Party Licht Steuerung – Programmentwurf für Lichttechniker mit LabView

Studienarbeit

für die Prüfung zum
Bachelor of Engineering

im Studiengang TIT08I
an der Dualen Hochschule Baden-Württemberg Mosbach

von

Tim Berger

17. Juni 2011

Bearbeitungszeitraum:	6. Theoriephase
Matrikelnummer:	115435
Ausbildungsfirma:	Kurtz Holding GmbH & Co. Beteiligungs KG
Gutachter der DHBW Mosbach:	Prof. Dr. Wolfgang Funk

Zusammenfassung

Abstract

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

LabView LV

LJ Light Jockey (dt. Lichttechniker)

NI National Instruments

1 Einleitung

Diese Studienarbeit dokumentiert den Programmentwurf einer Party-Lichtsteuerung, vom Design der Anwendung über die Implementierung bis hin zur Bereitstellung einer lauffähigen Applikation. Die Programmcode wird mit LabVIEW von National Instruments entwickelt.

1.1 Aufgabenstellung

Es ist ein Programm für Lichttechniker zu entwickeln, mit dem eine viel zahl von Scheinwerfer angesteuert werden kann.

1.2 Anforderungen

Der Light Jockey (LJ) stellt für verschiedene Lichtkanäle Intensität und Farbe ein. Für eine Gruppe von Lichkanälen (Set) kann eine Wartezeit, Überblendungszeit, Nachlaufzeit und Name eingestellt werden. Wählt der LJ die Schaltfläche zum aufnehmen, öffnet sich ein Fenster in dem die gewünschten Parameter übergeben werden. Nach der Bestätigung durch ein Klick auf die OK-Schaltfläche wird das erstellte Set hinten an die Queue angefügt.
Hat der Bediener einige Sets angelegt, wird mit der Abspiel-Schaltfläche das aufgenommene Programm durchlaufen. Ein Set das abgespielt wird wartet die angegebene Zeit, dann wird die Farbe bis zur Intensität über die Überblendungszeit hochgefahren. Jetzt beginnt die Nachlaufzeit, ist diese verstrichen wird mit dem nächsten Set aus der Queue fortgefahren. Der Bediener kann jeder Zeit eine abspielende Queue mit der Stopp-Schaltfläche anhalten.
Über die Menüleiste kann der Bediener mit dem Menüpunkt „Datei“ die Queue speichern und laden. In beiden Fällen öffnet sich eine Dialogbox in dem nach Speicher- bzw. Ladepfad gefragt wird.

1.3 Aufbau der Arbeit

Bla Bla Blaaa

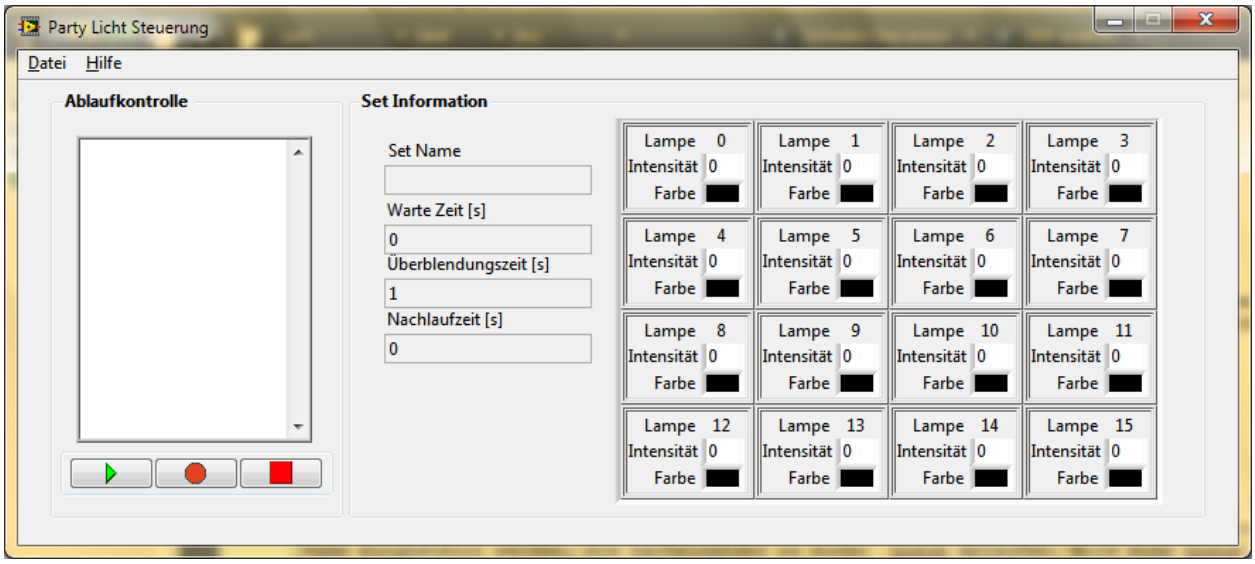

Abbildung 1: Bedienoberfläche für die Party-Lichtsteuerung

2 LabVIEW als Programmiersprache

LabVIEW ist ein grafisches Programmiersystem von National Instruments. Das Akronym steht für „Laboratory Virtual Instrumentation Engineering Workbench“. Die Programmierung erfolgt in der graphischen Programmiersprache „G“. LabVIEW-Programme werden als Virtuelle Instrumente (VIs) bezeichnet. [NI11a] [**?**] Sie bestehen aus drei Komponenten:

Frontpanel Das User-Interface, über welches der Anwender mit dem VI interagiert.

Blockdiagramm Stellt den Programmcode des VIs dar.

Anschluss Dient zur Anbindung an weitere VIs. Bestimmt Übergabe und Rückgabe Werte.

In LabVIEW liegt die Ausführung von VIs dem Datenflussmodell zugrunde. Ein Blockdiagrammknoten (Bsp. Addition) wird ausgeführt, sobald all seine Eingänge belegt sind. Ist die Ausführung eines Knotens abgeschlossen, werden die Daten an die Ausgabeanschlüsse übergeben und die Ausgabedaten dann an den nächsten Knoten im Datenflussdiagramm weitergeleitet. [**?**] Die unter LabVIEW erstellten Blockdiagramme werden von einem grafischen Compiler in optimierten Maschinencode übersetzt. Dadurch ist die Performance vergleichbar mit der anderer Hochsprachen wie C oder Pascal. [NI11b]
Abbildung 2 zeigt eine kleine Demonstration. Es wird aus den Eingängen A und B ein Ausgang C berechnet. Die Formel wird im Blockdiagramm abgebildet. Sie lautet:

$$C = \frac{A + B^2}{4}$$

Des Weiteren findet die Berechnung in einer While-Schleife statt. Die Abbruchbedingung ist die Betätigung der Stopp-Schaltfläche.

2.1 Entwurfsmuster - Design Pattern

Zur Entwicklung einer umfangreichen Applikation ist es unerlässlich mit Entwurfsmustern zu arbeiten. Sie helfen nicht nur dem Entwickler den Überblick nicht zu verlieren sondern machen es auch für Außenstehende einfacher den Code zu lesen und modifizieren.
LabView bietet neun verschiedene Entwurfsmuster. Für welches man sich entscheidet hängt von folgenden Kriterien ab:

- Gibt es eine feste Reihenfolge / Sequenzen von Befehlen?

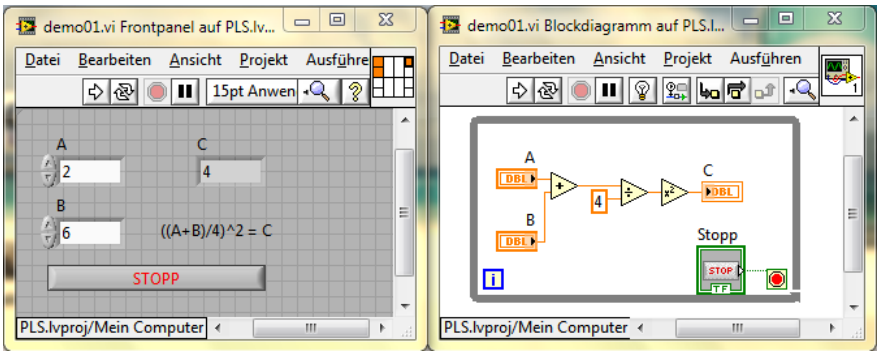

Abbildung 2: VI Demonstration: Links Frontpanel, Rechts Blockdiagramm

- Muss das Programm mit einem User-Interface agieren?
- Ist die Datenverarbeitung intensiv?
- Gibt es parallele Operationen?

Im folgenden gehe ich auf einige Entwurfsmuster ein, die für meine Problemstellung infrage kommen könnten. Das sind: der Zustandsautomat, Master/Slave-Entwurfsmuster, Ereignisbehandler für Benutzeroberfläche und das Erzeuger/Verbraucher-Entwurfsmuster. Später im Abschnitt 4.1 werde ich auf meine Wahl des Entwurfsmuster eingehen.

2.1.1 Zustandsautomat

Mit jedem Zustand wird ein bestimmter Blockdiagramm Ausschnitt ausgeführt und ermittelt, zu welchem Zustand weitergesprungen wird. In einer While-Schleife wird eine Case-Struktur ausgeführt.

2.1.2 Master/Slave-Entwurfsmuster

Bei diesem Entwurfsmuster gibt es eine Master-Schleife und mindestens eine Slave-Schleife. Die Master Schleife wird immer ausgeführt. Sie benachrichtigt Slave-Schleifen, einen bestimmten Code auszuführen. Die Slave-Schleifen werden vollständig ausgeführt und warten dann auf die nächste Benachrichtigung.

2.1.3 Einfacher Ereignisbehandler für Benutzeroberfläche

Dieses Entwurfsmuster wird verwendet für die Verarbeitung von Ereignissen der Benutzeroberfläche. Die Vorlage eignet sich für Dialogfelder und andere Programmoberflächen.

Des Weiteren kann man benutzerdefinierte Ereignisse erzeugen und ausführen, die wie Ereignisse der Benutzeroberfläche behandelt werden.

2.1.4 Erzeuger/Verbraucher-Entwurfsmuster

Hier werden zwei separate While-Schleifen unabhängig voneinander ausgeführt: Die erste Schleife erzeugt Daten, während die zweite Schleife die Daten verarbeitet. Obwohl sie parallel ausgeführt werden, werden zwischen den Schleifen über Queues Daten ausgetauscht. Diese Vorlage bietet die Möglichkeit bei Benutzereingriffen asynchron Code auszuführen, ohne die Reaktionszeit der Benutzeroberfläche zu beeinträchtigen. So kann man durch die parallele Ausführung der Schleifen Leistungssteigerung des Programms erzielen.

3 Programm Analyse

3.1 Ablaufdiagramm

Mit Ablaufdiagrammen wird der Programmfluss illustriert. Mit deren Hilfe kann man eine Aufgabe in handhabbare Funktionen teilen. Abbildung 3 zeigt das Ablauf Diagramm für die Abspiel-Funktion in der Party-Licht-Steuerung. Ein Knoten repräsentiert einen Zustand.

3.2 Datenfluss Diagramm

Datenfluss Diagramme haben die Aufgabe zu zeigen, welchen Weg die Daten durch eine Applikation nehmen. Abbildung 4 zeigt das Datenfluss Diagramm für die Abspiel-Funktion in der Party-Licht-Steuerung. Die Knoten (Kreise) repräsentieren die Prozesse. Eine externe Entität ist das Licht Kontroll-System. Die Pfeile zeigen die Richtung des Datenflusses an.

3.3 Datentypen

Lichterset Cluster ...

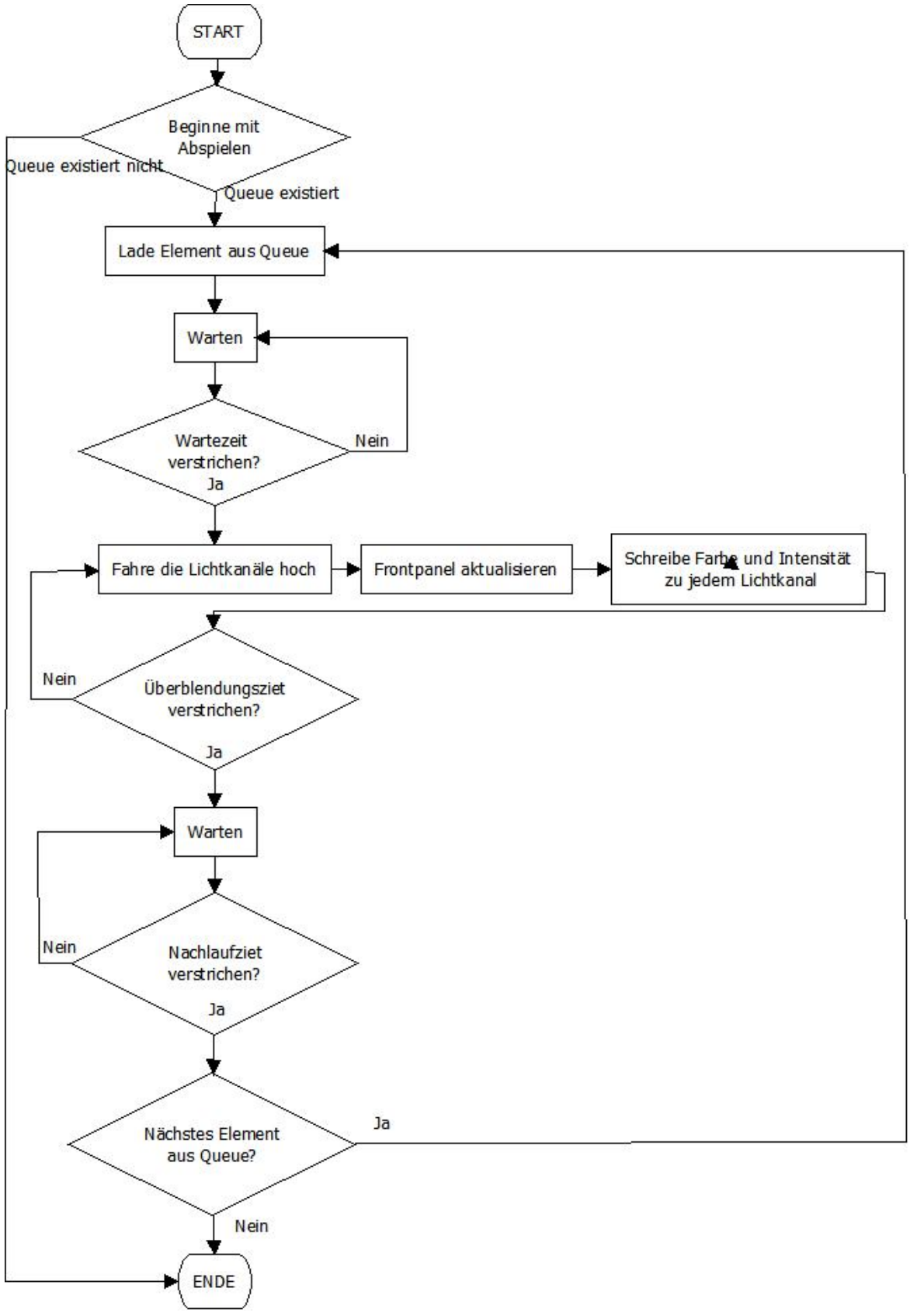

Abbildung 3: Ablaufdiagramm für die Abspiel-Funktion

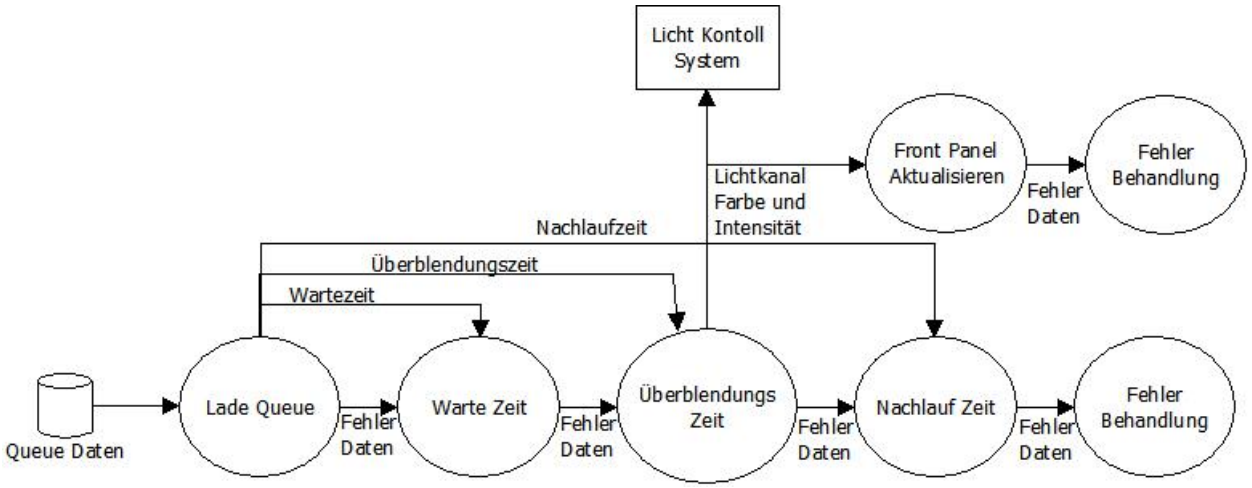

Abbildung 4: Datenfluss Diagramm für die Abspiel-Funktion

4 Code Implementierung

4.1 Auswahl des Design Pattern

Bei der Wahl des Entwurfsmusters habe ich mich für das Erzeuger/Verbraucher Design (siehe Abschnitt 2.1) entschieden. Bei diesem Pattern kann man die Ereignisbehandlung vom User-Interface und den auszuführenden Code gut trennen.
Das Entwurfsmuster wird wie folgend abgewandelt implementiert. Die Erzeuger Schleife reagiert auf Events vom User-Interface. Diese sind die drei Schaltflächen: Abspielen, Aufnehmen und Stoppen sowie die Menüauswahl: Speichern, Laden und Beenden.
Über eine Verbraucher-Queue tauscht die Erzeugerschleife Kommandos und Daten mit der Verbraucherschleife aus. Hier sind folgende Kommandos implementiert:

- initialisieren
- aufnehmen
- abspielen
- stoppen
- laden
- speichern und
- beenden

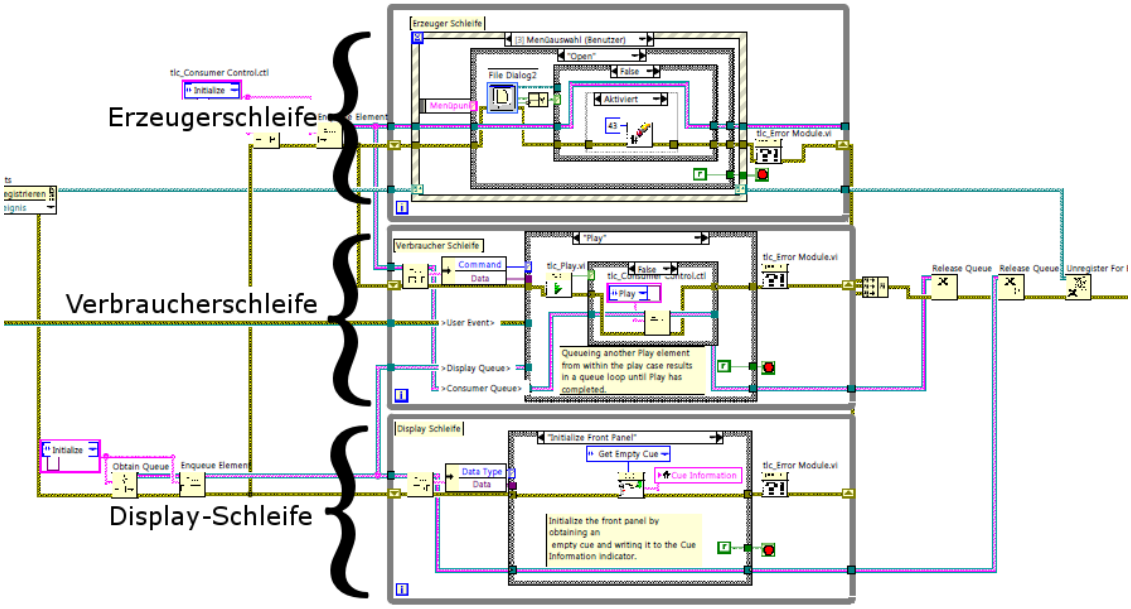

Abbildung 5: Design Struktur für das Blockdiagramm

Daten die die Erzeuger- an die Verbraucherschleife sendet sind Lampen-Sets die beim aufnehmen, speichern oder laden entstehen. In der Verbraucherschleife werden alle Berechnungen durchgeführt. Diese Schleife kommuniziert über eine Display-Queue mit der Displayschleife.
Sie hat die Aufgabe Änderungen am User-Interface durchzuführen. Diese können sein:

- Initialisierung des Front Panels
- Update der Lichtkanäle
- Auswahl eines Lichtsets
- De-/Aktivieren von Schaltflächen
- Update der Set-Ablaufliste und
- Stopp

Abbildung 5 zeigt die Struktur aus dem Blockdiagramm. Die Funktionen der Applikation sind in drei separate Schleifen aufgeteilt, der Vorteil dieser Architektur ist, das die Funktionalität der einzelnen Prozesse parallel ausgeführt werden kann. Des Weiteren ist diese Art der Architektur wartungsfreundlicher und besser skalierbar.

4.2 Init und Shutdown Funktion

Die Initialisierungsfunktion wird beim Start der Anwendung ausgeführt. Die setzt alle Module in einen definierten, sicheren Zustand und säubert das User-Interface. Abbildung 9 im Anhang zeigt das Blockdiagramm vom *„init.vi“*.
Wenn der Benutzer im Menü auf Datei→Beenden klickt wird die Anwendung sicher heruntergefahren, dass heißt es werden alle Speicher-Referenzen freigegeben. Abbildung 10 im Anhang zeigt das Blockdiagramm vom *„shutdown.vi“*.

4.3 User Interface

Die Display-Schleife sorgt für Updates auf dem User Interface. Abbildung 6 zeigt diese mit dem Stopp Case. Zu beginn wird ein Element aus der Display Queue genommen und dann nach Typ und Daten aufgeschlüsselt. Anhand des Datentyps, das die sie von der Verbraucherschleife erhalten wird der entsprechende Case aufgerufen. Bei dem Kommando Stopp wird die While-Schleife beendet. Sollte ein Fehler in einem Case aufgetreten sein, wird es vom *„Error Module.vi“* behandelt. Dazu später mehr, im Abschnitt XX Fehlerbehandlung. Im folgenden werden die einzelnen Cases in der Display-Schleife erläutert.

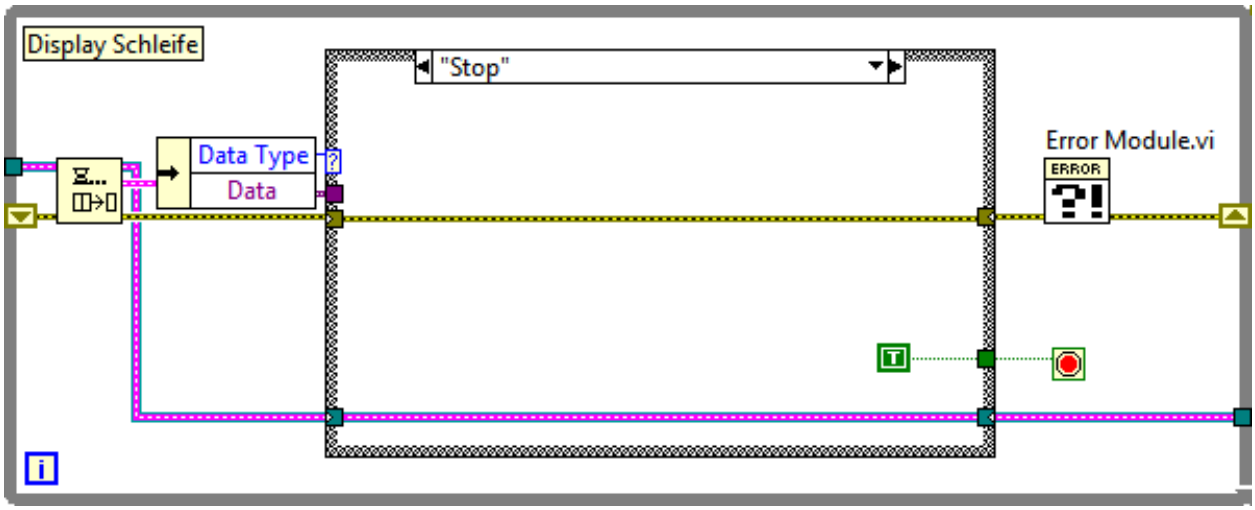

Abbildung 6: Display Schleife

4.3.1 Initialisierung des Front Panels

Dieser Case erstellt ein 2D-Array von Lichtkanälen und stellt es auf dem Front Panel dar. Jeder Kanal bekommt eine Nummer. Die Farbe wird auf schwarz und die Intensität auf 0

gesetzt.
Der entsprechende Ausschnitt aus dem Programmcode ist im Anhang auf Abbildung 11.

4.3.2 Auswahl eines Lichtsets aus der Lichterset Queue

In diesem Case wird eine Spalte in Ablaufkontrolle hervorgehoben. Entweder wenn der User darauf klickt oder wenn die Applikation beim Abspielen über die Queue von Lichtersets iteriert.
Der entsprechende Ausschnitt aus dem Programmcode ist im Anhang auf Abbildung 12.

4.3.3 De-/Aktivieren von Schaltflächen

Wenn eine Lichterset Queue abgespielt wird, schaltet dieser Case die Schaltfläche Aufnehmen und die Ablaufkontrollliste inaktiv. Das verhindert, das der Nutzer während eines Abspielvorgangs ein neues Lichterset anlegt oder die Ablaufkontrollliste durcheinander bringt.
Der entsprechende Ausschnitt aus dem Programmcode ist im Anhang auf Abbildung 13.

4.3.4 Update der Set-Ablaufliste

Dieser Case aktualisiert die Liste in der Ablaufkontrolle immer wenn ein neues Lichterset aufgenommen wurde.
Der entsprechende Ausschnitt aus dem Programmcode ist im Anhang auf Abbildung 14.

4.3.5 Update der Lichtkanäle

Dieser Case updatet das 2D-Array aus Lichtkanälen. Es wird immer dann aufgerufen, wenn sich Lichtkanal Daten ändern. Das ist der Fall wenn eine Lichterset Queue abgespielt wird und sich die Farbe und Intensität der Kanäle ändert oder der Nutzer auf ein Element in der Ablaufkontolle klickt um sich Informationen zum ausgewählten Lichterset anzuzeigen.
Der entsprechende Ausschnitt aus dem Programmcode ist im Anhang auf Abbildung 15.

4.4 Aufnahme-Funktion

Klickt der Anwender auf die Aufnahme Schaltfläche öffnet sich eine Dialogbox in der er nach Parametern für das zu erstellende Lichterset gefragt wird. Die einzugebenden Werte

sind:

- Setname
- Wartezeit
- Überblendungszeit
- Nachlaufzeit und
- Einstellungen für die einzelnen Lichtkanäle

Nach Bestätigung über die OK-Schaltfläche werden die gesammelten Daten in die Queue geschrieben und das User-Interface geupdated.
Die Abbildung 7 zeigt die Erzeuger-(oben) und Verbraucherschleife(unten). In der Erzeugerschleife wird die Dialogbox geöffnet. Sie gibt ein Objekt mit den gesammelten Daten zurück. Wurde der Dialog nicht abgebrochen werden die Daten zusammen mit dem Kommando „record“ in die Verbraucher Queue gesteckt. Die Verbraucherschleife nimmt sich das Objekt aus der Queue und hängt das neue Objekt an die Lichterset-Liste an. Dann gibt sie das Kommando zum updaten des User-Interface an die Display-Schleife.

4.5 Timing

Zur akkuraten Berechnung der Warte-, Überblendungs- und Nachlaufzeit beim Abspielen der Lichtersets dient das Timing VI. Dieses VI arbeitet als funktionale globale Variable (FGV).

4.5.1 Funktional globale Variable

In FGVs können in nicht initialisierten Schieberegistern von While- oder For-Schleifen Daten gehalten werden. Die Daten bleiben erhalten, solange sich das zugehörige VI im Speicher befindet. Die jeweilige Schleife in einer FGV wird bei einem Aufruf nur einmal durchlaufen. Liegen die Daten am Ende der Schleife (rechts) im Schieberegister an, stehen diese beim nächsten Aufruf wieder am Anfang (links) an.
Das hat den Vorteile das Daten nicht immer bei jedem VI Aufruf mit geführt werden müssen. Die Daten werden einmal beim initialisieren mitgegeben halten sich dann im Speicher.
Die Timing FGV hat zwei Kommandos: starten bzw. initialisieren und checken. Die Auswahl wird über eine Switch-Case-Anweisung getroffen. Soll eine Zeit gemessen werden,

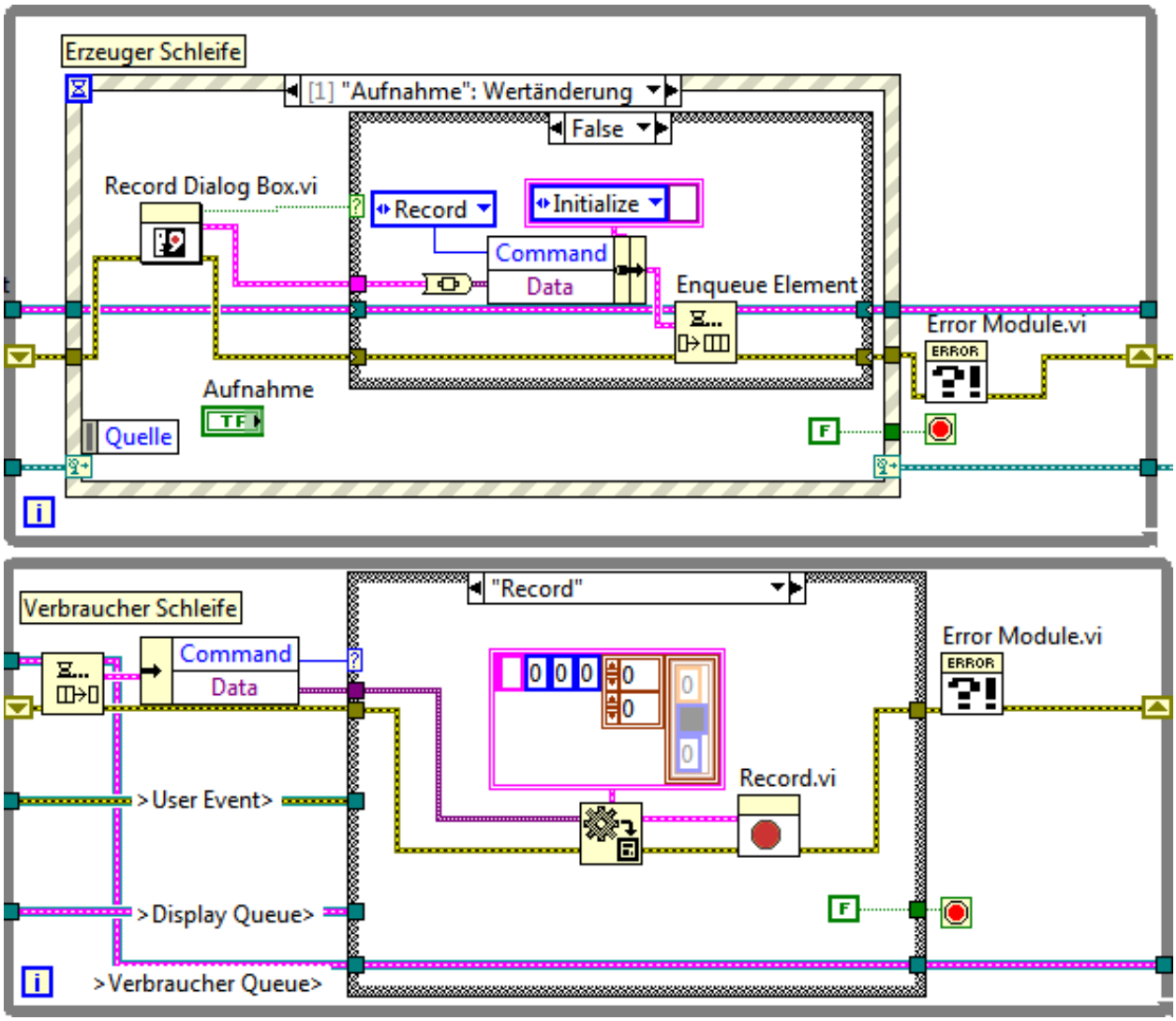

Abbildung 7: Aufnahme-Funktion

wird zu beginn der Messung das Timing VI mit der Ziel-Zeit und dem Kommando Start aufgerufen. Hier wird die aktuelle Systemzeit in Sekunden ermittelt (Startzeit) und ins Schieberegister geschrieben. In der Applikation wird in einer Schleife dann immer mit dem Kommando check abgefragt ob die Zeit abgelaufen ist. Ist das der Fall, gibt die FGV am Ausgang „Verstrichen“ True zurück, sonst False. Abbildung 8 zeigt die FGV mit ihren beiden Cases.

4.6 Abspiel-Funktion

Die Abspiel-Funktion wird als Zustandsautomat implementiert. Das Flussdiagramm aus Abbildung 3 zeigt die einzelnen Zustände.

Empfängt die Verbraucherschleife das Kommando zum Abspielen öffnet sie das *„re-*

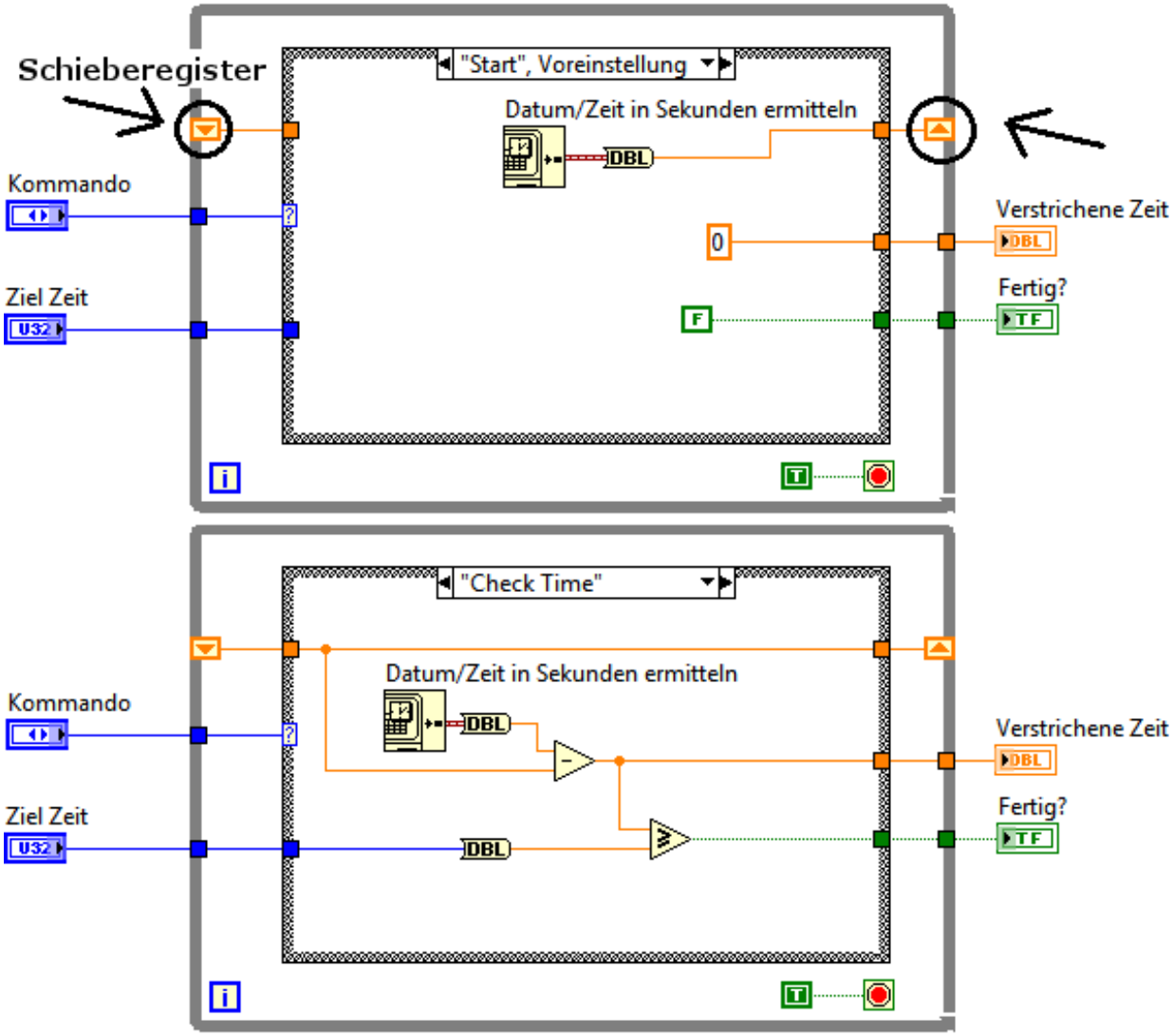

Abbildung 8: FGV Timing VI

cord.vi“. Das VI durchläuft ein Zustand. Gibt es zurück, das der Zustandsautomat noch nicht bis zum Ende durchgelaufen ist, steckt die Verbraucherschleife erneut das Kommando play in die Verbraucher Queue. Daraus resultiert eine Schleife in der bei jeder iteration ein Zustand durchläuft, solange bis der Zustandsautomat am Ende ist. Zur Berechnung, ob die verschieden Zeiten abgelaufen sind wird das *„timing.vi“* aufgerufen. Im Anhang in Abbildung 17 findet sich der Code für den Zustand Überblenden.

4.7 Stopp-Funktion

Will der Bediener den Abspiel-Vorgang abbrechen, klickt er auf die Stopp Schaltfläche. Jetzt wird der Zustandsautomaten der Abspiel-Funktion unterbrochen. Das geschieht indem die Verbraucher Queue geleert und der Zustandsautomat zurückgesetzt wird. So

wird garantiert, das keine Nachrichten mehr in der Verbraucher-Queue sind und der Zustandsautomat beim nächsten Anlauf wieder am Anfang startet. Der Code kann im Anhang bei Abbildung 18 gefunden werden.

4.8 Speichern und Lade Funktion

Die Funktionalität zum speichern und laden in eine Datei ist für ein LJ unerlässlich. So kann er während einer Probe alle Einstellungen setzen und diese in eine Datei schreiben. Zum Zeitpunkt des Events muss die Datei nur noch geladen werden.
Zum speichern oder laden klickt der LJ in die Menüleiste unter Datei auf speichern bzw. laden. Die Applikation (Erzeugerschleife) öffnet ein File-Dialog und prüft ob die ausgewählte Datei existiert. Dann schickt sie das entsprechende Kommando (load oder save) zusammen mit dem Dateipfad an die Verbraucherschleife.

4.8.1 In Datei speichern

In der Verbraucherschleife wird über eine For-Schleife ein 1D-Array aus Lichtersets erstellt. Dieses wird an das *„File-Modul.vi“* übergeben. Hier wird das 1D-Array in die angegebene Datei geschrieben. Hierfür die vorgefertigten LabView VIs genutzt: Öffnen, Schreiben und Schließen einer Datei. Der Code ist im Anhang unter Abbildung 19.

4.8.2 Aus Datei laden

Um aus einer Datei Elemente lesen zu können muss man der Lese-Funktion den Lichterset Datentyp mitgeben. Zurück bekommt man ein 1D-Array aus Lichtersets. Diese wird dann in die Lichterset Queue geschrieben. Der Code ist im Anhang unter Abbildung 20.

4.9 Fehlerbehandlung

5 Testen

6 Anwendung

6.1 Stand-Alone Applikation

6.2 Installer

6.3 Webservice

7 Abschließende Betrachtung

7.1 Multilingualität

7.2 Update

7.3 Information Hiding

7.4 Erweiterungen

Hardware Ansteuerung, Visualisierung größer

7.5 Probleme

Stop Button

Literatur

[NI11a] NI: *Einführung in LabVIEW - Dreistündiger Einführungskurs*. http://www.ni.com/pdf/academic/d/labview_3_hrs.pdf, 2011. Zugriff am 2.6.2011 um 14:27 in Datei *„labview3hrs.pdf“*.

[NI11b] NI: *Wie funktioniert der Compiler von NI LabVIEW?* http://zone.ni.com/devzone/cda/tut/p/id/11936, 2011. Zugriff am 2.6.2011 um 14:27 in Datei *„Compiler LabVIEW - Developer Zone - National Instruments.pdf“*.

Anhang

A.1 Initialisierungsfunktion

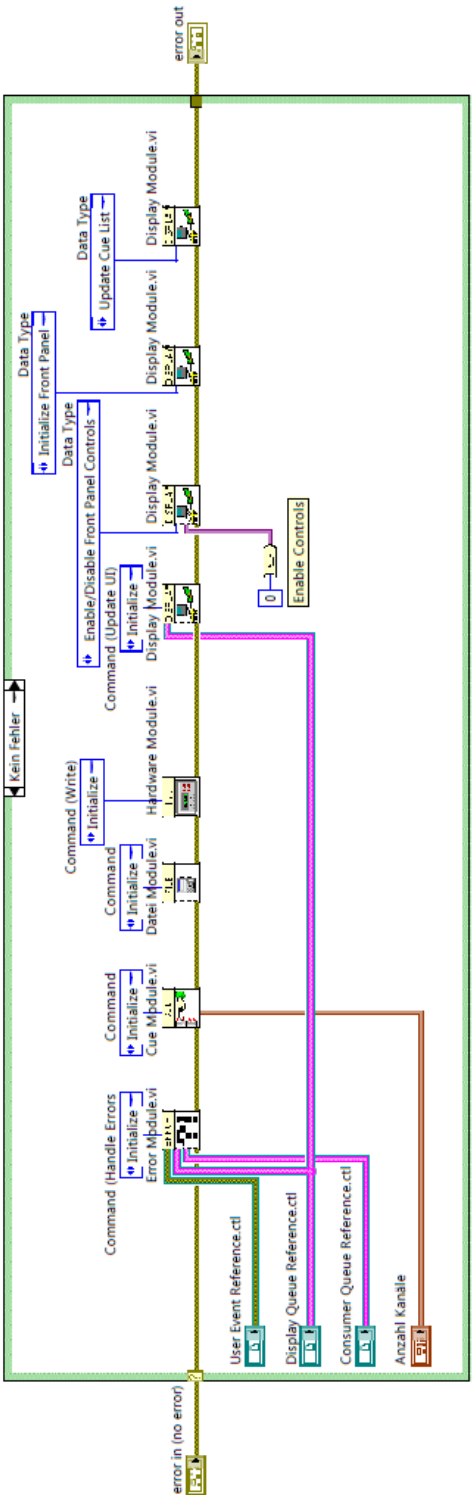

Abbildung 9: Initialisierungsfunktion

A.2 Shutdown-Funktion

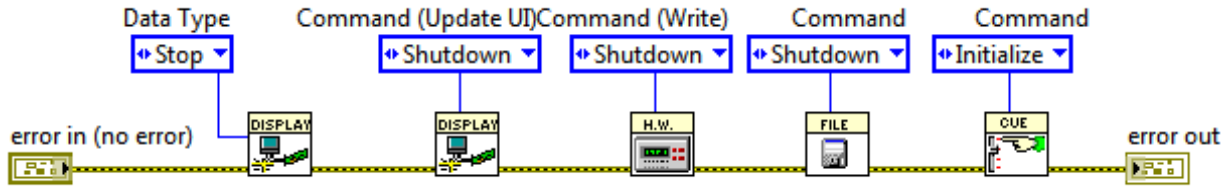

Abbildung 10: Shutdown-Funktion

A.3 Initialisierung des Front Panels

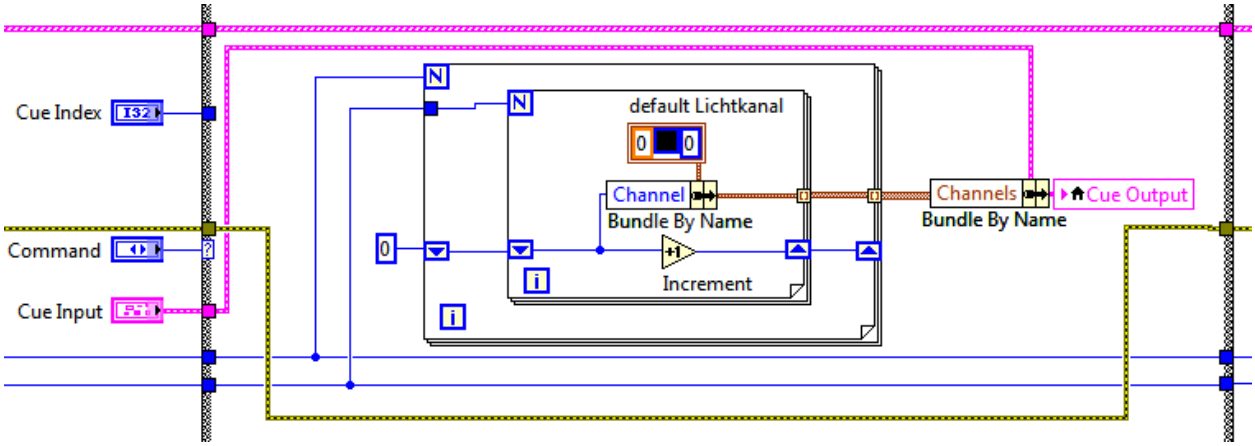

Abbildung 11: Initialisierung des Front Panels

A.4 Auswahl eines Lichtsets aus der Lichterset Queue

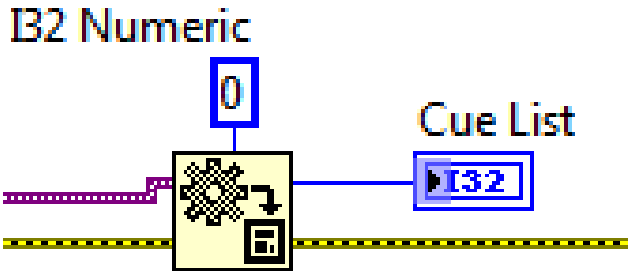

Abbildung 12: Auswahl eines Lichtsets aus der Lichterset Queue

A.5 De-/Aktivieren von Schaltflächen

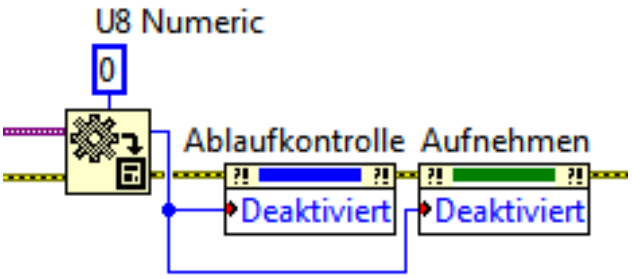

Abbildung 13: De-/Aktivieren von Schaltflächen

A.6 Update der Set-Ablaufliste

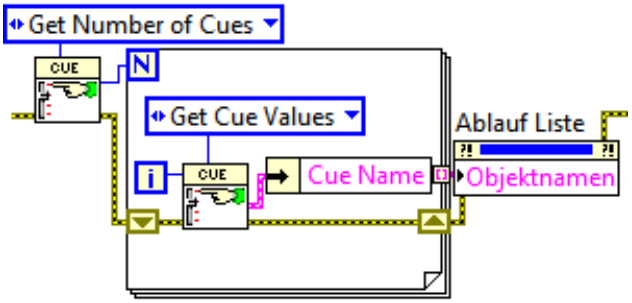

Abbildung 14: Update der Set-Ablaufliste

A.7 Update der Lichtkanäle

A.8 Timing Modul

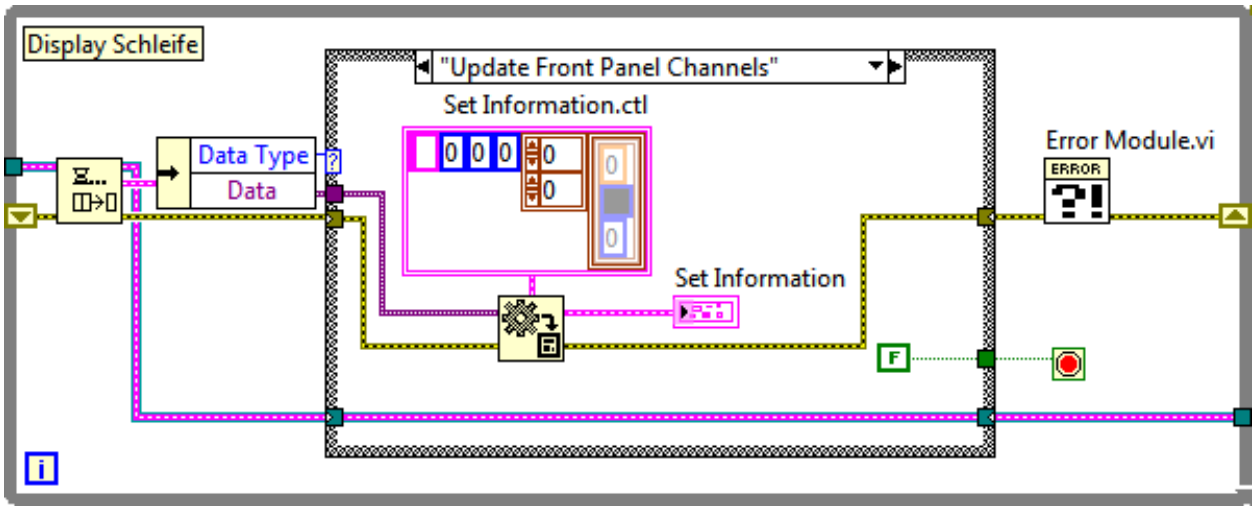

Abbildung 15: Update der Lichtkanäle

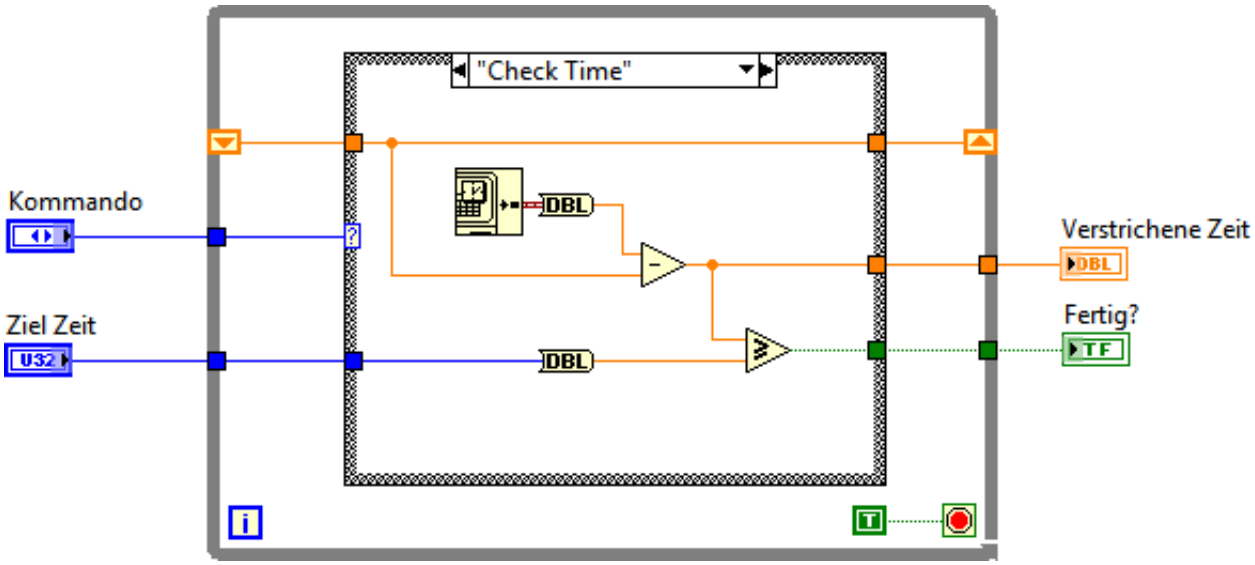

Abbildung 16: Timing Modul

A.9 Abspiel Zustandsautomat - Überblenden

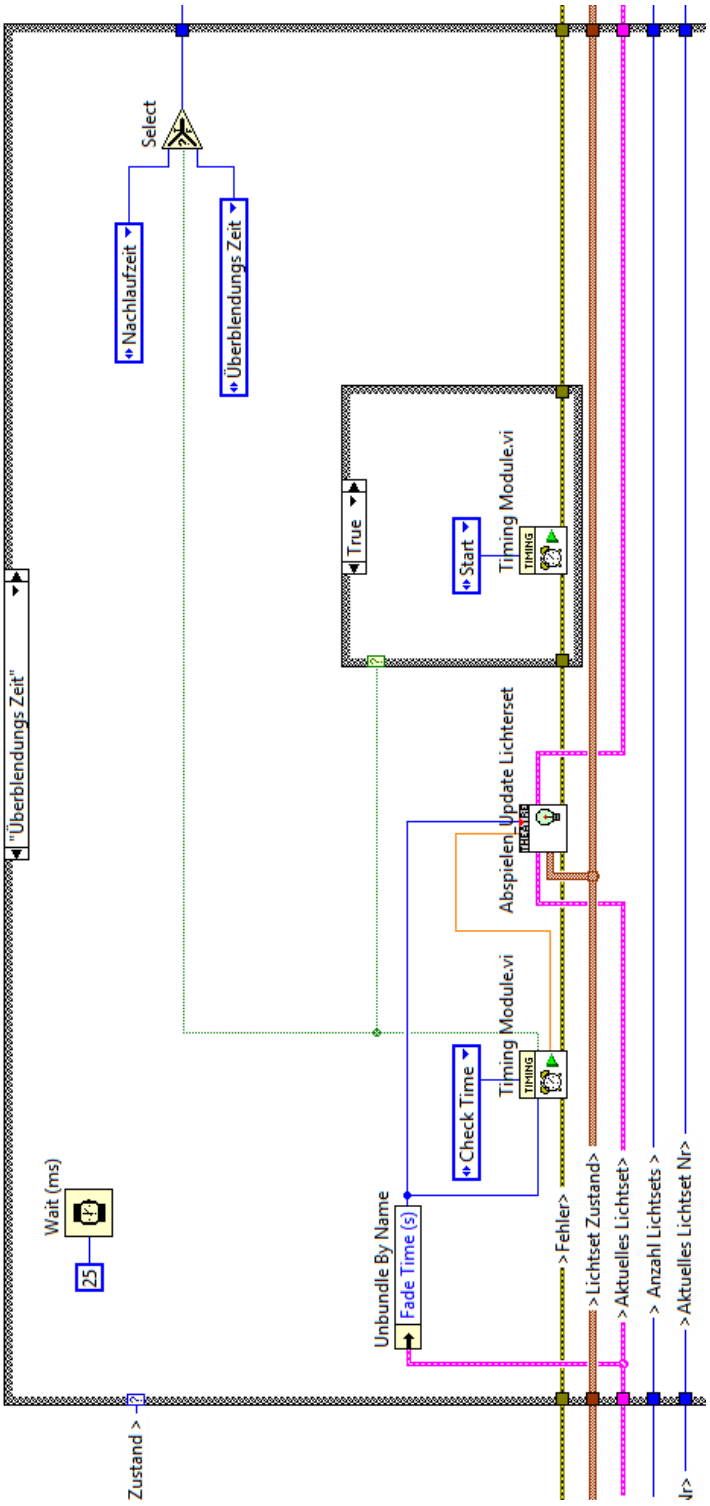

Abbildung 17: Abspiel Zustandsautomat - Überblenden

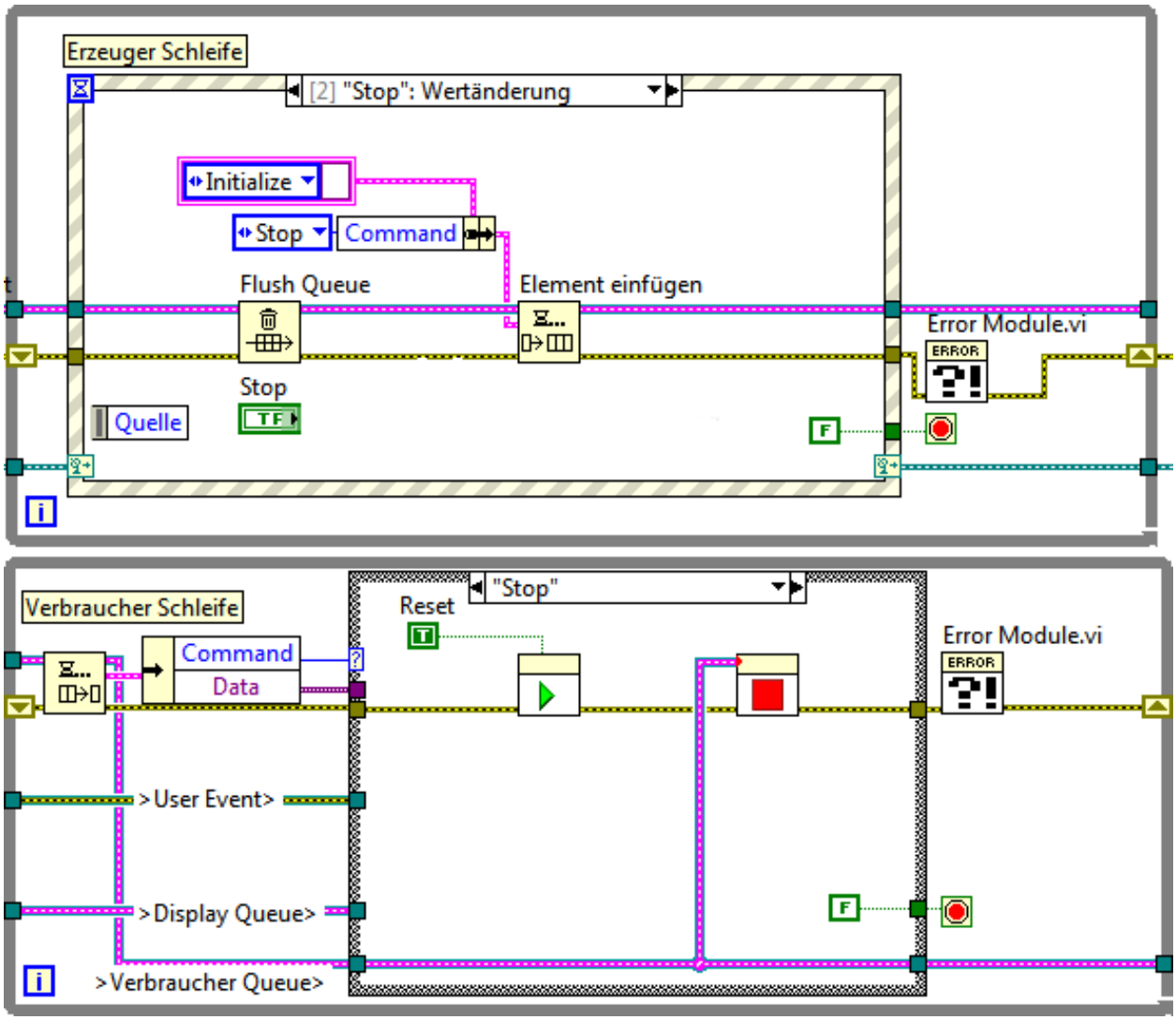

Abbildung 18: Stopp-Funktion

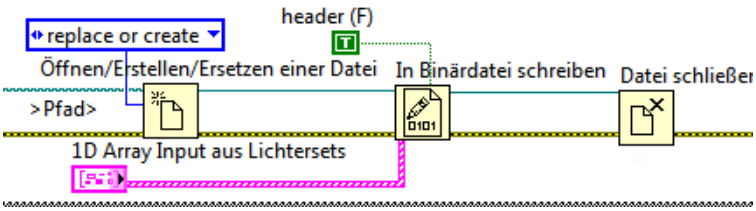

Abbildung 19: Speicher-Funktion

A.10 Stopp-Funktion

A.11 Speicher-Funktion

A.12 Lade-Funktion

A.13 Text

A.14 Text

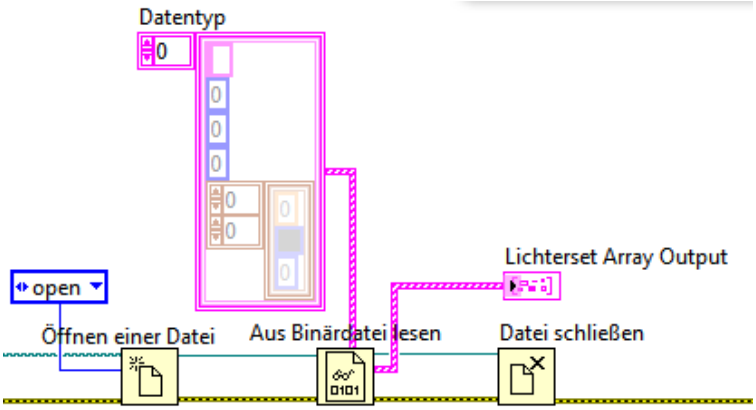

Abbildung 20: Lade-Funktion

Abbildung 21: Text

Abbildung 22: Text

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit mit dem Titel

Party Licht Steuerung – Programmentwurf für Lichttechniker mit LabView

selbständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe.

Mosbach, den 17. Juni 2011

Tim Berger