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

Prof. Dr. Andreas Koch
Holger Lange

26.06.2007

Einführung in Computer Microsystems
6. Aufgabenblatt
Sommersemester 2007

Aufgabe: Discount

Der aus der Vorlesung (Foliensatz 5 bis Folie 40) 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 des FG ESA 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.

Lösung:

Die modifizierten DISCOUNT-Module sind mit ***_res.v** (→ Resolution) bezeichnet. Achten Sie darauf, in der Testbench (**test_res.v**) die beiden mit *Anpassen!* markierten Zeilen auf die jeweils programmierte Auflösung einzustellen.

discount_res.v:

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

//------------------------------------------------------------------
//
// Modul discount_res - Version mit variabel programmierbarer Auflosung
//
// Display-Controller (Cathode-Ray-Tube-Controller)
//
//------------------------------------------------------------------

`include "discount_defs.v"

module discount_res (ADDR, PIXEL, CSYNC, HSYNC, VSYNC,
                     DATA, CLOCK, NRST, PADDR, WE, PDATA);

  // Ausgaenge
```

```
  output [`Asz-1:0] ADDR;   // Adresse fuer Bildspeicher
  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  [`Dsz-1:0] DATA;   // Daten aus dem Bildspeicher
  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] ADDR,
                    PADDR;
  wire   [`Dsz-1:0] DATA;
  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

  //
  // kombinierte Synchronisation
  //
  assign CSYNC = HSYNC ^ VSYNC;

  //
  // Instanzen
  //
  hcount_res HCOUNT (NXTAD, NXTLN, HSYNC, CLOCK, NRST, PADDR, PDATA, WE);
  vcount_res VCOUNT (ADRIE, VSYNC, NXTLN, CLOCK, NRST, PADDR, PDATA, WE);
  memacc_res MEMACC (ADDR,  PIXEL, DATA,  ADRIE, NXTAD,
                     CLOCK, NRST, PADDR, PDATA, WE);

endmodule  // discount_res
```

hcount_res.v:

```
//-----------------------------------------------------------------------
//
// Modul hcount_res - Version mit variabel programmierbarer Aufloesung
//
// 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)
// ADDR   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_res (NXTAD, NXTLN, HSYNC, CLOCK, NRST, ADDR, 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] ADDR;       // 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] ADDR;

// 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            ADDR2_REG;   // Um einen Takt verzoegertes ADDR[2]


//
// Bidirektionalen Bus asynchron treiben
// Daten einen Takt verzoegert zu Adressen -> ADDR2_REG
//
assign DATA = (WE | ADDR2_REG) ? `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;
    ADDR2_REG <= 0;
  end
  else begin
    case (ADDR[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
    ADDR2_REG <= ADDR[2];
  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 (ADDR[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_res
```

vcount_res.v:

```
//-----------------------------------------------------------------------
//
// Modul vcount_res - Version mit variabel programmierbarer Aufloesung
//
// 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)
//  ADDR   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_res (ADRIE, VSYNC, NXTLN, CLOCK, NRST, ADDR, 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] ADDR;       // 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] ADDR;

  // 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            ADDR2_REG;    // Um einen Takt verzoegertes ADDR[2]


  //
  // Bidirektionalen Bus asynchron treiben
  //
  assign DATA = (WE | ~ADDR2_REG) ? `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;
    ADDR2_REG <= 0;
  end
  else begin
    case (ADDR[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
    ADDR2_REG <= ADDR[2];
  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 (ADDR[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_res
```

memacc_res.v:

```
  //-------------------------------------------------------------------
  //
  // Modul memacc_res - Version mit variabel programmierbarer Aufloesung
  //
  // 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_res (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)                            // 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_res
```

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_res.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**:

```
`timescale 1 ns / 1 ps
//------------------------------------------------------------------
//
// 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 Dsz    8                    // Datenbreite
`define Bsz    8                    // Byte-Breite
`define Msz   (`H)*(`V) / `Bsz      // Speichergroesse
`define Rsz   10                    // Registerbreite H../V..

`endif
```

Testbench **test_res.v**, die beiden mit *Anpassen!* markierten Zeilen bitte auf die jeweils programmierte Auflösung einstellen:

```
//-----------------------------------------------------------------------
//
//  Modul test_res - Version mit variabel programmierbarer Aufloesung
//
//  Testumgebung fuer den Display-Controller
//
//-----------------------------------------------------------------------

`include "discount_defs.v"

module test_res;

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

  // Deklarationen
  reg [`Bsz-1:0]  MEM[0:`Msz-1];  // Bildspeicher
  reg [`Bsz-1:0]  TEMP;           // Hilfsvariable (Speicher)
  reg [`Dsz-1:0]  DATA;           // Bildspeicherausgang
  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 [`Asz-1:0] ADDR;           // Bildspeicheradresse
  wire            PIXEL,          // Bildpunkt
                  CSYNC,          // kombinierte Synchronisation
                  HSYNC,          // horizontale Synchronisation
```

```
                VSYNC;          // vertikale Synchronisation
integer         BYTE, BIT,      // Speicherposition
                H, V;           // Bildposition

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

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

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

//
// Speicher initialisieren, weisse Diagonale
//
initial begin
  BYTE = 0;                             // Initialisierung
  BIT = `Bsz-1;
  TEMP = MEM[0];
  for (V = 0; V < 200; V = V+1)         // Zeilen <- Anpassen!
    for (H = 0; H < 320; H = H+1) begin // Punkte <- Anpassen!
      if (V == H) TEMP[BIT] = 1;        // Diagonale
      else        TEMP[BIT] = 0;
      if (BIT == 0) begin               // neues Byte
        MEM[BYTE] = TEMP;
        if (BYTE < `Msz-1)              // nicht letztes Byte
          BYTE = BYTE+1;
        TEMP = MEM[BYTE];
        BIT = `Bsz-1;
      end
      else BIT = BIT-1;
    end
end

//
// 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

//
// Bildspeicherdaten auslesen
//
always @(ADDR) begin
  DATA = MEM[ADDR];
end

initial
  #`Simtime $stop;                    // Simulationsdauer

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

//
// Diagonale verifizieren
//
always @(posedge CLOCK) begin
  H = DISCOUNT.HCOUNT.HCNT-DISCOUNT.HCOUNT.H1-1;      // Bildposition
  V = DISCOUNT.VCOUNT.VCNT-DISCOUNT.VCOUNT.V1;

  if ((H>=0) && (H< DISCOUNT.HCOUNT.H) && // echter Bildpunkt
      (V> 0) && (V<=DISCOUNT.VCOUNT.V))

    if (((H==V) && (PIXEL==0)) ||      // Diagonale
        ((H!=V) && (PIXEL==1)))        // ausserhalb Diagonale
      $display(
      "falsche Diagonale bei Takt %0d, PIXEL = %b, H = %0d, V = %0d"
      ,$time/100, PIXEL, H, V);
end

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

//
// Aufloesung programmieren
//
initial begin
  PADDR = 0; WE = 0; PDATA_REG = `Rsz'bz;
  #200;
  @(negedge CLOCK);
  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);
  end

endmodule // test_res
```