AN205504

# F²MC-8FX Family MB95200H Series SPI SW Realization using GPIO

This application note describes about how to use the GPIO to realize SPI function on the MB95F200 series.

## Contents



# 1 Introduction

In this document, we will introduce how to use the GPIO to realize SPI function on the MB95F200 series.

# 2 SPI Overview

## 2.1 Background

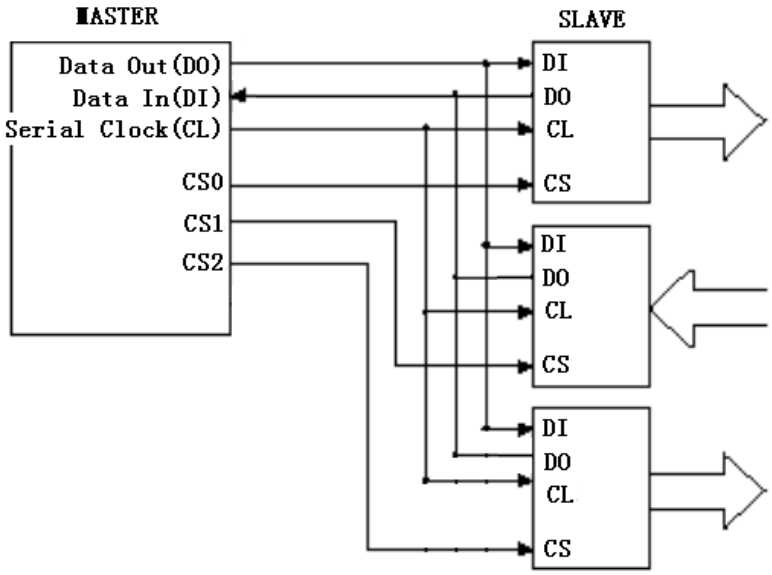
The SPI is Motorola's serial peripheral interface. It is a flexible synchronous serial interface superset standard. It allows duplex, synchronous, serial communication between the MCU and peripheral devices.

## 2.2 SPI Formats

SPI is a single master single slave synchronous serial communication. Each data bit from and to the master has its own clock pulse. Therefore SPI uses at least three different signals. Some SPI formats use a fourth signal called ‘Chip Select’ (CS) which enables the communication by the master. This signal can be positive or negative active. These signals are:

1. CL Serial Clock
2. DI Serial Input (Data from Slave to Master)
3. DO Serial Output (Data from Master to Slave)
4. CS Chip Select

Figure 1. SPI Application




## 2.3 SPI Clock (CL)

Because there is no common SPI specification, the timing of the SPI clock signal is device dependent and it seems that every producer uses its own timing.

For most SPI protocols four different settings are possible, which are mostly defined by the internal SPI master control settings: CPOL (Clock polarity) and CPHA (Clock phase).

CPOL defines the active state of the SPI serial clock and thus the mark level.

CPHA defines the clock phase in respect of the SO-data bit.

There is no common classification of SPI protocols and SPI devices, which defines the settings of CPOL and CPHA in respect of the protocol itself. So this document defines the settings as follows:

CPOL = 0: SPI clock has mark level “1”

CPOL = 1: SPI clock has mark level “0”

CPHA = 0: SPI clock is synchronous to SO-data bits

CPHA = 1: SPI clock is delayed by a half bit time in respect of SO-data bit

## 2.4 Data Order

In the most SPI formats the serial data direction is MSB first.

## 2.5 Communication Baudrate

The transfer rate of SPI communication is defined by the hardware itself. Mostly the speed is limited by set-up and hold timing of the data signals.

SPI is used in a wide communication speed range. The range reaches from KBits/s to MBits/s.

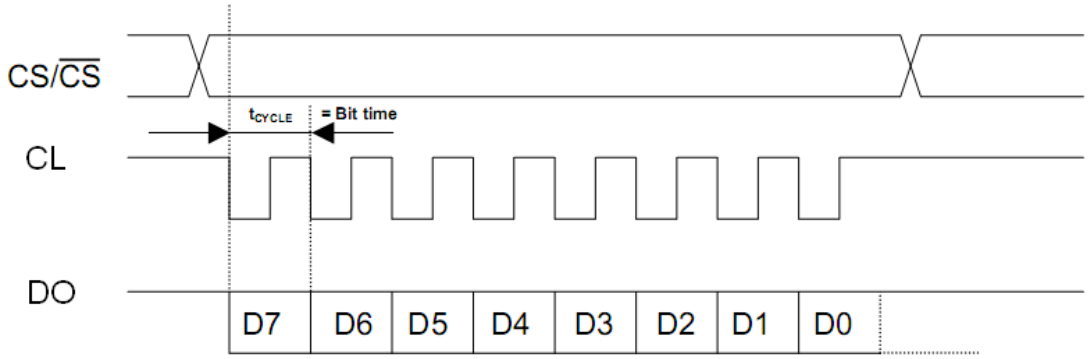
## 2.6 SPI Protocols

This section describes the most common different SPI protocol timings.

### 2.6.1 CPOL = 0, CPHA = 0

This SPI format is in accordance with the standard serial synchronous format. The timing of the signals is shown in the figure below:

Figure 2. SPI Protocol (CPOL = 0, CPHA = 0

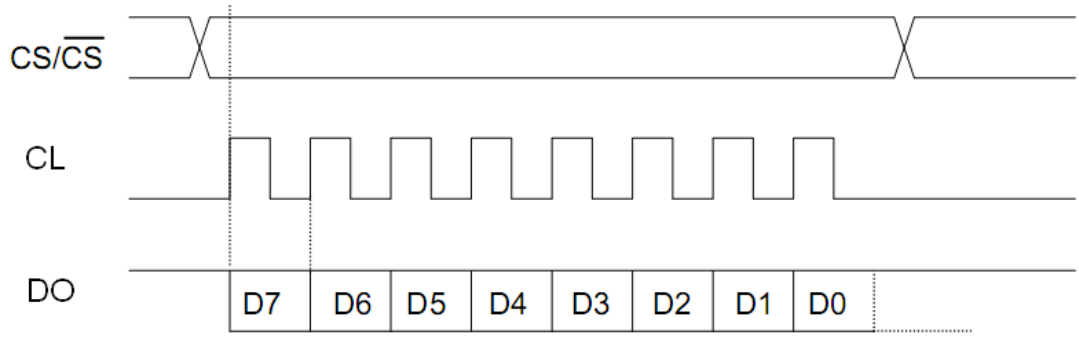



**Note:** SPI clock in phase with data (mark level = “1”).

### 2.6.2 CPOL = 1, CPHA = 0

This SPI format is in accordance with the standard serial synchronous format with inverted clock signal.

Figure 3. SPI Protocol (CPOL = 1, CPHA = 0)

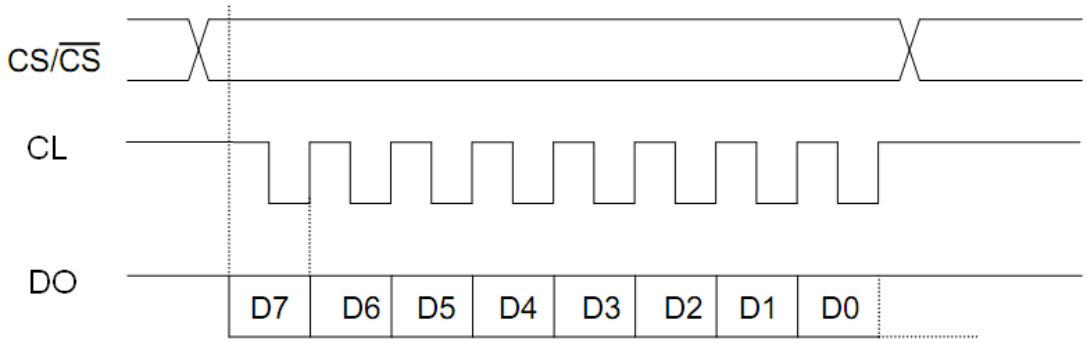



**Note:** SPI clock in phase with data (mark level = “0”, inverted).

### 2.6.3 CPOL = 0, CPHA = 1

In this SPI format the clock signal is delayed by a half bit time to the standard synchronous format.

Figure 4. SPI Protocol (CPOL = 0, CPHA = 1)

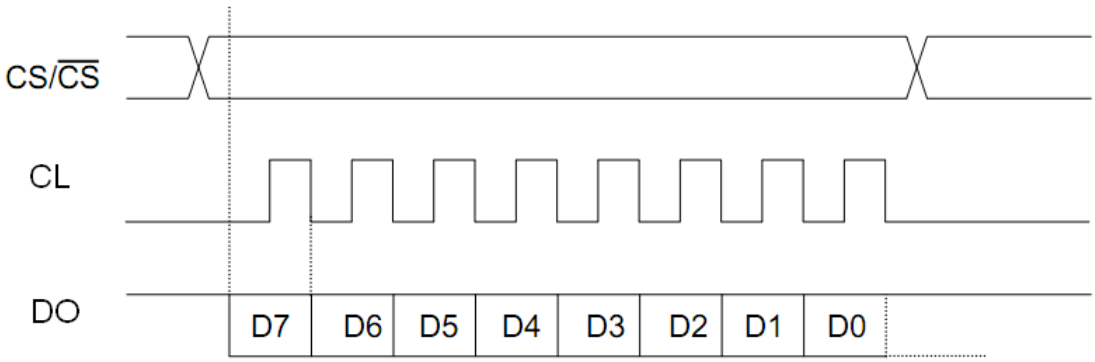



**Note:** SPI clock delayed by half bit time (mark level = “1”). In this format, the first data bit has to be set on DO before the first clock pulse occurs.

### 2.6.4 CPOL = 1, CPHA = 1

In this SPI format the clock signal is delayed by a half bit time to the standard synchronous protocol and the clock signal is inverted.

Figure 5. SPI Protocol (CPOL = 1, CPHA = 0)




**Note:** SPI clock delayed by half bit time (mark level = “0”, inverted). In this format, the first data bit has to be set on SO before the first clock pulse occurs.

# 3 SPI Driver

## 3.1 Peripheral Usage

The MCU pins used as below:

P00, used as DO;

P01, used as DI;

P02, used as CL;

P03, used as CE;

## 3.2 Driver Code

In this driver, the mode we used is “CPOL = 0, CPHA = 0”.

### 3.2.1 General Definition

```
typedef unsigned char     BOOLEAN;
typedef unsigned char     INT8U;          /* Unsigned   8 bit quantity */
typedef signed    char    INT8S;          /* Signed     8 bit quantity */
typedef unsigned  int     INT16U;         /* Unsigned   16 bit quantity */
typedef signed    int     INT16S;         /* Signed     16 bit quantity */
typedef unsigned long     INT32U;         /* Unsigned    32 bit quantity */
typedef signed    long    INT32S;         /* Signed      32 bit quantity */

#define BOOL              BOOLEAN
#define BYTE              INT8U
#define UBYTE             INT8U
#define WORD              INT16U
#define UWORD             INT16U
#define LONG              INT32S
#define ULONG             INT32U
#define UCHAR             INT8U
#define UINT              INT16U
#define DWORD             INT32U

#define TRUE              1
#define FALSE             0

#define CE                PDR0_P03
#define CL                PDR0_P02
#define DI                PDR0_P01
#define DO                PDR0_P00
```

### 3.2.2 SPI Routine

void DelayUS(UBYTE nDly)

Return : None.

Parameters : nDly,delay US.

Description : Delay nDly US on the SPI.

Example : DelayUS (3);

```
void DelayUS(UBYTE nDly)
{
  for(;nDly>0;nDly--){}
}
```

**Note:** Different MCUs have different clock loops, please change the 'DelayUS()' function to realize the us delay.

void SPI_Init ()

Return : None.

Parameters : None.

Description : Initialize the SPI.

Example : SPI_Init ();

```
void SPI_Init()
{
    DDR0_P03=1;//P03,P02,P01 is output,P00 is input
    DDR0_P02=1;
    DDR0_P01=1;
    DDR0_P00=0;
}
```

UBYTE SPIReadByte ()

Return : Data from the SPI.

Parameters : None.

Description : Read a data from the SPI.

Example : Tmp=SPIReadByte ();

```
UBYTE SPIReadByte()
{
  UBYTE i=0;
  UBYTE val=0;

  DelayUS(5);
  for(i=0;i<8;i++)
  {
    CL=LO;
    DelayUS(1);
    CL=HI;
    val=val<<1;
    if(DO) val|=0x01;
    DelayUS(1);
  }
  DelayUS(5);

  return val;
}
```

void SPIWriteByte(UBYTE nVal)

Return : None.

Parameters : nVal,data writen to the SPI.

Description : Write a data to the SPI.

Example : SPIWriteByte (0x65);

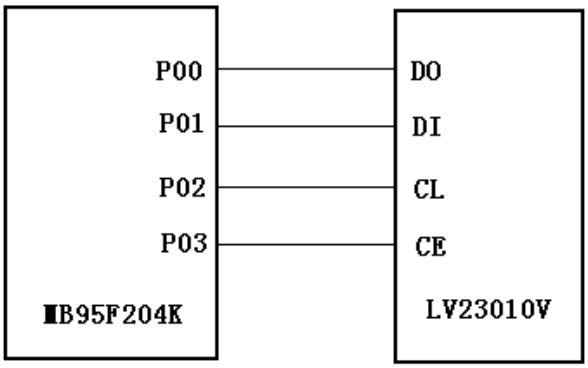
```
void SPIWriteByte(UBYTE nVal)
{
  UBYTE i=0;

  DelayUS(5);
  for(i=0;i<8;i++)
  {
    CL=LO;
    DI=nVal>>i;
    DelayUS(1);
    CL=HI;
    DelayUS(1);
  }
  DelayUS(5);
}
```

# 4 Typical Application

## 4.1 HW Design

In this application, the MCU used is MB95F204K and the SPI device is LV23010V. The HW is designed as Figure 2.

Figure 6. Hardware Design




## 4.2 Sample Code

The codes are applied on LV23010V. Please refer the LV23010V Manual to get the detailed information. In the demo, the read address is set to 0x2A and the write address is set to 0x28, 0x29.

void SPI_Read(UBYTE nAddr,UBYTE nBytes,UBYTE *pBuf)

Return : NONE.

Parameters : nAddr,the LV23010 read address,0x2A;

nBytes, read number;

pBuf,data buffer read from the LV23010V.

Description : Read a data from the LV23010V.

Example : BYTE buf[3];

SPI_Read (0x2A,3, buf);

```
void SPI_Read(UBYTE nAddr,UBYTE nBytes,UBYTE *pBuf)
{
  UBYTE j;

  CE=LO;
  SPIWriteByte(nAddr);
  CE=HI;
  for(j=0;j<nBytes;j++)
    pBuf[j]=SPIReadByte();
  CE=LO;
}
```

void SPI_Write(UBYTE nAddr,UBYTE nBytes,UBYTE *pBuf)

Return : NONE.

Parameters : nAddr,the LV23010 read address,0x28 or 0x29;
nBytes,write number;
pBuf,data buffer writen to the LV23010V.

Description : Write a data to the LV23010V.

Example : BYTE buf[3];
SPI_Write (0x29,3, buf);

```
void SPI_Write(UBYTE nAddr,UBYTE nBytes,UBYTE *pBuf)
{
  UBYTE j;

  CE=LO;
  SPIWriteByte(nAddr);
  CE=HI;
  for(j=0;j<nBytes;j++)
    SPIWriteByte(pBuf[j]);
  CE=LO;
}
```

## 4.3 SPI Communication Wave

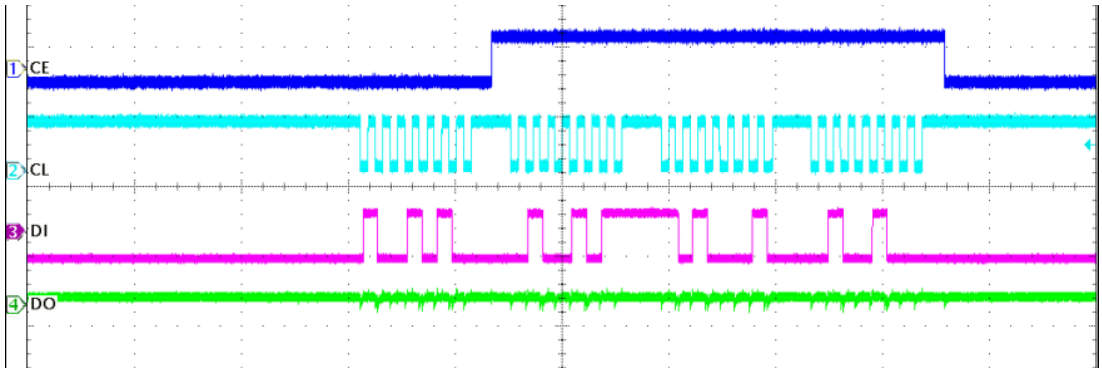
The baudrate is 500kbit/s.

### 4.3.1 Write Wave

Addr: 0x29

Data: 0xD2, 0x45, 0x12

Figure 7. Write Wave



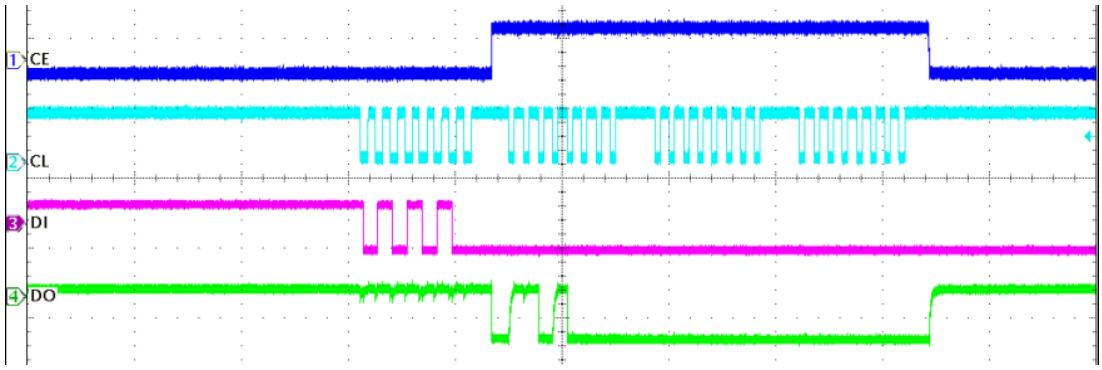

**Note:** Address is LSB first. Data is LSB first.

### 4.3.2 Read Wave

Addr: 0x2A

Data: 0xD0, 0x00, 0x00

Figure 8. Read Wave




**Note:** Address is LSB first. Data is MSB first.

# 5 Notes

SPI reading and writing address is always different. In other words, we only read or write on a address.

# 6 More Information

For more information on Cypress MB95200 products, please visit following website:

www.cypress.com/documentation/application-notes/mb95200-spi-sw-realization-using-gpio

## Document History

Document Title: AN205504 - F²MC-8FX Family MB95200H Series SPI SW Realization using GPIO

Document Number: 002-05504

| Revision | ECN | Orig. of Change | Submission Date | Description of Change |
|---|---|---|---|---|
| ** | — | HUAL | 02/04/2009 | Initial release. |
| | | | 03/24/2009 | Modify the format. |
| *A | 5261876 | HUAL | 05/09/2016 | Migrated Spansion Application note from MCU-AN-500035-E-10 to Cypress format. |

## 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 |
| Lighting & Power Control | cypress.com/powerpsoc |
| Memory | cypress.com/memory |
| PSoC | cypress.com/psoc |
| Touch Sensing | cypress.com/touch |
| USB Controllers | cypress.com/usb |
| Wireless/RF | cypress.com/wireless |

## PSoC® Solutions

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

## Cypress Developer Community

Forums | Projects | Videos | Blogs | Training | Components

## Technical Support

cypress.com/support



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

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com