5. Übung

Fachgebiet Architektur eingebetteter Systeme
Hardwarepraktikum (TI)

AES Embedded Systems Architecture
SoSe 17

Ausgabe: 24.05.2017

Theorie Präsentation: 31.05.2017
Praxis Abgabe: 07.06.2017

Arbeitsziel

Implementierung von Komponenten ohne eigenen Code mittels Assistent und praktischer Funktionstest aller Komponenten außerhalb des Prozessorkerns auf dem FPGA.

Arbeitsmaterialien

- Modul `ArmUncoreTop.vhd`
- Modul `ArmChipSelectGenerator.vhd`
- Modul `ArmSwitchDebounce.vhd`
- Modul `ArmSystemController.vhd`
- Testbench `ArmUncoreTop_tb.vhd`
- NGC-Netzliste `ArmMemInterface.ngc`
- NGC-Netzliste `ArmRS232Interface.ngc`

Theoretische Vorbetrachtungen

Die folgenden Fragen sind von der eingeteilten Gruppe in einem kurzen Vortrag zu beantworten. Alle Fragen bezüglich der ARM-Architekur beziehen sich immer auf ISA ARMv4:

- Was wird in der Digitaltechnik unter Taktversetz (clock skew) verstanden und warum kann selbiger Probleme verursache?
- Welche Möglichkeiten bieten FPGAs zu Minimierung des Taktversatzes?
- Was versteht man im Zusammenhang mit FPGAs unter einem IP-Core? Was ist der Unterschied zwischen einem Soft- und einem Hard-IP-Core?
- Ein Teil von Aufgabenblatt 4 war die Realisierung des Transmitters einer asynchron-seriellen Schnittstelle, die lediglich über die beiden Leitungen `TXD` zum Senden und `RXD` zum Empfangen von Daten verfügte. Der vorgegebene Empfänger puffert jeweils nur ein Byte (der Inhalt des Registers `RS232_RCV_REG`), das jeweils nach Empfang der nächsten vollständigen Datenübertragung überschrieben wird. Werden die empfangenen Daten nicht schnell genug durch den Prozessor verarbeitet (z.B. über den Datenbus aus `RS232_RCV_REG` gelesen und in den Arbeitsspeicher kopiert), gehen sie verloren.
  Es gibt zwei verschiedene Ansätze, um den jeweils sendenden Kommunikationspartner einer RS232-Datenübertragung anzuhalten (*flow control*, *Datenflusssteuerung*, *handshake* oder *Synchronisation* genannt) und so Datenverluste zu verhindern. Ein Ansatz ist hardwarebasiert (zusätzliche Steuerleitungen), der andere softwarebasiert. Erklären Sie kurz die Bedeutung der optionalen Steuerleitungen *RTS* und *CTS* einer seriellen Schnittstelle und die Funktionsweise des *XON/XOFF*-Protokolls.

Empfohlene Quellen zur Bearbeitung:

- Mikroprozessortechnik und Rechnerstrukturen, [1, Kapitel 7.5]
- Moderne Prozessorarchitekturen, [2, Kapitel 2.2.3]

Aufgabenbeschreibung

In dieser Aufgabe werden die außerhalb des Prozessorkerns liegenden Komponenten vervollständigt, miteinander verbunden und schließlich praktisch getestet.

Aufgabe 1 (3 Punkte) Erzeugung eines Kerntaktes mit und ohne Phasenverschiebung

Das FPGA auf dem Starter Kit wird mit einem externen 50-MHz-Taktsignal aus einem Taktgeber versorgt. Um diesen Takt unmittelbar für den Prozessor verwenden zu können, müsste die Signallaufzeit über den kritischen Pfad des Prozessors und der Peripherie unter 20 ns liegen. Dies ist nicht der Fall. Darüber hinaus wird für Arbeitsspeicher und Registerspeicher ein gegenüber allen anderen synchronen Komponenten 180° phasenverschobener Takt benötigt.

Für die Synthese neuer Taktsignale sind im FPGA sogenannte DCM (*Digital Clock Manager*) vorgesehen. Sie erzeugen auf Basis eines Eingangstaktes einen neuen Takt mit höherer oder niedrigerer Frequenz und bei Bedarf auch einen Phasenversatz zwischen Eingangs- und Ausgangstakt.

Zur unmittelbaren Verwendung des DCM in VHDL-Modulen müssen spezielle Vorlagen (*Language Templates*) verwendet werden. Sie sind in PlanAhead, neben vielen anderen, im Menü unter *Window* → *Language Templates* zugänglich (oder auch in der Editor-Seitenleiste über das Glühbirne-Icon).

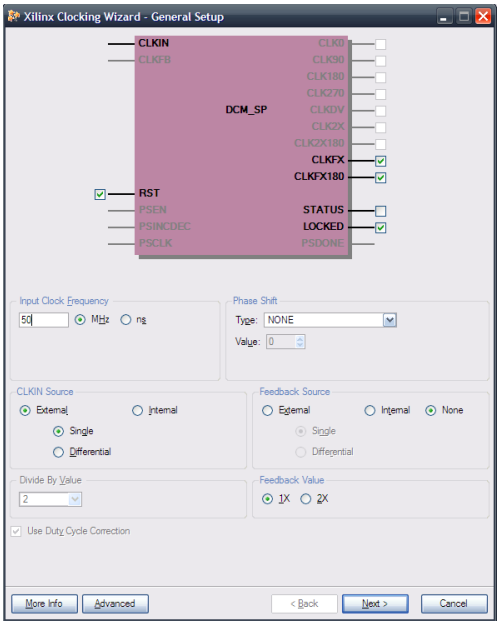
Hier soll aber der Weg über einen Assistenten (*Coregen*) zur geführten Erzeugung von Komponenten beschritten werden. Insbesondere müssen die dabei erzeugten Templates nicht mehr von Hand an die eigenen Bedürfnisse angepasst werden.

Normalerweise kann man über den Button *IP Catalog* direct den *CORE Generator wizard* öffnen, aber der DCM des Spartan-3E boards wird in PlanAhead nicht mehr unterstützt. Deshalb muss Coregen extern gestartet werden.

- Erstellen sie einen neuen Ordner in dem Coregen seine Projektdateien ablegen kann.
- Öffnen Sie Coregen. Geben Sie dazu im Terminal `coregen` ein, oder wählen Sie den entsprechenden Menü-Eintrag.
- Erstellen sie ein neues Projekt in dem gewählten Ordner.
- Wählen Sie als Board Spartan3E, xc3s500e, fg320, -4 und klicken Sie auf OK.
- Auf der linken Seite sehen Sie nun den IP Catalog für das ausgewählte Board. Wählen Sie die Ansicht *View by Function*.
- Wählen Sie aus *FPGA Features and Design* → *Clocking* → *Spartan-3E* → *Single DCM_SP* und doppelklicken Sie (bzw. Rechtsklick und *Customize and Generate*).

Sie werden durch folgende Dialoge geführt:

- Geben Sie zunächst als Name `ArmClkGen` ein. Überprüfen Sie im nächsten Fenster die Angaben für *Output File Type* (VHDL), *Synthesis Tool* (XST) und *Part* (xc3s500e-4fg320).
- Nun öffnet sich der eigentliche *Clocking Wizard*.
- Wählen Sie unter *Feedback Source: None*. Alle CLK-Ausgänge bis auf CLKFX und CLKFX180 sollten jetzt nicht mehr verfügbar sein. Setzen sie jeweils einen Haken an beide Signale.
- Aktivieren Sie *RST*.
- Tragen sie 50 MHz als *Input Clock Frequency* ein.
- Der *Phase Shift Type* ist *NONE*.
- *CLKIN Source* wird auf *External* und *Single* gesetzt.
- *Feedback Value* spielt keine Rolle.
- Setzen Sie den Haken bei *LOCKED*.



Erklärung der getroffenen Einstellungen:
Die DCM können Taktsignale entweder per DLL (*Delayed Locked Loop*) oder per *Digitalem Frequenzsynthesizer* (DFS) erzeugen. Die DLL ist weniger variabel in Ein- und Ausgangsfrequenz und benötigt überdies eine Rückkopplung des erzeugten Taktsignals, die bei Verwendung des DFS (dazu gehören beide FX-Ausgänge) nicht zwingend bereitgestellt werden muss. Durch den Haken bei *LOCKED* erzeugt die DCM ein Kontrollsignal, an dem abgelesen werden kann, ob der neue Takt stabil ist, was einige tausend Perioden des Eingangstaktes dauern kann. Das Locksignal wird später verwendet um System Controller und Prozessor zu starten, sobald der Kerntakt stabil vorliegt. Weitere Details der Spartan-3E DCM finden Sie im Spartan-3 Generation Userguide (ug331), Kapitel 3.

- Wenn alle Einstellungen getätigt sind, klicken Sie auf *Next*.
- Belassen sie die *Clock Buffer Settings* auf *Use Global Buffers for all selected clock outputs*.
- Im nächsten Auswahldialog wird die Zielfrequenz der DFS angegeben. Sie errechnet sich aus der Eingangsfrequenz * Multiplikator / Divisor.
- Sie können die Frequenz direkt eingeben oder M und D vorgeben. Letzteres ist vorteilhaft, weil so nur tatsächlich realisierbare Werte möglich sind. Wählen Sie für M = 2 und D = 10 um eine Zielfrequenz von 10 MHz zu erreichen. Offensichtlich sind mehrere Kombinationen mit dem gleichen Ergebnis möglich, wählen sie M und D immer so klein wie möglich.
- Beachten Sie: der hier erzeugte Systemtakt dient als Zeitbasis für Komponenten, die Signale mit festgelegtem Zeitverhalten erzeugen müssen, insbesondere der seriellen Schnittstelle. Für diese Komponenten sind im Package **ArmConfiguration** die Konstanten ARM_SYS_CLK_FREQ und ARM_SYS_CLK_PERIOD(_INT) vorgegeben. Sie müssen mit dem durch den Taktgenerator erzeugten Systemtakt übereinstimmen und sind ggf. anzupassen.
- Klicken Sie auf *Calculate* um das Ergebnis der Einstellungen anzeigen zu lassen und anschließend auf *Next*.
- Der nächste Dialog fasst ihre Einstellungen zusammen, klicken Sie auf *Finish*.

In den Projektquellen (Fenster *Project IP*) taucht nun die Komponente **ArmClkGen** auf, gekennzeichnet durch einen stilisierten Zauberstab. Ein Doppelklick auf die Komponente öffnet erneut den *Architecture Wizard*, wo sämtliche Einstellungen nachträglich geändert werden können. Im für Coregen angelegten Verzeichnis ist die *Coregen*-Komponente durch die Endung .xaw gekennzeichnet.

XAW-Dateien konnten in früheren Versionen direkt mit ISE verwendet werden, PlanAhead unterstützt jedoch nur noch die neueren XCO-Dateien. Fügen Sie daher die von Coregen generierte VHDL-Datei wie eine normale Quelldatei in PlanAhead über den *Add Sources*-Dialog zum Projekt hinzu. Wenn Sie sich den Inhalt dieser Datei ansehen, können sie nachvollziehen, dass *Coregen* in erster Linie einen Wrapper um das eigentliche DCM_SP Languagetemplate mit den von Ihnen gewünschten Einstellungen erzeugt hat.

Eine Instanz des von Ihnen generierten Taktgebers wird bereits in der Vorgabe **ArmSystemController.vhd** (den Sie spätestens jetzt ins Projekt importieren) verwendet. Natürlich kann der Taktgeber an einer beliebigen Stelle und Hierarchieebene im Design eingebunden werden. Da der System-Controller allerdings auf die Stabilisierung des Kerntakts warten muss, bevor er den Arbeitsspeicher mit einem Programm initialisieren kann, ist es sinnvoll, den Taktgenerator in diesem Modul zu instanziieren.

Allerdings ist der System-Controller für das Laden von Anwendung und einen anschließenden Neustart des Prozessors verantwortlich. Er wartet dazu die Stabilisierung des Kerntaktes ab (LOCK-Signal des DCM), nutzt die ebenfalls mit Kerntakt operierende serielle Schnittstelle zum Laden eines Programms und deaktiviert anschließend das Resetsignal des Prozessors, der damit letztlich gestartet wird. Insofern ist es passend, die Takterzeugung im System-Controller anzusiedeln, wo das LOCK-Signal ausgewertet werden muss.

Aufgabe 2 (4 Punkte) Verhaltenssimulation aller Komponenten außerhalb des Prozessorkerns

Kompilieren Sie alle Vorgaben, insbesondere das Modul **ArmUncoreTop.vhd** und die zugehörige Testbench **ArmUncoreTop_TB.vhd** in ModelSim. **ArmUncoreTop** enthält alle Komponenten unseres kleinen Prozessorsystems, die nicht unmittelbar Teil des Prozessorkerns sind: Der Datenbus ist in Form einiger Signale realisiert und mit dem Speicherinterface **ArmMemInterface** verbunden (der Instruktionsbus wird ohne Prozessorkern nicht benötigt). Ebenfalls an den Datenbus angebunden ist die von Ihnen teilweise implementierte serielle Schnittstelle **ArmRS232Interface**. Hinzu kommt das Modul **ArmSystemController.vhd**.

Funktionsweise des von ArmUncoreTop Sobald der Taktgeber ein stabiles Taktsignal durch *LOCKED* anzeigt, beginnt der System-Controller, den Speicher zu initialisieren. Dazu liest er über den Datenbus permanent das Statusregister der seriellen Schnittstelle aus, liest ggf. ein vorhandenes Datum aus dem Empfangsregister und kopiert es in den Arbeitsspeicher, beginnend an Adresse 0. Dies wiederholt sich, bis die durch `ARM_PROG_MEM_SIZE` aus dem Paket **ArmConfiguration** angezeigte Menge von Bytes eingelesen und in den Speicher kopiert wurde.

Anschließend liest der System-Controller den Speicher aus und schreibt dessen Inhalt zur Kontrolle wieder vollständig in die serielle Schnittstelle. Danach gibt er den Datenbus frei und führt ggf. ein (internes) Reset des Prozessors durch, durch das der Inhalt des Arbeitsspeichers aber gelöscht wird.
Der System-Controller initialisiert den Speicher nach Auftreten eines externen Resetsignals (ein Taster auf dem Board) genau einmal, wobei er das niederwertigste Byte der aktuellen Speicheradresse auf einem Statusausgang anzeigt. Der Statusausgang wird im Topmodul aus dem Prozessorsystem herausgeführt und mit den 8 LEDs auf dem Starter Kit verbunden. Nach der Speicherinitialisierung zeigt der Statusausgang des fertigen Prozessorsystems die Bits (9:2) der aktuellen Instruktionsbusadresse (also die 8 niederwertigen Bits der Instruktions-Wortadresse).

Ablauf des Tests Simulieren Sie mithilfe der Testbench **ArmUncoreTop_TB**, ob das Initialisieren des Speichers wie gewünscht durchgeführt wird. Die Testbench legt ein simuliertes 50-Mhz-Taktsignal an, führt ein Reset durch und simuliert die Übertragung der Speicher-Initialwerte über die serielle Schnittstelle mit Zufallswerten. Anschließend liest die Testbench die vom System-Controller zurückgelieferten Kontrolldaten ein und vergleicht sie mit den gesendeten. Erfolgt keine Antwort oder werden zu wenige Daten zurückgeschickt, tritt ein Timeout ein. Die gesamte Prozedur wird, inkl. des Resets, zweimal durchgeführt.

Die Testnench überprüft lediglich die grundlegende Funktionalität von **ArmUncoreTop**, Spezialtests, beispielsweise bzgl. des Timings der seriellen Schnittstelle, sind nicht enthalten. Sollten Fehler auftreten, suchen Sie deren Ursache selbstständig in ModelSim. Berücksichtigen Sie dabei, dass nicht nur der Verlauf der Schnittstellensignale von **ArmUncoreTop**, sondern der Verlauf aller Signale, Variablen und selbstdefinierter Speicher (Arrays) aller instanziierten Module in ModelSim angezeigt werden kann.

HINWEIS

Setzen Sie für eine erste Simulation `ARM_PROG_MEM_SIZE` auf einen niedrigen Wert, z.B. 128. Auf diese Weise halten sich Simulationszeit und anfallende Datenmenge für die Fehlersuche in Grenzen. Führen Sie nach Beheben aller Fehler aber auf jeden Fall einen erfolgreichen Test mit 16384 Bytes durch und denken Sie daran, `ARM_PROG_MEM_SIZE` für die Synthese auf diesen Wert zurückzusetzen.

Sind in einer der bisherigen Testbenches zu den in **ArmUncoreTop** instanziierten Modulen unkorrigierte Fehler aufgetreten, würden sich diese nun erneut bemerkbar machen. Damit das nicht geschieht und sie nur mit fehlerfrei implementierten Modulen arbeiten, werden Ihnen zukünftig vorkompilierte Simulationsmodelle zur Verfügung gestellt. Diese Modelle bilden die in ModelSim eingebundene Bibliothek **ARM_SIM_LIB**[1].

Falls ihre Implementierung bisher nicht fehlerfrei war, können Sie für dieses Aufgabenblatt die Simulationsmodelle **ARM_SIM_LIB.ArmRS232Interface** oder **ARM_SIM_LIB.ArmMemInterface** nutzen. Um die Modelle zu verwenden, kommentieren Sie die entsprechenden Zeilen zur Bibliothek **ARM_SIM_LIB** am Beginn von **ArmUncoreTop.vhd** ein und die entsprechenden Zeilen zur Bibliothek *work* aus:

```
library ARM_SIM_LIB;
    use ARM_SIM_LIB.ArmRS232Interface;
--  use ARM_SIM_LIB.ArmMemInterface;
library work;
--  use work.ArmRS232Interface;
    use work.ArmMemInterface;
```

Im Beispiel wird das vorgegebene Simulationsmodell der seriellen Schnittstelle, jedoch Ihre Implementierung des Arbeitsspeichers verwendet. Für die Synthese ist es später notwendig, alle USE-Anweisungen zur Simulationsbibliothek und die Zeile

```
    LIBRARY ARM_SIM_LIB;
```

auszukommentieren, da PlanAhead die Simulationsbibliothek nicht verarbeiten kann. Die Verwendung der Simulationsmodelle ist nur dann gestattet, wenn bereits Punkte auf die fehlerhaft implementierten Module abgezogen wurden.

---

[1] Die Bibliothek ist automatisch in ModelSim eingebunden, wenn sie direkt auf dem Server arbeiten. Falls dies bei Ihnen nicht der Fall ist, wählen Sie bei ModelSIM File -> New -> Library. Wählen Sie dann unbedingt "a map to an existing library". Der Name der Bibliothek ist `ARM_SIM_LIB` und sie befindet sich unter /afs/tu-berlin.de/units/Fak_IV/aes/lib/hwpti/ARM_SIM_LIB

Aufgabe 3 (5 Punkte) Synthese und Praxistest

Erklären Sie **ArmUncoreTop** zum Topmodul. Erzeugen Sie ein UCF für die Signale des Topmoduls mit folgender Abbildung von Portsignalen auf Leitungen und Bauteile des Boards:

`EXT_CLK`	Verbunden mit dem 50-MHz-Taktgeber.
`EXT_RST`	Verbunden mit dem „südlichen“ Push-Button.
`EXT_LDP`	Verbunden mit Switch Button 0.
`EXT_RXD`	Verbunden mit der RXD-Leitung der DCE-Schnittstelle (nicht der DTE- Schnittstelle!).
`EXT_TXD`	Verbunden mit der TXD-Leitung der DCE-Schnittstelle (nicht der DTE- Schnittstelle!).
`EXT_LED`	Verbunden mit den acht LEDs des Boards.

Sollten Sie, aufgrund von eigenen, fehlerhaften Implementierungen in älteren Aufgabenblättern, vorgegebene Simultionsmodelle genutzt haben, so müssen die für die Synthese in dem Modul **ArmUncoreTop.vhd** auskommentiert werden.

Erzeugen Sie ein Bitfile Ihres Designs. Lassen Sie den Synthesereport vom Betreuer überprüfen. Er wird einige nicht zu vermeidende Warnings, auch in vorgegebenen Modulen, enthalten. Darunter sollten jedoch keine bzgl. des Vorhandenseins von Latches oder asynchronen Rückkopplungen (combinatorial loops) sein. Konfigurieren Sie anschließend das FPGA und lassen Sie den Betreuer die Implementierung testen. Dabei wird der Speicher über die serielle Schnittstelle von einem Entwicklungsrechner mit Zufallsdaten aus einer Datei initialisiert und die Antwort des System-Controllers in eine weitere Datei geschrieben. Ein Vergleich beider Dateien darf keine Unterschiede zutage fördern.

HINWEIS

Die Synthese von Modulen, die in den bisherigen Tests Fehler gezeigt haben, ist nicht sinnvoll. Daher stellen wir auch hier fehlerfreie Vorgaben zur Verfügung. Verwenden Sie die Vorgaben nur, wenn bereits Punkte auf fehlerhaft implementierte Aufgaben abgezogen wurden. Die Komponenten werden als sogenannte NGC-Netzlisten bereitgestellt. Sie sind das Ergebnis der Synthese von Modulen in *PlanAhead* und noch architekturunabhängig. Die Vorgaben liegen im AFS im Verzeichnis */afs/tu-berlin.de/units/Fak_IV/aes/lib/hwpti/ARM_LIB*. Sie werden nicht in ihr Projektverzeichnis kopiert, sondern von dort aus eingebunden.

Gehen Sie zur Verwendung der Vorgaben folgendermaßen vor:

- Entfernen Sie alle bisher fehlerhaften Quellen aus dem Projekt (rechte Maustaste auf die Komponente → *Remove File from Project*). Die Dateien werden dabei zwar nicht aus dem Quellenverzeichnis gelöscht, ein Fragezeichen vor der Komponente zeigt aber an, das entsprechende Instanzen keiner Quelle mehr zugeordnet sind
- Fügen Sie die NGC-Dateien über den *Add Sources* Dialog hinzu, genau so wie Sie es von VHDL-Dateien gewohnt sind. Anstelle des runden *vh*-Icons erscheint nun ein kleines Icon mit einem *N* neben der Komponente im *Sources*-Fenster.

Mündliche Rücksprache - 4 Punkte

Literatur

[1] FLIK, THOMAS: *Mikroprozessortechnik und Rechnerstrukturen*. Springer Berlin, 7., neu bearb.

Aufl. Auflage, 2004. ISBN 3-540-22270-7.

[2] MENGE, MATTHIAS: *Moderne Prozessorarchitekturen. Prinzipien und ihre Realisierungen*. Springer, Berlin, 1 Auflage, März 2005. ISBN 3540243909.