Modul-Design SSC-Modul

Für das studentische Projekt *Sichere Eisenbahnsteuerung*

Datum	18.01.2011
Quelle	Google Code → Dokumente → 02_Design → 02.02_Moduldesign
Autoren	Norman Nieß Kai Dziembala
Version	1.0
Status	freigegeben

1 Historie

Version	Datum	Autor	Bemerkung
0.1	24.11.2009	Matthias Buß	Initial.
0.2	26.11.2009	Matthias Buß	Einleitung, Architektur
0.3	06.12.2009	Matthias Buß	Kapitel 5.4 umstrukturiert, Kapitel 6 geschrieben, kleine Korrekturen
0.4	10.12.2009	Matthias Buß	Kapitel 6 überarbeitet
0.5	13.12.2009	Matthias Buß	Kapitel 6 neu entworfen
0.6	03.01.2010	Matthias Buß	Kapitel 6 um SSC interrupt erweitert, Änderungen der Datentypen in Kapitel 6
0.7	11.01.2010	Matthias Buß	Variablen in 5.2 hinzugefügt, Methoden in 6. hinzugefügt und entfernt, Texte und UML-Diagramme in 6. verändert
0.8	13.01.2010	Matthias Buß	Work() umbenannt, initSSC() hinzugefügt, byteAufGueltigkeitPruefen() entfernt
0.9	05.05.2010	Kai Dziembala Norman Nieß	Abgleich mit dem vorhandenen Quellcode, Entfernung der äußeren Schnittstellen, Anpassung des Layouts, Korrektur von Rechtschreibfehlern
1.0	17.06.2010	Kai Dziembala	Freigabe des Moduldesigns 'SSC-Treiber'

2 Inhaltsverzeichnis

3 Einleitung

Die beiden redundant angelegten Mikrocontroller sind laut Hardware-Design direkt über ihre SSC-Schnittstelle miteinander verbunden. Dazu verfügt jeder Mikrocontroller über ein SSC-Modul, das den Datenaustausch steuern soll. Ein Mikrocontroller muss sowohl Daten an den anderen senden, als auch auch die Daten des anderen einlesen können. Das SSC-Modul liegt in der Treiberschicht des Mikrocontrollers und greift auf bestimmte Bereiche des Shared Memory zu.

4 Referenzierte Dokumente

Hardware-Design: Google Code → Dokumente → 02_Design → 02.01_Subsystemdesign → Hardware-Design

Software-Design: Google Code → Dokumente → 02_Design → 02.01_Subsystemdesign → Software-Design

5 Architektur

5.1 Funktionshierarchie

Das SSC-Modul befindet sich in der Treiberschicht und greift auf einen definierten Speicherbereich im Shared Memory zu. Damit keine Daten anderer Module beschädigt werden, ist auf die Ansteuerung des Speichers besondere Aufmerksamkeit zu legen.

5.2 Daten

In dem Modul sind die beiden Byte-Variablen „byteToSend“ und „byteToReceive“ vorgesehen. Diese sollen nur die Werte 1, 2 und 3 annehmen können. Außerdem ist ein Byte Array „temp_streckenbefehl[]“ mit drei Elementen vorgesehen, um die einzelnen empfangenen Bytes zwischen zu speichern.

5.3 Abhängigkeiten von anderen Modulen

Damit der Sotfware-Watchdog das SSC-Modul eindeutig zuweisen kann, hat dieses die module_id 4. Gibt das Modul die Ressourcen des Mikrocontrollers frei obwohl eine Aufgabe noch nicht abgearbeitet ist, so wird dem Watchdog eine Nachricht mit einem Statusbyte gesendet, das den Fortschritt der Abarbeitung wiedergibt.

Dies geschieht über die Schnittstelle hello(byte module_id, byte status) des Watchdogs. Das SSC-Modul hat einen festgelegten Speicherbereich, in den geschrieben werden darf und einen, aus dem nur gelesen werden darf.

6 Dynamisches Verhalten

Der Ablauf in der Funktion workSSC() wird Abbildung 1 dargestellt:

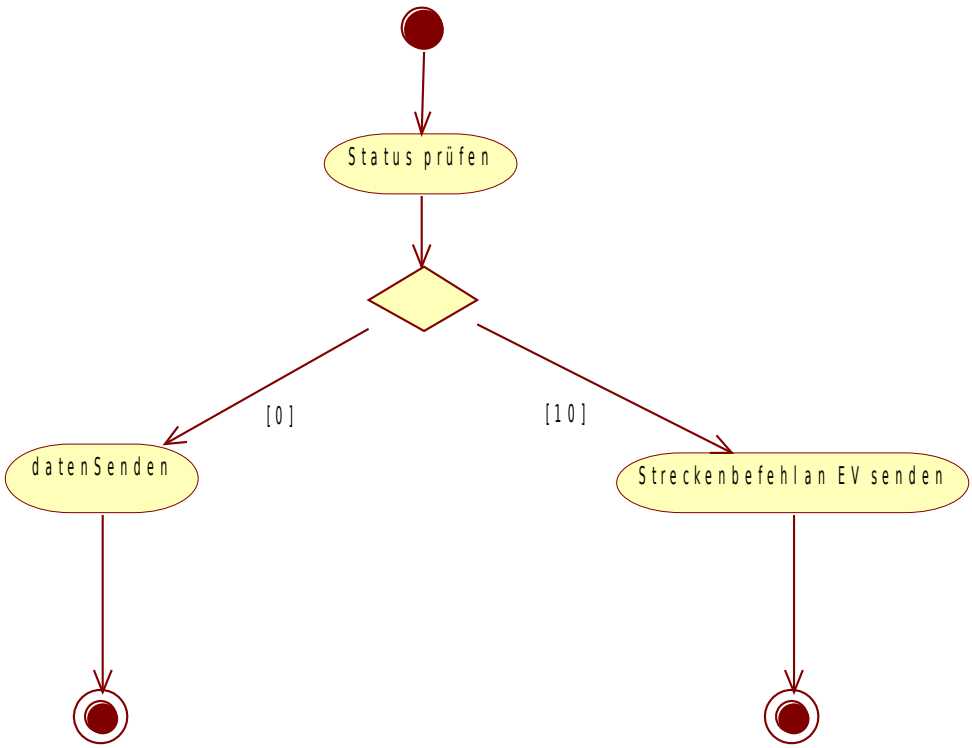

Abbildung 1: Aktivitätsdiagramm workSSC()

Am Anfang der Funktion workSSC() wird geprüft, mit welchem Status das Modul verlassen wurde. An diesem Punkt soll dann nach Prüfung einer Statusvariablen wieder weitergearbeitet werden. Standardmäßig wird die Aktivität datenSenden gewählt.

Diese läuft wie in Abbildung 2 beschrieben ab:

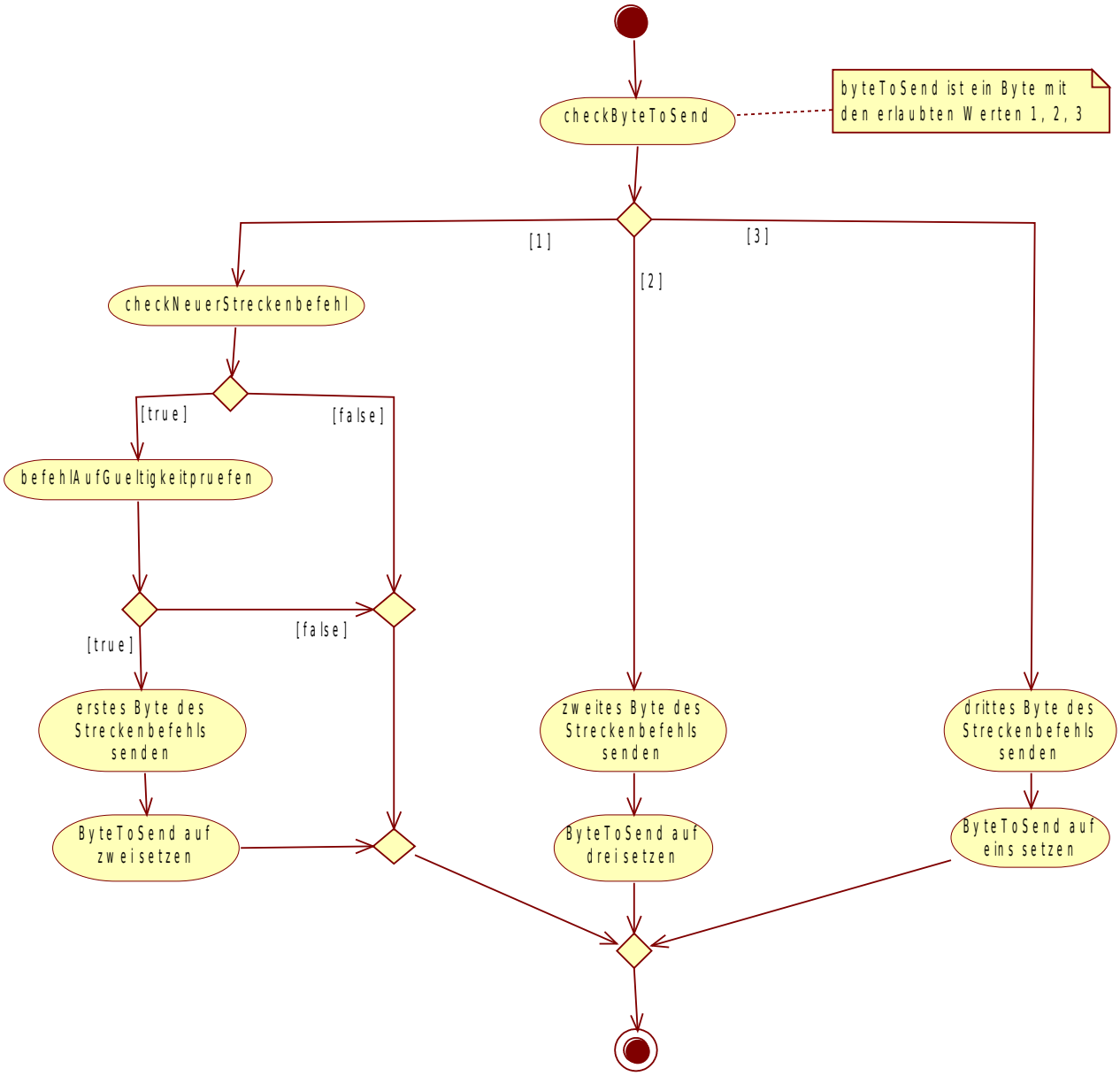

Abbildung 2: Aktivitätsdiagramm der Methode datenSenden()

In datenSenden() wird zunächst geprüft, welches Byte des Streckenbefehls zum anderen Mikrocontroller gesendet werden soll. Um diese Information zu speichern wird die Variable byteToSend benötigt, welche nur die Werte eins, zwei oder drei annehmen darf. Soll das zweite oder dritte Byte versendet werden, so wird das entsprechende Byte in das STB-Register geschrieben und die Variable ByteToSend um eins erhöht, wobei der Wert drei auf eins „erhöht“ wird.

Soll das erste Byte versendet werden, so wird geprüft, ob ein neuer Streckenbefehl in den Shared Memory geschrieben wurde. Besteht der Streckenbefehl aus lauter Einsen, so ist dies kein gültiger neuer Streckenbefehl. Dann wird nichts an den anderen Mikrocontroller gesendet. Ansonsten ist ein neuer Streckenbefehl vorhanden. Solange das SSC-Modul den zu sendenden Streckenbefehl noch nicht versendet hat, darf dieser von keinem anderen Modul überschrieben werden. Nach erfolgreichem Senden wird der Streckenbefehl mit lauter Einsen überschrieben und so anderen Modulen mitgeteilt, dass ein neuer Streckenbefehl gespeichert werden darf.

Wurde ein neuer Streckbefehl erkannt, so wird dieser mit der Funktion befehlAufGueltigkeitPruefen(Boolean) zunächst validiert. Ist dieser gültig, so wird das STB-Register mit dem ersten Byte des Streckenbefehls beschrieben und ByteToSend wird auf den Wert zwei erhöht.

Wurde eine Übertragung beendet oder eine Kollision festgestellt, wird ein Interrupt erzeugt. Dabei wird die Methode SSCinterrupt() aufgerufen. Darin wird zunächst geprüft, welches Ereignis den Interrupt ausgelöst hat und dann darauf reagiert, siehe Abbildung 2.

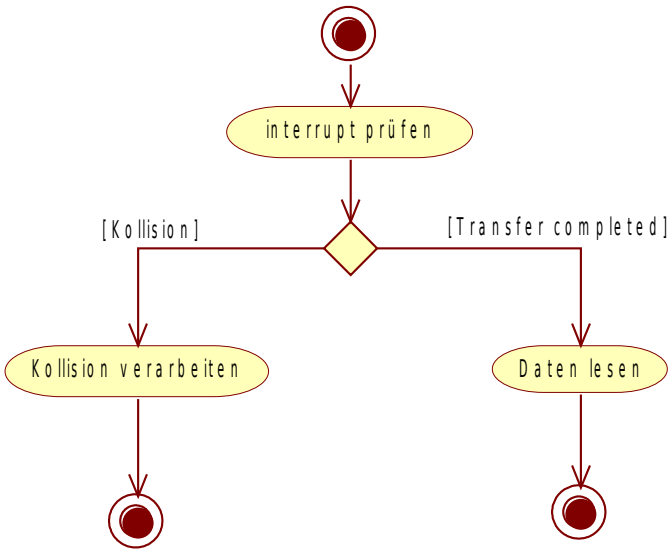

Abbildung 3: Aktivitätsdiagramm SSCinterrupt

Wurde ein Kollision festgestellt, so werden die Variablen, die das zu sendende und zu lesende Byte angeben dekrementiert und das Kollisionsbit im SCF Register gelöscht.

Wurde ein Übertragung erfolgreich beendet, so wird die Methode datenLesen() aufgerufen, deren Ablauf in Abbildung 3 beschrieben wird.

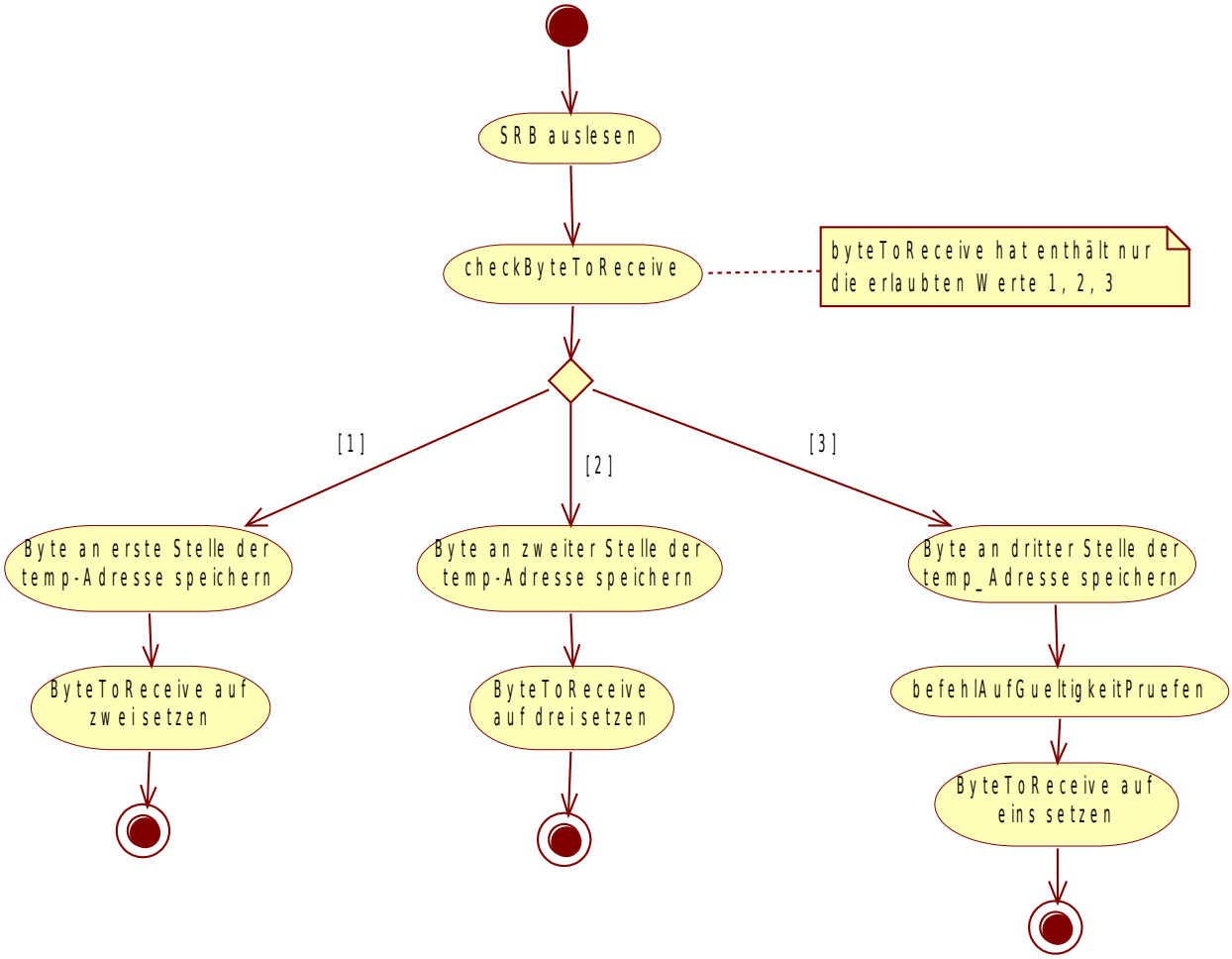

Abbildung 4: Aktivitätsdiagramm datenLesen()

Zunächst wird abgefragt, welches Byte des Streckenbefehls empfangen wurde. Wurde das erste Byte empfangen, so wird es als erstes Byte in der temp-Variable abgelegt und ByteToReceive um eins erhöht. ByteToReceive ist wie ähnlich wie ByteToSend eine Byte-Variable. Ist das empfangene Byte ungültig, wird nichts weiter unternommen.

Bei dem zweiten empfangenen Byte wird ähnlich vorgegangen und dieses als zweites Byte in der temp_Variable gespeichert.

Beim dritten empfangenen Byte wird dieses an dritter Stelle der temp-Variable geschrieben und der nun vollständige Streckenbefehl auf Gültigkeit geprüft. Anschließend wird ByteToReceive auf eins gesetzt. Ist das dritte empfangene Byte jedoch nicht gültig, wird wie im Fehlerfall beim zweiten Byte vorgegangen.

Am Ende der Funktion workSSC() soll das Token der Betriebsmittelverwaltung mit der Schnittstelle hello() zurückgegeben werden.

Die Funktion befehlAufGueltigkeitPruefen(Boolean) bekommt als Übergabeparameter ein Boolean übergeben. Bei gelesenem Streckenbefehl wird true, bei zu sendendem false

übergeben. Bei gelesenem Streckenbefehl wird dieser nach Validierung in den Shared Memory geschrieben.

Name:	befehlAufGueltigkeitPruefen()
Vorbedingung:	keine
Parameter:	boolean: true für Lesebefehl oder false für Schreibbefehl
Rückgabe:	boolean: true wenn gültig
Nachbedingung:	Fehlerart wurde in die entsprechende Stelle des Speichert geschrieben
	Übergabe true: Streckenbefehl wurde in den shared Memory geschrieben
Fehlerfall:	kein Zugriff auf Speicher

Name:	SSCinterrupt()
Vorbedingung:	SSC interrupt
Parameter:	keine
Rückgabe:	keine
Nachbedingung:	datenLesen() oder kollisionBearbeiten() aufgerufen
Fehlerfall:	Fehler beim Lesen oder Schreiben des Speichers

Name:	datenSenden()
Vorbedingung:	keine
Parameter:	keine
Rückgabe:	keine
Nachbedingung:	keine
Fehlerfall:	Fehler beim Lesen oder Schreiben des Speichers

Name:	datenLesen()
Vorbedingung:	SSC interrupt
Parameter:	keine
Rückgabe:	keine
Nachbedingung:	keine
Fehlerfall:	Fehler beim Lesen oder Schreiben des Speichers

Name:	checkNeuerStreckenbefehl()
Vorbedingung:	keine
Parameter:	keine
Rückgabe:	boolean: true, wenn neuer Streckenbefehl
Nachbedingung:	keine
Fehlerfall:	Fehler beim Lesen des Speichers

Name:	kollisionVerarbeiten()
Vorbedingung:	SSC interrupt
Parameter:	keine
Rückgabe:	keine
Nachbedingung:	keine
Fehlerfall:	Fehler beim Lesen oder Schreiben des Speichers

Name:	byteToReceiveInkrementieren()
Vorbedingung:	keine
Parameter:	keine
Rückgabe:	keine
Nachbedingung:	keine
Fehlerfall:	Fehler beim Lesen oder Schreiben des Speichers

Name:	byteToReceiveDekrementieren()
Vorbedingung:	keine
Parameter:	keine
Rückgabe:	keine
Nachbedingung:	keine
Fehlerfall:	Fehler beim Lesen oder Schreiben des Speichers

Name:	byteToSendInkrementieren()
Vorbedingung:	keine
Parameter:	keine
Rückgabe:	keine
Nachbedingung:	keine
Fehlerfall:	Fehler beim Lesen oder Schreiben des Speichers

Name:	byteToSendDekrementieren()
Vorbedingung:	keine
Parameter:	keine
Rückgabe:	keine
Nachbedingung:	keine
Fehlerfall:	Fehler beim Lesen oder Schreiben des Speichers