	TED (Technische Entwicklungsdokumentation) Bereich: Komponenten- / Geräteentwicklung Pflichtenheft-Spezifikation V1.00	D-021208-1
--	--	------------

Pflichtenheft-Spezifikation

086350-S1NN-ADLATUS-TMS470R-MucSuc

Flashloader Update zur Programmierung des Parrot CK5050+ Slave-Prozessors

Hinweis:

Dieses Dokument beschreibt das Anforderungsprofil einer Baugruppe oder eines Gerätes in Kurzform.

Jede Änderung an dieser Spezifikation muss in der nachfolgenden Änderungsliste beschrieben und abgezeichnet werden.

Erstellt: Datum _____ Unterschrift _____	Geprüft: Datum _____ Unterschrift _____	Freigegeben: Datum _____ Unterschrift _____
---	--	--

TED-PflichtenheftSpez.dot V1.10

Datum: 02.12.08	Name: maennel	O:\Projekte\Bootblock\S1NN\086350-S1NN-ADLATUS-TMS470R-MucSuc\2SW\10KONZ\PHSPEC-KUNDE\086350-S1NN-ADLATUS-TMS470-MucSuc_PHSPEZ_V1.00.doc	Seite: 1 von 26
-----------------	---------------	--	-----------------


Änderungsliste

Ausgabe	Änderung (Seite / Kapitel)	Datum	Unterschrift
V1.00	Auslieferung V1.00 der Software zusammen mit V1.00 der PHSPEC	01.12.08	M.Männel
V0.50	<ul style="list-style-type: none"> - Änderungen in der Anzahl der Blöcke - Flashvorgang neu verstanden 	12.11.2008	M.Männel
V0.40	<ul style="list-style-type: none"> - Änderung der Blockgröße zwischen TMS470R und Parrot - KWP2000 Modul entscheidet wann Flashfunktionen aufgerufen werden 	22.10.08	M.Männel
V0.30	Nach Review von KA am 17.10.2008	21.10.08	M.Männel
V0.20	Nach Review von KA am 30.09.2008	06.10.2008	M.Männel
V0.10	Basisversion	23.09.2008	M.Männel

Tabelle 1-1: Änderungsliste

Inhalt

1	Zielsetzung	5
2	Verwendete Begriffe und Abkürzungen	5
3	Systembeschreibung	6
3.1	Systembild	6
4	Realisierungskonzept Softwaredesign	7
4.1	Allgemeines	7
4.1.1	UDS Flashablauf	7
4.2	Kurzbeschreibung der Hardwarebasis	8
4.3	Modulplan / Schichtenmodell	9
4.3.1	Veränderungen im ADLATUS	9
4.3.2	Neuimplementierung	10
4.4	Schnittstellenbeschreibung	10
4.4.1	UDS → Parrot	10
4.4.2	Adlatus Konfiguration	12
4.4.3	Flashfunktionen und Flashparameter	14
4.5	Risiken	14
4.5.1	Technische Einschränkungen	14
4.5.2	Dauer des Flashvorgang	15
4.5.3	Flashablauf – Checksummenüberprüfung	15
4.5.4	Flashablauf – Konsistenzprüfung	16
4.5.5	Flashablauf – Kompatibilitätsprüfung	16
4.5.6	Startup des ADLATUS	16
4.5.7	Integration des PLF's im ODX	16
4.6	Headerbeschreibung / Strukturdefinitionen / Defines	17
4.6.1	ADLATUS_Global_cdf.h	17
4.6.2	ADLATUS_HISFlash.h	17
4.7	Fehlerbehandlung	19
4.7.1	KWP2000 Modul	19
4.7.2	Flash Funktionen	19
4.8	Timeouts	20
4.8.1	KWP2000	20
4.8.2	Parrot Modul	20
4.9	Modulbeschreibung FlashFunctions	20
4.9.1	PARROT_Init();	20
4.9.2	PARROT_Deinit();	21
4.9.3	PARROT_Write();	21
4.9.4	PARROT_Erase();	22
5	Anmerkungen	23
A	Anhang Bildverzeichnis	24
B	Anhang Tabellenverzeichnis	25

	TED (Technische Entwicklungsdokumentation) Bereich: Komponenten- / Geräteentwicklung Pflichtenheft-Spezifikation V1.00	D-021208-1
--	--	------------

C Anhang Literaturverzeichnis.....26

TED-PflichtenheftSpez.dot V1.00

Datum: 02.12.08	Name: maennel	O:\Projekte\Bootblock\S1NN\086350-S1NN-ADLATUS-TMS470R-MucSuc\2SW\10KONZ\PHSPEC-KUNDE\086350-S1NN-ADLATUS-TMS470-MucSuc_PHSPEZ_V1.00.doc	Seite: 4 von 26
-----------------	---------------	--	-----------------



1 Zielsetzung

Ziel des Projektes ist eine Erweiterung des ADLATUS. Durch diese Erweiterung wird das Programmieren eines mit dem TMS470R verbundenen Parrot CK5050+ Prozessors ermöglicht.

Das Ziel dieses Dokumentes ist es auf Änderungen und Erweiterung des ADLATUS einzugehen. Eine Beschreibung des ADLATUS und/oder des UDS-Flashvorgangs ist nicht Zweck dieses Dokumentes.

2 Verwendete Begriffe und Abkürzungen

Begriff	Bedeutung
0x	Kennzeichnung für einen hexadezimalen Wert
ADLATUS	Produktname der SMART Embedded-Lösung zur Reprogrammierung von Mikrocontroller
CAN	Controlled Area Network
CRC32ADL	Prüfsummenberechnungsalgorithmus des ADLATUS mit 4 Byte Ergebnis
EEPROM	Electrically Erasable Programmable Read-Only Memory
GIO	General-Purpose Input/Output
HIS	Hersteller Initiative Software
KWP2000	Key-Word-Protokoll 2000 (ISO 14230)
Nd	not defined / nicht definiert bzw. keine Anforderung
NRC	Negative Response Code (Fehlercode der negativen Antwort)
PLF	Parrot Link Format
SCI	Serial Communication Interface
SPI	Serial Peripheral Interface
SW	Software
Tbd	to be defined / noch zu definieren
TI	Texas Instruments
TMS470R	Prozessorbezeichnung
VW	Volkswagen
UART	universal asynchronous receiver-transmitter
UDS	Unified Diagnostic Services(ISO 14229)
UHV-NAR	S1nn interne Projektbezeichnung

Tabelle 2-1: Abkürzungen

3 Systembeschreibung

Die Firma S1NN entwickelt das Steuergerät für die Volkswagen AG. Die interne Projektbezeichnung bei S1nn ist UHV-NAR. Als Hauptprozessor wird der TMS470R1VF45AA eingesetzt. Auf dem Hauptprozessor ist bereits ein ADLATUS integriert

Der zweite Prozessor des Systems ist ein Bluetooth Modul der Firma Parrot. Die genaue Bezeichnung ist Parrot CK5050+ (CK5050P). Bisher aktualisiert man das Parrot Modul mit einer PC-Software über die serielle Schnittstelle des PC's. S1nn entwickelt für das Parrot Modul keine eigene Software. Die Download-dateien sind Firmware-Updates des Hersteller Parrot im PLF Format. Der Updatevorgang des Parrot Moduls ist beschrieben in [5].

Auf dem Parrot Modul wird kein ADLATUS implementiert. Der Hauptprozessor ist mit dem Zweitprozessor über 4 Leitungen verbunden.

In diesem Softwareprojekt soll der Zweitprozessor vom ADLATUS des TMS470R1VF45AA geflasht werden. Der ADLATUS des Hauptprozessors (TMS470R1VF45AA) dient als Flashgateway.

3.1 Systembild

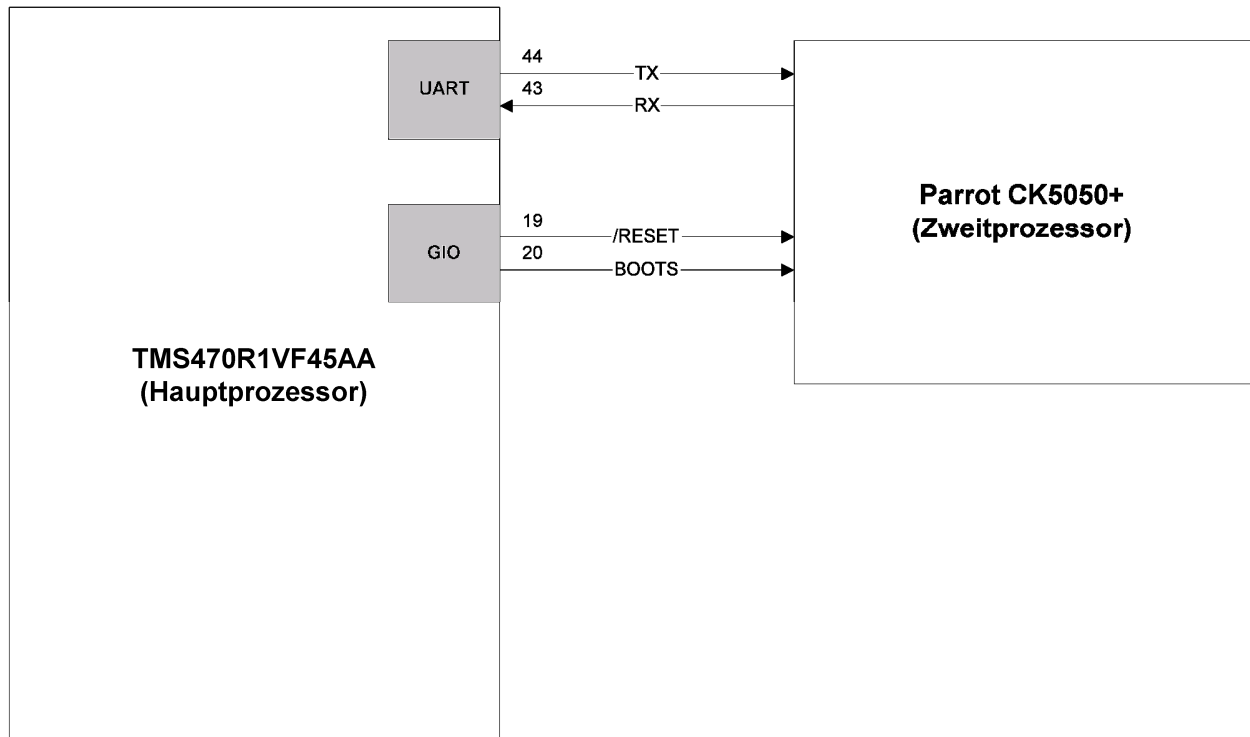


Abbildung 3-1: Gesamtschaltbild der Baugruppe

Die Ziffern bezeichnen die Nummer des Prozessorpins.

- SCI2TX (Pin 44); SCI2RX (Pin 43)
- GIOD[1] (Pin 20); GIOD[2] (Pin 19)

4 Realisierungskonzept Softwaredesign

4.1 Allgemeines

Als Grundlage für dieses Projekt dient der bereits bei S1nn integrierte TMS470R_Bflso-Source Bootloader. S1nn erhält die ADLATUS Software von VW und passt diese auf Projekt und Steuergerät an.

Die Erweiterung des ADLATUS ist so zu realisieren, dass dessen Softwarearchitektur nicht verändert wird. Eine funktionale Trennung innerhalb der Erweiterung ist dafür sinnvoll. Die Kommunikation zwischen TMS470R und Parrot wird in den Flashfunktionen gekapselt. Der ADLATUS kommuniziert über CAN mit dem Tester, speichert die Downloaddaten zwischen und ruft die Flashfunktionen auf.

Wie in [5] beschrieben, unterstützt das Parrot Modul 2 Updatevarianten, das Initial-Update und das Standard Update. Der Unterschied zwischen beiden Varianten ist, dass beim Initial-Update der gesamte Speicher des Parrot Moduls gelöscht wird.

Ein Parrot Update, unabhängig ob Initial oder Standard, ist immer in 3 Schritte aufgeteilt. Download des UART Bootloader, Download des Updaters (bei Initial Update Flasher) und Download der Daten. Der UART Bootloader wird vom Parrot Modul in den internen RAM geladen. Er ermöglicht den Empfang und das Ausführen von Programmcode im Parrot über die UART Schnittstelle. Der vom Bootloader empfangene und ausgeführte Programmcode ist der Updater, bzw. Flasher, der im 2. Schritt heruntergeladen wird. Der Updater überprüft die im Parrot vorhandenen Daten mit den empfangenen und programmiert nur neue Daten ins Parrot Modul. Im Gegensatz dazu löscht der Flasher das gesamte Parrot Modul und programmiert die empfangenen Daten.

Die zu programmierenden Daten werden im letzten Schritt an den Updater oder Flasher gesendet.

Die Entscheidung ob Initial oder Standard Update ist abhängig davon ob der Updater oder der Flasher im 2. Schritt ins Parrot Modul geladen wird.

Beim Download des Bootloaders in das Parrot Modul ist die Angabe der Gesamtlänge des Downloads kürzer als bei der Applikation und der Download wird mit einer anderen Sequenz von Befehlen abgeschlossen.

Nach dem Download des Bootloaders kann der Parrot Updatevorgang nur durch einen Reset verlassen werden. Dieser Reset wird am Ende der Programmierung durch den Tester mit einem HardReset-Request angefordert. Der Service für den Request muss so implementiert sein, dass auch das Parrot Modul gereset wird. In der aktuellen HW Basis muss hierfür keine gesonderter Code implementiert werden. Der Reset des Parrot geschieht durch das Rücksetzen der GIO Register des TMS470R auf Resetwerte bei einem SW Reset.

Weitere Informationen zum Updatevorgang des Parrot Moduls sind in [4] und [5] nachlesbar.


Der Flashvorgang für die Parrot Applikation soll den Anforderungen des Lastenheftes [6] entsprechen und mit dem VAS-Tester ausführbar sein. Die Downloaddaten für den Parrot (UART Bootloader, Updater und Daten) sind als logische Blöcke des Steuergerätes definiert. Für die Programmierung der logischen Blöcke sind spezielle Flashfunktionen nötig, die vorher geladen werden müssen.

4.1.1 UDS Flashablauf

Die 3 Schritte der Programmierung des Parrots sind in der Flash-ODX-Datei abgebildet. Die Reihenfolge der Definition der Blöcke im ODX-File bestimmt den Ablauf der Programmierung.

Folgender Ablauf muss eingehalten werden:

1. Download der Flashfunktionen
2. Download des Parrot UART Bootloaders
3. Download des Parrot Updaters
4. Download der Parrot Daten

	TED (Technische Entwicklungsdokumentation) Bereich: Komponenten- / Geräteentwicklung Pflichtenheft-Spezifikation V1.00	D-021208-1
--	--	------------

4.2 Kurzbeschreibung der Hardwarebasis

Hardwarebasis ist das Steuergerät von S1nn, interne Bezeichnung UHV-NAR. Auf dem Steuergerät sind folgende Hardwarebausteine für den ADLATUS von Bedeutung. Externer Watchdog (TI TP3828-33), externes SPI-EEPROM (ST M95160-W) und Parrot CK5050+. Der externe Watchdog ist via Jumper für die Entwicklung abschaltbar. Der interne Watchdog des Prozessors wird nicht verwendet und bleibt inaktiv.

Das Parrot Modul ist ein Bluetooth Prozessor. Zwischen TMS470R und Parrot wird eine 4-Draht Verbindung zur Steuerung und Kommunikation verwendet. Der TMS470R und das Parrot Modul kommunizieren über die UART Schnittstelle des TMS470R. Das Parrot Modul wird mit einer Reset- und einer Bootstrapleitung gesteuert. Mit diesen beiden Leitungen kann das Parrot Modul den Betriebszustand wechseln und neugestartet werden.

Der TMS470R1VF45AA arbeitet mit einer internen Taktfrequenz von 24MHz. Mit dieser Taktfrequenz kann im Onchip-Modul SCI2 eine Baudrate von 115200 Baud oder 230400 Baud eingestellt werden. Beide Baudraten werden vom Parrot Modul unterstützt. Die UART- Kommunikation ist durch Übertragung von 8 Datenbits, ein Start- und ein Stopbit sowie keine Parität gekennzeichnet.

Um die Daten der Downloaddatei zwischen zu speichern, ist ein Buffer von mindestens 512 Bytes erforderlich. Dieser Zwischenbuffer entspricht der Blockgröße während der Übertragung zwischen TMS470R und Parrot Modul und ist neben dem ADLATUS RAM-Verbrauch zu realisieren. Der TMS470R1VF45AA verfügt über 32Kbytes internes RAM, das für die Implementierung ausreichend ist.

Datum: 02.12.08	Name: maennel	O:\Projekte\Bootblock\S1NN\086350-S1NN-ADLATUS-TMS470R-MucSuc\2SW\10KONZ\PHSPEC-KUNDE\086350-S1NN-ADLATUS-TMS470-MucSuc_PHSPEZ_V1.00.doc	Seite: 8 von 26
-----------------	---------------	--	-----------------

4.3 Modulplan / Schichtenmodell

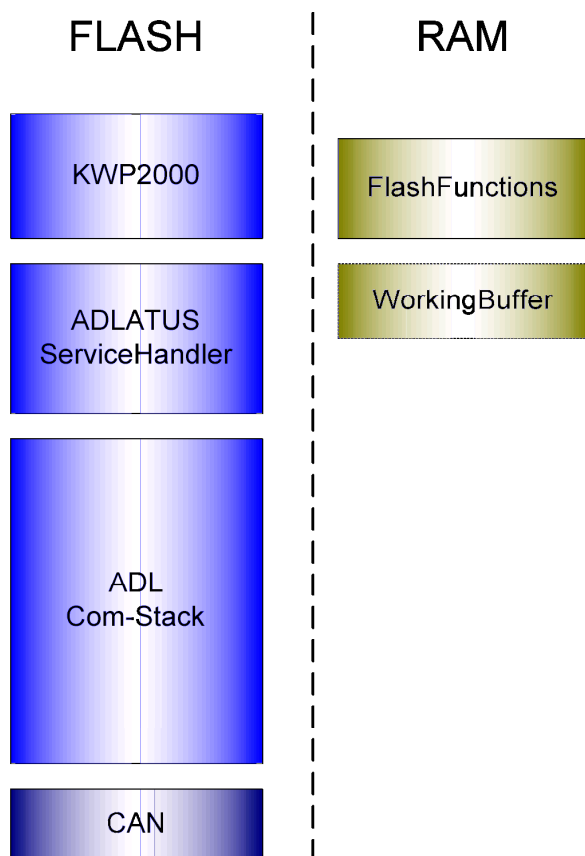


Abbildung 4-1: Modulplan

Die Architektur des ADLATUS ist in 3 Teile unterteilt. Unterschieden wird in kunden-, hardware- und unabhängig Module. Unabhängige Module werden in diesem Projekt nicht verändert.

Die Hardwareabhängigkeit ist bereits durch eine vorherige Integration des ADLATUS auf dem S1nn Steuergerät implementiert. Für die Erweiterung muss zusätzliche die Pinkonfiguration des ADLATUS verändert werden.

Die Erweiterung des ADLATUS zur Programmierung des Slaveprozessors wird durch Änderungen im kundenabhängigen Teil des ADLATUS, dem KWP2000 Modul, implementiert.

Dies entspricht der Vorgehensweise bei der Implementierung eines auf den Kunden zugeschnittenen Flashvorgangs.

Die Downloaddaten werden auf Testerseite im ODX Format vorliegen. Der binäre Datenstrom besteht aus einer kompletten PLF Datei. Innerhalb des ADLATUS müssen die Daten gesammelt und unverändert an das Parrot Modul weitergeleitet werden.

Die Flashfunktionen beinhalten alle Module um auf den Timer und die UART Schnittstelle des TMS470R zu greifen zu können.


4.3.1 Veränderungen im ADLATUS

Der ADLATUS muss so geändert werden, das weitere logische Blöcke mit neuen Speichertypen existieren. Der Flashvorgang für diese Blöcke wird ermöglicht, indem die KWP2000 Services für die Handhabung der neuen Speichertypen verändert (erweitert) werden.

Die Daten für die logischen Blöcke werden mit TransferData-Requests über CAN vom Tester an den ADLATUS gesendet und vom ADLATUS zwischengespeichert. In jedem TransferData-Request wird geprüft, ob ein komplette Block von 256 Bytes empfangen wurde. Sind nicht genügend empfangene Daten vorhanden, beendet sich die TransferData-Funktion mit positiven Ergebniss. Sobald ein vollständiger Block empfangen ist, werden die Flashfunktionen aufgerufen. Das Ergebnis der Flashfunktionen bestimmt das Ergebnis des TransferData-Requests.

Über die empfangenen Daten berechnet der ADLATUS im TransferData Service eine CRC32ADL Checksumme.

Eine weitere Änderung die im ADLATUS vorgenommen werden muss, ist das Vergrößern des Arbeitspuffers (WorkingBuffer). Dieser Buffer muss zwei volle Blöcke von 256 Byte Daten aufnehmen können. Die Größe von 512 Bytes resultiert daraus das im schlechtesten Fall 255 Daten-Bytes empfangen wurden

	TED (Technische Entwicklungsdokumentation) Bereich: Komponenten- / Geräteentwicklung Pflichtenheft-Spezifikation V1.00	D-021208-1
--	--	------------

und das letzte Byte im nächsten TransferData-Request mit der Blockgröße von 250 Bytes empfangen wird. Damit keine Daten verloren gehen, muss der Buffer die restlichen 249 Bytes aufnehmen.

Während der Übertragung zwischen ADLATUS und Parrot Bootloader, muss die Kommunikation auf CAN Seite mit Hilfe von ResponsePending Botschaften pausiert werden.

Es müssen die Dateien

- ADLATUS_ConfigProject.c
- ADLATUS_Peripherie.c
- ADLATUS_KWP2000Audi.c

geändert werden.

4.3.2 Neuimplementierung

Die Kommunikation mit dem Parrot Modul ist als eigenständiges Projekt außerhalb des ADLATUS implementiert. Hier wird das Prinzip der nachladbaren Flashfunktionen angewendet. Ein neuer Ordner, 11_parrot, enthält alle für diese Erweiterung notwendigen Dateien.

Neue Dateien sind:

- Projektdateien des CodeComposers
- ADLATUS_ParMain.c
- ADLATUS_ParFlash.c

Die Antwort des Parrot Moduls wird von den Flashfunktionen eingelesen und an den ADLATUS übergeben. Innerhalb der Flashfunktionen gibt es keine Fehlerbehebungsmechanismen, wie z.B. Blockwiederholungen. Ein Fehler in den Flashfunktionen führt zum sofortigen Flashabbruch.

4.4 Schnittstellenbeschreibung

4.4.1 UDS → Parrot

Die Programmierung des Parrots soll sich im Flashablauf wie ein Update eines logischen Blockes gestalten. Downloaddaten die mit Hilfe von UDS-Befehlen vom Tester zum ADLATUS des TMS470R übertragen werden, müssen an das Parrot Modul weitergegeben werden. Wobei die Übertragung vom ADLATUS zum Parrot Modul einem Update des Parrots entspricht. Dies geschieht im Zusammenspiel von KWP2000 Modul (ADLATUS_KWP2000Audi.c) und Flashfunktionen.

Folgend soll dargestellt werden, welche Funktionalitäten in den KWP2000 Funktionen im Bezug auf den Parrot ausgeführt werden müssen. Jede dieser hier aufgeführten KWP2000 Funktionen stellt einen über CAN aufrufbaren UDS-Service dar.

RoutineControl - EraseMemory:

- Wenn Parrot Daten gesendet werden, EEPROM Zelle auf ungültig setzen und Löschzähler inkrementieren

RequestDownload:

- Überprüfung des zuvor heruntergeladenen HIS-Container
- keine Längenüberprüfung, lediglich Abspeichern der Gesamtlänge des Downloads

Datum: 02.12.08	Name: maennel	O:\Projekte\Bootblock\S1NN\086350-S1NN-ADLATUS-TMS470R-MucSuc\2SW\10KONZ\PHSPEC-KUNDE\086350-S1NN-ADLATUS-TMS470-MucSuc_PHSPEZ_V1.00.doc	Seite: 10 von 26
-----------------	---------------	--	------------------

- Parrot-Init-Funktion für Verbindungsaufbau aufrufen (bei Bootloader Download Parrot in Boot-modus setzen)

TransferData:

- Daten sammeln
- Aufruf der Parrot-Write-Funktion
- Während Übertragung TMS470R → Parrot, ResponsePendings über CAN versenden
- Gesamtdatenlänge inkrementieren und abspeichern
- CRC32ADL über empfangene Daten berechnen
- Antworten des Parrot Moduls auswerten, ggf. Flashabbruch anzeigen
- Ergebnis der Programmierung in KWP2000 Kontrollstruktur speichern

TransferExit:

- Länge der empfangen Daten mit den erwarteten Länge vergleichen
- Prüfsummenberechnung abschließen

RoutineControl - CheckMemory:

- Überprüfung der berechneten Prüfsumme gegen vom Tester gesendete Prüfsumme
- Wenn Daten, Ergebnis der Programmierung im EEPROM Zelle speichern
- Ergebnis der Programmierung in RoutineResult kommunizieren
- Wenn Daten und erfolgreiche Programmierung, Programmierzähler im EEPROM inkrementieren

RoutineControl – CheckDependencies:

- Auswertung des Konsistenzbytes für Parrot Daten Block im EEPROM für jeden logischen Block (inkl. Parrot)
- Ausführen einer S1nn Kompatibilitätsprüfungsfunktion

Das Verhalten kann nicht mit den bisherigen Speichertypen verglichen werden.

Service	vergleichbarer Speichertyp	Unterschied
EraseMemory	RAM	
RequestDownload	FLASH	- Keine Längenüberprüfung gegen ADLATUS Konfiguration
TransferData	FLASH	- Vergleiche mit Entschlüsselung und Signaturberechnung über die übertragenen Daten (HIS-SecurityClass CCC) - Pagegröße 8192 - Übergabe der Bufferlänge an die Flashfunktionen
TransferExit	FLASH	-
CheckMemory	FLASH	- Verifikation der Signature (HIS-SecurityClass CCC)
CheckDependencies	FLASH	

Tabelle 4-1: Vergleich Speichermedien

Aus der Tabelle wird ersichtlich das eine Konfiguration weder als FLASH noch als RAM Speicher sinnvoll ist. Das unterschiedliche Verhalten wird über einen neuen Speichertyp (PARROT) definiert.

4.4.2 Adlatus Konfiguration

Im ADLATUS müssen 3 weitere logische Blöcke angelegt werden. Abhängig von diesen Blöcken müssen die unterschiedlichen Verhalten in den KWP2000 Services (siehe 4.5.2) implementiert werden.

Auszug aus ADLATUS_ConfigProject.c:

Parrot Daten Block:

```

/*-----*/
/* APPLICATION AREA */
/*-----*/
/* 02 - ASW Area 02 (PARROT) */
/*-----*/
{
    /* definition for His-header */
    FLASH_DRIVER_VERSION_INTERFACE_PARROT,
    0x00,
    FLASH_DRIVER_VERSION_MASKTYPE_PARROT,
    FLASH_DRIVER_VERSION_MCTYPE_PARROT,
    /* -----++-----*/
    /* 4 Byte - Startaddress */ d_CCadinfo_ParrotStartAdr,
    /* -----++-----*/
    /* 4 Byte - EndAddress */ d_CCadinfo_ParrotEndAdr,
    /* -----++-----*/
    /* 2 Byte - Memory Info */ (
    /* Entry Type */ d_CCglo_NormalEntry | \
    /* Memory Type */ d_CCglo_Parrot | \
    /* Erase state */ d_CCglo_NotErasable | \
    /* Programming state */ d_CCglo_Programable | \
    /* Check state */ d_CCglo_CheckRange ),
    /* -----++-----*/
    /* 2 Byte - Memory Index */ (WORD) 0x0004,
    /* -----++-----*/
    /* 1 Byte - Name */ (BYTE) d_CCconprj_ApplicationEntry2,
    /* -----++-----*/
    /* 2 Byte - Reserved */ (WORD) d_CCglo_NotUsed
    /* -----++-----*/
}, /* -01- -----*/

```

Die logischen Blöcke des Parrot Moduls ist durch folgende Merkmale definiert:

- nicht löschar
- programmierbar
- Speichertyp Parrot mit eigenen HIS-Container
- Adressen und Länge müssen einem bestimmten Bereich entsprechen

Die Start- und Endadresse des Blockes wird so gewählt, das der Bereich der Länge des maximal möglichen Downloads in den Parrot entspricht. In der folgenden Abbildung aus dem Dokument „CK5xxx - Up-

date specification“ sind die Speicherbereiche des Parrot dargestellt. Der größte Bereich beginnt bei der Adresse 0x00020000 und endet bei 0x07FFFFFF. Diese Start- und Endadressen addiert mit 0x80000000 werden für die Definition des logischen Blockes verwendet.

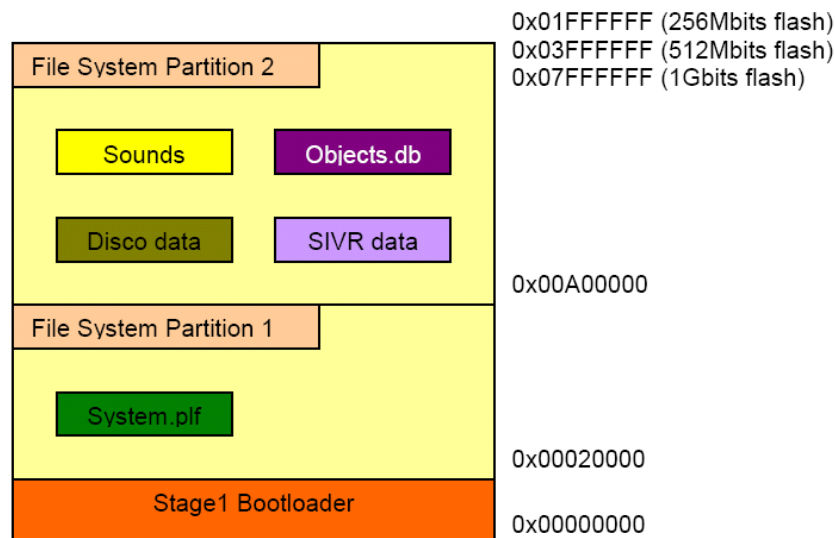


Abbildung 4-2: CK5050P, nand flash

Zusätzlich zum Parrot Daten Block gibt es 2 weitere Blöcke. Den Bootloader und den Updater Block. Diese Blöcke müssen mit unterschiedlichen Memorytypes definiert werden, um unterschiedliches Verhalten hervorrufen zu können.

Die Eigenschaften des Parrot Bootloader Blockes sind:

- bei Initialisierung der Flashfunktionen Parrot in Bootmodus setzen
- Speedcode Einstellung
- Längenangabe in den Flashfunktionen 2 Byte lang
- Ende des erfolgreichen Downloads muss dem Parrot signalisiert werden
- Keine EEPROM Verwaltungsdaten
- Addressbereich 0x0000 bis 0x1FFFF

Eigenschaften des Parrot Updater Blockes:

- Speedcode Einstellung
- Längenangabe in 4 Bytes
- Ende des erfolgreichen Downloads wird durch Parrot signalisiert
- Keine EEPROM Verwaltungsdaten
- Addressbereich 0x20000 bis Parrot Speicherende

Eigenschaften des Parrot Daten Blockes:

- Speedcode Einstellung
- Längenangabe in 4 Bytes
- Ende des erfolgreichen Downloads wird durch Parrot signalisiert
- EEPROM Verwaltungsdaten
- Addressbereich 0x20000 bis Parrot Speicherende

Die Blöcke werden auf Testerseite durch ihren Blockindex unterschieden.

4.4.3 Flashfunktionen und Flashparameter

Die Schnittstelle zwischen ADLATUS KWP2000 Modul und den Flashfunktionen sind die Flashparameter. In dieser Struktur werden Parameter aus den jeweiligen KWP2000 Funktionen an die Flashfunktionen übergeben.

Der Zugriff auf die Flashfunktionen geschieht über den in [8] definierten HIS Header. In diesem Header sind Informationen zur Hardware- Kompatibilität und Pointer auf die Modulfunktionen gespeichert.

Die Flashparameter und Konstanten für die Hardware- Kompatibilität sind in der Datei ADLATUS_HISFlash.h definiert. Da die Flashparameter einmalig für den ADLATUS kompiliert werden, ist es notwendig das alle Parameter die für die normale Flashprogrammierung, wie auch für die Parrot-Flashfunktionen gebraucht werden, in diesen Flashparametern enthalten sind. Da die normale Flashprogrammierung mehr Parameter wie die Parrot Programmierung verwendet, werden keine neuen Variablen in den Flashparametern definiert, sondern bestehende anders verwendet. Die Bedeutung der Variablen in den HIS-Container ist also unterschiedlich.

Die Flashfunktionen bestehen aus Initialisierungs-, Deinitialisierung-, Schreibe- und Löschroutinen.

4.5 Risiken

Die Risiken beschreiben technische Probleme die in Zusammenhang mit Parrot und UDS-Flashvorgang entstehen. Zu den Risiken werden in diesem Kapitel Lösungen vorgeschlagen.

4.5.1 Technische Einschränkungen

Das **Verhalten des Parrot Moduls** im Bootmodus ist nicht klar definiert. Es fehlen Informationen über das zeitliche und funktionale Verhalten während der Datenübertragung. Ohne diese Informationen muss davon ausgegangen werden, dass es diese Punkte nicht relevant sind. Die Implementierung wird so gewählt, das Punkte nachträglich in den Flashfunktionen realisiert werden können.

Das Parrot Modul bietet **keine Rücklesefunktionalität** an, d.h. es können keine Daten aus dem Parrot gelesen werden. Dies ist ein gravierender Unterschied zu allen bisher verwendeten Speichermedien. Dieser Diskrepanz hat Auswirkungen auf den Flashvorgang und die Veränderungen im KWP200 Modul.

4.5.1.1 Kritische Punkte

Nr.	Problem	Lösungsvorschlag	Status
1	Konsistenzcheck durch den Bootloader während des Start-up nicht möglich (Wartezeit zu hoch)	S1NN: Konsistenz des CK5050+ wird von Applikation geprüft. Bei Inkonsistenz wird entsprechende EEPROM Zelle von der Applikation manipuliert und ein Reset ausgelöst	gelöst
2	Befehlssatz für Flashen des CK5050+ unklar	Da im LH nicht genauer beschrieben, werden im Bootmodus keine bestimmten Befehle unterstützt. Bootmodus erlaubt lediglich Speed-CodeBefehle (0xA3,0xA5,0xA7,0xA9) Bestätigung (0xAA, 0xA3), Angabe der Downloadgröße und Datenübertragung	offen
3	Zeitverhalten während des Flashvorgangs unklar	100ms Timeout nach der Aktivierung des Bootmode. Wird dieser Timeout deaktiviert nach den ersten UART Nachrichten? Aufrechterhaltung der internen Kommunikation?	offen
4	funktionelles Verhalten während des Flashvorgangs unklar	Wann werden die Übertragungen bestätigt? Eine Bestätigung nur, wenn die angegebene Gesamtlänge erreicht ist, oder alle 8192 Bytes?	offen

Datum: 02.12.08	Name: maennel	O:\Projekte\Bootblock\S1NN\086350-S1NN-ADLATUS-TMS470R-MucSuc\2SW\10KONZ\PHSPEC-KUNDE\086350-S1NN-ADLATUS-TMS470-MucSuc_PHSPEZ_V1.00.doc	Seite: 14 von 26
-----------------	---------------	--	------------------

5	Blockgrößen zw. TMS470R und CK5050+ unklar	Laut Lastenheft sind maximal 8192. Sind auch kleinerer Blöcke möglich?	offen
6	Initial Update (Bootloader + Applikation) soll nicht unterstützt werden	S1NN muss einen UART Zugang für externe Wartung im Steuergerät bereitstellen. Sollte das Initial Update vom ADLATUS unterstützt werden, ist dies über eine 2. Flashfunktion zu realisieren, um den geänderten Ablauf ADLATUS unabhängig zu implementieren.	offen
7	Checksumme der Applikation	Berechnung, Länge unklar. Für das Bootlader Update (Initial Update) im LH beschrieben. Für Applikationsdownload fehlt die Beschreibung!	offen
8	PLF Format (Informationen zum Type und Bereich des zu programmierenden Speichers)	Das proprietärs Dateiformat ist unbekannt. Welche Informationen müssen enthalten sein? Abbildung des PLF als Flashfile im ODX notwendig. Muss das interne Format bekannt sein?	offen
9	Welche Schlechtfälle müssen für den Test der Programmierung des CK5050 durchgeführt werden	Z.B. Fehler in PLF Bedatung	offen

4.5.2 Dauer des Flashvorgang

Nutzdaten: Bsp.: 1,5 MByte

Übertragungsgeschwindigkeit: 100kBaud

ISO-TP: Blockgröße 0xFA (250Bytes)

Datalink: 11-Bit Identifier

Die Nutzdaten für den Parrot werden mit TransferData Requests vom Tester zum Hauptprozessor übertragen. In einem Request können maximal 248 Daten-Bytes übertragen werden. Ein Request ist in 40 CAN-Frames unterteilt. 36 CAN-Frames vom Tester zum SG und 4 CAN-Frames als Antworten des SG's. bei einer Baudrate von 100kBaud benötigt ein Frame 1,1 ms für die Übertragung. Sollen 1,5 MBytes im Parrot geflasht werden, bedeutet das, dass die Daten in 253720 Frames übertragen werden. Die Sendezeit auf CAN beträgt 280s. Zur Sendezeit addiert sich die Verarbeitungszeit und die Wartezeiten innerhalb der CAN Kommunikation zwischen Tester und Bootloader, sowie während der Übertragung vom TMS470R zum Parrot.

Die Flashzeit des Parrot CK5050 für die oben definierte Nutzdatenmenge wird auf 10 Minuten geschätzt. Dieser Schätzwert hängt stark von der Größe der Nutzdaten, der Fehlerbehandlung und der Buslast ab. Daraus lassen sich keine Performance Anforderungen für SMART ableiten.

In einem vergleichbaren Projekt (1,5MBytes Flash, 500kBaud) betrug die Flashdauer 6 Minuten.

4.5.3 Flashablauf – Checksummenüberprüfung


Die Anforderungen VW80126-343, VW80126-115, VW80126-117 aus [2] können nicht erfüllt werden. Um die Anforderungen zu erfüllen, muss es möglich sein, die geflashten Daten aus dem Parrot Modul zu lesen und über diese eine Checksumme zu rechnen.

Die vom Tester im CheckMemory-Request übertragene Prüfsumme darf nicht ignoriert werden. Eine pauschal versendetet positive Antwort auf den Request ist nicht erlaubt. Robustheitstests manipulieren eine korrekte Prüfsumme und erwarten das richtige Verhalten vom Bootloader.

Das Ergebnis des Flashvorgangs wird von der Funktion TransferData in der KWP2000 Kontroll-Struktur abgespeichert. Dieses Ergebnis und die über die empfangenen Daten berechnete Prüfsumme werden für die Checksummenüberprüfung verwendet. Nicht bestanden wird die Prüfung, wenn der Flashvorgang nicht erfolgreich war (Fehler in den Flashfunktionen aufgetreten) oder die Prüfsummen nicht übereinstimmen.

Bis zum Aufruf von CheckMemory bleibt der Parrot Block im EEPROM als inkonsistent markiert. Nach bestandener Checksummenüberprüfung wird das Bytes für Parrot Block im EEPROM als gültig markiert.

Datum: 02.12.08	Name: maennel	O:\Projekte\Bootblock\S1NN\086350-S1NN-ADLATUS-TMS470R-MucSuc\2SW\10KONZ\PHSPEC-KUNDE\086350-S1NN-ADLATUS-TMS470-MucSuc_PHSPEZ_V1.00.doc	Seite: 15 von 26
-----------------	---------------	--	------------------

	TED (Technische Entwicklungsdokumentation) Bereich: Komponenten- / Geräteentwicklung Pflichtenheft-Spezifikation V1.00	D-021208-1
--	--	------------

Eine Berechnung der Checksummen über die empfangenen Bytes ist nicht konform zum Lastenheft VW80126. Denn laut Anforderung darf die Berechnung erst beginnen nachdem alle Daten empfangen wurden. Ein vollständiger Empfang ist in diesem Projekt nicht möglich, weil der RAM-Speicher des TMS470R's nicht groß genug ist. Die Verbindung zwischen Flashergebnis und Integrationsprüfung der empfangenen Daten bietet genügend Sicherheit um einen Flashvorgang als erfolgreich bezeichnen zu können. Deshalb kann die Anforderung als erfüllt bezeichnet werden.

4.5.4 Flashablauf – Konsistenzprüfung

Die Prüfung der Konsistenz des Gesamtsystems erfolgt wie bisher in Audi Projekten. Alle Konsistenzbytes der einzelnen Blöcke werden zur Konsistenzprüfung des Gesamtsystems verwendet. Zeigt eines oder mehrer Bytes einen ungültigen Block an, gilt das Gesamtsystem als ungültig und die Applikation wird nicht gestartet.

4.5.5 Flashablauf – Kompatibilitätsprüfung

Da der ADLATUS keine Informationen zu den Versionen der Applikation und der Parrot Software hat, ist ein besonderer Mechanismus notwendig.

Wie im Kapitel 5.4 von [2] wird beschrieben wie der Programmiervorgang zu beenden ist. Dabei führt der Bootloader keinen Kompatibilitätscheck durch. Die Überprüfung muss in der Applikation durchgeführt werden. Der Bootloader wird mit dem Zustand System programmiert verlassen. Diesen Zustand muss die Applikation nach erfolgreichem Kompatibilitätscheck auf konsistent setzen. Der Zustand ist im EEPROM abgelegt. Weiter Information findet man im Readme des ADLATUS.

4.5.6 Startup des ADLATUS

Zusätzlich zur Ermittlung der Systemkonsistenz mit Hilfe der Konsistenzbytes im EEPROM, werden logische Blöcke auf deren Programmierzustand geprüft.

Die Speicherbereiche im Flash und im EEPROM werden vom ADLATUS in der Hochstartphase mit einer Schnellprüfung auf Füllung geprüft. Dies geschieht indem der ADLATUS an der Prüfsummenadresse des jeweiligen logischen Blockes liest und die gelesenen Werte auf Ungleichheit mit 0xFF oder 0x00 prüft. Ungleiche Werte bedeuten einen programmierten, nicht gelöschten, logischen Block.

In diesem Projekt kann alternativ auf Nachrichten eines programmierten Parrot Moduls gewartet werden. Da hierfür aber die Informationen über Inhalt und zeitliches Verhalten der Nachrichten fehlen, kann auf diese Alternative nicht zurückgegriffen werden.

Der Startupcheck für den Parrot Daten Block beschränkt sich nur auf die Auswertung der Verwaltungsdaten im EEPROM für diesen Block. Die Prüfung auf programmierten Speicher im Parrot und das Warten auf eine Nachricht des Parrot-Moduls wird nicht implementiert.

4.5.7 Integration des PLF's im ODX

Die Downloaddateien des Parrots liegen im PLF Format vor. Aus der Analyse von Log-Dateien des Updatevorganges und der Projektdefinition durch S1nn, die das Update des Parrots als Datenstromprogrammierung bezeichnet, wird der Inhalt der Downloaddatei als binär angenommen. Diese Binärdaten werden in eine Hexdatei umgewandelt und im ODX Container integriert.

Datum: 02.12.08	Name: maennel	O:\Projekte\Bootblock\S1NN\086350-S1NN-ADLATUS-TMS470R-MucSuc\2SW\10KONZ\PHSPEC-KUNDE\086350-S1NN-ADLATUS-TMS470-MucSuc_PHSPEZ_V1.00.doc	Seite: 16 von 26
-----------------	---------------	--	------------------

4.6 Headerbeschreibung / Strukturdefinitionen / Defines

4.6.1 ADLATUS_Global_cdf.h

Zur Definition des neuen Speicherbereiches:

```
#define d_CCglo_Parrot          (UWORD)  0x0800u    /* 0000 1000 0000 0000 */
#define d_CCglo_Parrot_Boot    (UWORD)  0x1800u    /* 0001 1000 0000 0000 */
#define d_CCglo_Parrot_Upd     (UWORD)  0x2800u    /* 0010 1000 0000 0000 */
#define d_CCglo_MemoryTypeMask (UWORD)  0x3800u    /* 0011 1000 0000 0000 */
```

4.6.2 ADLATUS_HISFlash.h

Defines in Verbindung mit der Identifizierung des Flash Treibers (HIS Flash Funktionen):

```
#define FLASH_DRIVER_VERSION_MCU_TYPE_PARROT (UBYTE) 0x01u
#define FLASH_DRIVER_VERSION_MASK_TYPE_PARROT (UBYTE) 0x81u
#define FLASH_DRIVER_VERSION_INTERFACE_PARROT (UBYTE) 0x02u
```

Struktur der Flashparameter: (nicht verwendete Variablen sind grau)

```
typedef struct
{
    /* initialisation : in-out Parameters */
    tBugfixVersion patchlevel;          /* flash driver patch level version */
    tMinorNumber minornumber;           /* Flash driver minor version number */
    tMajorNumber majornumber;           /* Flash driver major version number */
    unsigned char reserved1;             /* reserved for future use, set to 0x00 */

    /* return value */
    tFlashResult errorcode;
    unsigned short reserved2;

    /* erase / write . input parameters */
    tAddress address;                   /* logical target address */
    tLength length;                    /* length information in bytes */
    tData *data;                       /* pointer to data buffer */

    /* additional input parameters */
    tWDTriggerFct wdTriggerFct;        /* pointer to watchdog trigger routine */

    /* erase/write : optional output parameters : debugging information */
    tData intendedData[2];              /* intended data at error address */
    tData actualData[2];                /* actual data at error address */
    tAddress errorAddress;              /* address of error */
}
```



```
tCommand Command_UB;          /* command byte */
tDirection Direction_UB;       /* direction byte */
tFormat Format_UB;              /* format byte */
tMilTime Millisec_UB;          /* alignment byte */

tSector StartSector_US;        /* command byte */
tSector EndSector_US;          /* command byte */
tSector NbrOfSectors_US;       /* command byte */
tSector ActualSector_US;       /* command byte */

t_Delaytime Delaytime_UL;      /* cpu frequency in Hz */
} tFlashParam;
```


Interne Defines in den Flash Funktionen:

```
/* Error code HIS - KWP20000 */
#define kFlashOK                0x0000u
#define kVerificationError      0x0001u
#define kNotEnoughData          0x0002u
#define kFlashFailed            0x0004u
#define kDeviceOutOfRange       0x0008u
#define kMemoryOverflow         0x0010u
#define kTooMuchData            0x0020u
#define kAccessError            0x0040u
#define kProtectionViolation    0x0080u

/* Speedcode defines */
#define d_Parrot_SpeedCode_115200 0xA3u
#define d_Parrot_SpeedCode_230400 0xA5u
#define d_Parrot_SpeedCode_460800 0xA7u
#define d_Parrot_SpeedCode_921600 0xA9u
#define d_Parrot_SpeedCode_Confirm 0xA8u
#define d_Parrot_SpeedCode_Decline 0xAAu

/* error Code Parrot - HIS */
#define d_Parrot_ProgramSuccessful 0x55u /*download and programming is done and OK */
#define d_Parrot_E_BadChecksum      0xB0u /* bad checksum */
#define d_Parrot_E_NotEnoughBytes   0xB1u /* not enough bytes received */
#define d_Parrot_E_Flash            0xB2u /* error in flash */
#define d_Parrot_E_Device           0xB3u /* not targeted device */
#define d_Parrot_E_OutOfMemory      0xB4u /* not enough memory */
#define d_Parrot_E_BytesOverflow    0xB5u /* too many bytes received */
#define d_Parrot_E_PLF              0xB6u /* error in plf */
#define d_Parrot_E_Address          0xB7u /* forbidden flash address */
```

TED-PflichtenheftSpez.dot V1.00

	TED (Technische Entwicklungsdokumentation) Bereich: Komponenten- / Geräteentwicklung Pflichtenheft-Spezifikation V1.00	D-021208-1
--	--	------------

4.7 Fehlerbehandlung

In der Fehlerbehandlung sind nur Fehler, die im Zusammenhang mit dem Parrot Modul auftreten, oder die Abweichungen von der UDS Fehlerbehandlung, beschrieben. D.h. es gilt weiterhin die Fehlerbehandlung des UDS Flashvorgangs wie im Lastenheft [2] beschrieben.

4.7.1 KWP2000 Modul

4.7.1.1 Löschen (RoutineControl – EraseMemory)

Im Bezug auf Parrot keine besondere Fehlerbehandlung.

4.7.1.2 Anforderung der Datenübertragung (RequestDownload)

Im Bezug auf Parrot keine besondere Fehlerbehandlung.

Die variierende Länge der Nutzdaten muss innerhalb des definierten Adressbereiches liegen.

4.7.1.3 Datenübertragung (TransferData)

Wird ein Fehler aus den Flash Funktionen gemeldet, führt das zum sofortigen Flashabbruch. Dem Tester wird dieser Flashabbruch mit NRC 0x72 (GeneralProgrammingFailure) kommuniziert.

4.7.1.4 Abschließen der Datenübertragung (TransferExit)

Im Bezug auf Parrot keine besondere Fehlerbehandlung.

4.7.1.5 Integrationsprüfung (RoutineControl – CheckMemory)

Die geflashten Daten werden nicht überprüft. Intern wird eine Checksumme über die übertragenen Daten berechnet. Diese „On-The-Fly“ berechnete Prüfsumme wird mit der vom Tester gesendeten Prüfsumme verglichen.

4.7.1.6 Konsistentprüfung (RoutineControl – CheckDependencies)

Im Bezug auf Parrot keine besondere Fehlerbehandlung.

4.7.2 Flash Funktionen

In den Flashfunktionen ist die Fehlerbehandlung besonders während der Entwicklung wichtig. Während des UDS-Flashvorgang werden Flashabbrüche dem Tester oft nur über einen „Negativ Response Code“ (NRC) mitgeteilt. Der genauer Fehlergrund muss mit dem Debugger ermittelt werden.

4.7.2.1 Initialisierungsfunktion

In dieser Funktion dient die Fehlerbehandlung der Evaluierung des Kommunikationsaufbaus. Sollte es hier Problem geben, ist der Rückgabewert `kAccessError`.

4.7.2.2 Deinitialisierungsfunktion

Im Bezug auf Parrot keine besondere Fehlerbehandlung.

Datum: 02.12.08	Name: maennel	O:\Projekte\Bootblock\S1NN\086350-S1NN-ADLATUS-TMS470R-MucSuc\2SW\10KONZ\PHSPEC-KUNDE\086350-S1NN-ADLATUS-TMS470-MucSuc_PHSPEZ_V1.00.doc	Seite: 19 von 26
-----------------	---------------	--	------------------

4.7.2.3 Löschfunktion

Im Bezug auf Parrot keine besondere Fehlerbehandlung.

4.7.2.4 Schreibfunktion

Es findet eine Auswertung der Antworten des Parrots während der Datenübertragung vom TMS470R zum Parrot statt. Mögliche Fehler werden in Rückgabewerte der Flashfunktionen übersetzt.

Antwort des Parrots	Rückgabewert Flashfunktionen	Bedeutung
0xB0	kVerificationError	bad checksum
0xB1	kNotEnoughData	not enough bytes received
0xB2	kFlashFailed	error in flash
0xB3	kDeviceOutOfRange	not targeted device
0xB4	kMemoryOverflow	not enough memory
0xB5	kToMuchData	too many bytes received
0xB6	kAccessError	error in plf
0xB7	kProtectionViolation	forbidden flash address

Tabelle 4-2: Fehlercodes Flashfunktionen Schreibfunktion

4.8 Timeouts

4.8.1 KWP2000

Es gelten die durch die Lastenheft [2] und [3] definierten Timeouts.

4.8.2 Parrot Modul

Ein von Parrot definierter Timeout gilt zu Beginn des Updatesvorganges. Nachdem das Parrot Modul mit den Steuersignalen in den Bootmodus gesetzt ist, muss innerhalb von 100ms mit der UART Kommunikation begonnen werden.

Ein weitere Timeout ist definiert für das Empfangen von Antworten des Parrot. Nach jedem möglichem Empfang von Daten muss bis zur Wiederholung der Empfangsfunktion 10ms gewartet werden. Ein maximales Timeout bis die Antworten des Parrot Moduls eintreffen ist nicht vorgegeben. Hierfür werden Erfahrungswerte der Entwicklung herangezogen.

Das Wiederholungs- und Timeoutverhalten zwischen TMS470R und Parrot Modul ist in den Flashfunktionen implementiert und kann nachträglich geändert werden.

4.9 Modulbeschreibung FlashFunctions

Das Modul FlashFunctions ist nach der Spezifikation [8] implementiert.

4.9.1 PARROT_Init();

Aufgabe: Initialisierung der Flashparameter. Parrot in Bootmodus setzen.

Prototyp: `void PARROT_Init(tFlashParam *);`

Datum: 02.12.08	Name: maennel	O:\Projekte\Bootblock\S1NN\086350-S1NN-ADLATUS-TMS470R-MucSuc\2SW\10KONZ\PHSPEC-KUNDE\086350-S1NN-ADLATUS-TMS470-MucSuc_PHSPEZ_V1.00.doc	Seite: 20 von 26
-----------------	---------------	--	------------------



Syntax: **PARROT_Init (flashParam) ;**
Parameter: Flashparameter
Rückgabewert: Flashparameter
Funktionstyp: nicht-speicheranfordernd
Beschreibung: Die Funktion setzt die Flashparameter auf ihren Startwert. Zusätzlich wird das UART Modul des TMS470R initialisiert und die Kommunikation mit dem Parrot begonnen.

Beispiel:

```
// Initialisierung des Parrot Treibers
void main( void )
{
    tFlashParam fp;
    PARROT_Init( &fp);
    ...
}
```

4.9.2 PARROT_Deinit();

Aufgabe: Deinitialisierung der Flashparameter.
Prototyp: **void PARROT_Deinit (tFlashParam *);**
Syntax: **PARROT_Deinit (flashParam) ;**
Parameter: Flashparameter
Rückgabewert: Flashparameter
Funktionstyp: nicht-speicheranfordernd
Beschreibung: Die Funktion setzt alle Flashparameter auf Null.

Beispiel:


```
// Deinitialisierung des Parrot Treibers
void main( void )
{
    tFlashParam fp;
    PARROT_Deinit( &fp);
    ...
}
```

4.9.3 PARROT_Write();

Aufgabe: Datenübertragung vom TMS470R zum Parrot.
Prototyp: **void PARROT_Write (tFlashParam *);**
Syntax: **PARROT_Write (flashParam) ;**
Parameter: Flashparameter
Rückgabewert: Flashparameter
Funktionstyp: nicht-speicheranfordernd
Beschreibung: Die Funktion überträgt mit Hilfe des SCI2 Onchip- Modul Daten vom TMS470R zum Parrot und liest Antworten des Parrots..

Beispiel:


```
// Übertrage Daten
void main( void )
{
    flashParam -> data = &g_CCdp_WorkingBuffer_BUF[0];
    flashParam -> length = t_CCkwpaudi_KwpCtrl_ST.ProgLength_UL;
    PARROT_Write(flashParam);
    ...
}
```

	TED (Technische Entwicklungsdokumentation) Bereich: Komponenten- / Geräteentwicklung Pflichtenheft-Spezifikation V1.00	D-021208-1
--	--	------------

4.9.4 PARROT_Erase();

Aufgabe: Leere Funktion.
Prototyp: **void PARROT_Erase(tFlashParam *) ;**
Syntax: **PARROT_Erase(flashParam) ;**
Parameter: keine
Rückgabewert: keine
Funktionstyp: nicht-speicheranfordernd
Beschreibung: Diese Funktion ist eine Dummy Funktion. Es wird ein `return` ausgeführt.
Beispiel:


Datum: 02.12.08	Name: maennel	O:\Projekte\Bootblock\S1NN\086350-S1NN-ADLATUS-TMS470R-MucSuc\2SW\10KONZ\PHSPEC-KUNDE\086350-S1NN-ADLATUS-TMS470-MucSuc_PHSPEZ_V1.00.doc	Seite: 22 von 26
-----------------	---------------	--	------------------

	TED (Technische Entwicklungsdokumentation) Bereich: Komponenten- / Geräteentwicklung Pflichtenheft-Spezifikation V1.00	D-021208-1
--	--	------------

5 Anmerkungen

TED-PflichtenheftSpez.dot V1.00

Datum: 02.12.08	Name: maennel	O:\Projekte\Bootblock\S1NN\086350-S1NN-ADLATUS-TMS470R-MucSuc\2SW\10KONZ\PHSPEC-KUNDE\086350-S1NN-ADLATUS-TMS470-MucSuc_PHSPEZ_V1.00.doc	Seite: 23 von 26
-----------------	---------------	--	------------------

	TED (Technische Entwicklungsdokumentation) Bereich: Komponenten- / Geräteentwicklung Pflichtenheft-Spezifikation V1.00	D-021208-1
--	--	------------

A Anhang Bildverzeichnis

Abbildung 3-1: Gesamtschaltbild der Baugruppe	6
Abbildung 4-2: CK5050P, nand flash	13

Datum: 02.12.08	Name: maennel	O:\Projekte\Bootblock\S1NN\086350-S1NN-ADLATUS-TMS470R-MucSuc\2SW\10KONZ\PHSPEC-KUNDE\086350-S1NN-ADLATUS-TMS470-MucSuc_PHSPEZ_V1.00.doc	Seite: 24 von 26
-----------------	---------------	--	------------------

B Anhang Tabellenverzeichnis

Tabelle 1-1: Änderungsliste.....	2
Tabelle 2-1: Abkürzungen	5
Tabelle 4-1: Vergleich Speichermedien	11
Tabelle 4-2: Fehlercodes Flashfunktionen Schreibfunktion	20

C Anhang Literaturverzeichnis

lfd. Nr.	Bezeichnung	Version	Datum	Ersteller
[1]	ISO/DIS 14229-1.2 Road vehicles — Unified diagnostic services (UDS) — Part 1: Specification and requirements Thema: Spezifikationen und Anforderungen für Diagnose Dienste auf Applikationsebene	2.1	26.05.2004	ISO
[2]	VW80126 UDS konforme Programmierung von Steuergeräten	1.1	30.03.2006	AUDI/VW
[3]	VW80124 Unified Diagnostic Services Protocol UDS Application-Layer & Implementation	1.6	12.02.2007	AUDI/VW
[4]	Lastenheft Flashloader Update zweiter Prozessor	02	15.07.2008	S1nn
[5]	CK5xxx - Update specification	1.0	11.06.2008	Parrot
[6]	Partielle Programmierung von UDS-Steuergeräten	1.0	26.06.2007	Volkswagen AG
[7]	CodingGuidelines_v23_export_aus_Wiki_2008_08_12.pdf	2.3	12.08.2008	S1nn
[8]	Functional Specification of a Flash Driver	1.3	06.06.2002	HIS

Erstelldatum: 02.12.2008 Zuletzt gespeichert: 02.12.2008 Zuletzt gedruckt: 02.12.2008