

Laboraufgabe 5: SPI Slave Interface

Das **SPI-Interface** (Serial Peripheral Interface) ist ein Kommunikationsprotokoll, dass hauptsächlich in der Elektronik und Embedded-Systemen verwendet wird. Es wird genutzt, um Daten zwischen Mikrocontrollern und Peripheriegeräten wie Sensoren, Speicherchips oder Displays schnell und effizient auszutauschen.

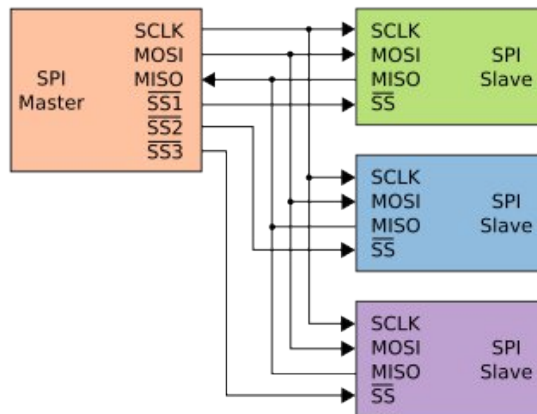


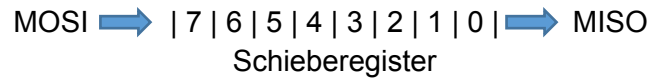
Abbildung 1: SPI Bus in Sterntopologie, Quelle: Wikipedia

Eigenschaften des SPI-Interfaces:

- **Vollduplex-Kommunikation:**
 - Daten können gleichzeitig in beide Richtungen übertragen werden.
- **Synchrone Datenübertragung:**
 - Ein gemeinsames Taktsignal wird verwendet, um die Synchronisation zwischen den Geräten sicherzustellen.
- **Master-Slave-Architektur:**
 - Ein **Master** (z. B. ein Mikrocontroller) steuert die Kommunikation, und ein oder mehrere **Slaves** (z. B. Sensoren) empfangen oder senden Daten.
- **Signalleitungen:** SPI nutzt typischerweise vier Hauptleitungen mit **jeweils einem Bit**:
 - **MOSI** (Master Out Slave In): Datenleitung vom Master zum Slave.
 - **MISO** (Master In Slave Out): Datenleitung vom Slave zum Master.
 - **SCLK** (Serial Clock): Taktsignal, das vom Master erzeugt wird.
 - **SS** (Slave Select): Wählt einen bestimmten Slave aus. Bei mehreren Slaves gibt es meist mehrere SS-Leitungen oder ein Daisy-Chain-System.
- **Flexibilität:**
 - Unterstützt verschiedene Datenraten, Bitordnungen und Taktschemata.
- **Hohe Geschwindigkeit:**
 - Im Vergleich zu anderen seriellen Schnittstellen wie I²C ist SPI oft schneller, da es keine Adressierung oder Bestätigung benötigt.

Das SPI-Interface ist besonders nützlich, wenn hohe Datenraten oder eine einfache Schnittstelle ohne Protokoll-Overhead benötigt werden.

Technisch gesehen ist das SPI Interface auf der Slave Seite ein Schieberegister, welches mit dem SS Signal aktiviert wird. Bei jedem Taktzyklus von SCLK findet eine Schiebeoperation statt, dabei wird der Wert von MOSI in das MSB des Schieberegisters geschrieben und der Rest nach rechts verschoben. Das LSB welches „rechts rausfallen“ würde, wird auf MISO geschrieben. Bei inaktivem SS werden MOSI und MISO Tristate (hochohmig) und verhalten sich dem restlichen Bus gegenüber neutral.



Bei inaktiv werdendem SS wird der Inhalt des Schieberegisters in ein Datenregister verschoben und ein Valid Signal gesetzt.

Der SPI Bus kann in unterschiedlichen Modi betrieben werden

- Clock Polarity (CPOL): Bestimmt den Ruhezustand der Taktleitung (SCK):
 - CPOL = 0: Der Takt ist in Ruhe niedrig (Low).
 - CPOL = 1: Der Takt ist in Ruhe hoch (High).
- Clock Phase (CPHA): Bestimmt, zu welchem Zeitpunkt die Daten gelesen oder geschrieben werden:
 - CPHA = 0: Die Daten werden am ersten (leitenden) Taktflankenübergang gelesen.
 - CPHA = 1: Die Daten werden am zweiten (folgenden) Taktflankenübergang gelesen.

Modus	CPOL	CPHA	Beschreibung
0	0	0	Takt startet bei Low, Daten werden bei steigender Flanke gelesen
1	0	1	Takt startet bei Low, Daten werden bei fallender Flanke gelesen
2	1	0	Takt startet bei High, Daten werden bei fallender Flanke gelesen
3	1	1	Takt startet bei High, Daten werden bei steigender Flanke gelesen

Aufgabenstellung

Ihre Aufgabe ist es, ein SPI – Slave Interface zu implementieren und dessen Funktion zu testen.

- Nutzen Sie Modus 0 (CPOL = 0 / CPHA = 0).
- Nutzen Sie ein Low-Active Slave-Select (SS)
- Nutzen Sie eine Wortbreite von 8 Bit
- MISO muss nicht implementiert werden

Anforderungen an die Implementierung:

- Entwerfen Sie ein vollständig Taktsynchrones Design.
 - Nutzen sie die 50 MHz Taktquelle auf dem jeweiligen Board
- Achten Sie auf sinnvolle Signalinitialisierungen, ein Reset ist nicht vorgeschrieben
 - Das System muss bei Power-On (ohne zusätzlichen Reset) funktionsfähig sein
- Es ist verboten, die Funktion „rising_edge()“, „clk'event“, „wait_until“ auf andere Signale als den Systemtakt anzuwenden
 - Sollten Sie Flankenerkennung bei Logiksignalen benötigen, Entwerfen Sie ein Modul, um steigende/fallende Flanken von Logiksignalen Taktsynchron zu detektieren
- Geben sie den aktuellen Zustand des Schieberegisters auf 8 LEDs Ihrer Wahl aus
- Der empfangene Wert wird bei inaktiv werdendem SS Signal gültig (Ende der Übertragung)
- Geben sie das vollständig empfangene Datenwort auf 7 Segment Anzeigen aus
 - Der Angezeigte Wert darf sich während des Empfangs eines neuen Wertes nicht ändern
- Bei inaktivem SS darf sich bei eingehendem SCLK / MOSI das empfangene Datum im Schieberegister nicht verändern
- Ihr Design muss nur Lesen können, das MISO Signal muss nicht implementiert werden

Das SPI Modul muss folgende Schnittstellen aufweisen

SPI Bus Seite		Systemseite	
MOSI	Input	Data [7..0]	Output
SS	Input	Valid	Output
SCLK	Input		

Machen Sie sich zu folgenden Punkten vor der Implementierung Gedanken:

- Der SPI Bus ist ein getakteter, synchroner Bus: Welche maximale Bus-Taktfrequenz kann theoretisch erreicht werden, wenn der FPGA-interne Abtasttakt 50 MHz beträgt
- Was passiert, wenn SS länger als 8 SCLK-Taktzyklen aktiv bleibt
- Wie viele Dezimalstellen (7-Segment Anzeigen) benötigen Sie für die Darstellung der empfangbaren Daten?
- Was sind zu erwartende Fehlerbilder? Wie können diese vorgebeugt werden?
- Vor der Implementierung ist es empfehlenswert, sich das zu implementierende System zu visualisieren, dies können Blockschaltbilder, Automatengraphen, Zeichnungen, ... sein. (erforderlich ist dies jedoch nicht)

Erzeugung der Signale des SPI Masters

Um die Funktion des Slaves nachweisen zu können, wird ein SPI Master benötigt, emulieren Sie den SPI Master mit Hilfe von vorhandener Board-Hardware.

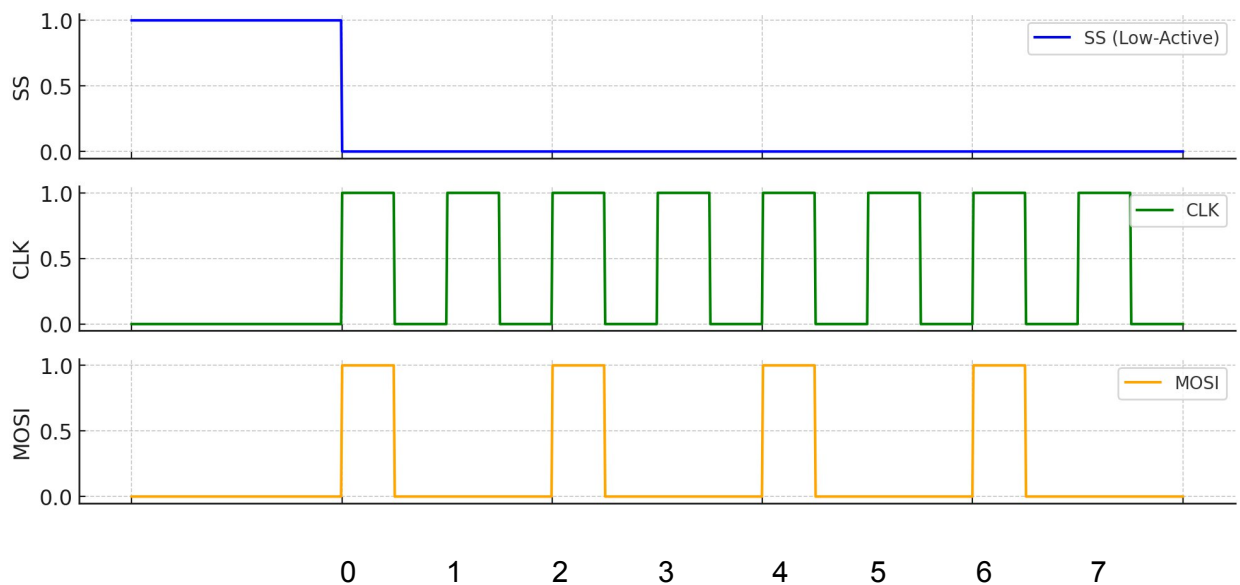
Als Datenquelle (SPI-Master) für Ihren SPI-Slave nutzen Sie folgende Signale

SPI Signal	Board	Entprellung notwendig
MOSI (Gelb)	Beliebiger Schiebeschalter	Nein
SS (Blau)	Beliebiger Schiebeschalter	Ja
SCLK (Grün)	Beliebiger Taster	Ja

ACHTUNG: Achten Sie auf Prellen und Polarität

Beispiel für Erzeugung eines Datenwortes:

SS 1 > 0 / MOSI setzen / SCLK Taster / MOSI setzen / ... / SCLK Taster / SS 0 > 1



Es wird das Datenwort [LSB] „01010101“ [MSB] übertragen.