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

In der vorliegenden Arbeit wird der Programmentwurf einer Party-Lichtsteuerung, vom Design der Anwendung über die Implementierung bis hin zur Bereitstellung einer lauffähigen Applikation dokumentiert. Die Entwicklungsumgebung ist LabVIEW. Abschließend findet sich ein Gegenüberstellung der Vor- und Nachteile mit dem entwickelten graphischen Programmiersystem.

Abstract

This student research project deals with the development of a Party-Light-Control system. Starting with the program design to the fully developt application. The development environment is LabVIEW. This paper gives advantages and disadvantages in graphical programming.

Inhaltsverzeichnis

Abbildungsverzeichnis **IV**

Abkürzungsverzeichnis **V**

1 Einleitung **1**

1.1 Aufgabenstellung 1
1.2 Anforderungen 1
1.3 Dokumentation 1
1.4 Aufbau der Arbeit 2

2 LabVIEW als Programmiersprache **3**

2.1 Entwurfsmuster - Design Pattern 4
2.1.1 Zustandsautomat 4
2.1.2 Master/Slave-Entwurfsmuster 4
2.1.3 Einfacher Ereignisbehandler für Benutzeroberfläche 4
2.1.4 Erzeuger/Verbraucher-Entwurfsmuster 5

3 Programm Analyse **5**

3.1 Daten Abstraktion 5
3.1.1 Objekte 5
3.1.2 Module 5
3.2 Ablaufdiagramm 6
3.3 Datenfluss Diagramm 8

4 Code Implementierung **8**

4.1 Auswahl des Design Pattern 8
4.2 Init und Shutdown Funktion 9
4.3 User Interface 10
4.3.1 Initialisierung des Front Panels 10
4.3.2 Auswahl eines Lichtsets aus der Lichterset Queue 10
4.3.3 De-/Aktivieren von Schaltflächen 11
4.3.4 Update der Set-Ablaufliste 11
4.3.5 Update der Lichtkanäle 11
4.4 Aufnahme-Funktion 12

4.5 Timing 12
4.5.1 Funktional globale Variable 12
4.6 Abspiel-Funktion 14
4.7 Stopp-Funktion 15
4.8 Speichern und Lade Funktion 15
4.8.1 In Datei speichern 15
4.8.2 Aus Datei laden 15
4.9 Fehlerbehandlung 16

5 Stress und Lade Test **16**

6 Einsetzen der Anwendung **17**
6.1 Webservice 18

7 Abschließende Betrachtung **18**
7.1 Fazit 18
7.2 Ausblick auf Erweiterungen 19
7.2.1 Ansteuerung von Hardware 19
7.2.2 Multilingualität 19

Literaturverzeichnis **21**

Anhang **22**
A.1 Initialisierungsfunktion 23
A.2 Shutdown-Funktion 24
A.3 Initialisierung des Front Panels 24
A.4 Auswahl eines Lichtsets aus der Lichterset Queue 24
A.5 De-/Aktivieren von Schaltflächen 25
A.6 Update der Set-Ablaufliste 25
A.7 Update der Lichtkanäle 26
A.8 Timing Modul 26
A.9 Abspiel Zustandsautomat - Überblenden 27
A.10 Stopp-Funktion 28
A.11 Speicher-Funktion 28
A.12 Lade-Funktion 29
A.13 Fehlerbehandlung 29

Erklärung **30**

Abbildungsverzeichnis

1 Bedienoberfläche für die Party-Lichtsteuerung 2
2 VI Demonstration: Links Frontpanel, Rechts Blockdiagramm 3
3 Ablaufdiagramm für die Abspiel-Funktion 7
4 Datenfluss Diagramm für die Abspiel-Funktion 8
5 Design Struktur für das Blockdiagramm 10
6 Display Schleife 11
7 Aufnahme-Funktion 13
8 FGV Timing VI 14
9 Verlauf der Prozessor- und Arbeitsspeicher Auslastung 17
10 NI PCI-6723 mit 32 Analogausgänge [NI11c] 19
11 Initialisierungsfunktion 23
12 Shutdown-Funktion 24
13 Initialisierung des Front Panels 24
14 Auswahl eines Lichtsets aus der Lichterset Queue 24
15 De-/Aktivieren von Schaltflächen 25
16 Update der Set-Ablaufliste 25
17 Update der Lichtkanäle 26
18 Timing Modul 26
19 Abspiel Zustandsautomat - Überblenden 27
20 Stopp-Funktion 28
21 Speicher-Funktion 28
22 Lade-Funktion 29
23 Fehlerbehandlung 29

Abkürzungsverzeichnis

FGV funktionale globale Variable

LabVIEW Laboratory Virtual Instrumentation Engineering Workbench

LJ Light Jockey (dt. Lichttechniker)

NI National Instruments

subVI unter Programm eines VIs

VI virtuelles Instrument

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 Dokumentation

Der LabVIEW Quellcode sowie eine Ausführbare Windows-Anwendung mit der notwendigen RunTime als Insatller findet sich auf der beigelegten CD. Auch enthalten sind die Programmcode ausschnitte als Bilder in hoher Detailansicht.

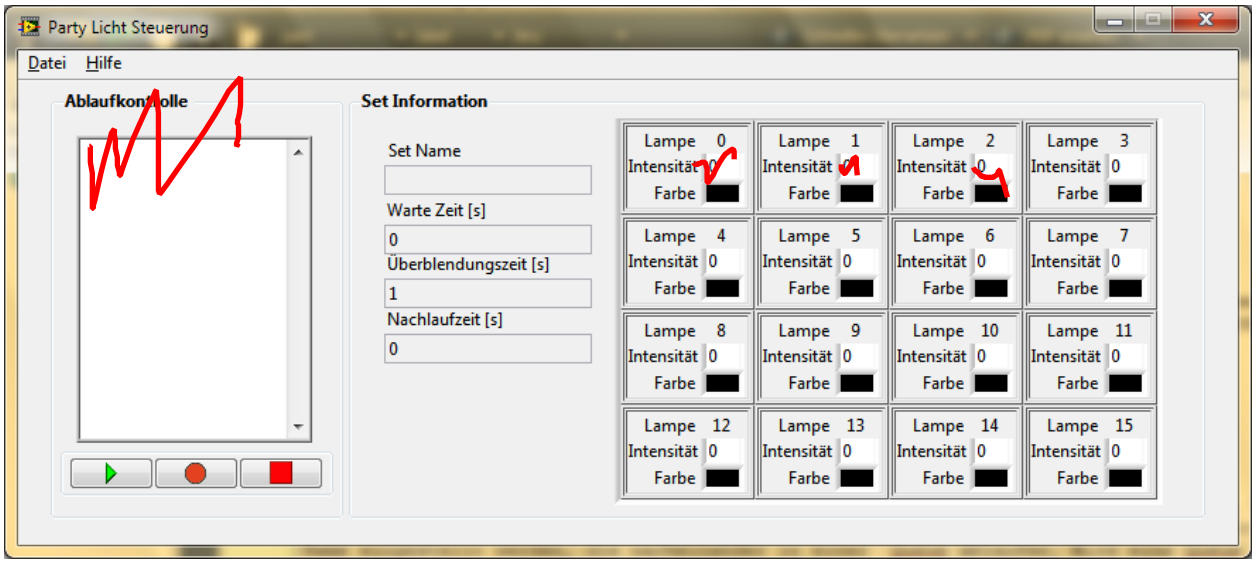

Abbildung 1: Bedienoberfläche für die Party-Lichtsteuerung

1.4 Aufbau der Arbeit

Zu Beginn wird auf LabVIEW als Programmiersprache eingegangen und verschiedene Entwurfsmuster für eine strukturierte Programmierung vorgestellt. Darauf folgt die Programmanalyse mit der Datenabstraktion, einem Ablauf- und Datenflussdiagramm. Im 4. Kapitel wird die Code Implementierung Dokumentiert. Im darauf folgenden Kapitel wird ein Testfall beschrieben und ausgewertet. Ein Fazit und Ausblick auf Erweiterungen bilden den Abschluss. Im Anhang finden sich vergrößerte Code Ausschnitte, sie wurden zur besseren Lesbarkeit aus dem Text ausgegliedert.

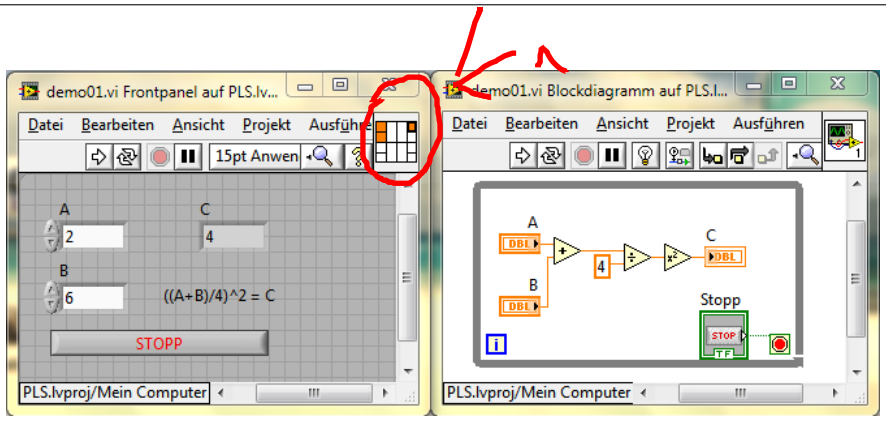

Abbildung 2: VI Demonstration: Links Frontpanel, Rechts Blockdiagramm

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. [Jam97] 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. [NI11e]
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?
- 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 Daten Abstraktion

Um in LabVIEW auf Objekten zu arbeiten gibt es Module. Das sind VIs die als Methoden agieren, sie haben Eingänge und Rückgabewerte.

3.1.1 Objekte

Folgende Objekte werden in der Party-Licht-Steuerung unterschieden:

Lichtkanal Eine Lampe mit einer Farbe, Intensität und Nummer.

Lichterset Ein 2D-Array aus Lampen mit einer Warte-, Überblendungs- und Nachlaufzeit sowie einem Namen.

Lichterset Queue Eine Liste mit allen Lichtersets.

Verbraucher Queue Die Warteschlange für die Verbraucherschleife, hier stehen Befehle von der Erzeugerschleife drin.

Display Queue Die Warteschlange für die Displayschleife, über diese kommunziert die Verbraucherschleife mit ihr. Enthält das selbe Cluster wie die Verbraucher Queue

3.1.2 Module

Dieser Abschnitt beschreibt die Module und nennt ihre Funktionen:

Lichterset Modul Das Lichterset Modul initialisiert das Lichterset Array und führt folgende Anweisungen darauf aus:

- Add Get und Set Lichterset
- getAnzahlLichtersets
- getLeeresLichterset

Display Modul Dieses Modul sorgt für die Aktualisierungen auf dem Front Panel.

- Init und Update Front Panel
- Auswahl Lichterset aus der Lichterset-Queue
- De-/Aktivieren von Schaltflächen
- Update Lichterset-Queue

Timing Modul Es sorgt für die Berechnung der verschiedenen Zeiten:

- Wartezeit
- Überblendungszeit
- Nachlaufzeit

Fehler Modul Hier wird die Fehlerbehandlung durchgeführt.

- Fehler ausgeben
- Fehler behandeln

Datei Modul Das Modul sorgt für den Datei Ein- und Ausgabe.

- Speicher Lichtersets
- Lade Lichtersets

Auf die Implementierung der Module wird im Kapitel 4 eingegangen.

3.2 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.

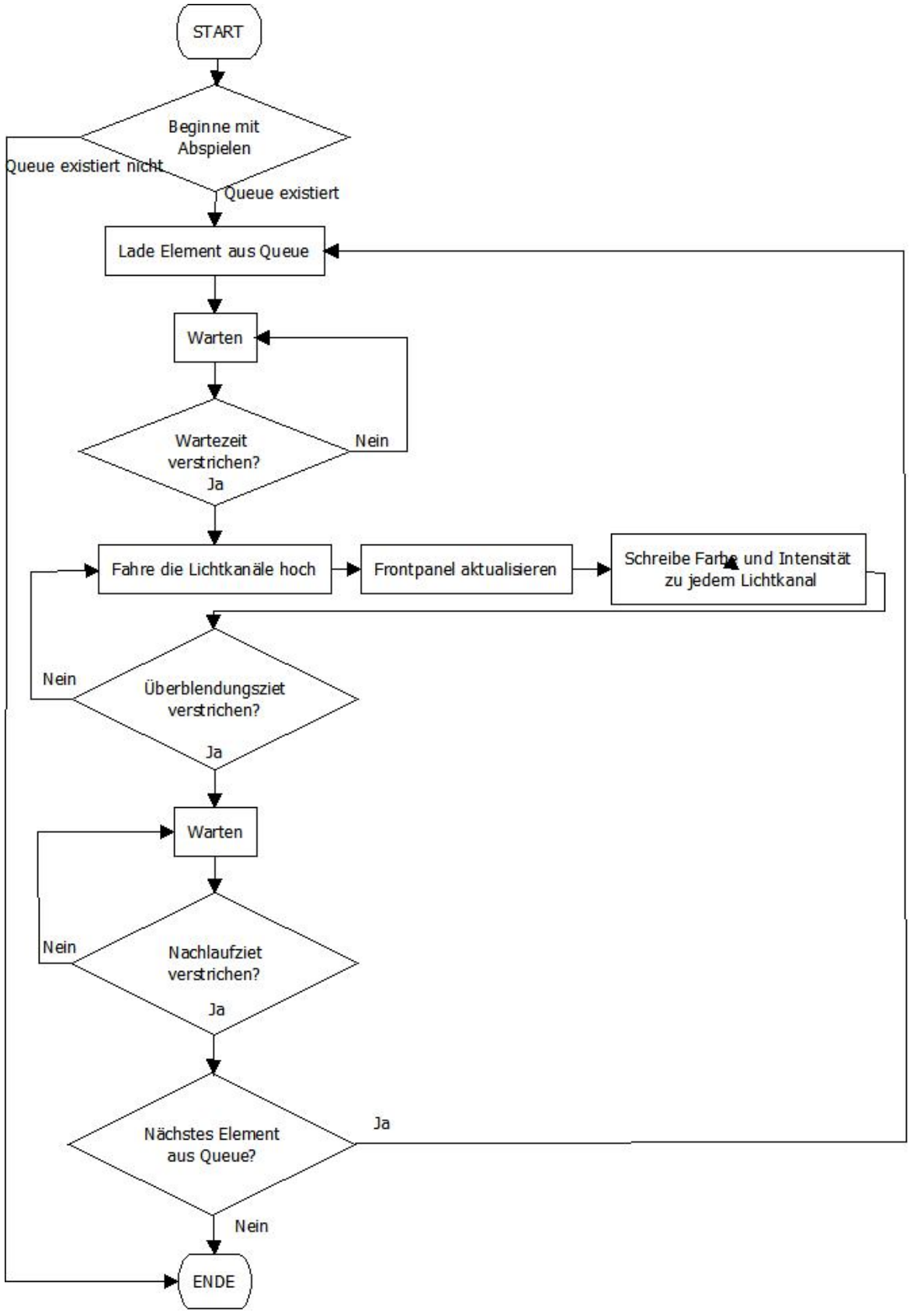

Abbildung 3: Ablaufdiagramm für die Abspiel-Funktion

3.3 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.

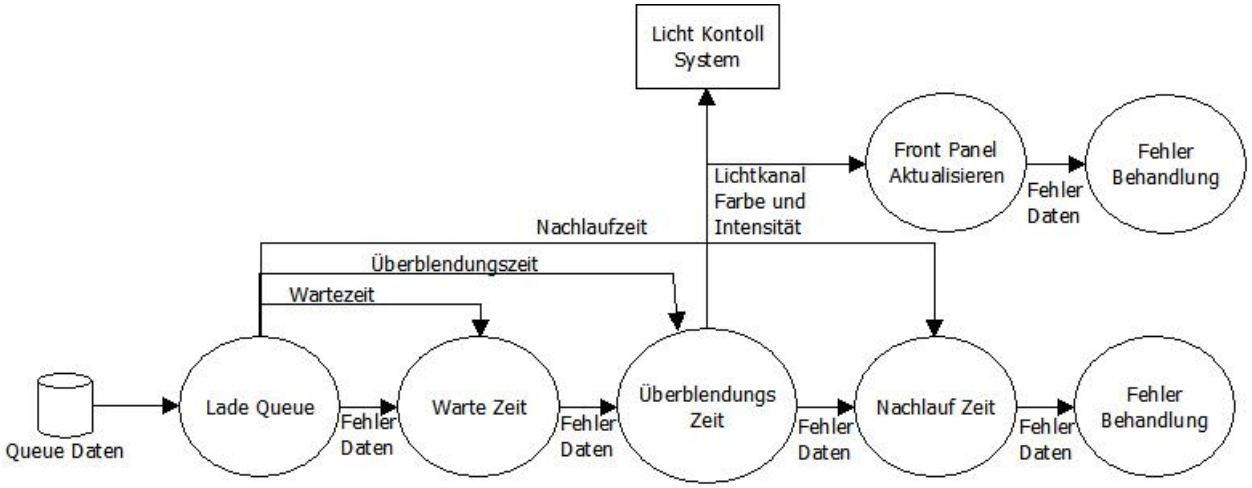

Abbildung 4: Datenfluss Diagramm für die Abspiel-Funktion

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

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 11 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 12 im Anhang zeigt das Blockdiagramm vom *„shutdown.vi“*.

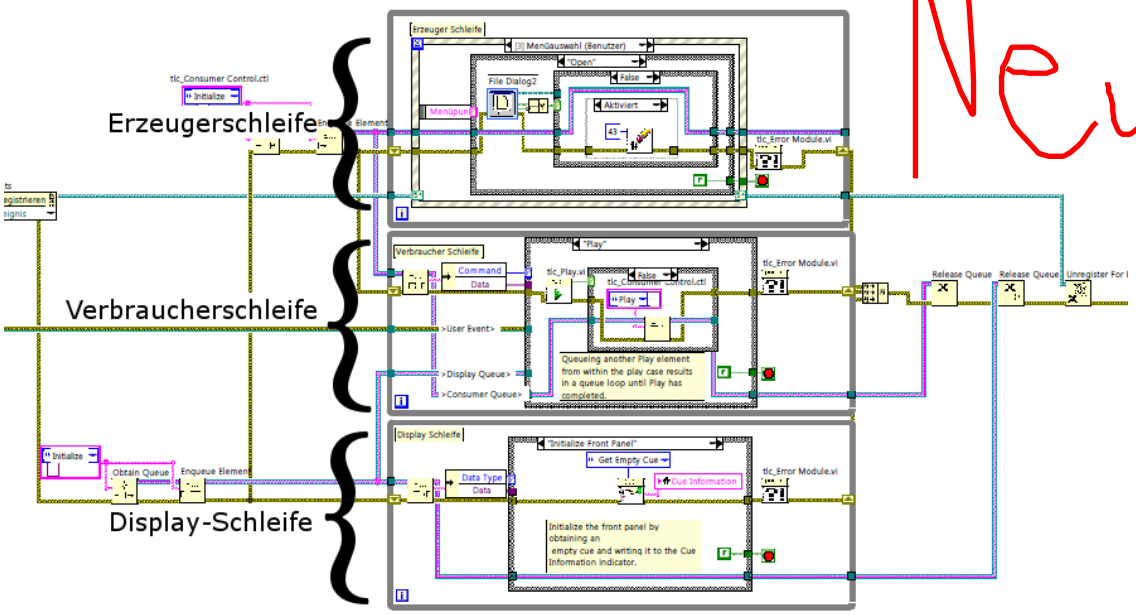

Abbildung 5: Design Struktur für das Blockdiagramm

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 4.9 Fehlerbehandlung. Im folgenden werden die einzelnen Cases in der Display-Schleife erläutert.

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 13.

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 Lichter-

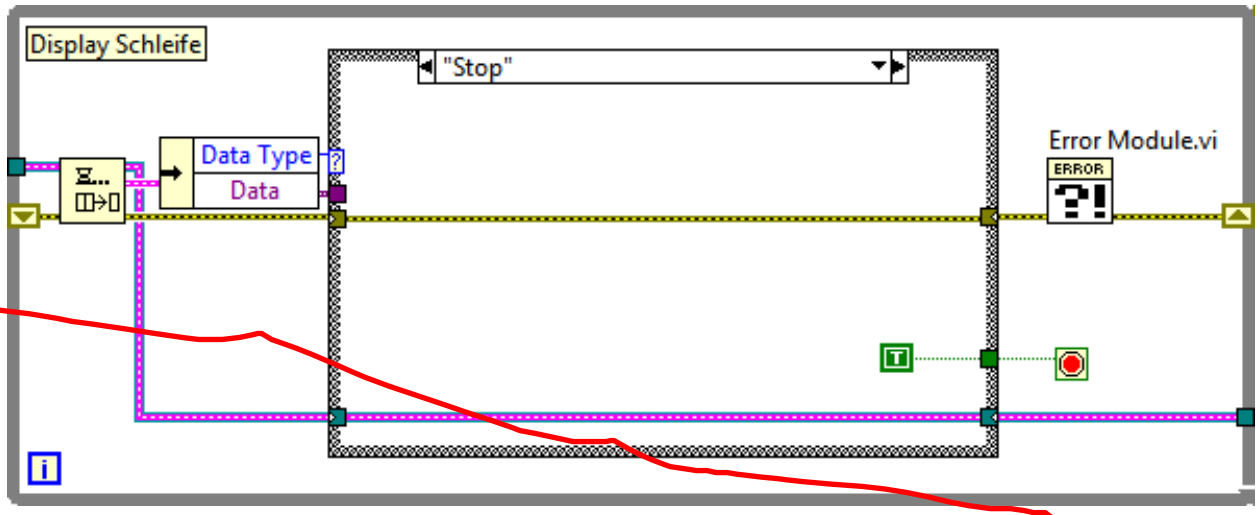

Abbildung 6: Display Schleife

sets iteriert.
Der entsprechende Ausschnitt aus dem Programmcode ist im Anhang auf Abbildung 14.

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 15.

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 16.

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 17.

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 funktionalen globalen Variablen (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

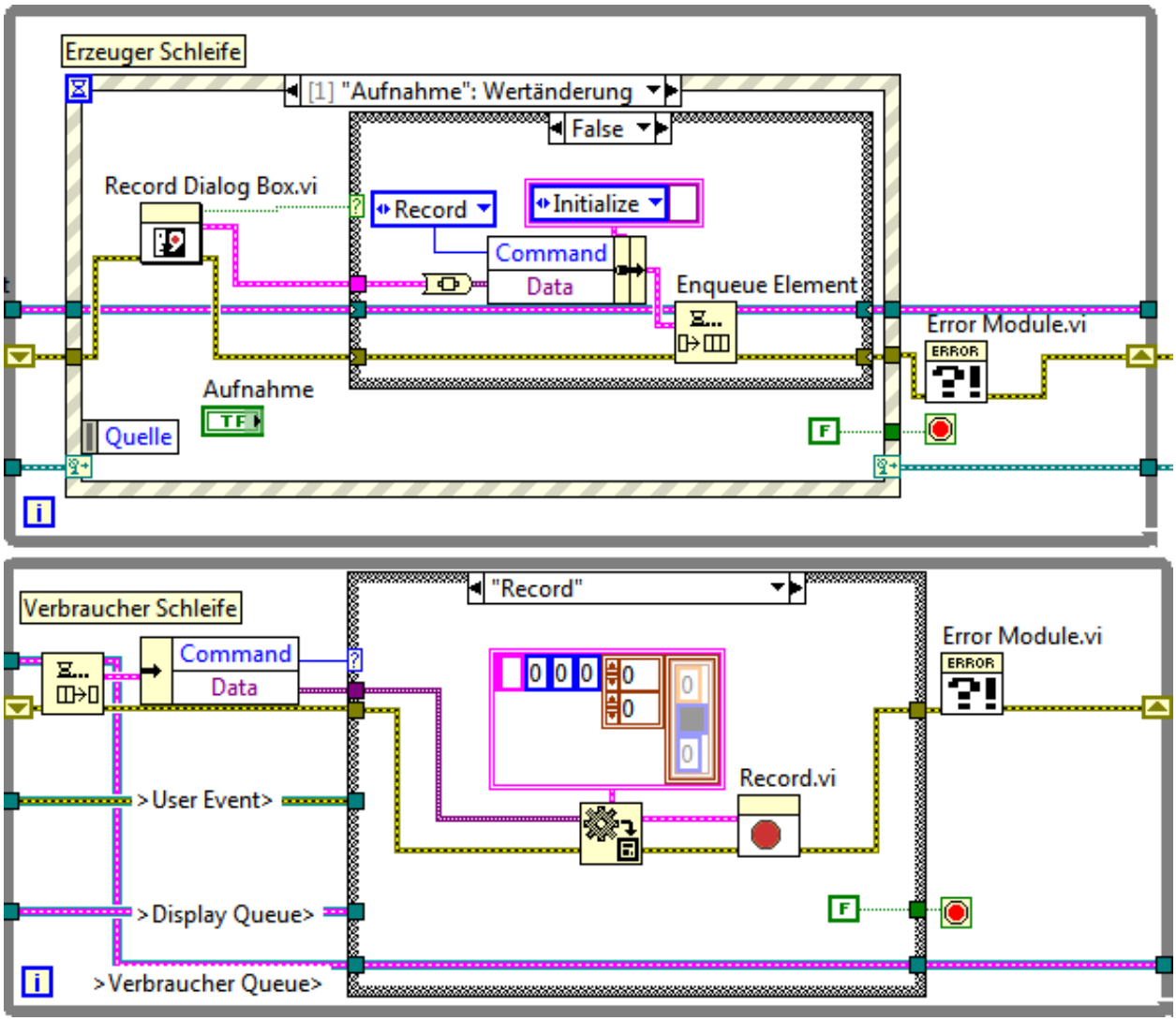

Abbildung 7: Aufnahme-Funktion

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. [NI10]

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, 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.

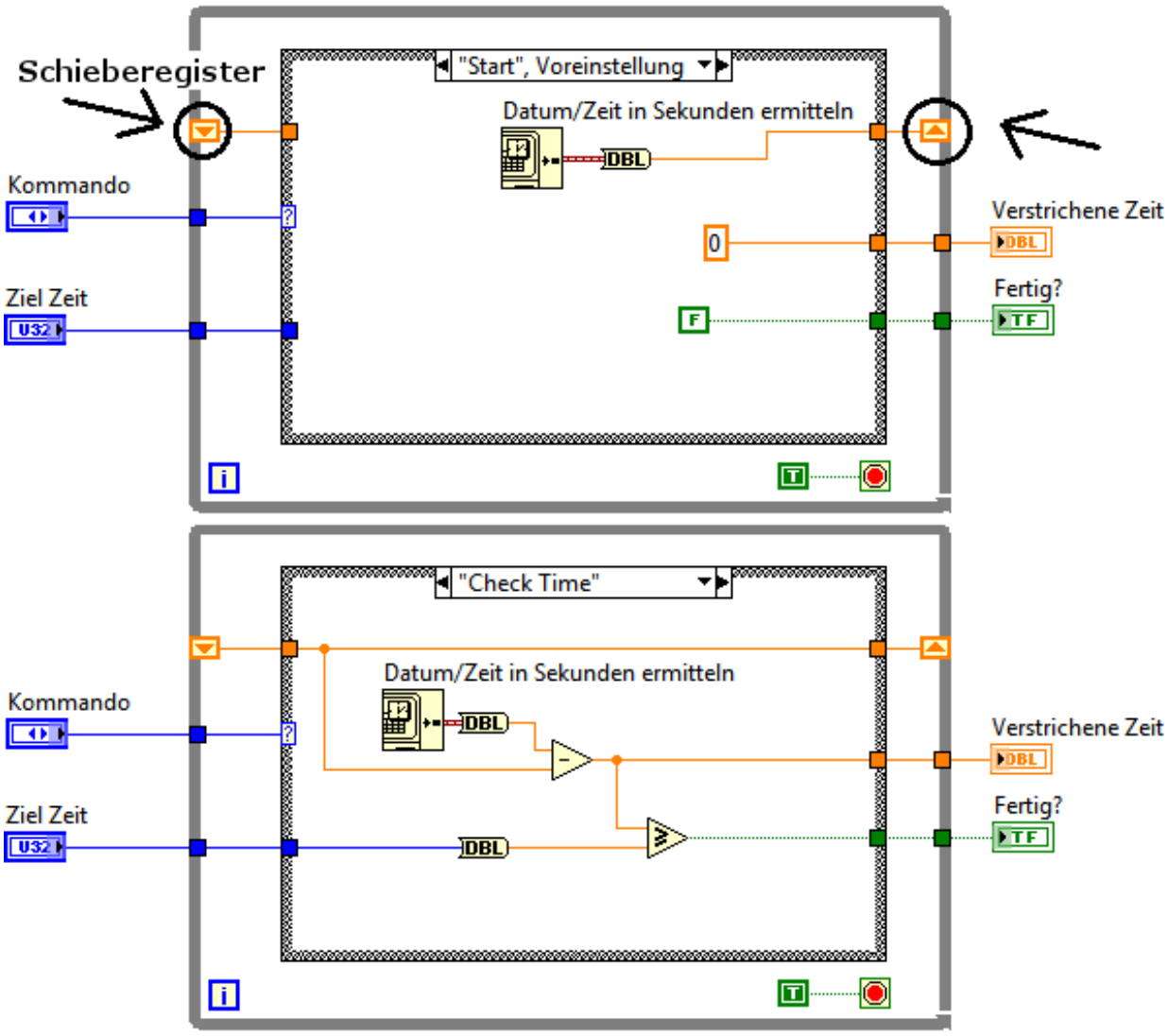

Abbildung 8: FGV Timing VI

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 *„record.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 19 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 20 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 21.

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 22.

4.9 Fehlerbehandlung

Zur Fehlerbehandlung gehört das sichere herunterfahren der Applikation wenn ein Fehler auftritt. Um die Erzeugerschleife zu beenden muss ein User-Event erzeugt werden. Die Verbraucherschleife wird beendet in dem das Kommando Exit in die Verbraucher Queue geschrieben wird. Zum beenden der Display-Schleife wird das Kommando Stopp an die Display Queue gesendet. Empfängt eine Schleife solch ein Kommando bzw. Event wird die Abbruchbedingung auf true gesetzt. Sind alle drei Schleifen beendet, wird in der letzten Sequenz der Applikation die Verbraucher und Display Queue freigegeben, die Ereignisregistrierung aufgehoben, die Benutzerereignisse gelöscht, der aufgetretene Fehler ausgegeben und die LabVIEW Anwendung beendet.
Der Code, der im Fall eines oder mehrere Fehler ausgeführt wird ist im Anhang unter Abbildung 23 zu finden.

5 Stress und Lade Test

Bevor die Anwendung freigegeben wird ist es notwendig die Leistung und Handhabbarkeit der Applikation zu testen.
Für den Test der Party-Licht-Steuerung wird ein VI geschrieben das 10.000 Lichtersets mit zufälligen Parametern für die Warte-, Überblendungs- und Nachlaufzeit sowie Lichtfarbe und -intensität in eine Datei schreibt.
Als Testwerkzeug dient der Windows Task-Manager. Er läuft mit während die Testdatei in die Anwendung geladen und abgespielt wird. Mit Hilfe des Windows Task-Manager kann überprüft werden, ob während der Laufzeit ein Leistungs- oder Speicherfehler auftritt.
Die Anwendung wurde auf einem Notebook mit einem Intel Core2 Duo Prozessor (2,17GHz) und 2 GB Hauptspeicher ausgeführt.
Abbildung 9 zeigt den Verlauf der Prozessor- und Arbeitsspeicher Auslastung während des Tests. Das erste rot umrahmte Viereck (von links nach rechts) zeigt den Ladevorgang der Applikation. Es werden circa 40 MB in den Arbeitsspeicher geladen. Das zweite Viereck umrahmt das laden der Testdatei (2,7 MB) hierfür werden in Arbeitsspeicher ungefähr 102 MB reserviert. Damit beansprucht die gesamte Applikation 141 MB des Hauptspeichers. Das letzte Viereck zeigt den Abspielvorgang bis zum 140. Lichterset. Nach ungefähr 4 Stunden sind alle 10.000 Sets durchgelaufen danach konnte die Anwendung ohne Fehler beendet werden und der zuvor reservierte Speicher wieder freigegeben. Während des Abspielvorgangs war der Prozessor im Mittel mit 14% ausgelastet.

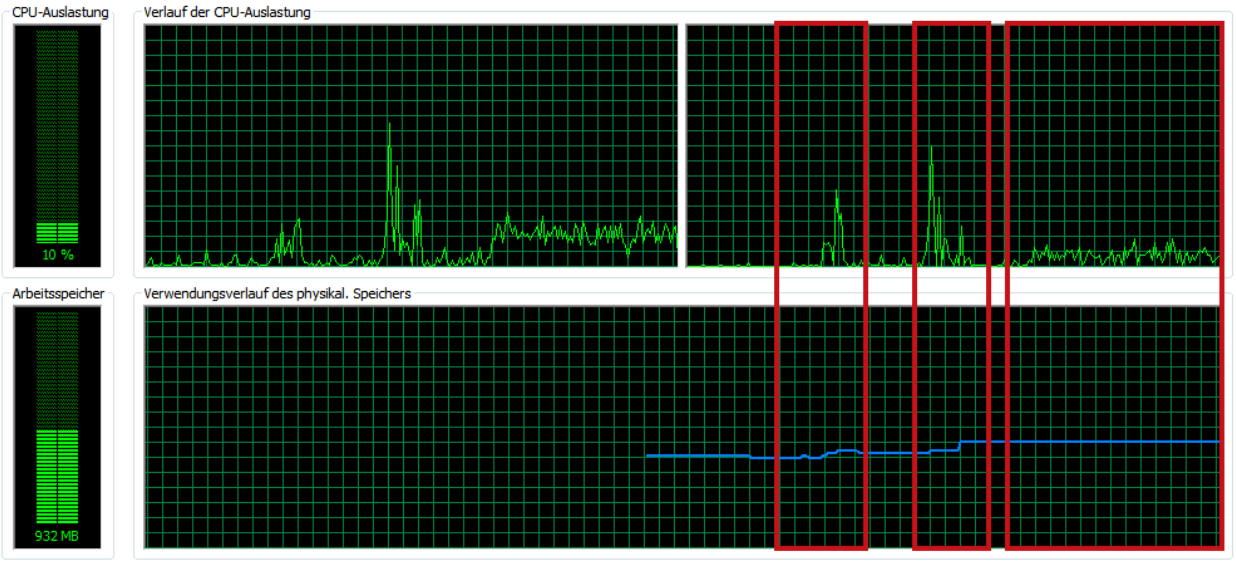

Abbildung 9: Verlauf der Prozessor- und Arbeitsspeicher Auslastung

Aus dem Test lässt sich das Ergebnis ziehen, das kein Speicherüberlauf auftrat, kein Speicher unnötig allokiert wurde und der Prozessor mit der Anwendung nicht überlastet war. Somit gilt der Test als bestanden.

6 Einsetzen der Anwendung

LabVIEW bietet die Möglichkeit über ein Dialog aus einem Projekt heraus eine ausführbare Windows- (.exe) oder MAC OS (.app) Anwendung oder Installer zu erstellen. Eine ausführbare Windowsanwendung und ein Installer mit der notwendigen Laufzeit-Umgebung finden sich auf der beigefügten CD. Für die Laufzeit-Umgebung gibt es eine kostenfreie Lizenz, sie kann beliebig oft vervielfältigt werden.
Die Mindestvoraussetzungen für ein Windowssystem sind [NI11d]:

- Prozessor: Pentium III/Celeron 866 MHz oder gleichwertig
- Hauptspeicher: 256 MB
- Festplattenspeicher: 1 GB
- Betriebssystem: XP

6.1 Webservice

Mit LabVIEW lässt sich ein Webinterface erstellen auf das der Anwender via Inter- oder Intranet mit einem Standartbrowser zugreifen kann. Rechenintensive Anwendungen können von einer Serverfarm bearbeitet werden und müssen nicht die Rechnerressourcen des Hosts beanspruchen. Auch lassen sich mit einem Webservice, Prozesse von der Ferne aus überwachen. Anwendungen hierfür wären zum Beispiel die Produktionsüberwachung für die Produktionsleitung. Mitarbeiter müssten nicht mehr von Maschinenterminal zu Maschinenterminal laufen sondern könnten zum Beispiel von einem Heimarbeitsplatz die ganze Produktionsanlage überwachen.

7 Abschließende Betrachtung

Bei dem Projekt ist eine voll funktionsfähige LabVIEW Anwendung entstanden, somit konnte das Projekt erfolgreich abgeschlossen werden.

7.1 Fazit

Ein wichtiger Vorteil in der graphischen Programmierung von LabVIEW ist die Einfachheit parallele Abläufe zu programmieren. So reicht es aus zwei Sub-VIs ohne Datenabhängigkeit nebeneinander zulegen. Man muss jedoch auf mögliche „Race Conditions“ achten, hierfür stehen verschiedene Möglichkeiten zur Verfügung. Zum Beispiel Semaphore oder Warteschlangen.
Das Front Panel von LabVIEW hat gezeigt, das es ein bequeme Möglichkeit bietet, schnell und gute Bedienoberflächen zu erstellen.
Im Blockdiagramm ist durch die graphische Darstellung des Programmablaufs die Lesbarkeit deutlich einfacher. Anhand verschiedener Farben lassen sich die Datentypen und ihr Ursprung besser erkennen.
Ein weiteren Vorteil den LabVIEW bietet ist die Unterstützung von Kommunikationsprotokollen wie TCP/IP. Es ist somit möglich auch weit entfernte Anwendungen zu steuern und nutzen.
Bei der Entwicklung von LabVIEW Programmen gibt es auch Nachteile, so ist man an die originale LabVIEW-Entwicklungsumgebung von National Instruments gebunden. Für diese fallen Lizenzen an. Die „NI Developer Suite Core“ für Windows kostet 5.800 Euro,

hier gibt es auch kostenlose Versionen mit verminderten Funktionsumfang für Studenten. [NI11b]
Auf einen weiteren Nachteil stößt man, wenn man LabVIEW-Programme verteilen will. Hat das Zielsystem keine Entwicklungsumgebung, ist es nötig eine Laufzeitumgebung zu installieren. Diese ist für die meisten Module kostenfrei.
Auch bei strukturierter Programmierung können kleine Änderungen im Programmfluss aufwendige Neustrukturierungen nach sich ziehen. Ärgerlich ist es beim Schaffen vom Raum im Blockdiagramm immer wieder Drähte und Symbole neu ordnen zu müssen.

7.2 Ausblick auf Erweiterungen

Die Party-Licht-Steuerung bietet noch Möglichkeiten für Erweiterungen.

7.2.1 Ansteuerung von Hardware

Möchte man externe Lichtkanäle ansteuern könnte man ein dritte Verbraucher Schleife hinzufügen. Diese muss dann einen „Data Acquisition-Task“ starten und die Ausgänge setzen. Als anzusteuernde Hardware empfiehlt sich das NI PCI-6723 Analogausgangsmodul. Es hat 32 Analogausgänge, damit könnte man von 16 Lampen die Farbe und Intensität steuern.[NI11c]

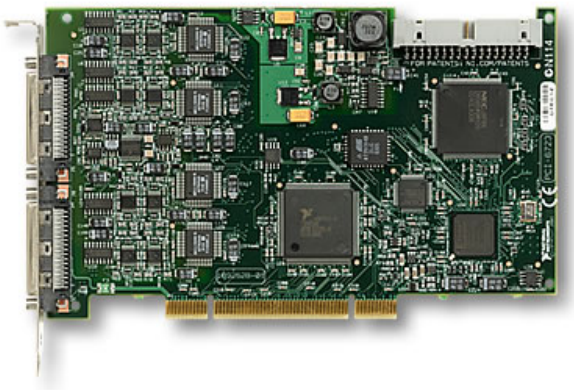

Abbildung 10: NI PCI-6723 mit 32 Analogausgänge [NI11c]

7.2.2 Multilingualität

Um die Applikation einer größeren Masse an Anwendern zur Verfügung zu stellen konnte eine Mehrsprachigkeit implementiert werden. Dazu könnte beim Start der Anwendung

Übersetzungen

der Nutzer nach seiner Sprache gefragt werden. Über eine Tabelle mit der zur Verfügung stehenden Sprachen könnte ein Auswahl für die Beschriftung der Front Panel Elemente getroffen werden.

Literatur

[Jam97] JAMAL, RAHMAL: *LabVIEW – Programmiersprache der vierten Generation*. Prentice Hall, 1997.

[NI10] NI: *LabVIEW-Hilfe*, Juni 2010. Integriert in die Entwicklungsumgebung.

[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: *NI Developer Suite Advisor*. `http://ohm.ni.com/advisors/devsuite/pages/cds/newcustomer.xhtml`, 2011. Zugriff am 5.6.2011 um 17:11 in Datei *„NI-Developer-Suite.pdf“*.

[NI11c] NI: *NI PCI-6723 Analogausgangsmodul für statische Signale und Signalverläufe: 13 bit, 32 Kanäle*. `http://sine.ni.com/nips/cds/view/p/lang/de/nid/12551`, 2011. Zugriff am 5.6.2011 um 17:11 in Datei *„NI-PCI-6723.pdf“*.

[NI11d] NI: *Systemvoraussetzungen für LabVIEW-Entwicklungssysteme und -Module*. `http://www.ni.com/labview/requirements/d/`, 2011. Zugriff am 5.6.2011 um 17:11 in Datei *„Systemvoraussetzungen-LabVIEW.pdf“*.

[NI11e] 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 **23**

A.2 Shutdown-Funktion **24**

A.3 Initialisierung des Front Panels **24**

A.4 Auswahl eines Lichtsets aus der Lichterset Queue **24**

A.5 De-/Aktivieren von Schaltflächen **25**

A.6 Update der Set-Ablaufliste **25**

A.7 Update der Lichtkanäle **26**

A.8 Timing Modul **26**

A.9 Abspiel Zustandsautomat - Überblenden **27**

A.10 Stopp-Funktion **28**

A.11 Speicher-Funktion **28**

A.12 Lade-Funktion **29**

A.13 Fehlerbehandlung **29**

A.1 Initialisierungsfunktion

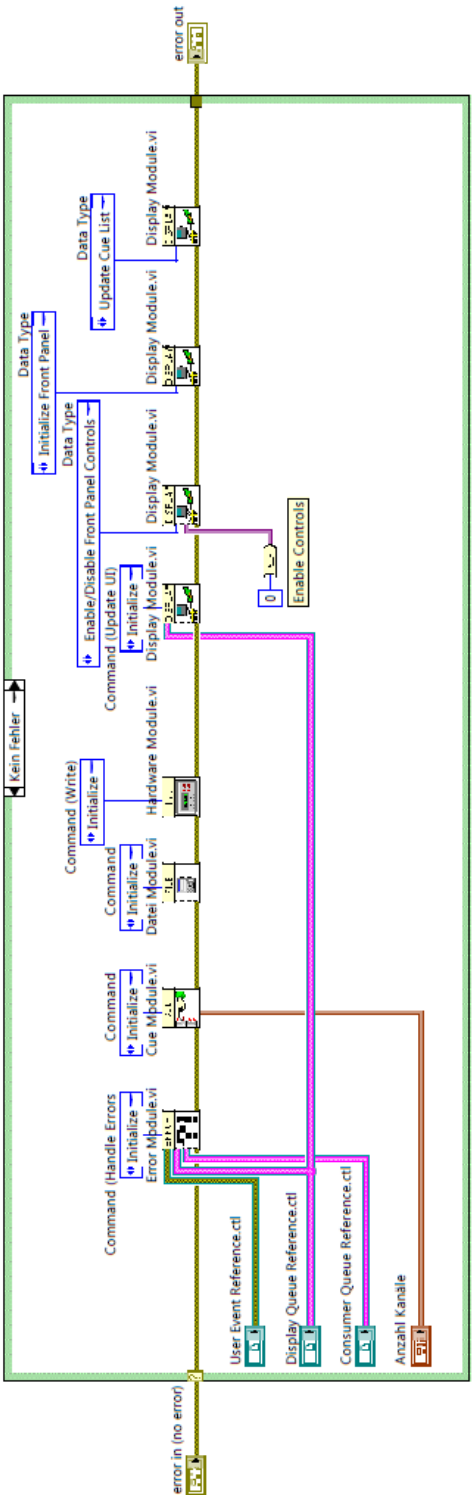

Abbildung 11: Initialisierungsfunktion

A.2 Shutdown-Funktion

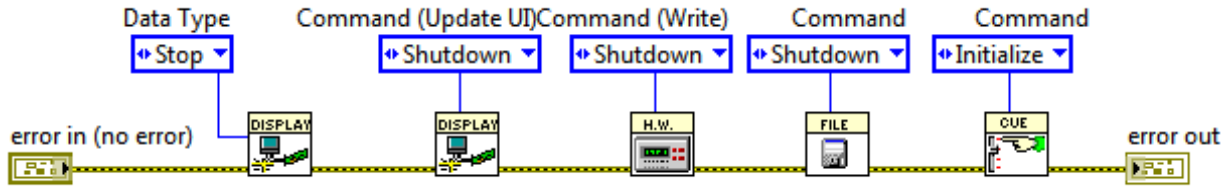

Abbildung 12: Shutdown-Funktion

A.3 Initialisierung des Front Panels

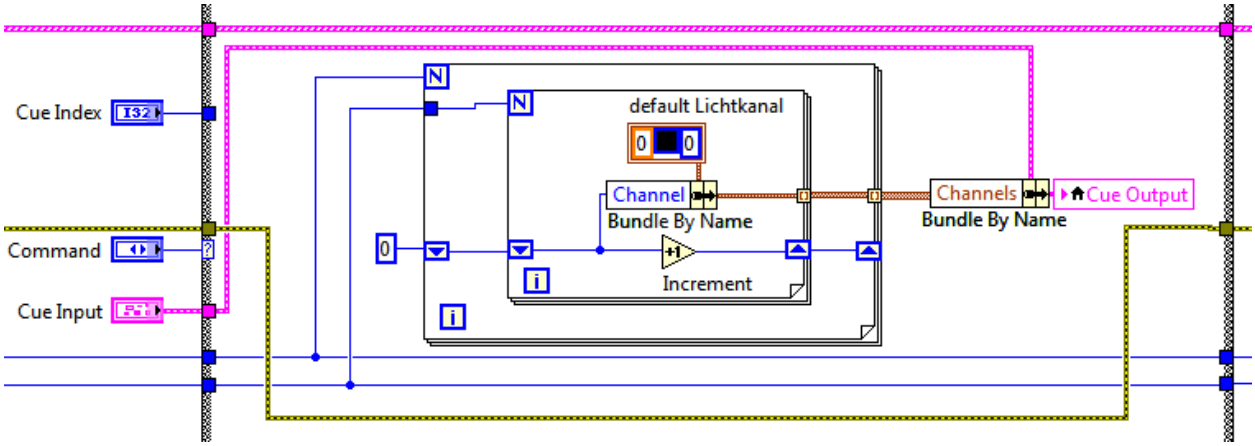

Abbildung 13: Initialisierung des Front Panels

A.4 Auswahl eines Lichtsets aus der Lichterset Queue

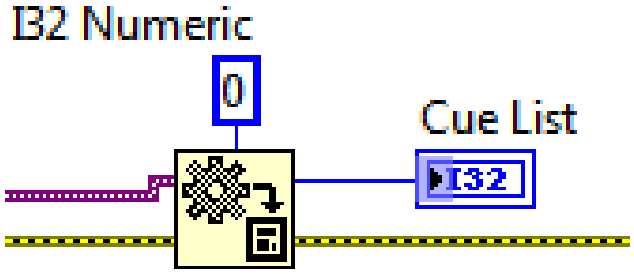

Abbildung 14: Auswahl eines Lichtsets aus der Lichterset Queue

A.5 De-/Aktivieren von Schaltflächen

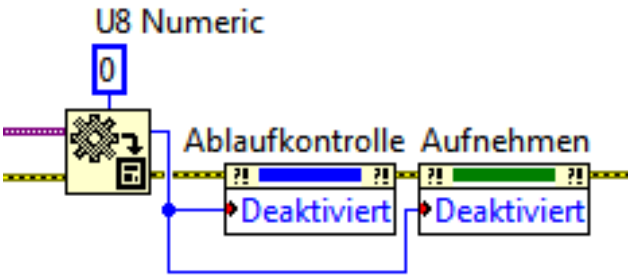

Abbildung 15: De-/Aktivieren von Schaltflächen

A.6 Update der Set-Ablaufliste

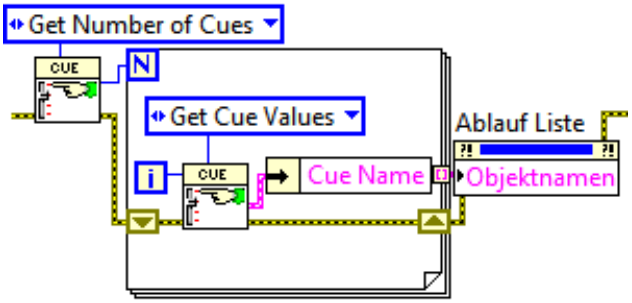

Abbildung 16: Update der Set-Ablaufliste

A.7 Update der Lichtkanäle

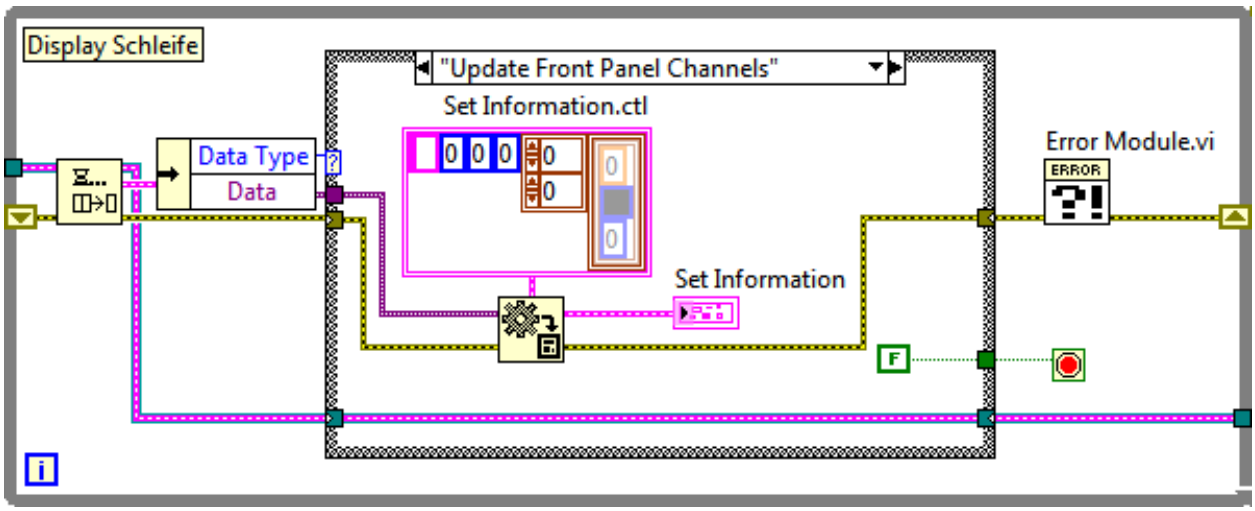

Abbildung 17: Update der Lichtkanäle

A.8 Timing Modul

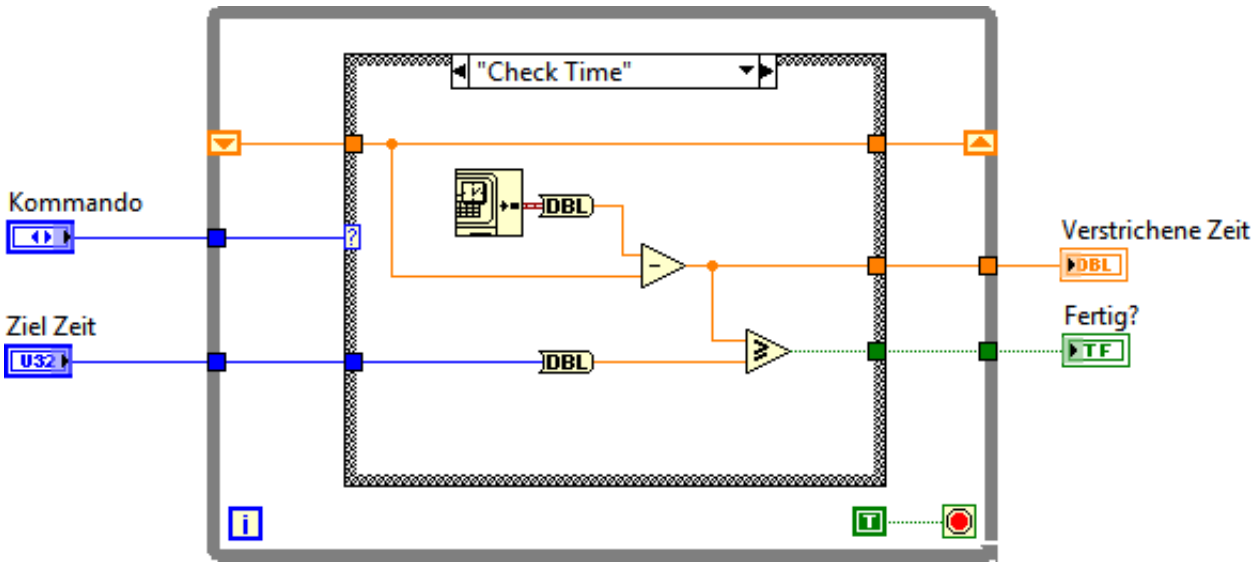

Abbildung 18: Timing Modul

A.9 Abspiel Zustandsautomat - Überblenden

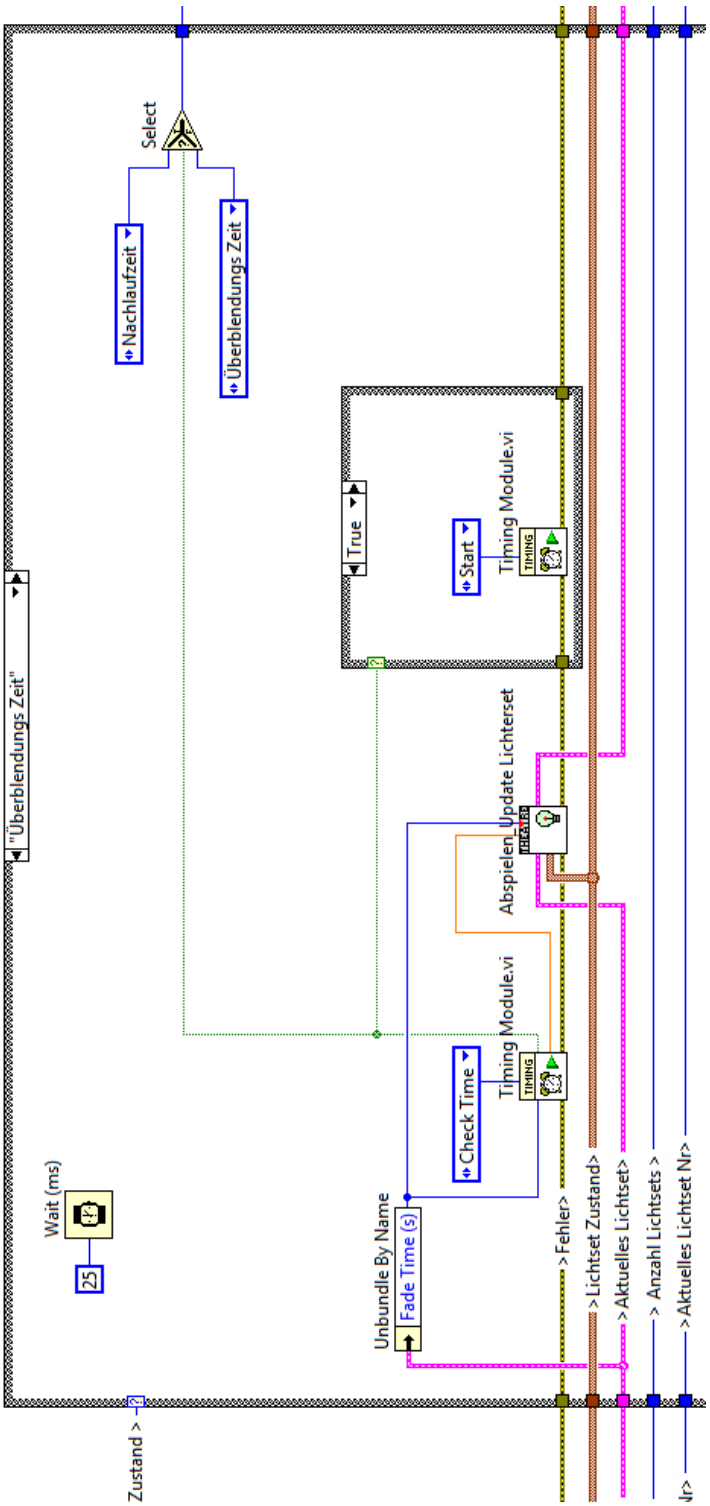

Abbildung 19: Abspiel Zustandsautomat - Überblenden

A.10 Stopp-Funktion

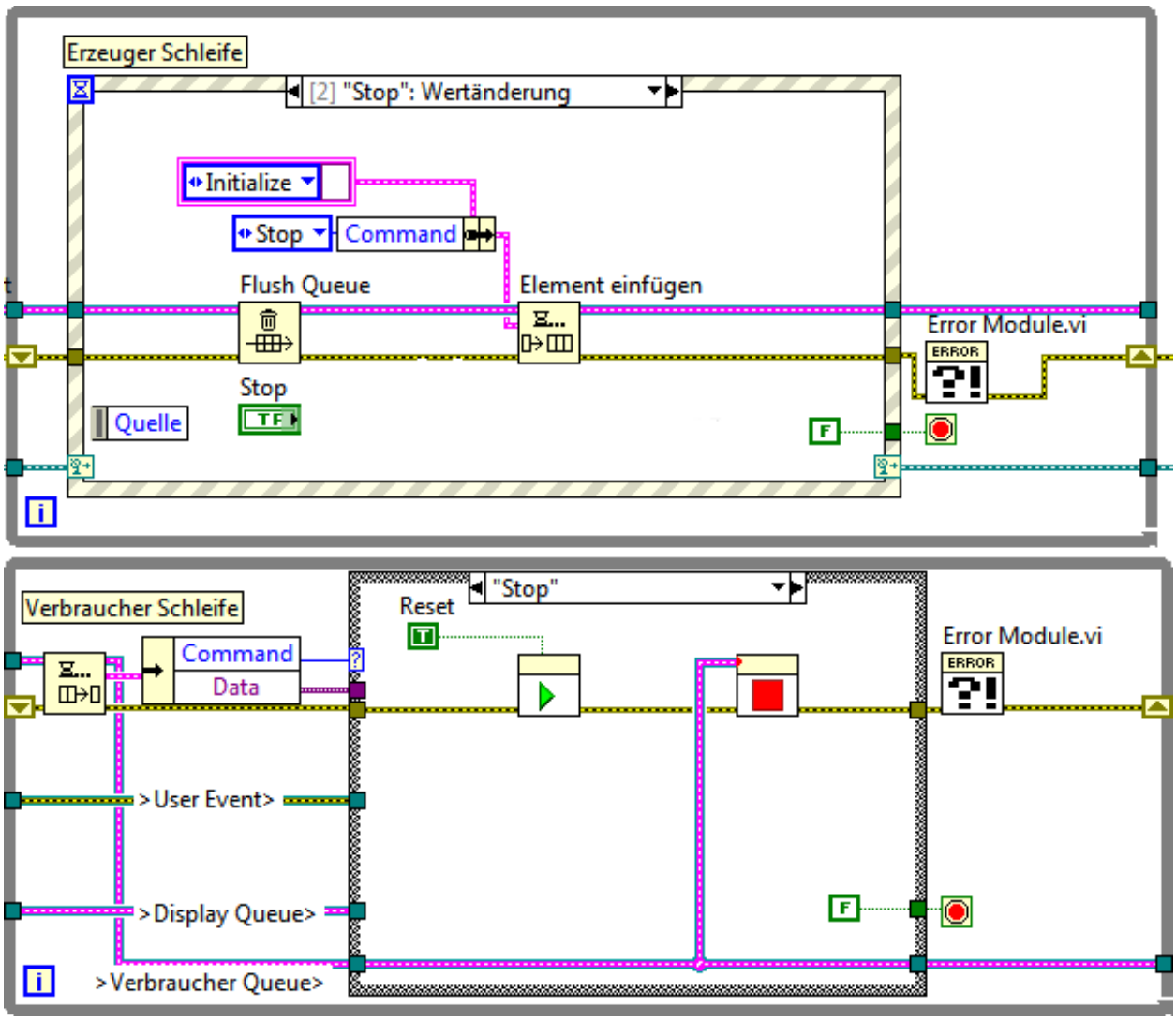

Abbildung 20: Stopp-Funktion

A.11 Speicher-Funktion

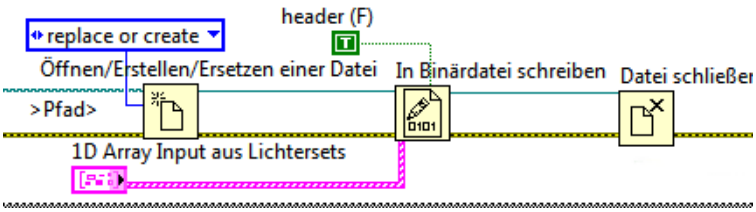

Abbildung 21: Speicher-Funktion

A.12 Lade-Funktion

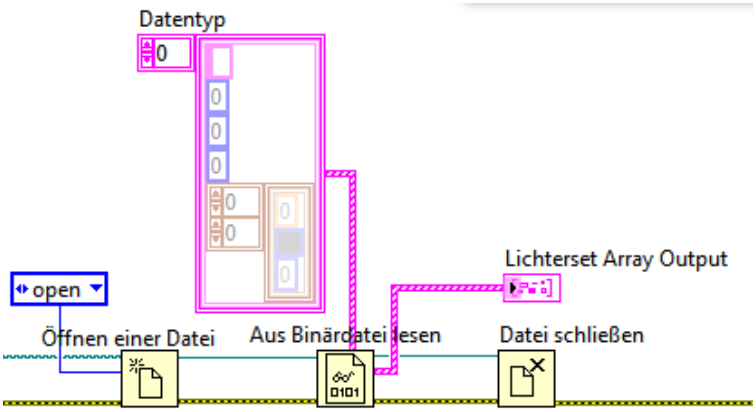

Abbildung 22: Lade-Funktion

A.13 Fehlerbehandlung

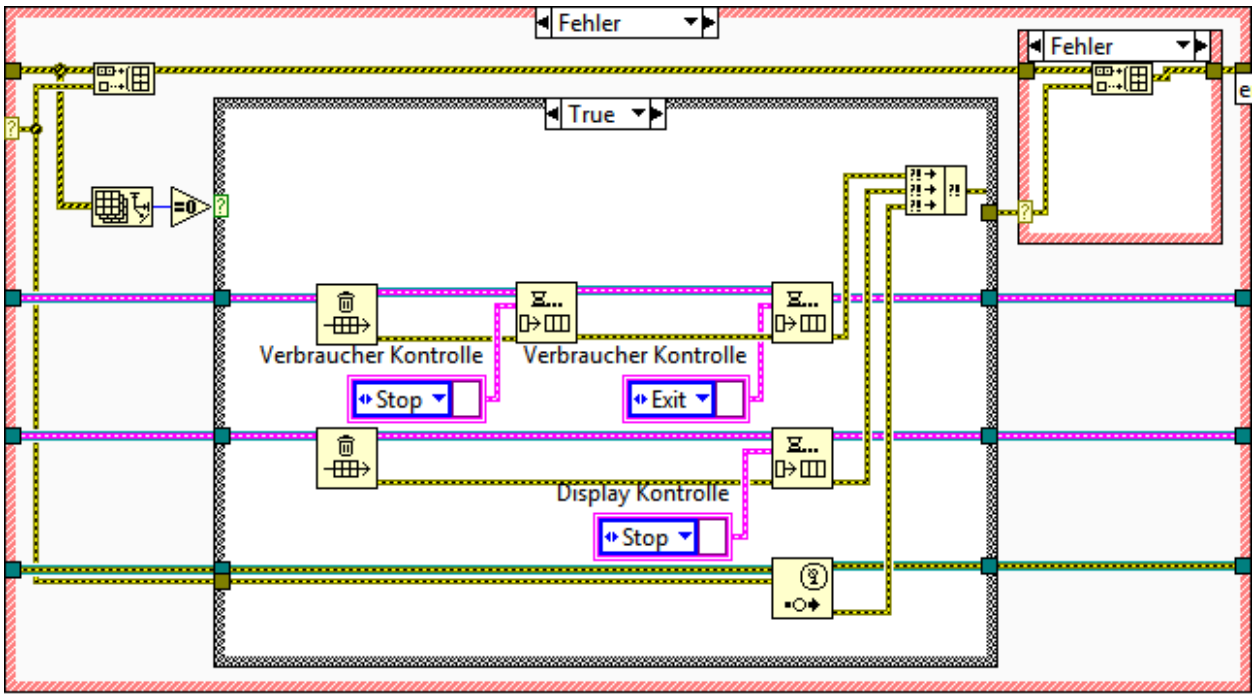

Abbildung 23: Fehlerbehandlung

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