EVLA Memo No. 39

Wayne Koski

April 2002

Application Note Using the Serial Peripheral Interface (SPI) For EVLA

SPI is a form of synchronous serial transmission. The basic aspect of SPI is that each SPI device has a serial input, a serial output, and serial clock. One SPI device will be considered to be the master and all additional SPI devices will be considered slaves. The master is simply the device that provides the serial clock and controls the data transfer.

The simplest form of SPI communications is one master and one slave. In this form, only the three lines mentioned above need to be connected in the proper fashion. If more then one slave exists, then an additional line usually called the slave select line (SS) is used to select a particular slave. The SS line acts as chip select lines, as well as a slave output enable line.

For the EVLA, it is planned that the MIB processor will always act as the master SPI device. All SPI devices on the User Device shall be slaves. Also, it is expected that the MIB will have to support multiple SPI devices, so the SPI slave components must have the SS line function. Figure 1 shows the basic connection need to connect the MIB SPI Master to multiple SPI slaves. The line designated Master-Output-Slave-Input (MOSI) simply states that the Master-Output connects to all Slave-Inputs. The same is true for the Master-Input-Slave-Output (MISO) in that the Master-Input must connect to all the Slave-Outputs. The SPI Clock (SCLK) is connected together for all the SPI devices. As stated before, the MIB is the source for the SPI Clock. The active low Slave Select (SS*) lines must be generated one per slave device. It can be generated directly from the MIB or via digital logic controlled by the MIB. The first slave shows the method to daisy chain multiple slaves together, where the SS* line on its low to high transition acts as a data valid signal for each device. This methodology has the problem in that when one needs to update one slave in the chain, the MIB must remember the other slave values. Especially after a power failure, it may in fact be impossible to load the current values from the MIB, so the daisy chain method would best be avoided.

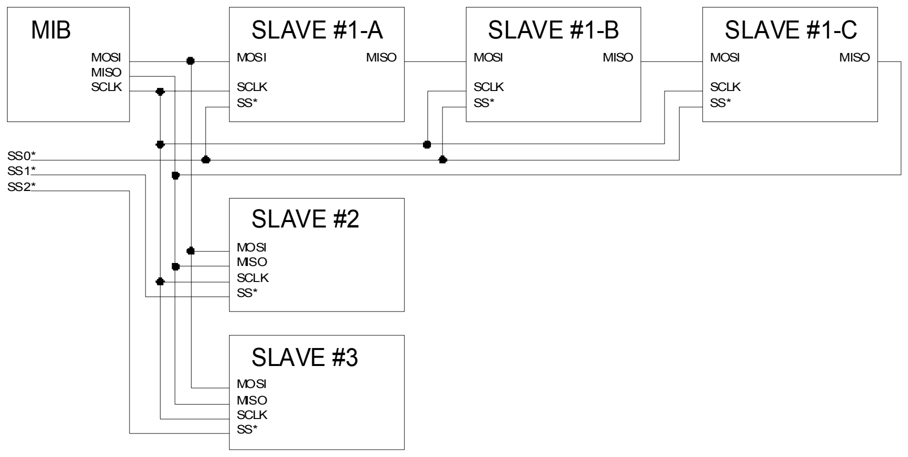

Figure 1: SPI Connections – MIB Master to Slaves

The actual communication using SPI is straightforward. The master and slave simply form a ring buffer, such that when the master transmits its buffer to the slave, the slave transmits its buffer to the master. One transition of the clock shifts the data out of each buffer. The next transition of the clock latches the data into each buffer. Figure 2 shows the timing for a byte transfer. In this transfer, the SS* is driven low to select the specific slave. Next the master clocks the data transfer. In this case, the falling edge of the clock causes the shift registers to place the next data bit onto MISO and MOSI. On the rising edge, this bit will be sampled and loaded into each buffer. When the transfer ends, the SS* will return high, thus deactivating the slave. The master has complete control over the clock's idle state, the transfer rate, which clock edge that shifts/latches the data, MSB/LSB first, and how many bits are to be transferred. Having that level of control calls for standardization however, each slave may require different setups. Therefore the M&C will standardize on the following:

● Clock Idle	= High
● Clock Transition	= Low to High Transition Latches Data
● Clock Speed	= 6 MHz
● Transfer Size	= Byte (8 bits)
● MSB/LSB First	= MSB First

Designers should select SPI Devices with the above in mind, in order to allow for simplified software management. Usage of logic gates that changes any of the above characteristics can easily implement different SPI devices. If an unusual SPI device is selected which doesn't fit the above, and gating cannot provide for its operation, then it would have to be coded as a special case. This will delay development time for devices.

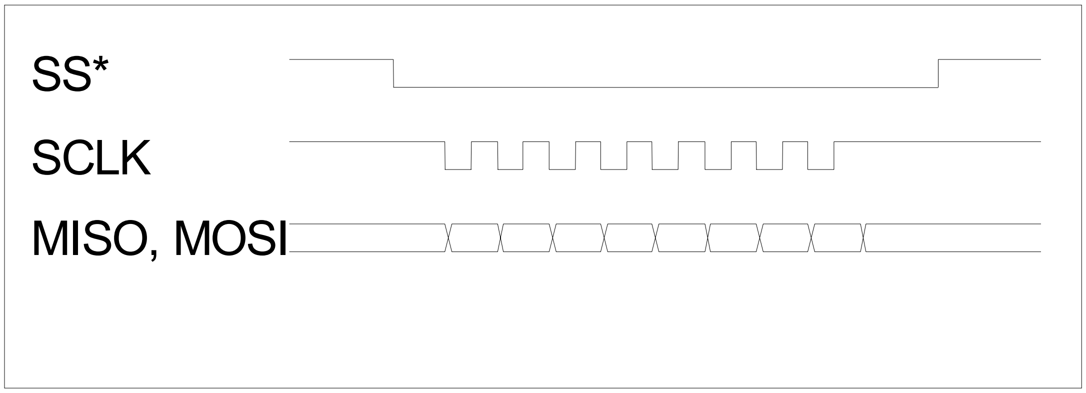

Figure 2: Basic SPI Transfer Timing

The next aspect is how to generate the SS* lines for multiple slaves. The TC11IB device can supply up to ninety-six I/O lines for this purpose. However, many of these lines are dedicated to other purposes so external generation of these SS* is going to be necessary. The basic idea for this, is to provide an address value to the hardware, which then decodes the value into a single SS* line. This address value can be passed to hardware via the SPI bus or by writing it out to a defined memory location using the parallel bus. Figure 3 shows a method of developing the SS* lines via the SPI bus. The circuit design uses Xilinx logic blocks for their FPGA series. It could be easily implemented into other hardware devices as the designer sees fit. For this design the MOSI line feeds a shift register at U3. In order for the shift register to shift in a new address value the MIB must set ADDRSEL* to the active low state. ADDRSEL* also disables U12 so that when a new address is being loaded, it will not cause SS/ lines to glitch during the shift period. The SPICLK is the SPI Clock with the assumption that the low to high is the transition to latch a bit. The PORST* is the power on reset line that clears the shift register and also via U3-Q15 disables U12. This hardware provides for 32K addresses and therefore should provide enough capability for very complex SPI buses. When DATASEL* is driven low, provided U3-Q15 had a high shifted into it, will cause one of the SS* lines to go low. This occurs because DATASEL* enables U12 that feeds the U8 decoder. One output of U8 will enable another decoder such as U4. The single output of the secondary decoder will be inverted and that line then drives a single SS* to the active low state. Additionally the U8 outputs are also inverted and brought out in order to provide control over the external SPI bus by grouping it into banks of sixteen. By simple logic expansion, each U8 output can control another 16 line decoder, so this design can provide a total of 256 SS* lines. In order to produce more lines, one would need to supply an additional layer of decode logic. Finally, the MISO line isn't needed, as setting the address is a write-only function.

For each SPI device that is selected, it will have a protocol in order to utilize it in the design. Designers should attempt to communicate with other designers to find SPI devices that can be used across designs. This will also simplify the software development, in that the number of protocols would be minimal. For the unique case where we might have MIB to microprocessor communications, NRAO would generate a standard protocol for that communication. In fact via ALMA, a protocol was established by NRAO for this purpose. It would be necessary for the EVLA to modify this protocol in order to use it more effectively.

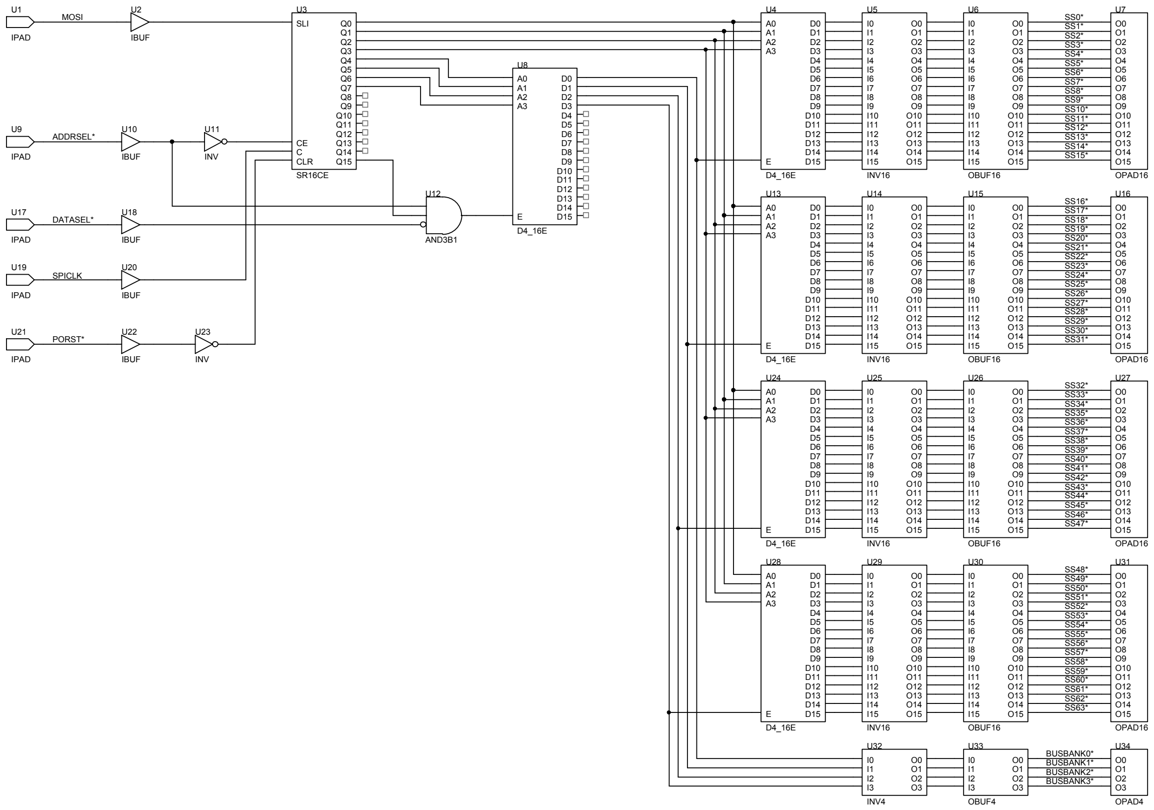

Figure 3: SS* Line Generator Using SPI Bus

In order to minimize RFI generation, the MIB can provide LVDS signals into and out of the MIB for MOSI, MISO, SCLK, ADDRSEL*, and DATASEL* for long distances. Designers should feedback whether or not for short distances, the MIB provides single-ended or LVDS signals.