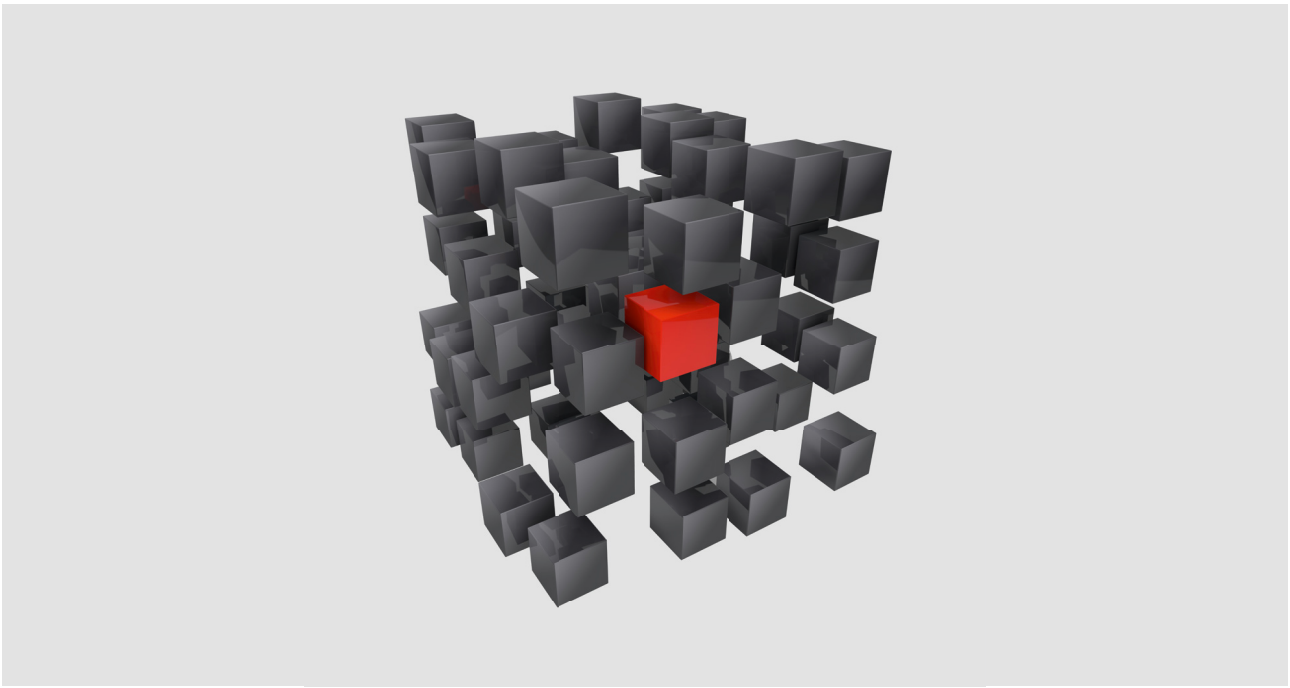
Entwicklerhandbuch

LJU Automatisierungstechnik GmbH



Richtlinien zur Entwicklung von Software für die Kommunikation zwischen Anlagensteuerung (SPS) und LJU-Master Control Unit (MCU)

iDM-System

A Member of GRENZEBACH

LJU Automatisierungstechnik GmbH
Am Schlahn 1
14476 Potsdam
Deutschland
Telefon: +49 33201 414-0
Telefax: +49 33201 414-19
E-Mail: info@lju-grenzebach.com
Internet: www.ljuonline.de
Originaldokument
VSH_0038, 1, de_DE

Rechtliche Hinweise

VORWORT

Die Grenzebach Unternehmensgruppe stellt Maschinen und Anlagen her.

Wir haben den Anspruch, neben der Erfüllung geltender Normen und Anforderungen dem „Stand der Technik“ so zu entsprechen, dass sich der Schutz von Mensch, Maschine und Umwelt in Maschinen und Anlagen entsprechend optimal realisieren lässt.

Rechtliche Vorgaben werden kontinuierlich erweitert und verschärft. Der Anwender erhält zu Maschinen und Anlagen Nutzerinformationen. Alle an der Anlage beschäftigen Personen müssen Nutzerinformationen begreifen und gewissenhaft anwenden. Der Betreiber muss seiner Sorgfaltspflicht nachkommen und dabei sicherstellen, dass alle an der Anlage beschäftigen Personen die Nutzerinformationen verinnerlicht haben und einhalten.

Warnhinweis-konzept

Diese Bedienungsanleitung enthält Hinweise, die Sie zu Ihrer persönlichen Sicherheit sowie zur Vermeidung von Sachschäden beachten müssen. Die Hinweise zu Ihrer persönlichen Sicherheit sind durch ein Warndreieck hervorgehoben, Hinweise zu alleinigen Sachschäden stehen ohne Warndreieck.

Beim Auftreten mehrerer Gefährdungsstufen wird immer der Warnhinweis zur jeweils höchsten Stufe verwendet. Wenn in einem Warnhinweis mit dem Warndreieck vor Personenschäden gewarnt wird, dann kann im selben Warnhinweis zusätzlich eine Warnung vor Sachschäden angefügt sein.

Signalwörter

Warnhinweise werden nach Gefährdungsstufe wie folgt dargestellt:

	Signalwort	Bedeutung und Folgen bei Missachtung
	GEFAHR!	**Unmittelbar drohende Gefahr** ▸ Das Signalwort bezeichnet eine Gefährdung mit einem hohen Risikograd, die, wenn Sie nicht vermieden wird, den Tod oder eine schwere Verletzung zur Folge hat.
	WARNUNG!	**Möglicherweise drohende Gefahr** ▸ Das Signalwort bezeichnet eine Gefährdung mit einem mittleren Risikograd, die, wenn Sie nicht vermieden wird, den Tod oder eine schwere Verletzung zur Folge hat.
	VORSICHT!	**Möglicherweise drohende Gefahr** ▸ Das Signalwort bezeichnet eine Gefährdung mit einem niedrigen Risikograd, die, wenn Sie nicht vermieden wird, geringfügige oder mäßige Verletzung zur Folge haben kann.
	HINWEIS!	**Mögliche Sachschäden** ▸ Beschädigung der Anlage bzw. Komponenten oder angrenzender Anlagenteile bzw. Komponenten.
	HINWEIS!	**Mögliche Gefahren für die Umwelt** ▸ Mögliche Schäden für die Umwelt.

Gefahrensymbole

Sind Warnhinweise der Kategorien **GEFAHR!** und **WARNUNG!** inhaltsbezogen, werden sie mit spezifischen Gefahrensymbolen versehen.

GEFAHR!

Gefährliche elektrische Spannung

Beschreibung

Handlungsanweisung

WARNUNG!

Heiße Oberfläche

Beschreibung

Handlungsanweisung

VORSICHT!

Titel (optional)

Beschreibung

Handlungsanweisung (opt.)

Sicherheitshinweise

Sicherheitshinweise sind in dieser Anleitung durch das folgende Symbol gekennzeichnet. Die Sicherheitshinweise werden durch Signalwörter eingeleitet, die das Ausmaß der Gefährdung zum Ausdruck bringen.

HINWEIS!

Diese Kombination aus Symbol und Signalwort weist auf eine möglicherweise gefährliche Situation hin, die zu Sachschäden führen kann, wenn sie nicht gemieden wird.

Tipps und Empfehlungen

Tipps und Empfehlungen sind in dieser Anleitung durch das folgende Symbol gekennzeichnet.

Dieses Symbol weist auf wichtige Informationen hin, die Ihnen den Umgang mit dem System erleichtern.

Aufbau der Warnhinweise

→ **SIGNALWORT**

↳ Art der Gefahr und ihrer Quelle

↳ Mögliche Folgen bei Nichtbeachtung

- ↳ Maßnahmen zur Abwendung der Gefahr
- ↳ Vorbeugende Maßnahmen

Anordnung der Warnhinweise

Beziehen sich Warnhinweise auf einen ganzen Abschnitt, stehen sie am Anfang des Abschnitts (z.B. Kapitelanfang).

Beziehen sich Warnhinweise auf eine spezielle Handlungsanweisung, stehen sie vor der jeweiligen Handlungsanweisung.

Qualifiziertes Personal

Das zu dieser Dokumentation zugehörige Produkt/System darf nur für die jeweilige Aufgabenstellung von qualifiziertem Personal genutzt werden. Das geschieht unter Beachtung der für die jeweilige Aufgabenstellung zugehörigen Dokumentation, insbesondere der darin enthaltenen Sicherheits- und Warnhinweise.

Qualifiziertes Personal ist auf Grund seiner Ausbildung und Erfahrung befähigt, im Umgang mit diesen Produkt/System Risiken zu erkennen und mögliche Gefährdungen zu vermeiden.

Bestimmungsgemäßer Gebrauch

Produkte der LJU Automatisierungstechnik GmbH dürfen nur für die in der zugehörigen technischen Dokumentation vorgesehenen Einsatzfälle verwendet werden. Falls Fremdprodukte und -komponenten zum Einsatz kommen, müssen diese von der LJU Automatisierungstechnik GmbH empfohlen bzw. zugelassen sein.

Der einwandfreie und sichere Betrieb der Produkte setzt sachgemäßen Transport, sachgemäße Lagerung, Aufstellung, Montage, Installation, Inbetriebnahme, Bedienung und Instandhaltung voraus. Die zulässigen Umgebungsbedingungen müssen eingehalten werden. Hinweise in den zugehörigen Dokumentationen müssen beachtet werden.

Marken

Alle mit dem Schutzrechtsvermerk ® gekennzeichneten Bezeichnungen sind eingetragene Marken der LJU Automatisierungstechnik GmbH. Die übrigen Bezeichnungen in dieser Schrift können Marken sein, deren Benutzung durch Dritte für deren Zwecke die Rechte der Inhaber verletzen kann.

VSH_0038, 1, de_DE

Haftungsausschluss

Der Inhalt der Bedienungsanleitung wurde auf Übereinstimmung mit der beschriebenen Hard- und Software geprüft. Dennoch können Abweichungen nicht ausgeschlossen werden, so dass für die vollständige Übereinstimmung keine Gewähr übernommen werden kann. Die Angaben in dieser Bedienungsanleitung werden regelmäßig überprüft, notwendige Korrekturen sind in den nachfolgenden Versionen enthalten.

Haftungsbeschränkung

Alle Angaben und Hinweise in dieser Beschreibung wurden unter Berücksichtigung der geltenden Normen und Vorschriften, des Stands der Technik sowie unserer langjährigen Erkenntnisse und Erfahrungen zusammengestellt.

Die LJU Automatisierungstechnik GmbH übernimmt keine Haftung für Schäden und Betriebsstörungen aufgrund:

- Nichtbeachtung der Beschreibung
- Nichtbestimmungsgemäßer Verwendung
- Einsatz von nicht ausgebildetem Personal
- Von eigenständigen Umbauen und Verändern von Geräten

Des Weiteren erlischt bei Nichtbeachtung der Beschreibung die Gewährleistungspflicht durch die LJU Automatisierungstechnik GmbH.

Konformität

Geräte der LJU Automatisierungstechnik GmbH sind zu den EU-Richtlinien konform ausgelegt. Eine Kopie der EU-Konformitätserklärung kann jederzeit bei der LJU Automatisierungstechnik GmbH angefordert werden.

Verwendung und Aufbewahrung der Dokumentation

Diese Anleitung ermöglicht den sicheren und effizienten Umgang mit dem Gerät. Die Anleitung ist Bestandteil des Geräts und muss für das Personal jederzeit zugänglich aufbewahrt werden.

Das Personal muss diese Anleitung vor Beginn aller Arbeiten sorgfältig durchgelesen und verstanden haben. Voraussetzung für sicheres Arbeiten ist die Einhaltung aller angegebenen Sicherheitshinweise und Handlungsanweisungen.

Abbildungen in dieser Anleitung dienen dem grundsätzlichen Verständnis und können von der tatsächlichen Ausführung abweichen. Aus eventuellen Abweichungen können keine Ansprüche abgeleitet werden.

Gebrauch der Dokumentation

Diese Dokumentation ist Bestandteil des Gerätes oder der Anwendungssoftware. Sie ist in leserlichem Zustand allen Personen, die mit Montage-, Installations-, Inbetriebnahme-, Betrieb-, Wartungs- und Servicetätigkeiten am Gerät oder der Anwendungssoftware tätig sind, jederzeit zur Verfügung zu stellen.

VSH_0038, 1, de_DE

Mitgeltende Unterlagen

Ist das Gerät/System Teil einer projektspezifischen Anlagenplanung, gelten auch die in der Projektdokumentation befindlichen Unterlagen. Die darin enthaltenen Hinweise – insbesondere Sicherheitshinweise – unbedingt beachten.

Garantie

Die Garantiebestimmungen sind in den Allgemeinen Geschäftsbedingungen der LJU Automatisierungstechnik GmbH enthalten.

Urheberschutz

Die inhaltlichen Angaben, Texte, Zeichnungen, Bilder und sonstige Darstellungen dieser Anleitung sind urheberrechtlich geschützt und unterliegen den gewerblichen Schutzrechten. Jede missbräuchliche Verwertung ist strafbar.

Die Vervielfältigung dieser Dokumentation oder von Teilen dieser Dokumentation ist nur in den Grenzen der gesetzlichen Bestimmungen des Urheberrechtsgesetzes zulässig. Jede Änderung oder Kürzung ohne ausdrückliche schriftliche Zustimmung durch die LJU Automatisierungstechnik GmbH ist untersagt.

Änderungen vorbehalten

Wir behalten uns das Recht vor, Änderungen an den in diesem Dokument enthaltenen Informationen vorzunehmen, die sich aus unserem ständigen Bemühen zur Verbesserung unserer Produkte ergeben.

Inhaltsverzeichnis

1 Änderungsverzeichnis

Änderungen vorbehalten

Wir behalten uns das Recht vor, Änderungen an den in diesem Dokument enthaltenen Informationen vorzunehmen, die sich aus unserem ständigen Bemühen zur Verbesserung unserer Produkte ergeben.

Dokumenten-version

Version	Datum	Bemerkung
1	05.2017	

1 Änderungsverzeichnis

2 Allgemein

Vorwort

Dieses Dokument enthält Hinweise zur Implementierung der Kommunikation zwischen einer iDM-MCU und einer SPS. Es wird davon ausgegangen, dass der Leser das Ziel hat allgemeingültige, anlagenunabhängige Bausteine zu entwickeln.

Dieses Dokument versucht die Beschreibung allgemeingültig und vom SPS-Typ unabhängig zu halten. Aus diesem Grund enthält es keine speziellen Umsetzungshinweise für die unterschiedlichen SPS- bzw. Feldbustypen oder Entwicklungsumgebungen. Es wird davon ausgegangen, dass der Leser über ausreichende Kenntnisse verfügt, um den beschriebenen Weg für die jeweilige Zielplattform umsetzen zu können.

Die übertragenen Nutzdaten werden in diesem Dokument nicht beschrieben. Hierfür stehen jeweils spezielle Anwenderdokumentationen zur Verfügung.

Als Alternative zu eigenen Entwicklungen bietet LJU für verschiedene SPS-Typen bereits fertig entwickelte und getestete Bausteine an.

2.1 Kommunikation

Die iDM-MCU unterscheidet zwei Arten der Kommunikation. Für beide Arten der Kommunikation werden Bausteine gebraucht, die die Kommunikation verwalten damit der Anwender einfacher mit der MCU kommunizieren kann.

Zyklische Kommunikation

Die zyklische Kommunikation läuft ständig. Über sie werden I/O-Daten (Befehle und Statusmeldungen) mit der MCU ausgetauscht. Die I/O-Daten enthalten Informationen die ständig und möglichst aktuell von der SPS benötigt werden, um das Verhalten der Anlage steuern zu können.

Azyklische Kommunikation

Die azyklische Kommunikation wird nur bei Bedarf verwendet. Durch azyklischen Kommunikation werden Daten ausgetauscht, die unregelmäßig benötigt werden. Solche Daten steuern zwar ebenso das Verhalten der Anlage, sind aber oft nicht zeitkritisch. Zudem werden sie meistens nur zu einem bestimmten Zeitpunkt gebraucht oder einmalig generiert und danach nicht mehr geändert (z.B. bis ein Auftrag abgearbeitet wurde).

3 Lesen und Schreiben von zyklischen Daten

Der zyklische Datenaustausch muss so gestaltet werden, dass die Informationen so schnell wie möglich zwischen der SPS und der MCU ausgetauscht werden. Nur so ist sichergestellt, dass beide Seiten rechtzeitig auf die Anlagensituation reagieren können. Je länger es dauert bis die Ausgangsdaten an die MCU übertragen werden, desto weniger Zeit hat sie die Eingangsdaten an die SPS zu übermitteln. Dies würde bedeuten, dass die Eingangsdaten zu spät eintreffen und die SPS einen zusätzlichen Zyklus mit „alten“ Daten durchlaufen muss.

Eine Möglichkeit ist das Beschreiben physikalischer Ausgänge oder das Triggern einer Aktualisierung des Systemabbildes.

Da die Datenmenge sehr groß werden kann, müssen die Bausteine so erstellt werden, dass sie bei Bedarf in der Lage sind ein Gesamtabbild in Datenblöcke aufzuteilen und diese Datenblöcke im Multiplex-Verfahren zu übertragen. Dies gilt in beiden Richtungen und ist in beiden Richtungen unabhängig voneinander. Da das Eingangsabbild (Status von MCU zur SPS) größer als das Ausgangsabbild (Befehle von SPS zur MCU) ist, kann es vorkommen, dass zwar in diese Richtung (MCU → SPS) die Daten im Multiplex-Verfahren übertragen werden, nicht aber Gegenrichtung.

Ebenso muss es möglich sein, kleine Datenmengen in einem kleinen Abbild übertragen zu können, um nicht unnötig Anwender-Datenspeicher zu nutzen.

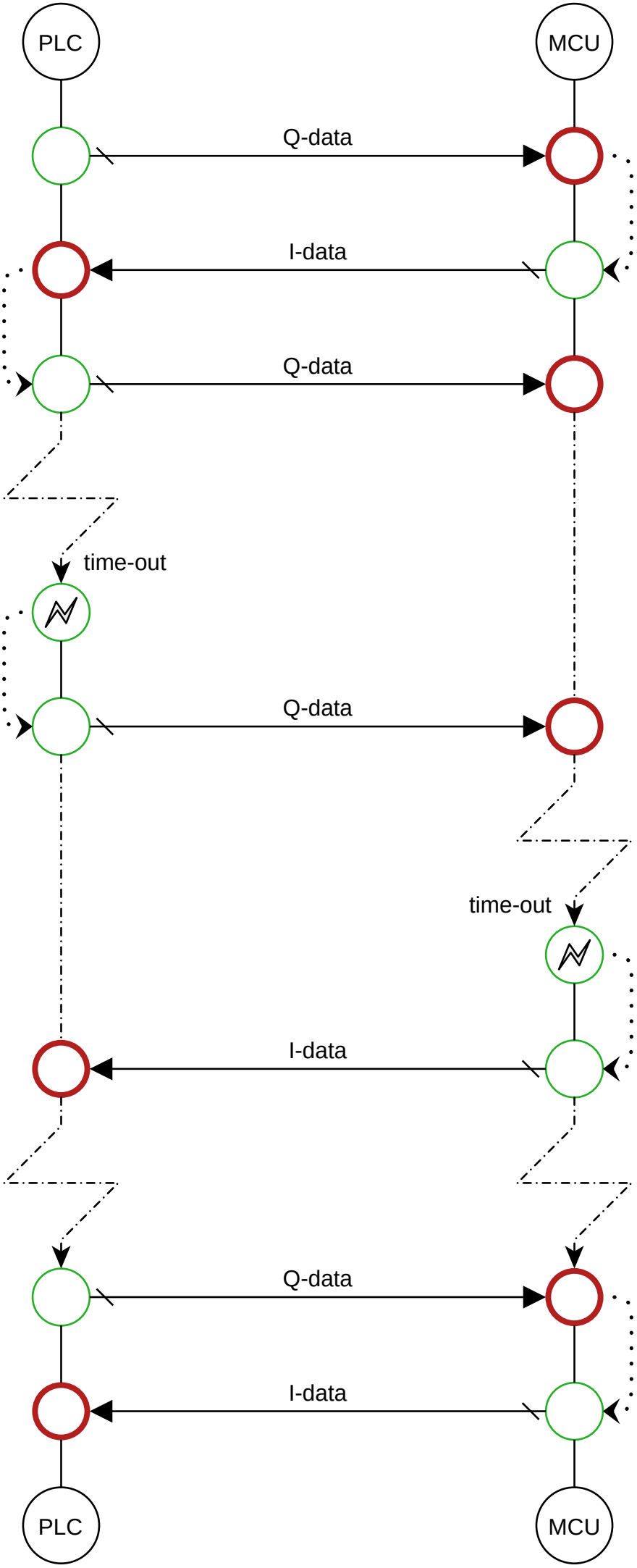
Um, aus Anwendersicht, die Handhabung des Systems möglichst einfach zu gestalten, muss es so aussehen, als stände ein klassisches E/A Abbild zur Verfügung.

Da das Schreiben und Lesen von E/A Datenblöcken aus Konsistenzgründen miteinander verknüpft ist, wäre es denkbar, dass nur ein Baustein erstellt wird, der für beide Richtungen die Kommunikation verwaltet. Da der Baustein dadurch aber eventuell unübersichtlich wird, wird empfohlen einen Baustein für die jeweilige Richtung zu erstellen und die Verknüpfung über die Parameter-Schnittstelle der Bausteine herzustellen.

Die erstellten Bausteine sollten bei der Anwendung an einer zentralen Stelle einmalig aufgerufen werden. Lediglich die zur Verfügung gestellten Abbilder sollten vom Anwender mehrfach im Zyklus beschrieben und gelesen werden.

3.1 Sequenzdiagramm

Der grundsätzliche E/A-Datenaustausch zwischen SPS und MCU wird im folgenden Diagramm dargestellt.


PLC
MCU
Q-data
I-data
Q-data
time-out
Q-data
time-out
I-data
Q-data
I-data
PLC
MCU

Beim Start sendet die SPS ihre Daten. Empfängt einer der Partner einen Datenblock, so muss er als Antwort sofort einen neuen Datenblock zurückschicken.

Auf beiden Seiten gibt es einen Time-Out. Nach Ablauf des Time-Outs muss die Kommunikation in ihren Anfangszustand zurückgesetzt und der 1. Datenblock eines neuen Abbildes gesendet werden. Nach jedem Senden läuft der Time-Out erneut an.

Da die Datenmenge sehr groß werden kann, werden bei Bedarf die Daten in mehrere Blöcke unterteilt. Die Datenblöcke müssen anschließend in aufsteigender Reihenfolge an die MCU verschickt werden. Wird die Reihenfolge nicht eingehalten, geht die MCU von einer Kommunikationsstörung aus. Anschließend verwirft sie die bis dahin empfangenen Daten und wartet auf den 1. Datenblock eines neuen Abbildes.

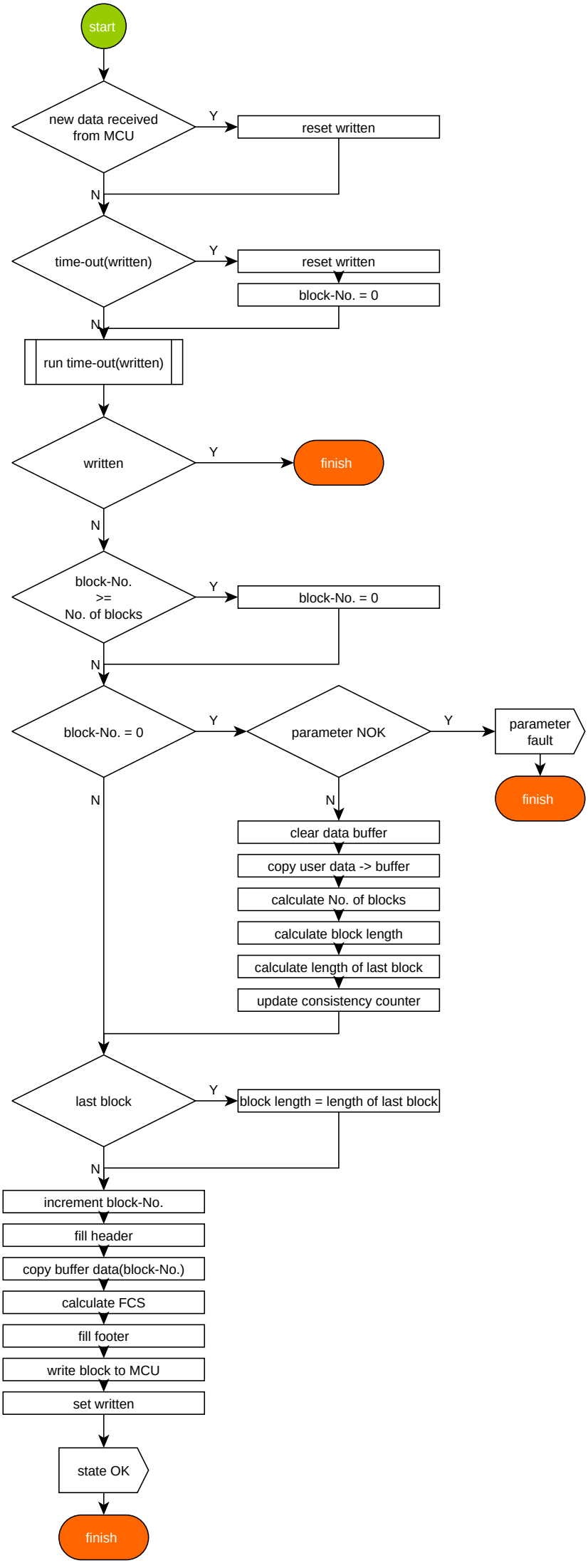
Die Größe der einzelnen Datenblöcke kann variieren. Es wird aber empfohlen, die Datenblöcke auf eine fixe Größe einzustellen und nur für den letzten Datenblock eine abweichende Größe zu wählen.

Da die MCU nicht ermitteln kann, wie groß der Empfangspuffer der SPS ist, wird beim Senden eines Datenblockes an die MCU übermittelt (Blockheader), wie viele Daten diese maximal an die SPS übermitteln kann.

Zur Übermittlung von einem Abbild können maximal 255 Datenblöcke verwendet werden. Dies gilt für das Schreiben und Lesen.

Das vom Anwender erstellte und übergebene Ausgangsabbild muss zunächst in einem Durchgang komplett und konsistent in einen internen Puffer kopiert werden. Erst nachdem der gesamte interne Puffer zur MCU versandt wurde (oder nach einem Time-Out), darf ein neues Ausgangsabbild in den internen Puffer kopiert werden.

MCU_PN_WRO Der Baustein schreibt das Ausgangsabbild an die MCU nach folgendem Diagramm:


start
new data received from MCU
Y
reset written
N
time-out(written)
Y
reset written
block-No. = 0
N
run time-out(written)
written
Y
finish
N
block-No. >= No. of blocks
Y
block-No. = 0
N
block-No. = 0
Y
parameter NOK
Y
parameter fault
finish
N
N
clear data buffer
copy user data -> buffer
calculate No. of blocks
calculate block length
calculate length of last block
update consistency counter
last block
Y
block length = length of last block
N
increment block-No.
fill header
copy buffer data(block-No.)
calculate FCS
fill footer
write block to MCU
set written
state OK
finish

consistency counter

Damit festgestellt werden kann, ob unterschiedliche Datenblöcke zum gleichen Abbild gehören, wird ein *[consistency counter]* eingesetzt. Der *[consistency counter]* muss für alle Datenblöcke eines Abbildes den gleichen Wert aufweisen.

Gültige Werte sind 1…255. Der Wert 0 ist nicht gültig und wird von der MCU ignoriert.

Es wird empfohlen den Wert für den [consistency counter] für jedes neue Abbild um 1 zu erhöhen. Prinzipiell steht jedoch jeder Wert, außer dem zuletzt benutzten zur Verfügung.

block length

Die *[block length]* gibt die Gesamtlänge des gesamten Datenblockes an. Sie umfasst also **Header**, **Data** und **Footer**. Sie darf daher den Wert für die Länge vom **Header**, **Footer** und mindestens 1 Nutzdatenbyte nicht unterschreiten. Aktuell sind dies 8 + 2 + 1 = 11 Byte.

Der Maximalwert wird von der Schnittstelle bestimmt die die MCU über den Feldbus mit der SPS aufbauen kann.

Die Länge kann auch eine ungerade Zahl sein.

block length inputs

Die *[block length inputs]* gibt an, wie viel Daten die MCU pro Datenblock maximal an die SPS senden kann. Dies muss der MCU übermittelt werden, da die MCU diese Größe nicht selbständig bei der SPS abfragen kann.

Der Wert umfasst **Header**, **Data** und **Footer**. Sie darf daher den Wert für die Länge vom **Header**, **Footer** und mindestens 1 Nutzdatenbyte nicht unterschreiten. Aktuell sind dies 8 + 2 + 1 = 11 Byte.

Der Maximalwert wird von der Schnittstelle bestimmt, die die MCU über den Feldbus mit der SPS aufbauen kann.

3.3.1.2 Data

user data

Die Nutzerdaten werden in diesem Entwicklerdokument nicht beschrieben.

Verweis
Weitere Informationen in der jeweiligen Anwenderdokumentation.

3.3.1.3 Footer

FCS

Um die Konsistenz und Richtigkeit der Daten innerhalb eines Datenblockes prüfen zu können, wird ein *[FCS]* (frame check sequence) eingesetzt. Speziell bei größeren Datenblöcken kann nicht garantiert werden, dass alle Daten auf einmal an die MCU übertragen werden. Ebenso kann nicht sichergestellt werden, dass die Daten sequenziell an die MCU übertragen werden. Aus diesem Grund muss mittels einer *[FCS]* geprüft werden, dass die Daten komplett, korrekt und konsistent sind.

Die *[FCS]* wird über den ganzen Datenblock bis zum **Footer**, also über **Header** und **Data** berechnet.

Beim Empfangen von Daten berechnet die MCU den Wert für die *[FCS]* und vergleicht ihn mit dem im **Footer** enthaltenen Wert. Unterscheiden sich die Werte, geht die MCU davon aus, dass die Daten noch nicht vollständig empfangen wurden und wartet.

[FCS]-Berechnung unter ↬ *Kapitel 5„Berechnen der FCS“ auf Seite 37*.

3.4 RDIO – Lesen des Eingangsabbildes von der MCU

Ziel des Bausteins soll es sein, einem Anwender ein Abbild der Eingänge von der MCU zu Verfügung zu stellen, das ohne weiteres verwendet werden kann. Es soll für den Anwender so aussehen, als ob ein Gerät direkt über physische Eingänge (oder ein Abbild dessen) angesprochen wird.

Eine MCU sendet erst dann einen Datenblock zur SPS, wenn sie zuvor von der SPS einen Datenblock empfangen hat. Umgekehrt gilt ebenfalls, dass die SPS erst einen weiteren Datenblock zur MCU schicken darf, wenn sie ein Datenblock von der MCU empfangen hat. Dieses Prinzip soll verhindern, dass es einen Datenüberlauf gibt wodurch inkonsistente Daten entstehen könnten.

Die Kommunikation startet mit dem Senden der Daten von der SPS zur MCU. Im ersten Zyklus muss davon ausgegangen werden, dass beim Lesen von der MCU noch keine Daten zur Verfügung stehen. Dies darf für den Baustein *[RDIO]* nicht zum Problem führen.

Da die Datenmenge sehr groß werden kann, unterteilt die MCU bei Bedarf die Daten in mehrere Blöcke. Die Datenblöcke müssen anschließend in aufsteigender Reihenfolge an die SPS verschickt werden. Wird die Reihenfolge nicht eingehalten, muss die SPS von einer Kommunikationsstörung ausgehen. Sie verwirft die bis dahin empfangenen Daten und wartet auf den 1. Datenblock eines neuen Abbildes.

VSH_0038, 1, de_DE

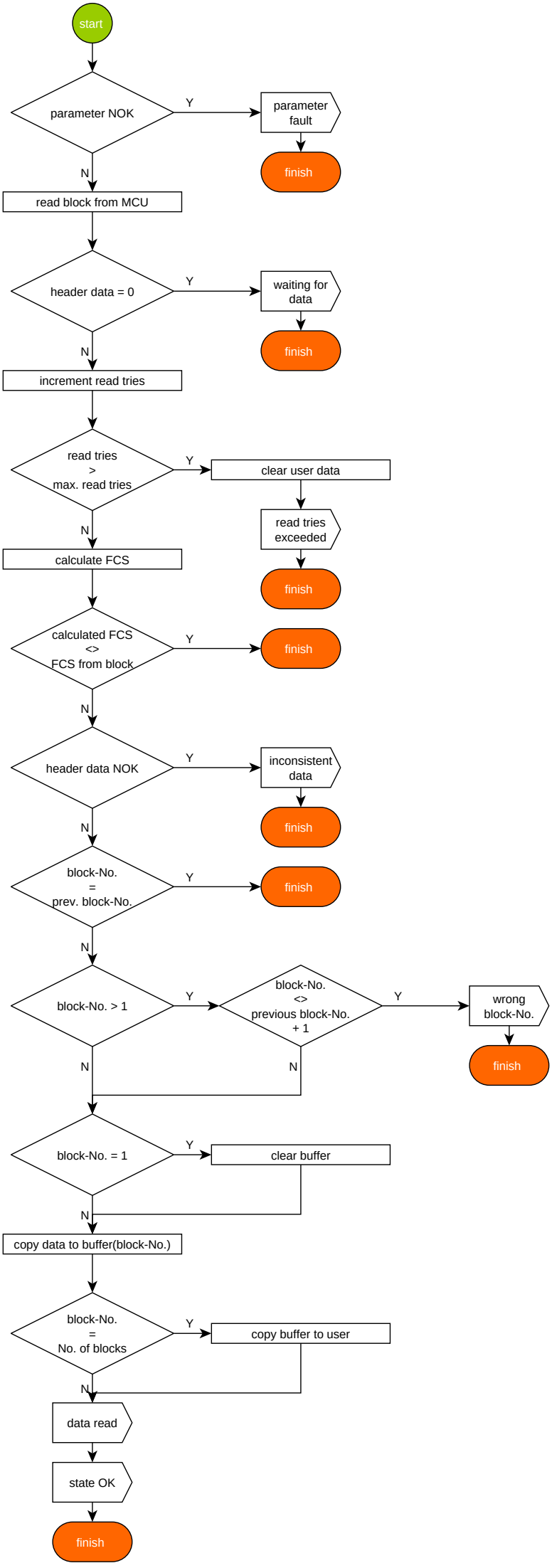
Da die Daten nicht gleichmäßig auf die Datenblöcke verteilt werden können, muss für den letzten Datenblock eine abweichende Blockgröße erwartet werden.

Da die MCU nicht ermitteln kann, wie groß der Empfangspuffer der SPS ist, wird beim Senden eines Datenblockes (↳ *Kapitel „WRIO – Schreiben des Ausgangsabbildes an die MCU“ auf Seite 16*) an die MCU übermittelt (Blockheader), wie viele Daten diese maximal an die SPS übermitteln kann (↳ *„block length inputs“ auf Seite 19*). Es können maximal 255 Datenblöcke verwendet werden um ein Abbild zu übermitteln.

Um die Datenkonsistenz des gesamten Abbildes sicherstellen zu können, muss zunächst das von der MCU empfangene Abbild in einem internen Puffer zwischengespeichert werden. Erst nachdem alle Datenblöcke von der MCU empfangen wurden, darf das Eingangsabbild komplett an den Anwender übergeben werden.

Tritt während der Kommunikation ein Fehler auf, muss die Kommunikation zurückgesetzt und mit dem Lesen des 1. Datenblocks eines neuen Abbildes neu begonnen werden. Die SPS muss bis dahin alle anderen Datenblöcke verwerfen.

MCU_PN_RDIO Der Baustein liest das Eingangsabbild von der MCU nach folgendem Diagramm:


start
parameter NOK
Y
parameter fault
finish
N
read block from MCU
header data = 0
Y
waiting for data
finish
N
increment read tries
read tries > max. read tries
Y
clear user data
read tries exceeded
finish
N
calculate FCS
calculated FCS <> FCS from block
Y
finish
N
header data NOK
Y
inconsistent data
finish
N
block-No. = prev. block-No.
Y
finish
N
block-No. > 1
Y
block-No. <> previous block-No. + 1
Y
wrong block-No.
finish
N
N
block-No. = 1
Y
clear buffer
N
copy data to buffer(block-No.)
block-No. = No. of blocks
Y
copy buffer to user
N
data read
state OK
finish

LJU behält sich das Recht vor dieses Byte bei zukünftigen Erweiterungen einzusetzen.

consistency counter

Damit festgestellt werden kann, ob unterschiedliche Datenblöcke zum gleichen Abbild gehören, wird ein *[consistency counter]* eingesetzt. Der *[consistency counter]* muss für alle Datenblöcke eines Abbildes den gleichen Wert aufweisen.

Gültige Werte sind 1…255. Der Wert 0 ist nicht gültig und muss von der SPS ignoriert werden.

Die MCU wird im störungsfreiem Betrieb diesen Zähler für jedes neue Abbild um 1 erhöhen. Prinzipiell könnte jedoch jeder Wert, außer dem zuletzt benutzten übermittelt werden (z.B. wenn die Kommunikation für einige Zeit unterbrochen war). Dies darf für den Baustein *[RDIO]* nicht zum Problem führen.

block length

Die *[block length]* gibt die Gesamtlänge des ganzen Datenblockes an. Sie umfasst also **Header**, **Data** und **Footer**. Sie darf daher den Wert für die Länge vom **Header**, **Footer** und mindestens 1 Nutzdatenbyte nicht unterschreiten. Aktuell sind dies 8 + 2 + 1 = 11 Byte.

Der Maximalwert wird von der Schnittstelle bestimmt die die MCU über den Feldbus mit der SPS aufbauen kann.

Die Länge kann auch eine ungerade Zahl sein.

Ist die *[block length]* außerhalb des erwarteten Bereichs, sind die bis dahin empfangene Daten zu verwerfen und die SPS muss auf den ersten Datenblock eines neuen Abbildes warten.

spare

Dieses Byte wird nicht benutzt.

LJU behält sich das Recht vor dieses Byte bei zukünftigen Erweiterungen einzusetzen.

3.4.1.2 Data

user data

Die Nutzerdaten werden in diesem Entwicklerdokument nicht beschrieben.

Verweis
Weitere Informationen in der jeweiligen Anwenderdokumentation.

3.4.1.3 Footer

FCS

Um die Konsistenz und Richtigkeit der Daten innerhalb eines Datenblockes prüfen zu können, wird eine [frame check sequence *[FCS]* eingesetzt. Speziell bei größeren Datenblöcken kann nicht garantiert werden, dass alle Daten auf einmal an die SPS übertragen werden. Ebenso wenig kann sichergestellt werden, dass die Daten sequenziell an die SPS übertragen werden. Aus diesem Grund muss mittels einer *[FCS]* geprüft werden, dass die Daten komplett, korrekt und konsistent sind.

Die *[FCS]* wird über den ganzen Datenblock bis zum **Footer**, also über **Header** und **Data** berechnet.

Beim Empfangen von Daten muss die SPS den Wert für die *[FCS]* berechnen und ihn mit dem im **Footer** enthaltenen Wert vergleichen. Unterscheiden sich die Werte, muss die SPS davon ausgehen, dass die Daten noch nicht vollständig empfangen wurden, und warten.

[FCS]-Berechnung unter ↬ *Kapitel 5„Berechnen der FCS“ auf Seite 37*.

4 Lesen und Schreiben von azyklischen Daten

Die Verwaltung und Organisation von azyklischen Aufträgen soll einfach sein. Dafür müssen die Bausteine für die azyklische Kommunikation so gestaltet sein, dass sie eine praktisch sinnvolle Anzahl von quasi-parallelen Aufträgen verarbeiten können.

Damit Aufträge unterschieden werden können, müssen sie durch Eigenschaften identifiziert werden. Diese Eigenschaften sind:

- Telegramm-Index
- Parameter der Datensätze

Die Bausteine müssen so geschrieben sein, dass sie von unterschiedlichen Stellen (z.B. für unterschiedliche Fahrzeuge) im Programm aufgerufen werden können. Unterschiedliche Datenbereiche sind nicht zulässig. Der jeweilige Aufruf darf immer nur seinen eigenen Status zurückerhalten.

HINWEIS!

Reihenfolge der eintreffenden Aufträge

Die Reihenfolge der eintreffenden Aufträge darf nicht verändert werden. Dies könnte zu unerwünschtem Verhalten der Anlage führen.

- Es muss sichergestellt sein, dass jeder Auftrag abgearbeitet werden kann. Es darf nicht möglich sein, dass Aufträge vorgeschoben werden und dadurch andere Aufträge nach hinten gestellt werden.

4.1 Verwendete Begriffe

Requested record length

[Requested record length] ist die Angabe wie viele Daten vom angegebenen Datensatz an die MCU übertragen werden.

Nicht immer kann die Maximallänge eines Telegramms übertragen werden. Mithilfe von [Requested record length] kann der Anwender explizit die Länge der Daten angeben.

Length of record

Dies ist die tatsächliche Länge des vom Anwender angegebenen Datensatzes.

Der angegebene Datensatz kann länger sein, als die Menge der Daten die man an die MCU übertragen möchte.

REQ

Mit dem Signal *[REQ]* steuert der Anwender, ob der Auftrag bearbeitet werden soll.

- Wenn *[REQ]* = TRUE, darf der Auftrag in den internen Puffer aufgenommen und bearbeitet werden.
- Wenn *[REQ]* = FALSE, darf der Auftrag nicht weiter bearbeitet, und **muss** aus dem internen Puffer gelöscht werden.

reject, busy, done, error, b/d/e

Diese Signale informieren den Anwender über den Zustand der Anfrage.

- *[reject]* → Auftrag war in Ordnung. Konnte aber nicht entgegen genommen werden, weil der interne Auftragspuffer voll ist.
- *[busy]* → Auftrag ist in Bearbeitung. Der tatsächliche Bearbeitungsstatus wird vom *[state]* angegeben.
- *[done]* → Auftrag wurde erfolgreich abgeschlossen.
- *[error]* → Auftrag konnte nicht erfolgreich ausgeführt werden. Die Ursache bzw. weitere Informationen findet man im *[state]*.

Die Signalkombination *[b/d/e]* steht für *[busy]*, *[done]* und *[error]* und wird aus Platzgründen nur da benutzt wo es möglich und sinnvoll ist.

state

Die Variable *[state]* gibt Auskunft über den Bearbeitungsstatus des Auftrages.

Die Variable sollte nach einer fehlerhaften Ausführung weitere Informationen über mögliche Ursachen enthalten.

reclen

Die Variable *[reclen]* gibt an, wie viele Daten von der MCU empfangen wurden (siehe ↳ *Kapitel 4.3.2.1„Header“ auf Seite 36*).

Die Angabe erfolgt in Byte.

buffer

Der *[buffer]* ist ein bausteininterner Puffer. Er speichert die Kenndaten der Aufträge um diese eindeutig identifizieren zu können.

Um eine relevante Anzahl Aufträge puffern zu können, ist es nicht sinnvoll die tatsächlichen Nutzdaten des Auftrages zu speichern. Dies würde den Puffer zu sehr belasten.

request-No.

Die *[request-No.]* ist die Nummer, unter der der Auftrag im Puffer eingetragen wurde.

Es ist sinnvoll nur den ersten Auftrag im Puffer zu bearbeiteten. Alle folgende Aufträge warten, bis dieser Auftrag aus dem Puffer ausgetragen wurde, wonach der nächste an erster Stelle steht und aktiv bearbeitet werden kann.

read()

Die Signale *[busy]*, *[done]*, *[error]*, *[state]* und *[reclen]* gibt es auch mit einem vorangestelltem *[read()]*

Für das tatsächliche Versenden eines Telegramms, muss ein SPS-eigener Systembaustein benutzt werden.

Die Signale mit einem vorangestelltem *[read()]* sind die Signale die von diesem SPS-eigenen Systembaustein generiert und ausgegeben werden.

logging on, logging data

Um den Anwender eine Möglichkeit zu geben, nachzuvollziehen welche Datensätze zwischen SPS und MCU ausgetauscht wurden, ist es hilfreich eine Logging-Funktion zu implementieren.

- *[logging on]* → Aktiviert Logging
- *[logging data]* → Logging-Datensatz

Es gibt keine feste Vorgabe, welche Daten genau in dem Logging-Datensatz aufgenommen werden sollen.

Hilfreich sind:

- *ein Datum/Zeit-Stempel*
- *ein Teil der Bausteinparameter*
- *die ersten Bytes des übertragenen Datensatzes*

4.2 WRREC – Schreiben eines Datensatzes an die MCU

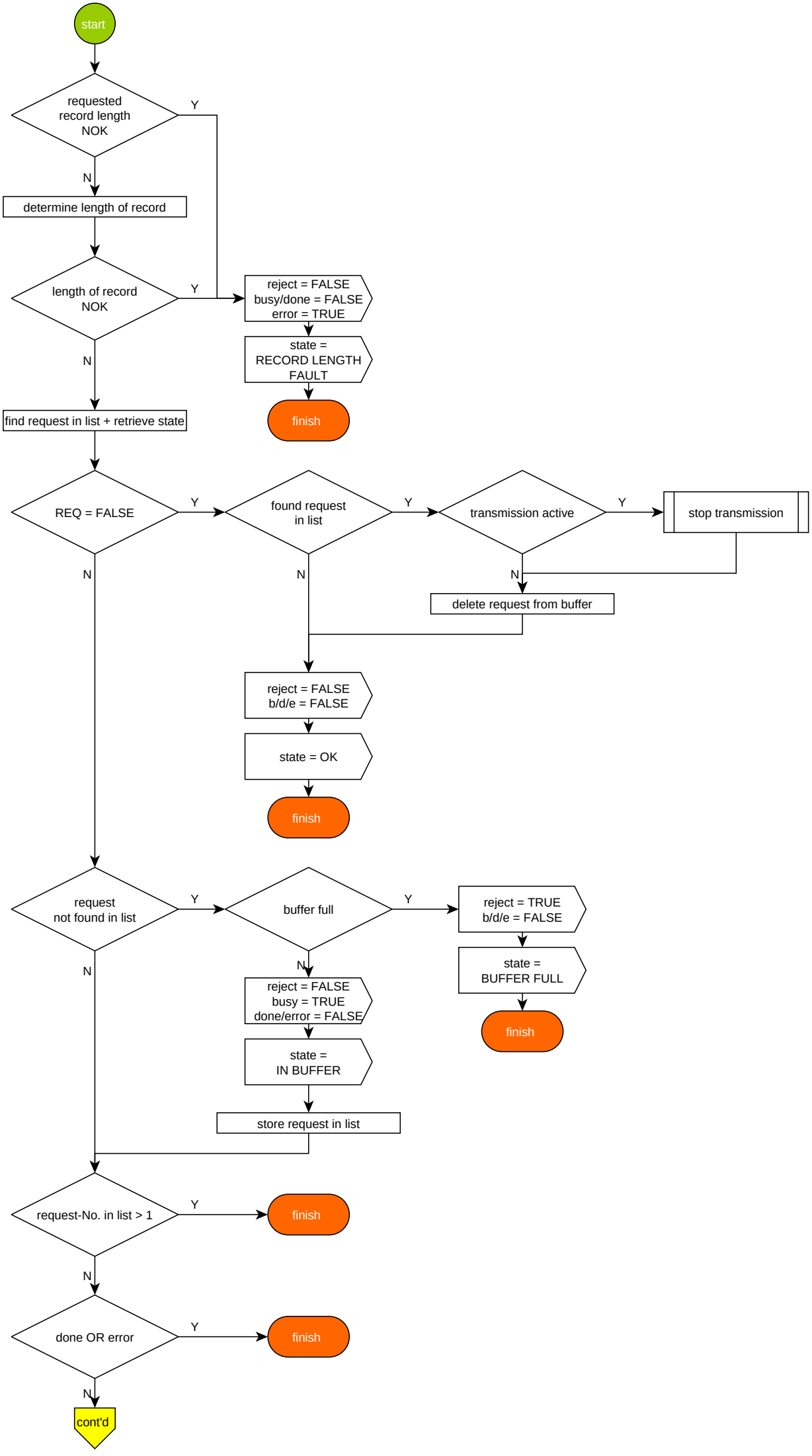
Ziel des Bausteines soll es sein, einem Anwender zu ermöglichen, auf einfachere Weise eine größere Anzahl von Telegrammen zu versenden. Wichtig ist dabei, dass die Reihenfolge, in der das Versenden der Telegramme angefordert wurde, nicht verfälscht wird. Speziell bei größeren Anlagen, mit vielen Fahrzeugen, muss davon ausgegangen werden, dass zeitweise viele Telegramme versandt werden müssen. Der Baustein muss innerhalb von praktikablen Grenzen dieses Aufkommen so gut wie möglich so organisieren, dass der Anwender nicht unnötig mit der Verwaltung belastet wird. Die Bausteine sollten es dem Anwender einfach machen, auch bei hohem Telegrammaufkommen seine Aufträge zügig absetzen zu können.

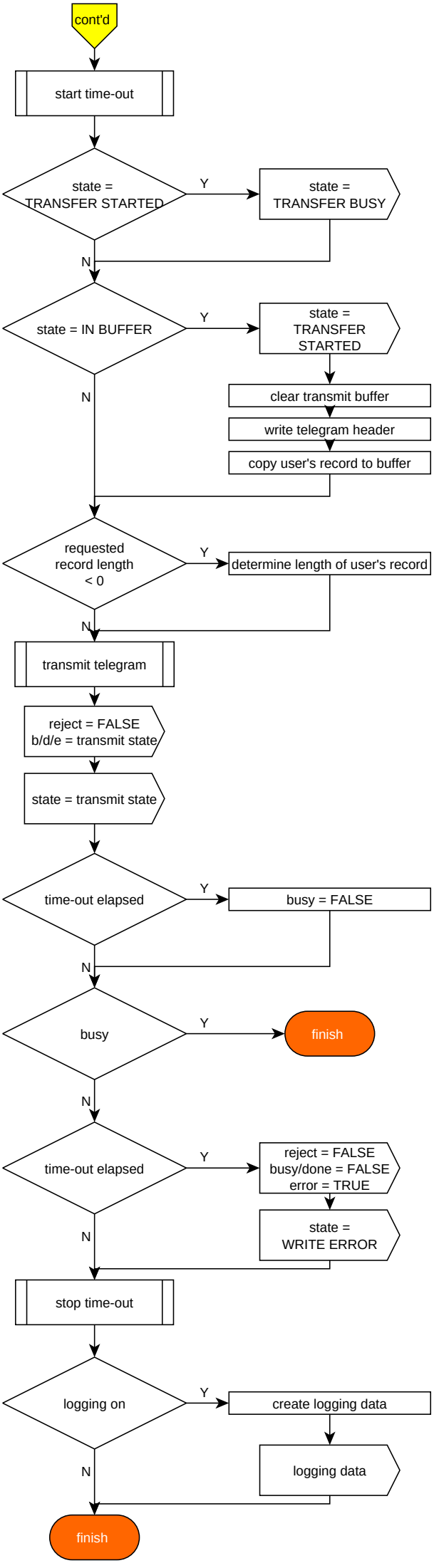
Es ist zu beachten, dass nach dem Versenden eines Telegramms, die MCU, protokollbedingt keine Antwort über den Empfang und/oder die Bearbeitung zurück schicken kann. Es ist nur möglich den Status des eigentlichen Sendevorgangs abzurufen.

Um abzurufen, ob die MCU die gesendeten Daten akzeptiert hat, kann dies durch einen anschließenden Lese-Auftrag realisiert werden. Alternativ stehen hierfür eventuell Bits in der I/O-Schnittstelle bereit. Letzteres ist vom Anwender mit LJU zu klären und ist nicht Bestandteil dieser Entwicklerdokumentation.

MCU_PN_WRREC

Dieser Baustein schreibt einen Datensatz an die MCU nach folgendem Diagramm.


start
requested record length NOK
Y
N
determine length of record
length of record NOK
Y
reject = FALSE
busy/done = FALSE
error = TRUE
state = RECORD LENGTH FAULT
finish
N
find request in list + retrieve state
REQ = FALSE
Y
found request in list
Y
transmission active
Y
stop transmission
N
N
N
delete request from buffer
reject = FALSE
b/d/e = FALSE
state = OK
finish
request not found in list
Y
buffer full
Y
reject = TRUE
b/d/e = FALSE
state = BUFFER FULL
finish
N
N
reject = FALSE
busy = TRUE
done/error = FALSE
state = IN BUFFER
store request in list
request-No. in list > 1
Y
finish
N
done OR error
Y
finish
N
cont'd


cont'd
start time-out
state = TRANSFER STARTED
Y
state = TRANSFER BUSY
N
state = IN BUFFER
Y
state = TRANSFER STARTED
clear transmit buffer
write telegram header
copy user's record to buffer
N
requested record length < 0
Y
determine length of user's record
N
transmit telegram
reject = FALSE
b/d/e = transmit state
state = transmit state
time-out elapsed
Y
busy = FALSE
N
busy
Y
finish
N
time-out elapsed
Y
reject = FALSE
busy/done = FALSE
error = TRUE
state = WRITE ERROR
N
stop time-out
logging on
Y
create logging data
logging data
N
finish

Dem Anwender muss eine Schnittstelle zur Verfügung stehen, womit er zu jeder Zeit die Nummern für alle 3 Geräte gleichzeitig angeben kann. Anschließend liegt es in seiner Verantwortung die Parameter richtig zu versorgen.

TCU No. Die *[TCU No.]* ist die Nummer der TCU, für die der Datensatz bestimmt ist.

FZ No. Die *[FZ No.]* ist die Nummer des Fahrzeuges, für das der Datensatz bestimmt ist.

type No. Die *[type No.]* ist die Nummer des Fahrzeugtyps, für den der Datensatz bestimmt ist.

4.2.1.2 Record

user record Die Nutzerdaten *[user record]* des Telegramms werden in diesem Entwicklerdokument nicht beschrieben.

Verweis
Weitere Informationen in der jeweiligen Anwenderdokumentation.

4.3 RDREC – Lesen eines Datensatzes von der MCU

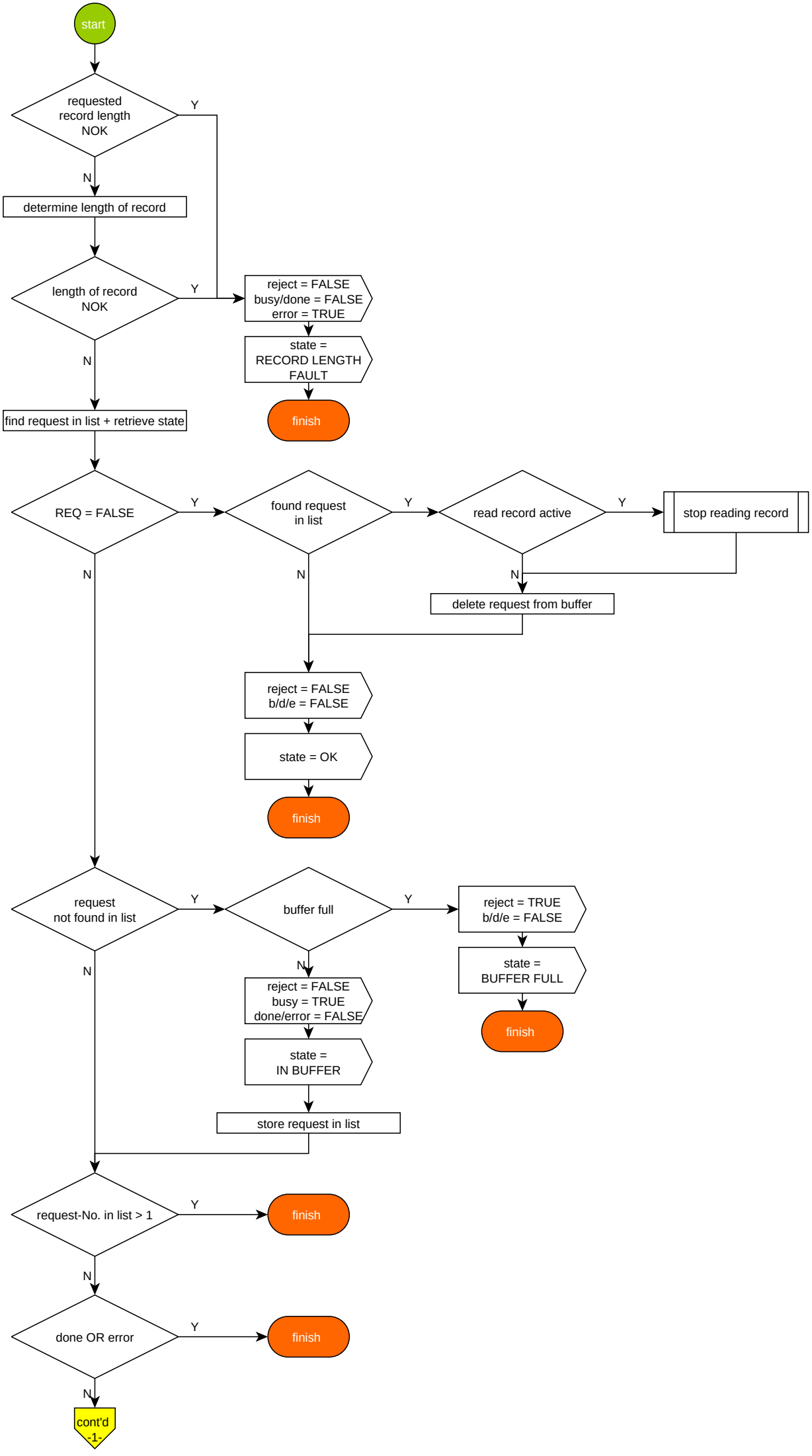
Ziel des Bausteines soll es sein, einem Anwender zu ermöglichen auf einfachere Weise eine größere Anzahl von Datensätzen zu lesen. Wichtig dabei ist, dass die Reihenfolge in der das Lesen der Datensätze angefordert wurde, nicht verfälscht wird. Speziell bei größeren Anlagen mit vielen Fahrzeugen muss davon ausgegangen werden, dass zeitweise viele Datensätze gelesen werden müssen. Der Baustein muss innerhalb von praktikablen Grenzen dieses Aufkommen so gut wie möglich so organisieren, dass der Anwender nicht unnötig mit der Verwaltung belastet wird. Die Bausteine sollten es dem Anwender einfach machen, auch bei hohem Telegrammaufkommen seine Aufträge zügig absetzen zu können.

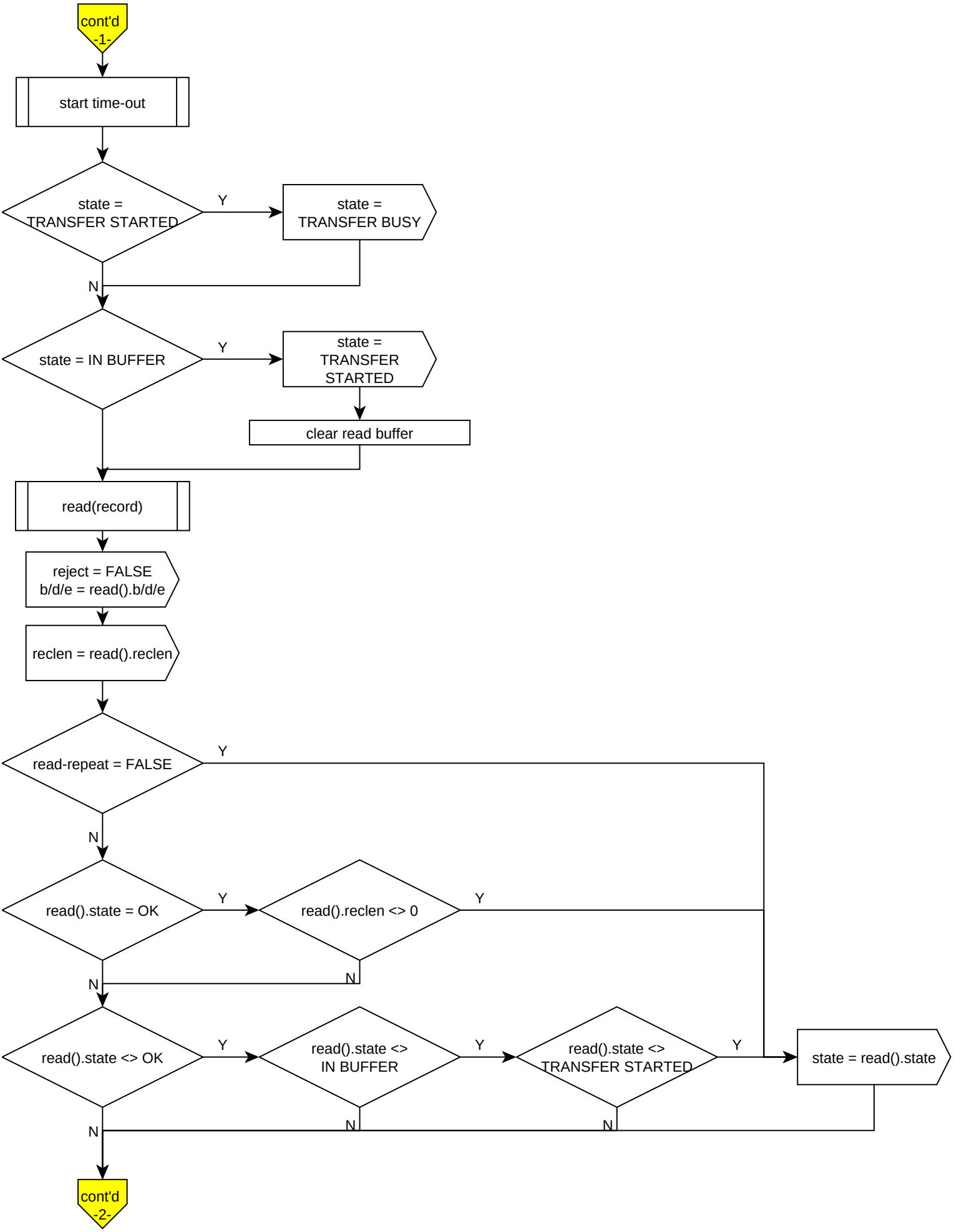
Es ist zu beachten, dass nach dem Anfordern eines Datensatzes, die MCU nicht immer in der Lage ist den Datensatz sofort zurück zu schicken. Dies könnte z.B. der Fall sein, wenn die MCU den Datensatz wiederum selbst bei einer TCU oder einem Fahrzeug anfordern muss. Tritt dieser Fall ein, merkt sich die MCU die Anforderung intern und sendet als Antwort sofort einen leeren Datensatz ohne Fehlermeldung zurück. Der Datensatz den die SPS empfängt enthält in dem Fall also weder Daten noch Fehlermeldungen. Die Anforderung kann sofort erneut gestellt werden. Eine Verzögerung bis zur erneuten Anforderung ist möglich, wird jedoch nicht empfohlen.

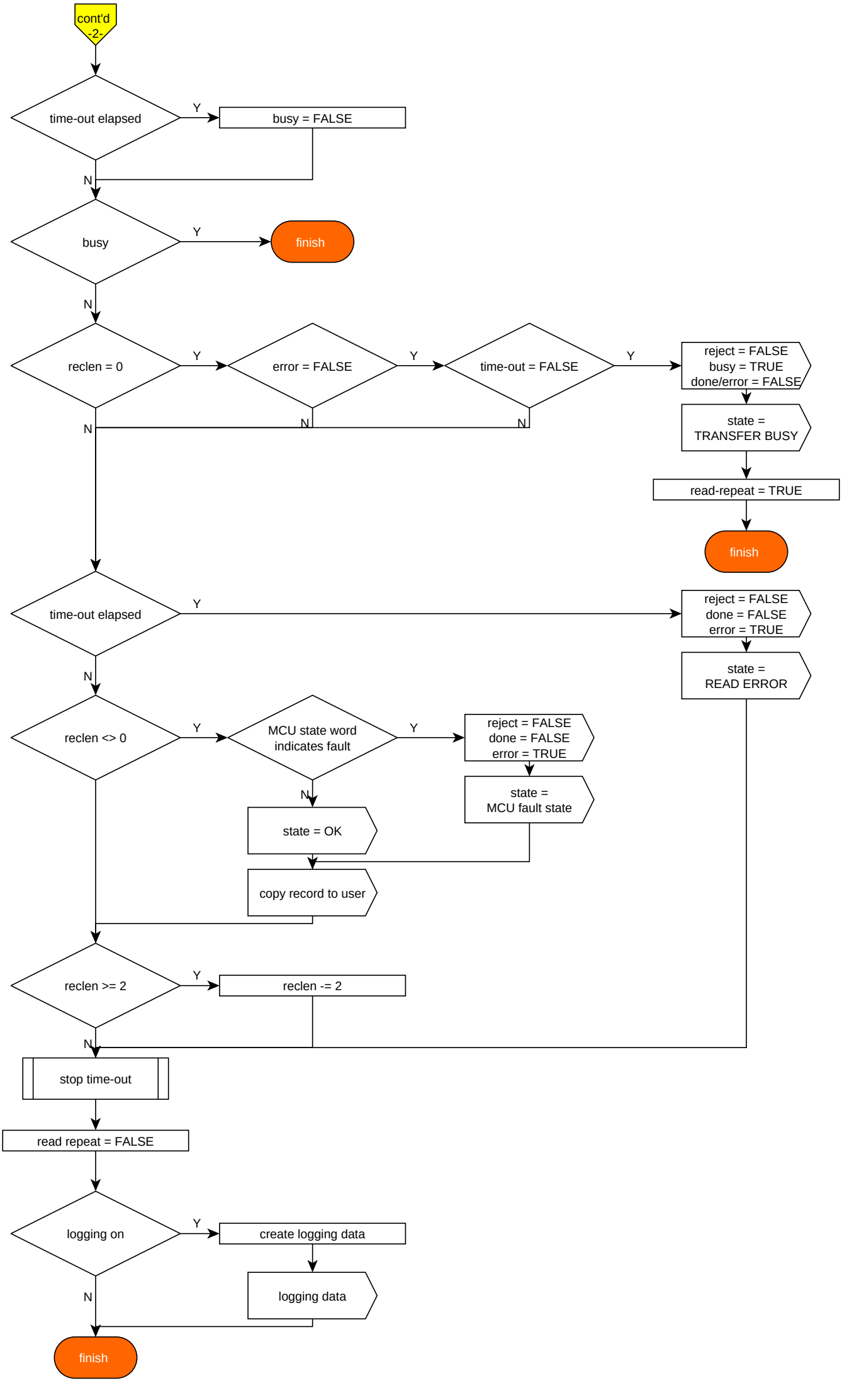
Mit einer Verzögerung kann im besten Fall das Telegrammaufkommen im Netzwerk äußerst geringfügig reduziert werden. Es verzögert nur den endgültigen Empfang der Daten. Nachdem die MCU die Daten zur Verfügung hat, werden diese bei der nächste Anfragewiederholung an die SPS geschickt und die MCU löscht den Auftrag aus seinem internen Speicher.

MCU_PN_RDREC

Dieser Baustein liest einen Datensatz von der MCU nach folgendem Diagramm:


start
requested record length NOK
Y
N
determine length of record
length of record NOK
Y
reject = FALSE
busy/done = FALSE
error = TRUE
state = RECORD LENGTH FAULT
finish
N
find request in list + retrieve state
REQ = FALSE
Y
found request in list
Y
read record active
Y
stop reading record
N
N
N
delete request from buffer
reject = FALSE
b/d/e = FALSE
state = OK
finish
request not found in list
Y
buffer full
Y
reject = TRUE
b/d/e = FALSE
state = BUFFER FULL
finish
N
N
reject = FALSE
busy = TRUE
done/error = FALSE
state = IN BUFFER
store request in list
request-No. in list > 1
Y
finish
N
done OR error
Y
finish
N
cont'd -1-


cont'd
-1-
start time-out
state =
TRANSFER STARTED
Y
state =
TRANSFER BUSY
N
state = IN BUFFER
Y
state =
TRANSFER
STARTED
clear read buffer
read(record)
reject = FALSE
b/d/e = read().b/d/e
reclen = read().reclen
read-repeat = FALSE
Y
N
read().state = OK
Y
read().reclen <> 0
Y
N
N
read().state <> OK
Y
read().state <>
IN BUFFER
Y
read().state <>
TRANSFER STARTED
Y
state = read().state
N
N
N
cont'd
-2-


cont'd
-2-
time-out elapsed
Y
busy = FALSE
N
busy
Y
finish
N
reclen = 0
Y
error = FALSE
Y
time-out = FALSE
Y
reject = FALSE
busy = TRUE
done/error = FALSE
N
N
N
state =
TRANSFER BUSY
read-repeat = TRUE
finish
time-out elapsed
Y
reject = FALSE
done = FALSE
error = TRUE
N
state =
READ ERROR
reclen <> 0
Y
MCU state word
indicates fault
Y
reject = FALSE
done = FALSE
error = TRUE
N
state =
MCU fault state
state = OK
copy record to user
reclen >= 2
Y
reclen -= 2
N
stop time-out
read repeat = FALSE
logging on
Y
create logging data
logging data
N
finish

Datentyp
In der Spalte Datentyp werden Vorschläge für den bevorzugten Datentyp und eine Alternativmöglichkeit angegeben. Prinzipiell kann jeder beliebige Datentyp gewählt werden, solange die Breite (Anzahl Bits) stimmt.

4.3.2.1 Header

status Der *[status]* enthält einen MCU-internen Status zum Leseauftrag. Im Fehlerfall kann der Anwender hiermit die Ursache ermitteln.

Eine ausführliche Beschreibung der Fehlermeldungen findet man in der Anwenderdokumentation.

- Es ist wichtig und sinnvoll diese Information dem Anwender weiterzuleiten, so dass er darauf reagieren kann.
- Es ist nicht sinnvoll mit dieser Information selbständig Maßnahmen einzuleiten. Ein Standardbaustein kann auf seiner Ebene nicht entscheiden, wie die Anwendersoftware reagieren möchte.

4.3.2.2 Record

user record Die Nutzerdaten *[user record]* des Telegramms werden in diesem Entwicklerdokument nicht beschrieben.

Verweis
Weitere Informationen in der jeweiligen Anwenderdokumentation.

5 Berechnen der FCS

Die verwendete FCS ist ein 16-Bit Wort. Sie wird nach folgender Formel berechnet:

FCS = $FCS_{high\ byte}$ XOR FCS-TABLE*[$FCS_{low\ byte}$ XOR $DATA_{byte\_num}$]*

Die Berechnung findet über **Header** und **Data** statt. Das Ergebnis wird im **Footer** eingetragen.

Pseudo-Code Da der Pseudo-Code nicht auf Details einer bestimmten Programmiersprache eingehen kann, gelten die folgenden Annahmen.

5.1 HEADER-LEN und DATA-LEN

Es wird angenommen, dass *[HEADER-LEN]* und *[DATA-LEN]* namentliche Konstanten sind.

Wenn es in der benutzten Programmiersprache keine namentlichen Konstanten gibt, können Variablen verwendet werden.

Namentliche Konstanten

Aus Konsistenzgründen wird die Verwendung namentlicher Konstanten empfohlen.

Direkte Konstanten

Es wird davon abgeraten mit direkten Konstanten (Zahlen) zu arbeiten. Unter Umständen wird das Programm diese Angaben nicht zur Berechnung verwenden können.

Die *[HEADER-LEN]* ist die Länge des Headers des Datenblocks, aktuell 8 Byte.

Die *[DATA-LEN]* ist die Länge der reinen Nutzdaten (Anwenderdaten) im Datenblock.

5.2 FCS, FCS{h} und FCS{l}

Die *[FCS]* ist die tatsächliche Frame Check Sequenz.

- Beliebiger Typ mit mindestens 16 Bit Breite
- Normal vom Typ WORD, UINT oder INT
- *[FCS{h}]* → High-Byte der *[FCS]*
- *[FCS{l}]* → Low-Byte der *[FCS]*

Zugriff auf High- / Low-Byte

Verschiedene Programmiersprachen halten verschiedene Konstrukte bereit auf das High- und/oder Low-Byte eines 16 Bit Wortes zuzugreifen. Dies kann zum Beispiel mittels eines `define`*, eines* `alias` *oder eines* `AT Konstrukts` *erfolgen, oder durch einen direkten Zugriff auf unterschiedliche Bytes eines breiteren Datentyps erfolgen, z.B. mittels* `FCS.B0` *oder* `FCS.B1`*.*

Alternativer Zugriff auf High- / Low-Byte

Hat die benutzte Programmiersprache keinen Zugriff, besteht die Möglichkeit, für [FCS{h}] und [FCS{l}] Einzelvariablen anzulegen und nach jeder Berechnung den Wert von [FCS] auf die Variablen [FCS{h}] und [FCS{l}] zu kopieren.

5.3 FCS Table

Die *[FCS Table]* ist eine Tabelle (`ARRAY[ ]`) die zuvor initialisiert wurde.

Die Art der Initialisierung ist nicht relevant.

Beispiele für Initialisierung

- Read-Only-Tabelle
  - mit vordefinierten Werten
- Tabelle
  - die vor jeder Verwendung aktiv mit Werten beschrieben wird

5.4 Register R0 und R1

Die Register *[R0]* und *[R1]* im Assembler Pseudo-Code sind Prozessor-Register die frei zur Verfügung stehen und als Adressregister benutzt werden können.

Befehl	
`T R0`	beschreibt Register *[R0]*
`L [R0]`	liest den Wert einer Adresse, auf die *[R0]* zeigt

Auch hier gilt wieder, dass unterschiedliche Sprachen unterschiedliche Konstrukte bereithalten (↬ *„Zugriff auf High- / Low-Byte“ auf Seite 38*).

Pseudo-Befehl	
`L ADDR(MCU-DATA)`	Lädt die Adresse der angegebenen Variablen, in diesem Fall *[MCU-DATA]*

Variable kann anschließend in ein Register transferiert werden.

5.5 FCS Programm Pseudo-Code

Folgender Pseudo-Code kann benutzt werden die FCS zu generieren:

```
FCS = 0xFFFF // initialize FCS
FOR i = 1 TO HEADER-LEN + DATA-LEN DO
FCS = FCS{h} XOR FCS-TABLE[FCS{l} XOR MCU-DATA[i]] // calculate FCS
END FOR // i
```

Bzw. alternativ in Assembler Pseudo-Code:

```
        L       0xFFFF              // initial value for FCS
        T       FCS                 // = 0xFFFF

        L       ADDR(MCU-DATA)      // address of 1st MCU data byte
        T       R0                  // save address in register R0

        L       HEADER-LEN          // length of header
        ADD     DATA-LEN            // + length of data

LOOP    T       NO-OF-BYTES         // = number of byte to check

        L       [R0]                // get value of MCU data byte
        XOR     FCS{l}              // XOR with FCS low byte
        ADD     ADDR(FCS-TABLE)     // add FCS-table start-address
        T       R1                  // save address in register R1

        L       [R1]                // get value from FCS-table
        XOR     FCS{h}              // XOR with FCS high byte
        T       FCS                 // new FCS value

        INC     R0                  // address of next MCU data byte

        L       NO-OF-BYTES
        SUB     1                   // decrement No. of bytes to
                                    check
        JNZ     LOOP                // jump if NO-OF-BYTES <> 0
```

5.6 FCS Berechnungstabelle

Die FCS Berechnungstabelle *[FCS-TABLE]* ist eine Tabelle mit 256 16-Bit Einträge. Sie enthält folgende Werte:

FCS-Tabelle Teil 1

		FCS low byte high nibble							
		0x0_	**0x1_**	**0x2_**	**0x3_**	**0x4_**	**0x5_**	**0x6_**	**0x7_**
FCS low byte low nibble	**0x_0**	0x0000	0x1081	0x2102	0x3183	0x4204	0x5285	0x6306	0x7387
	0x_1	0x1189	0x0108	0x308B	0x200A	0x538D	0x430C	0x728F	0x620E
	0x_2	0x2312	0x3393	0x0210	0x1291	0x6116	0x7197	0x4014	0x5095
	0x_3	0x329B	0x221A	0x1399	0x0318	0x709F	0x601E	0x519D	0x411C
	0x_4	0x4624	0x56A5	0x6726	0x77A7	0x0420	0x14A1	0x2522	0x35A3
	0x_5	0x57AD	0x472C	0x76AF	0x662E	0x15A9	0x0528	0x34AB	0x242A
	0x_6	0x6536	0x75B7	0x4434	0x54B5	0x2732	0x37B3	0x0630	0x16B1
	0x_7	0x74BF	0x643E	0x55BD	0x453C	0x36BB	0x263A	0x17B9	0x0738
	0x_8	0x8C48	0x9CC9	0xAD4A	0xBDCB	0xCE4C	0xDECD	0xEF4E	0xFFCF
	0x_9	0x9DC1	0x8D40	0xBCC3	0xAC42	0xDFC5	0xCF44	0xFEC7	0xEE46
	0x_A	0xAF5A	0xBFDB	0x8E58	0x9ED9	0xED5E	0xFDDF	0xCC5C	0xDCDD
	0x_B	0xBED3	0xAE52	0x9FD1	0x8F50	0xFCD7	0xEC56	0xDDD5	0xCD54
	0x_C	0xCA6C	0xDAED	0xEB6E	0xFBEF	0x8868	0x98E9	0xA96A	0xB9EB
	0x_D	0xDBE5	0xCB64	0xFAE7	0xEA66	0x99E1	0x8960	0xB8E3	0xA862
	0x_E	0xE97E	0xF9FF	0xC87C	0xD8FD	0xAB7A	0xBBFB	0x8A78	0x9AF9
	0x_F	0xF8F7	0xE876	0xD9F5	0xC974	0xBAF3	0xAA72	0x9BF1	0x8B70

FCS-Tabelle Teil 2

		FCS low byte high nibble							
		0x8_	**0x9_**	**0xA_**	**0xB_**	**0xC_**	**0xD_**	**0xE_**	**0xF_**
FCS low byte low nibble	**0x_0**	0x8408	0x9489	0xA50A	0xB58B	0xC60C	0xD68D	0xE70E	0xF78F
	0x_1	0x9581	0x8500	0xB483	0xA402	0xD785	0xC704	0xF687	0xE606
	0x_2	0xA71A	0xB79B	0x8618	0x9699	0xE51E	0xF59F	0xC41C	0xD49D
	0x_3	0xB693	0xA612	0x9791	0x8710	0xF497	0xE416	0xD595	0xC514
	0x_4	0xC22C	0xD2AD	0xE32E	0xF3AF	0x8028	0x90A9	0xA12A	0xB1AB
	0x_5	0xD3A5	0xC324	0xF2A7	0xE226	0x91A1	0x8120	0xB0A3	0xA022
	0x_6	0xE13E	0xF1BF	0xC03C	0xD0BD	0xA33A	0xB3BB	0x8238	0x92B9
	0x_7	0xF0B7	0xE036	0xD1B5	0xC134	0xB2B3	0xA232	0x93B1	0x8330
	0x_8	0x0840	0x18C1	0x2942	0x39C3	0x4A44	0x5AC5	0x6B46	0x7BC7
	0x_9	0x19C9	0x0948	0x38CB	0x284A	0x5BCD	0x4B4C	0x7ACF	0x6A4E
	0x_A	0x2B52	0x3BD3	0x0A50	0x1AD1	0x6956	0x79D7	0x4854	0x58D5
	0x_B	0x3ADB	0x2A5A	0x1BD9	0x0B58	0x78DF	0x685E	0x59DD	0x495C

FCS-Tabelle Teil 2

		FCS low byte high nibble							
		0x8_	**0x9_**	**0xA_**	**0xB_**	**0xC_**	**0xD_**	**0xE_**	**0xF_**
	0x_C	0x4E64	0x5EE5	0x6F66	0x7FE7	0x0C60	0x1CE1	0x2D62	0x3DE3
	0x_D	0x5FED	0x4F6C	0x7EEF	0x6E6E	0x1DE9	0x0D68	0x3CEB	0x2C6A
	0x_E	0x6D76	0x7DF7	0x4C74	0x5CF5	0x2F72	0x3FF3	0x0E70	0x1EF1
	0x_F	0x7CFF	0x6C7E	0x5DFD	0x4D7C	0x3EFB	0x2E7A	0x1FF9	0x0F78

6 Index