

Ausgabe: 19.04.2021

Handout Abgabe: 28.04.2021

Praxis Abgabe: 16.05.2021

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

31 Register werden auf 16 Register abgebildet

Abgabedateien

- Datei ArmRegaddressTranslation.vhd
- Datei ArmRegfile.vhd
- Datei ArmRAMB_4kx8.vhd

Theoretische Vorbetrachtungen

Die folgenden Fragen sind von der eingeteilten Gruppe in einem Handout zu beantworten, jedoch nicht nur stichpunktartig, sondern als detailliertes Handout. Die Fragen dienen lediglich als Anhaltspunkte die zwingend zu beantworten sind. Gliedern Sie Ihr Handout entsprechend und schreiben Sie auch eine kurze Einleitung zu dem jeweiligen Thema.

Alle Fragen bezüglich der ARM-Architektur beziehen sich immer auf ISA ARMv4:

- In welchen Betriebsmodi kann ein ARM-Prozessor arbeiten? Gehen Sie hier auch (kurz) auf die Eigenschaften der Modi ein.
- Welchen Einfluss hat der Betriebsmodus auf die sichtbaren (virtuellen) Register? Was ist hier der Unterschied zwischen FIQ und IRQ?
- Geben Sie eine Abbildung der virtuellen 'general purpose' Registeradressen, in Abhängigkeit des aktuellen Betriebsmodus, auf reale/physische Registeradressen an. Wieviele Register benötigen Sie insgesamt, wieviele sind stets zeitgleich sichtbar?
- Wie lassen sich in VHDL Speicher modellieren? Gehen Sie hier auch auf den Unterschied zwischen 'memory' und 'registers' ein.
- Welche Möglichkeiten zur Realisierung eines Speichers (memory) bietet der auf unserem Board genutzte Xilinx Artix 7 FPGA? Was ist hier der Unterschied, sowie kurz Vor- und Nachteile?

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]
- Vivado Synthesis Guide, [4, Chapter 4]

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. Anschließend wird die Grundkomponente für den Hauptspeicher realisiert. **Verwenden Sie dabei weiter das für Aufgabenblatt 1 erstellte Vivado-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

<code>EXT_ADDRESS (3:0)</code>	Das Signal entspricht den in ARM-Instruktionen verwendeten 4-Bit-Registeradressen. (<i>virtuelle Registeradresse</i>)
<code>THIS_MODE (4:0)</code>	<code>THIS_MODE</code> gibt den aktuellen Betriebsmodus des Prozessors an. Die gültigen Modus-Codes entsprechen den Konstanten des (Sub)Typs <code>MODE</code> aus dem Paket <code>ArmTypes.vhd</code> . Sie sollten in Ihrer Implementierung ebenfalls die vordefinierten Konstanten verwenden, um Flüchtigkeitsfehler bei den Modus-Codes zu vermeiden.
<code>USER_BIT</code>	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 <code>USER_BIT</code> gesetzt ist, wird innerhalb der Adressübersetzung <code>THIS_MODE</code> ignoriert und stattdessen der User-Modus für die Adressübersetzung angenommen.
Rückgabewert	Der Rückgabewert der Funktion ist eine 5-Bit-Registeradresse in Form eines <code>std_logic_vector</code> . (<i>physische Registeradresse</i>)

Tabelle 1: Argumentliste der Function `get_internal_address`

¹Die 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.

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 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`.

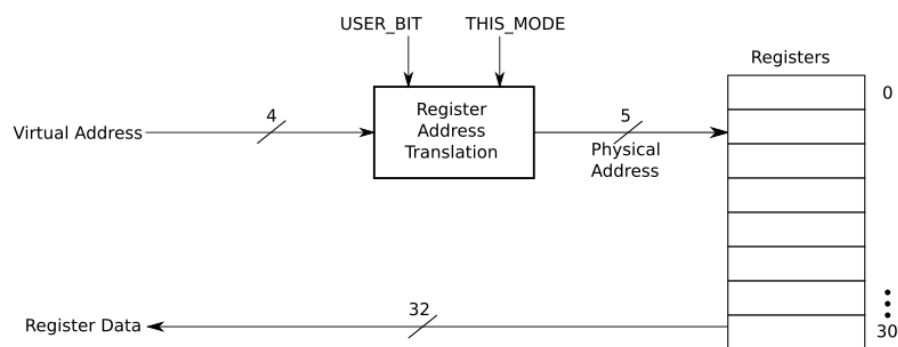


Abbildung 1: Schematische Funktionsweise der Register-Address-Translation (RAT) Einheit.

²Dies 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.

³Die Kommentare weisen das Synthesewerkzeug *Vivado Synthesis* 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 Eigenschaften 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 Tabelle 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 <u>nicht</u> , 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 ADDR B 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 korrespondierende Datum ausgegeben wird.
WEB	in	Mit ENB = 1 und WEB = 0 wird ein Lesezugriff an ADDR B 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 ADDR B 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 4: 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 Ihre Abschätzung mit dem Synthese Report. Vergleichen Sie die Realisierung der beiden Aufgaben.

Literatur

- [1] ARM Limited, Hrsg. *ARM Architecture Reference Manual*. 2005.
- [2] David A. Patterson und John L. Hennessy. *Rechnerorganisation und-entwurf*. Spektrum Akademischer Verlag, Sep. 2005, S. 593. ISBN: 3827415950, 9783827415950.
- [3] TU Berlin FG Rechnertechnologie. *Hardwarepraktikum Technische Informatik Übersicht über die ARM-Befehlssatzarchitektur*. Okt. 2010.
- [4] 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.