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

Ausgabe: 12.04.2021

Handout Abgabe: entfällt
Praxis Abgabe: 09.05.2021

Arbeitsziel

Kontaktaufnahme mit der ARM-Architektur durch Implementierung einer Datenreplikationseinheit, sowie des Instruktionsadressregisters. Es handelt sich um einfache VHDL Implementierungen.

Arbeitsmaterialien

- Vivado Synthesis Guide [5]
- ARM Architecture Reference Manual [1]
- **Übersicht über die ARM-Befehlssatzarchitektur**
- Modul `ArmDataReplication.vhd`
- Test Bench `ArmDataReplication_TB.vhd`
- Modul `ArmRamBuffer.vhd`
- Modul `ArmInstructionAddressRegister.vhd`
- Test Bench `ArmInstructionAddressRegister_tb.vhd`
- Package `ArmConfiguration.vhd`
- Package `ArmTypes.vhd`
- Package `TB_Tools.vhd`
- Package `ArmFilePaths.vhd`

Abgabedateien

- Datei `ArmDataReplication.vhd`
- Datei `ArmInstructionAddressRegister.vhd`

Theoretische Vorbetrachtungen

Die folgenden Fragen würden normalerweise von einer Gruppe in einem Handout beantwortet werden. Das Handout entfällt für Blatt 1. Da die Antworten wichtige Informationen zu ARM-Architekturen bilden und sind für das HWPTI äußerst relevant sind, empfehlen wir Ihnen als Vorbereitung die Abschnitte **1.2, 1.3, 1.4, 1.5** aus dem Übersicht über die ARM-Befehlsarchitektur zu lesen. Alle Fragen bezüglich der ARM-Architekur beziehen sich immer auf ISA ARMv4:

- Was versteht man unter „general-purpose“ Registern innerhalb eines Prozessors?
- Wieviele „general-purpose“ Prozessorregister besitzt die ARM-Architektur?
- Wie breit sind die internen „general-purpose“ Prozessorregister der ARM-Architektur?
- Welche Datentypen (bezogen auf die Operandenbreite) können durch die folgenden Befehlsgruppen von ARM-CPUs verarbeitet werden?
  - Lade- und Speicherbefehle
  - Arithmetische Befehle
- Wie breit sind ARM-Instruktionen und wie werden sie im Speicher ausgerichtet?

- Welche Besonderheit existiert bezüglich der unteren zwei Bits des PC Registers?

Empfohlene Quellen zur Bearbeitung:

- ARM-Befehlssatzarchitektur [4]
- Rechnerorganisation und -entwurf [3, S. 281 ff]
- Moderne Prozessorarchitekturen [2, S. 48 ff]
- ARM Architecture Reference Manual [1]

Aufgabenbeschreibung

Die folgenden Aufgaben dienen der Realisierung der ersten prozessorrelevanten Komponenten. Erstellen Sie dazu ein neues Vivado-Projekt (mit Namen **ARM**) für die Implementierung des ARM-Prozessors.

Aufgabe 1 (5 Punkte) Datenreplikationseinheit

Arm-Prozessoren können Bytes, Halbworte und Worte aus einem der Prozessorregister in den angeschlossenen Speicher schreiben (siehe dazu: [4, Abschnitte 1.2, 1.3 und 1.8.8]). Ein Prozessorregister besteht aus zwei Halbwörtern bzw. vier Bytes. Wird ein Prozessorregister nur mit einem Halbwort gefüllt ergeben sich dafür 2 mögliche Positionen. Logischerweise wird die Position verwendet, welche auch beim Laden das Halbwortes aus dem Speicher genutzt wird (siehe Theoretische Vorbetrachtungen).

Der HWPR-Prozessor ist durch einen 32 Bit breiten Datenbus mit dem Speicher verbunden. Ein Byte-Schreibzugriff kann sich auf eine der vier *ausgerichteten* Bytepositionen innerhalb der 32 Bit beziehen. Die Datenreplikationseinheit „verteilt“ das entsprechende Byte eines Prozessorregisters auf alle vier ausgerichteten Bytepositionen des Datenbus bzw. das entsprechende Halbwort eines Prozessorregisters auf beide Halbworte des Datenbus. Die Schnittstelle der Einheit kann der Tabelle 1 entnommen werden.

`DRP_INPUT(31:0)`	`in`	32 Bit breiter Dateneingang (Inhalt eines Prozessorregisters)
`DRP_OUTPUT(31:0)`	`out`	32 Bit breiter Datenausgang zum Datenbus
`DRP_DMAS(1:0)`	`in`	Steuervektor für das Verhalten des Moduls. Die möglichen Werte entsprechen den Konstanten `DMAS_BYTE`, `DMAS_HWORD`, `DMAS_WORD`, `DMAS_RESERVED` des Typs `DMAS_TYPE` aus dem Package `ArmTypes.vhd`.

Tabelle 1: Schnittstelle der Datenreplikationseinheit

Ergänzen Sie das vorgegebene Modul **ArmDataReplication.vhd**, sodass folgendes Verhalten realisiert wird:
Für `DRP_DMAS = DMAS_WORD` oder `DMAS_RESERVED` wird der Eingang des Moduls unverändert am Ausgang ausgegeben.
Für `DRP_DMAS = DMAS_HWORD` werden die niederwertigen 16 Bit des Eingangs (`DRP_INPUT(15 downto 0)`) sowohl auf den niederwertigen 16 Bit als auch die hochwertigen 16 Bit des Ausgangs ausgegeben. Anders ausgedrückt: das niederwertige Halbwort des Eingangs wird auf beiden Halbworten des Ausgangs ausgegeben.

Für `DRP_DMAS = DMAS_BYTE` wird das niederwertigste Byte des Eingangs auf allen vier Bytes des Ausgangs ausgegeben.[1]
Testen Sie Ihre Implementierung mit der vorgegebenen Testbench **ArmDataReplication_tb.vhd**.

Aufgabe 2 (7 Punkte) Instruktionsadressregister

Das Instruktionsadressregister-Modul (IAR)[2] bildet den Beginn der Instruktionsausführung im Prozessorkern. Von der am IAR ausgegebenen Instruktionsadresse wird die nächste Instruktion aus dem Arbeitsspeicher gelesen und diese im Kontrollpfad in ein Instruktionsregister geschrieben.
Vervollständigen Sie das Modul gemäß Abbildung 1.

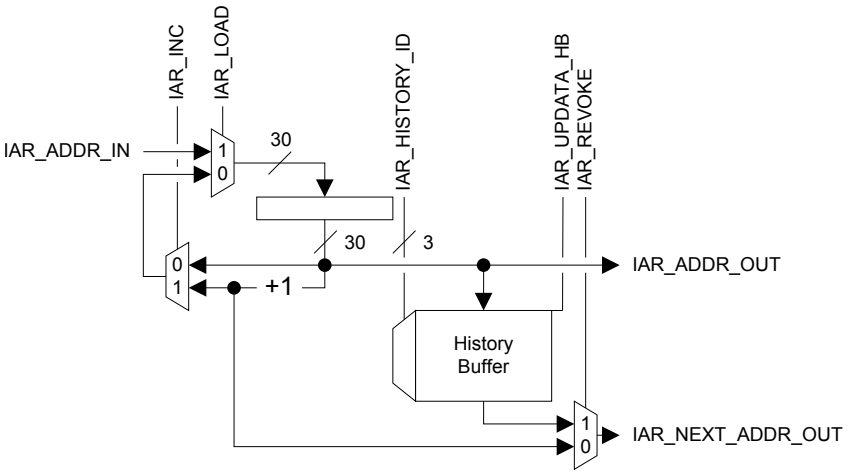

Abbildung 1: Innerer Aufbau des Instruktionsadressregisters (IAR).

Innerhalb des Moduls wird die aktuelle Instruktionsadresse in einem 30-Bit-Register verwaltet. Bekanntlich ist jede Instruktion an einer Wortadresse ausgerichtet, sodass die beiden niederwertigen Adressbits nicht berücksichtigt werden müssen. Die Instruktionsadresse wird mit der steigenden Flanke des (nicht gezeichneten) Taktsignals `IAR_CLK` aktualisiert. In Abhängigkeit von zwei Steuersignalen (`IAR_INC, IAR_LOAD`) wird der aktuelle Inhalt erhalten, die Wortadresse um eins inkrementiert oder ein neuer Inhalt vom Eingang `IAR_ADDR_IN` übernommen. Letzteres ist notwendig, um Sprungbefehle zu unterstützen. Ein **synchrones** Reset-Signal setzt den Inhalt des Registers auf 0[3].
Der Ausgang `IAR_ADDR_OUT` entspricht jederzeit dem Inhalt des Adressregisters und wird zum Treiben der Instruktionsbus-Adressleitungen verwendet. Der Ausgang `IAR_NEXT_ADDR_OUT` entspricht normalerweise der inkrementierten Wortadresse. [4]

Das IAR-Modul erfüllt neben der Adressierung des Speichers und der Aktualisierung des PC noch eine weitere wichtige Funktion: Wenn eine Instruktion eine Ausnahme verursacht, muss ein von der Instruktionsadresse abgeleitetes Datum gesichert werden. Diese sogenannten *Data Aborts* treten aber

[1] `DMAS` = Data Memory Access Size
[2] Das IAR ist nicht mit dem PC identisch, der einen Teil des Registerspeichers bildet, sein Inhalt wird aber vom Instruktionsadressregister abgeleitet.
[3] Null ist der Ausnahmevektor für Reset-Exceptions der meisten (aber nicht aller) ARM-Prozessoren, die Befehlsverarbeitung beginnt nach einem Reset an dieser Adresse.
[4] Dieses Datum wird im Datenpfad um zwei Stellen nach links geschoben (auf 32 Stellen erweitert), und dann in den PC-Port des Registerspeichers geschrieben. Wenn eine Instruktion am Ende der Decode-Stufe den PC ausliest, erhält sie den Wert der eigenen Adresse + 8 (Byte).

erst auf, wenn sich die betreffende Instruktion bereits in einer tiefer liegenden Pipeline-Stufe befindet. Zu diesem Zeitpunkt wurden bereits einige weitere Instruktionen geholt, sodass die Adresse der ausnahmeverursachenden Instruktion nicht mehr aus dem Adressregister abgeleitet werden kann. Deshalb umfasst das IAR einen kleinen Speicher, in dem die Adressen der zuletzt bearbeiteten Instruktionen abgelegt werden. Jeder Instruktion wird von der Prozessorsteuerung eine Identifikationsnummer (ID) zugeordnet. Sie ist 3 Bit breit und wird mit jeder Instruktion inkrementiert. Jede der maximal fünf gleichzeitig in Bearbeitung befindlichen Instruktionen hat daher eine eindeutige ID. Der Puffer (**ArmRamBuffer.vhd**) wird mit der ID (`IAR_HISTORY_ID`) adressiert und speichert die Adresse einer Instruktion, bis diese erfolgreich beendet wurde. Schreibzugriffe finden statt, wenn das Write-Enable-Signal des Puffers '1' ist, es muss mit `IAR_UPDATE_HB` des IAR-Moduls verbunden werden.

Tritt eine Data Abort Exception auf, legt die Prozessorsteuerung die ID der verursachenden Instruktion an `IAR_HISTORY_ID` an und setzt das Steuersignal `IAR_REVOKE`. Dadurch wird die Instruktionsadresse der fehlgeschlagenen Instruktion in den PC geschrieben und kann von dort (mit leichten Modifikationen in der ALU) gesichert werden.

Die (bereits gegebene) Pufferschaltung entspricht einem 1-Port-Speicher (nur eine Adresse, die wahlweise zum Lesen oder Schreiben verwendet wird) ohne Reset[5].

Testen Sie Ihre Implementierung mit Hilfe der vorgegebenen Testbench.

Hinweise zur VHDL Implementierung:
Es soll an dieser Stelle noch einmal deutlich gemacht werden, dass es ein 30-Bit Register gibt und nur diese Komponente synchron beschrieben wird. Sprich, nur für das Register sollte ein Prozess verwendet werden, alle anderen Komponenten sollten asynchron beschrieben werden.

Mündliche Rücksprache - 4 Punkte

Literatur

[1] ARM Limited. *ARM Architecture Reference Manual*. Hrsg. von ARM Limited. 2005. URL: https://silver.arm.com/download/download.tm?pv=1073121.

[2] Matthias Menge. *Moderne Prozessorarchitekturen. Prinzipien und ihre Realisierungen*. 1. Aufl. Springer, Berlin, März 2005. ISBN: 3540243909.

[3] David A. Patterson und John L. Hennessy. *Rechnerorganisation und-entwurf*. Spektrum Akademischer Verlag, Sep. 2005, S. 593. ISBN: 3827415950, 9783827415950.

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

[5] *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.

[5] Durch den Verzicht auf ein Reset kann die Schaltung im FPGA auf LUTs abgebildet werden, die unter dieser Voraussetzung als synchroner Speicher arbeiten können. LUTs mit dieser Funktion werden als *Distributed RAM* bezeichnet. Man findet Sie in den meisten FPGAs und sie stellen erheblich mehr Speicherkapazität zur Verfügung, als die flexibleren Slice-Flipflops.