March 2012 (document revision 4)

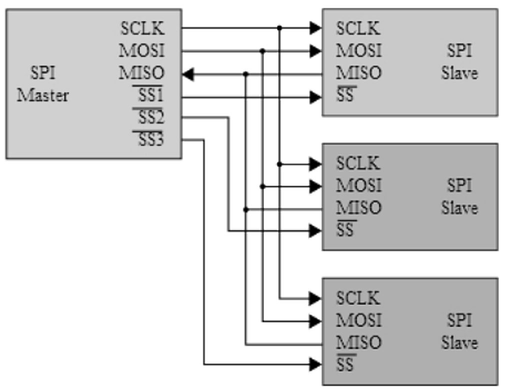
Application Note 0001
mSPI Explained

SPI (Serial Peripheral Interface) is a great serial interface, originally designed by Motorola and at present amongst the most preferred ones by the engineers. Existing in many different variants that have been described in countless number of documents, the original definition will not be presented in details here.
Along with its many advantages, SPI has some drawbacks as well. The most significant one is the need for more host lines when addressing a few slaves. In small and pin-constrained devices that may become an issue so significant, that the designer would eventually resort to another serial interface such as I^{2}C or RS-485, unneccessarily sactifising the benefits of using the SPI.

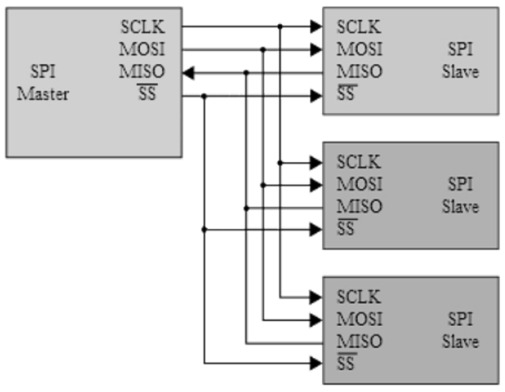
In the Mirtoo module, the available expansion lines are only six, and even further limited down to only four in case the terminal is used for interfacing with the internal interpreter.
Four lines are all it takes to have a SPI communication. But with one slave only. There is the chain formation of SPI slaves as a known way to get around the need for more ~SS lines, but it has its own problems, such as the position-dependent addressing and possible significant delays should the host needs to communicate with one particular slave device only.

mSPI® (mini-SPI) has been designed to allow the Mirtoo module work effectively using its only four available expansion bus lines, and yet be able to address a number of slave devices. In order to do that a very small change in the original interface concept has been made: slaves, who normally keep their MISO (Master In Slave Out) line in output state, will have it kept as input this releasing the line unused, under some conditions have been satisfied. In addition to that all the ~SS (Slave Select, active low) lines are tied together and fed by only one line from the host.

The two figures below clearly show the differences between the standard SPI and mSPI. On the left figure is drawn a system with master and three slaves using the standard SPI. It can be seen that each slave has its own ~SS line.
On the right figure is drawn the same system using mSPI. The three slaves get 'selected' all at the same time by using only a single ~SS line from the master. This is a major difference between SPI and mSPI – while the number of ~SS lines in SPI depends on the number of slave devices to be addressed, it is always reduced to one in mSPI. This hardware simplification leads to the prefix 'mini' in the name as normally mSPI requires less wires than the standard SPI.



'Standard' SPI system

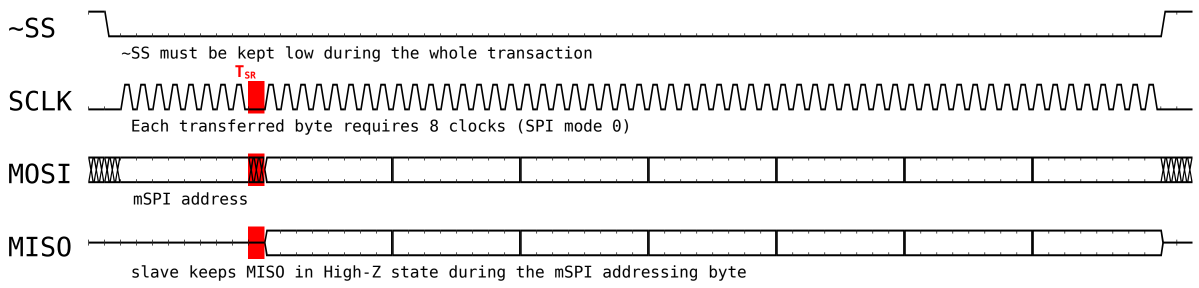


mini-SPI system

The principle of work does not change the way data is transmitted and received through the SPI bus. The only difference is an additional 'address' byte that the master is required to clock in before starting a transmission. Since all slaves will be simultaneously selected and receive the address byte, only the one which address coresponds with the address will change its MISO line to output and start communicating with the master in the normal SPI way. The communication lasts until the master return the ~SS line back in inactive state, which will 'deselect' all slaves thus forcing them back into the initial state where the MISO lines are all inputs. The transaction has been complete and the bus is ready for a new one.

A fixed delay after the last clock of the addressing byte and before the first clock of the following bytes, must be performed by the master in order to allow the addressed slave react and prepare itself for the transaction. The exact value of of this delay is different and depends on the exact slave device. Some slave devices may not need delay at all.

The figure below demostrates a typical mSPI transaction from master's point of view (SPI mode 0, $\mathbf{T_{SR}}$ delay after the address):


~SS
~SS must be kept low during the whole transaction
T_SR
SCLK
Each transferred byte requires 8 clocks (SPI mode 0)
MOSI
mSPI address
MISO
slave keeps MISO in High-Z state during the mSPI addressing byte

It can be seen that during the transmission of the address byte all slaves keep their MISO lines in an inactive state. Immediately after completing the address byte, a slave whose address matches the address byte activates and starts the actual communication which will finish with the master restoring the ~SS line back to high.

Common rules for all devices in an mSPI bus

1. Must use compatible logic levels.
2. Must communicate using the same SPI mode.
3. Must have unique address within the scope of the bus. Mixed length addresses are allowed as long as they don't interfere with each other. For example a device with 6-bit address 0x33 will collide with another using an 8-bit one in the range 0xEC ... 0xEF. It is duty of the mSPI master to know the specific address length requirements of each slave device and act accordingly.
4. All slaves are addressed by using a single ~SS line driven by the master.
5. All slaves keep their MISO lines inactive (input) until been addressed.
6. All slaves have unique within the bus address with length in bits equal to the length of an SPI word.
7. Data transfers start with the transmission of the highest bit in a word and end with the transmission of the lowest one.
8. Every first word transmitted after an active ~SS transition will carry the address of a slave.
9. All slaves are required to receive the address word and only the one whose address matches the address in it will switch its MISO line to an output state and start communication.
10. All communications are terminated by the master restoring the ~SS line back to inactive state. All slaves are required to assert their MISO lines back to inactive (input) state.

The described here principle does not impose any required changes into the existing SPI hardware. It can be done entirely in software and by using four pins only successfuly used in multi-slave systems while still enjoying the simplicity and effectiveness of the SPI bus.