Übung zur Vorlesung Einführung in Computer Microsystems

Prof. Dr. A. Koch
Thorsten Wink

TECHNISCHE UNIVERSITÄT DARMSTADT

Sommersemester 11
Übungsblatt 6

Aufgabe 6.1 Discount - variable Auflösung

Der aus der Vorlesung (Foliensatz 5) bekannte Video-Controller „Discount" soll nun variable Auflösungen ausgeben können. Da diese je nach Anwendung und angeschlossenem Monitor auch zur Laufzeit wechseln können, muss Discount um eine Programmierschnittstelle z.B. für Mikroprozessoren erweitert werden. Gehen Sie dazu wie folgt vor:

- Öffnen Sie das auf der Homepage bereitgestellte ISE-Projekt `discount.zip`, welches die Verilog-Quellen von Discount enthält. In der Datei `discount_defs.v` beschreiben die Konstanten `'H1`, `'H`, `'H2`, `'H3` und `'Hline` den horizontalen Bildaufbau, die Konstanten `'V1`, `'V`, `'V2`, `'V3` und `'Vframe` den vertikalen Bildaufbau. `'Msz` beschreibt die Größe des Bildspeichers. Ersetzen Sie diese Konstanten in den Modulen `discount`, `hcount`, `vcount` und `memacc` durch Register der Bitbreite 10 (Breite `'Asz` im Fall von Register `Msz`). Durch einen Reset via `NRST` sollen alle neuen Register den Wert der jeweiligen Konstanten als Default erhalten.

- Erweitern Sie die oben genannten Module um einen Programmierbus, bestehend aus den Unterbussen `PADDR` und `PDATA` sowie dem Signal `WE`, um die neuen Register beschreiben und wieder auslesen zu können (Modul `memacc` nur beschreiben). Das Signal `WE` gibt an, ob ein Register beschrieben (`WE == 1`) oder gelesen (`WE == 0`) werden soll. Der Adressbus `PADDR` überträgt die Adressen der einzelnen Register, `'H1`, `'H`, `'H2` und `'H3` erhalten die Adressen 0 bis 3 und `'V1`, `'V`, `'V2` und `'V3` die Adressen 4 bis 7. Der Datenbus `PDATA` ist bidirektional auszulegen, überträgt also sowohl Schreib- als auch Lesedaten. Entwickeln Sie für jedes der angesprochenen Module eine Adressdekodierlogik (siehe dazu auch Foliensatz 3, Abschnitt „Busse" ab Folie 24) , die bei Lesezugriffen im folgenden Takt auf `PDATA` den Wert des gewünschten Registers via Tri-State-Treiber bereitstellt und bei Schreibzugriffen im selben Takt den Wert auf `PDATA` in das adressierte Register speichert.

- Die neuen Register `Msz`, `Hline` und `Vframe` werden nicht direkt beschrieben, ihre Werte sollen laufend in der Hardware aus den übrigen neuen Registern berechnet werden, analog zur Berechnung der Konstanten in `discount_defs.v`.

- Passen Sie die Testbench an, indem Sie den Programmierbus hinzufügen. Testen Sie zunächst mit *inaktivem* Programmierbus, ob die Default-Auflösung 640*192 (Reset-Werte der neuen Register) noch korrekt ausgegeben wird. Schauen Sie sich auch die erzeugte PBM-Datei an.

- Schreiben Sie nun einen neuen `initial`-Block für die Testbench, der eine Auflösung von 320*200 Punkten über den Programmierbus einstellt (größere Auflösungen sind theoretisch mit breiteren Registern möglich, leider wird der ISE-Simulator dann sehr langsam und unbenutzbar). Verwenden Sie dazu folgende Werte für die einzelnen Register:

H1	H	H2	H3	V1	V	V2	V3
48	320	44	44	60	200	50	16

  Lesen Sie die Register `H` und `V` über den Programmierbus wieder zurück und geben Sie die hoffentlich korrekten Werte mit `$display` aus. Vergleichen Sie die nun erzeugte PBM-Datei.

Aufgabe 6.2 Discount - Framebuffer

Der in der vorherigen Aufgabe 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. Gehen Sie wie folgt vor:

- Verwenden Sie das bekannte On-Chip-ROM aus der Vorlesung (Folie 5-53). 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.

- Verbreitern Sie den Adressbus, indem Sie in `discount_defs.v` den Wert von `'Asz` anpassen. Implementieren Sie eine Adressdekoderlogik, so dass das 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.

- 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) 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 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.

Diese Hausaufgaben müssen bis 15.7.11, 18:00 über das Moodle-System abgegeben werden.

Hausaufgabe 6.1 Bresenham-Algorithmus (5 Punkte)

Der Bresenham-Algorithmus ist ein recht einfacher Linien-Raster-Algorithmus, um zwischen 2 Pixeln eine Linie zu zeichnen. In C kann er beispielsweise wie folgt implementiert werden:

```
void line(int x0, int y0, int x1, int y1)
{
  int dx =  abs(x1-x0), sx = x0<x1 ? 1 : -1;
  int dy = -abs(y1-y0), sy = y0<y1 ? 1 : -1;
  int err = dx+dy, e2;

  for(;;){
    setPixel(x0,y0);
    if (x0==x1 && y0==y1) break;
    e2 = 2*err;
    if (e2 >= dy) { err += dy; x0 += sx; }
    if (e2 <= dx) { err += dx; y0 += sy; }
  }
}
```

Im Internet finden sich viele Erläuterungen und auch Beispiele zum intuitiven Verständnis.

Implementieren Sie den Bresenham-Algorithmus in Verilog. Er soll folgendes Interface haben:

```
module bresenham(
  input  wire       clk,
  input  wire       reset,
  input  wire [7:0] x_0,           //Koordinaten des Startpunktes
  input  wire [7:0] y_0,
  input  wire [7:0] x_1,           //Koordinaten des Endpunktes
  input  wire [7:0] y_1,
  input  wire       input_valid,   //=1: Es liegen gültige Eingabedaten an
  output wire [7:0] x_out,         //Koordinaten des aktuellen Bildpunktes
  output wire [7:0] y_out,
  output wire       output_valid   //=1: aktueller Bildpunkt gültig
  output wire       ready_for_data //=1: Modul ist bereit, neue Eingabedaten anzunehmen
)
```

Schreiben Sie eine Testbench, um die korrekte Funktion zu überprüfen.

Hausaufgabe 6.2 Einbindung in Discount (5 Punkte)

Das Modul `bresenham` aus der vorherigen Aufgabe soll nun dazu verwendet werden, den Framebuffer des Discount-Videocontrollers zu beschreiben. Gehen Sie dazu wie folgt vor:

- In der bisherigen Testbench wurde der Framebuffer zu Beginn der Simulation aus einer externen Datei eingelesen. Dieser Teil entfällt nun, stattdessen werden die Daten aus dem Bresenham-Modul verwendet. Beachten Sie dabei, dass die Programmierung der Auflösung weiterhin möglich sein muss.
- Instanziieren Sie das Modul `bresenham` in der Testbench. Zeichnen Sie dann mehrere Linien, indem Sie nacheinander an das Modul `bresenham` die Koordinaten anlegen und die daraus resultierenden Ausgangsdaten (die den Koordinaten der Linienpunkten entsprechen) in den Framebuffer eintragen.
- Die Auflösung soll 320*200 betragen.

Plagiarismus

Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Weitere Infos unter www.informatik.tu-darmstadt.de/plagiarism