8. Übung

Fachgebiet Architektur eingebetteter Systeme
Hardwarepraktikum (TI)

AES Embedded Systems Architecture
SoSe 20

Ausgabe: 08.06.2020

Theorie Präsentation: 22./23.06.2020
Praxis Abgabe: 05.07.2020

Arbeitsziel

Grundverständnis für Zweck und Aufbau einer Pipeline. Implementierung einer Multiplikationseinheit sowie der ALU des ARM-Prozessors.

Arbeitsmaterialien

- Vivado Synthesis Guide [3]
- ARM Architecture Reference Manual [1]
- Übersicht über die ARM-Befehlssatzarchitektur
- Handout "Detailbeschreibung der Arithmetisch-logischen Instruktionen"
- Modul `ArmMultiplier.vhd`
- Modul `ArmALU.vhd`
- Test Bench `ArmALU_TB.vhd`
- Testvektordatei `ALU_TESTDATA`

Abgabedateien

- Datei `ArmMultiplier.vhd`
- Datei `ArmMultiplier_tb.vhd`
- Datei `ArmALU.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:

- Welche Instruktionen der ARM-ISA v4 (oder v5) benötigen eine Multiplikationseinheit?
- Wie werden die unterschiedlichen Multiplikationsoperationen von einander unterschieden?
- Welche Breite hat das Ergebnis einer Multipkation von zwei 32 Bit breiten Dualzahlen?
- Welche Auswirkungen hat das S-Bit bei arithmetischen Instruktionen auf die Bedingungsbits?
- Wie werden arithmetische Schaltungen (z.B. ein Addierer) in Xilinx FPGAs idealerweise platziert und warum? (Hinweis: zur Platzierung reicht eine Antwort wie, diagonal von links unten nach rechts oben)
- Wie unterscheiden sich, innerhalb von Prozessen, die Zuweisung an Variablen bzw. Signale bezüglich des Verhaltens?
- Wie können die Bedingungsbits für arithmetische Berechnungen erzeugt werden, wenn für die eigentliche Berechnung Bibliotheksfunktionen verwendet werden?

Empfohlene Quellen zur Bearbeitung:

- Übersicht über die ARM-Befehlssatzarchitektur [2]
- ARM Architecture Reference Manual [1]

- Übersicht über den ARM-Befehlssatzarchitektur [2, Kapitel 1.8.7]
- Handout zum Aufgabenblatt

Aufgabenbeschreibung

Die folgenden Aufgaben dienen der Realisierung der Arithmetisch-Logischen-Einheit sowie der Multiplikationseinheit.

Aufgabe 1 (4 Punkte) Multiplikationseinheit

Vervollständigen Sie das vorgegebene Modul **ArmMultiplier**, dessen Schnittstelle in der Tabelle 1 dargestellt ist. Der Multiplizierer führt entsprechend der ARM-Spezifikation lediglich die Multiplikationsoperation aus. Die Einheit erzeugt hierbei nur die unteren 32-Bit des Resultates. Verwenden Sie ein Design, was die im FPGA vorhandenen Komponenten zu Multiplikation verwendet. Testen Sie Ihre Implementierung mit Hilfe einer eigenen Testbench und nennen Sie die Datei `ArmMultiplier_tb.vhd`.

`MUL_OP1(31:0)`	`in`	Erster Operand der Multiplikation
`MUL_OP2(31:0)`	`in`	Zweiter Operand der Multiplikation
`MUL_RES(31:0)`	`out`	32-Bit Ergebnis der Multiplikation

Tabelle 1: Schnittstelle der Multiplikationseinheit

Aufgabe 2 (8 Punkte) Arithmetic Logic Unit

Vervollständigen Sie das vorgegebene Modul **ArmALU.vhd**, dessen Schnittstelle der Tabelle 2 zu entnehmen ist. Die ALU erzeugt die Ergebnisse arithmetisch-logischer Befehle, Adressen für Speicherzugriffsbefehle und wird auch für bestimmte Multiplikationsbefehle benötigt. Sie verfügt über zwei Operandeneingänge, einen Steuereingang, einen Condition-Code-Eingang (nur C-Bit und V-Bit), einen Ergebnisausgang und einen Condition-Code-Ausgang (N,Z,C,V). Die Operanden entstammen dem Registerspeicher des Prozessors oder sind Direktoperanden. Der Condition-Code ist im Statusregister (CPSR) gespeichert, die Signale des Steuereingangs werden in der Prozessorsteuerung erzeugt. Die in der ALU errechneten Ergebnisse werden in den Registerspeicher zurückgeschrieben oder dienen z.B. als Adresse für Speicherzugriffe. Der neu erzeugte Condition-Code wird ggf. in das Statusregister geschrieben. Einer der Eingangsoperanden hat vor der ALU einen Shifter oder einen Multiplizierer durchlaufen, dies spielt für die ALU-Implementierung keine Rolle.

Lesen Sie vor der Lösung der Aufgabe die Abschnitte 1.4 und 1.8.6 in der Übersicht der ARM-Befehlssatzarchitektur sowie die Detailbeschreibung der arithmetisch-logischen Instruktionen. Realisieren Sie die ALU so, dass permanent die Eingangsoperanden zum Ergebnis verknüpft werden und der vollständige neue Condition-Code produziert wird. Die ALU ist ein reines Schaltnetz, es dürfen bei der Synthese keine Flipflops oder Latches entstehen.

Testen Sie ihre Implementierung mit der Testbench **ArmALU_tb**. Kopieren Sie dazu alle vorgegebenen Packages in ihren Quellenordner und fügen Sie sie dem PlanAhead-Projekt durch *Add Sources* hinzu. Legen Sie in ihrem Projektverzeichnis einen Ordner für Testvektordateien an (z.B. **ATV**). Tragen Sie den vollständigen Pfad zum Testvektorordner im Package **ArmFilePaths** ein. Kopieren Sie

ALU_OP1(31:0)	in	Entspricht dem ersten Operanden eines arithmetischen oder logischen Befehls oder auch anderer Instruktionen.
ALU_OP2(31:0)	in	Entspricht dem zweiten Operanden eines arithmetischen oder logischen Befehls und hat ggf. bereits Shifter oder Multiplikationseinheit durchlaufen
ALU_CTRL(3:0)	in	Der 4-Bit-Vektor ALU_CTRL entspricht direkt dem in arithmetisch-logischen Befehlen angegeben Opcode. Die möglichen Bitmuster des Typs und ihre Bedeutung entnehmen Sie den Konstanten des Typs OPCODE_DATA im Package **ArmTypes**.
ALU_CC_IN(1:0)	in	Entspricht dem aktuell gültigen Condition-Code. ALU_CC_IN(1) entspricht dem Carry-Bit. Es hat bereits den Shifter durchlaufen und ist dort ggf. verändert worden. ALU_CC_IN(0) entspricht dem unveränderten Overflow-Bit. N- und Z-Bit werden von der ALU immer neu erzeugt und treten daher am CC-Eingang nicht auf.
ALU_RES(31:0)	out	Ist das Ergebnis einer arithmetischen oder logischen Operation. Das Ergebnis wird **immer** durch die ALU produziert, ggf. aber anschließend im Prozessor nicht weiterverwendet.
ALU_CC_OUT(3:0)	out	Die ALU produziert immer einen vollständigen Condition-Code, unabhängig davon, ob dieser anschließend im Statusregister gespeichert wird. ALU_CC_OUT(3:0) = [N|Z|C|V]

Tabelle 2: Schnittstelle der ALU

die vorgegebene Testvektordatei **ALU_TESTDATA** in den Ordner für Testvektoren. Sie können beliebige weitere Testvektoren hinzufügen, dürfen die bestehenden aber nicht löschen oder verändern.

Mündliche Rücksprache - 4 Punkte

Literatur

[1] ARM Limited. *ARM Architecture Reference Manual*. Hrsg. von ARM Limited. 2005. URL: https://static.docs.arm.com/ddi0100/i/DDI%2001001.pdf.

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

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