MOBA Mobile Automation AG

Lifecycle

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	

Dieser Lifecycle 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.

Der hier vorliegende Lifecycle beschreibt die funktionalen Änderungen von Version V1.103 nach Version V2.000. Um auf Codeebene die Unterschiede zu eruieren, sei an dieser Stelle auf die weiterführende Dokumentation ‚ChangeLog' verwiesen.

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 Glossar

Abkürzung / Fachbegriff	Beschreibung / Definition
MRW	Momenten unabhängige Redundante Wägezelle

2 Beschreibung

2.1 Begründung des Firmware-Updates V2.000

Aufgrund zweier Kundenreklamationen wurden die zurückgesendeten Wägezellen vom Typ ‚MRW420 digital' ausgiebig auf die Fehlerbeschreibung hin überprüft.

1. Die unbelastete Zelle zeigt einen sehr hohen Temperatureinfluss auf den Gewichtswert der unbelasteten Zelle
2. Trotz stetiger Lastzunahme bleibt der Ausgangsstrom bei ca. 8mA auf beiden Kanälen stehen, ohne die Anzeige eines Systemfehlers.

In beiden Fällen ließ sich der Fehler nachstellen und die Ursache in der Software begründen, wobei der zweite Punkt in Verbindung mit einem Hardware-Defekt steht.

2.2 Fehlerursache

2.2.1 Fehlerbild ‚Große Gewichtsabweichung bei hohen Temperaturen'

Eine Wägezelle zeigte mit zunehmender Abweichnug der Umgebungstemperatur von 20°C einen Anstieg bzw. Abfall des ermittelten Gewichtswerts. Nach ausgiebiger Untersuchung fand sich der Fehler in der falsch implementierten Routine zur E-Modul-Kompensation, bei der nicht nur der von der Korblast gebildete Messwert, sondern der gesamte Messwert inklusive des DMS-Brücken-Offsets zur Kompensation herangezogen wurde.

2.2.2 Fehlerbild ‚Ausgangsstrom beider Kanäle steigt nur bis zu einem gewissen Wert an'

Bei dieser Zelle zeigte sich, dass trotz Zunahme der Korblast, die Ausgangsströme beider Kanäle auf maximal ca.8mA einstellte. Die Untersuchung der Hardware ergab einen defekten StepUp-Regler des 17V-Netzteils. Dieser Defekt sorgte für die Durchschleifung der 5V-Eingangsspannung des Reglers auf dessen Ausgang. Aufgrund der nachfolgenden Beschaltung ergab sich letztendlich eine Versorgungsspannung der Stromregler von unter 4V – unstabilisiert. Da dieses Netzteil nicht redundant aufgebaut ist, zeigte sich der Fehler auf beiden Kanälen und kann nicht detektiert werden, zumal eine interne Überwachung zwischen Soll- und Ist-Strom nicht implementiert war.
Aufgrund der sehr geringen Ausfallwahrscheinlichkeit des StepUp-Reglers konnte bislang ein weiterer Defekt dieser Art im Feld nicht nachgewiesen werden.

2.2.3 Weitere nicht sicherheitskritische Firmwarefehler

Im Laufe der Untersuchungen zeigten sich noch andere Unzulänglichkeiten, welche im Rahmen dieses Updates behoben werden sollten.

2.2.3.1 Fehlerbild ‚Kein Schutz des RS232- und SPI-Empfangspuffers gegen Speicherbereichsüberlauf'

Die Speicherbereichsenden wurden weder beim RS232- als auch beim SPI-Empfangspuffer überwacht. In der Praxis ist dies nur ein Problem, sofern es zu einem Softwareupdate kommt, bei dem der zuerst programmierte Kanal auf der RS232-Schnittstelle mithört, oder bei extrem langen Kommandos, welche ein- oder zweimal hintereinander aufgrund fehlender Framebegrenzer (0x02 – STX und 0x03 – ETX) nicht korrekt empfangen wurden und somit aneinandergehängt werden. Aufgrund des Pufferüberlaufs ist nicht vorhersehbar, welche Folgen sich hieraus ergeben.
Im normalen Betrieb ist die RS232-Schnittstelle nicht aktiv und die SPI-Schnittstelle wird nur gelegentlich mit kurzen Befehlen angesprochen.

2.2.3.2 Fehlerbild ‚Überschreiben von Applikations-Parametern während des Software-Downloads'

Aufgrund der Teilung der RS232-Schnittstelle, hören stets beide Kanäle auf den Datenstrom. Das kann dazu führen, dass während des Software-Downloads der bereits zuvor programmierte Kanal eine vermeintliche Bitfolge empfängt, welche einem Uart-Befehl identisch ist und ihn zum Ändern von Applikations-Parametern auffordert. Dies geschieht auch, obwohl die Baudrate des Programmiervorgangs 115200Baud beträgt – gegenüber 9600Baud für die Applikations-Kommunikation.

2.3 Vorgaben zu den zu treffenden Maßnahmen

- Alle sicherheitsgerichteten Maßnahmen müssen gemäß EN ISO 13849 erfolgen. Dies beinhaltet u.a. die Umsetzung mit Hilfe des V-Modells.
- Alle zu treffenden Maßnahmen werden an der Firmware V1.103 v.07.03.2017 getroffen.
- Alle Änderungen dürfen die Ausgangssoftware nur minimal verändern.
- Alle Zellen mit alten Firmware-Versionen müssen updatebar sein.
- Die Problematik der falschen E-Modul-Kompensation ist durch Verlagerung der E-Modul-Kompensation hinter die Nullpunktverrechnung im Gewichtsermittlungsablauf zu beheben.
- Durch den Einbau einer Soll-/Iststrom-Überwachung können Fehler im Netzteil zur Versorgung der Stromschnittstelle erkannt werden. Diese führen zum dauerhaften (bis zum nächsten Spannungs-Reset) Systemfehler => Ausgangsstrom I_{out} = 0mA.
- Ab- und Anklemmen der Bürde darf nicht zum Systemfehler führen.
- Einbau einer Bereichsüberwachung der Eingangspuffer von SPI- und RS232-Schnittstelle in den zugehörigen Interrupt-Routinen der ADuC836-Bibliothek.
- Einbau einer gezielten Abschaltung der Verarbeitung von Befehlen über die RS232-Schnittstelle. Diese muss auch über einen Spannungsreset Bestand haben.
- Aufgrund Veränderungen im Eeprom-Speicher, müssen beim Update von älteren Versionen einzelne Parameter im Eeprom eingetragen und mit ihrem Defaultwert initialisiert werden. Die Statistikdaten sind zu löschen.

2.4 Umsetzung

2.4.1 Fehlerbild ‚Große Gewichtsabweichung bei hohen Temperaturen‘

2.4.1.1 Betroffene Dateien/Funktionen

In diesem Zusammenhang begrenzen sich die zu treffenden Maßnahmen auf die Funktion *Weight()* in der Datei *Weight.c*

2.4.1.2 Maßnahmen

Da die E-Modul-Kompensation nur am nullpunktbereinigten Messwert durchgeführt werden darf, muss der Funktionsaufruf zur E-Modul-Kompensation in der Folge hinter die Verrechnung des Nullpunkts.
Um die volle Kompatibilität zu älteren Firmwaren zu bewahren und nach einem Update mit den Original-Temperaturkennlinienwerten arbeiten zu können. Bedarf es noch der Anpassung zweier Variablen zur Speicherung von Daten während der Gewichtsermittlung:
Weight_ZeroCorrectedMeasurement und *Weight_FilteredMeasurement.*

Weight_ZeroCorrectedMeasurement:

Ursprünglich wurde dieser Wert
unmittelbar nach der Verrechnung des Nullpunkts abgelegt und beinhaltete auch die Temperatur- und E-Modul-Kompensation. Daher musste dieser jetzt direkt hinter der E-Modul-Kompensation gesetzt werden, um beide Berechnungen zu erfahren.

Weight_FilteredMeasurement:

Vor dem Update beinhaltete diese Variable den kompensierten und gefilterten Messwert. Da mit der neuen Firmwareversion das Verrechnen des Nullpunkts vor die E-Modul-Kompensation verlagert wurde, muss jetzt diese Variable nach der Kompensation angestellt und der Nullpunkt wieder extrahiert werden.

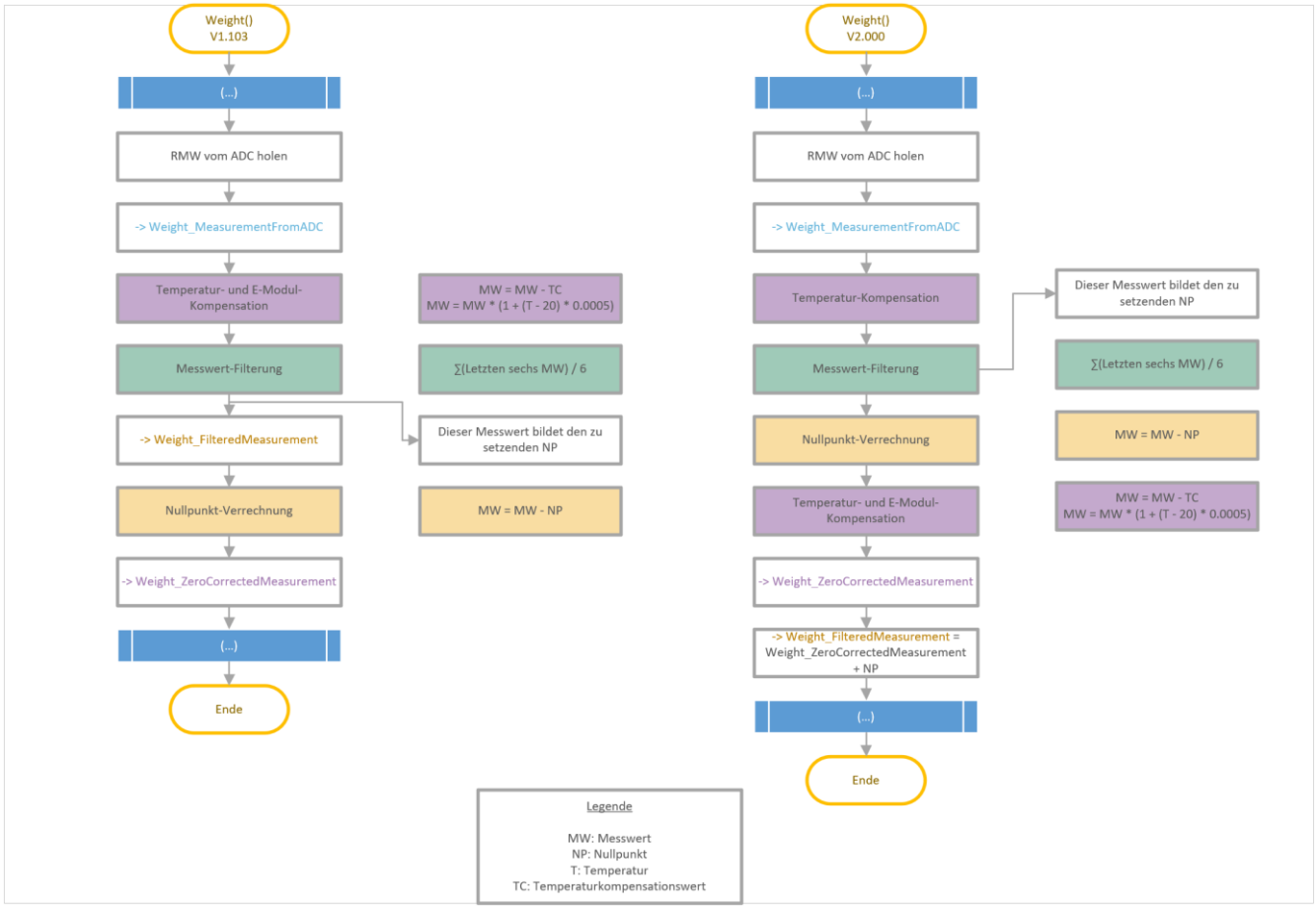


Abb.: Grafische Gegenüberstellung der Gewichtsermittlung von V1.103 und V2.000

2.4.2 Fehlerbild ‚Ausgangsstrom beider Kanäle steigt nur bis zu einem gewissen Wert an‘

2.4.2.1 Betroffene Dateien/Funktionen

- *CurrentInterface_Ini() – CurrentInterface.c*
- *CurrentInterface_Ini()_2 – CurrentInterface.c*
- *CurrentInterface_EvaluateDeviation() – CurrentInterface.c*
- *CurrentInterface_FeedBack() – CurrentInterface.c*
- *Diagnosis_SecurityCurrentInterface() – Diagnosis.c*

2.4.2.2 Maßnahmen

CurrentInterface_Ini():

Zur Behebung des Fehlers werden neue Parameter
Benötigt, welche zum Beginn initialisiert werden müssen. Um möglichst wenig Änderungen in den betriebsbewährten Funktionen vornehmen zu müssen, erfolgt hier nur der Aufruf der ergänzenden Initialisierungsfunktion *CurrentInterface_Ini()_2.*

CurrentInterface_Ini()_2:

Hier finden die Initialisierungen aller mit diesem Update
neu hinzugekommenen Parameter statt.
So wird hier der Wiederholungszähler zur Stromabweichungserkennung vorbesetzt und der Zellentyp (500/1000kg) über den Verstärkungsfaktor der Stromschnittstelle ermittelt. Über den Zellentyp bestimmt sich die maximal zulässige Stromabweichung, welche einem maximalen Fehler von umgerechnet 5kg entspricht.

CurrentInterface_EvaluateDeviation():

Diese Funktion ist mit diesem Update neu
mit eingeflossen und dient bei angeklemmter Bürde (Iststrom > 1mA) der Auswertung der Abweichung zwischen Ist- und Sollstrom.
Mit jeder erkannten Abweichung wird ein zuvor vorbesetzter Zähler dekrementiert. Ist dieser auf 0 zurückgelaufen, erfolgt die Überführung des Systems in den Sicherheitszustand (Ausgangsstrom = 0mA) und verbleibt dort bis zur Abschaltung der Versorgungsspannung. Der Zähler wird unterfolgenden Bedingungen zurückgesetzt: Systemaufstart, Stromabweichung innerhalb des Grenzwerts und Vorzeichenwechsel der Stromabweichung.

CurrentInterface_FeedBack():

Diese Funktion beinhaltet nun den Aufruf der
Funktion zur Auswertung der Stromabweichung (*CurrentInterface_EvaluateDeviation()*) und Maßnahmen zur Stromnachregelung bei angeklemmter Bürde (Iststrom > 1mA). Ergibt die Rückgabe der Funktion *CurrentInterface_EvaluateDeviation(),* dass die Bürde abgeklemmt ist (Rückgabe = 0), wird die bis dato ermittelte Regelgröße auf 0 gesetzt und die Ermittlung dieser erfolgt neu, sofern o.g. Funktion 1 (Bürde angeklemmt) zurückliefert und die ermittelte Stromabweichung < 5mA ist. Diese Abschaltung der Stromregelung bei einer Stromabweichung von mehr als 5mA ist notwendig, um bei der Zuschaltung einer Bürde keine zu große Stromabweichung aufgrund des noch voll aufgesteuerten Hardware-Reglers der Stromschnittstelle zu

erhalten. Dies würde eine sehr große Nachregelung in die falsche Richtung nach sich ziehen und kann leicht in die Überführung in den Sicherheitszustand aufgrund einer erkannten Stromabweichung führen. Daher wird für diesen Fall auf die Verlässlichkeit des Hardware-Reglers gesetzt.
Sobald die erkannte vorzeichenlose Stromabweichung kleiner 5mA ist, greift der Software-Regler und regelt mit jedem Durchlauf 50% der Abweichung gegen. Dies ist identisch mit der Version V1.103.

Diagnosis_SecurityCurrentInterface():

Diese Funktion wird beim Auftreten eines Fehlers im Bereich der Stromschnittstelle aufgerufen. Mit dem Update V2.000 ergänzt sich diese um die Überführung des Systems in den Sicherheitszustand durch den Aufruf der bewährten Funktion *System_SetSystemState(SYSTEM_ERROR)*.

2.4.3 Fehlerbild ‚Kein Schutz des RS232- und SPI-Empfangspuffers gegen Speicherbereichsüberlauf'

2.4.3.1 Betroffene Dateien/Funktionen

- *ADuC836_RS232Interrupt() – AduC836-Bibliothek – RS232_Interrupt.c*
- *ADuC836_SPIInterrupt() – AduC836-Bibliothek – SPI_Interrupt.c*
- *ADuC836_SPISendPtr() – AduC836-Bibliothek – SPI_SendPtr.c*

2.4.3.2 Maßnahmen

Die Maßnahmen finden an den beiden Interruptroutinen zum Empfang von Daten über die RS232- bzw. SPI-Schnittstelle statt. Dabei bedient man sich des gleichen Vorgehens.

ADuC836_RS232Interrupt():

Um Speicherplatz im RAM zu sparen entfällt die Variable *RS232.chSPICommunicationInProcess* und bedient sich des Hilfspointers *RS232.pchRecBufferIndex*, der im Fall des Wartens auf ein eingehendes STX auf 0 gesetzt wird.
Des Weiteren findet nach Eingang des Framebegrenzers STX mit jedem Eingangsdatum eine Überprüfung auf Empfangspufferüberschreitung statt, wobei die Grenze ein Byte vor Ende des Puffers liegt. Dies begründet sich darin, dass das letzte Zeichen für den Eintrag des Stringbegrenzers ‚\0' vorbehalten bleibt.
Ist der Eingangsstring zu lang, wird der Hilfspointer *RS232.pchRecBufferIndex* auf 0 gesetzt und der Empfang startet erneut mit dem Warten auf ein eingehendes STX.

ADuC836_SPIInterrupt():

Wie zuvor, jedoch entfällt hier die Variable *SPI.chSPICommunicationInProcess* und es kommt der Hilfspointer *SPI.pchRecBufferIndex* zum Einsatz.

ADuC836_SPISendPtr():

Aufgrund der Erweiterung der SPI-Interruptroutine, zieht dies eine längere Durchlaufzeit dieser nach sich. Dies hat zur Folge, dass bereits das nächste Zeichen über die SPI-Schnittstelle gesendet wird, bevor der Empfang des aktuellen abgeschlossen ist. Im Ergebnis zeigt sich das Fehlen einzelner Zeichen im Empfangsframe. Daher musste zur Behebung dieses Problems eine Pause nach jeder Sendung eingefügt werden. Durch eine einfache Schleife, bei der ein Variable mehrfach inkrementiert wird, erreicht man bei der getroffenen Konfiguration eine Pause von 200us.

2.4.4 Fehlerbild ‚Überschreiben von Applikations-Parametern während des Software-Downloads'

2.4.4.1 Betroffene Dateien/Funktionen

- *Communication_EnableCommunication() – Communication.c*
- *Communication_Ini() – Communication.c*
- *Communication_IsCommunicationEnabled() – Communication.c*
- *InstructionDecoder() - InstructionDecoder.c*
- *Load_Parameter() – Load_Save.c*
- *Save_Parameter() – Load_Save.c*

2.4.4.2 Maßnahmen

Zum Schutz vor dem Empfang vermeintlicher Befehlsframes im Laufe des Programmiervorgangs des Partnerkanals, wird mit diesem Update die Sperrung der Befehlsverarbeitung implementiert. Diese sperrt alle Befehle mit Ausnahme von *CDL* und *ENA*. Beide Befehle haben keine Auswirkung auf Parameter bzw. sicherheitskritische Parameter. Die Verriegelung erfolgt nur für den RS232-Kommunikationsweg.
Aus diesem Grund wurden im Befehlsdekoder zwei neue Befehle eingefügt – *DIS* und *ENA* – Verriegelung einschalten und Verriegelung ausschalten.
Beide Befehle sind mit einem 4byte breiten Code abgesichert und rufen die Funktion *Communication_EnableCommunication()* auf. Den Status der Befehlssperre gibt *Communication_IsCommunicationEnabled()* zurück.
Da die Sperre auch nach einem Spannungs-Reset erhalten bleiben muss, wird der 4byte breite Sperrcode im Eeprom abgelegt und beim Systemstart eingelesen.

Communication_EnableCommunication():

Diese Funktion ist neu hinzugefügt worden und dient dem Ein- und Ausschalten der Befehlsverriegelung über die Angabe des Übergabeparameters, der zum Ausschalten ‚1' sein muss. Andere Werte führen zum Einschalten der Sperre.
Das Einschalten führt dazu, dass der Sperrcode *0x55AA55AA* der Kontrollvariablen zugewiesen und im Eeprom hinterlegt wird. Im anderen Fall wird der bitinvertierte Wert zugewiesen und abgelegt.

Communication_Ini():

Sie wurde um die Initialisierung der Kontrollvariablen zur Steuerung der Befehlsverriegelung erweitert. Eine Erstinbetriebnahme, also das Setzen von Defaultwerten, löscht über den Aufruf von *Communication_EnableCommunication(1)* die Sperre.
Im normalen Initialisierungsdurchlauf wird hingegen der Sperrcode aus dem Eeprom ausgelesen und der Kontrollvariablen zugewiesen.

Communication_IsCommunicationEnabled():

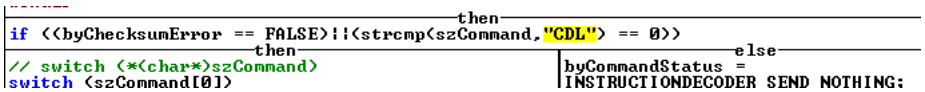
Auch diese Funktion wurde neu eingepflegt und dient der Ausgabe des Status der Befehlssperre. Dabei gilt folgende Definition der Rückgabe:

- 0: Befehlssperre aktiviert
- 1: Befehlssperre deaktiviert

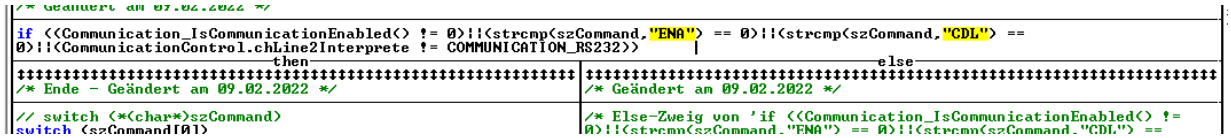
InstructionDecoder()

Im Bereich der Eingangsprüfung zur Befehlsverarbeitung, wurden folgende Bedingung ergänzt: Wenn die Befehlssperre aktiviert ist, werden nur Befehle über die SPI-Schnittstelle eingegangen ausgeführt oder die Befehle *CDL* oder *ENA*. Die Überprüfung auf einen Prüfsummenfehler wurde eliminiert, da in V1.103 die Abfrage stets erfüllt war.

V1.103:



V2.000:



Die Befehlsliste erhielt zwei Erweiterungen - die Kommandos *DIS* und *ENA.*
DIS steht für das setzen der Befehlssperre. Hierbei folgen dem Befehl zwei Parameter. Der erste spezifiziert den Kanal, für den der Befehl bestimmt ist. Der Sperrcode (1437226410) steht im zweiten Parameter. Ist dieser korrekt erfolgt der Aufruf von *Communication_EnableCommunication(0)* – die Verriegelung ist gesetzt. Im Fall eines falschen Codes erfolgt eine Fehlerausgabe (E005).
Gleiches gilt für das *ENA-Kommando* (Enable). Hier erfolgt der Aufruf der Funktion *Communication_EnableCommunication(1)* bei korrektem Code (1437226410) – die Befehlsverarbeitung ist freigegeben.

Load_Parameter()

Liest in diesem Fall den Wert der Kontrollvariablen zur Sperre der Befehlsverarbeitung aus dem Eeprom aus.
Hierzu wurde in der Funktion der case-Zweig LOAD_SAVE_RS232_DISABLE_CODE *ergänzt.* In diesem wird die Bibliotheksfunktion *Ee24c64_Read()* mit der Eeprom-Adresse 276_{dez} (Definition *EEPROM_RS232_DISABLE_CODE* in *Load_Save.def)* als erster Parameter und der festen Datenlänge von 4 Bytes als zweiter Parameter aufgerufen. Der letzte Parameter enthält die Adresse des Ziels, der Kontrollvariablen der Befehlssperre.

Save_Parameter()

Speichert in diesem Fall den Wert der Kontrollvariablen zur Sperre der Befehlsverarbeitung in das Eeprom ab.
Hierzu wurde in der Funktion der case-Zweig LOAD_SAVE_RS232_DISABLE_CODE *ergänzt.* In diesem wird die Bibliotheksfunktion *Ee24c64_Write()* mit der Eeprom-Adresse 276_{dez} (Definition *EEPROM_RS232_DISABLE_CODE* in *Load_Save.def)* als erster Parameter und der festen Datenlänge von 4 Bytes als zweiter Parameter aufgerufen. Der letzte Parameter enthält die Adresse des Quelle, der Kontrollvariablen der Befehlssperre.

2.4.5 Sekundärmaßnahmen – Reorganisation des Eeproms.

2.4.5.1 Betroffene Dateien/Funktionen

- *Load_Parameter() – Load_Save.c*
- *Main_UpdateRetains()*
- *Save_Parameter() – Load_Save.c*
- *Version_SoftwareVersionToLong() – Version.c*

2.4.5.2 Maßnahmen

Aufgrund der Firmware-Evolution wurden neue Parameter, im Eeprom liegend, hinzugefügt. Zwei Parameter aus dem Bereich ‚Statistik' kamen im Laufe der Zeit an anderen Positionen zu liegen. Aufgrund der Vorgabe, dass alle Auslieferungs-Versionen auf die Version V2.000 updatebar sein müssen, bedarf es der Neuanlage und Initialisierung dieser Parameter, sowie dem Löschen der statistischen Werte und damit einhergehend die Initialisierung dieser Werte.
Da diese Reorganisation nur einmalig zulässig ist, dient ein Eintrag im Eeprom, welcher die aktuelle Software-Version repräsentiert, als Zugangsflag. Dies wird am Ende der Reorganisation im Eeprom abgelegt und bei jedem Aufstart der Software eingelesen. Bei Nicht-Übereinstimmung mit der aktuellen Version, ist die beschriebenen Maßnahmen einzuleiten.

Load_Parameter():

Zur Ausgabe der als Long-Wert im Eeprom abgelegten Software-Version, wird dieser Funktion eine case-Anweisung (*LOAD_SAVE_VERSION*) zugefügt. Sofern das Ziel der Ausgabe kein Nullzeiger ist, erfolgt der Aufruf der Bibliotheksfunktion *Ee24c64_Read()* mit der Eeprom-Adresse 1048_{dez} (Definition *EEPROM_VERSION* in *Load_Save.def)* als erster Parameter und der Datenlänge von 4Byte als zweiter Parameter. Diesen folgt als dritter Parameter der Zeiger auf das Ziel der Daten. Eine Rückgabe gibt es nicht.

Main_UpdateRetains():

Sie ist die Schlüsselfunktion zur Reorganisation des Eeproms. Da, wie bereits oben erwähnt, die Reorganisation des Eeproms nur einmalig erfolgen darf, steht ganz oben auf der Liste der zu erledigenden Aufgaben die Überprüfung der im Eeprom abgelegten mit der aktuellen Firmware-Version an. Dazu wird zunächst mittels *Version_SoftwareVersionToLong()* der 4byte-Wert der aktuellen Version gebildet und anschließend mit dem im Eeprom abgelegten Wert verglichen. Bei Übereinstimmung, was nach erfolgter Reorganisation der Normalfall ist, wird die Funktion mit dem Rückgabewert ‚0' verlassen.
Sind die Versionen different, wird die Reorganisation eingeleitet. Dabei werden die Parameter der RS232-Baudrate (Defaultwert: 9600), ADC-Wandlerrate (Defaultwert: 5.35), Flag ‚SPI-Fehlerzäher nullen' (Defaultwert: 0), Flag 'Kein Reset bei mehrfachen SPI-Fehlern' (Defaultwert: 0), Code zur RS232-Kommunikationssperre (Defaultwert: 0) und der Proportinalanteil der SetDAC-Routine(Defaultwert: 50) nacheinander auf ihren Defaultwert gesetzt und im Eeprom eingetragen. Nun folgt die Initialisierung des Statistik-Speichers über die Statistikfunktion *Statistics_Ini(1)*. Diesem schließt sich das Setzen der 4byte-Firmware-Version im Eepromspeicher an.
Jeder Schritt in dieser Ablaufkette wird nur dann ausgeführt, wenn die vorherige Aktion erfolgreich abgeschlossen wurde. Im anderen Fall geht das System in den

Sicherheitszustand und verbleibt dort bis zum Aus- und wieder Einschalten der Betriebsspannung. Dies hat dann auch die Wiederholung der Maßnahmen zur Folge. Wurde auch das Setzen der Version im Eeprom korrekt durchgeführt, ist der Watchdog zu initialisieren, um einen anschließenden Systemreset herbeiführen zu können. Scheitert der und wird übersprungen, erfolgt wieder die Schaltung in den Sicherheitszustand.
Jede Fehlfunktion wird über den Funktions-Rückgabewert von ‚1' signalisiert.

Save_Parameter():

Zur Ablage der Software-Version als Long-Wert im Eeprom wird dieser Funktion eine case-Anweisung (*LOAD_SAVE_VERSION*) zugefügt. Durch den Aufruf von *Version_SoftwareVersionToLong()* gibt diese die aktuelle Firmware-Version als 4byte-Wert zurück. Alsdann erfolgt über die Bibliotheksfunktion *Ee24c64_Write()* mit der Eeprom-Adresse 1048_{dez} (Definition *EEPROM_VERSION* in *Load_Save.def)* als erster Parameter,der Datenlänge von 4Byte als zweiter Parameter und dem Zeiger auf die die Firmware-Version haltende Variable als dritten Parameter, die Abspeicherung der Daten ins Eeprom.
Im Falle eines Fehler erfolgt die Rückgabe ungleich 0.

Version_SoftwareVersionToLong():

Diese neue Funktion wandelt den als String im Codespeicher abgelegten Versionsstring (*TEXT_SOFTWARE_VERSION*) in einen 4byte-Wert um.
Als Ausgangsbasis ist die Zielvariable auf 0 zu setzen.
Nacheinander werden zeichenweise von der zweiten linken Stelle beginnend sechs Zeichen eingelesen. Handelt es sich dabei um eine Zahl (ASCII-Wert zwischen 0x30 und 0x39 (einschließlich)), wird zuerst die Zielvariable mit 10 multipliziert und dann der Zahlenwert, dem Zeichen entsprechend, hinzuaddiert.
Der Rückgabewert der Funktion ist der ermittelte Versionswert.

Beispiel: *TEXT_SOFTWARE_VERSION* = „V2.000"

1. „2" => 4byte-Version = 2
2. „." => 4byte-Version = 2
3. „0" => 4byte-Version = 20
4. „0" => 4byte-Version = 200
5. „0" => **4byte-Version = 2000**

3 Kommentare

4 Anhang