TECHNICAL NOTE

TN200

SPI Using the Rabbit Clocked Serial Ports

The Serial Peripheral Interface (SPI) is a four-wire full-duplex synchronous serial data link that is implemented in many microcontrollers and peripheral devices. The SPI was originally developed by Motorola to enable a glueless microcontroller interface with industry-standard serial devices such as serial EEPROMs, data converters, liquid crystal displays, as well as other peripherals and microcontrollers.

The SPI consists of shift registers that serially transmit and receive data at the same time. For serial communication to take place within a SPI-based system, one device has to act as a master and at least one other device has to act as a slave. In this form of communication, the master device controls clock generation and the flow of data, while the slave or slave units serially shift data in and out. Note that while one master device can transmit data to multiple slaves, only one slave can transmit data back to the master at any particular time. In a system where multiple microprocessors communicate with each other, the task of being the master SPI device can be assigned to any processor within the system.

This technical note provides a brief overview of master/slave SPI communication on the Rabbit and prescribes general guidelines on how to configure the Rabbit as a master SPI. The reason for focussing on the Rabbit as a master SPI is that, in most cases, the Rabbit will be communicating as a master with one or more peripheral devices via the SPI.

SPI I/O Signals

There are normally four I/O signals associated with SPI transfers. Motorola refers to them as Serial Clock (SCK), Master Out Slave In (MOSI), Master In Slave Out (MISO), and Slave Select (SS). Table 1 shows the correspondence between the Motorola and Rabbit SPI signals.

Table 1. Correspondence between Motorola and Rabbit SPI Signals (Master Mode)

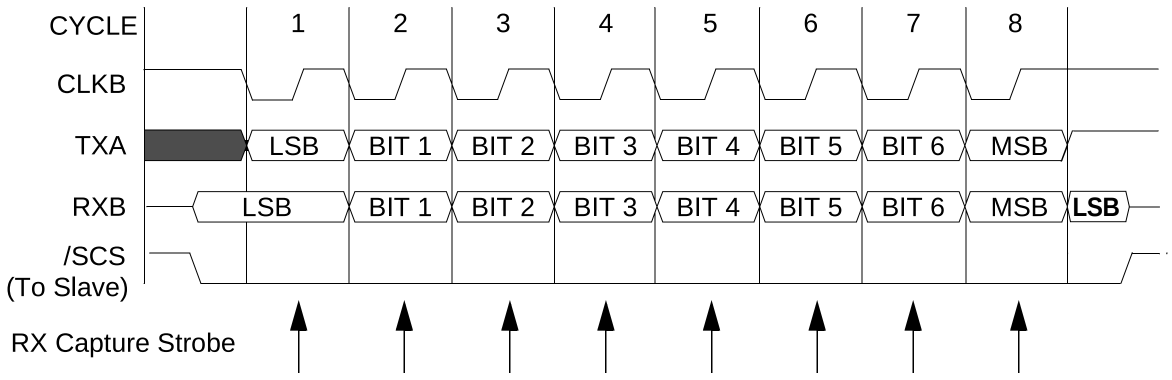
Motorola Signal Names	Rabbit Signal Names	Pin Function
SCK	CLKA or CLKB	Serial Clock
MOSI	TXA or TXB on parallel port C* ATXA or ATXB on parallel port D	Master Out Slave In
MISO	RXA or RXB on parallel port C ARXA or ARXB on parallel port D	Master In Slave Out
SS	Any general-purpose I/O can be configured for the Slave Select function	Slave/Chip Select

* Serial ports A and B can be multiplexed between parallel ports C and D; therefore, Serial ports A or B can be only configured to operate via one of the parallel ports at any particular time.

Serial Clock (SCK) Signal

The SCK signal functions differently depending on whether the processor is configured as a master or as a slave. In addition to synchronizing data communication between itself and a slave device, a master SPI automatically generates eight clock cycles every time it initiates a transfer. The Read/Write operation takes place within the same clock cycle in both the master and the slave devices. The Motorola SPI provides support for a user-configurable clock edge polarity and clock phase to accommodate various SPI transfer protocols. The Rabbit 2000 SPI is compatible with only one of the Motorola transmission formats, which is shown in Figure 1. The Rabbit 3000 supports all four Motorola transmission formats.

Figure 1. Rabbit-Compatible SPI Timing Diagram



NOTE: The Rabbit uses a LSB first format whereas the Motorola SPI uses a MSB first format.

MOSI and MISO Signals

From the standpoint of the master, MOSI and MISO are basically transmit and receive signals. The signals have opposite roles when viewed from the slave device. In a system where a single master communicates with multiple slaves, the clock signal from the master is tied to the clock signal of every slave device, the transmit line from the master is tied to the receive line of every slave, and only one slave device optionally shifts data back to the master.

SS Signal

The behavior and function of the slave/chip select is different depending on whether the device is configured as a master or a slave. In a master SPI device, the chip select normally acts as an active low output signal to enable or disable the slave device. Alternatively, the master can be configured to treat the chip select signal as an error-detection input.

SPI on the Rabbit

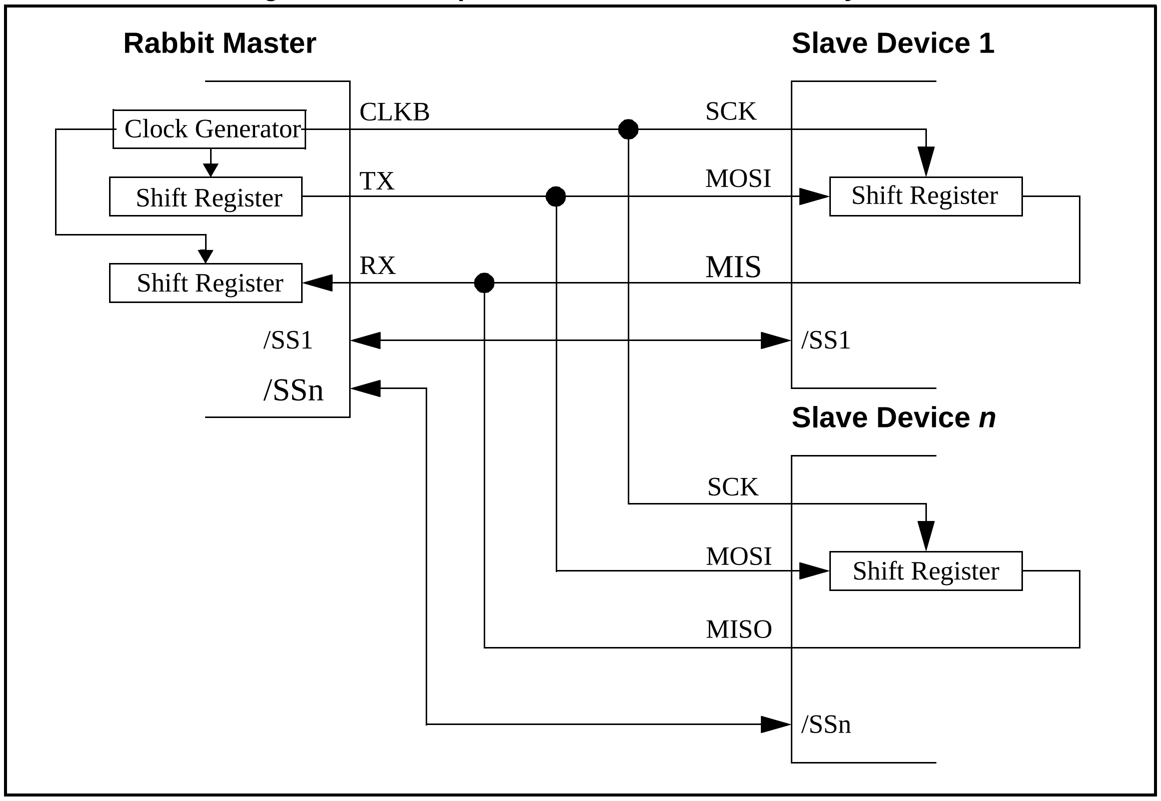
Two of the four serial ports on the Rabbit 2000 can be configured for synchronous communication with SPI devices—serial ports A and B can be set up to operate in full- or half-duplex clocked serial modes. Four of the six serial ports on the Rabbit 3000—serial ports A, B, C and D—can be configured for synchronous communication with SPI devices. The Rabbit can interface with microprocessor or peripheral devices that include an SPI for full-duplex synchronous serial communication.

The Rabbit is capable of operating as either a master or a slave device in the SPI mode. As a master device, the Rabbit provides the serial clock and initiates data transmission for SPI communication; when the Rabbit is used as a slave device, the clock signal is an input to the Rabbit, and data are shifted in and out of the Rabbit by the master. Regardless of whether the Rabbit is used as a master or as a slave, data are transmitted on the falling edge of the clock, and the received data are sampled on the rising edge of the clock.

Depending on the type of slave select signal required, an available I/O port pin can be configured to generate any type of fault error signal to the master or receive any type of slave select notification from the master.

Figure 2 shows a typical full-duplex master/slave SPI-based system consisting of a Rabbit master and multiple slave devices.

Figure 2. Full-Duplex Master/Slave Based SPI System



Using the Dynamic C SPI Driver

An SPI driver is included with Dynamic C starting with version 7.05. The `SPI.LIB` library in the `\LIB\SPI` directory implements the driver. The driver performs the required tasks discussed in the final two sections of this technical note, as well as the bit reversal required to be compatible with the Motorola SPI format. The comments in the source code for the `SPI.LIB` library provide additional details about the driver. The following API functions are included in the `SPI.LIB` library.

- `SPIinit()` — initializes the SPI port parameters for a serial interface.
- `SPIwrite()` — writes a block of bytes to the SPI port.
- `SPIread()` — reads a block of bytes from the SPI port.
- `SPIWrRd()` — reads and writes a block of bytes from/to the SPI port.

If you use the Dynamic C SPI driver, it is not necessary to read the remainder of this technical note.

Configuring the Rabbit for SPI Communication

Since serial port A is normally used for programming and debugging, we recommend that it not be used for SPI unless absolutely necessary.

The following section outlines the steps necessary to configure serial port B on parallel port C to operate in SPI mode:

1. Configure Parallel Port C Function Register.

Setting bit 4 in the PCFR register will configure parallel port C for its Alternate Output function as the serial port B output. Note that the output for the programming port (serial port A) is also configured with PCFR; therefore, bit 6 must also be set so that the serial port A output does not become disabled. The address of PCFR is 0x055.

Table 2. Parallel Port C Function Register (PCFR)

Bit(s)	Value	Description
7:0	0	The corresponding port bit functions normally.
	1	The corresponding port bit carries its alternate signal as an output. See Table 3 below. Only the bits that have alternate functions listed in Table 3 actually have a control bit in these registers.

Table 3. Alternate Output Function for Parallel Port C

bit	Port C
7	
6	TXA
5	
4	TXB
3	
2	TXC
1	
0	TXD

2. Configure Timer A5.

There are two steps that need to be taken to configure Timer A5 to generate the clock signal for serial port B. The first step is to load the Timer A5 Time Constant Register (TAT5R) with a user-defined value. The timer times out twice for every bit that is transmitted. For example, a time constant of 31 corresponds to 128 clock cycles for each bit being transmitted, that is, $4 \times (31 + 1)$. The second step is to set bit 1 in the Timer A Control/Status Register (TACSR) to enable the Timer A clock. Note that the processor is already configured for serial communication in Rabbit-based products, and so the main clock for Timer A is already enabled, making it unnecessary to enable the main clock a second time. The address of TAT5R is 0xAB.

Table 4. Timer A5 Time Constant Register (TAT5R)

Bit(s)	Value	Description
7:0	write	The time constant for the Timer A counter is stored. This time constant will take effect the next time that the Timer A counter counts down to zero. The timer counts modulo n+1, where n is the programmed time constant.

The address of TACSR is 0xA0.

Table 5. Timer A Control/Status Register (TACSR)

Bit(s)	Value	Description
7:4,1 (rd only)	0	The corresponding Timer A counter has not reached its terminal count.
	1	The corresponding Timer A counter has reached its terminal count. These status bits (not the interrupt enable bits) are cleared by the read of this register, as is the Timer A interrupt.
7:4,1 (wr only)	0	The corresponding Timer A interrupt is disabled.
	1	The corresponding Timer A interrupt is enabled.
3:2	00	These bits are unused and always read as zeros.
0	0	The main clock for Timer A is disabled.
	1	The main clock (CLK/2) for Timer A is enabled.

3. Configure Serial Port B Control Register (SBCR).

Serial port B can be configured to operate with or without interrupts. Interrupts can be disabled or enabled, and a serial port interrupt priority can be assigned by setting bits 0 and 1 in SBCR. (The drivers in the Dynamic C `SPI.LIB` library do not use interrupts.) The user must also decide on the source of the serial clock. If the Rabbit is used as a master SPI device, bits 3 and 2 in SBCR must be set to 11 so that the clock signal is generated internally by the Rabbit. If the Rabbit is used as a slave, bits 3 and 2 in SBCR must be set to 10 so that the clock signal is generated by the external master device. To instruct the processor which parallel port to use for serial data input, set bits 4 and 5—setting them to 00 instructs the processor to use parallel port C for input; setting them to 01 instructs the processor to use parallel port D for input.

It is important to note that the commands to begin a transmit or receive operation are not issued at this time, so bits 6 and 7 must be reset to zero. The address of SBCR is 0xD4.

Table 6. Serial Port B Control Register (SBCR)

Bit(s)	Value	Description
7:6	00	No operation. These bits are ignored in the async mode.
	01	In clocked serial mode, start a byte receive operation.
	10	In clocked serial mode, start a byte transmit operation.
5:4	00	Parallel port C is used for input.
	01	Parallel port D is used for input.
	1x	Disable the receiver input.
3:2	00	Async mode with 8 bits per character.
	01	Async mode with 7 bits per character. In this mode the most significant bit of a byte is ignored for transmit, and is always zero in receive data.
	10	Clocked serial mode with external clock.
	11	Clocked serial mode with internal clock.
1:0	00	The serial port interrupt is disabled.
	01	The serial port uses Interrupt Priority 1.
	10	The serial port uses Interrupt Priority 2.
	11	The serial port uses Interrupt Priority 3.

The above steps outline how to configure serial port B for synchronous serial communication. The following steps describe the method, requirements, and restrictions for transferring a byte in the SPI mode.

Transmitting and Receiving Data in SPI Mode

1. Write transmit data.

After properly preparing the Rabbit for serial communication, the first byte of data to be transmitted must be loaded into the serial port B Data register (SPDR). At this point no data will be sent because the command to transmit data has not been issued yet. Note that the commands to transmit and receive data are issued by writing to bits 6 and 7 in SBCR; however, they must be issued in a certain sequence in order to guarantee the timing required for full-duplex communication.

2. Issue the transmit and receive commands.

The key to configuring serial port B for full-duplex communication is to perform two writes to the control register. The data written to SBCR is the same in both cases except for what is written to bits 6 and 7. The first time the command is issued, bit 7 must be set to start a byte-transmit operation; the second time, bit 6 must be set to start a byte-receive operation. The two writes are necessary because the Rabbit has two separate shift registers, one for the transmitter and one for the receiver. Also note that the sequence in which the commands are issued is very important. The TX command must be issued first, followed within one-half bit time by the RX command. It is critical to issue the RX command in time because once the clock is enabled and data are loaded in the transmit buffer, the transmitter will start sending on the falling edge of the serial clock and the receiver will start sampling on the rising edge of the serial clock. Maintaining the proper timing relationship guarantees that data transmitted and received occur within the same clock phase. The functional timing diagram in Figure 1 shows the relationship among Serial Clock, Serial Data I/O, and Slave Chip Select. Note that, depending on the type of slave device interfaced to the Rabbit, the slave may require that the Slave Chip Select Line (/SCS) be toggled between each byte transmitted.

Bits 6 and 7 in SBCR may be set at the same time on the Rabbit 3000 to execute a simultaneous send/receive command.

The maximum serial bit time depends on how fast the RX command is issued. The following sample code shows that the maximum serial bit time has to be slower than eleven peripheral clocks, which is the time required to issue the RX command immediately following the TX command.

```
;****** Prepare clocked serial port to receive a byte    ******

ld hl,00d4h           ; (6-clocks) SBCR data: Int. Clk, Start Byte RX, INTR 1
ioi ld (hl),h          ; (5-clocks) write to SBCR
```

3. Polled or interrupt-driven serial communication.

The decision to employ a polled or interrupt-driven scheme to handle transmit and receive functions is entirely up to the user. The Rabbit contains hardware facilities to support both. Polling is by far the easiest.

4. Interrupt-driven communication.

If an interrupt-driven scheme is used, there are a few issues that a user must pay attention to.

Because serial port B does not have separate vectors for receive and transmit interrupts, when an interrupt occurs, the serial port B Status register (SBSR) must be read every time to determine the cause of the interrupt. SBSR is at address 0xD3.

Table 7. Serial Port B Status Register (SBSR)

Bit(s)	Value	Description
7	0	The receive data register is empty
	1	There is a byte in the data register of the receiver. The serial port will request an interrupt when the receiver sets this bit. The interrupt is cleared when the receiver buffer is read.
6	0	The byte in the receiver buffer is data.
	1	The byte in the receiver buffer is an address.
5	0	The receiver buffer was not overrun.
	1	The receiver buffer was overrun. This bit is cleared when the receiver buffer is read.
4		This bit is always zero.
3	0	The transmit data register is empty.
	1	The transmit data register is full. The serial port will request an interrupt when the transmitter clears this bit. The interrupt is cleared when the transmit data register is written, or any value (which will be ignored) is written to this register.
2	0	The transmitter is idle.
	1	The transmitter is sending a byte. The serial port will request an interrupt when the transmitter clears this bit, which occurs only if the transmitter is ready to start sending another byte but the transmit buffer is empty. The interrupt is cleared when the transmit data register is written, or any value (which will be ignored) is written to this register.
1:0		These bits are always zero.

In the case of a transmit, an interrupt is generated when a byte (in this case the first byte) is transferred from the buffer to the shift register; that is, when the buffer becomes empty. (Note that the Rabbit employs one level of buffering on the serial ports.) Since we're not ready to send another byte, we have to clear the interrupt by performing a dummy write to the Serial Port B Status Register (SBSR). The act of writing to the status register clears the transmit interrupt.

References

Motorola SPI Specifications, M68HC11 Reference Manual, Chapter 8, pp. 8-1 through 8-22.