Technische Universität München
Fakultät für Informatik
Rechnerarchitektur-Praktikum
SS 2015

SPEICHERTECHNOLOGIE – DRAM-INTERFACE

ENTWICKLERDOKUMENTATION

Bearbeitet von:

Mahdi Sellami

12.07.2015

INHALTSVERZEICHNIS

1. Einleitung

In Rahmen des Rechnerarchitektur Praktikums an der Technischen Universität München, wurde mithilfe der Hardwarebeschreibungssprache VHDL der Decoder Multiplexer eines DRAM Interfaces simuliert. Dieses Dokument enthält die entsprechende Entwicklerdokumentation, gerichtet an Entwickler, welche die Implementierung weiterentwickeln oder anpassen möchten.

2. Anforderungen

Das Projekt basiert auf GHDL (*http://home.gna.org/ghdl/*), einem VHDL Simulator, welcher wiederrum von der GCC Compiler-Suite (*https://gcc.gnu.org/)* abhängt. GHDL generiert bei der Simulation u.a. eine GHDL Wave Datei, welche zur Entwicklung u.U. hilfreich sein könnte. Daher wird die Verwendung von GTKWave (*http://gtkwave.sourceforge.net/*) zum Inspizieren solcher Dateien empfohlen. Eine Weiterentwicklung des Projektes ist jedoch auch ohne GTKWave möglich.

3. Ordnerstruktur

Im Folgenden wird die Organisation der Ordnerstruktur des Projektes dargestellt und die genaue Funktion jedes Ordners erläutert:

DecoderMultiplexer	root-Verzeichnis des Projektes
simulation	Enthält alle automatisiert generierten Dateien
src	Source Code des VHDL Programms
testbench	Source Code der Testbench für das VHDL Programm

4. Spezifikation der Ein- und Ausgänge

Die folgende Tabelle enthält alle Informationen zu allen Ein- und Ausgängen des Decoder Multiplexers:

Name	Modus	Datentyp	Beschreibung der Daten
MUX	IN	std_logic	Signal zum Initialisieren der Zeilenadresse
/CAS	IN	std_logic	Signal zum Initialisieren der Spaltenadresse und zur Übergabe der 4-Byte Enable-Signale (/BE)
CLK	IN	std_logic	Taktsignal
AD	IN	std_logic_vector (25 downto 2)	AD(25 downto 15): Zeilenadresse AD(14 downto 12): Selektion eines von vier DRAM Modulen AD(12 downto 2): Spaltenadresse
/BE	IN	std_logic_vector (3 downto 0)	4-Byte Enable-Signale
A	OUT	std_logic_vector (10 downto 0)	Ausgang für gemultiplexte Zeilen- und Spaltenadressen
/CAS0	OUT	std_logic_vector (3 downto 0)	Ausgang für 4-Byte Enable-Signale für DRAM Modul 0
/CAS1	OUT	std_logic_vector (3 downto 0)	Ausgang für 4-Byte Enable-Signale für DRAM Modul 1
/CAS2	OUT	std_logic_vector (3 downto 0)	Ausgang für 4-Byte Enable-Signale für DRAM Modul 2
/CAS3	OUT	std_logic_vector (3 downto 0)	Ausgang für 4-Byte Enable-Signale für DRAM Modul 3

5. Beschreibung des Programmablaufs

Im Folgenden wird zum besseren Verständnis der aktuelle Programmablauf der VHDL Simulation in Pseudo-Code dargestellt:

```
Wenn falling_edge(MUX)
    Initialisiere A(10:0) mit 11 höherwertigen Bits von AD(25:2)
Wenn falling_edge(/CAS)
    Initialisiere A(10:0) mit 11 niederwertigen Bits von AD(25:2)
    Wenn AD(14 downto 12) is
        case 00: Initialisiere /CAS0 mit /BE
        case 01: Initialisiere /CAS0 mit /BE
        case 10: Initialisiere /CAS0 mit /BE
        case 11: Initialisiere /CAS0 mit /BE
```

6. Ansätze zur Weiterentwicklung

Zum Zeitpunkt der Fertigstellung der ersten Version der VHDL Simulation bestehen bereits Ideen zur Weiterentwicklung des Programms. Auf diese werden im Folgenden genauer eingegangen:

1. **Optimierung der Testbench**: Die derzeitigen Tests, die durch die Testbench durchgeführt werden, sollten verbessert werden. Dies kann in erster Instanz durch das Hinzufügen von mehr Testfällen geschehen. In zweiter Instanz sollte der Fokus auf spezifischere Tests gelegt werden, sodass eine hohe Abdeckung des VHDL Programms durch die Testbench erreicht wird.
2. **Design Hierarchie:** Um ein strukturell wertvolles VHDL Programm im Sinne der Wartbarkeit bieten zu können, sollte zum Ziel gesetzt werden, die Zuweisung der 11 höher- und 11 niederwertigen Bits an A(10:0) in einer extra Component zu handhaben. Dies hat zum Vorteil, dass strukturell verstärkt mit Komponenten gearbeitet wird, wodurch eine Erweiterung des Programms erleichtert wird.