Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Diplomarbeit

Ramin Jeyrani-Mameghani

SoC-basierte Erprobungsplattform für CCD-Kameras mit Channel Link Interface

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Ramin Jeyrani-Mameghani

SoC-basierte Erprobungsplattform für CCD-Kameras mit Channel Link Interface

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Informations- und Elektrotechnik
Studienrichtung Informationstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Jürgen Reichardt
Zweitgutachter : Prof. Dr.-Ing. Bernd Schwarz

Abgegeben am 10. September 2009

Ramin Jeyrani-Mameghani

Thema der Diplomarbeit

SoC-basierte Erprobungsplattform für CCD-Kameras mit Channel Link Interface

Stichworte

CCD-Kamera, Channel Link, LVDS, Virtex-5 FPGA, ML507, MicroBlaze Prozessor, I²C, VGA

Kurzzusammenfassung

Diese Arbeit umfasst die Realisierung einer SoC-basierten Erprobungsplattform für CCD-Kameras mit Channel Link Interface. Der serielle Kameradatenstrom wird dem ML507 Evaluierungsboard im LVDS-Standard zugeführt, und in ISERDES-Blöcken parallelisiert. Der ebenfalls im LVDS-Standard zugeführte Pixeltakt wird für die Eintaktung der Datenbits in einer DCM-Konfiguration vervielfacht. Aus dem empfangenen Kameradatenstrom werden in einem VGA-Controller die für VGA-Monitore erforderlichen Synchronisationssignale erzeugt. Ein Multiplexer schaltet auf den Testbildgenerator um, der 4 horizontale Balken (Rot, Grün, Blau, Weiß) generiert. VGA-Controller und Testbildgenerator lassen sich mit Generics für beliebige Bildauflösungen konfigurieren. Mit der MicroBlaze-Konfiguration wird der Videcontroller CH7301C per I²C-Bus parametriert.

Ende des Textes

Ramin Jeyrani-Mameghani

Title of the paper

SoC-based evaluation platform for CCD cameras with Channel Link Interface

Keywords

CCD camera, Channel Link, LVDS, Virtex-5 FPGA, ML507, MicroBlaze processor, I²C, VGA

Abstract

Inside this report the realization of an evaluation platform for CCD cameras with Channel Link Interface is described. The data stream of the camera being sent in LVDS standard is fed to the ML507 Development Board where deserialization in ISERDES components takes place. The pixel clock, being also fed in LVDS standard will be multiplied in a DCM configuration in order to latch in the data bits. A VGA controller generates the needed synchronization signals for VGA monitors out of the continuous camera data stream. A multiplex switches between camera pictures and a test screen generator which generates 4 horizontal colour bars (red, green, blue, white). The video controller CH7301C is parameterized over the I²C-Bus of the MicroBlaze configuration.

End of text

Inhaltsverzeichnis

1 Einleitung

Im Rahmen des Forschungsprojektes **F**ahrerassistenz- und **Au**tonome **S**ysteme (FAUST) der Hochschule für Angewandte Wissenschaften Hamburg werden Prototypen für Ausweichassistenzsysteme, Fahrzeugregelungen, Stabilisierungsverfahren und Navigations- sowie Lokalisierungssysteme entwickelt und zusammen auf unterschiedlichen Plattformen integriert. Die Kfz-Industrie liefert hierfür durch ihre Forschungs- und Entwicklungsarbeiten im Bereich der Fahrerassistenzsysteme die Anregungen [5],[10]. Solche Systeme erfordern einerseits hohe Rechengeschwindigkeiten der Komponenten für die digitale Signalverarbeitung und andererseits geringe Stückkosten für die Massenproduktion. FPGA-basierte **S**ystem **o**n **C**hip-Plattformen (SoC) können diese gegensätzlichen Anforderungen mit parallelen DSP-Funktionselementen und integrierten Prozessoren bei niedrigen Taktfrequenzen erfüllen.

Im Bereich der autonomen Fahrzeugführung werden Informationen bezüglich der Umgebung aus Sensorendaten gewonnen. **C**harge **C**oupled **D**evice-Kameras (CCD) liefern dabei Aufnahmen von der unmittelbaren Umgebung. Das Fahrzeug kann mit aus Bilddaten extrahierten Merkmalen, wie beispielsweise Fahrbahnmarkierungen, entlang der Bahn geführt werden. Diese Aufgabe macht Kameras mit hohen Bildraten erforderlich, die die Aufnahme der Umgebung bei einer annehmbaren Fahrtgeschwindigkeit gestatten. Weiterhin muss die Hardwareplattform für die Echtzeitverarbeitung der Kameradaten ausreichend Performanz und Chipfläche für hohe Funktionalitätsimplementierungen bieten.

Diese Diplomarbeit befasst sich mit dem Aufbau einer SoC-basierten Erprobungsplattform für CCD-Kameras mit Channel Link Interface, auf der verschiedene Bildverarbeitungsalgorithmen des Forschungsprojektes FAUST implementiert und getestet werden können (vgl. Abbildung 1). Als Beispiel sei das Kalman-Filter zur Fahrspuridentifikation genannt (vgl. [1]). Der von der CCD-Kamera gelieferte Datenstrom wird in dedizierten Serien-Parallel-Umsetzern des zugrundeliegenden Virtex-5 FPGA deserialisiert und zu dem DVI-Videocontroller CH7301C des ML507 Evaluierungsboards geführt. Dieser übernimmt die Ansteuerung des VGA-Monitors, auf dem der kontinuierliche Kamera- bzw. durch Bildverarbeitungsalgorithmen manipulierte Bilddatenstrom zur visuellen Kontrolle dargestellt wird.

Die Schwerpunkte dieser Diplomarbeit lauten wie folgt:

1. Kopplung der CCD-Kamera Basler A301b mit dem Xilinx Evaluierungsboard ML507

2. Anpassen des vorhandenen VHDL-Channel Link Receiver-Moduls auf den Datenstrom der CCD-Kamera A301b

3. Entwurf eines flexibel parametrierbaren VGA-Controllers und Testbildgenerators für beliebige Bildauflösungen

4. Integration der VHDL-Module in ein MicroBlaze µController-System, das über ein I²C-Interface die Parametrierung des Videocontrollers CH7301C vornimmt

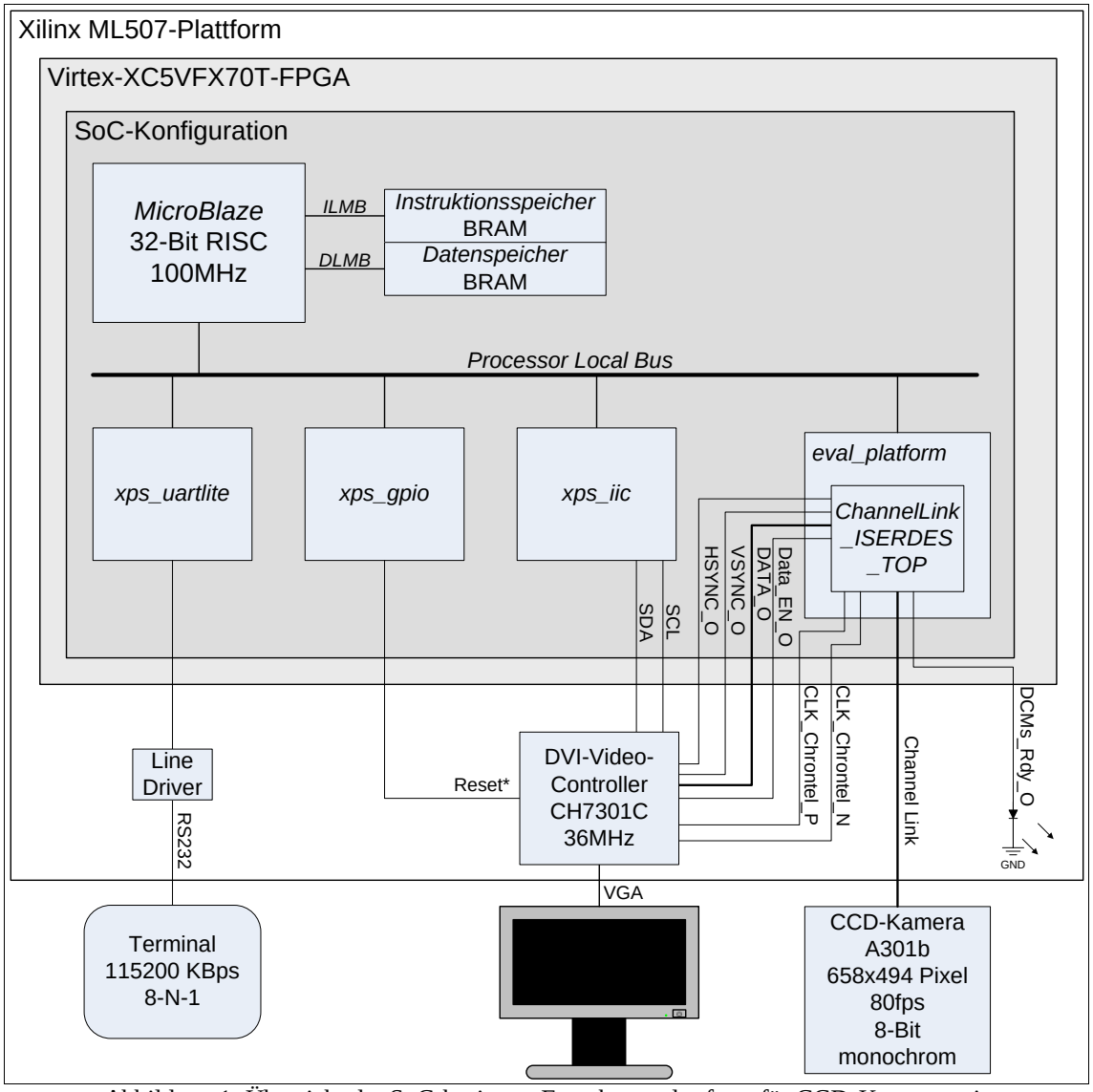

Abbildung 1: Übersicht der SoC-basierten Erprobungsplattform für CCD-Kameras mit Channel Link Interface

- Diese Arbeit beginnt mit der Vorstellung der der Erprobungsplattform zugrundeliegenden Hardwarekomponenten in Kapitel 2.

- In Kapitel 3 werden die für die Erprobungsplattform eingesetzten FPGA-Technologien und ein Verfahren vorgestellt, mit dem Laufzeitpfade von HDL-Designs durch feste Platzierung von FPGA-Ressourcen unverändert erhalten bleiben.

- In Kapitel 4 werden der Aufbau des seriellen Channel Link-Datenstroms und 2 verschiedene Implementierungen aufgezeigt, mit denen sich dieser deserialisieren lässt.

- Kapitel 5 beinhaltet den Aufbau des parametrierbaren VGA-Controllers, der sich für beliebige Bildauflösungen konfigurieren lässt und sich für die Darstellung kontinuierlicher Bilddatenströme eignet. Die Funktionalität des VGA-Controllers wird anhand von Simulationen und Oszilloskop-Messungen belegt.

- In Kapitel 6 wird der parametrierbare Testbildgenerator behandelt, der sich für beliebige Bildauflösungen konfigurieren lässt und sich für die Simulation von Kameradatenströmen eignet. Der Aufbau wird aufgezeigt und die Funktionalität durch Simulationen und Oszilloskop-Messungen belegt. Die Parametrierung des Testbildgenerators wird am Beispiel des Datenstroms der CCD-Kamera A301b aufgezeigt.

- In Kapitel 7 wird der aus Komponenten zuvor behandelter Kapitel zusammengesetzte Gesamtaufbau beschrieben. Zusätzlich wird das Ergebnis der Timing-Simulation zur Verifikation seines zeitlichen Verhaltens und der Gesamtressourcenbedarf aufgelistet.

- In Kapitel 8 wird die Implementierung des in Kapitel 7 behandelten Gesamtaufbaus in ein Peripheriemodul des Microblaze Soft-Mikroprozessorkerns erläutert. Zudem wird die Parametrierung des Videocontrollers CH7301C über den I²C-Bus erläutert und mit Messungen belegt.

2 Hardwarekomponenten der Erprobungsplattform

In diesem Kapitel werden die Hardwarekomponenten der angestrebten Erprobungsplattform für CCD-Kameras mit Channel Link Interface behandelt. Dazu wird in Kapitel 2.1 die für die Erprobungsplattform erforderliche Peripherie des ML507 Evaluierungsboards aufgelistet und in der Funktion erläutert. In Kapitel 2.3 und 2.4 werden 2 unterschiedliche Kameras mit Channel Link Interface vorgestellt. In Kapitel 2.5 werden beide Kameras in Bezug auf ihre Konfigurationsfähigkeit mit dem ML507 Evaluierungsboard miteinander verglichen.

2.1 ML507 Evaluierungsboard mit Virtex-5 FPGA

Das ML507 Evaluierungsboard bildet die Hardwaregrundlage der Erprobungsplattform für CCD-Kameras mit Channel Link Interface (vgl. Abbildung 2, [25]).

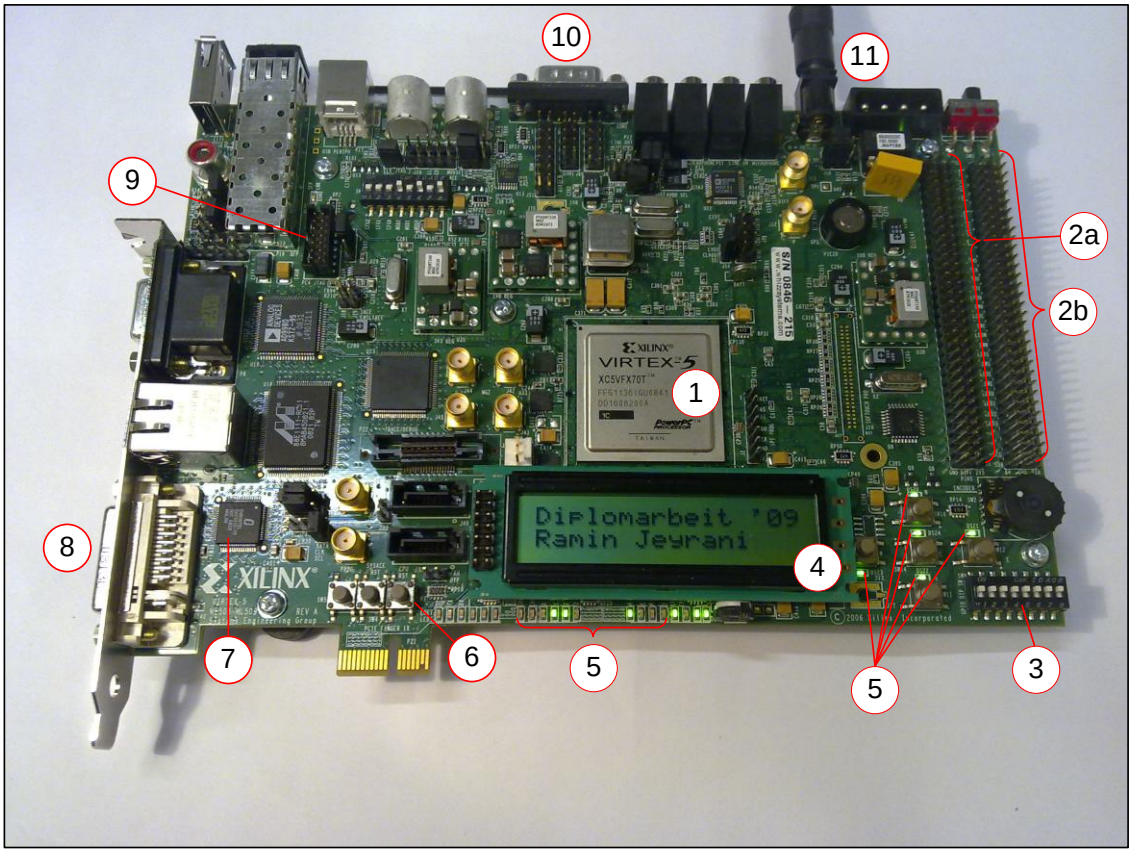

Abbildung 2: ML507 Evaluation Platform

Das ML507 Evaluierungsboard ist unter anderem mit folgender Peripherie bestückt (vgl. [35]):

(1) Virtex-5 FPGA XC5VFX70T, FFG1136, Speed Grade -1

Das XC5VFX70T FPGA entstammt der Virtex-5 FPGA-Reihe und setzt sich aus folgendem zusammen (vgl. [33]):

- 11.200 Slices,
- 71.680 Logic Cells,
- 820-Kbit Distributed RAM,
- 148*36-Kbit (= 5.328-Kbit) Block RAM,
- 12 **D**igital **C**lock **M**anagers (DCMs),
- 6 **P**hase **L**ocked **L**oops (PLLs)/**P**hase **M**atched **C**lock **D**ividers (PMCDs),
- 640 Single Ended Pins,
- 320 Differential I/O Pairs.

(2a) Xilinx Generic Interface (XGI) Expansion Headers *J4* und *J7*

An den insgesamt 32Pins der Pinleiste *J7* (rechte Reihe) liegen 2.5V DC für die Versorgung externer Peripherie etc. an. Die Pinleiste *J4* besteht aus 2 Pinreihen: an der linken liegt Ground an, die rechte Reihe führt zu 16 differentiellen Ein-/Ausgangspaaren des FPGAs, die beispielsweise für LVDS verwendet werden können (Anhang A1).

(2b) XGI Expansion Headers *J5* und *J6*

Die Pinleiste *J5* (linke Reihe) bietet externer Peripherie neben 5V DC und 3,3V DC Versorgungsspannung und IIC Bus- und JTAG chain-Anbindung noch die Verwendung einiger LEDs und Taster des Boards. An der linken Pinreihe von *J6* liegt Ground an, die insgesamt 32 Pins der rechten Pinreihe lassen sich als universelle FPGA-Ein-/Ausgänge nutzen.

(3) DIP-Schalter

Über die insgesamt 8 DIP-Schalter wird zwischen High- und Low-Pegel umgeschaltet.

(4) 16-Character x 2-Line LCD

Das zweizeilige Character-LCD dient der Ausgabe von 16 Zeichen je Zeile (vgl. [29]). Dies ermöglicht die Anzeige von Statusinformationen etc. Das LCD wird über einen 4-Bit-breiten Daten- und 3-Bit-breiten Steuerpfad betrieben (vgl. [21]).

(5) User LEDs

Die USER LEDs werden Active-High angesteuert und lassen sich für beliebige Zwecke einsetzen.

(6) (CPU Reset) Pushbutton

Dieser Active-Low-Taster schaltet bei Betätigung auf Low-Pegel. Der Taster ist nicht prellfrei.

(7) Videocontroller CH7301C

Mit dem Videocontroller CH7301C lassen sich analoge VGA- sowie **D**igital **V**isual **I**nterface-Monitore (DVI) ansteuern (vgl. Kapitel 2.2). Die Parametrierung des Videocontrollers erfolgt über den I²C-Bus, (vgl. Kapitel 8).

(8) DVI-I Interface

Über das DVI-I Interface lassen sich analoge sowie digitale Bilddaten übertragen (vgl. [20]). Analoge VGA-Monitore mit 15-poligem Mini-D-Sub-Stecker können über einen DVI-VGA-Adapter an das DVI-Interface angeschlossen werden.

(9) JTAG Configuration Port

Über den JTAG Configuration Port lässt sich die FPGA-Plattform konfigurieren und Debugging durchführen.

(10) RS-232 Serial Port

Die serielle Schnittstelle dient der Kommunikation zwischen FPGA-Plattform mit externer Hardware. In einer MicroBlaze-Konfiguration lässt sich diese Schnittstelle als Standard-aus- (stdout) und –eingabe (stdin) verwenden. Ein Line Driver-Chip sorgt für die Anpassung der Spannungspegel zwischen FPGA- und RS-232-Signalen. Die Schnittstelle ist als Host ausgelegt und erfordert daher ein Nullmodem-Kabel für die Verbindung mit beispielsweise einem PC.

(11) Input Power plug/connector

Die Energieversorgung der FPGA-Plattform kann über 2 verschiedene Wege erfolgen:

- 2,1mm*5,5mm Buchse (mitgeliefertes Netzteil)
- PC-Laufwerkskonnektor für den direkten Betrieb des ML507 Evaluierungsboards in einem PC-Gehäuse.

2.2 Kennwerte des DVI-Videocontrollers CH7301C

Der Videocontroller CH7301C ist ein Controllerchip, mit dem sich digitale sowie analoge Monitore ansteuern lassen (vgl. [8]). Digital zugeführte Eingangssignale werden im **T**ransition **M**inimized **D**ifferential **S**ignaling- (TMDS) und VGA-Standard herausgeführt. Das Blockschaltbild des Controllers ist in Abbildung 3 dargestellt. Das Interface des CH7301C besteht aus den folgenden Ein- und Ausgängen:

- 12-Bit-breiter Dateneingang (*D[11:0]*)
- differentieller Takteingang (*XCLK, XCLK**)
- Data Enable-Signal (*DE*)
- horizontale und vertikale Synchronisationseingangssignale (*H, V*)

- 3 differentielle Datenausgänge (*TDC0…2*, *TDC0*…2**)
- differentieller Taktausgang (*TLC*, *TLC**)
- 3 analoge Datenausgänge (*DAC2…0*)
- horizontale und vertikale Synchronisationsausgangssignale (*H SYNC*, *V SYNC*)
- Active-Low-Reset-Eingang (*Reset**)
- Takt- und Datenleitung für die I²C-Busanbindung zur Parametrierung des Videocontrollers (*SPC*, *SPD*)

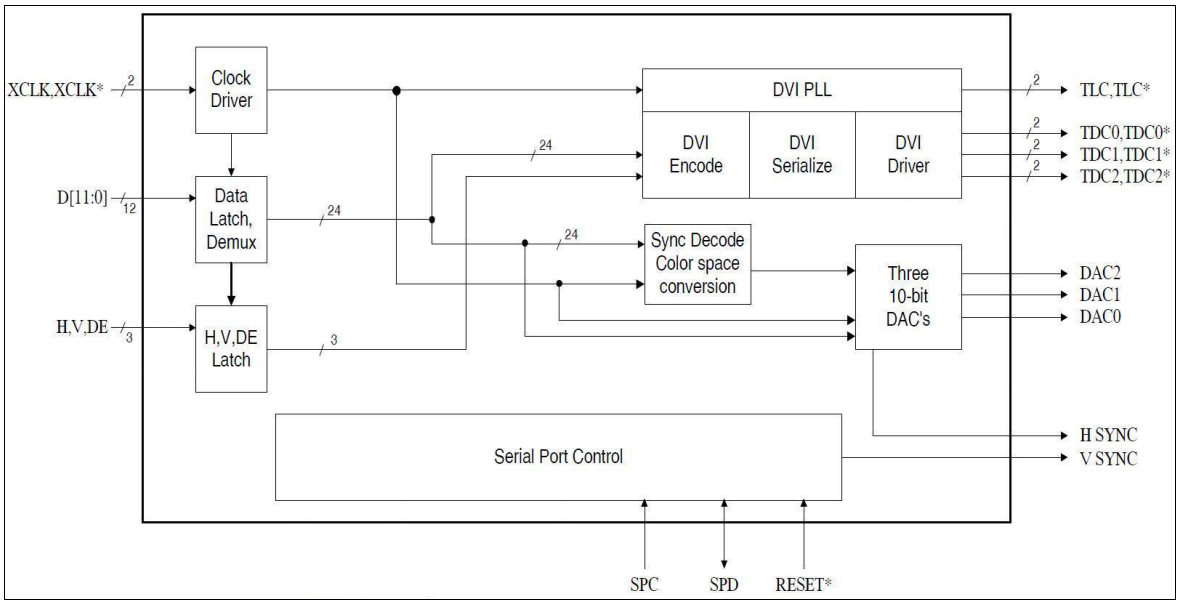

Abbildung 3: Blockshaltbild des Videocontrollers CH7301C

Der Videocontroller kennzeichnet sich durch folgende Kennwerte:

- 15-/16-/24-Bit Farbtiefe
- RGB-/YCrCb-Dateneingangsformat
- Farbraumkonversion
- minimaler Pixeltakt f_{pmin} = 25MHz
- maximaler Pixeltakt f_{pmax} = 165MHz
- PLL für die Taktvervielfachung
- 3* 10-Bit-Digital-Analog-Umsetzer ($1/f_{pmax}$ = 6,06ns minimale Umsetzzeit)
- Parametrierung per I²C-Interface (vgl. Kapitel 8.2)
- Single/Double Data Rate-Eintaktung der Pixeldaten

Jedes aus 3*8-Bit bestehende Pixel wird dem Videocontroller, nach einem bestimmten Dateneingangsformat sortiert (vgl. Tabelle 1), im **D**ouble **D**ata **R**ate-Modus (DDR) zugeführt (vgl. Abbildung 4). Bei steigender Flanke des Pixeltaktes ($XCLK_{(1x)}$) werden die ersten 12-Bit (*P0a*), bei fallender Flanke die restlichen 12-Bit (*P0b*) der Pixeldaten an den Eingang des Videocontrollers CH7301C angelegt. Alternativ zum DDR-Modus lassen sich die Pixeldaten bei jeder steigenden Flanke des doppelten Pixeltaktes ($XCLK_{(2x)}$) im **S**ingle **D**ata **R**ate-Modus (SDR) an den Eingang des Videocontrollers CH7301C anlegen.

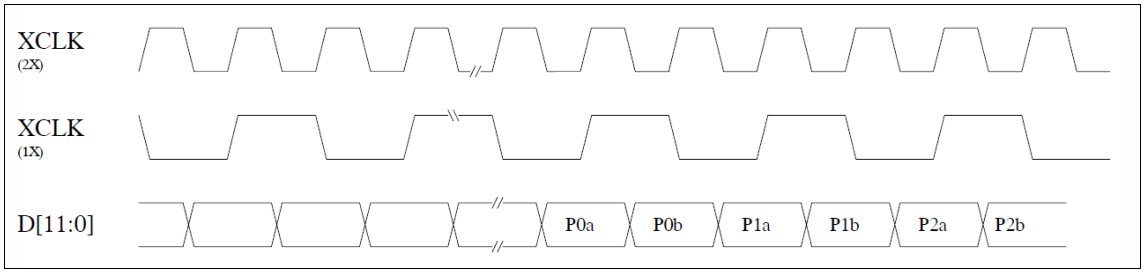

Abbildung 4: Verlauf der Datenzuführung zum Videocontroller CH7301C

	P0a	P0b	P1a	P1b	…
D[11]	GRN0[3]	RED0[7]	GRN1[3]	RED1[7]	…
D[10]	GRN0[2]	RED0[6]	GRN1[2]	RED1[6]	…
D[9]	GRN0[1]	RED0[5]	GRN1[1]	RED1[5]	…
D[8]	GRN0[0]	RED0[4]	GRN1[0]	RED1[4]	…
D[7]	BLU0[7]	RED0[3]	BLU1[7]	RED1[3]	…
D[6]	BLU0[6]	RED0[2]	BLU1[6]	RED1[2]	…
D[5]	BLU0[5]	RED0[1]	BLU1[5]	RED1[1]	…
D[4]	BLU0[4]	RED0[0]	BLU1[4]	RED1[0]	…
D[3]	BLU0[3]	GRN0[7]	BLU1[3]	GRN1[7]	…
D[2]	BLU0[2]	GRN0[6]	BLU1[2]	GRN1[6]	…
D[1]	BLU0[1]	GRN0[5]	BLU1[1]	GRN1[5]	…
D[0]	BLU0[0]	GRN0[4]	BLU1[0]	GRN1[4]	…

Tabelle 1: Dateneingangsformat bei IDF=0

2.3 CMOS-Kamera MV-D750E-20-CL mit Channel Link Interface

Die MV-D750E-20-CL ist eine **C**omplementary **M**etal-**O**xide **S**emiconductor-Kamera (CMOS). Sie weist u.a. folgende Kennwerte auf (vgl. [31]):

- CMOS-Monochrombildsensor
- 750x400 Pixel/Bild
- bis zu 60 Vollbilder/s
- 20MHz Pixeltakt
- 8-/10-Bit Graustufenauflösung
- Base Configuration des Camera Link Interface (Channel Link)
- 26-poliger **M**ini **D**-**R**ibbon-Konnektor (MDR, vgl. [6])

Das MV-D750E-20-CL verfügt über das Camera Link Interface in der Base Configuration: Bilddaten und Pixeltakt werden nach dem Channel Link-Protokoll übertragen (vgl. Anhang A1 → „Channel Link-Protokoll“). Zusätzlich steht ein Kamera-Kontrollsignal als externer Auslöseimpuls und die serielle Schnittstelle für die Kommunikation mit der Kamera zur Verfügung (vgl. Anhang A1 → „Camera Link Interface“). Die Kamera verfügt über einen in der Camera Link-Spezifizierung festgelegten MDR-Konnektor, über den die Verbindung zu Frame Grabbern hergestellt wird (vgl. [6]). Die Spannungsversorgung der Kamera erfolgt über einen Binder Subminiatur Rundsteckverbinder der Serie 712 (vgl. Abbildung 5, [4]).

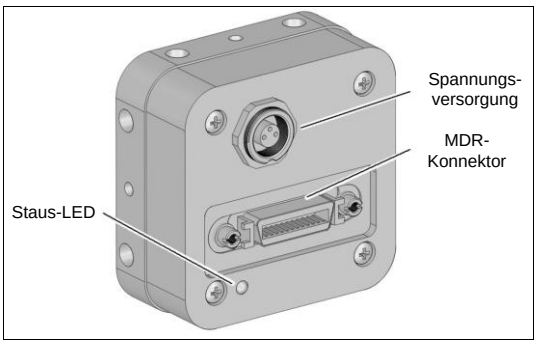

Abbildung 5: Rückansicht der CMOS-Kamera MV-D750E-20-CL

2.4 CCD-Kamera A301b mit Channel Link Interface

Die A301b ist eine **C**harge **C**oupled **D**evice-Kamera (CCD). Sie weist unter anderem folgende Kennwerte auf (vgl. [3]):

- CCD-Monochrombildsensor
- 658x494 Pixel/Bild
- bis zu 80 Vollbilder/s
- 18MHz Pixeltakt
- 36MHz effektiver Pixeltakt (2 Pixel je 18MHz-Takt)
- 8-/10-Bit Graustufenauflösung
- Base-Konfiguration des CameraLink Interface (Channel Link)
- 26-poliger D-Sub-Konnektor

Die A301b verfügt über das Camera Link Interface in der Base Configuration: Bilddaten und Pixeltakt werden nach dem Channel Link-Protokoll übertragen. Zusätzlich steht ein Kamera-Kontrollsignal als externer Auslöseimpuls und die serielle Schnittstelle für die Kommunikation mit der Kamera zur Verfügung (vgl. Anhang A1). Die Kamera verfügt über einen für Camera Link-Anwendungen nicht spezifizierten D-Sub-Konnektor (vgl. Abbildung 6). Ein herstellerspezifisches Adapterkabel führt MDR-, RS232- und Spannungsversorgungs-Konnektoren heraus.

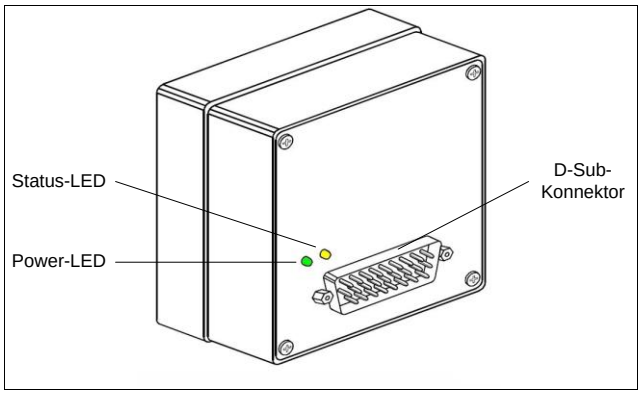

Abbildung 6: Rückansicht der CCD-Kamera A301b

2.5 Konfigurationsfähigkeit der Kameras A301b und MV-D750E-20-CL mit dem ML507 Evaluierungsboard

In diesem Kapitel werden die Kameras A301b und MV-D750E-20-CL in Bezug auf ihre Konfigurationsfähigkeit mit dem ML507 miteinander verglichen. Da beide Kameras über das standardisierte Channel Link Interface verfügen, ist die Bildsensor-Technologie (CCD bzw. CMOS) für die Konfigurationsfähigkeit mit dem ML507 irrelevant, einzig der bei beiden Kameras unterschiedliche Pixeltakt ist entscheidend (vgl. Tabelle 2).

Der aus dem Channel Link Interface eintreffende Pixeltakt wird in DCM-Blöcken vervielfacht, um unter anderem die seriellen Datenbits des Kameradatenstroms einzutakten (vgl. Kapitel 3.2). Des Weiteren werden die Kamerabilder dem Videocontroller CH7301C pixelweise (bei Pixeltakt) zugeführt, um diese auf einem VGA-Monitor darzustellen (vgl. Kapitel 5).

Der effektive Pixeltakt der CCD-Kamera A301b liegt mit 36MHz im Eingangsfrequenzbereich des Videocontrollers CH7301C (vgl. [8], Kapitel 2.4). Sein realer Pixeltakt hingegen liegt mit 18MHz außerhalb des Eingangstaktfrequenzbereichs der Viretx-5-DCM-Blöcke (vgl. Tabelle 2), wodurch die Funktionalität der DCM laut Datenblatt nicht gewährleistet werden kann [34]. Im praktischen Versuch konnte die CCD-Kamera A301b in Verbindung mit der in Kapitel 4.3.1 beschriebenen DCM-Konfiguration *DCM_18_TO_36_72_126* dennoch in Betrieb genommen werden. Allerdings ist eventuell ein mehrmaliges Reset durchzuführen, da die DCM-Blöcke nicht immer korrekt einschwingen.

Der Pixeltakt der CMOS-Kamera MV-D750E-20-CL liegt innerhalb des Eingangstaktfrequenzbereichs der Viretx-5-DCM-Blöcke, unterschreitet aber die minimale Eingangstaktfrequenz des Videocontrollers CH7301C und ist somit nicht für die Konfiguration mit dem ML507 Evaluierungsboard geeignet (vgl. [8], Tabelle 2).

		Eingangstaktfrequenzbereich	
Kamera	Pixeltakt in MHz	CH7301C	DCM-Blöcke des Virtex-5
MV-D750E-20-CL	20	25…165MHz	19…450MHz
A301b	18		

Tabelle 2: Kamera-Pixeltakt gegenüber Eingangsfrequenz von Videocontroller CH7301C und Virtex-5-DCM-Blöcken (Speed Grade -1)

Aufgrund oben genannter Ergebnisse aus dem Vergleich beider Kameras wurde die CCD-Kamera A301b im weiteren Verlauf dieser Arbeit verwendet.

3 FPGA-Technologie

In diesem Kapitel werden die in dieser Arbeit eingesetzten FPGA-Technologien der Virtex-5 FPGA-Reihe erläutert. In Kapitel 3.1 werden die differentiellen Eingangspins für die Einspeisung differentieller Channel Link Interface-Signale und in Kapitel 3.2 die DCM-Blöcke zur Taktvervielfachung des aus dem Channel Link Interface eintreffenden Pixeltaktes beschrieben. In Kapitel 3.3 werden die dedizierten Serien-Parallel-Umsetzer der Virtex-5 FPGA-Reihe für die Deserialisierung des seriellen Datenstroms von CCD-Kameras beschrieben. In Kapitel 3.4 wird letztendlich ein Verfahren aufgezeigt, mit dem sich FPGA-Ressourcen fest platzieren lassen, um gleichbleibende Laufzeitpfade trotz Modifikation bestehender HDL-Designs zu erhalten.

3.1 Differential I/O Pins

Das ML507 Evaluierungsboard verfügt über 16 differentielle Eingangs-/Ausgangs-Pinpaare, die für Signale mit hoher Datenrate ($f_{max\_Input} = 1/T_{IOPI} \approx 943MHz$, $f_{max\_Output} = 1/T_{IOOP} \approx 694MHz$) genutzt werden können (vgl. [34]). Für differentielle Daten- und Takteingänge sind die Librarymodule `IBUFGDS` bzw. `IBUFDS` im VHDL-Code der Topentity zu instanziieren. Die elektrischen Schnittstellenparameter für den 100Ω Abschlusswiderstand (`DIFF_TERM`) und den Schnittstellenstandard (`IOSTANDARD`) werden mit generics festgelegt (vgl. [37]). Diese Parametrierungen können im UCF-File überschrieben werden. Nachfolgend ist der VHDL-Code für einen differentiellen Dateneingang im LVDS-Standard abgebildet (vgl. Anhang A1):

```
DIFFERENTIAL_SIGNAL_INPUT: IBUFDS
  generic map(IOSTANDARD => "LVDS_25", DIFF_TERM => TRUE)
  port map(I => DIFF_SIG_I_P, IB => DIFF_SIG_I_N, O => SINGLE_SIG_O);
```

Positive (`I`) und negative (`IB`) Signalleitungen werden dem Buffer (`IBUFDS`) im LVDS-Standard (`"LVDS_25"`) zugeführt. Am Ausgang des Buffers wird das Signal als Einzelleitung (`O`) herausgeführt.

Die Zuführung differentieller Taktsignale an das ML507 ist neben der Wahl des korrekten Buffers an weitere Bedingungen geknüpft. Taktsignale sollten stets an ***C**lock **C**apable*-Pins (*CC*) geführt werden. Diese dedizierten Taktleitungen versorgen ihre Taktregion über spezielle Hardwareleitungen mit dem angelegten Taktsignal (vgl. Abbildung 7). Liegen Takteingangspins und die damit zu versorgende Taktregion im FPGA räumlich weit auseinander, sollten die Taktsignale an ***G**lobal **C**lock*-Pins (*GC*) geführt werden. Diese haben die Funktion, sämtliche Bereiche des FPGAs mit dem angelegten Taktsignal, bei geringer Clock Skew und Verzerrung des Tastverhältnisses, zu versorgen (vgl. [35]). Der Softwareprozess *ISE Map* der Entwicklungsumgebung Xilinx ISE v10.1.03 überprüft die genannten Bedingungen und bricht den Implementiervorgang bei Nichteinhaltung mit der Fehlermeldung `ERROR:Place:645` ab (vgl. [39]).

An der differentiellen Pinreihe *J4* des ML507 werden lediglich *CC*-Pins herausgeführt (vgl. Kapitel 2.1). Die DCM-Blöcke des ML507 wiederum befinden sich in anderen Takt-

regionen als die differentiellen Pins der Pinreihe *J4* (vgl. Abbildung 7). Aufgrund des Verdrahtungsweges eines differentiellen Eingangspinpaares mit der DCM entsteht eine gegenüber der Verwendung von *GC*-Pins erhöhte Signalverzögerung, die sich allerdings durch *Phase Shifting* kompensieren lässt (vgl. Kapitel 3.2). Durch den Eintrag `CLOCK_DEDICATED_ROUTE` im UCF-File lässt sich der Implementiervorgang fortzusetzen:

```
NET DIFF_CLK_I_P  LOC = PIN_LOCATION | CLOCK_DEDICATED_ROUTE = FALSE;
```

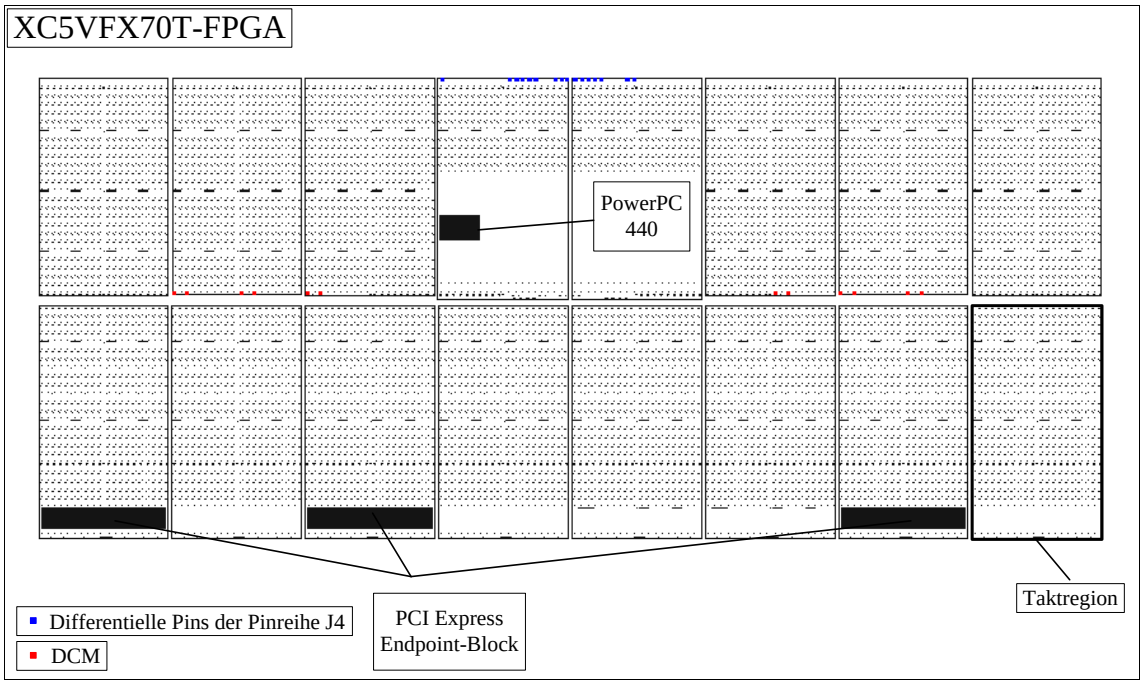

Abbildung 7: 16 quadratische Taktregionen des Virtex-5 FPGA XC5VFX70T

3.2 Digital Clock Manager - DCM

In vielen Anwendungen der Digitaltechnik werden mehrere Taktsignale genutzt, um gekoppelte Funktionen mit unterschiedlichen Datenraten betreiben zu können: Für die Eintaktung des aus dem Channel Link Interface eintreffenden seriellen Datenstroms ist das Vielfache (7X) des Pixeltaktes, der Bittakt, erforderlich (vgl. Kapitel 4). Der Bittakt wird in sogenannten DCM-Blöcken generiert (vgl. [35]).

Die DCM-Blöcke der Virtex-5 FPGA-Reihe, *DCM_ADV* genannt, befinden sich in den sogenannten **C**lock **M**anagement **T**iles (CMT). Jedes CMT verfügt über einen Phase Locked Loop- (PLL) und 2 DCM-Blöcke (vgl. Abbildung 8).

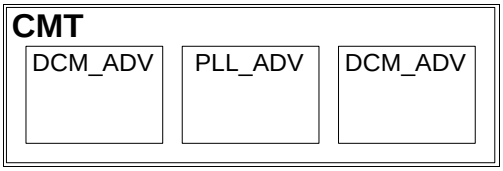

Abbildung 8: CMT der Virtex-5 FPGA-Reihe

Mit dem *PLL_ADV*-Block lassen sich Laufzeitdifferenzen auf Taktleitungen eliminieren und Ausgangstaktsignale unterschiedlicher Frequenz und Phasenlage generieren. Die Phasenverschiebung der Ausgangssignale ist in grobe Schritte unterteilt, sodass eine Feinjustierung der Phasenverschiebung wie bei der DCM nicht möglich ist (siehe weiter unten). Aus diesem Anlass kommen PLLs für den Einsatz in dieser Arbeit nicht in Frage (vgl. Kapitel 4.2.1, 4.3.1).

Die DCM-Blöcke der Virtex-5 FPGA-Reihe lassen sich über 2 Librarymodule, `DCM_ADV` (dynamisch rekonfigurierbar) bzw. `DCM_BASE` (statische Konfiguration), instanziieren und haben folgende grundsätzliche Funktionen (vgl. Abbildung 9):

- *Clock Deskew*
  Clock Skew beschreibt die Laufzeitdifferenz, die entsteht, wenn ein Taktsignal aufgrund unterschiedlicher Signalpfade verschiedene Bereiche einer digitalen Schaltung zu unterschiedlichen Zeitpunkten erreicht (vgl. [19]). Die **D**elay **L**ocked **L**oop (DLL) der DCM hat die Funktion, *Clock Skews* zu eliminieren (*Clock Deskew*). Sie besteht aus einer Kette von Verzögerungselementen, die vom Eingangstaktsignal durchlaufen werden. Der Ausgang jedes Verzögerungselementes entspricht dem um eine bestimmte Zeit verzögerten Eingangstaktsignal. Die Phasen des Eingangs- und des zurückgeführten Taktsignals werden miteinander verglichen. Daraus wird ermittelt, welches Verzögerungselement zu wählen ist, um Eingangs- und zurückgeführtes Taktsignal zu synchronisieren.

- *Frequency Synthesis*
  Aus dem Eingangstaktsignal (*CLKIN*) werden verschiedene Ausgangstaktfrequenzen abgeleitet (vgl. Abbildung 9):

  - die Originalfrequenz (*CLK0*),
  - die doppelte (*CLK2X*),
  - (durch Multiplikation und Division mit Integern) ein Vielfaches (*CLKFX*),
  - und (durch Division) eine geteilte Frequenz (*CLK_DIV*).

- *Phase Shifting*
  Mit *Phase Shifting* lassen sich die Ausgangstaktsignale in der Phase verschieben. *CLK0*, *CLK2X* und *CLKFX* sind um 0°, *CLK90* um 90°, *CLK180* und *CLKFX180* um 180° und *CLK270* um 270° in der Phase verschoben. Zusätzlich lässt sich die Phasenverschiebung sämtlicher Ausgangstaktsignale in $+\text{-}T_{CLK\_IN}/256$-Schritten justieren.

- *Dynamic Reconfiguration*
  Die Konfigurationen des DCM-Blocks lassen sich zur Laufzeit ohne Änderungen am FPGA-Design über Rekonfigurationseingänge vornehmen und über Statusausgänge auslesen.

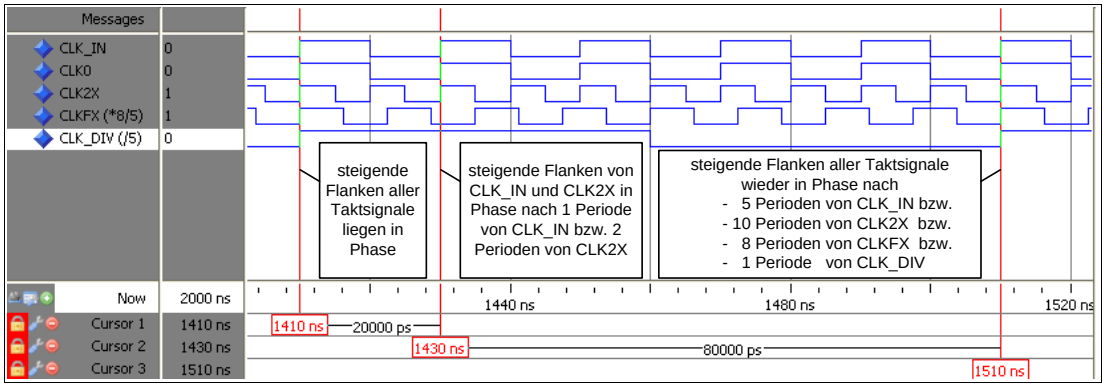

Abbildung 9: Funktionale Simulation eines Virtex-5-DCM-Blocks bei CLKIN=50MHz

Die DCM-Blöcke der Virtex-5 FPGA-Reihe werden über die Librarymodule `DCM_ADV` bzw. `DCM_BASE` im VHDL-Code der Topentity instanziiert. Das Einbinden einer `DCM_BASE`–Instanz bewirkt die Vernachlässigung der konfigurationsein- und Statusausgänge (vgl. Abbildung 10). Die Instanziierung von `DCM_ADV` gestattet die Rekonfiguration der DCM zur Laufzeit.

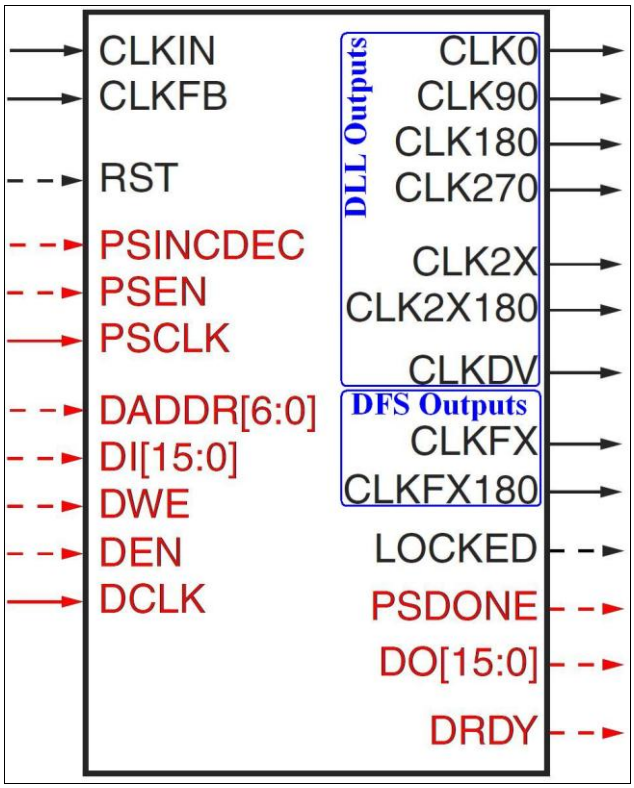

Abbildung 10: DCM-Blöcke der Virtex-5 FPGA-Reihe (schwarze Signale → `DCM_BASE`, schwarze+rote Signale → `DCM_ADV`)

Das *Locked*-Signal geht auf High-Pegel, sobald die DCM eingeschwungen ist und gültige Ausgangstaktsignale anliegen.

Der Reset-Eingang (*RST*) bewirkt das Zurücksetzen und Neueinschwingen der DCM. *RST* ist bei Rekonfiguration der DCM für mindestens 3 Taktperioden des Eingangstaktsignals (*CLKIN*) auf High-Pegel zu halten (vgl. Anhang A2).

In Abhängigkeit vom Betriebsmodus der DCM-Blöcke (Maximum Range bzw. Maximum Speed) und unter Verwendung der DLL Outputs (z.B. Rückführung *CLK0* → *CLKFB* zur Eliminierung der Phasenverschiebung zwischen Ein- und Ausgangstaktsignal) ergeben sich folgende, in Tabelle 3 dargestellte, Eingangstaktfrequenzbereiche:

Modus	Maximum Range	Maximum Speed
f_{CLKIN_Min}	19	32
f_{CLKIN_Max}	32	450

Tabelle 3: Zulässige Eingangstaktfrequenzen der DCM-Blöcke der Virtex-5 FPGA-Reihe, Speed Grade -1

3.3 Input Serializer/Deserializer - ISERDES

Die Virtex-5 FPGA-Reihe verfügt über dedizierte Serien-Parallel-Umsetzer, den ISERDES-Blöcken (vgl. Abbildung 11). Mit ihnen lassen sich die seriell auf der Datenleitung (*D*) eintreffenden Datenbits in 6-Bit-Datenworte umsetzen. Durch Verkettung zweier ISERDES-Blöcke über die *SHIFTIN1/2*-Ein- und *SHIFTOUT1/2*-Ausgänge lässt sich die Datenbreite von 6- auf bis zu 10-Bit erweitern. Ein ISERDES-Block verfügt über folgende Ports:

- *D*
  - serieller Dateneingang
- *CLKDIV*
  - langsamer Takt zur Einrahmung der seriellen Datenbits eines Zyklus (vgl. Abbildung 15→*CLK_1X*)
- *CE1, CE2*
  - Clock Enable-Signale
- *OCLK*
  - Hochgeschwindigkeitstakt für die Eintaktung der seriellen Datenbits bei Speichercontrolleranwendungen (vgl. [35])
- *CLK*
  - Hochgeschwindigkeitstakt für die Eintaktung der seriellen Datenbits (vgl. Abbildung 15→*CLK_7X*)
- *CLKB*
  - invertiertes Hochgeschwindigkeitstaktsignal
- *RST*
  - asynchrones Active-High-Reset
- *Bitslip*
  - Schiebeoperation des Ausgangsschieberegisters (*Q1…6*) mit Rückführung des MSB (*Q6*) auf das LSB (*Q1*): *Q1→Q2, Q2→G3,…Q6→Q1*
- *SHIFTIN1, SHIFTIN2*
  - Carry-Eingänge für die Datenbreitenexpansion durch Verkettung zweier ISERDES–Komponente n
- *SHIFTOUT1, SHIFTOUT2*
  - Carry-Ausgänge für die Datenbreitenexpansion durch Verkettung zweier ISERDES–Komponente n
- *Q1…6*
  - *CLKDIV*-getaktete, parallele Ausgänge des serien-parallel umgesetzten Eingangsdatenstroms

Die ISERDES-Blöcke der Virtex-5 FPGA-Reihe sind im VHDL-Code der Topentity über das Librarymodul `ISERDES_NODELAY` ohne vorherige Komponentendeklaration zu instanziieren.

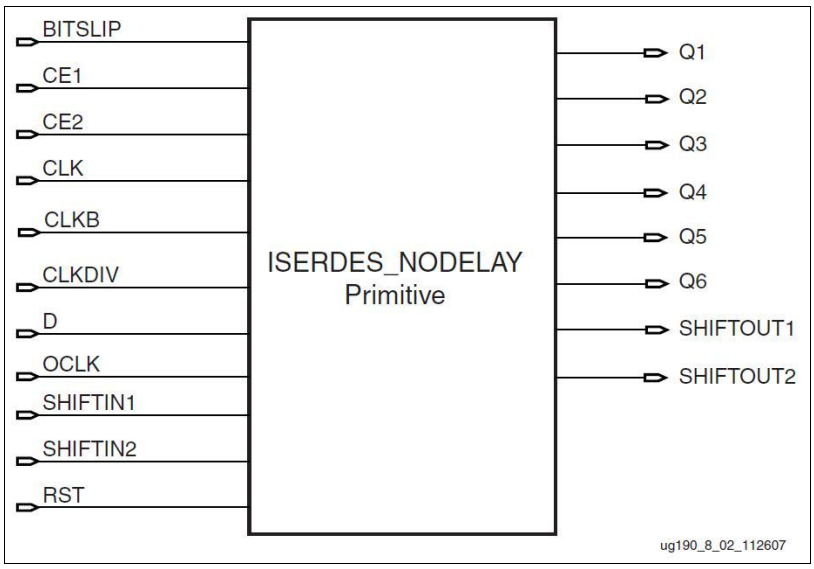

Abbildung 11: Blockschaltbild des `ISERDES_NODELAY`–Blocks

Die ISERDES-Blöcke befinden sich in so genannten *I/O Tiles*. Diese sind feste Bestandteile der Virtex-4 und Virtex-5 FPGA-Reihe und bestehen aus den folgenden Elementen (vgl. Abbildung 12):

- 2 *PAD*s
  - Signaleingangs-/ausgangspins des FPGA
- 2 **I**nput **O**utput **B**locks (*IOB*s)
  - Ein-/Ausgangssignaltreiber
- 2 *IODELAY*-Blöcke
  - feinjustierbare zeitliche Verzögerung von Ein-/Ausgangssignalen
  - in der Virtex-4-FPGA-Reihe noch Bestandteile der *ILOGIC-/ISERDES*-Blöcke (vgl. [32])
- 2 *ILOGIC*- bzw. *ISERDES*-Blöcke
  - Speicherelement für getaktetes Eingangssignal bzw.
  - Serien-Parallel-Umsetzer (Deserialisierer)
  - jeder *ILOGIC*-Block lässt sich wahlweise als *ISERDES* konfigurieren
- 2 *OLOGIC*- bzw. *OSERDES*-Blöcke
  - Speicherelement für getaktetes Ausgangssignal bzw.
  - Parallel-Serien-Umsetzer (Serialisierer)
  - jeder *OLOGIC*-Block lässt sich wahlweise als *OSERDES* konfigurieren

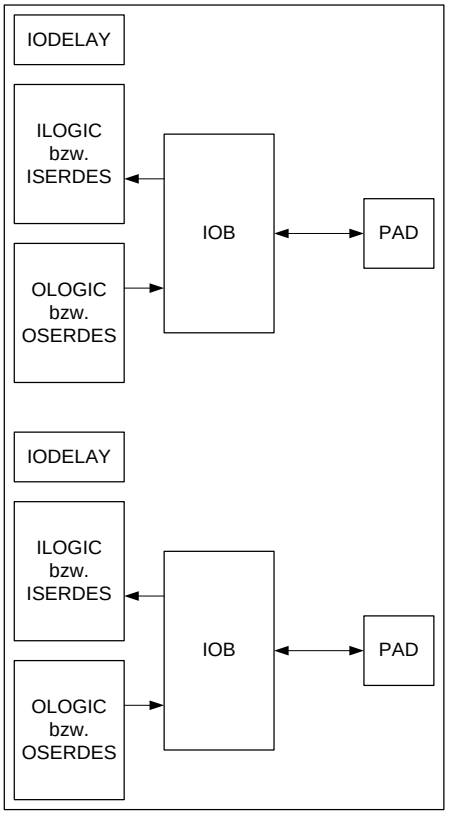

Abbildung 12: I/O Tile der Virtex-5 FPGA-Reihe

3.4 Placement-Locking für gleichbleibende Laufzeitpfade

Das Placement-Locking ist ein Verfahren, durch das für ein HDL-Design erforderliche FPGA-Ressourcen fest platziert werden, um gleichbleibende Signallaufzeitpfade zu erreichen.

Der *Place & Route*-Prozess (*PAR*) der Entwicklungsumgebung *ISE* übernimmt nach erfolgreicher Synthese des HDL-Entwurfs den Implementiervorgang (vgl. [17]). Die für das Design erforderlichen FPGA-Ressourcen, wie DCM-Blöcke, Flipflops, LUTs etc., werden unter Einhaltung kürzester Signalpfade und optimaler Taktversorgung ausgewählt und zu einer digitalen Schaltung verdrahtet. Das automatische *PAR* lässt sich wahlweise modifizieren bzw. gänzlich manuell vornehmen. Dies kann bei bestimmten Designs von Vorteil sein, wie in folgendem Beispiel erläutert wird:

1. Ein HDL-Entwurf sei mit einer DCM ausgelegt und diese (durch Phasenverschiebung) für ein festgelegtes Timing justiert worden.
2. Der Implementiervorgang mit automatischem *PAR* wird ausgeführt.
3. Die Durchführung einer Timing-Simulation ermöglicht die Verifikation der gewünschten Timings bzw. die Erkenntnis über den Nachjustierungsbedarf des DCM-Blocks.
4. Wird das Design im Nachhinein modifiziert bzw. erweitert, besteht die Möglichkeit, dass die DCM des Designs durch das automatische *PAR* in einen anderen DCM-Block positioniert wird und die gewünschten Timings aufgrund der veränderten Signalpfade nicht mehr eingehalten werden.

Demnach ist nach jeder Modifikation am Design eine erneute Timing-Simulation zur Verifikation der Timings durchzuführen, was durch das Placement-Locking entfällt. Dazu wird das Design einmalig dem automatischen *PAR* unterzogen und anschließend die DCM-Platzierung im *FPGA Editor* festgelegt (vgl. Abbildung 13). Der *FPGA Editor* ist eine Anwendung mit graphischer Bedienoberfläche, mit der sich FPGAs anzeigen und konfigurieren lassen (vgl. [14]).

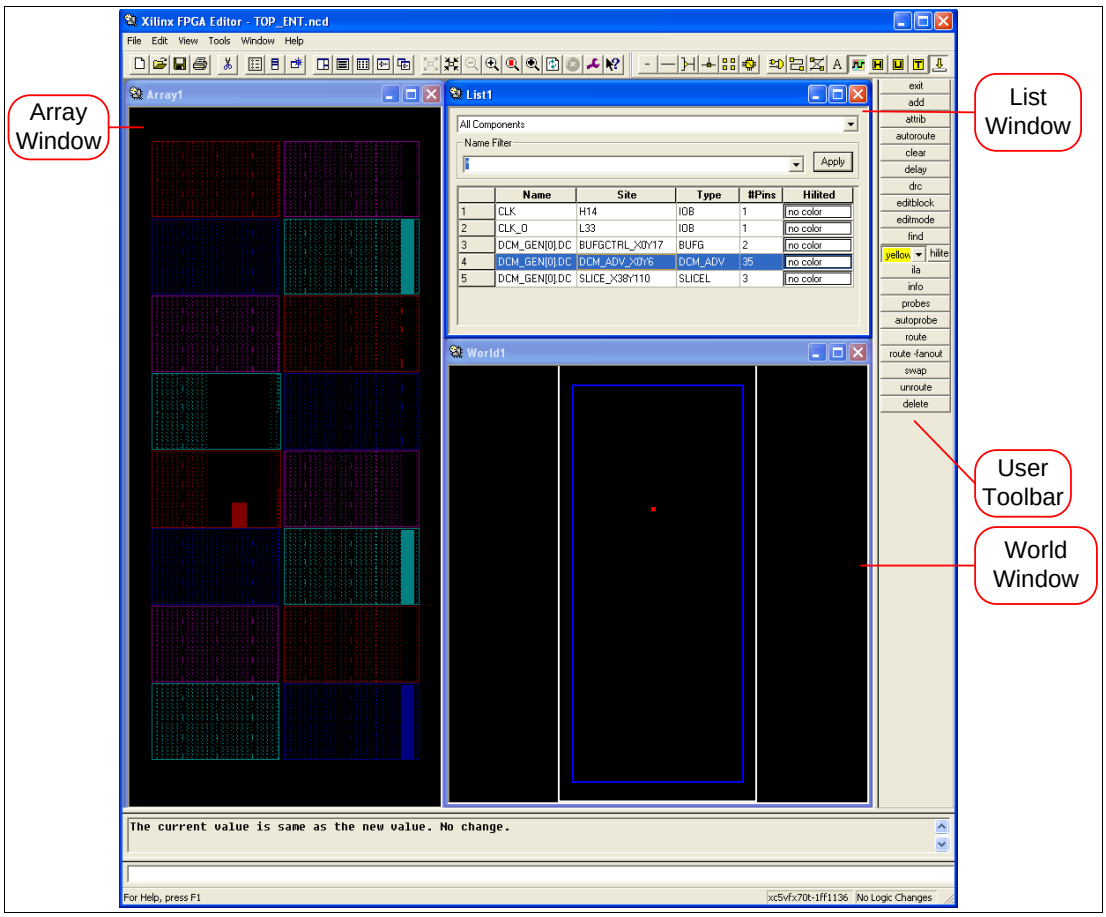

Abbildung 13: Xilinx v10.1.03 FPGA Editor

Das Hauptfenster des *FPGA Editor* besteht aus folgenden Elementen:

- *Array Window*
  Hardware-Komponenten und Verdrahtungen werden hier angezeigt und lassen sich direkt editieren.
- *List Window*
  Auflistung sämtlicher Komponenten des HDL-Entwurfs mit Namen (*Name*) , räumlichen Koordinaten im FPGA (*Site*), Typ (*Type*) und Anzahl Pins (*#Pins*).
- *World Window*
  Der räumliche Standort der im *List Window* markierten Komponenten. Diese Ansicht dient als Orientierungshilfe beim Zoomvorgang im *Array Window*.

Die Vorgangsweise für das Placement-Locking von Hardware-Komponenten wird mit der Platzierung eines DCM-Blocks erläutert:

1) In der Komponenten-Liste (*Window* → *New* → *List Window*) wird der DCM-Block (*Type* → *DCM_ADV*) ausgewählt und anschließend in der *User Toolbar* auf *attrib* geklickt.
2) Unter *Physical Constraints* wird ein Haken bei *Lock Placement* gesetzt und das Fenster wieder geschlossen (vgl. Abbildung 14).
3) Der gesamte Vorgang ist abzuspeichern (*File* → *Save*).

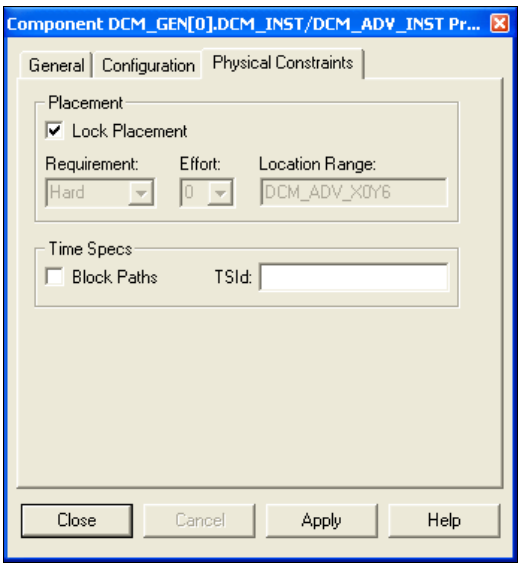

Abbildung 14: FPGA-Editor → Attribut der *DCM_ADV*-Komponente

Nachfolgend ausgeführte *PAR*-Prozesse ren den DCM-Block des HDL-Entwurfs fortan an der festgelegten Position im FPGA (vgl. Abbildung 14 → *Location Range*). Die im *FPGA Editor* vorgenommenen Einstellungen werden folgendermaßen in einem **P**hysical **C**onstraint **F**ile (PCF) abgespeichert:

```
COMP "DCM_ADV_INSTANCE_LABEL" LOCATE = SITE "DCM_ADV_X0Y6" LEVEL 0;
```

Weiterhin lassen sich die Koordinaten der jeweiligen Hardware-Komponente im **U**ser **C**onstraint **F**ile (UCF) angeben und dadurch fest platzieren:

```
INST "DCM_ADV_INSTANCE_LABEL" LOC = "DCM_ADV_X0Y6";
```

Die Einträge des UCF werden von denen des PCF überschrieben.

4 Serien-Parallel-Umsetzer für das Channel Link Interface

Das Channel Link-Protokoll mit einem Datendurchsatz von 2,38GBps und mehr wurde für Hochgeschwindigkeitsdatenübertragungen konzipiert (vgl. Anhang A1, [26]). Aufgrund der damit erreichbaren hohen Datenübertragungsraten bei geringem Systemtakt und der geringen Anzahl an Interfaceleitungen wurde das Camera Link Interface basierend auf dem Channel Link-Protokoll für digitale Kameras eingeführt. Im Unterschied zum Channel Link-Protokoll verfügt die *Base Configuration* des Camera Link Interface zusätzlich über serielle Kommunikations- und Kamerasteuerungsleitungen. Da die für die Bildübertragung erforderliche Taktleitung und 4 Datenleitungen unverändert erhalten bleiben, wird bei der Camera Link *Base Configuration* auch vom Channel Link Interface gesprochen.

Neben dem Channel Link Interface werden auch Kameras hergestellt, die mit softwarebasierten Interfacetypen wie USB ausgestattet sind. Nachfolgend werden diese beiden Interfacetypen miteinander verglichen:

Channel Link Interface	USB-Interface
- pixelweise Bildübertragung	- paketorientierte Bildübertragung
- kein Protokoll-Overhead vorhanden	- Protokoll-Overhead vorhanden
- ein Pixel ist abzuspeichern	- ganzes Datenpaket ist abzuspeichern
- Serien-Parallel-Umsetzer erforderlich für Datenempfang	- USB-PHY erforderlich für Datenempfang

Die Bilddaten werden bei Kameras mit Channel Link Interface in einem Datenstrom geliefert und liegen im Gegensatz zur paketorientierten Übertragungsweise „pur“ an. So lässt sich bei der Bildverarbeitung Pipelinearchitektur bei geringem Systemtakt realisieren. Datenpakete softwarebasierter Interfacetypen wiederum müssen bei höherem Systemtakt verarbeitet werden, um dem Protokoll-Overhead gerecht zu werden. Für den Empfang des Channel Link Datenstroms ist ein Channel Link Receiver erforderlich, der im Gegensatz zum USB-PHY für den Empfang von Daten über das USB-Interface in [13] bereits entworfen und implementiert worden ist.

Aufgrund des höheren Aufwands kommen in dieser Arbeit Kameras mit softwarebasierten Interfacetypen für die Echtzeitbildverarbeitung auf FPGAs nicht in Frage.

In den folgenden Abschnitten wird die Serien-Parallel-Umsetzung des Channel Link-Datenstroms erläutert. Dazu werden in Abschnitt 4.1 zunächst der Aufbau Channel Link-Datenstroms und die grundlegende Methode der Serien-Parallel-Umsetzung, in Kapitel 4.2 und 4.3 schließlich zwei unterschiedliche Implementierungen vorgestellt und in Kapitel 4.4 miteinander verglichen.

4.1 Aufbau des Channel Link-Datenstroms

Auf den 4 Datenleitungen (*RxIN3…0*) des Channel Link Interface werden innerhalb einer Pixeltaktperiode (T_{Pixel}) jeweils 7, insgesamt also 28 Datenbits seriell im LVDS-Standard übertragen (vgl. Abbildung 15). Der Pixeltakt *RxCLK* mit dem Tastgrad a_{RxCLK} = 4/7 um-

rahmt innerhalb einer Pixeltaktperiode 2 Datenbits des vorhergehenden und 5 Datenbits des aktuellen Zyklus ein. Jedes Datenbit liegt für die Dauer von $T_{Pixel}/7 = T_{Bit}$ an.

Die CCD-Kamera A301b liefert je Taktperiode von *RxCLK* (vgl. Kapitel 2.4):

- zwei auf 10- bzw. 8-Bit-Pixelauflösung parametrierbare Grauwertpixel (*Odd 7..0*, *Even 7…0*)
  (auf die 10-Bit-Pixelauflösung wird hier nicht weiter eingegangen),
- *LVAL* (Line Valid) zur Signalisierung einer gültigen Bildzeile und
- *FVAL* (Frame Valid) zur Signalisierung eines gültigen Bildes.

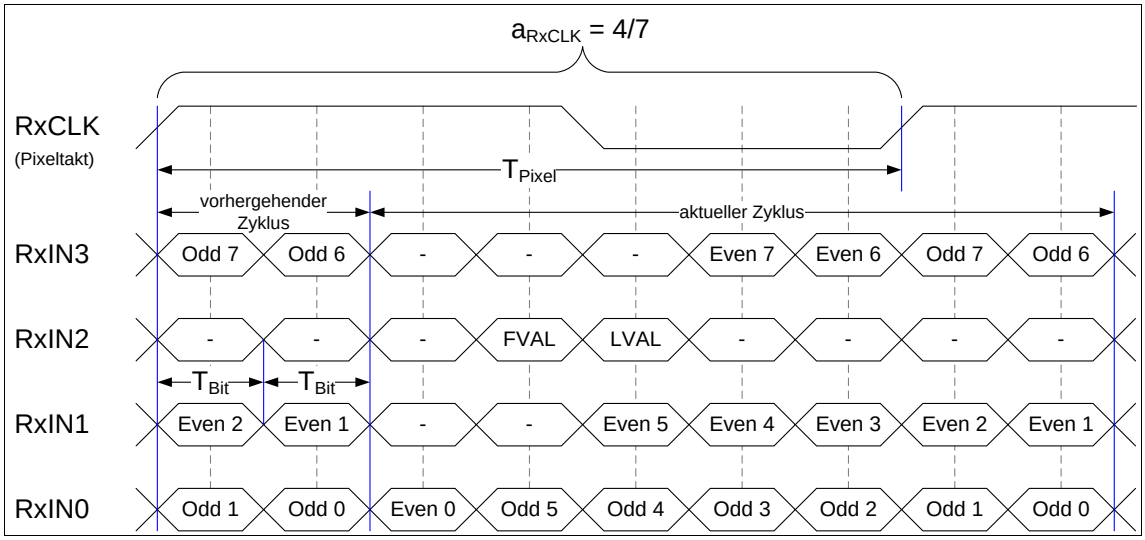

Abbildung 15: Channel Link-Pixelstrom der A301b-CCD-Kamera

4.2 Modifizierter Channel Link Receiver für die Serien-Parallel-Umsetzung des Channel Link-Datenstroms

In der Bachelorarbeit [13] wurde ein Channel Link Receiver mit robustem Synchronisationsverhalten für eine projektspezifische FPGA-Plattform entworfen und implementiert, die den seriellen Channel Link-Datenstrom einer projektspezifischen Kamera Serien-Parallel-umsetzt (vgl. Abbildung 17). Für den Einsatz des Channel Link Receivers mit unterschiedlichen Kameras und FPGA-Plattformen sind Modifikationen am Design erforderlich, da sich

- Datenstrom und Pixeltakt der Kamera und
- die FPGA-Technologie unterscheiden können.

Folgende Modifikationen wurden am Channel Link Receiver vorgenommen, um ihn ohne nachträgliche Änderungen am Design in dieser und nachfolgenden Arbeiten einsetzen zu können (vgl. gelbe Markierungen in Abbildung 17):

- Verlagerung des *Descramblers*
  Der *Descrambler* führt die Neustrukturierung der aus dem Datenstrom parallelisierten Datenbits aus (vgl. Abbildung 15). Die Datenströme verschiedener Kameras

können sich unterscheiden, wodurch eine modifizierte Neustrukturierung erforderlich ist. Das Descrambling wird beim modifizierten Design außerhalb der Komponente vorgenommen.

- Verlagerung der DCM-Konfiguration (vgl. Kapitel 4.2.1)
  In der DCM-Konfiguration (*DCMs*) wird aus dem Pixeltakt *RxCLK*
  - der phasenverschobene Rahmentakt *CLK_1X* gleicher Periodendauer mit dem Tastverhältnis $a_{CLK\_1X} = 0.5$
  - und das Siebenfache des Pixeltaktes, Bittakt *CLK_7X* generiert (vgl. Abbildung 16).

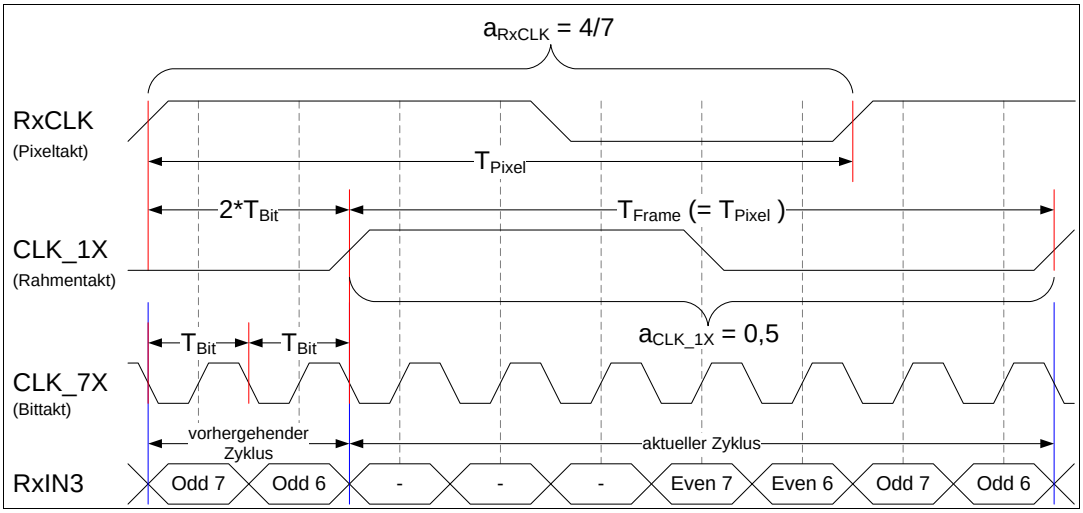

Abbildung 16: Rahmen- und Pixeltakt gegenüber Pixeltakt

Verschiedene Kameras können unterschiedliche Pixeltakte aufweisen, sodass eine Modifikation der DCM-Konfigurationen unter Berücksichtigung der jeweiligen FPGA-Technologie erforderlich ist. Aus diesem Anlass wird die Konfiguration der DCMs beim modifizierten Design außerhalb der Komponente vorgenommen.

- Entfernung des *DELAY_RST*
  Das *DELAY_RST* generiert das Reset-Signal für außerhalb des Channel Link Receivers liegende Komponenten. Sobald die *DCMs* ihre Bereitschaft signalisiert haben, hält *DELAY_RST* das Reset-Signal weitere 3 Takte lang und entfernt es anschließend. Da die *DCMs* aus dem Channel Link Receiver ausgelagert wurden, wird das Reset-Signal ebenfalls außerhalb der Komponente erzeugt.

- Änderung der Portdefinition und Signalnamen
  Die seriellen Datenbits an den Dateneingängen (*RX0…3_I*) werden deserialisiert an die Ausgänge (*RX0…RX3_O*) der Komponente gelegt. Somit stehen alle 28 empfangenen Datenbits parallel zur Verfügung. Weiterhin wurden die Takteingänge *CLK_1X* für den Pixeltakt und *CLK_7X* für das Siebenfache des Pixeltaktes herausgeführt.

Der Aufbau des originalen Channel Link Receivers ist in Abbildung 17, der Aufbau des modifizierten Channel Link Receivers (*ChannelLink_Receiver_v2*) in Abbildung 18 dargestellt.

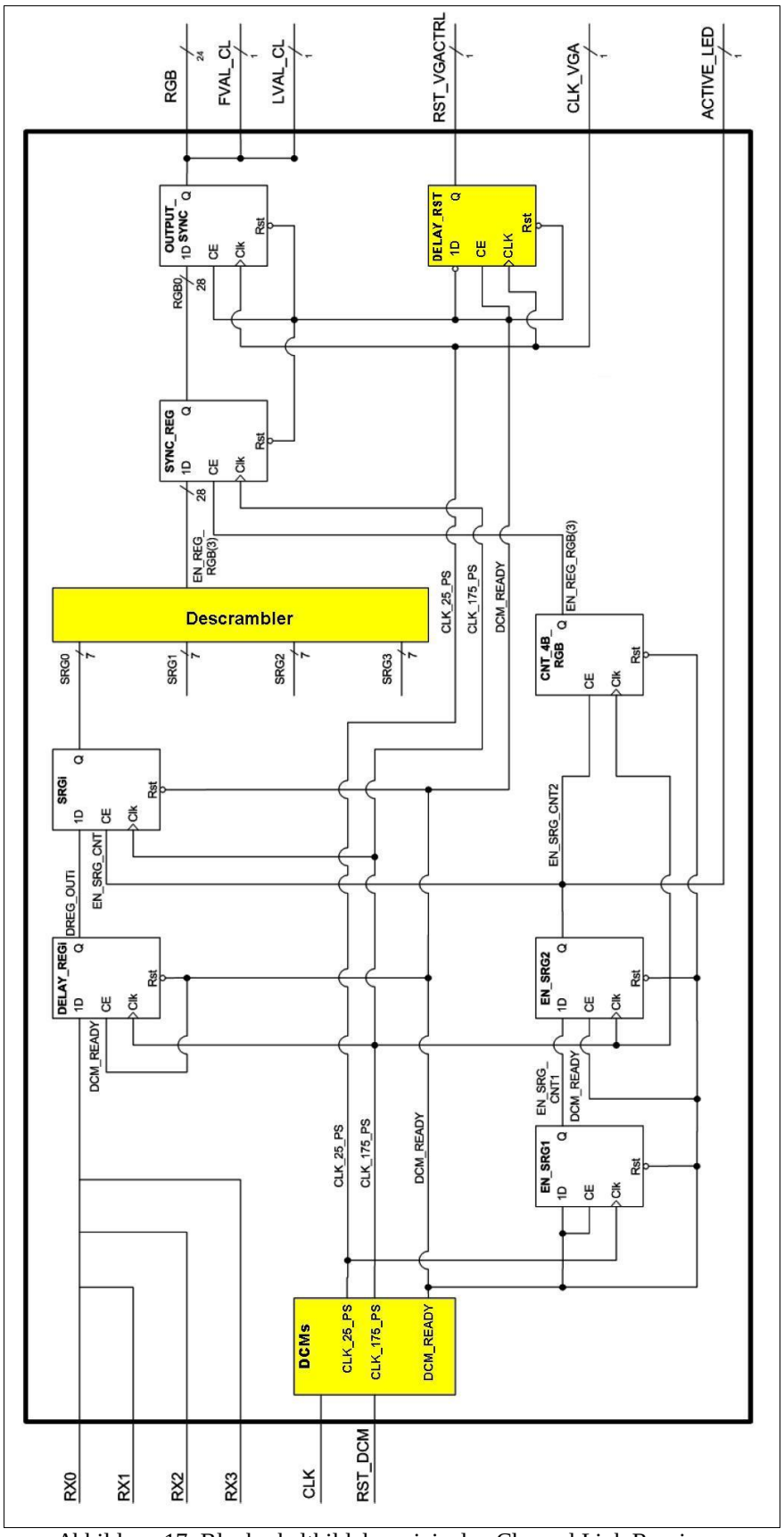

Abbildung 17: Blockschaltbild des originalen Channel Link Receivers

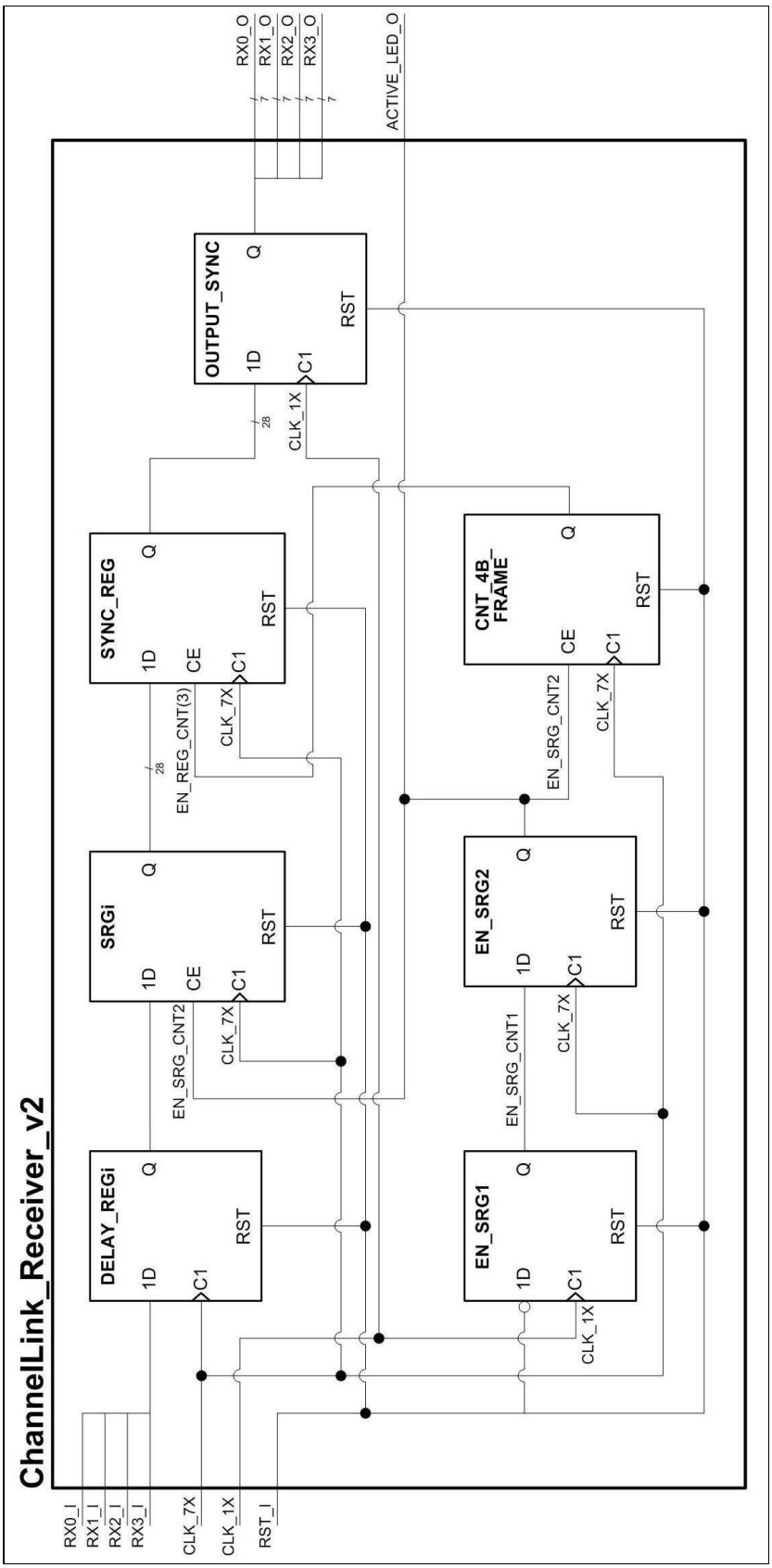

Abbildung 18: Blockschaltbild des modifizierten Channel Link Receivers

Das Ausgangsverhalten des modifizierten Channel Link Receivers wurde in einer funktionalen Simulation verifiziert (vgl. Abbildung 19). Dazu wurde eine Testbench erstellt, die die Eingänge des als Komponente eingebundenen modifizierten Channel Link Receivers stimuliert (vgl. Anhang A3).

(1) Der Rahmentakt *CLK_1X* ist gegenüber dem Pixeltakt *RxCLK* um 2 Bittaktperioden in der Phase verschoben.

(2) Bei steigender Flanke von *CLK_1X* ist das Reset-Signal *RST_I* auf High-Level, somit sind die Ausgänge der Komponente alle auf Low-Level.

(3) Bei steigender Flanke von *CLK_1X* ist *RST_I* auf Low-Level. Die Komponente beginnt bei den steigenden Flanken von *CLK_7X* die seriellen Datenbits einzutakten. Die unsortiert im Datenstrom liegenden Pixelbits wurden farblich markiert: gerade Pixel grün, ungerade Pixel rot.

(4) Die seriell zu übertragenen Pixelbits wurden hier zum Vergleich mit ihrer tatsächlichen Lage im Datenstrom sortiert aufgestellt.

(5) Die innerhalb einer Rahmentaktperiode eingetakteten Datenbits der Dateneingänge *RX3_I...RX0_I* liegen 2 Rahmentaktperioden später parallel an den Ausgängen *RX3_O...RX0_O*.

(6) Die parallel und unsortiert an den Ausgängen *RX3_O...RX0_O* anliegenden Pixelbits wurden hier zum Vergleich mit den seriell zu übertragenen Pixelbits sortiert aufgestellt.

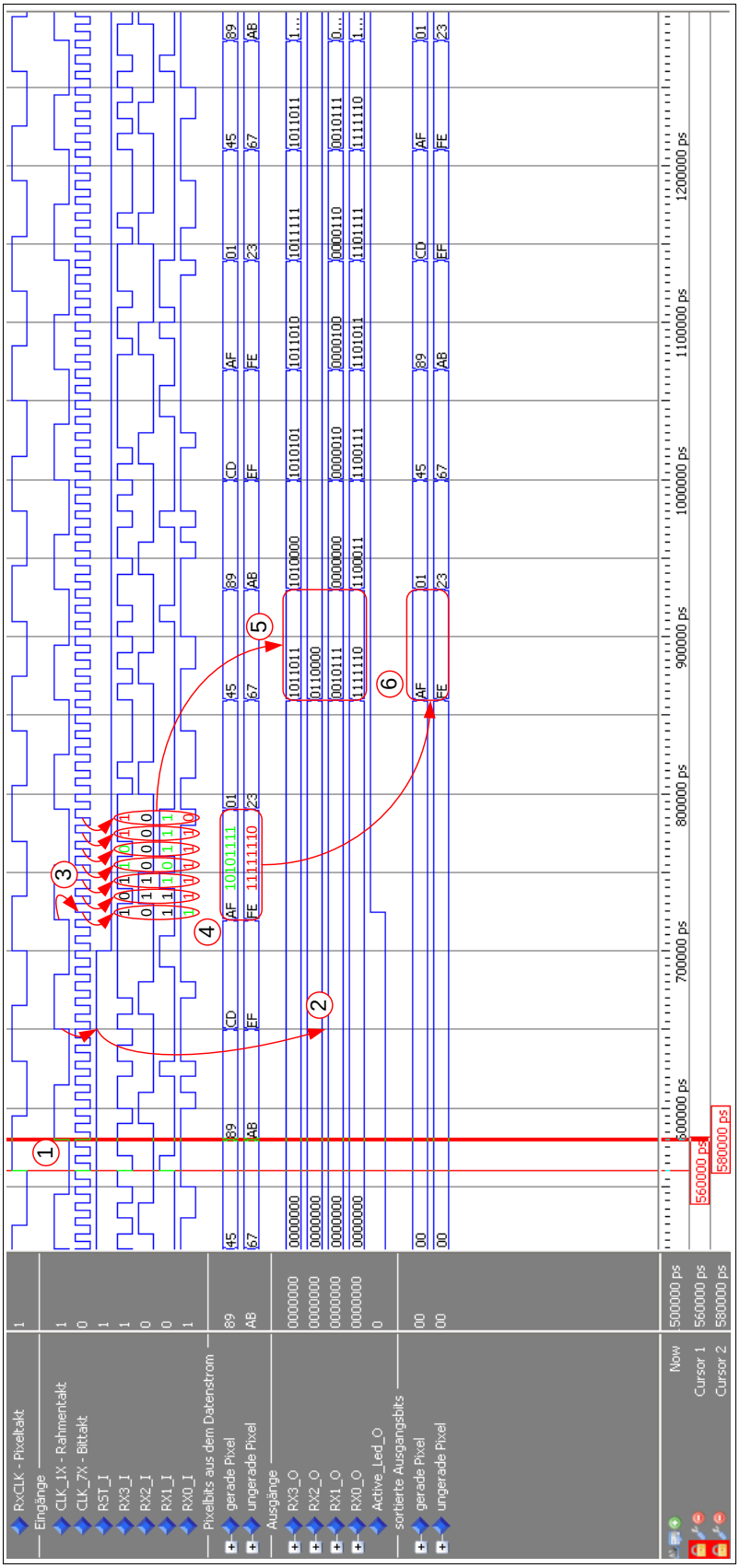

Abbildung 19: Funktionale Simulation des modifizierten Channel Link Receivers

4.2.1 DCM-Konfiguration für den modifizierten Channel Link Receiver

Die für den modifizierten Channel Link Receiver erforderlichen Taktsignale

- *CLK_1X*
  für die Einrahmung 7 zusammenhängender Datenbits aus 2 Zyklen (vgl. Abbildung 15)
- und *CLK_7X*
  für die Eintaktung der Datenbits

werden in DCMs generiert. Dazu wurde die aus 3 DCM-Blöcken bestehende Komponente *DCM_18_TO_18_36_72_126_PS* mit folgenden Ein-/Ausgängen erstellt (vgl. Abbildung 20):

- *CLK18_I* (→ *CLK_1X*)
  - Aus dem Channel Link Interface eintreffender 18MHz-Pixeltakt (vgl. Kapitel 2.4)
- *RST_I*
  - Setzt *DCM0_inst* über eine *DCM_RESET_LOGIC*–Komponente taktsynchron zurück (vgl. Anhang A2)
  - *DCM0_inst* setzt ebenfalls über eine *DCM_RESET_LOGIC*–Komponente *DCM1_inst* und diese *DCM2_inst* zurück
- *CLK18_O*
  - Gegenüber *CLK18_I* um $2*T_{Bit} = 15{,}873ns$ in der Phase verschobener Rahmentakt
- *CLK36_O*
  - Doppelter Pixeltakt
  - Jede 2. steigende Flanke synchron zur steigenden Flanke des Rahmentaktes
  - Verwendung: Multiplex zwischen *Even* und *Odd Pixel* (vgl. Abbildung 15)
- *CLK72_O*
  - Vierfacher Pixeltakt
  - Jede 4. steigende Flanke synchron zur steigenden Flanke des Rahmentaktes
  - Verwendung: Multiplex zwischen erstem und zweiten Pixelteil
- *CLK126_O* (*CLK_7X*)
  - Siebenfacher Pixeltakt, um 180° in der Phase verschoben
  - Jede 7. fallende Flanke synchron zur steigenden Flanke des Rahmentaktes
  - Verwendung: Eintaktung der seriellen Datenbits im modifizierten Channel Link Receiver (vgl. Kapitel 4.2)
- *DCMs_Rdy_O*
  - Bereitschaftssignal der DCMs
  - High-Pegel, wenn alle DCMs eingeschwungen sind und verlässliche Taktsignale liefern
  - Verwendung: Active-Low-Reset-Signal für andere Komponenten

Die einzelnen Blöcke der Komponente *DCM_18_TO_18_36_72_126_PS* haben folgende Funktionen:

- *DCM0_INST:*
  In diesem Block wird die Phasenverschiebung von $2*T_{Bit}$ = 15,873ns erzeugt. Das *CLK90*-Ausgangstaktsignal ist gegenüber dem Eingangstaktsignal *CLK18_I* um 90°, also $\varphi_{90°}$ = (90°/360°)*(1/f) = 0,25*(1/18MHz) = 0,25*55,556ns = 13,889ns in der Phase verschoben. Die restlichen 15,873ns – 13,889ns = 1,984ns werden über das *Phase Shifting*-Modul des DCM-Blocks justiert (vgl. Kapitel 3.2, 7).

- *DCM1_inst:*
  In diesem Block wird das Eingangstaktsignal von 18MHz auf 36MHz verdoppelt, da für die Generierung der 72MHz- und 126MHz-Taktsignale im *DCM2_inst*-Block eine Mindesttaktrate von 32MHz erforderlich ist (vgl. Kapitel 3.2).

- *DCM2_inst*
  In diesem Block werden sämtliche Ausgangstaktsignale erzeugt. Dadurch sind alle Ausgangstaktsignale synchron zueinander.

- *BUFG*
  Starke Stromstreiber, die für steile Taktflanken sorgen.

- *DCM_RESET_LOGIC*
  Die Komponente *DCM_RESET_LOGIC* erzeugt ein taktsynchrones Ausgangs-Reset und hält dieses für 4 Taktperioden des Eingangstaktsignals nach Entfernen des Eingangs-Resets (vgl. Anhang A2).

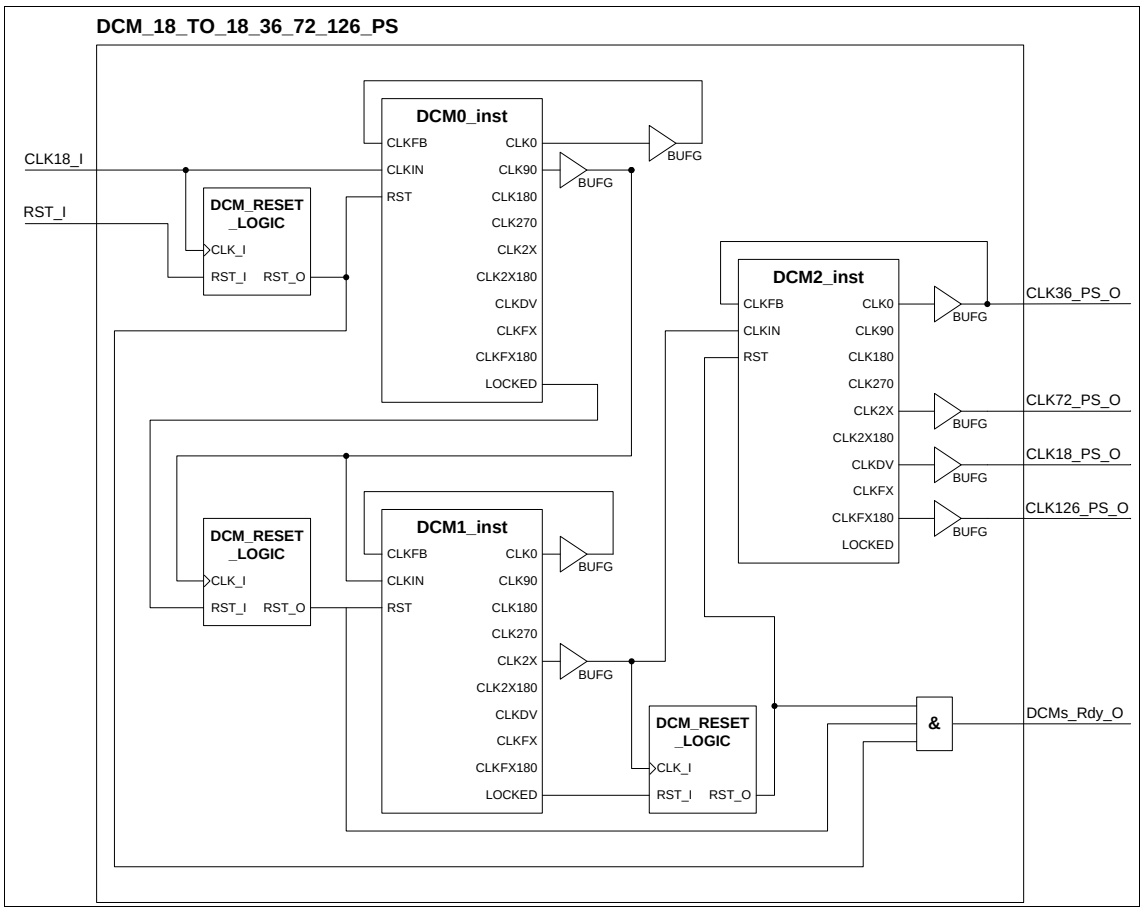

Abbildung 20: Blockschaltbild der Komponente *DCM_18_TO_18_36_72_126_PS*

Das Verhalten der Komponente *DCM_18_TO_18_36_72_126_PS* wurde in einer funktionalen Simulation verifiziert (vgl. Abbildung 21). Die Einzelschritte der Simulation werden nachfolgend erläutert:

(1) Das Bereitschaftssignal der DCMs (*DCMs_Rdy*) ist auf Low-Level, d.h. es werden noch keine verlässlichen Ausgangssignale geliefert: die Periodizität des Signals *CLK72_PS_O* (72MHz) ist noch nicht erreicht, *CLK126_PS_O* (126MHz) ist gänzlich auf Low-Level.

(2) Das Eingangstaktsignal *CLK18_I* hat eine Periodendauer von T_{CLK18_I} = 55,556ns (f_{CLK18_I} = 18MHz), eine Pulsdauer von 31,476ns und eine Tastgrad von 31,476ns/55,556ns = 4/7.

(3) Die Phasenverschiebung zwischen Eingangstaktsignal und Ausgangstaktsignalen beträgt 15,842ns bzw. 2 *CLK126_PS_O*-Perioden.

(4) Das Ausgangstaktsignal *CLK18_PS_O* hat eine Periodendauer von $T_{CLK18\_PS\_O}$ = T_{CLK18_I} = 55,556ns bei einer Pulsdauer von 27,778ns und einem Tastgrad von 27,778ns/55,556ns = 0,5. Innerhalb einer *CLK18_PS_O*-Periode befinden sich…
- 2 *CLK36_PS_O*-Perioden → $T_{CLK36\_PS\_O}$ = 27,778ns, $f_{CLK36\_PS\_O}$ = 36MHz
- 4 *CLK72_PS_O*-Perioden → $T_{CLK36\_PS\_O}$ = 13,889ns, $f_{CLK72\_PS\_O}$ = 72MHz

- 7 *CLK126_PS_O*-Perioden → $T_{CLK126\_PS\_O}$ = 7,936ns, $f_{CLK126\_PS\_O}$ = 126MHz

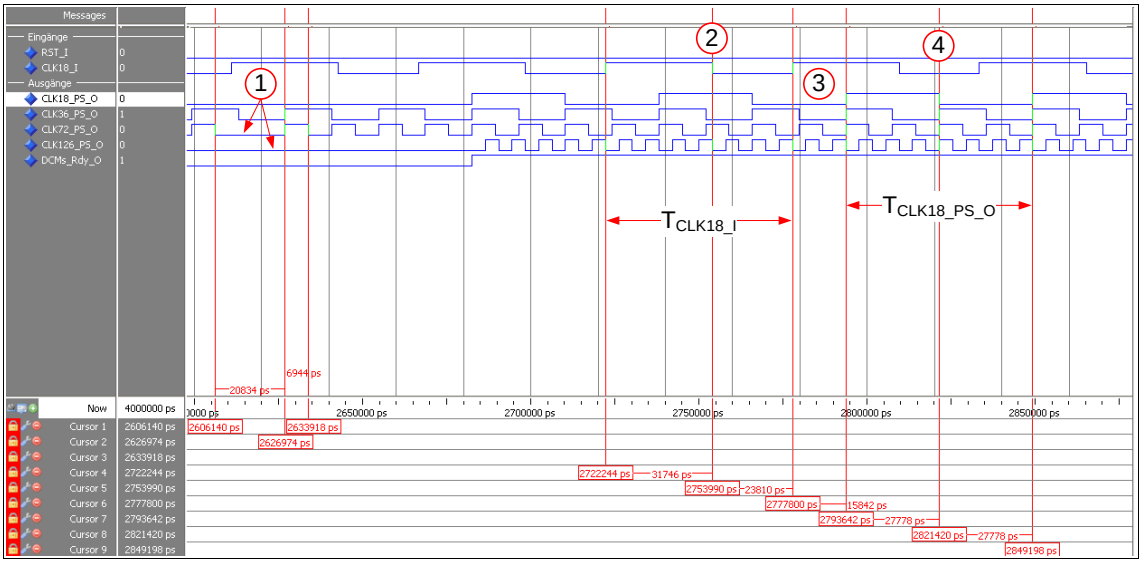

Abbildung 21: Funktionale Simulation der Komponente *DCM_18_TO_18_36_72_126_PS*

4.3 ISERDES für die Deserialisierung des Channel Link-Datenstroms

Aus 2 in Kapitel 3.3 beschriebenen `ISERDES_NODELAY`-Elementen wurde die 1:7-Deserialisierer-Komponente *ISERDES_1_ZU_7* für die Deserialisierung des Datenstroms einer separaten Channel Link-Datenleitung erstellt (vgl. Abbildung 22). Die Komponente verfügt über folgende Ports:

Eingänge:

- *CLK_7X*
  Taktsignal für die Eintaktung der seriellen Datenbits ohne Phasenverschiebung.
- *CLK_1X*
  Aus dem Channel Link Interface kommender Pixeltakt ohne Phasenverschiebung.
- *D_I*
  Serieller Dateneingang aus dem Channel Link Interface.
- *RST_I*
  Active-High-Reset-Eingang zum Zurücksetzen beider ISERDES. Das Reset-Signal ist der Komponente synchron zu *CLK_1X* zuzuführen.

Ausgänge:

- *D_O[6:0]*
  CLK_1X-getakteter Datenausgang. Die seriell eintreffenden Daten am Eingang *D_I* stehen 2 *CLK_1X*-Takte verzögert parallel am Ausgang.

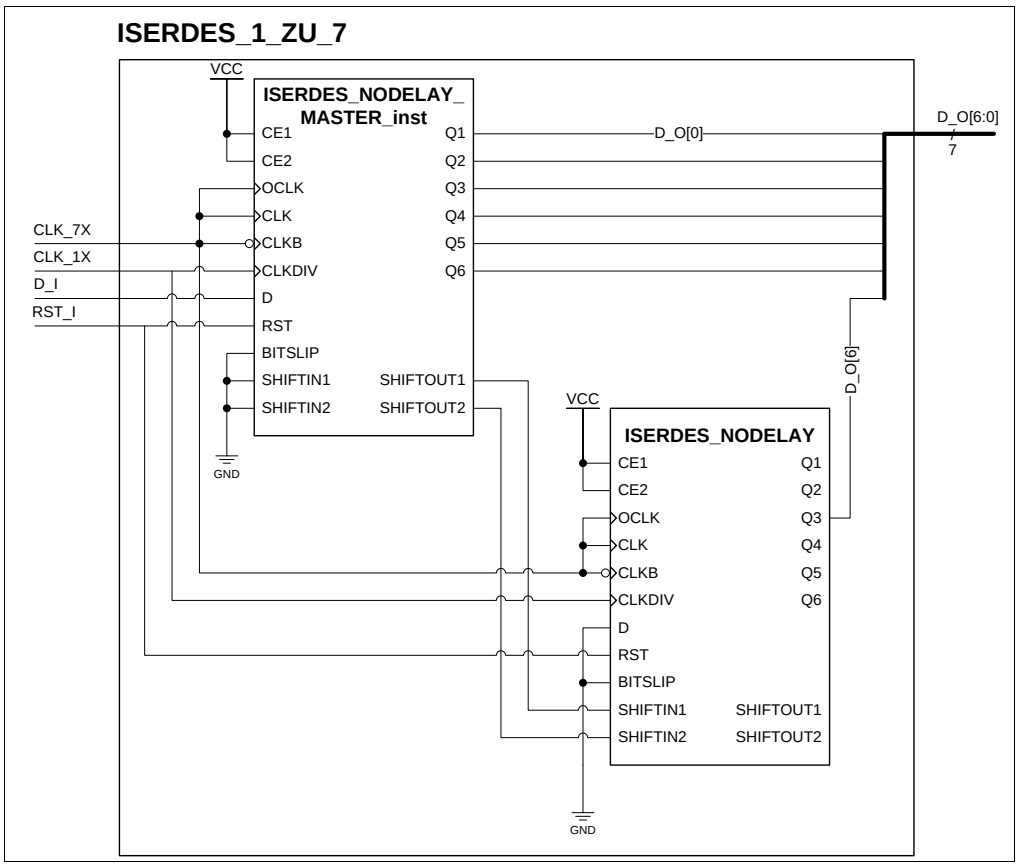

Abbildung 22: Blockschaltbild der Komponente *ISERDES_1_ZU_7*

Die Komponente *ISERDES_1_ZU_7* wurde zur Verifikation ihres Schaltungsverhaltens einer funktionalen Simulation unterzogen (vgl. Abbildung 23). Das Simulationsergebnis wird in folgenden Teilschritten erläutert.

(1) Das Reset-Signal *Reset_I* wird synchron zur steigenden Flanke von *CLK_1X* auf Low gesetzt. Es liegen 2-Bit des vorangegangenen Zyklus an (vgl. Kapitel 4.1).

(2) Zwei Bittakte (*CLK_7X*) später beginnt der Rahmen des aktuellen Zyklus. Bei steigender Flanke von *CLK_7X* werden die 7-Bit des Datenstroms *Serial_Data_I* seriell eingetaktet.

(3) 7 Bittakte später werden die 7-Bit des folgenden Zyklus eingetaktet.

(4) Die 7-Datenbit aus (2) stehen 5 Bit-Takte nach Eintaktung des letzten Bits synchron zur steigenden Flanke von *CLK_1X*-Takte parallel am Ausgang der Komponente (*Parallel_Data_O[6:0]*).

(5) Die 7-Datenbits des jeweiligen Zyklus liegen 5-Bittakte nach Eintaktung des letzten Datenbits am Ausgang an.

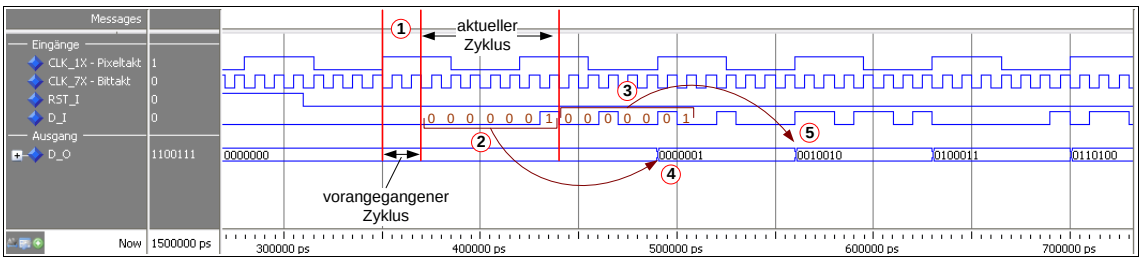

Abbildung 23: Funktionale Simulation der Komponente *ISERDES_1_ZU_7*

4.3.1 DCM-Konfiguration für die ISERDES-Blöcke

Für die Eintaktung der seriellen Datenbits am Dateneingang der Komponente *ISERDES_1_ZU_7* ist neben dem aus dem Channel Link Interface eintreffenden Pixeltakt noch der Bittakt (*CLK_7X*) erforderlich (vgl. Abbildung 22). Im Gegensatz zum modifizierten Channel Link Receiver ist hier kein Rahmentakt mehr erforderlich, sodass sich die DCM-Konfiguration ändert (vgl. 4.2). Der Bittakt wird, wie in Kapitel 4.2.1 beschrieben, innerhalb einer DCM-Konfiguration generiert. Dazu wurde die Komponente *DCM_18_TO_36_72_126* (vgl. Abbildung 24) auf Basis der Komponente *DCM_18_TO_18_36_72_126_PS* unter Wegfall des DCM-Blocks *DCM0_inst* erstellt (vgl. Abbildung 20). Die Komponente *DCM_18_TO_36_72_126* besitzt folgende Ports:

Eingänge:
- *CLK18_I*
  - Über das Channel Link Interface kommender 18MHz-Pixeltakt

- *RST_I*
  - Setzt *DCM1_inst* über eine *DCM_RESET_LOGIC*–Komponente taktsynchron zurück (vgl. Anhang A2)
  - *DCM1_inst* setzt ebenfalls über eine *DCM_RESET_LOGIC*–Komponente *DCM2_inst* zurück

Ausgänge:
- *CLK36_O*
  - Doppelter Pixeltakt
  - Jede 2. steigende Flanke synchron zur steigenden Flanke des Pixeltaktes
  - Verwendung: Multiplex zwischen *Even* und *Odd Pixel* (vgl. Abbildung 15)
- *CLK72_O*
  - Vierfacher Pixeltakt
  - Jede 4. steigende Flanke synchron zur steigenden Flanke des Pixeltaktes
  - Verwendung: Multiplex zwischen erstem und zweiten Pixelteil
- *CLK126_O* (*CLK_7X*)
  - Siebenfacher Pixeltakt um 180° in der Phase verschoben
  - Jede 7. fallende Flanke synchron zur steigenden Flanke des Pixeltaktes
  - Verwendung: Eintaktung der seriellen Datenbits in der Komponente *ISERDES_1_ZU_7* (vgl. 4.3)

- *DCMs_Rdy_O*
  - Bereitschaftssignal der DCMs
  - High, wenn beide DCMs eingeschwungen sind und verlässliche Taktsignale liefern
  - Verwendung: Active-Low-Reset-Signal für andere Komponenten

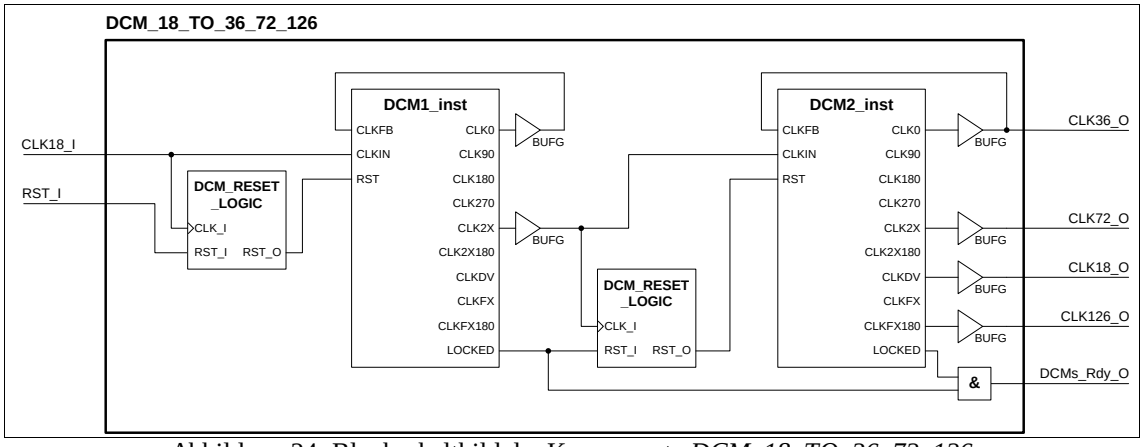

Abbildung 24: Blockschaltbild der Komponente *DCM_18_TO_36_72_126*

Die Funktionen der einzelnen Blöcke der Komponente *DCM_18_TO_36_72_126* sind identisch zu denen der Komponente *DCM_18_TO_18_36_72_126_PS* (vgl. Kapitel 4.2.1).

Das Verhalten der Komponente *DCM_18_TO_36_72_126* wurde in einer funktionalen Simulation verifiziert (vgl. Abbildung 25). Das Simulationsergebnis wird im Folgenden erläutert.

(1) Das Bereitschaftssignal der DCMs (*DCMs_Rdy*) ist auf Low-Level, d.h. es werden noch keine verlässlichen Ausgangssignale geliefert: die Periodizität des Signals *CLK72_O* (72MHz) ist noch nicht erreicht, *CLK126_O* (126MHz) ist gänzlich auf Low-Level.

(2) Das Eingangstaktsignal *CLK18_I* hat eine Periodendauer von $T_{CLK18\_I} = 55{,}556ns$ ($f_{CLK18\_I} = 18MHz$) bei einer Pulsdauer von 31,476ns und einem Tastgrad von $^{31{,}476ns}/_{55{,}556ns} = {}^{4}/_{7}$.

(3) Sämtliche Ausgangstaktsignale sind synchron zum Eingangstaktsignal *CLK18_I* und haben einen Tastgrad von 0,5. Innerhalb einer *CLK18_I* -Periode befinden sich
- 2 *CLK36_O*-Perioden → $T_{CLK36\_O} = 27{,}778ns$, $f_{CLK36\_O} = 36MHz$
- 4 *CLK72_O*-Perioden → $T_{CLK36\_O} = 13{,}889ns$, $f_{CLK72\_O} = 72MHz$
- 7 *CLK126_O*-Perioden → $T_{CLK126\_O} = 7{,}936ns$, $f_{CLK126\_O} = 126MHz$

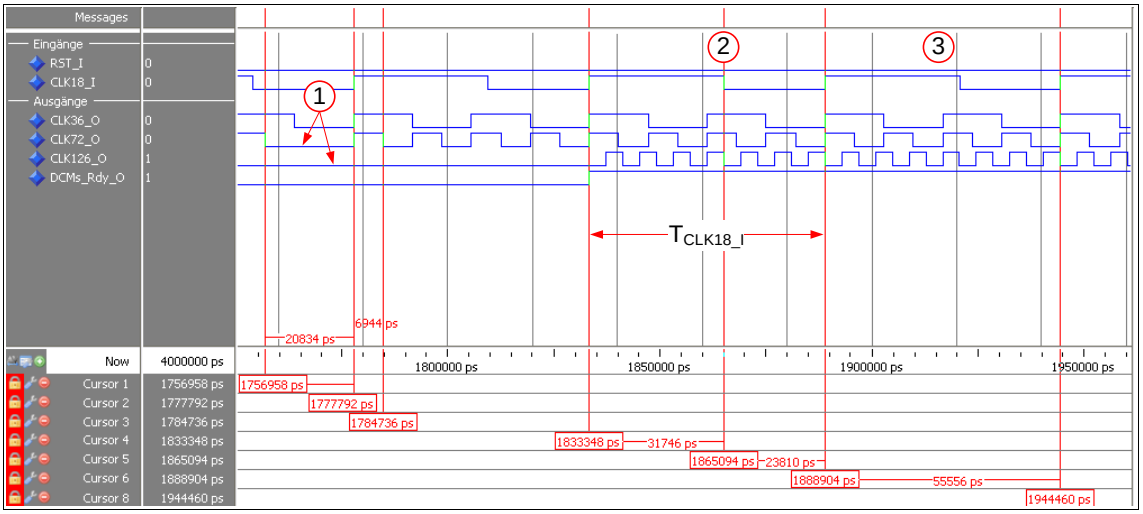

Abbildung 25: Funktionale Simulation der *DCM_18_TO_36_72_126*-Komponente

4.4 Vergleich zwischen modifiziertem Channel Link Receiver und ISERDES

Die in den Kapiteln 4.2 und 4.3 beschriebenen Channel Link Receiver-Designs sollen in diesem Kapitel, auf die Deserialisierung eines Channel Link-Datenstroms mit 18MHz-Pixeltakt angepasst, hinsichtlich ihres Ressourcenbedarfs miteinander verglichen werden. Dazu wurden die Komponenten

- *ChannelLink_Receiver_v2_TOP* (vgl. Abbildung 26)
  bestehend aus
  - 1x *ChannelLink_Receiver_v2* (vgl. Abbildung 18)
  - und 1x *DCM_18_TO_18_36_72_126_PS* (vgl. Abbildung 20)

- und *ChannelLink_Receiver_ISERDES_TOP* (vgl. Abbildung 27)
  bestehend aus
  - 4x *ISERDES_1_ZU_7* (vgl. Abbildung 22)
  - und 1x *DCM_18_TO_36_72_126* (vgl. Abbildung 24)

erstellt und in der Entwicklungsumgebung Xilinx ISE 10.1 für das Virtex-5 XC5VFX70T, FFG1136, Speed Grade -1 synthetisiert. Aus dem Syntheseergebnis wurde der Ressourcenbedarf für die Implementierung der jeweiligen Komponente ermittelt (vgl. Tabelle 4).

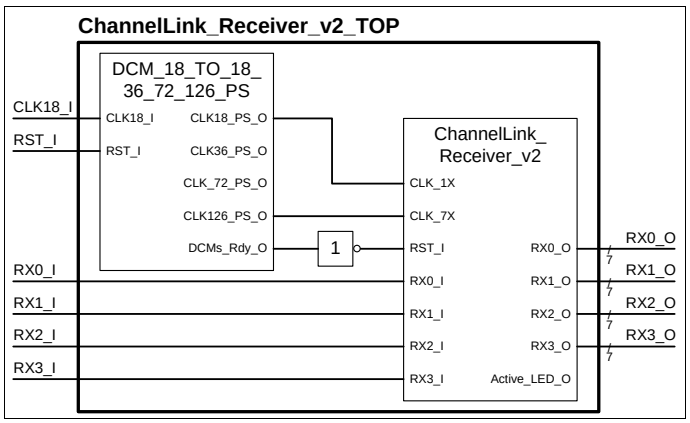

Abbildung 26: Blockschaltbild der Komponente *ChannelLink_Receiver_v2_TOP*

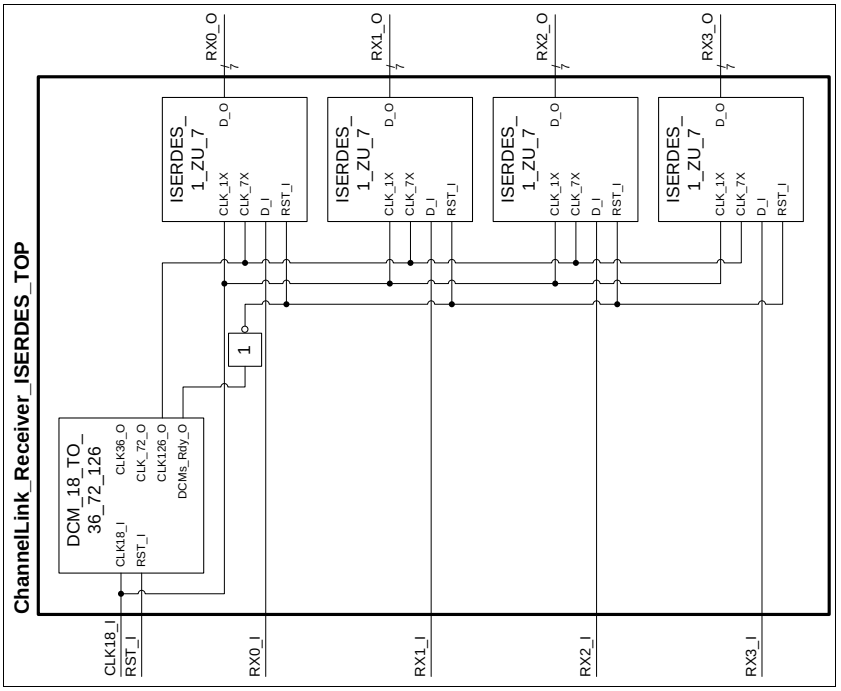

Abbildung 27: Blockschaltbild der Komponente *ChannelLink_Receiver_ISERDES_TOP*

Die Komponente *ChannelLink_Receiver_ISERDES_TOP* hat bei gleichbleibender Anzahl an Ein-/Ausgängen einen geringeren Ressourcenbedarf als die Komponente *ChannelLink_Receiver_v2_TOP*. Durch die Verwendung der *ISERDES*-Blöcke innerhalb der Komponente *ISERDES_1_ZU_7* entfällt die Implementierung von Serien-Parallel-Umsetzern als RTL-Design (vgl. 4.3), wodurch der Bedarf an Logikbausteinen sinkt und diese somit für andere Konfigurationen frei gehalten werden. Weiterhin sitzen die *ISERDES*-Blöcke in den *I/O Tile*s räumlich nah bei den *PAD*s, sodass der Signalpfad *PAD*→*ISERDES*(–Flipflops) so kurz, wie auf dem Virtex-5 XC5VFX70T FPGA möglich, gehalten wird (vgl. Abbildung 12). Die Serien-Parallel-Umsetzung bei der Komponente *ChannelLink_Receiver_v2* hingegen wird in Logikbausteinen außerhalb der *I/O Tiles* vorgenommen, wodurch längere Signalpfade, *PAD*→Flipflops zustande kommen und

somit die maximale Taktfrequenz des Designs sinkt (vgl. Tabelle 4). Die Komponente *ChannelLink_Receiver_ISERDES_TOP* ist auf die Virtex-5 FPGA-Reihe abgestimmt und wäre beispielsweise auf der SPARTAN-3 FPGA-Reihe aufgrund fehlender *ISERDES*-Blöcke nicht implementierbar, wohingegen sich die Komponente *ChannelLink_Receiver_v2_TOP* mit modifizierter DCM-Konfiguration auch auf der Spartan-3 FPGA-Reihe implementieren lässt.

Erforderliche Logik	*ChannelLink_Receiver_v2_TOP*	*ChannelLink_Receiver_ISERDES_TOP*	insgesamt verfügbar
Anzahl Slice Register	106	8	44800
Anzahl Slice LUTs	10	5	44800
Anzahl LUT-FF-Paare	7	0	109
Anzahl Ein-/Ausgänge	34	34	640
Anzahl BUFG/BUFGCTRLs	8	5	32
Anzahl DCM_ADVs	3	2	12
Max. Eingangstaktfrequenz	32,454MHz	615,764MHz	-

Tabelle 4: Ressourcenbedarf und Maximalfrequenz von *ChannelLink_Receiver_v2_TOP* und *ChannelLink_Receiver_ISERDES_TOP* auf dem Virtex-5 XC5VFX70T FPGA (vgl. Kap. 3)

Im Hinblick auf die Zielhardware dieser Arbeit (Kapitel 3) und dem geringeren Ressourcenbedarf wird die Komponente *ChannelLink_Receiver_ISERDES_TOP* als Serien-Parallel-Umsetzer des seriellen Kameradatenstroms zum Einsatz kommen. Der endgültige Hardwareaufbau und die Timing-Simulation werden in den Kapitel 7 erläutert.

5 VGA-Controller für die visuelle Kontrolle des kontinuierlichen Bilddatenstroms

Der kontinuierliche Bilddatenstrom der CCD-Kamera A301b (Kapitel 2.4) soll zur visuellen Kontrolle auf einem VGA-Monitor dargestellt werden. Da die Bilddaten nicht zwischengespeichert werden, stellt sich an einen VGA-Controller die Anforderung, aus den dem Kameradatenstrom entstammenden LVAL- und FVAL-Signalen die horizontalen und vertikalen Synchronisationssignale (*HSYNC_O*, *VSYNC_O*) zu generieren, die für den Betrieb eines VGA-Monitors erforderlich sind. Im Rahmen dieser Arbeit wurde dazu ein parametrisierbarer VGA-Controller (*VGA_CTRL_v1*) entwickelt, mit dem sich sowohl kontinuierliche Bilddatenströme, als auch zwischengespeicherte Bilddaten auf einem VGA-Monitor darstellen lassen (vgl. Abbildung 32). Die Bildauflösung des anzusteuernden VGA-Monitors lässt sich mit Generics parametrisieren, sodass der VGA-Controller für Kameras unterschiedlicher Bildauflösungen einsetzbar ist (vgl. Tabelle 5). Weiterhin werden die Polaritäten des horizontalen und vertikalen Synchronisationssignals mit den Generics `H_SYNC_POLARITY` und `V_SYNC_POLARITY` festgelegt und können somit an unterschiedliche VGA-Monitore angepasst werden. Geht das FVAL-Signal vor dem LVAL-Signal auf High-Pegel, so lässt sich diese zeitliche Differenz mit dem Generic `FVAL_BEFORE_LVAL` angeben, sodass der VGA-Controller diese zeitliche Differenz ausgleicht.

Generic-Name	Typ	Beschreibung	Standardwert
`H_PIXEL`	`natural`	Anzahl Pixel je Bildzeile.	`640`
`H_FRONT_PORCH`	`natural`	Länge des HSYNC-Front Porch (in Pixeltaktperioden).	`16`
`H_SYNC_PULSE`	`natural`	Länge des HSYNC-Sync Pulse (in Pixeltaktperioden).	`96`
`H_BACK_PORCH`	`natural`	Länge des HSYNC-Back Porch (in Pixeltaktperioden).	`48`
`H_SYNC_POLARITY`	`bit`	Polarität des horizontalen Synchronisationssignals.	`'0'`
`FVAL_BEFORE_LVAL`	`natural`	Zeit in Pixeltaktperioden, in der FVAL vor LVAL auf High-Pegel geht.	`0`
`V_FRONT_PORCH`	`natural`	Länge des VSYNC-Front Porch (in Bildzeilenperioden).	`10`
`V_SYNC_PULSE`	`natural`	Länge des VSYNC-Sync Pulse (in Bildzeilenperioden).	`2`
`V_BACK_PORCH`	`natural`	Länge des VSYNC-Back Porch (in Bildzeilenperioden).	`29`
`V_SYNC_POLARITY`	`bit`	Polarität des vertikalen Synchronisationssignals.	`'0'`

Tabelle 5: Generics des VGA-Controllers *VGA_Controller_v1*

Der VGA-Controller *VGA_CTRL_v1* verfügt über folgende Ein- und Ausgänge:

Portname	Signalbreite	Beschreibung
CLK_I	1	Taktsignaleingang (Pixeltakt) für sämtliche taktrelevante Elemente der Komponente. Der Pixeltakt ist der Komponente je nach Bildauflösung und Bildrate angepasst zuzuführen.
RST_I	1	Reset-Eingang zum Zurücksetzen der Komponente.
LVAL_I	1	Line Valid-Eingang (LVAL).
FVAL_I	1	Frame Valid-Eingang (FVAL).
RED_I	8	Getakteter Eingang des roten Pixelanteils.
GRN_I	8	Getakteter Eingang des grünen Pixelanteils.
BLU_I	8	Getakteter Eingang des blauen Pixelanteils.
HSYNC_O	1	Horizontales Synchronisationssignal für den VGA-Monitor.
VSYNC_O	1	Vertikales Synchronisationssignal für den VGA-Monitor.
BLANK_O	1	Blanking-Signal für den VGA-Monitor.
RED_O	8	Getakteter Ausgang des roten Pixelanteils.
GRN_O	8	Getakteter Ausgang des grünen Pixelanteils.
BLU_O	8	Getakteter Ausgang des blauen Pixelanteils.

Tabelle 6: Ports des VGA-Controllers *VGA_CTRL_v1*

Der VGA-Controller ist grundlegend aus 2 FSMs (*L_FSM*, *F_FSM*), 3 Zählwerken (*HCount_cnt*, *VCount_cnt*, *CLK_DIV_cnt*), kombinatorischer Logik und Ausgangsregistern zusammengesetzt (vgl. Abbildung 32). Die einzelnen Blöcke des VGA-Controllers haben folgende Aufgabe:

- *L_FSM*
  Moore-Automat für die Erkennung des LVAL-Signalpegels (vgl. Abbildung 28). Nach einem System-Reset geht *L_FSM* in den Zustand *PWRON*. Wird taktsynchron ein High-Pegel an *LVAL_I* detektiert, wechselt der Automat in den Zustand *HIGH_PERIOD*. Sobald taktsynchron ein Low-Pegel an *LVAL_I* detektiert wird, wechselt der Automat zum nächsten Takt in den Zustand *LOW_PERIOD* und aktiviert das Zählwerk *HCount_cnt*. Ein High-Pegel an *LVAL_I* führt wieder zum Wechsel in den Zustand *HIGH_PERIOD* usw.

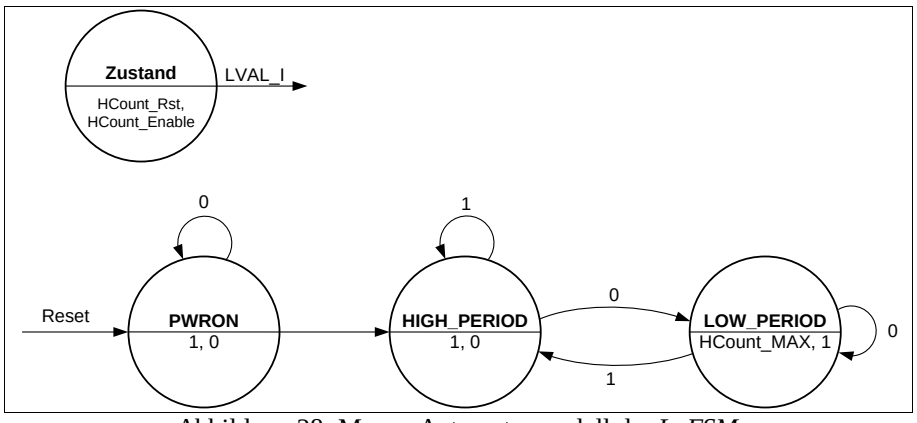

Abbildung 28: Moore-Automatenmodell der *L_FSM*

- *HCount_cnt*
  Zählwerk für die Generierung des horizontalen Synchronisationssignals. Das Zählwerk beginnt im Zustand *L_FSM→LOW_PERIOD* taktsynchron aufwärts zu zählen und wird bei Erreichen einer gesamten Bildzeilendauer (*HCount_MAX*) bzw. Im Zustand *L_FSM→HIGH_PERIOD* wieder zurückgesetzt.

- *HCount_comp_logic*
  Komparatorlogik für das bedingte Zurücksetzen von *HCount_cnt* und das Setzen des horizontalen Synchronisationssignals *HSYNC_sig* in Abhängigkeit des *FVAL_I*-Signals:

  FVAL_I-High-Phase: *LVAL_I* liegt periodisch an, *HCount_cnt* zählt für die Dauer der *LVAL_I*-Low-Phase (T_{L_Low}) und wird durch *L_FSM* bei der steigenden Flanke von *LVAL_I* wieder zurückgesetzt (vgl. Abbildung 29).

  FVAL_I-Low-Phase: *LVAL_I* ist nicht mehr periodisch, liegt dauerhaft auf Low-Pegel. *HCount_cnt* zählt periodisch für die Dauer einer Bildzeilenperiode ($T_{L\_Low} + T_{L\_High}$) und wird von *HCount_comp_logic* wieder zurückgesetzt. Ist das Generic `FVAL_BEFORE_LVAL` größer 0, d.h. besteht eine zeitliche Differenz Δt zwischen den steigenden Flanken von *FVAL_I* und *LVAL_I*, so wird diese Differenz dem ersten Zählerdurchlauf innerhalb der *FVAL_I*-Low-Phase angehängt → *HCount_cnt* läuft für die Dauer $T_{L\_Low} + T_{L\_High} + \Delta t$.

  Das horizontale Synchronisationssignal *HSYNC_O* wird kombinatorisch aus dem Vergleich des Zählstandes von *HCount_cnt* mit den Generics `H_FRONT_PORCH`, `H_SYNC_PULSE` und `H_BACK_PORCH` gesetzt (vgl. Tabelle 5).

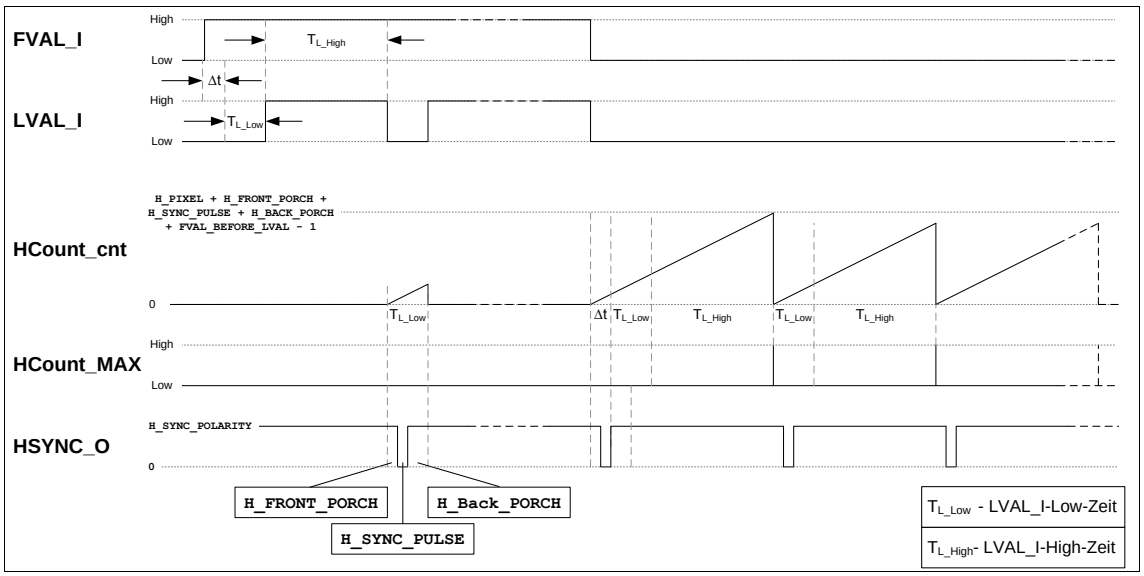

Abbildung 29: Generierung des horizontalen Synchronisationssignals *HSYNC_O*

- *F_FSM*
  Moore-Automat für die Erkennung des FVAL-Signalpegels (vgl. Abbildung 30). Nach einem System-Reset geht *F_FSM* in den Zustand *PWRON*. Wird taktsynchron ein High-Pegel an *FVAL_I* detektiert, wechselt der Automat in den Zustand *HIGH_PERIOD*. Wird taktsynchron ein Low-Pegel an *FVAL_I* detektiert, wechselt der Automat zum nächsten Takt in den Zustand *LOW_PERIOD* und aktiviert das Zählwerk *CLK_DIV_cnt*. Ein High-Pegel an *FVAL_I* führt wieder zum Wechsel in den Zustand *HIGH_PERIOD* usw.

  Das Blanking-Signal (*VBLANK_sig*) und die Rücksetzsignale für *CLK_DIV_cnt* und *VCount_cnt* (*CLK_DIV_Reset_Eable*, *VCount_ Reset_Enable*) werden den Zuständen entsprechend gesetzt.

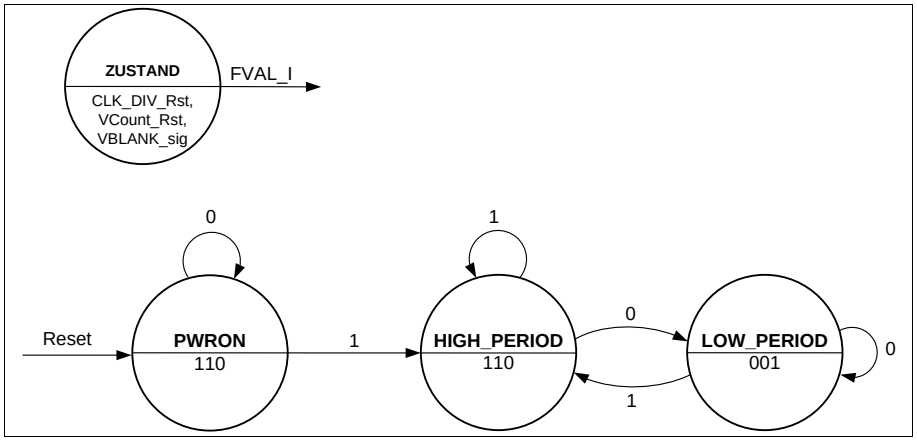

Abbildung 30: Moore-Automatenmodell der *F_FSM*

- *CLK_DIV_cnt*
  Zählwerk für das Abzählen einer Bildzeile. *CLK_DIV_cnt* beginnt auf der fallenden Flanke von *FVAL_I* (*F_FSM→LOW_PERIOD*) periodisch bei Pixeltakt (*CLK_I*) eine Bildzeile abzuzählen und wird im Zustand *F_FSM→HIGH_PERIOD* zurückgesetzt.

- *CLK_DIV_comp_logic*
  Komparatorlogik für das periodische Zurücksetzen von *CLK_DIV_cnt* und das Setzen des Enable-Signals für *VCount_cnt* (*CLK_DIV_MAX* → UND-Gatter). *CLK_DIV_MAX* wird kombinatorisch gesetzt, sobald das Zählwerk *CLK_DIV_cnt* eine Bildzeilenperiode abgezählt hat. *CLK_DIV_MAX* setzt sowohl *CLK_DIV_cnt* zurück, als auch das Enable-Signal für das Zählwerk *VCount_cnt*.

- *VCount_cnt*
  Zählwerk für das Abzählen von Bildzeilen und die Generierung des vertikalen Synchronisationssignals. *VCount_cnt* zählt im Zustand *F_FSM→LOW_PERIOD* bei steigender Taktflanke und anliegendem Enable-Signal aufwärts. Das Enable-Signal wird von *CLK_DIV_comp_logic* nach dem Abzählen einer jeden Bildzeile geliefert. *VCount_cnt* wird im Zustand *F_FSM→HIGH_PERIOD* zurückgesetzt.

- *VCount_comp_logic*
  Komparatorlogik für das Setzen des Enable-Signals für *VCount_cnt* (*VCount_Enable* → UND-Gatter) und das Setzen des vertikalen Synchronisationssignals *VSYNC_O*. Das Enable-Signal *VCount_Enable* bleibt für die Dauer der *FVAL_I*-Low-Phase gesetzt und geht nach Ablauf dieser Zeit auf Low.

  Das vertikale Synchronisationssignal *VSYNC_O* wird kombinatorisch aus dem Vergleich des Zählstandes von *VCount_cnt* mit den Generics `V_FRONT_PORCH`, `V_SYNC_PULSE` und `V_BACK_PORCH` gesetzt (vgl. Tabelle 5, Abbildung 31).

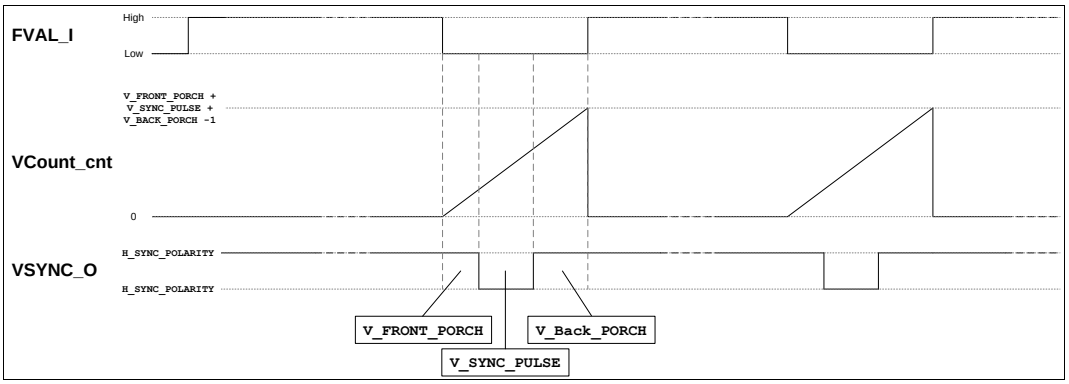

Abbildung 31: Generierung des vertikalen Synchronisationssignals *VSYNC_O*

- *OUTPUT_REG*
  Getaktetes Ausgangsregister. Sämtliche Ausgangssignale werden taktsynchron herausgeführt. Die Farbkanäle *RED_I*, *GRN_I* und *BLU_I* werden taktsynchron eingetaktet und herausgeführt, um synchron zu den von der Komponente erzeugten Ausgangssignalen zu sein.

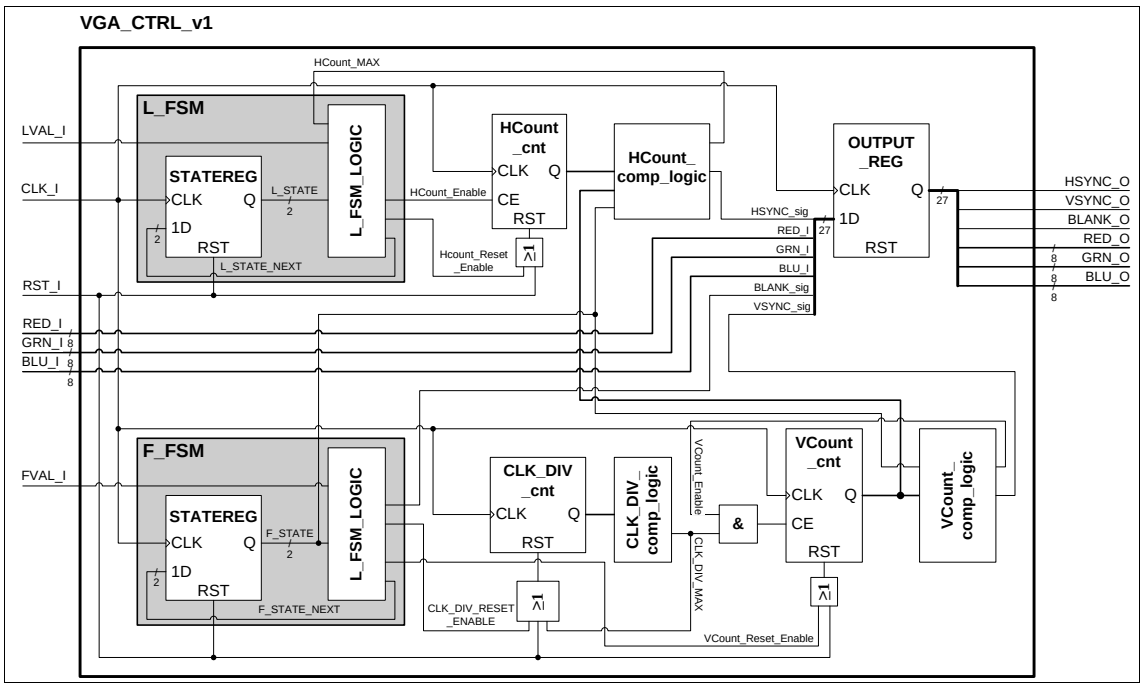

Abbildung 32: Blockschaltbild des parametrisierbaren VGA-Controllers *VGA_CTRL_v1*

5.1 Simulation und Messergebnisse

Das Verhalten des VGA-Controller *VGA_CTRL_v1* wurde in einer funktionalen Simulation verifiziert. Dazu wurde die Komponente für eine Bildauflösung von 640x480 Pixeln bei 60 Bildern/s mit folgenden Parametern konfiguriert:

Pixeltakt	25MHz	
H_PIXEL	640	} Bildzeilenperiode
H_FRONT_PORCH	16	
H_SYNC_PULSE	96	
H_BACK_PORCH	48	
H_SYNC_POLARITY	'0'	
FVAL_BEFORE_LVAL	0	
V_FRONT_PORCH	10	} Bildperiode
V_SYNC_PULSE	2	
V_BACK_PORCH	29	
V_SYNC_POLARITY	'0'	

In Abbildung 33 ist ein Ausschnitt aus der funktionalen Simulation des VGA-Controllers abgebildet, in dem die Generierung des horizontalen Synchronisationssignals *HSYNC_O* im Detail aufgezeigt wird. Nachfolgend wird das Simulationsergebnis erläutert:

(1) Innerhalb der Low-Phase von *FVAL_I*, in der *LVAL_I* ebenfalls Low-Pegel hat, werden die horizontalen Synchronisationssignale weiterhin generiert.

(2) Die Zeit zwischen der steigenden Flanke von *HSYNC_O* und der steigenden Flanke von *LVAL_I* ist der (horizontale) Back Porch [Nexys2] und beträgt 1880ns. Wird diese Zeit mit dem Pixeltakt multipliziert, ergibt sich daraus die Anzahl Zählschritte:
1880ns*25MHz = 47 → `H_BACK_PORCH` – 1.
Aufgrund des getakteten Ausgangs verschieben sich sämtliche Ausgangssignale um eine Taktperiode, wodurch sich die zeitliche Differenz zwischen Ausgang→Eingang eine Taktperiode verkürzt. Die geforderte Zeit des Back Porch wird somit eingehalten.

(3) Die Zeit zwischen der fallenden Flanke von *LVAL_I* und der fallenden Flanke von *HSYNC_O* ist der (horizontale) Front Porch und beträgt 680ns.
680ns*25MHz = 17 → `H_FRONT_PORCH` + 1
Da die zeitliche Differenz Eingang→Ausgang betrachtet wird, ist diese um eine Taktperiode (aufgrund des getakteten Ausgangs des VGA-Controllers) verlängert. Die geforderte Zeit des Front Porch wird somit eingehalten.

(4) Die Dauer des horizontalen Synchronisationspulses *HSYNC_O* ist der (horizontale) Sync Pulse und beträgt 3840ns.
3840ns*25MHz = 96 = `H_SYNC_PULSE`
Die geforderte Zeit des Sync Pulse wird eingehalten.

(5) Die geforderte Zeit des Back Porch wird bei periodisch anliegendem *LVAL_I* weiterhin eingehalten.

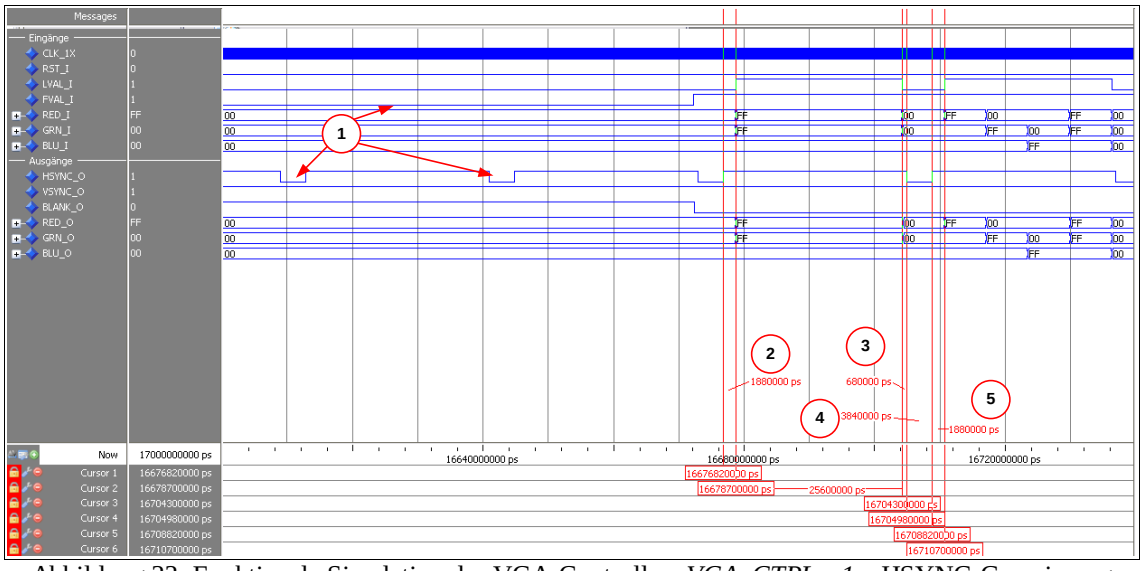

Abbildung 33: Funktionale Simulation des VGA-Controllers *VGA_CTRL_v1* – HSYNC-Generierung

In Abbildung 34 ist ein Ausschnitt aus der funktionalen Simulation des VGA-Controllers abgebildet, in dem die Generierung des vertikalen Synchronisationssignals *VSYNC_O* im Detail aufgezeigt wird. Nachfolgend wird das Simulationsergebnis erläutert:

(1) Die zeitliche Differenz zwischen fallender Flanke von *FVAL_I* und fallender Flanke von *VSYNC_O* ist der (vertikale) Front Porch und beträgt hier 320,040µs.
320,040µs*25MHz = 8001 = `V_FRONT_PORCH` * Bildzeilenperiode + 1.
Da die zeitliche Differenz Eingang→Ausgang betrachtet wird, ist diese um eine Taktperiode (aufgrund des getakteten Ausgangs des VGA-Controllers) verlängert. Die geforderte Zeit des Front Porch wird somit eingehalten.

(2) Die Dauer des vertikalen Synchronisationspulses *VSYNC_O* ist der (vertikale) Sync Pulse und beträgt 64µs.
64µs*25MHz = 1600 = `V_SYNC_PULSE` * Bildzeilenperiode
Die geforderte Zeit des Sync Pulse wird eingehalten.

(3) Die Zeit zwischen der steigenden Flanke von *VSYNC_O* und der steigenden Flanke von *FVAL_I* ist der (vertikale) Back Porch und beträgt 927,960µs.
927,960µs*25MHz = 23199 → `V_BACK_PORCH` – 1.
Aufgrund des getakteten Ausgangs verschieben sich sämtliche Ausgangssignale um eine Taktperiode, wodurch sich die zeitliche Differenz zwischen Ausgang→Eingang eine Taktperiode verkürzt. Die geforderte Zeit des Back Porch wird somit eingehalten.

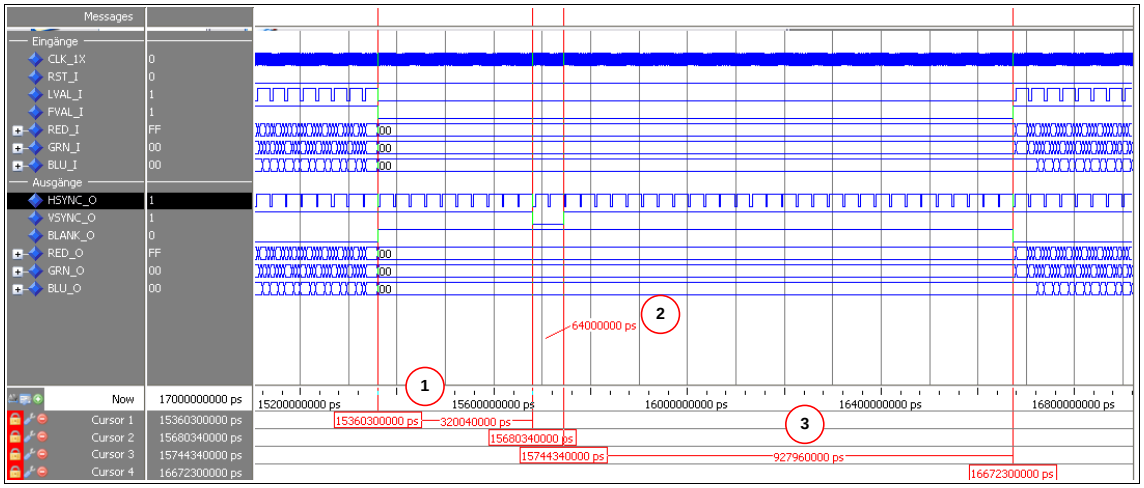

Abbildung 34: Funktionale Simulation des VGA-Controllers *VGA_CTRL_v1* – VSYNC-Generierung

Der VGA-Controller wurde auf dem Virtex-5 XC5VFX70T FPGA mit denselben Parametern wie bei der funktionalen Simulation implementiert und die tatsächlich generierten Signale mit einem Tektronix DPO 4054 Oszilloskop aufgenommen, um die Funktionalität der Komponente nachzuweisen (vgl. Abbildung 35, Abbildung 36).

In Abbildung 35 ist der Oszillograph von *LVAL_I* und *HSYNC_O* dargestellt. Die gemessenen Zeiten von Front Porch, Sync Pulse und Back Porch stimmen mit denen aus der Simulation mit Abweichungen im Nanosekundenbereich überein. In Abbildung 36 ist der Oszillograph von *FVAL_I* und *VSYNC_O* dargestellt. Die gemessenen Zeiten von Front Porch, Sync Pulse und Back Porch stimmen auch hier mit denen aus der Simulation

mit Abweichungen im Nanosekundenbereich überein. Diese Abweichungen sind auf Messungenauigkeiten zurückzuführen.

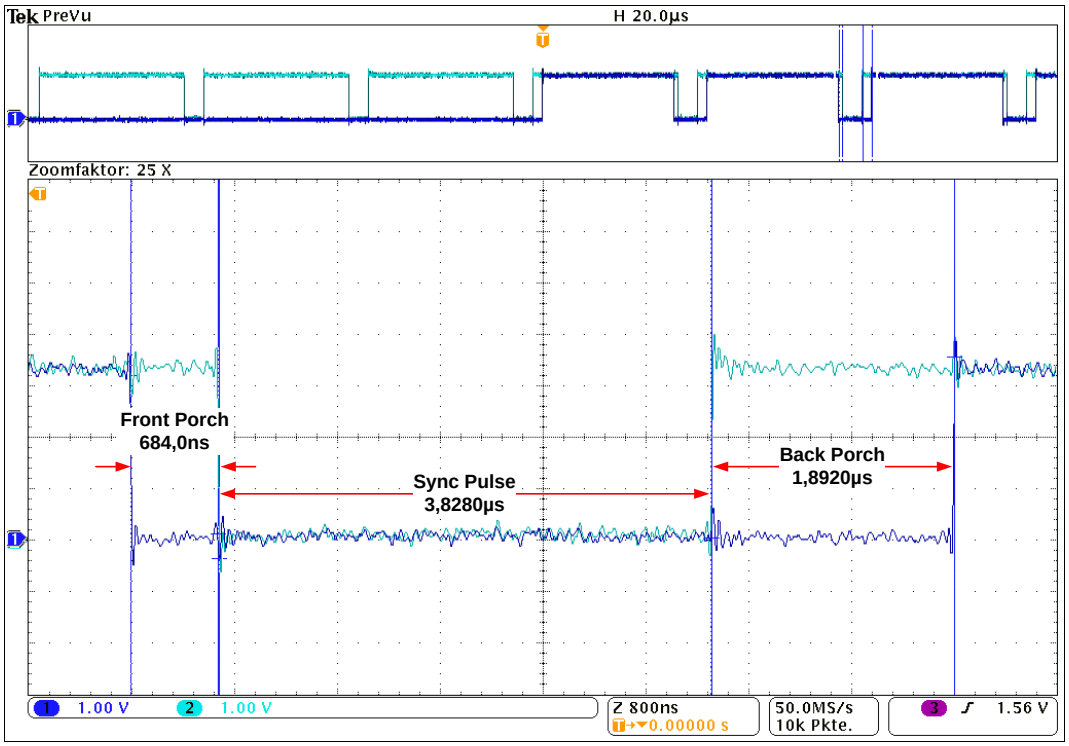

Abbildung 35: Messung des horizontalen Synchronisationssignals *HSYNC_O*

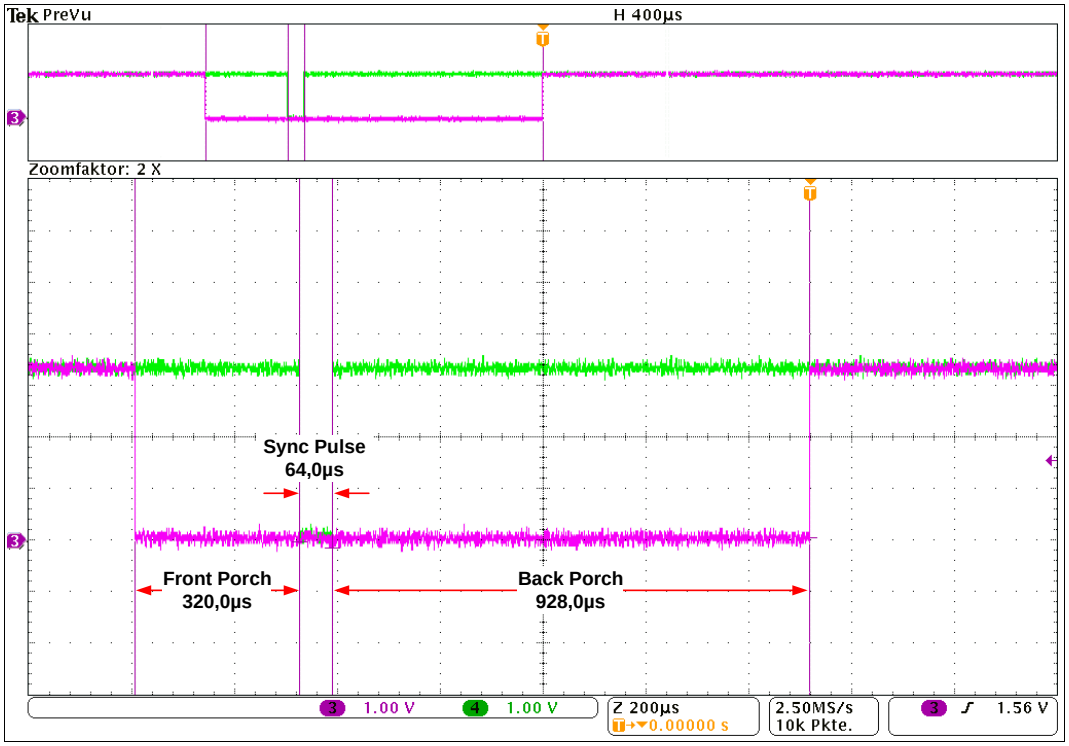

Abbildung 36: Messung des vertikalen Synchronisationssignals *VSYNC_O*

6 Parametrierbarer Testbildgenerator

Der Testbildgenerator (*TSG_v1*) erzeugt ein in 4 vertikale Farbstreifen (Rot, Grün, Blau, Weiß) unterteiltes Testbild und die zugehörigen Synchronisationssignale (Line Valid, Frame Valid). Die Bildauflösung, sowie das Timing der Synchronisationssignale lassen sich mit Generics beliebig konfigurieren (vgl. Tabelle 7). So lässt sich mit dem Testbildgenerator beispielsweise das Ausgangsverhalten einer CCD-Kamera simulieren und in Zusammenhang mit einem VGA-Controller testen (vgl. Kapitel 5).

In diesem Kapitel wird der Schaltungsentwurf des Testbildgenerators erläutert und in Kapitel 6.1 verifiziert. In Kapitel 6.2 wird der Testbildgenerator beispielhaft gemäß Signal-Timing der CCD-Kamera A301b (Kapitel 2.4) parametrisiert.

Nachfolgend werden sämtliche Generics aufgezählt und erläutert, mit denen sich der Testbildgenerator parametrisieren lässt:

Generic-Name	Typ	Beschreibung	Standardwert
H_PIXEL	natural	Anzahl Pixel je Bildzeile.	640
H_FRONT_PORCH	natural	Länge des HSYNC-Front Porch (in Pixeltaktperioden).	16
H_SYNC_PULSE	natural	Länge des HSYNC-Sync Pulse (in Pixeltaktperioden).	96
H_BACK_PORCH	natural	Länge des HSYNC-Back Porch (in Pixeltaktperioden).	48
LVAL_LOW_FIRST	boolean	Legt fest, ob das periodische LVAL-Signal mit der Low- oder High-Phase beginnen soll.	false
FVAL_BEFORE_LVAL	natural	Zeit in Pixeltaktperioden, in der FVAL vor LVAL auf High-Pegel geht.	0
V_LINES	natural	Anzahl Bildzeilen.	480
V_FRONT_PORCH	natural	Länge des VSYNC-Front Porch (in Bildzeilenperioden).	10
V_SYNC_PULSE	natural	Länge des VSYNC-Sync Pulse (in Bildzeilenperioden).	2
V_BACK_PORCH	natural	Länge des VSYNC-Back Porch (in Bildzeilenperioden).	29

Tabelle 7: Generics des Testbildgenerators *TSG_v1*

Der Testbildgenerator verfügt über folgende Ein- und Ausgänge (vgl. Abbildung 38):

Portname	Signalbreite	Beschreibung
CLK_I	1	Taktsignaleingang (Pixeltakt) für sämtliche taktrelevante Elemente der Komponente. Der Pixeltakt ist der Komponente je nach Bildauflösung und Bildrate angepasst zuzuführen.
RST_I	1	Reset-Eingang zum Zurücksetzen der Komponente.
LVAL_O	1	Horizontales Synchronisationssignal für den VGA-Controller.
FVAL_O	1	Vertikales Synchronisationssignal für den VGA-Controller.
RED_O	8	Getakteter Ausgang des roten Pixelanteils des Testbildes.
GRN_O	8	Getakteter Ausgang des grünen Pixelanteils des Testbildes.
BLU_O	8	Getakteter Ausgang des blauen Pixelanteils des Testbildes.

Abbildung 37: Ports des Testbildgenerators *TSG_v1*

Der grundlegende Aufbau des Testbildgenerators besteht aus 2 Zählwerken (*HPOS_cnt*, VPOS_cnt), kombinatorischer Logik zur Steuerung der Zählwerke und Generierung der Ausgangssignale und Ausgangsregistern für synchrone Signalausgänge. Die einzelnen Blöcke der Komponente haben folgende Funktionen:

- *HPOS_cnt*
  Zählwerk für das Abzählen einer Bildzeile. *HPOS_cnt* beginnt bei der steigenden Flanke von *CLK_I* aufwärts zu zählen und wird von *CNT_comp_logic* wieder zurückgesetzt, sobald eine Bildzeilenperiode abgezählt wurde.

- *VPOS_cnt*
  Zählwerk für das Abzählen von Bildzeilen. Nach jeder von *HPOS_cnt* abgezählten Bildzeile setzt *CNT_comp_logic* einen Takt lang das Enable-Signal für *VPOS_cnt*, sodass dieses bei steigender Flanke von *CLK_I* aufwärts zählt. Hat *VPOS_cnt* sämtliche Bildzeilen eines Vollbildes abgezählt, wird es von *CNT_comp_logic* wieder zurückgesetzt.

- *CNT_comp_logic*
  Komparatorlogik für das Setzen des Enable-Signals für *VCount_cnt* und das Zurücksetzen beider Zählwerke.

- *LVAL_GEN*
  Komparatorschaltung für das Setzen des LVAL-Signals (*LVAL_sig*). Aus den Zählständen beider Zählwerke wird *LVAL_sig* kombinatorisch gesetzt bzw. wieder entfernt.

- *FVAL_GEN*
  Komparatorschaltung für das Setzen des FVAL-Signals (*FVAL_sig*). Aus dem Zählstand des Zählwerkes *VPOS_cnt* wird *FVAL_sig* kombinatorisch gesetzt bzw. wieder entfernt.

- *PIXEL_GEN*
  Komparatorlogik für die Generierung der Farbpixel des Testbildes. Aus dem Zählstand des Zählwerks *HPOS_cnt* wird die horizontale und aus *VPOS_cnt* die vertikale Pixelposition ermittelt und der entsprechende Farbwert an den Ausgang gelegt.

- *OUTPUT_REG*
  Getaktetes Ausgangsregister. Sämtliche Ausgangssignale werden taktsynchron herausgeführt und für die Dauer einer Taktperiode gehalten.

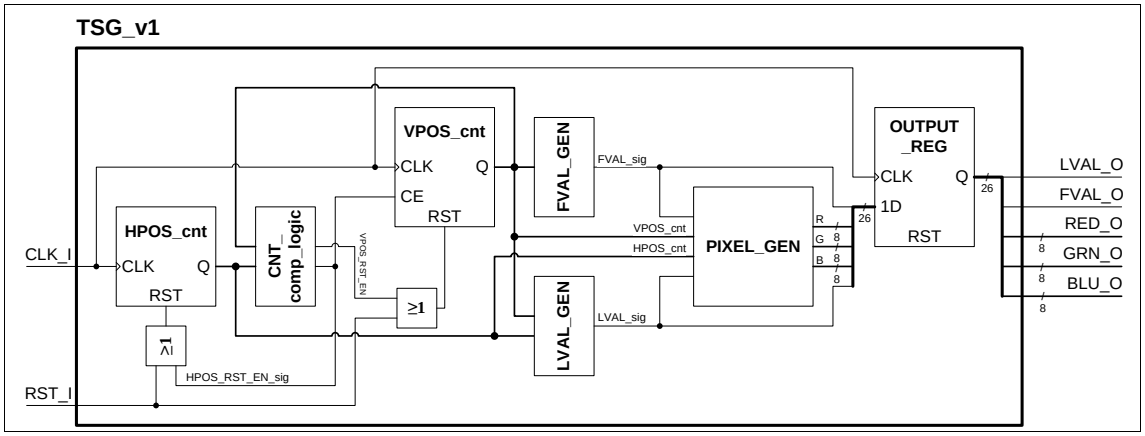

Abbildung 38: Blockschaltbild des Testbildgenerators *TSG_v1*

6.1 Simulation und Messergebnisse

Der Testbildgenerator wurde zur Verifikation des Schaltungsentwurfs einer funktionalen Simulation unterzogen. Dazu wurde der Testbildgenerator in einer Testbench für 4 Testfälle simuliert. Anschließend wurde die Komponente auf dem Virtex-5 XC5VFX70T FPGA implementiert und die generierten Synchronisationssignale *LVAL_O* und *FVAL_O* mit einem Tektronix DPO 4054 Oszilloskop aufgenommen.

Nachfolgend werden für die jeweiligen Testfälle zunächst die Simulations- und anschließend die Messergebnisse präsentiert und miteinander verglichen. Bei allen Testfällen wurden die Generics `LVAL_LOW_FIRST` und `FVAL_BEFORE_LVAL` variiert, die Restlichen wurden beim Standardwert belassen (vgl. Tabelle 7).

Testfall 1 (vgl. Abbildung 39):
f_{CLK_I} = 25MHz, `LVAL_LOW_FIRST => false`, `FVAL_BEFORE_LVAL => 0`

(1) Die Periodizität von *LVAL_O* beginnt mit der High-Phase
→ `LVAL_LOW_FIRST => false`
LVAL_O und *FVAL_O* gehen zeitgleich auf High-Pegel
→ `FVAL_BEFORE_LVAL => 0`

(2) Die High-Phase von *LVAL_O* beträgt 25600ns
→ 25600ns*f_{CLK_I} = 640 = `H_PIXEL`

(3) Die Low-Phase von *LVAL_O* beträgt 6400ns
→ 6400ns*f_{CLK_I} = 160 = `H_FRONT_PORCH+H_SYNC_PULSE+H_BACK_PORCH`

(4) *LVAL_O* wird periodisch weiter generiert

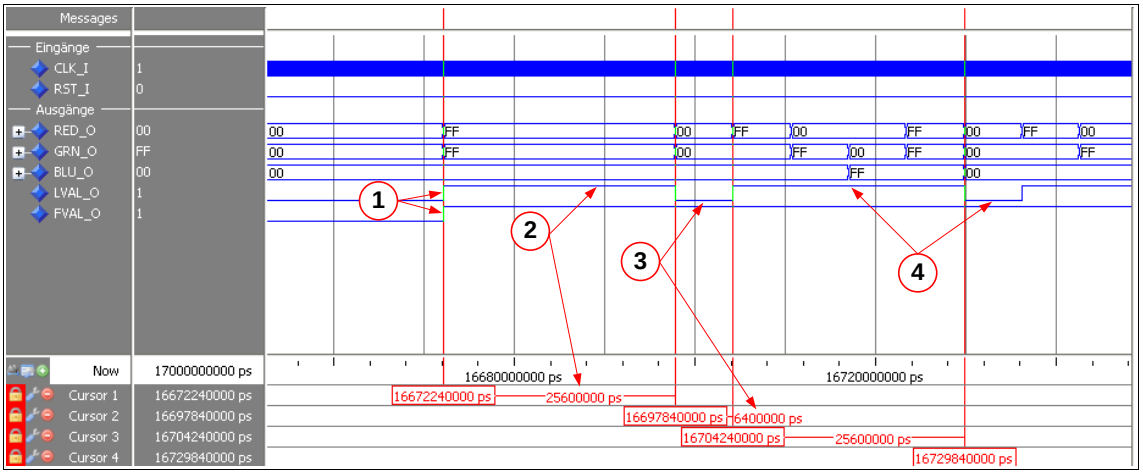

Abbildung 39: Funktionale Simulation des Testbildgenerators *TSG_v1* - Testfall 1

Die Messung (Abbildung 40) stimmt bis auf Abweichungen im Nanosekundenbereich bei LVAL-High- und Low-Zeit mit dem Simulationsergebnis überein (vgl. Abbildung 40 → (2), (3)).

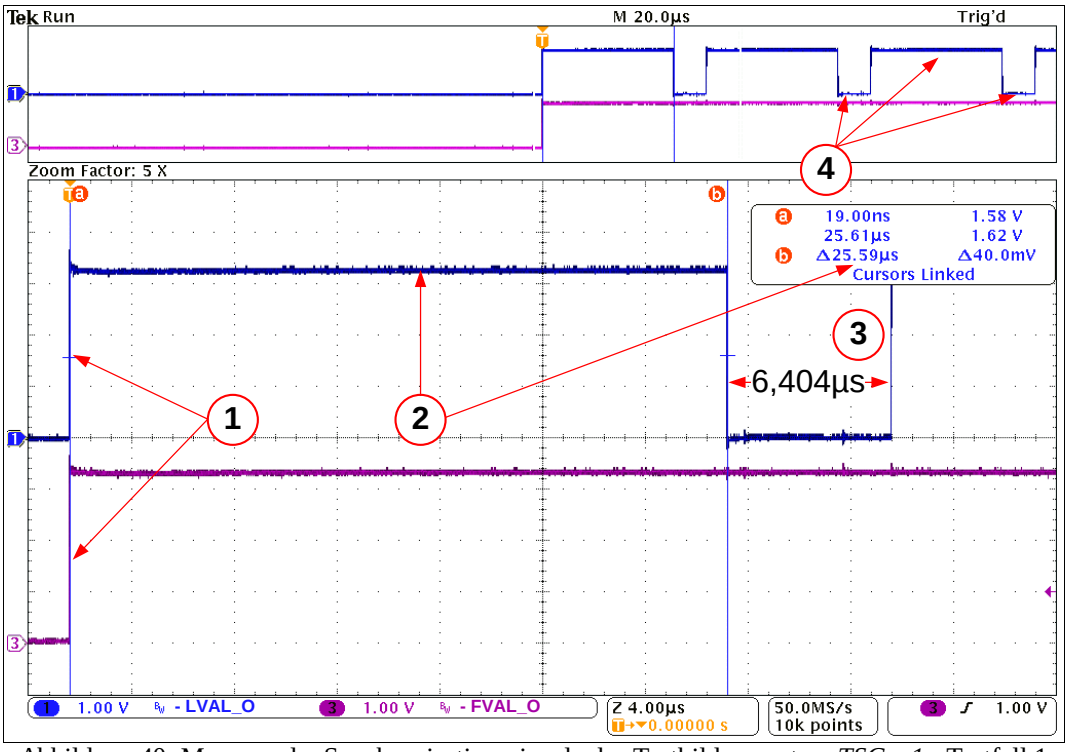

Abbildung 40: Messung der Synchronisationssignale des Testbildgenerators *TSG_v1* - Testfall 1

Testfall 2 (vgl. Abbildung 41):
f_{CLK_I} = 25MHz, `LVAL_LOW_FIRST => true`, `FVAL_BEFORE_LVAL => 0`

(1) Die Periodizität von *LVAL_O* beginnt mit der Low-Phase
→ `LVAL_LOW_FIRST => true`
LVAL_O geht 6400ns später als *FVAL_O* auf High-Pegel
→ 6400ns*f_{CLK_I} = 160 = `H_FRONT_PORCH+H_SYNC_PULSE+H_BACK_PORCH`
→ `FVAL_BEFORE_LVAL => 0`

(2) Die High-Phase von *LVAL_O* beträgt 25600ns
→ 25600ns*f_{CLK_I} = 640 = `H_PIXEL`

(3) Die Low-Phase von *LVAL_O* beträgt 6400ns
→ 6400ns*f_{CLK_I} = 160 = `H_FRONT_PORCH+H_SYNC_PULSE+H_BACK_PORCH`

(4) *LVAL_O* wird periodisch weiter generiert

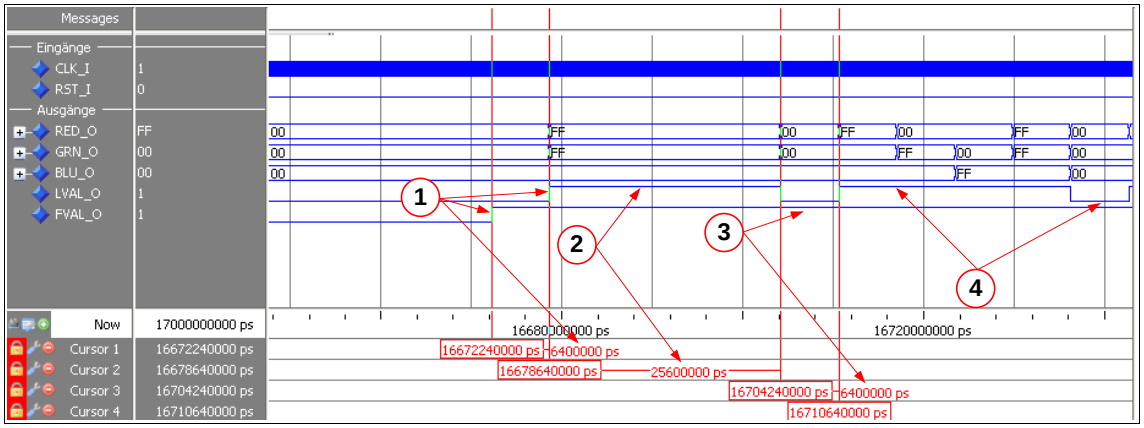

Abbildung 41: Funktionale Simulation des Testbildgenerators *TSG_v1* – Testfall 2

Die Messung (Abbildung 42) stimmt bis auf Abweichungen im Nanosekundenbereich bei LVAL-High- und Low-Zeit mit dem Simulationsergebnis überein (vgl. Abbildung 42 → (2), (3)).

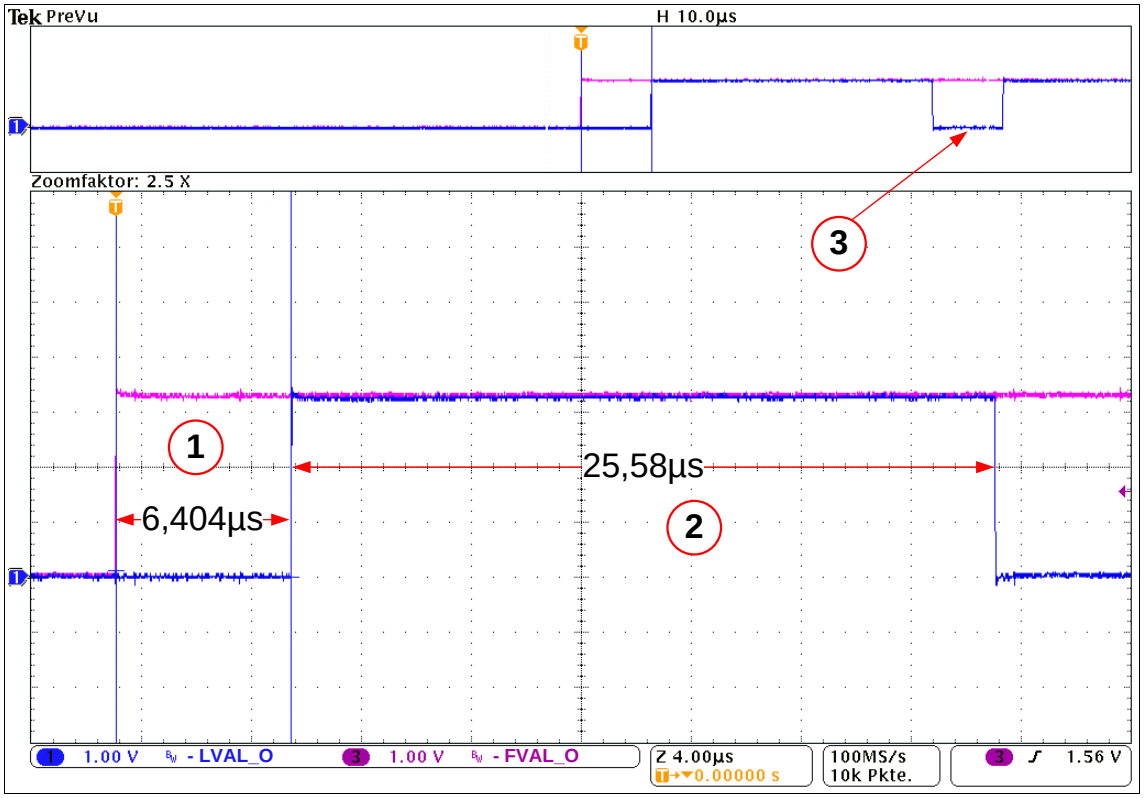

Abbildung 42: Messung der Synchronisationssignale des Testbildgenerators *TSG_v1* – Testfall 2

Testfall 3 (vgl. Abbildung 43):
f_{CLK_I} = 25MHz, `LVAL_LOW_FIRST => false`, `FVAL_BEFORE_LVAL => 200`

(1) Die zeitliche Differenz zwischen der steigenden Flanke von *FVAL_O* und *LVAL_O* geht beträgt 8µs
→ 8µs*f_{CLK_I} = 200
→ `FVAL_BEFORE_LVAL => 200`
Die Periodizität von *LVAL_O* beginnt nach Verstreichen der zeitlichen Differenz mit der High-Phase
→ `LVAL_LOW_FIRST => false`

(2) Die High-Phase von *LVAL_O* beträgt 25600ns
→ 25600ns*f_{CLK_I} = 640 = `H_PIXEL`

(3) Die Low-Phase von *LVAL_O* beträgt 6400ns
→ 6400ns*f_{CLK_I} = 160 = `H_FRONT_PORCH+H_SYNC_PULSE+H_BACK_PORCH`

(4) *LVAL_O* wird periodisch weiter generiert

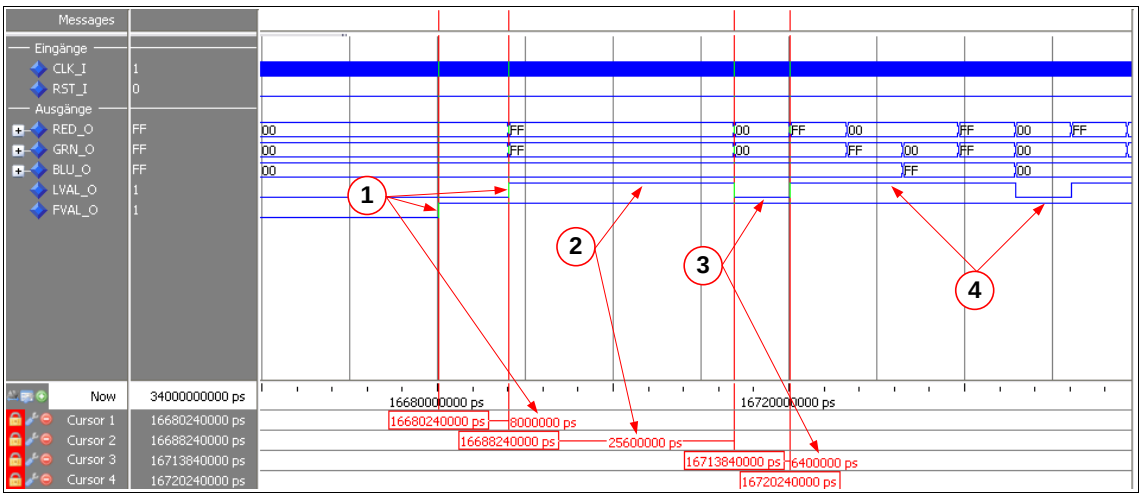

Abbildung 43: Funktionale Simulation des Testbildgenerators *TSG_v1* – Testfall 3

Die Messung (Abbildung 44) stimmt bis auf Abweichungen im Nanosekundenbereich bei der LVAL-High-Zeit mit dem Simulationsergebnis überein (vgl. Abbildung 44 → (2)).

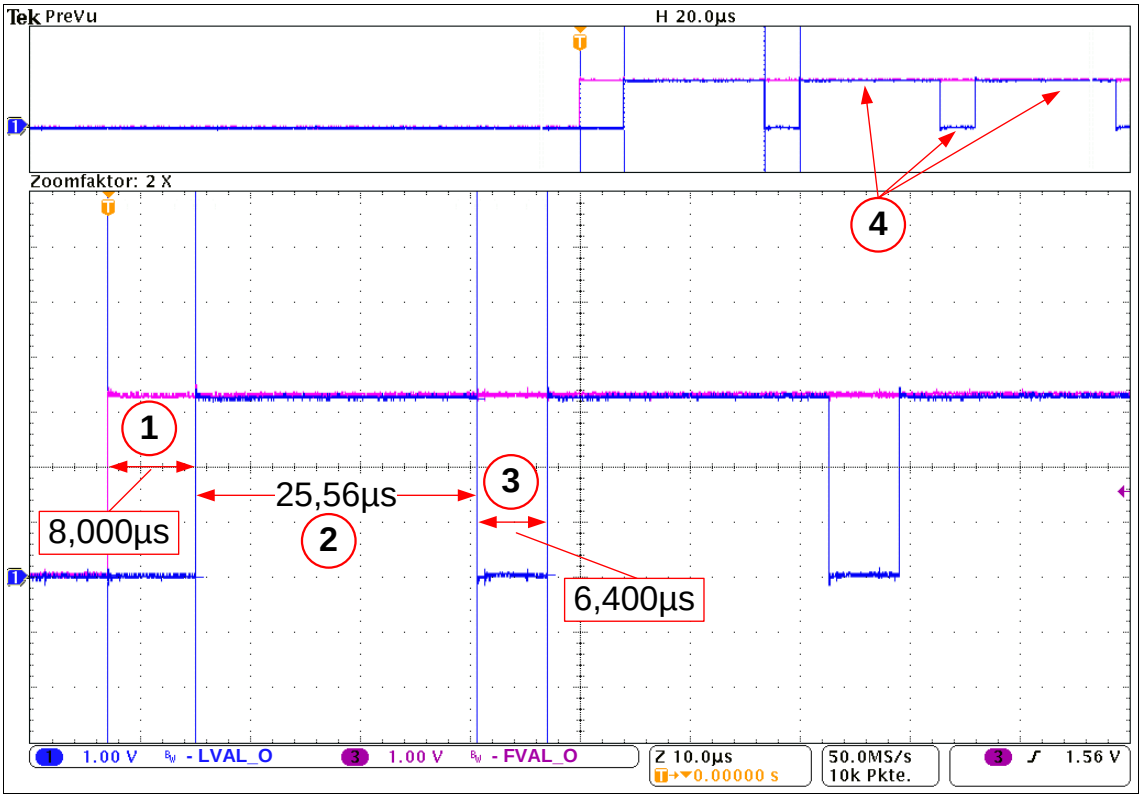

Abbildung 44: Messung der Synchronisationssignale des Testbildgenerators *TSG_v1* – Testfall 3

Testfall 3 (vgl. Abbildung 45):
f_{CLK_I} = 25MHz, `LVAL_LOW_FIRST => true`, `FVAL_BEFORE_LVAL => 200`

(1) Die zeitliche Differenz zwischen der steigenden Flanke von *FVAL_O* und *LVAL_O* geht beträgt 14,4µs
→ 14,4µs*f_{CLK_I} = 360 = 200 + 160 (`H_FRONT_PORCH+H_SYNC_PULSE+H_BACK_PORCH`)
→ `FVAL_BEFORE_LVAL => 200`
Die Periodizität von *LVAL_O* beginnt nach Verstreichen der zeitlichen Differenz mit der Low-Phase
→ `LVAL_LOW_FIRST => true`

(2) Die High-Phase von *LVAL_O* beträgt 25600ns
→ 25600ns*f_{CLK_I} = 640 = `H_PIXEL`

(3) Die Low-Phase von *LVAL_O* beträgt 6400ns
→ 6400ns*f_{CLK_I} = 160 = `H_FRONT_PORCH+H_SYNC_PULSE+H_BACK_PORCH`

(4) *LVAL_O* wird periodisch weiter generiert

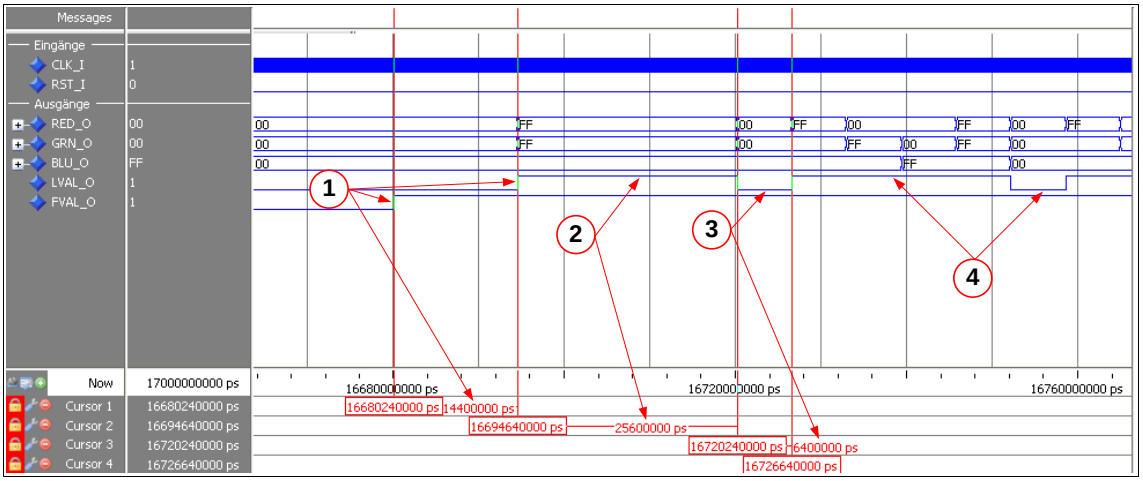

Abbildung 45: Funktionale Simulation des Testbildgenerators *TSG_v1* – Testfall 4

Die Messung (Abbildung 46) stimmt gänzlich mit dem Simulationsergebnis überein (vgl. Abbildung 46 → (2)).

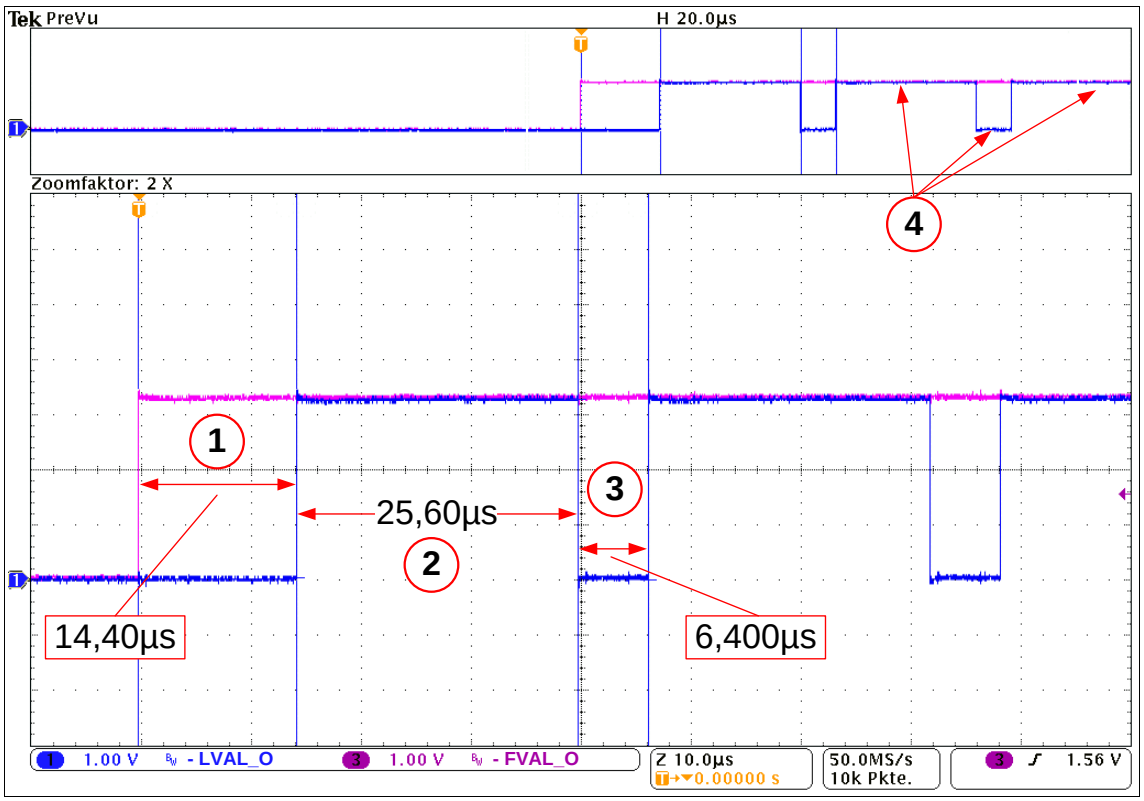

Abbildung 46: Messung der Synchronisationssignale des Testbildgenerators *TSG_v1* – Testfall 4

Aufgrund des Parameters `LVAL_LOW_FIRST => false` bei den Testfällen 1 und 3 geht *FVAL_O* 6,4µs später als *LVAL_O* auf Low-Pegel (vgl. Abbildung 47)

→ 0,64µs*f_{CLK_I} = 160 = `H_FRONT_PORCH+H_SYNC_PULSE+H_BACK_PORCH`

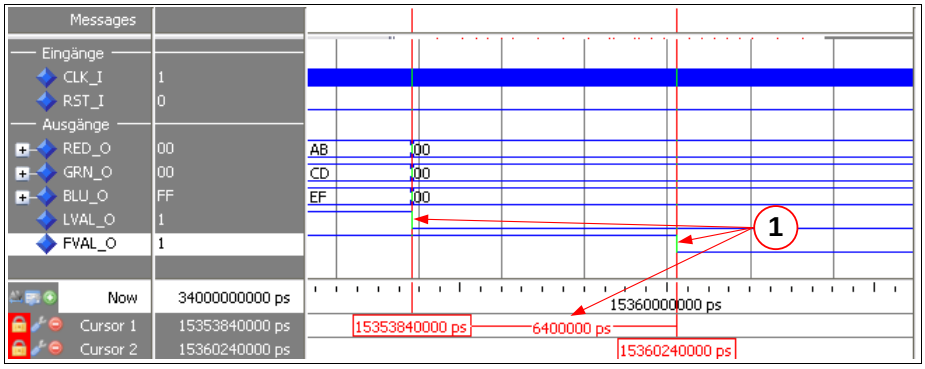

Abbildung 47: Funktionale Simulation des Testbildgenerators *TSG_v1* – Testfälle 1 und 3

Die Messung (vgl. Abbildung 48) stimmt gänzlich mit der Simulation überein.

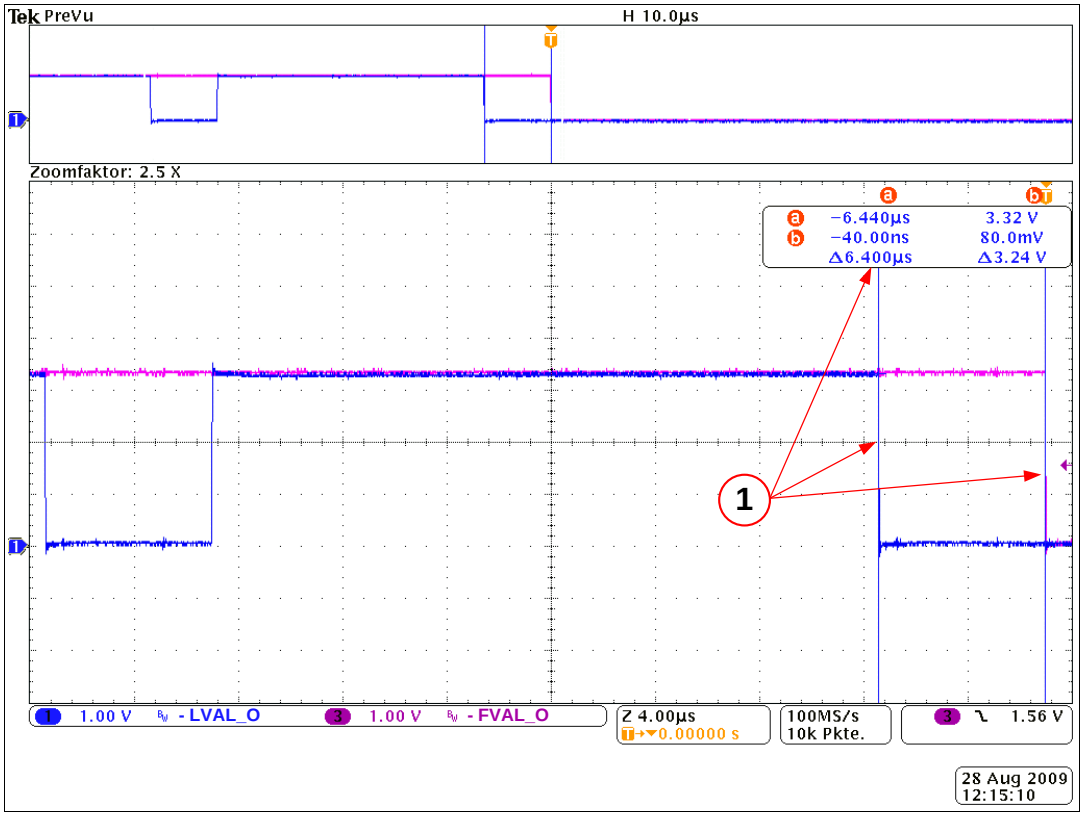

Abbildung 48: Messung der Synchronisationssignale des Testbildgenerators *TSG_v1* – Testfälle 1 und 3

Bei den Testfällen 2 und 4 gehen *LVAL_O* und *FVAL_O* sowohl in der Simulation (vgl. Abbildung 49), als auch bei der Messung (vgl. Abbildung 50) zeitgleich auf Low-Pegel.

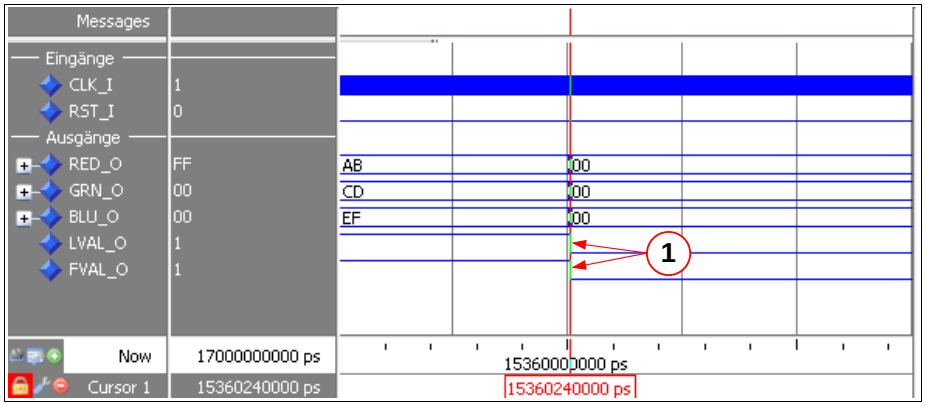

Abbildung 49: Funktionale Simulation des Testbildgenerators *TSG_v1* – Testfälle 2 und 4

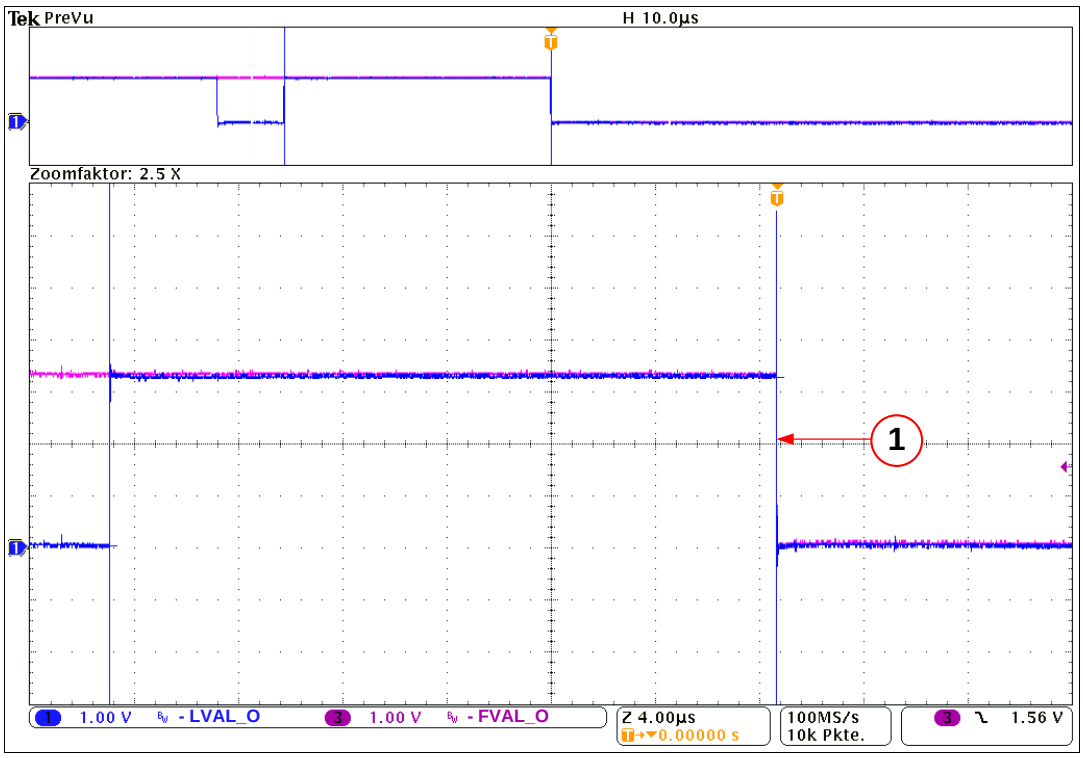

Abbildung 50: Messung der Synchronisationssignale des Testbildgenerators *TSG_v1* – Testfälle 1 und 3

6.2 Parametrierungsbeispiel des Testbildgenerators

Der Testbildgenerator soll im Folgenden an das Ausgangsverhalten der CCD-Kamera A301b angepasst, parametriert werden. Folgende Daten wurden dem Datenblatt [8] entnommen:

- Pixeltakt → 36MHz effektiv (bei 2 Pixeln je 18MHz-Pixeltakt)
- Anzahl Pixel je Bildzeile → `H_PIXEL => 658`
- Anzahl Bildzeilen je Vollbild → `V_LINES => 494`
- Maximale Bildwiederholrate → 80fps

Im Datenblatt ist zusätzlich ein Diagramm über die von der CCD-Kamera erzeugten Synchronisationssignale Frame Valid und Line Valid enthalten (vgl. Tabelle 8), dem folgende Informationen entnommen werden:

(1) Die LVAL-Low-Zeit beträgt 5,8µs, dies entspricht 5,8µs*36MHz = 208,8 ≈ 208 Pixelperioden
→ `H_FRONT_PORCH + H_SYNC_PULSE + H_BACK_PORCH => 208.`

(2) Die LVAL-High-Zeit beträgt 18,3µs, dies entspricht 18,3µs*36MHz = 658,8 ≈ 658 Pixelperioden. Die dem Datenblatt entnommene Anzahl Pixel je Bildzeile kann dadurch nachvollzogen werden.

(3) FVAL und LVAL gehen synchron auf Low-Pegel, d.h. LVAL beginnt aufgrund der Periodizität mit der Low-Phase
→ `LVAL_LOW_FIRST => true.`
Daher ist der zeitlichen Differenz zwischen steigender Flanke von FVAL und der steigenden Flanke von LVAL (19,0µs) die LVAL-Low-Zeit (5,8µs) abzuziehen, woraus sich folgendes ergibt:
(19,0µs-5,8µs)*36MHz = 475,2 ≈ 475 Pixelperioden
→ `FVAL_BEFORE_LVAL => 475.`

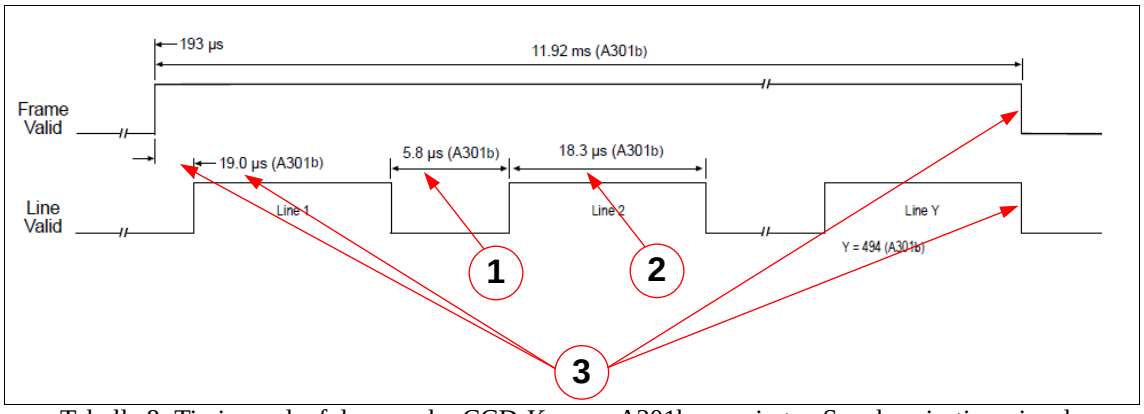

Tabelle 8: Timingverlauf der von der CCD-Kamera A301b generierten Synchronisationssignale

Die FVAL-Low-Zeit hängt von der parametrierbaren Bildrate n ab, da diese festlegt, wie viele FVAL-High-Perioden (= Vollbilder) innerhalb einer Sekunde vorkommen. Der effektive Pixeltakt ergibt sich aus der folgenden Gleichung:

$$f_{Pixel} = (F\_B4\_L + (H\_HI + H\_LO) * (V\_HI + V\_LO)) * n \quad (1)$$

mit

f_{Pixel}	:	effektiver Pixeltakt
F_B4_L	:	`FVAL_BEFORE_LVAL`
H_HI	:	`H_LINES`
H_LO	:	`H_FRONT PORCH + H_SYNC_PULSE + H_BACK_PORCH` (2)
V_HI	:	`V_LINES`
V_LO	:	`V_FRONT_PORCH + V_SYNC_PULSE + V_BACK_PORCH`
n	:	Bildrate

Wird Gleichung (1) nach V_LO umgestellt, ergibt sich folgende Gleichung, nach der sich die FVAL-Low-Zeit (gemessen in Bildzeilenperioden) errechnen lässt:

$$V\_LO = (((f_{Pixel} / n) - F\_B4\_L) / (H\_HI + H\_LO)) - V\_HI. \quad (3)$$

Die oben ermittelten Zahlenwerte werden in Gleichung (3) eingesetzt, womit sich für die FVAL-Low-Zeit folgender Zahlenwert ergibt:

$$V\_LO = (((36MHz / 80*s^{-1}) - 475) / (658 + 208)) - 494 = 25{,}08 \approx 25. \quad (4)$$

Die Aufteilung der ermittelten LVAL- und FVAL-Low-Zeiten auf die Generics `H_FRONT PORCH`, `H_SYNC_PULSE`, `H_BACK_PORCH` bzw. `V_FRONT_PORCH`, `V_SYNC_PULSE`, `V_BACK_PORCH` kann [30] entnommen werden.

7 Syntheseergebnis und Timing-Simulation

In diesem Kapitel wird der Schaltungsentwurf für die FPGA-Konfiguration der Erprobungsplattform erläutert und einer Timing-Simulation unterzogen. Das Syntheseergebnis des Schaltungsentwurfs wird im Anschluss aufgelistet.

Der Schaltungsentwurf für die FPGA-Konfiguration der Erprobungsplattform besteht aus mehreren zusammengesetzten Komponenten (vgl. Abbildung 51). Die Komponenten aus den vorangegangenen Kapiteln wurden grün gefärbt, die neu hinzugekommenen Komponenten rot gestrichelt umrahmt.

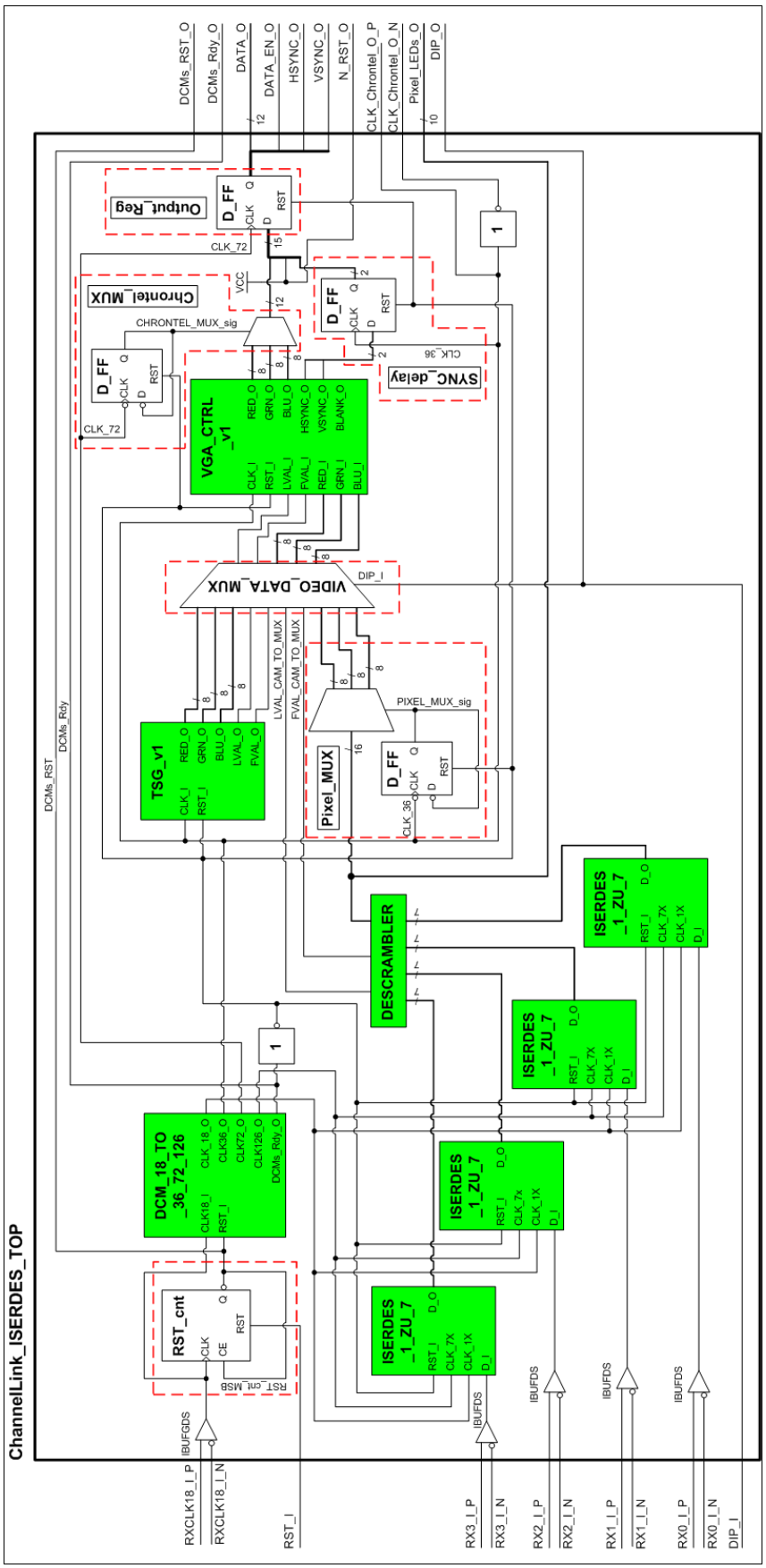

Abbildung 51: Blockschaltbild der FPGA-Konfiguration der Erprobungsplattform

Im Folgenden werden die neu hinzugefügten Komponenten erläutert (die bekannten Komponenten werden auf die jeweiligen Kapitel dieser Arbeit verwiesen):

- *IBUFGDS/IBUFDS*
  Eingangsbuffer für differentielle Takt-/Eingangssignale (vgl. Kapitel 3.1).

- *DCM_18_TO_36_72_126*
  DCM-Konfiguration zur Taktsignalgenerierung (vgl. Kapitel 4.3.1).

- *RST_cnt*
  19-Bit-breites Zählwerk für die Kompensation des Prellverhaltens des Reset-Tasters (vgl. Kapitel 2.1 → (6) (CPU Reset) Pushbutton). Bei Betätigung des Reset-Tasters wird das Zählwerk *RST_cnt* taktsynchron auf 0_{dez} zurückgesetzt. Sobald das Reset-Signal nicht mehr anliegt, beginnt *RST_cnt*, taktsynchron aufwärts zu zählen, bis das MSB des Zählwerks (*RST_cnt_MSB*) 1_{bin} ist. Das invertierte MSB dient als Enable-Signal des Zählwerks und als Reset-Signal für die DCM-Konfiguration (*DCM_18_TO_18_36_72_126*). Die DCM-Konfiguration wird für eine (Zähl-) Dauer von $2^{\text{Zählerbreite-1}}/f_{CLK\_I} = 2^{19-1}/18\text{MHz} = 14{,}56\text{ms}$ im Reset-Zustand gehalten. Die durch das Prellen des Tasters hervorgerufenen Pulse bewirken das mehrmalige Zurücksetzen des Zählwerks, wodurch sich zwar die Reset-Dauer verlängert, die DCM-Konfiguration aber ein prellfreies Reset-Signal erhält.

- *ISERDES_1_ZU_7*
  Kaskadierung von 2 dedizierten Serien-Parallel-Umsetzern der Virtex-5 FPGA-Reihe (vgl. Kapitel 4.3).

- *DESCRAMBLER*
  Logik zur Neustrukturierung des empfangen Datenstroms aus dem Channel Link Interface (vgl. Kapitel 4.2).

- *TSG_v1*
  Parametrisierbarer Testbildgenerator (vgl. Kapitel 6).

- *Pixel_MUX*
  Die mit der Erprobungsplattform eingesetzte CCD-Kamera A301b liefert je Pixeltaktperiode (18MHz) 2 Bildpixel im Datenstrom (vgl Kapitel 4.1). Die empfangenen 2 Bildpixel werden bei doppeltem Pixeltakt (36MHz) von einem Multiplexer umgeschaltet. Das Multiplexer-Steuersignal wird in einem Datenflipflop mit Rückführung des invertierten Ausgangs auf den Eingang generiert.

- *VIDEO_DATA_MUX*
  Umschalter zwischen Bilddaten und Synchronisationssignalen aus dem Testbildgenerator *TSG_v1* und der CCD-Kamera. Das Steuersignal kommt von einem zwischen High- und Low-Pegel umschaltbarem DIP-Schalter (vgl. Kapitel 2.1 → (3) DIP-Switches).

- VGA_ctrl_v1

VGA-Controller für die visuelle Kontrolle des kontinuierlichen Kameradatenstroms (vgl. Kapitel 5).

- *Chrontel_MUX*
  Umschalter für die partielle Zuführung der Bilddaten zum Videocontroller CH7301C nach einem bestimmten Dateneingangsformat (vgl. Kapitel 2.2).

- *SYNC_delay*
  D-Flipflops zur Verzögerung der von *VGA_CTRL_v1* erzeugten Synchronisationssignale um eine 36MHz-Taktperiode. Da bei Pixeltakt (18MHz) jeweils 2 Pixel eintreffen und diese bei 36MHz im *Pixel_MUX* umgeschaltet werden, werden die von *VGA_CTRL_v1* erzeugten Synchronisationssignale um eine 36MHz-Taktperiode verzögert, um synchron zu den Bilddaten herausgeführt zu werden.

- *Output_Reg*
  Ausgangsregister für synchron anliegende Ausgangssignale.

Der in Abbildung 51 dargestellte Schaltungsentwurf wurde einer Timing-Simulation unterzogen, um neben der reinen Funktionalität auch die physikalischen Signalverzögerungen von Logik- und Verdrahtungsressourcen zu simulieren und das Design zu verifizieren. In Abbildung 52 sind die durch *IBUFDS*-Komponenten hervorgerufenen Signalverzögerungszeiten auf den differentiellen Takt- (1) und Datenleitungen (2)-(5) markiert. Diese liegen bis auf leichte Abweichungen in der Größenordnung 5-50ps im gleichen Bereich.

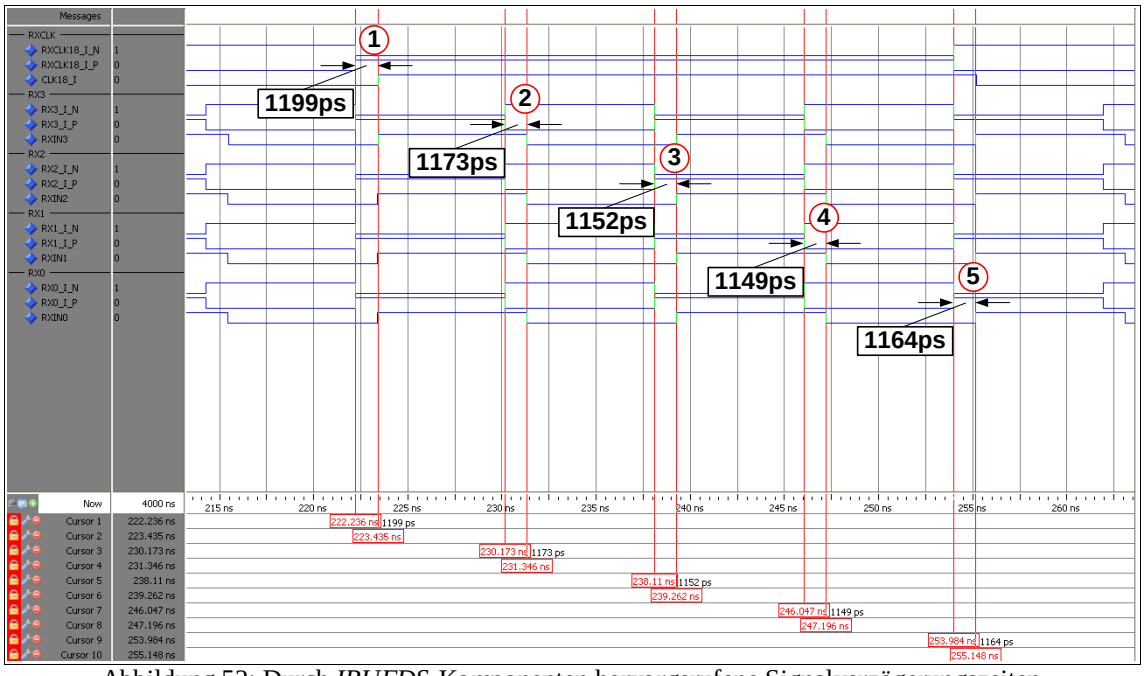

Abbildung 52: Durch *IBUFDS*-Komponenten hervorgerufene Signalverzögerungszeiten

In Abbildung 53 richtet sich das Hauptaugenmerk auf die *ISERDES*-Blöcke und den durch Verdrahtungslogik entstehende Signalverzögerung. Die Dateneingänge der *ISERDES*-Komponenten wurden für die Timing-Simulation mit jeweils nach einer Bittaktperiode alternierenden Signalen stimuliert. Aus der Simulation geht folgendes hervor:

(1) Der Bittakt kommt bei allen *ISERDES*-Instanzen mit Abweichungen im Pikosekundenbereich zeitgleich an (*RX1_ISERDES* und *RX0_ISERDES* sind hier nicht aufgeführt).

(2) Pixel- und Bittakt kommen aufgrund der internen Verdrahtungswege so bei den *ISERDES*-Instanzen an, dass die steigende Flanke von Pixel- und die fallende Flanke von Bittakt mit Abweichungen im Pikosekundenbereich in Phase liegen (vgl. Kapitel 4.3.1). Die steigende Flanke des Bittaktes liegt hinter der Mitte des Datenauges, bei ca. 60% der Datenauges, sodass die FPGA-spezifischen Setup- (T_{ISDCK_D} = 0,39ns) und Hold-Zeiten (T_{ISCKD_D} = -0,12ns) der *ISERDES*-Dateneingänge (*D*) eingehalten werden (vgl. [ds202]).

(3) Der seriell eintreffende Datenstrom '0','1','0','1','0','1','0' am Dateneingang *D_I* von *RX3_ISERDES* wird in „Zyklus n“ eingetaktet und liegt 5 Bittakte nach Beendigung des Zyklus parallel am Ausgang *D_O*.

(4) Der seriell eintreffende Datenstrom '1','0','1','0','1','0','1' am Dateneingang *D_I* von *RX2_ISERDES* wird in „Zyklus n“ eingetaktet und liegt ebenfalls 5 Bittakte nach Beendigung des Zyklus parallel am Ausgang *D_O*.

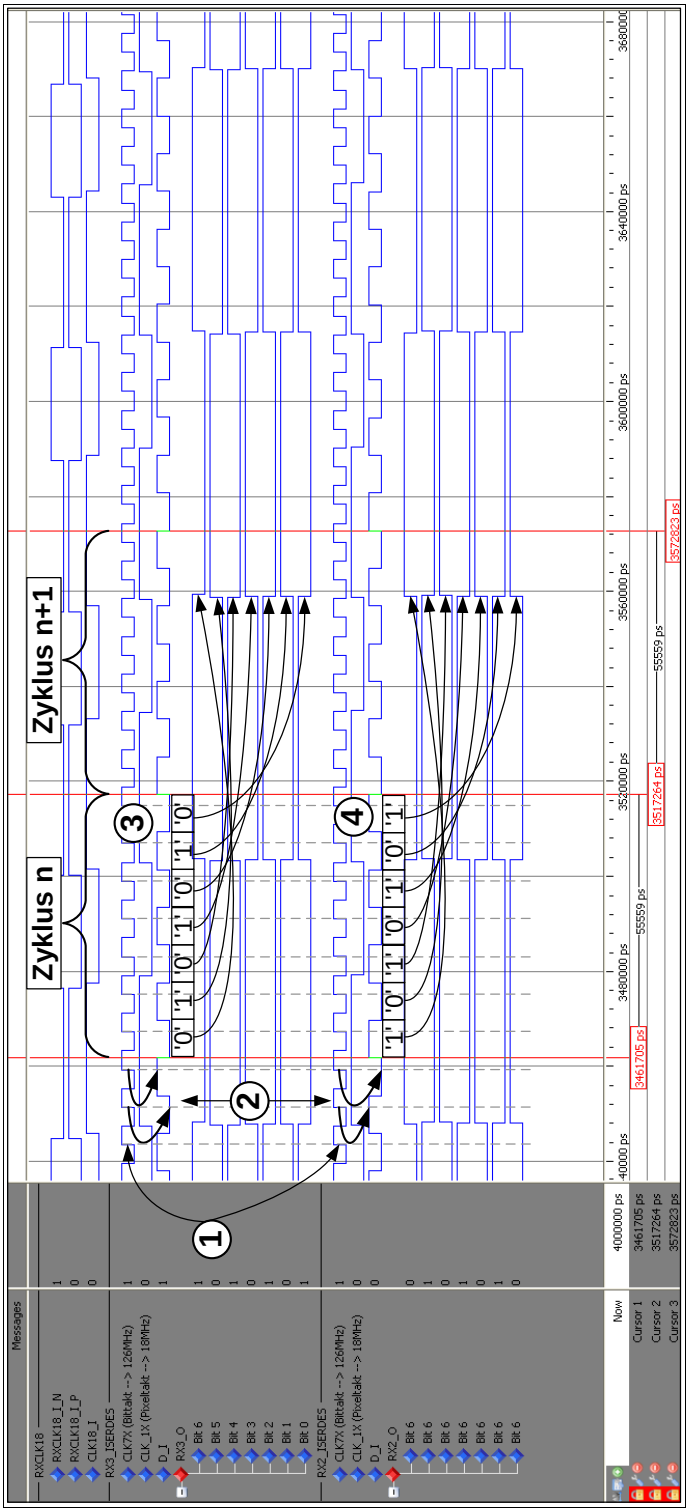

Abbildung 53: Durch Verdrahtung hervorgerufene Signalverzögerung

Der Schaltungsaufbau für die FPGA-Konfiguration der Erprobungsplattform wurde mit der Entwicklungsumgebung Xilinx ISE 10.1 synthetisiert. Das Syntheseergebnis ist in Tabelle 9 aufgelistet.

Erforderliche Logik	Anzahl	insgesamt verfügbar
Slice Register	135	44800
Slice LUTs	175	44800
LUT-FF-Paare	212	109
Ein-/Ausgänge	45	640
BUFG/BUFGCTRLs	5	32
DCM_ADVs	2	12
Max. Eingangstaktfrequenz	112,918MHz	-

Tabelle 9: Ressourcenbedarf und maximale Eingangstaktfrequenz des gesamten Projektaufbaus (vgl. Abbildung 51)

8 SoC-Konfiguration mit dem MicroBlaze-Soft-Mikroprozessorkern

System-**o**n-**C**hip (SoC) beschreibt die Integration aller Teilfunktionen eines Systems auf einem Chip und vereint Performanz nebenläufiger Hardware- und Flexibilität sequentieller Softwaremodule.

Das Virtex-5 FPGA XC5VFX70T verfügt über einen integrierten PowerPC 440 Hardware-Mikroprozessorblock und kann zusätzlich mit Soft-Mikroprozessorkernen wie dem MicroBlaze konfiguriert werden (vgl. Kapitel 2.1, [11], [22], [35]). Solche FPGA-basierten SoC-Plattformen lassen sich um Softcore-Peripheriemodule, den **I**ntellecutal **P**roperty Cores (*IP*) erweitern. Komplexe Algorithmen mit unverhältnismäßig hohem Hardwareimplementieraufwand werden in Softwaremodulen abgearbeitet, zeitintensive, sich wiederholende Algorithmen wiederum in Hardwaremodule (*IP Cores*) ausgelagert.

Die in Kapitel 7 vorgestellte Komponente *ChannelLink_ISERDES_TOP* wurde mit folgenden Zielen in einer SoC-Konfiguration instanziiert (vgl. Abbildung 54):

- Keine Prozessorlast bei der Serien-Parallel-Umsetzung des kontinuierlichen Channel Link-Datenstroms und Ansteuerung des Videocontrollers CH7301C zur Anzeige des kontinuierlichen Kamerabilddatenstroms aufgrund des nebenläufigen Designs des Hardwaremoduls.
- Parametrierung des Videocontrollers CH7301C und Steuerung der Komponente *ChannelLink_ISERDES_TOP* in Software.

In Kapitel 8.1 werden der Aufbau der in Abbildung 54 dargestellten SoC-Konfiguration und des Softcore-Peripheriemoduls *eval_platform* erläutert. Die Parametrierung des in Kapitel 2.2 vorgestellten Videocontrollers CH7301C wird in Kapitel 8.2 behandelt.

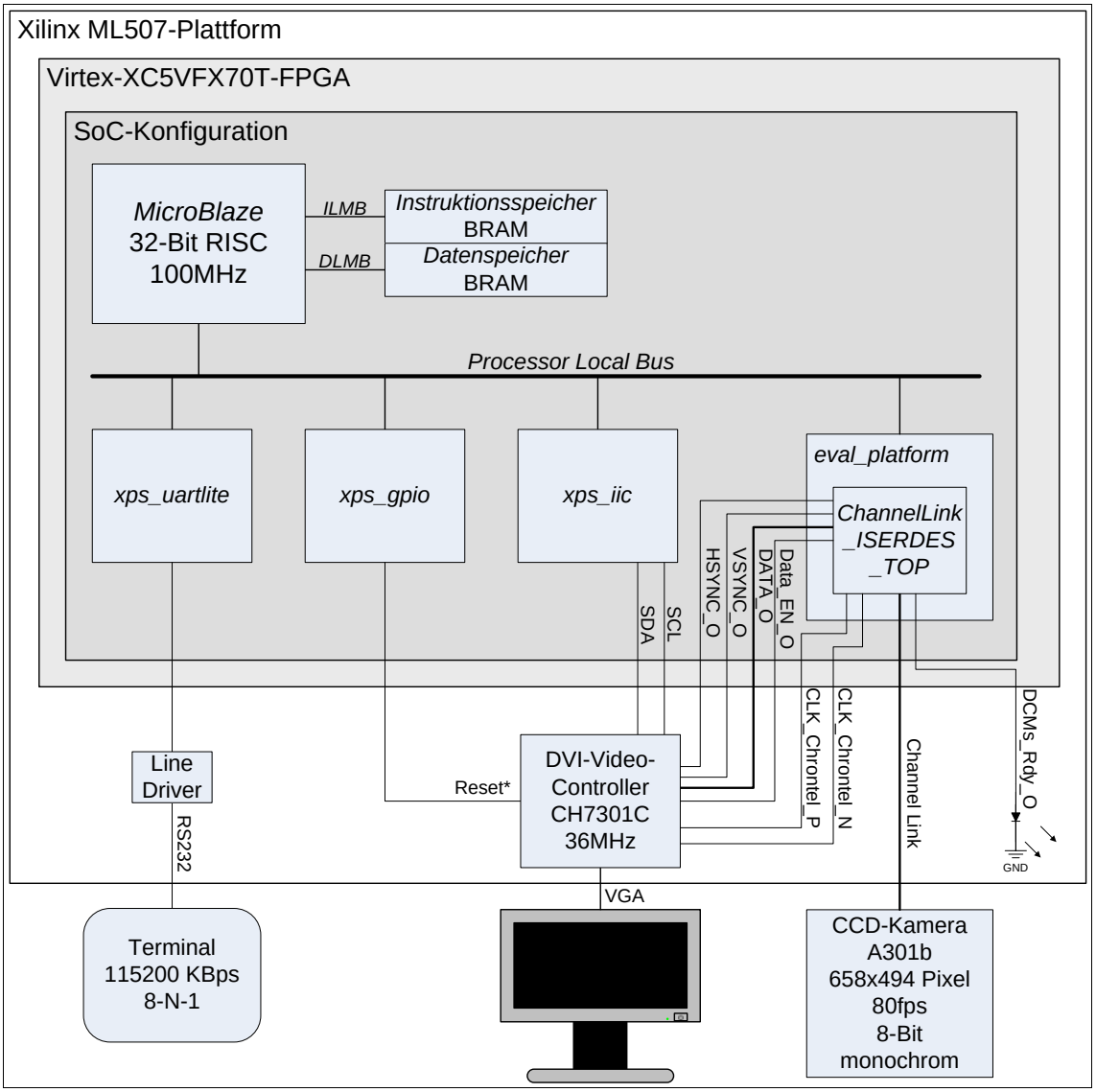

Abbildung 54: Blockschaltbild der SoC-Konfiguration mit Microblaze Soft-Mikroprozessorkern

8.1 Aufbau der SoC-Konfiguration

Der MicroBlaze Soft-Mikroprozessorkern der SoC-Konfiguration ist nach der Harvard-Architektur aufgebaut und verfügt daher über einen getrennten Instruktions- (*ILMB*) und Datenbus (*DLMB*). Folgende, über den Processor Local Bus angebundene Peripheriemodule wurden der SoC-Konfiguration hinzugefügt:

- *xps_uartlite*
  Serielle Schnittstelle nach RS232-Standard. Die Pegelumsetzung (TTL/CMOS ↔ RS232) übernimmt der *Line Driver*. Diese Komponente wird für Textausgaben auf dem *Terminal* verwendet.

- *xps_gpio*
  Standard-Ein-/Ausgangsmodul. Diese Komponente liefert die Active-Low-Reset-Leitung zum Videocontroller CH7301C (vgl. 2.2).

- *xps_iic* [40]
  I²C-Bus-Interfacemodul. Diese Komponente stellt die Verbindung mit dem Videocontroller CH7301C zur Parametrierung dessen über den I²C-Bus (*SCL*, *SDA*) her (vgl. Kapitel 8.2).

- *eval_platform*
  Anwenderspezifisches Softcore-Peripheriemodul mit der intern instanziierten Komponente *ChannelLink_ISERDES_TOP* (vgl. Abbildung 22).

Das Peripheriemodul *eval_platform* wurde auf Basis einer mit dem **C**reate or **I**mport **P**eripheral Wizard (*CIP*) der Entwicklungsumgebung Xilinx Platform Studio v10.1.03 erstellten Peripheriemodul-Vorlage realisiert (vgl. Abbildung 55). Es weist folgende Kenndaten auf:

- Anbindung an den MicroBlaze Soft-Mikroprozessorkern über den Processor Local Bus v4.6 als Slave-Modul
- 1x 32-Bit-breites Softwareregister (*slv_reg0*)
- Software-Reset-Modul zum Zurücksetzen des Peripheriemoduls in Software

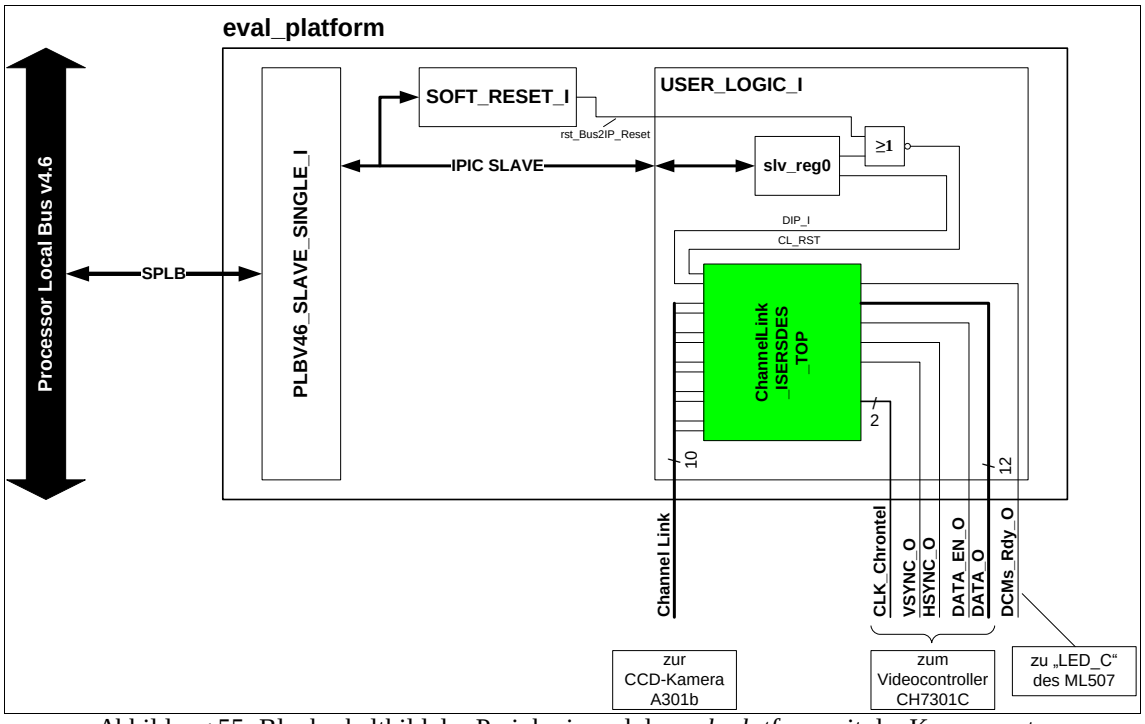

Abbildung 55: Blockschaltbild des Peripheriemoduls *eval_platform* mit der Komponente *ChannelLink_ISERDES_TOP* (grün markiert)

Die Signale des Channel Link Interface werden direkt zu den entsprechenden Eingangpins des FPGAs herausgeführt, an die die CCD-Kamera A301b angeschlossen ist (vgl. Kapitel 2.3). Die Ausgangssignale *CLK_Chrontel*, *VSYNC_O*, *HSYNC_O*, *DATA_EN_O* und *DATA_O* werden zum Videocontroller CH7301C geführt (vgl. Kapitel 2.2). *DCMs_Rdy_O* treibt die LED „LED_C“ des ML507 Evaluierungsboards, die aufleuchtet, sobald die DCM-Konfiguration verlässliche Ausgangssignale liefert (vgl. Abbildung 2 → (5) USER LEDs). Mit dem Modul *SOFT_RESET_I* lässt sich die Komponente *USER_LOGIC_I* von der Softwareschicht aus zurücksetzen. Das LSB des 32-Bit-breiten Softwareregisters (*slv_reg0(0)*, little endian) und der Reset-Ausgang (*rst_Bus2IP_Reset*) der Komponente *SOFT_RESET_I* dienen, über ein NOR-Gatter geführt, als Active-Low-Reset-Signal für die Komponente *ChannelLink_ISERDES_TOP*. Das MSB von *slv_reg0* (*slv_reg0(31)*) wird an den Eingang *DIP_I* von *ChannelLink_ISERDES_TOP* geführt. Dadurch lässt sich von der Softwareschicht aus zwischen Kamerabildern und denen des Testbildgenerators als Datenquelle des Videocontrollers CH7301C umschalten (vgl. Kapitel 7).

8.2 Parametrierung des Videocontrollers CH7301C

Das Evaluierungsboard ML507 ist neben dem XC5VFX70T-FPGA mit einem Videocontroller CH7301C bestückt. Dazu verfügt das ML507 über einen dedizierten Video-I²C-Bus, an den (u.a.) der Videocontroller CH7301C angeschlossen ist (vgl. Abbildung 56, [24]). Daten- und Taktleitung des Video-I²C-Bus werden, der I²C-Spezifikation [16] entsprechend, über Pull-UP-Widerstände auf High-Pegel gehalten. Bidirektionale Level-Shift-FETs gleichen die unterschiedlichen Spannungsniveaus von FPGA und Videocontroller CH7301C aus.

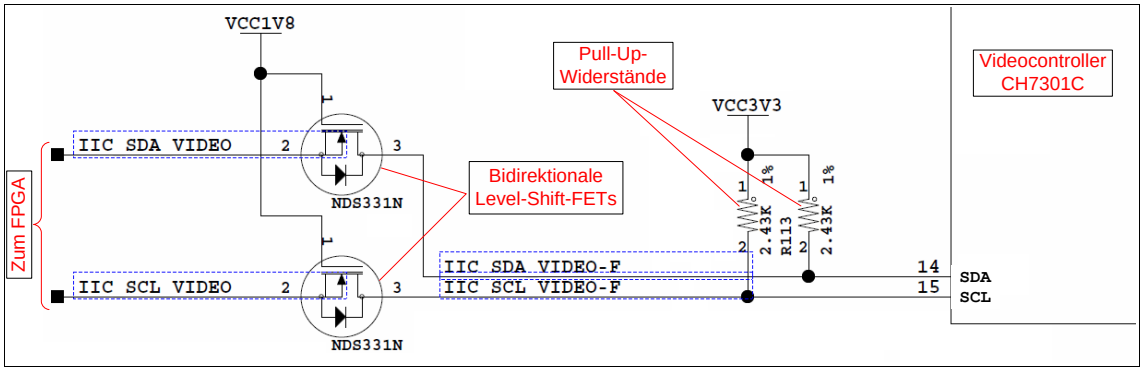

Abbildung 56: Video-I²C-Bus zwischen Videocontroller CH7301C und Virtex-5 FPGA XCVFX70T auf dem ML507

Die Parameter des Videocontrollers CH7301C, wie

- *Clock Mode* zur Konfiguration der Dateneintaktungsweise (SDR/DDR, steigende/fallende Flanke)
- *Input Data Format* zur Wahl des Dateneingangsformates (vgl. Kapitel 2.2)
- *Power Management* zum Aktivieren des analogen VGA-Ausgangs usw.

sind in Softwareregistern des Videocontrollers abgespeichert und lassen über den I²C-Bus konfigurieren. Das dafür erforderliche I²C-Bus-Interface wird der SoC-Konfiguration von dem Peripheriemodul *xps_iic* bereitgestellt [40].

Die Übertragungsgeschwindigkeit und der Adressiermodus des Peripheriemoduls *xps_iic* werden vor Synthese und Implementierung der SoC-Konfiguration mit Generics vorgenommen. Mit den zugehörigen Softwaretreibern lassen sich dynamisch weitere Konfigurationen vornehmen und Datentransfers initiieren. Folgende Konfigurationen wurden in dieser Arbeit vorgenommen:

statisch
- 100kHz Übertragungsgeschwindigkeit
- 7-Bit Adressiermodus

dynamisch
- Bus-Master-Betrieb
- Slave-Adresse (CH7301C → 0x76)
- *Stop Condition*-Generierung für den Schreibzugriff
- *Restart Condition*-Generierung für den Lesezugriff

Für das Auslesen eines 8-Bit-breiten-Parameterregisters des Videocontrollers CH7301C ist gemäß Datenblatt [7] folgender Ablauf einzuhalten (vgl. Abbildung 57):

1. Start (*Start Condition*)
   SDA geht auf Low, während SCL noch auf High-Pegel ist.
2. Adresse schreiben
   Der Master überträgt die 7-Bit-breite Slave-Adresse (*Device ID*) und signalisiert mit *R/W**='0' einen Schreibzugriff. Der Slave bestätigt den Empfang mit einem Low-Pegel (*ACK*) auf der Datenleitung. Der Master überträgt nun die Register-Adresse, dessen Inhalt ausgelesen werden soll (*RAB*), woraufhin der Slave dies wieder bestätigt (ACK).
3. Restart Condition
   SDA geht auf High, während *SCL* noch auf Low ist. *SDA* geht erst wieder auf Low, wenn *SCL* High-Pegel hat.
4. Daten auslesen
   Der Master überträgt die 7-Bit-breite Slave-Adresse (*Device ID*) und signalisiert mit *R/W**→'1' einen Lesezugriff. Der Slave bestätigt den Empfang mit einem Low-Pegel (*ACK*) und überträgt im Anschluss den Inhalt (*Data*) des zuvor adressierten Registers (*RAB*). Der Master signalisiert dem Slave das Ende des Übertragungszyklus mit einem High-Pegel (*ACK*).
5. Stopp (*Stop Condition*)
   Die Übertragung wird beendet, Master und Slave kehren wieder in den Urzustand zurück.

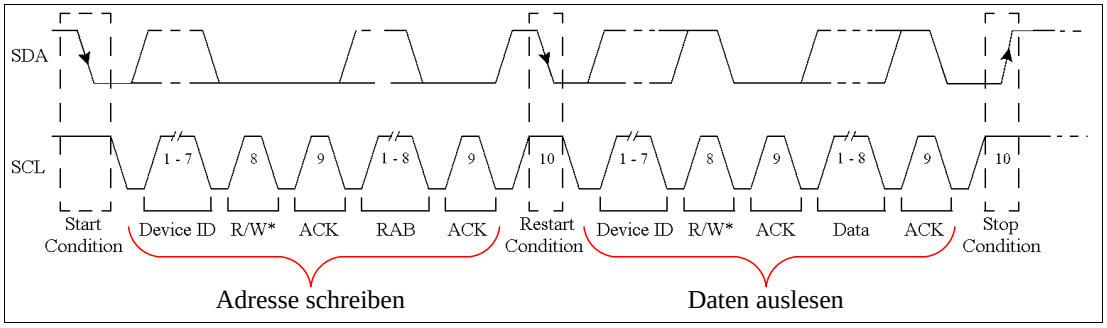

Abbildung 57: Llesezugriff eines Parameterregisters des Videocontrollers CH7301C im I²C-Standard

Das Beschreiben eines einzelnen Registers ist gemäß Datenblatt [7] in folgende Schritte untergliedert (vgl. Abbildung 58):

1. Start (*Start Condition*)
   SDA geht auf Low, während SCL noch auf High-Pegel ist.
2. Adresse schreiben
   Der Master überträgt die 7-Bit-breite Slave-Adresse (*Device ID*) und signalisiert mit *R/W**→'0' einen Schreibzugriff. Der Slave bestätigt den Empfang mit einem Low-Pegel (*ACK*). Der Master überträgt nun die Register-Adresse, dessen Inhalt ausgelesen werden soll (*RAB*), woraufhin der Slave dies wieder bestätigt (ACK).
3. Daten schreiben
   Der Master überträgt die 8-Bit-breiten Daten (*Data*) und signalisiert dem Slave das Ende des Übertragungszyklus mit einem High-Pegel (*ACK*).
4. Stopp (*Stop Condition*)
   Die Übertragung wird beendet, Master und Slave kehren wieder in den Urzustand zurück.

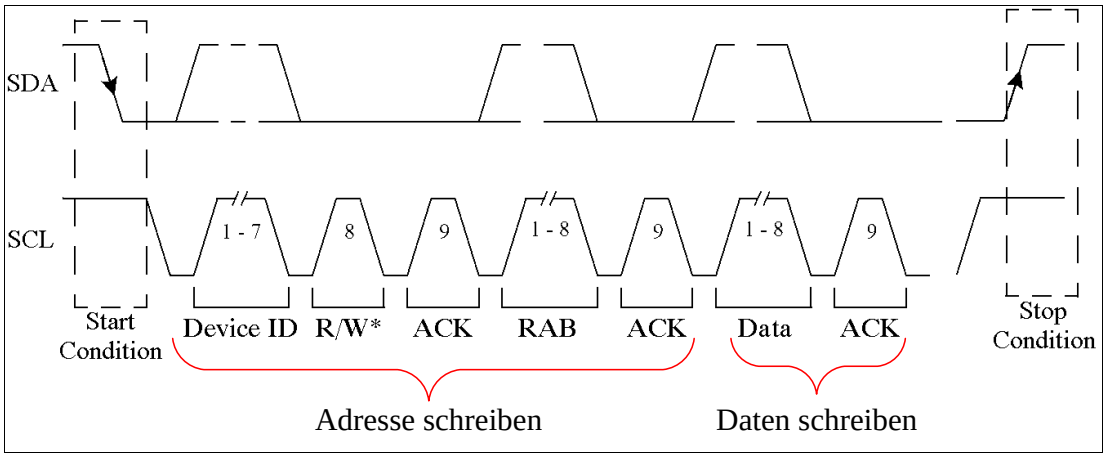

Abbildung 58: Schreibzugriff auf ein Parameterregister des Videocontrollers CH7301C im I²C-Standard

Auf Basis der API-Funktionen `XIic_Send(...)` und `XIic_Recv(...)` der *xps_iic*-Softwaretreiber wurden folgende Funktionen erstellt:

- `XStatus SendIIC(Xuint8 RegAddress, Xuint8 data)`
  Das Datenbyte `data` wird unter Verwendung von `XIic_Send(…)` in das Register mit der Adresse `RegAddress` geschrieben. Bei erfolgreicher Übertragung wird eine 0 (4-Byte), ansonsten eine 1 im LSB zurückgeliefert.

- `XStatus ReadIIC(Xuint8 RegAddress, Xuint8 *ReadBuffer)`
  Der Inhalt des Registers mit der Adresse `RegAddress` wird in den dereferenzierten Speicherbereich von `ReadBuffer` geschrieben. Dazu wird unter Verwendung von `XIic_Send(…)` das entsprechende Register adressiert und anschließend mit `XIic_Recv(…)` der Registerinhalt ausgelesen. Bei erfolgreicher Übertragung wird eine 0 (4-Byte), bei Fehlern eine 1 im LSB zurückgeliefert.

- `ReadIICReg(Xuint8 RegAddress)`
  Der Name und Inhalt des Registers mit der Adresse `RegAddress` werden unter Verwendung von `ReadIIC(…)` temporär abgespeichert und anschließend über *stdout* ausgegeben.

8.3 Verifikation des Schreib- und Lesezugriffs auf die Parameterregister des Videocontrollers CH7301C

In diesem Kapitel werden Schreib- und Lesezugriff auf die Parameterregister des Videocontrollers CH7301C verifiziert. Dazu werden die Video-I²C-Bussignale, *SDA* und *SCL*, über Tastköpfe an den Chipkontakten abgegriffen und auf einem PC-Oszilloskop Pico ADC-200/20 aufgenommen (vgl. Abbildung 59). Das Triggersignal ist die fallende Flanke auf der Datenleitung *SDA* (Kanal A).

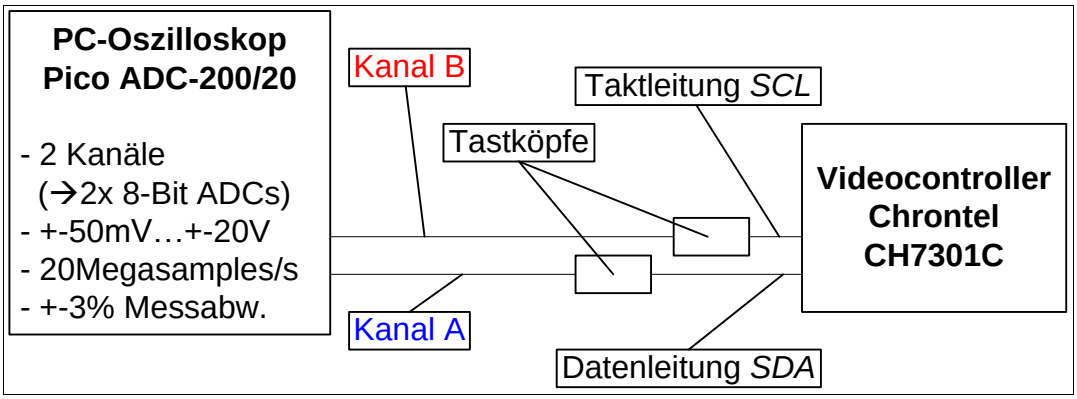

Abbildung 59: Messaufbau zur Messung der Video-I²C-Bussignale

Schreibzugriff

In das Parameterregister *Input Clock* (*IC*) mit der Adresse `0x1D` (und Standardwert `0x48`) soll mit dem Funktionsaufruf `SendIIC(0x1D,0x4F)` der Wert `0x4F` geschrieben werden.

Das in Abbildung 60 dargestellte Oszillogramm kennzeichnet den Bustransferzyklus des Schreibzugriffs.

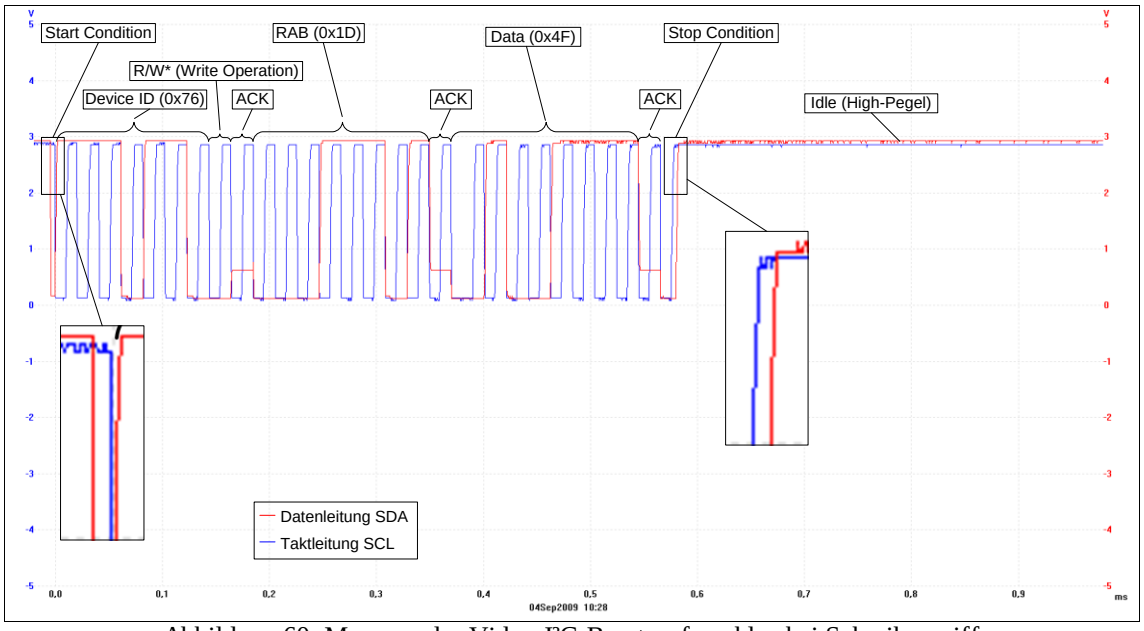

Abbildung 60: Messung des Video-I²C-Busrtansferzyklus bei Schreibzugriff

Lesezugriff

Das zuvor mit `0x4F` beschriebene *IC*-Register soll mit dem Funktionsaufruf `ReadIICReg(0x2B)` ausgelesen werden. Unter Einhaltung der im Datenblatt geforderten *Restart Condition* ist der Lesezugriff nicht realisierbar (vgl. Abbildung 61). Der Videocontroller CH7301C hält die Datenleitung *SDA* nach der zweiten Übertragung der *Device ID* auf Low-Pegel. Dadurch belegt der Videocontroller dauerhaft den I²C-Bus und ist auch nicht mehr ansprechbar
→ ein Reset des Videocontrollers ist zwingend erforderlich.

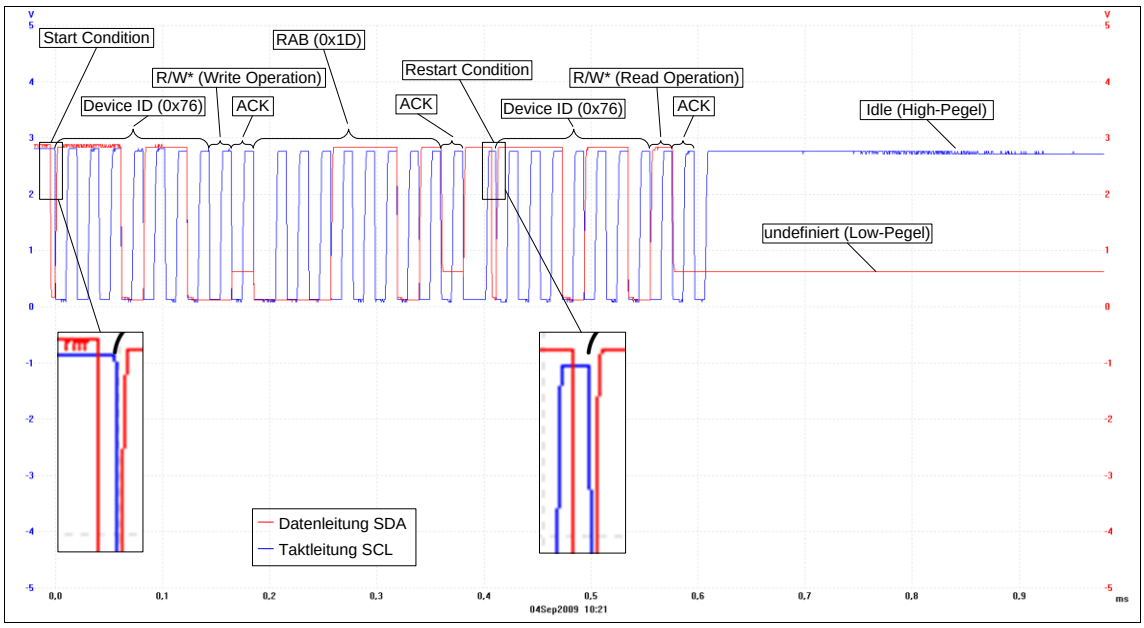

Abbildung 61: Messung des Video-I²C-Bustansferzyklus bei gescheitertem Lesezugriff

Die empirische Erprobung von *Stop Condition*- und *Restart Condition*-Kombinationen brachte hervor, dass der Lesezugriff unter Verwendung der *Stop Condition* anstelle der *Restart Condition* durchführbar ist (vgl. Kapitel 8.2 → `ReadIIC(…)`). Das in Abbildung 62 dargestellte Oszillogramm kennzeichnet den Bustransferzyklus des Lesezugriffs.

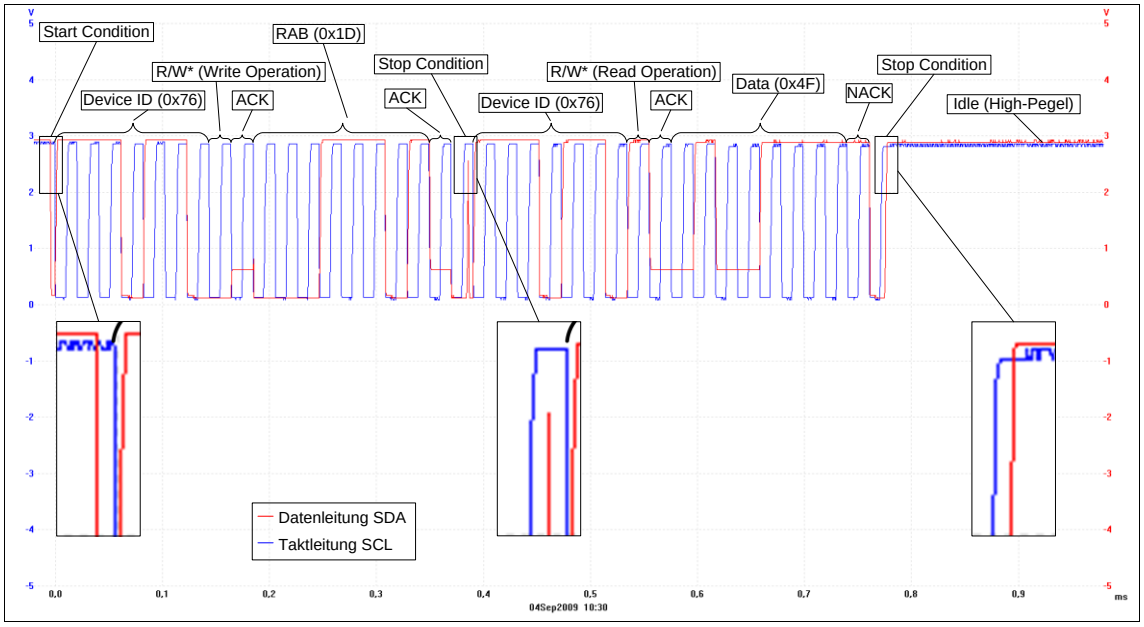

Abbildung 62: Messung des Video-I²C-Bustansferzyklus bei Lesezugriff

9 Zusammenfassung

Im Rahmen dieser Arbeit wurde eine SoC-Erprobungsplattform für CCD-Kameras mit Channel Link Interface realisiert.

Die der Erprobungsplattform zugrundeliegenden Hardwarekomponenten wurden vorgestellt und 2 Kameras mit Channel Link Interface in Bezug auf ihre Konfigurationsfähigkeit mit dem Evaluierungsboard ML507 miteinander verglichen.

Die für die Erprobungsplattform eingesetzte FPGA-Technolgie wurde untersucht und zusätzlich ein Verfahren zur Einhaltung gleichbleibender Laufzeitpfade aufgezeigt.

Der vorliegende Channel Link Receiver für die Deserialisierung des seriellen Channel Link-Datenstroms aus [13] wurde an die Hardwarekomponenten der Erprobungsplattform angepasst und letztendlich durch dedizierte Serien-Parallel-Umsetzer der Virtex-5 FPGA-Reihe ersetzt und erfolgreich implementiert.

Ein parametrierbarer VGA-Controller wurde realisiert, mit dem sich beliebige Bildauflösungen und kontinuierliche Datenströme auf einem VGA-Monitor darstellen lassen.

Es wurde ein parametrierbarer Testbildgenerator entworfen, mit dem sich beliebige Bildauflösungen und Synchronisationssignal-Timings erzeugen lassen. Mit dieser Komponente lassen sich unterschiedliche Kameradatenströme simulieren.

Die einzelnen Komponenten wurden zu einem Gesamtsystem zusammengesetzt und zur Verifikation des zeitlichen Verhaltens einer Timing-Simulation unterzogen. Anschließend wurde das Gesamtsystem auf demVirtex-5 XC5VFX70T FPGA des Evaluierungsboards ML507 implementiert. Die Funktionalität wurde durch Darstellung der wahlweise Kamera- bzw. Testbildgeneratorbilder auf einem VGA-Monitor verifiziert.

Das Gesamtsystem wurde in ein Peripheriemodul des MicroBlaze Soft-Mikroprozessorkerns ausgelagert. Die Umschaltung zwischen Testbildgenerator- und Kamerabildern wird in Software vorgenommen. Der Videocontroller CH7301C wird ebenfalls in Software für den Betrieb der Erprobungsplattform parametriert. Die Funktionalität des MicroBlaze-SoC konnte ebenfalls durch Darstellung der Kamera- bzw. Testbildgeneratorbilder auf einem VGA-Monitor verifiziert werden.

Abbildungsverzeichnis

Literaturverzeichnis

[1] Nischwitz: Computergrafik und Bildverarbeitung : Alles für Studium und Praxis, 2., verbesserte und erweiterte Auflage, 2007

[2] A. Erhard: Einführung in die Digitale Bildverarbeitung : Grundlagen, Systeme und Anwendungen, Online-Ausgabe, 2008, http://www.springerlink.com/content/r7g541/

[3] Basler A300b User's Manual, DA041002, 13. Juli 2001, http://www.baslerweb.com/downloads/23820/A300b_Users_Manual_1_.pdf

[4] Binder Subminiatur Rundsteckverbinder, Serie 712, Technische Daten, 21.08.2009, http://www.binder-connector.de/pdfs/produkte/TD_712.PDF

[5] C. Barnden: Driver Assistance Systems Pose FPGA Opportunities. Xilinx Xcell Journal No.66 4th quarter 08

[6] Camera Link, Specifications of the Camera Link Interface Standard for Digital Cameras and Frame Grabbers, Version 1.1, http://www.photonfocus.com/upload/specifications/CameraLink_v1_1.pdf

[7] Chrontel Application Notes AN-41: CH7009A/B, CH7010A/B, CH7012A/B, CH7301A/B Encoder Registers Read/Write Operation, (Rev. 1.3), http://www.chrontel.com/pdf/an41.pdf

[8] Chrontel CH7301C DVI Transmitter Device (Rev. 1.32), http://www.chrontel.com/pdf/7301ds.pdf

[9] Constraints Guide, 10.1, http://www.xilinx.com/itp/xilinx10/books/docs/cgd/cgd.pdf

[10] D. Bagni; R. Marzotto; P. Zoratti: Building Automotive Driver Assistance System Algorithms with Xilinx FPGA Platforms. Xilinx Xcell Journal No.66 4th quarter 08

[11] Embedded Processor Block in Virtex-5 FPGAs: Reference Guide, UG200 (v1.6), 20. Januar 2009, http://www.xilinx.com/support/documentation/user_guides/ug200.pdf

[12] F. Kesel und R. Bartholomä: Entwurf von digitalen Schaltungen und Systemen mit HDLs und FPGAs, Oldenbourg, 2006

[13] F. Paulo, Bildvorverarbeitungsmodule für eine FPGA-basierte CCD-Kamera mit Optimierung einer 700 MBit Schnittstelle, Bachelorarbeit HAW Hamburg, Department Informatik 2007

[14] FPGA Editor Help: FPGA Editor Overview, 23.07.2009, http://www.xilinx.com/itp/xilinx10/help/iseguide/mergedProjects/fpga_editor/fpga_editor.htm

[15] G. Bärwolff: Numerik für Ingenieure, Physiker und Informatiker, 2007

[16] I2C-bus specification and user manual, UM10204, Rev. 03, NXP, 19. Juni 2007, http://www.nxp.com/acrobat_download/usermanuals/UM10204_3.pdf

[17] ISE WebPACK Design Software, 23.07.2009, http://www.xilinx.com/tools/webpack.htm

[18] J. Reichardt, B. Schwarz, VHDL-Synthese: Entwurf digitaler Schaltungen und Systeme, 4. Auflage, Oldenbourg Verlag, Juli 2007

[19] J. Reichardt, Lehrbuch Digitaltechnik: eine Einführung mit VHDL, Oldenbourg Verlag, März 2009

[20] K. Jack, Video Demystified: A Handbook for the Digital Engineer, 4th Edition, 2004

[21] Microblaze 16x2 LCD Driver, 10. Juli 2009, http://www.fpgadeveloper.com/2008/10/microblaze-16x2-lcd-driver.html

[22] MicroBlaze Soft Processor Core, 24.07.2009, http://www.xilinx.com/tools/microblaze.htm

[23] ML505/6/7 Block Diagram
SCHEM, ROHS COMPLIANT
ML505/6/7 VIRTEX-5 EVALUATION PLATFORM, 1280415
0381241, Version A, Revision 02,
http://www.xilinx.com/support/documentation/boards_and_kits/ml50x_schematics.pdf

[24] ML505/ML506/ML507 Evaluation Platform Schematics (v1.0.2), http://www.xilinx.com/support/documentation/boards_and_kits/ml50x_schematics.pdf

[25] ML505/ML506/ML507 Evaluation Platform User Guide UG347 (v3.1), http://www.xilinx.com/support/documentation/boards_and_kits/ug347.pdf

[26] National Semiconductor, Channel Link Design Guide, June 2006, http://www.national.com/appinfo/lvds/files/channellink_design_guide.pdf

[27] National Semiconductor, LVDS Owner's Manual, Including High-Speed CML and Signal Conditioning, Fourth Edition, 2008, http://www.national.com/appinfo/lvds/files/National_LVDS_Owners_Manual_4th_Edition_2008.pdf

[28] P. Deuflhard, Andreas Hohmann: Numerische Mathematik: Eine algorithmisch orientierte Einführung, 1991

[29] Specification for LCD Module TM162ABCWVBYA, 13. Juli 2009, http://tianma-europe.com/downloads/tm162abcwvbya.pdf

[30] TinyVGA.com: VGA Microcontroller projects, 01.09.2009 http://www.tinyvga.com/vga-timing

[31] User Manual MV-D750E Series, CMOS Area Scan Cameras, MAN020 03/2008 v2.1, http://www.photonfocus.com/upload/manuals/MAN020_e_V2_1_MVD750E.series.pdf

[32] Virtex-4 FPGA User Guide, UG070 (v2.6), 1. Dezember 2008, http://www.xilinx.com/support/documentation/user_guides/ug070.pdf

[33] Virtex-5 Family Product Table, 10. Juli 2009, http://www.xilinx.com/publications/prod_mktg/V5_LX__TXT_psm_table.pdf

[34] Virtex-5 FPGA Data Sheet: DC and Switching Characteristics, DS202 (v5.0), http://www.xilinx.com/support/documentation/data_sheets/ds202.pdf

[35] Virtex-5 FPGA User Guide, UG190 (v5.0), http://www.xilinx.com/support/documentation/user_guides/ug190.pdf

[36] Virtex-5 FXT FPGA ML507 Evaluation Platform, 10. Juli 2009, http://www.xilinx.com/products/devkits/HW-V5-ML507-UNI-G.htm

[37] Virtex-5 Libraries Guide for HDL Designs, ISE 10.1 http://www.xilinx.com/itp/xilinx10/books/docs/virtex5_hdl/virtex5_hdl.pdf

[38] Virtex-5 Multi-Platform FPGA, 10. Juli 2009, http://www.xilinx.com/products/virtex5/index.htm

[39] Xilinx Answer Record AR #30355 – PAR 10.1 http://www.xilinx.com/support/answers/30355.htm

[40] XPS IIX Bus Interface (v2.00a), Product Specification, DS606, 24. April 2009 http://www.xilinx.com/support/documentation/ip_documentation/xps_iic.pdf

Anhang

Sämtliche im Rahmen dieser Arbeit erstellten Quellcodes befinden sich auf CD-ROM und können bei Herrn Prof. Dr. Jürgen Reichardt und Herr Prof. Dr.-Ing. Bernd Schwarz eingesehen werden.

A1 Channel Link-Protokoll, Camera Link Interface, LVDS-Standard

LVDS-Standard

Low **V**oltage **D**ifferential **S**ignaling (LVDS) ist ein Schnittstellenstandard, mit dem sich Hochgeschwindigkeitsdatenübertragungen realisieren lassen (vgl. [27]). Senderseitig treibt eine schaltbare Stromquelle einen konstanten Strom von 3,5mA. Dieser fließt je nach zu übertragendem Logikpegel über die positive/negative Leitung (*positive/negative side*) durch einen Abschlusswiderstand und schließlich über die negative/positive Leitung wieder zurück zum Treiber (*Driver*). Der mit 100Ω bemessene Abschlusswiderstand entspricht der Leitungsimpedanz und verhindert dadurch Reflexionen auf der Leitung. An dem Abschlusswiderstand bewirkt der durchfließende Strom einen Spannungsabfall von +-3,5mA*100Ω = +-350mV.

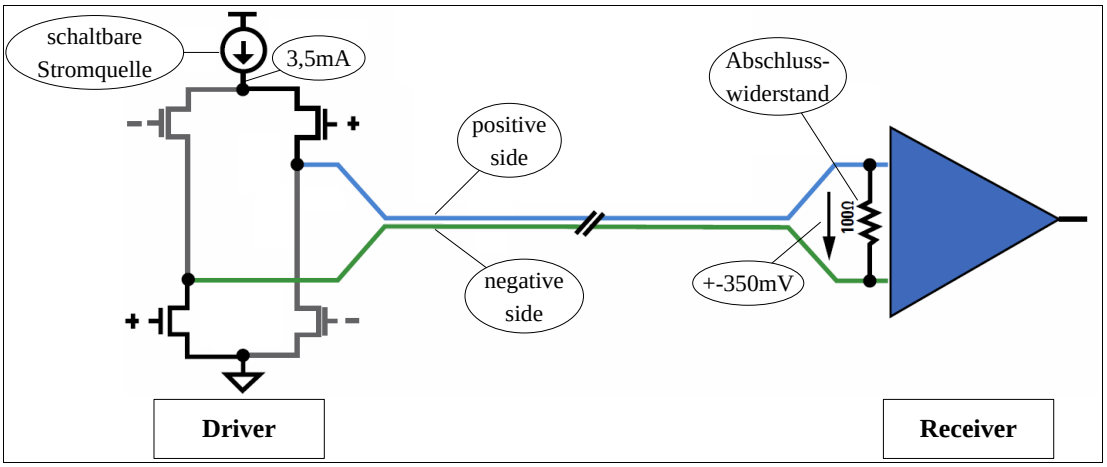

Abbildung 63: LVDS-Hardwareaufbau

Aus der Spannungsdifferenz beider Leiter geht der zu übertragene Logikpegel hervor. Beide Leiter liegen verdrillt beisammen. Störungen wirken sich auf beide Leiter gleichermaßen aus, die Spannungsdifferenz bleibt dennoch gleich.

Channel Link-Protokoll

Das Channel Link-Protokoll (vgl. Abbildung 64) spezifiziert die parallele Signalübertragung auf einem Takt- (*Clock*) und 4 Datenkanälen (*DATA*). Die Übertragung findet unidirektional von *Transmitter* zu *Receiver* im LVDS-Standard statt. Innerhalb eines Taktes (auf

der Taktleitung) werden je Datenkanal 7-Datenbit seriell, insgesamt also 28-Datenbit übertragen.

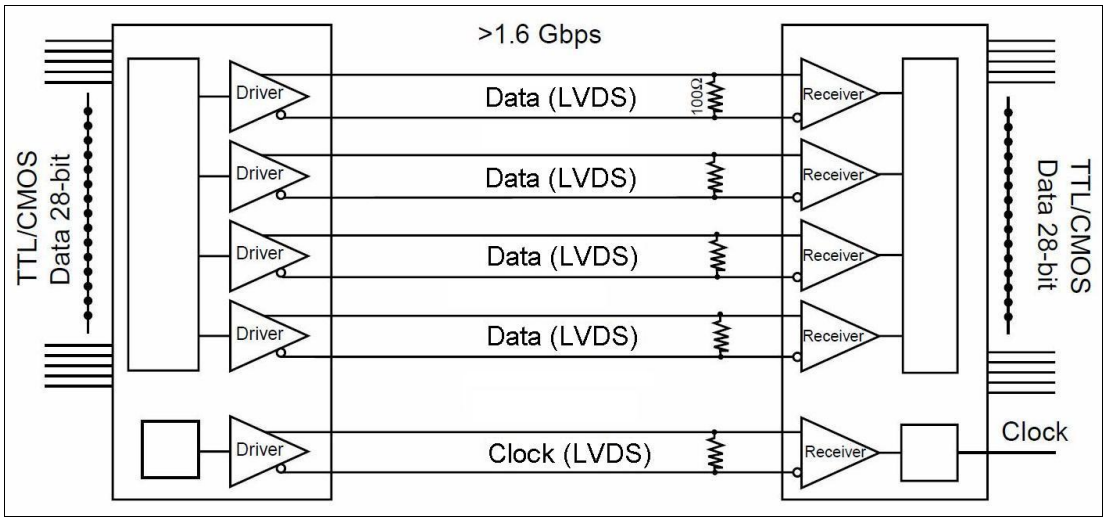

Abbildung 64: Channel Link Interface

Diese Übertragungsform ermöglicht hohe Datenraten bei geringem Systemtakt und geringer Anzahl an Leitungen.

Camera Link Interface

Das Camera Link Interface beruht auf dem *Channel Link-Protokoll*. Zusätzlich werden 4 **C**amera **C**ontrol-Signale (*CC1…4*) zur Kamerasteuerung und eine serielle Schnittstelle für die bidirektionale Kommunikation mit der Kamera im LVDS-Standard bereitgestellt (vgl. Abbildung 65). Das Camera Link Interface wird in 3 Konfigurationen unterteilt:

- *Base*
  - 1 Taktleitung
  - 4 Datenleitungen
  - Bidirektionale serielle Kommunikationsleitungen (*SerTC, SerTFG*)
  - 4 Kamera-Kontrollsignale (*CC1..4*)
  - 1 MDR-Konnektor
- *Medium*
  - 2 Taktleitungen
  - 8 Datenleitungen
  - Bidirektionale serielle Kommunikationsleitungen (*SerTC, SerTFG*)
  - 4 Kamera-Kontrollsignale (*CC1..4*)
  - 2 MDR-Konnektoren
- *Full*
  - 3 Taktleitungen
  - 12 Datenleitungen
  - Bidirektionale serielle Kommunikationsleitungen (*SerTC, SerTFG*)

- 4 Kamera-Kontrollsignale (*CC1..4*)
- 2 MDR-Konnektoren

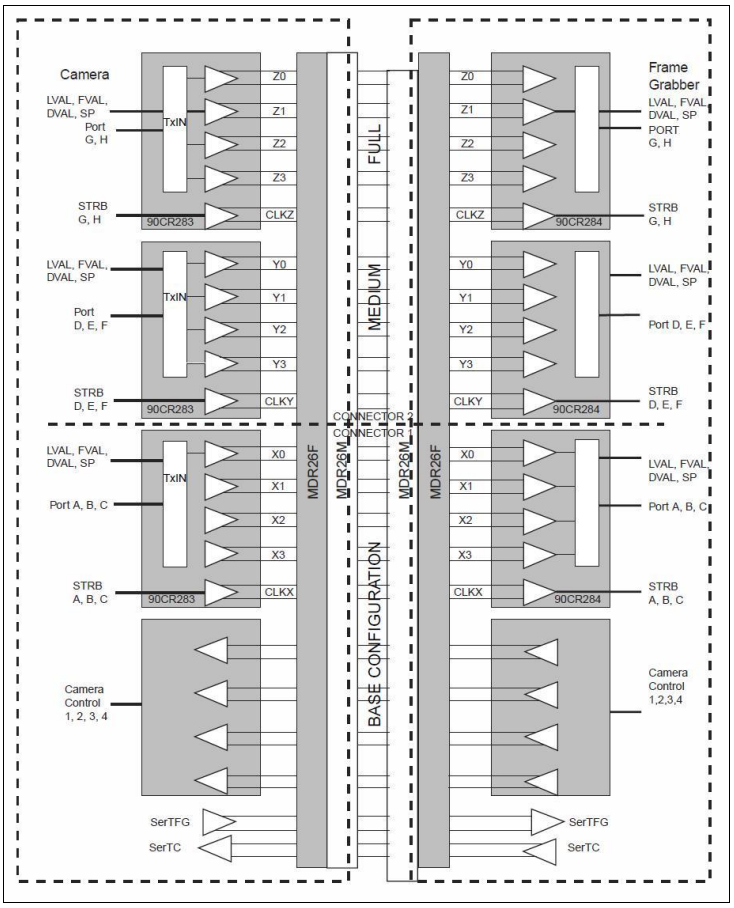

Abbildung 65: Blockschaltbild: Camera Link Interface
in Base, Medium und Full Configuration

A2 *DCM_RESET_LOGIC* zum synchronen Rücksetzen von *DCM*-Konfigurationen

Für den korrekten Betrieb der *DCM*-Blöcke der Virtex-5 FPGA-Reihe wird dringend empfohlen, das Eingangs-Reset-Signal für mindestens 3 Taktperioden des Eingangstaktsignals zu halten [35]. Die Komponente *DCM_RESET_LOGIC* erfüllt diese Anforderung (vgl. Abbildung 66). Das Eingangs-Reset-Signal (*RST_I*) wird an den Dateneingang des ersten von insgesamt 4 hintereinander geschalteten Datenflipflops (*FD*) gelegt. Die Ausgangssignale aller Datenflipflops werden ODER-verknüpft herausgeführt (*RST_O*), sodass das Eingangs-Reset-Signal um eine Taktperiode verzögert mindestens 4 Taktperioden lang am Ausgang gehalten wird. Der VHDL-Code der Komponente ist in Listing 1 dargestellt.

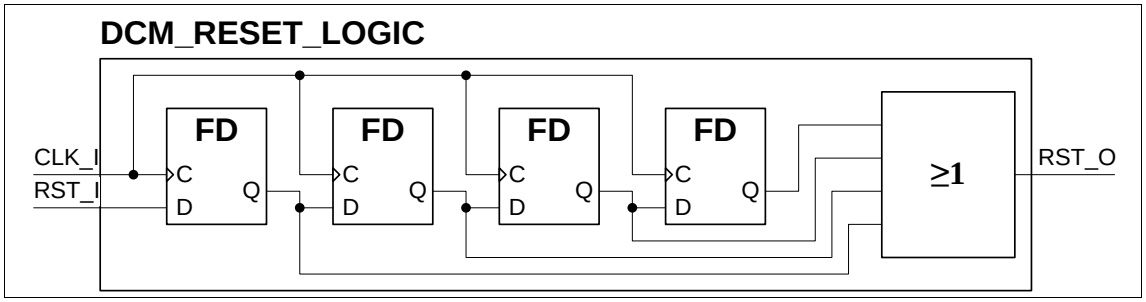

Abbildung 66: Blockschaltbild des modifizierten *DCM_Reset_Logic*

```
library IEEE;
use IEEE.std_logic_1164.ALL;
Library UNISIM;
use UNISIM.vcomponents.all;

entity DCM_RESET_LOGIC is
  port(CLK_I : in  std_logic;
       RST_I : in  std_logic;
       RST_O : out std_logic);
end DCM_RESET_LOGIC;

architecture Behavioral of DCM_RESET_LOGIC is

signal FD0_Q_OUT : std_logic;
signal FD1_Q_OUT : std_logic;
signal FD2_Q_OUT : std_logic;
signal FD3_Q_OUT : std_logic;

begin

  FD0_INST : FD
    port map (C=>CLK_I, D=>RST_I, Q=>FD0_Q_OUT);

  FD1_INST : FD
    port map (C=>CLK_I, D=>FD0_Q_OUT, Q=>FD1_Q_OUT);

  FD2_INST : FD
    port map (C=>CLK_I, D=>FD1_Q_OUT, Q=>FD2_Q_OUT);

  FD3_INST : FD
    port map (C=>CLK_I, D=>FD2_Q_OUT, Q=>FD3_Q_OUT);

  RST_O <= FD0_Q_OUT or FD1_Q_OUT or FD2_Q_OUT or FD3_Q_OUT;

end Behavioral;
```

Listing 1: VHDL-Code der Komponente *DCM_RESET_LOGIC*

Mit der Anwendung Xilinx CORE Generator v10.1.03 lässt sich optional zur *DCM*-Konfiguration eine *DCM*-Reset-Schaltung als HDL-Code generieren (vgl. Listing 1). Der Aufbau der Komponente bewirkt, dass das Reset-Signal direkt durchgeschaltet wird (vgl. Abbildung 67 → *OR2*). Daten- (*D*) und SET-Eingang (*S*) des ersten Datenflipflops werden dauerhaft auf Low-Pegel gesetzt, sodass durchgehend Low-Pegel am Ausgang von *OR3* liegt und das Ausgangs-Reset-Signal (*RST_IN*) somit nicht die minimal geforderten 3 Taktperioden gehalten wird.

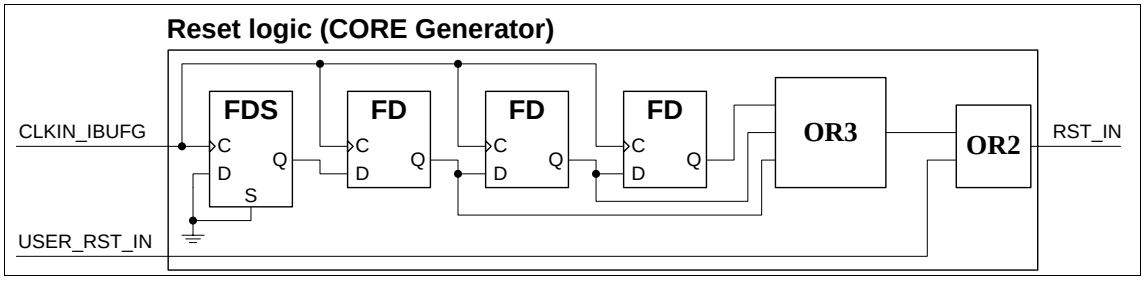

Abbildung 67: Blockschaltbild der *Reset logic* vom Xilinx CORE Generator

A3 VHDL-Code *TestBench_CL_Receiver_v2*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity TestBench_CL_Receiver_v2 is
end TestBench_CL_Receiver_v2;

ARCHITECTURE Behavior OF TestBench_CL_Receiver_v2 is

  -- Deklaration der zu simulierenden Komponente
  component ChannelLink_Receiver_v2
    port(CLK_1X       : in  std_logic;
         CLK_7X       : in  std_logic;
         RST_I        : in  std_logic;
         RX0_I        : in  std_logic;
         RX1_I        : in  std_logic;
         RX2_I        : in  std_logic;
         RX3_I        : in  std_logic;
         RX0_O        : out std_logic_vector(6 downto 0);
         RX1_O        : out std_logic_vector(6 downto 0);
         RX2_O        : out std_logic_vector(6 downto 0);
         RX3_O        : out std_logic_vector(6 downto 0);
         ACTIVE_LED_O : out std_logic);
  end component;


  -- Eingänge
  signal CLK_1X : std_logic := '0';
  signal CLK_7X : std_logic := '0';
  signal RST_I  : std_logic := '1';
  signal RX0_I  : std_logic := '0';
  signal RX1_I  : std_logic := '0';
  signal RX2_I  : std_logic := '0';
  signal RX3_I  : std_logic := '0';
   -- Ausgänge
  signal RX0_O        : std_logic_vector(6 downto 0);
  signal RX1_O        : std_logic_vector(6 downto 0);
  signal RX2_O        : std_logic_vector(6 downto 0);
  signal RX3_O        : std_logic_vector(6 downto 0);
  signal ACTIVE_LED_O : std_logic;

  -- Pixeltakt (als Referenz)
  signal RxCLK : std_logic;

  -- Datenstrom der 4 CL-Datenkanäle
  signal RX0_BIT_SEQUENCE : std_logic_vector(6 downto 0) := (others => '0');
  signal RX1_BIT_SEQUENCE : std_logic_vector(6 downto 0) := (others => '0');
  signal RX2_BIT_SEQUENCE : std_logic_vector(6 downto 0) := (others => '0');
  signal RX3_BIT_SEQUENCE : std_logic_vector(6 downto 0) := (others => '0');

  -- Eingangspixel (die seriell empfangen werden sollen)
  signal Pixel_Even_in : std_logic_vector(7 downto 0) := x"00";
  signal Pixel_Odd_in  : std_logic_vector(7 downto 0) := x"00";

  -- Ausgangspixel (als Referenz)
  signal Pixel_Even_received : std_logic_vector(7 downto 0) := x"00";
  signal Pixel_Odd_received  : std_logic_vector(7 downto 0) := x"00";

  constant CLK_1X_period : time := 70 ns;
  constant CLK_7X_period : time := 10 ns;
  constant RESET_Time    : time := CLK_1X_period * 10;

begin

  -- Stimulation der Pixel
  Pixel_STIM: process
  begin
    Pixel_Even_in <= x"00";
    Pixel_Odd_in  <= x"00";
```

```
    wait for CLK_7X_period*2; -- 2-Bits des vorangegangenen Zyklus überspringen
    wait for CLK_1X_period*5;
    while(true) loop
      -- ab hier gewünschte Pixel eingeben...
      -- Testfall 1
      Pixel_Even_in <= x"AF";
      Pixel_Odd_in  <= x"FE";
      wait for CLK_1X_period;
      -- Testfall 2
      Pixel_Even_in <= x"01";
      Pixel_Odd_in  <= x"23";
      wait for CLK_1X_period;
      -- hier weitere Testfälle eintragen
    end loop;
    wait;
  end process Pixel_STIM;


  -- Instanziierung der zu simulierenden Komponente
  uut: ChannelLink_Receiver_v2
    port map(CLK_1X => CLK_1X,
             CLK_7X => CLK_7X,
             RST_I  => RST_I,
             RX0_I  => RX0_I,
             RX1_I  => RX1_I,
             RX2_I  => RX2_I,
             RX3_I  => RX3_I,
             RX0_O  => RX0_O,
             RX1_O  => RX1_O,
             RX2_O  => RX2_O,
             RX3_O  => RX3_O,
             ACTIVE_LED_O => ACTIVE_LED_O);

  -- Simulation des Pixeltaktes (als Referenz)
  RxCLK_proc: process
  begin
    RxCLK <= '1';
    wait for CLK_1X_period/2;
    RxCLK <= '0';
    wait for CLK_1X_period/2;
  end process RxCLK_proc;

  -- Generierung des Rahmentaktes
  CLK_1X_proc: process
  begin
    CLK_1X <= '0';
    wait for CLK_7X_period*2; -- 2 Bittakte gegenüber Pixeltakt verschoben
    while(true) loop
      CLK_1X <= '1';
      wait for CLK_1X_period/2;
      CLK_1X <= '0';
    wait for CLK_1X_period/2;
    end loop;
    wait;
  end process CLK_1X_proc;

  -- Generierung des Bittaktes
  CLK_7X_proc :process
  begin
    CLK_7X <= '0';
    wait for CLK_7X_period/2;
    CLK_7X <= '1';
    wait for CLK_7X_period/2;
  end process CLK_7X_proc;

  -- System-Reset
  REST_proc: process
  begin
    RST_I <= '1';
    wait for RESET_TIME;
    RST_I <= '0';
    wait;
  end process REST_proc;
```

```
  -- Datenströme mit Bitsequenzen belegen
  RX3_BIT_SEQUENCE <= "101" & Pixel_Even_in(7 downto 6) & Pixel_Odd_in(7 downto 6);
  RX2_BIT_SEQUENCE <= "0110000";
  RX1_BIT_SEQUENCE <= "00" & Pixel_Even_in(5 downto 1);
  RX0_BIT_SEQUENCE <= Pixel_Even_in(0) & Pixel_Odd_in(5 downto 0);

  -- Serielle Datenströme generieren
  RX_STIM: process
  begin
    -- 2-Bit des verangegangenen Zyklus
    for i in 1 downto 0 loop
      RX3_I <= RX3_BIT_SEQUENCE(i);
      RX2_I <= RX2_BIT_SEQUENCE(i);
      RX1_I <= RX1_BIT_SEQUENCE(i);
      RX0_I <= RX0_BIT_SEQUENCE(i);
      wait for CLK_7X_period;
    end loop;
    -- 5-Bit des aktuellen Zyklus
    for i in 6 downto 2 loop
      RX3_I <= RX3_BIT_SEQUENCE(i);
      RX2_I <= RX2_BIT_SEQUENCE(i);
      RX1_I <= RX1_BIT_SEQUENCE(i);
      RX0_I <= RX0_BIT_SEQUENCE(i);
      wait for CLK_7X_period;
    end loop;
  end process RX_STIM;

  -- Empfangene Pixel (descrambled) für die Simulation
  Pixel_Even_received <= RX3_O(3 downto 2) & RX1_O(4 downto 0) & RX0_O(6);
  Pixel_Odd_received  <= RX3_O(1 downto 0) & RX0_O(5 downto 0);
end;
```

Listing 2: VHDL-Code der Testbench *TestBench_CL_Receiver_v2*

A4 VHDL-Code *TestBench_ISERDES*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entitiy TestBench_ISERDES is
end TestBench_ISERDES;

  -- Deklaration der zu simulierenden Komponente
architecture behavior OF TestBench_ISERDES is

  component ISERDES_1_ZU_7
    port(CLK_1X : in  std_logic;
         CLK_7X : in  std_logic;
         RST_I  : in  std_logic;
         D_I    : in  std_logic;
         D_O    : out  std_logic_vector(6 downto 0));
  end component;

  -- Eingänge
  signal CLK_1X : std_logic := '0';
  signal CLK_7X : std_logic := '0';
  signal RST_I  : std_logic := '1';
  signal D_I    : std_logic := '0';

  -- Ausgang
  signal D_O    : std_logic_vector(6 downto 0);

  -- Zu simulierender Datenstrom lässt sich als 8-Bit-breites
  -- Signal in Hexadezimalschreibweise zuweisen
  signal BIT_SEQUENCE : std_logic_vector(7 downto 0) := x"00";

  constant CLK_1X_period : time := 70 ns;
  constant CLK_7X_period : time := 10 ns;
  constant RESET_Time    : time := CLK_1X_period*4 + CLK_7X_period*3;

begin

  -- Stimulierung der Pixel
  Data_STIM: process
  begin
    BIT_SEQUENCE <= x"00";
    wait for CLK_7X_period*2; -- 2-Bits des vorangegangenen Zyklus überspringen
    wait for CLK_1X_period*5;
    while(true) loop
      -- ab hier gewünschte Pixel eingeben
      -- Testfall 1
      BIT_SEQUENCE <= x"01";
      wait for CLK_1X_period;
      -- Testfall 2
      BIT_SEQUENCE <= x"12";
      wait for CLK_1X_period;
      -- hier weitere Testfälle eintragen
    end loop;
    wait;
  end process Data_STIM;

  -- Instanziierung der zu simulierenden Komponente
  uut: ISERDES_1_ZU_7
    port map(CLK_1X => CLK_1X,
             CLK_7X => CLK_7X,
             RST_I  => RST_I,
             D_I    => D_I,
             D_O    => D_O);

  -- Generierung des Rahmentaktes
  CLK_1X_proc: process
  begin
    CLK_1X <= '1';
    wait for CLK_1X_period/2;
```

```
    CLK_1X <= '0';
    wait for CLK_1X_period/2;
  end process CLK_1X_proc;

  -- Generierung des Bittaktes
  CLK_7X_proc :process
  begin
    CLK_7X <= '0';
    wait for CLK_7X_period/2;
    CLK_7X <= '1';
    wait for CLK_7X_period/2;
  end process CLK_7X_proc;

  -- System-Reset
  REST_proc: process
  begin
    RST_I <= '1';
    wait for RESET_TIME;
    RST_I <= '0';
    wait;
  end process REST_proc;

  -- Generierung des seriellen Datenstroms
  RX_STIM: process
  begin
    -- 2-Bit des verangegangenen Zyklus
    for i in 1 downto 0 loop
      D_I <= BIT_SEQUENCE(i);
      wait for CLK_7X_period;
    end loop;
    -- 5-Bit des aktuellen Zyklus
    for i in 6 downto 2 loop
      D_I <= BIT_SEQUENCE(i);
      wait for CLK_7X_period;
    end loop;
  end process RX_STIM;

end;
```

Listing 3: VHDL-Code der Testbench *TestBench_ISERDES*

A5 C-Code für das Auslesen und Beschreiben von Softwareregistern des Videocontrollers CH7301C

Die C-Funktionen `SendIIC(…)`, `ReadIIC(…)` und `ReadIICReg(…)` greifen über die IIC-Komponente des MicroBlaze-Systems per I²C-Bus auf die Softwareregister des Videocontrollers CH7301C zu. In der Header-Datei *CH7301C_Regs.h* sind die Funktionsprototypen deklariert und die Softwareregister des Videocontrollers und ihre Adressen in Präprozessordirektiven definiert. In der C-Datei *CH7301C_Regs.c* befinden sich die ausprogrammierten, oben aufgelisteten C-Funktionen und ein char-String-Feld für die Bildschirmausgabe der Softwareregisternamen.

```
#ifndef CH7301C_REGS_H
#define CH7301C_REGS_H

#include "xparameters.h"
#include "XStatus.h"
#include "xiic.h"
#include "xio.h"

#define IIC_ADDRESS 0x76
#define REG_ADDRESS 0x33

/****************** Register Map *********************/
#define REG_OFFSET_CM
#define CM 0x1C
#define IC 0x1D
#define GPIO 0x1E
#define IDF 0x1F
#define CD 0x20
#define DC 0x21
#define HPD 0x23
#define TCTL 0x31
#define TPCP 0x33
#define TPD 0x34
#define TPVT 0x35
#define TPF 0x36
#define TCT 0x37
#define TSTP 0x48
#define PM 0x49
#define VID 0x4A
#define DID 0x4B
#define DSP 0x56


/*Ein Datebyte per I²C senden*/
XStatus SendIIC(Xuint8, Xuint8);

/*Ein Datenbyte per I²C einlesen*/
XStatus ReadIIC(Xuint8, Xuint8 *);

/*Ein Daytenbyte auslesen und formatiert ausgeben*/
void ReadIICReg(Xuint8);

#endif
```

Listing 4: Header-Datei *CH7301C_Regs.h*

```
#include "CH7301C_Regs.h"


/*********************************************************************
 *              Register-Namen für die Konsolenausgabe               *
 *********************************************************************/
char RegNames[][28]={"Clock-Mode",
```

```
                    "Input-Clock",
                    "GPIO-Control",
                    "Input-Data-Format",
                    "Connection-Detect",
                    "DAC-Control","",
                    "Hot-Plug-Detection", //0x23
                    "","","","","","","","",
                    "","","","","","",    //0x30
                    "DVI-Control-Input","",
                    "DVI-PLL-Charge-Pump-Control",
                    "DVI-PLL-Divider",
                    "DVI-PLL-Supply-Control",
                    "DVI-PLL-Filter",
                    "DVI-Clock-Test",  //0x37
                    "","","","","","","","","",
                    "","","","","","","","","",
                    "Test-Pattern-Control",
                    "Power-Management",
                    "Version-ID",
                    "Device-ID",
                    "","","","","","",
                    "","","","","","",
                    "DVI-Sync-Polarity"};


/*****************************************************************
 * Funktion zum Senden eines Datenbytes über I²C.                *
 *                                                               *
 * Parameter:                                                    *
 * 'RegAddress' - Adresse des zu beschreibenden Registers        *
 * 'data'       - zu schreibendes Byte                           *
 *                                                               *
 * Return:                                                       *
 * kein Fehler  - 0                                              *
 * Fehler       - Anzahl gesendeter Bytes (1 oder 0) im 2.Byte   *
 *                und eine '1' im LSB                            *
 *****************************************************************/
XStatus SendIIC(Xuint8 RegAddress, Xuint8 data)
{
  Xuint8 WriteBuffer[2];
  unsigned SentByteCount=0;

  WriteBuffer[0] = RegAddress;
  WriteBuffer[1] = data;

  SentByteCount = XIic_Send(XPAR_IIC_0_BASEADDR, IIC_ADDRESS,
                  WriteBuffer, 2, XIIC_STOP);

  if(SentByteCount != 2)
  //{
    //xil_printf("Fehler in SendIIC: %d von 2 Bytes gesendet.\n",SentByteCount);
    return( ((XStatus)(SentByteCount<<16)) | ((XStatus)(0x1)) );
  //}

  return XST_SUCCESS;
}

/*****************************************************************
 * Funktion zum Empfangen eines Datenbytes über I²C.             *
 *                                                               *
 * Parameter:                                                    *
 * 'RegAddress' - Adresse des einzulesenen Registers             *
 * 'ReadBuffer' - Speicher für das einzulesene Datenyte          *
 *                                                               *
 * Return:                                                       *
 * kein Fehler  - 0                                              *
 * Fehler       - Anzahl eingelesener Bytes (gewöhnlich 0) im 2.Byte*
 *                und eine '1' im LSB                            *
 *****************************************************************/
XStatus ReadIIC(Xuint8 RegAddress, Xuint8 *ReadBuffer)
{
  unsigned ReceivedByteCount=0;
  Xuint16 StatusReg;
```

```
  do
  {
    StatusReg = XIo_In8(XPAR_IIC_0_BASEADDR + XIIC_SR_REG_OFFSET);

    if(!(StatusReg & XIIC_SR_BUS_BUSY_MASK))
    {
      ReceivedByteCount = XIic_Send(XPAR_IIC_0_BASEADDR, IIC_ADDRESS,
                          &RegAddress, 1,XIIC_STOP);
                          // XIIC_REPEATED_START XIIC_STOP
    }
  }while(ReceivedByteCount != 1);

  ReceivedByteCount = XIic_Recv(XPAR_IIC_0_BASEADDR, IIC_ADDRESS,
                                ReadBuffer, 1, XIIC_STOP);//XIIC_REPEATED_START);
  if(ReceivedByteCount != 1)
  //{
    //xil_printf("Fehler in ReadIIC: %d von 1 Byte empfangen.\n",ReceivedByteCount);
    return( ( (XStatus)(ReceivedByteCount<<16)) | ((XStatus)(0x1) ) );
  //}
  else
    return(XST_SUCCESS);
}

/*********************************************************************
 * Funktion zur formatierten Ausgabe eines über I²C auszulesenen    *
 * Registers mit seinem Namen.                                       *
 *                                                                   *
 * Parameter:                                                        *
 * 'RegAddress' - Adresse des auszulesenen Registers                 *
 *********************************************************************/
void ReadIICReg(Xuint8 RegAddress)
{
  Xuint8 ReadBuffer;
  Xuint8 diff;
  Xuint8 i=0;

  if(!ReadIIC(RegAddress,&ReadBuffer)) //Register auslesen
  {//Registernamen ausgeben
    xil_printf("%s-Register: ",RegNames[RegAddress-REG_OFFSET]);
    //Alle Ausgaben gleichmäßig ausrichten
    diff = sizeof(RegNames[0]) - strlen(RegNames[RegAddress-REG_OFFSET]) - 1;
    if(diff)
      for(i=0;i<diff;i++)
        print(" ");
    //ausgelesenes Register hexadezimal ausgeben
    xil_printf("0x%02X   [",ReadBuffer);
    //ausgelesenes Register binär ausgeben
    for(i=0;i<8;i++)
    {
      if(i==4)
        print(" ");
      if(ReadBuffer&(128>>i))
        print("1");
      else
        print("0");
    }
    print("](Bin)\n");
  }
  else //Fehler
    print("Fehler beim Auslesen aufgetreten!\n");
}
```

Listing 5: C-Code *CH7301C_Regs.c*

A6 User Constraint File *ML507.ucf* mit den wichtigsten Pinbelegungen des Evaluierungsboards ML507

Nachfolgend sind die wichtigsten Pinouts des Evaluierungsboards ML507 zusammengetragen und aufgelistet.

```
##########################################################
#                 Active-Low-Reset-Eingang               #
##########################################################
NET "NOT_RST_I" LOC = "E9"; # Active-Low

##########################################################
#                   200MHz-Oszillator                    #
##########################################################
NET "CLK200_I_N"  LOC = "K19";
NET "CLK200_I_P"  LOC = "L19";

##########################################################
#              4-Himmerlsrichtungen-LEDs                 #
##########################################################
NET "LED_C" LOC = "E8";   #LED Center
NET "LED_N" LOC = "AF13"; #LED North
NET "LED_S" LOC = "AG12"; #LED South
NET "LED_E" LOC = "AG23"; #LED East
NET "LED_W" LOC = "AF23"; #LED West

##########################################################
#                     Error-LEDs                         #
##########################################################
NET "LED_Err1" LOC = "F6";  #LED Error1
NET "LED_Err2" LOC = "T10"; #LED Error2

##########################################################
#                      GP-LEDs                           #
##########################################################
NET "LEDs(0)" LOC = "H18"; #LED0
NET "LEDs(1)" LOC = "L18"; #LED1
NET "LEDs(2)" LOC = "G15"; #LED2
NET "LEDs(3)" LOC = "AD26"; #LED3
NET "LEDs(4)" LOC = "G16"; #LED4
NET "LEDs(5)" LOC = "AD25"; #LED5
NET "LEDs(6)" LOC = "AD24"; #LED6
NET "LEDs(7)" LOC = "AE24"; #LED7

##########################################################
#                    DIP-Swithces                        #
##########################################################
NET "DIP1" LOC = "U25";  #DIP1
NET "DIP2" LOC = "AG27"; #DIP2
NET "DIP3" LOC = "AF25"; #DIP3
NET "DIP4" LOC = "AF26"; #DIP4
NET "DIP5" LOC = "AE27"; #DIP5
NET "DIP6" LOC = "AE26"; #DIP6
NET "DIP7" LOC = "AC25"; #DIP7
NET "DIP8" LOC = "AC24"; #DIP8

##########################################################
#                  HDR 1-Pinreihe (J6)                   #
##########################################################
NET HDR1_2  LOC = H33;  #HDR1_2
NET HDR1_4  LOC = F34;  #HDR1_4
NET HDR1_6  LOC = H34;  #HDR1_6
NET HDR1_8  LOC = G33;  #HDR1_8
NET HDR1_10 LOC = G32;  #HDR1_10
NET HDR1_12 LOC = H32;  #HDR1_12
NET HDR1_14 LOC = J32;  #HDR1_14
```

```
NET HDR1_16 LOC = J34;  #HDR1_16
NET HDR1_18 LOC = L33;  #HDR1_18
NET HDR1_20 LOC = M32;  #HDR1_20
NET HDR1_22 LOC = P34;  #HDR1_22
NET HDR1_24 LOC = N34;  #HDR1_24
NET HDR1_26 LOC = AA34; #HDR1_26
NET HDR1_28 LOC = AD32; #HDR1_28
NET HDR1_30 LOC = Y34;  #HDR1_30
NET HDR1_32 LOC = Y32;  #HDR1_32
NET HDR1_34 LOC = W32;  #HDR1_34
NET HDR1_36 LOC = AH34; #HDR1_36
NET HDR1_38 LOC = AE32; #HDR1_38
NET HDR1_40 LOC = AG32; #HDR1_40
NET HDR1_42 LOC = AH32; #HDR1_42
NET HDR1_44 LOC = AK34; #HDR1_44
NET HDR1_46 LOC = AK33; #HDR1_46
NET HDR1_48 LOC = AJ32; #HDR1_48
NET HDR1_50 LOC = AK32; #HDR1_50
NET HDR1_52 LOC = AL34; #HDR1_52
NET HDR1_54 LOC = AL33; #HDR1_54
NET HDR1_56 LOC = AM33; #HDR1_56
NET HDR1_58 LOC = AJ34; #HDR1_58
NET HDR1_60 LOC = AM32; #HDR1_60
NET HDR1_62 LOC = AN34; #HDR1_62
NET HDR1_64 LOC = AN33; #HDR1_64

##########################################################
#                  HDR_2-Pinreihe (J4)                   #
##########################################################
NET HDR2_2  LOC = K34;  #HDR2_2
NET HDR2_4  LOC = L34;  #HDR2_4
NET HDR2_6  LOC = K32;  #HDR2_6
NET HDR2_8  LOC = K33;  #HDR2_8
NET HDR2_10 LOC = N32;  #HDR2_10
NET HDR2_12 LOC = P32;  #HDR2_12
NET HDR2_14 LOC = R34;  #HDR2_14
NET HDR2_16 LOC = T33;  #HDR2_16
NET HDR2_18 LOC = R32;  #HDR2_18
NET HDR2_20 LOC = R33;  #HDR2_20
NET HDR2_22 LOC = T34;  #HDR2_22
NET HDR2_24 LOC = U33;  #HDR2_24
NET HDR2_26 LOC = U31;  #HDR2_26
NET HDR2_28 LOC = U32;  #HDR2_28
NET HDR2_30 LOC = V33;  #HDR2_30
NET HDR2_32 LOC = V32;  #HDR2_32
NET HDR2_34 LOC = V34;  #HDR2_34
NET HDR2_36 LOC = W34;  #HDR2_36
NET HDR2_38 LOC = AA33; #HDR2_38
NET HDR2_40 LOC = Y33;  #HDR2_40
NET HDR2_42 LOC = AE34; #HDR2_42
NET HDR2_44 LOC = AF34; #HDR2_44
NET HDR2_46 LOC = AE33; #HDR2_46
NET HDR2_48 LOC = AF33; #HDR2_48
NET HDR2_50 LOC = AD34; #HDR2_50
NET HDR2_52 LOC = AC34; #HDR2_52
NET HDR2_54 LOC = AB32; #HDR2_54
NET HDR2_56 LOC = AC32; #HDR2_56
NET HDR2_58 LOC = AB33; #HDR2_58
NET HDR2_60 LOC = AC33; #HDR2_60
NET HDR2_62 LOC = AP32; #HDR2_62
NET HDR2_64 LOC = AN32; #HDR2_64
```

Listing 6: Auflistung des Großteils nutzbarer FPGA-Pins des ML507 Evaluierungsboards

A7 Analyse über die Polynomapproximation für die Fahrspurerkennung auf autonomen Fahrzeugen

In diesem Kapitel werden die Lagrange- und Newton-Interpolation als Ansätze für die Fahrspurerkennung auf autonomen Fahrzeugen analysiert und der Ressourcenbedarf für die Implementierung der Newton-Interpolation auf einem FPGA aufgelistet.

Verfahren der Fahrspurerkennung

Fahrbahnmarkierungen auf Straßen dienen als Vorgabe der Fahrspur. Diese geben Aufschluss darüber, welcher Lenksollwinkel zu realisieren und welche Geschwindigkeit einzuhalten ist, um dem Straßenverlauf folgen zu können. In Abbildung 68 ist beispielhaft die Aufnahme eines Fahrbahnteilabschnittes dargestellt. Die Aufnahme sei bereits binarisiert und skelettiert worden, sodass Informationen über den Verlauf der Fahrspur (schwarze Linien) in der Breite eines Bildpunktes vorliegen (vgl. [2]). Betrachtet wird eine der beiden Fahrspuren.

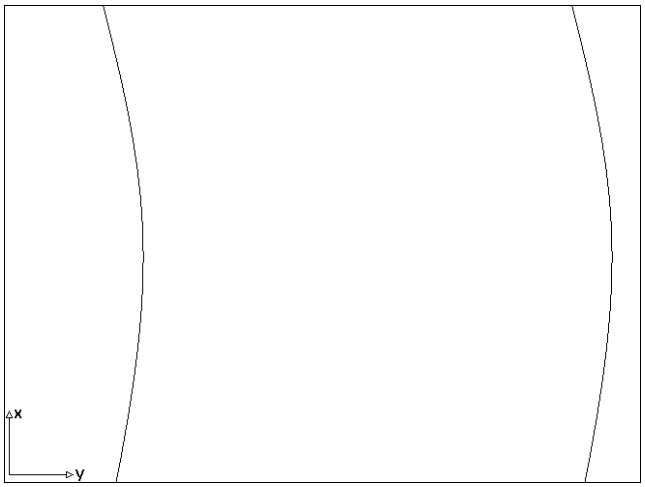

Abbildung 68: Aufnahme einer Fahrbahn,
binarisiert, skelletiert und invertiert

Wird die x-Achse als vertikale und die y-Achse als horizontale Achse mit ihrem Ursprung unten links festgelegt, so lässt sich die Fahrbahnmarkierung als mathematische Funktion $y = f(x)$ darstellen. Die mathematische Funktion des in Abbildung 68 dargestellten Verlaufs lässt durch eine Polynomfunktion $p_n(x)$ annähern, deren Grad n von der Anzahl der gewählten Stützstellen (x,y-Wertepaare) abhängt. Nachfolgend werden mathematische Verfahren der Polynomapproximation und deren Hardwarearchitekturen zur Implementierung mit einem FPGA vorgestellt.

Polynomapproximation für die Fahrspurerkennung

Die Polynomapproximation dient der Annäherung einer Funktionsbeschreibung an einen durch Stützstellen repräsentierten Funktionsverlauf. Die aus n+1 Stützstellen resultierende Polynomfunktion $p_n(x)$ hat bei der Lagrange- und Newton-Interpolation dabei maximal den Grad n.

Die Lagrange-Interpolation zur Polynomapproximation

Bei der Lagrange-Interpolation ([15], [28]) wird aus n+1 paarweise verschiedenen Stützstellen mit ihren Stützwerten durch

$$p_n(x) = \sum_{j=0}^{n} f_j \cdot L_j(x) \tag{5}$$

mit

$$L_j(x) = \prod_{i=0, i\neq j}^{n} \frac{(x - x_i)}{(x_j - x_i)} \tag{6}$$

und

$$\begin{aligned} &L_j - \textit{Lagrangesche Basisypolynome} \\ &x_j - \textit{Stützstellen} \\ &f_j - \textit{Stützwerte} \end{aligned} \tag{7}$$

genau ein Polynom p_n n-ten Grades bestimmt, das die aus dem Funktionsverlauf entnommenen Stützstellen durchläuft. Die Erweiterung der Anzahl an Stützpunkten erfordert die erneute Durchführung sämtlicher zuvor getätigter Rechenschritte.

Die Newton-Interpolation

Bei der Newton-Interpolation ([15], [28]) wird ebenfalls aus n+1 Wertepaaren das Polynom

$$\begin{aligned} &p_n(x) = c_0 + c_1 \cdot (x - x_0) + c_2 \cdot (x - x_0)(x - x_1) \\ &+ c_3 \cdot (x - x_0) \cdot (x - x_1) \cdot (x - x_2) + \ldots \end{aligned} \tag{8}$$

n-ten Grades bestimmt. Die Berechnung der Koeffizienten $c_0 \ldots c_n$ erfolgt durch

$$f_{i\ldots k} = \frac{f_{i\ldots k-1} - f_{i+1\ldots k}}{x_i - x_k} \tag{9}$$

und

$$c_k = f_{0\ldots k}. \tag{10}$$

Zur Übersicht wird folgendes Dreiecksschema aufgestellt:

$$\begin{array}{llllll} x_i & y_i & & & & \\ & & & & & \\ x_0 & y_0 (= f_0) & & & & \\ x_1 & y_1 & f_{01} & & & \\ x_2 & y_2 & f_{11} & f_{012} & & \\ x_3 & y_3 & f_{21} & f_{123} & f_{0123} & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{array} . \tag{11}$$

Das Hinzufügen jedes neuen Wertepaares hat die Erweiterung des Dreiecksschemas (6) um eine weitere Zeile zur Folge.

Rechenbeispiel

Die Newton-Interpolation eignet sich im Gegensatz zur Lagrange-Interpolation für die Polynomapproximation aus fortlaufend ermittelten Stützstellen. Die Lagrange-Interpolation setzt das vorab Vorhandensein einer festen Anzahl von Wertepaaren voraus, was unter Echtzeitbedingungen, wie sie bei Fahrassistenzsystemen herrschen, unpraktikabel, da sämtliche Wertepaare erst zwischengespeichert werden müssten. Die Newton-Interpolation hingegen erlaubt die schrittweise Berechnung des Polynoms $p_n(x)$, d.h. aufgenommene Wertepaare werden direkt zur Polynomberechnung geführt und müssen nicht zunächst allesamt zwischengespeichert werden. Aus diesem Anlass wird auf die Lagrange-Interpolation im Folgenden nicht weiter eingegangen. Die schrittweise Rechenoperation der Newton-Interpolation demonstriert das folgende Zahlenbeispiel 5 zur Veranschaulichung in ein Koordinatensystem eingetragenen Wertepaare (vgl. Abbildung 69).

i	x_i	y_i
0	1	1
1	3	3
2	4	2
3	6	4
4	9	6

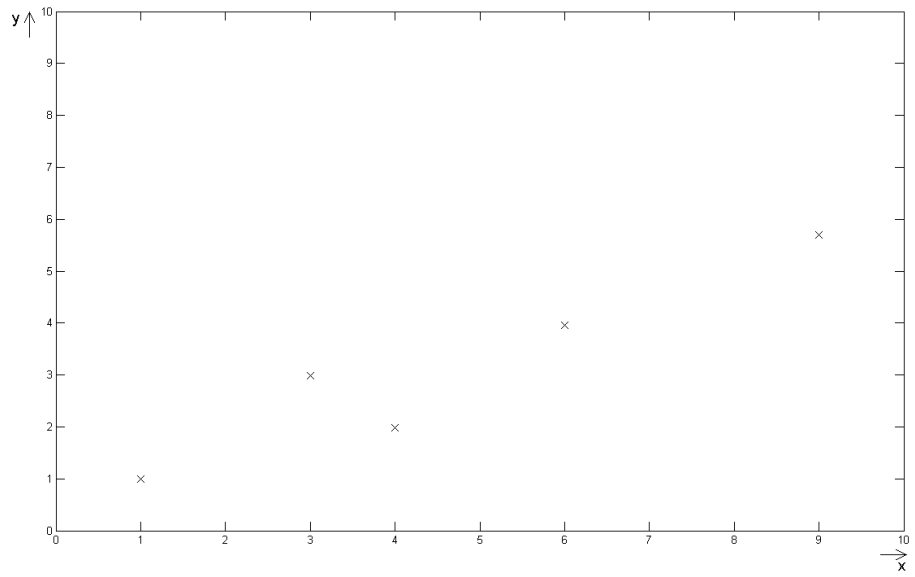

Abbildung 69: Wertetabelle links, Koordinatensystem mit den eingezeichneten Wertepaaren rechts

Der erste Schritt besteht in der Berechnung der Koeffizienten. Dazu wird das Dreiecksschema aus (6) aufgestellt:

i	x_i	y_i				
0	1	1				
1	3	3	$\frac{3-1}{3-1}=1$			
2	4	2	$\frac{2-3}{4-3}=-1$	$\frac{-1-1}{4-1}=-\frac{2}{3}$		
3	6	4	$\frac{4-2}{6-4}=1$	$\frac{1-(-1)}{6-3}=\frac{2}{3}$	$\frac{\frac{2}{3}-\left(-\frac{2}{3}\right)}{6-1}=0.267$	
4	9	6	$\frac{6-4}{9-6}=\frac{2}{3}$	$\frac{\frac{2}{3}-1}{9-4}=-\frac{1}{15}$	$\frac{-\frac{1}{15}-\frac{2}{3}}{9-3}=-0.122$	$\frac{-0.122-0.267}{9-1}=-0.049$

(12

Aus dem Dreiecksschema (7) ergeben sich folgende Werte für die Koeffizienten:

$$\begin{array}{ll} c_0 = 1 & c_3 = 0.267 \\ c_1 = 1 & c_4 = -0.049 \\ c_2 = -\frac{2}{3} & \end{array} \qquad (13)$$

Die gegebenen Stützwerte $x_0 ... x_4$ werden nun in (3) eingesetzt

$$\begin{aligned} p_4(x) &= c_0 \\ &+ c_1 \cdot (x-1) \\ &+ c_2 \cdot (x-1) \cdot (x-3) \\ &+ c_3 \cdot (x-1) \cdot (x-3) \cdot (x-4) \\ &+ c_4 \cdot (x-1) \cdot (x-3) \cdot (x-4) \cdot (x-6) \end{aligned} \tag{14}$$

, die Klammern werden aufgelöst

$$\begin{aligned} p_4(x) &= c_0 \\ &+ c_1 \cdot (x-1) \\ &+ c_2 \cdot (x^2 - 4 \cdot x + 3) \\ &+ c_3 \cdot (x^3 - 8 \cdot x^2 + 19 \cdot x - 12) \\ &+ c_4 \cdot (x^4 - 14 \cdot x^3 + 67 \cdot x^2 - 126 \cdot x + 72) \end{aligned} \tag{15}$$

und die Gleichung folgendermaßen umgestellt:

$$\begin{aligned} p_4(x) &= x^4 \cdot c_4 \\ &+ x^3 \cdot (-14 \cdot c_4 + c_3) \\ &+ x^2 \cdot (67 \cdot c_4 - 8 \cdot c_3 + c_2) \\ &+ x \cdot (-126 \cdot c_4 + 19 \cdot c_3 - 4 \cdot c_2 + c_1) \\ &+ 72 \cdot c_4 - 12 \cdot c_3 + 3 \cdot c_2 - c_1 + c_0 \end{aligned} \tag{16}$$

.

Nach Einsetzen der Zahlenwerte aus (8) in (1) ergibt sich folgende, durch die Newton-Interpolation approximierte Polynomfunktion 4-ten Grades:

$$p_4(x) = -0.049 \cdot x^4 + 0.953 \cdot x^3 - 6.086 \cdot x^2 + 14.914 \cdot x - 8.732. \tag{17}$$

Der Funktionsverlauf der approximierten Polynomfunktion (vgl. Abbildung 70, blaue Linie) durchläuft erwartungsgemäß die Koordinaten der vorgegebenen Wertepaare (vgl. Abbildung 70, Kreuze).

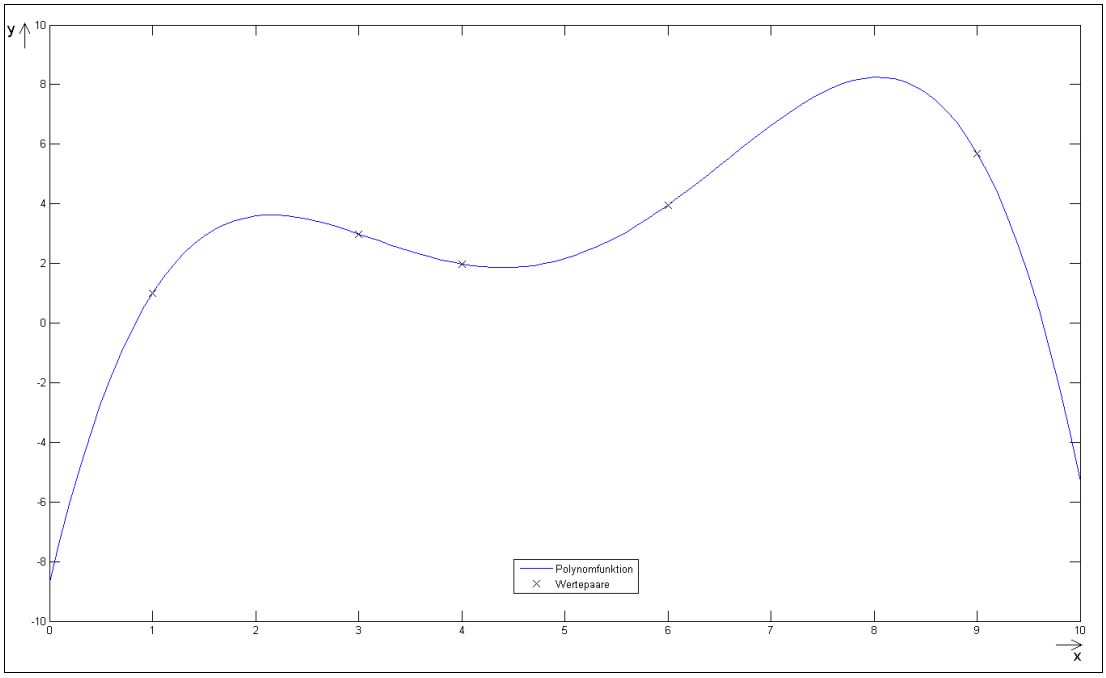

Abbildung 70: Funktionsverlauf der approximierten Polynomfunktion

Hardwarearchitektur der Newton-Polynomapproximation

Bei der Newton-Polynomapproximation werden die Koeffizienten c_i mit den mathematischen Operationen Subtraktion und Division berechnet (vgl. (7)) . Eine mögliche Hardwarearchitektur der Koeffizientenberechnung für ein Polynom 4. Grades ist in Abbildung 71 dargestellt. Es handelt sich dabei um einen Hardwareaufbau, der im Gegensatz zu einem ressource sharing-Modell, die Berechnungen parallel und damit schneller durchführt.

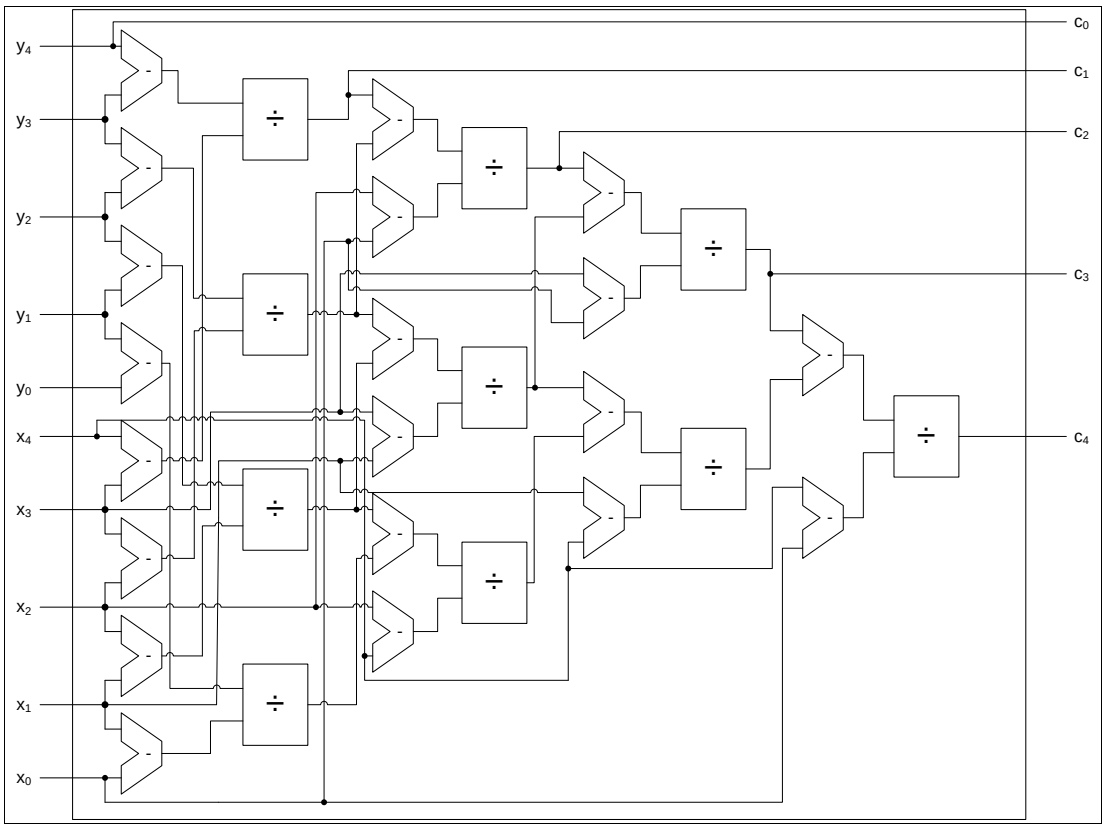

Abbildung 71: Hardwarearchitektur der Koeffizientenberechnung bei der Newton-Polynomapproximation für ein Polynom 4. Grades

Stützstellen x_i und Stützwerte y_i werden der Komponente parallel zugeführt, sodass die erste Stufe der dividierten Differenzen berechnet werden kann. Die Eingänge $x_0 \ldots x_4$ müssen bis zum Durchlauf der letzten (vierten) Stufe gehalten werden, da sie für diese Berechnungen erforderlich sind. Sowohl die Subtrahierer, als auch die Dividierer weisen eine bestimmte Signallaufzeit vom Eingang zum Ausgang auf. So wären an den Ausgängen der Dividierer Flipflops vorzusehen (hier nicht vorgenommen). Für ein Polynom 4. Gradeswürden 20 Subtrahierer und 10 Dividierer benötigt (vgl. Abbildung 71) . In Tabelle 10 ist der Hardwareressourcenbedarf (Subtrahierer, Dividierer) für eine verschiedene Anzahl von Stützstellen aufgelistet.

Stützstellen	Sub.	Div.	Stützstellen	Sub.	Div.
1	0	0	8	56	28
2	2	1	9	72	36
3	6	3	10	90	45
4	12	6	20	380	190
5	20	10	40	1560	780
6	30	15	50	2450	1225
7	42	21	51	2550	1275

Tabelle 10: Hardwareressourcenbedarf gegenüber Anzahl Stützstellen

Sind die Abstände der gewählten Stützstellen im Vorfeld bekannt, so entfällt die Hälfte der benötigten Subtrahierer, da die Differenzen der Stützstellen nicht erst berechnet werden müssen. Werden diese Abstände weiterhin so gewählt, dass ihre Differenzen eine Basis von 2 sind, so kann ein Teil der Dividierer durch Bitschiebeoperationen ersetzt werden. Sind die Koeffizienten ermittelt, müssen diese in die Polynomgleichung (vgl. (3)) eingesetzt werden. Dieser Rechenschritt besteht aus den Operationen Subtraktion/Addition und Multiplikation (vgl. (11)). Die zugehörige Hardwarearchitektur für ein Polynom 2. Grades (mit 3 Stützstellen) ist in Abbildung 72 dargestellt. Die im vorhergehenden Rechenschritt errechneten Koeffizienten $c_0 ... c_2$, sowie die ersten n der insgesamt $n+1$ Stützstellen dienen der Komponente als Eingangssignale. Am Ausgang erhält man die Faktoren $a_0 ... a_2$ der Polynomfunktion $p_2(x) = a_2 \cdot x^2 + a_1 \cdot x + a_0$. Dieser Entwurf setzt voraus, dass die Eingangssignale so lange am Eingang gehalten werden, bis die Faktorenberechnung beendet worden ist. Die Anzahl benötigter Hardwareelemente (Add., Sub., Mult.) variiert je nach Umstellung der Polynomfunktion

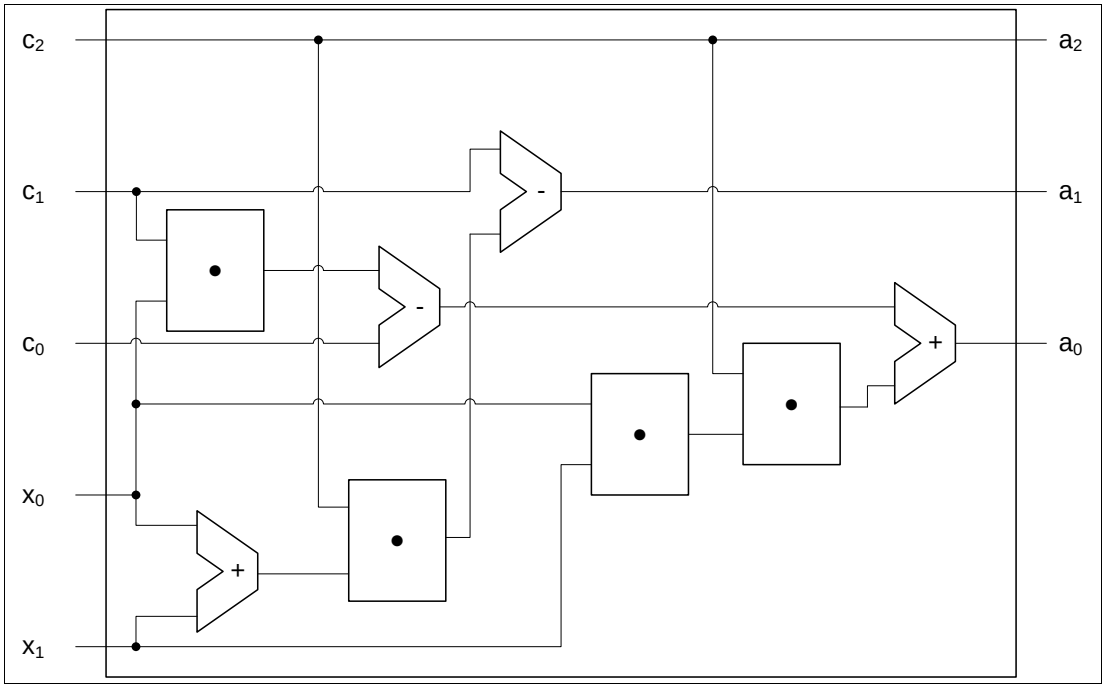

Abbildung 72: Hardwarearchitektur der Faktorenberechnung

Der Ressourcenbedarf für eine Newton-Interpolation für Polynome 5-ten Grades ist bereits so hoch, dass eine Implementierung mit FPGAs nicht praktikabel ist (vgl. Tabelle 10). In diesem Zusammenhang sei auf das Kalman-Filter als Fahrspurerkennungsalgorithmus verwiesen (vgl. [1]).

Versicherung über die Selbständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, den 10.09.2009 _____

Ort, Datum Unterschrift