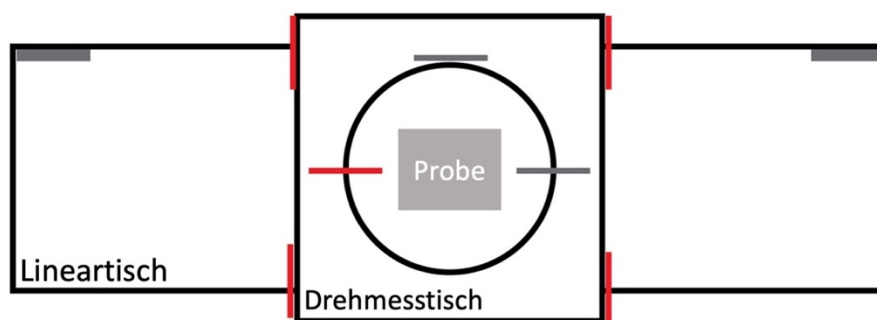


Entwicklung einer Softwareschnittstelle in LabView zur automatischen Steuerung von motorisierten Messtischen



Carlo Biermann
Studentische Hilfskraft
Fachbereich 1/Physik, HTW Berlin
carlo.biermann@student.htw-berlin.de

Inhaltsverzeichnis

| | |
|---|-----------|
| 0. Abkürzungen | 4 |
| 1. Allgemeines | 5 |
| 2. Technische Umsetzung..... | 8 |
| 2.1 Anforderungen..... | 9 |
| 2.2 Lösungsansatz | 10 |
| 3. Senden und Ausführen von Motor-Befehlen über die VISA-Schnittstelle | 11 |
| 3.1 Vorbereitung I: VISA Umgebung..... | 11 |
| 3.2 Vorbereitung II: Eine typische VISA-Anwendung..... | 11 |
| 3.3 Kommunikation zu den SMCs | 12 |
| 4. Entwicklung der Funktionsmodule | 13 |
| 4.1 Starten der Motoren | 14 |
| 4.1.1 Parallele Befehlsverarbeitung..... | 16 |
| 4.2 Stoppen der Motoren..... | 17 |
| 4.3 Motoren initialisieren | 18 |
| 4.3.1 SubVI benutzerAuswahl | 19 |
| 4.3.2 Schrittmodus g10 | 20 |
| 4.4 SubVIs als Module..... | 21 |
| 5. Architektur von OWIS-Steuerung-v1..... | 22 |
| 5.1 Aufbau der <i>Queued State Machine</i> | 23 |
| 5.2 Festlegen der <i>States</i> | 26 |
| 5.3 Beschreibung der <i>States</i> | 28 |
| 5.3.1 Leerlauf..... | 28 |
| 5.3.2 Initialisierung..... | 29 |
| 5.3.3 Pre-Positionierung der Messtische | 31 |
| 5.3.4 Einstellung der Motorgeschwindigkeit..... | 32 |
| 5.3.5 Manuelles Steuern der Motoren | 33 |
| 5.3.6 Fahrweg Zick-Zack | 34 |
| 5.3.7 Fahrweg Kreis | 35 |
| 5.3.8 Stoppen der Motoren..... | 36 |
| 5.3.9 Statusanzeige | 37 |
| 5.3.10 Beenden des Programms | 38 |
| 6. Fahrwegalgorithmen | 39 |
| 6.1 Ausgangssituation vor Beginn der Routinen | 40 |
| 6.2 Fahrweg Kreis | 40 |
| 6.2.1 Pre-Positionierung der Messtische | 41 |
| 6.2.2 Kreis Algorithmus | 41 |
| 6.2.3 Umsetzung in LabView | 43 |
| 6.3 Fahrweg Zick-Zack | 47 |
| 6.3.1 Pre-Positionierung der Messtische | 47 |
| 6.3.2 Zick-Zack Algorithmus | 48 |
| 6.3.3 Umsetzung in LabView | 50 |

| | |
|--------------------------------|-----------|
| 7. Notstopp-Taster..... | 54 |
| 8. Quellen | 55 |

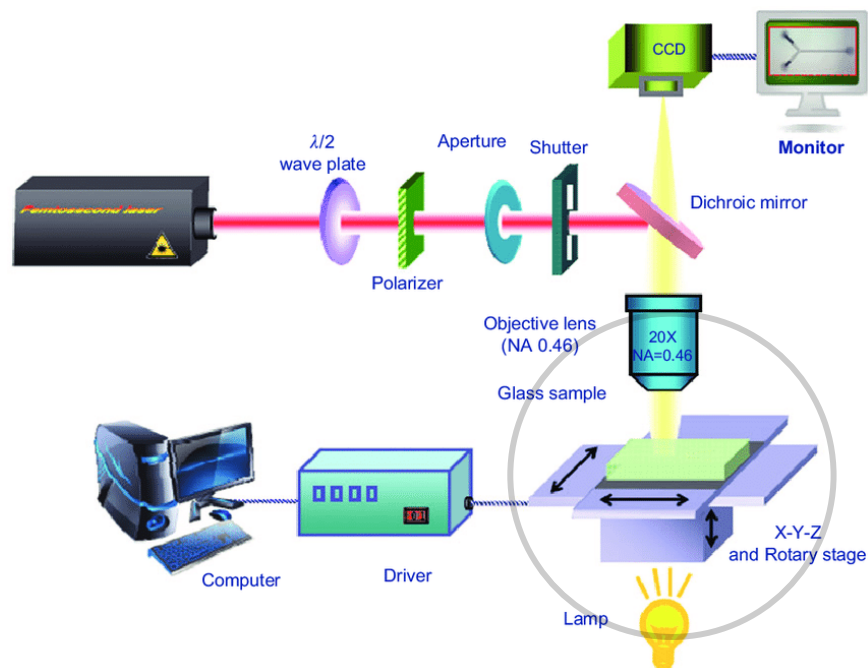
0. Abkürzungen

| | | |
|------|---|--|
| LIP | – | Laserinduzierte Plasmaspektroskopie |
| LT | – | Lineartisch |
| DMT | – | Drehmesstisch |
| NI | – | Native Instruments |
| VISA | – | Virtual Instrument Software Architecture |
| VI | – | Virtual Instrument |
| SMC | – | Stepper Motor Controller |
| QSM | – | Queued State Machine |
| DAQ | – | Data Acquisition |

1. Allgemeines

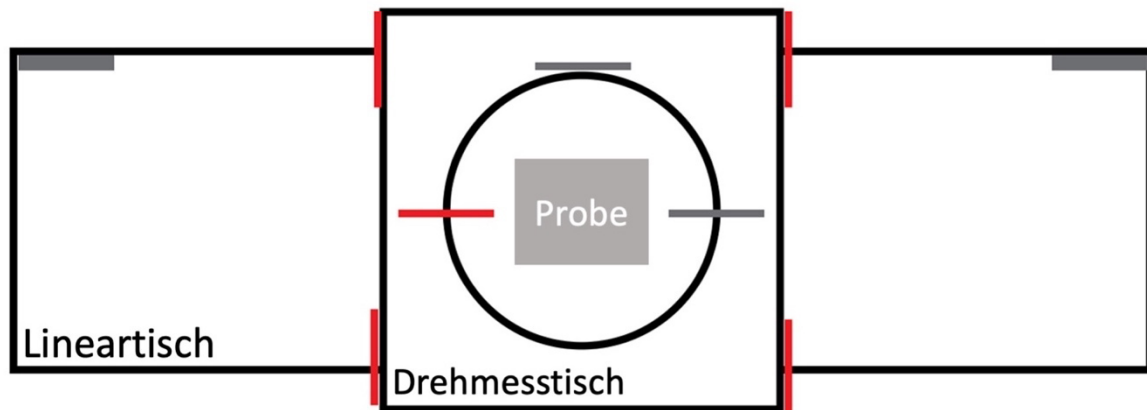
Gegenstand dieses Projekts war die Entwicklung einer Softwareschnittstelle um motorisierte Messtische über eine grafische Benutzeroberfläche in LabView anzusteuern. Dieser Aufbau soll anschließend in einen weiteren Messaufbau der **laserinduzierten Plasmaspektroskopie** (LIP) verwendet werden. Die Abb. 1 zeigt einen solchen schematischen Aufbau mit X-Y-Z- und Drehmesstischen.

Die Messproben werden dabei auf einen Messturm platziert und von oben mit einem Laser bestrahlt. Ziel ist es, die **Messproben kontinuierlich fortzubewegen** um genügend Plasma zu induzieren.



[Abb. 1 – LIP-Aufbau]

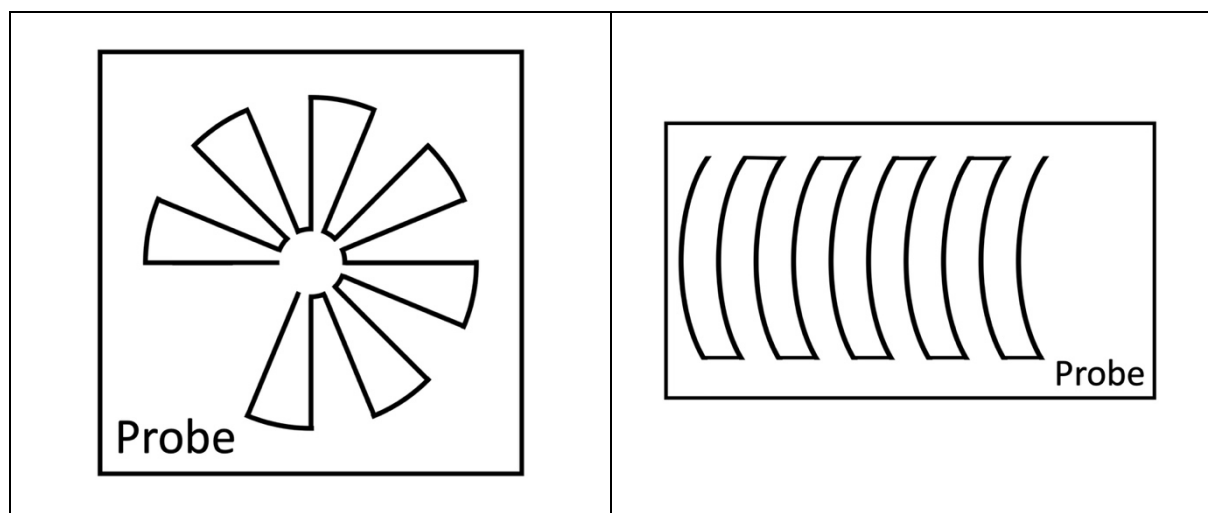
Für die Entwicklung stand ein Lineartisch und ein Drehmesstisch mit entsprechenden Schrittmotoren und Schrittmotor-Controllern zur Verfügung. Der Aufbau ist in Abb. 2 zu sehen.



[Abb. 2 – verwendeter Messbau]

Es galt dabei, einen Fahralgorithmus zu entwickeln, welcher die Messtische so fortbewegt, dass keine Stelle der Probe vom Laser doppelt abgefahren wird. Dabei sollen die Maße der Probe in Betracht gezogen werden und der Fahrweg an diese Maße angepasst werden.

Das Resultat der Entwicklung waren zwei Fahrwegalgorithmen welche ein Muster durch den Laser auf der Messprobe hinterlassen (siehe Abb. 3).

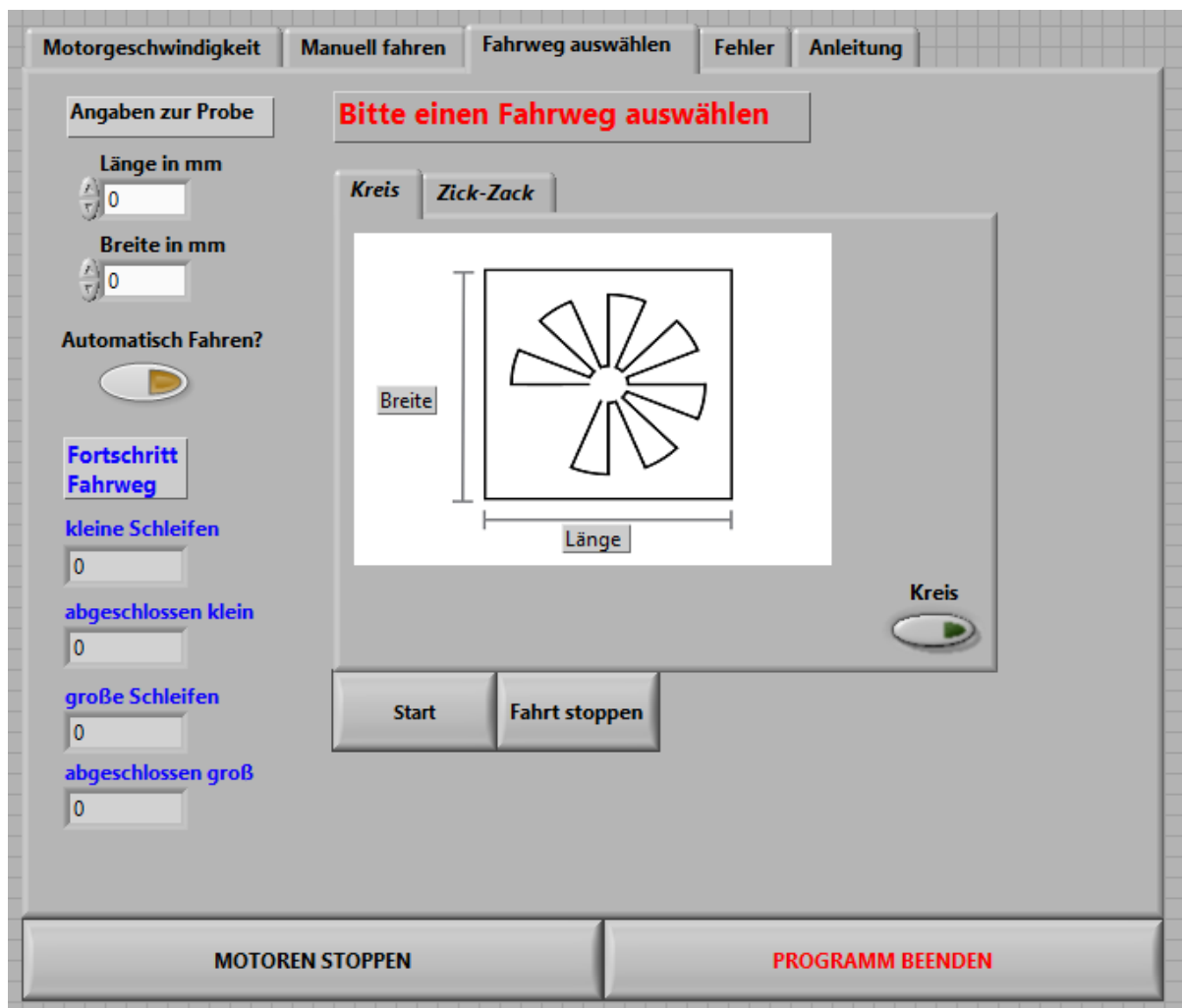


[Abb. 3 – Fahrwegmuster]

Angesteuert wird das gesamte Programm über das sog. *Frontpanel* in LabView, indem der/die Benutzer/in die Länge und Breite der Messprobe eingeben und einen Fahrweg auswählen kann (siehe Abb. 4). Zu weiteren Funktionen des Programms gehören die manuelle/separate Ansteuerung der Messtische und die Einstellung der Motorgeschwindigkeiten.

Ein bereits angefangener Fahrweg kann jederzeit gestoppt werden und die Messtische mit Notstopp-Tastern versehen werden, bei deren Betätigung das Programm den Fahrweg sofort beendet.

Außerdem wird bei der Software eine LabView-Architektur (*Queued State Machine*) verwendet, mit der die Funktionalitäten des Programms relativ simpel erweitert werden können. Das Programm eignet sich deshalb sehr gut dafür, als Ansatz für weitere wissenschaftliche Arbeiten und Entwicklungen von Messroutinen zu dienen.



[Abb. 4 – Frontpanel des Messprogramms]

2. Technische Umsetzung

Die entwickelten Virtual Instruments (VI) in LabView sind in ein Hauptprogramm und mehreren Unterprogrammen unterteilt. Das Hauptprogramm *OWIS-Steuerung-v1* verwendet eine Architektur namens *Queued State Machine* (QSM), bei der sich verschiedene *States* durch eine Benutzereingabe im Frontpanel nacheinander ausführen lassen.

Über das Frontpanel lassen sich folgende Aktionen Ausführen:

- Setzen der Drehgeschwindigkeit der Motoren
- Manuelles Fahren der Motoren durch Taster-Betätigung oder Anklicken der Pfeil-Symbole im Frontpanel
- Setzen der Schrittweite der Motoren
- Auswählen eines Fahrwegs, welcher die komplette Fläche der Probe abfährt
- Starten und Stoppen der Fahrwege

Die QSM-Architektur ermöglicht es algorithmische Fahrwege der Messtische abzufahren und diese jeder Zeit durch den Nutzer oder Endschalter-Betätigung zu unterbrechen/stoppen.

Die Parameter der Fahrwege hängen von der Größe der zu messenden Probe ab und werden im SubVI *counterParameter* berechnet.

Die Kommunikation zu den Schrittmotor-Controllern (SMCs) und Schrittmotoren erfolgt über die NI-VISA-Schnittstelle. Diese Schnittstelle ermöglicht das Ausführen von seriellen Befehlen im String-Format durch die Schrittmotoren.

2.1 Anforderungen

Vor der Entwicklung wurden folgende Anforderungen an das System gestellt:

- Die Messtische fahren den Fahrweg *automatisch* ab.
- Die Messtische sollen mit einer Geschwindigkeit von 100 $\mu\text{m/s}$ bewegt werden.
- Der Laser soll *keine* Stelle zweimal abfahren.
- Es soll so viel Fläche der Materialprobe durch den Laser abgefahren werden wie möglich.
- Die Messtische können manuell per Mausklick oder (Pfeil-) Tastendruck zum Voreinstellen/Kalibrieren hin und her gefahren werden.
- Wenn die Notstopp-Taster der Messtische betätigt werden, soll die komplette Messung gestoppt werden.
- Die Messung sollte auch ohne Betätigung der Notstopp-Taster vom Benutzer abgebrochen werden können.

Verwendete Hardware/Software:

- National Instruments LabView
- OWIS Drehmesstisch
- OWIS Lineartisch
- 2x Nanotec Schrittmotoren
- 2x Nanotec Schrittmotor Controller SMC I33-2
- Versuchsrechner/PC
- NI-USB 6001 DAQ - Messdatenerfassungskarte

2.2 Lösungsansatz

Die Messtische werden jeweils durch die Schrittmotoren angetrieben, welche ihre Achsen drehen und somit den Messturm entlang einer X/Y-Fläche vor und zurück bewegen.

Die Befehle an die Schrittmotoren kommen von jeweils einem Schrittmotor-Controller, welcher per USB mit dem Versuchsrechner verbunden ist. Die Kommunikationsschnittstelle zu den Controller wird über die LabView-Software ermöglicht.

Um diese Anforderungen zu realisieren war eine systematische Aufteilung der Problemstellung notwendig:

- 1) Start- und Stopp-Befehle an **einen Motor** senden
- 2) Start- und Stopp-Befehle an **zwei Motoren gleichzeitig** senden
- 3) Beide Motoren durch Pfeiltasten gleichzeitig steuern
- 4) Beide Motoren eine vorprogrammierte Strecke abfahren lassen

ANMERKUNG:

Bei allen Bewegungsmodi müssen die Motoren mit geringer Latenz gestoppt werden können.

Die nachfolgende Projektdokumentation führt den/die Leser/in schrittweise durch die Entwicklung des LabView-Programms.

Angefangen wird dabei mit der Kommunikationsschnittstelle zwischen LabView und den SMCs bzw. Schrittmotoren. Dabei wurden Funktionsmodule entwickelt, mit denen beide Schrittmotoren über LabView **gleichzeitig** angesteuert werden können. Dies gilt für die Initialisierung der Schrittmotoren, also der Einstellung der Schrittweiten und Drehgeschwindigkeiten und dem Starten und Stoppen der Motoren. Diese Funktionen wurden als Unterprogramme (SubVIs) im Hauptprogramm und weiteren SubVis verwendet.

Anschließend wird erläutert wie diese SubVis in der QSM in *States* implementiert wurden und wie diese States durch Benutzereingaben vom Frontpanel ausgeführt werden.

3. Senden und Ausführen von Motor-Befehlen über die VISA-Schnittstelle

National Instruments bietet eine Programmierschnittstelle (API) zur Steuerung von Messgeräten in LabView in Form von NI-VISA (*Virtual Instrument Software Architecture*).

Die Programmbausteine zur Kommunikation zu diversen Messgeräten sind nach Installation des NI-VISA Treibers in LabView zu finden. Eine ausführliche Dokumentation zu NI-VISA lässt sich auf der offiziellen Webseite finden:

<https://www.ni.com/de-de/support/documentation/supplemental/06/ni-visa-overview.html>

3.1 Vorbereitung I: VISA Umgebung

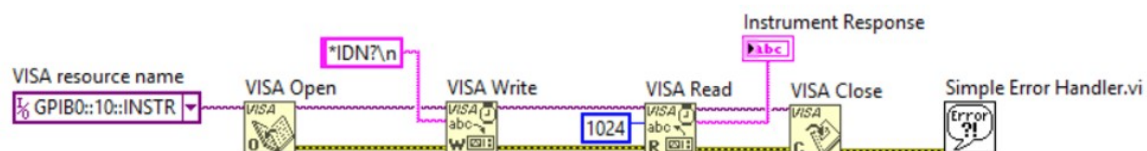
Jedes Messgerät wird innerhalb des Systems (LabView) als **VISA-Ressource** bezeichnet und über den VISA-Ressourcennamen adressiert (INSTR, SOCKET, RAW usw.).

Ein Kommunikationspfad zwischen LabView und dem Messgerät wird als **VISA-Session** bezeichnet.

3.2 Vorbereitung II: Eine typische VISA-Anwendung

Jede VISA-Anwendung läuft wie folgt ab (siehe Abb. 1):

1. Öffnen einer Session für eine bestimmte VISA-Ressource
2. Konfiguration der Kommunikationseinstellungen für die jeweiligen Ressource (Baudrate, Abschlusszeichen)
3. Durchführen von Schreib- und Lesevorgang für das Messgerät
4. Schließen der Sitzung für die Ressource
5. Anzeigen von Fehlern



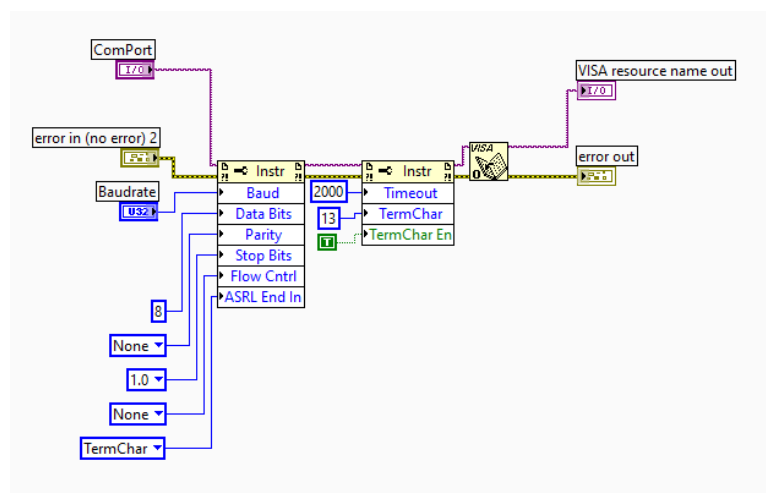
[Abb. 1 – VISA-Kommunikation]

3.3 Kommunikation zu den SMCs

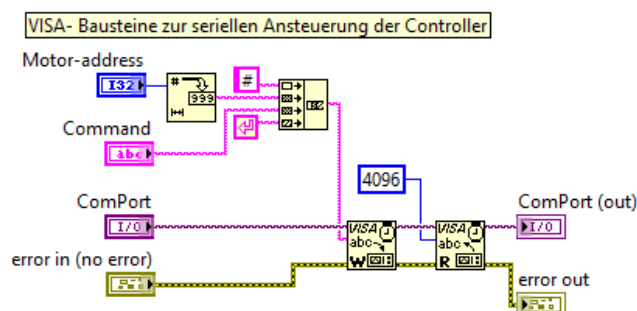
Der in Abschnitt 3.2 vorgestellte Kommunikationsablauf kann somit auf die Schrittmotoren von Nanotec angewandt werden, welche über die Schrittmotor-Controller SMC I-33 angesteuert werden.

Die SMCs sind per USB mit dem Versuchsrechner verbunden. Über das Programm **NI Max** und dem **Windows Gerätemanager** kann die serielle Kommunikation über die USB-Schnittstelle überprüft werden, sowie der **COM-Port** zur Adressierung der SMCs ausgemacht werden.

Die VISA-Bausteine zum Ausführen der Lese-/Schreib-Befehle sind in Abb. 2 (VISA-Open) und Abb. 3 (VISA-Read/Write) dargestellt.



[Abb. 2 – VISA-Open]



[Abb. 3 – VISA-Read/Write]





Die Bausteine wurden so angeordnet, dass über die Eingabe eines *Strings* im Frontpanel der jeweilige Befehl nach Ausführen des LabView-Programms an den SMC gesendet wird. Der SMC steuert daraufhin die Schrittmotoren an.

4. Entwicklung der Funktionsmodule

Im vorherigen Abschnitt wurde die grundlegende Kommunikation zu den Schrittmotoren behandelt, welche mittels VISA-Schnittstelle abläuft. Mit diesen Bausteinen können nun jegliche Befehlsstrings aus dem Programmierhandbuch ausgeführt werden.

Um das Programm übersichtlich zu halten, wurden die Schaltungen VISA-Open und VISA-Read/Write aus 3.3 zusammen gefasst als dedizierte SubVIs.

Die Blockdiagramme Abschnitt 3 werden im folgenden Verlauf der Dokumentation als Funktionsmodule abstrahiert dargestellt. Diese finden sich wieder als:

| | | | |
|---|---|--|---|
|  |  |  |  |
| openSMC1 | rwSMC1 | openSMC2 | rwSMC2 |
| VISA-Open Modul | VISA-Read/Write Modul | VISA-Open Modul | VISA-Read/Write Modul |

[Abb. 1 – verwendete VISA-SubVIs als modulare Bausteine]

Jeder SMC bekommt somit seine eigenen VISA-Module, damit auf beide SMCs gleichzeitig zugegriffen werden kann. Wäre dies nicht gegeben, könnten die SMC nur nacheinander ihre Befehle abarbeiten und nicht *parallel*.

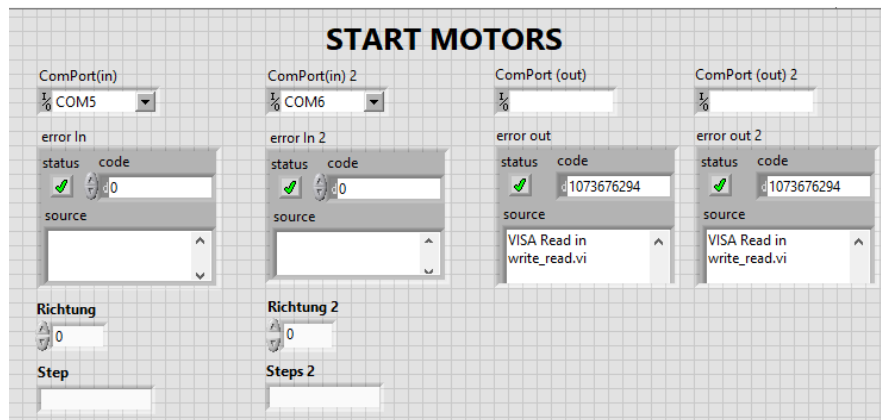
Mit diesen Modulen lassen sich nun weitere Funktionsmodule entwickeln, mit denen das Programm gewisse Grundfunktionen ausführen soll.

Zu diesen Grundfunktionen gehören:

1. Starten der Motoren → Modul Input: Anzahl der auszuführenden Schritte, Drehrichtung der Motoren
2. Stoppen der Motoren → Modul Input: kein Input, da die Motoren einfach nur Stoppen sollen
3. Initialisieren der Motoren → Modul Input: Schrittmodus der Motoren und Motorgeschwindigkeit

4.1 Starten der Motoren

Das grundlegende SubVI zur Ausführung der Motor-Befehle ist **startMotors** (siehe Abb. 2).



[Abb. 2 – Frontpanel von *startMotors*]

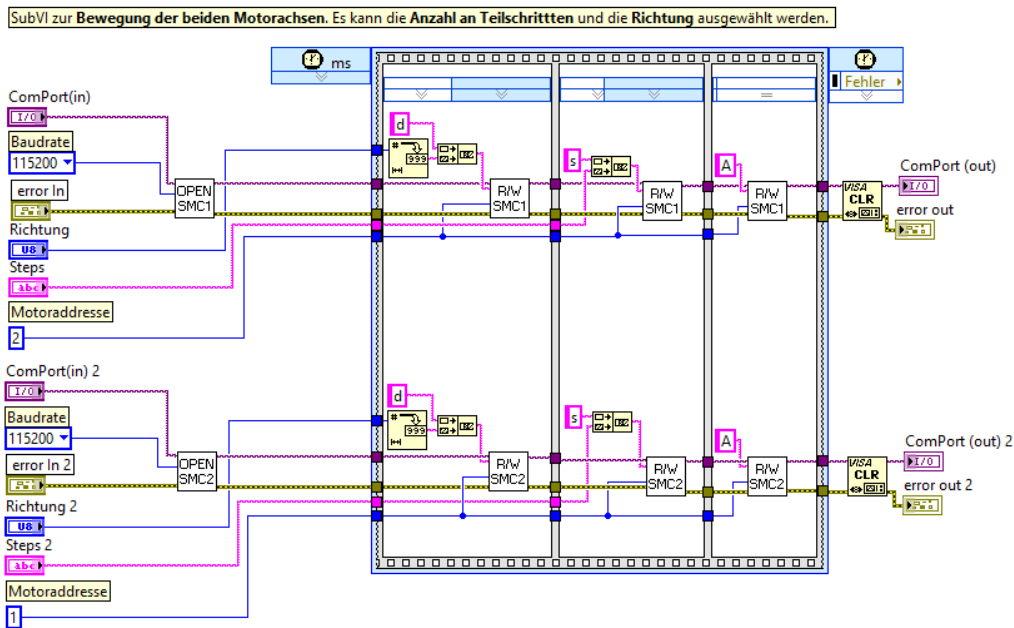
Über das Frontpanel lässt sich folgendes einstellen:

- COM-Ports der beiden Schrittmotoren (hier: COM5 und COM6)
- Drehrichtung **0** und **1** der Motoren
- Anzahl der Steps bzw. die Schrittweite der Motoren

ANMERKUNG:

Die Felder „Error in“, „error out“ und „ComPort (out)“ dienen zur Fehlerbehandlung und sind lediglich Anzeigeelemente.

Ein Blick in das Blockdiagramm von *startMotors* zeigt, wie die VISA-Bausteine aus 3.3 zum Einsatz kommen (siehe Abb. 3).



[Abb. 3 – Blockdiagramm von *startMotors*]

Man erkennt im Blockdiagramm die SubVIs *openSMC1*, *openSMC2*, *rwSMC1* und *rwSMC2*.

Das SubVI *startMotors* macht von einer herkömmlichen VISA Befehlsstruktur Gebrauch in dem über die *openSMC*-Bausteine zuerst die Kommunikationsparameter eingestellt werden.

Diese Parameter bestehen aus:

- COMPort (In)
- Baudrate
- error In

Somit wird die VISA-Session zum SMC gestartet.

Danach werden mit Hilfe einer zeitgesteuerten Sequenzstruktur und den Lese-/Schreib-Bausteinen *rwSMC* die String-Befehle an den Controller gesendet.

Die Befehle lauten:

- **d0** oder **d1** zur Auswahl der Drehrichtung der Motoren
- **s[ANZAHL DER SCHRITTE]** zum Ausführen einer benutzerdefinierten Anzahl an Schritten/Steps → Bsp. „**s200**“ für 200 Schritte
- **A** zum Ausführen der der eingelesenen Befehle

Die zeitgesteuerte Sequenzstruktur dient hierbei dazu, dass die einzelnen Befehle zeitgemäß eingelesen werden. Eine Ausführung der Befehlsstruktur ohne Sequenzstruktur könnte unter Umständen einen Fehler beim Einlesen der Befehle über die VISA-Bausteine erzeugen.

Bei allen Befehlen muss die *Motoradresse* mit angegeben werden, da diese im *rwSMC*-Baustein verwendet wird um die Befehle korrekt zu adressieren.

Abschließend wird die VISA-Session durch den CLR-Baustein beendet und eine Ausgabe eventueller Fehler und der verwendeten COM-Ports erzeugt.

4.1.1 Parallele Befehlsverarbeitung

Aus dem Blockdiagramm erkennt man, dass das Ansteuern der Motoren *parallel* erfolgt. LabView erlaubt dies, solange die verwendeten Bausteine *unabhängig* voneinander verknüpft sind.

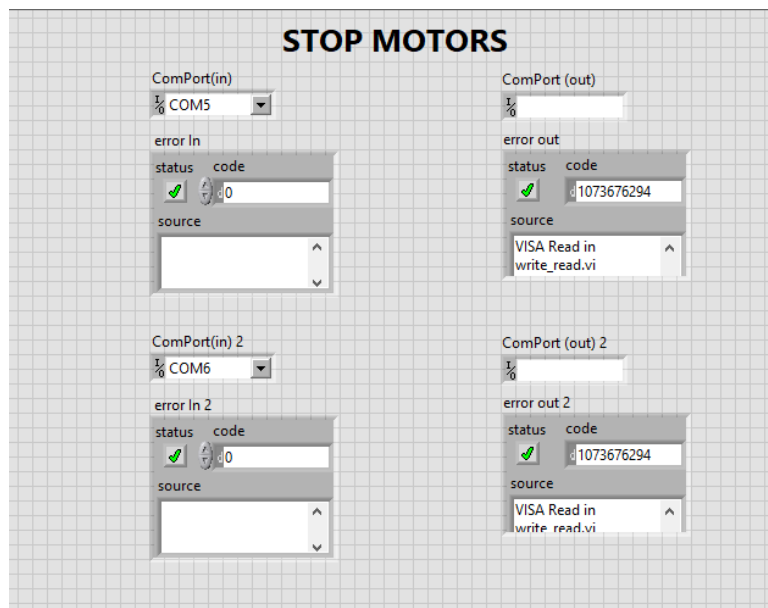
Somit ist es möglich beide Schrittmotoren durch *startMotors* gleichzeitig und mit unterschiedlichen Schrittweiten und Drehrichtungen anzusteuern.

ANMERKUNG:

Die Schrittmotoren können natürlich auch weitere Befehle ausführen wie z.B. die Schrittgeschwindigkeit beschleunigen oder verlangsamen. Für eine ausführlichere Dokumentation dieser Befehle verweise ich daher auf das offizielle Programmierhandbuch des Nanotec SMC I33-2 [\[Link\]](#) . Die dort beschriebenen Befehle können durch das gleiche Prinzip der VISA-Kommunikation (Open Session, Read/Write, Clear Session) an die Motoren geschickt werden.

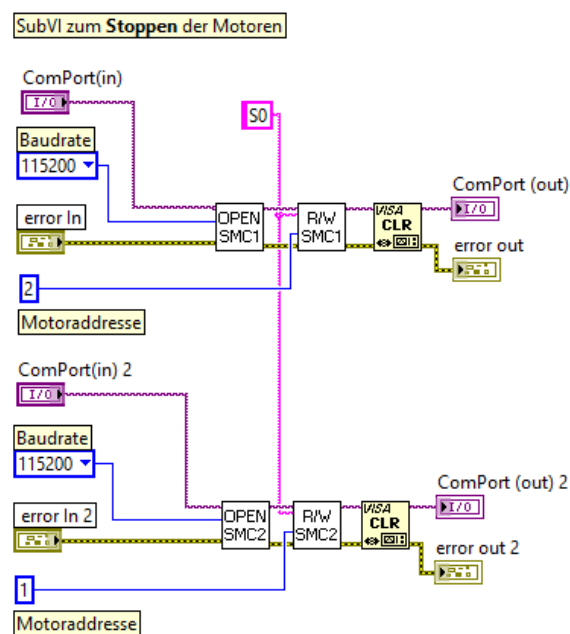
4.2 Stoppen der Motoren

Ähnlich wie bei *startMotors* wird in *stopMotors* der Stopp-Befehl an die Motoren gesendet.



[Abb. 4 – Frontpanel von *stopMotors*]

Da hier lediglich der String „S0“ an beide SMCs gesendet wird, erspart man sich die zeitgesteuerte Sequenzstruktur (siehe Abb. 5).



[Abb. 5 – Blockdiagramm von *stopMotors*]

4.3 Motoren initialisieren

Neben *startMotors* und *stopMotors* gibt es ein weiteres Modul, mit dem String-Befehle über die VISA-Schnittstelle direkt an die Schrittmotoren gesendet werden, das SubVI **initMotors** (siehe Abb. 6).

Es dient zur Initialisierung der Motoren durch das Einstellen folgender Parameter:

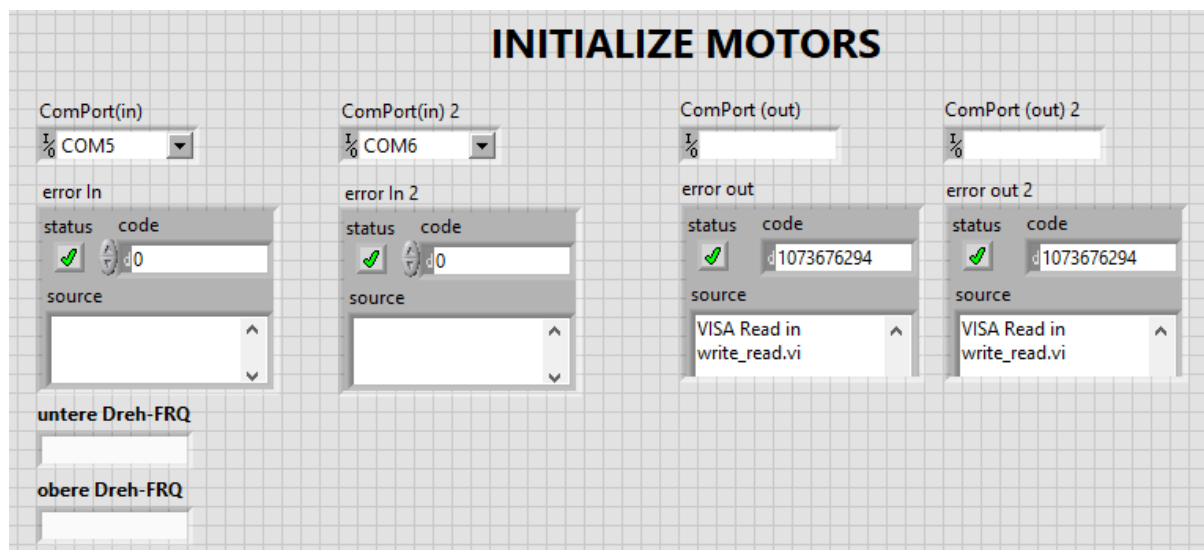
- Drehgeschwindigkeit der Achse → Durch Festlegen und Gleichsetzen der oberen und unteren Drehfrequenz der Motoren:

u[Anzahl der Schritte/Sekunde] (siehe Programmierhandbuch S.53)
o[Anzahl der Schritte/Sekunde] (siehe Programmierhandbuch S.54)

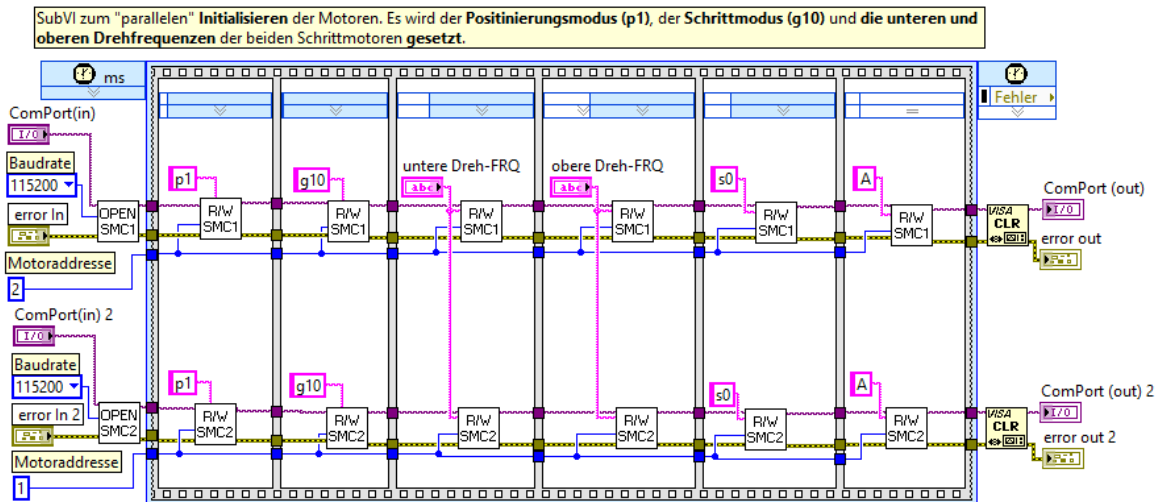
- Positioniermodus: **p1** (siehe Programmierhandbuch S.51)

- Schrittmodus: **g10** (siehe Programmierhandbuch S.20)

Da in diesem SubVI mehrere Befehle an den SMC gesendet werden, wird wieder von einer zeitgesteuerten Sequenzstruktur Gebrauch gemacht (siehe Abb. 7).



[Abb. 6 – Frontpanel von *initMotors*]



[Abb. 7 – Blockdiagramm von *initMotors*]

4.3.1 SubVI benutzerAuswahl

Der String für das Einstellen der unteren und oberen Drehfrequenzen kommt aus dem SubVI *benutzerAuswahl*, welcher die einstellbaren Schrittweiten und für Linear- und Drehmesstisch sowie Motorgeschwindigkeiten aus dem Frontpanel entnimmt. Diese Angaben der Schrittweiten und Drehfrequenzen werden anschließend in die tatsächlichen Befehlsstrings umwandelt. (siehe Abb. 8 und Abb. 9)

BENUTZERAUSWAHL

Motorgeschwindigkeit
 Sehr Langsam -> 0,1 Umdr./s

Schrittweite Lineartisch
 10 µm

Schrittweite Drehmesstisch
 0,1°

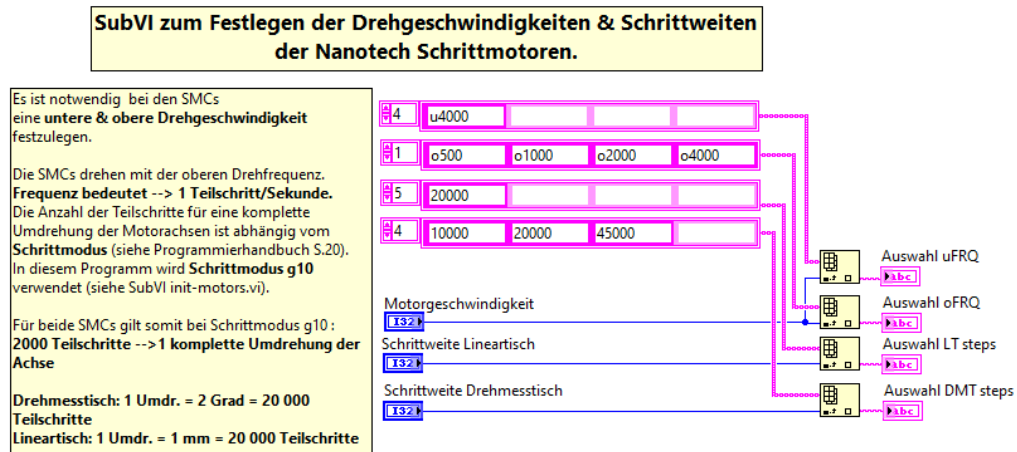
Auswahl uFRQ

Auswahl oFRQ

Auswahl LT steps

Auswahl DMT steps

[Abb. 8 – Frontpanel von *benutzerAuswahl*]



[Abb. 9 – Blockdiagramm von *benutzerAuswahl*]

4.3.2 Schrittmodus g10

Für den weiteren Programmverlauf ist ein besonderer Blick auf den Schrittmodus *g10* nötig. Dieser gibt an, wie viele *Teilschritte* pro Vollschrift von den Motoren ausgeführt werden.

Bisher gilt:

Für eine komplette Umdrehung der Motorachse werden 200 **Vollschritte** benötigt. Das heißt nun, dass im herkömmlichen Schrittmodus (*g1*) der Befehl *s200* die Motorachse eine **komplette Umdrehung** macht.

Bei dem Schrittmodus *g10* werden nun 10 Teilschritte pro Vollschrift ausgeführt. Daraus resultieren nun insgesamt **2000 Teilschritte** für eine komplette Umdrehung. Demnach führt der Motor im Schrittmodus *g10* bei dem Befehl *s2000* eine komplette Umdrehung der Motorachse durch.

Für den Lineartisch gilt: 1 komplette Umdrehung = 1 mm Fahrweg

Für den Drehmesstisch gilt: 1 komplette Umdrehung = 2 Grad Drehung

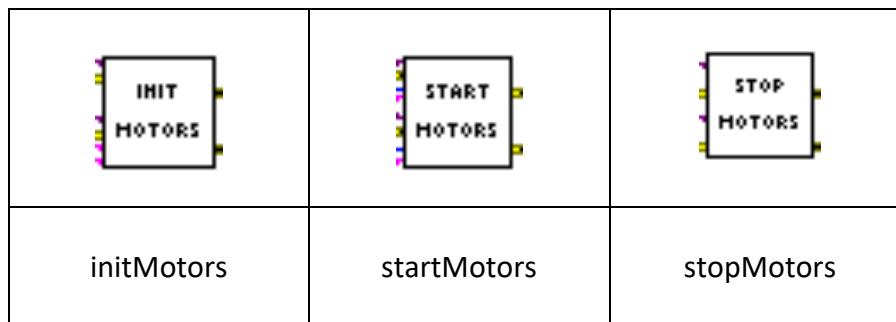
Dieses Wissen ist wichtig für die Berechnung der Fahralgorithmen, welche angeben wie oft eine festgelegte Anzahl an (Teil-)Schritten ausgeführt wird, um die gesamte erlaubte Fläche einer Messprobe mit einem Laser abzufahren.

ANMERKUNG:

Der Kommentar im Blockdiagramm (Abb. 9) beschreibt **20 000** Teilschritte für eine komplette Umdrehung des LTs und DMTs. Diese Aussage ist jedoch falsch, da es sich um **2000** Teilschritte für eine komplette Umdrehung beider Messtische handelt.

4.4 SubVIs als Module

Genau wie die abstrahierten VISA-Module dafür benutzt wurden, um komplexere Funktionen der Motoren zu realisieren (*Initialisierung, Starten und Stoppen*), werden die soeben behandelten 3 Grundfunktionen als Module zusammengefasst. Die Abb. 10 zeigt, diese modularen Bausteine und wie sie im restlichen Programmaufbau vorkommen.

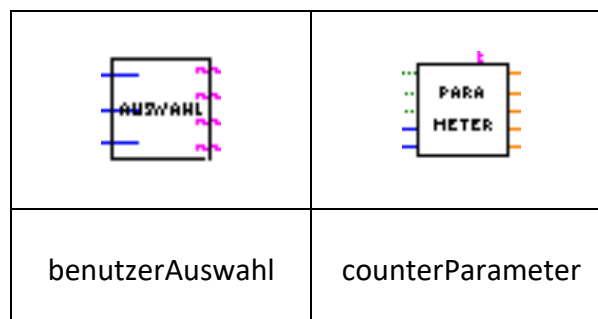


[Abb. 10 – verwendete SubVIs als modulare Bausteine]

Je nach Funktionalität und Teilaufgabe des Programms, werden die Module aus Abb. 10 von benutzerdefinierten Parametern über das Frontpanel bestimmt. Beispielsweise kann im **manuellen Fahrmodus** die gewünschte Geschwindigkeit und Schrittweite beider Motoren eingestellt werden. Dafür wird das SubVI *benutzerAuswahl* als **Input** für *initMotors* und *startMotors* benutzt.

Außerdem gibt es für den/die Benutzer/in die Möglichkeit die Maße der abzufahrenden Messprobe in *mm* über das Frontpanel einzugeben. Daraufhin werden über das Modul *counterParameter* **Zählparameter** von Schleifendurchläufen generiert, womit die komplette Fläche der Probe am effektivsten abgefahren wird. Dieser Vorgang wird in Abschnitt 6 detailliert beschrieben.

Auch diese Funktionen werden als SubVis abstrahiert, dessen Ausgänge mit den jeweiligen Eingängen von *initMotors* und *startMotors* verbunden werden können (siehe Abb. 11).



[Abb. 11 – weitere SubVIs als modulare Bausteine]

5. Architektur von OWIS-Steuerung-v1

In den vorherigen Abschnitten wurden die grundlegenden Befehle zum Starten, Stoppen und Initialisieren der Schrittmotoren behandelt. Auf diese drei SubVIs/Module wird im weiteren Programmaufbau wiederholt zugegriffen.

Da wir im Abschnitt 2.1 die gewünschten Funktionalitäten aufgelistet haben, liegt es nahe diese Anforderungen als Unteraufgaben des Programms zu beschreiben. Zu Grunde liegt nämlich folgendes:

Das Programm führt je nach Eingabe des Benutzers eine bestimmte Aufgabe aus.

Diese Aufgaben werden nun als aufeinander folgende *Zustände/States* definiert. In diesem Programm wurde die einfache Architektur einer *State Machine* (dt. Zustandsmaschine) um die *Queued State Machine* erweitert.

Dabei handelt es sich um eine Art Zustandsmaschine, welche es ermöglicht einen *weiteren Zustand* in eine **Befehlswarteschlange** (Queue) zu setzen.

Damit wird zuerst der aktuelle State ausgeführt und daraufhin der nächste State in der Warteschlange. Somit werden komplexere Aufgaben mit simplen Befehlsmodulen aufgebaut.

Außerdem ermöglicht die QSM-Architektur das *Löschen* der Befehlswarteschlange sowie das *Priorisieren* bestimmter Zustände. Man stellt damit sicher, dass *manche* Befehle immer an den **Anfang** der Befehlswarteschlange gelangen.

Für weitere Infos zu der QSM-Architektur verweise ich auf:

https://learn-cf.ni.com/teach/riodevguide/code/rt_queued-state-machine.html

Die im Programm verwendete QSM-Architektur basiert auf einer QSM, welche es ermöglicht Zustände durch eine **Eingabe des Benutzers über das Frontpanel** hervorzurufen:

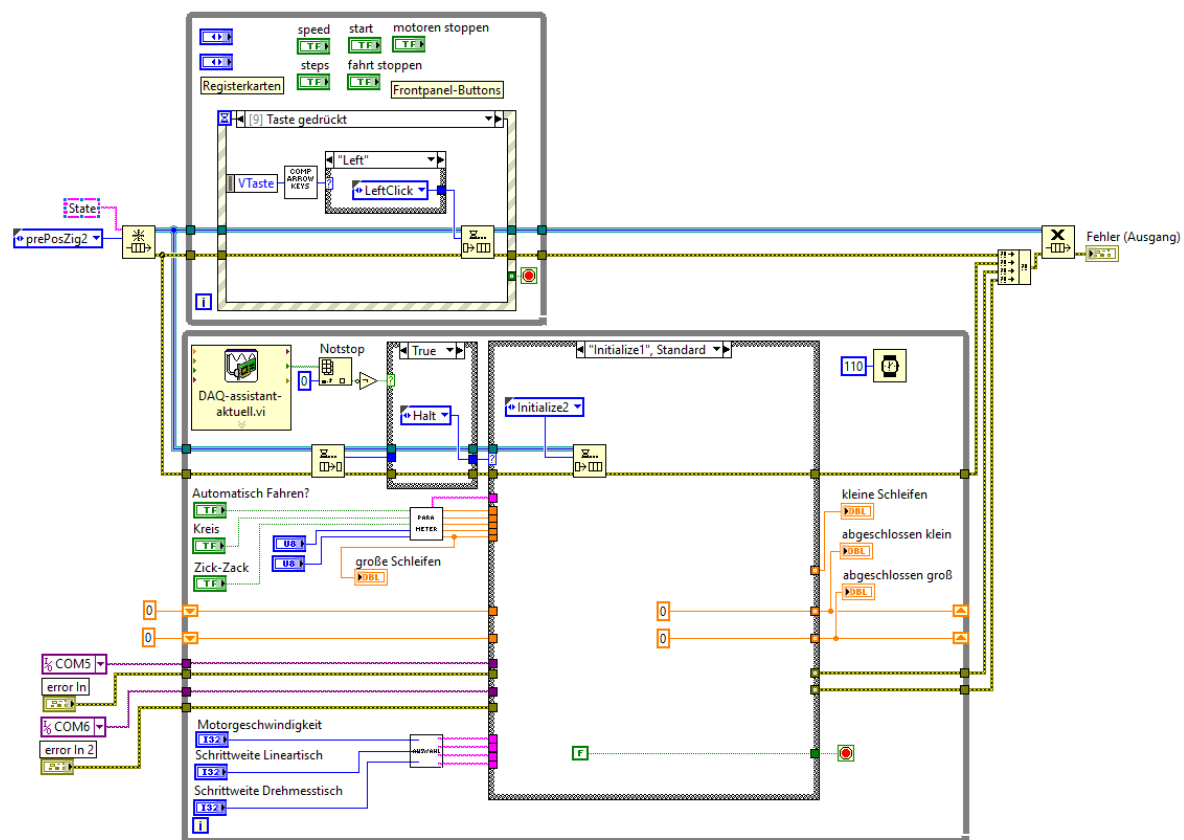
<https://forums.ni.com/t5/Example-Code/Queued-State-Machine-with-User-Input/ta-p/3497450?profile.language=en>

5.1 Aufbau der *Queued State Machine*

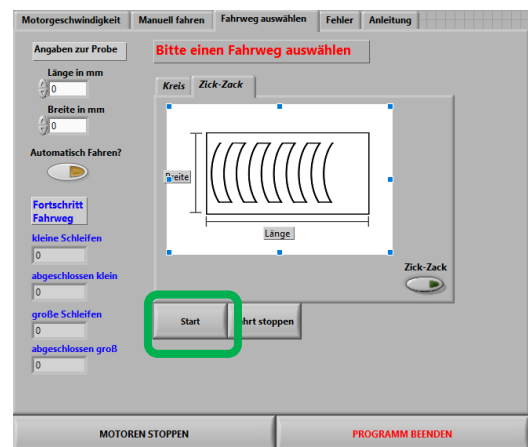
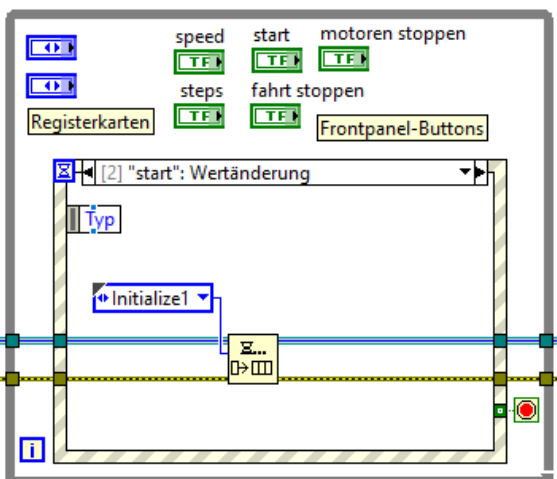
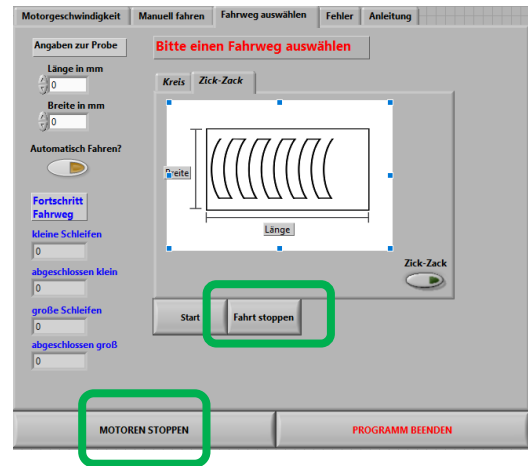
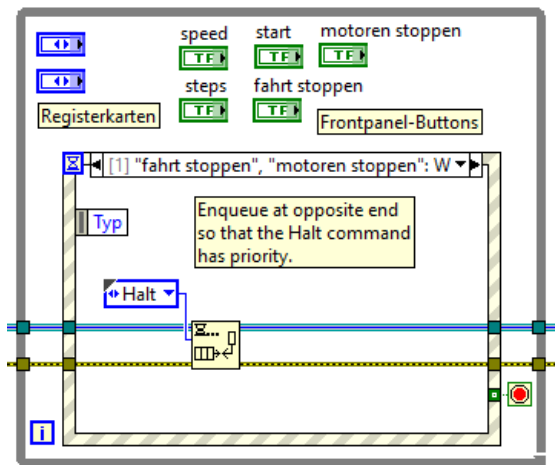
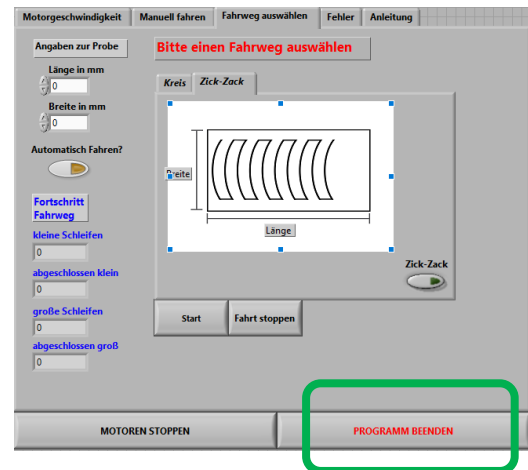
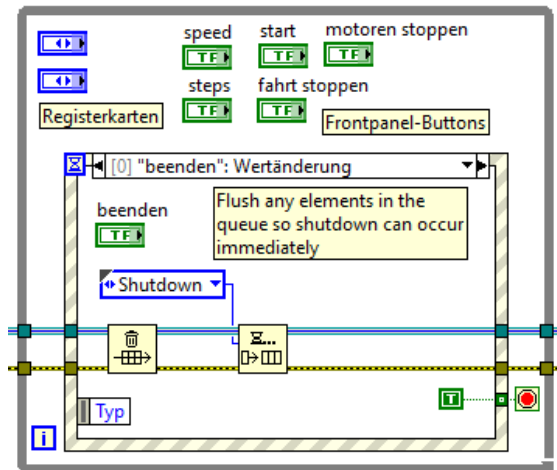
Die QSM in Abb. 1 wird durch While-Schleifen unterteilt in zwei Bereiche.

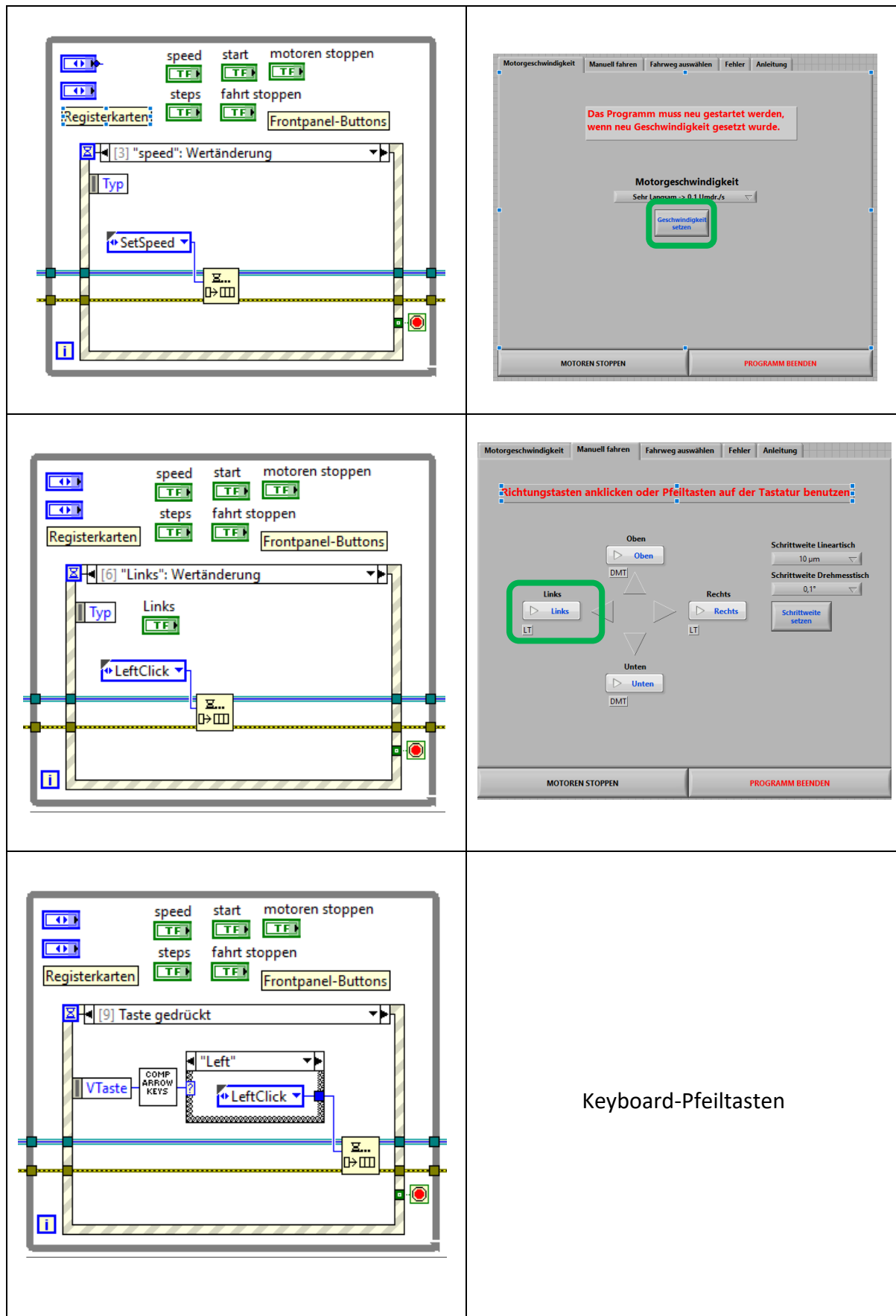
In der oberen While-Schleife sind die **Registerkarten** des Frontpanels definiert mit ihren dazugehörigen **Buttons**, welche einen Prozess in die *Queue* setzen können (siehe Abb. 2).

Die untere While-Schleife definiert daraufhin konkret, aus welchen Modulen und Befehlsstrukturen diese Prozesse bestehen. Abschnitt 6 behandelt alle verwendeten *States* im Detail.



[Abb. 1 – Blockdiagramm von *OWIS-Steuerung-v1*]





[Abb. 2 – Zusammenfassung der Frontpanel-Buttons]

5.2 Festlegen der States

Bevor die QSM entwickelt wurde bedarf es einer konkreten Festlegung der einzelnen Funktionen des Programms, welche vom Benutzer über das Frontpanel ausgeführt werden sollen.

In der nachfolgenden Tabelle sind diese Funktionen mit den dazugehörigen States, wie sie im LabView-Programm benannt wurden, in alphabetischer Reihenfolge aufgelistet.

| Nr. | Funktion | State | Verwendete Module | |
|-----|--------------------------------------|--|--|--|
| | | | Hauptmodule | Untermodule |
| 1 | Leerlauf | Idle | | |
| 2 | Initialisierung | Initialize 1 Initialize 2 Initialize 3 Initialize 4 Initialize 5 | benutzerAuswahl (als Eingang) | |
| | | | initMotors: | openSMC1 openSMC2 rwSMC1 rwSMC2 |
| 3 | Pre-Positionierung der Messtische | prePosCirc prePosZig prePosZig2 | startMotors: | openSMC1 openSMC2 rwSMC1 rwSMC2 |
| 4 | Einstellung der Motorgeschwindigkeit | SetSpeed | benutzerAuswahl (als Eingang) | |
| | | | initMotors: | openSMC1 openSMC2 rwSMC1 rwSMC2 |
| 5 | Manuelles Steuern der Motoren | UpClick DownClick LeftClick RightClick | compareKey benutzerAuswahl (als Eingang) | |
| | | | startMotors: | openSMC1 openSMC2 rwSMC1 rwSMC2 |
| 6 | Fahrweg Zick-Zack | pathZig1 pathZig2 pathZig3 pathZig4 | counterParameter | |
| | | | startMotors: | openSMC1 openSMC2 rwSMC1 |

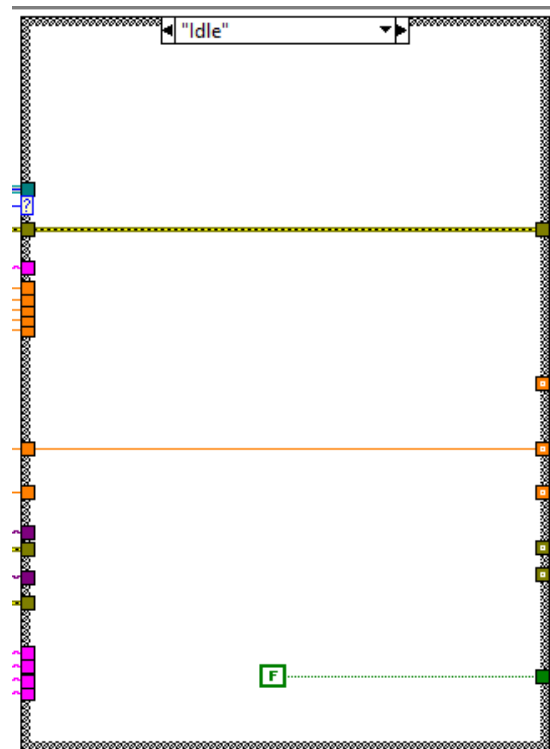
| | | | | |
|----|-----------------------|--|------------------|--|
| | | | | rwSMC2 |
| 7 | Fahrweg Kreis | pathCirc1 pathCirc2 pathCirc3 pathCirc4 | counterParameter | |
| | | | startMotors: | openSMC1 openSMC2 rwSMC1 rwSMC2 |
| 8 | Stoppen der Motoren | Halt | stopMotors: | openSMC1 openSMC2 rwSMC1 rwSMC2 |
| 9 | Statusanzeige | Report | | |
| 10 | Beenden des Programms | Shutdown | | |

[Abb. 3 – Zusammenfassung der States]

5.3 Beschreibung der States

5.3.1 Leerlauf

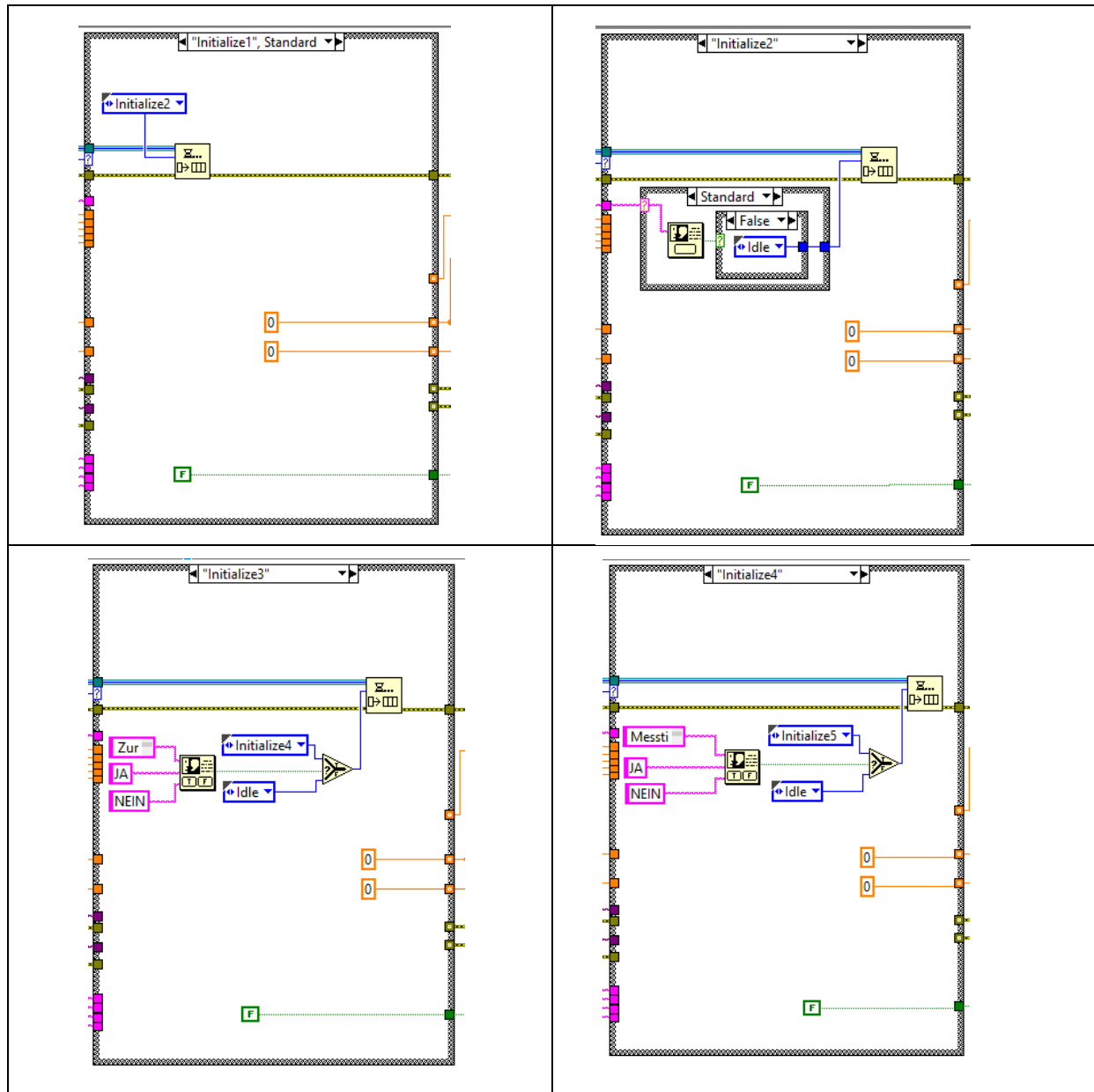
Der *Idle* Zustand ist Leerlauf-Zustand indem **keine** Aktion ausgeführt wird. Auf *Idle* wird verwiesen, wenn beispielsweise der/die Benutzer/in während der Initialisierungsroutine (siehe Abschnitt 5.3.2), die Aufforderungen mit „NEIN“ beantwortet. Dies lässt die anschließend Initialisierungsroutine wieder von neu beginnen.

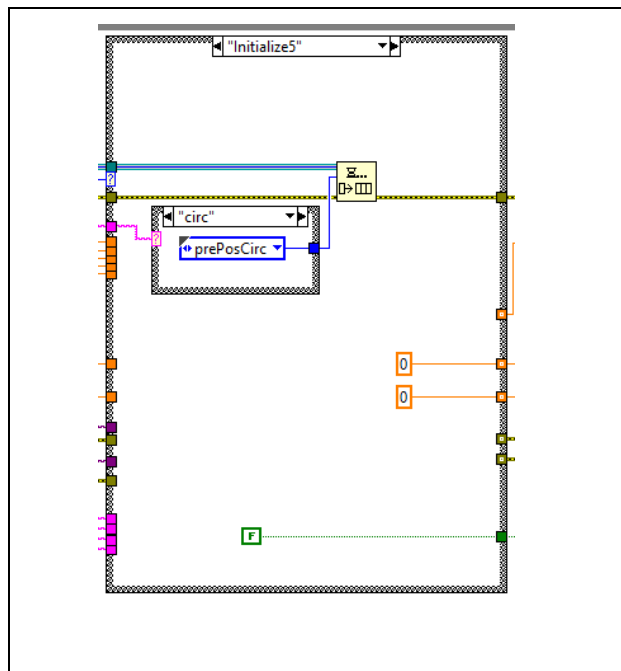


[Abb. 4 – *Idle*-State]

5.3.2 Initialisierung

Die Initialisierungsroutine ist in fünf Unterschritte aufgeteilt. Nacheinander werden die Schleifenzähler auf 0 gesetzt und der/die Benutzer/in gefragt, ob die notwendigen Vorbereitungsschritte für eine Fahrweg-Routine vorgenommen wurden.

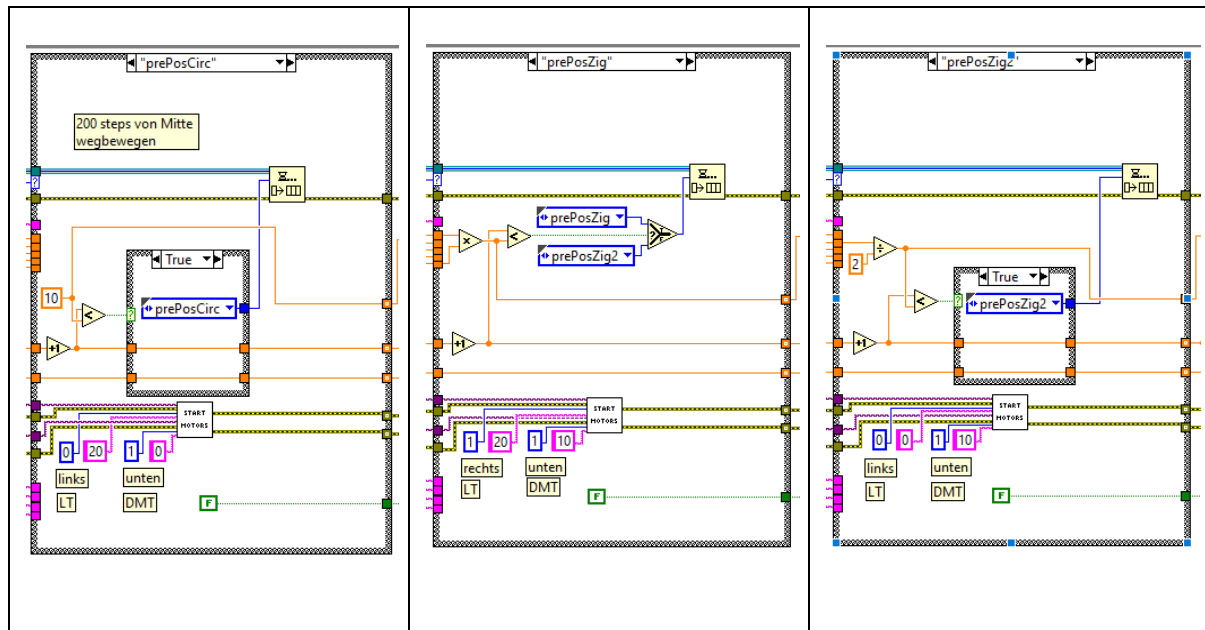




[Abb. 5 – *Initialize-States*]

5.3.3 Pre-Positionierung der Messtische

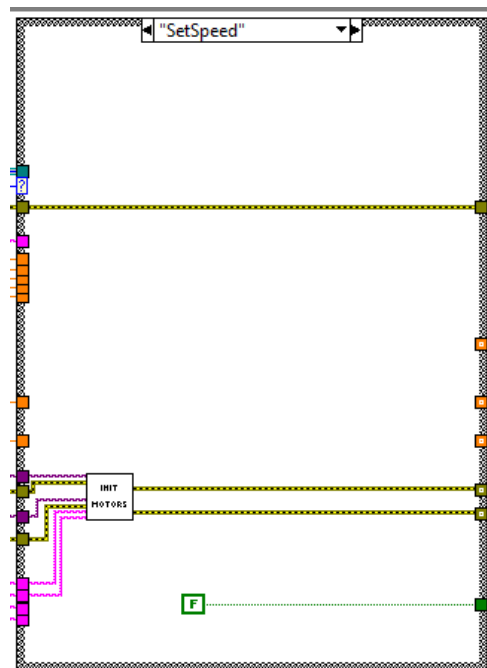
Je nach ausgewähltem Fahrweg müssen die Messtische vorher entsprechend positioniert werden, um mit einem Fahrweg zu beginnen. Die bereits vorher berechneten Schleifendurchläufe kommen aus dem Modul *counterParameter* und hängen von der Größe der Messprobe ab. Abschnitt 6 behandelt diese Berechnung der Schleifendurchläufe im Detail.



[Abb. 6 – prePosCirc/Zig-States]

5.3.4 Einstellung der Motorgeschwindigkeit

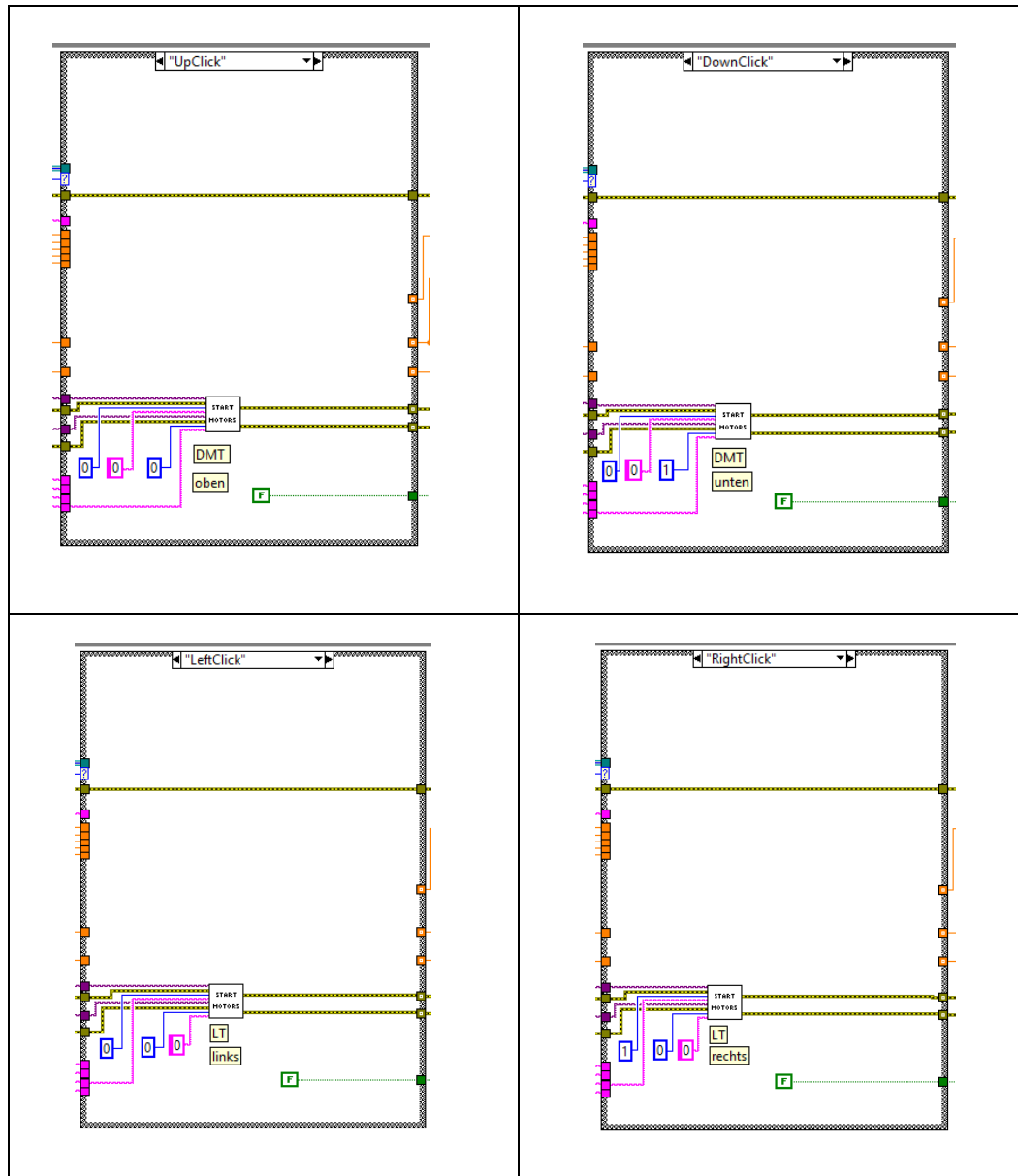
SetSpeed stellt die Motorgeschwindigkeit ein, welche der/die Benutzer/in im Frontpanel ausgewählt hat. Die Eingänge für *initMotors* kommen dementsprechend aus *benutzerAuswahl*.



[Abb. 7 – *SetSpeed*-State]

5.3.5 Manuelles Steuern der Motoren

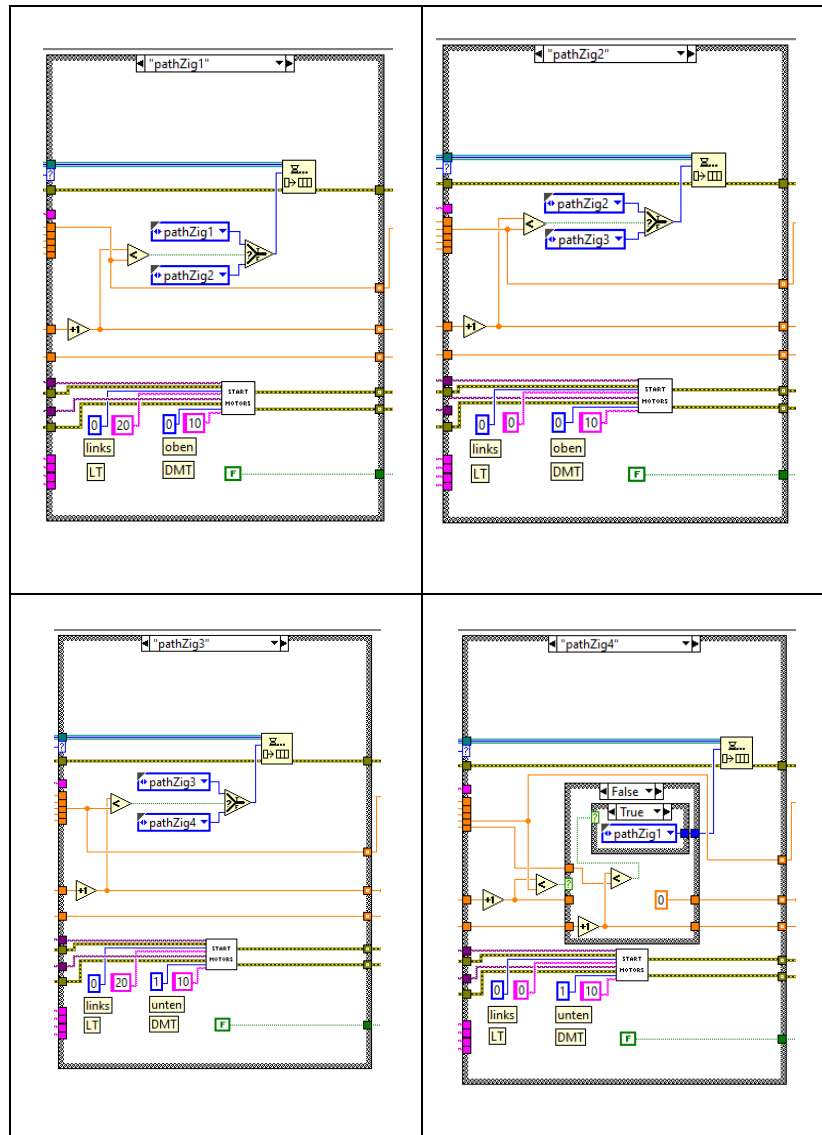
Beim manuellen Steuern der Motoren kann der Messturm in die 4 möglichen Richtungen bewegt werden. Für alle 4 Fälle wird jeweils *startMotors* benutzt. Dabei kommt die genaue Anzahl der zu fahrenden Schritte aus *benutzerAuswahl*, welches durch das Frontpanel bedient wird.



[Abb. 8 – Click-States]

5.3.6 Fahrweg Zick-Zack

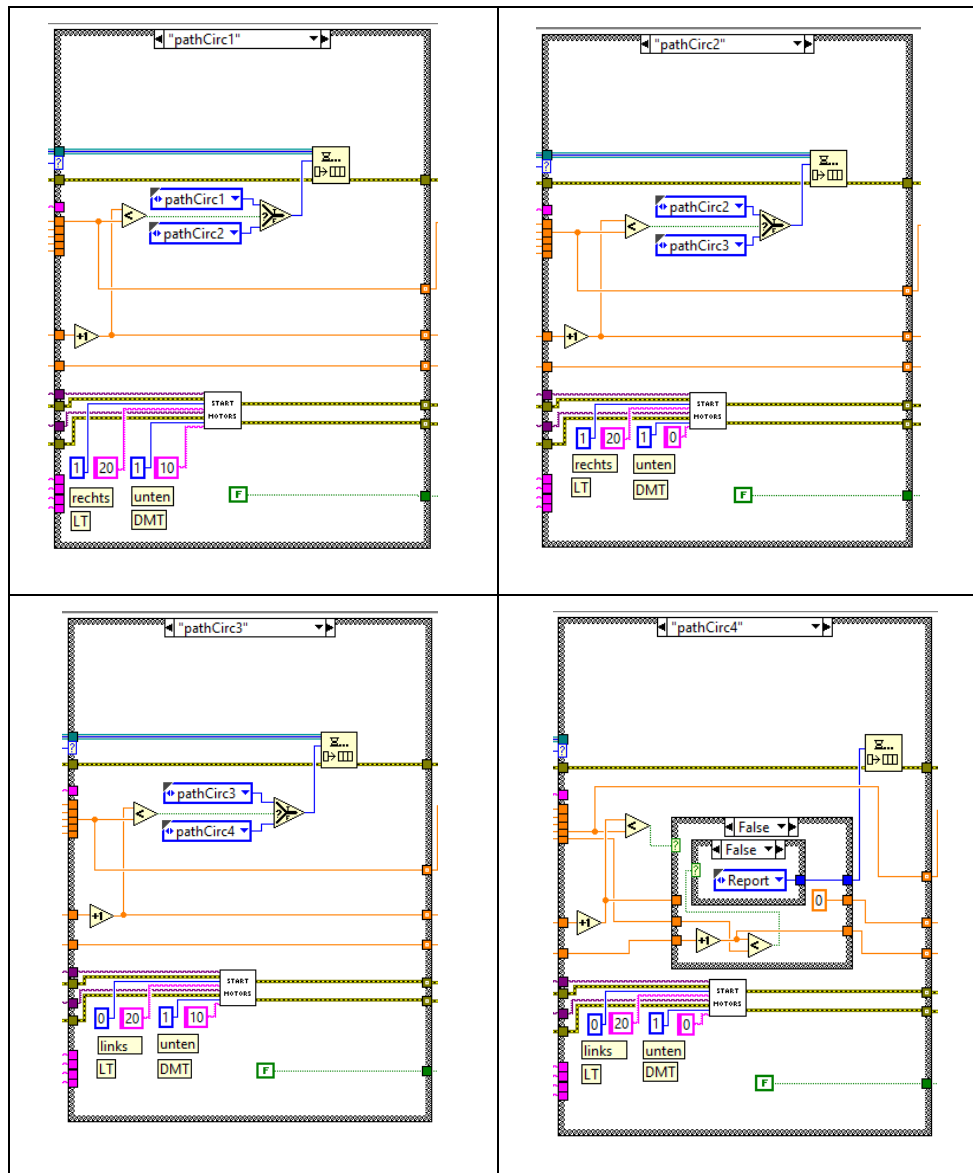
Bei dem Fahrweg Zick-Zack wird eine in *counterParameter* berechnete Anzahl an Schleifendurchläufen ausgeführt. Dabei werden in jeder Schleife entweder 20 Schritte bei dem Lineartisch oder 10 Schritte bei dem Drehmesstisch ausgeführt. Abschnitt 6 behandelt diese Berechnung der Schleifendurchläufe im Detail.



[Abb. 9 – pathZig-States]

5.3.7 Fahrweg Kreis

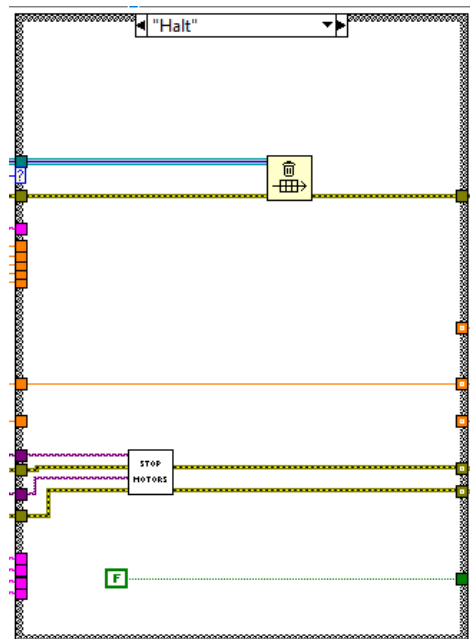
Bei dem Fahrweg Kreis wird eine in *counterParameter* berechnete Anzahl an Schleifendurchläufen ausgeführt. Dabei werden in jeder Schleife entweder 20 Schritte bei dem Lineartisch oder 10 Schritte bei dem Drehmesstisch ausgeführt. Abschnitt 6 behandelt diese Berechnung der Schleifendurchläufe im Detail.



[Abb. 10 – *pathCirc*-States]

5.3.8 Stoppen der Motoren

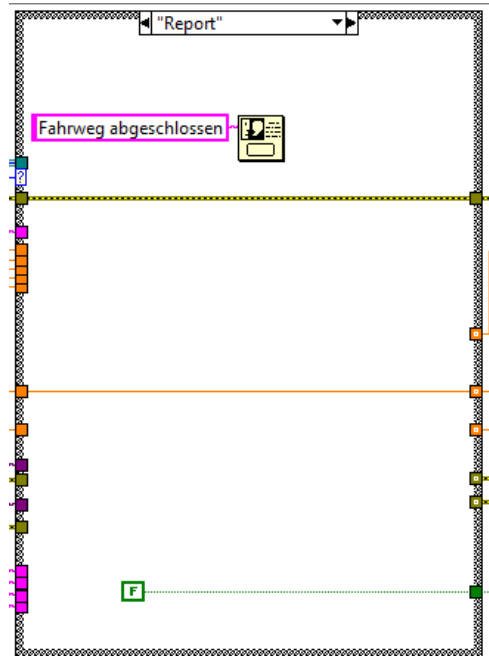
Beim Ausführen des *Halt*-States, werden beide Motoren mit sofortiger Wirkung gestoppt.



[Abb. 11 – *Halt*-State]

5.3.9 Statusanzeige

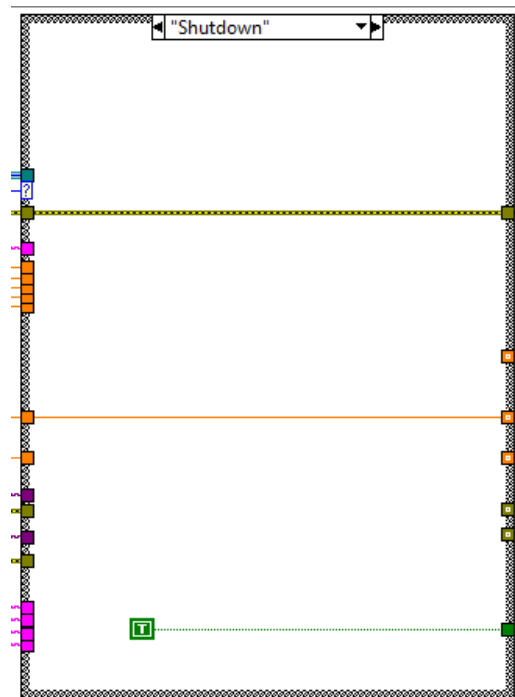
Der *Report*-State gibt dem/der Benutzer/in an, wann ein kompletter Fahrweg abgeschlossen wurde.



[Abb. 12 – *Report*-State]

5.3.10 Beenden des Programms

Beim Ausführen des *Shutdown*-States wird die untere While-Schleife der QSM unterbrochen und das Programm mit sofortiger Wirkung beendet.



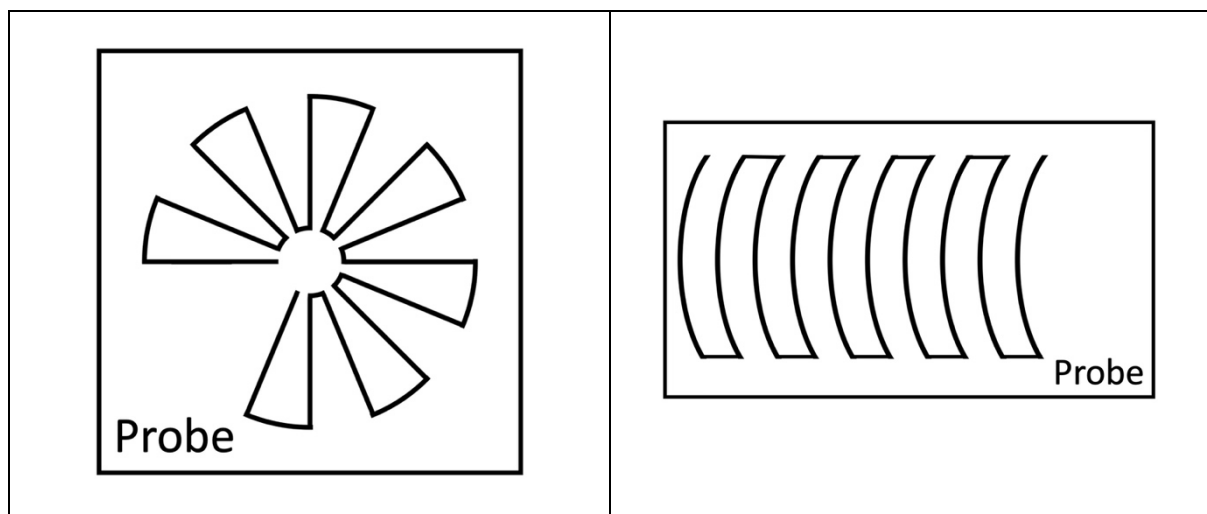
[Abb. 13 – *Shutdown-State*]

6. Fahrwegalgorithmen

Die zur Verfügung stehenden Fahrwegalgorithmen sind abhängig von den Maßen der Messproben. Je nach **Form** der Messprobe stehen entweder ein Kreis- oder ein Zick-Zack-Fahrweg zur Verfügung. Dabei wird der/die Benutzer/in nach der Eingabe und Wahl des Fahrwegs darauf hingewiesen, dass für eine *quadratische* Messprobe der Kreisfahrweg und für jede weitere *rechteckige* Messprobe ein Zick-Zack-Fahrweg am effektivsten ist. Effektiv bedeutet bei diesem Messaufbau, dass die **größte Fläche** der Messprobe mit dem Laser abgefahren wird.

Die Abb. 1 zeigt schematisch, wie diese Fahrwege mit dem Laser jeweils auf der Messprobe aussehen. Bei dieser Bewegung bleibt der Laser statisch fixiert, während sich die Messprobe bzw. die Messtische verschieben.

In den folgenden Abschnitten werden die Umsetzung dieser Fortbewegungsalgorithmen erläutert.

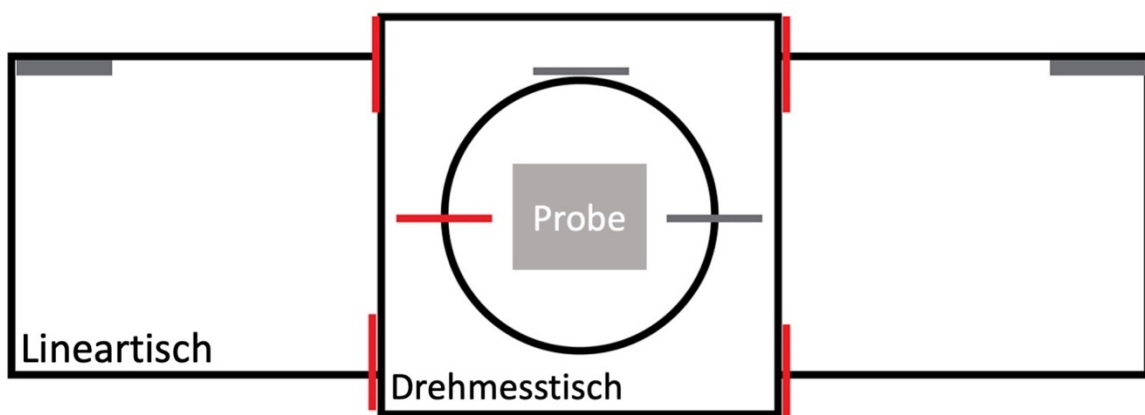


[Abb. 1 – Fahrwegmuster]

6.1 Ausgangssituation vor Beginn der Routinen

Unabhängig vom Fahrweg gelten folgende Einstellungen, welche vor dem Start vorgenommen werden:

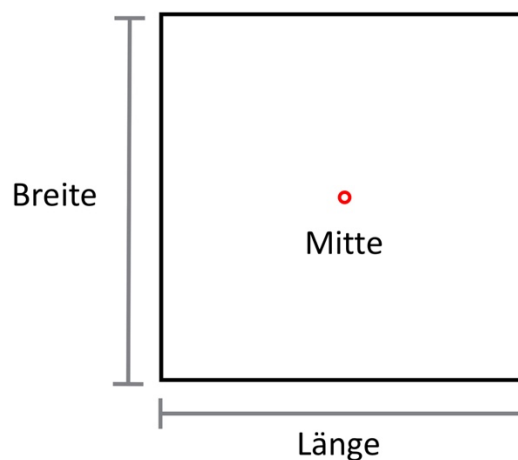
1. Drehmesstisch und Lineartisch werden so platziert, dass ein komplettes Abfahren der Messprobe innerhalb des Puffers möglich ist. Dies sichert, dass beide Messtische nicht während des Fahrwegs die Notstopp-Taster betätigen.
2. Die **Messprobe** wird in die **Mitte des Messturms** bzw. Drehmesstischs platziert.
3. Der **Laser** wird auf die **Mitte der Probe** gerichtet.



[Abb. 2 – Messaufbau]

6.2 Fahrweg Kreis

Bevor der Kreisfahrweg startet, wird der Laser auf die Mitte der quadratischen Probe gerichtet (siehe Abb. 3).

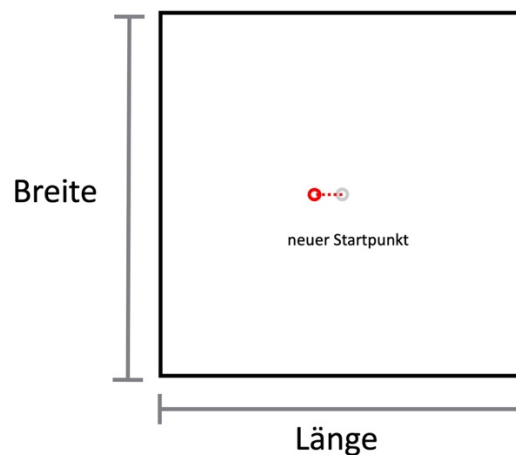


[Abb. 3 – Laser in der Mitte der Probe]

6.2.1 Pre-Positionierung der Messtische

Damit man einen Algorithmus erhält, welcher eine bereits abgefahrene Stelle der Messprobe nicht mehrmals abfährt, muss der Startpunkt des Fahrwegs etwas versetzt erfolgen.

In der QSM wird dementsprechend vor jedem Start des Fahrwegs eine Pre-Positionierungsroutine gestartet, welche den Laser um 200 Schritte von der Mitte versetzt.



[Abb. 4 – neuer Startpunkt]

6.2.2 Kreis Algorithmus

Bevor der Algorithmus in LabView entwickelt wurde, musste klar definiert werden, wie der Messtisch bzw. der Laser sich entlang der Fläche der Messprobe bewegen soll. Dieses Schema ist in Abb 5. und Abb. 6 zu sehen.

Der Algorithmus ist in fünf Teile aufgeteilt, wobei Teil 5 lediglich die Anzahl an Wiederholungen von Teil 1 bis Teil 4 ist.

Teil 1 – 4 spiegeln eine Bewegung des DMTs um 1° wider. Demnach werden für ein **komplettes** Abfahren der Fläche Teil 1 – 4 insgesamt 360 mal wiederholt ($360 \cdot 1^\circ$). Da der DMT jedoch keine komplette 360° Umdrehung zulässt, wurde mit einem Puffer von 30° gerechnet. Teile 1 – 4 werden also insgesamt 330 mal wiederholt, unabhängig von den Maßen der Messprobe.

Die folgenden Erklärungen der Abläufe werden mit der Bewegung des Lasers entlang der Messprobe erläutert. In der Realität bewegen sich die beiden Messtische in die entgegengesetzte Richtung. Die genauen Abläufe und Zählerparameter werden dafür in Abschnitt 6.2.3 erklärt.

Teil 1: Der Laser bewegt sich nach links bis zum Ende des sog. Puffer-Radius.

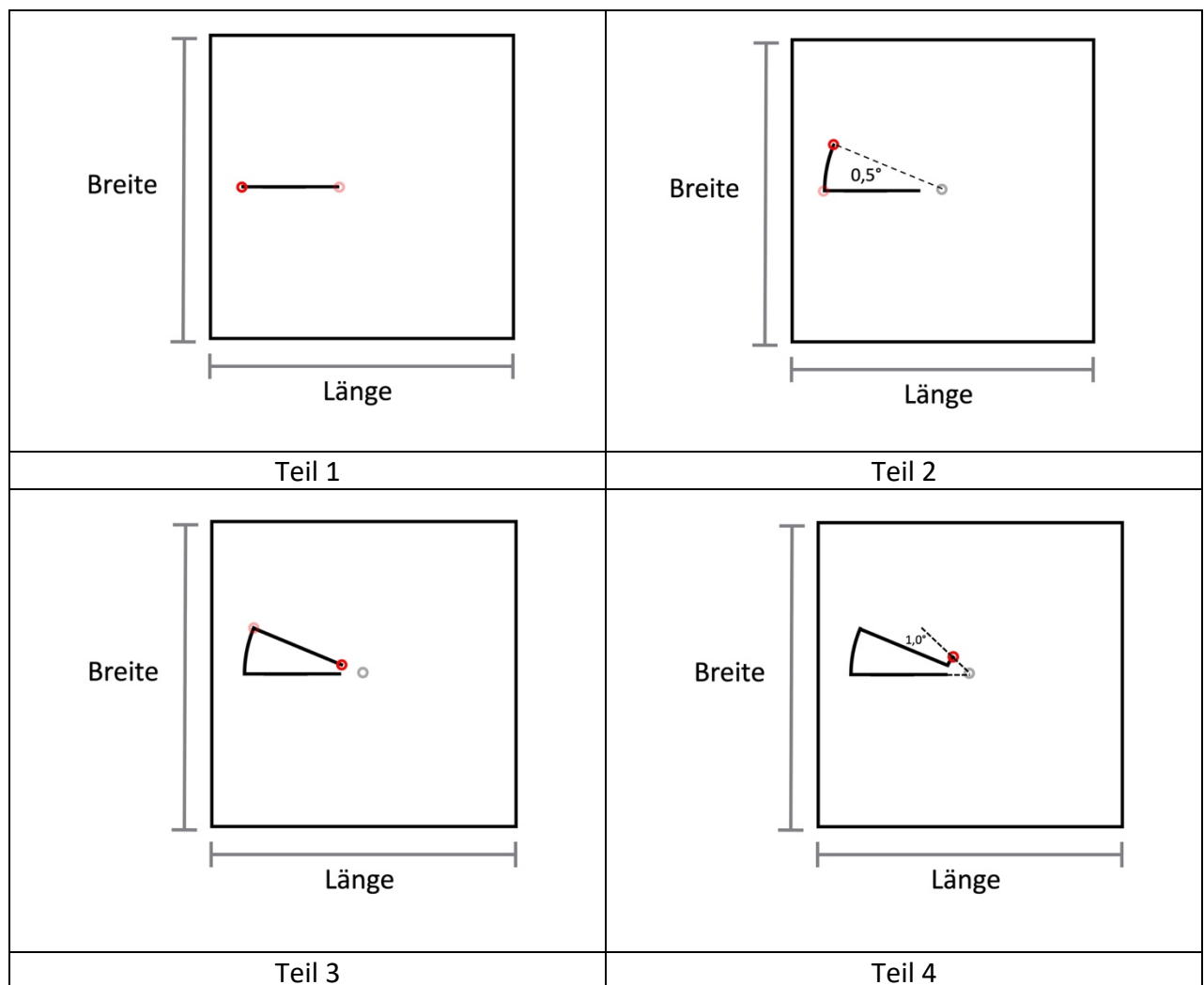
Teil 2: Der Laser bewegt sich um $0,5^\circ$ nach oben, entlang des äußeren Kreises.

Teil 3: Der Laser bewegt sich um die gleiche Länge aus Teil 1, dem Puffer-Radius, nach rechts.

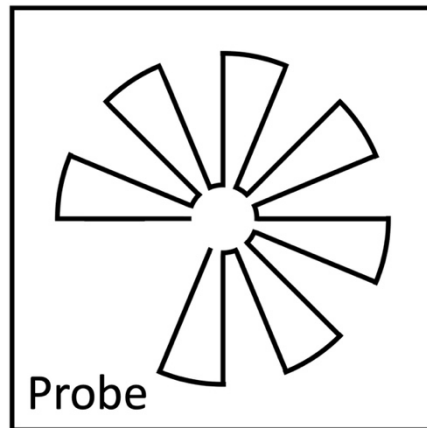
Teil 4: Der Laser bewegt sich um weitere $0,5^\circ$ nach oben, entlang des inneren Kreises.

Teil 5: Die Vorgänge aus Teil 1 – 4 werden wiederholt.

Da im realen Fahrweg die Vorgänge aus Teil 1 und Teil 2 teilweise parallel ablaufen, werden die resultierenden Fortbewegungsmuster etwas spitzer zu laufen als die dargestellten Zeichnungen.



[Abb. 5 – Teil 1 - 4]



[Abb. 6 – Teil 5]

6.2.3 Umsetzung in LabView

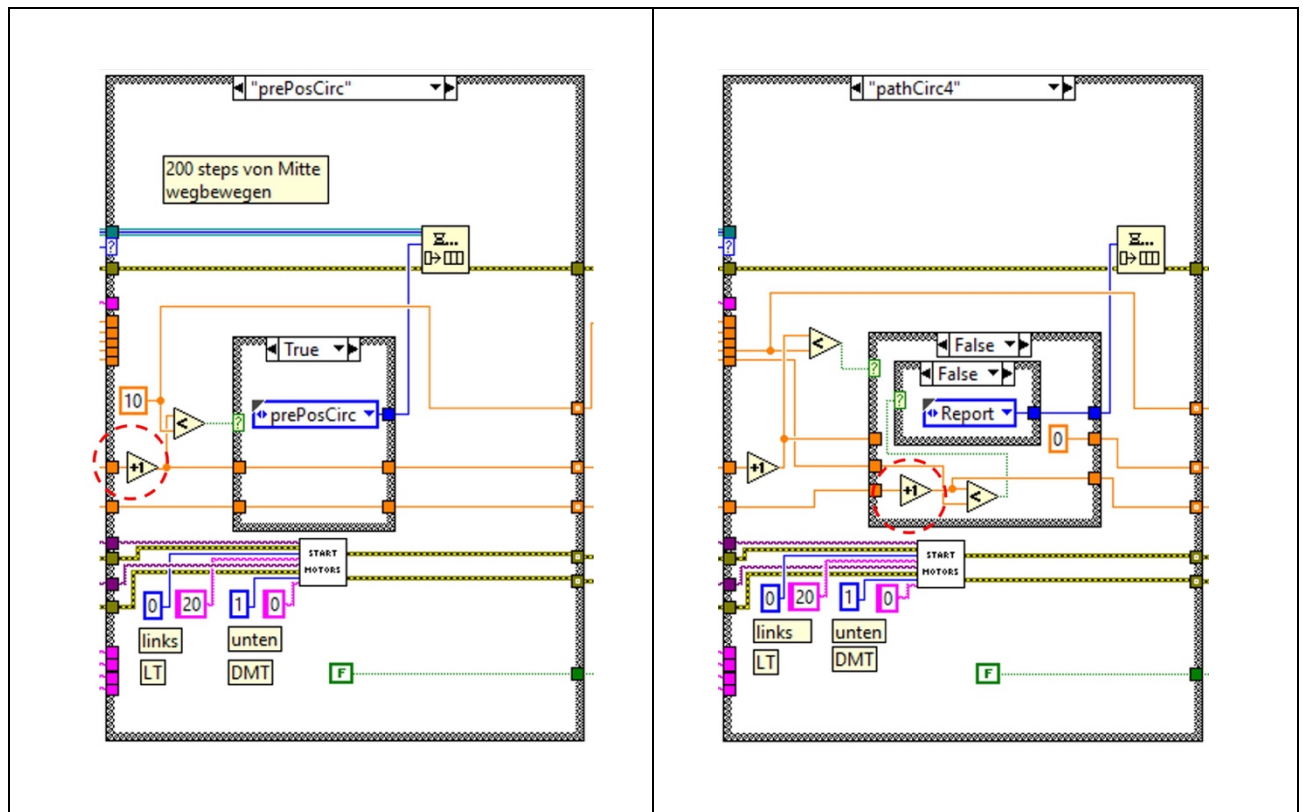
Die Umsetzung des in Abschnitt 6.2.2 erklärten Algorithmus wird in LabView anhand von einer While-Schleife und zwei Schleifenzählern realisiert. Die Abb. 7 zeigt die beiden Inkrement-Bausteine beider Zähler innerhalb der States, welche in einer While-Schleife ausgeführt werden. Diese Zähler werden bei jedem Schleifendurchlauf zusätzlich mit bestimmten Parametern verglichen, welche in *counterParameter* (siehe Abb. 8) berechnet werden.

Für die Umsetzung des Algorithmus werden die Schrittmotoren so initialisiert, dass zur Bewegung des LTs um 1 mm genau 2000 Schritte notwendig sind. Für die Verschiebung des DMTs um $1^\circ \rightarrow 1000$ Schritte des Schrittmotors.

Bei einem Fahrweg werden pro Schleifendurchlauf genau 20 (LT) bzw. 10 (DMT) Schritte ausgeführt.

Die Berechnung der sogenannten Schleifenparameter gibt an, *wie oft* ein Schleifendurchlauf, also eine Bewegung von 20 oder 10 Schritten für eine bestimmte Bewegungsrichtung erfolgen soll.

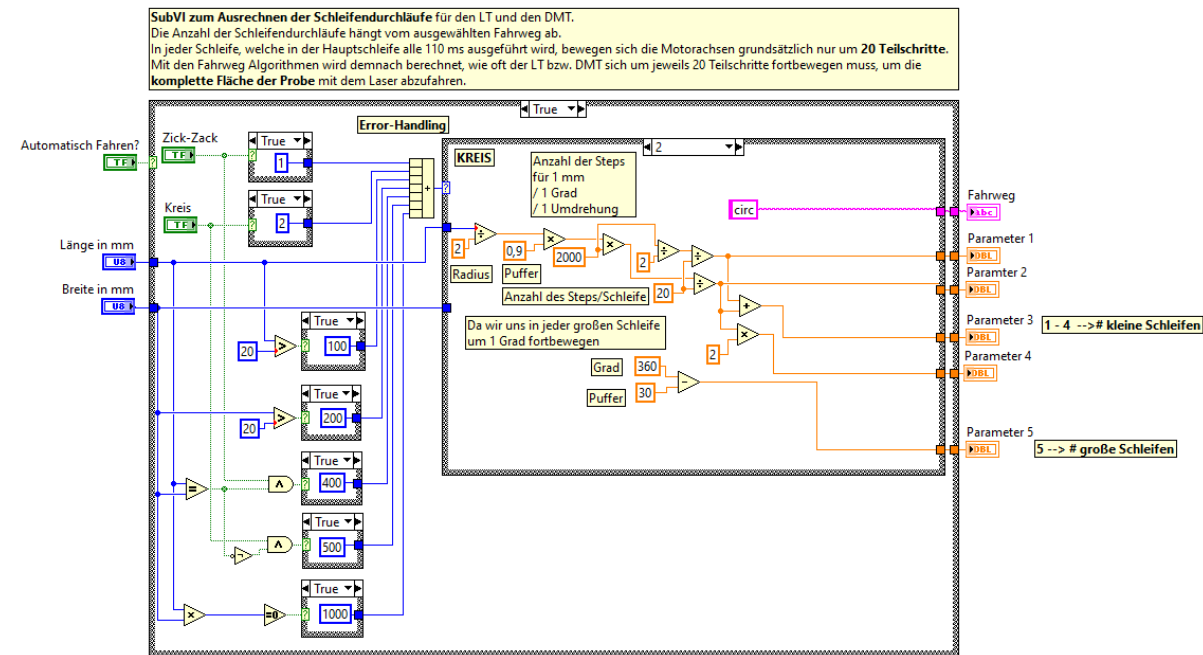
Wie bereits oben erwähnt, werden die Schleifendurchläufe lediglich mit 2 Zählern gezählt (siehe Abb. 7).



[Abb. 7 – Inkrement-Bausteine der Schleifenzähler]

Der Stand des Zähler 1 wird jeweils nachfolgend mit den Parametern 1 bis 4 verglichen. Die Berechnungen der Parameter sind die übersetzten **Bewegungsgrenzen**, welche in Abb. 5 gezeigt wurden. Teile 1 – 5 korrespondieren dementsprechend mit Parameter 1 – 5:

- Parameter 1 → Anzahl der 10-Schritt-Schleifen für eine Bewegung des DMT um $0,5^\circ$ (Teil 2).
- Parameter 2 → Anzahl der 20-Schritt-Schleifen für eine Bewegung des LT um die Länge des Puffer-Radius (Teil 1).
- Parameter 3 → Anzahl der 10-Schritt-Schleifen für eine weitere Bewegung des DMT um $0,5^\circ$ (Teil 3). Hierfür werden Parameter 1 und 2 zusammen addiert.
- Parameter 4 → Anzahl der 20-Schritt-Schleifen für eine Bewegung des LT um die Länge des Puffer-Radius wieder zurück (Teil 4). Hierfür wird Parameter 2 mit 2 multipliziert.
- Parameter 5 → Anzahl der Wiederholungen von Teil 1 – 4. In diesem Fall wird Parameter 5 immer auf 330 gesetzt.



[Abb. 8 – Blockdiagramm von *counterParameter*]

Ein Beispiel Fahrweg könnte somit wie folgt aussehen, wobei lediglich mit nur 2 Zählern der Fortschritt des Algorithmus und somit der Fortschritt des Fahrwegs evaluiert wird:

Zähler 1: von 0 bis 50

- Bewegung vom **DMT** nach **unten** um $0,5^\circ$
- Bewegung vom **LT** nach **rechts**

Zähler 1: von 51 bis 2300

- Fortführung der Bewegung vom **LT** nach **rechts**

Zähler 1: von 2301 bis 2351

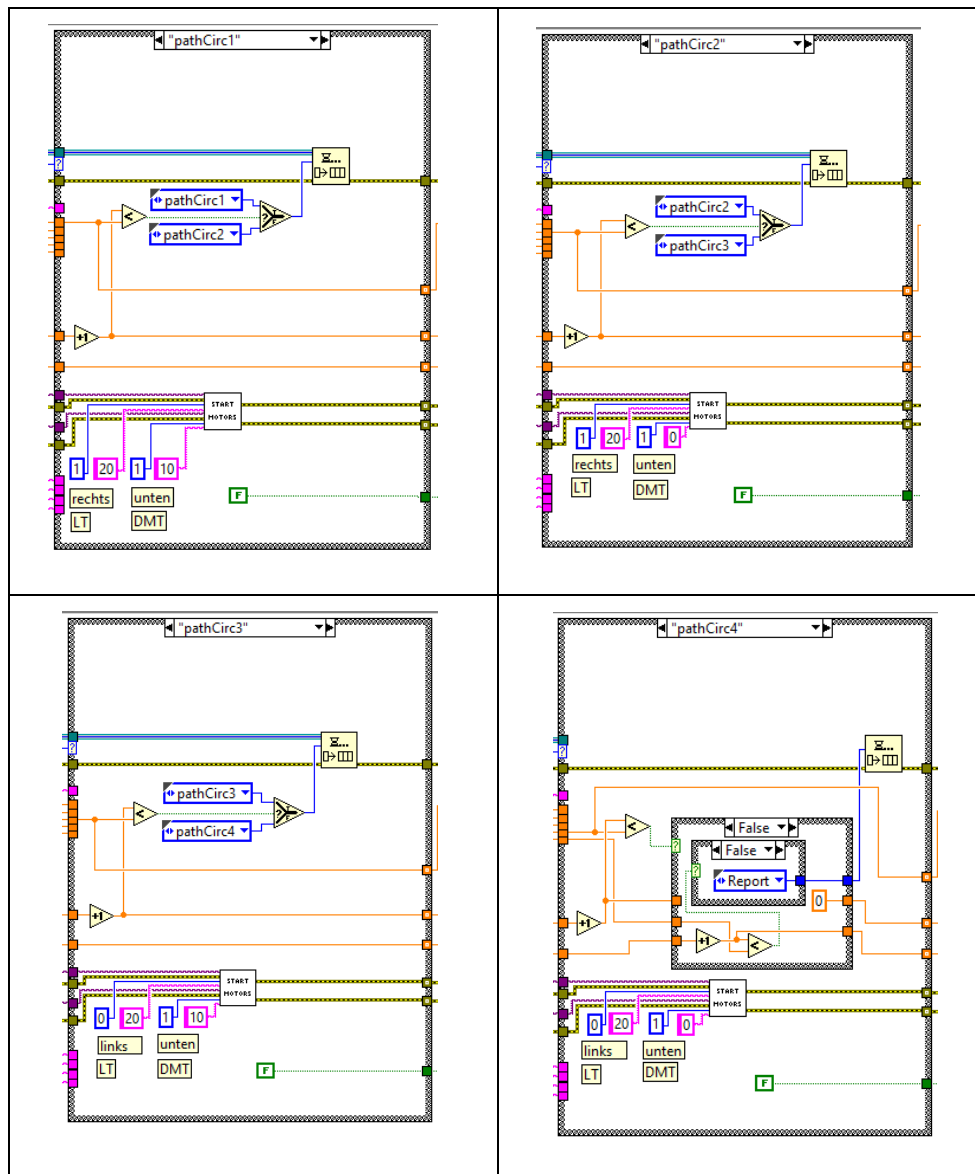
- Bewegung vom **DMT** um $0,5^\circ$ weiter nach **unten**
- Bewegung vom **LT** nach **links**

Zähler 1: von 2352 bis 4600

- Fortführung der Bewegung vom **LT** nach **links**, bis zum Startpunkt, jedoch jetzt um 1° verschoben.

Wenn *Zähler 1* = 4601, dann wird *Zähler 2* von 0 auf 1 erhöht → Wiederholung des Vorgangs und *Zähler 1* wird wieder auf 0 gesetzt.

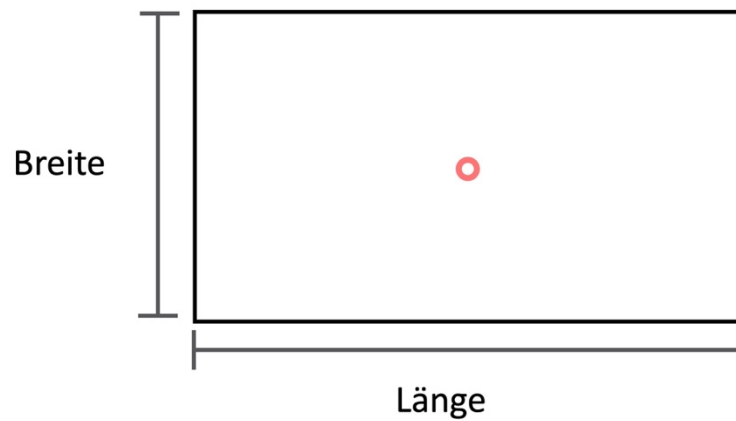
Der gesamte Vorgang wird somit fortgeführt bis *Zähler 2* auf 330 erhöht wurde.



[Abb. 9 – *pathCirc1* bis *pathCirc4*]

6.3 Fahrweg Zick-Zack

Wie beim Kreisfahrweg wird der Laser vor dem Zick-Zack-Fahrweg auf der der Mitte der Probe gerichtet (siehe Abb. 10).

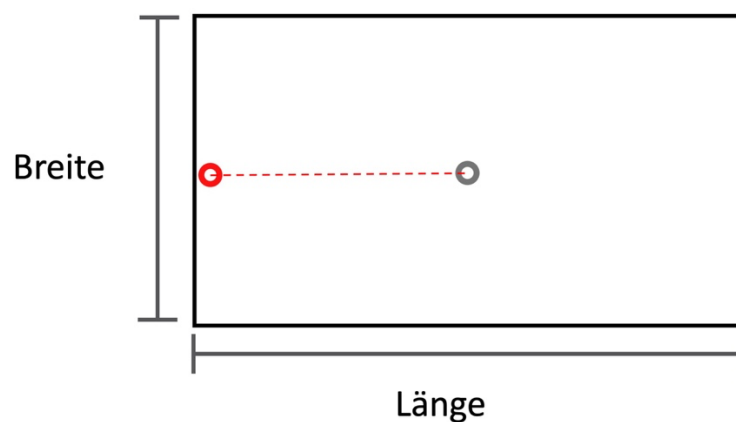


[Abb. 10 – Laser auf der Mitte der Probe]

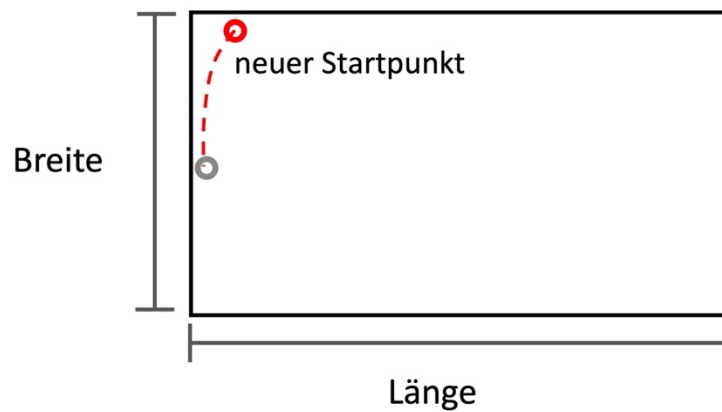
6.3.1 Pre-Positionierung der Messtische

Die Routine zur Pre-Positionierung der Messtische besteht beim Zick-Zack-Fahrweg aus zwei Teilen. Zuerst wird der Laser bis an den äußeren linken Rand der rechteckigen Messprobe positioniert.

Nachfolgend wird der DMT so weit gedreht, dass der Laser an den oberen Rand der Messprobe positioniert ist. Dies ist somit der neue Startpunkt für den Zick-Zack-Fahrweg.



[Abb. 11 – Pre-Positionierung]



[Abb. 12 – Pre-Positionierung]

6.3.2 Zick-Zack Algorithmus

Ähnlich wie der Kreis Algorithmus, wo die Bewegung des LT alterniert bei einer konstanten Bewegungsrichtung des DMT, werden beim Zick-Zack-Algorithmus diese Rollen vertauscht. Hierbei bewegt sich nämlich der DMT alternierend vor und zurück, während der LT sich konstant in eine Richtung fortbewegt.

Das Fortbewegungsmuster des Laser wird wie folgt erklärt:

Teil 1: Der Laser bewegt sich nach unten entlang eines Kreises. Der maximale Ausschlagwinkel versichert, dass der Laser nicht über die Messprobe hinaus gelangt.

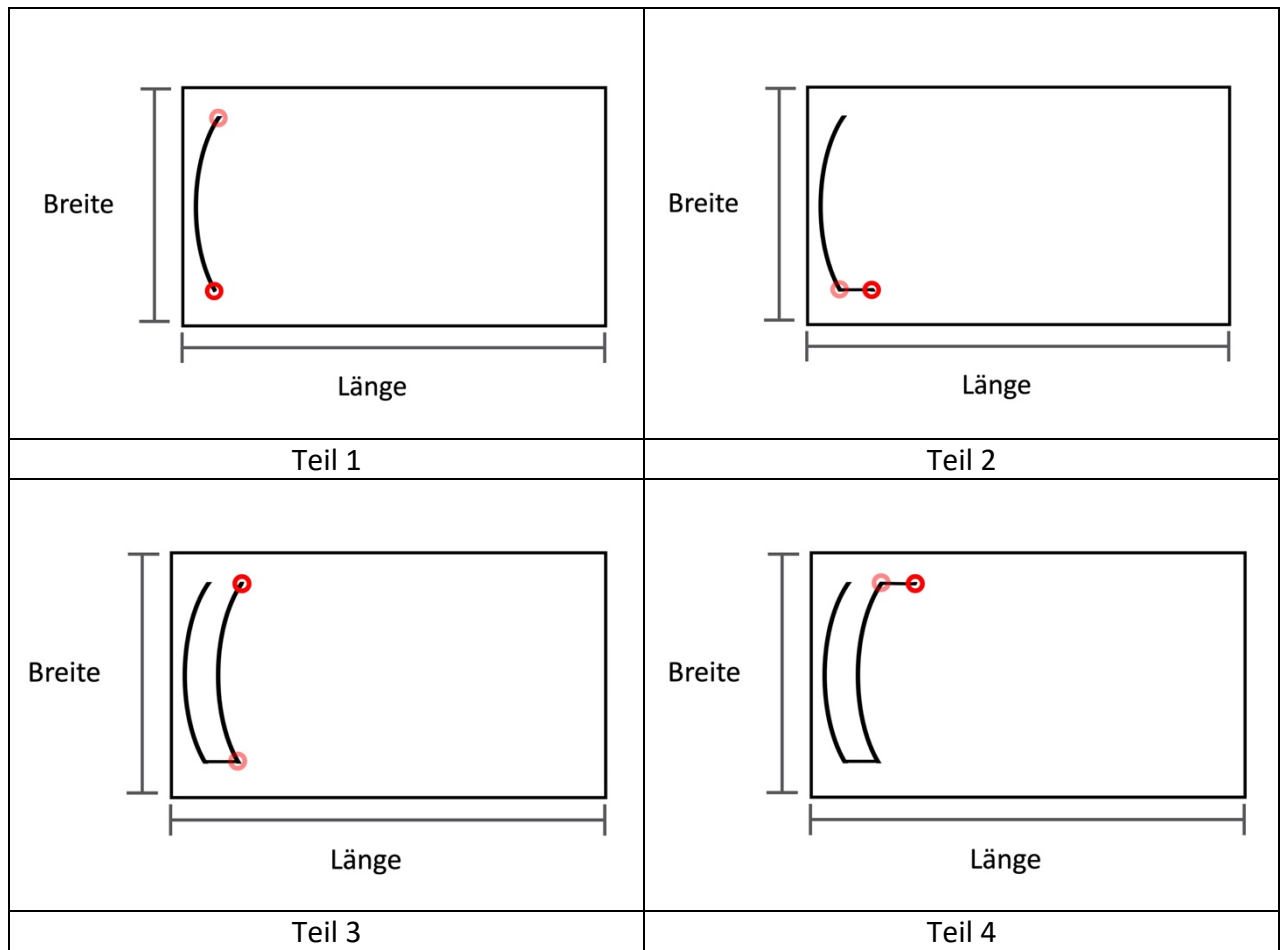
Teil 2: Der Laser bewegt sich um 0,5 mm nach rechts.

Teil 3: Der Laser bewegt sich um den gleichen Ausschlagwinkel aus Teil 1 nach oben.

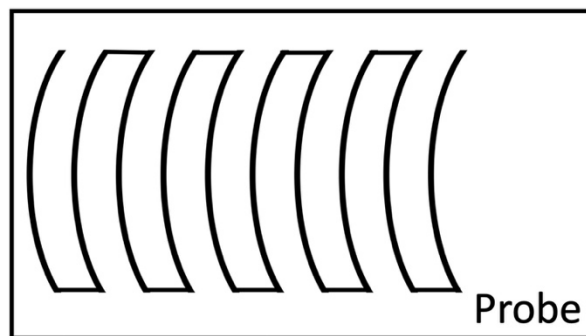
Teil 4: Der Laser bewegt sich um weitere 0,5 mm nach rechts.

Teil 5: Die Vorgänge aus Teil 1 – 4 werden wiederholt.

Da im realen Fahrweg die Vorgänge aus Teil 1 und Teil 2 teilweise parallel ablaufen, werden die resultierenden Fortbewegungsmuster etwas spitzer zu laufen als die dargestellten Zeichnungen.



[Abb. 13 – Teil 1 - 4]



[Abb. 14 - Teil 5]

6.3.3 Umsetzung in LabView

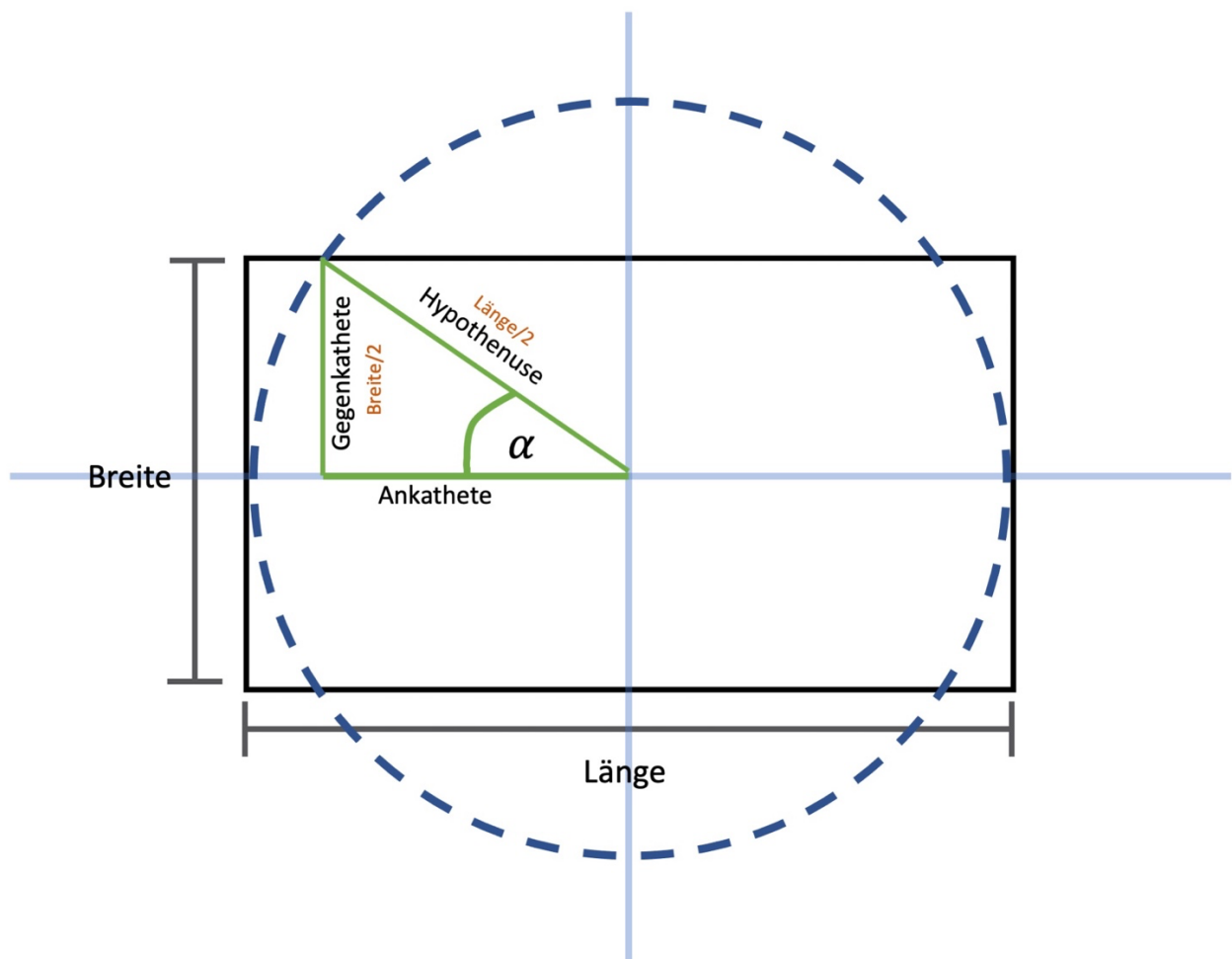
Zur Umsetzung des in Abschnitt 6.3.2 erklärten Algorithmus gelten die gleichen Annahmen wie sie in Abschnitt 6.2.3 erläutert werden. Auch hier gelten die 2000/1000 Schritte die für die Bewegung des LT/DMT um 1 mm/° notwendig sind. Ebenso werden die zwei Schleifenzähler mit den Schleifenparametern aus *counterParameter* verglichen.

Um die Berechnung dieser Parameter nachzuvollziehen, werfen wir zuerst einen Blick auf die trigonometrischen Größen die wir benötigen werden.

6.3.3.1 Ausschlagwinkel α

Abschnitt 6.3.2 hat den Algorithmus bereits soweit beschrieben, dass in Teil 1 der **Laser** sich nach entlang eines Kreises nach **unten** bewegt. Dies spiegelt eine Bewegung des **DMTs** wieder, welcher sich um einen vorgegebenen *Ausschlagwinkel* nach **oben** fortbewegt.

Ein Blick auf Abb. 15 verrät, wie sich dieser Ausschlagwinkel aus den gegebenen Größen berechnen lässt.



[Abb. 15 – Berechnung von α]

Da lediglich die **Breite** und **Länge** der Messprobe im LabView Frontpanel eingegeben werden, müssen aus diesen Größen die Schleifenparameter berechnet werden. Es wird der maximale Ausschlagwinkel gesucht, um den sich der DMT bewegen kann ohne dass der Laser über den Rand der Messprobe gelangt.

Zeichnen wir nun auf die Messprobe, ausgehend von der Mitte, einen Kreis mit dem Radius:

$$r = \frac{Länge}{2}$$

Es wird der Winkel α gesucht, bei dem die Gegenkathete der Hälfte der Breite entspricht. Wir gehen von der bekannten Formel aus:

$$\text{Sinus von } \alpha = \sin \alpha = \frac{\text{Gegenkathete}}{\text{Hypothenuse}} = \frac{\text{Breite}/2}{r} = \frac{\text{Breite}/2}{Länge/2}$$

Mit der Umkehroperation vom Sinus erhalten wir nun den Winkel α :

$$\sin^{-1}\left(\frac{\text{Breite}/2}{Länge/2}\right) = \alpha$$

Somit können wir sichergehen, dass bei einem Beginn der Messroutine vom *neuen Startpunkt* aus (siehe Abb. 12 in 6.3.1), der DMT sich maximal um 2α fortbewegen kann, ohne den Laser über die Messprobe hinaus zu versetzen.

6.3.3.2 Maximale Länge

Ebenso können wir mit der Ankathete berechnen um wie viele mm sich der LT fortbewegen kann. Diese maximale Länge setzt sich wie folgt zusammen:

$$l_{max} = (\cos \alpha) * \frac{Länge}{2} + \frac{Länge}{2}$$

oder:

$$l_{max} = \frac{Länge}{2} * (\cos \alpha + 1)$$

6.3.3.3 Berechnung der Parameter in LabView

Die beiden vorherigen Abschnitte beschreiben mathematisch, welche Berechnungen in *counterParameter* für den Fahrweg Zick-Zack benötigt werden. Genau wie beim Fahrweg Kreis werden 5 Parameter ausgegeben, mit denen die beiden vorhandenen Zähler verglichen werden. Auch hier korrespondieren die in Abb. 13 und 14 gezeigten Teile 1 – 5 des Algorithmus, mit den Parametern 1 – 5:

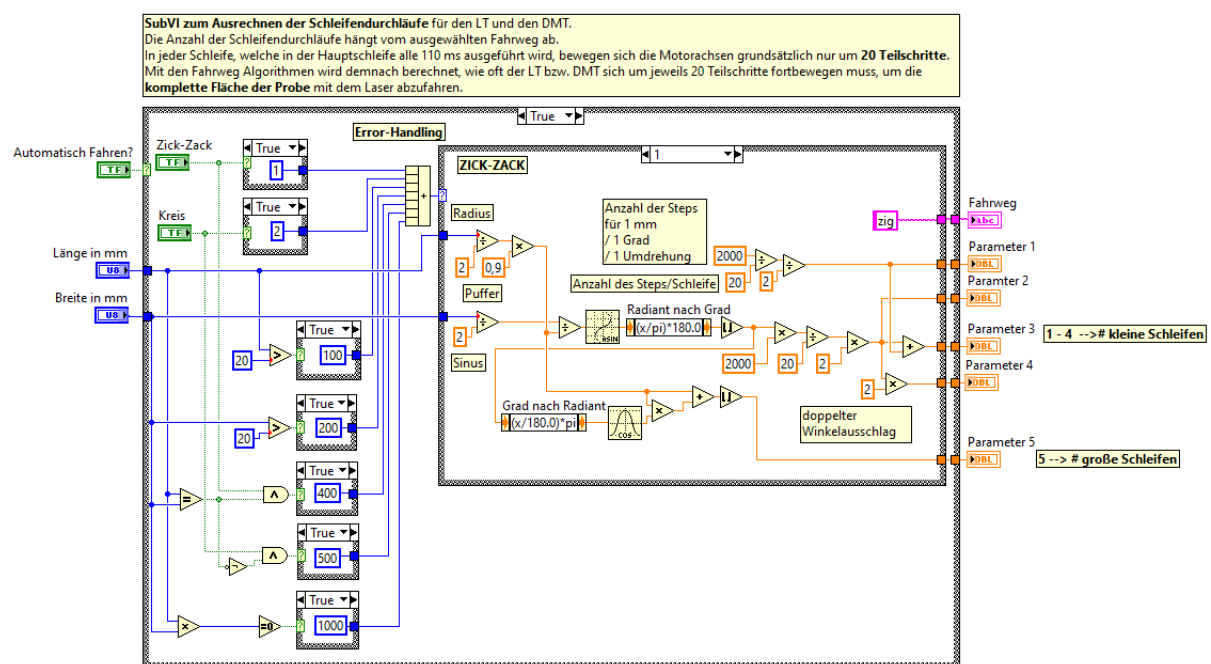
Parameter 1 → Anzahl der 20-Schritt-Schleifen für eine Bewegung des LT um 0,5 mm (Teil 2).

Parameter 2 → Anzahl der 10-Schritt-Schleifen für eine Bewegung des DMT um den doppelten Ausschlagwinkel 2α (Teil 1).

Parameter 3 → Anzahl der 20-Schritt-Schleifen für eine weitere Bewegung des LT um 0,5 mm (Teil 3). Hierfür werden Parameter 1 und 2 zusammen addiert.

Parameter 4 → Anzahl der 10-Schritt-Schleifen für eine Bewegung des DMT um den doppelten Ausschlagwinkel 2α wieder zurück (Teil 4). Hierfür wird Parameter 2 mit 2 multipliziert.

Parameter 5 → Anzahl der Wiederholungen von Teil 1 – 4. Dieser Parameter ergibt sich aus der maximalen Länge l_{max} . Da Teil 1 – 4 des Algorithmus insgesamt eine Fortbewegung der LT von **1 mm** beschreiben, resultiert die Anzahl der Wiederholungen von Teil 1 – 4 aus l_{max} in mm.



[Abb. 16 – Blockdiagramm von *counterParameter*]

Ein Beispiel Fahrweg könnte somit wie folgt aussehen, wobei lediglich mit nur 2 Zählern der Fortschritt des Algorithmus und somit der Fortschritt des Fahrwegs evaluiert wird:

Zähler 1: von 0 bis 50

- Bewegung vom **LT** nach **links** um 0,5 mm
- Bewegung vom **DMT** nach **oben**

Zähler 1: von 51 bis 2300

- Fortführung der Bewegung vom **DMT** nach **oben** um 2α

Zähler 1: von 2301 bis 2351

- Bewegung vom **LT** um 0,5 mm weiter nach **links**
- Bewegung vom **DMT** nach **unten**

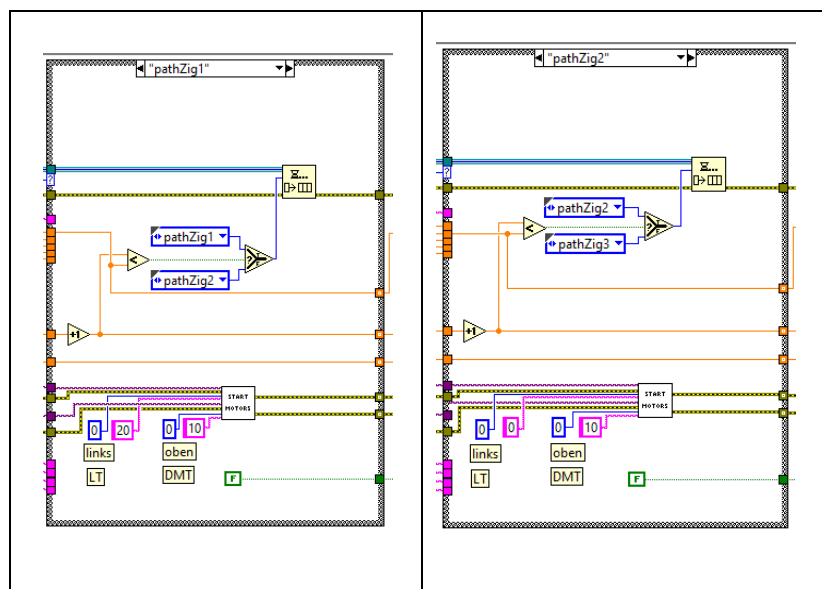
Zähler 1: von 2352 bis 4600

- Fortführung der Bewegung vom **DMT** nach **unten** um 2α , bis zum Startpunkt, jedoch jetzt um 1 mm verschoben.

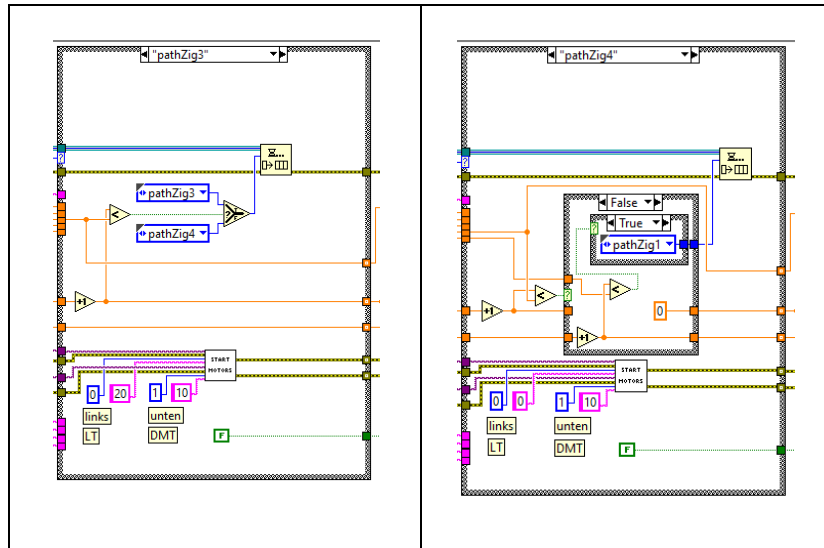
Wenn *Zähler 1* = 4601, dann wird *Zähler 2* von 0 auf 1 erhöht → Wiederholung des Vorgangs und *Zähler 1* wird wieder auf 0 gesetzt.

Der gesamte Vorgang wird somit fortgeführt bis *Zähler 2* auf l_{max} in mm erhöht wurde.

Die gleichen Parameter werden in der Pre-Positionierungsroutine benutzt, um den Laser um Länge/2 (mit Puffer) nach links und um den Winkel α nach oben zu verschieben.



[Abb. 17 – *pathZig1* und *pathZig2*]



[Abb. 18 – *pathZig3* und *pathZig4*]

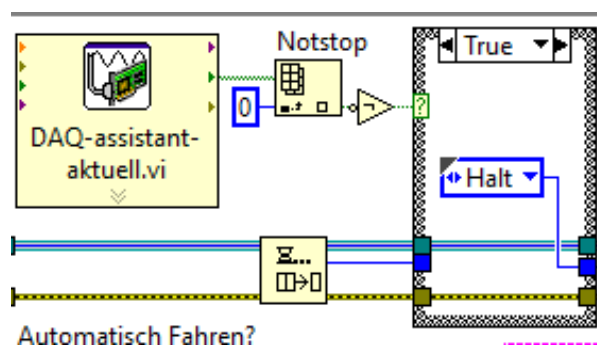
7. Notstopp-Taster

Die Notstopp-Taster beider Messtische sind parallel geschaltet und werden über eine digitale/analoge Messdaten-Erfassungskarte (National Instruments DAQ USB-6001) mit Spannung versorgt.

Die DAQ-Karte lässt sich in LabView über den DAQ-Assistenten konfigurieren, womit ein digitaler Eingang zur Messung des Parallel-Schaltkreises freigeschaltet wird.

In der QSM befindet sich diese Funktion (*DAQ-assistent-aktuell*) in der unteren While-Schleife der QSM. Es wird also mit jedem Schleifendurchlauf (alle 110 ms) die DAQ-Karte abgefragt.

Wird nun einer der Notstopp-Taster der Messtische betätigt, schließt sich der Stromkreis, womit am Eingang der DAQ-Karte eine Spannung anliegt. Anschließend wird der *Halt*-State priorisiert in die Queue gesetzt und beide Motoren mit sofortiger Wirkung gestoppt.



[Abb. 1 – *DAQ-assistent-aktuell*]

8. Quellen

[Abschnitt 1 Abb. 1] - K. Sugioka, Y. Cheng, Thoss Media & De Gruyter, Paper, 2012: *“A tutorial on optics for ultrafast laser materials processing: Basic microprocessing system to beam shaping and advanced focusing methods”*