M16C/60 Series

Clock-Synchronous Serial I/O

1. Abstract

The following article introduces Clock-Synchronous Serial I/O of M16C/60 series.

2. Introduction

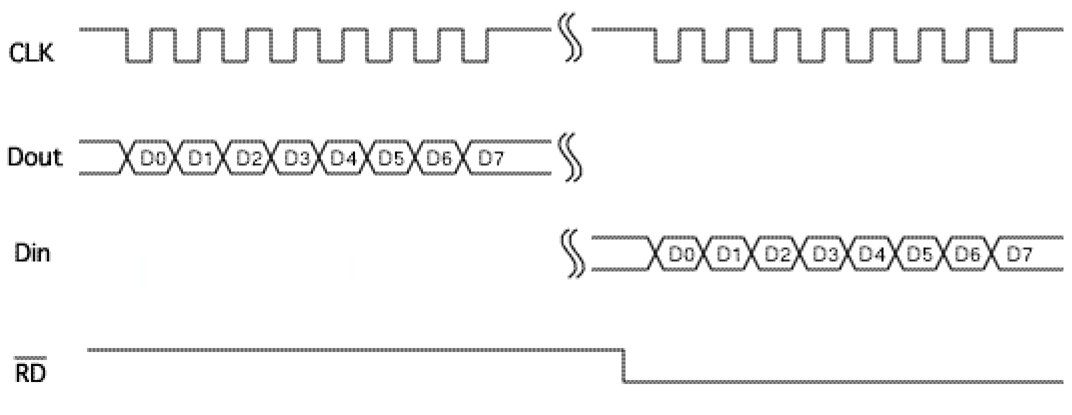
The explanation of this issue is applied of the following condition.

Applicable MCU: M16C/60 Series

3.0 Contents

3.1 Outline of Clock-Synchronous Serial I/O

Clock-Synchronous Serial I/O uses one clock line, one data line (2 lines in bi-directional transfers), and one or two control lines (depending on the system). The data is transferred while synchronizing the data line to the clock line bit by bit. Clock-Synchronous Serial I/O enables data transfer with less wiring than parallel transfers.



· Clock rises after setting data.

· Data is output from CPU when $\overline{RD}$ signal is "H".

· Data is input from the destination when $\overline{RD}$ signal is "L".

Figure1. Clock-Synchronous Serial Timing Diagram

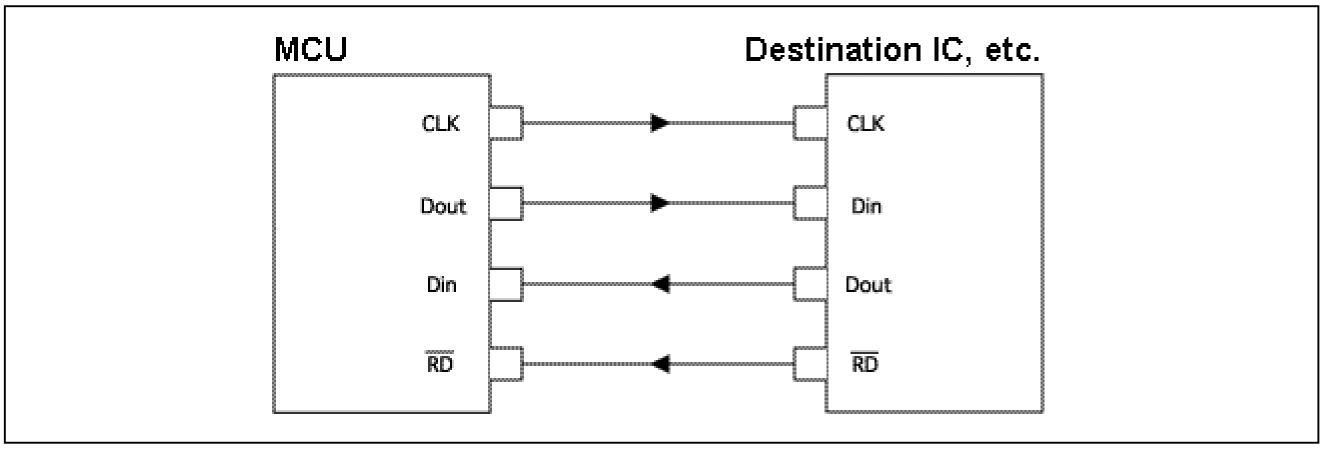


Figure2. Clock-Synchronous Serial Interface Example

3.2 Features of M16C/60 Series Clock-Synchronous Serial I/O

M16C/60 Series Clock-Synchronous Serial I/O offers sophisticated functions to communicate with various external devices as make software development easy.

Note: Not all serial I/Os in the M16C/60 Series are equipped with all of the following functions. In addition, certain functions cannot be used in combination with other functions

- **Programmable Data Input/Output Timing based on Transfer Clock**
  The I/O timing of the receive/transmit data can be selected from either the rising edge or the falling edge of the transfer clock.
- **Programmable Data Input/Output Order**
  The I/O order of the receive/transmit data can be selected from either MSB (bit 7) first or LSB (bit 0) first.

- **Reversible Logic for Input/Output Data (for applicable serial I/O only)**
  When writing the transmit data and when reading the received data, the data can be reversed.

- **Reversible Polarity for Data I/O Pins (for applicable serial I/O only)**
  The level of the transmit data output pin and the receive data input pin can be reversed.

- **Selectable Transmit Interrupt Factors**
  The generation timing for the transmit interrupt request can be selected from either when the transmit buffer register becomes empty or when the transmit is completed.

- **CTS/RTS Function**
  The CTS function can be used during data transmit or the RTS function can be used during data receive

- **Receive Error Detect Function**
  When the serial I/O fails to receive the data, it sets the error flag, allowing the user to recognize the error (overrun error only).

- **Transfer Clock Output from Dual Pins (for applicable serial I/O only)**
  M16C/60 Series is equipped with 2 transfer clock output pins and can switch the 2 output pins. Therefore, data can be transferred to two devices on the same data line.

- **Continuous Receive Mode**
  In addition to the normal method of writing dummy data to the transmit buffer register after each data receive in order to enable the next receive, you can select the continuous receive mode, which enables the next data receive, simply by reading the received data from the receive buffer register after writing dummy data to the transmit buffer register only once.

Comparison of Clock-Synchronous Serial I/O Functions by Group/Channel

Table1. M16C/61 Group Function

Function/Group Channel	M16C/61		
	Clock Sync./Async.		
	UART0	UART1	UART2
Clock Polarity Select	Available	Available	Available
LSB First/ MSB First Select	Available	Available	Available
Transfer Clock Output from Dual Pins	---	Available	---
CTS/RTS Function	Available	Available	Available
Serial Data Logic Reverse	---	---	Available
RxD/TxD I/O Polarity Reverse	---	---	Available
Continuous Receive Mode	Available	Available	Available
Transmit Interrupt Factor Select	Available	Available	Available

Table2. M16C/62 Group Function

Function/Group Channel	M16C/62				
	Clock Sync./Async			Clock Sync.	
	UART0	UART1	UART2	SIO3	SIO4
Clock Polarity Select	Available	Available	Available	---	---
LSB First/ MSB First Select	Available	Available	Available	Available	Available
Transfer Clock Output from Dual Pins	---	Available	---	---	---
CTS/RTS Function	Available	Available	Available	---	---
Serial Data Logic Reverse	---	---	Available	---	---
RxD/TxD I/O Polarity Reverse	---	---	Available	---	---
Continuous Receive Mode	Available	Available	Available	---	---
Transmit Interrupt Factor Select	Available	Available	Available	---	---

Note: There are some differences concerning settings and operations between the two clock-synchronous only modules (SI03, SI04) and the three clock sync./async. modules (UART0 to UART2), as shown below.

[Differences in Settings and Operations]

1. To initiate transmit/receive operation:
   Clock-synchronous only: Data transfer starts by writing the data to the SI/O transmission/reception register.
   Clock sync./async.: The transmit enable bit and receive enable bit must be set before writing the data.
2. Transmit/receive completion:
   Clock-synchronous only: Completion is determined by the interrupt request bit only.
   Clock sync./async.: Completion is determined by the interrupt request bits, transmit register empty flag, and the receive complete flag.
3. Continuous transmit/receive:
   Clock-synchronous only: The next transmit data cannot be set and the next data cannot be received until after the previous data has been transmitted and received.
   Clock sync./async.: Once the transmit buffer register or receive register is cleared, the next transmit data can be set or the next data can be received.
   (UART0 to UART2 have two registers for transmit use and two registers for receive use. In comparison, SIO3 and SIO4 have only one register for each module.)

3.3 Clock-Synchronous Serial I/O Operation

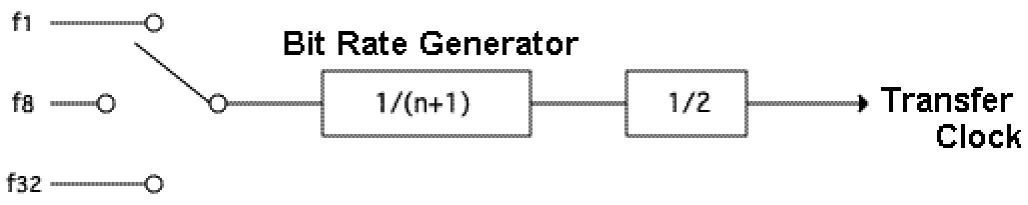
In M16C/60 Series clock-synchronous serial I/O, you can set the mode and transfer speed and enable transfers by setting each control register. Transfers can be initiated automatically by writing transmit data or dummy data to the transmit buffer register in transmit or receive operations, respectively. Completion of a receive or transmit can be determined by the transmit register empty flag or by the receive complete flag in the control registers.

Internal/External Clock Selection

When outputting a transfer clock to an external device, select the internal clock. A transfer is started by setting the transfer speed (described in detail later), enabling the transfer, then writing transmit data (for a transmit) or dummy data (for a receive) to the transmit buffer register.
When inputting a transfer clock from an external device, select the external clock. After enabling the transfer, prepare for the external clock to be input by writing transmit data (for a transmit) or dummy data (for a receive) to the transmit buffer register.

Setting the Transfer Speed

Select the transfer speed while operating with the internal clock. The transfer speed is determined by the count source (f1, f8, f32) and the set value (0 to 255) of the bit rate generator. (The output from the bit rate generator is divided in half again and becomes the transfer clock.)



Note: Clocks f1, f8 and f32 are the results of the main clock divided into 1/1, 1/8 and 1/32, respectively. n is the value set in the bit rate generator.

Figure3. Baud Rate: fi/2(n+1) (i = 1, 8, 32)

Transfer Speed Conditions

The transfer speed must meet the serial I/O timing requirements indicated under the electrical characteristics in the user's manual.

Table3. M16C/62 Group Serial I/O Timing Requirements (Vcc=5V)

Symbols	Parameters	Standards		Unit
		Min.	Max.	
$t_{c(CK)}$	CLKi input cycle time	200		ns
$t_{w(CKH)}$	CLKi input HIGH pulse width	100		ns
$t_{w(CKL)}$	CLKi input LOW pulse width	100		ns
$t_{d(C-Q)}$	TxDi output delay time		80	ns
$t_{h(C-Q)}$	TxDi hold time	0		ns
$t_{su(D-C)}$	RxDi input setup time	30		ns
$t_{h(C-D)}$	RxDi input hold time	90		ns

For example, when calculating the speed according to fi/2(n+1), the transfer speed will be a maximum of 8Mbps when Vcc=5V and f(XIN)=16MHz (when fi=fl and n=0). However, the minimum input cycle time of the clock, as defined in the timing requirements, is 200ns, which 8Mbps does not satisfy. In this case, the maximum value will be 4Mbps (n=1).
In regards to the connected device, please keep in consideration the maximum TxD output delay time while transmitting and the minimum RxD input setup time while receiving.

Transfer Initiation

To initiate the transmit, set the transmit enable bit then write the transmit data to the transmit buffer register. To receive, set both the transmit enable bit and the receive enable bit, and then write dummy data to the transmit buffer register.

Note: In the M16C/62 Group, TxD2 pin1 is N-channel open-drain output and must be pulled up.

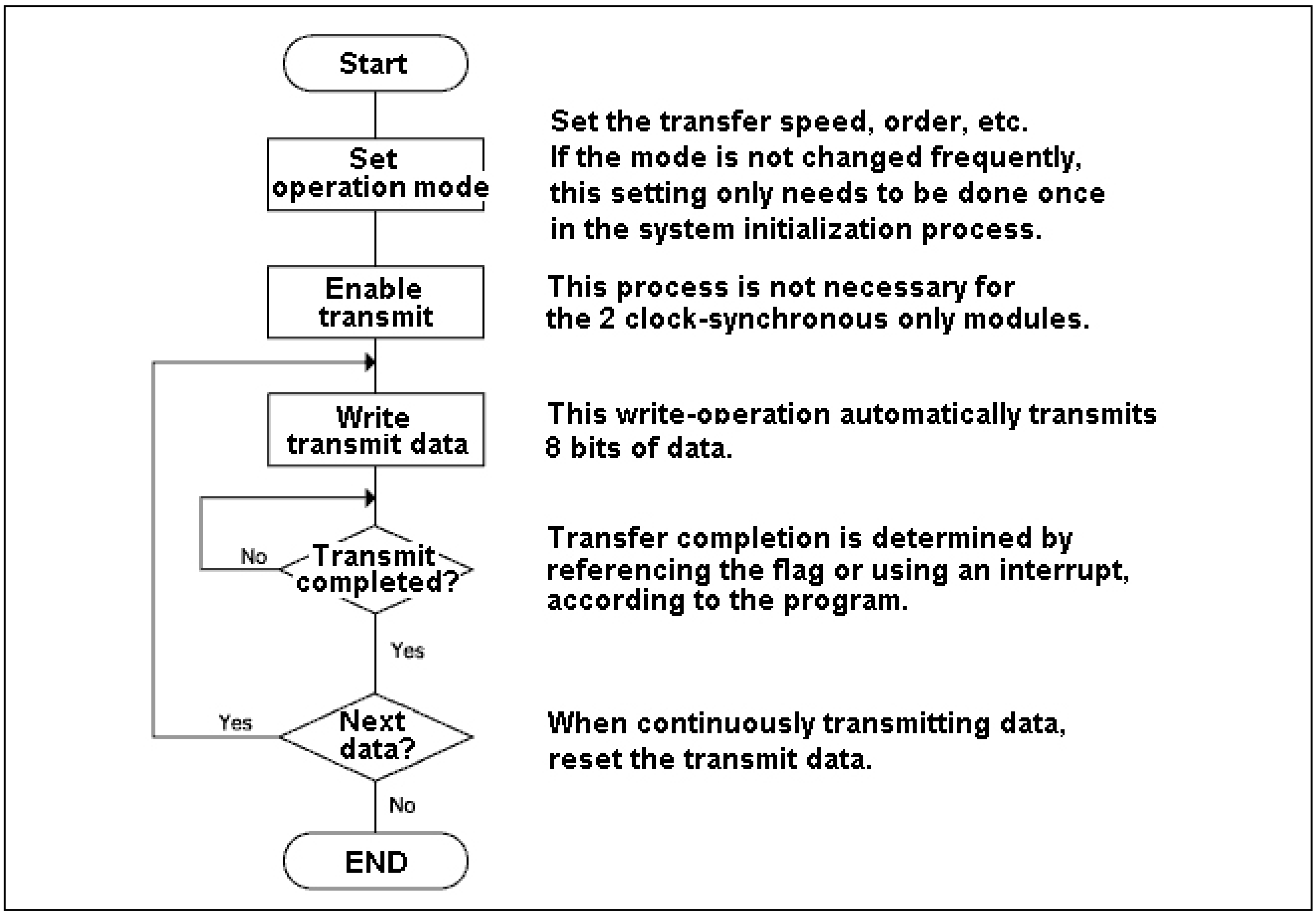

Start
Set operation mode
Set the transfer speed, order, etc.
If the mode is not changed frequently, this setting only needs to be done once in the system initialization process.
Enable transmit
This process is not necessary for the 2 clock-synchronous only modules.
Write transmit data
This write-operation automatically transmits 8 bits of data.
Transmit completed?
No
Yes
Transfer completion is determined by referencing the flag or using an interrupt, according to the program.
Next data?
Yes
No
When continuously transmitting data, reset the transmit data.
END

Figure4. Flowchart (Transmit)

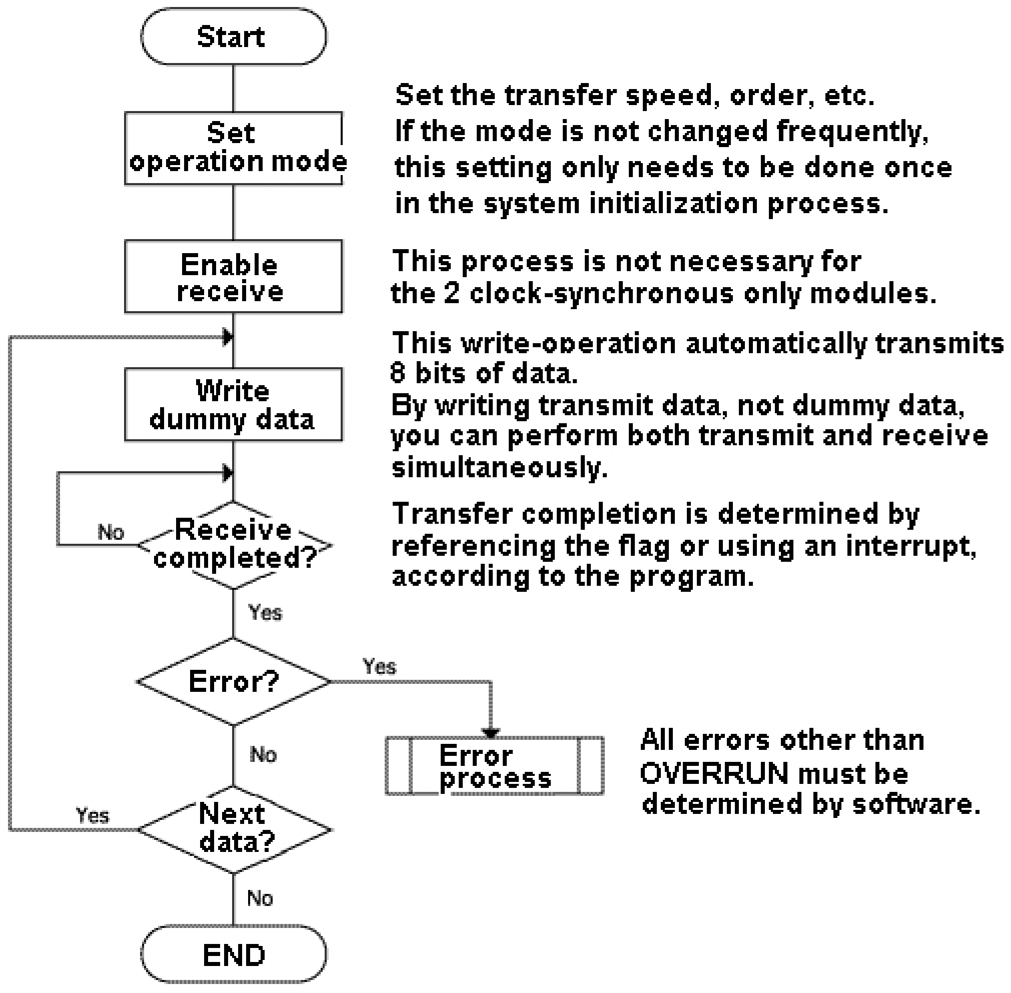

Start
Set operation mode
Set the transfer speed, order, etc.
If the mode is not changed frequently, this setting only needs to be done once in the system initialization process.
Enable receive
This process is not necessary for the 2 clock-synchronous only modules.
Write dummy data
This write-oeration automatically transmits 8 bits of data.
By writing transmit data, not dummy data, you can perform both transmit and receive simultaneously.
No
Receive completed?
Transfer completion is determined by referencing the flag or using an interrupt, according to the program.
Yes
Error?
Yes
Error process
All errors other than OVERRUN must be determined by software.
No
Yes
Next data?
No
END

Figure5. Flowchart (Receive)

3.4 Continuous Data Transmit Using an Interrupt

When setting and transmitting continuous data in an interrupt process, you can minimize the void between each data by setting the transmit interrupt factor to "transmit buffer empty".

The 8-bit data written to the transmit buffer register will be output from the TxD pin after being transferred to the transmit register. When the interrupt factor is set to "transmit completed" by the transmit interrupt cause select bit, the interrupt request is generated when the output of 8-bit data is completed. On the other hand, when the factor is set to "transmit buffer empty", the interrupt request will be generated when data is transferred from the transmit buffer register to the transmit register, and the next data can be set in the transmit buffer register before the current transmit is complete.

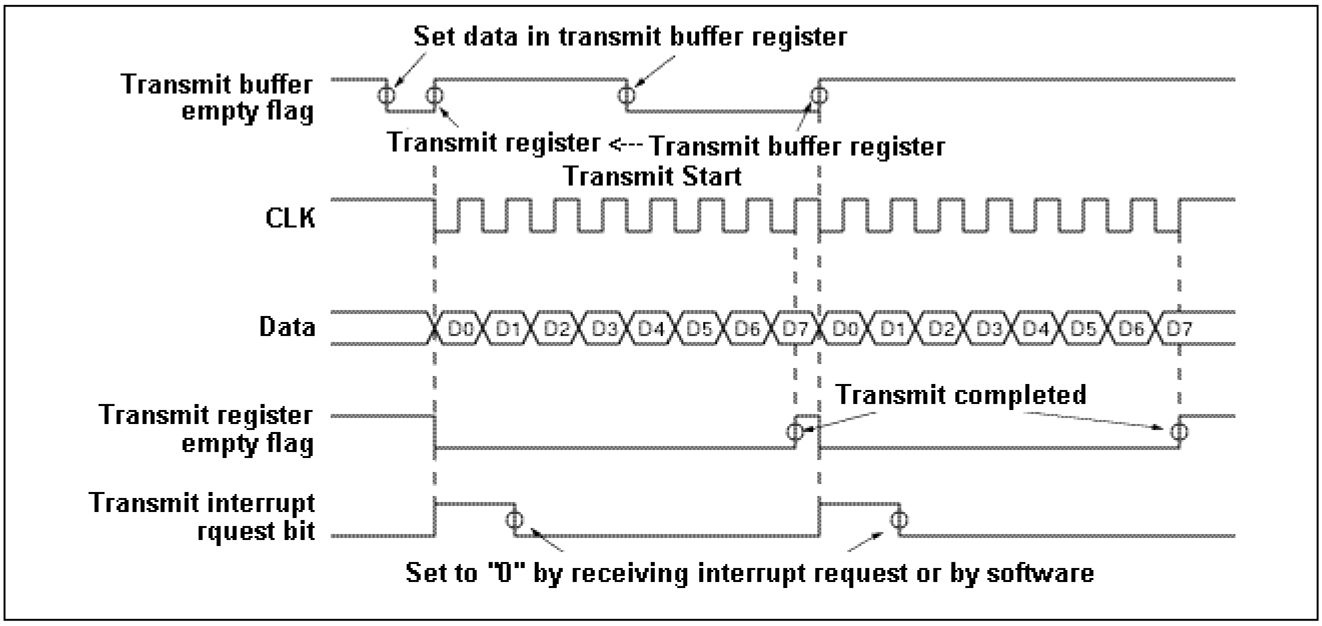

Set data in transmit buffer register
Transmit buffer empty flag
Transmit register <--- Transmit buffer register
Transmit Start
CLK
Data
D0 D1 D2 D3 D4 D5 D6 D7 D0 D1 D2 D3 D4 D5 D6 D7
Transmit register empty flag
Transmit completed
Transmit interrupt rquest bit
Set to "0" by receiving interrupt request or by software

Figure6. Operation Timing Diagram for Interrupt Factor Setting "Transmit Buffer Empty"

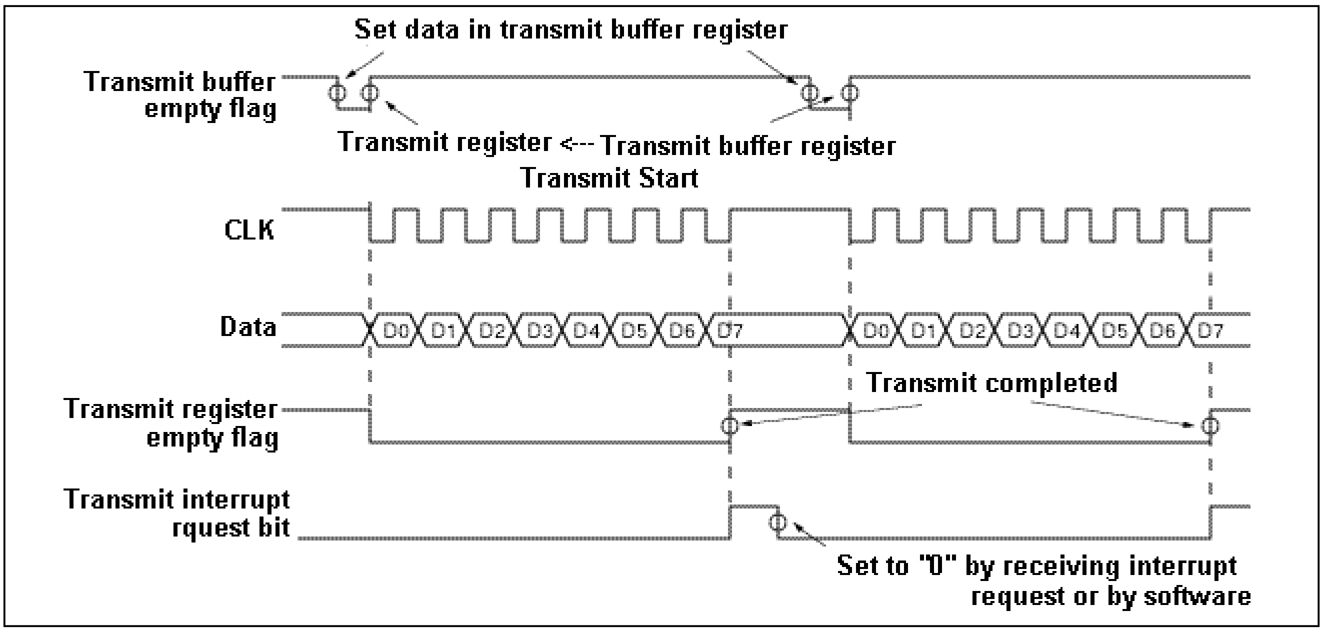

Set data in transmit buffer register
Transmit buffer empty flag
Transmit register <--- Transmit buffer register
Transmit Start
CLK
Data
D0 D1 D2 D3 D4 D5 D6 D7 D0 D1 D2 D3 D4 D5 D6 D7
Transmit register empty flag
Transmit completed
Transmit interrupt rquest bit
Set to "0" by receiving interrupt request or by software

Figure7. Operation Timing Diagram for Interrupt Factor Setting "Transmit Completed"

Transmit Process Example for Using an Interrupt

In a main routine, enable the transmit. In an interrupt process, transmit data.

[Outline]

1. Set each clock-synchronous serial I/O function.
2. Clear the transmit interrupt request bit to be used, set the interrupt enable flag.
3. To start the transmit, set the transmit enable bit and write the first byte of transmit data to the transmit buffer register.
4. Write 1-byte transmit data at a time to the transmit buffer register whenever transmit interrupt is generated.
5. Disable the transmit interrupt after writing the last data to the transmit buffer register.

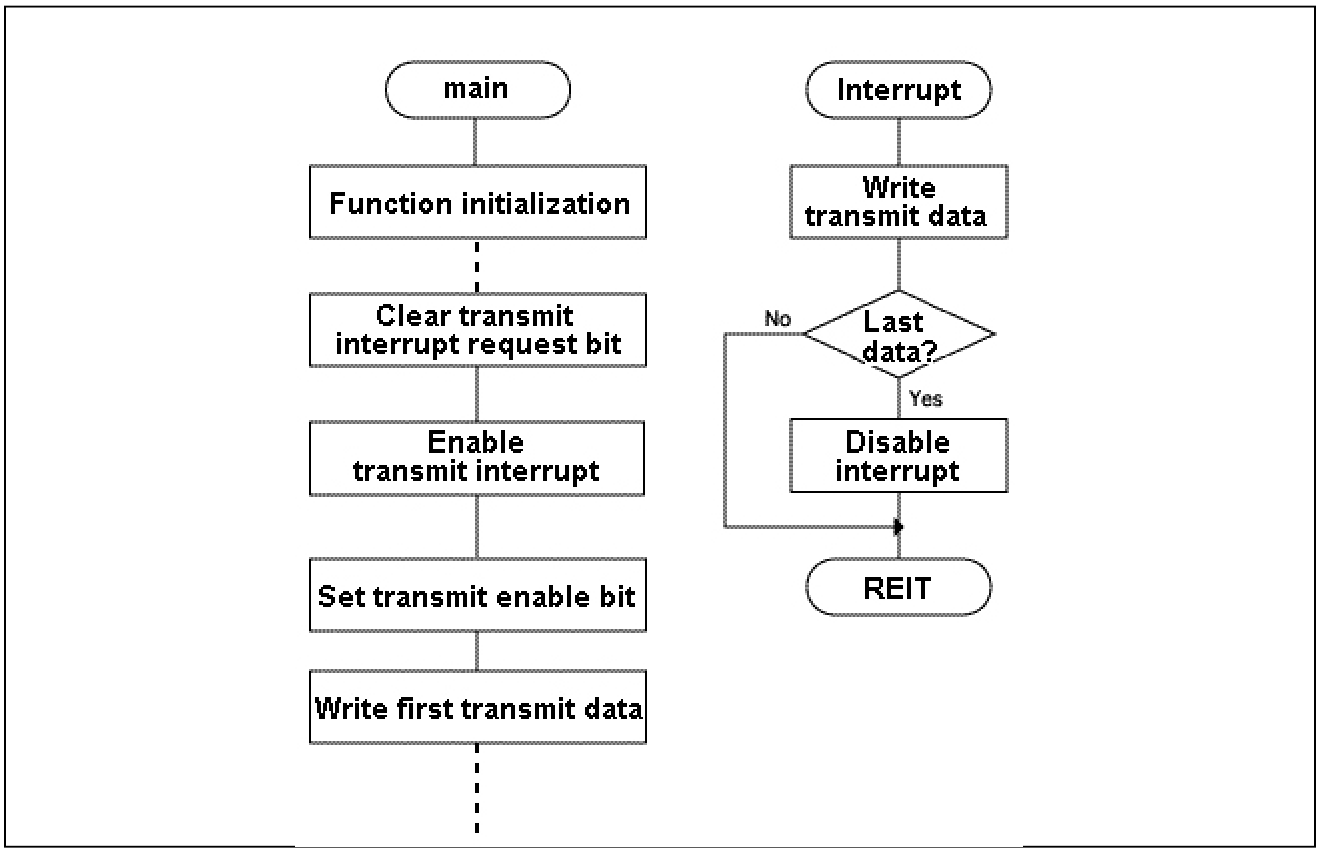


Figure8. General Flowchart: Transmit Process Example for Using an Interrupt

Sample Program

Transmitting data contained in addresses 500h to 503h.

```
                                    .
                                    .
                                    .
;====================   Main Program    ==================
MAIN:
    JSR         INIT             ; Go to UART0 initial settings
    FSET        I                ; Enable interrupt
                                 ;
    MOV.B       #01h,U0C1        ; Enable transmit
    MOV.B       [A0],U0TB        ; Write transmit data
                                    .
                                    .
                                    .
;=============   UART0 Function Initial Settings    ===========
INIT:
    MOV.B       #01h,U0MR        ; Select internal clock
    MOV.B       #10h,U0C0        ; Disable CTS function, select count source f1
    MOV.B       #00h,UCON        ; UART0 transmit interrupt cause = transmit buffer empty
    MOV.B       #0FFh,U0BRG      ; Transfer speed = f1/256 x 1/2
    MOV.W       #500h,DATA       ; Set position for reading transmit data
    MOV.W       #500h,A0
    MOV.B       #3,CNT           ; Set number of transfers (4-1)
    MOV.B       #01h,S0TIC       ; Set UART0 transmit interrupt priority level
INT_END:
    RTS
                                    .
                                    .
                                    .
;===========   UART0 Transmit Interrupt Process    ===========
S0T_INT:
    PUSH.W      A0               ; Save A0
    INC.B       DATA             ; Set position for reading next transmit data
    MOV.W       DATA,A0
    MOV.B       [A0],U0TB        ; Write next transmit data
    DEC.B       CNT              ; Count down number of transfers
    JNZ         S0T_INT_1        ; Keep interrupt enabled if more data to transmit
                                 ;
    MOV.B       #00h,S0TIC       ; If last data, disable transmit interrupt
S0T_INT_1:
    POP.W       A0               ; Restore A0
    REIT
                                    .
                                    .
                                    .
```

3.5 Continuous Receive Mode

Data can be received continuously in the continuous receive mode without writing dummy data to the transmit buffer register. (You must write dummy data to the transmit buffer register or read dummy data from the receive buffer register in order to start the receive operation.)

In the normal receive method, the next data receive is enabled by writing a dummy data to the transmit buffer register after each byte of data received. However, in the continuous receive mode, the next receive is enabled when the received data is read out from the receive buffer register. It is not necessary to write to the transmit buffer register. (In other words, by reading out the receive buffer register, dummy data is automatically written to the transmit buffer register.)

Note 1: Do not write dummy data to the transmit buffer register when in the continuous receive mode, except for the first time in order to initialize the receive operation.

Note 2: The value output by the TxD pin is undetermined.

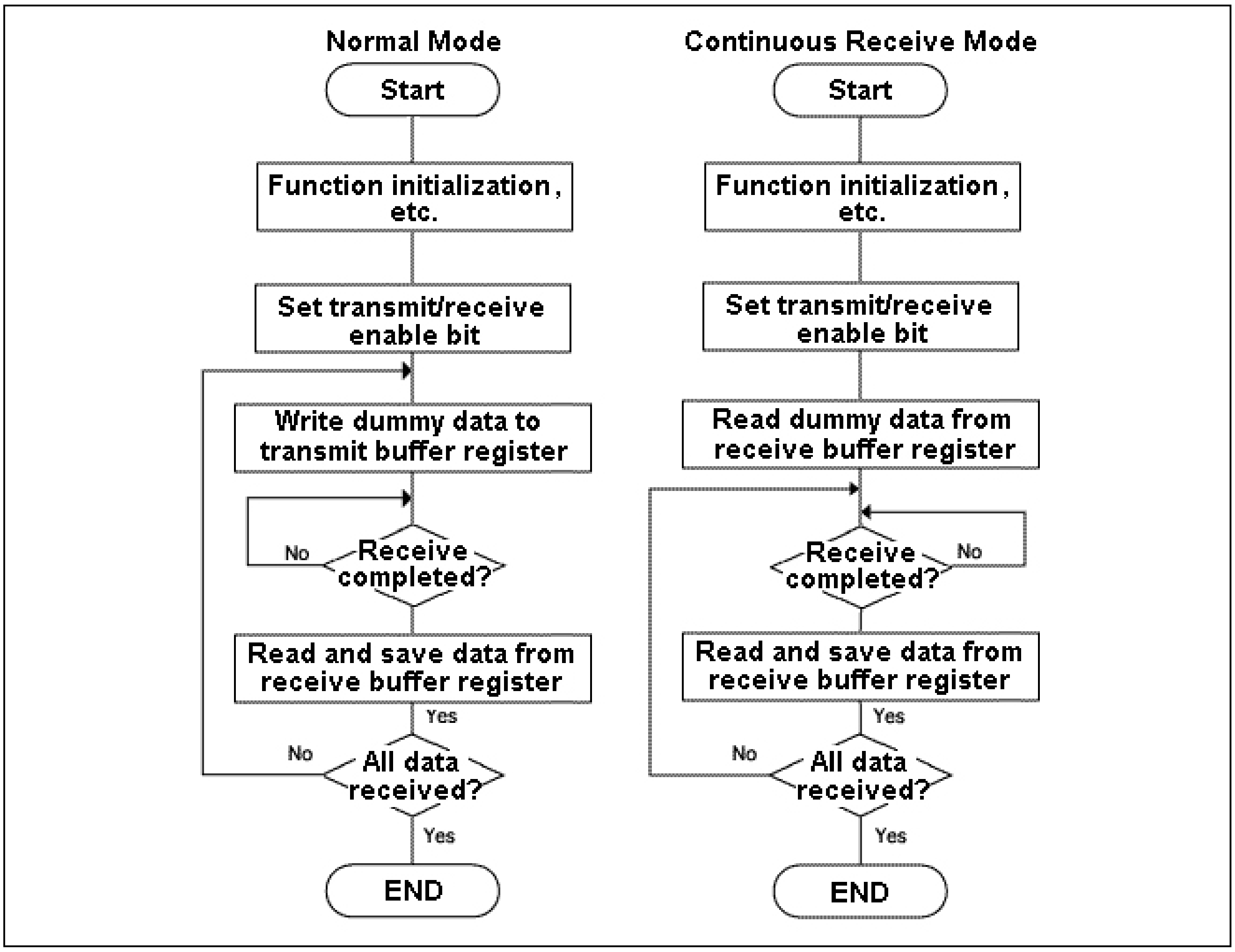

Normal Mode
Start
Function initialization, etc.
Set transmit/receive enable bit
Write dummy data to transmit buffer register
Receive completed?
No
Yes
Read and save data from receive buffer register
All data received?
No
Yes
END
Continuous Receive Mode
Start
Function initialization, etc.
Set transmit/receive enable bit
Read dummy data from receive buffer register
Receive completed?
No
Yes
Read and save data from receive buffer register
All data received?
No
Yes
END

Figure9. Receive Operation Comparison

Process Example in Continuous Receive Mode

In a main routine, enable the receive. In an interrupt process, read out data.

[Outline]

1. Enable the continuous receive mode and set each clock-synchronous serial I/O function.
2. Clear the receive interrupt request bit to be used, set the interrupt enable flag.
3. To start the receive operation, set the transmit enable bit and receive enable bit, and read a dummy data from the receive buffer register.
4. Read out the received data from the receive buffer register whenever receive interrupt is generated.
5. Disable the receive interrupt after reading the last data from the receive buffer register.

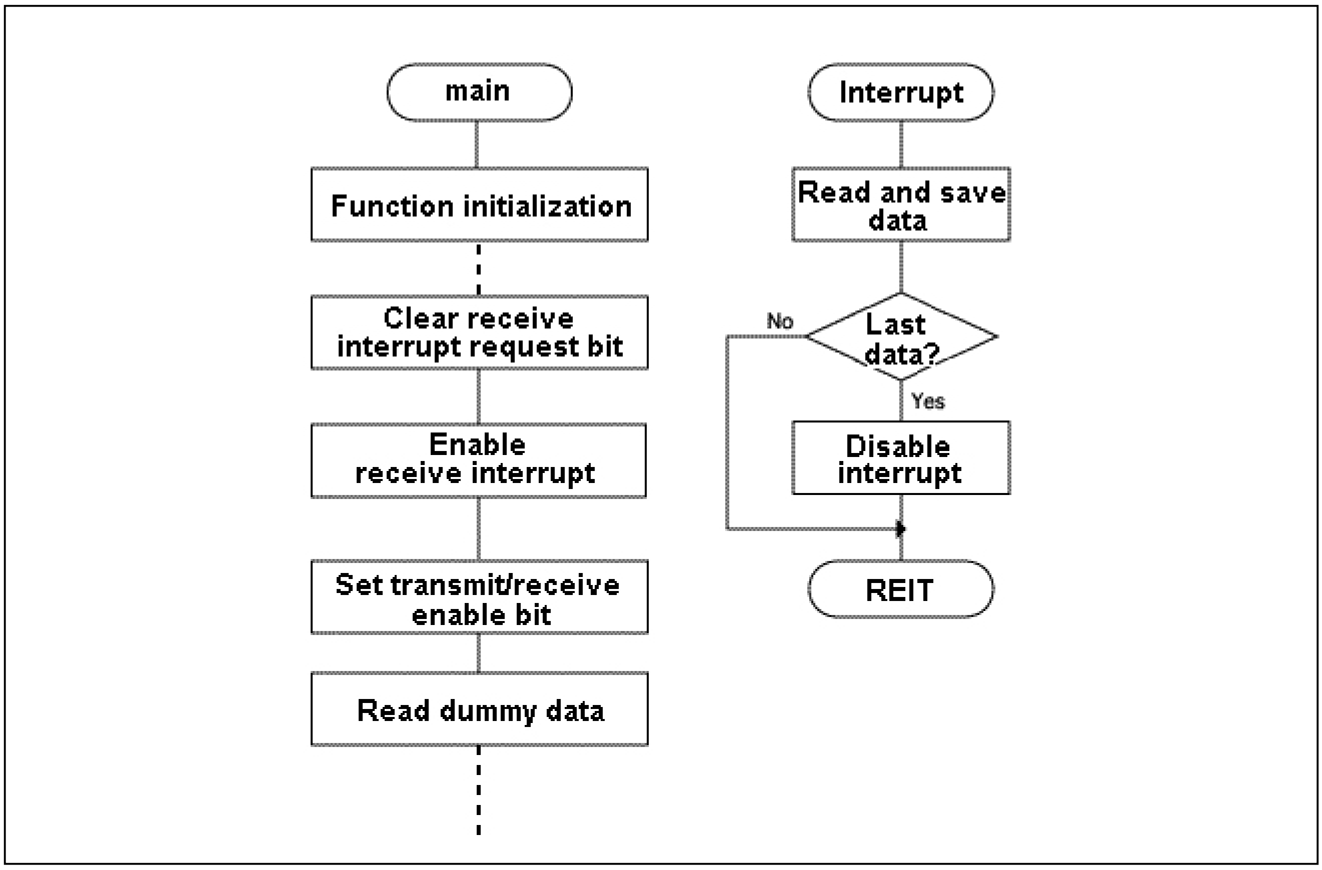


Figure10. General Flowchart: Process Example in Continuous Receive Mode

Sample Program

Store received data in addresses 500h to 503h.

```
                                        .
                                        .
                                        .
;====================  Main Program   ==================
MAIN:
    JSR         INIT                ; Go to UART0 initial settings
    FSET        I                   ; Enable interrupt
                                    ;
    MOV.B       #05h,U0C1           ; Enable receive
    MOV.B       U0RB,R0L            ; Read dummy data = start receive
                                        .
                                        .
                                        .
;============  UART0 Function Initial Settings   ============
INIT:
    MOV.B       #09h,U0MR           ; Select external clock
    MOV.B       #10h,U0C0           ; Disable RTS function
    MOV.B       #04h,UCON           ; Enable UART0 continuous receive mode
    MOV.W       #500h,DATA          ; Set position for storing received data
    MOV.B       #4,CNT              ; Set number of transfers
    MOV.B       #01h,S0RIC          ; Set UART0 receive interrupt priority level
INT_END:
    RTS
                                        .
                                        .
                                        .
;===========  UART0 Receive Interrupt Process    ===========
S0R_INT:
    PUSH.W      A0                  ; Save A0
    MOV.W       DATA,A0
    MOV.B       U0RB,[A0]           ; Read received data
    INC.B       DATA                ; Set position for storing next receive data
    DEC.B       CNT                 ; Count down number of transfers
    JNZ         S0R_INT_1           ; Keep interrupt enabled if more data to receive
                                    ;
    MOV.B       #00h,S0RIC          ; If last data, disable receive interrupt
S0R_INT_1:
    POP.W       A0                  ; Restore A0
    REIT
                                        .
                                        .
                                        .
```

3.6 Transfer Clock Dual-Pin Output

By selecting the transfer clock dual-pin output and switching two transfer clock output pins, you can transfer data to two external ICs using a common data line on a time-sharing basis. This will reduce the required number of pins as well as reduce on-board wiring.

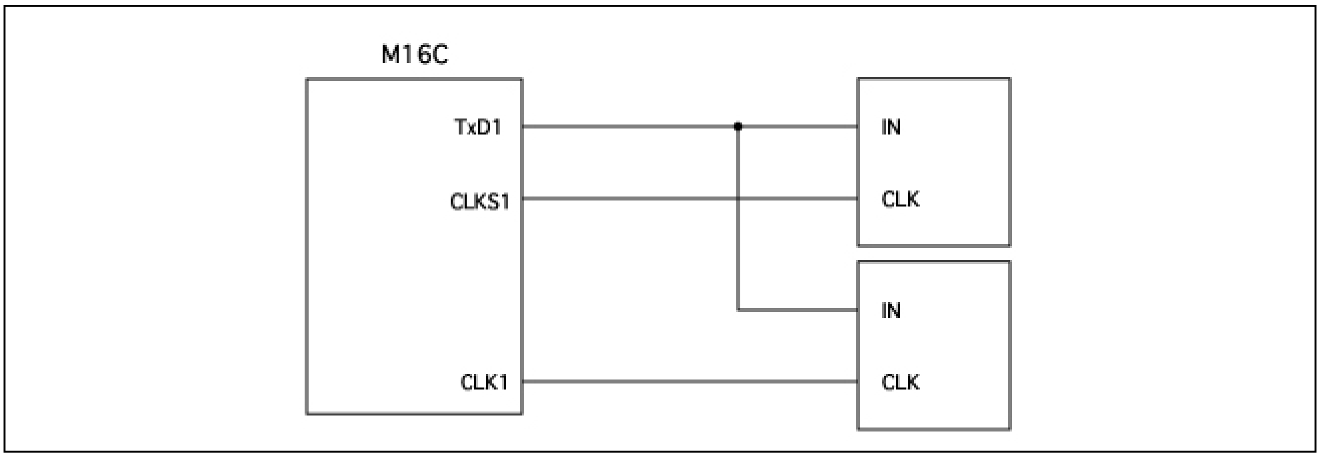


Figure11. Connection Example (internal clock, data transmit only)

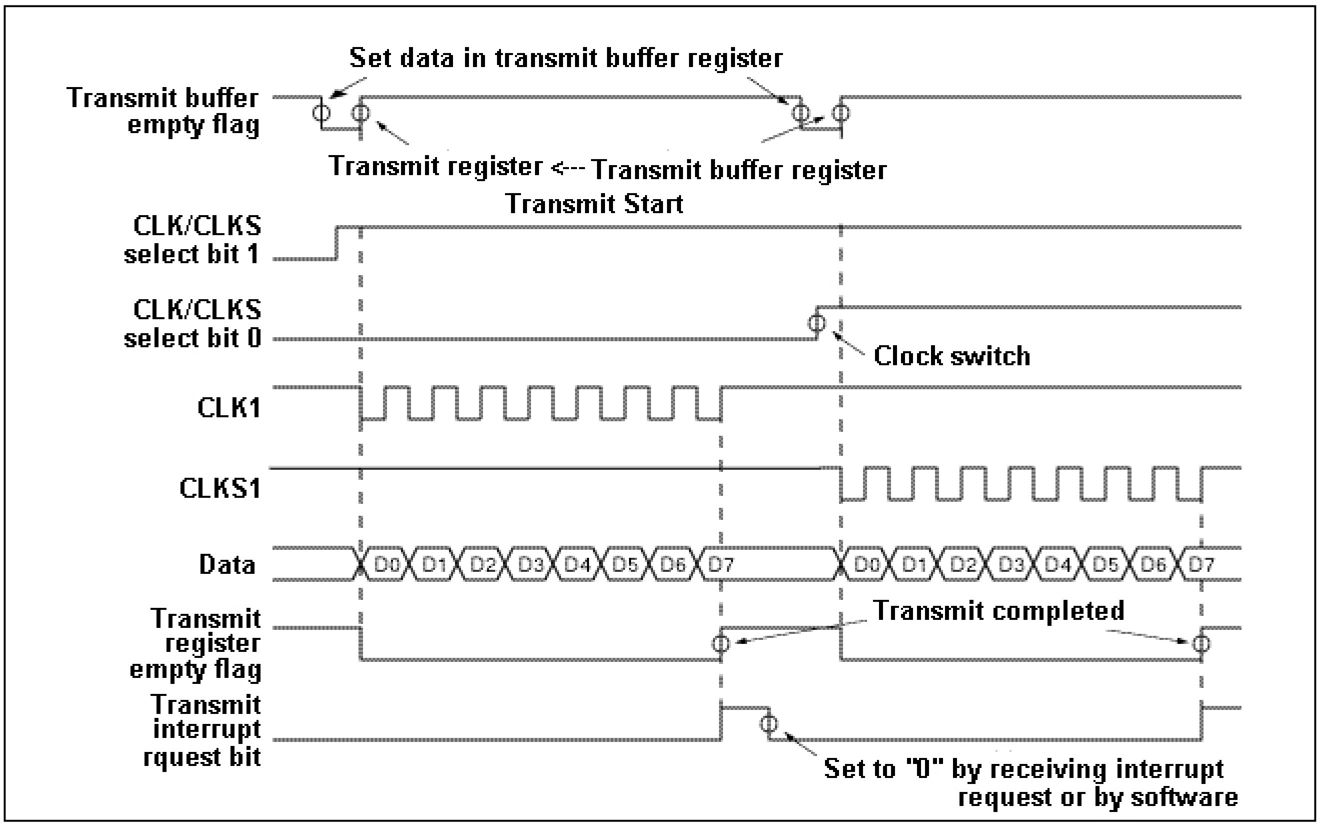

Set data in transmit buffer register
Transmit buffer empty flag
Transmit register <--- Transmit buffer register
Transmit Start
CLK/CLKS select bit 1
CLK/CLKS select bit 0
Clock switch
CLK1
CLKS1
Data
D0
D1
D2
D3
D4
D5
D6
D7
Transmit register empty flag
Transmit completed
Transmit interrupt rquest bit
Set to "0" by receiving interrupt request or by software

Figure12. Operation Timing Diagram of Transferring Clock Dual-Pin Output

Process Example for Transfer Clock Dual-Pin Output

1. In a main routine, enable the transmit and select CLKS1 as the transfer clock output pin.
2. Switch the transfer clock output pin (->CLK1 -> CLKS1 -> CLK1...) each time a transmit interrupt is generated and transmit data to two external ICs by turns.

[Outline]

1. Select the transfer clock dual-pin output and set each clock-synchronous serial I/O function.
2. Clear the transmit interrupt request bit to be used, set the interrupt enable flag.
3. To start the transmit, set the transmit enable bit and write the first byte of transmit data to the transmit buffer register. (The first data is sent to the CLKS1 side.)
4. Switch CLK1 and CLKS1 and write 1-byte transmit data at a time to the transmit buffer register, whenever transmit interrupt is generated.
5. Disable the transmit interrupt after writing the last data to the transmit buffer register.

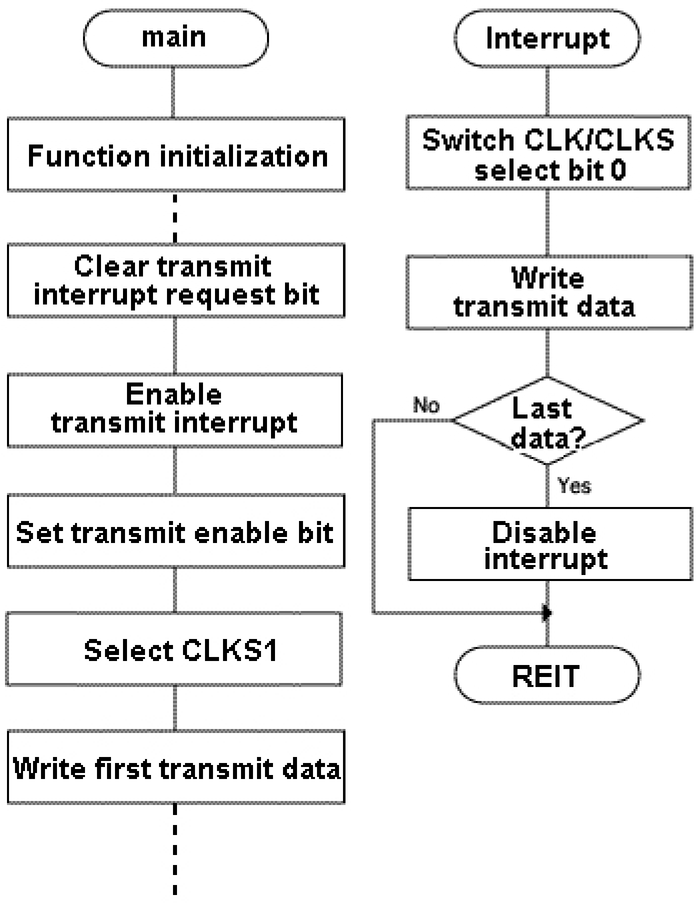


Figure13. General Flowchart of Transferring Clock Dual-Pin Output

Sample Program

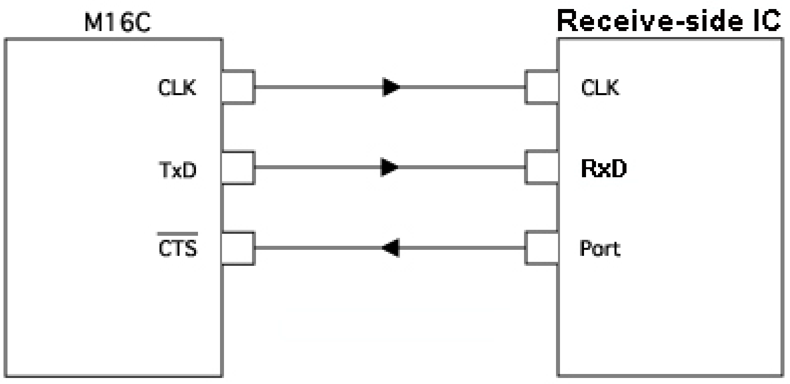
Transmitting data contained in addresses 500h to 503h.

```
                                  .
                                  .
                                  .
;====================   Main Program    ==================
MAIN:
     JSR        INIT               ; Go to UART1 initial settings
     FSET       I                  ; Enable interrupt
                                   ;
     MOV.B      #01h,U1C1          ; Enable transmit
     MOV.B      [A0],U1TB          ; Write transmit data
                                  .
                                  .
                                  .
;============== UART1 Function Initial Settings  ============
INIT:
     MOV.B      #01h,U1MR          ; Select internal clock
     MOV.B      #10h,U1C0          ; Disable CTS function, select count source f1
     MOV.B      #32h,UCON          ; Select transfer clock dual-pin output
                                   ; Set clock output pin to CLKS1
     MOV.B      #0FFh,U1BRG        ; Transfer speed = f1/256 x 1/2
     MOV.W      #500h,DATA         ; Set position for reading transmit data
     MOV.W      #500h,A0
     MOV.B      #3,CNT             ; Set number of transfers (4-1)
     MOV.B      #01h,S1TIC         ; Enable transmit interrupt
INT_END:
     RES
                                  .
                                  .
                                  .
;===========    UART1 Transmit Interrupt Process    ===========
S1T_INT:
     PUSH.W     A0                 ; Save A0
     XOR.B      #10h,UCON          ; Switch clock output pin (CLK1, CLKS1)
     INC.B      DATA               ; Set position for reading next transmit data
     MOV.W      DATA,A0
     MOV.B      [A0],U1TB          ; Write next transmit data
     DEC.B      CNT                ; Count down number of transfers
     JNZ        S1T_INT_1          ; Keep interrupt enabled if more data to transmit
                                   ;
     MOV.B      #00H,S1TIC         ; If last data, disable transmit interrupt
S1T_INT_1:
     POP.W      A0                 ; Restore A0
     REIT
                                  .
                                  .
                                  .
```

3.7 CTS/RTS Functions

The CTS (Clear To Send) function detects the status of the external IC and controls data transmit accordingly. When the CTS function is selected and the input level of the CTS pin goes to "L", the transfer clock is automatically generated and the transmit is started.
Because the level of the CTS pin is confirmed at the start of the transmit, if the level goes to "H" during the transmit, the transfer clock will be stopped after the current transmit is completed. When the CTS pin returns to the "L" level, the transfer clock will be generated again.
The RTS (Request To Send) function informs the external IC that the MCU is receive-ready. With the RTS function, the RTS pin will automatically output an "L" level when the MCU is receive-ready. The RTS pin goes to the "H" level at the first falling edge of the transfer clock.
There are three selections available for the CTS/RTS functions: CTS function only, RTS function only, both functions disabled.

*: If both functions are disabled, the CTS/RTS pin can be used as a programmable input/output port.



Note: **In the M16C/62 Group, TxD2 is N-channel open-drain output and must be pulled up.**

Figure14. Connection Example for CTS Function

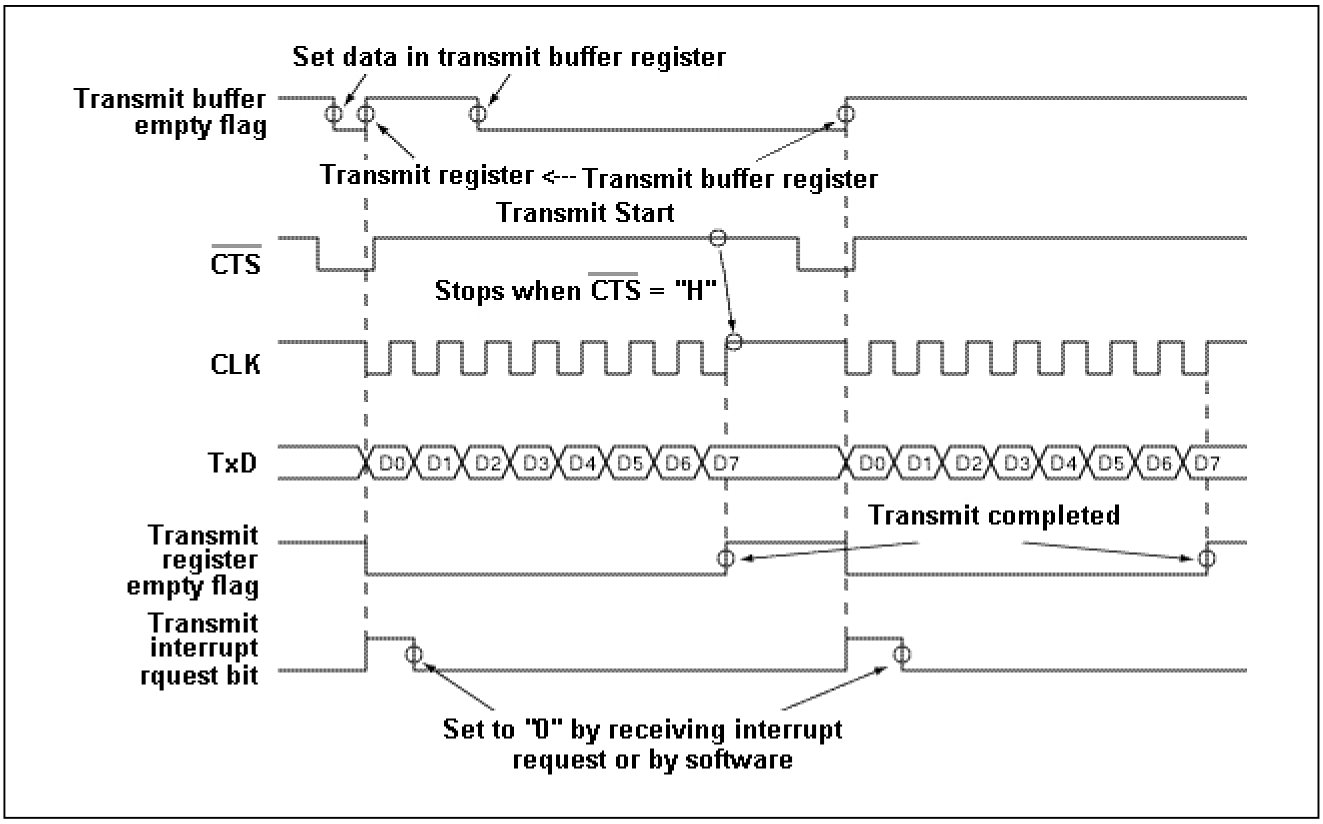
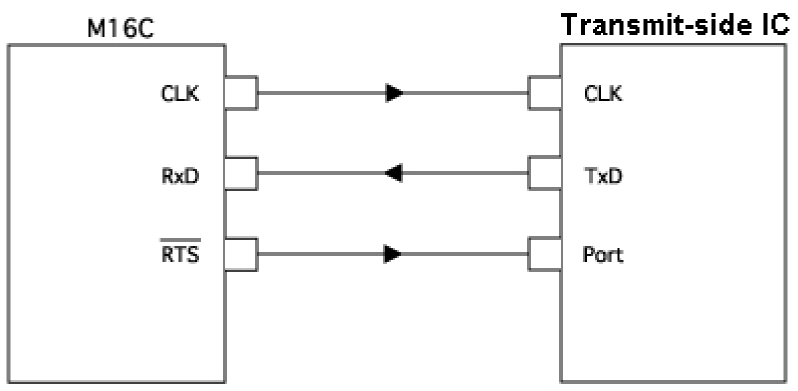
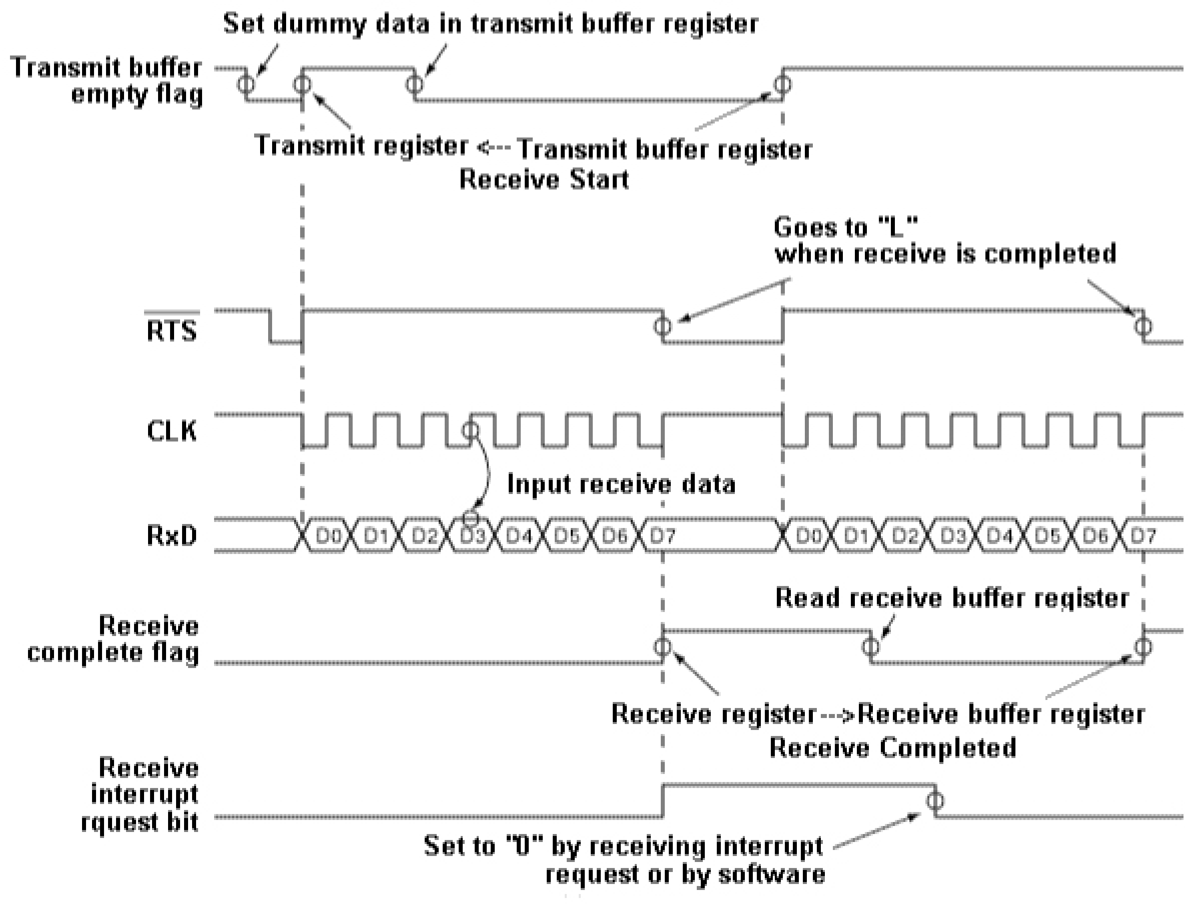

Set data in transmit buffer register
Transmit buffer empty flag
Transmit register <-- Transmit buffer register
Transmit Start
CTS
Stops when CTS = "H"
CLK
TxD
D0 D1 D2 D3 D4 D5 D6 D7
Transmit completed
Transmit register empty flag
Transmit interrupt rquest bit
Set to "0" by receiving interrupt request or by software

Figure15. Operation Timing Diagram of CTS Function



Note: **In the M16C/62 Group, TxD2 is N-channel open-drain output and must be pulled up.**

Figure16. Connection Example for RTS Function


Set dummy data in transmit buffer register
Transmit buffer empty flag
Transmit register <--- Transmit buffer register
Receive Start
Goes to "L" when receive is completed
RTS
CLK
Input receive data
RxD
D0 D1 D2 D3 D4 D5 D6 D7
Read receive buffer register
Receive complete flag
Receive register--->Receive buffer register
Receive Completed
Receive interrupt rquest bit
Set to "0" by receiving interrupt request or by software

Note: The RTS signal goes to "L" when the receive is completed (rising edge of clock for the 8th bit), not when the received data is read from the receive buffer register. Therefore, the next data receive may start upon completion of the current data receive. In this case, read the received data in the receive buffer register by the time the next data receive completes (in order to avoid an overrun error).

Figure17. Operation Timing Diagram of RTS Function

Process Example for RTS Function

In a main routine, enable the receive. In an interrupt process, read out data.

1. Select the RTS function and set each clock-synchronous serial I/O function.
2. Clear the receive interrupt request bit to be used, set the interrupt enable flag.
3. Set the transmit enable bit and the receive enable bit and write a dummy data to the transmit buffer register. The RTS pin output level will then go to "L" to inform the transmit-side that the MCU is in the receive-ready status.
4. Read out the received data from the receive buffer register whenever receive interrupt is generated.
5. In order to receive the next data, write a dummy data to the transmit buffer register until the last data receive has completed.
6. Disable the receive interrupt after reading the last data from the receive buffer register.

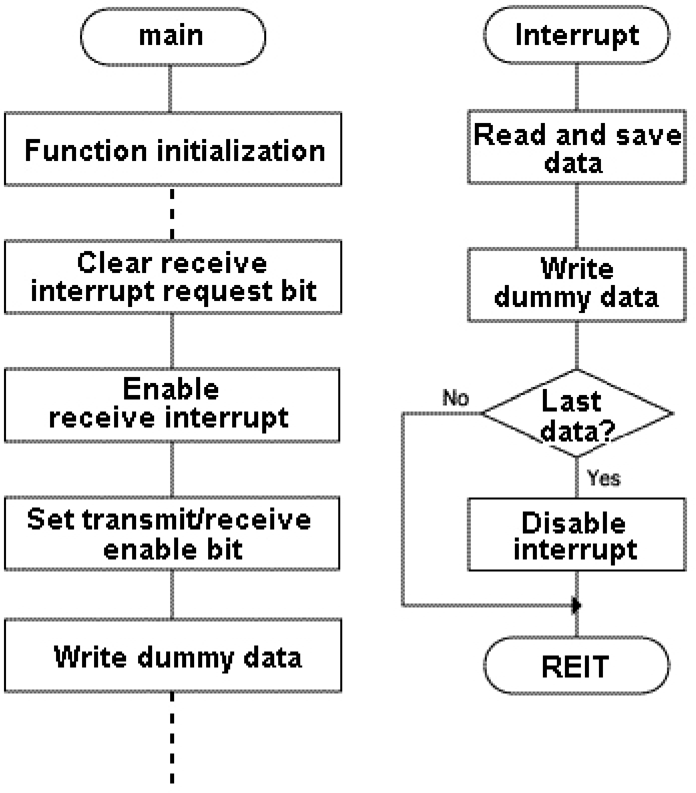


Figure18. General Flowchart of Process Example for RTS Function

Sample Program

Store received data in addresses 500h to 503h.

```
                                        .
                                        .
                                        .
;==================== Main Program ===================
MAIN:
      JSR         INIT              ; Go to UART0 initial settings
      FSET        I                 ; Enable interrupt
                                    ;
      MOV.B       #05h,U0C1         ; Enable receive
      MOV.B       R0L,U0RB          ; Write dummy data = start receive
                                        .
                                        .
                                        .
;============== UART0 Function Initial Settings =============
INIT:
      MOV.B       #09h,U0MR         ; Select external clock
      MOV.B       #04h,U0C0         ; Enable RTS function
      MOV.B       #00h,UCON         ; Disable UART0 continuous receive mode
      MOV.W       #500h,DATA        ; Set position for storing received data
      MOV.B       #4,CNT            ; Set number of transfers
      MOV.B       #01h,S0RIC        ; Set UART0 receive interrupt priority level
INT_END:
      RTS
                                        .
                                        .
                                        .
;============   UART0 Receive Interrupt Process    ===========
S0R_INT:
      PUSH.W      A0                ; Save A0
      PUSH.B      R0L               ; Save R0L
      MOV.W       DATA,A0
      MOV.B       U0RB,[A0]         ; Read received data
      INC.B       DATA              ; Set position for storing next receive data
      DEC.B       CNT               ; Count down number of transfers
      JNZ         S0R_INT_1         ; Keep interrupt enabled if more data to receive
                                    ;
       MOV.B      #00H,S0RIC        ; If last data, disable receive interrupt
S0R_INT_1:
      POP.B       R0L               ; Restore R0L
      POP.W       A0                ; Restore A0
      REIT
                                        .
                                        .
                                        .
```

3.8 How to Recover from an Error

Recovery procedures, such as initialization, must be executed if an error occurs during clock-synchronous serial I/O operations. In addition, after recovering from the error, it may be necessary to execute the resend process or resend request process with software.

Detection of Receive Error

The only detectable error in M16C/60 Series clock-synchronous serial I/O is the overrun error. An overrun error occurs when new data is lined up to the receive register before the previously received data is read out from the receive buffer register. When this error occurs, the last received data is stored in the receive buffer register. Also, the receive interrupt request bit is not set to "1". An overrun error causes the overrun error flag to go to "1".

Recovery from Error

Execute recovery process according to the following flowchart. Any changes in this setup order may result in MCU malfunction.
All errors other than overrun error must be detected through software.

Table5. Recovery from Error

Cause of Error	Error Determination	Recovery Method
Overrun error	Reference error flag	Initialization Resend request process
Incomplete transfer due to synchronization corruption	Time out determined by software	Initialization Resend process/Resend request process
Other (abnormal data, etc)	Data determined by software	Resend process/Resend request process

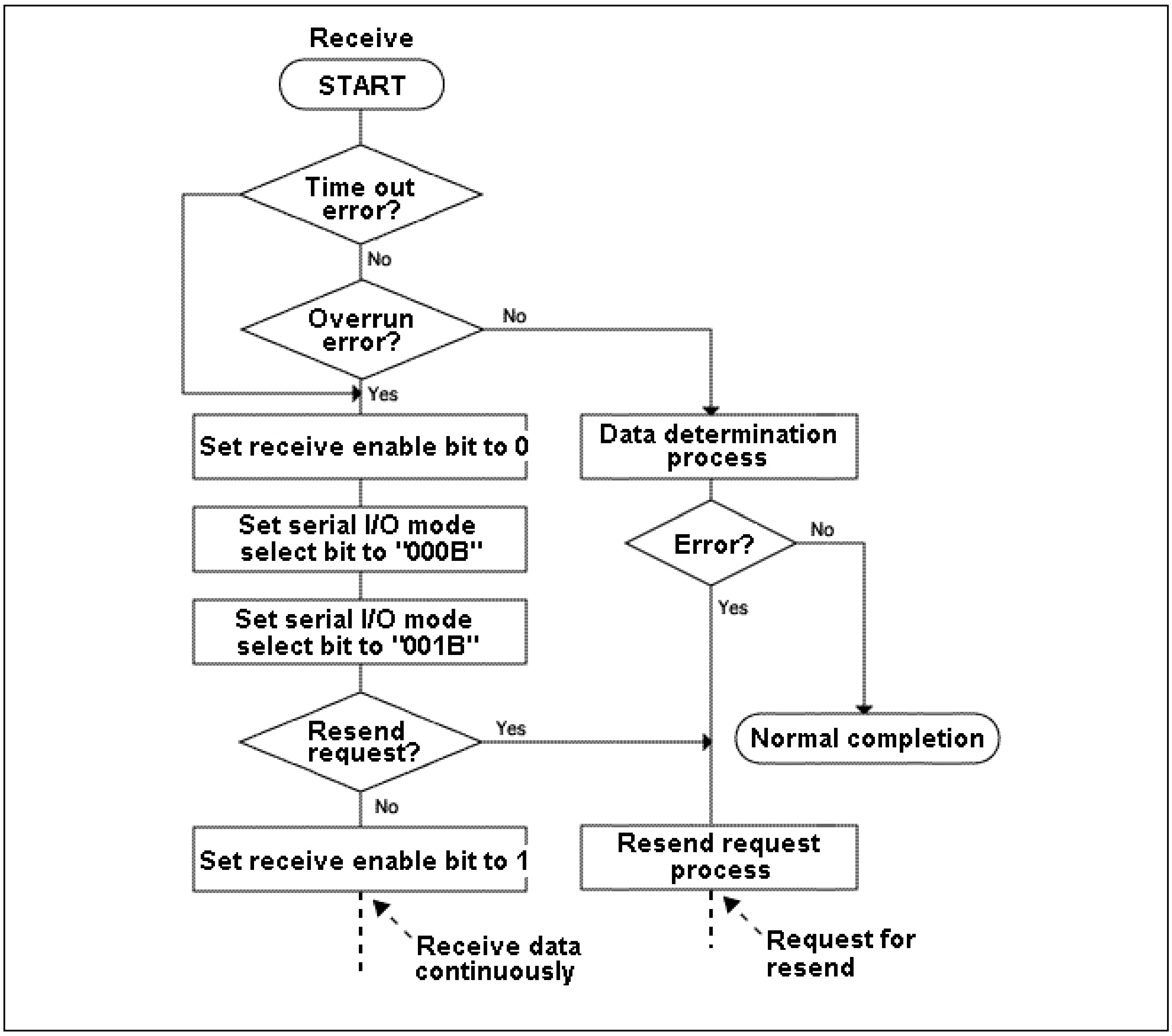

Receive
START
Time out error?
No
Overrun error?
No
Yes
Set receive enable bit to 0
Set serial I/O mode select bit to "000B"
Set serial I/O mode select bit to "001B"
Resend request?
Yes
No
Set receive enable bit to 1
Receive data continuously
Data determination process
Error?
No
Yes
Normal completion
Resend request process
Request for resend

Figure19. Recovery Process (Initialization, etc.) Flowchart: Receive

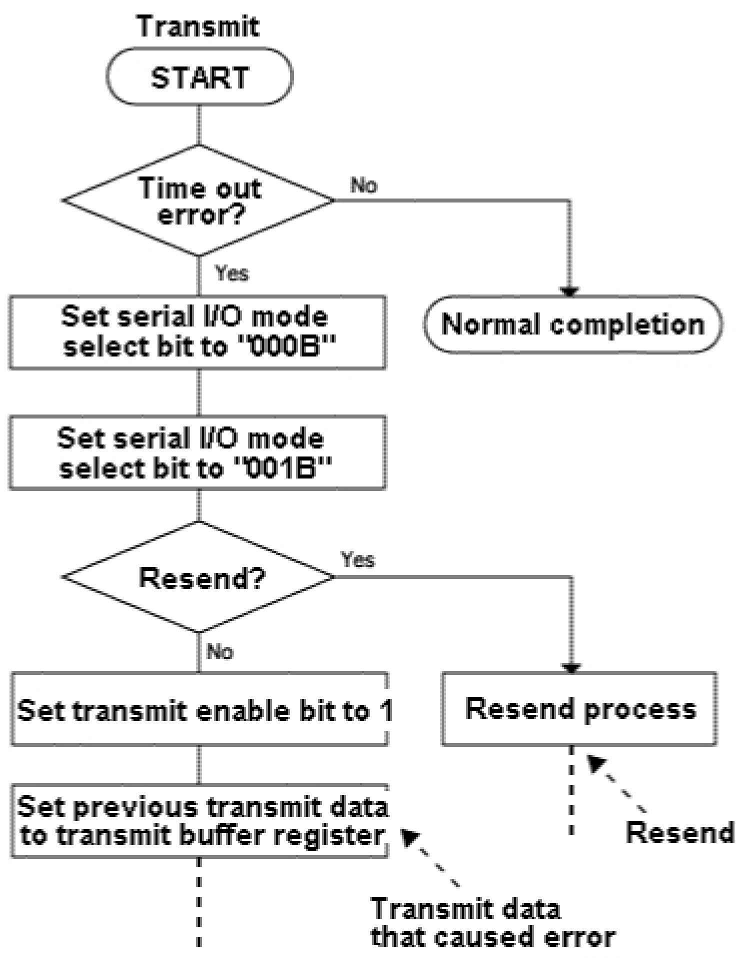

Transmit
START
Time out error?
No
Yes
Normal completion
Set serial I/O mode select bit to "000B"
Set serial I/O mode select bit to "001B"
Resend?
Yes
No
Set transmit enable bit to 1
Resend process
Set previous transmit data to transmit buffer register
Resend
Transmit data that caused error

Figure20. Recovery Process (Initialization, etc.) Flowchart: Transmit

3.9 Communications Between MCU and EEPROM using I^{2}C Mode

I^{2}C Bus is a bi-directional, multi-master bus communication protocol developed by Philips, utilized widely in many ICs. (For more details, please refer to the Philips I^{2}C Bus Specification.) The M16C/60 Series UART2 includes a simple I^{2}C mode, enabling I^{2}C bus interfaces (single-master/multi-master) when used in combination with the appropriate software.

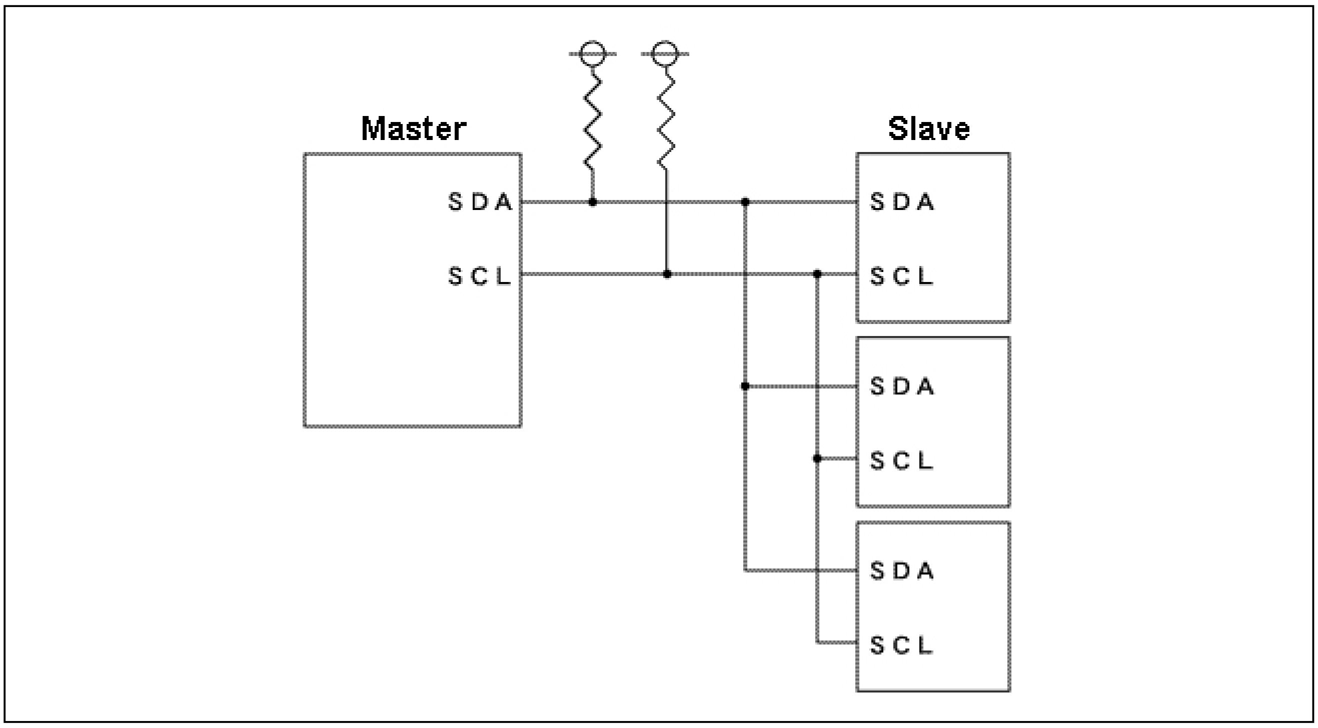
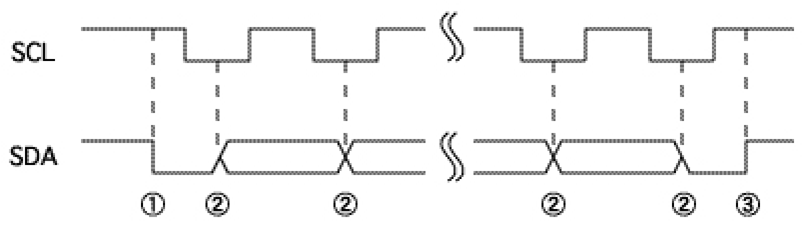


Figure21. Connection Example of MCU and EEPROM using I^{2}C Mode

Outline of I^{2}C Bus

The I^{2}C Bus enables serial interfacing using 2 lines; data (SDA) line and clock (SCL) line. Interface is achieved through indications from the levels and changing points of data and clock signals. I^{2}C Bus interfaces are executed in the following order: master sends START condition, data is sent or received by master or slave, master sends STOP condition.



① The START condition is triggered when the data is switched from "H" to "L" while the clock is at "H".

② The data bits can be modified only when the clock is at "L".

③ The STOP condition is triggered when the data is switched from "L" to "H" while the clock is at "H".

Figure22. Timing Diagram of SCL and SDA lines

Read/write formats in I^{2}C bus transfers are: START condition, 7-bit slave address, 1-bit read/write setting, 8-bit unit data, and STOP condition. In addition, during transfer, the data-receive side returns an acknowledge (SDA = "L") after each 8-bit transfer.

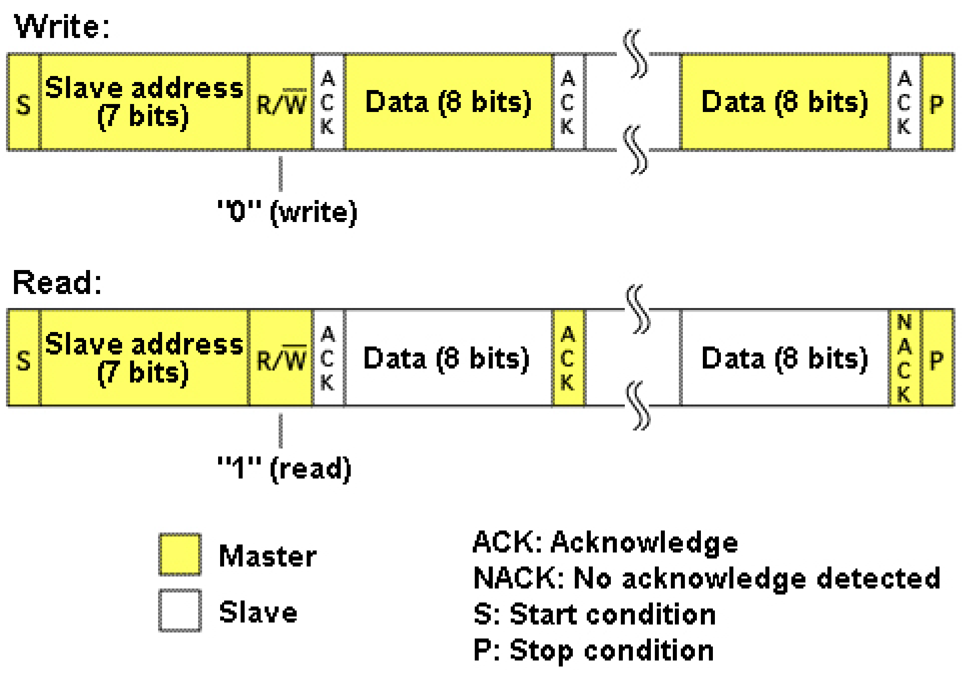


Figure23. Read/Write formats in I^{2}C bus transfers

Process Example in Simple I^2C Mode

Please refer to the general description of I^2C bus communication protocol and multi-master communication methods found in the "M16C/62 Group Application Note: Simple I^2C Bus Mode".

The following sample program describes I^2C Bus interfacing with an EEPROM, using an M16C/62 Group MCU as the single-master. (The read/write format is an example.)

Process Example in Write Mode

Write 1 byte of data from the CPU (master) to the EEPROM (slave).

[Outline]

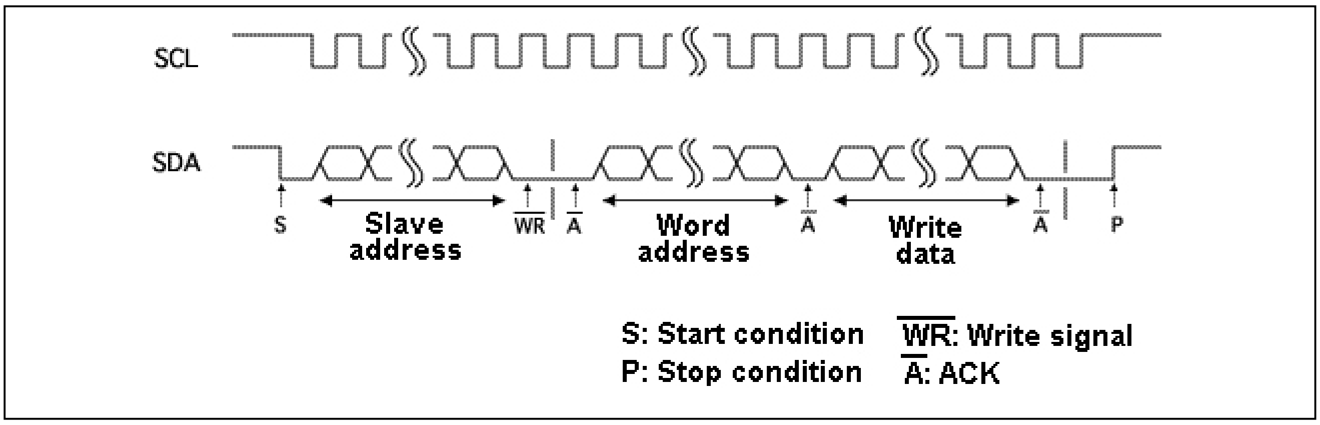


Figure24. Timing Diagram of SCL and SDA lines in Write Mode

1. Select the simple I^2C mode and set each clock-synchronous serial I/O function.
2. Master triggers START condition.
3. Master outputs slave address, checks for acknowledge.
4. Master outputs word address (write destination), checks for acknowledge.
5. Master outputs data, checks for acknowledge.
6. Master triggers STOP condition.

Note: When a NACK (No Acknowledge: SDA = "H") is detected during the acknowledge check, the rewrite process, etc., must be executed by software after the STOP condition is sent.

Sample Program

```
;****************************************************************************
; Copyright 2003 Renesas Technology America, Inc.
; AND Renesas Solutions Corporation
; All rights reserved
;=========================== Main Program ===================================
;
MAIN:
       JSR    INIT                  ; Function Initialization
       JSR    MASTER_OUT            ; Write Mode Process

;****************************************************************************
;                     Function Initialization
;****************************************************************************
;
INIT:
       BCLR   PD7_0                 ; Set P7_0 to input.
       BSET   P7_0                  ; P7_0 = "H"
       BCLR   PD7_1                 ; Set P7_1 to input.
       BSET   P7_1                  ; P7_1 = "H"
;
       MOV.B  #01h,U2SMR            ; Select IIC mode.
       MOV.B  #82h,U2SMR2           ; Validate ACK and NACK interrupts.
                                    ; Enable SCL synchronous function.
       MOV.B  #40h,U2SMR3
       MOV.B  #00h,U2MR             ; Invalidate serial I/O = Port control
       MOV.B  #90h,U2C0             ; Disable CTS/RTS function, select MSB first.
       MOV.B  #10h,U2C1             ; Disable transfer
       MOV.B  #(80-1),U2BRG         ; Transfer speed = 100Kbps
INIT_END:
       RTS
```

```
;*****************************************************************************
;                     Write Mode Process
;*****************************************************************************
;
MASTER_OUT:
       JSR    MAKE_START            ; START Condition Generation
       MOV.B  #0A0h,R0L             ; Slave address + Data direction bit (W)
       JSR    OUTPUT_DATA           ; Slave address and data direction bit (W) output.
       JNC    MASTER_OUT_10         ; C=1: No Acknowledge is detected.
       JSR    ABORT                 ; Go to error process.
       FSET   C                     ; C=1 -> Go to rewrite process, etc.
       RTS

MASTER_OUT_10:
       MOV.B  #80h,R0L              ; Word address
       JSR    OUTPUT_DATA           ; Word address output.
       JNC    MASTER_OUT_11         ; C=1: No Acknowledge is detected.
       JSR    ABORT                 ; Go to error process.
       FSET   C                     ; C=1 -> Go to rewrite process, etc.
       RTS

MASTER_OUT_11:
       MOV.B  #55h,R0L              ; Write data (55H)
       JSR    OUTPUT_DATA           ; Write data output.
       JNC    MASTER_OUT_12         ; C=1: No Acknowledge is detected.
       JSR    ABORT                 ; Go to error process.
       RTS

MASTER_OUT_12:
       BTST   P7_1
       JNC    MASTER_OUT_12         ; Wait while SCL (P7_1) is "L".
       BCLR   TE_U2C1                      ; Disable transmit
       JSR    MAKE_STOP             ; STOP Condition Generation
       RTS
```

```
;****************************************************************************
;                    START Condition Generation
;****************************************************************************
;
MAKE_START:
       JSR    WAIT_5microSEC
       JSR    CHG_scl_H            ; SCL = "H"
MAKE_ST1:
       BTST   P7_1
       JNC    MAKE_ST1             ; Wait until SCL (P7_1) input goes to "H".
       BTST   P7_0
       JC     MAKE_ST2             ; SDA (P7_0) = "H" -> MAKE_ST2
       JSR    CHG_sda_L            ; "L" output from SDA (P7_0).
       MOV.B  #00h,U2MR            ; Invalidate serial I/O = Port control.
       JSR    CHG_scl_L            ; "L" output from SCL (P7_1).
       JSR    WAIT_10microSEC
       JSR    CHG_sda_H            ; SDA = "H"
       JSR    WAIT_10microSEC
MAKE_ST2:
       JSR    CHG_sda_L            ; START condition output.
       JSR    WAIT_5microSEC
       MOV.B  #02h,U2MR            ; IIC mode settings.
       RTS

;****************************************************************************
;                    STOP Condition Generation
;****************************************************************************
;
MAKE_STOP:
       BCLR   TE_U2C1              ; Disable transmit
       BCLR   RE_U2C1              ; Disable receive
       BCLR   PD7_0                ; Set SDA (P7_0) to input.
       MOV.B  #00h,U2MR            ; Invalidate serial I/O = Port control
       JSR    CHG_scl_L            ; "L" output from SCL (P7_1).
       JSR    CHG_sda_L            ; "L" output from SDA (P7_0).
       JSR    WAIT_5microSEC
       JSR    CHG_scl_H            ; SCL (P7_1) = "H"
MAKE_SP_1:
       BTST   P7_1
       JNC    MAKE_SP_1            ; Wait until SCL (P7_1) goes to "H".
       JSR    WAIT_10microSEC
       JSR    CHG_sda_H            ; STOP condition output.
       RTS
```

```
;*****************************************************************************
;                     Data Transmit
;*****************************************************************************
;
OUTPUT_DATA:
       BSET   TE_U2C1                       ; Enable transmit
       BCLR   IR_S2TIC               ; Clear NACK interrupt request bit.
       BCLR   IR_S2RIC               ; Clear ACK interrupt request bit.
       OR.W   #100h,R0               ; Set SDA to "H" at ACK detection timing.
       MOV.W  R0,U2TB                       ; Set transmit data.
OUTPUT_1:
       BTST   IR_S2RIC               ; ACK interrupt request bit = 1?
       JNC    OUTPUT_2               ; Go to determination of NACK interrupt request
bit.
       JSR    WAIT_10microSEC
       FCLR   C
       RTS                           ; Acknowledge is detected when C=0.

OUTPUT_2:
       BTST   IR_S2TIC               ; NACK interrupt request bit = 1?
       JNC    OUTPUT_1               ; Go to determination of ACK interrupt request
bit.
       RTS                           ; No Acknowledge is detected when C=1.

;*****************************************************************************
;                     Stop Process (when NACK is detected)
;*****************************************************************************
;
ABORT:
       BCLR   PD7_0
       BSET   P7_0                   ; SDA (P7_0) = "H"
       BCLR   TE_U2C1                       ; Disable transmit
       MOV.B  #00h,U2MR              ; Invalidate serial I/O = Port control
       JSR    WAIT_5microSEC
       JSR    MAKE_STOP              ; STOP signal output.
       RTS
```

```
;***************************************************************************
;                     Port Switch Process
;***************************************************************************
;
CHG_scl_H:                          ; SCL = "H"
       BCLR   PD7_1
       RTS

CHG_scl_L:                          ; "L" output from SCL.
       BCLR   P7_1
       NOP
       BSET   PD7_1
       RTS

CHG_sda_H:                          ; SDA = "H"
       BCLR   PD7_0
       RTS

CHG_sda_L:                          ; "L" output from SDA.
       BCLR   P7_0
       NOP
       BSET   PD7_0
       RTS

;***************************************************************************
;                     Wait Time Process
;***************************************************************************
;
WAIT_10microSEC:
       .MREPEAT     40
       NOP
       .ENDR

WAIT_7_5_microSEC:
       .MREPEAT     40
       NOP
       .ENDR

WAIT_5microSEC:
       .MREPEAT     40
       NOP
       .ENDR

WAIT_2_5microSEC:
       .MREPEAT     40
       NOP
       .ENDR
       RTS
```

Process Example in Read Mode

CPU (master) reads 1 byte of data from the EEPROM (slave).

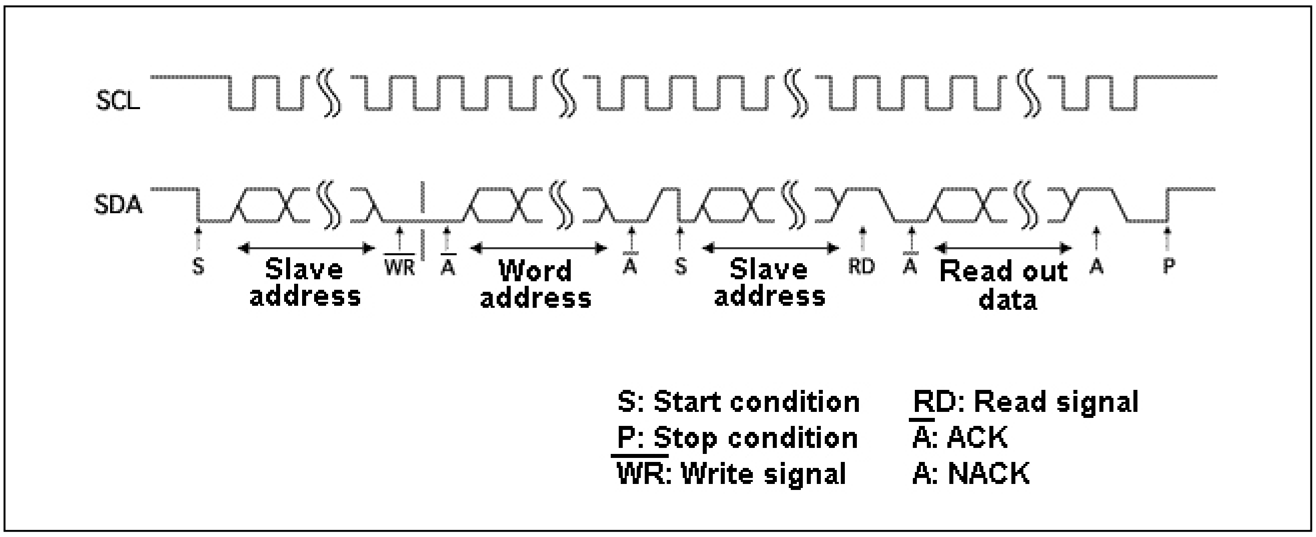


Figure25. Timing Diagram of SCL and SDA lines in Read Mode

1. Select the simple I^{2}C mode and set each clock-synchronous serial I/O function.
2. Master triggers START condition.
3. Master outputs slave address, checks for acknowledge.
4. Master outputs word address (read destination), checks for acknowledge.
5. Master triggers START condition.
6. Master outputs slave address, checks for acknowledge.
7. Master inputs transferred data, finally outputs a NACK.
8. Master triggers STOP condition.

Note: When a NACK (No Acknowledge: SDA = "H") is detected during the acknowledge check, the rewrite process, etc., must be executed by software after the STOP condition is sent.

Sample Program

```
;*****************************************************************************
; Copyright 2003 Renesas Technology Corporation
; All rights reserved
;************************* Work RAM Area Reserve ****************************
;
              .SECTION      WORK,DATA
              .ORG          00400H
WORKRAM_TOP:
DATA:         .BLKB         1      ; Store receive data.
WORKRAM_END:
;
;=========================== Main Program ===================================
;
MAIN:
       JSR    INIT                 ; Function Initialization
       JSR    MASTER_OUT           ; Read Mode Process

;*****************************************************************************
;                     Function Initialization
;*****************************************************************************
;
INIT:
       BCLR   PD7_0                ; Set P7_0 to input.
       BSET   P7_0                 ; P7_0 = "H"
       BCLR   PD7_1                ; Set P7_1 to input.
       BSET   P7_1                 ; P7_1 = "H"
;
       MOV.B  #01h,U2SMR           ; Select IIC mode.
       MOV.B  #82h,U2SMR2          ; Validate ACK and NACK interrupts.
                                   ; Enable SCL synchronous function.
       MOV.B  #40h,U2SMR3
       MOV.B  #00h,U2MR            ; Invalidate serial I/O = Port control
       MOV.B  #90h,U2C0            ; Disable CTS/RTS function, select MSB first.
       MOV.B  #10h,U2C1            ; Disable transfer
       MOV.B  #(80-1),U2BRG        ; Transfer speed = 100Kbps
INIT_END:
       RTS
```

```
;***************************************************************************
;                      Read Mode Process
;***************************************************************************
;
MASTER_IN:
       JSR     MAKE_START             ; START Condition Generation
       MOV.B   #0A0h,R0L              ; Slave address + Data direction bit (W)
       JSR     OUTPUT_DATA            ; Slave address and data direction bit (W) output.
       JNC     MASTER_IN_10
       JSR     ABORT
       FSET    C                      ; C=1 -> Go to rewrite process, etc.
       RTS

MASTER_IN_10:
       MOV.B   #80h,R0L               ; Word address
       JSR     OUTPUT_DATA            ; Word address output.
       JNC     MASTER_IN_11
       JSR     ABORT
       FSET    C                      ; C=1 -> Go to rewrite process, etc.
       RTS

MASTER_IN_11:
       BTST    P7_1
       JNC     MASTER_IN_11           ; Wait while SCL (P7_1) is "L".
       BCLR    TE_U2C1                       ; Disable transmit

       BCLR    PD7_0                  ; Set SDA (P7_0) to input.
       MOV.B   #00h,U2MR              ; Invalidate serial I/O = Port control
       JSR     CHG_scl_L              ; SCL = "L"

MASTER_IN_12:
       BTST    P7_0
       JNC     MASTER_IN_12           ; Wait while SDA (P7_0) is "L".
       JSR     WAIT_10microSEC

       JSR     MAKE_START             ; START Condition Generation
       MOV.B   #0A1h,R0L              ; Slave address + Data direction bit (R)
       JSR     OUTPUT_DATA            ; Slave address and data direction bit (R) output.
       JNC     MASTER_IN_13
       JSR     ABORT
       FSET    C                      ; C=1 -> Go to rewrite process, etc.
       RTS

MASTER_IN_13:
       BTST    P7_1
       JNC     MASTER_IN_13           ; Wait while SCL (P7_1) is "L".
       BCLR    TE_U2C1                       ; Disable transmit
       JSR     INPUT_DATA             ; Data Receive
       JSR     MAKE_STOP              ; STOP Condition Generation
       FCLR    C
       RTS
```

```
;****************************************************************************
;                      START Condition Generation
;****************************************************************************
;
MAKE_START:
       JSR    WAIT_5microSEC
       JSR    CHG_scl_H            ; SCL = "H"
MAKE_ST1:
       BTST   P7_1
       JNC    MAKE_ST1             ; Wait until SCL (P7_1) input goes to "H".
       BTST   P7_0
       JC     MAKE_ST2             ; SDA (P7_0) = "H" -> MAKE_ST2
       JSR    CHG_sda_L            ; "L" output from SDA (P7_0).
       MOV.B  #00h,U2MR            ; Invalidate serial I/O = Port control.
       JSR    CHG_scl_L            ; "L" output from SCL (P7_1).
       JSR    WAIT_10microSEC
       JSR    CHG_sda_H            ; SDA = "H"
       JSR    WAIT_10microSEC
MAKE_ST2:
       JSR    CHG_sda_L            ; START condition output.
       JSR    WAIT_5microSEC
       MOV.B  #02h,U2MR            ; IIC mode settings.
       RTS

;****************************************************************************
;                      STOP Condition Generation
;****************************************************************************
;
MAKE_STOP:
       BCLR   TE_U2C1              ; Disable transmit
       BCLR   RE_U2C1              ; Disable receive
       BCLR   PD7_0                ; Set SDA (P7_0) to input.
       MOV.B  #00h,U2MR            ; Invalidate serial I/O = Port control
       JSR    CHG_scl_L            ; "L" output from SCL (P7_1).
       JSR    CHG_sda_L            ; "L" output from SDA (P7_0).
       JSR    WAIT_5microSEC
       JSR    CHG_scl_H            ; Set SCL (P7_1) to input.
MAKE_SP_1:
       BTST   P7_1
       JNC    MAKE_SP_1            ; Wait until "H" is input to SCL (P7_1).
       JSR    WAIT_10microSEC
       JSR    CHG_sda_H            ; STOP condition output.
       RTS
```

```
;***************************************************************************
;                    Data Transmit
;***************************************************************************
;
OUTPUT_DATA:
       BSET   TE_U2C1                ; Enable transmit
       BCLR   IR_S2TIC               ; Clear NACK interrupt request bit.
       BCLR   IR_S2RIC               ; Clear ACK interrupt request bit.
       OR.W   #100h,R0               ; Set SDA to "H" at ACK detection timing.
       MOV.W  R0,U2TB                ; Set transmit data.
OUTPUT_1:
       BTST   IR_S2RIC               ; ACK interrupt request bit = 1?
       JNC    OUTPUT_2               ; Go to determination of NACK interrupt request
bit.
       JSR    WAIT_10microSEC
       FCLR   C
       RTS                           ; Acknowledge is detected when C=0.

OUTPUT_2:
       BTST   IR_S2TIC               ; NACK interrupt request bit = 1?
       JNC    OUTPUT_1               ; Go to determination of ACK interrupt request
bit.
       RTS                           ; No Acknowledge is detected when C=1.

;***************************************************************************
;                    Data Receive
;***************************************************************************
;
INPUT_DATA:
       BSET   TE_U2C1                ; Enable transmit
       BSET   RE_U2C1                ; Enable receive
       BCLR   IR_S2TIC               ; Clear NACK interrupt request bit.
       BCLR   IR_S2RIC               ; Clear ACK interrupt request bit.
       MOV.W  #1FFh,U2TB             ; Generate no acknowledge (last data).
ID_10:                               ; Wait until interrupt request bit is set to "1".
       BTST   IR_S2TIC
       JC     ID_20
       BTST   IR_S2RIC
       JNC    ID_10
ID_20:
       MOV.B  U2RB,DATA              ; Store received data.
       JSR    WAIT_10microSEC
       BCLR   TE_U2C1                ; Disable transmit
       BCLR   RE_U2C1                ; Disable receive
       RTS
```

```
;****************************************************************************
;                     Stop Process (when NACK is detected)
;****************************************************************************
;
ABORT:
       BCLR   PD7_0                 ; Set SDA (P7_0) to input.
       BSET   P7_0                  ; SDA (P7_0) = "H"
       BCLR   TE_U2C1               ; Disable transmit
       MOV.B  #00h,U2MR             ; Invalidate serial I/O = Port control
       JSR    WAIT_5microSEC
       JSR    MAKE_STOP             ; STOP signal output.
       RTS

;****************************************************************************
;                     Port Switch Process
;****************************************************************************
;
CHG_scl_H:                          ; SCL = "H"
       BCLR   PD7_1
       RTS

CHG_scl_L:                          ; "L" output from SCL.
       BCLR   P7_1
       NOP
       BSET   PD7_1
       RTS

CHG_sda_H:                          ; SDA = "H"
       BCLR   PD7_0
       RTS

CHG_sda_L:                          ; "L" output from SDA.
       BCLR   P7_0
       NOP
       BSET   PD7_0
       RTS

;****************************************************************************
;                     Wait Time Process
;****************************************************************************
;
WAIT_10microSEC:
       .MREPEAT      40
       NOP
       .ENDR

WAIT_7_5_microSEC:
       .MREPEAT      40
       NOP
       .ENDR

WAIT_5microSEC:
       .MREPEAT      40
       NOP
       .ENDR
```

4. Reference

Renesas Technology Corporation Semiconductor Home Page

http://www.renesas.com/

E-mail Support

support_apl@renesas.com

REVISION HISTORY

Rev.	Date	Description	
		Page	Summary
1.00	Feb 25, 2004	-	First edition issued