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.

Lösung:

Die modifizierten DISCOUNT-Module sind mit ***_ram.v** bezeichnet.

discount_ram.v:

```
//------------------------------------------------------------------
//
// DISCOUNT: Display-Controller
//
// Stellt einen 8-Bit-Speicher auf einem Video-Monitor monochrom dar
// (Aufloesung 640x192, Bildfrequenz 50 Hz).
//
//------------------------------------------------------------------
```

```
//-------------------------------------------------------------------
//
// Modul discount_ram - Version mit variabel programmierbarer Aufloesung
//                      und beschreibbarem Videospeicher
//
// Display-Controller (Cathode-Ray-Tube-Controller)
//
//-------------------------------------------------------------------

`include "discount_defs.v"

module discount_ram (PIXEL, CSYNC, HSYNC, VSYNC,
                     CLOCK, NRST, PADDR, WE, PDATA);

  // Ausgaenge
  output            PIXEL,  // Bildpunkt (0: dunkel, 1: hell)
                    CSYNC,  // kombinierte Synchronisation
                            //   (HSYNC und VSYNC)
                    HSYNC,  // horizontale (Zeilen-)Synchronisation
                            //   (1-aktiv)
                    VSYNC;  // vertikale (Bild-)Synchronisation
                            //   (1-aktiv)

  // Eingaenge
  input             CLOCK,  // Takt (einphasig)
                    NRST,   // Initialisierung (0-aktiv)
                    WE;     // Write-Enable fuer Programmierbus
  input  [`Asz-1:0] PADDR;  // Programmier-Adressbus

  // Bidirektional
  inout  [`Rsz-1:0] PDATA;  // Programmier-Datenbus

  // Port-Typen
  wire   [`Asz-1:0] PADDR;
  wire              PIXEL,
                    CSYNC,
                    HSYNC,
                    VSYNC,
                    CLOCK,
                    NRST,
                    WE;
  wire   [`Rsz-1:0] PDATA;

  // interne Variablen
  wire              ADRIE,    // Freigabe: Bilddaten uebernehmen und
                              //   Bildadresse erhoehen
                              //   (address increment enable)
                    NXTAD,    // Bildadresse hochzaehlen
                    NXTLN;    // Zeile hochzaehlen
  wire   [`Asz-1:0] ADDR;     // Adresse von MEMACC
  wire   [`Asz-1:0] RAM_ADDR; // Adresse fuer Bildspeicher
  wire   [`Dsz-1:0] DATA;     // Daten aus dem Bildspeicher

  //
  // kombinierte Synchronisation
```

```
    //
    assign CSYNC = HSYNC ^ VSYNC;

    //
    // Bildspeicheradresse
    // entweder vom Programmierbus (neuen Inhalt schreiben)
    // oder von MEMACC (Bilddaten darstellen)
    //
    assign RAM_ADDR = (WE & PADDR[14]) ? PADDR : ADDR;

    //
    // Instanzen
    //
    hcount_ram HCOUNT (NXTAD, NXTLN, HSYNC, CLOCK, NRST, PADDR, PDATA, WE);
    vcount_ram VCOUNT (ADRIE, VSYNC, NXTLN, CLOCK, NRST, PADDR, PDATA, WE);
    memacc_ram MEMACC (ADDR,  PIXEL, DATA,  ADRIE, NXTAD,
                       CLOCK, NRST, PADDR, PDATA, WE);
    ram        RAM    (CLOCK, NRST, RAM_ADDR[13:0],
                       PDATA[`Bsz-1:0], WE & PADDR[14], DATA);

endmodule  // discount_ram
```

hcount_ram.v:

```
//-----------------------------------------------------------------------
//
// Modul hcount_ram - Version mit variabel programmierbarer Aufloesung
//                    und beschreibbarem Videospeicher
//
// Timing der Zeilen; Steuersignale fuer Adressen, Bildpunkte,
// horizontale Synchronisation und Timing des Bildes
//
//-----------------------------------------------------------------------
//
// Schnittstelle
//
// NXTAD  im naechsten Takt Bildspeicherdaten uebernehmen,
//        zur naechsten Bildspeicheradresse weiterschalten (1-aktiv)
// NXTLN  im naechsten Takt Zeile hochzaehlen (1-aktiv)
// HSYNC  horizontale (Zeilen-)Synchronisation (1-aktiv)
// CLOCK  Takt
// NRST   Initialisierung (0-aktiv)
// PADDR  Programmier-Adressbus
// DATA   Programmier-Datenbus
// WE     Write-Enable fuer Programmierbus
//
//-----------------------------------------------------------------------
//
// Timing und Arbeitsperiode von hcount:
//
// H1 Takte  linker Bildrand
//           HSYNC=0, NXTAD=0, NTXLN=0
// H  Takte  Bildpunkte
//           HSYNC=0, NXTLN=0, NXTAD wird alle 8 Takte gesetzt
// H2 Takte  rechter Bildrand
//           HSYNC=0, NXTAD=0, NTXLN=0
```

```
// H3 Takte  horizontale Synchronisation
//            HSYNC=1, NXTAD=0, NXTLN wird im letzten Takt 1
//
//-----------------------------------------------------------------------

`include "discount_defs.v"

module hcount_ram (NXTAD, NXTLN, HSYNC, CLOCK, NRST, PADDR, DATA, WE);

  // Ausgaenge
  output NXTAD,      // Bildadresse hochzaehlen
         NXTLN,      // Zeile hochzaehlen
         HSYNC;      // Zeilensynchronisation

  // Eingaenge
  input            CLOCK,      // Takt
                   NRST,       // Initialisierung (0-aktiv)
                   WE;         // Write-Enable
  input [`Asz-1:0] PADDR;      // Adressbus

  // Bidirektional
  inout [`Rsz-1:0] DATA;       // Datenbus

  // Port-Typen
  reg              NXTAD,
                   NXTLN,
                   HSYNC;
  reg  [`Rsz-1:0] DATA_OUT;
  wire             CLOCK,
                   NRST;
  wire [`Asz-1:0] PADDR;

  // interne Variablen
  reg [`Rsz-1:0] HCNT;    // Taktzaehler innerhalb einer Zeile
  reg [1:0]      HPHASE;  // Phasen horizontal:
                          //   0 linker Bildrand
                          //   1 Bildpunkte
                          //   2 rechter Bildrand
                          //   3 horizontale Synchronisation
  reg [`Rsz-1:0] H1, // Register (variable Aufloesung)
                 H,
                 H2,
                 H3,
                 Hline;
  reg [`Rsz-1:0] Hline_temp,  // Temporaere Flip-Flops
                 Hline_temp2; // fuer Pipeline
  reg            ADDRESSED;   // Gueltige Adresse im vorigen Takt


  //
  // Bidirektionalen Bus asynchron treiben
  // Daten einen Takt verzoegert zu Adressen -> ADDRESSED
  //
  assign DATA = (WE | ~ADDRESSED) ? `Rsz'bz : DATA_OUT;
```

```
//
// Register synchron auslesen
//
always @(posedge CLOCK or negedge NRST) begin
  if (NRST == 0) begin                          // Reset, magic DisCount
    DATA_OUT <= `Rsz'hDC;
    ADDRESSED <= 0;
  end
  else begin
    case (PADDR[2:0])
      0: DATA_OUT <= H1;
      1: DATA_OUT <= H;
      2: DATA_OUT <= H2;
      3: DATA_OUT <= H3;
      default: DATA_OUT <= `Rsz'hBA;          // magic Bad Address
    endcase
    ADDRESSED <= ~PADDR[2] & ~PADDR[14];
  end
end

//
// Register schreiben
//
always @(posedge CLOCK or negedge NRST) begin
  if (NRST == 0) begin                          // Alle Register auf
    H1 <= `H1;                                  // Default-Werte setzen
    H  <= `H;
    H2 <= `H2;
    H3 <= `H3;
    Hline       <= `H1+`H+`H2+`H3;
    Hline_temp  <= `H1+`H;
    Hline_temp2 <= `H1+`H+`H2;
  end
  else begin
    Hline_temp <= H1 + H;                       // Summe in Pipeline
    Hline_temp2 <= Hline_temp + H2;             // berechnen
    Hline <= Hline_temp2 + H3;
    if (WE)                                     // Es wird geschrieben
      case ({PADDR[14],PADDR[2:0]})             // Daten uebernehmen
        0: H1 <= DATA;
        1: H  <= DATA;
        2: H2 <= DATA;
        3: H3 <= DATA;
      endcase
  end
end

//
// Taktzaehler aktualisieren
// (Periode Hline, Bereich 0..Hline-1)
//
always @(posedge CLOCK or negedge NRST) begin
  if (NRST == 0)         HCNT <= `Hline-1;    // Reset
  else
    if (HCNT == Hline-1) HCNT <= 0;           // neue Zeile
```

```
      else                  HCNT <= HCNT+1;
    end

    //
    // Phasenzaehler aktualisieren
    //
    always @(posedge CLOCK or negedge NRST) begin
      if (NRST == 0) HPHASE <= 3;        // Reset
      else
        case (HCNT)
          0:         HPHASE <= 0;        // linker Bildrand
          H1:        HPHASE <= HPHASE+1; // Bildpunkte
          H1+H:      HPHASE <= HPHASE+1; // rechter Bildrand
          H1+H+H2:   HPHASE <= HPHASE+1; // Bildsynchronisation
        endcase
    end

    //
    // horizontale Synchronisation aktualisieren;
    // HSYNC wird ueber ein Register ausgegeben, um die Verzoegerung
    // auf NXTAD von einem Takt zu kompensieren und so das Timing nach
    // aussen einzuhalten
    //
    always @(posedge CLOCK or negedge NRST) begin
      if (NRST == 0)     HSYNC <= 1;   // Reset
      else
        if (HPHASE == 3) HSYNC <= 1;
        else             HSYNC <= 0;
    end

    //
    // NXTAD und NXTLN aktualisieren
    //
    always @(HCNT or HPHASE) begin
      if ((HCNT % `Bsz == 1 )                    // neues Byte
        && (HPHASE == 1))               NXTAD = 1;
      else                              NXTAD = 0;
      if ((HCNT == 0) && (HPHASE == 3)) NXTLN = 1;
      else                              NXTLN = 0;
    end

  endmodule // hcount_ram
```

vcount_ram.v:

```
  //-----------------------------------------------------------------------
  //
  // Modul vcount_ram - Version mit variabel programmierbarer Aufloesung
  //                    und beschreibbarem Videospeicher
  //
  // Timing des Bildes; Steuersignale fuer Adressen, Bildpunkte und
  // vertikale Synchronisation
  //
  //-----------------------------------------------------------------------
  //
  // Schnittstelle
```

```
//
//  ADRIE  Freigabe: Bilddaten uebernehmen und Bildadresse erhoehen
//         (address increment enable) (1-aktiv)
//  VSYNC  vertikale (Bild-)Synchronisation (1-aktiv)
//  NXTLN  im naechsten Takt Zeile hochzaehlen (1-aktiv)
//  CLOCK  Takt
//  NRST   Initialisierung (0-aktiv)
//  PADDR  Programmier-Adressbus
//  DATA   Programmier-Datenbus
//  WE     Write-Enable fuer Programmierbus
//
//-----------------------------------------------------------------------
//
// Timing: die Arbeitsperiode von vcount dauert Vframe Zeilen und
// beginnt bei NXTLN=1 mit einer steigenden Taktflanke:
//
// V1 Zeilen  oberer Bildrand
//             VSYNC=0, ADRIE=0
// V  Zeilen  Bildzeilen
//             VSYNC=0, ADRIE=1
// V2 Zeilen  unterer Bildrand
//             VSYNC=0, ADRIE=0
// V3 Zeilen  vertikale Synchronisation
//             VSYNC=1, ADRIE=0
//
//-----------------------------------------------------------------------

`include "discount_defs.v"

module vcount_ram (ADRIE, VSYNC, NXTLN, CLOCK, NRST, PADDR, DATA, WE);

  // Ausgaenge
  output ADRIE,      // Freigabe: Bildadresse erhoehen
         VSYNC;      // Bildsynchronisation

  // Eingaenge
  input             NXTLN,      // Zeile hochzaehlen
                    CLOCK,      // Takt
                    NRST,       // Initialisierung (0-aktiv)
                    WE;         // Write-Enable
  input [`Asz-1:0] PADDR;      // Adressbus

  // Bidirektional
  inout [`Rsz-1:0] DATA;       // Datenbus


  // Port-Typen
  reg              ADRIE,
                   VSYNC;
  reg  [`Rsz-1:0] DATA_OUT;
  wire             NXTLN,
                   CLOCK,
                   NRST;
  wire [`Asz-1:0] PADDR;
```

```
// interne Variablen
reg [`Rsz-1:0] VCNT;     // Zeilenzaehler
reg [1:0]      VPHASE;   // Phasen vertikal:
                         //   0 oberer Bildrand
                         //   1 Bildbereich
                         //   2 unterer Bildrand
                         //   3 vertikale Synchronisation
reg [`Rsz-1:0] V1, // Register (variable Aufloesung)
               V,
               V2,
               V3,
               Vframe;
reg [`Rsz-1:0] Vframe_temp,  // Temporaere Flip-Flops
               Vframe_temp2; // fuer Pipeline
reg            ADDRESSED;    // Gueltige Adresse im vorigen Takt


//
// Bidirektionalen Bus asynchron treiben
// Daten einen Takt verzoegert zu Adressen -> ADDRESSED
//
assign DATA = (WE | ~ADDRESSED) ? `Rsz'bz : DATA_OUT;

//
// Register synchron auslesen
//
always @(posedge CLOCK or negedge NRST) begin
  if (NRST == 0) begin                        // Reset, magic DisCount
    DATA_OUT <= `Rsz'hDC;
    ADDRESSED <= 0;
  end
  else begin
    case (PADDR[2:0])
      4: DATA_OUT <= V1;
      5: DATA_OUT <= V;
      6: DATA_OUT <= V2;
      7: DATA_OUT <= V3;
      default: DATA_OUT <= `Rsz'hBA;          // magic Bad Address
    endcase
    ADDRESSED <= PADDR[2] & ~PADDR[14];
  end
end

//
// Register schreiben
//
always @(posedge CLOCK or negedge NRST) begin
  if (NRST == 0) begin                        // Alle Register auf
    V1 <= `V1;                                // Default-Werte setzen
    V  <= `V;
    V2 <= `V2;
    V3 <= `V3;
    Vframe       <= `V1+`V+`V2+`V3;
    Vframe_temp  <= `V1+`V;
    Vframe_temp2 <= `V1+`V+`V2;
```

```
  end
  else begin
    Vframe_temp <= V1 + V;                    // Summe in Pipeline
    Vframe_temp2 <= Vframe_temp + V2;         // berechnen
    Vframe <= Vframe_temp2 + V3;
    if (WE)                                   // Es wird geschrieben
      case ({PADDR[14],PADDR[2:0]})           // Daten uebernehmen
        4: V1 <= DATA;
        5: V  <= DATA;
        6: V2 <= DATA;
        7: V3 <= DATA;
      endcase
  end
end

//
// Zeilenzaehler aktualisieren
// (Periode Vframe, Bereich 0..Vframe-1)
//
always @(posedge CLOCK or negedge NRST) begin
  if (NRST == 0)            VCNT <= 0;         // Reset
  else
    if (NXTLN == 1) begin                      // neue Zeile
      if (VCNT == Vframe-1) VCNT <= 0;         // neues Bild
      else                  VCNT <= VCNT+1;
    end
end

//
// Phasenzaehler aktualisieren
//
always @(posedge CLOCK or negedge NRST) begin
  if (NRST == 0) VPHASE <= 3;        // Reset
  else
    if (NXTLN == 1)                  // neue Zeile
      case (VCNT)
        0:       VPHASE <= 0;        // oberer Bildrand
        V1:      VPHASE <= VPHASE+1; // Bildbereich
        V1+V:    VPHASE <= VPHASE+1; // unterer Bildrand
        V1+V+V2: VPHASE <= VPHASE+1; // vertikale Synchronisation
      endcase
end

//
// vertikale Synchronisation aktualisieren
//
always @(posedge CLOCK or negedge NRST) begin
  if (NRST == 0)     VSYNC <= 1;        // Reset
  else
    if (VPHASE == 3) VSYNC <= 1;
    else             VSYNC <= 0;
end

//
// ADRIE aktualisieren
```

```
  //
  always @(VPHASE) begin
    if (VPHASE == 1) ADRIE <= 1;
    else             ADRIE <= 0;
  end

endmodule // vcount_ram
```

memacc_ram.v:

```
//-----------------------------------------------------------------------
//
// Modul memacc_ram - Version mit variabel programmierbarer Aufloesung
//                    und beschreibbarem Videospeicher
//
// Adressen fuer den Bildspeicher generieren, Bilddaten als Bytes
// lesen und als Punkte ausgeben
//
//-----------------------------------------------------------------------
//
// Schnittstelle
//
//  ADDR   Adresse fuer Bildspeicher
//  PIXEL  Bildpunkt (0: dunkel, 1: hell)
//  DATA   Daten aus dem Bildspeicher
//  ADRIE  Freigabe: Bilddaten uebernehmen und Bildadresse erhoehen
//         (address increment enable)
//  NXTAD  Daten uebernehmen, Bildadresse hochzaehlen
//  CLOCK  Takt
//  NRST   Initialisierung (0-aktiv)
//  PADDR  Programmier-Adressbus
//  PDATA  Programmier-Datenbus
//  WE     Write-Enable fuer Programmierbus
//
//-----------------------------------------------------------------------
//
// Timing und Funktion:
//
// Bei einer positiven Taktflanke und ADRIE=NXTAD=1 werden das
// Bildpunkt-Schieberegister mit den momentanen Bildspeicherdaten
// geladen und die Bildspeicheradresse erhoeht. Letztere liegt im
// Bereich von 0 bis `Msz, worauf wieder 0 folgt.
// Ist ADRIE oder NXTAD 0, wird das Bildpunkt-Schieberegister
// weitergeschoben und beim niedrigstwertigen Bit mit 0 gefuellt.
//
//-----------------------------------------------------------------------

`include "discount_defs.v"

module memacc_ram (ADDR, PIXEL, DATA, ADRIE, NXTAD,
                   CLOCK, NRST, PADDR, PDATA, WE);

  // Ausgaenge
  output [`Asz-1:0] ADDR;   // Adresse fuer Bildspeicher
  output            PIXEL;  // Bildpunkt
```

```
// Eingaenge
input  [`Dsz-1:0] DATA;   // Daten aus dem Bildspeicher
input             ADRIE,  // Freigabe: Bildadresse erhoehen
                  NXTAD,  // Bildadresse hochzaehlen
                  CLOCK,  // Takt
                  NRST,   // Initialisierung (0-aktiv)
                  WE;     // Write-Enable fuer Programmierbus
input [`Asz-1:0]  PADDR;  // Programmieradressbus

// Bidirektional
inout [`Rsz-1:0]  PDATA;  // Programmierdatenbus

// Port-Typen
reg    [`Asz-1:0] ADDR;
wire   [`Dsz-1:0] DATA;
wire              PIXEL,
                  ADRIE,
                  NXTAD,
                  CLOCK,
                  NRST,
                  WE;
wire   [`Asz-1:0] PADDR;
wire   [`Rsz-1:0] PDATA;

// interne Variablen
reg    [`Dsz-1:0] PIXSR;  // Bildpunkt-Schieberegister
reg    [`Rsz-1:0] H,      // Register (variable Aufloesung)
                  V;
reg    [`Asz-1:0] Msz;

//
// Bildpunkt aus Schieberegister abgreifen
//
assign PIXEL = PIXSR[`Dsz-1];

//
// Register schreiben
//
always @(posedge CLOCK or negedge NRST) begin
  if (NRST == 0) begin                        // Alle Register auf
    H   <= `H;                                // Default-Werte setzen
    V   <= `V;
    Msz <= (`H)*(`V) / `Bsz;
  end
  else begin
    Msz <= H * V / `Bsz; // Bildgroesse berechnen (/`Bsz wird Verdrahtung!)
    if (WE)                                   // Es wird geschrieben
      case ({PADDR[14],PADDR[2:0]})           // Daten uebernehmen
        1: H <= PDATA;
        5: V <= PDATA;
      endcase
  end
end

//
```

```
  // Bildspeicheradresse aktualisieren
  //
  always @(posedge CLOCK or negedge NRST) begin
    if (NRST == 0)                    ADDR <= 0;  // Reset
    else
      if ((NXTAD == 1) && (ADRIE == 1)) begin     // neue Adresse
        if (ADDR == Msz-1) ADDR <= 0;             // Ende Bildspeicher
        else               ADDR <= ADDR+1;
      end
  end

  //
  // Bildpunkt-Schieberegister aktualisieren
  //
  always @(posedge CLOCK or negedge NRST) begin
    if (NRST == 0) PIXSR <= 0;                    // Reset
    else
      if ((NXTAD == 1) && (ADRIE == 1))
        PIXSR <= DATA;                            // neues Byte laden
      else PIXSR <= {PIXSR[`Dsz-2:0], 1'b0};      // schieben
  end

endmodule // memacc_ram
```

Bei **crt.v** wurde nur der Name der Toplevel-Testbench angepasst:

```
//-----------------------------------------------------------------------
//
//  Modul crt
//
//  Simuliert den an den Display-Controller angeschlossenen Monitor
//
//-----------------------------------------------------------------------

`include "discount_defs.v"

module crt(
  input PIXEL, CSYNC, HSYNC, VSYNC, SNAPSHOT
);

  // Stiehl clock von der Testbench
  // Es waere realistischer, die Clock aus den PIXEL-
  // und Synchronsignalen zurueckzugewinnen. Dies wuerde
  // allerdings mindestens ein Monitorbild mit diskreten
  // einzelnen weissen oder schwarzen Pixeln erfordern,
  // was aber nicht garantiert werden kann
  wire CLK = test_ram.CLOCK; //<-- Hier neuer Toplevel-Name

  // Bildpuffer
  reg [2**23-1:0] VBUF;

  // Index in Bildpuffer
  integer i;

  // Spalten und Zeilen
  integer hsize, vsize;
```

```
// Frame ist vollstaendig
reg FRAME = 0;

// Laufende Nr. fuer Bilddatei
integer snapshot_count = 0;

// Impliziter Zustandsautomat
always begin
  // neuer Frame (Achtung, nach Reset 'bx)
  while (VSYNC !== 0) @(posedge CLK);
  i = 0; hsize = 0; vsize = 0;
  FRAME = 0;
  // auf ersten HSYNC warten
  // (faengt auch halben Frame,
  // falls nicht mit VSYNC begonnen)
  while (!HSYNC) @(posedge CLK);
  while (HSYNC) @(posedge CLK);
  // Pixel bis zum naechsten HSYNC merken
  while (!VSYNC) begin
    while (!HSYNC) begin
      VBUF[i] = PIXEL;
      i = i + 1;
      @(posedge CLK);
    end
    // HSYNC abwarten
    while (HSYNC) @(posedge CLK);
    // Zeilenbreite einmal merken
    if (!hsize) hsize = i;
    vsize = vsize + 1;
  end
  FRAME = 1;
end

// Frame als PBM speichern (SNAPSHOT == 1)
always @(posedge SNAPSHOT or posedge FRAME) begin: local_SNAPSHOT
  integer i, fd;
  reg [8*256:1] fn;

  // Wenn Bild vollstaendig und SNAPSHOT aktiv
  if (SNAPSHOT && FRAME) begin
    $display("Writing bitmap 'snapshot_%0d.pbm' : hsize=%0d vsize=%0d Pixels=%0d",
             snapshot_count, hsize, vsize, hsize * vsize);
    // Dateinamen lfd. Nr. generieren und Datei oeffnen
    $swrite(fn, "snapshot_%0d.pbm", snapshot_count);
    fd = $fopen(fn, "wb");
    if (!fd) begin
      $write("*** SNAPSHOT: Couldn't open file for writing.\n");
      $finish;
    end
    // PBM Kopfdaten schreiben
    $fwrite(fd, "P1\n%0d %0d\n", hsize, vsize);
    // Bitmap schreiben
    for (i = 0; i < hsize * vsize; i = i + 1)
      $fwrite(fd, "%b ", ~VBUF[i]);
```

```
      $fclose(fd);
      snapshot_count = snapshot_count + 1;
    end
  end

endmodule
```

Definitionen in **discount_defs.v**, hier wurde lediglich **`Asz** auf 15 erhöht:

```
//-----------------------------------------------------------------------
//
// DISCOUNT: Display-Controller
//
// Stellt einen 8-Bit-Speicher auf einem Video-Monitor monochrom dar
// (Aufloesung 640x192, Bildfrequenz 50 Hz).
//
//-----------------------------------------------------------------------

`ifdef DISCOUNT_DEFS_V
`else
`define DISCOUNT_DEFS_V

`timescale 1ps / 1ps

                                // horizontale Parameter in Takten:
`define H1    96                // linker Bildrand
`define H    640                // Bildpunkte
`define H2    88                // rechter Bildrand
`define H3    88                // horizontale Synchronisation
`define Hline `H1+`H+`H2+`H3    // Zeilenlaenge

                                // vertikale Parameter in Zeilen:
`define V1    58                // oberer Bildrand
`define V    192                // Bildbereich
`define V2    48                // unterer Bildrand
`define V3    16                // vertikale Synchronisation
`define Vframe `V1+`V+`V2+`V3   // Bildaufbau in Zeilen

// Definitionen
//`define Asz   14                // Adressbreite
`define Asz   15                // Adressbreite (RAM-Version)
`define Dsz    8                // Datenbreite
`define Bsz    8                // Byte-Breite
`define Msz   (`H)*(`V) / `Bsz  // Speichergroesse
`define Rsz   10                // Registerbreite H../V..

`endif
```

Testbench **test_ram.v**:

```
//-----------------------------------------------------------------------
//
//  Modul test_ram - Version mit variabel programmierbarer Aufloesung
//                   und beschreibbarem Videospeicher
//
//  Testumgebung fuer den Display-Controller
```

```
//
//-----------------------------------------------------------------------

`include "discount_defs.v"

module test_ram;

  `define Simtime ((`Hline)*(`Vframe)*120)  // 1.2 Frames simulieren

  // Deklarationen
  reg             CLOCK,          // Takt
                  NRST,           // Initialisierung
                  WE,             // Write-Enable Programmierbus
                  SNAPSHOT;       // Bild speichern (PBM)
  reg  [`Asz-1:0] PADDR;          // Programmier-Adressbus
  wire [`Rsz-1:0] PDATA;          // Bidirektionaler Programmierdatenbus
  reg  [`Rsz-1:0] PDATA_REG;      // Bidirektionaler Programmierdatenbus
  wire            PIXEL,          // Bildpunkt
                  CSYNC,          // kombinierte Synchronisation
                  HSYNC,          // horizontale Synchronisation
                  VSYNC;          // vertikale Synchronisation
  integer         H, V;           // Bildposition
  reg  [0:34655]  BITMAP[0:0];    // Zwischenspeicher fuer Test-Bitmap

  //
  // Testinstanz des Display-Controllers
  //
  discount_ram DISCOUNT (PIXEL,CSYNC,HSYNC,VSYNC,
                         CLOCK,NRST,PADDR,WE,PDATA);

  //
  // Instanz des Testmonitors (schreibt PBM)
  //
  crt CRT (PIXEL,CSYNC,HSYNC,VSYNC,SNAPSHOT);

  //
  // Bidirektionalen Bus treiben
  assign PDATA = PDATA_REG;

  //
  // Takt erzeugen
  //
  always begin
    CLOCK = 1'b0;
    #50;
    CLOCK = 1'b1;
    #50;
  end

  //
  // Reset
  //
  initial begin
    NRST = 1'b0;
    @(negedge CLOCK);
```

```
  NRST = 1'b1;
end

initial
  #`Simtime $stop;                    // Simulationsdauer

//
// Textausgabe
//
initial
  $monitor("Takt=%d  PIXEL=%b  HSYNC=%b  VSYNC=%b",
    $time/100, PIXEL, HSYNC, VSYNC);

//
// Snapshot speichern
//
initial
  SNAPSHOT = 1;

//
// Aufloesung programmieren
// Test-Bitmap via Programmierbus in den Bildspeicher schreiben
//
initial begin
  PADDR = 0; WE = 0; PDATA_REG = `Rsz'bz;
  #200;
  @(negedge CLOCK);
  // Aufloesung 320*200 Punkte
  WE = 1; PDATA_REG = 48; #100;
  PADDR = 1; PDATA_REG = 320; #100;
  PADDR = 2; PDATA_REG = 44; #100;
  PADDR = 3; #100;
  PADDR = 4; PDATA_REG = 60; #100;
  PADDR = 5; PDATA_REG = 200; #100;
  PADDR = 6; PDATA_REG = 50; #100;
  PADDR = 7; PDATA_REG = 16; #100;
  // Wieder auslesen
  WE = 0; PDATA_REG = `Rsz'bz; PADDR = 1; #100;
  $display("Neue Aufloesung: x %d", PDATA);
  PADDR = 5; #100;
  $display("Neue Aufloesung: y %d", PDATA);
  //
  // Test-Bitmap aus Datei in temporäre Variable lesen
  $readmemb("athene_logo_sw_304x114.mem", BITMAP);
  // Bitmap via Programmierbus zeilenweise in Bildspeicher
  // schreiben, so dass die Bitmap zentriert erscheint
  // Bitmap hat 304x114 Punkte => V: 42-156, H: 8-312
  $display("Schreibe Bitmap in Videospeicher...");
  PADDR = 1 + (1 << 14); // Videospeicher adressieren
  WE = 1;                // schreiben
  for (V = 0; V < 114; V = V + 1) begin
    for (H = 0; H < 304; H = H + `Bsz) begin
      // Jeweils 8 Bit via Programmierbus schreiben,
      // einen Takt warten
      PDATA_REG = BITMAP[0][V*304+H +: `Bsz]; #100;
```

```
        // Adresse hochzaehlen
        PADDR = PADDR + 1;
      end
      // Am Zeilenende 16 Pixel ueberspringen
      // (Breite der Bitmap nur 304)
      PADDR = PADDR + 2;
    end
    WE = 0; PDATA_REG = 'Rsz'bz;
    $display("Bitmap schreiben beendet.");
  end

endmodule // test_ram
```