Technische Universität Darmstadt
FG Eingebettete Systeme und
ihre Anwendungen (ESA)

Prof. Dr. Andreas Koch
Holger Lange

01.07.2007

Einführung in Computer Microsystems
7. Aufgabenblatt
Sommersemester 2007

Aufgabe: Video-Speicher

Der mit dem 6. Aufgabenblatt um variabel programmierbare Auflösungen erweiterte Video-Controller „Discount" soll nun einen echten, beschreibbaren Video-Speicher (auch *Framebuffer* genannt) erhalten. Der Framebuffer soll zur Laufzeit ebenfalls über den bereits zur Manipulation der Auflösungsregister verwendeten Programmierbus beschrieben werden. Da der Framebuffer eine Grösse von 16 KBytes hat, muss der bisherige Programmieradressbus um ein Bit verbreitert werden, um in der unteren Hälfte der Adressen wie bisher die Auflösungsregister anzusprechen und in der oberen Hälfte den Framebuffer. Nehmen Sie die auf der Homepage des FG ESA bereitgestellte Lösung des 6. Aufgabenblattes (ISE-Projekt **`discount_loesung.zip`**) als Basis für Ihre weiteren Arbeiten und gehen Sie wie folgt vor:

a) Das zusätzlich auf der Homepage des FG ESA bereitgestellte Archiv **`video_speicher.zip`** enthält unter anderem die Verilog-Datei **`rom.v`**, welche das aus der Vorlesung (Foliensatz 5, ab Folie 44) bekannte On-Chip-ROM implementiert. Wandeln Sie dieses ROM-Modul in ein RAM-Modul um, indem Sie zusätzlich das **`WE`** (Write Enable) -Signal und den **`DI`** (Data In) -Bus an den **`RAMB16_S1`**-Instanzen verbinden und an die Modulschnittstelle führen. Das neue RAM-Modul wird nun als viertes Modul im Modul **`discount_ram`** (umbenannt von **`discount_res`**) instanziert. Deshalb wird der Pixeldaten- und Adressbus von **`memacc_ram`** (umbenannt von **`memacc_res`**) nun nicht mehr an die Modulschnittstelle von **`discount_ram`** geführt, sondern als interner Bus mit dem RAM verbunden. Das RAM soll über den Programmierbus nur beschrieben, nicht aber gelesen werden können. Schließen Sie den bidirektionalen Programmierdatenbus und den Pixeldatenbus von **`memacc_ram`** geeignet an die separaten, *unidirektionalen* Lese- und Schreibdatenbusse des RAMs an, um die geforderte Funktionalität möglichst einfach zu erreichen.

b) Verbreitern Sie den Adressbus, indem Sie in **`discount_defs.v`** den Wert von **`` `Asz ``** anpassen. Implementieren Sie eine Adressdekoderlogik, so dass das in a) instanzierte RAM von der oberen

Hälfte des verdoppelten Adressraumes angesprochen wird. Achtung, das RAM wird nun sowohl vom Programmierbus als auch von **memacc_ram** adressiert! Die Auflösungsregister dürfen nur auf die untere Hälfte der Adressen reagieren (präzise belegen sie nach wie vor die Adressen 0 bis 7). Passen Sie dazu die Adressdekoderlogiken der Module **memacc_ram**, **hcount_ram** (umbenannt von **hcount_res**) und **vcount_ram** (umbenannt von **vcount_res**) an.

c) Da der Framebuffer nun Teil von Discount und somit on-chip ist, werden alle Deklarationen, Initialisierungen und Verhaltensmodelle bezüglich des in der Testbench modellierten ROM-Speichers überflüssig. Entfernen Sie das modellierte ROM aus der Testbench sowie die Adress- und Datenleitungen (nun Discount-intern, siehe a)) aus der Instanzierung von **discount_ram**. Auch die Diagonaleninitialisierung und -überwachung ist bei einem frei beschreibbaren Framebuffer gegenstandslos. Stattdessen soll eine vorgegebene Bitmap über den Programmierbus in den Framebuffer geschrieben werden. Die ebenfalls im Archiv **video_speicher.zip** bereitgestellte Datei **athene_logo_sw_304x114.mem** enthält die Bitmap in ASCII-Darstellung, welche mittels folgender Deklarationen und Kommandos in der Testbench in die Variable **BITMAP** einzulesen ist:

```
...
  reg  [0:34655]  BITMAP[0:0];    // Zwischenspeicher fuer Test-Bitmap
...
  initial
    // Test-Bitmap aus Datei in temporäre Variable lesen
    $readmemb("athene_logo_sw_304x114.mem", BITMAP);
...
```

Die Variable **BITMAP** enthält nun die Bitmap der Größe 304*114 Punkte zeilenweise als eindimensionales Feld. Greifen Sie auf die einzelnen Bildpunkte (ein Pixel entspricht einem Bit) mittels eines Kommandos

```
Bildpunkt = BITMAP[0][X];
```

zu, wobei **X=0** die linke obere Ecke der Bitmap beschreibt, **X=303** die rechte obere Ecke sowie **X=34655** die rechte untere Ecke. Schreiben Sie das Bild *Byte*-weise von links oben nach rechts unten zeilenweise über den Programmierbus in den Framebuffer. Fügen Sie dazu am Ende des **initial**-Blocks hinter den schon vorhandenen Kommandos zur Auflösungseinstellung „320*200“ (diese Auflösungseinstellung wird beibehalten) ein geeignetes Schleifenkonstrukt ein. Die Bitmapdaten sollen zentriert dargestellt werden, etwa vertikal von Zeile 42 bis 156 sowie horizontal von Reihe 8 bis 312. Führen Sie eine Verhaltenssimulation durch und prüfen die nun erzeugten PBM-Dateien visuell.