2. Übung

Fachgebiet Architektur eingebetteter Systeme
Hardwarepraktikum (TI)

AES Embedded Systems Architecture
SoSe 17

Ausgabe: 03.05.2017

Theorie Präsentation: 10.05.2017
Praxis Abgabe: 17.05.2017

Arbeitsziel

Grundlegende Kenntnisse über den ARM-Registersatz. Übersicht über die ARM-Betriebsmodi. Implementierung eines geeigneten Registerspeichers und eines geeigneten Hauptspeichers in VHDL.

Arbeitsmaterialien

- Dokumentation ARM Architecture Reference Manual [1]
- Übersicht über die ARM-Befehlssatzarchitektur, Abschnitte 1.4 und 1.5 [3]
- Package `ArmRegaddressTranslation.vhd`
- Testbench `ArmRegaddressTranslation_tb.vhd`
- Modul `ArmRegfile.vhd`
- Testbench `ArmRegfile_tb.vhd`
- Package `ArmGlobalProbes.vhd`
- Modul `ArmRAMB_4kx8.vhd`
- Testbench `ArmRAMB_4kx8_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:

- In welchen Betriebsmodi kann ein ARM-Prozessor arbeiten?
- Welchen Einfluss hat der Betriebsmodus auf die sichtbaren (virtuellen) Register?
- Geben Sie eine Abbildung der virtuellen 'general purpose' Registeraddressen, in Abhängigkeit des aktuellen Betriebmodus, auf reale/physische Registeradressen an.
- Wie lassen sich in VHDL Speicher modellieren?
- Welche Möglichkeiten zur Realisierung eines Speichers bietet der Xilinx Spartan3E FPGA ?

Empfohlene Quellen zur Bearbeitung:

- ARM Architecture Reference Manual, [1, Kapitel A2]
- Rechnerorganisation und -entwurf, [2, Kapitel 4.2]
- Übersicht über die ARM-Befehlssatzarchitektur, [3, Abschnitte 1.4, 1.5]
- XST User Guide, [4, Chapter 3]

Aufgabenbeschreibung

Es wird eine Funktion für die Übersetzung von 4-Bit-ARM-Registeradressen in 5-Bit-Registeradressen für einen flachen Registeradressraum unter Berücksichtigung des aktuellen Betriebsmodus implementiert. Darauf aufbauend realisieren Sie einen Registerspeicher (Registerfile) für den ARM-Prozessor, der direkt mit den 5-Bit-Registeradressen arbeitet. Anschließen wird die Grundkompente für den Hauptspeicher realisiert. **Verwenden Sie dabei weiter das für Aufgabenblatt 1 erstellte PlanAhead-Projekt.**

Aufgabe 1 (3 Punkte)

Im Package ArmRegaddressTranslation finden sie den Prototyp der Funktion `get_internal_address` mit der Argumentliste aus Tabelle 1. Sie nimmt eine 4-Bit-Arm-Registeradresse (*virtuelle* Registeradresse) und den Betriebsmodus des Prozessors entgegen und erzeugt daraus eine 5-Bit-Registeradresse (*physische* Registeradresse) zum Ansprechen der physisch vorhandenen ARM-Register. Hinzu kommt eine Steuerinformation `USER_BIT`, die bewirkt, dass der aktuelle Betriebsmodus ignoriert und vom User-Modus ausgegangen wird. [1]

`EXT_ADDRESS(3:0)`	Das Signal entspricht den in ARM-Instruktionen verwendeten 4-Bit-Registeradressen.(*virtuelle Registeradresse*)
`THIS_MODE(4:0)`	`THIS_MODE` gibt den aktuellen Betriebsmodus des Prozessors an. Die gültigen Modus-Codes entsprechen den Konstanten des (Sub)Typs `MODE` aus dem Paket ArmTypes.vhd. Sie sollten in Ihrer Implementierung ebenfalls die vordefinierten Konstanten verwenden, um Flüchtigkeitsfehler bei den Modus-Codes zu vermeiden.
`USER_BIT`	Die Instruktionen LDM und STM können optional Zugriffe auf die Register des User-Modus durchführen, auch wenn sich der Prozessor in einem Ausnahmemodus befindet. Wenn das USER_BIT gesetzt ist, wird innerhalb der Adressübersetzung `THIS_MODE` ignoriert und stattdessen der User-Modus für die Adressübersetzung angenommen.
`Rückgabewert`	Der Rückgabewert der Funktion ist eine 5-Bit-Registeradresse in Form eines `std_logic_vector`.(*physische* Registeradresse)

Tabelle 1: Argumentliste der Function `get_internal_address`

Vervollständigen Sie die Funktion zur Übersetzung von virtuellen Registeradressen in physische Registeradressen im *Body* von ArmRegaddressTranslation. Dabei muss jeder gültigen Kombination aus virtueller Registeradresse, Betriebsmodus und User-Bit eine physischen Registeradresse zugeordnet werden. Die Abbildungsfunktion können Sie aus der Theoriepräsentation entnehmen.

Natürlich dürfen unterschiedliche Argumente nur dann zur gleichen physischen Adresse führen, wenn dies im Rahmen der ISA vorgesehen ist. R15 wird durch die Abbildungsfunktion beispielsweise immer auf die selbe physische Adresse abgebildet, unabhängig von Modus und User-Bit. R14 andererseits muss in jedem Modus (außer User und System) auf eine andere physische Adresse abgebildet

[1] Die Funktion wird später in der Decode-Stufe im Kontrollpfad des Prozessors verwendet und löst dort frühzeitig die komplexen Abhängigkeiten zwischen Registeradressen, Betriebsmodus und weiteren Randbedingungen auf, sodass diese Eigenarten der ARM-ISA für die Implementierung weiterer Prozessorkomponenten keine Rolle mehr spielen.

werden, solange das User-Bit nicht gesetzt ist. Die grobe Funktionsweise wird in Abb. 1 ersichtlich.

Es bleibt Ihnen überlassen, welches Verhalten die Übersetzungsfunktion zeigt, wenn Kombinationen nicht genutzt sind oder „ungültige Werte“ angelegt werden. Es sollte aber für jede denkbare Kombination der Argumente ein definierter Rückgabewert (kein ’X’ oder ’U’) erzeugt werden.[2]

Testen Sie ihre Implementierung mithilfe der Testbench **ArmRegaddressTranslation_tb.vhd**.

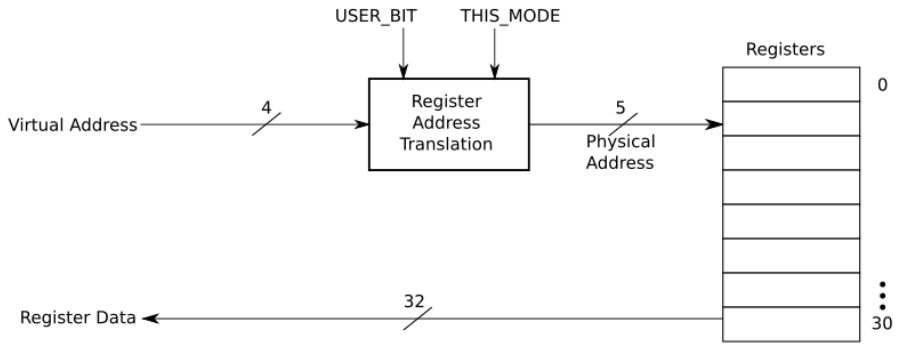

Abbildung 1: Schematische Funktionsweise der Register-Address-Translation (RAT) Einheit.

[2]Dies ist für die spätere Simulation des gesamten Prozessor notwendig. Da sonst u.U. zu einem Array Zugriff an der Stelle ’X’ oder ’U’ kommen kann.

Aufgabe 2 (5 Punkte)

Vervollständigen Sie das Modul **ArmRegfile.vhd**, dessen Schnittstelle der Tabelle 2 entnommen werden kann. Implementieren Sie einen Registerspeicher mit 3 Leseports, 2 allgemeinen Schreibports, und einem PC-Schreibport. Der PC-Schreibport adressiert immer das physische Register, dessen Adresse für R15 verwendet wird. Breite und Anzahl der Register entnehmen Sie Ihren theoretischen Vorarbeiten. Schreibzugriffe erfolgen bei steigender Flanke des Taktsignals, Lesezugriffe sind asynchron. Ein gemeinsames Resetsignal setzt alle Register auf 0.

HINWEIS

Jeder physikalische Adresse die von der Funktion `get_internal_address` erzeugt werden kann soll genau ein Register innerhalb des Registerspeichers zugeordnet sein.

`REF_CLK`	`in`	Gemeinsamer Takt aller Register. Schreibzugriffe erfolgen bei steigender Taktflanke.
`REF_RST`	`in`	Gemeinsames **synchrones**, highaktives Resetsignal aller Register, setzt alle Register auf 0.
`REF_W_PORT_A_ENABLE`	`in`	Nur wenn dieses Signal auf 1 gesetzt ist, haben Schreibzugriffe auf `W_PORT_A` eine Wirkung.
`REF_W_PORT_B_ENABLE`	`in`	Nur wenn dieses Signal auf 1 gesetzt ist, haben Schreibzugriffe auf `W_PORT_B` eine Wirkung.
`REF_W_PORT_PC_ENABLE`	`in`	Nur wenn dieses Signal auf 1 gesetzt ist, haben Schreibzugriffe auf `W_PORT_PC` eine Wirkung.
`REF_W_PORT_A_ADDR(4:0)`	`in`	Registeradresse, an die das Datum von Schreibport A geschrieben werden soll
`REF_W_PORT_B_ADDR(4:0)`	`in`	Registeradresse, an die das Datum von Schreibport B geschrieben werden soll.
`REF_R_PORT_A_ADDR(4:0)`	`in`	Adresse des Registers, dessen Inhalt an Leseport A ausgegeben wird.
`REF_R_PORT_B_ADDR(4:0)`	`in`	Adresse des Registers, dessen Inhalt an Leseport B ausgegeben wird.
`REF_R_PORT_C_ADDR(4:0)`	`in`	Adresse des Registers, dessen Inhalt an Leseport C ausgegeben wird.
`REF_W_PORT_A_DATA(31:0)`	`in`	Schreibport A Dateneingang
`REF_W_PORT_B_DATA(31:0)`	`in`	Schreibport B Dateneingang
`REF_W_PORT_PC_DATA(31:0)`	`in`	Schreibport PC Dateneingang
`REF_R_PORT_A_DATA(31:0)`	`out`	Leseport A Datenausgang
`REF_R_PORT_B_DATA(31:0)`	`out`	Leseport B Datenausgang
`REF_R_PORT_C_DATA(31:0)`	`out`	Leseport C Datenausgang

Tabelle 2: Schnittstelle des Registerspeichers

Berücksichtigen Sie: Theoretisch könnten mehrere gleichzeitige Schreibzugriffe auf die gleiche Registeradresse auftreten. Für diesen Fall sollen die Schreibports unterschiedliche Prioritäten aufweisen: `W_PORT_A` hat die höchste Priorität, `W_PORT_B` die mittlere und `W_PORT_PC` die niedrigste Priorität. Bei Adresskonflikten wird immer der Schreibzugriff auf dem Port mit der höchsten Priorität durchgeführt.

Schreibzugriffe über `REF_W_PORT_PC_DATA` beziehen sich auf das Register des Registerspeichers, welches R15 entsprechen soll. Um welches physische Register es sich dabei handelt, hängt von Ihrer Adressabbildungsfunktion ab. Sie sollten die Funktion deshalb verwenden, um das korrekte Register für den PC-Schreibzugriff auszuwählen.

Das Lesen aus dem Registerspeicher erfolgt asynchron. An `REF_R_PORT_A_DATA` wird immer der aktuelle Inhalt des zur Adresse `REF_R_PORT_A_ADDR` gehörenden Registers ausgegeben. Für die anderen Leseports gilt Entsprechendes.
Im vorgegebenen Modulrumpf von **ArmRegfile.vhd** finden Sie 31 unvollständige Signalzuweisungen innerhalb der speziellen Kommentare:

```
-- synthesis translate_off
-- synthesis translate_on
```

Weisen Sie den 31 Signalen die passenden Register Ihres Registerspeichers zu. [3]

Führen Sie den Funktionstest Ihres Registerspeichers mit der Testbench **ArmRegfile_tb** durch.

[3] Die Kommentare weisen das Synthesewerkzeug *XST* an, diese Signalzuweisungen nicht in Hardware zu übersetzen. Während einer Verhaltenssimulation sind sie jedoch gültig. Die 31 Signale sind im Package **ArmGlobalProbes** deklariert, welches in Ihr Projekt importiert werden muss. Eine Testbench hat gewöhnlich nur Zugriff auf die Schnittstellensignale des zu testenden Moduls, kann aber beispielsweise nicht auf interne Speicher zugreifen. Über den Umweg der globalen Probesignale können die Inhalte des Registerspeichers auch dann noch direkt ausgelesen werden, wenn er in einer höheren Ebene der Designhierarchie instanziiert und diese getestet wird.

Aufgabe 3 (4 Punkte) Bildung eines geeigneten 2-Port-Speichers aus Blockram-Komponenten

Zur Realisierung des Arbeitsspeicher soll in dieser Aufgabe ein 2-Port Speicher mit folgenden Eigeschaften beschrieben werden:

- Größe: 4096x8 Bit
- 2-Ports:
  - Port-A erlaubt Lesezugriff
  - Port-B erlaubt Lese- und Schreibzugriff
- Lesen und Schreiben erfolgt synchron
- Separate Enable Signal für jeden Port
- Extra Write-Enable Signal für Port-B
- Bei gleichzeitigen Lesen und Schreiben, wird auf dem Leseport der alte Wert aus dem Speicher ausgegeben (Read-First Mode)

Zur Lösung der Aufgabe steht Ihnen bereits eine geeignete Vorgabe zur Verfügung (`ArmRAMB_4kx8`). Die Schnittstelle dieser Komponente ist in Tabbelle 3 aufgelistet.

`RAM_CLK`	`in`	RAM-Takt, Zugriffe erfolgen auf der steigenden Flanke.
`ADDRA(11:0)`	`in`	Zugriffsadresse für Port A.
`DOA(7:0)`	`out`	Ausgangsregister von Port A, hält den Wert des letzten Lesezugriffs.
`ENA`	`in`	Mit `ENA = 0` sind alle Zugriffe auf den RAM an Port A wirkungslos. Das Daten-Ausgangsregister ändert sich nicht, auch wenn eine neue Adresse an den Speicher angelegt wird (unabhängig davon, ob ein Lese- oder Schreibzugriff durchgeführt werden soll).
`ADDRB(11:0)`	`in`	Zugriffsadresse für Port B.
`DIB(7:0)`	`in`	Daten, die bei einem Schreibzugriff auf Port B an die Adresse `ADDRB` geschrieben werden.
`DOB(7:0)`	`out`	Ausgangsregister von Port B, hält den Wert des letzten Lesezugriffs
`ENB`	`in`	Mit `ENB = 0` sind alle Zugriffe auf den RAM an Port B wirkungslos. Das Daten-Ausgangsregister ändert sich nicht, auch wenn eine neue Adresse an den Speicher gelegt wird. `ENB = 1` bewirkt das jeweils das zur Adresse korresponiderende Datum ausgegeben wird.
`WEB`	`in`	Mit `ENB = 1` und `WEB = 0` wird ein Lesezugriff an `ADDRB` durchgeführt und das gelesene Datum an `DOB` zur Verfügung gestellt. Mit `ENB = 1` und `WEB = 1` wird neben dem Lesezugriff auch ein Schreibzugriff mit dem Datum an `DIB` an Adresse `ADDRB` durchgeführt.

Tabelle 3: Schnittstelle des RAM Moduls

Achten Sie darauf, dass Sie die im FPGA vorhandenen Ressourcen nicht überschreiten. Realisieren Sie Ihren Speicher so, dass er auf Block RAM abbildet. Beispiele zur Realisierung von RAMs finden Sie u.a. in [4, Chapter 3: XST HDL Coding Techniques]

Testen Sie Ihr Design mit Hilfe der vorgegebenen Testbench (`ArmRAMB_4kx8_tb.vhd`).

Mündliche Rücksprache - 4 Punkte

HINWEIS

Schätzen Sie die benötigten Ressourcen auf dem FPGA ab und vergleichen Sie Ihr Abschätzung mit dem Synthese Report. Vergleichen Sie die Realisierung der beiden Aufgaben.

Literatur

[1] *ARM Architecture Reference Manual*, 2005.

[2] PATTERSON, DAVID A. und JOHN L. HENNESSY: *Rechnerorganisation und-entwurf*. Spektrum Akademischer Verlag, September 2005.

[3] RECHNERTECHNOLOGIE, TU BERLIN FG: *Hardwarepratikum Technische Informatik Übersicht über die ARM-Befehlssatzarchitektur*, Oktober 2010.

[4] XILINX: *XST User Guide*. `http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_3/xst.pdf`, Dezember 2010.