0. Übung	Fachgebiet Architektur eingebetteter Systeme **Hardwarepraktikum (TI)**	AES Embedded Systems Architecture SoSe 21

Ausgabe: 12.04.2021

Handout Abgabe: entfällt
Praxis Abgabe: entfällt

Arbeitsziel

Rekapitulieren Sie die Hardwarebeschreibung mit VHDL und erlernen Sie den Umgang mit der Entwicklungsumgebung *Xilinx Vivado* anhand einfacher synchroner und asynchroner Beispielkomponenten.

Hinweis: Im regulären Praktikumsbetrieb wären die nachfolgenden Beispielaufgaben (so wie auch einige Aufgaben später im Praktikum) auf dem FPGA getestet worden. Dies ist aufgrund der aktuellen Situation leider nicht möglich. Die entsprechenden Aufgabenteile wurden nicht aus dem Blatt entfernt, werden aber grau hervorgehoben, um Ihnen bei Interesse zu ermöglichen, sie dennoch nachzuvollziehen. Grau hervorgehobene Aufgabenteile fließen grundsätzlich nicht in die Bewertung ein.

Arbeitsmaterialien

- Projekttutorial
- Vivado User Guide[1]
- Basys 3 Reference Manual[2]

Aufgabenbeschreibung

Sie erlernen den Umgang mit der Entwicklungsumgebung *Vivado* anhand eines vollständig vorgegebenen Beispiels und frischen anschließend ihre VHDL-Kenntnisse durch einfache Beispielaufgaben auf. Eine Bewertung findet nicht statt.

Aufgabe 1 (Tut) Beispiel: ODER-Gatter

Beginnen Sie mit der im Projekttutorial[1] beschriebenen Beispielaufgabe: Anlegen eines neuen Projekts, Erstellen und Einbinden eines VHDL-Moduls und Simulation mittels einer Testbench-Waveform.

Aufgabe 2 (Tut) Beispiel: Zähler

Erstellen Sie in Vivado ein neues Projekt für den Zähler. Legen Sie eine Datei `counter.vhd` im Quellenverzeichnis an und binden Sie sie in das Projekt ein. Eine Beschreibung der Schnittstelle des Zählers finden Sie in Tabelle 1.

Benennen Sie die Entity mit `counter` und binden Sie folgende Pakete ein, wie es in Listing 1 dargestellt ist.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

Listing 1: Benötigte Pakete für den Zähler

[1]Das Projekttutorial finden Sie als Handout auf der ISIS Seite

EXT_RST	in	Synchrones, highaktives Resetsignal des Zählers. Für EXT_RST = '1' während einer steigenden Flanke an EXT_CLK wird der Zähler auf '0' gesetzt.
EXT_CLK	in	Taktsignal des Zählers. Zu jeder steigenden Flanke des Taktsignals wird der Zähler inkrementiert.
EXT_LED(7:0)	out	Zeigt den aktuellen Wert des Zählers an.

Tabelle 1: Beschreibung der Schnittstelle des Zählers

Ergänzen Sie eine Architecture `fast` und implementieren Sie folgendes Verhalten:

Der Zähler enthält ein synchrones 8-Bit-Register, das mit jeder steigenden Taktflanke von `EXT_CLK` seinen Inhalt inkrementiert. Überläufe müssen nicht abgefangen werden. `EXT_RST` ist ein synchrones Resetsignal, welches das Zählregister wieder auf Null setzt.

Die Realisierung von Registern mit synchronem Resetsignal entnehmen Sie bei Bedarf dem *Vivado Synthesis Guide* [3], Kapitel 4, *HDL Coding Techniques*. Der Port `EXT_LED` entspricht dem aktuellen Inhalt des Zählregisters. Erklären Sie ihre neue Komponente zum Topmodul.

Testen Sie die Funktionsweise ihres Zählers durch eine Testbench. Erstellen Sie die Testbench über den *Add Sources* Dialog als *Simulation-only source*. Achten Sie darauf, als Zielverzeichnis ihr Quellenverzeichnis anzugeben. Definieren Sie eine `entity` ohne Ports mit dem Namen `counter_tw`. Instanziieren Sie ihren Zähler innerhalb dieser und schließen Sie die Ein- und Ausgangssignale an interne Signale an. Definieren Sie die Input-Signale `EXT_CLK` und `EXT_RST` mithilfe des `after` Konstrukts, wie im Tutorial beschrieben. Legen Sie einen 100Mhz Takt an `EXT_CLK` und setzen Sie `EXT_RST` für zwei Takte auf `'1'` und dann wieder auf `'0'`.

Laden Sie ihre Testbench in QuestaSim und simulieren Sie sie für 400 ns. Zeigen Sie dabei die Signale `EXT_CLK`, `EXT_RST` und `EXT_LED` in der Wave an. Überprüfen Sie, dass ihr Design wie gewünscht funktioniert.

Treten keine Fehler auf, so passen Sie das bisherige XDC-File an ihr neues Modul an: die bisherigen *Location Constraints* werden auskommentiert (ein # am Zeilenbeginn) und das XDC-File um die neuen *Location Constraints* ergänzt: `EXT_CLK` wird auf den Takteingang abgebildet, an dem das 100Mhz Taktsignal des Entwicklungsboards anliegt. `EXT_RST` wird auf einen der Pushbuttons ihrer Wahl abgebildet. `EXT_LED` wird auf die 8 LEDs des Boards abgebildet.

Implementieren Sie ihr Design, womit gleichzeitig die Synthesefähigkeit ihres Codes überprüft wird.

Generieren Sie anschließend das neue Bitfile und spielen Sie es in den FPGA ein. Sie sollten sehen können, dass die 8 LEDs unterschiedlich hell leuchten, die Arbeitsweise des Zählers wird jedoch nicht nachzuvollziehen sein, weil er zu schnell zählt.

Aufgabe 3 (Tut) Beispiel: Zähler 2

Wie bereits erwähnt, würden die Beispieldesigns normalerweise auch auf dem FPGA getestet werden. Beim Zähler aus Aufgabe 2 zeigt sich dann, dass die Funktion des Zählers bei der Ausführung auf dem Board nicht wirklich nachvollzogen werden kann, da die Taktfrequenz zu hoch ist und die LEDs, die den Zählerstand anzeigen, zu schnell blinken. Darum soll in dieser Aufgabe die Implementierung des Zählers so angepasst werden, dass die Zählfrequenz reduziert wird.

Kopieren sie Ihre Architecture und nennen Sie die neue Architecture `slow`. Kommentieren Sie `fast` vollständig aus.

Realisieren Sie einen langsam zählenden Zähler mit Hilfe des vorgegebenen VHDL-Moduls **ArmWaitStateGenAsync.vhd**. Kopieren Sie dieses in Ihr Quellenverzeichnis und instantiieren Sie es in der Architektur `slow` des Zählers. Die Schnittstelle des vorgegebenen Moduls finden Sie in Listing 2.

```
entity ArmWaitStateGenAsync is
    port(
        SYS_CLK :       in STD_LOGIC;
        SYS_RST :       in STD_LOGIC;
        WSG_COUNT_INIT: in STD_LOGIC_VECTOR(31 downto 0);
        WSG_START :     in STD_LOGIC;
        WSG_WAIT :      out STD_LOGIC
    );
end ArmWaitStateGenAsync
```

Listing 2: Entity `ArmWaitStateGenAsync`

Dieser Waitstategenerator enthält einen eigenen Zähler. In jedem Takt, in dem `WSG_START = '1'` ist, wird der interne Zähler mit dem an `WSG_COUNT_INIT` anliegenden Wert initialisiert und das Signal `WSG_WAIT` auf `'1'` gesetzt. [2] Sobald `WSG_START` wieder auf `'0'` zurückkehrt, wird der interne Zähler mit jedem Takt dekrementiert. `WSG_WAIT` behält den Wert `'1'`, solange der interne Zähler nicht Null erreicht hat.

Starten Sie nach jedem Inkrementieren des Zählers den Waitstategenerator. Bis das Wartesignal wieder `'0'` ist, soll Ihr Zähler seinen Wert halten.

Simulieren Sie auch diese Version von `counter` in QuestaSim. Verwenden Sie als Initialwert des Zählers (`WSG_COUNT_INIT`) den Wert `X"00000100"` in der Simulation und `X"00100000"` für die Synthese.

Synthetisieren Sie Ihr Design erneut und überprüfen Sie den Synthesebericht.

Literatur

[1] *Vivado User Guide*. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_3/ug893-vivado-ide.pdf. Nov. 2017. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_2/ug893-vivado-ide.pdf.

[2] Xilinx. *Basys 3 Reference Manual*. https://reference.digilentinc.com/_media/reference/programmable-logic/basys-3/basys3_rm.pdf. 2017. URL: https://reference.digilentinc.com/_media/reference/programmable-logic/basys-3/basys3_rm.pdf.

[3] *Xilinx. Vivado Synthesis Guide*. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_3/ug901-vivado-synthesis.pdf. Nov. 2017. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_2/ug901-vivado-synthesis.pdf.

[2] Bei Initialwerten größer null wird der Wert bereits bei der Übernahme dekrementiert, da es sich aus der Sicht des übergeordneten Moduls bereits um den ersten Wartezyklus handelt.