Programmierung von IEEE1394 Kameras in LabVIEW

Programmierung virtueller Instrumente in LabVIEW 8.5 zur Aufnahme von IEEE1394 Bilderframes unter Linux

Bachelorarbeit

Autor :	Arne Bruns
Studiengang :	Elektro-/Informationstechnik
Fachbereich :	Medieninformatik
1. Prüfer:	Prof. Dr. Bernd Stock
2. Prüfer:	Dr. Matthias Schröter
Firma:	Max-Planck-Institut für Dynamik und Selbstorganisation
Abgabe:	1. Februar 2012

HAWK
Fakultät Naturwissenschaften und Technik


NATIONAL INSTRUMENTS
LabVIEW

1 Inhaltsverzeichnis

2 Abbildungsverzeichnis

3 Einleitung und Aufgabenstellung

Das Max-Planck-Institut für Dynamik und Selbstorganisation erforscht hochkomplexe Systeme aus der Physik [Max11]. In vielen Versuchsreihen werden dabei die Ergebnisse durch die Aufnahmen von zeitgesteuerten Bilderserien unterstützend dokumentiert.

Bei solchen Versuchsreihen kommen sogenannte IEE1394 Kameras, die den sogenannten DCAM Standard unterliegen, zum Einsatz. Die IEEE1394 ist heutzutage vielleicht besser bekannt als FireWire. Der Begriff FireWire wurde von Apple eingeführt und bezeichnet lediglich das verwendete Bussystem [Wiki_Firewire]. Der DCAM Standard bestimmt dagegen das generelle Verhalten einer Kamera, die nicht komprimierte Bilder ohne Audioanteil liefert. Der auch als IIDC 13944-based Digital Camera Specification bekannte Standard ist in der Version 1.31 momentan aktuell [Wiki_DCAM]. Da die Bilder die Struktur oder die Anordnung mehrerer Objekte darstellen sollen, kommen sogenannte monochrome Kameras zum Einsatz. Bei Bildaufnahmen, die nur die Struktur eines Objektes darstellen sollen, ist eine monochrome Kamera rein technisch besser geeignet als eine Farbkamera. Da es dabei nur um die Analyse der unterschiedlichen Lichtintensitäten geht, eignet sich der von Natur aus farbblinde Sensor der monochromen Kamera besser dafür [Bässmann10].

Um eine zeitgesteuerte Aufnahme von Bildern zu realisieren, benutzt das Max-Planck-Institut die Programmierentwicklungsumgebung LabVIEW 8.5 von National Instruments in Linux. LabVIEW zeichnet sich besonders durch seine Benutzerfreundlichkeit aus. Das Konzept, welches LabVIEW dabei verfolgt, ist das Prinzip der grafischen Programmierung. So werden einzelne Programmierelemente auf eine Art Tafel per Drag & Drop gezogen und miteinander verbunden. Dadurch entsteht ein logischer Programmablaufplan [Wiki_LabVIEW].

Das Basispaket beinhaltet allerdings nur die Grundfunktionen der Programmierung. Für die industrielle Bildverarbeitung bietet National Instruments ein so genanntes NI Vision Development Modul an. Dieses Modul hat die Möglichkeit Bilder von verschiedenen Kameratypen (IEEE1394, USB, IP-Kameras, u.a.) zu empfangen. Zudem bietet es eine Vielzahl von Funktionen, die für die industrielle Bildverarbeitung relevant sind (Objekterkennung, Lokalisierung, Abmessungsbestimmung, u.a.) [Nat11].

Auf dieses Modul möchte das Max-Planck-Institut in Zukunft aus Kostengründen verzichten. Für die Zwecke des Max-Planck-Instituts reicht es aus, eine IEEE1394 Kamera einzustellen und einzelne Bilder abzuspeichern. So wurde im Vorfeld ein Programm entwickelt mit dem man die Möglichkeit hat, eine IEEE1394 Kamera einzustellen und die Einstellungen in einer XML Datei abspeichern kann [Bru11].

Mit dem sogenannten Code Interface Node bietet LabVIEW den Softwareentwicklern die Möglichkeit externe Codes in ein Programm einzubinden. Die Programmiersprache, die dabei verwendet wird, ist C [Nat031]. Diese Programmierschnittstelle soll benutzt werden, um die benötigten Funktionen (Kamera einstellen, Bild aufnehmen) in LabVIEW zu realisieren. Diese Funktionen sollen für eine einfache Wiederverwendbarkeit in sogenannten virtuellen Instrumenten abgespeichert werden.

Für die generelle Kommunikation mit IEEE1394 Kameras, die dem DCAM Standard unterliegen, gibt es die libdc1394- Library. Diese Bibliothek bietet den großen Vorteil, dass sie herstellerunabhängig ist und alle Möglichkeiten bietet, die Kamera nach den individuellen Anforderungen einzustellen [Douxchamps11].

Die Funktionalität der einzelnen virtuellen Instrumente sollen im Anschluss in einer Testanwendung überprüft werden. Die Testanwendung soll dem Benutzer zunächst einmal die Möglichkeit geben die Kamera am Laborplatz neu auszurichten (Darstellung des Kamerabildes). Des Weiteren soll es dem Benutzer darüber hinaus erlauben Kamerabilder fortlaufend abzuspeichern. Diese überprüfende Testanwendung soll belegen, dass man eine Kamera mit zwei unterschiedlichen Einstellungen betreiben kann. Für die Versuche des Max-Planck-Instituts ist die Aufnahme von Bildern mit unterschiedlichen Belichtungszeiten von besonderem Interesse. Eine detailliertere Beschreibung der Programmfunktionen befindet sich im Kapitel 5.1.

4 Grundlagen

4.1 IEEE 1394 Standard

Der IEEE 1394 Standard ist heutzutage wohl besser bekannt als FireWire (Apple) oder I.LINK (Sony) Standard. Hinter diesem Standard verbirgt sich ein BUS System, welches die Kommunikation der an diesem Bus angeschlossen Geräte regelt. An diesem Bus können insgesamt bis zu 63 Geräte angeschlossen werden, wobei eine Verbindung zwischen 2 Geräten nicht mehr als über 16 Verbindungen gehen darf [ElekKomp10].

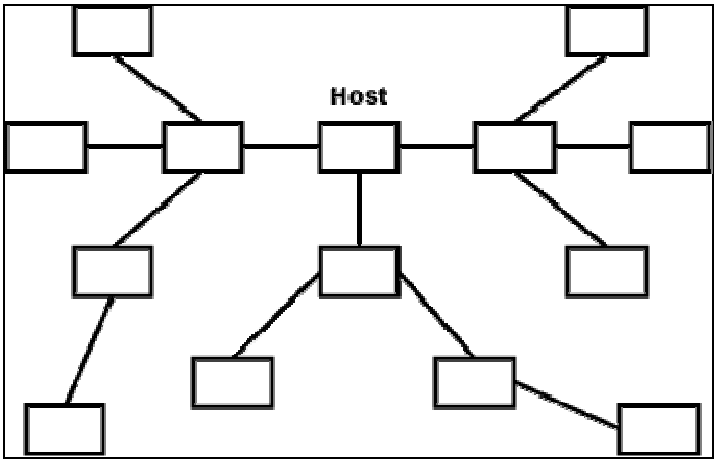

Abbildung 1 physikalischer Aufbau eines IEEE 1394-Bus. Quelle: [ElekKomp10]

In dem obigen Schaubild erkennt man sehr gut wie ein IEEE 1394 Bus physikalisch aufgebaut werden kann. IEEE 1394 Geräte weisen häufig zwei Anschlüsse auf, die diese Struktur erlauben. Dabei handelt es sich um eine Baumstruktur, in dem das zentrale Geräte der Host ist. Dieser Host bestimmt die Adressierung, Taktung und die Kommunikation der einzelnen Geräte. Anders als bei USB, benötigt IEE1394 keinen zentralen Controller, da jedes an dem Bus angeschlossene Gerät diese Funktion übernehmen kann [ElekKomp10].

Angeschlossene Geräte können über den 6 poligen IEEE Port mit Strom versorgt werden. Dabei liegt die Stromstärke bei 1,5 Ampere und die Spannung zwischen 8 und 30 Volt (in der Regel 12 Volt) [ElekKomp10].

Im Gegensatz zum USB Bus bietet IEEE 1394 neben dem asynchronen Übertragungsmodus noch den isochronen. Im Gegensatz zum asynchronen Übertragungsmodus wird beim isochronen Übertragungsmodus jedem angeschlossenen Gerät eine gewisse Bandbreite garantiert, was die

Grundvoraussetzung für Real-time-Anwendungen ist. Dieser isochrone Übertragungsmodus ist besonders in der Industrie gefragt [ElekKomp10].

Name	IEEE 1394a	IEEE1394b	IEEE 1394-2008
Apple Bezeichnung	FireWire S100, S200, S400	FreWire S800	FireWire S3200
Geschwindigkeit in Mbit/s	100, 200, 400	800	3200
Geschwindigkeit in MByte/s	12, 25, 50	100	400

Abbildung 2 Kennwerte heutiger IEEE 1394 Standards [ITHandbuch01]

Aus der Tabelle kann man die bis heute veröffentlichen Standards mit ihren möglichen Übertragungsraten ablesen [ITHandbuch01]. Hierbei ist zu beachten, dass für die drei unterschiedlichen Standards auch unterschiedliche Kabel verwendet werden.

4.2 IEEE 1394 Kameras

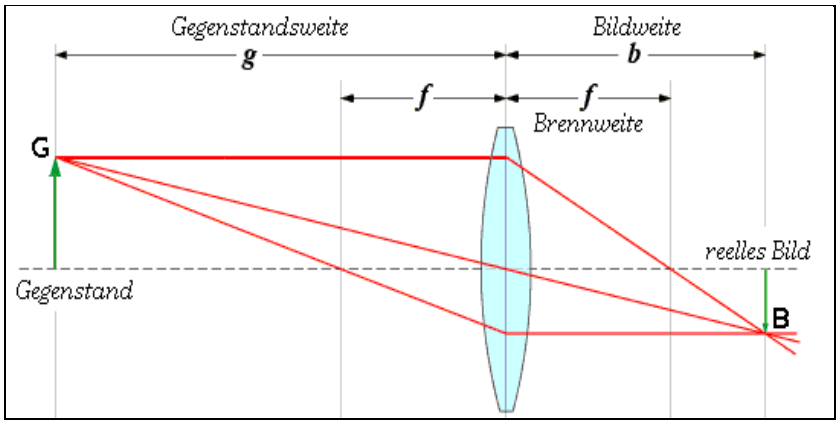

Abbildung 3 Darstellung der Funktion einer Sammellinse. Quelle: [Wiki_Linse]

Die Abbildung 2 stellt die Funktionsweise einer Kamera da. Das reflektierte Licht wird von einem Gegenstand über die Linse auf ein sogenanntes „reelles Bild“ projiziert. Das projizierte Bild wird dann mittels eines CMOS oder CCD Sensors digitalisiert. Da in der der Praxis die Bildweite häufig unbekannt ist, kann unter Anwendung der Formel, die richtige Linse ermittelt werden. So kann man mittels der Sensorhöhe, B, der Gegenstandshöhe, G, und der Gegenstandsweite, g, die benötigte Brennweite, f, berechnen [Wiki_Linse].

$$f = \frac{B}{(B+G)} * g$$

$$Brennweite = \frac{Gegenstandshöhe}{Gegenstanshöhe + Sensorhöhe} * Gegenstandsweite$$

Quelle: [Wiki_Linse]

In der Industrie werden heutzutage immer noch schwarz-weiß Kameras bevorzugt eingesetzt. Dies liegt zum einen daran, dass die Bildsensoren (CMOS oder CCD) von Natur aus farbenblind sind. Zum anderen, weil Industrieapplikationen häufig nur mit Kontrastunterschieden arbeiten und dabei die Farbe irrelevant ist. Eine schwarz-weiß Kamera nennt man auch monochrome Kamera [Bässmann10].

Für IEEE 1394 Kameras gibt es den sogenannten „IIDC 1394-based Digital Camera Specifikation“ Standard, der auch als DCAM Standard bekannt ist. Dieser Standard legt das generelle Verhalten von Kameras, die nicht komprimierte Bilder ohne Audioanteil über die IEEE1394 ausgeben, fest. Dabei kann es sich sowohl um eine monochrome als auch um eine Farbkamera handeln. Der Standard mit der Version 1.31 wurde im Jahr 2004 von der 1394 Trade Association verabschiedet [1394TA08].

In der Industrie werden aufgrund des isochronen Datenübertragungsmodus IEEE 1394 Kameras bevorzugt eingesetzt. Diese FireWire Kameras werden im Allgemeinen ohne Linse ausgeliefert. Die Linsen werden dann üblicherweise mittels einer sogenannten C-mount Steckverbindung mit der Kamera verbunden [Bässmann10].

4.3 LabVIEW

Die Firma National Instruments bietet ihren Kunden die Möglichkeit, Anwendungen im Bereich Prüf-, Mess-, Steuer, Regel- und Embedded-Design effizient zu entwickeln. Dabei stellt National Instruments neben der Software auch die Hardware zur Verfügung. Aus den gleichbleibenden Komponenten (Hard- und Software) entsteht ein Gesamtpaket, mit dem Ingenieure und Wissenschaftler Software vom Design bis hin zur Serienreife entwickeln können [Nat111].

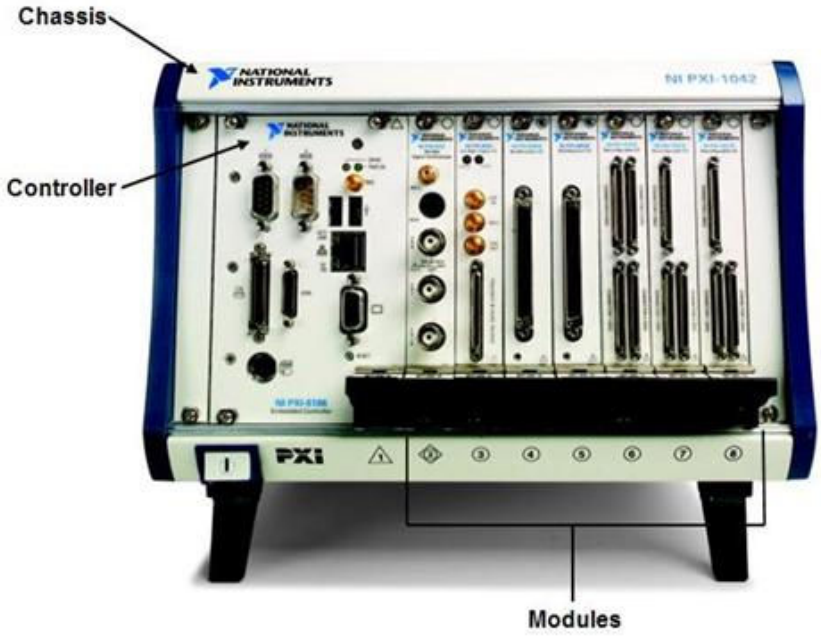

Abbildung 4 PXI System (Controller, Chasis, Modules)

Als Hardwaregrundlage empfiehlt National Instruments den Einsatz von PXI Systemen. Diese PXI Systeme bestehen aus Gehäuse, Controller und Modulen. Das Gehäuse stellt dabei die Verbindung zwischen den Controller und den einzelnen Modulen dar. So sind alle Komponenten über die Schnittstelle PXI verbunden [Nat112].

Die Abbildung 4 zeigt ein solches PXI System. Auf der linken Seite sieht man den Controller. Dieser Controller ist in der Regel ein auf einer Intel CPU basierendes System, welches über die PXI Schnittstelle mit dem Gehäuse verbunden wird. Die Controller besitzen in der Regel Anschlüsse, die auch ein normaler Computer besitzt (USB, VGA, LAN).

Das System lässt sich mit sogenannten Modulen erweitern und so auf die individuellen Zwecke des Benutzers anpassen. Die einzelnen Module werden über die Moduleinschübe mit dem Gehäuse über die PXI Schnittstelle verbunden. Die einzelnen Komponenten sind nun alle über die PXI Schnittstelle miteinander verbunden.

Für die Entwicklung von Anwendungen stellt National Instruments seine Systementwicklungsumgebung namens LabVIEW bereit. LabVIEW steht dabei für „Laboratory Virtual Instrument Engineering Workbench“ [Wiki_LabVIEW].

Diese Entwicklungsumgebung verfolgt dabei das Konzept der grafischen Programmierung und lässt sich in die Bereiche User Interface und logischen

Programmablaufplan untergliedern. Die Porgrammiersprache, die dabei eingesetzt wird, nennt man „G“ und die Programmierung erfolgt dabei nach dem Datenfluss-Modell. Die Ebene des logischen Programmablaufplans nennt man auch Blockschaltbildebene. Dabei werden Programmierelemente als Blockschaltbilder dargestellt und können mittels Drag und Drop auf das Blockschaltbild gezogen werden. Für das richtige Zusammenspiel der einzelnen Elemente werden diese auf dem Blockschaltbild miteinander verdrahtet. Auf der Ebene des User Interfaces bestimmt der Entwickler wie die einzelnen Ein- und Ausgabeelemente angeordnet werden sollen. Die Entwicklungsumgebung LabVIEW lässt sich auch ohne die von National Instruments angebotenen Hardware betreiben, so dass man auch Anwendungen mit LabVIEW entwickeln kann ohne die kostspielige Hardware zu besitzen [Wiki_LabVIEW].

VI steht für „Virtuell Instrument“ und stellt das entwickelte Hauptprogramm dar. Um die Übersicht in größeren Programmen zu bewahren, sollten Softwareentwickler ihr Programm in logische Abschnitte unterteilen und daraus einzelne virtuelle Instrumente erzeugen. Die Unterteilung eignet sich besonders gut für die Implementierung von Algorithmen.

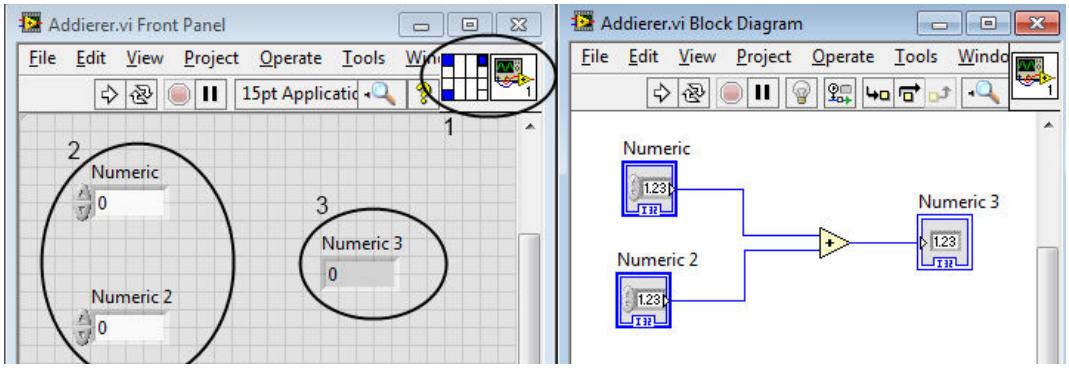

Abbildung 5 Beispiel eines Addierers als SubVI

Die Abbildung 5 zeigt einen einfach programmierten Addierer in LabVIEW. Auf der rechten Seite sieht man die logische Struktur des Programms. Dabei werden die beiden ganzzahligen Variablen „Numeric“ und „Numeric 2“ addiert und in der Zahl „Numeric 3“ abgespeichert. Auf der linken Seite sieht man das Frontpanel Design. Der Kreis mit der Nummer 2 bildet die Eingabemöglichkeiten des Benutzers ab, während Kreis Nummer 3 die Ergebnisausgabe darstellt. Um aus diesen VI ein SubVI zu erstellen, muss der Entwickler die Ein- und Ausgänge des Programms definieren. Dabei verdrahtet der Entwickler die gewünschten Ein- und Ausgänge auf der Front Panel Ebene mit dem Verbindungfeld im Kreis1. Durch die Bestimmung

der Ein- und Ausgänge entsteht aus einem VI ein Sub VI, welches man dann in jedem beliebigen VI wiederverwenden kann [Wiki_LabVIEW].

LabVIEW bietet seinen Nutzern ein reichhaltiges Angebot an Funktionen. Für hardwarespezifische Funktionen sind häufig bei dem Erwerb die dazugehörigen Hardware Module integriert. Um dem Benutzer dennoch die Möglichkeit zu geben, eigene Routinen zu schreiben, bietet LabVIEW in der Version 8.5 dem Entwickler zwei Möglichkeiten an.

Die eine Möglichkeit ist das Einbinden über einen sogenannten Call Library Function Knoten. Mithilfe dieses Knotens lassen sich einzelne Funktionen einer Library aufrufen. Der Entwickler muss also seine gewünschten Routinen in eine selbst entwickelte Library schreiben, um sie später mithilfe des Call Library Knotens aufzurufen [Nat031].

Die andere Möglichkeit ist der Aufruf eines sogenannten Code Interface Nodes. Hierbei wird eine Routine in einem sogenannten LSB File gespeichert. Im Gegensatz zu der Call Library Function kann man nur eine Funktion in dem LSB File speichern [Nat031].

Beide Möglichkeiten sind sehr nahe beieinander. So verwenden beide Methoden die Programmiersprache „C“. Der Vorteil der Call Library Funktion ist, dass für eine spätere Weiterentwicklung nur die Library erneuert werden muss. Der Aufruf der Call Library Funktion in dem Virtuellen Instrument muss hingegen nicht mehr erneuert werden. Zudem ist der Weg der Call Library Funktion der, der in der modernen Softwareentwicklung in der Regel angewandt wird. Auch wenn die Anwendung über den Code Interface Node heute in der Regel nicht mehr angewendet wird, bietet er im Vergleich zum Aufruf der Call Library Function einige Vorteile. So benötigt das System nach der erfolgreichen Einbindung des Codes Interface Nodes in das VI das dazugehörige LSB File nicht mehr, da der in dem LSB File dargestellte Code bereits komplett im VI übernommen wurden ist [Nat031 S.17-19].

5 Grobkonzept

5.1 Programmfunktionalitäten

Abbildung 6 Darstellung einzelner Programmmodule

Die Abbildung 6 stellt noch einmal die geforderten Programmfunktionen dar. Zunächst einmal muss eine XML Datei ausgelesen werden, die die Informationen für die Einstellungen der Kamera beinhaltet. LabVIEW 8.6 unterstützt allerdings nicht das Auslesen von XML Dateien. Somit muss für diese Funktionalität ein externer Code verwendet werden. Anhand der Daten der XML Datei soll dann die Kamera eingestellt werden. Diese Einstellung beinhalten die Festsetzung des Videomodus sowie die einzelnen relevanten Kameraregler (Brightness, Gain, Shutter, u.a.). Dabei sind in der vorliegenden XML Datei alle von der Kamera unterstützten Reglereinstellungen gespeichert. Um ein gleichbleibendes Bild zu erzeugen, müssen alle Regler auf den Modus manuell geschaltet werden.

Um vor einer Messreihe überprüfen zu können, ob die Kamera funktionstätig und korrekt ausgerichtet ist, soll es die Möglichkeit geben, dem Benutzer eine Bildvorschau anzuzeigen.

Das letzte Modul ist die Bildaufnahme. Hier soll das von der Kamera gesendete Bild in ein gängiges Bildformat (jpg, png) abgespeichert werden können.

5.2 Bibliotheken

Name	Version	Funktionsbeschreibung	Lizenz	Verweis
Libdc1394	2.1.2	Kommunikation mit IEEE1394 Kameras	LGPL	[Douxchamps11]
openCV	2.3.1	Funktionen zur Bildverarbeitung	BSD	[openCVdevTeam11]
Libxml2	2.7.8	Verarbeitung von XML	MIT	[Dan11]

Abbildung 7 Übersicht der benötigten Programmbibliotheken

Die Abbildung 7 zeigt die benötigen Bibliotheken, die für die Realisierung des Programms notwendig sind.

Die Libdc1394 ist eine Bibliothek, die es einem ermöglicht, Kameras, die den sogenannten 1394-based Digital Camera Specifications unterliegen, zu kontrollieren. Sie ist in der Version 2.1.2 aktuell und unterliegt der Open Source Lizenz LGPL. Mithilfe dieser Library ist es möglich, mit fast jeder DCAM Kamera zu kommunizieren. Darüber hinaus verfügt die Kamera noch über Funktionen, mit denen es möglich ist die Bilddaten in verschiedene Formate umzuwandeln (Monochrom, YUV, RGB, ...) [Douxchamps11]. Bei diesem Projekt ist sie für das Einstellen der Kamera sowie die Übertragung des nicht komprimierten Kamerabildes verantwortlich.

Die Programmbibliothek openCV (CV = Computer Vision) beinhaltet eine Vielzahl von Algorithmen für die Bildverarbeitung. Somit stehen dem Benutzer mehr als 2500 optimierte Algorithmen zur Verfügung. Zudem ist die Aufnahme von USB und FireWire Kameras möglich. Die Bibliothek ist in der Version 2.3 für die Plattformen Windows, Linux, Mac und Android verfügbar [openCVdevTeam11]. Die Bibliothek soll später für das Abspeichern des Kamerabildes verantwortlich sein.

Die Libxml2 bietet sämtliche Funktionen zum Erstellen und zum Auslesen von XML Dateien. Viele Linux Distributionen haben diese Bibliothek bereits standardmäßig eingebunden [Dan11]. Diese Bibliothek bekommt die Aufgabe, die XML Datei zu öffnen und die für das Programm relevanten Informationen auszulesen.

Alle drei Lizenzen sind sowohl für den privaten als auch für den kommerziellen Gebrauch frei verfügbar, so dass bei einer späteren Veröffentlichung des Programms unter Mitveröffentlichung des Quellcodes keine Copyright Verletzung auftreten.

6 Feinkonzept

6.1 VI Design

Als nächstes wurde untersucht, wie man aus den einzelnen Programmfunktionalitäten die entsprechenden VIs designen kann. Dabei sollen die einzelnen VIs ein klares Ziel haben und nicht mit Ein- und Ausgängen überladen sein. Zudem sollte jedes VI über einen Ein- und Ausgang für Fehlermeldungen verfügen.

Besonders der Aspekt, dass man später möglichst wenig Ein- und Ausgänge haben möchte, führt dazu, dass man die Programmfunktionalität des Auslesens der XML Datei und das Einstellen der Kamera in ein einzelnes VI zusammenlegt.

```
<!DOCTYPE camset>

<firecam>
 <Date day="8" month="11"
year="2011">08.11.2011</Date>
 <caminfo model="XCD-SX910 v3.00E"
guid="5765377267080253 26" vendor="SONY"/>
 <video videomode="81" framerate="35"/>
 <FeaturesSettings>
  <feature fvalue="256" fid="0">Brightness</feature>
  <feature fvalue="100" fid="1">Auto
Exposure</feature>
  <feature fvalue="129" fid="7">Shutter</feature>
  <feature fvalue="384" fid="8">Gain</feature>
  <feature fvalue="0" fid="12">Trigger</feature>
  <feature fvalue="56" fid="17">Pan</feature>
  <feature fvalue="40" fid="18">Tilt</feature>
 </FeaturesSettings>
</firecam>
```

Abbildung 8 Aufbau einer XML Datei

Die Abbildung 8 zeigt den Aufbau einer XML Datei, die die Einstellungen einer IEEE1394 Kamera aufweist. Diese XML Datei lässt sich in 4 einzelne Blöcke unterteilen. Der Block „Date“ enthält das Datum, an dem die Datei erstellt worden ist. Der Block „caminfo“ beinhaltet Informationen über den Kameratyp. Hier werden der Name des Herstellers und das Model der Kamera angegeben, sowie die eindeutige Kameraidentifikationsnummer „guid“. Der Block „video“ beinhaltet Informationen über den einzustellenden Videomodus. Dabei beinhaltet das Attribut „videomode“ Informationen zur Auflösung sowie zur Farbkodierung. Das Attribut

„framerate“ beinhaltet Informationen zur Bildwiederholungsrate. Der Block „FeaturesSettings“ beinhaltet sämtliche von der Kamera unterstützten Features.

Abbildung 9 Konzeptioneller Ausbau des VIs zum Einstellen der Kamera

Die Abbildung 9 zeigt die Funktionalität des ersten VIs. Dieses VI soll die XML Datei auslesen und die Kamera gleichzeitig einstellen. Dabei bekommt das VI zunächst den Dateinamen der XML Datei. Das VI soll nun den Dateinamen an den Code Interface Node übergeben. Der CIN liest die Datei aus und stellt gleichzeitig die Kamera anhand der in der XML Datei angegebenen Parameter ein. Damit die anderen VIs nicht mehr die XML Datei öffnen müssen, wird die eindeutige Identifikationsnummer der Kamera vom Code Interface Node ausgegeben und an den Ausgang des VIs gesendet. Den Ein- und Ausgang des Fehlercodes kann man sich als einen Fehlercode BUS vorstellen. So wird am Anfang des Programms überprüft, ob zuvor ein Fehler aufgetreten ist. Ist dies der Fall, so wird das Programm abgebrochen und der Fehler an den Ausgang weitergeleitet. Tritt während der Ausführung des Programms ein Fehler auf, so wird dieser ebenfalls an den Ausgang weitergeleitet.

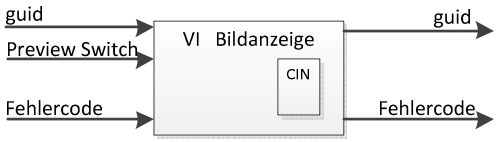

Abbildung 10 Konzeptioneller Aufbau des VIs zur Bildanzeige

Das VI zur Bildanzeige überprüft anhand des Fehlercodes zunächst, ob ein Fehler im Vorfeld aufgetreten ist. Ist dies der Fall, so wird der Fehler weitergeleitet und das Programm nicht weiter ausgeführt. Mit der Variablen „Preview Switch“ gibt der Benutzer an, ob eine Bildanzeige erwünscht ist. Ist diese Variable auf „TRUE“ geschaltet so wird das Programm gestartet und ein Bild von der Kamera mit der entsprechenden „guid“ aufgenommen. Das Bild wird lokal im temporären Ordner gespeichert. Danach wird das Bild mit dem Bildbetrachtungsprogramm „geeqie“ geöffnet und angezeigt.

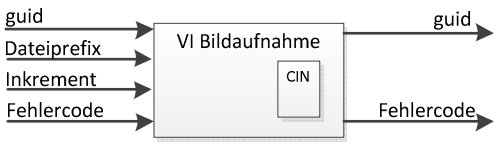

Abbildung 11 Konzeptioneller Aufbau des VIs zur Bildaufnahme

Genau wie bei den anderen beiden VIs wird zunächst der Fehlercode überprüft. Bei fehlerfreiem Aufruf des VIs wird zunächst ein Dateiname erstellt, der sich aus dem Dateiprefix und dem Inkrement Wert zusammensetzt. Dabei dient der Wert „Inkrement“ zur fortlaufenden Bildnummerierung. Der resultierende Dateiname und die „guid“ werden dann an den Code Interface Node übergeben. Der Code Interface Node sucht die der „guid“ entsprechende Kamera und nimmt ein einzelnes Bild auf. Dieses Bild wird dann abgespeichert und der Code Interface Node beendet. Tritt während der Ausführung des Code Interface Nodes ein Fehler auf, wird der entsprechende Fehler an den dafür entsprechenden Ausgang des VIs übermittelt.

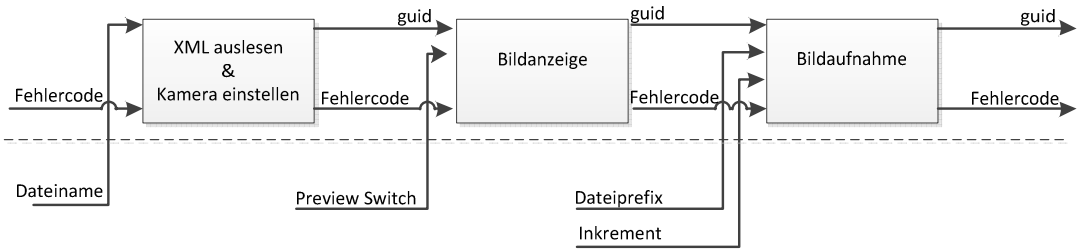

Abbildung 12 Mögliche Verbindung der resultierenden VIs

Die Abbildung 12 zeigt eine mögliche Verbindung der drei VIs. So werden die drei VIs über die Ein- und Ausgänge „guid“ miteinander verbunden und jeweils an den Fehlercode Bus angehängt. Der Benutzer muss letztendlich nur noch 3 Variablen bestimmen. Zum einen den variable „Dateiname“ der sich aus dem Namen der XML Datei ableitet. Zum anderen die beiden Variablen Dateiprefix und Inkrement, die letztendlich für die fortlaufende Nummerierung der Bilder bei der Bildaufnahme verantwortlich sind. Über den Schalter „Preview Switch“ schaltet der Benutzer die Bildvorschau ein und aus.

6.2 Error Codes

Um Fehler, die während der Ausführung der Code Interface Nodes auftreten können zu behandeln, müssen diese von dem Code Interface Node an einen Error Handler weitergegeben werden. Dabei ist der Entwickler für diese Weitergabe selbst

verantwortlich. LabVIEW bietet Entwicklern die Möglichkeit, eigene Fehlercodes zu bestimmen. Diese sogenannten Custom Error Codes können in einem Wertebereich von 5000- 9999 liegen und werden vom Entwickler in einer dafür vorgesehenen Datei gespeichert [Nat06].

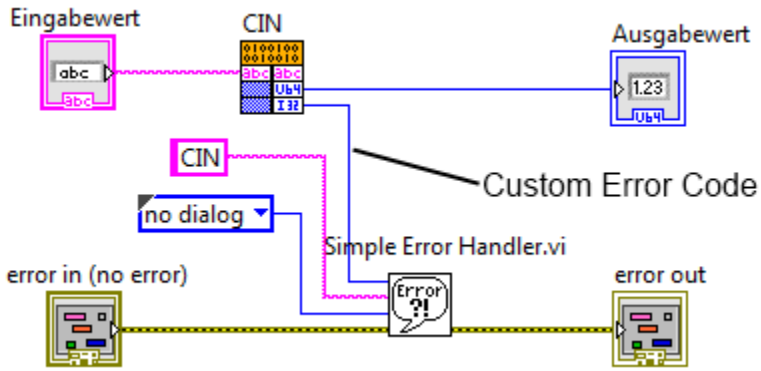

Abbildung 13 Beispiel zum Custom Error Handling

Die Abbildung 13 zeigt wie man beispielsweise einen bei der Ausführung eines Code Interface Nodes auftretenden Fehler behandelt. So verfügt der Code Interface Node über einen ganzahligen Ausgang, der den Wert des Errors beinhaltet. Ist dieser nicht null, so generiert der Simple Error Handler daraus eine Fehlermeldung, und leitet diese an die Variable „error out“ weiter. Im CIN ist der Entwickler dafür verantwortlich, dass ein Error Code weitergeleitet wird. Eine Liste der selbst definierten Error Codes befindet sich im Anhang. Der an den Simple Error Handler übergebene Wert „CIN“ ist eine Bezeichnung für die Quelle des Fehlers. Sie dient dazu, um den Benutzer darauf hinzuweisen an welcher Stelle seines VIs ein Fehler aufgetreten ist. Der übergebene Wert „no dialog“ gibt dem Simple Error Handler vor wie er einen auftretenden Fehler zu behandeln hat.

7 Implementation

7.1 VI „DC1394_xml“

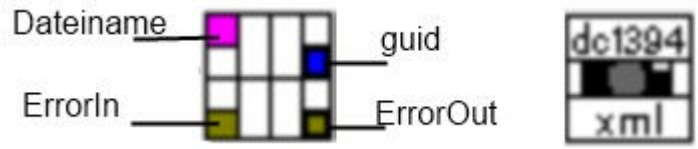

Abbildung 14 Symbole des VIs „DC1394_xml“

In der Abbildung 14 sieht man auf der rechten Seite das Symbol des VIs „DC1394_xml“. Das Symbol soll zeigen, dass es sich hauptsächlich, um ein VI handelt, welches die libdc1394 benutzt, um die Kamera anhand einer XML Datei einzustellen. Das Symbol ist bewusst in schwarz-weiß gehalten, um zu verdeutlichen, dass dieses VI nur für schwarz-weiß Kameras gedacht ist.

Auf der linken Seite sieht man die einzelnen Ein- und Ausgänge des VIs. So besitzt dieses VI einen Eingang vom Typ String. An diesem Eingang definiert der Benutzer den Namen der zu verwendenden XML Datei. „ErrorIn“ ist ein weiterer Eingang vom Typ Error, der zur Programmsteuerung im Fehlerfall dient. Tritt während der Ausführung des VIs ein Fehler auf, wird dieser an den Ausgang „ErrorOut“ gegeben. Bei erfolgreichem Auslesen der XML Datei gibt das Programm die Identifikationsnummer „guid“ der Kamera weiter.

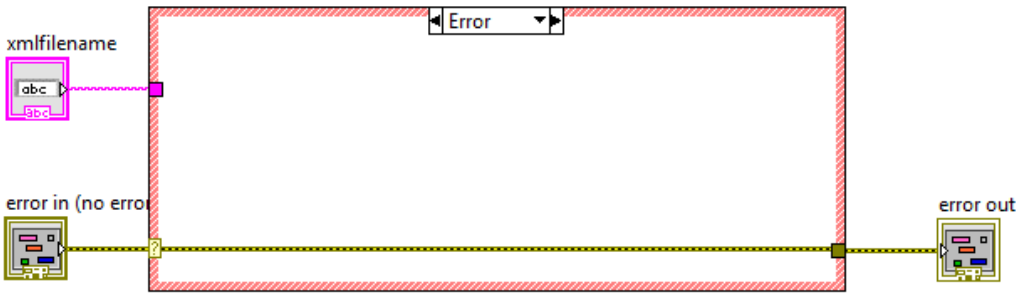

Abbildung 15 Blockschaltbild des VIs „DC1394_xml“

Abbildung 15 zeigt das Blockschaltbild des VIs zum Auslesen der XML Datei im Fehlerfall. So wird beim Aufruf des VIs überprüft ob am Eingang „error in“ ein Fehler aufgetreten ist. Ist dies der Fall wird der anliegende Fehler an den Ausgang „error out“ weitergeleitet und das VI beendet.

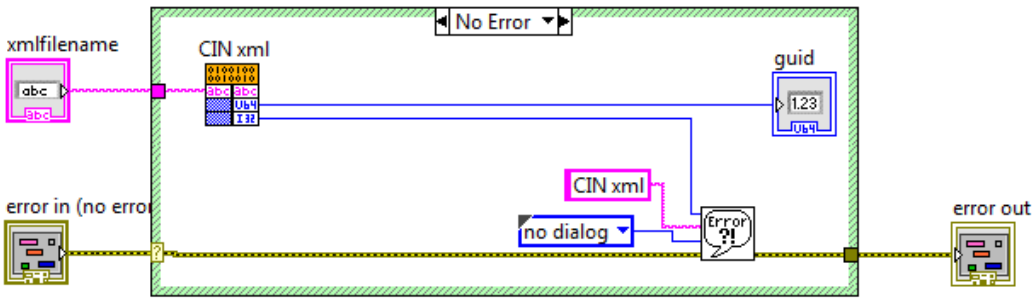

Abbildung 16 Blockschaltbild des VIs „DC1394_xml“- kein fehler

Die Abbildung 16 zeigt das Blockschaltbild für den Fall, dass bei dem Aufruf des VIs kein Fehler am Eingang „error in“ anliegt. So wird mit einer „if-else“-Anweisung zunächst der Eingang „error in“ überprüft. Liegt kein Fehler an, so wird der Dateiname der XML Datei an den Code Interface Node „CIN xml“ weitergeleitet. Der Code Interface Node liest dann die Kameraeinstellungen aus der Datei aus und stellt die Kamera ein. Als nächstes gibt der CIN die eindeutige Identifikationsnummer der Kamera „guid“ an den entsprechenden Ausgang des VIs weiter. Für den Fall, daß während der Ausführung des CINs ein Fehler aufgetreten ist, wird ein Errorcode an den Errorhandler weitergegeben. Dieser fügt dem Code die Quelle des Fehlers bei und leitet den Fehler ohne einen Dialog an den Ausgang „error out“ weiter.

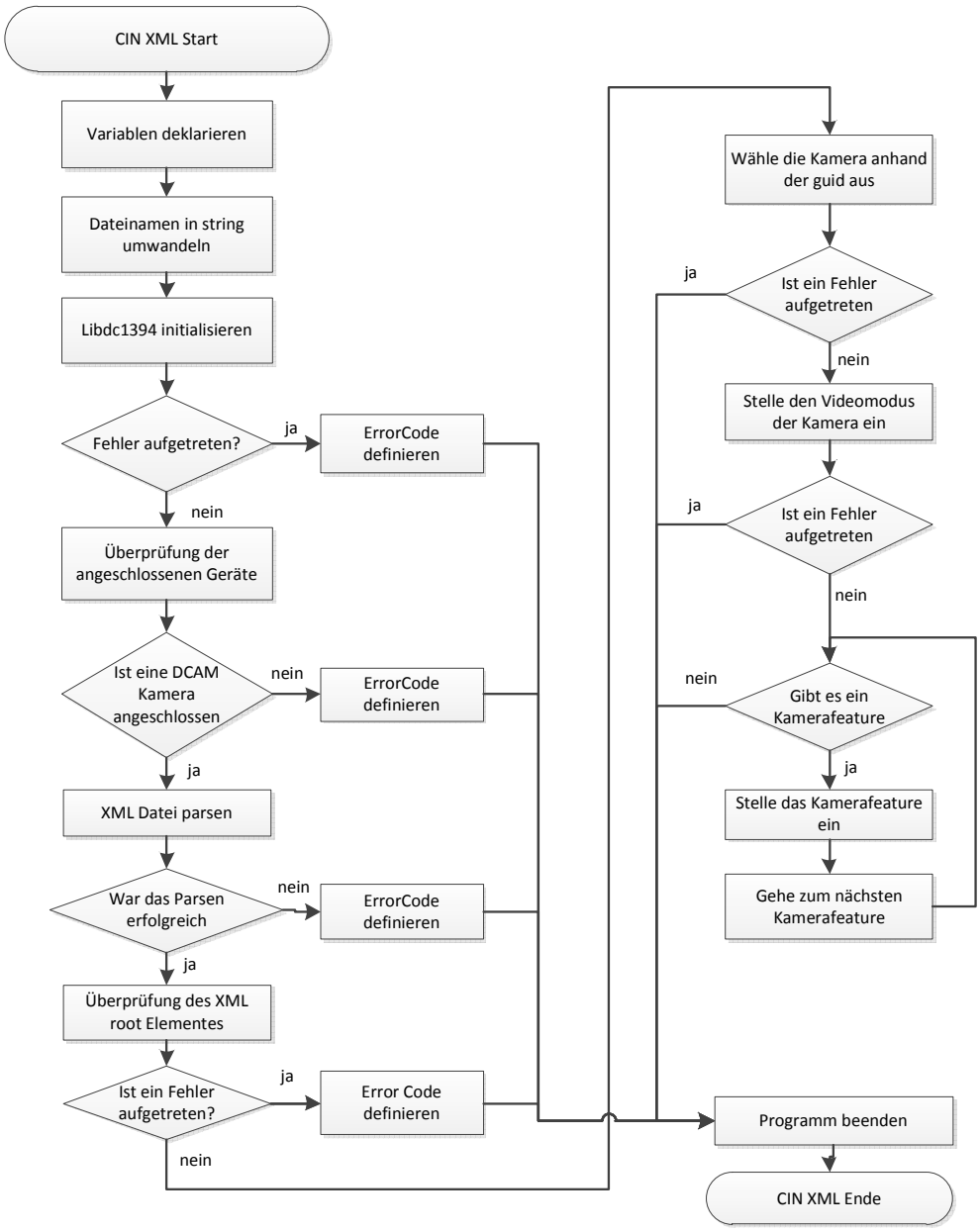

Abbildung 17 Vereinfachter Programablaufplan des CINs „CIN Xml“

Abbildung 17 zeigt den vereinfachten Programmablaufplan des Code Interface Nodes „CIN xml“. Nachdem der CIN vom VI gestartet wird, werden zunächst die benötigen Variablen deklariert. Danach wird zunächst der übergebene Dateiname von einem LabVIEW String in ein Char Array umgewandelt.

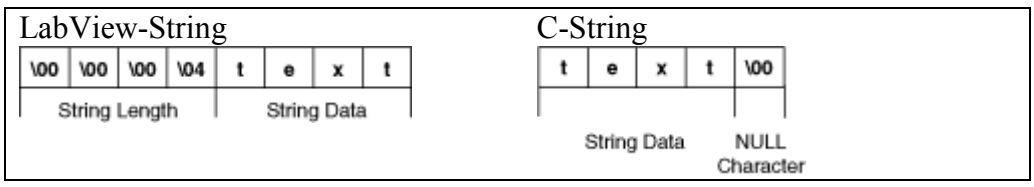

Abbildung 18 Vergleich LabVIEW String und C String

Die Abbildung 18 zeigt den Unterschied zwischen einem LabVIEW-String und einem C-String. So wird die Länge des LabVIEW-Strings als 32bit Integerzahl vor dem eigentlichen Stringinhalt angegeben. Danach folgen erst die Daten des Strings. Bei einem C-String kommen zuerst die Stringdaten. Das Ende des Strings wird dann durch das „NULL Symbol“ gekennzeichnet.

Als nächstes wird die libdc1394 initialisiert. Dies ist nötig, um den Benutzer Zugriff auf den IEEE1394 Bus zu ermöglichen. Falls dabei ein Fehler auftritt, wird ein Errorcode definiert und das Programm beendet. Fehlende Systemberechtigungen können zum Beispiel für eine fehlgeschlagene Initialisierung verantwortlich sein.

Als nächstes wird überprüft, ob überhaupt ein Gerät an den IEEE1394 Bus angeschlossen ist. Ist die Geräteanzahl gleich null, so wird ein Errorcode definiert und das Programm beendet.

```
//----------ist das xml file ein gültiges XML format---

    doc = xmlParseFile(filename);
    if (doc == NULL) {
        *errorout =5020;
        goto out2;
        }
    // zum Rooot zeigen
    cur = xmlDocGetRootElement(doc);
```

Abbildung 19 Quellcodeauszug des CINs „CIN xml“, parsen des XML files

Nach erfolgreicher Überprüfung des Bus wird die XML Datei geöffnet und in die einzelnen XML Elemente untergliedert. Den Vorgang des Untergliederns nennt man auch parsen. Schlägt das Parsen der Datei fehl, so wird ein Errorcode bestimmt und das Programm beendet. Fehler, die beim Parsen auftreten können, sind zum Beispiel das Nichtauffinden einer Datei oder Dateien, die eine ungültige XML Struktur aufweisen. Im nächsten Schritt wird untersucht, ob das root Element existiert und die richtige Bezeichnung („firecam“) aufweist. Bei einem Fehler wird erneut ein eigener Errorcode definiert und das Programm beendet. Die Abbildung 19 zeigt den dazu gehörigen Quellcode.

```
    //-----                    guid suchen                    --
    //-----------------------------------------------------------
    while(cur !=NULL)
    {
        if (!xmlStrcmp(cur->name, (const xmlChar *) "caminfo"))
            {
            attr =xmlHasProp(cur, (const xmlChar*) "guid");
            if (attr==NULL)
                {
                *errorout = 5023;
                goto out2;
                }
            else
                {
                //umwandeln der guid in eine ganze Zahl
                *guid=strtoll ( (char *)xmlGetProp(cur,
"guid"),NULL,0);
                camera = dc1394_camera_new (d, *guid);
                if (!camera)
                    {
                    *errorout= 5002;
                    goto out2;
                    }
                break;
                }
            }
    cur = cur->next;
    }
```

Abbildung 20 Quellcodeauszug des CINs „CIN xml“, selektieren der Kamera

Im nächsten Schritt wird die in der XML Datei vorhandene eindeutige Identifikationsnummer „guid“ gesucht. Die Abbildung 20 zeigt dabei den Quellcode für diese Aufgabe. So wird zunächst nach dem XML Node mit dem Namen „caminfo“ gesucht. Die Funktion „xmlStrcmp“ überprüft dabei, ob der Node den richtigen Namen hat. Als nächstes wird dann überprüft ob der Node „caminfo“ auch über das Attribut „guid“ verfügt. Ist dies der Fall, wird der XML Wert mittels der Funktion „strtoll“ in eine Zahl umgewandelt und in der Variable „guid“ gespeichert. Anhand der Nummer wird als nächstes die Kamera initialisiert. Schlägt die Initialisierung fehl, so wird das Programm mit den entsprechenden Errorcode beendet. Ein Fehler der dabei auftreten kann, ist beispielsweise, dass die in der XML Datei angegebene „guid“ nicht mit der am IEEE1394 Bus angeschlossene Kamera übereinstimmt.

Als nächstes werden die Videoeinstellungen aus der XML Datei ausgelesen und die Kamera entsprechend eingestellt. Tritt in diesem Fall ein Fehler auf, so wird das Programm mit den entsprechenden Errorcode beendet.

```
//  -------      video features    ----------

    err = dc1394_feature_get_all(camera, &features);
    if (err)
        goto out3;
    while(cur !=NULL)
        {
        if (!xmlStrcmp(cur->name, (const xmlChar *)
"FeaturesSettings"))
            {
            cur = cur->children;
            }
        if(!xmlStrcmp(cur->name, (const xmlChar *) "feature"))
            {
            tempvalue1=atoi( (char *)xmlGetProp(cur, "fid"));
            tempvalue2=atoi( (char *)xmlGetProp(cur,
"fvalue"));
            if(features.feature[tempvalue1].available ==
DC1394_TRUE)
                {
                err =
dc1394_feature_set_mode(camera,features.feature[tempvalue1].id,
DC1394_FEATURE_MODE_MANUAL) ;
                err =
dc1394_feature_set_value(camera,features.feature[tempvalue1].id
,tempvalue2);
                if(err)
                    goto out3;
                }
            }
        cur = cur->next;
        }
```

Abbildung 21 Quellcodeauszug des CINs „CIN xml“, einstellen der Kamerafeatures

Nachdem der Videomodus erfolgreich eingestellt wurde, werden die einzelnen Kamerafeatures eingestellt. Dazu wird in dem XML Dokument der Node mit dem Namen „FeaturesSettings“ gesucht. In diesem Node finden sich die einzelnen Features als Children Elemente wieder. Diese Children Elemente werden durchlaufen. Dabei werden die XML Attribute „fid“ und „fvalue“ ausgelesen und temporär gespeichert. Als nächstes wird überprüft, ob die Kamera auch tatsächlich die Kameraeinstellung unterstützt. Ist dies der Fall, so wird so wird das Feature auf „manuell“ gestellt und mit dem richtigen Wert eingestellt.

Sobald alle in der XML Datei angegeben Einstellungselemente durchlaufen wurden, wird das Programm beendet.

Nach fehlerfreier Ausführung des CINs ist die Kamera mit dem richtigen Videomodus und den einzelnen unterstützen Kamerafeatures eingestellt. Diese Werte werden flüchtig ins Kameraregister geschrieben und durch eine Unterbrechung der

Stromzufuhr auf die Werkseinstellungen zurückgesetzt. Die Kamera ist nun richtig eingestellt und betriebsbereit.

7.2 VI „DC1394_capture“

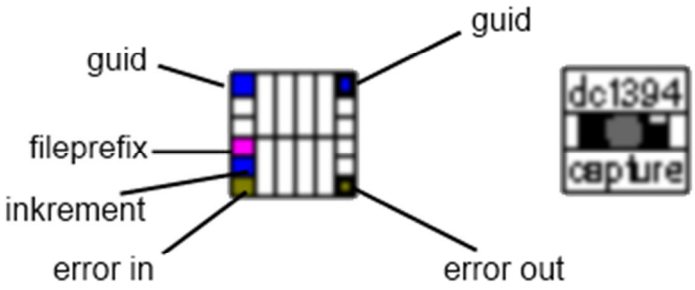

Abbildung 22 Symbole des VIs „DC1394_capture“

Die Abbildung 22 stellt das Blockschaltbildsymbol des VIs „DC1394_capture“ da. So sieht man auf der rechten Seite das grafische Symbol welches dann von dem Endbenutzer auf seine Blockschaltbildebene gezogen wird. Das Symbol ist bewusst in schwarz-weiß gehalten, um den Endbenutzer zu verdeutlichen das es sich hierbei um ein VI für monochrome Kameras handelt.

Das linke Symbol verdeutlicht die Ein- und Ausgänge, die von diesem virtuellen Instrument angenommen werden können. Erster Eingabewert ist die „guid“ Angabe, welche die eindeutige ID der Kamera bezeichnet. Dieser Wert wird von dem VI nicht verändert, sondern nur zur Kameraauswahl benutzt und zur weiteren Verwendung durchgeschleift. Die Input Variablen „fileprefix“ und „inkrement“ dienen dem VI zur Abspeicherung des Kamerabildes auf der Festplatte. Die Input Variable „error in“ dient Weitergabe von Fehlermeldungen. Tritt ein Fehler während der Ausführung des VIs auf, wird der Fehler über die Output Variable „error out“ weitergegeben.

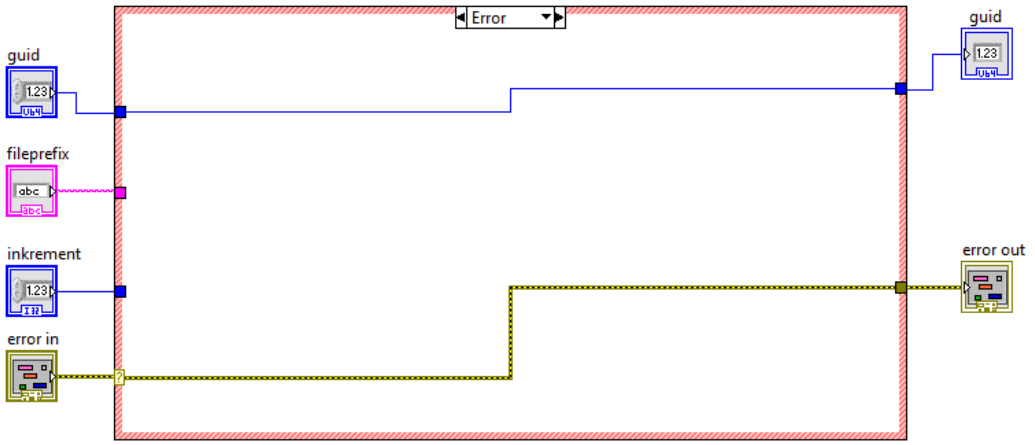

Abbildung 23 Blockschaltbild des VIs „DC1394_capture“ bei aufgetretenen Fehler

Abbildung 23 stellt die Struktur des Virtuellen Instruments „DC1394_capture“ dar. Auf der linken Seite sieht man die 4 Eingänge („guid“, „fileprefix“, „inkrement“, „error in“). Während auf der rechten Seite die Ausgänge („guid“, „error out“) zu sehen sind. Wird dieses VI aufgerufen, wird zunächst der „error in“ Wert überprüft. Ist ein Fehler im Vorfeld aufgetreten, so werden die Werte die „error in“ und „guid“ an die dazugehörigen Ausgänge weitergegeben.

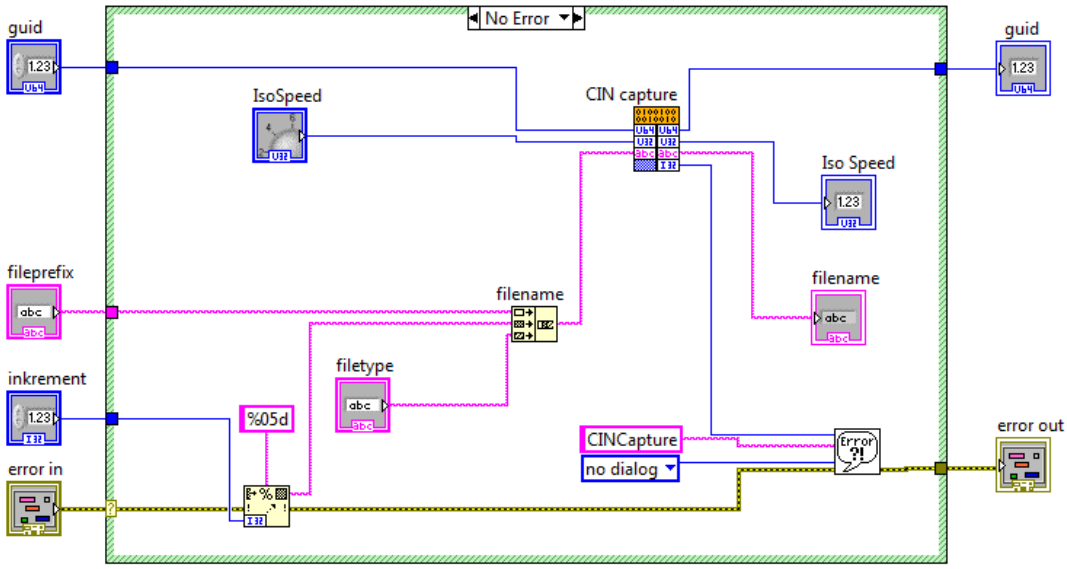

Abbildung 24 Blockschaltbild des VIs „DC1394_capture“ für keinen Fehler

Die Abbildung 24 zeigt das VI „DC1394_capture“ für den Fall, daß am „error in“ kein Fehler vorliegt. Zunächst wird der übergebene Wert „inkrement“ in einen fünf stelligen String umgewandelt. Danach wird der daraus entstandene String mit dem Input Wert „fileprefix“ und dem „filetype“ Wert kombiniert, so dass daraus der endgültige Dateiname, unter dem das Bild gespeichert wird, entsteht.

Der Code Interface Node „CIN capture“ bekommt die Werte „guid“, „IsoSpeed“ und „filename“ nun übergeben und wird ausgeführt. Der Wert „IsoSpeed“ bestimmt die Geschwindigkeit für die isochrone Übertragungsgeschwindigkeit der Kamera. Standardmäßig ist ein Wert von zwei, welcher eine Geschwindigkeit von 400 Mbit/s entspricht, eingestellt. Eine Liste aller Übertragungsgeschwindigkeiten befindet sich im Anhang. Falls ein Fehler in dem Code Interface Node „CIN capture“ aufgetreten ist, wird dieser danach an den Error Handler übergeben. Im Fehlerfall versieht der Error Handler den Fehler für eine spätere Identifizierung mit dem Quellnamen „CINCapture“ und leitet den Fehler danach ohne Unterbrechung des Programms an den Ausgang „error out“ weiter.

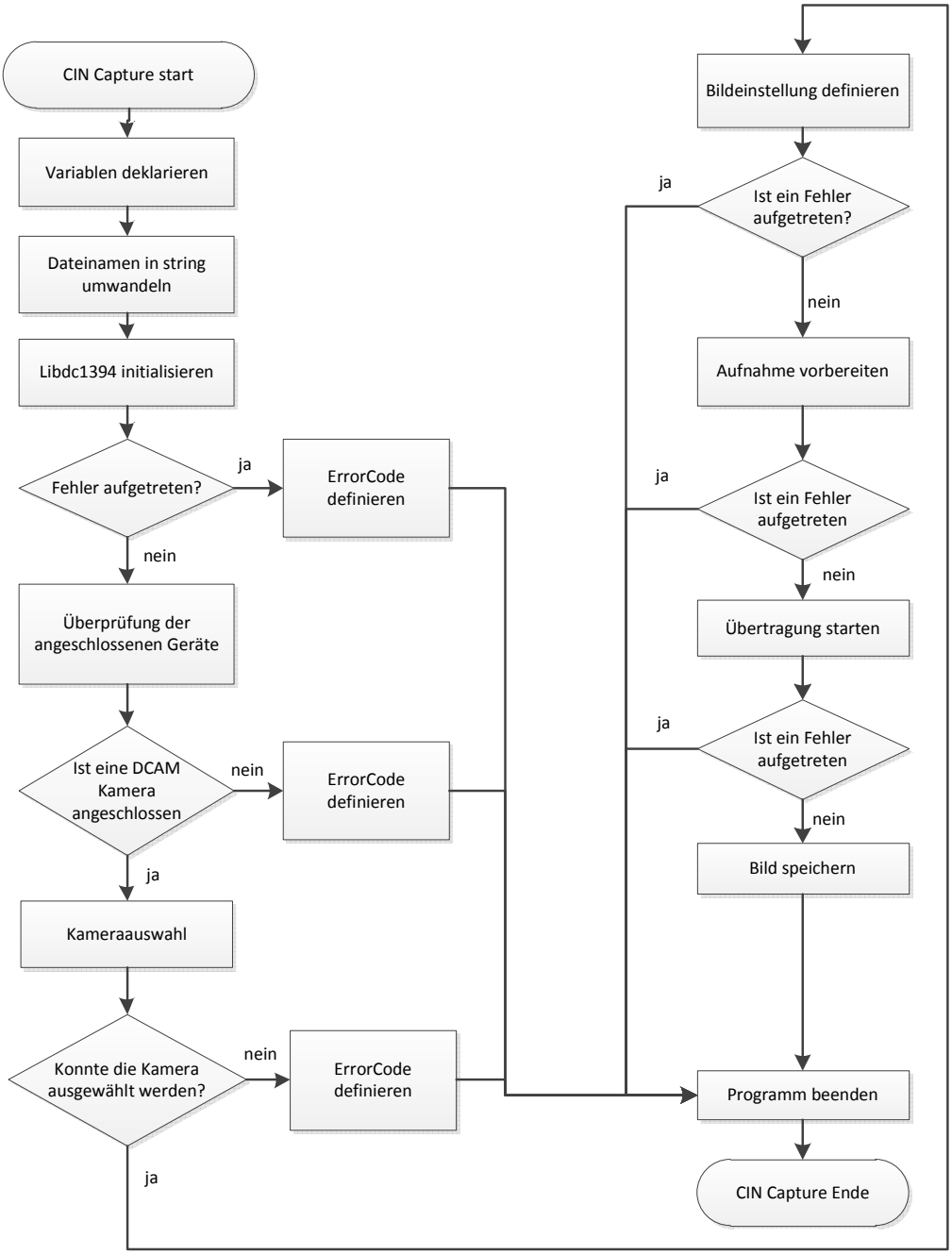

Abbildung 25 Vereinfachter Programmablaufplan des CINs „CIN capture“

Die Abbildung 25 stellt den vereinfachten Ablauf des Code Interface Nodes „CIN capture“ da. Beim Aufruf des CINs wird zunächst die benötigten Variablen deklariert. Danach wird der übergebene „LabVIEW String“ in ein „Char Array“ umgewandelt.

Als nächstes wird die „libdc1394“ initialisiert, um über die „libdc1394“ Library mit dem IEEE1394 BUS kommunizieren zu können. Schlägt die Initialisierung fehl, wird ein Errorcode definiert und der Code Interface Node beendet.

Nach einer erfolgreichen Initialisierung wird der IEEE1394 BUS auf Geräten untersucht, die dem DCAM Standard unterliegen. Können an dem BUS keine kompatiblen Geräte gefunden werden, wird erneut ein Errorcode definiert und der Code Interface Node beendet.

Als nächstes wird mit Hilfe der „guid Input“ Variable probiert die dazugehörige Kamera zu initialisieren. Wird keine Kamera mit entsprechenden „guid“ Variable gefunden, wird erneut ein Errorcode definiert und der Code Interface Node beendet.

```
  //1.videomode der Kamera abfragen

    err=dc1394_video_get_mode(camera,&video_mode);
    if(err)
        goto out3;
    //2.dimensionen abfragen
    err = dc1394_get_image_size_from_video_mode(camera,
video_mode, &width, &height);
    if(err)
        goto out3;
    //3. color coding abfragen
    err = dc1394_get_color_coding_from_video_mode(camera,
video_mode, &coding);
    if(err)
        goto out3;
    //4. IPL Frame einstelllen
    switch (coding)
        {
        case DC1394_COLOR_CODING_MONO8:
            bw_image = cvCreateImage(cvSize(width, height),
IPL_DEPTH_8U, 1);
            totalbyte=1;
            break;
        case DC1394_COLOR_CODING_MONO16:
            bw_image = cvCreateImage(cvSize(width, height),
IPL_DEPTH_16U, 1);
            totalbyte=2;
            break;
        case DC1394_COLOR_CODING_MONO16S:
            bw_image = cvCreateImage(cvSize(width, height),
IPL_DEPTH_16S, 1);
            totalbyte=2;
            break;
        default:
        //*cinstatus =err;
            bw_image=NULL;
            goto out3;
            break;
        }
```

Abbildung 26 Quellcodeauszug des CINs „CIN capture“, Einstellung des Bildes

Abbildung 26 zeigt den Code zur Bildformateinstellung. So wird zunächst mit der „libdc1394“ Funktion der in der Kamera aktuell eingestellte Videomodus abgefragt und in die Variable „video_mode“ abgespeichert. Im nächsten Schritt werden

mithilfe der Funktion „dc1394_get_image_size_from_video_mode“ und der Variable „video_mode“ die Bilddimensionen abgerufen und in die Variablen „height“ und „width“ abgespeichert. Als nächstes wird die verwendete Bildkodierung abgerufen und in die Variable „coding“ abgespeichert. Danach wird unter Hinzunahme einer „switch Case“- Anweisung, der IPLFrame „bw_image“ der vorgegebenen Bildkodierung angepasst. Der IPLFrame ist eine Bildstruktur der Library „openCV“ und ermöglicht die spätere Bildspeicherung auf der Festplatte.

Sobald eine der drei verwendeten „libdc1394“ Funktionen einen Fehler zurückgeben, wird das Programm an die Sprungmarke „out3“ geleitet. Bei dieser Sprungmarke handelt es sich um einen Programmausstiegspunkt, der den Errorcode weiter verarbeitet und die bis dahin verwendeten Systemresourcen wieder freigibt.

Nachdem der IPLFrame nun an das Format des Kamerabildes angepasst wurde, wird als nächstes die Aufnahme der Kamera vorbereitet.

```
//--------------   setup capture --------------------------
    err=dc1394_video_set_iso_speed(camera, *IsoSpeed);
    if(err)
        goto out4;
    err=dc1394_capture_setup(camera,6,
DC1394_CAPTURE_FLAGS_DEFAULT);
    if(err)
        goto out4;
//--------------   start capture  -------------------------
    err=dc1394_video_set_transmission(camera, DC1394_ON);
    if(err)
        goto out5;
    err=dc1394_capture_dequeue(camera,
DC1394_CAPTURE_POLICY_WAIT, &frame);
    if(err)
        goto out5;
```

Abbildung 27 Quellcodeauszug des CINs „CIN capture“, Bildaufnahme

Abbildung 27 zeigt den Quellcode, der für die Aufnahme des Kamerabildes verantwortlich ist. So wird zunächst der Kamera eine bestimmte Bandbreite auf den IEEE1394 BUS zugesichert. Danach wird der für die Kamera zu verwendende Ringbuffer bereitgestellt. Der dabei an die Funktion „DC1394_capture_setup“ übergebene Wert 6 stellt die Frameanzahl des Ringbuffers dar.

Mit dem Funktionsaufruf „dc1394_video_set_transmission (camera, DC1394_ON)“ wird dann die Übertragung der Kamera gestartet und an den Ringbuffer übergeben. Danach reiht man den nächsten von der Kamera übertragenen vollständigen Frame vom Ringbuffer aus. Dieser ausgereihte Frame kann nun für eine weitere

Verarbeitung benutzt werden. Die Kamera sendet während dessen weiter Frames an den Ringbuffer.

Gibt eine der vorher aufgerufenen Funktionen einen Error zurück, so wird das Programm an die dafür vorgesehene Sprungmarke geführt und das Programm beendet. Im Gegensatz zur Sprungmarke 4, wird bei der Sprungmarke 5 die gestartete Übertragung beendet.

```
//------------------------------Bild abspeichern-----------------
    memcpy(bw_image->imageData, (char *)frame->image,
totalbyte*width*height);
    cvSaveImage( filename, bw_image,0);
    //den Frame wieder dem Ringbuffer zur Verfügung stellen
    err=dc1394_capture_enqueue(camera, frame);
```

Abbildung 28 Quellcodeauszug des CINs „CIN capture“, abspeichern des Kameraframes

Abbildung 28 zeigt den verwendeten Code für die Abspeicherung des Kamerabildes auf der Festplatte an. So wird mit der Funktion „memcpy“, der aus dem Ringbuffer ausgegliederte Frame in den IPLFrame byteweise kopiert. Dabei bestimmt das Produkt aus Höhe, Breite und Bittiefe das Ende des zu kopierenden Frames. Das Bild der Kamera liegt nun im IPLFrame. Mit der Funktion „cvSaveImage“ wird nun der IPLFrame („bw_image“) als Bild auf der Festplatte gespeichert. Dabei bestimmt die Variable „Filename“ den Pfad, Dateinamen und Dateityp. Im nächsten Schritt wird der nun vollständig verarbeite Frame dem Ringbuffer wieder zur Verfügung gestellt.

Im letzten Schritt wird der Code Interface Node beendet. Für den Fall, daß während des Programmdurchlaufs ein Fehler aufgetreten ist, sind unterschiedliche Programmausstiegsmarken implementiert.

```
out5:
    if (err)
        *errorout =5100+err ;
    err = dc1394_video_set_transmission(camera, DC1394_OFF);
    err = dc1394_capture_stop(camera);
out4:
    cvReleaseImage(&bw_image);
out3:
    dc1394_camera_free(camera);
out2:
    dc1394_camera_free_list (list);
out1:
    dc1394_free (d);
    return noErr;
    }
```

Abbildung 29 Quellcodeauszug des CINs „CIN capture“, Programmausstiegsmarken

Die Abbildung 29 zeigt den Quellcodeauszug mit den fünf unterschiedlichen Programmausstiegsmarken „out1“ bis „out5“.Die Ansteuerung der unterschiedlichen Sprungmarken erfolgt in Abhängigkeit vom Zeitpunkt des Fehlersauftritts während des Programmdurchlaufs. Tritt ein Fehler beispielsweise unmittelbar nach der der Initialisierung des IPLFrames auf, so wird das Programm an die Sprungmarke „out4“ geführt. Der IPLFrame wird dann zunächst released, d.h. die Struktur gelöscht und der verwendete Speicher dem System wieder freigegeben. Danach werden die Kamerastruktur und die Kameraliste freigegeben. Als letzter Schritt wird mit der Funktion „dc1394_free“ die Steuerung über den IEEE1394 Bus ebenfalls wieder freigeben und das Programm ohne Fehlerrückgabe beendet. Die fehlerfreie Beendigung des Programms ist zwingend erforderlich, da LabVIEW im Fehlerfall das gesamte VI anhalten würde. Das eigentliche Errorhandling geschieht über die Variable „errorout“ nach Beendigung des Code Interface Nodes. Für den Fall, daß das Programm ohne einen Fehler beendet wird, durchläuft es den letzten Teil des Programms ab der Sprungmarke „out5“. Bei dieser Sprungmarke wird zunächst überprüft, ob ein Fehler bei dem Aufrufen der einzelnen „libdc1394“- Funktionen aufgetreten ist. Ist dies der Fall, wird der zurückgegebene Error mit 5100 addiert, um zur späteren Verarbeitung des Errors in dem von LabVIEW frei verfügbaren Errorcode Bereich zu gelangen. Danach wird die Übertragung der Kamera beendet und das Programm durchläuft die bereits erwähnten Sprungmarken „out4“ bis „out1“.

7.3 VI „DC1394_show“

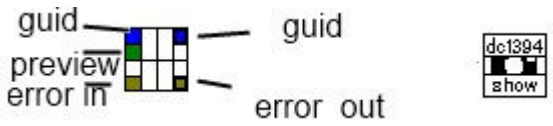

Abbildung 30 Symbole des VIs „DC1394_show“

Die Abbildung 30 zeigt auf der rechten Seite das Blockschaltbildsymbol für das VI „DC1394_show“. Auf der linken Seite sieht man die einzelnen Ein- und Ausgänge des VIs. So verfügt dieses VI genau wie das Capture VI über die Ein- und Ausgänge „guid“ und „error in“ und „error out“. Zudem verfügt dieses VI über einen Eingang „previrew“. Dieser Eingang ist vom Typ „Boolean“ und hat die Funktion, dem VI mitzuteilen, ob eine Bildvorschau erfolgen soll.

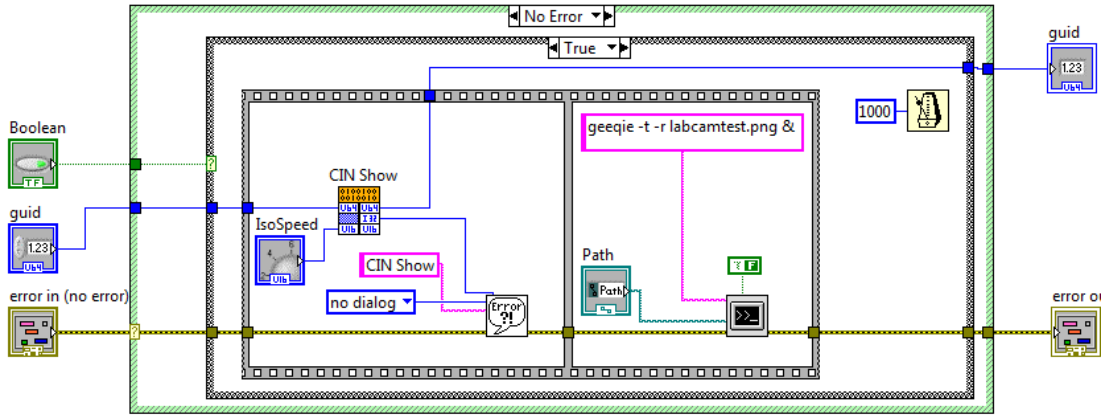

Abbildung 31 Blockschaltbild des VIs „DC1394_show“

Die Abbildung 31 zeigt den logischen Programmablauf des VIs „DC1394_show“. Analog zu den beiden anderen VIs, wird zunächst überprüft, ob im Vorfeld ein Fehler aufgetreten ist. Ist dies der Fall, so wird der Fehler und die „guid“ an die Ausgänge durchgeschleift und das Programm beendet. Ist kein Fehler aufgetreten, so überprüft das Programm im nächsten Schritt den „preview“- Eingang. Ist dieser Eingang auf „TRUE“, wird der Code Interface Node „CIN show“ gestartet. Dieser CIN unterscheidet sich vom VI „DC1394_capture“ lediglich darin, daß ein Bild mit dem Namen „labcamtest.png“ in den temporären Ordner von Linux gespeichert wird. Nach der Ausführung des CINs wird überprüft, ob während der Ausführung des CINs ein Fehler aufgetreten ist. Dabei wird der Fehlercode vom CIN von dem „Simple Error Handler“ überprüft.

Als nächstes wird das Bild mithilfe des Programms „geeqie“ geöffnet. Dieses geschieht mit dem Konsolenbefehl „geeqie –t –r labcamtest.png &“. Der Befehlszusatz „–t“ dient dazu, das „geeqie“ in einem Fenster ohne Toolleiste gestartet wird. Die Option „-r“ sorgt dafür, dass bei einem wiederholten Aufruf das Bild in derselben Instanz neugeladen wird.

Als nächstes misst das Programm mithilfe der Metronomfunktion wie viel Zeit bereits vergangen ist und stoppt bis genau 1000 ms vergangen sind, um es anschließend zu beenden.

7.4 Testumgebung

Um die einzelnen Programmfunktionalitäten zu testen, wurde ein kleines Testprogramm entwickelt. Das Testprogramm zeigt auf, dass es möglich ist, Bilder von einer Kamera mit unterschiedlichen Shutterzeiten aufzuzeichnen. So wurde im Vorfeld zunächst zwei XML Einstellungsdateien, die sich lediglich durch unterschiedliche Werte für die Shutterzeit unterscheiden, erstellt.

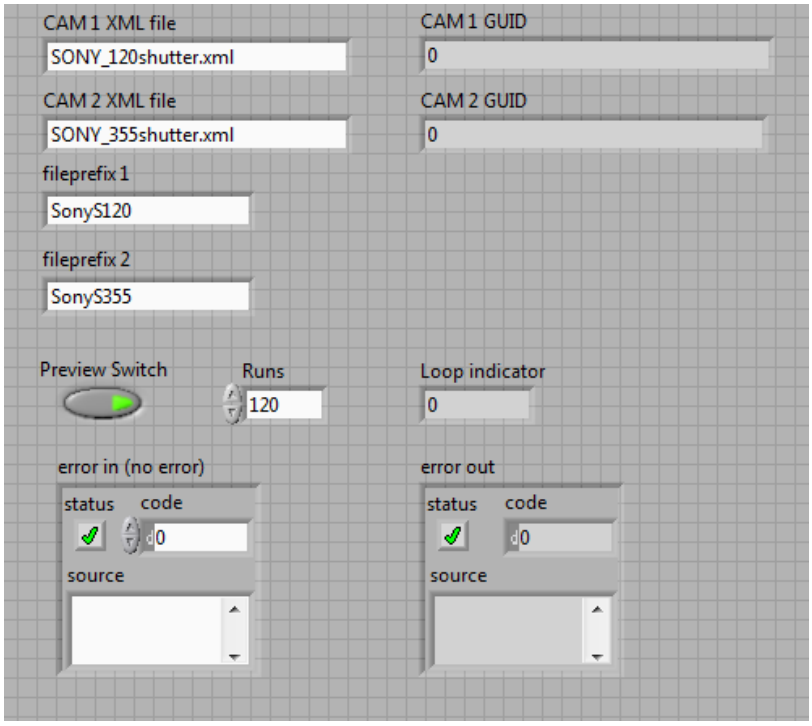

Abbildung 32 Frontpanelansicht der Testanwendung

Abbildung 32 zeigt die Frontpanelansicht, bei dem der Benutzer noch einige Eingaben vornehmen tätigen muss, um das Programm entsprechend der jeweiligen Anforderungen einzustellen. So muss der Benutzer zunächst die beiden XML Dateien benennen, in denen sich die unterschiedlichen Einstellungen der Kamera befinden. Zudem muss er für die zwei unterschiedlichen Aufnahmen jeweils einen Dateiprefix angeben. Der Preview Button sollte auf „TRUE“ stehen, um eine Preview der Bilder zu erhalten. Der Benutzer gibt über das Feld „Runs“ an, wie viele Aufnahmen insgesamt für die jeweilige Einstellung getätigt werden sollen. Das Feld „Loop Indicator“ zeigt dem Benutzer, auf wie viele Durchgänge das Programm bereits durchlaufen hat.

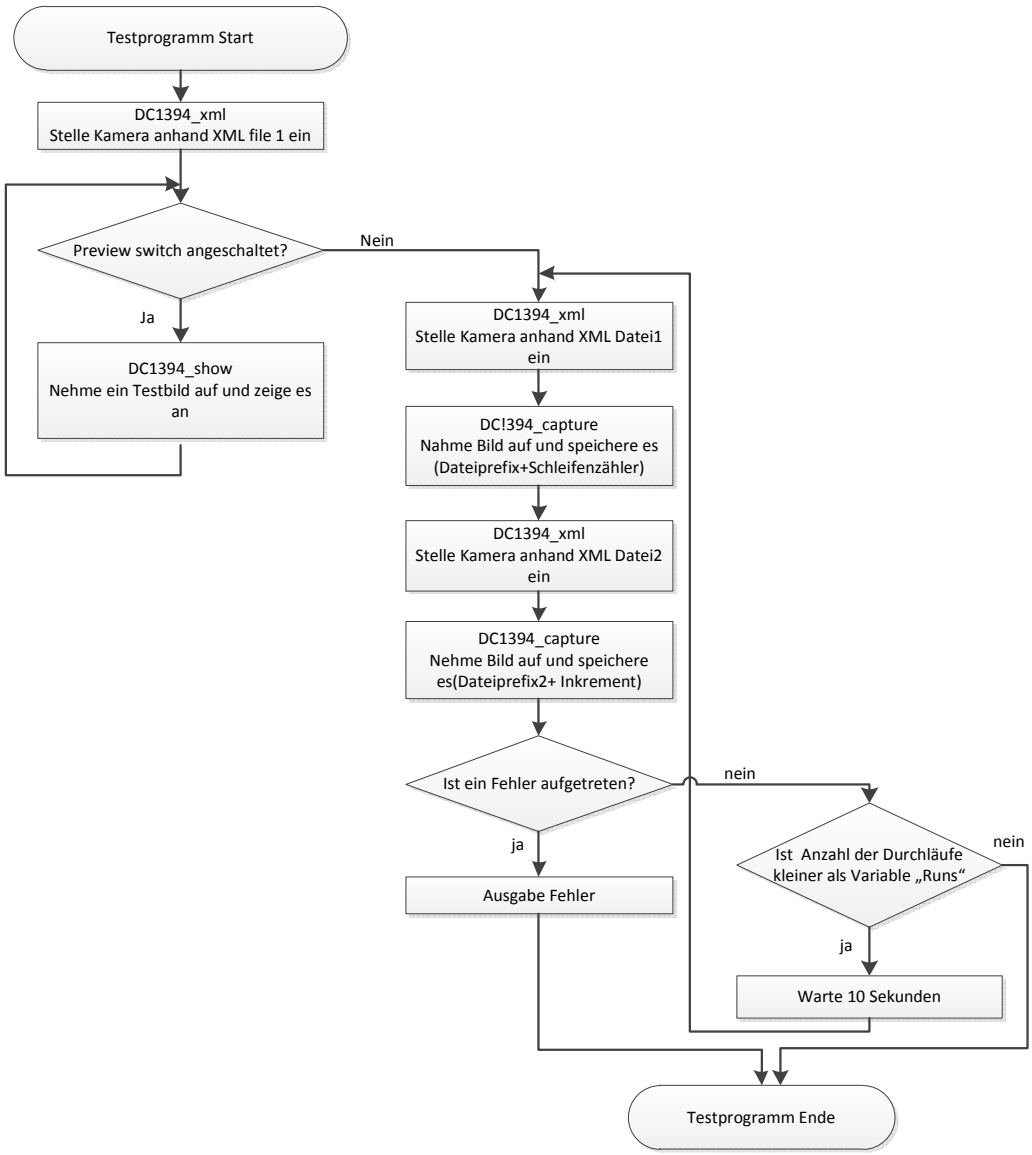

Abbildung 33 Programmablaufplan des Testprogramms

Die Abbildung 33 zeigt den Programmablaufplan des Testprogramms. Nachdem der Benutzer die oben geschilderten Eingaben getätigt hat, ist das Programm bereit zur Ausführung. Zunächst wird die Kamera das VI „DC1394_xml“ gestartet. Dabei wird die Kamera anhand der in der ersten angebenden Datei eingestellt. Danach wird mittels einer Schleife überprüft, ob der „preview“ Button gedrückt ist. Ist dies der Fall, so startet das VI „DC1394_show“ und das Testbild der Kamera wird angezeigt. Die Aufnahme und die Anzeige des Tesbilds wird solange wiederholt bis der Benutzer den „preview“ Button ausschaltet. Dadurch bekommt der Benutzer nun die Gelegenheit die Kamera an dem Laborplatz neu auszurichten.

Nachdem der Benutzer den „preview“ Button ausgeschaltet hat, geht das Programm in die Serienaufnahme der Bilder über. Dabei startet das Programm das VI

„DC1394_xml“ und stellt die Kamera anhand der angegebenen XML Datei 1 ein. Danach führt das Programm das VI „DC1394_capture“ aus und speichert das Kamerabild anhand des angegebenen Dateiprefixes und des Schleifendurchlaufzählers lokal auf der Festplatte ab. Direkt danach wird die Kamera durch den Aufruf des VI „DC1394_xml“ anhand der zweiten XML Datei neu eingestellt. Im Anschluss wird das VI „DC1394_capture“ aufgerufen. Mithilfe des Schleifendurchlaufzählers und des Dateiprefixes2 wird das Kamerabild auf der Festplatte gespeichert. Danach wird überprüft, ob während der Ausführung ein Fehler aufgetreten ist. Ist dies der Fall, wird der Fehler über eine DialogBox ausgegeben. Andernfalls vergleicht das Programm den Schleifendurchlaufzähler mit der Variable „Runs“. Ist der Wert Runs grösser als der Schleifendurchlaufzähler so werden die letzten Schritte nach der Bildvorschau erneut ausgeführt. Dadurch entsteht eine fortlaufende Bildspeicherung mit zwei verschiedenen Kameraeinstellungen. Ist die maximale Anzahl der gewünschten Durchläufe erreicht, wird das Programm beendet.

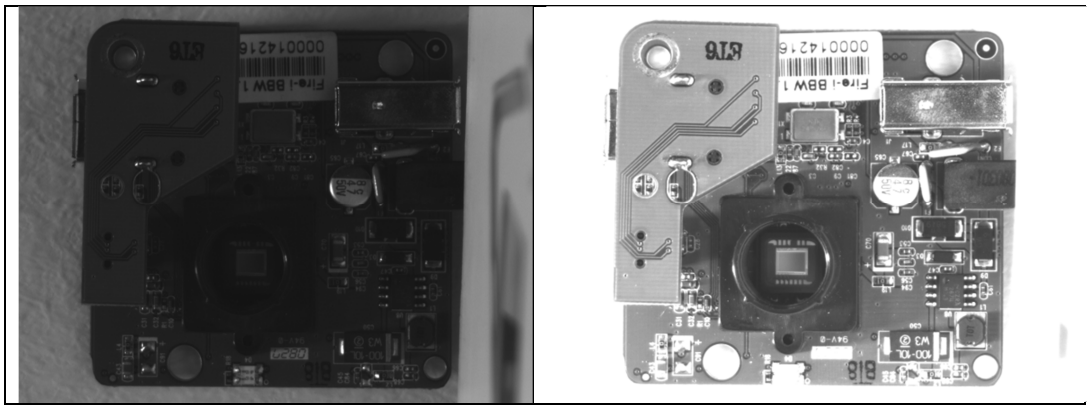

Abbildung 34 Vergleich zweier Aufnahmen eines Objektes mit unterschiedlichen Shutterzeiten

Die Abbildung 34 zeigt die bei einem Testdurchlauf entstandenen Aufnahmen. Das Objekt, das hier abgelichtet wurde, ist eine Firewire Platinen Kamera. Beide Aufnahmen wurden mit derselben Kamera durchgeführt. Die Einstellungen der Kamera waren bis auf die Belichtungszeit (Shutter) ebenfalls identisch. So gelang es durch ein Heraufsetzen der Belichtungszeit den störenden Hintergrund beinahe komplett auszublenden. Hinzu kommt, dass man die einzelnen Komponenten des Testobjektes viel besser erkennen kann. Diesen Effekt, der durch das Heraufsetzen der Belichtungszeit zustande kommt, kann man alternativ durch die Einstellung der Sensorempfindlichkeit (Gain) hervorrufen. Allerdings haben Versuche gezeigt, dass dies häufig ein Rauschen im Bild verursacht.

8 Zusammenfassung und Ausblick

Mit den drei entwickelten virtuellen Instrumenten ist es gelungen, eine Schnittstelle für monochrome Kameras, die dem sogenannten „DCAM“- Standard unterliegen zu entwickeln und in LabVIEW zu implementieren. So ist es mit dem VI „DC1394_xml“ möglich, anhand einer XML Datei eine Kamera einzustellen. Mit dem VI „DC1394_capture“ werden einzelne Bilder von der Kamera übertragen und in einem gängigen Bildformat abgespeichert. Zudem können Bilderserien aufgenommen werden. Das VI „DC1394_show“ bietet die Möglichkeit einer Bildvorschau. Durch die Implementation der selbst definierten Fehlercodes kann der Benutzer auftretende Fehler schnell identifizieren.

Die Bildanzeige, die in dem VI „DC1394_show“ implementiert ist, speichert das Kamerabild zunächst auf der Festplatte und öffnet es mit dem externen Bildbetrachtungsprogramm „geeqie“.Ein Umstand der die Bildwiederholungsrate stark limitiert. Eine Weiterentwicklung der Schnittstelle mit dem Ziel eine leistungsstärkere Bildanzeige zu implementieren, könnte je nach Anwendung sinnvoll sein.

Die Schnittstelle verfügt bisher nur über die Möglichkeit monochrome Bilder abzuspeichern. Die verwendeten Librarys unterstützten neben den monochromen Bildformaten auch verschieden Farbformate. Bei einer Weiterentwicklung könnte man die Schnittstelle mit einer Unterstützung für Farbkameras, die den „DCAM“-Standard unterliegen, erweitern.

Bei der Einbindung externer Codes in LabVIEW sollte in Zukunft die Call Library Function angewendet werden. Somit könnten alle Funktionen der Schnittstelle in einer zentralen Library abgebildet werden. Dies würde die Weiterentwicklung der bestehenden Funktionen wie auch die Implementation neuer Funktionen erleichtern.

Bisher ist es nur möglich einzelne Bilder über die Schnittstelle aufzunehmen. Die Library openCV bietet allerdings eine Vielzahl weiterer Funktionen, die in der Industrie Anwendung finden. So könnte man zum Beispiel ein Feature zur Objekterkennung implementieren.

Mit der Nutzung unterschiedlicher Librarys, die unterschiedlichen open Source Lizenzen unterliegen, ist es gelungen die Schnittstelle als ein Open Source Produkt zu entwickeln, das den Nutzern kostenlos zur Verfügung steht.

Das Ziel einer kostenlosen Aufzeichnung von monochromen Bildern ist erreicht und es wurde eine Alternative -auf Basis von open Source Quellen- zu den von National Instruments angebotenen Modul „NI Vison Development Modul, erfolgreich entwickelt.

9 Literaturverzeichnis

[1394TA08] 1394 Trade Association. (2008). *1394 Trade Association*. Abgerufen am 03. 08 2011 von http://www.1394ta.org

[Bässmann10] Bässmann, H. (2010). *Industrial Cameras*. Abgerufen am 02. 08 2011 von http://www.industrial-camera.com/

[Bru11] Bruns, A. (2011). *Programmierung von IEEE1394 Kameras.* Praxisprojektarbeit_Programmierung von IEEE1394 Kameras.pdf.

[Douxchamps11] Douxchamps, D. (kein Datum). *libdc1394 Homepage*. Abgerufen am 02. 08 2011 von http://damien.douxchamps.net/ieee1394/libdc1394/

[ElekKomp10] *Elektronik Kompendium.* (1997-2010). Abgerufen am 01. 08 2011 von Elektronik einfach und leicht verständlich: http://www.elektronik-kompendium.de/sites/com/0808111.htm

[ITHandbuch01] (2001). IT-HanBuch. In H.-J. P. Heinrich Hübschner, *IT-Handbuch* (S. 120). Braunschweig: Westermann Schulbuchverlag GmbH.

[Max11] *Max-Planck-Institut für Dynamik und Selbstorganisation.* (2011). Abgerufen am 15. 08 2011 von http://www.ds.mpg.de/instUeberInstitut/instForschungsthemen/index.html

[Nat11] National Instruments. (06. 12 2011). *10 Considerations When Choosing Vision Software.* Abgerufen am 20. 12 2011 von http://zone.ni.com/devzone/cda/tut/p/id/2957

[Nat06] National Instruments. (September 2006). *Custom Error Handling In LabVIEW.* Abgerufen am 05. 12 2011 von http://zone.ni.com/devzone/cda/tut/p/id/3209

[Nat111] National Instruments. (kein Datum). *Firemenprofil National Instruments.* Abgerufen am 20. 12 2011 von http://germany.ni.com/firmenprofil

[Nat112] National Instruments. (2011). *PXI-Hard- und Softwarearchitektur.* Abgerufen am 3. 1 2012 von http://www.ni.com/pxi/whatis/d/

[Nat031] National Instruments. (April 2003). *Using External Code in LabVIEW.* Abgerufen am 05. 12 2011 von http://www.ni.com/pdf/manuals/370109b.pdf

[openCVdevTeam11] opencv dev team. (2011). *OpenCV v2.3 documentation*. Abgerufen am 03. 08 2011 von http://opencv.itseez.com/

[Dan11] Veillard, D. (kein Datum). *The XML C parser and toolkit of Gnome.* Abgerufen am 19. 12 2011 von http://xmlsoft.org/index.html

[Wiki_LabVIEW] *Wikipedia.* (5. 10 2011). Abgerufen am 15. 01 2012 von Die freie Enzyklopädie: http://de.wikipedia.org/wiki/LabVIEW

[Wiki_DCAM] *Wikipedia.* (26. 10 2011). Abgerufen am 07. 12 2011 von Die freie Enzyklopädie: http://de.wikipedia.org/wiki/DCAM

[Wiki_Linse] *Wikipedia.* (19. 05 2011). Abgerufen am 02. 08 2011 von Die freie Enzyklopädie: http://de.wikipedia.org/wiki/Linsengleichung

[Wiki_Firewire] *Wikipedia.* (15. 01 2012). Abgerufen am 02. 08 2011 von Die freie Enzyklopädie: http://de.wikipedia.org/wiki/Firewire

10 Anhang

10.1 Quellcode CIN XML

```
/* CIN source file */
#include "extcode.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <libxml/parser.h>
#include <libxml/tree.h>
#include <dc1394/dc1394.h>

MgErr CINRun(LStrHandle xmlfilename, uInt64 *guid, int32 *errorout,
    LStrHandle vendor, LStrHandle model);

MgErr CINRun(LStrHandle xmlfilename, uInt64 *guid, int32 *errorout,
    LStrHandle vendor, LStrHandle model)
    {

    dc1394camera_t *camera;
    int i;
    xmlDocPtr doc;
    xmlNodePtr cur;
    xmlChar *value1, *value2;
    xmlAttrPtr attr,attr2;
    int tempvalue1, tempvalue2;
    dc1394featureset_t features;
    unsigned int width, height, totalbyte;
    dc1394video_frame_t *frame;
    dc1394_t * d;
    dc1394camera_list_t * list;
    dc1394error_t err;

    //Firewirebus durchsuchen
    d = dc1394_new ();
        if (!d){
        *errorout =5000;
            return noErr;
        }
        err=dc1394_camera_enumerate (d, &list);
        if (err)
            goto out1;
        if (list->num == 0) {
            *errorout =5001;
            goto out1;
            }
//------------Filename umwandeln---------------
    char* filename = (char*) malloc((*xmlfilename)-
>cnt*sizeof(uChar)+1);
    for ( i=0; i<(*xmlfilename)->cnt; ++i)
        filename[i] = (*xmlfilename)->str[i];
    filename[(*xmlfilename)->cnt]=0;
//----------ist das xml file ein gültiges XML format-----------
    doc = xmlParseFile(filename);
    if (doc == NULL) {
        *errorout =5020;
        goto out2;
        }
    // zum Rooot zeigen
    cur = xmlDocGetRootElement(doc);
//abfragen ob das xml Daten enthält
```

```
    if (cur == NULL)
        {
        *errorout=5021;
        goto out2;
        }
    //Abfragen ob es sich bei dem Dokument , um den richtigen Typ
handelt
    if (xmlStrcmp(cur->name, (const xmlChar *) "firecam")) {
        *errorout  =5022;
        goto out2;
        }
    else {}
    //----------XML Ebene tiefer gehn auf das nächste children-----
    cur = cur->children;
    //-----------------------------------------------------------------
//-----                    guid suchen                      ------
    //-----------------------------------------------------------------
    while(cur !=NULL)
    {
        if (!xmlStrcmp(cur->name, (const xmlChar *) "caminfo"))
            {
            attr =xmlHasProp(cur, (const xmlChar*) "guid");
            if (attr==NULL)
                {
                *errorout = 5023;
                goto out2;
                }
            else
                {
                //umwandeln der guid in eine ganze Zahl
                *guid=strtoll ( (char *)xmlGetProp(cur,
"guid"),NULL,0);
                camera = dc1394_camera_new (d, *guid);
                if (!camera)
                    {
                    *errorout= 5002;
                    goto out2;
                    }
                break;
                }
            }
    cur = cur->next;
    }
    //-----------------------------------------------------------------
    //-----                   Video mode                        ------
    //-----------------------------------------------------------------
while(cur !=NULL)
    {
    if (!xmlStrcmp(cur->name, (const xmlChar *) "video"))
        {
        attr =  xmlHasProp(cur, (const xmlChar*) "videomode");
        attr2 = xmlHasProp(cur, (const xmlChar*) "framerate");
        if (attr==NULL || attr2==NULL)
            {
            *errorout=5024;
            goto out3;
            }
        else
            {
            //Umwandlung der String werte in integer
            tempvalue1=atoi( (char *)xmlGetProp(cur, "videomode"));
            tempvalue2=atoi( (char *)xmlGetProp(cur, "framerate"));
            //Videomodus in der kamera einstellen
```

```
            err = dc1394_video_set_mode(camera,tempvalue1);
            err = dc1394_video_set_framerate(camera,tempvalue2);
            if (err)
                goto out3;
            break;
            }
        }
    cur = cur->next;
    }

//-----------------------------------------------------------------
//     -------------      video features                 ----------
//-----------------------------------------------------------------
    err = dc1394_feature_get_all(camera, &features);
    if (err)
        goto out3;
    while(cur !=NULL)
        {
        if (!xmlStrcmp(cur->name, (const xmlChar *)
"FeaturesSettings"))
            {
            cur = cur->children;
            }
        if(!xmlStrcmp(cur->name, (const xmlChar *) "feature"))
            {
            tempvalue1=atoi( (char *)xmlGetProp(cur, "fid"));
            tempvalue2=atoi( (char *)xmlGetProp(cur, "fvalue"));
            if(features.feature[tempvalue1].available ==
DC1394_TRUE)
                {
                err =
dc1394_feature_set_mode(camera,features.feature[tempvalue1].id,DC139
4_FEATURE_MODE_MANUAL) ;
                err =
dc1394_feature_set_value(camera,features.feature[tempvalue1].id,temp
value2);
                if(err)
                    goto out3;
                }
            }
        cur = cur->next;
        }
//Programmausttiege
out3:
    if (err)
        *errorout= err+5100;
    dc1394_camera_free(camera);
out2:
    xmlFreeDoc(doc);
    xmlCleanupParser();
out1:
    dc1394_camera_free_list (list);
        dc1394_free (d);
    return noErr;
}
```

10.2 Quellcode CIN Capture

```
/* CIN source file */
#include "extcode.h"
#include <dc1394/dc1394.h>
#include <opencv/cv.h>
#include <opencv/highgui.h>
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <inttypes.h>

MgErr CINRun(uInt64 *guid, uInt32 *IsoSpeed, LStrHandle arg1,
    int32 *errorout);

MgErr CINRun(uInt64 *guid, uInt32 *IsoSpeed, LStrHandle arg1,
    int32 *errorout)
    {
    /* Insert code here */
    dc1394camera_t *camera;
    //dc1394framerate_t framerate;
    dc1394video_mode_t video_mode ;
    dc1394color_coding_t coding;
    unsigned int i, width, height, totalbyte;
    dc1394video_frame_t *frame;
    dc1394_t * d;
    dc1394camera_list_t  *list;
    dc1394error_t err;
    IplImage* bw_image;
//-----------Umwandlung Labview String in C String------------------
//------------------------arg1 ist der LStrHandle für den filename--
    char* filename = (char*) malloc((*arg1)->cnt*sizeof(uChar)+1);
    for (i=0; i<(*arg1)->cnt; ++i)
        filename[i] = (*arg1)->str[i];
    filename[(*arg1)->cnt]=0;
//-------------------------------------------------------------------
--
 //------------------------------
    d = dc1394_new ();
    if (!d)
    {
        *errorout= 5000;
        return noErr;
    }
//Ist eine Kamera am Bus angeschlossen?
    err=dc1394_camera_enumerate (d, &list);
    if (list->num == 0)
        {
        *errorout= 5001;
        goto out1;
        }
/*-------------------------------------------------------------------
|          Inputz Variable:   guid         abfragen
-------------------------------------------------------------------*/
    camera = dc1394_camera_new (d, *guid);
    if (!camera)
        {
        //dc1394_log_error("Failed to initialize camera with guid ",
list->ids[0].guid);
        *errorout= 5002;
        goto out2;
        }
//-------------------------------------------------------------------
```

```
//----                    Bild einstelleungen                  -------
//-------------------------------------------------------------------
//1.videomode der Kamera abfragen
    err=dc1394_video_get_mode(camera,&video_mode);
    if(err)
        goto out3;
    //2.dimensionen abfragen
    err = dc1394_get_image_size_from_video_mode(camera, video_mode,
&width, &height);
    if(err)
        goto out3;
    //3. color coding abfragen
    err = dc1394_get_color_coding_from_video_mode(camera,
video_mode, &coding);
    if(err)
        goto out3;
    //4. IPL Frame einstelllen
    switch (coding)
        {
        case DC1394_COLOR_CODING_MONO8:
            bw_image = cvCreateImage(cvSize(width, height),
IPL_DEPTH_8U, 1);
            totalbyte=1;
            break;
        case DC1394_COLOR_CODING_MONO16:
            bw_image = cvCreateImage(cvSize(width, height),
IPL_DEPTH_16U, 1);
            totalbyte=2;
            break;
        case DC1394_COLOR_CODING_MONO16S:
            bw_image = cvCreateImage(cvSize(width, height),
IPL_DEPTH_16S, 1);
            totalbyte=2;
            break;
        default:
        //*cinstatus =err;
            bw_image=NULL;
            goto out3;
            break;
        }
/*-------------------------------------------------------------------
                        setup capture
 ------------------------------------------------------------------*/
    err=dc1394_video_set_iso_speed(camera, *IsoSpeed);
    if(err)
        goto out4;
    err=dc1394_capture_setup(camera,6,
DC1394_CAPTURE_FLAGS_DEFAULT);
    if(err)
        goto out4;
//-------------------------------------------------------------------
//                          start capture
//-------------------------------------------------------------------
    err=dc1394_video_set_transmission(camera, DC1394_ON);
    if(err)
        goto out5;
    err=dc1394_capture_dequeue(camera, DC1394_CAPTURE_POLICY_WAIT,
&frame);
    if(err)
        goto out5;
    memcpy(bw_image->imageData, (char *)frame-
>image,totalbyte*width*height);
//------------------------arg1 ist der String---------------------
```

```
//-----------------------------Bild abspeichern-------------------
    cvSaveImage( filename, bw_image,0);
    //den Frame wieder dem Ringbuffer zur Verfügung stellen
    err=dc1394_capture_enqueue(camera, frame);
//----------------------------------------------------------------
out5:
    if (err)
        *errorout =5100+err ;
    err = dc1394_video_set_transmission(camera, DC1394_OFF);
    err = dc1394_capture_stop(camera);
out4:
    cvReleaseImage(&bw_image);
out3:
    dc1394_camera_free(camera);
out2:
    dc1394_camera_free_list (list);
out1:
    dc1394_free (d);
    return noErr;
    }
```

10.3 Quellcode CIN Show

```
/* CIN source file */

#include "extcode.h"
#include "extcode.h"
#include <dc1394/dc1394.h>
#include <opencv/cv.h>
#include <opencv/highgui.h>
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <inttypes.h>

#define file "/tmp/labcamtest.png"

MgErr CINRun(uInt64 *guid, int32 *errorout, uInt16 *IsoSpeed);

MgErr CINRun(uInt64 *guid, int32 *errorout, uInt16 *IsoSpeed)
    {

    /* Insert code here */
    dc1394camera_t *camera;
    //dc1394framerate_t framerate;
    dc1394video_mode_t video_mode ;
    dc1394color_coding_t coding;
    unsigned int i,width, height, totalbyte;
    dc1394video_frame_t *frame;
    dc1394_t * d;
    dc1394camera_list_t  *list;
    dc1394error_t err;
    IplImage* bw_image, *preview_image;
//------------------------------
    d = dc1394_new ();
    if (!d)
    {
        *errorout= 5000;
        return noErr;
    }
//Ist eine Kamera am Bus angeschlossen?
    err=dc1394_camera_enumerate (d, &list);
    if (list->num == 0)
        {
        *errorout= 5001;
        goto out1;
        }
/*-------------------------------------------------------------------
|          Inputz Variable:   guid         abfragen
|
-------------------------------------------------------------------*/
    camera = dc1394_camera_new (d, *guid);
    if (!camera)
        {
        //dc1394_log_error("Failed to initialize camera with guid ",
list->ids[0].guid);
        *errorout= 5002;
        goto out2;
        }
//-------------------------------------------------------------------
//----                    Bild einstellungen                -------
//-------------------------------------------------------------
    //1.videomode der Kamera abfragen
    err=dc1394_video_get_mode(camera,&video_mode);
```

```
    if(err)
        goto out3;
    //2.dimensionen abfragen
    err = dc1394_get_image_size_from_video_mode(camera, video_mode,
&width, &height);
    if(err)
        goto out3;
    //3. color coding abfragen
    err = dc1394_get_color_coding_from_video_mode(camera,
video_mode, &coding);
    if(err)
        goto out3;
    //4. IPL Frame einstelllen
    switch (coding)
        {
        case DC1394_COLOR_CODING_MONO8:
            bw_image = cvCreateImage(cvSize(width, height),
IPL_DEPTH_8U, 1);
            totalbyte=1;
            break;
        case DC1394_COLOR_CODING_MONO16:
            bw_image = cvCreateImage(cvSize(width, height),
IPL_DEPTH_16U, 1);
            totalbyte=2;
            break;
        case DC1394_COLOR_CODING_MONO16S:
            bw_image = cvCreateImage(cvSize(width, height),
IPL_DEPTH_16S, 1);
            totalbyte=2;
            break;
        default:
        //*cinstatus =err;
            bw_image=NULL;
            goto out3;
            break;
        }
/*-----------------------------------------------------------------
 *                          setup capture
 *-----------------------------------------------------------------*/
    err=dc1394_video_set_iso_speed(camera, *IsoSpeed);
    if(err)
        goto out4;
    err=dc1394_capture_setup(camera,6,
DC1394_CAPTURE_FLAGS_DEFAULT);
    if(err)
        goto out4;
//-----------------------------------------------------------------
//                          start capture
//-----------------------------------------------------------------
    err=dc1394_video_set_transmission(camera, DC1394_ON);
    if(err)
        goto out5;
    err=dc1394_capture_dequeue(camera, DC1394_CAPTURE_POLICY_WAIT,
&frame);
    if(err)
        goto out5;
    memcpy(bw_image->imageData, (char *)frame-
>image,totalbyte*width*height);
//-----------------------arg1 ist der String----------------------

//-----------------------------Bild abspeichern--------------------
    cvSaveImage( file, bw_image,0);
    //den Frame wieder dem Ringbuffer zur Verfügung stellen
```

```
    err=dc1394_capture_enqueue(camera, frame);
//-------------------------------------------------------------------
out5:
    if (err)
        *errorout =5100+err ;
    err = dc1394_video_set_transmission(camera, DC1394_OFF);
    err = dc1394_capture_stop(camera);
out4:
    cvReleaseImage(&bw_image);
out3:
    dc1394_camera_free(camera);
out2:
    dc1394_camera_free_list (list);
out1:
    dc1394_free (d);
    return noErr;
    }
```

10.4 Erklärung der XML Einstellungswerte

videomode	Libdc1394 Enum Wert	Auflösung	Kodierung
64	DC1394_VIDEO_MODE_160x120_YUV444	160X120	YUV444
65	DC1394_VIDEO_MODE_320x240_YUV422	320x240	YUV422
66	DC1394_VIDEO_MODE_640x480_YUV411	640X480	YUV411
67	DC1394_VIDEO_MODE_640x480_YUV422	640X480	YUV422
68	DC1394_VIDEO_MODE_640x480_RGB8	640X480	RGB8
69	DC1394_VIDEO_MODE_640x480_MONO8	640X480	Monn8
70	DC1394_VIDEO_MODE_640x480_MONO16	640X480	Mono16
71	DC1394_VIDEO_MODE_800x600_YUV422	800x600	YUV422
72	DC1394_VIDEO_MODE_800x600_RGB8	800x600	RGB8
73	DC1394_VIDEO_MODE_800x600_MONO8	800x600	Mono8
74	DC1394_VIDEO_MODE_1024x768_YUV422	1024x768	YUV422
75	DC1394_VIDEO_MODE_1024x768_RGB8	1024x768	RGB8
76	DC1394_VIDEO_MODE_1024x768_MONO8	1024x768	Mono8
77	DC1394_VIDEO_MODE_800x600_MONO16	800x600	Mono16
78	DC1394_VIDEO_MODE_1024x768_MONO16	1024x768	Mono16
79	DC1394_VIDEO_MODE_1280x960_YUV422	1280x960	YUV422
80	DC1394_VIDEO_MODE_1280x960_RGB8	1280x960	RGB8
81	DC1394_VIDEO_MODE_1280x960_MONO8	1280x960	Mono8
82	DC1394_VIDEO_MODE_1600x1200_YUV422	1600x1200	YUV422
83	DC1394_VIDEO_MODE_1600x1200_RGB8	1600x1200	RGB8
84	DC1394_VIDEO_MODE_1600x1200_MONO8	1600x1200	Mono8
85	DC1394_VIDEO_MODE_1280x960_MONO16	1280x960	Mono16
86	DC1394_VIDEO_MODE_1600x1200_MONO16	1600x1200	Mono16
87	DC1394_VIDEO_MODE_EXIF	EXIF	
88	DC1394_VIDEO_MODE_FORMAT7_0	Format7	
89	DC1394_VIDEO_MODE_FORMAT7_1	Format7	
90	DC1394_VIDEO_MODE_FORMAT7_2	Format7	
91	DC1394_VIDEO_MODE_FORMAT7_3	Format7	
92	DC1394_VIDEO_MODE_FORMAT7_4	Format7	
93	DC1394_VIDEO_MODE_FORMAT7_5	Format7	
94	DC1394_VIDEO_MODE_FORMAT7_6	Format7	
95	DC1394_VIDEO_MODE_FORMAT7_7	Format7	

Abbildung 35 Tabelle der möglichen Werte videomode

```
<video videomode="81" framerate="35"/>
```

Codeauszug 1 Beispiel des Wertes „videomode“ aus der XML Kameraeinstellungsdatei

framerate	Libdc1394 Enum Wert	Bilder pro Sekunde
32	DC1394_FRAMERATE_1_875	1,875
33	DC1394_FRAMERATE_3_75	3,75
34	DC1394_FRAMERATE_7_5	7,5
35	DC1394_FRAMERATE_15	15
36	DC1394_FRAMERATE_30	30
37	DC1394_FRAMERATE_60	60
38	DC1394_FRAMERATE_120	120
39	DC1394_FRAMERATE_240	240

Abbildung 36 Tabelle der möglichen Werte framerate

```
<video videomode="81" framerate="35"/>
```

Codeauszug 2 Beispiel des Wertes „framerate“ aus der XML Kameraeinstellungsdatei

fid	*Libdc1394 Enum Wert*	*Beschreibung*
0	DC1394_FEATURE_BRIGHTNESS	Brightness Control
1	DC1394_FEATURE_EXPOSURE	Auto Exposure Control
2	DC1394_FEATURE_SHARPNESS	Sharpness Control
3	DC1394_FEATURE_WHITE_BALANCE	White Balance Control
4	DC1394_FEATURE_HUE	Hue Control
5	DC1394_FEATURE_SATURATION	Saturation Control
6	DC1394_FEATURE_GAMMA	Gamme Control
7	DC1394_FEATURE_SHUTTER	Shutter Speed Control
8	DC1394_FEATURE_GAIN	Gain Control
9	DC1394_FEATURE_IRIS	Iris Control
10	DC1394_FEATURE_FOCUS	Focus Control
11	DC1394_FEATURE_TEMPERATURE	Temperature Control
12	DC1394_FEATURE_TRIGGER	Trigger Control
13	DC1394_FEATURE_TRIGGER_DELAY	Trigger Delay Control
14	DC1394_FEATURE_WHITE_SHADING	White Shading Compensation Control
15	DC1394_FEATURE_FRAME_RATE	Frame rate prioritize control
16	DC1394_FEATURE_ZOOM	Zoom control
17	DC1394_FEATURE_PAN	Pan Control
18	DC1394_FEATURE_TILT	Tilt Control
19	DC1394_FEATURE_OPTICAL_FILTER	Optical Filter Control
20	DC1394_FEATURE_CAPTURE_SIZE	Capture image size for Format_6
21	DC1394_FEATURE_CAPTURE_QUALITY	Capture image quality for Format_6

Abbildung 37 Tabelle der möglichen Werte fid

```
<FeaturesSettings>
  <feature fvalue="0" fid="12">Trigger</feature>
</FeaturesSettings>
```

Codeauszug 3 Beispiel des Wertes „fid“ aus der XML Kameraeinstellungsdatei

10.5 IsoSpeed Werteerklärung

IsoSpeed	*Libdc1394 enum Wert*	*Übertragungsgeschwindigkeit*
0	DC1394_ISO_SPEED_100	100 Mbit/s
1	DC1394_ISO_SPEED_200	200 Mbit/s
2	DC1394_ISO_SPEED_400	400 Mbit/s (standard)
3	DC1394_ISO_SPEED_800	800 Mbit/s
4	DC1394_ISO_SPEED_1600	1600 Mbit/s
5	DC1394_ISO_SPEED_3200	3200 Mbit/s

Abbildung 38 Tabelle der möglichen Werte für den isochronen Übertragungsmodus

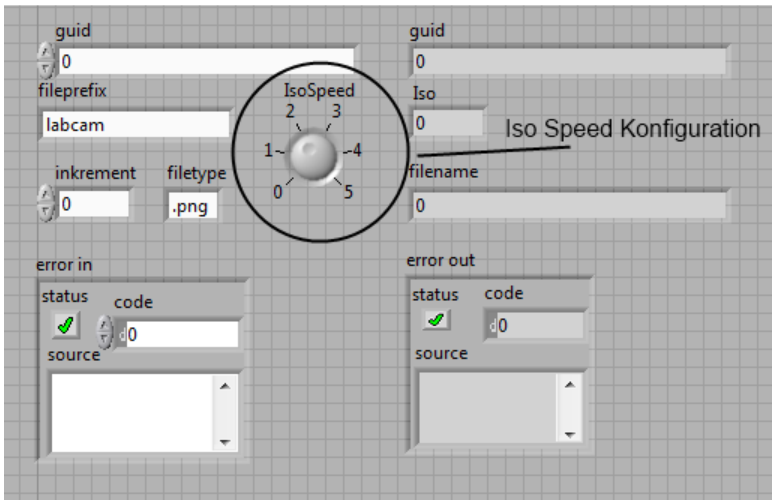

Abbildung 39 Auswahl des isochronen Übertragungsmoduses

10.6 Custom Error Codes Liste

Error Code	Beschreibung
5000	Could not initialisize libdc1394
5001	No Cameras found
5002	No Camera found with the given guid
5020	Failed to parse XML File.
5021	Cannot find the root Element in the XML file
5022	Cannot find root Element "firecam"
5099	DC1394_FAILURE
5098	DC1394_NO_FRAME
5097	DC1394_NO_CAMERA
5096	DC1394_NOT_A_CAMERA
5095	DC1394_FUNCTION_NOT_SUPPORTED
5094	DC1394_CAMERA_NOT_INITITIALIZED
5093	DC1394_INVALID_FEATURE
5092	DC1394_INVALID_FORMAT
5091	DC1394_INVALID_MODE
5090	DC1394_INVALID_FRAMERATE
5089	DC1394_INVALID_TRIGGER_MODE
5088	DC1394_INVALID_TRIGGER_SOURCE
5087	DC1394_INVALID_ISO_SPEED
5086	DC1394_INVALID_IIDC_VERSION
5085	DC1394_INVALID_COLOR_MODE
5084	DC1394_INVALID_FORMAT7_COLOR_TILE
5083	DC1394_REQ_VALUE_OUTSIDE_RANGE
5082	DC1394_INVALID_ERROR_CODE
5081	DC1394_MEMORY_ALLOCATION_FAILURE
5080	DC1394_TAGGED_REGISTER_NOT_FOUND
5079	DC1394_FORMAT7_ERROR_FLAG_1
5078	DC1394_FORMAT7_ERROR_FLAG_2
5077	DC1394_INVALID_BAYER_METHOD
5076	DC1394_HANDLE_CREATION_FAILURE
5075	DC1394_GENERIC_INVALID_ARGUMENT
5074	DC1394_INVALID_VIDEO1394_DEVICE
5073	DC1394_NO_ISO_CHANNEL
5072	DC1394_NO_BANDWIDTH
5071	DC1394_IOCTL_FAILURE
5070	DC1394_CAPTURE_IS_NOT_SET
5069	DC1394_CAPTURE_IS_RUNNING
5068	DC1394_RAW1394_FAILURE
5067	DC1394_BASLER_NO_MORE_SFF_CHUNKS
5066	DC1394_BASLER_CORRUPTED_SFF_CHUNK
5065	DC1394_BASLER_UNKNOWN_SFF_CHUNK

Abbildung 40 Tabelle aller Custom Error Codes