Smart Mechatronics

Hochschule Bochum
Bochum University
of Applied Sciences

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Engineering (B.Eng.)

Modellbasierte Entwicklung einer FPGA-Logik für Echtzeit-Bildverarbeitung

Autor: Felix Stefan Schneider
felix.schneider@hs-bochum.de
Matrikelnummer: 015214561

Erstgutachter: Prof. Dr.-Ing. Arno Bergmann
Zweitgutachter: Christoph Peters, M.Sc.

Abgabedatum: 11.12.2018

Eidesstattliche Erklärung

Eidesstattliche Erklärung zur Abschlussarbeit:
»Modellbasierte Entwicklung einer FPGA-Logik für Echtzeit-Bildverarbeitung«

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Unterschrift : *Ort, Datum* :

Inhaltsverzeichnis

Abkürzungsverzeichnis

CLB Configurable Logic Block

FF Flip-Flop

FIFO First In First Out

FPGA Field Programmable Gate Array

HDL Hardware Description Language

HLS High Level Synthesis

IC Integrated Circuit

IOB Input/Output Block

LSB Least Significant Bit

LUT Look Up Table

MSB Most Significant Bit

QSPI Quad Serial Peripheral Interface

UART Universal Asynchronous Receiver Transmitter

Symbolverzeichnis

Symbol	Bedeutung	Einheit
t_{s}	Setzzeit (setup time)	s
t_{h}	Haltezeit (hold time)	s
t_{cq}	Ausgangsverzögerung (clock to Q delay)	s
t_{dly}, t_{trace}	Ausbreitungsverzögerung	s
f_{Takt}	Taktfrequenz	1/s

1 Einleitung

Industrieprozesse erfordern häufig die Messung von Längen oder Geschwindigkeiten geförderter Materialien wie beispielsweise Papier, Metall oder Kabel. [1]

Diese Messung kann entweder durch kontaktierende Messfühler oder berührungslose Sensoren erfolgen. Nachteilig bei taktilen Verfahren ist der möglicherweise auftretende Schlupf zwischen Messobjekt und Sensor; manche Prozesse schließen außerdem eine Berührung des Materials aus, um Beschädigungen der Oberfläche oder des Sensors zu verhindern. [2, S. 1] [1]

Ein Verfahren für die berührungslose Messung von Geschwindigkeiten durch Bildverarbeitung ist das Ortsfrequenzfilterverfahren. Schnelle Zeilenkameras, die hierfür verwendet werden können, sind zunehmend leistungsfähiger und preiswerter geworden und haben die Entwicklung von Ortsfrequenzfiltersensoren begünstigt. [2, S. 4]

Am Institut für Systemtechnik wurde in Kooperation mit der Smart Mechatronics GmbH im Zuge des Vader-Projekts (Velocimeter using Advanced Digital filtER) ein Ortsfrequenzfiltersensor geplant und ein Prototyp für dessen Hardware entwickelt [3].

Der schematische Aufbau des Vader ist in Abbildung 1.1 dargestellt.

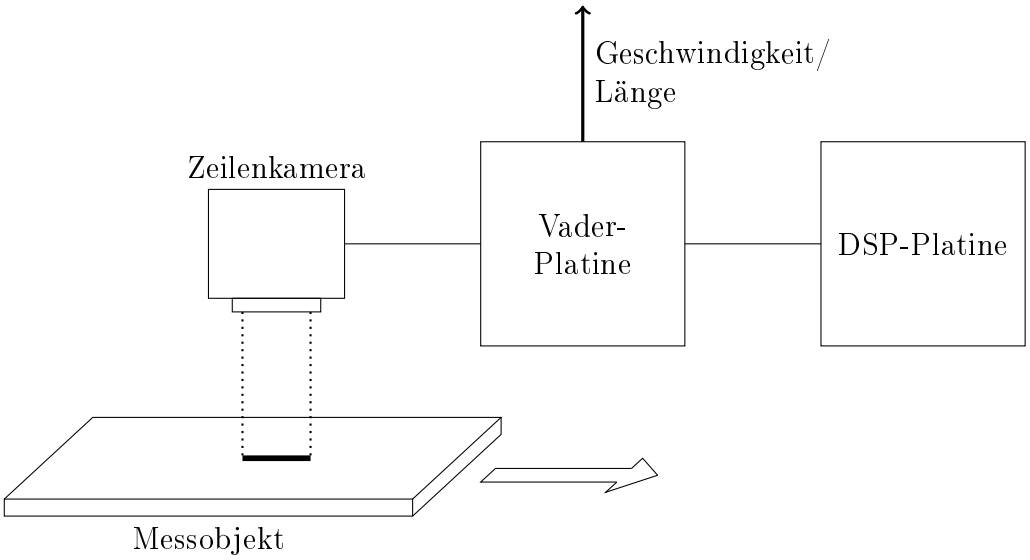


Abbildung 1.1: Aufbau des Vader-Prototypen

Eine Zeilenkamera nimmt Bilder entlang der Bewegungsrichtung eines Objekts auf.

Auf der Vader-Platine befindet sich ein konfigurierbarer Logikchip (FPGA), auf dem ein parametrierbarer Bildverarbeitungsalgorithmus implementiert ist.

Die gewonnenen Signale werden an eine zweite Platine mit einem digitalen Signalprozessor (DSP) übertragen, der die gesuchte Geschwindigkeit bzw. Länge berechnet und diese zurück an die Vader-Platine überträgt.

Von dort werden die Weginkremente über verschiedene Schnittstellen ausgegeben und können in einem Industrieprozess verwendet werden.

1.1 Aufgabenstellung

Gegenstand dieser Arbeit ist die Entwicklung der Firmware für das Vader-Projekt.

Zunächst werden die Anforderungen an die Bildverarbeitung und die Schnittstelle zwischen FPGA und DSP aus den Anforderungen an das Gesamtsystem abgeleitet.

Anschließend wird ein modellbasierter Ansatz mit Matlab/Simulink verfolgt, um entsprechende Lösungen zu entwickeln und zu verifizieren. Der modellbasierte Ansatz wird unter der Annahme gewählt, dass eine gemeinsame Umgebung für die Entwicklung und Verifikation mehrerer separater Teilsysteme auf unterschiedlicher Hardware (in diesem Fall FPGA und DSP) von Vorteil ist. Zudem wird erwartet, dass sich durch die Möglichkeiten, die Matlab/Simulink zur grafischen Auswertung von Simulationsergebnissen bietet, im Kontext des Ortsfrequenzfilters Möglichkeiten für Plausibilitätsprüfungen eröffnen.

Für die Anbindung der Simulink-Modelle an die Hardware werden Zwischenschichten (Harnesses) entwickelt und verifiziert.

Auf Seite des DSP befasst sich die vorliegende Arbeit mit der Harness, das Modell zur Verarbeitung der Ortsfrequenzfiltersignale ist nicht Teil des Projekts.

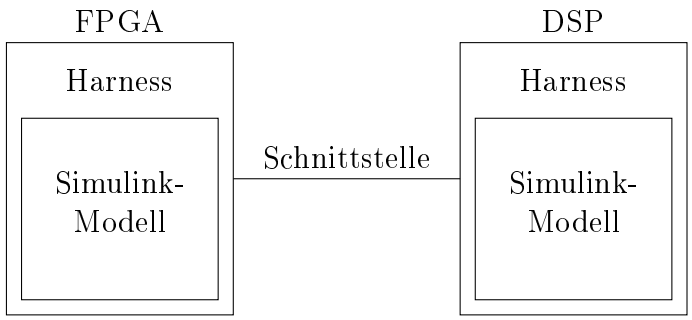


Abbildung 1.2: Architektur der Signalverarbeitungskette

2 Grundlagen

2.1 Ortsfrequenzfilter

Das Ortsfrequenzfilterverfahren beruht darauf, die Bewegung eines Objekts durch ein optisches Gitter zu beobachten. Unter der Bedingung, dass die Oberfläche des Objekts nicht perfekt homogen ist, ist die Intensität des vom Objekt reflektierten und das Gitter passierenden Lichts mit einer Frequenz periodisch, die proportional zur Bewegungsgeschwindigkeit des Objekts ist. [2, S. 7]

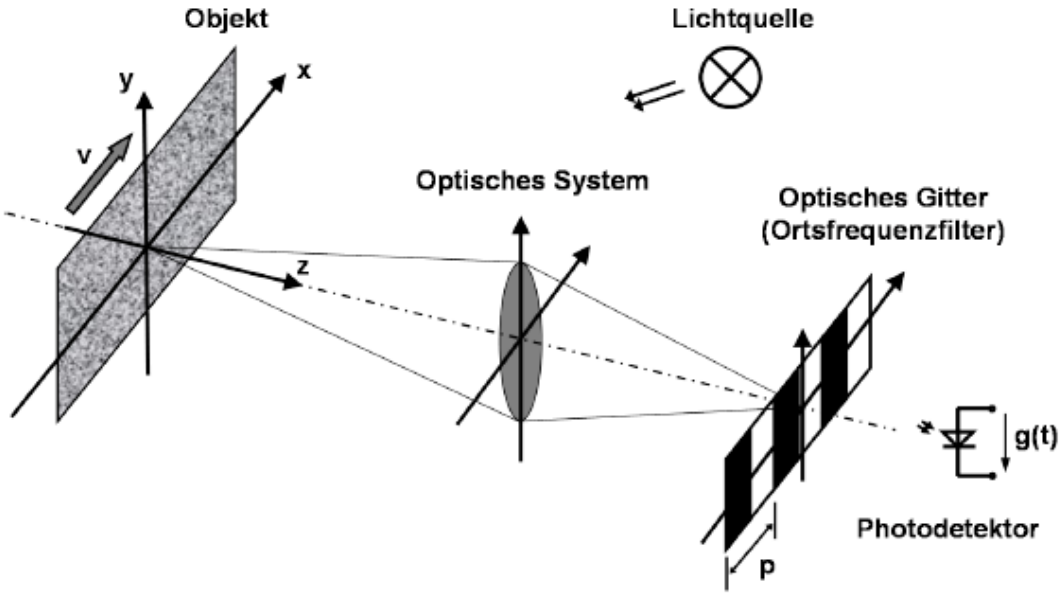


Abbildung 2.1: Prinzip eines Ortsfrequenzfiltersensors [2]

Das Grundprinzip eines Ortsfrequenzfilters ist in Abbildung 2.1 dargestellt. Die Objektoberfläche bewegt sich entlang der x-Achse und wird dabei von der Licht-

quelle beleuchtet. Das reflektierte Licht wird auf das Ortsfrequenzgitter abgebildet und fällt dahinter auf den Photodetektor. Die Intensität des Lichts, das durch das Gitter hindurchfällt, ergibt über der Zeit aufgetragen die Ortsfrequenzfilterfunktion, deren Grundfrequenz abhängig von der Bewegungsgeschwindkeit des Objekts ist. [2]

Diese Arbeit befasst sich mit einem digitalen Ortsfrequenzfilter, in dem kein reales optisches Gitter existiert. Stattdessen fällt das von der Objektoberfläche reflektierte Licht auf einen Zeilenkamerasensor, der entlang der Bewegungsrichtung des Objekts ausgerichtet ist. Die Anwendung des optischen Gitters erfolgt digital durch die Bewertung des Zeilenbilds mit entsprechenden Bewertungskoeffizienten.

Für das folgende Beispiel wird vereinfachend eine Oberfläche mit einem einzigen Strukturmerkmal angenommen (Abbildung 2.2).

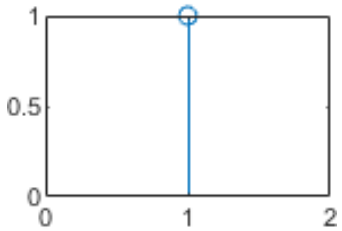


Abbildung 2.2: Beispielhaftes Strukturmerkmal mit der Breite 1 Pixel und Helligkeit 1

Als optisches Gitter wird ein sinusförmiges Bewertungsmuster gewählt.

Abbildung 2.3 zeigt im oberen Diagramm das Kamerabild des sich bewegenden Strukturmerkmals zu unterschiedlichen Zeitpunkten. Das mittlere Diagramm zeigt das stillstehende sinusförmige Gitter. Die Ortsfrequenzfilterfunktion im unteren Diagramm wird bestimmt, indem die Zeilenbilder für $t = 0, 1, 2...$ komponentenweise für jeden Pixelindex mit dem Filter multipliziert und das Ergebnis zu jeweils einem skalaren Wert aufaddiert wird, der für den entsprechenden Zeitpunkt in das untere Diagramm eingetragen wird. Dieser Vorgang entspricht dem Einsammeln des durch das Gitter fallenden Lichts im Photodetektor in Abbildung 2.1.

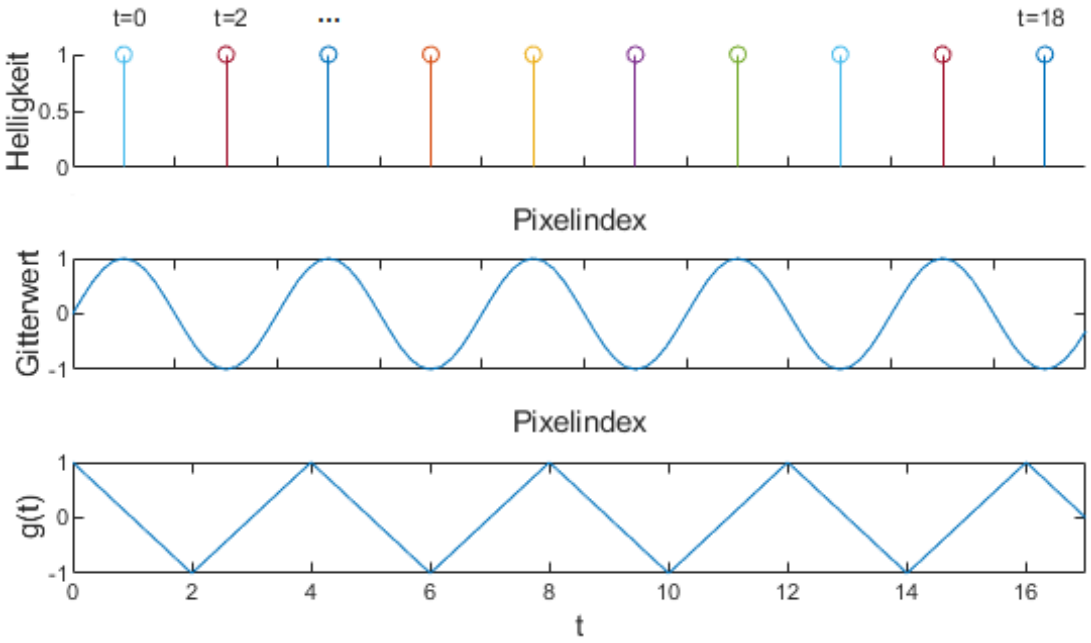


Abbildung 2.3: Beispiel Ortsfrequenzfilterung

Für das Beispiel wurde die Geschwindigkeit der Oberfläche aus Gründen der Übersichtlichkeit so gewählt, dass das Strukturmerkmal für ganzzahlige t immer auf ein Maximum, ein Minimum oder einen Nulldurchgang des Filters fällt. Würde sich das Strukturmerkmal nur mit der halben Geschwindigkeit am Filter vorbeibewegen, so entstünde ein Ortsfrequenzfiltersignal mit der doppelten Periode und der halben Frequenz.

Würde man die Abtastrate der Kamera erhöhen, so würde die Filterfunktion an zusätzlichen Zwischenstellen abgetastet werden und auch die Ortsfrequenzfilterfunktion würde einen sinusförmigeren Verlauf annehmen.

Das reale, digitale Ortsfrequenzfilter, das Thema dieser Arbeit ist, funktioniert nach dem gleichen Prinzip. Die Beiträge vieler Oberflächenmerkmale überlagen sich in der Realität zur Ortsfrequenzfilterfunktion [2, S. 9]. Sinusförmige Filter werden verwendet, weil für die Bestimmung der gesuchten Geschwindigkeit ausschließlich die Grundfrequenz der Ortsfrequenzfilterfunktion von Bedeutung ist und rechteckige Filter, wie in Abbildung 2.1, zu störenden Oberwellen führen [2, S. 20]

Neben der freien Wahl der Filterform und -anzahl liegt ein weiterer Vorteil eines digitalen Ortsfrequenzfilters darin, dass die Filterfunktion mit einer definierten Geschwindigkeit gegen das Kamerabild bewegt werden kann. Dies führt dazu, dass sich die Ausgangsfrequenzen verschieben: Ein stillstehendes Objekt führt durch die Bewegung des Filters zu einer messbaren Frequenzkomponente. Nach dem gleichen Prinzip kann durch die Bewegung des Filters auch eine Richtungserkennung durchgeführt werden. Dies ist mit einem stillstehenden Filter wie in dem Beispiel in Abbildung 2.3 nicht möglich. [2, S. 40]

2.2 FPGAs

Field programmable gate arrays (FPGAs) gehören zur Gruppe der programmierbaren Logikbausteine und ermöglichen die Implementierung von benutzerdefinierten digitalen Schaltungen in einem integrierten Schaltkreis (integrated circuit, IC) [4, S. 273ff]. Sie sind in der Regel rekonfigurierbar und erlauben dadurch die spätere Modifikation der Schaltung [5, S. 203].

FPGAs können eingesetzt werden, um große Datenmengen parallel zu verarbeiten und dabei höhere Rechenleistungen zu erreichen als ein Universal- oder Signalprozessor. [6, S. 251ff]

Im Vorfeld zu diesem Projekt wurden verschiedene Möglichkeiten für die Signalverarbeitungsarchitektur des Ortsfrequenzfiltersystems in Betracht gezogen. Die Vorverarbeitung der Bilddaten in einem FPGA wurde bei der spezifizierten Datenrate der Kamera (siehe [7]) als die einzige praktisch realisierbare Variante betrachtet.

Digitale Schaltungen werden für die Implementierung auf FPGAs in Hardwarebeschreibungssprachen (hardware description languages, HDLs) beschrieben und mit einem Syntheseprogramm für die Verwendung auf dem FPGA übersetzt. [5, S. 208]

Während des Vorprojekts wurde für dieses Projekt ein Xilinx Artix-7 FPGA ausgewählt [3], dessen für dieses Projekt relevanten Eigenschaften im Folgenden beschrieben werden.

2.2.1 Aufbau

Verbindungs- und Taktnetzwerke

Um Datensignale innerhalb des Bausteins zu verteilen, verbindet ein konfigurierbares Verbindungsnetzwerk (Interconnect) die verschiedenen Elemente des FPGAs. Die Ein- und Ausgangspins (IO-Pins) des Artix-7 sind über Puffer (IO-Blöcke, IOBs) mit dem Interconnect verbunden. Diese Puffer dienen als Schnittstelle zu anderen Schaltungsteilen außerhalb des FPGAs. [8]

Fast alle realen Digitalschaltungen sind synchrone Systeme, die ihre Zustände nur zu bestimmten Zeitpunkten ändern [4, S. 118]. Die Synchronisation wird über ein Taktsignal realisiert [5, S. 2].

Um eine Synchronisation der Schaltungsteile mit möglichst geringem zeitlichen Versatz zu ermöglichen, verfügen FPGAs über dedizierte Taktnetzwerke, über die Taktsignale innerhalb des ICs verteilt werden [4, S. 276].

Der Artix-7 bietet die Möglichkeit, Takte im gesamten IC (Global Clock Network) oder einzelnen Regionen (Clock Regions) zu verteilen. Externe Taktsignale können über spezielle IOBs in das Taktnetzwerk aufgenommen oder innerhalb des ICs durch Phasenregelschleifen (PLLs) oder Taktmanager (Mixed-Mode Clock Managers, MMCMs) auf Basis eines Referenztakts generiert werden. [9]

Logik

Die Logikelemente des Artix-7 Bausteins sind in konfigurierbaren Logikblöcken (CLBs) organisiert, die durch die Verbindungsmatrix entsprechend der zu implementierenden Funktion verschaltet werden können. [10]

Die CLBs enthalten Wahrheitstabellen (Look up tables, LUTs) mit mehreren Eingängen und einem oder mehreren Ausgängen [10]. Aus den LUTs kann kombinatorische Logik (z.B. AND- oder OR-Gatter) ohne Rückkopplung oder Speicherung aufgebaut werden [4, S. 85].

Um Schaltungen mit Zustandsspeicher realisieren zu können, enthalten die CLBs neben den LUTs auch Speicherelemente (Flip-Flops). [10]

Unter der Verwendung von Flip-Flops kann sequenzielle Logik aufgebaut werden, deren Ausgangswerte nicht nur von den aktuellen sondern auch den vorherigen Eingangswerten abhängen. [4, S. 115]

Die Betrachtungen für Flip-Flops in dieser Arbeit beziehen sich stets auf das taktenflankengesteuerte D-Flip-Flop, das in realen Schaltungen am häufigsten verwendet wird. Es übernimmt bei einer Taktflanke (Wechsel des Signalpegels von 0 auf 1 oder von 1 auf 0) den Wert seines Engangs an den Ausgang. [4, S. 122ff]

In dieser Arbeit wird stets davon ausgegangen, dass die Flip-Flops auf steigende Flanken reagieren.

Der Dateneingang des Flip-Flops wird mit dem Buchstaben D gekennzeichnet, der Takteingang mit einem Dreieck und der Ausgang mit dem Buchstaben Q. Zusätzlich kann das Flip-Flop über einen Clock Enable-Eingang verfügen; nur wenn dieser gesetzt ist, reagiert das Flip-Flop auf Taktflanken. [4, S. 127]

Spezialfunktionen

Um für größere, zusammenhängende Speicher keine CLB-Resourcen verwenden zu müssen, stehen im Artix-7 dedizierte RAM-Blöcke zur Verfügung [11]. Ein Anwendungsbeispiel hierfür ist die Zwischenspeicherung von Kamerabildern für die Ortsfrequenzfilterung.

Wie in Abschnitt 2.1 beschrieben, basiert die Berechnung der Ortsfrequenzfilterfunktion auf der Multiplikation von Bilddaten mit entsprechenden Filterkoeffizienten.

Binäre Multiplizierer können als kombinatorische Logik in CLBs implementiert werden, einige FPGAs verfügen allerdings über eigene Multiplizierer-Module. Der Vorteil dieses Ansatzes ist, dass keine CLB-Ressourcen für diesen Zweck verwendet werden müssen und sich höhere Durchsätze erzielen lassen. [4, S. 279]

Der in diesem Projekt verwendete Artix-7 Baustein verfügt über 70 DSP-Blöcke, die jeweils einen 18x25 bit Multiplizierer enthalten. [12]

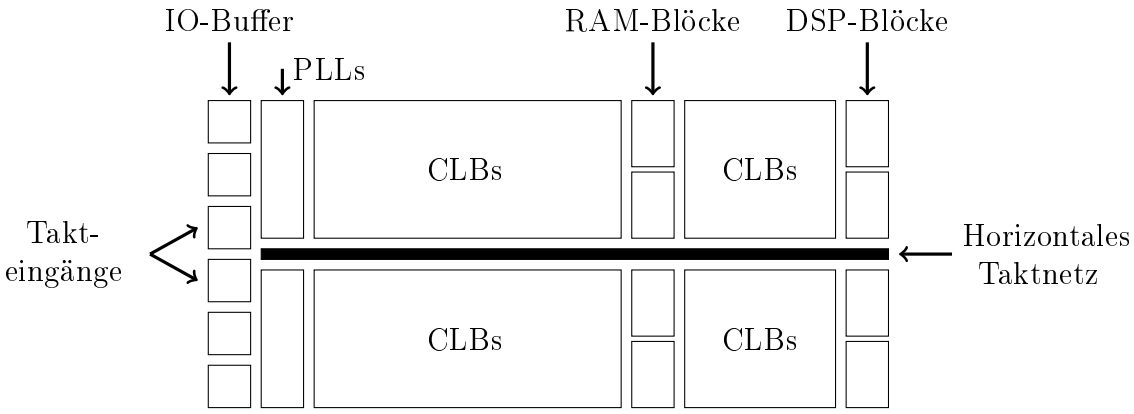


Abbildung 2.4: Ausschnitt des internen Aufbaus des verwendeten Artix-7

2.2.2 Taktrate und Timing

Da die Zustände eines synchronen Systems sich nur zu durch den Takt bestimmten Zeitpunkten ändern, bestimmt die Taktfrequenz den Durchsatz der digitalen Schaltung und beeinflusst ihre Leistungsfähigkeit. [4, S. 119]

Damit ein durch die Taktflanken angesteuertes Flip-Flop den Eingangswert korrekt übernimmt, muss dieser für eine bestimmte Setzzeit (Setup-Time, t_s) vor und für eine bestimmte Haltezeit (Hold-Time, t_h) nach der Flanke konstant sein. Werden die Setz- und Haltezeiten eingehalten, erscheint der Eingangswert mit einer Verzögerung gegenüber der Taktflanke (Clock-to-Q-Delay, t_cq) am Ausgang, andernfalls ist der Zustand undefiniert. [13, S. 216]

Abbildung 2.5 zeigt den Zeitverlauf eines Schaltvorgangs, bei dem die Setz- und Haltezeiten eingehalten werden.

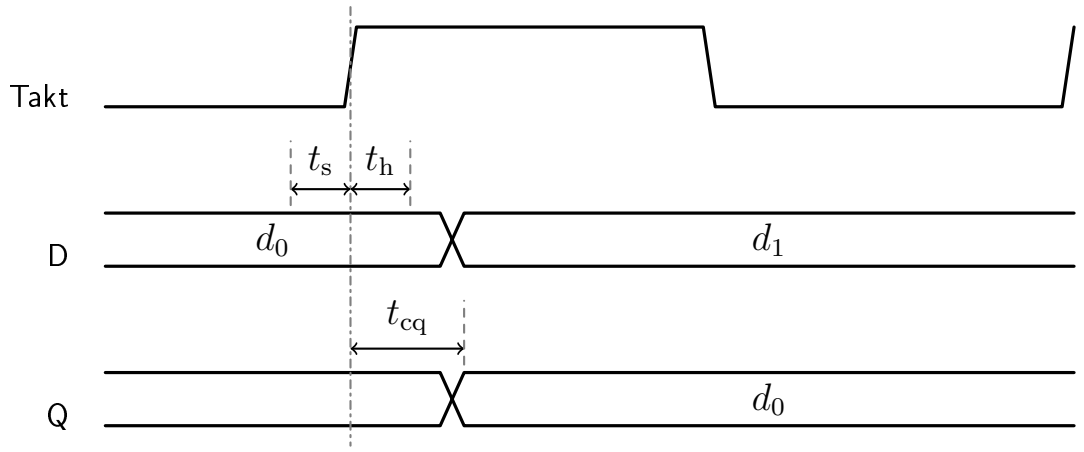


Abbildung 2.5: Timing-Parameter eines D-Flip-Flops

Auch die Logikelemente und Verbindungsnetzwerke in FPGAs sind nichtideal und verursachen Verzögerungen im Datenpfad. Die Setup- und Haltezeiten sowie das Clock-to-Q-Delay der Flip-Flops und die Verzögerungen der verschiedenen kombinatorischen Elemente (z.B. LUTs, Multiplikatoren) des Artix-7 sind im Datenblatt aufgeführt. [14]

Die Signallaufzeit zwischen zwei Elementen durch die Verbindungsmatrix ist von deren Platzierung innerhalb des ICs abhängig und wird während des Übersetzungsvorgangs durch das Syntheseprogramm abgeschätzt. [5, S. 208]

Signallaufzeiten von Ein- und Ausgangssignalen zwischen dem FPGA und anderen Teilen des Systems sind vom Übertragungskanal (z.B. Kabel, Leiterbahn) abhängig. Sie werden dem Syntheseprogramm als sogenannte Randbedingungen bekannt gemacht. [15]

2.2.3 Taktübergänge

Synchrone Schaltungsteile, die mit dem gleichen Takt angesteuert werden, werden als Taktbereich bezeichnet. Wenn ein Signal aus einem Taktbereich auf einen anderen Takt aufsynchronisiert wird, entsteht ein Taktübergang. [4, S. 193]

Taktübergänge entstehen zum Beispiel wenn in einem IC Daten von einem anderen IC mit anderem Systemtakt oder anderer Taktquelle empfangen und weiterverarbeitet werden sollen.

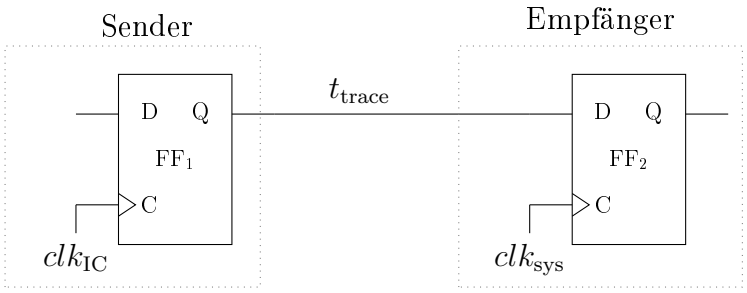


Abbildung 2.6: Beispiel eines Taktübergangs an einer UART-Schnittstelle

Abbildung 2.6 zeigt beispielhaft ein digitales System in dem Daten, die von einem externen IC über eine UART-Schnittstelle empfangen wurden, mit einem internen Systemtakt clk_{sys} verarbeitet werden.

Sender und Empfänger sind über eine Leiterbahn verbunden, die das Signal um t_{trace} verzögert. Die Gesamtverzögerungen t_{dly} vom Q-Ausgang des FF_1 zum D-Eingang des FF_2 besteht aus t_{trace} und weiteren Verzögerungen innerhalb der ICs z.B. durch die Verbindungsmatrizen.

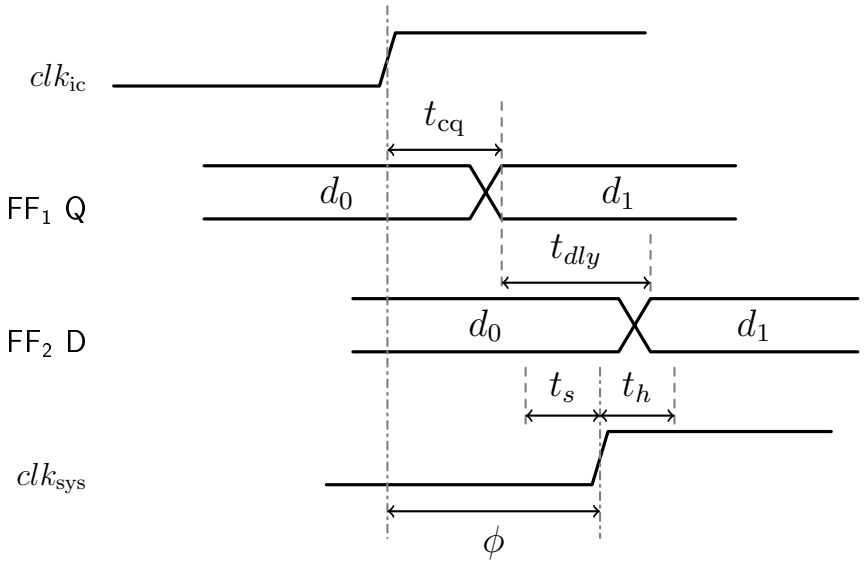


Abbildung 2.7: Taktübergang an einer UART-Schnittstelle

Das Timingdiagramm in Abbildung 2.7 zeigt jeweils eine Taktflanke des Systemtakts des externen ICs clk_{ic} und des Systemtakts des Empfängers clk_{sys}. Da die

beiden Takte aus unterschiedlichen Quellen stammen, lässt sich keine Aussage über die Phase zwischen den Taktflanken ϕ treffen.

In diesem Beispiel wurde ϕ so gewählt, dass die Haltezeit t_h des FF_2 verletzt wird, weil das Eingangssignal sich innerhalb dieser Zeitspanne nach der Flanke ändert. In diesem Fall kann das FF_2 in einen metastabilen Zustand übergehen, der Ausgang ist undefiniert und die nachfolgende Schaltung kann nicht zuverlässig arbeiten [13].

Das Auftreten von Metastabilität an einem Taktübergang lässt sich nie vollständig ausschließen, allerdings lässt sich die Auftrittswahrscheinlichkeit so weit wie für die Anwendung notwendig reduzieren. [16, S. 234]

Die Wahrscheinlichkeit, dass ein Flip-Flop nach dem Eintritt in die Metastabilität nicht wieder einen gültigen Logikzustand erreicht, sinkt exponentiell mit der Zeit. [17, S. 414]

Um ein asynchrones Signal auf einen Takt aufzusynchronisieren, können daher auf Empfängerseite mehrere Flip-Flops zu einem Synchronisierer hintereinandergeschaltet werden.

Die Wahrscheinlichkeit, dass sich ein metastabiler Zustand durch die Flip-Flops fortpflanzt sinkt mit deren Anzahl, gleichzeitig steigt mit jedem Flip-Flop die Latenz um eine Taktperiode. In der Praxis werden häufig Synchronisierer aus zwei Flip-Flops verwendet. Um mehrere Signale zwischen verschiedenen Taktbereichen zu übertragen, sind weitere Maßnahmen erforderlich. [16, S. 235]

2.2.4 FPGA-Entwurfsprozess

Als Ausgangspunkt bei der Konfiguration des FPGAs dient eine Beschreibung der digitalen Schaltung. Diese kann zum Beispiel in HDL erfolgen. Darüberhinaus bieten manche Hersteller die Möglichkeit zur Verwendung von Hochsprachen wie C oder C++ (High-Level Synthesis, HLS) oder Blockdiagrammen. [18]

Vor dem Übersetzungsvorgang kann die Schaltungsbeschreibung simuliert werden.

Neben der Funktion der Schaltung müssen auch ihre Randbedingungen definiert werden. Dazu zählt die Zuordnung von Ein- und Ausgängen der Schaltung zu

physikalischen Pins des FPGAs. Weitere Randbedingungen sind die Parameter der verwendeten Takte und Verzögerungen zwischen Takten und Ein- und Ausgangssignalen des FPGAs. Diese Information ist notwendig, damit das Syntheseprogramm die Einhaltung von Setz- und Haltezeiten verifizieren kann. [15] [30]

Während der Synthese wird die Beschreibung der Schaltung in eine Netzliste mit den verschiedenen Logikbausteinen (z.B. LUTs und Flip-Flops) übersetzt. In der anschließenden Implementierungsphase werden die Logikelemente aus der Netzliste so innerhalb des FPGAs platziert und verbunden, dass möglichst alle Randbedingungen eingehalten werden (Place & Route). Aus der Art und Platzierung der Logikelemente werden die Timing-Daten berechnet, die für eine anschließende Timing-Simulation verwendet werden können. Neben den Timing-Daten wird der Bitstream generiert, mit dem das FPGA für die beschriebene Funktion konfiguriert werden kann. [6]

Abbildung 2.8 fasst den FPGA-Entwicklungsprozess zusammen.

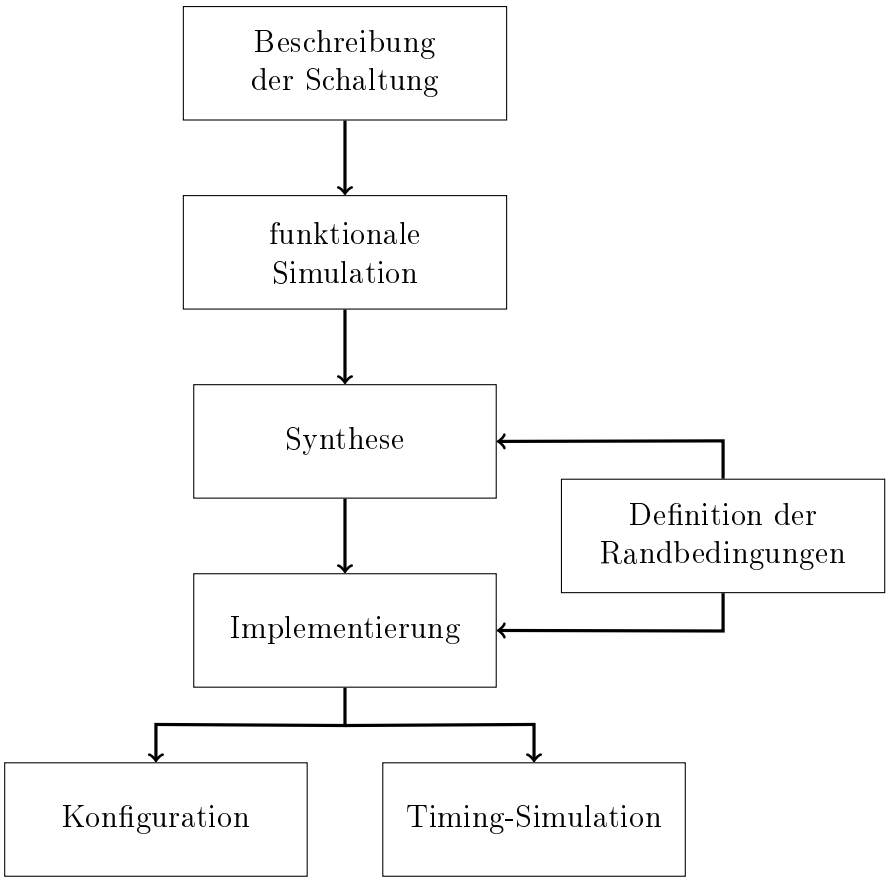


Abbildung 2.8: Schritte zur Erzeugung des FPGA-Bitstreams [6, S. 260ff] [18, S. 7]

2.3 Simulink & HDL Coder

In der vorliegenden Arbeit wird die Modellierungssoftware Matlab/Simulink verwendet um das Vader-System zu modellieren. Die HDL-Coder Toolbox ermöglicht das Generieren einer HDL-Schaltungsbeschreibung aus einem Simulink-Modell [19].

Die Schritte "Beschreibung der Schaltung" und "funktionale Simulation" aus Abbildung 2.8 lassen sich dadurch statt im Syntheseprogramm des FPGA-Herstellers in Matlab/Simulink durchführen.

2.3.1 Takte

Wie in Abbildung 1.1 dargestellt, ist die Vader-Platine mit verschiedenen anderen Systemen mit potentiell anderen Takten verbunden. Dieser Abschnitt behandelt den Umgang mit Takten in Simulink für die HDL-Codegenerierung.

Für die Simulation des digitalen Systems wird der diskrete Fixed-Step Solver von Simulink verwendet. Bei der Konfiguration dieses Solvers wird ein fundamentaler Zeitschritt festgelegt, der das kleinste Zeitintervall bestimmt, in dem die Zustände und Ausgänge des Systems berechnet werden. [20]

Teilen des Modells kann ein eigener diskreter Zeitschritt zugewiesen werden, der ein ganzzahliges Vielfaches der fundamentalen Schrittweite sein muss. Wird einem Block keine eigene Schrittweite zugewiesen, erbt er den Zeitschritt des Gesamtsystems und wird in jedem Zeitschritt ausgeführt. [20, S. 457]

Allgemein gilt für die Ausführungszeitpunkte t_n mit dem diskreten Zeitschritt T_s: [20, S. 444]

$$t_n = t_{n-1} + T_\mathrm{s} \tag{2.1}$$

Übertragen auf eine digitale Schaltung entspricht der fundamentale Zeitschritt dem kürzesten Zeitintervall ${f_\mathrm{Takt}}^{-1}$, in dem die synchronen Elemente neue Werte übernehmen.

Mit den Standardeinstellungen wird im generierten Code ein Takteingang erzeugt, der im FPGA-Harnesscode mit einem Takt aus dem Taktnetzwerk verbunden wird. Der HDL Coder erzeugt einen Takt-Controller, der für jede Schrittweite innerhalb des Modells basierend auf dem Systemtakt ein Clock Enable-Signal generiert, mit dem die entsprechenden synchronen Elemente angesteuert werden. [19, S. 886]

Weil die Schaltungsteile in diesem Fall trotzdem mit dem gleichen Takteingang angesteuert werden, sind sie Teil des selben Taktbereichs und es müssen keine besonderen Maßnahmen zur Synchronisation getroffen werden.

Wenn die Option "Multiple Clock Inputs" gewählt wird, wird im HDL-Code für jede Schrittweite ein eigener Takteingang erzeugt und der Nutzer kann die

synchronen Takte für die einzelnen Schrittweiten selbst generieren. Bei dieser Variante ist das Verhältnis der Eingangstaktfrequenzen durch das Verhältnis der Schrittweiten gegeben. [19, S. 886]

Um asynchrone Taktbereiche zu modellieren, können Triggered Subsystems mit der Option "Use Trigger As Clock" verwendet werden. In diesem Fall wird ein Taktsignal direkt an den Triggereingang eines Subsystems geführt und steuert die getakteten Elemente innerhalb des Subsystems an. Wegen dem potentiell unbekanntem Phasenverhältnis zwischen den Takten innerhalb und außerhalb des Subsystems erzwingt Simulink jeweils einen Delay-Block an den Ein- und Ausgängen des Subsystems. [19, S. 875]

Die Delay-Blöcke übernehmen in jedem Zeitschritt ihren Eingang an den Ausgang und entsprechen damit im Verhalten einem Flip-Flop. Die erzwungenen Delay-Blöcke modellieren einen einstufigen Synchronisierer; wie in Abschnitt 2.2.3 beschrieben, eignen sich Flip-Flop-Synchronisierer zur Übertragung einzelner Signale über Taktgrenzen.

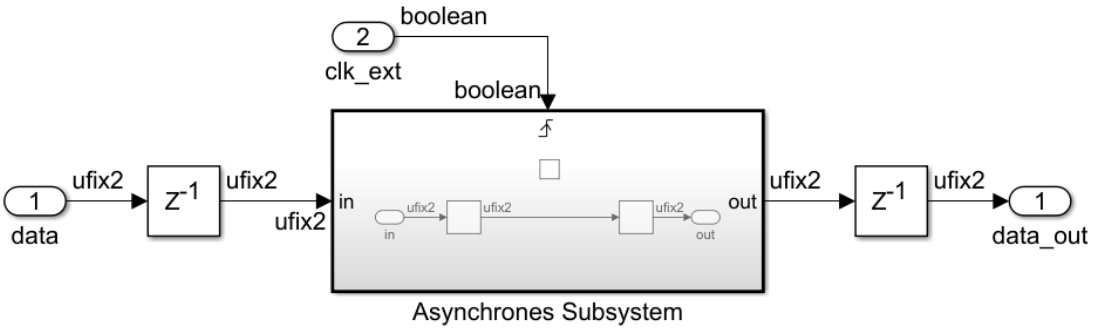


Abbildung 2.9: Beispiel eines Taktübergangs in Simulink

Abbildung 2.9 zeigt beispielhaft ein Subsystem, das über die "Trigger as Clock"-Funktion mit einem externen Takt versorgt wird und ein 2-Bit-Datenwort als Eingangsgröße hat. Das Prinzip dieses Taktübergangs entspricht einer vereinfachten Peripherie-Schnittstelle, die mit einem externen Takt getaktet ist und deren Datenwort mit dem Systemtakt geladen werden muss. Die durch Simulink vorgeschriebenen Synchronisierungs-Delays wurden eingefügt.

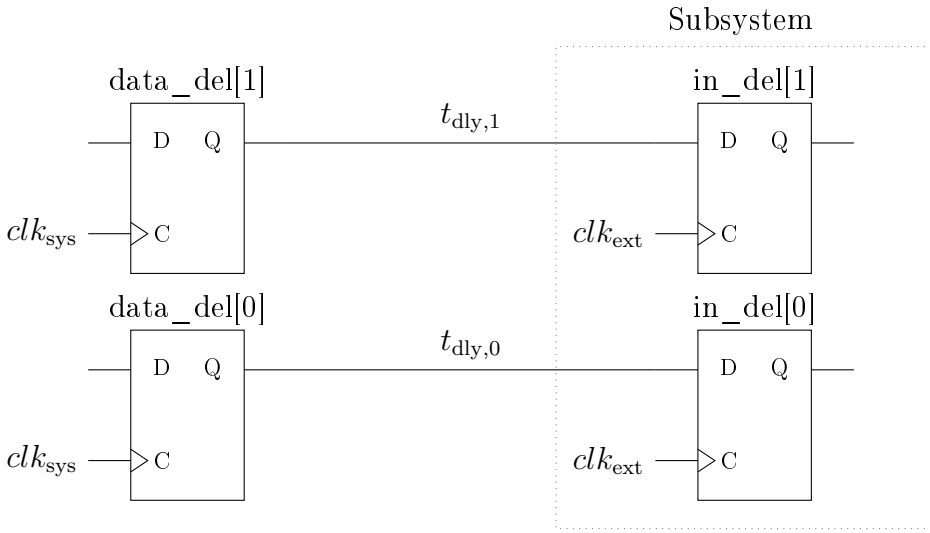


Abbildung 2.10: Ausschnitt der übersetzten Beispielschaltung

Abbildung 2.10 zeigt einen Ausschnitt der für den Artix-7 implementierten Beispielschaltung. In einer Timing-Simulation wird das Eingangs-Datenwort zuerst auf den Wert 1 gesetzt und nach einer Zeitspanne synchron zum Systemtakt auf den Wert 2 geändert.

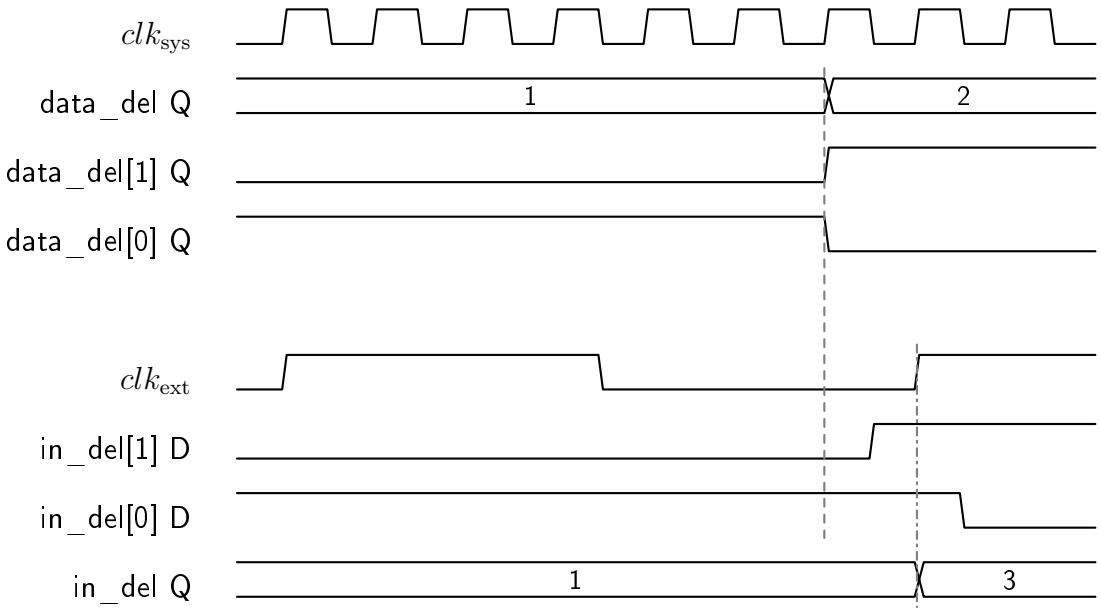


Abbildung 2.11: Vereinfachtes Ergebnis der Timing-Simulation

Abbildung 2.11 zeigt das vereinfachte Ergebnis einer Timing-Simulation dieses Vorgangs.

Durch die prinzipbedingt verschiedenen Verzögerungszeiten $t_{\mathrm{dly},0}$ und $t_{\mathrm{dly},1}$ liegen die geänderten Bits des Datenworts zu unterschiedlichen Zeiten an den empfangenden Flip-Flops an. In diesem Beispiel wurde der externe Takt so gewählt, dass die Eingänge des Eingangsregisters in der Zeitspanne zwischen dem Eintreffen des nullten und des ersten Bits gesampelt werden. Dies führt dazu, dass der Wert 3 empfangen wird, der auf Seite des Systemtakts nie geschrieben wurde.

Das Beispiel verdeutlicht das grundsätzliche Problem der Übertragung mehrerer paralleler Datenbits über eine Taktgrenze und zeigt, dass die Datenintegrität für Datenwörter aus mehreren Bits an einer Schnittstelle zu einem über die "Trigger as Clock"-Funktion getakteten Subsystem auch nicht durch Synchronisationsregister gewährleistet werden kann.

In der Literatur werden Handshaking-Protokolle vorgeschlagen, um das Problem zu lösen [16, S. 236]. Alternativ stellt Xilinx verschiedene asynchrone Speicher zur Verfügung, über die Datenwörter über Taktgrenzen übertragen werden können [21]. Diese haben als IP-Blöcke des FPGA-Herstellers keine direkte Entsprechung in Simulink und können daher nicht simuliert werden.

2.3.2 Adaptives Pipelining und Delay Balancing

In einer digitalen Schaltung können sich im Datenpfad zwischen zwei Flip-Flops ein oder mehrere kombinatorische Logikelemente befinden. [16, S. 22]

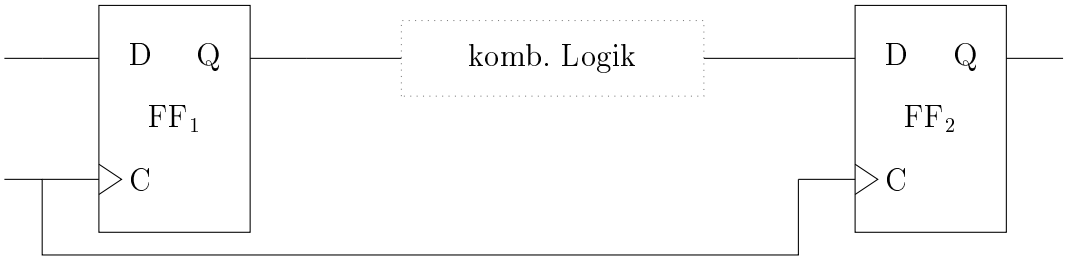


Abbildung 2.12: Kombinatorische Logik mit getakteten Ein- und Ausgängen

Abbildung 2.12 zeigt ein Beispiel einer solchen Schaltung. Je nach Anzahl und Art der Elemente zwischen den Flip-Flops ergibt sich vom Q-Ausgang des ersten

Flip-Flops zum D-Eingang des zweiten Flip-Flops eine Verzögerungszeit t_{dly} aus der Summe der Verzögerungen der Logikelemente und des Verbindungsnetzwerks. [16, S. 23ff]

Je nach der Größe der Verzögerung können die Setz- oder Haltezeiten des FF_2 verletzt werden.

Der Pfad mit der längsten Verzögerungszeit zwischen zwei Flip-Flops wird als kritischer Pfad bezeichnet [4, S. 190].

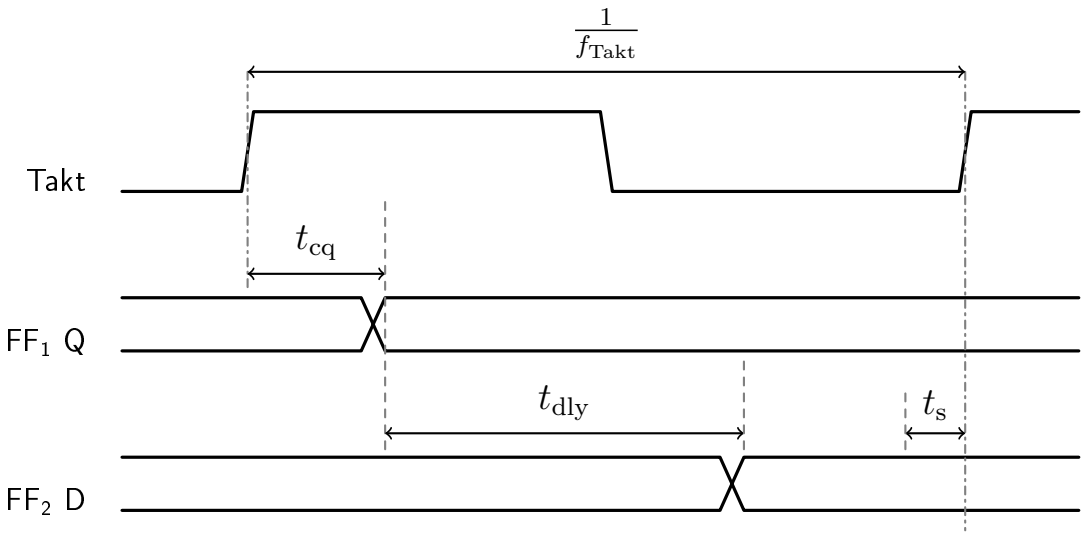


Abbildung 2.13: Auswirkung von Verzögerung im Datenpfad zwischen zwei Flip-Flops

Abbildung 2.13 zeigt eine Konfiguration, bei der die Verzögerungszeit t_{dly} ausreichend klein ist, sodass die Setzzeit des zweiten Flip-Flops bei der gewählten Taktfrequenz f_{Takt} nicht verletzt wird.

Unter der vereinfachenden Annahme, dass die Taktflanken gleichzeitig an beiden Flip-Flops anliegen, gilt für die möglichen Taktfrequenzen [16, S. 25]

$$f_{\mathrm{Takt}} < \frac{1}{t_{\mathrm{cq}} + t_{\mathrm{dly}} + t_{\mathrm{s}}} \tag{2.2}$$

Wenn die Taktfrequenz durch externe Komponenten oder den benötigten Durchsatz vorgegeben ist, ist es notwendig, die Verzögerungszeit t_{dly} zwischen den Flip-Flops so weit zu verringern, dass die Ungleichung 2.2 für die gewünschte Taktfrequenz erfüllt ist.

Dies wird dadurch erreicht, dass die kombinatorische Logik zwischen den Flip-Flops unterteilt wird und weitere Flip-Flops zwischen den Schaltungsteilen eingefügt werden. Dieser Vorgang wird als Pipelining bezeichnet. [4, S. 192]

Die Ungleichung 2.2 muss beim Pipelining nur zwischen zwei aufeinanderfolgenden Stufen der Pipeline erfüllt sein statt zwischen Ein- und Ausgangs-Flip-Flop. Durch jedes eingefügte Flip-Flop verzögert sich der Ausgang der Schaltung gegenüber dem Eingang um eine Taktperiode. Diese Verzögerung wird als Latenz bezeichnet. [4, S. 192ff]

Xilinx empfiehlt Pipelineregister an den die Ein- und Ausgängen der DSP-Blöcke um den möglichen Durchsatz zu erhöhen. [27]

Der HDL-Coder fügt daher vor und nach Multiplikationen (z.B. Product-Block, Gain-Block) automatisch Pipelinestufen ein, die diesen Zweck erfüllen. Diese Funktion wird als adaptives Pipelining bezeichnet und erzeugt eine Latenz im generierten Code, die im ursprünglichen Modell nicht vorhanden ist.

Der HDL-Coder kann beim Einfügen von zusätzlichen Latenzen ein Validierungsmodell generieren, das zyklusgleich zum generierten Code ist und mit dem ursprünglichen Modell verglichen werden kann. [19, S. 530]

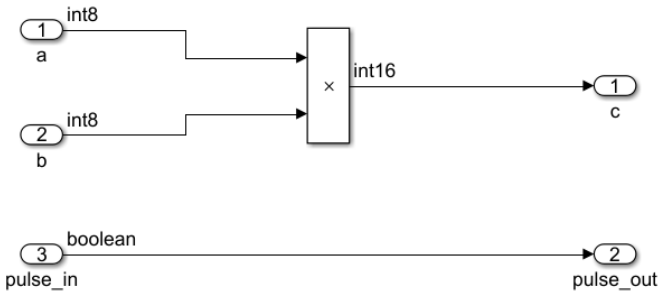


Abbildung 2.14: Multiplikation in Simulink

Abbildung 2.14 zeigt beispielhaft eine Multiplikation $c = a \cdot b$ in Simulink. Neben den Operanden der Multiplikation wird ein Kontrollsignal durch das System geführt.

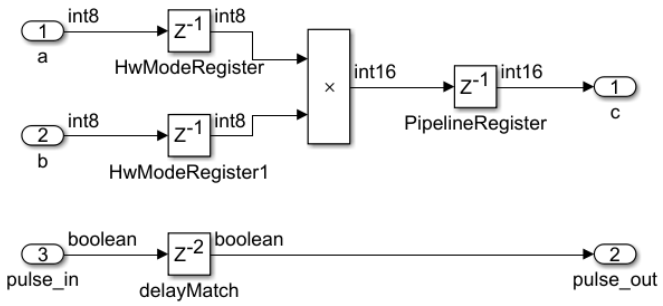


Abbildung 2.15: Validierungsmodell der Multiplikation in Simulink

Abbildung 2.15 zeigt das aus diesem System generierte Validierungsmodell. Neben den Pipelineregistern im Datenpfad wurde auch eine Verzögerung auf das Kontrollsignal angewendet. Dieser Vorgang wird als Delay Balancing bezeichnet. [19, S. 928]

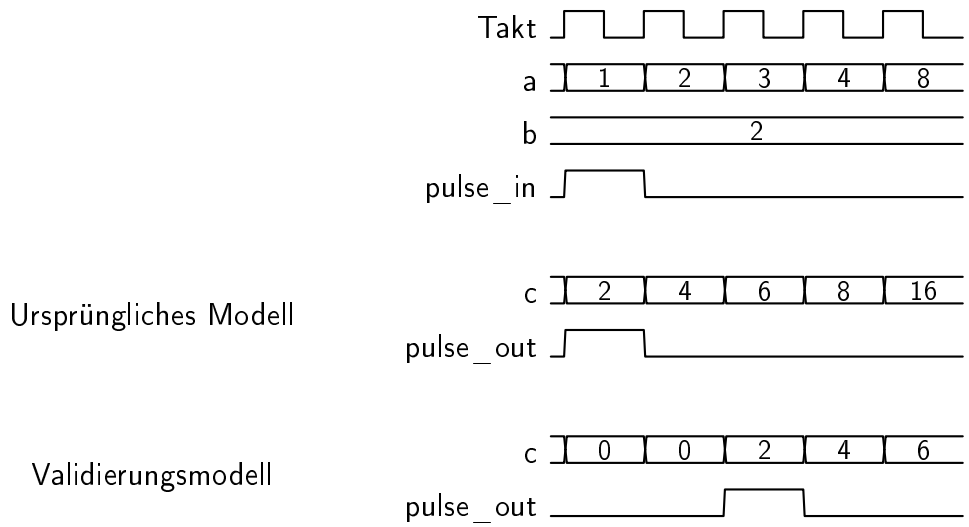


Abbildung 2.16: Vergleich des Verhaltens des ursprünglichen Modells und des Validierungsmodells

Durch das Delay Balancing kann bei der Übergabe von Daten durch mehrere Subsysteme ein paralleles Kontrollsignal geführt werden, das die gleichen Verzögerungen erfährt wie die Datensignale und von den nachfolgenden Subsystemen als Synchronisationssignal verwendet werden kann.

Bei der Verifikation des Systems muss beachtet werden, dass die erzeugte digitale Schaltung nicht zyklusgenau mit dem Modell übereinstimmt. Außerdem zeigt

sich, dass durch das standardmäßig für das gesamte Modell aktivierte Delay Balancing auch die Ausgänge von Subsystemen, die prinzipiell unabhängig von dem Pfad sind, in dem die Pipelineregister eingefügt wurden, verzögert werden. Dies führt zu Problemen, wenn der Ausgang eines bestimmten Subsystems in einer vorgegebenen Anzahl Takte auf einen Eingang reagieren muss, der Ausgang allerdings wegen dem Pipelining eines anderen Subsystems verzögert wird (siehe Abschnitt 4.1.10).

3 Systemkonzipierung

Im Vader-Aufbau wird ein Texas Instruments K2GXEVM-Entwicklungsboard mit einem 66AK2G12 SoC verwendet. Der SoC enthält einen ARM-Core und einen C6000-DSP-Kern[33]. In der vorliegenden Arbeit wird nur mit dem C6000-Kern gearbeitet.

Die im Vorprojekt entwickelte Vader-Platine wird auf einen Erweiterungssteckplatz des Entwicklungsboards aufgesetzt, über den die Kommunikation zwischen DSP und FPGA stattfindet.

Der Versuchsaufbau ist in Abbildung 3.1 gezeigt. Links ist das DSP-Entwicklungsboard zu sehen, auf den die Vader-Platine aufgesteckt ist. An der rechten Seite der Vader-Platine befindet sich der Camera Link-Stecker zur Verbindung mit der Zeilenkamera.

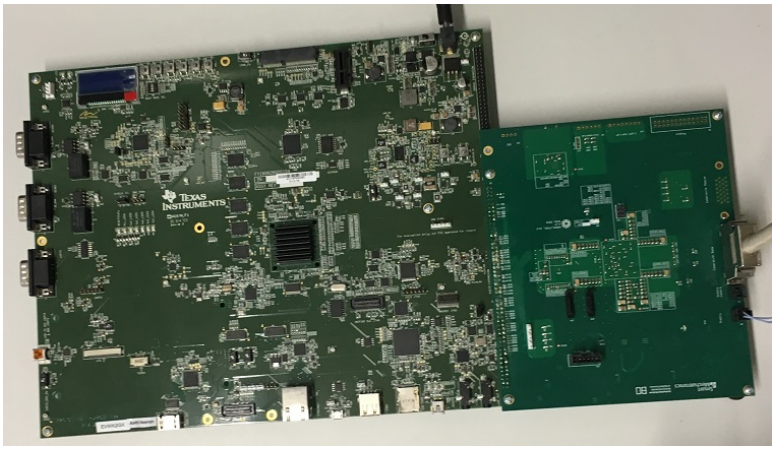


Abbildung 3.1: Aufbau mit DSP-Platine und Vader-Platine

3.1 Anforderungen

Die Anforderungen an die zu entwickelnden Komponenten der Vader-Firmware werden aus dem Lastenheft für das Gesamtsystem abgeleitet. Im Folgenden werden die entwickelten Anforderungen zusammengefasst. [7] [22]

Konfigurationsmodus (Anforderung 00)

Der Bildverarbeitungsalgorithmus auf dem FPGA lässt sich durch den DSP in einen Konfigurationsmodus versetzen, in dem die Parameter der Ortsfrequenzfilterung eingestellt werden können. Dies hat den Vorteil, dass die Filterbank durch Software auf verschiedene Messbedingungen angepasst werden kann, ohne dass eine erneute Synthese der digitalen Schaltung nötig ist.

Zeilenkamera (Anforderungen 01-03)

Es wird eine DALSA Spyder 3-Zeilenkamera verwendet. Die Zeilenbilder werden über ein Camera Link Basic-Interface empfangen und dekodiert. Die Belichtungszeit und Framerate der Kamera sind zur Laufzeit parametrierbar.

Zeilenbildfilter (Anforderung 04)

Um den Gleichanteil der empfangenen Zeilenbilder zu entfernen, werden sie durch ein FIR-Filter mit acht Koeffizienten mit einer Auflösung von je 8 Bit gefiltert. Die Filterkoeffizienten sind statisch und können nach der Synthese nicht mehr verändert werden.

Filterbank (Anforderungen 05-06)

Die Filterbank besteht aus acht unabhängigen Filtermodulen, deren Funktion im Folgenden beschrieben wird:

Wie in Abschnitt 2.1 erläutert, berechnet das Ortsfrequenzfilter aus jedem Kamerabild einen skalaren Wert. Die Bitbreite dieser Werte wird auf 16 Bit festgelegt.

Die Skalierung der Werte ist zur Laufzeit parametrierbar und ein Überlauf bei der Skalierung wird über ein Overflow-Flag an den DSP gemeldet.

Bevor die Filterkoeffizienten mit den Bilddaten verrechnet werden, durchlaufen diese eine parametrierbare Fensterfunktion. Die Fensterung dient dazu, Teile des Zeilenbildes (z.B. Einschwingvorgänge des Zeilenbildfilters oder Artefakte durch die Kameraoptik) von der Bewertung durch das Filter auszuschließen.

Die einzelnen Filter können pro Zeilenbild um eine festgelegte Anzahl Pixel gegen die Bilddaten verschoben werden. Das entspricht dem im Abschnitt 2.1 beschriebenen Verfahren zur Erkennung des Stillstands und der Bewegungsrichtung.

Die Bewertungsfaktoren der Fensterfunktion und die Filterkoeffizienten sind vorzeichenbehaftete 8-Bit-Zahlen und können im Konfigurationsmodus gesetzt werden.

Kantenerkennung (Anforderung 07)

Um Materialkanten in den Kamerabildern zu erkennen, wird ein Kantenerkennungsalgorithmus implementiert, der das Über- bzw. Unterschreiten bestimmter Schwellenwerte erkennt und die entsprechenden Pixelpositionen an den DSP meldet.

Die Kantenerkennung arbeitet nicht mit den durch das Zeilenbildfilter gefilterten Daten. Stattdessen werden die Rohbilddaten für die Kantenerkennung durch ein rekursives Mittelwertfilter, das einen Tiefpasscharakter hat, vorverarbeitet.

Die Schwellenwerte (Thresholds), der Betrachungsbereich (Region Of Interest, ROI) und die Bewertung des Mittelwertfilters werden im Konfigurationsmodus durch den DSP parametriert.

Schnittstellen für die Ausgabe des Wegsignals (Anforderungen 08-09)

Um in Industrieprozesse eingebunden werden zu können, verfügt der Vader über eine Inkrementalgeber- und eine synchron-serielle Ausgabefunktion. Die auszugebenden Weginkremente werden durch das DSP-Modell berechnet und an das FPGA übermittelt.

Datenübertragung zwischen FPGA und DSP (Anforderungen 11-12)

Neben den Ausgaben der Filterbank und der Kantenerkennung überträgt das FPGA in regelmäßigen Abständen ungefilterte Zeilenbilder an den DSP. Dieser kann die Rohbilder unter anderem zur Belichtungssteuerung und Stillstandserkennung verwenden.

Das FPGA empfängt vom DSP die in den restlichen Anforderungen spezifizierten Parameter der einzelnen Module.

Schnittstelle zu einem PC (Anforderung 13)

Über eine USB-Verbindung zu einem PC können die Bewertungskoeffizienten der Ortsfrequenzfilter eingestellt werden, um das System auf verschiedene Messsituationen einstellen zu können. Außerdem können Zeilenbilder der Kamera abgerufen werden. Diese Funktion kann verwendet werden, um die Ausrichtung der Kamera auf das Messobjekt zu prüfen.

Framegrabber

Während der Messung können die Rohbilddaten vom FPGA über eine SATA-Schnittstelle auf ein Speichermedium geschrieben werden. Diese Funktion dient dazu, den Algorithmus auf neue Messsituationen oder Materialien anpassen zu können. Die aufgezeichneten Daten können dazu in einer Simulation untersucht werden.

Die Umsetzung dieser Anforderung ist nicht Teil dieser Arbeit.

3.1.1 Funktionsstruktur

Abbildung 3.2 zeigt eine Funktionsstruktur aus den zentralen Elementen der entwickelten Anforderungen.

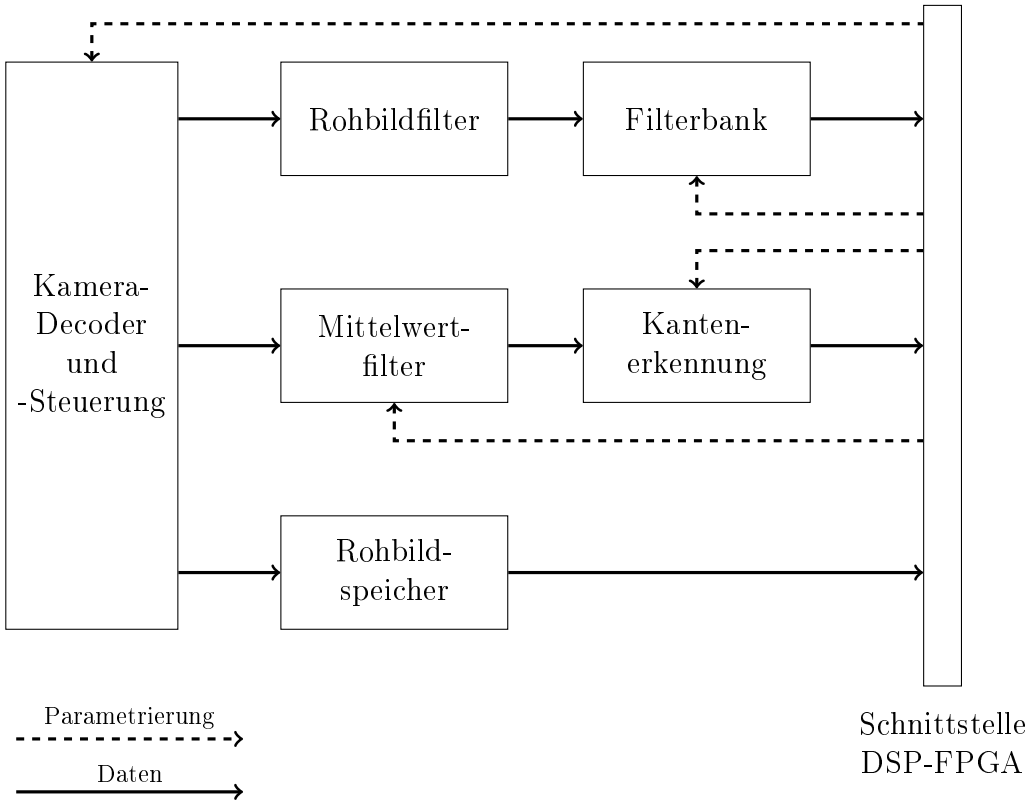


Abbildung 3.2: Funktionsstruktur

3.2 Datenschnittstelle DSP-FPGA

3.2.1 Datenrate

Der Signalprozessor verarbeitet die Ortsfrequenzfilterfunktionen in Blöcken (Batches), die aus den Daten einer bestimmten Anzahl Zeilenbilder bestehen. Mit jeder Batch wird ein vollständiges, ungefiltertes Rohbild der Kamera übertragen. [22]

Aus dem Lastenheft ergeben sich:

- Maximale Framerate der Kamera: $f_{\mathrm{f,max}} = 200\,\mathrm{kHz}$

- Breite der Kamerazeile: $n_{\text{px}} = 1024\,\text{Pixel}$
- Bittiefe der Kamerapixel: $b_{\text{px}} = 16\,\text{Bit}$ (12 Bit für Spyder 3)
- Bitbreite der Einträge der Ortsfrequenzfilterfunktionen: $b_{\text{sf}} = 16\,\text{Bit}$
- Anzahl der Zeilenbilder in einer Batch: $n_{\text{batch,min}} = 128\,\text{Frames}$ und $n_{\text{batch,max}} = 4096\,\text{Frames}$.

  Daraus die maximale Batchfrequenz: $f_{\text{batch,max}} = \frac{f_{\text{f,max}}}{n_{\text{batch,min}}} = 1{,}56\,\text{kHz}$

Die Overflow-Flags für die acht Ortsfrequenzfilter können in $b_{\text{of}} = 8\,\text{Bit}$ dargestellt werden.

Die Kantenerkennung gibt zwei Indizes innerhalb der 1024 Pixel breiten Kamerazeile zurück. Daraus ergibt sich je eine Breite von $b_{\text{ed}} = 10\,\text{Bit}$.

Für die Obergrenzen der Datenübertragungsraten vom FPGA zum DSP ergibt sich aus diesen Werten unter Annahme einer durchgängigen Übertragung:

$$\begin{aligned} C_{\text{sf,max}} &= (8 \cdot b_{\text{sf}} + 1 \cdot b_{\text{of}} + 2 \cdot b_{\text{ed}}) \cdot f_{\text{f,max}} &&= 31{,}2\,\text{Mbit/s} \\ C_{\text{frames,max}} &= n_{\text{px}} \cdot b_{\text{px}} \cdot f_{\text{batch,max}} &&= 25{,}6\,\text{Mbit/s} \end{aligned}$$

3.2.2 McASP

Die McASP-Schnittstelle (Multi Channel Audio Serial Port) bietet die Möglichkeit einer seriellen Datenübertragung über mehrere Kanäle. Sie ist für Audioanwendungen optimiert, kann allerdings auch für nicht-Audio Daten verwendet werden. [23, S. 2763]

Die McASP-Schnittstelle des verwendeten DSPs unterstützt einen Burst-Modus, der für dieses Projekt verwendet wird. Während andere Modi für die unterbrechungsfreie Übertragung von Audio-Streams ausgelegt sind, erlaubt der Burst-Modus die Übermittlung einzelner Datenworte, deren erste Bits jeweils mit dem Frame-Synchronisationssignal (FS) markiert werden. [23, S. 2794]

Das Vader-FPGA ist mit zwei seperaten McASP-Controllern des DSPs verbunden [3]. Eine Auswahl der Ein- und Ausgangssignale jedes McASP-Controllers ist in Tabelle 3.1 aufgeführt.

Signal	Bedeutung
ACLKX	Takt für Sendezweig
ACLKR	Takt für Empfangszweig
AFSX	Frame-Synchronisation für Sendezweig
AFSR	Frame-Synchronisation für Empfangszweig
AXR0 - AXRn-1	Schieberegister Sende- oder Empfangszweig

Tabelle 3.1: Übersicht der für dieses Projekt relevanten MCASP-Signale

Durch die zwei separaten Takt- und Synchronisationssignale kann jeder Controller gleichzeitig unabhängig Daten senden und empfangen. Dazu können die einzelnen AXR Pins jeweils der Sende- oder Empfangsrichtung zugeteilt werden. Die Taktsignale können entweder intern generiert und auf den Taktleitungen ausgegeben werden oder von einer externer Taktquelle gespeist werden. [23, S. 2781f]

Von den zwei verbundenen McASP Controllern (0 und 1) stehen jeweils 13 bzw 9 unabhängige Serialisierer zur Verfügung. [3]

Auf der Vader-Platine befinden sich in den MCASP-Verbindungsleitungen zwischen DSP und FPGA Widerstände (300 Ω) zum Schutz der ICs [3]. Diese Widerstände erzeugen in Kombination mit parasitären Kapazitäten und den Eingangskapazitäten der ICs ein Tiefpassverhalten [24, S. 1167], durch das die Flanken der Signale verschliffen werden.

Mit dieser Konfiguration wurden über die MCASP-Schnittstelle in einem Versuch eine Datenmenge von 4 GB mit einer Taktfrequenz von 10 MHz übertragen, ohne dass ein Übertragungsfehler festgestellt wurde. Diese Beobachtung wurde als Grundlage für die Annahme verwendet, dass die Auftrittswahrscheinlichkeit eines Übertragungsfehlers bei 10 MHz in einem für dieses Projekt akzeptablen Rahmen liegt. Im Kapitel 4 werden verschiedene Maßnahmen beschrieben, Übertragungsfehler zu erkennen.

Die McASP Controller und ihre Datenleitungen werden vor der Implementierungsphase den einzelnen Funktionen zugeteilt. Ziel ist dabei, dass die Ergebnisse der FPGA-Datenverarbeitung nach der Übertragung in einer Form im Speicher des DSP vorliegen, in der sie mit einem Simulinkmodell weiterverarbeitet werden können.

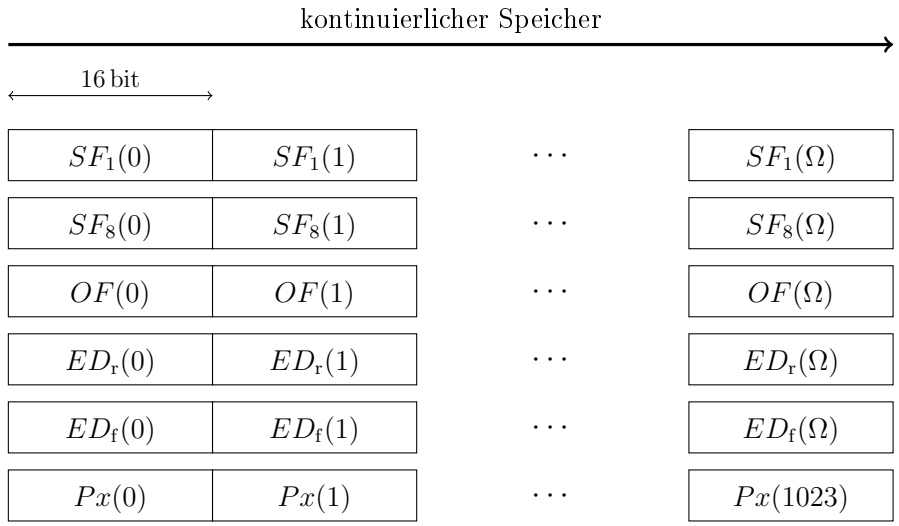


Abbildung 3.3: Daten einer Batch im Speicher des DSP (nicht maßstabsgetreu)

Um dieses Speicherlayout zu erreichen werden im Folgenden mehrere Möglichkeiten erörtert. Weil die Übertragung der Ausgaben der Filterbank und die der Rohbilder unterschiedlich häufig stattfinden (f_frame und f_batch), werden sie bei der Auslegung getrennt betrachtet.

Die Ausgangsdaten der Filterbank werden bei der Auslegung priorisiert weil sie aus mehreren einzelnen Komponenten bestehen und insgesamt eine höhere Datenrate als die Rohbildübertragung benötigen.

Für die Sende- und Empfangszweige der einzelnen McASP-Controller können jeweils eigene Framebreiten (Slots) festgelegt werden [23, S. 2763]. Datenwörter, die parallel übertragen werden sollen, müssen daher in ihrer Länge angepasst werden.

Weil die Filterbank-Daten mit den Einträgen der Ortsfrequenzfilterfunktionen acht 16 Bit breite Wörter enthält, wird die Slotbreite auf 16 Bit festgelegt und sowohl die Overflow Flags (ursprünglich 8 Bit) als auch die Indizes der steigenden und fallenden Flanken (ursprünglich 10 Bit) werden auf 16 Bit aufgefüllt (siehe

Abbildung 3.3). Durch Zwischenspeicherung der Daten wäre es stattdessen zum Beispiel möglich, durch 8·10 Bit = 5·16 Bit mehrere Einträge ohne Füllbits zusammenzufassen. Dies würde zwar einerseits die Nutzdatenrate erhöhen, andererseits wären auf Seite des DSP die Einträge aber nicht mehr einzeln adressierbar [25, S. 32] und müssten durch die CPU wieder getrennt werden. Aus diesem Grund werden diese Möglichkeiten im Folgenden nicht weiter betrachtet.

Die Darstellungen des Übertragungsvorgangs sind aus Gründen der Übersichtlichkeit vereinfacht: Die übertragenen Datenworte werden durchgängig dargestellt; tatsächlich wird während jedem Taktzyklus ein einzelnes Bit des entsprechenden Worts übertragen.

Filterbank: Möglichkeit 1

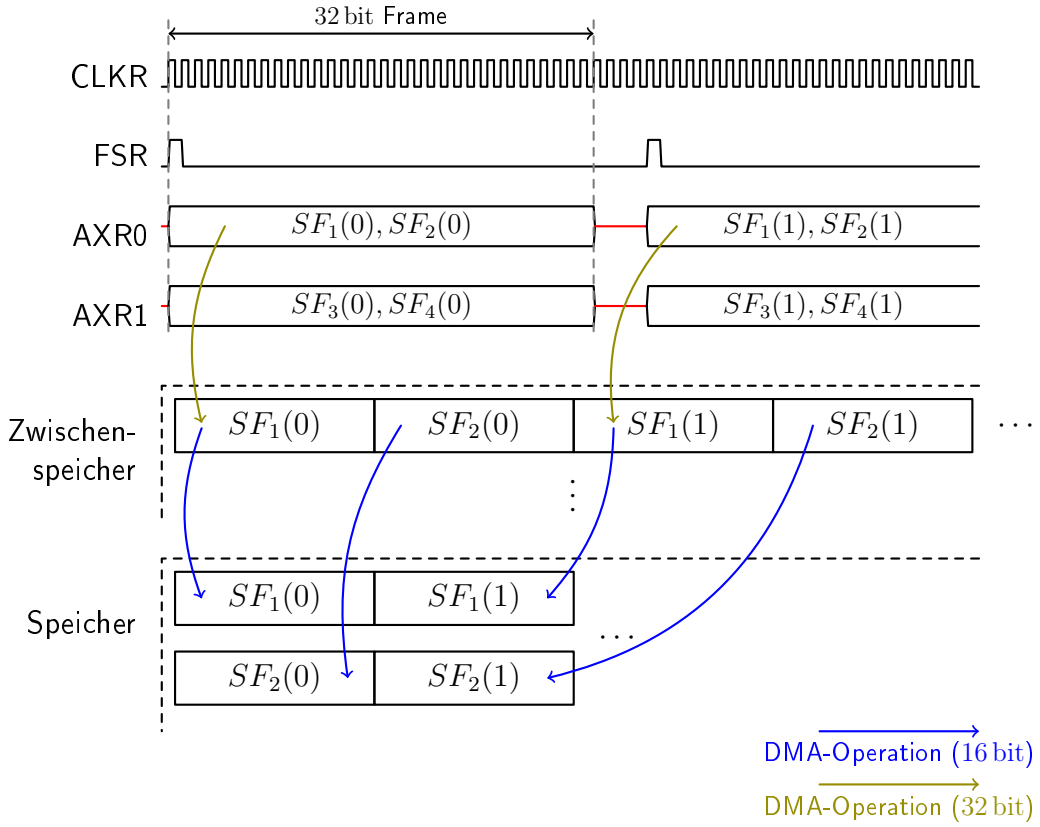


Abbildung 3.4: Übertragung der Filterbank-Daten in den DSP-Speicher: Variante 1

Bei diesem Ansatz werden jeweils zwei 16-Bit-Werte zu einem 32-Bit-Frame zusammengefasst. Dies hat den Vorteil, dass für die elf Elemente der Übertragung nur sechs Serialisierer notwendig sind und die restlichen für die anderen Funktionen verwendet werden können. Die notwendige Taktfrequenz ergibt sich aus der Anzahl Bits (und damit der Anzahl Taktperioden), die für jedes Zeilenbild pro Serialisierer übertragen werden müssen:

$$f_{\mathrm{CLKR,sf}} \geq 32 \cdot f_{\mathrm{f,max}} = 6{,}4\,\mathrm{MHz}$$

Ein Nachteil dieser Variante ist, dass die 32-Bit-Frames, die über den DMA-Controller aus den Empfangspuffern kopiert werden, noch nicht in dem in Abbildung 3.3 gezeigten Schema vorliegen. Dadurch wird ein Zwischenpuffer erforderlich, aus dem die Daten über zusätzliche DMA-Operationen in die gewünschte Form gebracht werden müssen.

Filterbank: Möglichkeit 2

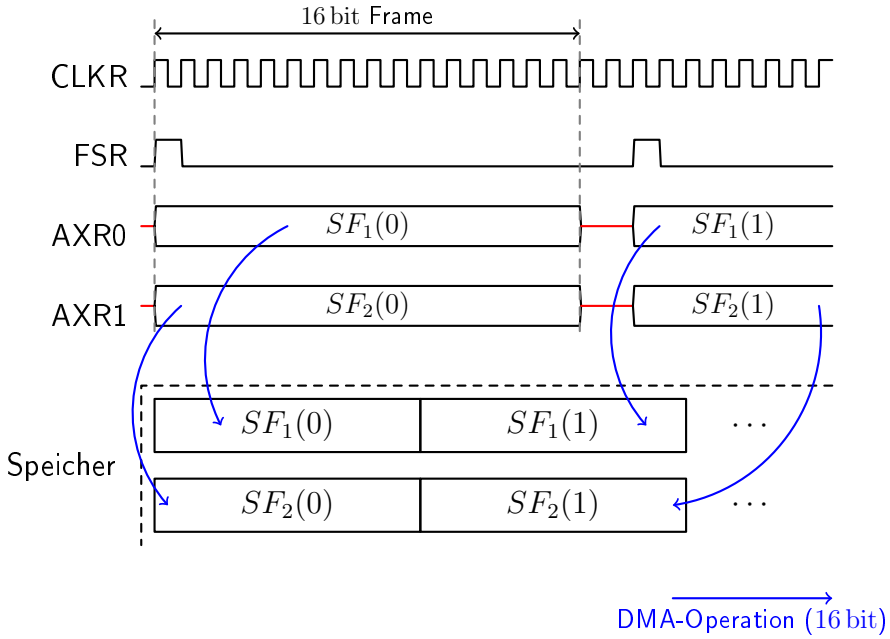


Abbildung 3.5: Übertragung der Filterbank-Daten in den DSP-Speicher: Variante 2

Bei dieser Variante wird für die einzelnen elf Elemente je ein eigener Serialisierer verwendet. Daraus ergibt sich die notwendige Taktfrequenz zu

$$f_{\mathrm{CLKR,sf}} \geq 16 \cdot f_{\mathrm{f,max}} = 3{,}2\,\mathrm{MHz}$$

Der Nachteil dieser Variante ist, dass elf Serialisierer gegenüber den sechs der Variante 1 benötigt werden. Diese Anzahl ist nur am McASP-Controller 0 verfügbar [3].

Aufgrund des Tiefpassverhaltens der Übertragungsstrecke wird angenommen, dass die Fehlerwahrscheinlichkeit durch die niedrigere Taktfrequenz gegenüber Variante 1 geringer ist. Ein weiterer Vorteil ist, dass die Daten direkt aus dem Empfangspuffer in das gewünschte Format gebracht werden.

Neben der in den Abbildungen 3.5 und 3.3 gezeigten Speicherformaten, die im Kontext der Simulink-Codegenerierung als mehrere eindimensionale Vektoren interpretiert werden können, ist es auch möglich, ohne Zwischenspeicherung Simulink-Matrizen im row-major oder column-major Format zu erzeugen [26, S. 623].

Durch das einschrittige Vorgehen wird auf Seiten des DSP Speicher gespart und der DMA-Controller für andere Aufgaben freigehalten.

Aus diesen Gründen wird Variante 2 gegenüber Variante 1 und anderen Möglichkeiten als am geeignetsten betrachtet und angewendet.

Rohbildübertragung

Um die benötigte Taktfrequenz für die Rohbildübertragung zu senken, werden die Daten auf vier Serialisierer verteilt, damit ergibt sich:

$$f_{\mathrm{CLKR,frames}} \geq \frac{16 \cdot 1024}{4} \cdot f_{\mathrm{batch,max}} = 6{,}4\,\mathrm{MHz}$$

Da wegen der Zuweisung von elf Serialisierern für die Übertragung der Filterbankdaten nur noch zwei Serialisierer des McASP-Controllers 0 verfügbar sind, muss die Rohbildübertragung auf den Controller 1 gelegt werden.

Das genaue Vorgehen bei der Übertragung der Rohbilddaten wird im Abschnitt 4.1.7 beschrieben.

Zusammenfassung

Durch die Übertragung der Rohbilddaten und der Filterbank-Daten sind beide Empfangsdatenströme (aus Sicht des DSP) belegt. Die verbleibenden Serialisierer des Controllers 1 werden für die Übertragung der Parametrierung vom DSP an den FPGA verwendet. Tabelle 3.2 fasst die Zuweisungen zusammen.

Controller	Signal	Funktion
0	ACLKR	Filterbank-Daten
	FSR	
	alle AXRi	
1	ACLKR	Rohbilder
	FSR	
	AXR0 - AXR3	
1	ACLKX	Parametrierung
	FSX	
	AXR4 - AXR9	

Tabelle 3.2: Zuweisung der McASP-Signale

3.3 Festlegung der Taktbereiche

In Abschnitt 2.2.3 wurde das Problem der Taktübergänge eingeführt. Durch die Wahl der verschiedenen Takte des Systems vor der Implementierungsphase wird versucht, das Problem so weit wie möglich zu umgehen.

Das Vader-FPGA ist mit verschiedenen Taktsignalen verbunden: [3]

1. 100-MHz-Oszillatorchip auf der Vader-Platine
2. Pixel-Ausgabetakt der Camera Link-Schnittstelle
3. drei MCASP-Taktleitungen (siehe Tabelle 3.2)

4. Takt der synchron-seriellen Schnittstelle

Wie in Abschnitt 3.2.2 beschrieben, kann der DSP die McASP-Taktleitungen sowohl als Ein- als auch als Ausgang verwenden. Um an dieser Stelle keine Taktübergänge zu erzeugen, wird festgelegt, dass die McASP-Takte innerhalb des FPGAs auf Basis des Systemtakts erzeugt werden. Dadurch besteht auf Seiten des FPGAs ein bekanntes Phasenverhältnis zwischen den Takten und das Problem des Taktübergangs verlagert sich auf die Hardware des DSP.

Grundsätzlich wäre es auch umgekehrt möglich, den McASP-Takt durch den DSP generieren zu lassen und ihn als Referenz für den Systemtakt zu verwenden. Allerdings erfordert der MMCM-Block des Artix-7 eine minimale Eingangsfrequenz von 10 MHz für die Frequenzsynthese, sodass mindestens ein McASP-Kanal mit mindestens 10 MHz betrieben werden müsste. Außerdem würde sich durch dieses Vorgehen zwingend ein Taktübergang zum Kameratakt ergeben.

Auf den Takt des SSI kann prinzipbedingt kein Einfluss genommen werden, sodass der Taktübergang an dieser Stelle nicht vermieden werden kann.

Für die Auswahl der Referenz für den Systemtakt werden der 100-MHz-Oszillatorchip und der Kameratakt als Optionen betrachtet. Ein Vorteil des Oszillatortakts ist, dass er immer anliegt, wenn eine Spannungsversorgung besteht während der Kameratakt nur bei angeschlossener und funktionsfähiger Kamera verfügbar ist.

Andererseits würde sich bei Wahl des Oszillators als Referenzquelle ein Taktübergang zwischen der Kamera und dem Vader-Verarbeitungsalgorithmus ergeben, dessen Behandlung, wie in Abschnitt 2.3.1 skizziert, vergleichsweise aufwendig ist.

Aus diesem Grund wird der Pixel-Ausgabetakt der Camera Link-Schnittstelle als Referenz für den Systemtakt gewählt. In dieser Konfiguration kann das Vader-FPGA als logische Erweiterung der Kamera-Hardware betrachtet werden, die nicht eigenständig funktioniert. Der Nachteil dieses Vorgehens ist, dass ohne eine angeschlossene und funktionsfähige Kamera keine Kommunikation (z.B. Abruf einer Status- oder Fehlermeldung) mit dem Vader-FPGA möglich ist. Grundsätzlich lässt sich dieser Nachteil ausgleichen, indem der Kameratakt überwacht wird und bei dessen Ausfall für die Bereitstellung von Statusmel-

dungen auf den Oszillatortakt gewechselt wird [9, S. 38]. Wegen der erhöhten Komplexität wird dieser Ansatz in der vorliegenden Arbeit nicht weiter verfolgt.

Abbildung 3.6 zeigt eine Zusammenfassung der geplanten Takte. Die Pfeile geben jeweils die Richtung eines Takts von seiner Quelle zu seiner Senke an.

Die grau hinterlegten Blöcke repräsentieren die Multiplikation der Taktfrequenz mit einem Faktor N. Faktoren $N > 1$ können dabei durch PLLs oder MMCMs realisiert werden [9]. Faktoren $N < 1$ können durch PLLs, MMCMs oder - unter bestimmten Voraussetzungen - durch Taktteiler auf Basis von Flip-Flops umgesetzt werden [13, S. 184].

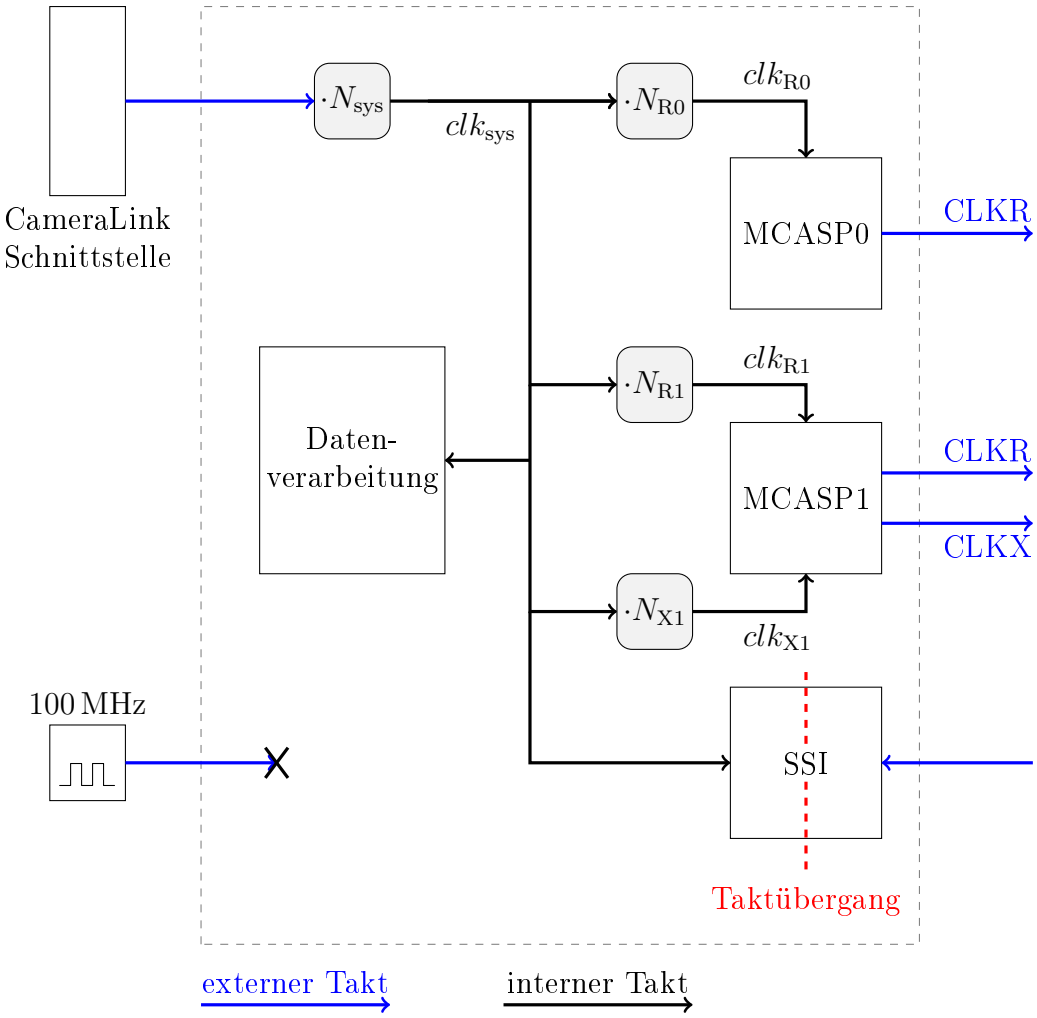


Abbildung 3.6: Takte des VADER-FPGA

4 Implementierung

Das folgende Kapitel beschreibt die Umsetzung der ermittelten Anforderungen. Außerdem wird die Verifikation von Teilsystemen beschrieben, für die eine Verifikation auf Modulebene als sinnvoll erachtet wurde.

4.1 FPGA

Bei der Modellierung der FPGA-Logik wird neben dem Haupt-Simulinkmodell eine Bibliothek erstellt (Vader_lib), in der mehrfach verwendete Blöcke und verschiedene Implementierungsvarianten gespeichert werden.

Zur Synthese des erzeugten HDL-Codes wird die Software Vivado 2018.1 des FPGA-Herstellers Xilinx verwendet. Die Definition von Randbedingungen erfolgt in einer XDC-Datei (Xilinx Design Constraints), die notwendig ist, um den generierten HDL-Code zu synthetisieren.

4.1.1 Parallelisierung und Taktfrequenz

Die größte Zahl, die sich in einem Zweierkomplement der Bitbreite n darstellen lässt, ist $2^{n-1}-1$, die betragsgrößte ist -2^{n-1} [5, S. 7].

Rechnung 4.1 zeigt, dass das Ergebnis einer Multiplikation zweier Zweierkomplementzahlen mit den Breiten n und m die Breite $n+m$ hat:

$$\begin{aligned}(-2^{n-1}) \cdot (-2^{m-1}) = 2^{n-1} \cdot 2^{m-1} = 2^{(n-1)+(m-1)} = 2^{n+m-2} \\ 2^{(n+m-1)-1} - 1 < 2^{n+m-2} < 2^{(n+m)-1} - 1\end{aligned} \tag{4.1}$$

Auf dieser Grundlage werden die Bitbreiten der Signalverarbeitungskette für die Ortsfrequenzfilterung - beginnend bei einem 12-Bit-Pixel der Spyder 3-Kamera -

ermittelt. Die Verarbeitung führt durch das Rohbildfilter, die Fensterung und die anschließende Ortsfrequenzfilterung, jeweils mit Koeffizienten der Breite 8 Bit.

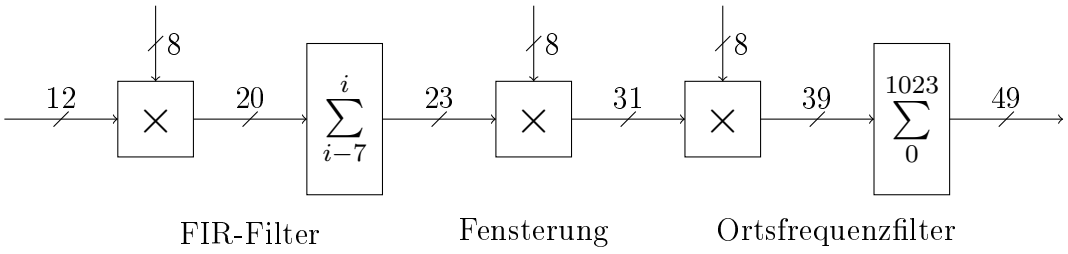


Abbildung 4.1: Bitbreiten der Datenverarbeitung

Die Summierung von acht Werten der Breite 20 Bit im FIR-Filter entspricht bezogen auf die Bitbreite des Ergebnisses einer Multiplikation mit $8 = 2^3$, sodass der Ausgang des FIR-Filters eine Bitbreite von 23 Bit besitzt. Das gleiche Prinzip wird auf den Ausgang des Ortsfrequenzfilters angewendet: Das Aufsummieren von 1024 Werten der Breite 39 Bit ergibt einen Wert der Breite 49 Bit.

Wie in Abschnitt 2.2 beschrieben, verfügt der Artix-7 über 70 Multiplizierer mit den Eingangsbreiten 18 und 25 Bit. Da die Eingangsbitbreiten der Ortsfrequenzfilterung nach Abbildung 4.1 31 Bit und 8 Bit sind, lässt sich diese Funktion nicht mehr mit einem Artix-7 Multiplizierer durchführen.

Mit $|x|_i$ der i-ten Stelle des Betrags von x in Binärdarstellung, beginnend mit der Einerstelle bei $i = 0$, wird der Betrag von x in eine Summe zerlegt:

$$|x| = \sum_{i=0}^{30}(|x|_i \cdot 2^i) = \sum_{i=0}^{16}(|x|_i \cdot 2^i) + \sum_{i=17}^{30}(|x|_i \cdot 2^i), \quad |x|_i \in \{0, 1\}$$

Durch das Assoziativgesetz lässt sich anschließend die Multiplikation der 31 Bit breiten Zweierkomplementzahl zerlegen:

$$\begin{aligned} y = a \cdot x &= \text{sign}(x) \cdot a \cdot |x| \\ &= \text{sign}(x) \cdot \left(a \cdot \sum_{i=0}^{16}(|x|_i \cdot 2^i) + a \cdot \sum_{i=17}^{30}(|x|_i \cdot 2^i) \right) \end{aligned} \tag{4.2}$$

Zu diesem Zweck beinhalten die DSP-Blöcke des Artix-7 17-Bit-Schiebeoperationen, durch die mehrere Teilmultiplikationen verknüpft wer-

den können [27]. Rechnung 4.2 zeigt, dass die gewünschte Operation in zwei Teilmultiplikationen aufgeteilt werden kann.

Die Berechnung der Fensterung und die Ortsfrequenzfilterung benötigen auf diese Weise für die Verarbeitung eines Pixels zusammen drei DSP-Blöcke. Weil die Filterbank acht dieser Operationen parallel ausführen muss, ergeben sich insgesamt 24 DSP-Blöcke. Unter der Annahme, dass das FIR-Filter mit acht Koeffizienten mit acht Multiplikatoren implementiert wird, erhöht sich die Anzahl auf 32 DSP-Blöcke.

Das verwendete Kameramodell verfügt über zwei Taps (Abgriffe am Schieberegister), über die mit der halben Pixelfrequenz jeweils zwei Pixel parallel ausgegeben werden. [28]

Weil durch die Verarbeitung eines Pixels nur 32 Multiplikatoren belegt sind, wird festgelegt, dass zwei Pixel parallel verarbeitet werden, woraus sich 64 verwendete DSP-Blöcke ergeben. Durch dieses Vorgehen wird der Systemtakt möglichst niedrig gehalten und die Multiplikatoren des verwendeten FPGAs besser ausgenutzt. Der Faktor zwischen Pixel-Ausgabetakt und Systemtakt ergibt sich zu $N_{\text{sys}} = 1$ (Abbildung 3.6).

Sollen statt den 12-Bit-Pixeln der Spyder 3-Kamera Pixel mit der maximal spezifizierten Breite von 16 Bit verarbeitet werden, erhöhen sich die Bitbreiten um vier Bit. Das führt dazu, dass die Fensterung zu einer 27x8 Bit Operation wird, die nicht mehr mit einem einzelnen Artix-7 Multiplizierer berechnet werden kann. Abschnitt 4.1.3 beschreibt eine Möglichkeit zur Reduzierung der Ausgangsbitbreite des FIR-Filters unter bestimmten Bedingungen.

4.1.2 Kameradecoder und -steuerung

Die Camera Link-Spezifikation sieht verschiedene Kommunikationskanäle zwischen Kamera und der Auswertungsschaltung vor [29]. Die Signale sind als differenzielle LVDS-Paare ausgeführt, die durch drei ICs auf der Vader-Platine zwischen den LVDS- und 3,3 V-Logiksignalen übersetzen [3].

Die Spyder 3-Kamera verwendet die folgenden Camera Link Signale: [28]

- Datenkanal aus einem Line-Valid-Signal (LVAL), das mit einem logischen High-Zustand gültige Pixeldaten anzeigt und drei 8 Bit breiten Datenbussen (PORTA, PORTB, PORTC), auf die die Pixelinformation verteilt wird
- Kontrollkanal aus vier Signalen, die zur Steuerung der Framerate und Belichtungszeit verwendet werden
- Kommunikationskanal, bestehend aus zwei seriellen Schnittstellen, je einen in Sende- und Empfangsrichtung

Für dieses Projekt wird die Spyder 3-Kamera im "Smart Exsync"-Modus betrieben, in dem Framerate und Belichtungszeit über ein einzelnes Logiksignal vorgegeben werden. Die Periode dieses Signals (EXSYNC) bestimmt die Framerate, die Dauer des High-Zustands bestimmt die Belichtungszeit. [28]

Für die Steuerung dieser Parameter wird ein 16-Bit-Counter modelliert, der mit zwei Werten parametriert wird. Der erste Parameter gibt die Periode des Counters (und damit des EXSYNC-Signals) in Systemtaktzyklen an, der zweite Parameter gibt die Anzahl der Systemtaktzyklen an, während derer das EXSYNC-Signal auf einem High-Pegel verweilt.

Der Zugriff auf die serielle Schnittstelle der Kamera wurde als im Betrieb nicht notwendig erachtet. Sie dient dem Abruf von Statusmeldungen und der Vornahme von Einstellungen wie dem An- und Abschalten des Testbildmodus. Die seriellen Schnittstellen der Kamera werden nicht in Simulink verarbeitet sondern in der FPGA-Harness an zwei extern abgreifbare Pins weitergeleitet, sodass der Entwicklungsrechner mit einem seriellen Adapter über das Vader-FPGA mit der Kamera kommunizieren kann.

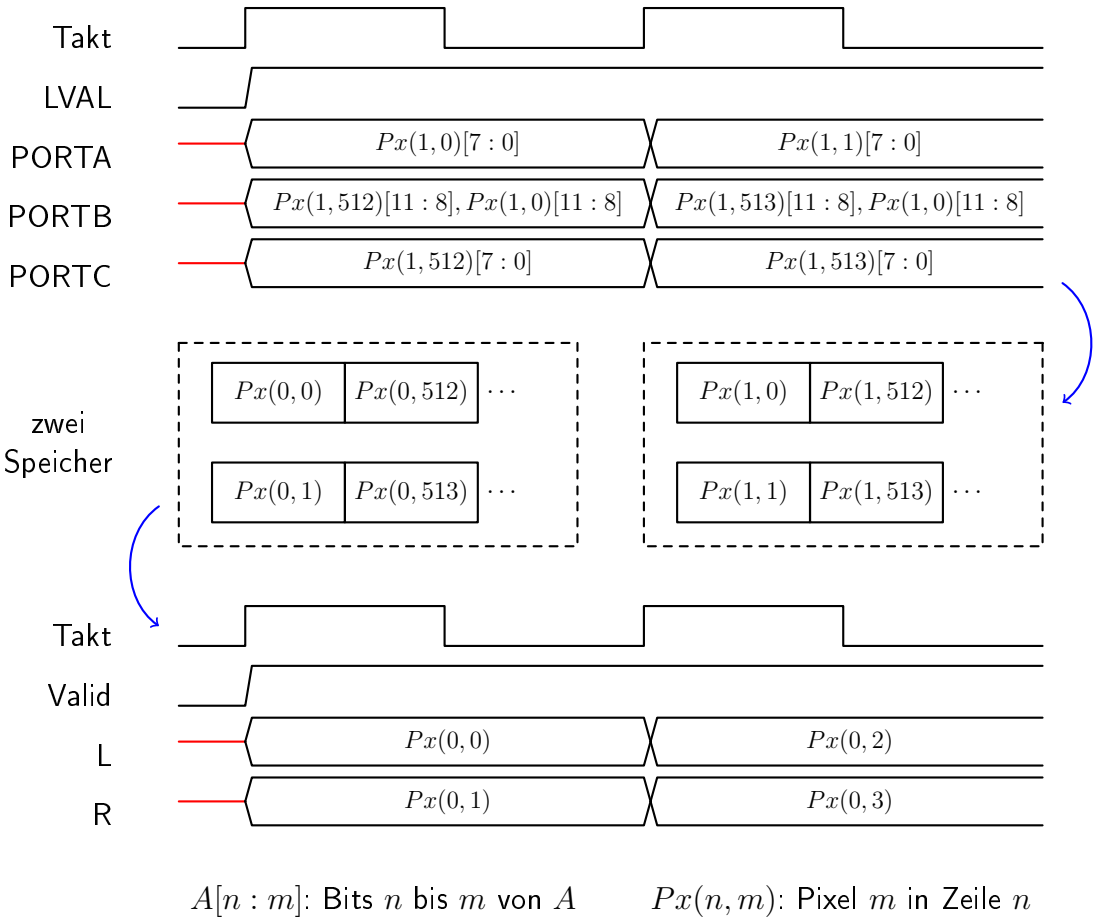


Abbildung 4.2: Vereinfachte Darstellung der Pixeldekodierung

Abbildung 4.2 stellt den Dekodierungsvorgang der Pixelinformation dar. Der Pixel-Ausgabetakt - und damit auch der Systemtakt - ist 40 MHz [28].

Die zwei Abgriffe an der Kamerazeile (Tap 1 und Tap 2) geben jeweils gleichzeitig ein Pixel aus der linken und rechten Hälfte des Bildes aus. Die Pixelinformation liegt synchron zur steigenden Flanke des Ausgabetakts an den drei Datenbussen an, PORTA und PORTC enthalten dabei jeweils die unteren acht Bit der Pixel von Tap 1 bzw Tap 2 und PORTB enthält die oberen vier Bit der beiden Pixel.

Die Pixeldaten werden aus den einzelnen Komponenten zusammengesetzt und in einem von zwei Zwischenspeichern abgelegt. Parallel dazu werden die Inhalte des jeweils anderen Zwischenspeichers, der das vorhergehende Zeilenbild enthält, als zwei benachbarte Pixel ausgegeben. L und R bezeichnen jeweils den linken

bzw rechten Pixel, der Pixelindex 0 wird als der linke Rand des Bildes definiert, Pixelindex 1023 als der rechte Rand.

Nach jedem Zeilenbild werden die Zwischenspeicher vertauscht, sodass immer ein Speicher beschrieben wird, während die Daten des anderen Speichers gelesen werden. Unter der Vernachlässigung der Latenz der Zwischenspeicher erzeugt dieses Vorgehen eine Latenz von einer Frameperiode.

Randbedingungen

Die Zuweisungen der einzelnen Camera Link-Signale zu den FPGA-IO-Pins werden dem Vader-Schaltplan, dem Datenblatt der Schnittstellen-ICs und der Camera Link-Spezifikation entnommen.

Als Timing-Randbedingungen werden für die Eingangspins des FPGA die minimale und maximale Verzögerung der Datenleitungen gegenüber der Taktleitung angegeben (Input Delays). [30, S. 158]

Die Pegelwandlung des Camera Link-Datenkanals wird auf der Vader-Platine durch einen DS90CR288-IC realisiert [3]. Angegeben sind in dessen Datenblatt die Setz- und Haltezeiten der Datensignale relativ zum Taktsignal: [31]

- RxOUT Setup to RxCLK OUT Min 3,5 ns (t_s)
- RxOUT Hold to RxCLK OUT Min 3,5 ns (t_h)

Weil sowohl der Takt als auch die Daten bei dieser Schnittstelle vom DS90-IC als Quelle zum FPGA als Senke verlaufen, wird angenommen, dass sie in etwa den gleichen Weg zurücklegen und zwischen den Verzögerungen der Takt- und Datenleitungen eine vernachlässigbare Differenz besteht.

Auf dieser Grundlage werden in Abbildung 4.3 die gegebenen Setz und Haltezeiten sowie die gesuchten Input Delays eingezeichnet. Der weiße Bereich in der Datenzeile kennzeichnet garantiert anliegende Daten, der graue Bereich einen unbekannten Zustand.

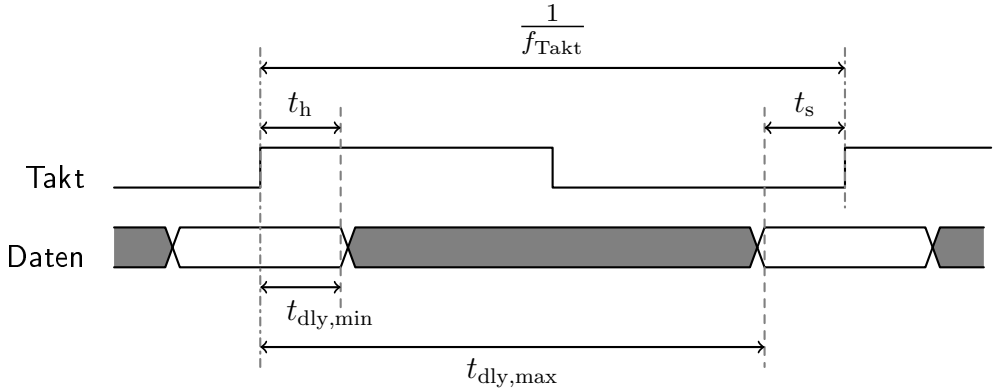


Abbildung 4.3: Input Delays der Camera Link-Schnittstelle (nicht maßstabsgetreu)

Aus Abbildung 4.3 ergibt sich unter Vernachlässigung der Anstiegszeit der Daten für das kleinste Input Delay:

$$t_{\text{dly,min}} = t_{\text{h}} = 3{,}5\,\text{ns}$$

Für das größte Input Delay ergibt sich:

$$t_{\text{dly,max}} = \frac{1}{f_{\text{Takt}}} - t_{\text{s}} = 25\,\text{ns} - 3{,}5\,\text{ns} = 21{,}5\,\text{ns}$$

Verifikation

Um das Kamera-Subsystem auf Modellebene zu verifizieren, wird zunächst ein Modell der Spyder 3-Kamera als Gegenstück entwickelt. Die Camera Link-Datensignale (PORTA, PORTB, PORTC und LVAL) werden dabei über ein Matlab-Skript aus einer Bilddatei generiert. Die zwei Modelle werden in einer Test-Harness verknüpft und die dekodierten Bilddaten mit der ursprünglichen Bilddatei verglichen.

Dieser Ansatz dient nur als erste Plausibilitätsprüfung, da davon ausgegangen werden muss, dass eventuelle Fehler die bei der Entwicklung des Dekoders gemacht werden, bei der Entwicklung des Kameramodells erneut gemacht werden.

In diesem Fall wäre die Verifikation auf Modellebene erfolgreich, das Kameramodell würde aber die Spyder 3-Hardware nicht korrekt abbilden und das Dekodieren von Pixelströmen einer realen Spyder 3-Kamera schlüge auf der Hardware fehl.

Aus diesem Grund wird aus dem Modell des Kameradekoders HDL-Code generiert und dieser für den verwendeten Artix-7 übersetzt. Die Spyder 3-Kamera wird angeschlossen und in den Testbildmodus versetzt. Die dekodierten Bilddaten werden über eine UART-Schnittstelle an einen Rechner übertragen und dort mit dem erwarteten Muster des Testbildes aus dem Spyder 3-Datenblatt verglichen.

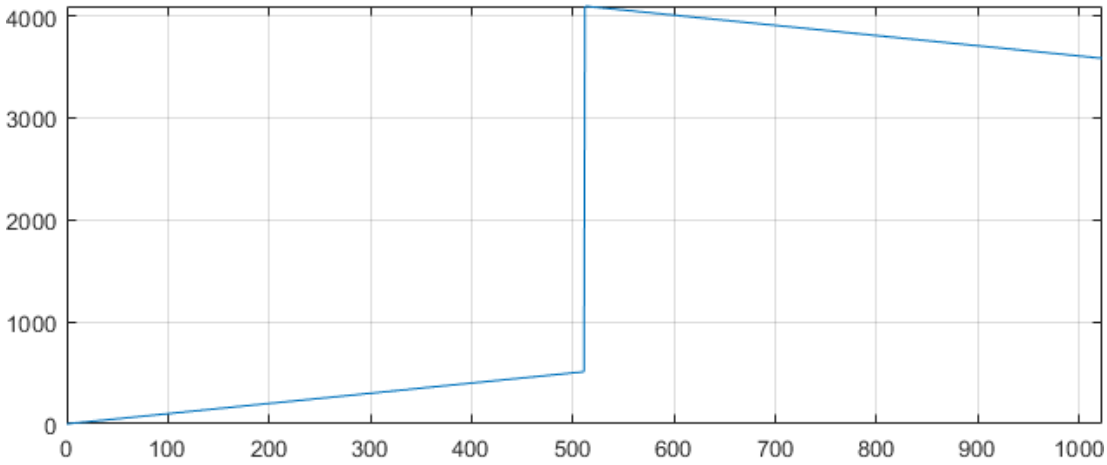


Abbildung 4.4: Dekodiertes Testbild

4.1.3 Rohbildfilter

Das Rohbildfilter verarbeitet die dekodierten Pixeldaten für die anschließende Verwendung in der Filterbank. Durch den Auftraggeber wurde die Umsetzung des Rohbildfilters als FIR-Filter mit acht Koeffizienten vorgegeben.

Die Ausgang eines solchen Filters y mit den Koeffizienten $f[0], \cdots, f[7]$ und den Eingangswerten x ist: [32, S. 80]

$$y[n] = \sum_{k=0}^{7} f[k]x[n-k] \tag{4.3}$$

Der Ausgang wird also aus der Summe der Produkte der Filterkoeffizienten mit den acht letzten Eingangswerten gebildet.

Da der Kamera-Dekoder jeweils zwei Pixel gleichzeitig ausgibt (L und R), muss das Rohbildfilter auch mit jedem Takt zwei Eingangswerte akzeptieren und zwei Ausgangswerte ausgeben. Der Dekoder gibt die Pixel beginnend mit dem Pixelindex 0 aus, das linke Pixel wurde dabei als das Pixel mit dem kleineren Pixelindex definiert und kommt daher in der Zeitreihe vor dem rechten Pixel. Die Werte der Eingangszeitreihe x werden über die Werte der Zeitreihen L_{in} und R_{in} ausgedrückt:

$$\begin{aligned} \cdots, x[-3] = L_{in}[-1], \qquad & x[-2] = R_{in}[-1], \\ x[-1] = L_{in}[0], \qquad & x[0] = R_{in}[0] \end{aligned}$$

Allgemein wird $x[n]$ mit geraden n jeweils ein rechter Pixel und $x[n]$ mit ungeraden n jeweils ein linker Pixel zugewiesen:

$$x[n] = \begin{cases} L_{in}[\frac{n+1}{2}] & \text{für } n \text{ ungerade} \\ R_{in}[\frac{n}{2}] & \text{für } n \text{ gerade} \end{cases} \tag{4.4}$$

Umgekehrt für die Ausgangswerte:

$$\begin{aligned} L_{out}[n] &= y[2n-1] \\ R_{out}[n] &= y[2n] \end{aligned}$$

Durch Einsetzen in Gleichung 4.3 folgt für die Ausgangswerte:

$$\begin{aligned} L_{out}[n] = y[2n-1] \quad &= \sum_{k=0}^{7} f[k]x[2n-1-k] \\ R_{out}[n] = y[2n] \quad &= \sum_{k=0}^{7} f[k]x[2n-k] \end{aligned}$$

Über die Beziehung 4.4 werden für $x[j]$ die entsprechenden linken und rechten Eingangspixel eingesetzt:

$$
\begin{aligned}
L_{out}[n] &= \sum_{k=0}^{3} f[2k] L_{in}[n-k] + \sum_{k=0}^{3} f[2k+1] R_{in}[n-k-1] \\
R_{out}[n] &= \sum_{k=0}^{3} f[2k] R_{in}[n-k] + \sum_{k=0}^{3} f[2k+1] L_{in}[n-k]
\end{aligned}
\tag{4.5}
$$

Die sich ergebenden Rechenvorschriften lauten in Worten:

Für das linke Ausgangspixel wird die Summe aus dem Produkt der Pixel $L_{in}[n]$ bis $L_{in}[n-3]$ mit den Koeffizienten mit geraden Indizes (0, 2, 4, 6) und dem Produkt der Pixel $R_{in}[n-1]$ bis $R_{in}[n-4]$ mit den Koeffizienten mit ungeraden Indizes (1, 3, 5, 7) gebildet.

Für das rechte Ausgangspixel wird die Summe aus dem Produkt der Pixel $R_{in}[n]$ bis $R_{in}[n-3]$ mit den Koeffizienten mit geraden Indizes und dem Produkt der Pixel $L_{in}[n]$ bis $L_{in}[n-3]$ mit den Koeffizienten mit ungeraden Indizes gebildet.

Die Gleichungen 4.5 werden in ein Simulink-Blockschaltbild übersetzt, dabei wird der Zugriff auf die letzten Pixelwerte über eine Kette von vier bzw drei Delay-Blöcken erreicht.

Wie in Abschnitt 2.3.2 erläutert, werden durch den HDL-Coder für die Synthese Pipeline-Register um die Multiplikatoren eingesetzt, sodass die Ausgangswerte sich gegenüber den Eingängen verzögern. Aus diesem Grund wird nach dem Schema aus Abbildung 2.14 ein line_valid-Signal durch den Block geführt, das durch das Delay Balancing in der Verzögerung an die Datenleitungen angepasst wird.

Einschwingverhalten

Die Delay-Blöcke werden nach jeder verarbeiteten Zeile auf den Initialwert 0 zurückgesetzt, damit die ersten Ausgangswerte des Filters in einer Zeile nicht von den letzten Pixeln der vorherigen Zeile, die noch in den Delay-Blöcken gespeichert wären, abhängig sind. Dadurch ergibt sich ein Einschwingvorgang wenn die ersten Pixel der Zeile in das Filter laufen und bewertet werden, die hinteren Koeffizienten allerdings noch keinen Einfluss haben weil sie mit den Delay-Werten 0 bewertet werden.

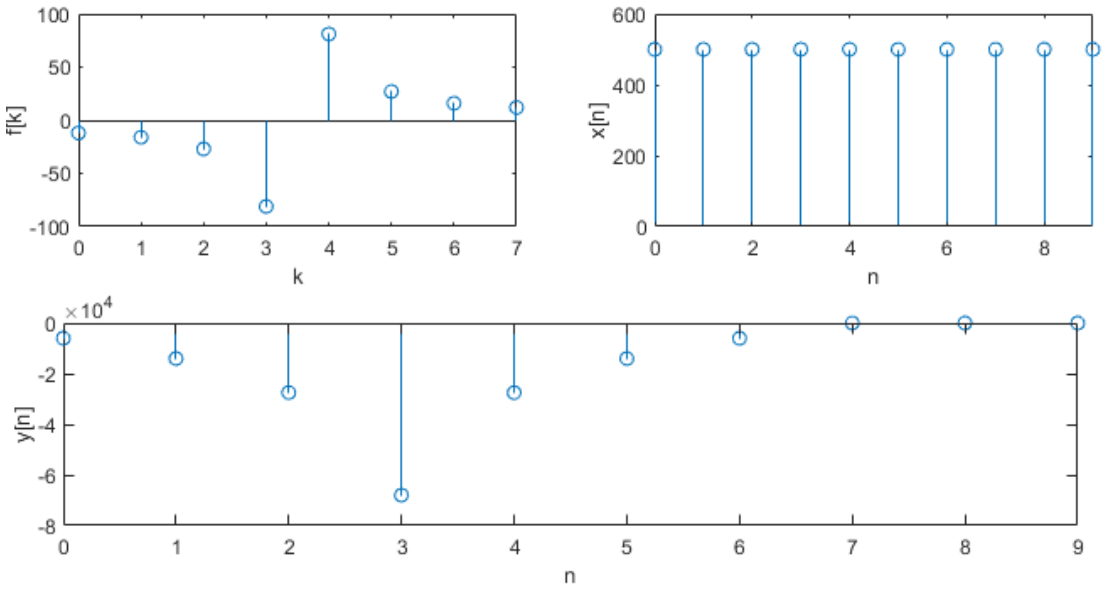


Abbildung 4.5: Einschwingverhalten des FIR-Filters

Abbildung 4.5 zeigt den Einschwingvorgang des verwendeten FIR-Filters bei einem konstanten Eingangssignal mit $x[n] = 500$. Da die ersten Filterkoeffizienten negativ sind und die Eingangs-Pixelwerte nicht negativ sein können, bildet sich das gezeigte Einschwingverhalten in Negativrichtung.

Der Einschwingvorgang, der sich unter der Annahme, dass die ersten Eingangspixel nicht null sind, immer in den ersten Einträgen der Ausgangsfunktion befindet, stellt aus Sicht des Ortsfrequenzfilters ein stillstehendes Oberflächenmerkmal dar und führt daher zu einer ungewollten Komponente mit Geschwindigkeit 0 in der Ortsfrequenzfilterfunktion. Diese Geschwindigkeitskomponente hat in einem stillstehenden Filter die Frequenz 0 und äußert sich in einem Beitrag zum Gleichanteil der Ortsfrequenzfilterfunktion. In einem bewegten Filter wird die Frequenzkomponente mit der Frequenz der Filterbewegung gemischt.

Um den Einfluss des Einschwingvorgangs auf die Ortsfrequenzfilterfunktion zu verhindern, kann die Fensterfunktion verwendet werden, um die ersten Einträge des gefilterten Pixelstroms in jeder Zeile mit dem Faktor 0 zu bewerten.

Ausgangsbitbreite

In Abschnitt 4.1.1 wurde die Ausgangsbitbreite des Rohbildfilters für Eingangswerte der Breite 12 Bit allgemein über die maximale Breite einer Summe von acht Zweierkomplementzahlen der Breite 20 Bit mit 23 Bit angegeben. Für die maximale spezifizierte Eingangsbitbreite von 16 Bit ergeben sich nach dem selben Prinzip Ausgangswerte der Breite 27 Bit, die in der anschließenden Fensterung nicht mehr mit voller Genauigkeit und einem einzelnen Multiplizierer des Artix-7 verarbeitet werden können.

Da die Filterkoeffizienten statisch sind und im Betrieb nicht geändert werden können, kann die Bitbreite auf Basis der tatsächlich erreichbaren größten Zahl festgelegt werden. Weil die Pixel-Eingangswerte vorzeichenlos sind, ergeben sich die betragsgrößten Ausgangswerte wenn alle positiven Koeffizienten mit dem größtmöglichen Pixel-Eingangswert und die negativen Koeffizienten mit null multipliziert werden oder wenn umgekehrt alle negativen Koeffizienten mit dem größtmöglichen Eingangswert und die positiven Koeffizienten mit null multipliziert werden.

Die für dieses Projekt vorgegebenen Koeffizienten lauten skaliert auf 8-Bit vorzeichenbehaftete Ganzzahlen: $-12, -16, -27, -81, 81, 27, 16, 12$

$$\begin{aligned} out_{\max} &= (2^{16}-1)\cdot(81+27+16+12) && < 2^{24}-1 \\ out_{\min} &= (2^{16}-1)\cdot\big(-81+(-27)+(-16)+(-12)\big) && > -2^{24} \end{aligned} \tag{4.6}$$

Die Rechnungen 4.6 zeigen, dass der Zahlenbereich der Ausgangswerte unter diesen Bedingungen in einer 25-Bit-Zweierkomplementzahl dargestellt werden kann. Die Ausgangssignale des FIR-Filters können mit den gewählten Filterkoeffizienten also auch für 16-Bit-Eingangspixel ohne Genauigkeitsverlust mit einem 25x18 Bit Multiplizierer gefenstert werden.

Reduzierung der Multiplizierer

Für die Koeffizienten des für dieses Projekt durch den Auftraggeber vorgegebenen FIR-Filters gilt

$$f[k] = -f[7-k], \quad k = 0, 1, \cdots, 7 \tag{4.7}$$

Durch diese Symmetrie kann die Anzahl der benötigten Multiplizierer halbiert werden, indem für die betragsgleichen Koeffizienten jeweils ein gemeinsamer Multiplizierer verwendet wird [32, S. 85f]. Eine entsprechende Umsetzung des FIR-Filters wird in diesem Projekt verwendet.

Soll ein FIR-Filter mit Koeffizienten verwendet werden, die die Bedingung 4.7 nicht erfüllen, muss das symmetrische FIR-Filter durch das allgemeine Filter aus der Vader-Bibliothek ausgetauscht werden.

Verifikation

Als Referenz für die Verifikation des Rohbildfilters wird der Simulink-eigene FIR-Filter-Block gewählt. Dazu werden die Blöcke in einer Test-Harness parallel geschaltet, Zufallsdaten an die Eingänge angelegt und die Ausgänge verglichen. Zufallsdaten werden gewählt, um die Wahrscheinlichkeit zu verringern, dass die Verifikation für einzelne Eingangszeitfolgen gelingt aber andere Eingangsfolgen, für die sie nicht gelingen würde, nicht geprüft werden.

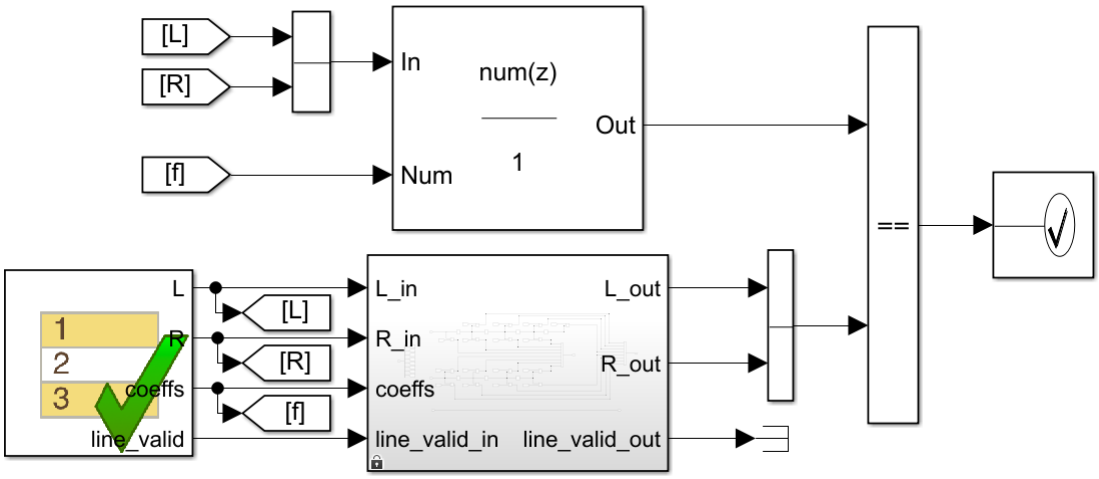


Abbildung 4.6: Verifikations-Harness für das FIR-Filter

4.1.4 Filterbank

Im Filterbank-Modul wird die Fensterfunktion auf die vorgefilterten Pixeldaten angewendet und der gefensterte Pixelstrom im Ortsfrequenzfilter weiterverarbei-

tet (vergleiche Abbildung 4.1).

Die Filterbank beinhaltet einen Speicher für Fenster- und Filterkoeffizienten für jedes der acht Ortsfrequenzfilter. Da in jedem Takt zwei benachbarte Pixel verarbeitet werden (L und R), werden diese Speicher in 512x16 Bit organisiert, sodass an jeder Speicherstelle die Faktoren für das jeweils linke und rechte Pixel nebeneinander liegen. Weil in jedem Taktzyklus nur eine Speicherstelle abgefragt werden kann, ist die Pixelverschiebung durch dieses Vorgehen auf ganzzahlige Vielfache von 2 beschränkt. Wenn der Algorithmus durch den DSP gestartet wird, wird die Verschiebung aller Filter auf 0 zurückgesetzt und erhöht sich anschließend nach jedem Zeilenbild um den vorgegebenen Faktor.

Filterkoeffizienten, die links aus dem Bild geschoben werden, werden rechts wieder hineingeschoben.

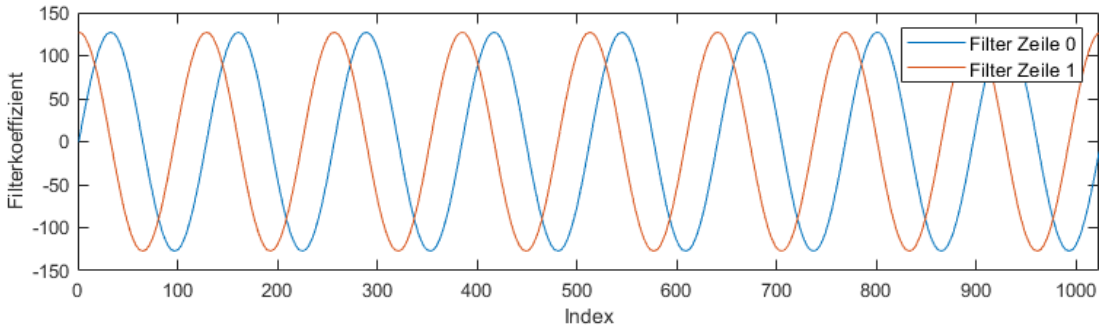


Abbildung 4.7: Verschiebung eines Filters mit Periode 128 Pixel und Verschiebung 32 Pixel/Frame

Wie in Abschnitt 4.1.1 gezeigt, steigt die Bitbreite durch die Multiplikation mit zwei 8-Bit-Werten (Fenster- und Filterfunktion) und das anschließende Aufsummieren von 1024 Einzelwerten um insgesamt 26 Bit. Das Lastenheft spezifiziert eine Breite der Einträge der Ortsfrequenzfilterfunktionen von 16 Bit, sodass eine Skalierung in der Filterbank notwendig wird.

Der DSP gibt dazu für jedes der acht Ortsfrequenzfilter einen Skalierungsparameter s an. Die Skalierung erfolgt durch eine Verschiebung des Datenworts unter Berücksichtigung des Vorzeichens um s Bit nach rechts, was einer Ganzzahldivision durch 2^s entspricht [5, S. 256].

Ist der Skalierungsfaktor nicht ausreichend groß, um das erste Vorzeichenbit von

links in die 16 Ausgabebits zu schieben, wird die Ausgabe je nach dem Vorzeichen der unskalierten Größe auf -2^{15} bzw. $+2^{15} - 1$ begrenzt. Dieser Sättigungsvorgang verhindert, dass ein zu großer positiver Wert durch die Skalierung in einen negativen Wert übergeht oder umgekehrt. Zusätzlich zeigt ein Overflow-Bit an, ob noch mehr als ein signifikantes Bit oberhalb der 16 Ausgabebits existiert, das durch die Skalierung verloren gegangen ist.

Abbildung 4.8 stellt den Skalierungsvorgang grafisch dar. Ein einfacher Overflow zwischen den Grenzen des 16-Bit-Wertebereichs und denen des 17-Bit-Wertebereichs kann nicht eindeutig erkannt werden, da durch die Sättigung nicht zwischen der Grenze des Wertebereichs und einem Overflow unterschieden werden kann. Durch das konstante Verweilen an der Grenze des Wertebereichs über mehrere Einträge ohne gleichzeitiges Overflow-Flag kann der DSP allerdings darauf schließen, dass wahrscheinlich ein 1-Bit-Overflow vorliegt. Ist das Overflow-Flag gesetzt, liegt mindestens ein 2-Bit-Overflow vor.

Dieses Vorgehen wurde unter der Annahme gewählt, dass der DSP dadurch schneller in der Lage sein wird, mit der optimalen Skalierung auf eine Änderung der Anregung des Filters zu reagieren.

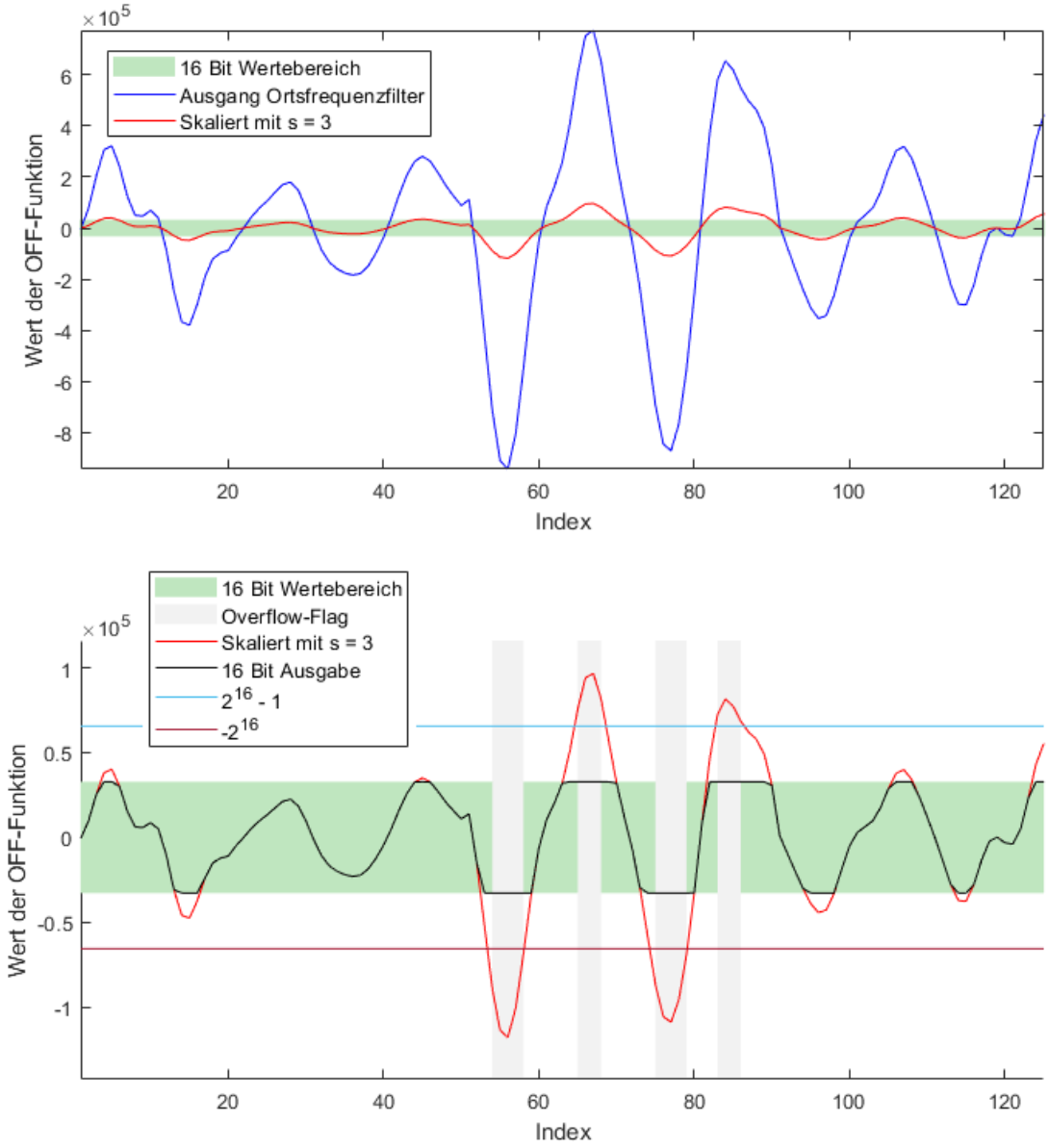


Abbildung 4.8: Ausgangsskalierung der Einträge der Ortsfrequenzfilterfunktionen

4.1.5 MCASP-Sender und -Empfänger

In Abschnitt 3.2.2 wurde das Prinzip der McASP-Schnittstelle eingeführt. Der folgende Abschnitt beschreibt die Modellierung eines McASP-Sendemoduls und eines McASP-Empfängermoduls in Simulink.

Aufgabe des Sendemoduls ist es, aus Eingangsdaten 8, 16 oder 32 Bit breite

McASP-Frames zu erzeugen und deren Beginn mit dem FS-Signal zu markieren.

Aufgabe des Empfangsmoduls ist es, auf ein FS-Signal zu warten und anschließend 8, 16 oder 32 Bit breite McASP-Frames zu empfangen und auszugeben.

Takterzeugung

Abbildung 3.6 zeigt, dass die drei McASP-Takte (für Filterbank-Daten, Rohbilder und Parametrierungsdaten) basierend auf dem Systemtakt als Referenz generiert werden um keine Taktübergänge an den MCASP-Schnittstellen zu erzeugen.

In Abschnitt 3.2.2 wurde die McASP-Übertragung basierend auf der Festsetzung einer maximalen McASP-Taktfrequenz von 10 MHz ausgelegt. Da der Systemtakt 40 MHz ist, ist keine Verfielfachung sondern nur eine Teilung des Takts notwendig.

Die minimale Ausgangsfrequenz des MMCM-Blocks beträgt 4,69 MHz [14, S. 38], die des PLL-Blocks beträgt 6,25 MHz [14, S. 40]. Daraus ergibt sich für die Generierung der McASP-Takte ein nutzbarer Frequenzbereich von 4,69 MHz - 10 MHz. Die MMCM- bzw PLL-Blöcke müssen explizit instantiiert werden [21] und haben als externe Komponenten keine direkte Entsprechung in Simulink.

Aus diesen Gründen wird für die Erzeugung der McASP-Takte auf die dedizierten Taktressourcen des Artix-7 verzichtet und stattdessen ein digitaler Teiler in Simulink modelliert.

Der Teiler basiert auf einem Zähler, der mit dem Systemtakt hochzählt und einem Delay-Block, dessen Ausgang jeweils bei erreichen eines bestimmten Zählwerts invertiert wird.

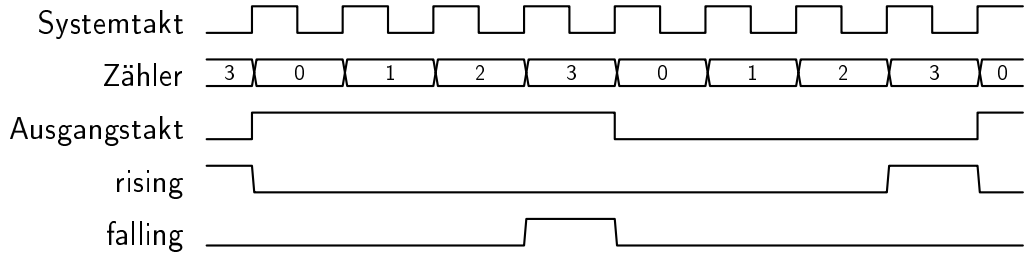


Abbildung 4.9: MCASP Taktteiler mit $Z_{\max} = 3$

Abbildung 4.9 zeigt die Funktionsweise des Taktteilers, der im Beispiel mit einem

maximalen Zählerwert von $Z_{\max} = 3$ betrieben wird. Es ergibt sich der Taktfaktor $N = (2 \cdot (Z_{\max} + 1))^{-1}$.

Für die möglichen McASP-Taktfrequenzen gilt mit diesem Frequenzteilermodul unter Berücksichtigung der festgelegten Obergrenze von 10 MHz:

$$f = \frac{40\,\mathrm{MHz}}{2 \cdot i} \quad , i = 2, 3, 4, \cdots$$

Neben dem Ausgabetakt werden zwei Signale (rising und falling) generiert, die eine steigende bzw fallende Flanke des Ausgangstakts ankündigen.

Da der Taktausgang an einem Delay-Block erzeugt wird, der, wie in Abschnitt 2.3.1 beschrieben, einem Flip-Flop entspricht, liegt der generierte Takt im synthetisierten FPGA am Q-Ausgang eines Flip-Flops an. Damit handelt es sich dabei aus Sicht der digitalen Logik um ein Datensignal, das über die Schaltmatrix und nicht das Taktnetzwerk verteilt wird. Dies ist ein weiterer Unterschied zur Takterzeugung über die erwähnten PLL- bzw MMCM-Ressourcen. In dieser Arbeit wird in diesem Kontext weiterhin der auf die Funktion bezogene Begriff Taktsignal verwendet, auch wenn es sich tatsächlich um ein zum Systemtakt synchrones Steuer- bzw Datensignal handelt.

Abbildung 4.10 zeigt den Übergang des McASP-Taktsignals von einem Datensignal im FPGA zu einem Taktsignal im DSP.

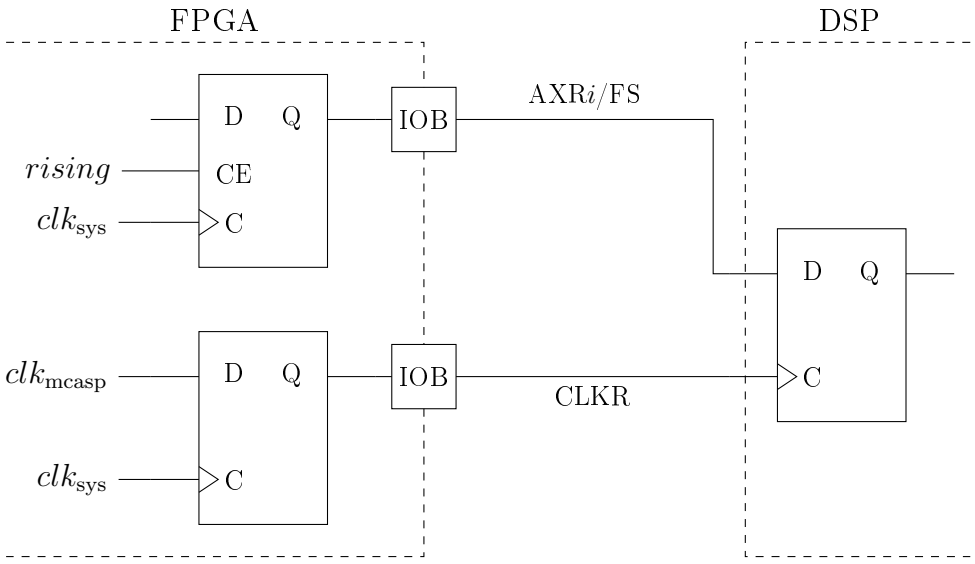


Abbildung 4.10: Schema des MCASP-Senders in Hardware

Die McASP-Schnittstelle sieht standardmäßig ein "shift on rising, sample on falling"-Verhalten vor [23, S. 2763ff]. Das bedeutet, dass der Sender neue Daten bei steigenden Taktflanken auf die Leitung gibt und der Empfänger sie bei der fallenden Flanke abruft.

Dadurch können die Daten gegenüber dem Takt (unter Vernachlässigung der Setzzeiten) um bis zu eine halbe Taktperiode verzögert sein, um noch korrekt gelesen zu werden.

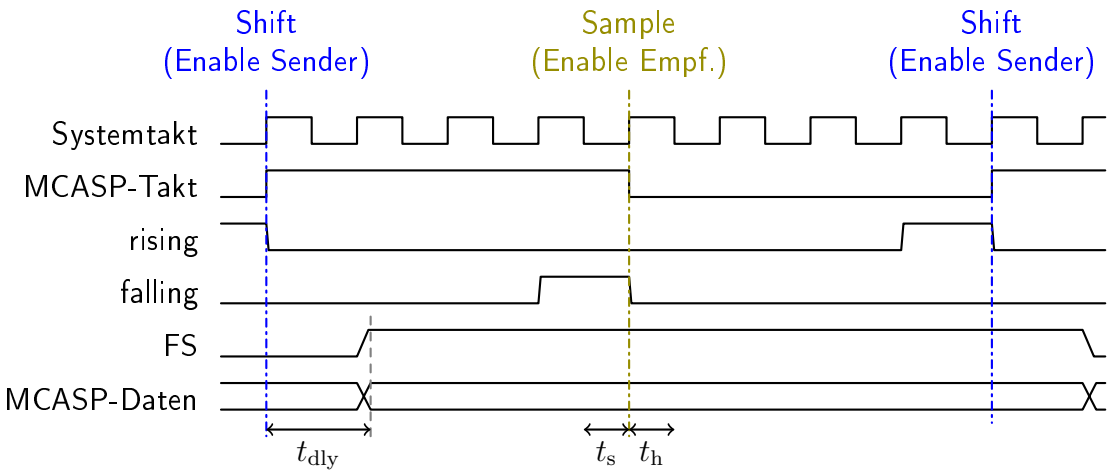


Abbildung 4.11: Shift- und Sample-Zeitpunkte bezogen auf McASP-Takt und Systemtakt

Abbildung 4.11 zeigt die Shift- und Samplingzeitpunkte relativ zum erzeugten MCASP-Takt und dem Systemtakt sowie qualitativ die Setz- und Haltezeiten relativ zur Sampling-Flanke und die Ausgangsverzögerung relativ zur Shifting-Flanke.

Die Abbildung zeigt außerdem, dass die Shift- und Samplingzeitpunkte zwar bezogen auf den McASP-Takt entsprechend dem "shift on rising, sample on falling"-Prinzip auf unterschiedlichen Flanken liegen, bezogen auf den Systemtakt allerdings (bedingt durch die Taktteilung) immer auf die steigenden Flanken fallen. Dies ist das geplante Verhalten, weil die Ein- und Ausgangsflipflops dadurch, wie der Rest des Systems, mit der steigenden Flanke des Systemtakts getaktet werden. Die rising- bzw falling-Signale dienen als Clock Enable-Signal für diese Flip-Flops und stellen sicher, dass sie nur ein mal pro McASP-Taktperiode zu den jeweiligen Taktflanken aktiv werden.

Dieses Verhalten entspricht prinzipiell einer eigenen Samplingrate in Simulink, wie sie in Abschnitt 2.3.1 beschrieben wurde.

Frame-Synchronisation

Für die Erzeugung bzw die Auswertung des FS-Signals werden Zustandsmaschinen in Stateflow modelliert.

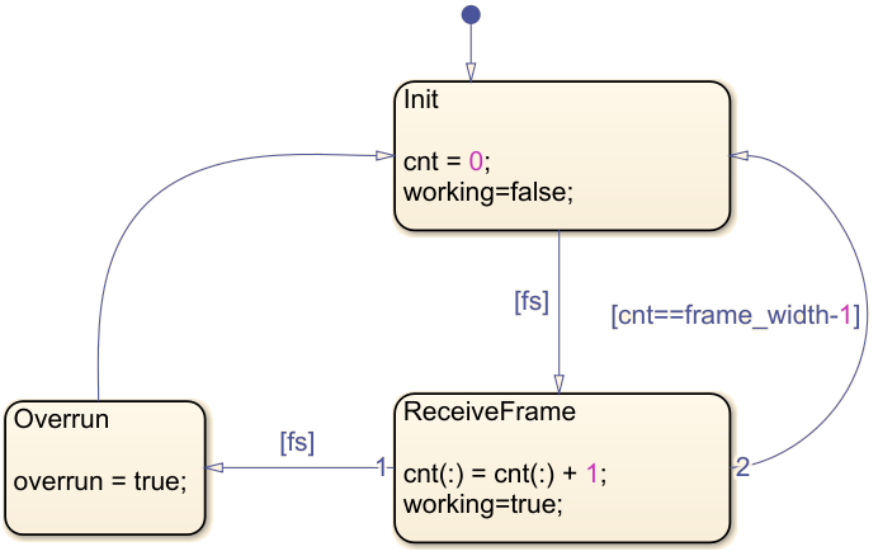


Abbildung 4.12: Zustandsmaschine für die Frame-Verarbeitung des MCASP-Empfängers

Abbildung 4.12 zeigt die Implementierung für das Empfängermodul: Sobald ein FS-Signal erkannt wird, verbleibt der Automat für die Dauer des Frames im ReceiveFrame-Zustand. Wird in diesem Zustand, also bevor das aktuelle Frame vollständig empfangen wurde, nochmals ein FS-Signal erkannt, wird ein Overrun-Fehler ausgelöst.

Die Zustandsmaschine wird über das falling-Signal als Enable-Eingang ein mal pro MCASP-Taktperiode ausgeführt. Dies stellt einerseits sicher, dass die empfangenen Bits synchron zum McASP Takt gezählt werden und dass andererseits der FS-Eingang nur beim Anliegen einer negativen Flanke ausgewertet wird.

Schieberegister

Das Empfangs-Schieberegister wird über das falling-Signal als Enable gesteuert. Wie beim Empfangs-Zustandsautomaten ist damit sichergestellt, dass das Eingangssignal nur bei der entsprechenden Flanke gesampelt wird um Synchronisationsprobleme zu vermeiden.

Das Sende-Schieberegister funktioniert im Prinzip gleich, der Schiebevorgang wird hier durch das rising-Signal ausgelöst. Zusätzlich können neue Daten am Eingang

bei jeder Flanke des Systemtakts übernommen werden. Durch diese Funktionalität müssen zu versendende Daten nicht bis zur nächsten McASP-Taktflanke in einem Halteglied gespeichert werden.

Randbedingungen

Die Pinzuordnungen für die McASP-Schnittstellen zu den FPGA-Pins werden aus dem Vader-Schaltplan ([3]) entnommen.

Für die Timing-Bedingungen werden aus dem Datenblatt des 66AK2G12 die Setz- und Haltezeiten für die McASP-Eingangspins des ICs und die minimale und maximale Verzögerung der McASP-Ausgangspins gegenüber dem McASP-Takt entnommen: [33, S. 138ff]

- $t_\mathrm{s} = 4\,\mathrm{ns}$
- $t_\mathrm{h} = 1{,}6\,\mathrm{ns}$
- $t_\mathrm{dly,min} = 2\,\mathrm{ns}$
- $t_\mathrm{dly,max} = 14\,\mathrm{ns}$

Die Leitungen führen vom FPGA über die Leiterbahnen auf der Vader-Platine und die Leiterbahnen auf der DSP-Platine zum DSP. Die Verzögerungen auf der DSP-Platine sind unbekannt, die größte Differenz zwischen den Verzögerungen zweier McASP-Leitungen auf der Vader-Platine wird aus dem Layout zu etwa 0,3 ns bestimmt.

Um die Zeitbedingungen zu verletzen, müsste die Verzögerung so groß werden, dass im zeitlichen Abstand einer halben McASP-Taktperiode die Setz- bzw Haltezeiten verletzt werden (siehe Abbildung 4.11). Beim maximalen McASP-Takt von 10 MHz entspricht die zeitliche Differenz zwischen der steigenden und der fallenden Flanke $\frac{1}{2 \cdot 10\,\mathrm{MHz}} = 50\,\mathrm{ns}$.

Dieser Wert wird als so groß gegenüber den Setz- und Haltezeiten sowie der Laufzeitdifferenzen auf der Vader-Platine betrachtet, dass die Timing-Randbedingungen für die McASP-Module als vernachlässigbar angesehen werden.

Verifikation

Die Modelle der McASP-Sender und -Empfänger werden in einer gemeinsamen Test-Harness auf Modulebene und in Kapitel 5 mit dem Gesamtmodell verifiziert. Außerdem wird aus den Modulen HDL-Code generiert und das Senden und Empfangen von Daten bei verschiedenen Taktfrequenzen mit dem DSP getestet.

4.1.6 Übertragung der Filterbank-Daten

Die Aufteilung der Filterbank-Daten auf die elf Serialisierer wurde in der Planungsphase festgelegt. Zusätzlich werden zwei Serialisierer für Metadaten hinzugefügt, über die auf dem DSP Fehler in der Übertragung erkannt werden können.

Auf dem ersten zusätzlichen Serialisierer wird ein fortlaufender Zählerwert übermittelt, über den der DSP die Kohärenz der Daten zwischen den einzelnen Einträgen der Ortsfrequenzfilterfunktionen und zwischen den Batches verifizieren kann.

Der zweite zusätzlich Serialisierer überträgt eine Prüfsumme über den anderen 12 Datenworten, durch die Übertragungsfehler erkannt werden können.

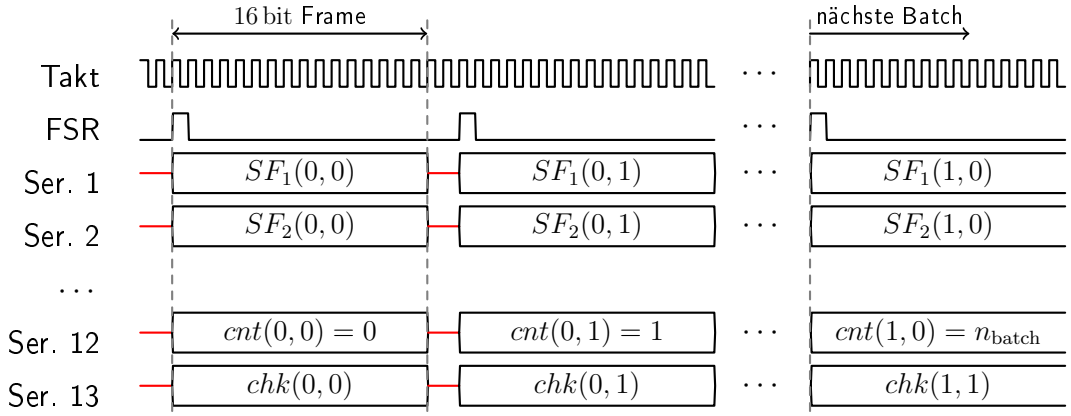


Abbildung 4.13: Übertragung der Filterbank-Daten

Abbildung 4.13 zeigt das Muster der Zählwerte über zwei Batches. Allgemein gilt für die Zählerwerte:

$$i \cdot n_{\text{batch}} + j \equiv cnt(i,j) \pmod{2^{16}}$$

Die Prüfsumme ist als Paritätswort implementiert, dessen einzelne Bits sich aus der Summe der jeweiligen Bits aller anderen Datenwörter modulo 2 berechnen. Auf Empfängerseite kann dadurch eine ungerade Anzahl an Bitfehlern je Stelle der 16-Bit-Wörter erkannt werden [13, S. 432].

4.1.7 Übertragung der Rohbilder

Für die Übertragung der Rohbilddaten wurden in der Planungsphase vier Serialisierer zugeteilt. Um möglichst wenige DMA-Events auf Empfängerseite auszulösen, werden die Pixeldaten in einem First-In-First-Out-Puffer (FIFO) zwischengespeichert und als 32-Bit-Frames versendet.

Übertragen wird jeweils das erste Rohbild innerhalb einer Batch. Über den Zähler, der mit den Filterdaten übermittelt wird, kann die Synchronisation der Rohbild- mit den Filterdaten geprüft werden.

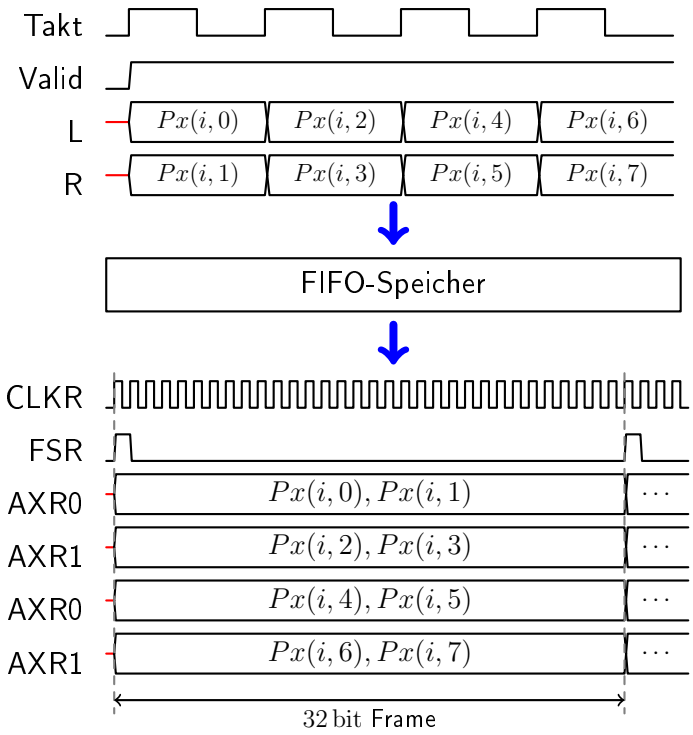


Abbildung 4.14: Zwischenspeicherung und Übertragung der Rohbilddaten

4.1.8 Kantenerkennung

Das Lastenheft schreibt eine auswählbare Vorfilterung der Rohbilddaten für die Kantenerkennung in einem rekursiven Mittelwertfilter der Längen $M \in \{16, 32, 64\}$ vor.

Die Gleichung eines solchen Filters lautet: [34, S. 277]

$$y_{\text{lit}}[n] = \frac{1}{M} \sum_{i=0}^{M-1} x[n+i] \tag{4.8}$$

Der Ausgangswert besteht also aus dem Mittelwert des aktuellen und der $M-1$ folgenden Eingangswerte. Da die Pixelwerte in diesem Projekt als Datenstrom verarbeitet werden und bei der Verarbeitung eines Pixels die nachfolgenden Pixelwerte noch nicht bekannt sind (die vorherigen aber durch Delay gespeichert werden können), wird eine andere Formel als Grundlage verwendet:

$$y[n] = y_{\text{lit}}[n-(M-1)] = \frac{1}{M} \sum_{i=0}^{M-1} x[n-i] \tag{4.9}$$

Hier besteht der Ausgangswert aus dem Mittelwert des aktuellen und der $M-1$ vorherigen Eingangswerte. Gegenüber Gleichung 4.8 entspricht das einer Verschiebung der Ausgangsfolge um $M-1$.

Die Gleichung kann rekursiv geschrieben werden: [34, S. 283]

$$y[n] = \frac{1}{M} \sum_{i=0}^{M-1} x[n-i] = \frac{1}{M} \Big(M \cdot y[n-1] - x[n-M] + x[n] \Big) \tag{4.10}$$

$x[n-M]$ ist dabei das Pixel, das in jedem Schritt aus dem Filter herausläuft, $x[n]$ ist das Pixel, das neu hinzukommt. Die Umsetzung dieser Rechenvorschrift erfordert es, M alte Pixelwerte zu speichern, um in jedem Schritt $x[n-M]$ zu kennen. Unter der Annahme, dass $x[n-M] \approx y[n-1]$ gilt, wird eine vereinfachte Formel geschrieben:

$$y[n] \approx \frac{1}{M} \Big(M \cdot y[n-1] - y[n-1] + x[n] \Big) = \frac{(M-1) \cdot y[n-1] + x[n]}{M} \tag{4.11}$$

Die Multiplikation mit M und Division durch M in dieser Formel lassen sich durch Schiebeoperationen realisieren, weil M auf Zweierpotenzen beschränkt ist. Die Umsetzung dieser Formel kann daher für dieses Projekt in Hardware ohne Multiplikatoren erfolgen. [35]

Analog zum Vorgehen beim FIR-Filter werden die Zeitreihen x und y mit den linken bzw. rechten Pixeln ausgedrückt:

$$\begin{aligned} x[2n-1] &= L_{\text{in}}[n] \\ x[2n] &= R_{\text{in}}[n] \\ L_{\text{out}}[n] &= y[2n-1] \\ R_{\text{out}}[n] &= y[2n] \end{aligned} \tag{4.12}$$

Einsetzen in die Rechenvorschrift 4.11 liefert:

$$\begin{aligned} L_{\text{out}}[n] = \quad y[2n-1] &\approx \frac{(M-1)\cdot y[2n-1-1] + x[2n-1]}{M} \\ &= \frac{(M-1)\cdot y[2(n-1)] + x[2n-1]}{M} \\ &= \frac{(M-1)\cdot R_{\text{out}}[n-1] + L_{\text{in}}[n]}{M} \end{aligned} \tag{4.13}$$

$$\begin{aligned} R_{\text{out}}[n] = \qquad y[2n] &\approx \frac{(M-1)\cdot y[2n-1] + x[2n]}{M} \\ &= \frac{(M-1)\cdot L_{\text{out}}[n] + R_{\text{in}}[n]}{M} \end{aligned}$$

Nach der Mittelwertfilterung vergleicht die Kantenerkennung die Pixelwerte in einem parametrierbaren Betrachtungsbereich (Region of Interest, ROI) mit parametrierbaren Schwellenwerten. Als steigende Flanke wird jeweils das erste Überschreiten des oberen Schwellenwerts zurückgegeben, als fallende Flanke das erste Unterschreiten des unteren Schwellenwerts nach dem letzten Überschreitens des oberen Schwellenwerts. Auf diese Weise werden als Kanten erkannte Strukturen auf der Objektoberfläche verworfen. [7, S. 18]

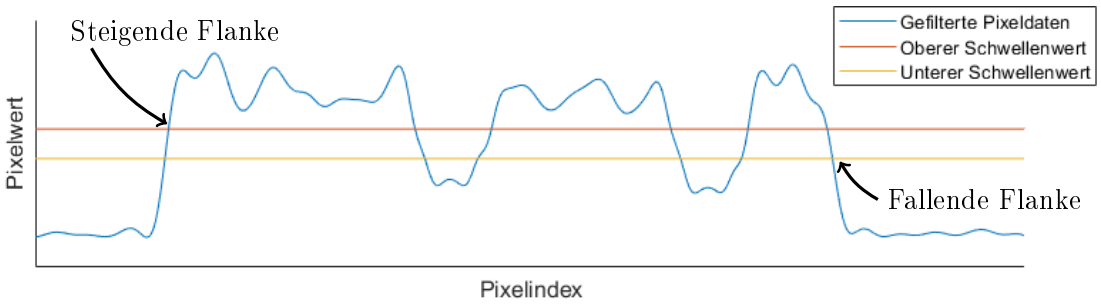


Abbildung 4.15: Funktionsprinzip der Flankenerkennung [7]

Die Kantenerkennung wurde während des Projekts auf zwei Arten implementiert.

Abbildung 4.16 gibt einen Überblick über die Komplexität der ersten Variante, die aus einem Simulink-Blockdiagramm besteht.

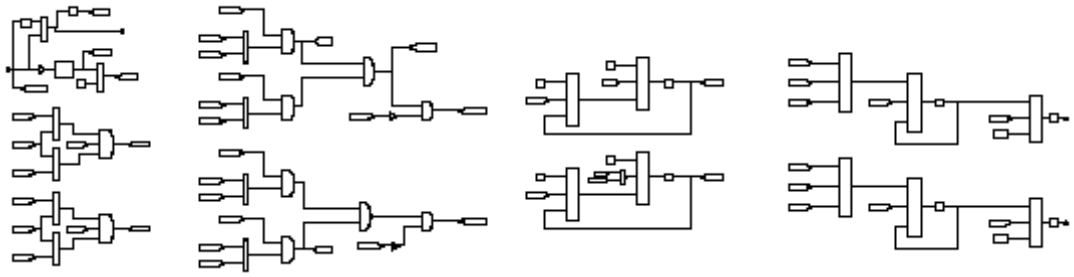


Abbildung 4.16: Implementierung der Kantenerkennung als Blockschaltbild

Bei der zweiten Variante wurde die Logik in verschachtelten Matlab-Kontrollstrukturen innerhalb eines Stateflow-Diagramms implementiert (Abbildung 4.17).

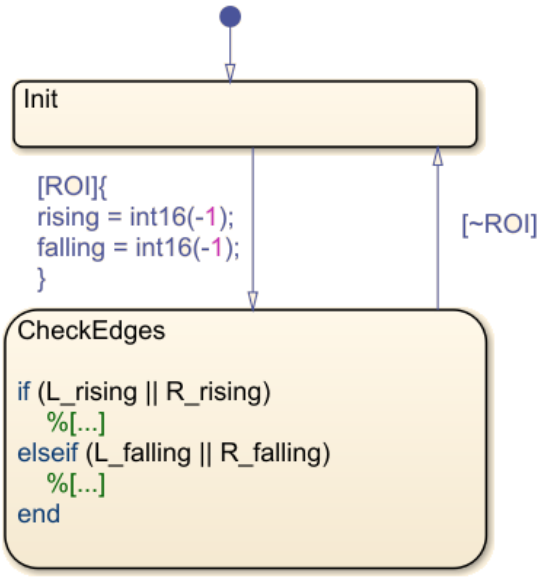


Abbildung 4.17: Darstellung der Kantenerkennung mit verschachtelten Kontrollstrukturen [35]

Die zweite Variante wird als übersichtlicher und besser wartbar betrachtet und daher im Projekt weiterverwendet.

Verifikation

Das Kantenerkennungsmodul wird durch eine grafische Plausibilitätsprüfung verifiziert. Dazu wird ein Rohbild generiert, das anschließend mit den Rechenvorschriften 4.10 (1) und 4.11 (2) gefiltert wird. Anschließend werden die erkannten Kanten grafisch dargestellt und mit dem erwarteten Verhalten verglichen.

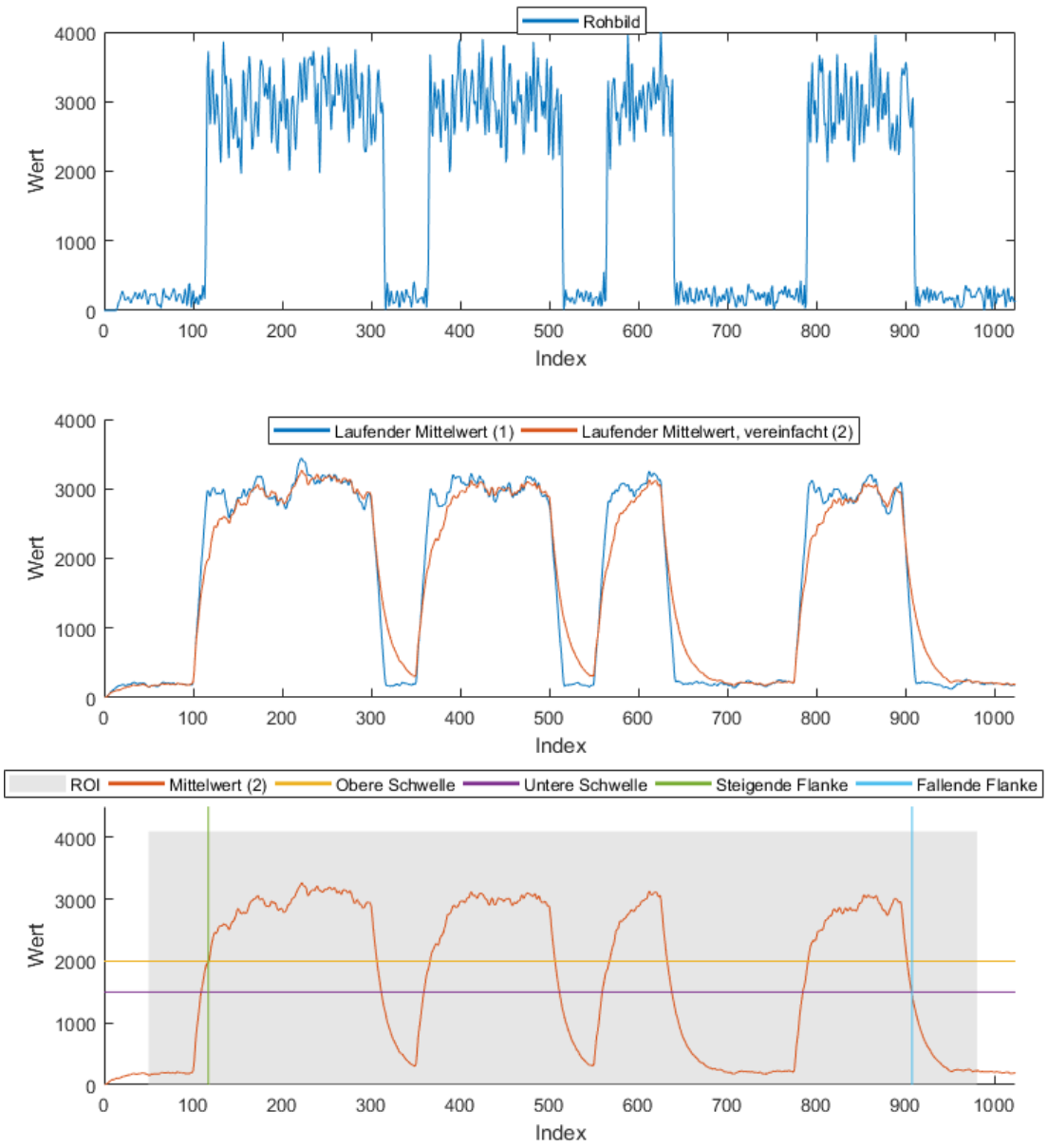


Abbildung 4.18: Grafische Verifikation der Kantenerkennung

4.1.9 Inkrementalgeber-Schnittstelle

Die Inkrementalgeber-Schnittstelle wird vom DSP mit einer Anzahl Flankenwechsel, einer Periode und einer Richtung parametriert und gibt auf zwei Spuren Quadratursignale aus. [22]

Die Periode wird als Intervall zwischen zwei Flankenwechseln in Systemtaktperi-

oden angegeben.

Das Modell wird durch eine Zustandsmaschine realisiert, die die Spuren nach jeweils einer Periodendauer in einer Reihenfolge abhängig vom Richtungsparameter umschaltet.

Verifikation

Die Inkrementalgeber-Schnittstelle wird in einer Timing-Simulation verifiziert.

Um den nahtlosen Übergang zwischen zwei Pulsfolgen zu verifizieren werden zwei Befehle zur Parametrierung der Schnittstelle nacheinander gesendet (siehe Abschnitt 4.1.11). Abbildung 4.19 zeigt ein Beispiel, in dem die erste Pulsfolge aus sieben und die zweite aus drei Pulsen besteht. In der Timing-Simulation wird die Zahl der Pulse, die Periodendauer (t_p) und der Übergang zwischen den Folgen verifiziert.

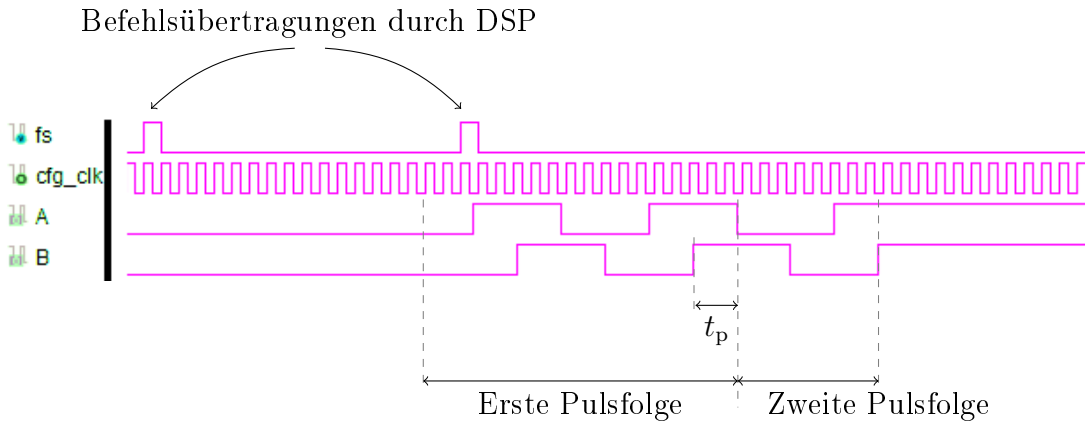


Abbildung 4.19: Timing-Simulation zur Verifikation der Inkrementalgeber-Schnittstelle

4.1.10 SSI

Die synchron-serielle Schnittstelle besteht aus einem Master, der Daten empfängt und einem Slave, der Daten versendet. Im Slave wird ein Schieberegister mit einem Weginkrement geladen. Der Master gibt einen Takt vor, auf dessen steigende Flanken die Daten bitweise, beginnend mit dem höchstwertigen Bit (MSB), aus dem Schieberegister auf die Datenleitung getaktet werden. [36]

Nach der Übertragung des LSB (Least Significant Bit) muss das Taktsignal vom Master für eine vereinbarte Zeitspanne t_m auf einem High-Pegel gehalten werden, damit ein neues Datenwort in das Schieberegister geladen wird. Beginnt der Master vor dem Ablauf dieser Zeitspanne wieder mit der Übertragung eines Takts, so wird das alte Datenwort erneut übertragen. Auf diese Weise kann der Master die gleichen Daten mehrmals abfragen, um Übertragungsfehler erkennen zu können. [36]

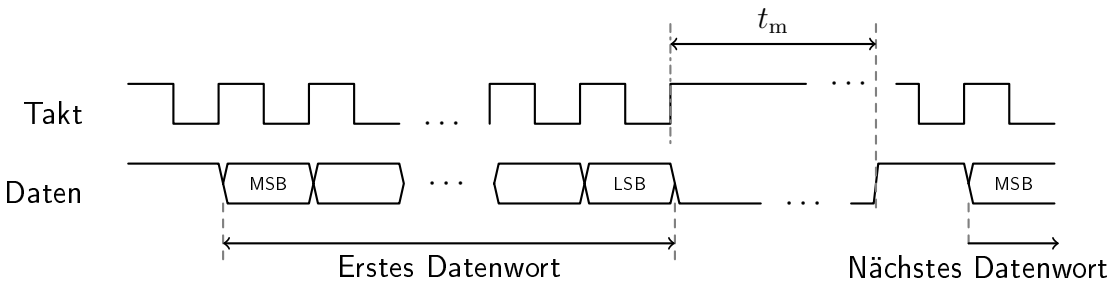


Abbildung 4.20: Funktionsprinzip SSI [36]

Die maximale Frequenz des SSI-Takts ist durch das Lastenheft auf 2 MHz festgelegt. Das Vader-FPGA hat als SSI-Slave keinen Einfluss auf die Phase oder Frequenz des SSI-Takts. Das SSI-Datenwort, das in das Schieberegister geladen werden muss, liegt synchron zum Systemtakt vor. Darüber hinaus kann die Zeit t_m nicht anhand des SSI-Takts gemessen werden, weil dieser prinzipbedingt während dieser Zeitspanne nicht aktiv ist und außerdem die Frequenz unbekannt ist. Aus diesen Gründen müssen grundsätzlich mindestens zwei Informationen über die Taktgrenze transportiert werden:

1. Das SSI Datenwort
2. Die Information über den Ablauf von t_m

Im Folgenden werden zwei Varianten zur Implementierung des SSI-Slaves im Vader-FPGA erörtert.

Beiden Varianten ist aus den genannten Gründen gemein, dass t_m im Systemtaktbereich gemessen wird und daher dort die Flanken des SSI-Takts ausgewertet werden. Da der SSI-Takt als asynchrones Signal vorliegt, wird er über den in der Literatur empfohlenen Synchronisierer aus zwei Flip-Flops einsynchroni-

siert. Dadurch ergibt sich, je nach Ankunftszeit der SSI-Taktflanke relativ zur Systemtaktflanke, eine Verzögerung von einem bis zwei Systemtaktzyklen.

Die erste Variante basiert auf einem Schieberegister im SSI-Taktbereich, das aus dem Systemtaktbereich geladen wird. [35]

Das spezielle Problem der Übertragung eines parallelen Busses über eine Taktgrenze wurde in Abschnitt 2.3.1 an einem Beispiel erläutert: Selbst wenn Setz- und Haltezeiten nicht verletzt werden, kann wegen der unterschiedlichen Laufzeiten der einzelnen Bits ohne besondere Maßnahmen nicht garantiert werden, dass das Datenwort korrekt übernommen wird.

Deshalb muss in dieser Variante sichergestellt sein, dass das Datenwort auf Systemtaktseite vor und während der Übernahme in den SSI-Taktbereich so lange konstant ist, dass die einzelnen Bits korrekt übertragen werden.

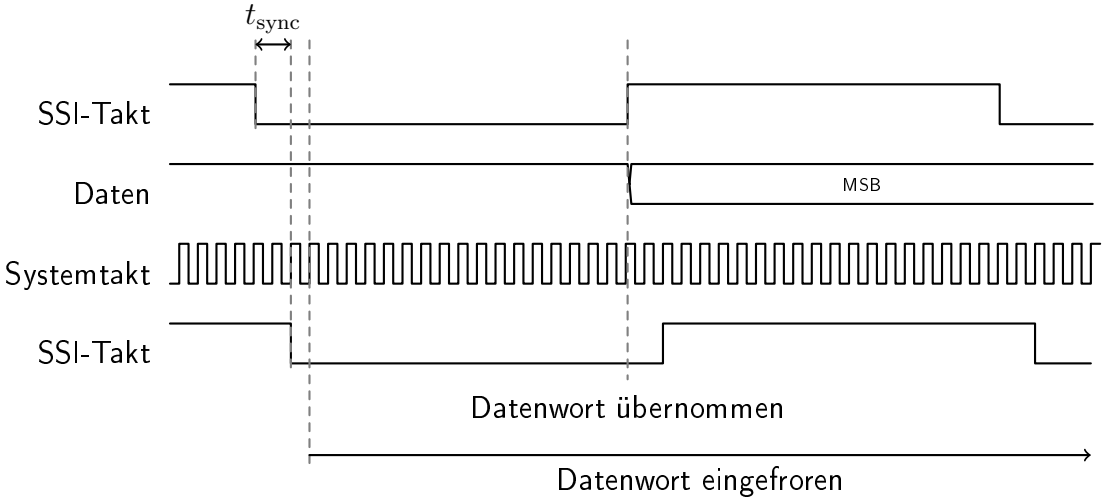


Abbildung 4.21: Funktionsprinzip SSI-Slave Variante 1 [35]

Das Prinzip, nachdem der Taktübergang in der ersten Variante gesichert wird, ist in Abbildung 4.21 dargestellt.

Das Datenwort wird im Systemtaktbereich nach dem Erkennen der ersten fallenden Flanke des SSI-Takts eingefroren. Weil der Systemtakt in diesem Projekt mindestens um den Faktor 20 größer ist als der SSI-Takt, wird diese fallende Flanke trotz der Verzögerung durch den Synchronisierer noch vor der ersten steigenden Flanke erkannt.

Es wird die Annahme getroffen, dass die einzelnen Bits des Datenbusses nach der verbleibenden Zeit bis zur steigenden Flanke (wenn das erste Bit ausgetaktet wird) korrekt im SSI-Taktbereich anliegen.

Eine Möglichkeit, um ein solches Subsystem mit asynchronem Taktsignal zu modellieren, wurde mit der "Trigger as Clock"-Funktion in Abschnitt 2.3.1 eingeführt. Unter den getroffenen Annahmen ist die korrekte Übernahme der Daten in das Subsystem (nach Abbildung 4.21) sichergestellt. Simulink erzwingt die Verwendung von Synchronisierungsregistern an den Ein- und Ausgängen allerdings generell in Subsystemen mit einem "Trigger as Clock".

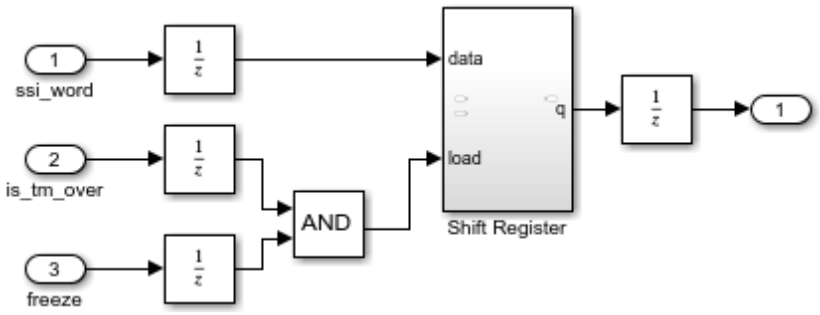


Abbildung 4.22: SSI-Slave Prinzip übertragen auf "Trigger as Clock"

Abbildung 4.22 stellt ein solches Subsystem nach dem erläuterten Prinzip dar. Es ergeben sich gegenüber dem Zeitverlauf aus Abbildung 4.21 zusätzliche Verzögerungen von einer SSI-Taktperiode an den Eingängen. Da das Schieberegister allerdings, wie die Abbildung zeigt, innerhalb der ersten Low-Halbperiode des SSI-Takts geladen werden muss, um das MSB rechtzeitig auf die Datenleitung zu takten, kann das erläuterte Prinzip nicht mit dem "Trigger as Clock"-Subsystem umgesetzt werden.

Bei der zweiten Variante wird das Schieberegister im Systemtaktbereich implementiert und nicht direkt mit dem SSI-Takt angesteuert:

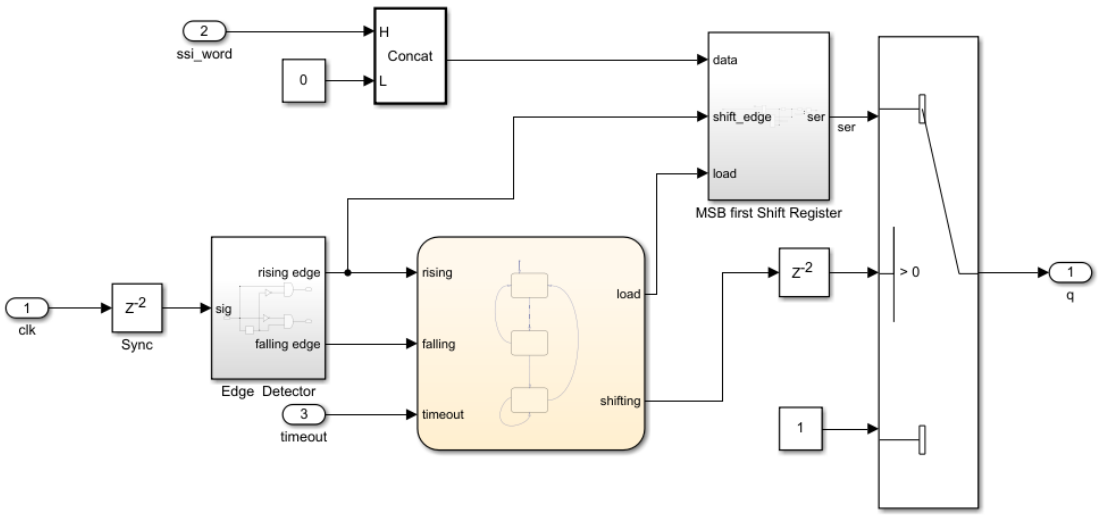


Abbildung 4.23: SSI-Slave im Systemtaktbereich

Nachteil dieser Variante ist, dass es sich nicht mehr um eine synchron-serielle Schnittstelle im Wortsinn handelt, da das Austakten der Daten nicht mehr wirklich synchron mit dem SSI-Takt, sondern mit einer auf den Systemtakt synchronisierten Version des SSI-Takts erfolgt. Durch die Synchronisationsregister im Takteingang verzögern sich die Daten gegenüber der echten Taktflanke.

Um die Verzögerung am Datenausgang nicht noch weiter zu erhöhen, muss sichergestellt werden, dass kein Delay Balancing auf das SSI-Subsystem angewendet wird. Aus diesem Grund wird das Delay Balancing global für das Modell deaktiviert und einzeln für die Module mit Multiplikatoren (Filterbank und Rohbildfilter) aktiviert.

Der Vorteil der zweiten Variante liegt in der weniger komplexen Implementierung in Simulink. Es wird nur ein einzelner Synchronisierer benötigt, der mit Delay-Blöcken modelliert werden kann (Abbildung 4.23).

Verifikation

Zur Verifikation der SSI-Schnittstelle werden die zwei Varianten auf dem Artix-7 implementiert und eine Timing-Simulation durchgeführt, aus der die entstehenden Verzögerungen abgelesen werden können.

In der Timing-Simulation (Abbildung 4.24) zeigt sich, dass die Verzögerung des SSI-Datensignals gegenüber der Taktleitung mit der zweiten Variante wegen der Synchronisierung des SSI-Takts etwa 75 ns beträgt, was drei Systemtaktperioden entspricht.

Bei Variante 1 ergibt sich eine geringere Verzögerung von etwa 7 ns durch die Verzögerungen der Schaltmatrix.

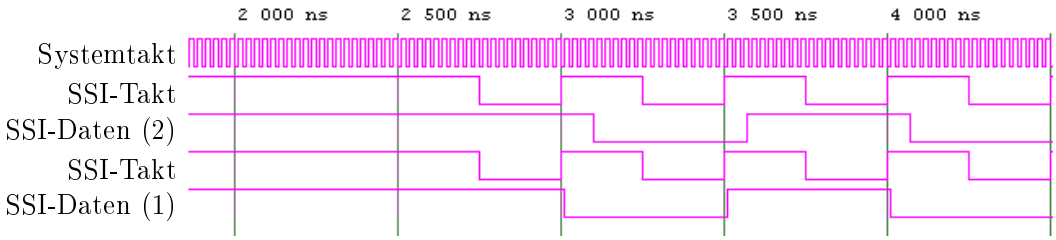


Abbildung 4.24: Timing-Simulation der SSI-Slave-Varianten

Der SSI-Master liest die Datenleitung zur fallenden Flanke ein [36].

Die bei Variante 2 entstehende Verzögerung wird als gegenüber der halben Taktperiode des maximalen SSI-Takts so klein betrachtet, dass angenommen wird, dass der SSI-Master die Daten korrekt empfängt. Wegen der einfachen Modellierbarkeit in Simulink wird die zweite Variante daher im Projekt verwendet.

Die physikalische Ebene des SSI wurde im Hardwareprojekt verifiziert. [3]

4.1.11 Parametrierungsmodul und Konfigurationsmodus

Um die verschiedenen Elemente der Signalverarbeitung im Betrieb parametrieren zu können, wird ein Konfigurationsmodul entwickelt.

Aus dem Lastenheft ergeben sich Parameter, die nur im Initialisierungsmodus gestellt werden müssen:

- Filterkoeffizienten und -verschiebung der acht Ortsfrequenzfilter
- Fensterungskoeffizienten der acht Ortsfrequenzfilter
- Framerate der Kamera

- Batchlänge
- ROI, Schwellenwerte und Filterauswahl für die Kantenerkennung

Weitere Parameter müssen im laufenden Betrieb änderbar sein:

- Timeout und Datenwort der synchron-seriellen Schnittstelle
- Richtung, Pulsanzahl und -periode der Inkrementalgeberschnittstelle
- Belichtungszeit der Kamera
- Skalierung der Ortsfrequenzfilterfunktionen

Konfigurationsmodus

Um das Vader-FPGA in den Initialiserungsmodus zu versetzen, wird ein eigener Pin zugeteilt (algorithm_enable). Liegt dieser Pin auf einem Low-Pegel, ist der Algorithmus angehalten und alle Parameter können verstellt werden. Wird der Pin vom DSP auf einen High-Pegel gezogen, startet der Algorithmus synchron zum Beginn des nächstes Zeilenbildes. Die Regelung des Konfigurationsmodus geschieht durch eine Stateflow-Zustandsmaschine.

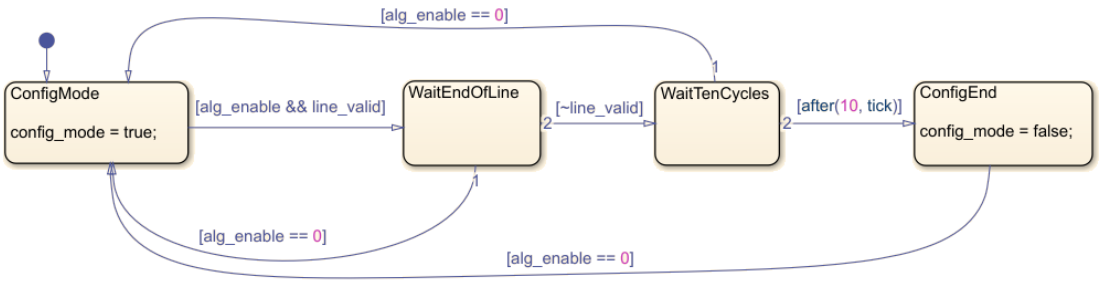


Abbildung 4.25: Zustandsmaschine des Konfigurationsmodus

Protokoll

Für die Übertragung der Parametrierungsdaten stehen nach Zuteilung der Serialisierer für die Filterbank- und Rohbilddaten noch sechs McASP-Serialisierer zur Verfügung. Um Übertragungsfehler erkennen zu können, wird festgelegt, dass

ein Serialisierer mit einer Prüfsumme belegt wird. Die restlichen fünf Serialisierer werden mit einem Befehlscode (Operation Code, Opcode) und zugehörigen Parametern belegt.

Die Breite des McASP-Frames wird auf 16 Bit festgelegt, da sich alle Module, die außerhalb des Konfigurationsmodus parametrierbar sein müssen, in je $5 \cdot 16Bit$ und somit mit einem einzelnen Befehl konfigurieren lassen:

- Inkrementalgeber: 32-Bit-Periode, 32-Bit-Pulsanzahl, 1-Bit-Richtung
- SSI: 32-Bit-Datenwort, 16-Bit-Timeout
- Skalierungsfaktoren der Ortsfrequenzfilter: $8 \cdot 6$ Bit
- Kamera: 16-Bit-Belichtungszeit

Basierend auf der Festlegung auf ein 16-Bit-Frame, wird folgende Befehlsstruktur vereinbart. Je nach Befehl können die Parameter als vier 16-Bit-Parameter oder als acht 8-Bit-Parameter interpretiert werden.

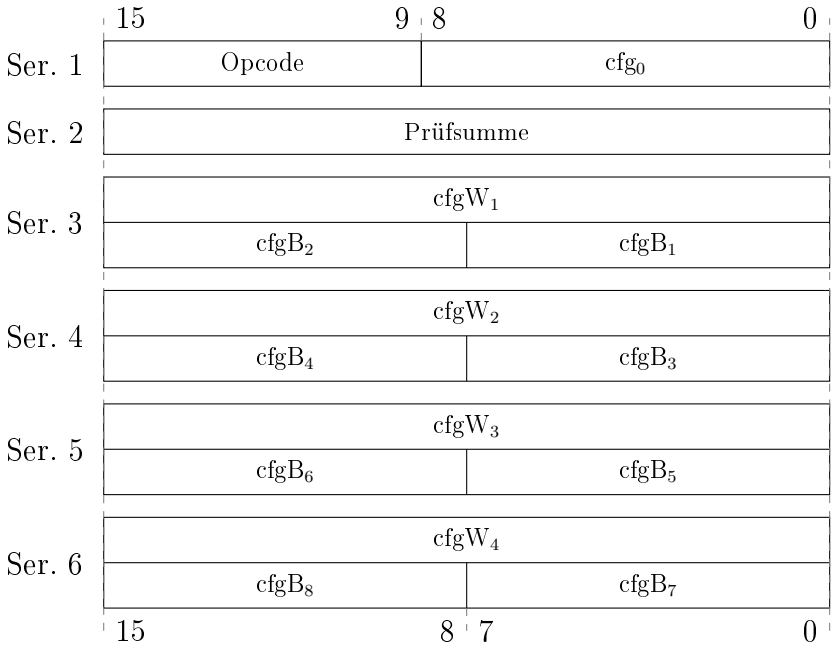


Abbildung 4.26: Framestruktur Parametrierung

Die Berechnung der Prüfsumme erfolgt analog zur Filterbank als Paritätswort über die einzelnen Bits der anderen fünf 16-Bit-Wörter. Wird ein Prüfsummenfehler erkannt, wird der fehlerhafte Befehl ignoriert und der error_chksum-Pin

wird auf einen High-Pegel gezogen. Der DSP erkennt den Prüfsummenfehler über das Abfragen des Pins oder einen Interrupt und setzt den Fehler durch das Übertragen eines speziellen Befehls zurück.

- OP_FILTER: Paralleles Setzen der Filterkoeffizienten der Ortsfrequenzfilter 1-4 bzw 5-8
- OP_WINDOW: Paralleles Setzen der Fensterungskoeffizienten der Ortsfrequenzfilter 1-4 bzw 5-8
- OP_SHIFT: Paralleles Setzen der Verschiebung der Filter 1-8
- OP_FRAME_SETUP: Setzen der Kamera-Framerate, Belichtungszeit und Batchlänge
- OP_EDGE_DETECT: Setzen der ROI, Schwellenwerte und Filterauswahl für die Kantenerkennung
- OP_CHKRESET: Zurücksetzen des Prüfsummenfehlers
- OP_SSI: Setzen des SSI-Datenworts und des SSI-Timeouts
- OP_INCREMENTAL: Setzen der Pulsanzahl, Periodendauer und Richtung des Inkrementalgebers
- OP_EXPOSURE: Setzen der Belichtungszeit
- OP_SCALE: Paralleles Setzen der Skalierungen der Ortsfrequenzfilter 1-8

4.2 FPGA-Konfigurationsspeicher

Das Vader-FPGA ist mit einem QSPI-Flash verbunden, von dem es den Bitstream zur Konfiguration laden kann. Es zeigte sich, dass der verwendete Flash-Baustein nicht von Xilinx Vivado unterstützt ist, sodass das Beschreiben des Flashspeichers mit einem Bitstream über Vivado nicht möglich ist.

Aus diesem Grund wird eine FPGA-Logik entwickelt, die über eine USB-Schnittstelle Daten von einem Rechner entgegennimmt und diese in den Flashspeicher schreibt. Die Entwicklung dieses Moduls erfolgt in Verilog, da kein Modell des Flashspeicherbausteins vorliegt und Simulink daher nicht zur Verifikation des Moduls verwendet werden kann.

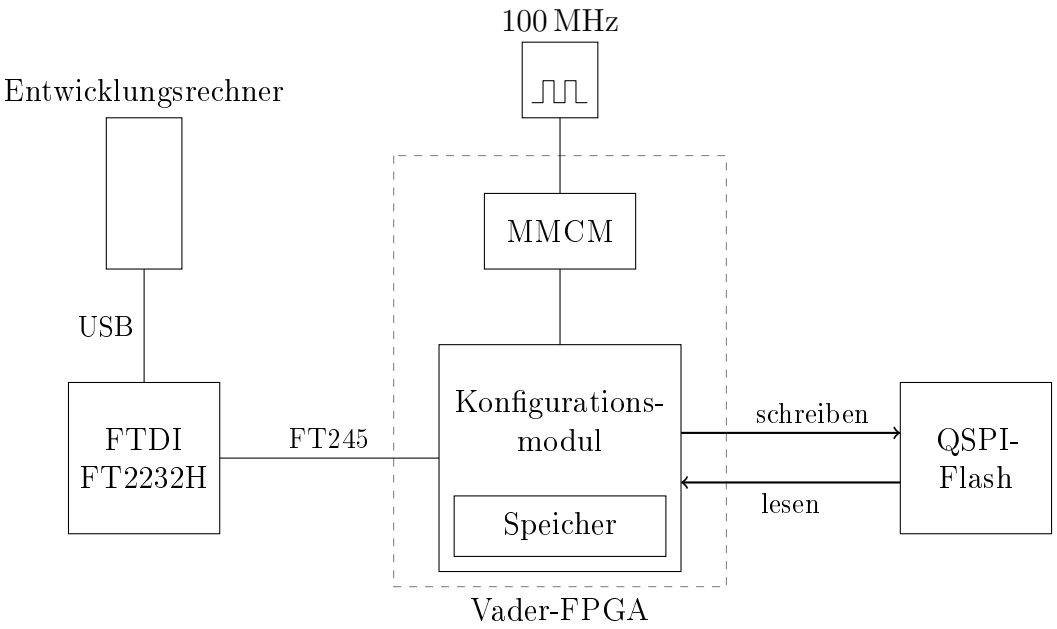


Abbildung 4.27: Funktionsprinzip des Konfigurationsmoduls

Da das Konfigurationsmodul nicht mit der Logik der Ortsfrequenzfilterung interagiert und somit keine Taktgrenzen zum Kameratakt überschritten werden müssen, wird der Takt auf Basis des 100-MHz-Oszillators auf der Vader-Platine abgeleitet. Dies hat den Vorteil, dass der Konfigurationsspeicher auch ohne angeschlossene Kamera beschrieben werden kann.

Der Entwicklungsrechner ist über eine USB-Verbindung mit der Vader-Platine verbunden. Dort stellt ein FTDI FT2232H USB-IC ein asynchrones First-in-First-out Interface ([37]) zur Kommunikation über die USB-Schnittstelle bereit. [3]

Der Rechner nutzt den entsprechenden FTDI-Treiber um jeweils 256-Byte-Blöcke des Bitstreams zu übertragen, die im Konfigurationsmodul zwischengespeichert und anschließend in den QSPI-Flashspeicher geschrieben werden. Die Daten werden zurückgelesen und mit den ursprünglichen Daten verglichen. Anschließend wird eine Erfolgs- oder Fehlermeldung an den Entwicklungsrechner gesendet, der mit dem nächsten 256-Byte-Block fortfahren kann.

Verifikation

Zur Verifikation des Konfigurationsmoduls wird ein Bitstream über das Flash-Programm in den Speicher geschrieben. Anschließend wird die Spannungsversorgung getrennt, neu verbunden und der Erfolg des Bootvorgangs über eine LED geprüft.

Der Flash-Vorgang dauert etwa 25 s, der Bootvorgang etwa 2 s.

4.3 DSP

Dieser Abschnitt behandelt die Entwicklung der DSP-Harness sowie eines Verifikationsmodells für die Schnittstellen zwischen dem DSP-Simulinkmodell und dem FPGA-Modell.

Texas Instruments stellt für den verwendeten 66AK2G12 DSP-SoC ein Echtzeitbetriebssystem (TI-RTOS) mit Treibern für die Peripheriemodule und einer Taskverwaltung zur Verfügung. [38]

Die Harness wird mit dem Ziel entwickelt, das DSP-Simulinkmodell über das TI-RTOS an die Hardware anzubinden. Abbildung 4.28 zeigt den Datenfluss durch die Treiber, das Echtzeitbetriebssystem, die Harness und das Simulinkmodell.

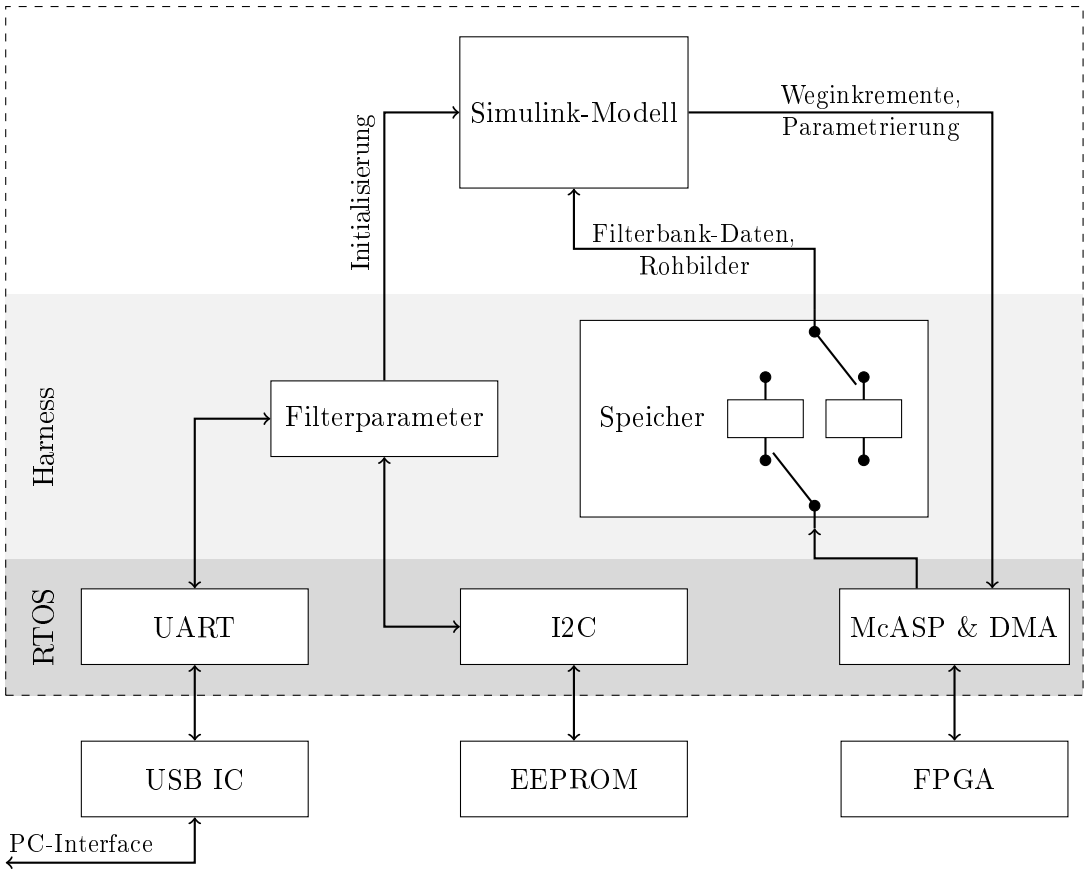


Abbildung 4.28: Funktionsprinzip der DSP-Harness

4.3.1 DSP-Modell

Abbildung 4.29 zeigt das Simulink-Modell der Daten-Schnittstellen zwischen DSP und FPGA.

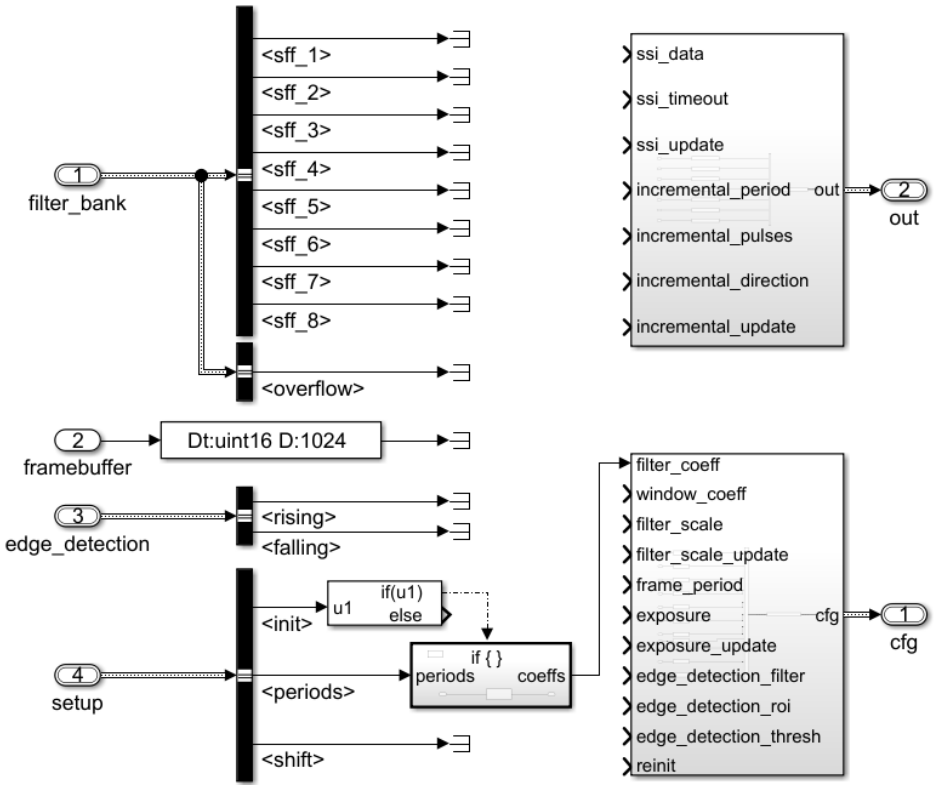


Abbildung 4.29: DSP-Modell mit Schnittstellen

Die Harness lädt während der Initialisierung die acht Filterperioden von einem externen EEPROM.

Anschließend wird der Modellschritt des DSP-Modells ein mal ausgeführt, dabei der init-Modelleingang gesetzt und die Filterperioden als Vektor übergeben. In diesem initialen Modellschritt berechnet das Simulinkmodell die Filterkoeffizienten aus den Perioden und gibt sie zurück. Neben den Filterkoeffizienten müssen auch die anderen Parametrierungsdaten, die im Konfigurationsmodus des FPGA gesetzt werden müssen (siehe Abschnitt 4.1.11), zurückgegeben werden. Im Initialisierungsmodellschritt liegen noch keine gültigen Filterbank- oder Rohbilddaten an.

Die Harness generiert die entsprechenden Befehle für die Parametrierung des FPGA aus den zurückgegebenen Daten und überträgt sie an das FPGA. Sobald das FPGA parametriert ist, werden n_{batch} Einträge der Ortsfrequenzfilterfunktionen und das zugehörige Rohbild zwischengespeichert und der Modellschritt (ohne

das Setzen des init-Modelleingangs) ausgeführt.

Das DSP-Modell kann in jedem Modellschritt die Änderung aller Parameter, die außerhalb des Konfigurationsmodus einstellbar sind, bewirken. Dazu muss die entsprechende Datenleitung belegt und das zugehörige Update-Signal auf true gesetzt werden. Um die Filterskalierungen zu verändern, muss beispielsweise ein Vektor mit acht Elementen, der die Skalierungsfaktoren der einzelnen Filter enthält, an filter_scale angelegt und filter_scale_update auf true gesetzt werden.

Die Harness generiert dann vor dem nächsten Modellschritt den entsprechenden Befehl und überträgt ihn an das FPGA.

4.3.2 Datenschnittstelle zum FPGA

Für die zu empfangenen Daten werden zwei Speicher angelegt, die jeweils n_{batch} Einträge der Filterbank-Daten sowie ein Rohbild enthalten. Wie in Abbildung 4.28 dargestellt, werden jeweils Daten in einen Speicher geschrieben während das DSP-Modell den anderen Speicher verarbeitet.

Es bleibt eine Zeitspanne $\frac{1}{f_{\text{batch}}}$ abzüglich der Zeit, die die Harness zum Verarbeiten der ausgehenden Befehle und der Verifikation der Prüfsummen und Framenummern benötigt, zum Verarbeiten der Batch. Die Verifikation der Batchdaten kann deaktiviert werden, wenn der Modellschritt mehr Zeit benötigt. Während der Testphase wurden keine Übertragungsfehler festgestellt.

4.3.3 PC-Interface

Um das Ortsfrequenzfiltersystem auf unterschiedliche Messsituationen anpassen zu können, kann ein Entwicklungsrechner über USB mit dem DSP-Board verbunden werden. Auf der Platine ist ein USB-UART-IC verbaut, über den die Harness mit dem Rechner kommuniziert.

Bei Benutzung des PC-Interfaces wird der Algorithmus angehalten und der Modellschritt nicht weiter ausgeführt. Vom Entwicklungsrechner können Rohbilder der Kamera abgefragt werden, um die Ausrichtung der Kamera zu prüfen. Außerdem können die Filterperioden und -verschiebungen in das EEPROM geschrieben werden.

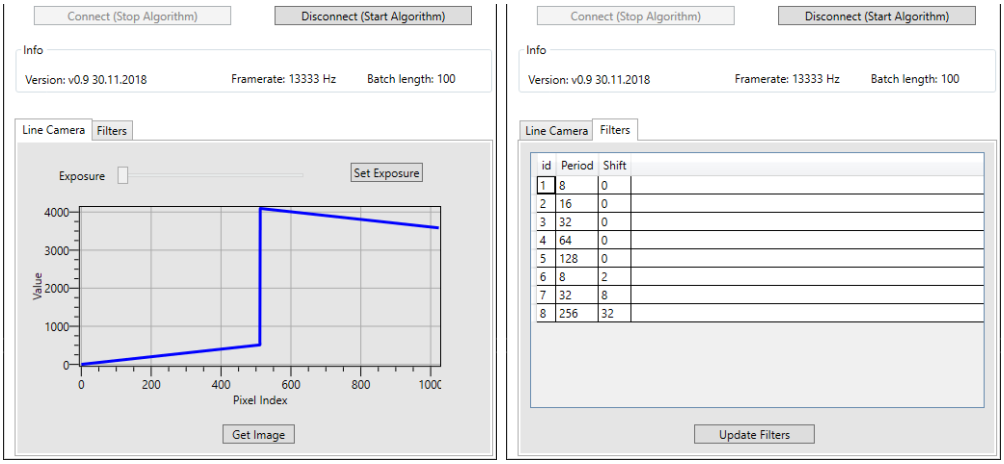


Abbildung 4.30: Abfragen eines Zeilenbilds und Setzen der Filterperioden über das PC-Interface

5 Verifikation

5.1 Modellbasierte Verifikation

Als Referenz für die modellbasierte Verifikation wird der Ortsfrequenzalgorithmus in Matlab implementiert.

Die Implementierung wird einer Plausibilitätsprüfung unterzogen indem ein Testbild erzeugt, mit bekannter Geschwindigkeit verschoben und mit der Matlab-Implementierung ausgewertet wird. Für die acht Filter werden verschiedene Perioden und Verschiebungen gewählt.

Abbildung 5.1 zeigt beispielhaft einen Ausschnitt eines Testbilds, in dem die Pixel mit der Geschwindigkeit 10 Pixel pro Zeile nach rechts verschoben werden.

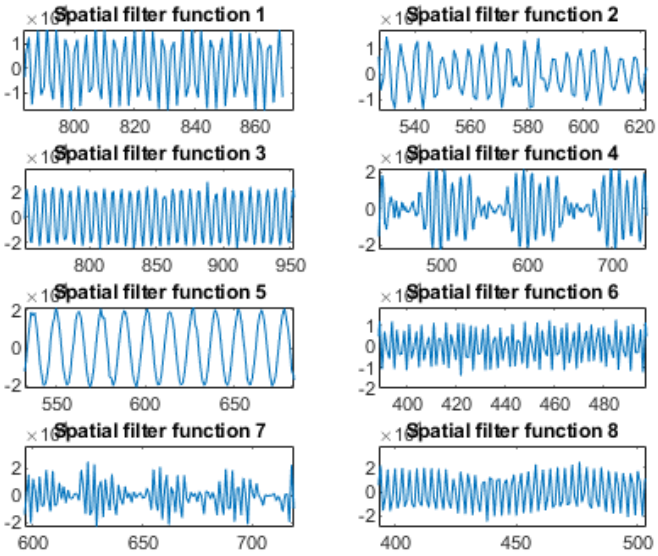
Abbildung 5.1: Testbild aus mehreren Testzeilen mit hellen Pixeln auf dunklem Untergrund

Das Ergebnis der Auswertung von 1000 Zeilen des gezeigten Testbilds ist in Abbildung 5.2 dargestellt. Zur besseren Darstellung der Signalformen wurden einzelne Abschnitte der einzelnen Ortsfrequenzfilterfunktionen vergrößert.

Aus der FFT der Signale wird jeweils unter Berücksichtigung der Filterperiode und -verschiebung der Index mit dem höchsten Leistungsanteil in eine Geschwindigkeit umgerechnet und die Ergebnisse mit der ursprünglichen Verschiebungsgeschwindigkeit verglichen.

Es wird beobachtet, dass sich in etwa 80% der Fälle Fehler von unter einem Prozent ergeben. Die teilweise größeren Fehler könnten auf die einfache aber un-

vorteilhafte Berechnung der Geschwindigkeit auf Grundlage des Maximums des Leistungsspektrums zurückzuführen sein [2, S. 51].




```
Filter 1: Max mag(fft()) @ 10.016000 Pixel/Frame
Filter 2: Max mag(fft()) @ 9.840000 Pixel/Frame
Filter 3: Max mag(fft()) @ 9.984000 Pixel/Frame
Filter 4: Max mag(fft()) @ 10.272000 Pixel/Frame
Filter 5: Max mag(fft()) @ 9.984000 Pixel/Frame
Filter 6: Max mag(fft()) @ 9.984000 Pixel/Frame
Filter 7: Max mag(fft()) @ 11.040000 Pixel/Frame
Filter 8: Max mag(fft()) @ 9.984000 Pixel/Frame
>>
```

Abbildung 5.2: Ergebnis der Auswertung des Testbildes mit acht verschiedenen Filtern

Auf Grundlage der Plausibilitätsprüfung wird angenommen, dass die entwickelte Matlab-Implementation des Algorithmus korrekt ist und als Referenz für die Verifikation des Simulink-Modells verwendet werden kann.

Aus dem Vader-Modell wird eine Test-Harness erzeugt, deren Struktur in Abbildung 5.3 gezeigt ist.

Das Testbild wird in einen Camera Link-Datenstrom übersetzt und über das entwickelte Modell der Spyder 3-Kamera als Eingang des Vader-Modells verwendet. Die Konfigurationsphase des DSP wird mit den im ersten Schritt gewählten Filterperioden und -verschiebungen sowie den zugehörigen Fensterfunktionen model-

liert und über die Modelle der McASP-Sender mit dem Vader-Modell verbunden. Die Ausgänge des Modells werden über McASP-Empfänger dekodiert und mit den Referenzwerten aus der Matlab-Implementation verglichen.

Dieses Vorgehen hat den Vorteil, dass das gesamte Modell inklusive des Kamera-Dekoders, der McASP-Schnittstellen und des Konfigurationsmodus simuliert und verifiziert wird.

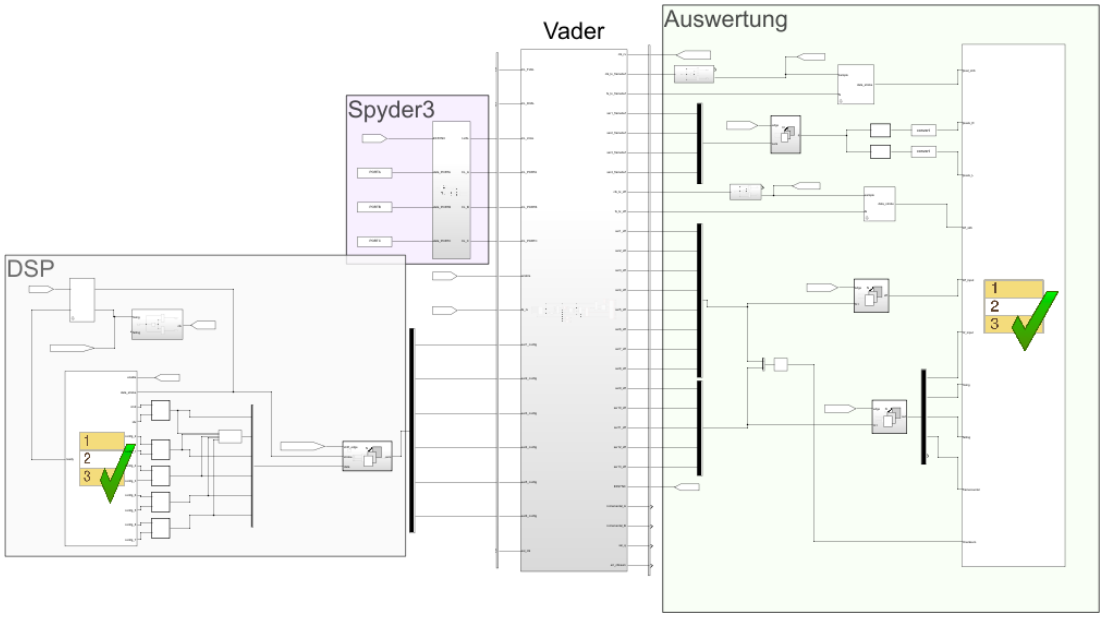
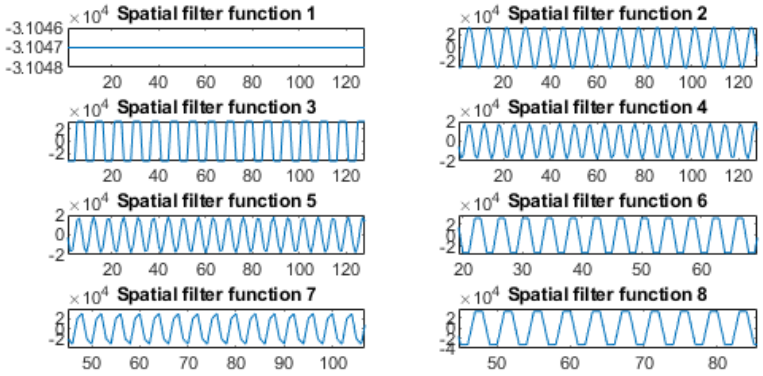


Abbildung 5.3: Verifikationsharness für das Gesamtmodell

5.2 Verifikation auf der Hardware

Um das FPGA-Modell und das DSP-Modell auf der Hardware zu verifizieren, wird das in Abbildung 5.1 gezeigte Testbild der Spyder 3-Kamera als Referenz gewählt. Durch dieses Vorgehen liegen bekannte Daten am Anfang der Signalverarbeitungskette an und die Ergebnisse können mit Erwartungswerten aus der Simulation verglichen werden.

Da das Testbild der Kamera prinzipbedingt stillsteht, ergibt sich mit stillstehenden Filtern ein reines Gleichsignal. Aus diesem Grund werden für diesen Test die Filter 2-8 jeweils mit unterschiedlichen Verschiebungen konfiguriert.




```
Filter 1: Max mag(fft()) @ 0.000000 Pixel/Frame
Filter 2: Max mag(fft()) @ 0.000000 Pixel/Frame
Filter 3: Max mag(fft()) @ 0.000000 Pixel/Frame
Filter 4: Max mag(fft()) @ 0.000000 Pixel/Frame
Filter 5: Max mag(fft()) @ 0.000000 Pixel/Frame
Filter 6: Max mag(fft()) @ 0.000000 Pixel/Frame
Filter 7: Max mag(fft()) @ 0.000000 Pixel/Frame
Filter 8: Max mag(fft()) @ 0.000000 Pixel/Frame
>>
```

Abbildung 5.4: Referenzdaten für die Hardware-Verifikation

Das Grundgerüst des DSP-Simulinkmodells aus Abbildung 4.29 wird um Blöcke zur Verifikation der Eingänge erweitert. Anschließend wird C-Code aus dem Modell generiert und über die Harness ausgeführt. Über den Verifikationsausgang des Modells wird die Übereinstimmung der empfangenen Daten mit den Erwartungswerten verifiziert.

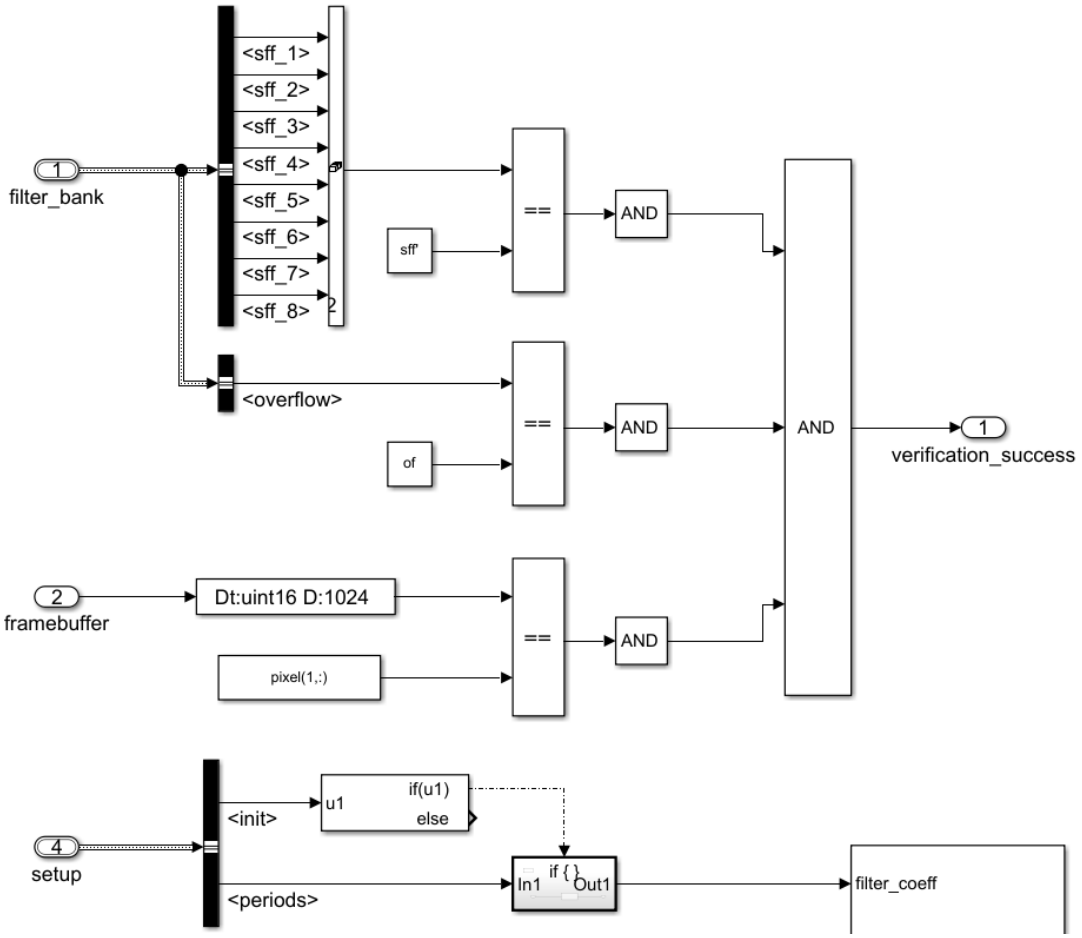


Abbildung 5.5: Simulink-Modell für die Verifikation auf der Hardware

6 Fazit

Die vorliegende Arbeit befasst sich mit einem Ortsfrequenzfiltersystem zur Messung von Bewegungsgeschwindigkeiten. Es wurde modellbasiert eine FPGA-Logik zur Verarbeitung von Zeilenkamerabildern entwickelt und verifiziert. Das FPGA ist über verschiedene Schnittstellen mit anderen Systemkomponenten verbunden. Dazu zählt ein digitaler Signalprozessor, auf dem mit diesem Projekt die Grundlage für die modellbasierte Entwicklung eines Auswertungsalgorithmus für die Ortsfrequenzfiltersignale gelegt wurde.

Sowohl die verschiedenen Filtermodule als auch die Schnittstellen zwischen den Systemkomponenten wurden erfolgreich in Simulink modelliert und für die HDL-Codegenerierung verwendet. Dazu wurde bereits in der Planungsphase die Systemarchitektur so konstruiert, dass sie möglichst gut in Simulink modelliert werden kann und keine Behandlung auf niedrigerer Abstraktionsstufe erfordert.

Insbesondere die Verifikation der verschiedenen Hoch- und Tiefpassfilter sowie der Ortsfrequenzfilter selbst konnte von den Möglichkeiten der grafischen Darstellung in Matlab/Simulink profitieren. Umgekehrt zeigte sich aber auch, dass die Modellierung von IO-Schnittstellen in Simulink zwar möglich, allerdings zeitaufwendig ist und wenig von den Vorteilen der Software profitiert.

Neben der modellbasierten Entwicklung des Auswertungsalgorithmus für den Signalprozessor kann ein nächster sinnvoller Schritt in dem Versuch bestehen, die Komplexität der Systemarchitektur zu verringern, indem die gesamte Signalverarbeitungskette auf einer einzelnen Platine oder sogar in einem einzelnen IC implementiert wird.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] Martin Schaeper. *Mehrdimensionale Ortsfiltertechnik*. Springer Vieweg, 2014.

[2] Arno Bergmann. »Verbesserung des Ortsfrequenzfilterverfahrens zur kamerabasierten Messung der translatorischen Geschwindigkeit bewegter Objekte«. Dissertation. Ruhr-Universität Bochum, 2010.

[3] Sean William Dalton. »Design and Implementation of High-Speed Electronics for a Spatial Filter Velocimeter«. Masterarbeit. Hochschule Bochum - Bochum University of Applied Sciences, 2018.

[4] Winfried Gehrke u. a. *Digitaltechnik*. 7. Auflage. Springer Vieweg, 2016.

[5] Klaus Fricke. *Digitaltechnik*. 8. Auflage. Springer Vieweg, 2018.

[6] Ralf Gessler. *Entwicklung Eingebetteter Systeme*. Springer Vieweg, 2014.

[7] Sean Dalton. *SFV VADER Lastenheft Version 1.0*. Hochschule Bochum. 2018.

[8] *UG471 v1.10 - 7 Series FPGAs SelectIO Resources*. Xilinx, Inc. 2018.

[9] *UG472 v1.14 - 7 Series FPGAs Clocking Resources*. Xilinx, Inc. 2018.

[10] *UG474 v1.8 - 7 Series FPGAs Configurable Logic Block*. Xilinx, Inc. 2016.

[11] *UG473 v1.12 - 7 Series FPGAs Memory Resources*. Xilinx, Inc. 2016.

[12] *DS180 v2.6 - 7 Series FPGAs Data Sheet: Overview*. Xilinx, Inc. 2018.

[13] Roland Woitowitz, Klaus Urbanski und Winfried Gehrke. *Digitaltechnik*. 6. Auflage. Springer, 2012.

[14] *DS181 v1.25 - Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics*. Xilinx, Inc. 2018.

[15] *UG903 v2012.2 - Vivado Design Suite User Guide Using Constraints.* Xilinx, Inc. 2012.

[16] Mark Balch. *Complete Digital Design.* McGraw-Hill, 2003.

[17] Neil H. E. Weste und David Money Harris. *CMOS VLSI Design - A circuits and systems perspective.* Fourth Edition. Addison-Wesley, 2011.

[18] *UG892 - Vivado Design Suite User Guide Design Flows Overview.* Xilinx, Inc. 2013.

[19] *HDL Coder User's Guide R2018b.* The MathWorks, Inc. `https://de.mathworks.com/help/pdf_doc/hdlcoder/hdlcoder_ug.pdf`, 2018.

[20] *Simulink User's Guide R2018b.* The MathWorks, Inc. `https://www.mathworks.com/help/pdf_doc/simulink/sl_using.pdf`, 2018.

[21] *UG953 - Vivado Design Suite 7 Series FPGA Libraries Guide.* Xilinx, Inc. 2012.

[22] Felix Schneider. *SFV VADER Datenerfassung und -vorverarbeitung Lastenheft Version 1.0.* Hochschule Bochum. 2018.

[23] *SPRUHY8H - 66AK2G1x Multicore DSP+ARM KeyStone II System-on-Chip (SoC) Technical Reference Manual.* Texas Instruments. 2018.

[24] Ulrich Tietze, Christoph Schenk und Eberhard Gamm. *Halbleiter-Schaltungstechnik.* 15. Auflage. Springer Vieweg, 2016.

[25] *SPRUGH7 - TMS320C66x DSP CPU and Instruction Set Reference Guide.* Texas Instruments. 2010.

[26] *Simulink Coder Reference R2018b.* The MathWorks, Inc. `https://de.mathworks.com/help/pdf_doc/rtw/rtw_ref.pdf`, 2018.

[27] *UG479 - 7 Series DSP48E1 Slice User Guide.* Xilinx, Inc. 2018.

[28] *Spyder 3 Camera Link User's Manual.* DALSA. 2008.

[29] *Camera Link - Specifications of the Camera Link Interface Standard for Digital Cameras and Frame Grabbers.* PULNiX America, Inc. 2010.

[30] *UG949 - UltraFast Design Methodology Guide for the Vivado Design Suite.* Xilinx, Inc. 2018.

[31] *SNLS056G - DS90CR287/DS90CR288A +3.3V Rising Edge Data Strobe LVDS 28-Bit Channel Link - 85MHz*. Texas Instruments. 2013.

[32] Uwe Meyer-Baese. *Digital Signal Processing with Field Programmable Gate Arrays*. Springer, 2001.

[33] *SPRSP07D - 66AK2G1x Multicore DSP+ARM KeyStone II System-on-Chip (SoC)*. Texas Instruments. 2018.

[34] Steven W. Smith. *Digital Signal Processing - A Practical Guide for Engineers and Scientists*. Newnes, 2003.

[35] Peter Meyer. *COVIDIS II Software Revision 3494*. INTACTON. 2010.

[36] *Implementation of SSI Master Interface Application Note*. POSITAL. 2013.

[37] *FT000061 - FT2232H Dual High Speed USB to Multipurpose UART/FIFO IC Datasheet Version 2.5*. FTDI.

[38] *SPRUHD4M - TI-RTOS 2.20 User's Guide*. Texas Instruments. 2016.

A Anhang

A.1 Inhalt Daten-CD

1 Dokumentation

lastenheft.docx

dokumentation.pdf

2 Projekt/

DSP/

FPGA/

Harness/

Vader/

EdgeDetectionVerif.m

Vader.slx

Vader_lib.slx

VaderParameters.m

Vader_Verification_calc.m

Vader_Verification_hw.m

VaderOpcode.m

PC/

fpga_flash

VaderControl