Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterthesis

Safa Camur

Synthese und Implementierung einer ereignisdiskreten Puffersteuerung für eine flexible Montageanlage

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Safa Camur

Synthese und Implementierung einer ereignisdiskreten Puffersteuerung für eine flexible Montageanlage

Masterthesis eingereicht im Rahmen der Masterprüfung
im Masterstudiengang Automatisierung
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Florian Wenck
Zweitgutachter : Prof. Dr.-Ing. Ulfert Meiners

Abgegeben am 05.12.2017

Safa Camur

Thema der Masterthesis

Synthese und Implementierung einer ereignisdiskreten Puffersteuerung für eine flexible Montageanlage

Stichworte

Ereignisdiskrete Systeme (DES), Supervisory Control Theory (SCT), Steuerungssynthese, SPS, Automaten, Generatoren, Formale Sprachen, TCT, DESTool, ACArrow, TIA-Portal

Kurzzusammenfassung

Diese Masterarbeit beinhaltet die Darstellung der Synthese einer Puffersteuerung für eine flexible Montageanlage. Dafür wird die Anlage als ereignisdiskret betrachtet. Die Steuerung wird nach der Supervisory Control Theory synthetisiert. Zur Implementierung wird ein Codegenerator verwendet, der den Code für eine speicherprogrammierbare Steuerung generiert. Abschließend wird die Implementierung validiert.

Safa Camur

Title of the paper

Synthesis and implementation of a discrete-event buffer-controller for a flexible assembly system

Keywords

Supervisory Control Theory (SCT), Discrete event systems, Control synthesis, PLC, Automata, Formal language, TCT, DESTool, ACArrow, TIA-Portal

Abstract

Inside this report the synthesis of a buffer-controller for a flexible assembly system is described. Therefore, the assembly system is considered as a discrete event system. The controller synthesized according to the supervisory control theory. For the implementation uses a code generator that generates the program code for a programmable logic controller. Finally, the implementation is validated.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

δ ..... Zustandsübergangsfunktion
ϵ ..... Leerer String
$\overline{L}$ ..... Präfix-Hülle der Sprache L
$\|$ ..... Parallele Komposition zweier Generatoren
Σ ..... Ereignisalphabet
σ ..... Ereignis
Σ^* ..... Kleene-Hülle
Σ_c ..... Menge der steuerbaren Ereignisse
Σ_o ..... Menge der beobachtbaren Ereignisse
Σ_{ce} ..... Menge der gemeinsamen Ereignisse
Σ_{pe} ..... Menge der privaten Ereignisse
Σ_{uc} ..... Menge der nicht steuerbaren Ereignisse
Σ_{uo} ..... Menge der nicht beobachtbaren Ereignisse
$\times$ ..... Produktkomposition zweier Generatoren
A ..... Adjazenzmatrix
$Ac(G)$ ..... Erreichbare Teil des Generators G
$cat(s,t)$ ..... Konkatenation der Strings s und t
$CoAC(G)$ ..... Ko-Erreichbare Teil des Generators G
G ..... Generator/Strecke/Plant
$G_\times$ ..... Produktgenerator
K ..... Formale Spezifikation
$K^\downarrow$ ..... Infimale präfix-abgeschlossene steuerbare Obersprache
$K^\uparrow$ ..... Supremale steuerbare Teilsprache
L ..... Formale Sprache
$L(G)$ ..... Reguläre Sprache von G
$L_m(G)$ ..... Markierendes Verhalten von G
P ..... Natürliche Projektion
P^{-1} ..... Inverse natürliche Projektion

S ..... Supervisor/Steuerung
s ..... String
S/G ..... Gesteuertes System (G gesteuert von S)
$Trim(G)$ ..... Erreichbarer und ko-erreichbarer Teil des Generators G
X ..... Zustandsmenge
x ..... Zustand
x_0 ..... Initialzustand
X_m ..... Menge der markierten Zustände
x_m ..... Markierter Zustand
A ..... Ausgang der SPS
AWL ..... Anweisungsliste
BSCOP-NB ..... Basic Supervisory Control and Observation Problem-Nonblocking Case
BSCP-NB ..... Basic Supervisory Control Problem-Nonblocking Case
c ..... controllable
DES ..... Discrete Event Systems
DFA ..... Deterministic finite automaton
DI ..... Digital Input
DO ..... Digital Output
E ..... Eingang der SPS/Nicht steuerbares Ereignis in der SCT
FUP ..... Funktionsplan
HMI ..... Human Machine Interface
KOP ..... Kontaktplan
M ..... Merker der SPS
NFA ..... Nondeterministic finite automaton
PLC ..... Programmable logic controller
PS ..... Produktsystem
R ..... Robotereinheit
RFID ..... Radio-frequency identification
ROB ..... Robotereinheit
SCL ..... Structured Control Language
SCT ..... Supervisory Control Theory
SPC ..... Strict Product Composition
SPS ..... Speicherprogrammierbare Steuerung
ST10 ..... Leitstation/Station 10

ST20 ..... Lagerstation/Station 20
ST30 ..... Montagestation/Station 30
ST40 ..... Elektrische Endprüfstation/Station 40
ST50 ..... Pneumatische Presse/Station 50
ST60 ..... Optische Endprüfstation/Station 60
SYPC ..... Synchronous Product Composition
TIA ..... Totally Integrated Automation
TS1 ..... Werkstückpaletten-Transfersystem
uc ..... uncontrollable

1. Einführung und Zielsetzung

1.1. Einführung und Motivation

In der industriellen Praxis werden Steuerungen üblicherweise intuitiv entworfen, ohne dass Streckenmodelle und Prozessbeschreibungen zugrunde gelegt werden. Die intuitiv entworfenen Steuerungen können bei größeren industriellen Anlagen sehr unübersichtlich und fehleranfällig werden. Die Verifizierung der Korrektheit dieser Steuerungen erfolgt in vielen Fällen auf der zu steuernden Anlage während einer Inbetriebnahmephase und anhand verschiedener Testszenarien. Zudem ist die verifizierte Korrektheit nur für getestete Szenarien sichergestellt. Des Weiteren ist die Korrektur einer Fehlfunktion meistens mit sehr großem Aufwand verbunden.

In den letzten Jahren haben synthetisierte Steuerungen in der industriellen Praxis zunehmend an Bedeutung gewonnen. Eine modellbasierte Steuerungssynthese basiert auf formalen Beschreibungen des ungesteuerten Systemverhaltens und auf einer Menge formaler Spezifikationen für das geforderte Systemverhalten. Die daraus generierte Steuerung ist beweisbar korrekt. Des Weiteren können durch rechnergestützten Entwurf zahlreiche Schritte bei der Steuerungssynthese automatisiert und menschliche Fehler weitestgehend reduziert werden. So können auch nachträgliche Änderungen oder Erweiterungen im Entwicklungsprozess schneller berücksichtigt werden. Die Motivation dieser Arbeit liegt darin, mit Hilfe der Supervisory Control Theory von P. J. Ramadge und W. M. Wonham [22, 31] eine ereignisdiskrete Puffersteuerung für eine flexible Montageanlage zu synthetisieren.

Die Arbeit wird in 8 Kapitel strukturiert. Nach der Einleitung in Kapitel 1 werden in Kapitel 2 für das weitere Verständnis dieser Arbeit die Grundlagen zu ereignisdiskreten Systemen, formalen Sprachen, Automaten und die Supervisory Control Theory erläutert. Die verwendete Montageanlage wird in Kapitel 3 vorgestellt. In Kapitel 4 wird die Konzeption vorgestellt. Kapitel 5, 6 und 7 beinhalten die konkrete Umsetzung der Konzeptionen. Die ungesteuerte Strecke wird in Kapitel 5 modelliert und auf dieser Grundlage die Spezifikationen im Kapitel 6 entwickelt. Die Synthese von Supervisor aus Modellen der Strecke und der Spezifikationen ist auch Inhalt von Kapitel 6. Kapitel 7 gibt einen Einblick in die Implementierung und Realisierung. Zudem werden hier die Betriebsergebnisse präsentiert. In Kapitel 8 wird die Arbeit zusammengefasst und mit einem Ausblick abgeschlossen.

1.2. Aufgabenstellung

Im Jahr 2016 wurde an der Hochschule für Angewandte Wissenschaften im Fachbereich Automatisierungstechnik eine vollautomatisierte flexible Montageanlage installiert. Die Anlage, die von der Firma Köster Systemtechnik GmbH stammt, bildet einen kompletten vollautomatischen Montage- und Prüfablauf einer realen Fabrik nach. Sie besteht aus unterschiedlichen industrieüblichen Komponenten. Als Produkt wird eine Relaiskarte im Gehäuse mit Deckel montiert und geprüft. Das eingesetzte Relais wird zum Beispiel in Windkraftanlagen eingesetzt. Im Rahmen dieser Masterarbeit ist für diese Anlage eine ereignisdiskrete Puffersteuerung nach Supervisory Control Theory von P. J. Ramadge und W. M. Wonham zu synthetisieren und zu implementieren. Die Aufgabenstellung beinhaltet die gesamte Montageanlage.

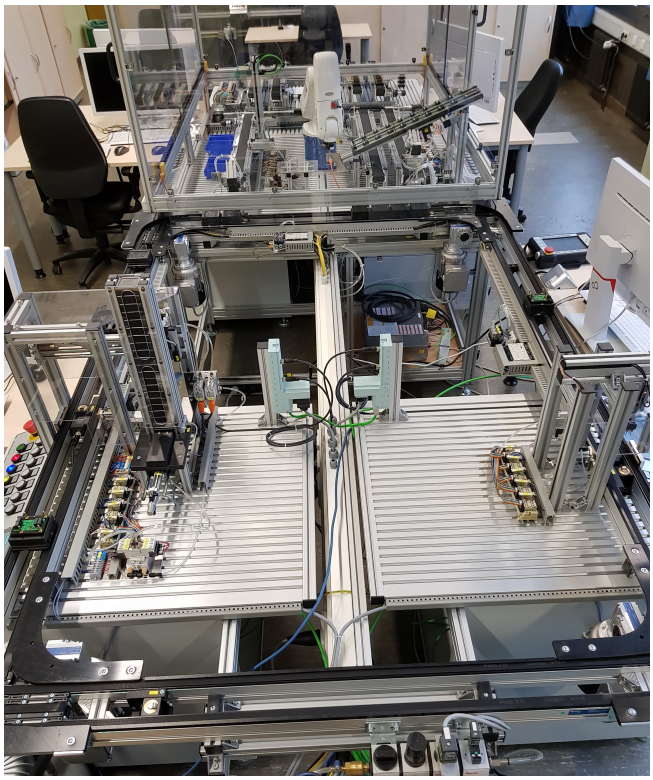


Abbildung 1.1.: Gesamtansicht der Montageanlage.

Die Anlage besteht aus mehreren Stationen auf denen jeweils unterschiedliche Montagen und Prüfungen vorgenommen werden. Die Stationen sind durch Pufferstrecken endlicher Kapazität miteinander gekoppelt. Die in dieser Arbeit zu entwerfende Puffersteuerung hat die Aufgabe, die Anzahl der Werkstücke in den einzelnen Puffern zu steuern und deren

Über- und Unterlauf zu verhindern. Des Weiteren kann die Anlage an ihre Belastungsgrenze stoßen und blockieren, wenn eine bestimmte Bestandsgrenze erreicht oder überschritten wird. Die zu entwerfende Steuerung soll zudem sicherstellen, dass die Belastungsgrenze der Anlage niemals überschritten wird.

Für den Steuerungsentwurf soll das Verhalten der ungesteuerten Strecke ereignisdiskret modelliert werden. Parallel zur Modellierung der ungesteuerten Strecke sollen Spezifikationen formal entworfen werden. Aus diesen Modellen der ungesteuerten Strecke und der Spezifikationen soll die Puffersteuerung synthetisiert werden. Für den Steuerungsentwurf mit Methoden der Supervisory Control Theory stehen verschiedene Entwurfsansätze zur Auswahl. Es sind geeignete Ansätze, Beschreibungsformen und Tools auszuwählen und anzuwenden.

In [6] wurde ein Codegenerator entwickelt. Mit Hilfe dieses Codegenerators und aus den Ergebnissen der durchzuführenden Synthese soll ein Programmcode generiert werden. Mit dem generierten Code soll anschließend das Programm auf einer Siemens-Steuerung realisiert und in das vorhandene Steuerungsprogramm integriert werden. Nach dieser Implementierung soll die Puffersteuerung ausgiebig getestet werden.

1.3. Zielsetzung

Das Ziel dieser Arbeit liegt darin, mit einer auf der Steuerungssynthese von P. J. Ramadge und W. M. Wonham beruhenden Methodik, eine ereignisdiskrete Puffersteuerung für eine flexible Montageanlage zu synthetisieren und zu implementieren. Diese soll die Aktionen der vorhandenen Steuerung einschränken oder erweitern, sodass die Anforderungen der Puffersteuerung zu jeder Zeit erfüllt werden.

2. Grundlagen der ereignisdiskreten Systemtheorie

In diesem Kapitel wird die Theorie der ereignisdiskreten Systeme (engl. discrete event systems, DES) und der Supervisory Control Theory (SCT) behandelt. Zunächst werden die sprachentheoretischen Grundlagen und die Grundlagen der Automaten erläutert. Darauf aufbauend werden Automaten für die Systembeschreibung näher beschrieben. Anschließend werden die theoretischen Grundlagen der Supervisory Control Theory vorgestellt und verschiedene Steuerungsentwurfsansätze gezeigt. Am Ende dieses Kapitels werden die verwendeten Softwaretools kurz beschrieben.

2.1. Ereignisdiskrete Systeme

Ein System ist eine Kombination aus mehreren miteinander verknüpften Komponenten, die zusammen eine bestimmte Funktion erfüllen. Es gibt unterschiedliche Klassen von Systemen. In dieser Arbeit wird nur die Klasse ereignisdiskreter Systeme mit diskreten Zustandsräumen betrachtet. Ein ereignisdiskretes System ist ein dynamisches, nichtlineares und zeitinvariantes System [2, S.46]. Dessen Signale haben einen diskreten Wertebereich. Nicht immer sind Systeme von Natur aus ereignisdiskret, können aber dennoch als ereignisdiskret betrachtet werden. So kann beispielsweise die kontinuierliche Bewegung eines Aufzugs abstrahiert und durch Folgen von diskreten Signalwerten abgebildet werden. Ein DES wird nach [30] folgendermaßen definiert:

Definition 2.1 (Ereignisdiskretes System, DES) *Ist ein dynamisches System mit einem diskreten Zustandsraum, dessen Zustände sich spontan durch das asynchrone Auftreten von Ereignissen über die Zeit ändern (ereignisgesteuert). Ein Ereignis ist eine Erscheinung oder eine Aktion ohne zeitliche Dauer.*

Ein DES ist dynamisch, weil der Zustand, der nach dem Auslösen eines Ereignisses eingenommen wird, von dem aktuellen Zustand und von dem ausgelösten Ereignis abhängt.

Das nichtlineare Verhalten erkennt man am diskontinuierlichen Auftreten der Ereignisse. Sie können zu jedem beliebigen Zeitpunkt auftreten [13][12].

Modelle für ereignisdiskrete Systeme heißen ereignisdiskrete Modelle. Man unterscheidet zwischen logischen und zeitbewerteten ereignisdiskreten Modellen. Die logischen Modelle beschreiben das Verhalten eines ereignisdiskreten Systems durch eine Folge von Ereignissen, die von dem System generiert werden. Die Zeitpunkte der auftretenden Ereignisse, werden in einem logischen Modell nicht berücksichtigt. Die zeitbewerteten Modelle dagegen geben zusätzlich zur Auftrittsreihenfolge der Ereignisse Auskunft über die exakten Zeitpunkte, an denen die Ereignisse auftreten. Neben solchen Modellen existieren auch stochastische und zeitbewertete stochastische Modelle, die Aussagen über die Auftrittswahrscheinlichkeit bestimmter Ereignisse oder Ereignisfolgen in nicht deterministischen Systemen machen. Ein System weist nicht deterministisches Verhalten auf, wenn dessen Modell nicht ausreichende Informationen enthält oder das Systemverhalten auf einem bestimmten Ereignis in einem bestimmten Zustand nicht eindeutig vorhersagbar ist [13].

Im weiteren Verlauf dieser Arbeit werden nur logische deterministische ereignisdiskrete Systeme betrachtet und deren Verhalten mittels deterministischen endlichen Automaten und regulären Sprachen beschrieben.

2.1.1. Sprachentheoretische Grundlagen

In diesem Unterkapitel werden sprachentheoretische Grundlagen erläutert. Alle sprachentheoretischen Begriffe und Definitionen sind der Literatur [30] und [13] entnommen. Ein Alphabet Σ ist eine endliche, nichtleere Menge von paarweisen verschiedenen Symbolen. Ein Symbol kann zum Beispiel ein Ereignis σ darstellen. Durch die Verkettung von Ereignissen $(\sigma_1\sigma_2\sigma_3...\sigma_k \text{ mit } \sigma_i \in \Sigma \text{ und für } k \geq 1)$ entsteht eine Zeichenkette, ein sogenannter String s. Die Verkettung von Ereignissen wird in der Literatur auch als Konkatenation oder als cat bezeichnet.

Die Menge Σ^+ enthält alle möglichen Strings des Alphabets Σ mit Ausnahme des leeren Strings ϵ. Ein leerer String $\epsilon \notin \Sigma$ enthält nur das leere Symbol und kann an beliebiger Stelle in einem String eingefügt werden. Dadurch ändert sich die Bedeutung des Strings nicht. Die leere Menge $\emptyset$ ist eine Menge, die kein Symbol enthält. Die Kleene-Hülle von Σ ist definiert als $\Sigma^* = \{\epsilon\} \cup \Sigma^+$. Σ ist also immer eine Teilmenge von Σ^*. Die Menge Σ^+ sowie die Menge Σ^* enthalten jeweils unendlich viele Elemente, weil die Länge der Strings nicht beschränkt ist. Sei beispielsweise $\Sigma = \{a\}$, dann ist $\Sigma^* = \{\epsilon, a, aa, aaa, aaaa, ...\}$. Im Folgenden werden weitere Begriffe und Definitionen zu formalen Sprachen dargestellt:

- Die Konkatenation von Strings $cat : \Sigma^* \rightarrow \Sigma^*$ ist die Abbildung $cat(\epsilon, s) = cat(s, \epsilon) = s$, $s \in \Sigma^*$ und $cat(s, t) = st$, für $s, t \in \Sigma^+$.

- Die Länge $|s|$ eines Strings s bezeichnet die Anzahl der Symbole in s. Ein leerer String hat immer die Länge $|\epsilon| = 0$.
- Es sei $s = cat(t, \sigma) = t\sigma$ mit $t \in \Sigma^*$, $\sigma \in \Sigma$, dann ist $\overline{s} = t$ Präfix von s.
- Es sei $s = u\sigma v$ mit $u \in \Sigma^*$, $\sigma \in \Sigma$, $v \in \Sigma^*$, dann ist v Suffix von s ab σ.
- $s = utv$ mit $u, t, v \in \Sigma^*$ ist t Substring von s.

Eine formale Sprache ist eine Menge von Strings und wird nach [30] wie folgt definiert:

Definition 2.2 (Formale Sprache) *Eine formale Sprache L über dem Alphabet Σ ist jede Teilmenge $L \subseteq \Sigma^*$.*

Nach Definition 2.2 sind Σ^*, die leere Menge $\emptyset$ sowie die Menge $\{\epsilon\}$ Sprachen. $L_1 = \{\epsilon, \sigma_1, \sigma_3, \sigma_3\sigma_2, \sigma_3\sigma_2\sigma_1\}$ und $L_2 = \{\sigma_1, \sigma_2, \sigma_3, \sigma_1\sigma_2, \sigma_1\sigma_2\sigma_3\}$ sind ein Beispiel für endliche und über $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$ definierte Sprachen.

Nach dem Theorem von Kleene [13, S.159] ist eine formale Sprache genau dann regulär, wenn sie von einem endlichen deterministischen Automaten erkannt wird. Das bedeutet, dass es für jede reguläre Sprache einen endlichen deterministischen Automaten gibt, der diese Sprache erkennt. Dieses Theorem deutet bereits darauf hin, dass eine reguläre Sprache in Form eines deterministischen endlichen Automaten grafisch dargestellt werden kann. Sprachen, für die man keinen endlichen Automaten finden kann, sind also nicht regulär.

Reguläre Sprachen sind bezüglich den aus der Mengenlehre bekannten Operationen, wie Vereinigung $(\cup)$, Schnitt $(\cap)$, Differenz $(\setminus)$, Komplement $(\bar{A})$ und Symmetrische Differenz $(\triangle)$, abgeschlossen [13, S.165]. Das heißt, die Anwendung dieser Operationen auf reguläre Sprachen ergibt wieder eine reguläre Sprache. Weitere reguläre Operationen aus der Sprachentheorie sind mit ihren Definitionen im Folgenden aufgelistet [30]:

- Konkatenation von Sprachen: Die Konkatenation zweier Sprachen L_1 und L_2 führt auf die Sprache $L_1L_2 = \{s \in \Sigma^* | (s = s_1s_2) \wedge (s_1 \in L_1) \wedge (s_2 \in L_2)\}$, $L_1, L_2 \in \Sigma^*$.
- Präfix-Hülle: Präfix-Hülle $\overline{L}$ einer Sprache L enthält alle Strings aus L und alle ihre Präfixe, $\overline{L} = \{s \in \Sigma^* | \forall t \in \Sigma^*(st \in L)\}$, $L \subseteq \Sigma^*$. Eine Sprache ist präfix-geschlossen, wenn: $L = \overline{L}$ (wenn $\overline{L}$ nur die Präfixe der Ereignisfolgen aus L enthält).
- Kleene-Hülle: Mit Kleene Hülle L^* bezeichnet man eine endliche Anzahl Konkatenation der Sprache L mit leerer String ϵ, $L^* = \{\epsilon\} \cup L \cup LL \cup LLL \cup ...$, $L \subseteq \Sigma^*$.

2.1.2. Automaten

Grundlagen

Das Verhalten von ereignisdiskreten Systemen kann durch die Klasse der deterministischen endlichen Automaten (engl. Deterministic Finite Automaton, DFA) beschrieben werden, bei denen Zustände, die Systemzustände repräsentieren, und Zustandsübergänge durch Ereignisse ausgelöst werden. Ein DFA besitzt endlich viele Zustände und unterscheidet sich von einem nicht deterministischen Automaten (engl. Nondeterministic Finite Automaton, NFA) dadurch, dass seine Zustandsübergänge eindeutig festgelegt sind. Es wird jedem Zustands-Ereignispaar (x, σ) genau ein Nachfolgezustand $x' = \delta(x, \sigma)$ zugeordnet, während bei einem NFA möglicherweise mehr als ein Nachfolgezustand zugeordnet wird. Zudem ist bei einem DFA der Initialzustand immer eindeutig festgelegt und sein Verhalten kann immer eindeutig vorhergesagt werden [13]. Dies ist bei einem NFA nicht immer der Fall.

Ein deterministischer endlicher Automat wird in der Literatur [30, 13] auch als Standardautomat bezeichnet. In der Supervisory Control Theory ist ein ereignisdiskretes System als Sprachgenerator definiert. Deshalb werden Standardautomaten in der SCT auch Generatoren genannt. Sämtliche Ereignisse werden von dem Generator generiert. Ein Generator ist nach [30] folgendermaßen definiert:

Definition 2.3 (Generator) *Ein Generator ist ein Fünftupel*

$$G = (X, \Sigma, \delta, x_0, X_m) \text{ mit} \tag{2.1}$$

- X *endlicher Zustandsmenge,*
- Σ *Menge der möglichen Ereignisse (Ereignismenge bzw. Ereignisraum),*
- $\delta : X \times \Sigma \to X$ *Zustandsübergangsfunktion,*
- $x_0 \in X$ *Initialzustand,*
- $X_m \subseteq X$ *Menge von markierten Zuständen (Endzustände).*

Die Mengen X, X_m, Σ und δ in Definition 2.3 sind immer endlich. Ein Generator ist aktiv und kann ausgehend von seinem Initialzustand spontan Ereignisse generieren, die zu Zustandsübergängen führen. Ein Standardautomat ist dagegen passiv und kann nur die Zustandsübergänge erkennen. Sie werden vorwiegend in der Spracherkennung eingesetzt, um zu überprüfen, ob eine gegebene Ereignisfolge zur Sprache des Automaten gehört und deshalb akzeptiert oder nicht akzeptiert wird [30].

Mit Hilfe der Zustandsübergangsfunktion δ werden die Zustandsübergänge beschrieben. Nach der Definition von Standardautomaten ist seine Zustandsübergangsfunktion eine totale Funktion. Das bedeutet, alle Ereignisse können in allen Zuständen auftreten. Eine totale Funktion wird nach [30] folgendermaßen definiert:

$$\forall(x,\sigma) \in (X \times \Sigma) \text{ ist } \delta(x,\sigma) \text{ definiert }, \delta(x,\sigma)! \tag{2.2}$$

Um eine totale Funktion zu erreichen beziehungsweise um einen unvollständigen Standardautomaten vollständig zu machen, wird oft ein zusätzlicher Zustand, ein sogenannter Dump-State, eingeführt. Der Automat fällt in einen solchen Zustand hinein, wenn eine Zustandsübergangsfunktion nicht definiert ist.

Bei einem Generator ist die Zustandsübergangsfunktion partiell definiert. Das bedeutet, dass für bestimmte Zustands-Ereignispaare (x,σ) kein Nachfolgezustand x' zugeordnet ist.

Die Zustandsübergangsfunktion

$$\delta : X \times \Sigma \to X \tag{2.3}$$

eines Generators, ordnet jedem Zustand $x \in X$ einen eindeutigen Nachfolgezustand $x' = \delta(x,\sigma)$ zu. In der Literatur verwendet man auch die Schreibweise $\delta(x,\sigma)!$ um darzustellen, dass die Zustandsübergangsfunktion für das Ereignis σ im aktuellen Zustand x definiert ist. $\neg(x,\sigma)!$ bedeutet, dass diese Zustandsübergangsfunktion nicht definiert ist. Die Zustandsübergangsfunktion δ kann auch auf Strings erweitert werden. Für $s \in \Sigma^*$ und $\sigma \in \Sigma$ gilt: $\delta(x,\epsilon) = x$, $\delta(x,s\sigma) = \delta(\delta(x,s),\sigma)$, solange $x' = \delta(x,s)$ und $\delta(x',\sigma)$ definiert ist.

In der Literatur unterscheidet man zwischen der Sprache $L(G)$ und der markierten Sprache $L_m(G)$ des Generators G. Die Sprache $L(G)$ ist regulär und enthält alle durch den Generator G von seinem Initialzustand x_0 ausgehend erzeugbaren Ereignisfolgen. Ob der Generator dabei einen markierten Zustand annimmt oder nicht, wird nicht beachtet. Es gilt

$$L(G) = \{s \in \Sigma^* | \delta(x_0,s)!\}. \tag{2.4}$$

Die markierende Sprache $L_m(G) \subseteq L(G)$ ist ebenfalls regulär und enthält die Gesamtheit aller Strings, die vom Initialzustand x_0 ausgehend auf einen markierten Zustand führen. Es gilt

$$L_m(G) = \{s \in L(G) | \delta(x_0,s) \in X_m\}. \tag{2.5}$$

Ein Zustand aus X_m wird immer dann erreicht, wenn beispielsweise das System eine Aufgabe vollendet hat. Abbildung 2.1 zeigt den Graph eines Beispielgenerators G mit $X = \{q0, q1, q2, q3\}$, $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$, $\delta(q0, \sigma_1) = q1$, $\delta(q0, \sigma_3) = q2$, $\delta(q1, \sigma_4) = q1$, $\delta(q1, \sigma_2) = q2$ und $\delta(q2, \sigma_1) = q0$. Die Zustände werden als Knoten (Kreise) und Zustandsübergänge, auch Transitionen genannt, als Kanten (Pfeile) dargestellt. Der Generator ist initialisiert mit dem Initialzustand $q0$. Die Menge der markierten Zustände X_m enthält ebenfalls nur $q0$. Die Zustandsübergangsfunktion ist partiell. Das heißt, sie ist nicht für jede Situation $\delta(q, \sigma)$ definiert. Tritt in dem Zustand $q0$ das Ereignis σ_1 auf, so wechselt der Generator in dem Zustand $q1$. Der Zustand $q1$ hat eine Schlinge, weil $\delta(q1, \sigma_4) = q1$ gilt. Eine Schlinge an einem Zustand wird auch als „Selfloop“ bezeichnet. Der Initialzustand x_0 wurde in dem Beispielgenerator durch einen Pfeil und der markierte Zustand x_m durch einen Doppelkreis gekennzeichnet.

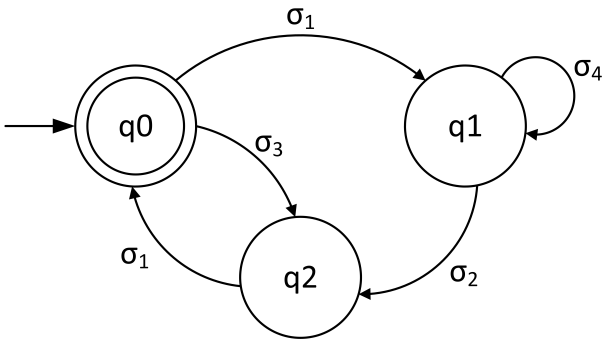


Abbildung 2.1.: Beispielgenerator.

Die Zustandsübergänge eines Generators können auch mit Hilfe einer **Adjazenzmatrix** algebraisch beschrieben werden. Dafür wird eine symmetrische (n, n)-Matrix eingeführt, deren ij-tes Element genau dann gleich eins ist, wenn die Zustandsübergangsfunktion δ dem Zustand j den Nachfolgezustand i zuordnet [30]:

$$A = (a_{ij}) \text{ mit } a_{ij} = \begin{cases} 1 & \text{wenn Kante j} \rightarrow \text{i existiert} \\ 0 & \text{sonst} \end{cases} \tag{2.6}$$

Für den Beispielgenerator aus Abbildung 2.1 lautet die Adjazenzmatrix

$$A = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix}.$$

Die reguläre Sprache, die der Beispielgenerator generiert,

$$L(G) = \{\epsilon, \sigma_1, \sigma_3, \sigma_1\sigma_4, \sigma_3\sigma_1, \sigma_1\sigma_4\sigma_2, \sigma_1\sigma_2\sigma_1, \sigma_1\sigma_4\sigma_4\sigma_2\sigma_1, ...\}$$

umfasst eine unendliche Anzahl von möglichen Ereignisfolgen, weil es im Generatorgraphen Zyklen gibt, die beliebig oft durchlaufen werden können. Die markierende Sprache des Generators enthält ebenso eine unendliche Anzahl von Ereignisfolgen und lautet

$$L_m(G) = \{\sigma_3\sigma_1, \sigma_1\sigma_2\sigma_1, \sigma_1\sigma_4\sigma_2\sigma_1, \sigma_1\sigma_4\sigma_4\sigma_2\sigma_1, ...\}.$$

Durch die **natürliche Projektion** und die **inverse natürliche Projektion** kann die Sprache eines Generators manipuliert werden. Unter der natürlichen Projektion P_{Σ_1} versteht man eine Funktion, die aus einem gegebenen String s des Alphabets Σ, diejenigen Ereignisse entfernt, die es im Alphabet Σ_1 nicht gibt. Die inverse natürliche Projektion $P_{\Sigma_1}^{-1}$ fügt einem String Ereignisse aus Σ hinzu, die nicht in Σ_1 enthalten sind und durch die natürliche Projektion wieder entfernt werden. Das bedeutet, die inverse natürliche Projektion fügt für jedes $\sigma \in \Sigma - \Sigma_1$ an jedem Zustand ein „Selfloop“ an. Auf Details der natürlichen und inversen natürlichen Projektionen wird hier nicht weiter eingegangen, es sei auf [30] und [13] verwiesen.

Komposition von Automaten

In dieser Arbeit werden die Generatoren für die ereignisdiskrete Systembeschreibungen G genannt. Ein ereignisdiskretes System besteht häufig aus mehreren Teilsystemen, die man bei der kompositionalen Modellbildung zunächst einzeln betrachtet und durch Generatoren darstellt. Die dabei erhaltenen Generatoren der Teilsysteme werden anschließend zum Modell des Gesamtsystems verkoppelt beziehungsweise zusammengefasst. Das Gesamtmodell besteht dann aus den Modellen der Teilsysteme und den Kopplungen. Solche Systeme werden in der Literatur auch als gekoppelte Systeme bezeichnet. In diesem Unterkapitel werden zwei Kompositionsoperatoren vorgestellt, die zeigen, wie Generatoren bei der kompositionalen Modellbildung verkoppelt werden können. Der durch solche Komposition entstehende Generator wird auch Kompositionsgenerator genannt.

Produktkomposition

Der erste Kompositionsoperator ist die Produktkomposition. In der Literatur wird die Produktkomposition auch als Strenge Synchronisation (engl. Strict Product Composition, SPC) bezeichnet und wird nach [30] folgendermaßen definiert:

Definition 2.4 (Strenge Synchronisation, SPC) *Das Produkt zweier Generatoren* $G_1 = (X_1, \Sigma_1, \delta_1, x_{01}, X_{m1})$ *und* $G_2 = (X_2, \Sigma_2, \delta_2, x_{02}, X_{m2})$ *ist*

$$G_\times = G_1 \parallel_{SPC} G_2 = (X_1 \times X_2, \Sigma_{ce}, \delta, (x_{01}, x_{02}), X_{m1} \times X_{m2}) \; mit \tag{2.7}$$

$$\delta_X((x_1, x_2), \sigma) = \begin{cases} \delta_1(x_1, \sigma) \times \delta_2(x_2, \sigma) & \textit{falls } \delta_1(x_1, \sigma)! \wedge \delta_2(x_2, \sigma)! \\ \textit{nicht definiert} & \textit{sonst.} \end{cases} \tag{2.8}$$

G_1 und G_2 werden auf die gemeinsamen Ereignisse

$$\Sigma_{ce} = \Sigma_1 \cap \Sigma_2 \tag{2.9}$$

synchronisiert und dürfen nur im totalen Gleichschritt schalten. Das heißt, die Zustandsübergangsfunktion δ ist definiert, wenn ein Ereignis σ in G_1 und G_2 gleichzeitig einen Zustandsübergang auslöst. Die privaten Ereignisse

$$\Sigma_{pe} = (\Sigma_2 - \Sigma_1) \cup (\Sigma_1 - \Sigma_2) \tag{2.10}$$

werden ignoriert und kommen im Generator $G_\times$ (Komponiertes System) nicht vor. Die reguläre Sprache des entstandenen Generators $G_\times$ ist der Durchschnitt von $L(G_1)$ und $L(G_2)$:

$$L(G_\times) = L(G_1 \parallel_{SPC} G2) = L(G_1) \cap L(G_2) \tag{2.11}$$

Haben die Generatoren G_1 und G_2 keine gemeinsamen Ereignisse, so enthält die Sprache des Generators $G_\times$ als einziges Element den leeren String ϵ, wenn die Initialzustände der beiden Generatoren gleichzeitig ihrer markierten Zustände sind. Ansonsten ist die Sprache des Generators $G_\times$ leer. Die Produktkomposition wird in der Supervisory Control Theory, zum Beispiel bei der Zusammenschaltung von Strecke und Steuerung, verwendet.

Parallele Komposition

Der zweite Kompositionsoperator ist die parallele Komposition. Die Verkopplung zweier Generatoren ist bezüglich der gemeinsamen Ereignisse mit der Produktkomposition identisch. Jedoch werden private Ereignisse nicht mehr ignoriert, sondern sind immer erlaubt, wenn die Zustandsübergangsfunktion δ im jeweiligen Generator definiert ist. Das heißt, die Teilsysteme können sich im komponierten System asynchron entsprechend ihrer privaten Ereignisse bewegen. In der Literatur wird die parallele Komposition auch als synchrones Produkt (SYPC) bezeichnet und wird nach [30] folgendermaßen definiert:

Definition 2.5 (Synchrones Produkt, SYPC) *Die parallele Komposition zweier Generatoren* $G_1 = (X_1, \Sigma_1, \delta_1, x_{01}, X_{m1})$ *und* $G_2 = (X_2, \Sigma_2, \delta_2, x_{02}, X_{m2})$ *ist*

$$G_{\|} = G_1 \|_{SYPC} G_2 = (X_1 \times X_2, \Sigma, \delta, (x_{01}, x_{02}), X_{m1} \times X_{m2}) \textit{ mit} \tag{2.12}$$

$$\delta_{\|}((x_1, x_2), \sigma) = \begin{cases} \delta_1(x_1, \sigma) \times \delta_2(x_2, \sigma) & \textit{falls } \delta_1(x_1, \sigma)! \wedge \delta_2(x_2, \sigma)! \\ \delta_1(x_1, \sigma) \times x_2 & \textit{falls } \delta_1(x_1, \sigma)! \wedge \neg\delta_2(x_2, \sigma)! \wedge \sigma \notin (\Sigma_1 \cap \Sigma_2) \\ x_1 \times \delta_2(x_2, \sigma) & \textit{falls } \neg\delta_1(x_1, \sigma)! \wedge \delta_2(x_2, \sigma)! \wedge \sigma \notin (\Sigma_1 \cap \Sigma_2) \\ \textit{nicht definiert} & \textit{sonst.} \end{cases} \tag{2.13}$$

Die reguläre Sprache des unter paralleler Komposition aus G_1 und G_2 entstandenen Generators $G_{\|}$ ist

$$L(G_{\|}) = L(G_1 \|_{SYPC} G_2) = P_{\Sigma_1}^{-1}(L(G_1)) \cap P_{\Sigma_2}^{-1}(L(G_2)). \tag{2.14}$$

Sind die Ereignisalphabete der beiden Generatoren gleich ($\Sigma_1 = \Sigma_2$), so ist die parallele Komposition identisch mit der Produktkomposition. Dieser Spezialfall wird auch „Meet“ genannt. Haben die Generatoren dagegen disjunktive Ereignisalphabete $\Sigma_1 \cap \Sigma_2 = \emptyset$, so bewegen sie sich unter der parallelen Komposition vollkommen unabhängig voneinander (asynchron). Dieser Spezialfall wird als „Shuffle“ bezeichnet. Die parallele Komposition wird für die Verkopplung von Generatoren, die parallel arbeiten und zusammen ein komplexes System modellieren, verwendet.

Erreichbarkeits- und Ko-Erreichbarkeitsanalyse von Generatoren

Eine Erreichbarkeitsanalyse beantwortet die Frage, welche Systemzustände $x \in X$ vom Initialzustand $x_0 \in X$ aus erreicht werden können. Im Generatorgraphen heißt dies, ob es einen Pfad von x_0 nach x gibt. Nicht erreichbare Zustände können auch die Folge eines Fehlers in der Modellierung sein. Ein Zustand x ist genau dann von einem Initialzustand x_0 aus erreichbar, wenn die Zustandsübergangsfunktion $\delta(x_0, s)!$ definiert ist oder $\delta(x_0, s) = x$ existiert. Nicht erreichbare Zustände und Zustandsübergänge, die zu nicht erreichbaren Zuständen gehen oder von nicht erreichbaren Zuständen abgehen, können durch Ac-Operation aus dem Generator entfernt werden. Diese Operation reduziert also einen Generator auf seinen vom Initialzustand ausgehend erreichbaren Teil. Dadurch ändern sich die Sprachen $L(G)$ und $L_m(G)$ nicht, weil es keine Ereignisfolgen gibt, die vom Initialzustand ausgehend zu den nicht erreichbaren Zuständen führen. Die Ac-Operation ist nach [2] formal wie folgt definiert:

Definition 2.6 (Ac-Operation) *Die Ac-Operation wird für einen Generator G formal definiert als*

$$Ac(G) = (X_{ac}, \Sigma, \delta_{ac}, x_0, X_{ac,m}) \text{ mit} \tag{2.15}$$

$$X_{ac} = \{x \in X | (\exists s \in \Sigma^*)(\delta(x_0, s) = x)\}, \tag{2.16}$$

$$X_{ac,m} = X_m \cap X_{ac}, \tag{2.17}$$

$$\delta_{ac} = \delta|_{X_{ac} x \Sigma \to X_{ac}}. \tag{2.18}$$

Die Notation $\delta|_{X_{ac} x \Sigma \to X_{ac}}$ symbolisiert die Restriktion der Zustandsübergangsfunktion δ auf die erreichbaren Zustände. Ein Generator G ist genau dann erreichbar, wenn die Zustandsmenge X des Generators identisch der Zustandsmenge des reduzierten Generators ist, also $X = X_{ac}$.

Eine weitere Erreichbarkeitsanalyse ist die Ko-Erreichbarkeit. Hierbei wird analysiert, ob ein markierter Zustand x_m von jedem beliebigen Zustand $x \in X$ aus erreicht werden kann. Formal wird ein Zustand als ko-erreichbar bezeichnet, wenn ein String $s \in \Sigma^*$ existiert, sodass $\delta(x, s) \in X_m$ gilt. Die CoAc-Operation entfernt alle nicht ko-erreichbaren Zustände aus einem Generator und reduziert ihn auf seinen ko-erreichbaren Teil. Sie ist nach [2] formal wie folgt definiert:

Definition 2.7 (CoAc-Operation) *Die CoAc-Operation wird für einen Generator G formal definiert als*

$$CoAc(G) = (X_{coac}, \Sigma, \delta_{coac}, x_{0,coac}, X_m) \text{ mit} \tag{2.19}$$

$$X_{coac} = \{x \in X | (\exists s \in \Sigma^*)(\delta(x, s) \in X_m)\}, \tag{2.20}$$

$$x_{0,coac} = \begin{cases} x_0 & \text{falls } x_0 \in X_{coac}, \\ \text{nicht definiert} & \text{sonst,} \end{cases} \tag{2.21}$$

$$\delta_{coac} = \delta|_{X_{coac} x \Sigma \to X_{coac}}. \tag{2.22}$$

Durch das Anwenden der CoAC-Operation auf einem Generator ändert sich seine reguläre Sprache $L(G)$, weil diese Operation auch erreichbare Zustände von G entfernen kann. Die Zustandsübergangsfunktion reduziert sich dann auf die verbleibende Zustandsmenge X_{coac}. Ein Generator ist ko-erreichbar, genau dann wenn $X = X_{coac}$ beziehungsweise $G = CoAC(G)$. Die reguläre Sprache $L(G)$ eines ko-erreichbaren Generators enthält nur

die Präfixe der Ereignisfolgen aus $\overline{L_m(G)}$. Ist $CoAC(G) \neq G$ dann ist G nicht blockierungsfrei, weil $\overline{L_m(G)} \neq L(G)$ ist. Das bedeutet also, ein ko-erreichbarer Generator kann kein String s generieren, der nicht zu einem markierten Zustand führt.

Die Trim-Operation fasst Ac- und CoAc-Operation zusammen und wird nach [2] formal folgendermaßen definiert:

Definition 2.8 (Trim-Operation) *Die Trim-Operation wird für einen Generator G definiert als*

$$Trim(G) = CoAc[Ac(G)] = Ac[CoAc(G)]. \tag{2.23}$$

Die Trim-Operation entfernt aus einem Generator G alle nicht erreichbaren und nicht ko-erreichbaren Zustände. Ist ein Generator erreichbar und ko-erreichbar, so heißt er trim und es gilt: $G = Trim(G)$ beziehungsweise $X = X_{ac} = X_{coac}$. Ist ein Generator trim, so ist er auch blockierungsfrei.

Deadlock, Livelock, Blockierung bei Generatoren

Es gibt zwei Arten von Blockierungen. Man spricht von einem Deadlock (totale Verklemmung), wenn ein Generator G einen nicht markierten Zustand $x \notin X_m$ erreicht mit $\neg\delta(x, \sigma)!, \forall\sigma \in \Sigma$ und aus diesem Zustand kein weiteres Ereignis mehr generieren kann. Der Generator kann sich dann nicht mehr bewegen. In diesem Fall gilt die strikte Inklusion

$$\overline{L_m(G)} \subset L(G). \tag{2.24}$$

Von einem Livelock (partielle Verklemmung) spricht man, wenn ein Generator eine Zustandsmenge $\tilde{X} \subseteq X$ mit $x \notin X_m, \forall x \in \tilde{X}$ erreicht und diese Zustandsmenge über keine Transition mehr verlassen kann. Der Generator kann dann weiterhin Ereignisse generieren aber keinen Zustand mehr aus der Zustandsmenge X_m annehmen. Hier gilt auch die strikte Inklusion 2.24. Entsprechend ist ein Generator nichtblockierend, genau dann wenn

$$\overline{L_m(G)} = L(G). \tag{2.25}$$

Deadlock- und Livelock-Zustände können sich beispielsweise bei der parallelen Komposition von Teilmodellen ergeben. Als Beispiel für Blockierungen bei Generatoren wird der in Abbildung 2.2 dargestellte Generator G mit der Zustandsmenge $X = \{0, 1, 2, 3, 4, 5, 6\}$ betrachtet. Die Menge der Zustände kann in vier Teilmengen X_1, X_2, X_3 und X_4 mit $X = X_1 \cup X_2 \cup X_3 \cup X_4 = \{0, 1, 2\} \cup \{3\} \cup \{4\} \cup \{5, 6\} = \{0, 1, 2, 3, 4, 5, 6\}$ zerlegt

werden, da zwischen diesen Mengen nur Kanten in einer Richtung existieren. Der Generator kann den markierten Zustand 0 nie mehr erreichen, wenn er einmal den Zustand 4 erreicht hat. In diesem Fall ist der Generator blockiert und kann diesen Zustand nicht wieder verlassen. Des Weiteren befindet sich der Generator in einem Livelock sobald er in der Zustandsmenge X_4 gefangen ist, von der er nicht zu seinem markierten Zustand 0 kommen kann. Der Zustand 3 ist nicht erreichbar. Die Anwendung der Ac-Operation auf diesem Generator würde den Zustand 3 entfernen. Die CoAc-Operation würde die Zustandsmengen X_3 und X_4 entfernen. Die Trim-Operation würde den Generator auf die Zustandsmenge X_1 reduzieren.

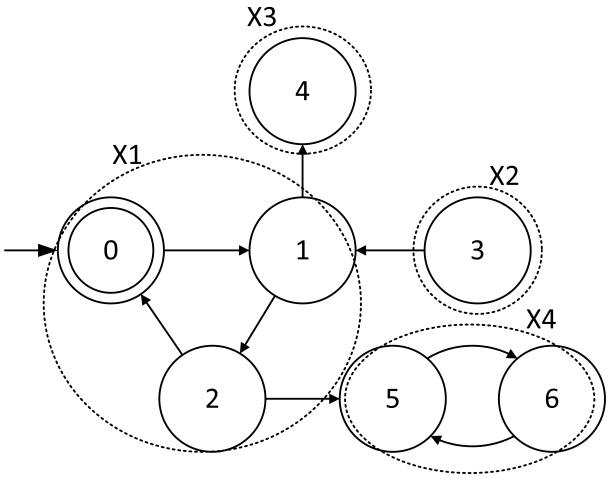


Abbildung 2.2.: Beispiel für einen Deadlock, Livelock und Blockierung bei Generatoren.

2.2. Supervisory Control Theory

Die Supervisory Control Theory (SCT) ist ein mathematisches Framework, das erstmals in den 1980er Jahren von P. J. Ramadge und W. M. Wonham [22, 30] eingeführt und im Laufe der Zeit in verschiedene Richtungen weiterentwickelt wurde. Die SCT stellt Methoden zur automatischen Synthese und Analyse von modellbasierten Steuerungen für zeitbewertete und logische (nichtzeitbewertete) ereignisdiskrete Systeme bereit. Die grundlegenden Arbeiten der SCT sind [22] und [31]. In ihrer ursprünglichen Formulierung basieren die Methoden auf Generatoren und formalen Sprachen. In [14] wurde bereits ein Ansatz vorgestellt, der ein Steuerungsentwurf nach SCT auch mittels Petri-Netzen ermöglicht.

Der Grundgedanke der SCT lehnt sich an die klassische Regelungstechnik an. Es wird aus dem Modell der ungesteuerten Strecke und der formalen Spezifikationen für das gewünschte

Systemverhalten eine Steuerung bestimmt, genannt Supervisor. Dieser schränkt das ungesteuerte Verhalten der Strecke derart ein, dass die Erfüllung der Spezifikationen gewährleistet werden muss. Die ungesteuerte Strecke und der Supervisor bilden also zusammen einen geschlossenen Steuerkreis. Der Supervisor verfolgt die in der Strecke auftretenden Ereignisse und beeinflusst das System über eine Rückführschleife durch Erlauben und Verbieten von Steuereingriffen, wie in Abbildung 2.3 dargestellt. Die korrekte Funktion des geschlossenen Steuerkreises kann mit Methoden der SCT und der Streckenmodelle formal nachgewiesen werden.

2.2.1. Grundlagen der Supervisory Control Theory

Die folgenden Beschreibungen für die SCT beziehen sich auf reguläre Sprachen und Generatoren. Die Strecke G wird ereignisdiskret mit Hilfe des Fünftupels $G = (X, \Sigma, \delta, x_0, X_m)$ nach Definition 2.3 als Generator beschrieben. $L(G)$ und $L_m(G)$ sind Sprachen, die von G generiert und markiert werden. Das Ereignisalphabet der Strecke Σ kann in zwei disjunkte Alphabete

$$\Sigma = \Sigma_{uc} \cup \Sigma_c \text{ mit } \Sigma_{uc} \cap \Sigma_c = \emptyset \tag{2.26}$$

partitioniert werden, wobei das Teilalphabet Σ_{uc} die Menge an nicht steuerbaren Ereignissen darstellt und das Teilalphabet Σ_c die Menge an steuerbaren Ereignissen (die Indizes kommen von den englischen Wörtern controllable beziehungsweise uncontrollable). Der Supervisor S kann die steuerbaren Ereignisse aus der Menge Σ_c erkennen und eine Ausführung verhindern, zum Beispiel das Ein- und Ausschalten von Aktoren. Er hat aber keinen unmittelbaren Einfluss auf die nicht steuerbaren Ereignisse aus der Menge Σ_{uc}. Ein Beispiel für das nicht steuerbare Ereignis ist das Signal einer Lichtschranke. Allerdings kann der Supervisor durch Verbieten und Erlauben von steuerbaren Ereignissen die unerwünschten Zustandsübergänge verhindern und somit das Streckenverhalten soweit einschränken, dass unerwünschte nicht steuerbare Ereignisse nicht auftreten. Des Weiteren soll der zu berechnende Supervisor minimal restriktiv sein. Das heißt, er soll einerseits die Spezifikationen für das gewünschte Verhalten der Strecke einhalten und anderseits das Verhalten der ungesteuerten Strecke so wenig wie möglich einschränken. Steuerbare Ereignisse werden bei der grafischen Darstellung des Generators mit einem Querstrich an der Transition gekennzeichnet.

Neben der steuerbaren und nicht steuerbaren Ereignissen wird zusätzlich zwischen beobachtbaren Σ_o und nicht beobachtbaren Ereignissen Σ_{uo} unterschieden (die Indizes werden

von den englischen Wörtern observable beziehungsweise unobservable abgeleitet), sodass gilt:

$$\Sigma = \Sigma_{uo} \cup \Sigma_o \text{ mit } \Sigma_{uo} \cap \Sigma_o = \emptyset \tag{2.27}$$

Die Beobachtbarkeit wird an dieser Stelle nicht weiter behandelt, da in der vorliegenden Arbeit alle Ereignisse für jeden zu berechnenden Supervisor global zur Verfügung stehen.

Systeme bestehen meistens aus mehreren Streckenkomponenten. Jede Komponente besitzt Zustände, Transitionen und Ereignisse. Fügt man alle Streckenkomponenten durch Komposition zusammen, entsteht ein ungesteuertes Gesamtmodell. Wird für ein solches Gesamtmodell der Strecke ein einziger Supervisor entworfen, so spricht man von einem monolithischen Ansatz. In Abbildung 2.3 stellt S den Supervisor und G die ungesteuerte Strecke dar. Die Steuerung soll gemeinsam mit der Strecke im geschlossenen Steuerkreis S/G die Spezifikation K erfüllen.

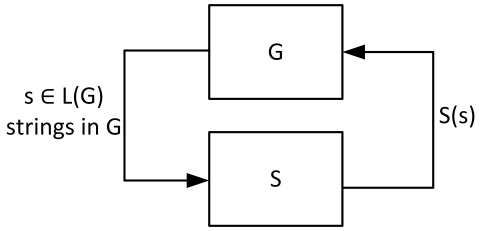


Abbildung 2.3.: Der geschlossene Steuerkreis S/G beim monolithischen Ansatz.

Die Strings, die von der Strecke generiert werden, werden an den Supervisor zurückgeführt. Der Supervisor befindet sich im Rückführzweig und berechnet für jeden von G generierten String $s \in \Sigma$ eine Menge $S(s)$, die die im nächsten Schritt erlaubten steuerbaren Ereignisse enthält. Die unerwünschten steuerbaren Ereignisse werden dabei deaktiviert und dürfen in der Menge $S(s)$ nicht auftauchen. Jede Menge $S(s)$ enthält zusätzlich noch alle aktuell möglichen nicht steuerbaren Ereignisse, da diese durch einen Supervisor nicht verhindert werden können. Das geschlossene Verhalten $L(S/G)$ von S/G wird nach [30] wie folgt definiert:

Definition 2.9 (Geschlossenes Verhalten) *Das geschlossene Verhalten $L(S/G) \subseteq L(G)$ von S/G ist rekursiv definiert mit $s \in \Sigma^*$ und $\sigma \in \Sigma$:*

1. $\epsilon \in L(S/G)$,
2. $(s \in L(S/G) \wedge \sigma \in S(s) \wedge s\sigma \in L(G)) \rightarrow s\sigma \in L(S/G)$,
3. *keine anderen Strings gehören zu* $L(S/G)$.

Die Definition besagt, dass der leere String ϵ immer in $L(S/G)$ enthalten ist und Strings s des gesteuerten Systems S/G nur dann durch Folgeereignisse σ erweitert werden dürfen, wenn diese Ereignisse in S erlaubt und in G möglich sind. Das markierende Verhalten des geschlossenen Steuerkreises $L_m(S/G)$ wird auf die erlaubten Strings reduziert, sodass gilt:

$$L_m(S/G) = L(S/G) \cap L_m(G) \tag{2.28}$$

Der Steuerkreis S/G wird als nichtblockierend bezeichnet, wenn die Sprache $L(S/G)$ nur die Präfixe der Ereignisfolgen aus $\overline{L_m(S/G)}$ enthält. Es gilt:

$$\overline{L_m(S/G)} = L(S/G) \tag{2.29}$$

Ist die Eigenschaft in Gleichung 2.29 erfüllt, so wird sich jede Ereignisfolge, die der Generator der Sprache $L(S/G)$ zu jedem beliebigen Zeitpunkt gerade generiert, immer auf einem Pfad in Richtung eines markierten Zustandes befinden. Sind dagegen Ereignisfolgen vorhanden, die in $L(S/G)$ enthalten sind aber nicht in $L_m(S/G)$, so sind in dem Generator der Sprache $L(S/G)$ Pfade vorhanden, die zu einem blockierenden Zustand führen.

In dieser Arbeit werden auch Spezifikationen $K \subseteq L(G)$ formal mit regulären Sprachen und Generatoren beschrieben. Eine Spezifikation enthält Festlegungen, welche Strings s in der gesteuerten Strecke auftreten dürfen oder welche Zustandsübergänge in welcher Reihenfolge auftreten sollen. Sie muss die Bedingung der Steuerbarkeit bezüglich G erfüllen. Diese Bedingung wird erfüllt, genau dann wenn

$$\overline{K}\Sigma_{uc} \cap L(G) \subseteq \overline{K}. \tag{2.30}$$

Diese Bedingung besagt, dass kein String $s \in \overline{K}$ durch das Auftreten eines nicht steuerbaren Ereignisses $\sigma \in \overline{K}\Sigma_{uc}$ zu keiner Zeit die Spezifikation K verletzen darf. Ansonsten ist K nicht steuerbar bezüglich G und man kann keine Steuerung finden, die K erfüllt. In diesem Fall wird eine Spezifikation gesucht, die möglichst gering von K abweicht, beziehungsweise etwas mehr einschränkend als K und steuerbar bezüglich G ist. Das ist die supremale steuerbare Teilsprache $K^{\uparrow C}$. Im besten Fall ist $K = K^{\uparrow c}$. Im ungünstigsten Fall findet man keine steuerbare Teilsprache, also $K^{\uparrow C} = \emptyset$. Die Definition und Algorithmen für supremale steuerbare Teilsprache finden sich in [4], [30] und [2].

Ist eine Spezifikation gefunden, die die Steuerbarkeitsbedingung erfüllt (Gleichung 2.30), muss noch die Existenzbedingung für eine nicht blockierende Steuerung überprüft werden. Die Steuerbarkeit besagt lediglich, dass eine Steuerung S für eine vorgegebene Spezifikation K und eine gegebene Strecke G grundsätzlich existiert, aber ob diese Steuerung

bezüglich G nicht blockierend ist, wird in der Steuerbarkeitsbedingung nicht berücksichtigt. Formal wird die Eigenschaft des Nichtblockierens wie folgt überprüft:

$$L_m(S/G) = K \wedge L(S/G) = \overline{K} \tag{2.31}$$

Diese Bedingung wird genau dann erfüllt, wenn

$$K = \overline{K} \cap L_m(G). \tag{2.32}$$

Diese Eigenschaft wird $L_m(G)$-Abgeschlossenheit von K genannt. Bei der grafischen Darstellung des Generators lautet dies wie folgt: Ist eine Spezifikation K steuerbar und nichtblockierend bezüglich G, so kann der Zustandsraum von $K \cap G$ nicht verlassen werden und somit ist das System immer in der Lage, einen markierten Zustand zu erreichen.

In [30] sind zwei Steuerungssyntheseprobleme und ihre Lösungen zusammengefasst. Das Basic Supervisory Control Problem-Nonblocking Case (BSCP-NB) und das Basic Supervisory Control and Observation Problem-Nonblocking Case (BSCOP-NB). Diese Steuerungssyntheseprobleme definieren Ansätze für die Berechnung eines Supervisors. In dieser Arbeit wird nur das BSCP-NB betrachtet. Probleme dieser Art beinhalten Strecken mit nicht steuerbaren aber keine mit nicht beobachtbaren Ereignissen. Das Problem wird nach [30] folgendermaßen definiert:

Definition 2.10 (BSCP-Nichtblockierend (BSCP-NB)) *Gegeben sei ein DES mit Ereignisalphabet Σ, der nicht steuerbaren Ereignismenge $\Sigma_{uc} \subseteq \Sigma$ und eine $L_m(G)$-abgeschlossene Spezifikation $K \subseteq L_m(G)$. Bestimme eine nicht blockierende Steuerung S, sodass gilt:*

1. $L_m(S/G) \subseteq K$
2. $L_m(S/G)$ *„maximal“ ist, also für jede andere nicht blockierende Steuerung S' mit $L_m(S/G) \subseteq K$ gilt:*

$$L_m(S'/G) \subseteq L_m(S/G) \tag{2.33}$$

Die erste Anforderung stellt sicher, dass die Spezifikation nicht verletzt wird. Die zweite Anforderung bedeutet, dass die gesuchte Steuerung S minimal restriktiv ist. Für die Lösung dieses Problems ist die gesuchte Steuerung folgendermaßen zu wählen:

$$L(S/G) = \overline{K^{\uparrow C}} \tag{2.34}$$

Die gefundene Steuerung kann entweder durch die Liste ihrer Steuereingriffe oder als Akzeptor, der die Sprache $\overline{K^{\uparrow C}}$ markiert, repräsentiert werden. Die Steuereingriffe $S(s)$ lassen sich nach [30] formal definieren durch

$$S(s) = [\Sigma_{uc} \cap \{\sigma \in \Sigma | \delta(\delta(x_0, s), \sigma)\}] \cup \{\sigma \in \Sigma_c | s\sigma \in \overline{K^{\uparrow C}}\}. \tag{2.35}$$

Für die Repräsentation als Akzeptor wird ein Akzeptor $R = \{Y, \Sigma, \zeta, y_0, Y_m\}$ konstruiert, dass $Y = Y_m$, Σ gleich dem Ereignisalphabet der Strecke, $R = Trim(R)$ und ζ, sodass $L_m(R) = L(R) = \overline{K^{\uparrow C}} = K^{\uparrow C}$. Das Verhalten des geschlossenen Steuerkreises lässt sich dann nach [30] mit der Produktkomposition folgendermaßen berechnen:

$$L(G \parallel_{SPC} R) = L(G) \cap L(R) = L(G) \cap \overline{K^{\uparrow C}} = L(S/G) \tag{2.36}$$

$$L_m(G \parallel R) = L_m(G) \cap L_m(R) = \overline{K^{\uparrow C}} \cap L_m(G) = L(S/G) \cap L_m(G) = L_m(S/G) \tag{2.37}$$

2.2.2. Strukturelle Steuerungsentwurfsansätze

Der monolithische Entwurfsansatz setzt ein unstrukturiertes Gesamtmodell der Strecke und eine Gesamtspezifikation voraus. Die Anzahl der Zustände eines Gesamtmodells steigt exponentiell mit der Anzahl der Zustände der Teilsysteme. So können bei größeren ereignisdiskreten Systemen durch parallele Komposition aller Teilsysteme schnell Zustandsräume entstehen die mit Standardrechnern nicht handhabbar sind. Zudem können monolithische Supervisor sehr viele Zustände enthalten, die in einer Steuerungshardware viel Speicher und Rechenzeit verbrauchen. In der Praxis hat man nicht immer genügend Rechenzeit oder Speicherplatz. Nachfolgend werden drei strukturelle Ansätze der SCT vorgestellt, die im Laufe der Zeit aus dem ursprünglichen monolithischen Ansatz entstanden sind. Alle diese Ansätze basieren auf Generatoren und regulären Sprachen und legen den Fokus auf eine Reduzierung der Komplexität der Supervisor-Synthese.

Modularer Entwurfsansatz

Das Ziel des modularen Ansatzes liegt darin, die globale Steuerungsaufgabe durch die Modularisierung auf mehrere Supervisor aufzuteilen, die parallel auf die Strecke einwirken. Dieser Ansatz benötigt wie der monolithische Ansatz ein unstrukturiertes Gesamtmodell der Strecke und setzt voraus, dass alle Streckenereignisse $\sigma \in \Sigma$ global für jeden Supervisor

beobachtbar sind. Die Architektur dieses Ansatzes S_{mod}/G wird in Abbildung 2.4 beispielhaft für zwei Supervisor dargestellt.

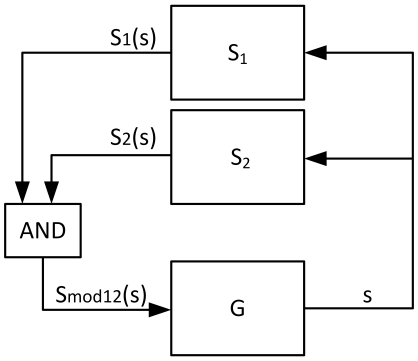


Abbildung 2.4.: Architektur des modularen Ansatzes.

Die Supervisor S_1 und S_2 überwachen den von G generierten String s, woraufhin jeder für sich die im nächsten Schritt erlaubten Steuereingriffe $S_1(s)$ und $S_2(s)$ berechnet. Ein Steuereingriff darf in der Strecke nur dann ausgeführt werden, wenn die Supervisor S_1 und S_2 diesen Eingriff erlauben, also $\sigma \in S_1(s) \wedge \sigma \in S_2(s)$. Die Steuereingriffe von $S_{mod} : L(G) \rightarrow 2^{\Sigma}$ setzen sich aus der Schnittmenge der Steuereingriffe der einzelnen Supervisor zusammen. Formal ist S_{mod} für n zulässige Supervisor nach [30] folgendermaßen definiert:

$$S_{mod}(s) = S_1(s) \cap S_2(s) \cap \ ... \ \cap S_n(s) \tag{2.38}$$

Aus Gleichung 2.38 folgt nach [30] für das Verhalten des geschlossenen Steuerkreises unter modularer Steuerung:

$$L(S_{mod}/G) = L(S_1/G) \cap L(S_2/G) \cap \ ... \ \cap L(S_n/G) \tag{2.39}$$

$$L_m(S_{mod}/G) = L_m(S_1/G) \cap L_m(S_2/G) \cap \ ... \ \cap L_m(S_n/G) \tag{2.40}$$

An einen mit dem modularen Ansatz berechneten Supervisor werden dieselben Güteanforderungen gestellt, wie an einen monolithischen Supervisor. Es wird gefordert, dass die Teilspezifikationen präfix-geschlossen sind. Das ist eine notwendige Bedingung, weil Steuerbarkeit bezüglich Schnittmengenbildung nur bei präfix-geschlossenen Sprachen erhalten bleibt.

Sind zwei Teilspezifikationen K_1 und K_2 steuerbar bezüglich G und präfix-geschlossen ($K_1 = \overline{K_1}, K_2 = \overline{K_2}$) so gilt nach [30]:

$$(K_1 \cap K_2)^{\uparrow C} = K_1^{\uparrow C} \cap K_2^{\uparrow C} \tag{2.41}$$

Das Modulare Supervisor Control Problem (MSCP) definiert, dass aus den Spezifikationen $K = K_1 \cap K_2 \cap ... \cap K_n$ mit $K_i = \overline{K_i}$, $i = [1, ..., n]$ ein modularer Supervisor S_{mod} berechnet werden soll, sodass gilt:

$$L(S_{mod}/G) = K^{\uparrow C} \tag{2.42}$$

Zur Lösung dieses Problems lassen sich wegen der präfix-geschlossenen Teilspezifikationen die einzelnen Supervisor S_i unabhängig voneinander berechnen sodass

$$L(S_i/G) = K_i^{\uparrow C} \text{ für } i = [1, ..., n] \tag{2.43}$$

gilt [30]. Anschließend ist S_{mod} nach 2.38 zu berechnen.

Die Eigenschaft des Nichtblockierens bleibt unter Schnittmengenbildung nicht erhalten, da sich die Supervisor gegenseitig blockieren können. Deshalb muss für den geschlossenen Steuerkreis zusätzlich noch die folgende Bedingung des Nichtblockierens für n Supervisor überprüft werden [30]:

$$\overline{L_m(S_1/G) \cap L_m(S_2/G) \cap \ ... \ \cap L_m(S_n/G)} = \overline{L_(S_1/G)} \cap \overline{L_(S_2/G)} \cap \ ... \ \cap \overline{L_(S_n/G)} \tag{2.44}$$

Da eine parallele Komposition der Generatoren der modularen Supervisor bei diesem Ansatz wegfällt, kann der modularer Entwurf den Zustandsraum der Supervisor gegenüber dem ursprünglichen monolithischen Modell signifikant verkleinern. Dadurch reduziert sich der Modellierungs- und Rechenaufwand gegenüber dem monolithischen ebenfalls signifikant.

Lokal-modularer Entwurfsansatz

Der lokal-modulare Entwurfsansatz ist als eine Erweiterung des monolithischen Ansatzes von Max. H. de Queiroz und J. E. R. Cury mit dem Ziel eingeführt, sowohl die globale Steuerungsaufgabe als auch die Gesamtstrecke jeweils in mehrere Teile aufzuteilen. Alle Defi-

nitionen, Begriffe und Zusammenhänge für diesen Ansatz wurden aus [19], [20] und [30] entnommen.

Dieser Ansatz setzt kein unstrukturiertes Gesamtmodell der Strecke voraus. Für jede Spezifikation wird ein lokales System berechnet. Dafür werden diejenigen Streckenkomponenten, die mit einer Spezifikation gemeinsame Ereignisse haben, durch parallele Komposition zu einem lokalen System zusammengefasst. Danach wird zu jeder Spezifikation auf Basis ihres lokalen Systems ein lokaler Supervisor bestimmt. Jeder Supervisor beobachtet und steuert im geschlossenen Steuerkreis ein bestimmtes lokales System (Teilsystem). Alle lokalen Supervisor zusammen erfüllen die globale Steuerungsaufgabe. Die Architektur des lokalmodularen Ansatzes ist für zwei Supervisor in Abbildung 2.5 dargestellt.

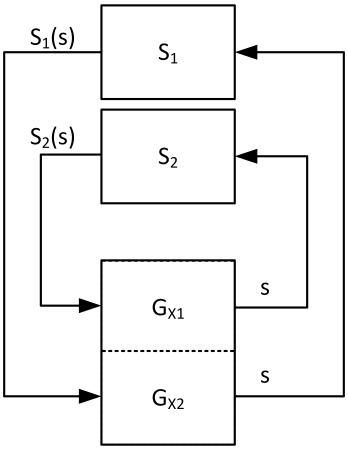


Abbildung 2.5.: Architektur des lokal-modularen Ansatzes.

Als erstes wird das Kompositionssystem aus n' Komponenten $G_i^{'}$ gebildet. Daraus ergeben sich das Gesamtmodell $G = \|_{i=1}^{n'} G_i^i$ und das Streckenalphabet $\Sigma = \bigcup_{i=1}^{n'} \Sigma_i^{'}$. Haben alle Teilmodelle disjunkte Ereignisalphabete, so sind sie asynchron zueinander und das Kompositionssystem wird Produktsystem (PS) genannt. Das Produktsystem stellt also das ungesteuerte Verhalten des Gesamtsystems dar. Das feinste Produktsystem ist das Produktsystem mit der größtmöglichen Anzahl an asynchronen Komponenten. Komponenten, die von einer Spezifikation zwanghaft synchronisiert werden, werden einem lokalen System zugeordnet.

Definition 2.11 (Lokale Systeme) *Sei G ein Produktsystem mit n Komponenten G_i und seinen K_{Xj} m gegebene Spezifikationen definiert über $\Sigma_{Xj} \subseteq \Sigma$. Dann sind die lokalen Systeme G_{Xj}, welche jeweils die von einer Spezifikation K_{Xj} zwanghaft synchronisierten Komponenten G_i enthalten definiert als*

$$G_{Xj} = \|_{i \in N_{Xj}} G_i \; mit \; N_{Xj} = \{k \in \{1, ..., n\} | \Sigma_k \cap \Sigma_{Xj} \neq \emptyset\}. \tag{2.45}$$

Inhärent asynchrone Komponenten, die durch keine Spezifikation eingeschränkt werden, lassen sich auch keinem lokalen System G_{Xj} zuordnen und können weggelassen werden. Daraus ergibt sich das eingeschränkte System

$$G_e = \|_{j=1}^{m} \text{ mit } \Sigma_e = \bigcup_{j=1}^{m} \Sigma_{Xj}. \tag{2.46}$$

Nun können die Spezifikationen K_{Xj} an die lokalen Systeme G_{Xj}, das eingeschränkte System G_e und das Gesamtsystem G folgendermaßen angepasst werden:

$$E_{Xj} = K_{Xj} \|_{SYPC} L_m(G_{Xj}) \tag{2.47}$$

$$E_{Xj_e} = K_{Xj} \|_{SYPC} L_m(G_e) \tag{2.48}$$

$$E_X = K_{Xj} \|_{SYPC} L_m(G) \tag{2.49}$$

Der parallele Kompositionsoperator $\|_{SYPC}$ ist auch für Sprachen definiert. Die Spezifikationen in den Gleichungen 2.47, 2.48 und 2.49 sind Sprachen. Für Generatoren gilt auch $E_{Xj} = K_{Xj} \|_{SYPC} G_{Xj}$. Für das bessere Verständnis der Anpassung des Systemmodells und der Spezifikationen wurde das Beispiel aus [30, S.158] aufgegriffen. In diesem Beispiel besteht das Gesamtsystem aus fünf Komponenten G_i und ihren Ereignisalphabeten Σ_i mit $i = [1, ..., 5]$. $K_{X1} \subset \Sigma_{X1}^*$ und $K_{X2} \subset \Sigma_{X2}^*$ sind lokale Spezifikationen. Das Venn-Diagramm in Abbildung 2.6 zeigt die Relationen zwischen den Ereignisalphabeten.

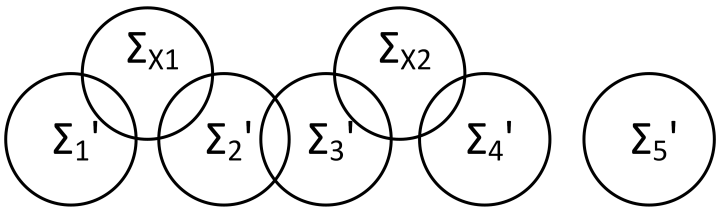


Abbildung 2.6.: Venn-Diagramm der Ereignisalphabete [30, S.158].

Die Komponenten G_2 und G_3 sind synchron, da sie gemeinsame Ereignisse in ihren Ereignisalphabeten haben. Das feinste Produktsystem ist: $G_1 = G_1^{'}$, $G_2 = G_2^{'} \|_{SYPC} G_3^{'}$, $G_3 = G_4^{'}$ und $G_4 = G_5^{'}$. G_4 wird durch keine der beiden Spezifikationen synchronisiert und kann weggelassen werden. Das eingeschränkte System ist somit $G_e = G_1 \|_{SYPC} G_2 \|_{SYPC} G_3$. Die lokalen Systeme sind: $G_{X1} = G_1 \|_{SYPC} G_2$ und $G_{X2} = G_2 \|_{SYPC} G_3$. Die Anpassung der lokalen Spezifikation an ihr lokales System erfolgt durch: $E_{X1} = K_{X1} \|_{SYPC} G_{X1}$ und $E_{X2} = K_{X2} \|_{SYPC} G_{X2}$.

Zur Sicherstellung, dass die lokalen Spezifikationen durch ihre Steuereingriffe das System und sich gegenseitig nicht blockieren, müssen sie auf lokale Modularität überprüft werden. Die lokale Modularität wird nach [30] folgendermaßen definiert:

Definition 2.12 (Lokale Modularität) *Sei I eine beliebige Indexmenge und $L_i \subseteq \Sigma_i^*$, $i \in I$. Die Menge $\{L_i, i \in I\}$ ist lokal modular, wenn*

$$\|_{i \in I} \overline{L_i} = \overline{\|_{i \in I} L_i}. \tag{2.50}$$

Nach dieser Definition kann die lokale Modularität für n supremale steuerbare Spezifikationen und bezüglich ihrer lokalen Systeme auch folgendermaßen formuliert werden:

$$\|_{i=1}^{n} \overline{L_m(supC(E_{Xi}, G_{Xi}))} = \overline{\|_{i=1}^{n} L_m(supC(E_{Xi}, G_{Xi}))} \tag{2.51}$$

Die folgende Gleichung basiert auf Generatoren und ist äquivalent zur Gleichung 2.51:

$$Trim(E_X) = \|_{i=1}^{n} E_{Xi} \tag{2.52}$$

In Gleichung 2.52 werden die Generatoren aller lokalen supremalen steuerbaren Spezifikationen durch parallele Komposition in einem Generator zusammengefasst. Ist der resultierende Generator trim, so ist jede Spezifikation lokal-modular bezüglich aller anderen Spezifikationen. Diese Bedingung stellt außerdem sicher, dass die lokal-modularen Supervisor in ihrer Gesamtheit das gleiche leisten, wie ein monolithischer Supervisor.

Die folgende Definition aus [30] zeigt den Zusammenhang für einen lokal-modularen Entwurf mit zwei Spezifikationen:

Definition 2.13 (Lokal-Modularer Entwurf für 2 Spezifikationen) *Sei G ein Kompositionssystem mit n' Komponenten G_i' und zwei lokalen Spezifikationen K_{X1} beziehungsweise K_{KX2}. Es sei $supC(E_{X1}, G_{X1})$ die supremale steuerbare Teilsprache von E_{X1} bezüglich G_{X1} und $supC(E_{X2}, G_{X2})$ die supremale steuerbare Teilsprache von E_{X2} bezüglich G_{X2}. Ist Menge $\{supC(E_{X1}, G_{X1}), supC(E_{X2}, G_{X2})\}$ lokal modular, dann gilt*

$$supC(E_{X1_e} \cap E_{X2_e}, G_e) = supC(E_{X1}, G_{X1}) \|_{SYPC} supC(E_{X2}, G_{X2}). \tag{2.53}$$

Die Definition besagt, dass es zum lokal-modularen Entwurf ausreichend ist, die Supervisor mittels $L(S_1/G_{X1}) = supC(E_{X1}, G_{X1})$ und $L(S_2/G_{X2}) = supC(E_{X2}, G_{X2})$ zu berechnen, vorausgesetzt, die Spezifikationen sind lokal modular. Die Definition kann dementsprechend für n Spezifikationen erweitert werden.

Der lokal-modulare Ansatz ermöglicht eine dezentrale Implementierung der Steuerungen. Die algorithmische Komplexität dieses Ansatzes ist immer geringer oder im schlechtesten Fall identisch mit der Komplexität des modularen Ansatzes. Der schlechteste Fall trifft zu, wenn eine lokale Spezifikation alle Streckenkomponenten synchronisiert. In diesem Fall ist ihr lokales System wieder das Gesamtsystem.

Dezentraler Entwurfsansatz

Der dezentrale Ansatz wurde von F. Lin und W. M. Wonham eingeführt [32]. Sie untersuchten in [33] das Supervisorsyntheseproblem mit nicht beobachtbaren Ereignissen. Im Grunde ist dieser Ansatz nur eine Erweiterung des modularen Ansatzes mit eingeschränkter Beobachtbarkeit. Zum Beispiel kann durch verteilte Implementierung der Supervisor auf mehrere Steuerungshardware das Auftreten bestimmter Ereignisse nur dezentral in der Steuerungshardware stattfinden und von außen für einen Supervisor, der auf einer anderen Steuerungshardware implementiert ist, nicht sichtbar sein. Jeder Supervisor erhält dann nur eine Teilmenge der Streckenereignisse $\Sigma_{i,o} \subseteq \Sigma$. Die Architektur dieses Ansatzes ist in Abbildung 2.7 dargestellt.

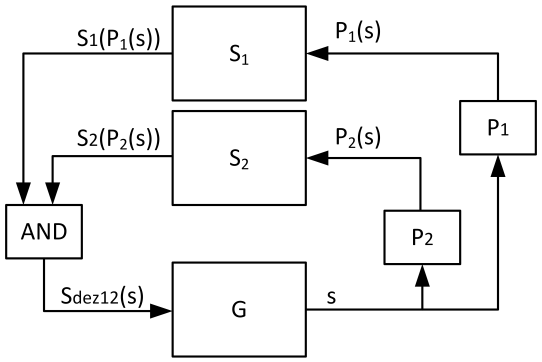


Abbildung 2.7.: Architektur des dezentralen Ansatzes.

Durch die natürliche Projektion $P : \Sigma \rightarrow \Sigma_o$ wird für jeden Supervisor ein Abbild $P_i(s)$ berechnet. Auf Details des dezentralen Ansatzes wird hier nicht weiter eingegangen, es sei auf [30], [32] und [33] verwiesen.

2.3. Entwicklungswerkzeuge

In diesem Unterkapitel werden die in dieser Arbeit verwendeten Entwicklungswerkzeuge überblicksartig vorgestellt.

2.3.1. Siemens Totally Integrated Automation Portal

Zur Programmierung der realen Hardware wurde das Totally Integrated Automation Portal (TIA-Portal) verwendet. Das TIA-Portal ist ein Engineering-Framework in dem sämtliche Werkzeuge für Projektierung, Steuerungsprogrammierung und Diagnose von Controllern, HMI-Systemen, Antrieben und Motion-Controllern von Siemens integriert sind.

Standardmäßig stellt das TIA-Portal fünf Programmiersprachen zur Verfügung [1]:

- Anweisungsliste (AWL), vergleichbar mit Assembler
- Strukturierte Sprache (S7-SCL), Pascal-ähnliche Hochsprache
- Kontaktplan (KOP) vergleichbar mit einem Stromlaufplan
- Funktionsbausteinsprache (FUP), vergleichbar mit Logik-Schaltungen
- Ablaufsprache (S7-Graph), ähnelt einem Zustandsdiagramm

Die neueren SIMATIC-S7-Controller, die von Siemens angeboten werden (insbesondere S7-1500er Serien), können nur mit dem TIA-Portal programmiert werden. In dieser Arbeit wird das TIA-Portal in Version 14 verwendet.

2.3.2. Entwicklungstool DESTool

Das Entwicklungswerkzeug DESTool von der Friedrich-Alexander Universität ist ein grafisches Frontend zum Entwurf ereignisdiskreter Steuerungen. Die Generatorenmodelle können mit einem integrierten grafischen Editor per „Drag and Drop“ entworfen werden. Die eigentlichen Analyse- und Synthesefunktionen sind in der eingebundenen C++ Bibliothek libFAUDES [16] hinterlegt. In Tabelle 2.1 sind die in dieser Arbeit verwendeten DESTool-Operationen aufgelistet.

Funktionen	Beschreibung
Product	Berechnet die Produktkomposition von zwei oder mehr Generatoren (Strenge Synchronisation, SPC-Operator)
Parallel	Bildet das synchrone Produkt von zwei oder mehr Generatoren (Parallele Komposition, SYPC-Operator)
isTrim	Prüft, ob G nicht erreichbare oder nicht ko-erreichbare Zustände besitzt
isAccessible	Prüft, ob G nicht erreichbare Zustände besitzt
isCoaccessible	Prüft, ob G nicht ko-erreichbare Zustände besitzt
Trim	Berechnet für G den zugehörigen erreichbaren und co-erreichbaren Anteil
isDetermistic	Prüft, ob G deterministisch ist
isNonblocking	Prüft, ob zwei oder mehrere Generatoren nichtblockierend sind
isControllable	Prüft Spezifikation K auf Steuerbarkeit bezüglich Strecke G
SupConNB	Berechnet für eine gegebene Strecke und Spezifikation die supremale steuerbare Teilmenge
SupReduce	Berechnet einen reduzierten Supervisor nach [24]

Tabelle 2.1.: Verwendete DESTool-Operationen.

Eine detallierte Beschreibung dieser Operationen findet sich in [15]. Die in dieser Arbeit verwendete DESTool-Version ist 0.83 und die eingebundene Bibliothek libFAUDES hat die Version 2.28c.

2.3.3. Codegenerator ACArrow

Das Ergebnis einer Synthese sind Modelle. Um diese Modelle praktisch nutzen zu können, ist es erforderlich, sie in einen Automatisierungsgerät ablauffähigen Programmcode umzuwandeln. Das manuelle Programmieren der Steuerungshardware wäre sehr aufwendig und würde Fehlerquellen erzeugen. Zur Unterstützung des Steuerungsentwurfs in dieser Arbeit wird der in [6] entwickelte Codegenerator ACArrow (Automatic Codegenerator Arrow) eingesetzt. ACArrow erhält als Eingabe die von DESTool erzeugte Projektdatei (.pro) und extrahiert daraus die zur Codegenerierung relevanten Daten. Dazu zählen Zustände, Zustandsübergänge, und Ereignisse. Es kann zwischen den Programmiersprachen STL/AWL und SCL ausgewählt werden. Die Syntax, IEC oder Siemens ist ebenfalls für beide Programmiersprachen

auswählbar. Als Ausgabe erzeugt ACArrow je nach gewähltem Ansatz, der Syntax und der Programmiersprache verschiedene Dateien, die den Programmcode enthalten. Des Weiteren enthält ACArrow einen Codegenerator für Petri-Netze. Er wird aber im Rahmen dieser Arbeit nicht verwendet.

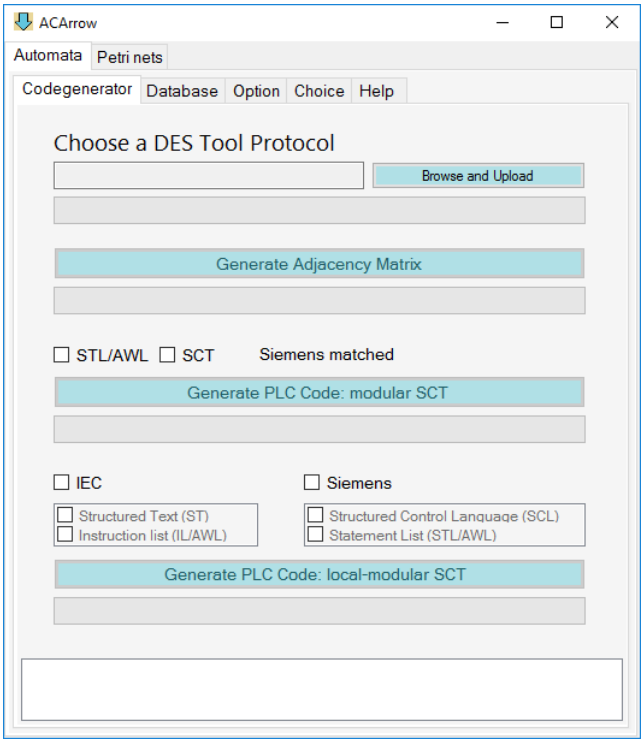


Abbildung 2.8.: Screenshot von ACArrow.

Für die Codegenerierung gibt es in der Literatur verschiedene Ansätze. Ein Ansatz für den lokal-modularen Entwurf ist beispielsweise DECON9-Ansatz von Leal, da Cruz und da Silva Hounsell [9]. Die Codegenerierung in ACArrow erfolgt für den lokal-modularen Entwurf auch mit DECON9-Ansatz. Ein anderer Codegenerierungsansatz für den modularen Entwurf zeigt die Masterarbeit [27]. Er basiert auf der Idee, die Supervisor zunächst in Adjazenzmatrizen umzuwandeln und anschließend daraus den Programmcode zu erzeugen. Dieser Ansatz ist ebenfalls in ACArrow implementiert.

3. Beschreibung der Anlage

Im Folgenden wird die verwendete Montageanlage mit ihren Komponenten beschrieben. Zunächst wird die verwendete Nomenklatur vorgestellt und eine kurze Prozessbeschreibung angegeben. Im Kapitel 3.3.7 wird auf die automatisierungstechnische Hardware eingegangen.

3.1. Nomenklatur und Anlagenschema

Die in dieser Arbeit verwendete Montageanlage ist aus einer Reihe von Komponenten aufgebaut. Jede Komponente ist durch einen Namen eindeutig identifizierbar. So lässt sich später eindeutig feststellen, welche Komponente und welche Aktion dem Ereignis zugeordnet ist. Außerdem kann anhand der Ereignisnamen auch festgestellt werden, ob es sich bei dem jeweiligen Ereignis um ein steuerbares oder nicht steuerbares Ereignis handelt.

In dieser Arbeit werden die Namen der Ereignisse wie folgt zusammengesetzt:

HARDWAREKLASSE.STATION.KOMPONENTE[.NUMMER].Aktion

Sollten sich auf einer Station mehrere Instanzen einer Komponente befinden, so werden diese mit einer zusätzlichen Nummerierung gekennzeichnet. Die verwendete Nomenklatur ist in Tabelle 3.1 erläutert. Eine vollständige Auflistung aller verwendeter Ereignisse enthält die Tabelle 5.9.

Die Montageanlage besteht aus zwei Montagestationen (Station 30 und Station 50), zwei Prüfstationen (Station 40 und Station 60), einer Lagerstation (Station 20) und einem Werkstückpaletten-Transfersystem (TS1). Die Arbeitsstationen 30, 40, 50 und 60 sind über das Transfersystem miteinander gekoppelt. Eine Leitstation (Station 10) kommuniziert mit den anderen Stationen der Anlage, steuert die Komponenten des Transfersystems und koordiniert die gesamten Montage- und Prüfprozesse. Des Weiteren wird die Primärenergie der gesamten Anlage (Druckluft und elektrische Stromversorgung) von der Leitstation aus an einzelne Stationen verteilt.

Hardwareklasse	**Beschreibung**
E	Beschreibt nicht steuerbare Ereignisse, wie zum Beispiel einen Eingang der SPS oder eine Rückmeldung von der Steuerung
STR	Beschreibt steuerbare Ereignisse, wie zum Beispiel Steuerbefehle
Station	**Beschreibung**
ST10	Leitstation (Station 10)
ST20	Lagerstation (Station 20)
ST30	Montagestation (Station 30)
ST40	Elektrische Endprüfstation (Station 40)
ST50	Pneumatische Presse (Station 50)
ST60	Optische Endprüfstation (Station 60)
TS1	Werkstückpaletten-Transfersystem
Komponente	**Beschreibung**
ROB	Robotereinheit
LIHNDL	Handlingeinheit mit Linearachse
HNDL1	Handlingeinheit 1
HNDL2	Handlingeinheit 2
AS	Abschieber
VS	Vision-Sensor
B	Näherungssensor, Lichtschranke, Palettensensor
HP	Palettenhub- und -positioniereinheit
PS	Palettenstopper
Nummer	**Beschreibung**
k=1,2,3,...	

Tabelle 3.1.: Ereignis-Notation.

Abbildung 3.1 stellt den grundlegenden physikalischen Aufbau der Anlage und die Architektur der automatisierungstechnischen Hardware dar. Die Richtung der Pfeile zeigt die Laufrichtung der Werkstücke zwischen den einzelnen Stationen an. Die gestrichelten Linien repräsentieren bidirektionale Kommunikationen. Die Stationen werden dezentral mit je einer eigenen speicherprogrammierbaren Siemens-Steuerung gesteuert. Diese sind an einem Profinet-IO-System angebunden und kommunizieren über Transferbereiche (E/A-Bereiche) miteinander. Die sämtlichen Buskommunikationen laufen auf der Leitstation zusammen. Des Weiteren beinhaltet die Anlage eine Mitkopplung (rot in Abbildung 3.1 dargestellt), über die die Werkstücke (reparabel) wieder in den Montageprozess eingelastet werden können.

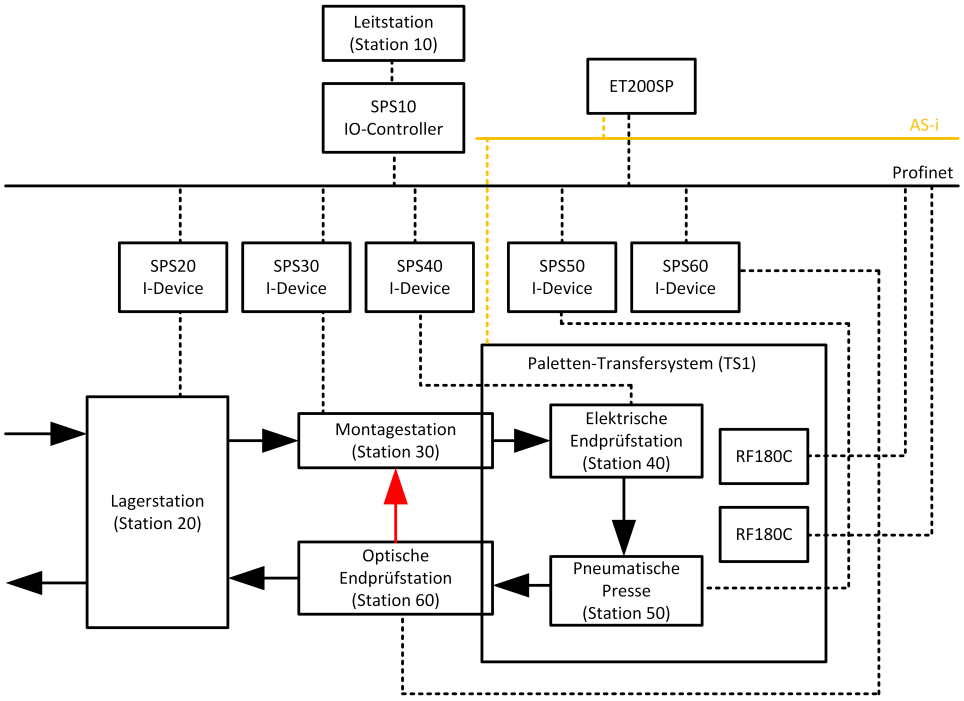


Abbildung 3.1.: Physikalischer Aufbau der Anlage und Architektur der automatisierungstechnischen Hardware.

3.2. Prozessbeschreibung

Auf der Montageanlage werden Relaiskarten mit einem oder zwei Schaltkontakten (einpolig oder zweipolig) entsprechend der Darstellung in Abbildung 3.2 in einem Gehäuse mit Deckel montiert. Ein fertiges Werkstück (Endprodukt) besteht also aus drei Einzelteilen, die in der Montage zusammengefügt und geprüft werden: einem Unterteil, einer Relaiskarte (einpolig oder zweipolig) und einem Deckel mit aufgedrucktem „HAW-Logo“.

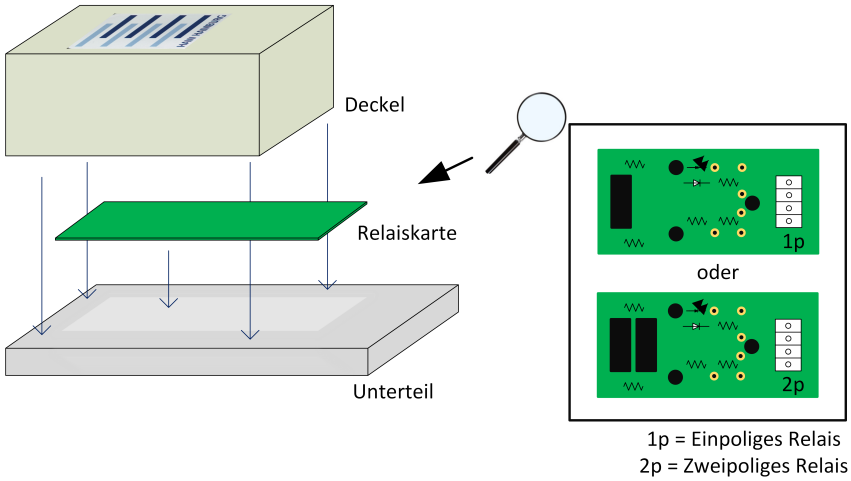


Abbildung 3.2.: Einzelteilen des fertigen Werkstücks.

Abbildung 3.3 stellt die gesamte Anlage mit ihren Komponenten schematisch dar. Aus Gründen der Übersichtlichkeit sind lediglich die Bereiche und Komponenten angegeben, die für den Steuerungsentwurf in dieser Arbeit relevant sind.

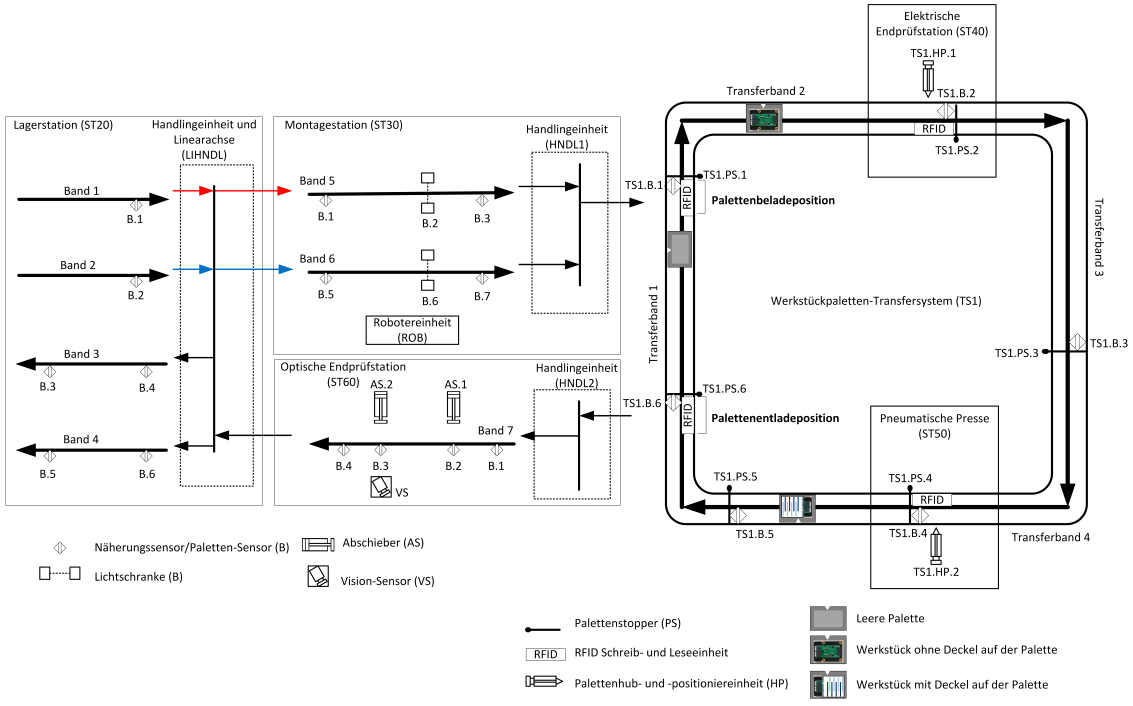


Abbildung 3.3.: Schematischer Aufbau der Gesamtanlage.

Der Montageprozess beginnt und endet in der Lagerstation (Station 20). Der Lagerbestand setzt sich aus unbearbeiteten Werkstücken (Unterteilen) und fertigen Werkstücken zusammen. Als erste Arbeitsstation folgt nach der Lagerstation eine Montagestation (Station 30), in der mittels Robotereinheit eine Relaiskarte auf dem Werkstück eingesetzt wird. Die Werkstücke werden, nach dem sie vormontiert sind, aus der Montagestation ausgelastet und mit einer Handlingeinheit auf einer vor dieser Station wartenden Palette des Transfersystems platziert. Das Transfersystem ist ein geschlossenes Umlaufsystem und ist mit RFID-System, Sensoren und Aktoren an den relevanten Positionen ausgestattet.

Das Werkstück wird auf der Palette durch weitere Prüf- und Montagestationen gefahren. An den Paletten sind RFID-Tags montiert, die Informationen über den Relaistyp (einpolig oder zweipolig) und Werkstückzustand (defekt oder nicht defekt) enthalten. Der Relaistyp wird im Zuge der Beladung auf das RFID-System gesendet und der entsprechende RFID-Tag beschrieben. In den nachfolgenden Stationen werden die Informationen aus dem RFID-Tag ausgelesen und der entsprechende Prozess durchgeführt. Auf dem RFID-Tag wird der Wert 0, 1, 2, 11, 12, 241 oder 242 gespeichert. Diese Werte haben folgende Bedeutungen:

Wert	**Bedeutung**
0	Palette ist leer
1	Relais einpolig und noch nicht geprüft
2	Relais zweipolig und noch nicht geprüft
11	Relais einpolig, geprüft und nicht defekt
12	Relais zweipolig, geprüft und nicht defekt
241	Relais einpolig und defekt
242	Relais zweipolig und defekt

Tabelle 3.2.: Bedeutung der Werte auf dem RFID-Tag.

An der elektrischen Endprüfstation (Station 40) durchläuft das Werkstück eine elektrische Prüfung. Aus dem RFID-Tag der angekommenen Palette wird der Relaistyp ausgelesen (RFID-Wert 1 oder 2) und die entsprechende Prüfroutine im Steuerungsprogramm gestartet. Im Anschluss der elektrischen Prüfung wird das Ergebnis der Prüfung auf dem RFID-Tag gespeichert und danach wird das Werkstück auf der Palette zur nächsten Station (Station 50) transportiert, wo es mit einem Deckel versehen wird. Defekte Werkstücke werden an dieser Station erkannt und nicht verdeckelt.

Über eine Pufferstrecke wird das Werkstück wieder dem Transfersystem entnommen und an die optische Endprüfstation (Station 60) übergeben. In dieser Station wird mittels Bilderverarbeitung die Deckelmontage des Werkstücks geprüft. Ist die Deckelmontage nicht in Ordnung, wird es aussortiert. Im Falle einer positiven optischen Endprüfung wird das Werkstück von

dieser Station entnommen und in der Lagerstation gelagert. Des Weiteren werden defekte Werkstücke an dieser Station über den RFID-Tag erkannt und aussortiert oder durch den Roboter demontiert und Teile davon wieder dem Montageprozess zugeführt.

3.3. Beschreibung der Komponenten

Nachfolgend werden die einzelnen Arbeitsstationen und deren Komponenten bezüglich Aufbau und Funktion detailliert beschrieben. Weiterhin wird in Unterkapitel 3.3.7 die automatisierungstechnische Hardware vorgestellt.

3.3.1. Lagerstation (Station 20)

Die Lagerstation hat die Aufgabe der Zuführung der unbearbeiteten Werkstücke (Unterteile) in den Montageprozess und der Lagerung der fertigen Werkstücke. Sie besteht aus vier Bandkomponenten, einer Linearachse und einer Handlingeinheit. Die Bänder haben jeweils ein Maß von $680mm \times 70mm$ (L/B). Abbildung 3.4 zeigt eine schematische Darstellung der Lagerstation mit ihren Komponenten.

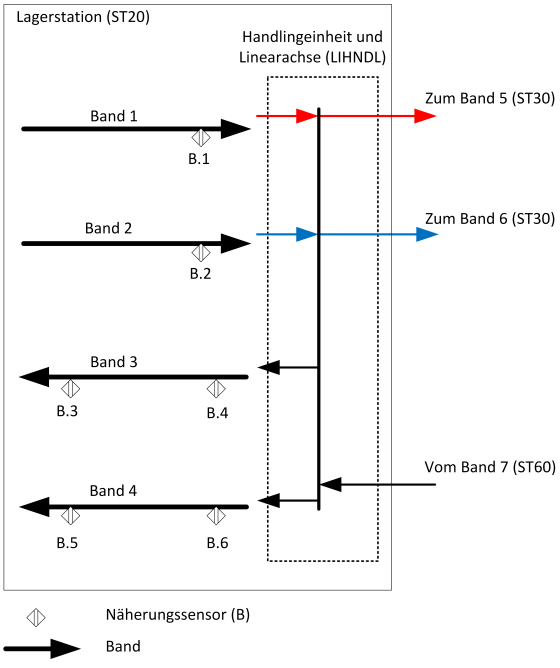


Abbildung 3.4.: Schematischer Aufbau der Lagerstation.

Die Bänder 1 und 2 dienen als Eingangslager. Auf diesen befinden sich unbearbeitete Werkstücke. Das Werkstück auf dem Band *i* für $i = [1, 2]$ läuft bei Rechtsdrehung des Bandmotors nach rechts bis zum Erreichen des Näherungssensors *B.i*. Auf den Bändern 3 und 4 werden fertige Werkstücke gelagert. Sie dienen als Ausgangslager. Sobald der Sensor *B.j* für $j = [4, 6]$ ein Werkstück detektiert, dreht der entsprechende Bandmotor für eine bestimmte Zeit links und transportiert das Werkstück ein Stück nach links. Befindet sich ein fertiges Werkstück im Ansprechbereich des Sensors *B.3* beziehungsweise *B.4*, so gilt das Band 3 beziehungsweise 4 als voll.

Handlingeinheit und Linearachse *LIHNDL*

Die Station 30 wird über eine pneumatische Dreiachs-Handlingeinheit vom Eingangslager mit unbearbeiteten Werkstücken beliefert. Zusätzlich hat die Handlingeinheit die Aufgabe, die fertigen Werkstücke aus Station 60 zum Ausgangslager zu bringen. Sie verfügt über einen pneumatischen Werkstückgreifer. Mit einer Linearachse wird sie zu den Bändern gefahren. Die Linearachse wird durch einen Servomotor angesteuert, auf dem ein Absolutwertgeber montiert ist. Der Motor wird über den Umrichter Sinamics S120 geregelt. Der Umrichter besitzt eine Profinet-Schnittstelle, über die er mit der übergeordneten SPS der Station 20 kommuniziert.

Die Handlingeinheit und Linearachse können folgende Aufgaben ausführen:

- Werkstück vom Band 1 (Station 20) zum Band 5 (Station 30) bringen,
- Werkstück vom Band 2 (Station 20) zum Band 6 (Station 30) bringen,
- Werkstück vom Band 7 (Station 60) zum Band 3 (Station 20) bringen,
- Werkstück vom Band 7 (Station 60) zum Band 4 (Station 20) bringen.

3.3.2. Montagestation (Station 30)

Die in dieser Station durchzuführende Aufgabe ist, die Relaiskarten auf die ankommenden Werkstücke zu montieren. Diese Station besteht im Wesentlichen aus zwei zueinander parallel angeordneten Bandkomponenten mit einer Länge und Breite von $980mm \times 70mm$ (L/B), einer Robotereinheit für die Unterstützung des Montageprozesses und einer pneumatischen Handlingeinheit. Der Roboter ist mit einem Sauggreifer zum Saugen von Relaiskarten ausgestattet. Abbildung 3.5 stellt die Montagestation schematisch dar.

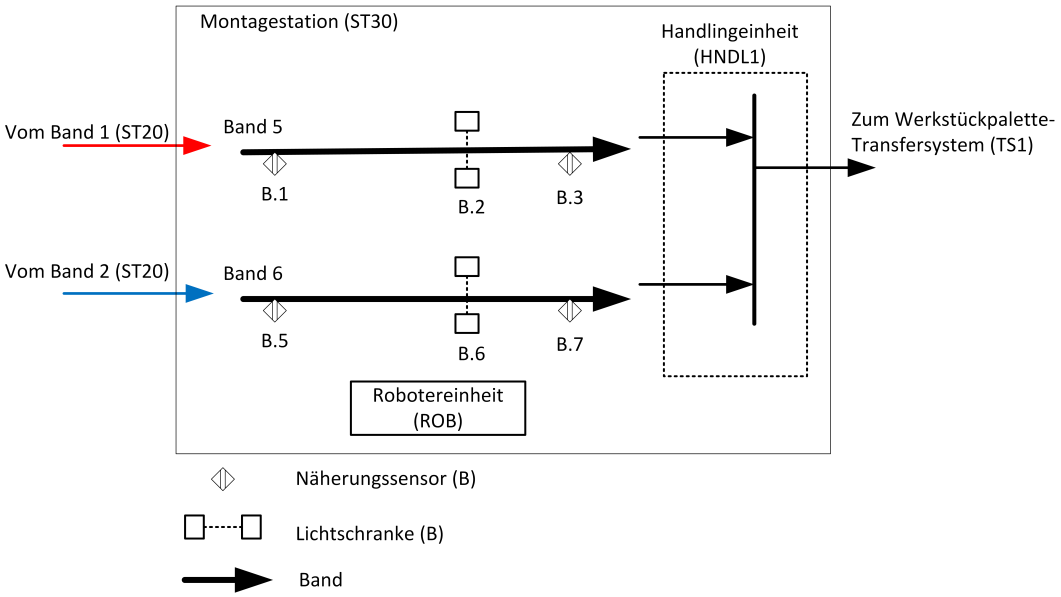


Abbildung 3.5.: Schematischer Aufbau der Montagestation.

Aus zwei Rutschen, die in Abbildung 3.5 nicht abgebildet sind, werden einpolige- und zweipolige Relaiskarten bereitgestellt. Am Ende der beiden Rutschen befinden sich Sensoren, die erkennen, ob die Rutschen belegt sind oder nicht. Die Bänder sind identisch in ihrem Aufbau. Bei Rechtsdrehung des Bandmotors läuft das Werkstück auf dem entsprechenden Band in Richtung des Transfersystems. Auf beiden Bändern können die Positionen Bandanfang, Robotermontage und Bandende durch Näherungssensoren und Lichtschranken detektiert werden. Sobald die Lichtschranke *ST30.B.i* für $i = [2, 6]$ ein Werkstück detektiert, wird das entsprechende Band angehalten und mittels Roboter eine Relaiskarte auf dem Werkstück eingesetzt. Da nur ein Roboter für beide Bänder zur Verfügung steht, muss ein Werkstück gegebenenfalls warten, bis der Roboter mit dem anderen Werkstück fertig ist. Des Weiteren werden auf dem Band 5 einpolige Relaiskarten eingesetzt und auf dem Band 6 zweipolige Relaiskarten. Deshalb kann ein Werkstück aus der Lagerstation nicht zu einem beliebigen Band zugeführt werden.

Robotereinheit *ROB*

Der Roboter ist ein Sechs-Achs-Roboter von DENSO und wird von dem Roboter-Controller RC8M gesteuert. Der Controller verfügt über digitale Ein- und Ausgänge, die mit den Ein- und Ausgängen der übergeordneten SPS der Station 30 verknüpft sind. Über die Eingänge nimmt er Befehle von der SPS entgegen und steuert den Roboter. Des Weiteren gibt er über die Ausgänge Signale an die SPS zurück.

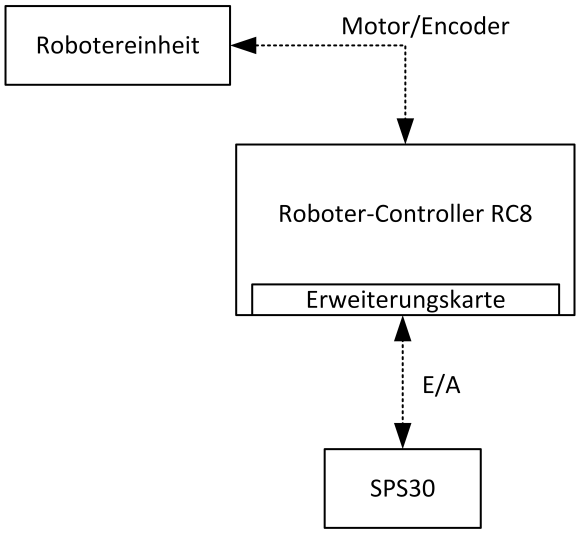


Abbildung 3.6.: Systemkonfiguration - SPS, Roboter und Roboter-Controller

Für den Roboter wurden bereits durch das Roboterprogramm auf dem Roboter-Controller folgende Aufträge definiert:

- Einpolige Relaiskarte von Rutsche 1 entnehmen und auf dem Werkstück einsetzen,
- Zweipolige Relaiskarte von Rutsche 2 entnehmen und auf dem Werkstück einsetzen,
- Relaiskarte entfernen, Werkstückunterteil vom Band 7 zum Band 5 bringen und anschließend eine einpolige Relaiskarte auf dem Werkstückunterteil einsetzen,
- Relaiskarte entfernen, Werkstückunterteil vom Band 7 zum Band 6 bringen und anschließend eine zweipolige Relaiskarte auf dem Werkstückunterteil einsetzen.

Handlingeinheit *HNDL1*

Im Ausgangsbereich dieser Station befindet sich eine pneumatische Dreiachs-Handlingeinheit, die die vormontierten Werkstücke von den Bändern einzeln entnimmt und auf einer vor dieser Station wartenden Werkstückpalette ablegt. Sie besteht aus vier Komponenten: der horizontalen Achse der vertikalen Achse, der Drehachse und dem Greifer. Durch die horizontale Achse können die Positionen Band 5 und Band 6 angefahren werden. Durch die vertikale Achse lässt sich der Greifer senken und heben. Die Drehachse lässt den Greifer in Richtung Station 30 beziehungsweise in Richtung Transfersystem steuern.

3.3.3. Elektrische Endprüfstation (Station 40)

Die Ankunft einer Palette an Station 40 wird von dem Palettensensor *TS1.B.2* detektiert. Hierauf wird die Palettenhub- und -positioniereinheit *HP.1* ausgefahren und die Palette fixiert. Diese Station hat die Aufgabe, die Elektronik auf der Relaiskarte auf die elektrische Funktion hin zu prüfen. Hierfür befinden sich mehrere Testpunkte in der Relaiskarte. Eine pneumatisch bewegte Handlingeinheit mit Prüfstiften kontaktiert die Testpunkte und überprüft durch eine entprechende Routine im Steuerungsprogramm die Funktionalität der Elektronik der Relaiskarte. Das Ergebnis der Prüfung wird der Leitstation mitgeteilt, die daraufhin das Ergebnis auf dem entsprechenden RFID-Tag speichert (RFID-Wert 11, 12, 241 oder 242). Die defekten Werkstücke sind reparabel. Sie werden in den nachfolgenden Stationen erkannt und anders gehandhabt.

3.3.4. Pneumatische Presse (Station 50)

Die Station 50 hat die Aufgabe, die durch Station 40 als in Ordnung bewerteten Werkstücke mit einem Deckel zu versehen. Hierfür befinden sich in dieser Station eine Dreiachs-Handlingeinheit, ein Zylindermagazin und ein Ausschieber. Die Ankunft einer Palette an dieser Station wird mit dem Palettensensor *TS1.B.4* detektiert. Daraufhin fährt die Palettenhub- und -positioniereinheit *HP.2* aus und fixiert die Palette. Ist auf dem RFID-Tag der Wert 241 oder 242 gespeichert, so handelt es sich um ein defektes Werkstück auf der Palette. In diesem Fall wird das Werkstück nicht verdeckelt. Die Palettenhub- und -positioniereinheit fährt wieder ein und gibt die Palette frei. Die Palette wird ebenfalls wieder sofort freigegeben, wenn auf dem RFID-Tag der Wert 0 (Palette leer) gespeichert ist. Ist auf dem RFID-Tag der Wert 11 oder 12 gespeichert, so wird aus dem Zylindermagazin ein Deckel ausgeschoben. Die Dreiachs-Handlingeinheit mit angebautem Unterdrucksauger fährt zum Deckel, saugt diesen an und fährt zum Werkstück, um den Deckel mit dem Werkstück zu verpressen. Nach der Montage fährt die Handlingeinheit wieder zu seiner Grundstellung. Danach wird die Palette freigegeben.

3.3.5. Optische Endprüfstation (Station 60)

Die Station 60 hat die Aufgabe der optischen Prüfung der verdeckelten Werkstücke. Sie besteht aus einer Bandkomponente, einem Bildverarbeitungssystem, zwei pneumatischen Abschiebern und einer Dreiachs-Handlingeinheit. Abbildung 3.7 zeigt eine schematische Darstellung der optischen Endprüfstation.

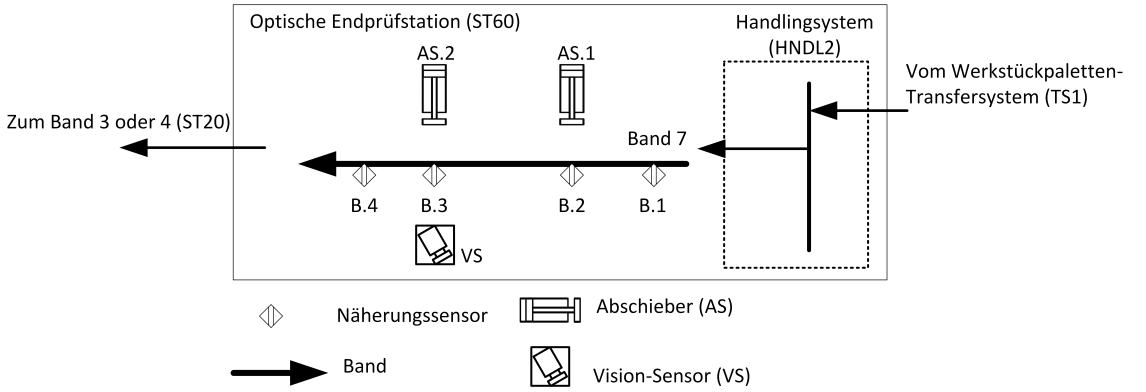


Abbildung 3.7.: Schematischer Aufbau der Station 60.

Die pneumatische Handlingeinheit *HNDL2* im Eingangsbereich der Station hat die Aufgabe, die Werkstücke aus den ankommenden Paletten zu entnehmen und dem Band 7 zuzuführen. Sie besteht aus drei Komponenten: einer vertikalen Achse, einer Drehachse und einem Greifer. Die vertikale Achse lässt sich heben und senken. Die Drehachse kann mit dem Greifer in Richtung des Transfersystems und in Richtung der Station angesteuert werden. Alle diese Komponenten werden in einer Schrittkette durch Aneinanderreihung von Aktionen gesteuert.

Das Band 7 hat ein Maß von $980mm \times 70mm$ (L/B) und verfügt über folgende Werkstücksensoren:

- Sensor am Bandanfang (*B.1*),
- Sensor vor dem ersten Abschieber (*B.2*),
- Sensor vor dem zweiten Abschieber (*B.3*),
- Sensor am Bandende (*B.4*).

Die defekten Werkstücke (RFID-Wert 241 oder 242) laufen bis vor dem Sensor *B.2* und werden dort entweder durch den ersten Abschieber *AS.1* aussortiert oder durch den Roboter demontiert und wieder in den Montageprozess eingelastet. Sie sind reparabel. Die verdeckelten Werkstücke hingegen laufen bis vor dem Sensor *B.3* und werden dort mittels eines Bildverarbeitungssystems optisch überprüft. Bei einem Fehlergebnis des Vision-Sensors wird das Werkstück mittels zweiten Abschiebers *AS.2* aussortiert. Diese Werkstücke sind nicht reparabel.

Bildverarbeitungssystem

Bei dem Bildverarbeitungssystem handelt es sich um Simatic MV440 der Firma Siemens AG. Es ist ein Vision-Sensor mit eingebauter Kamera, Prozessor für die Bildverarbeitung, Beleuchtung, Optik und Profinet-Kommunikationseinheit in einem Gehäuse. Auf dem Controller des Sensors ist ein Prüfprogramm implementiert, das die Deckelmontage der Werkstücke überprüft. Sollte beispielsweise bei einem Werkstück der Deckel falsch herum montiert sein („HAW“-Logo falsch herum) oder nicht richtig sitzen, wird es durch das Prüfprogramm als nicht in Ordnung klassifiziert und der übergeordneten SPS der Station 20 gemeldet.

3.3.6. Werkstückpaletten-Transfersystem (TS1)

Die Grundaufgaben des Werkstückpaletten-Transfersystems sind Transportieren, Stoppen, Aufstauen und Vereinzeln von Werkstückpaletten. Es ist als staufähiges Umlaufsystem ausgeführt. Der Transport der Paletten erfolgt mittels vier Transferbändern, die in der schematischen Abbildung 3.8 des Transfersystems durch dicke Pfeile dargestellt sind. Die Pfeilrichtung entspricht der Transportrichtung der Paletten.

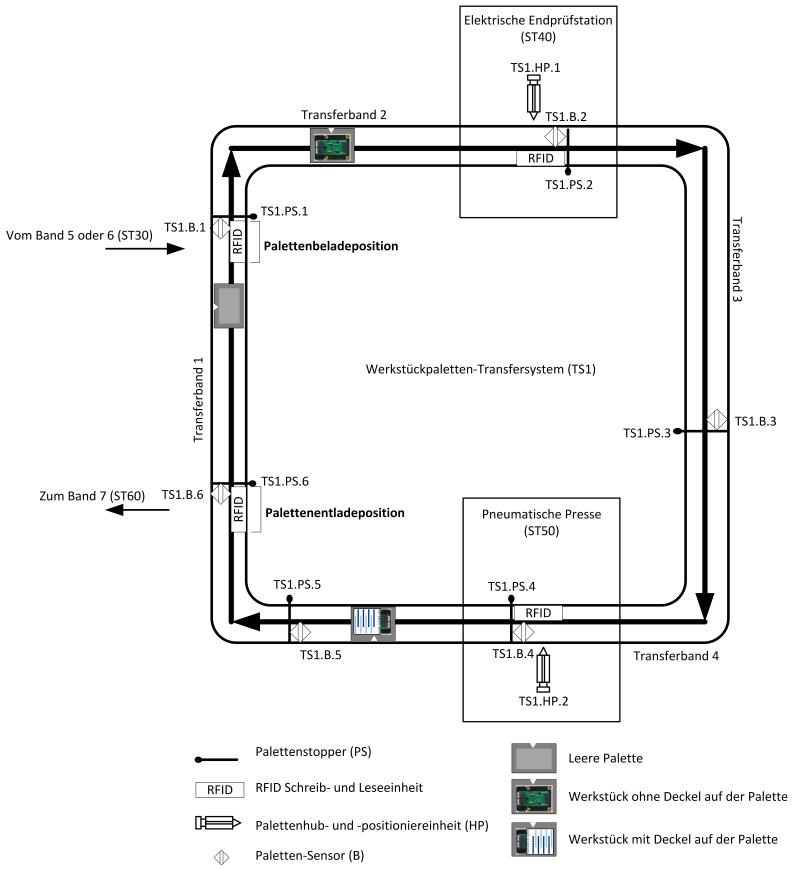


Abbildung 3.8.: Schematischer Aufbau des Werkstückpaletten-Transfersystems.

Die Transferbänder laufen permanent. Der Palettenfluss wird durch die pneumatisch angetriebenen Palettenstopper *PS.i* mit $i = [1, ..., 6]$ gesteuert. Im drucklosen Zustand ist er in Sperrstellung. An den Stoppern sind Sensoren *TS1.B.i* mit $i = [1, ..., 6]$ angebaut, die erkennen, ob eine Palette am Stopper liegt oder nicht. Des Weiteren befinden sich in den Arbeitsstationen 40 und 50 jeweils eine pneumatische Palettenhub- und -positioniereinheit, die die Palette etwas vom Band abhebt und auf der Prüfposition beziehungsweise Montageposition fixiert. Dadurch wird das Werkstück für die jeweilige Arbeitsstation in eine genau definierte Position gebracht.

Sämtliche Komponenten des Transfersystems sind an einem AS-i-Bus angeschlossen und werden von der SPS der Station 10 gesteuert. An den Paletten sind RFID-Tags montiert. An den relevanten Positionen des Transfersystems befinden sich RFID-Schreib- und Leseeinheiten. Die Kommunikation mit dem RFID-System wird ebenfalls von der SPS der Station 10 gesteuert.

3.3.7. Automatisierungstechnische Hardware

Die Architektur der automatisierungstechnischen Hardware ist in Abbildung 3.1 abstrakt dargestellt. Alle Stationen sind vom Automatisierungsgrad gleich aufgebaut und sind in einer Sterntopologie über SCALANCE X208 Switches in das echtzeitfähige Feldbus Profinet-IO-System eingebunden. Jede Station ist ausgestattet über eine eigene speicherprogrammierbare Steuerung vom Typ CPU S7-1512C-1PN der Siemens AG mit folgenden Eigenschaften [5]:

- Arbeitsspeicher: 1,25 MByte
- Datenspeicher für Bit/Wort-Merker: 1 MByte
- Programmspeicher: 250 kByte
- Maximale Anzahl der CPU-Bausteine (OB, FC, FB, DB): 2000
- Profinet-Schnittstelle

Jede SPS ist durch digitale und analoge Eingangs- und Ausgangsmodule ausgestattet. Die Steuerungen der Montage- und Prüfstationen sind in dem Profinet-IO-System als I-Devices (Slaves) konfiguriert und kommunizieren mit der SPS der Station 10, die in dem selben Profinet-IO-System als IO-Controller (Master) konfiguriert ist, und tauschen Daten aus. Die Kommunikation der Arbeitsstationen untereinander verläuft ebenso über den IO-Controller.

Die Hardware für das RFID-System wird komplett von Siemens bereit gestellt und besteht zum einen aus den RFID-Tags vom Typ MDSD-124 und zum anderen aus den vier

Schreib- und Leseeinheiten vom Typ Simatic RF240R. Jeweils zwei Schreib- und Leseeinheiten sind an einem RFID-Kommunikationsmodul vom Typ Simatic RF180C angeschlossen. Die Module sind in das Profinet-IO-System eingebunden.

Zusätzlich zu den in Abbildung 3.1 dargestellten Komponenten befinden sich im Profinet-IO-System noch weitere Komponenten, wie die HMI-Panels, der Vision-Sensor und der Siemens Umrichter Sinamics S120. Über die HMI-Panels sind die Stationen autark bedienbar. Die jeweilige Station kann zum Beispiel im Handbetrieb lokal gesteuert werden. Die Transferbänder werden durch den Sinamics-Umrichter gesteuert. Der Datenaustausch zwischen dem Profinet-IO-System und dem AS-i-Bus erfolgt über ein Profinet-Interfacemodul (AS-i-Master) der ET200SP.

4. Konzeption

In diesem Kapitel wird die Auswahl von Beschreibungsform, Entwicklungstool und Entwurfsansatz begründet.

4.1. Auswahl der Beschreibungsform

Für die Modellierung und Analyse logischer ereignisdiskreter Systeme und Steuerungssynthesen nach SCT sind im Besonderen netzartige Beschreibungsformen geeignet. In Unterkapitel 2.1.2 wurden deterministische endliche Automaten beziehungsweise Generatoren als mögliche netzartige Beschreibungsformen vorgestellt. Eine weitere mögliche netzartige Beschreibungsform sind die 1962 von Carl Adam Petri in seiner Dissertation vorgestellten Petri-Netze. Auch diese generieren Sprachen und können als ereignisdiskrete Beschreibungsform für die SCT verwendet werden. Die Frage, welche der beiden Beschreibungsformen für die jeweilige Anwendung am geeignetsten ist, hängt unter anderem davon ab, wie gut man bestimmte Abläufe aus den beiden Beschreibungsformen erkennen oder mit diesen beschreiben kann.

Die Petri-Netze eignen sich im Allgemeinen besonders gut für prozessorientierte Modellbildung. Mit ihnen lassen sich die Kopplungen zwischen den Teilprozessen, die nebenläufig ablaufen, durch parallele Pfade strukturell sichtbar darstellen. Die Generatoren dagegen basieren auf zustandsorientierter Modellbildung. Ein Zustand im Gesamtmodell modelliert einen kompletten Systemzustand. Nebenläufigkeit kann in Generatoren nicht strukturell sichtbar dargestellt werden. Bei sequenziellen Prozessen unterscheiden sich beide Beschreibungsformen nicht. Des Weiteren lässt sich das Verhalten eines Gesamtsystems aus vielen Komponenten mit Hilfe von Generatoren in strukturierter Weise nachbilden [13, S.213]. Auf der anderen Seite erfolgt der Steuerungsentwurf für Petri-Netze über S-Invarianten [7][14][2].

Für Generatoren existieren zur formalen Analyse und Steuerungssynthese nach SCT viele Methoden und Tools. In den meisten dieser Tools ist ein grafischer Editor zur Erstellung von Generatoren vorhanden und eine Supervisor-Reduzierung implementiert. Außerdem sind Generatoren verständlicher und leichter zu analysieren. Sie können gleichzeitig nicht mehr als in einem Zustand aktiv sein. Im Gegensatz zu Generatoren können in dem Petri-Netz eines Gesamtsystems viele Marken unterwegs sein. Zudem können die Stellen mit beliebig

vielen Marken belegt sein. Der aktuelle Zustand des Gesamtsystems kann dann in mehrere Teilzustände aufgeteilt sein, die sich teilweise unabhängig voneinander verändern können.

Des Weiteren setzt die Erreichbarkeitsanalyse der Petri-Netze als ersten Schritt die Berechnung des Erreichbarkeitsgraphen voraus. Diese Berechnung ist aber meistens nicht effizient durchführbar, weil man dann auf die Beschreibungsebene von Generatoren zurückkehren muss. Alternativ dazu gibt es algebraische Ansätze, die die Netzstruktur untersuchen und Analysen, wie Erreichbarkeit, Lebendigkeit oder Verklemmungsfreiheit, ermöglichen. An dieser Stelle wird auf weiterführende Literatur verwiesen (zum Beispiel [13] und [2]). Zur Steuerungssynthese nach SCT auf Basis von Petri-Netzen gibt es die Toolbox SPNBOX[8] für MathWorks MATLAB. Alternativen sind nicht bekannt. SPNBOX besitzt keinen grafischen Editor zur Erstellung der Petri-Netze und auch keinen integrierten Codegenerator. Die Netzmatrix muss hierbei entweder händisch oder mit Hilfe eines anderen Tools erstellt und danach in MATLAB eingetragen werden. Dies würde den Aufwand des Steuerungsentwurfs für komplexere Systeme deutlich erhöhen.

Es gibt für beide Beschreibungsformen diverse Tools, die die berechneten Modelle in einen Programmcode umsetzten. Auch in [6] wurde bereits ein solches Tool namens ACArrow entwickelt, das automatisch Modelle aus DESTool und SPNBOX direkt in einen Programmcode nach IEC 61131-3 oder Siemens-Syntax umsetzen kann. Der Code kann auch manuell erstellt werden, aber dieser Weg ist zu aufwendig und meistens fehleranfällig. Darüber hinaus sind in den Generatoren die Fehler leichter zu finden, da der Code trotz seiner Komplexität leicht verständlich ist.

Nach einer Abwägung von Vor- und Nachteilen der beiden Beschreibungsformen steht die Entscheidung für Generatoren als Beschreibungsform für die Anlage und Spezifikationen fest.

4.2. Auswahl des Entwicklungstools

Eine Modellbildung „von Hand“ für größere ereignisdiskrete Systeme sowie die darauf folgenden händischen Analysen und Synthesen sind meistens zu aufwendig oder sogar unmöglich. Daher sind für die erfolgreiche Anwendung der SCT und zur Beschleunigung der Synthese verfügbare und komfortable Tools essentiell.

Es gibt bereits zahlreiche akademische Tools, die auf Generatoren und formalen Sprachen basieren und zur ereignisdiskreten Steuerungssynthese nach SCT herangezogen werden können. Meist bekannt und auch frei erhältlich sind TCT [26], UMDES[1] Software Library mit

[1] `https://wiki.eecs.umich.edu/umdes/index.php/UMDES_Group`, Zugriffsdatum: 27.10.2017

DESUMA, Supremica [25] und DESTool [15] mit libFAUDES [16]. Alle dieser Tools haben ihre Eigenheiten und Vor- sowie Nachteile.

TCT wurde in der Arbeitsgruppe von W. M. Wonham an der Universität Toronto für den monolithischen Ansatz entwickelt und verfügt über viele effiziente Analyse- und Synthesefunktionen. Eine Supervisor-Reduzierung ist auch in diesem Tool implementiert. Darüber hinaus gibt es zahlreiche Dokumentationen und Publikationen, wie zum Beispiel [4], [3] und [28], in denen TCT Anwendung findet. Die einzigen Nachteile bei diesem Tool sind die Übersichtlichkeit und die Bedienbarkeit, aufgrund der textuellen Oberfläche.

Das Tool DESUMA[2] (Mount Allison University) ist als grafisches Frontend für UMDES Software Library entwickelt. Die in der University of Michigan entwickelte Bibliothek UMDES enthält eine Reihe C-Routinen zur ereignisdiskreten Modellbildung, Analyse und Supervisor-Synthese nach SCT. Das Tool Supremica bietet ein grafisches Frontend für die Erstellung von Modellen und besitzt viele effiziente Algorithmen, die dem aktuellen Stand der Technik entsprechen. Dieses Tool wird ständig weiter entwickelt und derzeit auch in einer Reihe von Forschungsprojekten eingesetzt. Supremica beinhaltet außerdem einen Codegenerator für verschiedene Programmiersprachen, darunter ANSI-C und VHDL. Die Bedienbarkeit dieses Tools ist aber im Gegensatz zu DESTool nicht so komfortable.

In dieser Arbeit wird zur Modellbildung, Analyse und Steuerungssynthese DESTool verwendet. Es wurde bereits in Unterkapitel 2.3.2 vorgestellt. Das Tool ist leicht zu bedienen und überzeugt neben Übersichtlichkeit durch zahlreiche Funktionen und gute Dokumentationen. Außerdem besitzt es ein Simulationsmodul, mit dem das Verhalten der gesteuerten Strecke mittels Simulation analysiert werden kann. In der eingebundenen Bibliothek libFAUDES ist auch eine Funktion zur Supervisor-Reduzierung vorhanden.

4.3. Auswahl des Entwurfsansatzes

Im Folgenden werden der monolithische Ansatz, der modulare Ansatz, der dezentrale Ansatz und der lokal-modulare Ansatz für den Entwurf einer Puffersteuerung für die in dieser Arbeit betrachtete Montageanlage gegenübergestellt und verglichen. Die Entwurfsansätze wurden in Unterkapitel 2.2.2 vorgestellt und beschrieben. Die algorithmische Komplexität spielt eine wichtige Rolle bei der Umsetzung dieser Entwurfsansätze. Unter algorithmischer Komplexität ist in dieser Arbeit die Rechenzeit und der Speicherplatz zu verstehen, die für die Modellbildung, Modellanalyse und Supervisor-Synthese notwendig sind. Die genaue Berechnung der algorithmischen Komplexität zeigt [30].

[2] `https://www.mta.ca/Community/Research_and_creative/GIDDES/DESUMA_info/DESUMA_info/`, Zugriffsdatum: 27.10.2017

Der **monolithische Ansatz** benötigt ein unstrukturiertes Gesamtmodell der Strecke und eine Gesamtspezifikation für die globale Steuerungsaufgabe. Eine Gesamtspezifikation kann auch durch Komposition mehrerer Teilspezifikationen erstellt werden. Dieser Ansatz ist für komplexere Systeme nicht flexibel. Deshalb wird bei dem **modularen Ansatz** die globale Steuerungsaufgabe beziehungsweise die Gesamtspezifikation in mehrere modulare Spezifikationen aufgeteilt. Dadurch weisen die modulare Supervisor meistens eine geringere Modellkomplexität auf als ein monolithischer Supervisor. Jedoch wird bei diesem Ansatz zur Überprüfung der Steuerbarkeit und des Nichtblockierens weiterhin ein unstrukturiertes Gesamtmodell der Strecke benötigt. Der **dezentrale Ansatz** ist eine Erweiterung des modularen Ansatzes und ist grundlegend für eine verteilte Implementierung, wenn zum Beispiel durch räumliche Trennung der Steuerungshardware nicht alle Streckenereignisse global für jeden Supervisor verfügbar stehen. Bei dem **lokal-modularen Ansatz** kann durch die Modularisierung der Strecke und der globalen Steuerungsaufgabe die algorithmische Komplexität im Vergleich zu den anderen Ansätzen weiter reduziert werden. Dieser Ansatz zeigt besonders seine Vorteile, wenn die Strecke viele asynchrone Prozesse enthält.

Ein Vergleich zwischen den Komplexitäten dieser Ansätze zeigt, dass der monolithische Ansatz und der dezentrale Ansatz für die Montageanlage nicht geeignet sind. Bei dem monolithischen Ansatz entsteht durch Zusammenwirken vieler Komponenten eine große Modellkomplexität, die nicht mit vertretbarem Zeit- und Speicheraufwand lösbar ist. Außerdem weist der monolithische Supervisor eine hohe Modellkomplexität auf, sodass eine Supervisor-Synthese unmöglich ist. Durch den dezentralen Ansatz kann hier nicht eine noch geringere algorithmische Komplexität als beim modularen Ansatz erzielt werden, weil alle Steuerungshardware an der Anlage untereinander vernetzt sind, sodass alle Streckenereignisse global für jeden Supervisor zur Verfügung stehen. In einer ersten Einschätzung sind der modulare Ansatz und der lokal-modulare Ansatz geeignet. Auch die verwendete CPU S7-1512C hat ausreichend RAM-Speicher, um jeden der beiden Ansätze zu realisieren.

Die Strecke beinhaltet 16 inhärent asynchrone Komponenten. Das Kompositionssystem aus allen Teilmodellen des Gesamtsystems ergibt einen Generator mit 2073600 Zuständen und 47416320 Transitionen. Die Berechnung wurde mit dem SYPC-Operator von DESTool in zwei Schritten durchgeführt. Die Rechenzeit betrug auf einem Standardrechner mit einem Intel i5 2,4 GHz-Prozessor und 8 GB RAM-Hauptspeicher etwa vier Minuten. Diese Rechenzeit zeigt, dass die Modellkomplexität des Gesamtsystems unter DESTool handhabbar ist. Jedoch würden sich für die Überprüfung der Steuerbarkeit unpraktikabel lange Rechenzeiten ergeben, weil jede Spezifikation bezüglich des Gesamtmodells der Strecke überprüft werden soll.

Der Vergleich des modularen Ansatzes mit dem lokal-modularen zeigt, dass der lokal-modulare Ansatz für die in dieser Arbeit betrachtete Montageanlage am geeignetsten ist. Die Relationen zwischen den Ereignisalphabeten der Strecke und der Spezifikationen zeigt die Abbildung 6.18. Alle Komponenten (vgl. Tabelle 5.8) werden von den 14 Spezifikationen

eingeschränkt. Dadurch ergibt sich für das eingeschränkte System G_e wieder das Kompositionssystem. Aber es gibt keine Spezifikation, die alle Streckenkomponenten einschränkt, sodass als lokales System wieder das Kompositionssystem aus allen Komponenten berechnet werden muss. In dieser Arbeit enthält das größte berechnete lokale System nur 225 Zustände und 1320 Transitionen (G_{X3} und G_{X4}, vgl. Tabelle 6.1). Daher weist dieser Ansatz für die Montageanlage eine viel geringere Komplexität auf als der modulare Ansatz.

5. Modellierung der ungesteuerten Strecke

Die formale Beschreibung der Strecke und der Spezifikationen bilden in dieser Arbeit die Grundlage für die Methoden der Steuerungssynthese nach SCT. Im ersten Teil dieses Kapitels werden Annahmen zur Modellbildung und zum Steuerungsentwurf getroffen. Im zweiten Teil wird das Verhalten der Streckenkomponenten modelliert.

5.1. Annahmen zur Modellbildung und zum Steuerungsentwurf

Für die Modellbildung und den Steuerungsentwurf wurden folgende Annahmen getroffen:

- Es wird nur das logische Verhalten der Komponenten modelliert. Die Zeitpunkte, in der die Ereignisse auftreten, werden bei der Modellbildung in dieser Arbeit nicht berücksichtigt.
- Es wird davon ausgegangen, dass alle Streckenkomponenten fehlerfrei arbeiten und nicht ausfallen.
- Sobald die Anlage eingeschaltet wird, befinden sich sämtliche Komponenten in ihrem definierten Initialzustand. Daraus ergeben sich die Initialzustände der Modelle.
- Die Kapazitäten des Ein- und Ausgangslagers sowie der Relaiskartenvorrat in Station 30 werden als unendlich angenommen. Der Deckelvorrat in Station 50 wird ebenfalls als unendlich angenommen.
- Die Werkstücke werden dem Eingangslager manuell zugeführt und die fertigen Werkstücke vom Ausgangslager manuell abtransportiert.
- Die Steuerung der Bänder wird durch unterlagerte Steuerungen übernommen und muss nicht durch die Puffersteuerung gestartet und gestoppt werden.

- Die vier Transferbänder an dem Transfersystem laufen permanent und können weder durch die unterlagerten Steuerungen noch durch die Puffersteuerung beeinflusst werden. Sie werden ebenfalls durch unterlagerte Steuerungen gesteuert.
- Die Rückmeldungen der Komponenten werden aus den unterlagerten Schrittkettensteuerungen generiert. Alle nicht steuerbaren Ereignisse werden bei steigenden Flanken erzeugt.

5.2. Komponentenmodelle

Es soll eine ereignisdiskrete Puffersteuerung für die in Kapitel 3 vorgestellte Montageanlage entworfen und implementiert werden. Hierfür wird die Anlage zuerst auf die zur Lösung dieser Steuerungsaufgabe notwendigen Komponenten abstrahiert. Es sollen also nicht alle Streckenkomponenten modelliert werden, sondern nur die für die Lösung der Steuerungsaufgabe relevanten Komponenten. Zudem ist eine abstrakte Betrachtung dieser Komponenten ausreichend.

5.2.1. Modellierung der Handlingeinheit mit der Linearachse *LIHNDL*

Die Handlingeinheit mit der Linearachse (*LIHNDL*) steuert den Werkstückfluss von Station 20 zu Station 30 und von Station 60 zu Station 20. Ihre Steuerung wird von einer Linearachsensteuerung und einer Schrittkette übernommen. Diese Schrittkette besteht neben anderen aus einer Aneinanderreihung von folgenden Prozessen:

- Linearachse zum Band 1, 2, 3, 4, 5, 6, 7 oder zur Grundstellung fahren,
- Drehachse zur Station 20 oder zum Transfersystem hin,
- Vertikale Achse heben oder senken,
- Greifer öffnen oder schließen.

Die Handlingeinheit mit der Linearachse kann vier Aufträge ausführen. Somit ergeben sich für ihr Verhalten fünf Zustände:

0. Initialzustand: Die Handlingeinheit und Linearachse befinden sich in ihrer Grundstellung,
1. Die Handlingeinheit bringt ein Werkstück vom Band 1 (ST20) zum Band 5 (ST30),
2. Die Handlingeinheit bringt ein Werkstück vom Band 2 (ST20) zum Band 6 (ST30),

3. Die Handlingeinheit bringt ein Werkstück vom Band 7 (ST60) zum Band 3 (ST20),
4. Die Handlingeinheit bringt ein Werkstück vom Band 7 (ST60) zum Band 4 (ST20).

Abbildung 5.1 bildet ihr ungesteuertes Verhalten als Generator nach.

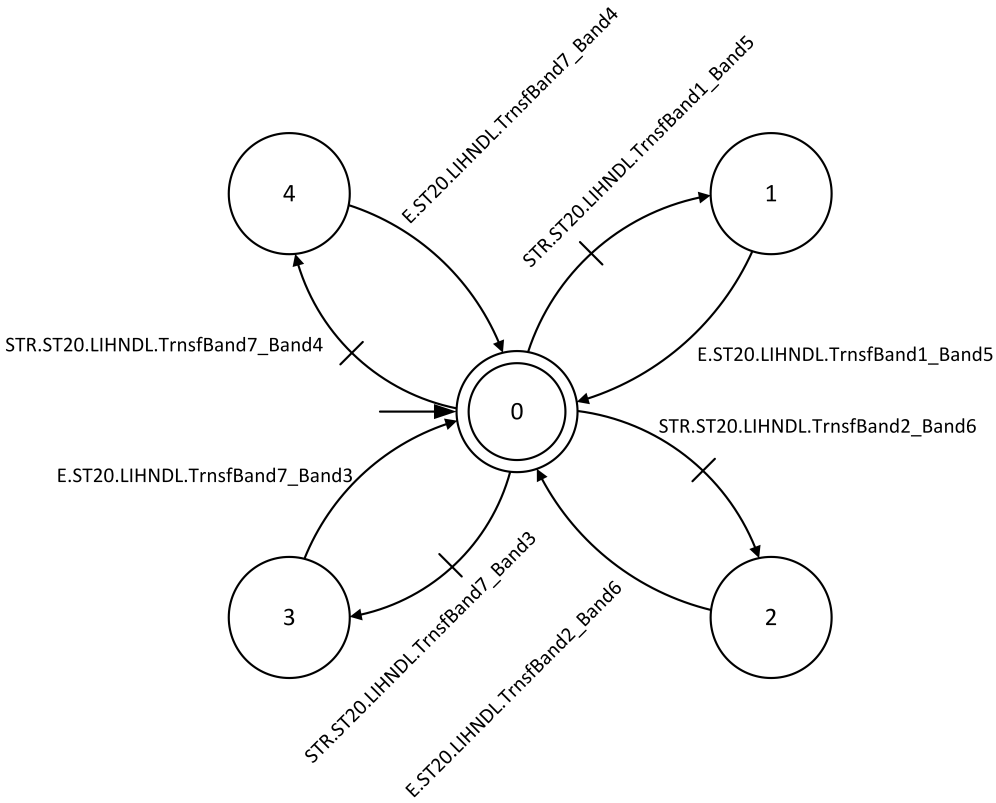


Abbildung 5.1.: Generatormodell für die Handlingeinheit mit Linearachse *LIHNDL* in Station 20.

Durch die steuerbaren Ereignisse wird die Schrittkette aktiviert und ein Auftrag gestartet. Nach Abarbeitung des Auftrags wird aus der Schrittkette die Rückmeldung *E.ST20.LIHNDL.TrnsfBandi_Bandj* (von Band i zu Band j) erzeugt. Ein neuer Auftrag kann nur gestartet werden, wenn die Handlingeinheit und Linearachse in ihre Grundstellungen zurückgekehrt sind. Die Bedeutungen der Ereignisse sind in Tabelle 5.1 zusammengefasst.

Ereignis	Beschreibung
STR.ST20.LIHNLD.TrnsfBand1_Band5	Starte Auftrag 1: Bringe Werkstück von Band 1 (ST20) zu Band 5 (ST30)
E.ST20.LIHNLD.TrnsfBand1_Band5	Ende des Auftrags 1
STR.ST20.LIHNLD.TrnsfBand2_Band6	Starte Auftrag 2: Bringe Werkstück von Band 2 (ST20) zu Band 6 (ST30)
E.ST20.LIHNLD.TrnsfBand2_Band6	Ende des Auftrags 2
STR.ST20.LIHNLD.TrnsfBand7_Band3	Starte Auftrag 3: Bringe Werkstück von Band 7 (ST60) zu Band 3 (ST20)
E.ST20.LIHNLD.TrnsfBand7_Band3	Ende des Auftrags 3
STR.ST20.LIHNLD.TrnsfBand7_Band4	Starte Auftrag 4: Bringe Werkstück von Band 7 (ST60) zu Band 4 (ST20)
E.ST20.LIHNLD.TrnsfBand7_Band4	Ende des Auftrags 4

Tabelle 5.1.: Ereignisdefinitionen für die Handlingeinheit und die Linearachse in Station 20.

5.2.2. Modellierung der Robotereinheit *ROB*

Das Verhalten des Roboters *ROB* wird ausschließlich durch das im Roboter-Controller implementierte Programm definiert. Es können vier Aufträge ausgewählt werden. Der gewünschte Auftrag wird über die digitalen Ausgänge der SPS der Station 30 ausgewählt. Nach Abarbeitung des gewählten Auftrags im Roboter-Controller wird dieser der SPS mittels Signal gemeldet. Danach wird aus der Schrittkette der Station die jeweilige Rückmeldung erzeugt. Des Weiteren muss das Roboterprogramm den aktuellen Auftrag abgearbeitet haben, bevor ein neuer Auftrag gestartet werden kann. Mehrere Aufträge können nicht gleichzeitig abgearbeitet werden. Abbildung 5.2 zeigt den Generator, der das ungesteuerte Verhalten des Roboters repräsentiert.

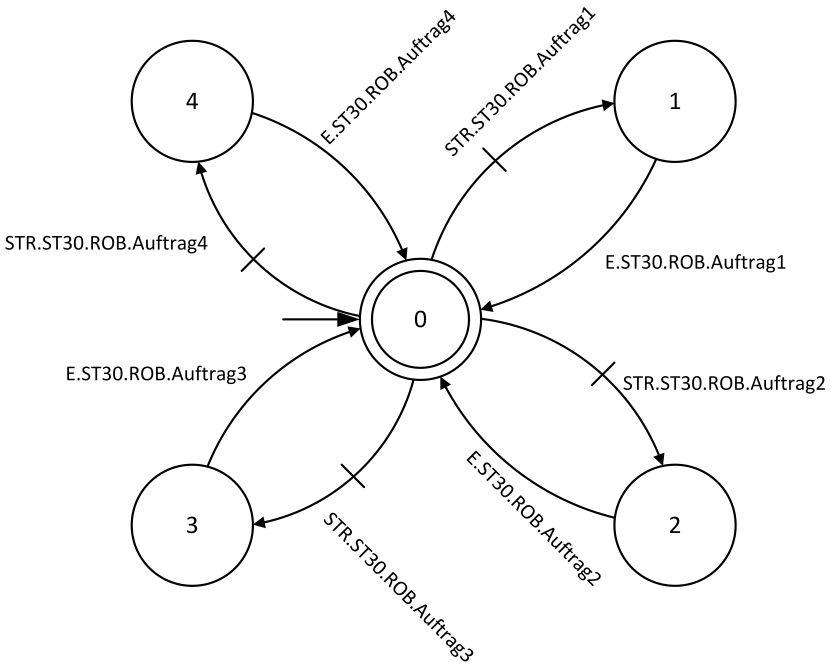


Abbildung 5.2.: Generatormodell des Roboters.

Die Bedeutungen der Ereignisse sind in Tabelle 5.2 beschrieben.

Ereignis	**Beschreibung**
STR.ST30.ROB.Auftrag1	Einpolige Relaiskarte einsetzen
E.ST30.ROB.Auftrag1	Ende des Auftrags 1
STR.ST30.ROB.Auftrag2	Zweipolige Relaiskarte einsetzen
E.ST30.ROB.Auftrag2	Ende des Auftrags 2
STR.ST30.ROB.Auftrag3	Relaiskarte entfernen, Werkstückunterteil vom Band 7 zum Band 5 bringen und anschließend eine einpolige Relaiskarte auf dem Werkstückunterteil einsetzen,
E.ST30.ROB.Auftrag3	Ende des Auftrags 3
STR.ST30.ROB.Auftrag4	Relaiskarte entfernen, Werkstückunterteil vom Band 7 zum Band 6 bringen und anschließend eine zweipolige Relaiskarte auf dem Werkstückunterteil einsetzen,
E.ST30.ROB.Auftrag4	Ende des Auftrags 4

Tabelle 5.2.: Ereignisdefinitionen für den Roboter.

5.2.3. Modellierung der Handlingeinheit *HNDL1*

Die Handlingeinheit *HNDL1* steuert den Werkstückfluss von Station 30 zum Transfersystem. Sie besteht aus mehreren Komponenten. Ihr Verhalten wird durch eine Schrittkettensteuerung definiert, die alle ihre Komponenten umfasst. Das Verhalten lässt sich in drei Zustände zusammenfassen:

0. Initialzustand: Die Handlingeinheit befindet sich in der Grundstellung,
1. Die Handlingeinheit belädt die Palette mit einem einpoligen Werkstück,
2. Die Handlingeinheit belädt die Palette mit einem zweipoligen Werkstück.

Der Generator in Abbildung 5.3 bildet ihr ungesteuertes Verhalten nach.

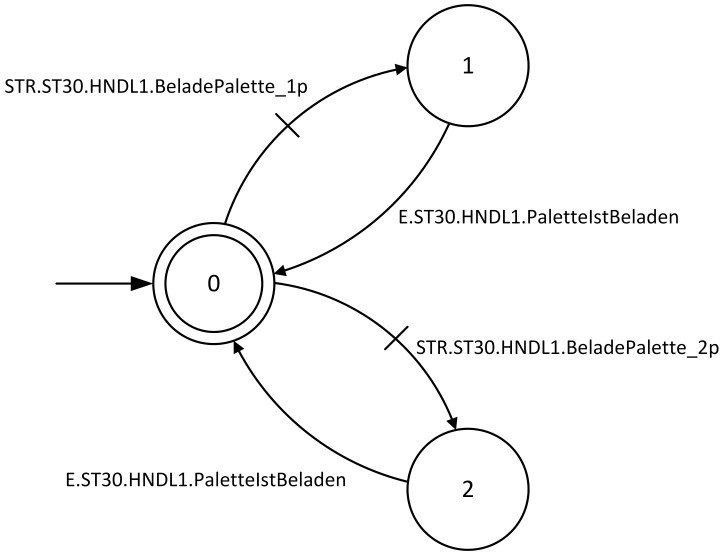


Abbildung 5.3.: Generatormodell für die Handlingeinheit in Station 30.

Durch das Ereignis *STR.ST30.HNLD1.Beladepalette_ip* mit $i = [1, 2]$ kann ein Auftrag gestartet werden. Erst wenn sich die Handlingeinheit wieder in der Grundstellung befindet, wird aus der unterlagerten Schrittkettensteuerung das Ereignis *E.ST30.HNDL1.PaletteIstBeladen* generiert. Die Bedeutungen der Ereignisse sind in Tabelle 5.3 zusammengefasst.

Ereignis	Beschreibung
STR.ST30.HNLD1.BeladePalette_1p	Starte Auftrag 1: Werkstück vom Band 5 entnehmen und auf einer Palette platzieren (Werkstück ist einpolig)
STR.ST30.HNDL1.BeladePalette_2p	Starte Auftrag 2: Werkstück vom Band 6 entnehmen und auf einer Palette platzieren (Werkstück ist zweipolig)
E.ST30.HNDL1.PaletteIstBeladen	Ende des Auftrags 1 oder 2

Tabelle 5.3.: Ereignisdefinitionen für die Handlingeinheit in Station 30.

5.2.4. Modellierung des Werkstückpaletten-Stopper *PS*

Der Palettenfluss auf dem Transfersystem wird durch die Palettenstopper beeinflusst. Da sie den gleichen Steuerungsablauf beinhalten ist für alle Stopper nur ein generisches Modell zu entwerfen, das für unterschiedliche Instanzen verwendet wird. Das generische Modell des Stoppers ist durch den Generator in Abbildung 5.4 nachgebildet.

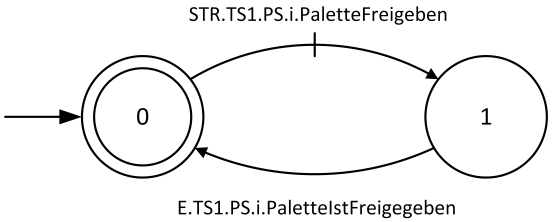


Abbildung 5.4.: Generisches Modell des Palettenstoppers.

Auf dem Transfersystem sind sechs Instanzen dieser Komponente mit den Ereignissen *STR.TS1.PS.i.PaletteFreigeben* und *E.TS1.PS.i.PaletteIstFreigegeben* mit $i = [1, \ldots, 6]$ vorhanden. Durch das steuerbare Ereignis *STR.TS1.PS.i.PaletteFreigeben* wird der Stopper *PS.i* entriegelt. Befindet sich danach eine Palette im Ansprechbereich des Palettensensors *TS1.B.i* wird der Stopper gelöst und die Palette kann passieren. Das Ereignis *E.TS1.PS.i.PaletteIstFreigegeben* wird generiert, nachdem die Palette den Sensor *TS1.B.i* passiert hat. Daraufhin wird der Initialzustand wieder erreicht und der Stopper *PS.i* wieder verriegelt.

Das Modell der Komponente *PS.1* wurde zusätzlich um einen weiteren Zustand erweitert (Der blaue Zustand, vgl. in Abbildung 5.5). Die Einführung dieses Zustandes begründet sich

folgendermaßen: Durch das Ereignis *STR.TS1.PS.1.PaletteFreigeben* wird vor einer Freigabe der Palette der entsprechende RFID-Tag beschrieben. Dieses Ereignis hat jedoch keinen Einfluss auf den zu schreibenden Wert. In der gesteuerten Strecke sollen die Stopper auch leere Paletten freigeben können. Hierfür soll vor einer Freigabe vom *PS.1* auf den RFID-Tag der entsprechenden Palette der Wert 0 geschrieben werden. Dies wird durch das Ereignis *STR.TS1.PS.1.LeerePaletteFreigeben* erzwungen. Die Schrittketten der Stationen 40, 50 und 60 wurden dementsprechend angepasst. Sie können leere Paletten erkennen und sofort wieder freigeben.

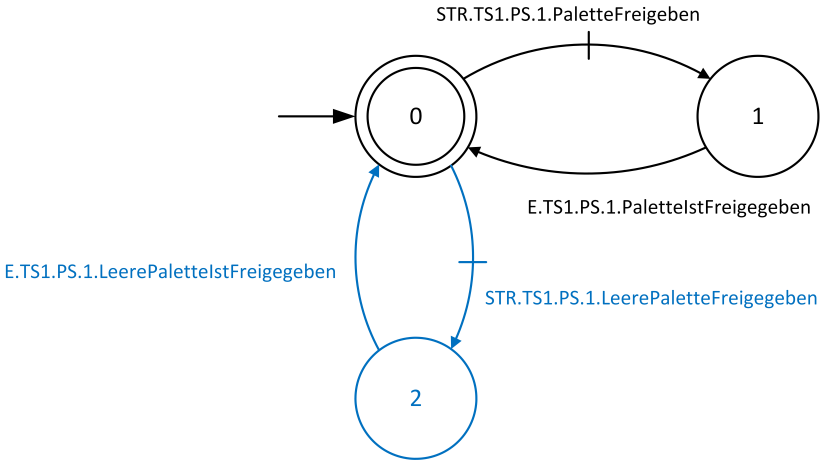


Abbildung 5.5.: Erweiterte Modell des Palettenstoppers *PS.1*.

5.2.5. Modellierung der Palettenhub- und -positioniereinheit *HP*

Das Aus- und Einfahren der pneumatischen Hub- und -positioniereinheit wird durch unterlagerte Steuerungen gesteuert und ist hier für die Puffersteuerung nicht zugänglich. An der Palettenhub- und -positioniereinheit ist ein Endschalter angeschlossen, über den die Steuerung registriert, ob sie eingefahren oder ausgefahren ist. Ihre Grundstellung ist der eingefahrene Zustand. Wenn sie eingefahren ist und die Endlage erreicht hat, wird der Endschalter aktiviert und ansonsten deaktiviert. Die Puffersteuerung benötigt hier nur die Information, ob die Palettenhub- und -positioniereinheit eingefahren oder ausgefahren ist. Deshalb wird in dieser Arbeit nur ihr Endschalter modelliert. Es befinden sich auf dem Transfersystem zwei Instanzen dieser Komponente mit den Ereignissen *E.TS1.HP.i.ausgefahren* und *E.TS1.HP.i.eingefahren* mit $i = [1, 2]$. Ihr generisches Modell ist in Abbildung 5.6 als Generator dargestellt.

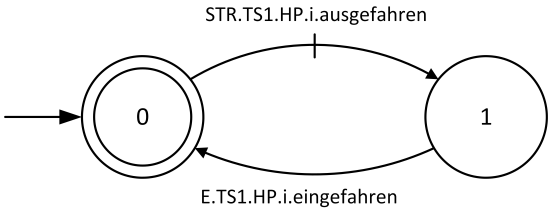


Abbildung 5.6.: Generisches Modell der Palettenhub- und -positioniereinheit.

5.2.6. Modellierung der Handlingeinheit *HNDL2*

Der Werkstückfluss vom Transfersystem zur Station 60 wird durch die Handlingeinheit *HNDL2* gesteuert. Die Steuerung wird von einer Schrittkette übernommen. Das Verhalten der Handlingeinheit lässt sich in zwei Zustände zusammenfassen:

0. Initialzustand: Die Handlingeinheit befindet sich in der Grundstellung,
1. Die Handlingeinheit entlädt die Palette.

Abbildung 5.7 zeigt den Generator, der das ungesteuerte Verhalten der Handlingeinheit nachbildet.

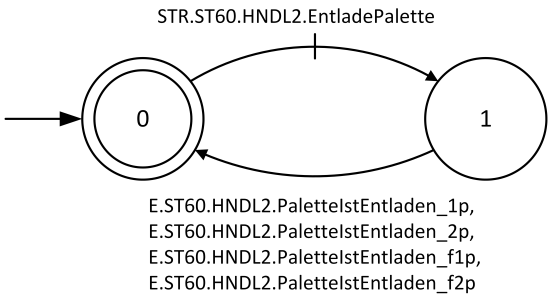


Abbildung 5.7.: Generatormodell für die Handlingeinheit *HNDL2* in Station 60.

Durch das Ereignis *STR.ST60.HNDL2.EntladePalette* wird ein Auftrag gestartet. Am Ende der Auftragsbearbeitung, wenn sich die Handlingeinheit wieder in der Grundstellung befindet, wird durch das Auslesen des RFID-Tags eine entsprechende Rückmeldung generiert. Dadurch erfolgt der Zustandswechsel von Zustand 1 nach Initialzustand 0. Die Bedeutungen der Ereignisse sind in Tabelle 5.4 zusammengefasst.

Ereignis	Beschreibung
STR.ST60.HNDL2.EntladePalette	Entlade Palette
E.ST60.HNDL2.PaletteIstEntladen_1p	Das von der Palette entnommene Werkstück ist einpolig (RFID-Wert 11).
E.ST60.HNDL2.PaletteIstEntladen_2p	Das von der Palette entnommene Werkstück ist zweipolig (RFID-Wert 12).
E.ST60.HNDL2.PaletteIstEntladen_f1p	Das von der Palette entnommene Werkstück ist einpolig und defekt (RFID-Wert 241).
E.ST60.HNDL2.PaletteIstEntladen_f2p	Das von der Palette entnommene Werkstück ist zweipolig und defekt (RFID-Wert 242).
E.ST60.HNDL2.KTeil	Palette ist leer (RFID-Wert 0).

Tabelle 5.4.: Ereignisdefinitionen für die Handlingeinheit *HNDL2* in Station 60.

5.2.7. Modellierung des Abschiebers *AS*

In Station 60 befinden sich zwei Instanzen dieser Komponente (*AS.1*, *AS.2*), wobei die Instanz *AS.1* für diese Arbeit nicht betrachtet wird, da die defekten Werkstücke (reparabel) nicht vom Band ausgeschleust, sondern demontiert und als Mitkopplung wieder in den Montageprozess eingelastet werden. Abbildung 5.8 bildet den zweiten Abschieber als Generator nach.

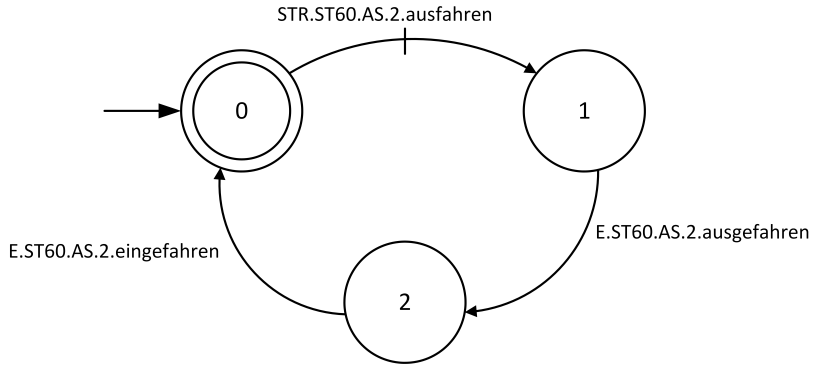


Abbildung 5.8.: Generatormodell für den Abschieber *AS.2*.

Die Grundstellung des Abschiebers *AS.2* ist der eingefahrene Zustand. Er wird durch das steuerbare Ereignis *STR.ST60.AS.2.ausfahren* ausgefahren und ein Werkstück vom Band

ausgeschleust. Das Setzen des Ausgangssignals für das Ausfahren des Abschiebers erfolgt im Zustand 1 durch eine unterlagerte Steuerung. Ist der Abschieber ausgefahren beziehungsweise seine Endposition erreicht, so wird das Ereignis *E.ST60.AS.2.ausgefahren* generiert. Dadurch findet ein Zustandswechsel zwischen 1 und 2 statt. Danach wird das Ausgangssignal zurückgesetzt, sodass der Abschieber wieder zurück fährt. Befindet sich der Abschieber danach in seiner Grundstellung, wird das Ereignis *E.ST60.AS.2.eingefahren* generiert. Die Bedeutungen der Ereignisse sind in Tabelle 5.5 zusammengefasst.

Ereignis	**Beschreibung**
STR.ST60.AS.2.ausfahren	Abschieber ausfahren
E.ST60.AS.2.ausgefahren	Abschieber ist ausgefahren
E.ST60.AS.2.eingefahren	Abschieber ist eingefahren

Tabelle 5.5.: Ereignisdefinitionen für den Abschieber *AS.2*.

5.2.8. Modellierung des Vision-Sensors *VS*

Der Vision-Sensor nimmt Bilder auf, wertet sie aus und sendet anschließend das Prüfergebnis an die übergeordnete SPS der Station 60. Das Prüfprogramm wird auf dem Controller des Vision-Sensors ausgeführt. Es wird aus der Schrittkette der Station aufgerufen und im Anschluss der Prüfung wird aus dem Prüfprogramm eine Rückmeldung („In Ordnung“ oder „Nicht in Ordnung“) an die Station gesendet. Der Generator in Abbildung 5.9 stellt das ungesteuerte Verhalten des Vision-Sensors dar.

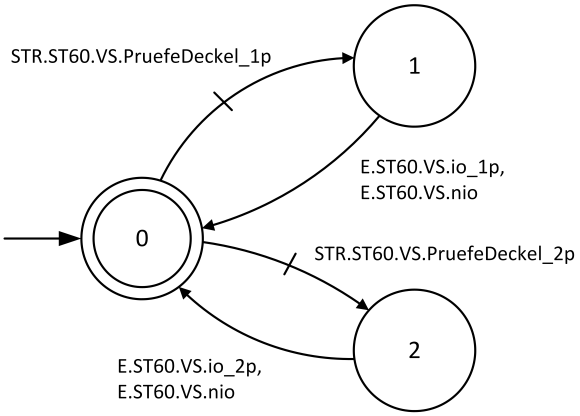


Abbildung 5.9.: Generatormodell für den Vision-Sensor.

Durch das steuerbare Ereignis *STR.ST60.VS.PruefeDeckel_1p* oder *STR.ST60.VS.PruefeDeckel_2p* wird die Abarbeitung des Prüfprogramms ausgelöst. Bei einem Fehlerergebnis wird aus der

Schrittkettensteuerung das Ereignis *E.ST60.VS.nio* generiert. Andernfalls wird das Ereignis *E.ST60.VS.io_1p* beziehungsweise *E.ST60.VS.io_2p* generiert. Die Bedeutungen der Ereignisse sind in Tabelle 5.6 zusammengefasst.

Ereignis	**Beschreibung**
STR.ST60.VS.PruefeDeckel_1p	Triggerereignis
E.ST60.VS.io_1p	Ergebnis der optischen Prüfung ist in Ordnung
STR.ST60.VS.PruefeDeckel_2p	Triggerereignis
E.ST60.VS.io_2p	Ergebnis der optischen Prüfung ist in Ordnung
E.ST60.VS.nio	Ergebnis Fehlteil

Tabelle 5.6.: Ereignisdefinitionen für den Vision-Sensor.

5.2.9. Modellierung der Sensorik

Die Präsenz eines unbearbeiteten Werkstücks in Station 20 kann durch den Sensor *ST20.B.1* und *ST20.B.2* an der Entnahme-Position von Band 1 und 2 festgestellt werden. Sie lassen sich nach dem generischen Generatormodell in Abbildung 5.10 nachbilden. Das Sensorereignis *B.on* wird bei einer positiven Flanke des Sensorsignals für einen Zyklus ausgelöst. Durch Negation des Sensorsignals lässt sich das Ereignis *B.off* erzeugen. Es ergeben sich somit zwei Zustände, die der Sensor einnehmen kann.

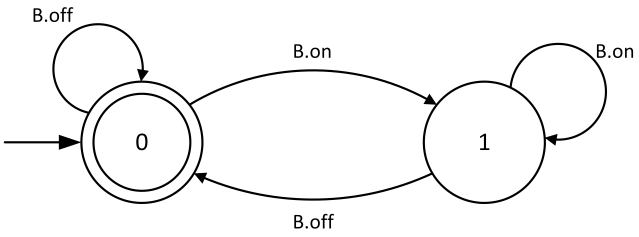


Abbildung 5.10.: Generisches Sensormodell.

Generatorname	**Komponente**	***B.on***	***B.off***
G2	ST20.B.1	E.ST20.B.1.on	E.ST20.B.1.off
G3	ST20.B.2	E.ST20.B.2.on	E.ST20.B.2.off

Tabelle 5.7.: Instanzen der Sensoren nach dem generischen Modell in Abbildung 5.10.

Die anderen Sensoren werden durch die unterlagerten Steuerungen ausgewertet. Sie werden bei einer späteren Supervisor-Synthese nicht benötigt.

5.3. Produktsystem und Ereignisalphabet der Strecke

Die ungesteuerte Strecke enthält insgesamt 16 Komponenten. Sie sind über disjunkte Ereignisalphabete definiert und laufen vollständig asynchron zueinander („Shuffle“). Somit liegt die Strecke als Produktsystem vor. Das feinste Produktsystem ist:

$$G_i = G_i', i = [1, ..., 16]$$

Die Tabelle 5.8 fasst alle Komponenten der Strecke mit deren Anzahl an Zuständen *n* und Transitionen *t* zusammen.

Name	**Komponente**	**n**	**t**	**Name**	**Komponente**	***n***	***t***
G1	ST20.LIHNDL	5	8	G9	TS1.PS.4	2	2
G2	ST20.B.1	2	4	G10	TS1.PS.5	2	2
G3	ST20.B.2	2	4	G11	TS1.PS.6	2	2
G4	ST30.HNDL1	3	4	G12	TS1.HP.1	2	2
G5	ST30.ROB	5	8	G13	TS1.HP.2	2	2
G6	TS1.PS.1	3	4	G14	ST60.AS.2	3	3
G7	TS1.PS.2	2	2	G15	ST60.HNDL2	2	6
G8	TS1.PS.3	2	2	G16	ST60.VS	3	6

Tabelle 5.8.: Alle Komponenten der Strecke mit deren Anzahl an Zuständen n und Transitionen t.

Die Tabelle 5.9 fasst alle Ereignisse zusammen, die in der Strecke vorkommen.

Ereignis	Ereignis
STR.ST20.LIHNDL.TrnsfBand1_Band5	E.TS1.PS.4.PaletteIstFreigegeben
E.ST20.LIHNDL.TrnsfBand1_Band5	STR.TS1.PS.5.PaletteFreigeben
STR.ST20.LIHNDL.TrnsfBand2_Band6	E.TS1.PS.5.PaletteIstFreigegeben
E.ST20.LIHNDL.TrnsfBand2_Band6	STR.TS1.PS.6.PaletteFreigeben
STR.ST20.LIHNDL.TrnsfBand7_Band3	E.TS1.PS.6.PaletteIstFreigegeben
E.ST20.LIHNDL.TrnsfBand7_Band3	E.TS1.HP.1.ausgefahren
STR.ST20.LIHNDL.TrnsfBand7_Band4	E.TS1.HP.1.eingefahren
E.ST20.LIHNDL.TrnsfBand7_Band4	E.TS1.HP.2.ausgefahren
STR.ST30.HNDL1.BeladePalette_1p	E.TS1.HP.2.eingefahren
STR.ST30.HNDL1.BeladePalette_2p	STR.ST60.AS.2.ausfahren
STR.ST30.HNDL1.PaletteIstBeladen	E.ST60.AS.2.ausgefahren
STR.ST30.ROB.Auftrag1	E.ST60.AS.2.eingefahren
E.ST30.ROB.Auftrag1	STR.ST60.HNDL2.EntladePalette
STR.ST30.ROB.Auftrag2	E.ST60.HNDL2.PaletteIstEntladen_1p
E.ST30.ROB.Auftrag2	E.ST60.HNDL2.PaletteIstEntladen_2p
STR.ST30.ROB.Auftrag3	E.ST60.HNDL2.PaletteIstEntladen_f1p
E.ST30.ROB.Auftrag3	E.ST60.HNDL2.PaletteIstEntladen_f2p
STR.ST30.ROB.Auftrag4	E.ST60.HNDL2.KTeil
E.ST30.ROB.Auftrag4	STR.ST60.VS.PruefeDeckel_1p
STR.TS1.PS.1.PaletteFreigeben	STR.ST60.VS.PruefeDeckel_2p
E.TS1.PS.1.PaletteIstFreigegeben	E.ST60.VS.io_1p
STR.TS1.PS.2.PaletteFreigeben	E.ST60.VS.io_2p
E.TS1.PS.2.PaletteIstFreigegeben	E.ST60.VS.nio
STR.TS1.PS.3.PaletteFreigeben	STR.TS1.PS.1.LeerePaletteFreigeben
E.TS1.PS.3.PaletteIstFreigegeben	E.TS1.PS.1.LeerePaletteIstFreigegeben
STR.TS1.PS.4.PaletteFreigeben	

Tabelle 5.9.: Ereignisalphabet Σ_G der Strecke.

6. Steuerungsentwurf

Nachfolgendes Kapitel untergliedert sich in drei Teile. Zunächst wird die Steuerungsaufgabe textuell definiert. Danach werden Spezifikationen formuliert. Anschließend werden in Unterkapitel 6.3 aus den in Kapitel 5 entwickelten Modellen und den formalen Spezifikationen aus dem vorliegenden Kapitel die Supervisor synthetisiert.

6.1. Definition der Steuerungsaufgabe

Es soll für die in Kapitel 3 vorgestellte Anlage eine ereignisdiskrete Puffersteuerung entwickelt werden. Hierzu sind vorerst Aussagen über die Pufferplätze zwischen den einzelnen Stationen beziehungsweise Prozessen zu treffen und die jeweiligen Pufferkapazitäten zu definieren. Die Puffersteuerung soll grundsätzlich die einzelnen Pufferbestände überwachen und steuern. Das Blockieren der Prozesse sowie der gesamten Anlage ist unbedingt zu vermeiden.

Die defekten Werkstücke (reparable) sollen mit Hilfe des Roboters demontiert und deren Unterteile wieder in den Montageprozess eingelastet werden. Es soll eine Belastungsschranke für die Anlage definiert werden. Kriterium für die Einlastung eines neuen Auftrags soll die Belastungsschranke sein. Ist die Belastungsschranke überschritten, dürfen keine Werkstücke mehr aus der Station 20 der Station 30 zugeführt werden, da sonst die Anlage blockieren kann, wenn sich noch defekte Werkstücke im Umlauf befinden.

Es ist unbedingt zu vermeiden, dass die Handlingeinheit *LIHNDL* ein Werkstück zum Band 5 beziehungsweise 6 bringt, während der Roboter ein Werkstück von Station 60 zum Band 5 beziehungsweise 6 bringt, da sonst beide Werkstücke auf dem Band kollidieren können. Die genauen Positionen der Werkstücke zwischen den Sensoren *ST30.B.1* und *ST30.B.2* beziehungsweise *ST30.B.4* und *ST30.B.5* können nicht detektiert werden. Des Weiteren soll die Handlingeinheit *LIHNDL* nach Abschluss der optischen Prüfung des Werkstücks an der Station 60 das fertige Werkstück, wenn es einpolig ist, zum Band 3 und, wenn es zweipolig ist, zum Band 4 bringen.

Auf dem Paletten-Transfersystem befinden sich mehrere Puffer mit beschränkter Kapazität. Zudem zirkulieren hier neun Paletten und jede Palette besitzt eine Werkstückkapazität

von eins. Für das Transfersystem soll zusätzlich noch eine globale Spezifikation formuliert werden, welche die Anzahl der beladenen Paletten auf dem gesamten Transfersystem überwacht, steuert und somit eine mögliche Blockierung durch den Palettenpuffer vor Station 30 vermeidet. Dieser Fall wird in Unterkapitel 6.2.12 näher erläutert.

6.2. Spezifikationen

In diesem Unterkapitel werden die Spezifikationen entwickelt. Abbildung 6.1 repräsentiert die pufferbezogene Dekomposition der Montageanlage. In dieser Repräsentation werden Puffer durch Kreise dargestellt. In den Klammern sind die jeweiligen Kapazitäten der Puffer angegeben, die in der gesteuerten Strecke maximal erlaubt sind. Die Pfeile stellen den Werkstückfluss beziehungsweise den Palettenfluss und somit den möglichen Verlauf von Werkstücken durch die Anlage dar. Die grün gestrichelten Linien grenzen die einzelnen Arbeitstationen voneinander ab. Des Weiteren sind die Pfeile in Abbildung 6.1 mit den Ein- und Ausgangsereignissen der einzelnen Puffer, die die Pufferbestände de- und inkrementieren, betitelt.

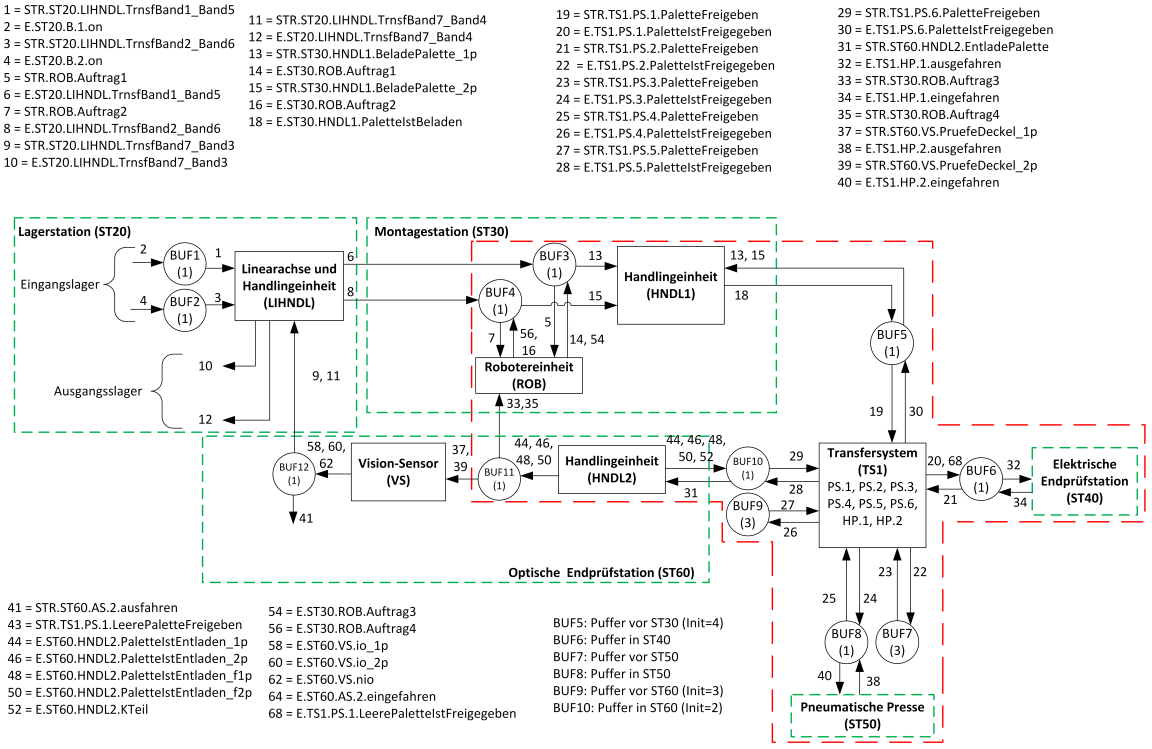


Abbildung 6.1.: Pufferbezogene Dekomposition der Montageanlage.

Die Puffer (engl. Buffer, BUF) werden durch Spezifikationen in Form von Generatoren formalisiert. Die insgesamt 14 Puffer heißen $BUFi$ und die zugehörigen Spezifikationen K_{Xi} mit $i = [1, ..., 14]$. Die Spezifikationen K_{X13} und K_{X14}) sind globale Pufferspezifikationen und beziehen sich auf die gesamte Anlage beziehungsweise auf einen Großteil der Anlage.

K_{X1} bis K_{X4} und K_{X11} bis K_{X12} stellen Spezifikationen zum Puffermanagement für Werkstückpuffer dar, während K_{X5} bis K_{X10} Spezifikationen zum Puffermanagement für Palettenpuffer darstellen. Aus Übersichtgründen wurden die einzelnen Pufferbereiche in dem in Abbildung 6.2 dargestellten Anlagenschema grün eingezeichnet. Die rote Pfeile stellen den Werkstückfluss von Station 60 nach Station 30 dar.

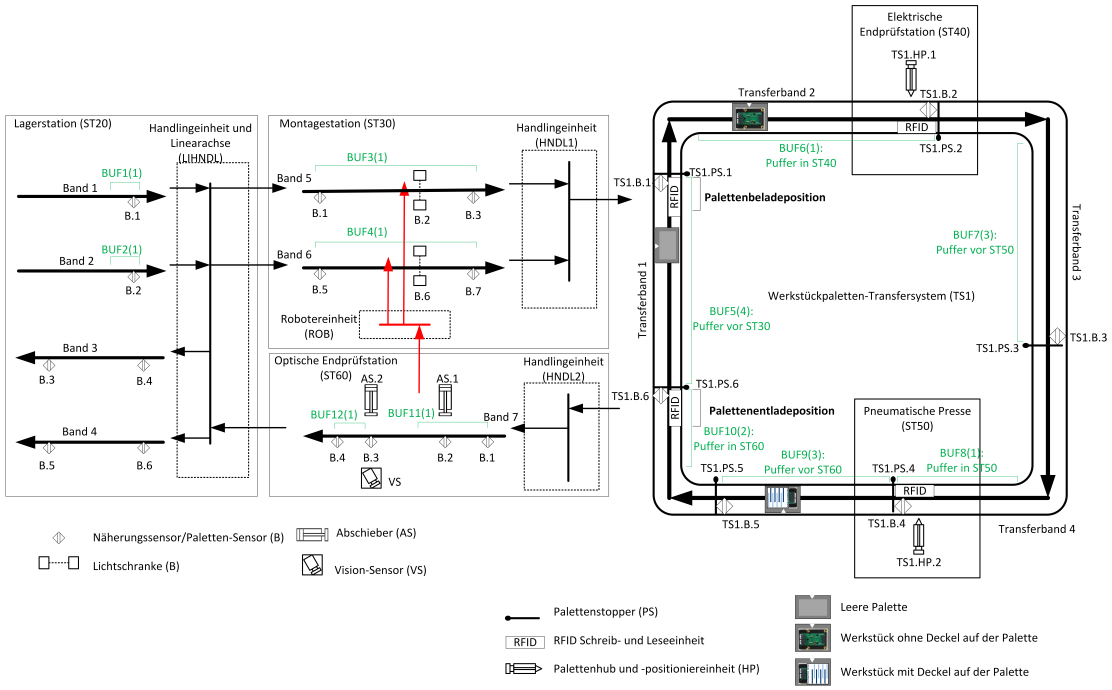
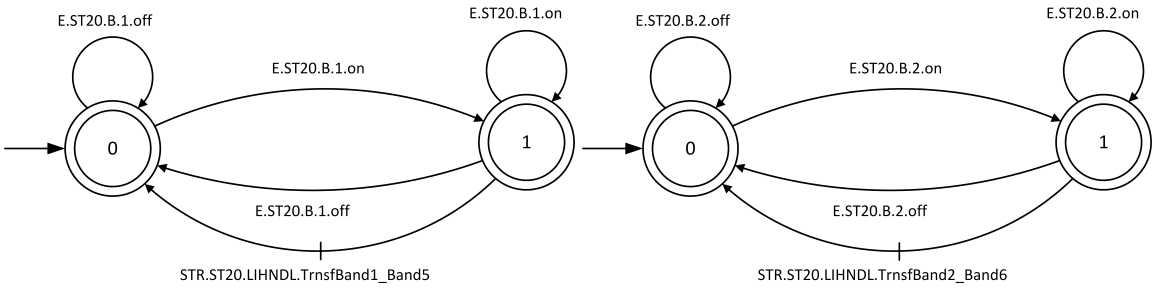


Abbildung 6.2.: Schematischer Aufbau der Anlage inklusive Puffer.

6.2.1. Spezifikation K_{X1} und K_{X2}: Puffermanagement für die Station 20 (BUF1 und BUF2)

Die Spezifikationen K_{X1} und K_{X2} beschreiben das Puffermanagement für die Bänder 1 und 2 in Station 20 (BUF1(1) und BUF2(1), vgl. Abbildung 6.1 und 6.2). Wie schon in Unterkapitel 5.1 erwähnt wurde, wird bei der Modellierung davon ausgegangen, dass stets genügend unbearbeitete Werkstücke auf dem Eingangslager auf den Montageprozess warten. Deshalb wird der Vorrat hier als unendlich angenommen. Die Kapazität des Ausgangslagers wird auch als unendlich angenommen und hier nicht weiter betrachtet. Es werden lediglich die

Werkstückentnahmepositionen an den Bandenden von 1 und 2 als Puffer modelliert. Sie haben jeweils die Kapazität 1. Abbildung 6.3 stellt die Spezifikationen K_{X1} und K_{X2} als Generator dar.



(a) Spezifikation K_{X1} als Generator.

(b) Spezifikation K_{X2} als Generator.

Abbildung 6.3.: Spezifikation K_{X1} und K_{X2}.

Stellt der Sensor *ST20.B.1* beziehungsweise *ST20.B.2* ein Werkstück fest, wird der Puffer BUF1 beziehungsweise BUF2 gefüllt. Der Puffer wird durch die Handlingeinheit *LIHNDL* entleert, sodass das Ausgangsereignis *STR.ST20.LIHNDL.TrnsfBand1_Band5* beziehungsweise *STR.ST20.LIHNDL.TrnsfBand2_Band6* ist. Entleert werden können die Puffer auch durch manuelle Eingriffe des Anwenders. In diesem Fall wird das Sensorereignis *E.ST20.B.1.off* beziehungsweise *E.ST20.B.2.off* ausgelöst und der entsprechende Puffer entleert.

6.2.2. Spezifikation K_{X3} und K_{X4}: Puffermanagement für die Station 30 (BUF3 und BUF4)

Die Bänder 5 und 6 verfügen über keinen Werkstückstopper, die die Werkstücke vor der Lichtschranke *ST30.B.2* beziehungsweise *ST30.B.5* oder die Werkstücke am Bandende vor dem Sensor *ST30.B.3* beziehungsweise *ST30.B.6* stoppen und aufstauen können. Deshalb sind die Bänder in dieser Station jeweils als Puffer der Kapazität 1 modelliert (BUF3(1) und BUF4(1), vgl. Abbildung 6.1 und 6.2). Abbildung 6.4 zeigt den Generator, der die Spezifikation K_{X3} repräsentiert.

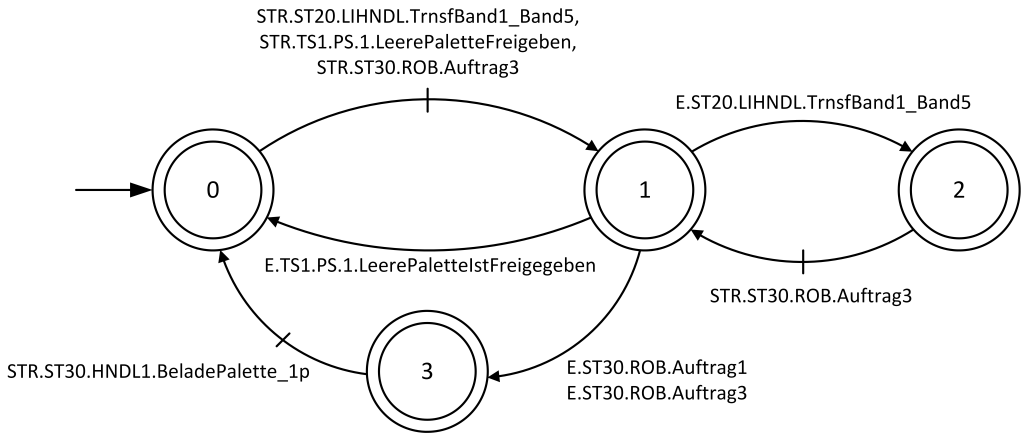


Abbildung 6.4.: Spezifikation K_{X3} als Generator.

Im Gegensatz zu den Generatoren aus Abbildung 6.3 stellt in Spezifikation K_{X3} die Nummerierung der Zustände nicht die Pufferbestände dar. Die Zustände 1 und 2 sind Zwischenzustände. Der Bestand des Puffers wird durch Zuführen eines Werkstücks aus dem Lager mittels *STR.ST20.LIHNDL.TrnsfBand1_Band5* oder durch Zuführen eines Werkstücks aus der Station 60 mittels *STR.ST30.ROB.Auftrag3* gefüllt. Die beiden Ereignisse sind in der ungesteuerten Strecke gleichzeitig möglich, weil sie nicht aus demselben Generator stammen. Sie dürfen aber nicht gleichzeitig stattfinden und werden in K_{X3} durch gegenseitigen Ausschluss verhindert. Dadurch entsteht der Zwischenzustand 1. Sind beide Ereignisse im gleichen Programmzyklus möglich, findet *STR.ST30.ROB.Auftrag3* priorisiert statt. Die priorisierte Auswahl wird durch die Codegenerierung festgelegt. Die Codegenerierung wird in Kapitel 7 näher beschrieben. Der Generator wechselt in den Zustand 2, wenn das Werkstück von der Handlingeinheit *LIHNDL* zugeführt wurde. Das Ereignis *STR.ST30.ROB.Auftrag1* ist als Folgeereignis von *E.ST20.LIHNDL.TrnsfBand1_Band5* erlaubt. Sobald eine Relaiskarte auf das Werkstück eingesetzt wurde, wird das Ereignis *E.ST30.ROB.Auftrag1* generiert und der Zustand 3 aktiviert. Entleert wird der Puffer durch die Handlingeinheit *HNDL2*, die mit dem Ereignis *STR.ST30.HNLD1.BeladePalette_1p* ausgelöst wird.

Die Spezifikation K_{X4} in Abbildung 6.5 entsteht in Analogie zur Spezifikation K_{X3} aus Abbildung 6.4.

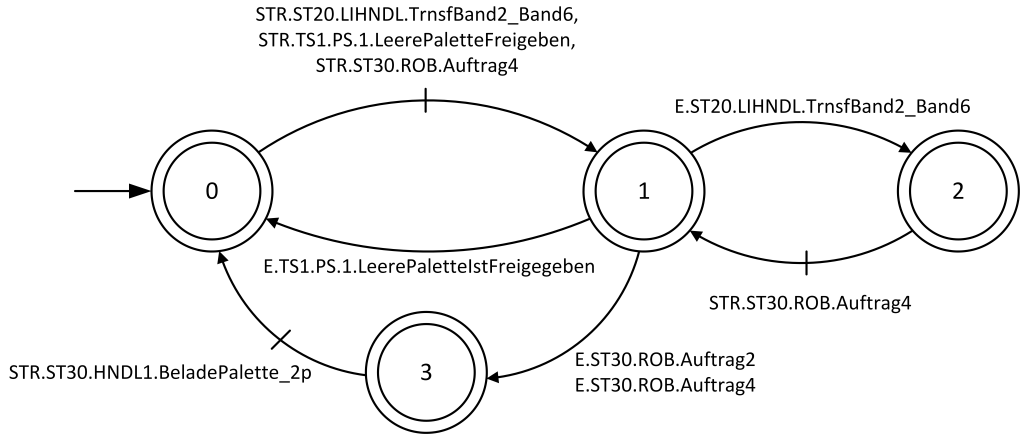


Abbildung 6.5.: Spezifikation K_{X4} als Generator.

6.2.3. Spezifikation K_{X5}: Puffermanagement für den Palettenpuffer vor Station 30 (BUF5)

Die Spezifikation für den Puffer BUF5 besteht aus zwei Teilspezifikationen. Sie sind formal zusammengefasst in K_{X5}. Diese ist mit 15 Zuständen und 41 Transitionen die größte Spezifikation der Steuerung. Die erste Teilspezifikation K_{X5_1} wird durch den in Abbildung 6.6 dargestellten Generator repräsentiert.

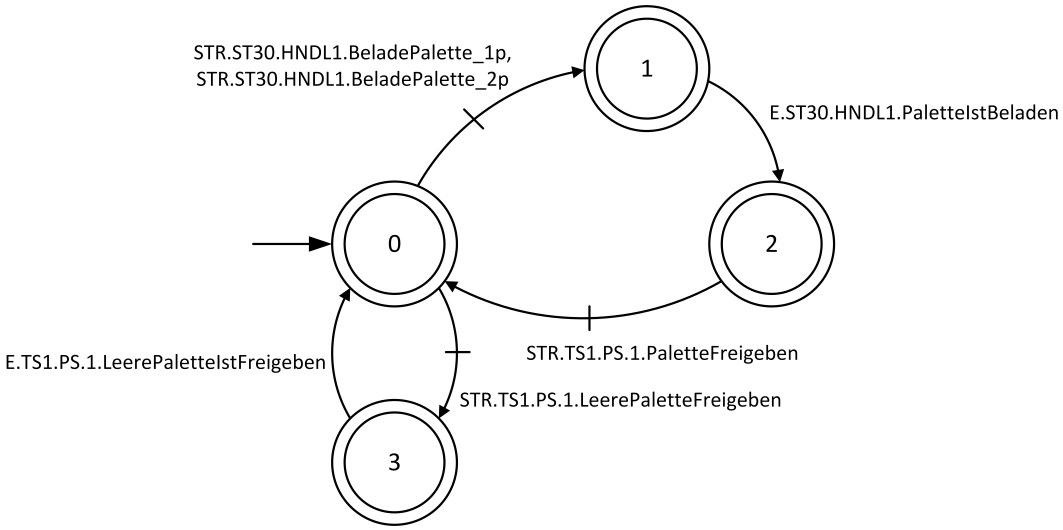


Abbildung 6.6.: Spezifikation K_{X5_1} als Generator.

Diese Spezifikation stellt zum einen sicher, dass der Stopper *PS.1* keine leere Palette freigibt, wenn im Puffer BUF3 oder BUF4 (BUF3(1) und BUF4(1), vgl. Abbildung 6.1 und 6.2) ein Werkstück vorhanden ist, und zum anderen, dass eine Palette nur mit einem Werkstück beladen wird. Die Ereignisse *STR.ST30.HNDL1.BeladePalette_1p* und *STR.ST30.HNDL1.BeladePalette_2p* stammen aus demselben Generator und können nicht gleichzeitig stattfinden. Der Zustand 2 ist ein Zwischenzustand und wird nach der Beladung der Palette, wenn die Handlingeinheit *HNDL1* wieder ihre Grundstellung erreicht hat, durch das Ereignis *E.ST30.HNDL1.PaletteIstBeladen* aktiviert. Anschließend kann die beladene Palette durch das Ereignis *STR.TS1.PS.1.PaletteFreigeben* freigegeben werden. Verbieten die Spezifikationen K_{X3} und K_{X4} die Ereignisse *STR.ST30.HNDL1.BeladePalette_1p* und *STR.ST30.HNDL1.BeladePalette_2p*, kann in der Spezifikation K_{X5_1} der Zustandsübergang von 0 nach 1 sowie von 2 nach 0 nicht stattfinden. Daher entsteht der Zustand 3. Durch das Ereignis *STR.TS1.PS.1.LeerePaletteFreigeben* kann eine leere Palette freigegeben werden.

Abbildung 6.7 zeigt die zweite Teilspezifikation K_{X5_2}. Sie beschränkt die Anzahl der Paletten im Bereich zwischen den Stopper *PS.6* und *PS.1* auf vier Paletten (BUF5(4), vgl. Abbildung 6.1 und 6.2). Diese Entscheidung begründet sich dadurch, dass Paletten am Stopper *PS.1* auf ihre Freigabe an die Station 40 warten und durch das permanent laufende Transferband 1 eine hohe mechanische Kraft auf den Stopper ausüben. Um den Stopper nicht zu überlasten, darf der Bestand dieses Puffers vier Paletten niemals überschreiten. Das Eingangsereignis des Puffers ist *E.TS1.PS.6.PaletteIstFreigegeben*, das Ausgangsereignis des Puffers ist *STR.TS1.PS.1.PaletteFreigeben*. Ist der Puffer voll, entspricht der Zustand 4, so kann durch das Ereignis *STR.TS1.PS.1.LeerePaletteFreigeben* eine leere Palette freigegeben werden. Dieses Ereignis ist außerdem steuerbar und Teil der Spezifikationen K_{X3}, K_{X4}, K_{X6}, K_{X10} und K_{X14}. Damit eine leere Palette freigegeben werden kann, müssen alle diese Spezifikationen das Ereignis erlauben. Sind alle Paletten leer, entspricht der Initialzustand der Spezifikation K_{X14}, dann tritt dieses Ereignis nicht auf.

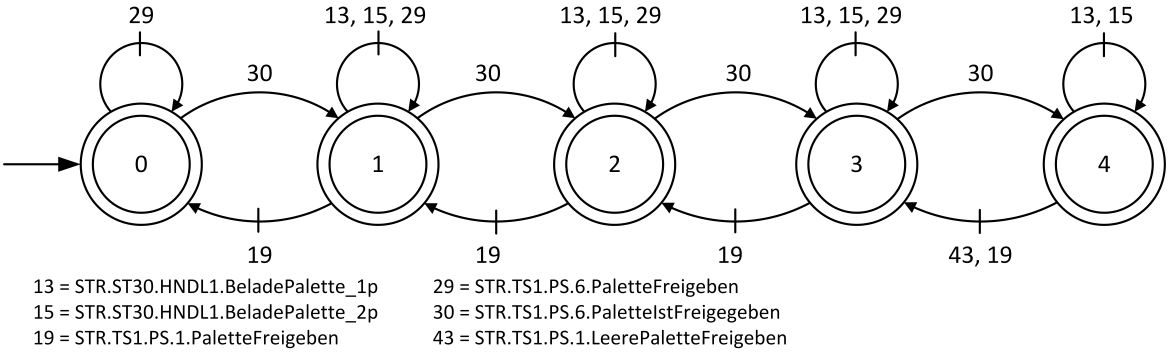


Abbildung 6.7.: Spezifikation K_{X5_2} als Generator.

6.2.4. Spezifikation K_{X6}: Puffermanagement für den Puffer vor Station 40 (BUF6)

Der Bereich zwischen den Stoppern *PS.1* und *PS.2* (BUF6(1), vgl. Abbildung 6.1 und 6.2) stellt einen physikalischen Palettenpuffer der Kapazität 17 dar. Die Station 40 darf aus physikalischen Gründen maximal eine Palette enthalten. Zudem kann eine durch *PS.1* freigegebene Palette nicht mehr vom Eintritt in die Station 40 abgehalten werden. Abbildung 6.8 zeigt die Spezifikation K_{X6}, die die Anzahl an Freigaben von *PS.1* auf eins beschränkt und erst dann weitere Freigaben erlaubt, wenn der Stopper *PS.2* eine Palette freigibt.

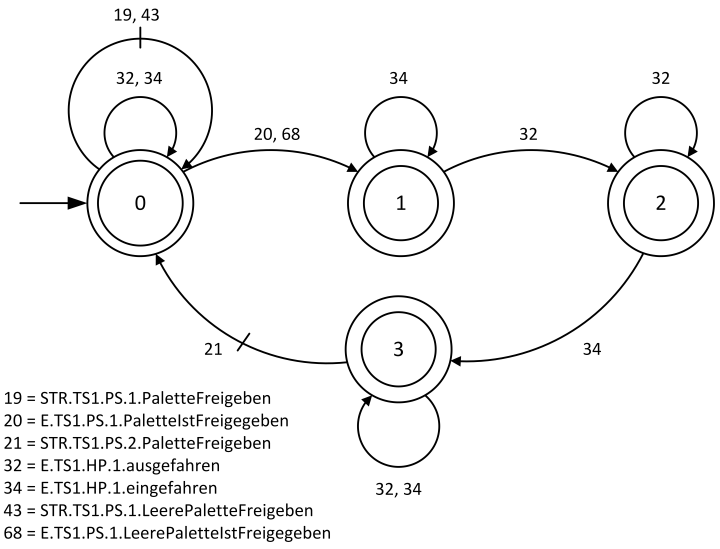


Abbildung 6.8.: Spezifikation K_{X6} als Generator.

Im Zustand 0 (Initialzustand) ist der Puffer leer. Die steuerbaren Ereignisse *STR.TS1.PS.1.PaletteFreigeben* und *STR.TS1.PS.1.LeerePaletteFreigeben*, die das Lösen von *PS.1* erlauben und so die Palette freigeben, sind nur in dem Zustand 0 erlaubt. Die Zustände 1 und 2 sind Zwischenzustände. Diese Zwischenzustände müssen in der Spezifikation berücksichtigt werden, weil sonst die Palette beim Erreichen des Palettensensors *TS1.B.2* sofort vom *PS.2* wieder freigegeben wird und dann das Werkstück auf der Palette ungeprüft in die Station 50 läuft. Im Zustand 1 befindet sich die Palette im Bereich zwischen den Stoppern *PS.1* und *PS.2*. Nachdem die Palette an der Station durch die Pakettenhub- und -positioniereinheit *HP.1* gespannt und fixiert ist, wird der Zustand 2 aktiviert und das Werkstück wird zur elektrischen Prüfung freigegeben. Das Spannen der Palette wird durch das Endschalterereignis *E.TS1.HP.1.ausgefahren* gemeldet. Nach der Fertigmeldung der Station wird die gespannte Palette wieder entspannt und es wird das Endschalterereignis

E.TS1.PS.1.eingefahren gesetzt. Dadurch erfolgt der Zustandswechsel von Zustand 2 zum Zustand 3. Die Palette kann anschließend vom *PS.2* freigegeben werden.

6.2.5. Spezifikation K_{X7}: Puffermanagement für den Puffer vor Station 50 (BUF7)

Der Bereich zwischen den Stoppern *PS.2* und *PS.3* bis zur Kurve (BUF7(3), vgl. Abbildung 6.1 und 6.2) weist eine physikalische Kapazität von 11 Paletten auf. An den Kurven dürfen keine Paletten stehen bleiben, da die Kurven nicht staufähig sind und ein Hindernis darstellen. Des Weiteren ist der Bestand in diesem Bereich auf drei Paletten zu beschränken, um eine Überlastung des Stoppers *PS.3* zu verhindern. Die Begründung erfolgte schon in Unterkapitel 6.2.3. Abbildung 6.9 stellt die Spezifikation K_{X7} als Generator dar.

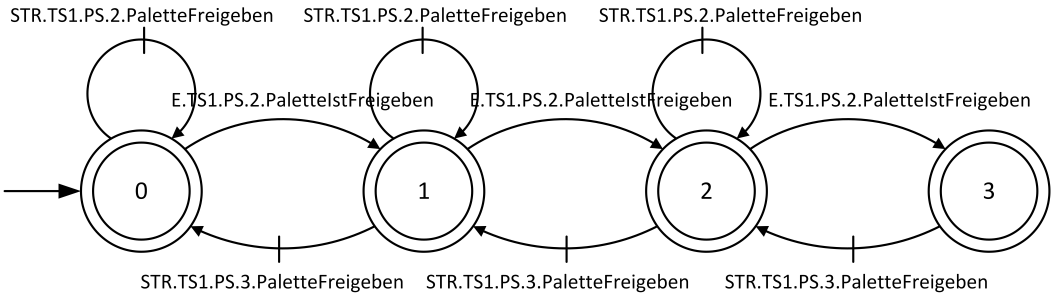


Abbildung 6.9.: Spezifikation K_{X7} als Generator.

Jeder Zustand repräsentiert den Pufferbestand. Die Anzahl an Paletten in diesem Bereich wird um eins inkrementiert, wenn das nicht steuerbare Ereignis *E.TS1.PS.2.PalettelstFreigegeben* auslöst. Das Ereignis *STR.TS1.PS.2.PaletteFreigeben*, welches das Lösen von *PS.2* erlaubt, ist nur in den Zuständen 0, 1 und 2 erlaubt. Ist der Puffer voll, entspricht der Zustand 3, dürfen keine Paletten mehr vom Stopper *PS.2* freigegeben werden. Der Bestand wird um eins dekrementiert, wenn der Stopper *PS.3* eine Palette freigibt.

6.2.6. Spezifikation K_{X8}: Puffermanagement für den Puffer in Station 50 (BUF8)

Die Spezifikation in Abbildung 6.10 modelliert den Puffer zwischen den Stopper *PS.3* und *PS.4* (BUF8(1), vgl. Abbildung 6.1 und 6.2) und weist eine ähnliche Zustands- und Transitionsstruktur auf, wie der Generator der Spezifikation K_{X6}. In diesem Bereich können physi-

kalisch neun Paletten Platz finden. Aber die physikalische Kapazität der Station 50 beträgt eins. Weitere Paletten können vom Eintritt in die Station nicht abgehalten werden. Die letzte Möglichkeit, eine Blockierung in dieser Station zu verhindern, stellt also der Stopper *PS.3* dar. Wenn dieser eine Palette freigibt, wird der Puffer gefüllt. Der Puffer wird beim Austritt aus dem Zustand 3 durch das steuerbare Ereignis *STR.TS1.PS.4.PaletteFreigeben* entleert.

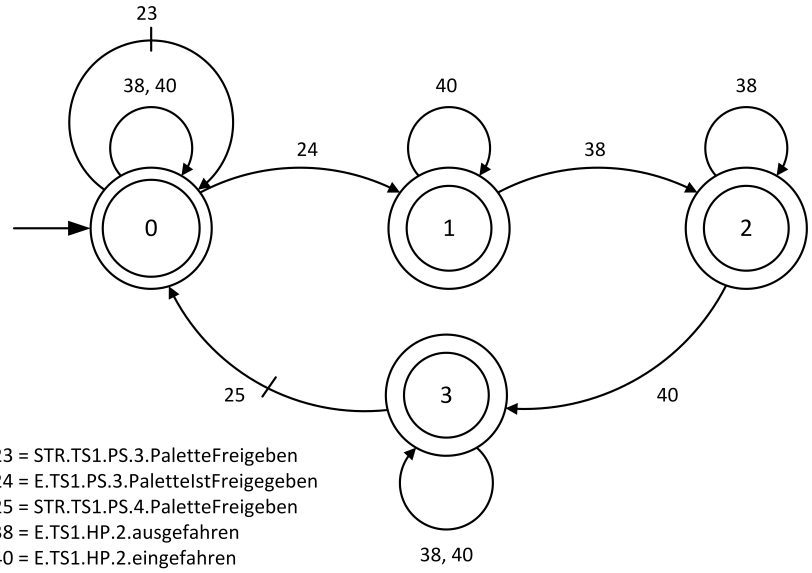


Abbildung 6.10.: Spezifikation K_{X8} als Generator.

6.2.7. Spezifikation K_{X9}: Puffermanagement für den Puffer vor Station 60 (BUF9)

Das Puffermanagement für den Puffer vor Station 60 (BUF9(3), vgl. Abbildung 6.1 und 6.2) erfolgt analog zur Spezifikation K_{X7} aus Abbildung 6.9, jedoch mit den Ereignissen *STR.TS1.PS.4.PalettFreigeben*, *STR.TS1.PS.5.PalettFreigeben* und *E.TS1.PS.4.PalettFreigegeben*. Der Unterlauf des Pufferbestandes wird durch das Verbieten des Ereignisses *STR.TS1.PS.5.PaletteFreigeben* im Zustand 0 verhindert. Abbildung 6.11 zeigt die Spezifikation K_{X9} als Generator.

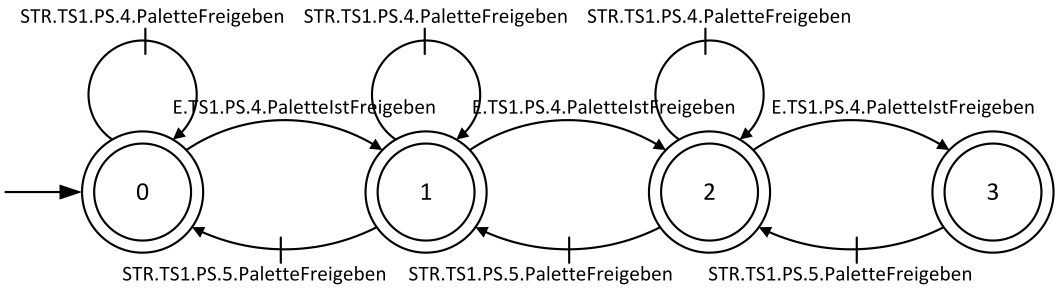


Abbildung 6.11.: Spezifikation K_{X9} als Generator.

6.2.8. Spezifikation K_{X10}: Puffermanagement für den Palettenpuffer in Station 60 (BUF10)

Die Spezifikation K_{X10} besteht aus zwei Teilspezifikationen, die durch parallelen Komposition zusammengefasst sind. Sie enthält 7 Zustände und 25 Transitionen und beschränkt die Anzahl der Paletten im Bereich zwischen den Stoppern *PS.5* und *PS.6* bis zur Kurve auf zwei Paletten (BUF10(2), vgl. Abbildung 6.1 und 6.2). Abbildung 6.12 zeigt die erste Teilspezifikation K_{X10_1} als Generator. Sie stellt sicher, dass der Stopper *PS.6* nur entladene Paletten oder leere Paletten freigibt. In dem nachfolgenden Palettenpuffer BUF5 befinden sich dann nur leere Paletten.

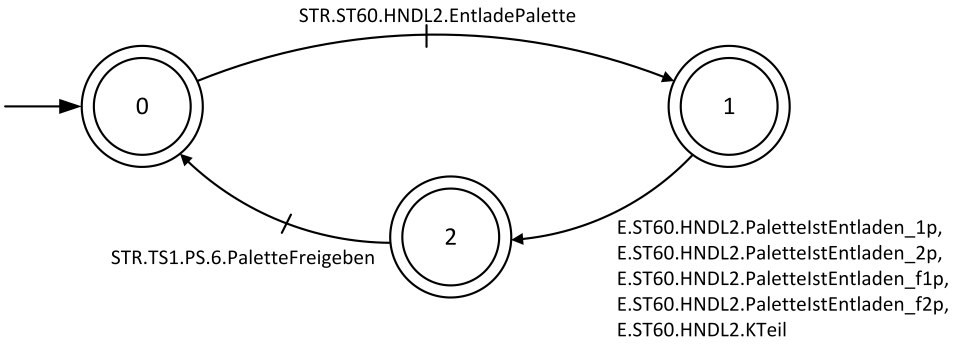


Abbildung 6.12.: Spezifikation K_{X10_1} als Generator.

Abbildung 6.13 stellt die zweite Teilspezifikation K_{X10_2} als Generator dar. Sie beschränkt die Anzahl an Freigaben von *PS.5* auf zwei und erlaubt erst dann weitere Freigaben, wenn der Stopper *PS.6* mindestens einmal aktiviert wurde.

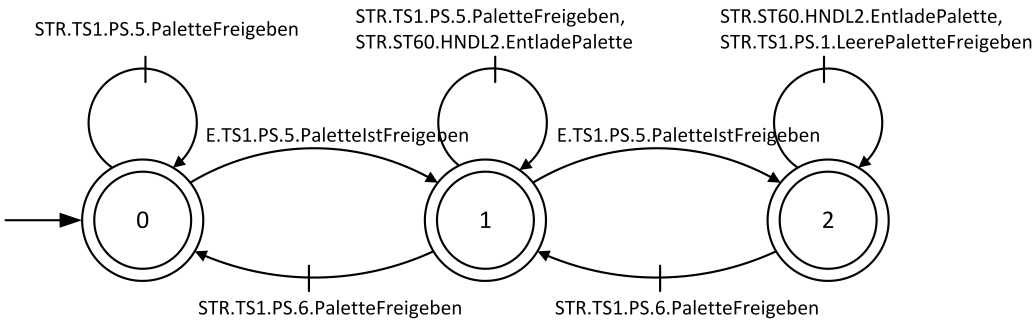


Abbildung 6.13.: Spezifikation K_{X10_2} als Generator.

6.2.9. Spezifikation K_{X11}: Puffermanagement für die Station 60 (BUF11)

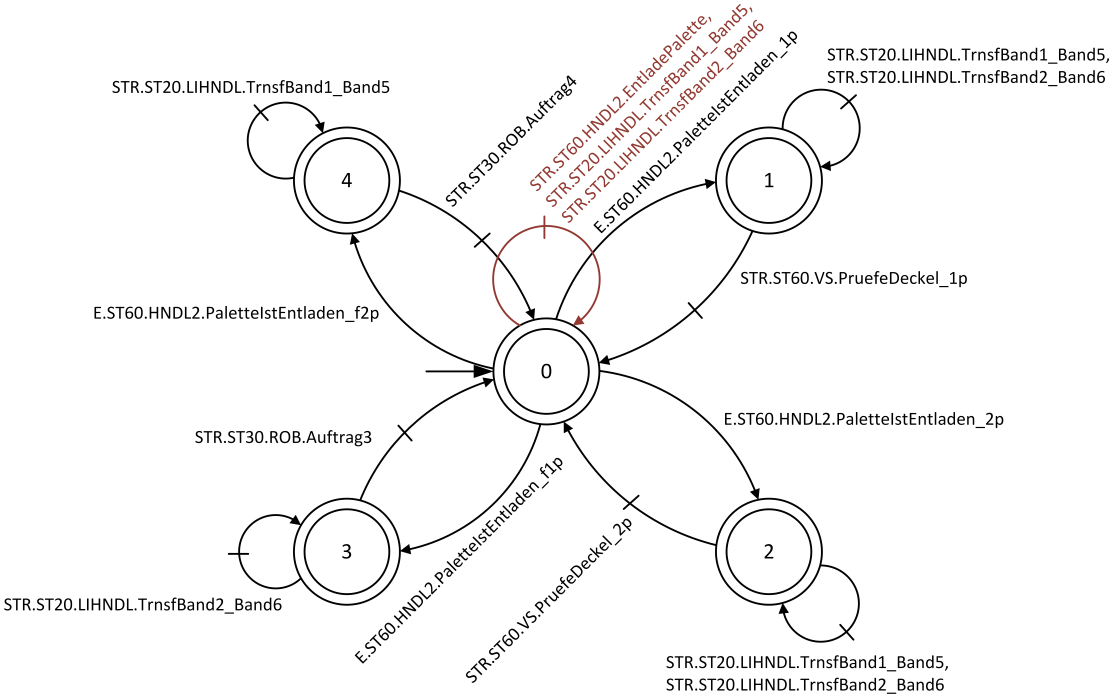


Abbildung 6.14.: Spezifikation K_{X11} als Generator.

Die Spezifikation K_{X11} beschränkt den maximalen Bestand an Werkstücken auf dem Band 7 im Bereich zwischen den Sensoren *ST60.B.1* und *ST60.B.2* auf ein Werkstück (vgl. BUF11(1) in Abbildung 6.2). Der Generator in Abbildung 6.14 formali-

siert diese Spezifikation. Er besteht aus fünf Zuständen. Nach der Entladung der Palette an der Station 60 tritt je nach Relaistyp und Werkstückzustand eines der Ereignisse *E.ST60.HNDL2.PaletteIstEntladen_1p*, *E.ST60.HNDL2.PaletteIstEntladen_2p*, *E.ST60.HNDL2.PaletteIstEntladen_f1p* oder *E.ST60.HNDL2.PaletteIstEntladen_f2p* auf und der Puffer wird gefüllt. Der Zustand, der danach aktiviert wird, bestimmt den Weg des Werkstücks zur Station 30 oder Station 20. Die defekten Werkstücke laufen bis vor den Sensor *ST60.B.2* und werden dort von dem Roboter demontiert und wieder der Station 30 zugeführt (Zustand 3 und 4). Der Werkstückfluss zur Station 30 wird durch die steuerbaren Ereignisse *STR.ST30.ROB.Auftrag3* und *STR.ST30.ROB.Auftrag4* gesteuert. Die verdeckelten Werkstücke laufen bis vor den Näherungssensor *ST60.B.3* und werden dort durch den Vision-Sensor *VS* optisch überprüft (Zustand 3 und 4).

Um den Überlauf des Puffers BUF3 und BUF4 zu verhindern, dürfen die Ereignisse *STR.ST30.ROB.Auftrag3* und *STR.ST20.LIHNLD.TrnsfBand1_Band5* sowie die Ereignisse *STR.ST30.ROB.Auftrag4* und *STR.ST20.LIHNLD.TrnsfBand2_Band6* nicht gleichzeitig auftreten. Diese Anforderungen werden in den Spezifikationen K_{X3} und K_{X4} durch gegenseitigen Ausschluss der jeweiligen Ereignisse sichergestellt. Um eine mögliche Blockierung durch ein defektes Werkstück in Station 60 zu vermeiden, soll der Roboter **absolute Priorität** gegenüber der Handlingeinheit *LIHNDL* haben. Dies wird bei der Implementierung durch priorisierte Ereignisauswahl erreicht. Jedoch findet die priorisierte Auswahl hier erst dann statt, wenn die entsprechenden Ereignisse im gleichen Programmzyklus auftreten. Wenn beispielsweise die Spezifikation K_{X11} in dem Zustand 3 oder 4 steht und das Ereignis *STR.ST30.ROB.Auftrag3* oder *STR.ST30.ROB.Auftrag4* durch das Produktsystem nicht erlaubt wird, weil bereits das Ereignis *STR.ST30.ROB.Auftrag2* oder *STR.ST30.ROB.Auftrag3* aufgetreten ist, kann hier eine priorisierte Auswahl nicht stattfinden. In diesem Fall wird je nach Auftragslage immer das Ereignis *STR.ST20.LIHNLD.TrnsfBand1_Band5* oder *STR.ST20.LIHNLD.TrnsfBand2_Band6* stattfinden. Da dann das defekte Werkstück in Station 60 nicht bearbeitet werden kann, können weitere Paletten auch nicht entladen werden, weil der entsprechende Puffer voll ist. Um das zu verhindern, muss zusätzlich noch in der Spezifikation K_{X11} das Ereignis *STR.ST20.LIHNLD.TrnsfBand1_Band5* im Zustand 4 und das Ereignis *STR.ST20.LIHNLD.TrnsfBand2_Band6* im Zustand 3 verboten werden. In den anderen Zuständen erlaubt K_{X11} die beiden Ereignisse.

6.2.10. Spezifikation K_{X12}: Puffermanagement für den Entnahmepuffer am Band 7 (BUF12)

Die Anzahl an Werkstücken auf dem Band 7 im Bereich zwischen den Sensoren *ST60.B.3* und *ST60.B.4* (vgl. BUF12(1) in Abbildung 6.2) ist aus physikalischen Gründen auf eins zu beschränken. Diese Spezifikation wird durch den Generator in Abbildung 6.15 formalisiert. Der Puffer wird durch je eines der Ereignisse *E.ST60.VS.io_1p*, *E.ST60.VS.io_2p*

und *E.ST60.VS.nio* gefüllt. Ein möglicher Pufferüberlauf wird durch verbieten der Ereignisse *STR.ST60.VS.PruefeDeckel_1p* und *STR.ST60.VS.PruefeDeckel_2p* im Zustand 0 verhindert. Bei einem positiven Ergebnis des Vision-Sensors, wechselt der Generator in den Zustand 1 oder 2. Diese Zustände entscheiden, ob das fertige Werkstück zum Band 3 oder 4 transportiert werden soll. Ist das Werkstück einpolig und optisch in Ordnung (Zustand 1), wird es durch das Ereignis *STR.ST20.LIHNDL.TrnsfBand7_Band3* zum Band 3 transportiert und gleichzeitig wird der Puffer entleert. Ist das Werkstück zweipolig und optisch in Ordnung (Zustand 2), dann wird es durch das Ereignis *STR.ST20.LIHNDL.TrnsfBand7_Band4* zum Band 4 transportiert und der Puffer wird entleert. Bei einem Fehlergebnis des Vision-Sensors wechselt der Generator in den Zustand 3. In diesem Zustand wird durch das Ereignis *STR.ST60.AS.2.ausfahren* der Abschieber *AS.2* ausgefahren und das Werkstück aus dem Band entfernt. Durch dieses Ereignis wechselt der Generator wieder in den Zustand 0 (Puffer leer).

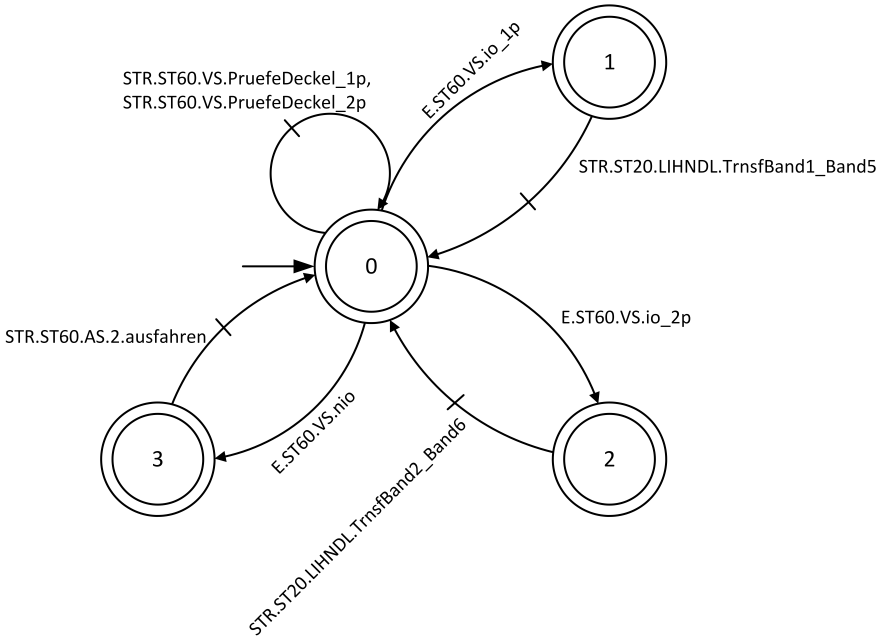


Abbildung 6.15.: Spezifikation K_{X12} als Generator.

6.2.11. Spezifikation K_{X13}: Belastungsschranke

Für die Festlegung der Belastungsschranke der Anlage wird in dieser Arbeit der geschlossene Pfad über die Stationen 30, 40, 50, 60 und den Roboter betrachtet. Die rote Linie in der Abbildung 6.1 stellt diesen Pfad dar. Die Summe aller auf diesem Pfad befindlichen Pufferkapazitäten beträgt 11 Werkstücke. Im ungünstigsten Fall, wenn die Anzahl der Werkstücke

auf diesem Pfad gleich die Summe aus allen auf demselben Pfad befindlichen Pufferplätzen ist und alle diese Werkstücke als Mitkopplung (defekt aber reparabel) wieder in den Montageprozess hinzugefügt werden, kann es zu einer Blockierung der Anlage kommen, da dann keine Spezifikation mehr auf dieser Strecke einen Steuereingriff tätigen kann. Daher muss die Obergrenze der Belastungsschranke um eins weniger sein als 11. Zudem ist bei der Festlegung der Belastungsschranke die Anzahl der verfügbaren Paletten auf dem Transfersystem als Nebenbedingung zu betrachten. Es sind insgesamt neun Paletten verfügbar und jede hat eine Kapazität von einem Werkstück. Es ergibt sich eine Belastungsschranke von 10 Werkstücken. Jeder Auftrag, dessen Einlastung diese Belastungsschranke überschreiten würde, wird gespeichert aber nicht freigegeben. In anderen Worten, neue Aufträge werden in die Anlage eingelastet, solange die Belastungsschranke nicht überschritten ist. Abbildung 6.16 zeigt die Spezifikation K_{X13} als Generator.

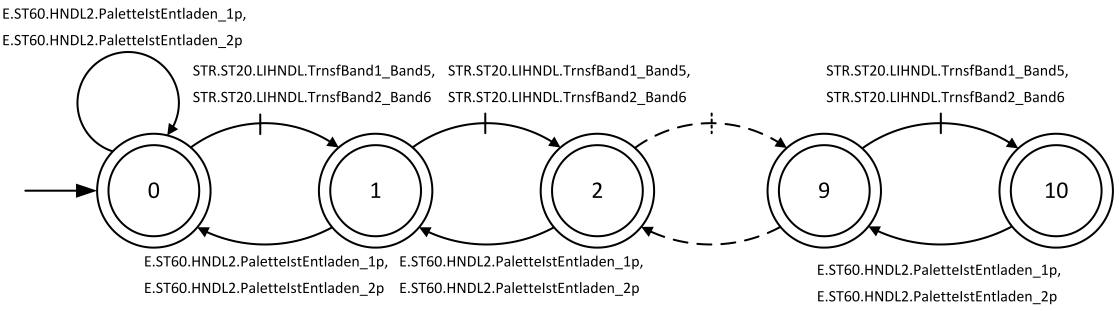


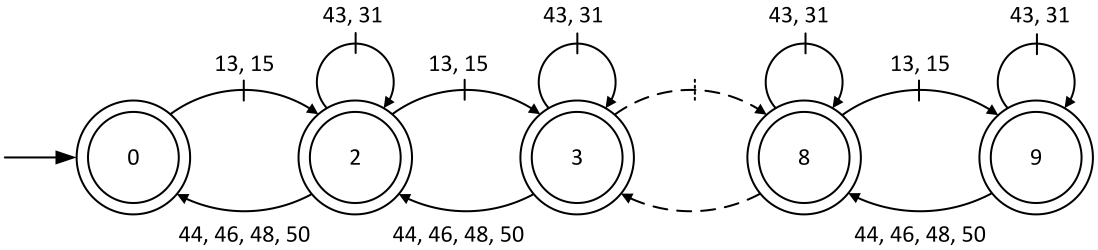
Abbildung 6.16.: Spezifikation K_{X13} für die Belastungsschranke der Anlage als Generator.

Jeder Zustand repräsentiert den Bestand an Werkstücken auf dem roten Pfad (vgl. Abbildung 6.1 und 6.2). Das Auftreten je eines der Ereignisse *STR.ST20.LIHNDL.TrnsfBand1_Band5* und *STR.ST20.LIHNDL.TrnsfBand2_Band6* erhöht den Gesamtbestand. Ist die Belastungsschranke erreicht (Zustand 10), wird das weitere Einlasten von Werkstücken in die Anlage erst dann erlaubt, wenn ein verdeckeltes Werkstück dem Transfersystem entnommen und an die Station 60 übergeben wird. Das Ereignis *E.ST60.HNDL2.PaletteIstEntladen_ip* mit $i = [1, 2]$ muss also mindestens einmal aktiviert sein. Nur die verdeckelten Werkstücke verlassen die Rückführschleife und können das Ausgangslager erreichen.

6.2.12. Spezifikation K_{X14}: Globales Puffermanagement für das Paletten-Transfersystem (TS1)

Weiterhin existiert ein globales Puffermanagement im Paletten-Transfersystem. Hier dürfen maximal neun beladene Paletten im Umlauf sein. Das wird durch die Spezifikation K_{X14} gewährleistet. Sind keine freien Plätze in Puffer BUF5 und BUF10 vorhanden und sind zudem

die Puffer in Station 30 leer, dann kommt es zu einer Blockierung, wenn sich noch beladene Paletten im Umlauf befinden, denn sie können die Palettenentladeposition nicht mehr erreichen. In diesem Fall tätigt die Spezifikation K_{X14} einen Steuereingriff, sodass der Stopper *PS.1* eine leere Palette freigibt. Dadurch werden Plätze in den vorgelagerten Puffern frei. Dies geht soweit, bis alle beladenen Paletten die Entladeposition (Palettenentladeposition, vgl. Abbildung 6.2) erreichen und entladen werden. Abbildung 6.17 zeigt die Spezifikation K_{X14} als Generator.



13 = STR.ST30.HNDL1.BeladePalette_1p
15 = STR.ST30.HNDL1.BeladePalette_2p
31 = STR.ST60.HNDL2.EntladePalette
43 = STR.TS1.PS.1.LeerePaletteFreigeben
44 = E.ST60.HNDL2.PaletteIstEntladen_1p
46 = E.ST60.HNDL2.PaletteIstEntladen_2p
48 = E.ST60.HNDL2.PaletteIstEntladen_f1p
50 = E.ST60.HNDL2.PaletteIstEntladen_f2p

Abbildung 6.17.: Spezifikation K_{X14} als Generator.

Eine Inkrementierung des Werkstückbestandes auf dem Transfersystem erfolgt durch Übergabe des Werkstücks von Station 30 an das Transfersystem mittels *STR.ST30.HNDL1.BeladePalette_1p* oder *STR.ST30.HNDL1.BeladePalette_2p*. Die Zustände stellen die Anzahl der Werkstücke beziehungsweise beladene Paletten auf dem Transfersystem dar. Durch Übergabe des Werkstücks vom Transfersystem an Station 60 wird der Bestand dekrementiert. Die Handlingeinheit *HNDL1* wird durch die Spezifikation daran gehindert, weitere Paletten zu beladen, sobald neun beladene Paletten im Umlauf sind. Die Freigabe einer leeren Palette vom *PS.1* ist bei jedem Pufferbestand möglich, außer bei leerem Bestand (Zustand 0).

6.3. Supervisor Synthese

In diesem Unterkapitel werden nach dem lokal-modularen Ansatz Supervisor aus Modellen der Strecke und der Spezifikationen berechnet. Für die Synthese wurde das Entwicklungstool DESTool angewendet. Durch die 14 Spezifikationen werden die Generatoren der Streckenkomponenten synchronisiert. Abbildung 6.18 zeigt die Relationen zwischen den Ereignisalphabeten der Komponenten der Strecke

$$G_i' = (X_i', \Sigma_i', \delta_i', x_{i,0}', X_{i,0}'), i = [1, ..., 16]$$

und der Spezifikationen $K_{Xj} \subseteq \Sigma_{Xj}^*, j = [1, ..., 14]$.

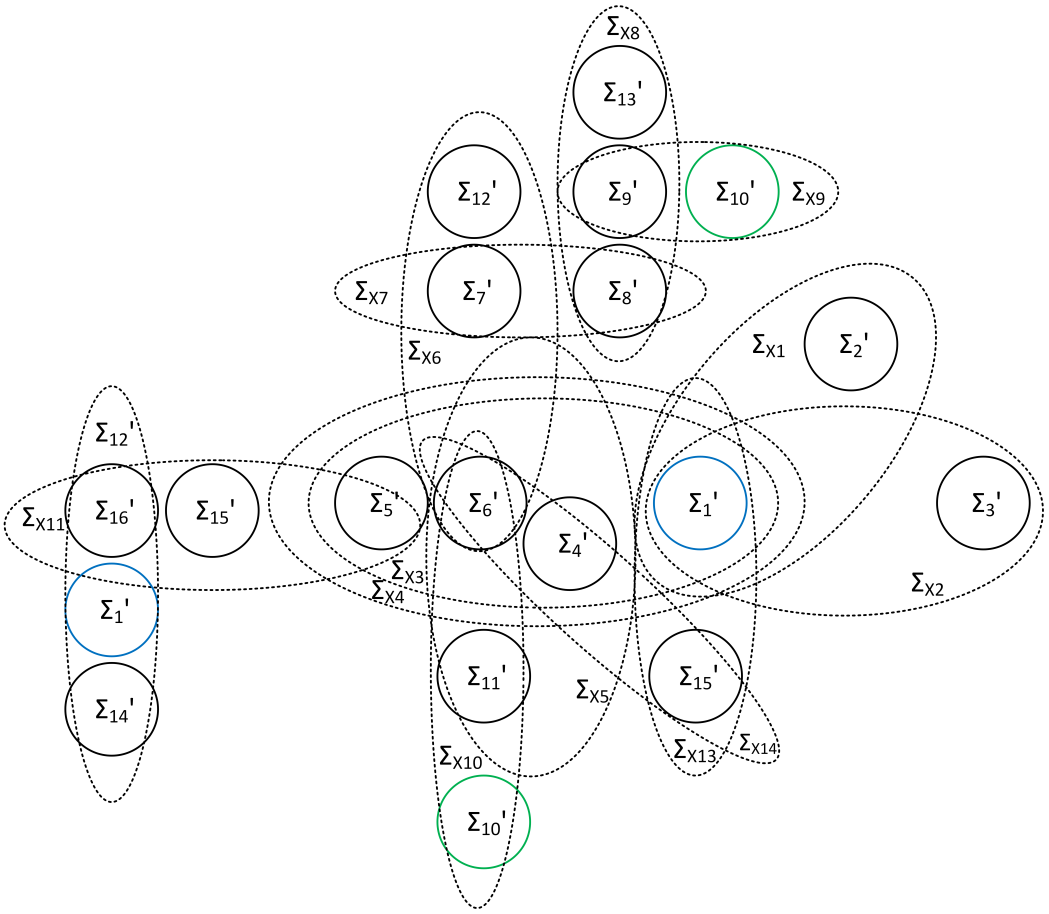


Abbildung 6.18.: Relationen zwischen den Ereignisalphabeten der Strecke und der Spezifikationen.

Das Produktsystem wurde bereits in Unterkapitel 5.3 berechnet. Als nächster Schritt werden die lokalen Systeme und die angepassten Spezifikationen mit Hilfe des DESTool-Operators *SYPC* berechnet. Die lokalen Systeme $G_{Xi}, i = [1, ..., 14]$ setzen sich folgendermaßen zusammen:

$$G_{X1} = G_1 \parallel G_2 \text{ mit } \Sigma_{G_{X1}} = \Sigma_{G_1} \cup \Sigma_{G_2}$$
$$G_{X2} = G_1 \parallel G_3 \text{ mit } \Sigma_{G_{X2}} = \Sigma_{G_1} \cup \Sigma_{G_3}$$

$$G_{X3} = G_1 \parallel G_4 \parallel G_5 \parallel G_6 \text{ mit } \Sigma_{G_{X3}} = \Sigma_{G_1} \cup \Sigma_{G_4} \cup \Sigma_{G_5} \cup \Sigma_{G_6}$$
$$G_{X4} = G_1 \parallel G_4 \parallel G_5 \parallel G_6 \text{ mit } \Sigma_{G_{X4}} = \Sigma_{G_1} \cup \Sigma_{G_4} \cup \Sigma_{G_5} \cup \Sigma_{G_6}$$
$$G_{X5} = G_4 \parallel G_6 \parallel G_{11} \text{ mit } \Sigma_{G_{X5}} = \Sigma_{G_4} \cup \Sigma_{G_6} \cup \Sigma_{G_{11}}$$
$$G_{X6} = G_6 \parallel G_7 \parallel G_{12} \text{ mit } \Sigma_{G_{X6}} = \Sigma_{G_6} \cup \Sigma_{G_7} \cup \Sigma_{G_{12}}$$
$$G_{X7} = G_7 \parallel G_8 \text{ mit } \Sigma_{G_{X7}} = \Sigma_{G_7} \cup \Sigma_{G_8}$$
$$G_{X8} = G_8 \parallel G_9 \parallel G_{13} \text{ mit } \Sigma_{G_{X8}} = \Sigma_{G_8} \cup \Sigma_{G_9} \cup \Sigma_{G_{13}}$$
$$G_{X9} = G_9 \parallel G_{10} \text{ mit } \Sigma_{G_{X9}} = \Sigma_{G_9} \cup \Sigma_{G_{10}}$$
$$G_{X10} = G_6 \parallel G_{10} \parallel G_{11} \parallel G_{15} \text{ mit } \Sigma_{G_{X10}} = \Sigma_{G_6} \cup \Sigma_{G_{10}} \cup \Sigma_{G_{11}} \cup \Sigma_{G_{15}}$$
$$G_{X11} = G_1 \parallel G_5 \parallel G_{15} \parallel G_{16} \text{ mit } \Sigma_{G_{X11}} = \Sigma_{G_1} \cup \Sigma_{G_5} \cup \Sigma_{G_{15}} \cup \Sigma_{G_{16}}$$
$$G_{X12} = G_1 \parallel G_{14} \parallel G_{16} \text{ mit } \Sigma_{G_{X12}} = \Sigma_{G_1} \cup \Sigma_{G_{14}} \cup \Sigma_{G_{16}}$$
$$G_{X13} = G_1 \parallel G_{15} \text{ mit } \Sigma_{G_{X13}} = \Sigma_{G_1} \cup \Sigma_{G_{15}}$$
$$G_{X14} = G_4 \parallel G_6 \parallel G_{15} \text{ mit } \Sigma_{G_{X14}} = \Sigma_{G_4} \cup \Sigma_{G_6} \cup \Sigma_{G_{15}}$$

Tabelle 6.1 stellt die Anzahl der Zuständen *n* und der Transitionen *t* der einzelnen lokalen Systeme dar.

Lokales System	***n***	***t***	**Lokales System**	***n***	***t***
G_{X1}	10	36	G_{X8}	8	24
G_{X2}	10	36	G_{X9}	4	8
G_{X3}	225	1320	G_{X10}	24	152
G_{X4}	225	1320	G_{X11}	150	1230
G_{X5}	18	66	G_{X12}	45	207
G_{X6}	12	40	G_{X13}	10	46
G_{X7}	4	8	G_{X14}	18	102

Tabelle 6.1.: Lokale Systeme.

Nach der Berechnung aller lokalen Systeme werden im zweiten Schritt mittels der parallelen Komposition die lokalen Spezifikationen K_{Xi} an die lokalen Systeme G_{Xi} angepasst. Die daraus resultierenden angepassten lokalen Spezifikationen E_{Xi} lauten:

$$E_{Xi} = G_{Xi} \parallel K_{Xi}, i = [1, ..., 14]$$

E_{Xi} ist lokal bezüglich ihres lokalen Systems G_{Xi}. Die Überprüfung der angepassten Spezifikationen auf Steuerbarkeit bezüglich ihrer lokalen Systeme ergibt folgende Ergebnisse:

$$IsControllable(E_{X1}, G_{X1})\surd$$
$$IsControllable(E_{X2}, G_{X2})\surd$$
$$IsControllable(E_{X3}, G_{X3})\surd$$
$$IsControllable(E_{X4}, G_{X4})\surd$$
$$IsControllable(E_{X5}, G_{X5})\surd$$
$$IsControllable(E_{X6}, G_{X6})\surd$$
$$IsControllable(E_{X7}, G_{X7})\surd$$
$$IsControllable(E_{X8}, G_{X8})\surd$$
$$IsControllable(E_{X9}, G_{X9})\surd$$
$$IsControllable(E_{X10}, G_{X10})\surd$$
$$IsControllable(E_{X11}, G_{X11})\surd$$
$$IsControllable(E_{X12}, G_{X12})\surd$$
$$IsControllable(E_{X13}, G_{X13})\surd$$
$$IsControllable(E_{X14}, G_{X14})\surd$$

Die angepassten lokalen Spezifikationen E_{Xi} sind alle steuerbar bezüglich ihrer lokalen Systeme. Es muss demnach keine supremale steuerbare Teilsprache $supC(E_{Xi}, G_{Xi})$ für $i = [1, ..., 14]$ berechnet werden. Im nachfolgenden Schritt werden mit Hilfe des DESTool-Operators *IsNonblocking* die angepassten Spezifikationen auf lokale Modularität überprüft. Formal ist also die folgende Bedingung zu überprüfen:

$$\|_{i \in I}\ \overline{E_{Xi}} = \overline{\|_{i \in I}\ E_{Xi}} \text{ für } I = [1, ..., 14] \tag{6.1}$$

Ein Versuch die Überprüfung aller angepassten Spezifikationen auf lokale Modularität in einem Schritt durchzuführen führt zum Absturz von DESTool. Deshalb wird diese Überprüfung fünferweise und sechserweise durchgeführt. Die Ergebnisse sind im Folgenden dargestellt:

$$IsNonblocking(E_{X1}, E_{X2}, E_{X3}, E_{X4}, E_{X5}, E_{X6})\surd$$
$$IsNonblocking(E_{X1}, E_{X2}, E_{X3}, E_{X7}, E_{X8}, E_{X9})\surd$$
$$IsNonblocking(E_{X1}, E_{X2}, E_{X3}, E_{X10}, E_{X11}, E_{X12})\surd$$

$$IsNonblocking(E_{X1}, E_{X2}, E_{X3}, E_{X13}, E_{X14})\surd$$
$$IsNonblocking(E_{X4}, E_{X5}, E_{X6}, E_{X7}, E_{X8}, E_{X9})\surd$$
$$IsNonblocking(E_{X4}, E_{X5}, E_{X6}, E_{X10}, E_{X11}, E_{X12})\surd$$
$$IsNonblocking(E_{X4}, E_{X5}, E_{X6}, E_{X13}, E_{X14})\surd$$
$$IsNonblocking(E_{X7}, E_{X8}, E_{X9}, E_{X10}, E_{X11}, E_{X12})\surd$$
$$IsNonblocking(E_{X7}, E_{X8}, E_{X9}, E_{X13}, E_{X14})\surd$$
$$IsNonblocking(E_{X10}, E_{X11}, E_{X12}, E_{X13}, E_{X14})\surd$$

Das Resultat der Überprüfung der angepassten Spezifikationen auf lokale Modularität zeigt 14 blockierungsfreie Spezifikationen. Sie stellen als lokale Supervisor $S_i = E_{Xi}$ mit $i = [1, ..., 14]$ in ihrer Gesamtheit eine geeignete lokal-modulare Puffersteuerung für die Anlage dar. Die Supervisor sind mit der jeweils entsprechenden Anzahl an Zuständen und Transitionen in Tabelle 6.2 dargestellt.

Supervisor	**Berechnung**	***n***	***t***
S_1	$supC(E_{X1}, G_{X1}) = E_{X1}$	15	52
S_2	$supC(E_{X2}, G_{X2}) = E_{X2}$	15	52
S_3	$supC(E_{X3}, G_{X3}) = E_{X3}$	318	1638
S_4	$supC(E_{X4}, G_{X4}) = E_{X4}$	318	1638
S_5	$supC(E_{X5}, G_{X5}) = E_{X5}$	63	151
S_6	$supC(E_{X6}, G_{X6}) = E_{X6}$	22	51
S_7	$supC(E_{X7}, G_{X7}) = E_{X7}$	14	24
S_8	$supC(E_{X8}, G_{X8}) = E_{X8}$	18	37
S_9	$supC(E_{X9}, G_{X9}) = E_{X9}$	14	24
S_{10}	$supC(E_{X10}, G_{X10}) = E_{X10}$	66	270
S_{11}	$supC(E_{X11}, G_{X11}) = E_{X11}$	450	2360
S_{12}	$supC(E_{X12}, G_{X12}) = E_{X12}$	90	269
S_{13}	$supC(E_{X13}, G_{X13}) = E_{X13}$	110	502
S_{14}	$supC(E_{X14}, G_{X14}) = E_{X14}$	171	927
Gesamt		1684	7995

Tabelle 6.2.: Ergebnis der Supervisor-Synthese.

Eine Implementierung der in Tabelle 6.2 dargestellten lokalen Supervisor würde aufgrund der hohen Anzahl an Zuständen und Transitionen zu einem enormen Speicherbedarf auf der SPS führen. Deshalb sind im letzten Schritt der Synthese die Supervisor mit Hilfe des DESTool-Operators *SupReduce* nach [24] auf ein Minimum zu reduzieren. Dem Operator

werden ein lokales System und ein lokaler Supervisor als Parameter übergeben. Daraus berechnet DESTool einen alternativen Supervisor mit einer reduzierten Anzahl an Zuständen und Transitionen. Dieser Operator wurde für die 14 Supervisor sukzessiv ausgeführt. Jedoch führte ein Versuch die Supervisor S_3, S_4 und S_{11} zu reduzieren zum Absturz von DESTool. Dies ist auf die jeweiligen Modellkomplexitäten zurückzuführen, die unter DESTool nicht handhabbar sind. Diese Supervisor konnten aber mittels *Supreduce*-Algorithmus des Entwicklungstools TCT (Version 20170501) reduziert werden. Die reduzierten und nicht reduzierten Supervisor sind mit der jeweils entsprechenden Anzahl an Zuständen und Transitionen in Tabelle 6.3 zusammengefasst.

Supervisor	***n***	***t***	**Reduzierter Supervisor**	***n***	***t***
$E_{X1} = S_1$	15	52	$SupReduce(S_1, G_{X1})$	2	5
$E_{X2} = S_2$	15	52	$SupReduce(S_2, G_{X2})$	2	5
$E_{X3} = S_3$	318	1638	$Supreduce(S_3, G_{X3})$	4	9
$E_{X4} = S_4$	318	1638	$Supreduce(S_4, G_{X4})$	4	9
$E_{X5} = S_5$	63	151	$SupReduce(S_5, G_{X5})$	17	56
$E_{X6} = S_6$	22	51	$SupReduce(S_6, G_{X6})$	4	12
$E_{X7} = S_7$	14	24	$SupReduce(S_7, G_{X7})$	4	9
$E_{X8} = S_8$	18	37	$SupReduce(S_8, G_{X8})$	4	10
$E_{X9} = S_9$	14	24	$SupReduce(S_9, G_{X9})$	4	9
$E_{X10} = S_{10}$	66	270	$SupReduce(S_{10}, G_{X10})$	5	22
$E_{X11} = S_{11}$	450	230	$Supreduce(S_{11}, G_{X11})$	5	17
$E_{X12} = S_{12}$	90	269	$SupReduce(S_{12}, G_{X12})$	4	8
$E_{X13} = S_{13}$	110	502	$SupReduce(S_{13}, G_{X13})$	11	42
$E_{X14} = S_{14}$	171	927	$SupReduce(S_{14}, G_{X14})$	10	72
Gesamt	1684	7995	Gesamt	80	285

Tabelle 6.3.: Ergebnis der Supervisor-Reduzierung.

Tabelle 6.3 ist zu entnehmen, dass die reduzierten Supervisor etwa 95 % kleiner sind als die nicht reduzierten Supervisor.

Durch eine Supervisor-Reduzierung nach [24] reduziert sich die Menge der Steuereingriffe der Supervisor nicht. Jedoch verlieren die Supervisor durch die Reduktion Informationen über die Strecke. Daher müssen zum Ausgleich das Produktsystem und die reduzierten Supervisor gemeinsam implementiert werden (Steuerungsstruktur, vgl. Abbildung 7.1). Dadurch werden insgesamt 122 Zustände und 346 Transitionen verwendet.

7. Realisierung und Inbetriebnahme

Das vorliegende Kapitel teilt sich in drei Unterkapitel auf. In Unterkapitel 7.1 wird die Umsetzung der Generatoren in Programmcode beschrieben. In Unterkapitel 7.2 wird auf die Implementierung der lokal-modularen Steuerung auf die Steuerungshardware eingegangen. Es wird jedoch nur ein kurzer Überblick über die Implementierung gegeben. In Unterkapitel 7.3 werden die Betriebsergebnisse der lokal-modularen Steuerung vorgestellt.

7.1. Umsetzung der Generatoren in SPS-Code

Für die Implementierung der lokal-modularen Steuerung auf einer SPS müssen zuerst die im vorhergehenden Kapitel berechneten Supervisor (reduzierte Supervisor) und das Produktsystem aus Kapitel 5 in einen Programmcode übersetzt werden. Zur Übersetzung wird in dieser Arbeit der in Unterkapitel 2.3.3 vorgestellte Codegenerator ACArrow eingesetzt. Zuvor wurden die 16 Streckenkomponenten in DESTool als Streckenmodelle und die 14 reduzierten Supervisor als Supervisormodelle gekennzeichnet. In ACArrow wurde als Ansatz Lokal-Modular, als Syntax Siemens und als Programmiersprache für die Codegenerierung SCL (Structured Control Language) gewählt. Als Ausgabe erzeugte ACArrow verschiedene S7-SCL-Quelldateien, die die folgenden Funktionen (FC-Bausteine) enthalten:

1. ***Main_SCT()*** ruft alle Unterfunktionen der Codegenerierung auf. Sie wird Taktzyklisch aufgerufen und ausgeführt.
2. ***Init_SCT()*** wird nur einmal zum Programmstart aufgerufen. Sie setzt alle Initialzustände der Generatoren.
3. In ***Read_Input_SCT()*** werden Signale, die eine Rückmeldung geben, über die Flankendetektion eingelesen. Tritt eine Rückmeldung auf, wird das jeweilige Ereignis gesetzt und im selben Zyklus von den Supervisor und dem Produktsystem verarbeitet. Im nächsten Zyklus wird es wieder zurückgesetzt. In dieser Arbeit werden alle Ereignisse bei steigenden Signalflanken gesetzt. Die Rückmeldesignale können auch mehr als einen Zyklus aktiv sein.

4. ***Load_2_Memory_UC_Events_SCT()*** verhindert den Lawineneffekt der nicht steuerbaren Ereignisse.

5. ***Load_2_Memory_C_Events_SCT()*** verhindert den Lawineneffekt der steuerbaren Ereignisse.

6. ***UC_Events_Plant_SCT()*** wertet die nicht steuerbaren Ereignisse des Produktsystems aus. Ein nicht steuerbares Ereignis darf pro Generator des Produktsystems und Zyklus nur einen Zustandsübergang bewirken. Nach dem Auswerten wird das Ereignis in der Abfrage wieder zurückgesetzt und kann keinen weiteren Zustandsübergang mehr bewirken (Lawineneffekt).

7. ***UC_Events_Super_SCT()*** wertet die nicht steuerbaren Ereignisse der Supervisor aus. Ein nicht steuerbares Ereignis darf pro Supervisor nur einen Zustandsübergang bewirken (Lawineneffekt). In dieser Funktion wird vor jedem Supervisor die Funktion *Load_2_Memory_UC_Events_SCT()* einmal aufgerufen.

8. **C_Events_Enable_SCT()** erlaubt oder deaktiviert die steuerbaren Ereignisse vom Supervisor. Dafür werden die Zustände aus verschiedenen Supervisor, in denen das Ereignis erlaubt ist, mit einer Konjunktion verknüpft. Zustände aus demselben Supervisor, die das Ereignis erlauben, werden mit einer Disjunktion verknüpft.

9. ***Choice_SCT()*** ist für das Auswahlproblem zuständig. Ein Auswahlproblem liegt vor, wenn von einem Zustand mehrere steuerbare Ereignisse abgehen. Sie dürfen nicht gleichzeitig stattfinden, da sonst mehrere Zustände aktiviert werden. Das ist nicht erlaubt. In diesem Fall findet entweder eine priorisierte oder zufällige Auswahl statt. Die Art der Auswahl wird bei der Codegenerierung festgelegt. Nach dem Einlesen der DESTool-Projektdatei werden in ACArrow alle Zustände, die ein Auswahlproblem haben, aufgelistet. Es kann dann für jeden dieser Zustände die Art der Auswahl (priorisierte oder zufällige Auswahl) einzeln gewählt werden.

10. In ***C_Events_Plant_SCT()*** erfolgen die Zustandsabfragen mit den durch den Supervisor aktivierten steuerbaren Ereignissen. Ist ein steuerbares Ereignis nicht zuvor durch einen Supervisor deaktiviert worden und in der Strecke möglich, erfolgt der Zustandsübergang im Produktsystem und das steuerbare Ereignis wird gesetzt.

11. In ***C_Events_Super_SCT()*** erfolgen die Zustandsabfragen der Supervisor mit den aus der Strecke gesetzten steuerbaren Ereignissen. Nach einem Zustandsübergang wird das gesetzte Ereignis wieder zurückgesetzt, um einen Lawineneffekt zu vermeiden. Ein steuerbares Ereignis darf pro Supervisor und Zyklus nur ein Zustandsübergang bewirken. Daher wird in dieser Funktion die Funktion *Load_2_Memory_C_Events_SCT()* vor jedem Supervisor einmal aufgerufen, um nach einem Zustandsübergang zurückgesetzte Ereignisse wieder zu setzen. Am Ende der Funktion werden alle steuerbaren

Ereignisse zurückgesetzt. Anschließend wird die Funktionen *Write_Output_SCT()* aufgerufen.

12. ***Write_Output_SCT()*** ist für das Setzen der Ausgänge zuständig. Da in dieser Arbeit alle Ausgänge durch unterlagerte Steuerungen angesteuert werden, kann diese Funktion hier weggelassen werden.

Die oben genannten Funktionen beziehen sich ausschließlich auf die Codegenerierung mit dem DECON9-Ansatz (lokal-modular) und wurden bereits in [6] detailliert beschrieben. Die Quelldateien wurden über externe Quellen in das TIA-Portal importiert und mit anschließendem Übersetzen aus diesen Quelldateien FC-Bausteine erzeugt. Der komplette, automatisch generierte Code ist im Anhang A.2 zu finden.

Des Weiteren stellte ACArrow für die Anlage einige Auswahlprobleme fest. Ein Auswahlproblem wurde bereits in Unterkapitel 6.2.2 und 6.2.9 beschrieben. Es muss ausgewählt werden, ob ein Werkstück mittels Handlingeinheit aus dem Eingangslager eingelastet werden soll oder mittels Roboter aus der Station 60. Diese Auswahl muss im Supervisor S3 Zustand 1 sowie im Supervisor S4 Zustand 1 getroffen werden. In diesem Fall findet die Auswahl immer priorisiert für den Roboter statt. Bei der Codegenerierung wird das entsprechende Ereignis zuerst abgefragt. Für alle anderen Zustände wurde das zufällige Auswahlsystem gewählt, weil diese für die Anlage nicht zutreffen.

7.2. Implementierung der Steuerung

Abbildung 7.1 zeigt die implementierte Steuerungsstruktur. Die unterlagerten Steuerungen sind Schrittkettensteuerungen aus vorhandenen Steuerungen der Anlage und werden hier als Teil der Strecke gesehen. Sie wurden für die Implementierung der lokal-modularen Steuerung erweitert und angepasst. Das Produktsystem und die unterlagerten Steuerungen fungieren als Schnittstellen zwischen der realen Hardware und den lokal-modularen Supervisor.

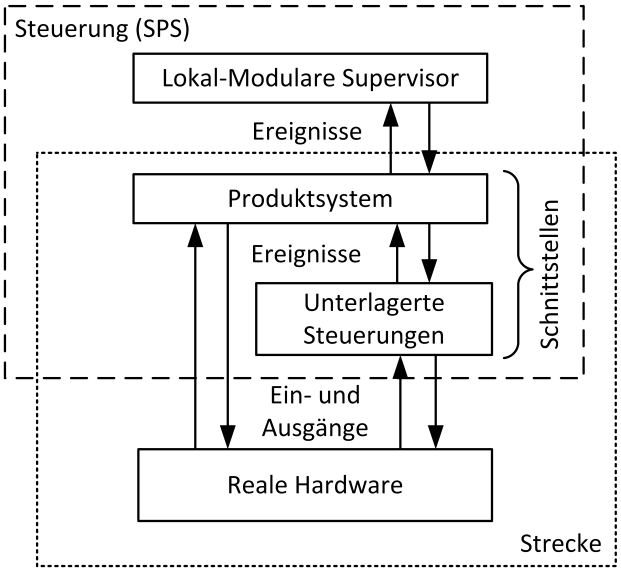


Abbildung 7.1.: Struktur des Steuerungsprogramms.

Ein steuerbares Ereignis wird nur dann erlaubt und in der Strecke ausgeführt, wenn es durch alle Supervisor, die dasselbe Ereignis beinhalten, erlaubt wird und zudem noch in der Strecke möglich ist. Alle nicht erlaubten steuerbaren Ereignisse werden deaktiviert und dürfen in der Strecke nicht ausgeführt werden. Die Supervisor müssen die nicht steuerbaren Ereignisse immer erlauben. Die lokal-modulare Steuerung kann die reale Hardware auch direkt über Ein- und Ausgänge steuern. In den Modellen wurden aber keine Ausgänge verwendet. Diese werden durch unterlagerte Steuerungen angesteuert. Die Eingänge dienen als Rückmeldung von der realen Hardware. Sie werden entweder direkt oder über die Schnittstellen eingelesen und an die Supervisor geleitet.

7.2.1. Schnittstellen

In diesem Unterkapitel werden die Schnittstellen beschrieben. Die Prozesse in den einzelnen Arbeitsstationen werden im Automatikbetrieb von Schrittketten abgearbeitet. Die vorhandenen Steuerungsprogramme wurden nicht nach den SCT-Ansätzen aufgebaut, sondern sind klassische Industrie-Steuerungen. Die lokal-modularen Supervisor sind nicht dafür verantwortlich, eine der erlaubten Steuerungsaktionen auszuführen. Dies ist Aufgabe der unterlagerten Steuerungen. Die Supervisor teilen den unterlagerten Steuerungen lediglich über die Übergabeparameter mit, welche Aktionen in der Strecke ausgeführt werden dürfen und welche nicht. Die Übergabeparameter sind in diesem Fall die steuerbaren Ereignisse beziehungsweise sie werden durch die steuerbaren Ereignissen gesetzt.

Die lokal-modulare Steuerung wurde auf der SPS der Station 10 implementiert (vgl. IO-Controller in Abbildung 3.1). Das vorhandene Steuerungsprogramm besteht hier aus einem Hauptprogramm OB1 und mehrfach aufrufbaren Unterprogrammen, wie Organisationsbausteine, Funktionen und Funktionsbausteine. Der generierte Code benötigt etwa 266 kByte im Ladespeicher. Dies entspricht etwa 40 % der gesamten Ladespeicherauslastung. Des Weiteren wurden an den Steuerungen der Stationen weitere Ein- und Ausgabebereiche (Transferbereiche) projektiert. Über diese Bereiche werden von der SPS der Station 10 (IO-Controller) Übergabeparameter an die anderen Stationen gesendet und Rückmeldungen empfangen.

Die **Rückmeldungen der Komponenten** werden direkt an die Unterfunktion *Read_Input_SCT()* der Codegenerierung weitergeleitet. Bei einer positiven Flanke eines Rückmeldesignals wird das jeweilige Ereignis für einen Zyklus gesetzt und durch das Produktsystem und den Supervisor ausgewertet.

In Abbildung 7.2 ist die Schrittkette für die *Ansteuerung des Palettenstoppers PS.3* dargestellt. Durch das steuerbare Ereignis *STR.TS1.PS.3.PaletteFreigeben* wird die Schrittkette aus dem Supervisor heraus aktiviert. Befindet sich eine Palette über dem Palettensensor *TS1.B.3* am Stopper *PS.3*, wird der Schritt S3 aktiviert. Das Sensorsignal soll zwei Sekunden lang true sein. Ist diese Bedingung erfüllt, so schaltet der Ausgang zum Lösen des Stoppers für 300 ms. Anschließend wird der Ausgang zurückgesetzt und der Stopper wieder gesperrt. Bei fallender Flanke des Sensors *TS1.B.3* wird der Schritt S5 zurückgesetzt und der Schritt S6 gesetzt. In diesem Schritt wird für die Codegenerierung die Rückmeldung *E.TS1.PS.3.PaletteIstFreigegeben* erzeugt. Die Ausführung dieses Schritts dauert genau einen Zyklus an. Abschließend wird die Schrittkette beendet. Die Schrittketten der anderen Stopper sind ähnlich aufgebaut. Die Schrittketten der *PS.1*, *PS.2*, *PS.4* und *PS.6* warten vor einer Palettenfreigabe zusätzlich noch auf die Schreibbestätigung der jeweiligen RFID-Schreib- und Leseeinheit.

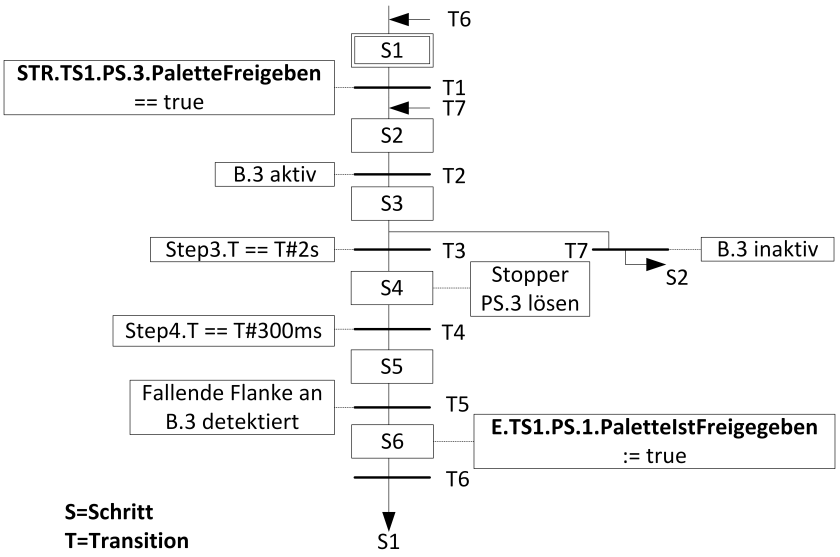


Abbildung 7.2.: Schrittkette zur Steuerung des Palettenstoppers *PS.3*.

Die Schrittkette für die Steuerung der **Robotereinheit *ROB*** und der **Handlingeinheit *HNDL1*** wird von der SPS der Station 30 abgearbeitet. Hier wird aus dem Hauptprogramm OB1 die SPS-Funktion FC50 aufgerufen, welche die Schrittkette für den gesamten Automatisierungsprozess der Station 30 ausführt. Diese Funktion wird Taktzyklisch ausgeführt. Ein Roboterauftrag wird im Steuerungsprogramm durch das Setzen der jeweiligen Merkerbits M301.3 bis M301.6 erlaubt:

- **M301.3:** Roboterauftrag 1,
- **M301.4:** Roboterauftrag 2,
- **M301.5:** Roboterauftrag 3,
- **M301.6:** Roboterauftrag 4.

Nach dem Start eines Roboterauftrags meldet Station 30 an Station 10 das Work-Signal und hält diese auf true, bis der Roboter den aktuellen Auftrag beendet und das Werkstück durch die Handlingeinheit *HNDL1* am Bandende auf der Palette abgelegt wurde. Wie schon im Unterkapitel 3.3.2 erläutert, kommunizieren SPS der Station 30 und Roboter-Controller über digitale Ein- und Ausgänge miteinander. Ein Auftrag wird durch das Setzen der jeweiligen Ausgangsbits von der SPS der Station 30 gestartet:

In dem Programm auf dem Roboter-Controller werden in einer Endlosschleife die Signale DI24, DI25, DI26 und DI27 abgefragt. Sobald die jeweiligen Signale aktiv sind, wird der entsprechende Auftrag gestartet. Dies wird der SPS der Station 30 mitgeteilt. Danach wird

Signale SPS	Signale Roboter	Beschreibung
A5.0	R_DI24 (Port 8)	Starte Auftrag 1
A5.1	R_DI25 (Port 9)	Starte Auftrag 2
A5.2 & A5.3	R_DI26 & R_DI27 (Port 10 & 11)	Starte Auftrag 3
A5.0 & A5.2 & A5.3	R_DI24 & R_DI26 & R_DI27 (Port 8 & 10 & 11)	Starte Auftrag 4

Tabelle 7.1.: Signale SPS30 und Roboter-Controller.

im nachfolgenden Schrittkettenschritt der gesetzte Merker (M301.3 bis M301.6) wieder zurückgesetzt. Die steuerbaren Ereignisse können in der Schrittkette nicht direkt verwendet werden, weil sie nur einen Zyklus lang aktiv sind und im nachfolgenden Zyklus durch die automatische Codegenerierung in der SPS der Station 10 zurückgesetzt werden. Sobald ein Auftrag erledigt ist, wird das Ausgangssignal DO26 des Roboter-Controllers gesetzt. Dies führt in der SPS der Station 30 zum Weiterschalten in den nächsten Schritt, in der die Abarbeitung eines Auftrags mit den Ausgangsbits A104.2 bis A104.5 an die SPS der Station 10 zurückgemeldet wird. Die Rückmeldungen werden dann im generierten Code als nicht steuerbare Ereignisse verwendet.

Die Schnittstellen der anderen Komponenten sind ähnlich aufgebaut. Die Schrittkette für die Steuerung der **Linearachse und Handlingeinheit (*LIHNDL*)** wird von der SPS der Station 20 durch Aufruf der SPS-Funktion FC40 (für den Automatikbetrieb) aus dem Hauptprogramm OB1 ausgeführt. Hier werden weitere Funktionen aufgerufen, die beispielsweise zur Steuerung der Linearachse verwendet werden. Durch das Setzen des Merkerbits M700.1, M700.3. M700.5 oder M700.7 durch die folgenden steuerbaren Ereignisse, werden Aufträge zur Handlingeinheit und zur Linearachse gegeben:

- **M700.1:** Auftrag 1: Bringe Werkstück von Band 1 zu Band 5,
- **M700.3:** Auftrag 2: Bringe Werkstück von Band 2 zu Band 6,
- **M700.5:** Auftrag 3: Bringe Werkstück von Band 7 zu Band 3,
- **M700.7:** Auftrag 4: Bringe Werkstück von Band 7 zu Band 4.

Nach dem Start eines Auftrag wird das Work-Signal für dieser Station auf true gesetzt. Durch die Ausgangsbits A125.0 bis A125.3 meldet die SPS der Station 20 die Abarbeitung des Auftrags an die SPS der Station 10 zurück und setzt das Work-Signal wieder auf false:

- **A125.0:** Auftrag 1 ist fertig,
- **A125.1:** Auftrag 2 ist fertig,
- **A125.2:** Auftrag 3 ist fertig,

- **A125.3:** Auftrag 4 ist fertig.

Die **Initialisierung** der Supervisor findet direkt nach dem Einschalten der Anlage oder durch die steigende Flanke des Starttasters an Station 10 statt. Es wird danach die Funktion Init_SCT() aufgerufen in der alle Merker der Generatoren zurückgesetzt und nur ihre Initialzustände gesetzt werden. Zuvor muss aber sichergestellt werden, dass alle Streckenkomponenten sich in ihrem Initialzustand befinden und alle Stationen gestoppt sind. Befinden sich noch Werkstücke auf der Strecke, müssen diese manuell entfernt werden. Beim Start müssen die Stationen in der Reihenfolge 30, 40, 50, 60, 10 und 20 gestartet werden. In Tabelle 7.2 sind die einzelnen Puffergrößen zusammengefasst. Die Anfangsbestände der einzelnen Puffer und Initialzustände der Supervisor sind ebenfalls aus der Tabelle zu entnehmen.

Puffer	**Puffergröße**	**Initialbestand**	**Supervisor Initialzustand**
BUF1	1	0	S1 Zustand 1
BUF2	1	0	S2 Zustand 1
BUF3	1	0	S3 Zustand 1
BUF4	1	0	S4 Zustand 1
BUF5	4	4	S5 Zustand 9
BUF6	1	0	S6 Zustand 1
BUF7	3	0	S7 Zustand 1
BUF8	1	0	S8 Zustand 1
BUF9	3	3	S9 Zustand 4
BUF10	2	2	S10 Zustand 3
BUF11	1	0	S11 Zustand 1
BUF12	1	0	S12 Zustand 1
BUF13	10	0	S13 Zustand 1
BUF14	9	0	S14 Zustand 1

Tabelle 7.2.: Puffergrößen und Initialbestände.

7.3. Betriebsergebnisse

Abschließend wurde die Implementierung anhand von Testszenarien validiert. Bei der Modellierung der Strecke und beim Entwurf der Spezifikationen wurden alle Beschränkungen der Anlage beachtet. Das bedeutet, alle Puffer wurden korrekt modelliert und die Strecke kann unter keinen Umständen in einen verbotenen, undefinierten oder blockierenden Zustand gelangen.

Die Ergebnisse der Testszenarien zeigten, dass die erstellten Modelle und Spezifikationen für die gegebene Steuerungsaufgabe korrekt sind. Um das Schalten der Supervisor zu visualisieren und die einzelnen Pufferbestände darzustellen, wurde eine HMI-Panel basierte Visualisierung im TIA-Portal erstellt. Abbildung 7.3 zeigt ein Bild der Visualisierung, auf dem die Spezifikationen K_{X6} und K_{X7} dargestellt sind. Die aktiven Zustände sind grün markiert.

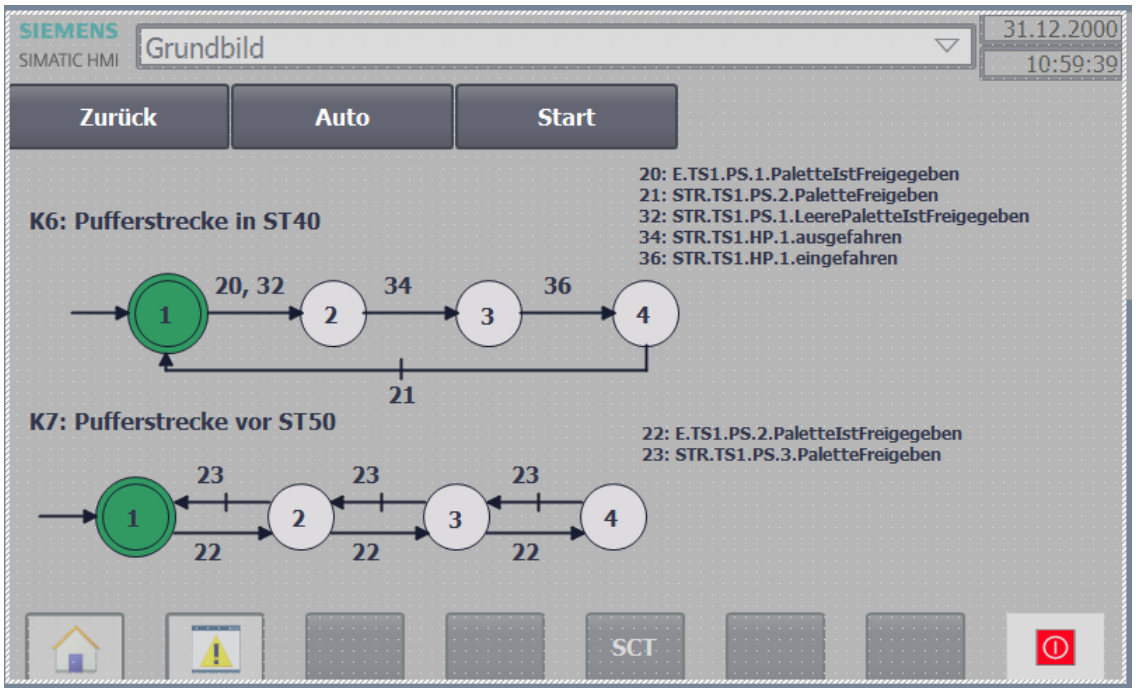


Abbildung 7.3.: Screenshot der Visualisierung.

Das Einlasten von Aufträgen in die Anlage, deren Kapazität auf 10 beschränkt ist, war ein Testszenario. Die Spezifikation K_{X13} (Belastungsschranke) verriegelte ab der kritischen Zahl (>10) die Komponente *LIHNDL*, sodass sie keine neuen Werkstücke mehr aus dem Eingangslager einlasten konnte. Erst wenn ein Werkstück den kritischen Pfad (rot in Abbildung 6.1 und 6.2 dargestellt) verlassen hatte, konnte wieder ein Werkstück aus dem Eingangslager in die Anlage eingelastet werden.

Ein zweites Testszenario war das Sortieren der fertigen Werkstücke auf die entsprechenden Bänder. Nach der Spezifikation K_{X12} soll die Komponente *LIHNLD* die einpoligen Werkstücke auf dem Band 3 und die zweipoligen Werkstücke auf dem Band 4 lagern. Der lokalmodulare Steuerungsentwurf berücksichtigt dieser Anforderung und bei der Implementierung traten keine Probleme auf. Dieses geforderte Verhalten wird also erfüllt.

Zum Schluss wurde das Verhalten der Palettenpuffer getestet. Während der Produktion wurden Paletten manuell gestoppt, um Steuereingriffe der Supervisor zu aktivieren. Der Palettenstopper *PS.1* wird durch die Spezifikation daran gehindert, weitere Paletten freizugeben, sobald sich zwischen den Stoppern *PS.1* und *PS.2* eine Palette befindet. Der Stopper *PS.2* lässt keine Palette durch, sobald sich in dem Puffer BUF7 (Palettenpuffer vor Station 50) 3 Paletten befinden. Eine weitere Spezifikation verhindert die Freigabe von Paletten über den Stopper *PS.6*, wenn genau 4 Paletten am Stopper *PS.1* warten. Des Weiteren greift der entsprechende Supervisor ein und entriegelt den Stopper *PS.1*, wenn sich noch beladene Paletten im Umlauf befinden und die Puffer BUF5 (Palettenpuffer vor Station 30) und BUF10 (Palettenpuffer in Station 60) voll sind. Damit wird eine leere Palette freigegeben. Alle beladenen Paletten konnten somit die Entladeposition vor Station 60 erreichen und entladen werden.

8. Zusammenfassung und Ausblick

8.1. Zusammenfassung

Im Rahmen dieser Masterarbeit wurde für eine flexible Montageanlage eine ereignisdiskrete Puffersteuerung konzipiert und synthetisiert. Die Synthese wurde mit Methoden der Supervisory Control Theory von Ramadge und Wonham durchgeführt. Die Grundlage dieser Synthese ist die Definition der notwendigen Komponentenmodelle und die Formulierung verschiedener Spezifikationen für die Steuerungsaufgabe. Als Beschreibungssprache für das Verhalten einzelner Komponenten und für die Spezifikationen kommen Generatoren zum Einsatz.

Auf der Grundlage der entstandenen Modelle der Anlage und der Spezifikationen der Steuerungsaufgabe wurden die durch Generatoren dargestellten Supervisor, die das gewünschte Verhalten gewährleisten, in einem Programm für eine speicherprogrammierbare Steuerung umgesetzt. Zur Durchführung der Steuerungssynthese wurde die Software DESTool und die Bibliothek libFAUDES verwendet. Die Steuerung wurde mit dem lokal-modularen Ansatz synthetisiert. Für die Umsetzung der mit Hilfe von DESTool berechneten Generatoren in einen Programmcode wurde der Codegenerator ACArrow eingesetzt. ACArrow konnte aus dem DESTool-Projekt die Modelldaten extrahieren. Bei der Durchführung der Synthese und Codegenerierung traten keine Probleme auf.

Die Implementierung der lokal-modulare Steuerung erfolgte auf eine Siemens CPU S7-1512C-1PN. Die mit Hilfe von ACArrow generierten SCL-Quelldateien konnten erfolgreich in das TIA-Portal importiert und übersetzt werden. Die Schnittstellen zwischen der lokal-modularen Steuerung und der vorhandenen Industrie-Steuerung wurden manuell erstellt. Die unterlagerten Schrittkettensteuerungen werden aus den Supervisor heraus aktiviert. Des Weiteren werden aus den unterlagerten Schrittketten Rückmeldungen für die Supervisor erstellt. Die Sensorereignisse werden direkt an die Supervisor weitergeleitet.

Durch verschiedene Testszenarien konnte die Implementierung validiert werden. Die lokal-modulare Steuerung erfüllt als eine übergeordnete Steuerung zu der Industrie-Steuerung die in Unterkapitel 6.1 definierte Steuerungsaufgabe.

8.2. Ausblick

Durch weitere Werkstückpaletten kann der Gesamtdurchsatz der Anlage erhöht werden. Es muss aber ein Kompromiss zwischen Gesamtdurchsatz und Auslastung gefunden werden, weil ab einer bestimmten Anzahl an Paletten weitere Paletten nur noch die Bestände erhöhen, nicht aber den Gesamtdurchsatz der Anlage. Dafür sollten die Durchlaufzeiten der Paletten an den einzelnen Stationen sowie die Auftragsdurchlaufzeiten ermittelt werden. Außerdem kann durch das Einbeziehen der zeitbewerteten Modelle die Steuerung noch verfeinert werden.

In einer weiterführenden Arbeit kann die Puffersteuerung für ein neues Zwischenlager zwischen den Stationen 30 und 60 erweitert werden. Der Roboter sollte nach der Demontage eines defekten Werkstücks, dessen Unterteil auf diesem Lager zwischenlagern, wenn der Puffer in Station 30 voll ist. Die Station 30 kann sich dann später aus diesem Lager bedienen und unabhängig von der Station 20 mit dem Montageprozess beginnen.

Das Entwicklungstool Supremica sollte laut einer Publikation noch effizientere Algorithmen besitzen als die Algorithmen in libFAUDES. In einer weiterführenden Arbeit kann dieses Tool eingesetzt und bewertet werden. Es wird derzeit auch in der Industrie und in vielen Forschungsprojekten eingesetzt. Das Tool besitzt einen integrierten Codegenerator. Gegebenenfalls kann der bereits vorhandene Codegenerator ACArrow für das Einlesen der Projektdateien von Supremica erweitert werden.

Literaturverzeichnis

[1] BERGER, Hans: *Automatisieren mit SIMATIC S7-1500*. Publicis Verlag, 2014. – ISBN 978-3-89578-403-3

[2] CASSANDRAS, Christos G. ; LAFORTUNE, Stephane: *Introduction to Discrete Event Systems*. Springer Science+Business Media, LLC, 2010. – ISBN 978-1-4419-4119-0

[3] DAS, Abhishek ; GUPTA, Minali ; NAGAR, DR. S. K.: Discrete Event Systems Modelling and Supervisory Control. In: *Research India Publications* (2014), Nr. 4, S. 561–570. – ISSN: 2231-1297

[4] FENG, Lei ; WONHAM, W. M.: TCT: A Computation Tool for Supervisory Control Synthesis. In: *Proceedings of the 8th Intertational Workshop on Discrete Event Systems* (2006), S. 388–389

[5] GERÄTEHANDBUCH, Simatic S7-1500 CPU 1512C-1 P.: *Siemens AG*. 2016. – URL `https://cache.industry.siemens.com/dl/files/676/109478676/att_895884/v1/s71500_cpu1512c_1_pn_manual_de-DE_de-DE.pdf`. – Zugriffsdatum: 02.11.2017

[6] GOHERT, Nadine: *Automatische SPS-Codegenerierung für Syntheseverfahren der Supervisor Control Theory*, Hochschule für Angewandte Wissenschaften Hamburg, Masterthesis, 2014. – URL `http://edoc.sub.uni-hamburg.de/haw/volltexte/2015/2887/pdf/MAThesis.pdf`. – Zugriffsdatum: 28.06.2017

[7] IORDACHE, Marian V. ; ANTSAKLIS, Panos J.: Software Tools for the Supervisory Control of Petri Nets Based on Place Invariants. In: *Technical report of the ISIS Group*. – ISIS-2002-003, Department of Electrical Engineering, University of Notre Dame, April 2002.

[8] IORDACHE, Marian V. ; ANTSAKLIS, Panos J.: *SPNBOX*. 2013. – URL `http://www.letu.edu/people/marianiordache/abs/spnbox/`. – Zugriffsdatum: 28.06.2017

[9] LEAL, A. B. ; CURY, D. L. L. da ; S. HOUNSELL, Marcelo da: PLC-Based Implementation of Local Modular Supervisory Control for Manufacturing System. In: *Manufacturing System* (2012), Mai, S. 160–181. – ISBN: 978-953-51-0530-5

[10] LEDUC, Ryan J. ; LAWFORD, M. ; WONHAM, W. M.: Hierarchical Interface-Based Supervisory Control - Part 2: Parallel Case. In: *IEEE Trans. on Automatic Control 50* (2005), Nr. 9, S. 1336–1348

[11] LEDUC, Ryan J. ; LAWFORD, Mark ; DAI, Pengcheng: Hierarchical Interface-Based Supervisory Control of a Flexible Manufacturing System. In: *IEEE Trans. on Automatic Control 14* (2006), Nr. 4, S. 654–668

[12] LEON, Fernando P. ; KIENCKE, Uwe: *Ereignisdiskrete Systeme Modellierung und Steuerung verteilter Systeme*. Oldenbourg Verlag, 2013. – ISBN 978-3-486-73574-1

[13] LUNZE, Jan: *Ereignisdiskrete Systeme: Modellierung und Analyse dynamischer Systeme mit Automaten, Markovketten und Petrinetze*. Oldenbourg Wissenschaftsverlag GmbH, 2012. – ISBN 978-3-486-71885-0

[14] MOODY, John O. ; ANTSAKLIS, Panos J.: *Supervisory Control of Discrete Event Systems Using Petri Nets*. Kluwer Academic Publishers Boston/Dordrecht/London, 1998. – ISBN 0-7923-8199-8

[15] MOOR, T.: *DESTool: Software-Tool for Analysis and Design of Discrete-Event Systems*. 2016. – URL `http://www.rt.eei.uni-erlangen.de/FGdes/destool/`. – Zugriffsdatum: 28.06.2017

[16] MOOR, T.: *libFAUDES: Discrete event systems library*. 2016. – URL `http://www.rt.eei.uni-erlangen.de/FGdes/faudes/`. – Zugriffsdatum: 28.06.2017

[17] MOOR, T. ; SCHMIDT, K. ; PERK, S.: Applied Supervisory Control for a Flexible Manufacturing System. In: *Workshop on Discrete Event Systems (WODES)* (2010), S. 263–268

[18] QUEIROZ, M. H. de ; CURY, J. E. R. ; WONHAM, W. M.: Multitasking Supervisory Control of Discrete-Event Systems. In: *Discrete Event Dynamic Systems* (2005), Nr. 4, S. 375–395. – ISSN: 1573-7594

[19] QUEIROZ, Max H. de ; CURY, J. E. R.: Modular Control of Composed Ssystems. In: *Proceedings of the American Control Conference* (2000)

[20] QUEIROZ, Max H. de ; CURY, J. E. R.: Modular Supervisory Control of Large Scale Discrete Event Systems. In: *R. Boel, G. Stremersch (eds.) Discrete Event Systems: Analysis and Control* (2000), S. 103–110

[21] QUEIROZ, Max H. de ; CURY, J. E. R.: Synthesis and Implementation of Local Modular Supervisory Control for a Manufacturing Cell. In: *Sixth International Workshop on Discrete Event Systems* (2002)

[22] RAMADGE, P. J.: *Supervision of discrete event processes*, Dep. of Electrical and Computer Engineering, University of Toronto, Kanada, Dissertation, 1983

[23] STEP 7 V14 FUNKTIONSHANDBUCH, Simatic P. mit: *Siemens AG*. 2016. – URL https://www.siemens.de/Digital-Factory/download/EventDocs/profinet_step7_v14_function_manual_de-DE_de-DE.pdf. – Zugriffsdatum: 06.11.2017

[24] SU, R. ; WONHAM, W. M.: Supervisor Reduction for Discrete-Event Systems. In: *Discrete Event Dynamic Systems* 14 (2004), Nr. 1, S. 31–53

[25] SUPREMICA: *Supremica - Supervisory Control Tool*. 2017. – URL http://supremica.org. – Zugriffsdatum: 02.11.2017

[26] TCT: *TCT: Software-Tool for Analysis and Design of Discrete-Event Systems*. 2014. – URL http://www.control.utoronto.ca/people/profs/wonham/. – Zugriffsdatum: 28.06.2017

[27] URLAND, Christian: *Ereignisdiskrete Steuerungssynthese und Codegenerierung am Beispiel einer Fertigungszelle*, Hochschule für Angewandte Wissenschaften Hamburg, Masterthesis, 2012. – URL http://edoc.sub.uni-hamburg.de/haw/volltexte/2012/1738/pdf/Masterthesis.pdf. – Zugriffsdatum: 28.06.2017

[28] UZAM, M. ; GELEN, G. ; DALCI, R.: A New Approach for the Ladder Logic Implementation of Ramadge-Wonham Supervisors. In: *XXII International Symposium on Information, Communication and Automation Technologies* (2009), S. 1–7

[29] WELLENREUTHER, Günter ; ZASTROW, Dieter: *Automatisieren mit SPS-Theorie und Praxis*. Vieweg und Teubner, 2014. – ISBN 978-3-8348-1504-0

[30] WENCK, Florian: *Modellbildung, Analyse und Steuerungsentwurf für gekoppelte ereignisdiskrete Systeme*. Shaker Verlag, 2006. – ISBN 978-3-8322-5573-2

[31] WONHAM, W. M.: *Supervisory Control of Discrete-Event Systems*. Dept. of Electrical and Computer Engineering, University of Toronto, 2004

[32] WONHAM, W. M. ; LIN, F.: Decentralized supervisory control of discrete-event systems. In: *Information Sciences 44* (1988), Nr. 3

[33] WONHAM, W. M. ; LIN, F.: Decentralized control and coordination of discrete-event-systems with partial observation. In: *IEEE Transactions on Automatic Control 35* (1990), Nr. 12, S. 1330–1337

A. Anhang

Der Anhang dieser Masterthesis befindet sich auf der beiliegenden DVD, die beim Erst- und Zweitprüfer eingesehen werden kann.

A.1. DESTool Projekt (DVD)

Der Ordner „DESTool“ enthält das DESTool Projekt (Version 0.83) mit allen Streckenkomponenten und Spezifikationen.

A.2. S7-SCL-Quelldateien (DVD)

Der Ordner „ACArrow“ enthält die mit Hilfe des Codegenerators ACArrow generierten S7-SCL-Quelldateien.

A.3. Siemens Steuerung und Visualisierung (DVD)

Der Ordner „Siemens Steuerung und Visualisierung“ enthält das archivierte TIA-Portal Projekt zur Steuerung der Montageanlage und die Visualisierung der einzelnen Supervisor.

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 05.12.2017	
Ort, Datum	Unterschrift