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 eine Gegenüberstellung der Vor- und Nachteile des verwendeten graphischen Programmiersystems.

Abstract

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

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

FGV funktionale globale Variable

LabVIEW Laboratory Virtual Instrumentation Engineering Workbench

LJ Light Jockey (dt. Lichttechniker)

LLVM Low-Level Virtual Machine

NI National Instruments

subVI unter Programm eines VIs

VI virtuelles Instrument

VM virtuelle Maschine

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 Vielzahl von Scheinwerfern angesteuert werden kann.

1.2 Anforderungen

Der Light Jockey (LJ) stellt für verschiedene Lichtkanäle Intensität und Farbe ein. Für eine Gruppe von Lichtkanälen (Set) kann eine Wartezeit, Überblendungszeit, Nachlaufzeit und ein 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 einen Klick auf die OK-Schaltfläche wird das erstellte Set an das Ende einer wählbaren Queue angefügt.
Hat der Bediener einige Sets angelegt und zu einer Queue zusammengefügt, 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. Danach 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 der nach Speicher- bzw. Ladepfad gefragt wird.

1.3 Dokumentation

Der LabVIEW Quellcode sowie eine ausführbare Windows-Anwendung mit dem notwendigen RunTime-Installer befindet sich auf der beigelegten CD. Ebenfalls enthalten sind Detailansichten der wesentlichen Programmcodeausschnitte.

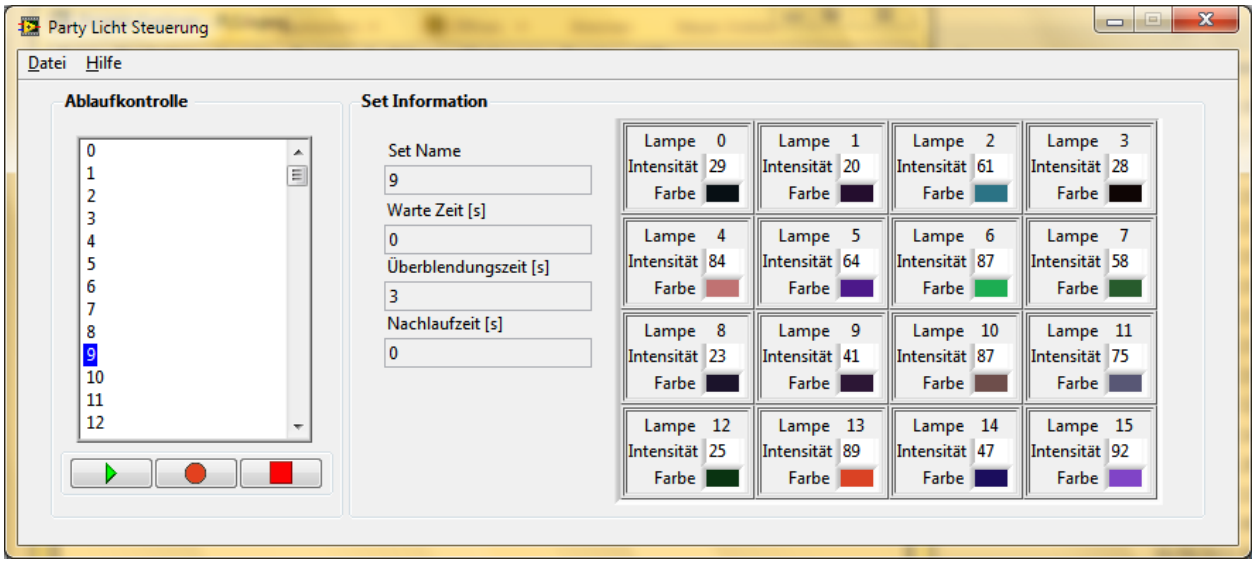

Abbildung 1: Bedienoberfläche für die Party-Lichtsteuerung

1.4 Aufbau der Arbeit

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

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 das Datenflussmodell der Ausführung von VIs zugrunde. Ein Blockdiagrammknoten (Bsp. Addition) wird ausgeführt, sobald alle 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 zur Laufzeit vergleichbar mit der anderer Hochsprachen wie C oder Pascal. Dieser Compiler ist der Schlüssel für die Produktivität von LabVIEW. Er abstrahiert Aufgaben wie die Speicherzuweisung und die Verwaltung von Threads. In der aktuellen LabVIEW Version 2010 wurde in den Compiler eine „Low-Level Virtual Machine“ (LLVM) aufgenommen. Die LLVM ist eine Open-Source-Compiler-Infrastruktur und soll die Codeausführung beschleunigen. „Die Architektur von LLVM basiert auf einer virtuellen Maschine (VM), die einen Prozessor virtualisiert. Die VM ist in der Lage, die intern generierte Sprache des Compilers während der Ausführung für den Prozessor des aktuellen Systems zu übersetzen.“[Sam11] Besonders hierbei ist, dass LLVM hocheffizient ist, was die Übersetzung auch in Echtzeit ermöglicht. [Lat11] [NI11e]
Abbildung 2 zeigt eine Demonstration einer LabVIEW Applikation. Es wird aus den Eingängen A und B ein Ausgang C berechnet. Die Formel wird im Blockdiagramm abgebildet. Sie lautet:

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

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

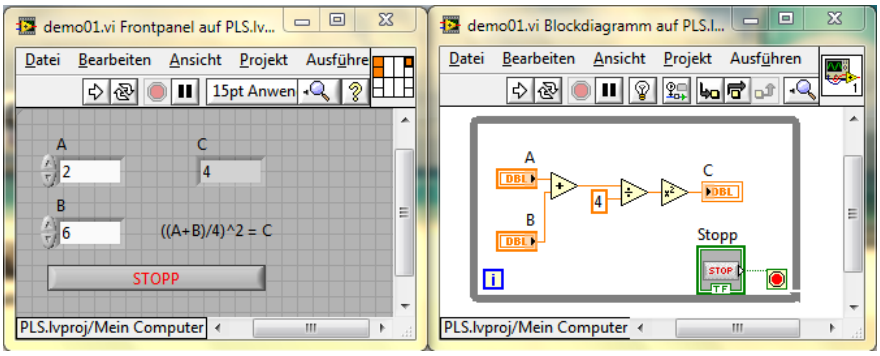

Abbildung 2: VI Demonstration: Links Frontpanel, Rechts Blockdiagramm

2.1 Entwurfsmuster - Design Patterns

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

- Gibt es eine feste Reihenfolge / Sequenz 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 die Problemstellung in Frage kommen. Das sind: der Zustandsautomat, das Master/Slave-Entwurfsmuster, der Ereignisbehandler für Benutzeroberfläche und das Erzeuger/Verbraucher-Entwurfsmuster. Später im Abschnitt 4.1 werde ich auf die getroffene 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. ihre Abbruchbedingung erreicht hat.

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 die Slave-Schleifen, einen bestimmten Code auszuführen. Die Slave-Schleifen werden vollständig ausgeführt und warten dann auf die nächste Benachrichtigung.

2.1.3 Einfache Ereignisbehandlungsroutine für die 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 Programmentwurf

3.1 Datenabstraktion

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 aller nacheinander ablaufenden Lichtersets.

Verbraucher-Queue Die Warteschlange für die Verbraucherschleife - sie enthält Befehle der Erzeugerschleife für die Verbraucherschleife.

Display-Queue Die Warteschlange für die Displayschleife - über diese kommuniziert die Verbraucherschleife mit der Displayschleife.

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:

- *AddLichterset*, *GetLichterset* und *SetLichterset*
- *getAnzahlLichtersets*
- *getLeeresLichterset*

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

- Initialisierung und Update des Front Panels
- Auswahl des Lichtersets aus der Lichterset-Queue
- De-/Aktivieren von Schaltflächen
- Update der Lichterset-Queue

Timing Modul Das Modul sorgt für die Berechnung der verschiedenen Zeiten:

- Wartezeit
- Überblendungszeit
- Nachlaufzeit

Fehler Modul Das Modul führt die Fehlerbehandlung durch.

- Fehler ausgeben
- Fehler behandeln

Datei Modul Das Modul sorgt für die 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. Das Programm wird in handhabbare Funktionen und Verzweigungen gegliedert und dargestellt. Abbildung 3 zeigt das Ablaufdiagramm für die Abspiel-Funktion in der Party-Licht-Steuerung. Ein Knoten repräsentiert einen Zustand.

3.3 Datenflussdiagramm

Datenflussdiagramme haben die Aufgabe, zu zeigen, welchen Weg die Daten durch eine Applikation nehmen. Abbildung 4 zeigt das Datenflussdiagramm 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.

4 Implementierung

4.1 Auswahl des Design Patterns

Bei der Wahl des Entwurfsmusters habe ich mich für das Erzeuger/Verbraucher - Entwurfsmuster (siehe Abschnitt 2.1) entschieden. Bei diesem Pattern kann man die Ereignisbehandlung vom User-Interface und dem auszuführenden Code gut trennen. Des Weiteren ist es möglich die verschiedenen Schleifen parallel auszuführen, durch hinzufügen weiterer Schleifen bleibt das Programm gut skalierbar. Ein anderer Grund für die Verwendung dieses Design Patterns ist die interne Kommunikation über Queues, das erleichtert das steuern des User-Interfaces.
Das Entwurfsmuster wird wie folgt abgewandelt implementiert. Die Erzeugerschleife reagiert auf Events vom User-Interface. Hauptbestandteile des User-Interfaces sind die drei

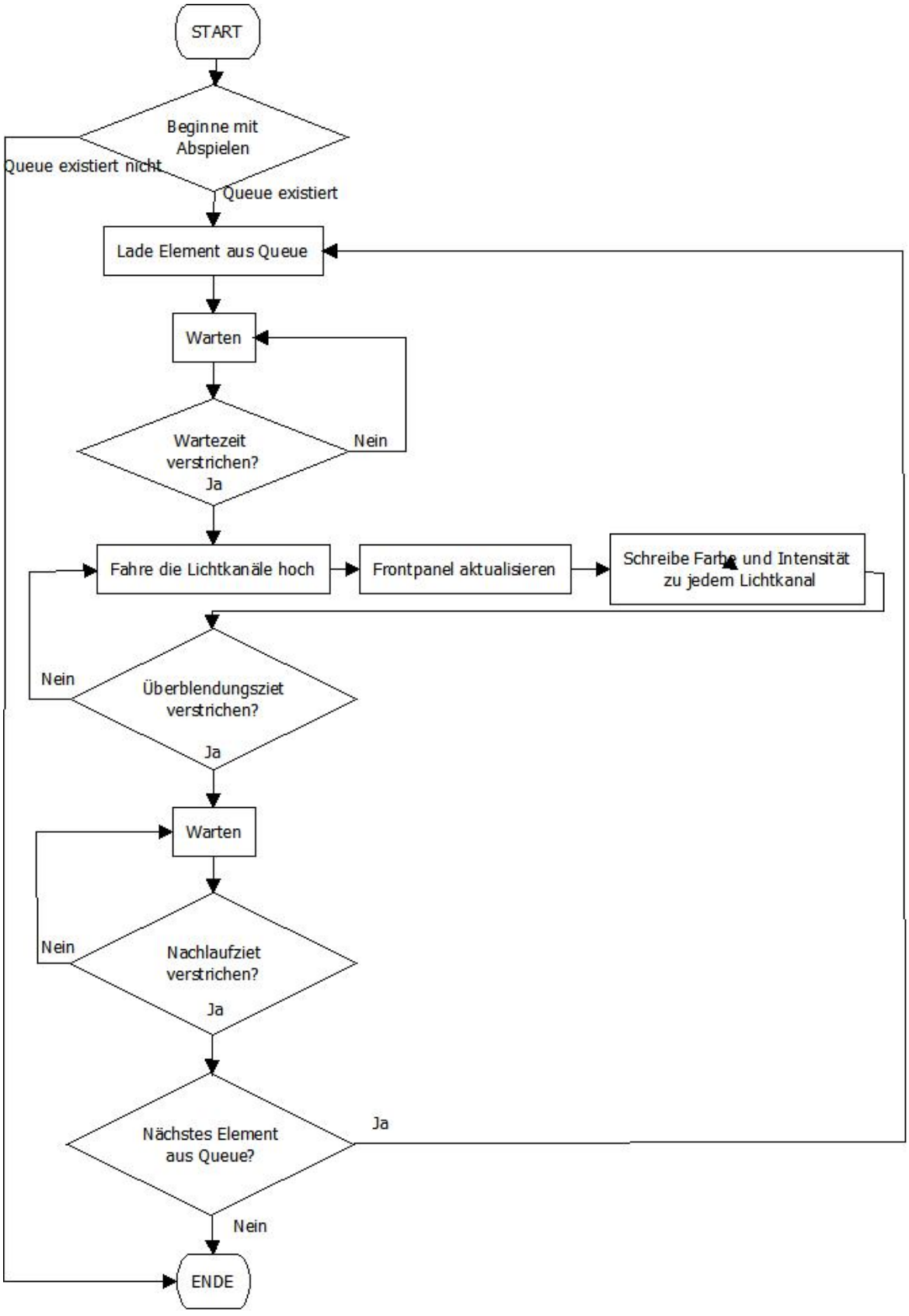

Abbildung 3: Ablaufdiagramm für die Abspiel-Funktion

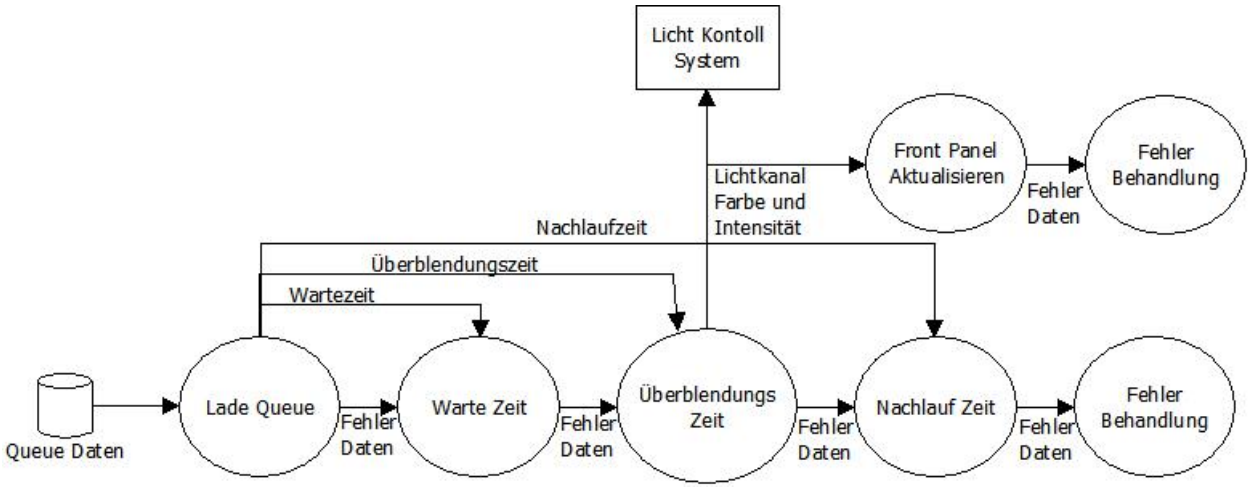

Abbildung 4: Datenflussdiagramm für die Abspiel-Funktion

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
- beenden

Daten, welche die Erzeuger- an die Verbraucherschleife sendet, sind die Daten der Lampen-Sets die beim Aufnehmen, Speichern oder Laden entstehen. In der Verbraucherschleife werden alle Berechnungen durchgeführt. Die Verbraucherschleife kommuniziert über eine Display-Queue mit der Displayschleife. Die Displayschleife hat die Aufgabe, Änderungen am User-Interface durchzuführen. Diese können sein:

- Initialisierung des Front Panels
- Update der Lichtkanäle

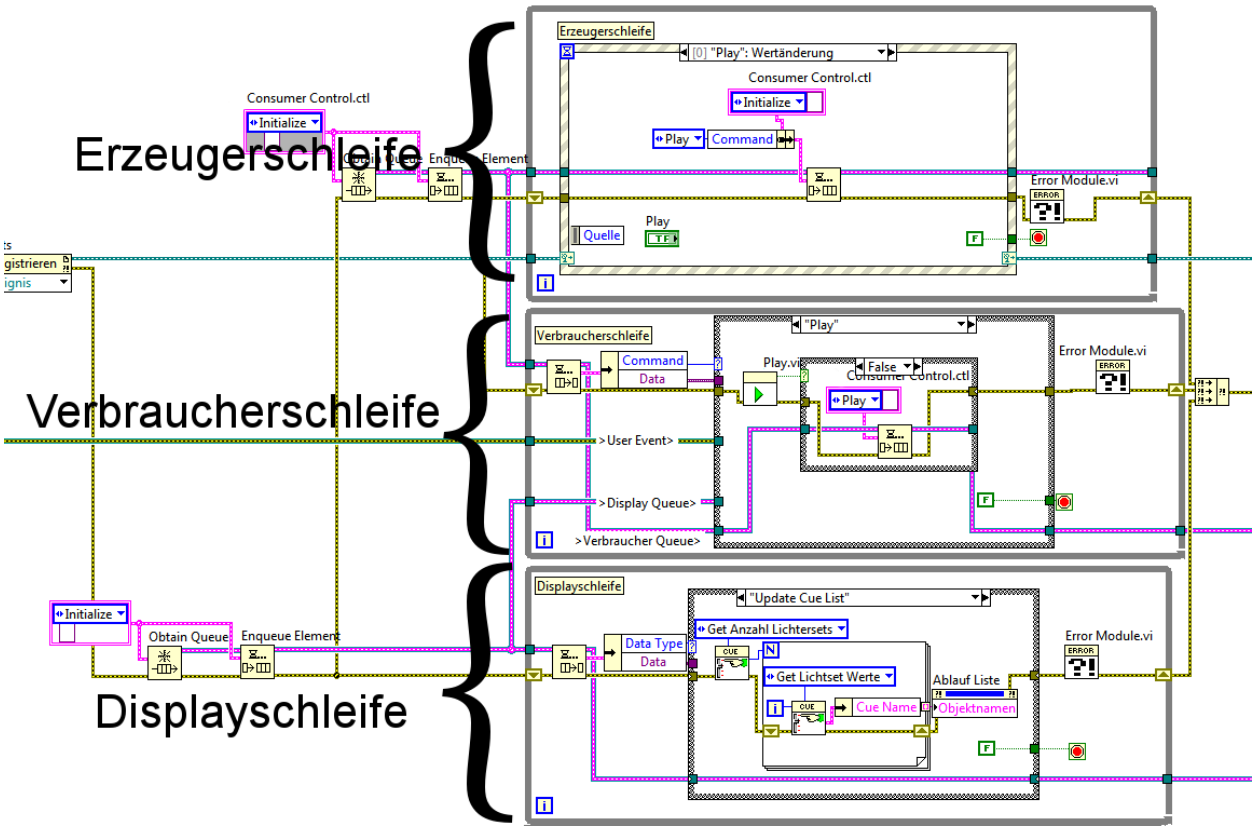

Abbildung 5: Design Struktur für das Blockdiagramm

- Auswahl eines Lichtsets
- Aktivieren und Deaktivieren von Schaltflächen
- Update der Set-Ablaufliste
- 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 wartungsfreundlich und gut 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 ßäubert"das User-Interface. Abbildung 12 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 13 im Anhang zeigt das Blockdiagramm vom *„shutdown.vi“*.

4.3 User Interface

Die Displayschleife 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 Displayschleife von der Verbraucherschleife erhält, wird der entsprechende Case aufgerufen. Beim Kommando Stopp wird die While-Schleife beendet. Sollte ein Fehler in einem Case aufgetreten sein, wird dieser vom *„Error Module.vi“* behandelt. Dazu mehr im Abschnitt 4.9 Fehlerbehandlung. Im Folgenden werden die einzelnen Cases in der Displayschleife erläutert.

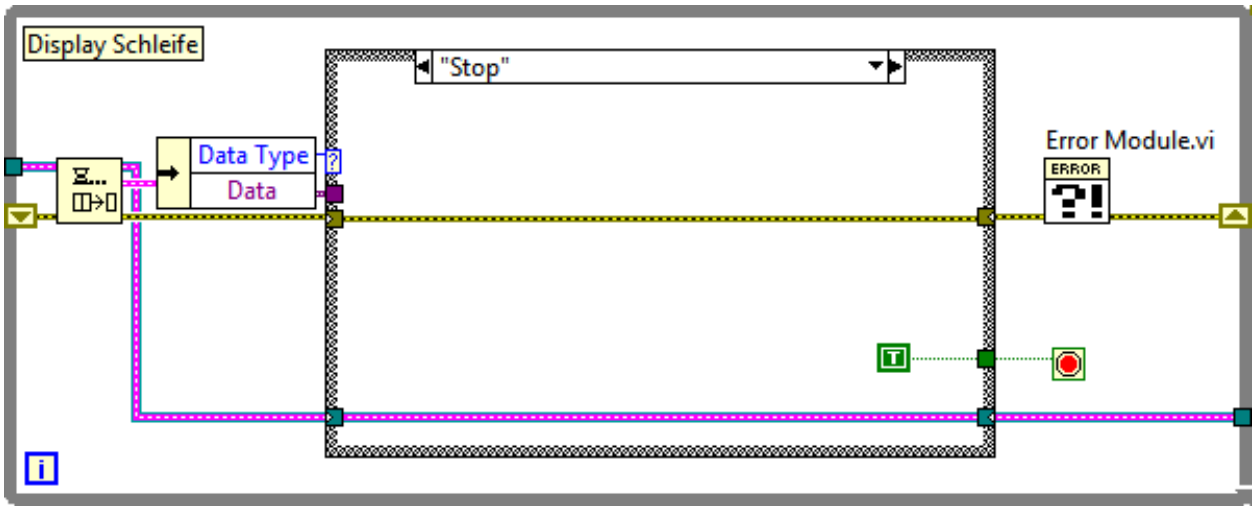

Abbildung 6: Displayschleife

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 in Abbildung 14 dargestellt.

4.3.2 Auswahl eines Lichtsets aus der Lichterset Queue

In diesem Case wird eine Spalte in der 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 in Abbildung 15 dargestellt.

4.3.3 Aktivieren und Deaktivieren von Schaltflächen

Wenn eine Lichterset Queue abgespielt wird, schaltet dieser Case die Schaltfläche Aufnehmen und die Ablaufkontrollliste inaktiv. Das verhindert, dass der Nutzer während eines Abspielvorgangs ein neues Lichterset anlegt und die Ablaufkontrollliste "durcheinander bringt".
Der entsprechende Ausschnitt aus dem Programmcode ist im Anhang in Abbildung 16 gezeigt.

4.3.4 Update der Set-Ablaufliste

Dieser Case aktualisiert die Liste in der Ablaufkontrolle wenn ein neues Lichterset aufgenommen wurde.
Der entsprechende Ausschnitt aus dem Programmcode ist im Anhang in Abbildung 17 gezeigt.

4.3.5 Update der Lichtkanäle

Dieser Case aktualisiert 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 Informationen zum ausgewählten Lichterset anzuzeigen.
Der entsprechende Ausschnitt aus dem Programmcode ist im Anhang in Abbildung 18 dargestellt.

4.4 Aufnahme-Funktion

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

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

Nach Bestätigung über die OK-Schaltfläche werden die gesammelten Daten in die Queue geschrieben und das User-Interface aktualisiert.
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 das Objekt aus der Queue und hängt das neue Objekt an die Lichterset-Liste an. Dann gibt sie das Kommando zum Aktualisieren des User-Interfaces an die Displayschleife.

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 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.

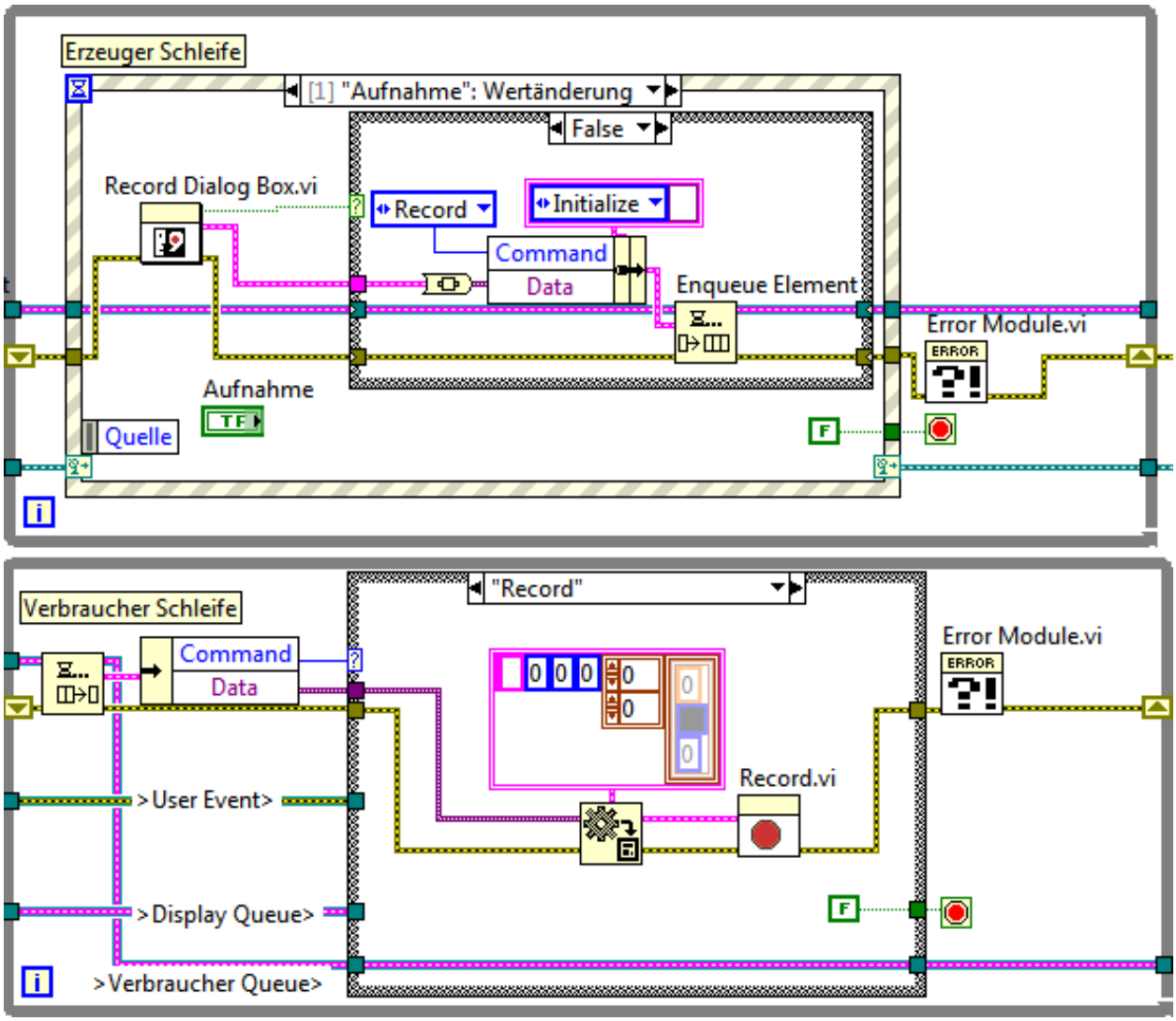

Abbildung 7: Aufnahme-Funktion

Das hat den Vorteil, dass Daten nicht immer bei jedem VI Aufruf mitgeführt werden müssen. Die Daten werden einmal beim Initialisieren mitgegeben und halten sich dann im Speicher. [NI10]

Die Timing FGV hat zwei Kommandos: starten(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 im Anschluss 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.

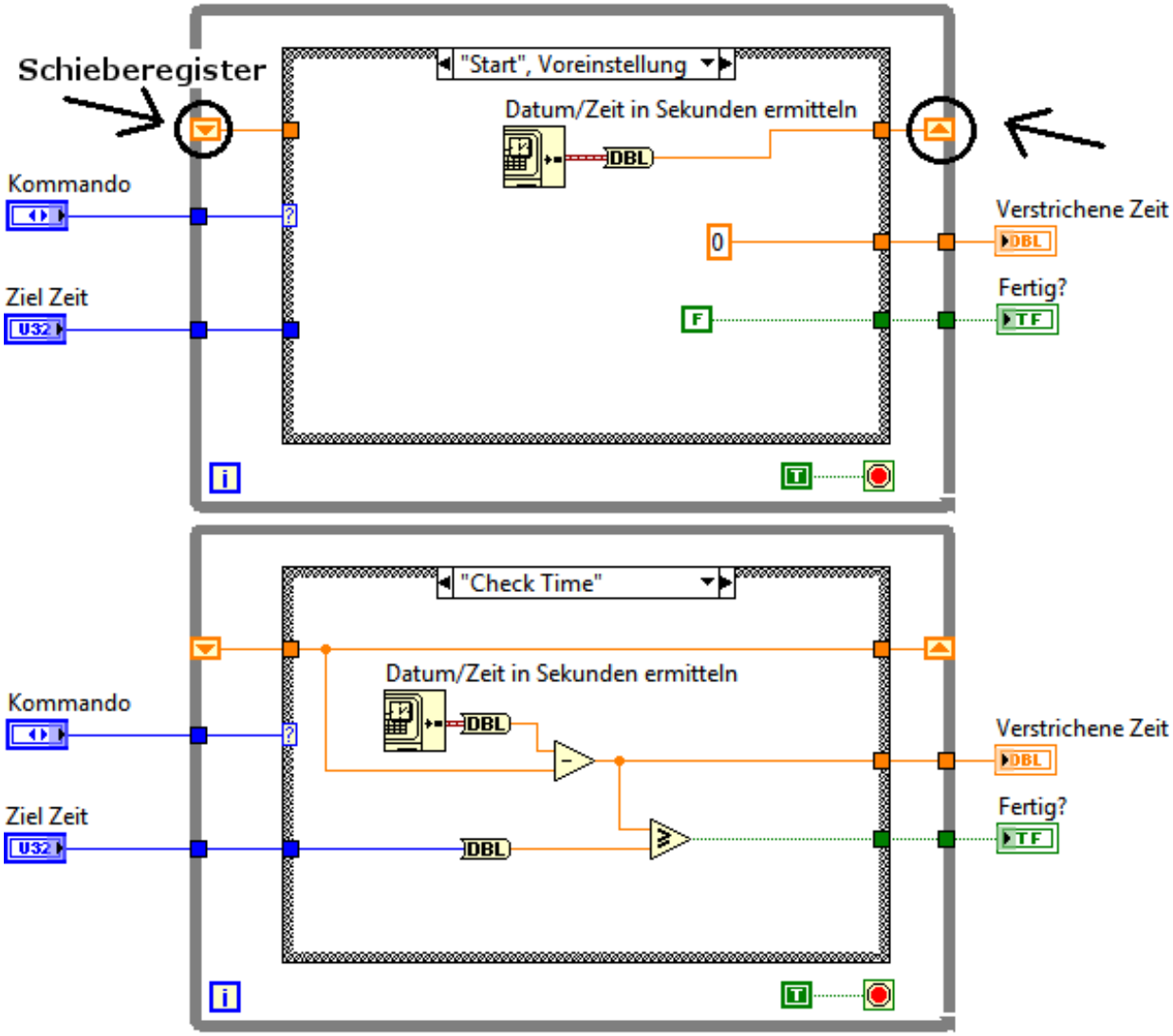

Abbildung 8: FGV Timing VI

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 einen Zustand. Gibt es zurück, dass 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 durchlaufen wird, solange bis der Zustandsautomat am Ende ist. Zur Berechnung, ob die verschieden Zeiten abgelaufen sind, wird das *„timing.vi“* aufgerufen. Im Anhang in Abbildung 20 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 Zustandsautomat der Abspiel-Funktion unterbrochen. Das geschieht, indem die Verbraucher Queue geleert und der Zustandsautomat zurückgesetzt wird. So wird garantiert, dass keine Nachrichten mehr in der Verbraucher-Queue sind und der Zustandsautomat beim nächsten Anlauf wieder am Anfang startet. Der Code ist im Anhang in Abbildung 21 gezeigt.

4.8 Speichern und Lade Funktion

Die Funktionalität zum Speichern und Laden in eine Datei ist für ein LJ unerlässlich. So kann der LJ während einer Probe alle Einstellungen setzen und diese in einer Datei speichern. 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 einen 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 werden die vorgefertigten LabView VIs genutzt: Öffnen, Schreiben und Schließen einer Datei. Der Code ist im Anhang in Abbildung 22 dargestellt.

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. Die Elemente dieses Arrays werden dann in die Lichterset Queue geschrieben. Der Code ist im Anhang in Abbildung 23 dargestellt.

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 indem das Kommando Exit in die Verbraucher Queue geschrieben wird. Zum Beenden der Displayschleife wird das Kommando Stopp an die Display Queue gesendet. Empfängt eine Schleife ein Stopp-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 in Abbildung 24 zu finden.

5 Funktionstest

Bei einem funktionsorientierten Test wird ein Modul auf seine Spezifikationen geprüft. Exemplarisch wird hier das Timing Modul einem Black-Box-Funktionstest unterzogen. Das heißt, das die genaue Beschaffenheit des Moduls nicht betrachtet wird. Nur nach außen sichtbares Verhalten fließt in den Test ein.
Die Spezifikation kann dem Abschnitt 4.5 entommen werden. Das Modul verfügt über folgende Eingänge:

Kommando Gibt an, was das Modul tun soll. Der Datentyp ist eine Typdefinition mit einem Enum aus Strings(start und check).

Zielzeit Gibt die Zeit in Sekunden deren Ablauf geprüft werden soll. Der Datentyp ist ein 32 Bit langes vorzeichenloses Long. Gültig sind Werte im Bereich von 0 bis 4.294.967.295.

Fehler Enthält Fehler Informationen von vorangegangenen VIs.

Die Ausgänge des Moduls sind:

Verstrichene Zeit Gibt die Zeit an, die seit dem Start verstrichen ist. Der Datentyp ist ein 64 Bit Double.

Fertig Ein Boolean der true zurück gibt wenn die Zielzeit abgelaufen ist und false wenn die Zeit noch läuft.

Fehler Gibt Fehler aus, die im VI oder durch vorangegangene VIs entstanden sind.

Folgende vier Testfälle werden auf dem Modul ausgeführt. Als Testwerkzeug dient das Frontpanel des VIs und eine Stoppuhr.

Zeit [s]	Eingabe Parameter	Erwartete Ausgabe	Test Resultat
0	Kommando: start, Zielzeit: 5 kein Fehler	Fertig: false Verstrichene Zeit: 0 kein Fehler	erfolgreich
3	Kommando: check Zielzeit: 5 kein Fehler	Fertig: false Verstrichene Zeit: 3 kein Fehler	erfolgreich
6	Kommando: check kein Fehler	Fertig: true Verstrichene Zeit: 6 kein Fehler	erfolgreich
10	Kommando: start Zielzeit: -3 kein Fehler	Fertig: false Verstrichene Zeit = 0 Fehler: "negative Startzeit“	erfolgreich
15	Kommando: start Zielzeit: 5 Fehler:„Queue Daten Error“	Fertig: false Verstrichene Zeit: 0 Fehler: „Queue Daten Error“	erfolgreich
20	Kommando: check Zielzeit: 5 Fehler:„Queue Daten Error“	Fertig: false Verstrichene Zeit: 0 Fehler: „Queue Daten Error“	erfolgreich

Abbildung 9: Testfälle Timing Modul

6 Stress- und Ladetest

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-Managers 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 10 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 im 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. Der zuvor reservierte Speicher wurde wieder freigegeben. Während des Abspielvorgangs war der Prozessor im Mittel mit 14% ausgelastet.

- Die Anwendung verursacht keinen Speicherüberlauf.
- Es wird kein Speicher unnötig allokiert.
- Der Prozessor ist mit der Anwendung nicht überlastet.

Somit gilt der Test als bestanden.

7 Einsatz der Anwendung

LabVIEW bietet die Möglichkeit, über einen Dialog aus einem Projekt heraus eine ausführbare Windows- (.exe) oder MAC OS (.app) Anwendung wahlweise mit Installer zu erstellen. Eine ausführbare Windows-Anwendung 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 die Windows-Laufzeitumgebung sind [NI11d]:

- Prozessor: Pentium III/Celeron 866 MHz oder gleichwertig
- Hauptspeicher: 256 MB

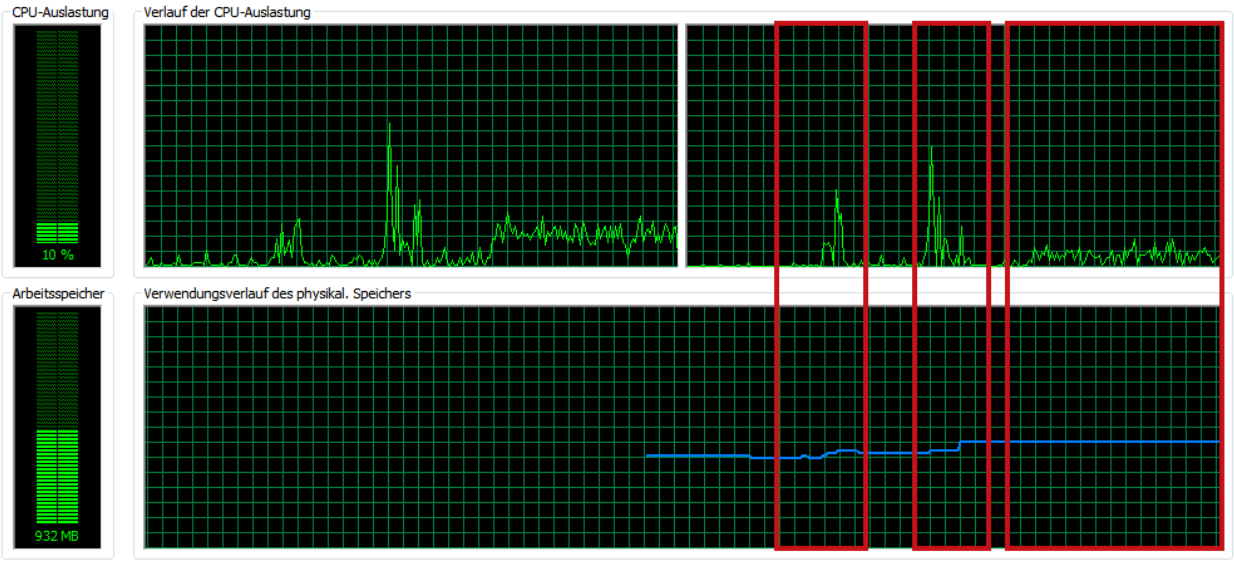

Abbildung 10: Verlauf der Prozessor- und Arbeitsspeicher Auslastung

- Festplattenspeicher: 1 GB
- Betriebssystem: XP

7.1 Webservice

Mit LabVIEW lässt sich ein Webservice erstellen auf den der Anwender via Inter- oder Intranet mit einem Standard-Browser zugreifen kann. Rechenintensive Anwendungen können von einem Server bearbeitet werden und müssen nicht die Rechnerressourcen des Clients beanspruchen. Mit einem Webservice lassen sich Prozesse von der Ferne aus überwachen. Anwendungen hierfür sind zum Beispiel die Produktionsüberwachung und Produktionssteuerung von Maschinen. Mitarbeiter müssen nicht von Maschinenterminal zu Maschinenterminal laufen sondern könnten zum Beispiel von einem Heimarbeitsplatz via Internet eine ganze Produktionsanlage überwachen.
VIs können von jedem HTTP-fähigen Web-Client unter Angabe einer URL aus aufgerufen werden. Der Austausch von Daten kann dann mit herkömmlichen HTTP-Methoden wie POST erfolgen.
Web-Clients tauschen Daten mit einer Webdienstapplikation aus, indem Sie HTTP-Anfragen an definierte URLs senden. Der Webdienst akzeptiert die Anfrage und sendet Daten im XML-, HTML-, JSON- oder TXT-Format zurück an den Web-Client. Ein Web-Client kann beispielsweise eine Anfrage mit zwei Werten an einen Webdienst senden, der dann die Summe aus diesen Werten bildet und das Ergebnis an den Client sendet.

[NI10]

8 Abschließende Betrachtung

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

8.1 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“ [1)] achten. Hierfür stehen verschiedene Möglichkeiten zur Verfügung. Zum Beispiel Semaphore oder Warteschlangen.
Die Umsetzung es Projekts über das Front Panel von LabVIEW hat gezeigt, dass es eine bequeme Möglichkeit bietet, schnell eine einfache und gute Bedienoberflächen zu erstellen.
Das Blockdiagramm ermöglicht durch die graphische Darstellung des Programmablaufs eine gute Lesbarkeit. Anhand verschiedener Farben lassen sich die Datentypen und ihr Ursprung gut erkennen.
Ein weiterer Vorteil, den LabVIEW bietet, ist die Unterstützung von Kommunikationsprotokollen wie TCP/IP. Es ist somit möglich auch 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 Lizenze-Kosten an. Die „NI Developer Suite Core“ für Windows kostet 5.800 Euro. Hier gibt es auch kostenlose Versionen mit vermindertem 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.

[1)] Race Condition: Ein kritischer Wettlauf, auch Wettlaufsituation ist in der Programmierung eine Konstellation, in der das Ergebnis einer Operation vom zeitlichen Verhalten bestimmter Einzeloperationen abhängt.[Wik11]

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.
Abschließend lässt sich sagen, das LabVIEW trotz der beschriebenen Nachteile eine adäquate Lösung bietet, Programme schnell und effizient zu entwickeln.

8.2 Ausblick auf Erweiterungen

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

8.2.1 Ansteuerung von Hardware

Möchte man externe Lichtkanäle ansteuern, könnte man ein dritte Verbraucherschleife 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 Farbe und Intensität von 16 Lampen steuern.[NI11c]

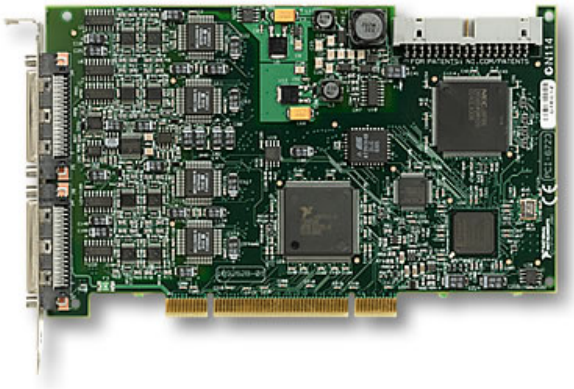

Abbildung 11: NI PCI-6723 mit 32 Analogausgänge [NI11c]

8.2.2 Multilingualität

Um die Applikation einer größeren Zahl an Anwendern zur Verfügung zu stellen, kann eine Mehrsprachigkeit implementiert werden. Dazu kann der Nutzer beim Start der Anwendung nach seiner präferierten Sprache gefragt werden. Über eine Tabelle mit den zur Verfügung stehenden Sprachen kann ein Auswahl für die Beschriftung der Front Panel Elemente getroffen werden.

Literatur

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

[Lat11] LATTNER, CHRIS: *LLVM Language Reference Manual*. `http://llvm.org/docs/LangRef.html`, 2011. Zugriff am 11.6.2011 um 15:03 in Datei *„LLVM Assembly Language Reference Manual.pdf“*.

[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“*.

[Sam11] SAMOCHODOW, MARKI: *Low Level Virtual Machine*. `http://www.opel.samochodziki.info/de/Clang.html`, 2011. Zugriff am 11.6.2011 um 15:14 in Datei *„Lattner-llvm.pdf“*.

[Wik11] WIKIPEDIA: *Race Condition*. `http://de.wikipedia.org/wiki/Race_Condition`, 2011. Zugriff am 7.6.2011 um 13:03 in Datei *„wiki-raceconditions.pdf“*.

Anhang

A.1 Initialisierungsfunktion

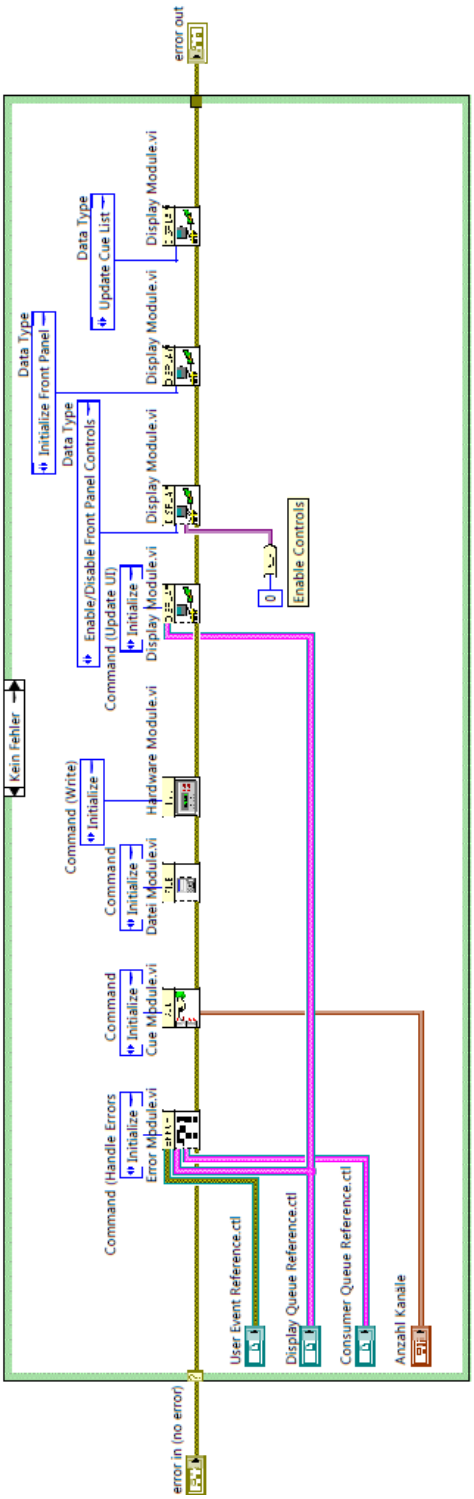

Abbildung 12: Initialisierungsfunktion

A.2 Shutdown-Funktion

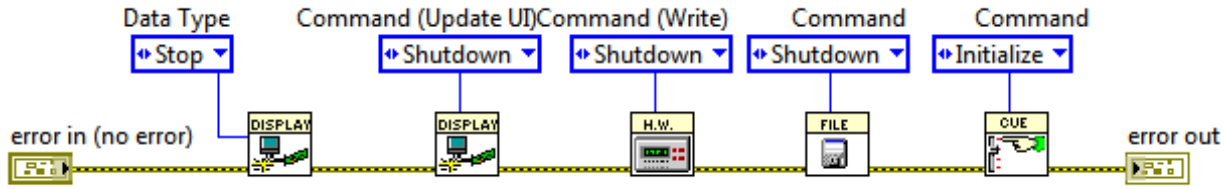

Abbildung 13: Shutdown-Funktion

A.3 Initialisierung des Front Panels

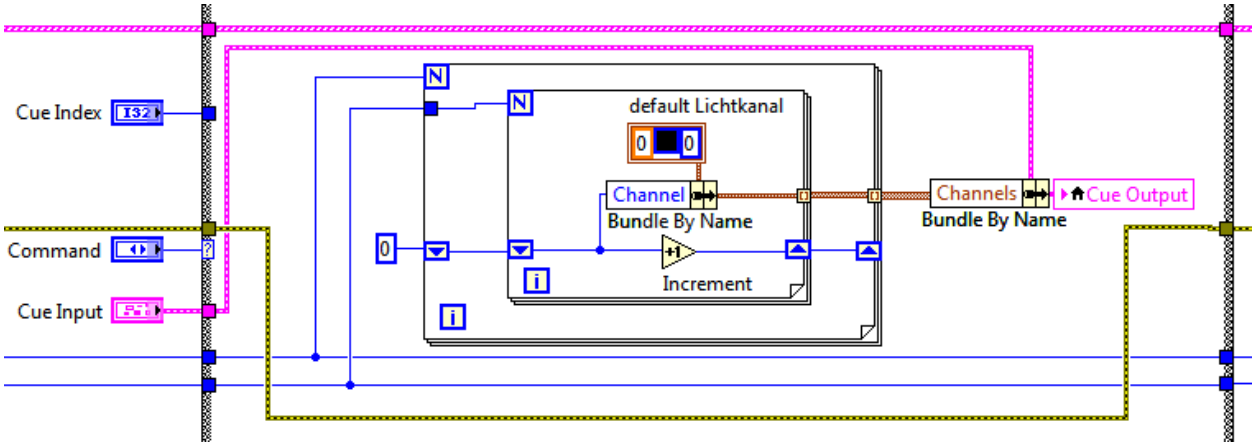

Abbildung 14: Initialisierung des Front Panels

A.4 Auswahl eines Lichtsets aus der Lichterset Queue

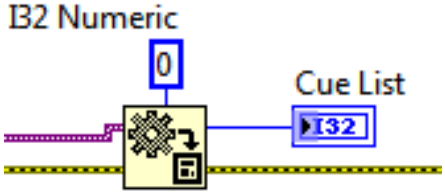

Abbildung 15: Auswahl eines Lichtsets aus der Lichterset Queue

A.5 De-/Aktivieren von Schaltflächen

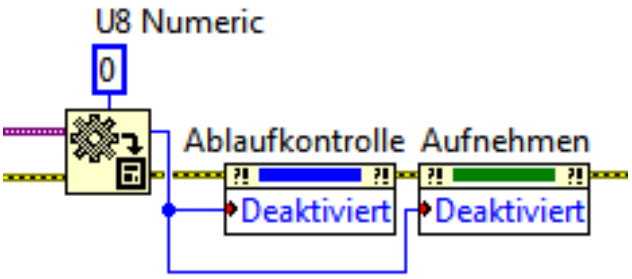

Abbildung 16: De-/Aktivieren von Schaltflächen

A.6 Update der Set-Ablaufliste

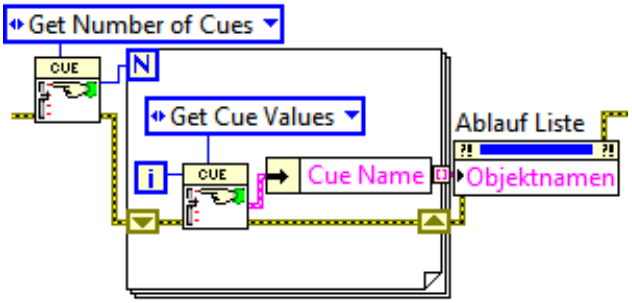

Abbildung 17: Update der Set-Ablaufliste

A.7 Update der Lichtkanäle

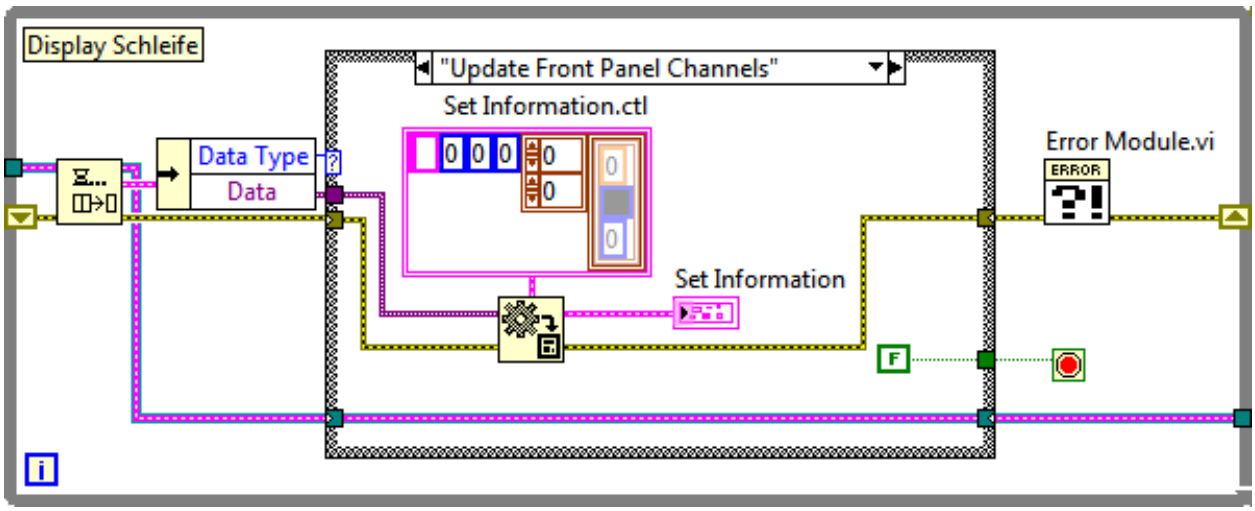

Abbildung 18: Update der Lichtkanäle

A.8 Timing Modul

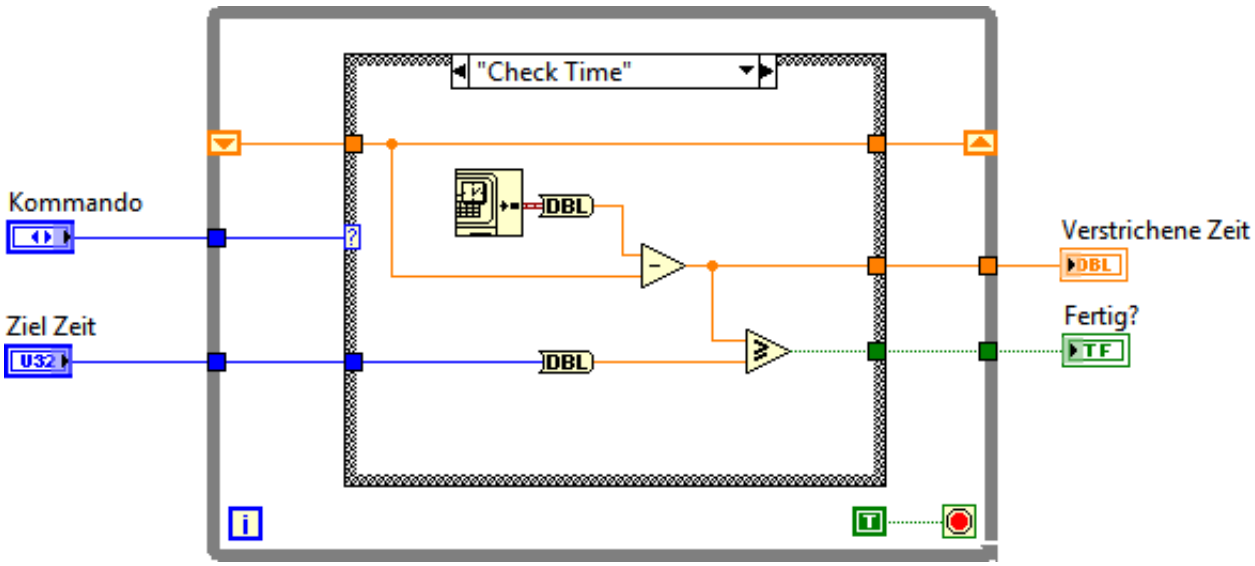

Abbildung 19: Timing Modul

A.9 Abspiel Zustandsautomat - Überblenden

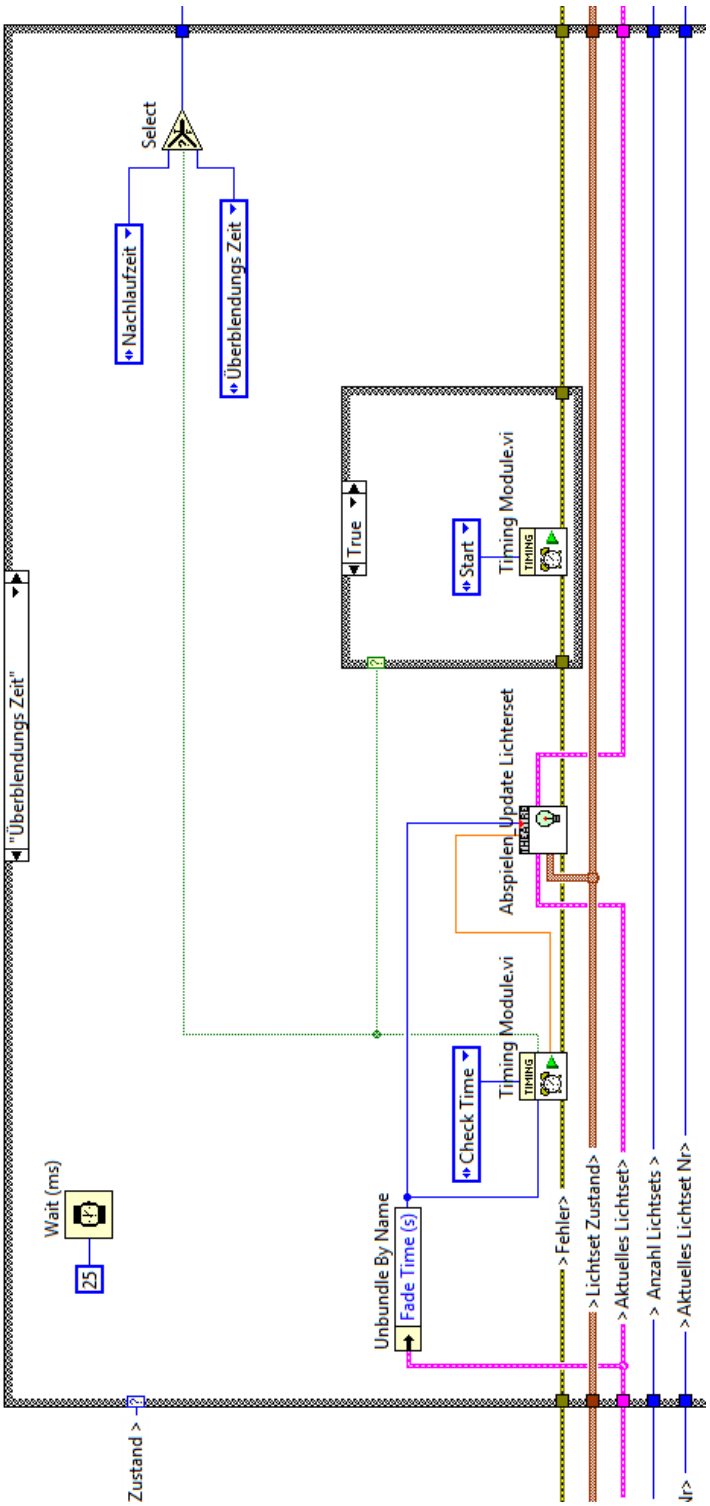

Abbildung 20: Abspiel Zustandsautomat - Überblenden

A.10 Stopp-Funktion

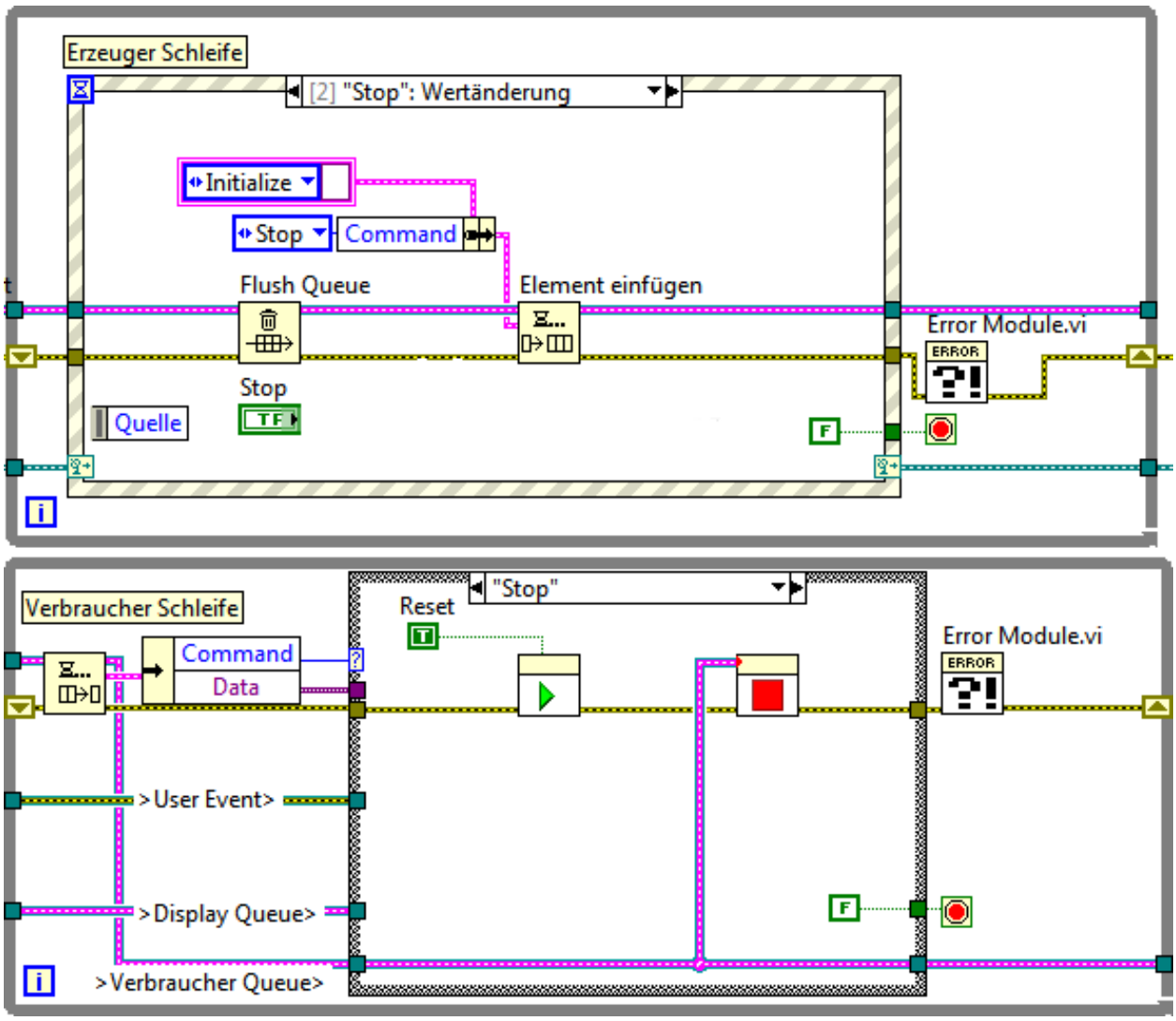

Abbildung 21: Stopp-Funktion

A.11 Speicher-Funktion

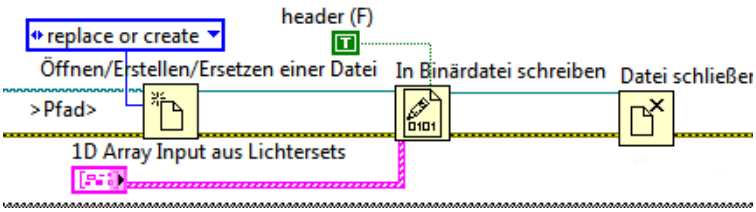

Abbildung 22: Speicher-Funktion

A.12 Lade-Funktion

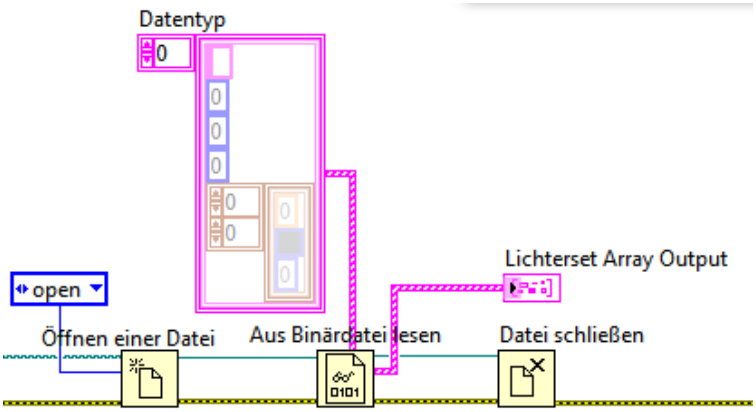

Abbildung 23: Lade-Funktion

A.13 Fehlerbehandlung

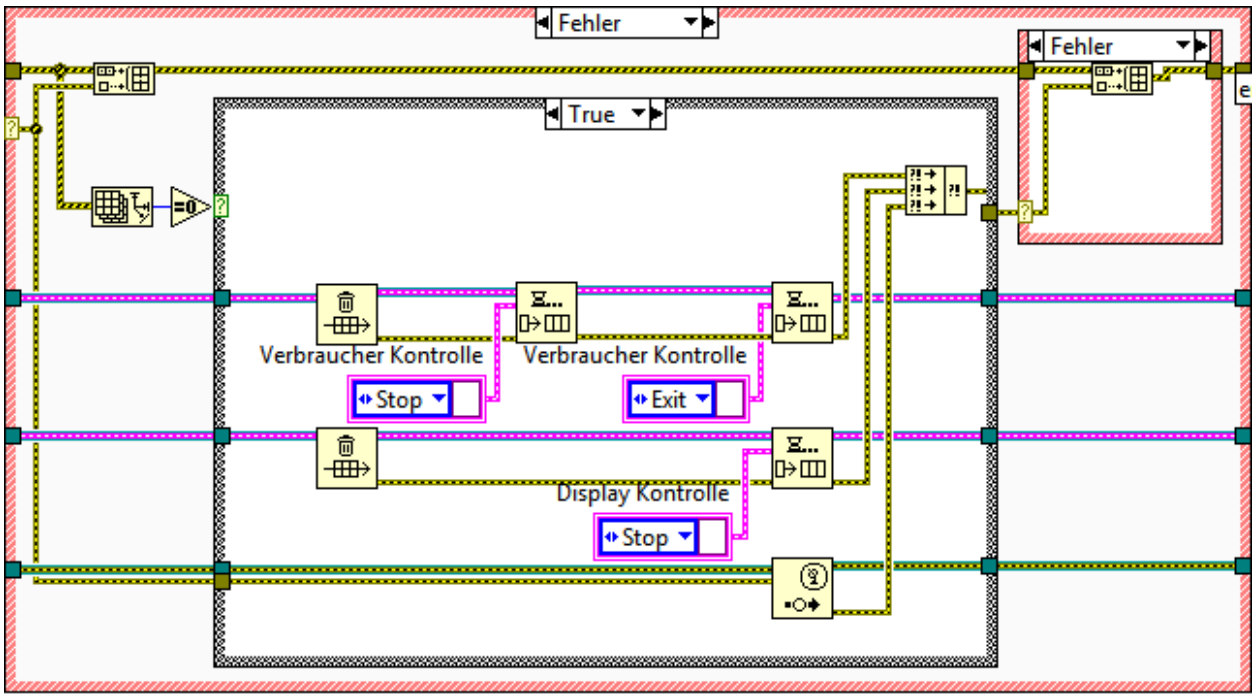

Abbildung 24: Fehlerbehandlung

A.14 Gesamter Überblick über das Main VI

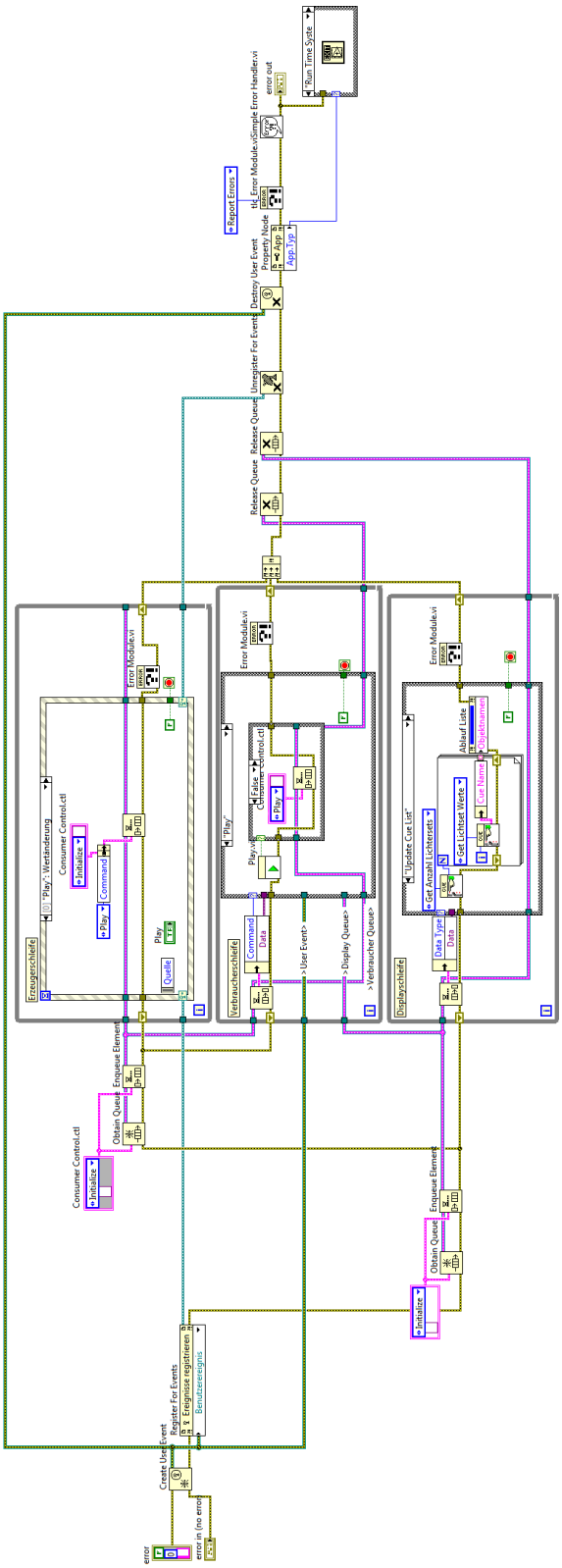

Abbildung 25: Gesamter Überblick über das Main VI

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