Serial Peripheral Interface (SPI) Bus Tutorial

Introduction

The Serial Peripheral Interface (SPI) bus was developed by Motorola to provide full-duplex synchronous serial communication between master and slave devices. The SPI bus is commonly used for communication with flash memory, sensors, real-time clocks (RTCs), analog-to-digital converters, and more.

As shown in Figure 1, standard SPI masters communicate with slaves using the serial clock (SCK), Master Out Slave In (MOSI), Master In Slave Out (MISO), and Slave Select (SS) lines. The SCK, MOSI, and MISO signals can be shared by slaves while each slave has a unique SS line.

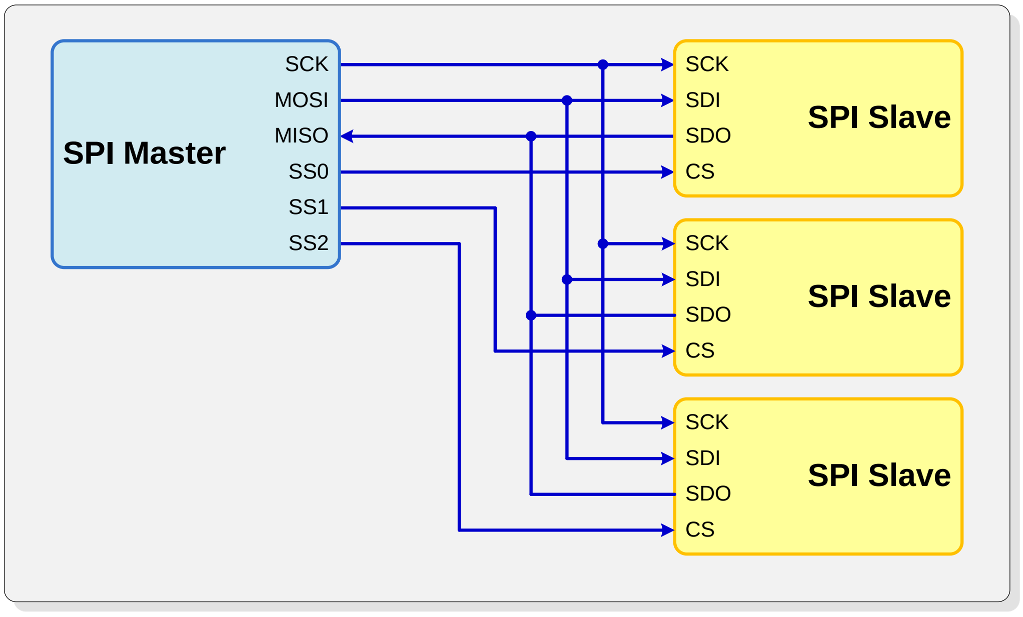


Figure 1. 4-wire SPI bus configuration with multiple slaves

Polarity and Clock Phase

The SPI bus defines no protocol for data exchange, limiting overhead and allowing for high speed data streaming. Clock polarity (CPOL) and clock phase (CPHA) can be specified as ‘0’ or ‘1’ to form four unique modes to provide flexibility in communication between master and slave as shown in Figure 2.

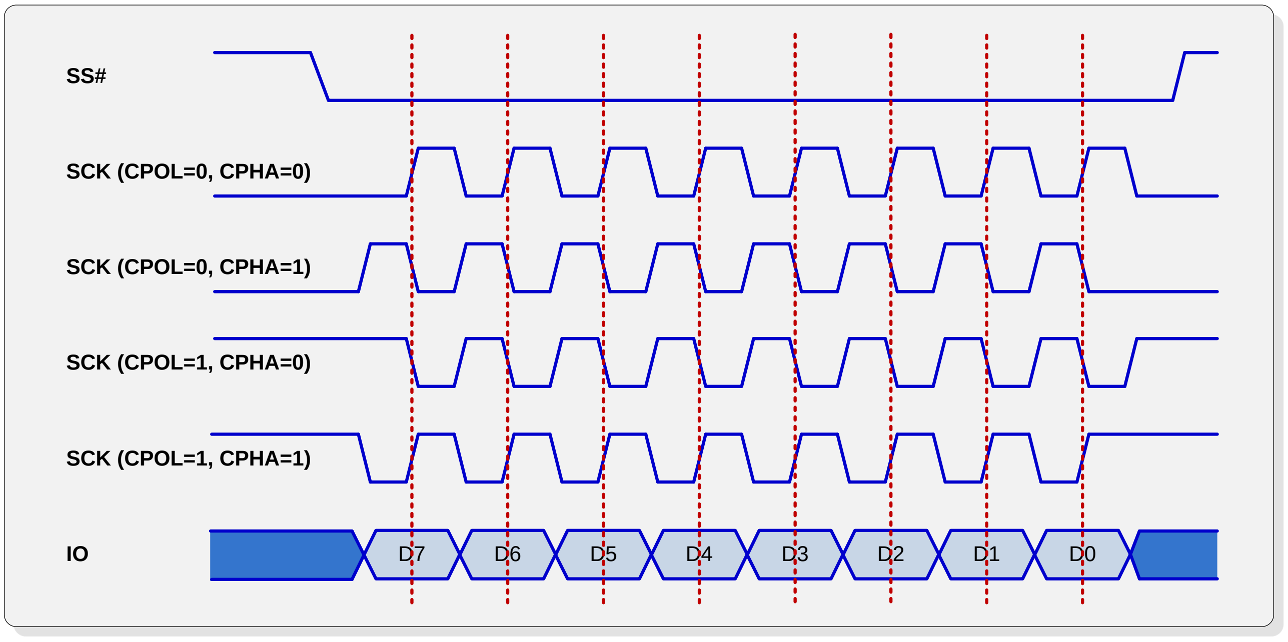


Figure 2. SPI bus timing

If CPOL and CPHA are both ‘0’ (defined as Mode 0) data is sampled at the leading rising edge of the clock. Mode 0 is by far the most common mode for SPI bus slave communication. If CPOL is ‘1’ and CPHA is ‘0’ (Mode 2), data is sampled at the leading falling edge of the clock. Likewise, CPOL = ‘0’ and CPHA = ‘1’ (Mode 1) results in data sampled at on the trailing falling edge and CPOL = ‘1’ with CPHA = ‘1’ (Mode 3) results in data sampled on the trailing rising edge. Table 1 below summarizes the available modes.

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Table 1. SPI mode definitions

3-Wire and Multi-IO Configurations

In addition to the standard 4-wire configuration, the SPI bus has been extended to include a variety of IO standards including 3-wire for reduced pin count and dual or quad I/O for higher throughput.

In 3-wire mode, MOSI and MISO lines are combined to a single bidirectional data line as shown in Figure 3. Transactions are half-duplex to allow for bidirectional communication. Reducing the number of data lines and operating in half-duplex mode also decreases maximum possible throughput; many 3-wire devices have low performance requirements and are instead designed with low pin count in mind.

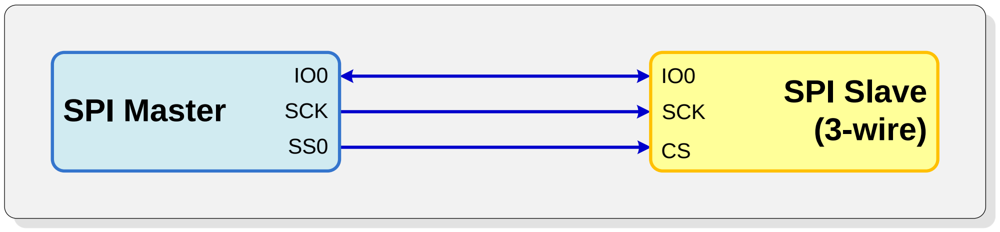


Figure 3. 3-wire SPI configuration with one slave

Multi I/O variants such as dual I/O and quad I/O add additional data lines to the standard for increased throughput. Components that utilize multi I/O modes can rival the read speed of parallel devices while still offering reduced pin counts. This performance increase enables random access and direct program execution from flash memory (execute-in-place).

Quad I/O devices can, for example, offer up to 4 times the performance of standard 4-wire SPI when communicating with a high speed device. Figure 4 shows an example of a single quad IO slave configuration.

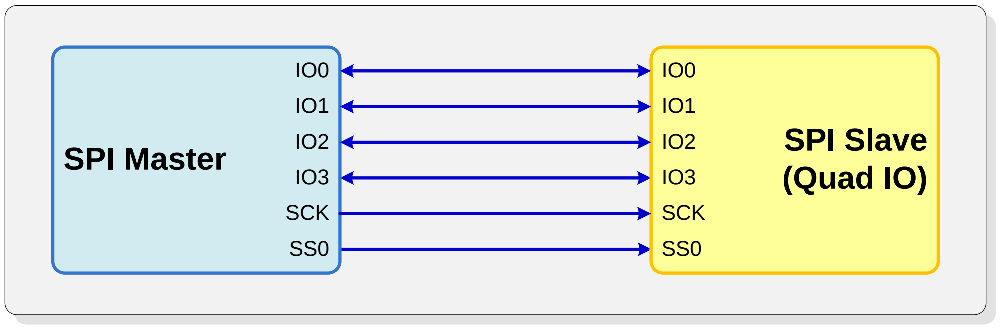


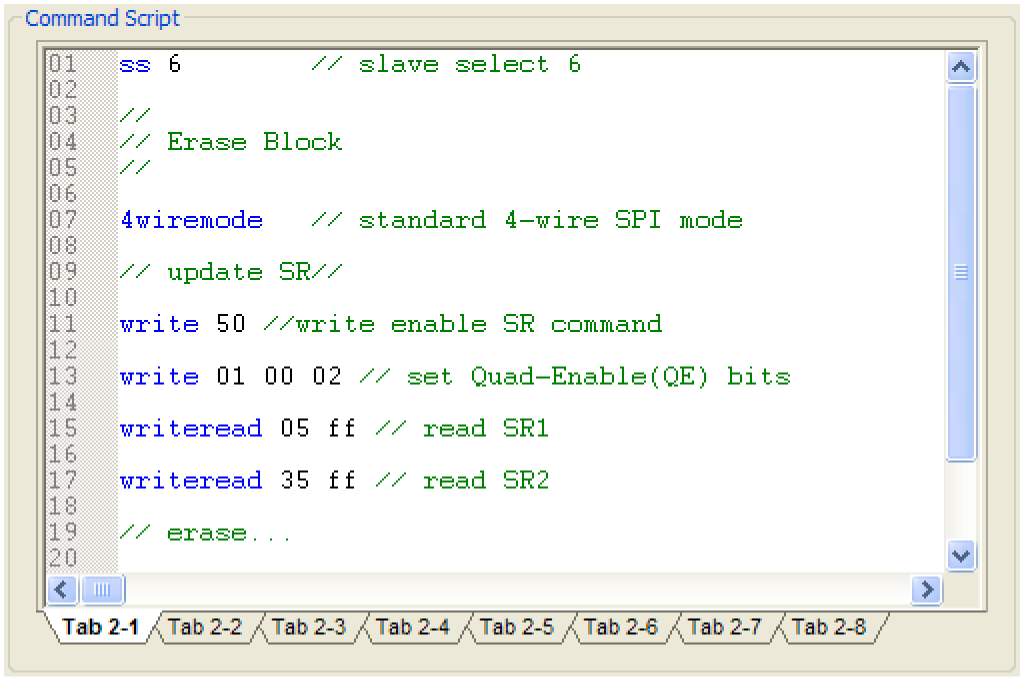
Figure 4. Quad IO SPI configuration with one slave

Example SPI Transactions

The SPI protocol does not define the structure of the data stream; the composition of data is completely up to the component designer. However, many devices follow the same basic format for sending and receiving data, allowing interoperability between parts from different vendors.

Corelis BusPro-S Host Adapter

In this document, we'll refer to Corelis SPI Exerciser Debugger code for creating SPI transactions. The Debugger module command script interface depicted in Figure 5 supports a simple command language for communication with SPI slave devices.



```
ss 6          // slave select 6

//
// Erase Block
//

4wiremode   // standard 4-wire SPI mode

// update SR//

write 50 //write enable SR command

write 01 00 02 // set Quad-Enable(QE) bits

writeread 05 ff // read SR1

writeread 35 ff // read SR2

// erase...
```

Figure 5. SPI Exerciser Debugger command script interface

The BusPro-S includes automatic slave select (SS) signal handling—once a slave select signal has been selected with the “SS” command or defined in the application interface, the software will assert that slave select signal at the beginning of each command and de-assert the signal to complete the transaction. In this document, we will always explicitly specify the slave select state with “SSON” and “SSOFF” commands. In practical use, this is only required for grouping multiple commands into a single continuous transaction.

Simple Write Transaction

Most SPI flash memories have a write status register command that writes one or two bytes of data, as shown in Figure 6. To write to the status register, the SPI host first enables the slave select line for the current device. The master then outputs the appropriate instruction followed by two data bytes that define the intended status register contents. Since the transaction does not need to return any data, the slave device keeps the MISO line in a high impedance state and the master masks any incoming data. Finally, slave select is de-asserted to complete the transaction.

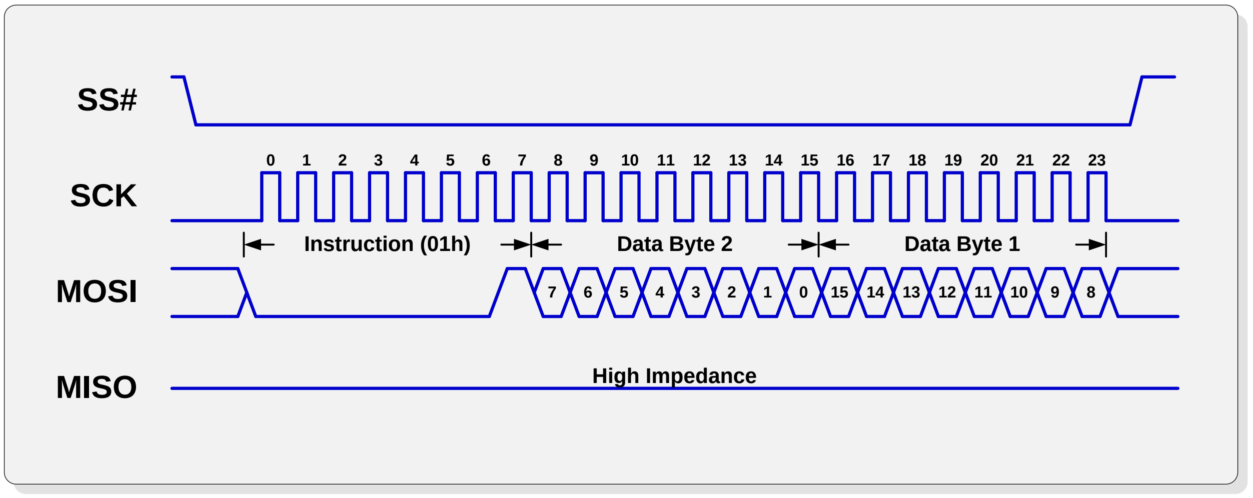


Figure 6. Write command using a single-byte instruction and two-byte data word

Using the Corelis SPI Exerciser command language, this transaction can be accomplished with the following code, where Data Byte 2 is “55” and Data Byte 1 is “AA”. For brevity we are using the short version of commands; the command script language supports both full commands such as “write, read” as well as abbreviated versions “wt, rd”. Figure 7 lists example code for a simple write transaction.

```
sson            // Activate slave select
   wt 01 55 AA // Write instruction 01h and data bytes 55h, AAh
ssoff           // Deactivate slave select
```

Figure 7. Example code for a simple write transaction

Simple Read Transaction

A status register read transaction would be similar to the write transaction, but now takes advantage of data returned from the slave as shown in Figure 8. After sending the read status register instruction, the slave begins transmitting data on the MISO line at a rate of one byte per eight clock cycles. The host receives the bitstream and completes the transaction by de-asserting SS#.

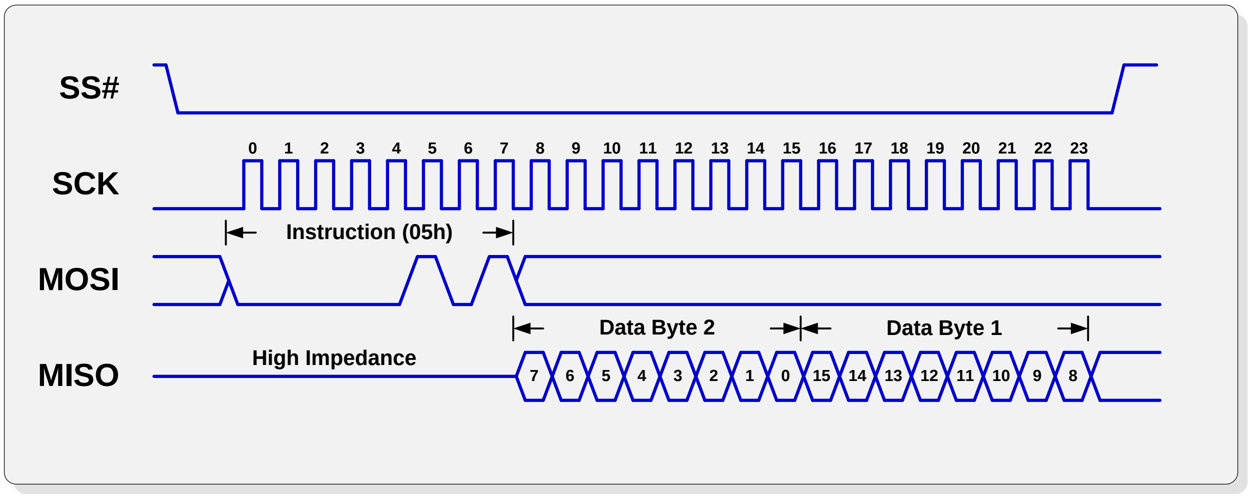


Figure 8. Read command using a single-byte instruction and two-byte data word

This sequence is really a single-byte write followed by a two byte read; it can be considered a three-byte combined write/read command or a single byte write and two-byte read. We can create this transaction with the Corelis SPI Exerciser debugger command sequence shown in Figure 9.

```
sson        // Activate slave select
   wt 05    // Write instruction 05h
   rd 2     // Read two data bytes
ssoff       // Deactivate slave select
```

Figure 9. Example code for a simple read transaction

The data values from the read command will be displayed in the transaction log. The transaction log displays the data being transferred between the BusPro-S host and slave when performing read, write, or combined write/read commands.

Quad IO Transaction

Quad IO is gaining popularity with flash memories for its increased performance. Instead of using a single output and single input interface, Quad IO utilizes 4 separate half-duplex data lines for both transmitting and receiving data for up to four times the performance of standard 4-wire SPI.

Figure 10 shows an example read command for a Spansion S25FL016K serial NOR flash device. To read from the device, a fast read command (EBh) is first sent by the master on the first IO line while all others are tristated. Next, the host sends the address; since the interface now has 4 bidirectional data lines, it can utilize these to send a complete 24-bit address along with 8 mode bits in just 8 clock cycles. The address is then followed with 2 dummy bytes (4 clock cycles) to allow the device additional time to set up the initial address.

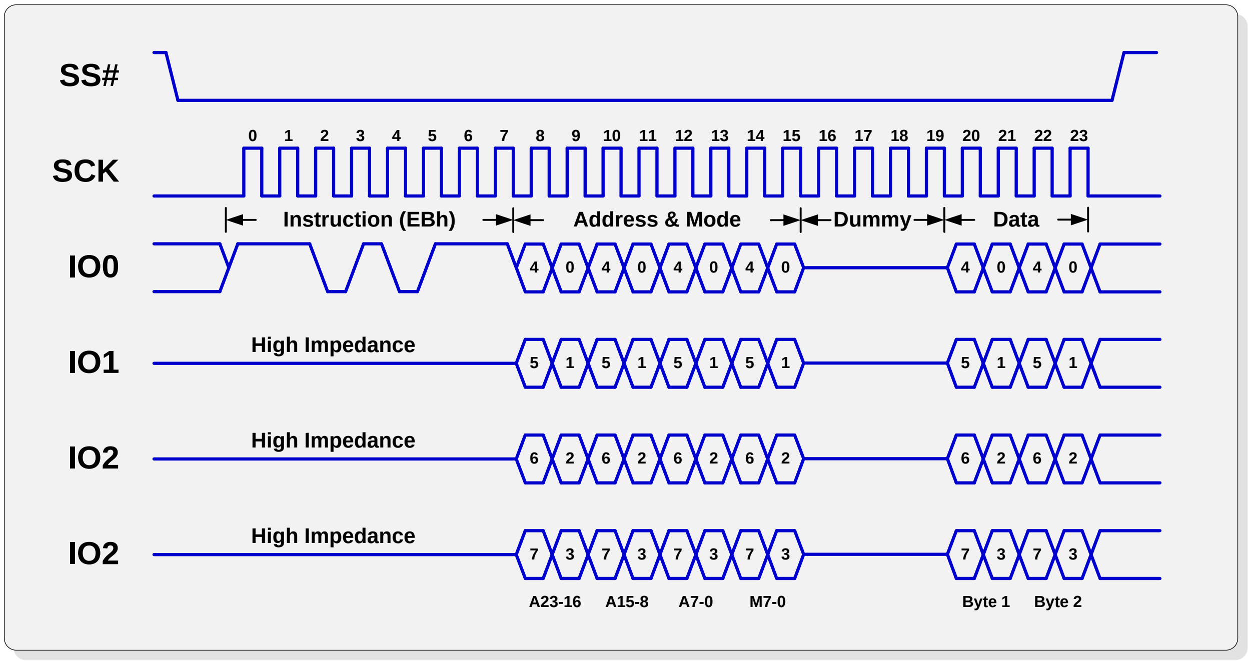


Figure 10. Quad mode fast read sequence for Spansion S25FL016K or equivalent

After the address cycle and dummy bytes have been sent by the host, the component begins sending data bytes; each clock cycle consists of a data nibble spread across the 4 IO lines, for a total of two clock cycles per byte of data. Compare this to the 16 clock cycles required for our simple read transaction and it’s easy to see why quad mode is gaining popularity for high speed flash memory applications!

To create this sequence in the SPI Exerciser command language, we would use the example code shown in Figure 11.

```
4m                // Start in 4-wire mode
sson              // Activate slave select
   wt EB          // Write instruction EBh
   qm             // Switch to quad mode
   wt AA AA AA 00 // Write 3-byte address and 8 read mode bits
   wt 55 55       // Write 2 dummy bytes
   rd 2           // Read two data bytes
ssoff             // Deactivate slave select
```

Figure 11. Example code for a quad mode read transaction

Note that we're changing from 4-wire mode to quad mode in the middle of the transaction. In quad mode, the software automatically distributes the data bytes among the IO lines using the same bit pattern depicted in Figure 10 above.

Conclusion

The SPI bus is straightforward and versatile, enabling simple and fast communication with a variety of peripherals. A high speed multi-IO mode host adapter like the Corelis BusPro-S can be an invaluable tool in debugging as well as adding SPI communication capabilities to any test system. For more information about Corelis serial bus products, visit the Corelis website at http://www.corelis.com/products-bus-analyzers/.