Wisconsin MMC
FPGA SPI Configuration Interface

31-Jan-2013

T. Gorski,
University of Wisconsin

This specification describes the enhanced SPI interface supported by the Wisconsin MMC, the purpose of which is to act as a geographically-addressable configuration interface for FPGA-based AMC cards. The MMC acts as the SPI master, supporting up to 3 FPGAs as slave devices.

Signal Definition. The table below lists the on-board signals used to implement the interface.

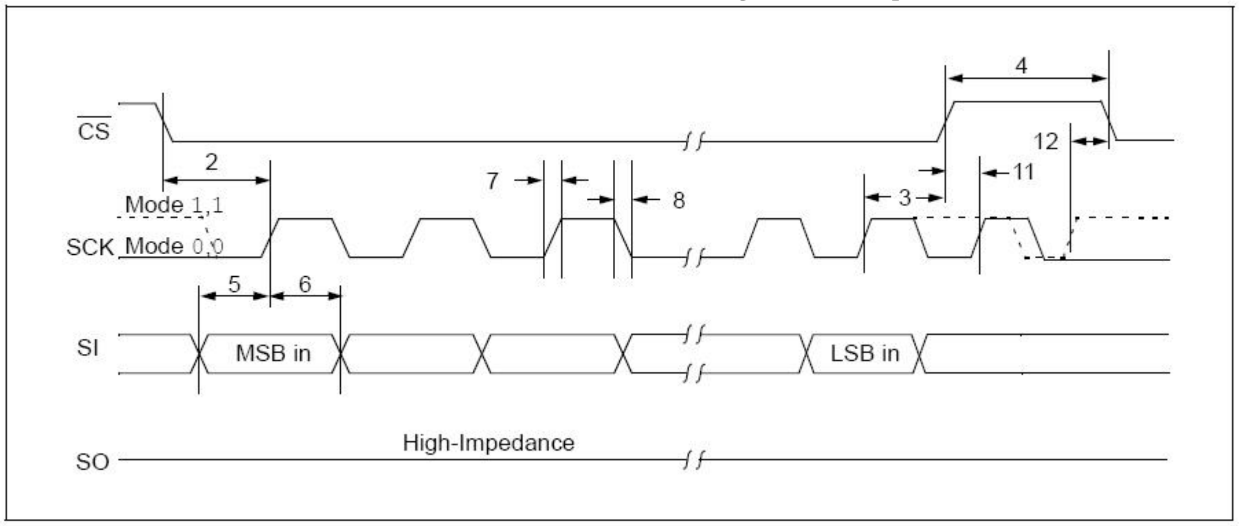
Signal Name	Signal Direction at FPGA	Description
FSIO_SCK	*Input*	SPI Interface serial clock. Operates at 4 MHz.
FSIO_MOSI	*Input*	SPI Interface Data Out from Master. Connected to all slave FPGA devices.
FSIO_MISO	*Output*	SPI Interface Data In to Master. Connected to all slave FPGA devices, and driven by the selected slave.
#FSIO_SS0 -- #FSIO_SS2	*Bidirectional*	SPI Interface Slave Select/Slave Device ID. Connected one signal to each FPGA used. When #FSIO_SCANSLV is inactive (high), acts as the SPI slave-select for that device. When #FSIO_SCANSLV is active (low), drives a logic 0, indicating a slave connection to the MMC on that line.
#FSIO_SCANSLV	*Input*	Slave ID request line. Connected to all devices. Inactive (high) during normal SPI operation. When driven low, FPGA slaves should respond by driving their #FSIO_SSx line low.

Functional Description. The functional behavior for the FPGA-based SPI slave is based on the Microchip 25LC640A EEPROM. Specifically, a byte data interface, supported by a 16-bit address space, with a Control and Status register. Note that there are differences in bit definitions in those registers between this FPGA application and the EEPROM.

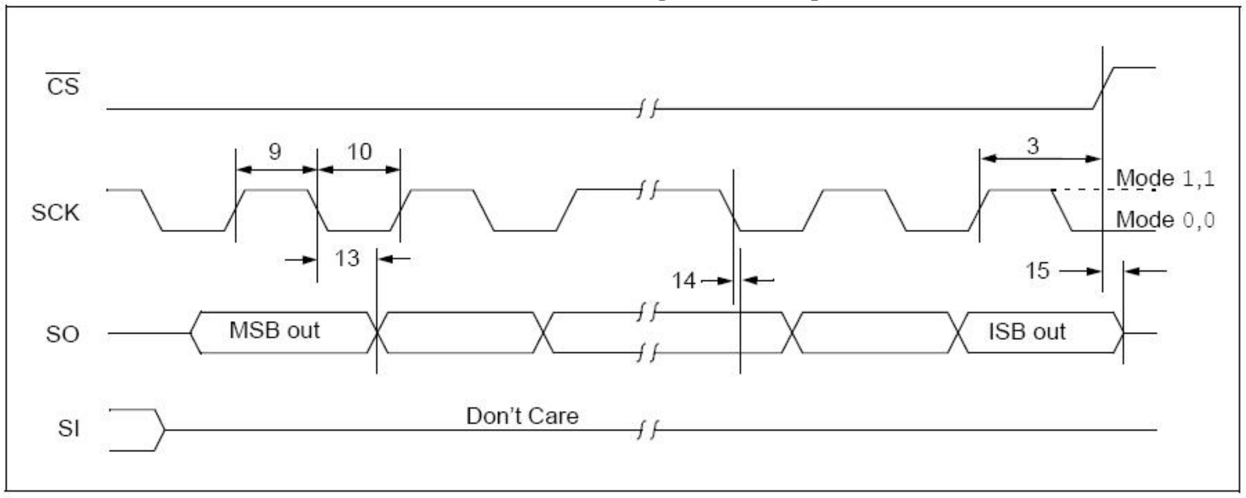
Signal transitions on the SPI interface are shown in the figures below, which are extracted from the EEPROM data sheet. For input data (to the FPGA) on the MISO line, the MMC drives the MSB of the first byte after Slave Select is active, and the subsequent bits *on the falling edge of each clock pulse thereafter*. Thus the FPGA slave should capture the MSB of the first serial input bit on the first rising edge of SCK, and subsequent bits *on the rising edge of each SCK clock pulse thereafter.*

For output data from the FPGA to the MMC, the roles are reversed, with the FPGA putting the output data on the MISO line, MSB first, on the falling edge of SCK, and the MMC capturing that bit on the next rising edge of SCK.

MOSI / Slave-Select / SCK Timing Relationship



MISO / SCK Timing Relationship



Register Definitions. While the FPGA SPI configuration interface uses a Control and Status register similar to the 25LC640A EEPROM, the bit definitions are specific to this application, and thus different from those in the EEPROM. The tables below describe the control and status register bit definitions.

The Status Register is a read-only register consisting of 4 defined bits. Two of the bits, HF1 and HF2, are available for general-purpose handshaking between application software and the FPGA. They are supported by the MMC IPMI Command Interface, but not part of the FPGA configuration sequence. The remaining two bits, CFGRDY, and REQCFG, are part of the configuration sequence, and their functionality must be implemented and used as specified.

Status Register Bit Definitions

Bit	Name	Description
7	*HF1*	Handshake Flag 1. A generic handshake flag for coordinating post-configuration data transfers between the FPGA and application software.

6	*HF2*	Handshake Flag 2. A generic handshake flag for coordinating post-configuration data transfers between the FPGA and application software.
5	*CFGRDY*	Configuration Ready. Set by the IPMI Management System (via Config register access) after delivery of a configuration byte stream via the FPGA SPI interface. Setting of this bit indicates to the FPGA that the config data is ready for use in the FPGA SPI memory.
4	*REQCFG*	Request Configuration. Set by the FPGA after device configuration (flash memory load). When the IPMI management system detects that this bit is set, it will transfer the appropriate configuration data to memory in the FPGA SPI interface.
3—0	*Reserved*	

The Config Register is a write-only register, the purpose of which is to set or clear specific Status Register bits via the FPGA SPI interface. Bits are set or cleared by setting the corresponding marker bit, and the set or clear flag as appropriate. Bits which are unmarked will not be changed by the Config Register write. If neither or both SETFLG and CLRFLG bits are set, then there should be no change to the Status Register bits.

Config Register Bit Definitions

Bit	Name	Description
7	*HF1*	Marker for Handshake Flag 1. If this bit is set, it marks the HF1 bit in the status register for setting or clearing (depending on the state of bits 0-1 in the write data byte).
6	*HF2*	Marker for Handshake Flag 2. If this bit is set, it marks the HF2 bit in the status register for setting or clearing (depending on the state of bits 0-1 in the write data byte).
5	*CFGRDY*	Marker for Config Ready. If this bit is set, it marks the CFGRDY bit in the status register for setting or clearing (depending on the state of bits 0-1 in the write data byte).
4—2	*Reserved*	
1	*SETFLG*	Set Flag. Set this bit to indicate that the marked bits should be set in the Status Register.
0	*CLRFLG*	Clear Flag. Set this bit to indicate that the marked bits should be cleared in the Status Register.

SPI Interface Commands. All SPI operations begin with the transmission of an 8-bit command word on the MOSI line. The commands and their byte values given in the following table.

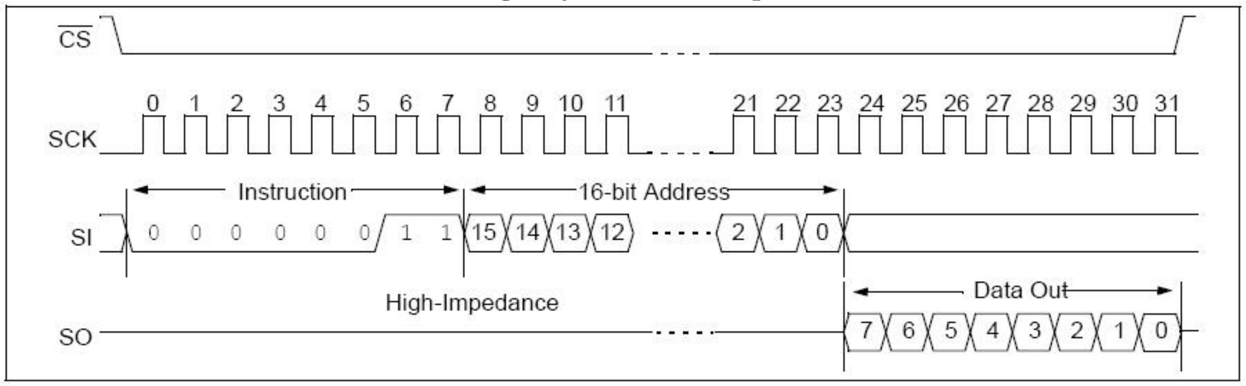
SPI Command Definitions

Name	Byte Value	Description
READ_DATA	0x03	Read Data. The command byte is followed on the MOSI signal by a 16-bit byte starting address, after which the FPGA should return on the MISO signal the contents of configuration memory, automatically incrementing the read byte address until Slave Select is deasserted.
WRITE_DATA	0x02	Write Data. The command byte is followed on the MOSI signal by a 16-bit byte starting address, followed thereafter by write data bytes. The FPGA should automatically increment the byte address for each successive write data byte until Slave Select is deasserted.
READ_STATUS	0x05	Read Status. After transmission of the Read Status command, the FPGA should immediately return the value of the Status Register on the next 8 SCK clock cycles, after which the MMC SPI Master will deassert the

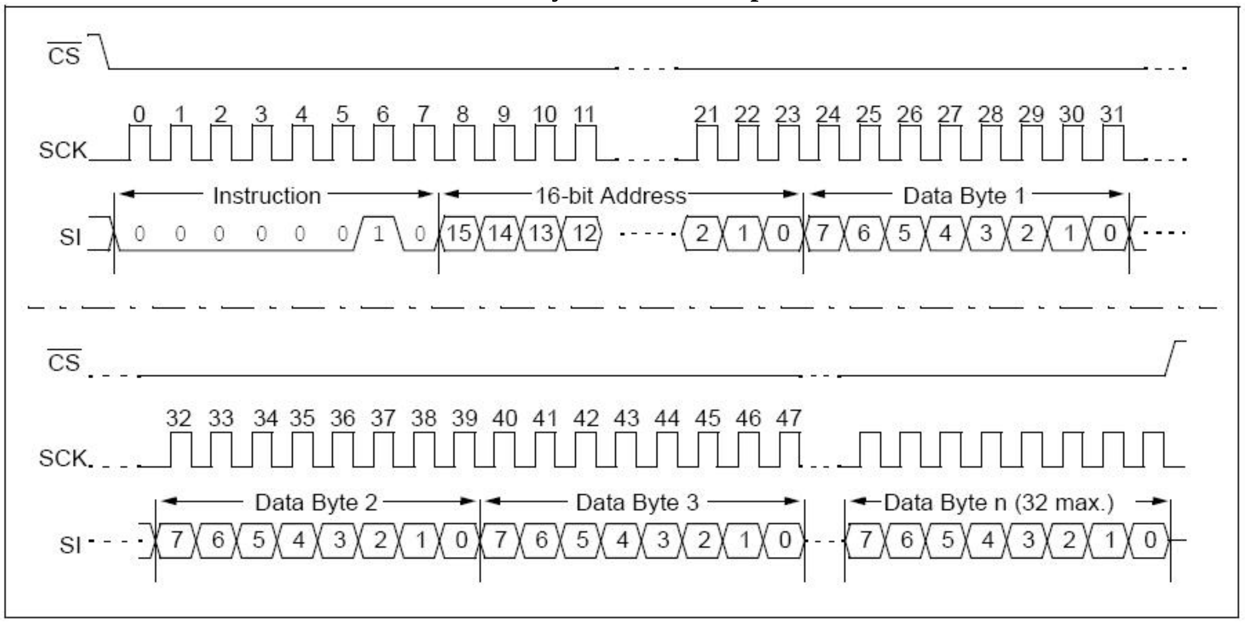
		Slave Select Line
WRITE_CTL	0x07	Write Control. The command byte is followed by the Control Register data byte, and the Slave Select signal is deasserted. Deasserting the Slave Select signal should cause the command to take effect.

SPI Interface Command Sequences. The following diagrams, adapted from the EEPROM data sheet, show examples of the clock and data line activity associated with some of the commands.

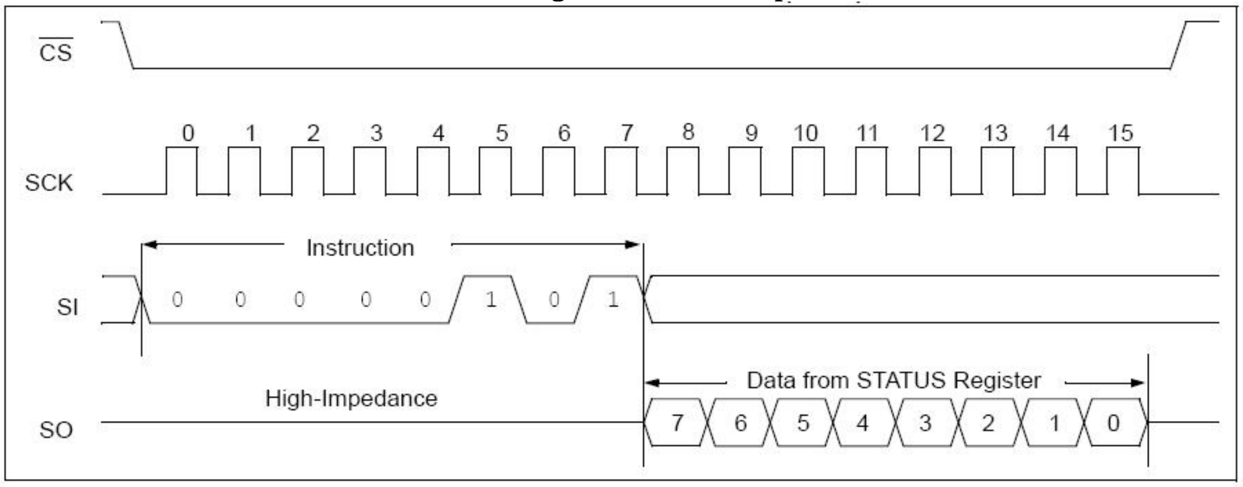
Single Byte Read Example



Multi-byte Write Example



Status Register Read Example



FPGA Configuration from MMC EEPROM Memory. As part of keeping the state of the IPMI Configuration sensor up to date, the MMC automatically polls the state of the FPGA SPI port, whenever back end power is enabled. If the MMC detects that *all of the following conditions apply,* it will transfer a byte sequence from its EEPROM memory to a specified address within the FPGA SPI 16-bit byte address space.

Conditions checked at MMC Startup (application of Management Power to the AMC):

- Valid checksum on Config Header Record
- For each SPI Slave (0-2) for which a record is indicated in the Config Header:
    - Valid header checksum on the Port Config Record
    - Valid transfer length for the Port Config Record (64 bytes max)
    - Valid payload checksum on the Port Config Record

  Note there is no validation check on the destination address in the Port Config Record. This is in keeping with the general philosophy that the FPGA SPI interface is to a large extent simply a transfer mechanism, leaving the details of write address, length and byte format to the specific FPGA application.

Conditions checked during post-power-up polling of the FPGA SPI interface, for each SPI Port (0-2):

- Auto-configuration via MMC enabled in the Payload Manager nonvolatile settings
- Valid SPI port detected on #FSIO_SSx in response to strobing of #FSIO_SCANSLV.
- FPGA SPI Status Register read returns REQCFG set, and CFGRDY clear.
- A configuration record for this port was validated at MMC startup.

In the Config Header Record, a bit may be set which tells the MMC to replace (after checksum validation), the first byte of the configuration data stream with a byte containing the AMC slot ID. In this way, the MMC may provide a configuration record that largely consists of static values from EEPROM, but also contains a dynamic slot address.

Assuming all of the conditions are met, the FPGA transfers the byte string (up to 64 bytes) to the FPGA SPI port, and follows this up with a Control Register write operation, setting the CFGRDY bit. It does this separately for each configured and identified FPGA SPI slave. Thereafter it will perform no more transactions on the FPGA SPI port except for Status Register reads, or unless it detects a recurrence of the condition where REQCFG is set and CFGRDY is not set on a port. This can happen if back end power is cycled to the card, for example as a result of momentarily opening the eject handle and then reclosing it without extracting the module from the crate. In this situation, the MMC responds to subsequent power-ups in the identical manner as it does to the first power-up.

The FPGA itself can also trigger a retransmission of the configuration record by internally setting the Status Register to the condition where REQCFG is set and CFGRDY is cleared. Within 2 seconds (the nominal auto-config poll interval), the MMC will respond and re-execute the auto-config sequence.

FPGA Configuration via LAN-based IPMI Agent. The alternative method for delivering a configuration record to the FPGAs, and possibly the preferred method for operational use in the experiment, is to store it on the supervisory computer, and deliver it via the LAN and crate IPMB. To support this mode of operation, a custom IPMI sensor is defined which makes it possible for a PC application to determine which SPI ports have valid FPGA slaves on them, and what the states are of the REQCFG and CFGRDY flags.

In this mode of operation, a set of custom IPMI commands in the Wisconsin MMC allow the PC application to deliver the configuration payload. From the perspective of the FPGA, there is no functional difference between the two modes of configuration.