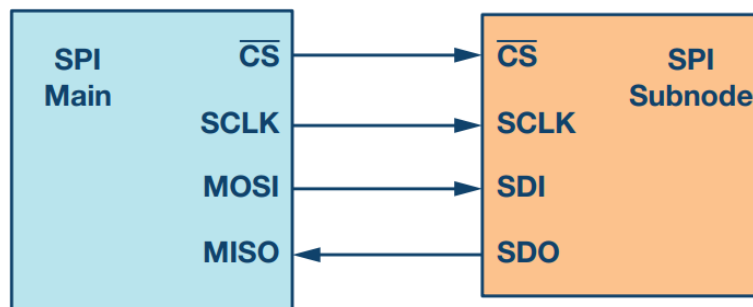


Chapter 7: Serial Peripheral Interface (SPI)

7.1 Introduction:

Serial peripheral interface (SPI) is one of the most widely used interfaces between microcontroller and peripheral ICs such as sensors, ADCs, DACs, shift registers, SRAM, and others. This article provides a brief description of the SPI interface followed by an introduction to Analog Devices' SPI enabled switches and muxes, and how they help reduce the number of digital GPIOs in system board design. SPI is a synchronous, full duplex master-slave-based interface. The data from the master or the slave is synchronized on the rising or falling clock edge. Both master and slave can transmit data at the same time. The SPI interface can be either 3-wire or 4-wire. This article focuses on the popular 4-wire SPI interface.

7.1.1 Interface



4-wire SPI devices have four signals: X Clock (SPI CLK, SCLK) X Chip select (\overline{CS}) X Master out, slave in (MOSI) X Master in, slave out (MISO) The device that generates the clock signal is called the master. Data transmitted between the master and the slave is synchronized to the clock generated by the master. SPI devices support much higher clock frequencies compared to I2C interfaces. Users should consult the product data sheet for the clock frequency specification of the SPI interface. SPI interfaces can have only one master and can have one or multiple slaves. Figure 1 shows the SPI connection between the master and the slave. The chip select signal from the master is used to select the slave. This is normally an active low signal and is pulled high to disconnect the slave from the SPI bus. When multiple slaves are used, an individual chip select signal for each slave is required from the master. In this article, the chip select signal is always an active low signal. MOSI and MISO are the data lines.

MOSI transmits data from the master to the slave and MISO transmits data from the slave to the master.

7.1.2 Data Transmission

To begin SPI communication, the master must send the clock signal and select the slave by enabling the CS signal. Usually chip select is an active low signal; hence, the master must send a logic 0 on this signal to select the slave. SPI is a full-duplex interface; both master and slave can send data at the same time via the MOSI and MISO lines respectively. During SPI communication, the data is simultaneously transmitted (shifted out serially onto the MOSI/SDO bus) and received (the data on the bus (MISO/ SDI) is sampled or read in). The serial clock edge synchronizes the shifting and sampling of the data. The SPI interface provides the user with flexibility to select the rising or falling edge of the clock to sample and/or shift the data. Please refer to the device data sheet to determine the number of data bits transmitted using the SPI interface.

7.1.3 Clock Polarity and Clock Phase

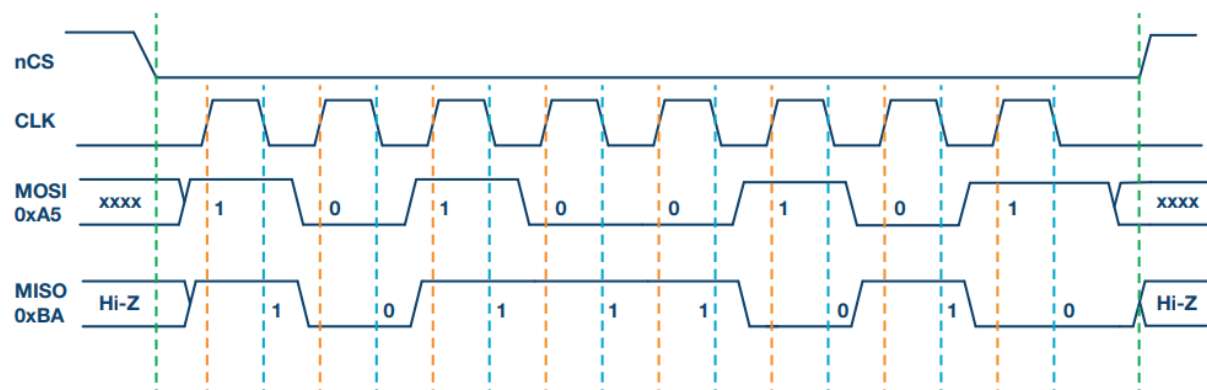
In SPI, the master can select the clock polarity and clock phase. The CPOL bit sets the polarity of the clock signal during the idle state. The idle state is defined as the period when CS is high and transitioning to low at the start of the transmission and when CS is low and transitioning to high at the end of the transmission. The CPHA bit selects the clock phase. Depending on the CPHA bit, the rising or falling clock edge is used to sample and/or shift the data. The master must select the clock polarity and clock phase, as per the requirement of the slave. Depending on the CPOL and CPHA bit selection, four SPI modes are available. Table 1 shows the four SPI modes.

SPI Mode	CPOL	CPHA	Clock Polarity in Idle State	Idle State Clock Phase Used to Sample and/or Shift the Data
0	0	0	Logic low	Data sampled on rising edge and shifted out on the falling edge

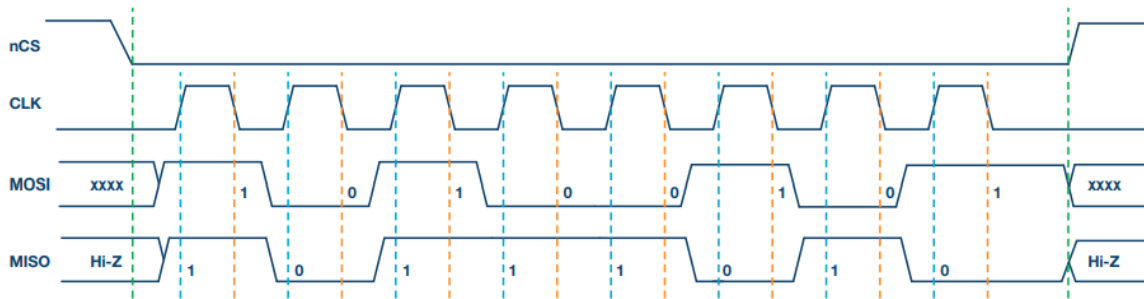
1	0	1	Logic low	Data sampled on the falling edge and shifted out on the rising edge
2	1	0	Logic high	Data sampled on the falling edge and shifted out on the rising edge
3	1	1	Logic high	Data sampled on the rising edge and shifted out on the falling edge

Figure 2 through Figure 5 show an example of communication in four SPI modes. In these examples, the data is shown on the MOSI and MISO line. The start and end of transmission is indicated by the dotted green line, the sampling edge is indicated in orange, and the shifting edge is indicated in blue. Please note these figures are for illustration purpose only. For successful SPI communications, users must refer to the product data sheet and ensure that the timing specifications for the part are met.

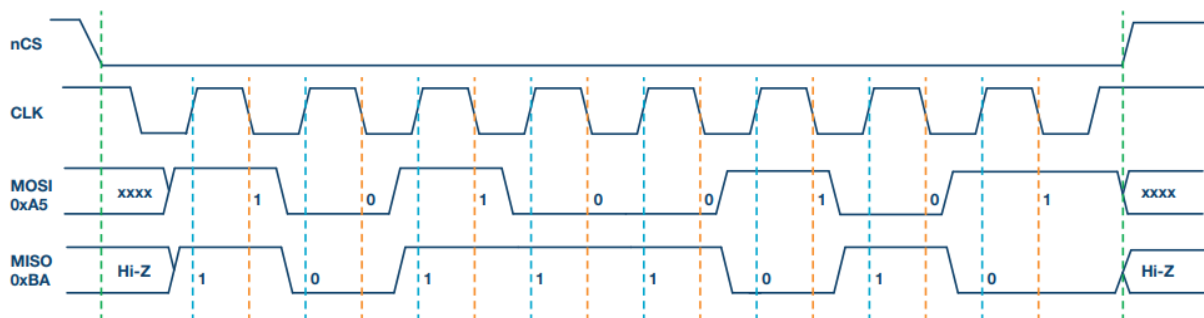
SPI Mode 0, CPOL = 0, CPHA = 0: CLK idle state = low, data sampled on rising edge and shifted on falling edge.



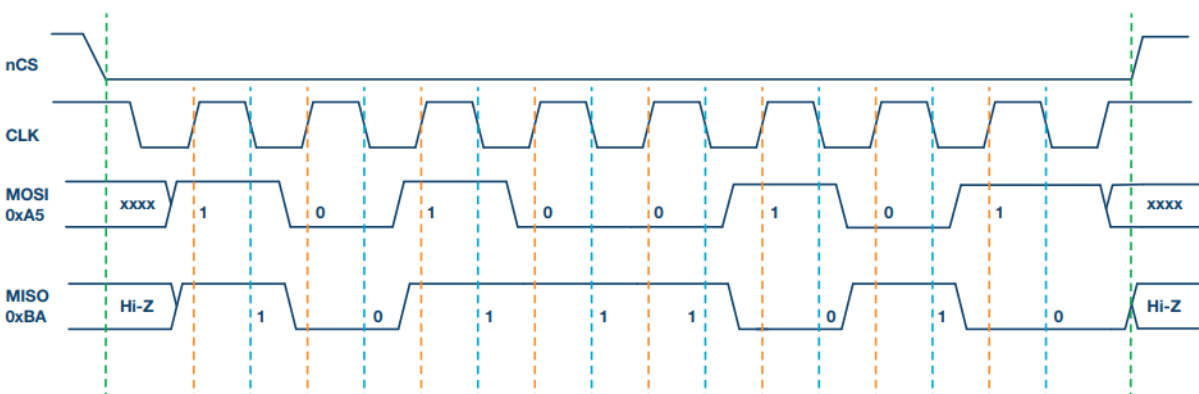
SPI Mode 1, CPOL = 0, CPHA = 1: CLK idle state = low, data sampled on the falling edge and shifted on the rising edge.



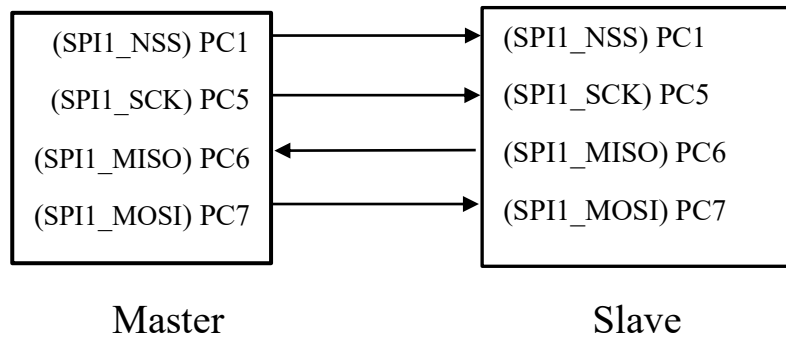
SPI Mode 2, CPOL = 1, CPHA = 0: CLK idle state = high, data sampled on the falling edge and shifted on the rising edge



SPI Mode 3, CPOL = 1, CPHA = 1: CLK idle state = high, data sampled on the rising edge and shifted on the falling edge.



7.2 Hardware Connections:



7.3 SPI Example Code:

SPI Master Code:

```
/****** (C) COPYRIGHT *****
*****
* File Name      : master.c
* Author         : WCH
* Version        : V1.0.0
* Date           : 2022/08/08
* Description    : Master program body.
*****
****
* Copyright (c) 2021 Nanjing Qinheng Microelectronics Co., Ltd.
* Attention: This software (modified or not) and binary are used for
* microcontroller manufactured by Nanjing Qinheng Microelectronics.
*****
**/

/*
 * @Note
 * Two-wire full duplex mode, master/slave mode, data transceiver:
 * Master: SPI1_NSS(PC1)、SPI1_SCK(PC5)、SPI1_MISO(PC7)、SPI1_MOSI(PC6).
 * Slave: SPI1_NSS(PC1)、SPI1_SCK(PC5)、SPI1_MISO(PC7)、SPI1_MOSI(PC6).
 *
 * This example demonstrates simultaneous full-duplex transmission and reception
 * between Master and Slave.
 * Note: The two boards download the Master and Slave programs respectively, and
 * power on at the same time.
 * Hardware connection:
 *      PC1 -- PC1
 *      PC5 -- PC5
 *      PC6 -- PC6
 *      PC7 -- PC7
 *
 */
```

```

#include "debug.h"
#include "string.h"

/* SPI Mode Definition */
#define HOST_MODE      0
#define SLAVE_MODE     1

/* SPI Communication Mode Selection */
#define SPI_MODE        HOST_MODE
// #define SPI_MODE      SLAVE_MODE

/* Global define*/
#define Size            18

/* Global Variable*/

ul6 TxData[Size];
ul6 RxData[Size];

/*****
 * @fn          SPI_FullDuplex_Init
 *
 * @brief       Configuring the SPI for full-duplex communication.
 *
 * @return      none
 */
void SPI_FullDuplex_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure = {0};
    SPI_InitTypeDef SPI_InitStructure = {0};

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC | RCC_APB2Periph_SPI1,
ENABLE);

    #if(SPI_MODE == HOST_MODE)
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_Init(GPIOC, &GPIO_InitStructure);

        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_Init(GPIOC, &GPIO_InitStructure);

        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
        GPIO_Init(GPIOC, &GPIO_InitStructure);

        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_Init(GPIOC, &GPIO_InitStructure);
    #endif
}

```

```

#elif(SPI_MODE == SLAVE_MODE)
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

#endif

#if(SPI_MODE == HOST_MODE)
    SPI_SSOutputCmd(SPI1, ENABLE);

#endif

    SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;

#if(SPI_MODE == HOST_MODE)
    SPI_InitStructure.SPI_Mode = SPI_Mode_Master;

#elif(SPI_MODE == SLAVE_MODE)
    SPI_InitStructure.SPI_Mode = SPI_Mode_Slave;

#endif

    SPI_InitStructure.SPI_DataSize = SPI_DataSize_16b;
    SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
    SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge;
    SPI_InitStructure.SPI_NSS = SPI_NSS_Hard;
    SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_64;
    SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
    SPI_InitStructure.SPI_CRCPolynomial = 7;
    SPI_Init(SPI1, &SPI_InitStructure);

    SPI_Cmd(SPI1, ENABLE);
}

/*****
 * @fn      master
 *
 * @brief   Master program.
 *
 * @return  none
 */
int main(void)
{

```

```

u8 value;

SystemCoreClockUpdate();
Delay_Init();
USART_Printf_Init(115200);
printf("SystemClk:%d\r\n", SystemCoreClock);
printf("ChipID:%08x\r\n", DBGMCU_GetCHIPID());

char strToSend[] = "Hello, World!"; // Replace with your desired string

// Calculate string length including null terminator
int stringLength = strlen(strToSend) + 1;

// Copy string characters (including null terminator) to TxData array
for (int i = 0; i < stringLength; i++) {
    TxData[i] = strToSend[i]; // Convert char to u16 for transmission
}

#if(SPI_MODE == SLAVE_MODE)
    printf("Slave Mode\r\n");
    Delay_Ms(1000);
#endif

    SPI_FullDuplex_Init();

#if(SPI_MODE == HOST_MODE)
    printf("Host Mode\r\n");
    Delay_Ms(2000);
#endif

while (1) {
    int i = 0;
    int j = 0;

    while ((i < stringLength) || (j < stringLength)) {
        if (i < stringLength) {
            if (SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) != RESET) {
                SPI_I2S_SendData(SPI1, TxData[i]);
                i++;
            }
        }

        if (j < stringLength) {
            if (SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE) != RESET) {
                RxData[j] = SPI_I2S_ReceiveData(SPI1);
                j++;
            }
        }
    }

    value = memcmp(TxData, RxData, Size);

    if(value == 0)
    {

```



```

printf("Same\r\n");
printf("Communication Successful");

printf("Received String: ");
for (int i = 0; i < stringLength; i++) {
    // Print each 16-bit value as its corresponding character
    printf("%c", (char) (RxData[i] & 0xFF)); // Mask to get lower 8 bits
for character
}
printf("\r\n");
}
else
{
    printf("Different\r\n");
    printf("Communication Failed");

}

while(1);
}
}

```

SPI Slave Code:

```

/***** (C) COPYRIGHT
*****
* File Name      : master.c
* Author         : WCH
* Version        : V1.0.0
* Date           : 2022/08/08
* Description    : Master program body.

*****
****
* Copyright (c) 2021 Nanjing Qinheng Microelectronics Co., Ltd.
* Attention: This software (modified or not) and binary are used for
* microcontroller manufactured by Nanjing Qinheng Microelectronics.

*****
**/

/*
* @Note
* Two-wire full duplex mode, master/slave mode, data transceiver:
* Master: SPI1_NSS(PC1)、SPI1_SCK(
) 、SPI1_MISO(PC7)、SPI1_MOSI(PC6) .
* Slave: SPI1_NSS(PC1)、SPI1_SCK(PC5)、SPI1_MISO(PC7)、SPI1_MOSI(PC6) .
*
* This example demonstrates simultaneous full-duplex transmission and reception
* between Master and Slave.
* Note: The two boards download the Master and Slave programs respectively, and
* power on at the same time.
* Hardware connection:
* PC1 -- PC1
* PC5 -- PC5

```

```

*                               PC6 -- PC6
*                               PC7 -- PC7
*
*
*/

#include "debug.h"
#include "string.h"

/* SPI Mode Definition */
#define HOST_MODE      0
#define SLAVE_MODE     1

/* SPI Communication Mode Selection */
// #define SPI_MODE      HOST_MODE
#define SPI_MODE      SLAVE_MODE

/* Global define */
#define Size           18

/* Global Variable */

u16 TxData[Size] = {0x0101, 0x0202, 0x0303, 0x0404, 0x0505, 0x0606,
                    0x1111, 0x1212, 0x1313, 0x1414, 0x1515, 0x1616,
                    0x2121, 0x2222, 0x2323, 0x2424, 0x2525, 0x2626};
u16 RxData[Size];

/*****
* @fn      SPI_FullDuplex_Init
*
* @brief   Configuring the SPI for full-duplex communication.
*
* @return  none
*/
void SPI_FullDuplex_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure = {0};
    SPI_InitTypeDef  SPI_InitStructure = {0};

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC | RCC_APB2Periph_SPI1,
ENABLE);

    #if(SPI_MODE == HOST_MODE)
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_Init(GPIOC, &GPIO_InitStructure);

        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_Init(GPIOC, &GPIO_InitStructure);

        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
        GPIO_Init(GPIOC, &GPIO_InitStructure);
    #endif
}

```

```

        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_Init(GPIOC, &GPIO_InitStructure);

#elif(SPI_MODE == SLAVE_MODE)
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
        GPIO_Init(GPIOC, &GPIO_InitStructure);

        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
        GPIO_Init(GPIOC, &GPIO_InitStructure);

        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_Init(GPIOC, &GPIO_InitStructure);

        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
        GPIO_Init(GPIOC, &GPIO_InitStructure);

#endif

#if(SPI_MODE == HOST_MODE)
    SPI_SSOutputCmd(SPI1, ENABLE);

#endif

    SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;

#if(SPI_MODE == HOST_MODE)
    SPI_InitStructure.SPI_Mode = SPI_Mode_Master;

#elif(SPI_MODE == SLAVE_MODE)
    SPI_InitStructure.SPI_Mode = SPI_Mode_Slave;

#endif

    SPI_InitStructure.SPI_DataSize = SPI_DataSize_16b;
    SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
    SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge;
    SPI_InitStructure.SPI_NSS = SPI_NSS_Hard;
    SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_64;
    SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
    SPI_InitStructure.SPI_CRCPolynomial = 7;
    SPI_Init(SPI1, &SPI_InitStructure);

    SPI_Cmd(SPI1, ENABLE);
}

/*****
* @fn      master
*
* @brief   Master program.
*****/

```

```

*
* @return none
*/
int main(void)
{
    u8 value;

    SystemCoreClockUpdate();
    Delay_Init();
    USART_Printf_Init(115200);
    printf("SystemClk:%d\r\n", SystemCoreClock);
    printf("ChipID:%08x\r\n", DBGMCU_GetCHIPID() );

    char strToSend[] = "Hello, World!"; // Replace with your desired string

    // Calculate string length including null terminator
    int stringLength = strlen(strToSend) + 1;

    // Copy string characters (including null terminator) to TxData array
    for (int i = 0; i < stringLength; i++) {
        TxData[i] = strToSend[i]; // Convert char to u16 for transmission
    }

    #if(SPI_MODE == SLAVE_MODE)
        printf("Slave Mode\r\n");
        Delay_Ms(1000);
    #endif

    SPI_FullDuplex_Init();

    #if(SPI_MODE == HOST_MODE)
        printf("Host Mode\r\n");
        Delay_Ms(2000);
    #endif

    while (1) {
        int i = 0;
        int j = 0;

        while ((i < stringLength) || (j < stringLength)) {
            if (i < stringLength) {
                if (SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) != RESET) {
                    SPI_I2S_SendData(SPI1, TxData[i]);
                    i++;
                }
            }

            if (j < stringLength) {
                if (SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE) != RESET) {
                    RxData[j] = SPI_I2S_ReceiveData(SPI1);
                    j++;
                }
            }
        }
    }
}

```

```

    }

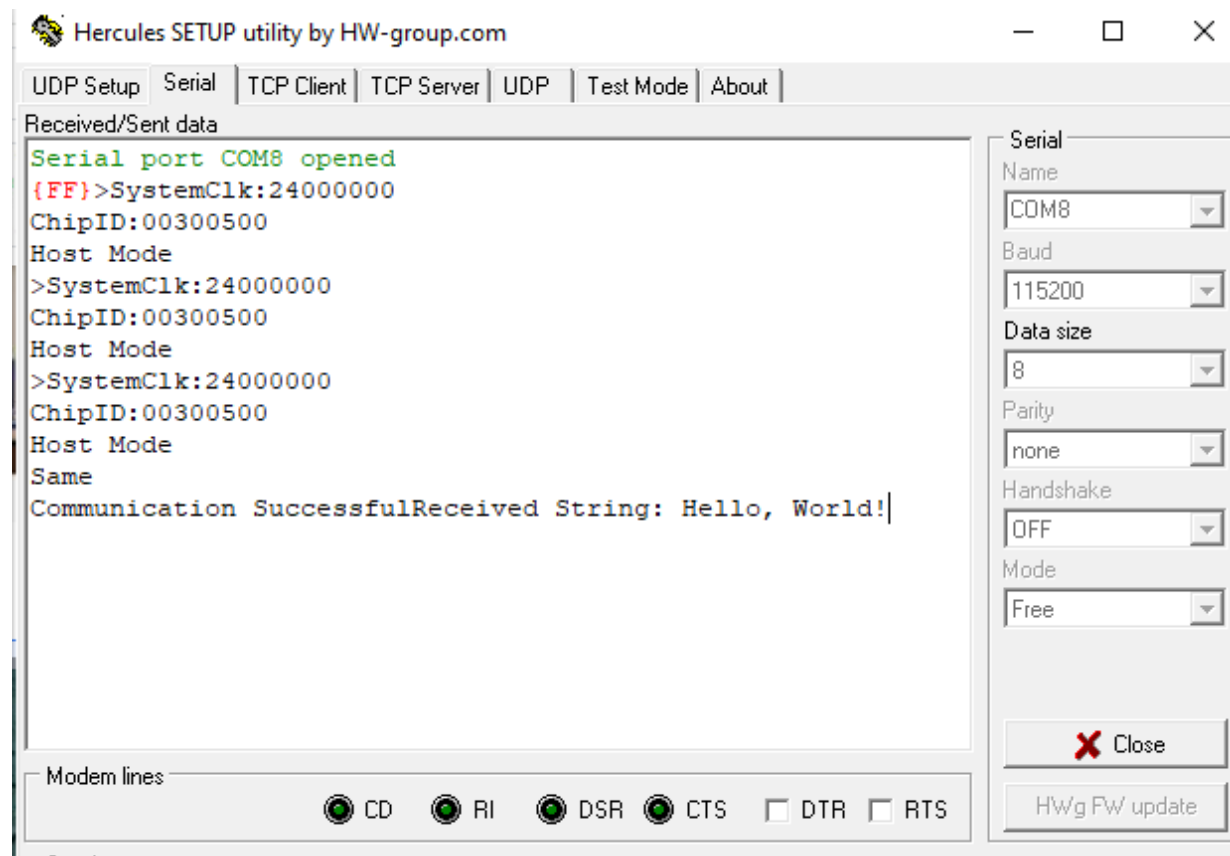
    value = memcmp(TxData, RxData, Size);

    if(value == 0)
    {
        printf("Same\r\n");
    }
    else
    {
        printf("Different\r\n");
    }

    while(1);
}
}

```

7.4 Output



Example 2: Using SPI to Control LED on Slave based on GPIO input of Master

Hardware Connections:

Diagram:

Master Code:

```
#include "debug.h"
#include "string.h"

/* SPI Mode Definition */
#define HOST_MODE      0
#define SLAVE_MODE     1

/* SPI Communication Mode Selection */
#define SPI_MODE       HOST_MODE

/* Global Variables */
#define GPIO_INPUT_PIN  GPIO_Pin_2  // GPIO input pin on Master
#define SLAVE_SELECT_PIN GPIO_Pin_1 // Slave select pin

/*****
 * @fn      SPI_Master_Init
 *
 * @brief   Configuring the SPI for master communication.
 *
 * @return  none
 */
void SPI_Master_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure = {0};
    SPI_InitTypeDef  SPI_InitStructure = {0};

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC | RCC_APB2Periph_SPI1,
ENABLE);

    // Configure GPIO pins for SPI
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1; // NSS
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5; // SCK
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6; // MOSI
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7; // MISO
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
```

```

GPIO_Init(GPIOC, &GPIO_InitStructure);

// Configure GPIO pin for input
GPIO_InitStructure.GPIO_Pin = GPIO_INPUT_PIN;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU; // Pull-up input
GPIO_Init(GPIOC, &GPIO_InitStructure);

SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b; // Use 8-bit data size
SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge;
SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_64;
SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
SPI_InitStructure.SPI_CRCPolynomial = 7;
SPI_Init(SPI1, &SPI_InitStructure);

SPI_Cmd(SPI1, ENABLE);
}

/*****
 * @fn      main
 *
 * @brief   Main program.
 *
 * @return  none
 */
int main(void)
{
    SystemCoreClockUpdate();
    Delay_Init();
    USART_Printf_Init(460800);
    printf("SystemClk:%d\r\n", SystemCoreClock);
    printf("ChipID:%08x\r\n", DBGMCU_GetCHIPID());

    SPI_Master_Init();

    while (1)
    {
        uint8_t gpioState = GPIO_ReadInputDataBit(GPIOC, GPIO_INPUT_PIN);
        uint8_t command = (gpioState == Bit_SET) ? 1 : 0;

        GPIO_SetBits(GPIOC, SLAVE_SELECT_PIN);
        SPI_I2S_SendData(SPI1, command);
        while (SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_BSY) == SET); // Wait
for transmission complete
        GPIO_ResetBits(GPIOC, SLAVE_SELECT_PIN);

        Delay_Ms(100); // Small delay to avoid excessive communication
    }
}

```

Slave Code:

```
#include "debug.h"
```

```

#include "string.h"

/* SPI Mode Definition */
#define SLAVE_MODE    1

/* SPI Communication Mode Selection */
#define SPI_MODE      SLAVE_MODE

/* Global Variables */
#define LED_PIN       GPIO_Pin_0 // LED pin on Slave

/*****
 * @fn          SPI_Slave_Init
 *
 * @brief       Configuring the SPI for slave communication.
 *
 * @return      none
 */
void SPI_Slave_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure = {0};
    SPI_InitTypeDef  SPI_InitStructure = {0};

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC | RCC_APB2Periph_SPI1,
ENABLE);

    // Configure GPIO pins for SPI
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1; // NSS
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5; // SCK
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6; // MOSI
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7; // MISO
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    // Configure GPIO pin for LED
    GPIO_InitStructure.GPIO_Pin = LED_PIN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
    SPI_InitStructure.SPI_Mode = SPI_Mode_Slave;
    SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b; // Use 8-bit data size
    SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
    SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge;
    SPI_InitStructure.SPI_NSS = SPI_NSS_Hard;
    SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_64;

```



```

    SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
    SPI_InitStructure.SPI_CRCPolynomial = 7;
    SPI_Init(SPI1, &SPI_InitStructure);

    SPI_Cmd(SPI1, ENABLE);
}

/*****
 * @fn      main
 *
 * @brief   Main program.
 *
 * @return  none
 */
int main(void)
{
    uint8_t receivedData;

    SystemCoreClockUpdate();
    Delay_Init();
    USART_Printf_Init(460800);
    printf("SystemClk:%d\r\n", SystemCoreClock);
    printf("ChipID:%08x\r\n", DBGMCU_GetCHIPID());

    SPI_Slave_Init();

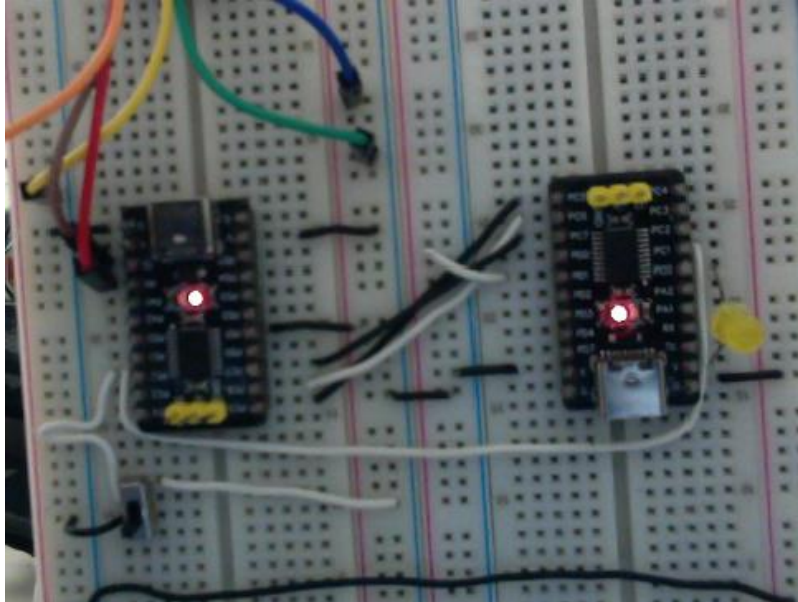
    while (1)
    {
        while (SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE) == RESET); //
Wait for data reception
        receivedData = SPI_I2S_ReceiveData(SPI1);

        if (receivedData == 1)
        {
            GPIO_SetBits(GPIOC, LED_PIN); // Turn LED ON
        }
        else
        {
            GPIO_ResetBits(GPIOC, LED_PIN); // Turn LED OFF
        }
    }
}

```

Implementation:

GPIO Input on Master = LOW → LED on Slave is OFF



GPIO Input on Master = High → LED on Slave is ON

