

Requirements and Design Documentation

(RDD)

Version 0.7

ESEP - Praktikum - Wintersemester 2016/2017

Lüdemann	Mona	2212744	mona.luedemann1@haw-hamburg.de
Butkereit	Marvin	2247550	marvin.butkereit@haw-hamburg.de
Schumacher	Wilhelm	2245216	wilhelm.schumacher@haw-hamburg.de
Melkonyan	Anushavan	2243668	anushavan.melkonyan@haw-hamburg.de
Colbow	Marco	2177095	marco.colbow@haw-hamburg.de
Cakir	Mehmet	2195657	mehmet.cakir@haw-hamburg.de

10. Januar 2017

Änderungshistorie:

Version	Author	Datum	Anmerkungen/Änderungen
0.1	Mehmet Cakir	2016-10-18	Kapitel 1-4 und Testkonzept
0.2	Mehmet Cakir	2016-10-26	Korrekturen an Formulierung, Visualisierungen noch nicht festgelegt.
0.3	Mehmet Cakir	2016-11-03	Testtabelle umformatiert. Tests zu Grundfunktionen, HAL-UML, Systemgrenzen, Systemarchitektur und Visualisierungsentscheidung sowie entsprechend kurzen Text hinzugefügt.
0.4	Mehmet Cakir	2016-11-16	Neugliederung der Kapitel 4 und 7, Systemkontakte zusammengeführt, verwendete Werkzeuge ergänzt, Zeitmessung und FSM/HSM eingepflegt, Abbildung 6 zur Zeit erfassung aktualisiert, diverse Umformulierungen.
0.5	Mehmet Cakir	2016-11-17	Aktualisiertes UML-Klassendiagramm der HAL und Tests der Sensorik eingefügt.
0.6	Mehmet Cakir	2017-01-09	Einleitender Text, angepasster Terminplan, Tests und Dokumentation zur seriellen Schnittstelle, Implementierung(ISR, Threads und Dispatcher), Lessons Learned, Anregung zum Praktikum, Korrekturen an Ausdruck/-Formulierung.

Version	Author	Datum	Anmerkungen/Änderungen
0.7	Mehmet Cakir	2017-01-09	Timer in Design umformuliert. Screenshots zu ISR, Threads und Dispatcher geändert. Zwei Unterkapitel aus Testen entfernt. Einleitung umformuliert. Verweis zu Milestones in Teamorganisation und Prozess geändert. Testkonzept für die Abnahme aktualisiert. SW-Architektur/Komponentendiagramm aktualisiert. Testszenarien hinzugefügt.

Inhaltsverzeichnis

Inhaltsverzeichnis	3
1 Einleitung	5
2 Teamorganisation	5
2.1 Verantwortlichkeiten	5
2.2 Absprachen	5
2.3 Repository-Konzept	6
3 Projektmanagement	6
3.1 Prozess	6
3.2 PSP/Zeitplan/Tracking	6
3.3 Qualitätssicherung	7
4 Randbedingungen	7
4.1 Entwicklungsumgebung	7
4.2 Werkzeuge	7
4.3 Sprachen	7
5 Requirements and Use Cases	8
5.1 Stakeholder	8
5.2 Anforderungen	9
5.3 Systemkontext	12
5.3.1 Softwareebene	12
5.3.2 Systemebene	15
5.4 Use Cases	17
6 Design	18
6.1 Systemarchitektur	18
6.1.1 Förderband intern	18
6.1.2 Gesamtsystem	19
6.2 Datenmodellierung	19
6.2.1 HAL	19
6.3 Zeitmessung	21
6.3.1 Vorbedingungen	21
6.3.2 Ausführung	21
6.4 Verhaltensmodellierung	23
7 Implementierung	28
7.1 Serielle Schnittstelle	28
7.2 ISR	31
7.3 Threads(SignalHandler)	33
7.4 Dispatcher	35
8 Testen	36
8.1 HAL der Aktorik	36
8.2 HAL der Sensorik	37
8.3 Serielle Schnittstelle	38
8.4 Testszenarien	39
8.5 Testkonzept für die Abnahme	40

9 Lessons Learned	41
9.1 Pro	41
9.2 Contra	41
10 Anregung zum Praktikum	41
10.1 Positiv	41
10.2 Negativ	42

1 Einleitung

Diese Dokumentation beschreibt für dieses Projekt im Rahmen des ESE Praktika im Wintersemester 2016/2017 sämtliche Beschlüsse, Schritte und Maßnahmen die während des Projekt- bzw. Entwicklungszeitraums getroffen wurden. Das Projekt umfasst die Implementierung von Software zur Ansteuerung von drei baugleichen Förderbändern und die Kommunikation dieser untereinander des Unternehmens Festo, womit eine Werkstück-Sortieranlage realisiert werden soll. Über einen eigenen GEME-Rechner erfolgt die Ansteuerung pro Förderband. Zur Kommunikation der Förderbänder untereinander werden die GEME-Rechner über serielle Schnittstellen gekoppelt.

2 Teamorganisation

Grundsätzlich kann jedes Teammitglied eine Aufgabe seiner Wahl übernehmen. Bei jedem Meeting werden die Aufgaben verteilt, worüber im folgenden Meeting über den Fortschritt diskutiert wird. Falls ein Mitglied seine Aufgabe fertiggestellt hat, übernimmt er eine Neue. Bei Nichteinhaltung des Zeitplans werden entsprechend der Zeitpuffer andere Aufgaben zurückgestellt. Die Aufgaben richten sich nach den festgelegten Milestones im zum Projekt angefertigten Gantt Diagramm. Für die Projektleitung und die Pflege des RDD-Dokuments wurde jeweils eine Person bestimmt, welche im Unterkapitel 2.1 eingesehen werden kann.

2.1 Verantwortlichkeiten

Aufgabe	Zuständige/r	Bemerkung
Projektleitung	Mona	Die Projektleitung überwacht den Projektfortschritt und benachrichtigt insbesondere bei Nichteinhalten des Zeitplans alle Teammitglieder. Außerdem hat die Projektleitung bei Unstimmigkeiten immer das letzte Wort.
RDD-Pflege	Mehmet	Der Zuständige ist für die Gestaltung und für die Vollständigkeit des RDDs verantwortlich. Er kann andere Gruppenmitglieder dazu auftfordern Inhalte für das Dokument zu erarbeiten und ihm bereit zu stellen.
Protokollführung	Alle Teammitglieder	Die Protokollführung wird reihum von Gruppenmitgliedern übernommen. Dabei wird folgende Reihenfolge eingehalten: <i>Mona</i> → <i>Marvin</i> → <i>Marco</i> → <i>Wilhelm</i> → <i>Mehmet</i> → <i>Anushavan</i>

Tabelle 1: Zuteilung von Verantwortlichkeiten

2.2 Absprachen

Zur Kommunikation außerhalb der Praktikumstermine werden die Messengerdienste Slack und WhatsApp verwendet. Unstimmigkeiten, Fragen und Inkonsistenzen können somit interaktiv geklärt bzw. mitgeteilt werden. Es wird erwartet, dass jedes Teammitglied in einem Zeitfenster von 24 Stunden auf eine Nachricht entsprechend mit einer Nachricht antwortet. In folgender Abbildung 1 werden die Termine der Meetings dargestellt:

Terminplan für Meetings				
Oktober	Mi, 05.10. ab 16:00 Uhr	Do, 13.10. ab 12:00 Uhr	Mi, 19.10. ab 16:00 Uhr	Mi, 26.10. ab 16:00 Uhr
November	Do, 03.11. ab 12:00 Uhr	Do, 10.11. ab 12:00 Uhr	Mi, 16.11. ab 16:00 Uhr	Mi, 23.11. ab 16:00 Uhr
Dezember	Do, 01.12. ab 12:00 Uhr	Mi, 07.12. ab 16:00 Uhr	Mi, 14.12. ab 16:00 Uhr	Do, 22.12. ab 12:00 Uhr
Weitere Termine können/müssen je nach Bedarf in der Gruppe vereinbart werden.				

Abbildung 1: Terminplan der Meetings

2.3 Repository-Konzept

Das Projekt wird mit dem Versionskontrollsysteem Git verwaltet. Zentral wurde ein Repository auf GitHub angelegt. Erreichbar ist das Repository unter <https://github.com/mbutkereit/conveyor>. Änderungen werden lokal auf einem Branch vorgenommen, jedoch nicht auf dem Master. Sind die Änderungen erfolgreich abgeschlossen, kann der Master mit dem lokalen Branch zusammengeführt werden. Bevor ein push durchgeführt wird, muss gepullt werden. Nachdem ggf. Mergekonflikte gelöst wurden, kann vom Masterbranch aus auf das Repository gepusht werden.

3 Projektmanagement

Für die Gewährleistung einer guten Teamarbeit, werden in den folgenden Kapiteln erklärt wie die Teammitglieder mit ihren Aufgaben umgehen bzw. wann eine gegenseitige Benachrichtigung über ihren Fortschritt spätestens stattfinden sollte.

3.1 Prozess

Das Projekt wird auf Grundlage der festgelegten Milestones im zum Projekt angefertigten Gantt Diagramm umgesetzt. Für jede Implementierung ist zuvor ein geeignetes, sowie selbsterklärendes bzw. verständliches, aber auch möglichst vollständiges Diagramm anzufertigen. Die Visualisierung sollte vor der Implementierung allen anderen Teammitgliedern vorgestellt werden, um mögliche Verbesserungen einzuholen und ggf. Konflikte früh zu erkennen, sowie sie zu lösen. In der Tabelle 2 sind für die jeweiligen Spezifikationen die festgelegte Modellierung aufgelistet.

Spezifikation	Modellierung
Klassen	UML Diagramm
Verhalten bzw. logische Abläufe	Zustandsautomat
Systemarchitektur	Komponentendiagramm

Tabelle 2: Festgelegte Modellierung zur jeweiligen Spezifikation

3.2 PSP/Zeitplan/Tracking

Zu jedem Praktikumstermin wird erwartet, dass die verteilten Aufgaben bzw. Milestones erfüllt werden. Um dies zu gewährleisten, muss jedes Teammitglied bei Schwierigkeiten die Projektleitung darüber sofort in Kenntnis setzen, damit frühzeitig ausgeholfen werden kann. Dazu wurden Arbeitspakete definiert und als Milestones in einem Gantt-Diagramm festgehalten.

3.3 Qualitätssicherung

Hinsichtlich der Qualitätssicherung, werden die vier Punkte Team, Modellierung, Code und Förderband herangezogen.

1. **Team:** Jedes Teammitglied sollte über seine eigenen Fähigkeiten im Klaren sein und möglichst nur Aufgaben übernehmen, wofür es sich am besten geeignet fühlt. Darauf hinaus muss jedes Teammitglied bei Möglichkeit stets seine Unterstützung anbieten. Bei Problemen oder Überforderung müssen alle anderen Teammitglieder darüber unterrichtet und Aufgaben ggf. neu verteilt werden.
2. **Modellierung:** Vor der Implementierung muss eine geeignete Visualisierung erstellt, anderen Teammitgliedern vorgestellt und diskutiert werden.
3. **Code:** Für den Code werden bekannte Pattern eingesetzt und verständliche sowie übersichtliche Realisierungen angestrebt. Den Maßstab hierfür setzen die Teammitglieder. Treten beim Code Review keine schwerwiegenden Anmerkungen bzw. Verständnisprobleme auf, gilt der Code als verständlich und übersichtlich.
4. **Förderband:** Um hohen Durchsatz sowie Effizienz bei der Aussortierung zu erzielen, werden die Komponenten mit der höchstmöglichen Geschwindigkeit für die jeweilige Situation angetrieben, während die Sicherheit des Bedieners im Vordergrund steht. Dabei werden Fehler- bzw. Ausnahmezustände ggf. durch einfache Signalcodes mithilfe der Ampel dem Bediener mitgeteilt.

4 Randbedingungen

In diesem Kapitel werden die Bedingungen genannt unter denen das Projekt umgesetzt wird und die Mittel, die für die Umsetzung herangezogen werden.

4.1 Entwicklungsumgebung

Die drei Förderbänder werden über drei QNX Systeme gesteuert, die über eine serielle Schnittstelle verbunden sind. Als IDE wird QNX Momentics auf Windows 7 verwendet.

4.2 Werkzeuge

- QNX Momentics IDE 5.0
- Latex(MiKTeX 2.9, Texmaker 4.5)
- Git 2.8.1
- Visual Paradigm 13.2
- Gantt Project 2.8.1
- Microsoft Visio 2016

4.3 Sprachen

Das System wird im C++03 Standard programmiert. Dabei werden vorgegebene Bibliotheken verwendet, welche in folgender Tabelle 3 aufgelistet sind:

Name	Version	Autor
HWaccess.h	Unknown	Prof. Dr. Stephan Pareigis
HAWThread.h	Unknown	Prof. Dr. Stephan Pareigis
Lock.h	0.1	Simon Brummer

Tabelle 3: Verwendete Programmierbibliotheken

5 Requirements and Use Cases

Mithilfe der Requirements werden die Anforderungen an die einzelnen Komponenten des Förderbandes ermittelt. Dabei werden die Interessen der Stakeholder berücksichtigt.

5.1 Stakeholder

Stakeholder	Interessen
Kunde	<ul style="list-style-type: none">- fehlerfreie Umsetzung der Anforderungen- erfolgreiche Beendigung des Projektes
Designer	<ul style="list-style-type: none">- übersichtliches, leicht erweiterbares Design- sorgfältige Dokumentation
Entwickler	<ul style="list-style-type: none">- präzises Design- sinnvolle Kommentare- lesbarer Code
Tester	<ul style="list-style-type: none">- übersichtliches, vollständiges Testkonzept
Bediener (Mitarbeiter, die das Laufband später bedienen sollen)	<ul style="list-style-type: none">- einfache und intuitive Bedienung
Instandhalter	<ul style="list-style-type: none">- robustes System
Andere Mitarbeiter	<ul style="list-style-type: none">- Kenntnis über System und Funktionsweise

Tabelle 4: Stakeholder und ihre Interessen

5.2 Anforderungen

Titel	Beschreibung
Ansteuerung der Ampeln	<p>Die Software soll die Ampeln aller Förderbänder für folgende Fälle entsprechend ansteuern können:</p> <ul style="list-style-type: none"> - grünes Licht bei Normalbetrieb, fehlerfrei - gelbes Licht bei Warnungen - rotes Licht bei Fehler
Ansteuerung der Motoren	<p>Die Motoren der Förderbänder sollen in folgenden Varianten ansteuerbar sein:</p> <ul style="list-style-type: none"> - Rechtslauf langsam/schnell - Linkslauf langsam/schnell - Stopp
Ansteuerung der Weichen	<p>Die Stellungen „offen“ und „geschlossen“ der Weichen müssen angesteuert werden. Außerdem soll beachtet werden, dass die Weichen nur für kurze Zeit die Stellung „offen“ halten, um eine Beschädigung der Weichen zu vermeiden.</p>
Erkennung von Werkstücken	<p>Das erste und zweite Förderband müssen drei Arten von Werkstücken erkennen können:</p> <ul style="list-style-type: none"> - Flache Werkstücke - Werkstücke mit Metalleinsatz (Bohrung liegt nach oben oder unten) - Werkstücke ohne Metalleinsatz (Bohrung liegt nach oben oder unten)
Aussortierung von Werkstücken	<p>Flache Werkstücke und Werkstücke, bei der die Bohrung nach unten liegt, sollen auf dem ersten und zweiten Förderband aussortiert werden.</p>
Reihenfolge der Werkstücke	<p>Am Ende vom zweiten Förderband sollen die Werkstücke vereinzelt in folgender Reihenfolge ankommen:</p> <p>Bohrung oben ohne Metall → Bohrung oben ohne Metall → Bohrung oben mit Metall</p>

Tabelle 5: Anforderungen(Teil 1)

Titel	Beschreibung
Erkennung von Überschlagen der Werkstücke + Aussortierung des betreffenden Werkstücks	Das zweite Förderband muss eine erneute Prüfung des aktuellen Werkstücks durchführen, um es im Falle eines Überschlagens auszusortieren.
Langsamer Transport bei Höhenmessung	Wenn ein Werkstück durch die Höhenmessung transportiert wird, soll das Förderband langsam laufen.
Konsolenausgabe am Ende vom zweiten Förderband	<p>Wenn ein Werkstück das Ende vom zweiten Förderband erreicht, sollen auf der Konsole vom zweiten Förderband folgende Werkstückdaten ausgegeben werden:</p> <ul style="list-style-type: none"> - ID - Typ - Ermittelter Höhen-Messwert vom ersten Förderband - Ermittelter Höhen-Messwert vom zweiten Förderband
Konsolenausgabe am Ende vom dritten Förderband	Am Ende des dritten Förderbandes sollen die Werkstückdaten ankommender Werkstücke auf der Konsole des dritten Förderbandes ausgegeben werden.
Stopp der Förderbänder bei keinen Werkstücken	Alle drei Förderbänder sollen jeweils stoppen, wenn sich kein Werkstück auf ihnen befindet.
Erkennung voller Rutschen	Volle Rutschen müssen mithilfe des Sensors am Rutscheneingang erkannt werden.
Rutschen koordinieren	Ist die Rutsche vom ersten Förderband voll, so soll die Aussortierung über das zweite Förderband erfolgen. Umgekehrt, ist die Rutsche vom zweiten Förderband voll, so soll die Aussortierung bereits auf dem ersten Förderband erfolgen. Da das dritte Förderband nur Werkstücke bündelt und weiterleitet, entfällt das Berücksichtigen der Rutsche des dritten Förderbandes.
Gebündelter Transport von Werkstückgruppen auf drittem Förderband	Die drei sortierten Werkstücke sollen gebündelt (im Abstand von 1,5cm) an das Ende des dritten Förderbandes transportiert werden.
Fehlererfassung: Verschwinden von Werkstücken + Reaktion	Mittels Zeitmessung soll das Verschwinden von Werkstücken erfasst werden. Wenn an einer nachfolgend benachbarten Lichtschranke kein Werkstück erfasst wird und dabei zuviel Zeit vergeht, tritt folgende Reaktion auf: Bandstopp, Fehlermeldung.

Tabelle 6: Anforderungen(Teil 2)

Titel	Beschreibung
Fehlererfassung: Hinzufügen von Werkstücken + Reaktion	Mittels Zeitmessung soll das zu schnelle oder fehlerhafte Hinzufügen von Werkstücken erfasst werden. Wenn zwischen zwei benachbarten Lichtschranken die erwartete Zeit unterschritten wird, in der ein Werkstück erfasst werden müsste, dann tritt folgende Reaktion auf: Bandstop, Fehlermeldung
Fehlererfassung: Beide Rutschen voll + Reaktion	Es soll erkannt werden, wenn beide Rutschen vom ersten und zweiten Förderband voll sind. Reaktion: Bandstop, Fehlermeldung

Tabelle 7: Anforderungen(Teil 3)

5.3 Systemkontext

Zum Systemkontext fallen Anforderungen aus Sicht der Software und des Systems an. Während bei der Software die Ansteuerung der Komponenten eines Förderbandes und die dazugehörigen Schnittstellen anfallen, ist aus Sicht der Systemebene die Kommunikation der Komponenten eines Förderbandes unter sich und die der drei Förderbänder miteinander nötig.

5.3.1 Softwareebene

Im Folgenden sind die Schnittstellen zur Ansteuerung der Komponenten aufgelistet. Die Akteure sind über **Port A** ansteuerbar. Über **Port B** können die Sensoren abgefragt werden.

Port A (Ausgabeport)

Aktor	Methodenname
Motor Rechtslauf	<code>right()</code>
Motor Linkslauf	<code>left()</code>
Motor langsam	<code>slow()</code>
Motor schnell	<code>fast()</code>
Motor Stopp	<code>stop()</code>
Weiche auf/zu	<code>switchOpen()</code> <code>switchClosed()</code>
Ampel Grün	<code>turnGreenOn()</code> <code>turnGreenOff()</code>
Ampel Gelb	<code>turnYellowOn()</code> <code>turnYellowOff()</code>
Ampel Rot	<code>turnRedOn()</code> <code>turnRedOff()</code>

Tabelle 8: API auf Port A(Ausgabeport) - Akteure

Port B (Eingabepoert)

Sensor	Methodenname
Einlauf Werkstück	<code>isItemRunningIn()</code>
Werkstück in Höhenmessung	<code>isItemAltimetry()</code>
Höhenmessung	<code>isItemInAltimetryToleranceRange()</code>
Werkstück in Weiche	<code>isItemSwitch()</code>
Werkstück Metall	<code>isItemMetal()</code>
Weiche offen	<code>isSwitchOpen()</code>
Rutsche voll	<code>isSkidFull()</code>
Auslauf Werkstück	<code>isItemRunningOut()</code>

Tabelle 9: API auf Port B (Eingabepoert) - Sensoren

Port C (Ein-/Ausgabeport)

Aktoren/Sensoren	Methodenname
LED Starttaste	<code>turnLedStartOn()</code> <code>turnLedStartOff()</code>
LED Resettaste	<code>turnLedResetOn()</code> <code>turnLedResetOff()</code>
LED Q1	<code>turnLedQ1On()</code> <code>turnLedQ1Off()</code>
LED Q2	<code>turnLedQ2On()</code> <code>turnLedQ2Off()</code>
Taste Start	<code>isButtonStartPressed()</code>
Taste Stopp	<code>isButtonStopPressed()</code>
Taste Reset	<code>isButtonResetPressed()</code>
Taste E-Stopp	<code>isButtonEStopPressed()</code>

Tabelle 10: API auf Port C (Ein-/Ausgabeport) - Aktoren/Sensoren

5.3.2 Systemebene

Die nachfolgende Abbildung 2 visualisiert die Systemgrenzen einer Förderbandanlage. Dabei sind die dazugehörigen Sensoren und Aktoren abgebildet, durch welche eine Förderbandanlage mit der Umwelt und seiner Nachbarsysteme kommuniziert. Die Tabellen 11 und 12 listen die Aufgaben der Sensoren und Aktoren auf.

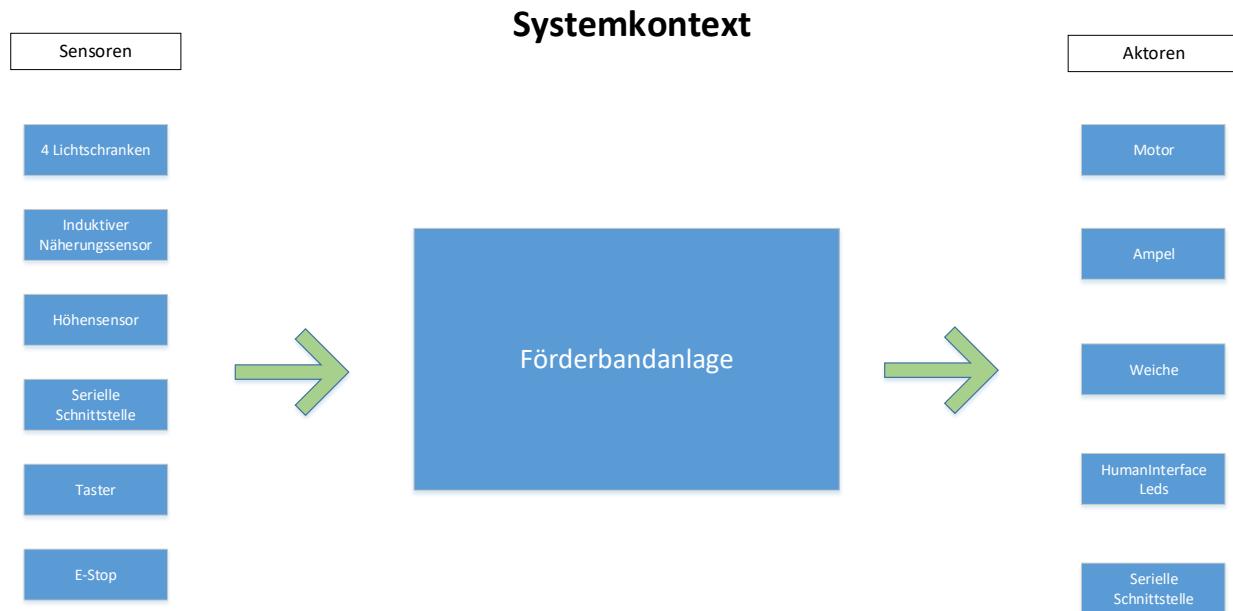


Abbildung 2: Systemgrenzen der Förderbandanlage mit Sensoren und Aktoren

Sensor	Aufgabe
4 Lichtschranken	Erfassen, ob gerade ein Werkstück sich auf der Höhe der jeweiligen Lichtschranke befindet.
Induktiver Näherungssensor	Stellt fest, ob es sich um ein metallisches Werkstück handelt.
Höhensensor	Misst die Höhe des Werkstücks.
Serielle Schnittstelle	Ermöglicht den Empfang und das Senden von Datenpaketen anderer Nachbarsysteme.
Taster	Löst je nach Programmierung entsprechende Aktion aus.
E-Stop	Löst E-Stop Aktion aus.

Tabelle 11: Sensoren und deren Aufgaben

Aktor	Aufgabe
Motor	Treibt das band der Förderbandanlagen an.
Ampel	Signalisiert entsprechend anliegender Ereignisse.
Weiche	Dient zur Sortierung von Werkstücken.
HumanInterface Leds	Signalisieren dem Bediener Sensorereignisse
Serielle Schnittstelle	Ermöglicht das Versenden von Datenpaketen an andere Nachbarsysteme

Tabelle 12: Aktoren und deren Aufgaben

5.4 Use Cases

1. Flache Werkstücke aussortieren

Akteure: Mitarbeiter (legt die Werkstücke auf das Band), Höhenmessung, Weiche

Auslösendes Ereignis: Höhenmessung erkennt das flache Werkstück.

Kurzbeschreibung: Die flachen Werkstücke werden auf dem ersten und zweiten Förderband mit der Höhenmessung erkannt und über die jeweils eigene Weiche aussortiert.

2. Werkstückdaten ausgeben

Akteure: letzte Lichtschranke, Display, zweites Förderband

Auslösendes Ereignis: Die letzte Lichtschranke des zweiten Förderbandes wird durchquert.

Kurzbeschreibung: Wenn ein Werkstück das Ende vom zweiten Förderband erreicht, werden die Werkstückdaten auf dem Display ausgegeben.

3. Ausgabe der Werkstücke auf drittem Förderband in der richtigen Reihenfolge

Akteure: Lichtschranke, Mitarbeiter (nimmt die Werkstücke in Empfang), Weiche

Auslösendes Ereignis: Es sind die drei richtigen Werkstücke auf dem dritten Förderband vorhanden.

Kurzbeschreibung: Auf dem dritten Förderband werden jeweils drei Werkstücke gebündelt in der richtigen Reihenfolge (Bohrung oben ohne Metall → Bohrung oben ohne Metall → Bohrung oben mit Metall) ausgegeben.

6 Design

Im Designkapitel sind Systemarchitektur, Datenmodellierung und Verhaltensmodellierung der Förderbänder enthalten.

6.1 Systemarchitektur

Die Systemarchitektur setzt sich aus den internen Architekturen der drei Förderbänder und der Architektur des Gesamtsystems, welche die Schnittstellen der drei Förderbänder zueinander darstellt, zusammen.

6.1.1 Förderband intern

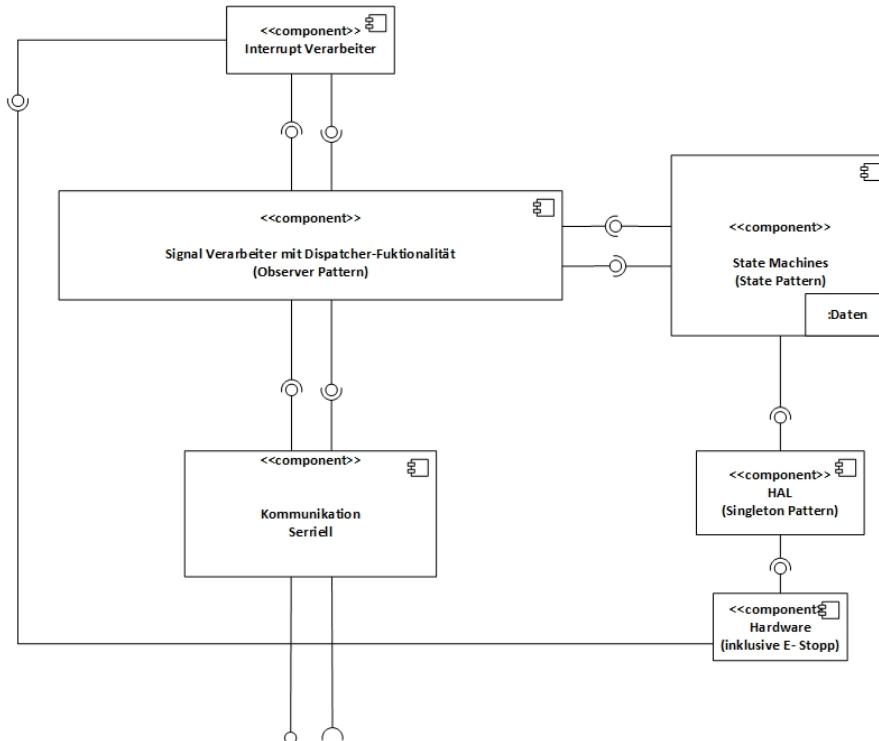


Abbildung 3: Interne Systemarchitektur eines Förderbandes

Komponente	Aufgabe
Interrupt Verarbeiter	Verarbeitet Interrupts aus Timer, Signal Verarbeiter und Hardware.
Timer	Dient zur Zeiterfassung und Überprüfung von verschwundenen oder fehlerhaft hinzugefügten Werkstücken.
Signal Verarbeiter	Verarbeitet Signale aus Interrupt Verarbeiter, Timer, State-machine und Kommunikation seriell.
Statemachine	Steuert den logischen Ablauf.
Kommunikation seriell	Bildet die Schnittstelle zwischen Förderband und Gesamtsystem.
HAL	Hardwareabstraktionsschicht zur Ansteuerung der Komponenten eines Förderbandes.
Hardware	Hardware des Förderbandes

Tabelle 13: Aufgaben der Komponenten eines Förderbandes

6.1.2 Gesamtsystem

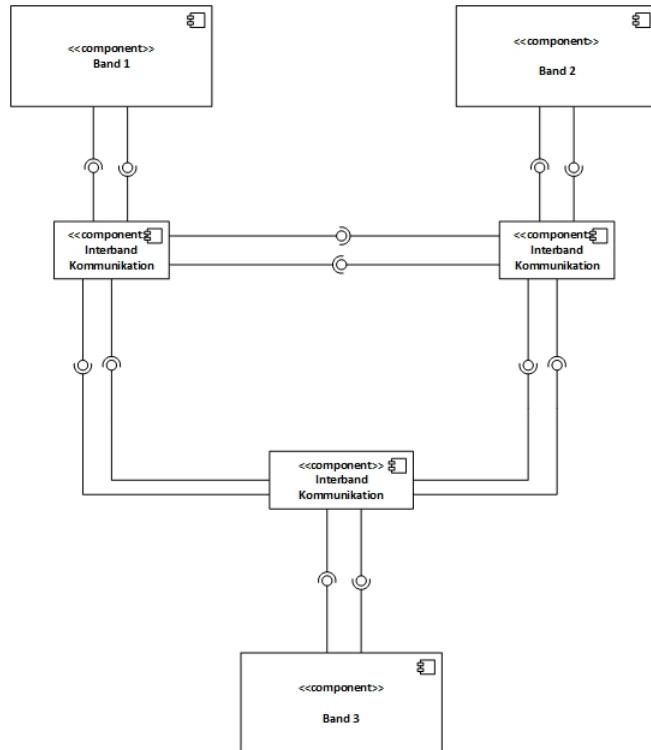


Abbildung 4: Systemarchitektur des Gesamtsystems

Komponente	Aufgabe
Band 1	Erstes Förderband, welches die Sortierung entsprechend der Reihung durchführt.
Band 2	Zweites Förderband, welches die Sortierung entsprechend der Reihung durchführt mit anschließender Konsolenausgabe von Werkstückdaten.
Band 3	Drittes Förderband, welches die Gruppierung der Werkstücke übernimmt und anschließend übergibt.
Serialisierer/Deserialisierer	Serialisiert bzw. deserialisiert Datenpakete zur Kommunikation.
Interband Kommunikation	Empfängt bzw. versendet serialisierte Datenpakete.

Tabelle 14: Aufgaben der Komponenten des Gesamtsystems

6.2 Datenmodellierung

Die Modellierung der Klassen und dessen Methoden sind mithilfe von UML-Diagrammen realisiert.

6.2.1 HAL

Mit der Klasse HAL werden die Hardwarekomponenten eines Förderbandes angesteuert. Dabei wird jede Hardwarekomponente nach dem Singleton-Pattern instanziert. Die nachfolgende Abbildung 5 stellt alle Klassen zur HAL mit ihren Methoden dar.

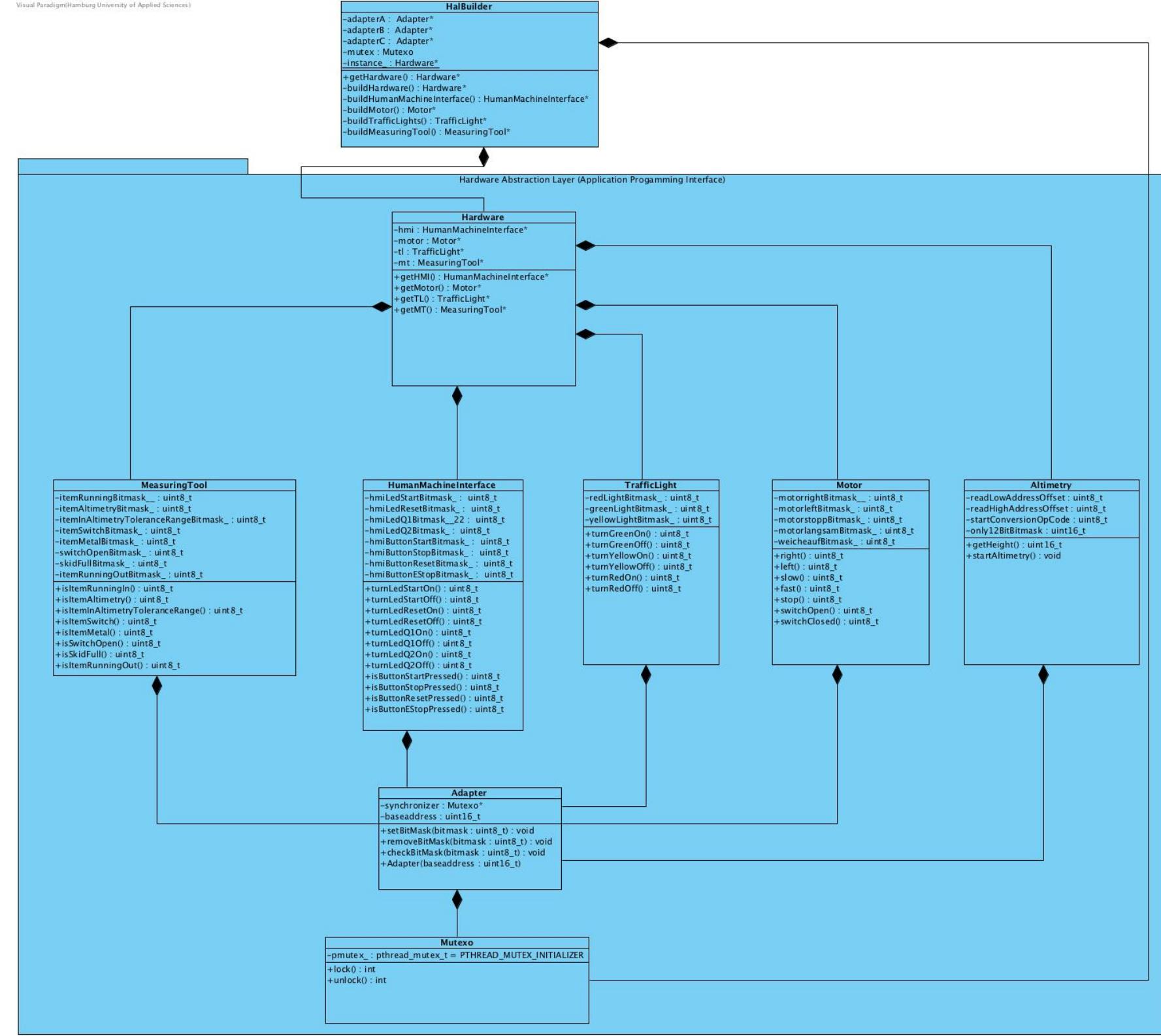


Abbildung 5: UML-Diagramm zur HAL

6.3 Zeitmessung

Auf dem ersten und zweiten Förderband müssen Zeitmessungen durchgeführt werden, um Zeiten zu erfassen, mit denen im Normalbetrieb unsachgemäß hinzugefügte Werkstücke oder das Verschwinden von Werkstücken erkannt werden kann.

6.3.1 Vorbedingungen

Für die Zeitmessung muss die gleiche Implementation der Förderbandlogik ablaufen, die auch im Normalbetrieb für die beiden ersten Förderbänder ablaufen wird, wobei die Höhenmessung ausgelassen wird und das Werkstück bedingungslos durchgelassen wird. Falls in der Routine Geschwindigkeitsänderungen des Bandes vorliegen, müssen diese jedoch miteinbezogen werden. Da ein Werkstück am Anfang des ersten Förderbändes unterschiedlich hinzugefügt werden kann (siehe Abbildung 6) und der Timer eine relativ hohe Zeitauflösung hat, müssen **best** und **worst case** Zeiten ermittelt werden.

6.3.2 Ausführung

Für den **best case** wird das Werkstück beim Hinzufügen von der ersten Lichtschranke aus gesehen am **linken Rand** der Führung angelegt, während beim **worst case** am **rechten Rand** angelegt werden muss. Dabei muss das Werkstück die erste Lichtschranke **LS_A** auch unterbrechen, was auch im Normalbetrieb beim Hinzufügen weiterer Werkstücke beachtet werden muss! Die Zeitmessung beginnt, nachdem das Werkstück die Lichtschranke am Anfang nicht mehr unterbricht. Unterbricht das Werkstück die nachfolgenden Lichtschranken, wird jeweils die Zeit gestoppt. Die nachfolgende Abbildung 6 zeigt wann die Zeiten t_0, t_H, t_W und t_E erfasst werden.

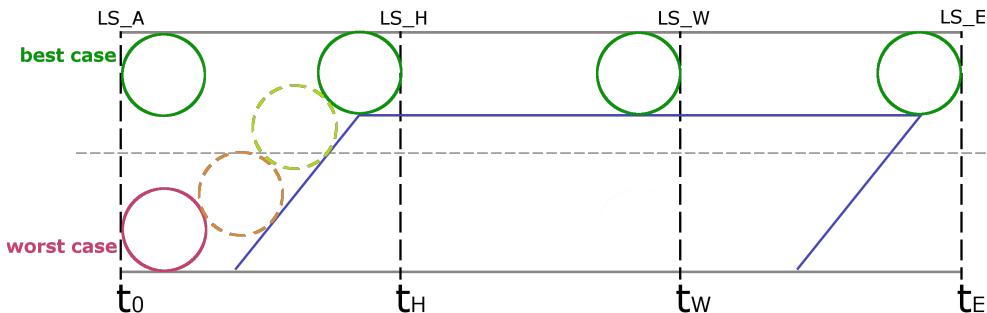


Abbildung 6: Stoppen der Zeit an den Lichtschranken: *LS_A, LS_H, LS_W, LS_E*

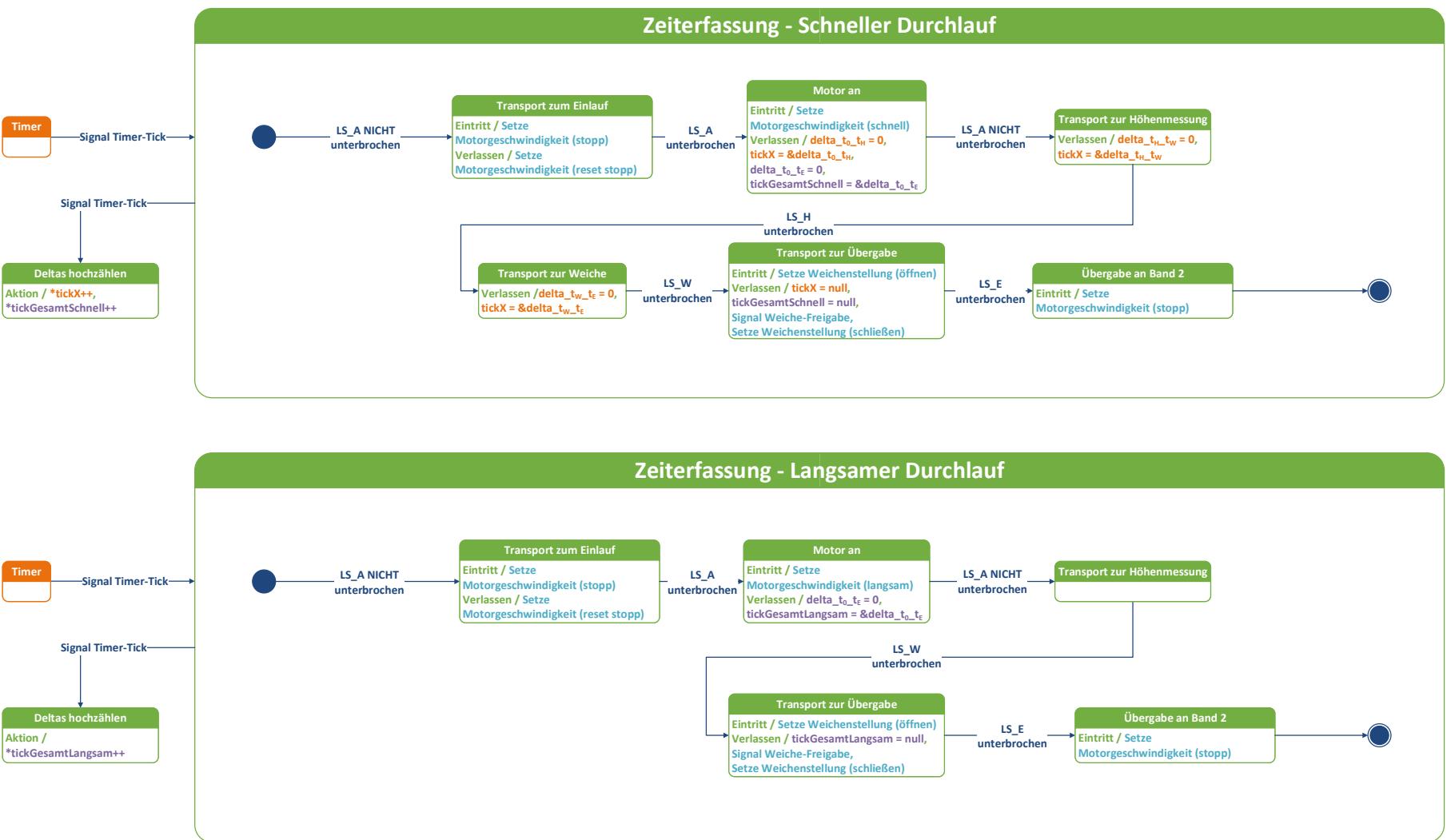
Abkürzung	Beschreibung
LS_A	Lichtschranke-Anfang
LS_H	Lichtschranke-Höhenmessung
LS_W	Lichtschranke-Weiche
LS_E	Lichtschranke-Ende

Tabelle 15: Abkürzende Lichtschrankenbezeichnung

Auf der nachfolgenden Seite sind die endlichen Automaten in Abbildung 7 zur Zeitmessung abgebildet.

Abbildung 7: FSM zur Zeitmessung für schnellen und langsamen Bandlauf

22



Version	0.1	Autor:	Mahmet Cakir	Datum	13.11.2016	Hochschule für Angewandte Wissenschaften Hamburg	Statemachine Zeiterfassung
0.2			Mona Lüdemann	13.12.2016			
0.3			Mona Lüdemann	05.01.2017			
...			
...			

Embedded System Engineering Datei: Statemachines.vsd Blatt: 5

6.4 Verhaltensmodellierung

Das Verhalten der drei Förderbänder ist mithilfe von hierarchischen Zustandsautomaten realisiert. Die Abbildungen 8, 9 und 10 auf den folgenden Seiten bilden die Förderbänder 1, 2 und 3 ab. Anschließend sind in Abbildung 11 die Zustandsautomaten zu den Förderbandkomponenten und dem Timer sowie zum Timeout zu sehen. Die Förderbandkomponenten Motor und Weiche sowie der Timer sind für jedes Förderband auch in Software nur **einmal** vorhanden. Der Timeout Automat wird jedoch für jeden einzelnen Puck einmal erstellt und verwendet.

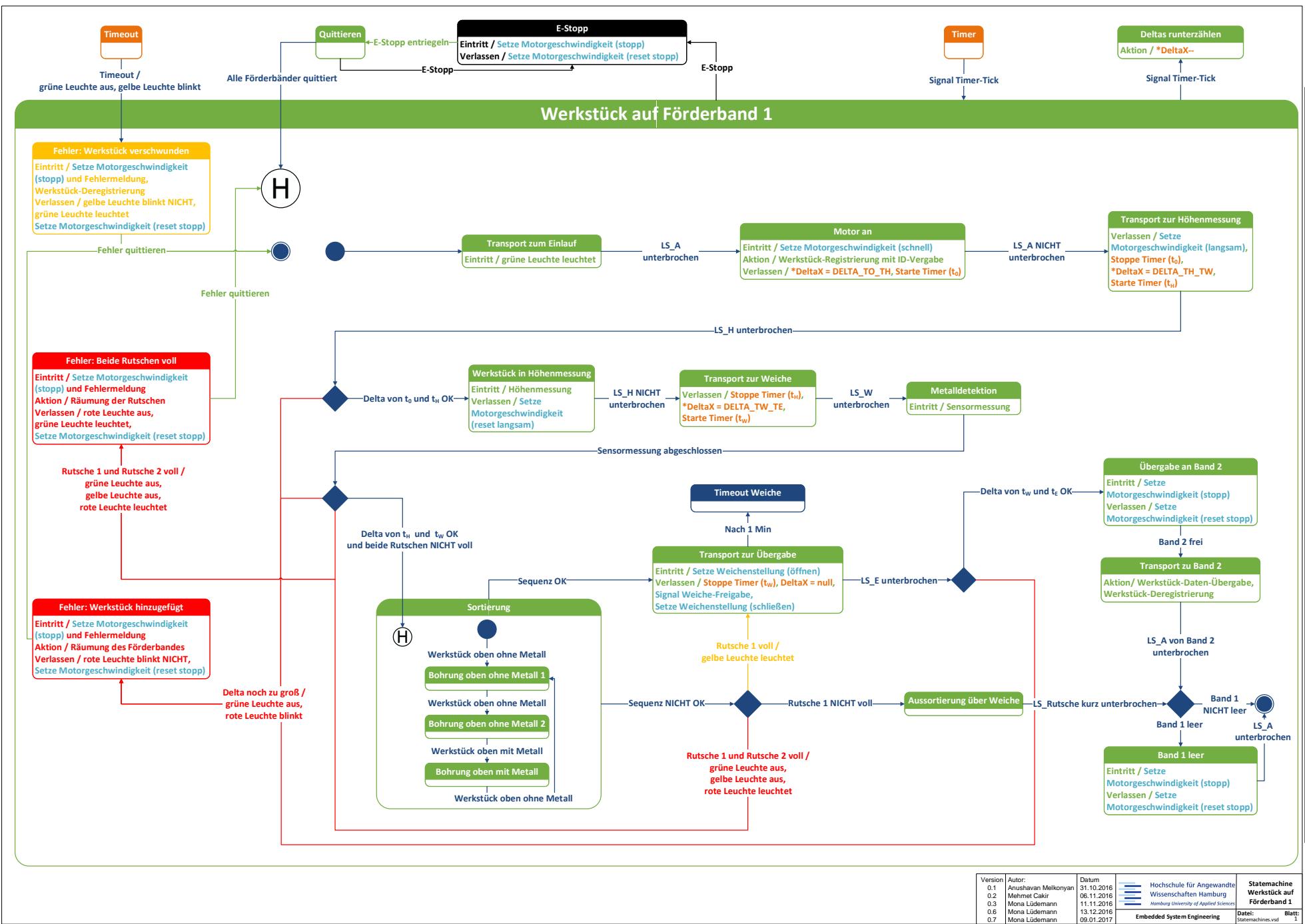
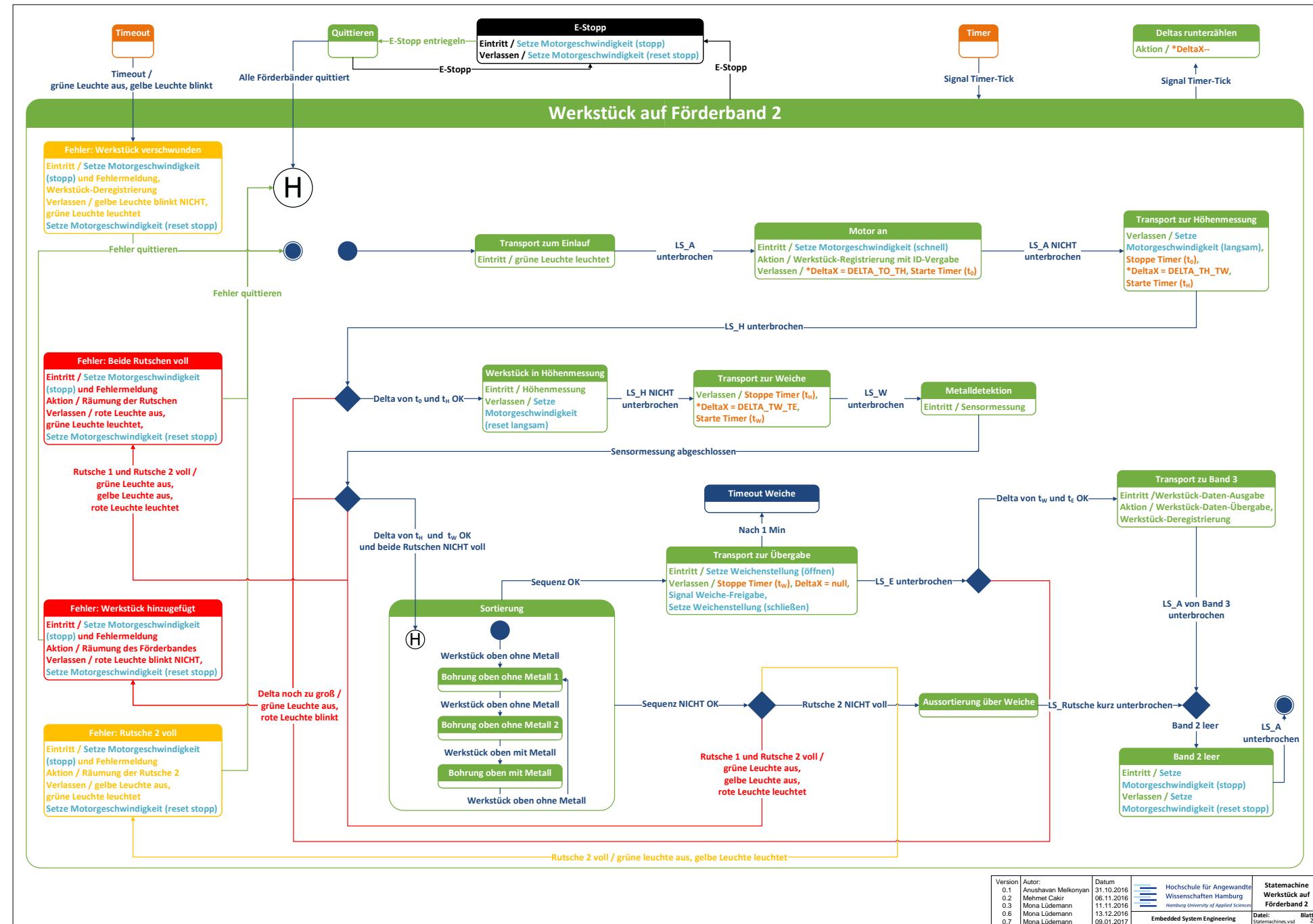
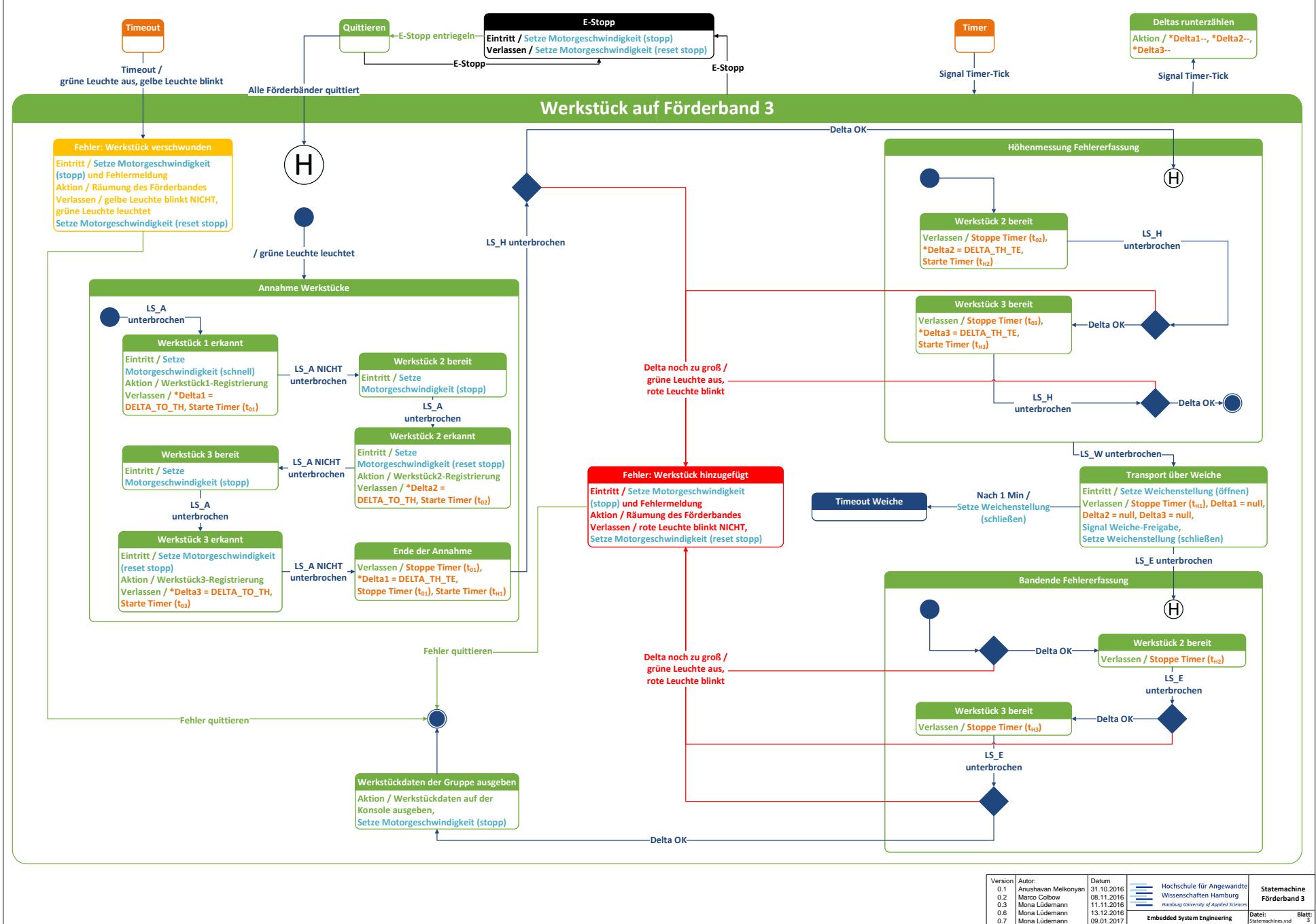


Abbildung 8: HSM zu Förderband 1

Abbildung 9: HSM zu Förderband 2



Version	Autor:	Datum	Hochschule für Angewandte Wissenschaften Hamburg	Statemachine Werkstück auf Förderband 2
0.1	Anushavan Melkonyan	31.10.2016		
0.2	Mehmet Cakir	06.11.2016		
0.3	Mona Lüdemann	11.11.2016		
0.6	Mona Lüdemann	13.12.2016		
0.7	Mona Lüdemann	09.01.2017		
			Embedded System Engineering	Datei: Blatt: 2 statemachines.vsd



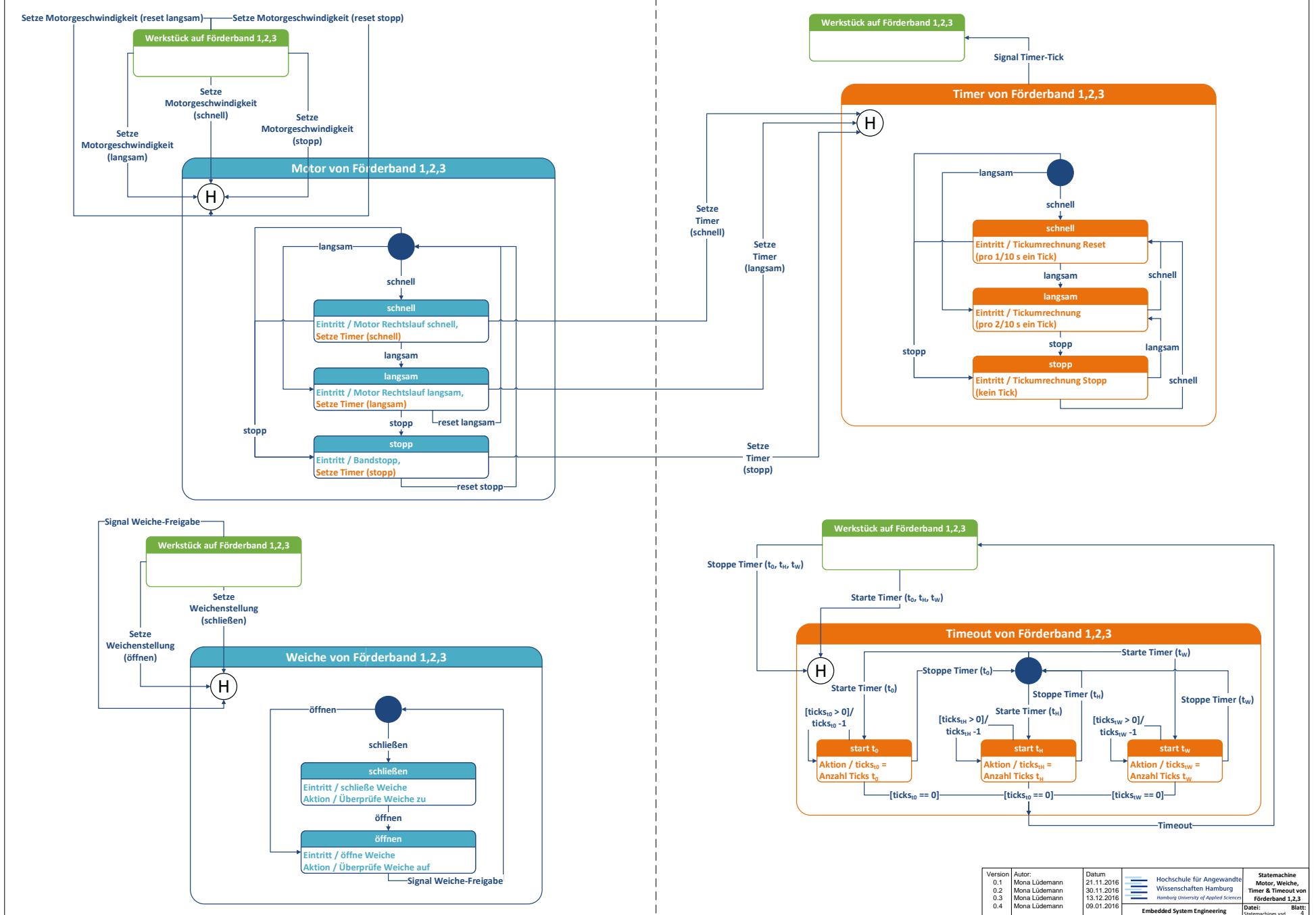
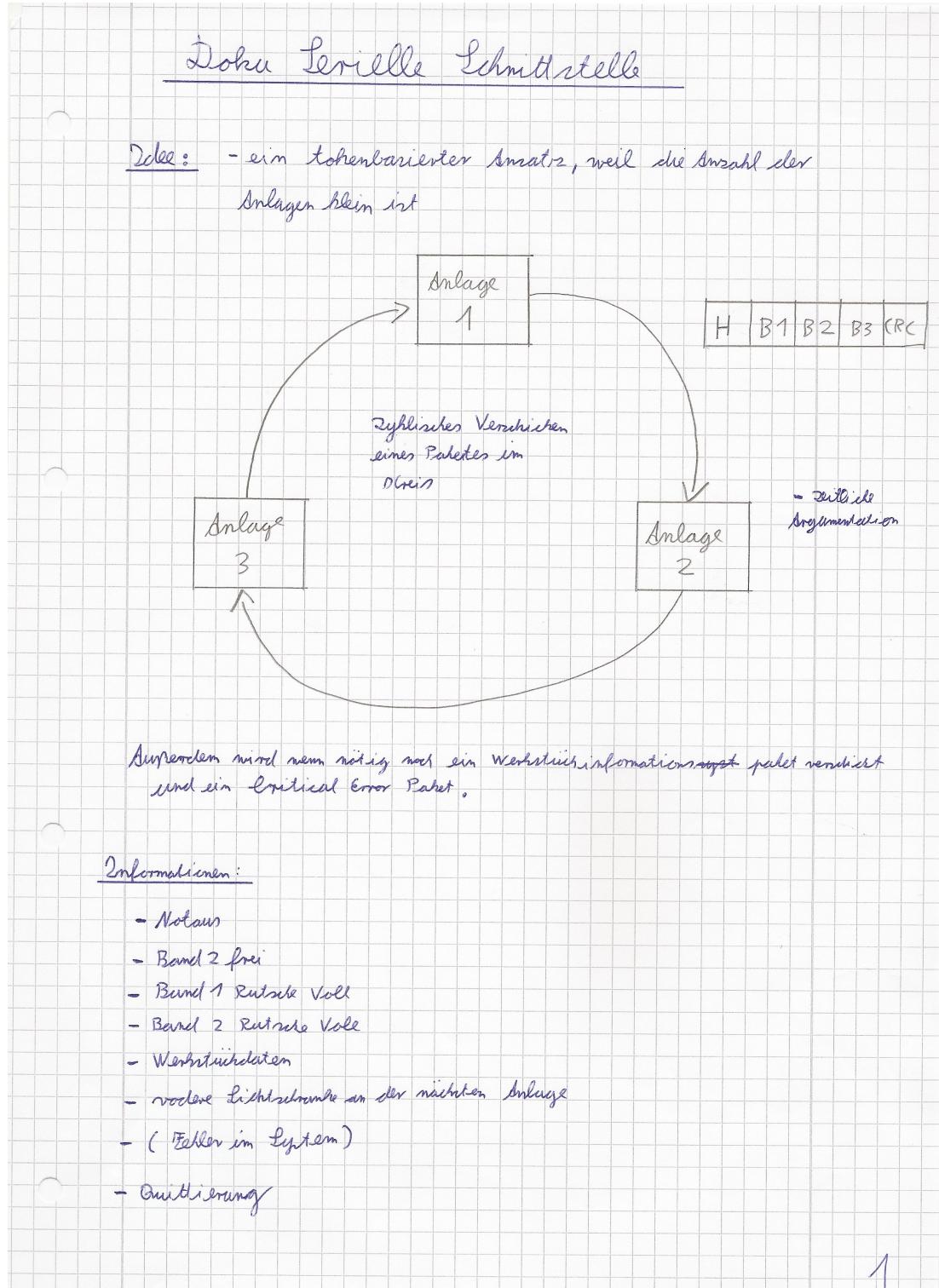


Abbildung 11: HSM zu Motor, Weiche, Timer und Timeout

7 Implementierung

Für die Implementierung der Software, welche die Logik der Werkstück-Sortieranlage realisiert, ist das Reagieren auf Änderungen an Sensoren sowie die Kommunikation unter den Förderbändern essentiell.

7.1 Serielle Schnittstelle



Pakettypen:

INFO Paket:

1 Byte	Typ	0 1
1 Byte	Version	0 1
1 Byte	INFO A	Bit 0: Ratsche Band 1 Voll Bit 1: Erton Bit 2: Anzahlierung Bit 3: Reserved Bit 4: L B interrupted

1 Byte	INFO B	Bit 0: Ratsche Band 2 Voll Bit 1: Erton Bit 2: Anzahlierung Bit 3: Band 2 frei Bit 4: L B interrupted
--------	--------	---

1 Byte	INFO C	Bit 0: reserved Bit 1: Erton Bit 2: Anzahlierung Bit 3: reserved Bit 4: L B interrupted
--------	--------	---

=> zählt alle 20 ms ein Paket 5 Byte
(19200 band rate)

L=> noch genug Bandbreite für die anderen Pakete vorhanden

Wertstückpaket:

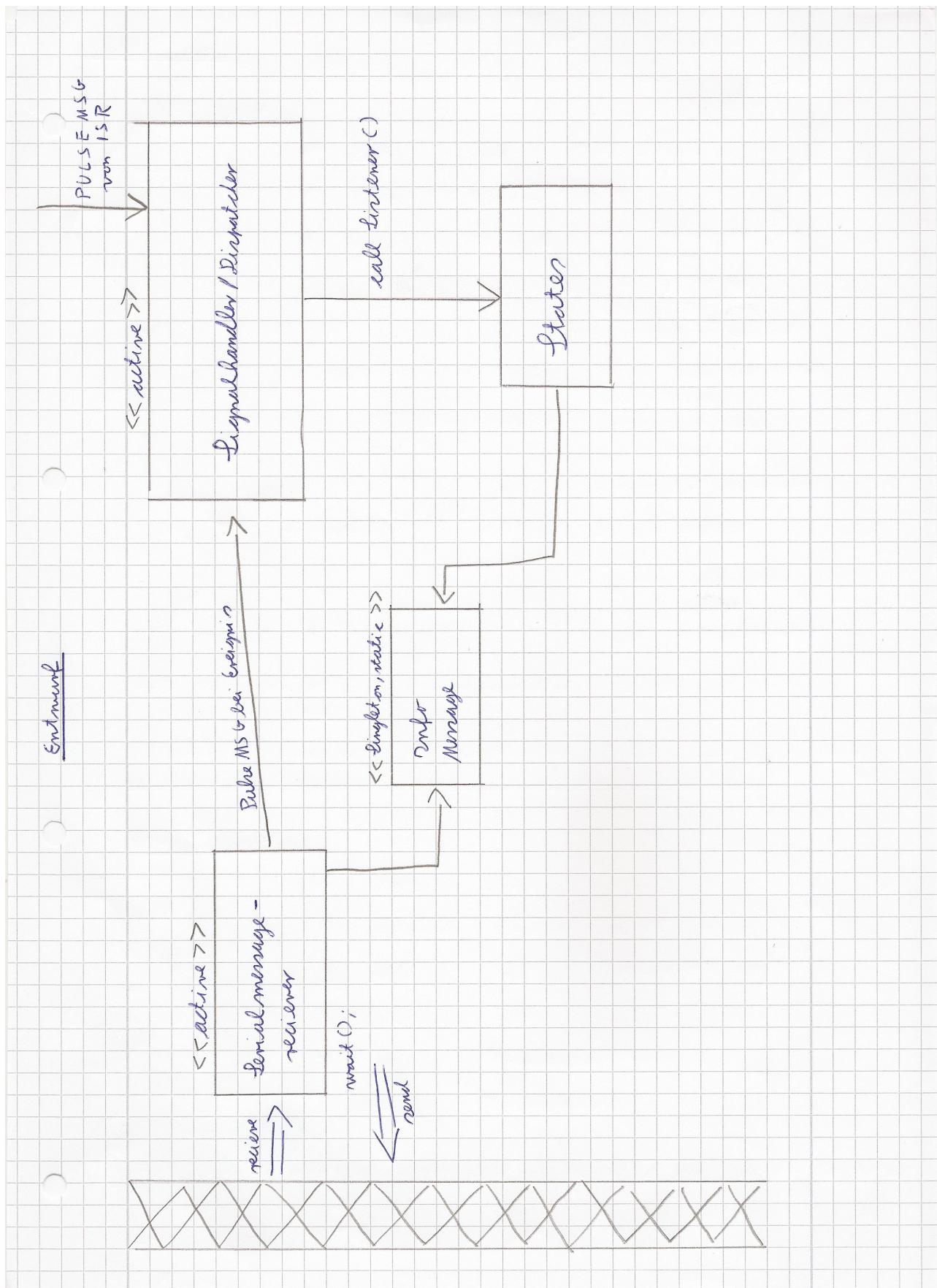
1 Byte	Typ	0 2
1 Byte	Version	0 1
2 Byte	ID	(0 - 65536)
1 Byte	Typ	
2 Byte	Datenmenge 1	
2 Byte	Datenmenge 2	

=> 9 Byte wird geschickt wenn ein Wertstück am Ende des Bandes ist
(letzte Litzstruktur)

=> wird nie weitergeschickt

1 -> 2 2 -> 3

2



7.2 ISR

Das Programm benutzt zwei Interrupt-Service-Routinen um externe Events auszuwerten. Die eine ISR ist dabei für die Interrupts von der Analogkarte zuständig und die andere ISR für die Interrupts von der Digitalkarte. Um die beiden Interrupt-Service-Routinen in das Programm einzubinden, gibt es die Methode `registerISR`. Diese Methode aktiviert einerseits die Interrupts der beiden Karten durch das Setzen der richtigen Bits an einer bestimmten Adresse und bindet die ISRs außerdem an die entsprechenden IRQs durch den Aufruf von `InterruptAttach`.

```

148 void registerISR(void) {
149
150     if (ThreadCtl(_NTO_TCTL_IO_PRIV, 0) == -1) {
151         exit (EXIT_FAILURE);
152     }
153     if ((isrtChannel_ = ChannelCreate(0)) == -1) {
154         exit (EXIT_FAILURE);
155     }
156
157     if ((isrtConnection_ = ConnectAttach(0, 0, isrtChannel_, 0, 0)) == -1) {
158         exit (EXIT_FAILURE);
159     }
160
161     out8(INTERRUPT_RESET_DIO, 0);
162
163     // Einstellen der Interrupts fuer die DIO.
164     out8(0x30B, 0b11111001);
165
166     SIGEV_PULSE_INIT(&isrtEvent_, isrtConnection_, SIGEV_PULSE_PRIO_INHERIT, 0,
167                      0);
168     coid = InterruptAttach(11, ISR_DIO, &isrtEvent_, sizeof(isrtEvent_), 0);
169     if (coid == -1) {
170         exit (EXIT_FAILURE);
171     }
172
173     out8(INTERRUPT_RESET_AIO, 0);
174
175     // Aktivieren der AIO.
176     out8(0x321, 0b10000100);
177
178     SIGEV_PULSE_INIT(&isrtEvent_2, isrtConnection_, SIGEV_PULSE_PRIO_INHERIT, 0,
179                      0);
180     coid2 = InterruptAttach(14, ISR_AIO, &isrtEvent_2, sizeof(isrtEvent_2), 0);
181     if (coid2 == -1) {
182         exit (EXIT_FAILURE);
183     }
184 }
```

Abbildung 12: `registerISR` Methode aus `InterruptHandler.cpp`.

Die ISR von der Digitalkarte läuft so ab, dass zuerst überprüft wird, ob der Interrupt von Port B oder C kommt. Wenn dies der Fall ist, dann wird der Interrupt anschließend zurückgesetzt und der Kernel schickt eine Pulse Message mit einem Code an den Signal-Handler. Der Code enthält die Information welcher Sensor den Interrupt ausgelöst hat.

```
3 const struct sigevent* ISR_DIO(void* arg, int id) {
4     struct sigevent* event = (struct sigevent*) arg;
5     int code = 0;
6     uint8_t interruptregister = in8(0x30F);
7
8     // Ist der Interrupt von Port b oder c
9     if ((interruptregister & 0xA) != 0) {
10
11         // Interrupt Reset
12         out8(INTERRUPT_RESET_DIO, 0);
13
14         uint8_t portb = in8(0x301);
15         uint8_t portc = in8(0x302);
16
17         // Hilfs Variable um Positive und Negative Interrupts ab zu fangen.
18         static char interruptflags = 0;
19
20         // Lichtschranken Interrupts.
21
22         // Lichtschranke vorne
23         if (((portb & 0x1) == 0)
24             && (interruptflags & IS_RUNNING_IN_STATE) == 0) {
25             code = code + LIGHT_BARRIER_BEGIN_INTERRUPTED;
26             interruptflags = interruptflags | IS_RUNNING_IN_STATE;
27         }
28
29         // Lichtschranke vorne nicht mehr unterbrochen
30         if (((portb & 0x1) != 0)
31             && (interruptflags & IS_RUNNING_IN_STATE) != 0) {
32             code = code + LIGHT_BARRIER_BEGIN_NOT_INTERRUPTED;
33             interruptflags = interruptflags - IS_RUNNING_IN_STATE;
34         }
```

Abbildung 13: Ausschnitt aus InterruptHandler.cpp zur Digitalkarte.

Die ISR der Analogkarte ist für die Höhenmessung verantwortlich und läuft nach dem gleichen Prinzip wie die andere ISR ab. Es wird auch eine Pulse Message vom Kernel an den SignalHandler geschickt. Zusätzlich gibt noch eine Methode um die ISRs abzumelden.

7.3 Threads(SignalHandler)

Der SignalHandlerthread empfängt alle eingehenden Pulse von der ISR(Kernel) und dem Thread von der seriellen Schnittstelle. Anschließend wird überprüft, ob ein Ereignis vorliegt und wenn ja wird es bei allen Listenern für dieses Event ausgeführt. Listener werden hinzugefügt, wenn die vordere Lichtschranke durchbrochen wird und Listener werden entfernt, wenn die Automaten den Endzustand erreicht haben. Abhängig davon welche von den drei verschiedenen Anlagen das Band ist, wird jeweils ein anderer Automat erstellt und als Listener für ein Event registriert.

```

25 void SignalHandlerThread::execute(void*) {
26     struct _pulse pulse;
27     int globalerID_Zaehler = 0;
28     std::vector<Puck> puckvector;
29     int skidcounter = 0;
30     Dispatcher* disp = new Dispatcher();
31     std::vector<ContextI*> contextContainer;
32     InfoMessage* message = InfoMessage::getInfoMessage();
33     if (ThreadCtl(_NTO_TCTL_IO_PRIV, 0) == -1) {
34         LOG_DEBUG << "SignalHandlerThread kann keine rechte bekommen.";
35         exit(EXIT_FAILURE);
36     }
37     do {
38         if (MsgReceivePulse(isrtChannel_, &pulse, sizeof(pulse), NULL) == -1) {
39             LOG_DEBUG << "Fehler beim Signal Handler Message Recieve.";
40             exit(EXIT_FAILURE);
41         }
42         short pulsecode = pulse.code;
43         if (pulsecode == 0xE) {
44             int code = pulse.value.sival_int;
45             if (code & ESTOP) {
46                 disp->callListeners(ESTOPSIGNAL);
47             }
48
49             if (code & LIGHT_BARRIER_BEGIN_INTERRUPTED) {
50 #if defined BAND && BAND == 1
51                 ContextI* context= new ContextTopFSM1(globalerID_Zaehler++, &puckvector, &skidcounter);
52 #endif

```

Abbildung 14: Ausschnitt aus SignalHandlerThread.cpp. Bei Unterbrechung der vorderen Lichtschranke werden immer neue Automaten erstellt.

```
176     //Checkt ob ein Automat den Endzustand erreicht und wenn dies so ist,  
177     //dann wird der Automat gelöscht.  
178     for (uint8_t i = 0; i < contextContainer.size(); i++) {  
179         if (contextContainer[i]->isContextimEndzustand()) {  
180             disp->remListeners(contextContainer[i], LBEGININTERRUPTED);  
181             disp->remListeners(contextContainer[i], LBEGINNOTINTERRUPTED);  
182             disp->remListeners(contextContainer[i], LBENDINTERRUPTED);  
183             disp->remListeners(contextContainer[i], LBENDNOTINTERRUPTED);  
184             disp->remListeners(contextContainer[i], LBALTIMETRYINTERRUPTED);  
185             disp->remListeners(contextContainer[i], LBALTIMETRYNOTINTERRUPTED);  
186             disp->remListeners(contextContainer[i], LBSKIDINTERRUPTED);  
187             disp->remListeners(contextContainer[i], LBSKIDNOTINTERRUPTED);  
188             disp->remListeners(contextContainer[i], LBSWITCHINTERRUPTED);  
189             disp->remListeners(contextContainer[i], LBSWITCHNOTINTERRUPTED);  
190             disp->remListeners(contextContainer[i], RESETSIGNAL);  
191             disp->remListeners(contextContainer[i], STARTSIGNAL);  
192             disp->remListeners(contextContainer[i], ESTOPSIGNAL);  
193             disp->remListeners(contextContainer[i], STOPSIGNAL);  
194             disp->remListeners(contextContainer[i], ALTEMETRYCOMPLETE);  
195             disp->remListeners(contextContainer[i], TIMINTR);  
196  
197  
198 #if defined BAND && BAND == 1  
199  
200         ContextI *contextpointer = (ContextI*) contextContainer[i];  
201         contextContainer.erase(contextContainer.begin() + i);  
202 #endif
```

Abbildung 15: Ausschnitt aus SignalHandlerThread.cpp. Ist ein Automat im Endzustand, wird er gelöscht.

7.4 Dispatcher

Der Dispatcher wendet das Observer Pattern an. Es gibt eine Methode um Listener für ein Event zu registrieren und eine Methode um Listener wieder von einem bestimmten Event abzumelden. Des Weiteren wird immer die Methode callListener aufgerufen, wenn eines der Events stattgefunden hat. Der Listener kann nun entscheiden wie und ob er auf das Event reagieren möchte. Die Listener für die Events sind Klassen(Automaten) die von einem Interface geerbt haben.

```
31a Dispatcher::~Dispatcher() {
32 }
33
34a void Dispatcher::addListener(ContextI *listener, EVENTS event) {
35     // Add Listener to be called on a specific Event
36     listeners_[event].push_back(listener);
37 }
38
39a void Dispatcher::remListeners(ContextI *listener, EVENTS event) {
40     // Remove Listener from a specific Event
41     for (unsigned i = 0; i < listeners_[event].size(); ++i) {
42         if (listeners_[event][i] == listener){
43             listeners_[event].erase(listeners_[event].begin() + i);
44         }
45     }
46 }
47
48a void Dispatcher::callListeners(EVENTS event) {
49     // Call for every registered Listener
50     // the Method that corresponds with event.
51     for (unsigned i = 0; i < listeners_[event].size(); ++i) {
52         (listeners_[event][i]->*methods[event])();
53     }
54 }
```

Abbildung 16: Die drei wichtigsten Methoden aus Dispatcher.cpp.

8 Testen

Das Testen der Software findet nach Fertigstellung von Milestones statt, welche die Ansteuerung der Förderbänder sowie deren Kommunikation untereinander enthält. Die Tests reichen von essentiellen Ansteuerungstests einzelner Komponenten der Förderbänder bis hin zur Ablaufsteuerung über alle drei Bänder. Für die Abnahme werden definierte Tests durchgeführt.

8.1 HAL der Aktorik

ID	Funktion	Test erfolgreich?	Anmerkung
1	Rote Lampe an.	Ja	
2	Rote Lampe aus.	Ja	
3	Gelbe Lampe an.	Ja	
4	Gelbe Lampe aus.	Ja	
5	Grüne Lampe an.	Ja	
6	Grüne Lampe aus.	Ja	
7	Motor langsam.	Ja	
8	Motor schnell.	Ja	
9	Motor links.	Ja	
10	Motor rechts.	Ja	
11	Motor stoppen.	Ja	
12	Weiche auf.	Ja	
13	Weiche zu.	Ja	
14	Led Start an.	Ja	
15	Led Start aus.	Ja	
16	Led Reset an.	Ja	
17	Led Reset aus.	Ja	
18	Led Q1 an.	Ja	
19	Led Q1 aus.	Ja	
20	Led Q2 an.	Ja	
21	Led Q2 aus.	Ja	

Tabelle 16: Testauswertung zur HAL der Aktorik

8.2 HAL der Sensorik

ID	Funktion	Test erfolgreich?	Anmerkung
1	Lichtschranke Anfang unterbrochen.	Ja	
2	Lichtschranke Anfang nicht mehr unterbrochen.	Ja	
3	Lichtschranke Höhenmessung unterbrochen	Ja	
4	Lichtschranke Höhenmessung nicht mehr unterbrochen	Ja	
5	Lichtschranke Weiche unterbrochen	Ja	
6	Lichtschranke Weiche nicht mehr unterbrochen	Ja	
7	Lichtschranke Ende unterbrochen	Ja	
8	Lichtschranke Ende nicht mehr unterbrochen	Ja	
9	Lichtschranke Rutsche unterbrochen	Ja	
10	Lichtschranke Rutsche nicht mehr unterbrochen	Ja	
11	E-Stopp betätigt	Ja	
12	Stopp betätigt	Ja	
13	Start betätigt	Ja	
14	Reset betätigt	Ja	
15	Korrekte Höhenmessung	Ja	
16	Metalldetektion	Ja	

Tabelle 17: Testauswertung zur HAL der Sensorik

8.3 Serielle Schnittstelle

Basierend auf unseren Schnittstellen der seriellen Schnittstelle haben wir uns verschiedene Testszenarien überlegt, um die Funktionsfähigkeit und Korrektheit der Schnittstelle sicherzustellen.

ID	Aktion	Erfolgreich?	Datum
01	Kommen Nachrichten zyklisch (alle 50 ms) an der Anlage an?	Ja	16.12.2016
02	Funktionieren die Band2 Methoden: <code>InfoMessage::istBand2Frei()</code> <code>InfoMessage::setBand2Frei()</code> <code>InfoMessage::setBand2Leer()</code>	Ja	16.12.2016
03	Funktionieren die Rutsche1 Methoden: <code>InfoMessage::istBand1RutscheVoll()</code> <code>InfoMessage::setBand1RutscheVoll()</code> <code>InfoMessage::setBand1RutscheLeer()</code>	Ja	16.12.2016
04	Funktionieren die Rutsche2 Methoden: <code>InfoMessage::istBand2RutscheVoll()</code> <code>InfoMessage::setBand2RutscheVoll()</code> <code>InfoMessage::setBand2RutscheLeer()</code>	Ja	16.12.2016
05	Kommt das Ereignis SignalNextLBInterrupted an der richtigen Anlage an: <code>InfoMessage::setLBinterruptedBit()</code>	Ja	16.12.2016
06	Kommt das Ereignis EStop überall an: <code>InfoMessage::setESTOP()</code> <code>InfoMessage::removeESTOP()</code>	Ja	16.12.2016
07	Funktioniert die Quittierung nicht, solange Estop gedrückt ist: <code>InfoMessage::wurdeUeberallQuitert()</code> <code>InfoMessage::setQuittierung()</code>	Ja	16.12.2016
08	Funktioniert die Quittierung: <code>InfoMessage::wurdeUeberallQuitert()</code> <code>InfoMessage::setQuittierung()</code>	Ja	16.12.2016
09	Funktioniert die Werkstückinformationsübergabe: <code>WorkpieceMessage::send(uint16_t hohehnmesswert_1, uint16_t hohehnmesswert_2, uint8_t typ, uint16_t id)</code> <code>struct workpiece_package_without_ch</code> <code>WorkpieceMessage::getWorkpieceInfo()</code>	Ja	16.12.2016

Tabelle 18: Testauswertung zur seriellen Schnittstelle

8.4 Testszenarien

1. Flaches Werkstück → Höhenmessung → Band_1 sortiert aus → grüne Leuchte an
[Rutsche_1 = 1]
2. Werkstück Bohrung unten → Höhenmessung → Band_1 sortiert aus → grüne Leuchte an
[Rutsche_1 = 2]
3. Werkstück Bohrung oben mit Metall → Höhenmessung → Metalldetektion → Band_1 sortiert aus → grüne Leuchte an
[Rutsche_1 = 3]
4. Werkstück verschwindet → Timeout → Band stoppt → gelbe Leuchte blinkt
5. Werkstück Bohrung oben ohne Metall → Metalldetektion → Band_1.OK → Höhenmessung → Metalldetektion → Band_2.OK → Band_3.wartet → grüne Leuchte an
[Band_3 = 1]
6. Werkstück Bohrung oben ohne Metall → Höhenmessung → Metalldetektion → Band_1.OK → Werkstück_überschlagen → Höhenmessung → Band_2 sortiert aus → grüne Leuchte an
[Rutsche_2 = 1]
7. Werkstück Bohrung oben mit Metall → Höhenmessung → Metalldetektion → Band_1 sortiert aus → grüne Leuchte an
[Rutsche_1 = 4] → gelbe Leuchte an
8. Flaches Werkstück → Band_1.RutscheVoll → Höhenmessung → Band_2 sortiert aus → grüne Leuchte an
[Rutsche_2 = 2]
9. Werkstück hinzufügen → Timeout → Band stoppt → rote Leuchte blinkt
10. Werkstück Bohrung oben ohne Metall → Metalldetektion → Band_1.OK → Höhenmessung → Metalldetektion → Band_2.OK → Band_3.wartet → grüne Leuchte an
[Band_3 = 2]
11. Werkstück Bohrung unten → Band_1.RutscheVoll → Höhenmessung → Band_2 sortiert aus → grüne Leuchte an
[Rutsche_2 = 3]
12. Werkstück Bohrung oben mit Metall → Höhenmessung → Metalldetektion → Band_1.OK → Höhenmessung → Metalldetektion → Band_2.OK → Band_3.wartet → Band_3 Transport und Ausgabe der 3er Gruppe → grüne Leuchte an
[Band_3 = 3]
13. Werkstück Bohrung oben ohne Metall → Höhenmessung → Metalldetektion → Band_1.OK → Werkstück_überschlagen → Höhenmessung → Band_2 sortiert aus → grüne Leuchte an
[Rutsche_2 = 4]
14. Rutsche_1 und Rutsche_2 voll → Alle Förderbänder anhalten → rote Leuchte an

8.5 Testkonzept für die Abnahme

ID	Funktion	Erfolgreich?	Datum
1	Erkennung der Werkstücke am Anfang jedes Förderbandes.		
2	Flache Werkstücke werden auf dem ersten und zweiten Förderband aussortiert.		
3	Werkstücke mit der Bohrung nach unten werden auf dem ersten und zweiten Förderband aussortiert.		
4	Bei voller Rutsche des ersten Förderbandes wird eine Fehlermeldung ausgegeben und die gelbe Leuchte leuchtet.		
5	Bei voller Rutsche des zweiten Förderbandes wird eine Fehlermeldung auf der eigenen Konsole ausgegeben. Das zweite Förderband stoppt und die gelbe Leuchte leuchtet.		
6	Bei voller Rutsche vom ersten und zweiten Förderband, stoppen das erste und zweite Förderband. Die rote Leuchte vom ersten und zweiten Förderband leuchtet.		
7	Die Förderbänder stoppen jeweils, wenn sich auf ihnen keinen Werkstück befindet.		
8	Die Förderbänder stoppen jeweils, wenn ein Werkstück auf ihnen verschwindet. Entsprechend wird eine Fehlermeldung auf der eigenen Konsole ausgegeben und eigene gelbe Leuchte blinkt.		
9	Die Förderbänder stoppen jeweils, wenn ein Werkstück mitten auf dem eigenen Laufband hinzugefügt wird. Entsprechend wird eine Fehlermeldung auf der eigenen Konsole ausgegeben und die eigene rote Leuchte blinkt.		
10	Das erste Förderband sortiert die gewünschte Reihenfolge bis die eigene Rutsche voll ist.		
11	Am Ende vom zweiten Förderband entsteht die gewünschte Reihenfolge.		
12	Am Ende vom zweiten Förderband werden die Werkstückdaten auf der eigenen Konsole ausgegeben.		
13	Am Ende vom dritten Förderband werden die Werkstückdaten als dreier Gruppe auf der eigenen Konsole ausgegeben.		
14	Das dritte Förderband transportiert die Werkstücke erst dann bis zum Ende des Laufbandes, wenn die dreier Gruppe vollständig ist.		
15	Wenn der E-Stopp an einem der drei Förderbänder gedrückt wird, halten alle Förderbänder an.		
16	Erst wenn alle Förderbänder nach einem Fehlerzustand quittiert wurden, kann die Anlage wieder gestartet werden.		

Tabelle 19: Testauswertung der Abnahmetests(Teil 1)

ID	Funktion	Erfolgreich?	Datum
17	Am Ende vom dritten Förderband werden die Werkstückdaten auf der Konsole als 3er Gruppe ausgegeben.		
18	Förderband 3 transportiert die Werkstücke erst dann bis zum Ende des Bandes wenn die 3er Gruppe vollständig ist.		

Tabelle 20: Testauswertung der Abnahmetests(Teil 2)

9 Lessons Learned

Während des Praktikums konnten Erkenntnisse gesammelt werden, mit welcher Arbeitsweise und Planung das Team produktiv gewesen ist. Andererseits ist durch Fehlverhalten deutlich geworden, dass die Effizienz des Teams darunter leiden musste. Im Folgenden sind einige Punkte zu Pro und Contra aufgezählt, welche während eines Teammeetings zusammengetragen wurden.

9.1 Pro

- Termine eingehalten und die Zeit gut eingeteilt
- Durch Rollenzuweisungen konnten Kräfte optimal verteilt werden.
- Faire Verteilung der Aufgaben
- gute Planung im Voraus
- Konsequentes Anwenden der Visualisierungen führte zu wartbarem Code.
- Gutes Teamwork
- Aus Fehlern gelernt

9.2 Contra

- Zu späte Absprache über konkrete Implementierung
- Spätes Zusammenführen der Anlagen und Testen
- Späte Absprachen an den Stellen, wo zwei Teams thematisch zusammengetroffen sind (Implementation der FSMs / Dispatcher / HAL)

10 Anregung zum Praktikum

Die folgenden Auflistungen zeigen Punkte auf, welche uns gut gefallen haben und welche uns negativ aufgefallen sind.

10.1 Positiv

- Tutorium war hilfreich
- Mangel an Kabeln wurden zügig von Enrico behoben
- Enrico war hilfsbereit und stets erreichbar
- Herr Korf war sehr hilfreich

10.2 Negativ

- Stattfinden von Tutorium der Mechatroniker im gleichen Raum, da wir den Raum verlassen mussten und kein anderer Raum die Anlagen zu Verfügung stellt.
- Räume zu voll daher keine Anlagen frei gewesen
- Bei drei Anlagen war es immer schwer alle Förderbänder und Kabel parat zu haben, weshalb die Planung verändert/verzögert werden musste.