Institut für Rechnergestützte Ingenieursysteme
Universität Stuttgart
Universitätsstraße 38
D – 70569 Stuttgart

Diplomarbeit Nr. 3065

Framework zur Steuerung von prototypischen 3D-Ink-Jet-Druckanlagen auf der Basis von National Instruments Embedded Controllern

Lucian Cristian Depold

Studiengang: Informatik

Prüfer: Prof. Dr. Otto Eggenberger

Betreuer: Dipl.-Ing. Oliver Refle

begonnen am: 01. Mai 2009

beendet am: 31. Oktober 2010

CR-Klassifikation: B.4.2, D.1.3, D.1.7, D.2.2, D.3.3

Inhaltsverzeichnis

1 Einleitung **4**

2 Grundlagen **5**

2.1 Generative Fertigungsverfahren auf Basis der Ink-Jet-Drucktechnik . . . 5

2.1.1 Generativer Fertigungsprozess . . . 6

2.1.2 3D-Printing Verfahren nach dem MIT-Patent . . . 6

2.1.3 Ink-Jet-Drucktechnik . . . 8

2.2 Sensoren und Aktoren in Ink-Jet-Druckanlagen . . . 10

2.2.1 Temperatur Sensoren . . . 10

2.2.2 Elektrische Heizelemente . . . 11

2.2.3 Ventile . . . 11

2.2.4 Elektrische Membranpumpen . . . 12

2.2.5 Schrittmotoren . . . 12

2.2.6 Drucksensoren . . . 13

2.2.7 Induktive Sensoren . . . 13

2.3 NI Embedded Controller - CompactRio 9014 . . . 14

2.3.1 I/O-Module (Input/Output-Module) . . . 14

2.3.2 FPGA (Field Programmable Gate Array) . . . 15

2.3.3 Echtzeitbetriebssystem . . . 16

2.4 LabVIEW - Ein graphisches Programmiersystem . . . 17

2.4.1 Virtuelle Instrumente, Bedienoberfläche und Blockdiagramm . . . 17

2.4.2 Datenflussprogrammierung und Nebenläufigkeit . . . 18

2.4.3 Unterprogramme (SubVIs) . . . 19

2.4.4 Compiler, Programmausführung und Debugging . . . 20

2.4.5 Datentypen und Datenstrukturen . . . 21

2.4.6 Kontrollstrukturen und Variablen . . . 23

2.4.7 Zustandsautomaten . . . 24

2.5 Frameworks . . . 26

2.5.1 Merkmale und Eigenschaften . . . 26

2.5.2 Vor- und Nachteile . . . 28

3 Anforderungsanalyse und Entwurf des Frameworks **29**

3.1 Analyse einer prototypischen Keramikdruckanlage . . . 29

3.1.1 Aufbau und Funktion . . . 30

3.1.2 Bedienung der Druckanlage . . . 32

3.2 Anforderungen an das Framework 33
3.2.1 Funktionale Anforderungen 33
3.2.2 Flexibilitätsanforderungen 34
3.3 Entwurf des Frameworks 34
3.3.1 Komponenten der PC-Ausführungsumgebung 36
3.3.2 Komponenten der RT-Ausführungsumgebung 37
3.3.3 Komponenten der FPGA-Ausführungsumgebung 39

4 Umsetzung des Frameworks mit LabVIEW 40
4.1 Übersicht 40
4.2 Initialisierung der Queues 41
4.2.1 Typdefinitionen zur Kommunikation mit dem Zustandsautomaten 41
4.2.2 Sonstige Typdefinitionen 43
4.3 PC RT Kommunikation 44
4.3.1 Verbindungsaufbau 45
4.3.2 Datenaustausch 45
4.3.3 Verbindungsabbruch 45
4.4 Zustandsautomat 46
4.4.1 Senden von Statusmeldungen 47
4.4.2 Klassisches und spezielles Zustandsverhalten 49
4.4.3 Reaktion auf Übergangsbefehle 50
4.4.4 Reaktion auf Abbruchbefehle 52
4.4.5 Senden von Befehlsantworten 53
4.4.6 Autonomes Verhalten des Zustandsautomaten 53
4.4.7 Minimalautomat 54
4.5 Benutzerinterface und Eventhandler 54
4.5.1 Benutzerinterface für den Endanwender 56
4.6 Sensoren und Aktoren Mapping 57
4.7 Sensordaten auslesen 58
4.8 Temperaturregelung 58
4.9 Druckregelung 59
4.10 Schrittmotorsteuerung 61
4.10.1 Beschreiben der Look-Up Tables 61
4.10.2 Motorbewegung 62
4.11 Druckkopfansteuerung 63
4.11.1 Übertragung der Druckdaten an den FPGA 63
4.11.2 Druckprozess 64

5 Anwendungsentwicklung mit dem Framework **66**
5.1 Anpassung an die Aktoren und Sensoren 66
5.1.1 Schrittmotoren 66
5.1.2 Druckköpfe 67
5.1.3 Pumpen 67
5.1.4 Erzeugen der FPGA-Bitfile 67
5.1.5 Sensordaten auslesen 67
5.2 Anpassung an die benötigten Regelungen 67
5.3 Anpassung des Minimalautomaten 68
5.3.1 Einbetten von Benutzerquellcode 68
5.4 Testen der Anwendung 69
5.5 Benutzerinterface für den Endanwender 70
5.6 Kompilieren der Anwendung 70

6 Evaluation **71**
6.1 Kommunikation 71
6.2 Benutzerinterface und Zustandsautomat 71
6.3 Druckregelung 73
6.4 Temperaturregelung 73
6.5 Schrittmotorsteuerung 73
6.6 Druckkopfansteuerung 74

7 Zusammenfassung und Ausblick **75**

Abkürzungsverzeichnis **76**

Abbildungsverzeichnis **77**

Literatur **79**

1 Einleitung

Die Geschichte der Ink-Jet-Drucktechnik begann in den 1960er Jahren mit der Entwicklung der ersten Tintenstrahldrucker.
Zunächst war das Bedrucken von Oberflächen nur mit einem kontinuierlichen Tintenstrahl möglich, jedoch wurden mit der Zeit auch Verfahren entwickelt, mit denen das Ausstoßen einzelner Tintentröpfchen realisierbar wurde.
Dieses als „Drop On Demand“ (DOD) bezeichnete Verfahren findet man heutzutage in den Druckköpfen vieler Drucker aus dem Büro- und Heimbereich wieder. Sie werden dazu verwendet, digitale Grafiken oder Textdokumente auf Papier zu drucken.
Ein weiterer Einsatzbereich von DOD-Druckköpfen ist die generative Fertigung von Bauteilen. Die Bauteile werden dabei durch das schichtweise Verdrucken von Bindemitteln aus Pulver oder Granulaten erzeugt. Theoretisch können so Bauteile aus beliebigen Materialien generiert werden, da ein Bindemittel mit entsprechenden Eigenschaften für beliebige Materialien als Formgeber dienen kann [4].
Die Fraunhofer Gesellschaft betreibt Forschung zur Auffindung neuer Pulver-Bindemittel Kombinationen. In diesem Zusammenhang ist es beim Fraunhofer IPA in Stuttgart zur Planung und Konstruktion von mehreren prototypischen 3D-Ink-Jet-Druckanlagen gekommen. Für diese Druckanlagen werden Maschinensteuerungen benötigt, damit daran die Eigenschaften von neuen Bindemitteln und Pulver getestet werden können.
Da das Fraunhofer IPA in der Vergangenheit gute Erfahrung mit Embedded Controllern von National Instruments bei der Umsetzung von Maschinensteuerungen gemacht hat, sollen diese auch zur Steuerung der 3D-Ink-Jet-Druckanlagen verwendet werden.
Neben diesen Embedded Controllern besitzen die Druckanlagen weitere Gemeinsamkeiten in ihrer Funktionsweise und der dazu verwendeten Hardware.
Ziel dieser Arbeit ist es die Gemeinsamkeiten dieser 3D-Ink-Jet-Druckanlagen aufzudecken und sie in einem Software-Framework zu vereinen, um so die Zeit und der Arbeitsaufwand zur Realisierung der Maschinensteuerungen zu verkürzen.
Die in der vorliegenden Arbeit genannten Firmen- und Markennamen werden ohne Hinweis auf eventuell eingetragene Warenzeichen verwendet. Ein fehlender Hinweis auf ein eingetragenes Warenzeichen bedeutet nicht, dass es sich nicht um ein eingetragenes Warenzeichen handelt.

2 Grundlagen

In diesem Kapitel sollen die Grundlagen für das mit dieser Diplomarbeit entwickelte Framework gebildet werden. Dazu wird zunächst das allgemeine Prinzip der generativen Fertigung erläutert und dabei der Einsatz von Ink-Jet-Druckköpfen aufgezeigt. Es werden außerdem für Druckanlagen typische Sensoren und Aktoren sowie ein Embedded Controller von National Instruments vorgestellt. Abschließend wird auf die LabVIEW Entwicklungsumgebung und die Definition eines Frameworks eingegangen.

2.1 Generative Fertigungsverfahren auf Basis der Ink-Jet-Drucktechnik

Durch die immer größer gewordene Leistungsfähigkeit des Computers entstanden in den 1980er Jahren auf Basis rechnerinterner 3D-Modellen von Objekten neue Methoden zur günstigen und werkzeugfreien Fertigung von Mustern, Prototypen und Endprodukten, die heutzutage als generative Fertigungsverfahren bezeichnet werden.
Die Objekte werden dabei schichtweise mit Hilfe von chemischen und physikalischen Prozessen aus formlosen Materialien, wie Flüssigkeiten oder Pulver, aufgebaut. Dieses Schichtbauprinzip gewährt weitestgehende Geometriefreiheit bei der Fertigung und erlaubt die Herstellung komplexer geometrischer Strukturen, die mit traditionellen Verfahren, wie zum Beispiel Fräsen oder Gießen, nur schwer oder gar nicht realisierbar sind.

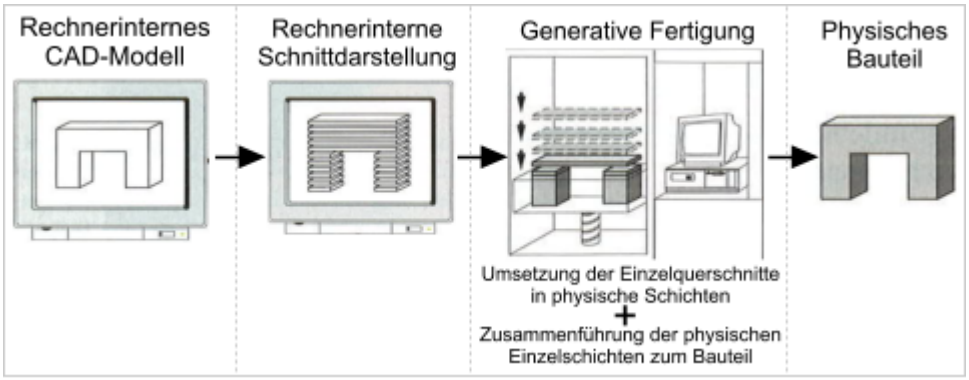

Abbildung 1: Prinzip der generativen Fertigung [4]

2.1.1 Generativer Fertigungsprozess

Ausgangspunkt der generativen Fertigung ist ein rechnerinternes CAD-Modell, das für die Generierung des Bauteils mit mathematischen Methoden in virtuelle Schichten gleicher Höhe geschnitten wird (Abb. 1). Diese virtuellen Schichten werden dann in einer generativen Fertigungsanlage durch einen physikalischen oder chemischen Prozess in physische Schichten umgewandelt und die Einzelschichten zum vollständigen Bauteil aufeinander geschichtet.
Je nach verwendetem Verfahren ist danach eventuell noch eine Weiterverarbeitung[1] des Bauteils notwendig, da nicht jeder Fertigungsprozess eine Verbindung der einzelnen Schichten untereinander möglich macht. Eine der verbreitesten Methoden ist die Erzeugung der Schichten mit einem oder mehreren fokussierten Laserstrahlen aus Kunststoffpulvern oder flüssigen Monomeren [4].
Die generative Herstellung mit Hilfe von Ink-Jet-Druckköpfen durch das Verdrucken von Bindemittel stellt weitere Methode dar. Auf dieses als 3D-Printing[2] bezeichnete Verfahren wird im Folgenden näher eingegangen.

2.1.2 3D-Printing Verfahren nach dem MIT-Patent

Das 3D-Printing ist ein vom „Massachusetts Institute of Technology“ (MIT) entwickeltes 3D-Druckverfahren, bei dem Pulver oder Granulate mit Hilfe von Bindemittel verklebt werden [13]. Das Bindemittel wird mittels Ink-Jet-Druckköpfen zeilenweise auf eine Pulver- oder Granulatschicht aufgetragen. Durch die Verwendung mehrerer Druckköpfe und kolorierter Binder können mit diesem Verfahren auch mehrfarbige Modelle gefertigt werden.
Die zur Umsetzung dieses Verfahrens benötigten Elemente sind in Abbildung 2 zu sehen. Vor dem eigentlichen Druckprozess wird mittels einer Walze und des mit Pulver gefüllten Pulverraums ein sogenanntes Pulverbett im Bauraum erzeugt:

1. Die Plattform des zunächst leeren Bauraums wird ganz nach oben gefahren.

2. Danach wird die Bauraumplattform um eine Schicht abgesenkt.

3. Die Plattform des bis zum Ansatz gefüllten Pulverraums wird ein Stück nach oben gefahren, damit Pulver über den Pulverraum hinausragt.

4. Mit Hilfe einer Walze wird nun das überstehende Pulver in den um eine Schicht abgesenkten Bauraum gleichmäßig aufgetragen.

[1] Zum Beispiel durch Infiltration mit einem Bindemittel oder durch Erhitzen
[2] Diese Bezeichnung wird durch ein Basispatent des MIT festgeschrieben [4]

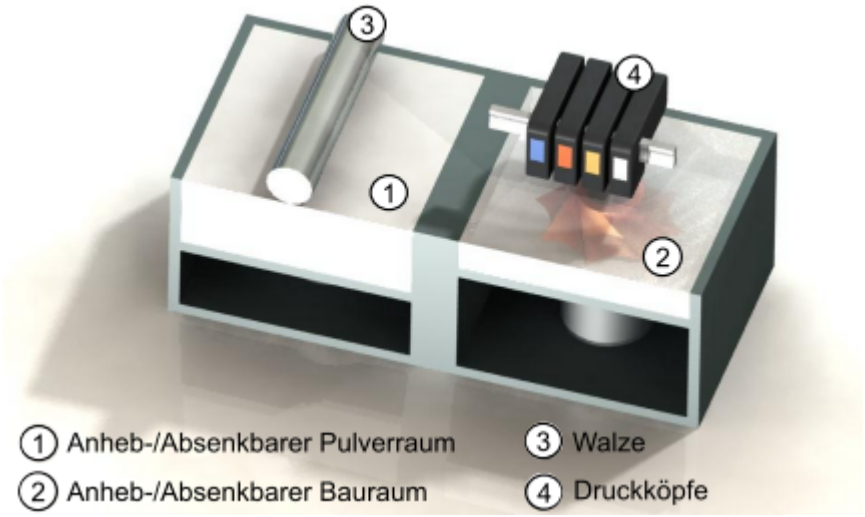

Abbildung 2: Grundelemente des 3D-Printing Verfahrens [21]

5. Das Auftragen von Pulverschichten (Punkt 2 bis 4) wird solange wiederholt bis ein genügend starkes Pulverbett im Bauraum erzeugt wurde.

Dieses Pulverbett wird benötigt, um das zu erzeugende Bauteil während dem Druckprozess zu stabilisieren [4]. Der Druckprozess beginnt anschließend, indem eine neue Pulverschicht in den Bauraum aufgetragen wird.

Die Druckköpfe bewegen sich über diese Pulverschicht (Abb. 3) und Be-

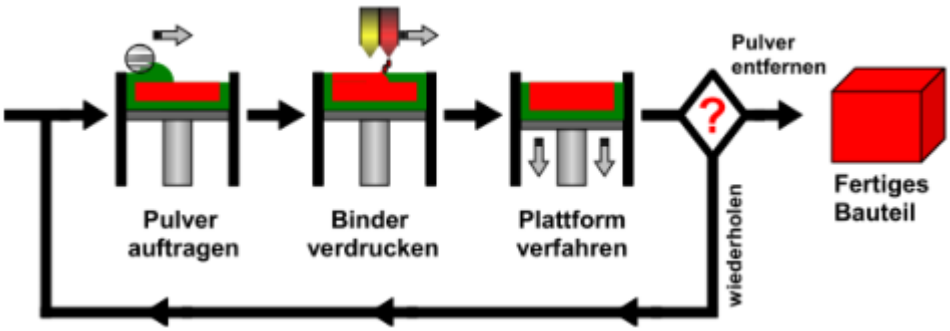

Abbildung 3: 3D-Printing Druckprozess [21]

drucken die zu verfestigenden Stellen mit Bindemittel. Der Binder erzeugt sowohl einen lokalen Zusammenhalt zwischen den einzelnen Pulverteilchen in der Ebene als auch mit der darunter liegenden Schicht. Dieses Auftragen von Pulverschichten mit anschließendem Bedrucken wird so lange wiederholt, bis das Bauteil vollständig erzeugt wurde.

Vorteile des 3D-Printing Verfahrens

- Theoretisch unbegrenzte Materialauswahl: Für die Erstellung eines Modells kommt es auf eine abgestimmte Materialkombination aus Bindemittel und Pulver an [4].
- Beimischung von Zusätzen: Im Pulver und Bindemittel können Zusätze wie zum Beispiel Partikel, Farbstoffe beigemischt werden.
- Darstellung von Farben: Kein anderes generatives Fertigungsverfahren erlaubt die Herstellung von farbigen Bauteilen [21].

Nachteile des 3D-Printing Verfahrens

- Nachbehandlung notwendig: In den meisten Fällen muss das erzeugte Bauteil weiterverarbeitet werden [4]. Wird beispielsweise aus Metallpulver ein Bauteil erzeugt, dient das Bindemittel nur als Formgeber. Das Bauteil muss im Anschluss erhitzt werden, um eine chemische Verbindung zwischen den einzelnen Metallteilchen zu erreichen.
- Schrumpfen des Bauteils: Bei der Weiterverarbeitung, zum Beispiel durch Erhitzen, kann es zur Verkleinerung des Bauteils kommen. Dieser Effekt kann jedoch im Allgemeinen durch Erfahrung beherrscht werden.

2.1.3 Ink-Jet-Drucktechnik

Unter dem Begriff Ink-Jet-Drucktechnik versteht man das Bedrucken von Oberflächen durch kontrollierten Ausstoß von Flüssigkeitstropfen mit Hilfe von Düsen. Die Düsen sind in einem sogenannten Druckkopf angeordnet, der sie mit der zu druckenden Flüssigkeit versorgt und entweder statisch oder dynamisch über der zu bedruckenden Oberfläche positioniert ist.
Durch das Erzeugen einer Bewegung zwischen Druckkopf und Oberfläche und dem dazu synchronisierten Auftragen der Tropfen können Rasterbilder gedruckt werden, deren Auflösung allein von der Tröpfchengrösse abhängig ist. Man unterscheidet zwischen Druckköpfen, die einen kontinuierlichen Tintenstrahl (Continuous Ink Jet) erzeugen, und Druckköpfen, die einzelne Tropfen ausstoßen können (Drop on Demand).

Tintenstrahl-Druckköpfe (Continuous Ink Jet)
Tintenstrahl-Druckköpfe erzeugen mit ihren Düsen einen oder mehrere kontinuierliche Tintenstrahlen (Abb. 4), die mit Hilfe eines piezoelektrischen Wandlers gleichmäßig in einzelne Tropfen zerlegt werden.
Die benötigten Tropfen werden anschließend elektrostatisch aufgeladen[3], sodass sie durch Elektroden entweder in einem Auffangbehälter oder auf die

[3]Manche Druckköpfe laden die nicht benötigten Tropfen auf.

zu bedruckende Oberfläche gelenkt werden können. Dabei werden die aufgefangenen Tropfen wieder in den Druckkreislauf aufgenommen. Durch unterschiedlich starke Aufladung der Tropfen einer Düse ist bei manchen Druckköpfen das Erzeugen von breiteren Druckzeilen möglich, da der Grad der Ablenkung gesteuert werden kann.

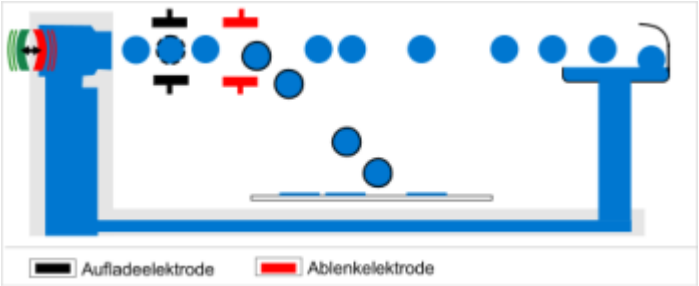

Abbildung 4: Prinzip eines ContinuousInkJet-Druckkopfes

Bubble-Jet-Druckköpfe (Drop on Demand)
Diese Druckköpfe erzeugen nur die benötigten Tintentropfen, indem sie mit Hilfe eines Heizelementes das in der Tinte enthaltene Wasser in unmittelbarer Nähe der Düsenöffnung plötzlich erhitzen (Abb. 5). Dadurch entsteht eine Dampfblase, durch deren Druck ein Tropfen aus der Düse gepresst wird. Dieses Verfahren erfordert eine thermische Stabilität der verwendeten Flüssigkeiten und ist anfällig für Korrosionsprozesse am Heizelement, da Temperaturen bis zu 400°C erreicht werden können [4]. Außerdem ist eine Reinigung der Druckköpfe notwendig, da die verwendeten Flüssigkeiten eintrocknen und die Düsen verstopfen können.

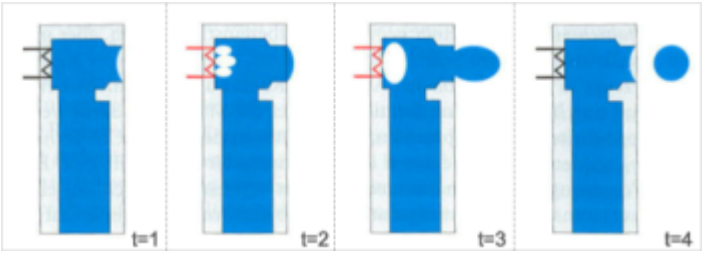

Abbildung 5: Prinzip eines BubbleJet-Druckkopfes [4]

Piezo-Druckköpfe (Drop on Demand)
Piezo-Druckköpfe nutzen Keramikelemente, die sich durch elektrische Spannungsimpulse verformen, um kurzzeitig Druck auf die sich innerhalb der Dü-

sen befindliche Tinte auszuüben (Abb. 6). Dadurch wird der benötigte Tropfen, der proportional zur Stärke der Verformung des Keramikelementes in der Größe variieren kann, durch die Düsenöffnung gepresst.
Da bei diesem Verfahren keine hohen Temperaturen nötig sind, können mehr Flüssigkeiten verwendet werden als bei Bubble-Jet-Druckköpfen. Zusätzlich zum Eintrocknen der Flüssigkeit kann es bei diesem Verfahren auch zur Bildung von Luftblasen kommen, welche die Düsen verstopfen und eine Reinigung erforderlich machen.

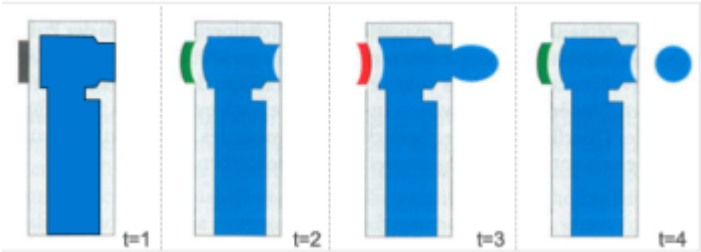

Abbildung 6: Prinzip eines Piezo-Druckkopfes [4]

2.2 Sensoren und Aktoren in Ink-Jet-Druckanlagen

Neben den im vorherigen Kapitel vorgestellten Druckköpfen besitzen Ink-Jet-Druckanlagen weitere Aktoren und Sensoren, deren Funktionsprinzip und Verwendung im Folgenden näher erklärt werden soll.

2.2.1 Temperatur Sensoren

Temperatursensoren sind elektronische Bauelemente, die Wärme auf eine elektrische Größe abbilden. Hierzu existieren eine Reihe von Verfahren, die sich teilweise unterschiedliche physikalische Effekte zu Nutze machen. Die größte Bedeutung haben Temperatursensoren auf Basis von Widerständen oder Thermoelementen [15].

Temperaturmessung mit Widerständen
Bestimmte Materialien, wie Keramiken und Silizium, verringern oder erhöhen ihren elektrischen Widerstand in Abhängigkeit der Temperatur. Dieser Umstand kann zur Temperaturmessung verwendet werden, da die bei Stromfluss gemessene Spannung am Widerstand in Relation zu dessen Temperatur steht.

Temperaturmessung mit Thermoelementen
Ein Thermoelement (Abb. 7) besteht aus zwei unterschiedlichen Metallen,die an einem Ende miteinander verschweißt sind [30]. Wird deren Verbindungsstelle erwärmt, entsteht eine Temperaturdifferenz zu den unverbundenen Enden der beiden Metalle. Aufgrund der Temperaturdifferenz kommt es an den freien Enden zu einer messbaren elektrischen Spannungsdifferenz, die ein Rückschluss auf die Temperatur an der Verbindungsstelle erlaubt.

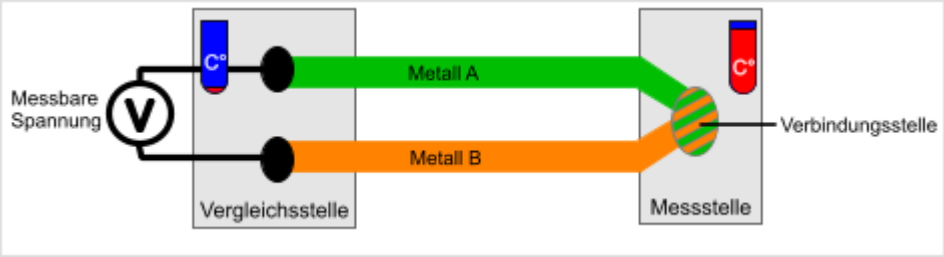

Abbildung 7: Schematische Darstellung eines Thermoelementes [30]

Anwendung in Druckanlagen
Temperatursensoren werden in Kombination mit elektrischen Heizelementen zur Temperierung von Flüssigkeiten oder Elementen verwendet.

2.2.2 Elektrische Heizelemente

Elektrische Heizelemente erzeugen Wärme mit Hilfe von elektrischem Strom. Sie bestehen in der Regel aus einem hohen elektrischen Widerstand, wie zum Beispiel einem hochohmigen Draht, der sich bei Stromfluss erhitzt und gegenüber dem zu erwärmenden Stoff elektrisch isoliert ist [26]. Prinzipiell können hierzu alle Metalle verwendet werden, jedoch werden Legierungen wie Konstantan oder Manganin bevorzugt, da diese in einem breiten Temperaturspektrum einen annähernd gleichen Widerstand und einen hohen Schmelzpunkt besitzen.

Anwendung in Druckanlagen
Heizelemente werden in InkJet-Druckanlagen dazu verwendet, die zu druckenden Flüssigkeiten oder Elemente, welche mit ihnen in Berührung kommen, auf bestimmte Temperaturen zu erwärmen.

2.2.3 Ventile

Ventile dienen dem kontrollierten Ein- und Auslass von Flüssigkeiten und Gasen sowie der Steuerung von Durchflussmengen und deren Durchflussrich-

tung. Sie werden nach Form, Betätigungs- und Bauart des Absperrkörpers unterschieden [32].

Anwendung in Druckanlagen
Ventile werden in Druckanlagen beispielsweise zur kontrollierten Versorgung der Druckköpfe mit Flüssigkeiten verwendet (2.2.4).

2.2.4 Elektrische Membranpumpen

Eine elektrische Membranpumpe dient der Förderung von Flüssigkeiten und Gasen [27]. Sie besteht aus einem elektrischen Antrieb, der von dem zu förderndem Medium durch eine Membran getrennt ist. Mit Hilfe des Antriebs kann die Membran eine Saug- und eine Pumpstellung annehmen . In der Saugstellung wird das Medium von einer Seite in eine Kammer gesaugt (Abb. 8,links). In der Pumpstellung wird die Kammer zur anderen Seite hin geleert(Abb. 8,rechts). Durch schnelles Abwechseln zwischen diesen beiden Stellungen kommt es zur Förderung des Mediums.

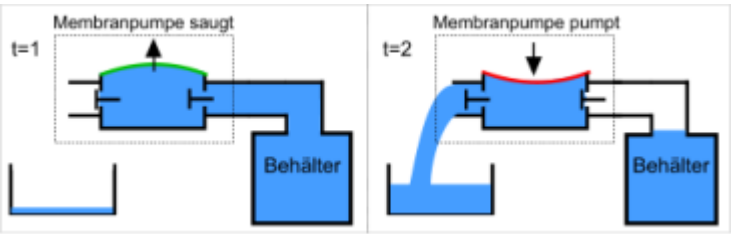

Abbildung 8: Prinzip einer Membranpumpe, links: Saugstellung, rechts: Pumpstellung

Anwendung in Druckanlagen
Membranpumpen werden in Kombination mit Ventilen zur Erzeugung von Über- und Unterdruck verwendet, wodurch beispielsweise die Versorgung der Druckköpfe mit Flüssigkeiten und deren Reinigung möglich wird.

2.2.5 Schrittmotoren

Ein Schrittmotor ist ein Motor, dessen Rotor schrittweise um einen minimalen Winkel gedreht werden kann [29]. Die Drehbewegung wird durch ein elektromagnetisches Feld erzeugt, welches an diskreten Stellen des unbeweglichen Motorteils so aufgebaut wird, dass der bewegliche Rotor durch Anziehung in Drehung versetzt wird (Abb. 9). Die Drehgeschwindigkeit kann durch Verkürzung oder Verlängerung der Zeitspanne zwischen dem Aufbau der Magnetfelder gesteuert werden.

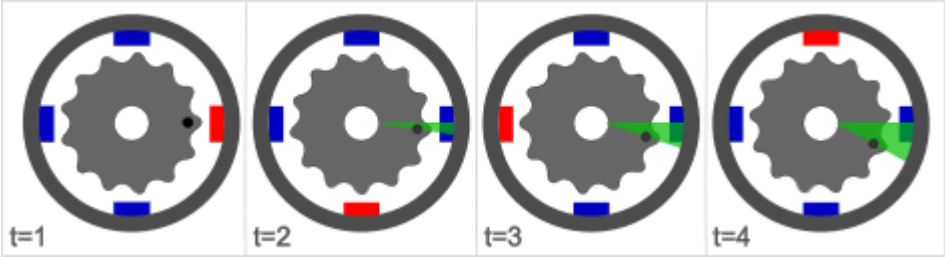

Abbildung 9: Schematischer Aufbau und Funktionsweise eines Schrittmotors

Anwendung in Druckanlagen
Schrittmotoren werden beispielsweise zur Bewegung der Druckeinheit über dem zu bedruckenden Material oder zur Anhebung und Absenkung von Bau- und Pulverräumen verwendet.

2.2.6 Drucksensoren

Ein Drucksensor dient der Erfassung des physikalischen Druckes in einem Medium, wie zum Beispiel in einer Flüssigkeit oder einem Gas, und dessen Abbildung auf eine elektrische Ausgangsgrösse. Je nach ausgenutzten physikalischen Messeffekten unterscheidet man beispielsweise zwischen induktiven, kapazitiven,piezoelektrischen und piezorestriktiven Drucksensoren [23].

Anwendung in Druckanlagen
Drucksensoren spielen beispielsweise bei der Befüllung der Druckkopfdüsen mit Flüssigkeit eine Rolle. Die Befüllung wird durch Überdruck erreicht. Dieser Überdruck darf bestimmte Werte nicht über- bzw. unterschreiten und muss deshalb geregelt werden. Die zur Regelung benötigten Messwerte werden von Drucksensoren geliefert.

2.2.7 Induktive Sensoren

Induktive Sensoren bestehen aus einer Spule, die bei Stromfluss ein elektromagnetisches Feld aussendet. Das Magnetfeld ändert sich, wenn mit elektrisch leitenden Materialien eine Bewegung in Sensornähe hervorgerufen wird. Diese Änderung kann dann in ein elektrisches Signal umgewandelt werden, wodurch eine berührungslose Erkennung von Gegenständen möglich ist.

Anwendung in Druckanlagen
Induktive Sensoren werden in Drucksystemen beispielsweise zum Referenzieren von beweglichen Einheiten verwendet. Sie werden an Stellen platziert,

die prozesskritische Punkte, wie Grund- bzw. Endstellung der Bewegungseinheit, markieren. Dadurch kann festgestellt werden, ob und wann eine solche Markierung passiert wurde, um gegebenenfalls die Bewegung rechtzeitig abbrechen zu können.

2.3 NI Embedded Controller - CompactRio 9014

Das CompactRio 9014 ist ein von National Instruments (NI) hergestelltes Steuer-, Regel- und Datenerfassungssystem. Es umfasst ein Echtzeitbetriebssystem, einen rekonfigurierbaren FPGA-Chip und die Möglichkeit, über Steckplätze unterschiedliche I/O-Module an das System anzuschließen [9]. Die Konfiguration des Systems erfolgt über einen integrierten Ethernet-Anschluss mit Hilfe der LabVIEW Entwicklungsumgebung.

Abbildung 10: NI Embedded Controller: CompactRio 9014 mit vier I/O Modulen [9]

2.3.1 I/O-Module (Input/Output-Module)

Ein I/O-Modul dient der Kommunikation zwischen CompactRio und externer Hardware. National Instruments stellt dazu eine Reihe unterschiedlicher Module zur Verfügung. Hier einige Beispiele:

- Analoge Ein- und Ausgänge mit simultaner Abtastung für 10 V
- Digitale Ein- und Ausgänge mit Ausgangsstromstärken von bis zu 1 A für 24 V
- Differentielle/TTL-Digitaleingänge mit geregeltem 5-V-Netzteilausgang
- Thermoelementeingänge für 80 mV

Die I/O-Module werden je nach Bedarf in die freien Steckplätze des CompactRio eingesetzt. Deren Ein- und Ausgänge können dann über den FPGA ausgelesen bzw. beschrieben werden.

2.3.2 FPGA (Field Programmable Gate Array)

Der FPGA des CompactRio ist ein integrierter Schaltkreis, in dem eine logische Schaltung mit Hilfe der LabVIEW Entwicklungsumgebung programmiert werden kann. Ein FPGA besteht aus Basisblöcken sowie Ein- und Ausgabeblöcken [25], die in Form einer Gitterstruktur angeordnet sind und miteinander konfigurierbar verschaltet werden können (Abb. 11, links). Die Ba-

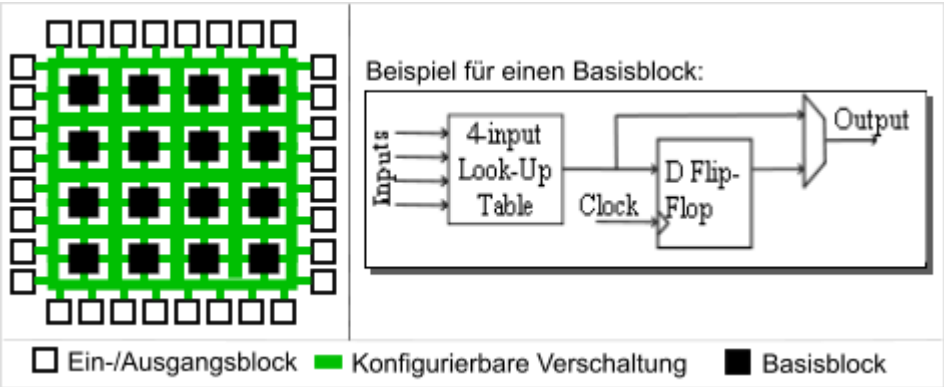

Abbildung 11: Links: Schematischer Aufbau eines FPGA,Rechts: Beispielhafter Basisblock [25]

sisblöcke (Abb. 11, rechts) setzen sich in der Regel aus einer programmierbaren Look-Up Table (LUT) mit einer bestimmten Anzahl von Binäreingängen und einem 1-Bit-Register (Flipflop) zusammen. Je nach Anzahl der Eingänge können damit durch Hinterlegen einer entsprechenden Wahrheitswertetabelle beliebige Binärfunkionen realisiert werden, deren Ausgabe im Flipflop gespeichert wird.
Die konfigurierbare Verschaltung der Basisblöcke untereinander wird durch Multiplexer erreicht, wodurch die Einbindung oder Umgehung der Flipflops, die Rückkopplung von deren Ausgängen und die Verbindung zu Nachbarblöcken kontrolliert werden kann. Der FPGA bietet damit die Möglichkeit Informationen vollkommen parallel zu verarbeiten, da beliebige Anordnungen von digitalen Schaltungsfunktionen realisiert werden können. Jedoch muss für jede Operation ein Stück Hardware zur Verfügung gestellt werden, weshalb die Anzahl der Operationen durch den Bestand an Logikgattern eingeschränkt ist [25]. Durch die Ein- und Ausgangsblöcke des FPGA ist beim CompactRio

die Kommunikation mit den I/O-Modulen und dem Echtzeitbetriebssystem möglich.

2.3.3 Echtzeitbetriebssystem

Die CompactRio Echtzeitumgebung basiert auf einen 400-MHz-MPC5200-Prozessor von Freescale und dem Echtzeitbetriebssystem VxWorks.
Das Echtzeitbetriebssystem garantiert ein vorhersagbares Zeitverhalten, wodurch immer von einer oberen Grenze für die Dauer eines Vorgangs ausgegangen werden kann. Dies ist insbesondere bei zeitkritischen Anwendungen, wie zum Beispiel Regelungs- und Steuerungsalgorithmen, von Bedeutung. Die CompactRio Echtzeitumgebung kann über einen PCI-Bus mit dem FPGA kommunizieren und erlaubt die Ausführung von LabVIEW Anwendungen.

2.4 LabVIEW - Ein graphisches Programmiersystem

LabVIEW ist eine graphische Entwicklungsumgebung von National Instruments, in der die Programmierung mit der visuellen, auf strukturiertem Datenfluss basierenden Programmiersprache ***G*** erfolgt. Ziel von National Instruments war es ursprünglich ein Werkzeug zur Produktivitätssteigerung bei der Automatisierung von Messungen zu erstellen, jedoch hat sich LabVIEW mittlerweile zu einer General Purpose Language (GPL) wie C++ oder Java gewandelt und sich in den Bereichen der Steuer- und Regelungstechnik sowie der Datenerfassung und Datenvisualisierung als Standard etabliert[14]. Die LabVIEW-Entwicklungsumgebung ist für die Betriebssysteme Mac OSX, Windows und Linux erhältlich. Die folgenden Erläuterungen beziehen sich auf die Version „LabVIEW 2009“ für Windows.

2.4.1 Virtuelle Instrumente, Bedienoberfläche und Blockdiagramm

Ein LabVIEW-Programm wird als Virtuelles Instrument (*VI*) bezeichnet, da dessen Aussehen und Funktionalität an reale (Mess-)Geräte erinnern. Jedes *VI* besteht aus einer Bedienoberfläche (auch Frontpanel genannt) und einem Blockdiagramm. In der Bedienoberflächen-Ansicht (Abb. 12) können Bedien-

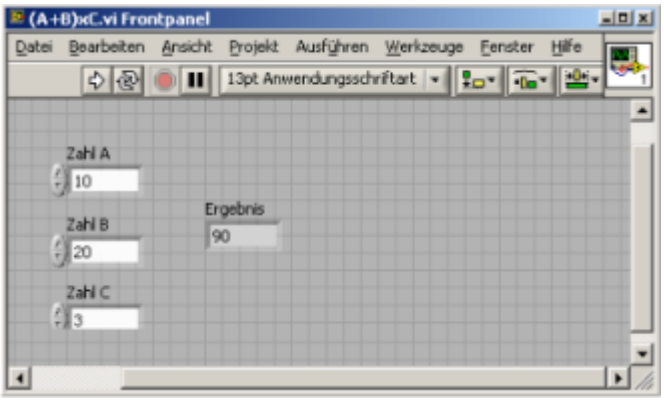

Abbildung 12: (A+B)xC.vi - Bedienoberfläche (Frontpanel)

und Anzeigeelemente platziert werden, auf deren Eigenschaften im Blockdiagramm über repräsentative Symbole lesend bzw. schreibend zugegriffen werden kann. Die Bedienoberfläche enthält also selbst keinen Quellcode, sondern stellt viel mehr eine Art Benutzerschnittstelle dar, über die beispielsweise Ausführungsparameter verändert und Ergebnisse visualisiert werden können. Die eigentliche Programmierung erfolgt in der Blockdiagramm-Ansicht (Abb. 13) durch Einfügen und Verbinden von graphischen Funktionsknoten mit Datenleitungen. In beiden Ansichten stehen Paletten und Menus zur

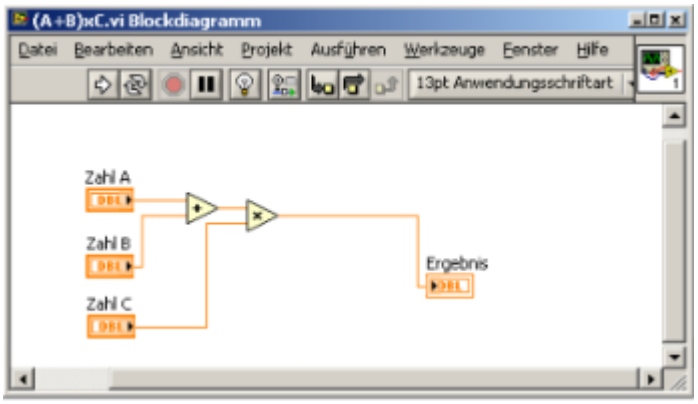

Abbildung 13: (A+B)xC.vi - Blockdiagramm

Verfügung, aus denen Bedienelemente bzw. Funktionsknoten ausgewählt und platziert werden können.

2.4.2 Datenflussprogrammierung und Nebenläufigkeit

LabVIEW Anwendungen werden nach dem Prinzip des strukturierten Datenflusses ausgeführt. Dazu ist der Quellcode innerhalb des Blockdiagramms in Funktionsknoten und Datenleitungen unterteilt.
Als Knoten können alle graphischen Elemente mit Ein- oder Ausgängen angesehen werden wie zum Beispiel mathematische Operatoren, Kontrollstrukturen oder SubVIs (2.4.3). Die Verbindungen zwischen den Knoten stellen die Datenleitungen dar. Ein Funktionsknoten kann dabei erst ausgeführt werden, wenn alle erforderlichen Daten an den Eingängen vorliegen. Liegen an mehreren Knoten gleichzeitig alle benötigten Daten vor, werden diese nebenläufig ausgeführt. Multitasking ist somit eine Grundeigenschaft von LabVIEW.

Beispiel zur Datenflussprogrammierung
Das Prinzip des strukturierten Datenflusses soll durch das *VI* aus Abbildung 12 und Abbildung 13 verdeutlicht werden.
Die Bedienoberfläche dieses *VI*s besteht aus den drei numerischen Bedienelementen `Zahl A`, `Zahl B`, `Zahl C` und dem numerischen Anzeigeelement `Ergebnis`.
Im Blockdiagramm tauchen diese vier Elemente nochmals als Symbole auf, sodass die zuvor in der Bedienoberfläche eingestellten Werte (`Zahl A=10`, `Zahl B=20`, `Zahl C=3`) gelesen bzw. das Endergebnis der Operation in das Anzeigeelement `Ergebnis` geschrieben werden kann.
Bei Programmstart wird zunächst nur der Additionsknoten ausgeführt, da die Werte der Bedienelemente `Zahl A` und `Zahl B` sofort vorhanden sind.

Nach Abarbeitung des Additionsknoten steht auch der zweite Eingang des Multiplikationsknoten zur Verfügung, so dass die Multiplikation mit `Zahl C` ausgeführt und das Ergebnis dem Anzeigelement `Ergebnis` übergeben werden kann. Die Programmausführung endet anschließend.

Beispiel zur Nebenläufigkeit
Ein Beispiel zur Nebenläufigkeit ist im Blockdiagramm von Abbildung 14 zu sehen. Darin befinden sich zwei While-Schleifen, die jeweils den Wert einer numerischen Anzeige mit jedem Schleifendurchlauf vergrößern. Die Schleifen werden nach Programmstart nebenläufig ausgeführt, da die numerische Konstante `0` an ihren Eingänge gleichzeitig anliegt und keine sonstigen Dateneingänge vorhanden sind. Die Ausführung dieses VIs endet theoretisch

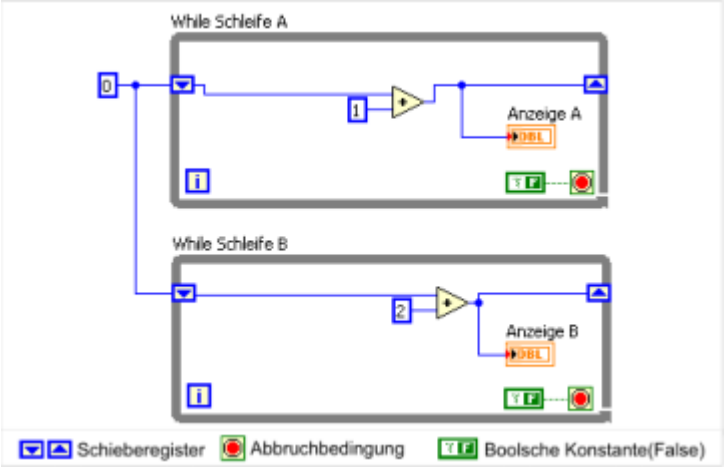

Abbildung 14: Parallele Ausführung zweier While-Schleifen

nie, da die Abbruchbedingung bei jedem Schleifendurchlauf den boolschen Wert `False` erhält.

2.4.3 Unterprogramme (SubVIs)

Jedes *VI* kann prinzipiell in andere *VI*s als selbstgestaltetes graphisches Element eingebunden werden. Dafür müssen bestimmte Frontpanel-Elemente des untergeordneten *VI*s (auch *SubVI* genannt) explizit als Ein- bzw. Ausgänge gekennzeichnet werden. Diese Ein- bzw. Ausgänge dienen dann dem übergeordneten *VI* als Schnittstelle zum *SubVI*.

Beispiel zur SubVI Erstellung
Aus dem VI aus Abbildung 12 bzw. 13 wurde ein SubVI erzeugt, indem die Bedienelemente `Zahl A`,`Zahl B`,`Zahl C` als Eingänge und das Anzeigeelement `Ergebnis` als Ausgang markiert wurden. Dieses SubVI wurde als selbstgestalteter Funktionsknoten in das Blockdiagramm des VIs aus Abbildung 15 eingefügt, um damit die Berechnung `(A+B)xC` durchführen zu können.

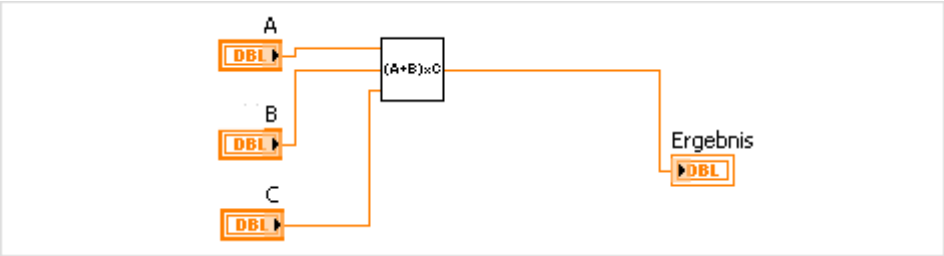

Abbildung 15: Blockdiagramm mit SubVI

Durch die konsequente Verwendung von *SubVI*s und aussagekräftigen *SubVI*-Symbolen lässt sich die Übersichtlichkeit und Verständlichkeit des Quellcodes deutlich steigern, da damit beliebig viele Abstraktionsebenen eingeführt werden können [14].

2.4.4 Compiler, Programmausführung und Debugging

Im Gegensatz zu konventionellen Programmiersprachen ist der LabVIEW-Compiler während der Quellcode-Erstellung ständig aktiv und überprüft dessen Ausführbarkeit. Wurden Fehler im Haupt-VI oder in den verwendeten SubVIs gefunden ist, die Programmausführung nicht möglich. Ansonsten kann über das Pfeilsymbol (Abb. 16, siehe auch 12 und 13) die Anwendung zu Testzwecken direkt gestartet und optional in den Debug-Modus gewechselt werden.

Abbildung 16: Bedienelemente zur Programmausführung

Im Debug-Modus wird der Datenfluss mit Hilfe von sich in Flussrichtung bewegenden Punkten auf den Verbindungsleitungen visualisiert (Abb. 17)

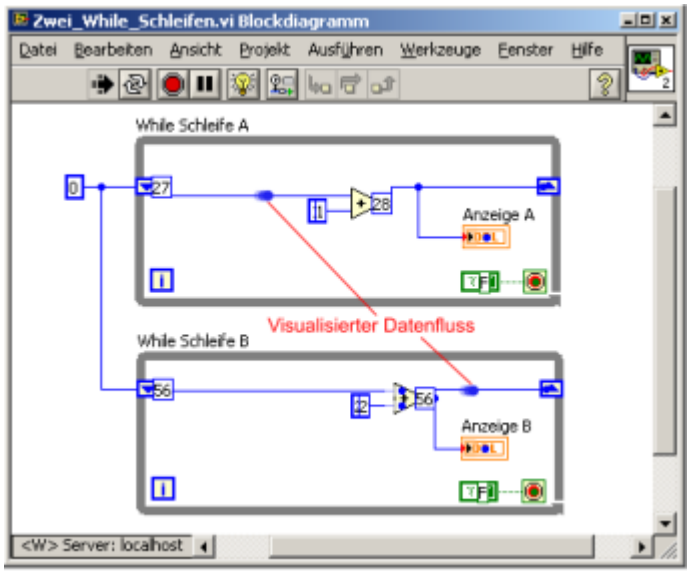

Abbildung 17: Ausführung im Debug-Modus, siehe zum Vergleich ohne Debug-Modus Abb.14

und die Ausgabewerte von Funktionsknoten angezeigt. Die Anwendung wird dabei verlangsamt ausgeführt, um dem Datenfluss folgen zu können.
Auf den Verbindungsleitungen können außerdem Haltepunkte und Sonden angebracht werden.
Haltepunkte pausieren die Programmausführung, immer wenn der Datenfluss diese durchwandert. Dadurch lassen sich zuvor angebrachte Sonden, die anliegende Werte auf den Datenleitungen tabellarisch in einem separaten Fenster anzeigen, zu bestimmten Zeitpunkten der Programmausführung auswerten.
Aus dem Quellcode lässt sich jederzeit eine von der LabVIEW Entwicklungsumgebung unabhängige Anwendung erzeugen.

2.4.5 Datentypen und Datenstrukturen

LabVIEW unterstützt, neben grundlegenden Datentypen wie Boolean, Character, Integer oder Floating Point Number [17], eine Vielzahl weiterer Datentypen und Datenstrukturen, auf die hier nur begrenzt eingegangen werden soll.

Verbund (Cluster)
Ein Verbund von unterschiedlichen Datentypen und Datenstrukturen wird in LabVIEW als Cluster bezeichnet. Damit lassen sich logisch zusammenhängende Datenleitungen zu einer einzigen Datenleitung bündeln (Abb. 18), die

dann gewissermaßen als Datenbus durch ein Blockdiagramm geführt werden kann. Durch die konsequente Verwendung von Clustern lässt sich die Übersichtlichkeit des Quellcodes steigern, da dadurch die Anzahl der Verbindungsleitungen sowie die Anzahl der Ein- und Ausgänge von SubVis minimiert werden kann.

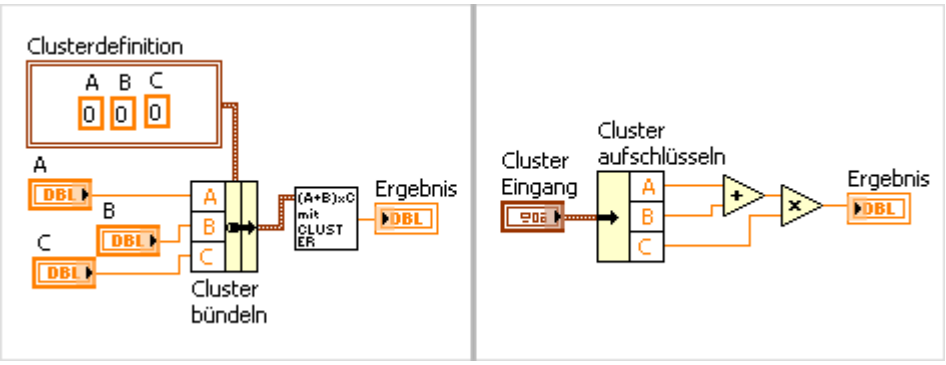

Abbildung 18: Erzeugung und Verwendung eines Clusters, links: Haupt-VI, rechts: SubVI (A+B)xC.vi mit Cluster

Aufzählungstyp (Enum)
Der Datentyp Enum (Enumerated Type) entspricht einer Aufzählung von Elementen (Abb. 19), die jeweils aus einer benutzerdefinierten Bezeichnung und einer eindeutig vergebenen Zahl bestehen. Mit diesem Datentyp lassen sich beispielsweise Auswahllisten zur Programmsteuerung realisieren.

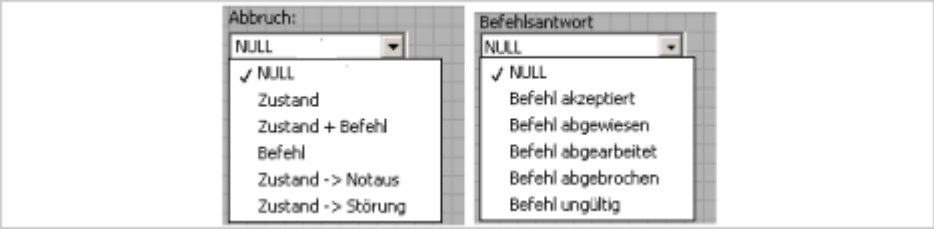

Abbildung 19: Beispiele für den Enum-Datentyp

Typdefinitionen
Mit Hilfe von Typdefinitionen lassen sich Datentypen und der Aufbau von Datenstrukturen festlegen und wiederverwenden. Der Vorteil einer Typdefinition ist, dass diese an mehreren Stellen eines Projektes verwendet werden kann, so dass Änderungen oder Erweiterungen von allen VIs und SubVIs, in denen sie vorkommt, automatisch übernommen werden. Typdefinitionen

werden häufig bei der Definition von Enums und Clustern verwendet.
Wird beispielsweise an mehreren Stellen eines Projektes die selbe Auswahlliste benötigt, erzeugt man eine Typdefinition für den Enum-Datentyp und fügt dort einmalig alle benötigten Listenelemente ein. Aus dieser Typdefinition lassen sich nun an den benötigten Stellen eines Projektes Enum-Datentyp-Elemente erzeugen, die jede Änderung an der Typdefinition übernehmen.

Warteschlangen (Queues)
Wie in vielen anderen Programmiersprachen ist auch in LabVIEW die Verwendung von Warteschlangen möglich. Eine Warteschlange kann eine Menge von Elementen gleichen Datentyps nacheinander aufnehmen und entsprechend dieser Reihenfolge wieder ausgeben (FIFO-Prinzip). In LabVIEW ist es jedoch auch möglich, ein Element an den Anfang einer Warteschlange hinzuzufügen, wodurch die Simulation eines Stapelspeichers möglich ist (LIFO-Prinzip).
Warteschlangen können in LabVIEW beispielsweise zur Kommunikation zwischen nebenläufigen Prozessen verwendet werden.

2.4.6 Kontrollstrukturen und Variablen

LabVIEW unterstützt neben grundlegenden Kontrollstrukturen, wie zum Beispiel CASE-Bedingungen, FOR- und WHILE-Schleifen, auch Einige, die teilweise nur in der graphischen Programmierung Anwendung finden [17]. Im Folgenden finden sich hierzu ein paar Beispiele:

Sequenzen
Da in LabVIEW die sequenzielle Ausführung des Quellcodes nicht immer gewährleistet ist, kann durch die Verwendung von Sequenzstrukturen die definierte Reihenfolge bei der Abarbeitung des graphischen Quellcodes erzwungen werden.

Formelknoten
In einem Formelknoten können umfangreiche Berechnungen in Textform eingefügt werden, wobei eine C-ähnliche Syntax verwendet wird. Daneben ermöglicht LabVIEW die Verwendung von MathScript-Formelnoten, die praktisch vollständig kompatibel zur MATLAB-Syntax sind [4]. Die Kommunikation mit dem Blockdiagramm erfolgt dabei durch Abbildung von graphischen Ein- und Ausgängen des Formelknotens auf Variablen innerhalb des textbasierten Quellcodes.

Event-Struktur
Eine Event-Struktur besteht aus einem oder mehreren Sub-Diagrammen, deren Ausführung an Ereignisse gekoppelt werden kann, die nicht durch den Datenfluss im Blockdiagramm verursacht werden. Ein Ereignis kann beispielsweise das Anklicken eines Bedienelementes oder die Wertänderung einer Variable sein. Event-Strukturen werden häufig in Verbindung mit dem Benutzerinterface verwendet, um auf die Aktionen des Anwenders zu reagieren.

Lokale, globale und verteilte Variablen
In LabVIEW wird zwischen lokalen und globalen Variablen sowie verteilten Variablen, auch Netzwerkvariablen genannt, unterschieden.
Während eine lokale Variable nur innerhalb des VIs, in dem diese erzeugt wurde, zur Verfügung steht, kann auf eine globale Variable auch in allen anderen VIs bzw. SubVIs einer Anwendung lesend und schreibend zugegriffen werden.
Die Verwendung von verteilten Variablen erlaubt die Kommunikation zwischen unterschiedlichen LabVIEW-Anwendungen über eine Netzwerkverbindung.

2.4.7 Zustandsautomaten

Neben der Bedienoberflächen- und Blockdiagramm-Ansicht zur Bearbeitung von VIs, exisitiert in LabVIEW auch eine Zustandsdiagramm-Ansicht, mit der Zustandsautomaten erstellt werden können (Abb. 20).
Die Zustandsdiagramm-Ansicht wurde von National Instruments in Anleh-

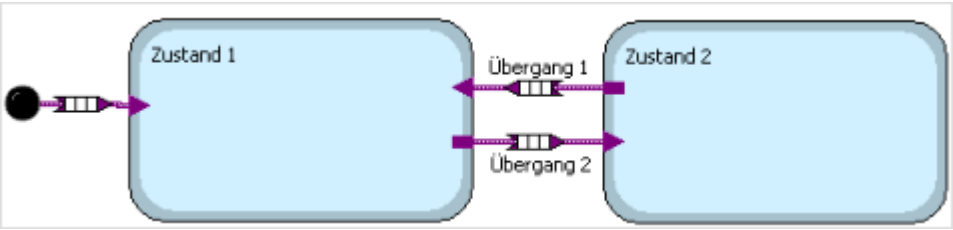

Abbildung 20: Zustandsautomat in der Zustandsdiagramm-Ansicht

nung an die Spezifikationen der „Unified Modeling Language“ (UML) entwickelt [6]. Somit stehen einem zum Aufbau eines Zustandsautomaten dieselbe Elemente zur Verfügung, wie man sie von der Modellierung mit UML-Zustandsdiagrammen her kennt. Das Verhalten mancher[4] dieser Elemente

[4]Das Verhalten von Pseudozuständen [34], wie zum Beispiel Startzustand, Vereinigung und Gabelung, kann nicht angepasst werden.

kann mit Hilfe der grafischen Datenflussprogrammierung definiert werden. Dem Element Zustand können beispielsweise folgende Verhaltensspezifikationen mittels LabVIEW-Code zugeordnet werden:

- `Entry Action`: Verhalten beim Betreten des Zustandes.
- `State Guard`: Bedingung für die Ausführung der `State Action`.
- `State Action`: Verhalten beim Verweilen im Zustand.
- `Exit Action`: Verhalten beim Verlassen des Zustandes.

Dem Element Zustandsübergang:

- `Transition Guard`: Bedingung für den Übergang und dem Ausführen der `Transition Action`.
- `Transition Action`: Verhalten beim Übergang zum nächsten Zustand.

Ausführungsreihenfolge
Beim Betreten eines Zustandes von dem `n` Zustandsübergänge ausgehen, wird zunächst die `Entry Action` des Zustandes ausgeführt (Abb. 21).

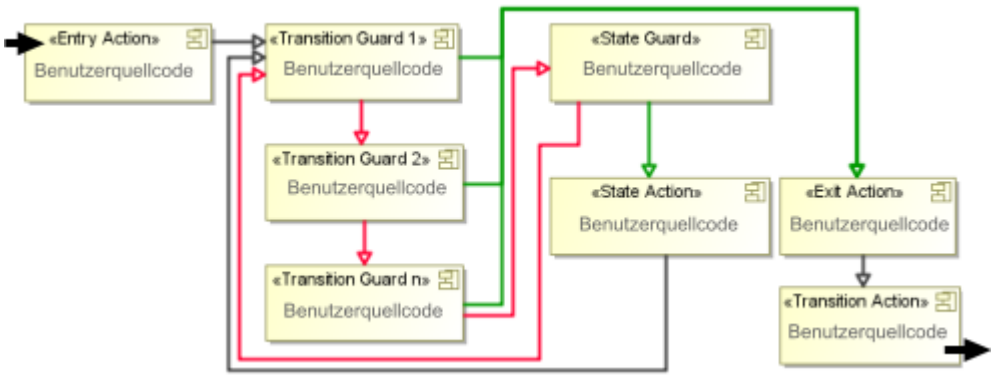

Abbildung 21: Ausführungsreihenfolge der Elemente eines Zustandes mit n Zustandsübergängen.

Anschließend werden die `Transition Guards` der Zustandsübergänge der Reihe nach abgearbeitet, bis entweder ein `Transition Guard` den Wert `TRUE` annimmt oder alle `Transition Guards` mit `FALSE` ausgewertet wurden.
Ist ein `Transition Guard` als `TRUE` ausgewertet, wird der Zustand sofort verlassen, indem die `Exit Action` und anschließend die zum `Transition Guard` zugehörige `Transition Action` ausgeführt werden.

Wurden alle `Transition Guards` mit `FALSE` ausgewertet, kommt es zur Ausführung des `State Guards` und je nachdem, ob dieser den Wert `TRUE` oder `FALSE` annimmt, auch zur Ausführung der `State Action`.
Anschließend werden wieder die `Transition Guards` überprüft und der Zustand entweder verlassen oder dessen `State Guard` bzw. `State Action` erneut ausgeführt.

Einbindung in bestehende VIs
Jeder Zustandsautomat kann, ähnlich wie ein SubVI, in bestehende VIs eingebunden werden (Abb. 22). Die Kommunikation mit dem VI erfolgt über Eingangs- und Ausgangscluster am Zustandsautomaten, auf die beim Bearbeiten von Zuständen und Zustandsübergängen zugegriffen werden kann. Die Datenstruktur dieser Cluster lässt sich über Typdefinitionen vom Anwendungsentwickler anpassen.

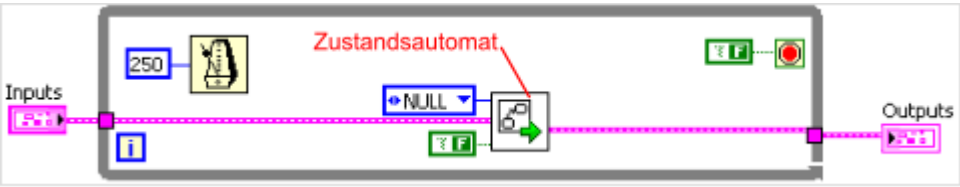

Abbildung 22: Zustandsautomat eingebunden in der While-Schleife eines VIs

2.5 Frameworks

Unter einem Framework versteht man einen „Anwendungsrahmen“, der die Architektur und den Kontrollfluss darauf aufbauender Anwendungen definiert [2]. Das Framework deckt dabei typische Merkmale eines Anwendungsbereiches ab,wodurch dem Anwendungsentwickler ein effizientes Wiederverwendungskonzept zur Verfügung steht, aus dem er durch Anpassungen und Erweiterungen die gewünschte Anwendung erzeugen kann.
Die meisten Frameworks basieren auf dem Paradigma der Objektorientierung, da damit charakteristische Eigenschaften eines Frameworks, wie Modularität und Wiederverwendbarkeit, besonders gut unterstützt werden. Jedoch können diese Eigenschaften prinzipiell auch mit anderen Programmierparadigmen erreicht werden[2].

2.5.1 Merkmale und Eigenschaften

Die Entwicklung von Frameworks ist insbesondere dann sinnvoll, wenn die zu erstellenden Anwendungen einer klar abgegrenzten Domäne angehören und

diese dennoch eine starke Variabilität im Detail aufweisen[2]. Je spezieller dabei der Anwendungsbereich ist, desto größer wird die Wahrscheinlichkeit viele Gemeinsamkeiten zwischen den Anwendungen zu finden, die bei der Framework-Entwicklung zu berücksichtigen sind. Somit kann die Bindung an einem bestimmten Anwendungsbereich als ein typisches Merkmal eines Frameworks gesehen werden.

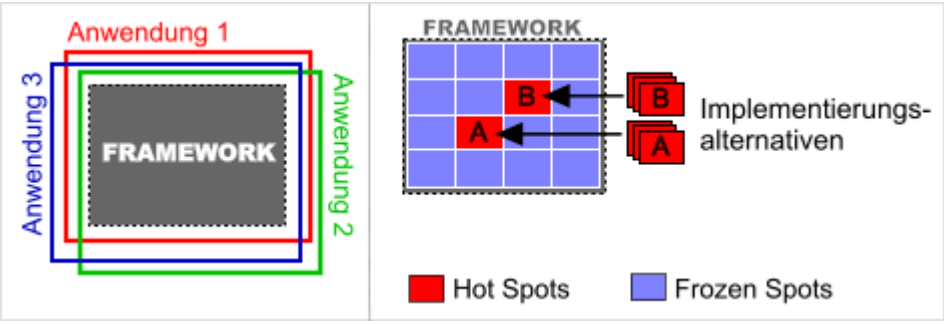

Abbildung 23: Links:Domänenbindung, Rechts: Hot / Frozen Spots

Hot / Frozen Spots
Ein weiteres Merkmal von Frameworks ist deren Aufbau aus invariablen (Frozen Spots) und variablen Bestandteilen (Hot Spots). Die invariablen Bestandteile spiegeln die Gemeinsamkeiten aller Anwendungen einer Domäne wieder und sind Teil jeder aus dem Framework erzeugten Software. Die variablen Bestandteile repräsentieren die möglichen Unterschiede zwischen den einzelnen Anwendungen und müssen vom Entwickler an die Anforderungen der zu erzeugenden Anwendungen angepasst werden.
Das Auffinden dieser variablen und invariablen Bestandteilen erfordert ein tiefgehendes Verständis über den Anwendungsbereich und seine zukünftige Entwicklung.

Invertierter Kontrollfluss
Ziel bei der Entwicklung eines Frameworks ist es eine grösstmögliche Schnittmenge an Gemeinsamkeiten zwischen den Anwendungen einer Domäne zu finden, um diese mit dem Framework abzudecken. Diese Schnittmenge bezieht sich dabei nicht nur auf die Funktionalität, sondern auch auf einen gemeinsamen Kontrollfluss der Anwendungen. Die Umkehrung des Kontrollflusses ist deshalb ein weiteres typisches Merkmal von Frameworks. Bei der Anwendungsentwicklung mit Bibliotheken steuert der Entwickler beispielsweise selbst den globalen Kontrollfluss und greift nur an bestimmten Stellen auf die Funktionen der Bibliothek zu. Im Gegensatz dazu rufen bei einem

Abbildung 24: Invertierter Kontrollfluss in Frameworks

Framework die invarianten Bestandteile den Quellcode des Entwicklers auf, wodurch der globale Kontrollflusses größtenteils durch das Framework selbst definiert wird.

2.5.2 Vor- und Nachteile

- *Zeit- und Kostenersparnis:* Durch die Wiederverwendung eines bewährten Frameworks sinkt die Entwicklungszeit der Anwendung, da beispielsweise weniger anwendungspezifischer Quellcode geschrieben werden muss. Das führt in der Arbeitswelt zu Kostenersparnis [1].

- *Kapselung von Domänenwissen:* Durch die Auslegung auf einen bestimmten Anwendungsbereich, ist das Wissen über diese Domäne im Framework gekapselt. Der Anwendungsentwickler muss sich dadurch nicht mehr mit der Komplexität der Domäne auseinandersetzten [2].

- *Verbesserte Wartbarkeit:* Die gemeinsame Codebasis mehrerer Anwendungen hat eine verbesserte Wartbarkeit zur Folge, da die Anwendungen nicht unabhängig voneinander betrachtet werden müssen[2].

- *Einarbeitungszeit:* Die korrekte und effiziente Benutzung eines Frameworks erfordert eine gewisse Einarbeitungszeit und eine mehrmalige Anwendung des Frameworks [1].

- *Fehleranfälligkeit:* Fehler im Framework wirken sich auf alle daraus erzeugten Anwendungen aus. Deshalb muss das Framework vor der Verwendung ausreichend getestet werden [1].

3 Anforderungsanalyse und Entwurf des Frameworks

Die Entwicklung eines Frameworks zur Steuerung von prototypischen 3D-Ink-Jet-Druckanlagen erfordert ein tiefgehendes Verständnis dieses Anwendungsbereiches [3], um daraus die variablen und invariablen Bestandteile des Frameworks extrahieren zu können.
Aus diesem Grund soll zunächst der Aufbau und die Funktion einer prototypischen Druckanlage beschrieben und die geplante Verwendung durch einen Benutzer aufgezeigt werden. Diese Analyse dient dann als Grundlage für die Definition der Anforderungen an das zu entwickelnde Framework.

3.1 Analyse einer prototypischen Keramikdruckanlage

Bei der im Folgenden beschriebenen Druckanlage handelt es sich um einen Versuchsstand zur Herstellung von keramischen Bauteilen, der am Fraunhofer IPA[5] entstanden ist. Für einen möglichen Einsatz in der Herstellung von Zahnersatz, sollen an dieser Anlage verschiedene Bindemittel und Keramikpulver getestet werden.

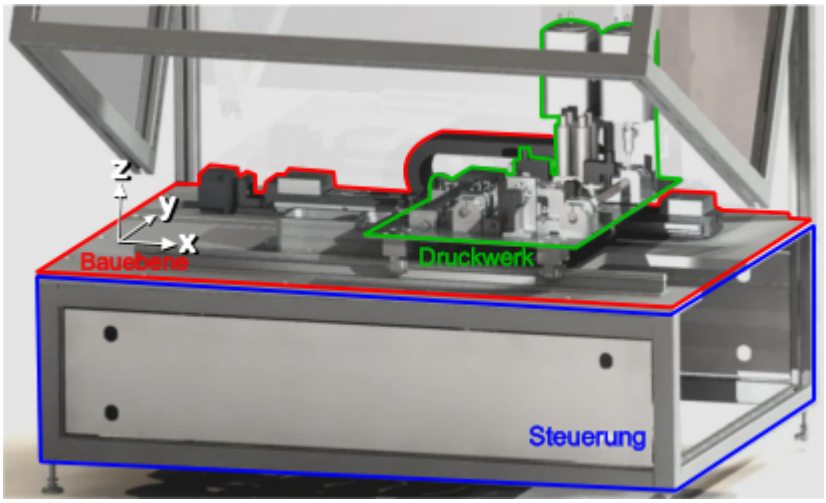

Abbildung 25: Keramikdruckanlage mit Einteilung in drei Bereiche.

[5]Fraunhofer Institut für Produktionstechnik und Automatisierung

3.1.1 Aufbau und Funktion

Der Aufbau dieser Druckanlage (Abb. 25) kann in ein Druckwerk, eine Bauebene und der Hardware zur Steuerung und Signalanpassung eingeteilt werden.

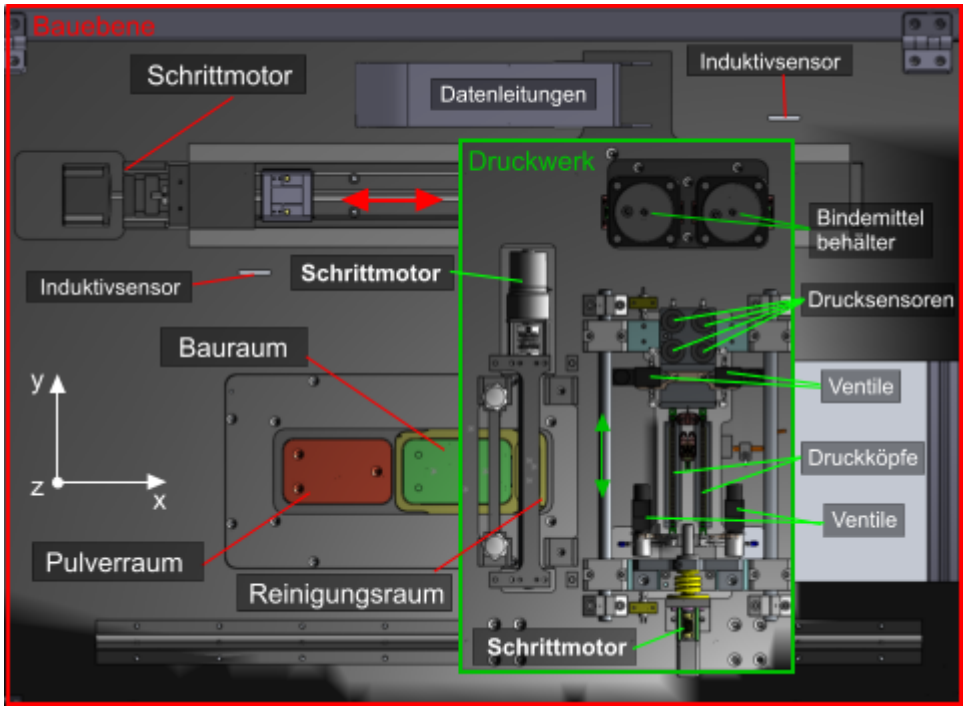

Abbildung 26: Aufbau des Druckwerks und der Bauebene.

Druckwerk
Das Druckwerk besteht aus zwei Druckköpfen und zwei Bindemittelbehältern (Abb. 25, Abb. 26, grüner Bereich), an denen Heizelemente und Temperatursensoren angebracht sind, damit die Druckköpfe auf bestimmte Temperaturen erwärmt werden können. Eine Temperierung der Druckköpfe und Behälter wird benötigt, um:

- Mehr Bindemittel verdrucken zu können: Ink-Jet-Druckköpfe können nicht mit beliebig zähen Flüssigkeiten operieren. Da die Viskosität von Flüssigkeiten jedoch stark temperaturabhängig ist [33], kann durch eine Temperaturerhöhung die Palette an druckbaren Flüssigkeiten erweitert werden.

- Temperaturschwankungen auszugleichen: Die Umgebungstemperatur der Druckanlage kann von dem Ort und von der Jahreszeit abhän-

gen. Um trotzdem konstante Temperaturen für den Druckprozess zu gewähren, ist eine Temperaturregelung notwendig.

Jeder Druckkopf ist mit einem Bindemittelbehälter über ein System[6] von Schläuchen, Drucksensoren, Ventilen und Pumpen miteinander verbunden [21]. Dieses System dient der Erzeugung von regelbarem Unter- bzw. Überdruck, um beispielsweise:

- Die Druckköpfe zu reinigen: Bindemittel wird mit starkem Überdruck durch die Düsen gepresst, damit Verstopfungen entfernt werden.
- Die Düsen zu befüllen: Bindemittel wird mit leichtem Überdruck durch die Düsen gepresst, bis es zur Tropfenbildung an den Düsenöffnungen kommt.
- Die Tropfenbildung zu verhindern: Nach der Befüllung der Düsen verhindert leicher Unterdruck ein weiteres Tropfen der Düsen.
- Das Bindemittel zu entgasen: Unterdruck entfernt Luftblasen aus Schläuchen und Flüssigkeit.Dabei ist die Verbindung zu den Druckköpfen gesperrt, damit keine Luft durch die Düsenöffnungen angesaugt werden kann.

Um eine höhere Druckauflösung zu ermöglichen, können die Druckköpfe mit Hilfe eines Schrittmotors minimal in y-Richtung bewegt werden. Dazu sind die Druckköpfe auf Schienen direkt über eine Öffnung zur Bauebene angebracht.
Über einen weiteren Schrittmotor wird ein Mechanismus zum Auftragen von Pulverschichten betrieben. Der genaue Aufbau dieses Mechanismus war jedoch zum Zeitpunkt dieser Ausarbeitung noch unklar.

Bauebene
Das Druckwerk ist in der Bauebene (Abb. 25, Abb. 26, roter Bereich) auf zwei Schienen angebracht und kann mit Hilfe eines Schrittmotors in x-Richtung über Bau-, Pulver- und Reinigungsraum bewegt werden. Bau- und Pulverraum bestehen aus Plattformen, die sich mit unter der Bauebene befindlichen Schrittmotoren in z-Richtung verfahren lassen. Diese Räume erfüllen folgende Zwecke (2.1.2):

- Im Bauraum wird das keramische Bauteil durch das Bedrucken von Pulverschichten erzeugt.

[6] Aus Platzgründen befindet sich ein Teil dieses Systems unter der Bauebene. Schlauchverbindungen zwischen Druckköpfen und Bindemittelbehältern sind in Abbildung 26 nicht eingezeichnet.

- Der Pulverraum enthält das für den Druckprozess benötigte Pulver.
- Der Reinigungsraum dient als Auffangbehälter für Flüssigkeiten und Pulver, die während des Druckprozesses bzw. bei der Reinigung der Druckköpfe anfallen.

Zur Referenzierung und Bewegungsbegrenzung des Druckwerks und der Plattformen sind auf und unter der Bauebene Induktivsensoren angebracht.

Embedded Controller und Hardware zur Signalanpasung
Unter der Bauebene (Abb.25,blauer Bereich) befindet sich ein CompactRio 9014 Embedded Controller von National Instruments zur Steuerung der Druckanlage. Die I/O Module des Embedded Controllers sind über eine Hardware zur Signalanpassung mit allen Druckköpfen, Sensoren und Aktoren der Druckanlage verbunden.
Diese Signalanpassung wird benötigt, um die Ausgänge der Sensoren und die Eingänge der Aktoren mit dem Spannungsbereich der I/O Module kompatibel zu machen.

3.1.2 Bedienung der Druckanlage

Die Bedienung der Druckanlage soll mit Hilfe eines Benutzerinterfaces erfolgen. Dazu ist der Embedded Controller über eine Ethernet-Verbindung an einen PC angeschlossen. Neben der Visualisierung von Sensordaten und dem aktuellen Zustand der Druckanlage, sollte das Benutzerinterface auch folgende Funktionen erfüllen:

- Zustand der Druckanlage verändern: Der Benutzer sollte die Druckanlage in diverse Zustände versetzen können. Er muss beispielsweise Bau- und Pulverraum anheben und absenken oder den Druckprozess starten können.
- Laufende Prozesse abbrechen: Autonom ablaufende Prozesse sollten aus Sicherheitsgründen jederzeit abgebrochen werden können.
- Regelparameter ändern: Da mit der Druckanlage verschiedene Bindemittel getestet werden sollen, müssen die Parameter der Temperatur- und Druckregelungen vom Benutzer für unterschiedliche Bindemittel angepasst werden können.

3.2 Anforderungen an das Framework

Folgende Anforderungen konnten nach der Analyse der Keramikdruckanlage in Zusammenarbeit mit den Mitarbeitern am Fraunhofer IPA ermittelt werden.

3.2.1 Funktionale Anforderungen

Die im Folgenden beschriebenen funktionalen Anforderungen spiegeln die Gemeinsamkeiten zwischen unterschiedlichen Ink-Jet-Druckanlagen wider und sind deshalb fester Bestandteil jeder Steuerungsanwendung. Sie entsprechen den Frozen Spots des Frameworks:

- Umsetzung mit LabVIEW: Zur Steuerung der Druckanlagen sollen Embedded Controller von National Instruments verwendet werden (Referenz Kapitel Grundlage), weshalb die Umsetzung des Frameworks mit LabVIEW geschehen muss.

- Kommunikation zwischen PC und Embedded Controller: Der Datenaustausch zwischen PC und dem Embedded Controller ist zur Bedienung einer Druckanlage über ein Benutzerinterface unbedingt notwendig, weshalb mit dem Framework eine Kommunikation zwischen diesen beiden realisiert muss.

- Benutzerinterface: Ein Benutzerinterface wird zur Bedienung jeder Druckanlage benötigt, dessen Aufbau sollte deshalb durch das Framework unterstützt werden.

- Zustandsautomat: Ein Zustandsautomat wird als Modell für das Verhalten einer Druckanlage in jeder Steuerungsanwendung benötigt. Er muss auf bestimmte Eingaben des Benutzers mit Zustandsübergängen reagieren, Informationen zu seinem aktuellen Zustand bereitstellen und jederzeit die Möglichkeit bieten, autonom ablaufende Prozesse abbrechen zu können.

- Druckkopfansteuerung: Für die Druckanlagen sollen Ink-Jet-Druckköpfe verwendet werden, deren Druckkopfdüsen in Abhängigkeit eines Schieberegisters zum Feuern gebracht werden können. Mit dem Framework muss deshalb eine Ansteuerung solcher Druckköpfe realisiert werden.

- Schrittmotorsteuerung: Schrittmotoren sollen in jeder Druckanlage zur Bewegung von Elementen verwendet werden, da durch das Ausführen einer minimalen Bewegung pro Schritt zum Beispiel ein Rückschluss auf die aktuelle Position eines Elementes möglich ist.

- Unter- und Überdruckregelung: Eine Druckregelung wird in jeder Druckanlage zum Betreiben von Ink-Jet-Druckköpfen benötigt (3.1.1).
- Temperaturregelung: Eine Temperaturregelung wird benötigt, um zum Beispiel konstante Temperaturen für den Druckprozess gewähren zu können (3.1.1).

3.2.2 Flexibilitätsanforderungen

Die im Anschluss beschriebenen Flexibilitätsanforderungen sollen als Variabilitäten (Hot Spots) im Framework berücksichtigt werden. Sie zeigen die möglichen Unterschiede zwischen prototypischen 3D-Ink-Jet-Druckanlagen auf.

- Anzahl der Druckköpfe und Bindemittelbehälter: Die Anzahl an Druckköpfen und Bindemittelbehältern kann von Druckanlage zu Druckanlage unterschiedlich sein. Deshalb muss die Druckkopfansteuerung sowie die Temperatur- und Druckregelung auf mehrere Druckköpfe bzw. Bindemittelbehälter anwendbar sein.
- Anzahl der fahrbaren Elementen: Die Schrittmotorsteuerung muss auf mehrere Schrittmotoren anwendbar sein, da die Anzahl der fahrbaren Elemente in Abhängigkeit der jeweiligen Druckanlage unterschiedlich sein kann.
- Anzahl der Sensoren und Aktoren: Neben den Druckköpfen und Schrittmotoren ist die Anzahl weiterer Aktoren und Sensoren zum Beispiel abhängig von den benötigten Temperatur- und Druckregelungen. Deshalb muss mit dem Framework der Zugriff auf eine unbekannte Anzahl an Aktoren und Sensoren möglich sein.
- Verhalten des Zustandsautomaten: Das Verhalten des Zustandsautomaten muss an jede Druckanlage angepasst werden können.
- Funktionalität des Benutzerinterfaces: Da mit dem Benutzerinterface die Druckanlage bedient werden soll, ist seine Funktionalität abhängig vom Verhalten des Zustandsautomaten und muss somit individuell für jede Druckanlage angepasst werden können.

3.3 Entwurf des Frameworks

In Abhängigkeit der zuvor beschriebenen Anforderungen wurde ein Framework zur Steuerung von prototypischen 3D-Ink-Jet-Druckanlagen entworfen. Das Framework ist auf drei Ausführungsumgebungen verteilt (Abb. 27):

- PC mit LabVIEW-kompatiblem Betriebssystem (Abb.28, PC)
- Echtzeitbetriebssystem des Embedded Controllers (Abb.29, RT: Real Time)
- FPGA des Embedded Controller (Abb.29, FPGA)

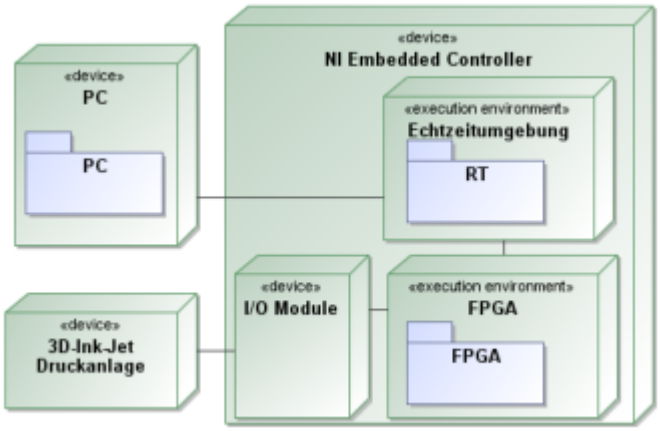

Abbildung 27: Übersicht: Ausführungsumgebungen des Frameworks

Im Folgenden wird auf die einzelnen Umgebungen mit den Abkürzungen PC, RT und FPGA Bezug genommen.

Die Komponenten der PC- und RT-Ausführungsumgebung wurden als nebenläufige Prozesse entworfen, die auf eine Menge von Queues lesend bzw. schreibend zugreifen können. Diese Queues können in `PC-Queues` und `RT Queues` eingeteilt werden.

PC-Queues
Die Elemente der `PC-Queues` entsprechen den Daten die der PC an den RT sendet, zum Beispiel:

- Übergangsbefehle an den Zustandsautomaten
- Abbruchbefehle an den Zustandsautomaten
- Stellwerte für die Regelungen

RT-Queues
Die Elemente der `RT-Queues` umfassen alle Daten, die vom RT zum PC gesendet und zur Kommunikation zwischen den einzelnen Komponenten der RT-Ausführungsumgebung benötigt werden, zum Beispiel:

- Statusmeldungen des Zustandsautomaten
- Befehlsantworten als Reaktion des Zustandsautomaten auf gesendete Übergangsbefehle
- Stellwerte des Zustandsautomaten für die Regelungen
- Messwerte
- RT-Daten

Im Folgenden sollen die Komponenten der PC-,RT- und FPGA-Ausführungsumgebung näher erläutert werden.

3.3.1 Komponenten der PC-Ausführungsumgebung

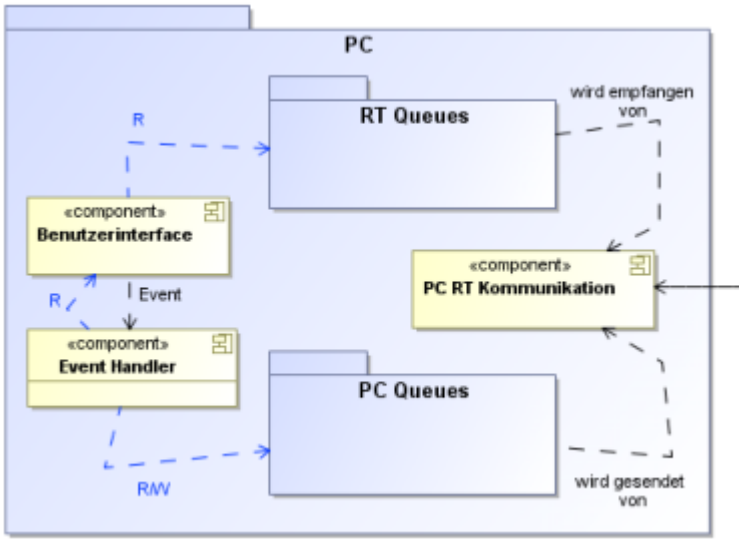

Abbildung 28: Komponenten der PC-Ausführungsumgebung

PC RT Kommunikation
Diese Komponente ist für den Datenaustausch zwischen PC und RT verantwortlich. Sie empfängt Elemente der `RT-Queues` vom RT und sendet Elemente der `PC-Queues` an den RT.

Benutzerinterface
Das Benutzerinterface stellt Bedien- und Anzeigeelemente zur Verfügung. Während die Anzeigeelemente bestimmte Daten aus den `RT-Queues` visualisieren, lösen bestimmte Bedienelemente Events aus, die vom Eventhandler verarbeitet werden.

Eventhandler
Der Eventhandler reagiert auf bestimmte Events beispielsweise durch Einfügen von Übergangsbefehlen, Abbruchbefehlen oder Stellwerten in die `PC-Queues`. Dazu kann der Eventhandler die in den Bedienelementen des Benutzerinterfaces eingestellten Werte (zum Beispiel Stellwerte) auslesen, um sie an die `PC-Queues` weitergeben zu können.

3.3.2 Komponenten der RT-Ausführungsumgebung

PC RT Kommunikation
Diese Komponente ist für den Datenaustausch zwischen PC und RT verantwortlich. Sie empfängt Elemente der `PC-Queues` vom PC und sendet Elemente der `RT-Queues` an den PC.

Temperaturregelung
Die Temperierung eines Bindemittelbehälters oder eines Druckkopfes wird mit dieser Komponente umgesetzt. Sie hat über die `RT-Queues` Zugriff auf die aktuellen Messwerte und Stellwerte und kann Heizelemente über den FPGA ein- bzw. ausschalten.

Unter- und Überdruckregelung
Mit dieser Komponente wird die Druckregelung zwischen einem Bindemittelbehälter und einem Druckkopf realisiert. Sie hat über die `RT-Queues` Zugriff auf die aktuellen Messwerte der Drucksensoren und kann Ventile und Pumpen über die FPGA-Eingänge so bedienen, dass es zur Erzeugung von Unter- und Überdruck kommt.

Zustandsautomat
Der Zustandsautomat repräsentiert das Verhalten der Druckanlage. Er reagiert auf Übergangsbefehle und Abbruchbefehle mit einem Zustandswechsel und aktualisiert dabei die `RT-Queues` mit seinem aktuellen Zustand.

Sensordaten lesen
Diese Komponente fragt in regelmäßigen Zeitabständen die aktuellen Sensordaten von den FPGA-Ausgängen ab und fügt sie in `RT-Queues` ein.

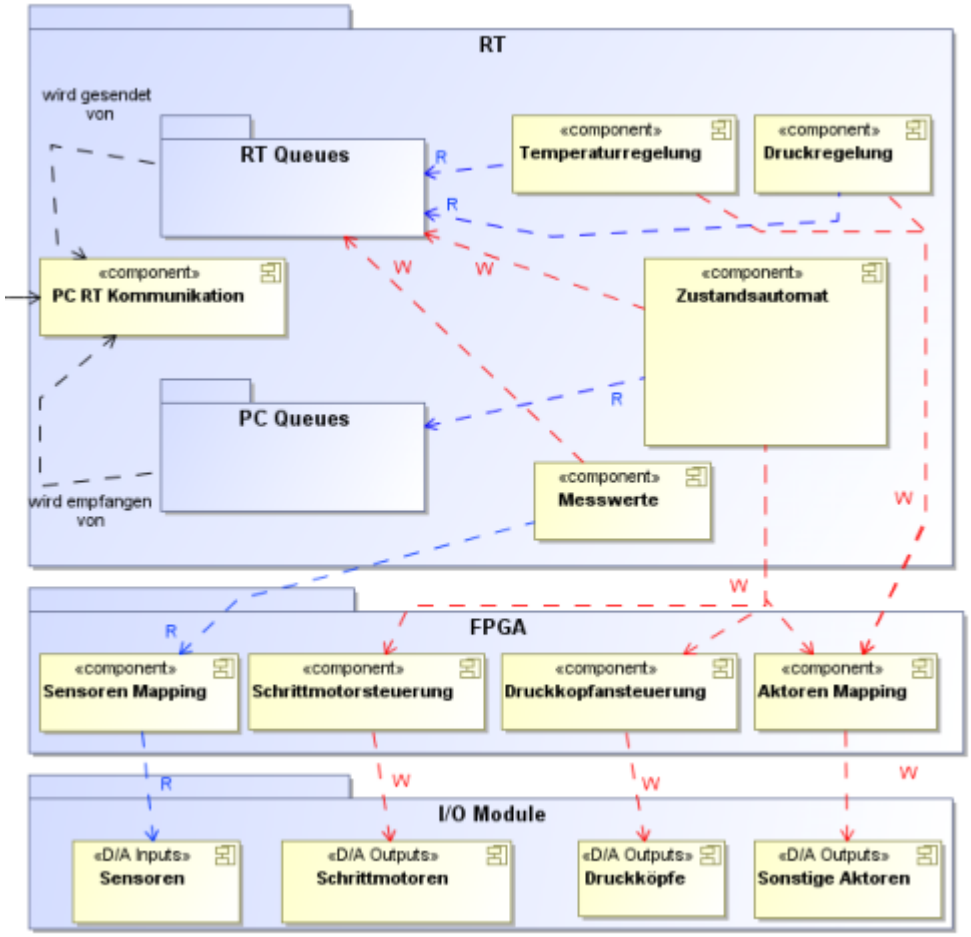

Abbildung 29: Komponenten der RT- und FPGA-Ausführungsumgebung

3.3.3 Komponenten der FPGA-Ausführungsumgebung

Sensoren Mapping
Die RT-Ausführungumgebung kann nicht direkt auf die Eingänge der I/O Module zugreifen [9]. Diese Komponente stellt dem RT deshalb die Sensordaten als Ausgänge des FPGA zur Verfügung.

Aktoren Mapping
Die RT-Ausführungumgebung kann die Ausgänge der I/O Module nicht direkt beschreiben[9], weshalb diese Komponente dem RT einen Zugriff auf die Aktoren über die FPGA-Eingänge ermöglicht. Die I/O Modul Ausgänge zu den Schrittmotoren und Druckköpfen sollten davon ausgenommen sein, da diese nur über die entsprechenden FPGA-Komponenten angesteuert werden sollten.

Schrittmotorsteuerung
Mit dieser Komponente wird die Steuerung eines Schrittmotors realisiert. Sie stellt FPGA-Eingänge zur Verfügung, über die der Motor von der RT-Ausführungsumgebung aus gestartet werden kann.

Druckkopfansteuerung
Mit dieser Komponente wird die Ansteuerung der Druckköpfe realisiert. Sie stellt FPGA-Eingänge zur Verfügung, über die das Feuern der Druckkopfdüsen in Abhängigkeit von Druckdaten von der RT-Ausführungsumgebung aus gestartet werden kann.

4 Umsetzung des Frameworks mit LabVIEW

In diesem Kapitel wird die Umsetzung des Frameworks mit LabVIEW aufgezeigt. Es werden zunächst die Datentypen und Datenstrukturen des Frameworks und die Kommunikation zwischen den Ausführungsumgebungen vorgestellt. Anschließend wird die Umsetzung des Zustandsautomaten, des Benutzerinterfaces und der Komponenten zur Steuerung und Regelung erläutert.

4.1 Übersicht

Ausgangspunkt für die Umsetzung des Frameworks und die spätere Verwendung durch einen Anwendungsentwickler ist die LabVIEW-Projektübersicht. Die Projektübersicht entspricht einer Ordnerstruktur, die in Abhängigkeit der drei Ausführungsumgebungen unterteilt ist (Abb. 30) und den Zugriff auf alle VIs des Frameworks ermöglicht.

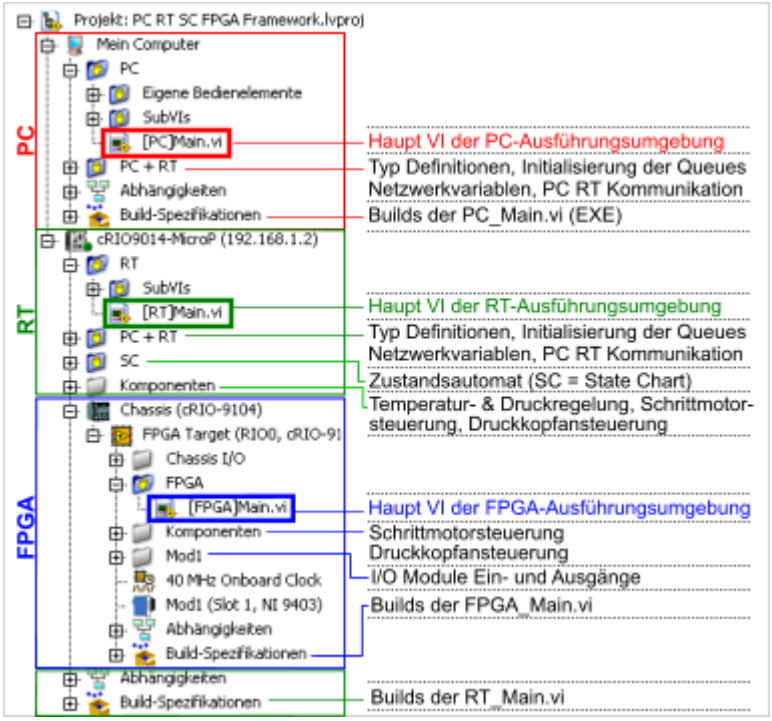

Abbildung 30: LabVIEW-Projektübersicht des Frameworks

Jedem dieser drei Bereiche wurde ein Haupt-VI zugewiesen, das die Ausführung der restlichen VIs der jeweiligen Ausführungsumgebung koordiniert:

- In der `[PC]Main.vi` sind die Komponenten der PC-Ausführungsumgebung enthalten.
- Die `[RT]Main.vi` enthält alle SubVIs der RT-Ausführungsumgebung. Sie wird zur Ausführung durch die LabVIEW-Entwicklungsumgebung auf das Echtzeitbetriebssystem des CompactRio 9014 übertragen.
- Die `[FPGA]Main.vi` entspricht der FPGA Funktionalität. Die LabVIEW-Entwicklungsumgebung generiert daraus eine Bitfile und konfiguriert damit den FPGA. Ein- und Ausgänge der `[FPGA]Main.vi` entsprechen dann den Ein- und Ausgängen des FPGA, auf die über eine Referenz zur `[FPGA]Main.vi` von der RT-Ausführungsumgebung aus zugegriffen werden kann (4.7).

Im Folgenden wird die Umsetzung der einzelnen Komponenten mit LabVIEW näher erläutert.

4.2 Initialisierung der Queues

Die Initialisierung der Framework-Queues in den beiden Haupt-VIs der PC- und RT-Ausführungsumgebung wurde mit dem VI `[PCRT]Queues_Init.vi` realisiert. Dieses VI wird einmalig ausgeführt und erzeugt ein Cluster von Queue-Referenzen (Abb. 33), die an den einzelnen Komponenten der PC- und RT-Ausführungsumgebung weitergereicht werden.
Grundlage für die Initialisierung der Queues sind einige Typdefinitionen (TD), die den Datentypen bzw. Datenstrukturen des Frameworks entsprechen.

4.2.1 Typdefinitionen zur Kommunikation mit dem Zustandsautomaten

Die im Folgenden vorgestellten Typdefintionen dienen der Kommunikation mit dem Zustandsautomaten (4.4). Die Abhängigkeit der PC- bzw. RT-Queues von diesen Typdefinitionen bzw. Cluster-Typdefinitionen (CTD) ist in Abbildung 31 zu sehen.

TD_Übergangsbefehl
Die Typdefinition `TD_Übergangsbefehl` (Datentyp Enum) besteht aus einer Menge von Bezeichnungen, die Übergangsbefehle repräsentieren sollen, auf die der Zustandsautomat mit einem Zustandsübergang reagieren kann (4.4.3). Sie muss vom Anwendungsentwickler mit den benötigten Übergangsbefehlen für den Aufbau des Zustandsautomaten ergänzt werden (5.3).

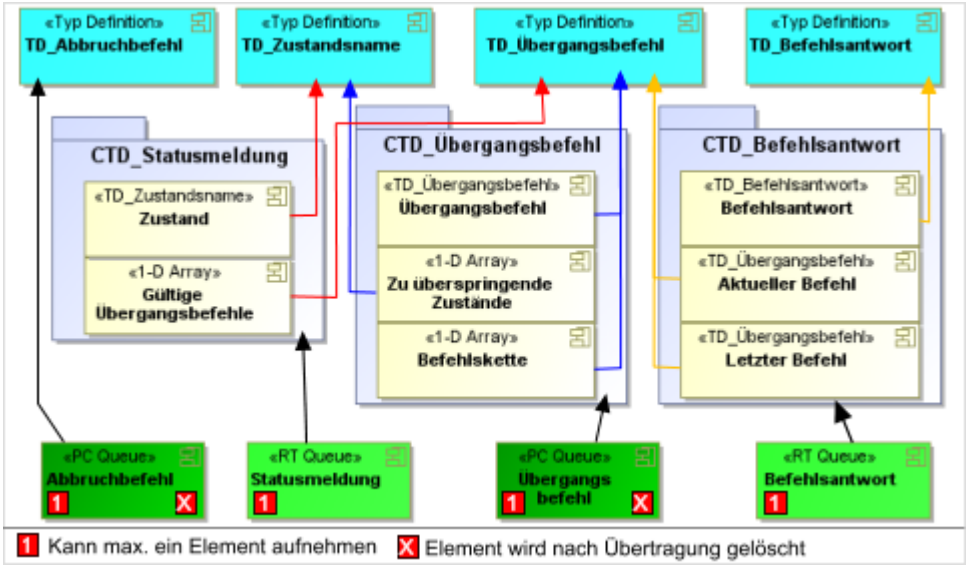

Abbildung 31: Abhängigkeit der Queues von den Typdefinitionen

TD_Zustandsname
Die Typdefinition `TD_Zustandsname` (Datentyp Enum) besteht aus einer Menge von Bezeichnungen, die der Benennung von Zuständen des Zustandsautomaten dient (4.4). Sie muss vom Anwendungsentwickler mit den benötigten Zustandsnamen für den Aufbau des Zustandsautomaten ergänzt werden (5.3).

TD_Befehlsantwort
Die Typdefinition `TD_Befehlsantwort` (Datentyp Enum) besteht aus einer Menge von Bezeichnungen (Tab. 1), die als Antwort auf übermittelte Übergangsbefehle vom Zustandsautomaten an die PC-Ausführungsumgebung gesendet werden können (4.4.5).

Befehl akzeptiert	Befehl abgewiesen	Befehl abgearbeitet
Befehl abgebrochen	Befehl ungültig	

Tabelle 1: Elemente der Typdefintion TD_Befehlsantwort

TD_Abbruchbefehl - Datentyp der PC-Queue Abbruchbefehl
Die Typdefinition `TD_Abbruchbefehl` (Datentyp Enum) besteht aus einer Menge von Bezeichnungen (Tab. 2), die dem Senden von Abbruchbefehlen an den Zustandsautomaten dient (4.4.4).

Zustand abbrechen	Befehl abbrechen	Zustand + Befehl
Notaus	Störung	

Tabelle 2: Elemente der Typdefintion TD_Abbruchbefehl

CTD_Übergangsbefehl - Datenstrukt. der PC-Queue Übergangsbefehl
Diese Cluster-Typdefinition dient dem Senden von Übergangsbefehlen an den Zustandsautomaten. Neben einem einzelnen Übergangsbefehl kann sie auch eine Befehlskette sowie eine Menge an zu überspringenden Zuständen aufnehmen.

CTD_Statusmeldung - Datenstruktur der RT-Queue Statusmeldung
Diese Cluster-Typdefinition repräsentiert die Statusmeldung, die der Zustandsautomat an den PC sendet. Sie besteht aus dem aktuellen Zustandsnamen und einem Array von Übergangsbefehlen, auf die der aktuelle Zustand mit einem Übergang reagieren kann (4.4.1).

CTD_Befehlsantwort - Datenstruktur der RT-Queue Befehlsantwort
Neben der Befehlsantwort, repräsentiert diese Cluster-Typdefinition den letzten und den aktuell vom Zustandsautomaten ausgeführten Befehl. Sie wird als Antwort auf gesendete Übergangsbefehle vom Zustandsautomaten an den PC gesendet.

4.2.2 Sonstige Typdefinitionen

Die folgenden Typdefinitionen und die damit initialisierten Queues dienen der Übermittlung von allgemeinen Daten, Stellwerten, Messwerten und Druckdaten zwischen den Komponenten der PC- und RT-Ausführungungsumgebung. Sie müssen vom Anwendungsentwickler an die vorhandenen Sensoren und Aktoren sowie den benötigten Druck- bzw. Temperaturregelungen der Druckanlage angepasst werden. Jede Cluster-Typdefinition kann dabei eine Menge unterschiedlicher Datentypen aufnehmen.

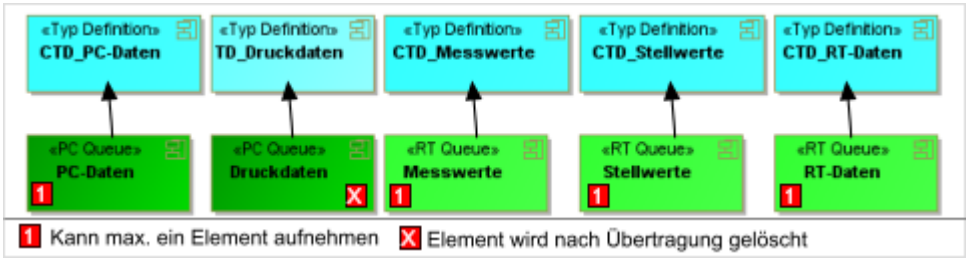

Abbildung 32: Abhängigkeit der Queues von den Typdefinitionen

TD_Druckdaten - Datentyp der PC-Queue Druckdaten
Mit dieser Typdefinition wird ein Array aus 32 64-Bit-Integerwerten definiert, das die zu druckenden Daten repräsentiert (4.11).

CTD_PC-Daten - Datenstruktur der PC-Queue PC-Daten
Diese Cluster-Typdefinition dient der Übermittlung von allgemeinen Daten an den RT. Sie kann beispielsweise zum Versenden von Stellwerten für die Temperatur- und Druckregelung verwendet werden.

CTD_RT-Daten - Datenstruktur der RT-Queue RT-Daten
Diese Cluster-Typdefinition dient der Übermittlung von allgemeinen Daten an den PC.

CTD_Stellwerte - Datenstruktur der RT-Queue RT-Stellwerte
Die `CTD_Stellwerte` wird benötigt, um die von der PC-Ausführungsumgebung gesendeten Stellwerte (PC-Queue PC-Daten) in Abhängigkeit des aktuellen Zustandes bzw. Prozesses des Zustandsautomaten an die Regelungen weiterzugeben.

CTD_Messwerte - Datenstruktur der RT-Queue Messwerte
Diese Cluster-Typdefinition wird zur Erfassung und Übergabe von Sensordaten an die Komponenten der RT-Ausführungsumgebung sowie zu deren Übermittlung an den PC benötigt.

4.3 PC RT Kommunikation

Die Komponente zur Kommunikation zwischen RT und PC wurde mit dem VI `[PCRT]Kommunikation.vi` mit Hilfe von Netzwerkvariablen umgesetzt. Sie ist als SubVI in den beiden Haupt-VIs der PC- und RT-Ausführungsumgebung integriert und kann entweder ein PC- oder ein RT-Verhalten annehmen (Abb. 33).

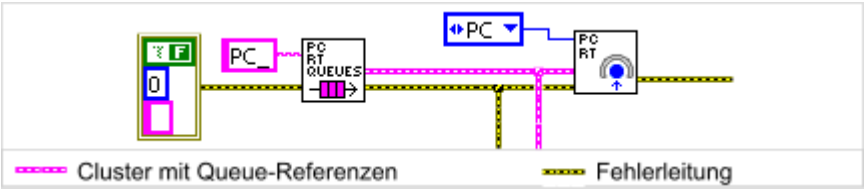

Abbildung 33: Ausschnitt aus dem [PC]Main.vi - Initialisierung der Queues und Übergabe der Queue-Referenzen als Cluster an das [PCRT]Kommunikation.vi mit PC-Verhalten.

Im Folgenden wird das PC-Verhalten dieser Komponente aus Sicht der PC-Ausführungsumgebung beschrieben. Die Beschreibung des RT-Verhaltens entspricht der des PC-Verhaltens, jedoch muss der Prefix „PC“ mit „RT“ vertauscht werden.

4.3.1 Verbindungsaufbau

Der Verbindungsaufbau zwischen PC und RT erfolgt durch die zwei boolschen Netzwerkvariablen `PC_Connected` und `RT_Connected`, die beide mit dem Wert `FALSE` initialisiert werden. Das `[PCRT]Kommunikation.vi` ändert den Wert der `PC_Connected` in `TRUE` und überprüft in regelmäßigen Zeitabständen, ob `RT_Connected` vom `[PCRT]Kommunikation.vi` mit RT-Verhalten ebenfalls auf `TRUE` gesetzt wurde. Besteht eine Netzwerkverbindung zwischen der PC- und RT-Ausführungsumgebung sind beide Netzwerkvariablen auf `TRUE` gesetzt und es kann zum Datenaustausch kommen.

4.3.2 Datenaustausch

Der Datenaustausch erfolgt ebenfalls mit Hilfe von Netzwerkvariablen. Für jede PC- und RT-Queue exisitiert eine korrespondierende Netzwerkvariable vom selben Datentyp bzw. von der selben Datenstruktur (4.2.1 und 4.2.2).
Das `[PCRT]Kommunikation.vi` schreibt Elemente der PC-Queues in die entsprechende PC-Netzwerkvariable und fügt Daten der RT-Netzwerkvariablen in die entsprechende RT-Queue ein.

4.3.3 Verbindungsabbruch

Während dem Datenaustausch wird mit Hilfe der boolschen Netzwerkvariable `PINGPONG` ständig überprüft, ob noch eine Verbindung zwischen PC und RT vorhanden ist.
Das `[PCRT]Kommunikation.vi` negiert den Wert von `PINGPONG` und signalisiert damit dem `[PCRT]Kommunikation.vi` mit RT-Verhalten, dass noch eine Verbindung besteht. Anschließend wartet das `[PCRT]Kommunikation.vi` eine bestimmte Zeit darauf, dass das `[PCRT]Kommunikation.vi` mit RT-Verhalten die Verbindung auch bestätigt, indem es den Wert ebenfalls negiert. Dieser Prozess wiederholt sich so lange, bis es zu einem Verbindungsabbruch kommt.
Bei einem Verbindungsabbruch wird die Netzwerkvariable `PINGPONG` nicht rechtzeitig geändert und das `[PCRT]Kommunikation.vi` fällt wieder in den Verbindungsaufbau zurück.

4.4 Zustandsautomat

Mit der Zustandsdiagramm-Ansicht bietet LabVIEW die Möglichkeit Zustandsautomaten nach dem UML-Standard aufzubauen (siehe 2.4.7). Das Verhalten der Zustände und Übergänge muss dabei jedoch vollständig vom Anwendungsentwickler spezifiziert werden.
Zur Erfüllung der Anforderungen an das Framework und um dem Anwendungsentwickler den Aufbau eines Zustandsautomaten für eine Druckanlage zu vereinfachen, wurden VIs entwickelt, die diesen Zuständen und Übergängen eine gewisse Standardfunktionalität vergeben.

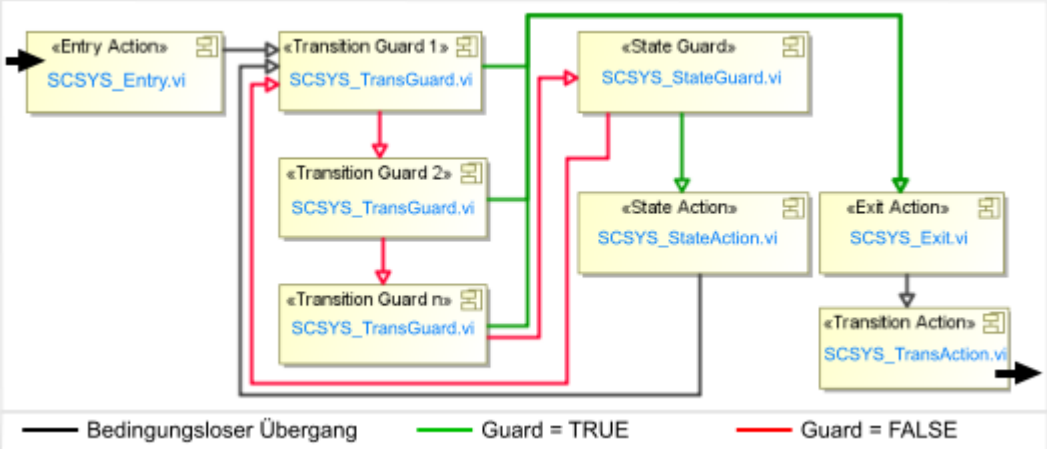

Abbildung 34: SCSYS-VIs in einem Zustand mit n Übergängen.

Diese VIs werden in den Elementen eines Zustandes und eines Überganges eingebettet[7] (Abb. 34). Sie haben Zugriff auf die PC- und RT-Queues, können miteinander kommunizieren und müssen für jeden Zustand und Übergang individuell konfiguriert werden (Abb. 35).
Ein Zustand wird durch die `[SCSYS]Entry.vi` in der `Entry Action` konfiguriert. Der Anwendungsentwickler muss dabei Folgendes spezifizieren:

- Zustandsname aus einer Auswahlliste, die mit der Typdefinition `TD_Zustandsname` gekoppelt ist.
- Auswahl zwischen einem klassischen und speziellen Zustandsverhalten (4.4.2).

[7] Die Einbettung dieser VIs muss nur ein Mal vorgenommen werden. Anschließend können neue Zustände und Übergänge aus bestehenden Zuständen und Übergängen durch Copy&Paste erzeugt werden.

- Array von Übergangsbefehlen, für die der Zustand einen Endzustand darstellt (4.4.5). Jedes Array-Element wird dabei durch eine Auswahlliste, die mit der Typdefinition `TD_Übergangsbefehl` verknüpft ist, spezifiziert.

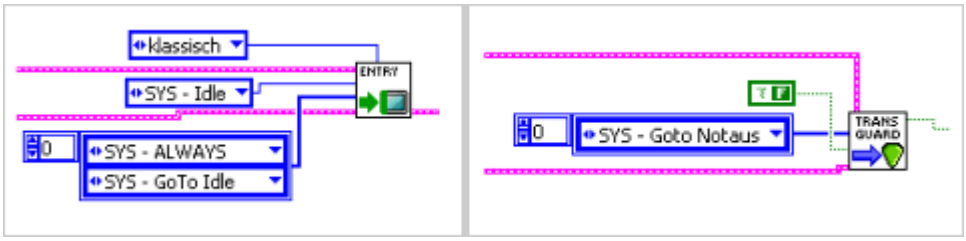

Abbildung 35: Konfiguration eines Zustandes (links) und Zustandsübergangs (rechts)

Ein Übergang wird durch die `[SCSYS]TransGuard.vi` im `Transition Guard` konfiguriert. Der Anwendungsentwickler muss dabei folgendes spezifizieren:

- Array von Übergangsbefehlen, auf die der `Transition Guard` mit einem Übergang reagieren soll (4.4.3). Jedes Array-Element wird dabei durch eine Auswahlliste, die mit der Typdefinition `TD_Übergangsbefehl` verknüpft ist, spezifiziert.

- Ob das übergebene Array negiert werden soll. Der Zustandsübergang reagiert dann auf die nicht die ausgewählten Übergangsbefehle mit einem Zustandswechsel.

- Name des Zustandes zu dem der Zustandsübergang hinführt. Diese Angabe ist optional und wird nur benötigt, wenn Benutzerquellcode in `[SCSYS]TransGuard.vi` eingefügt werden soll.

Der Anwendungsentwickler kann in den SCSYS-VIs individuell für jeden Zustand und Übergang eigenen Quellcode einbetten 5.3.1. Die SCSYS-VIs können dabei die Ausführung des Benutzerquellcodes blockieren oder freigeben. Bei den im Folgenden beschriebenen Funktionalitäten sind immer Zustände und Übergänge gemeint, deren Elemente (`Entry Action`, `State Guard` usw.) die SCSYS-VIs beinhalten.

4.4.1 Senden von Statusmeldungen

Wird ein Zustand betreten, generieren die SCSYS-VIs eine Statusmeldung der Typdefinition `CTD_Statusmeldung`, die an die PC-Ausführungsumgebung

gesendet wird. Die Statusmeldung besteht aus dem in der `[SCSYS]Entry.vi` eingestellten Zustandsnamen des aktuellen Zustands und einem Array von Übergangsbefehlen, auf die der Zustand reagieren kann (4.4.3).

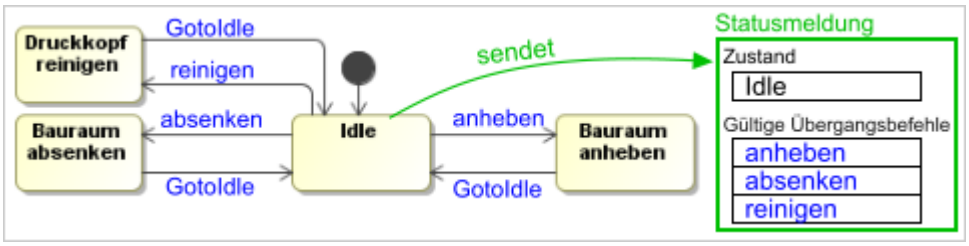

Abbildung 36: Beispiel zum Senden einer Statusmeldung

Dieses Array kann beispielsweise zum Aktivieren oder Deaktivieren von Bedienelementen des Benutzerinterfaces verwendet werden.
Befindet sich der Zustandsautomat aus Abbildung 36 im Zustand `Idle`, können die Bedienelemente zum Anheben und Absenken des Bauraums und zur Druckkopfreinigung aktiviert werden, da der Zustand die Übergangsbefehle `anheben`, `absenken` und `reinigen` verarbeiten kann. Im Zustand `Druckkopf reinigen` werden diese Bedienelemente jedoch deaktiviert, da der Zustand keinen entsprechenden Übergang erlaubt.

Aufbau der gültigen Übergangsbefehle
Beim Betreten eines Zustandes wird zunächst die `[SCSYS]Entry.vi` und der darin enthaltene Benutzerquellcode ausgeführt (Abb. 37). Anschließend werden die einzelnen `Transition Guards` der Zustandsübergänge zum ersten Mal durchlaufen. Beim ersten Durchlauf blockiert die `[SCSYS]TransGuard.vi`

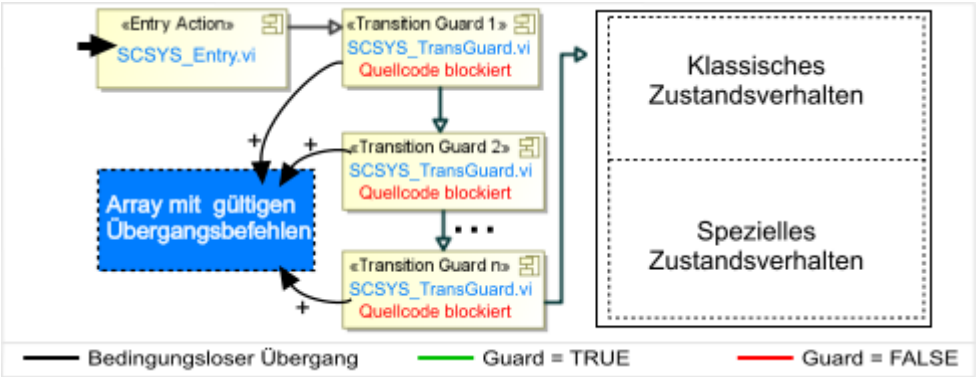

Abbildung 37: Aufbau des Übergangsbefehls-Arrays

die Ausführung des Benutzerquellcodes und sperrt einen möglichen Übergang

zum nächsten Zustand. Sie erzeugt dabei das Array von gültigen Übergangsbefehlen, indem sie die eingestellten gültigen bzw. ungültigen Übergangsbefehle auswertet.
Je nachdem ob in der `[SCSYS]Entry.vi` eine klassisches oder spezielles Zustandsverhalten eingestellt wurde, wird die darauf folgende Ausführung des Benutzerquellcodes im `State Guard` bzw. `State Action` blockiert oder freigegeben (4.4.2).

4.4.2 Klassisches und spezielles Zustandsverhalten

Der Anwendungsentwickler kann zwischen einem klassischen und einem speziellen Zustandsverhalten in der `[SCSYS]Entry.vi` (Abb. 35, links) auswählen.

Klassisches Zustandsverhalten
Das klassische Zustandsverhalten simuliert die normale Ausführungsreihenfolge (2.4.7) der Elemente eines Zustandes und Übergangs.

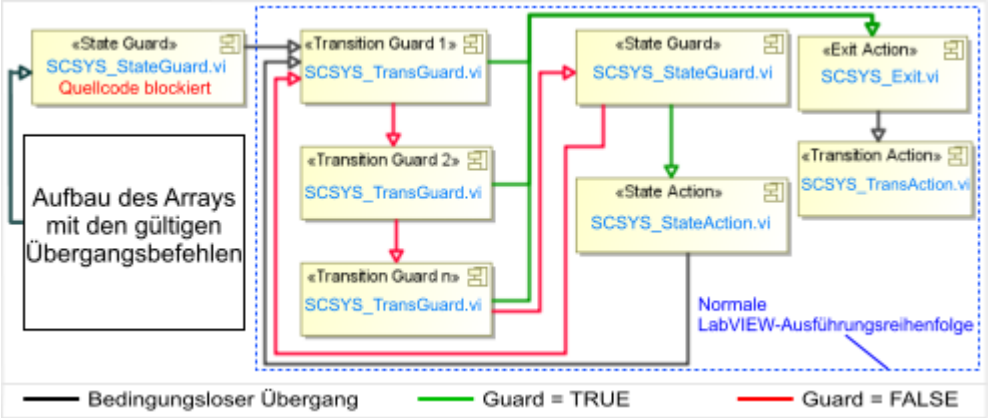

Abbildung 38: Klassisches Zustandsverhalten

Nach dem ersten Durchlaufen der `Transition Guards` zum Aufbau des Array mit den gültigen Übergangsbefehlen blockiert die `[SCSYS]StateGuard.vi` die Ausführung des `State Guard` Benutzerquellcodes und damit auch die Ausführung der `State Action` (Abb. 38). Das Blockieren des Benutzerquellcodes ist notwendig, um wieder in die normale LabVIEW Ausführungsreihenfolge zu gelangen.

Spezielles Zustandsverhalten
Das spezielle Zustandsverhalten ermöglicht einen Zustandsübergang erst, wenn der Benutzerquellcode in der `State Action` diesen explizit freigibt.

Nach dem ersten Durchlaufen der `Transition Guards` zum Aufbau des Ar-

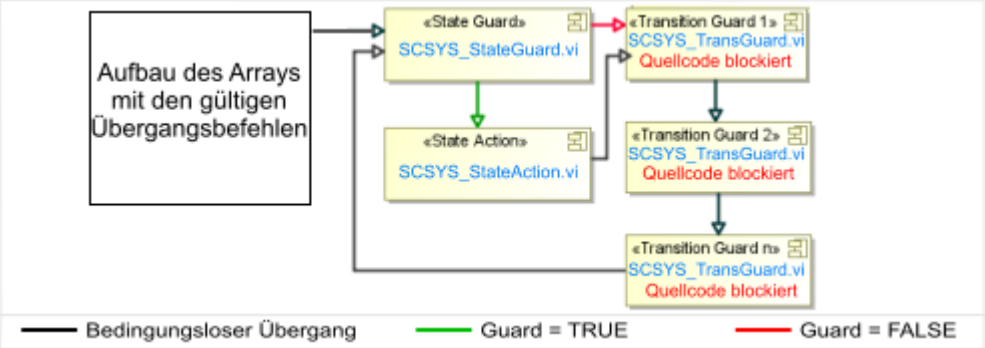

Abbildung 39: Spezielles Zustandsverhalten (GotoNextState=FALSE)

ray mit den gültigen Übergangsbefehlen, wird die `[SCSYS]StateGuard.vi` ausgeführt und je nachdem ob der `State Guard` mit `TRUE` oder `FALSE` evaluiert wurde, auch der Benutzerquellcode in der `[SCSYS]StateAction.vi`.
Die anschließend folgende Ausführung der `Transition Guards` wird immer wieder blockiert (Abb. 39) bis im Benutzerquellcode der `State Action` der Ausgang `GotoNextState` auf `TRUE` gesetzt wird. Wird `GotoNextState` auf `TRUE` gesetzt, nimmt der Zustand das klassische Verhalten an und erlaubt einen Zustandsübergang. Das spezielle Zustandsverhalten kann dazu eingesetzt werden, in einem Zustand so lange zu verweilen, bis beispielsweise eine bestimmte Berechnung ausgeführt wurde oder die Sensordaten bestimmten Kriterien entsprechen.

4.4.3 Reaktion auf Übergangsbefehle

Ein `Transition Guard` reagiert auf einen empfangenen Übergangsbefehl der Typdefinition `CTD_Übergangsbefehl` mit einem Zustandsübergang, falls einer der in der `[SCSYS]TransGuard.vi` eingestellten Übergangsbefehle dem empfangenen Übergangsbefehl entspricht. Mit einem Übergangsbefehl kann nicht nur ein einzelner Zustandswechsel erreicht werden, sondern auch ein Pfad durch mehrere Zustände hindurch.

Überspringen von Zuständen
Wird mit dem gesendeten Übergangsbefehl auch ein Array mit zu überspringenden Zuständen gesendet, blockieren die SCSYS-VIs beim Betreten die Ausführung des Benutzerquellcodes in den einzelnen Elementen des betroffenen Zustandes. Das Überspringen von Zuständen kann im Zustandsautomaten aus Abbildung 41 beispielsweise dazu verwendet werden, die gesamten

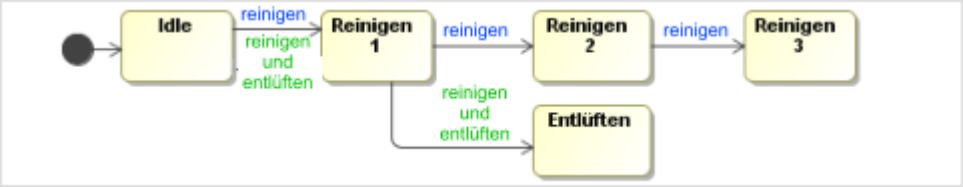

Abbildung 40: Beispiel zur Ausführung eines Übergangsbefehls durch mehrere Zustände hindurch.

oder eine Teilmenge der durch den Übergangsbefehl `reinigen` markierten Reinigungszustände auszuführen. Damit kann der Zustandsautomat kompakter gehalten werden, da für das ein-, zwei- und dreimalige Reinigen keine drei unterschiedlichen Übergangsbefehle bzw. Zustandspfade benötigt werden.

Ausführen einer Befehlskette
Wird mit dem Übergangsbefehl ein Array von Übergangsbefehlen gesendet, arbeitet der Zustandsautomat diese der Reihe nach ab. Sobald ein Übergangsbefehl abgearbeitet oder als ungültig befunden wurde (4.4.5), wird der nächste Befehl in der Befehlskette ausgeführt. In Abbildung 41 ist dazu ein Beispiel zu sehen.

Abbildung 41: Beispiel zur Reaktion auf eine Befehlskette

Während sich der Zustandsautomat im Zustand `Z0` befindet wird von der PC-Ausführungsumgebung ein Übergangsbefehl mit einer Befehlskette gesendet. Da der erste Befehl `GotoZ1` lautet, findet ein Übergang zum Zustand `Z1` statt. Der Zustand `Z1` wurde als Endzustand des Befehls `GotoZ1` markiert, wodurch als nächstes der Befehl `GotoZ0` aktiv wird. Die vollständige Ausführungsreihenfolge der Zustände ist rechts in Abbildung 41 zu sehen.

Übergangsbefehl SYS-ALWAYS
Der spezielle Übergangsbefehl `SYS-ALWAYS` ist bereits in der Typdefinition `TD_Übergangsbefehl` enthalten und ermöglicht einen bedingungslosen Zustandsübergang. Wurde ein `Transition Guard` zur Reaktion auf diesen

Übergangsbefehl eingestellt, wird immer ein Übergang ausgeführt, unabhängig davon, ob tatsächlich ein Übergangsbefehl von der PC-Ausführungsumgebung gesendet wurde oder nicht. Dieser Übergangsbefehl kann zum Beispiel dazu verwendet werden, nach Abarbeitung eines Zustandes bzw. eines Zustandpfades zurück zum Ausgangszustand zu gelangen.

4.4.4 Reaktion auf Abbruchbefehle

Abbruchbefehle der Typdefinition `TD_Abbruchbefehl` werden von der PC-Ausführungsumgebung an den Zustandsautomaten gesendet, um die weitere Ausführung von Übergangsbefehlen und Zuständen zu verhindern oder den Minimalautomaten in den Zustand `SYS-Notaus` bzw. `SYS-Störung` zu versetzen (4.4.7). Im Folgenden wird die Reaktion der SCSYS-VIs auf übergebene Abbruchbefehle erläutert.

Befehl abbrechen
Die SCSYS-VIs des aktuellen Zustands reagieren auf diesen Abbruchbefehl, indem sie den aktiven Übergangsbefehl aus der PC-Queue Übergangsbefehl entfernen.

Zustand abbrechen
Wird der Abbruchbefehl „Zustand abbrechen“ gesendet, verhindern die SCSYS-VIs eine weitere Ausführung der `State Action` und dem `State Guard` des aktuellen Zustands. Ein Zustandsabbruch ist in Notfällen wichtig, da im Benutzerquellcodes des `State Guard` oder der `State Action` auch While-Schleifen ausgeführt werden können. Eine While-Schleife verhindert während ihrer Ausführung einen Zustandsübergang, da die `Transition Guards` erst nach der Ausführung durchlaufen werden. Diese While-Schleifen müssen deshalb in Notfällen immer abgebrochen werden können.

Zustand und Befehl abbrechen
Mit diesem Abbruchbefehl wird die Ausführung des aktuellen Zustands abgebrochen und der aktive Übergangsbefehl in der PC-Queue Übergangsbefehl gelöscht.

Notaus und Störung
Mit diesen Abbruchbefehlen wird ein Übergang in den Zustand `SYS-Notaus` bzw. `SYS-Störung` des Minimalautomaten erzwungen. Dazu wird der aktuelle Zustand abgebrochen und der aktive Übergangsbefehl mit dem Übergangsbefehl `SYS-GotoNotaus` bzw. `SYS-GotoStörung` überschrieben (4.4.7).

4.4.5 Senden von Befehlsantworten

Der Zustandsautomat sendet Befehlsantworten als Reaktion auf empfangene Übergangsbefehle an die PC-Ausführungsumgebung. Sie bestehen aus dem aktuellen und den zuletzt verarbeiteten Übergangsbefehl und können die im Folgenden beschriebenen Meldungen beinhalten.

Befehl akzeptiert
Ein Befehl wird akzeptiert, wenn kein anderer Befehl vom Zustandsautomaten verarbeitet wird.

Befehl abgewiesen
Ein Übergangsbefehl wird vom Zustandsautomaten abgewiesen, wenn sich bereits ein Übergangsbefehl in der PC-Queue Übergangsbefehl befindet.

Befehl abgearbeitet
Wird eine Zustand betreten, der als Endzustand des aktuellen Übergangsbefehls deklariert wurde, antwortet der aktuelle Zustand mit „Befehl abgearbeitet“ und leert die PC-Queue Übergangsbefehl, um einen neuen Befehl empfangen zu können. Wurde mit dem Übergangsbefehl eine Befehlskette gesendet, wird nur der abgearbeitete Befehl aus der Befehlskette gelöscht und der nächste Übergangsbefehl aktiv.

Befehl abgebrochen
Die Befehlsantwort „Befehl abgebrochen“ wird gesendet, wenn der aktuelle Übergangsbefehl vom Benutzer durch einen Abbruchbefehl abgebrochen wurde (4.4.4).

Befehl ungültig
Befindet sich der Zustandsautomat in einem Zustand, dessen Zustandsübergänge nicht auf den gesendeten Befehl reagieren können, wird mit „Befehl ungültig“ geantwortet und die PC-Queue Übergangsbefehl geleert. Das Erkennen und Löschen von ungültigen Befehlen ist insbesondere in der Entwicklungsphase der Maschinensteuerung notwendig, da der Anwendungsentwickler versehentlich einen ungültigen Befehl an den Zustandsautomaten senden und damit die weitere Ausführung des Zustandsautomaten blockieren könnte.

4.4.6 Autonomes Verhalten des Zustandsautomaten

Der bisher vorgestellte Zustandsautomat reagierte immer auf Übergangsbefehle, die von der PC-Ausführungsumgebung gesendet wurden. Jedoch muss eine Druckanlage bestimmte Prozesse auch autonom ausführen und selbst

entscheiden können, ob diese Prozesse beendet werden müssen oder nicht. Der Druckprozess muss beispielsweise so lange ausgeführt werden, bis die gesamten Druckdaten verarbeitet worden sind. Ein solches autonomes Verhalten ist mit den SCSYS-VIs möglich, indem sich der Zustandsautomat in bestimmten Zuständen selbst Übergangsbefehle sendet und damit eigenständig über den nächsten Zustand entscheidet.

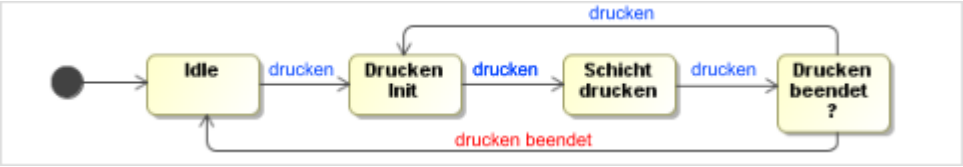

Abbildung 42: Beispiel für autonomes Verhalten

Dazu ist in Abbildung 42 ein Beispiel zu sehen. Im Zustand `Idle` wird dem Zustandsautomaten der Befehl `drucken` von der PC-Ausführungsumgebung übergeben. Der Zustandsautomat führt diesen Übergangsbefehl so lange aus bis der Zustand `Drucken beendet?` entscheidet, dass alle Schichten gedruckt worden sind und den Übergangsbefehl `drucken` der PC Queue Übergangsbefehl mit dem Übergangsbefehl `drucken beendet` überschreibt.

4.4.7 Minimalautomat

Auf der Basis der SCSYS-VIs wurde ein Minimalautomat entwickelt (Abb. 43), der als Ausgangspunkt für den Aufbau des Zustandsautomaten durch den Anwendungsentwickler dienen soll und den Übergang durch Abbruchbefehle in den darin enthaltenen Zuständen `SYS-Notaus` und `SYS-Störung` immer ermöglicht. Dazu wurden die Typdefinitionen `TD_Zustandsname` und `TD_Übergangsbefehl` mit den entsprechenden Zustandsnamen und Übergangsbefehlen erweitert. Das Verhalten der Zustände des Minimalautomaten muss dabei vom Anwendungsentwickler an die Druckanlage angepasst werden.

4.5 Benutzerinterface und Eventhandler

Zur Realisierung der Maschinensteuerung wurde ein Benutzerinterface umgesetzt (Abb. 44), dass alle Daten aus den RT-Queues visualisiert und dem Anwendungsentwickler die Möglichkeit bietet, über einen Eventhandler Übergangsbefehle, Abbruchbefehle und allgemeine Daten an den RT zu senden. Das Benutzerinterface und der zugehörige Eventhandler sind in der `[PC]Main.vi`

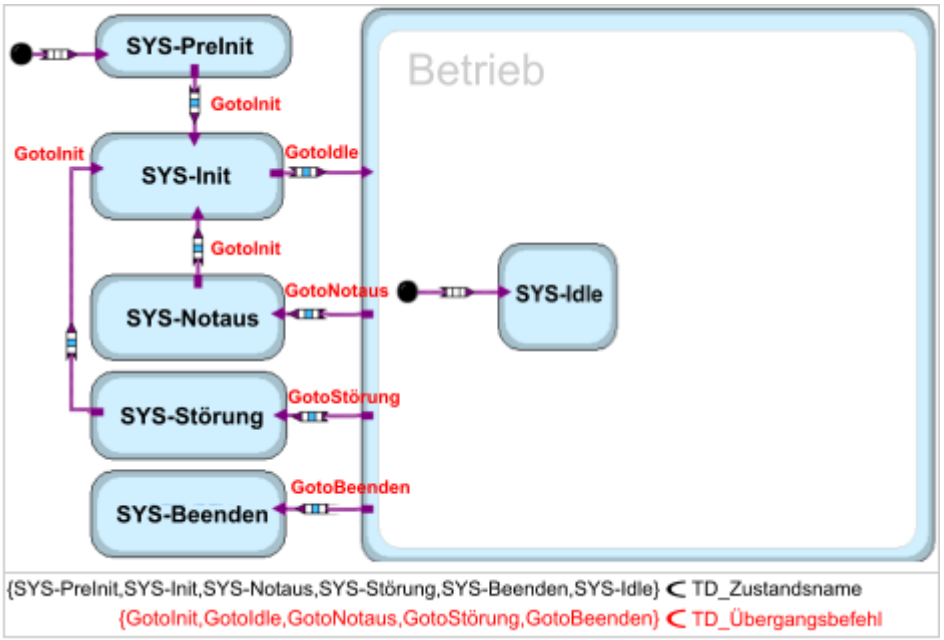

Abbildung 43: Minimalautomat in der LabVIEW-Zustandsdiagrammansicht

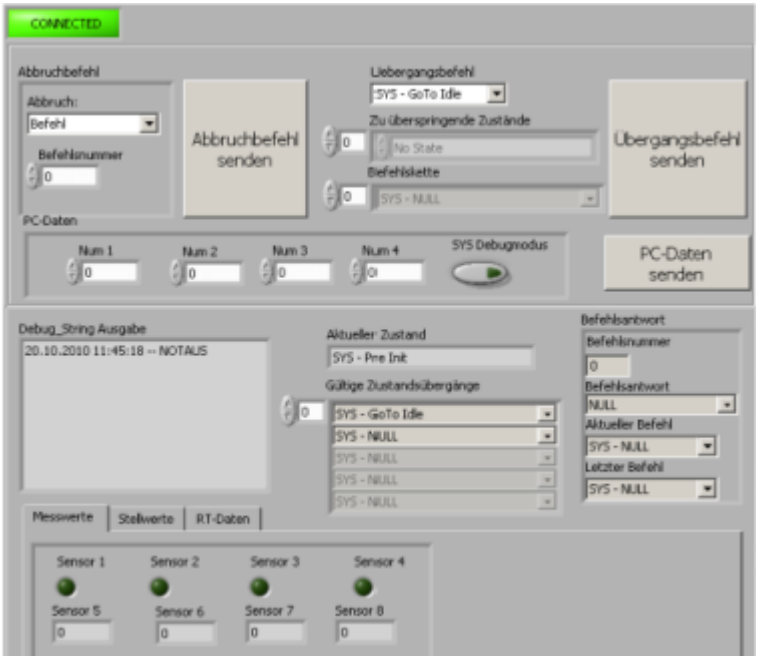

Abbildung 44: Benutzerinterface für den Anwendungsentwickler

als nebenläufige Prozesse enthalten. Der Eventhandler reagiert auf die Schaltfläche `Übergangsbefehl senden`, indem der eingestellte Übergangsbefehl ausgelesen und in die PC-Queue Übergangsbefehl einfügt wird. Ähnlich verhält er sich beim Betätigen der Schaltflächen `Abbruchbefehl senden` und `PC-Daten senden`.
Jedes Bedien- und Anzeigeelement ist an die zuvor vorgestellten Typdefinitionen gekoppelt, wodurch Änderungen an den Typdefinitionen[8] auch eine Änderungen in den entsprechenden Bedien- und Anzeigeelementen zur Folge haben. Dieses Benutzerinterface ist nicht zur Verwendung durch einen Endanwender gedacht, sondern zur Realisierung und zum Testen des Zustandsautomaten durch den Anwendungsentwickler.

4.5.1 Benutzerinterface für den Endanwender

Das Benutzerinterface zur Steuerung einer Druckanlage durch den Endanwender sollte übersichtlich und eindeutig sein, damit es in Notfällen sicher bedient werden kann [22]. Das zuvor vorgestellte Benutzerinterface für den Anwendungsentwickler ist dazu nicht geeignet, da beispielsweise jederzeit alle möglichen Übergangsbefehle an den Zustandsautomaten gesendet werden können, selbst wenn der aktuelle Zustand diese nicht verarbeiten kann.
Aus diesem Grund wurde ein Benutzerinterface zur Steuerung des Minimal-

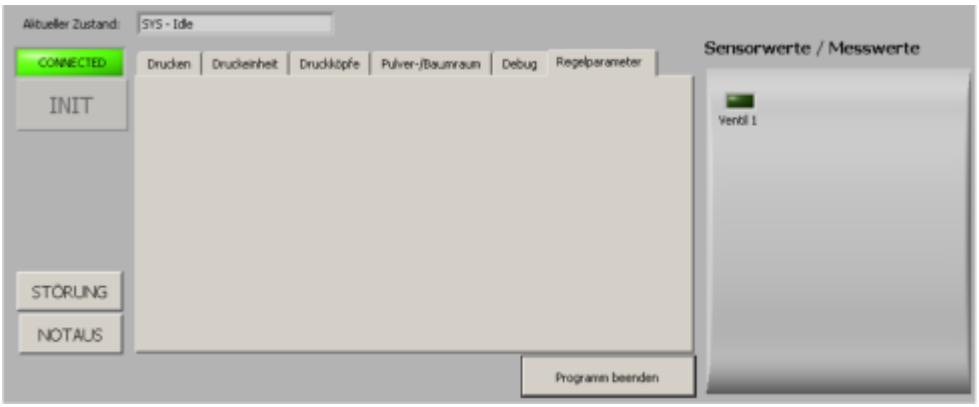

Abbildung 45: Benutzerinterface für den Endanwender im Zustand `Idle` des Minimalautomaten

automaten für einen Endanwender entwickelt (Abb. 45), dessen Schaltflächen an einen individuellen Übergangsbefehl gekoppelt sind und in Abhängigkeit der möglichen Zustandsübergänge deaktiviert werden.

[8] Damit sind zum Beispiel das Einfügen von neuen Übergangsbefehlen oder Stellwerten gemeint.

Die `Init` Schaltfäche aus Abbildung 45 ist beispielsweise deaktiviert bzw. ausgegraut, da sich der Minimalautomat im Zustand `SYS-Idle` befindet und von dort kein Übergang in den Zustand `SYS-Init` möglich ist. Diese Schaltfläche kann erst aus den Zuständen `SYS-PreInit`,`SYS-Notaus` bzw. `SYS-Störung` des Minimalautomaten betätigt werden, was dann das Senden des Übergangsbefehls `SYS-GotoInit` zur Folge hat. Ferner ist das Betätigen der Schaltflächen `Notaus` und `Störung` jederzeit möglich, wodurch ein Übergang in die entsprechenden Zustände des Minimalautomaten ausgelöst werden kann.
Dieses Benutzerinterface und der zugehörige Eventhandler können vom Anwendungsentwickler als Vorlage für den Aufbau des endgültigen Benutzerinterfaces für den Endanwender verwendet werden.

4.6 Sensoren und Aktoren Mapping

Das Sensoren und Aktoren Mapping ist Teil der `[FPGA]Main.vi` und ermöglicht das Auslesen und Beschreiben der I/O Modul Ein- und Ausgänge durch die RT-Ausführungsumgebung.

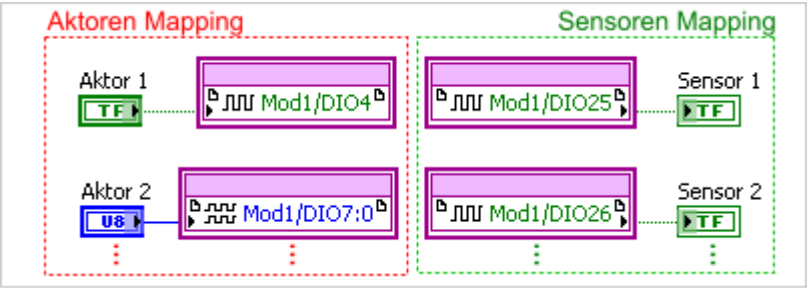

Abbildung 46: Beispiel-Ausschnitt der [FPGA]Main.vi: Sensoren und Aktoren Mapping

Diese Komponente muss vom Anwendungsentwickler in Abhängigkeit der verwendeten I/O Module und der daran angeschlossenen Sensoren und Aktoren vollständig spezifiziert werden. Dazu können LabVIEW-eigene FPGA-Funktionsblöcke zum Auslesen und Beschreiben der I/O Modul Ein- bzw. Ausgänge verwendet werden. Diese Funktionsblöcke müssen vom Anwendungsentwickler mit benannten Ein- und Ausgängen innerhalb der `[FPGA]Main.vi` verbunden werden.
In Abbildung 46 ist dazu ein Beispiel zu sehen. Der `[FPGA]Main.vi` Eingang `Aktor 1` wurde hier über einen Funktionsblock zum Beschreiben des I/O Moduls `Mod1` mit dem `Mod1`-Ausgang `DIO4` verknüpft. Ähnliches gilt für den Eingang `Aktor 2` sowie den Ausgängen `Sensor 1` und `Sensor 2`.

Über eine Referenz zur `[FPGA]Main.vi` ist damit der Zugriff auf die Aktoren und Sensoren von der RT-Ausführungsumgebung möglich (Abb. 47).

4.7 Sensordaten auslesen

Das Auslesen der Sensordaten entspricht einem nebenläufigen Prozess in der `[RT]Main.vi`, der in regelmäßigen Zeitabständen die aktuellen Sensordaten über eine Referenz zur `[FPGA]Main.vi` abfragt und in die RT-Queue Messwerte einfügt (Abb. 47). Diese Komponente muss vom Anwendungsentwickler

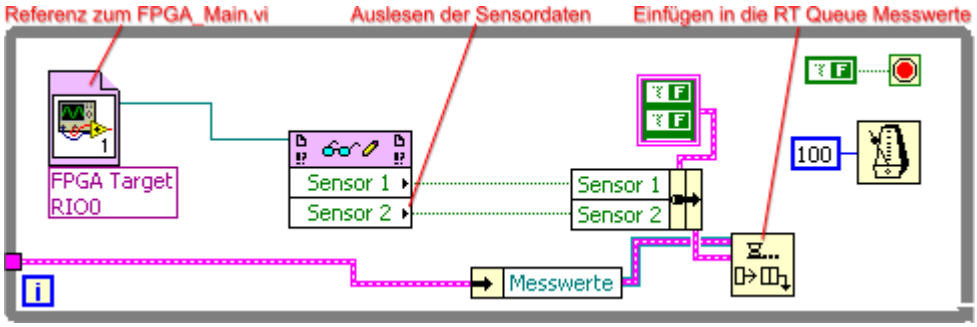

Abbildung 47: Auslesen der Sensordaten in der [RT]Main.vi

in Anlehnung an Abbildung 47 vollständig spezifiziert werden, damit jederzeit ein Zugriff auf die Sensordaten über die RT Queue Messwerte möglich ist. Im Falle analoger Sensordaten ist eventuell noch eine Formatierung der Messwerte notwendig.

4.8 Temperaturregelung

Die Komponente zur Temperierung von Bindemittelbehältern und Druckköpfen wurde mit dem VI `[RT]Temperierung.vi` in Anlehnung an einen Zweipunktregler umgesetzt, da dies ein gängiges Verfahren zur Regelung von Temperaturen darstellt [35].
Sie ist als nebenläufiges SubVI in der `[RT]Main.vi` integriert und kann für die Regelung mehrerer Temperaturen verwendet werden.
Dieses VI erhält über dessen Eingänge einen Soll- und einen Istwert. Sie vergleicht Soll- und Istwert miteinander und schaltet über eine Referenz zur `[FPGA]Main.vi` die entsprechenden Heizelemente bei zu niedriger Temperatur ein und bei zu hoher Temperatur aus.
Dieses VI erfordert eine Anpassung an den ausgewiesenen Stellen innerhalb des Quellcodes an die zu verwendenden Heizelemente und muss mit dem entsprechenden Soll- und Istwert aufgerufen werden (5.2).

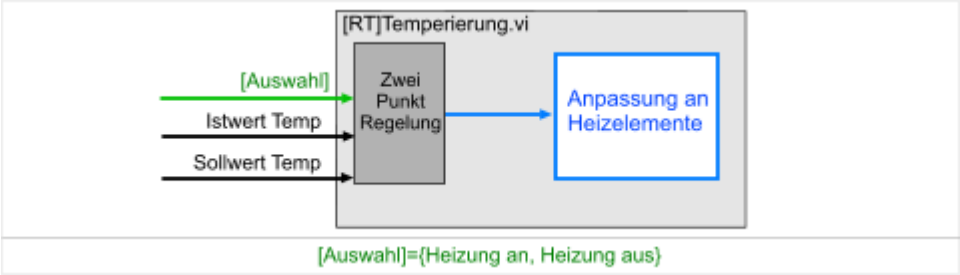

Abbildung 48: [RT]Temperierung.vi

4.9 Druckregelung

Die Regelung eines Systems zur Unter- und Überdruckerzeugung mit Hilfe von Ventilen, Drucksensoren und Pumpen wurde mit dem VI `[RT]Druckregelung.vi` umgesetzt. Sie ist als nebenläufiges SubVI in der `[RT]Main.vi` integriert und kann in Abhängigkeit ihres Eingangs `Auswahl` (Datentyp Enum) fünf Verhaltensweisen annehmen (Abb. 49).

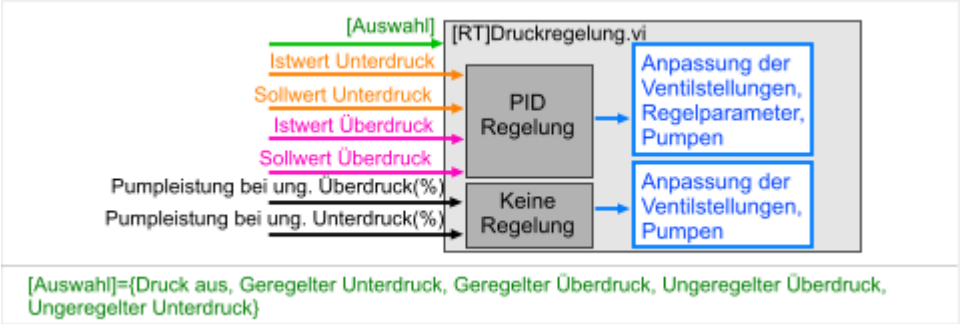

Abbildung 49: [RT]Druckregelung.vi

- Ausgeschalteter Druck: Bei ausgeschaltetem Druck werden die Pumpen zur Erzeugung von Unter- und Überdruck ausgeschaltet und die Ventile in eine Grundstellung versetzt.
- Ungeregelter Über- und Unterdruck: Hier werden die Ventilstellungen so verändert, dass bei Aktivierung der Über- bzw. Unterdruckpumpe in Abhängigkeit der entsprechenden prozentualen Angabe am Eingang des VI `[RT]Druckregelung.vi` ein Über- bzw. Unterdruck erzeugt wird.
- Geregelter Über- und Unterdruck: Die Regelung der Über- bzw. Unterdruck Istwerte wurde durch die Verwendung eines PID-Reglers[9] umge-

[9] LabVIEW bietet dazu entsprechende Funktionsblöcke an.

setzt, da dieser von den Standard-Reglern am anpassungsfähigsten an eine Regelstrecke ist und bei konstantem Sollwert eine bleibende Regelabweichung verhindert [28]. Um Über- bzw. Unterdruck erzeugen zu können, müssen auch hier die Ventilstellungen entsprechend verändert werden.

Beim Testen der Druckregelungen an einem Versuchsstand wurde festgestellt, dass es beim Wechsel von der Überdruckerzeugung zur geregelter Unterdruckerzeugung (und umgekehrt) zum starken Überschwingen des Systems kommt. Diese Überschwingung musste vermieden werden, da beispielsweise bei zu großer Überschreitung des Unterdruck-Sollwerts zur Vermeidung der Tropfenbildung an den Druckköpfen Luft durch die Druckkopfdüsen angesaugt wird.
Als Ursache dieses Verhaltens konnten die Parameter der PID-Regelungen ausgemacht werden, die an eine Regelung aus einem drucklosen Zustand angepasst wurden. Dieses Problem konnte behoben werden, indem bestimmte Ventile beim Wechsel von der Überdruckerzeugung zur Unterdruckerzeugung so eingestellt werden, dass es zum Abbau des vorher erzeugten Über- bzw. Unterdrucks im System kommt und der PID-Regler erst danach aktiviert wird.
Dieses VI kann zur Regelung mehrerer Druckerzeugungssysteme verwendet werden (5.2). Zur Regelung von Unter- und Überdruck wurde außerdem eine Ansteuerung von Pumpen über digitale Ausgänge benötigt.

Ansteuerung einer Pumpe über einen digitalen Ausgang
Die Ansteuerung einer Pumpe über einen digitalen I/O Modul Ausgang wurde mit dem VI `[FPGA]PWM.vi` durch eine Pulsweitenmodulation auf der FPGA-Ausführungsumgebung realisiert.
Die Pulsweitenmodulation erlaubt das Erzeugen von quasi-analogen Signalen mit Hilfe digitaler Ausgänge, indem das Tastverhältnis eines Rechtecksignals bei einer fester Grundfrequenz variiert wird [12].
Im Falle einer Pumpe, die durch einen digitalen Ausgang angesteuert wird und deshalb entweder keine oder eine volle Pumpleistung erzielt, kann durch die Pulsweitenmodulation deren Versorgungsspannung so variiert werden, dass die Pumpe nur einen Teil der möglichen elektrischen Energie in Pumpleistung umwandelt und dadurch Leistungszwischenwerte erzeugt.
Das `[FPGA]PWM.vi` erhält dazu über ihren einzigen Eingang einen 8-Bit Integerwert, der die zu erzielenden Pumpleistung repräsentiert. Sie gibt bei einer fester Grundfrequenz so viele Impulse an die Pumpe weiter, wie durch den Integerwert spezifiziert wurde. Dadurch sind insgesamt 256 Leistungsabstufungen der Pumpe möglich.

4.10 Schrittmotorsteuerung

Die Steuerung eines Schrittmotors wurde mit Hilfe zweier FPGA-LUTs realisiert, aus denen der Zeitabstand zwischen zwei Motorschritten ausgelesen wird. Eine LUT dient dabei der Beschleunigung des Motors auf eine bestimmte Drehzahl, die andere LUT dient zu dessen Abbremsung. Eine schrittweise Beschleunigung des Schrittmotors ist notwendig, da der Motor beim Sprung auf eine zu hohe Drehzahl innerhalb eines Schrittes, Schrittverluste erleiden oder vollständig blockieren kann [29]. Ähnliches gilt auch für die Abbremsung.

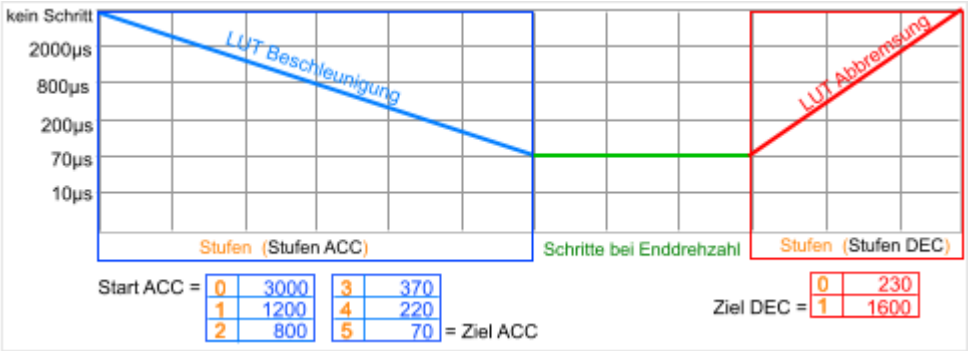

Abbildung 50: Beispiel zum Beschreiben der Beschleunigungs- und Abbremsungs-LUT

4.10.1 Beschreiben der Look-Up Tables

Vor der eigentlichen Motorbewegung werden die zwei LUTs mit Hilfe des VIs `[RT]ACC_DEC_Write.vi` von der RT-Ausführungsumgebung aus beschrieben[10]. Für das Beschreiben der Beschleunigungs-LUT erhält dieses VI über ihre Eingänge folgende Daten:

- Referenz zur Beschleunigungs-LUT
- Start ACC (int): Initialer Zeitabstand zwischen zwei Schritten in Microsekunden.
- Ziel ACC (int): Finaler Zeitabstand zwischen zwei Schritten in Microsekunden (Enddrehzahl).
- Stufen ACC (int): Anzahl der Stufen (LUT-Einträge) vom initialen bis zum zu erreichenden Zeitabstand.

[10] Für das Beschreiben der LUTs stehen Eingänge der [FPGA]Main.vi zur Verfügung.

Sie erzeugt aus diesen Angaben LUT-Einträge (Abb. 50), die einen diskreten Übergang vom initialen zum finalen Zeitabstand zwischen zwei Motorschritten repräsentieren.
Das Beschreiben der Abbremsungs-LUT erfolgt in Anlehnung an die Beschleunigungs-LUT, jedoch ist dazu nur noch die Angabe der Stufenanzahl und der Drehzahl (`Stufen DEC` und `Ziel DEC`, Abb. 50), auf die abgebremst werden soll, notwendig.

4.10.2 Motorbewegung

Die Motorbewegung kann anschließend über eine Referenz zur `[FPGA]Main.vi` und der darin eingebundenen `[FPGA]Schrittmotor.vi` gestartet werden. Dazu werden folgende FPGA-Eingänge bereitgestellt:

- Motorbewegung(Boolean): Startet den Motor falls `TRUE`.
- Richtung (Boolean): Drehrichtung des Motors.
- Schritte nach ACC (Integer): Anzahl der Schritte, die nach der Beschleunigungsphase mit der Enddrehzahl ausgeführt werden sollen.

Die `[FPGA]Schrittmotor.vi` gibt Impulse über die I/O Modul Ausgänge an den Schrittmotor weiter. Jeder Impuls hat einen Schritt des Schrittmotors zur Folge. In der Beschleunigungsphase werden die Zeitabstände zwischen zwei Impulsen aus der Beschleunigungs-LUT ausgelesen, bis das Ende der LUT erreicht wurde. Mit dem letzten Wert der LUT werden anschließend so viele Impulse erzeugt, wie mit dem FPGA-Eingang `Schritte nach ACC` spezifiziert wurde (Abb. 50, grüne Linie). In der danach folgenden Abbremsphase wird die Drehzahl des Schrittmotors in Abhängigkeit der Abremsungs-LUT schrittweise auf den Zeitabstand `Ziel DEC` verringert und nach dem Erreichen des letzten LUT-Eintrages die Motorbewegung vollständig gestoppt.
Die `[FPGA]Schrittmotor.vi` muss vom Anwendungsentwickler an den ausgewiesenen Stellen innerhalb des Quellcodes an den zu steuernden Schrittmotor angepasst werden. Es muss dabei Folgendes spezifiziert werden:

- Welche LUTs zur Beschleunigung und Abbremsung verwendet werden sollen.
- Ausgänge der I/O Module zu den `ENABLE`, `STEP` und `DIRECTION` Eingängen des Schrittmotors.

Zur Verwendung mit mehreren Schrittmotoren können mehrere Kopien der `[FPGA]Schrittmotor.vi` erzeugt und in die `[FPGA]Main.vi` mit unterschiedlich benannten Eingängen eingefügt werden (5.1.1).

4.11 Druckkopfansteuerung

Die Druckkopfansteuerung wurde für Druckköpfe (Abb. 51) entwickelt, deren Düsen in Abhängigkeit eines Schieberegisters zum Feuern gebracht werden können. Das Schieberegister lässt sich dabei über die Eingänge `CLOCK` und `DATA` beschreiben. Eine positive Flanke am `FIRE`-Eingang hat das Feuern der

Abbildung 51: Druckkopf mit 64-Bit Schieberegister

Druckkopfdüsen in Abhängigkeit des Schieberegisters zur Folge. Diese Druckköpfe lassen sich über ihre Ausgänge in Reihe schalten, so dass das Beschreiben der Schieberegister mehrerer Druckköpfe über eine `CLOCK`- und `DATA`-Leitung möglich ist. Die Ansteuerung der Druckköpfe und die Druckdatenübertragung an den FPGA wurde mit dem FPGA-VI `[FPGA]Druckkopf.vi` und dem RT-VI `[RT]WriteToDMA.vi` realisiert.

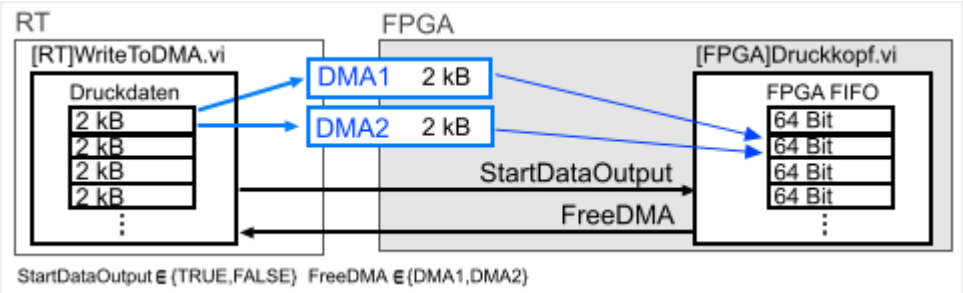

Abbildung 52: [RT]WriteToDMA.vi und [FPGA]Druckkopf.vi

4.11.1 Übertragung der Druckdaten an den FPGA

Die Übertragung der Druckdaten erfolgt mittels zweier DMA-Kanäle, die abwechselnd von der `[RT]WriteToDMA.vi` mit Elementen aus der PC-Queue Druckdaten beschrieben werden (Abb. 52).
Beim Aufrufen der `[RT]WriteToDMA.vi` werden die DMA-Kanäle `DMA1` und `DMA2` beschrieben. Der `[FPGA]Druckkopf.vi` wird über den FPGA-Eingang `StartDataOutput` signalisiert, dass Druckdaten vorliegen und der Druckprozess (4.11.2) beginnen kann. Anschließend wartet die `[RT]WriteToDMA.vi` darauf (Abb. 53), dass der DMA-Kanal `DMA1` bzw. `DMA2` über den FPGA-Ausgang `FreeDMA` von der `[FPGA]Druckkopf.vi` freigegeben wird.

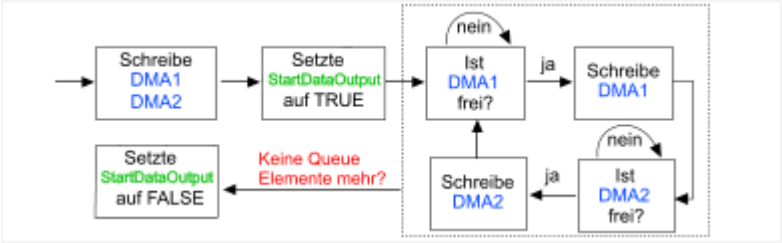

Abbildung 53: Verhalten der [RT]WriteToDMA.vi

Die `[FPGA]Druckkopf.vi` liest die DMA-Kanäle abwechselnd aus und fügt sie in einen FPGA-internen FIFO-Speicher ein. Sobald ein DMA-Kanal vollständig ausgelesen wurde, wird der FPGA-Ausgang `FreeDMA` auf den entsprechenden Wert[11] gesetzt und der `[RT]WriteToDMA.vi` damit signalisiert, dass darin neue Daten eingefügt werden können. Dieser Prozess wiederholt sich solange, bis alle Elemente der PC-Queue Druckdaten an den FPGA übergeben wurden.

4.11.2 Druckprozess

Der Druckprozess entspricht einem zum Empfang der Druckdaten parallelen Prozess in der `[FPGA]Druckkopf.vi`, der, sobald Elemente in dem FPGA-internen FIFO-Speicher vorhanden sind, in Abhängigkeit eines externen Trigger-Signals ausgeführt wird. Der externe Trigger dient hierbei der Synchronisation der Feuerimpulse zur Bewegung der Druckeinheit über den Bauraum und kann zum Beispiel von der Anzahl der ausgeführten Schritte eines Schrittmotors abhängig gemacht werden.
Dieser Prozess entnimmt die zuerst eingefügten Elemente aus dem FIFO-Speicher und beschreibt die Schieberegister der Druckköpfe über I/O Modul Ausgänge mit entsprechenden Bit-Werten. Dabei werden so viele 64-Bit Werte entnommen wie für die Anzahl und Auflösung der in den Druckköpfen befindlichen Schieberegister benötigt werden. Bei zwei Druckköpfen mit 128-Bit Schieberegistern werden zum Beispiel vier 64-Bit Elemente entnommen. Anschließend wird auf ein Trigger-Signal gewartet, um durch eine positive Flanke am `FIRE`-Eingang der Druckköpfe, die Druckkopfdüsen zu feuern. Das Beschreiben der Schieberegister und das Feuern in Abhängigkeit eines Trigger-Signals wird so lange wiederholt, bis alle Elemente des FIFO-Speichers abgearbeitet wurden.

[11] Wenn beispielsweise `DMA1` ausgelesen wurde, wird `FreeDMA` auf den Wert „DMA1“ gesetzt.

5 Anwendungsentwicklung mit dem Framework

In diesem Kapitel wird ein Vorschlag für die Vorgehensweise bei der Entwicklung einer Maschinensteuerung für eine prototypische 3D-InkJet-Druckanlage mit dem Framework gemacht. Die folgenden Unterkapitel müssen deshalb nicht zwingend in der aufgezeigten Reihenfolge abgearbeitet werden.

5.1 Anpassung an die Aktoren und Sensoren

Das Framework sollte zunächst an die vorhandenen I/O Modulen sowie den Sensoren und Aktoren der zu steuernden Druckanlage angepasst werden. Die `[FPGA]Main.vi` muss deshalb in Anlehnung an Kapitel 4.5 (Sensoren und Aktoren Mapping) mit entsprechenden Ein- und Ausgängen erweitert werden, damit der Zugriff auf die Sensoren und Aktoren von den Komponenten der RT-Ausführungsumgebung möglich ist. Außerdem ist eine Anpassung an die Anzahl der vorhandenen Pumpen, Schrittmotoren und Druckköpfe notwendig.

5.1.1 Schrittmotoren

Je nachdem wie viele Schrittmotoren in der Druckanlage vorhanden sind, sollten mehrere Kopien der `[FPGA]Schrittmotor.vi` erzeugt und als SubVI in die `[FPGA]Main.vi` integriert werden. Der Anwendungsentwickler muss dabei abschätzen wie viele Beschleunigungs- und Abbremsungs-LUTs für die Motorbewegungen verwendet werden sollen und diese dann über die LabVIEW Entwicklungsumgebung anlegen. Anschließend muss der Quellcode

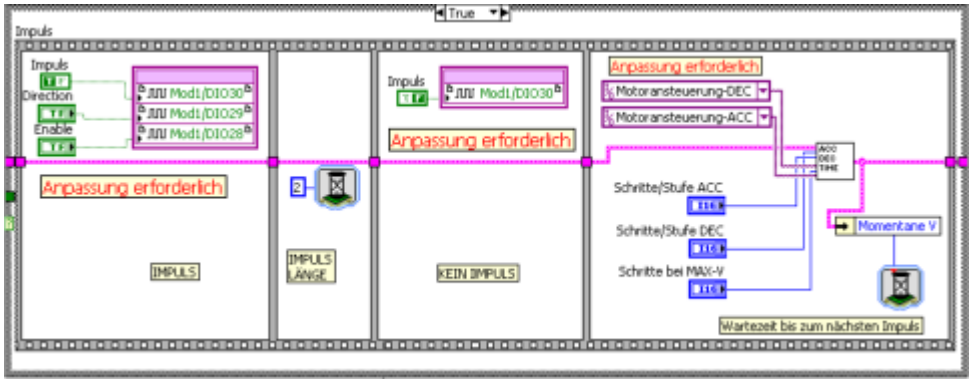

Abbildung 54: Ausschnitt aus dem [FPGA]Schrittmotor.vi mit markierten Anpassungsstellen

jeder Kopie an die zu verwendenden LUTs und den I/O Modul Ausgängen

zu den `ENABLE`, `STEP` und `DIRECTION` Eingängen jedes Schrittmotors angepasst werden (Abb. 54). Die Parameter zum Beschreiben der LUTs mit dem `[RT]ACC_DEC_Write.vi` müssen anhand der Schrittmotorspezifikation und der zu erfüllenden Motorbewegung festgelegt werden.

5.1.2 Druckköpfe

Die in der `[FPGA]Main.vi` platzierte `[FPGA]Druckkopf.vi` muss an die Anzahl der Druckköpfe und der Auflösung der Schieberegister angepasst werden, indem die Anzahl der auszulesenden 64-Bit Elemente aus dem FPGA-internen FIFO-Speicher definiert wird. Es müssen die zu verwendenden I/O Modul Ausgänge zum Beschreiben der Schieberegister und zum Feuern der Druckkopfdüsen sowie ein externer Auslöser angegeben werden.

5.1.3 Pumpen

Um die Ansteuerung mehrerer Pumpen über digitale Ausgänge zu unterstützen, müssen Kopien der `[FPGA]PWM.vi` mehrfach in die `[FPGA]Main.vi` eingebunden werden und der Quellcode jeder Kopie an den zu verwendenden I/O Modul Ausgang für die Impulserzeugung angepasst werden.

5.1.4 Erzeugen der FPGA-Bitfile

Wurde die `[FPGA]Main.vi` an die Sensoren und Aktoren der Druckanlage angepasst, sollte daraus mit Hilfe der LabVIEW Entwicklungsumgebung ein FPGA-Bitfile erzeugt und die fehlerfreie Ausführung anschließend getestet werden.

5.1.5 Sensordaten auslesen

Die Sensordaten müssen durch Auslesen der `[FPGA]Main.vi` Ausgänge und Einfügen in die RT-Queue Messwerte in Anlehnung an Kapitel 4.7 der PC- und RT-Ausführungsumgebung bereitgestellt werden. Das erfordert eine Anpassung der Typdefinition `CTD_Messwerte` an die benötigten Datentypen zur Erfassung der Sensorwerte.

5.2 Anpassung an die benötigten Regelungen

Je nachdem wie viele unabhängige Temperaturregelungen und Druckregelungen benötigt werden, können mehrere Kopien der `[RT]Temperierung.vi` bzw. `[RT]Druckregelung.vi` in die `[RT]Main.vi` platziert werden. Die Eingänge dieser VIs müssen anschließend mit den entsprechenden Stell- und

Messwerten aus der RT-Queue Stellwerte bzw. RT-Queue Messwerte verbunden werden. Die Typdefinition `CTD_Stellwerte` sollte dazu bereits die benötigten Stellwert- und `Auswahl`-Eingänge der Regelungs-VIs abdecken. Danach ist eine individuelle Anpassung an ausgewiesenen Stellen innerhalb des Quellcodes notwendig.
Die Kopien der `[RT]Temperierung.vi` müssen so konfiguriert werden, dass die richtigen Heizelemente über die FPGA-Eingänge ein- bzw. ausgeschaltet werden. Die Kopien der `[RT]Druckregelung.vi` erfordern eine Anpassung an die zu verwendenden Ventilstellungen und Pumpen für jede der fünf möglichen Verhaltensweisen und eine individuelle Kalibrierung [18] der Parameter der beiden PID-Regler für jedes Druckerzeugungssystem.

5.3 Anpassung des Minimalautomaten

Die Typdefinitionen `TD_Zustandsname` und `TD_Übergangsbefehl` müssen vor der Anpassung des Minimalautomaten mit den benötigten Zustandsnamen und Übergangsbefehlen zur Konfiguration der Zustände und Zustandsübergänge ergänzt werden.
Anschließend sollten neue Zustände und Zustandsübergänge durch das Kopieren bestehender Zustände und Zustandsübergänge des Minimalautomaten erzeugt werden, da die SCSYS-VIs darin bereits enthalten sind und die `[SCSYS]Entry.vi` bzw. die `[SCSYS]TransGuard.vi` gleich konfiguriert (4.4) werden können.
Damit jeder neue Zustand einen Übergang in die Zustände `SYS-Notaus` und `SYS-Störung` des Minimalautomaten durch Abbruchbefehle ermöglicht, sollte die Anpassung ausschließlich innerhalb des Bereiches „Betrieb“ (Abb. 43) vorgenommen werden.

5.3.1 Einbetten von Benutzerquellcode

Das Einbetten von Quellcode in den SCSYS-VIs erfolgt über korrespondierende SC-VIs, die innerhalb der SCSYS-VIs mit Informationen zum aktuellen Zustand bzw. Zustandsübergang aufgerufen werden und vom Anwendungsentwickler angepasst werden können.
Jede dieser SC-VIs besteht aus einer Case-Struktur, deren Cases das Verhalten eines Zustandes bzw. Zustandsübergangs repräsentieren.
Beim Betreten der `State Action` eines Zustandes wird zum Beispiel die `[SC]StateActions.vi` von der `[SCSYS]StateAction.vi` mit dem Zustandsnamen des aktuellen Zustandes aufgerufen. Diese führt dann in Abhängigkeit des übergebenen Zustandsnamen den entsprechenden Case der Case-Struktur

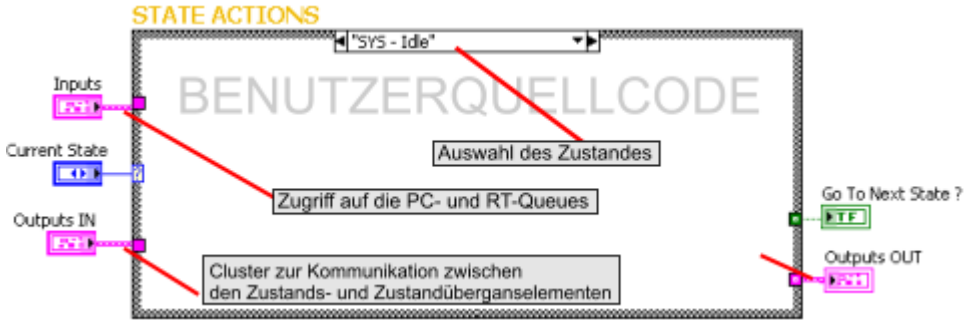

Abbildung 55: [SC]StateActions.vi zum Bearbeiten der `State Actions` von Zuständen.

aus (Abb. 55).
Der `[SC]TransGuards.vi` und `[SC]TransActions.vi` werden neben dem Namen des aktuellen Zustands auch die im `[SCSYS]TransGuard.vi` eingestellte „Richtung“[12] der Transition übergeben, um auch hier jedem[13] `Transition Guard` und jeder `Transition Action` über eine Case-Struktur Benutzerquellcode zuweisen zu können.
In jeder der SC-VIs hat der Anwendungsentwickler über ein Cluster von Queue-Referenzen Zugriff auf die PC- und RT-Queues.

5.4 Testen der Anwendung

Die Anwendung kann durch das Ausführen (2.4.4) der `[PC]Main.vi` und der `[RT]Main.vi` jederzeit getestet werden. Die LabVIEW Entwicklungsumgebung überträgt zur Ausführung der `[RT]Main.vi` über eine bestehende Verbindung zum Embedded Controller die dazu benötigten VIs auf das Echtzeitbetriebssystem und beschreibt den FPGA selbstständig mit der zuvor erzeugten FPGA-Bitfile.
Mit dem in der `[PC]Main.vi` enthaltenen Benutzerinterface für den Anwendungsentwickler kann anschließend das Verhalten des Zustandsautomaten getestet und gegebenenfalls korrigiert werden.

[12] Die Richtung entspricht einem weiteren Zustandsnamen, der in der `[SCSYS]TransGuard.vi` eingestellt wurde.

[13] Das setzt jedoch voraus, dass nur ein Übergang von Zustand A nach Zustand B existiert.

5.5 Benutzerinterface für den Endanwender

Wurde die Anwendung erfolgreich getestet, kann das Benutzerinterface für den Endanwender aufgebaut werden. Dazu kann das in Kapitel 4.5.1 vorgestellte Benutzerinterface in der `[PC]EndanwenderUI.vi` und der darin enthaltene Quellcode zum Aktivieren und Deaktivieren von Schaltflächen als Vorlage für den Aufbau verwendet werden.

5.6 Kompilieren der Anwendung

Der letzte Schritt in der Anwendungsentwicklung ist die Erzeugung einer von der LabVIEW Entwicklungsumgebung unabhängigen Applikation aus der `[PC]Main.vi`. Dazu kann der von LabVIEW bereitgestellte „Application Builder“ verwendet werden [7]. Ausserdem muss für den Embedded Controller eine sogenannte Real-Time-Applikation aus der `[RT]Main.vi` erstellt werden, die beim Einschalten des Embedded Controllers automatisch vom Echtzeitbetriebssystem ausgeführt wird. Hierzu kann das LabVIEW „Real-Time Modul“ verwendet werden [8].

6 Evaluation

Das Framework konnte nicht als Ganzes an eine 3D-Ink-Jet-Druckanlage angewendet werden, da die Hardware der Anlagen nicht innerhalb des Bearbeitungszeitraums dieser Diplomarbeit fertiggestellt werden konnte. Die einzelnen Komponenten des Frameworks wurden jedoch unabhängig voneinander an verschiedenen Versuchsständen getestet.

6.1 Kommunikation

Um die Kommunikation zwischen PC- und RT-Ausführungsumgebung zu testen, wurden die Cluster-Typdefinition `CTD_PC-Daten` und `CTD_RT-Daten` mit der gleichen Menge an unterschiedlichen Datentypen erweitert. Anschließend wurden Testwerte über die PC-Queue PC-Daten an den RT gesendet, die dann dort ausgelesen und über die RT-Queue RT-Daten zurück an den PC gesendet wurden. Die gesendeten und empfangenen Werte wurden dann miteinander verglichen.
Die Druckdatenübertragung wurde durch Einfügen mehrerer Arrays der Typdefinition `TD_Druckdaten` in die PC-Queue Druckdaten erfolgreich getestet. Die Integerwerte eines Arrays entsprachen dabei der Position, in der das Array in die Queue eingefügt wurde. Das zuerst eingefügte Array bestand beispielsweise aus lauter Einsen. Die Elemente PC-Queue Druckdaten wurden dann von einem Test-VI der RT-Ausführungsumgebung ausgelesen und die Integerwerte der Arrays aufsummiert. Das Ergebnis der Summation wurde dann mit dem erwarteten Ergebnis verglichen.
Um Verbindungsabbrüche zu simulieren wurde das Ethernet-Kabel am Embedded Controller ausgesteckt und nach einiger Zeit wieder eingesteckt. Dabei verfiel die `[PCRT]Kommunikation.vi` zurück in den Verbindungsaufbau und nahm nach dem Einstecken des Ethernet-Kabels die Verbindung erfolgreich wieder auf.

6.2 Benutzerinterface und Zustandsautomat

Die Funktion der SCSYS-VIs wurde durch den Aufbau eines Zustandsautomaten (Abb. 56) für die in Kapitel 3.1 vorgestellte Keramikdruckanlage getestet. Da manche Zustände erst mit gewissen Sensorwerten einen Übergang erlauben sollten, wurde die Typdefinition `CTD_PC-Daten` mit Datentypen erweitert, die der Simulation von Sensordaten dienten. Beispielsweise musste ein Übergang von Zustand `Aufheizphase` in den Zustand `Print Ready` aus Abbildung 56 erst möglich sein, wenn eine bestimmte Temperatur überschritten wurde.

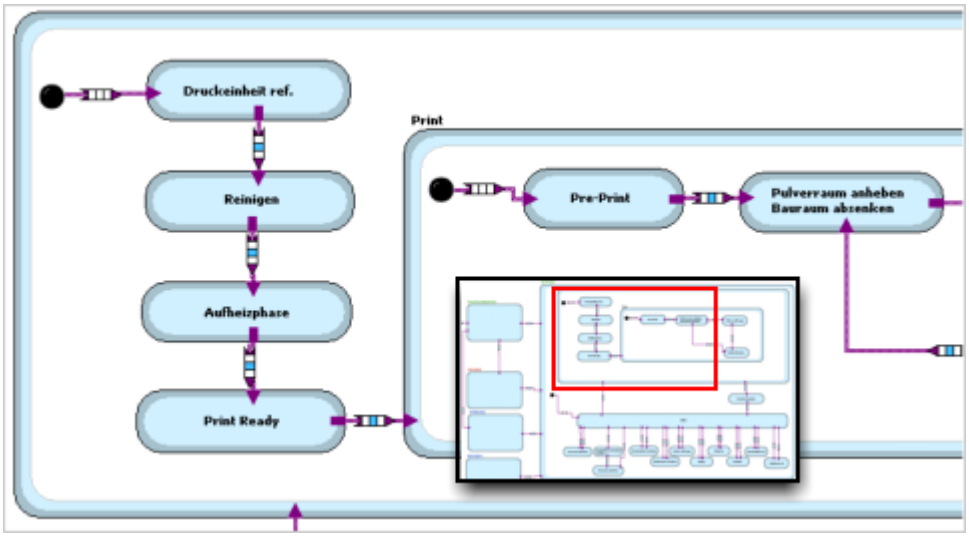

Abbildung 56: Ausschnitt des Zustandsautomaten zur Steúerung der Keramikdruckanlage

Mit diesem Zustandsautomaten wurde die korrekte Ausführung von Übergangsbefehlen, Befehlsketten, Abbruchbefehlen und das Überspringen von Zuständen überprüft. Dabei wurden die Statusmeldungen und Befehlsantworten mit den erwarteten Meldungen verglichen. Zur Steuerung dieses Zu-

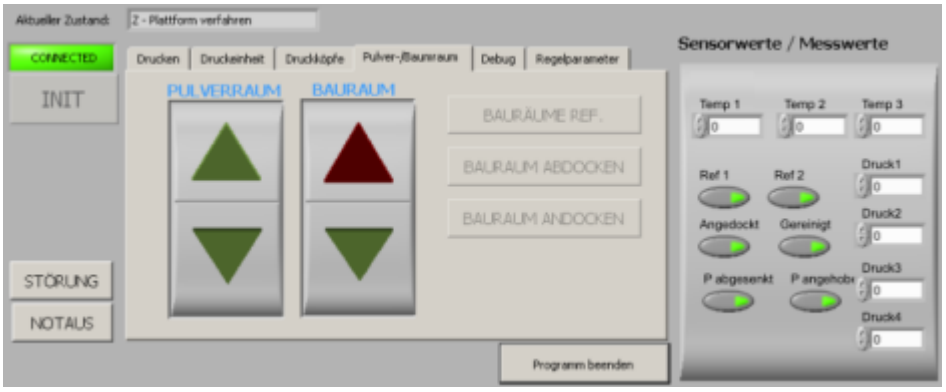

Abbildung 57: Benutzerinterface zur Steuerung der Keramikdruckanlage

standsautomaten und um das Aktiveren und Deaktivieren von Schaltflächen in Abhängigkeit der möglichen Übergänge eines Zustandes zu testen, wurde ein Benutzerinterface zur Steuerung der Keramikdruckanlage umgesetzt (Abb. 57).

6.3 Druckregelung

Die Druckregelung wurde an einem Versuchsstand zum Testen von Druckköpfen der Firma TRIDENT angewendet. Die Ansteuerung der Druckköpfe wurde dabei von einer TRIDENT-Steuerungshardware [19] übernommen, weshalb für diesen Versuchsstand kein Embedded Controller eingesetzt wurde. Die `[RT]Druckregelung.vi` wurde auf einem LabVIEW-kompatiblem Computersystem ausgeführt, das über ein USB-Mess-Modul von Meilhaus [11] Zugriff auf den Drucksensor, die Pumpe und die drei Ventilen des Versuchsstandes hatte.
Da die `[RT]Druckregelung.vi` dafür auslegt war, die Aktoren und Sensoren über den FPGA zu steuern bzw. auszulesen, mussten Modifikationen an diesem VI vorgenommen werden, um eine Steuerung der Pumpen und Ventilstellungen sowie das Auslesen der Drucksensordaten über USB zu ermöglichen. Das `[FPGA]PWM.vi` konnte hier nicht getestet werden, da keine FPGA-Ausführungsumgebung zur Verfügung stand.
Um die Regelparameter an den Versuchsstand anzupassen, wurde ein LabVIEW-Signalverlaufsgraph [10] verwendet, mit dem der Verlauf des geregelten Unter- und Überdrucks für verschiedene Regelparameter getestet und angepasst wurde. Neben dem in Kapitel 4.9 beschriebenem Problem beim Wechsel von der Überdruck- zur Unterdruckerzeugung, konnte die Versorgung und Reinigung des Druckkopfes mit Flüssigkeit erfolgreich durchgeführt werden.

6.4 Temperaturregelung

Die Temperaturregelung wurde an einem von einer Heizfolie umschlossenen Temperatursensor getestet. Dazu musste die `[RT]Temperierung.vi` ähnlich wie die `[RT]Druckregelung.vi` in Kapitel 6.3 modifiziert werden, da die Heizfolie und der Temperatursensor über USB-Mess-Modul von Meilhaus angesteuert bzw. auslesen wurde. Mit diesem Versuchsaufbau konnte die Regelung unterschiedlicher Temperaturen erfolgreich durchgeführt werden.

6.5 Schrittmotorsteuerung

Die Schrittmotorsteuerung wurde an einem Schrittmotor der Firma Trinamic [20] erfolgreich getestet. Dazu wurden die `ENABLE`, `STEP` und `DIRECTION` Eingänge der Treiberplatine des Schrittmotors mit digitalen Ausgängen eines I/O Moduls verbunden und die `[FPGA]Schrittmotor.vi` in der `[FPGA]Main.vi` entsprechend angepasst.
Es wurden verschiedene Beschleunigungs- und Abbremsrampen gefahren und dabei die Anzahl der Schritte bei maximaler Geschwindigkeit variiert. Mit

einem Impulszähler wurden die erzeugten STEP-Impulse gezählt und mit der erwarteten Anzahl verglichen.

6.6 Druckkopfansteuerung

Die Druckkopfansteuerung wurde durch Unterstützung der Mitarbeiter des Fraunhofer IPA mit einem Versuchsaufbau getestet. Neben einem Embedded Controller mit integriertem I/O Modul bestand der Versuchsaufbau aus einem 4-Kanal-Oszilloskop, einem Impulsgenerator sowie einem Impulszähler (Abb. 58). Während der Impulsgenerator als Trigger-Signal für das Feuern der Druckkopfdüsen diente, wurden mit dem Impulszähler die Anzahl der Impulse der DATA-Leitung gezählt. Das Oszilloskop wurde dabei zur Impuls-Visualisierung genutzt.

Zum Testen der Druckkopfansteuerung wurden unterschiedliche periodische

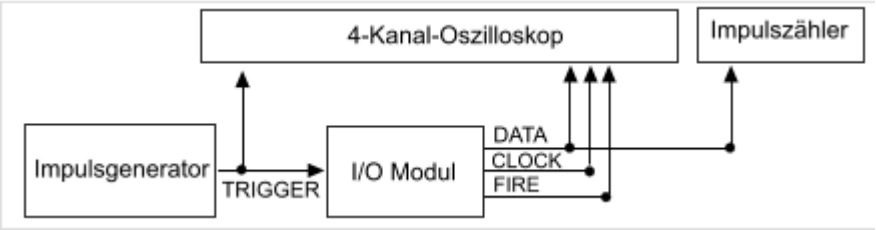

Abbildung 58: Schematischer Aufbau des Versuchsstands zum Testen der Druckkopfansteuerung

Bitmuster erzeugt, damit diese als periodische Ausgabe am Oszilloskopen beobachtet werden können. Die Länge der Druckdaten wurde dabei so gewählt, dass die erzeugten Impulse mehrere Sekunden lang am Oszilloskopen beobachtet werden können. Die Druckkopfansteuerung wurde mit den Bit-

Abbildung 59: Beispiel für periodische Testdruckdaten

mustern erfolgreich getestet. Bei den Versuchen wurde darauf geachtet, dass die FIRE-Impulse synchron zu den TRIGGER-Signalen erzeugt werden. Außerdem musste am Oszilloskopen ein zu den CLOCK-Impulen synchroner und dem Bitmuster entsprechendem Signalverlauf der DATA-Leitung sichtbar sein. Ein weiteres Kriterium war die Anzahl der erzeugten DATA-Impulse, die mit Hilfe des Impulszählers gezählt und mit den Druckdaten verglichen wurden.

7 Zusammenfassung und Ausblick

Ziel dieser Diplomarbeit war es, ein Framework zur Steuerung von prototypischen 3D-Ink-Jet-Druckanlagen auf Basis von National Instruments Embedded Controllern zu entwickeln, um die Realisierung einer Maschinensteuerung zu beschleunigen.
Die Anforderungen an das Framework wurden durch die Analyse einer Druckanlage zur Herstellung von keramischen Bauteilen in Zusammenarbeit mit den Mitarbeitern des Fraunhofer IPA ermittelt.
Neben der Ansteuerung von Schrittmotoren und Druckköpfen sowie der Regelung von Temperaturen und Drücke, waren die Repräsentation des Verhaltens einer Druckanlage durch einen Zustandsautomaten und dessen Steuerung durch ein Benutzerinterface Hauptanforderungen, die mit dem Framework erfüllt werden mussten.
Die Umsetzung des Frameworks sollte außerdem mit der LabVIEW Entwicklungsumgebung geschehen, da das Echtzeitbetriebssystem und der FPGA der zur Verfügung stehenden Embedded Controller nur darüber programmiert werden können.
Mit dem Framework wurde die Kommunikation zwischen Computersystemen und Embedded Controllern abgedeckt und der Zugriff auf Aktoren sowie das Auslesen von Sensordaten ermöglicht. Es wurde ein vom Anwendungsentwickler anzupassender Zustandsautomat auf dem Echtzeitbetriebssystem realisiert, der auf empfangene Übergangsbefehle und Abbruchbefehle reagiert und Informationen über seinen aktuellen Zustand sowie den möglichen Zustandsübergängen an das Computersystem weitergibt. Der Zustandsautomat lässt sich dabei über ein Benutzerinterface steuern, das auch der Visualisierung von empfangenen Sensordaten und Statusmeldungen des Zustandsautomaten dient. Außerdem wurden Komponenten zur Steuerung von Schrittmotoren, Druckköpfen und Pumpen sowie zur Regelung von Temperaturen und Drücke umgesetzt, die vom Anwendungsentwickler an die jeweilige Druckanlage angepasst und mehrfach verwendet werden können.
Eine mögliche Fortsetzung dieser Arbeit könnte in der Umwandlung von 3D-CAD-Datenformate in ein zur Druckkopfansteuerung kompatibles Format bestehen. Dazu müsste das Benutzerinterface so erweitert werden, dass mittels einer Dateiauswahl eine oder mehrere zu druckende CAD-Dateien selektiert werden können. Die durch die Dateien repräsentierten Objekte könnten dann vom Endanwender zunächst in einen virtuellen Bauraum ihren Abmessungen nach angeordnet und anschließend entsprechend formatiert an die Maschinensteuerung gesendet werden.

Abkürzungsverzeichnis

CAD ..... Computer-aided design
DMA ..... Direct Memory Access
DOD ..... Drop on Demand
Enum ..... Enumerated Type
FIFO ..... First In First Out
FPGA ..... Field Programmable Gate Array
GPL ..... General Purpose Language
IKTS ..... Institut für Keramische Technologien und Systeme
IPA ..... Institut für Produktionstechnik und Automatisierung
LIFO ..... Last In First Out
LUT ..... Look-Up Table
MATLAB ..... Matrix Laboratory
MIT ..... Massachusetts Institute of Technology
NI ..... National Instruments
PCI ..... Peripheral Component Interconnect
RT ..... Real Time
SubVI ..... Untergeordnetes Virtuelles Instrument
UML ..... Unified Modeling Language
VI ..... Virtuelles Instrument

Abbildungsverzeichnis

1 Prinzip der generativen Fertigung [4] 5
2 Grundelemente des 3D-Printing Verfahrens [21] 7
3 3D-Printing Druckprozess [21] 7
4 Prinzip eines ContinuousInkJet-Druckkopfes 9
5 Prinzip eines BubbleJet-Druckkopfes [4] 9
6 Prinzip eines Piezo-Druckkopfes [4] 10
7 Schematische Darstellung eines Thermoelementes [30] 11
8 Prinzip einer Membranpumpe, links: Saugstellung, rechts: Pumpstellung 12
9 Schematischer Aufbau und Funktionsweise eines Schrittmotors 13
10 NI Embedded Controller: CompactRio 9014 mit vier I/O Modulen [9] 14
11 Links: Schematischer Aufbau eines FPGA,Rechts: Beispielhafter Basisblock [25] 15
12 (A+B)xC.vi - Bedienoberfläche (Frontpanel) 17
13 (A+B)xC.vi - Blockdiagramm 18
14 Parallele Ausführung zweier While-Schleifen 19
15 Blockdiagramm mit SubVI 20
16 Bedienelemente zur Programmausführung 20
17 Ausführung im Debug-Modus, siehe zum Vergleich ohne Debug-Modus Abb.14 21
18 Erzeugung und Verwendung eines Clusters, links: Haupt-VI, rechts: SubVI (A+B)xC.vi mit Cluster 22
19 Beispiele für den Enum-Datentyp 22
20 Zustandsautomat in der Zustandsdiagramm-Ansicht 24
21 Ausführungsreihenfolge der Elemente eines Zustandes mit n Zustandsübergängen. 25
22 Zustandsautomat eingebunden in der While-Schleife eines VIs 26
23 Links:Domänenbindung, Rechts: Hot / Frozen Spots 27
24 Invertierter Kontrollfluss in Frameworks 28
25 Keramikdruckanlage mit Einteilung in drei Bereiche. 29
26 Aufbau des Druckwerks und der Bauebene. 30
27 Übersicht: Ausführungsumgebungen des Frameworks 35
28 Komponenten der PC-Ausführungsumgebung 36
29 Komponenten der RT- und FPGA-Ausführungsumgebung . . 38
30 LabVIEW-Projektübersicht des Frameworks 40
31 Abhängigkeit der Queues von den Typdefinitionen 42
32 Abhängigkeit der Queues von den Typdefinitionen 43

33 Ausschnitt aus dem [PC]Main.vi - Initialisierung der Queues und Übergabe der Queue-Referenzen als Cluster an das [PCRT]Kommunikation.vi mit PC-Verhalten. 44
34 SCSYS-VIs in einem Zustand mit n Übergängen. 46
35 Konfiguration eines Zustandes (links) und Zustandsübergangs (rechts) 47
36 Beispiel zum Senden einer Statusmeldung 48
37 Aufbau des Übergangsbefehls-Arrays 48
38 Klassisches Zustandsverhalten 49
39 Spezielles Zustandsverhalten (GotoNextState=FALSE) 50
40 Beispiel zur Ausführung eines Übergangsbefehls durch mehrere Zustände hindurch. 51
41 Beispiel zur Reaktion auf eine Befehlskette 51
42 Beispiel für autonomes Verhalten 54
43 Minimalautomat in der LabVIEW-Zustandsdiagrammansicht . 55
44 Benutzerinterface für den Anwendungsentwickler 55
45 Benutzerinterface für den Endanwender im Zustand `Idle` des Minimalautomaten 56
46 Beispiel-Ausschnitt der [FPGA]Main.vi: Sensoren und Aktoren Mapping 57
47 Auslesen der Sensordaten in der [RT]Main.vi 58
48 [RT]Temperierung.vi 59
49 [RT]Druckregelung.vi 59
50 Beispiel zum Beschreiben der Beschleunigungs- und Abbremsungs-LUT 61
51 Druckkopf mit 64-Bit Schieberegister 63
52 [RT]WriteToDMA.vi und [FPGA]Druckkopf.vi 63
53 Verhalten der [RT]WriteToDMA.vi 64
54 Ausschnitt aus dem [FPGA]Schrittmotor.vi mit markierten Anpassungsstellen 66
55 [SC]StateActions.vi zum Bearbeiten der `State Actions` von Zuständen. 69
56 Ausschnitt des Zustandsautomaten zur Steúerung der Keramikdruckanlage 72
57 Benutzerinterface zur Steuerung der Keramikdruckanlage . . . 72
58 Schematischer Aufbau des Versuchsstands zum Testen der Druckkopfansteuerung 74
59 Beispiel für periodische Testdruckdaten 74

Literatur

[1] DocForge: *Framework.* Webseite. http://docforge.com/wiki/Framework, 15 August 2010.

[2] Dujmovic, Stjepan: *Anwendungsentwicklung mit Komponenten- Frameworks in der Automatisierungstechnik.* Doktorarbeit, Universität Stuttgart - Fakultät Elektrotechnik und Informationstechnik, 2001.

[3] Fink, Andreas: *Enzyklopaedie der Wirtschaftsinformatik.* Webseite. http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/is-management/Systementwicklung/Hauptaktivitaten-der-Systementwicklung/Softwareentwurf/Framework-basierter-Softwareentwurf, 15 August 2010.

[4] Gebhardt, Andreas: *Generative Fertigungsverfahren.* Carl Hanser Verlag, 2007.

[5] Instruments, National: *Anwendungsentwicklung mit dem LabVIEW Statechart Module.* Webseite. http://zone.ni.com/devzone/cda/tut/p/id/6928, 13 September 2010.

[6] Instruments, National: *Introduction to UML Terminology in the LabVIEW Statechart Module.* Webseite. http://zone.ni.com/devzone/cda/tut/p/id/7413, 13 September 2010.

[7] Instruments, National: *LabVIEW Application Builder.* Webseite. http://zone.ni.com/devzone/cda/tut/p/id/3303, 07 November 2010.

[8] Instruments, National: *LabVIEW Real-Time Modul.* Webseite. http://sine.ni.com/nips/cds/view/p/lang/de/nid/13742, 07 November 2010.

[9] Instruments, National: *NI CompactRIO: Steuer-, Regel- und Erfassungssystem.* Webseite. http://www.ni.com/compactrio/d/whatis.htm, 13 August 2010.

[10] Instruments, National: *Signalverlaufsgraph.* Webseite. http://www.ni.com/labview/whatis/user-interface/d/, 4 November 2010.

[11] Meilhaus: *Meilhaus Redlab Datenblatt.* Datenblatt. http://www.meilhaus.de/fileadmin/upload/pdf/Datasheets_deutsch/redlab.pdf, 4 November 2010.

[12] MIKROCONTROLLER.NET: *Pulsweitenmodulation.* Webseite. http://www.mikrocontroller.net/articles/Pulsweitenmodulation, 18 Oktober 2010.

[13] MIT.EDU: *What is Three Dimensional Printing?* Webseite. http://www.mit.edu/~tdp/whatis3dp.html, 1 November 2010.

[14] MÜTTERLEIN, BERNWARD: *Handbuch für die Programmierung mit LabVIEW.* Spektrum Akademischer Verlag, 2007.

[15] PARTHIER, RAINER: *Messtechnik.* Vieweg, 2008.

[16] RALPH E. JOHNSON, BRIAN FOOTE: *Designing Reusable Classes.* Journal of Object- Oriented Programming, 1(2):22–35, 1988.

[17] RICK BITTER, TAQI MOHIUDDIN, MATT NAWROCKI: *LabVIEW Advanced Programming Techniques.* CRC Press, 2007.

[18] RN-WISSEN.DE: *Regelungstechnik.* Webseite. http://www.rn-wissen.de/index.php/Regelungstechnik, 15 Oktober 2010.

[19] TRIDENT: *Trident PixelJet128 Datenblatt.* Datenblatt. http://www.trident-itw.com/Portals/3/PDFs/PixelJet128_InfoSheet.pdf, 4 November 2010.

[20] TRINAMIC: *Trinamic PANdrive PD-013-42.* Webseite. http://www.trinamic.com/tmc/render.php?sess_pid=258, 4 November 2010.

[21] VISOTSCHNIG, ROLAND: *Medienversorgung für einen 3D-Keramikdrucker.* Studienarbeit. Universität Stuttgart - ISW, Fraunhofer IPA.

[22] WIKIPEDIA: *Benutzerschnittstelle.* Webseite. http://de.wikipedia.org/wiki/Benutzerschnittstelle, 18 Oktober 2010.

[23] WIKIPEDIA: *Drucksensor.* Webseite. http://de.wikipedia.org/wiki/Drucksensor, 10 August 2010.

[24] WIKIPEDIA: *Echtzeit.* Webseite. http://de.wikipedia.org/wiki/Echtzeit, 04 September 2010.

[25] WIKIPEDIA: *FPGA.* Webseite. http://de.wikipedia.org/wiki/Fpga, 13 August 2010.

[26] WIKIPEDIA: *Heizelement.* Webseite. http://de.wikipedia.org/wiki/Heizelement, 08 August 2010.

[27] WIKIPEDIA: *Membranpumpe*. Webseite. http://de.wikipedia.org/wiki/Membranpumpe, 10 August 2010.

[28] WIKIPEDIA: *Regler*. Webseite. http://de.wikipedia.org/wiki/Regler, 27 Oktober 2010.

[29] WIKIPEDIA: *Schrittmotor*. Webseite. http://de.wikipedia.org/wiki/Schrittmotor, 08 August 2010.

[30] WIKIPEDIA: *Thermoelement*. Webseite. http://de.wikipedia.org/wiki/Thermoelement,08 August 2010.

[31] WIKIPEDIA: *Tintenstrahldrucker*. Webseite. http://de.wikipedia.org/wiki/Tintenstrahldrucker,06 Juli 2010.

[32] WIKIPEDIA: *Ventil*. Webseite. http://de.wikipedia.org/wiki/Ventil, 08 August 2010.

[33] WIKIPEDIA: *Viskosität*. Webseite. http://de.wikipedia.org/wiki/Viskosität,13 August 2010.

[34] WIKIPEDIA: *Zustandsdiagram (UML)*. Webseite. http://de.wikipedia.org/wiki/Zustandsdiagramm, 13 September 2010.

[35] WIKIPEDIA: *Zweipunktregler*. Webseite. http://de.wikipedia.org/wiki/Heizungsregler, 08 August 2010.

Erklärung

Hiermit versichere ich, diese Arbeit
selbständig verfasst und nur die
angegebenen Quellen benutzt zu haben.

(Lucian Cristian Depold)