Technische Universität München

Fakultät für Informatik

Rechnerarchitektur-Praktikum

SS 2015

**SPEICHERTECHNOLOGIE – DRAM-INTERFACE**

ANWENDERDOKUMENTATION

Bearbeitet von:

**Mahdi Sellami**

12.07.2015

**INHALTSVERZEICHNIS**

1. Einleitung 3

2. Beschreibung des Programms 3

3. VERWENDUNG DES programms 3

3.1. benötigte Software 3

3.2. Ausführen des programms 4

3.3. beschreibung der verwendeten ghdl-kommandos 5

# Einleitung

Im Rahmen des ERA-Praktikums im Sommersemester 2015 soll die Gruppe 52 das Projekt “Speichertechnologie – DRAM-Interface” in VHDL realisieren. Dies ist die Anwenderdokumentation. Sie soll dem Anwender die Aufgabe des Programms und deren Verwendung erklären.

# Beschreibung des Programms

Ein DRAM besteht aus einem Timing Generator, 4 DRAM-Module und einem Decoder Multiplexer.

In VHDL wurde eine Simulation des Decoder Multiplexers entwickelt. Ihm stehen die Eingänge /RAS, /MUX, /CAS, AD(25:2), /BE(3:0) und CLK zur Verfügung. Seine Aufgabe besteht darin, das in AD(25:2) anliegende Signal in Zeilen- und Spaltenadresse aufzuteilen und an den A(10:0) Ausgang zu übermitteln. außerdem selektiert er ein entsprechendes DRAM-Modul durch die  Ausgänge /CASx(3:0) mithilfe von /BE(3:0), /CAS und MUX.

# Verwendung des Programms

# 

# 3.1. Benötigte Software

Um das Programm erfolgreich zu kompilieren sind folgende Softwares erforderlich:

-          GHDL: ein open-source Simulator für die VHDL Sprache. GHDL ermöglicht es, ein VHDL Code in einem Computer zu kompilieren und auszuführen. (Mehr Details zur Installation unter http://home.gna.org/ghdl/)

-          GTKWave: ein Anzeigeprogramm für digitale Signalformen. (Mehr Details zur Installation unter http://gtkwave.sourceforge.net/)

== Windows ==

-          Unter Windows soll noch die Cygwin-Umgebung installiert sein. (Mehr Details zur Installation unter https://www.cygwin.com/)

# 3.2. Ausführen des Programms

Im Ordner liegen folgende Files:

-Makefile

-decoderMultiplexer.vhd (Code des Decoder Multiplexer)

-decoderMultiplexer\_tb (Code für Testbench)

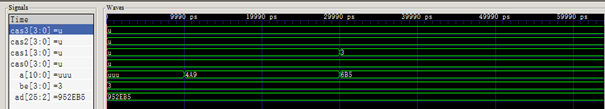
Mit “make run” wird decoderMultiplexer.vhd und decoderMultiplexer\_tb.vhd kompiliert und gelinked und daraus ein ausführbares Programm (decoderMultiplexer.ghw) erzeugt.

|  |
| --- |
| $ make run  make -C simulation run  make[1]: Entering directory ‘../decodermultiplexer/simulation’  ghdl -a ../src/decoderMultiplecer.vhd  ghdl -s ../src/decoderMultiplecer.vhd  ghdl -a ../testbench/decoderMultiplexer\_tb.vhd  ghdl -s ../testbench/decoderMultiplecer\_tb.vhd  ghdl -e decoderMultiplecer\_tb  ghdl -r decoderMultiplecer\_tb --stop-time-15sec --wave=decoderMultiplexer.ghw  ../testbench/decoderMultiplexer\_tb.vhd:73:16:@0ms:(report note): DecoderMultiplexer initiated.  ../testbench/decoderMultiplexer\_tb.vhd:74:16:@0ms:(report note): Staring testbench process…  ../testbench/decoderMultiplexer\_tb.vhd:75:16:@0ms:(report note): Initializing AD(25:2) with test value (100101010010111010110101).  ../testbench/decoderMultiplexer\_tb.vhd:77:16:@0ms:(report note): Initializing BE(3:0) with test value (0011).  ../testbench/decoderMultiplexer\_tb.vhd:80:16:@0ms:(report note): Simulating falling\_edge(MUX)  ../src/decoderMultiplexer.vhd:46:32:@10ns:(report note): falling\_edge(MUX)  ../testbench/decoderMultiplexer\_tb.vhd:87:16:@20ns:(report note): A = 1193  ../testbench/decoderMultiplexer\_tb.vhd:88:16:@20ns:(report note): AD(25 downto 15) = 1193  ../testbench/decoderMultiplexer\_tb.vhd:93:16:@20ns:(report note): Simulating falling\_edge(CAS)  ../src/decoderMultiplexer.vhd:55:32:@30ns:(report note): falling\_edge(CAS)  ../testbench/decoderMultiplexer\_tb.vhd:100:16:@40ns:(report note): A = 1717  ../testbench/decoderMultiplexer\_tb.vhd:101:16:@40ns:(report note): AD(12 downto 2) = 1717  ../testbench/decoderMultiplexer\_tb.vhd:107:16:@40ns:(report note): BE = 3  ../testbench/decoderMultiplexer\_tb.vhd:108:16:@40ns:(report note): CAS1 = 3  ../testbench/decoderMultiplexer\_tb.vhd:116:16:@40ns:(report note): Test completed. |

*Ausgabe der erfolgreiche Kompilierung des Programms*

Das erzeugte Programm decoderMultiplexer.ghw kann man danach im Ordner “simulation” finden und mit GTKWave öffnen.

Die aus dem Testlauf generierte Waveform sieht wie auf dem folgenden Bild aus:



Mit „make clean“ können danach alle generiete Dateien aus dem Ordner ausgelöscht werden.

# 3.3. Beschreibung der verwendeten GHDL-Kommandos:

-a analysiert bzw. kompiliert das VHD-File:    ghdl -a file\_name.vhd

-s überprüft die Syntax des VHD-Files:          ghdl -s file\_name.vhd

-e elaboriert and koppelt:        ghdl -e unit\_name

-r führt die Simulation aus:        ghdl -r unit\_name

--clean löscht die generierten Files aus:        ghdl --clean

--remove löscht die generierten Files and Libraries aus:   ghdl --remove