VHDL – Aufgabe 1

SPI Schnittstelle

Aufgabenbeschreibung

Im ersten Teil des VHDL Praktikums soll eine synthesefähige SPI Schnittstelle (Master) als universell einsetzbare Komponente entwickelt werden. Die Funktion ist durch ausführliches Testen mit Modelsim nachzuweisen. Eine Übertragung des Designs in die Zielhardware ist zu diesem Zeitpunkt noch nicht vorgesehen.

SPI

Bei dem „Serial Peripheral Interface“ handelt es sich um eine serielle Schnittstelle für schnelle Datenverbindungen zwischen integrierten Schaltungen. Die maximale Übertragungsrate beträgt ungefähr 20 Mbps. Die bidirektionale Übertragung erfolgt synchron zu dem vom Master vorgegebenen Takt.
Die Details zur Spezifikation der Schnittstelle sind im Internet verfügbar. Es ist teil der Aufgabenstellung eine geeignete Quelle zu finden. Für einen ersten Überblick empfehle ich folgende Seite:
http://elm-chan.org/docs/spi_e.html

Anforderungen

In jedem Fall müssen sich Datenbreite und Übertragungsgeschwindigkeit des SPI Masters an die spezifischen Anforderungen anpassen lassen.
In der Spezifikation von SPI ist zu erkennen, dass es vier mögliche Modi zur Übertragung der Daten gibt. Als Minimallösung dieser Aufgabe genügt es Mode 0 zu implementieren. Hierbei werden die Daten bei der steigenden Flanke seriell eingelesen und bei der fallenden geschoben.
Die zu verwendende Entity befindet sich auf der folgenden Seite und kann zusätzlich von meiner Website herunter geladen werden.

```
--------------------------------------------------------------------------------------------------------
--  Master für die SPI Schnittstelle.
--  Sobald das Startflag für eine Periode des Basistaktes gesetzt ist, werden die Daten am
--  parallelen Eingang txData eingelesen und anschließend übertragen. Während der Übertragung ist
--  das Busyflag gesetzt.
--  Sobald das Busyflag gelöscht wird, stehen die empfangenen Daten am parallenen Ausgang rxData zur
--  Verfügung.
--
--  Generics:
--      gMode:
--          Definiert den Übertragungsmodus der Schnittstelle.
--
--      gDataWidth:
--          Definiert die Breite der zu übertragenden Daten in Bit.
--
--      gClkDiv:
--          Definiert die Übertragungsgeschwindigkeit abhängig vom Basistakt.
--          clk_data = clk / gClkDiv
--
--
--  Port:
--      rst     : asynchroner Reset, high-aktiv
--      clk     : Basistakt
--      txData  : Zu sendende Daten
--      start   : Startflag, beginnt die Übertragung
--      miso    : SPI master in slave out
--      rxData  : Empfangene Daten
--      busy    : Busyflag
--      mosi    : SPI master out slave in
--      sck     : SPI Takt
--
--  Autor: xxx
--  Datum: xxx
--------------------------------------------------------------------------------------------------------
entity E_SPIMaster is
    generic(
        gMode       : integer range 0 to 3;
        gDataWidth  : positive;
        gClkDiv     : positive
    );

    port (
        rst     : in std_logic;
        clk     : in std_logic;
        txData  : in std_logic_vector(gDataWidth - 1 downto 0);
        start   : in std_logic;
        miso    : in std_logic;
        rxData  : out std_logic_vector(gDataWidth - 1 downto 0);
        busy    : out std_logic;
        mosi    : out std_logic;
        sck     : out std_logic
    );
end E_SPIMaster;
```

Ablauf

Die Aufgabe soll in folgender Reihenfolge bearbeitet werden:

1. **Planungsphase:**
   Macht Euch anhand eines/einiger Blockdiagramme Gedanken zum Aufbau des Designs. Verwendet für die strukturierte Darstellung des Designs das Top-Down-Verfahren.
2. **Implementierungsphase:**
   Nach der Planungsphase implementiert Ihr das Design in VHDL. Hierfür steht in CAE2 die Entwicklungsumgebung Quartus II zur Verfügung. Achtet bitte darauf, dass Ihr möglichst früh anfangt einzelne Teile des Designs mit Modelsim zu testen!

3. **Simulationsphase:**
   Erstellt als Erstes eine Liste mit sinnvollen Testfällen um die Funktion des Designs möglichst vollständig zu prüfen. Es kann übrigens sinnvoll sein diese Liste bereits vor der Implementierungsphase zu erstellen um sicherzustellen, dass der Test nicht an die vorhandenen Fehler angepasst wird und um sich vorab die genaue gewünschte Funktion des Designs klar zu machen.
   Schreibt nun eine oder mehrere Testbench(-es), um das Design mit den erstellten Testfällen zu prüfen. Achtet darauf den Test möglichst übersichtlich zu gestalten. Hierbei hilft die Verwendung von Prozeduren, Funktionen und Schleifen. Für die automatische Auswertung der Tests lassen sich asserts verwenden.
   Das Design wird zunächst funktional und erst dann als post-synthese Simulation mit Timing getestet.
4. **Vorführung:**
   Nach erfolgreichem Testen des Designs erfolgt eine Vorführung beim zuständigen Assistenten. Dieser wird das Design mit einer eigenen Testbench prüfen.
5. **Dokumentation:**
   Die Dokumentation der Aufgabe ist **wichtiger** Bestandteil bei der Vergabe der Punkte für dieses Praktikum. Hierzu gehört neben einer schriftlichen Ausarbeitung in gedruckter Form eine CD mit allen selbst erstellten und für die Inbetriebnahme nötigen Dateien, sowie der Ausarbeitung in digitaler Form.
   In der Ausarbeitung müssen sich mindestens die folgende Punkte befinden:
   1. Planung:
      1. Problemanalyse
      2. Blockschaltbild(er)
      3. Beschreibung der Schnittstellen
   2. Implementierung:
      1. Erklärung wesentlicher Design-Teile
      2. Besonders wichtige Codeabsschnitte
      3. Zustandsdiagramme der Automaten
   3. Simulation:
      1. Beschreibung der Testfälle
      2. Beschreibung der Testbenches
      3. Testergebnisse (Timing-Diagramme, ...)
      4. Auswertung / Interpretation der Testergebnisse

Abgabe

Diese Aufgabe muss beim zuständigen Assistenten bis zum 19.05.09 erfolgreich vorgeführt und die Dokumentation abgegeben sein.
Wer diesen Termin versäumt, bekommt zwar keinen Fehlversuch angerechnet, kann aber das Praktikum im laufenden Semester nicht fortsetzen.