ST-Bus Protokoll V3.6.4 (BETA)

11.10.09

Grundlagen:

- **2 Draht-Verbindung (RS485 57600 BPS 8N1 halbduplex)**
- **Bis zu 64 Busteilnehmer**
- **Multimasterfähig**
- **255 Adressen (eine Broadcastadresse)**
- **Jedes Paket hat die Länge 16 Byte**
- **Fehlererkennung mit CRC8**

Aufbau eines Datenpaketes:

"Request" Data Block

Token	Bus Adr. Quelle	Bus Adr. Ziel	Dta. Adr. high	Dta. Adr low	Data[0] high	Data[0] low		CRC8

"Reply" Data Block

Return Code	Bus Adr. Quelle	Bus Adr. Ziel	Dta. Adr. high	Dta. Adr low	Data[0] high	Data[0] low		CRC8

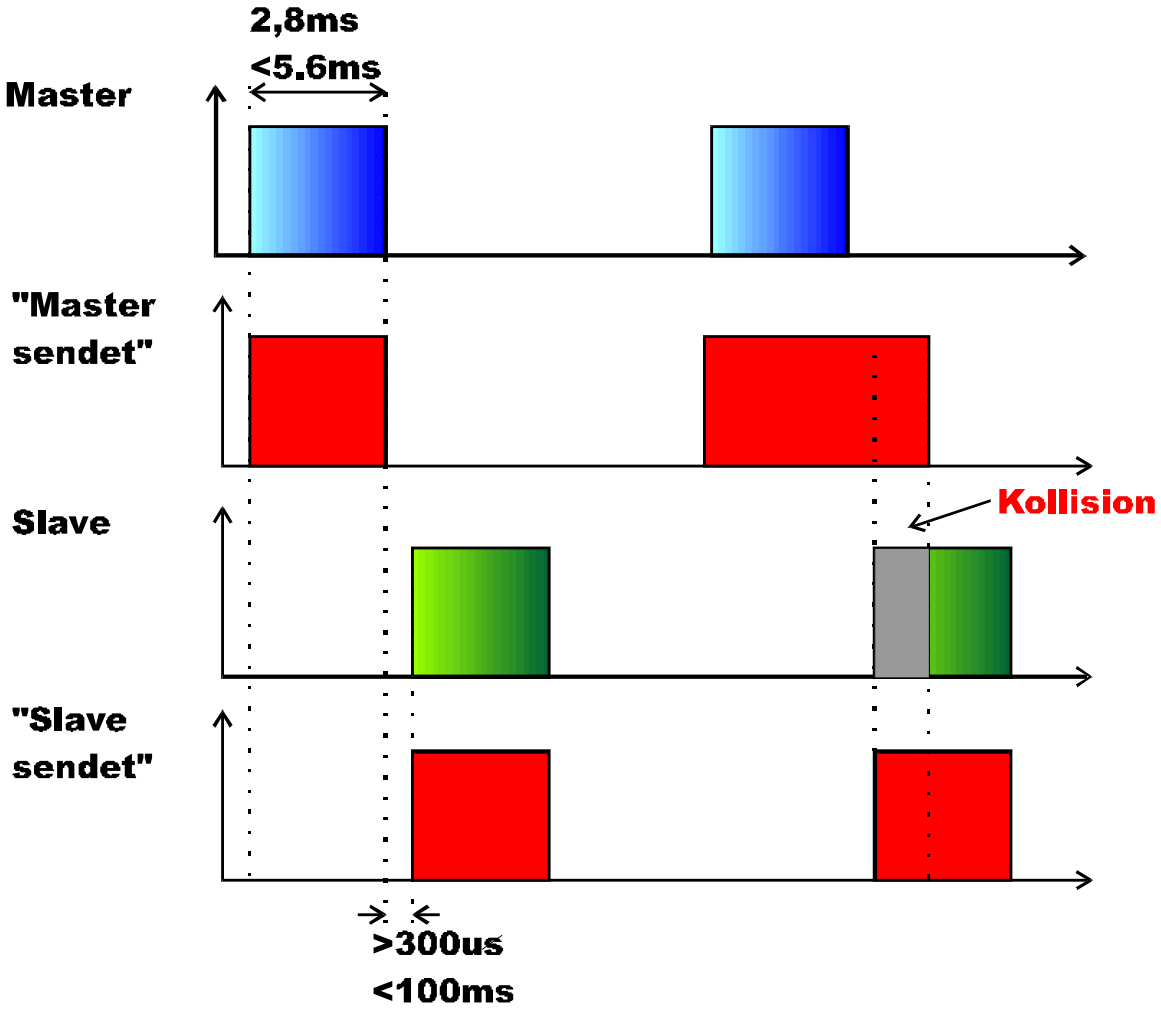
- **Verwaltungsinfo**
- **Nutzdaten**
- **Sicherungsdaten**

Übertragungsdauer eines Pakets ca. 2.7ms @ 57600 BPS

Definition:
Master: kann von sich eine neue Kommunikation beginnen.
Slave: reagiert nur auf externe Anfragen.

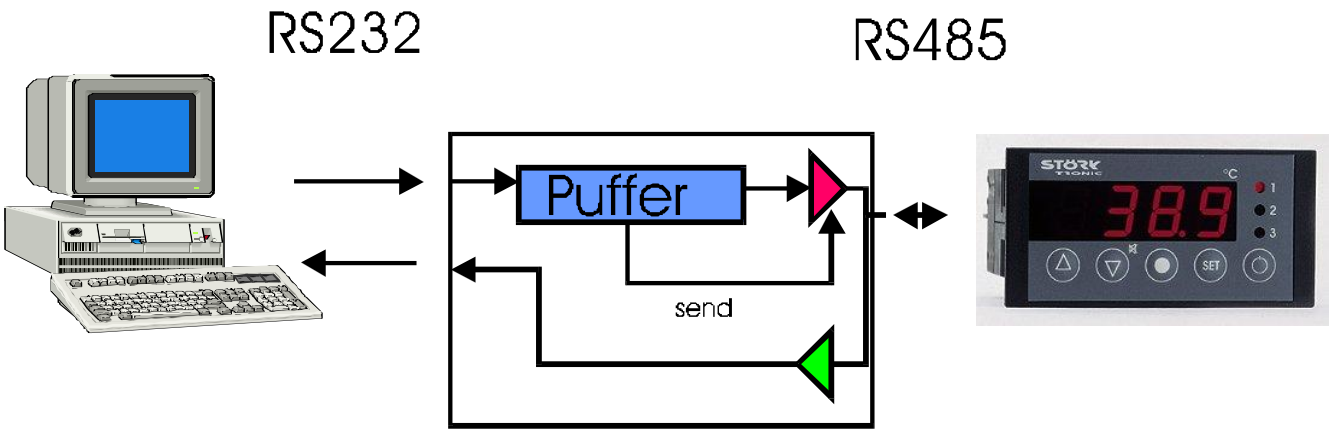
Kommunikationsgrundlagen

Die Übertragung erfolgt im Halbduplexverfahren, d.h. es kann immer nur ein Busteilnehmer senden. Der Sender muß sofort - nachdem er seinen Datenblock verschickt hat – die Leitung wieder freigeben. Ein anderer Busteilnehmer wird frühestens in 300us antworten (maximal dürfen 1ms vergehen):



Problematik der PC Anbindung

Obige Darstellung zeigt im ersten Fall eine „saubere“ Kommunikation, im zweiten Fall eine Kollision, die durch einen Master verursacht wurde, der seine Sendetreiber zu lange eingeschaltet hat. In dem Fall kann das Antwortpaket nicht korrekt gelesen werden (oder verschwindet ganz).
Bei der Ankopplung an einen PC entsteht dieses Problem: Die Sendetreiber werden eingeschaltet, das Datenpaket wird dem Schnittstellentreiber übergeben und abgeschickt.
Nachdem alle Daten verschickt wurden, wird ein IRQ ausgelöst und der Sendetreiber abgeschaltet.
Da in der Regel zwischen IRQ Auslösung und Reaktion des Betriebssystems eine Zeit >300us vergeht, ist diese Art der Ankopplung äußerst kritisch.
Als Lösung wird ein Businterface angeboten:



Das Interface speichert jeweils einen gesamten Datenblock zwischen und fängt erst dann mit der Übertragung an, wenn dieser vollständig im Speicher steht. Das Einschalten und Ausschalten des Sendetreibers wird vollständig vom Interface gesteuert und ist somit unabhängig von Verzögerungen auf der PC Seite.

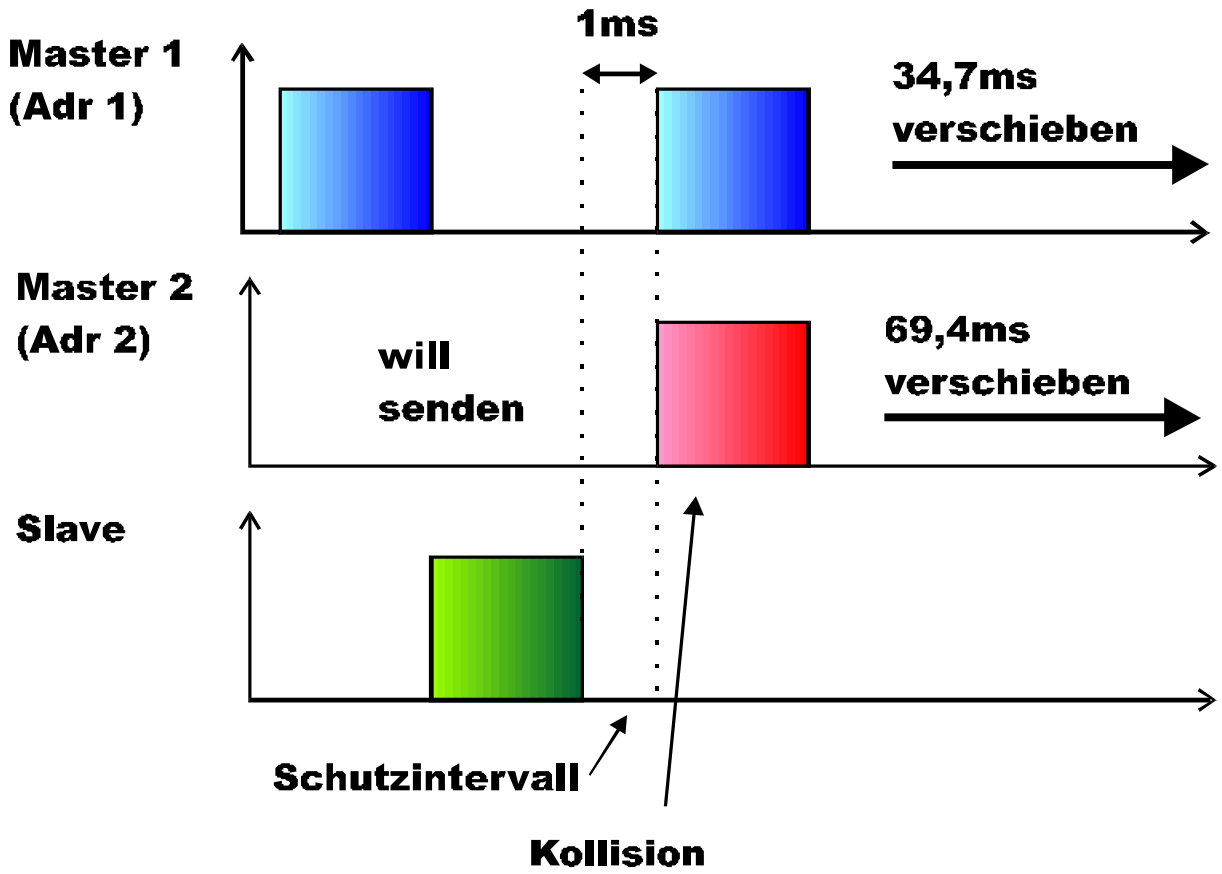
Kollisionserkennung

(siehe Ethernet „CSMA/CD“ =Carrier Sense Multiple Access With Collision Detection Access)

Aufgrund der Schnittstellenbeschaltung werden alle gesendeten Daten permanent auch zurückgelesen; auf diese Weise ist eine Kollisionserkennung realisierbar.

Es gelten folgende Gesetzmäßigkeiten:

- **Wird eine Kollision erkannt, wird sofort die Übertragung abgebrochen und ein erneuter Sendeversuch erst nach ca. 34,7ms * EIGENE_ADRESSE durchgeführt.**
- **(ein Slave unternimmt keinen selbständigen Versuch, die Übertragung wieder aufzunehmen).**
- **Der Busverkehr wird permanent mitgehört. Der Master darf erst zu senden anfangen, wenn für mindestens 1ms lang kein Busverkehr stattfand („Schutzintervall“)**



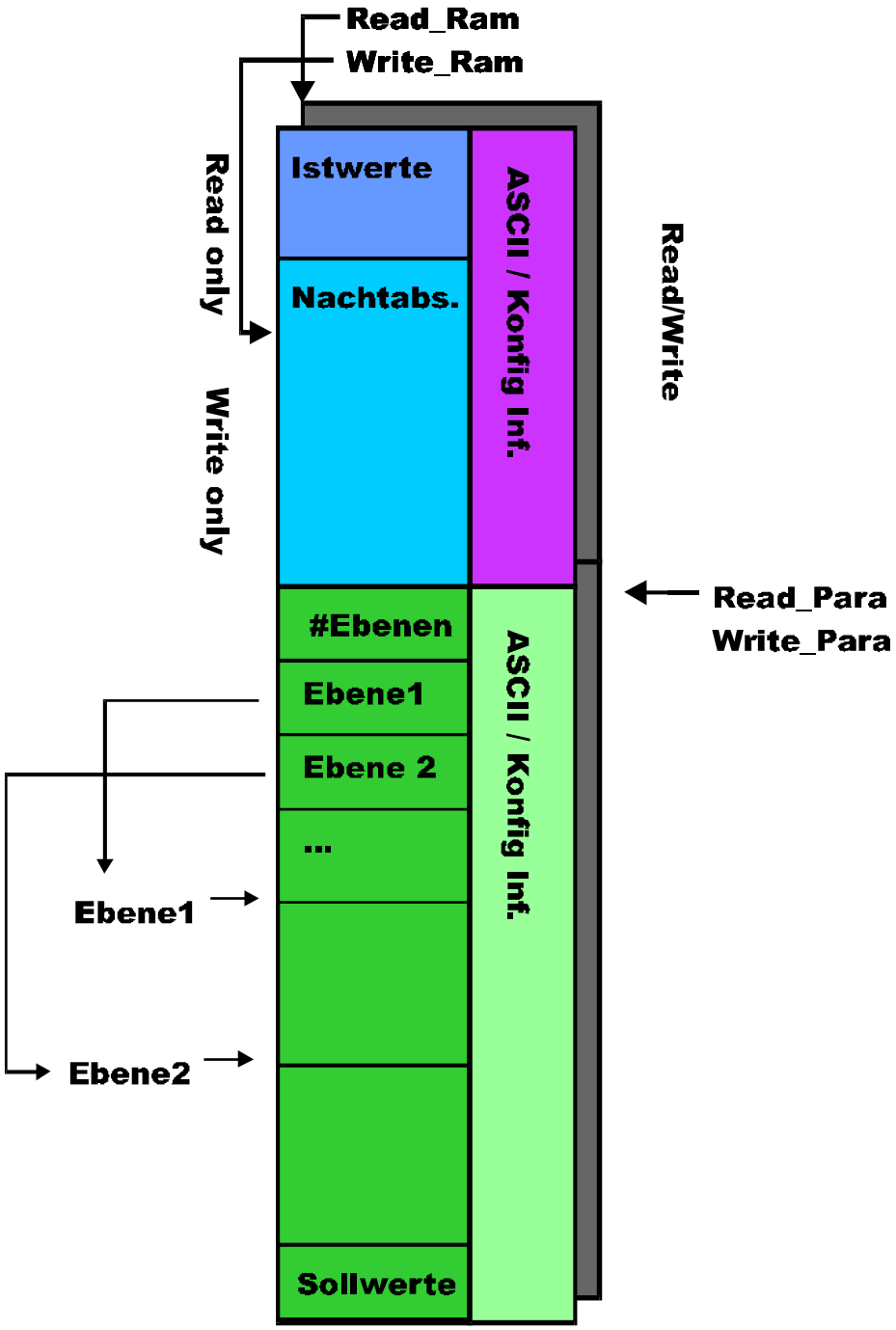
Master 2 will senden, muß aber den Ablauf des 1ms Schutzintervalls abwarten. Eine Kollision passiert dann, wenn zwei Master innerhalb des ersten Übertragunsbytes zu senden anfangen (da der Bus erst nach Senden eines vollst. Bytes als belegt erkannt werden kann).

Protokoll (Grundsätzliches)

Der Zugriff auf den Regler erfolgt über definierte Pseudospeicherzellen. Diese Pseudospeicherstellen sind in Rambereich und Rombereich aufgeteilt (mit entsprechend unterschiedlichen Zugriffsbefehlen). „Ram“ bezeichnet hierbei einen Speicherbereich, der teilweise nur gelesen oder geschrieben werden kann aber in der Regel indirekt Zugriff auf Ramspeicherzellen des Reglers erlaubt.
„Para“ steht für den Parameterspeicher, der die eingestellten Werte permanent hält. Auf ihn kann sowohl lesend als auch schreibend zugegriffen werden.
Der Parameterspeicher teilt sich eine veränderliche Anzahl von Ebenen auf. Das erste Element des Parameterspeichers enthält die Gesamtzahl der Ebenen. Es schließt sich jeweils pro Ebene ein Datenwort an, das die Länge dieser Ebene enthält. Auf diese Weise ist ein direktes Zugreifen auf jede Ebene möglich.
Alle Sollwerte werden in der letzten Ebene zusammengefaßt.
Die Startadresse einer Ebene ergibt sich somit folgendermaßen:

Startadresse = Anzahl Ebenen + Summe der Längen aller vorherigen Ebenen +1

Diese Startadresse zeigt dann auf das Passwort für diese Ebene.


Read_Ram
Write_Ram
Istwerte
Nachtabs.
ASCII / Konfig Inf.
Read only
Write only
Read/Write
Read_Para
Write_Para
#Ebenen
Ebene1
Ebene 2
...
ASCII / Konfig Inf.
Ebene1
Ebene2
Sollwerte

Das erste Element jeder Ebene enthält das Passwort für diese. Steht hier ein Wert "0" ist die Ebene frei. Das Passwort wird nicht im Slave, sondern im Master überprüft. Das Passwort wird von der Slave Seite wie ein normaler Parameter behandelt und kann über das Netzwerk folglich auch beschrieben werden. Die abfragbare Ebenenlänge schließt dieses Passwort mit ein!
Die "Anzahl Parameter" schließt nicht die Ebeneninfo ein (Anzahl Parameter = Summer der Parameter aller Ebenen). Sollwerte werden ebenfalls hier nicht berücksichtigt.

Beispiel:

Speicher-Adresse	*Wert*	*Kommentar*
0	7	# Ebenen (7)
1	15	# Elemente Ebene 1
2	8	# Elemente Ebene 2
3	30	# Elemente Ebene 3
4	20	# Elemente Ebene 4
5	23	# Elemente Ebene 5
6	6	# Elemente Ebene 6
7	3	# Elemente Ebene 7
8	0	Passwort Ebene 1
9	...	1. Element Ebene 1
...	...	[...]
23	42	Passwort Ebene 2
...	...	[...]
113	123	Sollwert 1
[...]	[...]	[...]

Anzahl Parameter: 105

Übersicht der Token (Befehle)

Token-Nummer		Token-Text
#	hex	
0	00	Read_Para_1
1	01	Read_Para_2
2	02	Write_Para
3	03	Read_Ram
4	04	Write Ram
5	05	Read_Number
6	06	Set_Relais
7	07	Write EEprom
8	08	Read EEprom
9	09	Write Data (auto-Inkrement)
10	0A	Read Data (burst)
11	0B	Read Time
12	0C	Set Time
13	0D	Read Version / Info
14	0E	Read Generic 1
15	0F	Read Generic 2
16	10	Write Generic
17	11	Search String
18	12	Start Test
19	13	
20	14	Read Data Info
21	15	ReadRamBurst (*15)
22	16	
23	17	FreezeTime (Datalogger)
24	18	BusVersion (16*)
25	19	ReadStatus (12*)
26	1A	ReadRamDebug (*19)
27	1B	
28	1C	
29	1D	Logger (V2.0)
30	1E	
31	1F	
32	20	ClearStatus (12*)
33	21	SetStatus (12*)
34	22	Ping (nur über Broadcast an alle!)
35	23	Shut Up! (nur Broadcast) (17*)
36	24	Wake Up! (nur Broadcast) (17*)
37	25	
38	26	
39	27	
61	3d	Reserviert für Bootloader
62	3e	Reserviert für Textdownload
63	3f	Reserviert für erweiterte Gatewayfunktionen

Befehle in rot müssen implementiert werden!

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Token	Bus Adr Quelle	Bus Adr Ziel	Dta Adr high	Dta Adr low	Data high [0]	Data low[0]	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]	CRC8
Antwort	Bus Adr Quelle	Bus Adr Ziel	Dta Adr high	Dta Adr low	Data high [0]	Data low[0]	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]	CRC8
Read_Para_1															
0x00			16 Bit		xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	
?(*3)			-"-		Wert		Zusätzliche Dez.stelle (*2)	Status (*6)	Einheit	Ascii Txt			Mode (*4)	Exp	
Read_Para_2															
0x01			16 Bit		xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	
?(*3)			-"-		Default		Zusätzliche Dez.stelle (*2)	(*11)	Max		Min		xxx		
Write_Para															
0x02			16 Bit		Wert		Zusätzliche Dez.stelle (*2)						CRCb	XOR	
?(*3)			-"-		xxx										
Read_Ram															
0x03			16 Bit		xxx										
?(*3)			-"-		Wert		Zusätzliche Dez.stelle (*2)	Status	Einheit	Ascii Txt			Mode (*4)	Exp	
ReadRamBurst (*15) (erforderlich, wenn mehr als ein Istwert vorhanden)															
0x15			Index												
?(*3)			vld (*15)	Idx (*15)	Wert 1		Wert 2		Wert 3		Wert 4		Wert 5		
Write Ram															
0x04			16 Bit		Wert		Zusätzliche Dez.stelle (*2)		xxx				CRCb	XOR	
?(*3)															
Read_Number (liest Anzahl Parameter und Ramzellen)															
0x05			0		xxx										
?(*3)					Anzahl Parameter+Sollwerte		Anzahl Ram		Anzahl Sollwerte		Anzahl Status "16" (*12)		Anzahl Status "64" (*12)		

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Token	Bus Adr Quelle	Bus Adr Ziel	Dta Adr high	Dta Adr low	Data high [0]	Data low[0]	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]	CRC8
Antwort	Bus Adr Quelle	Bus Adr Ziel	Dta Adr high	Dta Adr low	Data high [0]	Data low[0]	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]	CRC8
Set_Relais															
0x06			0	Block	Ein		Aus		Toggle						
?(*3)			0	Block	akt. Zustand										
Write EEprom															
0x07			Adresse				Wert								
?(*3)			Adresse												
Read EEprom															
0x08			Adresse				Wert								
?(*3)			Adresse												
Write Data (auto-Inkrement) ; inkrementiert Adresszähler nach Schreiben automatisch															
0x09			Data 1		Data 2		Data 3		Data 4		Data 5		Data 6		
?(*3)			Adresse f. nächsten Schreibzgr.												
Read Data (burst) ; liest 5 Words auf einmal															
0x0a			Adresse				Amode	Kanal							
?(*3)			vld (*15)	Idx (*15)	Data 0		Data 1		Data 2		Data 3		Data 4		
Read Data Info															
0x14				Index											
?(*14)															
Read Time															
0x0b			16 Bit												
?(*3)			-"-		DoW	Tag	Monat	Jahr (0-99)	Stunde (24h)	Minute	Sekunde	Zeitmodus (*5)	Mask. (*10)	Offset*2	
Set Time															
0x0c			16 Bit		DoW (0=Mo)	Tag	Monat	Jahr (0-99)	Stunde (24h)	Minute	Sekunde	Zeitmodus (*5)		Offset*2	
?(*3)			-"-		Rückgabewert wie oben										

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Token	Bus Adr Quelle	Bus Adr Ziel	Dta Adr high	Dta Adr low	Data high [0]	Data low[0]	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]	CRC8
Antwort	Bus Adr Quelle	Bus Adr Ziel	Dta Adr high	Dta Adr low	Data high [0]	Data low[0]	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]	CRC8
Read Version / Info															
0x0d				Index											
?(*3)					Info (*8)										
Read Generic 1 (allgemeiner Speicherzugriff)															
0x0e			siehe Read Para 1												
?(*3)			siehe Read Para 1												
Read Generic 2 (allgemeiner Speicherzugriff)															
0x0f			siehe Read Para 2												
?(*3)			siehe Read Para 2												
Write Generic (allgemeiner Speicherzugriff)															
0x10			siehe Write Para												
?(*3)			siehe Write Para												
Search String (in Parameter Liste)															
0x11			Text												
?(*3)			Text		Adresse		Result (*9)								
Start Test															
0x12			Prüfschritt		Wert								CRCb	XOR	
?(*3)			-"-		Wert		0	Status (*6)							
FreezeTime (Datalogger)															
0x17															
?(*3)			-"-				(Zeit) neuester gesp. Datensatz				(Zeit) ältester gesp. Datensatz				
BusVersion (16*)															
0x18															
?(3*)					Version		Class								

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Token	Bus Adr Quelle	Bus Adr Ziel	Dta Adr high	Dta Adr low	Data high [0]	Data low[0]	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]	CRC8
Antwort	Bus Adr Quelle	Bus Adr Ziel	Dta Adr high	Dta Adr low	Data high [0]	Data low[0]	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]	CRC8
ReadStatus (12*)															
0x19			Index										--- frei ---		
?(3*)					Status										
ClearStatus (12*)															
0x20			Index										CRCb / XOR		
?(3*)					Status										
SetStatus (12*)															
0x21			Index										CRCb / XOR		
?(3*)					Status										
Shut Up! (nur Broadcast) (17*)															
0x23		0							ID						
Wake Up! (nur Broadcast) (17*)															
0x24		0							ID						
Ping (nur über Broadcast an alle!)															
0x22		0			Adr Hi, Lo (*18)										
?(3*)			Adr		Adr		Adr		Adr		Adr		Adr (*18)		
ReadRamDebug (19*)															
0x1a				Speicher	Adr										
?(3*)			Status		Data 4		Data 3		Data 2		Data 1		Data 0		
Logger (30*)															
0x1d															
?(3*)															

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Token	Bus Adr Quelle	Bus Adr Ziel	Dta Adr high	Dta Adr low	Data high [0]	Data low[0]	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]	CRC8
Antwort	Bus Adr Quelle	Bus Adr Ziel	Dta Adr high	Dta Adr low	Data high [0]	Data low[0]	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]	CRC8
BootloaderCommand (21*)															
0x3d			0	0	Con	Adr	Code	CRC					CRCb	XOR	
GatewayCommand															
0x3f															

Befehle in rot müssen implementiert werden!

<u>Ergänzung:</u> Adresse 0 steht für „Broadcast“. D.h. alle Regler sind angesprochen – es erfolgt kein Reply ausser bei "Ping"!

Beschreibung

(*2) zusätzliche Dezimalstelle

Wenn Bit 7 gesetzt ist, wird hier eine zusätzliche Dezimalstelle übertragen. Dazu wird der exakte Wert durch 10 geteilt und gerundet. Die Rundung erfolgt bei positivem Wert zum nächst größeren Wert, bei negativen Wert zum nächst kleineren Wert. Dieser Wert wird im Feld „Wert“ (Data[0]) übertragen. Im Feld „zusätzliche Dezimalstelle“ (Data high [1]) wird die Differenz aus dem exakten Wert und dem übertragenen 16-Bit-Wert eingetragen. Bit 6 zeigt dabei das Vorzeichen an. Hierbei gilt die folgende Zuordnung:

0xfb	0xfc	0xfd	0xfe	0xff	0xf0	0x81	0x82	0x83	0x84	0x85
-5	**-4**	**-3**	**-2**	**-1**	**0**	**1**	**2**	**3**	**4**	**5**

Die Kommastellen, die in Feld „Mode“ (Data high [4], siehe *4) übertragen werden, beziehen sich auf den übertragenen 16-Bit-Wert. Zur Bestimmung des exakten Wertes gilt demzufolge:

Exakter Wert = (16-Bit-Wert * 10) + zusätzliche Dezimalstelle

(Source Code für die Dekodierung im Anhang 3.

(*3) Return Code

Bit 7: Fehlerbit (wenn dieses Bit gesetzt ist, gilt unten stehende Fehlertabelle).
Bit 0-5 enthält als Rückgabe den Token des Requestblocks) – im Fehlerfall wird im "Adresse high" Feld der Fehler folgendermaßen kodiert:
Dies betrifft nur Fehler auf Ebene des Kommandos selbst (Fühlerfehler werden so nicht übertragen – siehe (*6)).

Return – Code	Bedeutung
0x01	Adressüberschreitung
0x02	Wertüberschreitung
0x03	CRC Fehler
0x04	Token existiert nicht.
0x05	Verbotener Schreibbefehl
0x06	Falsche Prüfsumme im Schreibbefehl
0x07	Wait
0x08	Timeout für EepromReadBurst
0x09	Befehl gesperrt (z.B Uhrzeit verstellen)
0x0a	Kein Datensatz vorhanden (Indexüberschreitung)

Bit 6 (Ab ST Bus V3.0): zeigt an, dass es sich um eine Antwort (ACK) von einer Anfrage handelt.

(*4) Belegung des Konfigurationsbytes:

Bit	7	6	5	4	3	2	1	0
Funktion	Un-signed	Spezial Modus (s.u.)		Read only	Komma			

Spezial Modus:

Bit 6	Bit 5	
0	1	Bitfeld
1	0	Time
1	1	Datum (Tage seit Jahresbeginn wird in Format dd.mm angezeigt)

(*5) Belegung des Zeitmodus:

Bit	7	6	5	4	3	2	1	0
Funktion	MEZ	MESZ	RTC/ DCF	Valid	-	-	-	-

(*6) Status:

Bit	7	6	5	4	3	2	1	0
Funktion	1: On 0: Off	Not activ	Over-flow	Under-flow				Valid

graue Felder: optional

"Valid":
0: Nutzdatenwort Fehlercode (siehe ReadRamBurst)
1: Nutzdatenwort enthält gültigen Wert

„On/Off“: z.B. Datenlogger - Messwerte bei ausgeschaltetem Regler

(*7) Idee hinter der Passwortabfrage:

Das Display handelt die Passwortabfrage komplett, d.h. es ist dem Display überlassen, wie Passworte abgefragt werden. Die Steuerung kümmert sich nicht um Passwortabfragen und hinterfragt auch Passwortänderungen nicht.
Es wird vorausgesetzt, daß ein Busteilnehmer weiß, was er tut. Abgesichert wird erst an der Schnittstelle Mensch/Bus.

(*8) Info und Versionsabfrage.

Index 0 ist genormt und wird folgendermaßen belegt:

"Pgm Nummer" [2 Byte] "Pgm Version" [2 Byte] "Regler Info" [6 Byte]
Die "Pgm Nummer" setzt sich aus 2 hexadezimal kodierten Bytes zusammen. Das erste Byte steht für die Programmnummer, das zweite für die Variante. Die nachfolgendende 16 Bit Zahl kodiert die Version. Beispiel: "AV40_01" Version 0.20 wird zu "0x40 0x01 0x00 0x14".
Die Version wird im Display mit 2 Stellen angezeigt ("a.bb"). Sofern sich "a" der Steuerung vom (im Display) hinterlegten "a" unterscheidet, wird ein "Versionsfehler" angezeigt.

Bei Atmelprogrammen werden die 6 Byte mit der Info gefüllt, die die Programmiereinrichtung generiert (siehe "prüfsumme.doc").
Für Lonprogramme wird in der ersten Zeile die Kodierung "0x00 0x00 0x 0x**" + ID verwendet - in der zweiten Zeile folgt der Programmname im Klartext ("**" kann Programmversion enthalten).**
Die nachfolgenden Indizes können mit Textinfo belegt werden. Es gibt keine Info über die Größe dieses Bereichs (wird sowieso sequentiell gelesen), das Ende wird über die Fehlermeldung (Adressüberschreitung) erkannt.

Regler ID bei Atmel Programmen:

```
   Regler ID          Datum             Nummer frei       CRC
Länge:        20             16          10    2    8      8
 |---- ID ----------|YYYYYYYMMMMDDDDD| lfdNr  |x|xxxxxxx| CRC8  |
 00000000111111112222222233333333444444455555555666666667777777 Byte Nummer
```

Bedeutung: "Y" = Day; "M" = Month; "Y" = Year; "ID" =Identnr.; "lfdNr" = laufende Nummer; "CRC8" = Programmspeicher CRC

(*9) Resultcode:

0: "OK"
1: "not found"
Es wird die Adresse ausgegeben, bei der der ASCII String des Parameters zum ersten Mal auftaucht.

(*10) Maskierung der Uhrzeit:

Bit	7	6	5	4	3	2	1	0
Funktion	Jahr	Monat	Tag	DoW	MESZ /MEZ	Verst. verb.		

Nicht alle Felder müssen von jeder Uhr geliefert werden. Jedes gesetzte Bit steht für einen von der Uhr gelieferten Wert.

Einheit:

Zuordnung der Einheitenkodes (es werden keine Exponenten mitkodiert – dafür gibt es das "EXP" Feld):

Code	Einheit
0	keine Einheit
1	Zähler/Zahl
2	binär
3	Temp. abs.
4	Temp. rel.
5	bar
6	Pascal
7	Siemens
8	Meter
9	Volt
10	Ampere
11	Stunden
12	Minuten
13	Sekunden
14	Uhrzeit
15	m/s
16	Newton
17	Gramm
18	rel. Feuchte
19	Hertz
20	Ohm
21	Prozent
22	l/min
23	l/h

"Temp. abs." bedeutet eine absolute Temperatur (z.B. Sollwert).
"Temp. rel." steht für eine relativ geltende Temperatur (z.B. Hysterese).
Beim Umrechnen in Fahrenheit wird in diesem Fall kein Offset von 32° berücksichtigt.
Alle Temperaturen werden grundsätzlich in °Celsius / Kelvin übertragen!

(*11) Default Wert aktiv:

Steht an der Stelle von Bit 0 eine "1", so wird kein Defaultwert übergeben.
Überspringen von Parametern: Bit 1: "1" => Parameter steht auf "hidden".

(*12) Anzahl Status:

Wieviele Statusworte sind im Rambereich enthalten?
(=> "Anzahl Ram">="Anzahl Status"!)

Ab Version 3.0 werden die Statusworte über einen extra Befehl abgefragt. Der Befehl "ReadNumber" liefert weiterhin die Anzahl der alten Statusworte (16 Bit - Ende Ram Bereich) und jetzt zusätzlich die Anzahl der neuen (64 Bit) Worte zurück. Falls beide implementiert werden, entsprechen die alten Worte den ersten 16 Bit der neuen (d.h. die ersten 16 Bit des 64 Bit Wortes wird an die passende Stelle im Rambereich gemappt)

Die Reihenfolge ist aufgrund der 16-Bit Übertragung etwas verwürfelt. Es besteht folgender Zusammenhang, wobei hier zur Verdeutlichung von einem 8x8Bit Array ausgegangen wird:

Dta Adr low	Data high [0]	Data low[0]	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]
	Byte [1]	Byte [0] LSB	Byte [3]	Byte [2]	Byte [5]	Byte [4]	Byte [7] MSB	Byte [6]		

Statuswort 0 (64 Bit ab Version 3.0)

Byte	Bit	Funktion
Data_low[0]	0	On/Off Gesamt (read/write)
	1	On/Off System 1 (read/write)
	2	On/Off System 2 (read/write)
	3	Abtau (read/write)
	4	Alarmquittierung (write)
	5	Reset des Reglers
	6	
	7	
Data_high[0]	8	Batteriebetrieb (Geräte mit Akku bei Netzausfall)
	9	Ep0 (read) "allgemeiner intern. Hardwarefehler"
	10	Ep1 (read) "Eeprom Fehler"
	11	Ep2 (read) "Externer Speicherfehler"
	12	Pr (read) "Prüfen nicht absolviert"
	13	rtc (read) "Uhr stellen"
	14	Hupe (read)
	15	allg. Fehler / Sammelalarm (read)
Data_low[1]	16	Parametersatz 1 aktivieren
	17	Parametersatz 2 aktivieren
	18	Fahrenheit (1=Fahrenheit

Wenn im Statuswort 1 ein Bit markiert wird, muß auch das Bit "allg. Fehler" gesetzt sein.

Statuswort 1 (64 Bit ab Version 3.0):
Bit 0 ⇔ "E0" ... Bit 63 ⇔ "E63"

(*14) Zusatzfunktionen für Datenloggerabfrage per Bus

1. Schritt: Abfrage der Anzahl und der Art der vorhandenen Kanäle: "ReadDataInfo" mit Index=0.

Antwortformat:

Dta Adr low	Data high [0]	Data low[0]	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]
	Analog Ein (Words)	Analog Aus (Words)	Dig. Ein (Bits)	Dig Aus (Bits)	Dig Intern (Bits)				Länge Zeitintervall (in s)	

Dieses Wort enthält die Info über:
- **Zahl der analogen Ein- und Ausgänge (in 16-Bit Worten)**
- **Zahl der digitalen Stati (jeweils in Bit, wobei das nächste Word jeweils gefüllt wird).**
- **Länge eines Sampleintervall in Sekunden.**

Mit den folgenden Indizes kann die Information der gespeicherten analogen Kanäle abgerufen werden.
Definition des Antwortrahmens wie bei "ReadRam" (das "Wert" Feld bleibt unbenutzt).

2. Schritt: Einfrieren der Systemzeit ("Remote")

"Jetzt-Zeit" des Loggers merken ("Freeze Time"):
Ab diesem Moment wird das "jetzt" eingefroren. Wenn während der Abfrage weitere Samplewerte hinzukommen, bleibt das für die Abfrage ohne Bedeutung.
Als Antwort wird die eingefrorene Zeit zurückgeschickt.

3. Schritt: Abfrage der Info eines bestimmten Kanals.

Belegung des Abfragebefehls ("ReadData Burst" - Adressfeld):

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Zeitindex				AbMode	Kanal	

AbMode=0: Abfrage des kontinuierlichen Zeitspeichers
In diesem Modus wird über den Zeitindex die erste zu lesende Samplenummer (die jüngste) des jeweiligen Kanals gewählt. Die Samplenummer bezeichnet die Nummer der Messung vom "jetzt" aus gesehen. Von dieser Nummer aus werden 5 Samples auf einmal zurückgegeben (vom Index aus in die Vergangenheit).
(siehe *3)
Kanal: welcher Kanal wird abgefragt
Bei den digitalen Signalen wird immer ein komplettes Wort gesendet, auch wenn weniger Bits definiert sind.

Antwortformat:

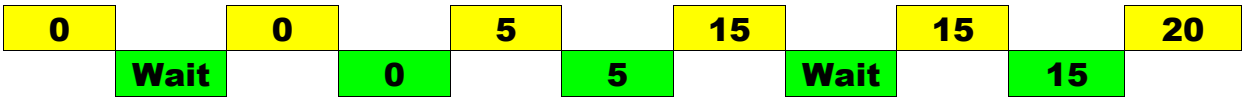
Dta Adr high	Dta Adr low	Data high [0]	Data low[0]	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]
Valid	Idx	Data 0		Data 1		Data 2		Data 3		Data 4	

Idx: Low Byte des Zeitindexes der Anfrage
Valid: gültige Datenworte markieren:

Bit	7	6	5	4	3	2	1	0
Funktion	---	---	---	Data 4	Data 3	Data 2	Data 1	Data0

Ist ein Datenwort ungültig, so enthält das gesendete Datenwort eine Fehlernummer: (siehe unter (*15) ReadRamBurst)

Antwortschema



Sollte im Zyklus ein "Wait" auftreten, so muss der letzte Index noch einmal abgefragt werden.

AbMode=1: Abfrage des Ereignisspeichers
Auch hier wird mit Index 0 begonnen, der die "Jetztzeit" markiert. Steigende Indizes gehen tiefer in die Vergangenheit. Antwortformat:

Data 0	Data 1	Data 2	Data 3	Data 4	Data 5
Timestamp		Bitmuster			

Aufbau Timestamp: Sekunden nach dem 1.1.2001.

(*15) ReadRamBurst

**Idee: eine Gruppe von Istwerten wird auf einmal eingelesen (funktioniert nur, wenn sich die Einheiten und Kommas nicht verändern).
Je nach Index wird die Antwort mit den vorhandenen Analogeingängen gefüllt (i.d. Reihenfolge der Speicherstellen).
Index 0 entspricht der Gruppe mit den Adressen 0-4; Index 1 den Adressen 5-9 usw.**

Antwortformat:

Dta Adr high	Dta Adr low	Data high [0]	Data low[0]	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]
Valid	Idx	Wert 1		Wert 2		Wert 3		Wert 4		Wert 5	

**Idx: Low Byte des angefragten Indexes
Valid: gültige Datenworte markieren:**

Bit	7	6	5	4	3	2	1	0
Funktion	---	---	---	Wert 5	Wert 4	Wert 3	Wert 2	Wert 1

Ist ein Wert ungültig, so enthält das gesendete Datenwort eine Fehlernummer. Die Fehlernummern sind allgemein und gelten auch für die Befehle ReadPara1 (Data low[1] Bit 1), ReadRAM (Data low[1] Bit 1) bzw. ReadData (burst).

Low-Byte (Nummer)	
0	Unbekannter Fehler
1	Kanal deaktiviert
2	Daten ungültig wegen Netzausfall
3	Sensorfehler (allgemein)
4	Keine Hardware vorhanden
5	Adressüberschreitung
...	
High-Byte (bitorientiert -> Kombinationen möglich)	
1	Underflow (Kurzschluss)
2	Overflow (Kabelbruch)
4	Deaktiviert

Der Wert –10000 wurde in den ersten Geräten verwendet und diente nur zur Anzeige „---„ in einem angeschlossenen Display.

(*16) Busversion

"Version": Ausgabe der Busversion (Wert "30" == "3.0").
"Class": Welche Implementierungsstufe hat das Gerät?

Bits 0 und 1 nur in Mastermodus belegt.

Bit	7	6	5	4	3	2	1	0
Funktion	Master /_Slave						Slave-fähig	Multi-master-fähig

Es gibt folglich diese Arten von BusMastern:

"10000000": Master Class I:
Nur ein BusMaster im Netz erlaubt.
Der Master kann nicht von außen angesprochen werden.

"10000001": Master Class II:
Mehrere BusMaster im Netz erlaubt.
Die Master können nicht von außen angesprochen werden.

"10000010": Master Class III:
Ein BusMaster im Netz erlaubt.
Der Master kann von anderen angesprochen werden (nur zur Parametrierung / kein wirklicher Mutlimasterbetrieb).

"10000011": Master Class IV:
Mehrere BusMaster im Netz erlaubt.
Der Master kann von anderen angesprochen werden

"Shut Up" schaltet den Regler einer ID (Broadcast) oder einer Adresse (explizite Adressierung) selektiv vom Bus. Nach einem Timeout von 60-120s regeneriert dieser sich selbständig.
"Wake Up" schaltet den Regler mit dieser ID wieder zu.

Sinn: Es kann automatisch das vorhandene Netz gescannt werden. Wurde auf einer Netzwerkadresse ein Teilnehmer gefunden, wird dieser vom Netz getrennt. Auf dieser Adresse darf kein weiterer Teilnehmer antworten, ansonsten liegt ein Adresskonflikt vor.

(*18) Ping

Der "Ping" Befehl veranlaßt alle Regler dieses Netzwerkbereiches zu antworten.
Der Netzwerkbereich definiert sich durch das Maskierungswort. Ein gesetztes Bit markiert einen aktiven Bereich.
Jeder angeschlossene Regler überprüft, ob er sich im angesprochenen Bereich befindet.
"Adr Lo" steht für die unterste (noch gültige) Adresse und "Adr Hi" für die oberste. D.h. der Wertebereich [Adr_Lo; Adr_Hi] ist gültig.
Die Adresse wird im Datenwort 6 mal im 16 Bit Format gesendet. Das Highbyte besteht aus der Adresse selbst, das Lowbyte aus der invertierten Adresse.
Somit besteht die Möglichkeit, sogar aus überlappenden Antworten mehrerer Regler deren Adressen zu generieren.

(*19) ReadRamDebug

Erlaubt das Lesen beliebiger Speicherarten (jeweils 5 Words - sofern vorhanden).
Word 6 enthält als Statusinfo für jede tatsächlich zurückgegebene Ramzelle ein Bit.
Die Zuordnung erfolgt byteweise => 5 Words = 10 Bytes = 10 Statusbits.
Vordefinierte Speicherarten:

0: Ram Intern
1: Flash Intern
2: Ram extern
3: Flash extern
4: NV Ram

Bootloader

Es soll möglich sein innerhalb eines Netzwerkes einzelne Regler mit neuer Software zu laden. Es können sowohl Flash, EEprom und externe Speichermedien (SPI-Flash) über den Bootloader programmiert werden. Über einen speziellen ST-Bus Befehl kann der Regler aus der laufenden Applikation in den Boot-Mode gebracht werden. Dazu muss an einer festgelegten Speicherstelle im EEProm ein Code und die derzeitige Netzwerkadresse (4:Con, 3:Adr, 2:Code, 1:CRC) hinterlegt und danach ein Reset ausgelöst werden.

Funktionsprinzip

Nachdem *RESET* überprüft der Bootloader ob im EEPROM an Speicherstelle EE_BOOT_CODE = E2END + (1 – 2) ungleich 0xFF steht. Entsprechend dieser Codierung reagiert der Bootloader. An der Stelle EE_BOOT_ADR = E2END + (1 – 3) steht die aktuelle Netzwerkadresse und an EE_BOOT_CON = E2END + (1 – 4) die Verknüpfung des Reglers. Evtl. kann bei EE_BOOT_CRC = E2END + (1 – 1) eine Checksumme über diese 3 Bytes gebildet werden.

Im Flash (Applikation) muss die Programmversion geändert werden. Die Seriennummer, Fertigungsdatum usw. wird vom Bootloader NICHT überschrieben.

0xFF: Sprung zur Applikation

0x1y:Bootloader UART1
- 0:ST-Bus
- 1:RS232
- 2:Kommunikationsmodul (ARM7)

0x2y:Bootloader UART2
- 0:ST-Bus
- 1:RS232
- 2: Kommunikationsmodul (ARM7)

0x3y:Bootloader SPI
- 0:
- 1:
- 2: Kommunikationsmodul (ARM7)

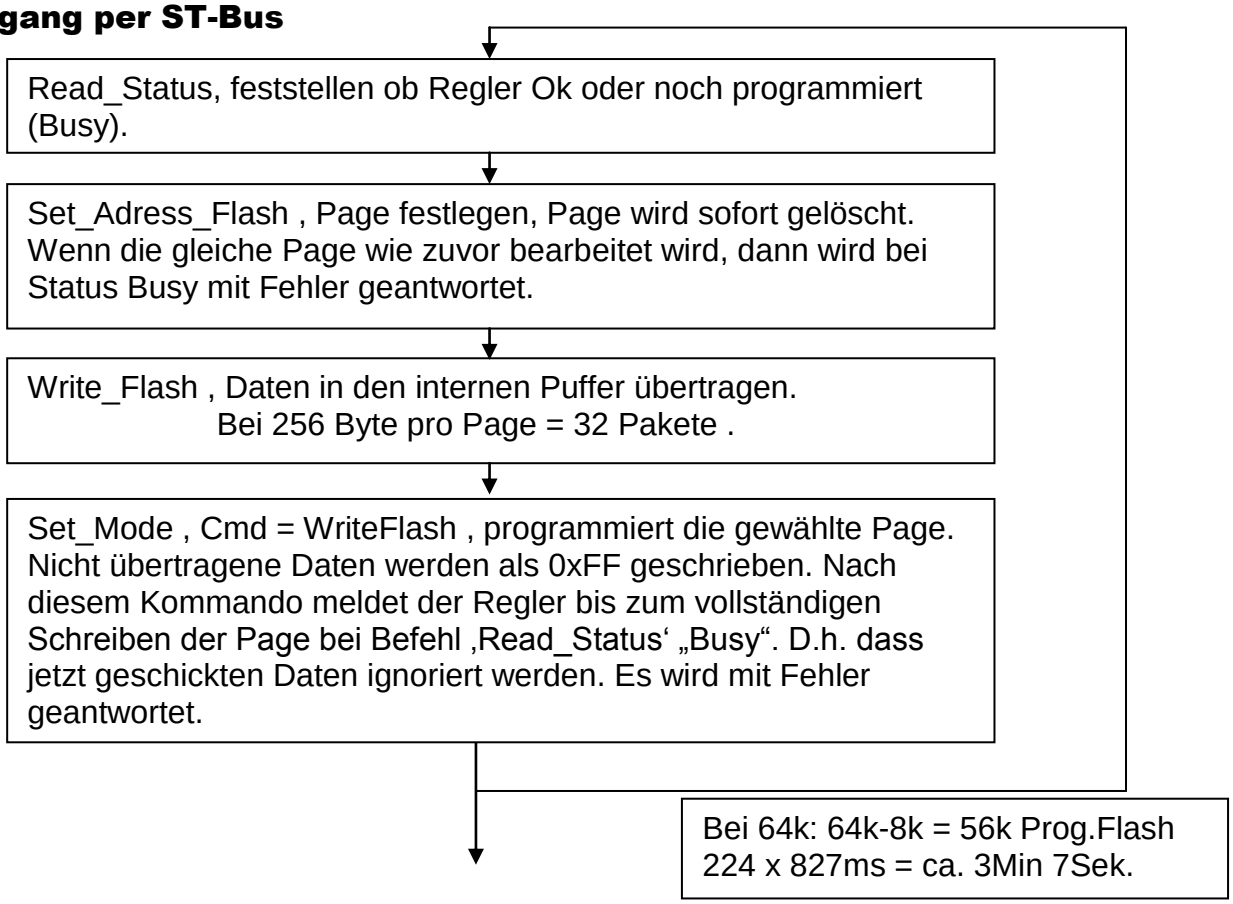
0x4y:Bootloader CAN
- 0:
- 1:
- 2:

Nach erfolgreicher Programmierung wird EE_BOOT wieder auf 0xFF gesetzt und ein Reset ausgelöst. Die Applikation initialisiert das EEprom bis auf die letzten vier Speicherstellen. Steht bei EE_BOOT_ADR bzw. EE_BOOT_CON ein Wert ungleich 0xFF werden diese als neuer Netzwerkparameter L0, L1 übernommen.

Ablauf eines Flashvorgangs mit Bootloader V1 (Stand 3.7.2009)

Der Prozessor befindet sich im Bootmodus (Adresse EE_BOOT_CODE wurde nach dem Reset auf ungleich 0xFF positiv geprüft).

Vorgang per ST-Bus



(*21) Bootloader

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Token	Bus Adr Quelle	Bus Adr Ziel	Dta Adr high	Dta Adr low	**Befehl**	**Anzahl Data**	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]	CRC8
Antwort	Bus Adr Quelle	Bus Adr Ziel	Dta Adr high	Dta Adr low	**Befehl**	**Anzahl Data**	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]	CRC8
Read_Status (Ok, Busy, Error, ErrorFuse, LastPageWrite_Error, LastPageWrite_OK)															
0x3d			0		0										
?(*3)			0		0		StatusH	StatusL	CRC LastPage			Version	Device ID	BootSize	
Set_Mode (Boot, Run, Reset, Program, WriteRam, WriteFlash, CalcCRC)															
0x3d			0		3		Cmd								
?(*3)			0		3			CRC PageBuffer							
Read_Signature															
0x3d			0		4										
?(*3)			0		4		Device ID	FlashSize	EepromSize	BootSize	PageSize	BootMode		Version	
Set_Adress_Flash (AbsAdresse = PageAdr * PageSize => PageAnzahl = FlashSize/PageSize)															
0x3d			PageAdr		5		Anz Block H	Anz Block L	Crc high	Crc low					
?(*3)					5										
Read_Flash (Adresse innerhalb Page: Adresse = AbsAdresse + BlockNr * 8)															
0x3d			BlockNr		6	0									
?(*3)					6	1-8	DataH[0]	DataL[0]	DataH[1]	DataL[1]	DataH[2]	DataL[2]	DataH[3]	DataL[3]	
Write_Flash (Adresse innerhalb Page: Adresse = AbsAdresse + BlockNr * 8))															
0x3d			BlockNr		7	1-8	DataH[0]	DataL[0]	DataH[1]	DataL[1]	DataH[2]	DataL[2]	DataH[3]	DataL[3]	
?(*3)					7		StatusH	StatusL							
Set_Adress_Eeprom (Startadresse)															
0x3d			Adr		9		Anzahl high	Anzahl low	Crc high	Crc low					
?(*3)					9										
Read_Eeprom (Adresse relativ zur Startadresse)															
0x3d			AdrNr		10										

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Token	Bus Adr Quelle	Bus Adr Ziel	Dta Adr high	Dta Adr low	**Befehl**	**Anzahl Data**	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]	CRC8
Antwort	Bus Adr Quelle	Bus Adr Ziel	Dta Adr high	Dta Adr low	**Befehl**	**Anzahl Data**	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]	CRC8
?(*3)					10	1-8	DataH[0]	DataL[0]	DataH[1]	DataL[1]	DataH[2]	DataL[2]	DataH[3]	DataL[3]	
Write_Eeprom (Adresse relativ zur Startadresse)															
0x3d			AdrNr		11	1-8	DataH[0]	DataL[0]	DataH[1]	DataL[1]	DataH[2]	DataL[2]	DataH[3]	DataL[3]	
?(*3)					11										
Read_Signature_Extern															
0x3d					12										
?(*3)					12		Device ID	FlashSize	PageSize				*EraseAnz*		
Set_Adress_Extern															
0x3d			PageAdr		13		Anz Block H	Anz Block L	Crc high	Crc low					
?(*3)					13										
Read_Extern															
0x3d			BlockNr		14										
?(*3)					14	1-8	DataH[0]	DataL[0]	DataH[1]	DataL[1]	DataH[2]	DataL[2]	DataH[3]	DataL[3]	
Write_Extern															
0x3d			BlockNr		15	1-8	DataH[0]	DataL[0]	DataH[1]	DataL[1]	DataH[2]	DataL[2]	DataH[3]	DataL[3]	
?(*3)					15										
Read_Lock															
0x3d					16										
?(*3)					16		DataH[0]	DataL[0]	DataH[1]	DataL[1]					
Write_Lock															
0x3d					17		DataH[0]	DataL[0]	DataH[1]	DataL[1]					
?(*3)					17										
Read_Fuse															
0x3d					18										
?(*3)					18		DataH[0]	DataL[0]	DataH[1]	DataL[1]					
Write_Fuse															
0x3d					19		DataH[0]	DataL[0]	DataH[1]	DataL[1]					

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Token	Bus Adr Quelle	Bus Adr Ziel	Dta Adr high	Dta Adr low	**Befehl**	**Anzahl Data**	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]	CRC8
Antwort	Bus Adr Quelle	Bus Adr Ziel	Dta Adr high	Dta Adr low	**Befehl**	**Anzahl Data**	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]	CRC8
?(*3)					19										

Write_Flash (Adresse innerhalb Page: Adresse = AbsAdresse + BlockNr * 8))

Bei Anzahl Data == 0 wird der Empfangspuffer geleert. Das ist zwingend wenn keine volle Page zum Prozessor geschrieben werden soll.

Hinweise zur Erstellung der HEX-Dateien für die Fertigung

Für Fertigungsprozesse gibt es nur eine HEX-Datei. Diese muss die Aplikationssoftware und Bootloadersoftware enthalten.
Dazu kopiert man aus der Applikations-HEX-Datei den Adressbereich 0xE000-FFFF und fügt ihn in der Bootloader-HEX-Datei ein.

- Compilierte ‚Release'-HEX-Dateien müssen für das Flashen per Bootloader bearbeitet werden:
  - Die Adress-Bereiche 0x7000 bis

Datenlogger V2.0

Ziel ist es eine Funktion zu schaffen, welche jeden Datenlogger über ST-Bus auslesen kann.
In einer späteren Version soll auch ein Datenlogger über den ST-bus konfiguriert werden können.

11.11.2009 V0.01 LoggerControl.GetInfo.AdrLow = 1

Allgemein
Die Datensätze werden im Speicher als Sekunden seit 1.1.2001 00:00:00 (GMT=0 „*Greenwich* Mean Time“) gespeichert. Der ST-Bus überträgt ebenfalls GMT=0. Die Umrechnung in die aktuelle Zeitzone erfolgt erst in der Anzeige des Datenloggers oder mit einem PC-Programm.
Somit hat das PC-Programm die Möglichkeit entweder die Uhrzeit am Standort des Datenloggers oder die Uhrzeit am Standort des PCs zu ermitteln.

Unter GetInfo() wird die aktuelle Zeitzone des Standortes des Datenloggers eingetragen. Dazu muss ein Parameter reserviert werden, welcher den Offset von GMT=0 mit einer Auflösung von 1 Stunde beinhaltet. Ferner muss ein Parameter reserviert werden, welche die automatische Sommerzeitumschaltung aktiviert bzw. sperrt. Die Korrektur von GMT bedingt durch die Sommerzeit erfolgt automatisch.

Parameter Zeitzone:	Offset = ParameterWert * 1h	Wird in GetInfo.DataLow[4] übertragen
Parameter MEZ/AUTO	0:MEZ 1:Auto (MEZ/MESZ)	Wird in GetInfo.DataHigh[4] übertragen

(*30) Datenlogger (V2.0)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Token	Bus Adr Quelle	Bus Adr Ziel	Dta Adr High	Dta Adr low	Data high [0]	Data low[0]	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]	CRC8
Antwort	Bus Adr Quelle	Bus Adr Ziel	Dta Adr High	Dta Adr low	Data high [0]	Data low[0]	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]	CRC8
Logger – GetInfo															
0x1d			0	0											
?(*3)				Version	AnzAnalog		AnzEvent						MEZ/AUTO	Zeitzone	
Logger – GetInfoAnalog (AnalogNr -> Einheit, Name, Datentyp) (*38)															
0x1d			0	1	0…(AnzAnalog-1)										
?(*3)				ExtAdr (*38)	AnzHeaderMax		AnzIndexMax		Einheit	Ascii Txt			Datentyp (*31)	Komma/Bit	
Logger – GetInfoAnalog (AnalogNr -> Zusatzinformationen für XML) (*38)															
0x1d			1..X	1	0…(AnzAnalog-1)										
?(*3)															
Logger – GetInfoAnalogHeader (AnalogNr, HeaderNr -> Startzeit, Intervall, Datensätze)															
0x1d			0	2	0…(AnzAnalog-1)		0… (AnzHeader-1)								
?(*3)					Datum, Uhrzeit (IndexNr = 0)				Intervall		AnzIndex				
Logger – GetInfoEvent (EventNr -> Einheit, Name, Datentyp) (*38)															
0x1d			0	3	0…(AnzEvent-1)										
?(*3)				ExtAdr (*38)					Einheit	Ascii Txt			Datentyp (*31)	Komma/Bit	
Logger – GetInfoEvent (EventNr -> Zusatzinformationen für XML) (*38)															
0x1d			1..X	3	0…(AnzEvent-1)										
?(*3)															
Logger – GetIndexAnalog (*35) (Start, Stop -> Header, Index) (mind. 1 Wait Zyklus)															
0x1d			0	4	StartZeit				StopZeit				0..(AnzAnalog-1)		
?(*3)			AnzDaten				StartHeaderNr		StartIndexNr		StopHeaderNr		StopIndexNr		
Logger – GetIndexEvent (Start, Stop, EventNr -> Index)															
0x1d			0	5	StartZeit				StopZeit				-1:Alle, 0..(AnzEvent-1)		
?(*3)			AnzDaten, sofern EventNr ≠ -1				StartIndexNr				StopIndexNr				

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Token	Bus Adr Quelle	Bus Adr Ziel	Dta Adr High	Dta Adr low	Data high [0]	Data low[0]	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]	CRC8
Antwort	Bus Adr Quelle	Bus Adr Ziel	Dta Adr High	Dta Adr low	Data high [0]	Data low[0]	Data high [1]	Data low[1]	Data high [2]	Data low[2]	Data high [3]	Data low[3]	Data high [4]	Data low[4]	CRC8
Logger – WriteCfgAnalog (Befehle um Datenlogger über ST-Bus programmieren zu können)															
0x1d			0…X	6											
?(*3)															
Logger – WriteCfgEvent (Befehle um Datenlogger über ST-Bus programmieren zu können)															
0x1d			0…X	7											
?(*3)															
Logger – ReadAnalog (*37)															
0x1d			0	0x80	0..(AnalogAnz-1)		HeaderNr		IndexNr					(Anzahl)	
?(*3)			Status (*34)		Daten (*32)										
Logger – ReadEvent (*36) ???															
0x1d			0	0x81	IndexNr				Offset				-1:Alle, 0..(AnzEvent-1)		
?(*3)			Datum, Uhrzeit				EventNr		Daten (*33)						
Logger_Data –															
0x1e			0	2											
?(*3)															
Logger_Data –															
0x1e			0	3											
?(*3)															

(*31)Datentypen

Code	Binär	Datentyp	Komma/Bit	Anmerkung
0x10	0001 0000	ui8	Komma	
0x11	0001 0001	i8	Komma	
0x12	0001 0010	Bitfeld8	Bit (0xFF:Alle)	
0x20	0010 0000	ui16	Komma	Standard
0x21	0010 0001	i16	Komma	Standard
0x22	0010 0010	Bitfeld16	Bit (0xFF:Alle)	_S0, _S1,…, _I0, …, _E0,…_E4
0x28	0100 1000	Uhrzeit (ui16)	-	mmm:ss oder hhh:mm =(x / 60): (x % 60)
0x40	0100 0000	ui32	Komma	z.B. Hundertstel übertragen
0x41	0100 0001	i32	Komma	z.B. Hundertstel übertragen
0x42	0100 0010	Bitfeld32	Bit (0xFF:Alle)	
0x44	0100 0100	Float	-	
0x48	0100 1000	Datum, Uhrzeit (ui32)	-	Sekunden seit 1.1.2001 00:00:00 (GMT=0)
0x80	1000 0000	ui64	Komma	Datentyp nicht als Event möglich
0x81	1000 0001	i64	Komma	Datentyp nicht als Event möglich
0x82	1000 0010	Bitfeld64	Bit (0xFF:Alle)	Datentyp nicht als Event möglich
0x84	1000 0100	Double	-	Datentyp nicht als Event möglich
0x6C	0110 1100	ParameterNr, WertAlt, WertNeu	-	Parameter geändert
				Zugang über Passwort X (Tastatur, Modem, ST-Bus)

Wie setzt sich der Code für den Datentyp zusammen

Code					Anmerkung
[7..4]	[3..0]				
15…1	X				Datenlänge in Byte
X	0	?	?	0	Unsigned
X	0	?	?	1	Signed
X	0	?	1	?	Bitfeld
X	0	1	?	?	Float, Double
X	1	X	X	X	Sonderdatentypen

Frage: Ist der Datentyp Uhrzeit notwendig ? Könnte auch über die Einheit „UHR“ und Datentyp 0x20 = ui16 übertragen und richtig dekodiert werden werden.

(*32) Analogkanal - Datentyp

	Status[ui16]								Daten									
ST	AdrH				AdrL				DataH[0]	DataL[0]	DataH[1]	DataL[1]	DataH[2]	DataL[2]	DataH[3]	DataL[3]	DataH[4]	DataL[4]
LE	[L]				[H]				[L]	[H]	[L]	[H]	[L]	[H]	[L]	[H]	[L]	[H]
0x1X	4	3	2	1	8	7	6	5	Data1	Data2	Data3	Data4	Data5	Data6	Data7	Data8	0	0
0x2X	4	3	2	1				5	Data1 [L,H]		Data2 [L,H]		Data3 [L,H]		Data4 [L,H]		Data5 [L,H]	
0x4X			2	1					Data1 [LL,LH,HL,HH]				Data2 [LL,LH,HL,HH]				0	0
0x8X				1					Data1 [LLL,LLH,LHL,LHH,HLL,HLH,HHL,HHH]								0	0

(*33) Eventkanal - Datentyp

	ui32				ui16								Anmerkung
Typ	AdrH	AdrL	DataH[0]	DataL[0]	DataH[1]	DataL[1]	DataH[2]	DataL[2]	DataH[3]	DataL[3]	DataH[4]	DataL[4]	
LE	[LL]	[LH]	[HL]	[HH]	[L]	[H]	[LL]	[LH]	[HL]	[HH]			
0x1X	Datum, Uhrzeit				EventNr		ui8, i8, Bit						Bit, Data, Bitfeld8
0x2X	Datum, Uhrzeit				EventNr		ui16, i16 [LL,LH]						Data, Bitfeld16
0x4X	Datum, Uhrzeit				EventNr		ui32, i32 [LL,LH,HL,HH]						Data, Bitfeld32
0x48	Datum, Uhrzeit (NEU)				EventNr		Datum, Uhrzeit (ALT) (Sekunden seit 1.1.2001)						Event Uhrzeitänderung
0x6C	Datum, Uhrzeit				EventNr		ParameterNr		WertAlt		WertNeu		Event Parameteränderung
	Datum, Uhrzeit				EventNr								Event Passwort
	Datum, Uhrzeit				EventNr								
	Datum, Uhrzeit				EventNr								
	Datum, Uhrzeit				EventNr								

(*34) Status

1	0		Anmerkung
X	0	Error	Daten = Fehlercode
X	1	Valid	Daten = Wert
0	X	Aus	Regler Aus
1	X	Ein	Regler Ein

Codierung der Statusmeldungen zum Datenwort

AdrH				AdrL				DataH[0]	DataL[0]	…	DataL[4]
[L]				[H]				[L]	[H]	…	[H]
Status 4	Status 3	Status 2	Status 1	Status 8	Status 7	Status 6	Status 5	Daten1	Daten2	…	Daten8

(*35) GetIndexAnalog

StopHeader darf nicht kleiner als der StartHeader sein. Somit muss bei internem (physikalischem) Blocküberlauf ein Offset addiert werden.

HeaderNr (Index=0)	Datum, Uhrzeit	Anmerkung
0	16:00	Neuster Eintrag
1	10:00	Ältester Eintrag
2	12:00	
3	14:00	

z.B. 4 Blöcke mit AnzHeaderMax = 4
Datenbereich abfragen
GetIndexAnalog (9:30, 12:00) => AnzDaten=2, StartHeader = 1, StopHeader = 2
ReadAnalog(1…2) => 10:00, 12:00

Datenbereich abfragen
GetIndexAnalog (9:30, 16:30) => AnzDaten=4, StartHeader = 1, StopHeader = 4(0) Interner Offset.
ReadAnalog(1…4) => 10:00, 12:00, 14:00, 16:00 BlockAdrPhy = Adr(AnalogNr, HeaderNr, IndexNr) % AnzHeaderMax

Datenbereich abfragen
GetIndexAnalog (14:30, 16:30) => AnzDaten=1, StartHeader = 0, StopHeader = 0
ReadAnalog(0) => 16:00

Somit ist ReadAnalog(0) **identisch** mit ReadAnalog(4)

Datenbereich abfragen
GetIndexAnalog (10:30, 11:30) =>AnzDaten=0, StartHeader =0, StopHeader = 0 Returncode hat Priorität
Keine Daten im Bereich => Returncode (*3) 0x0a

Ältester Eintrag und Neuster Eintrag ermitteln
GetIndexAnalog (Startzeit = 0, StopZeit = 0xFFFFFFFF) => AnzDaten=4, StartHeader = 1, StopHeader = 4(0)
ReadAnalog(1) => Ältester Eintrag
ReadAnalog(4) => Neuster Eintrag. Falls dem Master AnzHeaderMax bekannt ist, kann auch ReadAnalog(0) verwendet werden.(NICHT empfohlen)

(*36) GetIndexEvent

StopAdr darf nicht kleiner als der StartAdr sein. Somit muss bei internem (physikalischem) Adressenüberlauf ein Offset addiert werden.

IndexNr	Datum, Uhrzeit	EventNr (-1:Alle)	Anmerkung
0	18:00	E4	Neuster Eintrag
1	13:00	E1	Ältester Eintrag
2	13:00	E2	
3	14:00	E1	
4	15:00	E1	
5	16:00	E1	
6	16:00	E2	
7	17:00	E3	

Datenbereich abfragen
GetIndexEvent (9:30, 12:00, -1) => AnzDaten=0, StartIndex = 0, StopIndex = 0
Keine Daten im Bereich => Returncode (*3) 0x0b ;Returncode hat Priorität

Datenbereich abfragen
GetIndexEvent (9:30, 16:30, -1) => AnzDaten=6, StartIndex = 1, StopIndex = 6
ReadEvent (1..6, 0, -1) => 13:00, 13:00, 14:00, 15:00,16:00, 16:00

Datenbereich abfragen (Regler durchsucht seinen Speicherbereich – WAIT wird bis zur Antwort gesendet)
GetIndexEvent (9:30, 16:30, E1) => AnzDaten=4, StartIndex = 1, StopIndex = 5 ;AnzDaten **muss** für Offset berechnet werden
ReadEvent (1, 0, E1) => 13:00 oder ReadEvent (1, 0, 0) => 13:00 ;Identisch, da erster gefundener Wert = StartIndex
ReadEvent (1, 1, E1) => 14:00 ;Adr = StartIndex + ADR_NextEvent(1)
ReadEvent (1, 2, E1) => 15:00 ;Adr = StartIndex + ADR_NextEvent(2)
ReadEvent (1, 3, E1) => 16:00 oder ReadEvent (5, 0, 0) => 16:00 ;Identisch, da letzter gefundener Wert = StopIndex

Datenbereich abfragen
GetIndexEvent (9:30, 18:30, -1) => AnzDaten=8, StartIndex = 1, StopIndex = 8(0) ;Interner Offset.
ReadEvent (1…8, 0, -1) => 13:00, 13:00, 14:00, 15:00, 16:00, 16:00, 17:00, 18:00 ;AdrPhy = Adr(Adresse, Offset, EventNr) % 8

Ältester Eintrag und Neuster Eintrag ermitteln
GetIndexEvent (Startzeit = 0, StopZeit = 0xFFFFFFFF, -1) => AnzDaten=8, StartIndex = 1, StopIndex = 8
ReadEvent(1,0,-1) => Ältester Eintrag
ReadEvent(8,0,-1) => Neuster Eintrag oder ReadEvent(0,0,-1)

(*37) ReadAnalog

In Abhängigkeit vom DatenTyp ist ReadAnalog unterschiedlich einsetzbar.

ReadAnalog (AnalogNr, HeaderNr, IndexNr, Anzahl)
=> Status[N..0], Daten(HeaderNr, IndexNr+0), Daten(HeaderNr, IndexNr+1),...,Daten(HeaderNr, IndexNr+N) //N=Anzahl-1

Beispiel
ReadDataAnalog (1, 1, 0, 1) => Status[0], Daten(1,0)
ReadDataAnalog (1, 1, 1, 1) => Status[0], Daten(1,1)
ReadDataAnalog (1, 1, 2, 1) => Status[0], Daten(1,2)
=> 3 ST-Bus Zugriffe

ReadDataAnalog (1, 1, 0, 3) => Status[2..0], Daten(1,0) , Daten(1,1), Daten(1,2)
=> 1 ST-Bus Zugriff (Je nach Datentyp)

Wenn **Anzahl** größer ist, als die durch den **DatenTyp** mögliche übertragbare Datensätze wird die maximal mögliche Anzahl an Datensätze übertragen.
Ob eine Fehlermeldung über den ReturnCode übertragen werden soll, muss noch geprüft werden.

Bekannte Probleme
Ein Block kann während eines Lesezugriffs mit ReadAnalog() bzw. ReadEvent() gelöscht worden sein. Somit könnten falsche Daten gelesen werden.
Es muss immer der **älteste** Block vor einem Löschzugriff durch das Device geschützt werden, sofern gerade dort Daten ausgelesen werden.

Lösung:
Es wird zwischen dem neusten Header und dem ältesten Header ein „Leerheader“ eingeführt, welcher über eine erneute Abfrage von GetIndexAnalog() bzw. GetIndexEvent() nicht mehr zur Verfügung steht. Dieser „Leerheader“ enthält immer den **ältesten** Datensatz, welcher als nächstes gelöscht werden wird.
Die Daten im „Leerheader“ sind für die Zeit tmin=AnzIndex*Intervall oder bis zum nächsten geänderten Aufzeichnungsintervall über ReadAnalog(), ReadEvent() verfügbar bis diese tatsächlich überschrieben werden.
Bei Abfrage großer Datenmengen ist es somit ratsam mit dem **ältesten** Datensatz zu beginnen.

(* 38) Daten Dekodierung über XML-Datei

Um im PC-Programm über eine XML-Datei die Bezeichnung von Statusvariablen bzw. Kanäle zuordnen zu können, sind einige Zusatzinformationen notwendig. Ferner wir zur Dekodierung von Daten eines Datenloggers, welcher selber Daten von externen Reglern aufzeichnet auch noch die externe Adresse (ExtAdr) des jeweiligen Reglers oder besser gleich dessen Softwareversion benötigt.

Analogkanal

ExtAdr	Adresse des externen Reglers aus dem die Daten ausgelesen wurden (0: Regler = Datenlogger)
„ASCii Text“	Bezeichnung des Analogkanals (z.B. I_0)
DatenTyp	Enthält den Typ zur Dekodierung der Daten (bool, ui16, i16, Bitfeld16, float,…)
Komma/(Bit)	Je nach Datentyp wird die Position des Kommas übertragen.

Eventkanal

ExtAdr	Adresse des externen Reglers aus dem die Daten ausgelesen wurden (0: Regler = Datenlogger)
„ASCii Text“	Bezeichnung des Ereigniskanals (z.B. _S1, 5) oder auch eines Parameters (z.B. P 0)
DatenTyp	Enthält den Typ zur Dekodierung der Daten (bool, ui16, i16, Bitfeld16, float,…)
(Komma)/Bit	Je nach Datentyp wird die Position des Kommas oder die Bitposition übertragen.

Der Host kann jetzt über ReadVersion (ExtAdr) die Softwareversion des Reglers direkt über den ST-Bus lesen und das entsprechende XML-File nachladen.
Dabei ist es notwendig, dass sich sowohl der Datenlogger, als auch der Regler im **gleichen** Netzwerk befinden.

Um auch Regler aus anderen Netzen verarbeiten zu können, werden die Befehle GetInfoAnalog(), GetInfoEvent() erweitert. Diese Befehle müssen nur Datenlogger unterstützen, welche selber über den ST-Bus Daten von anderen Reglern (ExtAdr) aufzeichnen müssen. Evtl. könnten auch Daten von ModBus oder Can so in das ST-Bus System integriert werden.

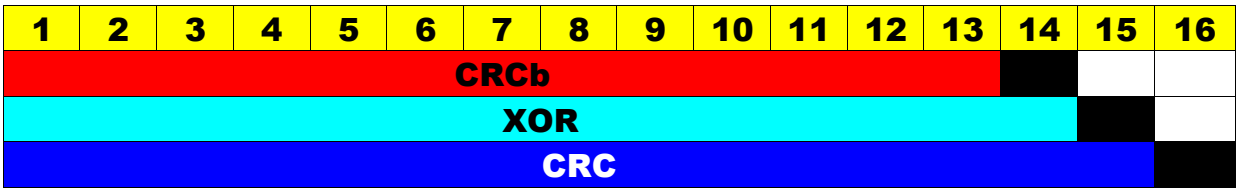
Damit ist es nun möglich über ASCii Text und Komma/Bit der Funktion einen Namen aus dem XML-File zuzuordnen.
Somit sind im PC auch mehrsprachige Texte möglich.

Fehlererkennung

Für alle Befehle gilt ein CRC 8 als Sicherungsmassnahme. Für Schreibbefehle wird zusätzliche Sicherheit durch Einführung eines reinen Paritys (initialisiert mit 0xaa) und eines zusätzlichen CRC 8 Wortes (initialisiert mit 0x55) gewährleistet.

(Source Code für die CRC Generierung in Anhang 1.

Die Darstellung zeigt, welche Prüfsumme welchen Bereich abdeckt (die letzte Zeile gilt für alle Befehle).



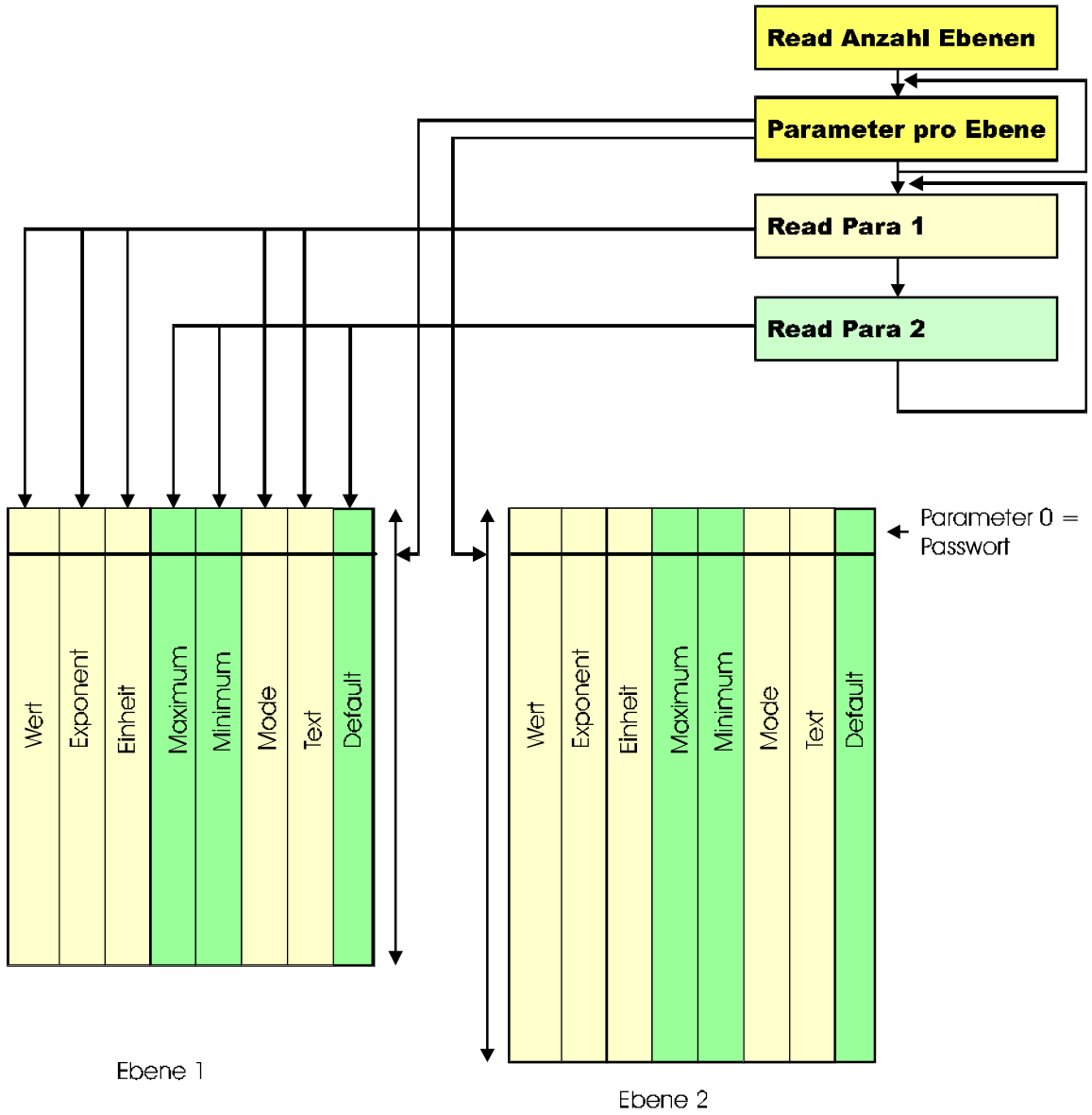
Nachfolgend findet sich eine Beispielimplementierung eines tabellenbasierenden CRC 8 Algorithmus (für >8 Bit Rechner – für alle anderen Rechner kann die Anweisung „&0xff“ entfallen).
Der CRC Speicher muß vor jeder Verwendung mit dem Wert 0xff initialisiert werden.

Das verwendete Generator-Prüfpolynom für den CRC8 lautet:

$$G_x = X^8 + X^4 + X^3 + X^2 + 1$$

(Source Code für die Prüfsummen Generierung in Anhang 2.

Beispiel: Auslesen der Parameterliste



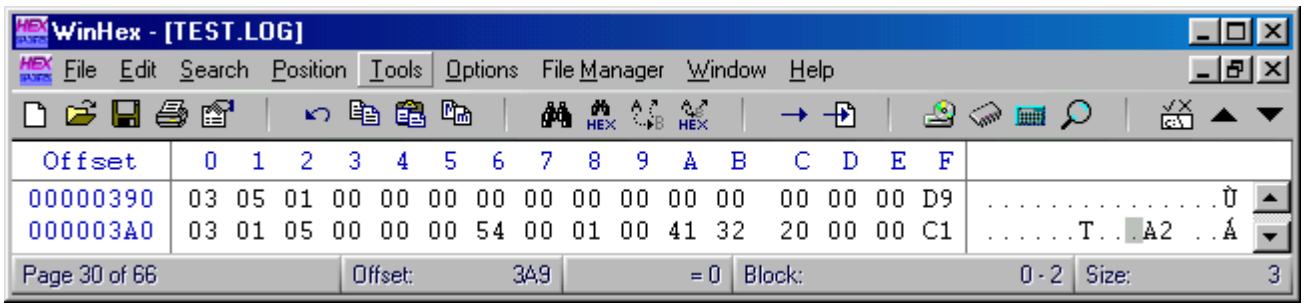
Vorgang, um die komplette Parameterliste aus dem Regler auszulesen.

1. Anzahl der im Regler enthaltenen Parameterebenen bestimmen.

2. Aus den nachfolgenden Bytes jeweils die Länge für jede Parameterebene lesen.

3. Über "Read Para 1" und "Read Para 2" die Ebenen auslesen

Kommunikationsbeispiel:



Obiges Bild zeigt einen typischen Ausschnitt einer Kommunikation zwischen Knoten 5 und 1 (5=Master).
In der ersten Zeile fragt der Master die Ramzelle 0 ab und erhält in der zweiten Zeile die Antwort "84" mit der Kennung "A2".
Dieses Beispiel kann auch verwendet werden, um die Richtigkeit der CRC Implementierung zu überprüfen.

Zusatzvereinbarungen

Netzwerkparameter (eigene Adresse) sollten nach Möglichkeit "L 0" heißen (mit der Einführung der XML basierenden Reglerdefinition kann dieser Zwang gelockert werden).
Nach einer Änderung der Netzwerkadresse muss die Quittierung noch auf dieser Adresse erfolgen; die neue Adresse muss ab dem nächsten Befehl gültig sein.

Bei einer nachträglichen Erweiterung der Parameterliste darf auf *keinen Fall* ein bisher schon verwendeter Parameter eine neue Funktion erhalten.
Beispiel: r0 "Hysterese K1" darf nicht in "Hysterese K2" verändert werden.
Eine Erweiterung des Wertebereichs ist erlaubt.
Jede Änderung muss natürlich zwingend als Versionssprung angezeigt werden.

Noch in der Diskussion:
Ein Master der im 100ms Takt Abfragezyklen startet, hat die Wichtung "1". Ein Master, der im 50 ms Takt abfragt, hat die Wichtung "2" usw:

Wichtung = 100 ms / Abfragezyklus

Der Bus muss eine Gesamt Dauer Last der Wichtung 10 verkraften.

Anhang 1

CRC Implementierung

```
// *************************
// * CRC 8 bestimmen       *
// *************************

void CRS485Dlg::DoCRC8(unsigned int * uiIn, unsigned int * uicrc)
{
      const unsigned int
      CRC_LOOK_UP[16]={0x00,0x1d,0x3a,0x27,0x74,0x69,0x4e,0x53,
                       0xe8,0xf5,0xd2,0xcf,0x9c,0x81,0xa6,0xbb};

    unsigned int uiNibble1,uiNibble2;

    uiNibble1=*uiIn&0x0f;                       // unteres Nibble Datum speichern
    uiNibble2=(*uicrc>>4);                      // oberes Nibble CRC speichern
    *uicrc=((*uicrc<<4|uiNibble1)&0xff)^CRC_LOOK_UP[uiNibble2];

    uiNibble1=(*uiIn>>4)&0x0f;
    uiNibble2=(*uicrc>>4);                      // oberes Nibble CRC speichern
    *uicrc=((*uicrc<<4|uiNibble1)&0xff)^CRC_LOOK_UP[uiNibble2];
}
```

Anhang 2

Prüfsummen

```
uiCRCb=0x55;
for (iLoop=0; iLoop<13; iLoop++)        // CRC Pruefsumme b  pruefen
{
    ApplyCrc(uiDataIn[iLoop],&uiCRCb);
}

uiXOR=0xaa;
for (iLoop=0; iLoop<14; iLoop++)        // XOR Pruefsumme pruefen
{
    uiXOR^=uiDataIn[iLoop];
}
```

Anhang 3:

Zusatzauflösung dekodieren

```
// i16Dummy enthält 16 Bit Wert (z.B. READ_RAM Rückgabewert)
// ucData enthält die serielle Kommunikation

cDummy=ucData[7+SER_OFFSET];

if((cDummy&0x80)==0x80)
{
      i16Dummy*=10;           // 16 Bit Wert * 10
      cDummy&=~0x80;          // oberstes Bit ausblenden

      if((cDummy&0x40)==0x40)
      {
            cDummy|=0xc0;     // neg Zahl wiederherstellen
      }

      i16Dummy+=cDummy;
}
```

History:

26.09.02	Start/Ende Test eingeführt
17.09.02	Umstellung: ab jetzt wird alles in Fahrenheit übertragen!
10.09.02	Erweiterung um Sicherheitscode beim Schreiben (*13) + entsprechender Fehlermeldung. Einheit entscheidet über °C/°F relativ/absolut
21.08.02	"Status" ergänzt (*12)
16.08.02	"Hidden" Funktion für Parameter eingefügt (*11)
17.01.03	"Versionsfehler" genau definiert.
09.06.03	Ergänzung zur Datenloggerabfrage
14.06.03	Ergänzung der ReadTime/Settime Befehle, damit Sperrung möglich wird. "ReadRamBurst" und "ReadRamBurstConfig" eingeführt
14.08.03	Read Time / Set Time: Adresse eingefügt; wird zwar nicht verwendet, aber die Daten stehen dann im Datenbereich, wie bei den anderen Token.
20.08.03	Logger: Fehlercode "0x0a" (Leerstelle im Datenbereich - Zeitstempel neu setzen).
16.09.03	"ReadEEprom burst" in "ReadData burst" umbenannt. "ReadDataInfo" - Intervall verkürzt.
28.09.03	Datenloggermode verändert
02.10.03	GetMaxTime eingef. Auf Seite 6 Ergänzung zur Anzahl der Parameter eingefügt.
14.02.04	Search String erweitert, es wird der ASCII Code mit zurückgegeben. Prüfschritte wurden überarbeitet.
03.03.04	Versionskodierung und Programmnummerkodierung genauer erklärt (V1.8 / DoS). Prüfsumme beim Parameterschreiben entfernt. ID für Lon eingefügt.
20.04.04	V3.0 Ergänzung: das "ACK" auf einen Request muß durch das Bit 6 im Tokenbereich gekennzeichnet werden. Unterscheidung der Implementierungsversionen über Command "BusVersion". Wird dieser Befehl nicht erkannt, handelt es sich um ST-Bus V2.0, der das "ACK" Bit nicht verwendet. Definition der BusMasterklassen
21.04.04	Neuer Befehl "ReadStatus"
22.04.04	SetStatus, ClearStatus, ToggleStatus (siehe 12*)
26.04.04	V3.1 2 Kleine Ergänzungen zur Berechnung der Parameteranzahl und "ReadNumber".
30.04.04	V 3.2 Sicherheitsprüfsummen bei Schreibbefehlen definiert. ReadRamBurst ohne Configbefehl.
26.07.04	V 3.3 Ergänzende Bemerkung zu "Readinfo" "Shut Up" / "Wake Up" Befehle generiert.
30.07.04	V 3.4 Beispielprogramme für Zusatzprüfsummen eingebaut. Statuswort auf 64 Bit verringert und Prüfsummen eingebaut.
02.08.04	"Ping" Befehl eingeführt.
09.08.04	"Ping" Befehl korrigiert.
10.08.04	ReadRamDebug eingeführt.

25.08.04	Tabelle Übersicht Token eingefügt
05.10.04	V3.5.2 kleine Fehlerkorrekturen (DoS)
13.10.04	V3.5.3 Korrekturen (*8) (DoS)
19.10.04	V3.5.4 ID bei Atmel Programmen erklärt.
12.11.04	V3.5.5 Fehler im Zusatzprüfsummen-Beispielprogr. beseitigt.
02.03.05	V3.5.6 Spezial Modus Feld für Datum (abwärtskompatibel zur bisherigen Definition).
14.03.05	V3.5.7 Fehler Prüfsummenprogramm beseitigt.
24.03.05	V3.5.8 Beispiel für Ebenenaufteilung eingefügt ReadRamBurst präzisiert.
02.05.05	„SetTime“ 0=Montag ergänzt
27.09.05	V3.5.9 Status, Fehlercode, ReadRamBurst, Valid-Bit
05.10.05	V3.5.10 Fahrenheit wird als Statusinfo-Bit übertragen
18.10.05	V3.5.11 Shutup, Wakeup entfernt. Broadcast Def. Korrigiert (Ping erwartet natürlich eine Antwort).
25.10.05	Gatewaykommando definiert
19.04.06	Kommandos 0x3f und 0x3e reserviert
02.11.06	Status64 (Read, Clear und Set) definiert in der Reihenfolge der Bytes
25.04.07	Return Code (*3) 0x0a fehlte in der Tabelle
07.05.07	(An) *2 „zusätzliche Dezimalstelle“: Definition genauer beschrieben, Name 1/100 entfernt, weil meiner Meinung nach dieser Name nur bei Temp. gilt, wir aber hier eine allgemeingültige Definition brauchen/einführen.
09.05.07	Sourcecode für "zus. Dezstelle" eingefügt, Sourcecode in Anhänge ausgelagert. Zusatzseite mit wichtigen Vereinbarung eingeführt.
14.09.07	Bootloader (BETA)
13.08.08	Einführung Prozent (= 21) als Einheit
03.04.09	Einheiten l/min und l/h eingeführt (DoS)
06.07.09	Datenlogger Protokoll V2.0 (RDE)
15.09.09	bReset in Statuswort eingeführt (sae)