

iCE65 as SPI Port Expander

Overview

This design example illustrates the implementation of a SPI Port Expander using iCE65 FPGAs. Port expanders provide the system host the capability and advantage of expanding its ports thereby reducing pins on the host system. This port expander uses the standard 4 wire SPI bus interface that are available on most host systems to expand into 16 serial input and 16 serial output ports.

Description

SPI™, or Serial Peripheral Interface is a popular 4 wire serial interface that is adapted in most systems. It is a Master - Slave system using 4 lines (3 common and 1 exclusive) as follows:

- MOSI (Master Out, Slave in)
- MISO (Master In, Slave Out)
- SCLK (Serial Clock)
- CS (Chip select or Slave select)

The main features of the SPI interface, unlike many other serial interfaces, is that it allows for a full duplex serial communication. This is possible due to the presence of exclusive lines for data in both directions. The availability of another exclusive line used to select a particular Slave, the CS line, reduces the overhead of address decoding in a multi-Slave environment. These two features combine to add great speeds on the SPI bus (with clock speeds up to 70 MHz).

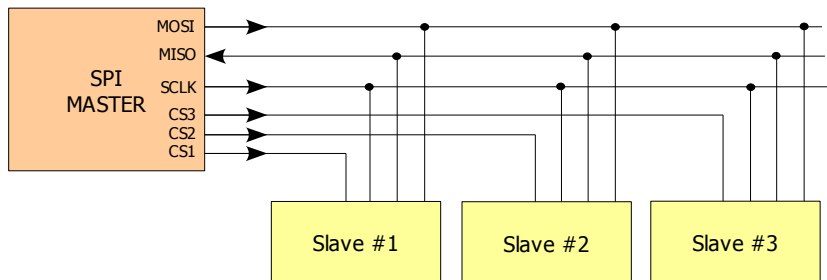


Fig 1: SPI bus system

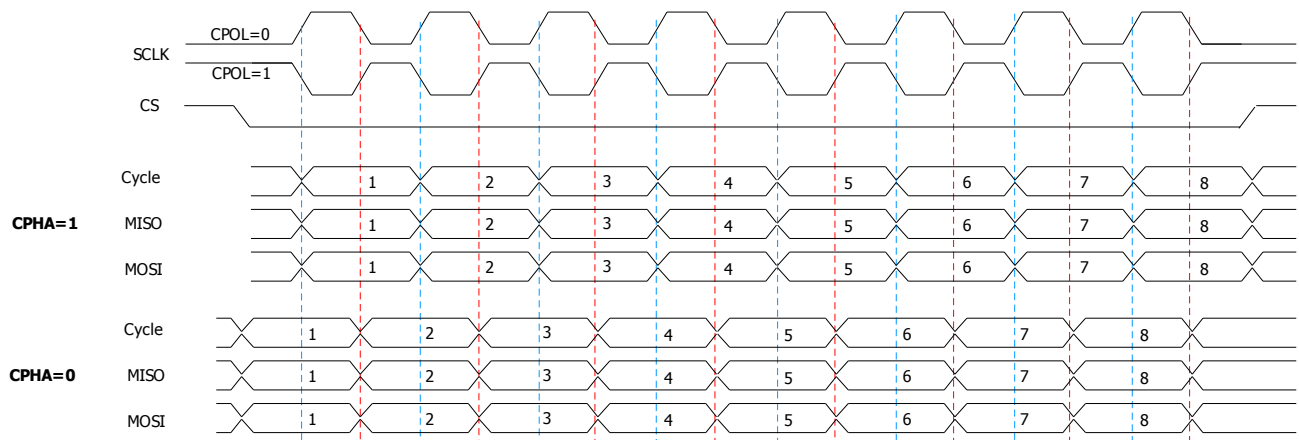


Fig 2: SPI variants and typical timing

This design example of the SPI Port Expander is implemented with configuring the iCE FPGA as a SPI Slave. The Slave connects to the system SPI bus and expands to 16 serial inputs and 16 serial outputs.

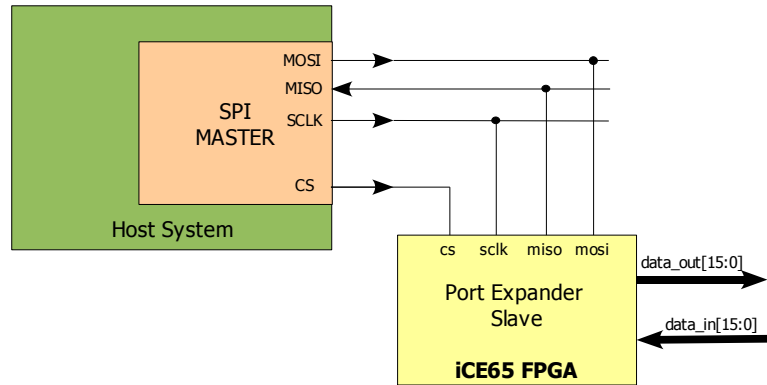


Fig 3: SPI Port Expander

Port	Direction	Description
cs	Input	Active low Slave select
sclk	Input	SPI serial clock
miso	Output	Master input, Slave output line
mosi	Input	Master output, Slave input line
data_out[15:0]	Output	16 expanded output ports
data_in[15:0]	Input	16 expanded input ports

Table 1: Pin Description

Implementation

The SPI port expander is implemented in the iCE65 FPGA by decoding appropriate bits in the SPI's MOSI data frame to obtain the address of the input and output port the Slave has to expand to. Once this address is decoded in the Slave, the serial data that the Master sends to the Slave on its MOSI line is routed to the appropriate output port on the Slave. At the same time the serial input data that the Slave received on a selected input port is routed on to the MISO line. This selection of the input port is also by means of a 4 bit address in the SPI Master's MOSI data frame. This port expander is designed for a 16 bit SPI frame, with the MSB being sent first. The first nibble of the 16 bit MOSI frame signifies the address of the input port to be expanded into. The next nibble signifies the address of the output port the Slave has to expand the MOSI data in to. The balance 8 bits in the MOSI frame contain the data byte to be expanded to the Slave's output port. The first 8 bits on the MISO data frame are insignificant and are ignored. The following 8 bits contain the input data byte that the Slave received on a selected input port.

The data format of the SPI 16 bit word and the significance of its bit positions are indicated in Fig 4 in the following page.

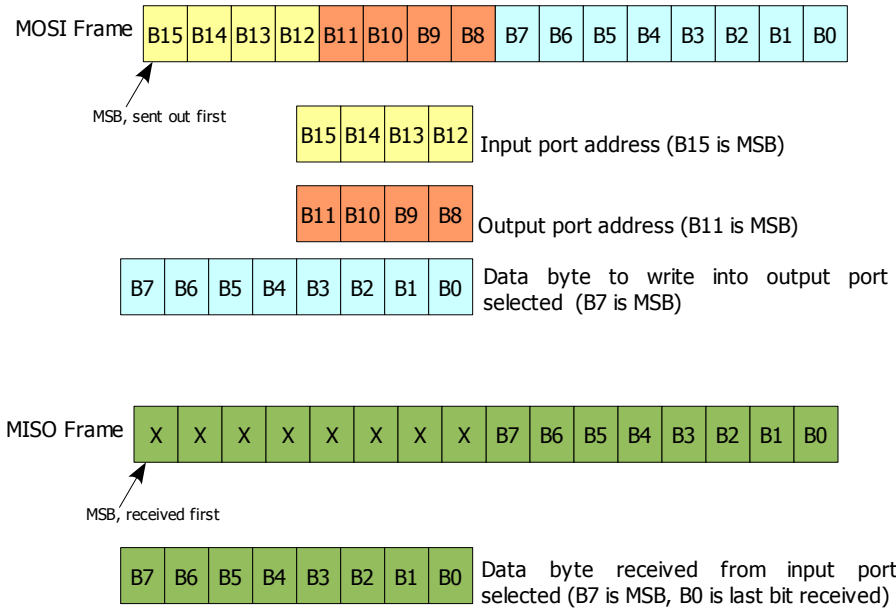


Fig 4: SPI data frame and bit position significances

Table 2 below shows the post P&R resource utilization summary of this design when implemented on an iCE65 FPGA.

Device	Logic Cells	IO Cells
iCE65L04-UCB284	79	36

Table 2: Resource Utilization

Conclusion

This design example demonstrates the implementation of a SPI Slave featuring Port Expansion using iCE FPGAs. The low pin count properties of the 4 wire SPI serial interface, and its high throughput characteristics are well complemented by iCE FPGA's very low power capabilities. This makes iCE FPGAs an obvious choice for implementing SPI Slaves such as port expanders in space and power constrained applications such as battery operated compact/handheld devices.

Disclaimer

SiliconBlue is providing this document, design example, or information "as is." SiliconBlue does not ensure that this implementation or information is void of any claims of infringement. As a reader / implementer, you are responsible for obtaining any rights you may require for your implementation. SiliconBlue also does not warranty the fitness of this design example to be readily implemented in any specific context. It is your sole responsibility to validate this design example and its correctness while implementing it for your requirement.