Software-Design

Für das studentische Projekt *Sichere Eisenbahnsteuerung*

Beschreibung:	**Detaillierte Beschreibung des Subsystems Software**
Autor/en:	Ole Bohn
	Felix Geber
Ablageort:	Dokumente\02_Design\02.01_Subsystemdesign\ 02.01.02_noch nicht freigegebene Dokumente\Software-Design.pdf
Version	2.0
Status	freigegeben
Datei:	Software-Design.pdf
Datum:	24.06.2010

1 Historie

Version	Datum	Autoren	Änderung
0.0	13.04.10	O. Bohn F. Geber	Überarbeiten der Dokumente aus dem Projekt: F4 TI PROJEKT Bredereke WiSe0910
0.1	28.04.10	O. Bohn	Überarbeiten von Kapitel 6: Einfügen der Schnittstellenbeschreibung für die einzelnen Module. Darstellung des syntaktischen Aufrufs der Schnittstellen sowie deren dynamischem Verhalten bei einem Aufruf. Überarbeiten von Kapitel 4: Ergänzen der Liste referenzierter Dokumente um die Modul-Design-Dokument der in Kapitel 6 beschriebenen Module. Einfügen von Abschnitt 6.4.1 und 6.4.2.
0.11	29.04.10	F. Geber	Kleine grammatikalische Änderungen
0.12	03.05.10	K. Dziembala	Korrektur der Rechtschreibfehler
0.13	05.05.10	O. Bohn	Anpassen des Dokuments an das Review vom 03.05.2010. Abschnitt 4: Dateipfad ergänzen um Aulis. Abschnitt 6: Ergänzen des Schlüsselworts void bei der syntaktischen Beschreibung einiger Schnittstellen, Neuformulierung von Beschreibungen. Abschnitt 6.3.1: Entfernen des Satzes „Außerdem findet eine Prüfung der eingegangenen Daten auf Gültigkeit statt und ggf. wird ein Fehlersignal an die Sicherheitsschicht gesendet“ Abschnitt 6.3.1.2: Ändern des dynamischen Verhaltens beim Aufruf von get_sensor_data ().
0.14	18.05.10	O. Bohn	Anpassen von Querverweisen in Abschnitt 6.3.1 und 6.3.2. Abschnitt 6.2.3.1, 6.2.4.1, 6.3.5.1: Ergänzen der globalen Variablen des jeweiligen Moduls.
1.0	17.06.10	K. Dziembala	Freigabe des Softwaredesigns
2.0	24.06.10	F. Geber	Anpassen der Abb. 1 an die aktuelle Softwarearchitektur

2 Inhaltsverzeichnis

3 Einleitung

Das Software-Subsystemdesign für das Projekt "Sichere Eisenbahnsteuerung" beschreibt die Aufteilung und das Zusammenspiel aller benötigten Module für die softwaregetriebene Steuerung einer Modelleisenbahn. Das Design ist für die Anforderungen des Pflichtenhefts konzeptioniert.

Die aus dem Design resultierende Software ist für den parallelen Einsatz auf zwei Mikrocontrollern des Typs C515C bestimmt. Auf Subsystemdesign-Ebene existiert noch das Hardware-Design, welches die hardwareseitigen Randbedingungen für das Software-Design vorgibt.

In diesem Dokument soll neben der allgemeinen Beschreibung der Module die äußeren Schnittstellen dieser Module näher beleuchtet werden. Grund dieser Tätigkeit ist die Offenlegung der Schnittstellen, damit letztendlich ein Modul durch diese Schnittstellen auf Fehler, Zeitverhalten und ähnliches getestet werden können.

4 Referenzierte Dokumente

- Dokumente\01_Anforderungsanalyse\01.00_Pflichtenheft\01.00.02_noch nicht freigegebene Dokumente\Pflichtenheft.pdf
- Dokumente\02_Design\02.00_Systemdesign\02.00.02_noch nicht freigegebene Dokumente\System-Design.pdf
- Dokumente\02_Design\02.01_Subsystemdesign\02.01.02_noch nicht freigegebene Dokumente\Hardware-Design.pdf
- Aulis > Magazin > Fakultät 4: Elektrotechnik und Informatik > Technische Informatik (TI BSc) > F4 TI PROJEKT Bredereke WiSe0910 > Projektverzeichnis > 02: Design > 03: Moduldesign: Befehlsvalidierung-Modul-Design.pdf
- Aulis > Magazin > Fakultät 4: Elektrotechnik und Informatik > Technische Informatik (TI BSc) > F4 TI PROJEKT Bredereke WiSe0910 > Projektverzeichnis > 02: Design > 03: Moduldesign: Ergebnisvalidierung-Modul-Design.pdf
- Aulis > Magazin > Fakultät 4: Elektrotechnik und Informatik > Technische Informatik (TI BSc) > F4 TI PROJEKT Bredereke WiSe0910 > Projektverzeichnis > 02: Design > 03: Moduldesign: Auditing-System-Modul-Design.pdf
- Aulis > Magazin > Fakultät 4: Elektrotechnik und Informatik > Technische Informatik (TI BSc) > F4 TI PROJEKT Bredereke WiSe0910 > Projektverzeichnis > 02: Design > 03: Moduldesign: Betriebsmittelverwaltung-Modul-Design.pdf
- Aulis > Magazin > Fakultät 4: Elektrotechnik und Informatik > Technische Informatik (TI BSc) > F4 TI PROJEKT Bredereke WiSe0910 > Projektverzeichnis > 02: Design > 03: Moduldesign: Fahrprogramm-Modul-Design.pdf
- Aulis > Magazin > Fakultät 4: Elektrotechnik und Informatik > Technische Informatik (TI BSc) > F4 TI PROJEKT Bredereke WiSe0910 > Projektverzeichnis > 02: Design > 03: Moduldesign: Leitzentrale-Modul-Design.pdf
- Aulis > Magazin > Fakultät 4: Elektrotechnik und Informatik > Technische Informatik (TI BSc) > F4 TI PROJEKT Bredereke WiSe0910 > Projektverzeichnis > 02: Design > 03: Moduldesign: Befehlsvalidierung-Modul-Design.pdf
- Aulis > Magazin > Fakultät 4: Elektrotechnik und Informatik > Technische Informatik (TI BSc) > F4 TI PROJEKT Bredereke WiSe0910 > Projektverzeichnis > 02: Design > 03: Moduldesign: Auditingssystem-Modul-Design.pdf

5 Architektur

Die Architektur der Software ist in Abb. 1 dargestellt und repräsentiert die im Pflichtenheft definierten Anforderungen an das zu entwickelnde System. Hervorzuheben ist hier vor allem die Forderung nach einem sicheren System, das heißt nach einem System, dass es zu keinem Zeitpunkt zulässt, dass auf den Gleisanlagen der Modelleisenbahn ein unsicherer Zustand entsteht.

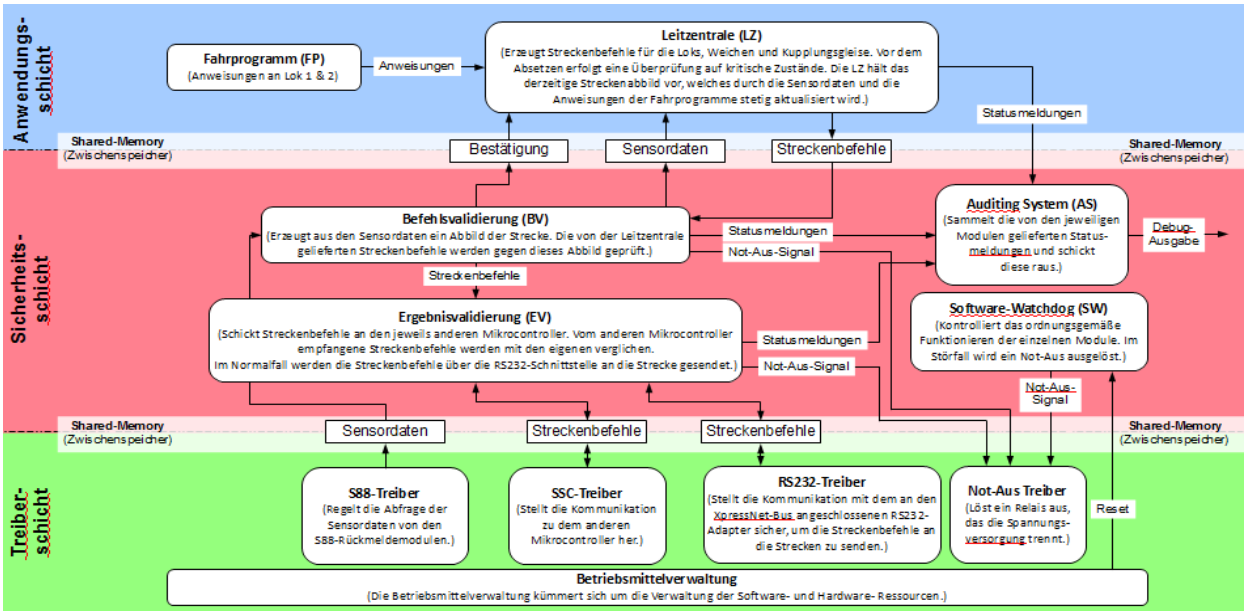

Abb. 1: Software-Architektur

Um dies zu gewährleisten, ist die Software nach den Kriterien für sicherheitsrelevante Systeme entwickelt worden. Der Aufbau ist in Anwendungs-, Sicherheits- und Treiberschicht getrennt.

Die Module in der Anwendungsschicht übernehmen die eigentliche Steuerung der Züge. Die Module der Sicherheitsschicht validieren die Ergebnisse der Anwendungsschicht und überprüfen die Plausibilität der von der Leitzentrale abgesetzten Befehle. Die Module der Treiberschicht stellen die Verbindung zwischen Hard- und Softwaresubsystem her. Zusätzlich werden die empfangenen Daten daraufhin überprüft, ob sie konform zum jeweiligen Protokoll sind und ggf. werden Fehlersignale an die Sicherheitsschicht übertragen.

Für den Datenaustausch zwischen den Schichten werden die Daten jeweils in einem Shared-Memory zwischengespeichert.

6 Decomposition

In diesem Abschnitt werden die einzelnen Softwaremodule des Systems „Sichere Eisenbahnsteuerung“ sowie deren Schnittstellen beschrieben. Hierbei wird sowohl der syntaktische Aufruf der Schnittstellen, als auch das Verhalten bei einem Aufruf dargestellt. Eine nähere Beschreibung der einzelnen Module kann in den jeweiligen Modul-Design-Dokumenten nachgelesen werden.

6.1 Beschreibung der Komponenten der Anwendungsschicht

6.1.1 Fahrprogramm

Das Fahrprogramm enthält eine Folge von Anweisungen, die von dem System ausgeführt werden müssen, damit die Loks ihre Fahraufgabe jeweils erfolgreich durchlaufen. Diese Anweisungen werden an die Leitzentrale gesendet, welche diese dann nach Abarbeitung bestätigt, sodass das Fahrprogramm die nächsten Anweisungen absetzen kann.

6.1.1.1 Syntaktik der Schnittstelle

void initFP();

get_command(byte lok)

6.1.1.2 Verhalten des Moduls bei Aufruf

Die Schnittstelle void initFP() wird aus der Betriebsmittelverwaltung heraus aufgerufen und dient dazu, das Modul Fahrprogramm zu initialisieren.

Durch den Aufruf der get_command(byte lok)-Methode aus dem Modul Leitzentralle wird ein Byte lok übergeben. Dieses wird anhand einer if-Anweisung ausgewertet und das dazugehörige Fahrprogramm aufgerufen. Das Objekt, auf das der Index zeigt, ist immer die aktuelle Fahranweisung der jeweiligen Lok und wird an die Leitzentrale zurückgegeben. Durch jeden Aufruf der get_command()-Funktion wird der Index inkrementiert.

6.1.2 Leitzentrale (LZ)

Die Leitzentrale erzeugt aus den, von den Fahrprogrammen kommenden, Anweisungen die jeweiligen Streckenbefehle für Loks, Weichen und Abkupplungsgleise. Vor dem Absetzen des Streckenbefehls an die Befehlsvalidierung erfolgt in der Leitzentrale eine Überprüfung, ob die Ausführung der Anweisungen das System in einen kritischen Zustand bringen könnte. In einem solchen Fall entscheidet die Leitzentrale, welcher Anweisung Vorrang gewährt wird und regelt die Geschwindigkeit bzw. bremst die nachrangig behandelte Lok. Die Leitzentrale hält das derzeitige Streckenabbild vor, welches durch die Sensordaten und die Anweisungen der Fahrprogramme stetig aktualisiert wird.

6.1.2.1 Syntaktik der Schnittstelle

```
void initLZ();

void workLZ();
```

Darüber hinaus gibt es keine Schnittstelle, die von einem anderen Modul aufgerufen werden kann. Die Leitzentrale schreibt lediglich Daten in den Shared-Memory und kommuniziert über diesen mit den anderen Modulen.

6.1.2.2 Verhalten des Moduls bei Aufruf

Die Schnittstelle void initLZ() wird aus der Betriebsmittelverwaltung heraus aufgerufen und dient zur Initialisierung des Moduls. Analog dazu wird auch die Schnittstelle void workLZ() aus dem Modul Betriebsmittelverwaltung heraus aufgerufen.

Da bei diesem Modul die Interaktion mit anderen Modulen, ausgenommen der Betriebsmittelverwaltung, nicht über eine explizite Schnittstelle, sondern über den Shared-Memory erfolgt, wird in diesem Abschnitt das Verhalten nicht näher beschrieben.

6.2 Beschreibung der Komponenten der Sicherheitsschicht

6.2.1 Befehlsvalidierung (BV)

Die Befehlsvalidierung erzeugt aus den Sensordaten, die vom S88-Treiber empfangen werden, ein Abbild der Strecke. Die von der Leitzentrale gelieferten Streckenbefehle werden gegen dieses Abbild geprüft. Sollte sich daraus ein inkonsistenter Zustand ergeben, so wird das System angehalten. Andernfalls wird der Streckenbefehl der Leitzentrale quittiert und an die Ergebnisvalidierung (EV) weitergeleitet. Sollte von der Treiberschicht ein kritischer Fehler gemeldet werden, so wird ein Not-Aus-Signal gesendet.

6.2.1.1 Syntaktik der Schnittstelle

void initBV(void);

void workBV (void);

Neben diesen beiden Methoden stehen vier globale Variablen zur Verfügung, die vom Modul Befehlsvalidierung erstellt werden und von anderen Modulen (z.B. von der Leitzentrale) genutzt werden können.

BV_streckentopologie: Gleisabschnitt[]

BV_gleisBelegung: int8[]

BV_weichenBelegung: int8[]

BV_zugPosition: int8[]

6.2.1.2 Verhalten des Moduls bei Aufruf

Die Schnittstellen void initBV(void) und void workBV(void) werden aus der Betriebsmittelverwaltung heraus aufgerufen und dienen zur Initialisierung bzw. zum Modulaufruf.

Verbunden mit der Initialisierung des Moduls bei einem Aufruf von void initBV(void) ist die Übergabe einer Kopie der Streckenbelegung an die Leitzentrale. Dazu wird die Methode copyStreckenBelegung() intern aufgerufen.

Ein Aufruf der Schnittstelle void workBV(void) führt zu dem nachfolgend beschriebenen Verhalten. Entsprechend einer Variablen nextState wird zwischen vier unterschiedlichen Aktionen unterschieden: Es werden die Sensordaten überprüft, falls neue Daten eingetroffen sind. Andernfalls werden keine Aktionen ausgeführt. Gegebenenfalls ist hieraus ein Not-Aus zu generieren, sofern die Sensordaten nicht in Ordnung sind. Darüber hinaus wird das Gleisabbild auf kritische Zustände überprüft. Ist dies der Fall wird ebenfalls ein Not-Aus generiert. Im nächsten Schritt werden die Streckenbefehle überprüft. Fällt diese Überprüfung positiv aus, werden diese an die Ergebnisvalidierung gesendet. Abschließend wird die Streckentopologie überprüft. Sollte hierbei ein Fehler auftreten, wird ebenfalls ein Not-Aus erzeugt.

Die vier globalen Variablen dienen, wie bereits beschrieben, dem Modul Leitzentrale. Dies erhält daraus unter anderem eine Kopie der Streckentopologie sowie einen Teil des aktuellen Gleisabbildes.

6.2.2 Ergebnisvalidierung (EV)

Die EV empfängt die von der Befehlsvalidierung weitergeleiteten Streckenbefehle. Diese schickt sie über den SSC-Treiber an den jeweils anderen Mikrocontroller, der dies ebenso macht. Ist der Streckenbefehl vom anderen Mikrocontroller empfangen, so wird er mit dem selbst errechneten verglichen. Unterscheiden sich die Streckenbefehle, wird ein Not-Aus-Signal ausgegeben, ansonsten schickt der Mikrocontroller, der über die RS232-Schnittstelle mit dem XpressNet verbunden ist, den Streckenbefehl an den RS232-Treiber. Sollten von den Treibern (RS232, SSC) kritische Fehler gemeldet werden, so ist ebenfalls ein Not-Aus-Signal zu senden.

6.2.2.1 Syntaktik der Schnittstelle

```
void initEV(void);

void workEV(void);
```

6.2.2.2 Verhalten des Moduls bei Aufruf

Die Schnittstelle void initEV(void) wird von der Betriebsmittelverwaltung aufgerufen und dient dem Initialisieren des Moduls. Nach dem Aufruf dieser Methode, wurden alle intern im Modul verwendeten Variablen zurückgesetzt.

Die Methode void workEV(void) wird ebenfalls aus der Betriebsmittelverwaltung heraus aufgerufen und stellt die gesamte Funktionalität der Ergebnisvalidierung zur Verfügung. Bedingung für den Aufruf ist, dass das Modul vorher bereits initialisiert wurde. Innerhalb der Methode ist das Verhalten bei einem Aufruf in 6 Verarbeitungsschritte unterteilt:

1. Kommunikationsprobleme mit den Treibern überprüfen
2. Internen Streckenbefehl verarbeiten
3. Externen Streckenbefehl verarbeiten
4. Streckenbefehle vergleichen
5. Senden des internen Streckenbefehls an den RS 232 Treiber
6. Interne Variablen zurücksetzen

6.2.3 Auditing-System (AS)

Das Auditing-System sammelt die von den jeweiligen Modulen gelieferten Statusmeldungen (Ergebnisvalidierung, Befehlsvalidierung und Leitzentrale) und stellt diese zum Auslesen über eine interne Hardware-Schnittstelle zur Verfügung. Aus den gespeicherten Statusmeldungen muss der zeitliche Ablauf der Ereignisse hervorgehen.

6.2.3.1 Syntaktik der Schnittstelle

void initAS();

void workAS();

void sendMsg(byte msg[6], byte module_id);

void reportAllMsg();

Darüber hinaus werden die vier folgenden globalen Variablen zur Verfügung gestellt:

xdata byte AS_msg_array[MAX_MELDUNGEN][7];

byte AS_read_next_msg;

byte AS_fill_next_msg;

byte AS_msg_counter;

6.2.3.2 Verhalten des Moduls bei Aufruf

Die Schnittstelle void initAS() wird von der Betriebsmittelverwaltung zu Beginn der Laufzeit aufgerufen und führt dazu, dass die Steuerungsvariablen und der Ringpuffer initialisiert werden. Nach dem Aufruf dieser Schnittstelle werden die Bytes AS_read_next_msg, AS_fill_next_msg und AS_msg_counter jeweils auf 0 gesetzt.

Analog dazu wird auch die Schnittstelle void workAS() von der Betriebsmittelverwaltung aufgerufen. Damit dieser Aufruf erfolgen kann, muss vorher die Methode void initAS() aufgerufen werden. Nach einem Aufruf wird überprüft, ob der Ringpuffer leer ist. Ist dies der Fall, müssen keine weiteren Aktionen ausgeführt werden. Andernfalls wird überprüft, ob der Lesezeiger AS_read_next_msg auf einen gültigen Index zeigt. Daraufhin wird eine definierte Wartezeit eingelegt, bevor eine schreibende Verbindung zum Arduino aufgebaut wird. Im Anschluss daran werden maximal vier Meldungen aus dem Ringpuffer ausgelesen und versendet.

Die Schnittstelle void reportAllMsg() wird vom Not-Aus-Treiber aufgerufen und dient dem Auslesen aller Meldungen der Module aus dem Ringpuffer. Bevor diese Methode aufgerufen werden kann, muss void initAS() ausgeführt worden sein. Des Weiteren muss der Watchdog deaktiviert werden.

Nach Aufruf dieser Schnittstelle wird überprüft, ob der Schreibezeiger *AS_fill_next_msg* auf einen gültigen Index zeigt. Ggf. wird dieser durch

Nullsetzen korrigiert. Dadurch soll sichergestellt werden, dass im Folgenden nicht wahllos in den Speicher geschrieben werden
kann.

Die Schnittstelle void sendMsg(byte msg[6], byte module_id) dient dem Aufnehmen aktueller Statusmeldungen der Module.

6.2.4 Software Watchdog (SW)

Der SW wird jedes Mal, wenn ein von der Betriebsmittelverwaltung aufgerufenes Modul zurückkehrt, von dieser zurückgesetzt. Beim Zurücksetzen setzt der SW einen Hardware Timer des Mikrocontrollers zurück. Bleibt ein Rücksetzen aus, so sendet der SW ein Not-Aus-Signal. Sollte ein Modul zurückkehren, bevor es seine Aufgabe fertig abgearbeitet hat, so schickt es eine Nachricht an den SW, die den aktuellen Status enthält. Falls von einem Modul zwei Mal der gleiche Status gesendet wird, so geht der SW davon aus, dass das Modul keinen Fortschritt erzielt hat und wahrscheinlich in einer Endlosschleife hängen geblieben ist. In diesem Fall wird wiederum ein Not-Aus-Signal ausgegeben.

6.2.4.1 Syntaktik der Schnittstelle

void initSW();

void hello();

void helloModul(byte modul_id, byte status);

void StopSW();

Darüber hinaus wird folgende globale Variable zur Verfügung gestellt:

byte SW_status_array[7];

6.2.4.2 Verhalten des Moduls bei Aufruf

Die Schnittstelle void initSW() wird vom Modul Betriebsmittelverwaltung zu Beginn der Laufzeit einmal aufgerufen.

Nach dem Aufruf dieser Schnittstelle wird zuerst der Hardwarezähler Timer0 des Mikrocontrollers C515C gestoppt. Danach wird das Speicherfeld für den zuletzt gemeldeten Status der Module in Form eines globalen Arrays mit dem Wert FF_H gefüllt. Dann wird der Interrupt Timer0Overflow freigegeben und Timer0 wieder aktiviert (siehe Abschnitt 4, Referenzierte Dokumente, Modul - Design Software Watchdog).

Der Aufruf der Schnittstelle void hello() erfolgt ebenfalls aus dem Modul Betriebsmittelverwaltung heraus und dient zur Rückmeldung innerhalb des vorgegebenen Zeitfenster zur Bewältigung von Aufgaben. Nach dem Aufruf wird der Hardwarezähler Timer0 des Mikrocontrollers C515C auf null zurückgesetzt.

Ein Aufruf der Schnittstelle void helloModul(byte module_id, byte status) führt dazu, dass zunächst überprüft wird, ob die erhaltene ModulID ungültig ist. In diesem Fall wird der Not-Aus-Treiber kontaktiert und somit der Programmfluss angehalten. Danach wird geprüft, ob der erhaltene Status bereits durch einen vorherigen Aufruf im globalen Array gesetzt wurde. In diesem Fall wird ebenso der Not-Aus-Treiber kontaktiert. Sollten beide Prüfungen negativ verlaufen sein, das heißt es ist kein Fehler aufgetreten, so wird der erhaltene Status im globalen Array gespeichert.

Abschließend sorgt ein Aufruf der Schnittstelle void StopSW() dafür, dass der Hardwarezähler Timer0 des Mikrocontrollers C515C inaktiv geschaltet wird. Um dies zu erreichen, wird TR0 = 0 gesetzt.Aufgerufen werden kann diese Schnittstelle nur vom Modul Not-Aus-Treiber.

Das globale Array byte SW_status_array[7] dient zum Speichern der Status der sechs zu überwachenden Module. Hierbei ist jedes Element in dem Array einem Modul zugeordnet.

6.2. Beschreibung der Komponenten der Treiberschicht

6.2.5 S88-Treiber

Der S88-Treiber regelt die Abfrage der Sensordaten von den S88-Rückmeldemodulen. Hierzu gehört auch die Erzeugung der jeweils benötigten Signale für den S88-Bus (CLOCK, LOAD, RESET).

6.2.5.1 Syntaktik der Schnittstelle

```
void initS88 (void);
void workS88 (void);
get_sensor_data();
```

6.2.5.2 Verhalten des Moduls bei Aufruf

Die Schnittstelle void initS88(void) wird von dem Modul Betriebsmittelverwaltung aufgerufen und dient der Initialisierung des Moduls S88-Treiber. Nach dem Aufruf werden die Variablen S88_PS, S88_RESET, S88_Clk, S88_Data, sensor_count, S88_BV_sensordaten.Byte0 und S88_BV_sensordaten.Byte1 initialisiert.

Der Aufruf von void workS88(void) erfolgt ebenfalls aus dem Modul Betriebsmittelverwaltung heraus. Hier wird die Funktionalität des Moduls zur Verfügung gestellt, das heißt, die Sensordaten werden abgefragt.

Die Schnittstelle get_sensor_data() des Moduls S88-Treiber wird durch die Betriebsmittelverwaltung aufgerufen und ruft die Daten von den S88-Rückmeldemodulen ab.

6.2.6 SSC-Treiber

Der SSC-Treiber stellt die Kommunikation zwischen den Mikrocontrollern her. Außerdem findet eine Prüfung der vom Shared-Memory ausgelesenen Daten auf Gültigkeit statt und ggf. wird ein Fehlersignal an die Sicherheitsschicht gesendet. Dabei soll überprüft werden, ob die empfangenen Befehle definierte Streckenbefehle sind. Über den SSC-Bus werden die von den Mikrocontrollern unabhängig voneinander berechneten Streckenbefehle zum Vergleich zum jeweils anderen Mikrocontroller gesendet bzw. von diesem empfangen.

6.2.6.1 Syntaktik der Schnittstelle

void workSSC();

void initSSC();

6.2.6.2 Verhalten des Moduls bei Aufruf

Nach dem Aufruf der Schnittstelle void workSSC() wird der Status überprüft, in dem das Modul beim letzten Zugriff verlassen wurde. Ist dies erfolgt, wird in den meisten Fällen die Methode datenSenden() ausgeführt. Nähere Details zum weiteren Ablauf können im entsprechenden Modul-Design-Dokument nachgelesen werden (siehe Abschnitt 4, Referenzierte Dokumente).

Durch den Aufruf der Schnittstelle initSSC() wird die Schnittstelle konfiguriert.

Der Aufruf der beiden beschriebenen Schnittstellen erfolgt aus dem Modul Betriebsmittelverwaltung heraus.

6.2.7 RS232-Treiber

Der RS232-Treiber stellt die Kommunikation mit dem an den XpressNet-Bus angeschlossenen RS232-Adapter sicher, um die Streckenbefehle an die Lokomotiven, Weichen und Abkupplungsgleise zu senden. Außerdem findet eine Prüfung der vom Shared-Memory ausgelesenen Daten auf Gültigkeit statt und ggf. wird ein Fehlersignal an die Sicherheitsschicht gesendet. Dabei soll überprüft werden, ob die empfangenen Befehle definierte Streckenbefehle sind. Der RS232-Treiber wird nur verwendet, wenn der Mikrocontroller über den RS232-Bus mit dem RS232- Adapter mit dem XpressNet verbunden ist.

6.2.7.1 Syntaktik der Schnittstelle

void workRS232();

void initRS232();

6.2.7.2 Verhalten des Moduls bei Aufruf

Die oben genannten Schnittstellen des Moduls RS232 Treiber werden beide aus dem Modul Betriebsmittelverwaltung heraus aufgerufen. Die Methode void initRS232() dient hierbei wiederum dem Initialisieren des Moduls.

Void workRS232() hingegen stellt den Großteil der Funktionalität zur Verfügung. Ausgehend von dieser Methode werden die anderen lokalen Funktionen innerhalb des Moduls aufgerufen. Nähere Details können im Moduldesigndokument nachgelesen werden (siehe Abschnitt 4: Referenzierte Dokumente).

6.2.8 Not-Aus-Treiber

Der Not-Aus-Treiber empfängt im Fehlerfall ein Signal, das ihn dazu veranlasst ein Relais, das mit der Spannungsversorgung verbunden ist, zu schalten. Dadurch soll das System im Notfall zum Stillstand gebracht werden.

6.2.8.1 Syntaktik der Schnittstelle

void emergency_off(void);

void initNOTAUS(void);

6.2.8.2 Verhalten des Moduls bei Aufruf

Wird die Schnittstelle void emergency_off(void) des Moduls Not-Aus Treibers aufgerufen, wird die Strecke stromlos geschaltet. Dazu wird der Port 2.0 des Not - Aus - Relais genutzt. Des Weiteren wird der Software Watchdog angehalten sowie die verbliebenen Meldungen der Module werden versendet.

Die Schnittstelle void initNOTAUS (void) dient zum initialisieren des Moduls und wird aus der Betriebsmittelverwaltung heraus aufgerufen.

6.2.9 Betriebsmittelverwaltung

Die Betriebsmittelverwaltung kümmert sich um die Verwaltung der Software- und Hardware-Ressourcen. Sie reserviert für jedes Modul ausreichend Speicherplatz und betreibt das Scheduling. Hierfür wird Kooperatives Multitasking eingesetzt. Dabei wird jedem Modul ein festgelegtes Zeitintervall zugeteilt, innerhalb dessen es seine Aufgaben erledigt haben muss und die Kontrolle an die Betriebsmittelverwaltung zurückgegeben hat.

Wird dieses Modul vor Ende des Zeitintervalls nicht fertig, so muss es die zur Fortführung der Aufgabe benötigten Ressourcen im Memory zwischenspeichern und bei der nächsten Zuteilung eines Zeitintervalls an der richtigen Stelle wieder einsetzen.
Sollte ein Modul innerhalb des zugewiesenen Zeitintervalls zurückkehren, so wird der Software Watchdog durch ein Reset- Signal zurückgesetzt. Neben Speicherplatzverwaltung und Scheduling behandelt die Betriebsmittelverwaltung die Zuteilung von weiteren Ressourcen wie Timern, Parallelports und seriellen Ports.

6.2.9.1 Syntaktik der Schnittstelle

Das Modul weist aufgrund seiner Funktion innerhalb des Programms keine Schnittstelle auf, die von den anderen Modulen aufgerufen werden kann. Vielmehr ist die Betriebsmittelverwaltung, wie im vorangegangenen Abschnitt beschrieben, für die Verwaltung der Zeitscheiben, also für den Aufruf der anderen Module zuständig.

Unabhängig davon stellt die Betriebsmittelverwaltung jedoch eine Reihe von globalen Konstanten zur Verfügung, die von den anderen Modulen genutzt werden können.

Streckenbefehl LZ_BV_streckenbefehl = {LEER,LEER,LEER,0};

Streckenbefehl BV_EV_streckenbefehl = {LEER,LEER,LEER,0};

Streckenbefehl SSC_EV_streckenbefehl = {LEER,LEER,LEER,0};

Streckenbefehl EV_SSC_streckenbefehl = {LEER,LEER,LEER,0};

Streckenbefehl EV_RS232_streckenbefehl = {LEER,LEER,LEER,0};

Streckenbefehl RS232_EV_streckenbefehl = {LEER,LEER,LEER,0};

Sensordaten BV_LZ_sensordaten = {LEER,LEER,0};

Sensordaten S88_BV_sensordaten = {LEER,LEER,0};

byte BV_LZ_bestaetigung = LEER;

byte EV_SSC_failure = LEER;

byte SSC_EV_failure = LEER;

6.2.9.2 Verhalten des Moduls bei Aufruf

Dieser Abschnitt entfällt bei dem Modul Betriebsmittelverwaltung da kein Funktionsaufruf durch andere Module erfolgt. Die zur Verfügung gestellten globalen Variablen können jedoch von den anderen Modulen genutzt werden.

6.3 Beschreibung der (globalen) Datenstrukturen

6.3.1 Shared-Memory zwischen Anwendungsschicht und Sicherheitsschicht

Die Befehlsvalidierung liest, sofern von der Leitzentrale neue Daten für sie im Shared-Memory hinterlegt wurden, diese aus und verarbeitet die Daten. Nach erfolgreicher Prüfung der Daten wird dies im Shared-Memory quittiert.

Der Shared-Memory setzt sich aus den Bereichen für Sensordaten und Streckenbefehle zusammen. Ein Byte der Streckenbefehle oder der Sensordaten wird als „nicht gesetzt“ durch Belegung mit dem Wert „0xFF“ markiert.

Streckenbefehle:

Geschwindigkeit (Zug, Richtung, Stufe), (1 Byte)

Parameter	Stufe	Richtung	Zug
Bitposition	7..2	1	0
Kodierung	0: Stand 1-62: Geschwindigkeitsbetrag (Aufsteigend)	0: Rückwärts 1: Vorwärts	0: Lok #1 1: Lok #2

Weichen stellen (Weiche, Stellung), (1 Byte)

Parameter	Weiche	Stellung
Bitposition	7..1	0
Kodierung	0-126: Weiche #	0: Geradeaus 1: Abbiegen

Abkuppeln (Kupplungsgleis, Aktion), (1 Byte)

Parameter	Kupplungsgleis	Aktion
Bitposition	7..1	0
Kodierung	0-126: Kupplungsgleis #	0: Kupplung anheben 1: Kupplung senken

Zusätzlich enthält jeder Streckenbefehl ein 1 Byte-großes Fehlerfeld, das jedoch im Shared-Memory zwischen Anwendungs- und Sicherheitsschicht nicht benutzt wird und mit „0xFF“, also „nicht gesetzt“ belegt wird (siehe Kap. 6.3.2).

Sensordaten:

Weichenstellung (Weiche, Stellung), (1 Byte)

Parameter	Weiche	Stellung
Bitposition	7..1	0
Kodierung	0-126: Weiche #	0: Geradeaus 1: Abbiegen

Gleispunktbelegung, (2 Byte)

Parameter	Gleispunktbelegung
Bitposition	15..0
Kodierung	Jedes Bit steht für die Belegung eines Gleis-Sensors. Ist Gleis-Sensor 1 belegt, so ist zum Beispiel Bit 0 gesetzt. Beispiel: 0000000000010100 Gleis-Sensor 3 und 5 belegt.

Bestätigung:

Befehlsbestätigung (1 Byte)

Parameter	Befehlsbestätigung
Adressbereich	7..0
Kodierung	0: Unbestätigt 1: Bestätigt

6.3.2 Shared-Memory zwischen Treiberschicht und Sicherheitsschicht

Die Treiber der Treiberschicht legen in den ihnen zugeordneten Bereichen des Shared-Memory die von den Hardwareschnittstellen erhaltenen Daten ab bzw. holen sich aus dem Shared-Memory die von der Sicherheitsschicht für sie dort hinterlegten Daten ab und senden diese über die Hardwareschnittstellen.

Der Shared-Memory setzt sich aus den Bereichen für Sensordaten (S88) und Streckenbefehle (RS32,SSC) sowie Bereichen zum Speichern von Fehlern, die von den Treibern erkannt wurden, zusammen. Die Kodierung der Fehler ist den Modulen selbst überlassen. Je nach Schwere des Fehlers kann der Fehler als kritisch oder normal eingestuft werden. Alle Fehler sollten mit Hilfe des Auditing Systems protokolliert werden, kritische Fehler müssen zum Ausführen des Not-Aus führen. Zusätzlich kann die Betriebsmittelverwaltung im Debug-Modus Debug-Informationen aus dem Shared-Memory holen und diese zur weiteren Auswertung über den RS232-Bus schicken.

Sensordaten (2 Byte)

Parameter	Weichenstellung	Gleispunktbelegung	Fehler
Bitposition	7..1	0	7..0
Kodierung	Siehe Kapitel 6.3.1 (Streckenbefehle, Kodierung)		0: Kein Fehlerfall 1-127: Normaler Fehler 128-255: Kritischer Fehler

Streckenbefehle (SSC schreibend, Ergebnisvalidierung lesend), (4 Byte)

Parameter	Geschwindigkeit	Weiche stellen	Abkuppeln	Fehler
Bitposition	7..0	7..0	7..0	7..0
Kodierung	Siehe Kapitel 6.3.1 (Streckenbefehle, Kodierung)			0: Kein Fehlerfall 1-127: Normaler Fehler 128-255: Kritischer Fehler

Streckenbefehle (SSC lesend, Ergebnisvalidierung schreibend), (4 Byte)

Parameter	Geschwindigkeit	Weiche stellen	Abkuppeln	Fehler
Bitposition	7..0	7..0	7..0	7..0
Kodierung	Siehe Kapitel 6.3.1 (Streckenbefehle, Kodierung)			0: Kein Fehlerfall 1-127: Normaler Fehler 128-255: Kritischer Fehler

Streckenbefehle (RS232), (4 Byte)

Parameter	Geschwindigkeit	Weiche stellen	Abkuppeln	Fehler
Bitposition	7..0	7..0	7..0	7..0
Kodierung	Siehe Kapitel 6.3.1 (Streckenbefehle, Kodierung)			0: Kein Fehlerfall 1-127: Normaler Fehler 128-255: Kritischer Fehler

7 Quellenverzeichnis

- Poster „Sichere Eisenbahnsteuerung“ aus dem WiSe 09/10