MOBA Mobile Automation AG

Spezifikation

InstructionDecoder()

Version 2.000

Produkt	**MRW 4-20mA** (**M**omenten unabhängige **R**edundante **W**ägezelle)
Auftraggeber	**MOBA Mobile Automation AG** Kapellenstraße 15 65555 Limburg Germany
Auftragnehmer	**MOBA Mobile Automation AG** Kapellenstraße 15 65555 Limburg Germany

Dokument erstellt von	Datum	Unterschrift
M.Offenbach	12.05.2022	

Diese Dokumentation des Unittests basiert auf einem Vordruck der MOBA AG.

MOBA Mobile Automation AG
Kapellenstr. 15
D-65555 Limburg
Internet: www.moba.de

Inhaltsverzeichnis

1 Einführung

1.1 Vorwort

Die MOBA AG versteht sich als Partner für die Entwicklung und Lieferung kundenspezifischer Elektronikkomponenten und daraus zusammengestellter Steuerungssysteme, die für den Einsatz an mobilen Maschinen konzipiert sind.

Die hier vorliegende Spezifikation beschreibt das exakte Verhalten der Funktion *InstructionDecoder()* der Datei *InstructionDecoder.c*
Dies beginnt mit der Angabe der Übergabeparameter sowie dem Rückgabewert der Funktion. Es folgen dann die Beschreibungen des Verhaltens der Funktion
Jede Beschreibung wird indiziert festgehalten. Somit ist in weiteren Dokumenten leicht Bezug auf die Spezifikation zu nehmen.

1.2 Änderungshistorie

Version	Datum	Kapitel	Änderung / Ergänzung
1.0	12.05.2022	alle	Erstellung

1.3 Ansprechpartner

MOBA Mobile Automation AG
Kapellenstraße 15
65555 Limburg

Name	Position	Telefonnummer	E-Mail
Boris Zils	Produktmanager	+49(0)6431-9577-123	b.zils@moba.de
Sebastian Schlesies	Vertrieb	+49(0)6431-9577-267	s.schlesies@moba.de
Jürgen Stiller	Entwicklungsleiter	+49(0)6431-9577-282	j.stiller@moba.de
Norbert Lipowski	Entwicklung	+49(0)6431-9577-137	n.lipowski@moba.de

1.4 Anhänge

Dokumentname	Beschreibung

1.5 Glossar

Abkürzung / Fachbegriff	Beschreibung / Definition
MRW	Momenten unabhängige Redundante Wägezelle
DMS	Dehnungsmessstreifen

2 InstructionDecoder()

2.1 Beschreibung

Zur Kommunikation zwischen Benutzer und Firmware bedient man sich eines einfachen zeichenorientierten Kommunikationsprotokoll. Dabei ist die Sende- wie auch die Empfangssequenz in einem mit STX ($0x02_{hex}$) und ETX ($0x03_{hex}$) umklammernden Frame eingefasst. Eine Prüfsumme existiert nicht.
Die Auswertung der eingehenden Sequenzen ist die Aufgabe von *InstructionDecoder()*.
Aufgrund des Umfangs der Funktion sollen hier nur die Spezifikationen für die Änderungen im Umfang des Updates auf Version V2.000 beschrieben werden.
Dies umfasst u.a. die Befehle *DIS* und *ENA* zur Sperrung der RS232-Kommunikation.

2.2 Spezifikation

Alle Spezifikationen sind in aufsteigender Reihenfolge zu erfüllen!

InstructionDecoder()		
Index	**Parameter**	**Datentyp**
7.2.0.0	./.	void
	Rückgabe	**Datentyp**
7.2.1.0	./.	void
	Verhalten	**Bemerkung**
7.2.2.0	Zur Sperrung der RS232-Kommunikation, ist zu Beginn der Befehlsausführung der Status der Sperre über die Funktion *Communication_IsCommunicationEnabled()* abzufragen.	Kommunikationssperre gesetzt
7.2.2.1	<u>Kommunikationssperre ist gesetzt</u> Wenn es sich bei dem zu verarbeitenden Befehl um das Kommando *ENA* oder *CDL* handeln, oder das Kommando über eine andere Schnittstelle als das RS232-Interface eingegangen sein sollte, ist mit der Befehlsauswertung fortzufahren.	Weitere Freigabekriterien zur Befehlsauswertung untersuchen COMMUNICATION_RS232 = 0
7.2.2.2	<u>Befehlsauswertung gesperrt</u> Maßnahmen ergreifen, dass es zu keiner Ausgabe über die RS232-Schnittstelle kommt – byCommandStatus auf *INSTRUCTIONDECODER_SEND_NOTHING* setzen	Befehlsauswertung gesperrt INSTRUCTIONDECODER_SEND_NOTHING = 255
7.2.2.3	<u>Befehlsauswertung freigegeben</u> Über einen Stringvergleich ist der auszuwertende Befehl zu identifizieren.	Befehl identifizieren
	Befehl *DIS*	
7.2.2.4	<u>Befehl *DIS* identifiziert (Stringvergleich über die ersten drei Zeichen)</u> • Zunächst ist eine Statusausgabe zu unterbinden (byCommandStatus = *INSTRUCTIONDECODER_SEND_NOTHING*) • Über die Funktion *InstructionDecoder_CheckNbParameters()* prüfen, ob die Anzahl der Parameter im Befehlsstring der Vorgabe (hier: 2) entspricht.	Befehl *DIS* verarbeiten SHOW_ERRORS_BY_CH0 = 0x01
7.2.2.5	<u>Anzahl der Parameter entspricht nicht der Vorgabe</u> Aufgrund des Parameters *SHOW_ERRORS_BY_CH0* in der Funktion *InstructionDecoder_CheckNbParameters()* bedarf es an dieser Stelle keiner weiteren Maßnahme. Die Befehlsverarbeitung ist unmittelbar abzubrechen.	Anzahl der Parameter falsch – Abbruch SHOW_ERRORS_BY_CH0 = 0x01
7.2.2.6	<u>Anzahl der Parameter entspricht der Vorgabe</u> Die beiden Parameter als Long-Werte in den Variablen *Parameter[0].nLong* bzw. *Parameter[1].nLong* ablegen. Hierzu sich der	Parameter einlesen

	Funktion *Communication_GetLongParameter()* bedienen	
7.2.2.7	Ist die Kanalangabe ungültig (*Parameter[0].nLong* > 1) ist die Ausgabe eines Fehlerframes vorzubereiten. Hierzu die Variable *byCommandStatus* auf *INSTRUCTIONDECODER_SEND_E001* setzen	Kanalangabe prüfen *INSTRUCTIONDECODER_SEND_E001* = 1
7.2.2.8	Kanalangabe gültig – Kanal 0 Nachfolgende Spezifikationen (7.2.2.9 - 7.2.2.10) sind nur gültig, wenn die Definition *CHANNEL_0* getroffen wurde.	Kanal 0 – Spezifikationen 7.2.2.9 bis 7.2.2.10 nur bei definiertem *CHANNEL_0*

7.2.2.9	Kanalangabe gültig – Kanal 0
	Entspricht der zweite Parameter dem Sperrcode *COMMUNICATION_DISABLE_CODE*, ist:
	• über die Funktion *Communication_EnableCommunication(0)* die RS232-Sperre zu setzen und den Staus der Funktionsausführung in *byRetVal* ablegen
	• Mitteilungstext gemäß byRetVal setzen.
	  • Ist der Wert = 0 (erfolgreiche Funktionsausführung) den Ausführungsstatus (*byCommandStatus*) auf *INSTRUCTIONDECODER_SEND_OK* und für den Docklight-Betrieb den Code des Mitteilungstextes (lText2Send) auf *TEXT_RS232_COMMUNICATION_DISABLED_CHANNEL_0* setzen.
	  • Im Fehlerfall (*byRetVal* != 0) ist der Ausführungsstatus (*byCommandStatus*) gleich *INSTRUCTIONDECODER_SEND_E005*
		RS232-Kommunikationssperre setzen *COMMUNICATION_DISABLE_CODE* = 0x55AA55AA
		
		INSTRUCTIONDECODER_SEND_OK = 0
		
		TEXT_RS232_COMMUNICATION_DISABLED_CHANNEL_0 = 997
		
		*INSTRUCTIONDECODER_SEND_E00*5 = 5

7.2.2.10	Der zweite Parameter entspricht nicht dem Sperrcode Ausführungsstatus (byCommandStatus) gleich *INSTRUCTIONDECODER_SEND_E005*	Falscher Sperrcode *COMMUNICATION_DISABLE_CODE* = 0x55AA55AA Ausführungsstatus auf *INSTRUCTIONDECODER_SEND_E005* (5)
7.2.2.11	Kanalangabe gültig – Kanal 1 Nachfolgende Spezifikationen (7.2.2.12 - 7.2.2.13) sind nur gültig, wenn die Definition *CHANNEL_1* getroffen wurde.	Kanal 1 – Spezifikationen 7.2.2.12 bis 7.2.2.13 nur bei definiertem *CHANNEL_1*
7.2.2.12	Kanalangabe gültig – Kanal 1 Entspricht der zweite Parameter dem Sperrcode () ist: • über die Funktion *Communication_EnableCommunication(0)* die RS232-Sperre zu setzen und den Staus der Funktionsausführung in *byRetVal* ablegen • Mitteilungstext gemäß byRetVal setzen.	RS232-Kommunikationssperre setzen *COMMUNICATION_DISABLE_CODE* = 0x55AA55AA

	• Ist der Wert = 0 (erfolgreiche Funktionsausführung) den Ausführungsstatus (*byCommandStatus*) auf *INSTRUCTIONDECODER_SEND_OK* und für den Docklight-Betrieb den Code des Mitteilungstextes (lText2Send) auf *TEXT_RS232_COMMUNICATION_DISABLED_CHANNEL_1* setzen. Im Fehlerfall (*byRetVal* != 0) ist der Ausführungsstatus (*byCommandStatus*) gleich *INSTRUCTIONDECODER_SEND_E005*	*INSTRUCTIONDECODER_SEND_OK* = 0 *TEXT_RS232_COMMUNICATION_DISABLED_CHANNEL_1* = 1997 *INSTRUCTIONDECODER_SEND_E00*5 = 5
7.2.2.13	Der zweite Parameter entspricht nicht dem Sperrcode Ausführungsstatus (byCommandStatus) gleich *INSTRUCTIONDECODER_SEND_E005*	Falscher Sperrcode *COMMUNICATION_DISABLE_CODE* = 0x55AA55AA Ausführungsstatus auf *INSTRUCTIONDECODER_SEND_E005* (5)
7.2.2.14	Die Befehlsverarbeitung *DIS* ist beendet	Abschluss
	Befehl *ENA*	
7.2.2.15	Befehl *ENA* identifiziert (Stringvergleich über die ersten drei Zeichen) • Zunächst ist eine Statusausgabe zu unterbinden (byCommandStatus = *INSTRUCTIONDECODER_SEND_NOTHING*) • Über die Funktion *InstructionDecoder_CheckNbParameters()* prüfen, ob die Anzahl der Parameter im Befehlsstring der Vorgabe (hier: 2) entspricht.	Befehl *ENA* verarbeiten SHOW_ERRORS_BY_CH0 = 0x01
7.2.2.16	Anzahl der Parameter entspricht nicht der Vorgabe Aufgrund des Parameters *SHOW_ERRORS_BY_CH0* in der Funktion *InstructionDecoder_CheckNbParameters()* bedarf es an dieser Stelle keiner weiteren Maßnahme. Die Befehlsverarbeitung ist unmittelbar abzubrechen.	Anzahl der Parameter falsch – Abbruch SHOW_ERRORS_BY_CH0 = 0x01
7.2.2.17	Anzahl der Parameter entspricht der Vorgabe Die beiden Parameter als Long-Werte in den Variablen *Parameter[0].nLong* bzw. *Parameter[1].nLong* ablegen. Hierzu sich der Funktion *Communication_GetLongParameter()* bedienen	Parameter einlesen
7.2.2.18	Ist die Kanalangabe ungültig (*Parameter[0].nLong* > 1) ist die Ausgabe eines Fehlerframes vorzubereiten. Hierzu die Variable *byCommandStatus* auf *INSTRUCTIONDECODER_SEND_E001* setzen	Kanalangabe prüfen *INSTRUCTIONDECODER_SEND_E001* = 1
7.2.2.19	Kanalangabe gültig – Kanal 0 Nachfolgende Spezifikationen (7.2.2.20 - 7.2.2.21) sind nur gültig, wenn die Definition *CHANNEL_0* getroffen wurde.	Kanal 0 – Spezifikationen 7.2.2.20 bis 7.2.2.21 nur bei definiertem *CHANNEL_0*

7.2.2.20	<u>Kanalangabe gültig – Kanal 0</u> Entspricht der zweite Parameter dem Sperrcode *COMMUNICATION_DISABLE_CODE*, ist: • über die Funktion *Communication_EnableCommunication(1)* die RS232-Sperre aufheben und den Staus der Funktionsausführung in *byRetVal* ablegen • <u>Mitteilungstext gemäß byRetVal</u> setzen. • Ist der Wert = 0 (erfolgreiche Funktionsausführung) den Ausführungsstatus (*byCommandStatus*) auf *INSTRUCTIONDECODER_SEND_OK* und für den Docklight-Betrieb den Code des Mitteilungstextes (lText2Send) auf *TEXT_RS232_COMMUNICATION_ENABLED_CHANNEL_0* setzen. Im Fehlerfall (*byRetVal* != 0) ist der Ausführungsstatus (*byCommandStatus*) gleich *INSTRUCTIONDECODER_SEND_E005*	RS232-Kommunikationssperre setzen *COMMUNICATION_DISABLE_CODE* = 0x55AA55AA *INSTRUCTIONDECODER_SEND_OK* = 0 *TEXT_RS232_COMMUNICATION_ENABLED_CHANNEL_0* = 998 *INSTRUCTIONDECODER_SEND_E00*5 = 5
7.2.2.21	<u>Der zweite Parameter entspricht nicht dem Sperrcode</u> Ausführungsstatus (byCommandStatus) gleich *INSTRUCTIONDECODER_SEND_E005*	Falscher Sperrcode *COMMUNICATION_DISABLE_CODE* = 0x55AA55AA Ausführungsstatus auf *INSTRUCTIONDECODER_SEND_E005* (5)
7.2.2.22	<u>Kanalangabe gültig – Kanal 1</u> Nachfolgende Spezifikationen (7.2.2.23 - 7.2.2.24) sind nur gültig, wenn die Definition *CHANNEL_1* getroffen wurde.	Kanal 1 – Spezifikationen 7.2.2.23 bis 7.2.2.24 nur bei definiertem *CHANNEL_1*
7.2.2.23	<u>Kanalangabe gültig – Kanal 1</u> Entspricht der zweite Parameter dem Sperrcode () ist: • über die Funktion *Communication_EnableCommunication(1)* die RS232-Sperre aufheben und den Staus der Funktionsausführung in *byRetVal* ablegen • <u>Mitteilungstext gemäß byRetVal</u> setzen. • Ist der Wert = 0 (erfolgreiche Funktionsausführung) den Ausführungsstatus (*byCommandStatus*) auf *INSTRUCTIONDECODER_SEND_OK* und für den Docklight-Betrieb den Code des Mitteilungstextes (lText2Send) auf *TEXT_RS232_COMMUNICATION_ENABLED_CHANNEL_1* setzen. Im Fehlerfall (*byRetVal* != 0) ist der Ausführungsstatus (*byCommandStatus*) gleich *INSTRUCTIONDECODER_SEND_E005*	RS232-Kommunikationssperre setzen *COMMUNICATION_DISABLE_CODE* = 0x55AA55AA *INSTRUCTIONDECODER_SEND_OK* = 0 *TEXT_RS232_COMMUNICATION_ENABLED_CHANNEL_1* = 1998 *INSTRUCTIONDECODER_SEND_E00*5 = 5
7.2.2.24	<u>Der zweite Parameter entspricht nicht dem Sperrcode</u>	Falscher Sperrcode *COMMUNICATION_DISABLE_CODE* =

	Ausführungsstatus (byCommandStatus) gleich *INSTRUCTIONDECODER_SEND_E005*	0x55AA55AA Ausführungsstatus auf *INSTRUCTIONDECODER_SEND_E005* (5)
7.2.2.25	Die Befehlsverarbeitung *ENA* ist beendet	Abschluss

3 Kommentare

4 Anhang