MOBA Mobile Automation AG

Spezifikation

ADuC836_SPIInterrupt()

aus der ADuC836-Bibliothek

Version 1.016

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	16.05.2022	

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 Bibliotheksfunktion *ADuC836_SPIInterrupt()* der Datei *ADuC836_SPIInterrupt.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	16.05.2022	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 ADuC836_SPIInterrupt()

2.1 Beschreibung

ADuC836_SPIInterrupt ist die Interruptroutine zum Empfang von über die SPI-Schnittstelle gesendete Zeichen. Diese Frames sind in die Framebegrenzungszeichen ‚STX' (0x02) und ‚ETX' (0x03) gebettet.
Der Empfang und damit der Eintrag der Zeichen an die erste Stelle im Empfangspuffer startet mit dem ersten Datum nach dem empfangenen ‚STX' und endet mit dem letzten Zeichen vor dem Empfang von ‚ETX'. Mit ‚ETX' muss noch ein Stringendezeichen (0x00) dem Puffer angefügt werden, um das Ende der Empfangszeichen erkennbar zu machen.
Während des Datenempfangs ist auf einen etwaigen Pufferüberlaufs zu prüfen. In diesem Fall werden die Daten verworfen und mit dem nächsten ‚STX' startet der Empfang erneut.

2.2 Spezifikation

Alle Spezifikationen sind in aufsteigender Reihenfolge zu erfüllen!

ADuC836_SPIInterrupt() ()		
Index	**Parameter**	**Datentyp**
21.2.0.0	./.	void
	Rückgabe	**Datentyp**
21.2.1.0	./.	void
	Verhalten	**Bemerkung**
21.2.2.0	Das Empfangszeichen ist aus dem SPI-Datenregister (SPIDAT) auszulesen und zwischen zu speichern	SPI-Datenregister auslesen
21.2.2.1	Zunächst ist zu prüfen, ob noch ein nicht verarbeitetes(r) Frame/Befehl im Empfangspuffer liegt (SPI.chNewCommandReceived = 1). Ist dies der Fall, ist die Interruptroutine sofort zu verlassen	Überprüfung auf ein noch nicht verarbeitetes Frame
21.2.2.2	Empfangspuffer ist leer: Handelt es sich beim Empfangszeichen um ‚STX', ist ein Empfangspointer (‚SPI.pchRecBufferIndex') auf die Startadresse des Empfangspuffers (‚SPI.pchRecBuffer') zu legen.	‚STX' empfangen
21.2.2.3	Wurde kein ‚STX' empfangen muss untersucht werden, ob dies zuvor stattgefunden hat und damit der Empfangsprozess eingeleitet wurde. Hierzu bediene man sich des Empfangspointers ‚SPI.pchRecBufferIndex'. Ist dieser 0, wurde zuvor noch kein ‚STX' empfangen – der Empfangsprozess läuft nicht.	Kein ‚STX' empfangen. Prüfung auf eingeleiteten Empfangsprozess
21.2.2.4	Empfangsprozess läuft: Es ist nun zu untersuchen, ob das Empfangszeichen dem Frameabschlusszeichen ‚ETX' entspricht.	Empfangsprozess läuft – Auswertung des nächsten Zeichens
21.2.2.5	Das aktuelle Empfangszeichen ist kein ‚ETX': Um die Daten nicht fälschlich hinter dem Empfangspuffer abzulegen, ist prüfen, ob der Empfangspointer ‚SPI.pchRecBufferIndex' innerhalb des Puffers liegt. Dies erfolgt mittels der bekannten Puffergröße (‚SPI.chBufferSize'):	Kein ‚ETX' empfangen Überprüfung auf Pufferüberlauf
21.2.2.6	Das aktuelle Empfangszeichen ist kein ‚ETX' und es findet kein Pufferüberlauf statt: Empfangszeichen an die Adresse des Empfangspointers (‚SPI.pchRecBufferIndex') ablegen und den Empfangspointer inkrementieren.	Kein ‚ETX' empfangen und kein Pufferüberlauf – Zeichen ablegen
21.2.2.7	Das aktuelle Empfangszeichen ist kein ‚ETX' aber es findet ein Pufferüberlauf statt: Empfangsprozess durch Setzen des Empfangspointers (‚SPI.pchRecBufferIndex') auf 0 anhalten. Damit wird erneut auf ein eingehendes ‚STX' gewartet	Kein ‚ETX' empfangen aber ein Pufferüberlauf – Empfangsprozess anhalten
21.2.2.8	Das aktuelle Empfangszeichen ist ein ‚ETX': An die Adresse des Empfangspointer	‚ETX' empfangen

	‚SPI.pchRecBufferIndex' ist zur Erkennung des Frameendes eine 0 zu schreiben. Durch das Setzen des Empfangspointers auf 0 wird der Empfangsprozess angehalten. Damit andere Funktionen darüber informiert werden, dass ein neues Kommando vorhanden ist, nun noch das Flag ‚Neuer Befehl' (‚SPI.chNewCommandReceived') auf 1 setzen.	Empfangsprozess anhalten und Flag ‚Neuer Befehl' setzen

3 Kommentare

4 Anhang