter is below the error passive level
1: The Receive Error Counter has reached the error passive level as defined in the CAN Specification

Bit 6~0 **REC6~REC0**: Receive Error Counter
Actual state of the Receive Error Counter.

Bit Timing Registers

The bit time is divided into four segments which are the Synchronisation Segment, the Propagation Time Segment, the Phase Buffer Segment 1 and the Phase Buffer Segment 2. Each segment consists of a specification, programmable number of time quanta. The length of the time quantum (tq), which is the basic time unit of the bit time, is defined by the device HT45B3305H CAN controller's system clock f_{CAN} and the Baud Rate Prescaler (BRP) and the BRP Extension Register.

The time quantum is defined as:

$$t_q = (\mathbf{BRPE[3:0]} \times 0x40 + \mathbf{BRP[5:0]} + 1) / f_{CAN}$$

Where f_{CAN} = HT45B3305H CAN module clock frequency

The contents of the **BTRL** register define the values of the Baud Rate Prescaler and the (Re) Synchronisation Jump Width(SJW). The **BTRH** register bits define the length of the time segment before and after the sample point. The registers are writable only when bits **CCE** and **INIT** in the CAN Control Register are set.

• BTRL Register (ADDRESS: 0x06H)

Bit	7	6	5	4	3	2	1	0
Name	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	1

Bit 7~6 **SJW1~SJW0**: (Re)Synchronisation Jump Width
0x00~0x03: Valid programmed values are 0~3.
The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.

Bit 5~0 **BRP5~BRP0**: Baud Rate Prescaler (CAN module value)
0x01~0x3F: The value by which the system clock frequency is divided for generating the bit time quanta. The bit time is built up from a multiple of this time quanta. Valid values for **BRP[5:0]** are 0~63.
The actual interpretation by the hardware of this value is such that one more than the programmed value is used.

• BTRH Register (ADDRESS: 0x07H)

Bit	7	6	5	4	3	2	1	0
Name	—	TSG2D2	TSG2D1	TSG2D0	TSG1D3	TSG1D2	TSG1D1	TSG1D0
R/W	—	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	0	1	0	0	0	1	1

Bit 7 Unimplemented, read as "0"

Bit 6~4 **TSG2D2~TSG2D0**: The time segment after the sample point
0x00~0x07: Valid values for **TSG2D[2:0]** are 0~7. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.

Bit 3~0 **TSG1D3~TSG1D0**: The time segment before the sample point
0x01~0x0F: Valid values for TSG1D[3:0] are 1~15. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.

Note: If f_{CAN}=8MHz, the reset value of BTRL=0x01 and BTRH=0x23 configures the CAN for a bit rate of 500 kBit/s.

• BRPERL Register (ADDRESS: 0x0CH)

This BRPERL register configures the BRP extension for Classic CAN operation. The register is writable by setting CCE bit.

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	BRPE3	BRPE2	BRPE1	BRPE0
R/W	—	—	—	—	R/W	R/W	R/W	R/W
POR	—	—	—	—	0	0	0	0

Bit 7~4 Unimplemented, read as "0"

Bit 3~0 **BRPE3~BRPE0**: Baud Rate Prescaler Extension
0x00~0x0F: By programming **BRPE[3:0]** the Baud Rate Prescaler can be extended to values up to 1023.
The actual interpretation by the hardware is that one more than the value programmed by **BRPE[3:0]** (MSBs) and **BRP[5:0]** (LSBs) is used.

Message Interface Registers

There are two sets of Interface Registers which are used to control the CPU access to the Message RAM. The Interface Registers avoid conflicts between CPU access to the Message RAM and CAN message reception and transmission by buffering the data to be transferred. A complete Message Object or parts of the Message Object may be transferred between the Message RAM and the IFn Message Buffer registers in one single transfer.

The function of the two interface register sets is identical (except for test mode **Basic**). They can be used the way that one set of registers is used for data transfer to the Message RAM while the other set of registers is used for the data transfer from the Message RAM, allowing both processes to be interrupted by each other.

Each set of Interface Registers consists of Message Buffer Registers controlled by their own Command Registers. The Command Mask Register specifies the direction of the data transfer and which parts of a Message Object will be transferred. The Command Request Register is used to select a Message Object in the Message RAM as target or source for the transfer and to start the action specified in the Command Mask Register.

IF1 Register Set	Description	IF2 Register Set	Description
IF1CREQH/ IF1CREQL	IF1 Command Request	IF2CREQH/IF2CREQL	IF2 Command Request
IF1CMSKL	IF1 Command Mask	IF2CMSKL	IF2 Command Mask
IF1MSK1H/ IF1MSK1L	IF1 Mask 1	IF2MSK1H/IF2MSK1L	IF2 Mask 1
IF1MSK2H/ IF1MSK2L	IF1 Mask 2	IF2MSK2H/IF2MSK2L	IF2 Mask 2
IF1ARB1H/ IF1ARB1L	IF1 Arbitration 1	IF2ARB1H/IF2ARB1L	IF2 Arbitration 1
IF1ARB2H/ IF1ARB2L	IF1 Arbitration 2	IF2ARB2H/IF2ARB2L	IF2 Arbitration 2
IF1MCTRH/ IF1MCTRL	IF1 Message Control	IF2MCTRH/IF2MCTRL	IF2 Message Control
IF1DTA1H/ IF1DTA1L	IF1 Data A1	IF2DTA1H/IF2DTA1L	IF2 Data A1
IF1DTA2H/ IF1DTA2L	IF1 Data A2	IF2DTA2H/IF2DTA2L	IF2 Data A2
IF1DTB1H/ IF1DTB1L	IF1 Data B1	IF2DTB1H/IF2DTB1L	IF2 Data B1
IF1DTB2H/ IF1DTB2L	IF1 Data B2	IF2DTB2H/IF2DTB2L	IF2 Data B2

IF1 and IF2 Message Interface Register Sets

IFn Command Request Registers

A message transfer is started as soon as the CPU has written the message number to the Command Request Register. With this write operation the **BUSYn** bit is automatically set to '1' and output **can_wait_b** is activated to notify the CPU that a transfer is in progress. After a wait time of 3 to 6 **can_clk** periods, the transfer between the Interface Register and the Message RAM has completed. The **BUSYn** bit is set back to zero and **can_wait_b** is deactivated (see Module Integration Guide).

- **IFnCREQL Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	MSGnN5	MSGnN4	MSGnN3	MSGnN2	MSGnN1	MSGnN0
R/W	—	—	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	—	0	0	0	0	0	1

Bit 7~6 Unimplemented, read as "0"

Bit 5~0 **MSGnN5~MSGnN0**: Message Number
0x00: Not a valid Message Number, interpreted as 0x20
0x01~0x20: Valid Message Number, the Message Object in the Message RAM is selected for data transfer.
0x21~0x3F: Not a valid Message Number, interpreted as 0x01-0x1F

Note: When a Message Number that is not valid is written into the Command Request Register, the Message Number will be transformed into a valid value and that Message Object will be transferred.

• **IFnCREQH Register**

Bit	7	6	5	4	3	2	1	0
Name	BUSYn	—	—	—	—	—	—	—
R/W	R/W	—	—	—	—	—	—	—
POR	0	—	—	—	—	—	—	—

Bit 7 **BUSYn**: Busy Flag
0: Reset to zero when read/write action has finished.
1: Set to one when writing to the IFn Command Request Register.

Bit 6~0 Unimplemented, read as "0"

IFn Command Mask Registers

The control bits of the IFn Command Mask Register specify the transfer direction and select which of the IFn Message Buffer Registers as source or target of the data transfer.

• **IFnCMSKL Register**

Bit	7	6	5	4	3	2	1	0
Name	TDnDIR	MASKn	ARBn	CTRLn	CINTPNDn	TQnDTA	DATAnA	DATAnB
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **TDnDIR**: Write / Read Selection
0: Read - Transfer data from the Message Object addressed by the Command Request Register into the selected Message Buffer Registers.
1: Write - Transfer data from the selected Message Buffer Registers to the Message Object addressed by the Command Request Register.
The other bits of IFn Command MASKn Register have different functions depending on the transfer direction:

Direction=Write (TDnDIR=1)

Bit 6 **MASKn**: Access MASK Bits
0: MASK bits unchanged.
1: Transfer Identifier MASKn + MDIRn + MXTDn to Message Object.

Bit 5 **ARBn**: Access Arbitration Bits
0: Arbitration bits unchanged.
1: Transfer Identifier + DIRn + XTDn + MSGnVA to Message Object.

Bit 4 **CTRLn**: Access Control Bits
0: Control Bits unchanged.
1: Transfer Control Bits to Message Object.

Bit 3 **CINTPNDn**: Clear Interrupt Pending Bit
0: INTPND bit remains unchanged.
1: Clear INTPND bit in the Message Object.
Note: When writing to a Message Object, this bit is ignored.

Bit 2 **TQnDTA**: Access Transmission Request Bit
0: TREQ bit unchanged
1: Set TREQ bit
If a transmission is requested by programming bit TQnDTA in the IFn Command Mask Register, bit TnREQ in the IFn Message Control Register will be ignored.

Bit 1 **DATAnA**: Access Data Bytes 0-3
0: Data Bytes 0-3 unchanged.
1: Transfer Data Bytes 0-3 to Message Object.

Bit 0 **DATAnB**: Access Data Bytes 4-7
0: Data Bytes 4-7 unchanged
1: Transfer Data Bytes 4-7 to Message Object.

Direction=Read (TDnDIR=0)

Bit 6 **MASKn**: Access MASK Bits
0: MASK bits unchanged.
1: Transfer Identifier MASKn + MDIRn + MXTDn to IFn Message Buffer Register.

Bit 5 **ARBn**: Access Arbitration Bits
0: Arbitration bits unchanged.
1: transfer Identifier + DIRn + XTDn + MSGnVA to IFn Message Buffer Register.

Bit 4 **CTRLn**: Access Control Bits
0: Control Bits unchanged.
1: Transfer Control Bits to IFn Message Buffer Register.

Bit 3 **CINTPNDn**: Clear Interrupt Pending Bit
0: INTPND bit remains unchanged.
1: Clear INTPND bit in the Message Object.

Bit 2 **TQnDTA**: Access New Data Bit
0: NDTA bit remains unchanged.
1: Clear NDTA bit in the Message Object.

Note: A read access to a Message Object can be combined with the reset of the control bits **INTnPND** and **NnDTA**.The values of these bits transferred to the IFn Message Control Register always reflect the status before resetting these bits.

Bit 1 **DATAnA**: Access Data Bytes 0-3
0: Data Bytes 0-3 unchanged.
1: Transfer Data Bytes 0-3 to IFn Message Buffer Register.

Bit 0 **DATAnB**: Access Data Bytes 4-7
0: Data Bytes 4-7 unchanged.
1: Transfer Data Bytes 4-7 to IFn Message Buffer Register.

IFn Message Buffer Registers

The bits of the Message Buffer registers mirror the Message Objects in the Message RAM.

• IFnMSK1L Register

Bit	7	6	5	4	3	2	1	0
Name	MSKn07	MSKn06	MSKn05	MSKn04	MSKn03	MSKn02	MSKn01	MSKn00
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	1	1	1	1	1	1	1	1

Bit 7~0 **MSKn07~MSKn00**: Identifier MASK
0: The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering.
1: The corresponding identifier bit is used for acceptance filtering.

• IFnMSK1H Register

Bit	7	6	5	4	3	2	1	0
Name	MSKn15	MSKn14	MSKn13	MSKn12	MSKn11	MSKn10	MSKn09	MSKn08
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	1	1	1	1	1	1	1	1

Bit 7~0 **MSKn15~MSKn08**: Identifier MASK
0: The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering.
1: The corresponding identifier bit is used for acceptance filtering.

• **IFnMSK2L Register**

Bit	7	6	5	4	3	2	1	0
Name	MSKn23	MSKn22	MSKn21	MSKn20	MSKn19	MSKn18	MSKn17	MSKn16
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	1	1	1	1	1	1	1	1

Bit 7~0 **MSKn23~MSKn16**: Identifier MASK
0: The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering.
1: The corresponding identifier bit is used for acceptance filtering.

• **IFnMSK2H Register**

Bit	7	6	5	4	3	2	1	0
Name	MXTDn	MDIRn	—	MSKn28	MSKn27	MSKn26	MSKn25	MSKn24
R/W	R/W	R/W	—	R/W	R/W	R/W	R/W	R/W
POR	1	1	—	1	1	1	1	1

Bit 7 **MXTDn**: MASK Extended Identifier
0: The extended identifier bit (IDE) has no effect on the acceptance filtering.
1: The extended identifier bit (IDE) is used for acceptance filtering.

Bit 6 **MDIRn**: MASK Message Direction
0: The message direction bit (DIR) has no effect on the acceptance filtering.
1: The message direction bit (DIR) is used for acceptance filtering.

Bit 5 Unimplemented, read as "1"

Bit 4~0 **MSKn28~MSKn24**: Identifier MASK
0: The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering.
1: The corresponding identifier bit is used for acceptance filtering.

• **IFnARB1L Register**

Bit	7	6	5	4	3	2	1	0
Name	IDn07	IDn06	IDn05	IDn04	IDn03	IDn02	IDn01	IDn00
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **IDn07~IDn00**: Message Identifier 7~0

• **IFnARB1H Register**

Bit	7	6	5	4	3	2	1	0
Name	IDn15	IDn14	IDn13	IDn12	IDn11	IDn10	IDn09	IDn08
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **IDn15~IDn8**: Message Identifier 15~8

• **IFnARB2L Register**

Bit	7	6	5	4	3	2	1	0
Name	IDn23	IDn22	IDn21	IDn20	IDn19	IDn18	IDn17	IDn16
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **IDn23~IDn16**: Message Identifier 23~16

• **IFnARB2H Register**

Bit	7	6	5	4	3	2	1	0
Name	MSGnVA	XTDn	DIRn	IDn28	IDn27	IDn26	IDn25	IDn24
R/W	RW	RW	RW	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **MSGnVA**: Message Valid Bits (of all Message Objects)
0: This Message Object is ignored by the Message Handler
1: This Message Object is configured and should be considered by the Message Handler

Bit 6 **XTDn**: Extended Identifier
0: The 11-bit ("standard") Identifier will be used for this Message Object.
1: The 29-bit ("extended") Identifier will be used for this Message Object.

Bit 5 **DIRn**: Message Direction
0: Direction=receive. On **TREQ**, a Remote Frame with the identifier of this Message Object is transmitted. On reception of a Data Frame with matching identifier, that message is stored in this Message Object.
1: Direction=transmit. On **TREQ**, the respective Message Object is transmitted as a Data Frame. On reception of a Remote Frame with matching identifier, the **TREQ** bit of this Message Object is set (if **RMTnEN**=1).

Bit 4~0 **IDn28~IDn24**: Message Identifier 28~24

IFn Message Control Registers

• **IFnMCTRL Register**

Bit	7	6	5	4	3	2	1	0
Name	EOBn	—	—	—	DLCn3	DLCn2	DLCn1	DLCn0
R/W	R/W	—	—	—	R/W	R/W	R/W	R/W
POR	0	—	—	—	0	0	0	0

Bit 7 **EOBn**: End of Buffer
0: Message Object belongs to a FIFO Buffer and is not the last Message Object of that FIFO Buffer.
1: Single Message Object or last Message Object of a FIFO Buffer.
This bit is used to concatenate two or more Message Objects (up to 32) to build a FIFO Buffer. For single Message Objects (not belonging to a FIFO Buffer), this bit must always be set to one.

Bit 6~4 Unimplemented, read as "0"

Bit 3~0 **DLCn3~DLCn0**: Data Length Code
0~8: CAN: Frame has 0 ~ 8 data bytes
9~15: CAN: Frame has 8 data bytes
The Data Length Code of a Message Object must be defined the same as in all the corresponding objects with the same identifier at other nodes.
When the Message Handler stores a data frame, it will write the DLCn to the value given by the received message.

• **IFnMCTRH Register**

Bit	7	6	5	4	3	2	1	0
Name	NnDTA	MSGnLST	INTnPND	UMASKn	TXnIEN	RXnIEN	RMTnEN	TnREQ
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **NnDTA**: New Data Bits
0: No new data has been written into the data portion of this Message Object by the Message Handler since last time this flag was cleared by the CPU.
1: The Message Handler or the CPU has written new data into the data portion of this Message Object

Bit 6 **MSGnLST**: Message Lost (only valid for Message Objects with direction=receive)
0: No message lost since last time this bit was reset by the CPU.
1: The Message Handler stored a new message into this object when NDTA was still set, the CPU has lost a message.

Bit 5 **INTnPND**: Interrupt Pending Bits
0: This message object is not the source of an interrupt.
1: This message object is the source of an interrupt. The Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with higher priority.

Bit 4 **UMASKn**: Use Acceptance Mask
0: MASK ignored.
1: Use MASK (MSKn[28:00], MXTDn, and MDIRn) for acceptance filtering.

Bit 3 **TXnIEN**: Transmit Interrupt Enable
0: INTPND will be left unchanged after the successful transmission of a frame.
1: INTPND will be set after a successful transmission of a frame.

Bit 2 **RXnIEN**: Receive Interrupt Enable
0: INTPND will be left unchanged after a successful reception of a frame.
1: INTPND will be set after a successful reception of a frame.

Bit 1 **RMTnEN**: Remote Enable
0: At the reception of a Remote Frame, TREQ is left unchanged.
1: At the reception of a Remote Frame, TREQ is set.

Bit 0 **TnREQ**: Transmission Request Bits
0: This Message Object is not waiting for transmission.
1: The transmission of this Message Object is requested and is not yet done.

IFn Data A and Data B Registers

In a CAN Data Frame, Data0 is the first, Data7 is the last byte to be transmitted or received. In CAN's serial bit stream, the MSB of each byte will be transmitted first.

DATA0: 1st data byte of a CAN Data Frame

DATA1: 2nd data byte of a CAN Data Frame

DATA2: 3rd data byte of a CAN Data Frame

DATA3: 4th data byte of a CAN Data Frame

DATA4: 5th data byte of a CAN Data Frame

DATA5: 6th data byte of a CAN Data Frame

DATA6: 7th data byte of a CAN Data Frame

DATA7: 8th data byte of a CAN Data Frame

The data bytes of CAN messages are stored in the IFn Message Buffer Registers in the following order:

• IFnDTA1L Register

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA0, 1st data byte of a CAN Data Frame

• IFnDTA1H Register

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA1, 2nd data byte of a CAN Data Frame

• IFnDTA2L Register

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA2, 3rd data byte of a CAN Data Frame

• IFnDTA2H Register

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA3, 4th data byte of a CAN Data Frame

• **IFnDTB1L Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA4, 5th data byte of a CAN Data Frame

• **IFnDTB1H Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA5, 6th data byte of a CAN Data Frame

• **IFnDTB2L Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA6, 7th data byte of a CAN Data Frame

• **IFnDTB2H Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA7, 8th data byte of a CAN Data Frame

Note: Byte **DATA0** is the first data byte shifted into the shift register of the CAN Core during a reception, byte **DATA7** is the last. When the Message Handler stores a Data Frame, it will write all the eight data bytes into a Message Object. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by **non specified values**.

Message Handler Registers

All Message Handler registers are read-only. Their contents, which include the **TREQ**, **NDTA**, **INTPND**, and **MSGVA** bits of each Message Object and the Interrupt Identifier) is status information provided by the Message Handler FSM(Finite State Machine).

Interrupt Registers

The interrupt registers allow the identification of an interrupt source. When an interrupt occurs, a CAN interrupt will be indicated to the CPU and an active interrupt level will output on the CANMINT pin to inform users the HT45B3305H CAN interrupt.

The interrupt register appears to the CPU as a read only memory.

• **INTRL Register (ADDRESS: 0x08H)**

Bit	7	6	5	4	3	2	1	0
Name	INTID7	INTID6	INTID5	INTID4	INTID3	INTID2	INTID1	INTID0
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **INTID7~INTID0**: Interrupt Identifier

• **INTRH Register (ADDRESS: 0x09H)**

Bit	7	6	5	4	3	2	1	0
Name	INTID15	INTID14	INTID13	INTID12	INTID11	INTID10	INTID9	INTID8
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **INTID15~INTID8**: Interrupt Identifier
The interrupt identifier INTID[15:0] indicates the source of the interrupt.

INTID[15:0] Value	Indicated Interrupt
0000H	No interrupt is pending
0001H~0020H	Number of Message Object which caused the interrupt
0021H~7FFFH	Unused
8000H	Status Interrupt
8001H~FFFFH	Unused

If several interrupts are pending, the CAN Interrupt Register will point to the pending interrupt with the highest priority disregarding their chronological order. An interrupt remains pending until the CPU has cleared it. If **INTID15~INTID0** is different from 0x0000 and **CANIE** is set, the **CANMINT** pin is driven an active level by the device and will remain the active level until **INTID15~INTID0** is back to value 0x0000 (the cause of the interrupt is reset) or until **CANIE** is reset. The active HT45B3305H CAN interrupt level is determined by the CANIV bit in the FOCFG register.

The Status Interrupt has the highest priority. Among the message interrupts, the Message Object's interrupt priority decreases with increasing message number.

A message interrupt is cleared by clearing the Message Object's **INTPND** bit. The Status Interrupt is cleared by reading the Status Register resp.

The Status Interrupt value 0x8000(INTID15~INTID0) indicates that an interrupt is pending because the CAN Core has updated (not necessarily changed) the Status Register (Error Interrupt or Status Interrupt). This interrupt has the highest priority. The CPU can update (reset) the Status Register bits **RXOK, TXOK** and **LEC** by writing to the Status Register. A write access by the CPU to the Status Registers can never generate or reset an interrupt.

All other values indicate that the source of the interrupt is one of the Message Objects, **INTID** points to the pending message interrupt with the highest interrupt priority.

The CPU controls whether a change of the Status Registers may cause the Interrupt Register to be set to **INTID** Status interrupt and whether the interrupt line becomes active when the Interrupt Register is different from zero (bit **CANIE** in the CAN Control Register). The Interrupt Register will be updated even when **CANIE** is not set.

The CPU has two possibilities to follow the source of a message interrupt. First it can follow the **INTID** in the Interrupt Register and second it can poll the Interrupt Pending Register.

An interrupt service routine reading the message that is the source of the interrupt may read the message and reset the Message Object's **INTPND** at the same time (bit **CINTPNDn** in the IFn Command Mask Register). When **INTPND** is cleared, the Interrupt Register will point to the next Message Object with a pending interrupt.

Transmission Request Registers

These registers hold the **TREQ[32:0]** bits of the 32 Message Objects. By reading out the **TREQ[32:0]** bits, the CPU can check for which Message Object a Transmission Request is pending. The **TREQ** bit of a specific Message Object can be set or reset by the CPU via the IFn Message Interface Registers or by the Message Handler after reception of a Remote Frame or after a successful transmission.

• TREQR1L Register (ADDRESS: 0x80H)

Bit	7	6	5	4	3	2	1	0
Name	TREQ8	TREQ7	TREQ6	TREQ5	TREQ4	TREQ3	TREQ2	TREQ1
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **TREQ8~TREQ1**: Transmission Request Bits of Message Object 8 ~ Message Object 1
0: This Message Object is not waiting for transmission.
1: The transmission of this Message Object is requested and is not yet done.

• TREQR1H Register (ADDRESS: 0x81H)

Bit	7	6	5	4	3	2	1	0
Name	TREQ16	TREQ15	TREQ14	TREQ13	TREQ12	TREQ11	TREQ10	TREQ9
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **TREQ16~TREQ9**: Transmission Request Bits of Message Object 16 ~ Message Object 9
0: This Message Object is not waiting for transmission.
1: The transmission of this Message Object is requested and is not yet done.

• TREQR2L Register (ADDRESS: 0x82H)

Bit	7	6	5	4	3	2	1	0
Name	TREQ24	TREQ23	TREQ22	TREQ21	TREQ20	TREQ19	TREQ18	TREQ17
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **TREQ24~TREQ17**: Transmission Request Bits of Message Object 24 ~ Message Object 17
0: This Message Object is not waiting for transmission.
1: The transmission of this Message Object is requested and is not yet done.

• TREQR2H Register (ADDRESS: 0x83H)

Bit	7	6	5	4	3	2	1	0
Name	TREQ32	TREQ31	TREQ30	TREQ29	TREQ28	TREQ27	TREQ26	TREQ25
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **TREQ32~TREQ25**: Transmission Request Bits of Message Object 32 ~ Message Object 25
0: This Message Object is not waiting for transmission.
1: The transmission of this Message Object is requested and is not yet done.

New Data Registers

These registers hold the NDTA bits of the 32 Message Objects. By reading out the NDTA bits, the CPU can check for which Message Object the data portion was updated. The NDTA bit of a specific Message Object can be set/reset by the CPU via the IFn Message Interface Registers or by the Message Handler after reception of a Data Frame or after a successful transmission.

• **NEWDT1L Register**

Bit	7	6	5	4	3	2	1	0
Name	NDTA8	NDTA7	NDTA6	NDTA5	NDTA4	NDTA3	NDTA2	NDTA1
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **NDTA8~NDTA1**: New Data Bits of Message Object 8 ~ Message Object 1.
0: No new data has been written into the data portion of this Message Object by the Message Handler since last time this flag was cleared by the CPU.
1: The Message Handler or the CPU has written new data into the data portion of this Message Object.

• **NEWDT1H Register**

Bit	7	6	5	4	3	2	1	0
Name	NDTA16	NDTA15	NDTA14	NDTA13	NDTA12	NDTA11	NDTA10	NDTA9
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **NDTA16~NDTA9**: New Data Bits of Message Object 16 ~ Message Object 9.
0: No new data has been written into the data portion of this Message Object by the Message Handler since last time this flag was cleared by the CPU.
1: The Message Handler or the CPU has written new data into the data portion of this Message Object.

• **NEWDT2L Register**

Bit	7	6	5	4	3	2	1	0
Name	NDTA24	NDTA23	NDTA22	NDTA21	NDTA20	NDTA19	NDTA18	NDTA17
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **NDTA24~NDTA17**: New Data Bits of Message Object 24 ~ Message Object 17
0: No new data has been written into the data portion of this Message Object by the Message Handler since last time this flag was cleared by the CPU.
1: The Message Handler or the CPU has written new data into the data portion of this Message Object.

• **NEWDT2H Register**

Bit	7	6	5	4	3	2	1	0
Name	NDTA32	NDTA31	NDTA30	NDTA29	NDTA28	NDTA27	NDTA26	NDTA25
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **NDTA32~NDTA25**: New Data Bits of Message Object 32 ~ Message Object 25.
0: No new data has been written into the data portion of this Message Object by the Message Handler since last time this flag was cleared by the CPU.
1: The Message Handler or the CPU has written new data into the data portion of this Message Object.

Interrupt Pending Registers

These registers hold the **INTPND** bits of the 32 Message Objects. By reading out the **INTPND** bits, the CPU can check for which Message Object an interrupt is pending. The **INTPND** bit of a specific Message Object can be set/reset by the CPU via the IFn Message Interface Registers or by the Message Handler after reception or after a successful transmission of a frame. This will also affect the value of **INTID** in the Interrupt Register.

• **INTPND1L Register (ADDRESS: 0xA0H)**

Bit	7	6	5	4	3	2	1	0
Name	INTPND8	INTPND7	INTPND6	INTPND5	INTPND4	INTPND3	INTPND2	INTPND1
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **INTPND8~INTPND1**: Interrupt Pending Bits of Message Object 8 ~ Message Object 1
0: This message object is not the source of an interrupt.
1: This message object is the source of an interrupt.
If the message object whose interrupt pending bit is set high is the source of an interrupt, the Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with higher priority.

• **INTPND1H Register (ADDRESS: 0xA1H)**

Bit	7	6	5	4	3	2	1	0
Name	INTPND16	INTPND15	INTPND14	INTPND13	INTPND12	INTPND11	INTPND10	INTPND9
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **INTPND16~INTPND9**: Interrupt Pending Bits of Message Object 16 ~ Message Object 9
0: This message object is not the source of an interrupt.
1: This message object is the source of an interrupt.
If the message object whose interrupt pending bit is set high is the source of an interrupt, the Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with higher priority.

• **INTPND2L Register (ADDRESS: 0xA2H)**

Bit	7	6	5	4	3	2	1	0
Name	INTPND24	INTPND23	INTPND22	INTPND21	INTPND20	INTPND19	INTPND18	INTPND17
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **INTPND24~INTPND17**: Interrupt Pending Bits of Message Object 24 ~ Message Object 17
0: This message object is not the source of an interrupt.
1: This message object is the source of an interrupt.
If the message object whose interrupt pending bit is set high is the source of an interrupt, the Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with higher priority.

• **INTPND2H Register (ADDRESS: 0xA3H)**

Bit	7	6	5	4	3	2	1	0
Name	INTPND32	INTPND31	INTPND30	INTPND29	INTPND28	INTPND27	INTPND26	INTPND25
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **INTPND32~INTPND25**: Interrupt Pending Bits of Message Object 32 ~ Message Object 25.
0: This message object is not the source of an interrupt.
1: This message object is the source of an interrupt.
If the message object whose interrupt pending bit is set high is the source of an interrupt, the Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with higher priority.

Message Valid Registers

These registers hold the MSGVA bits of the 32 Message Objects. By reading out the MSGVA bits, the CPU can check which Message Object is valid. The MSGVA bit of a specific Message Object can be set/reset by the CPU via the IFn Message Interface Registers.

• MSGVAL1L Register (ADDRESS: 0xB0H)

Bit	7	6	5	4	3	2	1	0
Name	MSGVA8	MSGVA7	MSGVA6	MSGVA5	MSGVA4	MSGVA3	MSGVA2	MSGVA1
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **MSGVA8~MSGVA1**: Message Valid Bits of Message Object 8 ~ Message Object 1
0: This Message Object is ignored by the Message Handler.
1: This Message Object is configured and should be considered by the Message Handler.

• MSGVAL1H Register(ADDRESS: 0xB1H)

Bit	7	6	5	4	3	2	1	0
Name	MSGVA16	MSGVA15	MSGVA14	MSGVA13	MSGVA12	MSGVA11	MSGVA10	MSGVA9
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **MSGVA16~MSGVA9**: Message Valid Bits of Message Object 16 ~ Message Object 9
0: This Message Object is ignored by the Message Handler.
1: This Message Object is configured and should be considered by the Message Handler.

• MSGVAL2L Register (ADDRESS: 0xB2H)

Bit	7	6	5	4	3	2	1	0
Name	MSGVA24	MSGVA23	MSGVA22	MSGVA21	MSGVA20	MSGVA19	MSGVA18	MSGVA17
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **MSGVA24~MSGVA17**: Message Valid Bits of Message Object 24 ~ Message Object 17.
0: This Message Object is ignored by the Message Handler.
1: This Message Object is configured and should be considered by the Message Handler.

• MSGVAL2H Register (ADDRESS: 0xB3H)

Bit	7	6	5	4	3	2	1	0
Name	MSGVA32	MSGVA31	MSGVA30	MSGVA29	MSGVA28	MSGVA27	MSGVA26	MSGVA25
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **MSGVA32~MSGVA25**: Message Valid Bits of Message Object 32 ~ Message Object 25
0: This Message Object is ignored by the Message Handler.
1: This Message Object is configured and should be considered by the Message Handler.

Note: The CPU must reset the **MSGVA** bit of all unused Messages Objects during the initialization before it resets bit INIT in the CAN Control Register. This bit must also be reset before the identifier **IDn[28:00]**, the control bits **XTDn**, **DIRn**, or the Data Length Code **DLCn[3:0]** are modified, or if the Messages Object is no longer required.

Core Release Registers

The design step of a HT45B3305H CAN implementation can be identified by reading the Core Release Registers Low/High.

Release	Step	SubStep	Year	Month	Day	Name
2	1	0	5	02	27	Revision 2.1.0, Date 2015/02/27

Example for Coding of Revisions

- **CRLL Register**

Bit	7	6	5	4	3	2	1	0
Name	DAY7	DAY6	DAY5	DAY4	DAY3	DAY2	DAY1	DAY0
R/W	R	R	R	R	R	R	R	R
POR	0	0	1	0	0	1	1	1

Bit 7~0 **DAY7~DAY0**: Time Stamp Day
Two digits, BCD-coded. Configured by constant on HT45B3305H CAN synthesis.

- **CRLH Register**

Bit	7	6	5	4	3	2	1	0
Name	MON7	MON6	MON5	MON4	MON3	MON2	MON1	MON0
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	1	0

Bit 7~0 **MON7~MON0**: Time Stamp Month
Two digits, BCD-coded. Configured by constant on HT45B3305H CAN synthesis.

- **CRHL Register**

Bit	7	6	5	4	3	2	1	0
Name	SUBSTEP3	SUBSTEP2	SUBSTEP1	SUBSTEP0	YEAR3	YEAR2	YEAR1	YEAR0
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	1	0	1

Bit 7~4 **SUBSTEP3~SUBSTEP0**: Sub-step of Core Release
One digit, BCD-coded.

Bit 3~0 **YEAR3~YEAR0**: Time Stamp Year (2010 + digit)
One digit, BCD-coded. Configured by constant on HT45B3305H CAN synthesis.

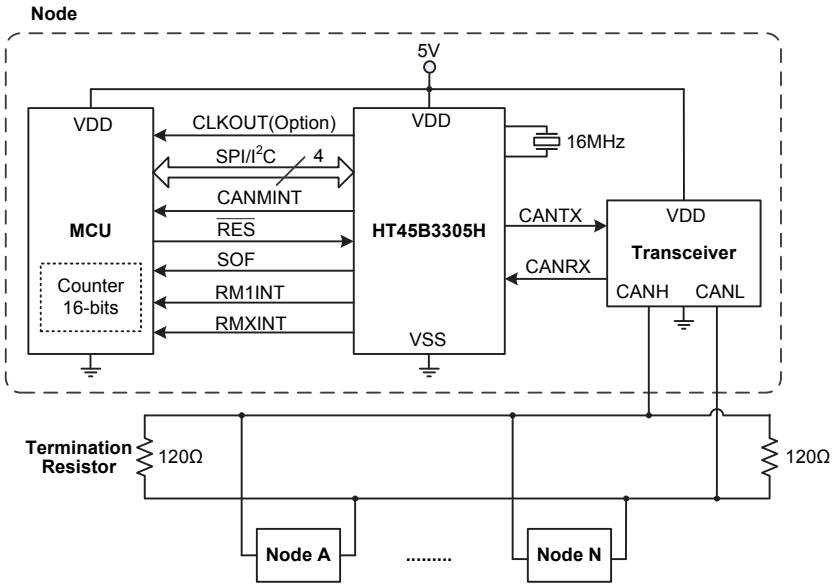
- **CRHH Register**

Bit	7	6	5	4	3	2	1	0
Name	REL3	REL2	REL1	REL0	STEP3	STEP2	STEP1	STEP0
R/W	R	R	R	R	R	R	R	R
POR	0	0	1	0	0	0	0	1

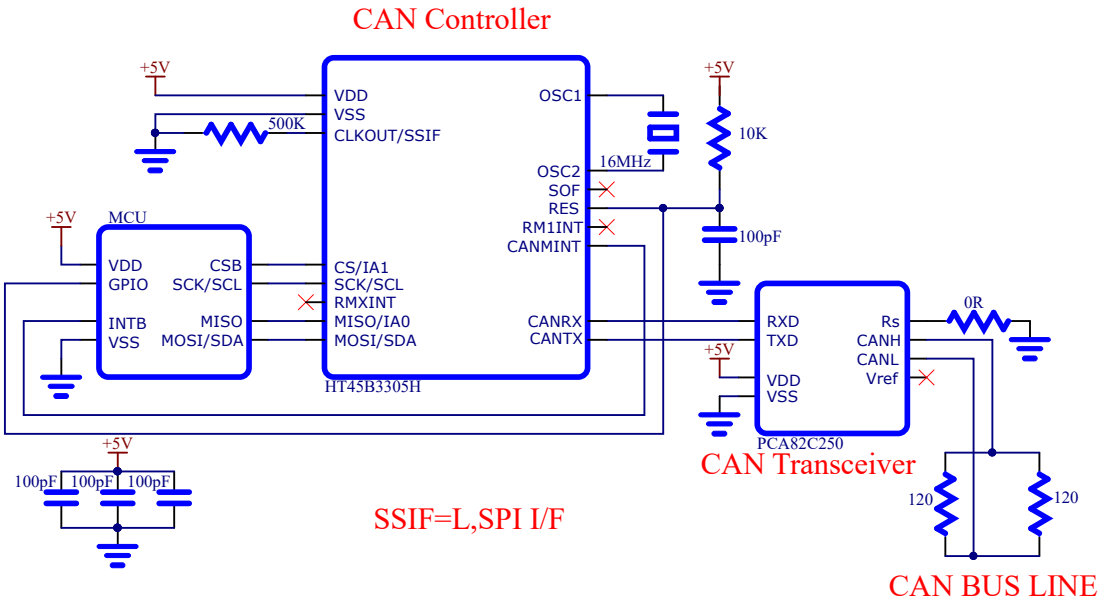
Bit7~4 **REL3~REL0**: Core Release
One digit, BCD-coded.

Bit 3~0 **STEP3~STEP0**: Step of Core Release
One digit, BCD-coded.

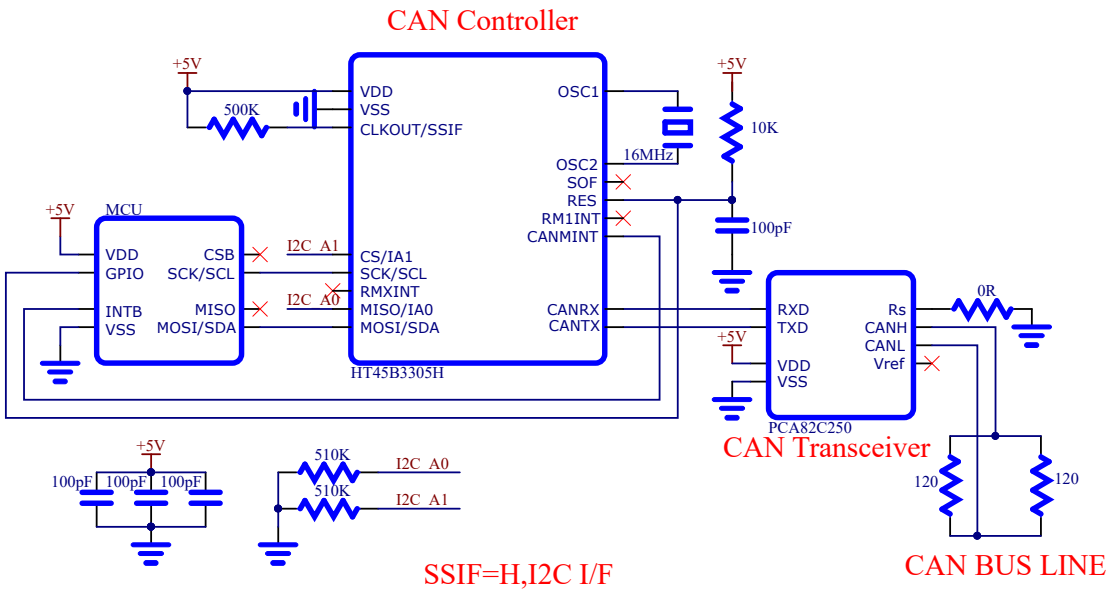
Application Circuits



SPI Serial Interface



I²C Serial Interface



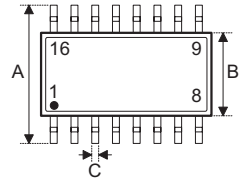
Package Information

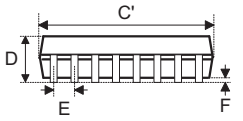
Note that the package information provided here is for consultation purposes only. As this information may be updated at regular intervals users are reminded to consult the Holtek website for the latest version of the Package Information.

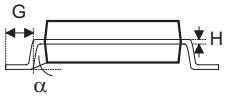
Additional supplementary information with regard to packaging is listed below. Click on the relevant section to be transferred to the relevant website page.

- Package Information (include Outline Dimensions, Product Tape and Reel Specifications)
- The Operation Instruction of Packing Materials
- Carton information

16-pin NSOP (150mil) Outline Dimensions



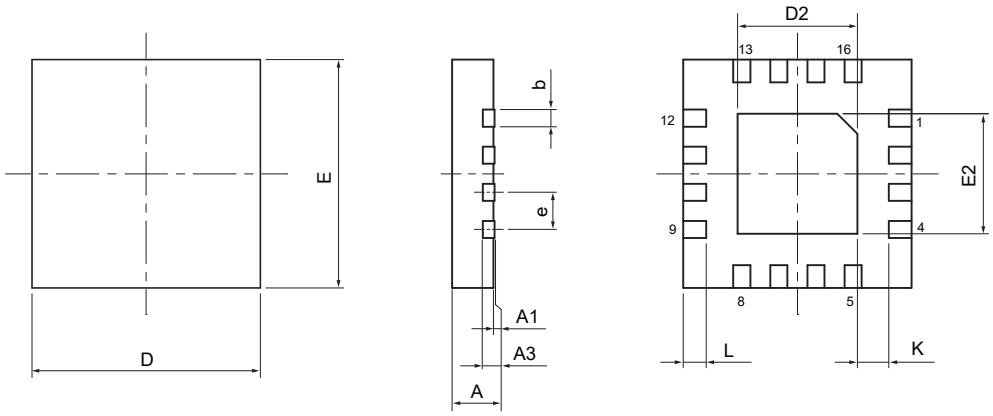




Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	—	0.236 BSC	—
B	—	0.154 BSC	—
C	0.012	—	0.020
C'	—	0.390 BSC	—
D	—	—	0.069
E	—	0.050 BSC	—
F	0.004	—	0.010
G	0.016	—	0.050
H	0.004	—	0.010
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	—	6.000 BSC	—
B	—	3.900 BSC	—
C	0.31	—	0.51
C'	—	9.900 BSC	—
D	—	—	1.75
E	—	1.270 BSC	—
F	0.10	—	0.25
G	0.40	—	1.27
H	0.10	—	0.25
α	0°	—	8°

SAW Type 16-pin QFN (3mm×3mm for FP0.25mm) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	0.028	0.030	0.031
A1	0.000	0.001	0.002
A3	—	0.008 REF	—
b	0.007	0.010	0.012
D	—	0.118 BSC	—
E	—	0.118 BSC	—
e	—	0.020 BSC	—
D2	0.063	—	0.069
E2	0.063	—	0.069
L	0.008	0.010	0.012
K	0.008	—	—

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	0.70	0.75	0.80
A1	0.00	0.02	0.05
A3	—	0.203 REF	—
b	0.18	0.25	0.30
D	—	3.00 BSC	—
E	—	3.00 BSC	—
e	—	0.50 BSC	—
D2	1.60	—	1.75
E2	1.60	—	1.75
L	0.20	0.25	0.30
K	0.20	—	—