AN202495

Using the Multifunction Serial Interface of the Traveo™ Family

Author: Masahide Karino

Associated Part Family: Traveo™ Family S6J3110/3120/3200/3310/3320/3330/3340/3350/3360/3370/3400 Series

Related Documents: For a complete list, see Related Documents

This application note describes how to use the multifunction serial interface of the Cypress Traveo family S6J3110/3120/3200/3310/3320/3330/3340/3350/3360/3370/3400 series.

Contents

1 Introduction

This application note is intended for users of the Cypress Traveo family S6J3110/3120/3200/3310/3320/3330/3340/3350/3360/3370/3400 series. It describes how to use the multifunction serial interface.

2 Overview of Multifunction Serial Interface

The multifunction serial interface provides the UART (asynchronous serial interface), CSIO (SPI-supported, clock synchronous serial interface), LIN interface v2.1 (LIN communication interface v2.1), and I²C communication function. UART is a general-purpose serial data communication interface for asynchronous communication with the external devices. It supports bidirectional communication (normal mode) and master/slave communication (multiprocessor mode: master and slave roles are both supported).

CSIO is a general-purpose serial data communication interface for synchronous communication with external devices (supports SPI). It supports bidirectional communication and the chip select signal control function.

LIN interface v2.1 provides the functions to support the LIN bus and operate as a master/slave device in bidirectional LIN communication. It supports manual mode and assist mode. In assist mode, the LIN header section can be automatically transmitted or detected.

The I²C communication control interface supports the I²C bus and operates as a master/slave device on the I²C bus. It supports standard mode (100 kbps) and fast mode (400 kbps). Table 1 describes the I²C mode supported by the Traveo family.

Table 1. I²C Mode of Traveo Family

Series	100 kbps	100 kbps / 400 kbps
S6J3110	All channels	–
S6J3120	ch.0, ch.3, ch.4, ch.8 to ch.11	–
S6J3200	ch.4, ch.10, ch.12	ch.16, ch.17
S6J3310/3320/3330/3340	ch.0, ch.1, ch.4, ch.8 to ch.12	ch.16, ch.17
S6J3350	ch.0, ch.1, ch.4, ch.8 to ch.12	ch.16, ch.17
S6J3360/3370	ch.0, ch.1, ch.4, ch5, ch8 to ch.11	ch.6, ch.7
S6J3400	ch.0 to ch.2, ch.4, ch.6 to ch.13	ch.3, ch.5

3 Multifunction Serial Interface Settings

Specific settings are required to use the multifunction serial interface of the S6J3110/3120/3200/3310/3320/3330/3340/3350/3360/3370/3400 series. This section describes how to set up the UART, CSIO, LIN, and I²C.

3.1 UART Communication Settings

Figure 1 shows an example of the UART communication connection.

Figure 1. Example of UART Communication Connection

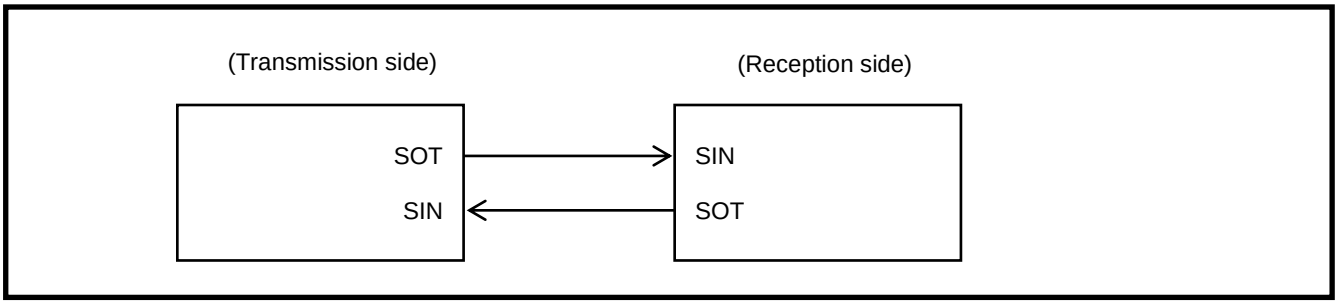


Figure 2 and Figure 3 illustrate example UART communication flow charts.

Figure 2. Example UART Communication Flow Chart (FIFO not used)

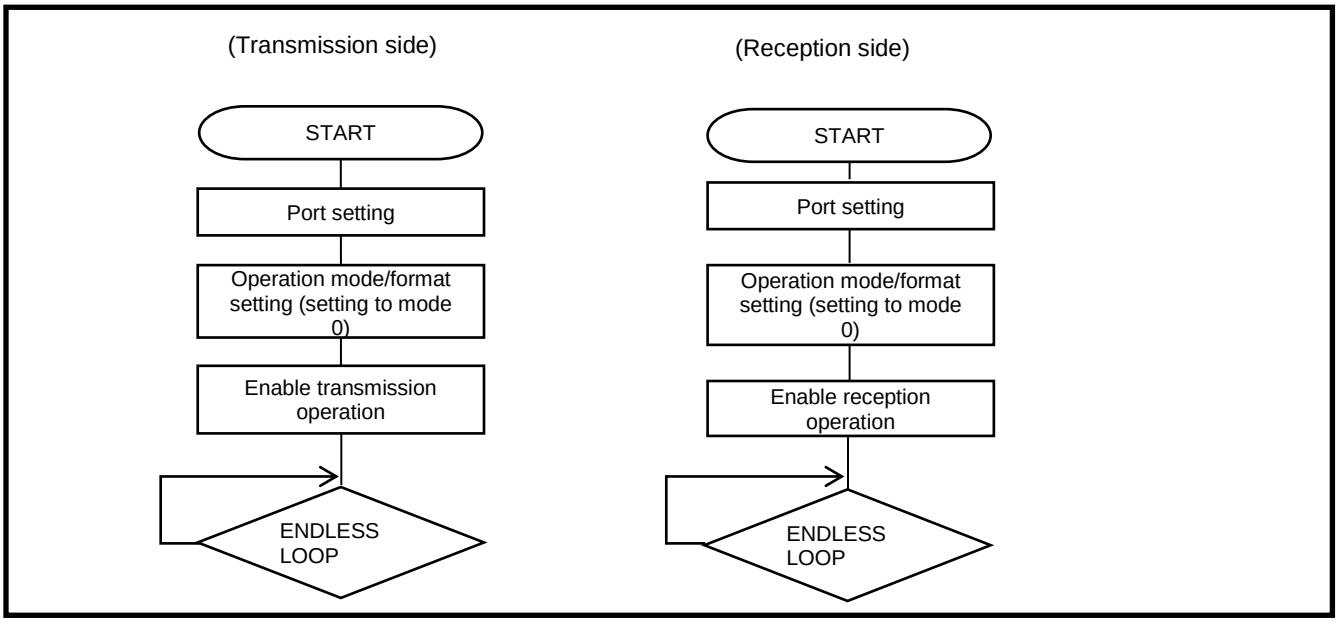
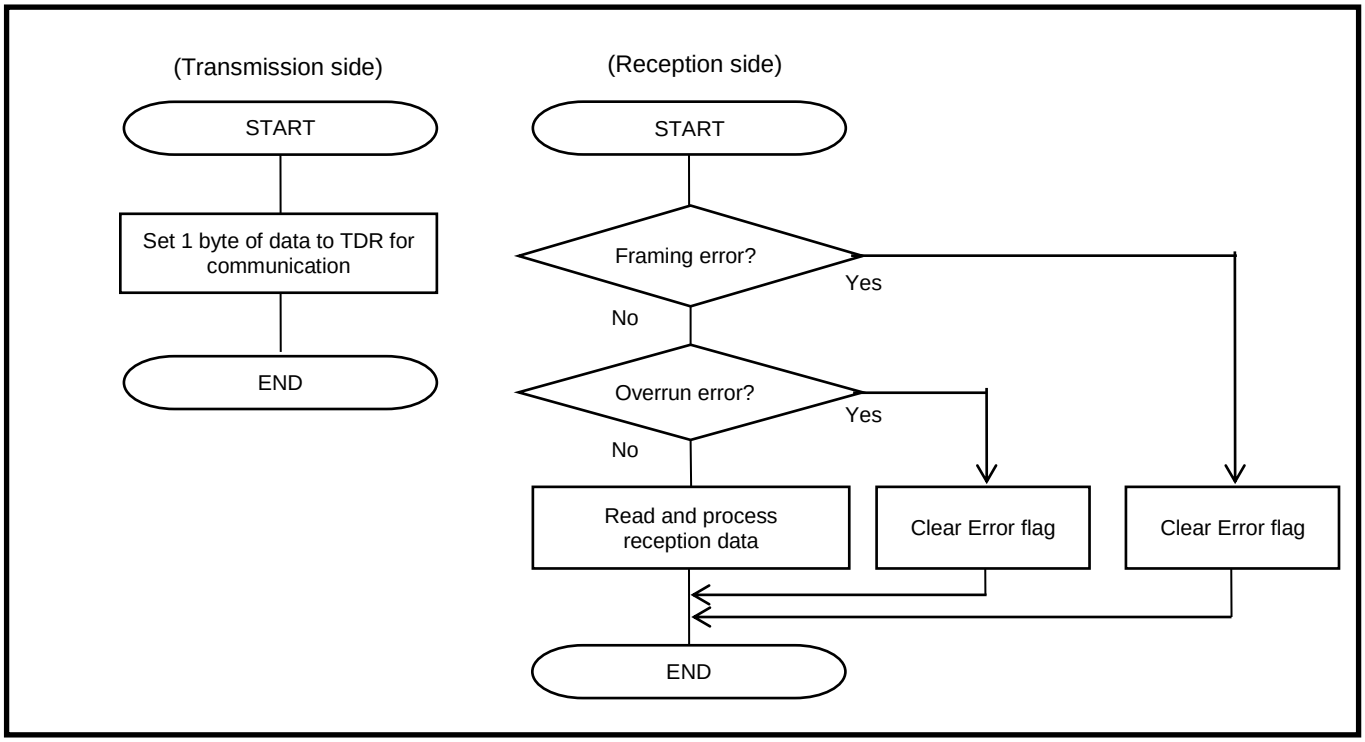


Figure 3. Example UART Communication Interrupt Routine Flow Chart (FIFO not used)



3.1.1 Baud Rate Setting

The Baud Rate Generator Register (BGR0/1) sets the 15-bit reload counter. The baud rate calculation formulas are as follows.

- Calculation formulas of the reload value

  V = (f/b) – 1

  V: Reload value, f: Bus clock frequency or external clock frequency (Hz), b: Baud rate (bps)

- Calculation example of the reload value

  When the bus clock is 36 MHz, using the bus clock, the baud rate is 115200 bps. The reload value can be calculated as follows.

  - Reload value

    V = (36000000 / 115200) – 1 = 311

    Therefore, the baud rate is

    b = (36000000 / (311+1) = 115384 bps

3.1.2 UART Transmission/Reception Port Settings of Example Program

Code 1 describes the UART transmission/reception settings of the example program.

Code 1. UART Settings of Example Program (for S6J3110 series)

```
int main(void)
{   /* Keycode register setting for SOT0_0  */
    PPC_KEYCDR=0x10000028;
    PPC_KEYCDR=0x50000028;
    PPC_KEYCDR=0x90000028;
    PPC_KEYCDR=0xD0000028;
    PPC_PCFGR020=0xC002;

    /* Keycode register setting for SIN1_0 */
    PPC_KEYCDR=0x10000038;
    PPC_KEYCDR=0x50000038;
    PPC_KEYCDR=0x90000038;
    PPC_KEYCDR=0xD0000038;
    PPC_PCFGR028=0x1000;

    /* Keycode register setting for AN2 analog input disable */
    ADER_KEYCDR=0x20000804;
    ADER_KEYCDR=0x60000804;
    ADER_KEYCDR=0xA0000804;
    ADER_KEYCDR=0xE0000804;
    ADER_ADER0=0xFFFFFFFB

    /* Keycode register setting for SOT0_0 output */
    GPIO_KEYCDR=0x20000204;
    GPIO_KEYCDR=0x60000204;
    GPIO_KEYCDR=0xA0000204;
    GPIO_KEYCDR=0xE0000204;
    GPIO_DDR0=0x00100000;

    /* Keycode register setting for port input enable */
    GPIO_KEYCDR=0x20000400;
    GPIO_KEYCDR=0x60000400;
    GPIO_KEYCDR=0xA0000400;
    GPIO_KEYCDR=0xE0000400;
    GPIO_PORTEN=0x00000001;

/* Uart ch1 Rx */
   Asynch_Uart1_RX();

/* Uart ch0 Tx */
   Asynch_Uart0_TX();

  /* Endless loop */
  for(;;)
      {
          ClearWatchdog();
      }
  }
```

Port setting

Keycode register must set this setting.

SOT0_0 setting by PPC_PCFGR register

SIN1_0 setting by PPC_PCFGR register

ADER_ADER0 register setting for AN2 pin

GPIO_DDR0 register setting for SOT0_0 output

GPIO_PORTEN0 register setting for port input enable

Operation mode/format setting

Reception initial setting and operation start

Transmission initial setting and operation start

3.1.3 UART Transmission-Side Settings of Example Program

Code 2 and Code 3 describe the UART transmission-side settings of the example program.

Code 2. UART Transmission-Side Settings of Example Program (for S6J3110 series)

```
 /* Asynchronous Serial Interface (Transmission) */
void Asynch_Uart0_TX(void)
{
   /* UART0 status initialize */
   CPG_MFS00_UART_SCR_UPCL=1;

   /* Operation mode setting (Asynchronous normal mode)*/
   CPG_MFS00_UART_SMR=0x00;

   /* Enable serial data output */
   CPG_MFS00_UART_SMR_SOE=1;

   /* Baud rate setting (115.2kbps, 36MHz) */
   CPG_MFS00_UART_BGR=311;

   /* UART0 transmission setting initialize */
   CPG_MFS00_UART_SCR=0x00;

   /* Data format setting (8bit length) */
   CPG_MFS00_UART_ESCR=0x00;

   /* Enable transmission interrupt */
   CPG_MFS00_UART_SCR_TIE=1;

   /* Enable transmission operation */
   CPG_MFS00_UART_SCR_TXE=1;
}
```

UART transmission-side initial setting

Operation initial setting

Baud rate setting

Data format setting

Transmission-side interrupt enable

Transmission-side operation enable

Code 3. UART Transmission-Side Settings of Interrupt Routine (for S6J3110 series)

```
/* UART0 Transmission interrupt routine */
FN_IRQ_DEFINE_BEGIN(Uart0_Tx_Interrupt, INTERRUPTS_IRQ_NUMBER_65)
{
     /* Transmission data register setting */
     CPG_MFS00_UART_TDR=tr_data[tr_num];

 }
     /* Data count */
     tr_num++;
        if(tr_num == 10){
           tr_num = 0;
        }
}
FN_IRQ_DEFINE_END()
```

Interrupt routine

Sets 1 byte of data to TDR for communication

3.1.4 UART Reception-Side Settings of Example Program

Code 4 and Code 5 describe the UART reception-side settings of the example program.

Code 4. UART Reception-Side Settings of Example Program (for S6J3110 series)

```
 /* Asynchronous Serial Interface (Reception) */
void Asynch_Uart1_RX(void)
{
   /* UART0 status initialize */
   CPG_MFS01_UART_SCR_UPCL=1;

   /* Operation mode setting (Asynchronous normal mode)*/
   CPG_MFS01_UART_SMR=0x00;

   /* Enable serial data output */
   CPG_MFS01_UART_SMR_SOE=1;

   /* Baud rate setting (115.2kbps, 36MHz) */
   CPG_MFS01_UART_BGR=311;

   /* UART0 setting initialize */
   CPG_MFS01_UART_SCR=0x00;

   /* Data format setting (8bit length) */
   CPG_MFS01_UART_ESCR=0x00;

   /* Enable reception interrupt */
   CPG_MFS01_UART_SCR_RIE=1;

   /* Enable reception operation */
   CPG_MFS01_UART_SCR_RXE=1;
}
```

UART reception-side initial setting

Operation initial setting

Baud rate setting

Data format setting

Reception-side interrupt enable

Reception-side operation enable

Code 5. UART Reception-Side Settings of Interrupt Routine (for S6J3110 series)

```
/* UART1 Reception interrupt routine */
FN_IRQ_DEFINE_BEGIN(Uart1_Rx_Interrupt, INTERRUPTS_IRQ_NUMBER_66)
{
/* Framing error */
    if(CPG_MFS01_UART_SSR_FRE==1){

              /* Reception error flag clear */
              CPG_MFS01_UART_SSR_REC=1;
       }
    /* Overrun error */
   else if(CPG_MFS01_UART_SSR_ORE==1){
              /* Reception error flag clear */
              CPG_MFS01_UART_SSR_REC=1;
    }
    else{
        /* Reception data register read */
        re_data[re_num]=CPG_MFS01_UART_RDR;

       /* Data count */
        re_num++;
             if(re_num == 10){
                re_num = 0;
               }
    }
}
FN_IRQ_DEFINE_END()
```

Interrupt routine

Framing and overrun error check and flag clearing

Reading reception data

3.2 CSIO Communication Settings

Figure 4 illustrates the CSIO communication connection.

Figure 4. Example CSIO Communication Connection

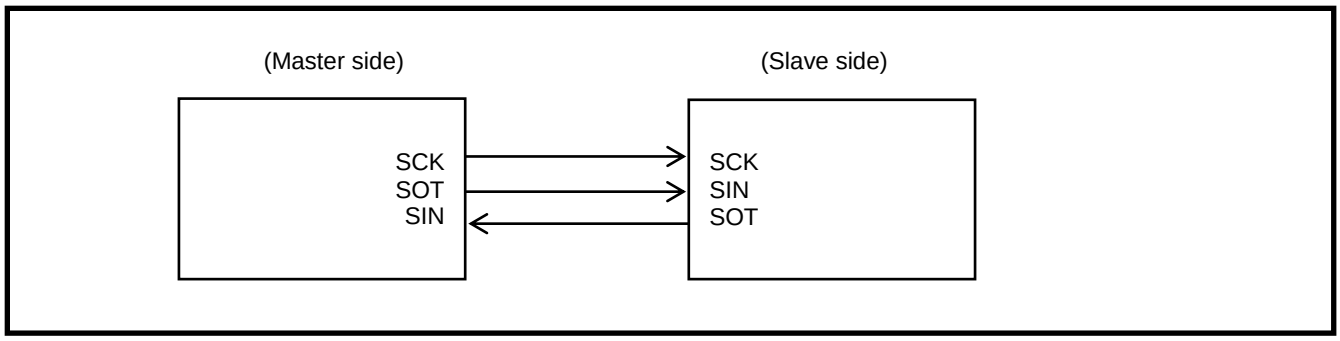


Figure 5 and Figure 6 illustrate example CSIO communication flow charts.

Figure 5. Example CSIO Communication Flow Chart (FIFO not used)

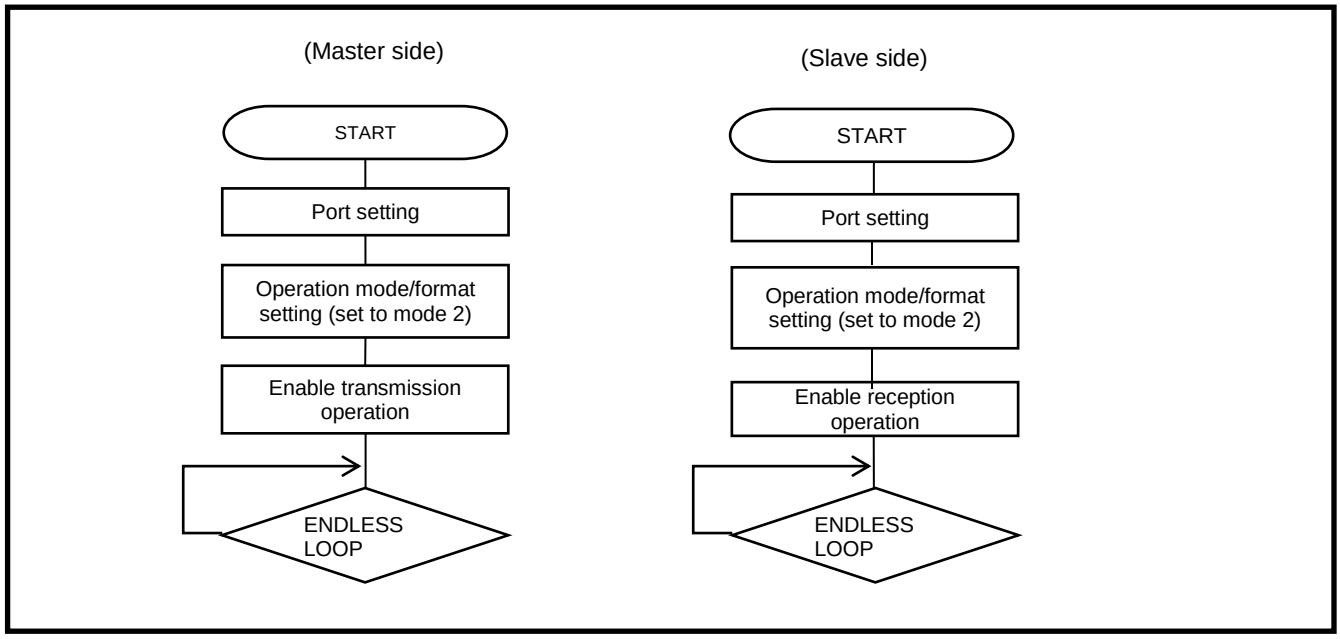
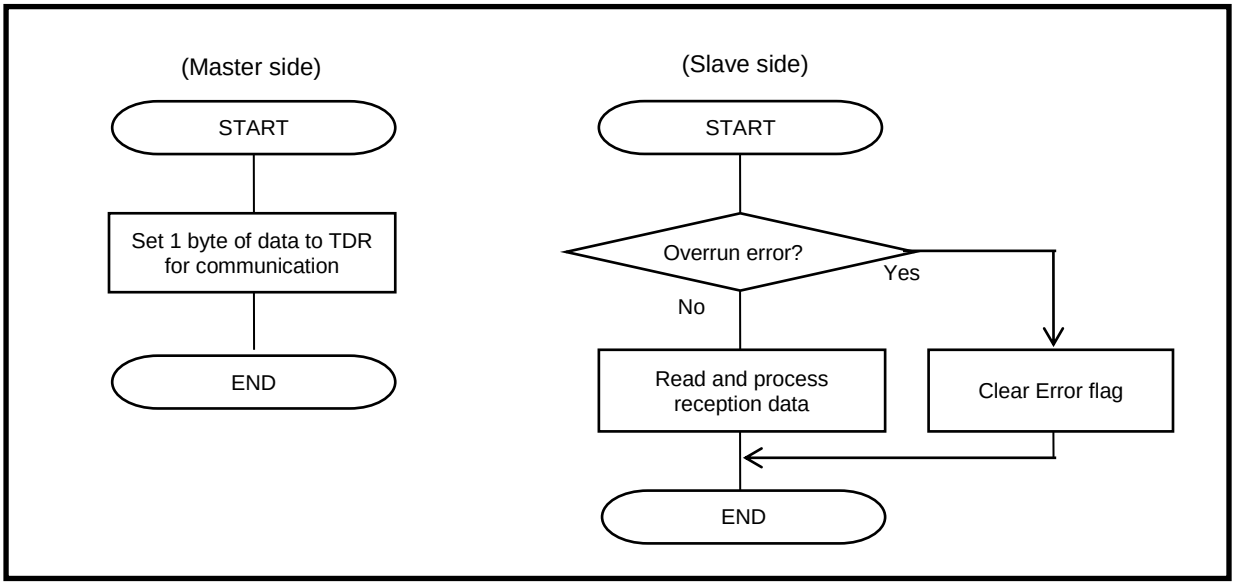


Figure 6. Example CSIO Communication Interrupt Routine Flow Chart (FIFO not used)



3.2.1 Baud Rate Setting

The BGR0/1 register sets the 15-bit reload counter. The baud rate calculation formulas are as follows.

- Calculation formulas of the reload value

  V = (f/b) – 1

  V: Reload value, f: Bus clock frequency or external clock frequency (Hz), b: Baud rate (bps)

- Calculation example of the reload value

  When the bus clock is 36 MHz, using the bus clock, the baud rate is 115200 bps. The reload value can be calculated as follows.

  - Reload value:

    V = (36000000 / 115200) – 1 = 311

    Therefore, the baud rate is

    b = (36000000 / (311+1) = 115384 bps

3.2.2 CSIO Master/Slave Port Settings of Example Program

Code 6 describes the CSIO master/slave port settings of the example program.

Code 6. CSIO Master/Slave Port Settings of Example Program (for S6J3110 series)

```
int main(void)
{   /* Keycode register setting for SOT0_0 */
    PPC_KEYCDR=0x10000028;
    PPC_KEYCDR=0x50000028;
    PPC_KEYCDR=0x90000028;
    PPC_KEYCDR=0xD0000028;
    PPC_PCFGR020=0xC002;

    /* Keycode register setting for SIN1_0 */
    PPC_KEYCDR=0x10000038;
    PPC_KEYCDR=0x50000038;
    PPC_KEYCDR=0x90000038;
    PPC_KEYCDR=0xD0000038;
    PPC_PCFGR028=0x1000;

    /* Keycode register setting for SCK0_0 */
    PPC_KEYCDR=0x1000002A;
    PPC_KEYCDR=0x5000002A;
    PPC_KEYCDR=0x9000002A;
    PPC_KEYCDR=0xD000002A;
    PPC_PCFGR021=0x1002;

   /* Keycode register setting for AN2 analog input disable */
   ADER_KEYCDR=0x20000804;
   ADER_KEYCDR=0x60000804;
   ADER_KEYCDR=0xA0000804;
   ADER_KEYCDR=0xE0000804;
   ADER_ADER0=0xFFFFFFFB

    /* Keycode register setting for SCK1_0 */
    PPC_KEYCDR=0x10000040;
    PPC_KEYCDR=0x50000040;
    PPC_KEYCDR=0x90000040;
    PPC_KEYCDR=0xD0000040;
    PPC_PCFGR100=0x1002;

    /* Keycode register setting for SOT0_0 output */
    GPIO_KEYCDR=0x20000204;
    GPIO_KEYCDR=0x60000204;
    GPIO_KEYCDR=0xA0000204;
    GPIO_KEYCDR=0xE0000204;
    GPIO_DDR0=0x00100000;

    /* Keycode register setting for port input enable */
    GPIO_KEYCDR=0x20000400;
    GPIO_KEYCDR=0x60000400;
    GPIO_KEYCDR=0xA0000400;
    GPIO_KEYCDR=0xE0000400;
    GPIO_PORTEN=0x00000001;

 /* CSIO ch1 Rx */
     Synch_CSIO1_RX();

 /* CSIO ch0 Tx */
     Synch_CSIO0_TX();


     for(;;){/* Endless loop */
      ClearWatchdog();}
 }
```

Port setting

Keycode register must be set to the PPC_PCFGR, ADER_ADER, GPIO_DDR, GPIO_PORTEN register.

SOT0_0 setting by PPC_PCFGR register

SIN1_0 setting by PPC_PCFGR register

SCK0_0 setting by PPC_PCFGR register

ADER_ADER0 register setting for AN2 pin

SCK1_0 setting by PPC_PCFGR register

GPIO_DDR0 register setting for SOT0_0 output

GPIO_PORTEN0 register setting for port input enable

Operation mode/format setting

Reception initial setting and operation start

Transmission initial setting and operation start

3.2.3 CSIO Master-Side Settings of Example Program

Code 7 and Code 8 describe the CSIO master-side settings of the example program.

Code 7. CSIO Master-Side Settings of Example Program (for S6J3110 series)

```
/* Clock Synchronous Serial Interface (Transmission)*/
void Synch_CSIO0_TX(void)
{
   /* CSIO0 status initialize */
   CPG_MFS00_CSIO_SCR_UPCL=1;

   /* Operation mode setting (Clock synchronous mode)*/
   CPG_MFS00_CSIO_SMR=0x40;

   /* Enable serial clock output  */
   CPG_MFS00_CSIO_SMR_SCKE=1;

   /* Enable serial data output */
   CPG_MFS00_CSIO_SMR_SOE=1;

   /* Baud rate setting (115.2kbps, 36MHz)*/
   CPG_MFS00_CSIO_BGR=311;

   /* CSIO0 setting initialize (Master)*/
   CPG_MFS00_CSIO_SCR=0x00;

   /* Data format setting (8bit length)*/
   CPG_MFS00_CSIO_ESCR=0x00;

   /* Enable transmission interrupt */
   CPG_MFS00_CSIO_SCR_TIE=1;

   /* Enable transmission operation */
   CPG_MFS00_CSIO_SCR_TXE=1;
}
```

CSIO transmission-side initial setting

Operation initial setting

Baud rate setting

Data format setting

Master-side interrupt enable

Master-side operation enable

Code 8. CSIO Master-Side Settings of Interrupt Routine (for S6J3110 series)

```
/* CSIO0 Transmission interrupt routine */
FN_IRQ_DEFINE_BEGIN(Uart0_Tx_Interrupt, INTERRUPTS_IRQ_NUMBER_65)
{
    /* Transmission data register setting */
     CPG_MFS00_CSIO_TDR=tr_data[tr_num];


     tr_num++;
     if(tr_num == 10){
         tr_num = 0;
        }
}
FN_IRQ_DEFINE_END()
```

Interrupt routine

Sets 1 byte of data to TDR for communication

3.2.4 CSIO Slave-Side Settings of Example Program

Code 9 and Code 10 describe the CSIO slave-side settings of the example program.

Code 9. CSIO Slave-Side Settings of Example Program (for S6J3110 series)

```
/* Clock Synchronous Serial Interface (Reception)*/
void Synch_CSIO1_RX(void)
{
   /* CSIO1 status initialize */
   CPG_MFS01_CSIO_SCR_UPCL=1;

   /* Operation mode setting (Clock synchronous mode)*/
   CPG_MFS01_CSIO_SMR=0x40;

   /* Enable serial data output */
   CPG_MFS01_CSIO_SMR_SOE=1;

   /* CSIO1 setting initialize (Slave)*/
   CPG_MFS01_CSIO_SCR=0x40;

   /* Data format setting (8bit length) */
   CPG_MFS01_CSIO_ESCR=0x00;

   /* Enable reception interrupt */
   CPG_MFS01_CSIO_SCR_RIE=1;

   /* Enable reception operation */
   CPG_MFS01_CSIO_SCR_RXE=1;
}
```

CSIO reception-side initial setting

Operation initial setting

Data format setting

Reception-side interrupt enable

Reception-side operation enable

Code 10. CSIO Slave-Side Settings of Interrupt Routine (for S6J3110 series)

```
/* CSIO1 Rx interrupt routine */
FN_IRQ_DEFINE_BEGIN(Uart1_Rx_Interrupt, INTERRUPTS_IRQ_NUMBER_66)
{
    /* Overrun error */
     if(CPG_MFS01_CSIO_SSR_ORE==1){

               /* Reception error flag clear */
               CPG_MFS01_CSIO_SSR_REC=1;
     }
     else{
      /* Receive data register read */
      re_data[re_num]=CPG_MFS01_CSIO_RDR;

               /* Data count */
               re_num++;
               if(re_num == 10){
                  re_num = 0;
                     }
            }
}
FN_IRQ_DEFINE_END()
```

Interrupt routine

Overrun error check and flag clearing

Reading reception data

3.3 LIN Communication Settings

Figure 7 shows an example of LIN communication connection.

Figure 7. Example of LIN Communication Connection

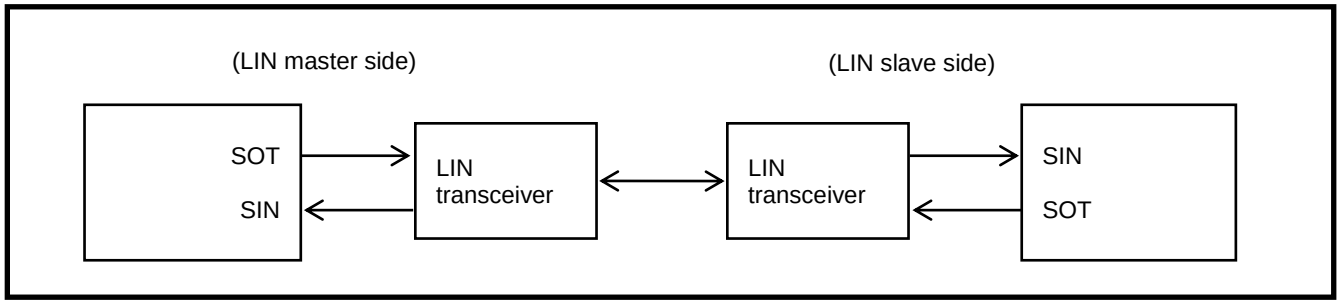
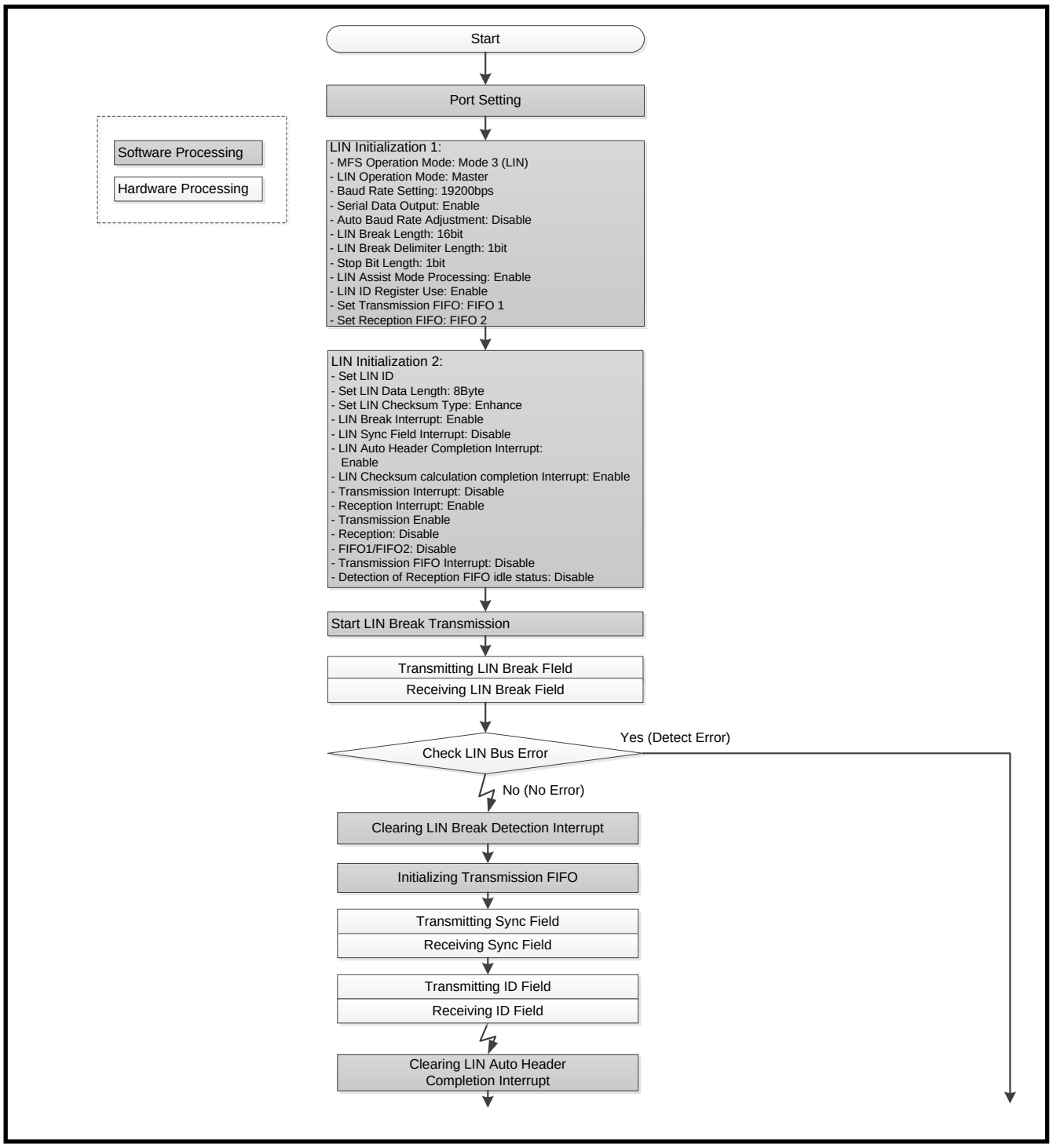


Figure 8 and Figure 9 show examples of the LIN communication flow chart.

Figure 8. Example of LIN Communication Master Mode (LIN hardware assist mode and FIFO used)



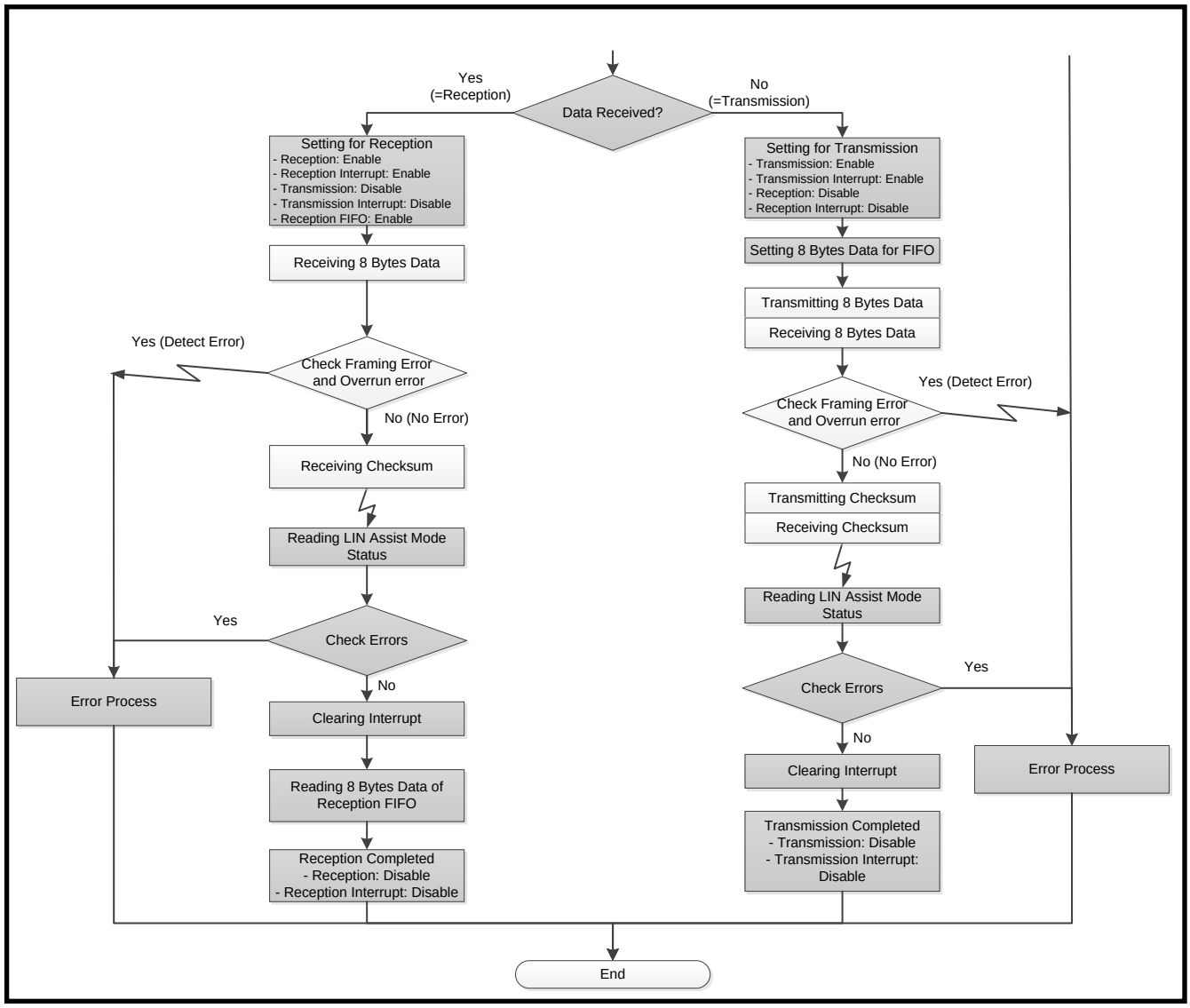
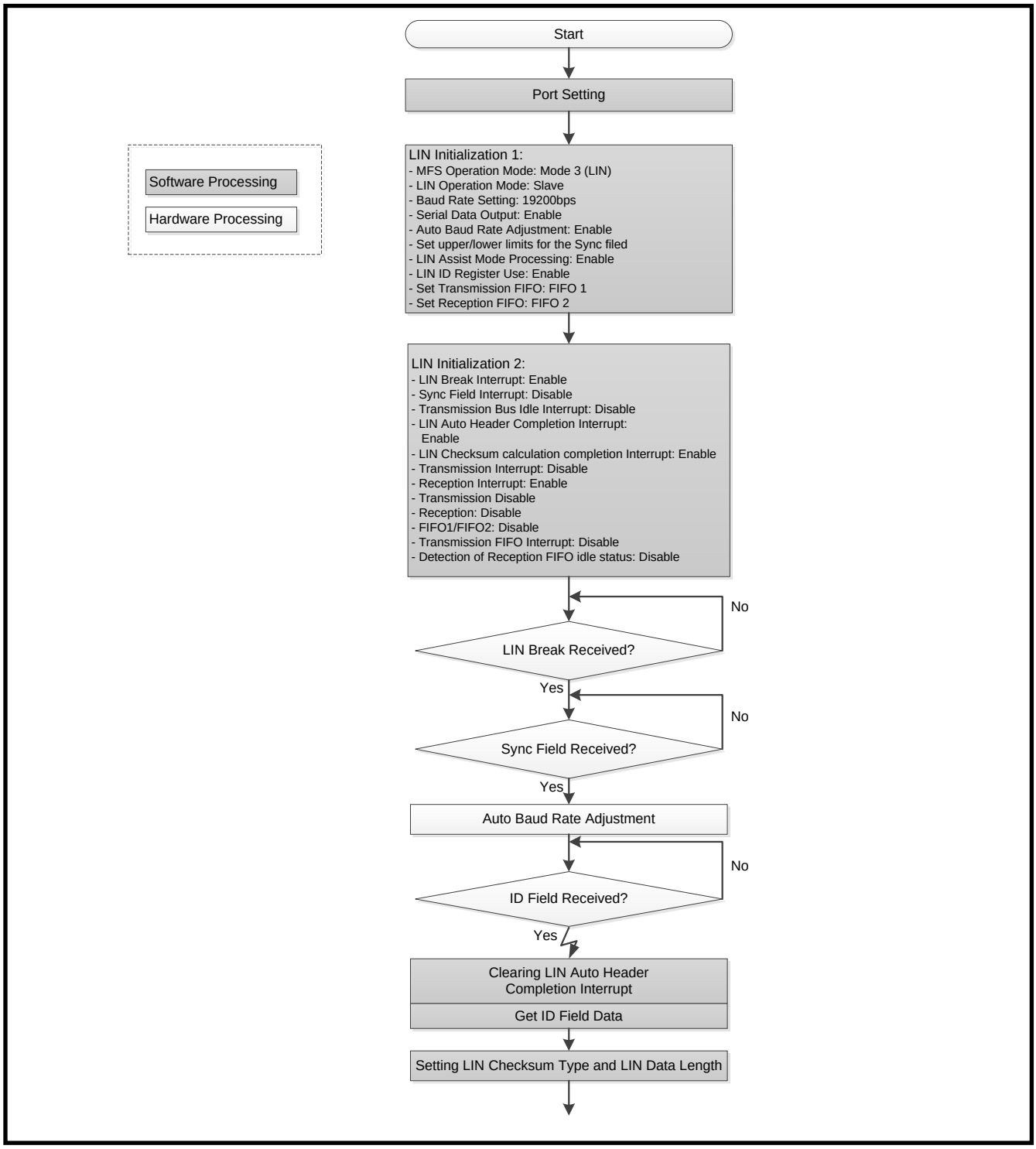
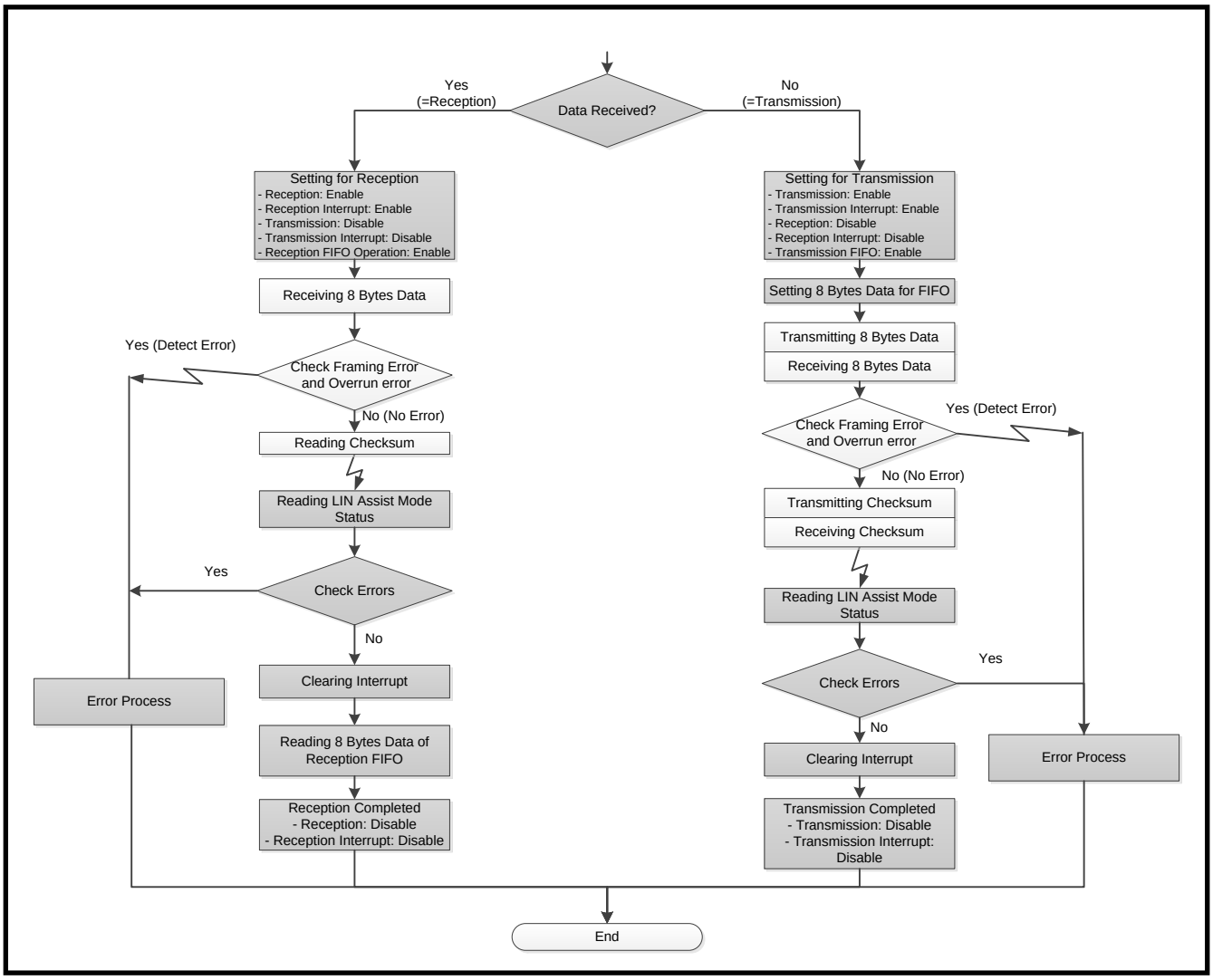

Data Received?
Yes
(=Reception)
No
(=Transmission)
Setting for Reception
- Reception: Enable
- Reception Interrupt: Enable
- Transmission: Disable
- Transmission Interrupt: Disable
- Reception FIFO: Enable
Receiving 8 Bytes Data
Check Framing Error and Overrun error
Yes (Detect Error)
No (No Error)
Receiving Checksum
Reading LIN Assist Mode Status
Check Errors
Yes
No
Error Process
Clearing Interrupt
Reading 8 Bytes Data of Reception FIFO
Reception Completed
- Reception: Disable
- Reception Interrupt: Disable
Setting for Transmission
- Transmission: Enable
- Transmission Interrupt: Enable
- Reception: Disable
- Reception Interrupt: Disable
Setting 8 Bytes Data for FIFO
Transmitting 8 Bytes Data
Receiving 8 Bytes Data
Check Framing Error and Overrun error
Yes (Detect Error)
No (No Error)
Transmitting Checksum
Receiving Checksum
Reading LIN Assist Mode Status
Check Errors
Yes
No
Clearing Interrupt
Transmission Completed
- Transmission: Disable
- Transmission Interrupt: Disable
Error Process
End

Figure 9. Example of LIN Communication Slave Mode (LIN hardware assist mode and FIFO used)





3.3.1 Baud Rate Setting

The BGR0/1 register sets the 15-bit reload counter. The baud rate calculation formulas are as follows.

- Calculation formulas of the reload value

  V = (f/b) – 1

  V: Reload value, f: Bus clock frequency or external clock frequency (Hz), b: Baud rate (bps)

- Calculation example of the reload value

  When the bus clock is 36 MHz, using the bus clock, the baud rate is 19200 bps. The reload value can be calculated as follows.

  - Reload value:

    V = (36000000 / 19200) – 1 = 1874

    Therefore, the baud rate is

    b = (36000000 / (1874+1) = 19200 bps

3.3.2 LIN Master/Slave Port Settings of Example Program

Code 11 describes the LIN master/slave port settings of the example program.

Code 11. LIN Master/Slave Port Settings of Example Program (for S6J3110 series)

```
/* Initialize port setting */
static void SampleLinPortInit(void)
{
    stc_port_pin_config_t stcPinConf = { PortOutputResourceGPIO };

    /* SIN3 setting for master side */
    stcPinConf.bInputEnable = TRUE; /* Enable input */
    stcPinConf.bNoiseFilterEnable = FALSE;
    stcPinConf.enGpioDirection = PortGpioInput;
    stcPinConf.enGpioInitOutputLevel = PortGpioHigh;
    stcPinConf.enInputLevel = PortInputLevelCmosA;
    stcPinConf.enOutputDrive = PortOutputDriveA;
    stcPinConf.enOutputFunction = PortOutputResourceGPIO;
    stcPinConf.enPullResistor = PortPullResistorNone;
    (void)Port_SetPinConfig(0, 5, &stcPinConf);

    /* SIN2_1 setting for slave side */
    (void)Port_SelectInputPort(6, PortInputPortB); /*Select input port SIN2_1*/
    (void)Port_SetPinConfig(4, 21, &stcPinConf);

    /* SOT3 setting for master side */
    stcPinConf.bInputEnable = FALSE;
    stcPinConf.enGpioDirection = PortGpioOutput;
    stcPinConf.enOutputFunction = PortOutputResourceC;
    (void)Port_SetPinConfig(0, 6, &stcPinConf);

    /* SOT2_1 setting for slave side */
    stcPinConf.enOutputFunction=PortOutputResourceD; /*Select output port SOT2_1*/
    (void)Port_SetPinConfig(0, 0, &stcPinConf);

    /* For LIN transceiver */
    /* P007 = LIN NSLP1 */
    stcPinConf.bInputEnable = FALSE;
    stcPinConf.enGpioDirection = PortGpioOutput;
    stcPinConf.enOutputFunction = PortOutputResourceGPIO;
    (void)Port_SetPinConfig(0, 7, &stcPinConf);
    /* P001 = LIN NSLP2 */
    (void)Port_SetPinConfig(0, 1, &stcPinConf);

    (void)Port_EnableInput();
}
```

SIN3_0 setting by PPC_PCFGR register

SIN2_1 setting by PPC_PCFGR register

SOT3_0 setting by PPC_PCFGR register

SOT2_1 setting by PPC_PCFGR register

P007 pin setting for LIN transceiver NSLP1

P001 pin setting for LIN transceiver NSLP2

GPIO_PORTEN0 register setting for port input enable

3.3.3 LIN Master-Side Initial Settings of Example Program

Code 12 describes the LIN master-side initial settings of the example program.

Code 12. LIN Master-Side Initial Settings of Example Program (for S6J3110 series)

```
/* Initialize LIN setting */
static en_result_t SampleLinInit(void)
{
    /* Initialize LIN ch 3 (Master) */
     stc_lin_config_t stcLinConfig;
    /* Master mode */
     stcLinConfig.bMasterMode = TRUE;
    /* Break length=16bits*/
     stcLinConfig.enBreakFieldLength
           = LinBreakFieldLength16bits;
    /*Break delimiter length = 1bit */
     stcLinConfig.enBreakDelimiterLength
           = LinBreakDelimiterLength1bits;
     /* Stop bit = 1bit */
     stcLinConfig.enStopBit = LinOneStopBit;
     /* Baud rate = 19200bps */
     stcLinConfig.u32DataRate = 19200;
     /* Status callback */
     stcLinConfig.pfnStatusCb = SampleStatusCallBack_ch3;
     enResult = Lin_Init(&CPG_MFS03_LIN, &stcLinConfig);
}
```

Master-side setting in LIN channel 3

LIN sync break length setting

LIN stop bit length setting

Baud rate setting

Operation mode, LIN assist mode setting

Interrupt enable setting (reception error, auto header completion, checksum completion, LIN break detection)

3.3.4 LIN Slave-Side Initial Settings of Example Program

Code 13 describes the LIN slave-side initial settings of the example program.

Code 13. LIN Slave-Side Initial Settings of Example Program (for S6J3110 series)

```
/* Initialize LIN setting */
static en_result_t SampleLinInit(void)
{
        /* Initialize LIN ch 2 (Slave) */
        stcLinConfig.bMasterMode = FALSE;/* Slave mode */
       /* Auto adjustment upper limit (maximum value)*/
       stcLinConfig.u16BgrUpperLimit = 0x7FFF;
       /* Auto adjustment lower limit (minimum value) */
       stcLinConfig.u16BgrLowerLimit = 0x3;
       /* Status callback */
        stcLinConfig.pfnStatusCb = SampleStatusCallBack_ch2;
       /* Baud rate = 19200bps */
        stcLinConfig.u32DataRate = 19200;
       enResult = Lin_Init(&CPG_MFS02_LIN, &stcLinConfig);
 }
```

Slave-side setting in LIN channel 2

(LIN sync break length, stop bit, baud rate settings are same as master side.)

Operation mode, LIN assist mode, interrupt settings are same as master side.

3.3.5 LIN Master/Slave Operation Settings of Example Program

Code 14 describes the LIN master/slave operation settings of the example program.

Code 14. LIN Master/Slave Operation Settings of Example Program (for S6J3110 series)

```
int main(void)
  {
      /* Initialize port setting */
      (void)SampleLinPortInit();

      /* Initialize LIN */
      enResult = SampleLinInit();
      if (enResult == Ok)
      {
          SampleLinCh2Status = NoOperation;
          SampleLinCh3Status = NoOperation;

          /* Send Master to Slave */
          /* Start auto header */
          (void)Lin_SetAutoHeader(&CPG_MFS03_LIN, SAMPLE_ID_MASTER_TO_SLAVE,
              SAMPLE_DATA_LENGTH, LinChecksumTypeExtended);

          /* Wait until auto header complete */
          while (SampleLinCh2Status != CompleteAutoHeader);

          /* Get ID */
          (void)Lin_GetReceivedId(&CPG_MFS02_LIN, &u8LinCh2ReceivedId,
                 &u8LinCh2ReceivedParity);

          /* Enable reception */
          (void)Lin_EnableRx(&CPG_MFS02_LIN, SAMPLE_DATA_LENGTH,
                  LinChecksumTypeExtended);

          /* Write Data Master to slave */
          (void)Lin_WriteData(&CPG_MFS03_LIN, au8LinCh3TxData, SAMPLE_DATA_LENGTH,
                  LinChecksumTypeExtended);

          /* Wait until checksum calculation complete */
          while (SampleLinCh2Status != CompleteChecksum);

          /* Read response data
          (void)Lin_ReadData(&CPG_MFS02_LIN, au8LinCh2RxBuf,
           &u8LinCh2ReceivedLength);

          /* Finally disable Rx */
          (void)Lin_DisableRx(&CPG_MFS02_LIN);

          /* Finally disable Tx */
          (void)Lin_DisableTx(&CPG_MFS03_LIN);

          /* Wait time for next frame (usually done by scheduler
          for (uint32_t u32Cnt = 0; u32Cnt < 500000; u32Cnt++)
          {
              NOP();
          }
      }
       /* Endless loop */
      for(;;)
      {
          ClearWatchdog();
      }
  }
```

Port settings for master side (SIN3_0, SOT3_0) and slave side (SIN2_1, SOT2_1)

LIN header format and baud rate settings

LIN ID, data length, checksum type settings in the master

Auto header reception waiting in the slave

Reception LIN ID and parity bit in the slave

Reception enable, FIFO enable settings in the slave

Transmission data setting and transmission enable in the master

Checksum completion checking in the slave

Data reception in the slave

Reception disable in the slave

Transmission disable in the master

Next frame waiting

Code 15. LIN Master-Side Interrupt Operation Settings of Example Program (for S6J3110 series)

```
/* Status callback function for LIN ch3 (master) */
static void SampleStatusCallBack_ch3(un_lin_interrupt_trigger_t unIntTrigger,
                                     un_lin_detected_error_t    unDetectedError)
{
    /* Detected Error */
    if (unIntTrigger.stcBits.DetectError != 0)
    {
        /* Implement error handling code here */
        /* Save error flag */
        unLinCh3Error.u8Byte = unDetectedError.u8Byte;
        SampleLinCh2Status = DetectError;
    }
    /* Checksum complete */
    else if(unIntTrigger.stcBits.CompleteChecksum != 0)
    {
        SampleLinCh2Status = CompleteChecksum;
    }
    /* Header complete */
    else if(unIntTrigger.stcBits.CompleteAutoHeader != 0)
    {
        SampleLinCh2Status = CompleteAutoHeader;
    }
    /* Break is detected */
    else if(unIntTrigger.stcBits.DetectBreak != 0)
    {
        SampleLinCh2Status = DetectBreak;
    }
    else
    {
        /* do nothing */
    }
}
```

Master-side callback function by interrupt

Framing error, overrun error interrupt case

Checksum complete interrupt case

Auto header complete interrupt case

LIN sync break detection interrupt case

Code 16. LIN Slave-Side Interrupt Operation Settings of Example Program (for S6J3110 series)

```
/* Status callback function for LIN ch2 (slave) */
static void SampleStatusCallBack_ch2(un_lin_interrupt_trigger_t unIntTrigger,
                                     un_lin_detected_error_t    unDetectedError)
{
    /* Detected Error */
    if (unIntTrigger.stcBits.DetectError != 0)
    {
        /* Implement error handling code here */
        /* Save error flag */
        unLinCh2Error.u8Byte = unDetectedError.u8Byte;
        SampleLinCh2Status = DetectError;
    }
    /* Checksum complete */
    else if(unIntTrigger.stcBits.CompleteChecksum != 0)
    {
        SampleLinCh2Status = CompleteChecksum;
    }
    /* Header complete */
    else if(unIntTrigger.stcBits.CompleteAutoHeader != 0)
    {
        SampleLinCh2Status = CompleteAutoHeader;
    }
    /* Break is detected */
    else if(unIntTrigger.stcBits.DetectBreak != 0)
    {
        SampleLinCh2Status = DetectBreak;
    }
    else
    {
        /* do nothing */
    }
}
```

Slave-side callback function by interrupt

Interrupt case settings are same as master side.

3.4 I²C Communication Settings

Figure 10 shows an example of the I²C communication control interface.

Figure 10. Example of I²C Communication Connection

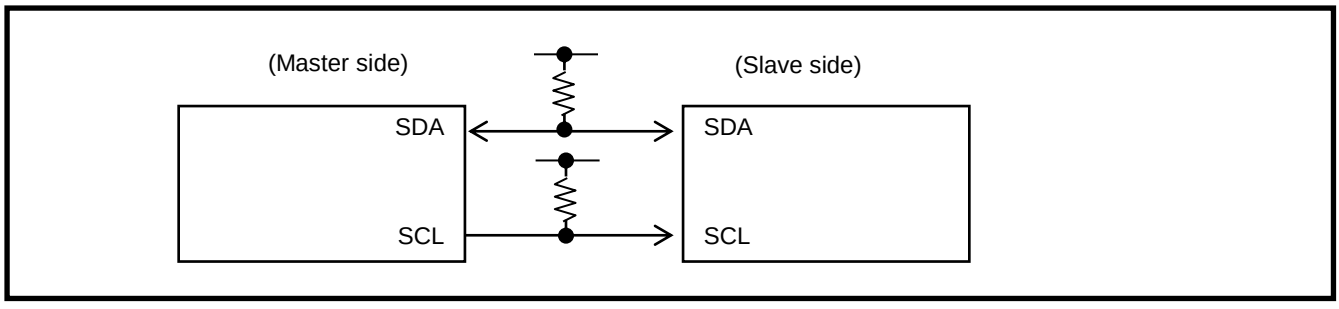


Figure 11, Figure 12, and Figure 13 show examples of the I²C communication flow chart.

Figure 11. Example of I²C Communication Flow Chart (DMA and FIFO not used) (1 of 3)

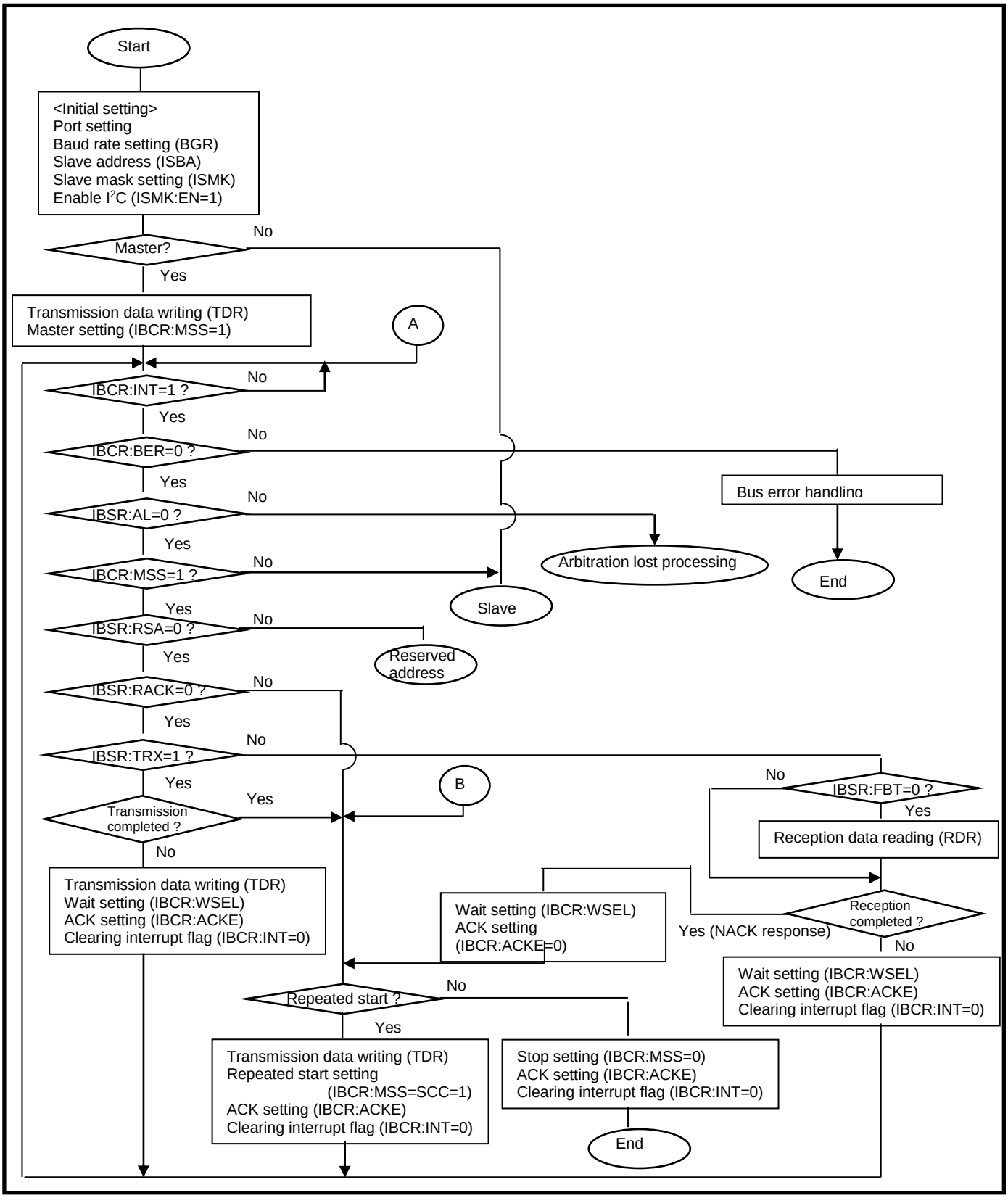


Figure 12. Example of I²C Communication Flow Chart (DMA and FIFO not used) (2 of 3)

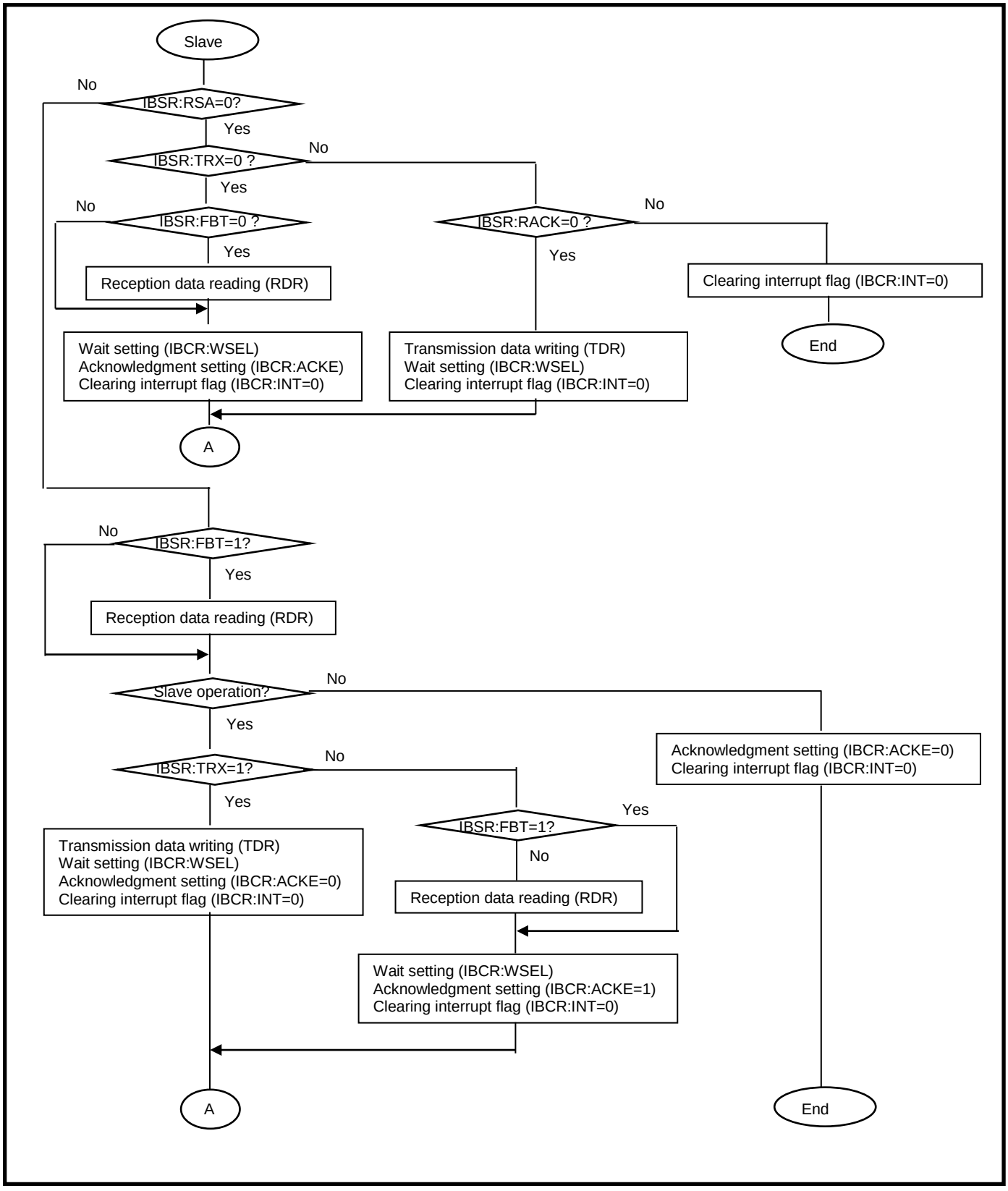
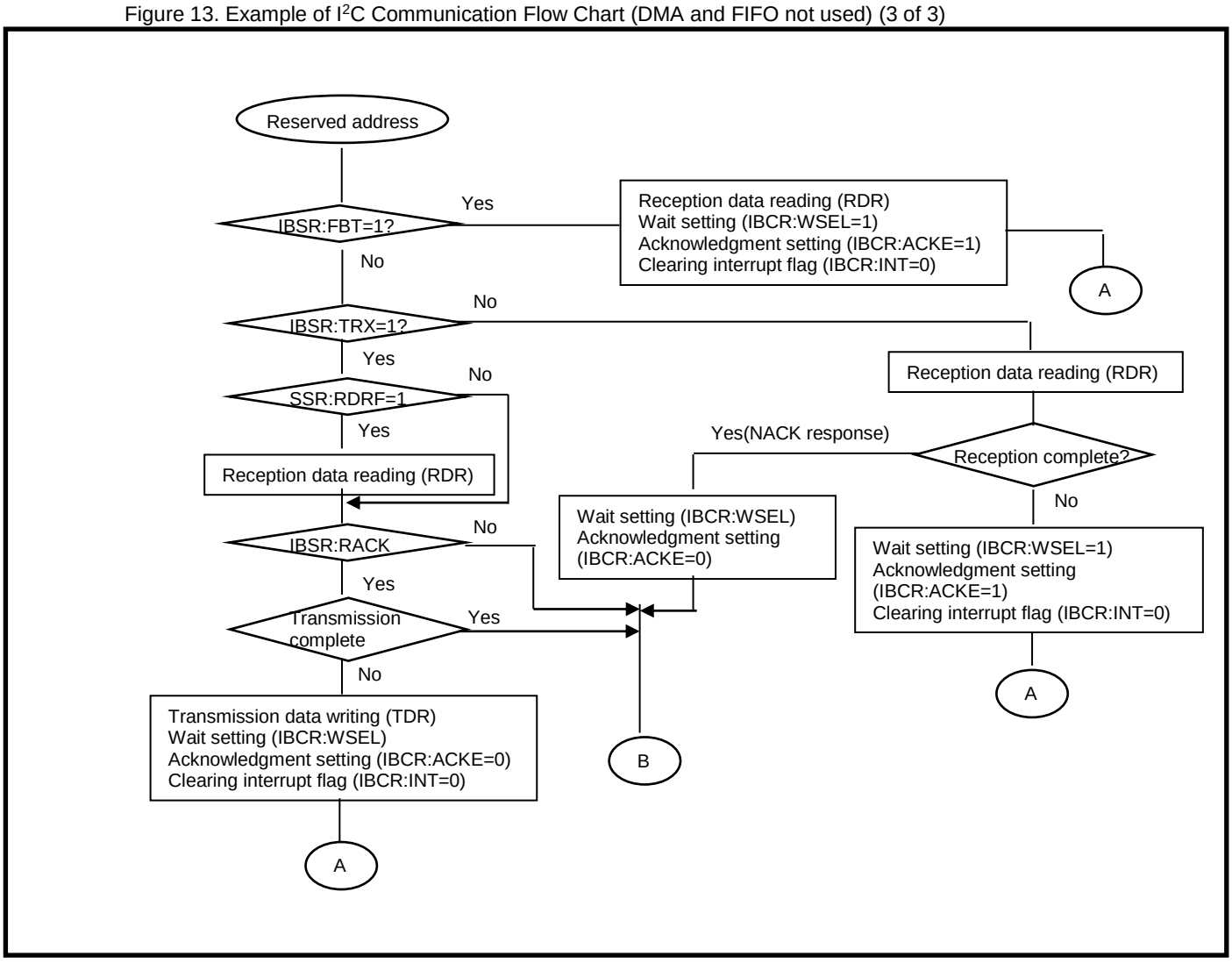


Figure 13. Example of I²C Communication Flow Chart (DMA and FIFO not used) (3 of 3)



3.4.1 Baud Rate Setting

The BGR0/1 register sets the 15-bit reload counter. The baud rate calculation formulas are as follows.

Calculation formulas of the reload value

V = (f/b) – 1

V: Reload value, f: Bus clock frequency or external clock frequency (Hz), b: Baud rate (bps)

Calculation example of the reload value

When the bus clock is 16 MHz, using the bus clock, the baud rate is 100 kbps. The reload value can be calculated as follows.

Reload value:

V = (16000000 / 100000) – 1 = 159

Therefore, the baud rate is

b = (16000000 / (159+1)) = 100 kbps

3.4.2 I²C Port Settings of Example Program

Code 17 describes the I²C port settings of the example program.

Code 17. I²C Port Settings of Example Program (for S6J3110 series)

```
/* Port settings */
PpcConf.bInputEnable = TRUE;
PpcConf.bNoiseFilterEnable = FALSE;
PpcConf.enGpioDirection = PortGpioOutput;
PpcConf.enInputLevel = PortInputLevelCmosA;
PpcConf.enOutputDrive = PortOutputDriveA;
PpcConf.enOutputFunction = PortOutputResourceG;
PpcConf.enPullResistor = PortPullResistorNone;
PpcConf.enGpioInitOutputLevel = PortGpioLow;

/* SDA0_0 (P020) */
Port_SetPinConfig(0, 20, &PpcConf);
/* SCL0_0 (P021) */
 Port_SetPinConfig(0, 21, &PpcConf);
```

SDA0_0, SCL0_0 setting by PPC_PCFGR register

3.4.3 I²C Master-Side Initial Settings of Example Program

Code 18 describes the I²C master-side initial settings of the example program.

Code 18. I²C Master-Side Initial Settings of Example Program (for S6J3110 series)

```
/* I2C settings */
stc_i2c_config_t stcI2cConfig = {
    .u32DataRate             = 100000,
    .u8SlaveAddress          = I2C_SLAVE_ADDR_STD,
    .bSlaveAddressEnable     = TRUE,
    .bAcknowledgeEnable      = FALSE,
    .bWaitSelect             = FALSE,
    .bInterruptEnable        = FALSE,
    .bDmaModeEnable          = FALSE,
};

/* I2C initialize */
I2c_Init(&CPG_MFS00_I2C, &stcI2cConfig);

/* write transmission buffer 'write_buf[]' */
writeBuf[0] = 0x11;
writeBuf[1] = 0x22;
writeBuf[2] = 0x33;
writeBuf[3] = 0x44;

/* Write I2C */
I2c_WriteSequence(&CPG_MFS00_I2C, I2C_SLAVE_ADDR_STD, 0xAA, writeBuf, 4);
```

I²C baud rate and slave address setting

Transmission data writing

3.4.4 I²C Slave-Side Initial Settings of Example Program

Code 19 describes the I²C slave-side initial settings of the example program.

Code 19. I²C Slave-Side Initial Settings of Example Program (for S6J3110 series)

```
/* I2C settings */
stc_i2c_config_t stcI2cConfig = {
    .u32DataRate             = 100000,
    .u8SlaveAddress          = I2C_SLAVE_ADDR_STD,
    .bSlaveAddressEnable     = TRUE,
    .bAcknowledgeEnable      = TRUE,
    .bWaitSelect             = FALSE,
    .bInterruptEnable        = FALSE,
    .bDmaModeEnable          = FALSE,
};

/* I2C initialize */
I2c_Init(&CPG_MFS00_I2C, &stcI2cConfig);

/* Slave address */
CPG_MFS00_I2C_ISBA_SA = I2C_SLAVE_ADDR_STD >> 1;

/* Enable I2C */
CPG_MFS00_I2C_ISMK_EN = 1;
```

I²C baud rate, slave address and acknowledge enable setting

I²C communication enable

3.4.5 I²C Master/Slave Operation Settings of Example Program

Code 20 describes the I²C master/slave operation settings of the example program.

Code 20. I²C Master/Slave Operation Settings of Example Program (for S6J3110 series)

```
int main(void)
{
    /* SDA0_0, SCL0_0 Port settings */


    /* I2C initial settings (Master or Slave) */


    /* Endless loop */
    for (;;)
    {
     #if M_S_SELECTION == SLAVE
        if (CPG_MFS00_I2C_IBCR_INT == 1)
        {
            /* Check for Reserved start address is detected */
            if ((CPG_MFS00_I2C_IBSR_RSA == 1) ||
                (CPG_MFS00_I2C_IBSR_TRX == 1))
            {
                /* clear transfer end interrupt flag */
                CPG_MFS00_I2C_IBCRC_INTC   = 1;
            }
            else
            {
                /* Read I2C */
                if (readCnt < READ_COUNT_MAX)
                {
                    I2c_Read(&CPG_MFS00_I2C, &readValue);

                    readBuf[readCnt] = readValue;
                    readCnt++;
                }
            }
        }
      #endif
        ClearWatchdog();
    }
}
```

Refer to 3.4.2

Refer to 3.4.3 and 3.4.4

Slave-side settings only

Data transmission or data address detection

Data reception

4 Summary

The multifunction serial interface supports the serial communication function used mainly in automotive applications. This function allows bidirectional communication between devices and offers a flexible network.

The Cypress S6J3110/3120/3200/3310/3320/3330/3340/3350/3360/3370/3400 series supports the multifunction serial interface. This application note guided you in implementing the multifunction serial interface with the S6J3110/3120/3200/3310/3320/3330/3340/3350/3360/3370/3400 series smoothly.

5 Related Documents

Traveo family series datasheets and hardware manuals:

- S6J311E/D/C/B Series Datasheet (Doc. No.002-05681)
- S6J311A/9/8 Series Datasheet (Doc. No.002-04632)
- S6J3110 Series Hardware Manual (Doc.No.002-10667)
- S6J3120 Series Datasheet (Doc.No.002-04863)
- S6J3120 Series Hardware Manual (Doc.No.002-04855)
- S6J3200 Series Datasheet (Doc.No.002-05682)
- S6J3200 Series Hardware Manual (Doc.No.002-04852)
- S6J32E/F/G Series Datasheet (Doc.No.002-10689)
- S6J32E/F/G Series Hardware Manual (Doc.No.002-12500)
- Traveo Family Hardware Manual Platform Part for S6J3200 Series (Doc.No.002-04854)
- S6J3310/20/30/40 Series Datasheet (Doc.No.002-10635)
- S6J3350 Series Datasheet (Doc.No.002-10634)
- S6J3310/20/30/40/50 Series Hardware Manual (Doc.No.002-10185)
- Traveo Family HardwareManual Platform Part for S6J3310/3320/3330/3340/3350 Series (Doc.No.002-07884)
- S6J3360/70 Series Datasheet (Doc.No.002-03359)
- S6J3360/70 Series Hardware Manual (Doc.No.002-18302)
- Traveo Family HardwareManual Platform Part for S6J3360/3370 Series (Doc.No.002-07884)
- S6J3400 Series Datasheet (Doc.No.001-97829)
- S6J3400 Series Hardware Manual (Doc.No.002-09919)
- Traveo Family Hardware Manual Platform Part for S6J3400 Series (Doc.No.002-07884)

Document History

Document Title: AN202495 – Using the Multifunction Serial Interface of the Traveo™ Family

Document Number: 002-02495

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	4798907	KSUN	10/27/2015	New application note
*A	5167685	KSUN	03/18/2016	Added S6J3200/S6J3300/S6J3350 series. Added I^2C function.
*B	5352690	MKAR	08/05/2016	Added S6J3400 series. Updated template
*C	5703200	MKAR	04/20/2017	Added S6J3360/S6J3370 series. Updated template

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709