1 Kommunikationsprotokoll

Das **Inter Board Protocoll (IBP)** dient als Vereinbarung von Kommunikationsregeln Ã¼ber eine serielle Kommunikationsschnittstelle. Innerhalb dieses Projektes soll es die Kommunikation Ã¼ber eine serielle USB Schnittstelle erleichtern.

1.1 Liste von Anforderungen und Konzepten

Im folgenden werden alle Anforderungen im Rahmen dieses Teilprojektes aufgelistet.

Manche Anforderungen wurden erst spÃ¤ter im Entwicklungsprozess entdeckt oder erdacht, weshalb zu jeder Anforderung auch eine Versionsnummer des ersten Auftretens angegeben ist. Zusammen mit der VersionsÃ¼bersicht kann das eine Zeitliche EinschÃ¤tzung des Auftretens der Anforderung ermÃ¶glichen.

Dadurch das die tatsÃ¤chlichen Anforderungen an das Protokoll zu Beginn der Entwicklung noch nicht genau festgelegt werden konnten, wurden im mÃ¶gliche Anforderungen und Features gesammelt.

A01 Frage-Antwort-Schema
: Auf Grund der Art einer seriellen Ã¤bertragung ist es vorteilhaft, wenn das Protokoll ein Frage-Antwort Konzept mit einem dominanten Kommunikationspartner abbildet. Das heiÃŸt es existieren ein Master und ein Slave. Der Slave wird Ã¼ber Befehle vom Master zum Handeln aufgefordert und kann dann ein Antwort im Rahmen von vorher getroffener Vereinbarungen senden.

A02 ID → Anzahl verschiedener Befehle
: Eine Menge von 256 verschiedenen Befehlen ist mÃ¶glich. Ein Befehl kann durch eine Identifikationsnummer(ID) erkannt werden. Die Zahl 256 wird auf Grund von maschineller reprÃ¤sentierbarkeit Festgelegt.(8bit)

A03 Priorisierung
: Eine PrioritÃ¤t zur Ã¤bertragung mehrerer Befehle favorisiert Befehle mit kleinerer ID. So kÃ¶nnen Ã¼ber das Protokoll weitere FunktionalitÃ¤ten, wie EchtzeitfÃ¤higkeit, durch geschickte Wahl der ID ermÃ¶glicht werden.

A04 MaximalgrÃ¶ÃŸe
: Eine MaximalgrÃ¶ÃŸe eines Befehls wird auf 255 Bytes auf Grund von maschineller reprÃ¤sentierbarkeit Festgelegt.(8bit) Diese GrÃ¶ÃŸenvereinbarung wÃ¤re hinsichtlich einer geplanten Speicherung oder Ã¤bertragung der GrÃ¶ÃŸe interessant.

A05 Vereinbarung der Ãœbertragungslängen der Payloads
Die Kommunikationspartner mÃ¼ssen Informationen Ã¼ber die LÃnge der Verschiedenen Ãœbertragungen besitzen, um empfangene Daten ihrem Zweck zuweisen zu kÃ¶nnen und den Start der nÃchsten Sendung zu ermitteln.

A05.1 statisch Eine statische GrÃ¶ÃŸenvereinbarung ist zu Beginn der Laufzeit bei allen Kommunikationspartnern bekannt.

A05.2 dynamisch Eine dynamische GrÃ¶ÃŸenvereinbarung wird bei laufender Kommunikation jedesmal neu vereinbart. Die dynamische Art einer Nachricht muss jedoch statisch bekannt sein, sodass die Kommunikationspartner auf einen dynamischen Austausch einstellen kÃ¶nnen.

Hinsichtlich des Frage-Antwort-Schemas sollte also fÃ¼r jede Frage und Antwort jeweils eine GrÃ¶ÃŸe bekannt gemacht werden (Requestsize und Respnse-/Answersize). Alternativ zu einer GrÃ¶ÃŸe soll angegeben werden kÃ¶nnen, das die Art einer Nachricht dynamisch ist. Die GrÃ¶ÃŸe muss dann wÃhrend der Kommunikation ausgehandelt werden.

A06 Fehlererkennung

- Checksummen/Hashes helfen Fehlerhafte Ãœbertragung durch redundante Zusatzinformation zu erkennen.

A07 Fehlerbehandlung

- Fehlerbekanntmachung
  Fehler werden, wenn nÃ¶tig, dem Kommunikationspartner propagiert. Um den Ã¼brigen Kommunikationsablauf dabei nicht bis wenig zu belasten, kann hierfÃ¼r ein Statusfeld verwendet und mitgeschickt werden.
- Negative Antwort Fehler auf Slave-Seite fÃ¼hren zu Fehlerhaften Antwortdaten. Desshalb hat der Slave die mÃ¶glichkeit eine Negative Antwort (negative response)zu senden, die keine Antwortdaten mehr mitfÃ¼hrt. Die Negative Antwort bietet jedoch wiederum MÃ¶glichkeiten verschiedene Fehler anzugeben.
- Von Masterseite kÃ¶nnen Fehler pragmatischer durch wiederholtes senden behandelt werden. Durch den Status kann die Kommunikationsschnittstelle hier hauptsÃchlich nebenbei bestimmte

konfigurierungen der Schnittstelle auf den Slaves zur Laufzeit anregen. Szenarien wie ein kontrollierter Verbindungsabbruch sind denkbar.

A08 plattformspezifische Problembehandlung Auf Grund der Unterschiedlichen Beschaffenheit der mÃ¶glichen Zielplattformen kÃ¶nnen Implementierungen variieren. Die MÃ¶glichkeiten der Plattformen sollten dabei jeweils optimal ausgenutzt werden.

A08.1 auf Raspberry Pi 3

A08.2 auf Arduino Uno

A09 Benutzerfreundlichkeit Der Prozess der Benutzung des Protokolls soll effizient gekapselt werden. FÃ¼r den tatsÃ¤chlichen Protokollablauf wird so eine Schicht erstellt, in dem stÃ¶rungsfrei gearbeitet werden kann, ohne das sich fÃ¼r die Benutzung benÃ¶tigte Schnittstellen zu oft Ändern wÃ¼rden.

A09.1 auf Raspberry Pi 3

A09.2 auf Arduino Uno

1.2 Realisierung

Benutzte Programmiersprachen waren C++. C code wurde wenn vorhanden auf C++ geportet, in dem Funktionen und Daten in Klassen gekapselt wurden. FÃ¼r zusÃ¤tzliche Tools wurde Python verwendet. (funktionieren mit Python2 und Python3)

Das Protokoll wurde zum Teil iterativ Entwickelt. Das soll heiÃŸen es wurde wiederholt ein funktionsfÃ¤higes Produkt mit Features erstellt und integriert. Beim Auftreten neuer oder verÄnderter Anforderungen oder von Fehlern wurde die Komponente Kommunikationsprotokoll aber auch teils komplett neu implementiert. Das bedeutet aber auch, dass bestimmte Features, die in einer frÃ¼heren Version vorhanden waren, in eine spÃ¤teren Version jedoch vernachlÃ¤ssigt oder nicht mehr reimplementiert wurden. Im folgenden werden daher mehrere Versionen, ihre Probleme und die Reaktion auf diese Probleme aufgefÃ¼hrt, erklÃ¤rt und begrÃ¼ndet.

Entwickelt wurden Komponenten auf beiden Benutzten Plattformen, Raspberry Pi 3 und Arduino Uno.

VersionsÃ¼bersicht :

Im folgenden ist eine Ãœbersicht Ã¼ber die durchlaufenen Versionen wÃ¤hrend der Entwicklung zu Orientierung bereitgestellt. Dabei sind auch zeitlich Beginn und Ende des Entwicklunszeitraumes angegeben.

VNr.	Wesentliche Ã„nderungen	Begin	Ende
0.0	Erste Versuche		
0.1	Kontrollstrukturen und Kapselung des Prozesses		

1.2.1 Version

$V0.0$

Anforderungen erfÃ¼llt :

$A01$

$A02$

Diese Version is als erster minimalistischer Versuch gedacht. Sie wurde erstellt um erste Anforderungen zu testen und um eventuell Ã¼bersehene Basisanforderungen zu finden. Die Arbeiten an dieser Version waren hauptsÃ¤chlich auf dem Raspberry Pi angesiedelt. Die FunktionalitÃ¤t des Slaves wurde dabei zunÃ¤chst simuliert. Daduch, dass die Benutzung der Seriellen Schnittstelle in einer Komponente bereits funktioniert hat, konnte ohne tatsÃ¤chliche Integrationstests ausgekommen werden. Des weiteren wurde zunÃ¤chst ein simpler Protokollablauf verwendet, wie folgt:

In Version 0 alle benÃ¶tigten Aufrufe wurden zunÃ¤chst testmÃ¤ÃŸig hart codiert. Ein Unit-Test bestÃ¤tigte die ordnungsgemÃ¤ÃŸe FunktionalitÃ¤t. Keine weiteren Anforderungen kamen zum Vorschein.

1.2.2 Version

$V0.1$

Anforderungen erfÃ¼llt :

In diese Version sollten mehr Ã¤berlegungen einflieÃŸen, kontollierende Strukturen sollen erstellt werden und der Prozess der Protokollbenutzung gekapselt. Eine automatisiertere, benutzbare Softwarekomponente wird hier zunÃ¤chst fÃ¼r die Masterplattform Raspberry Pi angestrebt. Die Implementierung auf der Slaveplattform Arduino Uno wurde zunÃ¤chst vernachlÃ¤ssigt.

GemÃ¤ÃŸ [A08.1] wurden zunÃ¤chst die Eigenschaften des Raspberry Pi eingeschÃ¤tzt. Der Raspberry Pi nimmt dabei die Rolle des Masters bei der Kommunikation ein. Folgende Eigenschaften waren dabei besonders interessant :

- Raspberry Pi 3 wurde mit Betriebssystem ausgestattet (Rasbian) → Raspberry ermÃ¶glicht Multitasking und -threading
- FÃ¼r einen Controller besitzt der Raspberry vergebend viel Arbeitsspeicher (1GB) =¿ Das anlegen eines Buffers fÃ¼r Nachrichten ist nicht kritisch.
- Das Raspbian Betriebssystem hat ein Filesystem → Konfigurationsdateien direkt auf der Plattform sind mÃ¶glich.

Daher Modellidee :

- Ein dedizierter Thread ist fÃ¼r das Senden und Empfangen, bzw. den Ablauf des Protokolls, zustÃndig.
- Benutzer geben ihre Befehle an den Slave als Pakete dem Thread. Die zugehÃ¶rige Softwarekomponente wird ¿Packet¡ genannt.
- Eine Softwarekomponente kapselt die FunktionalitÃt des Threads, diese wird ¿Transceiver¡ genannt.
- Ein Benutzer kann mittels einer weiteren Softwarekomponente, ¿Inbox¡, auf Antworten warten.
- Informationen Ã¼ber die LÃnge der Payload eines Request oder einer Response kann Ã¼ber eine Configurationsdatei bewerkstelligt werden. Das einlesen der Datei und die Bereitstellung der Information wird durch die Komponente ¿Rule¡ verwaltet.

Dadurch kÃ¶nnen Anforderungen bedient werden :

A01 Die Frage wird den Thread Ã¼bergeben, die Antwort landet in der ¿Inbox¡.

A02 ¿Packet¡en wird eine Nachrichtenart, ID, zugeschrieben.

A03 Der Thread kann selbst entscheiden in welcher Reihenfolge angekommene ¿Packet¡e verschickt werden. Damit ist Priorisierung umsetzbar.

A04 ¿Packet¡e fassen ein Maximum an Ãœbertragungsdaten ein.

A05.1 Eine Ã¼ber ¿Rule¡ eingelesene Konfigurationsdatei enthÃlt Informationen Ã¼ber die GrÃ¶ÃŸe der Payload fÃ¼r eine Nachrichtenart (ID).

A08 Raspberry Eigenschaften wurden ausgenutzt.

und erzeugt weitere Annehmlichkeiten :

- Benutzer kÃ¶nnen in ihren eigenen Threads arbeiten, d.h. verschiedenen Komponenten kÃ¶nnen potentiell Zugang

– TatsÃchliche Kommunikation lÃuft zentral (nicht jeder Benutzer baut eine eigene LÃ¶sung) → Kommunikation leichter zu verwalten.

Es entstehen dadurch auch neue Anforderungen:

F09 : Die Benutzung der Prokollsoftware muss auf der Zielplattform Pi von allen laufenden Threads der Software aus potentiell machbar sein.

F09.1 : Die Protokollsoftware auf der Zielplattform Pi ist threadsafe implementiert. (Vermeidung von Race-Conditions)

Implementierung :

Die Implementierung sieht fÃ¼r jede der besprochenen Komponenten eine Klasse vor. Die Klassen wurden zum grÃ¶ßten Teil in der selben Reihenfolge implementiert.

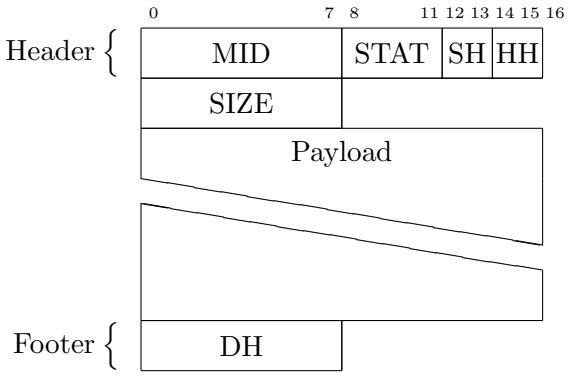
Konzeptionell

Aufbau

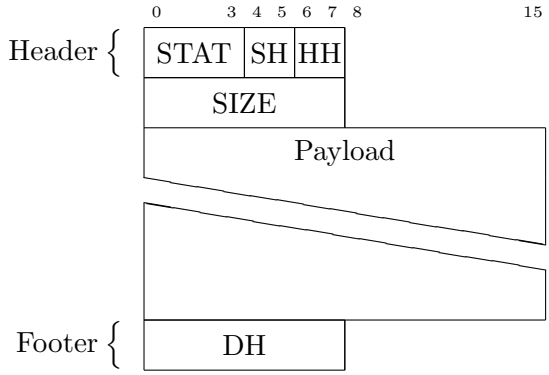
Legende:

MID	Identifikationsnummer der Anfrage
EID	Nummer des Fehlers bzgl. MID
SIZE	GrÃ¶ße einer dynamischen Payload, nicht existent bei statisch vereinbarter Ãœbert
STAT	Status der Ãœbertragung;Protokollinterne Fehlererkennung
HH	Hash des Headers
SH	Hash der dynamischen GrÃ¶ße (unwichtig bei statisch)
DH	Hash der Payload
Payload	Nutzdaten

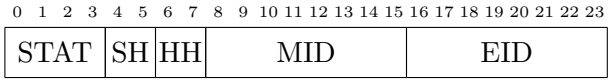
Request



Response



Negative Response



1.3 Status

Das Statusfeld wird verwendet um den Status der Kommunikationspartner zu kommunizieren. Die Ã¼bliche DatenÃ¼bertragung kann dabei nebenher unbeeintrÃchtigt weiterlaufen. Das Statusfeld ist 4bit lang =¿ 16 verschiedene Stati mÃ¶glich. Die Bedeutung des Status kann je nach Art des Kommunikationspartners (Master oder Slave) variieren, d.h. der Status 8 beduetet beispielsweise auf dem Master etwas anderes als auf dem Slave. Das Statusfeld hat im Kontrollfluss die wichtige Aufgabe, dem Master eine MÃ¶glichkeit zu geben, zwischen positiver oder negativer Antwort zu unterscheiden.

1.3.1 Master

Bit 0 wird als STOP Befehl benutzt. Sein Versand bedeutet das Ende der Kommunikation. Bit 1 wird als REINIT Befehlt benutz. Sein Versand fordert eine Reinitialisierung der Kommunikation auf Slave-Seite. Bit 2 und 3 wird verwendet um eine erneute Sendung mit einer Nummer 0-3 zu kennzeichnen. Dies geschieht wenn eine Sendung im Vorhinein fehlgeschlagen hat. Eine Sendung kann bis zu 3 Mal wiederholt werden.

bit	Zweck
0	STOP
1	REINIT
2,3	Resend counter

1.3.2 Slave

Bit 0 wird verwendet um einen internen Fehler am Slave zu propagieren, der zur Folge hat das dieser nicht korrekt oder gar nicht auf Befehle reagiert. Der Master kann dadurch auf den Zustand reagieren um zum Beispiel den Fehler zu loggen, den Sendevorgang fÃ¼r eine bestimmte Zeit einzustellen, Systeme kontrolliert herunterfahren zu lassen, etc.

Bit 1, 2 und 3 werden zum Kommunizieren von Fehlern der 3 Checksummen des Protokolls wie folgt verwendet.

bit	Zweck
0	interner Slave Fehler
1	Headerhash falsch
2	Sizehash falsch
3	Datahash falsch