07.06.2023
Version 1.13

DOKUMENTENMANAGEMENT

Projektbezeichnung	Unmanned Surface Vehicle	
Projektleiter	Jörg Grabow	
Verantwortlich	Jörg Grabow	
Erstellt am	10.02.2020	
Zuletzt geändert	07.06.2023	
Bearbeitungsstand	i.B.	in Bearbeitung vorgelegt fertig gestellt
Dokumentenablage	https://github.com/Joe-Grabow/USV	

Änderungsverzeichnis

Änderung			geänderte Kapitel	Beschreibung	Autor	neuer Zustand
Nr.	Datum	Version				
1	10.02.20	1.00	-	Startversion	Gr.	f.g.
2	20.02.20	1.01	Leitungsaufbau	Tab. 2 mit daisy chain eingeführt	Gr.	f.g.
3	21.02.20	1.02	Slave	Slavefunktionen erweitert	Gr.	f.g.
4	12.11.20	1.03	Leitungsaufbau	Pinbelegung CAT-6	Gr.	f.g.
5	18.11.20	1.04	Datenblock	Speicherblock für Datenaustausch	Gr.	i.B.
6	24.11.20	1.05	Datenblock	Ruderausschlag / GPS-Geschwindigkeit	Gr.	i.B.
7	27.11.20	1.06	Datenblock	Schubnormierung auf +-1; signed Fixp.	Fr.	i.B
8	09.12.20	1.07	Datenblock	SatFix, Error-Byte lokal / global	Gr.	i.B.
9	17.12.20	1.08	Datenblock	Timestamp, GPS	Gr.	i.B.
10	21.12.20	1.09	Anhang 3	ATMega328PB Watchdog	Gr.	f.g.
11	22.12.20	1.10	Anhang 3	Watchdog-Timer	Gr.	f.g.
12	08.01.21	1.11	Datenstruktur	Adresse Ruderausschlag	Gr.	f.g.
13	13.01.21	1.12	Datenstruktur	Q-Number Format	Gr.	f.g.
14	03.02.21	1.13	Datenstruktur	Adresse Stellgrößen	Gr.	f.g.

in Bearbeitung (i.B.)
Vorlage (Vg.)
fertig gestellt (f.g.)

Inhaltsverzeichnis

	Kapitel	Verweise auf andere Dokumente
RS485-Bus	1.0	
Master-Slave System	2.0	
Hardwarerealisierung	3.0	
Datenstruktur	4.0	
Q-Number Format	Anhang 1	
Geokoordinaten	Anhang 2	
RTC Code	Anhang 3	
AVR Watchdog-Timer	Anhang 4	

0. Einleitung

Für eine weitgehend autonome Funktion der Bootsplattform sind eine Reihe von Sensoren und Aktuatoren notwendig. Um diese modular in das Gesamtkonzept zu integrieren, wird ein Bussystem auf der Basis einer Zweidrahtschnittstelle (RS485 – halbduplex) vorgeschlagen. Das folgende Dokument beschreibt die dazu notwendigen Überlegungen.

1. Der RS485-Bus

RS-485 bezeichnet, ist ein Industriestandard für eine physische Schnittstelle für die asynchrone serielle Datenübertragung. Eine symmetrische Leitungsführung erhöht die elektromagnetische Verträglichkeit. RS-485 benutzt ein Leitungspaar, um den invertierten und einen nichtinvertierten Pegel eines 1-Bit-Datensignals zu übertragen. Am Empfänger wird aus der Differenz der beiden Spannungspegel das ursprüngliche Datensignal rekonstruiert. Das hat den Vorteil, dass sich Gleichtaktstörungen nicht auf die Übertragung auswirken und somit die Störsicherheit vergrößert wird. Im Gegensatz zu RS-232 sind so wesentlich längere Übertragungsstrecken und höhere Taktraten möglich. Gegenüber dem RS-422-Standard besitzen die Sender durch einen integrierten Widerstand kurzschlussfeste Ausgangsstufen, so dass auch ein Gegensenden zweier Sender nicht zu Defekten führt. An einem Adernpaar dürfen außerdem mehrere Sender und mehrere Empfänger angeschlossen sein („Multipoint“).

1.1 Technik

Die symmetrische Leitung der RS-485-Schnittstelle arbeitet senderseitig mit Signalspannungen von mindestens ±1,5 V, höchstens ±6 V. Typischerweise verwendet der Treiber eine Brückenschaltung, sodass die Signalamplitude knapp unter der Betriebsspannung von z. B. 5 V oder 3,3 V liegt. Die Schaltschwelle des Empfängers muss im Bereich ±0,2 V liegen. Im Gegensatz zur massebezogenen RS-232-Schnittstelle ist durch den symmetrischen Aufbau der Signalleiter ein RS-485-Empfänger gegenüber Gleichtaktstörungen weitgehend unempfindlich.

Die Schnittstelle benutzt in der Regel nur ein Adernpaar und wird halbduplex betrieben, mit zwei Aderpaaren ist Vollduplexbetrieb möglich. Die Verbindung ist multipointfähig, das heißt, es können bis zu 32 Teilnehmer an den RS-485-Bus angeschlossen werden. Üblicherweise werden Kabellängen bis zu 1,2 km und Übertragungsraten bis 10 MBit/s unterstützt, wobei die maximale Übertragungsrate nur bei Leitungslängen bis zu 12 m erreicht wird. Die tatsächlich mögliche maximale Netzwerkgröße und die maximale Übertragungsrate sind außerdem stark vom Aufbau des Netzwerks abhängig. Sterntopologien sind nicht vorgesehen, üblich ist der Aufbau in Kettenform (Daisy Chain).

Da die RS-485-Schnittstelle ein Bussystem darstellt, sollten die Leitungsenden (zumindest bei größeren Leitungslängen bzw. größeren Übertragungsraten) abgeschlossen werden. In der Regel wird ein passiver Abschluss durch Verbinden der Signalleitungen über jeweils einen 120-Ω-Widerstand an den beiden Busenden verwendet. Ein optionales Bias-Netzwerk (die 720-Ohm-Widerstände) verbessert für den Fall inaktiver Leitungstreiber durch eine von null verschiedene Spannung den Störabstand, der sonst lediglich durch die Hysterese des Empfängers gegeben ist.

Bei großen Leitungslängen oder abweichenden Erdpotenzialen kann es zu größeren Potentialdifferenzen zwischen den Busteilnehmern kommen, die oberhalb der Gleichtaktunterdrückung liegen und daher die Kommunikation behindern. Das kann durch Mitführen der Masseleitung verbessert werden.

1.2 Leitungsaufbau

Die symmetrische Leitungsführung der RS485-Schnittstelle wird durch handelsübliche Twisted-Pair-Kabel (CAT-6, CAT-7) realisiert. Diese Twisted-Pair-Kabel enthalten 4 Adernpaare aus je zwei miteinander verdrillten Paaren von Einzeladern (Abb. 1).

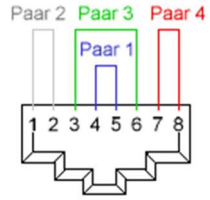

Abb. 1: CAT-6 Buchse

Da die Stromversorgung der Slavemodule wenn möglich über die Busleitung erfolgen soll, werden 2 oder 3 Adernpaare ausschließlich dafür verwendet.

Pin	Paar	Funktion
1	2	+ 12V
2	2	GND
3	3	+ 12V
4	1	A
5	1	B
6	3	GND
7	4	+ 12V
8	4	GND

Tab. 1: 3 x Power

Pin	Paar	Funktion
1	2	+12V / +24V
2	2	GND
3	3	B+
4	1	A+
5	1	A-
6	3	B-
7	4	+ 12V / +24V
8	4	GND

Tab. 2: 2 x Power, daisy chain

2. Master-Slave System

Der hohe Modularisierungsgrad wird durch ein Master-Slave System realisiert. Ein Teilnehmer ist der (Bus)Master, alle anderen sind die Slaves. Der Master hat als einziger das Recht, unaufgefordert auf die gemeinsame Datenbasis (Sensor-, Kommunikations- und Steuerdaten) zuzugreifen. Der Slave kann von sich aus nicht auf die gemeinsame Datenbasis zugreifen; er muss warten, bis er vom Master abgefragt wird (Polling) oder die Datenbasis übermittelt bekommt. Dazu führt der Master zyklisch die beiden Funktionen ***Polling*** und ***Broadcast*** aus.

2.1 Initialisierung

Der Master führt eine Initialisierung durch und trägt alle derzeit im Bussystem aktiven Sensoren in eine Sensortabelle ein. Jeder Sensor hat eine feste, durch die Hardware vorgegebene, Sensornummer.

2.2 Polling

Master fragt alle Sensoren, welche in einer Sensortabelle hinterlegt sind, zyklisch ab und hinterlegt die empfangenen Sensordaten in einer Datei. Antwortet ein aktiver Sensor nicht mehr, wird er nach einem Timeout aus der Sensortabelle gelöscht. Zyklisch (in größeren Zeitabständen) wird die Initialisierungsroutine wiederholt, um neue Sensoren oder vermeintlich fehlerhafte Sensoren wieder aufzunehmen.

2.3 Broadcast

Nach einem vollständig durchgeführten Pollingzyklus wird die *gesamte* erfasste Sensordatei über Broadcast an alle Aktoren übermittelt. Somit können sehr schnell und effizient alle Informationen an

Aktor- und Kommunikationseinheiten übertragen werden. Der jeweilige Aktor entnimmt dabei immer nur die für ihn relevanten Informationen aus der Sensordatei.

3. Hardwarerealisierung

3.1 Master

Der Master besteht aus einer Mikrocontroller-Baugruppe, welche über einen geeigneten Hardwaretreiber die RS485-Busfunktionalität bereitstellt. Per Software werden die Funktionen

- Initialisierung
- Polling
- Broadcast

realisiert.

3.2 Slave

Der Slave besteht aus einer Interface-Baugruppe und weiteren funktionsabhängigen Baugruppen. Die Interface-Baugruppe stellt zum einen die Hardwaretreiber für die RS485-Busfunktionalität bereit, zum anderen die Grundfunktionen für das Polling und den Broadcastempfang. Somit können alle Slave Module, unabhängig von ihrer konkreten Funktion, mit einheitlichen Interface-Baugruppen ausgestattet werden. Die Kommunikation zwischen der Interface-Baugruppe und den unktionsabhängigen Slave-Baugruppen (User-Unit) erfolgt über RS232. Weiterhin werden die Spannungen 12V, 5V und 3.3V für eigene Anwendungen bereitgestellt.

3.3 Monitor

Der Monitor realisiert eine Zusatzbaugruppe, welche die drei folgenden Funktionen realisiert.

- „Sniffen“ der Buskommunikation und Weiterleitung über USB
- Simulation eines Sensors
- Simulation eines Aktors

Damit kann in der Entwicklungs- und Servicephase die komplette Buskommunikation überwacht werden. Weiterhin lassen sich schon im Vorfeld geplante Sensoren und Aktoren auf ihre Integration im Gesamtsystem testen.

4. Datenstruktur

Der Datenblock umfasst eine Größe von **XXX** Byte und ist sowohl im RAM-Bereich des Masters als auch des Slave hinterlegt. Über eine slavespezifische ID erfolgt die Zugriffssteuerung auf den Datenblock. Eine Sensorbaugruppe kann prinzipiell nur lesend auf den Datenblock zugreifen. Eine Aktorbaugruppe kann schreibend auf genau den Bereich des Datenblocks zugreifen, der durch seine ID spezifiziert ist. Die folgende Tabelle zeigt eine Übersicht über den Datenblock, seine Adressbereiche und die entsprechenden Adressinhalte.

Adresse	Bytes	Funktion	Beschreibung	Format
			Sensorblock	
0x000	1	GESB	globales Error Status Byte (GESB)	UQ8.0
0x001	4	Längengrad	GPS-Koordinate in DG[1] (Dezimalgrad WGS 84)	Q9.17
0x005	4	Breitengrad	GPS-Koordinate in DG (Dezimalgrad WGS 84)	Q9.17
0x009	1	SatFix	Anzahl der gefixten Satelliten	UQ8.0
0x00A	2	Geschwindigkeit	GPS-Geschwindigkeitsangabe in m/s	UQ8.8
0x00C	2	Kurswinkel	Kurswinkel in Grad (von Nordrichtung im Uhrzeigersinn)	UQ9.7
0x00E	3	Timestamp	Zeitstempel	RTC-Code
			Führungsgrößen der Antriebsregelung	
0x100	8	Länge, Breite	Führungsgröße Punkt A (4 Byte Länge + 4 Byte Breite)	Q9.17
0x108	8	Länge, Breite	Führungsgröße Punkt B (4 Byte Länge + 4 Byte Breite)	Q9.17
0x110	2	Geschwindigkeit	mittlere Geschwindigkeit zwischen A und B	UQ8.8
0x112	2	Epsilon	Epsilon Schlauch	UQ8.8
			Stellgrößen der Antriebsregelung	
0x120	2	Schub	Stellgröße Schub (+/- 1)	Q1.15
0x122	2	Ruderausschlag	Stellgröße Ruderausschlag (+/- 1)	Q1.15
			lokaler Error Block	
0x200	1	ESB GPS	Error Status Byte für GPS-Slave	UQ8.0
0x201	1	ESB Kompass	Error Status Byte für Kompass-Slave	UQ8.0
0x202	1	ESB Control	Error Status Byte für Regler-Slave	UQ8.0

Tab. 3: Aufbau Datenblock

Vorzeichenregelung für GPS-Koordinaten

Richtung	Vorzeichen
Nord	positiv
Süd	negativ
Ost	positiv
West	negativ

[1] Umrechnung von Geokoordinaten im Anhang 2

Anhang 1 – Q-Number Format

Das Q-Number Format ist ein binäres Fixpunktformat bei der die Anzahl der Vor- und Nachkomma Bits spezifiziert ist. So hat z.B. eine Q3.7-Zahl 3 Vorkomma-Bits und 7 Nachkomma-Bits. Das Q-Format wird häufig in Prozessoren ohne Gleitkommaeinheit verwendet.
Q-Zahlen werden prinzipiell im Zweierkomplement nach der folgenden Bildungsvorschrift gespeichert:

Q-Zahl ohne Vorzeichen

UQm.n	U	unsigned
	m	Anzahl der Bits vor dem Komma
	n	Anzahl der Bits nach dem Komma

Bsp.

UQ5.3	$m = 5\,;\, n = 3$
MSB	$2^{m} = 32$
LSB	$2^{-n} = 0.125$
Bitanzahl	$m + n = 8$
Minimum	0
Schrittweite	$2^{-n} = 0.125$
Maximum	$2^{m} = 32$
Wertebereich	$\left[0, 2^{-n}, 2^{m}\right]$

Q-Zahl mit Vorzeichen

Qm.n		signed
	m	Anzahl der Bits vor dem Komma
	n	Anzahl der Bits nach dem Komma

Bsp.

Q15.1	$m = 15\,;\, n = 1$
MSB	$2^{m-1} = 16384$
LSB	$2^{-n} = 0.5$
Bitanzahl	$m + n = 16$
Minimum	$-2^{m-1} = -16384$
Schrittweite	$2^{-n} = 0.5$
Maximum	$2^{m-1} - 2^{-n} = 16383.5$
Wertebereich	$\left[-2^{m-1}, -2^{-n}, 2^{m} - 2^{-n}\right]$

Anhang 2 – Umrechnung von Geokoordinaten

Formate für Geo-Koordinaten sind u.a. Grad, Minute und Sekunde (z.B. 52° 7′ 30,9″) oder Grad und Dezimalminute (z.B. 52° 7,515′) oder Dezimalgrad (z.B. 52,12525°). Verschiedene GPS-Geräte können Geo-Koordinaten in unterschiedlichen Formaten liefern. Auch für verschiedene Kartendarstellungen können die Koordinaten in verschiedenen Formaten benötigt werden.

Umrechnung von Grad, Minute und Sekunde in Dezimalgrad (DG)

Berechnungsvorschrift

- Sekunden durch 60 dividieren
- Minuten hinzuaddieren
- Ergebnis durch 60 dividieren
- Grad hinzuaddieren

Beispielkoordinate: 50° 39′ 54,021″ Nord

Ergebnis in DG:

$$50^o\ 39'\ 54{,}021'' = \frac{\frac{54{,}021''}{60} + 39'}{60} + 50^o = 50.66500583$$

Umrechnung von Dezimalgrad (DG) in Grad, Minute und Sekunde

Berechnungsvorschrift

- der ganzzahlige Anteil des Dezimalgrads entspricht dem Grad-Wert
- den Nachkommaanteil mit 60 multiplizieren
- der ganzzahlige Anteil entspricht dem Minuten-Wert
- den Nachkommaanteil mit 60 multiplizieren
- das Ergebnis entspricht dem Sekunden-Wert

Beispielkoordinate: 11.120563900937652 Ost

Ergebnis in G, M, S:

$$\text{Grad} = 11$$
$$x = (11.1205639 - Grad) \cdot 60 = 7.233834$$
$$\text{Minute} = 7$$
$$\text{Sekunde} = (x - \text{Minuten}) \cdot 60 = 14.03$$
$$\text{Ergebnis} = 11^o 07' 14.03''$$

Anhang 3 – RTC Code

Adr.	Funktion	Bereich	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x00E	Sekunde	0-59	0	10 Sekunden			Sekunden			
0x00F	Minute	0-60	0	10 Minuten			Minuten			
0x010	Stunde	0-24	0	0	10 Stunden		Stunden			

Anhang 4 – ATMega328PB Watchdog

Der Watchdog enthält einen eigenen Watchdog Oszillator (Abb. A3.1), welcher mit einem intern erzeugten Takt von 128 kHz getaktet wird. Nachdem der Watchdog aktiviert und der gewünschte Vorteiler eingestellt wurde, beginnt der Counter von 0 an hochzuzählen. Wenn die je nach Vorteiler eingestellte Anzahl Zyklen erreicht wurde, löst der Watchdog einen Reset aus. Um im Normalbetrieb den Reset zu verhindern, muss der Watchdog regelmäßig wieder neu gestartet bzw. rückgesetzt werden. (Watchdog Reset). Das Watchdog Reset sollte innerhalb der Hauptschleife ausgeführt werden.

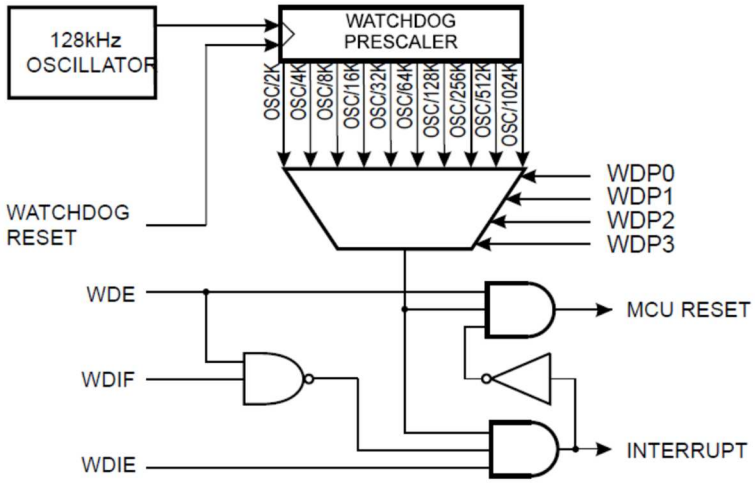

Abb. A3.1: Watchdog-System des ATMega328PB

Bleibt das Watchdog Reset aus, wird je nach Einstellung, nach einer einstellbaren Zeit ein Reset durchgeführt, ein Interrupt ausgelöst oder beides. Ein Interrupt vor dem Reset kann zum Beispiel dazu genutzt werden, um:

- sichere Bedingungen bei der Baugruppe herzustellen
- Parameter auf einem EEPROM zum Zwecke der Fehleranalyse zu speichern
- den Stand des Programmes zu speichern, damit es nach dem Reset an der richtigen Stelle seine Arbeit fortsetzt

Die Steuerung des Watchdog erfolgt über das ***WDTCSR*** – Watchdog Timer Control Register. Um ein unbeabsichtigtes Ausschalten des Watchdogs zu verhindern, muss ein spezielles Prozedere verwendet werden, um den WD auszuschalten. Dazu müssen zuerst die beiden Bits WDTOE und WDE in einer einzelnen Operation jeweils auf 1 gesetzt werden. Dann muss innerhalb der nächsten 4 Taktzyklen das Bit WDE auf 0 gesetzt werden.

Bit No.	7	6	5	4	3	2	1	0
Identifer	**WDIF**	**WDIE**	**WDP3**	**WDCE**	**WDE**	**WDP2**	**WDP1**	**WDP0**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

WDTCSR-Register

WDIF: Watchdog Interrupt Flag – das Bit wird bei Time-Out des Watchdog Timers gesetzt, wenn zuvor auch WDIE gesetzt wurde

WDIE: Watchdog Interrupt Enable – bei gesetztem Bit löst ein Time-Out des Watchdog Timers einen Interrupt aus

WDPx: Watchdog Timer Prescaler – legt die Periode des Watchdog Timers entsprechend der nachfolgenden Tabelle Tab. A3.1 fest

WDCE: Watchdog Change Enable – Bit zur Änderung der Watchdog Timer Einstellungen

WDE: Watchdog System Reset Enable – bei gesetztem Bit löst ein Time-Out des Watchdog Timers ein Reset des Microcontrollers aus.

Tab A3.1

WDP3	WDP2	WDP1	WDP0	Osc. Cycles	Time Out
0	0	0	0	2K	16 ms
0	0	0	1	4K	32 ms
0	0	1	0	8K	64 ms
0	0	1	1	16K	0.125 s
0	1	0	0	32K	0.25 s
0	1	0	1	64K	0.5 s
0	1	1	0	128K	1 s
0	1	1	1	256K	2 s
1	0	0	0	512K	4 s
1	0	0	1	1024K	8 s

Achtung! Das Setzen von WDCE und Änderungen am Prescaler dürfen nicht zeitgleich erfolgen. In einem Befehl muss gleichzeitig eine 1 in die Bits WDCE und WDE geschrieben werden. Die 1 muss unabhängig vom vorherigen Zustand von WDE in WDE geschrieben werden. Innerhalb der nächsten vier Taktzyklen muss die 0 in das WDE-Bit oder die neuen Prescaler-Bits WDP[3:0] geschrieben werden.

Bsp. 1: Watchdog Timer mit System Reset

```
disable_interrupt();                          // Interrupt sperren
watchdog_reset();                             // Watchdog-Timer Reset
WDTCSR |= (1<<WDCE) | (1<<WDE);               // WDCE und WDE setzen, Prescaler nicht ändern
WDTCSR = (1<<WDE) | (1<<WDP3);                // Timeout 4s, no Interrupt, System Reset
enable_interrupt();                           // Interrupt zulassen
```

Bsp. 2: Watchdog Timer mit Interrupt und anschließendem System Reset

```
disable_interrupt();                          // Interrupt sperren
watchdog_reset();                             // Watchdog-Timer Reset
WDTCSR |= (1<<WDCE) | (1<<WDE);               // WDCE und WDE setzen, Prescaler nicht ändern
WDTCSR = (1<<WDIE) | (1<<WDE) | (1<<WDP3);    // Timeout 4s, Interrupt, System Reset
enable_interrupt();                           // Interrupt zulassen
```

Bsp. 3: Abschalten des Watchdog Timers

```
disable_interrupt();                          // Interrupt sperren
watchdog_reset();                             // Watchdog-Timer Reset
MCUSR &= ~(1<<WDRF);                          // löscht WDRF
WDTCSR |= (1<<WDCE) | (1<<WDE);               // WDCE und WDE setzen, Prescaler nicht ändern
WDTCSR = 0x00;                                // Watchdog-Timer aus
enable_interrupt();                           // Interrupt zulassen
```

Watchdog Timer „immer an“ mit dem Fuse Bit WDTON

Der Watchdog Timer kann auch dauerhaft über das Fuse Bit **WDTON** aktiviert werden. In diesem Fall ist der reine ***Reset Modus*** ohne Watchdog Interrupt aktiviert. In diesem Modus kann nur die Time-Out Zeit geändert werden.

Bei der Verwendung des Watchdog-Timers ist eine weitere Besonderheit zu beachten!
Löst der Watchdog Timer ein Systemreset aus, so werden dabei auch die Prescaler Register gelöscht, der Prescaler steht also auf 16 ms. Da der Watchdog Timer nach dem Reset oder einem Flashvorgang weiter läuft (!) kann es passieren, dass die Initialisierungsroutinen des Hauptprogramms nie angesprungen werden, da vorher immer ein erneutes Reset ausgelöst wird. Der Programstart muss also zwingend mit der Deaktivierungsroutine des WDT beginnen.