7. Übung

Fachgebiet Architektur eingebetteter Systeme
Hardwarepraktikum (TI)

AES Embedded Systems Architecture
SoSe 20

Ausgabe: 01.06.2020

Theorie Präsentation: 15./16.06.2020
Praxis Abgabe: 28.06.2020

Arbeitsziel Verständnis für die Funktionsweise eines Barrelshifters. Grundlegende Kenntnisse über die Möglichkeiten, Shift- und Rotationsoperationen auf Operanden in ARM-Instruktionen zu codieren.
Implementierung von generischen Komponenten.

Arbeitsmaterialien

- Barrelshifter im Hamburger Design System (Hades) [1]
- Dokumentation ARM Architecture Reference Manual [2]
- Übersicht über die die ARM-Befehlssatzarchitektur, Abschnitt 1.8.3
- Dokumentation Anforderungen an den Shifter
- Modul `ArmBarrelShifter.vhd`
- Modul `ArmBarrelShifter_HILEVEL.vhd` (vollständige Barrelshifterreferenz)
- Modul `ArmShifter.vhd`
- Testbench `ArmBarrelShifter_tb.vhd`
- Testbench `ArmShifter_tb.vhd`
- Testvektordatei `BARRELSHIFTER_TESTDATA`
- Testvektordatei `SHIFTER_TESTDATA`

Abgabedateien

- Datei `ArmBarrelShifter.vhd`
- Datei `ArmShifter.vhd`
- Datei `ArmBarrelShifter4Bit_tb.vhd`

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:

- Beschreiben Sie den grundlegenden Aufbau und die Funktionsweise eines multiplexerbasierten Barrelshifters.
- Wie verhält sich die Zahl der Stufen eines multiplexerbasierten Barrelshifters zur maximalen Schiebeweite?
- Zeichnen Sie ein Strukturbild eines 4-Bit-Barrelshifter. Der Shifter soll die folgenden Operationen ermöglichen:
  - kein Shift
  - Linksshift
  - Rechtsshift
  - Rechtsrotation

- Wie funktioniert die iterative Instantiierung in VHDL?
- Welche Besonderheit existiert in der ARMv4 beim Schieben von Operanden? (*Hinweis*: 2 Operand, es gibt keine expliziten Schiebeoperation)

Empfohlene Quellen zur Bearbeitung:

- Beschreibung des Barrelshifters bei Hades [1]
- ARM9TDMI Technical Reference Manual [3][Kapitel 1]
- Dokumentation der Anforderungen an den Shifter des HWPR-Prozessors (Handout)
- Übersicht über den ARM-Befehlssatzarchitektur [4, Kapitel 1.8.3]

Aufgabenbeschreibung

Die folgenden Aufgaben dienen der Realisierung des Shifters eines ARM-Prozessors. Der Shifter besteht aus zwei Komponenten: einem multifunktionalen, nicht auf diesen Prozessor beschränkten Barrelshifter und einem Wrapper (einer Abstraktionsschicht) zur Anpassung an Besonderheiten der ARM-Architektur.

Aufgabe 1 (4 Punkte) Implementierung eines allgemeinen 4-Bit-Barrelshifters

Vervollständigen Sie nun das vorgegebene Modul **ArmBarrelShifter.vhd**, dessen Schnittstelle der Tabelle 1 entnommen werden kann. Berücksichtigen Sie, dass Sie ihre Lösung in Aufgabenteil 2 zu einem 32-Bit-Shifter erweitert werden soll. Sie müssen daher eine Lösung anstreben, die durch *Generics* an andere Operandengrößen angepasst werden kann (konkret: 32 statt 4 Bit). Achten Sie darauf einen Barrelshifter zu implementieren und keinen einfachen Shifter. Verwenden Sie nicht die in VHDL vorhandenen Shiftoperatoren! Bei der Synthese entsteht damit eine Lösung, die zwar kleine Signallaufzeiten aufweist, jedoch mehr FPGA-Ressourcen verbraucht. Informationen zur Implementierung, insbesondere zu den Carry-Bits, können auch in der "Übersicht über die ARM-Befehlssatzarchitektur" auf den S. 19-21 gefunden werden.

`OPERAND(3:0)`	`in`	Zu schiebender/rotierender Operand, 4 Bit breit
`AMOUNT(1:0)`	`in`	Die Zahl der Stellen (codiert im Dualcode), um die der `OPERAND` geschoben oder rotiert wird
`MUX_CTRL(1:0)`	`in`	Bestimmt die Art der Operation, die durch die Multiplexer auf den Operanden angewendet wird: `00` kein Shift `01` Linksshift `10` Rechtsshift `11` Rechtsrotation
`ARITH_SHIFT`	`in`	Ist das Signal gesetzt während `MUX_CTRL` einen Rechtsshift anzeigt, so wird ein arithmetischer Rechtsshift durchgeführt, sonst ein logischer Rechtsshift
`C_IN`	`in`	Carry-Eingangssignal, das beim Schieben berücksichtigt wird
`DATA_OUT(3:0)`	`out`	Datenausgang des Shifters
`C_OUT`	`out`	Durch den Shifter erzeugtes Carry-Signal

Tabelle 1: Schnittstelle des 4-Bit-Barrelshifters

Testen Sie ihre Implementierung selbstständig mit einer eigenen Testbench, welche qualitativ dem Signalverlauf aus Abbildung 1 entspricht. Erstellen Sie dazu eine Datei `ArmBarrelShifter4Bit_tb.vhd`, die auch mit abgegeben werden soll. Halten Sie sich aber an die Reihenfolge der vorgegebenen Testfälle. Es ist ihnen freigestellt, weitere Signalkombinationen neben den vorgegebenen zu testen.

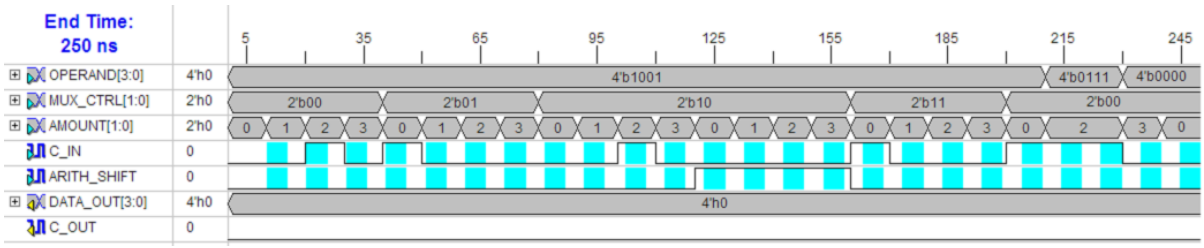

Abbildung 1: Signalverlauf einer Testbench für den 4-Bit Barrelshifter

HINWEIS

Für die Lösung der Aufgabe sind verschiedene Herangehensweisen möglich. Sie könnten beispielsweise ein 4-zu-1-Multiplexermodul anlegen und durch Generate-Statements mehrfach instanziieren. Alternativ bietet sich die Verwendung von Schleifen in Prozessen an. Dabei könnte z.B. die Schleifenvariable einer äußeren Schleife den Stufen des Shifters entsprechen, während in einer inneren Schleife die Elemente des Shifters in dieser Stufe beschrieben werden. Bedenken Sie dabei das unterschiedlichen Verhalten von Zuweisungen an Signale und Variablen in Prozessen! Im Hinblick auf Aufgabenteil 2 sollten Sie nicht das Verhalten jedes einzelnen Multiplexers individuell beschreiben, denn dieser Ansatz scheitert für größere Operanden. Zum Verständnis der Abhängigkeiten zwischen den Multiplexerstufen ist es hilfreich, einige der Operationen im Shifter zu skizzieren.

Schauen Sie sich das Schematic Ihrer Implementierung an und vergleichen Sie es mit dem präsentierten Strukturbild.

Aufgabe 2 (3 Punkte) Implementierung eines allgemeinen 32-Bit-Barrelshifters

Erweitern Sie Ihre Lösung aus Aufgabenteil 1 so, dass 32-Bit-Operanden um 0 bis 31 Stellen geschoben/rotiert werden können. Ändern Sie Ihre Generics entsprechend auf 32-Bit. Achten Sie darauf, ggf. auch die Breite der Ports anzupassen.

Testen Sie Ihre Implementierung mit der Testbench **ArmBarrelShifter_tb**. Die Testbench vergleicht für zahlreiche Testwerte die Ergebnisse Ihres Shifters mit denen der Referenzimplementierung. Die Referenzimplementierung finden Sie in der Datei **ArmBarrelShifter_HILEVEL.vhd**. Dieses Design muss für die Simulation ebenfalls mit *vcom* übersetzt werden. Kopieren Sie beide vorgegeben Testvektordateien in Ihren Ordner für Testvektoren.

Aufgabe 3 (5 Punkte) Implementierung eines ARM-spezifischen Shifters

Lesen Sie vor der Bearbeitung der Aufgabe die Dokumentation der Anforderungen an den Shifter des HWPR-Prozessors, welche auf ISIS 2.0 als Handout zu diesem Aufgabenblatt zur Verfügung steht..
Vervollständigen Sie das vorgegebene Modul **ArmShifter.vhd** mit der in Tablle 2 spezifizierten Schnittstelle. Instanziieren Sie zur Lösung der Aufgabe den Barrelshifter aus Aufgabenteil 2 in **ArmShif-**

SHIFT_OPERAND(31:0)	in	32 Bit Datum, auf das eine von 5 möglichen Operationen angewendet wird.
SHIFT_AMOUNT(7:0)	in	Gibt die Zahl der Stellen an, um die das Datum geschoben/rotiert wird. Beachten Sie, dass Werte zwischen 0 und 255 möglich sind.
SHIFT_TYPE_IN(1:0)	in	Gibt die Operation an, die auf das Datum angewendet wird. Die Codierung entspricht den Konstanten des Typs SHIFT_TYPE aus dem Package **ArmTypes.vhd**. 00 SH_LSL Linksshift 01 SH_LSR log. Rechtsshift 10 SH_ASR arith. Rechtsshift 11 SH_ROR Rechtsrotation
SHIFT_RRX	in	Ist das Steuersignal gesetzt, werden SHIFT_TYPE_IN und SHIFT_AMOUNT ignoriert und stattdessen eine RRX-Operation durchgeführt.
SHIFT_C_IN	in	Übertragseingang des Shifters
SHIFT_RESULT(31:0)	out	Datenausgang des Shifters
SHIFT_C_OUT	out	Übertragsausgang des Shifters

Tabelle 2: Schnittstelle des ARM-spezifischen Shifters

ter.vhd. Alle LSL,LSR,ASR und ROR-Operationen zwischen 0 und 31 Stellen sollen durch den Barrelshifter durchgeführt werden. Die Wirkung von Sonderfällen, also RRX sowie der anderen Operationen mit Schiebeweiten > 31 Bit wird direkt im ArmShifter formuliert. Testen Sie ihre Implementierung mithilfe der Testbench **ArmShifter_TB.vhd**.

HINWEIS

Beachten Sie, dass `SHIFT_TYPE_IN` nicht unmittelbar auf `MUX_CTRL` des Barrelshifters abgebildet werden kann. `MUX_CTRL` enthält u.a. eine Codierung, für die die Multiplexer den Operanden unverändert durchleiten. `SHIFT_TYPE_IN` steht immer für eine Veränderung des Operanden, die allerdings nur dann ausgeführt wird, wenn `SHIFT_AMOUNT` > 0 ist.

Beachten Sie ebenfalls, dass auf dem bereitgestellten Handout ein Flüchtigkeitsfehler vorliegt. Dort wird `RRX` fälschlicherweise als `RXX` bezeichnet.

Mündliche Rücksprache - 4 Punkte

Literatur

[1] Universität Hamburg (TAMS). *Barrelshifter*. `https://tams.informatik.uni-hamburg.de/applets/hades/webdemos/10-gates/60-barrel/shifter8.html`. zuletzt abgerufen am 01.01.2020.

[2] ARM Limited, Hrsg. *ARM Architecture Reference Manual*. 2005.

[3] ARM Limited. *ARM9TDMI - TechnicalReference Manual*. Hrsg. von ARM Limited. 2000. URL: `http://infocenter.arm.com/help/topic/com.arm.doc.ddi0180a/DDI0180.pdf`.

[4] TU Berlin FG Rechnertechnologie. *Hardwarepratikum Technische Informatik Übersicht über die ARM-Befehlssatzarchitektur*. Okt. 2010.