M32C/83,85 Group
Using Simple I^2C Bus Mode on M32C/83,85

REJ05B0145-0100Z
REV.1.00
2003.08.22

1.0 Abstract

The M32C/83,85group of microcomputers contains an I^2C bus circuit in their serial I/O circuit (UART).
The I^2C bus circuit when used in combination with software makes it possible to control the I^2C bus interface.
This application note outlines I^2C bus specifications and describes the I^2C bus functions showing
the method and a sample program for using each function to materialize an I^2C bus interface.

2.0 Introduction

This document was prepared to provide reference information on how to control the I^2C bus
that is incorporated in the M32C/83,85 group of RENESAS CMOS microcomputers.
The information in this document only describes the communication operation of the I^2C bus,
and does not necessarily guarantee its performance in the user application.
Therefore, please make sure your application is thoroughly evaluated before putting it into use.
For details about the instruction architecture in the M32C/83,85 group of microcomputers,
please consult the M32C/80 Series Software Manual along with this document. For the hardware aspects of
the M32C/83,85 group of microcomputers, see the user's manual included with the microcomputer you use.
For the development support tools, see the user's manual included with each tool you use.
The example applications presented in this document assume use of the following type of microcomputer.

・Microcomputer :M32C/83,85group(M3083XXXXP/3085XXXXP)

The readers of this document are assumed to have the basic knowledge of electrical
and logic circuits and microcomputers.
This document consists of three chapters.
The following suggests the chapters or sections to be consulted when specific information is needed.

・To know the structure of the serial I/O in the M32C/83,85 group of microcomputers
See Section 1.1, "Functions Available with the Serial I/O," in Chapter 1, "Functions of the M32C/83,85 UART."
・To know the I^2C bus block diagram and the register structure of the serial I/O in the M32C/83,85 group of microcomputers
See Sections 1.2 to 1.4 in Chapter 1, "Functions of the M32C/83,85 UART."
・To understand how to use each function of the simple I^2C bus in the M32C/83,85 group of microcomputers
See Chapter 2, "Each Function of Simple I^2C Bus Mode."
・To know the precautions to be taken when using I^2C bus mode
See Chapter 3, "Precautions on Simple I^2C Bus Mode."
・To refer to a sample program for the I^2C bus interface unit using the M32C/83 group of microcomputers
See Appendix, "Sample Program."

3.0 Contents

Chapter 1 Functions of the M32C/83,85 UART

Chapter 2 Each Function of Simple I^2C Bus Mode

Chapter 3 Precautions on Simple I^2C Bus Mode

Appendix

Chapter 1

Functions of the M32C/83,85 UART

1.1 Functions Available with the Serial I/O
1.2 Simple I^{2}C Bus Mode Block Diagram
1.3 Changeable Pin Functions and Interrupt Sources in Simple I^{2}C Bus Mode
1.4 Register Settings during Simple I^{2}C Bus Mode

The serial I/O in the M32C/83,85 group of microcomputers consists of five UART channels,from 0 to 4.
Each UART channel has a dedicated transfer clock generating timer and can operate independently of each other.
This chapter describes in detail how to set simple I^{2}C bus mode which is one function of these UART channels.

1.1 Functions Available with the Serial I/O

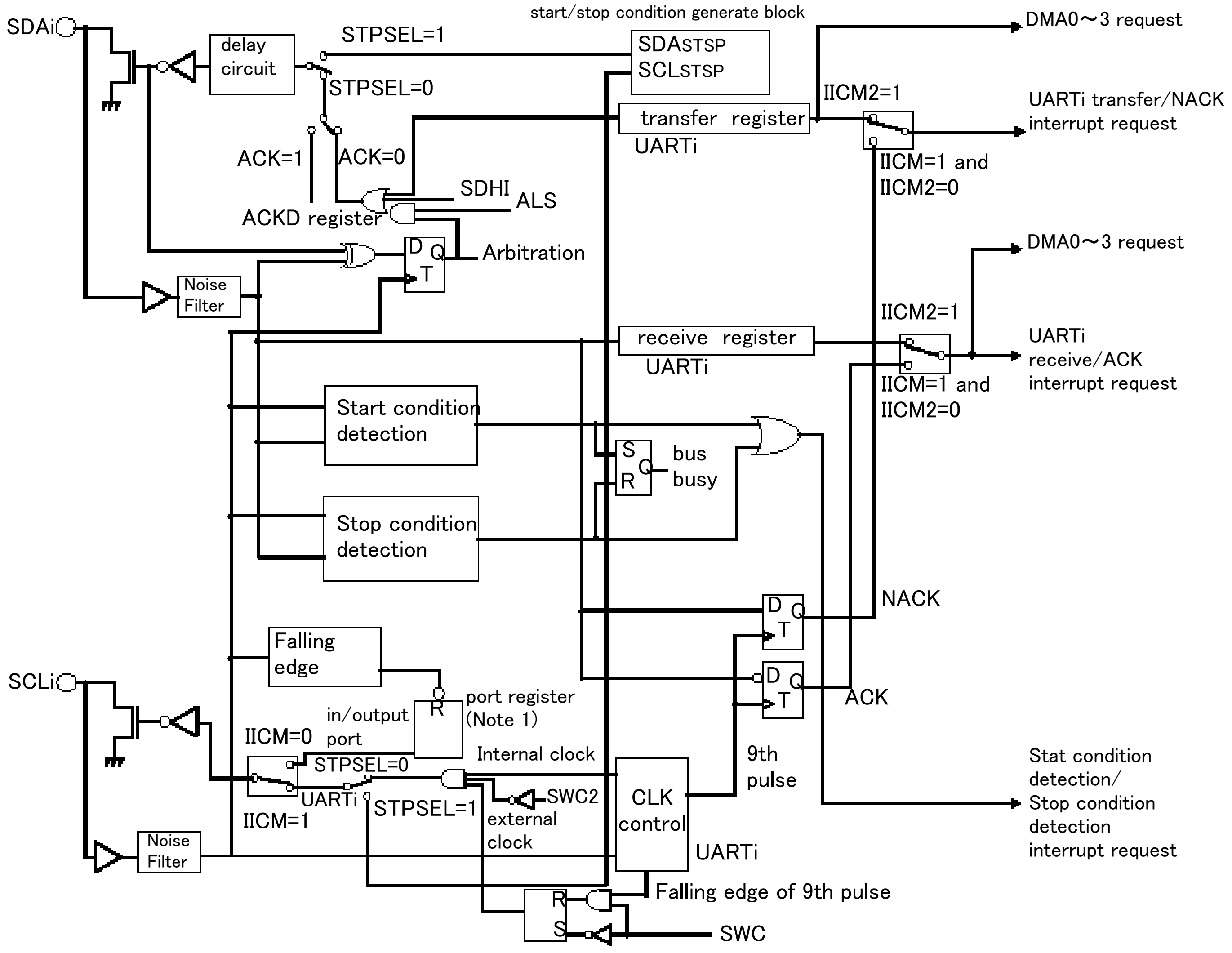
UART channels 0–4 are functionally the same. The IE–Bus or the I^2C bus interface can be used in any of these channels.
UART channels 0–4 each is used in one of three modes available to choose:
Clock–synchronous serial I/O mode, Clock–asynchronous serial I/O mode or I^2C interface mode.
The M32C/83,85 has bus collision detection and other necessary functions to materialize the IEBus interface.
For details about these functions, see the M32C/83,85 Data Sheet.
A block diagram of simple I^2C bus mode and various related registers in the M32C/83,85 are explained in this chapter. Various functions necessary to materialize the I^2C bus interface in the M32C/83,85 are detailed in the next chapter.

Configuration of the serial I/O in the M32C/83,85

- ·UART0
  - Clock synchronous serial I/O
    - **I^2C interface functions**
  - Clock asynchronous serial I/O
    - IEBus interface functions
- ·UART1
  - Clock synchronous serial I/O
    - **I^2C interface functions**
  - Clock asynchronous serial I/O
    - IEBus interface functions
- ·UART2
  - Clock synchronous serial I/O
    - **I^2C interface functions**
  - Clock asynchronous serial I/O
    - IEBus interface functions
- ·UART3
  - Clock synchronous serial I/O
    - **I^2C interface functions**
  - Clock asynchronous serial I/O
    - IEBus interface functions
- ·UART4
  - Clock synchronous serial I/O
    - **I^2C interface functions**
  - Clock asynchronous serial I/O
    - IEBus interface functions

1.2 Simple I^2C Bus Mode Block Diagram

Simple I^2C bus mode is used to materialize the I^2C bus interface. Simple I^2C bus mode is entered into by setting the SMD0-2 register bits to '010B' and the I^2C mode select bit [IICM] to 1. This enables the circuit necessary to materialize the I^2C bus interface.
A block diagram of simple I^2C bus mode is shown below.



i = 0-4
This block diagram is for the case where the UiMR register SMD2-0 bits = 010_2
and the UiSMR register IICM bit = 1.
IICM: UiSMR register bit
IICM2: UiSMR2 register bit
Note1: While the IICM bit = 1, even if the direction bit for any pin is set to 1 (= output), the pin can be read.

1.3 Changeable Pin Functions and Interrupt Sources in Simple I^2C Bus Mode

The table below shows the function of each pin when simple I^2C bus mode is selected.

Pin function	Simple I^2C bus mode	Normal mode
P6_3 pin function	SDA0 (input/output)	TXD0 (output)
Start value of P6_3 output	When serial I/O is disabled, the value set in P6_2.	H level (when CLK polarity select bit=0)
P6_2 pin function	SCL0 (input/output)	RXD0 (input)
Read of the P6_2 pin	The pin is read no matter how the direction register is set.*	The pin is read , when the direction register is 0.
P6_1 pin function	Port6_1	CLK0
P6_7 pin function	SDA1 (input/output)	TXD1(output)
Start value of P6_7 output	When serial I/O is disabled, the value set in P6_7.	H level (when CLK polarity select bit=0)
P6_6 pin function	SCL1 (input/output)	RXD1(input)
Read of the P6_6 pin	The pin is read no matter how the direction register is set.*	The pin is read , when the direction register is 0.
P6_5 pin function	Port6_5	CLK1
P7_0 pin function	SDA2 (input/output)	TXD2(output)
Start value of P7_0 output	When serial I/O is disabled, the value set in P7_0.	H level (when CLK polarity select bit=0)
P7_1 pin function	SCL2 (input/output)	RXD2(input)
Read of the P7_1 pin	The pin is read no matter how the direction register is set.*	The pin is read , when the direction register is 0.
P7_2 pin function	Port7_2	CLK2
P9_2 pin function	SDA3 (input/output)	TXD3(output)
Start value of P9_2 output	When serial I/O is disabled, the value set in P9_2.	H level (when CLK polarity select bit=0)
P9_1 pin function	SCL3 (input/output)	RXD3(input)
Read of the P9_1 pin	The pin is read no matter how the direction register is set.*	The pin is read , when the direction register is 0.
P9_0 pin function	Port9_0	CLK3
P9_6 pin function	SDA4 (input/output)	TXD4(output)
Start value of P9_6 output	When serial I/O is disabled, the value set in P9_6.	H level (when CLK polarity select bit=0)
P9_7 pin function	SCL4 (input/output)	RXD4(input)
Read of the P9_7 pin	The pin is read no matter how the direction register is set.*	The pin is read , when the direction register is 0.
P9_5 pin function	Port9_5	CLK4

Precautions on using bit manipulating instructions for ports

If the data register (port latch) for any input/output port is rewritten using a bit manipulating instruction, the value of some unspecified bit may change.
Reason: This is because the bit manipulating instructions are read-modify-write type instructions and read or write to the register in bytes.
Therefore, if such an instruction is executed on some bits in any input/output port data register, the following processing is applied to all bits in that data register.

・Bits set for input:
The pin value is read by the CPU, which after bit manipulation is written to the bit.
・Bits set for output:
The data register bit value is read by the CPU, which after bit manipulation is written back to the bit.

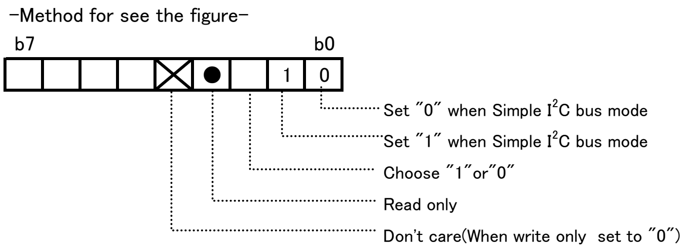
*: Be aware that if a read-modify-write instruction is executed on any port, the SCL or SDA output value may inadvertently be changed.

Interrupt sources
(i=0–4)

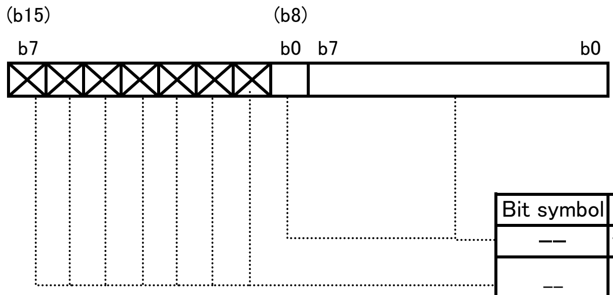
Function	Simple I^2C bus mode(IICM=1)		Normal mode(IICM=0)
	[IICM2] =0	[IICM2] =1	
Interrupt sources of interrupt numbers 39,40 and 41(NOTE 1)	Start/Stop condition detection	Start/Stop condition detection	Bus collision detection
Interrupt sources of interrupt numbers 17,19,33,35 and 37	No acknowledgment detection (NACK)	UARTi transfer	UARTi transfer
Interrupt sources of interrupt numbers 18,20,34,36 and 38	Acknowledgment detection (ACK)	UARTi receive	UARTi receive
DMA sources	Acknowledgment detection (ACK)	UARTi receive	UARTi receive

Note 1: Interrupt sources of interrupt numbers 40 and 41 are assigned to UART0/3 and UART1/4, respectively.
Therefore,UART0 or 3 and UART1 or 4 must be chose one,when you use.

1.4 Register Settings during Simple I^2C Bus Mode


–Method for see the figure–
b7
b0
1
0
Set "0" when Simple I2C bus mode
Set "1" when Simple I2C bus mode
Choose "1"or"0"
Read only
Don't care(When write only set to "0")

UARTi Transmit Buffer Register (Note 1) (i=0–4)

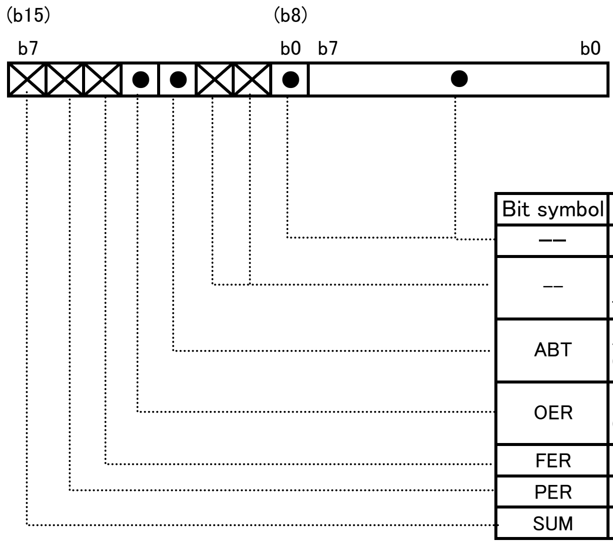


Symbol	Address	When reset
U0TB	$036B_{16}$, $036A_{16}$	Indeterminate
U1TB	$02EB_{16}$, $02EA_{16}$	Indeterminate
U2TB	$033B_{16}$, $033A_{16}$	Indeterminate
U3TB	$032B_{16}$, $032A_{16}$	Indeterminate
U4TB	$02FB_{16}$, $02FA_{16}$	Indeterminate

Bit symbol	Function	R	W
––	Transmit data(The bit8 is ACK)	––	○
––	No functions are assigned. To write to these bits, write 0. When read, the values of these bits are indeterminate.	––	––

Note 1: Use the MOV instruction to write to this register.

UARTi Receive Buffer Register (i=0–4)

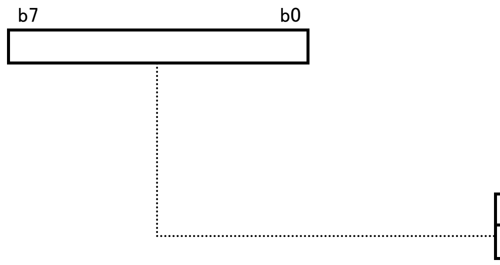


Symbol	Address	When reset
U0RB	$036F_{16}$, $036E_{16}$	Indeterminate
U1RB	$02EF_{16}$, $02EE_{16}$	Indeterminate
U2RB	$033F_{16}$, $033E_{16}$	Indeterminate
U3RB	$032F_{16}$, $032E_{16}$	Indeterminate
U4RB	$02FF_{16}$, $02FE_{16}$	Indeterminate

Bit symbol	Bit name	Function	R	W
––	Receive data(The bit8 is ACK or R/W bit)		○	––
––	No functions are assigned. To write to these bits, write 0. When read, the values of these bits are indeterminate.		––	––
ABT	Arbitration lost flag (Note 1)	0:No detection(win) 1:Detection(lost)	○	○
OER	Overrun error flag (Note 2)	0:Overrun error not occurred 1:Overrun error occurred	○	––
FER	Framing error flag	Has no effect during simple I^2C bus mode.	○	––
PER	Parity error flag	Has no effect during simple I^2C bus mode.	○	––
SUM	Error sum flag	Has no effect during simple I^2C bus mode.	○	––

Note 1: Only writing 0 is accepted.
Note 2: This bit is cleared to 0 by setting the serial I/O mode select bits (address 036816, 02E816, 033816, 032816 or 02F816, bits 2–0) to '000_2' or the receive enable bit to 0.

UARTi Baud Rate Register Notes 1, 2 (i=0–4)



Symbol	Address	When reset
U0BRG	0369_{16}	Indeterminate
U1BRG	$02E9_{16}$	Indeterminate
U2BRG	0339_{16}	Indeterminate
U3BRG	0329_{16}	Indeterminate
U4BRG	$02F9_{16}$	Indeterminate

Bit symbol	Functon	Values that can be set	R	W
––	Assuming the set value = n, BRG divides the count source by n + 1.	00_{16}–FF_{16}	––	○

Note 1: Use the MOV instruction to write to this register.
Note 2: Make sure transmission is inactive when writing to this register.

UARTi Transmit/receive mode Register 0 (i=0–4)

b7	b6	b5	b4	b3	b2	b1	b0
0	☒	☒	☒		0	1	0

Symbol	Address	When reset
U0MR	0368_{16}	00_{16}
U1MR	$02E8_{16}$	00_{16}
U2MR	0338_{16}	00_{16}
U3MR	0328_{16}	00_{16}
U4MR	$02F8_{16}$	00_{16}

Bit symbol	Bit name	Function	R	W
SMD0	Serial I/O mode select bit	000 : Serial I/O is no effect(Port control)	○	○
SMD1		010 : Simple I^2C Bus Mode	○	○
SMD2		(Note 1)	○	○
CKDIR	Internal/external clock select bit	0 : Internal clock 1 : external clock	○	○
STPS	Stop bit length select bit	Has no effect during simple I^2C bus mode.	○	○
PRY	Odd/even parity select bit	Has no effect during simple I^2C bus mode.	○	○
PRYE	Parity enable bit	Has no effect during simple I^2C bus mode.	○	○
IOPOL	TxD,RxD input/ output polarity switch bit	0 : reversed 1 : No reversed (Note 2)	○	○

Note1:To select simple I^2C bus mode, make sure the serial I/O mode select bits are set to '010_2.'

Note 2:In simple I^2C bus mode, set this bit to 1.

UARTi Transmit/receive Control Register 0 (i=0–4)

b7	b6	b5	b4	b3	b2	b1	b0
1	0	1	1	●	☒		

Symbol	Address	When reset
U0C0	$036C_{16}$	08_{16}
U1C0	$02EC_{16}$	08_{16}
U2C0	$033C_{16}$	08_{16}
U3C0	$032C_{16}$	08_{16}
U4C0	$02FC_{16}$	08_{16}

Bit symbol	Bit name	Function	R	W
CLK0	BRG count source select bit	00 : f1 is selected 01 : f8 is selected	○	○
CLK1		10 : f2n is selected 11:Must not be set	○	○
CRS	$\overline{\text{CST}}$/$\overline{\text{RTS}}$ function select bit	Has no effect during simple I^2C bus mode.	○	○
TXEPT	Transmit register empty flag	0 : Data present in transmit register (during transmission) 1:No data present in transmit register (transmission completed)	○	––
CRD	$\overline{\text{CTS}}$/$\overline{\text{RTS}}$ disable bit	0 : $\overline{\text{CTS}}$/$\overline{\text{RTS}}$ function enabled 1 : $\overline{\text{CTS}}$/$\overline{\text{RTS}}$ function disabled In simple I^2C bus mode, set this bit to 1.	○	○
NCH	Data output select bit (Note1)	0 : TXDi pin is CMOS output 1 : TXDi pin is N–channel open drain output In simple I^2C bus mode, set this bit to 1.	○	○
CKPOL	CLK polarity select bit	0:Transmit data is output at falling edge of transfer clock and receive data is input at rising edge 1 :Transmit data is output at rising edge of transfer clock and receive data is input at falling edge In simple I^2C bus mode, set this bit to 0.	○	○
UFORM	Transfer format select bit	0 : LSB first 1 : MSB first In simple I^2C bus mode, set this bit to 1.	○	○

Note 1: The UART2 SDA and SCL pins are N–channel open–drain pins.
Therefore, CMOS output cannot be selected for these pins. When write , set U2C0's bit 5 to 0.

UARTi Transmit/receive Control Register 1 (i=0–4)

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0		●		●	

Symbol	Address	When reset
U0C1	$036D_{16}$	02_{16}
U1C1	$02ED_{16}$	02_{16}
U2C1	$033D_{16}$	02_{16}
U3C1	$032D_{16}$	02_{16}
U4C1	$02FD_{16}$	02_{16}

Bit symbol	Bit name	Function	R	W
TE	Transmit enable bit	0:Transmission disabled 1: Transmission enabled	○	○
TI	Transmit buffer empty flag	0: Data present in transmit buffer register 1: No data present in transmit buffer register	○	--
RE	Receive enable bit	0: Reception disabled 1: Reception enabled	○	○
RI	Receive complete flag	0: Data present in receive buffer register 1: No data present in receive buffer register	○	--
UiIRS	UARTi transmit interrupt cause select bit (Note1)	0: Transmit buffer empty (TI = 1) 1: Transmit is completed (TXEPT = 1)	○	○
UiRRM	UARTi continuous receive mode enable bit	0: Continuous receive mode disabled 1: Continuous receive mode enabled In simple I^2C bus mode, set this bit to 0.	○	○
UiLCH	Data logic select bit	0: No reverse 1: Reverse In simple I^2C bus mode, set this bit to 0.	○	○
UiERE	Clock divide synchronizing stop bit /error signal output enable bit	0: Synchronizing stop 1: Synchronous start In simple I^2C bus mode, set this bit to 0.	○	○

Note 1: UiIRS has no effect when IICM = 1 and IICM2 = 0.

UARTi special mode register (i=0–4)

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	0	●		1

Symbol	Address	When reset
U0SMR	0367_{16}	00_{16}
U1SMR	$02E7_{16}$	00_{16}
U2SMR	0337_{16}	00_{16}
U3SMR	0327_{16}	00_{16}
U4SMR	$02F7_{16}$	00_{16}

Bit symbol	Bit name	Function	R	W	Related section
IICM	IIC mode select bit(Note1)	0: Normal mode 1: IIC mode	○	○	1.3
ABC	Arbitration lost detecting flag control bit	0: Update per bit 1: Update per byte	○	○	2.5
BBS	Bus busy flag(Note2)	0: STOP condition detected 1: START condition detected	○	○	2.2
LSYN	SCLL sync output enable bit	0: Disabled 1: Enabled In simple I^2C bus mode, set this bit to 0.	○	○	----
ABSCS	Bus collision detect sampling clock select bit	set this bit to 0.	○	○	----
ACSE	Auto clear function select bit of transmit enable bit	set this bit to 0.	○	○	----
SSS	Transmit start condition select bit	set this bit to 0.	○	○	----
SCLKDIV	Clock divide set bit	set this bit to 0.	○	○	----

Note 1 : To select simple I^2C bus mode, make sure the serial I/O mode select bits are set to '010_2.'
Note 2: Only writing 0 is accepted.

UARTi special mode register 2 (i=0–4)

b7	b6	b5	b4	b3	b2	b1	b0
0							

Symbol	Address	When reset
U0SMR2	0366_{16}	00_{16}
U1SMR2	$02E6_{16}$	00_{16}
U2SMR2	0336_{16}	00_{16}
U3SMR2	0326_{16}	00_{16}
U4SMR2	$02F6_{16}$	00_{16}

Bit symbol	Bit name	Function	R	W	Related section
IICM2	IIC mode select bit 2	See Table1	○	○	----
CSC	Clock synchronous bit	0: Disabled 1: Enabled	○	○	2.5
SWC	SCL wait output bit	0: Disabled 1: Enabled	○	○	2.4
ALS	SDA output stop bit	0: Disabled 1: Enabled	○	○	2.5
STAC	UARTi initialize bit	0: Disabled 1: Enabled	○	○	2.6
SWC2	SCL wait output bit 2	0: UARTi clock 1: 0 output	○	○	2.2
SDHI	SDA output inhibit bit	0: Disabled 1: Enabled (high impedance)	○	○	2.6
SU1HIM	External clock synchronizing enable bit	0: Synchronous disabled 1: Synchronous enabled In simple I²C bus mode, set this bit to 0.	○	○	----

Table 1. Functions during Simple I²C Bus Mode (IICM = 1)

Function	[IICM2] =0	[IICM2] =1
Interrupt sources of interrupt numbers 39,40 and 41(Note 1)	Start/Stop condition detection	Start/Stop condition detection
Interrupt sources of interrupt numbers 17,19,33,35 and 37	No acknowledgment detection (NACK)	UARTi transfer
Interrupt sources of interrupt numbers 18,20,34,36 and 38	Acknowledgment detection (ACK)	UARTi receive
DMA sources	Acknowledgment detection (ACK)	UARTi receive
The timing at which data is transferred from the UARTi receive sift register to the receive buffer register	The last receive clock pulse goes high	The last receive clock pulse goes low
The timing at whitch a UARTi-receive/ Acknowledge-detected interrupt request generated	The last receive clock pulse goes high (Acknowledge detected)	The last receive clock pulse goes low (UARTi receive)

Note 1: Interrupt sources of interrupt numbers 40 and 41 are assigned to UART0/3 and UART1/4, respectively.
Therefore,UART0 or 3 and UART1 or 4 must be chose one,when you use.

UARTi special mode register 3 (i=0～4)

b7	b6	b5	b4	b3	b2	b1	b0
			0	0	0		0

Symbol	Address	When reset
U0SMR3	0365_{16}	00_{16}
U1SMR3	$02E5_{16}$	00_{16}
U2SMR3	0335_{16}	00_{16}
U3SMR3	0325_{16}	00_{16}
U4SMR3	$02F5_{16}$	00_{16}

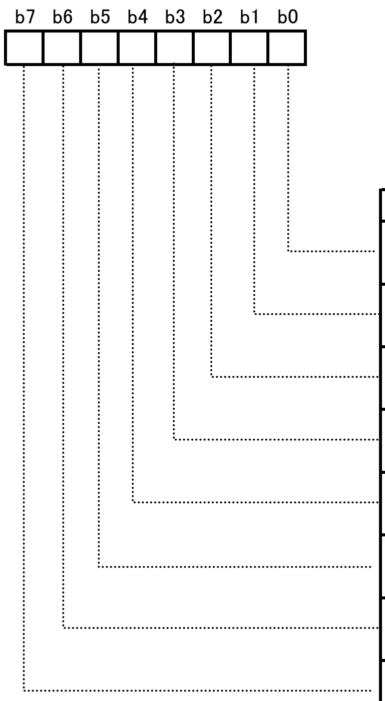
Bit symbol	Bit name	Function	R	W	Related section
SSE	$\overline{SS}$ port function enable bit	0: $\overline{SS}$ function disabled 1: $\overline{SS}$ function enabled In simple I^2C bus mode, set this bit to 0.	○	○	----
CKPH	Clock phase set bit	0: Without clock delay 1: With clock delay	○	○	2.6
DINC	Serial input port set bit	0: Select TxDi and RxDi (master mode) 1: Select STxDi and SRxDi (slave mode) In simple I^2C bus mode, set this bit to 0.	○	○	----
NODC	Clock output select bit	0: CLKi is CMOS output 1: CLKi is N-channel open drain output In simple I^2C bus mode, set this bit to 0.	○	○	----
ERR	Fault error flag	0: Without fault error 1: With fault error In simple I^2C bus mode, set this bit to 0.	○	○	----
DL0	SDAi(TxDi) digital delay time set bit (Note 1,2)	000 :Without delay 001 :2-cycle of BRG count source 010 :3-cycle of BRG count source 011 :4-cycle of BRG count source 100 :5-cycle of BRG count source 101 :6-cycle of BRG count source 110 :7-cycle of BRG count source 111 :8-cycle of BRG count source	○	○	2.6
DL1			○	○	
DL2			○	○	

Note 1: These bits provide a digital means of generating a delay in SDAi (TxDi) output when using UARTi as the I^2C bus interface.
Otherwise, always be sure to set these bits to '000_2.'

Note 2: If an external clock is selected, the actual delay is greater by about 100 ns than the set value.

UARTi special mode register 4 (i=0～4)

Symbol	Address	When reset
U0SMR4	0364_{16}	00_{16}
U1SMR4	$02E4_{16}$	00_{16}
U2SMR4	0334_{16}	00_{16}
U3SMR4	0324_{16}	00_{16}
U4SMR4	$02F4_{16}$	00_{16}



Bit symbol	Bit name	Function	R	W	Related section
STAREQ	Start condition generate bit (Note1)	0: Clear 1: Start	○	○	2.2
RSTREQ	Restart condition generate bit(Note1)	0: Clear 1: Start	○	○	2.2
STPREQ	Stop condition generate bit (Note1)	0: Clear 1: Start	○	○	2.2
STSPSEL	SCL, SDA output select bit	0: Ordinal block 1: Start/stop condition generate block	○	○	2.2
ACKD	ACK data bit	0: ACK 1: NACK	○	○	2.3
ACKC	ACK data output enable bit	0: SI/O data output 1: ACKD output	○	○	2.3
SCLH	SCL output stop enable bit	0: Disabled 1: Enabled	○	○	2.6
SWC9	SCL wait output bit 3	0: SCL ″L″ hold disabled 1: SCL ″L″ hold enabled	○	○	2.6

Note 1: When each condition is generated, the bit is automatically cleared to 0.

External interrupt request cause select register

b7 b6 b5 b4 b3 b2 b1 b0

Symbol	Address	When reset
IFSR	$031F_{16}$	00_{16}

Bit symbol	Bit name	Function	R	W
IFSR0	INT0 interrupt polarity select bit(Note1,2)	0 : One edge 1 : Both edges	○	○
IFSR1	INT1 interrupt polarity select bit(Note1,2)	0 : One edge 1 : Both edges	○	○
IFSR2	INT2 interrupt polarity select bit(Note1,2)	0 : One edge 1 : Both edges	○	○
IFSR3	INT3 interrupt polarity select bit(Note1,2)	0 : One edge 1 : Both edges	○	○
IFSR4	INT4 interrupt polarity select bit(Note1,2)	0 : One edge 1 : Both edges	○	○
IFSR5	INT5 interrupt polarity select bit(Note1,2)	0 : One edge 1 : Both edges	○	○
IFSR6	UART0/3 interrupt cause select bit	0 : UART3 bus collision /start,stop detect/ false error detect 1 : UART0 bus collision /start,stop detect/ false error detect	○	○
IFSR7	UART1/4 interrupt cause select bit	0 : UART4 bus collision /start,stop detect/ false error detect 1 : UART1 bus collision /start,stop detect/ false error detect	○	○

Note 1: These bits are irrelevant to the simple I^2C bus.
Note 2: If "Level sense" is selected, set this bit to 0.
To select "Both edges," make sure the corresponding INTi Interrupt Control Register's polarity select bit (bit 4) is set to 0 (= falling edge).

Function select register A0

b7 b6 b5 b4 b3 b2 b1 b0

Symbol	Address	When reset
PS0	$03B0_{16}$	00_{16}

Bit symbol	Bit name	Function	R	W
PS0_0	Port P60 function select bit (Note1)	0 : I/O port 1 : UART0 output ($\overline{RTS0}$)	○	○
PS0_1	Port P61 function select bit (Note1)	0 : I/O port 1 : UART0 output (CLK0 output)	○	○
PS0_2	Port P62 function select bit	0 : I/O port 1 : Function that was selected in bit2 of PSL0	○	○
PS0_3	Port P63 function select bit	0 : I/O port 1 : UART0 output (TXD0/SDA0)	○	○
PS0_4	Port P64 function select bit (Note1)	0 : I/O port 1 : Function that was selected in bit4 of PSL0	○	○
PS0_5	Port P65 function select bit (Note1)	0 : I/O port 1 : UART1 output (CLK1 output)	○	○
PS0_6	Port P66 function select bit	0 : I/O port 1 : Function that was selected in bit6 of PSL0	○	○
PS0_7	Port P67 function select bit	0 : I/O port 1 : UART1 output (TXD1/SDA1)	○	○

Note 1: These bits are irrelevant to the simple I^2C bus.

Function select register A1

b7 b6 b5 b4 b3 b2 b1 b0

Symbol	Address	When reset
PS1	$03B1_{16}$	00_{16}

Bit symbol	Bit name	Function	R	W
PS1_0	Port P70 function select bit	0 : I/O port 1 : Function that was selected in bit0 of PSL1	○	○
PS1_1	Port P71 function select bit	0 : I/O port 1 : Function that was selected in bit1 of PSL1	○	○
PS1_2	Port P72 function select bit (Note1)	0 : I/O port 1 : Function that was selected in bit2 of PSL1	○	○
PS1_3	Port P73 function select bit	0 : I/O port 1 : Function that was selected in bit3 of PSL1	○	○
PS1_4	Port P74 function select bit (Note1)	0 : I/O port 1 : Function that was selected in bit4 of PSL1	○	○
PS1_5	Port P75 function select bit (Note1)	0 : I/O port 1 : Function that was selected in bit5 of PSL1	○	○
PS1_6	Port P76 function select bit (Note1)	0 : I/O port 1 : Function that was selected in bit6 of PSL1	○	○
PS1_7	Port P77 function select bit (Note1)	0 : I/O port 1 : Intelligent I/O group 0 output (OUTC01/ISCLK0)	○	○

Note 1: These bits are irrelevant to the simple I^2C bus.

Function select register A3 (Note1)

Symbol	Address	When reset
PS3	$03B5_{16}$	00_{16}

b7	b6	b5	b4	b3	b2	b1	b0

Bit symbol	Bit name	Function	R	W
PS3_0	Port P90 function select bit (Note2)	0 : I/O port 1 : UART3 output (CLK3)	○	○
PS3_1	Port P91 function select bit	0 : I/O port 1 : Function that was selected in bit1 of PSL3	○	○
PS3_2	Port P92 function select bit	0 : I/O port 1 : Function that was selected in bit2 of PSL3	○	○
PS3_3	Port P93 function select bit (Note2)	0 : I/O port 1 : UART3 output ($\overline{RTS3}$)	○	○
PS3_4	Port P94 function select bit (Note2)	0 : I/O port 1 : UART4 output ($\overline{RTS4}$)	○	○
PS3_5	Port P95 function select bit (Note2)	0 : I/O port 1 : UART4 output (CLK4)	○	○
PS3_6	Port P96 function select bit	0 : I/O port 1 : UART4 output (TXD4/SDA4)	○	○
PS3_7	Port P97 function select bit	0 : I/O port 1 : Function that was selected in bit7 of PSL3	○	○

Note 1: To rewrite this register, make sure the PRCR register PRC2 bit is set to 1 (write enabled).
Note 2: These bits are irrelevant to the simple I^2C bus.

Function select register B0

Symbol	Address	When reset
PSL0	$03B2_{16}$	00_{16}

b7	b6	b5	b4	b3	b2	b1	b0
0		0		0		0	0

Bit symbol	Bit name	Function	R	W
―――	Reserve bit	Must always be "0".	○	○
―――			○	○
PSL0_2	Port P62 peripheral function select bit	0 : UART0 output (SCL0) 1 : UART0 output (STXD0)	○	○
―――	Reserve bit	Must always be "0".	○	○
PSL0_4	Port P64 peripheral function select bit(Note1)	0 : UART1 output ($\overline{RTS1}$) 1 : Intelligent I/O group 2 output (OUTC21/ISCLK2)	○	○
―――	Reserve bit	Must always be "0".	○	○
PSL0_6	Port P66 peripheral function select bit	0 : UART1 output (SCL1) 1 : UART1 output (STXD1)	○	○
―――	Reserve bit	Must always be "0".	○	○

Note 1: This bit is irrelevant to the simple I^2C bus.

Function select register B1

Symbol	Address	When reset
PSL1	03B3$_{16}$	00$_{16}$

b7	b6	b5	b4	b3	b2	b1	b0
0							

Bit symbol	Bit name	Function	R	W
PSL1_0	Port P70 peripheral function select bit	0 : Function that was selected in bit0 of PSC 1 : Timer output (TA0OUT)	○	○
PSL1_1	Port P71 peripheral function select bit	0 : Function that was selected in bit1 of PSC 1 : UART2 output (STXD2)	○	○
PSL1_2	Port P72 peripheral function select bit(Note1)	0 : Function that was selected in bit2 of PSC 1 : Timer output (TA1OUT)	○	○
PSL1_3	Port P73 peripheral function select bit(Note1)	0 : Function that was selected in bit3 of PSC 1 : Three-phase PWM output (V)	○	○
PSL1_4	Port P74 peripheral function select bit(Note1)	0 : Function that was selected in bit4 of PSC 1 : Three-phase PWM output (W)	○	○
PSL1_5	Port P75 peripheral function select bit(Note1)	0 : Function that was selected in bit5 of PSC 1 : Intelligent I/O group 1 output (OUTC12)	○	○
PSL1_6	Port P76 peripheral function select bit(Note1)	0 : Function that was selected in bit16of PSC 1 : Timer output (TA3OUT)	○	○
---	Reserve bit	Must always be "0".	○	○

Note 1: These bits are irrelevant to the simple I^{2}C bus.

Fu Function select register B3

Symbol	Address	When reset
PSL3	03B3$_{16}$	00$_{16}$

b7	b6	b5	b4	b3	b2	b1	b0
							0

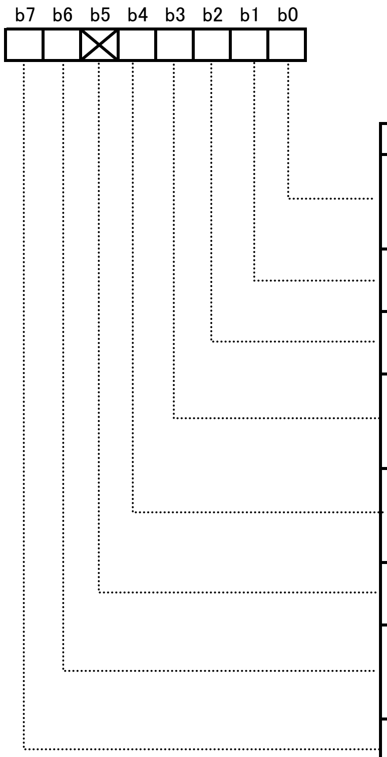
Bit symbol	Bit name	Function	R	W
---	Reserve bit	Must always be "0".	○	○
PSL3_1	Port P91 peripheral function select bit	0 : UART3 output (SCL3) 1 : UART3 output (STXD3)	○	○
PSL3_2	Port P92 peripheral function select bit	0 : UART3 output (TXD3/SDA3) 1 : Intelligent I/O group 2 output (OUTC20/IEOUT)	○	○
PSL3_3	Port P93 peripheral function select bit(Note1,2)	0 : Input peripheral function enabled(Expect DA0 output) 1 : Input peripheral function disabled (DA0 output)	○	○
PSL3_4	Port P94 peripheral function select bit(Note1,2)	0 : Input peripheral function enabled(Expect DA1 output) 1 : Input peripheral function disabled (DA1 output)	○	○
PSL3_5	Port P95 peripheral function select bit(Note1,2)	0 : Input peripheral function enabled(Expect ANEX0 output) 1 : Input peripheral function disabled (ANEX0 output)	○	○
PSL3_6	Port P96 peripheral function select bit	0 : Input peripheral function enabled(Expect ANEX1 output) 1 : Input peripheral function disabled (ANEX1 output)	○	○
PSL3_7	Port P97 peripheral function select bit(Note2)	0 : UART4 output (SCL4) 1 : UART4 output (STXD4)	○	○

Note 1: These bits are irrelevant to the simple I^{2}C bus.

Note 2: DA0, DA1, ANEX0 or ANEX1 can be used even when these bits are set to 0, in which case the power supply current may increase, however.

Function select register C

Symbol	Address	When reset
PSC	$03AF_{16}$	$00X0\ 0000_2$



Bit symbol	Bit name	Function	R	W
PSC_0	Port P70 peripheral function select bit	0 : UART2 output (TXD2/SDA2) 1 : Intelligent I/O group 2 output (OUTC20/ ISTXD2/IEOUT)	○	○
PSC_1	Port P71 peripheral function select bit	0 : UART2 output (SCL2) 1 : Intelligent I/O group 2 output (OUTC22)	○	○
PSC_2	Port P72 peripheral function select bit(Note1)	0 : UART2 output (CLK2) 1 : Three-phase PWM output (V)	○	○
PSC_3	Port P73 peripheral function select bit(Note1)	0 : UART2 output ($\overline{RTS2}$) 1 : Intelligent I/O group 1 output (OUTC10/ ISTXD1/BE1OUT)	○	○
PSC_4	Port P74 peripheral function select bit(Note1)	0 : Timer output (TA2OUT) 1 : Intelligent I/O group 1 output (OUTC11/ ISCLK1)	○	○
–––	Noting is assigned. When write, set to "0". When read, its content is indeterminate.		––	––
PSC_6	Port P76 peripheral function select bit(Note1)	0 : Intelligent I/O group 0 output (OUTC00/ISTXD0/BE0OUT) 1 : CAN output (CANOUT)	○	○
PSC_7	Port P77 peripheral function select bit(Note1)	0 : Enabled 1 : Disabled	––	○

Note 1: These bits are irrelevant to the simple I^2C bus.

Serial I/O used in simple I^2C bus mode	Method for set function select register
UART0	PS0_2=1 PS0_3=1 PSL0_2=0
UART1	PS0_6=1 PS0_7=1 PSL0_6=0
UART2	PS1_0=1 PS1_1=1 PSL1_0=0 PSL1_1=0 PSC_0=0 PSC_1=0
UART3	PS3_1=1 PS3_2=1 PSL3_1=0 PSL3_2=0
UART4	PS3_6=1 PS3_7=1 PSL3_6=0

Chapter 2
Each Function of Simple I^{2}C Bus Mode

2.1 Method for Sending and Receiving Byte Data
2.2 Start and Stop Conditions
2.3 Acknowledge
2.4 Judgment of the Specified Local Address
2.5 Arbitrating Contention for Communication
2.6 Other Functions

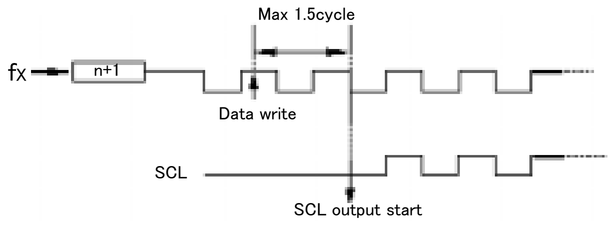
This chapter explains how to use each hardware function of simple I^{2}C bus mode in order to materialize the I^{2}C bus interface when using the M32C/83,85 in simple I^{2}C bus mode.

2.1 Method for Sending and Receiving Byte Data

The following explains how to set the registers to send SCL in simple I^{2}C bus mode by using the M32C/83,85 as the master, as well as how to set the registers to send and receive one byte of data.

Method for generating SCL (during master)
Before the M32C/83,85 can be used as the master, the speed of the transmit clock (SCL) must be set.
To set it, use the registers described below as in the case of ordinary serial I/O transmission.
SCL is sent out within 1.5 SCL cycles after writing data to the transmit buffer.

[SCL transmit timing]
fx: BRG count source
n: UiBRG set value (i = 0–4)
SCL: Waveform when CKPH = 1



For details about CKPH, see the clock delay function in Section 2.6, "Other Functions."

[Related Registers]

UARTi transmit/receive mode register (i=0–4)

b7	b6	b5	b4	b3	b2	b1	b0
0	☒	☒	☒	0	0	1	0

U0MR:0368$_{16}$, U1MR:02E8$_{16}$, U2MR:0338$_{16}$
U3MR:0328$_{16}$, U4MR:02F8$_{16}$

[SMD]010: Simple I^{2}C Bus Mode
[CKDIR] 0: Selected internal clock (during master. When slave mode, 1:Selected external clock)
[IOPOL]0:No reversed

UARTi special mode register (i=0–4)

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	0	●		1

U0SMR:0367$_{16}$, U1SMR:02E7$_{16}$, U2SMR:0337$_{16}$
U3SMR:0327$_{16}$, U4SMR:02F7$_{16}$

[IICM]1 : Simple I^{2}C Bus Mode
[ABC] Arbitration-lost is updated 0 : per bit or 1 : per byte

UARTi transmit/receive control register 0 (i=0–4)

b7	b6	b5	b4	b3	b2	b1	b0
1	0	1	1	●	☒		

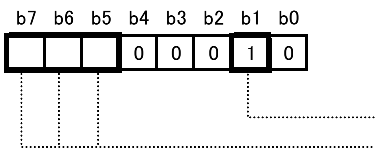
U0C0:036C$_{16}$, U1C0:02EC$_{16}$, U2C0:033C$_{16}$
U3C0:032C$_{16}$, U4C0:02FC$_{16}$

[CLK1] [CLK0] selecte BRG count source
0 0 :f1is selected、0 1 :f8 is selected、1 0 :f2n is selected、1 1 :Must not be set
[CRD] 1 :CTS/RTS function disabled
[NCH] 1 : SCL/SDA pin is N-channel open drain output(Note1)
[CKPOL] 0 :Transmit data is output at falling edge of transfer clock and receive data is input at rising edge
[UFORM] Transfer format select 0 :LSB first or 1 :MSB first

Note 1: The UART2 SDA and SCL pins are N-channel open-drain pins.
CMOS output cannot be selected for these pins. When write , To write to bit 5, write 0.

Method for generating SCL (during master)(Continued from the preceding page)

UARTi special mode register 3 (i=0-4)

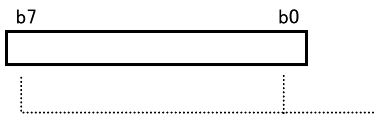


U0SMR3 : 0365_{16} , U1SMR3 : $02E5_{16}$, U2SMR3 : 0335_{16}
U3SMR3 : 0325_{16} , U4SMR3 : $02F5_{16}$

[CKPH] 0 : Without clock delay 1:With clock delay
[DL2][DL1][DL0] This bit set digital delay time
000:Without delay 001 : 2-cycle of BRG count source
010 : 3-cycle of BRG count source 011 : 4-cycle of BRG count source
100 : 5-cycle of BRG count source 101 : 6-cycle of BRG count source
110 : 7-cycle of BRG count source 111 : 8-cycle of BRG count source

For details about CKPH, see the clock delay function in Section 2.6, "Other Functions.
For details about DL2, DL1 and DL0, see the digital output delay function in Section 2.6, Other Functions."

UARTi bit rate generator (i=0-4)



U0BRG : 0369_{16} , U1BRG : $02E9_{16}$, U2BRG : 0339_{16}
U3BRG : 0329_{16} , U4BRG : $02F9_{16}$

BRGi divides the count source by n+1

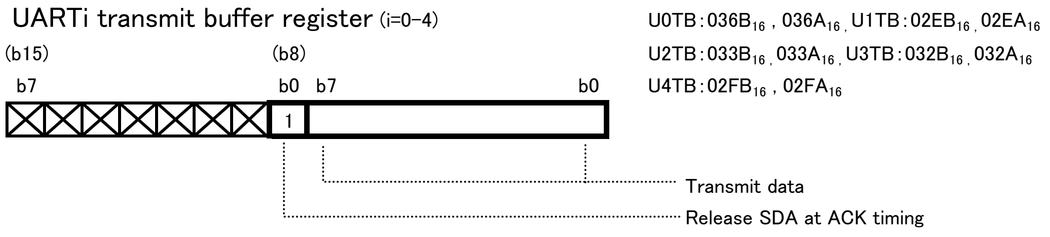
[Example settings]
To set the transmission rate to 100 kbps when using a 10 MHz original oscillator frequency
・UiMR = 00000010_2 (In simple I^2C mode, with internal clock selected)
・UiC0 = 10010000_2 (BRG count source chosen to be f1)
・ UiBRG =49

[Settings during slave]
To use the M32C/83,85 as a slave, set the UARTi Transmit/receive Mode Register (UiMR) bit 3 [CKDIR] = 1 to select an external clock.
In this case, settings of the BRG count source select bits [CLK0], [CLK1] and those of the UARTi Baud Rate Register (UiBRG) have no effect.

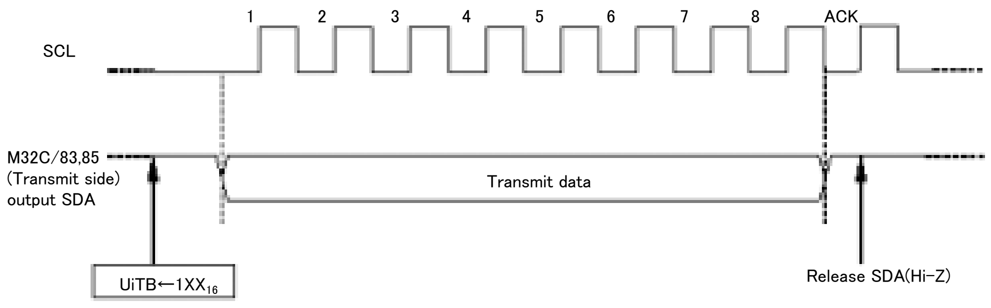
Method for sending byte data

When the M32C/83,85 is operating as a transmitter, 8bits transmit data is sent out from the SDA pin.
In this case,the SDA pin of the M32C/83,85 must be released (high-impedance state) in order to receive an acknowledge signal at the 9th transmit clock pulse. This is accomplished by setting the appropriate data in the transmit buffer. Set 9 bits of data in the transmit buffer. In the I^2C bus, send the data beginning with the MSB.
In the M32C/83,85,if the transfer format is set to MSB first and 9 bits long, the data is sent out in order of bit 7 -- bit 6 -- ...-- bit 0 -- bit 8. Therefore, the timing at which an acknowledge signal can be received is when the MSB bit is sent out.For the SDA pin to be released at this time, set data "1" in the MSB bit, which causes the SDA output of the M32C/83,85 to be placed in the high-impedance state. This is how to send byte data.

[Related Registers]


UARTi transmit buffer register (i=0-4)
(b15)
(b8)
b7
b0
b7
b0
1
Transmit data
Release SDA at ACK timing
U0TB:036B$_{16}$, 036A$_{16}$, U1TB:02EB$_{16}$, 02EA$_{16}$
U2TB:033B$_{16}$, 033A$_{16}$, U3TB:032B$_{16}$, 032A$_{16}$
U4TB:02FB$_{16}$, 02FA$_{16}$

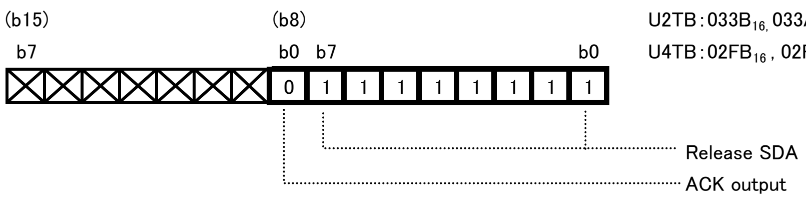
[Timing Figure]



Method for receiving byte data
When the M32C/83,85 is operating as a receiver, the SDA pin of the M32C/83,85 must be released (high-impedance stat
while receiving 8 bits of data from the SDA pin. Furthermore, at the 9th clock pulse, the SDA pin must be pulled low
to generate an acknowledge signal. This operation can be accomplished simply because if judgment of the receiver
address specified on the master side has finished, and if it has been confirmed that data is being sent
to the local device,an acknowledge signal can be sent by writing the appropriate data to the transmit buffer.
Even when receiving data, set 9 bits of data as dummy data in the transmit buffer of the M32C/83,85 as
when sending data. To release the SDA pin while sending 8 bits of data, set data "1" in the 8 low-order bits.
To generate an acknowledge signal, set data '0' in the last bit to be sent (bit 8). This is how to receive byte data.

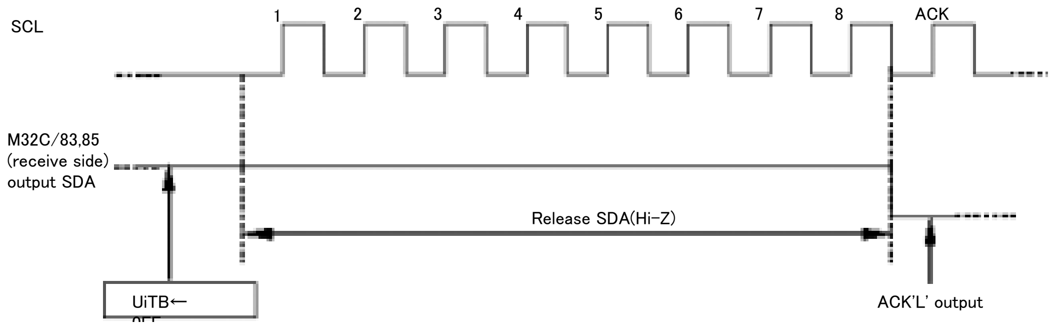
[Related Registers]

UARTi receive buffer register (i =0-4)



U0TB : $036B_{16}$, U1TB : $02EB_{16}$, $02EA_{16}$
U2TB : $033B_{16}$, $033A_{16}$, U3TB : $032B_{16}$, $032A_{16}$
U4TB : $02FB_{16}$, $02FA_{16}$

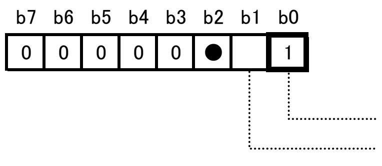
[Timing Figure]



Transmit and receive interrupts
When the M32C/83,85 is operating as a transmitter, completion of data transmission can be detected by a "UARTi Transmit Interrupt." Similarly, when the M32C/83,85 is operating as a receiver, completion of data reception can be detected by a "UARTi Receive Interrupt." These interrupts are assigned to interrupt numbers 17–20 and interrupt numbers 33–38, respectively. The interrupt sources for these interrupt numbers respectively are chosen to be UARTi transmission and UARTi reception by setting the I^{2}C mode select bit 2 [IICM2] = 1. In this case, the timing at which a transmit interrupt is generated is when the start pulse of the transmit clock goes low if UARTi transmit interrupt source select bit [UiIRS] = 0 or when the first bit of the next data goes low if [UiIRS] = 1 (when CKPH = 1). The timing at which a receive interrupt is generated is when the last receive clock pulse goes low.
(For details, see Table 1, "Functions during Simple I^{2}C Bus Mode (IICM = 1)," for UARTi Special Mode Register 2 in Section 1.4, "Register Settings during Simple I^{2}C Bus Mode.")
Be aware that if the receive buffer is read before the last receive clock pulse goes high (e.g., during a reception-finished interrupt in simple I^{2}C bus mode), the received data has its bit positions changed when read out. (See the timing diagram shown in the page that follows.)

[Related Registers]

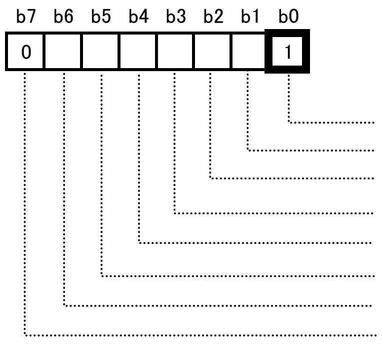
UARTi special mode register (i=0-4)



U0SMR:0367$_{16}$, U1SMR:02E7$_{16}$, U2SMR:0337$_{16}$
U3SMR:0327$_{16}$, U4SMR:02F7$_{16}$

[IICM]1:Simple I^{2}C Bus Mode
[ABC] Arbitration-lost is updated 0:per bit or 1:per byte

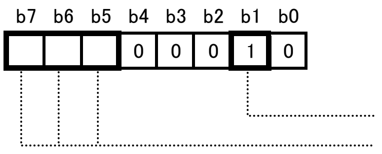
UARTi special mode register 2 (i=0-4)



U0SMR2:0366$_{16}$, U1SMR2:02E6$_{16}$, U2SMR2:0336$_{16}$
U3SMR2:0326$_{16}$, U4SMR2:02F6$_{16}$

[IICM2] 1:UARTi transfer/receive interrupt
[CSC] 0:Clock synchronous is disabled 1:Clock synchronous is enabled
[SWC] 0:SCL wait output is disabled 1:SCL wait output is enabled
[ALS] 0:SDA output stop is disabled 1:SDA output stop is enabled
[STAC] 0:UARTi initialize is disabled 1:UARTi initialize is enabled
[SWC2] 0:UARTi clock 1:SCL output "L"
[SDHI] 0:SDA output enable 1:SDA output disable(Hi-Z)
In simple I2C bus mode, set this bit to 0.

UARTi special mode register 3 (i=0-4)



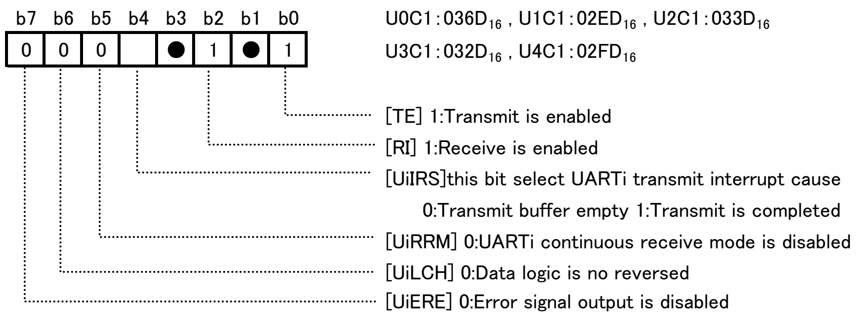
U0SMR3:0365$_{16}$, U1SMR3:02E5$_{16}$, U2SMR3:0335$_{16}$
U3SMR3:0325$_{16}$, U4SMR3:02F5$_{16}$

[CKPH] 1:With clock delay
[DL2][DL1][DL0] This bit set digital delay time
000:Without delay 001:2-cycle of BRG count source
010:3-cycle of BRG count source 011:4-cycle of BRG count source
100:5-cycle of BRG count source 101:6-cycle of BRG count source
110:7-cycle of BRG count source 111:8-cycle of BRG count source

For details about CKPH, see the clock delay function in Section 2.6, "Other Functions.
" For details about DL2, DL1 and DL0, see the digital output delay function in Section 2.6, "Other Functions."

Transmit and receive interrupts (Continued from the preceding page)

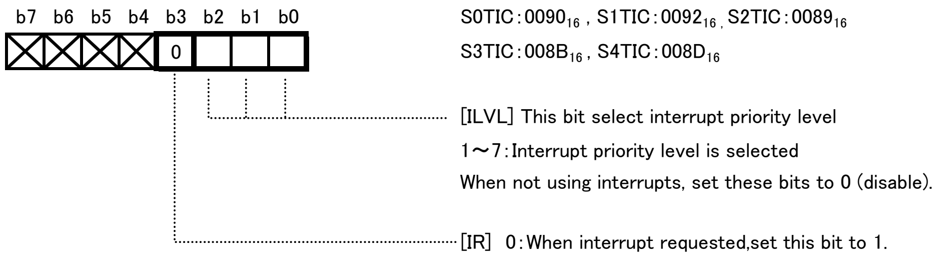
UARTi transmit/receive control register 1 (i=0–4)



U0C1 : $036D_{16}$, U1C1 : $02ED_{16}$, U2C1 : $033D_{16}$
U3C1 : $032D_{16}$, U4C1 : $02FD_{16}$

[TE] 1:Transmit is enabled
[RI] 1:Receive is enabled
[UiIRS]this bit select UARTi transmit interrupt cause
0:Transmit buffer empty 1:Transmit is completed
[UiRRM] 0:UARTi continuous receive mode is disabled
[UiLCH] 0:Data logic is no reversed
[UiERE] 0:Error signal output is disabled

UARTi Transmit interrupt control register (i=0～4)

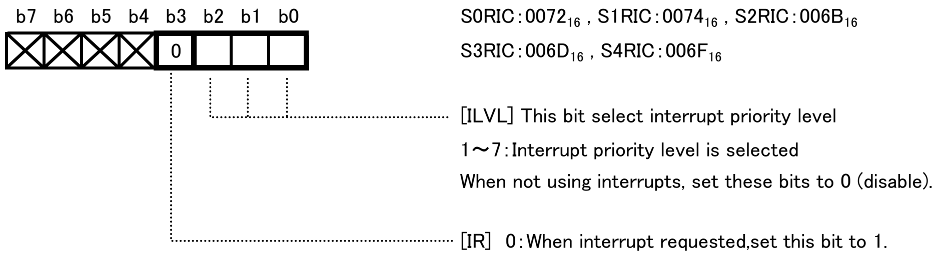


S0TIC : 0090_{16} , S1TIC : 0092_{16} , S2TIC : 0089_{16}
S3TIC : $008B_{16}$, S4TIC : $008D_{16}$

[ILVL] This bit select interrupt priority level
1～7 : Interrupt priority level is selected
When not using interrupts, set these bits to 0 (disable).

[IR] 0 : When interrupt requested,set this bit to 1.

UARTi receive interrupt control register (i=0～4)



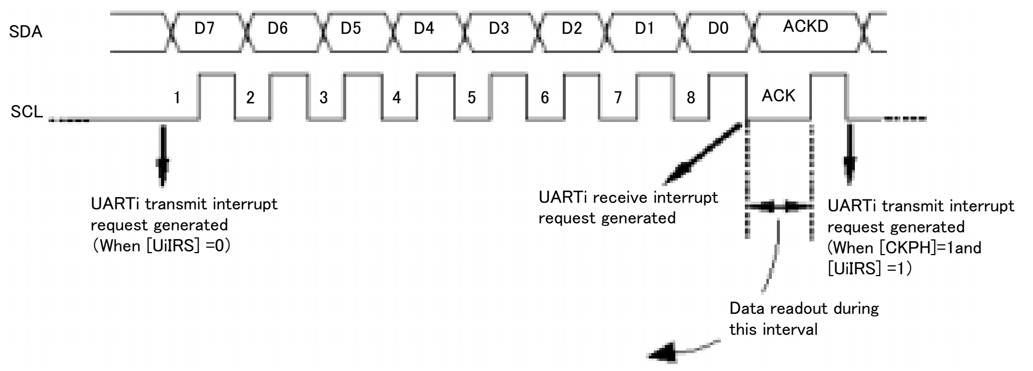
S0RIC : 0072_{16} , S1RIC : 0074_{16} , S2RIC : $006B_{16}$
S3RIC : $006D_{16}$, S4RIC : $006F_{16}$

[ILVL] This bit select interrupt priority level
1～7 : Interrupt priority level is selected
When not using interrupts, set these bits to 0 (disable).

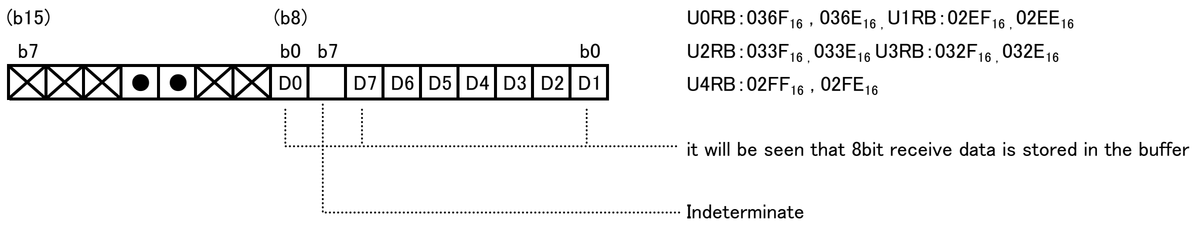
[IR] 0 : When interrupt requested,set this bit to 1.

For interrupts to be generated upon interrupt request, set the I flag to 1.

[Timing Figure]　(When IICM2=1)


SDA
D7
D6
D5
D4
D3
D2
D1
D0
ACKD
SCL
1
2
3
4
5
6
7
8
ACK
UARTi transmit interrupt request generated (When [UiIRS] =0)
UARTi receive interrupt request generated
UARTi transmit interrupt request generated (When [CKPH]=1and [UiIRS] =1)
Data readout during this interval

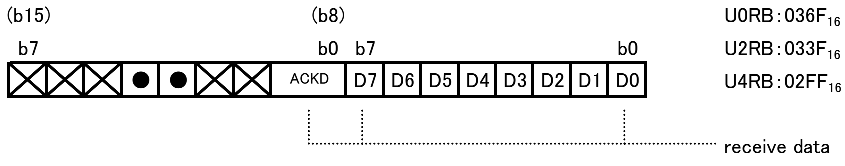
UARTi receive buffer register (i=0～4)


(b15)
(b8)
b7
b0
b7
b0
D0
D7
D6
D5
D4
D3
D2
D1
it will be seen that 8bit receive data is stored in the buffer
Indeterminate

U0RB : $036F_{16}$, $036E_{16}$, U1RB : $02EF_{16}$, $02EE_{16}$
U2RB : $033F_{16}$, $033E_{16}$ U3RB : $032F_{16}$, $032E_{16}$
U4RB : $02FF_{16}$, $02FE_{16}$

If data is read from the receive buffer before the last receive clock pulse goes high after the 8'th receive clock pulse (e.g., during a reception-finished interrupt in simple I^{2}C bus mode), it will be seen that the data is stored in the buffer in order of D0 and D7–D1 as shown above.
Then, when the data is read from the buffer after the last receive clock pulse goes high (e.g., during a transmission-finished interrupt in simple I^{2}C bus mode), the data is read out in order of ACKD and D7–D0.

UARTi receive buffer register (i=0～4) (Afer the last receive clock pulse)



U0RB : $036F_{16}$, $036E_{16}$, U1RB : $02EF_{16}$, $02EE_{16}$
U2RB : $033F_{16}$, $033E_{16}$ U3RB : $032F_{16}$, $032E_{16}$
U4RB : $02FF_{16}$, $02FE_{16}$

2.2 Start and Stop Conditions

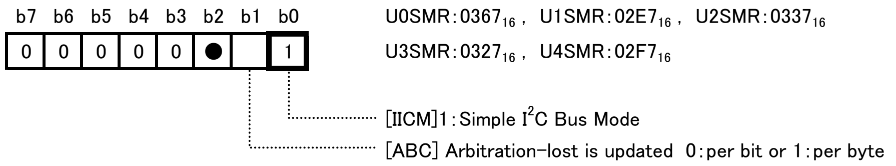
UARTi generates a start condition when it starts sending or receiving data, or a stop condition when it finishes sending or receiving data.
The M32C/83,85 when used as a slave provides the function in hardware to assert an interrupt to detect the start or stop condition generated by the master. When used as the master, the M32C/83,85 provides the function in hardware to generate and send start and stop conditions. This function is called the "Start/stop Condition Detection Interrupt." Furthermore, the M32C/83,85 provides two other functions in hardware, one to detect the bus usage condition to know whether the bus is busy when it generates a start condition, and one to forcibly output a low-level signal from the SCL pin to disable clock outputs from other devices before it starts communication after sending a start condition.

The former is called the "Bus Busy Detection Function," and the latter is called the "SCL Pin Low Output Function 2."

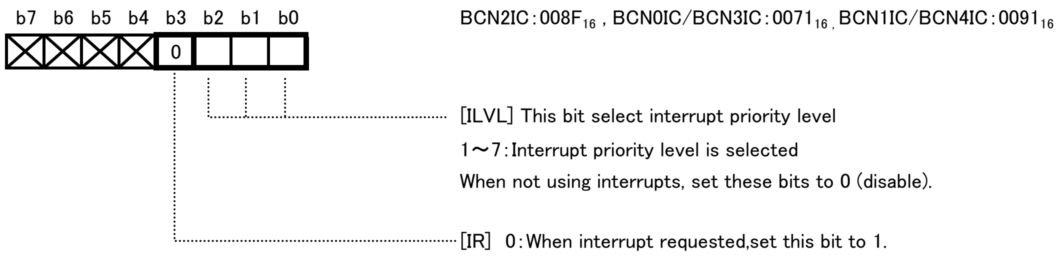
Detecting the start and stop conditions
The start condition is recognized as a high–to–low transition of SDA when SCL is high,
and the stop condition is recognized as a low–to–high transition of SDA when SCL is high.
These conditions can be detected using the M32C/83,85's Start/stop Condition Detection Interrupt.
This interrupt is assigned to the software interrupt numbers 39–41.
When I^2C mode is selected (IICM = 1), the interrupt sources of interrupt numbers 39–41 change to the Start/stop Condition Detection Interrupt. If this interrupt is detected, check the bus busy flag (BBS) to determine which condition, start or stop, has occurred. Note, however, that the start/stop condition detection setup time and hold time in the M32C/83,85 do not always conform to I^2C bus standards. (See the Start/stop Condition Setup and Hold Times in Section 3.1, "Electrical Characteristics.")

[Related Registers]

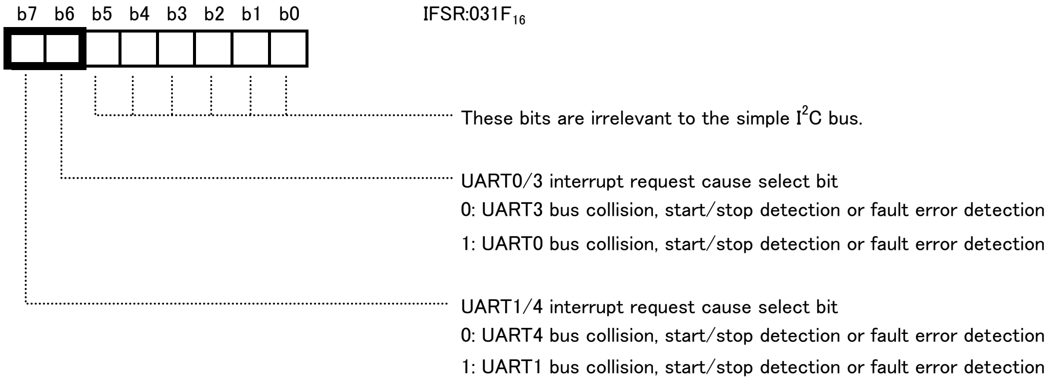
UARTi special mode register (i=0–4)


b7 b6 b5 b4 b3 b2 b1 b0
0 0 0 0 0 ● 1
U0SMR:0367$_{16}$, U1SMR:02E7$_{16}$, U2SMR:0337$_{16}$
U3SMR:0327$_{16}$, U4SMR:02F7$_{16}$
[IICM]1:Simple I^2C Bus Mode
[ABC] Arbitration–lost is updated 0:per bit or 1:per byte

UARTi Bus collision detection interrupt control register (i=0～4)


b7 b6 b5 b4 b3 b2 b1 b0
0
BCN2IC:008F$_{16}$, BCN0IC/BCN3IC:0071$_{16}$, BCN1IC/BCN4IC:0091$_{16}$
[ILVL] This bit select interrupt priority level
1～7:Interrupt priority level is selected
When not using interrupts, set these bits to 0 (disable).
[IR] 0:When interrupt requested,set this bit to 1.

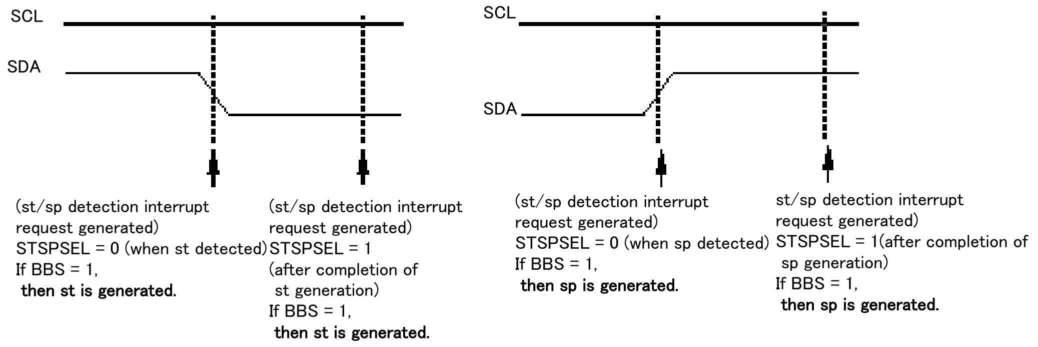
External interrupt request cause select register

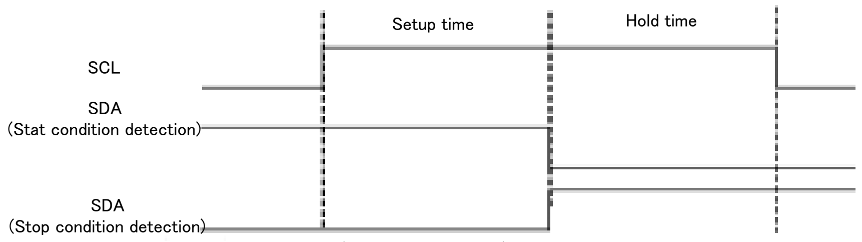

b7 b6 b5 b4 b3 b2 b1 b0
IFSR:031F$_{16}$
These bits are irrelevant to the simple I^2C bus.
UART0/3 interrupt request cause select bit
0: UART3 bus collision, start/stop detection or fault error detection
1: UART0 bus collision, start/stop detection or fault error detection
UART1/4 interrupt request cause select bit
0: UART4 bus collision, start/stop detection or fault error detection
1: UART1 bus collision, start/stop detection or fault error detection

For interrupts to be generated upon interrupt request, set the I flag to 1.

Detecting the start and stop conditions (Continued from the preceding page)
[Timing Figure]

st :Start condition detection、sp :Stop condition detection


SCL
SDA
(st/sp detection interrupt request generated)
STSPSEL = 0 (when st detected)
If BBS = 1,
then st is generated.
(st/sp detection interrupt request generated)
STSPSEL = 1
(after completion of st generation)
If BBS = 1,
then st is generated.
SCL
SDA
(st/sp detection interrupt request generated)
STSPSEL = 0 (when sp detected)
If BBS = 1,
then sp is generated.
st/sp detection interrupt request generated)
STSPSEL = 1(after completion of sp generation)
If BBS = 1,
then sp is generated.



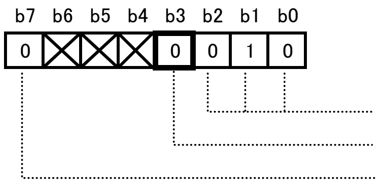
3–6 cycles (setup and hold times)
Cycle number shows main clock input oscillation frequency f(XIN)cycle number.

Sending out the start, stop and restart conditions

When the M32C/83,85 is operating as the master, the start, stop and restart conditions can be generated in hardware.
Set the STAREQ bit to 1 (= start), and a start condition is generated.
Set the STPREQ bit to 1 (= start), and a stop condition is generated (after waiting until SCL is released if SCL is low).
Set the RSTREQ bit to 1 (= start), and a restart condition is generated (after waiting until SCL is released if SCL is low).
Setting the STSPSEL bit to 1 outputs each condition generated above.

[Related Registers]

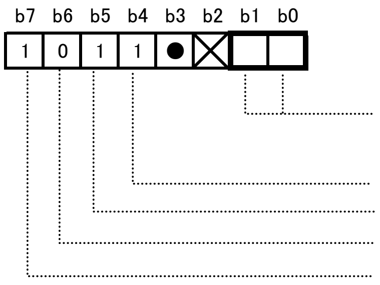
UARTi transmit/receive mode register (i=0–4)



U0MR:0368$_{16}$, U1MR:02E8$_{16}$, U2MR:0338$_{16}$
U3MR:0328$_{16}$, U4MR:02F8$_{16}$

[SMD]010: Simple I^{2}C Bus Mode
[CKDIR] 0: Selected internal clock
[IOPOL]0:No reversed

UARTi transmit/receive control register 0 (i=0–4)

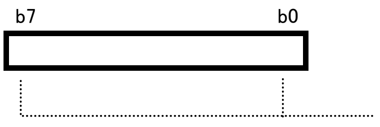


U0C0:036C$_{16}$, U1C0:02EC$_{16}$, U2C0:033C$_{16}$
U3C0:032C$_{16}$, U4C0:02FC$_{16}$

[CLK1] [CLK0] selecte BRG count source
0 0 :f1is selected、0 1 :f8 is selected、1 0 :f2n is selected、1 1 :Must not be set
[CRD] 1 :CTS/RTS function disabled
[NCH] 1 : SCL/SDA pin is N-channel open drain output(Note1)
[CKPOL] 0 :Transmit data is output at falling edge of transfer clock and receive data is input at rising edge
[UFORM] Transfer format is selected 1 :MSB first

Note 1: The UART2 SDA and SCL pins are N-channel open-drain pins.
CMOS output cannot be selected for these pins. When write , To write to bit 5, write 0.

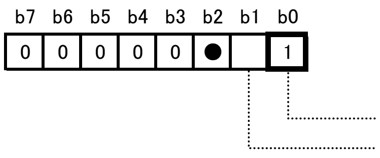
UARTi bit rate generator (i=0–4)



U0BRG:0369$_{16}$, U1BRG:02E9$_{16}$, U2BRG:0339$_{16}$
U3BRG:0329$_{16}$, U4BRG:02F9$_{16}$

BRGi divides the count source by n+1

UARTi special mode register (i=0–4)

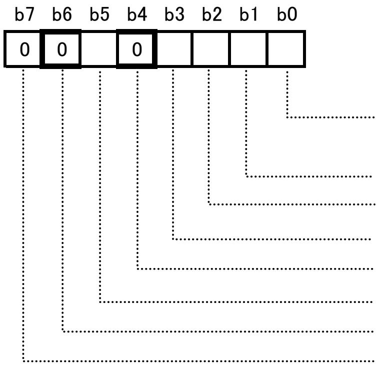


U0SMR:0367$_{16}$, U1SMR:02E7$_{16}$, U2SMR:0337$_{16}$
U3SMR:0327$_{16}$, U4SMR:02F7$_{16}$

[IICM]1: Simple I^{2}C Bus Mode
[ABC] Arbitration-lost is updated 0: per bit or 1: per byte

Sending out the start, stop and restart conditions (Continued from the preceding page)

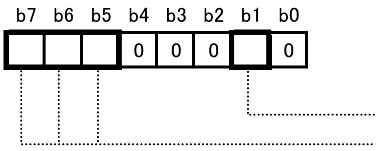
UARTi special mode register 2 (i=0–4)



U0SMR2 : 0366_{16} , U1SMR2 : $02E6_{16}$, U2SMR2 : 0336_{16}
U3SMR2 : 0326_{16} , U4SMR2 : $02F6_{16}$

[IICM2] (For details, see Table 1, "Functions during Simple I^2C Bus Mode (IICM = 1),"
for UARTi Special Mode Register 2 in Section 1.4, "Register Settings during Simple I^2C Bus Mode.")
[CSC] 0 : Clock synchronous is disabled 1 : Clock synchronous is enabled
[SWC] 0 : SCL wait output is disabled 1 : SCL wait output is enabled
[ALS] 0 : SDA output stop is disabled 1 : SDA output stop is enabled
[STAC] 0 : UARTi initialize is disabled
[SWC2] 0 : UARTi clock 1 : SCL output "L"
[SDHI] 0 : SDA output enable
In simple I^2C bus mode, set this bit to 0.

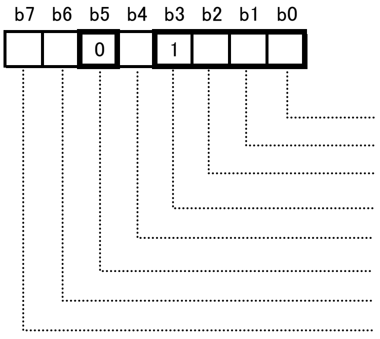
UARTi special mode register 3 (i=0–4)



U0SMR3 : 0365_{16} , U1SMR3 : $02E5_{16}$, U2SMR3 : 0335_{16}
U3SMR3 : 0325_{16} , U4SMR3 : $02F5_{16}$

[CKPH] 0 : Without clock delay 1:With clock delay
[DL2][DL1][DL0] This bit set digital delay time
000:Without delay 001 : 2–cycle of BRG count source
010 : 3–cycle of BRG count source 011 : 4–cycle of BRG count source
100 : 5–cycle of BRG count source 101 : 6–cycle of BRG count source
110 : 7–cycle of BRG count source 111 : 8–cycle of BRG count source

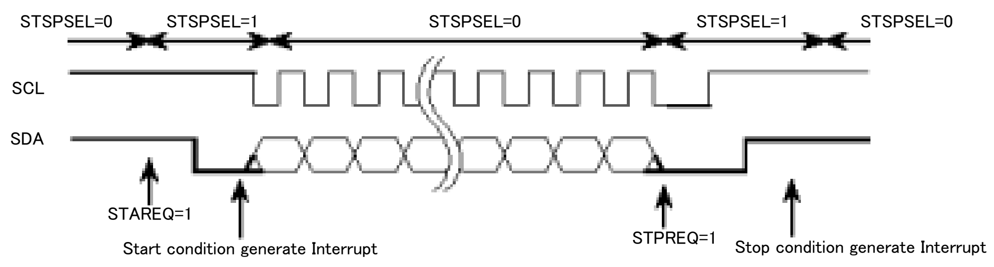
UARTi special mode register 4 (i=0–4)



U0SMR4 : 0364_{16} , U1SMR4 : $02E4_{16}$, U2SMR4 : 0334_{16}
U3SMR4 : 0324_{16} , U4SMR4 : $02F4_{16}$

[STAREQ] 0 : Clear 1 : Start
[RSTAREQ] 0 : Clear 1 : Start
[STPREQ] 0 : Clear 1 : Start
[STSPSEL] 1:Start/stop condition generate block selected
[ACKD] 0 : ACK 1 : NACK
[ACKC] 0:SI/O data output
[SCLHI] 0:Disabled 1:Enabled
[SWC9] 0:SCL "L"hold disabled 1:SCL "L"hold enabled

Sending out the start, stop and restart conditions (Continued from the preceding page)
[Timing Figure]


STSPSEL=0
STSPSEL=1
STSPSEL=0
STSPSEL=1
STSPSEL=0
SCL
SDA
STAREQ=1
Start condition generate Interrupt
STPREQ=1
Stop condition generate Interrupt

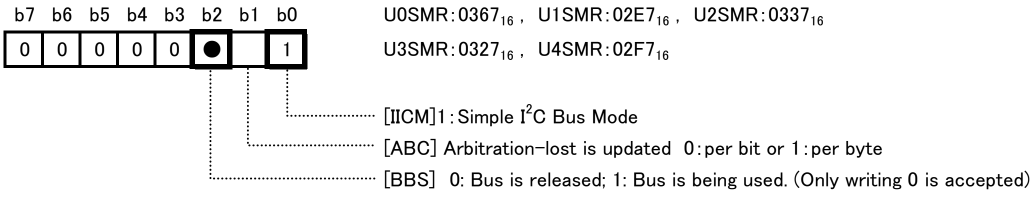
Bus Busy Detection

Before a start condition can be sent out, it is necessary to confirm that the other device has released control of the bus. In simple I^2C bus mode of the M32C/83,85, the bus usage condition can be detected by checking the bus busy flag (BBS).
The BBS flag is set to 1 when a start condition is detected or cleared to 0 when a stop condition is detected.
Therefore, if BBS = 1 when the master attempts to send a start condition,
it must wait until BBS is cleared to 0 before sending a start condition because the bus is being used by the other device.

[Related Registers]

UARTi special mode register (i=0–4)

U0SMR : 0367_{16} , U1SMR : $02E7_{16}$, U2SMR : 0337_{16}
U3SMR : 0327_{16} , U4SMR : $02F7_{16}$


b7 b6 b5 b4 b3 b2 b1 b0
0 0 0 0 0 ● 1
[IICM]1 : Simple I^2C Bus Mode
[ABC] Arbitration–lost is updated 0 : per bit or 1 : per byte
[BBS] 0: Bus is released; 1: Bus is being used. (Only writing 0 is accepted)

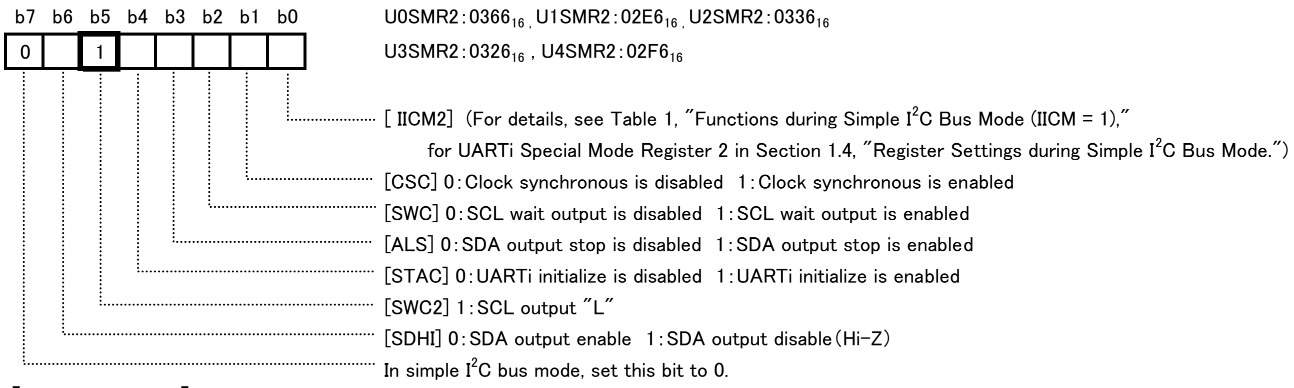
SCL Pin Low Output Function 2

Although the bit displacement condition shown below can be avoided by using a clock delay function (see Section 2.6, "Other Functions"), the explanation here is given assuming a diverted use of the conventional simple I^2C bus firmware that does not have this new function available. The serial I/O of the M32C/83,85 requires 1.5 transfer clock (SCL) cycles at maximum before the transfer clock (SCL in simple I^2C bus mode) is sent out after writing transmit data to the transmit buffer. Furthermore, because the SCL synchronization function of the M32C/83,8 (see Section 2.5, "Arbitrating Contention for Communication") becomes effective after sending the first SCL pulse, if another device sends the first clock pulse before the clock line (SCL) synchronization function becomes effective (see the upper timing diagram), a bit displacement may occur.

For this reason, the M32C/83,85 has a SCL pin low output function to disable clock outputs from other devices after sending a start condition. If this function is used, the transmitting device can start outputting a low-level signal from the SCL pin at the same time it writes data to the transmit buffer, thus keeping other devices in a wait state (see the lower timing diagram). This function is enabled by setting the wait output bit 2 [SWC2] = 1, and is disabled by clearing this bit to 0.

[Related Registers]

UARTi special mode register 2 (i=0-4)

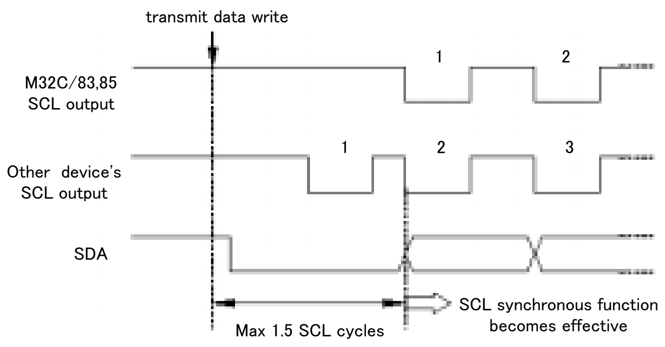


U0SMR2 : 0366_{16}, U1SMR2 : $02E6_{16}$, U2SMR2 : 0336_{16}
U3SMR2 : 0326_{16} , U4SMR2 : $02F6_{16}$

[IICM2] (For details, see Table 1, "Functions during Simple I^2C Bus Mode (IICM = 1),"
for UARTi Special Mode Register 2 in Section 1.4, "Register Settings during Simple I^2C Bus Mode.")
[CSC] 0 : Clock synchronous is disabled 1 : Clock synchronous is enabled
[SWC] 0 : SCL wait output is disabled 1 : SCL wait output is enabled
[ALS] 0 : SDA output stop is disabled 1 : SDA output stop is enabled
[STAC] 0 : UARTi initialize is disabled 1 : UARTi initialize is enabled
[SWC2] 1 : SCL output "L"
[SDHI] 0 : SDA output enable 1 : SDA output disable (Hi-Z)
In simple I^2C bus mode, set this bit to 0.

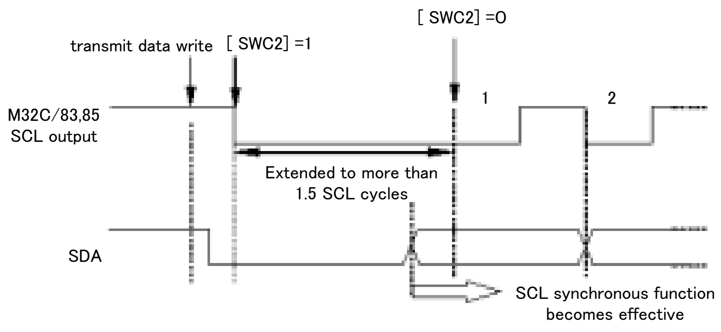
[Timing Figure]

When not using the SCL pin low output function



* Normally, a bit displacement occurs

When using the SCL pin low output function



2.3 Acknowledge

During data transmission/reception, an acknowledge signal (ACK) is attached every byte.
When the M32C/83,85 is operating as a transmitter, the presence of ACK returned from the receiver needs to be detected for each byte transmitted. To this end, the M32C/83,85 has two necessary functions in hardware: Acknowledge Detected Interrupt and Acknowledge Undetected Interrupt.
Also, when the M32C/83,85 is operating as a receiver in one-for-one communication, ACK can easily be generated by setting data '0' in the 9th bit of the transmit data. (For details, see Section 2.1, "Method for Sending and Receiving Byte Data.")
Before the Acknowledge Detected Interrupt and Acknowledge Undetected Interrupt can be used, the I^{2}C mode select bit 2 (IICM2) must be set to 0. This setting makes the interrupt source of interrupt number 17, 19, 33, 35 or 37 and that of interrupt number 18, 20, 34, 36 or 38 usable as the Acknowledge Undetected Interrupt and Acknowledge Detected Interrupt, respectively.
In that case, the timing at which data is transferred from the UARTi receive buffer to the receive buffer register is when the last receive clock pulse goes high.
(For details, see Table 1, "Functions during Simple I^{2}C Bus Mode (IICM = 1), for UARTi Special Mode Register 2 in Section 1.4, Register Settings during Simple I^{2}C Bus Mode.")

Acknowledge Detected

If the SDA line on the transmitter side that has been released (i.e., in the high-impedance state) is found low at the rising edge of the 9th transmit clock pulse, the transmitter can recognize that ACK has been returned from the receiver. For the M32C/83,85, the presence of ACK can be detected using the "Acknowledge Detected Interrupt" function. This interrupt is assigned to the software interrupt number 18, 20, 34, 36 or 38, and the interrupt source of that interrupt number is made the Acknowledge Detected Interrupt only when I^2C mode is selected (IICM = 1) and the I^2C mode select bit 2 (IICM2) is set to 0.

[Related Registers]

UARTi special mode register (i=0-4)

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	0	●		1

U0SMR:0367_{16} , U1SMR:$02E7_{16}$, U2SMR:0337_{16}
U3SMR:0327_{16} , U4SMR:$02F7_{16}$

[IICM]1:Simple I^2C Bus Mode
[ABC] Arbitration-lost is updated 0:per bit or 1:per byte

UARTi special mode register 2 (i=0-4)

b7	b6	b5	b4	b3	b2	b1	b0
0							0

U0SMR2:0366_{16} , U1SMR2:$02E6_{16}$, U2SMR2:0336_{16}
U3SMR2:0326_{16} , U4SMR2:$02F6_{16}$

[IICM2] 0:Acknowledge detection／undetected interrupt is disabled
[CSC] 0:Clock synchronous is disabled 1:Clock synchronous is enabled
[SWC] 0:SCL wait output is disabled 1:SCL wait output is enabled
[ALS] 0:SDA output stop is disabled 1:SDA output stop is enabled
[STAC] 0:UARTi initialize is disabled 1:UARTi initialize is enabled
[SWC2] 0:UARTi clock 1:SCL output "L"
[SDHI] 0:SDA output enable 1:SDA output disable(Hi-Z)
In simple I^2C bus mode, set this bit to 0.

UARTi receive interrupt control register (i=0～4) —When using the Acknowledge Detected Interrupt—

b7	b6	b5	b4	b3	b2	b1	b0
☒	☒	☒	☒	0			

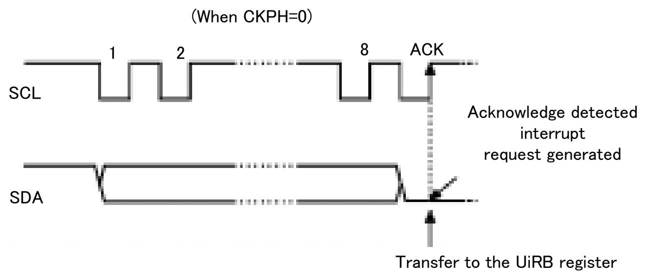
S0RIC:0072_{16} , S1RIC:0074_{16} , S2RIC:$006B_{16}$
S3RIC:$006D_{16}$, S4RIC:$006F_{16}$

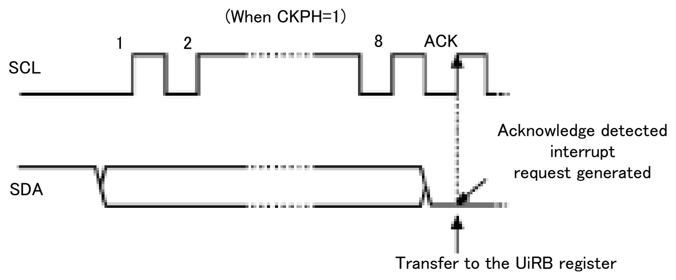
[ILVL] This bit select interrupt priority level
1～7:Interrupt priority level is selected
When not using interrupts, set these bits to 0 (disable).

[IR] 0:When interrupt requested,set this bit to 1.

For the interrupt to be generated upon interrupt request, set the I flag to 1.

[Timing Figure]


(When CKPH=0)
1
2
8
ACK
SCL
SDA
Acknowledge detected interrupt request generated
Transfer to the UiRB register


(When CKPH=1)
1
2
8
ACK
SCL
SDA
Acknowledge detected interrupt request generated
Transfer to the UiRB register

Acknowledge Undetected
If the SDA line on the transmitter side that has been released (i.e., in the high-impedance state) is found high at the rising edge of the 9th transmit clock pulse, the transmitter recognizes that ACK has not been returned from the receiver. For the M32C/83,85, the absence of ACK can be detected using the "Acknowledge Undetected Interrupt" function. This interrupt is assigned to the software interrupt numbers 17, 19, 33, 35 and 37, and the interrupt sources of these interrupt numbers are made the Acknowledge Undetected Interrupt only when I^{2}C mode is selected (IICM = 1) and the I^{2}C mode select bit 2 (IICM2) is set to 0.

[Related Registers]

UARTi special mode register (i=0-4)

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	0	●		1

U0SMR:0367$_{16}$, U1SMR:02E7$_{16}$, U2SMR:0337$_{16}$
U3SMR:0327$_{16}$, U4SMR:02F7$_{16}$

[IICM]1:Simple I^{2}C Bus Mode
[ABC] Arbitration-lost is updated 0:per bit or 1:per byte

UARTi special mode register 2 (i=0-4)

b7	b6	b5	b4	b3	b2	b1	b0
0							0

U0SMR2:0366$_{16}$, U1SMR2:02E6$_{16}$, U2SMR2:0336$_{16}$
U3SMR2:0326$_{16}$, U4SMR2:02F6$_{16}$

[IICM2] 0:Acknowledge detection/undetected interrupt is disabled
[CSC] 0:Clock synchronous is disabled 1:Clock synchronous is enabled
[SWC] 0:SCL wait output is disabled 1:SCL wait output is enabled
[ALS] 0:SDA output stop is disabled 1:SDA output stop is enabled
[STAC] 0:UARTi initialize is disabled 1:UARTi initialize is enabled
[SWC2] 0:UARTi clock 1:SCL output "L"
[SDHI] 0:SDA output enable 1:SDA output disable(Hi-Z)
In simple I^{2}C bus mode, set this bit to 0.

UARTi Transmit interrupt control register (i=0~4) —When using the Acknowledge Undetected Interrupt—

b7	b6	b5	b4	b3	b2	b1	b0
☒	☒	☒	☒	0			

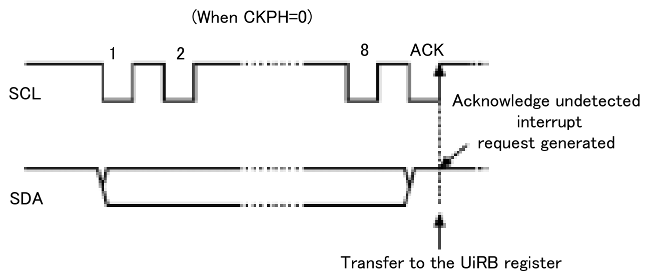
S0TIC:0090$_{16}$, S1TIC:0092$_{16}$, S2TIC:0089$_{16}$
S3TIC:008B$_{16}$, S4TIC:008D$_{16}$

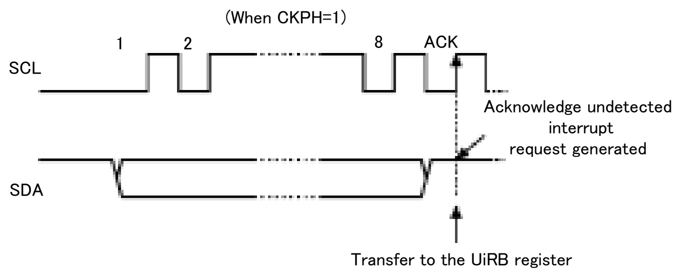
[ILVL] This bit select interrupt priority level
1~7:Interrupt priority level is selected
When not using interrupts, set these bits to 0 (disable).

[IR] 0:When interrupt requested,set this bit to 1.

For the interrupt to be generated upon interrupt request, set the I flag to 1.

[Timing Figure]


(When CKPH=0)
SCL
SDA
1
2
8
ACK
Acknowledge undetected interrupt request generated
Transfer to the UiRB register


(When CKPH=1)
SCL
SDA
1
2
8
ACK
Acknowledge undetected interrupt request generated
Transfer to the UiRB register

2.4 Judgment of the Specified Local Address

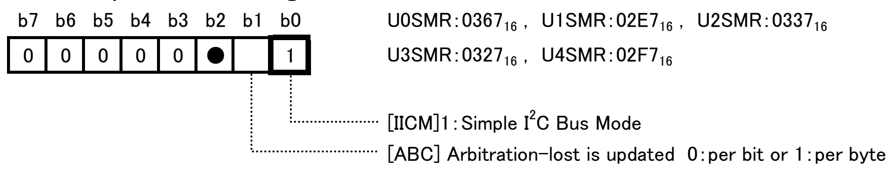
When the M32C/83,85 is operating as a slave, the address sent from the master is compared with the local address and when they match, the slave sends an acknowledge signal to the master. In simple I^2C bus mode of the M32C/83,85, these address comparison and acknowledge transmission are performed in software. However,
because the SCL pin must be held low to keep the master waiting during that time, two necessary functions are provided in hardware: SCL Pin Low Output Function and ACK/NACK Transmit Function.

ACK/NACK Transmit Function

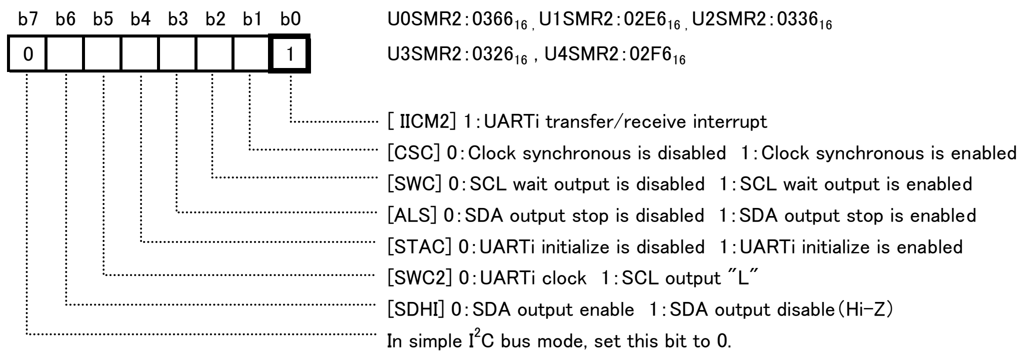
The M32C/83,85 can send ACK or NACK by setting it in the 9th bit of the transmit data, as well as by controlling the ACK data bit (ACKD) after setting the ACK data output enable bit (ACKC) = 1.When the M32C/83,85 is operating as a slave, the address sent from the master is compared with the local address and when they match, the slave sends an acknowledge signal to the master. In simple I^2C bus mode of the M32C/83,85, these address comparison and acknowledge transmission ar performed in software. In that case, acknowledge transmission is accomplished by setting ACKC to 1 which enables ACK or NACK to be sent out.

[Related Registers]

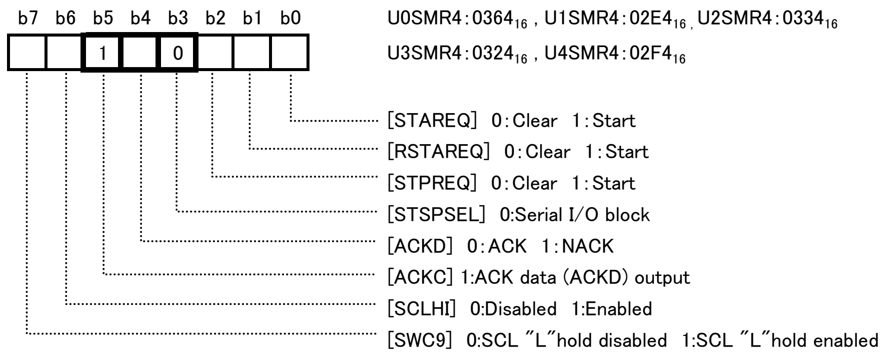
UARTi special mode register (i=0–4)


b7 b6 b5 b4 b3 b2 b1 b0
0 0 0 0 0 ● 1
U0SMR:0367$_{16}$, U1SMR:02E7$_{16}$, U2SMR:0337$_{16}$
U3SMR:0327$_{16}$, U4SMR:02F7$_{16}$
[IICM]1:Simple I^2C Bus Mode
[ABC] Arbitration-lost is updated 0:per bit or 1:per byte

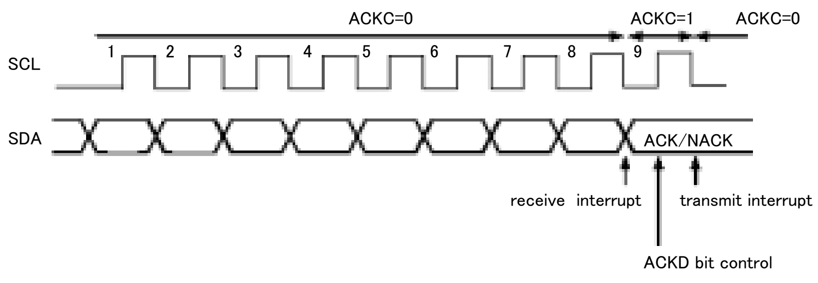
UARTi special mode register 2 (i=0–4)


b7 b6 b5 b4 b3 b2 b1 b0
0 1
U0SMR2:0366$_{16}$, U1SMR2:02E6$_{16}$, U2SMR2:0336$_{16}$
U3SMR2:0326$_{16}$, U4SMR2:02F6$_{16}$
[IICM2] 1:UARTi transfer/receive interrupt
[CSC] 0:Clock synchronous is disabled 1:Clock synchronous is enabled
[SWC] 0:SCL wait output is disabled 1:SCL wait output is enabled
[ALS] 0:SDA output stop is disabled 1:SDA output stop is enabled
[STAC] 0:UARTi initialize is disabled 1:UARTi initialize is enabled
[SWC2] 0:UARTi clock 1:SCL output "L"
[SDHI] 0:SDA output enable 1:SDA output disable(Hi-Z)
In simple I^2C bus mode, set this bit to 0.

UARTi special mode register 4 (i=0–4)


b7 b6 b5 b4 b3 b2 b1 b0
1 0
U0SMR4:0364$_{16}$, U1SMR4:02E4$_{16}$, U2SMR4:0334$_{16}$
U3SMR4:0324$_{16}$, U4SMR4:02F4$_{16}$
[STAREQ] 0:Clear 1:Start
[RSTAREQ] 0:Clear 1:Start
[STPREQ] 0:Clear 1:Start
[STSPSEL] 0:Serial I/O block
[ACKD] 0:ACK 1:NACK
[ACKC] 1:ACK data (ACKD) output
[SCLHI] 0:Disabled 1:Enabled
[SWC9] 0:SCL "L"hold disabled 1:SCL "L"hold enabled

[Timing Figure]



SCL Pin Low Output Function

In the I^2C bus, the specified slave address is sent in the first byte after detecting a start condition (during 7-bit address mode). The slave requires processing to compare the 7 bits of received data in the first byte sent from the master with its local address, as well as to generate (or not to generate) an acknowledge signal synchronously with the 9th clock pulse. The M32C/83,85 has the SCL Pin Low Output Function to accomplish this processing.
This function enables the M32C/83,85 to output a low-level signal from the SCL pin synchronously with the negative transition of the 9th SCL pulse after receiving the first 8 bits of data, thereby forcibly keeping the master waiting.
Then, when the M32C/83,85 has finished address comparison processing in software, it can generate (or not generate) an acknowledge signal by using the ACK/NACK Transmit Function (explained earlier in this section). (When using the M32C/83,85 in one-for-one communication, address reception and acknowledge transmission can be accomplished following the method explained in Section 2.1, "Method for Sending and Receiving Byte Data.")
This function is enabled to work by setting the wait output bit (SWC) to 1, and is disabled by setting SWC to 0.
Also, when the SCL pin is pulled low by this function, it can be returned high by setting SWC to 0. When using this function to perform address comparison processing, be aware that the content of the receive buffer register is read out before the last clock pulse goes high, the received data thus read out has its bit positions changed.
(See the transmit interrupt/receive interrupt timing diagrams in Section 1.1.)

[Related Registers]

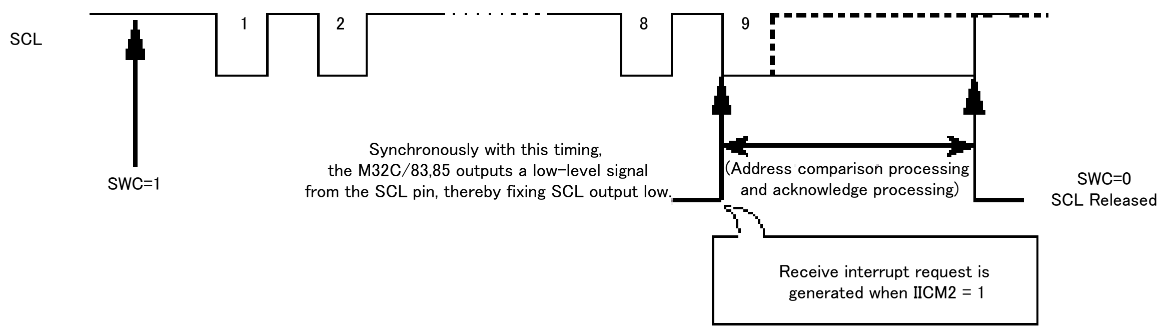
UARTi special mode register 2 (i=0–4)

U0SMR2 : 0366_{16} , U1SMR2 : $02E6_{16}$, U2SMR2 : 0336_{16}
U3SMR2 : 0326_{16} , U4SMR2 : $02F6_{16}$

b7	b6	b5	b4	b3	b2	b1	b0
0					1		1

- b0 [IICM2] 1 : UARTi transfer/receive interrupt
- b1 [CSC] 0 : Clock synchronous is disabled 1 : Clock synchronous is enabled
- b2 [SWC] 1 : SCL wait output is enabled
- b3 [ALS] 0 : SDA output stop is disabled 1 : SDA output stop is enabled
- b4 [STAC] 0 : UARTi initialize is disabled 1 : UARTi initialize is enabled
- b5 [SWC2] 0 : UARTi clock 1 : SCL output "L"
- b6 [SDHI] 0 : SDA output enable 1 : SDA output disable (Hi-Z)
- b7 In simple I^2C bus mode, set this bit to 0.

[Timing Figure]

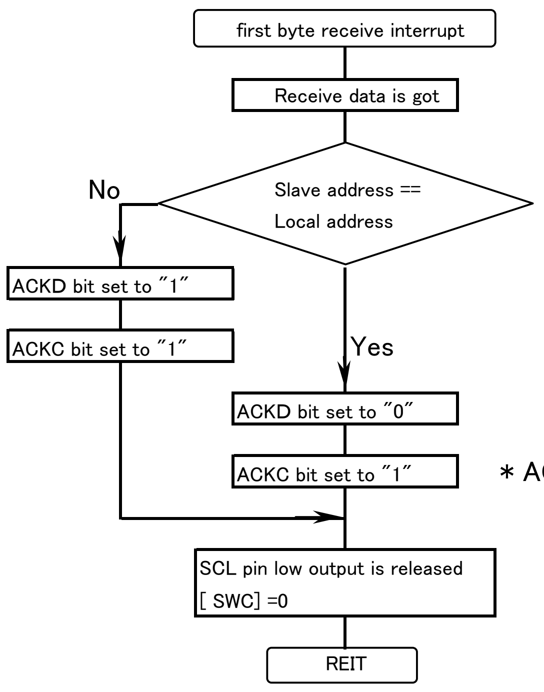

SCL
1
2
8
9
SWC=1
Synchronously with this timing, the M32C/83,85 outputs a low-level signal from the SCL pin, thereby fixing SCL output low.
(Address comparison processing and acknowledge processing)
SWC=0
SCL Released
Receive interrupt request is generated when IICM2 = 1

Example of Local Address Judgment

The slave address is specified in one of two formats available: 7–bit address and 10–bit address.
The following explains how the received slave address is determined and responded to by using the 7–bit address format as an example. The same applies to the 10–bit address format.
In the example below, note that a first byte receive interrupt is generated after receiving a start bit.
Note also that SWC is assumed to have been set to 1 (SCL Pin Low Output Function enabled) before receiving the first byte, and that only part of the interrupt handling routine is shown.

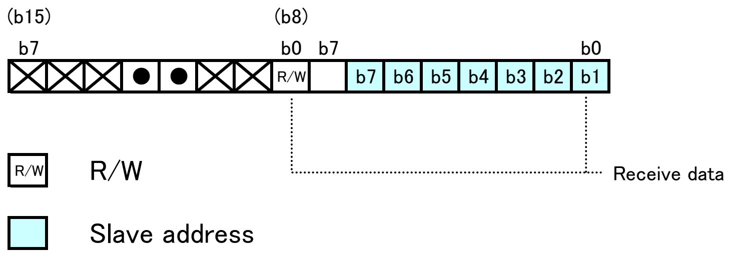
[Flow chart]

* Settings in the main routine (Example)
・SWC = 1 (SCL Pin Low Output Function is enabled)
・IICM2 = 1 (Interrupt source of interrupt number 18, 20, 34, 36 or 38 is used as the receive interrupt, which is generated on the positive transition of the last clock pulse)
・S2RIC = 7 (Receive interrupt is enabled)
・I flag = 1 (Interrupts enabled)

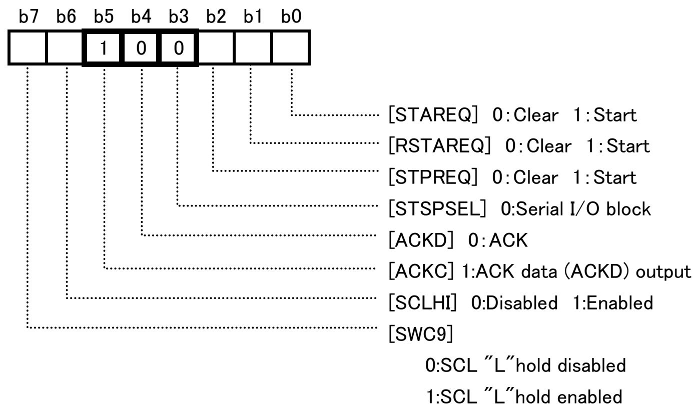


* ACK is sent out.

UARTi receive buffer register (i=0～4)
U0RB:036F$_{16}$, 036E$_{16}$, U1RB:02EF$_{16}$, 02EE$_{16}$
U2RB:033F$_{16}$, 033E$_{16}$ U3RB:032F$_{16}$, 032E$_{16}$
U4RB:02FF$_{16}$, 02FE$_{16}$



UARTi special mode register 4 (i=0–4)
U0SMR4:0364$_{16}$, U1SMR4:02E4$_{16}$, U2SMR4:0334$_{16}$
U3SMR4:0324$_{16}$, U4SMR4:02F4$_{16}$



2.5 Arbitrating Contention for Communication

When using the I^2C bus in a multi-master environment, it is possible that two or more masters generate a start condition attempting to start data transmission at the same time (giving rise to the need to arbitrate contention). In the I^2C bus system, contention for communication between multiple masters is resolved by arbitration.
Simple I^2C bus mode of the M32C/83,85 provides two necessary functions in hardware to recover communication in case of arbitration-lost. These functions are called the "Arbitration-lost Detection Function" and the " SDA Output Disable Function in Case of Arbitration-lost." In addition to recovery from arbitration-lost, simple I^2C bus mode has the "SCL Synchronizing Function" as a means of arbitrating contention for communication based on clock synchronization.

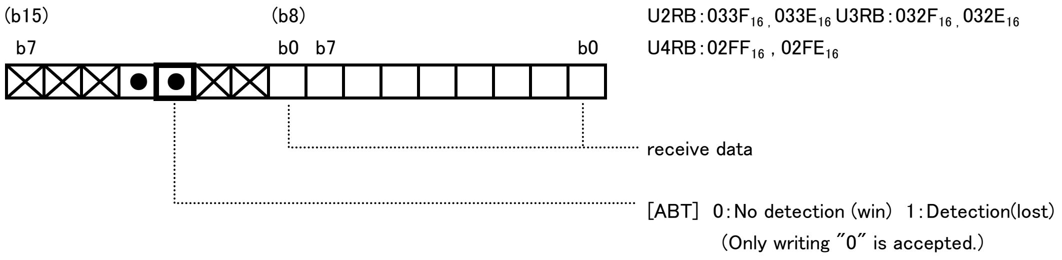
Arbitration–lost Detection Function (i = 0–4)

Simple I^2C bus mode of the M32C/83,85 has an arbitration–lost detection flag [ABT]. This flag is assigned to the UARTi Receive Buffer Register bit 3. If unmatching of the internal data level and the SDA level is detected on the positive transition of SCL, the arbitration–lost detection flag [ABT] is set to 1. The arbitration–lost detection flag control bit [ABC] may be used to choose whether the arbitration–lost detection flag is to be updated every bit (= 0) or updated every byte (= 1). The ABC bit must be fixed to 0 when both I^2C mode select bit (IICM) and I^2C mode select bit 2 (IICM2) = 1. The SDA output is high and SDA input is low at the time an acknowledge signal is received, causing the arbitration–lost detection flag to be set. Therefore, the arbitration–lost detection flag must be cleared to 0 before transmission can start.

[Related Registers]

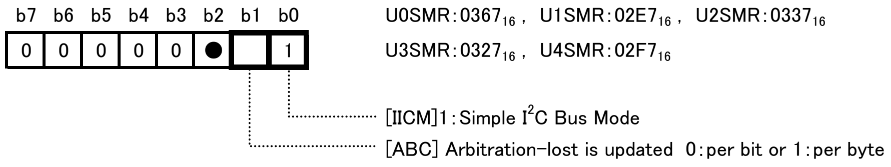
UARTi receive buffer register (i=0～4)

U0RB : $036F_{16}$, $036E_{16}$, U1RB : $02EF_{16}$, $02EE_{16}$
U2RB : $033F_{16}$, $033E_{16}$ U3RB : $032F_{16}$, $032E_{16}$
U4RB : $02FF_{16}$, $02FE_{16}$


(b15)
(b8)
b7
b0
b7
b0
receive data
[ABT] 0 : No detection (win) 1 : Detection(lost)
(Only writing "0" is accepted.)

UARTi special mode register (i=0–4)

U0SMR : 0367_{16} , U1SMR : $02E7_{16}$, U2SMR : 0337_{16}
U3SMR : 0327_{16} , U4SMR : $02F7_{16}$


b7 b6 b5 b4 b3 b2 b1 b0
0 0 0 0 0 ● 1
[IICM]1 : Simple I^2C Bus Mode
[ABC] Arbitration–lost is updated 0 : per bit or 1 : per byte

SDA Output Disable Function in Case of Arbitration-lost

If the master detects occurrence of an arbitration-lost condition, it must turn the SDA output off at that point in time. Simple I^2C bus mode of the M32C/83,85 allows to select the function to automatically turn the SDA output off in hardware when an arbitration-lost condition occurs. This function is enabled by setting the SDA output stop bit [ALS] to 1, and is disabled by setting it to 0. If the SDA output is turned off by this function, it can be turned back on again by clearing the SDA output stop bit [ALS] or the arbitration-lost detection flag [ABT]. Note that while this function is enabled, an arbitration-lost condition is assumed to have occurred when receiving an acknowledge signal and the SDA output is turned off. Therefore, clear the arbitration-lost detection flag [ABT] to 0 before sending the next byte data. Also make sure the arbitration-lost detection flag control bit [ABC] is fixed to 0.

[Related Registers]

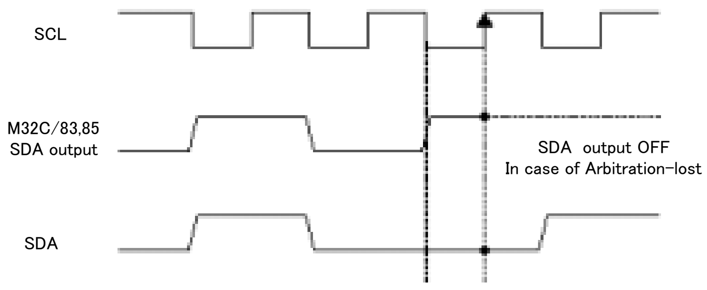
UARTi special mode register 2 (i=0–4)

b7	b6	b5	b4	b3	b2	b1	b0
0				1			

U0SMR2 : 0366_{16}, U1SMR2 : $02E6_{16}$, U2SMR2 : 0336_{16}
U3SMR2 : 0326_{16}, U4SMR2 : $02F6_{16}$

[IICM2] (For details, see Table 1, "Functions during Simple I^2C Bus Mode (IICM = 1)," for UARTi Special Mode Register 2 in Section 1.4, "Register Settings during Simple I^2C Bus Mode.")
[CSC] 0 : Clock synchronous is disabled 1 : Clock synchronous is enabled
[SWC] 0 : SCL wait output is disabled 1 : SCL wait output is enabled
[ALS] 1 : SDA output stop is enabled
[STAC] 0 : UARTi initialize is disabled 1 : UARTi initialize is enabled
[SWC2] 0 : UARTi clock 1 : SCL output "L"
[SDHI] 0 : SDA output enable 1 : SDA output disable (Hi-Z)
In simple I^2C bus mode, set this bit to 0.

[Timing Figure]

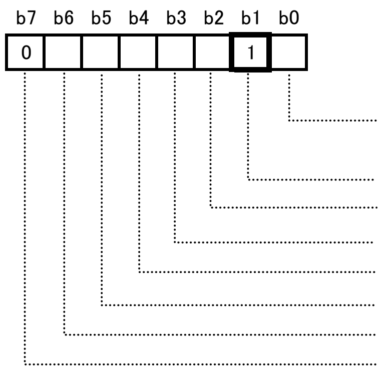


SCL Synchronizing Function

If the M32C/83,85 is connected to a device whose processing speed is slow, a situation may occur that some other device pulls the SCL line low to forcibly keep the clock sent from the master waiting. Simple I^2C bus mode of the M32C/83,85 has the function called the "SCL Synchronizing Function" which automatically places the M32C/83,85 into a wait state when its SCL line is pulled low by other devices, and when the SCL line is returned high, places it out of the wait state. This function is enabled to work by setting the clock synchronization bit [CSC] to 1, and is disabled by setting it to 0.
This function can only be used when using the M32C/83,85 as the master (internal clock mode).

[Related Registers]

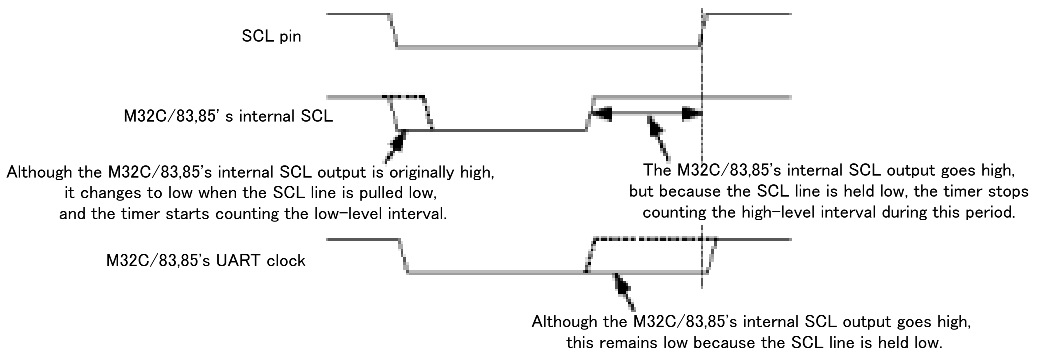
UARTi special mode register 2 (i=0–4)

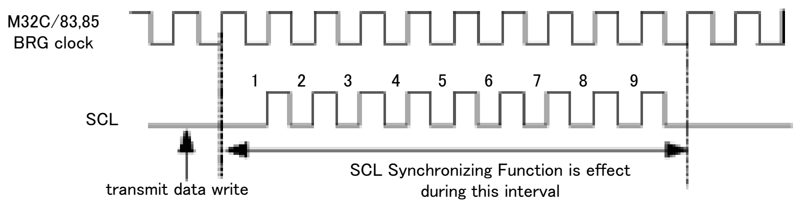


U0SMR2 : 0366_{16}, U1SMR2 : $02E6_{16}$, U2SMR2 : 0336_{16}
U3SMR2 : 0326_{16}, U4SMR2 : $02F6_{16}$

[IICM2] (For details, see Table 1, "Functions during Simple I^2C Bus Mode (IICM = 1)," for UARTi Special Mode Register 2 in Section 1.4, "Register Settings during Simple I^2C Bus Mode.")
[CSC] 1 : Clock synchronous is enabled
[SWC] 0 : SCL wait output is disabled 1 : SCL wait output is enabled
[ALS] 0 : SDA output stop is disabled 1 : SDA output stop is enabled
[STAC] 0 : UARTi initialize is disabled 1 : UARTi initialize is enabled
[SWC2] 0 : UARTi clock 1 : SCL output "L"
[SDHI] 0 : SDA output enable 1 : SDA output disable (Hi-Z)
In simple I^2C bus mode, set this bit to 0.

[Timing Figure]


SCL pin
M32C/83,85' s internal SCL
Although the M32C/83,85's internal SCL output is originally high, it changes to low when the SCL line is pulled low, and the timer starts counting the low-level interval.
The M32C/83,85's internal SCL output goes high, but because the SCL line is held low, the timer stops counting the high-level interval during this period.
M32C/83,85's UART clock
Although the M32C/83,85's internal SCL output goes high, this remains low because the SCL line is held low.



2.6 Other Functions

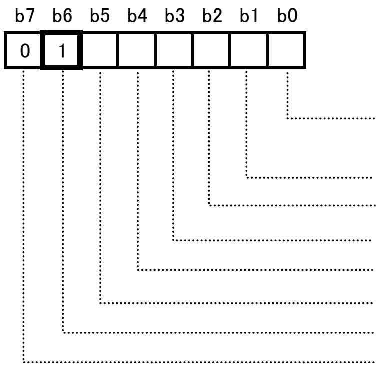
In addition to the functions described in Sections 2.1 to 2.5, the simple I^2C bus mode of the M32C/83,85 has the following function in hardware that facilitates I^2C bus control.

SDA Output Disable Function

When the M32C/83,85 is operating as a slave, if the address specified by the master and the local address are found not matching by address determination in the first byte after receiving a start condition, the M32C/83,85 must turn the SDA output off(placed in the high-impedance state). To turn the SDA output off in such a case, set data '1FFh' in the M32C/83,85's transmit buffer register every 9th SCL pulse (every time a receive interrupt request is generated).
Or the M32C/83,85's SDA Output Disable Function may be used to turn the SDA output off. This function is enabled by setting the SDA output disable bit [SDHI] to 1, in which case the M32C/83,85's SDA output can be placed in the high-impedance state without having to set data '1FFh' in the transmit buffer register. This function is disabled by setting the SDA output disable bit [SDHI] to 0, in which case the value set in the transmit buffer is output synchronously with the next SCL input.

[Related Registers]

UARTi special mode register 2 (i=0-4)



U0SMR2 : 0366_{16} , U1SMR2 : $02E6_{16}$, U2SMR2 : 0336_{16}
U3SMR2 : 0326_{16} , U4SMR2 : $02F6_{16}$

[IICM2] (For details, see Table 1, "Functions during Simple I^2C Bus Mode (IICM = 1),"
for UARTi Special Mode Register 2 in Section 1.4, "Register Settings during Simple I^2C Bus Mode.")
[CSC] 0 : Clock synchronous is dsiabled 1 : Clock synchronous is enabled
[SWC] 0 : SCL wait output is disabled 1 : SCL wait output is enabled
[ALS] 0 : SDA output stop is disabled 1 : SDA output stop is enabled
[STAC] 0 : UARTi initialize is disabled 1 : UARTi initialize is enabled
[SWC2] 0 : UARTi clock 1 : SCL output "L"
[SDHI] 1 : SDA output disable (Hi-Z)
In simple I^2C bus mode, set this bit to 0.

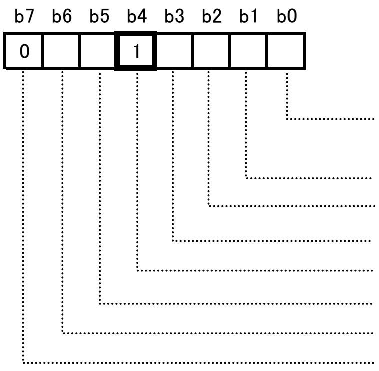
UARTi Initialization Function (i = 0–4)

Simple I^{2}C bus mode of the M32C/83,85 has the function to automatically initialize UARTi synchronously with the timing at which a start condition is detected. This function is used when the M32C/83,85 is operating as a slave.
This function is enabled by setting the UARTi initialization bit [STAC] to 1, and is disabled by setting it to 0.
When a start bit is detected, UARTi is initialized in the manner described below.
(1) The transmit register is initialized, and the content of the transmit buffer register is transferred to the transmit register. This eliminates the need to set data in the transmit buffer register newly again when receiving data, and UARTi starts sending data synchronously with the next clock pulse supplied. However, because the transmit data here is the same data that was being transmitted last, the SDA output disable bit [SDHI] must be set to 1 in order to disable the transmit data from being output.
(2) The receive register is initialized, and UARTi starts receiving data synchronously with the next clock pulse supplied. No overrun error occurs at this time, because the receive register is initialized before reading data out of the receive buffer register.
(3) The wait output bit [SWC] is set to 1. The SLC pin low output function is thereby enabled, and a low-level signal is output from the SCL pin on the negative transition of the 9th transfer clock pulse.
This function can only be used when an external clock is selected. Note also that if this function is enabled when UARTi starts sending or receiving, the transmit buffer empty flag does not change state.

[Related Registers]

UARTi special mode register 2 (i=0–4)

U0SMR2 : 0366$_{16}$, U1SMR2 : 02E6$_{16}$, U2SMR2 : 0336$_{16}$
U3SMR2 : 0326$_{16}$, U4SMR2 : 02F6$_{16}$



[IICM2] (For details, see Table 1, "Functions during Simple I^{2}C Bus Mode (IICM = 1)," for UARTi Special Mode Register 2 in Section 1.4, "Register Settings during Simple I^{2}C Bus Mode.")
[CSC] 0 : Clock synchronous is disabled 1 : Clock synchronous is enabled
[SWC] 0 : SCL wait output is disabled 1 : SCL wait output is enabled
[ALS] 0 : SDA output stop is disabled 1 : SDA output stop is enabled
[STAC] 0 : UARTi initialize is disabled 1 : UARTi initialize is enabled
[SWC2] 0 : UARTi clock 1 : SCL output "L"
[SDHI] 0 : SDA output enable 1 : SDA output disable (Hi-Z)
In simple I^{2}C bus mode, set this bit to 0.

SCL Output Stop Function

When using the I^2C bus in a multi–master environment, it is possible that while the M32C/83,85 is operating as the master sending or receiving data, other masters will generate a stop condition. In such a case, the M32C/83,85 must release SCL and SDA to terminate communication. The SDA Output Disable Function in Case of Arbitration–lost (see Section 2.5, "Arbitrating Contention for Communication") may be used to release SDA. The SCL Output Stop Function described here may be used to release SCL. This function is enabled by setting SCLHI to 1.

[Related Registers]

UARTi special mode register (i=0–4)

U0SMR:0367_{16} , U1SMR:$02E7_{16}$, U2SMR:0337_{16}
U3SMR:0327_{16} , U4SMR:$02F7_{16}$

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	0	●		1

[IICM]1 : Simple I^2C Bus Mode
[ABC] Arbitration–lost is updated 0 : per bit or 1 : per byte
[BBS] 0: Bus is released; 1: Bus is being used. (Only writing 0 is accepted)

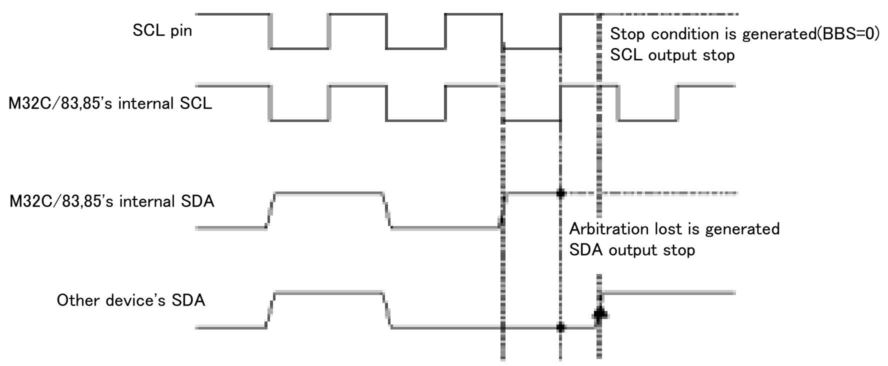
UARTi special mode register 4 (i=0–4)

U0SMR4:0364_{16} , U1SMR4:$02E4_{16}$, U2SMR4:0334_{16}
U3SMR4:0324_{16} , U4SMR4:$02F4_{16}$

b7	b6	b5	b4	b3	b2	b1	b0
	1						

[STAREQ] 0 : Clear 1 : Start
[RSTAREQ] 0 : Clear 1 : Start
[STPREQ] 0 : Clear 1 : Start
[STSPSEL] 0:Serial I/O block 1:Start/stop condition generate block selected
[ACKD] 0 : ACK 1 : NACK
[ACKC] 0:SI/O data output 1:ACK data (ACKD) output
[SCLHI] 1:Enabled
[SWC9] 0:SCL "L"hold disabled 1:SCL "L"hold enabled

[Timing Figure]

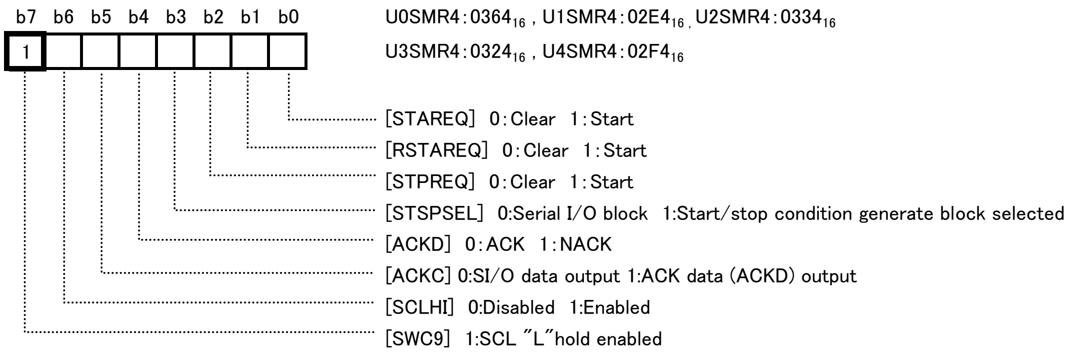

SCL pin
M32C/83,85's internal SCL
M32C/83,85's internal SDA
Other device's SDA
Stop condition is generated(BBS=0)
SCL output stop
Arbitration lost is generated
SDA output stop

SCL Pin Low Output Function 2

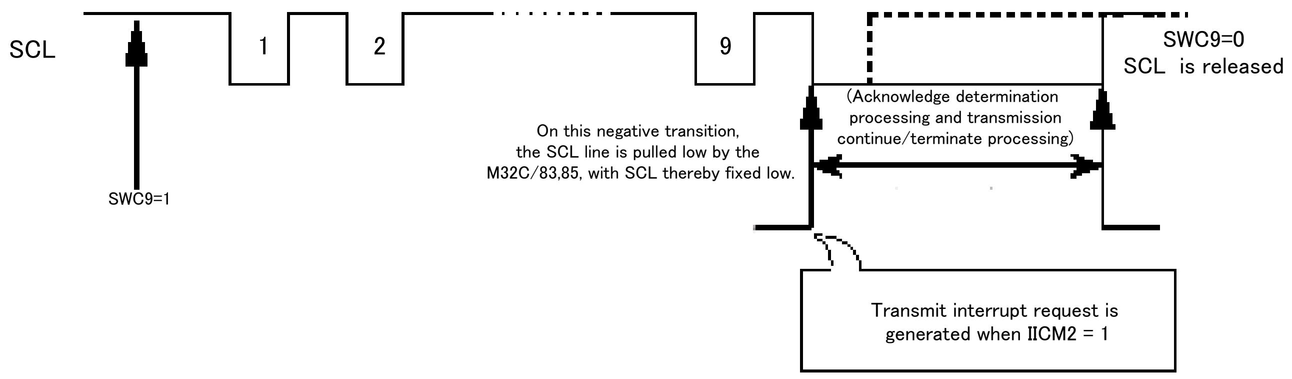
When performing slave transmission in the I^{2}C bus, the master generates (or not generate) an acknowledge signal synchronously with the 9th clock pulse. At this time, the slave checks for acknowledge and if an acknowledge signal is detected, continues to send (by setting the next transmit data). If an acknowledge signal is not detected, the slave terminates transmission. As a function to perform this processing, the M32C/83,85 has SCL Pin Low Output Function 2. This function enables the M32C/83,85 to output a low-level signal from the SCL pin synchronously with the negative transition of the 9th SCL pulse after receiving the first 9 bits of data (ACK/NACK), thereby forcibly keeping the master waiting. Then, when the M32C/83,85 has finished acknowledge determination processing in software, it can continue to send or terminate transmission as necessary.
This function is enabled to work by setting the wait output bit 2 [SWC9] to 1, and is disabled by setting it to 0.
If the SCL pin is pulled low (= 0) by this function, it can be returned high by setting SWC9 to 0.

[Related Registers]

UARTi special mode register 4 (i=0–4)


b7 b6 b5 b4 b3 b2 b1 b0
1
U0SMR4:0364$_{16}$, U1SMR4:02E4$_{16}$, U2SMR4:0334$_{16}$
U3SMR4:0324$_{16}$, U4SMR4:02F4$_{16}$
[STAREQ] 0:Clear 1:Start
[RSTAREQ] 0:Clear 1:Start
[STPREQ] 0:Clear 1:Start
[STSPSEL] 0:Serial I/O block 1:Start/stop condition generate block selected
[ACKD] 0:ACK 1:NACK
[ACKC] 0:SI/O data output 1:ACK data (ACKD) output
[SCLHI] 0:Disabled 1:Enabled
[SWC9] 1:SCL "L"hold enabled

[Timing Figure]

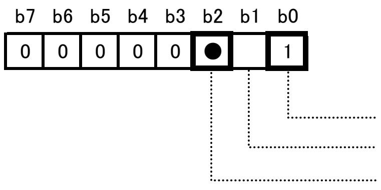

SCL
1
2
9
SWC9=1
On this negative transition, the SCL line is pulled low by the M32C/83,85, with SCL thereby fixed low.
(Acknowledge determination processing and transmission continue/terminate processing)
SWC9=0
SCL is released
Transmit interrupt request is generated when IICM2 = 1

Clock Delay Function

When using the M32C/83,85 in the I^{2}C bus, UARTi (i = 0–4) may be used by making most of a clock delay function. This function makes it possible to output a clock waveform in which SCL is low at the beginning and again low at end, with the result that start and stop conditions are connected smoothly. Furthermore, because a write to the UiRB register occurs on the negative transition of the 8th SCL pulse and the positive transition of the 9th SCL pulse, it is made possible to receive the ACK/NACK bit. What's more, if IICM2 = 1, a transmit interrupt can be generated after receiving the ACK/NACK bit. This function is enabled by setting IICM to 1 and then CKPH to 1.

[Related Registers]

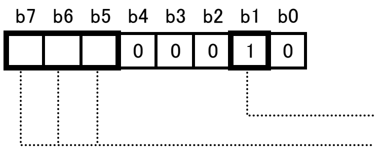
UARTi special mode register (i=0–4)



U0SMR:0367$_{16}$, U1SMR:02E7$_{16}$, U2SMR:0337$_{16}$
U3SMR:0327$_{16}$, U4SMR:02F7$_{16}$

[IICM]1:Simple I^{2}C Bus Mode
[ABC] Arbitration–lost is updated 0:per bit or 1:per byte
[BBS] 0: Bus is released; 1: Bus is being used. (Only writing 0 is accepted)

UARTi special mode register 3 (i=0–4)

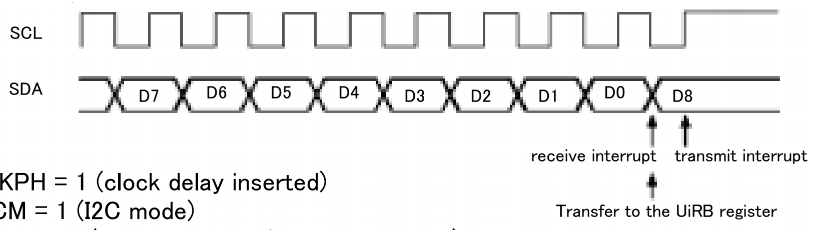


U0SMR3:0365$_{16}$, U1SMR3:02E5$_{16}$, U2SMR3:0335$_{16}$
U3SMR3:0325$_{16}$, U4SMR3:02F5$_{16}$

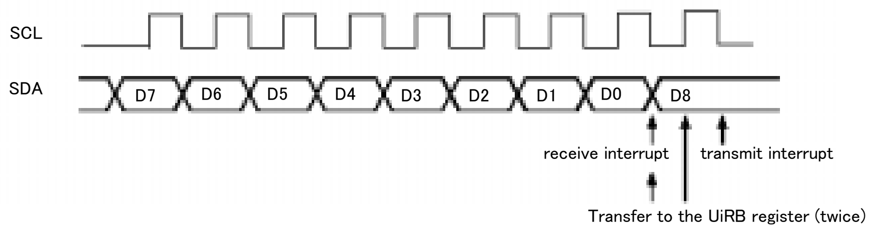
[CKPH] 1:With clock delay
[DL2][DL1][DL0] This bit set digital delay time
000:Without delay 001:2–cycle of BRG count source
010:3–cycle of BRG count source 011:4–cycle of BRG count source
100:5–cycle of BRG count source 101:6–cycle of BRG count source
110:7–cycle of BRG count source 111:8–cycle of BRG count source

[Timing Figure]

・CKPH = 0 (no clock delay)
IICM = 1 (I2C mode)
IICM2 = 1 (UART transmit/receive



・CKPH = 1 (clock delay inserted)
IICM = 1 (I2C mode)
IICM2 = 1 (UART transmit/receive interrupt)



UART Output Digital Delay Function

When performing transmission in the I^2C bus, the SDA output data must be changed over while SCL remains low.
If SDA changes state while SCL is high, a start or stop condition may be detected erratically.
The M32C/83,85 uses the UART Output Digital Delay Function to ensure that the SDA output data will be changed while SCL is low.
This function is enabled by setting IICM to 1 and then DL0-2 to any value between '1' to '7.'

[Related Registers]

UARTi special mode register (i=0-4)

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	0	●		1

U0SMR:0367_{16} , U1SMR:$02E7_{16}$, U2SMR:0337_{16}
U3SMR:0327_{16} , U4SMR:$02F7_{16}$

[IICM]1:Simple I^2C Bus Mode
[ABC] Arbitration-lost is updated 0:per bit or 1:per byte
[BBS] 0: Bus is released; 1: Bus is being used. (Only writing 0 is accepted)

UARTi special mode register 3 (i=0-4)

b7	b6	b5	b4	b3	b2	b1	b0
			0	0	0	1	0

U0SMR3:0365_{16} , U1SMR3:$02E5_{16}$, U2SMR3:0335_{16}
U3SMR3:0325_{16} , U4SMR3:$02F5_{16}$

[CKPH] 1:With clock delay
[DL2][DL1][DL0] This bit set digital delay time
000:Without delay 001:2-cycle of BRG count source
010:3-cycle of BRG count source 011:4-cycle of BRG count source
100:5-cycle of BRG count source 101:6-cycle of BRG count source
110:7-cycle of BRG count source 111:8-cycle of BRG count source

UARTi transmit/receive control register 0 (i=0-4)

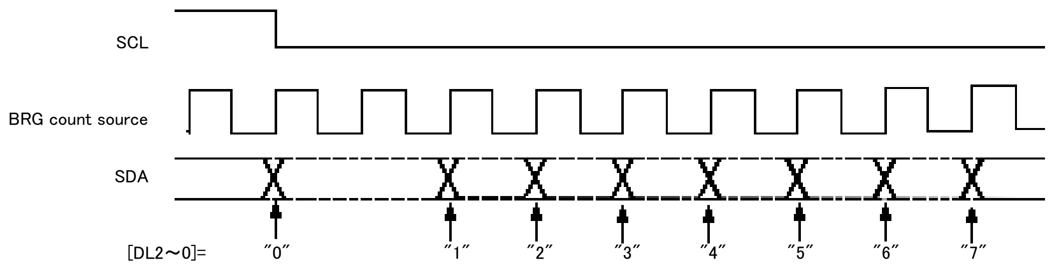
b7	b6	b5	b4	b3	b2	b1	b0
1	0	1	1	●	☒		

U0C0:$036C_{16}$, U1C0:$02EC_{16}$, U2C0:$033C_{16}$
U3C0:$032C_{16}$, U4C0:$02FC_{16}$

[CLK1] [CLK0] selecte BRG count source
0 0 :f1is selected、0 1 :f8 is selected、1 0 :f2n is selected、1 1 :Must not be set
[CRD] 1 :CTS/RTS function disabled
[NCH] 1:SCL/SDA pin is N-channel open drain output(Note1)
[CKPOL] 0 :Transmit data is output at falling edge of transfer clock and receive data is input at rising edge
[UFORM] Transfer format select :MSB first

Note 1: The UART2 SDA and SCL pins are N-channel open-drain pins.
CMOS output cannot be selected for these pins. When write , To write to bit 5, write 0.

[Timing Figure]



SDA output changeover timing when DL2-0 are set to any value between "0" to"7"

Chapter 3
Precautions on Simple I^2C Bus Mode

3.1 Electrical Characteristics

This chapter describes the precautions and limitations to be observed when using simple I^2C bus mode of the M32C/83,85 to control the I^2C bus protocol.

3.1 Electrical Characteristics

The electrical characteristics of the M32C/83,85 do not all conform to and partly differ from I^{2}C bus standard
The I^{2}C bus standards are listed in the table below.

Parameter	Symbol	Standard mode		High-speed mode		Unit
		Min.	Max.	Min.	Max.	
Low Level Input Voltage: When the input level is constant When the input level changes with VDD	VIL	 -0.5 -0.5	 1.5 0.3VDD	 -0.5 -0.5	 1.5 0.3VDD	V
High Level Input Voltage: When the input level is constant When the input level changes with VDD	VIH	 3.0 0.7VDD	 *1) *1)	 3.0 0.7VDD	 *1) *1)	V
Schmitt Trigger Input Hysteresis: When the input level is constant When the input level changes with VDD	Vhys	 n/a n/a	 n/a n/a	 0.2 0.05VDD	 — —	V
Pulse width of spikes suppressed by an input filter	t_{SP}	n/a	n/a	0	50	ns
Low Level Output Voltage (Open-drain or open-collector): When sink current = 3 mA When sink current = 6 mA	 VOL1 VOL2	 0 n/a	 0.4 n/a	 0 0	 0.4 0.6	V
Output fall time from VIH min. to VIL max. when buscapacitance = 10 pF to 400 pF (up to 6 mA through VOL2parallel resistance): When maximum sink current at VOL1 = 3 mA When maximum sink current at VOL2 = 6 mA	t_{OF}	 — n/a	 250 2) n/a	 20+0.1Cb 2)	 250 250 3)	ns
Input current at each I/O pin when input voltage = 0.4 V to 0.9 VDD max.	Ii	−10	10	−10 3)	10 3)	μa
Capacitance of each I/O pin	Ci	—	10	—	10	pF

n/a = Not available

1) Maximum VIH = VDD max. + 0.5 V

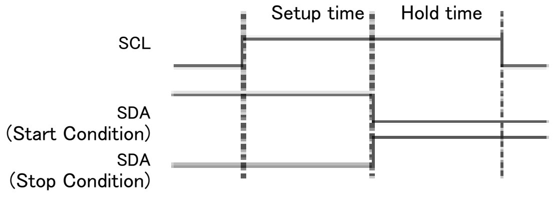
2) Cb = capacitance (in pF) on one bus line. The maximum tF (300 ns) of the SDA and SCL bus lines are greater than the maximum tOF (250 ns) at the output stage.
Consequently, series protective resistors (Rs) can be connected between the SDA/SCL pins and the SDA/SCL bus lines without causing the fall time to exceed the maximum rated tF.

3) It is necessary that when VDD supply is cut off, I/O pins will not disturb the SDA and SCL lines.

Start/Stop Condition Setup and Hold Times
The M32C/83,85's start/stop condition detection setup and hold times do not always conform to I^{2}C bus standards.(During high-speed mode)

The M32C/83,85's start/stop condition detection setup and hold times during this mode are given below.

Setup time > 3-6 cycles Note 1
Hold time > 3-6 cycles Note 1



Note 1: The duration of time here is indicated by the number of main clock input f(Xin) cycles.
In high-speed mode I^{2}C bus standards, both the start condition and stop condition setup and hold times are stipulated to be 600 ns at minimum. On the other hand, the M32C/83,85's setup and hold times are equal to 6 f(Xin) cycles at minimum. Consequently, if the main clock f(Xin) of 10 MHz is used, the setup and hold times of the M32C/83,85's simple I^{2}C bus is 600 ns at minimum, compliant with the high-speed mode I^{2}C bus standards. However, if the main clock of less than 10 MHz is used, the M32C/83,85's setup and hold times do not satisfy the high-speed mode I^{2}C bus standards.

High and Low Level Input Voltages

The electrical characteristics of the M32C/83,85 are such that when operating with 2.7 V to 5.5 V, the high and low level input voltages respectively are guaranteed to be
High level input voltage (VIH) = 0.8 Vcc min.
Low level input voltage (VIL) = 0.2 Vcc max.
These values differ from I^2C bus standards, in which VIH and VIL respectively are stipulated to be 3 V and 1.5 V when operating with 5 V, or 0.7 V and 0.3 V when operating with a supply voltage other than that.
Also, the M32C/83,85's output low voltage (VOL) is guaranteed to be 2.0 V max.when Vcc = 5 V and IOL = 5 mA. This also differ from I^2C bus standards, in which VOL is stipulated to be 0.6 V max. (at IOL = 6 mA).
In the standard characteristics of the M32C/83,85, however, the output low voltage (VOL) is approx. 0.6 V when Vcc = 5 V and IOL = 5 mA.

3.2 Limitations on Maximum Transfer Rate by BRG Count Source

The time that the M32C/83,85 requires before it can recognize the SCL level depends on the sampling period. At maximum, this is equivalent to three BRG count source clock cycles.
Therefore, the maximum transfer rate of the I^2C bus that can be connected to the simple I^2C bus of the M32C/83,85 is limited by the operating clock frequency of the M32C/83,85 and the clock period of the BRG count source selected by the BRG count source setting bit.
Unless the I^2C bus is used at the transfer rate that satisfies the condition given below, a bit displacement may occur.
Maximum transfer rate of I^2C bus (Hz) < **BRG count source (Hz)** / 3
Example: When the original oscillator frequency is 10 MHz and the selected BRG count source is fc32,
Maximum transfer rate of I^2C bus (Hz) < **10 MHz** / **32** / 3 = 104 Kbps
The maximum transfer rate of the I^2C bus in this case is 104 Kbps.

3.3 Limitations on Maximum oscillation frequency for Simple I^2C Bus mode

The maximum oscillation frequency is depend on M32C/83,85 chip's Maximum oscillation frequency.
Therefore,The maximum oscillation frequency is 30MHz.

4.0 Reference

Renesas Technology Corporation Semiconductor Home Page
http://www.renesas.com

E-mail Support
E-mail: support_apl@renesas.com

Data Sheet
M32C/83 group REV.1.02
(Use the latest version on the home page: http://www.renesas.com)

・Appendix

I^{2}C Bus Controller Software Specifications

・This program is provided for only reference purposes, and does not guarantee the communication operation of the I^{2}C bus in user applications. Furthermore, because the communication operation in a system cannot be evaluated with this software alone,
it is recommended that the communication operation be evaluated in the user's final system.

Table of Contents

1. Overview
2. Functional Description
   2.1 Addresses
   2.2 Transfer Rate
   2.3 Transfer Data Length
   2.4 Multi-Masters
3. Hardware Description
4. How to Use
   4.1 How to Incorporate
   4.2 Memory Used
   4.3 Functions
5. Program List

1. Overview

This software is designed to implement the I^2C bus communication protocol by controlling the simple I^2C bus hardware built into the M32C/83 group of microcomputers.
The I^2C bus communication protocol can be complied with when used under the conditions given below.
<Operating condition>
Oscillator frequency: 20 MHz (zero wait, not divided)
[Note] If used without 20MHz,UiBRG and UiSMR3's DL2–0 must be set suitable value.
For reference about it,see Method for Sending and Receiving Byte Data in Section 2.1, the clock delay function in Section 2.6, Chapter 3 Precautions on Simple I2C Bus Mode
<Specificational limitations>
・Communication between only 7–bit address devices is supported.
・No special addresses (e.g., general call address) can be used.
・Coexistence with other I^2C bus compatible protocols such as C–BUS and M3L–BUS is not supported.
・ No communication formats in which slaves are switched over by using a restart condition cannot be put onto the bus. (Communication may be adversely affected.)
・Because the bus collision interrupt vector is shared, UART0 and UART3 cannot be used in I^2C bus mode at the same time. Nor can UART1 and UART4 be used at the same time.

2. Functional Description

2.1 Addresses

<Master device>
Send and receive data to and from slave devices with 7–bit addresses.
<Slave devices>
Have a 7–bit address.
Note: Transmission/reception to and from special addresses (e.g., general call address) is not supported.

2.2 Transfer Rate

The useful transfer rate is 0 to 100 Kbps. Therefore, communication with high–speed mode masters cannot be performed.

2.3 Transfer Data Length

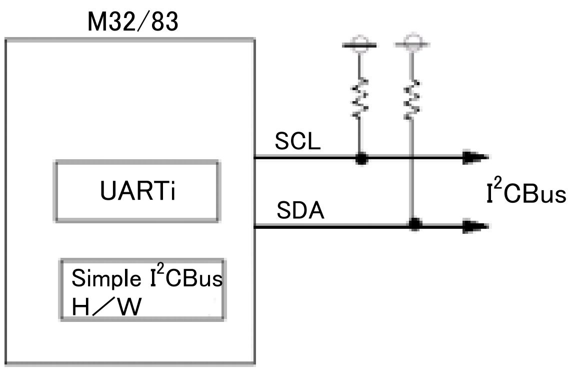
<Master device>
Can send and receive 1 to 256 bytes of data.
<Slave devices>
The data length is passed to iic_index which is the argument to the iic0_slave_end() function.
Its range is 1 to 256.

2.4 Multi–Masters

Multiple devices connected to the I^2C bus can send data to other devices.

3. Hardware Description

The M32C/83's UARTi (i = 0–4) and its internal simple I^2C bus hardware only are used to implement the I^2C bus communication protocol. The appropriate pullup resistors that suit the user system need to be selected.



4. How to Use

4.1 How to Incorporate

[ncrt.a30]

(1) Add the line shown below as the last section entry in the near area.

It determines a location in which the 10 bytes of RAM used by the I^2C bus software is reserved.

```
.section iicbus, data, align
```

(2) Add the interrupt vectors shown below. (i= 0–4)

Software interrupt numbers 39, 40 and 41 (bus collision detection interrupt)

```
.glb s?s_int
.lword s?s_int (? = 0–4)
```

Software interrupt numbers 18, 20, 34, 36 and 38 (UARTi receive interrupt)

```
.glb s?r_int
.lword s?r_int (? = 0–4)
```

Software interrupt numbers 19, 33, 35 and 37 (UARTi transmit interrupt)

```
.glb s?t_int
.lword s?t_int (? = 0–4)
```

[Access–inhibited registers]

Do not modify the registers listed below.

(i = 0–4. However, this is limited to the UART that is using the I^2C bus.)

Register name	bit							
	7	6	5	4	3	2	1	0
UARTi Bus collision detection interrupt control register	×	×	×	×	×	×	×	×
UARTi Transmit control register	×	×	×	×	×	×	×	×
UARTi Receive control register	×	×	×	×	×	×	×	×
UARTi Special mode register	×	×	×	×	×	×	×	×
UARTi Special mode register 2	×	×	×	×	×	×	×	×
UARTi Special mode register 3	×	×	×	×	×	×	×	×
UARTi Special mode register 4	×	×	×	×	×	×	×	×
UARTi Transmit/receive mode register	×	×	×	×	×	×	×	×
UARTi Bit rate generator	×	×	×	×	×	×	×	×
UARTi Transfer buffer register	×	×	×	×	×	×	×	×
UARTi Transmit/receive control register 0	×	×	×	×	×	×	×	×
UARTi Transmit/receive control register 1	×	×	×	×	×	×	×	×
UARTi Receive buffer register	×	×	×	×	×	×	×	×
Port P6 register	×1	×1	○	○	×0	×0	○	○
Port P6 direction register	×1	×1	○	○	×0	×0	○	○
Port P7 register	○	○	○	○	○	○	×2	×2
Port P7 direction register	○	○	○	○	○	○	×2	×2
Port P9 register	×4	×4	○	○	○	×3	×3	○
Port P9 direction register	×4	×4	○	○	○	×3	×3	○
Function select register A0	×1	×1	○	○	×0	×0	○	○
Function select register A1	○	○	○	○	○	○	×2	×2
Function select register A3	×4	×4	○	○	○	×3	×3	○
Function select register B0	×1	×1	○	○	×0	×0	○	○
Function select register B1	○	○	○	○	○	○	×2	×2
Function select register B3	×4	×4	○	○	○	×3	×3	○
Function select register C	○	○	○	○	○	○	×2	×2
External interrupt request cause select register	×B	×A	○	○	○	○	○	○

xi: Access is inhibited when using UARTi as the simple I^2C bus. (i = 0–4)
xA: Access is inhibited when using UARTj as the simple I^2C bus. (j = 0 or 3)
xB: Access is inhibited when using UARTk as the simple I^2C bus. (k = 1 or 4)

4.2 Memory Used

RAM size: 10 bytes
ROM size: 1,430 bytes

4.3 Functions

・Initialization function:
char iic_ini (char SWITCH);

Description: This function initializes the I^2C bus to allow for transmission/reception to be performed on it. When this processing is completed and interrupts are enabled, UARTi starts operating as a slave device. Furthermore, by calling the functions to start master transmission/reception described below, UARTi can operate as a master device.

Arguments	SWITCH	0: I^2C facility disabled
		1: I^2C facility enabled
Returns		0: Failed
		1: Succeeded

Other:
If the I^2C facility is disabled, the next functions indicated cannot be used.

・Master start function
char iic_master_start (char SLAVE, char RW, char * BUF, char LEN);

Description: This function starts master control.
Before this function can be used, the I^2C bus must be readied for use by iic_ini.

Arguments	SLAVE	0x00–0x7f: Slave device address to be specified
	RW	0: Master transmit operation
		1: Master receive operation
	*BUF	: Pointer to the transmit or receive buffer
	LEN	0x00–0xff: Communication data length
Returns		0: Failed to start master control
		1: Succeeded to start master control

・Master EEPROM random read start function
char iic_master_randomread (char SLAVE, char ROM_ADR, char * BUF, char LEN);

Description: This function starts random read on EEPROM.
Before this function can be used, the I^2C bus must be readied for use by iic_ini.

Arguments	SLAVE	0x00–0x7f: EEPROM address to be specified
	ROM_ADR	: Address in EEPROM from which to read
	*BUF	: Pointer to the receive buffer
	LEN	0x00–0xff: Number of received data
Returns		0: Failed to start master control
		1: Succeeded to start master control

[I^{2}C Bus Software Functions Created by the User]
In the I^{2}C bus software, the following functions are called that have as arguments the transmit/receive status and the data count thereof.
These functions must be supplied to the I^{2}C bus software by the user.

・Master control–finished function
    void iic_master_end (char STATUS);
Description: This function is called by the firmware after master control is completed.
    The status in which master communication has terminated is notified to the user by the arguments below.

Arguments	STATUS	High-order 4 bits	0: Master transmission
			1: Master reception
			2: EEPROM random read
		Low-order 4 bits	0: Terminated normally
			1: Lost in first byte bus contention
			2: Lost in bus contention
			3: Terminated in NACK
			4: Start condition error
			5: Stop condition error
			6: Unknown error
Returns	None		

Other: Called from within the interrupt handling of the I^{2}C bus software.

・ Slave check function
    *char iic_id_chk (char ID, char RW);
Description: This function is called by the firmware after receiving the first byte.
    The contents of requests from the master to slaves are notified to the user by the arguments shown below. If a null pointer is returned, slave specification is denied; if a pointer to the communication buffer is returned, slave operation is initiated.

Arguments	ID	0x00–0x7f: Slave device address specified by the master
	RW	0: Reception requested by the master (Slave performs reception)
		1: Transmission requested by the master (Slave performs transmission)
Returns	NULL pointer	: Slave specification denied
	pointer	: Pointer to the transmit or receive buffer

・Slave control–finished function
    void iic_slave_end (char STATUS, char IIC_INDEX);
Description: This function is called by the firmware after slave control is completed.
    The status in which slave communication has terminated is notified to the user by the arguments below.

Arguments	STATUS	High-order 4 bits	0: Slave reception
			1: Slave transmission
		Low-order 4 bits	0: Terminated normally
			1: Lost in bus contention
			2: Terminated in NACK
			3: Start condition error
			4: Stop condition error
			5: Unknown error
	IIC_INDEX	0x00–0xff: Number of received data	
Returns	None		

4.4 Communication Method

4.4.1 Preparation

For I^2C bus communication to be performed, iic_ini must be called at the initial stage of program operation (such as the initial routine) as shown by an example below.
When calling iic_ini, pass one argument to it.
The first argument indicates whether the I^2C bus facility is enabled or not.

```
(System initialize function)
char iic_ini(1);            //IIC-Bus Initialize
asm("fset I");              //Flag I set
```

4.4.2 Master Communication

To start master communication, call iic_master_start. When calling iic_master_start, pass four arguments to it.
The first argument specifies the address of the other device to which to send. Do not use any functional address to specify this address.
The second argument specifies the type of master communication to be performed, transmission or reception. If the value is 0, transmission is performed; if 1, reception is performed.
The third argument specifies the start address of a location in which data is to be stored.
If this location is not freed until after master communication terminates, this location can be anywhere in the RAM area with the near attribute.
The fourth argument specifies the transmit data length. If the value 0 is specified, 256 bytes are transmitted which is the maximum transmit data length available.
Note that iic_master_start has a return value. It returns the value 0 when master communication is started, or the value 1 when master communication is not started.

In the example below, 5 bytes of data is transmitted from iic_ram to the slave device whose address is 55_{16}.
Example:

```
if (iic_master_start (0x55, 0, char *iic_ram, 5)!=0){
    // Processing to check whether master communication has failed
}else{
    // Processing to check whether master communication has started
}
```

When master communication has finished, the I^{2}C bus software calls iic_master_end. This function needs to be created by the user. When the I^{2}C bus software calls iic_master_end, it passes one argument to the called function. This argument indicates the status in which master communication has terminated. The content of the status is shown in Section 4.3. Next, an example of iic_master_end is shown below.

```
//Prototype
 void iic_master_end(char);
//Master control finished function
 void iic_master_end(char status){
    if((status&0xf0)==0x10){      //Transmit mode
        switch (status&0x0f){     //Transmit status processing
            case 0:
                //Terminated normally
                break;
            case 1:
                //Detected NACK signal when transmitted the first byte
                break;
            case 2:
                //Detected NACK signal when transmitted on end after second byte
                break;
            case 3:
                //Lost in bus contention
                break;
            case 4:
                //Start condition error
                break;
            case 5:
                //Stop condition error
                default:break;
        }
    }else if((status&0xf0)==0x20){    //Receive mode
                                      //Receive status processing. Perform this processing
    }                                 //in the same way as for transmit status processing.
    }else if((status&0xf0)==0x30){    //EEPROM mode
                                      //EEPROM mode status processing. Perform this processing
    }                                 //in the same way as for transmit status processing.
}
```

4.4.3 Slave Communication

When receiving one byte of data from the master, the I^{2}C bus software calls the iic_id_chk function.
This function needs to be created by the user.
The I^{2}C bus software passes the contents of requests made to slaves by the master to the called function by using the arguments shown below.
The first argument indicates the address of the slave device specified by the master.
The second argument indicates the content of communication requested by the master.
If the function returns a null pointer, it means that slave specification is defined; if a pointer to the communication buffer is returned, it means that slave operation is started.

```
//Prototupe
* char iic_id_chk(char, char);

//Slave check function
unsigned char sw_buf[256]           //Transmit buffer
unsigned char sr_buf[256]           //Receive buffer
* char iic_id_chk(char ID, char RW){
    if(ID==0x55){                   //when local-unit's address is 0x55
        if(R/W==1){
            return(&sw_buf[0]);     //Slave transmission
        }else{
            return(&sr_buf[0]);     //Slave reception
        }
    }else{
        return(0)                   //Nothing slave control
    }
}
```

When slave communication has finished, the I^{2}C bus software calls iic_slave_end. This function need to be created by the user. When the I^{2}C bus software calls iic_slave_end, it passes two arguments to the called function.
The first argument indicates the status of slave communication.
The second argument indicates the number of data received by the slave.
The contents of the status are shown in Section 4.3.
Next, an example of iic_slave_end is shown below.

```
//Prototupe
void iic_slave_end(char,char);

//Slave check function
 void iic_slave_end(char status,char iic_index){
    if((status&0xf0)==0x10){       //Transmit mode
        switch (status&0x0f){      //Transmit status processing
            case 0:
                //Terminated normally
                break;
            case 1:
                //Detected NACK signal when transmitted the first byte
                break;
            case 2:
                //Detected NACK signal when transmitted on end after second byte
                break;
            case 3:
                //Lost in bus contention
                break;
            case 4:
                //Start condition error
                break;
            case 5:
                //Stop condition error
                default:break;
        }
    }else{                         //Receive mode
        switch (status&0x0f){      //Receive status processing
            case 0:
                //Terminated normally(Received data that size of iic_index)
                break;
            case 1:
                //Detected NACK signal when transmitted the first byte
                break;
            case 2:
                //Detected NACK signal when transmitted on end after second byte
                break;
            case 3:
                //Lost in bus contention
                break;
            case 4:
                //Start condition error
                break;
            case 5:
                //Stop condition error
                default:break;
    }
}
```

5. Program List

```
/*""file comment""****************************************************************
;System:IIC-BUS F/W Ver0.81(Sample program)
;Outline description: M32C/83(20 MHz, not divided internally)
;Multi-master/slave communication
;Communication rate: 100 Kbps
;Functional addresses and 10-bit addresses inhibited
;Only C language supported for the interface
;C bus, M3Low, etc. cannot be connected
;None of fail-safe features incorporated
;Date:Sep./5/2002(Thu)
;Object file name:i2cbus.c
;*******************************************************************************
; copyright 2003 Renesas Technology Corporation
; and Renesas Solutions Corporation
;""file comment end""**********************************************************/

/*******************************************************************************
; Prototype definition
;******************************************************************************/
#define uarti 0       //Used UARTx when #define uarti x (x=0～4)
#if uarti == 0
unsigned char iic0_ini(unsigned char);
unsigned char iic0_master_start(unsigned char,
        unsigned char,
        unsigned char*,
        unsigned char);
unsigned char iic0_master_randomread(unsigned char,
        unsigned char,
        unsigned char*,
        unsigned char);
void iic0_master_end(unsigned char);
unsigned char* iic0_id_check(unsigned char,
        unsigned char);
void iic0_slave_end(unsigned char,
        unsigned char);
void s0s_int(void);
void s0r_int(void);
void s0t_int(void);
#elif uarti == 1
unsigned char iic1_ini(unsigned char);
unsigned char iic1_master_start(unsigned char,
        unsigned char,
        unsigned char*,
        unsigned char);
unsigned char iic1_master_randomread(unsigned char,
        unsigned char,
        unsigned char*,
        unsigned char);
void iic1_master_end(unsigned char);
unsigned char* iic1_id_check(unsigned char,
        unsigned char);
void iic1_slave_end(unsigned char,
        unsigned char);
void s1s_int(void);
void s1r_int(void);
void s1t_int(void);
```

```
#elif uarti == 2
unsigned char iic2_ini(unsigned char);
unsigned char iic2_master_start(unsigned char,
        unsigned char,
        unsigned char*,
        unsigned char);
unsigned char iic2_master_randomread(unsigned char,
        unsigned char,
        unsigned char*,
        unsigned char);
void iic2_master_end(unsigned char);
unsigned char* iic2_id_check(unsigned char,
        unsigned char);
void iic2_slave_end(unsigned char,
        unsigned char);
void s2s_int(void);
void s2r_int(void);
void s2t_int(void);
#elif uarti == 3
unsigned char iic3_ini(unsigned char);
unsigned char iic3_master_start(unsigned char,
        unsigned char,
        unsigned char*,
        unsigned char);
unsigned char iic3_master_randomread(unsigned char,
        unsigned char,
        unsigned char*,
        unsigned char);
void iic3_master_end(unsigned char);
unsigned char* iic3_id_check(unsigned char,
        unsigned char);
void iic3_slave_end(unsigned char,
        unsigned char);
void s3s_int(void);
void s3r_int(void);
void s3t_int(void);
#elif uarti == 4
unsigned char iic4_ini(unsigned char);
unsigned char iic4_master_start(unsigned char,
        unsigned char,
        unsigned char*,
        unsigned char);
unsigned char iic4_master_randomread(unsigned char,
        unsigned char,
        unsigned char*,
        unsigned char);
void iic4_master_end(unsigned char);
unsigned char* iic4_id_check(unsigned char,
        unsigned char);
void iic4_slave_end(unsigned char,
        unsigned char);
void s4s_int(void);
void s4r_int(void);
void s4t_int(void);
#endif
static void sta_int(void);
static void stp_int(void);
```

```
/****************************************************************************
; SFR
;****************************************************************************/
#if uarti == 0
#pragma ADDRESS uismr4 0364H
#pragma ADDRESS uismr3 0365H
#pragma ADDRESS uismr2 0366H
#pragma ADDRESS uismr 0367H
#pragma ADDRESS uimr 0368H
#pragma ADDRESS uibrg 0369H
#pragma ADDRESS uitb 036aH
#pragma ADDRESS uic0 036cH
#pragma ADDRESS uic1 036dH
#pragma ADDRESS uirb 036eH
#pragma ADDRESS bcniic 0071H
#pragma ADDRESS sitic 0090H
#pragma ADDRESS siric 0072H
#pragma ADDRESS ps0  03b0H
#pragma ADDRESS psl0 03b2H
#pragma ADDRESS p6  03c0H
#pragma ADDRESS pd6  03c2H
#pragma ADDRESS ifsr 031fH
#elif uarti == 1
#pragma ADDRESS uismr4 02e4H
#pragma ADDRESS uismr3 02e5H
#pragma ADDRESS uismr2 02e6H
#pragma ADDRESS uismr 02e7H
#pragma ADDRESS uimr 02e8H
#pragma ADDRESS uibrg 02e9H
#pragma ADDRESS uitb 02eaH
#pragma ADDRESS uic0 02ecH
#pragma ADDRESS uic1 02edH
#pragma ADDRESS uirb 02eeH
#pragma ADDRESS bcniic 0091H
#pragma ADDRESS sitic 0092H
#pragma ADDRESS siric 0074H
#pragma ADDRESS ps0  03b0H
#pragma ADDRESS psl0 03b2H
#pragma ADDRESS p6  03c0H
#pragma ADDRESS pd6  03c2H
#pragma ADDRESS ifsr 031fH
#elif uarti == 2
#pragma ADDRESS uismr4 0334H
#pragma ADDRESS uismr3 0335H
#pragma ADDRESS uismr2 0336H
#pragma ADDRESS uismr 0337H
#pragma ADDRESS uimr 0338H
#pragma ADDRESS uibrg 0339H
#pragma ADDRESS uitb 033aH
#pragma ADDRESS uic0 033cH
#pragma ADDRESS uic1 033dH
#pragma ADDRESS uirb 033eH
#pragma ADDRESS bcniic 008fH
#pragma ADDRESS sitic 0089H
#pragma ADDRESS siric 006bH
#pragma ADDRESS ps1  03b1H
#pragma ADDRESS psl1 03b3H
#pragma ADDRESS psc  03afH
#pragma ADDRESS p7  03c1H
#pragma ADDRESS pd7  03c3H
```

```
#elif uarti == 3
#pragma ADDRESS uismr4 0324H
#pragma ADDRESS uismr3 0325H
#pragma ADDRESS uismr2 0326H
#pragma ADDRESS uismr 0327H
#pragma ADDRESS uimr 0328H
#pragma ADDRESS uibrg 0329H
#pragma ADDRESS uitb 032aH
#pragma ADDRESS uic0 032cH
#pragma ADDRESS uic1 032dH
#pragma ADDRESS uirb 032eH
#pragma ADDRESS bcniic 0071H
#pragma ADDRESS sitic 008bH
#pragma ADDRESS siric 006dH
#pragma ADDRESS ps3  03b5H
#pragma ADDRESS psl3 03b7H
#pragma ADDRESS p9  03c5H
#pragma ADDRESS pd9  03c7H
#pragma ADDRESS ifsr 031fH
#elif uarti == 4
#pragma ADDRESS uismr4 02f4H
#pragma ADDRESS uismr3 02f5H
#pragma ADDRESS uismr2 02f6H
#pragma ADDRESS uismr 02f7H
#pragma ADDRESS uimr 02f8H
#pragma ADDRESS uibrg 02f9H
#pragma ADDRESS uitb 02faH
#pragma ADDRESS uic0 02fcH
#pragma ADDRESS uic1 02fdH
#pragma ADDRESS uirb 02feH
#pragma ADDRESS bcniic 0091H
#pragma ADDRESS sitic 008dH
#pragma ADDRESS siric 006fH
#pragma ADDRESS ps3  03b5H
#pragma ADDRESS psl3 03b7H
#pragma ADDRESS p9  03c5H
#pragma ADDRESS pd9  03c7H
#pragma ADDRESS ifsr 031fH
#endif
#pragma ADDRESS prcr 000aH
union{
 struct{
  char b0:1;
  char b1:1;
  char b2:1;
  char b3:1;
  char b4:1;
  char b5:1;
  char b6:1;
  char b7:1;
 }bit;
 char byte;
}bcniic,sitic,siric,prcr,
uimr,uibrg,uic0,uic1,uismr,uismr2,uismr3,uismr4,
#if uarti == 0 || uarti == 1
ps0,psl0,p6,pd6,ifsr;
#elif uarti == 2
ps1,psl1,psc,p7,pd7;
#elif uarti == 3 || uarti == 4
ps3,psl3,p9,pd9,ifsr;
#endif
```

```
union{
 struct{
  char b0:1;
  char b1:1;
  char b2:1;
  char b3:1;
  char b4:1;
  char b5:1;
  char b6:1;
  char b7:1;
  char b8:1;
  char b9:1;
  char b10:1;
  char b11:1;
  char b12:1;
  char b13:1;
  char b14:1;
  char b15:1;
 }bit;
 struct{
  char bytel:8;
  char byteh:8;
 }byte;
 int word;
}uitb,uirb;
#define UiSMR4 uismr4.byte
#define UiSMR3 uismr3.byte
#define UiSMR2 uismr2.byte
#define UiSMR uismr.byte
#define UiMR uimr.byte
#define UiBRG uibrg.byte
#define UiTB uitb.word
#define UiC0 uic0.byte
#define UiC1 uic1.byte
#define UiRB uirb.word
#define SiSIC bcniic.byte
#define SiTIC sitic.byte
#define SiRIC siric.byte
#define stareq uismr4.bit.b0  // Start condition generate bit
#define rstareq uismr4.bit.b1 // Restart condition generate bit
#define stpreq uismr4.bit.b2  // Stop condition generate bit
#define stspsel uismr4.bit.b3 // SCL,SDA output select bit
#define ackd uismr4.bit.b4    // ACK data bit
#define ackc uismr4.bit.b5    // ACK data output enable bit
#define sclhi uismr4.bit.b6   // SCL output stop enable bit
#define swc9 uismr4.bit.b7    // SCL wait output bit3(final pulse)
#define ckdir uimr.bit.b3     // Internal/external clock select bit
#define csc  uismr2.bit.b1    // Clock synchronous bit
#define swc  uismr2.bit.b2    // SCL wait output bit(9th pulse)
#define als  uismr2.bit.b3    // SDA output stop bit
#define stc  uismr2.bit.b4    // UARTi initialize bit
#define abl  uirb.bit.b11     // Arbitration lost detecting flag
#define PRCR prcr.byte        //
#if uarti == 0
#define ps_scl ps0.bit.b2     // Function select A reg. scl bit
#define psl_scl psl0.bit.b2   // Function select B reg. scl bit
#define ps_sda ps0.bit.b3     // Function select A reg. sda bit
#define p_sda p6.bit.b3       // SDA port data bit
#define p_scl p6.bit.b2       // SCL port data bit
#define pd_sda pd6.bit.b3     // SDA port direction bit
#define pd_scl pd6.bit.b2     // SCL port direction bit
#define IFSR ifsr.byte        //
```

```
#elif uarti == 1
#define ps_scl ps0.bit.b6     // Function select A reg. scl bit
#define psl_scl psl0.bit.b6   // Function select B reg. scl bit
#define ps_sda ps0.bit.b7     // Function select A reg. sda bit
#define p_sda p6.bit.b7       // SDA port data bit
#define p_scl p6.bit.b6       // SCL port data bit
#define pd_sda pd6.bit.b7     // SDA port direction bit
#define pd_scl pd6.bit.b6     // SCL port direction bit
#define IFSR ifsr.byte        //
#elif uarti == 2
#define ps_scl ps1.bit.b1     // Function select A reg. scl bit
#define psl_scl psl1.bit.b1   // Function select B reg. scl bit
#define psc_scl psc.bit.b1    // Function select C reg. scl bit
#define ps_sda ps1.bit.b0     // Function select A reg. sda bit
#define psl_sda psl1.bit.b0   // Function select B reg. sda bit
#define psc_sda psc.bit.b0    // Function select C reg. sda bit
#define p_sda p7.bit.b0       // SDA port data bit
#define p_scl p7.bit.b1       // SCL port data bit
#define pd_sda pd7.bit.b0     // SDA port direction bit
#define pd_scl pd7.bit.b1     // SCL port direction bit
#elif uarti == 3
#define ps_scl ps3.bit.b1     // Function select A reg. scl bit
#define psl_scl psl3.bit.b1   // Function select B reg. scl bit
#define ps_sda ps3.bit.b2     // Function select A reg. sda bit
#define psl_sda psl3.bit.b2   // Function select B reg. sda bit
#define p_sda p9.bit.b2       // SDA port data bit
#define p_scl p9.bit.b1       // SCL port data bit
#define pd_sda pd9.bit.b2     // SDA port direction bit
#define pd_scl pd9.bit.b1     // SCL port direction bit
#define IFSR ifsr.byte        //
#elif uarti == 4
#define ps_scl ps3.bit.b7     // Function select A reg. scl bit
#define psl_scl psl3.bit.b7   // Function select B reg. scl bit
#define ps_sda ps3.bit.b6     // Function select A reg. sda bit
#define p_sda p9.bit.b6       // SDA port data bit
#define p_scl p9.bit.b7       // SCL port data bit
#define pd_sda pd9.bit.b6     // SDA port direction bit
#define pd_scl pd9.bit.b7     // SCL port direction bit
#define IFSR ifsr.byte        //
#endif
/****************************************************************************
; Memories definition
;****************************************************************************/
typedef union{
 struct{
  unsigned char b0:1;
  unsigned char b1:1;
  unsigned char b2:1;
  unsigned char b3:1;
  unsigned char b4:1;
  unsigned char b5:1;
  unsigned char b6:1;
  unsigned char b7:1;
 }bit;
 unsigned char all;
}byte_dt;
```

```
typedef union{
 struct{
  unsigned char b0:1;
  unsigned char b1:1;
  unsigned char b2:1;
  unsigned char b3:1;
  unsigned char b4:1;
  unsigned char b5:1;
  unsigned char b6:1;
  unsigned char b7:1;
  unsigned char b8:1;
  unsigned char b9:1;
  unsigned char b10:1;
  unsigned char b11:1;
  unsigned char b12:1;
  unsigned char b13:1;
  unsigned char b14:1;
  unsigned char b15:1;
 }bit;
 struct{
  unsigned char byte0:8;
  unsigned char byte1:8;
 }byte;
 unsigned int all;
}word_dt;
static byte_dt iic_md;              // IICbus mode
#define iic_mode iic_md.all
#define f_rw iic_md.bit.b0          //  0:wite  1:read
#define f_ms iic_md.bit.b4          //  0:slave  1:master
#define f_ep iic_md.bit.b5          //  0:slave  1:master
#define f_sr iic_md.bit.b7          //  0:receive 1:send
static byte_dt iic_sl;              // Master 1st byte
#define iic_slave iic_sl.all
#define iic_rw iic_sl.bit.b0        //  0:write  1:read
static unsigned char iic_length;    // Master length
static unsigned char iic_index;
static unsigned char *iic_pointer;  // pointer
static unsigned char iic_eeplen;    //
static unsigned char iic_eepadr;    //
/*******************************************************************************
; IICbus initialize function
;""func comment end""*********************************************************/
#if uarti == 0
unsigned char iic0_ini(unsigned char ini){
#elif uarti == 1
unsigned char iic1_ini(unsigned char ini){
#elif uarti == 2
unsigned char iic2_ini(unsigned char ini){
#elif uarti == 3
unsigned char iic3_ini(unsigned char ini){
#elif uarti == 4
unsigned char iic4_ini(unsigned char ini){
#endif
 if(ini == 1){      // IICbus mode initialize(START)
  UiMR = 0x0a;      // 9bit SI/O mode(ext. clock)
  UiBRG = 100-1;    // 100KBPS
  UiC0 = 0xb0;      // MSB first,f1,Nch,CTS disable
  UiSMR = 0x01;     // IICbus mode,Arbitration lost flag Update per byte
  UiSMR2 = 0x11;    // transfer/receive interrupt,disalbe Clock sync,UART initialize enable
  UiSMR3 = 0x62;    // SDA delay = 4-cycle of BRG count source
  UiSMR4 = 0x30;    // ACK data output(SDA="H")
  UiC1 = 0x15;      // Transfer/Receive enable
#if uarti == 0
  ps_scl = 1;        // 0:port62  1:psl62
  ps_sda = 1;        // 0:port63  1:TXD0/SDA0
  psl_scl = 0;       // 0:SCL0  1:STXD0
  IFSR |= 0x40;      //
#elif uarti == 1
  ps_scl = 1;        // 0:port66  1:psl67
  ps_sda = 1;        // 0:port67  1:TXD1/SDA1
  psl_scl = 0;       // 0:SCL1  1:STXD1
  pd_sda = 0;        // SDA-port input
  pd_scl = 0;        // SCL-port input
  IFSR |= 0x80;      //
#elif uarti == 2
  ps_scl = 1;        // 0:port71  1:psl71
  ps_sda = 1;        // 0:port70  1:psl70
  psl_scl = 0;       // 0:psc71  1:STXD2
  psl_sda = 0;       // 0:psc70  1:TA0out
  psc_scl = 0;       // 0:SCL2  1:OUTC22
  psc_sda = 0;       // 0:SDA2/TXD2 1:IEout/ISTXD2/OUTC20
  pd_sda = 0;        // SDA-port input
  pd_scl = 0;        // SCL-port input
#elif uarti == 3
  PRCR = 0x04;
  ps_scl = 1;        // 0:port91  1:psl91
  PRCR = 0x04;
  ps_sda = 1;        // 0:port92  1:psl92
  psl_scl = 0;       // 0:SCL3  1:STXD3
  psl_sda = 0;       // 0:SDA3/TXD3 1:IEout/OUTC20
  PRCR = 0x04;
  pd_sda = 0;        // SDA-port input
  PRCR = 0x04;
  pd_scl = 0;        // SCL-port input
  IFSR &= 0xbf;      //
#elif uarti == 4
  PRCR = 0x04;
  ps_scl = 1;        // 0:port97  1:psl97
  PRCR = 0x04;
  ps_sda = 1;        // 0:port96  1:TXD4/SDA4
  psl_scl = 0;       // 0:SCL4  1:STXD4
  PRCR = 0x04;
  pd_sda = 0;        // SDA-port input
  PRCR = 0x04;
  pd_scl = 0;        // SCL-port input
  IFSR &= 0x7f;      //
#endif
  iic_mode = 0x00;   // Slave mode
  iic_index = 0x00;  //
  SiSIC = 0x00;
  SiTIC = 0x00;
  SiRIC = 0x01;      // Receive int. enable
 }
 else{               // Invalidate IICbus(Stop sequence)
  SiSIC = 0x00;
  SiTIC = 0x00;
  SiRIC = 0x00;
  UiC1 = 0x10;       // Transfer/Receive disable
  UiSMR2 = 0x01;     // UART initialize disable
 }
 return(1);
}
```

```
/*******************************************************************************
; IICbus master mode starts function
;""func comment end""**********************************************************/
#if uarti == 0
unsigned char iic0_master_start(unsigned char slave,
#elif uarti == 1
unsigned char iic1_master_start(unsigned char slave,
#elif uarti == 2
unsigned char iic2_master_start(unsigned char slave,
#elif uarti == 3
unsigned char iic3_master_start(unsigned char slave,
#elif uarti == 4
unsigned char iic4_master_start(unsigned char slave,
#endif
        unsigned char rw,
        unsigned char *buf,
        unsigned char len){
 if(uismr.bit.b2 == 1){
  return(0);
 }
 else{
  asm("pushc FLG");
  asm("fclr I");
  UiSMR = 0x01;            // All bit clear without bit0
  UiSMR4 = 0x70;           // SCL and SDA is output"H"
  UiMR = 0x00;             //
  UiMR = 0x02;             //
  UiBRG = 100-1;           //
  SiSIC = 0x01;            // Start con. int. enable
  UiC1 = 0x10;             // Transfer/Receive disable
  UiSMR2 = 0x03;           // UART init. disable,Clock sync. enable
  UiSMR4 = 0x71;           // Start condition generate
  UiSMR4 = 0x09;           // STSP output enable
  iic_slave = slave << 1;  //
  iic_length = len;        //
  iic_pointer = buf;       //
  if(rw == 0){             //
   iic_mode = 0x10;        // Master transfer mode
   iic_rw = 0;
  }
  else{
   iic_mode = 0x11;        // Master receive mode
   iic_rw = 1;
  }
  asm("popc FLG");
  return(1);
 }
}
/*******************************************************************************
; IIC master EEPROM randam-read function
;""func comment end""**********************************************************/
#if uarti == 0
unsigned char iic0_master_randomread(unsigned char slave,
#elif uarti == 1
unsigned char iic1_master_randomread(unsigned char slave,
#elif uarti == 2
unsigned char iic2_master_randomread(unsigned char slave,
#elif uarti == 3
unsigned char iic3_master_randomread(unsigned char slave,
#elif uarti == 4
unsigned char iic4_master_randomread(unsigned char slave,
#endif
         unsigned char rom_adr,
         unsigned char *buf,
         unsigned char len){
 if(uismr.bit.b2 == 1){
  return(0);
 }
 else{
  UiSMR = 0x01;            // All bit clear without bit0
  UiSMR4 = 0x70;           // SCL and SDA is output"H"
  UiMR = 0x00;             //
  UiMR = 0x02;             //
  UiBRG = 100-1;           //
  SiSIC = 0x01;            // Start int. enable
  UiC1 = 0x10;             // Transfer/Receive disable
  UiSMR2 = 0x03;           // UART init. disable,Clock sync. enable
  UiSMR4 = 0x71;           // Start condition generate
  UiSMR4 = 0x09;           // STSPoutput enable
  iic_slave = slave << 1;  //
  iic_length = 1;          //
  iic_eeplen = len;        //
  iic_eepadr = rom_adr;    //
  iic_pointer = buf;       //
  iic_mode = 0x30;         // Master transfer mode
  iic_rw = 0;
  return(1);
 }
}
/*****************************************************************************
; IIC start/stop condition interrupt function
;""func comment end""*********************************************************/
#if uarti == 0
#pragma INTERRUPT s0s_int
void s0s_int(void){
#elif uarti == 1
#pragma INTERRUPT s1s_int
void s1s_int(void){
#elif uarti == 2
#pragma INTERRUPT s2s_int
void s2s_int(void){
#elif uarti == 3
#pragma INTERRUPT s3s_int
void s3s_int(void){
#elif uarti == 4
#pragma INTERRUPT s4s_int
void s4s_int(void){
#endif
 if(uismr.bit.b2 == 1){   // Start condition interrupt
  sta_int();
 }
 else{                    // Stop condition interrupt
  stp_int();
 }
}
```

```
/******************************************************************************
; IIC start condition function
;""func comment end""*********************************************************/
static void sta_int(void){
 word_dt temp;
 UiC1 = 0x10;                 // Transfer/Receive disable
 UiC1 = 0x15;                 // Transfer/Receive enable
 UiMR = 0x02;
 UiSMR4 = 0x00;               // STSPSEL = 0 (SI/O output sel)
 temp.byte.byte0 = iic_slave; // Slave address set
 temp.byte.byte1 = 0x01;      // NACK data set
 UiTB = temp.all;             // Start 1st byte transfer
 UiRB = 0x00;                 // Arbitration lost flag clear
 UiSMR2 = 0x1f;               // UART init. enable,Clock sync. enable
 SiSIC = 0x01;                // Stop int. enable
 SiRIC = 0x01;                // /Receive int. enable
 iic_index = 0x00;            //
}
/******************************************************************************
; IIC stop condition function
;""func comment end""*********************************************************/
static void stp_int(void){
 PRCR = 0x04;
 ps_scl = 0;
 PRCR = 0x04;
 ps_sda = 0;
 UiMR = 0x00;                     // port set(Purpose:TXFUL,TBFUL flag must be cleared when
slave
                                  // receive)
 UiMR = 0x0a;                     // ext. clock sel
 UiSMR2 = 0x11;                   //
 UiSMR4 = 0x30;                   // ACK data output"H"
 UiC1 = 0x15;                     // transfer/receive enable
 PRCR = 0x04;
 ps_scl = 1;
 PRCR = 0x04;
 ps_sda = 1;
 SiRIC = 0x01;                    // receive int. enable
 SiTIC = 0x00;                    // transfer int. disable
 SiSIC = 0x00;                    // start/stop int. disable
 if(f_ms == 0 && f_sr == 0){      // slave receive
  --iic_index;
#if uarti == 0
  iic0_slave_end(0x00,iic_index); // slave receive complete
#elif uarti == 1
  iic1_slave_end(0x00,iic_index); // slave receive complete
#elif uarti == 2
  iic2_slave_end(0x00,iic_index); // slave receive complete
#elif uarti == 3
  iic3_slave_end(0x00,iic_index); // slave receive complete
#elif uarti == 4
  iic4_slave_end(0x00,iic_index); // slave receive complete
#endif
 }
 iic_mode = 0x00;                 // slave mode
 iic_index = 0x00;                //
}
```

```
/*******************************************************************************
; IIC receive(Falling edge of 9th pulse)interrupt function
;""func comment end""**********************************************************/
#if uarti == 0
#pragma INTERRUPT s0r_int
void s0r_int(void){
#elif uarti == 1
#pragma INTERRUPT s1r_int
void s1r_int(void){
#elif uarti == 2
#pragma INTERRUPT s2r_int
void s2r_int(void){
#elif uarti == 3
#pragma INTERRUPT s3r_int
void s3r_int(void){
#elif uarti == 4
#pragma INTERRUPT s4r_int
void s4r_int(void){
#endif
 word_dt temp;
 temp.all = UiRB;              // Read receive buffer
 if(iic_index == 0x00){        // = On the 1st time =
  f_sr = f_ms ^ temp.bit.b8;   // Saved transfer/receive flags
  if(f_ms == 1){               // = When master=
   if(abl == 1){               // = When Arbitration lost =
#if uarti == 0
    iic0_master_end(0x03);     // Arbitration lost error found
#elif uarti == 1
    iic1_master_end(0x03);     // Arbitration lost error found
#elif uarti == 2
    iic2_master_end(0x03);     // Arbitration lost error found
#elif uarti == 3
    iic3_master_end(0x03);     // Arbitration lost error found
#elif uarti == 4
    iic4_master_end(0x03);     // Arbitration lost error found
#endif
    f_ms = 0;                  // Changed slave mode
    f_sr = ~f_sr;              // Reversed transfer/receive mode
    UiMR = 0x0a;
    goto r1_slave;             // Go to slave function
   }
   else{                       // = When Arbitration win =
    SiTIC = 0x01;              // Transfer int. enable
    UiSMR2 = 0x1b;             // Released SCL line
   }
  }
  else{                        // = When Slave =
r1_slave:
   temp.bit.b7 = 0;            // Masked bit7
   if(f_sr == 1){
#if uarti == 0
    iic_pointer = iic0_id_check(temp.byte.byte0,1); // Slave transfer request
#elif uarti == 1
    iic_pointer = iic1_id_check(temp.byte.byte0,1); // Slave transfer request
#elif uarti == 2
    iic_pointer = iic2_id_check(temp.byte.byte0,1); // Slave transfer request
#elif uarti == 3
```

```
#elif uarti == 4
    iic_pointer = iic4_id_check(temp.byte.byte0,1); // Slave transfer request
#endif
   }
   else{
#if uarti == 0
    iic_pointer = iic0_id_check(temp.byte.byte0,0); // Slave receive request
#elif uarti == 1
    iic_pointer = iic1_id_check(temp.byte.byte0,0); // Slave receive request
#elif uarti == 2
    iic_pointer = iic2_id_check(temp.byte.byte0,0); // Slave receive request
#elif uarti == 3
    iic_pointer = iic3_id_check(temp.byte.byte0,0); // Slave receive request
#elif uarti == 4
    iic_pointer = iic4_id_check(temp.byte.byte0,0); // Slave receive request
#endif
   }
   if(iic_pointer != 0){               // Agreed address
    UiSMR4 = 0xa0;                     // ACK-data output enable
                                       // When Falling edge of last pulse,
                                       // "L"hold enable
    SiTIC = 0x01;                      // Transfer int. enable
    SiSIC = 0x01;                      // Start/stop int. enable
    if(f_sr == 1){
     temp.byte.byte0 = *iic_pointer;   // send-data set
     temp.byte.byte1 = 0x01;           // NACK-send set
     UiTB = temp.all;                  // Data1 transfer start
    }
    else{
     UiTB = 0x00ff;         // dummy data (with ACK)transfer start
    }
   }
   else{                    // disagreed address
    UiSMR4 = 0x30;          // NACK-data output enable
    UiMR = 0x0a;            //
    stc = 1;                // UART initialize enable
    SiRIC = 0x01;           //
    iic_mode = 0x00;
    iic_index = 0x00;
   }
   UiSMR2 = 0x11;           // ALS clear, CSC clear(for Arbitration lost)
  }
 }
 else{                       // = On and after the 2nd time =
  if(f_ms == 1){             // = When master=
   if(f_sr == 1){            // = When transfer =
    if(abl == 1){            // = When Arbitration lost =
#if uarti == 0
     iic0_master_end(0x02);  // Arbitration lost error found
#elif uarti == 1
     iic1_master_end(0x02);  // Arbitration lost error found
#elif uarti == 2
     iic2_master_end(0x02);  // Arbitration lost error found
#elif uarti == 3
     iic3_master_end(0x02);  // Arbitration lost error found
#elif uarti == 4
     iic4_master_end(0x02);  // Arbitration lost error found
#endif
```

```
         UiMR = 0x0a;                   // Ext. clock sel
         UiSMR4 = 0x30;                 //
         UiSMR2 = 0x11;                 //
         iic_mode = 0x00;               // Slave mode set
         iic_index = 0x00;              //
        }
        else{                           // = When Arbitration win =
         als = 0;                       // When Arbitration lost,SDA "HiZ" disable
         SiTIC = 0x01;                  // Transfer int. enable
        }
       }
       else{                            // = When receive =
        abl = 0;                        // Arbitration lost flag clear
        SiTIC = 0x01;                   // Transfer int. enable
       }
       swc = 0;                         // Released SCL line
      }
      else{                             // = When slave =
       if(f_sr == 1){                   // = When transfer =
        temp.byte.byte0 = *iic_pointer; // send-data set
        temp.byte.byte1 = 0x01;         // NACK-data set
        UiTB = temp.all;                // Data1 transfer start
       }
       else{                            // = When receive =
        UiTB = 0x00ff;                  //
       }
       SiTIC = 0x01;                    // Transfer int. enable
       swc = 0;                         //
       swc9 = 1;                        //
      }
     }
    }
    /*******************************************************************************
    ; IIC transfer(Falling edge of last pulse)interrupt function
    ;""func comment end""**********************************************************/
    #if uarti == 0
    #pragma INTERRUPT s0t_int
    void s0t_int(void){
    #elif uarti == 1
    #pragma INTERRUPT s1t_int
    void s1t_int(void){
    #elif uarti == 2
    #pragma INTERRUPT s2t_int
    void s2t_int(void){
    #elif uarti == 3
    #pragma INTERRUPT s3t_int
    void s3t_int(void){
    #elif uarti == 4
    #pragma INTERRUPT s4t_int
    void s4t_int(void){
    #endif
     word_dt temp;
     temp.all = UiRB;
     if(iic_index == 0x00){       // = On the 1st time =
      if(f_ms == 1){              // = When master =
       if(temp.bit.b8 == 1){      // = When NACK found =
        if(f_ep == 0){
         if(f_sr == 1){
```

```
#if uarti == 0
      iic0_master_end(0x03);  // When the 1st byte , NACK found finish
#elif uarti == 1
      iic1_master_end(0x03);  // When the 1st byte , NACK found finish
#elif uarti == 2
      iic2_master_end(0x03);  // When the 1st byte , NACK found finish
#elif uarti == 3
      iic3_master_end(0x03);  // When the 1st byte , NACK found finish
#elif uarti == 4
      iic4_master_end(0x03);  // When the 1st byte , NACK found finish
#endif
     }
     else{
#if uarti == 0
      iic0_master_end(0x13);  // When the 1st byte , NACK found finish
#elif uarti == 1
      iic1_master_end(0x13);  // When the 1st byte , NACK found finish
#elif uarti == 2
      iic2_master_end(0x13);  // When the 1st byte , NACK found finish
#elif uarti == 3
      iic3_master_end(0x13);  // When the 1st byte , NACK found finish
#elif uarti == 4
      iic4_master_end(0x13);  // When the 1st byte , NACK found finish
#endif
     }
    }
    else{
#if uarti == 0
      iic0_master_end(0x23);  // When the 1st byte , NACK found finish
#elif uarti == 1
      iic1_master_end(0x23);  // When the 1st byte , NACK found finish
#elif uarti == 2
      iic2_master_end(0x23);  // When the 1st byte , NACK found finish
#elif uarti == 3
      iic3_master_end(0x23);  // When the 1st byte , NACK found finish
#elif uarti == 4
      iic4_master_end(0x23);  // When the 1st byte , NACK found finish
#endif
    }
    als = 0;                // When Arbitration lost,SDA "HiZ" disable
    UiSMR4 = 0x04;          // Stop condition generate
    UiSMR4 = 0x3c;          // ST/SP output enable
    UiSMR2 = 0x01;          //
    SiRIC = 0x01;           // receive int. enable

   }
   else{                    // = When ACK found =
    if(f_sr == 0){          // = When receive =
     als = 0;               // When Arbitration lost,SDA "HiZ" disable
     if(iic_length == 1){   // = When receive at the last byte=
      UiTB = 0x01ff;        // Send NACK
     }
     else{                  // = When continuous receive =
      UiTB = 0x00ff;        // Send ACK
     }
    }
```

```
      else{        // = When transfer =
       if(f_ep == 0){
        temp.byte.byte0 = *iic_pointer;
       }
       else{
        temp.byte.byte0 = iic_eepadr;
       }
        temp.byte.byte1 = 0x01;      // NACK-data set
        UiTB = temp.all;
        abl = 0;                     // Arbitration lost flag clear
        als = 1;                     // When Arbitration lost,SDA "HiZ" disable
       }
       swc = 1;                      // SCL"L"Hold enable
       SiRIC = 0x01;                 // Receive int. enable
       if(f_ep == 0 || f_sr == 0){   // When EEPROM read mode address set,Pointer no touch
        ++iic_pointer;               // Pointer moved
      }
      ++iic_index;
     }
    }
     else{                           // = When slave =
      UiMR = 0x0a;                   // Ext. clock sel.
      stspsel = 0;                   // SI/O output enable
      ackc = 0;                      // ACKoutput disable
      swc9 = 0;                      // SCL"L"Hold3 disable
      swc = 1;                       // SCL"L"Hold enable
      ++iic_pointer;                 // Pointer moved
     ++iic_index;
     SiRIC = 0x01;
    }
   }
    else{                            // = On and after the 2nd time =
     if(f_ms == 1){                  // = When master =
      if(iic_length == iic_index){   // = When last data =
       if(f_sr == 0){                // = When receive =
       --iic_pointer;
       *iic_pointer = temp.byte.byte0;
       ++iic_pointer;
       if(abl == 1){             // Arbitration lost found(Self-uint send NACK < other-master
                                 // send ACK)
        ackd = 1;                // Other-master still transmitting,So transmit finish and stop
                                 // condition no generate
        ackc = 1;
        UiMR = 0x0a;             // Ext. clock sel
        if(f_ep == 0){
#if uarti == 0
         iic0_master_end(0x10); // Master normally finish
#elif uarti == 1
         iic1_master_end(0x10); // Master normally finish
#elif uarti == 2
         iic2_master_end(0x10); // Master normally finish
#elif uarti == 3
         iic3_master_end(0x10); // Master normally finish
#elif uarti == 4
         iic4_master_end(0x10); // Master normally finish
#endif
        }
        else{
```

```
#if uarti == 0
       iic0_master_end(0x20); // Master normally finish
#elif uarti == 1
       iic1_master_end(0x20); // Master normally finish
#elif uarti == 2
       iic2_master_end(0x20); // Master normally finish
#elif uarti == 3
       iic3_master_end(0x20); // Master normally finish
#elif uarti == 4
       iic4_master_end(0x20); // Master normally finish
#endif
      }
      iic_mode = 0x00;
      iic_index = 0x00;
     }
     else{
      UiSMR4 = 0x04;
      UiSMR4 = 0x3c;          // Stop condition generate
      if(f_ep == 0){
#if uarti == 0
       iic0_master_end(0x10); // Master normally finish
#elif uarti == 1
       iic1_master_end(0x10); // Master normally finish
#elif uarti == 2
       iic2_master_end(0x10); // Master normally finish
#elif uarti == 3
       iic3_master_end(0x10); // Master normally finish
#elif uarti == 4
       iic4_master_end(0x10); // Master normally finish
#endif
      }
      else{
#if uarti == 0
       iic0_master_end(0x20); // Master normally finish
#elif uarti == 1
       iic1_master_end(0x20); // Master normally finish
#elif uarti == 2
       iic2_master_end(0x20); // Master normally finish
#elif uarti == 3
       iic3_master_end(0x20); // Master normally finish
#elif uarti == 4
       iic4_master_end(0x20); // Master normally finish
#endif
      }

     }
    }
    else{
     if(f_ep == 0){
#if uarti == 0
      iic0_master_end(0x00);  //
#elif uarti == 1
      iic1_master_end(0x00);  //
#elif uarti == 2
      iic2_master_end(0x00);  //
#elif uarti == 3
      iic3_master_end(0x00);  //
#elif uarti == 4
      iic4_master_end(0x00);  //
#endif
```

```
        UiSMR4 = 0x04;          // Stop condition generate
        UiSMR4 = 0x3c;

       }
       else{
        iic_mode = 0x31;
        iic_length = iic_eeplen;
        iic_rw = 1;
        UiSMR4 = 0x02;          // Restart condition generate
        UiSMR4 = 0x3a;

       }
      }
      UiSMR2 = 0x03;            //
      SiRIC = 0x01;             // receive int. enable
      SiTIC = 0x00;             // transfer int. disable
     }
     else{                      // = When continue =
      if(f_sr == 0){            // = When receive =
       --iic_pointer;
       *iic_pointer = temp.byte.byte0;
       ++iic_pointer;
       ++iic_pointer;           // Pointer moved
       ++iic_index;
       if(iic_length == iic_index){
        UiTB = 0x01ff;          // Send NACK
       }
       else{
        UiTB = 0x00ff;          // Send ACK
       }
       swc = 1;
       SiRIC = 0x01;            // receive int. enable
      }
      else{                     // = When transfer =
       if(temp.bit.b8 == 1){    // = When NACK found =
#if uarti == 0
        iic0_master_end(0x03);  // When N byte , NACK found finish
#elif uarti == 1
        iic1_master_end(0x03);  // When N byte , NACK found finish
#elif uarti == 2
        iic2_master_end(0x03);  // When N byte , NACK found finish
#elif uarti == 3
        iic3_master_end(0x03);  // When N byte , NACK found finish
#elif uarti == 4
        iic4_master_end(0x03);  // When N byte , NACK found finish
#endif
        als = 0;
        UiSMR4 = 0x04;          // Stop condition generate
        UiSMR4 = 0x3c;          //
        UiSMR2 = 0x01;
        SiRIC = 0x01;           // receive int. enable

       }
```

```
        else{                      // = When ACK found =
         temp.byte.byte0 = *iic_pointer;
         temp.byte.byte1 = 0x01;  // NACK-data set
         UiTB = temp.all;
         abl = 0;                  // Arbitration lost flag clear
         als = 1;                  // When Arbitration lost,SDA "HiZ" disable
         swc = 1;
         SiRIC = 0x01;             // receive int. enable
         ++iic_pointer;            // Pointer moved
         ++iic_index;
        }
       }
      }
     }
     else{                        // = When slave =
      UiMR = 0x0a;
      if(f_sr == 1){              // = When transfer =
       if(temp.bit.b8 == 1){      // = When NACK found =
        ackd = 1;                 // Output NACK-data
        ackc = 1;                 // NACK-data output enable
        swc9 = 0;                 // SCL"L"Hold3 disable
        SiSIC = 0x00;             // stop int. disable
        SiRIC = 0x01;             // Receive int. enable
#if uarti == 0
        iic0_slave_end(0x10,iic_index); // Slave transfer complete
#elif uarti == 1
        iic1_slave_end(0x10,iic_index); // Slave transfer complete
#elif uarti == 2
        iic2_slave_end(0x10,iic_index); // Slave transfer complete
#elif uarti == 3
        iic3_slave_end(0x10,iic_index); // Slave transfer complete
#elif uarti == 4
        iic4_slave_end(0x10,iic_index); // Slave transfer complete
#endif
        iic_index = 0x00;  //
        iic_mode = 0x00;   // Slave mode set
       }
       else{               // = When ACK =
        swc9 = 0;          // SCL"L"HOLD3 disable
        swc = 1;           // SCL"L"HOLD enable
        ++iic_pointer;     // Pointer moved
        ++iic_index;
        SiRIC = 0x01;
       }
      }
      else{                // = When receive =
       --iic_pointer;
       *iic_pointer = temp.byte.byte0;
       ++iic_pointer;
       UiTB = 0x00ff;      // Send ACK
       swc9 = 0;           // SCL"L"HOLD3 disable
       swc = 1;            // SCL"L"HOLD enable
       ++iic_pointer;      // Pointer moved
       ++iic_index;
       SiRIC = 0x01;
      }
     }
    }
}
```