



A. Koch

# Einführung in Computer Microsystems

## 5. Anwendungsbeispiel: Video-Controller

Andreas Koch

FG Eingebettete Systeme und ihre Anwendungen  
Informatik, TU Darmstadt

Sommersemester 2007



A. Koch

# Einführung

# Anwendungsbeispiel



A. Koch

- Demonstriert Benutzung von Verilog
- Lösung einer Aufgabe durch Hardware
- Keine Software-Instruktionen
- **Keine** prozessorartigen Strukturen
  - Standen im Fokus von **TGDI2**

# Was macht ein Video-Controller?

Nun Konkurrenz für ...



A. Koch



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

**nVIDIA.**

# DISCOUNT



A. Koch



- Ausgabe von Schwarzweißbildern auf Monitor
- Genauer:
  - Bild ist in **Speicher** abgelegt
  - DISCOUNT generiert ein **brauchbares** Videosignale

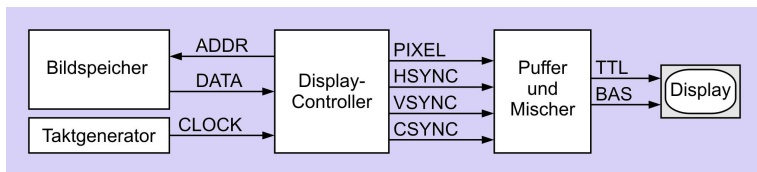


- Bild-Austast-Synchron-Signal
- Für **Graustufenbilder**
- Daten **analog** kodiert als Spannungspegel
  - Bilddaten (0,3 V = schwarz, 1 V = weiß)
  - Synchronisation (Zeilen, Bilder) bei 0 V
  - Austastbereich (0,3 V)
  - Vorgegebene Dauer und Reihenfolge der Bereiche
- Ab jetzt **Nachrichtentechnik** ???
- Nein, **keine** Panik
- Alles Informatik-kompatibel vereinfacht!
  - Hier: Reines Schwarz und Weiß (Pixelwerte 0 und 1)
  - Nur Erzeugung von **Steuersignalen** für Analogteil
  - **Keine** Handhabung von Analogsignalen (z.B. via DAC)

# Spezifikation: Aufbau



A. Koch

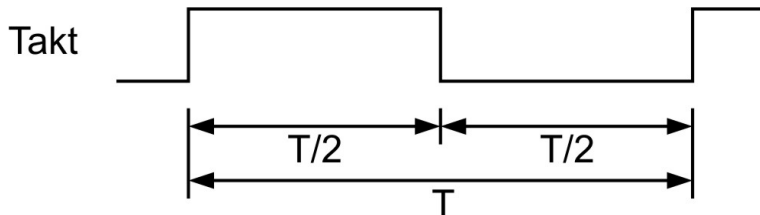


ADDR	Adresse
DATA	Bildspeicher [ADDR]
CLOCK	Basistakt
PIXEL	Bildpunkt (1=hell, 0=dunkel)
HSYNC	horizontale Synchronisation der Bildzeilen
VSYNC	vertikale Synchronisation ganzer Bilder
CSYNC	kombiniertes HSYNC und VSYNC
TTL	Schnittstelle für getrennte PIXEL, HSYNC, VSYNC
BAS	gemischtes Signal

# Spezifikation: Timing

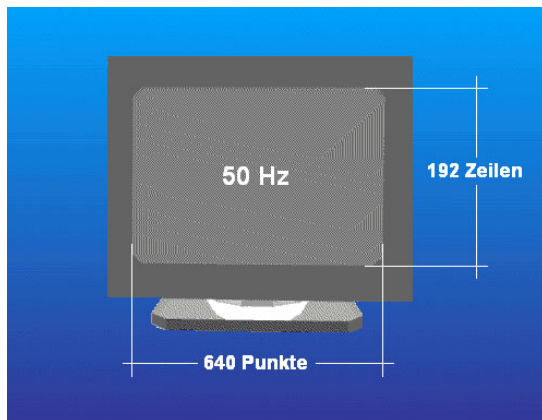


A. Koch



- Nur selten: "So schnell wie möglich"
- In der Regel: Erfüllen externer Anforderungen
- Aber **nicht** langsamer!





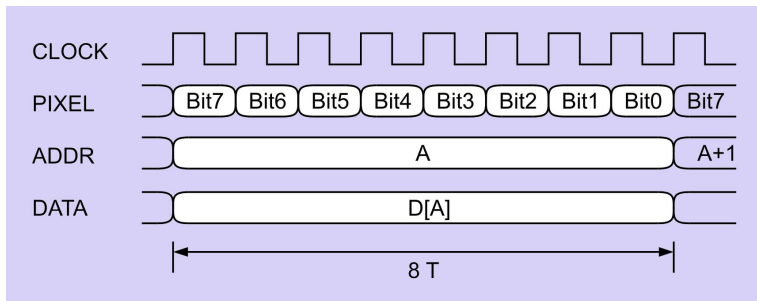
Zwar kein HDTV, aber reicht zum Üben ...

# Quelle für Bilddaten: Speicher

1 Bit pro Bildpunkt



A. Koch

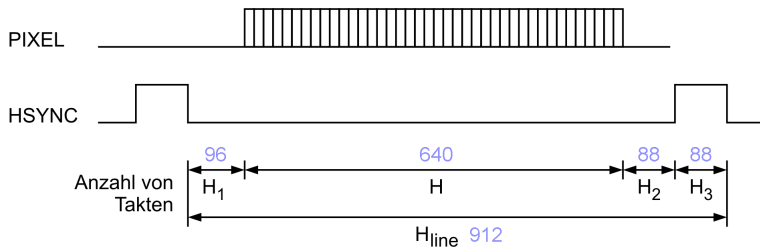


Wie hoch muß Taktfrequenz sein, um  $192 \cdot 640$  Punkte mit 50 Hz auf Ausgang PIXEL zu liefern?

6,144 MHz?

**Nein**, Videosignal ist komplizierter!

# Bildaufbau: Zeile aus Pixel



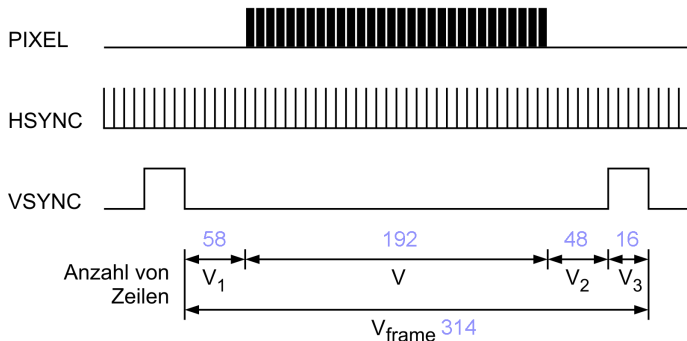
A. Koch

- H1** Hintere Schwarzschulter, legt u.a. Schwarzpegel fest
- H** Bilddaten
- H2** Vordere Schwarzschulter, Verzögerungszeit zur Erkennung von HSYNC
- H3** Horizontale Synchronisation, zeigt Zeilenende an ( $\rightarrow$  nächste Zeile)

# Bildaufbau: Bild aus Zeilen



A. Koch



- $V_1$  Oberer nicht sichtbarer Teil (wäre unscharf)
- $V$  Sichtbare Zeilen
- $V_2$  Unterer nicht sichtbarer Teil (wäre unscharf)
- $V_3$  Vertikale Synchronisation, setzt Elektronenstrahl nach links oben

## Timing, 2. Versuch



A. Koch

912 Spalten \* 314 Zeilen  
= 286.368 Pixel je Bild je 1/50 Sekunde  
= 14.318.400 Pixel / Sekunde  
 $\approx$  **14,3 MHz** Pixeltaktfrequenz (Periode 70 ns)

Mehr als doppelt so hoch wie die Frequenz der reinen  
Bilddaten!

# Organisation des Bildspeichers



A. Koch

- Breite = 1 Byte = 8 Bit
- Je Bildzeile:

$$\frac{H \text{ Pixel}}{8 \text{ Pixel / Byte}} = 80 \text{ Bytes}$$

- \* 192 Zeilen → 15360 Bytes pro Bild
- Speicher mit 14b Adressen adressieren
  - Würde für  $2^{14}$  Bytes = 16384 Bytes reichen
- Linke obere Bildecke = Bit 7 des Bytes an Adresse 0
- Während der Austastlücken: **keine** Daten



A. Koch

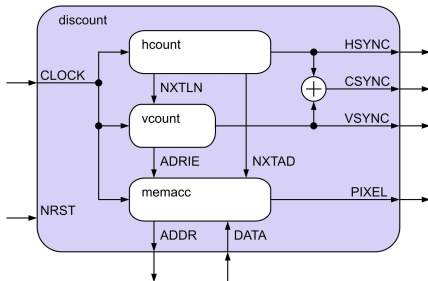
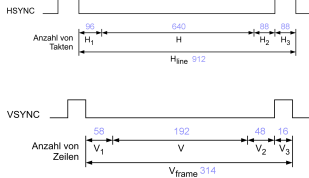
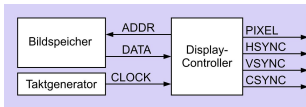
# Verilog-Modell

# Grobstruktur

## Wie vorgehen?



A. Koch





# Nützliche Konstanten



A. Koch

```
'ifndef DISCOUNT_DEFS_V
'else
'define DISCOUNT_DEFS_V

'define H1 96 // horizontale Parameter in Takten:
'define H 640 // linker Bildrand
'define H2 88 // Bildpunkte
'define H3 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
'endif
```

# Top-Level-Modul



A. Koch

```
include "discount_defs.v"

module discount (ADDR, PIXEL, CSYNC, HSYNC, VSYNC, DATA, CLOCK, NRST);

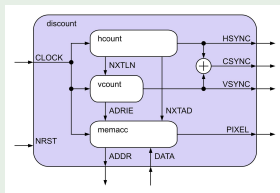
    ... // Eingänge / Ausgänge

    // 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 HCOUNT (NXTAD, NXTLN, HSYNC, CLOCK, NRST);
    vcount VCOUNT (ADRIE, VSYNC, NXTLN, CLOCK, NRST);
    memacc MEMACC (ADDR, PIXEL, DATA, ADRIE, NXTAD, CLOCK, NRST);

endmodule // discount
```



# HCOUNT: Kopf



A. Koch

```
// Timing und Arbeitsperiode von hcount:  
//  
// 'H1 Takte linker Bildrand  
//           HSYNC=0, NXTAD=0, NXLN=0  
// 'H Takte Bildpunkte  
//           HSYNC=0, NXLN=0, NXTAD wird alle 8 Takte gesetzt  
// 'H2 Takte rechter Bildrand  
//           HSYNC=0, NXTAD=0, NXLN=0  
// 'H3 Takte horizontale Synchronisation  
//           HSYNC=1, NXTAD=0, NXLN wird im letzten Takt 1  
//  
//-----
```

```
'include "discount_defs.v"
```

```
module hcount (NXTAD, NXLN, HSYNC, CLOCK, NRST);
```

```
... // Ein-/Ausgaenge
```

# HCOUNT: Rumpf 1/2



A. Koch

```
// interne Variablen
reg [9:0] HCNT; // Taktzaehler innerhalb einer Zeile
reg [1:0] HPHASE; // Phasen horizontal:
                // 0 linker Bildrand
                // 1 Bildpunkte
                // 2 rechter Bildrand
                // 3 horizontale Synchronisation

// 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
```

# HCOUNT: Rumpf 2/2



A. Koch

```
//  
// horizontale Synchronisation aktualisieren;  
// HSYNC wird ueber ein Register ausgegeben, um die Verzoeigerung  
// 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
```

# VCOUNT: Kopf



A. Koch

```
//-----  
//  
// 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 (ADRIE, VSYNC, NXTLN, CLOCK, NRST);  
  
    ... // Eingaenge / Ausgaenge
```

# VCOUNT: Rumpf 1/3



A. Koch

```
// interne Variablen
reg [8:0] VCNT; // Zeilenzaehler
reg [1:0] VPHASE; // Phasen vertikal:
                // 0 oberer Bildrand
                // 1 Bildbereich
                // 2 unterer Bildrand
                // 3 vertikale Synchronisation

//
// 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
end
```

# VCOUNT: Rumpf 2/3



A. Koch

```
//  
// 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
```



# VCOUNT: Rumpf 3/3



A. Koch

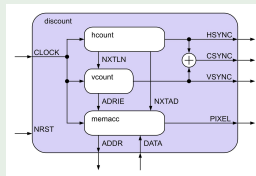
```
//  
// vertikale Synchronisation aktualisieren  
//  
always @(posedge CLOCK or negedge NRST) begin  
  if (NRST == 0) VSYNC <= 1;    // Reset  
  else  
    if (VPHASE == 3) VSYNC <= 1;  
    else          VSYNC <= 0;  
end  
  
//  
// ADRIE aktualisieren  
//  
always @(VPHASE) begin  
  if (VPHASE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

# MEMACC: Modulkopf



A. Koch

```
//-----  
//  
// 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  
// weiterschoben und beim niedrigstwertigen Bit mit 0 gefuell.  
//  
//-----
```



```
'include "discount_defs.v"
```

```
module memacc (ADDR, PIXEL, DATA, ADRIE, NXTAD, CLOCK, NRST);
```

```
... // Eingänge und Ausgänge
```

# MEMACC: Rumpf 1/2



A. Koch

```
// interne Variablen
reg  [Dsz-1:0] PIXSR; // Bildpunkt-Schieberegister

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

//
// 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
```

# MEMACC: Rumpf 2/2



A. Koch

```
//  
// 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
```



- Problem: Signalverläufe sind
  - Hunderte von Takten lang
  - auf Details zu überprüfen
- Manuell
  - mühsam
  - fehleranfällig
- Idee
  - Automatische Überprüfung von Ausgaben
- Hier
  - Verwende Diagonale als Eingabebild
  - Überprüfe, ob Ausgabedaten ebenfalls Diagonale beschreiben

# Testrahmen für den Video-Controller 1/4

## Kopf und UUT-Instanziierung



A. Koch

```
'include "discount_defs.v"

module test;

  '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
  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 DISCOUNT (ADDR,PIXEL,CSYNC,HSYNC,VSYNC,DATA,CLOCK,NRST);
```

# Testrahmen für den Video-Controller 2/4

## Verwaltungsarbeiten



A. Koch

```
// Takt erzeugen: Periode von 100 Zeiteinheiten
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: Hier kombinatorisches ROM!
always @(ADDR) begin
    DATA = MEM[ADDR];
end

'define Simtime (('Hline)*('Vframe)*120) // 1.2 Frames simulieren
initial
    #Simtime $stop;           // Simulationsdauer

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

# Testrahmen für den Video-Controller 3/4

Erzeugen einer Diagonale als Eingabebild



A. Koch

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



# Testrahmen für den Video-Controller 4/4

## Automatische Überprüfung der Ausgabe auf Diagonalfom



A. Koch

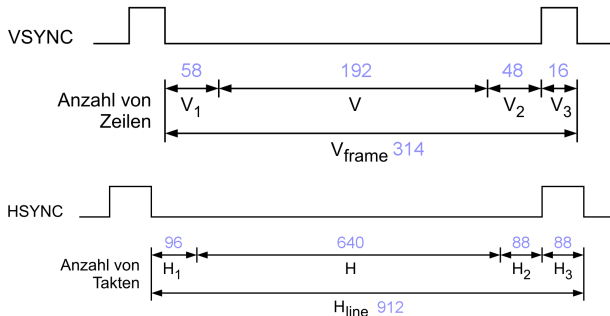
```
//  
// Diagonale verifizieren  
//  
always @(posedge CLOCK) begin  
  H = DISCOUNT.HCOUNT.HCNT-'H1-1; // Bildposition (White-Box-Test!)  
  V = DISCOUNT.VCOUNT.VCNT-'V1;  
  
  if ((H>=0) && (H<'H) &&          // echter Bildpunkt  
      (V> 0) && (V<='V))  
  
    if (((H==V) && (PIXEL==0)) ||    // 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
```

# Interpretation der Simulationsergebnisse

## 1. Überlegung: Erster gesetzter Pixel bei Diagonale



A. Koch



Rechnung: **Heller** Pixel muss auftauchen nach:

58 Zeilen (obere Austastlücke,  $V_1$ ) zu je 912 Takten

+ 96 Takte bis Beginn der Bilddaten ( $H_1$ )

+ 3 Takte bis Ende des 1. VS<sub>SYNC</sub>

= 52995

# Interpretation der Simulationsergebnisse

Vergleich mit Ausgabe der `$monitor`-Anweisung



Da Zählung der Takte bei 0 anfängt: Takt Nr. **52994**

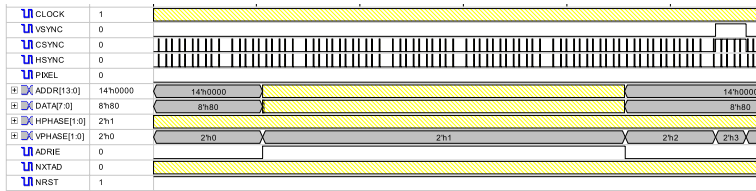
A. Koch

Takt=	50162	PIXEL=0	HSYNC=0	VSYNC=0
Takt=	50986	PIXEL=0	HSYNC=1	VSYNC=0
Takt=	51074	PIXEL=0	HSYNC=0	VSYNC=0
Takt=	51898	PIXEL=0	HSYNC=1	VSYNC=0
Takt=	51986	PIXEL=0	HSYNC=0	VSYNC=0
Takt=	52810	PIXEL=0	HSYNC=1	VSYNC=0
Takt=	52898	PIXEL=0	HSYNC=0	VSYNC=0
<b>Takt=</b>	<b>52994</b>	<b>PIXEL=1</b>	HSYNC=0	VSYNC=0
Takt=	52995	PIXEL=0	HSYNC=0	VSYNC=0
Takt=	53722	PIXEL=0	HSYNC=1	VSYNC=0
Takt=	53810	PIXEL=0	HSYNC=0	VSYNC=0
Takt=	53907	PIXEL=1	HSYNC=0	VSYNC=0

# Signalverlauf: Ein ganzes Bild



A. Koch

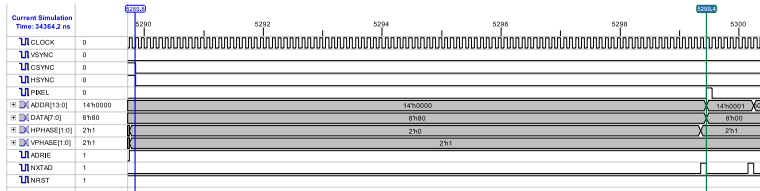


# Signalverlauf: Anfang einer Zeile

Präziser: 1. Zeile



A. Koch

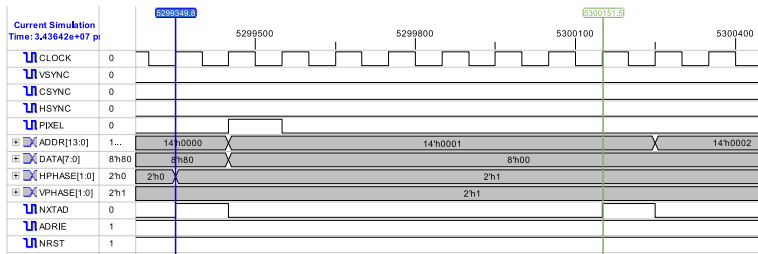


# Signalverlauf: Einzelne Pixel

Präziser: 1. Pixel der 1. Zeile



A. Koch

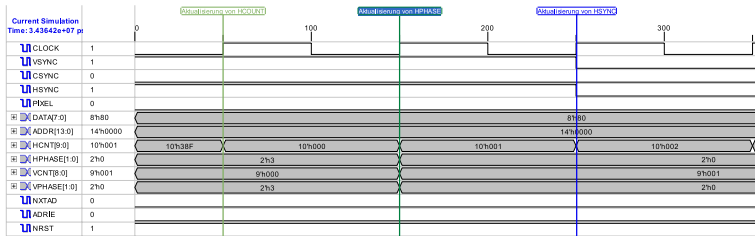


# Signalverlauf: Anfang des ersten Bildes

Diskussion der sequentiellen Verzögerungen



A. Koch





A. Koch

# Platzieren und Verdrahten



# Abbildung auf ein Xilinx XC3020 FPGA

... einen sehr alten (= sehr einfachen) Baustein



## DESIGN SUMMARY:

Part type=3020PC68-70

63 of 64 CLBs used

30 of 58 I/O pins used

...

50 CLB flipflops used

Xdelay: traced 490 paths, from 56 path sources.

Xdelay Report File:

Minimum Clock Period : 66.8 ns

Estimated Maximum Clock Speed : 15.0 Mhz

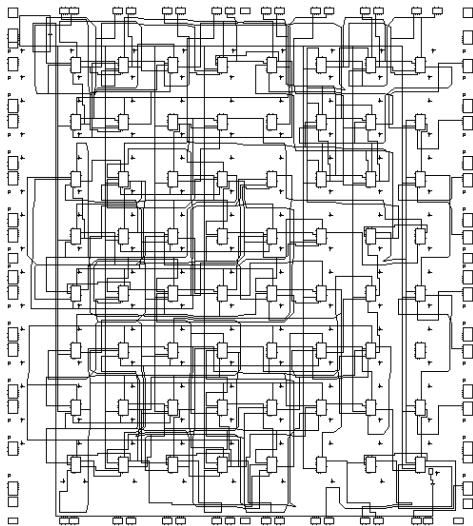
A. Koch

Hält Pixel-Takt für vorgegebene Auflösung ein!

# Layout



A. Koch

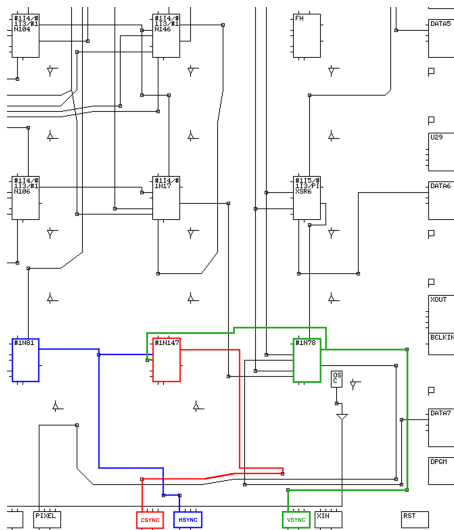


# Wiederfinden von Signalen und Logik

Hier **HSYNC**, **VSYNC** und Berechnung von **CSYNC**



A. Koch

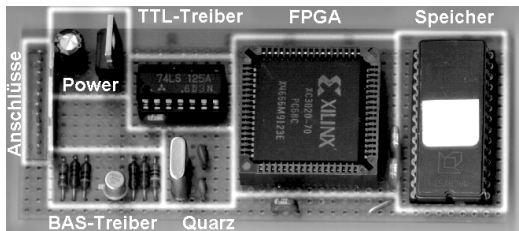




A. Koch

# Bildspeicher-ROM auf den Chip

# Bildspeicher



A. Koch

- Bisherige Annahme: **Externer** ROM-Chip
- Nur einfaches Verhalten simuliert
  - // Bildspeicherdaten auslesen: Hier kombinatorisches ROM!
  - always** @(ADDR) **begin**
  - DATA = MEM[ADDR];
  - end**
- Nun Idee: ROM direkt in DISCOUNT **integrieren**

# Schrittweises Vorgehen

Zunächst nur in **Simulation** behandeln



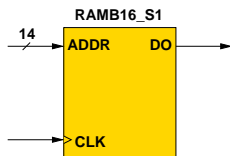
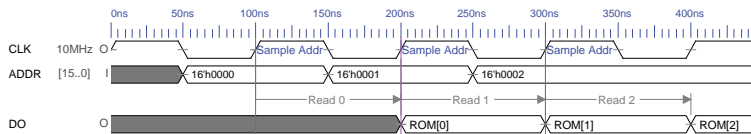
- Feld aus  $15360 \cdot 8 = 122880$  Registern?
- Nein, viel zu **ineffizient**
- Ersetze Pseudo-ROM durch **echte** Hardware-Modelle
- Werden vom FPGA/ASIC-Hersteller zur Verfügung gestellt
- Unsere **Anforderungen**
  - Platz für 15360 Bytes
  - 14b Adressen
  - 8b Datenausgang
- **Verfügbar** auf Ziel-FPGA
  - Platz für 16384 **Bits**
  - 14b Adressen
  - **1b** Datenausgang
- Passt nicht ganz ...

A. Koch

# Trotzdem genauer anschauen: RAMB16\_S1



A. Koch



- Adressen werden zur **steigenden** Flanke ausgewertet
  - Unterschied zum **kombinatorischen** Pseudo-ROM
- Ausgang **do** liefert adressierte Daten **nach nächster** steigender Flanke



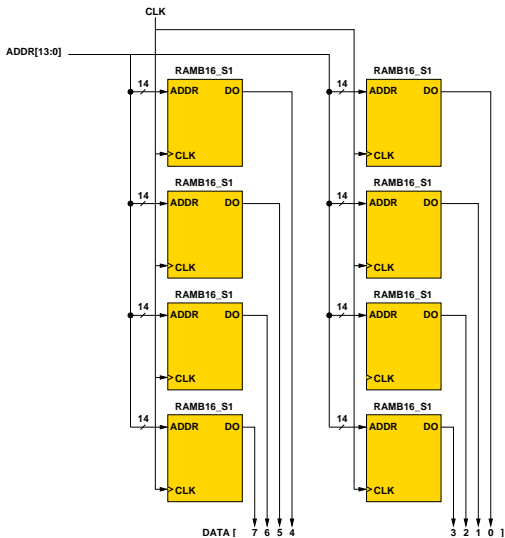
- Ein ROM-Block namens **RAMB16\_S1** ??
- Der Block kann noch mehr
- Kommt noch (→ Übungen)
- Hier wirklich nur als **ROM** benutzt
- Dazu verschiedene **zusätzliche** Steuersignale auf **feste** Werte legen
  - **EN=1' b1**, Block ist **immer** aktiv
  - **WE=1' b0**, **niemals** schreiben (soll ja ein **ROM** sein!)
  - **SSR=1' b0**, **kein** gesonderter Reset-Wert (Daten stehen im ROM)
- Aber nach wie vor
  - Ausgang **DO** ist nur **1b** breit, wir brauchen aber **8b**



# Verschaltung zum einem 16K x 8b ROM



A. Koch



# Timing: Pseudo-ROM ./.. echtes ROM



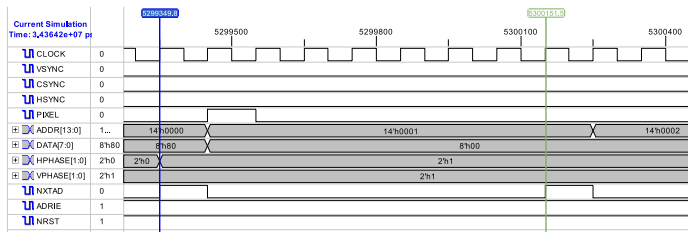
A. Koch

- Pseudo-ROM war **kombinatorisch**
- Hat **sofort** auf Änderungen der Adresse reagiert
- Echtes ROM ist **sequentiell**
- Reagiert erst bei nächster **positiver Flanke**
- Alles neu entwerfen??
- Nein, in diesem **speziellen** Fall nicht nötig
- Adresse liegt 8 Takte **vor** Übernehmen des Datums an
- **Genug Zeit**, damit echtes ROM seine Daten ausgibt

# Zeitverhalten Pseudo-ROM ./.. echtes ROM

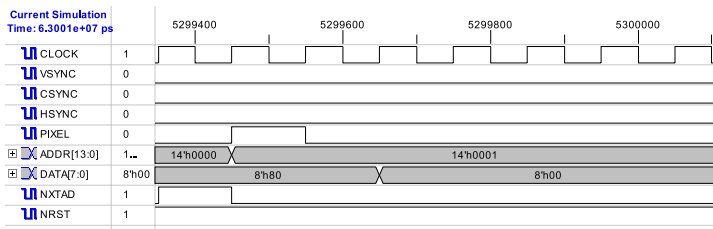


Pseudo-ROM: Änderungen an **ADDR** ändern **sofort** **DATA**



A. Koch

Echtes ROM: Änderungen an **ADDR** ändern **DATA verzögert**



# Einbindung des echten ROMs in Verilog

Kapseln innerhalb eines eigenen Moduls



A. Koch

```
// Bildspeicher aus Xilinx-ROM-Blöcken

#include "discount_defs.v"

module rom(
    input          CLK, // Clock
    input          NRST, // Reset#
    input  [*Asz-1:0] ADDR, // 14-bit Address Input
    output [*Bsz-1:0] DATA // 8-bit Data Output
);

    integer i;
```

# Erstellen des Feldes aus acht `RAMB16_S1`

Statt Copy & Paste schlaueres Konstrukt



```
// Instanziere 8 RAMB16_S1 namens INST[0].ROMBLOCK ... INST[7].ROMBLOCK
generate
  genvar j;
  for (j = 0; j < 8; j = j + 1) begin: INST
    RAMB16_S1 ROMBLOCK(
      .DO(DATA[j]), // bitweiser Anschluß
      .ADDR(ADDR),
      .CLK(CLK),
      .EN(1'b1),    // Enable, immer aktiv
      .WE(1'b0),    // Nie schreiben
      .SSR(1'b0)   // ROM nicht zuruecksetzen
    )
  end
endgenerate
```

A. Koch

- Erzeugt genau das erforderliche Feld
- Präfix (hier `INST`) vermeidet Namenskonflikte
- Nicht durch `for` ersetzbar
  - `for` kann keine **Instanzierungen** vornehmen
  - Sondern enthält nur **prozeduralen** Code

# Initialisieren der Bilddaten

Weisse Diagonale von links oben nach rechts unten



A. Koch

```
always @(negedge NRST) // <-- bei jedem Reset
// i: Index auf Byte, immer 8 Zeilen + 1 Byte weiter
for (i = 0; i < 'Msz; i = i + 'H/8*8 + 1) begin
// 7. ROM-Bank, Pixel ganz links im Byte
INST[7].ROMBLOCK.mem[i] = 1'b1;
// 6. ROM-Bank, 2. Pixel von links im Byte, 1 Zeile tiefer
INST[6].ROMBLOCK.mem[i + 'H/8] = 1'b1;
// 5. ROM-Bank, 3. Pixel von links im Byte, 2 Zeilen tiefer
INST[5].ROMBLOCK.mem[i + 'H*2/8] = 1'b1;
// usw.
INST[4].ROMBLOCK.mem[i + 'H*3/8] = 1'b1;
INST[3].ROMBLOCK.mem[i + 'H*4/8] = 1'b1;
INST[2].ROMBLOCK.mem[i + 'H*5/8] = 1'b1;
INST[1].ROMBLOCK.mem[i + 'H*6/8] = 1'b1;
INST[0].ROMBLOCK.mem[i + 'H*7/8] = 1'b1;
end
```



- Gleiches Verhalten wie Version mit Pseudo-ROM
- Vorbereitung für weitere Verbesserung
- **Unterschiedliche** Bilder
  - **RAM** statt ROM
  - Auf dem Chip, auch in **Synthese**



A. Koch

# Optimierung



# Untersuchung des Synthese-Ergebnisses

Speziell: Berechnung der **NXTAD** und **NXTLN**-Signale

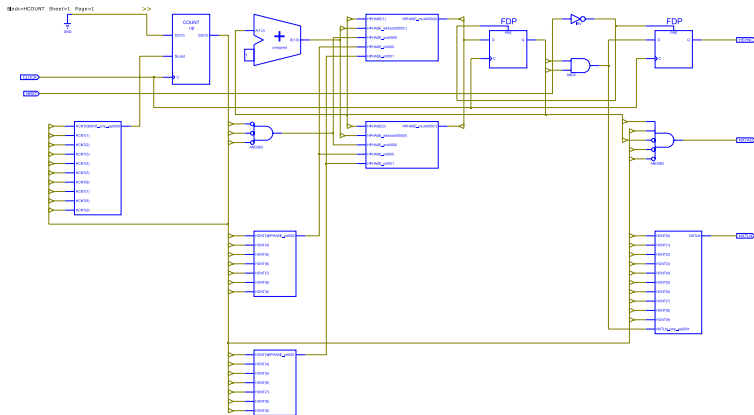


A. Koch

```
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
```

# Untersuchung des Synthese-Ergebnisses

## Speziell: Berechnung der **NXTAD** und **NXTLN**-Signale



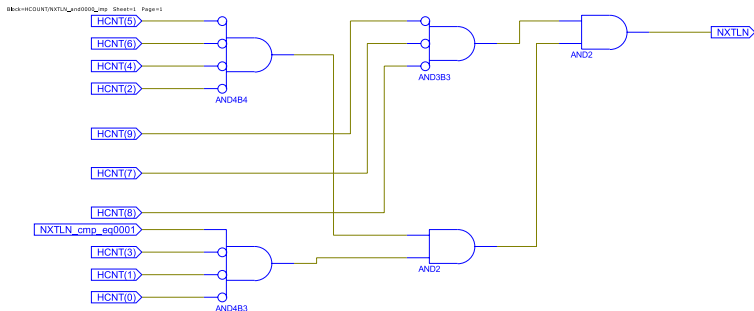
A. Koch

# Untersuchung des Synthese-Ergebnisses

Noch genauer: Berechnung des **NXTLN**-Signals



A. Koch



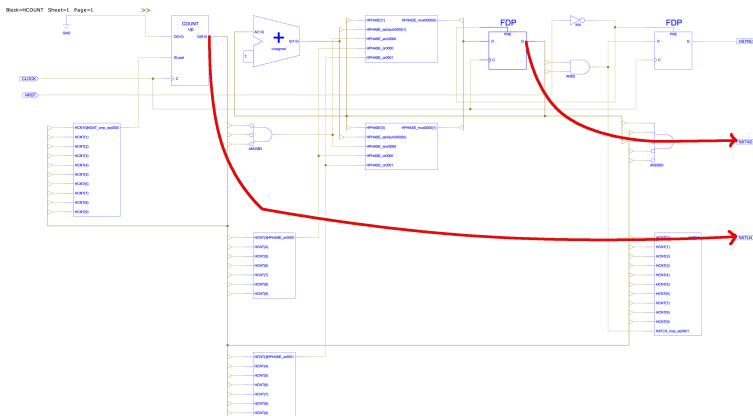
Durchläuft mindestens zwei Wertetabellen.

# Annahme: Signale sind zu langsam

Wegen Logik und Verdrahtungsverzögerungen



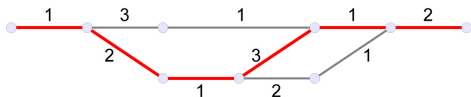
A. Koch



# Kritischer Pfad



A. Koch



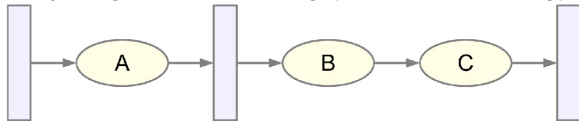
- Längste Verzögerung **zwischen** Registern bestimmt Taktfrequenz
- Idee: Pfad(e) **verkürzen**
- Verschiedene Ansätze

# Möglichkeiten der **Logik**-Synthese

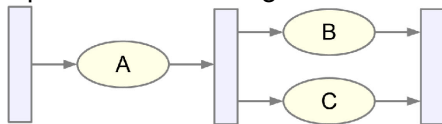


A. Koch

Ursprüngliche Schaltung (Pfad B-C zu lang)



Optimierte Schaltung



**Parallelisieren** von Berechnungen

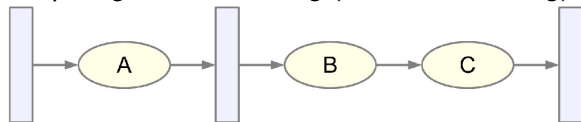
Beachte: Sequentielles Verhalten bleibt **unverändert**!

# Möglichkeiten der High-Level-Synthese

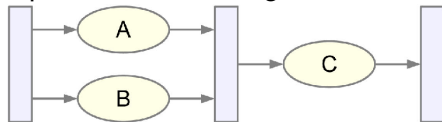
## 1. Verschieben von Berechnungen über Taktgrenzen



Ursprüngliche Schaltung (Pfad B-C zu lang)



Optimierte Schaltung



**Vorziehen** von Berechnungen **über** Taktgrenzen hinweg

Beachte: Sequentielles Verhalten bleibt **auch hier unverändert!**

A. Koch

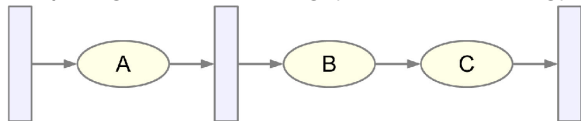
# Möglichkeiten der High-Level-Synthese

## 2. Einfügen von zusätzlichen Registern (Pipelining)

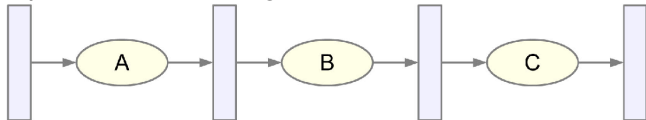


A. Koch

Ursprüngliche Schaltung (Pfad B-C zu lang)



Optimierte Schaltung



**Aufteilen** von langen Berechnungen **über** mehrere Takte  
Beachte: Sequentielles Verhalten wird nun **verändert!**



# Diskussion der Methoden für unser Problem

**NXTAD** und **NXTLN** seien zu langsam



A. Koch

- Möglichkeiten der **Logiksynthese** bereits ausgenutzt
    - ... das macht unser Synthesewerkzeug ja **automatisch**
    - Reicht hier aber nicht!
  - Methoden der **High-Level-Synthese** verwenden
    - ... nun aber **manuell**, die gibt das Werkzeug nicht her
  - Welche nehmen?
    - Wir wollen manuell einen möglichst **kleinen** Eingriff durchführen
    - Also versuche, sequentielles Verhalten der Signale **beizubehalten**
- ➔ **Verschieben** von Berechnungen!

# Verschieben von `NXTAD` und `NXTLN`



A. Koch

- Idee: Breche lange Pfade durch eingefügte **Register** auf
- Nun also `NXTAD` und `NXTLN` **nicht** mehr kombinatorisch
- ... sondern **sequentiell** berechnen
- **Problem** bei naivem Vorgehen (einfach Register einfügen)
  - Signale **verzögern** sich nach aussen um einen Takt
  - Der Rest der Schaltung funktioniert nicht mehr!
- Lösungsidee
  - Zwar **neue** Register einbauen
  - Aber Ergebnis einen Takt im **voraus** berechnen
  - Damit bleibt von **aussen** sichtbares sequentielles Verhalten gleich

# Berechnung von **NXTLN**

Alle Bedingungsabfragen um einen Takt vorziehen



## Ursprüngliche Berechnung

```
if ((HCNT == 0) && (HPHASE == 3))  
    NXTLN = 1;  
else  
    NXTLN = 0;
```

A. Koch

Nun **einen Takt** vorziehen

```
if ((HCNT == 'Hline-1'))  
    NXTLN = 1;  
else  
    NXTLN = 0;
```

Überlegung: Bei  $HCNT == 'Hline-1'$  ist **in jedem Fall** **HPHASE==3**  
Abfrage auf **HPHASE** unnötig!

# Berechnung von `NXTAD`



Ursprüngliche Berechnung

```
if ((HCNT % 'Bsz == 1) && (HPHASE == 1))  
    NXTAD = 1  
else  
    NXTAD = 0;
```

A. Koch

Nun **einen Takt** vorziehen

```
if ((HCNT % 'Bsz == 0) && (HPHASE == 1))  
    NXTAD = 1  
else  
    NXTAD = 0;
```

Reicht aber **nicht** aus: `HPHASE` wird erst zu **spät** 1!

# HPHASE eher berechnen

Idee: Phasengrenzen um einen Takt nach vorne schieben



## Ursprüngliche Berechnung

A. Koch

```
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
```

## Um einen Takt vorgezogenes Signal

```
case (HCNT)
  'Hline-1:  HPHASE <= 0;          // linker Bildrand
  'H1-1:     HPHASE <= HPHASE+1; // Bildpunkte
  'H1+'H-1:  HPHASE <= HPHASE+1; // rechter Bildrand
  'H1+'H+'H2-1: HPHASE <= HPHASE+1; // Bildsynchronisation
endcase
```

DISCOUNT funktioniert nicht: Nun kommt **HSYNC** zu früh!

# Berechnung von HSYNC



## Ursprüngliche Berechnung

```
always @(posedge CLOCK or negedge NRST) begin
  if (NRST == 0)    HSYNC <= 1;
  else
    if (HPHASE == 3) HSYNC <= 1;    // <-- nun zu früh!
    else            HSYNC <= 0;    //   -- " --
end
```

A. Koch

## Neu: Um einen Takt verzögertes HSYNC-Signal via HSYNC\_E

```
always @(posedge CLOCK or negedge NRST) begin
  if (NRST == 0) begin
    HSYNC    <= 1;
    HSYNC_E <= 1;
  end else begin
    if (HPHASE == 3) HSYNC_E <= 1;
    else            HSYNC_E <= 0;
    HSYNC <= HSYNC_E;
  end
end
```

Nun ist HSYNC selbst wieder richtig.

# Vorgezogene `NXTAD`-Berechnung durch eingefügte Register



A. Koch

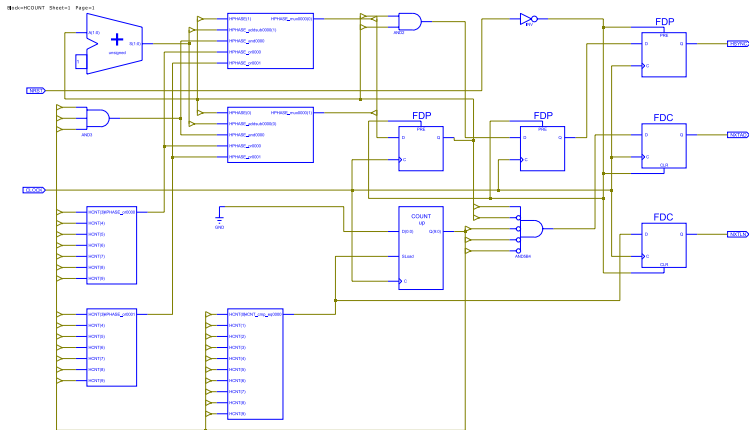
```
always @(posedge CLOCK or negedge NRST) begin
  if (NRST == 0) begin                               // Reset
    NXTAD <= 0;
    NXTLN <= 0;
  end else begin
    if ((HCNT % 'Bsz == 0 )                          // neues Byte
        && (HPHASE == 1))
      NXTAD <= 1;
    else
      NXTAD <= 0;
    if (HCNT == 'Hline-1)
      NXTLN <= 1;
    else
      NXTLN <= 0;
  end
end
```

Damit sequentielles Verhalten nach aussen **gleich** geblieben

# Synthese-Ergebnis der optimierten Schaltung



A. Koch





# Timing-Analyse der endgültigen Schaltung

Nach Platzierung und Verdrahtung auf modernem Chip (Xilinx XC2VP30)



## Vor Optimierung

Timing summary:

-----

Design statistics:

Minimum period : 3.898ns  
(Maximum frequency: 256.542MHz)

A. Koch

## Nach Optimierung

Timing summary:

-----

Design statistics:

Minimum period : 3.197ns  
(Maximum frequency: 312.793MHz)



A. Koch

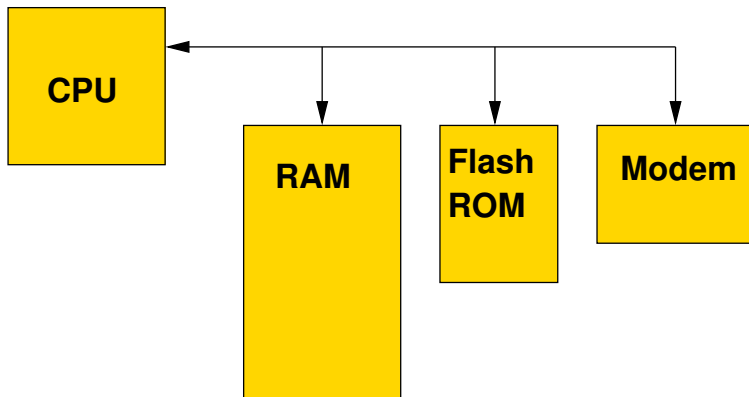
# Kommunikation und Adressierung

# Hardware-System



- Verschiedene Untereinheiten
- Kommunikation untereinander
- Auf verschiedene technische Weisen realisierbar

A. Koch



# Häufig verwendet: Busse



A. Koch

- Gemeinsame Verbindungen zwischen Komponenten
- Maximal ein **Initiator** / **Master** gleichzeitig
  - **Veranlasst** Aktivitäten auf Bus
- Ein oder mehrere **Targets** / **Slaves**
  - **Reagieren** auf Aktivitäten auf dem Bus
- Grundlegende **Transaktionen** auf dem Bus
  - Initiator fordert Daten von Target an: **Lesezugriff**
  - Initiator überträgt Daten zum Target: **Schreibzugriff**
- Busorganisation in **einfachen** Systemen
  - Nur **ein** Master, häufig die **CPU**
- In **leistungsstärkeren** Systemen
  - **Mehrere** Master (*multi-master* Ansatz, DMA)
  - Beispiele: Gb/s-Ethernet, Festplatten-Controller, Graphikprozessor, USB 2.0. . .
  - Hier nicht weiter behandelt

# Idee von Bussen



A. Koch

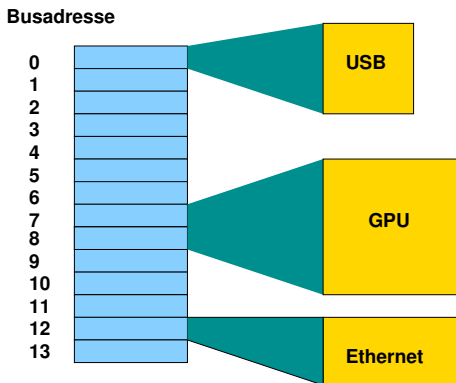
- **Gemeinsam** genutztes Medium
- Vorteile
  - **Einfache** Realisierung
  - Wenig **Chip-Fläche**
- Nachteile
  - Nur **eine** Verbindung gleichzeitig
    - Vereinfacht, kann **etwas** verbessert werden
    - → Disconnect/Reconnect, Split Transaction, ...
  - Probleme bei Bussen
    - Wie Bus vergeben, wenn **mehrere** Initiator/Master **gleichzeitig** Zugriff benötigen? → nicht weiter behandelt
    - Wie Target **gezielt** ansprechen? → kommt jetzt

# Busadressen



- Vergabe von **Adressen** für Teilnehmer an Bus
- Können auch **sehr** große Bereiche sein
  - Beispiel: Moderne GPU braucht ca. 800 MB an **Adressraum**

A. Koch



# Buszugriff von CPU als Master



A. Koch

- CPU **kennt schon** Adressen
- Nämlich für den **Speicher**
- Wie damit **Buszugriffe** realisieren?
- Zwei Alternativen

# 1. Getrennter Adressbereich für Bus



A. Koch

- Häufig **Ein-/Ausgabe-Adressbereich** genannt
- Verwendet z.B. bei X86-Architektur
- Spezielle **Instruktionen** für Ein-/Ausgabe
- Interpretieren angegebene Adresse immer als **Busadresse**
- Beispiel: `outb %a1, 0x60`
  - **Schreibt** Inhalt des **AL**-Registers (1 Byte) an Busadresse 0x60
- Beispiel: `inb 0x71, %a1`
  - **Liest** 1 Byte von Busadresse 0x71 in **AL** Register
- Beispiele für die klassische **PC-Architektur**
  - **Tastatur** liegt auf 0x60 und 0x64
  - **1. Serielle Schnittstelle** liegt auf 0x3f8-3ff
  - **1. Festplatten-Controller** liegt auf 0x1f0-1f7 und 0x3f6-3f7

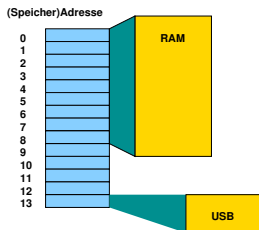


## 2. Gemeinsamer Adressbereich



A. Koch

- **Nachteil** des getrennten Adressbereichs
  - **Zusätzliche** Instruktionen
  - Ungünstig, Instruktionsbits sind **rares Gut**
- Anderer Ansatz: **Blende** Busadressen in **normalen** Adressraum ein: **memory mapped I/O**
  - So bei den meisten anderen Prozessoren
  - Auch bei X86 möglich, heute überwiegend verwendet
- Zugriff nun mit ganz normalen Lade/Speicherbefehlen
  - **lw \$t0, 7**: Lade 32b Wort aus **RAM-Speicher**
  - **sb \$t1, 13**: Gebe Byte auf **USB-Schnittstelle** aus



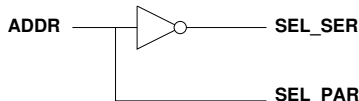
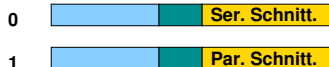
# Adressdekodierung



A. Koch

- Problem: Wie erkennen Busteilnehmer, ob sie **angesprochen** werden?
  - Also Target / Slave eines Zugriffs sind
- Gängige Technik
  - **Select**-Signal pro Teilnehmer
  - Wird **aktiviert**, wenn Teilnehmer vom Initiator/Master angesprochen wird
- **Adressdekodierung** übersetzt Busadressen in Select-Signale
- Triviales Beispiel: Zwei Teilnehmer auf Bus
  - Auf Adressen 0 und 1

Adresse



# Komplizierteres Szenario

Annahme: 64KB Adressraum



A. Koch

Startadresse	Endadresse	Teilnehmer
0x0000	0x1FFF	RAM 8KB
0x2000	0x23FF	Flash-Speicher 1KB
0xFE00	0xFFFF	Modem 512B

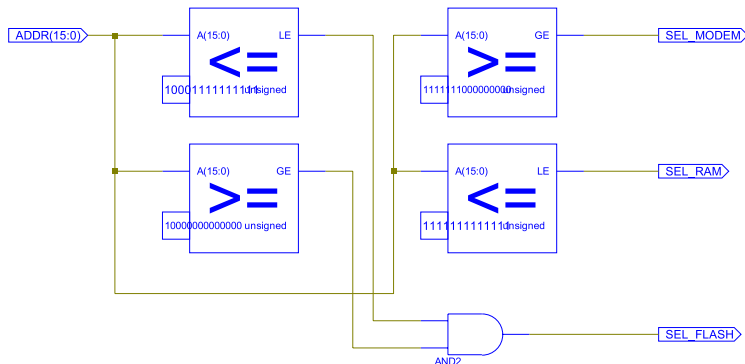
Wie realisierbar? Erster Versuch:

```
module decoder1 (  
  input [15:0] ADDR,  
  output      SEL_RAM, SEL_FLASH, SEL_MODEM  
);  
  
assign SEL_RAM      = (ADDR >= 16'h0000 && ADDR <= 16'h1FFF);  
assign SEL_FLASH    = (ADDR >= 16'h2000 && ADDR <= 16'h23FF);  
assign SEL_MODEM    = (ADDR >= 16'hFE00 && ADDR <= 16'hFFFF);  
  
endmodule
```

# Diskussion: Syntheseresult



A. Koch



- Optimiert: Immerhin nur **vier statt sechs** Vergleicher
- Jeder einzelne Vergleicher aber **groß und langsam!**

# Neue Überlegung



Müssen wir immer **alle** Bits vergleichen?

A. Koch

Startadresse	Endadresse	Teilnehmer
16'b0000_0000_0000_0000	16'b0001_1111_1111_1111	RAM 8KB
16'b0010_00 <b>0</b> 0_0000_0000	16'b0010_0011_1111_1111	Flash-Speicher 1KB
16'b1111_11 <b>1</b> 0_0000_0000	16'b1111_1111_1111_1111	Modem 512B

- **Nein**, versuche nur **Startadressen** eindeutig zu unterscheiden
- Mit möglichst **wenigen** Bits!
- Beginne mit **höchstwertigen** (=linken) Bits
  - Vermeide so falsche Erkennung von Adressbereichen
  - Gegenbeispiel: Verwende Bit 9, um Modem (=1) und Flash (=0) auseinanderzuhalten
  - Klappt nicht: Bei Zugriff auf RAM kann Bit 9 **auch** 1 werden!
  - Bereiche dürfen sich **nicht** überlappen
  - Niemals **gleichzeitig** mehr als ein Select-Signal aktiv

# Nächster Versuch ...

Konstruiere **Entscheidungsbaum** von links nach rechts



A. Koch

Startadresse	Endadresse	Teilnehmer
16'b0000_0000_0000_0000	16'b0001_1111_1111_1111	RAM 8KB
16'b0010_0000_0000_0000	16'b0010_0011_1111_1111	Flash-Speicher 1KB
16'b1111_1110_0000_0000	16'b1111_1111_1111_1111	Modem 512B

- $\text{ADDR}[15] == 1 \rightarrow$  Zugriff auf **Modem**
- $\text{ADDR}[15] == 0 \ \&\& \ \text{ADDR}[13] == 0 \rightarrow$  Zugriff auf **RAM**
- $\text{ADDR}[15] == 0 \ \&\& \ \text{ADDR}[13] == 1 \rightarrow$  Zugriff auf **Flash**

## Diskussion

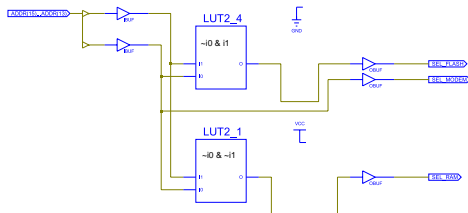
- **Überlappungsfrei**
- Select-Signal wird korrekt 1 bei Zugriff auf **Startadresse** und folgende Adressen
- Aber was ist mit der **Endadresse**?  $\rightarrow$  später!

# Verilog-Implementierung



```
module decoder2 (  
  input [15:0] ADDR,  
  output      SEL_RAM, SEL_FLASH, SEL_MODEM  
);  
  
  assign SEL_RAM    = ~ADDR[15] & ~ADDR[13];  
  assign SEL_FLASH  = ~ADDR[15] & ADDR[13];  
  assign SEL_MODEM  = ADDR[15];  
  
endmodule
```

A. Koch



# Dekodierung **innerhalb** eines Teilnehmers

Einfaches Beispiel 1KB Flash-ROM, organisiert als 1Kx8b



A. Koch

```
module rom1kx8 (  
  input      SELECT,  
  input [9:0] ADDR,  
  output [7:0] DATA  
);  
  
  reg [7:0] MEM [0:1023]  
  
  assign DATA = (SELECT) ? MEM[ADDR] : 8'bz;  
  
  initial begin          // einige Beispieldaten eintragen  
    MEM[0]   = 8'h42;  
    MEM[1]   = 8'h23;  
    ...  
    MEM[1022] = 8'h20;  
    MEM[1023] = 8'h07;  
  end  
  
endmodule
```



# Dekodierung **innerhalb** eines Teilnehmers

Komplexeres Beispiel: Modem-Steuerregister



A. Koch

```
module modem (
  input      CLOCK,
  input      SELECT,
  input [8:0] ADDR,
  input      WRITE,
  inout [7:0] DATA
);

  reg [7:0] baudrate;
  reg [1:0] parity;
  reg [7:0] inchar, outchar;

  assign DATA = (~SELECT | WRITE) ? 8'bz :
    ((ADDR==0) ? baudrate :
     (ADDR==1) ? {6'b0,parity} :
     (ADDR==2) ? inchar // <-- ADDR=2 liest Zeichen
     : 8'h42);          // <-- Default-Wert für Debugging

  always @(posedge CLOCK) begin
    if (WRITE)
      case (ADDR)
        0 : baudrate <= DATA;
        1 : parity  <= DATA[1:0];
        2 : outchar <= DATA;          // <-- ADDR=2 schreibt Zeichen
      endcase
  end

endmodule
```

# Anschluss an Bus

Am Beispiel des 1KB Flash-ROMs



A. Koch

```
module mysystem;
...

wire [15:0] ADDR;
wire [7:0] DATA;
wire      SEL_RAM, SEL_FLASH, SEL_MODEM;

// Adressdecoder
decoder2 DECODER (ADDR, SEL_RAM, SEL_FLASH, SEL_MODEM);

// Flash-ROM
rom1kx8 FLASH (SEL_FLASH, ADDR[9:0], DATA);

...
endmodule
```

# Verhalten des Flash-ROMs



A. Koch

Startadresse	Endadresse	Teilnehmer
16'b0000_0000_0000_0000	16'b0001_1111_1111_1111	RAM 8KB
16'b0010_0000_0000_0000	16'b0010_0011_1111_1111	Flash-Speicher 1KB
16'b1111_1110_0000_0000	16'b1111_1111_1111_1111	Modem 512B

Zugriff auf Busadresse	Select	Zugriff auf ROM-Adresse	Ausgabedatum	
16'h0000	0	10'h000	8'bz	Zugriff auf RAM-Bereich
16'hFE00	0	10'h200	8'bz	Zugriff auf Modem-Bereich
16'h2000	1	10'h000	8'h42	Zugriffe auf Flash-Bereich
16'h2001	1	10'h001	8'h23	
16'h23FE	1	10'h3FE	8'h20	
16'h23FF	1	10'h3FF	8'h07	
16'h2400	1	10'h000	8'h42	Hinter Flash-Ende!
16'h2401	1	10'h001	8'h23	
16'h27FE	1	10'h3FE	8'h20	
16'h27FF	1	10'h3FF	8'h07	
16'h2800	1	10'h000	8'h42	
16'h2801	1	10'h001	8'h23	

...

➔ Der Flash-Bereich **wiederholt** sich!

# Memory Aliasing



A. Koch

- Speicherbereich **wiederholt** sich
- Es sind aber immer die **gleichen** Daten
- Sichtbarkeit der gleichen *lokalen* Adressen an unterschiedlichen *Busadressen*: **Aliasing**
- Schadet in vielen Fällen **nicht**
  - Manchmal schon: **Hack** der Microsoft XBOX
  - Schreibzugriffe auf 80008008 werden **abgefangen**
  - Aber Fehler im **Adressdecoder** der Southbridge
  - Adressen sind auch **aliased** als 80008x08
  - Zugriffe **erlaubt** für  $x \neq 0$
  - So Zugriff auf "geheimen" **Boot-Code** möglich
  - → MIST Premature Unmap Attack
- Erlaubt aber sehr **einfache und schnelle** Adressdekodierung

# Zusammenfassung Adressdekodierung



A. Koch

- Zeige Busteilnehmern durch **Select-Signal** an, wenn sie angesprochen werden
- Select-Signale werden aus **Busadressen** erzeugt
- Adressbereiche müssen **überlappungsfrei** sein
- Gleichzeitig darf **maximal ein** Select-Signal aktiv sein
- Erstelle Adressdekodierlogik durch Aufbau eines **Entscheidungsbaumes**
  - Von höherwertigen zu niederwertigen Adressbits
  - Erfülle Anforderungen dabei durch Auswertung von möglichst **wenigen** Adressbits
- Dabei **darf** derselbe Adressbereich i.d.R. mehrfach auftauchen (aliasing)
- Er **muß** aber mindestens an den spezifizierten Adressen erreichbar sein



A. Koch

# Zusammenfassung

# Zusammenfassung



- DISCOUNT ist **software-ferne** Anwendung
- Demonstriert diverse Techniken
- Applikations-spezifische **Constraints**
- Hier: Minimale **Taktfrequenz** durch Auflösung vorgegeben
- Verwendung von **technologiespezifischen** Blöcken
- Hier: **Xilinx** ROM-Block
- **Optimierung** der Schaltung
- Jenseits der Fähigkeiten des **Logiksynthesewerkzeugs**
- Methoden der **High-Level-Synthese**
- Hier: **Vorziehen** von Berechnungen über Taktgrenzen
- Adressdekodierung
  - Wurde in Übungen schon angeschnitten (DISCOUNT mit RAM)

A. Koch