Freescale Semiconductor
Application Note

Document Number: AN4162
Rev. 0, 08/2010

SPI Driver for the MC9S08GW64

by: **Tanya Malik**
Reference Design and Applications Group
Noida
India

Contents

1 Introduction

This document describes a driver for the Serial Peripheral Interface (SPI). This allows users to customize all the possible configurations for this peripheral.

The software architecture is designed to provide seamless migration between devices that have the same peripheral module.

In this application note, the driver interfaces are explained. Various applications for the MC9S08GW64 can make use of this driver. The following sections describe the details and steps for creating an application using the SPI driver.

2 Serial Peripheral Interface (SPI)

- SPI is an 8-bit synchronous serial data link standard that operates in full duplex mode.
- The SPI devices communicate in master and slave mode where the master device initiates the transaction.
- There can be a single master with multiple slaves.

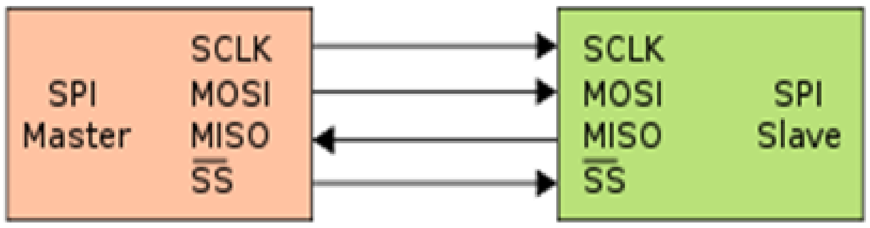


Figure 1. SPI communication

2.1 SPI in MC9S08GW64

There are three SPI in the MC9S08GW64—SPI0, SPI1, and SPI2. SPI0 is designed for an Automatic Meter Reading (AMR) operation by making the pins SS, SCLK, MISO (if slave), and MOSI (if master) open drain. Open drain circuits are used to interface different families of devices that have different operating logic voltage levels or to control external circuitry that requires a higher voltage level. Thus SPI0 can be compatible with devices with higher voltage levels such as 5 V devices.

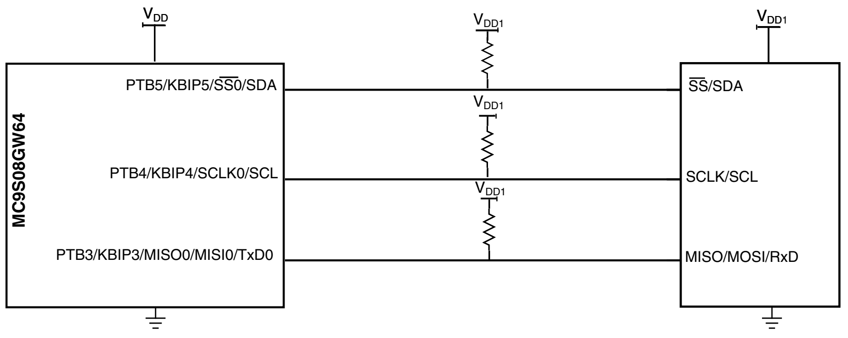


Figure 2. AMR communication

This figure shows how the SPI0 can communicate with a device with operating voltage V_{DD1}.

2.2 Clock Gating In SPI

SPI module clock gating is controlled by SCGC2 (SCGC2_SPI0, SCGC2_SPI1, SCGC2_SPI2). On Reset the clock is gated to all the SPI blocks.

2.3 BaudRate Generation

The three pre-scale bits (SPPR2:SPPR1:SPPR0) choose a pre-scale divisor of 1, 2, 3, 4, 5, 6, 7, or 8. The three rate select bits (SPR3:SPR2:SPR1:SPR0) divide the output of the pre-scale stage by 2, 4, 8, 16, 32, 64, 128, 256, or 512 to get the internal SPI master mode bit-rate clock.

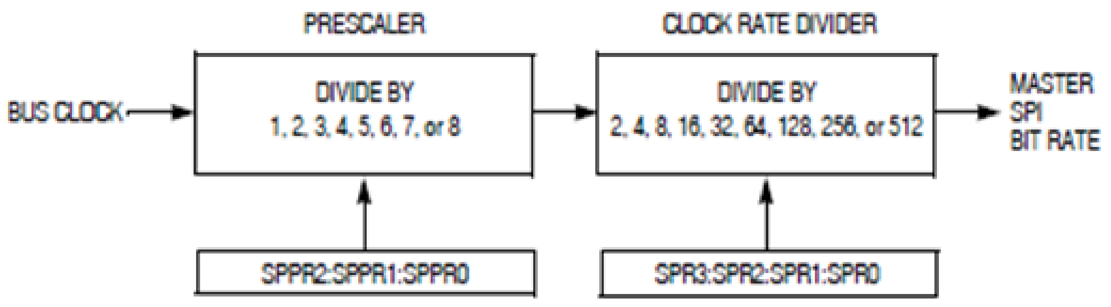


Figure 3. BaudRate Generation

For example—If bus clock is 20 MHz, the expected baudrate is 250 kHz.
(20 MHz / 250 KHz) = 80 = 5 * 16
Pre-scale value = 5
Clock rate divider = 16

2.4 Signal Description

- Each SPI shares four port pins (SS, SCLK, MOSI, MISO). The bit SPI System Enable (SPE) of SPIxC1 decides whether they behave as GPIOs or SPI pins. Refer to the reference manual titled *MC9S08GW64 MC9S08GW32 Reference Manual* Section 9.4.1 “ SPI Control Register 1 (SPIC1), ” (document number MC9S08GW64RM).
  - SPE = 0 —These four pins revert to being general-purpose port I/O pins
  - SPE = 1—The function of these pins depends on the settings of the SPI control bits

- The combination of Master Mode Fault Function Enable (MODFEN) and Slave Select Output Enable (SSOE) decides the functionality of the SS pin. Refer to the reference manual titled *MC9S08GW64 MC9S08GW32 Reference Manual* Section 9.4.1 “ SPI Control Register 1 (SPIC1), ” and Section 9.4.2 “ SPI Control Register 2 (SPIC2), ” (document number MC9S08GW64RM). The following table shows the various configurations of SS pins.

Table 1. SS pin configuration

MODFEN	SSOE	Master Mode	Slave Mode
0	0	General Purpose I/O (not SPI)	Slave select input
0	1	General Purpose I/O (not SPI)	Slave select input
1	0	$\overline{\text{SS}}$ input for mode fault	Slave select input
1	1	Automatic $\overline{\text{SS}}$ output	Slave select input

- The SPC0 (SPI Pin Control) bit chooses between single-wire bidirectional mode or double-wire full duplex mode. Refer to the reference manual titled *MC9S08GW64 MC9S08GW32 Reference Manual* Section 9.4.2 “ SPI Control Register 2 (SPIC2). ” The following figure shows various configurations.

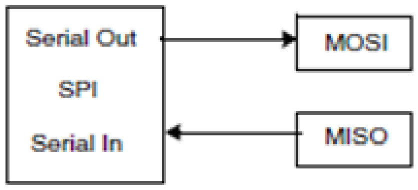
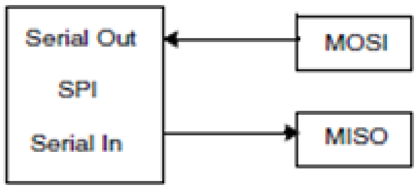
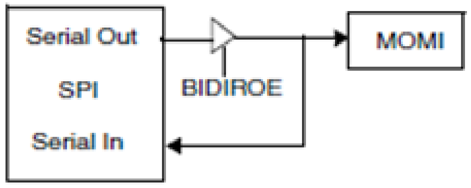
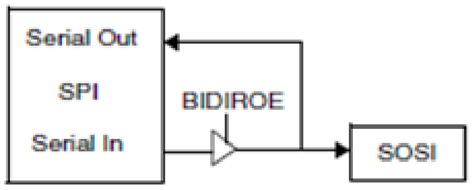
When SPE = 1	Master Mode MSTR = 1	Slave Mode MSTR = 0
Normal Mode SPC0 = 0		
Bidirectional Mode SPC0 = 1		

Figure 4. SPI pin control configurations

2.5 Mode Fault Detection

The SS pin is configured to be the mode fault input signal when the MSTR = 1 (Master mode), MODFEN = 1 (Mode fault detection enabled), SSOE = 0 (Slave Select Output disabled). This feature is used in a system where more than one SPI device may become a master at the same time. The error is detected when a master's SS pin is low, indicating that another SPI device is trying to address this master as if it were a slave.

3 Software Driver Description

The SPI driver is provided as C code files. You can add these files to your applications. With the integration of the SPI driver, you can call SPI driver APIs to use the SPI functionality in your application.

There are three files associated with the SPI driver. The following is a brief description:

spi.h—Contains all the high level API declarations and the various macros used in the functions. It defines the structure of the various SPI registers.

spi_config.h—This file contains the various defines to control the configuration of the SPI. The user can make the changes in this file to get the required configuration.

spi.c—It is the main file for the driver. It contains the various high level API definitions.

3.1 spi.h

NOTE

The macros provided are passed as arguments to the respective functions to get the required configuration. It has been explained in details in Section 3.3 spi.c.

Table 2. Defined macros

Macros	Description
# define SPI_Index0	Macros to choose between the three SPIs. The three SPIs can also be chosen simultaneously. The following three macros are required to choose between the three SPIs.
# define SPI_Index1	
# define SPI_Index2	
# define SPI_MASTER_MODE	Macros to select between master and slave mode. These two macros are for this.
# define SPI_SLAVE_MODE	
# define SPI_SS_PIN_GPIO	Macros to select the required functionality of the SS pin in master mode. In slave mode the SS pin acts as slave select input pin. These three macros are available for configuration in master mode.
# define SPI_SS_PIN_MODE_FAULT	
# define SPI_SS_PIN_SS_OUTPUT	
# define SPI_CLOCK_POL_0	Macros to configure the clock polarity. These two are used for this.
# define SPI_CLOCK_POL_1	
# define SPI_CLOCK_PHASE_0	Macros to configure the clock phase. These two are used for this.
# define SPI_CLOCK_PHASE_1	
# define SPI_INTERRUPT_ENABLE	Macros for enabling and disabling the interrupt for the SPI. These macros are used.
# define SPI_INTERRUPT_DISABLE	
# define SPI0_RX	Macros to indicate that the data has been received on the specific SPI port .The following three macros are used for the three SPIs.
# define SPI1_RX	
# define SPI2_RX	
# define SPI0_TX	There are macros to indicate that data has been transmitted from a specific SPI port. These three macros are used for three SPIs
# define SPI1_TX	
# define SPI2_TX	
unsigned char SPI0_Data	Global variables are also defined which are used globally and can be accessed by any function. There are three global variables used to store data transmitted in case of an interrupt. There are three different global variables for different SPI.
unsigned char SPI1_Data	
unsigned char SPI2_Data	

3.2 spi_config.h

This file contains various defines to control the configuration of the SPI. The user can make the changes in this file to get the required configuration.

# define BUS_CLK

This macro is used to define the value of the bus clock.

NOTE

It does not change the bus clock of the device. It sets the value of the macro used in the calculation of baudrate.

Example:

# define BUS_CLK 20000000 defines the bus clock to be 20 MHz. Therefore, the baudrate is calculated assuming the bus clock to be 20 MHz

There are macros used for pin muxing. The same pins of the SPI are muxed to two different ports. These macros help in selecting the required port.

NOTE

No macro is provided for SPI0 pin muxing because there is no pin muxing in SPI0.

Only port B is used with Pin B2 for MOSI0, pin B3 for MISO1, Pin B4 for SCLK0, and Pin B5 for SS0

The following macros are used for pin muxing in for SPI1

# define SPI1_Pins_PortC

It is used to configure pin C0 for MOSI1, C1 for MISO1, Pin C2 for SCLK1, and Pin C3 for SS1
Example:
# define SPI1_Pins_PortC 1

# define SPI1_Pins_PortG

It is used to configure pin G0 for MOSI1, G1 for MISO1, Pin G2 for SCLK1, and Pin G3 for SS1
Example:
# define SPI1_Pins_PortG 1

NOTE

Both SPI1_Pins_PortC and SPI1_Pins_PortG cannot be 1 at the same time. It gives an error.

The following macros are used for pin muxing in case of SPI2

# define SPI2_Pins_PortA

It is used to configure pin A0 for MOSI2, A1 for MISO2, Pin A2 for SCLK2, and Pin A3 for SS2
Example:
# define SPI2_Pins_PortA 1

# define SPI1_Pins_PortD

It is used to configure pin D0 for MOSI2, D1 for MISO2, Pin D2 for SCLK2, Pin D3 for SS2
Example:
# define SPI2_Pins_PortD 1

NOTE

Both SPI2_Pins_PortA and SPI2_Pins_PortD cannot be one at the same time. It gives an error.

3.3 spi.c

This file contains the definition of various functions.

3.3.1 SPI_Init

Description:

This function is used to initialize the specific SPI by configuring the internal registers. The user needs to enter the input arguments to set the master and slave mode, configure the slave select pin, set the clock phase, and polarity. The function sets the baudrate to a value of 250 KHz assuming the bus clock to be 20 MHz.

Prototype:

void SPI_Init(unsigned char SPI_Index, unsigned char Master_Slave_Mode,
unsigned char Configure_Slave_Select_Pin,
unsigned char Clock_Polarity, unsigned char Clock_Phase, void (*p)(unsigned char1, unsigned char2))

Input parameters:

1. SPI_Index—To select the SPI to be initiated SPI_Index0, SPI_Index1, or SPI_Index2
2. Master_Slave_Mode—To select between slave and master mode SPI_SLAVE_MODE and SPI_MASTER_MODE
3. Configure_Slave_Select_Pin—To configure the functionality of a slave select pin SPI_SS_PIN_GPIO, SPI_SS_PIN_SS_OUTPUT, and SPI_SS_PIN_MODE_FAULT
4. Clock_Polarity—To select the clock polarity of the specific SPI SPI_CLOCK_POL_0 and SPI_CLOCK_POL_1
5. Clock_Phase—To select the phase of the clock of the specific SPI SPI_CLOCK_PHASE_0 and SPI_CLOCK_PHASE_1
6. p—Function is used in case of interrupts only. The user can pass the address of the callback function or pass null.
7. char1—It is passed as an argument to the callback function. It specifies whether the interrupt is transmission or reception.
8. char2—It is passed as an argument to the callback function. It specifies whether the interrupt is transmission or reception.

Output parameters:

None

Example:

```
void func (unsigned char,unsigned char)
{}
```

SPI_Init (SPI_Index1, SPI_MASTER_MODE, SPI_SS_PIN_GPIO, SPI_CLOCK_POL_0, SPI_CLOCK_PHASE_1, &func);

Initializes SPI1 and configures it in master mode with the SS pin configured as a GPIO. The clock polarity is 0 and the clock phase is 1. It passes the address of a function for the callback.

3.3.2 SPI_Set_BaudRate

Description:

This function is used to set the required baudrate of the specific SPI. The user is required to specify the SPI and the baudrate required (in KHz). The function calculates the values of pre-scale and rate Divisor and sets the baudrate.

Prototype:

unsigned char SPI_Set_BaudRate(unsigned char SPI_Index, unsigned int Baud_Rate)

Input parameters:

1. SPI_Index—To select the SPI to be initiated SPI_Index0/SPI_Index1/SPI_Index2
2. Baud_Rate—The baudrate to be set. It must be entered in KHz.

Output parameters:

None

Example:

If a baudrate of 250 KHz is to be set for SPI0
SPI_Set_Baud_Rate(SPI_Index0,250);

3.3.3 SPI_Write_Char

Description:

Function sends one byte of data to the specific SPIx port.

Prototype:

void SPI_Write_Char(unsigned char SPI_Index, unsigned char Data, unsigned char Interrupt_Enable)

Input parameters:

1. SPI_Index—To select the SPI to be initiated SPI_Index0/SPI_Index1/SPI_Index2
2. Data—The one byte data to be sent
3. Interrupt_Enable—To enable or disable the interrupt SPI_INTERRUPT_ENABLE/SPI_INTERRUPT_DISABLE

Output parameters:

None

Example:

SPI_Write_Char (SPI_Index1, 0xAA, SPI_INTERRUPT_DISABLE);
Sends a char 0xAA to SPI1 with the interrupt disabled

3.3.4 SPI_Read_Char

Description:

This function reads one byte of data from the SPIx port.

Prototype:

unsigned char SPI_Read_Char(unsigned char SPI_Index, unsigned char Interrupt_Enable)

Input parameters:

1. SPI_Index—To select the SPI to be initiated SPI_Index0/SPI_Index1/SPI_Index2
2. Interrupt_Enable—To enable or disable the interrupt SPI_INTERRUPT_ENABLE/SPI_INTERRUPT_DISABLE

Output parameters:

None

Example:

unsigned char data_read;
data_read = SPI_Read_Char (SPI_Index1, SPI_INTERRUPT_DISABLE);
Reads a byte from SPI1 and stores it in data_read with the interrupt disabled.

3.3.5 Send_Data

Description:

This function is used internally within other functions to send data to the SPIx port.

Prototype:

void Send_Data(unsigned char SPI_Index, unsigned char Data, unsigned char Interrupt_Enable)

Input parameters:

1. SPI_Index—To select the SPI to be initiated
   SPI_Index0/SPI_Index1/SPI_Index2
2. Data—The one byte data to be sent
3. Interrupt_Enable—To enable or disable the interrupt
   SPI_INTERRUPT_ENABLE/SPI_INTERRUPT_DISABLE

Output parameters:
None

3.3.6 Wait_For_Data

Description:
This function is used internally within other functions to read and return the data (1 byte) read from the selected SPI port.

Prototype:
char Wait_For_Data(unsigned char SPI_Index, unsigned char Interrupt_Enable)

Input parameters:

1. SPI_Index—To select the SPI to be initiated SPI_Index0/SPI_Index1/SPI_Index2
2. Interrupt_Enable—To enable or disable the interrupt SPI_INTERRUPT_ENABLE/SPI_INTERRUPT_DISABLE

Output parameters:
Returns the data byte read from the selected SPI port.

3.3.7 Interrupt Subroutines

There are three types of interrupts:

- SPI Receive buffer full
- Mode Fault Error
- SPI Transmit Buffer Empty

SPI0_ISR

Description:
It is an interrupt subroutine for SPI0. The function is executed when any of the above interrupts occur, this is provided the respective interrupts are enabled. The routine identifies what interrupt occurs, clears the respective interrupts, and takes the required action.

Prototype:
void interrupt VectorNumber_Vspi0 SPI0_ISR()

Input parameters:
None

Output parameters:
None

SPI1_ISR

Description:

It is an interrupt subroutine for SPI1. The function is executed when any of the above interrupts occur, this is provided the respective interrupts are enabled. The routine identifies which interrupt occurs, clears the respective interrupts, and takes the required action.

Prototype:
void interrupt VectorNumber_Vspi1 SPI1_ISR()

Input parameters:
None

Output parameters:
None

SPI2_ISR

Description:
It is an interrupt subroutine for SPI2. The function is executed when any of the above interrupts occur, provided the respective interrupts are enabled. The routine identifies which interrupt occurs, clears the respective interrupts, and takes the required action.

Prototype:
void interrupt VectorNumber_Vspi2 SPI2_ISR()

Input parameters:
None

Output parameters:
None

4 Assumptions

The descriptions in this document assumes the person reading it has full knowledge of all the configuration registers of all the blocks in MC9S08GW64, especially LCD and Internal Clock Source (ICS) blocks.

5 Use Case

Assuming that the clock settings are done and the bus clock is running on 20 Mhz. Include spi.h in the main file.

To initialize the respective SPI with the desired configuration and establish communication:

Step 1—If you want to choose port C for SPI make this setting in spi_config.h
# define SPI1_Pins_PortC 1

Step 2—Suppose there is a callback function defined as

```
void callback(unsigned char a, unsigned char b)
{
  return;
}
```

SPI_Init (SPI_Index1, SPI_MASTER_MODE, SPI_SS_PIN_GPIO, SPI_CLOCK_POL_0, SPI_CLOCK_PHASE_0,& callback);

This initialized the SPI1 in master mode, configures the SS pin as a GPIO, clock polarity as 0, clock phase as 0, and the address of callback function is passed which is used in case of interrupts. The default baudrate is set as 250 KHz.

Step 3—To send data, the following function is called:

SPI_Write_Char (SPI_Index1, 0xAA, SPI_INTERRUPT_DISABLE);

Sends a char 0xAA to SPI1 with the interrupt disabled.

6 Conclusion

This driver provides a software base for applications that need the implementation of a SPI.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN4162
Rev. 0, 08/2010